Name : Manaswi Kabadi
Class : D15C
Roll no. : 27
Batch : B

EXP - 2

Aim : Implement Multi Regression, Lasso, and Ridge Regression on real-world datasets

Dataset Source : https://www.kaggle.com/datasets/vedavyasv/usa-housing

Dataset Description :

The **USA Housing Dataset** is a real-world real-estate dataset that contains information related to housing prices across different regions in the United States. The dataset is commonly used to analyze how various socio-economic and structural factors influence house prices.
The dataset consists of approximately **5,000 records**, where each record represents a housing instance from a specific area. It is provided in CSV (Comma Separated Values) format and is structured in tabular form. All attributes in the dataset are numerical, which makes it highly suitable for regression-based machine learning algorithms with minimal preprocessing.

Dataset Attributes:

Variable Name	Data Type	Measuring Unit	Description
Avg. Area Income	Numerical (Float)	Currency	Average income of residents in the area
Avg. Area House Age	Numerical (Float)	Years	Average age of houses in the area
Avg. Area Number of Rooms	Numerical (Float)	Count	Average number of rooms per house
Avg. Area Number of Bedrooms	Numerical (Float)	Count	Average number of bedrooms per house
Area Population	Numerical (Float)	Count	Population of the area
Price	Numerical (Float)	Currency	Price of the house

Target Variables Used in the Experiment

- **Price :**A continuous numerical variable representing the house price.This variable is used as the target for **Multiple Linear Regression**, **Ridge Regression**, and **Lasso Regression**..

Theory

Multiple Linear Regression is a supervised machine learning and statistical technique used to predict a continuous dependent variable using two or more independent variables. It is an extension of Simple Linear Regression, which models the relationship between a single predictor and an output. When multiple predictors are involved, the relationship is represented by a multidimensional plane or a hyperplane in higher-dimensional space.
In real-world problems such as house price prediction, a single factor is not sufficient to estimate the target value accurately. Housing prices depend on multiple attributes such as average income of the area, population, number of rooms, and age of houses. Multiple Linear Regression allows all these factors to be considered simultaneously, enabling a more realistic and interpretable prediction model.
In this experiment, Multiple Linear Regression is applied to the USA Housing dataset to predict house prices based on various socio-economic and housing-related features.

Mathematical Formulation of the Algorithm :

In Simple Linear Regression, the relationship between the dependent and independent variable is expressed as:

$$y = \beta_0 + \beta_1 x$$

For Multiple Linear Regression, the equation is extended to include multiple independent variables:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n + \varepsilon$$

Where:

- y is the dependent variable (house price)
- $x1, x2, ..., xn$ are the independent variables (housing features)
- $\beta 0$ is the intercept, representing the predicted value of y when all x values are zero
- $\beta 1, \beta 2, ..., \beta n$ are regression coefficients that represent the effect of each feature on the house price
- ε is the error term, representing unexplained variation

Assumptions of Multiple Linear Regression

For the model to produce reliable predictions, the following assumptions should be satisfied:

- **Linearity:** Each independent variable has a linear relationship with the target variable.

- **No Multicollinearity:** Independent variables should not be highly correlated with one another.
- **Homoscedasticity:** The variance of residuals remains constant across different levels of predictors.
- **Normality of Errors:** Residuals are assumed to be normally distributed.

Algorithm Limitations :

- **Multicollinearity**
  Multiple Linear Regression assumes independence among input features. In the USA Housing dataset, variables such as average number of rooms and number of bedrooms can be correlated.
  Failure Condition: High correlation among predictors leads to unstable coefficient estimates and reduces interpretability.
- **Overfitting with Many Features**
  Adding more predictors increases model complexity.
  Failure Condition: If the number of features becomes large relative to the number of data points, the model may overfit the training data and perform poorly on unseen data.
- **Inability to Capture Feature Interactions**
  The model assumes additive effects of features on the target.
  Failure Condition: Real-world housing prices may depend on interactions between features (e.g., income impact varying with population), which standard Multiple Linear Regression cannot model unless interaction terms are added explicitly.

Methodology / Workflow :

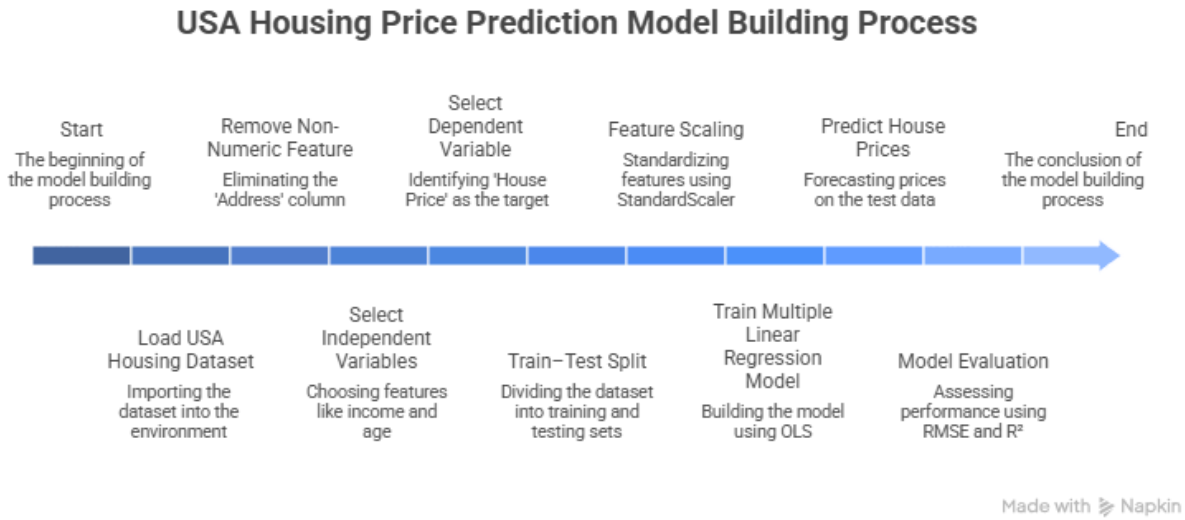
Data Collection:
The USA Housing dataset is loaded into a Pandas DataFrame. It contains numerical housing and socio-economic attributes along with the target variable, Price.
Data Preprocessing:
The non-numeric Address column is removed. Input features are selected, and the dataset is checked for missing values. Feature scaling is applied to ensure uniform contribution of all variables.
Train–Test Split:
The dataset is split into 80% training data and 20% testing data to evaluate model performance on unseen samples.
Model Training and Prediction:
A Multiple Linear Regression model is trained on the training data. The trained model is then used to predict house prices for the test dataset.

Model Evaluation:

Performance is evaluated using **Root Mean Squared Error (RMSE)** and **R^2 score**. Actual and predicted prices are compared to assess prediction accuracy.

Conclusion:

Multiple Linear Regression provides a strong baseline model for predicting house prices and highlights the influence of key housing attributes on pricing.



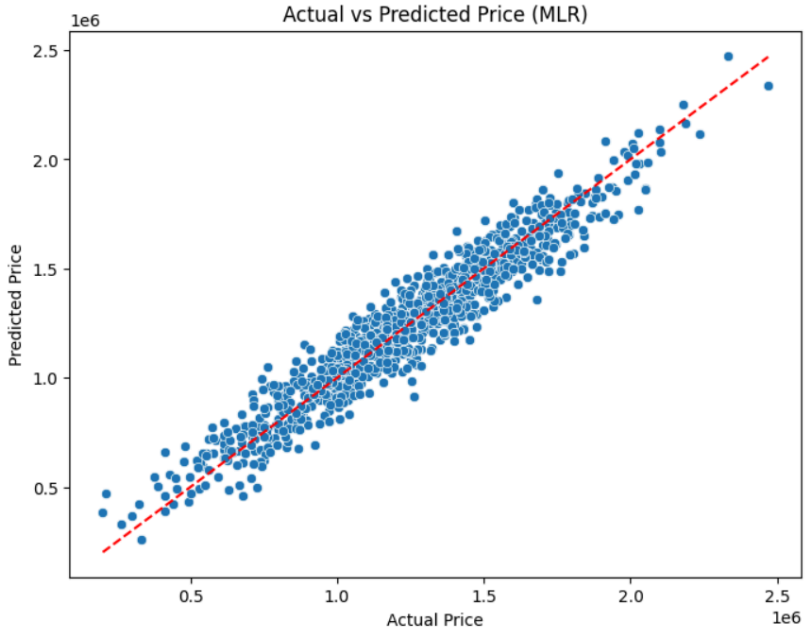
CODE :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
# Load dataset
df = pd.read_csv("USA_Housing.csv")
# Features & target
X = df.drop(['Price', 'Address'], axis=1)
y = df['Price']
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
# Scaling
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Model training
mlr = LinearRegression()
mlr.fit(X_train_scaled, y_train)
# Prediction
y_pred = mlr.predict(X_test_scaled)
# Evaluation
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print("MULTIPLE LINEAR REGRESSION ")
print("RMSE:", rmse)
print("R² Score:", r2)
# Graph: Actual vs Predicted
plt.figure(figsize=(8,6))
sns.scatterplot(x=y_test, y=y_pred)
plt.plot([y_test.min(), y_test.max()],[y_test.min(), y_test.max()], 'r--')
plt.xlabel("Actual Price")
plt.ylabel("Predicted Price")
plt.title("Actual vs Predicted Price (MLR)")
plt.show()
```

OUTPUT :

```
MULTIPLE LINEAR REGRESSION
RMSE: 100444.06055558482
R² Score: 0.9179971706834331
```



Performance Analysis

The Multiple Linear Regression model is evaluated using **RMSE** and **R^2 score**.

- **RMSE:** Represents the average prediction error in **USD**, indicating how far predicted house prices deviate from actual values on average.
- **R^2 Score:** Measures the proportion of variance in house prices explained by the model. A higher R^2 value indicates a better fit.

The obtained results demonstrate that the model captures the primary factors influencing house prices effectively, making it a reliable baseline approach.

Hyperparameter Tuning

Standard Multiple Linear Regression does not involve tunable hyperparameters because it computes a unique optimal solution using the Ordinary Least Squares method. Since there are no parameters to adjust manually, **hyperparameter tuning is not applicable** for this algorithm.

To control model complexity and handle multicollinearity, regularized extensions such as **Ridge Regression** and **Lasso Regression** are required.

RIDGE REGRESSION

Theory

Ridge Regression is a regularized version of Multiple Linear Regression designed to address some of the key limitations of standard linear regression, particularly **multicollinearity** and **overfitting**. While Multiple Linear Regression aims to fit the data as closely as possible, Ridge Regression introduces a penalty term that discourages excessively large coefficient values.

In real-world datasets such as **USA Housing**, many input features are related to each other. For example, average number of rooms, number of bedrooms, and population of an area may be correlated. Such correlations can make coefficient estimates unstable in standard linear regression. Ridge Regression mitigates this issue by shrinking coefficients toward zero, resulting in a more stable and generalized model.

In this experiment, Ridge Regression is applied to predict **house prices** while controlling model complexity using regularization.

Mathematical Formulation of the Algorithm :

Ridge Regression modifies the objective function of Multiple Linear Regression by adding an **L2 regularization term**. The loss function is given by:

$$Cost = \underbrace{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}_{\text{Error (RSS)}} + \underbrace{\lambda \sum_{j=1}^{p} \beta_j^2}_{\text{L2 Penalty}}$$

Where:

- y_i is the actual house price
- $y\hat{}_i$ is the predicted house price
- β_j are the regression coefficients
- λ (alpha) is the regularization parameter

The first term represents the **sum of squared errors**, while the second term penalizes large coefficient values. As λ\lambdaλ increases, the penalty becomes stronger, forcing coefficients to shrink toward zero.Unlike Lasso Regression, Ridge Regression **does not eliminate features entirely**; instead, it reduces their influence smoothly.

Assumptions of Ridge Regression

Ridge Regression shares the same basic assumptions as Multiple Linear Regression, with additional considerations:

- **Linearity:** The relationship between predictors and target remains linear.
- **Scaled Features:** All input features should be on a similar scale for regularization to be effective.
- **Reduced Multicollinearity Impact:** Ridge Regression can handle correlated predictors better than standard linear regression.

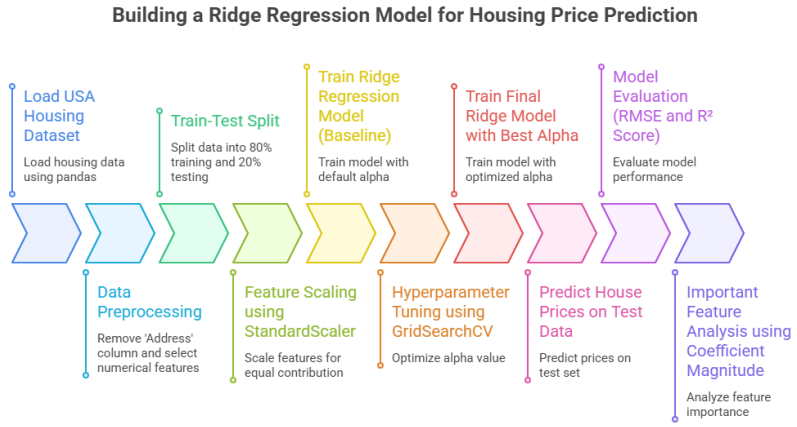
Algorithm Limitations :

- **No Feature Selection**
  Ridge Regression reduces the magnitude of coefficients but does not set them exactly to zero.**Limitation:** All features remain in the model, which may reduce interpretability.
- **Dependency on Regularization Parameter (λ)**
  The performance of Ridge Regression heavily depends on the choice of λ\lambdaλ.
  Limitation: An inappropriate value can lead to underfitting or insufficient regularization.

- **Requires Feature Scaling**
  Ridge Regression is sensitive to feature scale.**Limitation:** Without normalization or standardization, regularization effects may be biased toward features with larger numerical ranges.

Methodology / Workflow :

Data Collection:
The USA Housing dataset is loaded, containing numerical housing attributes and the target variable, Price.
Data Preprocessing:
The Address column is removed as it contains non-numeric data. All remaining features are numerical. Feature scaling is applied using StandardScaler to ensure effective regularization.
Train–Test Split:
The dataset is divided into 80% training data and 20% testing data.
Baseline Model Training:
Ridge Regression is first trained using default parameters to establish baseline performance.
Hyperparameter Tuning:
The regularization parameter (λ) is optimized using GridSearchCV with cross-validation.
Prediction and Evaluation:
The tuned Ridge model predicts house prices on test data, and performance is evaluated using RMSE and R^2 score.



CODE :

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
```

```
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
# Load dataset
df = pd.read_csv("USA_Housing.csv")
# Drop non-numeric column
X = df.drop(['Price', 'Address'], axis=1)
y = df['Price']
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# RIDGE REGRESSION (WITHOUT HYPERPARAMETER TUNING)
ridge_default = Ridge()
ridge_default.fit(X_train_scaled, y_train)
y_pred_def = ridge_default.predict(X_test_scaled)
rmse_def = np.sqrt(mean_squared_error(y_test, y_pred_def))
r2_def = r2_score(y_test, y_pred_def)
print("RIDGE REGRESSION (WITHOUT TUNING)")
print(f"RMSE: {rmse_def:.2f} USD")
print(f"R² Score: {r2_def:.4f}")
# RIDGE REGRESSION (WITH HYPERPARAMETER TUNING)
ridge_params = {'alpha': np.logspace(-2, 2, 50)}
ridge_grid = GridSearchCV(
    Ridge(),
    ridge_params,
    cv=5,
    scoring='r2')
ridge_grid.fit(X_train_scaled, y_train)
best_ridge = ridge_grid.best_estimator_
y_pred = best_ridge.predict(X_test_scaled)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print("\nRIDGE REGRESSION (WITH TUNING)")
print("Best Alpha:", ridge_grid.best_params_)
print(f"RMSE: {rmse:.2f} USD")
print(f"R² Score: {r2:.4f}")
# IMPORTANT FEATURES (RIDGE)
```

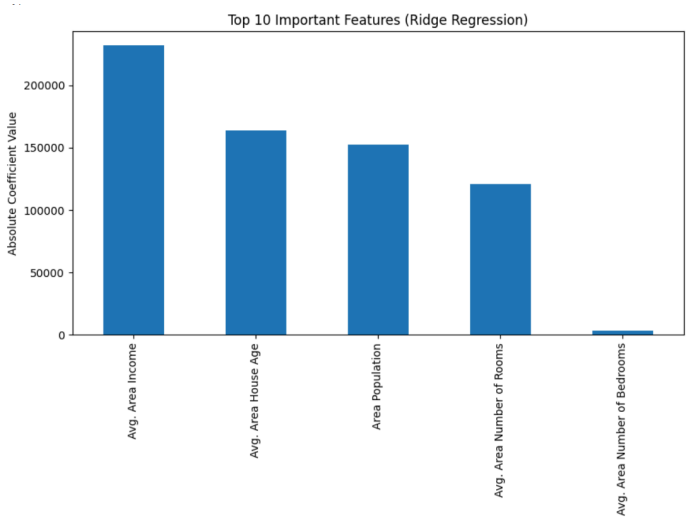
```
ridge_importance = pd.Series(
    best_ridge.coef_,index=X.columns
).abs().sort_values(ascending=False)
print("\nTop 5 Most Important Features (Ridge):")
print(ridge_importance.head(5))
# Graph: Ridge Coefficients
plt.figure(figsize=(10,5))
ridge_importance.head(10).plot(kind='bar')
plt.title("Top 10 Important Features (Ridge Regression)")
plt.ylabel("Absolute Coefficient Value")
plt.show()
```

OUTPUT

```
RIDGE REGRESSION (WITHOUT TUNING)
RMSE: 100444.03 USD
R² Score: 0.9180

RIDGE REGRESSION (WITH TUNING)
Best Alpha: {'alpha': np.float64(0.6250551925273969)}
RMSE: 100444.03 USD
R² Score: 0.9180

Top 5 Most Important Features (Ridge):
Avg. Area Income                231704.561631
Avg. Area House Age             163554.299388
Area Population                 152210.965642
Avg. Area Number of Rooms       120699.601622
Avg. Area Number of Bedrooms      3004.214501
dtype: float64
```



Performance Analysis

The Ridge Regression model is evaluated using **Root Mean Squared Error (RMSE)** and **R^2 score**.

- **RMSE** measures the average prediction error in **USD**, indicating how far predicted house prices deviate from actual values.
- **R^2 score** represents the proportion of variance in house prices explained by the model.

Compared to Multiple Linear Regression, Ridge Regression typically produces **more stable coefficients** and improved generalization, especially when predictor variables are correlated.

Hyperparameter Tuning

Hyperparameter tuning in Ridge Regression focuses on selecting the optimal value of the regularization parameter λ.A range of λ values is evaluated using **GridSearchCV** with k-fold cross-validation. The value that maximizes the R^2 score is selected as the optimal parameter. This tuning process helps balance **bias and variance**, ensuring that the model neither overfits nor underfits the data. In this experiment, hyperparameter tuning confirmed that regularization improves model robustness for the USA Housing dataset.

LASSO REGRESSION

Theory

Lasso Regression (Least Absolute Shrinkage and Selection Operator) is a regularized linear regression technique that extends Multiple Linear Regression by adding a penalty term to the loss function. Like Ridge Regression, it is used to control model complexity and prevent overfitting. However, Lasso Regression has an additional advantage: it can automatically perform feature selection.

In datasets such as USA Housing, not all input variables contribute equally to house price prediction. Some features may have very little influence on the target variable. Lasso Regression helps identify these less important features by shrinking their coefficients exactly to zero, thereby removing them from the model.

This makes Lasso Regression especially useful when model interpretability is important and when we want to identify the most influential housing attributes affecting price.

Mathematical Formulation of the Algorithm :

Lasso Regression modifies the loss function of Multiple Linear Regression by adding an **L1 regularization term**.

The objective function is defined as:

$$Cost = \underbrace{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}_{\text{Error (RSS)}} + \underbrace{\lambda \sum_{j=1}^{p} |\beta_j|}_{\text{L1 Penalty}}$$

Where:

- y_i is the actual house price
- $\hat{y}_i$ is the predicted house price
- β_j are the regression coefficients
- λ (alpha) is the regularization parameter

The L1 penalty encourages sparsity in the model by forcing some coefficients to become exactly zero when the regularization strength is sufficiently large. As a result, Lasso Regression produces a simpler and more interpretable model.

Assumptions of Lasso Regression

Lasso Regression follows most of the assumptions of Multiple Linear Regression, with additional considerations:

- **Linearity:** The relationship between independent variables and the target variable is assumed to be linear.
- **Feature Scaling:** Input features must be scaled so that the regularization penalty is applied uniformly.
- **Sparsity Preference:** Lasso assumes that only a subset of features significantly influences the target variable.

Algorithm Limitations :

- **Instability with Correlated Features**
  When two or more features are highly correlated, Lasso Regression may arbitrarily select one feature and discard others.
  Limitation: This can lead to inconsistent feature selection across different training samples.
- **Sensitivity to Regularization Parameter (λ)**
  The choice of λ\lambdaλ strongly affects model behavior.
  Limitation: A very large λ\lambdaλ may eliminate important predictors, leading to underfitting.
- **Requires Feature Scaling**
  Like Ridge Regression, Lasso is sensitive to feature scale.
  Limitation: Without proper standardization, coefficients may be unfairly penalized.

Methodology / Workflow :

Data Collection:

The USA Housing dataset is loaded, containing numerical housing attributes and the target variable, Price.

Data Preprocessing:

The Address column is removed due to its non-numeric nature. All remaining features are numerical. Feature scaling is applied using StandardScaler to ensure fair regularization.

Train–Test Split:

The dataset is divided into 80% training data and 20% testing data to evaluate generalization performance.

Baseline Model Training:

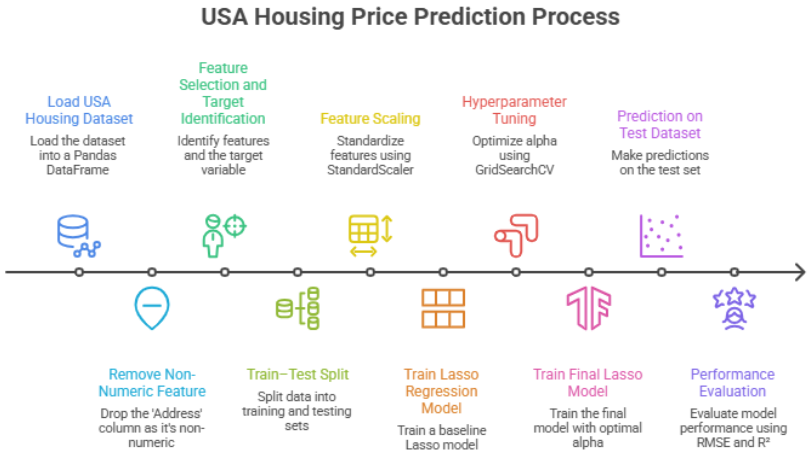
Lasso Regression is initially trained using default parameters to establish baseline performance.

Hyperparameter Tuning:

The regularization parameter (λ\lambdaλ) is tuned using GridSearchCV with cross-validation.

Prediction and Evaluation:

The tuned Lasso model predicts house prices on test data, and performance is evaluated using RMSE and R^2 score.



Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
# Load dataset
df = pd.read_csv("USA_Housing.csv")
```

```
# Drop non-numeric column
X = df.drop(['Price', 'Address'], axis=1)
y = df['Price']
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
# Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# LASSO REGRESSION (WITHOUT HYPERPARAMETER TUNING)
lasso_default = Lasso(max_iter=5000)
lasso_default.fit(X_train_scaled, y_train)
y_pred_def = lasso_default.predict(X_test_scaled)
rmse_def = np.sqrt(mean_squared_error(y_test, y_pred_def))
r2_def = r2_score(y_test, y_pred_def)
print("LASSO REGRESSION (WITHOUT TUNING)")
print(f"RMSE: {rmse_def:.2f} USD")
print(f"R² Score: {r2_def:.4f}")
# LASSO REGRESSION (WITH HYPERPARAMETER TUNING)
lasso_params = {'alpha': np.logspace(-4, 1, 50)}
lasso_grid = GridSearchCV(
    Lasso(max_iter=5000),
    lasso_params,
    cv=5,
    scoring='r2')
lasso_grid.fit(X_train_scaled, y_train)
best_lasso = lasso_grid.best_estimator_
y_pred = best_lasso.predict(X_test_scaled)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
print("\nLASSO REGRESSION (WITH TUNING)")
print("Best Alpha:", lasso_grid.best_params_)
print(f"RMSE: {rmse:.2f} USD")
print(f"R² Score: {r2:.4f}")
# IMPORTANT FEATURES (LASSO)
lasso_importance = pd.Series(
    best_lasso.coef_,
    index=X.columns
).abs().sort_values(ascending=False)
```

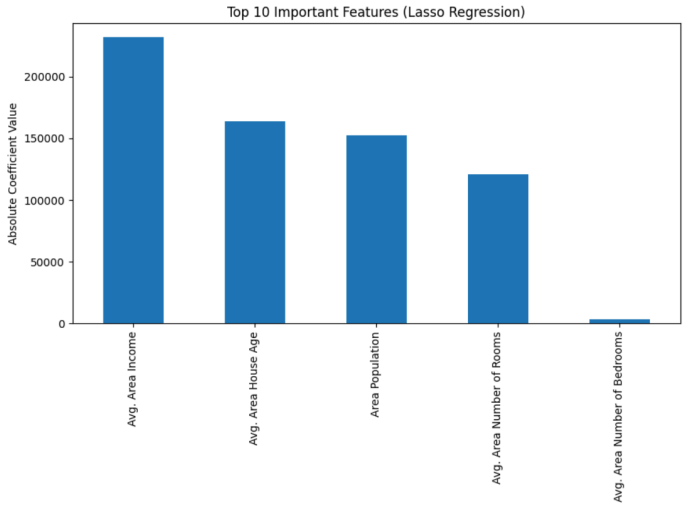
```
print("\nTop 5 Most Important Features (Lasso):")
print(lasso_importance.head(5))
# Graph: Lasso Coefficients
plt.figure(figsize=(10,5))
lasso_importance.head(10).plot(kind='bar')
plt.title("Top 10 Important Features (Lasso Regression)")
plt.ylabel("Absolute Coefficient Value")
plt.show()
```

OUTPUT :

```
LASSO REGRESSION (WITHOUT TUNING)
RMSE: 100444.01 USD
R² Score: 0.9180

LASSO REGRESSION (WITH TUNING)
Best Alpha: {'alpha': np.float64(1.5264179671752334)}
RMSE: 100443.90 USD
R² Score: 0.9180

Top 5 Most Important Features (Lasso):
Avg. Area Income                231740.607670
Avg. Area House Age             163579.448970
Area Population                 152234.290495
Avg. Area Number of Rooms       120726.839601
Avg. Area Number of Bedrooms      2989.966111
dtype: float64
```



Performance Analysis :

The performance of the Lasso Regression model is assessed using **Root Mean Squared Error (RMSE)** and **R^2 score**.

- **RMSE** represents the average prediction error in **USD**, indicating how far predicted house prices deviate from actual values.
- **R^2 score** indicates the proportion of variance in house prices explained by the selected features.

Hyperparameter Tuning

Hyperparameter tuning in Lasso Regression focuses on identifying the optimal value of the regularization parameter λ.A range of λ values is evaluated using GridSearchCV with k-fold cross-validation. The value that yields the highest R^2 score is selected as the optimal parameter.This tuning process helps balance prediction accuracy and feature sparsity, ensuring that only the most relevant housing attributes are retained in the final model.