DEPARTMENT OF COMPUTER & SOFTWARE ENGINEERING

COLLEGE OF E&ME, NUST, RAWALPINDI

EC452 Machine Learning

Linear and Logistic Regression

SUBMITTED TO:

Prof Dr Asad Mansoor khan

SUBMITTED BY:

Uzair Waheed

Reg # 371828

DE- 43 CE

Submission Date:04/04/24

Tasks:

Single/Multiple/Polynomial Regression:

- *For this regression, you can use Life_expectancy as the target column. Your job, then, is to use different features to accurately predict this value. You are free to use any combination of features and any type of regression you see fit.*
- *Now, for all the features that you have selected in the above part, use the first 12 years data (2000 to 2012) to predict the remaining values of 3 years for those features. Use these predicted values instead of the actual values and discuss its effects on the results*

Solution:

Code:

```
import pandas as pd

from sklearn.preprocessing import StandardScaler
import copy
import numpy as np
import math
from gd import *
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import Lasso
# from sklearn.tree import DecisionTreeRegressor
import shap
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score


def shapImp(f_train, o_train):
    print("Wait for some time...")

    model = RandomForestRegressor()
    model = model.fit(f_train, o_train.values.ravel())

    explainer = shap.Explainer(model)
    shap_values = explainer(f_train)
    shap.plots.bar(shap_values)

    shap_imp = shap_values.values
```

```
    abs_imp = np.abs(shap_imp)
    mean_imp = abs_imp.mean(axis=0)
    shap_imp = pd.DataFrame({'Feature': f_train.columns, 'Importance':
mean_imp})
    sorted_values = shap_imp.sort_values('Importance', ascending=False)

    print(sorted_values)
    total = sorted_values['Importance'].sum()
    sorted_values['Cumulative'] = sorted_values['Importance'].cumsum() / total
    selected = []
    for i, row in sorted_values.iterrows():
        if row['Cumulative'] <= 0.95:
            selected.append(row['Feature'])
    print("Selected Features are:", selected)
    return selected


def featureImp(f_train,o_train):
    res = DecisionTreeRegressor()
    res.fit(f_train, o_train)
    feature_importances = res.feature_importances_
    feature_df = pd.DataFrame({"Feature": f_train.columns, "Importance":
feature_importances})
    sorted_values = feature_df.sort_values(by="Importance", ascending=False)
    print(sorted_values)
    plt.figure(figsize=(10, 5))
    plt.barh(feature_df["Feature"], feature_df["Importance"], color="skyblue")
    plt.xlabel("Feature Importance")
    plt.ylabel("Feature Name")
    plt.title("Feature Importance from Decision Tree")
    plt.show()

def plotPred(y_test,y_pred):
        # Create figure
    plt.figure(figsize=(10, 6))

    # Plot actual vs. predicted values
    plt.scatter(y_test, y_pred, alpha=0.5, label='Predicted vs Actual')
    # plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r-
-', label='Perfect Fit')
    plt.scatter(y_test,y_test,alpha =0.5,label = 'Perfect Model')

    # Customize plot

    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.title('Actual vs. Predicted Values')
    plt.legend()
```

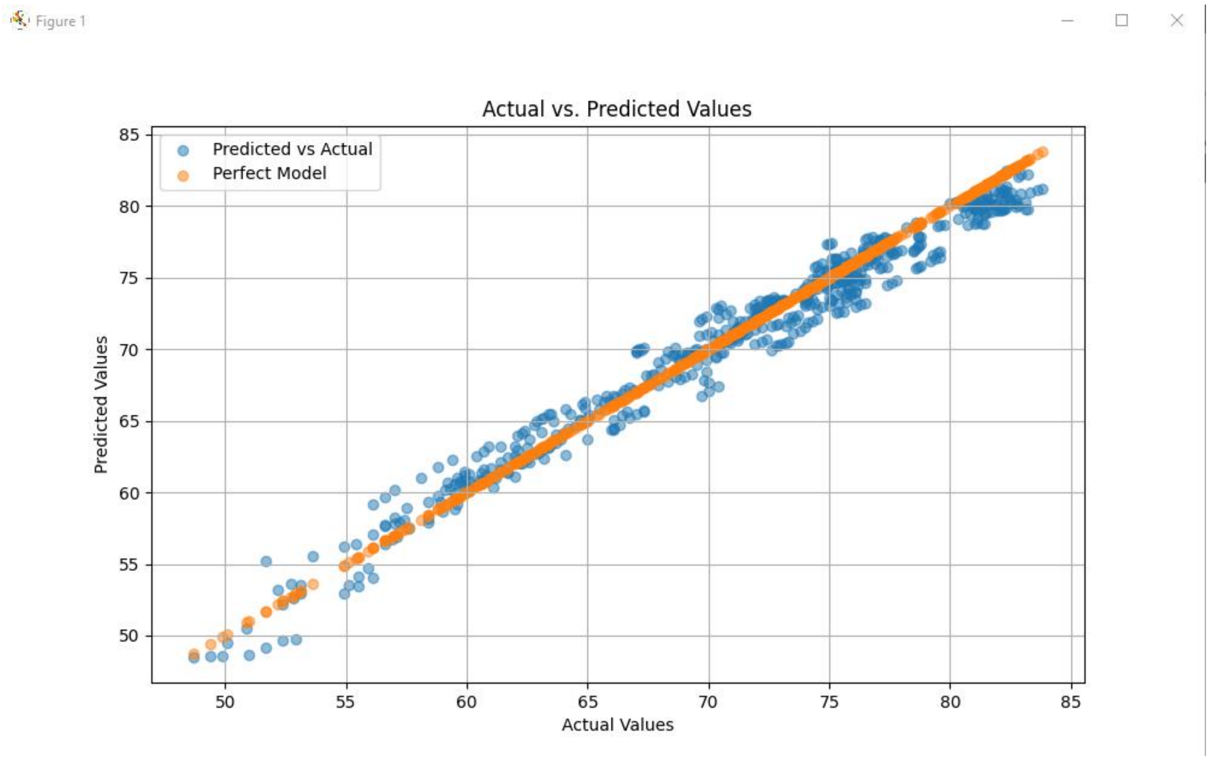
```
    plt.grid(True)
    plt.show()

def split(feature,output):
    #features
    train_data = []
    test_data = []
    for index,row in feature.iterrows():
        if row["Year"] < 2013:
            train_data.append(row.values)
        else:
            test_data.append(row.values)
    #output
    train_out = []
    test_out = []
    for index,row in output.iterrows():
        if row["Year"] <= 2012:
            train_out.append(row.values)
        else:
            test_out.append(row.values)

    f_train = pd.DataFrame(train_data, columns=feature.columns)
    f_train = f_train.drop(columns=["Year"])
    f_test = pd.DataFrame(test_data, columns=feature.columns)
    f_test = f_test.drop(columns=["Year"])
    o_train = pd.DataFrame(train_out, columns=output.columns)
    o_train = o_train.drop(columns=["Year"])
    o_test = pd.DataFrame(test_out, columns=output.columns)
    o_test = o_test.drop(columns=["Year"])
    return f_train,f_test,o_train,o_test

dataset = pd.read_csv("Life-Expectancy-Data-Updated.csv")
feature = dataset.drop(columns=["Life_expectancy","Country","Region"])
print(feature.columns)
output = dataset[["Life_expectancy","Year"]]

x_train,x_test,y_train,y_test = split(feature,output)
feature = feature.drop(columns=["Year"])
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train,columns = feature.columns)

x_test = scaler.transform(x_test)
x_test = pd.DataFrame(x_test,columns = feature.columns)

slcted_features = shapImp(x_train,y_train)
x_train = x_train[slcted_features]
x_test = x_test[slcted_features]
```

```
print("after feature selection : ", x_train)

######## gd without using lib #########
# initial_w = np.ones(x_train.shape[1])
# initial_b = 0.0
# w, b, cost_hist, w_hist = gradient_descent(
#     x_train,
#     y_train.values.ravel() if hasattr(y_train, 'values') else
y_train.ravel(),
#     initial_w,
#     initial_b,
#     compute_cost,
#     compute_gradient,
#     alpha=0.01,
#     num_iters=5000
# )
# print("Wj",w)
# print("b",b)
# y_pred = np.dot(x_test, w) + b
# plotPred(y_test.values,y_pred)
# cost = mean_squared_error(y_test, y_pred)
# print(f"Final cost (MSE): {cost:.2f}")

# ######## gd using lib #########
# gd = SGDRegressor(max_iter=5000)
# gd.fit(x_train, y_train)
# w = gd.coef_
# b = gd.intercept_[0]
# print("Wj",w)
# print("b",b)
# y_pred = gd.predict(x_test)
# plotPred(y_test.values,y_pred)
# cost = mean_squared_error(y_test, y_pred)
# print(f"Final cost (MSE): {cost:.2f}")

######## linear regression model(skl) #########
model = LinearRegression()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
plotPred(y_test.values,y_pred)
cost = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"R-squared: {r2:.2f}")
print(f"Final cost (MSE): {cost:.2f}")


######## polynomial regression #########
poly = PolynomialFeatures(degree=4, include_bias=False)
```

```
X_train_poly = poly.fit_transform(x_train.values)
model = Lasso(alpha=0.01)
model.fit(X_train_poly, y_train)
X_test_poly = poly.transform(x_test.values)
y_pred = model.predict(X_test_poly)
plotPred(y_test.values, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"R-squared: {r2:.2f}")

cost = mean_squared_error(y_test, y_pred)
print(f"Final cost (MSE): {cost:.2f}")
```

Output:

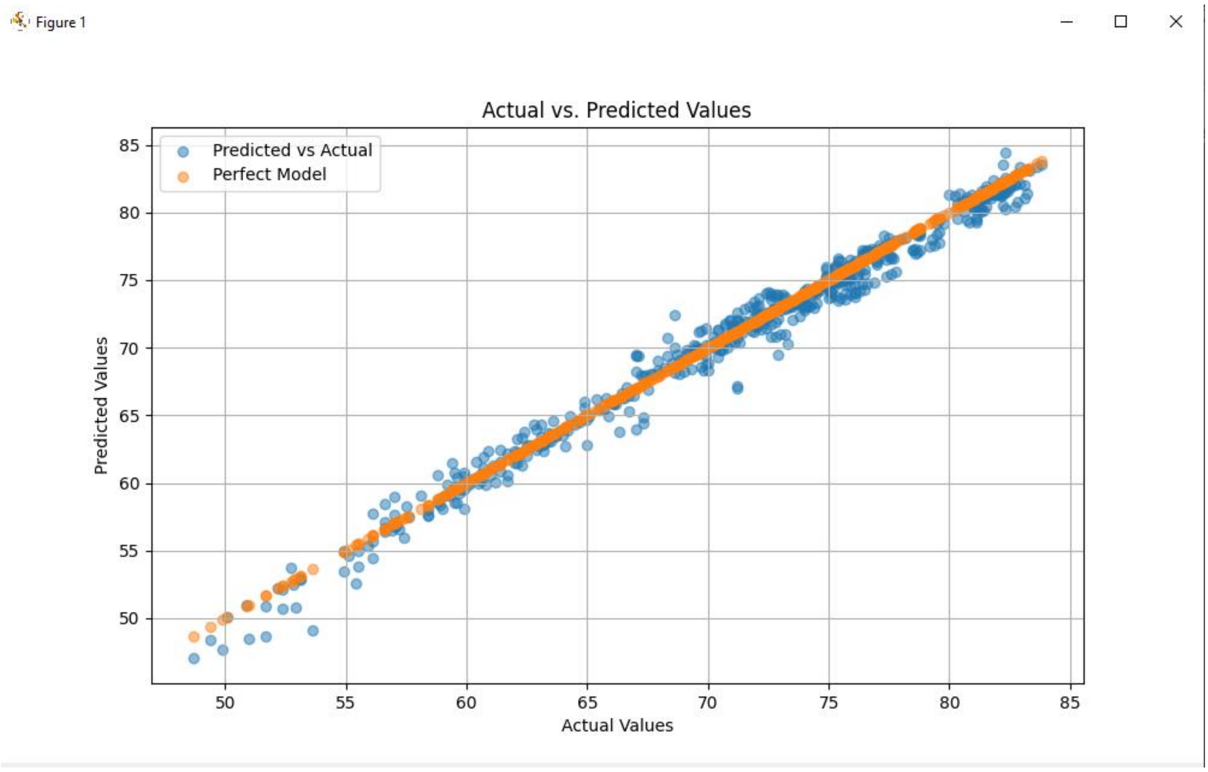
- *Feature dropped are country and region*
  Multiple regression:




```
R-squared: 0.97
Final cost (MSE): 2.07
```

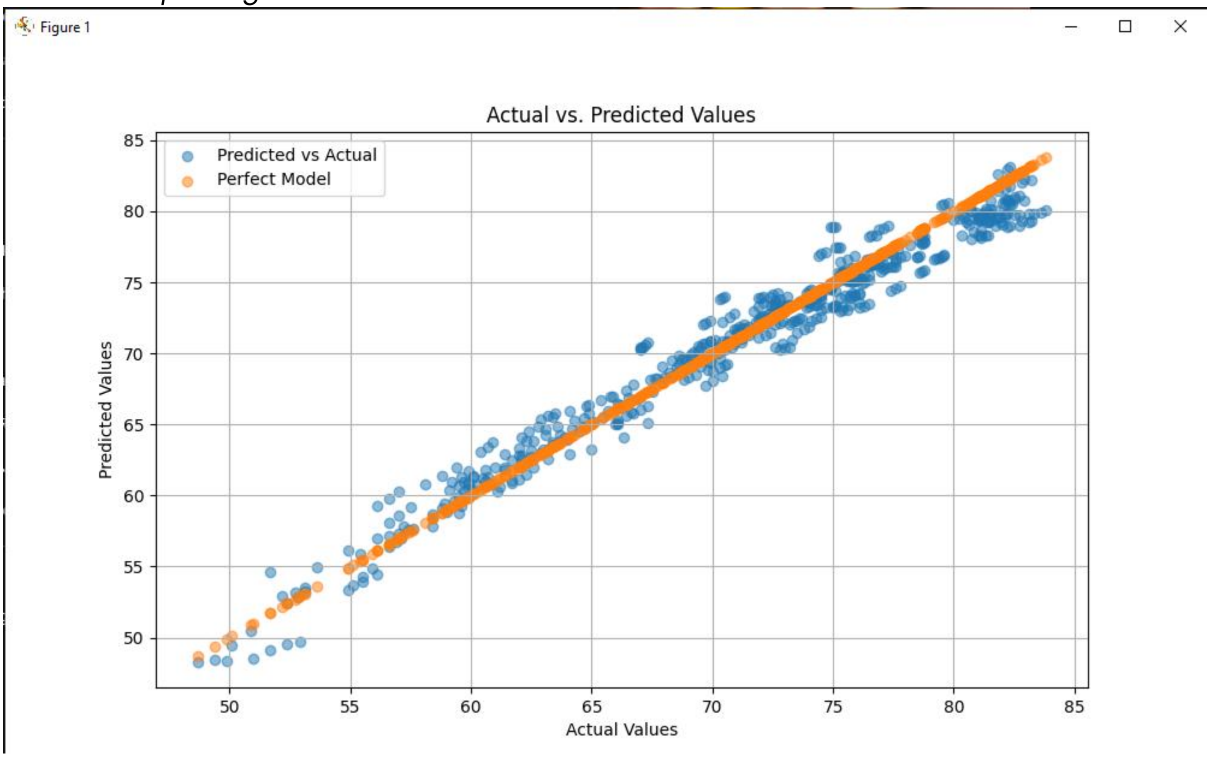
Polynomial regression:

Degree-4 if we incr degree the cost will start increasing.




```
R-squared: 0.98
Final cost (MSE): 1.09
```

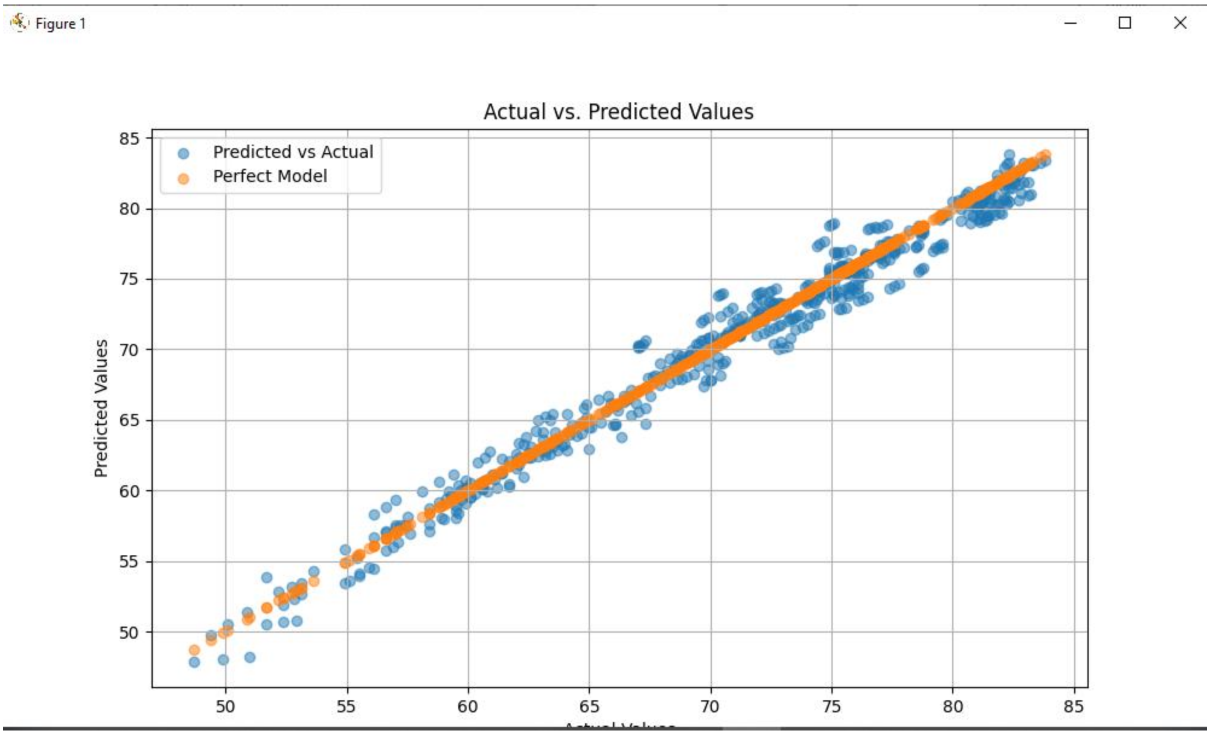
- *Feature Selected are using shap analysis (using randomForestRegressor) Multiple Regression*




```
R-squared: 0.96
Final cost (MSE): 2.35
```

Polynomial regression

Degree-4




```
R-squared: 0.97
Final cost (MSE): 1.64
```

Logistic Regression:

- *For Logistic Regression, you can use the columns Economy_status_Developed, Economy_status_Developing as your target column. Repeat both the experiments performed for Single/Multiple/Polynomial Regression.*

Code:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegressionCV
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.preprocessing import PolynomialFeatures
import shap
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LogisticRegression
```

```
def shapImp(f_train, o_train):
    print("Wait for some time...")

    model = RandomForestRegressor()
    model = model.fit(f_train, o_train.values.ravel())

    explainer = shap.Explainer(model)
    shap_values = explainer(f_train)
    shap.plots.bar(shap_values)

    shap_imp = shap_values.values
    abs_imp = np.abs(shap_imp)
    mean_imp = abs_imp.mean(axis=0)
    shap_imp = pd.DataFrame({'Feature': f_train.columns, 'Importance':
mean_imp})
    sorted_values = shap_imp.sort_values('Importance', ascending=False)

    print(sorted_values)
    total = sorted_values['Importance'].sum()
    sorted_values['Cumulative'] = sorted_values['Importance'].cumsum() / total
    selected = []
    for i, row in sorted_values.iterrows():
        if row['Cumulative'] <= 0.95:
            selected.append(row['Feature'])
    print("Selected Features are:", selected)
    return selected

def log_reg(f_train,f_test,o_train,o_test):
    # model = LogisticRegressionCV(penalty='l1', solver='saga',
max_iter=10000)
    model = LogisticRegression()
    model.fit(f_train,o_train)
    o_pred = model.predict(f_test)
    acc = accuracy_score(o_test,o_pred)
    print(f"Accuracy: {acc:.2f}")
    print("Classification Report:")
    print(classification_report(o_test, o_pred))


def split(feature,output):
    #features
    train_data = []
    test_data = []
    for index,row in feature.iterrows():
        if row["Year"] < 2013:
            train_data.append(row.values)
        else:
            test_data.append(row.values)
    #output
```

```
    train_out = []
    test_out = []
    for index,row in output.iterrows():
        if row["Year"] <= 2012:
            train_out.append(row.values)
        else:
            test_out.append(row.values)

    f_train = pd.DataFrame(train_data, columns=feature.columns)
    f_train = f_train.drop(columns=["Year"])
    f_test = pd.DataFrame(test_data, columns=feature.columns)
    f_test = f_test.drop(columns=["Year"])
    o_train = pd.DataFrame(train_out, columns=output.columns)
    o_train = o_train.drop(columns=["Year"])
    o_test = pd.DataFrame(test_out, columns=output.columns)
    o_test = o_test.drop(columns=["Year"])
    return f_train,f_test,o_train,o_test

dataset = pd.read_csv("Life-Expectancy-Data-Updated.csv")

#for developed country
feature = dataset.drop(columns =
['Economy_status_Developed','Economy_status_Developing'])
output = dataset[['Economy_status_Developed','Year']]
# feature = pd.get_dummies(feature, columns=['Country', 'Region'],
drop_first=True)
# print(feature)
# print(feature.head())
# print(feature.iloc[0])
# print(f"Number of features: {feature.shape[1]}")
feature = feature.drop(columns=['Country','Region'])
# print(f"Number of features: {feature.shape[1]}")

x_train,x_test,y_train,y_test = split(feature,output)
pol = PolynomialFeatures(degree=2, include_bias=False)

feature = feature.drop(columns=['Year'])
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train,columns = feature.columns)
x_test = scaler.transform(x_test)
x_test = pd.DataFrame(x_test,columns = feature.columns)

slcted_features = shapImp(x_train,y_train)
x_train = x_train[slcted_features]
x_test = x_test[slcted_features]
print(", ".join(slcted_features))
```

```
x_train_pol = pol.fit_transform(x_train)
x_test_pol = pol.transform(x_test)


print("For developed Country")
print("Multiple Logistic Regression")
log_reg(x_train,x_test,y_train,y_test)
print("Polynomial Logistic Regression")
log_reg(x_train_pol,x_test_pol,y_train,y_test)

#for developing country
feature = dataset.drop(columns =
['Economy_status_Developing','Economy_status_Developed'])
output = dataset[['Economy_status_Developing','Year']]
# feature = pd.get_dummies(feature, columns=['Country', 'Region'],
drop_first=True)
# print(feature)
# print(feature.head())
# print(feature.iloc[0])
# print(f"Number of features: {feature.shape[1]}")
feature = feature.drop(columns=['Country','Region'])
# print(f"Number of features: {feature.shape[1]}")

x_train,x_test,y_train,y_test = split(feature,output)

feature = feature.drop(columns=['Year'])
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_train = pd.DataFrame(x_train,columns = feature.columns)
x_test = scaler.transform(x_test)
x_test = pd.DataFrame(x_test,columns = feature.columns)

slcted_features = shapImp(x_train,y_train)
x_train = x_train[slcted_features]
x_test = x_test[slcted_features]
print(", ".join(slcted_features))
x_train_pol = pol.fit_transform(x_train)
x_test_pol = pol.transform(x_test)

# print("For developing Country")
print("Multiple Logistic Regression")
log_reg(x_train,x_test,y_train,y_test)
print("Polynomial Logistic Regression")
log_reg(x_train_pol,x_test_pol,y_train,y_test)
```

Output:

For Developed Countries

Feature dropped are country and region

Multiple Logistic regression:

```
Accuracy: 0.98
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.99       426
           1       0.90      1.00      0.95       111

    accuracy                           0.98       537
   macro avg       0.95      0.99      0.97       537
weighted avg       0.98      0.98      0.98       537
```

Polynomial regression:

```
Accuracy: 0.98
Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       426
           1       0.96      0.96      0.96       111

    accuracy                           0.98       537
   macro avg       0.97      0.98      0.97       537
weighted avg       0.98      0.98      0.98       537
```

Feature Selected are using shap analysis (using randomForestRegressor)

Selected features

Infant_deaths, Alcohol_consumption, Under_five_deaths, GDP_per_capita, Schooling, Incidents_HIV, Life_expectancy

Multiple Regression

```
Accuracy: 0.97
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.97      0.98       426
           1       0.89      1.00      0.94       111

    accuracy                           0.97       537
   macro avg       0.94      0.98      0.96       537
weighted avg       0.98      0.97      0.97       537
```

Polynomial regression

```
Accuracy: 0.99
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       426
           1       0.94      1.00      0.97       111

    accuracy                           0.99       537
   macro avg       0.97      0.99      0.98       537
weighted avg       0.99      0.99      0.99       537
```

For Developing Countries

Feature dropped are country and region

Multiple Logistic regression:

```
Accuracy: 0.98
Classification Report:
              precision    recall  f1-score   support

           0       0.90      1.00      0.95       111
           1       1.00      0.97      0.99       426

    accuracy                           0.98       537
   macro avg       0.95      0.99      0.97       537
weighted avg       0.98      0.98      0.98       537
```

Polynomial regression:

```
Accuracy: 0.99
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.96      0.96       111
           1       0.99      0.99      0.99       426

    accuracy                           0.99       537
   macro avg       0.98      0.98      0.98       537
weighted avg       0.99      0.99      0.99       537
```

Feature Selected are using shap analysis (using randomForestRegressor)

Selected features

Infant_deaths, Alcohol_consumption, Under_five_deaths, GDP_per_capita, Schooling, Incidents_HIV, Life_expectancy

Multiple Regression

```
Accuracy: 0.97
Classification Report:
              precision    recall  f1-score   support

           0       0.87      1.00      0.93       111
           1       1.00      0.96      0.98       426

    accuracy                           0.97       537
   macro avg       0.94      0.98      0.96       537
weighted avg       0.97      0.97      0.97       537
```

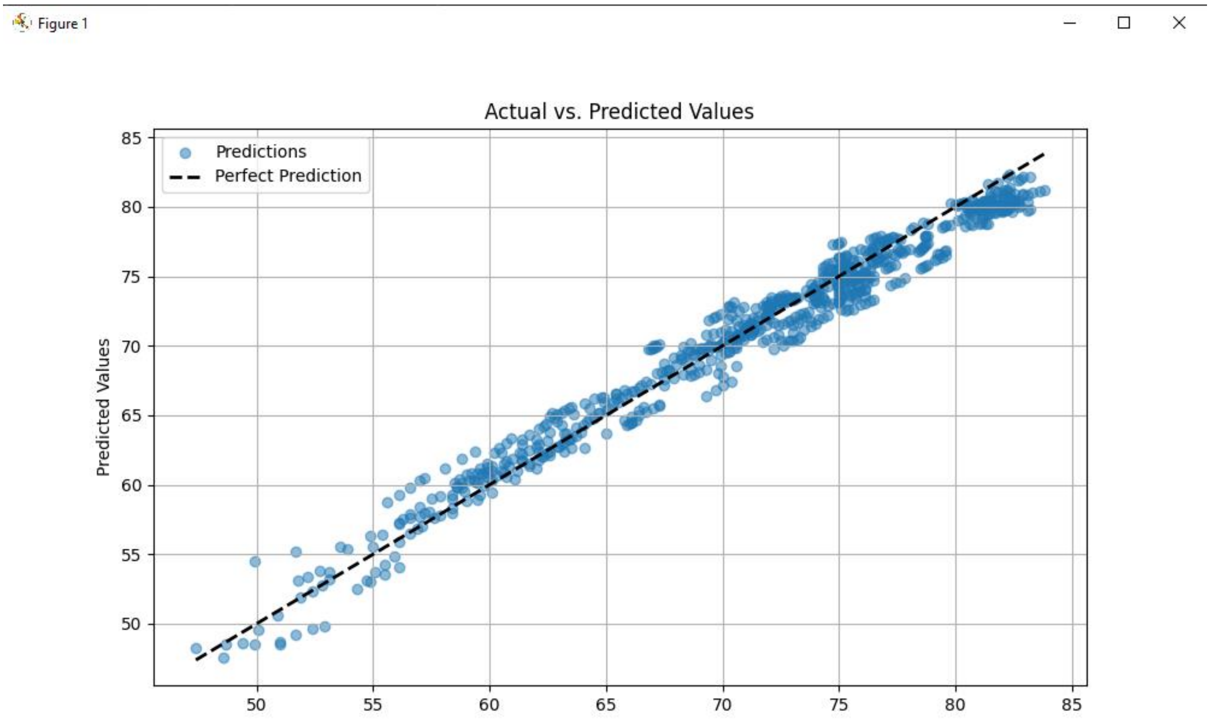
Polynomial regression

```
Accuracy: 0.97
Classification Report:
              precision    recall  f1-score   support

           0       0.87      1.00      0.93       111
           1       1.00      0.96      0.98       426

    accuracy                           0.97       537
   macro avg       0.93      0.98      0.95       537
weighted avg       0.97      0.97      0.97       537
```

TESTING:

Features dropped are Country, Region, Population_mln.

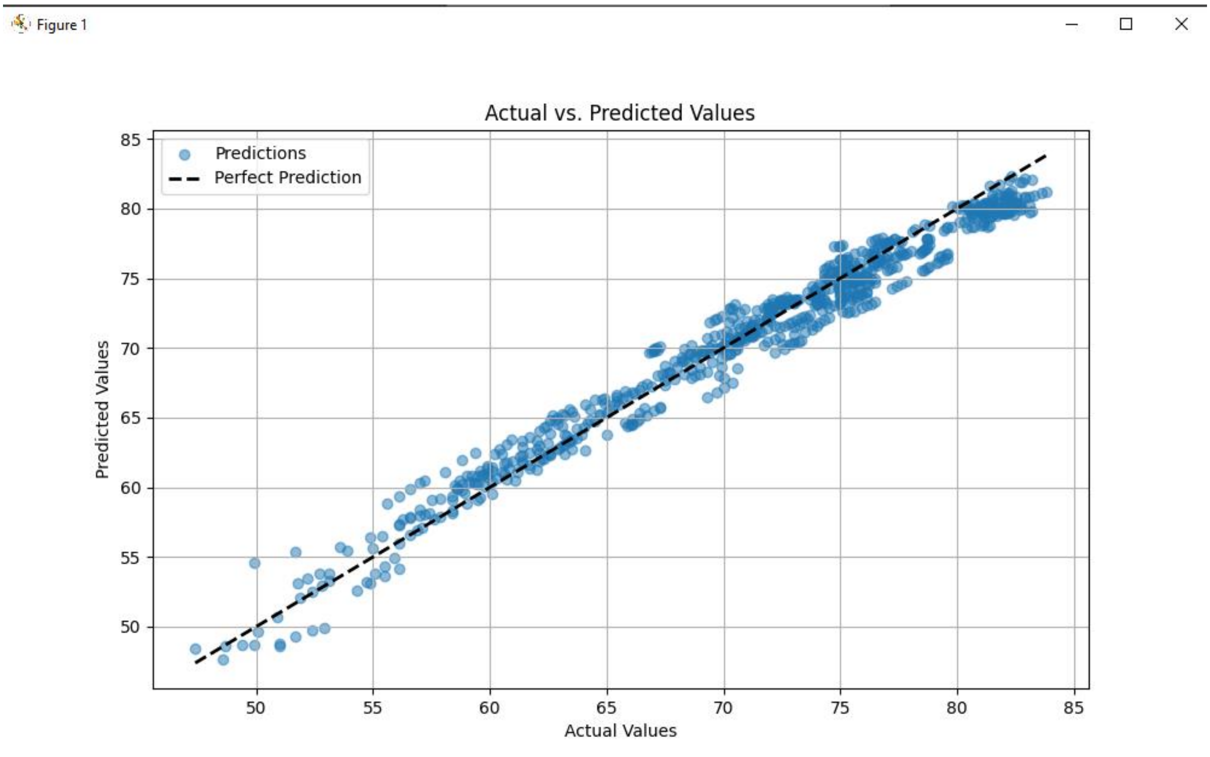
Not using any lib for gradient descent:



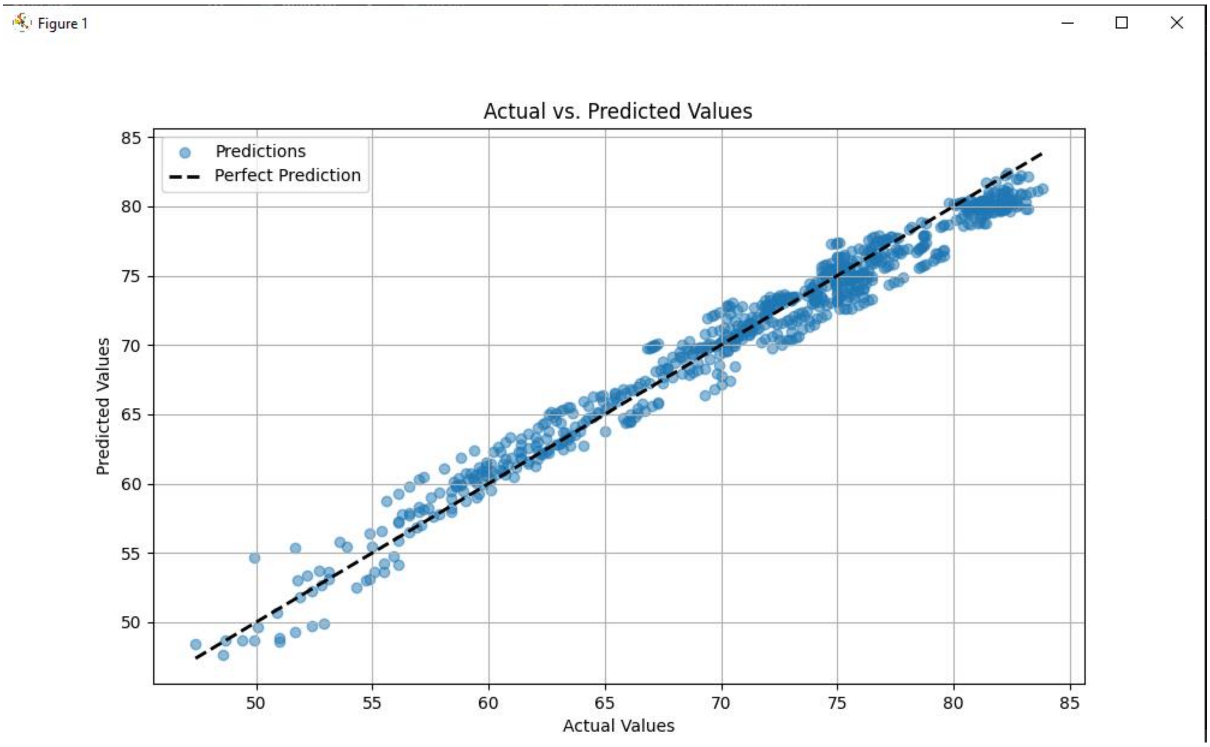
```
Wj [ 4.69611941e-02 -1.56534023e+00 -2.29888472e+00 -5.43808805e+00
  2.61174220e-01 -1.49676760e-01 -4.23465678e-03 -3.41263407e-01
  3.01121520e-02  5.21523685e-02  1.99270258e-01  3.97769989e-01
 -2.07875519e-01  3.88974236e-02  2.64338659e-01  1.10998746e+00
  8.90012542e-01]
b 68.80021268791072
Final cost (MSE): 2.0625
```

using sickit learn for gradient descent:

```
Wj [ 5.70328172e-02 -1.65366278e+00 -2.20555220e+00 -5.40183554e+00
  2.65046878e-01 -1.40700507e-01 -5.14296288e-03 -3.53062513e-01
  9.80474333e-02 -1.46320200e-02  2.11566222e-01  3.94533716e-01
 -1.74079134e-01  3.22857429e-02  2.65373435e-01  1.11202737e-01
 -1.11202737e-01]
b 68.79696494286677
Final cost (MSE): 2.1081
```



Multi regerssion from sklearn :




```
PS E:\Uni\Sem 8\Machine Learning\Assignment # 2> python main.py
Final cost (MSE): 2.0690
```

Feature imp using Decision trees :

	Feature	Importance
1	Under_five_deaths	0.742639
2	Adult_mortality	0.212515
0	Infant_deaths	0.025203
9	Incidents_HIV	0.005168
13	Schooling	0.002246
6	BMI	0.002090
3	Alcohol_consumption	0.002041
11	Thinness_ten_nineteen_years	0.001690
10	GDP_per_capita	0.001598
14	Economy_status_Developed	0.001408
12	Thinness_five_nine_years	0.001198
5	Measles	0.000789
7	Polio	0.000668
4	Hepatitis_B	0.000500
8	Diphtheria	0.000245
15	Economy_status_Developing	0.000002

Feature imp using SHAP(random trees)

	Feature	SHAP_Importance
1	Under_five_deaths	5.405550
2	Adult_mortality	3.091080
0	Infant_deaths	0.476691
10	GDP_per_capita	0.143851
13	Schooling	0.129400
3	Alcohol_consumption	0.124964
9	Incidents_HIV	0.114010
6	BMI	0.070457
11	Thinness_ten_nineteen_years	0.053750
12	Thinness_five_nine_years	0.053717
5	Measles	0.036139
7	Polio	0.025242
8	Diphtheria	0.021431
4	Hepatitis_B	0.021342
15	Economy_status_Developing	0.020857
14	Economy_status_Developed	0.018470

Using shap (95% of variance) without shap

1.Multiple regression

```
Final cost (MSE): 2.2425
```

```
Final cost (MSE): 2.1488
```

2.Polynomail regression

(2nd degree)

```
Final cost (MSE): 1.74
```

```
Final cost (MSE): 1.42
```

(3rd degree)

```
Final cost (MSE): 1.56
```

```
Final cost (MSE): 1.40
```

(4th degree)

```
model = cd_fast.enet_coo
Final cost (MSE): 1.54
```

```
Final cost (MSE): 1.21
```

(5th degree) after increasing the complexity of polynomial , now the cost is started to increase