

Boston Housing Linear Regression



Here's the history of the project based on the prompts and responses, showing the **key prompts** and **code generations** as per the **version control** records.

Step 1: Business Understanding

Prompt: *"I'm asked to do a project using multiple feature linear regression to analyze the Boston housing dataset to predict the house price. Please review your setting and make a plan."*

Response Summary: A comprehensive CRISP-DM framework plan was proposed, including steps for business understanding, data understanding, data preparation, modeling, evaluation, and deployment. The focus was on feature selection methods and model evaluation criteria.

Step 2: Data Understanding

Prompt-Step2: *"Let's move to Step 2 for data understanding. Write the code to fulfill Step 2."*

Code-Step2:

```
python

# Import necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
url = "https://raw.githubusercontent.com/jesva86/datasets/master/BostonHousing.csv"
df = pd.read_csv(url)

# Display the first few rows of the dataset
df.head()

# Descriptive statistics
df.describe()

# Check for missing values
df.isnull().sum()

# Correlation matrix and heatmap to see the relationships between features and the target (medv)
plt.figure(figsize=(12, 8))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap of Boston Housing Dataset')
plt.show()

# Scatter plots of key features with the target variable (medv)
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['rm'], y=df['medv'])
plt.title('Number of Rooms (RM) vs House Price (MEDV)')
plt.xlabel('Average number of rooms per dwelling (RM)')
plt.ylabel('Median value of owner-occupied homes (MEDV)')
plt.show()

plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['lstat'], y=df['medv'])
plt.title('% Lower Status Population (LSTAT) vs House Price (MEDV)')
plt.xlabel('% Lower status of the population (LSTAT)')
plt.ylabel('Median value of owner-occupied homes (MEDV)')
plt.show()
```

Step 3: Data Preparation

Prompt-Step3: *"Let's move to Step 3 for data preparation. Please outline the plan before starting, then break down each topic and write the code. Also, ensure the data split includes a training, validation, and test set split."*

Key Change: Training data was split further into **training and validation sets** (not just train-test split), and feature scaling was performed using **StandardScaler**.

Code-Step3:

```
python

# Import necessary libraries for data preparation
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Define features (X) and target (y)
X = df.drop('medv', axis=1)
y = df['medv']

# Step 1: Train-Test split (80% train, 20% test)
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Step 2: Further split the training set into training (80%) and validation (20%) sets
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.2, random_state=42)

# Step 3: Feature scaling using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

Step 4: Modeling and Feature Selection

Prompt-Step4: *"Let's move to Step 4 to build the multiple linear regression model. We've been asked to analyze the importance of each feature using the train & validation data with the following methods: Lasso, MIFS, RFE, and SelectKBest. Provide the rank of feature importance, MSE, and R² values iteratively as features are added. Compare the best performance across methods and list the best-performing features."*

Key Changes:

- Multiple feature selection methods were applied: **Lasso, MIFS, RFE, SelectKBest**.
- Feature importance was ranked, and MSE/R² were calculated iteratively.

Code-Step4:

```
python

# LassoCV for feature selection
lasso = LassoCV(cv=5, random_state=42).fit(X_train_scaled, y_train)
lasso_coefs = np.abs(lasso.coef_)
lasso_ranked_features = np.argsort(lasso_coefs)[::-1]
lasso_ranked_feature_names = [feature_names[i] for i in lasso_ranked_features]

# Store results for Lasso with feature names
lasso_results_with_names = []
for i in range(1, len(lasso_ranked_features) + 1):
    selected_features = lasso_ranked_features[:i]
    mse, r2 = evaluate_model(X_train_scaled, y_train, X_val_scaled, y_val, selected_features)
    selected_feature_names = lasso_ranked_feature_names[:i]
    lasso_results_with_names.append((i, selected_feature_names, mse, r2))
```

Step 5: Model Evaluation

Prompt-Step5: *"Please read previous code (Step 1 to Step 4), and let's move to Step 5: Evaluation using the union of the most important features. As usual, consider the version control for the prompt and code."*

Key Change:

- A new **linear regression model** was trained and evaluated on the **validation** and **test sets** using the **union of important features**.

Code-Step5:

```
python

# Union of important features
union_features = ['rm', 'lstat', 'dis', 'nox', 'ptratio', 'rad', 'indus', 'crim', 'b', 'tax']

# Select the relevant features from the training, validation, and test sets
X_train_union = X_train_scaled[:, [feature_names.index(f) for f in union_features]]
X_val_union = X_val_scaled[:, [feature_names.index(f) for f in union_features]]
X_test_union = X_test_scaled[:, [feature_names.index(f) for f in union_features]]

# Train a linear regression model using the selected features
final_model = LinearRegression()
final_model.fit(X_train_union, y_train)

# Evaluate on validation set
y_val_pred = final_model.predict(X_val_union)
val_mse = mean_squared_error(y_val, y_val_pred)
val_r2 = r2_score(y_val, y_val_pred)

# Evaluate on test set
y_test_pred = final_model.predict(X_test_union)
test_mse = mean_squared_error(y_test, y_test_pred)
test_r2 = r2_score(y_test, y_test_pred)

# Print the evaluation results
print(f"Validation Set - MSE: {val_mse:.2f}, R²: {val_r2:.2f}")
print(f"Test Set - MSE: {test_mse:.2f}, R²: {test_r2:.2f}")
```

Step 6: Model Deployment Using IPython Widgets

Prompt-Step6: *"Let's go to Step 6 and deploy the model using Jupyter Notebook (with IPython Widgets) for a simple demonstration. Make sure to use a new scaler based on the selected features."*

Key Changes:

- IPython Widgets** were used to deploy the model.
- A new **scaler** was created specifically for the **selected features**.

Code-Step6:

```
python

# Import necessary libraries
import ipywidgets as widgets
from IPython.display import display
from sklearn.preprocessing import StandardScaler
import numpy as np

# Union of important features
union_features = ['rm', 'lstat', 'dis', 'nox', 'ptratio', 'rad', 'indus', 'crim', 'b', 'tax']

# Select the relevant features from the training data for re-scaling
X_train_union = X_train.loc[:, union_features]

# Create a new scaler and fit it on the selected features
new_scaler = StandardScaler()
new_scaler.fit(X_train_union)

# Define widgets for each feature input
rm_widget = widgets.FloatSlider(value=6, min=3, max=9, step=0.1, description='Rooms (RM)')
lstat_widget = widgets.FloatSlider(value=12, min=1, max=40, step=1, description='LSTAT (%)')
dis_widget = widgets.FloatSlider(value=4, min=1, max=12, step=0.1, description='DIS')
nox_widget = widgets.FloatSlider(value=10, min=0.5, max=1.0, step=0.01, description='NOX (ppm)')
ptratio_widget = widgets.FloatSlider(value=18, min=10, max=25, step=0.1, description='PTRATIO')
rad_widget = widgets.IntSlider(value=5, min=1, max=24, step=1, description='RAD')
indus_widget = widgets.FloatSlider(value=10, min=1, max=30, step=0.5, description='INDUS')
crim_widget = widgets.FloatSlider(value=0.1, min=0, max=10, step=0.01, description='CRIM')
b_widget =
```

Boston Housing Linear Regression



Code-Step6 (continued):

python

```
b_widget = widgets.FloatSlider(value=350, min=0, max=400, step=1, description='B')
tax_widget = widgets.IntSlider(value=300, min=100, max=700, step=10, description='TAX')

# Create a button to trigger prediction
predict_button = widgets.Button(description="Predict House Price")

# Function to predict house price when button is clicked
def predict_price(button):
    # Collect input values from widgets
    input_data = [[rm_widget.value, lstat_widget.value, dis_widget.value,
                    nox_widget.value, ptratio_widget.value,
                    rad_widget.value, indus_widget.value, crim_widget.value,
                    b_widget.value, tax_widget.value]]

    # Use the new scaler to scale the input data
    scaled_input = new_scaler.transform(input_data)

    # Use the final model to predict house price
    predicted_price = final_model.predict(scaled_input)

    # Display the predicted price
    print(f"Predicted House Price: ${predicted_price[0] * 1000:.2f}")

# Attach the prediction function to the button
predict_button.on_click(predict_price)

# Display all widgets and the button in the notebook
display(rm_widget, lstat_widget, dis_widget, nox_widget, ptratio_widget, rad_widget,
        indus_widget, crim_widget, b_widget, tax_widget, predict_button)
```

Key Changes for Step 6:

1. **New Scaler:** A new scaler was created based on the **union of important features**.
2. **IPython Widgets:** Interactive widgets were deployed to collect feature values and predict house prices dynamically.

Conclusion:

This project was executed following the **CRISP-DM framework**, with each phase completed interactively through **natural language programming** (prompting and code generation). Version control was applied to keep track of prompts and code at each step, ensuring a clear path from **business understanding** to **model deployment** using **IPython Widgets** in a Jupyter Notebook environment.

If you have any further questions or need adjustments, feel free to ask!