Lab No:02

Objective: To understand the complete Machine Learning (ML) pipeline by applying it step-by-step to a Linear Regression Problem.

- To train and evaluate linear regression models.
- To interpret model coefficients and intercepts.
- To compare single-feature vs multi-feature regression models.

Submitted By: Apil Maraseni (ACE079BCT012)

Theory:
Linear Regression is a supervised machine learning algorithm used to predict continuous numerical values by modeling a linear relationship between independent variables and a dependent variable. The machine learning pipeline involves data collection, preprocessing, splitting data into training and testing sets, training the model, evaluating performance using metrics such as Mean Squared Error and R^2 score, and interpreting the results. The model fits a best-fit line that minimizes prediction error, where coefficients indicate how much the target variable changes with a unit change in each feature, and the intercept represents the predicted value when all features are zero. Simple linear regression uses one feature, while multiple linear regression uses multiple features to potentially improve prediction accuracy.

Task 1: Simple Linear Regression (Single Feature)
The goal is to build a simple linear regression model using only one input feature(housing_median_age) to predict housing prices.

1️⃣ Data Retrieval and Collection

In [ ]:
```
import pandas as pd

# Load the California Housing dataset (housing.csv already set up)
df = pd.read_csv("housing.csv")

# Display shape of the dataset
print("Dataset Shape:", df.shape)

# Display column names
print("Column Names:")
print(df.columns)
```

In [ ]:
```
df.info()
```

2️⃣ Data Cleaning

In [ ]:
```
# Check missing values
print("Missing values before treatment:")
```

```
print(df.isnull().sum())

# Handle missing values
# California housing usually has missing values in 'total_bedrooms'
if 'total_bedrooms' in df.columns:
    df['total_bedrooms'] = df['total_bedrooms'].fillna(df['total_bedrooms'].median(


# Verify again
print("\nMissing values after treatment:")
print(df.isnull().sum())

# Verify data types
print("\nData Types:")
print(df.dtypes)
```

3 Feature Design

In [ ]:
```
# Feature (independent variable)
X = df[['housing_median_age']]

# Label (dependent variable)
y = df['median_house_value']

print("Feature Shape:", X.shape)
print("Label Shape:", y.shape)

print("\nFirst 5 Feature values:")
print(X.head())

print("\nFirst 5 Label values:")
print(y.head())
```

4 Algorithm Selection

In [ ]:
```
# Import Linear Regression model
from sklearn.linear_model import LinearRegression

# Create model object
model = LinearRegression()

print("Linear Regression model initialized successfully.")
```

Linear Regression was chosen because the task involves predicting a continuous numerical value (median_house_value). It is appropriate for modeling the relationship between a numerical feature (housing_median_age) and house price, and it provides easily interpretable coefficients and intercept.

5 Loss Function Selection

In [ ]:
```
# Import Mean Squared Error
from sklearn.metrics import mean_squared_error
```

```
print("Mean Squared Error (MSE) will be used as the loss function.")
```

Mean Squared Error (MSE) was selected as the loss function because it measures the average squared difference between actual and predicted values. It is appropriate for regression tasks where the goal is to minimize prediction error. Since larger errors are squared, MSE penalizes them more heavily, helping the model learn more accurate predictions.

6️⃣ Model Learning (Training)

In [ ]:
```
from sklearn.model_selection import train_test_split

# Split data (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train the model
model.fit(X_train, y_train)

print("Model training completed successfully.")
```

The dataset was divided into training and testing sets using an 80:20 ratio. The training data was used to teach the model the relationship between housing_median_age and median_house_value. During training, the linear regression algorithm learns the best-fit line by calculating the optimal coefficient and intercept that minimize the Mean Squared Error. The testing data is kept separate to evaluate how well the model performs on unseen data.

7️⃣ Model Evaluation

In [ ]:
```
from sklearn.metrics import mean_squared_error, r2_score

# Make predictions on test data
y_pred = model.predict(X_test)

# Calculate evaluation metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error (MSE):", mse)
print("R² Score:", r2)
```

The trained model was evaluated using the test dataset. Mean Squared Error (MSE) measures the average squared difference between actual and predicted house values, where a lower value indicates better performance. The R^2 score represents how well the feature explains the variation in house prices, with values closer to 1 indicating a stronger relationship. If the R^2 value is low, it suggests that housing_median_age alone does not strongly predict house prices.

EXTRA

1️⃣ Visualize the Regression Line

In [ ]:
```
import matplotlib.pyplot as plt

# Plot actual data points
plt.scatter(X_test, y_test)

# Plot regression line
plt.plot(X_test, y_pred)

plt.xlabel("Housing Median Age")
plt.ylabel("Median House Value")
plt.title("Linear Regression Line")

plt.show()
```

2️⃣ Plot Predicted vs Actual Values

In [ ]:
```
plt.scatter(y_test, y_pred)

plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Actual vs Predicted House Values")

plt.show()
```

The regression line shows the linear relationship between housing_median_age and median_house_value. If the data points are widely scattered around the line, it indicates a weak relationship.

In the predicted vs actual plot, points closer to the diagonal indicate better predictions. Large deviations suggest that the single feature may not strongly explain house prices.

Assumptions of Linear Regression:

Linearity – The relationship between feature and target is linear.

Independence – Observations are independent of each other.

Homoscedasticity – Constant variance of errors.

Normality – Residuals are approximately normally distributed.

No multicollinearity – (Important in multiple regression) features should not be highly correlated.

📈 Model Interpretation

In [ ]:
```
# Get coefficient and intercept
coefficient = model.coef_[0]
```

```
intercept = model.intercept_

print("Coefficient (Slope):", coefficient)
print("Intercept:", intercept)
```

The coefficient (slope) represents how much the median house value changes for every one-unit increase in housing_median_age. If the coefficient is positive, house value increases as the age increases; if negative, house value decreases as the age increases.

The intercept represents the predicted median house value when housing_median_age is zero. In this context, it is the starting value of the regression line, although it may not always have practical meaning if zero is not a realistic value in the dataset.

Task 2: Multiple Linear Regression (All Features) The goal is to build a multiple linear regression model using all available input features(All remaining features except median_house_value) to predict housing prices.

1️⃣ Data Retrieval and Collection

In [ ]:
```
import pandas as pd

df = pd.read_csv("housing.csv")

print("Dataset Shape:", df.shape)
print("Column Names:", df.columns.tolist())
```

2️⃣ Data Cleaning

In [ ]:
```
# Check missing values
print("Missing values before treatment:\n", df.isnull().sum())

# Fill missing total_bedrooms with median (common in this dataset)
df['total_bedrooms'] = df['total_bedrooms'].fillna(df['total_bedrooms'].median())

print("\nMissing values after treatment:\n", df.isnull().sum())
print("\nData Types:\n", df.dtypes)
```

3️⃣ Feature Design (All Features)

In [ ]:
```
# Separate features (X) and label (y)
X = df.drop(columns=['median_house_value'])
y = df['median_house_value']

print("X shape:", X.shape)
print("y shape:", y.shape)
```

4️⃣ Algorithm Selection (Multiple Linear Regression)

In [ ]:
```
from sklearn.linear_model import LinearRegression
```

```
model = LinearRegression()
print("Multiple Linear Regression model initialized.")
```

Linear Regression is appropriate because the target (median_house_value) is continuous. With multiple features, the model learns a linear combination of inputs to predict housing prices.

5 Loss Function Selection (MSE)

In [ ]:
```
from sklearn.metrics import mean_squared_error, r2_score

print("MSE will be used to measure prediction error.")
```

Mean Squared Error (MSE) measures the average squared difference between actual and predicted values. Lower MSE indicates better predictive performance.

6 Model Learning (Training) + Feature Scaling/Encoding

In [ ]:
```
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline

# Split first (prevents leakage)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Identify numeric and categorical columns
numeric_features = X_train.select_dtypes(include=["int64", "float64"]).columns
categorical_features = X_train.select_dtypes(include=["object"]).columns

# Preprocessing: scale numeric, one-hot encode categorical
preprocess = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_features),
        ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_features),
    ]
)

# Full pipeline: preprocess + model
multi_lr = Pipeline(steps=[
    ("preprocess", preprocess),
    ("model", LinearRegression())
])

# Train
multi_lr.fit(X_train, y_train)

print("Model training completed (multiple linear regression).")
```

The data was split into training and testing sets (80/20). Numeric features were scaled using StandardScaler to bring them to a similar range, and the categorical feature

(ocean_proximity) was converted into numeric form using one-hot encoding. The model then learned coefficients that best minimize error on the training data.

7️⃣ Model Evaluation

In [ ]:
```
# Predict on test set
y_pred_multi = multi_lr.predict(X_test)

# Metrics
mse_multi = mean_squared_error(y_test, y_pred_multi)
r2_multi = r2_score(y_test, y_pred_multi)

print("Multiple LR - Mean Squared Error (MSE):", mse_multi)
print("Multiple LR - R² Score:", r2_multi)
```

MSE indicates average squared prediction error (lower is better). R^2 shows how much variance in house prices is explained by the features (closer to 1 is better). Multiple features typically increase R^2 compared to single-feature regression because the model uses more information relevant to pricing.

Coefficients & Intercept for Multiple Regression

In [ ]:
```
import numpy as np

# Get feature names after preprocessing
feature_names_num = numeric_features.tolist()
feature_names_cat = multi_lr.named_steps["preprocess"] \
    .named_transformers_["cat"] \
    .get_feature_names_out(categorical_features).tolist()

all_feature_names = feature_names_num + feature_names_cat

# Extract coefficients + intercept
coef = multi_lr.named_steps["model"].coef_
intercept = multi_lr.named_steps["model"].intercept_

# Show top 10 largest (by absolute value) coefficients
top_idx = np.argsort(np.abs(coef))[::-1][:10]
print("Intercept:", intercept)
print("\nTop 10 coefficients (by absolute value):")
for i in top_idx:
    print(f"{all_feature_names[i]}: {coef[i]}")
```

Each coefficient shows how the predicted house value changes with a one-unit increase in that feature (after preprocessing), while holding other features constant. The intercept is the baseline prediction when all processed feature values are zero.

📈 Model Interpretation (Multiple Linear Regression)

In [ ]:
```
# Extract intercept
intercept = multi_lr.named_steps["model"].intercept_
```

```
# Extract coefficients
coefficients = multi_lr.named_steps["model"].coef_

# Get feature names after preprocessing
feature_names_num = numeric_features.tolist()
feature_names_cat = multi_lr.named_steps["preprocess"] \
    .named_transformers_["cat"] \
    .get_feature_names_out(categorical_features).tolist()

all_feature_names = feature_names_num + feature_names_cat

print("Intercept:", intercept)

print("\nAll Coefficients:")
for name, coef in zip(all_feature_names, coefficients):
    print(f"{name}: {coef}")
```

The intercept represents the predicted house value when all processed input features are zero (after scaling and encoding).

Each coefficient represents how much the predicted median house value changes when that specific feature increases by one unit, while keeping all other features constant.

A positive coefficient means the house value increases as that feature increases.

A negative coefficient means the house value decreases as that feature increases. For categorical features (like ocean_proximity), coefficients represent the change in price relative to the reference category.

Difference from Single-Feature Model

The single-feature model uses only one predictor (housing_median_age), so it explains price variation using only that factor. The multiple linear regression model considers all relevant features simultaneously, allowing it to capture more complex relationships and usually achieve better prediction performance.

📊 Model Comparison

The multiple linear regression model (Task 2) performs better than the single-feature model (Task 1) if it has a lower MSE and a higher R^2 score. This is because it uses several relevant features such as income, rooms, population, and location, which together provide more information about housing prices than just housing_median_age alone.

Using multiple features generally improves performance because house prices are influenced by many factors. However, if irrelevant or highly correlated features are included, performance may not improve significantly and the model may become more complex.

The single-feature model is easier to interpret because it has only one coefficient and clearly shows the effect of one variable. The multiple-feature model provides better accuracy but is

more complex to interpret due to the presence of many coefficients.

Discussion and Conclusion

This assignment applied the complete Machine Learning pipeline to predict housing prices using linear regression. The simple linear regression model, built using only housing_median_age, demonstrated a basic linear relationship but showed limited predictive performance since housing prices depend on multiple factors. The multiple linear regression model incorporated all relevant features and included preprocessing steps such as scaling and encoding, resulting in improved accuracy and better explanatory power.

The implementation successfully demonstrated both simple and multiple linear regression within a structured ML workflow. While the single-feature model was easier to interpret, the multiple-feature model provided better prediction performance by utilizing more information. Overall, the assignment strengthened understanding of data preprocessing, model training, evaluation metrics, and interpretation of regression coefficients and intercepts.