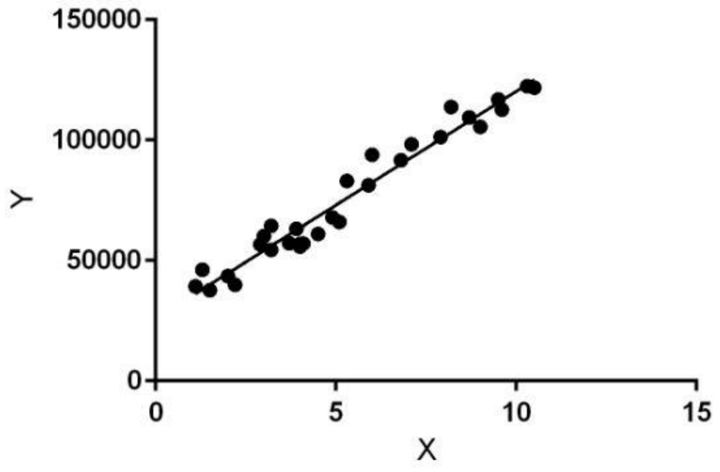
Experiment No. 1
Analyze the Boston Housing dataset and Apply appropriate Regression Technique
Date of Performance:
Date of Submission:

Aim: Analyze the Boston Housing dataset and Apply appropriate Regression Technique.

Objective: Ablility to perform various feature engineering tasks, apply linear regression on the given dataset and minimise the error.

Theory:

Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables they are considering, and the number of independent variables getting used.



Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

Dataset:

The Boston Housing Dataset

The Boston Housing Dataset is a derived from information collected by the U.S. Census Service concerning housing in the area of Boston MA. The following describes the dataset columns:

CRIM - per capita crime rate by town
ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS - proportion of non-retail business acres per town.
CHAS - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
NOX - nitric oxides concentration (parts per 10 million)
RM - average number of rooms per dwelling
AGE - proportion of owner-occupied units built prior to 1940
DIS - weighted distances to five Boston employment centres
RAD - index of accessibility to radial highways
TAX - full-value property-tax rate per $10,000
PTRATIO - pupil-teacher ratio by town
B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
LSTAT - % lower status of the population
MEDV - Median value of owner-occupied homes in $1000's

CODE & OUTPUT:

```
import pandas as pd
import numpy as np

file_path = 'BostonHousing.csv'  # Update this if your file path is differ
ent
data = pd.read_csv(file_path)

print(data.head())

# Display information about the dataset
print(data.info())

# Check for missing values
print(data.isnull().sum())
```

```
      crim    zn  indus  chas    nox     rm   age     dis  rad  tax  ptrat
io  \
0  0.00632  18.0   2.31     0  0.538  6.575  65.2  4.0900    1  296     15
.3
1  0.02731   0.0   7.07     0  0.469  6.421  78.9  4.9671    2  242     17
.8
2  0.02729   0.0   7.07     0  0.469  7.185  61.1  4.9671    2  242     17
.8
3  0.03237   0.0   2.18     0  0.458  6.998  45.8  6.0622    3  222     18
.7
4  0.06905   0.0   2.18     0  0.458  7.147  54.2  6.0622    3  222     18
.7

        b  lstat  medv
0  396.90   4.98  24.0
1  396.90   9.14  21.6
2  392.83   4.03  34.7
3  394.63   2.94  33.4
4  396.90   5.33  36.2
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   indus    506 non-null    float64
 3   chas     506 non-null    int64
 4   nox      506 non-null    float64
 5   rm       501 non-null    float64
 6   age      506 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    506 non-null    float64
 13  medv     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
None
crim       0
zn         0
indus      0
chas       0
nox        0
rm         5
age        0
dis        0
rad        0
tax        0
```

```
ptratio    0
b          0
lstat      0
medv       0
dtype: int64
```

```
data = data.dropna()
```

```
# Initial linear regression with all parameters
X_all = data.drop(columns=['medv'])
y_all = data['medv']
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

X_train_all, X_test_all, y_train_all, y_test_all = train_test_split(X_all,
 y_all, test_size=0.2, random_state=42)

# Initialize the Linear Regression model
model_all = LinearRegression()

# Train the model
model_all.fit(X_train_all, y_train_all)
```

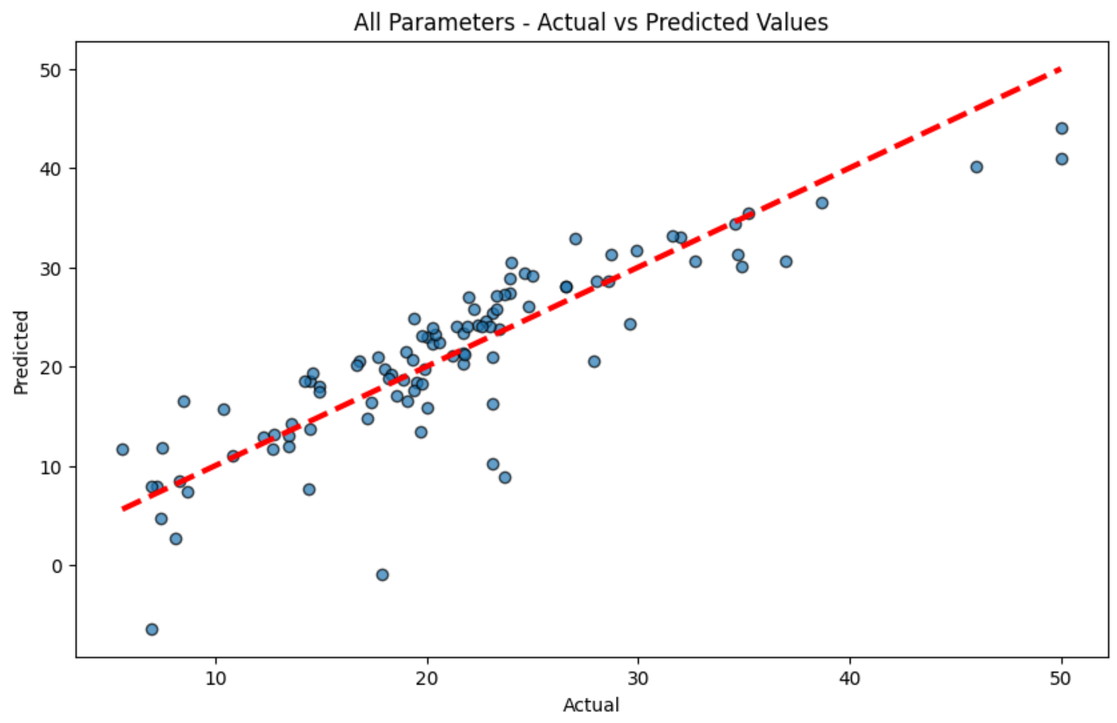
```
LinearRegression()
```

```
y_pred_all = model_all.predict(X_test_all)
```

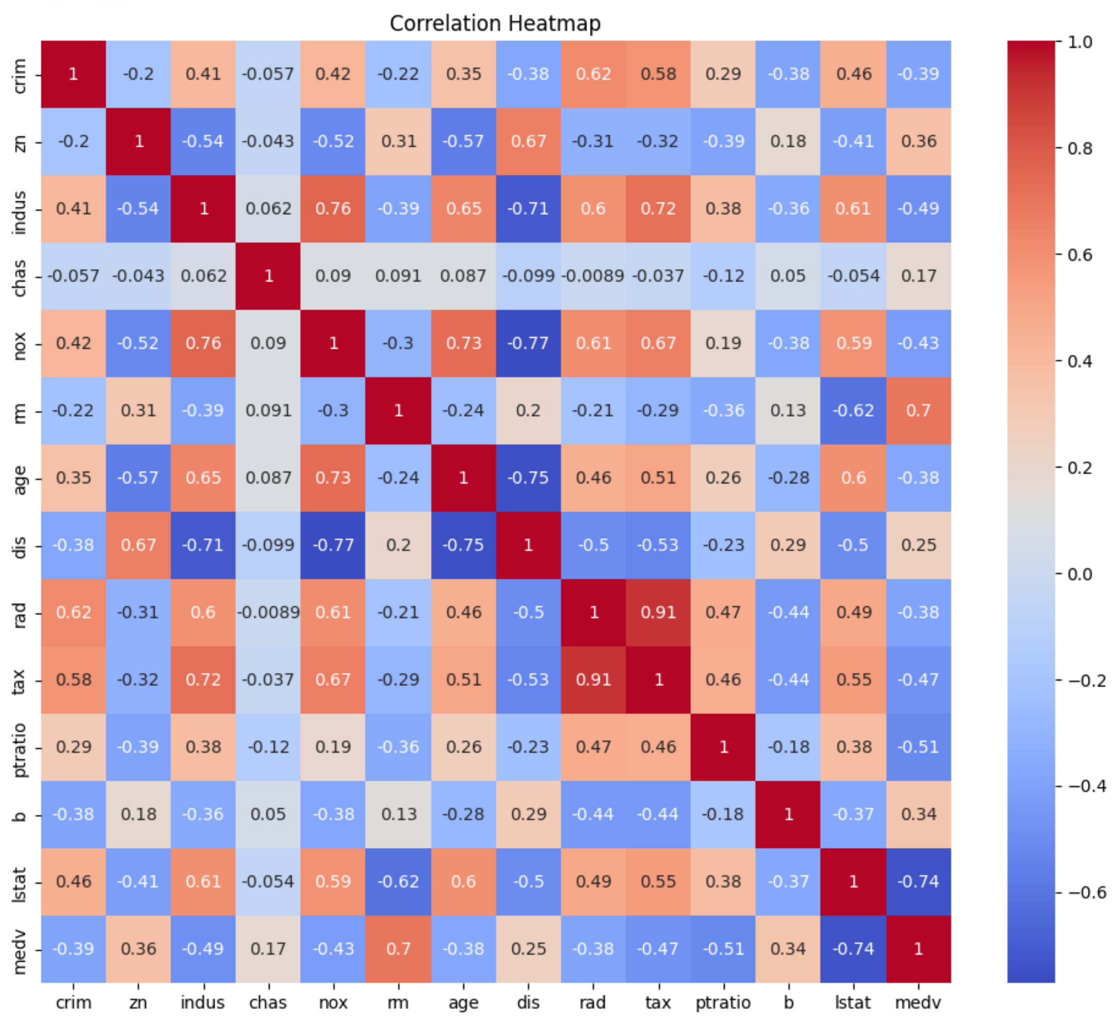
```
mse_all = mean_squared_error(y_test_all, y_pred_all)
rmse_all = np.sqrt(mse_all)
r2_all = r2_score(y_test_all, y_pred_all)

print(f"All Parameters - Mean Squared Error: {mse_all}")
print(f"All Parameters - Root Mean Squared Error: {rmse_all}")
print(f"All Parameters - R-squared: {r2_all}")
```

```
All Parameters - Mean Squared Error: 20.687720473048476
All Parameters - Root Mean Squared Error: 4.548375586189917
All Parameters - R-squared: 0.7200277678580317
```

```
# Plotting actual vs predicted values for all parameters
plt.figure(figsize=(10, 6))
plt.scatter(y_test_all, y_pred_all, edgecolor='k', alpha=0.7)
plt.plot([y_test_all.min(), y_test_all.max()], [y_test_all.min(), y_test_a
ll.max()], 'r--', lw=3)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('All Parameters - Actual vs Predicted Values')
plt.show()
```



```
# Draw a heatmap of correlations
plt.figure(figsize=(12, 10))
corr_matrix = data.corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



```
relevant_features = corr_matrix.index[abs(corr_matrix["medv"]) > 0.5].toli
st()
relevant_features.remove('medv')
print("Selected relevant features:", relevant_features)
```

```
Selected relevant features: ['rm', 'ptratio', 'lstat']
```

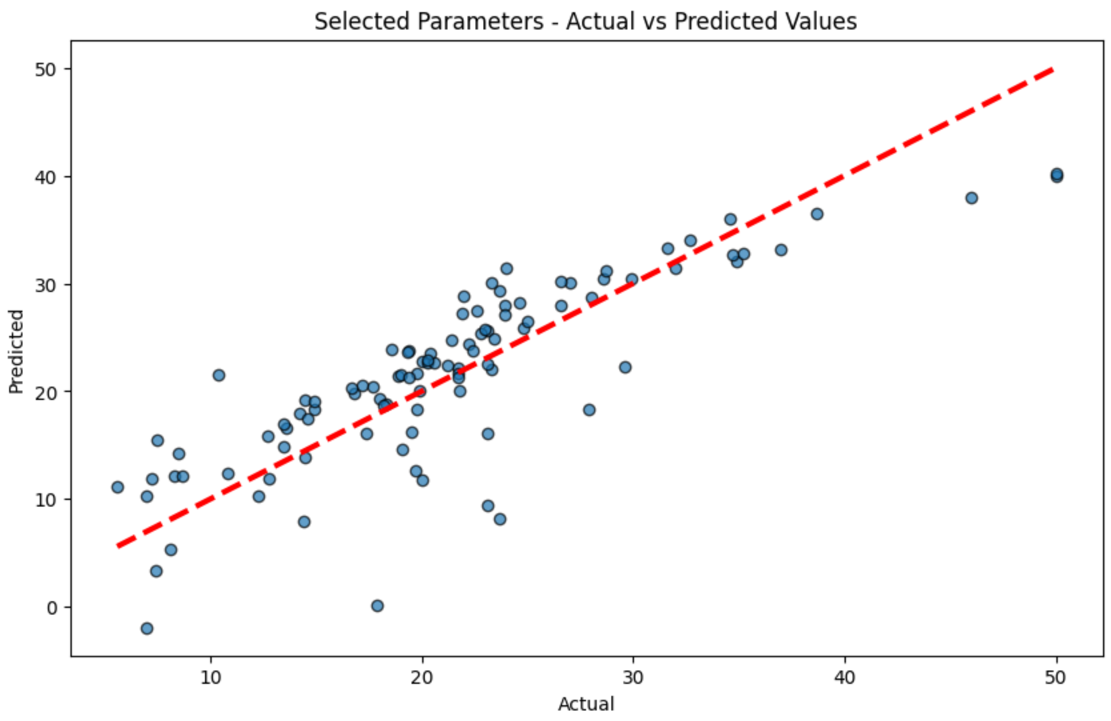
```
# Linear regression with selected parameters
X_relevant = data[relevant_features]
y_relevant = data['medv']

# Split the data into training and testing sets
X_train_rel, X_test_rel, y_train_rel, y_test_rel = train_test_split(X_rele
vant, y_relevant, test_size=0.2, random_state=42)

# Initialize the Linear Regression model
model_rel = LinearRegression()

# Train the model
model_rel.fit(X_train_rel, y_train_rel)
```

```
LinearRegression()
```

```
y_pred_rel = model_rel.predict(X_test_rel)

# Evaluate the model
mse_rel = mean_squared_error(y_test_rel, y_pred_rel)
rmse_rel = np.sqrt(mse_rel)
r2_rel = r2_score(y_test_rel, y_pred_rel)

print(f"Selected Parameters - Mean Squared Error: {mse_rel}")
print(f"Selected Parameters - Root Mean Squared Error: {rmse_rel}")
print(f"Selected Parameters - R-squared: {r2_rel}")
```

```
Selected Parameters - Mean Squared Error: 24.601921143326106
Selected Parameters - Root Mean Squared Error: 4.960032373213516
Selected Parameters - R-squared: 0.6670558853281565
```

```
# Plotting actual vs predicted values for selected parameters
plt.figure(figsize=(10, 6))
plt.scatter(y_test_rel, y_pred_rel, edgecolor='k', alpha=0.7)
plt.plot([y_test_rel.min(), y_test_rel.max()], [y_test_rel.min(), y_test_r
el.max()], 'r--', lw=3)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Selected Parameters - Actual vs Predicted Values')
plt.show()
```



```
# Compare results
print("\nComparison:")
print(f"All Parameters - RMSE: {rmse_all}, R-squared: {r2_all}")
print(f"Selected Parameters - RMSE: {rmse_rel}, R-squared: {r2_rel}")
```

```
Comparison:
All Parameters - RMSE: 4.548375586189917, R-squared: 0.7200277678580317
Selected Parameters - RMSE: 4.960032373213516, R-squared: 0.66705588532815
65
```

Conclusion:

The Mean Squared Error (MSE) calculated for the model using all features is lower, suggesting that this model better captures the variability in the housing prices. This could be due to the inclusion of additional information, even if some features contribute noise. Conversely, the higher MSE for the model with selected features indicates that important information might have been lost by excluding certain features, leading to less accurate predictions. This highlights the complexity of feature selection, where excluding less correlated features doesn't always result in improved model performance.