Graphic Era Deemed to be University

Dehradun, Uttarakhand

A Mini Project Report

on

GDP(GROSS DOMESDTIC PRODUCT) PREDICTION USING MACHINE LEARNING

Name : SAHIL SAKLANI

University Roll No. : 2016979

Course : B. Tech (CSE)

Semester : 5^{TH}

Guided By : MR.VIKAS TOMER

Department of Computer Science and Engineering

CANDIDATE’S DECLARATION

I hereby certify that the work which is being presented in the project report entitled “GDP prediction using machine learning” in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering of the Graphic Era (Deemed to be University), Dehradun shall be carried out by the under the mentorship of Mr. Vikas Tomer, Assistant Professor, Department of Computer Science and Engineering, Graphic Era (Deemed to be University), Dehradun.

NAME :SAHIL SAKLANI
UNIVERSITY ROLL NO-2016979

Acknowledgment

I take this opportunity to express our sincere gratitude to

MR.VIKAS TOMER for his valuable guidance in this project report without which it would not have been completed. I am very much grateful to him for his untiring assistance in this report and he has been encouraging us in eliminating the errors. The report has been developed as a result of his valuable advice.

I am thankful to my classmates and friends for their support and guidance. I also like to thank researchers and scholars whose papers and thesis have been utilized in this report.

Table of Contents

Chapter No.	Description	Page No.
1.	INTRODUCTION	1-2
2.	METHODOLOGY:PHASE 1	3-28
3.	METHODOLOGY:PHASE 2	29-32
4.	RESULT AND DISCUSSION	33
5.	CONCLUSION AND FUTURE	34
6.	REFRENCES	35

PROJECT INTRODUCTION

GDP Prediction Using Machine Learning

AIM

The aim of this project is to predict the GDP(Gross Domestic Product) of a country using machine learning algorithms.

MOTIVATION

Since GDP is an accurate indicator of size of an economy and the GDP growth rate is probably the single best indicator of economic growth,while GDP per capita has a close correlation with the trend in living standards over time so predicting GDP is quite interesting task to be performed using machine learning tools. After the covid analyzing GDP is very useful for many countries to see their growth in future years so I took this topic.

ABSTRACT

GDP(Gross Domestic Product)

GDP measures the monetary value of final goods and services—that is, those that are bought by the final user—produced in a country in a given period of time (say a quarter or a year). It counts all of the output generated within the borders of a country. GDP is composed of goods and services produced for sale in the market and also includes some nonmarket production, such as defense or education services provided by the government

Machine Learning

Machine learning is a subfield of artificial intelligence, which is broadly defined as the capability of a machine to imitate intelligent human behavior. Artificial

intelligence systems are used to perform complex tasks in a way that is similar to how humans solve problems.

Linear Regression

Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable.

Random Forest

Random forest is a commonly-used machine learning algorithm trademarked by Leo Breiman and Adele Cutler, which combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fueled its adoption, as it handles both classification and regression problems.

Gradient Boosting

Gradient boosting is a machine learning technique used in regression and classification tasks, among others. It gives a prediction model in the form of an ensemble of weak prediction models, which are typically decision trees.[1][2] When a decision tree is the weak learner, the resulting algorithm is called gradient-boosted trees; it usually outperforms random forest.[1][2][3] A gradient-boosted trees model is built in a stage-wise fashion as in other boosting methods, but it generalizes the other methods by allowing optimization of an arbitrary differentiable loss function.

Methodology

In this project we have used certain libraries pandas_datareader, matplotlib, numpy, sklearn, tensorflow , keras, math.

I have done this project in two main parts :

1. Testing 3 machine learning algorithms on the data and selecting the best performer.
2. On the basis of performance I have created a web app using streamlit python lib where we can predict the gdp of the country by giving the values of some of the attributes such as population etc.

1. TESTING PHASE :

1. Import Data and Data Cleaning

For collecting the data of the countries of the world for gdp prediction we have used Kaggle.com. Using this website we have downloaded the csv file of the countries of the world. To read the data from the file we have used pandas data reader .

```
In [1]: import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        %matplotlib inline
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn import metrics
        from sklearn.model_selection import GridSearchCV
        from sklearn.linear_model import LinearRegression
        from sklearn.svm import SVR
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.ensemble import GradientBoostingRegressor
        from sklearn.model_selection import cross_val_score
```

```
In [2]: data = pd.read_csv('countries of the world.csv')
```

```
In [3]: data.head(3)
```

Out[3]:

	Country	Region	Population	Area (sq. mi.)	Pop. Density (per sq. mi.)	Coastline (coast/area ratio)	Net migration	Inf morta (1 birt
0	Afghanistan	ASIA (EX. NEAR EAST)	31056997	647500	48,0	0,00	23,06	163
1	Albania	EASTERN EUROPE	3581655	28748	124,6	1,26	-4,93	21
2	Algeria	NORTHERN AFRICA	32930091	2381740	13,8	0,04	-0,39	

As we are using EDA here which means:

Exploratory Data Analysis (EDA) is an approach to analyze the data using visual techniques. It is used to discover trends, patterns, or to check assumptions with the help of statistical summary and graphical representations.

For this we will perform following tasks on our dataset:

- Getting insights about the dataset
- Handling missing values
- Data Visualization

Now we will see how our data attributes looks like such as the name of the attributes and its data type:

```
In [4]: data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227 entries, 0 to 226
Data columns (total 20 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   Country                             227 non-null    object 
 1   Region                              227 non-null    object 
 2   Population                          227 non-null    int64  
 3   Area (sq. mi.)                      227 non-null    int64  
 4   Pop. Density (per sq. mi.)          227 non-null    object 
 5   Coastline (coast/area ratio)        227 non-null    object 
 6   Net migration                       224 non-null    object 
 7   Infant mortality (per 1000 births)  224 non-null    object 
 8   GDP ($ per capita)                  226 non-null    float64
 9   Literacy (%)                        209 non-null    object 
 10  Phones (per 1000)                   223 non-null    object 
 11  Arable (%)                          225 non-null    object 
 12  Crops (%)                           225 non-null    object 
 13  Other (%)                           225 non-null    object 
 14  Climate                             205 non-null    object 
 15  Birthrate                           224 non-null    object 
 16  Deathrate                           223 non-null    object 
 17  Agriculture                         212 non-null    object 
 18  Industry                            211 non-null    object 
 19  Service                             212 non-null    object 
dtypes: float64(1), int64(2), object(17)
memory usage: 35.6+ KB
```

Here we see an issue; except for 'Country' and 'Region', all other columns are numerical, yet only 'Population', 'Area', and 'GDP' are float/int type; whi;e the rest (15/20) are identified as object type. We need to conver those into float type to continue our data analysis.

Now we will change the datatypes into category and float so that we can apply the machine learning algorithms easily.

In [6]:
```
data.country = data.country.astype('category')

data.region = data.region.astype('category')

data.density = data.density.astype(str)
data.density = data.density.str.replace(",",".").astype(float)

data.coastline_area_ratio = data.coastline_area_ratio.astype(str)
data.coastline_area_ratio = data.coastline_area_ratio.str.replace(",",".").astype(float)

data.net_migration = data.net_migration.astype(str)
data.net_migration = data.net_migration.str.replace(",",".").astype(float)

data.infant_mortality = data.infant_mortality.astype(str)
data.infant_mortality = data.infant_mortality.str.replace(",",".").astype(float)

data.literacy = data.literacy.astype(str)
data.literacy = data.literacy.str.replace(",",".").astype(float)

data.phones = data.phones.astype(str)
data.phones = data.phones.str.replace(",",".").astype(float)

data.arable = data.arable.astype(str)
data.arable = data.arable.str.replace(",",".").astype(float)

data.crops = data.crops.astype(str)
data.crops = data.crops.str.replace(",",".").astype(float)

data.other = data.other.astype(str)
data.other = data.other.str.replace(",",".").astype(float)

data.climate = data.climate.astype(str)
data.climate = data.climate.str.replace(",",".").astype(float)
```

```
data.industry = data.industry.str.replace(",",".").astype(float)

data.service = data.service.astype(str)
data.service = data.service.str.replace(",",".").astype(float)
```

In [7]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227 entries, 0 to 226
Data columns (total 20 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   country                227 non-null    category
 1   region                 227 non-null    category
 2   population             227 non-null    int64
 3   area                   227 non-null    int64
 4   density                227 non-null    float64
 5   coastline_area_ratio   227 non-null    float64
 6   net_migration          224 non-null    float64
 7   infant_mortality       224 non-null    float64
 8   gdp_per_capita         226 non-null    float64
 9   literacy               209 non-null    float64
 10  phones                 223 non-null    float64
 11  arable                 225 non-null    float64
 12  crops                  225 non-null    float64
 13  other                  225 non-null    float64
 14  climate                205 non-null    float64
 15  birthrate              224 non-null    float64
 16  deathrate              223 non-null    float64
 17  agriculture            212 non-null    float64
 18  industry               211 non-null    float64
 19  service                212 non-null    float64
dtypes: category(2), float64(16), int64(2)
memory usage: 43.0 KB
```

Show statistical analysis of our data set

Let's show min, max, mean, std, and count of each column in the dataset.

Now our data description is as follows:

```
In [8]: data.describe()
```

Out[8]:

	population	area	density	coastline_area_ratio	net_migration	infant_mortality	gdp_per_capita	literacy	phones	arable	
count	2.270000e+02	2.270000e+02	227.000000	227.000000	224.000000	224.000000	226.000000	209.000000	223.000000	225.000000	22
mean	2.874028e+07	5.982270e+05	379.047137	21.165330	0.038125	35.506964	9689.823009	82.838278	236.061435	13.797111	4
std	1.178913e+08	1.790282e+06	1660.185825	72.286863	4.889269	35.389899	10049.138513	19.722173	227.991829	13.040402	8
min	7.026000e+03	2.000000e+00	0.000000	0.000000	-20.990000	2.290000	500.000000	17.600000	0.200000	0.000000	0
25%	4.376240e+05	4.647500e+03	29.150000	0.100000	-0.927500	8.150000	1900.000000	70.600000	37.800000	3.220000	0
50%	4.786994e+06	8.660000e+04	78.800000	0.730000	0.000000	21.000000	5550.000000	92.500000	176.200000	10.420000	1
75%	1.749777e+07	4.418110e+05	190.150000	10.345000	0.997500	55.705000	15700.000000	98.000000	389.650000	20.000000	4
max	1.313974e+09	1.707520e+07	16271.500000	870.660000	23.060000	191.190000	55100.000000	100.000000	1035.600000	62.110000	50

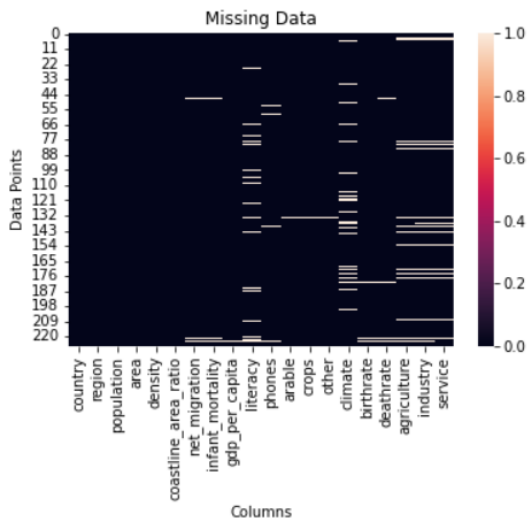
Now we will look for missing data values in the data set:

```
In [9]: print(data.isnull().sum())
```

```
country                 0
region                  0
population              0
area                    0
density                 0
coastline_area_ratio    0
net_migration           3
infant_mortality        3
gdp_per_capita          1
literacy               18
phones                  4
arable                  2
crops                   2
other                   2
climate                22
birthrate               3
deathrate               4
agriculture            15
industry               16
service                15
dtype: int64
```

We will see the missing data values using heat map:

```
In [11]: sns.heatmap(data.isnull()).set(title = 'Missing Data', xlabel = 'Columns', ylabel ='Data Points');
```



net_migration: 3 missing data points. all of them belong to very small nations. We will put zero for those 3.

infant_mortality: 3 missing data points. all of them belong to very small nations. We will put zero for those 3.

gdp_per_capita: 1 missing value. West Sahara, from internet search, their gdp per capita is $2500, and we will put this value into our data set.

literacy: 18 missing values, replaces by the mean literacy of each missing value's region.

phones: 4 missing values, replaces by the mean phones of each missing value's region.

arable, crops, and other: 2 missing values of very small islands, replace with zero.

climate: 22 missing, replace with 0, where zero will represent a 'unknown' value.

birthrate, and deathrate: 3 missing, replace with their region's mean rates, since those rates are per 1000, and not population related.

agricultue, industry, and service: 15 missing values, all belong to very small island nations. After inspection for similar nations, we found that those kind of nations usually have economies that rely heavily on services, with some agricultural and industrial activities. So we will replace the missing values with tthe following: agricultue = 0.15, industry = 0.05. service = 0.8.

```
In [20]: data['net_migration'].fillna(0, inplace=True)
data['infant_mortality'].fillna(0, inplace=True)
data['gdp_per_capita'].fillna(2500, inplace=True)
data['literacy'].fillna(data.groupby('region')['literacy'].transform('mean'), inplace= True)
data['phones'].fillna(data.groupby('region')['phones'].transform('mean'), inplace= True)
data['arable'].fillna(0, inplace=True)
data['crops'].fillna(0, inplace=True)
data['other'].fillna(0, inplace=True)
data['climate'].fillna(0, inplace=True)
data['birthrate'].fillna(data.groupby('region')['birthrate'].transform('mean'), inplace= True)
data['deathrate'].fillna(data.groupby('region')['deathrate'].transform('mean'), inplace= True)
data['agriculture'].fillna(0.17, inplace=True)
data['service'].fillna(0.8, inplace=True)
data['industry'].fillna((1 - data['agriculture'] - data['service']), inplace= True)
```

Now check if any null value exist:

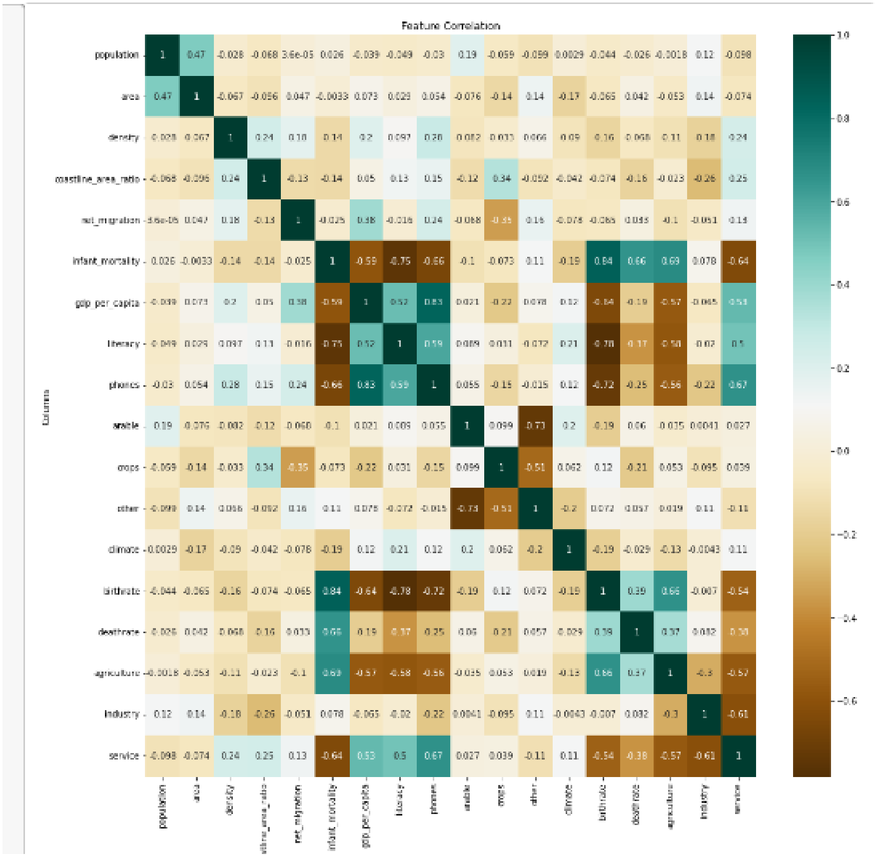
```
In [21]: print(data.isnull().sum())

country                0
region                 0
population             0
area                   0
density                0
coastline_area_ratio   0
net_migration          0
infant_mortality       0
gdp_per_capita         0
literacy               0
phones                 0
arable                 0
crops                  0
other                  0
climate                0
birthrate              0
deathrate              0
agriculture            0
industry               0
service                0
dtype: int64
```

Now our dataset contains no Null values

Now we will plot the correlation heatmap to visualize the strength of relationships between numerical variables. It is used to understand which variables are related to each other and the strength of this relationship.

```
In [22]: fig, ax = plt.subplots(figsize=(16,16))
         sns.heatmap(data.corr(), annot=True, ax=ax, cmap='BrBG').set(
             title = 'Feature Correlation', xlabel = 'Columns', ylabel = 'Columns')
         plt.show()
```

Our correlation heatmap looks like this :



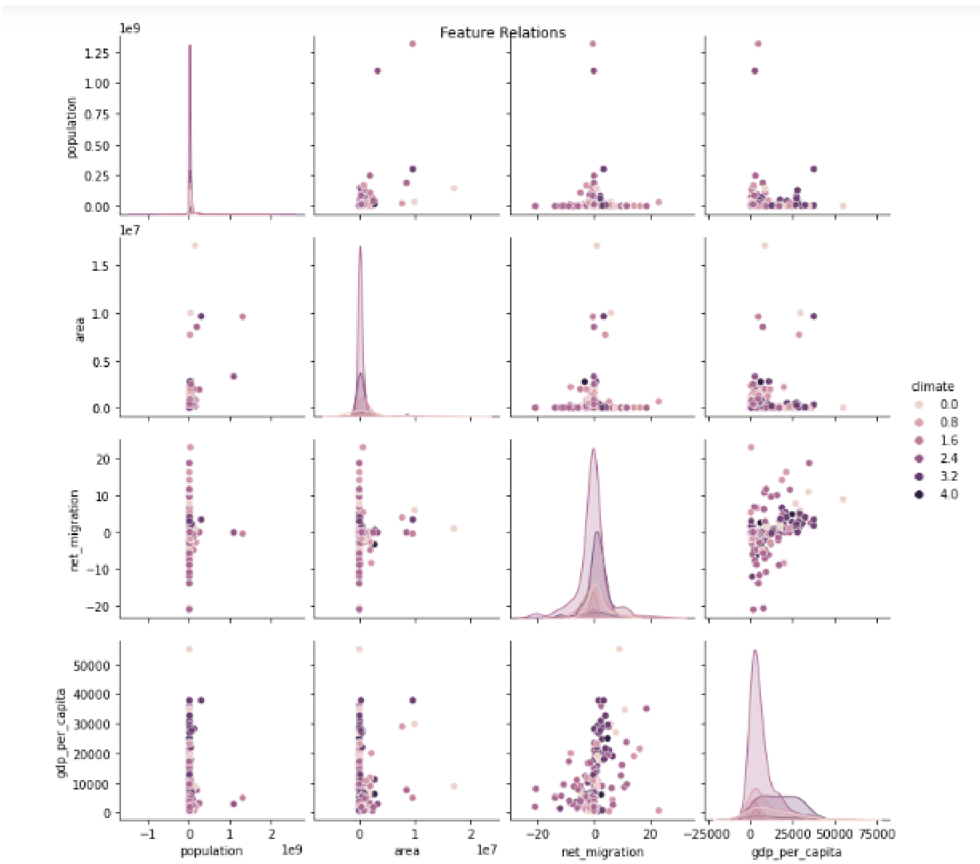
Some insights from the above correlation heatmap:

- expected stronge correlation between infant_mortality and birthrate
- unexpected stronge correlation between infant_mortality and agriculture
- expected stronge correlation between infant_mortality and literacy
- expected stronge correlation between gdp_per_capita and phones
- expected stronge correlation between arable and other (other than crops)

- expected stronge correlation between birthrate and literacy (the less literacy the higher the birthrate)
- unexpected stronge correlation between birthrate and phones

let's now show correlation among a few of our features:

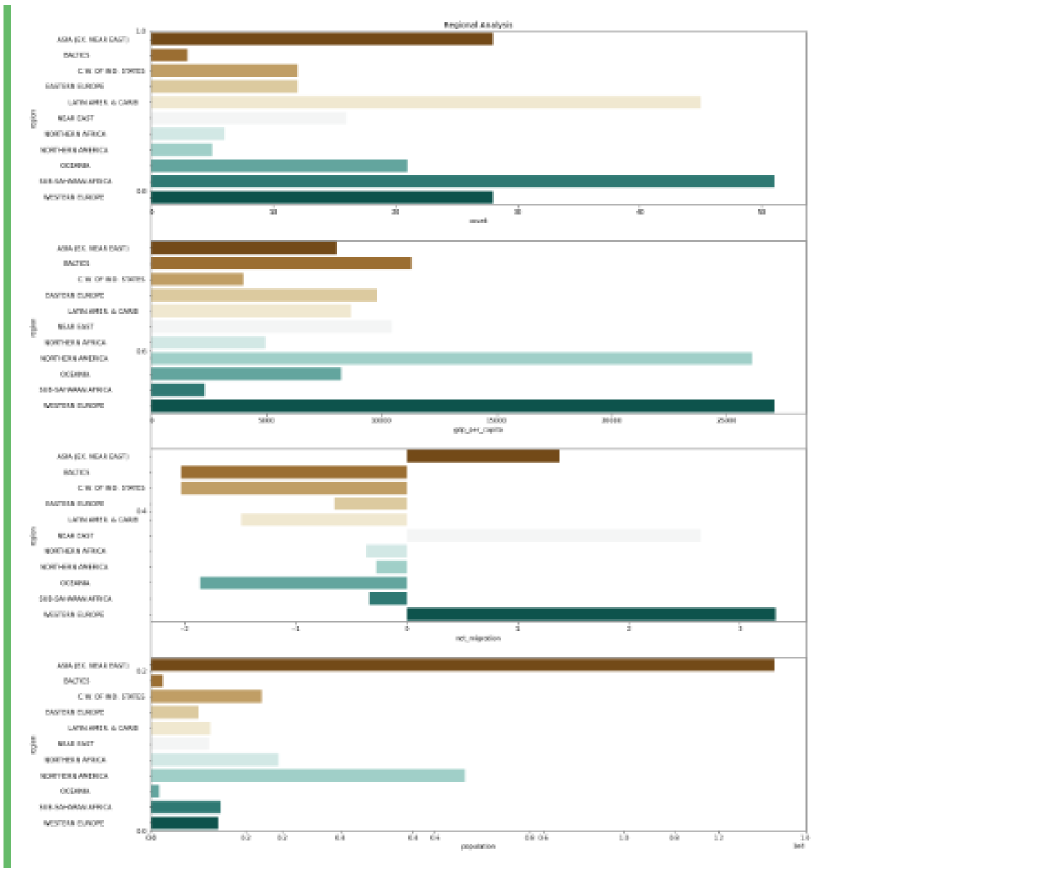
```
In [23]: g = sns.pairplot(data[['population', 'area', 'net_migration', 'gdp_per_capita', 'climate']], hue='climate')
         g.fig.suptitle('Feature Relations')
         plt.show()
```



We can see a fair correlation between GDP and migration, which makes sense, since migrants tend to move to countries with better opportunities and higher GDP per capita.

Now we will do regional analysis:

```
In [24]: fig = plt.figure(figsize=(18, 24))
         plt.title('Regional Analysis')
         ax1 = fig.add_subplot(4, 1, 1)
         ax2 = fig.add_subplot(4, 1, 2)
         ax3 = fig.add_subplot(4, 1, 3)
         ax4 = fig.add_subplot(4, 1, 4)
         sns.countplot(data= data, y= 'region', ax= ax1, palette='BrBG')
         sns.barplot(data= data, y= 'region', x= 'gdp_per_capita', ax= ax2, palette='BrBG', ci= None)
         sns.barplot(data= data, y= 'region', x= 'net_migration', ax= ax3, palette='BrBG', ci= None)
         sns.barplot(data= data, y= 'region', x= 'population', ax= ax4, palette='BrBG', ci= None)
         plt.show()
```



From the above figures, we can notice the following:

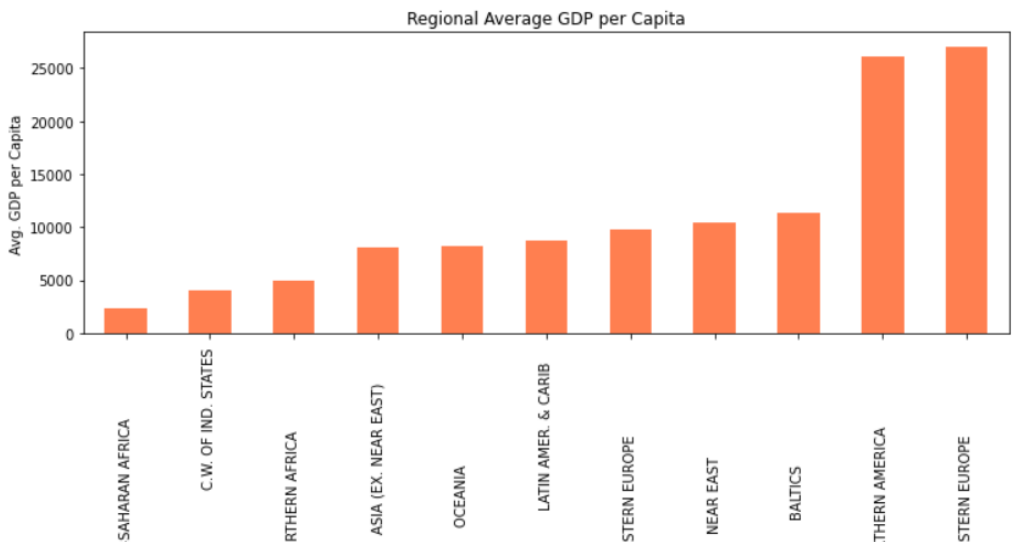
- Sub-Saharian Africa and Latin America regions have the most countries whithen them.
- Western Europe and North America have the highest GDP per capita, while Sub-Saharian Africa has the lowest GDP per capita.

- Asia, North America, and North Europe, are the main regions where migrants from other regions go.
- Asia has the largest population, Oceania has the smallest.

Now we will perform GDP Analysis:

The figure below shows the regional ranking according to the average GDP per capita. As expected, North America and Western Europe have the highest GDP per capita, while Sub Saharian Africa has the lowest, and that may describes the large migration trends in the world in the past decade.

```
In [25]: fig = plt.figure(figsize=(12, 4))
data.groupby('region')['gdp_per_capita'].mean().sort_values().plot(kind='bar', color='coral')
plt.title('Regional Average GDP per Capita')
plt.xlabel("Region")
plt.ylabel('Avg. GDP per Capita')
plt.show()
```

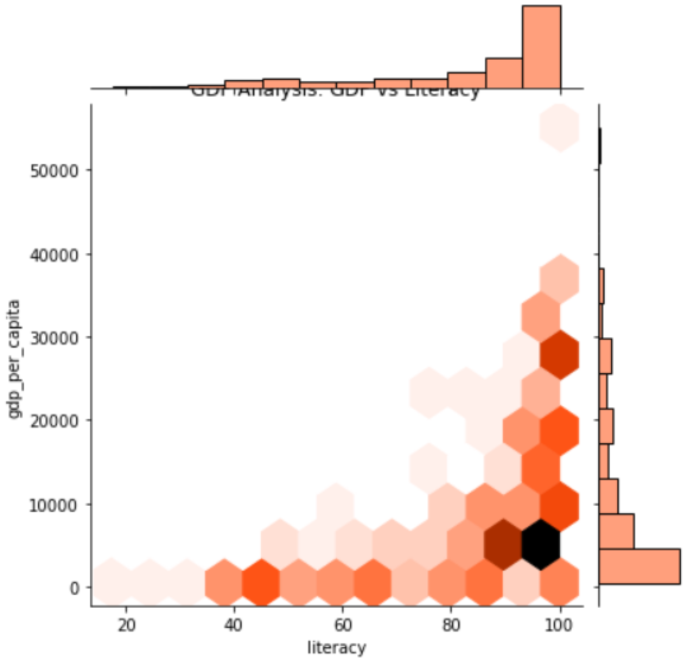


Now we will plot graphs to compare GDP with most related attributes :

1. Literacy vs GDP

In [26]:
```
fig = plt.figure(figsize=(12, 12))
sns.jointplot(data= data, x= 'literacy', y= 'gdp_per_capita', kind= 'hex',color='coral')
plt.title('GDP Analysis: GDP vs Literacy')
plt.show()
```

```
<Figure size 864x864 with 0 Axes>
```

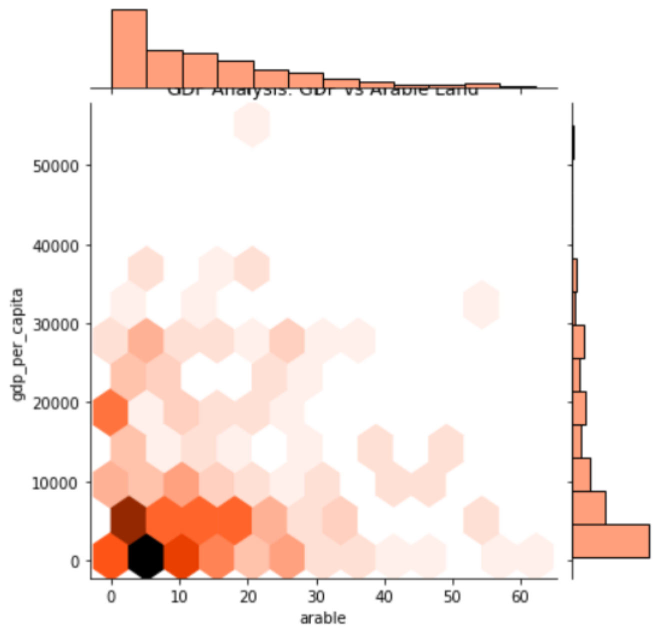


Higher the country's GDP, the more literate the population is, and vive versa .

2. Arable Land vs GDP

In [27]:
```
fig = plt.figure(figsize=(12, 12))
sns.jointplot(data= data, x= 'arable', y= 'gdp_per_capita', kind= 'hex', color='coral')
plt.title('GDP Analysis: GDP vs Arable Land')
plt.show()
```

```
<Figure size 864x864 with 0 Axes>
```

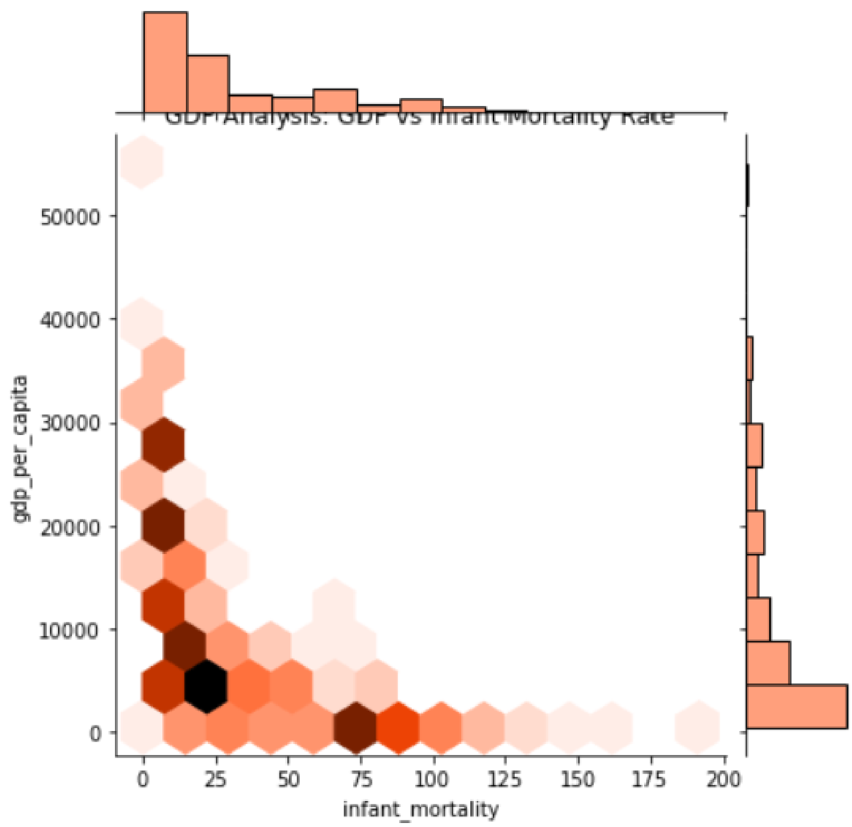


No clear relationship between GDP and percentage of arable land, an indecation that agriculture is not the strongest factor economically, as it used to be for the most of the human history in the last 60000 years.

3. Infant Mortality Rate vs GDP

```
plt.title('GDP Analysis: GDP vs Infant Mortality Rate')
plt.show()
```

```
<Figure size 864x864 with 0 Axes>
```



From the above graph it is clear that poor countries suffer more from infant mortality.

Preprocess the data- Train and Test

In this section we will make our data ready for model training. This will include:

- Transform 'region' column into numerical values.
- Split data set into training and testing parts (80/20), while dropping the countries column (string, and not going to be used to train the models), and separating gdp_per_capita column, where it will be used as labels.
- We will try different splits of our dataset (with/without feature selection, with/without feature scaling.

```
In [29]: data_final = pd.concat([data,pd.get_dummies(data['region'], prefix='region')], axis=1).drop(['region'],axis=1)
         print(data_final.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 227 entries, 0 to 226
Data columns (total 30 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   country                            227 non-null    category
 1   population                         227 non-null    int64
 2   area                               227 non-null    int64
 3   density                            227 non-null    float64
 4   coastline_area_ratio               227 non-null    float64
 5   net_migration                      227 non-null    float64
 6   infant_mortality                   227 non-null    float64
 7   gdp_per_capita                     227 non-null    float64
 8   literacy                           227 non-null    float64
 9   phones                             227 non-null    float64
 10  arable                             227 non-null    float64
 11  crops                              227 non-null    float64
 12  other                              227 non-null    float64
 13  climate                            227 non-null    float64
 14  birthrate                          227 non-null    float64
 15  deathrate                          227 non-null    float64
 16  agriculture                        227 non-null    float64
 17  industry                           227 non-null    float64
 18  service                            227 non-null    float64
 19  region_ASIA (EX. NEAR EAST)        227 non-null    uint8
 20  region_BALTICS                     227 non-null    uint8
 21  region_C.W. OF IND. STATES         227 non-null    uint8
 22  region_EASTERN EUROPE              227 non-null    uint8
 23  region_LATIN AMER. & CARIB         227 non-null    uint8
 24  region_NEAR EAST                   227 non-null    uint8
 25  region_NORTHERN AFRICA             227 non-null    uint8
```

Data Split 1: all of our final dataset, no scaling

In [32]:
```
y = data_final['gdp_per_capita']
X = data_final.drop(['gdp_per_capita','country'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=101)
```

Data Split 2: all of our final dataset, with scaling

In [34]:
```
sc_X = StandardScaler()

X2_train = sc_X.fit_transform(X_train)
X2_test = sc_X.fit_transform(X_test)
y2_train = y_train
y2_test = y_test
```

Data Split 3: feature selected dataset, no scaling

We will select only a portion of our features, the ones with coreelation score larger than -/+ 0.3 with gdp_per_capita.

In [35]:
```
y3 = y
X3 = data_final.drop(['gdp_per_capita','country','population', 'area', 'coastline_area_ratio', 'arable',
                      'crops', 'other', 'climate', 'deathrate', 'industry'], axis=1)

X3_train, X3_test, y3_train, y3_test = train_test_split(X3, y3, test_size=0.2, random_state=101)
```

Data Split 4: feature selected dataset, with scaling

In [36]:
```
sc_X4 = StandardScaler()

X4_train = sc_X4.fit_transform(X3_train)
X4_test = sc_X4.fit_transform(X3_test)
y4_train = y3_train
y4_test = y3_test
```

Now Our Data is ready for applying machine learning algorithms :

1. Linear Regression

Model Training

In [37]:
```
lm1 = LinearRegression()
lm1.fit(X_train,y_train)

lm2 = LinearRegression()
lm2.fit(X2_train,y2_train)

lm3 = LinearRegression()
lm3.fit(X3_train,y3_train)

lm4 = LinearRegression()
lm4.fit(X4_train,y4_train)
```

Out[37]:
```
LinearRegression()
```

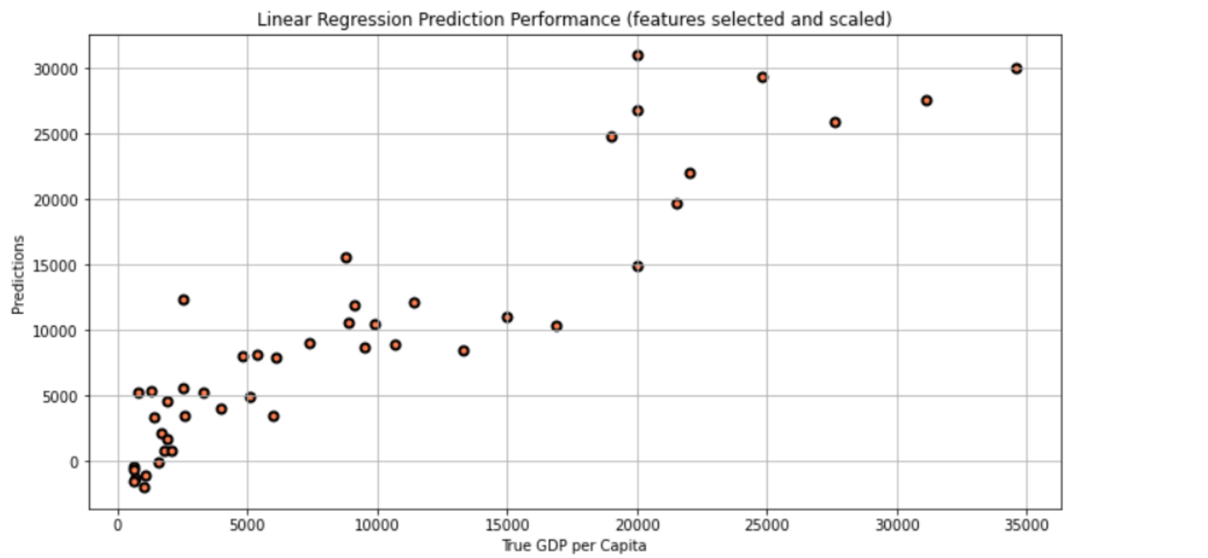
Predictions

In [38]:
```
lm1_pred = lm1.predict(X_test)
lm2_pred = lm2.predict(X2_test)
lm3_pred = lm3.predict(X3_test)
lm4_pred = lm4.predict(X4_test)
```

Evaluation

```
In [39]: print('Linear Regression Performance:')

         print('\nall features, No scaling:')
         print('MAE:', metrics.mean_absolute_error(y_test, lm1_pred))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, lm1_pred)))
         print('R2_Score: ', metrics.r2_score(y_test, lm1_pred))

         print('\nall features, with scaling:')
         print('MAE:', metrics.mean_absolute_error(y2_test, lm2_pred))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y2_test, lm2_pred)))
         print('R2_Score: ', metrics.r2_score(y2_test, lm2_pred))

         print('\nselected features, No scaling:')
         print('MAE:', metrics.mean_absolute_error(y3_test, lm3_pred))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y3_test, lm3_pred)))
         print('R2_Score: ', metrics.r2_score(y3_test, lm3_pred))

         print('\nselected features, with scaling:')
         print('MAE:', metrics.mean_absolute_error(y4_test, lm4_pred))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y4_test, lm4_pred)))
         print('R2_Score: ', metrics.r2_score(y4_test, lm4_pred))

         fig = plt.figure(figsize=(12, 6))
         plt.scatter(y4_test,lm4_pred,color='coral', linewidths=2, edgecolors='k')
         plt.xlabel('True GDP per Capita')
         plt.ylabel('Predictions')
         plt.title('Linear Regression Prediction Performance (features selected and scaled)')
         plt.grid()
         plt.show()
```

```
Linear Regression Performance:

all features, No scaling:
MAE: 330350.8586600643
RMSE: 1570337.5456386511
R2_Score:  -29843.120383337

all features, with scaling:
MAE: 569019.4687587288
RMSE: 1283170.8219650008
R2_Score:  -19925.99011845563

selected features, No scaling:
MAE: 2965.9357229398815
RMSE: 4088.7945802479585
R2_Score:  0.7976685756858989

selected features, with scaling:
MAE: 2879.5213243944386
RMSE: 3756.436588502965
R2_Score:  0.8292247702712091
```



From the metrics above, it is clear that feature selection is essintial for linear regression model training, in order to get acceptable results on this dataset. On the other hand, feature scaling has a small positive effect on LR's prediction performance. we got decent prediction performance from LR with feature selection and scaling.

2. RANDOM FOREST

Model Training

In [40]:
```
rf1 = RandomForestRegressor(random_state=101, n_estimators=200)
rf3 = RandomForestRegressor(random_state=101, n_estimators=200)

rf1.fit(X_train, y_train)
rf3.fit(X3_train, y3_train)
```

Out[40]:
```
RandomForestRegressor(n_estimators=200, random_state=101)
```

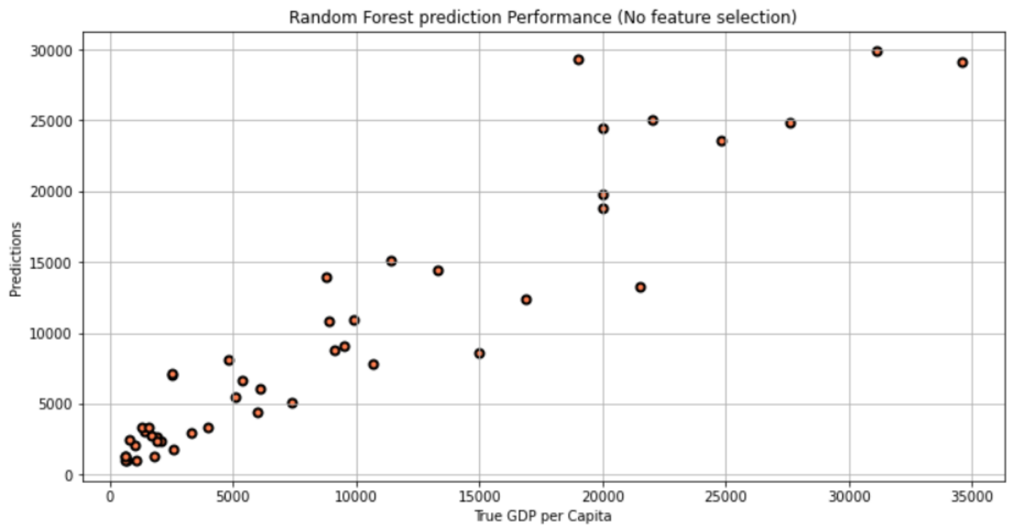
Predictions

In [41]:
```
rf1_pred = rf1.predict(X_test)
rf3_pred = rf3.predict(X3_test)
```

Evaluation

In [42]:
```
print('Random Forest Performance:')

print('\nall features, No scaling:')
print('MAE:', metrics.mean_absolute_error(y_test, rf1_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, rf1_pred)))
print('R2_Score: ', metrics.r2_score(y_test, rf1_pred))

print('\nselected features, No scaling:')
print('MAE:', metrics.mean_absolute_error(y3_test, rf3_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y3_test, rf3_pred)))
print('R2_Score: ', metrics.r2_score(y3_test, rf3_pred))

fig = plt.figure(figsize=(12, 6))
plt.scatter(y_test,rf1_pred,color='coral', linewidths=2, edgecolors='k')
plt.xlabel('True GDP per Capita')
plt.ylabel('Predictions')
plt.title('Random Forest prediction Performance (No feature selection)')
plt.grid()
plt.show()
```

```
Random Forest Performance:

all features, No scaling:
MAE: 2142.1304347826085
RMSE: 3097.1944738255706
R2_Score:  0.8839060185534444

selected features, No scaling:
MAE: 2416.0652173913045
RMSE: 3533.590316058036
R2_Score:  0.8488858452472634
```


Random Forest prediction Performance (No feature selection)
Predictions
True GDP per Capita
0
5000
10000
15000
20000
25000
30000
35000

3. Gradient Boosting Regressor

Model Training

```
In [37]: gbm1 = GradientBoostingRegressor(learning_rate=0.1, n_estimators=100, min_samples_split=2, min_samples_leaf=1, max_depth=3,
                                          subsample=1.0, max_features= None, random_state=101)
         gbm3 = GradientBoostingRegressor(learning_rate=0.1, n_estimators=100, min_samples_split=2, min_samples_leaf=1, max_depth=3,
                                          subsample=1.0, max_features= None, random_state=101)

         gbm1.fit(X_train, y_train)
         gbm3.fit(X3_train, y3_train)

Out[37]: GradientBoostingRegressor(random_state=101)
```

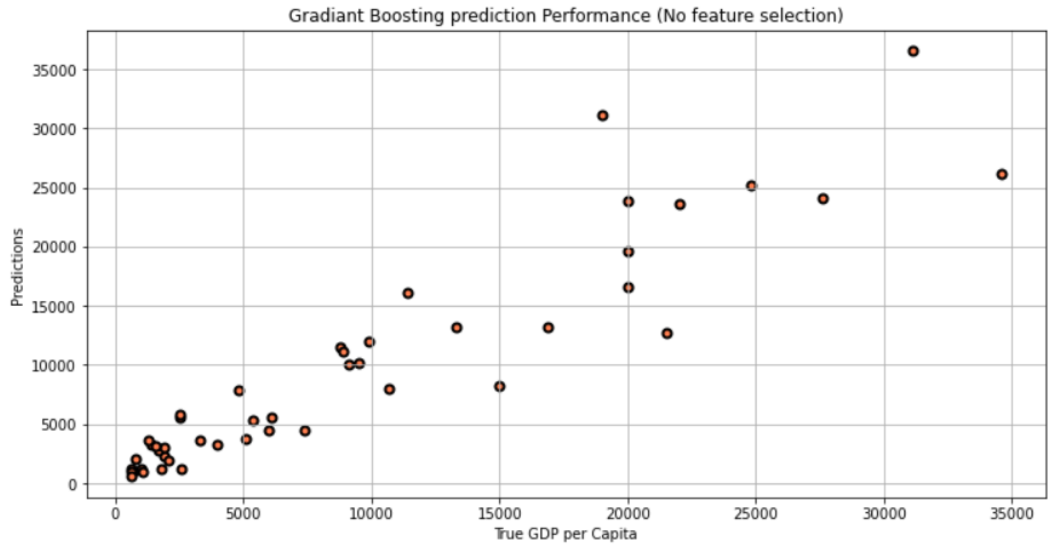
Prediction

```
In [38]: gbm1_pred = gbm1.predict(X_test)
         gbm3_pred = gbm3.predict(X3_test)
```

Evaluation

```
In [39]: print('Gradiant Boosting Performance:')

         print('\nall features, No scaling:')
         print('MAE:', metrics.mean_absolute_error(y_test, gbm1_pred))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, gbm1_pred)))
         print('R2_Score: ', metrics.r2_score(y_test, gbm1_pred))

         print('\nselected features, No scaling:')
         print('MAE:', metrics.mean_absolute_error(y3_test, gbm3_pred))
         print('RMSE:', np.sqrt(metrics.mean_squared_error(y3_test, gbm3_pred)))
         print('R2_Score: ', metrics.r2_score(y3_test, gbm3_pred))

         fig = plt.figure(figsize=(12, 6))
         plt.scatter(y_test,gbm1_pred,color='coral', linewidths=2, edgecolors='k')
         plt.xlabel('True GDP per Capita')
         plt.ylabel('Predictions')
         plt.title('Gradiant Boosting prediction Performance (No feature selection)')
         plt.grid()
         plt.show()
```

```
Gradiant Boosting Performance:

all features, No scaling:
MAE: 2280.4625959347395
RMSE: 3413.6352435789836
R2_Score:  0.8589714692004253

selected features, No scaling:
MAE: 2467.2081266874507
RMSE: 3789.2979753946875
R2_Score:  0.8262238105475073
```



In this project, we used countries_of_the_world dataset to build a GDP predictor. 3 different learning regressors (Linear Regression, Random Forest,Gradient Boosting Regressor) were tested, and we have acheived the best prediction performance using Gradient Boosting, followed by Random Forest.

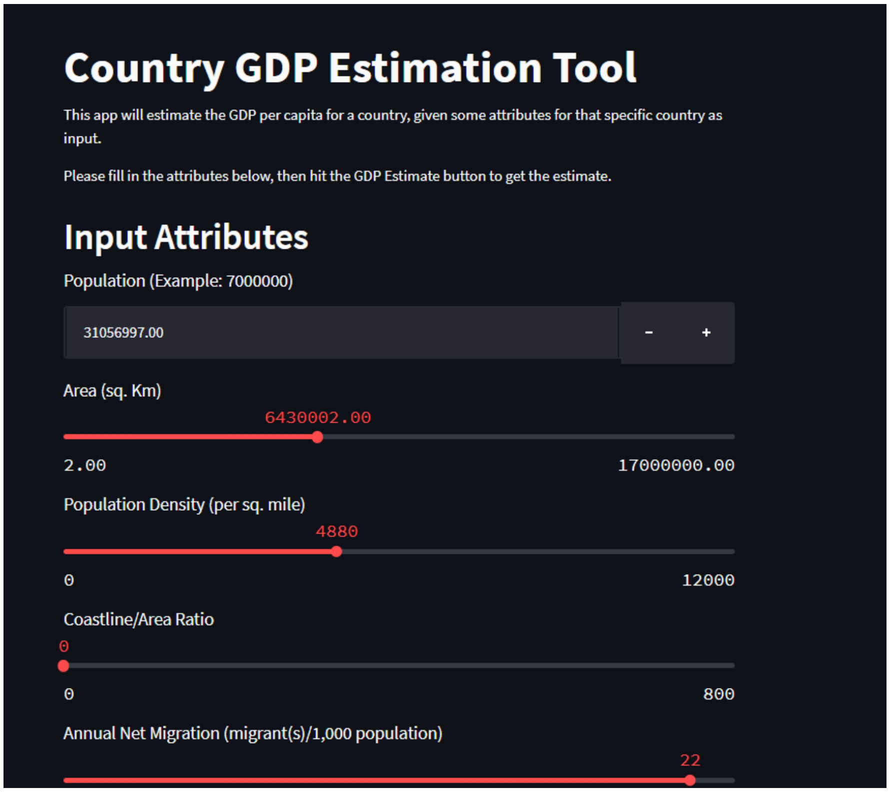
Second phase: GDP prediction web App using Gradient Boosting Regressor Machine Learning Algorithm

1. First we will take some input data from user using user interface of streamlit.

```
st.title('Country GDP Estimation Tool')
st.write('''
         This app will estimate the GDP per capita for a country, given some
         attributes for that specific country as input.

         Please fill in the attributes below, then hit the GDP Estimate button
         to get the estimate.
         ''')

st.header('Input Attributes')
att_popl = st.number_input('Population (Example: 7000000)', min_value=1e4, max_v
att_area = st.slider('Area (sq. Km)', min_value= 2.0, max_value= 17e6, value=6e5
att_dens = st.slider('Population Density (per sq. mile)', min_value= 0, max_valu
att_cost = st.slider('Coastline/Area Ratio', min_value= 0, max_value= 800, value
att_migr = st.slider('Annual Net Migration (migrant(s)/1,000 population)', min_v
att_mort = st.slider('Infant mortality (per 1000 births)', min_value= 0, max_val
att_litr = st.slider('Population literacy Percentage', min_value= 0, max_value=
att_phon = st.slider('Phones per 1000', min_value= 0, max_value= 1000, value=250
att_arab = st.slider('Arable Land (%)', min_value= 0, max_value= 100, value=25,
att_crop = st.slider('Crops Land (%)', min_value= 0, max_value= 100, value=5, st
att_othr = st.slider('Other Land (%)', min_value= 0, max_value= 100, value=70, s
st.text('(Arable, Crops, and Other land should add up to 100%)')
att_clim = st.selectbox('Climate', options=(1, 1.5, 2, 2.5, 3))
st.write('''
         * 1: Mostly hot (like: Egypt and Australia)
         * 1.5: Mostly hot and Tropical (like: China and Cameroon)
         * 2: Mostly tropical (like: The Bahamas and Thailand)
         * 2.5: Mostly cold and Tropical (like: India)
         * 3: Mostly cold (like: Argentina and Belgium)
         '''
```

Data required by user is :

- Population of country
- Area in sq.Km
- Population Density
- Coastline/Area Ratio
- Annual Net Migration
- Infant Mortality
- Population Literacy
- Phones per 1000
- Arable Land
- Crops Land
- Other Land
- Annual Birth Rate
- Annual Death Rate
- Agricultural Economy
- Industrial Economy
- Services Economy
- Its Continent Region


Country GDP Estimation Tool
This app will estimate the GDP per capita for a country, given some attributes for that specific country as input.
Please fill in the attributes below, then hit the GDP Estimate button to get the estimate.
Input Attributes
Population (Example: 7000000)
31056997.00
-
+
Area (sq. Km)
6430002.00
2.00
17000000.00
Population Density (per sq. mile)
4880
0
12000
Coastline/Area Ratio
0
0
800
Annual Net Migration (migrant(s)/1,000 population)
22


Annual Birth Rate (births/1,000)
45
7
50
Annual Death Rate (deaths/1,000)
20
2
30
Agricultural Economy
0.35
0.00
1.00
Industrial Economy
0.25
0.00
1.00
Services Economy
0.40
0.00
1.00
(Agricultural, Industrial, and Services Economy should add up t
Region

2. Now we will take this data into a array so that we can give this user data to our model for predicting the GDP of country.

```
user_input = np.array([att_popl, att_area, att_dens, att_cost, att_migr,
                       att_mort, att_litr, att_phon, att_arab, att_crop,
                       att_othr, att_clim, att_brth, att_deth, att_agrc,
                       att_inds, att_serv, att_regn_1, att_regn_2, att_regn_3,
                       att_regn_4, att_regn_5, att_regn_6, att_regn_7,
                       att_regn_8, att_regn_9, att_regn_10, att_regn_11]).reshape(1,-1)
```

3. Now we will prepare the model as specified in phase 1 on Gradient Boosting.

```
#Data Split
y = data_final['gdp_per_capita']
X = data_final.drop(['gdp_per_capita','country'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=101)

#model training
gbm_opt = GradientBoostingRegressor(learning_rate=0.01, n_estimators=500,
                                    max_depth=5, min_samples_split=10,
                                    min_samples_leaf=1, subsample=0.7,
                                    max_features=7, random_state=101)
RandomForestRegressor(random_state=101, n_estimators=500)
gbm_opt.fit(X_train,y_train)
```

4. Now predict the GDP of the country.

```
#making a prediction
gbm_predictions = gbm_opt.predict(user_input) #user_input is taken from input attrebutes
st.write('The estimated GDP per capita is: ', gbm_predictions)
```

RESULT

As a result of the above model we have successfully predicted the gdp of country by looking at the various data of the country such as its population, area, literacy rate , birth rate, death rate, agicultural land, migration etc.

We have used machine learning algorithm in predicting the GDP value.

The best prediction performance was acheived using Gradient Boosting regressor, using all features in the dataset, and resulted in the following metrics:

Mean Absolute Error (MAE): 2280.46

Root mean squared error (RMSE): 3413.63

R-squared Score (R2_Score): 0.85

```
Gradiant Boosting Performance:

all features, No scaling:
MAE: 2280.4625959347395
RMSE: 3413.6352435789836
R2_Score:  0.8589714692004253

selected features, No scaling:
MAE: 2467.2081266874507
RMSE: 3789.2979753946875
R2_Score:  0.8262238105475073
```

DISCUSSION

As predicting the GDP of country is quite difficult task but using our model we have predicted the value nearly accurate to the actual value so this model can be used to predict the future economy of the country.

CONCLUSION

In this project, we used countries_of_the_world dataset to build a GDP predictor. 3 different learning regressors (Linear Regression, Random Forest, and Gradiant Boosting) were tested, and we have acheived the best prediction performance using Gradient Boosting, followed by Random Forest, and then Linear Regression .

The best prediction performance was acheived using Gradient Boosting regressor, using all features in the dataset, and resulted in the following metrics:

Mean Absolute Error (MAE): 2280.46

Root mean squared error (RMSE): 3413.63

R-squared Score (R2_Score): 0.85

FUTURE SCOPE

If we get more of the resources we can try extending the ranges of grid for gradient boosting and random forest machine learning algorithms.

REFRENCES

1. https://www.tutorialspoint.com/
2. https://www.geeksforgeeks.org/
3. https://www.sciencedirect.com/
4. https://www.kaggle.com/