Project Name: End-To-End ML with Deployment Countries GDP Prediction – Random Forest (RF) – Regression

Table of Contents

4)Model Visualization
5)Feature Importance
6)Model Optimization
7)Model Prediction, Evaluation and Visualization after optimization
8)Feature Importance

Checking the Model Visualization

- Basic Model Evaluation Graphs

Creation of App

Technical Aspect

Installation

Run/How to Use/Steps

Directory Tree/Structure of Project

To Do/Future Scope

Technologies Used/System Requirements/Tech Stack

Download the Material

Conclusion

- Modelling
- Analysis

Credits

Paper Citation

Demo

Data information

```
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
```

Show statistical analysis of our data set

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

```
data.describe()
```

	population	area	density	coastline_area_ratio	net_migration	infant_mortality	gdp_per_capita	literacy	phones	arable	22
count	2.270000e+02	2.270000e+02	227.000000	227.000000	224.000000	224.000000	226.000000	209.000000	223.000000	225.000000	
mean	2.874028e+07	5.982270e+05	379.047137	21.165330	0.038125	35.506964	9689.823009	82.838278	236.061435	13.797111	
std	1.178913e+08	1.790282e+06	1660.185825	72.286863	4.889269	35.389899	10049.138513	19.722173	227.991829	13.040402	
min	7.026000e+03	2.000000e+00	0.000000	0.000000	-20.990000	2.290000	500.000000	17.600000	0.200000	0.000000	
25%	4.376240e+05	4.647500e+03	29.150000	0.100000	-0.927500	8.150000	1900.000000	70.600000	37.800000	3.220000	
50%	4.786994e+06	8.660000e+04	78.800000	0.730000	0.000000	21.000000	5550.000000	92.500000	176.200000	10.420000	
75%	1.749777e+07	4.418110e+05	190.150000	10.345000	0.997500	55.705000	15700.000000	98.000000	389.650000	20.000000	
max	1.313974e+09	1.707520e+07	16271.500000	870.660000	23.060000	191.190000	55100.000000	100.000000	1035.600000	62.110000	5

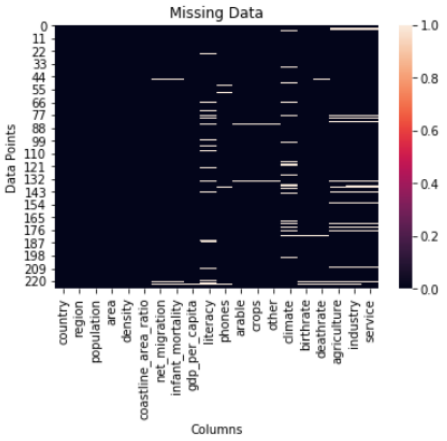
Show if there are missing datapoints

```
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
```

```
sns.heatmap(data.isnull()).set(title = 'Missing Data', xlabel = 'Columns', ylabel = 'Data Points')
```

```
[Text(33.0, 0.5, 'Data Points'),
 Text(0.5, 14.09375, 'Columns'),
 Text(0.5, 1, 'Missing Data')]
```



We can see from above that we have some missing data points, but it is not extensive. 14/20 of our columns have missing data points, the maximum percentage of missing data is in the 'Climate' column, and it is less than 10% (22/227).

Invistigating undefined features

We need to understand what different values in the Climate, agriculture, industry, and service columns refer to.

```
data.loc[:, ['country', 'region', 'climate', 'agriculture', 'industry', 'service']].head()
```

	country	region	climate	agriculture	industry	service
0	Afghanistan	ASIA (EX. NEAR EAST)	1.0	0.380	0.240	0.380
1	Albania	EASTERN EUROPE	3.0	0.232	0.188	0.579
2	Algeria	NORTHERN AFRICA	1.0	0.101	0.600	0.298
3	American Samoa	OCEANIA	2.0	NaN	NaN	NaN
4	Andorra	WESTERN EUROPE	3.0	NaN	NaN	NaN

It is clear here that the values in (agriculture, industry, and service) columns are the percentages those different sectors in the economic activity in each country. For example: agriculture is generating 38% of Afganistan's GDP, industry generates 24%, while service generates 38%; the total is 100%.

```
data.climate.unique()
```

```
array([1. , 3. , 2. , nan, 4. , 1.5, 2.5])
```

```
h1 = data.loc[:, ['country', 'region', 'climate']][data.climate == 1].head()
h2 = data.loc[:, ['country', 'region', 'climate']][data.climate == 2].head()
h3 = data.loc[:, ['country', 'region', 'climate']][data.climate == 3].head()
h4 = data.loc[:, ['country', 'region', 'climate']][data.climate == 4].head()
h5 = data.loc[:, ['country', 'region', 'climate']][data.climate == 1.5].head()
h6 = data.loc[:, ['country', 'region', 'climate']][data.climate == 2.5].head()
pd.concat([h1, h2, h3, h4, h5, h6])
```

	country	region	climate
0	Afghanistan	ASIA (EX. NEAR EAST)	1.0
2	Algeria	NORTHERN AFRICA	1.0
11	Australia	OCEANIA	1.0
13	Azerbaijan	C.W. OF IND. STATES	1.0
15	Bahrain	NEAR EAST	1.0
3	American Samoa	OCEANIA	2.0
6	Anguilla	LATIN AMER. & CARIB	2.0
7	Antigua & Barbuda	LATIN AMER. & CARIB	2.0
10	Aruba	LATIN AMER. & CARIB	2.0
14	Bahamas, The	LATIN AMER. & CARIB	2.0
1	Albania	EASTERN EUROPE	3.0
4	Andorra	WESTERN EUROPE	3.0
8	Argentina	LATIN AMER. & CARIB	3.0
12	Austria	WESTERN EUROPE	3.0
19	Belgium	WESTERN EUROPE	3.0
9	Armenia	C.W. OF IND. STATES	4.0
18	Belarus	C.W. OF IND. STATES	4.0
25	Bosnia & Herzegovina	EASTERN EUROPE	4.0
69	France	WESTERN EUROPE	4.0
106	Kazakhstan	C.W. OF IND. STATES	4.0
24	Bolivia	LATIN AMER. & CARIB	1.5
35	Cameroon	SUB-SAHARAN AFRICA	1.5
42	China	ASIA (EX. NEAR EAST)	1.5
63	Eritrea	SUB-SAHARAN AFRICA	1.5
107	Kenya	SUB-SAHARAN AFRICA	1.5
94	India	ASIA (EX. NEAR EAST)	2.5
112	Kyrgyzstan	C.W. OF IND. STATES	2.5
154	Swaziland	SUB-SAHARAN AFRICA	2.5

So, along with nan (representing missing data), climate has 6 unique values, and they are: 1, 1.5, 2, 2.5, 3, and 4. our observations:

1. Countries with mostly desert/hot climate have 1
2. Countries with mostly tropical climate have 2
3. Countries with mostly cold/cool Climate have 3
4. Countries with Climate almost equally divided between hot and tropical have 1.5
5. Countries with Climate almost equally divided between cold and tropical have 2.5
6. Countries under 'Climate' = 4, are also belonging to cold/cool climate group; It is not metioned in the dataset source why this group is separate from group 3 ; yet we will compine both groups together in the data cleaning section of the project.
7. There are 22 countries with null values for the climate column, those will be replaces by 0 in a later step, where 0 will represent 'unknown' value.

```
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)
```

Let's check our missing data if any:

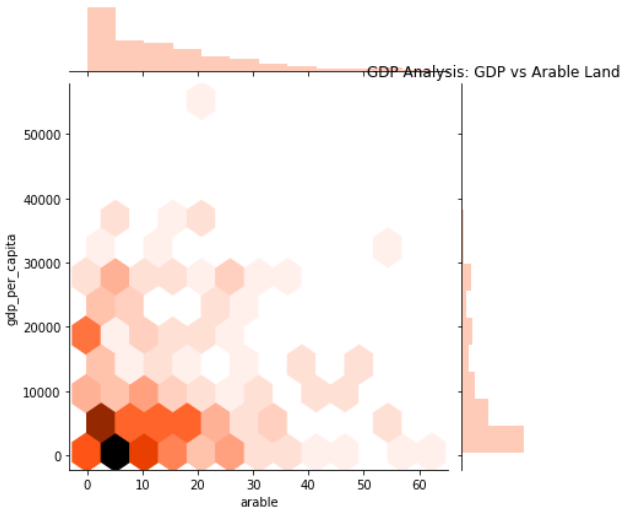
```
print(data.isnull().sum())
```

```
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
```

No missing data anymore.

```
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()
```

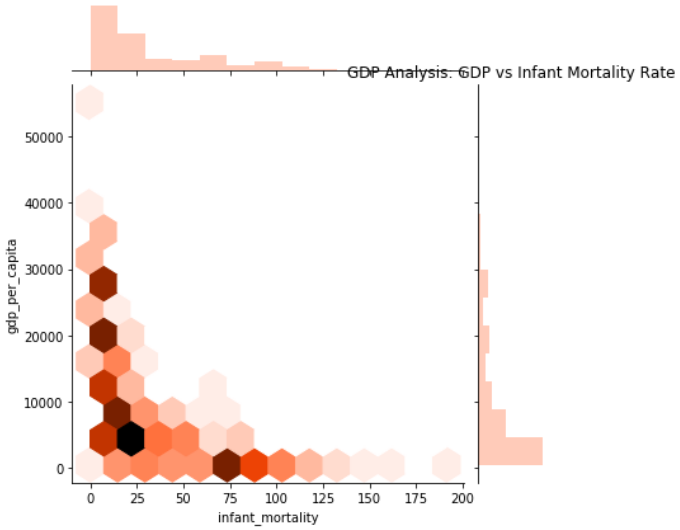
```
<Figure size 864x864 with 0 Axes>
```



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.

```
fig = plt.figure(figsize=(12, 12))
sns.jointplot(data= data, x= 'infant_mortality', y= 'gdp_per_capita', kind= 'hex',color='coral')
plt.title('GDP Analysis: GDP vs Infant Mortality Rate')
plt.show()
```

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



from the figure above, it is very clear that poor countries suffer more from infant mortality.

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

```
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

```
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.

```
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

```
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
```

SVM

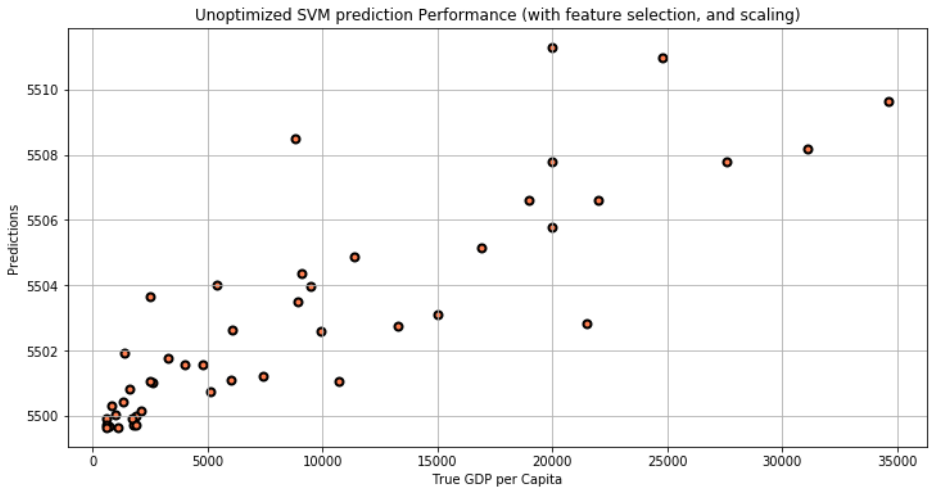
Model Training

```
svm1 = SVR(kernel='rbf')
svm1.fit(X_train,y_train)

svm2 = SVR(kernel='rbf')
svm2.fit(X2_train,y2_train)

svm3 = SVR(kernel='rbf')
svm3.fit(X3_train,y3_train)

svm4 = SVR(kernel='rbf')
svm4.fit(X4_train,y4_train)
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

Predictions

```
svm1_pred = svm1.predict(X_test)
svm2_pred = svm2.predict(X2_test)
svm3_pred = svm3.predict(X3_test)
svm4_pred = svm4.predict(X4_test)
```



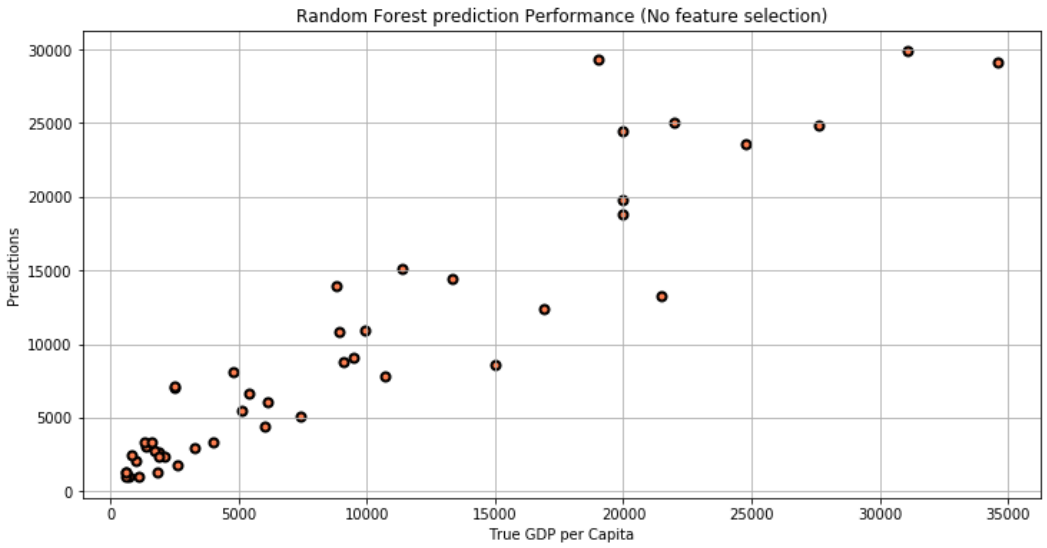
Feature scaling, and feature selection, made almost no difference in the prediction performance of the SVM algorithm.

The results of SVM is worse than that of Linear Regression, so we will try to improve SVM's performance by optimizing its parameters using grid search.

Evaluation

```
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
```



Optimization

We will use grid search in order to obtain good parameters for our RF regressor. Of course our optimization here will be limited due to time and computing power constraints. The parameters we will optimiz are:

- n-estimators
- min_samples_leaf
- max_features
- bootstrap

```
rf_param_grid = {'max_features': ['sqrt', 'auto'],
              'min_samples_leaf': [1, 3, 5],
              'n_estimators': [100, 500, 1000],
             'bootstrap': [False, True]}
```

```
rf_grid = GridSearchCV(estimator= RandomForestRegressor(), param_grid = rf_param_grid,  n_jobs=-1, verbose=0)
```

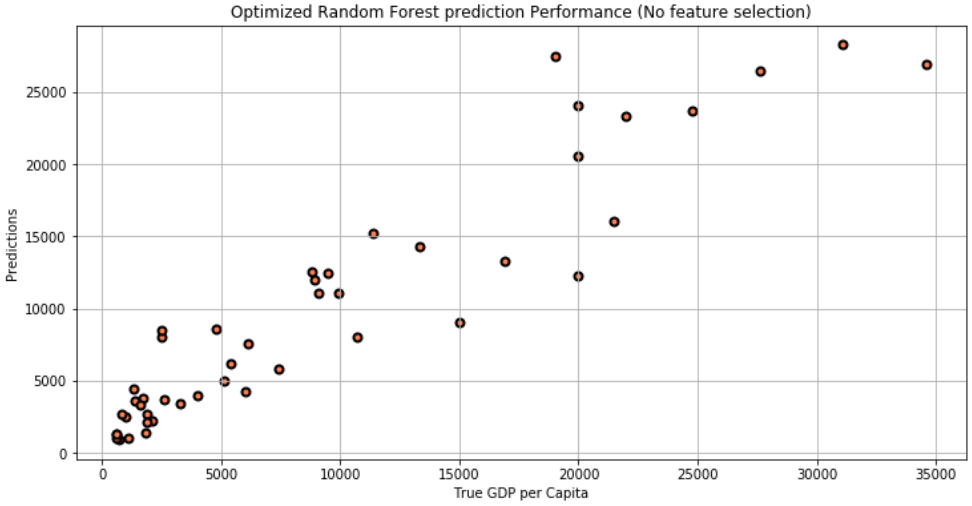
```
rf_grid.fit(X_train,y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             max_samples=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators=100, n_jobs=None,
                                             oob_score=False, random_state=None,
                                             verbose=0, warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'bootstrap': [False, True],
                         'max_features': ['sqrt', 'auto'],
                         'min_samples_leaf': [1, 3, 5],
                         'n_estimators': [100, 500, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
rf_grid_predictions = rf_grid.predict(X_test)
```

```
print('MAE:', metrics.mean_absolute_error(y_test, rf_grid_predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, rf_grid_predictions)))
print('R2_Score: ', metrics.r2_score(y_test, rf_grid_predictions))
fig = plt.figure(figsize=(12, 6))
plt.scatter(y_test,rf_grid_predictions,color='coral', linewidths=2, edgecolors='k')
plt.xlabel('True GDP per Capita')
plt.ylabel('Predictions')
plt.title('Optimized Random Forest prediction Performance (No feature selection)')
plt.grid()
plt.show()
```

```
MAE: 2360.747826086956
RMSE: 3219.9245794569947
R2_Score:  0.8745229924626883
```

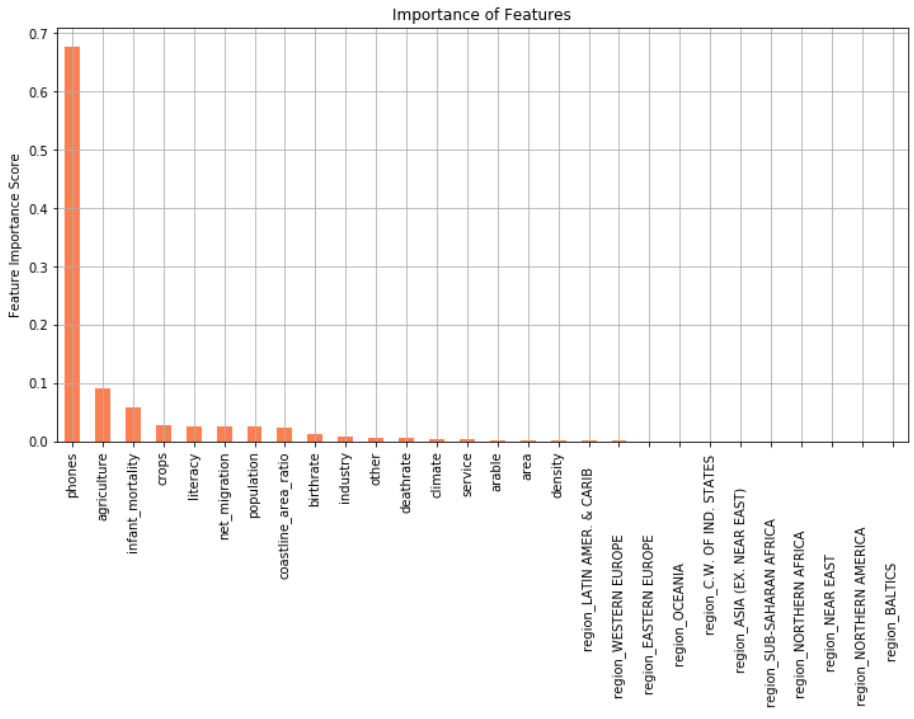


We can see that the optimization process on RF regressor has not changed the performance in a noticebale manner, yet the slight change was actually to the worst, that is probably because our initial parameters were already very close to the optimum ones.

Feature Importance

We can plot how the GBM regressor sees the importance of different features in the dataset.

```
feat_imp = pd.Series(gbm1.feature_importances_, list(X_train)).sort_values(ascending=False)
fig = plt.figure(figsize=(12, 6))
feat_imp.plot(kind='bar', title='Importance of Features', color= 'coral')
plt.ylabel('Feature Importance Score')
plt.grid()
plt.show()
```



Live Webapp Demo Link

https://share.streamlit.io/monicadesai-tech/project_78/main/app.py
Deployment on Heroku: https://mlgdp.herokuapp.com/

Abstract

The purpose of this report will be to use the countries of the world.csv to predict GDP of given country depending on details provided by user. This can be used to gain insight into how and why GDP is such at a given time. This can also be used as a model to gain a marketing advantage, by advertisement targeting those which countries have higher GDP because of few laws or reasons or targeting those countries which are developing nations and their impact and relations with other countries. Countries GDP Prediction is a regression problem, where using the various parameters or inputs from user model will supply result.

This is diving into Countries GDP Prediction through Machine Learning Concept.
End to End Project means that it is step by step process, starts with data collection, EDA, Data Preparation which includes cleaning and transforming then selecting, training and saving ML Models, Cross-validation and Hyper-Parameter Tuning and developing web service then Deployment for end users to use it anytime and anywhere.
This repository contains the code for Countries GDP Prediction using python's various libraries.
It used numpy, pandas, matplotlib, seaborn, sklearn and streamlit libraries.
These libraries help to perform individually one particular functionality.
Numpy is used for working with arrays. It stands for Numerical Python.
Pandas objects rely heavily on Numpy objects.
Matplotlib is a plotting library.
Seaborn is data visualization library based on matplotlib.
Sklearn has 100 to 200 models.
Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning.
The purpose of creating this repository is to gain insights into Complete ML Project.
These python libraries raised knowledge in discovering these libraries with practical use of it.
It leads to growth in my ML repository.
These above screenshots and video in Video_File Folder will help you to understand flow of output.

Motivation

The reason behind building this project is, because I personally like to travel and explore new places. At the same time, to gain knowledge about respective country, its culture, their language, its currency and about the government. As this indicates power of citizens/unity or discipline mannerism in individuals and reason that it lacks in other countries. So, I created Countries GDP Prediction Project to gain insights from IT perspective to know working of the model. Hence, I continue to spread tech wings in IT Heaven.

Acknowledgement

Dataset Available: https://www.kaggle.com/fernandol/countries-of-the-world

The Data

Show data header

```
data.head(3)
```

	Country	Region	Population	Area (sq. mi.)	Pop. Density (per sq. mi.)	Coastline (coast/area ratio)	Net migration	Infant mortality (per 1000 births)	GDP ($ per capita)	Literacy (%)	Phones (per 1000)	Arable (%)	Crops (%)	Other (%)	Climate	Birthrate
0	Afghanistan	ASIA (EX. NEAR EAST)	31056997	647500	48,0	0,00	23,06	163,07	700.0	36,0	3,2	12,13	0,22	87,65	1	46,6
1	Albania	EASTERN EUROPE	3581655	28748	124,6	1,26	-4,93	21,52	4500.0	86,5	71,2	21,09	4,42	74,49	3	15,11
2	Algeria	NORTHERN AFRICA	32930091	2381740	13,8	0,04	-0,39	31	6000.0	70,0	78,1	3,22	0,25	96,53	1	17,14

It has 20 columns with three columns being numeric and rest all categorical.

Data information

```
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.

Also, column names are very long, We have to fix them as well.

Re-naming Column names.

Fix column names

Many columns in the dataset have long names, we will change them to be shorter and better describtive.

```
data.columns = (["country","region","population","area","density","coastline_area_ratio","net_migration","infant_mortality","gdp_
                 "literacy","phones","arable","crops","other","climate","birthrate","deathrate","agriculture","industry",
                 "service"])
```

Fix data types

Many columns in that dataset have *object* as type. We will fix this by assigning float/string types to them.

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

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

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

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

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

```
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: 44.9 KB
```

Now that looks good.

It displays number of missing/ null values in each column.

Show if there are missing datapoints

```
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
```

Analysis of Data

Let's start by doing a general analysis of the data as a whole.

- Basic Statistics

Show statistical analysis of our data set

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

```
data.describe()
```

	population	area	density	coastline_area_ratio	net_migration	infant_mortality	gdp_per_capita	literacy	phones	arable	22
count	2.270000e+02	2.270000e+02	227.000000	227.000000	224.000000	224.000000	226.000000	209.000000	223.000000	225.000000	
mean	2.874028e+07	5.982270e+05	379.047137	21.165330	0.038125	35.506964	9689.823009	82.838278	236.061435	13.797111	
std	1.178913e+08	1.790282e+06	1660.185825	72.286863	4.889269	35.389899	10049.138513	19.722173	227.991829	13.040402	
min	7.026000e+03	2.000000e+00	0.000000	0.000000	-20.990000	2.290000	500.000000	17.600000	0.200000	0.000000	
25%	4.376240e+05	4.647500e+03	29.150000	0.100000	-0.927500	8.150000	1900.000000	70.600000	37.800000	3.220000	
50%	4.786994e+06	8.660000e+04	78.800000	0.730000	0.000000	21.000000	5550.000000	92.500000	176.200000	10.420000	
75%	1.749777e+07	4.418110e+05	190.150000	10.345000	0.997500	55.705000	15700.000000	98.000000	389.650000	20.000000	
max	1.313974e+09	1.707520e+07	16271.500000	870.660000	23.060000	191.190000	55100.000000	100.000000	1035.600000	62.110000	5

Data validation checks that data are valid, sensible, reasonable, and secure before they are processed.

Data Validity Check

In order to have more certainty, we will pick a few countries and attributes (features) in random, and we will do some internet reseach to make sure the values in our data set is nottotally wrong.

Countries to check: Brazil, Cuba, Italy, Libya, Vietnam. The features we will check are: (p)opulation, (a)rea, (c)oastline/Area ratio, and (G)DP Source of information: https://www.jetpunk.com/info/countries-by-coastline, https://en.wikipedia.org/

From those sources: Brazil: (p)210,147,125 (a)3,287,956 (c)0.0035 (G)17,016 Cuba: (p)011,209,628 (a)00042,426 (c)0.0842 (G)08,822 Italy: (p)060,317,116 (a)00116,350 (c)0.0321 (G)40,470 Libya: (p)006,871,292 (a)00679,363 (c)0.0016 (G)07,803 Vietnam: (p)095,545,962 (a)00127,882 (c)0.0280 (G)08,066

Now let's compare those values to the ones we have in our data set:

```
data.loc[[27,51, 101, 118, 219], ['country', 'population', 'area', 'coastline_area_ratio', 'gdp_per_capita']]
```

	country	population	area	coastline_area_ratio	gdp_per_capita
27	Brazil	188078227	8511965	0.09	7600.0
51	Cuba	11382820	110860	3.37	2900.0
101	Italy	58133509	301230	2.52	26700.0
118	Libya	5900754	1759540	0.10	6400.0
219	Vietnam	84402966	329560	1.05	2500.0

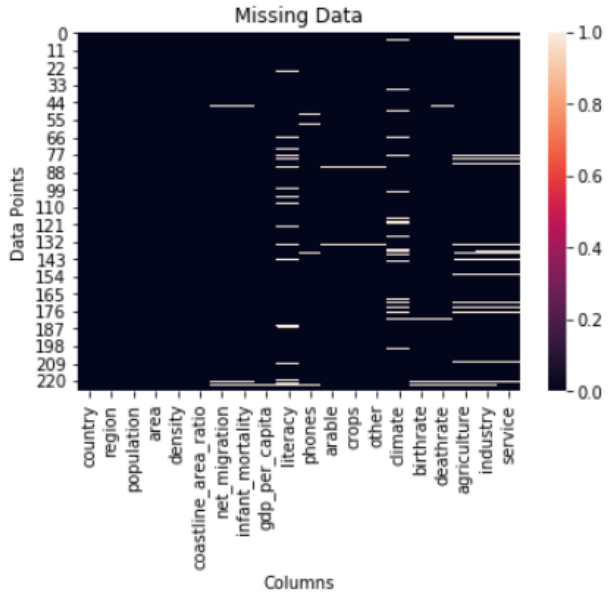
From this quick inspection above, we can see that our data set is a bit old. GDP and Population are a few years old (not updated), while the area and costline ratio are actually in Km and not miles as mentioned in the colum names.

- Graphing of Features

Graph Set 1

```
sns.heatmap(data.isnull()).set(title = 'Missing Data', xlabel = 'Columns', ylabel = 'Data Points')
```

```
[Text(33.0, 0.5, 'Data Points'),
 Text(0.5, 14.09375, 'Columns'),
 Text(0.5, 1, 'Missing Data')]
```

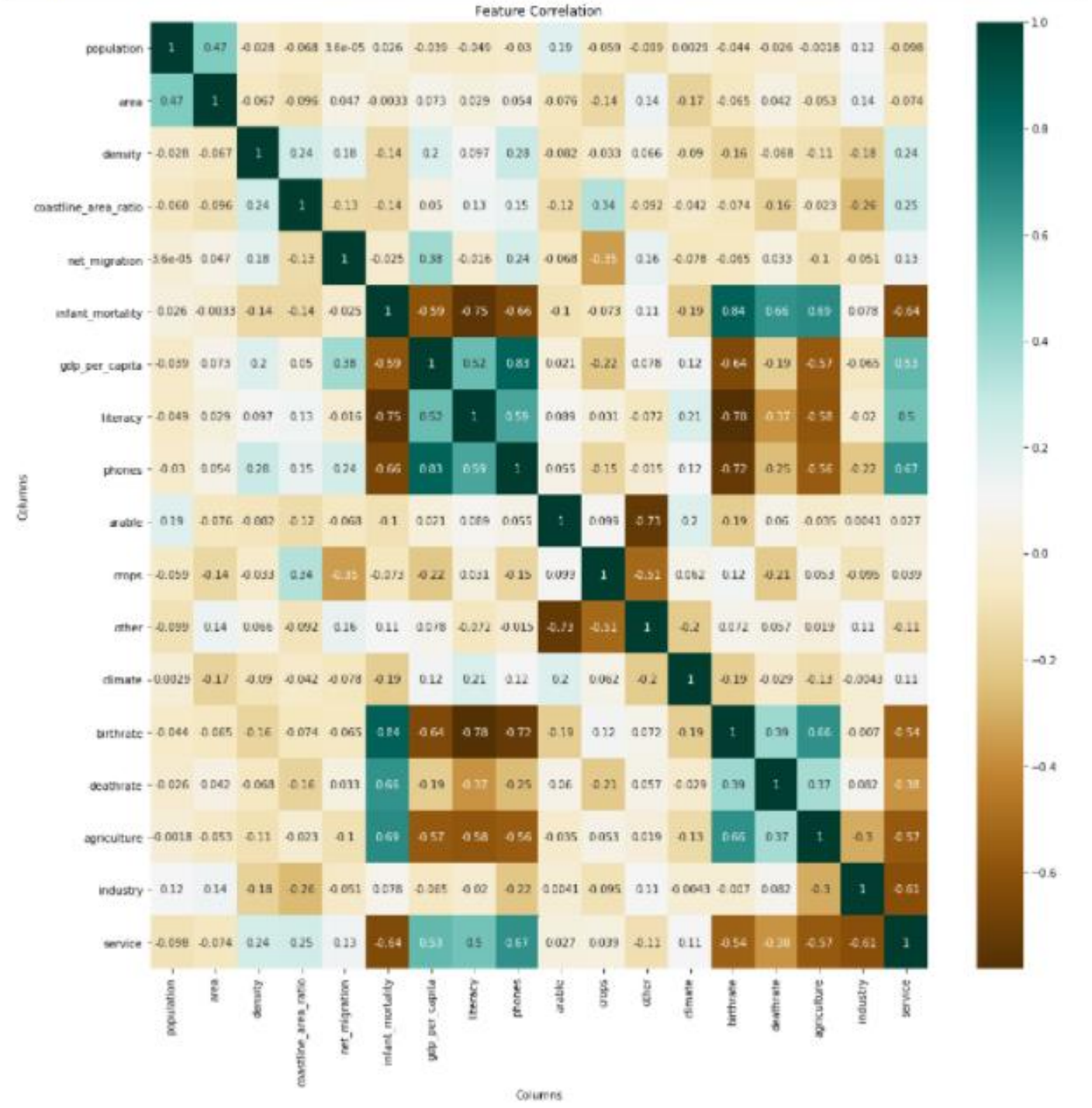


We can see from above that we have some missing data points, but it is not extensive. 14/20 of our columns have missing data percentage of missing data is in the 'Climate' column, and it is less than 10% (22/227).

We can see from above that we have some missing data points but it is not extensive, 14/20 of our columns have missing data percentage of missing data is in the 'Climate' column and it is less than 10% (22/227).

Graph Set 2

```
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()
```

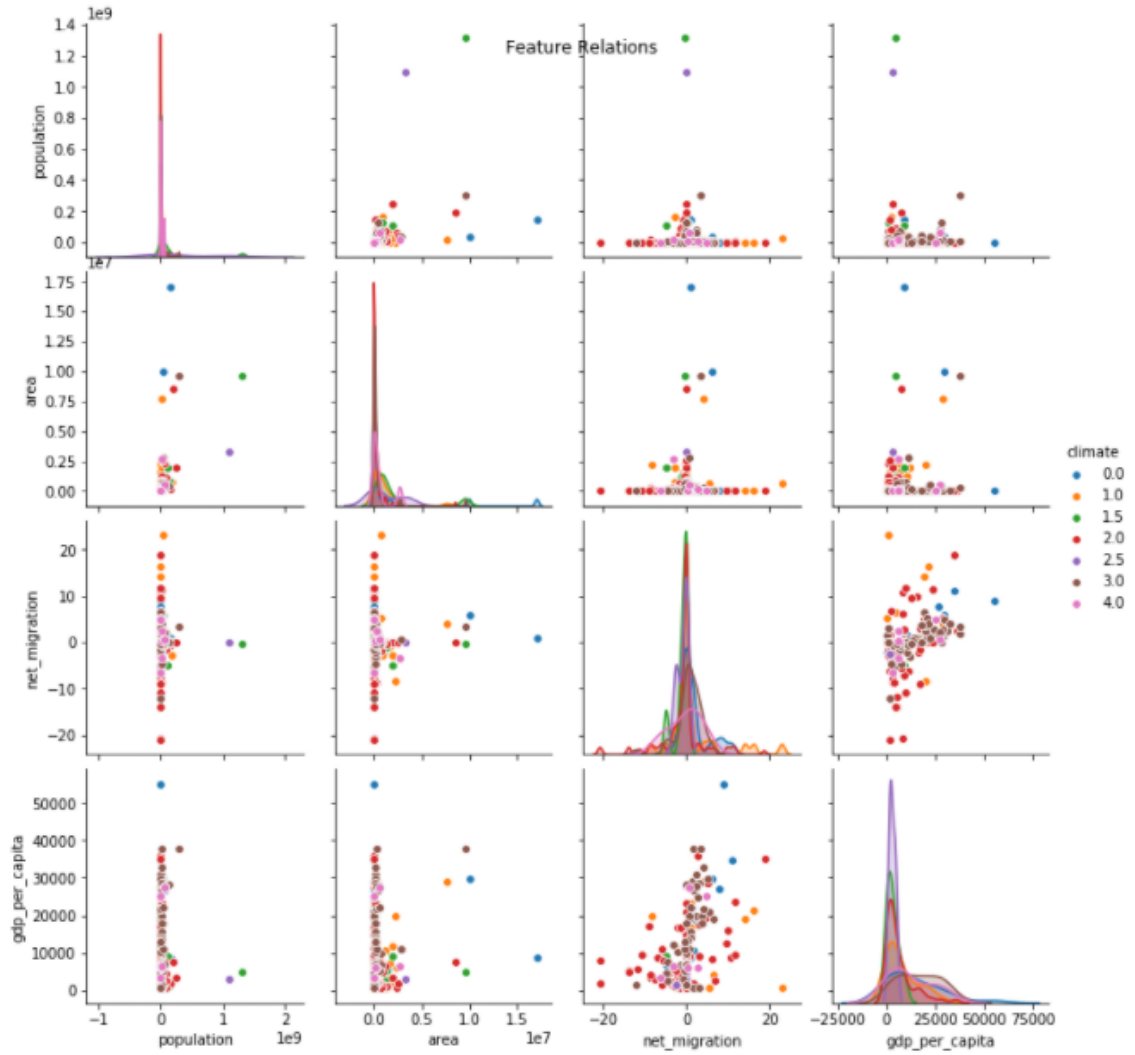


Some insights from the above correlation heatmap:

1. expected strong correlation between infant_mortality and birthrate.
2. unexpected strong correlation between infant_mortality and agriculture.
3. expected strong correlation between infant_mortality and literacy.
4. expected strong correlation between gdp_per_capita and phones.
5. expected strong correlation between arable and other (other than crops).
6. expected strong correlation between birthrate and literacy (the less literacy the higher the birthrate).
7. unexpected strong correlation between birthrate and phones.

Graph Set 3

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

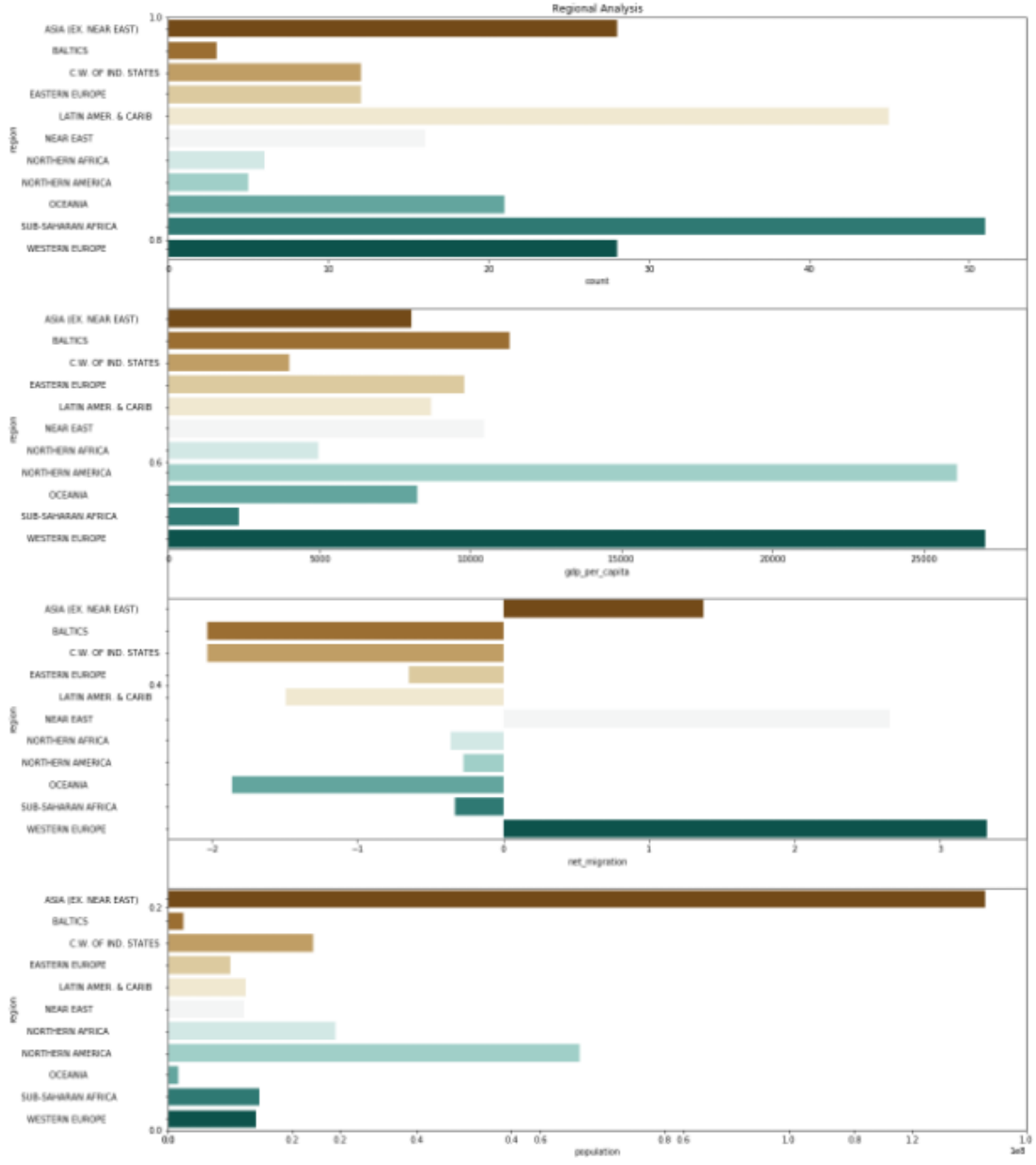


We can see a fair correlation between GDP and migration, which makes sense, since migrants tend to move to countries with better opportunities and higher GDP per capita.

Graph Set 4

Regional Analysis

```
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:

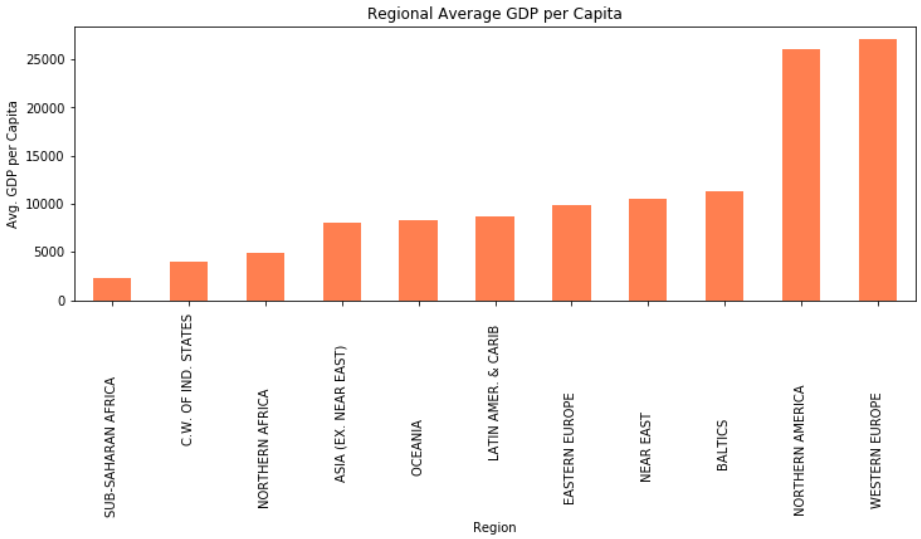
1. Sub-Saharian Africa and Latin America regions have the most countries within them.
2. Western Europe and North America have the highest GDP per capita, while Sub-Saharian Africa has the lowest GDP per capita.
3. Asia, North America, and North Europe, are the main regions where migrants from other regions go.
4. Asia has the largest population; Oceania has the smallest.

Graph Set 5

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.

```
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()
```

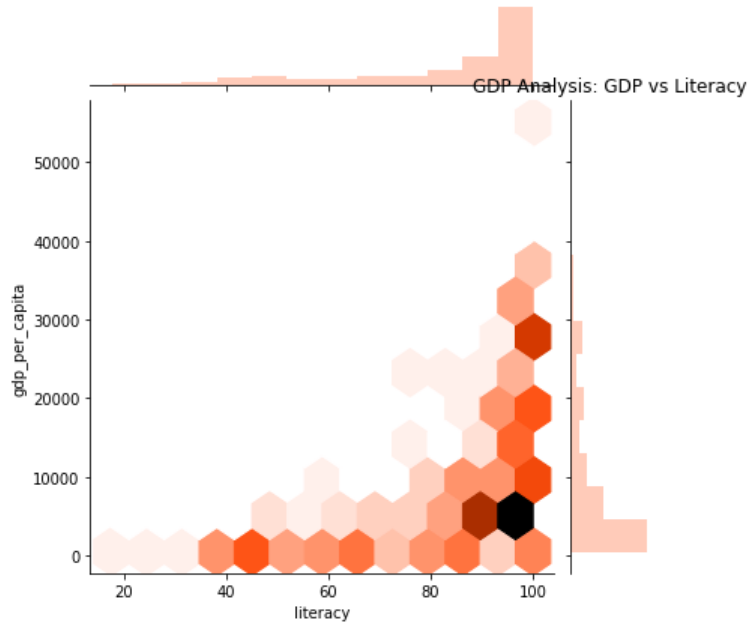


The figure below shows the regional ranking according to the average GDP per capita. As expected, North America and Western Europe have the highest GDP per capita, while Sub Saharian Africa has the lowest, and that may describe the large migration trends in the world in the past decade.

Graph Set 6

```
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>
```



From the above figure, it is clear that the higher the country's GDP, the more literate the population is, and vice-versa.

Modelling

The purpose of these models will be to get effective insight into the following:

1. If GDP of country depending on various factors:
    - This insight can be used for Market Targeting.
2. Get insight into how changing RMSE of the predictions affect:
    - Spending more money to target the researchers/political legends that are most likely to retain innovations and on strict following of laws or spending less money on education and basic health-care infrastructure or reasons responsible for GDP.

Where to Use RF Regression: RF Regression Example

Let's say you want to estimate the average household income in your town. You could easily find an estimate using the Random Forest Algorithm. You would start off by distributing surveys asking people to answer a number of different questions. Depending on how they answered these questions, an estimated household income would be generated for each person.

After you've found the decision trees of multiple people you can apply the Random Forest Algorithm to this data. You would look at the results of each decision tree and use random forest to find an average income between all of the decision trees. Applying this algorithm would provide you with an accurate estimate of the average household income of the people you surveyed.

- Math behind the metrics

MAE (Mean absolute error) represents the difference between the original and predicted values extracted by averaged the absolute difference over the data set.

RMSE (Root Mean Squared Error) is the error rate by the square root of MSE.

R-squared (Coefficient of determination) represents the coefficient of how well the values fit compared to the original values. The value from 0 to 1 interpreted as percentages. The higher the value is, the better the model is.

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}|$$

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$$

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

Where,

$\hat{y}$ – *predicted value of y*

$\bar{y}$ – *mean value of y*

Linear regression equation

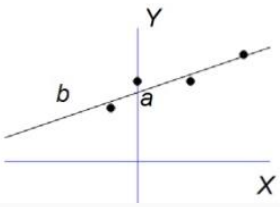
(without error)

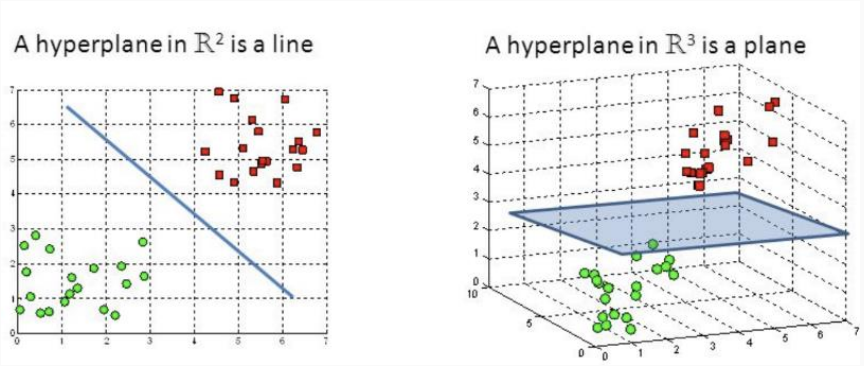
$$\hat{Y} = bX + a$$

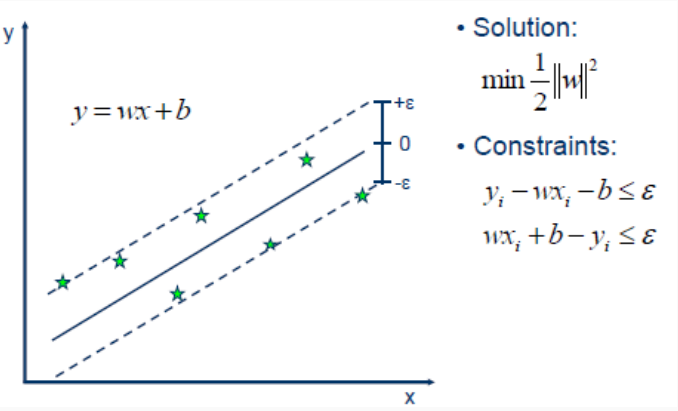
predicted values of *Y*

b = slope = rate of predicted ↑/↓ for Y scores for each unit increase in *X*

Y-intercept = level of Y when X is 0







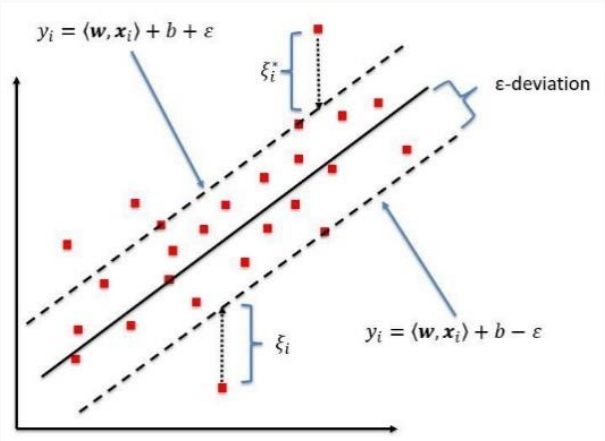
- Solution:

$$\min \frac{1}{2}\|w\|^2$$

- Constraints:

$$y_i - wx_i - b \le \varepsilon$$

$$wx_i + b - y_i \le \varepsilon$$



Algorithm for Random Forest Regression:

1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

GridSearchCV Vs RandomSearchCV:

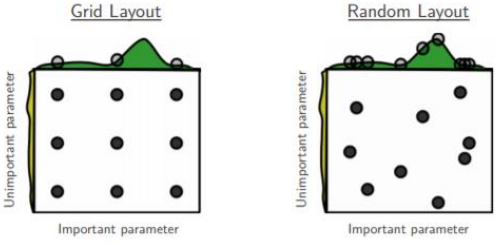


Figure 1: Grid and random search of nine trials for optimizing a function $f(x,y) = g(x) + h(y) \approx g(x)$ with low effective dimensionality. Above each square $g(x)$ is shown in green, and left of each square $h(y)$ is shown in yellow. With grid search, nine trials only test $g(x)$ in three distinct places. With random search, all nine trials explore distinct values of g. This failure of grid search is the rule rather than the exception in high dimensional hyper-parameter optimization.

Algorithm for Gradient Boosting Regression:

Algorithm 1 Friedman's Gradient Boost algorithm

Inputs:

- input data $(x, y)_{i=1}^{N}$
- number of iterations M
- choice of the loss-function $\Psi(y, f)$
- choice of the base-learner model $h(x, \theta)$

Algorithm:

1: initialize $\widehat{f}_0$ with a constant
2: **for** $t = 1$ to M **do**
3: compute the negative gradient $g_t(x)$
4: fit a new base-learner function $h(x, \theta_t)$
5: find the best gradient descent step-size ρ_t:

$$\rho_t = \arg\min_{\rho} \sum_{i=1}^{N} \Psi\left[y_i, \widehat{f}_{t-1}(x_i) + \rho h(x_i, \theta_t)\right]$$

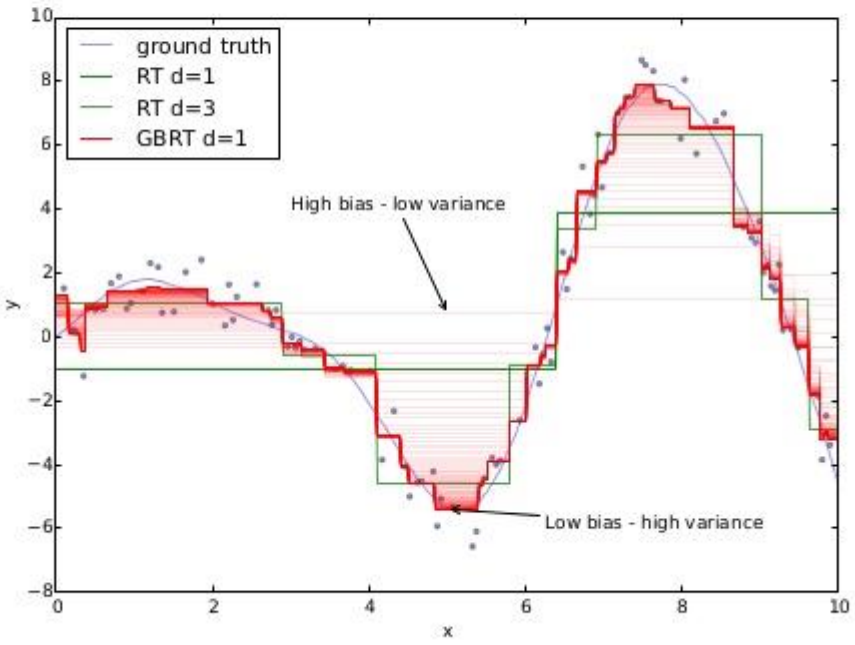
6: update the function estimate:

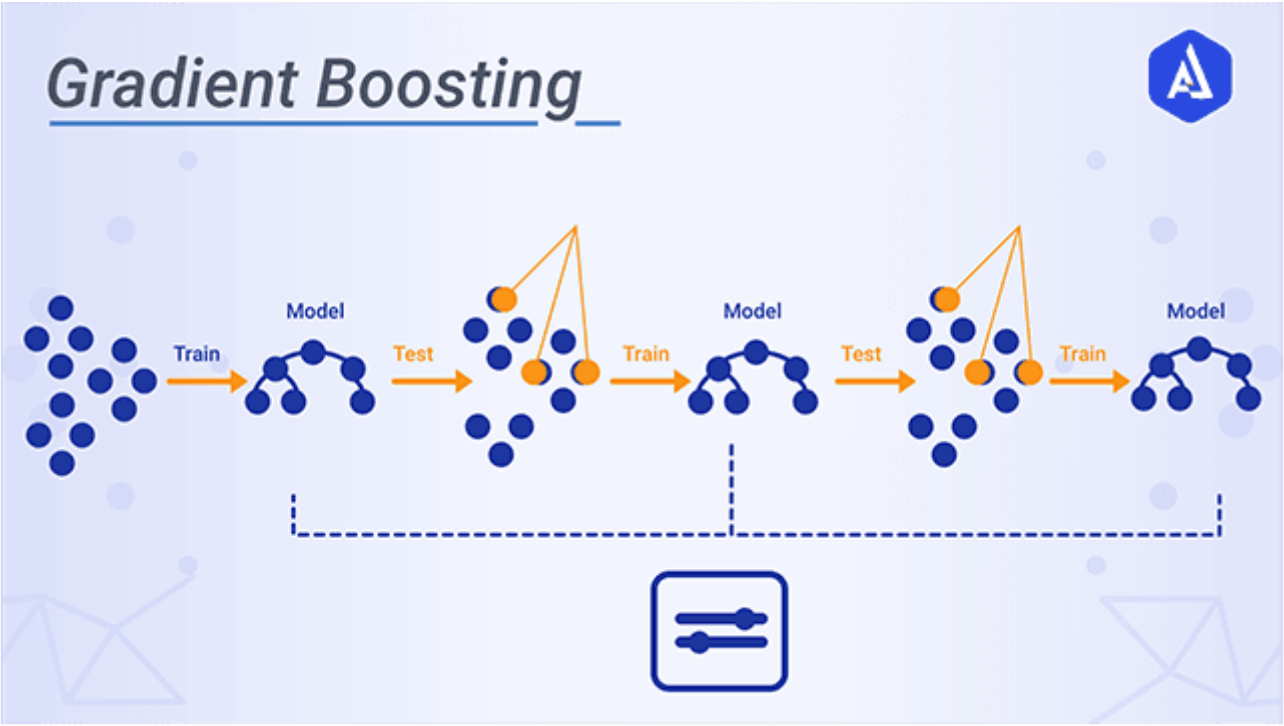
$$\widehat{f}_t \leftarrow \widehat{f}_{t-1} + \rho_t h(x, \theta_t)$$

7: **end for**

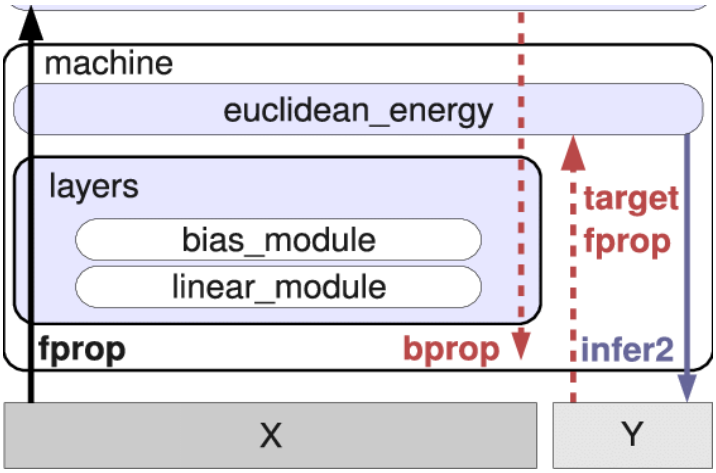
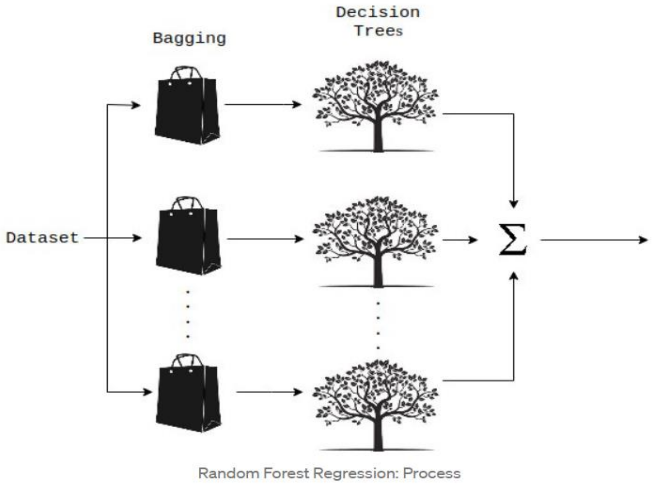
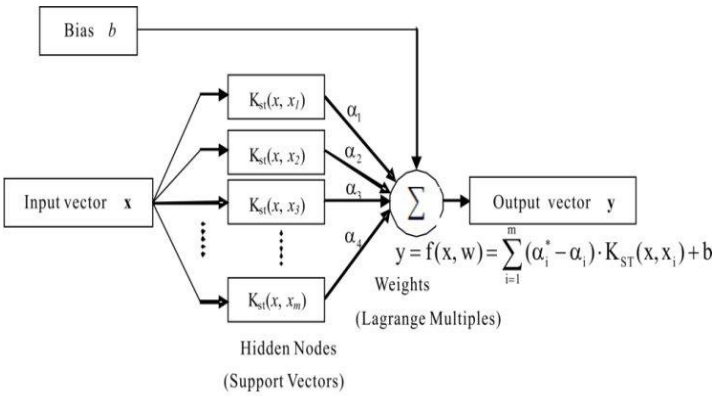
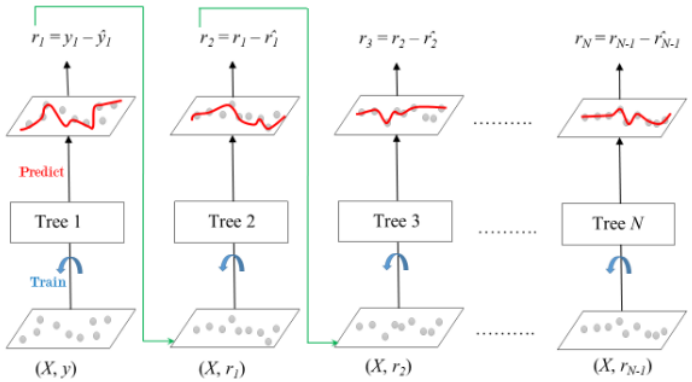
Example

```
from sklearn.ensemble import GradientBoostingRegressor
est = GradientBoostingRegressor(n_estimators=2000, max_depth=1).fit(X, y)
for pred in est.staged_predict(X):
    plt.plot(X[:, 0], pred, color='r', alpha=0.1)
```





- Model Architecture Process Through Visualization

Linear Regression	Random Forest Regression
	
Support Vector Regression	**Gradient Boosting Regression**
	

Random Forest Regression: Process

- Quick Notes

Step 1: Imported required libraries.
Step 2: Read the Data.
Step 3: Analysed the data.
Step 4: Performed Data Cleaning.
Step 5: Performed EDA.
Step 6: Performed Data Pre-conditioning.
Step 7: Performed Model Building, Prediction, Evaluation and Optimization.

- Linear Regression
- SVM
- Random Forest
- Gradient Boosting

Step 8: Created Web App.

- The Model Analysis

Imported required libraries – When we import modules, we're able to call functions that are not built into Python. Some modules are installed as part of Python, and some we will install through pip. Making use of modules allows us to make our programs more robust and powerful as we're leveraging existing code.

Read the Data – You can import tabular data from CSV files into pandas data frame by specifying a parameter value for the file.

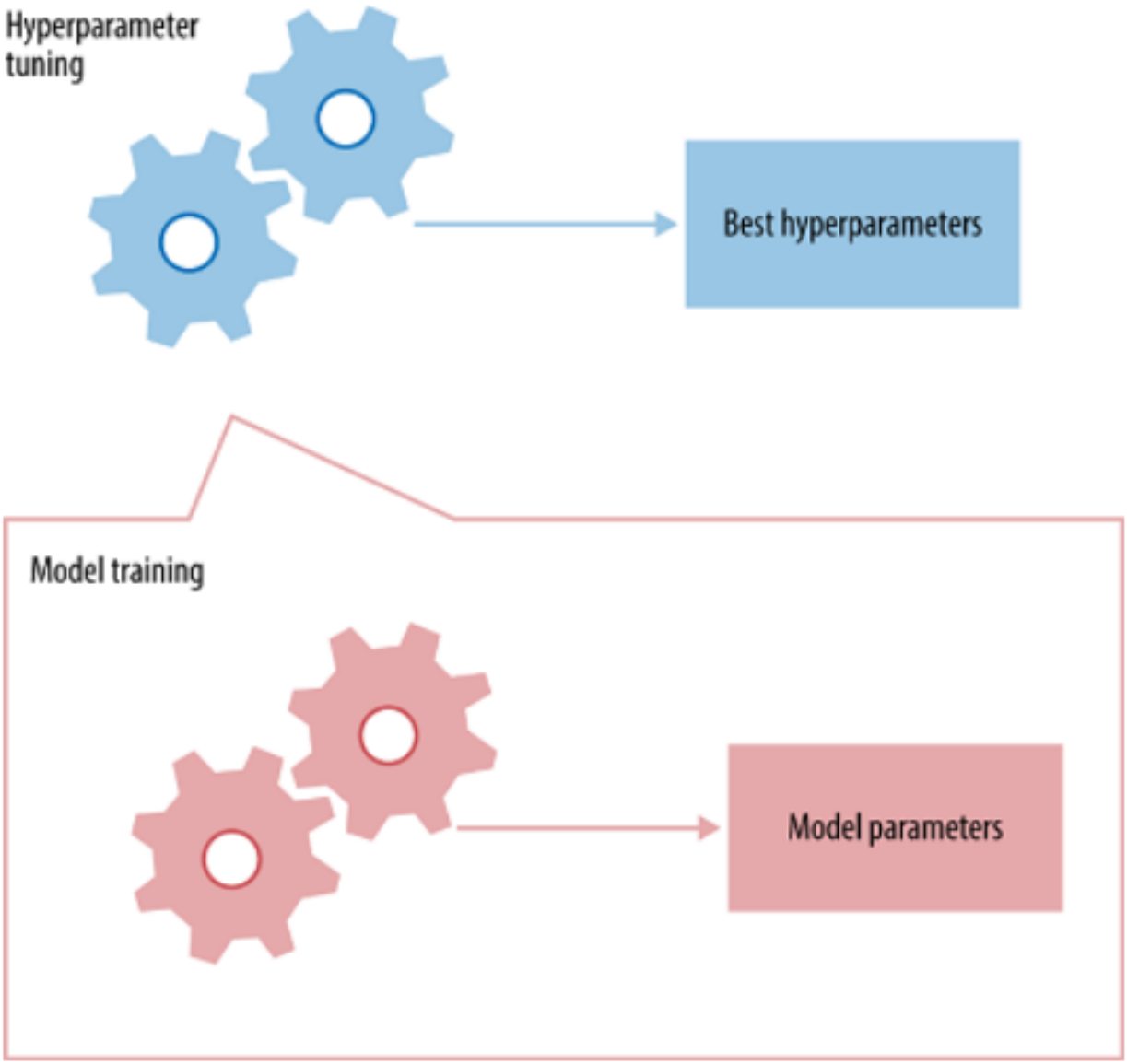
Analysed the data – Using '.info(), .astype(), .isnull().sum(), .unique()' commands and imputing it. Checking unique number of categories in particular column. Checking missing values to deal with them.

Performed Data Cleaning – Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mis-labeled. To perform a Python data cleansing, you can drop the missing values, replace them, replace each NaN with a scalar value, or fill forward or backward. Checked number of missing values in each column and imputed it with either replacing with zero or mean value.

Performed EDA – The primary goal of EDA is to maximize the analyst's insight into a data set and into the underlying structure of a data set, while providing all of the specific items that an analyst would want to extract from a data set, such as: a good-fitting, parsimonious model and a list of outliers. There are many libraries available in python like pandas, NumPy, matplotlib, seaborn to perform EDA. The four types of EDA are univariate non-graphical, multivariate non- graphical, univariate graphical, and multivariate graphical. First, I plotted correlation heatmap. Insights from it is, expected strong correlation between infant_mortality and birthrate, unexpected stronge correlation between birthrate and phones. A correlation heatmap uses colored cells, typically in a monochromatic scale, to show a 2D correlation matrix (table) between two discrete dimensions. Correlation ranges from -1 to +1. Values closer to zero means there is no linear trend between the two variables. The close to 1 the correlation is the more positively correlated they are; that is as one increases so does the other and the closer to 1 the stronger this relationship is. Secondly, I plotted pairplot graph. Insights from it is, 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. The pairplot function creates a grid of Axes such that each variable in data will by shared in the y-axis across a single row and in the x-axis across a single column. To plot multiple pairwise bivariate distributions in a dataset, you can use the pairplot() function. This shows the relationship

for (n,2) combination of variable in a DataFrame as a matrix of plots and the diagonal plots are the univariate plots. Third, I plotted bar-plot on regional analysis. Insights from it is, Western Europe and North America have the highest GDP per capita, while Sub-Saharian Africa has the lowest GDP per capita. A barplot shows shows comparisons among discrete categories. Fourth, I plotted joint-plot. Insights from it is, it is clear that the higher the country's GDP, the more literate the population is and poor countries suffer more from infant mortality. Joint-plot is displays a relationship between 2 variables (bivariate) as well as 1D profiles (univariate) in the margins.
Performed Data Pre-conditioning – Here, I am making the data ready for training. '.get_dummies' will convert your categorical string values into dummy variables. 'pd.get_dummies' create a new data frame containing unique values as columns which consists of zeros and ones. First, I Transform 'region' column into numerical values. Splitting data into train and test set in order to prediction w.r.t X_test. This helps in prediction after training data. Secondly, I 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. Thirdly, performed splits of dataset - with/without feature selection, with/without feature scaling. 'fit_transform()' means to do some calculation and then do transformation (say calculating the means of columns from some data and then replacing the missing values). So, for training set, you need to both calculate and do transformation. 'StandardScalar()' transforms the data in such a manner that it has mean as 0 and standard deviation as 1. In short, it standardizes the data. Standardization is useful for data which has negative values. It arranges the data in a standard normal distribution.
Performed Model Building, Prediction, Evaluation and Optimization – The fit() method takes the training data as arguments, which can be one array in the case of unsupervised learning, or two arrays in the case of supervised learning. Predicted test data w.r.t. X_test. It helps in better analysing performance of model. Now, performed model evaluation by using MAE, RMSE and R2 Score Error Measures. It indicates how better model is performing. Optimization algorithm is a procedure which is executed iteratively by comparing various solutions till an optimum or a satisfactory solution is found. You can either use GridSearchCV or RandomizedSearchCV method. The only difference between both the approaches is in grid search we define the combinations and do training of the model whereas in RandomizedSearchCV the model selects the combinations randomly. GridSearchCV helps to loop through predefined hyperparameters and fit your estimator (model) on your training set. So, in the end, you can select the best parameters from the listed hyperparameters. RandomizedSearchCV is a technique where random combinations of the hyperparameters are used to find the best solution for the built model. It is similar to grid search, and yet it has proven to yield better results comparatively. Gradient boost identifies difficult observations by large residuals computed in the previous iterations. In Gradientboost "shortcomings" are identified by gradients. Random Forest works well with both categorical and continuous values. RF provides accurate results most of the time. Scikit-Learn implements a set of sensible default hyperparameters for all models, but these are not guaranteed to be optimal for a problem. The best hyperparameters are usually impossible to determine ahead of time, and tuning a model is where machine learning turns from a science into trial-and-error based engineering. Hyperparameter tuning relies more on experimental results than theory, and thus the best method to determine the optimal settings is to try many different combinations evaluate the performance of each model. However, evaluating each model only on the training set can lead to one of the most fundamental problems in machine learning: overfitting. If we optimize the model for the training data, then our model will score very well on the training set, but will not be able to generalize to new data, such as in a test set. When a model performs highly on the training set but poorly on the

test set, this is known as overfitting, or essentially creating a model that knows the training set very well but cannot be applied to new problems. It's like a student who has memorized the simple problems in the textbook but has no idea how to apply concepts in the messy real world. An overfit model may look impressive on the training set, but will be useless in a real application. Therefore, the standard procedure for hyperparameter optimization accounts for overfitting through cross validation. To evaluate the prediction error rates and model performance in regression analysis. Once you have obtained your error metric/s, take note of which X's have minimal impacts on y. Removing some of these features may result in an increased accuracy of your model. RMSE: Most popular metric, similar to MSE, however, the result is square rooted to make it more interpretable as it's in base units. It is recommended that RMSE be used as the primary metric to interpret your model. All of them require two lists as parameters, with one being your predicted values and the other being the true values. MAE: The easiest to understand. Represents average error.



Hyperparameters and Parameters

- Linear Regression – Base Model

1)Model Training

Model Training

```
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)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

2)Predictions

Predictions

```
lm1_pred = lm1.predict(X_test)
lm2_pred = lm2.predict(X2_test)
lm3_pred = lm3.predict(X3_test)
lm4_pred = lm4.predict(X4_test)
```

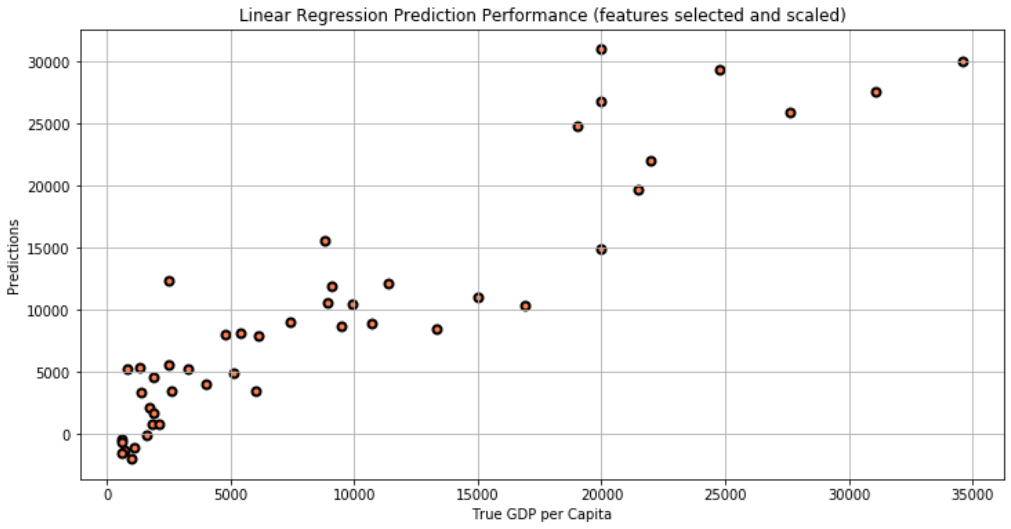
3)Model Evaluation

Evaluation

```
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: 330350.858660171
RMSE: 1570337.5456391599
R2_Score:  -29843.12038335633

all features, with scaling:
MAE: 569019.4687589019
RMSE: 1283170.8219653955
R2_Score:  -19925.990118467882

selected features, No scaling:
MAE: 2965.935722939867
RMSE: 4088.79458024794
R2_Score:  0.7976685756859008

selected features, with scaling:
MAE: 2879.5213243944418
RMSE: 3756.4365885029674
R2_Score:  0.8292247702712089
```

4)Model Visualization



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.

- SVM - First Model

1)Model Training

Model Training

```
svm1 = SVR(kernel='rbf')
svm1.fit(X_train,y_train)

svm2 = SVR(kernel='rbf')
svm2.fit(X2_train,y2_train)

svm3 = SVR(kernel='rbf')
svm3.fit(X3_train,y3_train)

svm4 = SVR(kernel='rbf')
svm4.fit(X4_train,y4_train)
```

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

2)Predictions

Predictions

```
svm1_pred = svm1.predict(X_test)
svm2_pred = svm2.predict(X2_test)
svm3_pred = svm3.predict(X3_test)
svm4_pred = svm4.predict(X4_test)
```

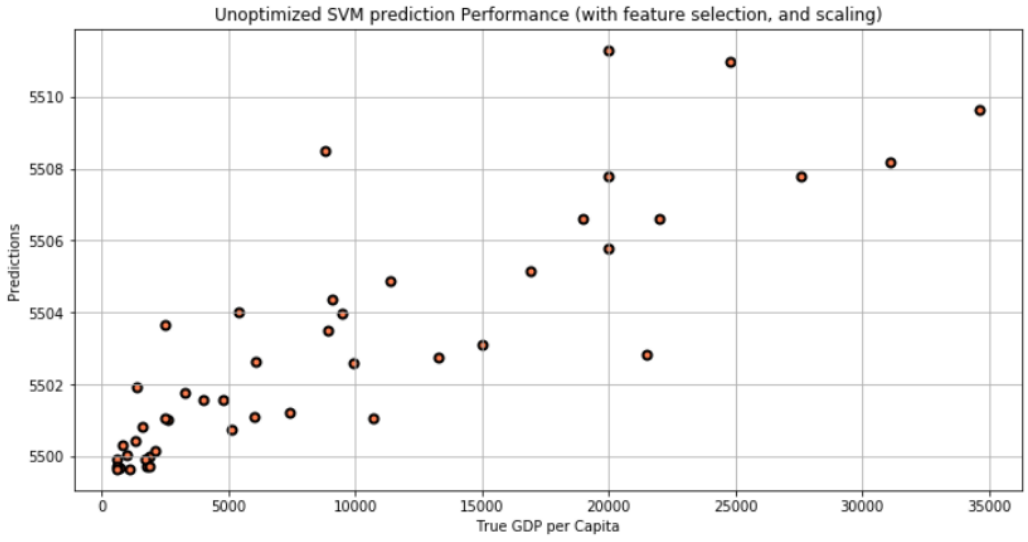
3) Model Evaluation

Evaluation

```
print('SVM Performance:')

print('\nall features, No scaling:')
print('MAE:', metrics.mean_absolute_error(y_test, svm1_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, svm1_pred)))
print('R2_Score: ', metrics.r2_score(y_test, svm1_pred))

print('\nall features, with scaling:')
print('MAE:', metrics.mean_absolute_error(y2_test, svm2_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y2_test, svm2_pred)))
print('R2_Score: ', metrics.r2_score(y2_test, svm2_pred))

print('\nselected features, No scaling:')
print('MAE:', metrics.mean_absolute_error(y3_test, svm3_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y3_test, svm3_pred)))
print('R2_Score: ', metrics.r2_score(y3_test, svm3_pred))

print('\nselected features, with scaling:')
print('MAE:', metrics.mean_absolute_error(y4_test, svm4_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y4_test, svm4_pred)))
print('R2_Score: ', metrics.r2_score(y4_test, svm4_pred))

fig = plt.figure(figsize=(12, 6))
plt.scatter(y3_test,svm3_pred,color='coral', linewidths=2, edgecolors='k')
plt.xlabel('True GDP per Capita')
plt.ylabel('Predictions')
plt.title('Unoptimized SVM prediction Performance (with feature selection, and scaling)')
plt.grid()
plt.show()
```

```
SVM Performance:

all features, No scaling:
MAE: 7049.984895264721
RMSE: 9811.73631340298
R2_Score:  -0.16510345624387246

all features, with scaling:
MAE: 7042.737596769212
RMSE: 9800.406046613498
R2_Score:  -0.16241416444556656

selected features, No scaling:
MAE: 7047.711927073501
RMSE: 9807.997922107874
R2_Score:  -0.16421578810668724

selected features, with scaling:
MAE: 7040.043820847137
RMSE: 9794.58886537642
R2_Score:  -0.1610346364957338
```

4)Model Visualization



Feature scaling, and feature selection, made almost no difference in the prediction performance of the SVM algorithm.

The results of SVM is worse than that of Linear Regression, so we will try to improve SVM's performance by optimizing its parameters using grid search.

5)Model Optimization

Optimizing SVM

In [55]:
```
param_grid = {'C': [1, 10, 100], 'gamma': [0.01,0.001,0.0001], 'kernel': ['rbf']}
grid = GridSearchCV(SVR(),param_grid,refit=True,verbose=3)
```

In [56]:
```
grid.fit(X4_train,y4_train)
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV] C=1, gamma=0.01, kernel=rbf .....................................
[CV] ........ C=1, gamma=0.01, kernel=rbf, score=-0.324, total=   0.0s
[CV] C=1, gamma=0.01, kernel=rbf .....................................
[CV] ........ C=1, gamma=0.01, kernel=rbf, score=-0.156, total=   0.0s
[CV] C=1, gamma=0.01, kernel=rbf .....................................
[CV] ........ C=1, gamma=0.01, kernel=rbf, score=-0.115, total=   0.0s
[CV] C=1, gamma=0.01, kernel=rbf .....................................
[CV] ........ C=1, gamma=0.01, kernel=rbf, score=-0.372, total=   0.0s
[CV] C=1, gamma=0.01, kernel=rbf .....................................
[CV] ........ C=1, gamma=0.01, kernel=rbf, score=-0.026, total=   0.0s
[CV] C=1, gamma=0.001, kernel=rbf ....................................
[CV] ....... C=1, gamma=0.001, kernel=rbf, score=-0.325, total=   0.0s
[CV] C=1, gamma=0.001, kernel=rbf ....................................
[CV] ....... C=1, gamma=0.001, kernel=rbf, score=-0.158, total=   0.0s
[CV] C=1, gamma=0.001, kernel=rbf ....................................
[CV] ....... C=1, gamma=0.001, kernel=rbf, score=-0.117, total=   0.0s
[CV] C=1, gamma=0.001, kernel=rbf ....................................
[CV] ....... C=1, gamma=0.001, kernel=rbf, score=-0.373, total=   0.0s
[CV] C=1, gamma=0.001, kernel=rbf ....................................
```

In [57]:
```
grid.best_params_
```

Out[57]:
```
{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
```

In [58]:
```
grid.best_estimator_
```

Out[58]:
```
SVR(C=100, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma=0.01,
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

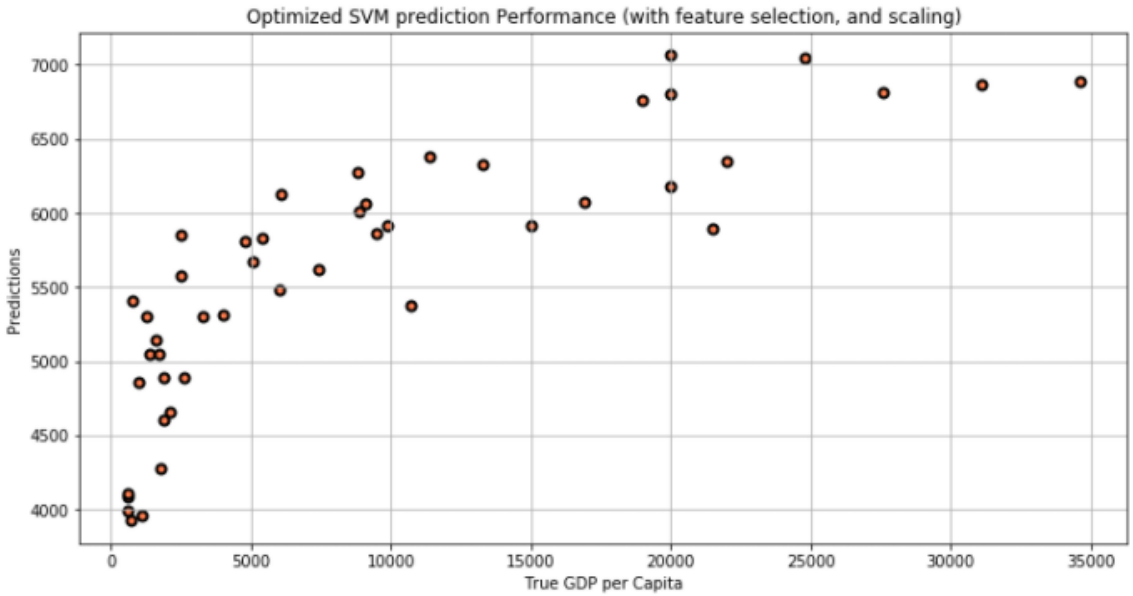
In [61]:
```
grid_predictions = grid.predict(X4_test)
```

6)Model Prediction, Evaluation and Visualization after optimization

```
grid_predictions = grid.predict(X4_test)
```

```
print('MAE:', metrics.mean_absolute_error(y4_test, grid_predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y4_test, grid_predictions)))
print('R2_Score: ', metrics.r2_score(y4_test, grid_predictions))

fig = plt.figure(figsize=(12, 6))
plt.scatter(y4_test,grid_predictions,color='coral', linewidths=2, edgecolors='k')
plt.xlabel('True GDP per Capita')
plt.ylabel('Predictions')
plt.title('Optimized SVM prediction Performance (with feature selection, and scaling)')
plt.grid()
plt.show()
```

```
MAE: 6386.413128432553
RMSE: 9133.499345710767
R2_Score:  -0.009594923559210988
```



SVM has improved a little with grid search, but it still performs below linear regression.

- Random Forest - Second Model

1)Model Training

Training

```
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)
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=200, n_jobs=None, oob_score=False,
                      random_state=101, verbose=0, warm_start=False)
```

2)Predictions

Prediction

```
rf1_pred = rf1.predict(X_test)
rf3_pred = rf3.predict(X3_test)
```

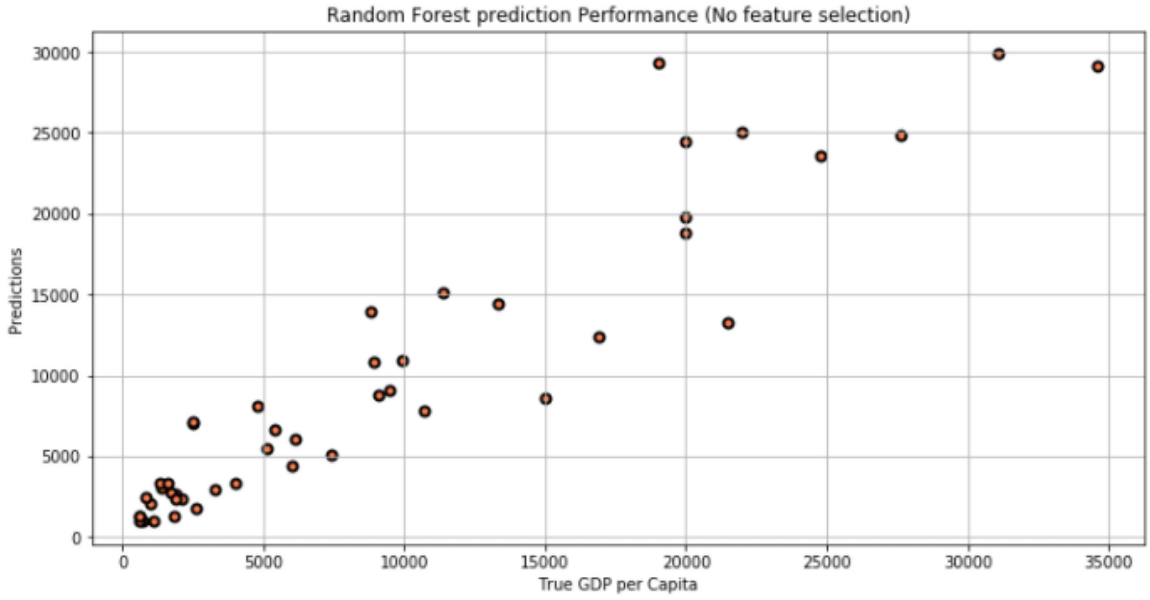
3)Model Evaluation

Evaluation

```
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
```

4)Model Visualization



5) Model Optimization

Optimization

We will use grid search in order to obtain good parameters for our RF regressor. Of course our optimization here will be limited due to time and computing power constraints. The parameters we will optimiz are:

- n-estimators
- min_samples_leaf
- max_features
- bootstrap

```
rf_param_grid = {'max_features': ['sqrt', 'auto'],
              'min_samples_leaf': [1, 3, 5],
              'n_estimators': [100, 500, 1000],
             'bootstrap': [False, True]}
```

```
rf_grid = GridSearchCV(estimator= RandomForestRegressor(), param_grid = rf_param_grid,  n_jobs=-1, verbose=0)
```

```
rf_grid.fit(X_train,y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=RandomForestRegressor(bootstrap=True, ccp_alpha=0.0,
                                             criterion='mse', max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             max_samples=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators=100, n_jobs=None,
                                             oob_score=False, random_state=None,
                                             verbose=0, warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'bootstrap': [False, True],
                         'max_features': ['sqrt', 'auto'],
                         'min_samples_leaf': [1, 3, 5],
                         'n_estimators': [100, 500, 1000]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
rf_grid.best_params_
```

```
{'bootstrap': False,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'n_estimators': 500}
```

```
rf_grid.best_estimator_
```

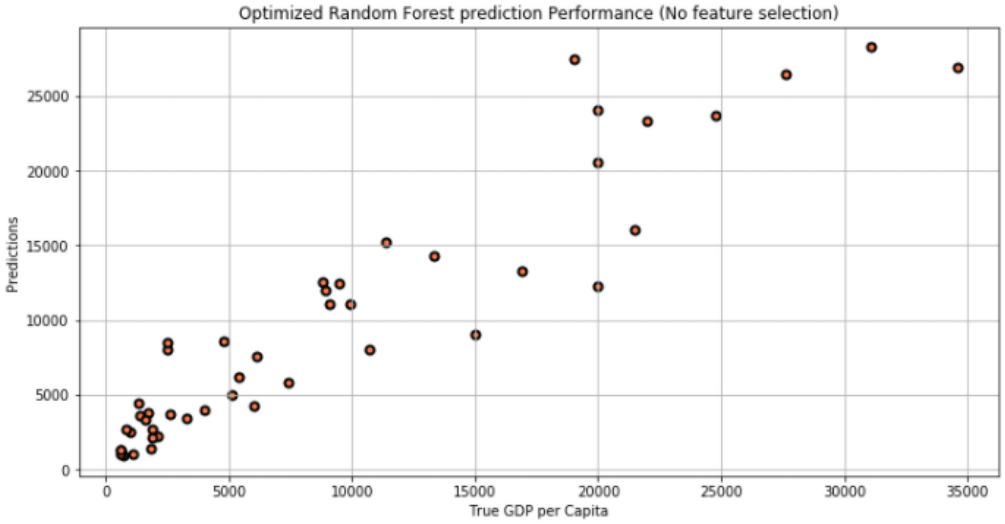
```
RandomForestRegressor(bootstrap=False, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='sqrt', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=500, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

6)Model Prediction, Evaluation and Visualization after optimization

```
rf_grid_predictions = rf_grid.predict(X_test)
```

```
print('MAE:', metrics.mean_absolute_error(y_test, rf_grid_predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, rf_grid_predictions)))
print('R2_Score: ', metrics.r2_score(y_test, rf_grid_predictions))
fig = plt.figure(figsize=(12, 6))
plt.scatter(y_test,rf_grid_predictions,color='coral', linewidths=2, edgecolors='k')
plt.xlabel('True GDP per Capita')
plt.ylabel('Predictions')
plt.title('Optimized Random Forest prediction Performance (No feature selection)')
plt.grid()
plt.show()
```

```
MAE: 2360.747826086956
RMSE: 3219.9245794569947
R2_Score:  0.8745229924626883
```



We can see that the optimization process on RF regressor has not changed the performance in a noticebale manner, yet the slight change was actually to the worst, that is probably because our initial parameters were already very close to the optimum ones.

- Gradient Boosting – Third Model

1)Model Training

Training

We will first traing the GBM regressor with the default parameter values, then we will try optimizing its parameters.

```
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)
```

```
GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                          init=None, learning_rate=0.1, loss='ls', max_depth=3,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=None,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='deprecated',
                          random_state=101, subsample=1.0, tol=0.0001,
                          validation_fraction=0.1, verbose=0, warm_start=False)
```

2)Predictions

Prediction

```
gbm1_pred = gbm1.predict(X_test)
gbm3_pred = gbm3.predict(X3_test)
```

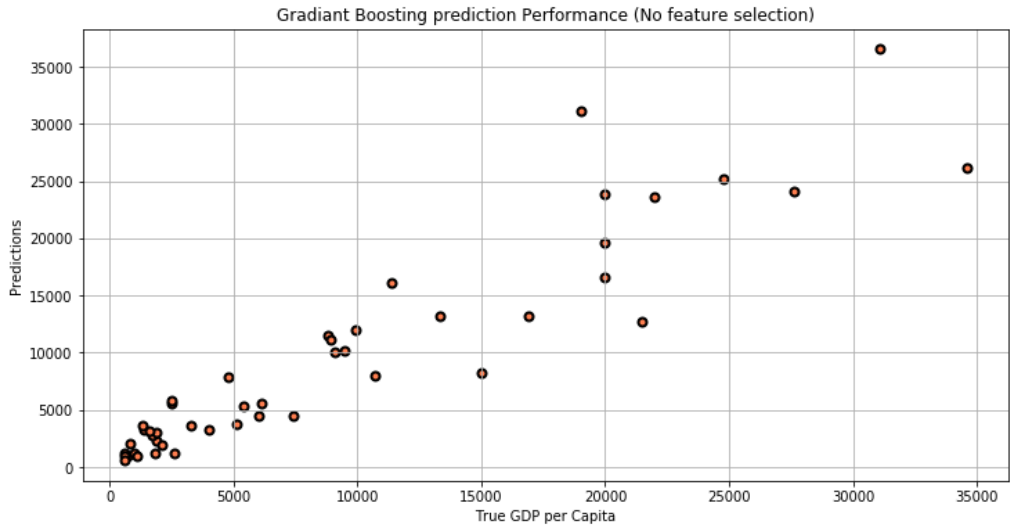
3)Model Evaluation

Evaluation

```
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
```

4)Model Visualization



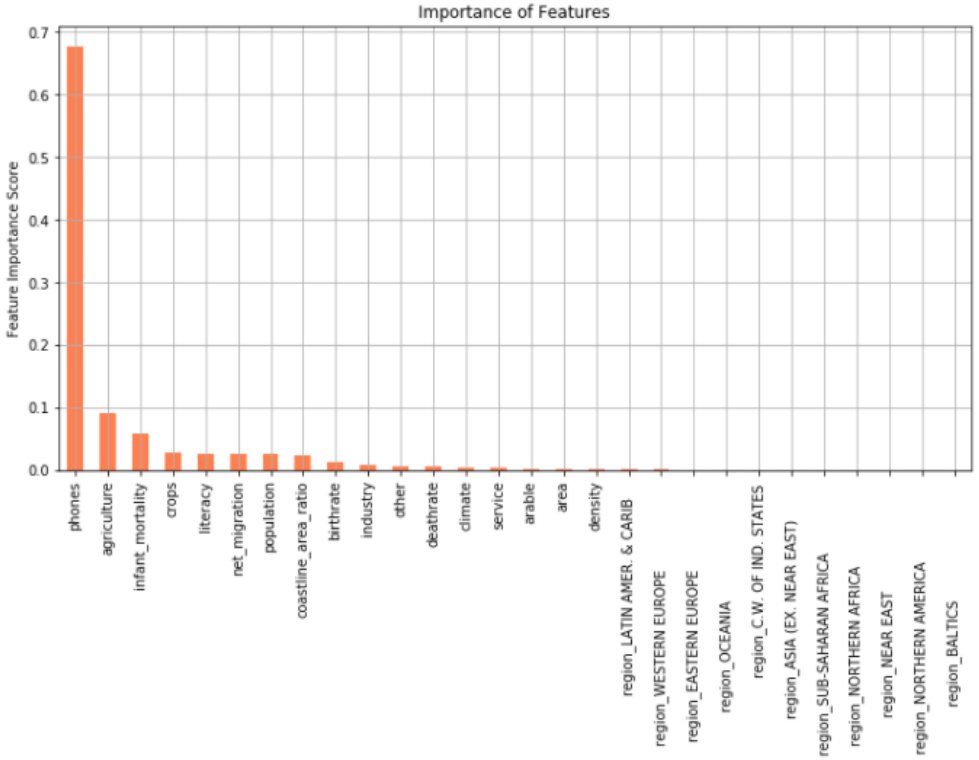
It is clear that Gradiant Boosting gave us good performance even before optimization. Its performance on our dataset is very close to that of Random Forest.

5)Feature Importance

Feature Importance

We can plot how the GBM regressor sees the importance of different features in the dataset.

```
feat_imp = pd.Series(gbm1.feature_importances_, list(X_train)).sort_values(ascending=False)
fig = plt.figure(figsize=(12, 6))
feat_imp.plot(kind='bar', title='Importance of Features', color= 'coral')
plt.ylabel('Feature Importance Score')
plt.grid()
plt.show()
```



The number of phones seems to have the highest predictive power.

With this first model, we obtain an R2_Score of 0.8590, which is not very far behind that of Random Forest regressor. Next we will try to optimize GBM, and compare its performance to that of Random Forest and the one above. Also, we will plot the feature importance chart, and observe if GBM changed the features importance score after optimization.

6)Model Optimization

GBM Optimization

We will use grid search in order to obtain good parameters for our GBM regressor. Of course our optimization here will be limited due to time and computing power constraints. The parameters we will optimiz are:

- n-estimators: 100, 500, 1000
- learning_rate: 0.001, 0.01, 0.1, 1
- max_depth: 3, 5, 8
- subsample: 0.7, 1 (Values lower than 1 generally lead to a reduction of variance and an increase in bias)
- min_samples_leaf: 1, 20
- min_samples_split: 0.5-1% of our data --> we have 227 datapoints --> 10 -20
- max_features: 4, 7 (sqrt of number of features is a good guess)

```
gbm_param_grid = {'learning_rate':[1,0.1, 0.01, 0.001],
          'n_estimators':[100, 500, 1000],
         'max_depth':[3, 5, 8],
         'subsample':[0.7, 1],
         'min_samples_leaf':[1, 20],
         'min_samples_split':[10, 20],
         'max_features':[4, 7]}

gbm_tuning = GridSearchCV(estimator =GradientBoostingRegressor(random_state=101),
                          param_grid = gbm_param_grid,
                          n_jobs=-1,
                          cv=5)

gbm_tuning.fit(X_train,y_train)
print(gbm_tuning.best_params_)
```

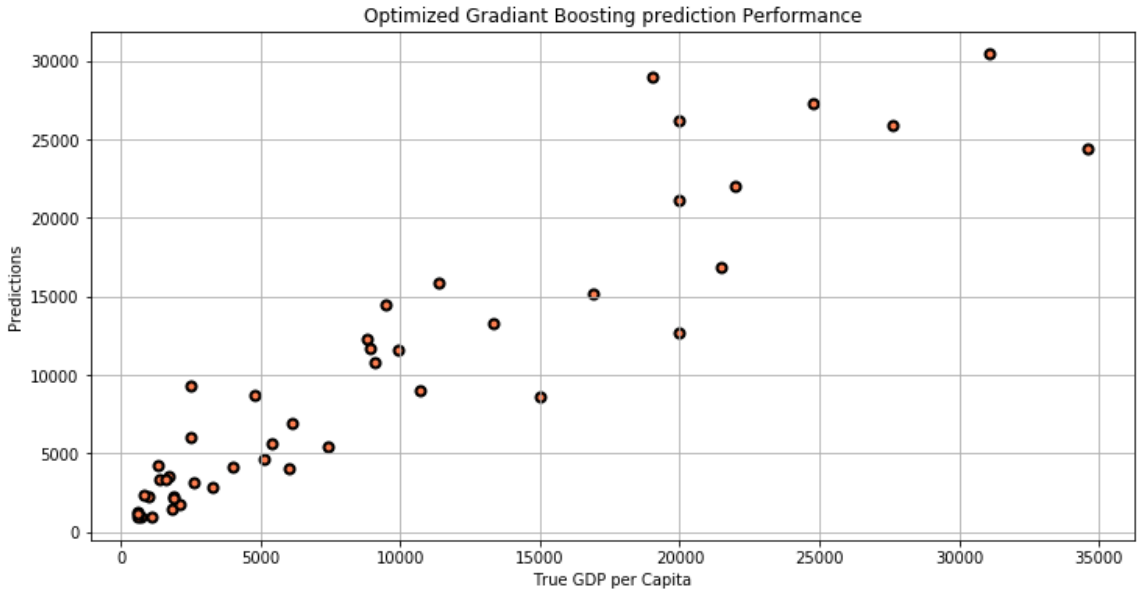
```
{'learning_rate': 0.01, 'max_depth': 5, 'max_features': 7, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 500,
'subsample': 0.7}
```

7)Model Prediction, Evaluation and Visualization after optimization

```
gbm_grid_predictions = gbm_tuning.predict(X_test)
```

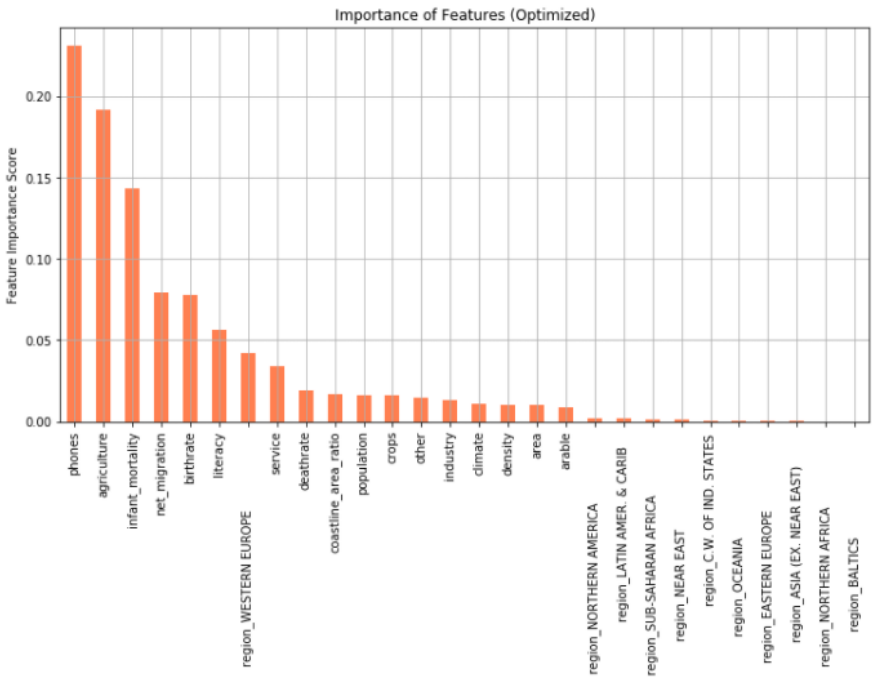
```
print('MAE:', metrics.mean_absolute_error(y_test, gbm_grid_predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, gbm_grid_predictions)))
print('R2_Score: ', metrics.r2_score(y_test, gbm_grid_predictions))
fig = plt.figure(figsize=(12, 6))
plt.scatter(y_test,gbm_grid_predictions,color='coral', linewidths=2, edgecolors='k')
plt.xlabel('True GDP per Capita')
plt.ylabel('Predictions')
plt.title('Optimized Gradiant Boosting prediction Performance')
plt.grid()
plt.show()
```

```
MAE: 2362.9354066125907
RMSE: 3469.360881371261
R2_Score:  0.8543294671599899
```



8)Feature Importance

```
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)
gbm_opt.fit(X_train,y_train)
feat_imp2 = pd.Series(gbm_opt.feature_importances_, list(X_train)).sort_values(ascending=False)
fig = plt.figure(figsize=(12, 6))
feat_imp2.plot(kind='bar', title='Importance of Features (Optimized)', color= 'coral')
plt.ylabel('Feature Importance Score')
plt.grid()
plt.show()
```



We can see that the grid search actually decreased the GBM performance a bit, the reason is that we could not extend the grid search limits, due to processing limits. Yet, we can notice that grid search resulted in a different features importance scores. In general, we can say that GBM has a similar performance to that of Random Forest on our dataset.

Created Web App – Built web app in streamlit for end-users.

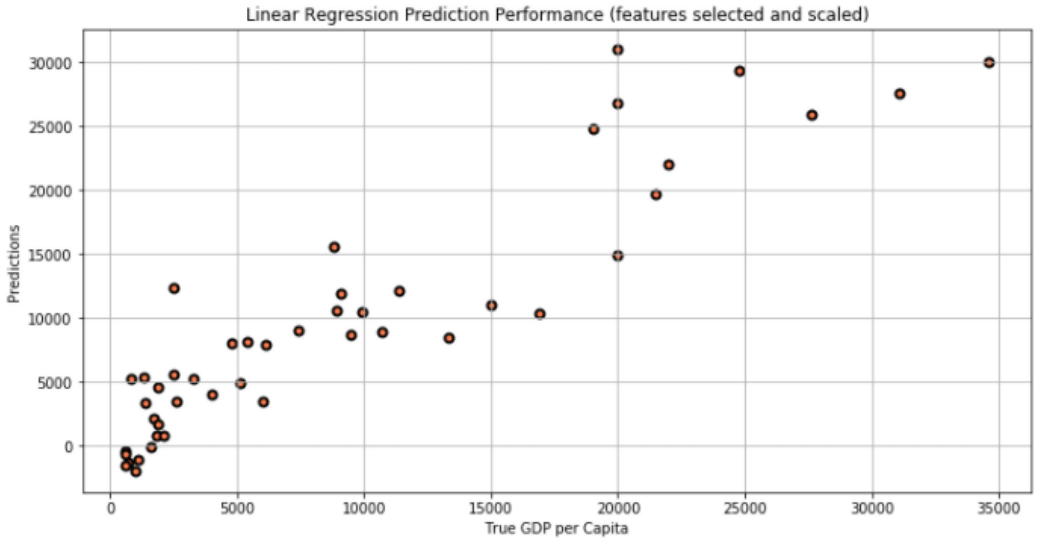
Challenge that I faced in this project is, performance of model was not upto the mark. So, I tried various models to find the best one. One can also use RandomizedSearchCV method to get useful results.

Overall Model Analysis:

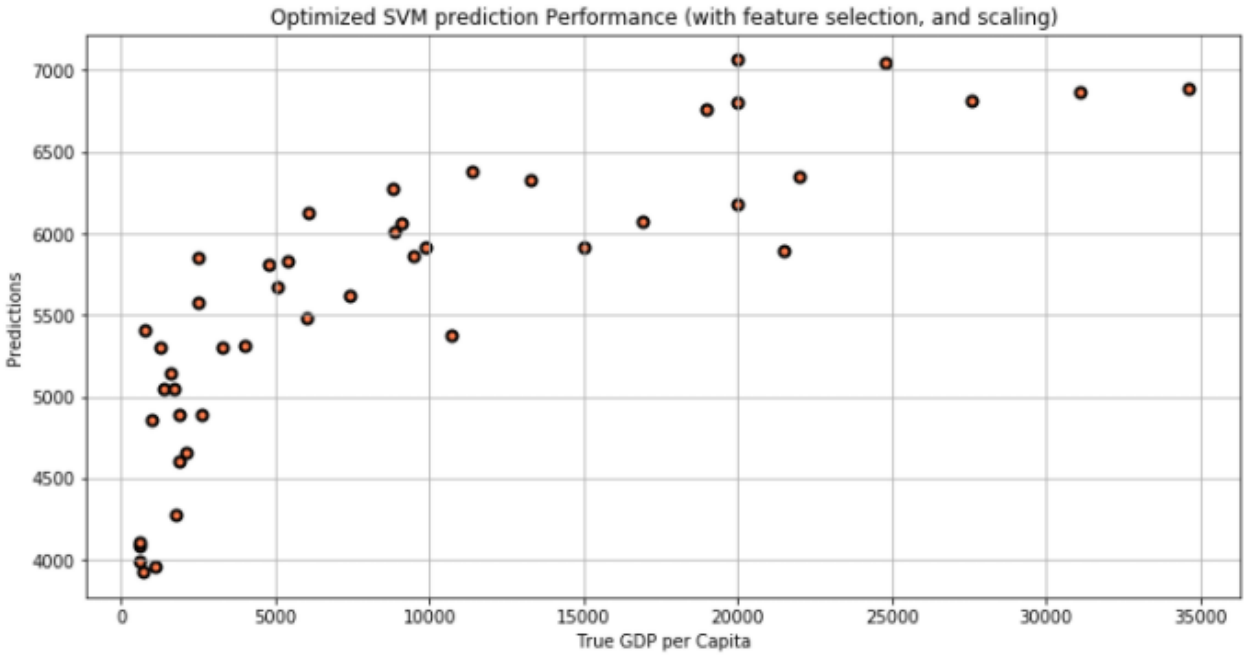
Content	**Base Model**	**First Model**	**Second Model**	**Third Model**
Type of Model Used	Linear Regression	Supply Vector Regressor	Random Forest Regressor	GradientBoostingRegressor
Rank Of Model Selection	3	4	1	2
Model Training	.fit()	.fit()	.fit()	.fit()
Predictions	.predict()	.predict()	.predict()	.predict()
Evaluations	MAE=330350.85 RMSE=1570337.5 R2_Score= -29843.12	MAE=7049.98 RMSE= 9811.73 R2_Score= -0.16	MAE= 2142.13 RMSE=3097.19 R2_Score=0.88	MAE= 2280.46 RMSE=3413.63 R2_Score=0.85
Optimization	-	GridSearchCV	GridSearchCV	GridSearchCV
Evaluations	-	MAE= 6386.41 RMSE= 9133.49 R2_Score= -0.009	MAE= 2360.74 RMSE=3219.92 R2_Score=0.87	MAE= 2362.93 RMSE=3469.36 R2_Score=0.85

Checking the Model Visualization

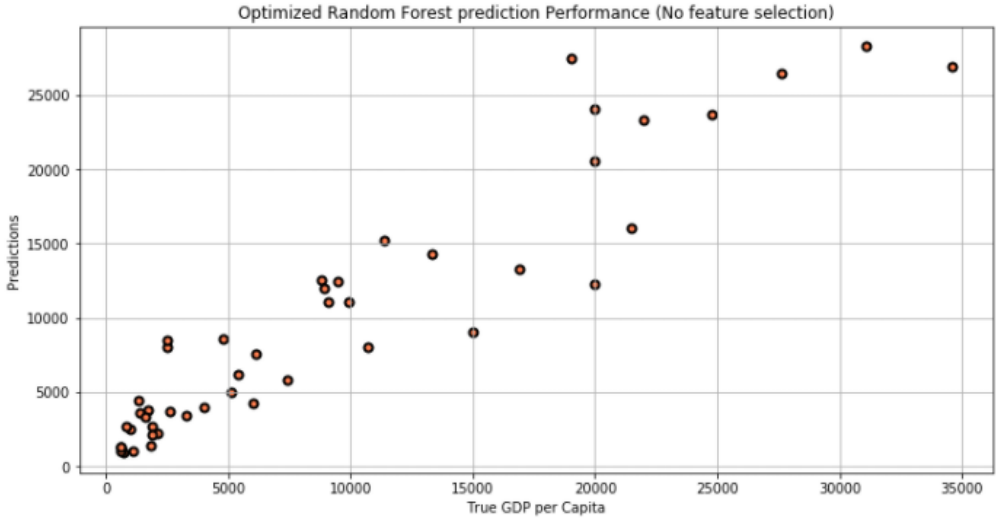
- Basic Model Evaluation Graphs



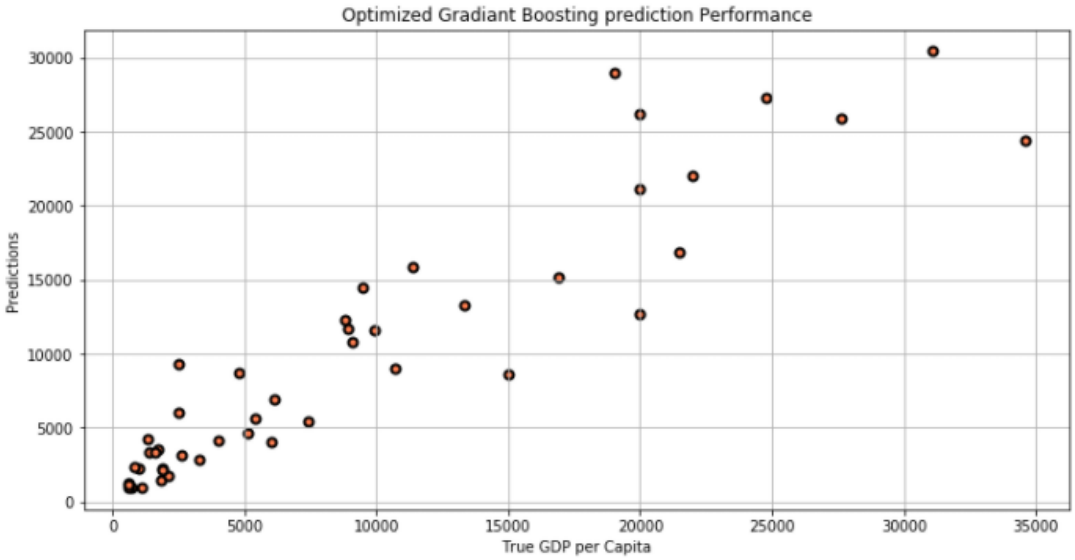
From the metrics above, it is clear that feature selection is essintial for linear regression model training, in order to get acceptable results on this dataset. On the other hand, feature scaling has a small positive effect on LR's prediction performance. we got decent prediction performance from LR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



SVM has improved a little with grid search, but it still performs below linear regression.



We can see that the optimization process on RF regressor has not changed the performance in a noticebale manner, yet the slight change was actually to the worst, that is probably because our initial parameters were already very close to the optimum ones.



Creation of App

Here, I am creating Streamlit App. It takes the value or input from the user. Performs Calculations. See the data type matches. Check for any missing value. It transforms the region. Then split the data. And making the final predictions.

Technical Aspect

Numpy used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. It contains a multi-dimensional array and matrix data structures. It can be utilised to perform a number of mathematical operations on arrays.

Pandas module mainly works with the tabular data. It contains Data Frame and Series. Pandas is 18 to 20 times slower than Numpy. Pandas is seriously a game changer when it comes to cleaning, transforming, manipulating and analyzing data.

Matplotlib is used for EDA. Visualization of graphs helps to understand data in better way than numbers in table format. Matplotlib is mainly deployed for basic plotting. It consists of bars, pies, lines, scatter plots and so on. Inline command display visualization inline within frontends like in Jupyter Notebook, directly below the code cell that produced it.

Seaborn provides a high-level interface for drawing attractive and informative statistical graphics. It provides a variety of visualization patterns and visualize random distributions.

Sklearn is known as scikit learn. It provides many ML libraries and algorithms for it. It provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.
Need to train_test_split - Using the same dataset for both training and testing leaves room for miscalculations, thus increases the chances of inaccurate predictions.
The train_test_split function allows you to break a dataset with ease while pursuing an ideal model. Also, keep in mind that your model should not be overfitting or underfitting.
Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science. In just a few minutes you can build and deploy powerful data apps. It is a very easy library to create a perfect dashboard by spending a little amount of time. It also comes with the inbuilt webserver and lets you deploy in the docker container. When you run the app, the localhost server will open in your browser automatically.
'StandardScaler()' removes the mean and scales each feature/variable to unit variance. This operation is performed feature-wise in an independent way. 'StandardScaler()' can be influenced by outliers (if they exist in the dataset) since it involves the estimation of the empirical mean and standard deviation of each feature.
"cross_val_score" splits the data into say 5 folds. Then for each fold it fits the data on 4 folds and scores the 5th fold. Then it gives you the 5 scores from which you can calculate a mean and variance for the score. You cross_val to tune parameters and get an estimate of the score.

Installation

Using intel core i5 9th generation with NVIDIA GFORECE GTX1650.
Windows 10 Environment Used.
Already Installed Anaconda Navigator for Python 3.x
The Code is written in Python 3.8.
If you don't have Python installed then please install Anaconda Navigator from its official site.
If you are using a lower version of Python you can upgrade using the pip package, ensuring you have the latest version of pip, *python* -m pip install --*upgrade pip and press Enter*.

Run/How to Use/Steps

Keep your internet connection on while running or accessing files and throughout too.
Follow this when you want to perform from scratch.
Open Anaconda Prompt, Perform the following steps:
cd <PATH>
pip install numpy
pip install pandas
pip install matplotlib
pip install seaborn
pip install streamlit
Note: If it shows error as 'No Module Found' , then install relevant module.
You can also create requirement.txt file as, pip freeze > requirements.txt
Create Virtual Environment:
conda create -n gdp python=3.6
y
conda activate gdp
cd <PATH-TO-FOLDER>

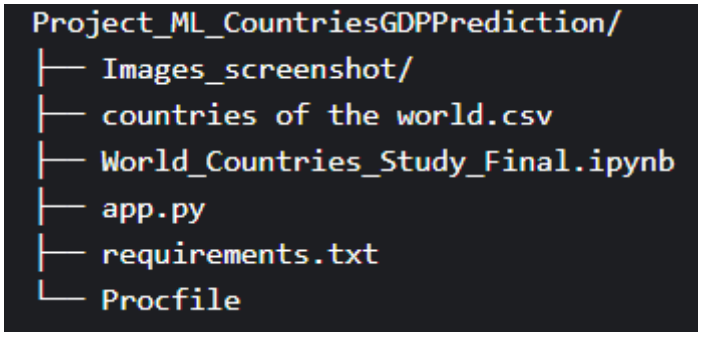
run .py or .ipynb files.
Paste URL to browser to check whether working locally or not.

Follow this when you want to just perform on local machine.
Download ZIP File.
Right-Click on ZIP file in download section and select Extract file option, which will unzip file.
Move unzip folder to desired folder/location be it D drive or desktop etc.
Open Anaconda Prompt, write cd <PATH> and press Enter.
eg: cd C:\Users\Monica\Desktop\Projects\Python Projects 1\
23)End_To_End_Projects\Project_10_ML_FileUse_End_To_End_Countries_GDP_Prediction\
Project_ML_CountriesGDPPrediction

conda create -n gdp python=3.6
y
conda activate gdp
In Anconda Prompt, pip install -r requirements.txt to install all packages.
In Anconda Prompt, write streamlit run app.py and press Enter.
Paste the URL (if it doesnot open automatically) to browser to check whether working locally or not.
Please be careful with spellings or numbers while typing filename and easier is just copy filename and then run it to avoid any silly errors.
Note: cd <PATH>
[Go to Folder where file is. Select the path from top and right-click and select copy option and paste it next to cd one space <path> and press enter, then you can access all files of that folder]
[cd means change directory]

Directory Tree/Structure of Project

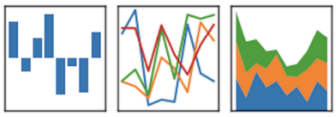
Folder: 23)End_To_End_Projects > Project_10_FileUse_End_To_End_Countries_GDP_Prediction> Project_ML_CountriesGDPPrediction
countries of the world.csv
World_Countries_Study_Final.ipynb
app.py
requirements.txt
Procfile



To Do/Future Scope

Can deploy on AWS and Google Cloud.

Technologies Used/System Requirements/Tech Stack



Download the Material

You can Download Dataset from here: https://github.com/monicadesAI-tech/Project_78/blob/main/countries%20of%20the%20world.csv

You can Download Entire Project from here: https://github.com/monicadesAI-tech/Project_78

You can Download Model Building from here:
https://github.com/monicadesAI-tech/Project_78/blob/main/World_Countries_Study_Final.ipynb

You can Download Web App_File from here: https://github.com/monicadesAI-tech/Project_78/blob/main/App/app.py

You can see Detailed Project at Website here: https://github.com/monicadesAI-tech.github.io/project/countriesgdp.html

Download dependencies from here: https://github.com/monicadesAI-tech/Project_78/blob/main/App/requirements.txt

Conclusion

- Modelling

More data cleaning may give better results with GBM.

- Analysis

I received MAE = 2142.13, RMSE = 3097.19 and R2_Score = 0.88 for Random Forest Model which is best in all comparatively.

Credits

Zeglam

Paper Citation

https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf
https://www.etasr.com/index.php/ETASR/article/view/2311