ASSIGNMENT 3 - REGRESSION

** Objective:** This assignment aims to assess your understanding of regression techniques in supervised learning by applying them to a real-world dataset.

Dataset: The California Housing dataset from the sklearn library is used. It includes information about house features in California and their median prices.

In [1]:
```
import warnings
import sys
if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1. Loading and Preprocessing:

Load the California Housing dataset using the fetch_california_housing function from sklearn. Convert the dataset into a pandas DataFrame for easier handling. Check for missing values and handle them if any are found. Perform feature scaling (e.g., standardization) to ensure the data is properly prepared for regression models. Explain the preprocessing steps and why they are important for this dataset.

In [3]:
```
from sklearn.datasets import fetch_california_housing

# Load the California Housing dataset
data = fetch_california_housing()

# Convert the dataset into a pandas DataFrame
X = pd.DataFrame(data.data, columns=data.feature_names)
Y = pd.Series(data.target)

# Display the first few rows of the dataset
print("Initial Dataset:")
print(X.head())
print("\nInitial Target Dataset:")
print(Y.head())
```

```
Initial Dataset:
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88   
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86   
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85   
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85   
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85   

   Longitude  
0    -122.23  
1    -122.22  
2    -122.24  
3    -122.25  
4    -122.25  

Initial Target Dataset:
0    4.526
1    3.585
2    3.521
3    3.413
4    3.422
dtype: float64
```

In [4]:
```python
print("Feature properties of the dataset is:")
print("\t")
X.info()
print("\nFeature property of Target Dataset:")
Y.info()
```

```
Feature properties of the dataset is:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   MedInc      20640 non-null  float64
 1   HouseAge    20640 non-null  float64
 2   AveRooms    20640 non-null  float64
 3   AveBedrms   20640 non-null  float64
 4   Population  20640 non-null  float64
 5   AveOccup    20640 non-null  float64
 6   Latitude    20640 non-null  float64
 7   Longitude   20640 non-null  float64
dtypes: float64(8)
memory usage: 1.3 MB

Feature property of Target Dataset:
<class 'pandas.core.series.Series'>
RangeIndex: 20640 entries, 0 to 20639
Series name: None
Non-Null Count  Dtype  
--------------  -----  
20640 non-null  float64
dtypes: float64(1)
memory usage: 161.4 KB
```

In [5]:
```python
print("Statistical Analysis of the dataset")
print("\t")
X.describe()
```

Statistical Analysis of the dataset

Out[5]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOc
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333

In [6]:
```
print("Statistical Analysis of Target Dataset:")
Y.describe()
```

```
Statistical Analysis of Target Dataset:
```

Out[6]:
```
count    20640.000000
mean         2.068558
std          1.153956
min          0.149990
25%          1.196000
50%          1.797000
75%          2.647250
max          5.000010
dtype: float64
```

In [7]:
```
# Step 1: Check for missing values
missing_values = X.isnull().sum()
print("\nMissing Values:")
print(missing_values)
```

```
Missing Values:
MedInc        0
HouseAge      0
AveRooms      0
AveBedrms     0
Population    0
AveOccup      0
Latitude      0
Longitude     0
dtype: int64
```

In [8]:
```
missing_values = Y.isnull().sum()
print("\nMissing Values:")
print(missing_values)
```

```
Missing Values:
0
```

In [9]:
```
# Find duplicates in Initial dataset
X.duplicated().sum()
```

Out[9]: np.int64(0)

In [10]:
```
# Step 2: Feature Scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_X = scaler.fit_transform(X)

# Display the scaled dataset
print("\nScaled Dataset:")
print(scaled_X[:10])
```

```
Scaled Dataset:
[[ 2.34476576  0.98214266  0.62855945 -0.15375759 -0.9744286  -0.04959654
   1.05254828 -1.32783522]
 [ 2.33223796 -0.60701891  0.32704136 -0.26333577  0.86143887 -0.09251223
   1.04318455 -1.32284391]
 [ 1.7826994   1.85618152  1.15562047 -0.04901636 -0.82077735 -0.02584253
   1.03850269 -1.33282653]
 [ 0.93296751  1.85618152  0.15696608 -0.04983292 -0.76602806 -0.0503293
   1.03850269 -1.33781784]
 [-0.012881    1.85618152  0.3447108  -0.03290586 -0.75984669 -0.08561576
   1.03850269 -1.33781784]
 [ 0.08744664  1.85618152 -0.26972966  0.01466934 -0.89407076 -0.08961842
   1.03850269 -1.33781784]
 [-0.11136631  1.85618152 -0.2009177  -0.3066332  -0.29271158 -0.0907249
   1.03382082 -1.33781784]
 [-0.39513665  1.85618152 -0.25523193 -0.07354166 -0.23707923 -0.12347647
   1.03382082 -1.33781784]
 [-0.94235915  1.06160074 -0.45870257  0.04425393 -0.19380963 -0.1004992
   1.03382082 -1.34280914]
 [-0.09446958  1.85618152 -0.18528316 -0.22468709  0.1108437  -0.08650142
   1.03382082 -1.33781784]]
```

In [11]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(scaled_X, Y, test_size=0.2,

print("\nTraining set shape of X: ", X_train.shape)
print("Test set shape of X:", X_test.shape)
print("\nTraining set shape of Y", Y_train.shape)
print("Test set shape of Y:", Y_test.shape)
```

```
Training set shape of X:  (16512, 8)
Test set shape of X: (4128, 8)

Training set shape of Y (16512,)
Test set shape of Y: (4128,)
```

1. Loading and Preprocessing:

Load the California Housing dataset using the fetch_california_housing function from sklearn. Convert the dataset into a pandas DataFrame for easier handling. Check for missing values and handle them if any are found. Perform feature scaling (e.g., standardization) to ensure the data is properly prepared for regression models. Explain the preprocessing steps and why they are important for this dataset.

2. Regression Algorithm Implementation

Steps:

1. **Algorithms to Implement:**

    - Linear Regression
    - Decision Tree Regressor
    - Random Forest Regressor
    - Gradient Boosting Regressor
    - Support Vector Regressor (SVR)
2. **For Each Algorithm:**

    - Provide a brief explanation of how it works.
    - Explain why it might be a good fit for this dataset.

---

3. Model Evaluation and Comparison

Steps:

1. **Evaluate Performance Using Metrics:**

    - Mean Squared Error (MSE)
    - Mean Absolute Error (MAE)
    - R-squared Score (R^2)
2. **Compare Results:**

    - Identify the best-performing algorithm and justify why it performed well.
    - Highlight the worst-performing algorithm and explain its shortcomings.

In [12]:
```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Regression Algorithms Implementation
models = {
    "Linear Regression": LinearRegression(),
    "Decision Tree Regressor": DecisionTreeRegressor(random_state=42),
    "Random Forest Regressor": RandomForestRegressor(random_state=42),
    "Gradient Boosting Regressor": GradientBoostingRegressor(random_state=42),
    "Support Vector Regressor": SVR()
}

# Train and evaluate each model
for name, model in models.items():
    print(f"\n{name}")
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)
    Y_pred

    # Calculate performance metrics
    mse = mean_squared_error(Y_test, Y_pred)
```

```python
    mae = mean_absolute_error(Y_test, Y_pred)
    r2 = r2_score(Y_test, Y_pred)

    # Display results
    print(f"Mean Squared Error: {mse}")
    print(f"Mean Absolute Error: {mae}")
    print(f"R² Score: {r2}")
```

```
Linear Regression
Mean Squared Error: 0.5558915986952444
Mean Absolute Error: 0.5332001304956565
R² Score: 0.5757877060324508

Decision Tree Regressor
Mean Squared Error: 0.4942716777366763
Mean Absolute Error: 0.4537843265503876
R² Score: 0.6228111330554302

Random Forest Regressor
Mean Squared Error: 0.25549776668540763
Mean Absolute Error: 0.32761306601259704
R² Score: 0.805024407701793

Gradient Boosting Regressor
Mean Squared Error: 0.29399901242474274
Mean Absolute Error: 0.37165044848436773
R² Score: 0.7756433164710084

Support Vector Regressor
Mean Squared Error: 0.3551984619989409
Mean Absolute Error: 0.3977630963437857
R² Score: 0.728940759795647
```

Explanations:

1. Linear Regression

How it works: Linear Regression models the relationship between the features (independent variables) and the target (dependent variable) by fitting a linear equation (a straight line or hyperplane). It minimizes the sum of squared residuals to find the best fit.

Suitability: It works well when there is a linear relationship between features and the target. It is suitable for the California Housing dataset if housing prices (target) are influenced by independent variables in a linear manner. However, it may not capture complex interactions between features.

2. Decision Tree Regressor

How it works: A Decision Tree splits the data into subsets using feature thresholds. Each split reduces the error in predicting the target. The process continues until a stopping criterion (e.g., minimum samples per leaf) is met.

Suitability: It captures non-linear relationships and feature interactions. This makes it effective for the California Housing dataset, where housing prices are influenced by non-linear factors like location, population density, and median income.

3. Random Forest Regressor

How it works: Random Forest combines multiple Decision Trees, each trained on random subsets of data and features. It averages the predictions from all the trees to produce a more stable and accurate result, reducing overfitting.

Suitability: It handles non-linearity and high-dimensional data well. This makes it suitable for the California Housing dataset, as it can manage large feature spaces and capture complex relationships between features.

4. Gradient Boosting Regressor

How it works: Gradient Boosting builds models sequentially, where each model corrects the errors made by the previous one. It optimizes a loss function (e.g., Mean Squared Error) using gradient descent.

Suitability: It excels at capturing complex relationships with fine-tuned models. It is suitable for the California Housing dataset when high accuracy is the goal, as it minimizes errors from previous models effectively.

5. Support Vector Regressor (SVR)

How it works: SVR finds a hyperplane (or curve) that fits the data points within a specified margin of tolerance (epsilon). Kernel functions can be applied to map the data to higher dimensions, enabling the model to capture non-linear relationships.

Suitability: It works well with small- to medium-sized datasets and handles non-linear relationships effectively. It is suitable for the California Housing dataset to capture specific non-linear patterns, though it may struggle with larger datasets due to computational complexity.

Model Evaluation

Best Performing Model: Random Forest Regressor Mean Squared Error (MSE): 0.2555 (lowest among all models) Mean Absolute Error (MAE): 0.3276 (lowest among all models) R^2 Score: 0.8050 (highest among all models) Explanation: The Random Forest Regressor achieved the highest R^2 score, indicating the best fit to the data and explaining the highest proportion of variance in the target variable. It also has the lowest MSE and MAE, showing that its predictions are closest to the actual values on average. This combination of high accuracy and low error makes it the best-performing model.

Worst Performing Model: Linear Regression Mean Squared Error (MSE): 0.5559 (highest among all models) Mean Absolute Error (MAE): 0.5332 (highest among all models) R^2 Score: 0.5758 (lowest among all models) Explanation: The Linear Regression model has the lowest R^2 score, meaning it explains the least variance in the target variable. It also has the highest MSE and MAE, indicating that its predictions are the farthest from the actual values on average compared to other models. This poor performance in both accuracy and error metrics makes it the worst-performing model for this dataset.