Project ARI3205 Interpretable AI for Deep Learning Models *(Part 1.1)*

Name: Andrea Filiberto Lucas
ID No: 0279704L

Importing Necessary Libraries

```python
import json

# Read the libraries from the text file
with open('../Libraries/Part1_Lib.json', 'r') as file:
    libraries = json.load(file)

# ANSI escape codes for colored output
GREEN = "\033[92m"  # Green text
RED = "\033[91m"    # Red text
RESET = "\033[0m"   # Reset to default color

# Function to check and install libraries
def check_and_install_libraries(libraries):
    for lib, import_name in libraries.items():
        try:
            # Attempt to import the library
            __import__(import_name)
            print(f"[{GREEN}✔{RESET}] Library '{lib}' is already 
installed.")
        except ImportError:
            # If import fails, try to install the library
            print(f"[{RED}✖{RESET}] Library '{lib}' is not installed. 
Installing...")
            %pip install {lib}

# Execute the function to check and install libraries
check_and_install_libraries(libraries)

# Import necessary libraries for data analysis and modeling
import pandas as pd
# Data manipulation and analysis                        #type: ignore
import numpy as np
# Numerical computations                                #type: ignore
import matplotlib.pyplot as plt
# Data visualization                                    #type: ignore
import seaborn as sns
# Statistical data visualization                        #type: ignore
import statsmodels.formula.api as smf
# Statistical models                                    #type: ignore
```

```
from sklearn.model_selection import train_test_split
# Train-test split                                  #type: ignore
from tensorflow.keras.models import Sequential
# Neural network model                              #type: ignore
from tensorflow.keras.layers import Dense
# Neural network layers                             #type: ignore

from tensorflow.keras.optimizers import Adam
# Neural network optimizer                          #type: ignore
from sklearn.preprocessing import StandardScaler
# Data scaling                                      #type: ignore
from sklearn.impute import SimpleImputer
# Missing value imputation                          #type: ignore
from sklearn.inspection import PartialDependenceDisplay,
permutation_importance     # Feature importance
#type: ignore
from sklearn.neural_network import MLPRegressor
# Neural network model                              #type: ignore
from sklearn.metrics import mean_squared_error
# Model evaluation                                  #type: ignore
from alibi.explainers import ALE, plot_ale
# ALE plots                                         #type: ignore

# Suppress specific warnings
import warnings
warnings.filterwarnings("ignore", message="X does not have valid
feature names")
```

```
[✔] Library 'tensorflow' is already installed.
[✔] Library 'scikit-learn' is already installed.
[✔] Library 'matplotlib' is already installed.
[✔] Library 'seaborn' is already installed.
[✔] Library 'pandas' is already installed.
[✔] Library 'numpy' is already installed.
[✔] Library 'scipy' is already installed.
[✔] Library 'alibi' is already installed.
```

General Information on Boston Housing Dataset

https://www.kaggle.com/datasets/altavish/boston-housing-dataset/data

```
# Define the filename
filename = '../Datasets/Boston/Boston.csv'

# Load the dataset
try:
    boston_data = pd.read_csv(filename)
    print(f"'{filename}' dataset loaded successfully.")
except FileNotFoundError:
```

```
    print(f"Error: The file '{filename}' was not found. Please ensure
it is in the correct directory.")
    exit()
except pd.errors.EmptyDataError:
    print(f"Error: The file '{filename}' is empty.")
    exit()
except pd.errors.ParserError:
    print(f"Error: There was a problem parsing '{filename}'. Please
check the file format.")
    exit()

# Dataset insights
print("\nDataset Overview:")
print(boston_data.info())
print("\nStatistical Summary:")
print(boston_data.describe())
```

```
'../Datasets/Boston/Boston.csv' dataset loaded successfully.

Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     486 non-null    float64
 1   ZN       486 non-null    float64
 2   INDUS    486 non-null    float64
 3   CHAS     486 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      486 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    486 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
None

Statistical Summary:
             CRIM          ZN       INDUS        CHAS         NOX
RM  \
count  486.000000  486.000000  486.000000  486.000000  506.000000
506.000000
mean     3.611874   11.211934   11.083992    0.069959    0.554695
6.284634
```

```
std       8.720192   23.388876    6.835896    0.255340    0.115878
0.702617
min       0.006320    0.000000    0.460000    0.000000    0.385000
3.561000
25%       0.081900    0.000000    5.190000    0.000000    0.449000
5.885500
50%       0.253715    0.000000    9.690000    0.000000    0.538000
6.208500
75%       3.560263   12.500000   18.100000    0.000000    0.624000
6.623500
max      88.976200  100.000000   27.740000    1.000000    0.871000
8.780000

              AGE         DIS         RAD         TAX     PTRATIO
B  \
count  486.000000  506.000000  506.000000  506.000000  506.000000
506.000000
mean    68.518519    3.795043    9.549407  408.237154   18.455534
356.674032
std     27.999513    2.105710    8.707259  168.537116    2.164946
91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000
0.320000
25%     45.175000    2.100175    4.000000  279.000000   17.400000
375.377500
50%     76.800000    3.207450    5.000000  330.000000   19.050000
391.440000
75%     93.975000    5.188425   24.000000  666.000000   20.200000
396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000
396.900000

            LSTAT        MEDV
count  486.000000  506.000000
mean    12.715432   22.532806
std      7.155871    9.197104
min      1.730000    5.000000
25%      7.125000   17.025000
50%     11.430000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

Feed-Forward Neural Network

```
# Separate features and target
X = boston_data.drop(columns=['MEDV'])  # Features
y = boston_data['MEDV']  # Target

# Handle missing values with mean imputation
```

```
imputer = SimpleImputer(strategy='mean')
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)

# Standardize the features
scaler = StandardScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X_imputed),
columns=X.columns)

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)
print("Training data shape:", X_train.shape)
print("Test data shape:", X_test.shape)
```

```
Training data shape: (404, 13)
Test data shape: (102, 13)
```

```
# Build the feed-forward neural network
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1)  # Output layer for regression
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='mse',
metrics=['mae'])

# Train the model
history = model.fit(X_train, y_train, validation_split=0.2, epochs=50,
batch_size=32, verbose=1)

# Evaluate the model
test_loss, test_mae = model.evaluate(X_test, y_test, verbose=1)
print(f"Test Loss: {test_loss:.4f}, Test MAE: {test_mae:.4f}")
```

```
/opt/anaconda3/lib/python3.11/site-packages/keras/src/layers/core/
dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim`
argument to a layer. When using Sequential models, prefer using an
`Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer,
**kwargs)
```

```
Epoch 1/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 610.9976 - mae:
22.8815 - val_loss: 539.9886 - val_mae: 21.7096
Epoch 2/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 612.5311 - mae: 22.7226
- val_loss: 507.0454 - val_mae: 20.9489
Epoch 3/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 536.2062 - mae: 21.3987
```

```
- val_loss: 470.3633 - val_mae: 20.0871
Epoch 4/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 487.4485 - mae: 20.1953
- val_loss: 425.6573 - val_mae: 19.0079
Epoch 5/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 439.9500 - mae: 19.0457
- val_loss: 372.1023 - val_mae: 17.6258
Epoch 6/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 383.8455 - mae: 17.3025
- val_loss: 309.6705 - val_mae: 15.8827
Epoch 7/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 310.1775 - mae: 15.6345
- val_loss: 241.4313 - val_mae: 13.8186
Epoch 8/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 243.0562 - mae: 13.4235
- val_loss: 176.3131 - val_mae: 11.5594
Epoch 9/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 173.2832 - mae: 11.1322
- val_loss: 118.8051 - val_mae: 9.0884
Epoch 10/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 127.6825 - mae: 8.9653
- val_loss: 77.8989 - val_mae: 6.9037
Epoch 11/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 83.9577 - mae: 7.2203 -
val_loss: 56.1831 - val_mae: 5.5618
Epoch 12/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 67.8990 - mae: 6.5558 -
val_loss: 45.2629 - val_mae: 4.8866
Epoch 13/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 61.0328 - mae: 5.9132 -
val_loss: 38.2517 - val_mae: 4.4154
Epoch 14/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 49.0374 - mae: 5.5509 -
val_loss: 33.7060 - val_mae: 4.1265
Epoch 15/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 42.0957 - mae: 4.9648 -
val_loss: 31.0094 - val_mae: 3.9563
Epoch 16/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 37.2767 - mae: 4.4003 -
val_loss: 29.3667 - val_mae: 3.8668
Epoch 17/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 31.7819 - mae: 4.2120 -
val_loss: 28.4223 - val_mae: 3.8305
Epoch 18/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 27.9506 - mae: 4.0479 -
val_loss: 28.1628 - val_mae: 3.8181
Epoch 19/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step - loss: 26.2952 - mae: 3.7631
- val_loss: 27.6732 - val_mae: 3.7870
```

```
Epoch 20/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 23.9321 - mae: 3.6995 -
val_loss: 27.5123 - val_mae: 3.7783
Epoch 21/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 25.4681 - mae: 3.7127 -
val_loss: 27.4019 - val_mae: 3.7606
Epoch 22/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 16ms/step - loss: 24.0510 - mae: 3.6659
- val_loss: 26.6372 - val_mae: 3.7184
Epoch 23/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 18.1317 - mae: 3.2892 -
val_loss: 26.2528 - val_mae: 3.6857
Epoch 24/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 20.0552 - mae: 3.3898 -
val_loss: 26.0505 - val_mae: 3.6825
Epoch 25/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 23.3051 - mae: 3.3978 -
val_loss: 25.6083 - val_mae: 3.6522
Epoch 26/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 20.8880 - mae: 3.3739 -
val_loss: 24.9881 - val_mae: 3.5586
Epoch 27/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 19.5687 - mae: 3.3424 -
val_loss: 24.4712 - val_mae: 3.5129
Epoch 28/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 20.0316 - mae: 3.3424 -
val_loss: 24.1334 - val_mae: 3.4745
Epoch 29/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step - loss: 16.8976 - mae: 3.0855
- val_loss: 23.7792 - val_mae: 3.4335
Epoch 30/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 16.8489 - mae: 3.0665 -
val_loss: 23.6043 - val_mae: 3.4182
Epoch 31/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 19.3166 - mae: 3.1728
- val_loss: 23.2198 - val_mae: 3.4048
Epoch 32/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 21.4817 - mae: 3.1872 -
val_loss: 22.9691 - val_mae: 3.3908
Epoch 33/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.6022 - mae: 3.1756 -
val_loss: 23.1211 - val_mae: 3.3862
Epoch 34/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 18.8767 - mae: 3.0806 -
val_loss: 23.0233 - val_mae: 3.3929
Epoch 35/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - loss: 16.3944 - mae: 2.9686
- val_loss: 22.6667 - val_mae: 3.3721
Epoch 36/50
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 18.7178 - mae: 3.2670 -
val_loss: 22.1120 - val_mae: 3.3367
Epoch 37/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 16.5506 - mae: 3.0255 -
val_loss: 21.8547 - val_mae: 3.3120
Epoch 38/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 19.5920 - mae: 3.2243 -
val_loss: 21.2016 - val_mae: 3.2525
Epoch 39/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 17.1421 - mae: 3.0439
- val_loss: 20.7961 - val_mae: 3.2071
Epoch 40/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 14.4766 - mae: 2.8270 -
val_loss: 20.5739 - val_mae: 3.1850
Epoch 41/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.1502 - mae: 2.8702 -
val_loss: 20.6122 - val_mae: 3.2112
Epoch 42/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - loss: 14.2076 - mae: 2.7147
- val_loss: 20.5578 - val_mae: 3.2263
Epoch 43/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 16.2328 - mae: 2.9353 -
val_loss: 20.0096 - val_mae: 3.1807
Epoch 44/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 15.4274 - mae: 2.8205 -
val_loss: 19.6446 - val_mae: 3.1490
Epoch 45/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 16.5106 - mae: 2.8110 -
val_loss: 19.5938 - val_mae: 3.1421
Epoch 46/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 14.8909 - mae: 2.7844
- val_loss: 19.7601 - val_mae: 3.1517
Epoch 47/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 14.3406 - mae: 2.7273 -
val_loss: 20.1172 - val_mae: 3.1564
Epoch 48/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 12.7872 - mae: 2.6688 -
val_loss: 19.4098 - val_mae: 3.1072
Epoch 49/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - loss: 12.6852 - mae: 2.6272 -
val_loss: 18.8763 - val_mae: 3.0776
Epoch 50/50
11/11 ━━━━━━━━━━━━━━━━━━━━ 0s 11ms/step - loss: 14.3075 - mae: 2.7343
- val_loss: 18.8568 - val_mae: 3.0921
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - loss: 11.2812 - mae: 2.2573
Test Loss: 15.3235, Test MAE: 2.3687
```

Surrogate Model - MLPRegressor

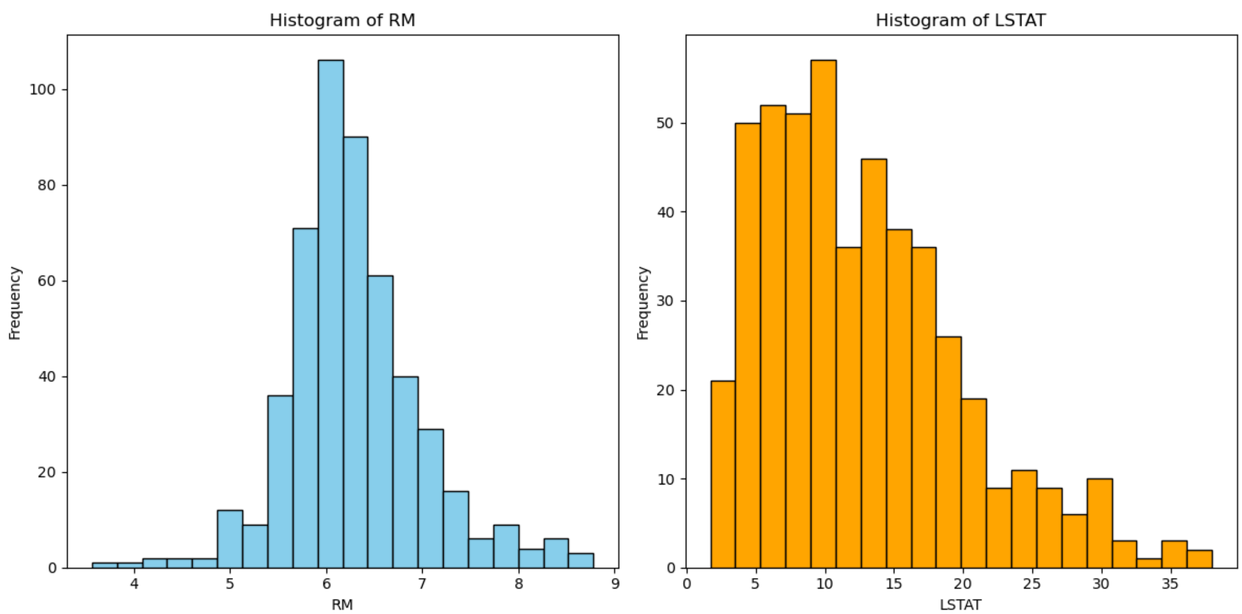
```
# Train an MLPRegressor as a surrogate model
surrogate_model = MLPRegressor(hidden_layer_sizes=(64, 32),
max_iter=1000, random_state=42)
surrogate_model.fit(X_train, y_train)

# Evaluate the surrogate model
y_pred = surrogate_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Surrogate Model Mean Squared Error: {mse:.4f}")
```

```
Surrogate Model Mean Squared Error: 12.7475
```

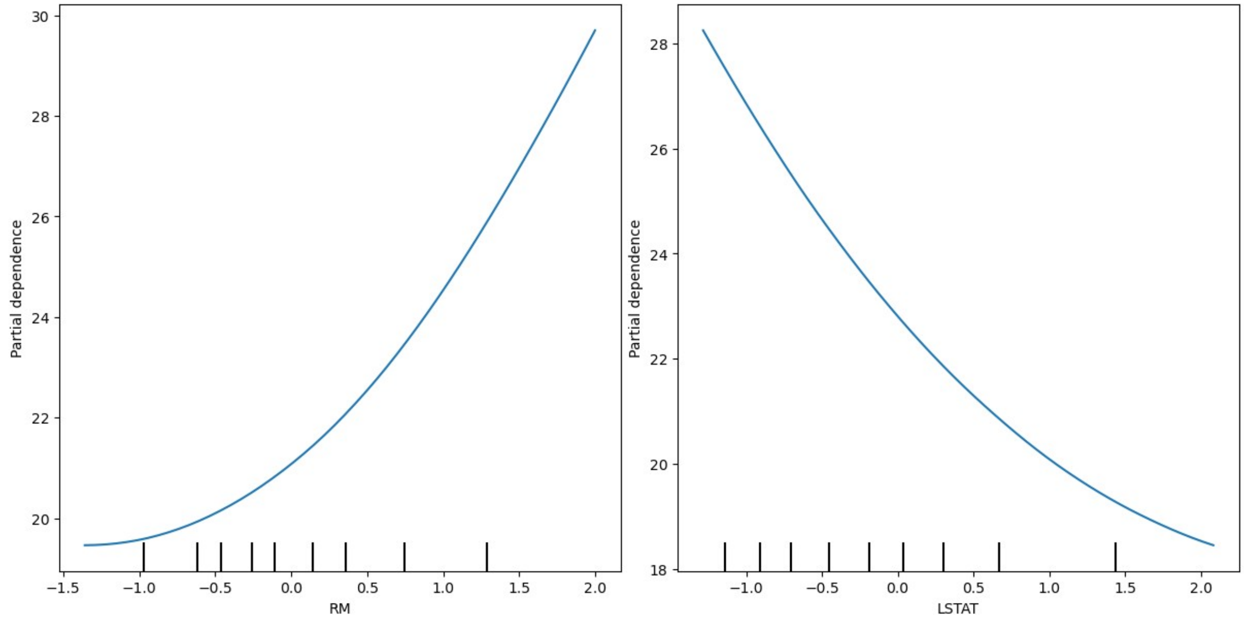
Partial Dependence Plots (PDP) and Individual Conditional Expectation (ICE) plots

```
# Partial Dependence Plots (PDP)
def plot_pdp(features):
    print("\nGenerating PDP for features:", features)
    fig, ax = plt.subplots(1, len(features), figsize=(12, 6),
constrained_layout=True)
    for i, feature in enumerate(features):
        PartialDependenceDisplay.from_estimator(
            surrogate_model,  # The trained surrogate model
(MLPRegressor)
            X_train,          # Training data
            features=[feature],  # Single feature for PDP
            kind="average",   # PDP only
            ax=ax[i] if len(features) > 1 else ax,
            grid_resolution=50,
        )
        ax[i].set_title(f"PDP for {feature}")
    plt.show()

# Individual Conditional Expectation (ICE) Plots
def plot_ice(features):
    print("\nGenerating ICE for features:", features)
    fig, ax = plt.subplots(1, len(features), figsize=(12, 6),
constrained_layout=True)
    for i, feature in enumerate(features):
        PartialDependenceDisplay.from_estimator(
            surrogate_model,  # The trained surrogate model
(MLPRegressor)
            X_train,          # Training data
            features=[feature],  # Single feature for ICE
            kind="both",      # PDP and ICE
            ax=ax[i] if len(features) > 1 else ax,
            grid_resolution=50,
```

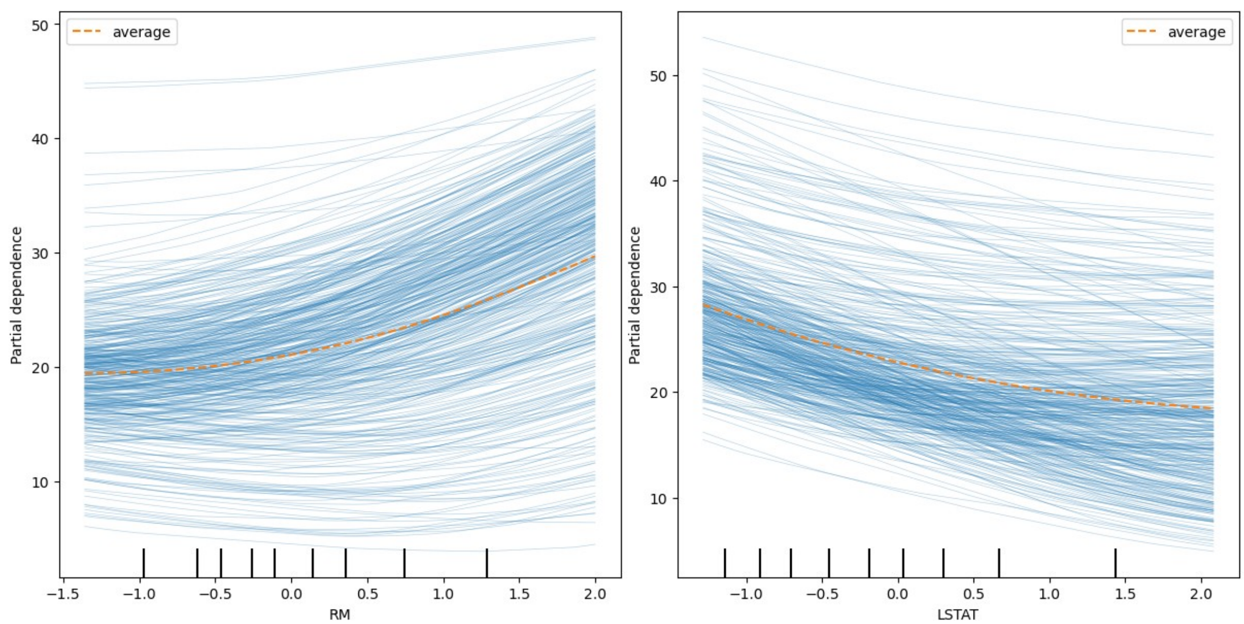
```
        )
        ax[i].set_title(f"ICE and PDP for {feature}")
    plt.show()

# Call PDP and ICE plot functions
features_to_analyze = ['RM', 'LSTAT']

# Plot histograms for features_to_analyze
plt.figure(figsize=(12, 6))
for i, feature in enumerate(features_to_analyze):
    plt.subplot(1, len(features_to_analyze), i + 1)
    plt.hist(boston_data[feature], bins=20, edgecolor='black',
color='skyblue' if i % 2 == 0 else 'orange')
    plt.title(f'Histogram of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()

plot_pdp(features_to_analyze)
plot_ice(features_to_analyze)
```




```
Generating PDP for features: ['RM', 'LSTAT']
```




```
Generating ICE for features: ['RM', 'LSTAT']
```



Explain what insights PDP and ICE give about the model's behaviour.

The **Partial Dependence Plots (PDPs)** and **Individual Conditional Expectation (ICE) plots** for the features `RM` (average number of rooms per dwelling) and `LSTAT` (percentage of lower-status population) provide critical insights into the predictive behavior of the trained model. These features were selected due to their strong correlation with the target variable, `MEDV` (median house prices), and their contrasting trends, which are both relevant and interpretable in the context of housing prices.

Partial Dependence Plots (PDP)

PDPs offer a global perspective on the relationship between features and the predicted target variable. The PDP for `RM` shows a **positive monotonic relationship**, indicating that an increase in the number of rooms correlates with higher predicted house prices. This trend aligns with real-world expectations, as larger homes are typically associated with higher market values. On the other hand, the PDP for `LSTAT` reveals a **negative monotonic relationship**, suggesting that areas with a higher percentage of lower-status populations are associated with lower predicted house prices. This reflects the model's ability to capture the socioeconomic factors influencing housing prices effectively.

While PDPs are excellent for understanding the overall trends, they average the effects across all data points and fail to capture individual variations or interactions between features. Thus, they provide a general but limited view of feature importance.

Individual Conditional Expectation (ICE) Plots

ICE plots complement PDPs by revealing how predictions vary for individual instances when the feature values change. For `RM`, the ICE plots demonstrate that most individual instances follow the positive trend shown in the PDP. However, there are subtle variations in the slopes of individual lines, indicating that the sensitivity of predictions to `RM` differs across instances. For example, homes in different neighborhoods or price ranges might respond differently to changes in the number of rooms. Similarly, for `LSTAT`, the ICE plots reveal a consistent negative slope across most instances, consistent with the PDP. However, the variability in slopes highlights heterogeneous relationships, where certain neighborhoods or homes might be less affected by changes in the percentage of lower-status populations.

ICE plots are especially valuable for identifying outliers or subgroups that deviate from the average behavior. They provide a granular view, offering insights into how specific instances behave, which is critical for understanding model predictions at an individual level.

Insights from Combining PDP and ICE

The combined analysis of PDPs and ICE plots offers a comprehensive understanding of the model's behavior. PDPs provide a **global average perspective**, while ICE plots uncover **instance-level variability** and highlight heterogeneous effects. For example, while the global trend for `RM` suggests a steady increase in house prices with more rooms, the ICE plots reveal that the degree of sensitivity varies across instances. Similarly, for `LSTAT`, while the global trend shows a decrease in prices with higher percentages of lower-status populations, the ICE plots expose differences in how sensitive various neighborhoods are to this feature.

Key Takeaways

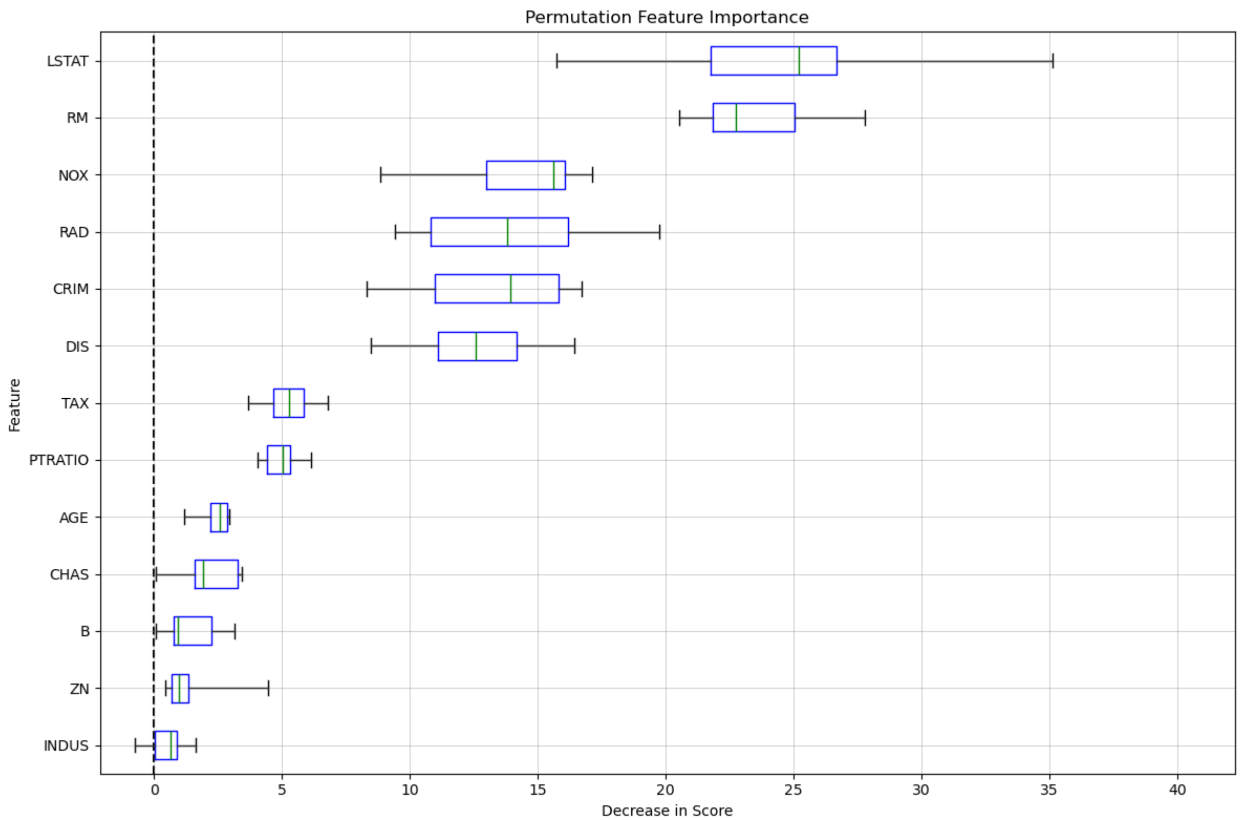
The analysis of `RM` and `LSTAT` underscores their relevance to housing prices. The positive relationship of `RM` with house prices reflects the impact of housing size and quality, while the negative relationship of `LSTAT` highlights the influence of socioeconomic factors. By leveraging both PDPs and ICE plots, we gain a robust understanding of the model's predictions, ensuring that they align with real-world expectations while identifying potential areas for improvement. Together, these tools enhance interpretability, providing both broad insights and detailed individual-level analysis.

Permutation Feature Importance (PFI)

```
# Compute Permutation Feature Importance
def compute_pfi(model, X_test, y_test, feature_names):
    pfi_result = permutation_importance(model, X_test, y_test,
n_repeats=10, random_state=42, scoring='neg_mean_squared_error')

    # Convert PFI results into a DataFrame for better visualization
    importance_df = pd.DataFrame({
        'Feature': feature_names,
        'Importance': pfi_result.importances_mean,
        'Std': pfi_result.importances_std
    })

    # Sort features by importance
    importance_df = importance_df.sort_values(by='Importance',
ascending=False)

    print("\nPermutation Feature Importance:\n", importance_df)
    return importance_df

# Plot Permutation Feature Importance as a Boxplot
def plot_pfi(model, X, y, feature_names):
    result = permutation_importance(model, X, y,
scoring='neg_mean_squared_error', n_repeats=10, random_state=42,
n_jobs=2)

    sorted_importances_idx = result.importances_mean.argsort()
    importances =
pd.DataFrame(result.importances[sorted_importances_idx].T,
                               columns=[feature_names[i] for i in
sorted_importances_idx])

    ax = importances.plot.box(vert=False, whis=10, figsize=(12, 8),
color=dict(boxes="blue", whiskers="black", medians="green",
caps="black"))
    ax.axvline(x=0, color="k", linestyle="--")

    # Add faint grey lines across the graph for each feature
    for i in range(len(importances.columns)):
        plt.axhline(y=i + 1, color="grey", linestyle="-",
linewidth=0.5, alpha=0.5)

    # Add faint grey lines upwards from the x-axis ticks
    xticks = ax.get_xticks()
    for tick in xticks:
        plt.axvline(x=tick, color="grey", linestyle="-",
linewidth=0.5, alpha=0.5)

    # Set the x-axis limits
```

```
    ax.set_xlim(left=0 - 0.05 * (ax.get_xlim()[1] - 0))

    ax.set_xlabel("Decrease in Score")
    ax.set_ylabel("Feature")
    ax.set_title("Permutation Feature Importance")
    plt.tight_layout()
    plt.show()

feature_names = X_test.columns
importance_df = compute_pfi(surrogate_model, X_test, y_test,
feature_names)
plot_pfi(surrogate_model, X_test, y_test, feature_names)
```

```
Permutation Feature Importance:
     Feature  Importance       Std
12     LSTAT   24.594452  4.819524
5         RM   23.492247  2.306411
4        NOX   14.488889  2.533761
8        RAD   13.979112  3.490214
0       CRIM   13.408561  2.758039
7        DIS   12.747733  2.383452
9        TAX    5.263374  0.893353
10   PTRATIO    4.990779  0.644088
6        AGE    2.380155  0.586128
3       CHAS    2.149071  1.070397
11         B    1.384748  1.012892
1         ZN    1.331404  1.110311
2      INDUS    0.470038  0.717250
```



Permutation Feature Importance Results

The **Permutation Feature Importance (PFI)** results, visualized in the boxplot above, rank features based on their impact on the model's predictions for `MEDV` (median house prices). `LSTAT` (percentage of lower-status population) emerges as the most critical feature with a mean importance score of 24.59 and a standard deviation of 4.82. This underscores its significant negative influence on housing prices, aligning with socioeconomic realities. `RM` (average number of rooms per dwelling) follows closely, with an importance score of 23.49 and a lower standard deviation of 2.31, reflecting its strong positive correlation with property values.

Other influential features include `NOX` (nitric oxide concentration), `RAD` (accessibility to radial highways), and `CRIM` (per capita crime rate), which reflect environmental and neighborhood-related factors affecting housing prices, with importance scores of 14.49, 13.98, and 13.41, respectively. Conversely, features like `PTRATIO`, `AGE`, and `CHAS` show relatively lower importance, suggesting limited predictive value, while `INDUS` and `ZN` have the least influence, indicating minimal relevance to the model.

The boxplot highlights variability in feature importance scores across permutations. `LSTAT` and `RM` show compact whiskers, indicating consistent importance, while features like `CHAS` display greater variability, suggesting more context-dependent contributions. This ranking and variability provide a clear understanding of which features are robustly influential and which may have situational relevance.

Explain what the term “important” means when using the PFI method.

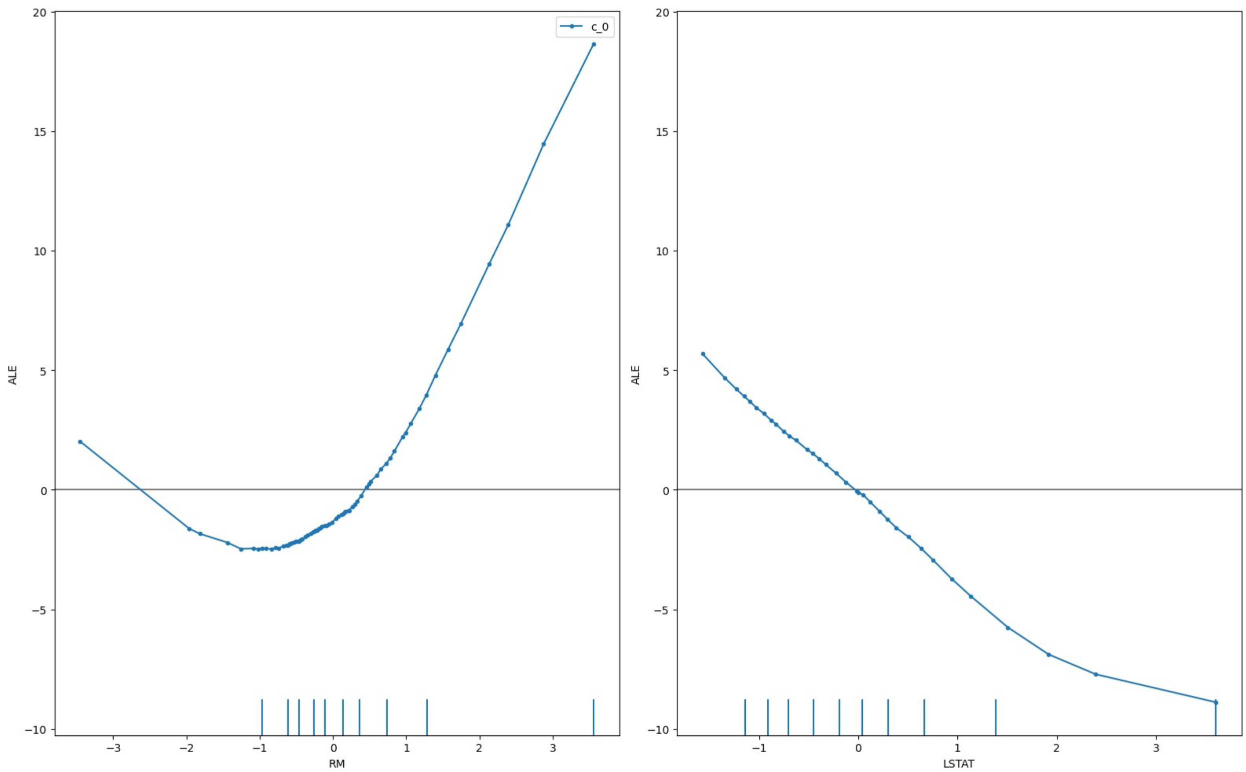
In the PFI method, **importance** quantifies the contribution of a feature to the model's predictive performance. This is measured by observing the increase in prediction error when a feature's values are permuted randomly while keeping other features unchanged. A higher importance score indicates that randomization significantly degrades the model's accuracy, implying that the feature provides critical information. Conversely, a lower score suggests that the feature's randomization has minimal effect, reflecting limited predictive value.

For the given results, the high importance scores of `LSTAT` and `RM` illustrate their dominant role in capturing critical factors like socioeconomic status and home size, directly influencing housing prices. The consistent importance (low standard deviations) of these features indicates their robust and global relevance across the dataset. In contrast, the low scores for `INDUS` and `ZN` show that these features contribute minimally to the model's predictions. The variability in features like `CHAS` highlights that their importance may vary depending on specific data subsets or interactions with other features. This demonstrates how PFI captures both direct and indirect feature effects, offering a comprehensive view of feature relevance.

Accumulated Local Effects (ALE)

```
# Combine features and target for context if needed
data = pd.concat([X_train, y_train], axis=1)
# Define feature names
feature_names = X_train.columns

# Ensure valid input for ALE explainer
X_train_array = X_train.to_numpy()  # Convert to NumPy array to avoid
warnings

# Create and compute ALE explainer
ale_explainer = ALE(surrogate_model.predict,
feature_names=feature_names)
ale_explanation = ale_explainer.explain(X_train_array)

# Plot ALE for all features
plot_ale(
    ale_explanation,
    features=['RM', 'LSTAT'], # Select specific features
    n_cols=4,  # Arrange plots in 4 columns for better visualization
    fig_kw={'figwidth': 16, 'figheight': 10}  # Adjust figure size for
clarity
)
```

```
array([[<Axes: xlabel='RM', ylabel='ALE'>,
        <Axes: xlabel='LSTAT', ylabel='ALE'>]], dtype=object)
```



Comparing ALE and PDP for `RM` and `LSTAT`

The **Accumulated Local Effects (ALE)** plots and **Partial Dependence Plots (PDPs)** offer valuable insights into the relationships between features and the target variable (`MEDV`, median house prices). Both techniques aim to interpret model behavior but differ in their methodologies, which is evident when examining the features `RM` (average number of rooms per dwelling) and `LSTAT` (percentage of lower-status population).

The ALE plots for `RM` and `LSTAT` provide a localized view of feature effects by calculating the average change in predictions within intervals of the feature values. For `RM`, the ALE plot reveals a strong positive and nonlinear relationship, with house prices increasing steeply as the number of rooms rises, particularly for higher values of `RM`. Interestingly, the plot also highlights a slight dip for lower values of `RM`, indicating a small negative impact on house prices in cases of very small homes before the overall upward trend dominates. For `LSTAT`, the ALE plot shows a consistent negative relationship across the feature range. Housing prices decline as the percentage of lower-status populations increases, with the effect intensifying at higher values of `LSTAT`. These localized trends highlight how the model captures nuanced relationships that vary across different feature ranges.

In comparison, the PDPs for `RM` and `LSTAT` provide a global perspective by showing the average effect of each feature across all instances. For `RM`, the PDP depicts a smooth monotonic increase, reinforcing the positive association between the number of rooms and housing prices. Similarly, the PDP for `LSTAT` demonstrates a monotonic decline, confirming that higher percentages of lower-status populations are correlated with lower house prices. However, unlike ALE, the PDPs do not capture the subtle dip observed for lower values of `RM`. This is because PDPs average predictions across the dataset, which can smooth out localized variations and obscure interactions between features.

The primary distinction between ALE and PDP lies in their interpretability. PDPs provide a straightforward global view, offering an easy-to-understand summary of feature effects. However, they may be influenced by feature correlations, as the marginalization process does not account for dependencies between features. In contrast, ALE plots focus on localized effects and are more robust to feature correlations. They offer a clearer picture of nonlinear relationships and feature interactions, as evidenced by the additional details visible in the ALE plot for `RM`.

Both techniques agree on the general trends for `RM` and `LSTAT`: `RM` has a positive impact on housing prices, while `LSTAT` has a negative influence. However, the ALE plots add granularity by capturing localized behaviors and variations that the PDPs overlook. Together, these methods complement each other, with PDPs providing a broad overview and ALE offering detailed insights into localized effects, enabling a deeper understanding of the model's behavior.

Global Surrogates

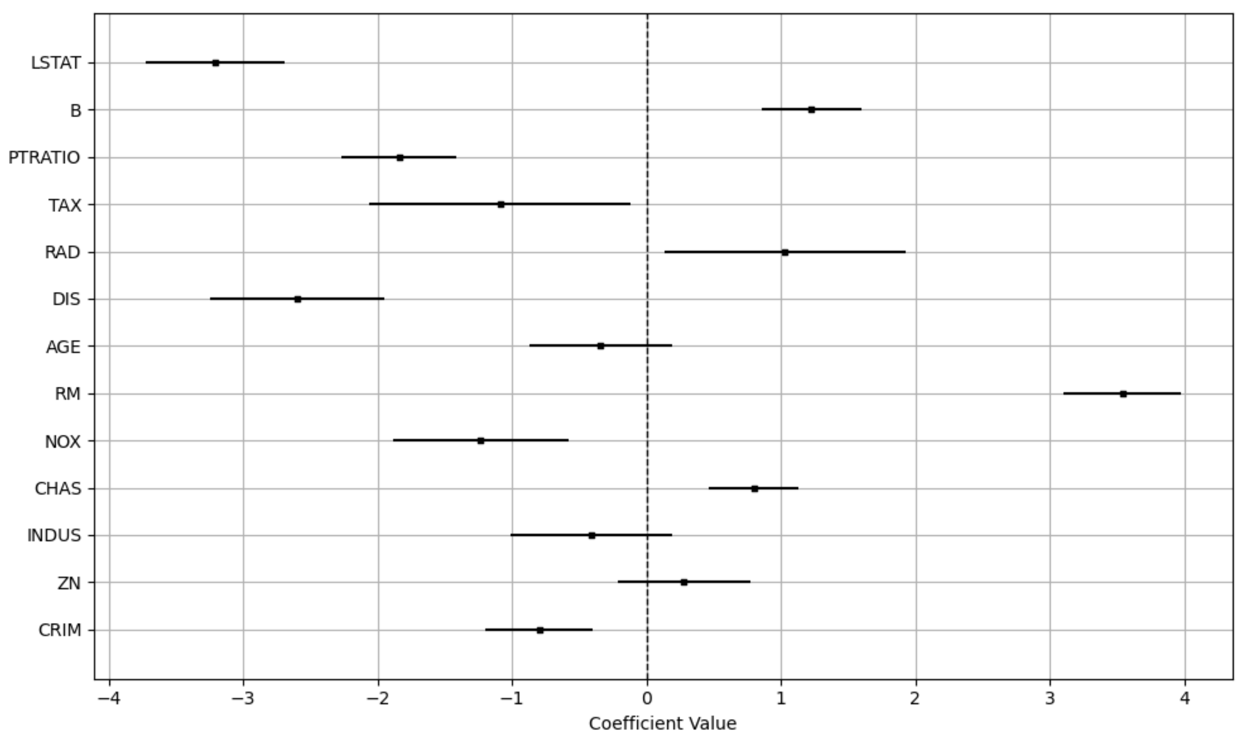
```
# Generate predictions from the neural network
NN_labels = model.predict(X_train).flatten()
X_train['NN_labels'] = NN_labels

# Train an interpretable linear regression model
formula = 'NN_labels ~ ' + ' + '.join(X_train.columns[:-1])  # Include all features in the formula
lin_reg = smf.ols(formula=formula, data=X_train).fit()
print(lin_reg.summary())
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step
                            OLS Regression Results

==============================================================================
Dep. Variable:              NN_labels   R-squared:
0.865
Model:                            OLS   Adj. R-squared:
0.861
Method:                 Least Squares   F-statistic:
192.2
Date:                Fri, 17 Jan 2025   Prob (F-statistic):
1.75e-160
Time:                        16:47:05   Log-Likelihood:
-1047.3
No. Observations:                 404   AIC:
2123.
Df Residuals:                     390   BIC:
2179.
Df Model:                          13
```

```
Covariance Type:            nonrobust

==============================================================================
========
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
--------
Intercept     22.4288      0.164    136.478      0.000      22.106
22.752
CRIM          -0.7999      0.204     -3.931      0.000      -1.200
-0.400
ZN             0.2769      0.251      1.105      0.270      -0.216
0.770
INDUS         -0.4080      0.305     -1.339      0.181      -1.007
0.191
CHAS           0.7963      0.168      4.739      0.000       0.466
1.127
NOX           -1.2324      0.332     -3.709      0.000      -1.886
-0.579
RM             3.5376      0.221     15.982      0.000       3.102
3.973
AGE           -0.3417      0.271     -1.262      0.208      -0.874
0.191
DIS           -2.5984      0.328     -7.919      0.000      -3.243
-1.953
RAD            1.0275      0.456      2.251      0.025       0.130
1.925
TAX           -1.0905      0.493     -2.212      0.028      -2.060
-0.121
PTRATIO       -1.8416      0.216     -8.539      0.000      -2.266
-1.418
B              1.2243      0.189      6.494      0.000       0.854
1.595
LSTAT         -3.2076      0.263    -12.197      0.000      -3.725
-2.691
==============================================================================
========
Omnibus:                       63.781   Durbin-Watson:
2.068
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
121.955
Skew:                           0.880   Prob(JB):
3.30e-27
Kurtosis:                       5.036   Cond. No.
9.69
==============================================================================
========
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is
correctly specified.
```

```python
# Extract coefficients and confidence intervals
err_series = lin_reg.params - lin_reg.conf_int()[0]
coef_df = pd.DataFrame({
    'coef': pd.to_numeric(lin_reg.params.values[1:], errors='coerce'),
    'err': pd.to_numeric(err_series.values[1:], errors='coerce'),
    'varname': err_series.index.values[1:]
})

# Visualize the coefficients and confidence intervals
fig, ax = plt.subplots(figsize=(10, 6))
ax.barh(coef_df['varname'], coef_df['coef'], xerr=coef_df['err'],
color='none', edgecolor=None)
ax.scatter(y=coef_df['varname'], x=coef_df['coef'], marker='s', s=10,
color='black')
ax.axvline(x=0, linestyle='--', color='black', linewidth=1)

ax.set_xlabel('Coefficient Value')
ax.set_ylabel('')
ax.grid(True)

plt.tight_layout()
plt.show()
```



Analyse the surrogate model's effectiveness and discuss when such approximations are helpful.

The surrogate model, represented by a linear regression trained on the predictions of the neural network, demonstrates strong performance, with an R-squared value of 0.865. This suggests that 86.5% of the variance in the neural network's predictions (`NN_labels`) is captured by the linear regression model. The feature coefficients provide interpretable insights into the relationships between predictors and predictions, which are visualized in the coefficient plot. For example, `LSTAT` (percentage of lower-status population) shows the strongest negative relationship with a coefficient of -3.21, while `RM` (average number of rooms per dwelling) exhibits the strongest positive relationship with a coefficient of 3.54. These findings are consistent with prior domain knowledge, further validating the surrogate model's ability to approximate the behavior of the original neural network.

However, it is essential to recognize the limitations of such approximations, particularly in the context of these results. While the linear regression surrogate successfully captures the general trends of the neural network, it may oversimplify complex nonlinear interactions or dependencies between features. For instance, the neural network may model intricate relationships between features like `NOX` (nitric oxide concentration) and `DIS` (distance to employment centers) that are not reflected in the linear coefficients. This potential oversimplification becomes evident when considering features with weaker coefficients, such as `INDUS` (proportion of non-retail business acres) or `ZN` (proportion of residential land zoned for large lots). The neural network might account for interactions or nonlinearities involving these features, but the surrogate model reduces them to linear, independent contributions.

The visualized coefficients further support this observation. Features like `PTRATIO` (pupil-teacher ratio) and `TAX` (property tax rate), which have significant but modest coefficients, might have interactions with other variables that the linear model cannot capture. This limitation underscores the risk of misinterpretation if the surrogate model is used as the sole explanation of the neural network's behavior. For example, while `CHAS` (proximity to the Charles River) has a positive coefficient in the surrogate model, its influence in the neural network may be conditional on other features, such as `RM` or `DIS`.

In conclusion, surrogate models like the linear regression used here are valuable for enhancing interpretability while preserving much of the predictive power of the original model. They are particularly effective in distilling insights from complex models into an accessible form, as seen with the clear contributions of `LSTAT` and `RM` to predictions. However, these approximations come with trade-offs. Care must be taken to communicate that the linear regression model provides a simplified view of the neural network's behavior, and its insights should be complemented with other interpretability techniques to ensure a more comprehensive understanding of the model's decision-making process.