**How the model was created**

**1. Data Loading and Initial Cleaning**

We started by loading the dataset from a CSV file named Dataset RSL.csv. We then dropped columns that had no data (PowerCode Code, Reference Period Code, Flags, Flag Codes, Reference Period, and YEA).

df = pd.read\_csv("Dataset RSL.csv")

df = df.drop(columns=['PowerCode Code', 'Reference Period Code', 'Flags', 'Flag Codes', 'Reference Period', 'YEA'])

**2. Handling Missing Values**

Next, we identified and imputed missing values for numerical columns using the median value of each column.

numerical\_columns = df.select\_dtypes(include=[np.number]).columns

for column in numerical\_columns:

df[column].fillna(df[column].median(), inplace=True)

**3. Outlier Detection and Removal**

We used boxplots and z-scores to identify and remove outliers from the dataset. Outliers were defined as data points with z-scores greater than 3.

z\_scores = np.abs(stats.zscore(df[numerical\_columns]))

threshold = 3

outliers = (z\_scores > threshold).any(axis=1)

df = df[~outliers]

**4. Exploratory Data Analysis (EDA)**

To understand the data better, we performed EDA by generating descriptive statistics, plotting correlation heatmaps, and visualizing the distribution of numerical features with histograms.

descriptive\_stats = df.describe()

df = pd.get\_dummies(df, columns=df.select\_dtypes(include=[object]).columns, drop\_first=True)

# Correlation heatmap

plt.figure(figsize=(12, 8))

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

plt.title('Correlation Heatmap')

plt.show()

# Histograms for numerical features

for column in numerical\_columns:

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

sns.histplot(df[column], kde=True, bins=30)

plt.title(f'Distribution of {column}')

plt.xlabel(column)

plt.ylabel('Frequency')

plt.show()

**5. Feature Engineering and Scaling**

We separated the features (X) from the target variable (Value), and scaled the features using StandardScaler.

X = df.drop(columns=['Value'])

y = df['Value']

scaler = StandardScaler()

X\_scaled = pd.DataFrame(scaler.fit\_transform(X), columns=X.columns)

**6. Model Training**

We split the data into training and testing sets and trained a LinearRegression model on the training data. We evaluated the model using Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-squared (R²) score.

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

model = LinearRegression()

model.fit(X\_train, y\_train)

# Predictions and evaluation

y\_pred = model.predict(X\_test)

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

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

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

**7. Model Tuning**

We performed hyperparameter tuning using GridSearchCV on a Ridge regression model to find the best alpha parameter. The tuned model was evaluated using the same metrics as the initial model.

param\_grid = {'alpha': [0.1, 1.0, 10.0, 100.0]}

ridge = Ridge()

grid\_search = GridSearchCV(ridge, param\_grid, cv=5, scoring='neg\_mean\_squared\_error')

grid\_search.fit(X\_train, y\_train)

# Best model from grid search

best\_model = grid\_search.best\_estimator\_

y\_pred\_tuned = best\_model.predict(X\_test)

# Calculate accuracy metrics for the tuned model

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

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

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

**8. Saving the Model and Scaler**

Finally, we saved the trained model and scaler to disk using pickle for future use in predictions.

import pickle

# Save the model

with open('linear\_regression\_model.pkl', 'wb') as file:

pickle.dump(model, file)

# Save the scaler

with open('scaler.pkl', 'wb') as file:

pickle.dump(scaler, file)