Lab NO: 2

Title: Linear Regression

Objective: To understand and apply the complete Machine Learning (ML) pipeline using Linear Regression for a regression (price prediction) problem.

Submitted By : Lijima Shrestha (ACE079BCT035)

Background: AI, ML, DL, and Data Science

- **Artificial Intelligence (AI):** A broad field focused on making machines perform intelligent tasks like reasoning, learning, and decision-making.
- **Machine Learning (ML):** A subset of AI where models learn patterns from data to make predictions.
- **Deep Learning (DL):** A subset of ML using multi-layer neural networks for complex patterns (images, text, speech).
- **Data Science:** Working with data end-to-end: collection, cleaning, analysis, modeling, and decision making.

Used Dependencies

- **NumPy:** Numerical operations.
- **Pandas:** Loading and preprocessing the dataset.
- **Matplotlib:** Visualization (scatter plots, regression line).
- **Scikit-learn:** Train/test split, preprocessing, Linear Regression, evaluation metrics.

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

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

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
```

Task 1: Simple Linear Regression (Single Feature)

Goal: Build a linear regression model using only **housing_median_age** to predict **median_house_value**.

Following **7-step ML pipeline** as required in the assignment.

1Data Retrieval and Collection

We load the dataset (`housing.csv`) and check:

- Shape (rows, columns)
- Column names
- Basic information

```
df = pd.read_csv("housing.csv")
df.shape, df.columns
```

```
((20640, 10),
 Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
        'total_bedrooms', 'population', 'households', 'median_income',
        'median_house_value', 'ocean_proximity'],
       dtype='object'))
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
,RangeIndex: 20640 entries, 0 to 20639
,Data columns (total 10 columns):
, #   Column              Non-Null Count  Dtype
,---  ------              --------------  -----
, 0   longitude           20640 non-null  float64
, 1   latitude            20640 non-null  float64
, 2   housing_median_age  20640 non-null  float64
, 3   total_rooms         20640 non-null  float64
, 4   total_bedrooms      20433 non-null  float64
, 5   population          20640 non-null  float64
, 6   households          20640 non-null  float64
, 7   median_income       20640 non-null  float64
, 8   median_house_value  20640 non-null  float64
, 9   ocean_proximity     20640 non-null  object
,dtypes: float64(9), object(1)
,memory usage: 1.6+ MB
```

2 Data Cleaning

We check for missing values.
In this dataset, **total_bedrooms** has missing values, so we fill it using the **median** of that column.

- **Why median?** It is robust to outliers (unlike mean).

```
df.isna().sum().sort_values(ascending=False)
```

```
total_bedrooms        207
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
population              0
households              0
median_income           0
median_house_value      0
ocean_proximity         0
dtype: int64
```

```
median_bedrooms = df["total_bedrooms"].median()
df["total_bedrooms"] = df["total_bedrooms"].fillna(median_bedrooms)

df.isna().sum().sort_values(ascending=False).head()
```

```
longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
dtype: int64
```

3 Feature Design

For **simple linear regression**, we use:

- Feature (**X**): `housing_median_age`
- Label (**y**): `median_house_value`

Why choose housing_median_age?

- It is a numeric feature that may have some relationship with price.
- Single-feature models are easy to interpret (slope + intercept).

```
X = df[["housing_median_age"]]    # feature
y = df["median_house_value"]        # label

X.head(), y.head()
```

```
(   housing_median_age
 0                41.0
 1                21.0
 2                52.0
 3                52.0
 4                52.0,
 0    452600.0
 1    358500.0
 2    352100.0
 3    341300.0
 4    342200.0
 Name: median_house_value, dtype: float64)
```

4 Algorithm Selection

We select **Linear Regression** because:

- The output (house price) is continuous.
- We want a simple baseline relationship between one feature and the label.

5 Loss Function Selection

We use **Mean Squared Error (MSE)** as the loss function.

- MSE measures the average squared difference between actual and predicted values.
- Lower MSE means better prediction.

6 Model Learning (Training)

We split the dataset:

- **80% training**
- **20% testing**

Then train the Linear Regression model on the training set.

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

model1 = LinearRegression()
model1.fit(X_train, y_train)

model1

LinearRegression()
```

7 Model Evaluation

We evaluate on the test set and report:

- **MSE**
- **R² Score** (optional but useful)

R² close to 1 means better fit; close to 0 means weak explanatory power.

```
y_pred = model1.predict(X_test)

mse1 = mean_squared_error(y_test, y_pred)
r2_1 = r2_score(y_test, y_pred)

mse1, r2_1
```

```
(12939617265.100323, 0.012551235533311389)
```

□ Model Interpretation

For a simple linear regression:

$$\hat{y} = mx + c$$

- **Coefficient (slope, m):** change in predicted price for 1 unit increase in `housing_median_age`.
- **Intercept (c):** predicted price when `housing_median_age = 0`.

```
coef1 = model1.coef_[0]
intercept1 = model1.intercept_

coef1, intercept1
```

```
(np.float64(951.4618671495982), np.float64(179975.00158647486))
```

Interpretation (Task 1):

- **Slope (coefficient) ≈ 951.46**
  → For each **+1 year** increase in `housing_median_age`, the model predicts price increases by about **951.46 USD** (on average, holding nothing else).
- **Intercept ≈ 179975.00**
  → Predicted price when median age is 0 years (not very realistic, but part of the line equation).

Note: The R² score is very low here, meaning *housing age alone* does not explain much of the price variation.

Extra: Visualization (Regression Line + Predicted vs Actual)

We plot: 1) Scatter plot + regression line
2) Predicted vs actual values

Assumptions of Linear Regression

- Linear relationship between X and y
- Errors are normally distributed (approximately)
- Constant variance of errors (homoscedasticity)
- Observations are independent

Task 2: Multiple Linear Regression (All Features)

Goal: Build a linear regression model using **all features** (except `median_house_value`) to predict `median_house_value`.

Following the same **7-step ML pipeline**.

1Data Retrieval and Collection

We already loaded and cleaned the dataset in Task 1.
Now we separate:

- **X (all features)** except label
- **y (label)** = `median_house_value`

```
X2 = df.drop(columns=["median_house_value"])
y2 = df["median_house_value"]

X2.head(), y2.head()

(   longitude  latitude  housing_median_age  total_rooms  
total_bedrooms  \
 0    -122.23     37.88                41.0        880.0  
129.0   
 1    -122.22     37.86                21.0       7099.0  
1106.0   
 2    -122.24     37.85                52.0       1467.0  
190.0   
 3    -122.25     37.85                52.0       1274.0  
235.0   
 4    -122.25     37.85                52.0       1627.0  
280.0   

    population  households  median_income ocean_proximity  
 0       322.0       126.0         8.3252        NEAR BAY  
 1      2401.0      1138.0         8.3014        NEAR BAY  
 2       496.0       177.0         7.2574        NEAR BAY  
 3       558.0       219.0         5.6431        NEAR BAY  
 4       565.0       259.0         3.8462        NEAR BAY   ,
0    452600.0
1    358500.0
2    352100.0
3    341300.0
4    342200.0
Name: median_house_value, dtype: float64)
```

2 Data Cleaning

We already handled missing `total_bedrooms` using median imputation.

We still need to handle **categorical data**:

- `ocean_proximity` is categorical, so we must encode it for regression.

```
# Check missing values (again, because Task 2 uses ALL features)

df.isna().sum().sort_values(ascending=False)

longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
ocean_proximity       0
dtype: int64


# Handle missing values

df["total_bedrooms"] =
df["total_bedrooms"].fillna(df["total_bedrooms"].median())

# Verify after cleaning
df.isna().sum().sort_values(ascending=False).head()

longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
dtype: int64
```

3 Feature Design

- Numeric features are **scaled** using `StandardScaler` (optional but helpful).

- `ocean_proximity` is **one-hot encoded** using `OneHotEncoder`.

We drop the first category in one-hot encoding to avoid the **dummy variable trap**.

```
# Use all features except the label

X2 = df.drop(columns=["median_house_value"])
y2 = df["median_house_value"]

X2.head(), y2.head()
```

```
(   longitude  latitude  housing_median_age  total_rooms
total_bedrooms  \
 0    -122.23     37.88                41.0        880.0
129.0
 1    -122.22     37.86                21.0       7099.0
1106.0
 2    -122.24     37.85                52.0       1467.0
190.0
 3    -122.25     37.85                52.0       1274.0
235.0
 4    -122.25     37.85                52.0       1627.0
280.0

    population  households  median_income ocean_proximity
 0       322.0       126.0         8.3252        NEAR BAY
 1      2401.0      1138.0         8.3014        NEAR BAY
 2       496.0       177.0         7.2574        NEAR BAY
 3       558.0       219.0         5.6431        NEAR BAY
 4       565.0       259.0         3.8462        NEAR BAY  ,
 0    452600.0
 1    358500.0
 2    352100.0
 3    341300.0
 4    342200.0
 Name: median_house_value, dtype: float64)
```

```
# Identify numeric and categorical columns

num_cols = X2.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = [c for c in X2.columns if c not in num_cols]  # e.g.,
ocean_proximity

print("Numeric columns:", num_cols)
print("Categorical columns:", cat_cols)
```

```
Numeric columns: ['longitude', 'latitude', 'housing_median_age',
'total_rooms', 'total_bedrooms', 'population', 'households',
'median_income']
,Categorical columns: ['ocean_proximity']
```

```
# Feature scaling + One-hot encoding setup

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder

preprocess = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), num_cols),  # scaling numeric
        ("cat", OneHotEncoder(handle_unknown="ignore", drop="first"),
cat_cols)  # encoding categorical
    ]
)

preprocess
```

```
ColumnTransformer(transformers=[('num', StandardScaler(),
                                 ['longitude', 'latitude',
'housing_median_age',
                                  'total_rooms', 'total_bedrooms',
'population',
                                  'households', 'median_income']),
                                ('cat',
                                 OneHotEncoder(drop='first',

handle_unknown='ignore'),
                                 ['ocean_proximity'])])
```

4 Algorithm Selection

We again use **Multiple Linear Regression**:

- Output is continuous (price)
- We want to use many factors together to better predict price.

```
# Multiple Linear Regression

from sklearn.linear_model import LinearRegression

lr_model = LinearRegression()
lr_model
```

```
LinearRegression()
```

5 Loss Function Selection

We use the same loss: **Mean Squared Error (MSE)**.

```
# We use Mean Squared Error (MSE) for evaluation
```

```
from sklearn.metrics import mean_squared_error, r2_score
mean_squared_error
```

```
<function sklearn.metrics._regression.mean_squared_error(y_true,
y_pred, *, sample_weight=None, multioutput='uniform_average')>
```

6 Model Learning (Training)

We build a pipeline: 1) Preprocess numeric + categorical features
2) Train Linear Regression model

```
# Train-test split
X2_train, X2_test, y2_train, y2_test = train_test_split(
    X2, y2, test_size=0.2, random_state=42
)

num_cols = X2.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = [c for c in X2.columns if c not in num_cols]

preprocess = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), num_cols),
        ("cat", OneHotEncoder(handle_unknown="ignore", drop="first"),
cat_cols)
    ]
)

model2 = Pipeline(steps=[
    ("preprocess", preprocess),
    ("model", LinearRegression())
])

model2.fit(X2_train, y2_train)
model2
```

```
Pipeline(steps=[('preprocess',
                 ColumnTransformer(transformers=[('num',
StandardScaler(),
                                                  ['longitude',
'latitude',

'housing_median_age',
                                                   'total_rooms',
                                                   'total_bedrooms',
                                                   'population',
'households',
                                                   'median_income']),
                                                 ('cat',

OneHotEncoder(drop='first',
handle_unknown='ignore'),

['ocean_proximity'])])),
                ('model', LinearRegression())])
```

7 Model Evaluation

We compute:

- **MSE**
- **R² Score**

```
y2_pred = model2.predict(X2_test)

mse2 = mean_squared_error(y2_test, y2_pred)
r2_2 = r2_score(y2_test, y2_pred)

mse2, r2_2
```

```
(4908476721.156619, 0.6254240620553604)
```

Model Interpretation

- **Intercept:** base predicted value when all (processed) features are 0

- **Coefficients:** effect of each feature on prediction

⚠ Since numeric features are **standardized**, their coefficients represent:

> change in prediction for a **1 standard deviation** increase in that feature.

For `ocean_proximity` (one-hot), coefficients represent effect compared to the **dropped baseline category**.

```
# Extract intercept and coefficients with names
pre = model2.named_steps["preprocess"]
feature_names = pre.get_feature_names_out()

coefs = model2.named_steps["model"].coef_
intercept = model2.named_steps["model"].intercept_

intercept
```

```
np.float64(219899.77658329843)
```

```
coef_table = pd.DataFrame({
    "Feature": feature_names,
    "Coefficient": coefs
})
coef_table["abs"] = coef_table["Coefficient"].abs()
```

```
coef_table = coef_table.sort_values("abs",
ascending=False).drop(columns="abs")

coef_table

                                Feature    Coefficient
9       cat__ocean_proximity_ISLAND  136125.072615
7                 num__median_income   75167.774766
1                      num__latitude  -54415.696144
0                     num__longitude  -53826.648016
5                    num__population  -43403.432427
4                num__total_bedrooms   43068.181842
8       cat__ocean_proximity_INLAND  -39786.656161
6                    num__households   18382.196324
2            num__housing_median_age   13889.866189
3                   num__total_rooms  -13094.251162
10    cat__ocean_proximity_NEAR BAY   -5136.642217
11  cat__ocean_proximity_NEAR OCEAN    3431.140073
```

Model Comparison (Task 1 vs Task 2)

- **Performance:** Task 2 performs much better because it uses many important predictors (especially `median_income`, location, etc.).
- **Interpretability:** Task 1 is easiest to interpret (only one slope + intercept), but it has weak predictive power.
- **Why multiple features help:** House prices depend on many factors; using only house age ignores most of them.