REGRESSION

Data Preparation Tasks

Import necessary libraries:

Import pandas, numpy, and scikit-learn libraries.

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Load a dataset:

Load a dataset (e.g., Boston Housing dataset) into a pandas DataFrame.

In [4]:
```
df=pd.read_csv("HousingData.csv")
df
```

Out[4]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	NaN
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	NaN	2.5050	1	273	21.0	396.90	7.88

506 rows × 14 columns

Inspect the dataset:

Display the first few rows and summary statistics of the dataset.

In [5]:
```
df.info()
```

```
<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
```

In [6]:
```
df.describe()
```

Out[6]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	486.000000	486.000000	486.000000	486.000000	506.000000	506.000000	486.000000	506.00
mean	3.611874	11.211934	11.083992	0.069959	0.554695	6.284634	68.518519	3.79
std	8.720192	23.388876	6.835896	0.255340	0.115878	0.702617	27.999513	2.10
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.12
25%	0.081900	0.000000	5.190000	0.000000	0.449000	5.885500	45.175000	2.10
50%	0.253715	0.000000	9.690000	0.000000	0.538000	6.208500	76.800000	3.20
75%	3.560263	12.500000	18.100000	0.000000	0.624000	6.623500	93.975000	5.18
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.12

In [7]:
```
df.head()
```

Out[7]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN

In [8]:
```
df.columns #n. of  columns
```

Out[8]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'MEDV'],
      dtype='object')
```

In [60]:
```
df.count() # number of rows
```

Out[60]:
```
CRIM       506
ZN         506
INDUS      506
CHAS       506
NOX        506
RM         506
AGE        506
DIS        506
RAD        506
TAX        506
PTRATIO    506
B          506
LSTAT      506
MEDV       506
dtype: int64
```

In [9]:
```
df.shape
```

Out[9]:
```
(506, 14)
```

Check for missing values:

Check and handle any missing values in the dataset.

In [40]:
```
df.isnull().sum()
```

Out[40]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

In [41]:
```
print("Infinite values in each feature:\n", np.isinf(x).sum())
```

```
Infinite values in each feature:
 CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
dtype: int64
```

In [42]:
```
df.fillna(df.mean(), inplace=True)  #handiling missing values
```

In [57]:
```
print(f'Any NaN values? {df.isna().values.any()}')
```

```
Any NaN values? False
```

In [58]:
```
print(f'Any duplicates? {df.duplicated().values.any()}')
```

```
Any duplicates? False
```

Feature selection:

Select relevant features for the regression model.

In [43]:
```
from sklearn.feature_selection import RFE, RFECV
```

In [44]:
```
from sklearn.linear_model import LinearRegression
```

In [49]:
```
from sklearn.preprocessing import StandardScaler
```

In [63]:
```
# Define the features and target variable
X = df.drop('MEDV', axis=1)  # Features
y = df['MEDV']  # Target
```

In [72]:
```
# Normalize the data
scaler = StandardScaler()
scaled_x = scaler.fit_transform(X)
```

In [73]:
```
model = LinearRegression()
rfe = RFE(model, n_features_to_select=5)
rfe = rfe.fit(X, y)
```

In [74]:
```
# Get the ranking of the features
print('Ranking of features:', rfe.ranking_)
```

```
Ranking of features: [5 6 3 1 1 1 8 1 4 7 1 9 2]
```

Split the data:

Split the data into training and testing sets.

In [68]:
```
from sklearn.model_selection import train_test_split
```

In [71]:
```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

# Verify the shape of the training and testing sets
print(f'x_train shape: {x_train.shape}')
print(f'x_test shape: {x_test.shape}')
print(f'y_train shape: {y_train.shape}')
print(f'y_test shape: {y_test.shape}')
```

```
x_train shape: (354, 13)
x_test shape: (152, 13)
y_train shape: (354,)
y_test shape: (152,)
```

Model Training and Evaluation Tasks

Train a Linear Regression model:

Train a simple linear regression model on the training data.

In [76]:
```
lr = LinearRegression()
lr.fit(x_train, y_train)
```

Out[76]:
```
LinearRegression()
```

Make predictions:

Make predictions on the testing data.

In [77]:
```
y_pred = lr.predict(x_test)
```

In [78]:
```
y_pred
```

Out[78]:
```
array([28.94684244, 37.080947  , 15.16334344, 25.59977001, 18.52050399,
       22.92953198, 17.99840423, 14.43800382, 22.06639946, 20.81944231,
       25.11881906, 18.72123522, -6.3071011 , 21.86734043, 19.02489041,
       25.40599155, 19.37239698,  5.95086419, 40.85794132, 17.18432226,
       24.93251127, 30.36869589, 11.39935905, 22.76748541, 17.50857422,
       15.11943012, 21.39728476, 14.47830797, 23.13536511, 19.56542429,
       22.18119889, 25.26140438, 25.38274556, 17.33845553, 16.24266113,
       17.25502918, 30.91188566, 20.39390975, 24.67921133, 22.85756708,
       14.52166278, 31.79401526, 42.81650058, 17.99338659, 27.3605525 ,
       16.56320925, 13.95314318, 26.53462251, 19.75194991, 30.2586963 ,
       21.02578   , 33.48015966, 15.97922002, 26.27389748, 39.58817583,
       22.50675409, 18.73574376, 33.02789505, 25.25340181, 13.16505164,
       22.85496664, 31.01835795, 31.53871931, 16.756762  , 21.22003054,
       17.10830699, 19.99132099, 26.38841982, 31.29685552, 11.7231931 ,
       20.97258967, 26.63335287, 10.95888213, 13.4158528 , 23.97236272,
        5.66817291, 21.45418183, 41.56712305, 18.17594507,  9.48043905,
       21.41371834, 12.83886454, 21.86984499,  9.17605135, 22.77784869,
       32.52178098, 19.08870649, 25.76101077, 29.03718152, 19.79942549,
       25.99768914,  5.55973693, 19.67592421, 15.58843642, 12.05864065,
       20.27908381, 23.71715385, -0.33118013, 13.574537  , 15.58002707,
       22.63744043, 24.63223041, 10.44203817, 19.7020956 , 23.19945908,
       12.54800472, 18.49880298, 25.97268923, 21.20986934, 24.74534596,
        7.87558719, 20.25419972, 22.14202929, 27.49138377, 32.02443374,
       14.76313946, 34.77670429, 13.04840788, 21.34756531, 28.27990076,
       15.36800974, 25.1865523 ,  3.35128356, 23.72015504, 26.26550372,
       23.10384338, 25.43047681, 33.18312767, 21.64835442, 38.47259389,
       14.41283553, 26.16157834, 17.60366271, 20.79190337, 10.45939827,
       20.49798709, 22.67008757, 31.99471793, 31.79670093, 15.52452541,
       17.02050129, 29.09484276, 25.06877192, 17.53901345,  6.54114984,
       25.94712745, 23.79112755, 20.48659363, 13.59421774, 40.22597747,
       16.39307918, 17.86909777])
```

Evaluate the model:

Calculate the Mean Absolute Error (MAE) of the model.

In [79]:
```python
from sklearn.metrics import mean_absolute_error, r2_score
```

In [85]:
```python
mae = mean_absolute_error(y_test, y_pred)
```

In [86]:
```python
mae
```

Out[86]:
```
3.1410525671083374
```

Calculate R-squared:

Calculate the R-squared value to assess model performance.

In [83]:
```python
from sklearn.metrics import mean_absolute_error, r2_score
```

In [84]:
```python
r2 = r2_score(y_test, y_pred)
r2
```

Out[84]:
```
0.7071862632031873
```

Plot actual vs predicted values:

Create a scatter plot of actual vs predicted values.

'k--': This is a string that specifies the style of the line:

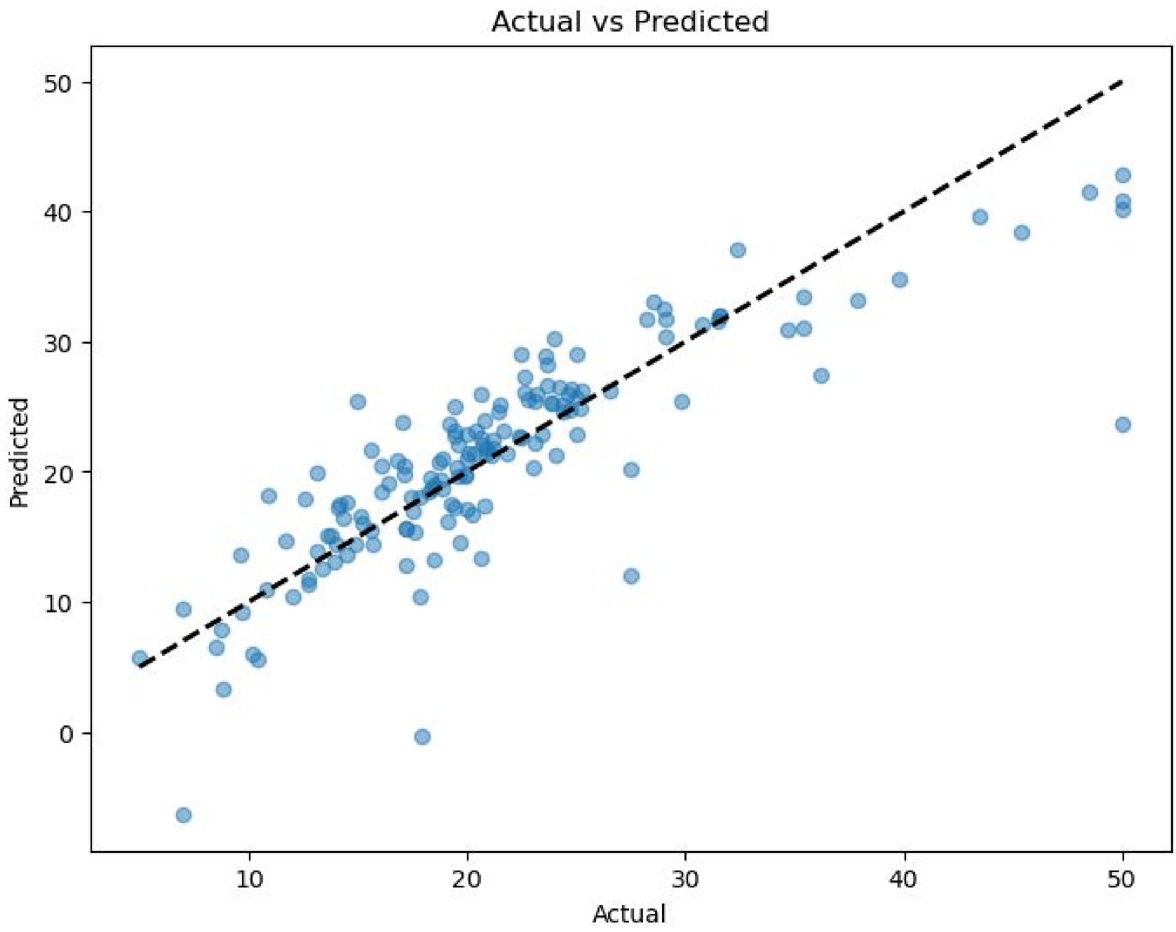
'k' indicates the color of the line (black in this case). '--' indicates that the line should be dashed.

[y_test.min(), y_test.max()]: This specifies the x-coordinates of the points that define the line.

y_test.min() gets the minimum value from the actual target values (the ground truth). y_test.max() gets the maximum value from the actual target values. This means the line will start from the minimum actual value and extend to the maximum actual value

In [87]:

```
plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--', lw=2)
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Actual vs Predicted')
plt.show()                            # Lw=2 for thickness
```



Advanced Regression Techniques Tasks

Polynomial Regression:

Train a polynomial regression model.

Evaluate Polynomial Regression:

Calculate the MAE and R-squared for the polynomial regression model.

In [89]:
```
from sklearn.preprocessing import PolynomialFeatures
```

PolynomialFeatures(degree=2) creates new features by taking the original features and raising them to powers up to the specified degree (2 in this case)

In [94]:
```
poly= PolynomialFeatures(degree= 2)
x_train_poly= poly.fit_transform(x_train)
x_test_poly= poly.transform(x_test)
```

In [130…
```python
poly_model = LinearRegression()
poly_model.fit(x_train_poly, y_train)
```

Out[130]:
```
LinearRegression()
```

In [131…
```python
y_pred_ploy = poly_model.predict(x_test_poly)
```

Evaluate Polynomial Regression

In [135…
```python
poly_mae = mean_absolute_error(y_test, y_pred_ploy)
poly_r2 = r2_score(y_test, y_pred_ploy)
print(poly_r2)
```

```
0.5374014284812872
```

In [136…
```python
print(poly_mae)
```

```
3.2312574297458343
```

Ridge Regression:

Train a ridge regression model.

Evaluate Ridge Regression:

Calculate the MAE and R-squared for the ridge regression model.

In [99]:
```python
from sklearn.linear_model import Ridge
```

Ridge Regression (Ridge(alpha=1.0)): Adds L2 regularization, which penalizes large coefficients but does not shrink them to zero, making it useful when all features are expected to contribute to the prediction

In [101…
```python
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(x_train, y_train)
```

Out[101]:
```
Ridge()
```

In [102…
```python
ridge_y_pred = ridge_model.predict(x_test)
```

In [103…
```python
ridge_mae = mean_absolute_error(y_test, ridge_y_pred)
ridge_r2 = r2_score(y_test, ridge_y_pred)
```

In [104…
```python
print(ridge_r2)
```

```
0.7005989750756149
```

In [112...
```
print(ridge_mae)
```

```
3.158193755884402
```

Lasso Regression:

Train a lasso regression model.

Evaluate lasso Regression:

Calculate the MAE and R-squared for the lasso regression model.

In [105...
```
from sklearn.linear_model import Lasso
```

Lasso Regression (Lasso(alpha=0.1)): Adds L1 regularization to the model, which can shrink some coefficients to zero, making it useful for feature selection.

In [106...
```
lasso_model = Lasso(alpha=0.1)
lasso_model.fit(x_train, y_train)
```

Out[106]:
```
Lasso(alpha=0.1)
```

In [107...
```
lasso_y_pred = lasso_model.predict(x_test)
```

In [109...
```
lasso_mae = mean_absolute_error(y_test, lasso_y_pred)
lasso_r2 = r2_score(y_test, lasso_y_pred)
```

In [110...
```
print(lasso_mae)
```

```
3.224332395349057
```

In [111...
```
print(lasso_r2)
```

```
0.6914578902133115
```

ElasticNet Regression:

Train an ElasticNet regression model.

Evaluate ElasticNet Regression:

Calculate the MAE and R-squared for the ElasticNet regression model.

In [115...
```
from sklearn.linear_model import ElasticNet
```

In [116…

```
elastic_model = ElasticNet(alpha=0.5, l1_ratio=0.5)
elastic_model.fit(x_train, y_train)
elastic_y_pred = elastic_model.predict(x_test)
elastic_mae = mean_absolute_error(y_test, elastic_y_pred)
elastic_r2 = r2_score(y_test, elastic_y_pred)
print(f'ElasticNet Regression: MAE = {elastic_mae:.2f}, R-squared = {elastic_r2:.2f
```

```
ElasticNet Regression: MAE = 3.44, R-squared = 0.68
```

Compare all models:

Compare the performance of all regression models and visualize the results.

In [139…

```
results = {
    'Model': ['Polynomial Regression', 'Ridge Regression', 'Lasso Regression', 'Ela
    'MAE': [poly_mae, ridge_mae, lasso_mae, elastic_mae],
    'R-squared': [poly_r2, ridge_r2, lasso_r2, elastic_r2]
}

results_df = pd.DataFrame(results)
```

In [140…

```
print(results_df)
```

```
                   Model       MAE  R-squared
0  Polynomial Regression  3.231257   0.537401
1       Ridge Regression  3.158194   0.700599
2       Lasso Regression  3.224332   0.691458
3  ElasticNet Regression  3.440487   0.683913
```

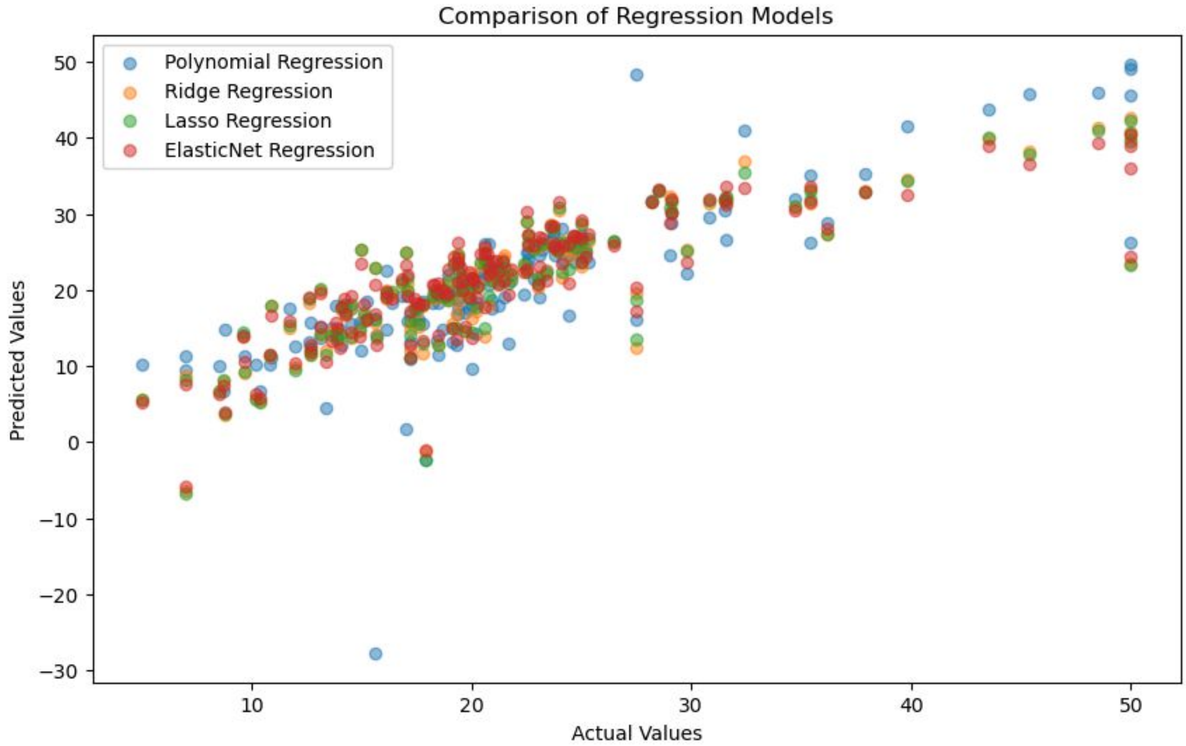
In [141…

```
# Visualize the results
fig, ax = plt.subplots(figsize=(10, 6))

ax.scatter(y_test, y_pred_ploy, label='Polynomial Regression', alpha=0.5)
ax.scatter(y_test, ridge_y_pred, label='Ridge Regression', alpha=0.5)
ax.scatter(y_test, lasso_y_pred, label='Lasso Regression', alpha=0.5)
ax.scatter(y_test, elastic_y_pred, label='ElasticNet Regression', alpha=0.5)

# Add labels and title
ax.set_xlabel('Actual Values')
ax.set_ylabel('Predicted Values')
ax.set_title('Comparison of Regression Models')
ax.legend()

# Display the plot
plt.show()
```



Advanced Regression Techniques

Train a Decision Tree Regressor:

Train a decision tree regressor on the training data.

In [142…

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

In [143…

```
dt_regressor = DecisionTreeRegressor()
dt_regressor.fit(x_train, y_train)
```

Out[143]:

```
DecisionTreeRegressor()
```

Evaluate the performance using MSE and R-squared.

In [144…

```
y_pred_dt = dt_regressor.predict(x_test)
```

In [145…

```
dt_mse = mean_squared_error(y_test, y_pred_dt)
dt_r2 = r2_score(y_test, y_pred_dt)
```

In [146…

```
print(dt_mse)
print(dt_r2)
```

```
24.429868421052632
0.6721399341300787
```

Train a Random Forest Regressor:

Train a random forest regressor on the training data.

In [147…
```
from sklearn.ensemble import RandomForestRegressor
```

In [148…
```
rf_regressor = RandomForestRegressor()
rf_regressor.fit(x_train, y_train)
```

Out[148]:
```
RandomForestRegressor()
```

In [149…
```
y_pred_rf = rf_regressor.predict(x_test)
```

Evaluate the performance using MSE and R-squared.

In [150…
```
rf_mse = mean_squared_error(y_test, y_pred_rf)
rf_r2 = r2_score(y_test, y_pred_rf)
```

In [151…
```
print(rf_mse)
print(rf_r2)
```

```
9.684998394736843
0.8700228688537617
```

Train a Gradient Boosting Regressor:

Train a gradient boosting regressor on the training data.

In [155…
```
from sklearn.ensemble import GradientBoostingRegressor
```

In [157…
```
gb_regressor = GradientBoostingRegressor()
gb_regressor.fit(x_train, y_train)
```

Out[157]:
```
GradientBoostingRegressor()
```

In [158…
```
y_pred_gb = gb_regressor.predict(x_test)
```

Evaluate the performance using MSE and R-squared

In [159…
```
gb_mse = mean_squared_error(y_test, y_pred_gb)
gb_r2 = r2_score(y_test, y_pred_gb)
```

In [160…
```
print(gb_mse)
print(gb_r2)
```

```
8.42175152481148
0.8869762226273248
```

In [163…

```
results = {
    'Model': ['dt_regressor', 'rf_regresso', 'gb_regressor'],
    'MAE': [dt_mse, rf_mse, gb_mse],
    'R-squared': [dt_r2, rf_r2, gb_r2]
}

results_df7 = pd.DataFrame(results)
```

In [164…

```
print(results_df7)
```

```
          Model        MAE  R-squared
0  dt_regressor  24.429868   0.672140
1   rf_regresso   9.684998   0.870023
2  gb_regressor   8.421752   0.886976
```

Model Tuning and Optimization:::

Hyperparameter Tuning for Ridge Regression:

Perform hyperparameter tuning for the alpha parameter of the ridge regression model using cross-validation.

In [165…

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import GridSearchCV


param_grid = {'alpha': [0.1, 1, 10, 100]}
ridge_regressor = Ridge()
grid_search = GridSearchCV(ridge_regressor, param_grid, cv=5)
grid_search.fit(x_train, y_train)

print(f'Best Alpha: {grid_search.best_params_["alpha"]}')
print(f'Best Score: {grid_search.best_score_}')
```

```
Best Alpha: 0.1
Best Score: 0.6766815039525739
```

Hyperparameter Tuning for Lasso Regression:

Perform hyperparameter tuning for the alpha parameter of the lasso regression

In [166…

```
from sklearn.linear_model import Lasso
from sklearn.model_selection import GridSearchCV
```

In [170…

```
param_grid= {'alpha': [0.1, 1, 10,100] }
lasso_regressor= Lasso()
grid_search= GridSearchCV(lasso_regressor,param_grid, cv=5)
grid_search.fit(x_train, y_train)
```

```
print(f'BEST ALPHA: {grid_search.best_params_["alpha"]}')
print(f'BEST SCORE: {grid_search.best_score_}')
```

```
BEST ALPHA: 0.1
BEST SCORE: 0.6684514748259278
```

Hyperparameter Tuning for Random Forest:

Perform hyperparameter tuning for the number of estimators and max depth of the random forest regressor using cross-validation

In [171…

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Assume X_train, y_train are your training data
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [5, 10, 15]
}
rf_regressor = RandomForestRegressor()
grid_search = GridSearchCV(rf_regressor, param_grid, cv=5)
grid_search.fit(x_train, y_train)

print(f'Best n_estimators: {grid_search.best_params_["n_estimators"]}')
print(f'Best max_depth: {grid_search.best_params_["max_depth"]}')
print(f'Best Score: {grid_search.best_score_}')
```

```
Best n_estimators: 100
Best max_depth: 15
Best Score: 0.7941907158853029
```

Hyperparameter Tuning for Gradient Boosting:

Perform hyperparameter tuning for the learning rate and number of estimators of the gradient boosting regressor using cross-validation.

In [175…

```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV

param_grid = {
    'learning_rate': [0.1, 0.01, 0.001],
    'n_estimators': [100, 200, 300]
}
gb_regressor = GradientBoostingRegressor()
```

```
grid_search = GridSearchCV(gb_regressor, param_grid, cv=5)
grid_search.fit(x_train, y_train)
```

Out[175]:

```
GridSearchCV(cv=5, estimator=GradientBoostingRegressor(),
             param_grid={'learning_rate': [0.1, 0.01, 0.001],
                         'n_estimators': [100, 200, 300]})
```

In [176…

```
print(f'Best learning_rate: {grid_search.best_params_["learning_rate"]}')
print(f'Best n_estimators: {grid_search.best_params_["n_estimators"]}')
print(f'Best Score: {grid_search.best_score_}')
```

```
Best learning_rate: 0.1
Best n_estimators: 100
Best Score: 0.8265513459241814
```

In [ ]:

```

```