ASSIGNMENT 3 - REGRESSION

Objective:

The objective of this assignment is to evaluate your understanding of regression techniques in supervised learning by applying them to a real-world dataset.

Dataset:

Use the California Housing dataset available in the sklearn library. This dataset contains information about various features of houses in California and their respective median prices.

In [8]:
```
import warnings
import sys
if not sys.warnoptions:
    warnings.simplefilter("ignore")
```

In [10]:
```
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.

○ Handle missing values (if any) and perform necessary feature scaling (e.g., standardization).

○ Explain the preprocessing steps you performed and justify why they are necessary for this dataset.

In [12]:
```
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 [14]:
```
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 [16]:
```
print("Statistical Analysis of the dataset")
print("\t")
X.describe()
```

```
Statistical Analysis of the dataset
```

Out[16]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070655	35.631861	-119.569704
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050	2.135952	2.003532
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308	32.540000	-124.350000
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741	33.930000	-121.800000
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116	34.260000	-118.490000
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261	37.710000	-118.010000
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333	41.950000	-114.310000

In [18]:
```
print("Statistical Analysis of Target Dataset:")
Y.describe()
```

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

Out[18]:
```
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 [20]:
```
# 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 [22]:
```
missing_values = Y.isnull().sum()
print("\nMissing Values:")
print(missing_values)
```

```
Missing Values:
0
```

In [24]:
```
# Find duplicates in Initial dataset
X.duplicated().sum()
```

Out[24]: 0

In [26]:
```
# 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 [28]:
```python
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, random_state=42)

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,)
```

Explanation:

○ Loaded the California Housing dataset using the fetch_california_housing function from sklearn. Then the dataset was converted into a pandas DataFrame for further analysis.

○ 'X.info()' and 'Y.info()' were executed to determine the features and structure of the dataset. 'X.describe()' and 'Y.describe()' were executed to acquire statistical analysis of the dataset. Both functions will give us a brief idea of the dataset we are working with.

○ Missing values can impact the model's ability to learn from the data. So 'isnull()' function was used to detect any missing values. Similarly 'X.duplicated().sum()' function was executed to ascertain presence of duplicate values, which has to be removed to ensure data accuracy. No missing/duplicate values were found.

○ Using 'StandardScaler()' function, feature scaling was performed in order to ensure feature uniformity. This will assist in creating an efficient machine learning model.

○ Finally, the dataset was split into training and testing subsets to perform machine learning model evaluation.

All these preproccessing steps ensure we have a clean and accurate dataset, which is required to create a well-rated machine learning model.

2. Regression Algorithm Implementation:

Implement the following regression algorithms:

○ Linear Regression

○ Decision Tree Regressor

○ Random Forest Regressor

○ Gradient Boosting Regressor

○ Support Vector Regressor (SVR)

For each algorithm:

○ Provide a brief explanation of how it works.

○ Explain why it might be suitable for this dataset.

3. Model Evaluation and Comparison:

- **Evaluate the performance of each algorithm using the following metrics:**
  - Mean Squared Error (MSE)
  - Mean Absolute Error (MAE)
  - R-squared Score (R^2)

- **Compare the results of all models and identify:**
  - The best-performing algorithm with justification.
  - The worst-performing algorithm with reasoning.

In [30]:
```
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)
```

```
    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: 0.5558915986952441
Mean Absolute Error: 0.5332001304956565
R² Score: 0.575787706032451

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: 0.3551984619989429
Mean Absolute Error: 0.397763096343787
R² Score: 0.7289407597956454
```

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) to minimize the sum of squared residuals.

Suitability: Works well when there is a linear relationship between features and the target. Suitable for the California Housing dataset if housing prices (target) are influenced by independent variables in a linear manner. However, it might 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 is met (e.g., minimum samples per leaf).

Suitability: Captures non-linear relationships and feature interactions. Effective for the California Housing dataset as housing prices often depend on non-linear factors like location, population density, and median income.

3. Random Forest Regressor

How it works: Random Forest combines multiple decision trees (trained on random subsets of data and features) to produce a more stable and accurate prediction. It averages the predictions from all the trees to reduce overfitting.

Suitability: Handles non-linearity and high-dimensional data well. Suitable for the California Housing dataset because of its ability to 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: Excels in capturing complex relationships with fine-tuned models. Suitable for the California Housing dataset when the goal is high accuracy, as it effectively minimizes errors from previous models.

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 used to map the data to higher dimensions for non-linear relationships.

Suitability: Works well with small- to medium-sized datasets and handles non-linear relationships. Suitable for the California Housing dataset for capturing 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 the highest proportion of variance in the target variable explained by the model. Additionally, its MSE and MAE are the lowest, showing that the model's 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. Additionally, it has the highest MSE and MAE, indicating that its predictions are farthest from the actual values on average compared to the other models. This poor performance in both accuracy and error metrics makes it the worst-performing model for this dataset.