Data Analysis and Interpretation Capstone

Report on Net Migration causing factors (population, industry, urbanization) in year 2012

Introduction

This report is examining factors that cause human migration in year 2012. We will examine and check any relationships between predictor variables (FDIs,Health,Facilities,Populations) in regards to target variable which is Net Migration.

The dataset is provided by World Bank.

Data Dictionary

Field	Description
x131_2012	FOREIGN DIRECT INVESTMENT, NET INFLOWS (% OF GDP)
x142_2012	GDP PER CAPITA (CURRENT US$)
x150_2012	HEALTH EXPENDITURE, TOTAL (% OF GDP)
x155_2012	IMPROVED SANITATION FACILITIES (% OF POPULATION WITH ACCESS)
x156_2012	IMPROVED WATER SOURCE (% OF POPULATION WITH ACCESS)
x1_2012	ACCESS TO ELECTRICITY (% OF POPULATION)
x258_2012	RURAL POPULATION (% OF TOTAL POPULATION)
x283_2012	URBAN POPULATION (% OF TOTAL)
x31_2012	AGRICULTURAL LAND (% OF LAND AREA)
x195_2012	NET MIGRATION

Data Preparation

In [1]:
```
import pandas as pd
```

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

Out[2]:

	country	x1_2012	x2_2012	x9_2012	x11_2012	x12_2012	x14_2012	x15_2012	x16_2012	x18_2012	...	x244_2013
0	AFGHANISTAN	43.00000	19.480962	1.852684e+10	623.236804	0.677345	8.965464	1.833022e+09	1.600000	0.125552	...	0.0
1	ALBANIA	100.00000	62.086412	1.037147e+10	3575.766967	0.386039	12.381013	1.521033e+09	2.842804	2.730370	...	13.1
2	ALGERIA	100.00000	99.990000	1.480000e+11	3942.202841	0.620760	7.019903	1.417333e+10	4.467196	19.550386	...	2.4
3	AMERICAN SAMOA	59.32891	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN
4	ANDORRA	100.00000	100.000000	NaN	NaN	0.152983	9.237132	2.919443e+08	3.100000	0.000000	...	NaN

5 rows × 163 columns

In [3]:
```
df2 = df[["x131_2012","x142_2012","x150_2012","x155_2012","x156_2012","x1_2012","x258_2012","x283_2012","x31_20
```

In [4]:
```
df2.head()
```

Out[4]:

	x131_2012	x142_2012	x150_2012	x155_2012	x156_2012	x1_2012	x258_2012	x283_2012	x31_2012	x195_2012
0	0.299592	690.842629	8.479199	30.5	51.6	43.00000	74.532	25.468	58.067580	473007.0
1	7.468318	4247.485437	5.627439	92.1	95.4	100.00000	45.670	54.330	43.843066	-91750.0
2	0.717733	5583.616160	6.007278	87.0	84.9	100.00000	31.130	68.870	17.381700	-143268.0
3	NaN	NaN	NaN	62.5	100.0	59.32891	12.587	87.413	24.500000	NaN
4	NaN	39666.369210	8.302019	100.0	100.0	100.00000	13.292	86.708	42.978723	NaN

In [5]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 248 entries, 0 to 247
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   x131_2012  219 non-null    float64
 1   x142_2012  225 non-null    float64
 2   x150_2012  221 non-null    float64
 3   x155_2012  226 non-null    float64
 4   x156_2012  227 non-null    float64
 5   x1_2012    245 non-null    float64
 6   x258_2012  245 non-null    float64
 7   x283_2012  245 non-null    float64
 8   x31_2012   240 non-null    float64
 9   x195_2012  227 non-null    float64
dtypes: float64(10)
memory usage: 19.5 KB
```

In [6]:
```
df2.columns = ["fdi","gdp","health","sanitation","water","electricity","ruralpop","urbanpop","agri","migration"
```

In [7]:
```
df2
```

Out[7]:

	fdi	gdp	health	sanitation	water	electricity	ruralpop	urbanpop	agri	migration
0	0.299592	690.842629	8.479199	30.50000	51.600000	43.000000	74.532000	25.468000	58.067580	473007.0
1	7.468318	4247.485437	5.627439	92.10000	95.400000	100.000000	45.670000	54.330000	43.843066	-91750.0
2	0.717733	5583.616160	6.007278	87.00000	84.900000	100.000000	31.130000	68.870000	17.381700	-143268.0
3	NaN	NaN	NaN	62.50000	100.000000	59.328910	12.587000	87.413000	24.500000	NaN
4	NaN	39666.369210	8.302019	100.00000	100.000000	100.000000	13.292000	86.708000	42.978723	NaN
...	...	...	...	...	...	...	...	...	...	...
243	0.559266	2782.905026	NaN	92.20000	65.100000	97.697830	25.423000	74.577000	43.355482	-43750.0
244	2.537712	10460.113770	9.940811	65.93004	89.487651	84.583874	47.551252	52.448806	37.785216	0.0
245	-0.044394	1289.034078	5.553370	53.30000	54.900000	48.406710	67.126000	32.874000	44.604807	-50000.0
246	6.942853	1686.618024	4.754491	43.20000	63.000000	22.062560	60.413000	39.587000	32.063923	-34490.0
247	3.223668	850.827694	NaN	37.30000	77.500000	40.462560	67.166000	32.834000	41.876696	-219922.0

248 rows × 10 columns

In [8]:
```
df2.to_csv("wbresearch.csv", index=False)
```

In [9]:
```
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv("wbresearch.csv")
df.head()
```

Out[9]:

	fdi	gdp	health	sanitation	water	electricity	ruralpop	urbanpop	agri	migration
0	0.299592	690.842629	8.479199	30.5	51.6	43.00000	74.532	25.468	58.067580	473007.0
1	7.468318	4247.485437	5.627439	92.1	95.4	100.00000	45.670	54.330	43.843066	-91750.0
2	0.717733	5583.616160	6.007278	87.0	84.9	100.00000	31.130	68.870	17.381700	-143268.0
3	NaN	NaN	NaN	62.5	100.0	59.32891	12.587	87.413	24.500000	NaN
4	NaN	39666.369210	8.302019	100.0	100.0	100.00000	13.292	86.708	42.978723	NaN

Sample

The World Bank dataset has 248 observations with 163 columns. For this research, I chose 9 features to predict the net migration of people based on these factors.

Measures

There are six features measured in percentages. All the data is numeric type.

Analysis

I will use Pearson correlation to check the strength of collinearity with the target variable.

In [10]:
```
df.corr()
```

Out[10]:

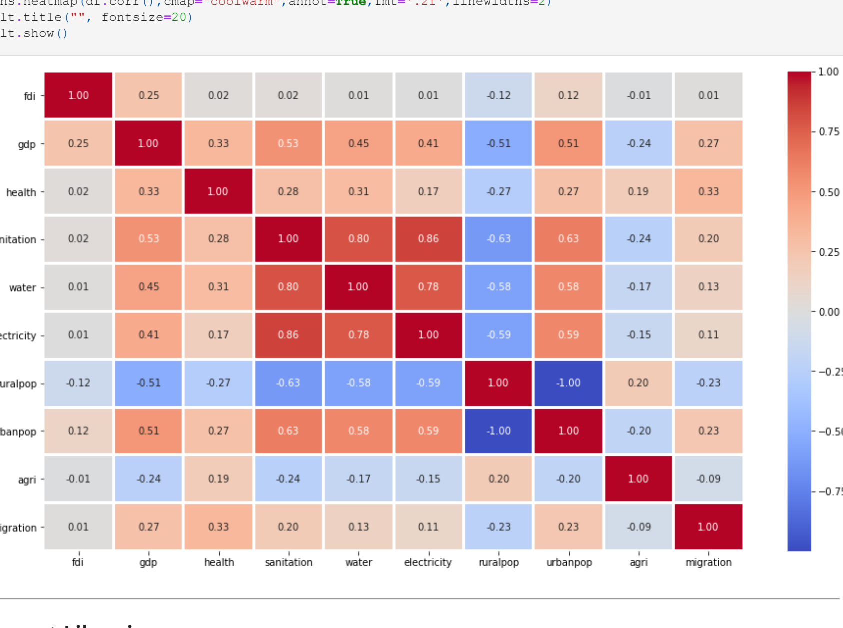
	fdi	gdp	health	sanitation	water	electricity	ruralpop	urbanpop	agri	migration
fdi	1.000000	0.250559	0.016223	0.023753	0.008043	0.005692	-0.121705	0.121705	-0.013489	0.011989
gdp	0.250559	1.000000	0.334146	0.533179	0.451355	0.411777	-0.508069	0.508069	-0.237129	0.266345
health	0.016223	0.334146	1.000000	0.276713	0.307054	0.165047	-0.271735	0.271734	0.194480	0.332993
sanitation	0.023753	0.533179	0.276713	1.000000	0.804177	0.858107	-0.631365	0.631364	-0.244808	0.202127
water	0.008043	0.451355	0.307054	0.804177	1.000000	0.781008	-0.581884	0.581884	-0.170557	0.134436
electricity	0.005692	0.411777	0.165047	0.858107	0.781008	1.000000	-0.585040	0.585039	-0.149063	0.107798
ruralpop	-0.121705	-0.508069	-0.271735	-0.631365	-0.581884	-0.585040	1.000000	-1.000000	0.197438	-0.228288
urbanpop	0.121705	0.508069	0.271734	0.631364	0.581884	0.585039	-1.000000	1.000000	-0.197438	0.228288
agri	-0.013489	-0.237129	0.194480	-0.244808	-0.170557	-0.149063	0.197438	-0.197438	1.000000	-0.085710
migration	0.011989	0.266345	0.332993	0.202127	0.134436	0.107798	-0.228288	0.228288	-0.085710	1.000000

In [11]:
```
df.corr()["migration"].sort_values()
```

Out[11]:

```
ruralpop      -0.228288
agri          -0.085710
fdi            0.011989
electricity    0.107798
water          0.134436
sanitation     0.202127
urbanpop       0.228288
gdp            0.266345
health         0.332993
migration      1.000000
Name: migration, dtype: float64
```

In [12]:
```
plt.figure(figsize=(16,9))
sns.heatmap(df.corr(),cmap="coolwarm",annot=True,fmt='.2f',linewidths=2)
plt.title("", fontsize=20)
plt.show()
```



Import Libraries

In [13]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import random

import sklearn


import statsmodels.api as sm


import datetime
from datetime import datetime, timedelta

import scipy.stats

#import xgboost as xgb
#from xgboost import XGBClassifier, XGBRegressor
#from xgboost import to_graphviz, plot_importance

#from sklearn.experimental import enable_hist_gradient_boosting
#from sklearn.linear_model import ElasticNet, Lasso, LinearRegression, LogisticRegression, Ridge
#from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor, ExtraTreesClassifier, ExtraTreesRe
#from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor, HistGradientBoostingClassi


%matplotlib inline
#sets the default autosave frequency in seconds
%autosave 60
sns.set_style('dark')
sns.set(font_scale=1.2)

plt.rc('axes', labelsize=14)
plt.rc('xtick', labelsize=12)
plt.rc('ytick', labelsize=12)


from sklearn.model_selection import cross_val_score, train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import cross_validate, KFold, RepeatedStratifiedKFold
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler, OneHotEncoder
#from sklearn.pipeline import Pipeline
#from sklearn.feature_selection import RFE, RFECV, SelectKBest, f_classif, f_regression, chi2
#from sklearn.inspection import permutation_importance

from sklearn.tree import export_graphviz, plot_tree
from sklearn.metrics import confusion_matrix, classification_report, mean_absolute_error, mean_squared_error,r2
from sklearn.metrics import plot_confusion_matrix, plot_precision_recall_curve, plot_roc_curve, accuracy_score
from sklearn.metrics import auc, f1_score, precision_score, recall_score, roc_auc_score

import warnings
warnings.filterwarnings('ignore')


# Use Feature-Engine library

#import feature_engine.missing_data_imputers as mdi
#from feature_engine.outlier_removers import Winsorizer
#from feature_engine import categorical_encoders as ce


#from pycaret.classification import *
#from pycaret.clustering import *
#from pycaret.regression import *

pd.set_option('display.max_columns',None)
#pd.set_option('display.max_rows',100)
pd.set_option('display.width', 1000)
pd.set_option('display.float_format','{:.2f}'.format)


random.seed(0)
np.random.seed(0)
np.set_printoptions(suppress=True)
```

```
Autosaving every 60 seconds
```

Exploratory Data Analysis

In [14]:
```
df = pd.read_csv("wbresearch.csv")
```

In [15]:
```
df
```

Out[15]:

	fdi	gdp	health	sanitation	water	electricity	ruralpop	urbanpop	agri	migration
0	0.30	690.84	8.48	30.50	51.60	43.00	74.53	25.47	58.07	473007.00
1	7.47	4247.49	5.63	92.10	95.40	100.00	45.67	54.33	43.84	-91750.00
2	0.72	5583.62	6.01	87.00	84.90	100.00	31.13	68.87	17.38	-143268.00
3	NaN	NaN	NaN	62.50	100.00	59.33	12.59	87.41	24.50	NaN
4	NaN	39666.37	8.30	100.00	100.00	100.00	13.29	86.71	42.98	NaN
...	...	...	...	...	...	...	...	...	...	...
243	0.56	2782.91	NaN	92.20	65.10	97.70	25.42	74.58	43.36	-43750.00
244	2.54	10460.11	9.94	65.93	89.49	84.58	47.55	52.45	37.79	0.00
245	-0.04	1289.03	5.55	53.30	54.90	48.41	67.13	32.87	44.60	-50000.00
246	6.94	1686.62	4.75	43.20	63.00	22.06	60.41	39.59	32.06	-34490.00
247	3.22	850.83	NaN	37.30	77.50	40.46	67.17	32.83	41.88	-219922.00

248 rows × 10 columns

In [16]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 248 entries, 0 to 247
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   fdi          219 non-null    float64
 1   gdp          225 non-null    float64
 2   health       221 non-null    float64
 3   sanitation   226 non-null    float64
 4   water        227 non-null    float64
 5   electricity  245 non-null    float64
 6   ruralpop     245 non-null    float64
 7   urbanpop     245 non-null    float64
 8   agri         240 non-null    float64
 9   migration    227 non-null    float64
dtypes: float64(10)
memory usage: 19.5 KB
```

In [17]:
```
df.describe(include='all')
```

Out[17]:

	fdi	gdp	health	sanitation	water	electricity	ruralpop	urbanpop	agri	migration
count	219.00	225.00	221.00	226.00	227.00	245.00	245.00	245.00	240.00	227.00
mean	5.74	14751.38	6.81	72.49	88.26	78.44	42.16	57.84	38.14	-98189.77
std	11.78	21611.52	2.79	28.58	14.54	29.21	23.50	23.50	21.15	2705600.34
min	-5.50	244.20	1.41	6.60	39.90	5.06	0.00	8.79	0.47	-16449428.00
25%	1.64	1719.04	4.82	49.49	84.75	59.33	22.88	38.49	22.10	-138149.50
50%	2.90	5967.00	6.23	84.10	94.10	96.12	42.29	57.71	38.58	-10000.00
75%	5.70	15317.14	8.59	97.00	99.25	100.00	61.51	77.12	53.33	29999.00
max	142.26	149160.76	17.05	100.00	100.00	100.00	91.20	100.00	81.30	16458326.00

In [18]:
```
df.shape
```

Out[18]:

```
(248, 10)
```

In [19]:
```
df.columns
```

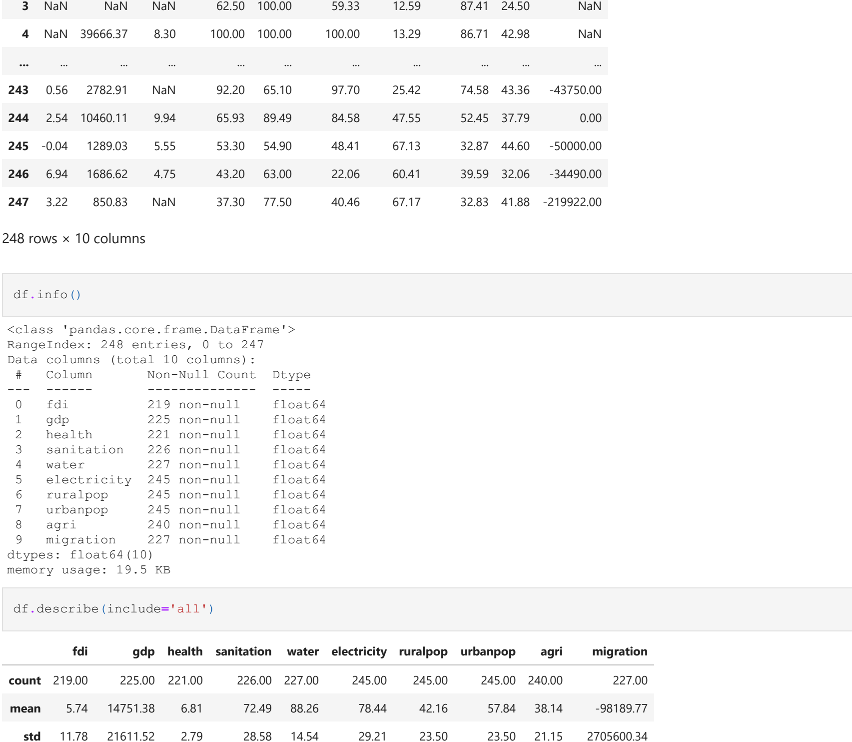
Out[19]:

```
Index(['fdi', 'gdp', 'health', 'sanitation', 'water', 'electricity', 'ruralpop', 'urbanpop', 'agri', 'migratio
n'], dtype='object')
```

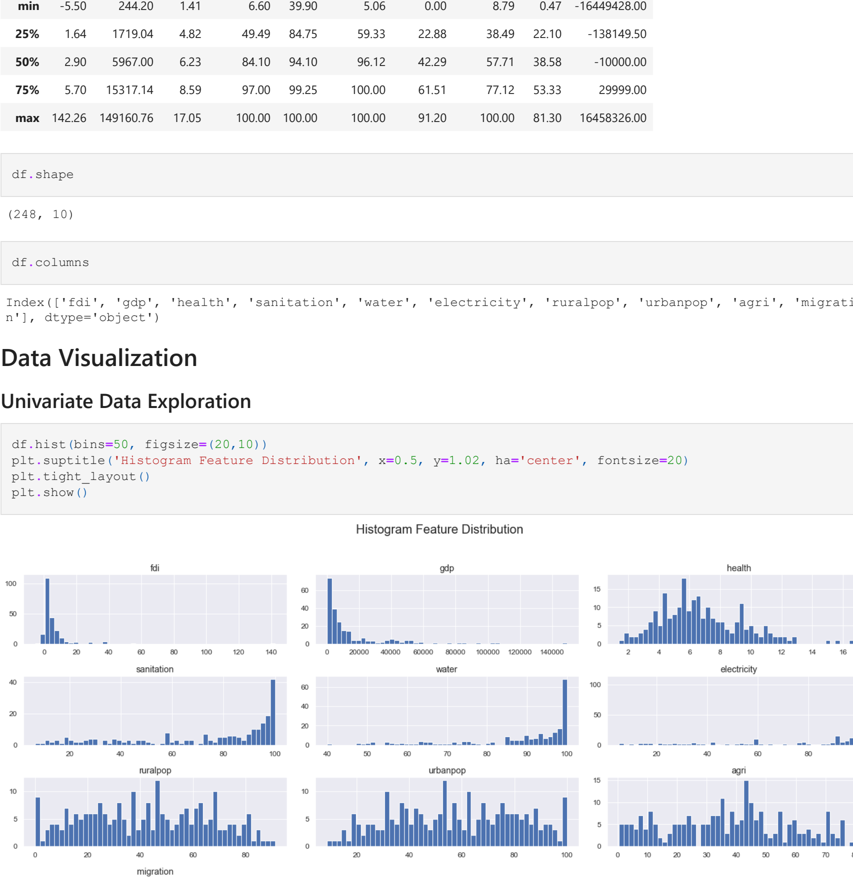
Data Visualization

Univariate Data Exploration

In [20]:
```
df.hist(bins=50, figsize=(20,10))
plt.suptitle('Histogram Feature Distribution', x=0.5, y=1.02, ha='center', fontsize=20)
plt.tight_layout()
plt.show()
```



In [21]:
```
df.boxplot(figsize=(20,10))
plt.suptitle('BoxPlots Feature Distribution', x=0.5, y=1.02, ha='center', fontsize=20)
plt.tight_layout()
plt.show()
```

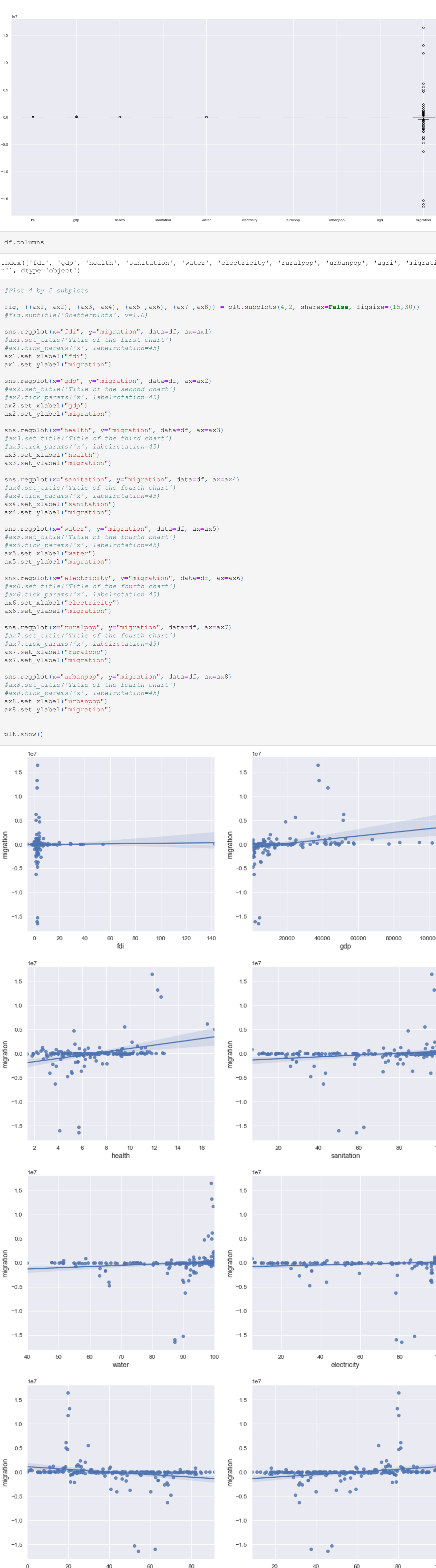


In [22]:
```
df.columns
```

Out[22]:

```
Index(['fdi', 'gdp', 'health', 'sanitation', 'water', 'electricity', 'ruralpop', 'urbanpop', 'agri', 'migratio
n'], dtype='object')
```

In [23]:
```
#Plot 4 by 2 subplots

fig, ((ax1, ax2), (ax3, ax4), (ax5 ,ax6), (ax7 ,ax8)) = plt.subplots(4,2, sharex=False, figsize=(15,30))
#fig.suptitle('Scatterplots', y=1.0)

sns.regplot(x="fdi", y="migration", data=df, ax=ax1)
#ax1.set_title('Title of the first chart')
#ax1.tick_params('x', labelrotation=45)
ax1.set_xlabel("fdi")
ax1.set_ylabel("migration")

sns.regplot(x="gdp", y="migration", data=df, ax=ax2)
#ax2.set_title('Title of the second chart')
#ax2.tick_params('x', labelrotation=45)
ax2.set_xlabel("gdp")
ax2.set_ylabel("migration")

sns.regplot(x="health", y="migration", data=df, ax=ax3)
#ax3.set_title('Title of the third chart')
#ax3.tick_params('x', labelrotation=45)
ax3.set_xlabel("health")
ax3.set_ylabel("migration")

sns.regplot(x="sanitation", y="migration", data=df, ax=ax4)
#ax4.set_title('Title of the fourth chart')
#ax4.tick_params('x', labelrotation=45)
ax4.set_xlabel("sanitation")
ax4.set_ylabel("migration")

sns.regplot(x="water", y="migration", data=df, ax=ax5)
#ax5.set_title('Title of the fourth chart')
#ax5.tick_params('x', labelrotation=45)
ax5.set_xlabel("water")
ax5.set_ylabel("migration")

sns.regplot(x="electricity", y="migration", data=df, ax=ax6)
#ax6.set_title('Title of the fourth chart')
#ax6.tick_params('x', labelrotation=45)
ax6.set_xlabel("electricity")
ax6.set_ylabel("migration")

sns.regplot(x="ruralpop", y="migration", data=df, ax=ax7)
#ax7.set_title('Title of the fourth chart')
#ax7.tick_params('x', labelrotation=45)
ax7.set_xlabel("ruralpop")
ax7.set_ylabel("migration")

sns.regplot(x="urbanpop", y="migration", data=df, ax=ax8)
#ax8.set_title('Title of the fourth chart')
#ax8.tick_params('x', labelrotation=45)
ax8.set_xlabel("urbanpop")
ax8.set_ylabel("migration")


plt.show()
```



Pairplots

In [24]:
```
sns.pairplot(df)
plt.suptitle('Pairplots of features', x=0.5, y=1.02, ha='center', fontsize=20)
plt.show()
```

In [25]:
```
-----------------------------------
```

```
  File "<ipython-input-25-f9c020b672ea>", line 1
    -----------------------------------
                                       ^
SyntaxError: invalid syntax
```

Data Preprocessing

Feature Engineering

In [ ]:

In [ ]:

In [ ]:

Drop unwanted features

In [ ]:
```
df.columns
```

In [ ]:
```
df.drop()
```

In [ ]:
```
df
```

Treat Missing Values

In [ ]:
```
df.isnull().sum()
```

In [ ]:
```
df[''] = df[''].replace(np.nan,df.mean())
```

In [ ]:
```
#imputer = mdi.MeanMedianImputer(imputation_method='median',variables=None)
```

In [ ]:
```
#imputer.fit(df)
```

In [ ]:
```
#df = imputer.transform(df)
```

In [ ]:
```
df.isnull().sum()
```

Replacing values

In [ ]:
```
df.replace()
```

In [ ]:

Rounding Values

In [ ]:
```
###pandas.DataFrame.round
df[['internetuserate']] = df[['internetuserate']].round(decimals=0)
```

Treat Duplicate Values

In [ ]:
```
df.duplicated(keep='first').sum()
```

In [ ]:
```
df[df.duplicated(keep=False)] #Check duplicate values
```

In [ ]:
```
df.drop_duplicates(ignore_index=True, inplace=True)
```

In [ ]:

Treat Outliers

In [ ]:
```
df.columns
```

In [ ]:
```
df
```

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

In [ ]:
```
#windsorizer = Winsorizer(distribution='skewed',tail='both',fold=1.5, variables=[])
```

In [ ]:
```
#windsorizer.fit(df)
```

In [ ]:
```
#df2 = windsorizer.transform(df)
```

In [ ]:
```
#df2
```

In [ ]:
```
#df2.describe()
```

In [ ]:
```
#windsorizer.left_tail_caps_
```

In [ ]:
```
#windsorizer.right_tail_caps_
```

Treat Data Types

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

In [ ]:
```
df["breastcancerper100th"] = df["breastcancerper100th"].astype('int')
```

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

Perform One-Hot Encoding

In [ ]:
```
pd.get_dummies(df_category)
```

In [ ]:
```
df["has_gas"] = pd.get_dummies(data=df["has_gas"],drop_first=True)
```

Visualize pairplots for multi-classification

In [ ]:
```
#Must convert all strings to numbers
sns.pairplot(df2,vars=df2.columns[:-1],hue='logo')
plt.show()
```

In [ ]:

Create and save processed dataset

In [ ]:
```
df.to_csv("",index=False)
```

In [ ]:

Train Test Split

```
X = df.iloc[:,:]
y = df.iloc[:,:]
```

```
df.shape
```

Treat Imbalance Data

```
y.value_counts()
```

```
ros = RandomOverSampler(sampling_strategy='all',random_state=0)
```

```
new_X, new_y = ros.fit_resample(X, y)
```

```
new_y[].value_counts()
```

```
new_X
```

Train Test Split Cont'd

```
X.values, y.values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Train Test Split to create Train, Validation and Test Set

```
#Set test set size
X_df, X_test, y_df, y_test = train_test_split(X.values, y.values, test_size=0.2, random_state=0)
```

```
X_df.shape, X_test.shape, y_df.shape, y_test.shape
```

```
X_train, X_val = train_test_split(X_df, test_size=0.2, random_state=0)
```

```
X_train.shape, X_val.shape
```

Feature Selection

Using SelectKBest

```
X_new = SelectKBest(f_regression, k=10).fit_transform(X_train,y_train)
```

```
X_new[0:5]
```

```

```

Univariate Feature Selection

```
select_feature = SelectKBest(chi2, k=10).fit(X_train,y_train)
```

```
select_feature.scores_
```

Recursive Feature Elimination

```
rfe = RFE(estimator=XGBRegressor(),n_features_to_select=10,verbose=1, step=1)
```

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

```
selected_rfe_features = pd.DataFrame({'Feature':list(X_train.columns),'Ranking':rfe.ranking_})
```

```
selected_rfe_features
```

Recursive Feature Elimnation with Cross Validation

```
rfecv = RFECV(estimator=XGBRegressor(), cv=5,scoring="neg_mean_squared_error",verbose=1, step=1)
```

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

```
print("Optimal no of features:", rfecv.n_features_)
```

```
print("Best features:", rfecv.support_)
```

Feature Scaling

```
X_train
```

```
encoder = LabelEncoder()
```

```
scaler = StandardScaler()
```

```
minmax = MinMaxScaler()
```

```
ohe = OneHotEncoder()
```

```
X_train_scaled = minmax.fit_transform(X_train)
```

```
X_test_scaled = minmax.transform(X_test)
```

```
X_train_scaled
```

```
X_test_scaled
```

```

```

```

```

```

```

Model Training

Using PyCaret

```
exp_reg = setup(data = df, target = '', session_id=0, normalize=True)
```

```
compare_models(exclude=['catboost','lightgbm','lda','qda','mlp','ada','nb','ridge','rbfsvm','svm'],fold=5)  #For
```

```
compare_models(exclude=['omp','br','ard','par','ransac','tr','huber','kr','svm','knn','dt','rf',
                        'et','ada','gbr','mlp','xgboost','lightgbm','catboost'],fold=5)  # For Regressor
```

```
model_selected = create_model('catboost')
```

```
print(model_selected)
```

```
tuned_model = tune_model(catboost, optimize='mse')
```

```
print(tuned_model)
```

```
plot_model(tuned_model)
```

```
plot_model(tuned_model, plot = 'error')
```

```
plot_model(tuned_model, plot='feature')
```

```
interpret_model(tuned_model)
```

```
evaluate_model(tuned_model)
```

```
predict_model(tuned_model)
```

```
final_model = finalize_model(tuned_model)
```

```
unseen_predictions = predict_model(final_model, data=data_unseen)
unseen_predictions.head()
```

Using Regression or Classification Models

```
reg_model
```

```
classi_model
```

K-Fold Cross-Validation (Generalization Performance)

```
lasso = Lasso(random_state=0)
```

```
kf = KFold(n_splits=5, shuffle=True, random_state=0)
```

```
lasso_cv = cross_validate(estimator=lasso, X=X_train_scaled, y=y_train, scoring="neg_root_mean_squared_error",
                          cv=kf, n_jobs=-1,return_train_score=True)
```

```
lasso_cv
```

```
np.mean(lasso_cv["train_score"]), np.std(lasso_cv["train_score"])
```

```
np.mean(lasso_cv["test_score"]), np.std(lasso_cv["test_score"])
```

```

```

```

```

```

```

```

```

```

```

Using TPOT

```
# tpot = TPOTClassifier(generations=3,population_size=10,scoring='accuracy', cv=5, verbosity=2, random_state=0,
#                       early_stop=1)
```

```
# tpot.fit(X_train,y_train)
```

```
# tpot.score(X_test, y_test)
```

Using XGBoost (Scikit-Learn)

Using RandomSearchCV

```
model = XGBRegressor(random_state=0, n_estimators=100, objective='reg:squarederror')
```

```
model = XGBClassifier(random_state=0, n_estimators=100, objective='softmax:multi')
```

```
parameters = {'max_depth': np.arange(3,10,1),
              'eta': np.arange(0.05,0.3,0.05),
              'n_estimators':np.arange(100,1000,100),
              'min_child_weight': np.arange(1,4,1),
              'gamma':np.arange(0,10,2),
              'subsample':np.arange(0.5,0.9,0.1),
              'colsample_bytree':np.arange(0.5,0.9,0.1),
              'reg_alpha':np.arange(0,1,0.1),
              'reg_lambda':np.arange(0,1,0.1)
             }
```

```
randm = RandomizedSearchCV(estimator=model, param_distributions = parameters, cv = 5, n_iter = 50,
                           n_jobs=-1, scoring='')
```

```
randm.fit(X, y)
```

```
randm.best_estimator_
```

```
randm.best_score_
```

```
randm.best_params_
```

```

```

```

```

Final Model

```
xgbmodel = XGBRegressor(random_state=0, n_estimators=100, objective='reg:squarederror')
```

```
xgbmodel = XGBClassifier(random_state=0, n_estimators=100, objective='binary:logistic')
```

```
xgbmodel = XGBClassifier(random_state=0, n_estimators=100, objective='softmax:multi')
```

```
xgbmodel.fit(X_train_scaled,y_train,eval_set=[(X_test_scaled,y_test)],eval_metric='rmse',early_stopping_rounds=
```

```
xgbmodel.fit(X_train_scaled,y_train,eval_set=[(X_test_scaled,y_test)],eval_metric='error',early_stopping_rounds
```

```
xgbmodel.fit(X_train_scaled,y_train,eval_set=[(X_test_scaled,y_test)],eval_metric='mlogloss',early_stopping_rou
```

```
y_pred = xgbmodel.predict(X_test_scaled)
```

```
y_pred
```

Model Evaluation

```
cm = confusion_matrix(y_test,y_pred)
cm
```

```
fig , ax = plt.subplots(figsize=(10,5))
sns.heatmap(cm, annot=True,fmt='.4g',linewidths=2, cmap='viridis')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
fig , ax = plt.subplots(figsize=(10,5))
plot_confusion_matrix(xgbmodel,X_test_scaled,y_test,values_format='.4g',ax=ax)
plt.show()
```

```
print(classification_report(y_test,y_pred))
```

```
plot_roc_curve(xgbmodel,X_test,y_test)
plt.show()
```

```
mse = mean_squared_error(y_test,y_pred)
mse
```

```
rmse = np.sqrt(mse)
rmse
```

```
r2score = r2_score(y_test,y_pred)
r2score
```

```
fig, ax = plt.subplots(figsize=(10,8))
sns.regplot(x=y_test, y=y_pred, ax=ax)
plt.title("Plot to compare actual vs predicted", fontsize=20)
plt.ylabel("Predicted")
plt.xlabel("Actual")
plt.show()
```

Plot Feature Importances

```
rf.feature_importances_
```

```
feat_importances = pd.Series(rf.feature_importances_, index=X.columns)
```

```
feat_importances
```

```
feat_importances.nlargest(10).plot(kind='barh', figsize=(10,10))
plt.title('Feature Importances')
plt.show()
```

The permutation based importance

```
perm_importance = permutation_importance(rf,X_test,y_test, random_state=0, scoring='neg_mean_squared_error')
```

```
sorted_idx = perm_importance.importances_mean.argsort()
plt.figure(figsize=(10,10))
plt.title("Permutation-based Importance")
plt.barh(X.columns[sorted_idx], perm_importance.importances_mean[sorted_idx])
plt.xlabel("Permutation Importance")
plt.show()
```

Compute Importance from SHAP Values

```
explainer = shap.TreeExplainer(rf)
```

```
shap_values = explainer.shap_values(X_test)
```

```
shap.summary_plot(shap_values, X_test, plot_type="bar")
```

```
shap.summary_plot(shap_values, X_test)
```

Available importance_types = [‘weight’, ‘gain’, ‘cover’, ‘total_gain’, ‘total_cover’]

```
X.columns
```

```
xgbmodel.get_booster().feature_names = [X.columns]
```

```
fig, ax = plt.subplots(figsize=(20,10))
xgb.plot_importance(xgbmodel.get_booster(),ax=ax)
plt.show()
```

```
xgb.to_graphviz(xgbmodel,num_trees=100)
```

Example:

f = 'gain'

XGBClassifier.get_booster().get_score(importance_type= f)

Plot Tree

```
X.columns
```

```
plt.figure(figsize=(40,25))
plot_tree(treeclf, feature_names=X.columns,class_names=['0','1'], fontsize=14, filled=True)
plt.show()
```

```

```

Cross-Validation

```
cv = cross_val_score(xgbmodel,X,y,cv=5,verbose=1,scoring='')
```

```
cv.mean()
```

Using XGBoost (API)

```

```

```
dtrain = xgb.DMatrix(data=X_train,label=y_train)
dtest = xgb.DMatrix(data=X_test,label=y_test)
```

```
params = {'n_estimators':,
          'learning_rate':,
          'max_depth':,
          'objective': '',
          'num_class': ,
          'seed': 0,
          'eval_metric':''}
```

```
xgbmodel = xgb.train(params=params,dtrain=dtrain,num_boost_round=100,evals=[(dtest,"Test")],
                     early_stopping_rounds=10)
```

```
y_pred = xgbmodel.predict(dtest)
```

```
y_pred
```

Cross-Validation (API)

```
cv = xgb.cv(params=params,
            dtrain=dtrain,
            num_boost_round=100,
            nfold=5,
            stratified=False,
            folds=None,
            metrics=('merror'),
            obj=None,
            feval=None,
            maximize=False,
            early_stopping_rounds=10,
            fpreproc=None,
            as_pandas=True,
            verbose_eval=None,
            show_stdv=True,
            seed=0,
            callbacks=None,
            shuffle=True,)
```

```
cv
```

```
cv['test-merror-mean'].min()
```

Model Evaluation (Classification)

```
cm = confusion_matrix(y_test,y_pred)
cm
```

```
fig , ax = plt.subplots(figsize=(10,5))
sns.heatmap(cm, annot=True,fmt='.4g',linewidths=2, cmap='viridis')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

```
fig , ax = plt.subplots(figsize=(10,5))
plot_confusion_matrix(xgbmodel,X_test_scaled,y_test,values_format='.4g',ax=ax)
plt.show()
```

```
print(classification_report(y_test,y_pred))
```

```
plot_roc_curve(xgbmodel,X_test,y_test)
plt.show()
```

Model Evaluation (Regression)

```
mse = mean_squared_error(y_test,y_pred)
mse
```

```
rmse = np.sqrt(mse)
rmse
```

```
r2score = r2_score(y_test,y_pred)
r2score
```

```
fig, ax = plt.subplots(figsize=(10,8))
sns.regplot(x=y_test, y=y_pred, ax=ax)
plt.title("Plot to compare actual vs predicted")
plt.ylabel("Predicted")
plt.xlabel("Actual")
plt.show()
```

Table Formatted View

```
table = X_test.copy()
```

```
table["True Value"] = y_test.copy()
```

```
table["Predicted"] = np.round(lr_pred,2)
```

```
table
```

Cross-Validation

```
cv = cross_val_score(xgbmodel,X,y,cv=5,verbose=1,scoring='accuracy')
```

```
cv.mean()
```

Feature Selection

```
df.columns
```

```
df2 = df[['']]
```

```
df2
```

```
X = df2.iloc[:,0:7]
y = df2.iloc[:,7]
```

```
X.values, y.values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
xgbmodel2 = XGBClassifier(random_state=0, n_estimators=100, objective='binary:logistic')
```

```
xgbmodel2 = XGBRegressor(random_state=0, n_estimators=100, objective='reg:squarederror')
```

```
xgbmodel2.fit(X_train,y_train,eval_set=[(X_test,y_test)],eval_metric='error',early_stopping_rounds=10)
```

```
xgbmodel2.fit(X_train,y_train,eval_set=[(X_test,y_test)],eval_metric='rmse',early_stopping_rounds=10)
```

```
y_pred = xgbmodel2.predict(X_test)
```

```
y_pred
```

```
fig , ax = plt.subplots(figsize=(5,5))
plot_confusion_matrix(xgbmodel2,X_test,y_test,values_format='4g',ax=ax)
plt.show()
```

```
print(classification_report(y_test,y_pred))
```

```
plot_roc_curve(xgbmodel2,X_test,y_test)
plt.show()
```

Model Prediction

```
testdata = pd.read_csv()
```

```
answer = xgbmodel.predict(testdata)
```

```
answer
```

Model Tuning

Using RandomSearchCV

```
model = XGBClassifier(random_state=0, n_estimators=100, objective='binary:logistic')
```

```
model = XGBRegressor(random_state=0, n_estimators=100, objective='reg:squarederror')
```

```
parameters = {'max_depth': np.arange(3,10,1),
              'learning rate': np.arange(0.05,0.3,0.03),
              'n_estimators':np.arange(100,1000,100),
              'min_child_weight': np.arange(1,4,1),
              'gamma':np.arange(0,50,2),
              'subsample':np.arange(0.5,0.9,0.1),
              'colsample_bytree':np.arange(0.5,0.9,0.1)
             }
```

```
randm = RandomizedSearchCV(estimator=model, param_distributions = parameters, cv = 5, n_iter = 50,
                           n_jobs=-1, scoring='')
```

```
randm.fit(X, y)
```

```
randm.best_estimator_
```

```
randm.best_score_
```

```
randm.best_params_
```

Using GridSearchCV

```
model = XGBClassifier(random_state=0, n_estimators=100, objective='binary:logistic')
```

```
parameters = {'max_depth': np.arange(3,10,1),
              'learning rate': np.arange(0.05,0.3,0.03),
              'n_estimators':np.arange(100,1000,100),
              'min_child_weight': np.arange(1,4,1),
              'gamma':np.arange(0,50,2),
              'subsample':np.arange(0.5,0.9,0.1),
              'colsample_bytree':np.arange(0.5,0.9,0.1)
             }
```

```
grids = GridSearchCV(estimator=model,param_grid=parameters,scoring='accuracy',
                     n_jobs=-1,cv=5,verbose=1,return_train_score=True)
```

```
grids.fit(X,y)
```

```
grids.best_estimator_
```

```

```

Final Model

```
xgbnew = XGBClassifier(random_state=0, n_estimators=, objective='binary:logistic',max_depth=,
                       gamma=, min_child_weight=,learning_rate=,subsample=,colsample_bytree=)
```

```
xgbnew = XGBRegressor(random_state=0, n_estimators=, objective='binary:logistic',max_depth=,
                      gamma=, min_child_weight=,learning_rate=,subsample=,colsample_bytree=)
```

```
xgbnew.fit(X_train,y_train,eval_set=[(X_test,y_test)],eval_metric='error',early_stopping_rounds=10)
```

```
y_pred = xgbnew.predict(X_test)
```

```
y_pred
```

```
fig , ax = plt.subplots(figsize=(5,5))
plot_confusion_matrix(xgbnew,X_test,y_test,values_format='4g',ax=ax)
plt.show()
```

```
print(classification_report(y_test,y_pred))
```

```
plot_roc_curve(xgbnew,X_test,y_test)
plt.show()
```

Save the Model

```
filename = 'model.sav'
dump(xgbnew,open(filename,'wb'))
```

Load the Model

```
loaded_model = load(open(filename,'rb'))
```

```
loaded_model
```

Python code done by Dennis Lam

```

```