import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error, r2\_score

df=pd.read\_csv("Life Expectancy Data.csv")

Df

print("Dataset Information:")

df.info()

print("\nFirst 5 Rows of the Dataset:")

print(df.head())

print("\nSummary Statistics:")

print(df.describe().T)

print("\nData Types:")

print(df.dtypes)

print("\nDataset Shape:")

print(df.shape)

missing\_values = df.isnull().sum()

missing\_percentage = (missing\_values / len(df)) \* 100

print("\nMissing Values Count before handling:")

print(missing\_values)

print("\nPercentage of Missing Values:")

print(missing\_percentage)

print("\nNumber of Unique Values in Each Column:")

print(df.nunique())

excluded\_columns= ['Country','Status','Alcohol','Hepatitis B', 'Total expenditure', 'GDP', 'Population', 'Income composition of resources', 'Schooling']

df.drop(excluded\_columns, axis=1, inplace= True)

df.columns

df = df.dropna(subset=["Life expectancy "])

print("After Dropping missing values in Target column: ")

df.shape

# Handle missing values in features by imputing with median

imputer = SimpleImputer(strategy="median")

df[:] = imputer.fit\_transform(df) # Ensure correct assignment

# Check if any missing values remain after imputation

print("Missing values after imputation:", df.isnull().sum().sum())

X = df.drop(columns=["Life expectancy "])

y = df["Life expectancy "]

# Standardize the features

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Convert back to DataFrame to check for missing values after scaling

X = pd.DataFrame(X, columns=df.drop(columns=["Life expectancy "]).columns)

# Check for missing values after scaling

print("Missing values in X after imputation and scaling:", X.isnull().sum().sum())

plt.figure(figsize=(10,6))

sns.heatmap(df.corr(), annot=True, cmap="coolwarm", fmt=".2f")

plt.title("Correlation Heatmap")

plt.show()

corr\_matrix = df.corr()

# Sort features by correlation with the target variable

correlation\_target = corr\_matrix["Life expectancy "].drop("Life expectancy ").sort\_values(ascending=False)

print(correlation\_target)

# Plot correlation heatmap

plt.figure(figsize=(10, 5))

sns.heatmap(correlation\_target.to\_frame(), annot=True, cmap="coolwarm", linewidths=0.5)

plt.title("Feature Correlation with Life Expectancy")

plt.show()

X\_mlr = df[[" BMI ", "Diphtheria ", "Polio", "Adult Mortality", " HIV/AIDS"]]

y = df["Life expectancy "]

X\_train\_mlr, X\_test\_mlr, y\_train, y\_test = train\_test\_split(X\_mlr, y, test\_size=0.2, random\_state=42)

mlr\_model = LinearRegression()

mlr\_model.fit(X\_train\_mlr, y\_train)

y\_pred\_mlr = mlr\_model.predict(X\_test\_mlr)

mse\_mlr = mean\_squared\_error(y\_test, y\_pred\_mlr)

mae\_mlr = mean\_absolute\_error(y\_test, y\_pred\_mlr)

rmse\_mlr = np.sqrt(mse\_mlr)

r2\_mlr = r2\_score(y\_test, y\_pred\_mlr)

print(f"Mean Squared Error: {mse\_mlr}")

print(f"Mean Absolute Error: {mae\_mlr}")

print(f"Root Mean Squared Error: {rmse\_mlr}")

print(f"R-squared Score: {r2\_mlr}")

plt.figure(figsize=(8,6))

plt.scatter(y\_test, y\_pred\_mlr, alpha=0.7, s=50)

plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], '--r', label="Perfect Fit")

plt.xlabel("Actual Life Expectancy")

plt.ylabel("Predicted Life Expectancy")

plt.title("Actual vs Predicted Life Expectancy")

plt.legend()

plt.show()

Output:





















