Impact of Covid-19 Pandemic on the Global Economy

Summary

The dataset for this project was collected from Mendeley Data: [The Impact of Covid-19 Pandemic on the Global Economy: Emphasis on Poverty Alleviation and Economic Growth](#).

The data I investigate here consists of records on the impact of covid-19 on the global economy including 210 countries.

Main objective of the analysis is to focus on prediction. In this project, we will employ linear regression algorithms to find relationship between common GDP and human development index and total number of death.

We will then choose the best candidate algorithm from preliminary results.

The goal with this implementation is to construct a model that accurately predicts how the global economy of each country is affected.

Libraries

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import plotly.express as px
import pycountry_convert as pc
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import KFold, cross_val_predict
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet, RidgeCV, LassoCV, ElasticNetCV
from sklearn.pipeline import Pipeline

import warnings
warnings.filterwarnings('ignore', module='sklearn')
```

Exploratory Data Analysis

In [2]:
```
df = pd.read_csv("Impact of Covid-19 Pandemic on the Global Economy.csv")
```

In [3]:
```
df
```

Out[3]:

	iso_code	location	date	total_cases	total_deaths	stringency_index	population	gdp_per_capita	human_development_index	Unnamed: 9	Unnamed: 10
0	AFG	Afghanistan	2019-12-31	0.0	0.0	0.00	38928341	1803.987	0.498	NaN	NaN
1	AFG	Afghanistan	2020-01-01	0.0	0.0	0.00	38928341	1803.987	0.498	NaN	NaN
2	AFG	Afghanistan	2020-01-02	0.0	0.0	0.00	38928341	1803.987	0.498	NaN	NaN
3	AFG	Afghanistan	2020-01-03	0.0	0.0	0.00	38928341	1803.987	0.498	NaN	NaN
4	AFG	Afghanistan	2020-01-04	0.0	0.0	0.00	38928341	1803.987	0.498	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...
50413	ZWE	Zimbabwe	2020-10-15	8055.0	231.0	76.85	14862927	1899.775	0.535	8.994048	5.442418
50414	ZWE	Zimbabwe	2020-10-16	8075.0	231.0	76.85	14862927	1899.775	0.535	8.996528	5.442418
50415	ZWE	Zimbabwe	2020-10-17	8099.0	231.0	76.85	14862927	1899.775	0.535	8.999496	5.442418
50416	ZWE	Zimbabwe	2020-10-18	8110.0	231.0	76.85	14862927	1899.775	0.535	9.000853	5.442418
50417	ZWE	Zimbabwe	2020-10-19	8147.0	231.0	76.85	14862927	1899.775	0.535	9.005405	5.442418

50418 rows × 14 columns

Column Preproccess stage

In [4]:
```
df = df.rename(columns={'iso_code':'CODE',"population":"POPULATION",'location':'COUNTRY','date':'DATE','total_cases':'TOTAL CASE
S','total_deaths':'TOTAL DEATHS','human_development_index':'HDI',"gdp_per_capita":"GDPCAP"})
```

In [5]:
```
df = df.drop(['Unnamed: 9', 'Unnamed: 10', 'Unnamed: 11', 'Unnamed: 12', 'Unnamed: 13'], axis = 1)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50418 entries, 0 to 50417
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   CODE              50418 non-null  object
 1   COUNTRY           50418 non-null  object
 2   DATE              50418 non-null  object
 3   TOTAL CASES       47324 non-null  float64
 4   TOTAL DEATHS      39228 non-null  float64
 5   stringency_index  43292 non-null  float64
 6   POPULATION        50418 non-null  int64
 7   GDPCAP            44706 non-null  float64
 8   HDI               44216 non-null  float64
dtypes: float64(5), int64(1), object(3)
memory usage: 3.5+ MB
```

In [7]:
```
df.describe()
```

Out[7]:

	TOTAL CASES	TOTAL DEATHS	stringency_index	POPULATION	GDPCAP	HDI
count	4.732400e+04	39228.000000	43292.000000	5.041800e+04	44706.000000	44216.000000
mean	6.621927e+04	2978.767819	56.162022	4.251601e+07	20818.706240	0.720139
std	4.045582e+05	13836.644013	27.532685	1.564607e+08	20441.365392	0.160902
min	0.000000e+00	0.000000	0.000000	8.090000e+02	661.240000	0.000000
25%	1.260000e+02	10.000000	37.960000	1.399491e+06	5338.454000	0.601000
50%	1.594000e+03	64.000000	61.110000	8.278737e+06	13913.839000	0.752000
75%	1.584775e+04	564.000000	78.700000	2.913681e+07	31400.840000	0.847000
max	8.154595e+06	219674.000000	100.000000	1.439324e+09	116935.600000	0.953000

Feature Exploration

CODE: country code

COUNTRY: Name of the country

DATE: The date of the record

TOTAL CASES: The number of COVID19 cases

TOTAL DEATHS: The number of COVID19 deaths

stringency_index: The Stringency Index provides a computable parameter to evaluate the effectiveness of the nationwide lockdown. It is used by the Oxford COVID-19 Government Response Tracker with a database of 17 indicators of government response such as school and workplace closings, public events, public transport, stay-at-home policies. The Stringency Index is a number from 0 to 100 that reflects these indicators. A higher index score indicates a higher level of stringency.

POPULATION: The population of the Country

GDPCAP: A country's GDP or gross domestic product is calculated by taking into account the monetary worth of a nation's goods and services after a certain period of time, usually one year. It's a measure of economic activity.

HDI: The HDI was created to emphasize that people and their capabilities should be the ultimate criteria for assessing the development of a country, not economic growth alone. The Human Development Index (HDI) is a summary measure of average achievement in key dimensions of human development: a long and healthy life, being knowledgeable and have a decent standard of living. The HDI is the geometric mean of normalized indices for each of the three dimensions.

Preprocessing Stage

In [8]:
```
print('The initial total number of records: '+str(len(df.index)))
print('Number of countries: '+str(len(df['COUNTRY'].unique())))
print('Number of missing values: \n' + str(df.isnull().sum()))
```

```
The initial total number of records: 50418
Number of countries: 210
Number of missing values:
CODE                  0
COUNTRY               0
DATE                  0
TOTAL CASES        3094
TOTAL DEATHS      11190
stringency_index   7126
POPULATION            0
GDPCAP             5712
HDI                6202
dtype: int64
```

At first we have to take actions concerning the columns contain missing values: total_cases, total_deaths, stringency_index, population, gdp_per_capita, hdi.

I decided to drop the rows with missing data as we would still have enough data to train our models.

In [9]:
```
df = df.dropna(axis = 0)
print('The updated total number of records: '+str(len(df.index)))
print('The updated number of countries: '+str(len(df['COUNTRY'].unique())))
print('Number of missing values after preprocessing:')
df.isnull().sum()
```

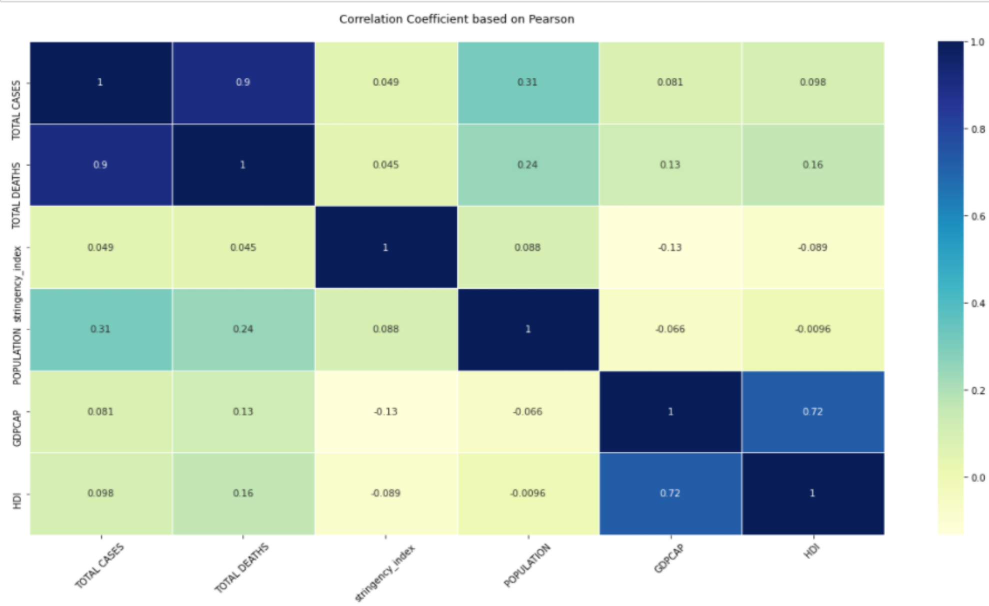
```
The updated total number of records: 31518
The updated number of countries: 154
Number of missing values after preprocessing:
```

Out[9]:
```
CODE              0
COUNTRY           0
DATE              0
TOTAL CASES       0
TOTAL DEATHS      0
stringency_index  0
POPULATION        0
GDPCAP            0
HDI               0
dtype: int64
```

Let's look now at the correlation coefficient. A coefficient close to 1 means that there's a very strong positive correlation between the two variables.

In our case we can quickly see that: The Human Development Index (HDI) is strongly correlated to the GDP per Capita (GDPCAP) and total CASES to total DEATHS. The population also has a strong correlation to the number of total cases and deaths. As it was expected, a high population will have a higher number of cases and deaths. What we are looking for is the relationship between GDP per capita(or HDI) and total number of cases or deaths.

In [10]:
```
corr = df.corr(method='pearson')
fig = plt.subplots(figsize = (20, 10))
sns.heatmap(corr,
            xticklabels=corr.columns,
            yticklabels=corr.columns,
            cmap='YlGnBu',
            annot=True,
            linewidth=0.5)
plt.xticks(rotation=45)
plt.title("Correlation Coefficient based on Pearson\n")
plt.show()
```

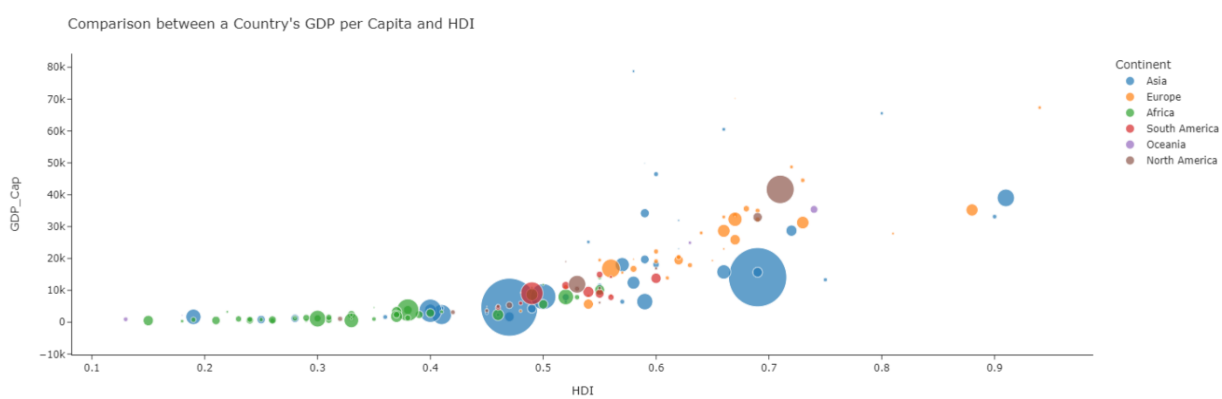


From the heatmap it seems that GDPCAP and HDI are both more affected by the number of deaths than the number of cases.

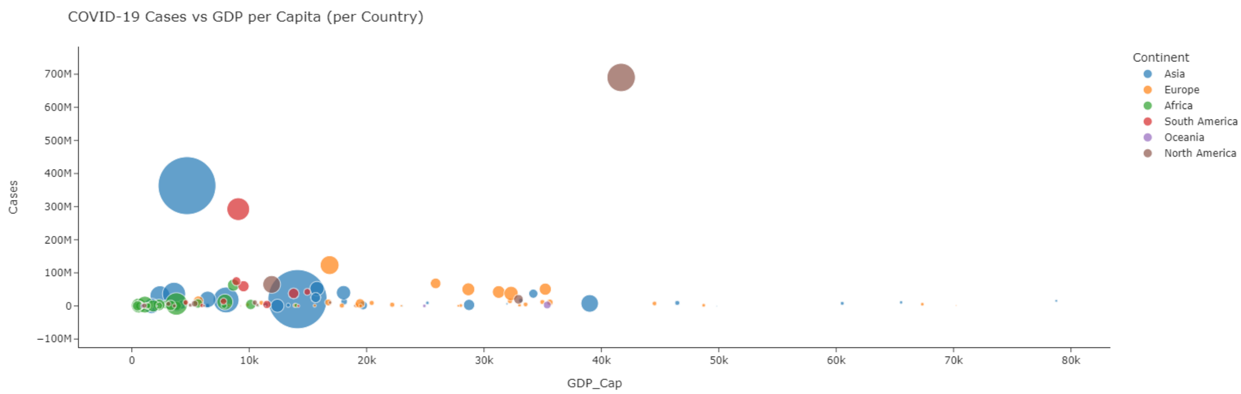
In [11]:
```
df = df[df.COUNTRY != 'Kosovo']

Country = df.COUNTRY.unique().tolist()
country_code = df.CODE.unique().tolist()
pop_world = df.POPULATION.unique().tolist()
hdi_world = []
gdp_world = []
Cases_Country = []
Death_Country = []
stringency_index = []

for i in Country:
    hdi_world.append((df.loc[df.COUNTRY == i, 'HDI']).sum()/294)
    gdp_world.append(df.loc[df.COUNTRY == i, 'GDPCAP'].sum()/294)
    stringency_index.append(df.loc[df.COUNTRY == i, 'stringency_index'].sum()/294)
    Cases_Country.append(df.loc[(df["COUNTRY"] == i), "TOTAL CASES"].sum())
    Death_Country.append(df.loc[(df["COUNTRY"] == i), "TOTAL DEATHS"].sum())
```

In [12]:
```
alpha2_code = []
for i in country_code:
    alpha2_code.append(pc.country_alpha3_to_country_alpha2(i))
continent_code = []
for i in alpha2_code:
    try:
        continent_code.append(pc.country_alpha2_to_continent_code(i))
    except:
        continent_code.append('Unknown')

data_agg = pd.DataFrame(list(zip(country_code, Country, pop_world, Cases_Country, Death_Country, hdi_world, gdp_world, stringenc
y_index, continent_code)), columns =['Code', 'Country', 'Population', 'Cases', 'Deaths', 'HDI', 'GDP_Cap','Stringency_Index', 'C
ontinent'])
data_agg = data_agg.replace({'AF':'Africa', 'AN':'Antarctica', 'AS':'Asia', 'EU':'Europe', 'NA':'North America', 'OC':'Oceania',
'SA':'South America'})
data_agg = data_agg.round(2)
```

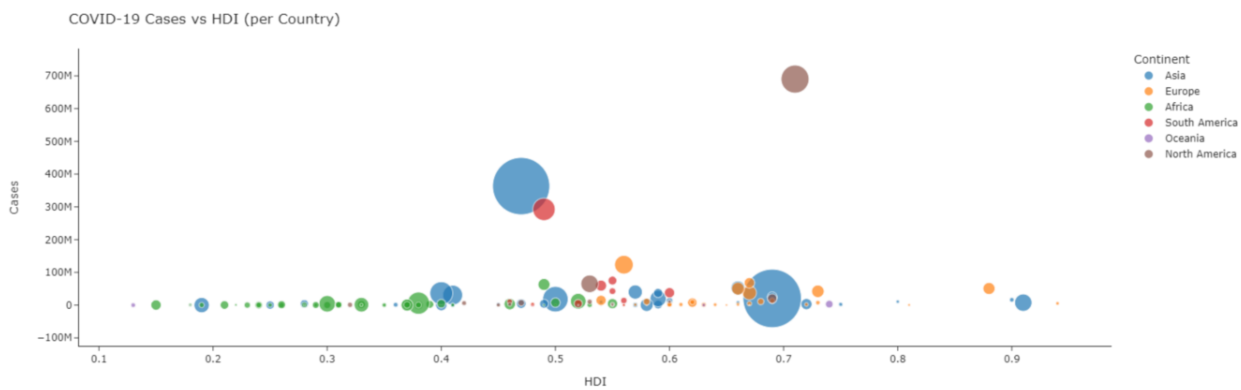
In [35]:
```
fig = px.scatter(data_agg, x="HDI", y="GDP_Cap", size="Population", hover_name="Country", color='Continent', template='simple_wh
ite', size_max=50)
fig.update_layout(
    height=500,
    title_text="Comparison between a Country's GDP per Capita and HDI"
)
fig.show()
```

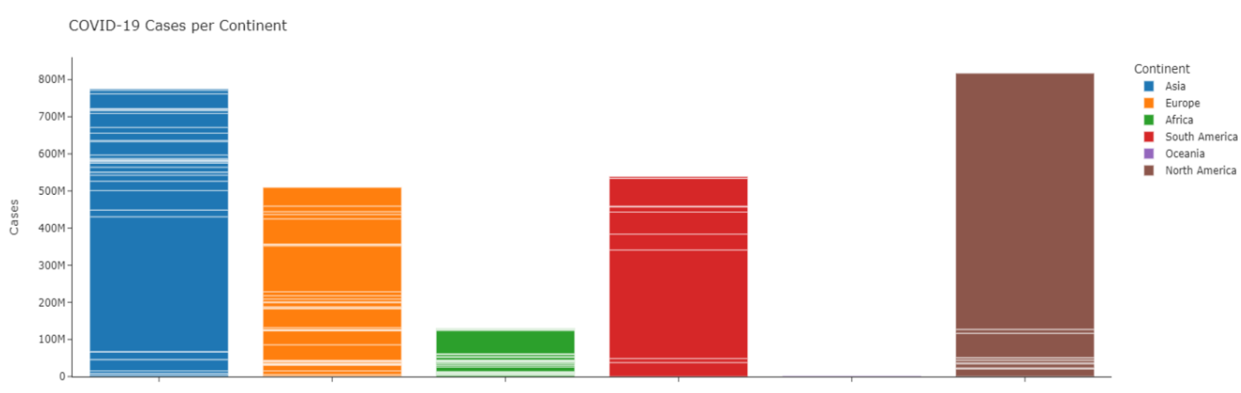


```
In [34]: fig = px.scatter(data_agg, x="GDP_Cap", y="Cases", size="Population", hover_name="Country", color='Continent', template='simple_
white', size_max=50)
fig.update_layout(
    height=500,
    title_text="COVID-19 Cases vs GDP per Capita (per Country)"
)
fig.show()
```




```
In [33]: fig = px.scatter(data_agg, x='HDI', y='Cases', hover_name='Country', color='Continent', size='Population', template="simple_whit
e", size_max=50)
fig.update_traces(textposition='top center')
fig.update_layout(
    height=500,
    title_text='COVID-19 Cases vs HDI (per Country)'
)
fig.show()
```

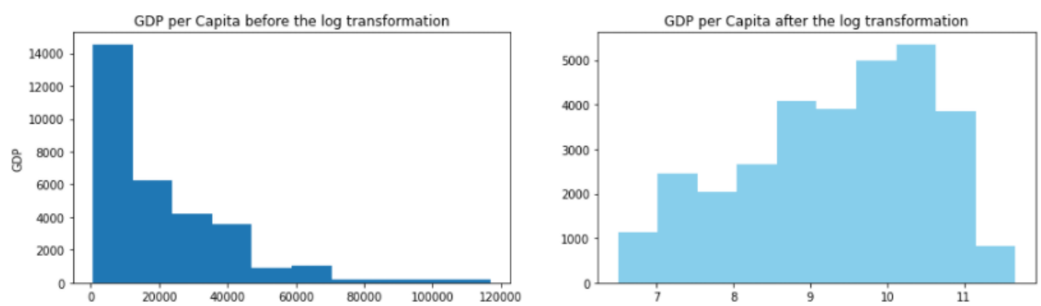


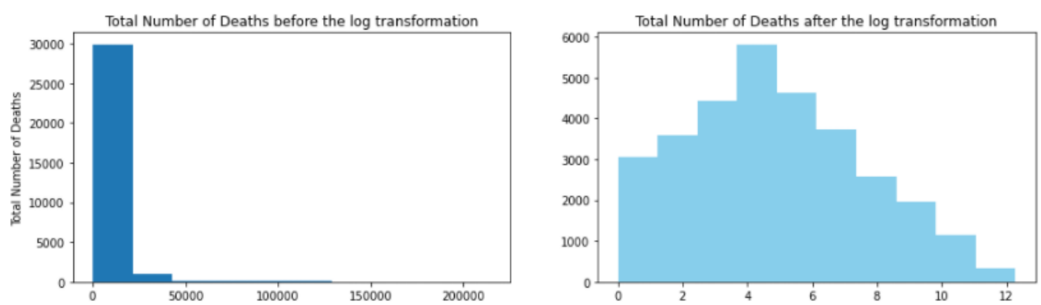


In [17]:
```
# Log-transform the skewed features
gdp_transformed = df['GDPCAP'].apply(lambda x: np.log(x + 1))
total_deaths_transformed = df['TOTAL DEATHS'].apply(lambda x: np.log(x + 1))
```

In [18]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (15, 4))

ax1.hist(df['GDPCAP'])
ax2.hist(gdp_transformed,color = "skyblue")
ax1.set_title("GDP per Capita before the log transformation")
ax2.set_title("GDP per Capita after the log transformation")
ax1.set_ylabel("GDP")

fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (15, 4))

ax1.hist(df['TOTAL DEATHS'])
ax2.hist(total_deaths_transformed,color = "skyblue")
ax1.set_title("Total Number of Deaths before the log transformation")
ax2.set_title("Total Number of Deaths after the log transformation")
ax1.set_ylabel("Total Number of Deaths")
```

Out[18]:
```
Text(0, 0.5, 'Total Number of Deaths')
```





In [19]:
```
df['GDPCAP'] = gdp_transformed
df['TOTAL DEATHS'] = total_deaths_transformed
```

Scale Data - Data Normalization

This ensures that each feature is treated equally when applying supervised learners.

In [20]:
```
scaler = MinMaxScaler()
numerical = ['TOTAL DEATHS', 'GDPCAP']
features_log_minmax_transform = pd.DataFrame(data = df)
features_log_minmax_transform[numerical] = scaler.fit_transform(df[numerical])
features_log_minmax_transform
```

Out[20]:

	CODE	COUNTRY	DATE	TOTAL CASES	TOTAL DEATHS	stringency_index	POPULATION	GDPCAP	HDI
0	AFG	Afghanistan	2019-12-31	0.0	0.000000	0.00	38928341	0.193801	0.498
1	AFG	Afghanistan	2020-01-01	0.0	0.000000	0.00	38928341	0.193801	0.498
2	AFG	Afghanistan	2020-01-02	0.0	0.000000	0.00	38928341	0.193801	0.498
3	AFG	Afghanistan	2020-01-03	0.0	0.000000	0.00	38928341	0.193801	0.498
4	AFG	Afghanistan	2020-01-04	0.0	0.000000	0.00	38928341	0.193801	0.498
...	...	...	...	...	...	...	...	...	...
50413	ZWE	Zimbabwe	2020-10-15	8055.0	0.443642	76.85	14862927	0.203796	0.535
50414	ZWE	Zimbabwe	2020-10-16	8075.0	0.443642	76.85	14862927	0.203796	0.535
50415	ZWE	Zimbabwe	2020-10-17	8099.0	0.443642	76.85	14862927	0.203796	0.535
50416	ZWE	Zimbabwe	2020-10-18	8110.0	0.443642	76.85	14862927	0.203796	0.535
50417	ZWE	Zimbabwe	2020-10-19	8147.0	0.443642	76.85	14862927	0.203796	0.535

31298 rows × 9 columns

Scale Data - Data Normalization

This ensures that each feature is treated equally when applying supervised learners.

In [20]:
```
scaler = MinMaxScaler()
numerical = ['TOTAL DEATHS', 'GDPCAP']
features_log_minmax_transform = pd.DataFrame(data = df)
features_log_minmax_transform[numerical] = scaler.fit_transform(df[numerical])
features_log_minmax_transform
```

Out[20]:

	CODE	COUNTRY	DATE	TOTAL CASES	TOTAL DEATHS	stringency_index	POPULATION	GDPCAP	HDI
0	AFG	Afghanistan	2019-12-31	0.0	0.000000	0.00	38928341	0.193801	0.498
1	AFG	Afghanistan	2020-01-01	0.0	0.000000	0.00	38928341	0.193801	0.498
2	AFG	Afghanistan	2020-01-02	0.0	0.000000	0.00	38928341	0.193801	0.498
3	AFG	Afghanistan	2020-01-03	0.0	0.000000	0.00	38928341	0.193801	0.498
4	AFG	Afghanistan	2020-01-04	0.0	0.000000	0.00	38928341	0.193801	0.498
...	...	...	...	...	...	...	...	...	...
50413	ZWE	Zimbabwe	2020-10-15	8055.0	0.443642	76.85	14862927	0.203796	0.535
50414	ZWE	Zimbabwe	2020-10-16	8075.0	0.443642	76.85	14862927	0.203796	0.535
50415	ZWE	Zimbabwe	2020-10-17	8099.0	0.443642	76.85	14862927	0.203796	0.535
50416	ZWE	Zimbabwe	2020-10-18	8110.0	0.443642	76.85	14862927	0.203796	0.535
50417	ZWE	Zimbabwe	2020-10-19	8147.0	0.443642	76.85	14862927	0.203796	0.535

31298 rows × 9 columns

After one-hot encoding the location column and the first experiment, I found that the model was overfitting hence I decided to drop that column as it's not necessary for the learning algorithm.

In [21]:
```
X_data =  features_log_minmax_transform[['TOTAL CASES','TOTAL DEATHS','stringency_index','POPULATION','HDI']]
y_data = features_log_minmax_transform['GDPCAP']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size = 0.3, random_state = 42)

print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

```
Training set has 21908 samples.
Testing set has 9390 samples.
```

Experiment Stage

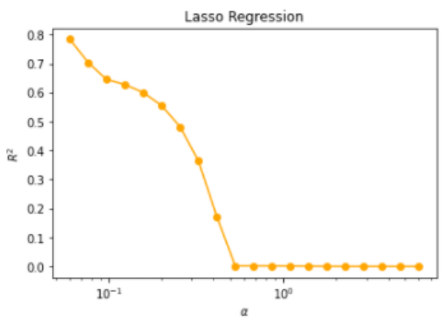
- Training Stage of the following models: Linear Regression, Ridge Regression, Lasso Regression, RidgeCV, LassoCV, Elastic Net
- Compare accuracy scores
- Compare root-mean square errors
- Plot the results: prediction vs actual

In [22]:
```
kf = KFold(shuffle=True, random_state=42, n_splits=3)
```

In [23]:
```
# Linear Regression and K-fold cross validation
s = StandardScaler()
lr = LinearRegression()

X_train_s = s.fit_transform(X_train)
lr.fit(X_train_s, y_train)
X_test = s.transform(X_test)
y_pred = lr.predict(X_test)
score = r2_score(y_test.values, y_pred)

# with pipeline
estimator = Pipeline([("scaler", s),("regression", lr)])
predictions_lr = cross_val_predict(estimator, X_train, y_train, cv=kf)
linear_score = r2_score(y_train, predictions_lr)

linear_score, score #almost identical
```

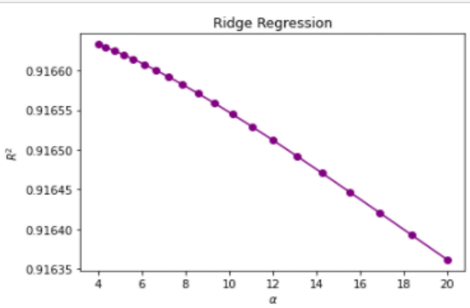
Out[23]:
```
(0.9118952636924754, 0.9118875846079257)
```

In [24]:
```python
# lasso regression and K-fold cross validation
s = StandardScaler()
pf = PolynomialFeatures(degree=3)
scores = []
alphas = np.geomspace(0.06, 6.0, 20)
predictions_lsr = []
for alpha in alphas:
    las = Lasso(alpha=alpha, max_iter=100000)

    estimator = Pipeline([
        ("scaler", s),
        ("make_higher_degree", pf),
        ("lasso_regression", las)])

    predictions_lsr = cross_val_predict(estimator, X_train, y_train, cv = kf)

    score = r2_score(y_train, predictions_lsr)

    scores.append(score)
plt.semilogx(alphas, scores, '-o', color='orange')
plt.title('Lasso Regression')
plt.xlabel('$\\alpha$')
plt.ylabel('$R^2$');
```



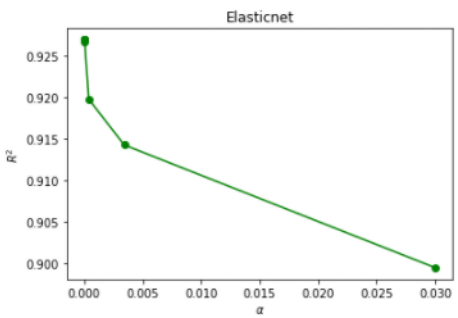
In [25]:
```python
best_estimator = Pipeline([
                    ("scaler", s),
                    ("make_higher_degree", PolynomialFeatures(degree=2)),
                    ("lasso_regression", Lasso(alpha=0.03))])

best_estimator.fit(X_train, y_train)
lasso_score = best_estimator.score(X_train, y_train)
```

In [26]:
```python
# ridge regression and K-fold cross validation
pf = PolynomialFeatures(degree=2)
alphas = np.geomspace(4, 20, 20)
scores=[]
predictions_rr = []
for alpha in alphas:
    ridge = Ridge(alpha=alpha, max_iter=100000)

    estimator = Pipeline([
        ("scaler", s),
        ("polynomial_features", pf),
        ("ridge_regression", ridge)])

    predictions_rr = cross_val_predict(estimator, X_train, y_train, cv = kf)
    score = r2_score(y_train, predictions_rr)
    scores.append(score)

plt.plot(alphas, scores, '-o', color='purple')
plt.title('Ridge Regression')
plt.xlabel('$\\alpha$')
plt.ylabel('$R^2$');
```



In [27]:
```
best_estimator = Pipeline([
                    ("scaler", s),
                    ("make_higher_degree", PolynomialFeatures(degree=2)),
                    ("ridge_regression", Ridge(alpha=0.03))])

best_estimator.fit(X_train, y_train)
ridge_score = best_estimator.score(X_train, y_train)
```

In [28]:
```
# ElasticNet and K-fold cross validation
pf = PolynomialFeatures(degree=3)
alphas = np.geomspace(1e-10, 0.03, 10)
scores=[]
predictions_rr = []
for alpha in alphas:
    Elasticnet = ElasticNet(alpha=alpha, max_iter=100000)

    estimator = Pipeline([
        ("scaler", s),
        ("polynomial_features", pf),
        ("ElasticNet", Elasticnet)])

    predictions_rr = cross_val_predict(estimator, X_train, y_train, cv = kf)
    score = r2_score(y_train, predictions_rr)
    scores.append(score)

plt.plot(alphas, scores, '-o', color='green')
plt.title('Elasticnet')
plt.xlabel('$\\alpha$')
plt.ylabel('$R^2$');
```



In [29]:
```
best_estimator = Pipeline([("scaler", s),("make_higher_degree", PolynomialFeatures(degree=3)),("elasticNet", ElasticNet(alpha=1e
-8))])

best_estimator.fit(X_train, y_train)
elastic_net_score = (best_estimator.score(X_train, y_train))*100
```

In [30]:
```
pd.DataFrame([[linear_score, lasso_score, ridge_score,elastic_net_score]],columns=['Linear', 'Lasso', 'Ridge','Elastic_Net'], in
dex=['score'])
```

Out[30]:

	Linear	Lasso	Ridge	Elastic_Net
score	0.911895	0.894812	0.916781	92.196192

Conclusion: Hypertuned Elastic_net, Lasso and Ridge Reggresion give better results than plain Linear Regression! The best candidate based on score results is Hypertuned Elastic Net !

In [31]:
```
def rmse(ytrue, ypredicted):
    return np.sqrt(mean_squared_error(ytrue, ypredicted))

# Fit a basic linear regression model
linearRegression = LinearRegression().fit(X_train, y_train)
linearRegression_rmse = rmse(y_test, linearRegression.predict(X_test))

# Fit a regular (non-cross validated) Ridge model
alphas = [0.005, 0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 80]
ridgeCV = RidgeCV(alphas=alphas, cv=4).fit(X_train, y_train)
ridgeCV_rmse = rmse(y_test, ridgeCV.predict(X_test))

# Fit a Lasso model using cross validation and determine the optimum value for α
alphas2 = np.array([1e-5, 5e-5, 0.0001, 0.0005])
lassoCV = LassoCV(alphas=alphas2,
                  max_iter=5e4,
                  cv=3).fit(X_train, y_train)
lassoCV_rmse = rmse(y_test, lassoCV.predict(X_test))

# Fit elastic net with the same set of alphas as lasso
l1_ratios = np.linspace(0.1, 0.9, 9)
elasticNetCV = ElasticNetCV(alphas=alphas2,
                            l1_ratio=l1_ratios,
                            max_iter=1e4).fit(X_train, y_train)
elasticNetCV_rmse = rmse(y_test, elasticNetCV.predict(X_test))


rmse_vals = [linearRegression_rmse, ridgeCV_rmse, lassoCV_rmse, elasticNetCV_rmse]

labels = ['Linear', 'Lasso', 'Ridge' 'ElasticNet']

rmse_df = pd.DataFrame([[linearRegression_rmse, ridgeCV_rmse, lassoCV_rmse, elasticNetCV_rmse]],columns=['Linear', 'Lasso', 'Rid
ge', 'ElasticNet'], index=['rmse'])
rmse_df
```

Out[31]:

	Linear	Lasso	Ridge	ElasticNet
rmse	1.599355	1.599337	1.598805	1.598698

Conclusion 2:

Elastic Net gives the smallest Root-mean-square error, however the difference in errors are not significant and almost identical.

The best candidate based on Root-mean-square error and score results is Elastic Net, therefore we recommend elasticNetCV as a final model that best fits the data in terms of accuracy.