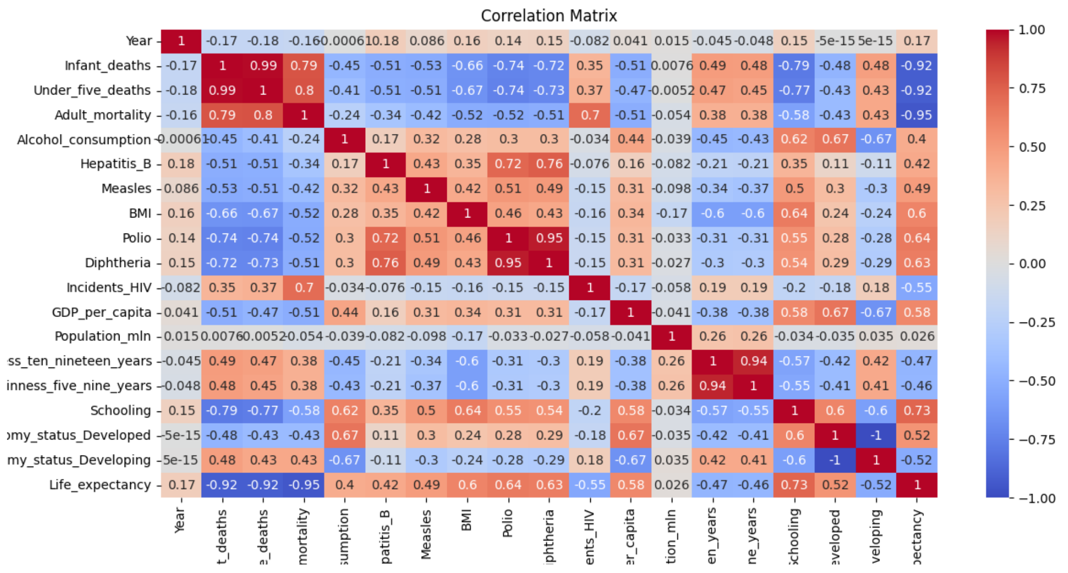
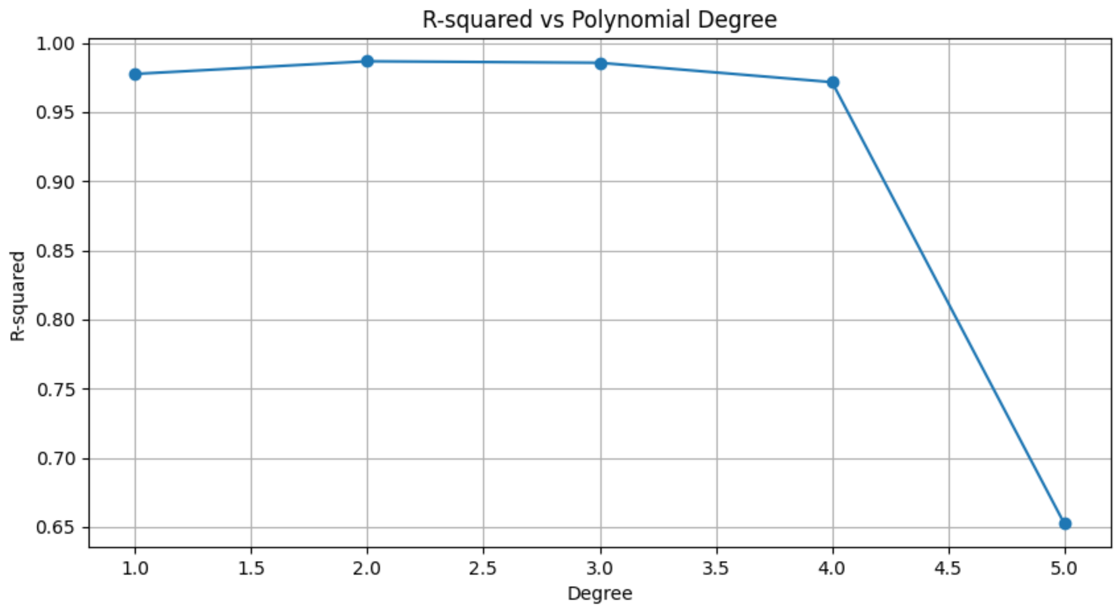
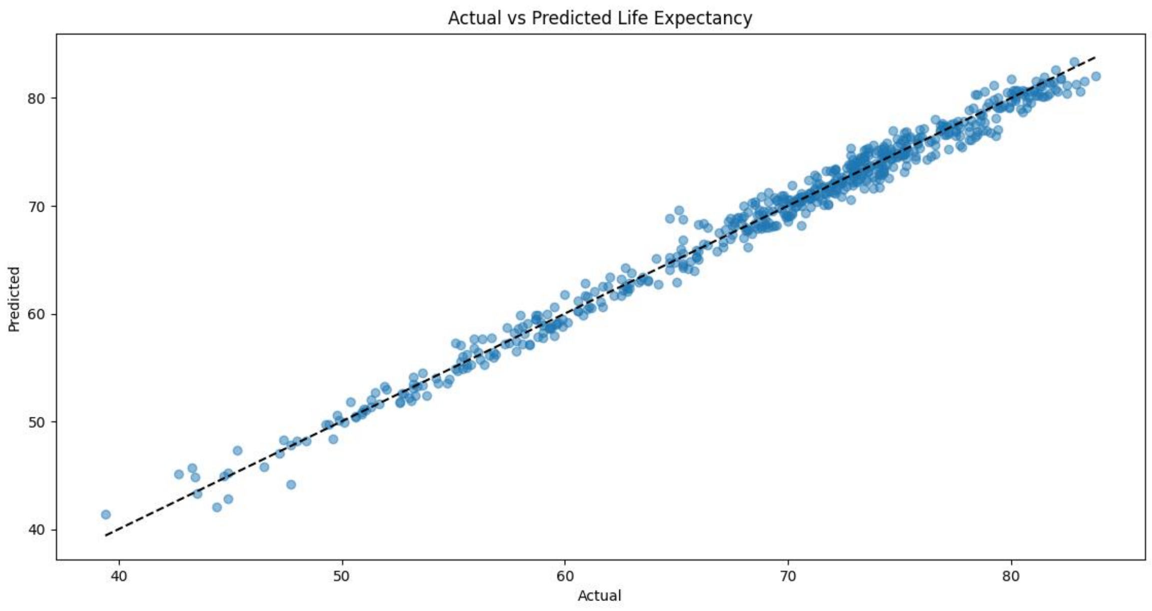
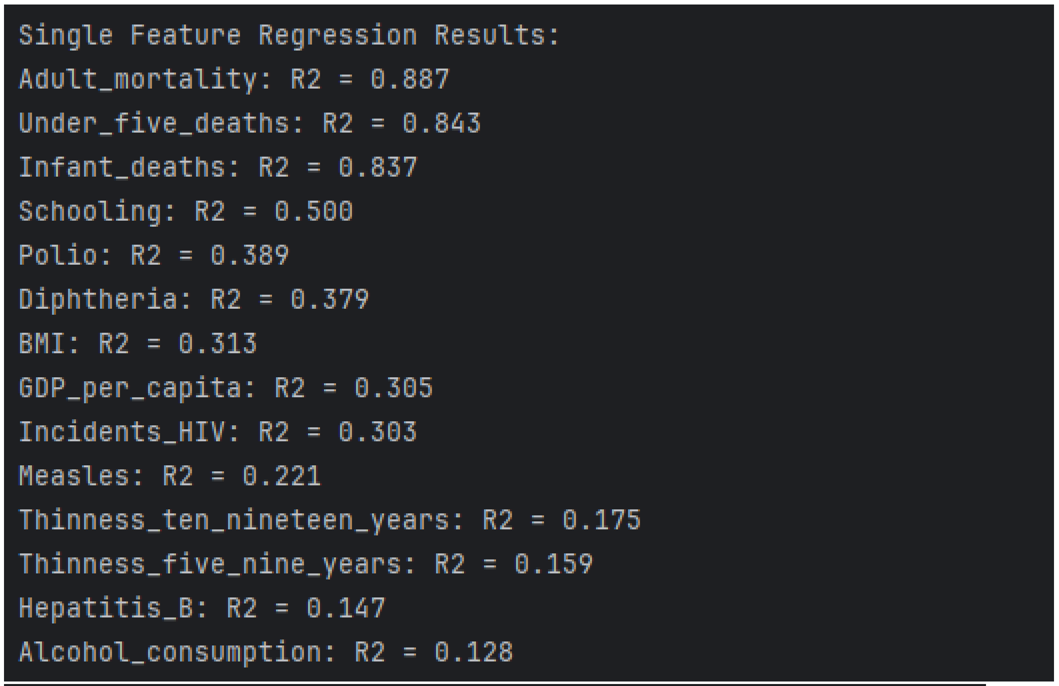
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Load data
df = pd.read_csv('C:/Users/Immad/Downloads/Life-Expectancy-Data-Updated.csv')



# Correlation analysis
corr_matrix = df.corr(numeric_only=True)
life_expectancy_corr =
corr_matrix['Life_expectancy'].sort_values(ascending=False)

# Visualize correlation matrix
plt.figure(figsize=(10, 6))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()

# Select top correlated features
top_features = life_expectancy_corr[abs(life_expectancy_corr) >
0.3].index.tolist()
top_features.remove('Life_expectancy')  # Remove target
top_features.remove('Economy_status_Developed')  # Remove target
top_features.remove('Economy_status_Developing')  # Remove target

print("Top correlated features with Life_expectancy:", top_features)
final_features = top_features

# Single feature regression
results = {}
for feature in final_features:
    X = df[[feature]]
    y = df['Life_expectancy']
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    r2 = r2_score(y_test, y_pred)
    results[feature] = r2

# Sort results by R2 score
sorted_results = sorted(results.items(), key=lambda x: x[1], reverse=True)
print("\nSingle Feature Regression Results:")
for feature, r2 in sorted_results:
    print(f"{feature}: R2 = {r2:.3f}")
best_single_feature = sorted_results[0][0]
print(f"\nBest single feature: {best_single_feature} with R2 =
{sorted_results[0][1]:.3f}")

# Multiple Linear Regression (MLR)
X = df[final_features]
y = df['Life_expectancy']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

mlr = LinearRegression()
mlr.fit(X_train, y_train)
y_pred = mlr.predict(X_test)

mlr_r2 = r2_score(y_test, y_pred)
mlr_mse = mean_squared_error(y_test, y_pred)

print("\nMultiple Linear Regression Results:")
print(f"R-squared: {mlr_r2:.4f}")
print(f"MSE: {mlr_mse:.4f}")
print("\nCoefficients:")
for feature, coef in zip(final_features, mlr.coef_):
    print(f"{feature}: {coef:.4f}")
print(f"Intercept: {mlr.intercept_:.4f}")

# Polynomial Regression (degree=2)
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly_train = poly.fit_transform(X_train)
X_poly_test = poly.transform(X_test)

poly_model = LinearRegression()
poly_model.fit(X_poly_train, y_train)

y_pred = poly_model.predict(X_poly_test)
poly_r2 = r2_score(y_test, y_pred)
poly_mse = mean_squared_error(y_test, y_pred)

print("\nPolynomial Regression (degree=2) Results:")
print(f"R-squared: {poly_r2:.4f}")
print(f"MSE: {poly_mse:.4f}")

# Compare with previous linear results
print("\nModel Comparison:")
print(f"{'Model':<20} {'R-squared':<10} {'MSE':<10}")
print(f"{'Linear':<20} {mlr_r2:.4f} {mlr_mse:.4f}")
print(f"{'Polynomial (deg=2)':<20} {poly_r2:.4f} {poly_mse:.4f}")

# Calculate R-squared improvement
improvement = ((poly_r2 - mlr_r2) / mlr_r2) * 100
print(f"\nR-squared improvement: {improvement:.2f}%")

# Test different polynomial degrees
degrees = [1, 2, 3, 4, 5]
results = []
for degree in degrees:
    model = make_pipeline(
        PolynomialFeatures(degree=degree, include_bias=False),
        LinearRegression()
    )

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    r2 = r2_score(y_test, y_pred)
    mse = mean_squared_error(y_test, y_pred)
    results.append((degree, r2, mse))
    print(f"\nDegree {degree}: R2 = {r2:.4f}, MSE = {mse:.4f}")

# Find best polynomial degree
best_degree = max(results, key=lambda x: x[1])[0]
print(f"\nOptimal polynomial degree: {best_degree}")

# Plot R2 vs degree
degrees, r2_scores, mses = zip(*results)
plt.figure(figsize=(10, 5))
plt.plot(degrees, r2_scores, marker='o')
plt.title('R-squared vs Polynomial Degree')
plt.xlabel('Degree')
plt.ylabel('R-squared')
plt.grid(True)
plt.show()

# Plot actual vs predicted for the best model
best_model = make_pipeline(
    PolynomialFeatures(degree=best_degree, include_bias=False),
    LinearRegression()
)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)

plt.figure(figsize=(10, 5))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--')
plt.title('Actual vs Predicted Life Expectancy')
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.show()
```

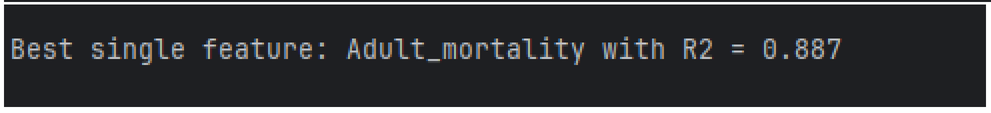

Correlation Matrix
Year
Infant_deaths
Under_five_deaths
Adult_mortality
Alcohol_consumption
Hepatitis_B
Measles
BMI
Polio
Diphtheria
Incidents_HIV
GDP_per_capita
Population_mln
ss_ten_nineteen_years
inness_five_nine_years
Schooling
omy_status_Developed
my_status_Developing
Life_expectancy
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


R-squared vs Polynomial Degree
R-squared
Degree
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

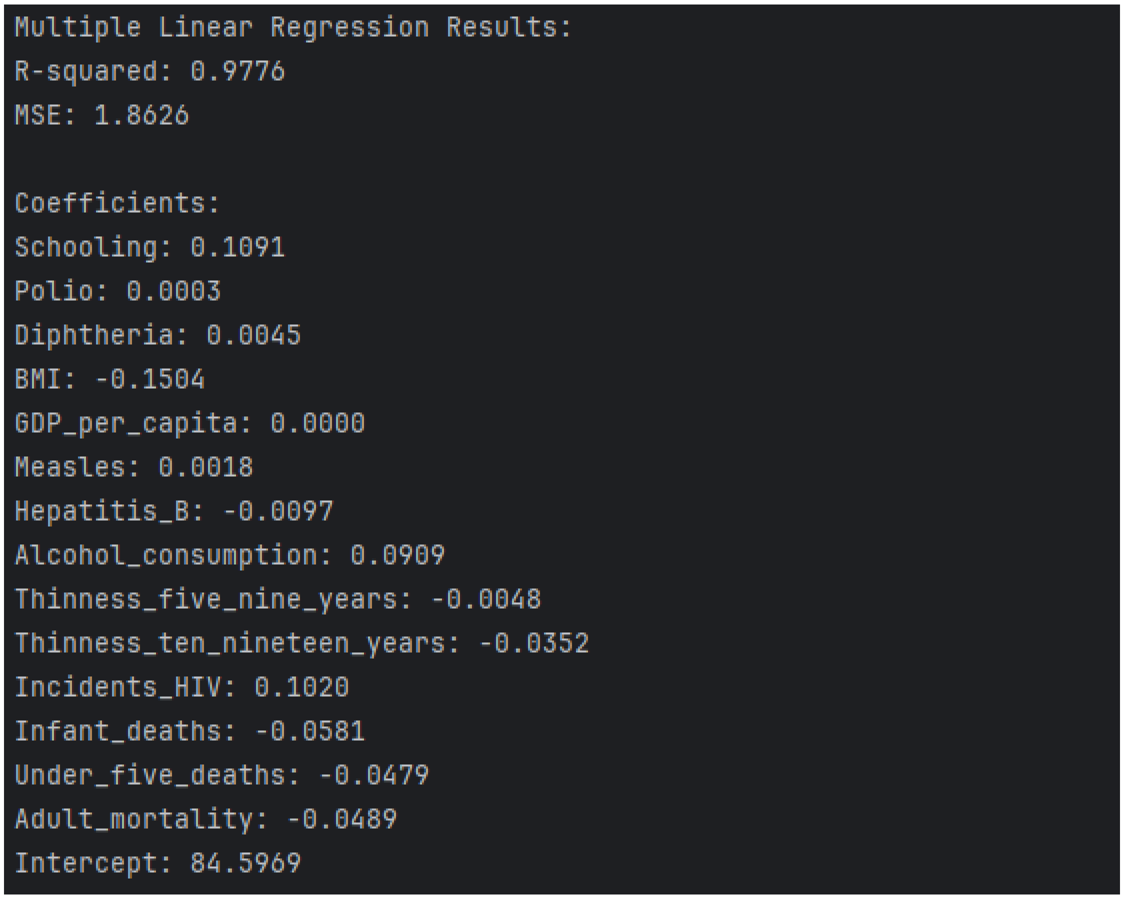




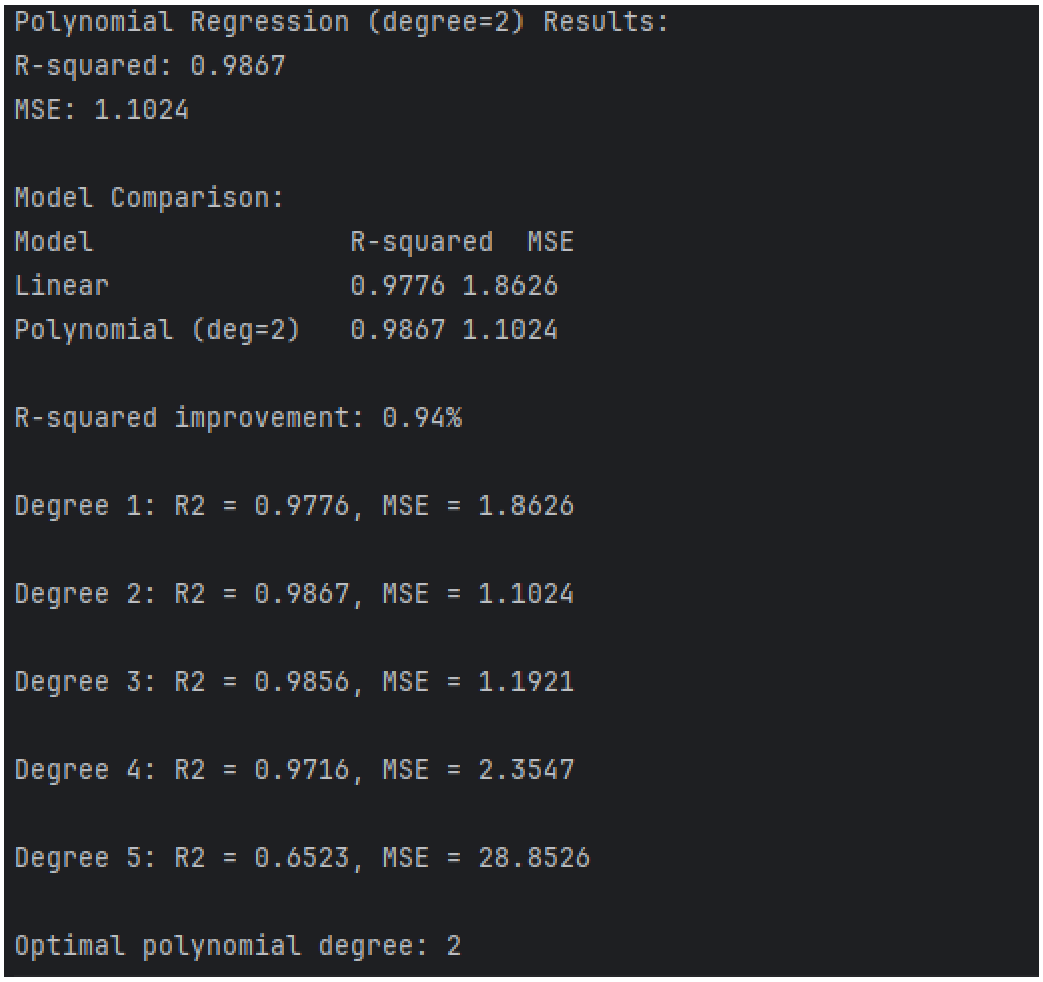
```
Single Feature Regression Results:
Adult_mortality: R2 = 0.887
Under_five_deaths: R2 = 0.843
Infant_deaths: R2 = 0.837
Schooling: R2 = 0.500
Polio: R2 = 0.389
Diphtheria: R2 = 0.379
BMI: R2 = 0.313
GDP_per_capita: R2 = 0.305
Incidents_HIV: R2 = 0.303
Measles: R2 = 0.221
Thinness_ten_nineteen_years: R2 = 0.175
Thinness_five_nine_years: R2 = 0.159
Hepatitis_B: R2 = 0.147
Alcohol_consumption: R2 = 0.128
```



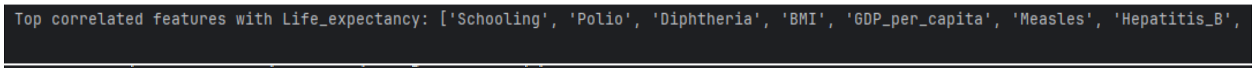
```
Best single feature: Adult_mortality with R2 = 0.887
```



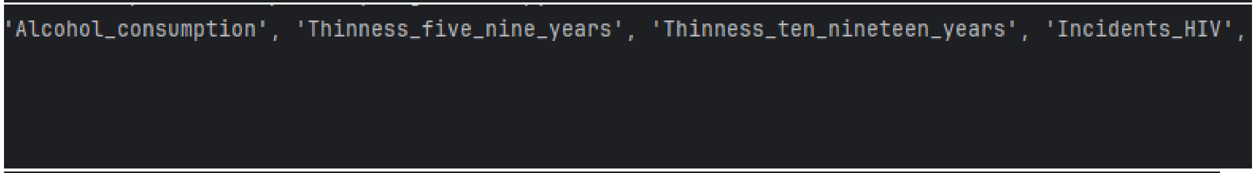
```
Multiple Linear Regression Results:
R-squared: 0.9776
MSE: 1.8626

Coefficients:
Schooling: 0.1091
Polio: 0.0003
Diphtheria: 0.0045
BMI: -0.1504
GDP_per_capita: 0.0000
Measles: 0.0018
Hepatitis_B: -0.0097
Alcohol_consumption: 0.0909
Thinness_five_nine_years: -0.0048
Thinness_ten_nineteen_years: -0.0352
Incidents_HIV: 0.1020
Infant_deaths: -0.0581
Under_five_deaths: -0.0479
Adult_mortality: -0.0489
Intercept: 84.5969
```



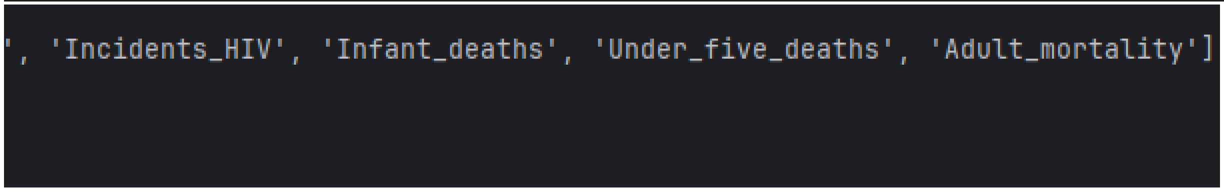
```
Polynomial Regression (degree=2) Results:
R-squared: 0.9867
MSE: 1.1024

Model Comparison:
Model                R-squared  MSE
Linear               0.9776 1.8626
Polynomial (deg=2)   0.9867 1.1024

R-squared improvement: 0.94%

Degree 1: R2 = 0.9776, MSE = 1.8626

Degree 2: R2 = 0.9867, MSE = 1.1024

Degree 3: R2 = 0.9856, MSE = 1.1921

Degree 4: R2 = 0.9716, MSE = 2.3547

Degree 5: R2 = 0.6523, MSE = 28.8526

Optimal polynomial degree: 2
```



```
Top correlated features with Life_expectancy: ['Schooling', 'Polio', 'Diphtheria', 'BMI', 'GDP_per_capita', 'Measles', 'Hepatitis_B',
```



```
'Alcohol_consumption', 'Thinness_five_nine_years', 'Thinness_ten_nineteen_years', 'Incidents_HIV',
```



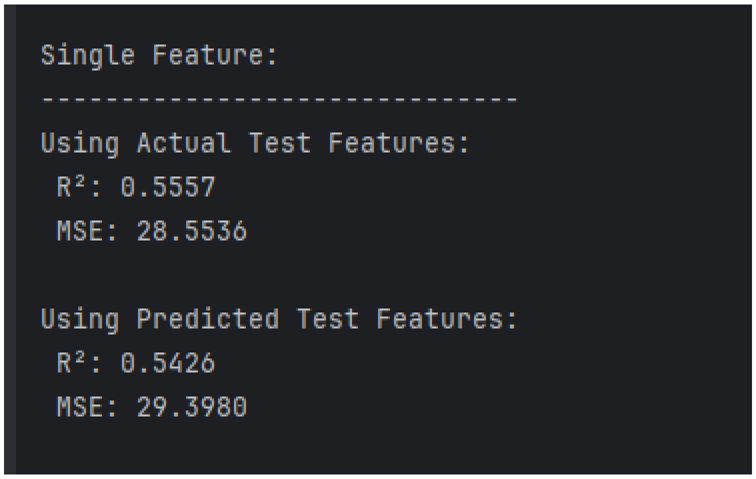
```
', 'Incidents_HIV', 'Infant_deaths', 'Under_five_deaths', 'Adult_mortality']
```

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score, mean_squared_error
```

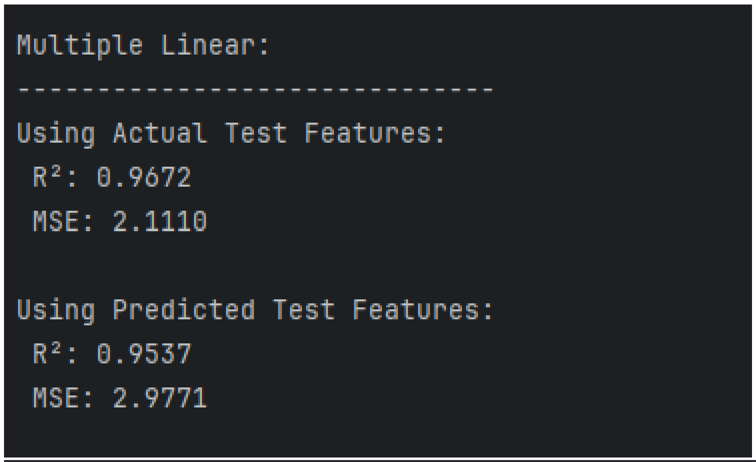
```python
import pandas as pd

df = pd.read_csv('C:/Users/Immad/Downloads/Life-Expectancy-Data-Updated.csv')



# Now, you can sort the dataframe
df = df.sort_values(['Country', 'Year'])


# 1. Prepare data
df = df.sort_values(['Country', 'Year'])  # Sort by country and year
train = df[df['Year'].between(2000, 2012)]
test = df[df['Year'].between(2013, 2015)]

# Define final_features (assuming it's previously defined or here)
final_features = ['Schooling', 'Polio', 'Diphtheria', 'BMI',
'GDP_per_capita', 'Measles', 'Hepatitis_B',
                  'Alcohol_consumption', 'Thinness_five_nine_years',
'Thinness_ten_nineteen_years',
                  'Incidents_HIV', 'Infant_deaths', 'Under_five_deaths',
'Adult_mortality']


# 2. Predict features for each country's 2013-2015 values
def predict_country_features(train_data, features):
    """Predict features for next years for each country"""
    all_predictions = []
    countries = train_data['Country'].unique()

    for country in countries:
        country_data = train_data[train_data['Country'] == country]
        if len(country_data) < 3:  # Need at least 3 points for trend
            continue

        predictions = {'Country': country}
        for feature in features:
            X = country_data['Year'].values.reshape(-1, 1)
            y = country_data[feature].values
            model = LinearRegression()
            model.fit(X, y)
            future_years = np.array([2013, 2014, 2015]).reshape(-1, 1)
            predictions[feature] = model.predict(future_years)

        # Create DataFrame for this country's predictions
        country_pred = pd.DataFrame({
            'Country': [country] * 3,
            'Year': [2013, 2014, 2015],
            **{f: predictions[f] for f in features}
        })
        all_predictions.append(country_pred)

    return pd.concat(all_predictions)


predicted_features = predict_country_features(train, final_features)
```

```python
# 3. Create complete test set with predicted features
test_predicted = test[['Country', 'Year']].copy()
for feature in final_features:
    # Create mapping of (Country, Year) to predicted feature value
    pred_map = predicted_features.set_index(['Country', 'Year'])[feature]
    test_predicted[feature] = test_predicted.set_index(['Country',
'Year']).index.map(pred_map)

# Add back the target variable
test_predicted['Life_expectancy'] = test['Life_expectancy']

# 4. Prepare datasets
X_train = train[final_features]
y_train = train['Life_expectancy']
# Two test sets:
X_test_actual = test[final_features]
y_test_actual = test['Life_expectancy']
X_test_predicted = test_predicted[final_features]
y_test_predicted = test_predicted['Life_expectancy']

# 5. Define and train models
models = {
    'Single Feature': LinearRegression().fit(X_train[[final_features[0]]],
y_train),
    'Multiple Linear': LinearRegression().fit(X_train, y_train),
    'Polynomial (deg=2)': make_pipeline(PolynomialFeatures(degree=2),
LinearRegression()).fit(X_train, y_train)
}

# 6. Evaluate models
results = []
for name, model in models.items():
    # Predict using actual test features
    if name == 'Single Feature':
        y_pred_actual = model.predict(X_test_actual[[final_features[0]]])
    else:
        y_pred_actual = model.predict(X_test_actual)

    # Predict using predicted test features
    if name == 'Single Feature':
        y_pred_predicted =
model.predict(X_test_predicted[[final_features[0]]])
    else:
        y_pred_predicted = model.predict(X_test_predicted)

    results.append({
        'Model': name,
        'Actual Features': {
            'R2': r2_score(y_test_actual, y_pred_actual),
            'MSE': mean_squared_error(y_test_actual, y_pred_actual)
        },
        'Predicted Features': {
            'R2': r2_score(y_test_predicted, y_pred_predicted),
            'MSE': mean_squared_error(y_test_predicted, y_pred_predicted)
        }
    })
```

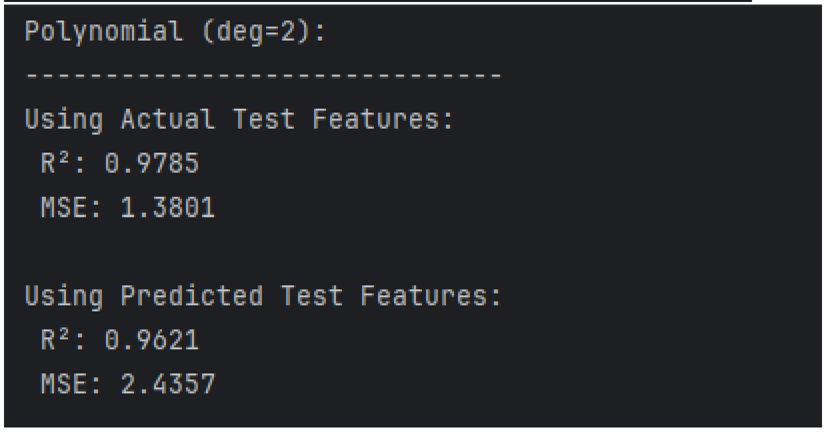
```python
# 7. Display results
print("MODEL PERFORMANCE COMPARISON")
print("=" * 50)
for result in results:
    print(f"\n{result['Model']}:")
    print("-" * 30)
    print("Using Actual Test Features:")
    print(f" R²: {result['Actual Features']['R2']:.4f}")
    print(f" MSE: {result['Actual Features']['MSE']:.4f}")
    print("\nUsing Predicted Test Features:")
    print(f" R²: {result['Predicted Features']['R2']:.4f}")
    print(f" MSE: {result['Predicted Features']['MSE']:.4f}")

# 8. Feature prediction accuracy
print("\nFEATURE PREDICTION ACCURACY")
print("=" * 50)
for feature in final_features:
    actual = test[feature].values
    predicted = test_predicted[feature].values
    r2 = r2_score(actual, predicted)
    mse = mean_squared_error(actual, predicted)
    print(f"\n{feature}:")
    print(f" R²: {r2:.4f}")
    print(f" MSE: {mse:.4f}")
    print(f" Avg Error: {np.mean(np.abs(actual - predicted)):.4f}")
```



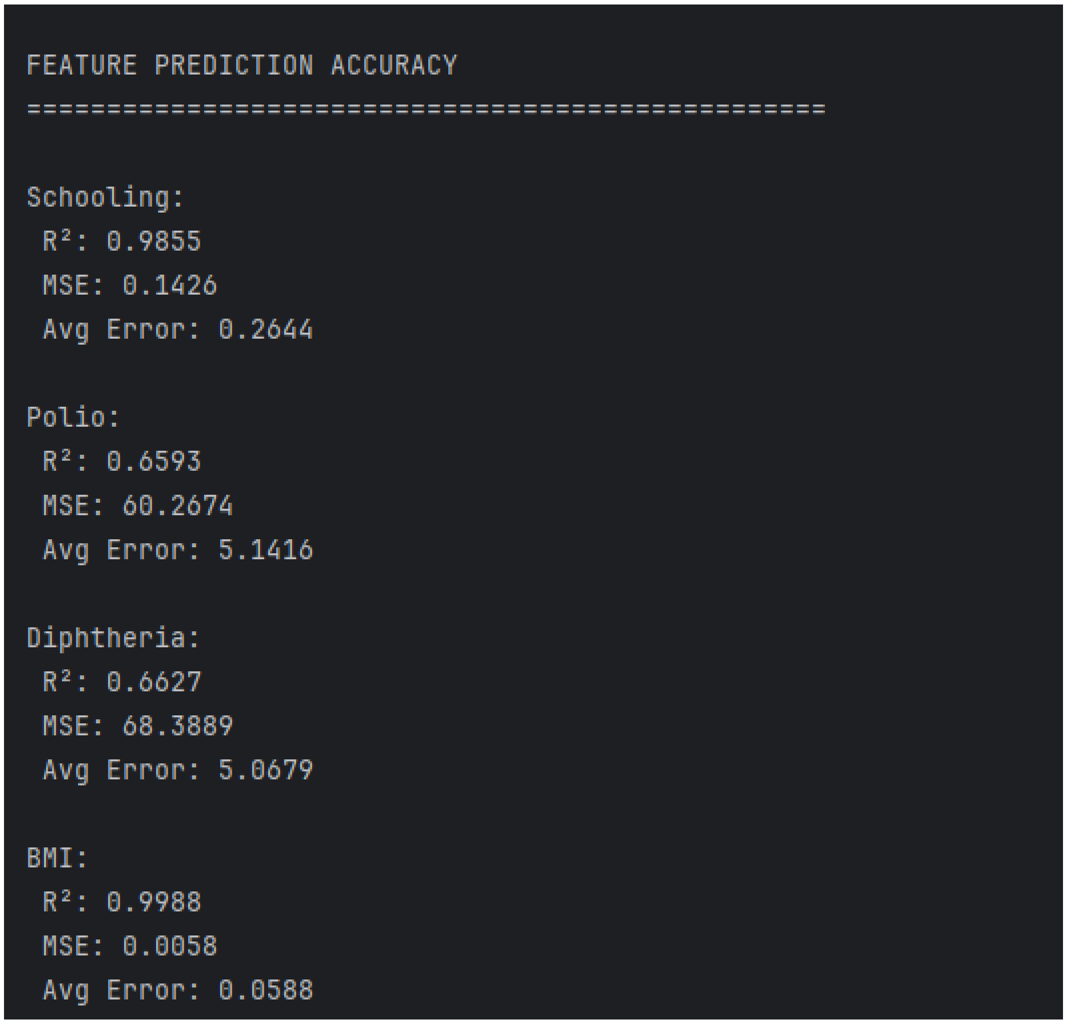
```
Single Feature:
------------------------------
Using Actual Test Features:
 R²: 0.5557
 MSE: 28.5536

Using Predicted Test Features:
 R²: 0.5426
 MSE: 29.3980
```



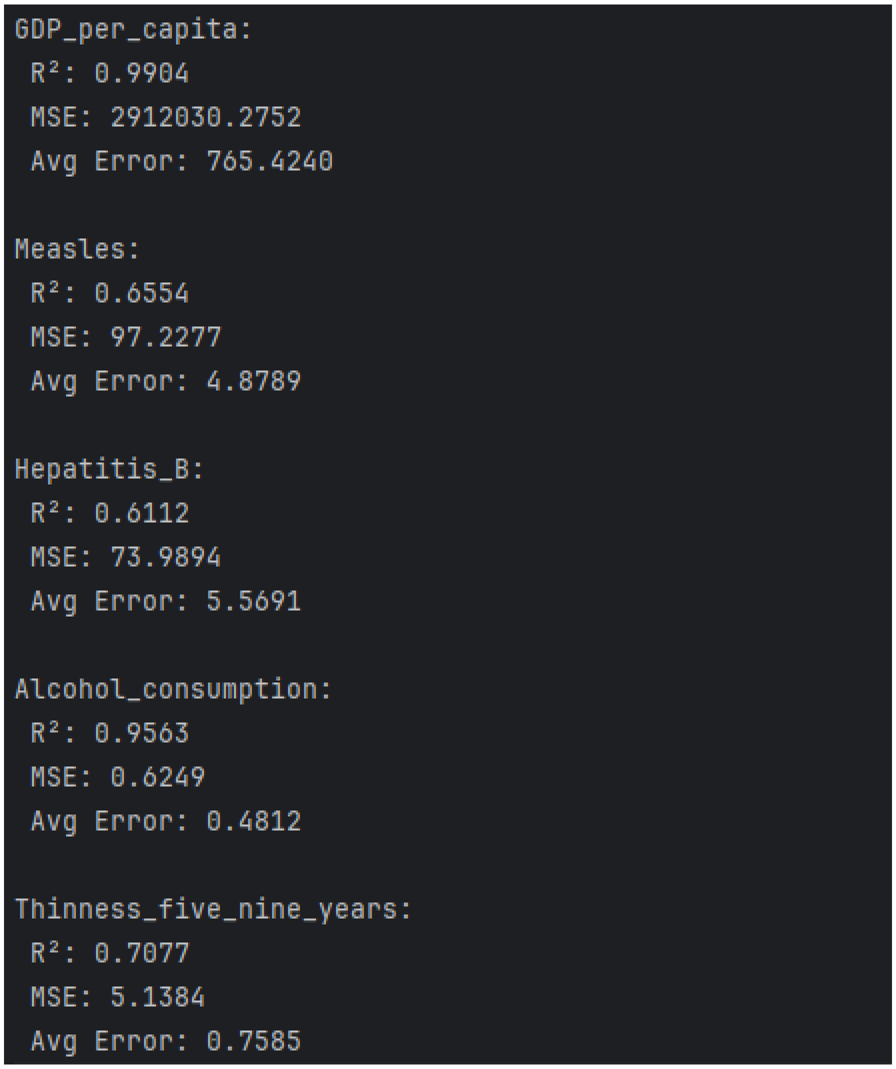
```
Multiple Linear:
------------------------------
Using Actual Test Features:
 R²: 0.9672
 MSE: 2.1110

Using Predicted Test Features:
 R²: 0.9537
 MSE: 2.9771
```



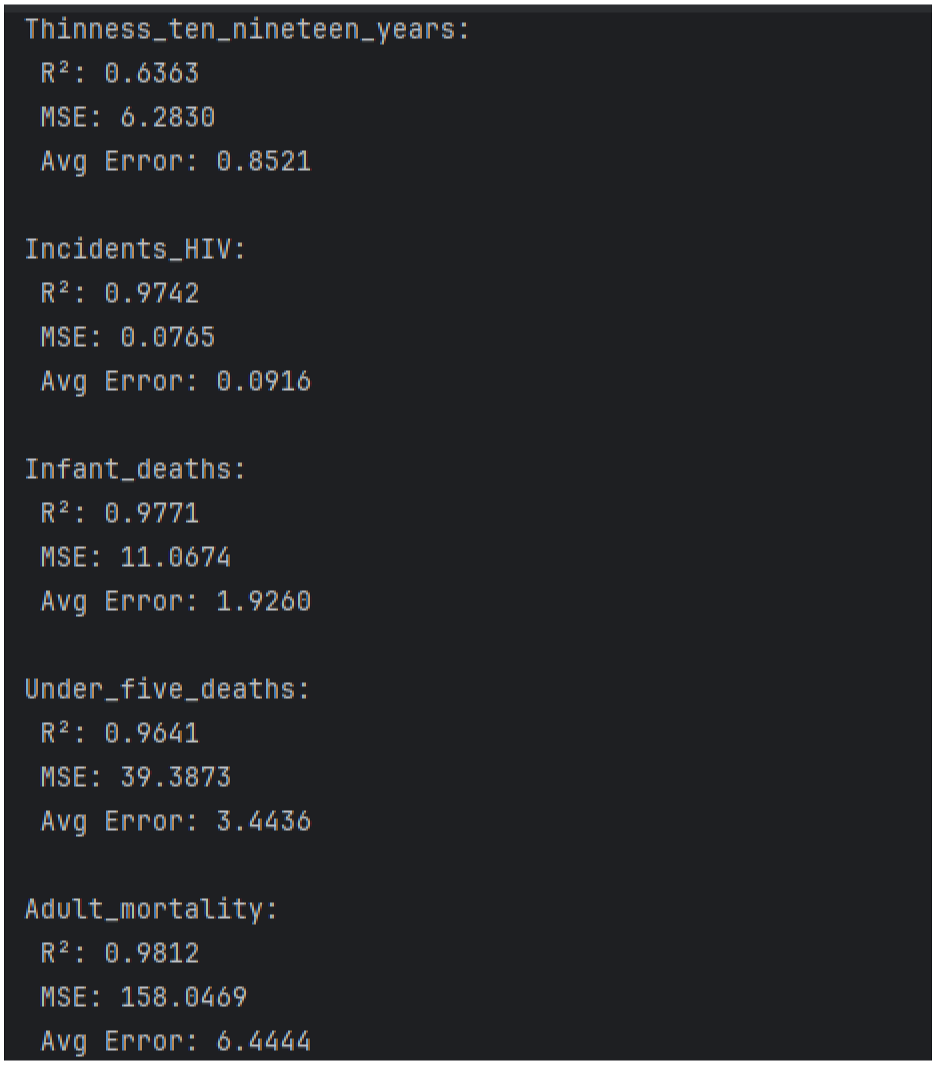
```
Polynomial (deg=2):
------------------------------
Using Actual Test Features:
 R²: 0.9785
 MSE: 1.3801

Using Predicted Test Features:
 R²: 0.9621
 MSE: 2.4357
```



```
FEATURE PREDICTION ACCURACY
==================================================

Schooling:
 R²: 0.9855
 MSE: 0.1426
 Avg Error: 0.2644

Polio:
 R²: 0.6593
 MSE: 60.2674
 Avg Error: 5.1416

Diphtheria:
 R²: 0.6627
 MSE: 68.3889
 Avg Error: 5.0679

BMI:
 R²: 0.9988
 MSE: 0.0058
 Avg Error: 0.0588
```

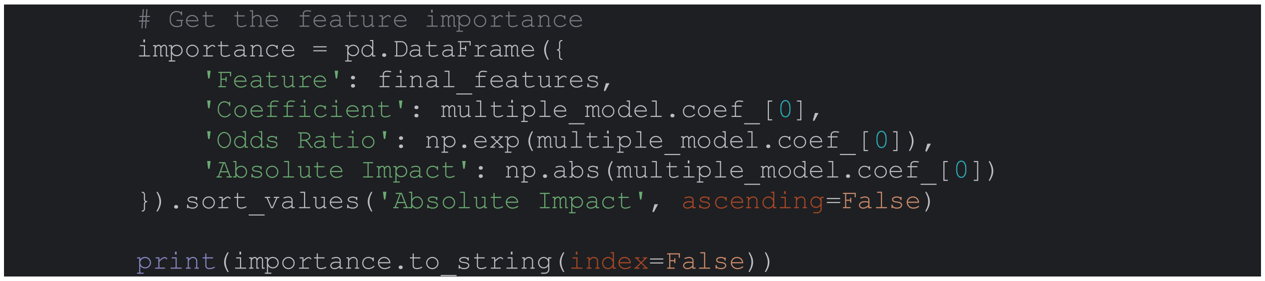


```
GDP_per_capita:
 R²: 0.9904
 MSE: 2912030.2752
 Avg Error: 765.4240

Measles:
 R²: 0.6554
 MSE: 97.2277
 Avg Error: 4.8789

Hepatitis_B:
 R²: 0.6112
 MSE: 73.9894
 Avg Error: 5.5691

Alcohol_consumption:
 R²: 0.9563
 MSE: 0.6249
 Avg Error: 0.4812

Thinness_five_nine_years:
 R²: 0.7077
 MSE: 5.1384
 Avg Error: 0.7585
```

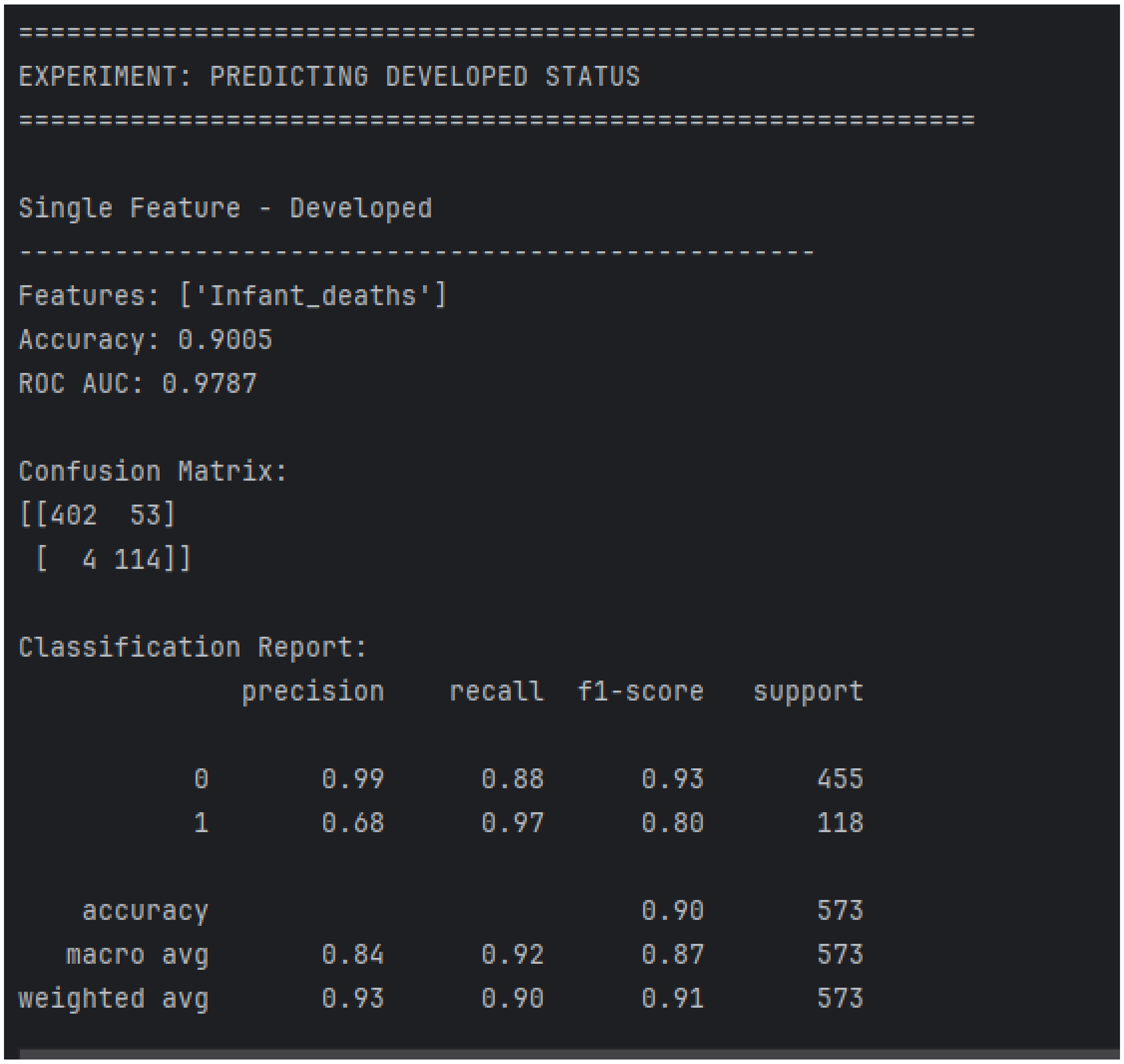


```
Thinness_ten_nineteen_years:
 R²: 0.6363
 MSE: 6.2830
 Avg Error: 0.8521

Incidents_HIV:
 R²: 0.9742
 MSE: 0.0765
 Avg Error: 0.0916

Infant_deaths:
 R²: 0.9771
 MSE: 11.0674
 Avg Error: 1.9260

Under_five_deaths:
 R²: 0.9641
 MSE: 39.3873
 Avg Error: 3.4436

Adult_mortality:
 R²: 0.9812
 MSE: 158.0469
 Avg Error: 6.4444
```

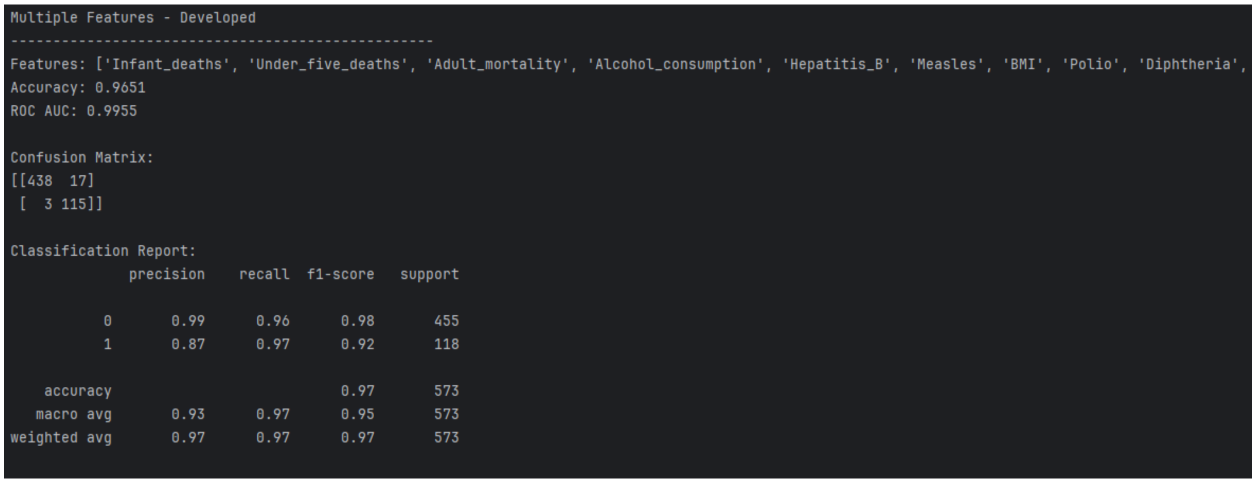
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
roc_auc_score, confusion_matrix

df = pd.read_csv('C:/Users/Laiba Mazhar/Downloads/Life-Expectancy-Data-
Updated.csv')


# 2. Specify the final features to use for training (replace with your actual
feature list)
final_features = ['Infant_deaths', 'Under_five_deaths', 'Adult_mortality',
'Alcohol_consumption',
                  'Hepatitis_B', 'Measles', 'BMI', 'Polio', 'Diphtheria',
'Incidents_HIV',
                  'GDP_per_capita', 'Population_mln',
'Thinness_ten_nineteen_years',
                  'Thinness_five_nine_years', 'Schooling']

# 3. Define the targets
targets = {
    'Developed': 'Economy_status_Developed',
    'Developing': 'Economy_status_Developing'
}

# 4. Define models (reuse for both targets)
models = {
    'Single Feature': {
        'model': LogisticRegression(max_iter=1000, class_weight='balanced'),
        'features': [final_features[0]]  # Use the first feature only
    },
    'Multiple Features': {
        'model': LogisticRegression(max_iter=1000, class_weight='balanced'),
        'features': final_features  # Use all features
    },
    'Polynomial (deg=2)': {
        'model': make_pipeline(
            PolynomialFeatures(degree=2, include_bias=False),
            StandardScaler(),
            LogisticRegression(max_iter=1000, class_weight='balanced')
        ),
        'features': final_features
    }
}

# 5. Run experiments for both targets
all_results = []
for target_name, target_col in targets.items():
    print(f"\n{'=' * 60}")
    print(f"EXPERIMENT: PREDICTING {target_name.upper()} STATUS")
    print('=' * 60)

    y = df[target_col]

    # Split data (stratified by target)
    X = df[final_features]
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42, stratify=y)

    # Scale features
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Train and evaluate models
    target_results = []
    for model_name, config in models.items():
        # Select features based on model
        if model_name == 'Single Feature':
            X_train_model = X_train_scaled[:, [0]]  # Using only the first
feature
            X_test_model = X_test_scaled[:, [0]]
        else:
            X_train_model = X_train_scaled
            X_test_model = X_test_scaled

        # Train the model
        model = config['model']
        model.fit(X_train_model, y_train)

        # Make predictions
        y_pred = model.predict(X_test_model)
        y_proba = model.predict_proba(X_test_model)[:, 1]

        # Store results
        target_results.append({
            'Model': model_name,
            'Accuracy': accuracy_score(y_test, y_pred),
            'ROC AUC': roc_auc_score(y_test, y_proba),
            'Confusion Matrix': confusion_matrix(y_test, y_pred),
            'Classification Report': classification_report(y_test, y_pred),
            'Features Used': config['features']
        })

    # Display results for this target
    for result in target_results:
        print(f"\n{result['Model']} - {target_name}")
        print("-" * 50)
        print(f"Features: {result['Features Used']}")
        print(f"Accuracy: {result['Accuracy']:.4f}")
        print(f"ROC AUC: {result['ROC AUC']:.4f}")
        print("\nConfusion Matrix:")
        print(result['Confusion Matrix'])
        print("\nClassification Report:")
        print(result['Classification Report'])

    all_results.append({
        'Target': target_name,
        'Results': target_results
    })

# 6. Feature importance analysis
print("\nFEATURE IMPORTANCE ANALYSIS")
print("=" * 60)
for target_name, target_col in targets.items():
    print(f"\nFor {target_name} Status Prediction:")
    # Get the multiple logistic regression results
    multiple_model = [r for r in all_results if r['Target'] ==
target_name][0]['Results'][1]['model']

    if isinstance(multiple_model, make_pipeline):
        # For polynomial models, feature importance is not straightforward
        print("Polynomial model - feature importance not straightforward")
    else:
```



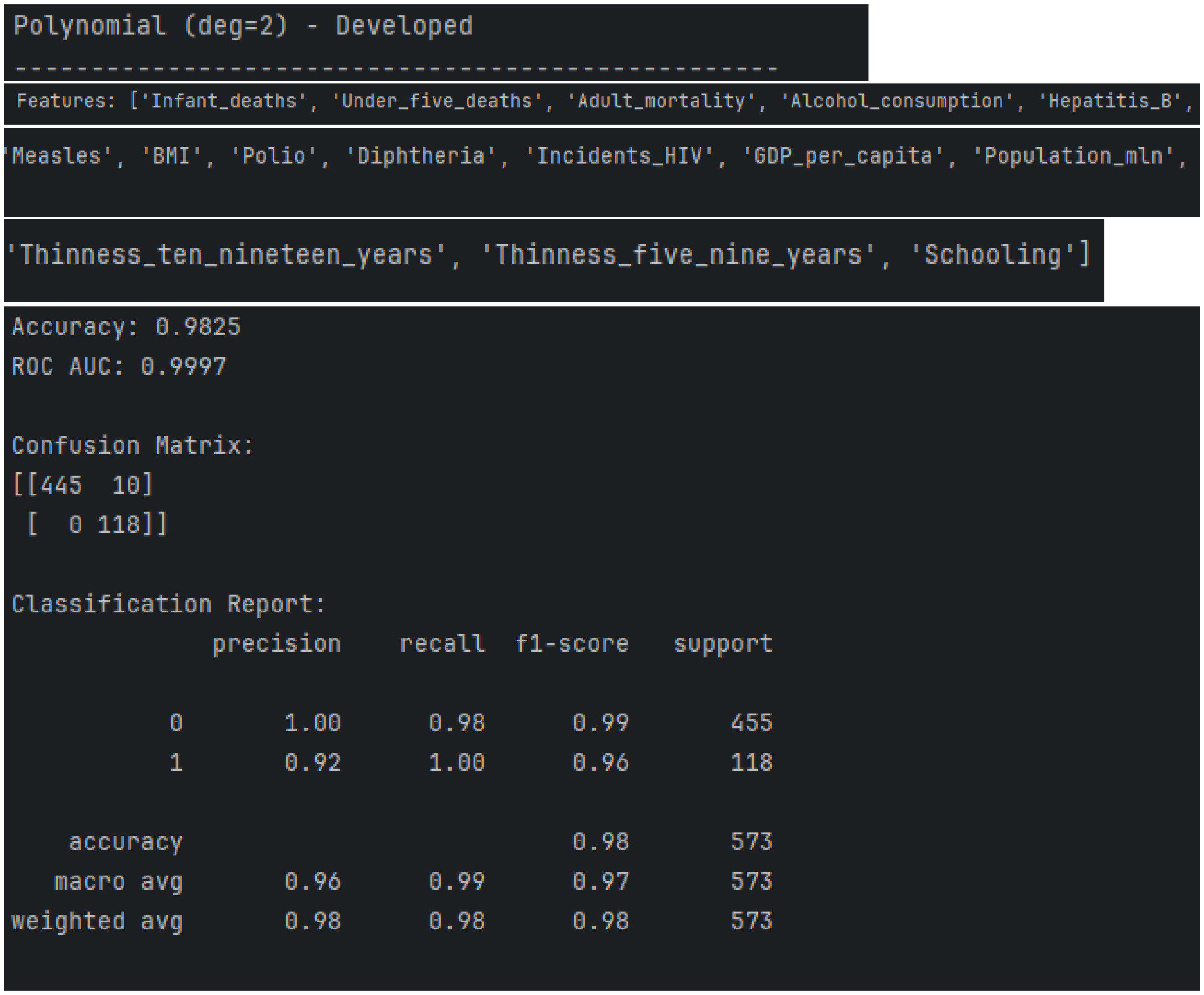
```python
        # Get the feature importance
        importance = pd.DataFrame({
            'Feature': final_features,
            'Coefficient': multiple_model.coef_[0],
            'Odds Ratio': np.exp(multiple_model.coef_[0]),
            'Absolute Impact': np.abs(multiple_model.coef_[0])
        }).sort_values('Absolute Impact', ascending=False)

        print(importance.to_string(index=False))
```



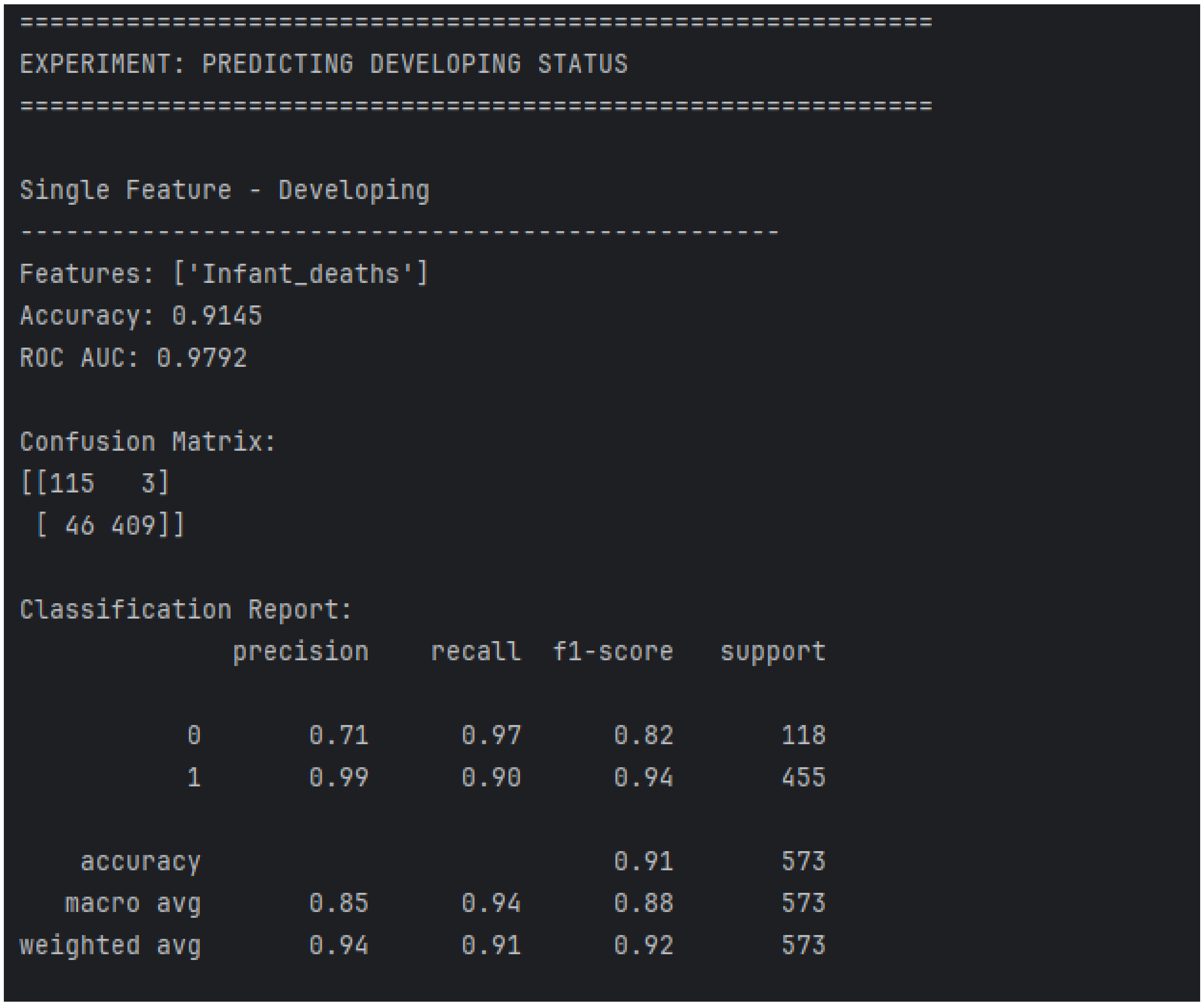
```
============================================================
EXPERIMENT: PREDICTING DEVELOPED STATUS
============================================================


Single Feature - Developed
--------------------------------------------------
Features: ['Infant_deaths']
Accuracy: 0.9005
ROC AUC: 0.9787

Confusion Matrix:
[[402  53]
 [  4 114]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.88      0.93       455
           1       0.68      0.97      0.80       118

    accuracy                           0.90       573
   macro avg       0.84      0.92      0.87       573
weighted avg       0.93      0.90      0.91       573
```



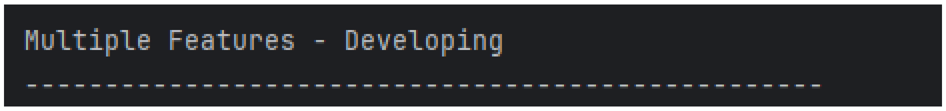
```
Multiple Features - Developed
--------------------------------------------------
Features: ['Infant_deaths', 'Under_five_deaths', 'Adult_mortality', 'Alcohol_consumption', 'Hepatitis_B', 'Measles', 'BMI', 'Polio', 'Diphtheria',
Accuracy: 0.9651
ROC AUC: 0.9955

Confusion Matrix:
[[438  17]
 [  3 115]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.96      0.98       455
           1       0.87      0.97      0.92       118

    accuracy                           0.97       573
   macro avg       0.93      0.97      0.95       573
weighted avg       0.97      0.97      0.97       573
```



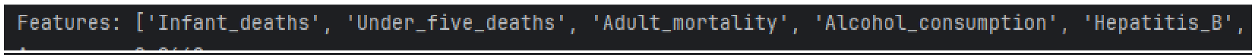
```
Polynomial (deg=2) - Developed
--------------------------------------------------
Features: ['Infant_deaths', 'Under_five_deaths', 'Adult_mortality', 'Alcohol_consumption', 'Hepatitis_B',
'Measles', 'BMI', 'Polio', 'Diphtheria', 'Incidents_HIV', 'GDP_per_capita', 'Population_mln',
'Thinness_ten_nineteen_years', 'Thinness_five_nine_years', 'Schooling']
Accuracy: 0.9825
ROC AUC: 0.9997

Confusion Matrix:
[[445  10]
 [  0 118]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       455
           1       0.92      1.00      0.96       118

    accuracy                           0.98       573
   macro avg       0.96      0.99      0.97       573
weighted avg       0.98      0.98      0.98       573
```



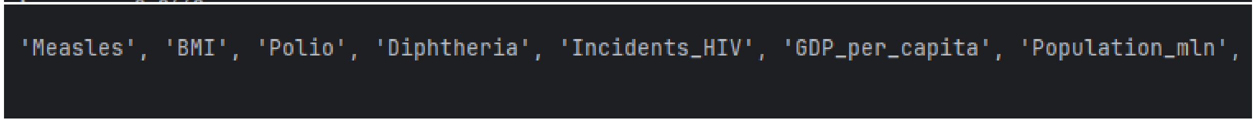
```
============================================================
EXPERIMENT: PREDICTING DEVELOPING STATUS
============================================================

Single Feature - Developing
--------------------------------------------------
Features: ['Infant_deaths']
Accuracy: 0.9145
ROC AUC: 0.9792

Confusion Matrix:
[[115   3]
 [ 46 409]]

Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.97      0.82       118
           1       0.99      0.90      0.94       455

    accuracy                           0.91       573
   macro avg       0.85      0.94      0.88       573
weighted avg       0.94      0.91      0.92       573
```



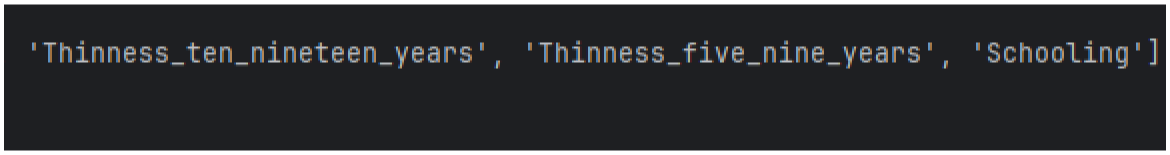
```
Multiple Features - Developing
--------------------------------------------------
```



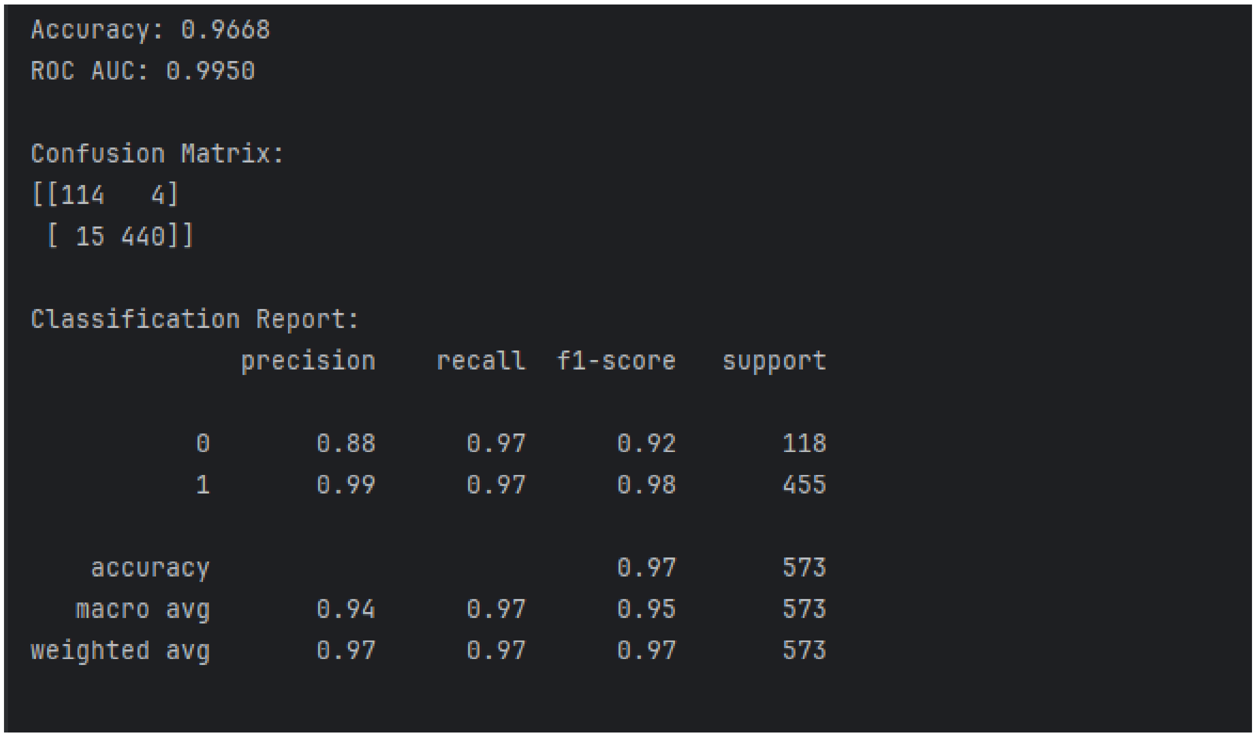
```
Features: ['Infant_deaths', 'Under_five_deaths', 'Adult_mortality', 'Alcohol_consumption', 'Hepatitis_B',
```



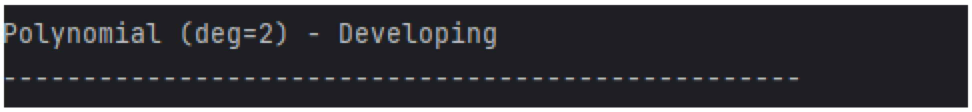
```
'Measles', 'BMI', 'Polio', 'Diphtheria', 'Incidents_HIV', 'GDP_per_capita', 'Population_mln',
```



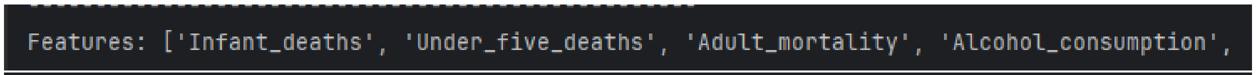
```
'Thinness_ten_nineteen_years', 'Thinness_five_nine_years', 'Schooling']
```



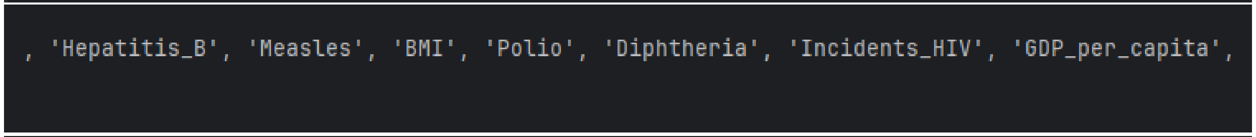
```
Accuracy: 0.9668
ROC AUC: 0.9950

Confusion Matrix:
[[114   4]
 [ 15 440]]

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.97      0.92       118
           1       0.99      0.97      0.98       455

    accuracy                           0.97       573
   macro avg       0.94      0.97      0.95       573
weighted avg       0.97      0.97      0.97       573
```



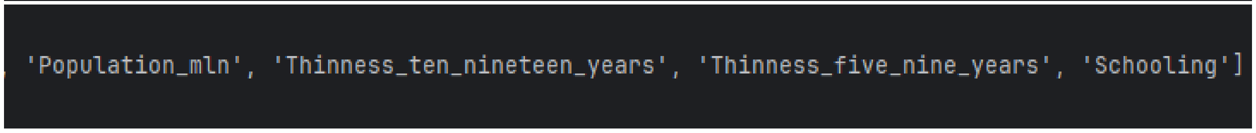
```
Polynomial (deg=2) - Developing
----------------------------------------------------
```



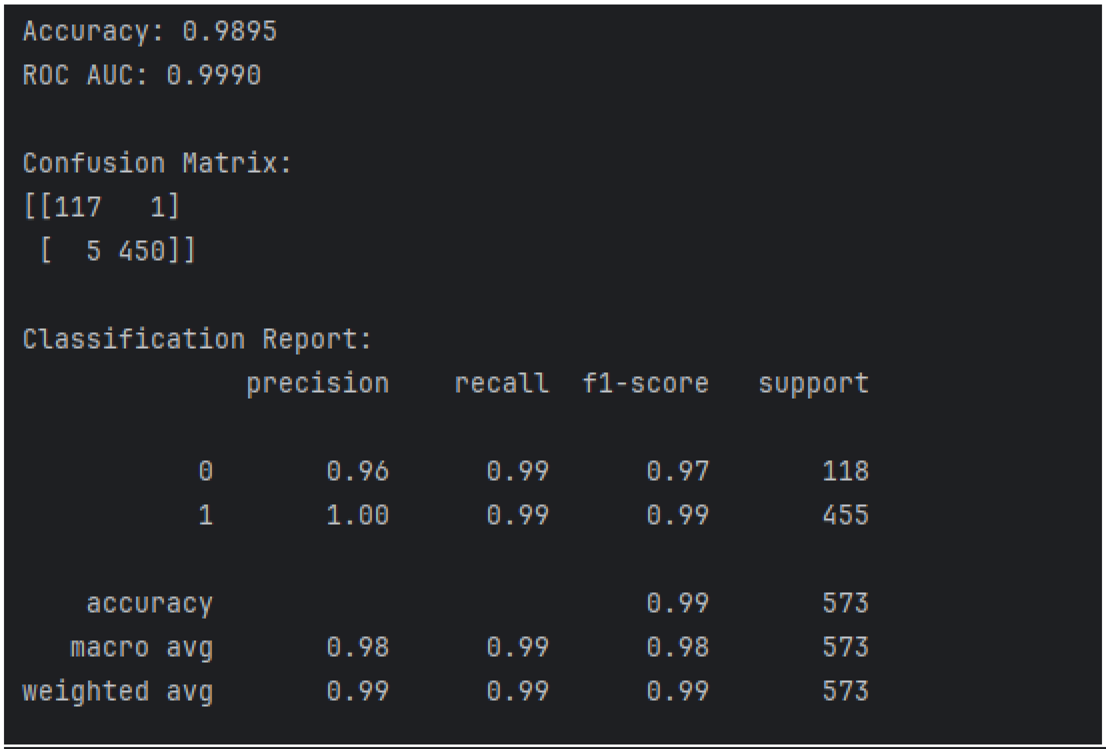
```
 Features: ['Infant_deaths', 'Under_five_deaths', 'Adult_mortality', 'Alcohol_consumption',
```



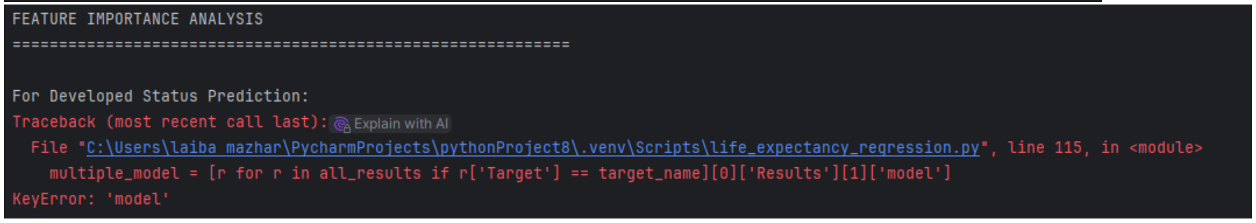
```
, 'Hepatitis_B', 'Measles', 'BMI', 'Polio', 'Diphtheria', 'Incidents_HIV', 'GDP_per_capita',
```



```
, 'Population_mln', 'Thinness_ten_nineteen_years', 'Thinness_five_nine_years', 'Schooling']
```



```
Accuracy: 0.9895
ROC AUC: 0.9990

Confusion Matrix:
[[117   1]
 [  5 450]]

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.99      0.97       118
           1       1.00      0.99      0.99       455

    accuracy                           0.99       573
   macro avg       0.98      0.99      0.98       573
weighted avg       0.99      0.99      0.99       573
```



```
FEATURE IMPORTANCE ANALYSIS
============================================================

For Developed Status Prediction:
Traceback (most recent call last):
  File "C:\Users\laiba mazhar\PycharmProjects\pythonProject8\.venv\Scripts\life_expectancy_regression.py", line 115, in <module>
    multiple_model = [r for r in all_results if r['Target'] == target_name][0]['Results'][1]['model']
KeyError: 'model'
```