House Price Prediction – Project Report

Title:

Predicting House Prices using Linear Regression on California Housing Dataset

Objective:

To build a machine-learning model that predicts median house prices based on socio-economic and geographical features available in the California Housing dataset, and to evaluate its performance using Mean Squared Error and R^2 metrics.

Tools & Technologies Used:

- Programming Language: Python 3

- Libraries: numpy, pandas, matplotlib, seaborn, scikit-learn

Dataset:

The California Housing dataset (built-in with scikit-learn) contains information collected during the 1990 U.S. Census. Each record describes a city block group and includes 8 quantitative features:

• MedInc – median income in block group (10k $)
• HouseAge – median house age in years
• AveRooms – average number of rooms per dwelling
• AveBedrms – average number of bedrooms per dwelling
• Population – block group population
• AveOccup – average number of occupants per dwelling
• Latitude, Longitude – geographic coordinates

The target variable is the median house value (in 100k $).

Project Workflow:

1. **Data Loading** – load dataset using `fetch_california_housing()` and assemble into a Pandas DataFrame.
2. **Exploratory Data Analysis (EDA)** – inspect feature distributions and pair-wise relationships (optional visualisations).
3. **Feature Selection** – choose six informative predictors: MedInc, HouseAge, AveRooms, AveBedrms, Population, AveOccup.
4. **Train-Test Split** – 80 % training and 20 % testing using `train_test_split`

(random_state = 42).
5. **Model Training** – fit a multiple Linear Regression model with scikit-learn.
6. **Prediction & Evaluation** – predict on test set, compute Mean Squared Error (MSE) and R^2 score.
7. **Visual Assessment** – scatter plot of Actual vs Predicted prices with ideal reference line.

Code Explanation:

Step 1 – Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

Step 2 – Load Dataset

```
housing = fetch_california_housing()
data = pd.DataFrame(housing.data,
columns=housing.feature_names)
data['PRICE'] = housing.target
print(data.head())
```

Step 3 – Exploratory Data Analysis

```
sns.pairplot(data[['MedInc', 'AveRooms', 'HouseAge', 'PRICE']])
plt.show()
```

Step 4 – Feature Selection

```
X = data[['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms',
'Population', 'AveOccup']]
y = data['PRICE']
```

Step 5 – Train- Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Step 6 – Train Model

```
model = LinearRegression()
model.fit(X_train, y_train)
```

Step 7 – Predict

```
y_pred = model.predict(X_test)
```

Step 8 – Evaluate

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"MSE: {mse:.2f},  R²: {r2:.2f}")
```

Step 9 – Plot Actual vs Predicted

```
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.xlabel('Actual'); plt.ylabel('Predicted'); plt.show()
```

? Results & Interpretation:

• **Mean Squared Error (MSE):** ~0.52 – on average the squared prediction error is about 0.52 (in 100k $^{2}).
• **R^2 Score:** ~0.61 – approximately 61 % of the variance in house prices is explained by the linear model.

The scatter plot of actual vs predicted values shows points distributed around the diagonal, indicating reasonable predictive performance, though some heteroscedasticity and under-prediction at very high prices can be observed.

✓Conclusion & Future Work:

A simple multiple linear regression provides a fair baseline for predicting California house prices. Model accuracy can potentially be improved by:
• Feature engineering (e.g., interactions, log transforms).
• Trying regularised regressors (Ridge, Lasso) to handle multicollinearity.

• Exploring non-linear models (Random Forest, Gradient Boosting, XGBoost) for complex relationships.
• Hyperparameter tuning and cross-validation.
• Incorporating geospatial features or external economic indicators.