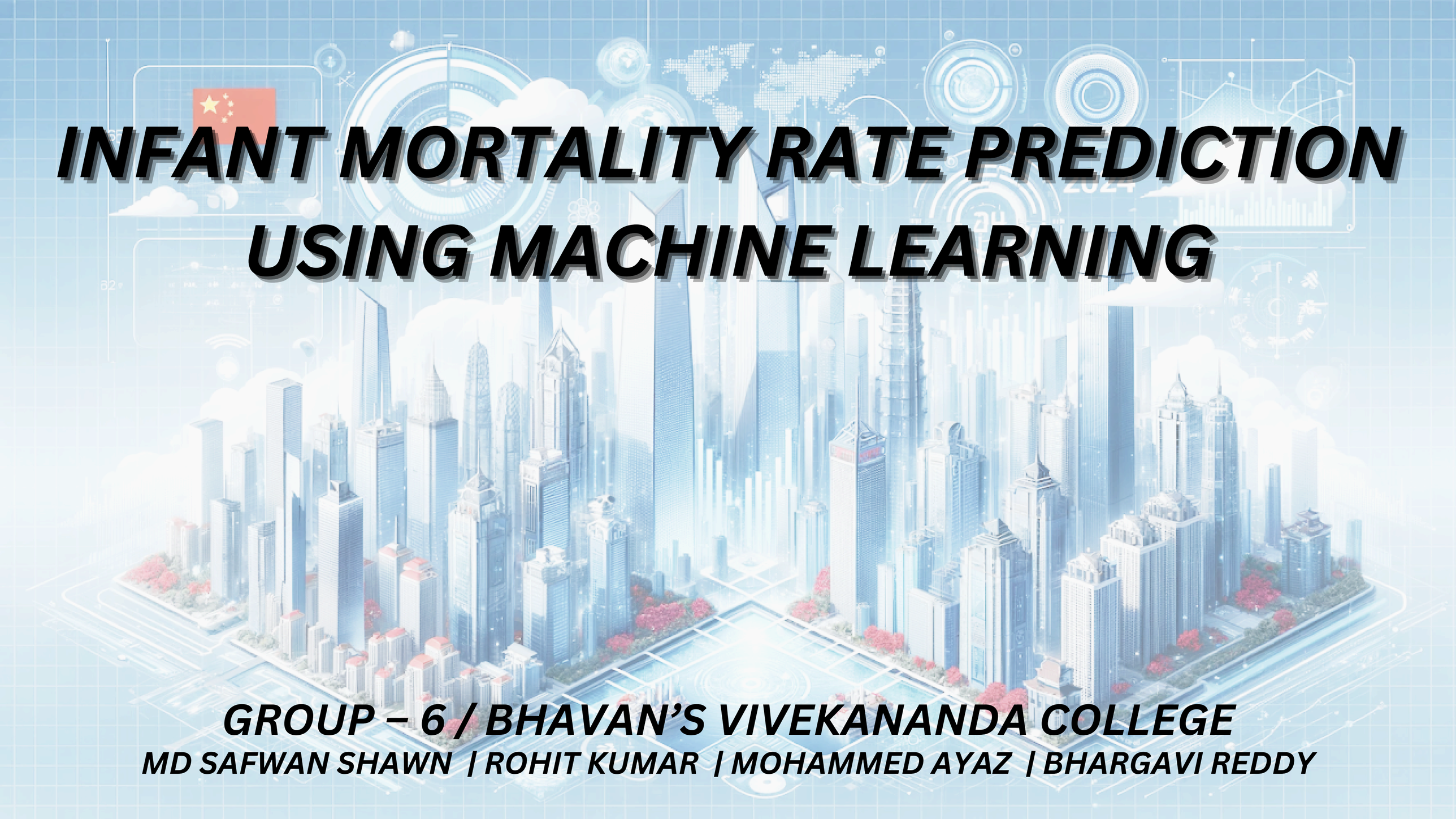


INFANT MORTALITY RATE PREDICTION USING MACHINE LEARNING

GROUP – 6 / BHAVAN'S VIVEKANANDA COLLEGE

MD SAFWAN SHAWN | ROHIT KUMAR | MOHAMMED AYAZ | BHARGAVI REDDY

ABSTRACT

The "Global Country Macro Statistics Prediction" project aims to develop a comprehensive predictive model for macroeconomic indicators across various countries. By leveraging advanced statistical techniques and machine learning algorithms, the project seeks to forecast key economic metrics such as GDP growth, inflation rates, unemployment rates, and trade balances. The model will utilize historical data, real-time economic indicators, and external factors like geopolitical events and policy changes to provide accurate and timely predictions. The ultimate goal is to assist policymakers, economists, and businesses in making informed decisions by offering insights into future economic trends and potential risks.

OBJECTIVE

To find the suitable machine learning model to predict the Global Country Statistics for implementing eco-friendly production methods.

CONTENTS

- *Introduction* 4
- *Data preprocessing* 5
- *EDA* 8
- *Machine Learning Algorithm's* 15
- *Summary* 24
- *Future Scope & Insights* 25
- *Appendix* 29

Introduction

- ***The "Global Country Macro Statistics Prediction" project has the potential to be a game-changer in how we forecast and respond to economic trends. By leveraging sophisticated algorithms and diverse data sources, it could provide accurate, real-time insights into critical macroeconomic indicators, supporting more informed decision-making at the global level. However, careful attention will be needed to manage data quality, model transparency, and the unpredictable nature of global events.***

DATA PREPROCESSING

Data

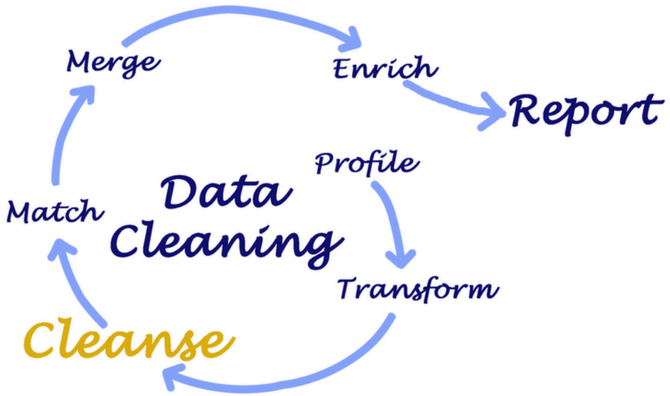
Dataset: ***Our data consist of 35 variables and 195 records***

Variables:

Continuous Variables	Categorical Variables
Birth Rate	CPI
Fertility Rate	CO2 Emissions
Infant Mortality	CPI Change(%)
Life Expectancy	GDP
	Population

Data Cleaning:

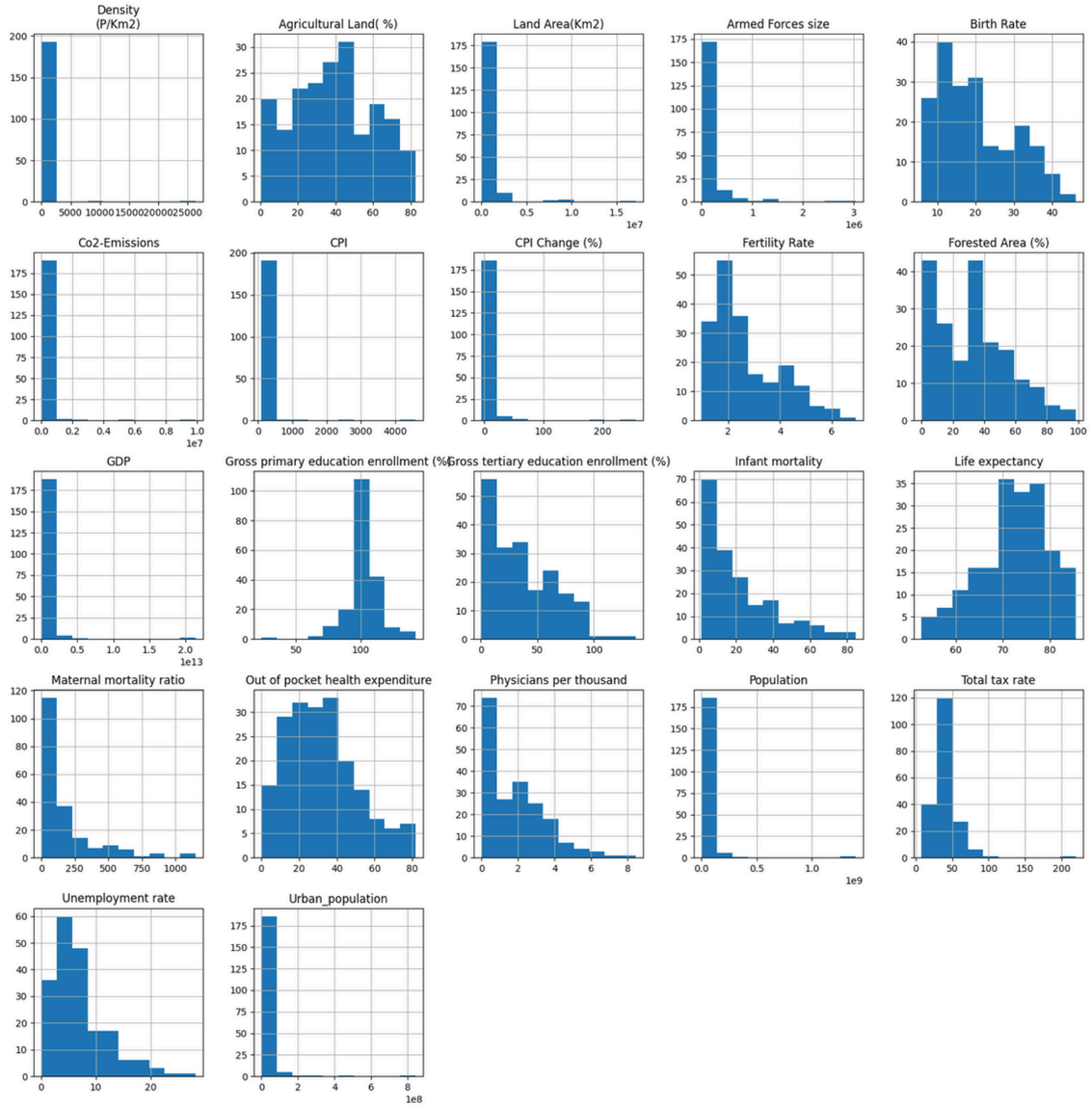
Data cleaning is the process of fixing or removing incorrect, incomplete, or duplicate data from a dataset Data cleaning is an important step in data preprocessing. It helps to ensure that the data used for analysis or machine learning is reliable and high quality.



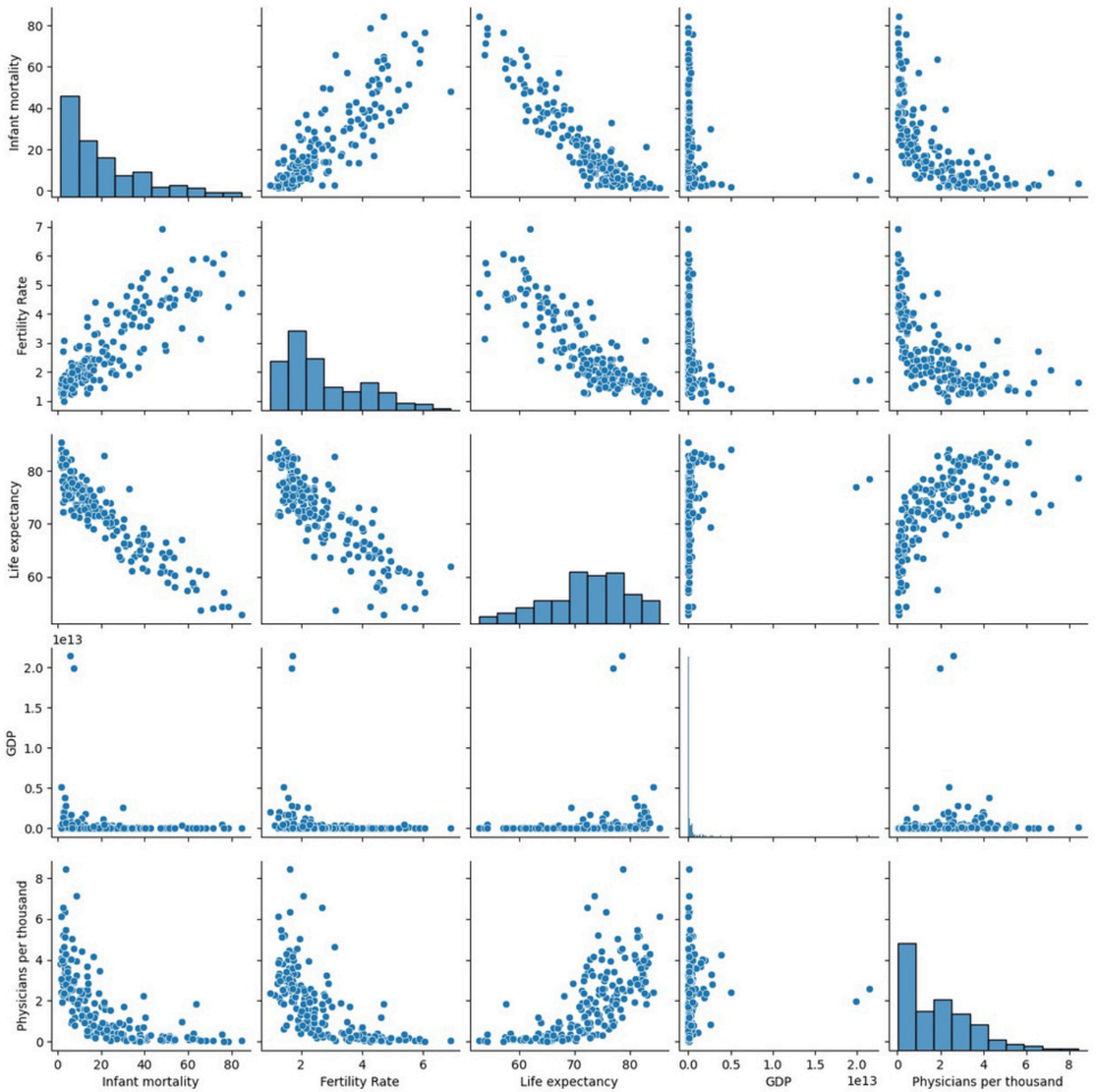
- *To handle missing values:*
- *Identified Missing Values: Checked for missing data across variables.*
- *Threshold Applied: Removed variables with missing values above a set threshold (e.g., 30%).*
- *Dummy Variable Added: Created a binary dummy variable to flag missing entries (1 for missing, 0 for present).*

EXPLORATORY DATA ANALYSIS

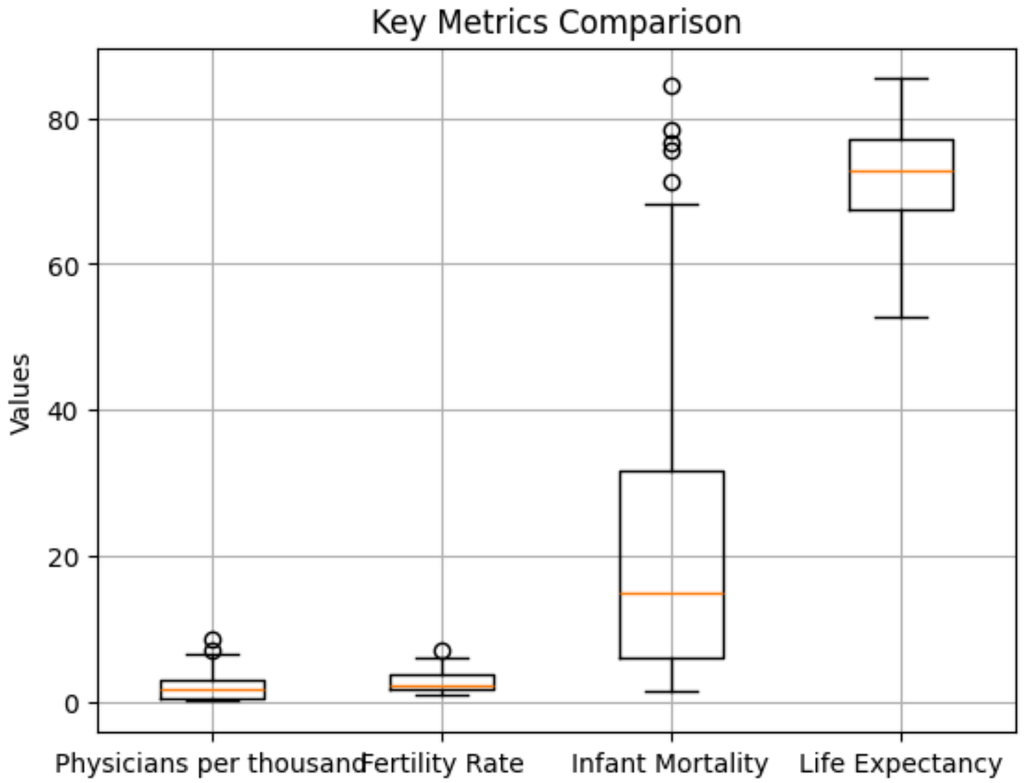
Histogram

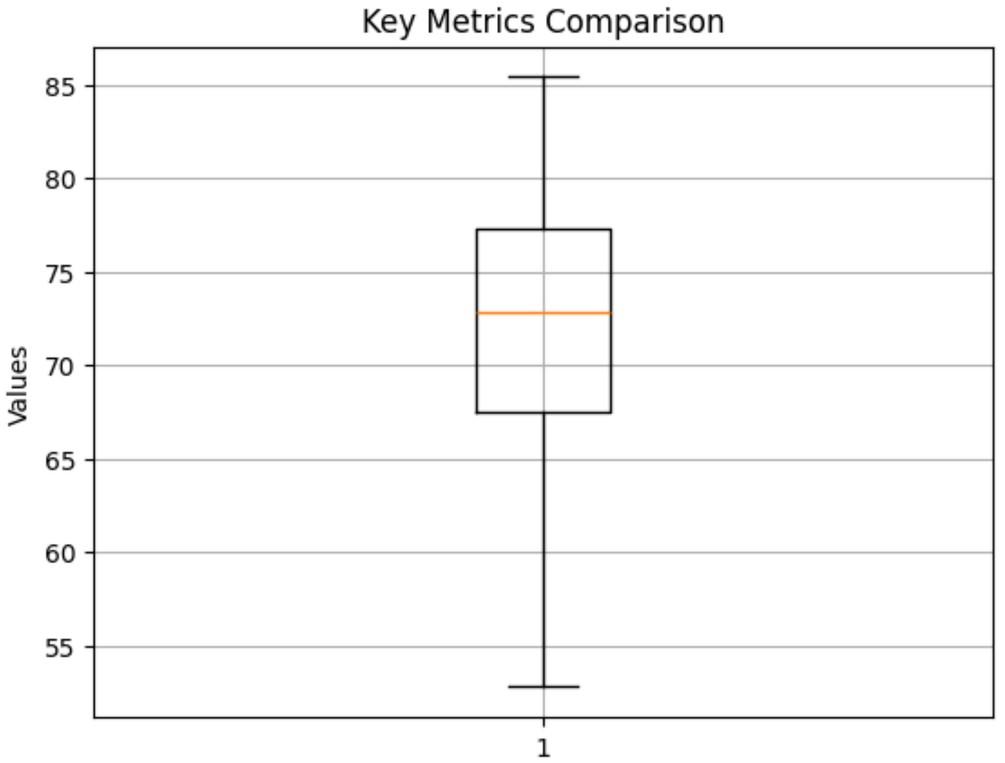

Density (P/Km2)
Agricultural Land(%)
Land Area(Km2)
Armed Forces size
Birth Rate
Co2-Emissions
CPI
CPI Change (%)
Fertility Rate
Forested Area (%)
GDP
Gross primary education enrollment (%)
Gross tertiary education enrollment (%)
Infant mortality
Life expectancy
Maternal mortality ratio
Out of pocket health expenditure
Physicians per thousand
Population
Total tax rate
Unemployment rate
Urban_population

Pair Plot


Infant mortality
Fertility Rate
Life expectancy
GDP
1e13
Physicians per thousand
Infant mortality
Fertility Rate
Life expectancy
GDP
1e13
Physicians per thousand

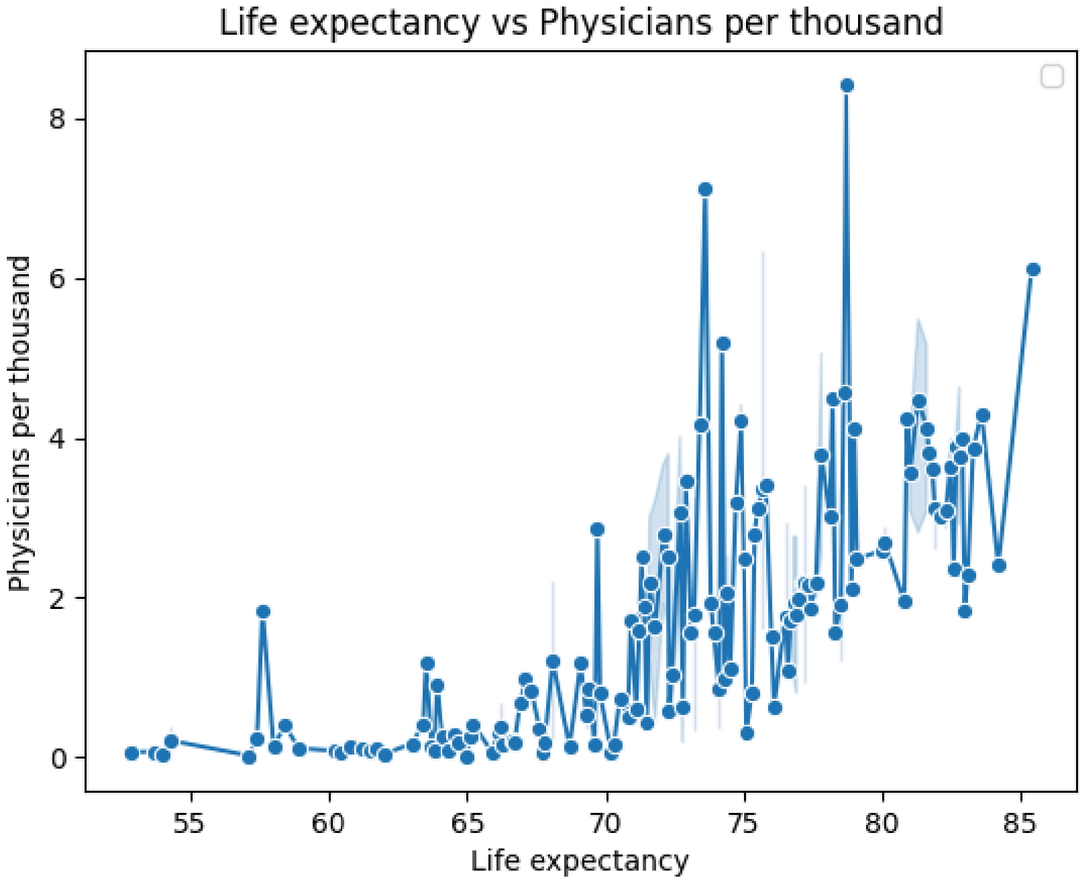
Box Plot



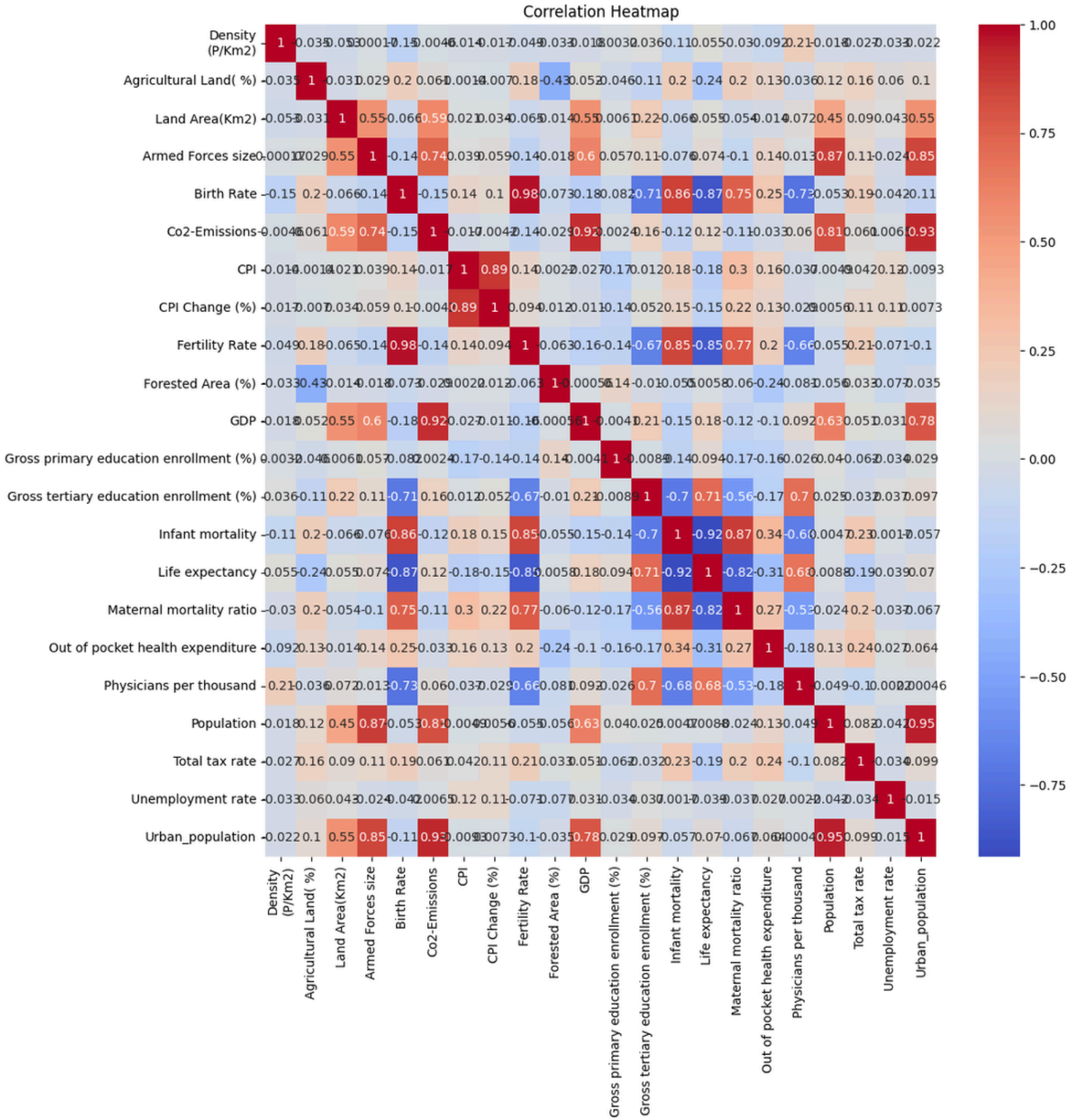


Life Expectancy

Line Plot


Life expectancy vs Physicians per thousand
Physicians per thousand
Life expectancy
0
2
4
6
8
55
60
65
70
75
80
85

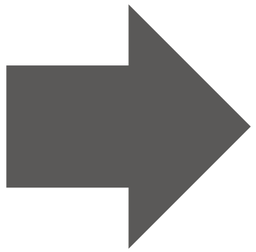
Correlation Matrix



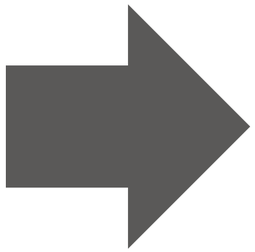
Multicollinearity Check

We dropped the variables with a VIF greater than 3.4

	variables	VIF
0	Density\n(P/Km2)	1.4
1	Agricultural Land(%)	1.4
2	Land Area(Km2)	2.0
3	Armed Forces size	5.4
4	Birth Rate	49.0
5	Co2-Emissions	24.2
6	CPI	5.7
7	CPI Change (%)	5.4
8	Fertility Rate	40.7
9	Forested Area (%)	1.4
10	GDP	8.8
11	Gross primary education enrollment (%)	1.2
12	Gross tertiary education enrollment (%)	2.9
13	Life expectancy	0.9
14	Maternal mortality ratio	3.2
15	Out of pocket health expenditure	1.4
16	Physicians per thousand	3.2
17	Population	26.4
18	Total tax rate	1.3
19	Unemployment rate	1.0
20	Urban_population	49.6



	variables	VIF
0	Density\n(P/Km2)	1.1
1	Agricultural Land(%)	1.4
2	Land Area(Km2)	1.8
3	Armed Forces size	5.3
4	CPI	5.7
5	CPI Change (%)	5.4
6	Fertility Rate	3.1
7	Forested Area (%)	1.3
8	GDP	2.3
9	Gross primary education enrollment (%)	1.0
10	Gross tertiary education enrollment (%)	2.8
11	Life expectancy	0.9
12	Maternal mortality ratio	3.1
13	Out of pocket health expenditure	1.3
14	Physicians per thousand	2.6
15	Population	5.0
16	Total tax rate	1.3
17	Unemployment rate	1.0

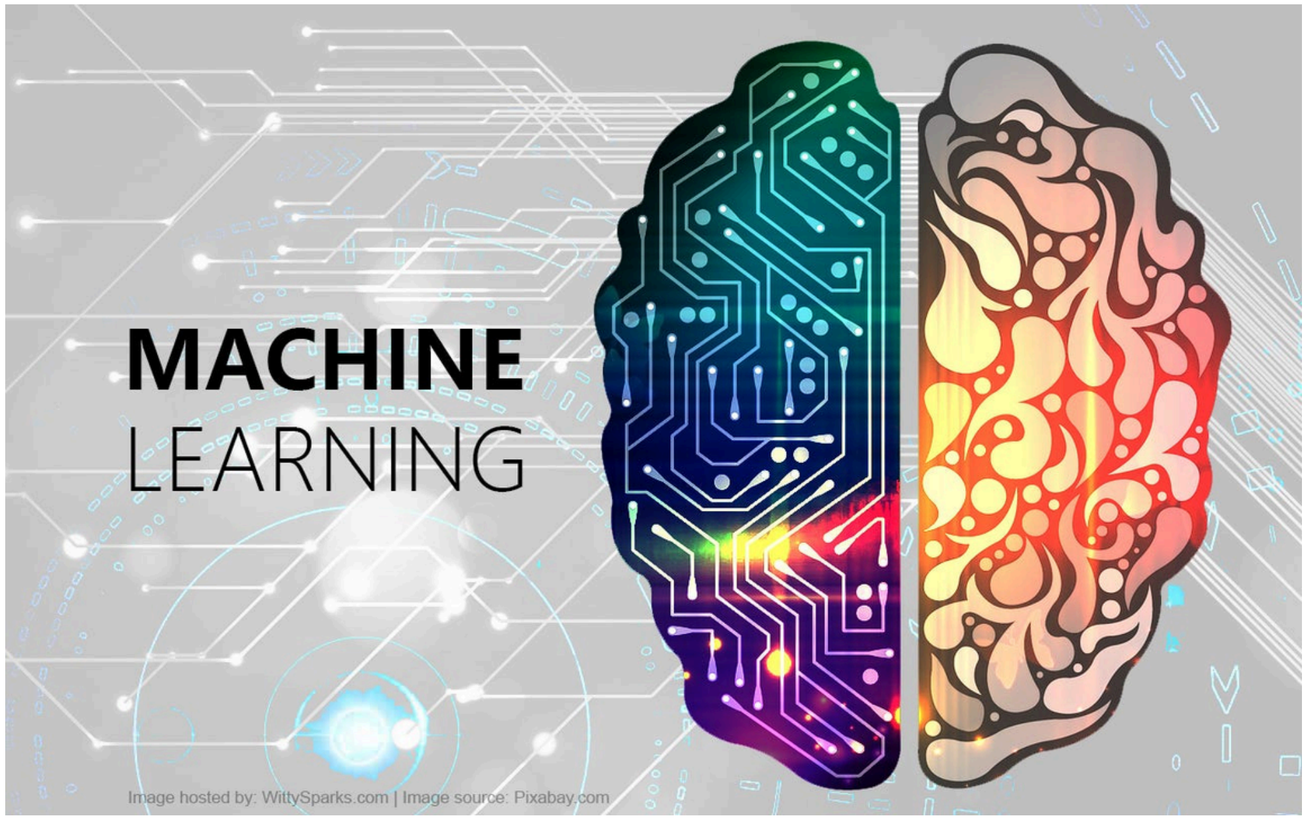


	variables	VIF
0	Density\n(P/Km2)	1.1
1	Agricultural Land(%)	1.4
2	Land Area(Km2)	1.6
3	CPI Change (%)	1.1
4	Fertility Rate	3.1
5	Forested Area (%)	1.3
6	GDP	2.3
7	Gross primary education enrollment (%)	1.0
8	Gross tertiary education enrollment (%)	2.8
9	Life expectancy	0.9
10	Maternal mortality ratio	2.9
11	Out of pocket health expenditure	1.3
12	Physicians per thousand	2.6
13	Population	1.9
14	Total tax rate	1.2
15	Unemployment rate	1.0

MACHINE LEARNING ALGORITHM

ML algorithms

- *Multiple Linear Regression*
- *K-Nearest Neighbors(KNN)*
- *Decision Tree*
- *Random Forest*
- *Bagging*
- *Boosting*
- *Support Vector Machine(SVM)*



80-20 TRAIN-TEST SPLIT

Algorithms	Model 1 r2 score	Model 1 RMSE	Model 2 r2 score	Model 2 RMSE
Linear Regression	0.897	7.700	0.893	7.861
KNN	-0.012	24.249	-0.012	24.249
Decision Tree	0.862	8.935	0.900	7.613
Random Forest	0.897	7.703	0.881	8.299
XGBoost	0.893	7.855	0.897	7.712
AdaBoost	0.870	8.683	0.891	7.942
SVM	-0.187	26.262	-0.187	26.262

Model 1: Before VIF | Model 2: After VIF

75:25 TRAIN TEST SPLIT

Algorithms	Model 1 r2 score	Model 1 RMSE	Model 2 r2 score	Model 2 RMSE
Linear Regression	0.902	7.169	0.895	7.401
KNN	0.047	22.307	0.047	22.307
Decision Tree	0.825	9.570	0.850	8.848
Random Forest	0.890	7.573	0.903	7.108
XGBoost	0.892	7.521	0.899	7.684
AdaBoost	0.897	7.327	0.866	8.365
SVM	-0.167	24.686	-0.167	24.686

Model 1: Before VIF | Model 2: After VIF

70:30 TRAIN-TEST SPLIT

Algorithms	Model 1 r2 score	Model 1 RMSE	Model 2 r2 score	Model 2 RMSE
Linear Regression	0.902	6.734	0.894	6.988
KNN	0.020	21.266	0.020	21.266
Decision Tree	0.819	9.144	0.834	8.755
Random Forest	0.889	7.164	0.897	6.908
XGBoost	0.904	6.662	0.902	6.712
AdaBoost	0.879	7.461	0.919	6.122
SVM	-0.105	22.579	-0.105	22.579

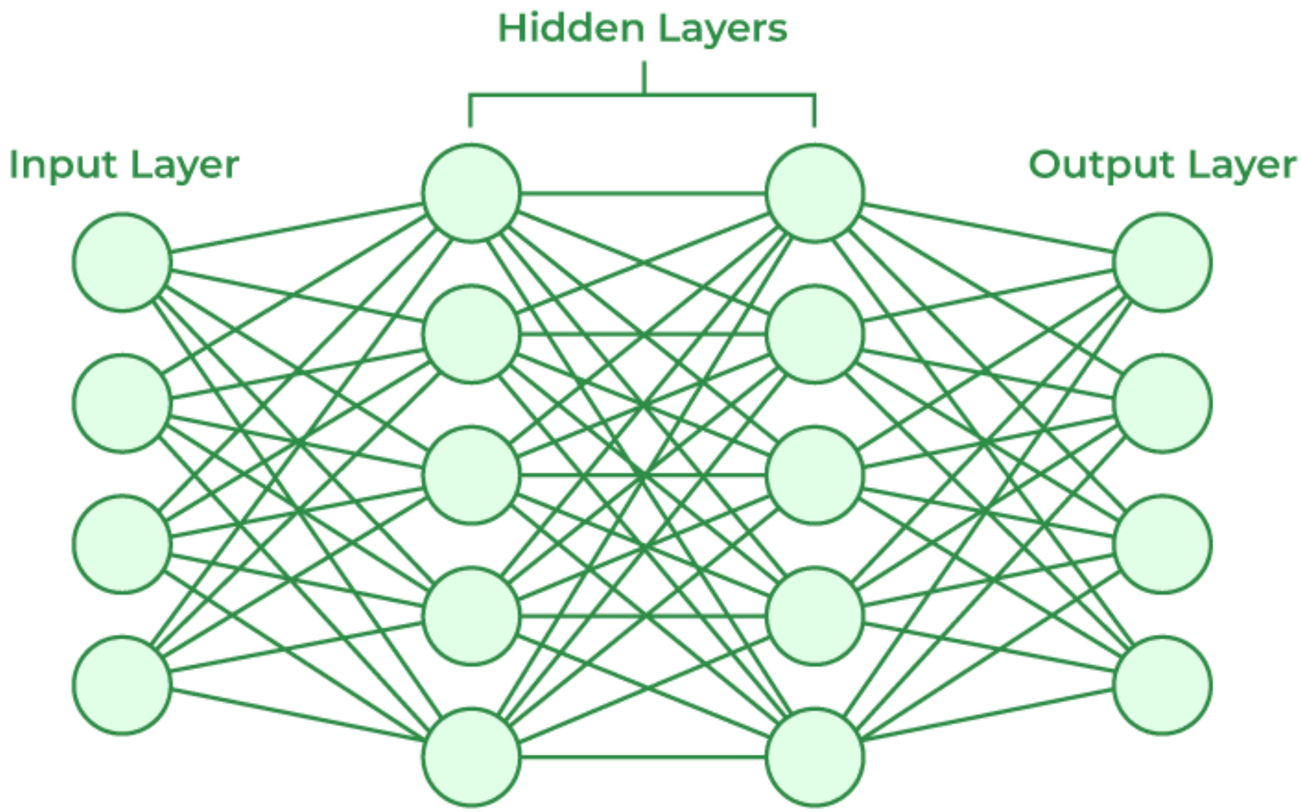
Model 1: Before VIF | Model 2: After VIF

60:40 TRAIN -TEST SPLIT

Algorithms	Model 1 r2 score	Model 1 RMSE	Model 2 r2 score	Model 2 RMSE
Linear Regression	0.896	6.606	0.892	6.735
KNN	0.099	19.430	0.099	19.430
Decision Tree	0.784	9.521	0.779	9.637
Random Forest	0.888	6.850	0.872	7.335
XGBoost	0.882	7.025	0.877	7.172
AdaBoost	0.801	9.140	0.804	9.062
SVM	-0.143	21.903	-0.144	21.903

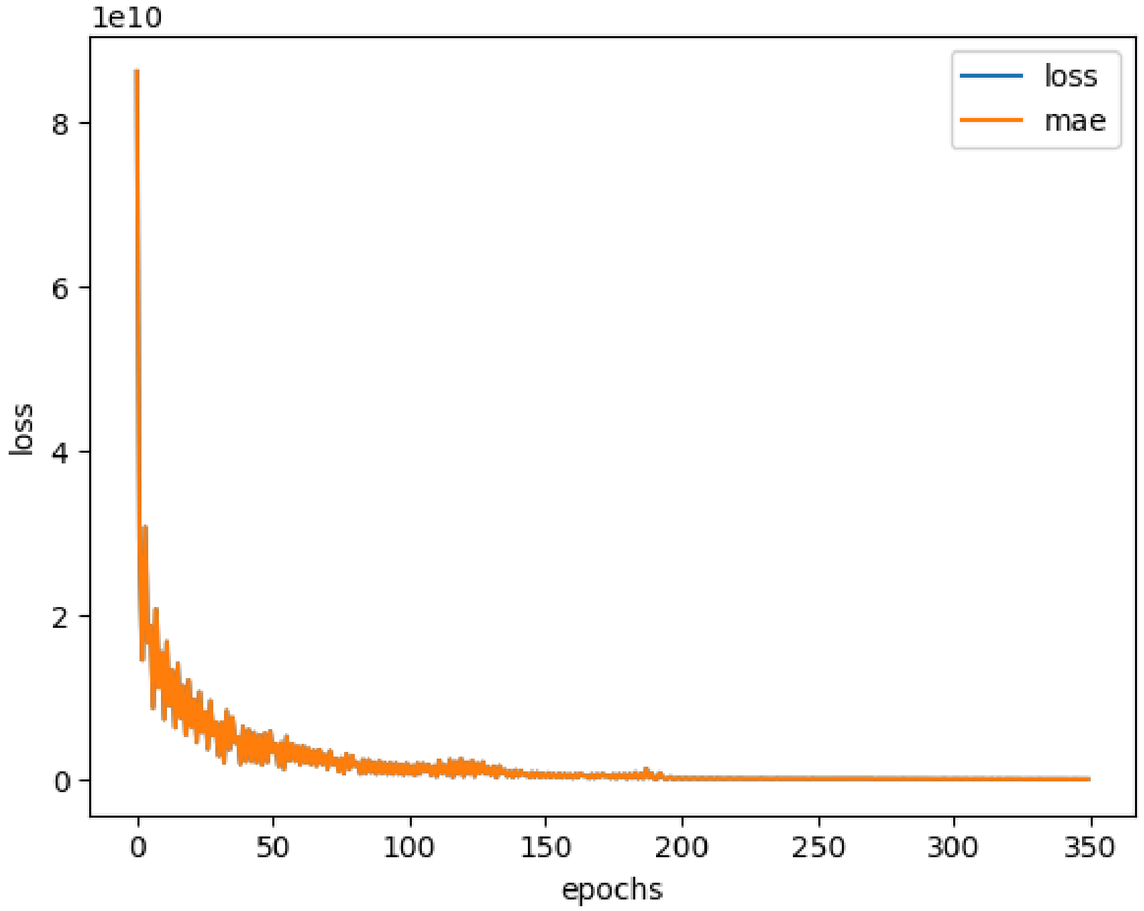
Model 1: Before VIF | Model 2: After VIF

Artificial Neural Networks



Train-Test	Architecture	Optimizer	Epochs	MAE
80-20	30-25-20-15-10-5-1	Adam	200	7.008
80-20	30-25-20-15-10-5-1	Adam	200	1.781
75-25	40-30-20-10-5-1	Adam	300	4.774
75-25	40-30-20-10-1	Adam	200	4.546
70-30	60-30-20-15-10-5-3-1	Adam	350	2.642
70-30	24-23-22-21-20-10-1	Adam	350	1.233
60-40	24-23-22-21-20-10-1	Adam	250	1.777
60-40	60-30-20-15-10-5-3-1	Adam	250	4.624

NEURAL NETWORK PLOT



Train-Test Split	**70-30**
Architecture	**24-23-22-21-20-10-1**
Optimizer	**Adam**
Epochs	**350**

SUMMARY

- **The main objective of this study was to build a model that predicts the infant mortality rate of countries based on multiple factors.**
- **From the above results, we can clearly see that in Model 1 the XGBoost has the best results.**
- **Whereas , in Model 2 the AdaBoost algorithm got the highest r2 score with least MAE.**

Insights

1. ***Significance of Infant Mortality Prediction:*** Infant mortality is a critical indicator of a nation's health, socio-economic conditions, and access to medical services**.**
2. ***Global Economic Factors:*** Key economic indicators like GDP, literacy rate, healthcare expenditure, poverty rates, and access to sanitation significantly influence infant mortality rates.
3. ***Model Insights***: Comparing models (e.g., regression, decision trees, neural networks) offers clarity on the trade-offs between interpretability and predictive accuracy

Future Scope

1. ***Expanded Features:*** Incorporating non-economic factors such as climate, geographical features, political stability, and cultural practices to enhance the model's accuracy.
2. ***Using real-time data sources:*** Such as satellite data or health reports, for dynamic predictions.
3. ***Integration with Healthcare Systems:*** Partnering with governments or NGOs to use predictive models for proactive resource allocation in vulnerable areas.

Work Distribution

M. BHARGAVI REDDY	Collecting basic information about medical insurance and Literature Review.
MOHAMMED AYAZ	Data preprocessing
K. ROHIT KUMAR	Exploratory Data Analysis
MD SAFWAN SHAWN	Applying Machine Learning Algorithms

Colab Link

THANK YOU

DONE BY

MD SAFWAN SHAWN
K. ROHIT KUMAR
MOHAMMED AYAZ
M. BHARGAVI REDDY

Appendix

Uploading the dataset

```
import pandas as pd
import numpy as np
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Insurance.csv to Insurance (1).csv
```

Double-click (or enter) to edit

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
df = pd.read_csv("/content/Insurance.csv")
df
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400

Data Preprocessing

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       1338 non-null   int64
 1   sex       1338 non-null   object
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64
 4   smoker    1338 non-null   object
 5   region    1338 non-null   object
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

```
for i in range(df.shape[1]):
  print(df.iloc[:,i].unique())
  print(df.iloc[:,i].value_counts())
```

```
[19 18 28 33 32 31 46 37 60 25 62 23 56 27 52 30 34 59 63 55 22 26 35 24
 41 38 36 21 48 40 58 53 43 64 20 61 44 57 29 45 54 49 47 51 42 50 39]
```

Checking data type

```
[1338 rows x 8 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   age         1338 non-null   int64
 1   sex         1338 non-null   object
 2   bmi         1338 non-null   float64
 3   children    1338 non-null   int64
 4   smoker      1338 non-null   object
 5   region      1338 non-null   object
 6   charges     1338 non-null   float64
 7   ChargesNew  1338 non-null   category
dtypes: category(1), float64(2), int64(2), object(3)
memory usage: 74.7+ KB
```

```
df=df.drop(['charges'],axis=1)
print(df)
```

```
df=df.drop(['charges'],axis=1)
print(df)
```

```
      age     sex     bmi  children smoker     region ChargesNew
0      19  female  27.900         0    yes  southwest          1
1      18    male  33.770         1     no  southeast          0
2      28    male  33.000         3     no  southeast          0
3      33    male  22.705         0     no  northwest          1
4      32    male  28.880         0     no  northwest          0
...   ...     ...     ...       ...    ...        ...        ...
1333   50    male  30.970         3     no  northwest          0
1334   18  female  31.920         0     no  northeast          0
1335   18  female  36.850         0     no  southeast          0
1336   21  female  25.800         0     no  southwest          0
1337   61  female  29.070         0    yes  northwest          1

[1338 rows x 7 columns]
```

```
x= df.drop(["ChargesNew"],axis=1)
y= df["ChargesNew"]
```

Dropping old columns

```
x=pd.get_dummies(x,dtype='int',drop_first=True)
print(x)
```

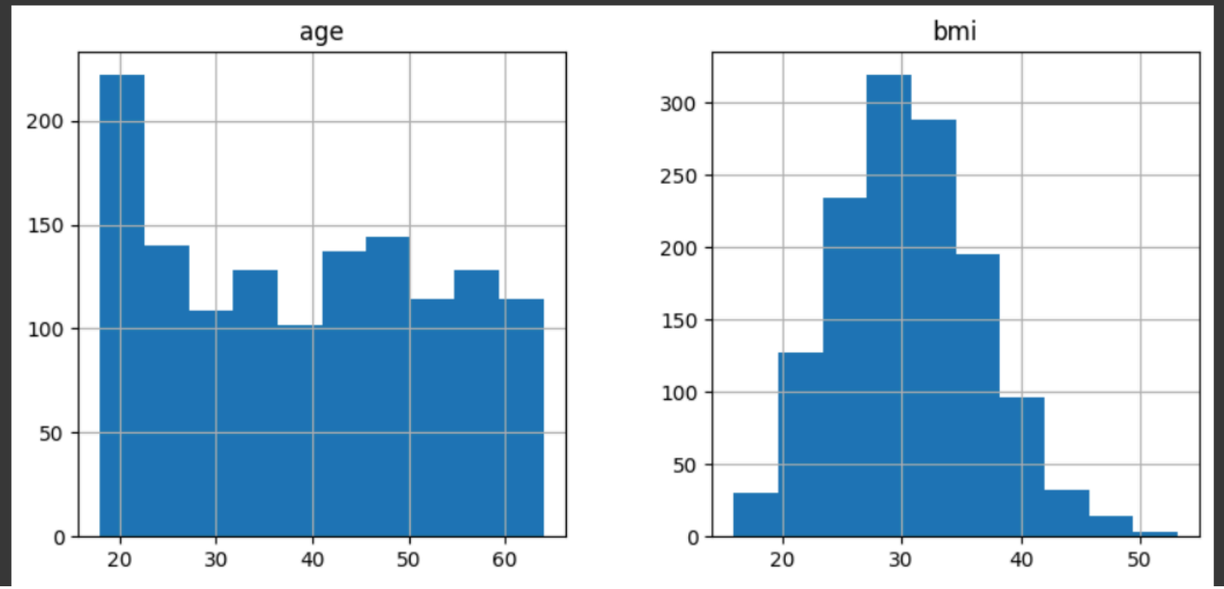
```
      age     bmi  children  sex_male  smoker_yes  region_northwest  \
0      19  27.900         0         0           1                 0
1      18  33.770         1         1           0                 0
2      28  33.000         3         1           0                 0
3      33  22.705         0         1           0                 1
4      32  28.880         0         1           0                 1
...   ...     ...       ...       ...         ...               ...
1333   50  30.970         3         1           0                 1
1334   18  31.920         0         0           0                 0
1335   18  36.850         0         0           0                 0
1336   21  25.800         0         0           0                 0
1337   61  29.070         0         0           1                 1

      region_southeast  region_southwest
0                    0                 1
1                    1                 0
2                    1                 0
3                    0                 0
4                    0                 0
...                ...               ...
1333                 0                 0
1334                 0                 0
1335                 1                 0
1336                 0                 1
1337                 0                 0
```

Getting dummies variables

```
[ ] import seaborn as sns
    import matplotlib.pyplot as plt
```

```
df.hist(figsize=(10,10))
plt.show()
```



Exploratory Data Analysis

Splitting the data

Splitting the **data**

```
import statsmodels.formula.api as smf
from sklearn import metrics
from sklearn.model_selection import train_test_split

X_train1, X_test1, y_train1, y_test1 = train_test_split(x, y, test_size=0.20, train_size=0.80,random_state=0)
X_train2, X_test2, y_train2, y_test2 = train_test_split(x, y, test_size=0.25, train_size=0.75,random_state=0)
X_train3, X_test3, y_train3, y_test3 = train_test_split(x, y, test_size=0.30, train_size=0.70,random_state=0)
X_train4, X_test4, y_train4, y_test4 = train_test_split(x, y, test_size=0.40, train_size=0.60,random_state=0)
```

80-20

```
[ ]  logreg = LogisticRegression(C=1e9)

     logreg.fit(X_train1, y_train1)
     predictions1 = logreg.predict(X_test1)
     print(predictions1)
```

```
['0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '0'
 '0' '1' '1' '0' '0' '1' '0' '0' '0' '0' '0' '0' '0' '1' '0' '0' '1' '0'
```

Linear Reg Before VIF

KNN Before VIF

80-20

```
[ ]  model=KNeighborsClassifier(n_neighbors=25)
     model.fit(X_train1, y_train1)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=25)
```

```
[ ]  y_pred1 = model.predict(X_test1)
     knn = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
     knn
```

	Predicted	Actual
578	0	0
610	0	0

80-20

```
[ ]  model1 = SVC(kernel='linear')
     model1.fit(X_train1, y_train1)
```

```
SVC
SVC(kernel='linear')
```

```
[ ]  y_pred1 = model1.predict(X_test1)
     svm = pd.DataFrame({'Predicted':y_pred1,'Actual':y_test1})
     print(svm)
```

```
      Predicted Actual
578           0      0
```

SVM Before VIF

80-20

```
dt=DecisionTreeClassifier()
dt.fit(X_train1,y_train1)
```

```
DecisionTreeClassifier
DecisionTreeClassifier()
```

```
[ ]  y_pred1=dt.predict(X_test1)
```

```
print("Accuracy:",accuracy_score(y_test1,y_pred1))
print(classification_report(y_test1, y_pred1))
```

```
Accuracy: 0.8544776119402985
              precision    recall  f1-score   support

           0       0.91      0.88      0.89       186
           1       0.74      0.80      0.77        82

    accuracy                           0.85       268
   macro avg       0.83      0.84      0.83       268
```

Decision Tree Before VIF

Random Forest

```
[ ] from sklearn.ensemble import RandomForestClassifier
```

```
[ ] rf=RandomForestClassifier()
```

80-20

```
[ ] rf=RandomForestClassifier()
    rf.fit(X_train1,y_train1)
```

```
RandomForestClassifier
RandomForestClassifier()
```

Bagging Before VIF

XGboost

```
[ ] import xgboost as xgb
```

80-20

```
[ ] model1 = xgb.XGBRegressor()
    model2 = xgb.XGBRegressor(n_estimators=100, max_depth=8, learning_rate=0.1, subsample=0.5

    train_model1 = model1.fit(X_train1, y_train1)
    train_model2 = model2.fit(X_train1, y_train1)
```

```
[ ] pred1 = train_model1.predict(X_test1)
```

Boosting Before VIF

Multicollinearity

```python
# Import library for VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor
import pandas as pd # Make sure pandas is imported

def calc_vif(x):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = x.columns
    vif["VIF"] = [variance_inflation_factor(x.values, i).round(1) for i in range(x.shape[1])]

    return(vif)

# Now call calc_vif with x_train:
calc_vif(x)
```

	variables	VIF
0	Density\n(P/Km2)	1.4
1	Agricultural Land(%)	1.4
2	Land Area(Km2)	2.0

Multicollinearity

Linear Regression

80-20

```
linreg = LinearRegression()
linreg.fit(x_nomulti_train1, y_nomulti_train1)
y_pred1 = linreg.predict(x_nomulti_test1)
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_nomulti_test1, y_pred1)))
print("R2 score:",metrics.r2_score(y_nomulti_test1,y_pred1))
```

```
RMSE: 7.861081162181842
R2 score: 0.893591831170952
```

75-25

Linear Reg After VIF

KNN After VIF

KNN

80-20

```
[ ] model=KNeighborsRegressor(n_neighbors=5)
model.fit(x_nomulti_train1, y_nomulti_train1)
y_pred1 = model.predict(x_nomulti_test1)
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_nomulti_test1, y_pred1)))
print("R2 score:",metrics.r2_score(y_nomulti_test1,y_pred1))
```

```
RMSE: 24.249806506196535
R2 score: -0.012574449124429732
```

Decision Tree

80-20

```
[ ]  reg1 = DecisionTreeRegressor()
     reg1.fit(x_nomulti_train1,y_nomulti_train1)
     y_pred1 = reg1.predict(x_nomulti_test1)
     print("RMSE:",np.sqrt(metrics.mean_squared_error(y_nomulti_test1, y_pred1)))
     print("R2 score:",metrics.r2_score(y_nomulti_test1,y_pred1))
```

```
RMSE: 9.422774488692069
R2 score: 0.8471138993922712
```

Decision Tree After VIF

Bagging After VIF

Random Forest

80-20

```
[ ]  rf.fit(x_nomulti_train1,y_nomulti_train1)
     y_pred1=rf.predict(x_nomulti_test1)
     print("RMSE:",np.sqrt(metrics.mean_squared_error(y_nomulti_test1, y_pred1)))
     print("R2 score:",metrics.r2_score(y_nomulti_test1,y_pred1))
```

```
RMSE: 8.131395236946897
R2 score: 0.8861480296142714
```

XGBoost

80-20

```
[ ]  model1 = xgb.XGBRegressor()
     model2 = xgb.XGBRegressor(n_estimators=100, max_depth=8, learning_rate=0.1, subsample=0.5)

     train_model1 = model1.fit(x_nomulti_train1, y_nomulti_train1)
     train_model2 = model2.fit(x_nomulti_train1, y_nomulti_train1)
```

```
[ ]  pred1 = train_model1.predict(x_nomulti_test1)
     pred2 = train_model2.predict(x_nomulti_test1)
     print("RMSE:",np.sqrt(metrics.mean_squared_error(y_nomulti_test1, pred1)))
     print("RMSE:",np.sqrt(metrics.mean_squared_error(y_nomulti_test1, pred2)))
     print("R2 score:",metrics.r2_score(y_nomulti_test1,pred1))
     print("R2 score:",metrics.r2_score(y_nomulti_test1,pred2))
```

```
RMSE: 7.843745205973438
RMSE: 8.24014560640801
R2 score: 0.8940606352152326
```

Boosting After VIF

SVM

```
[ ]  model = SVR(kernel='rbf')
```

80-20

```
model.fit(x_nomulti_train1, y_nomulti_train1)
y_pred1=model.predict(x_nomulti_test1)
print("R2 score:",metrics.r2_score(y_nomulti_test1,y_pred1))
print("RMSE:",np.sqrt(metrics.mean_squared_error(y_nomulti_t
```

```
R2 score: -0.18760862938312095
RMSE: 26.262221972629998
```

SVM After VIF

ANN

```
[ ]  import tensorflow as tf
```

```
tf.random.set_seed(42)

# STEP1: Creating the model

model= tf.keras.Sequential([
                            tf.keras.layers.Dense(30),
                            tf.keras.layers.Dense(25),
                            tf.keras.layers.Dense(20),
                            tf.keras.layers.Dense(15),
                            tf.keras.layers.Dense(10),



                            tf.keras.layers.Dense(5),


                            tf.keras.layers.Dense(1)
```

ANN After VIF

ANN

```
[ ]  import tensorflow as tf
```

80:20

```
tf.random.set_seed(42)

# STEP1: Creating the model

model= tf.keras.Sequential([
                            tf.keras.layers.Dense(30),
                            tf.keras.layers.Dense(25),
                            tf.keras.layers.Dense(20),
                            tf.keras.layers.Dense(15),
                            tf.keras.layers.Dense(10),



                            tf.keras.layers.Dense(5),
```

ANN After VIF