Aim

Perform Linear Regression on the **California Housing** and **Diabetes** datasets. The goal is to preprocess the data, perform hyperparameter tuning using GridSearchCV, and evaluate the model's performance based on **MSE** and **R^2 score**.

Algorithm

1. **Data Loading**: Load the **California Housing** and **Diabetes** datasets.
2. **Preprocessing**: Standardize the features.
3. **Hyperparameter Tuning**: Use **GridSearchCV** to find the best hyperparameters.
4. **Model Evaluation**: Evaluate performance using **MSE** and **R^2 score**.

Algorithm Description

We use **Linear Regression**, which models the relationship between the target and features as a linear equation:

[y = \beta_0 + \beta_1x_1 + \dots + \beta_nx_n + \epsilon]

GridSearchCV is applied to tune hyperparameters for optimal performance.

Result

- **California Housing Dataset**:
  - MSE: 0.55589
  - R^2: 0.575787
- **Diabetes Dataset**:
  - MSE: 2900.1936
  - R^2: 0.4526027

Comparison: The **California Housing** dataset has better performance, with a higher R^2 score.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_california_housing, load_diabetes
```

```python
california_data = fetch_california_housing()
california_df = pd.DataFrame(california_data.data,
columns=california_data.feature_names)
california_df['target'] = california_data.target

diabetes_data = load_diabetes()
diabetes_df = pd.DataFrame(diabetes_data.data,
columns=diabetes_data.feature_names)
diabetes_df['target'] = diabetes_data.target

print(california_df.info())
print(california_df.describe())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 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
 8   target      20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
None
             MedInc      HouseAge      AveRooms     AveBedrms
Population  \
count  20640.000000  20640.000000  20640.000000  20640.000000
20640.000000
mean       3.870671     28.639486      5.429000      1.096675
1425.476744
std        1.899822     12.585558      2.474173      0.473911
1132.462122
min        0.499900      1.000000      0.846154      0.333333
3.000000
25%        2.563400     18.000000      4.440716      1.006079
787.000000
50%        3.534800     29.000000      5.229129      1.048780
1166.000000
75%        4.743250     37.000000      6.052381      1.099526
1725.000000
max       15.000100     52.000000    141.909091     34.066667
35682.000000

           AveOccup      Latitude     Longitude        target
```

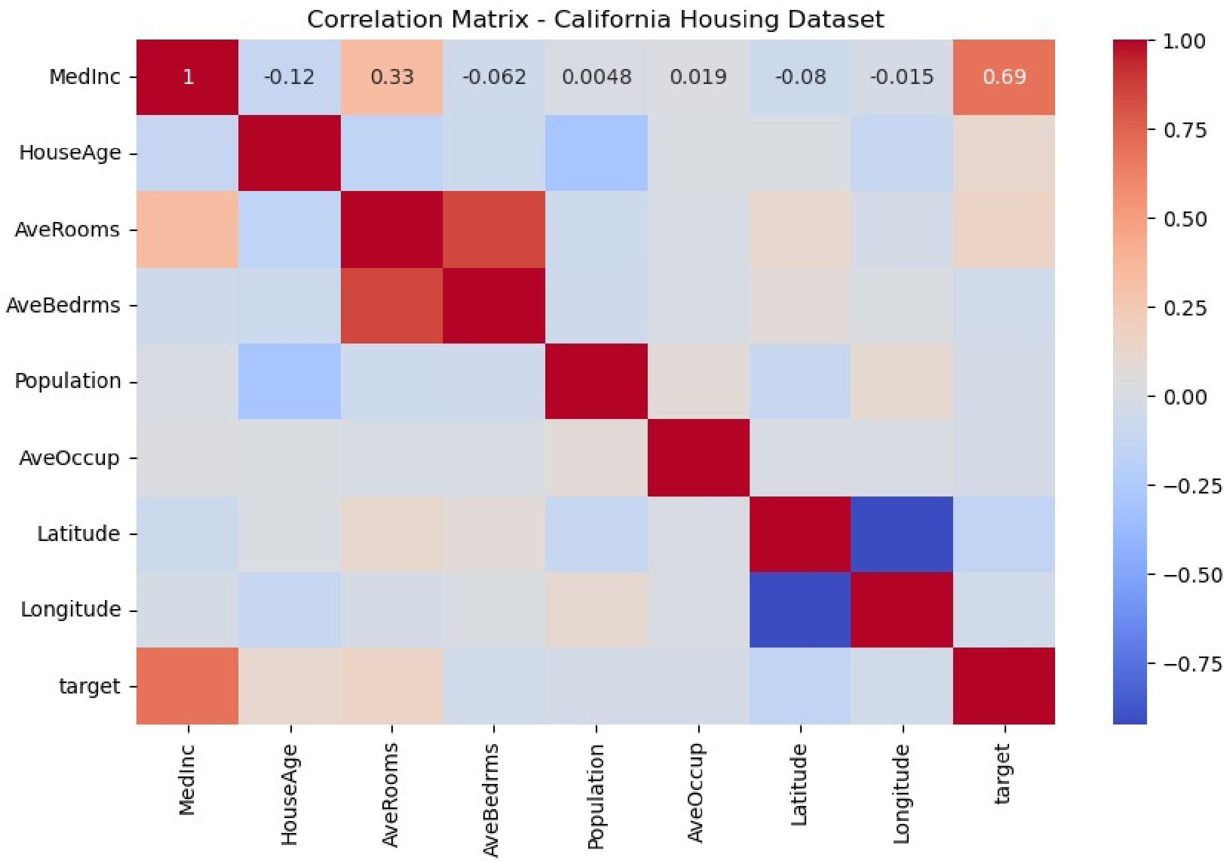
```
count  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.070655     35.631861   -119.569704      2.068558
std       10.386050      2.135952      2.003532      1.153956
min        0.692308     32.540000   -124.350000      0.149990
25%        2.429741     33.930000   -121.800000      1.196000
50%        2.818116     34.260000   -118.490000      1.797000
75%        3.282261     37.710000   -118.010000      2.647250
max     1243.333333     41.950000   -114.310000      5.000010
```

```
print(diabetes_df.info())
print(diabetes_df.describe())
```

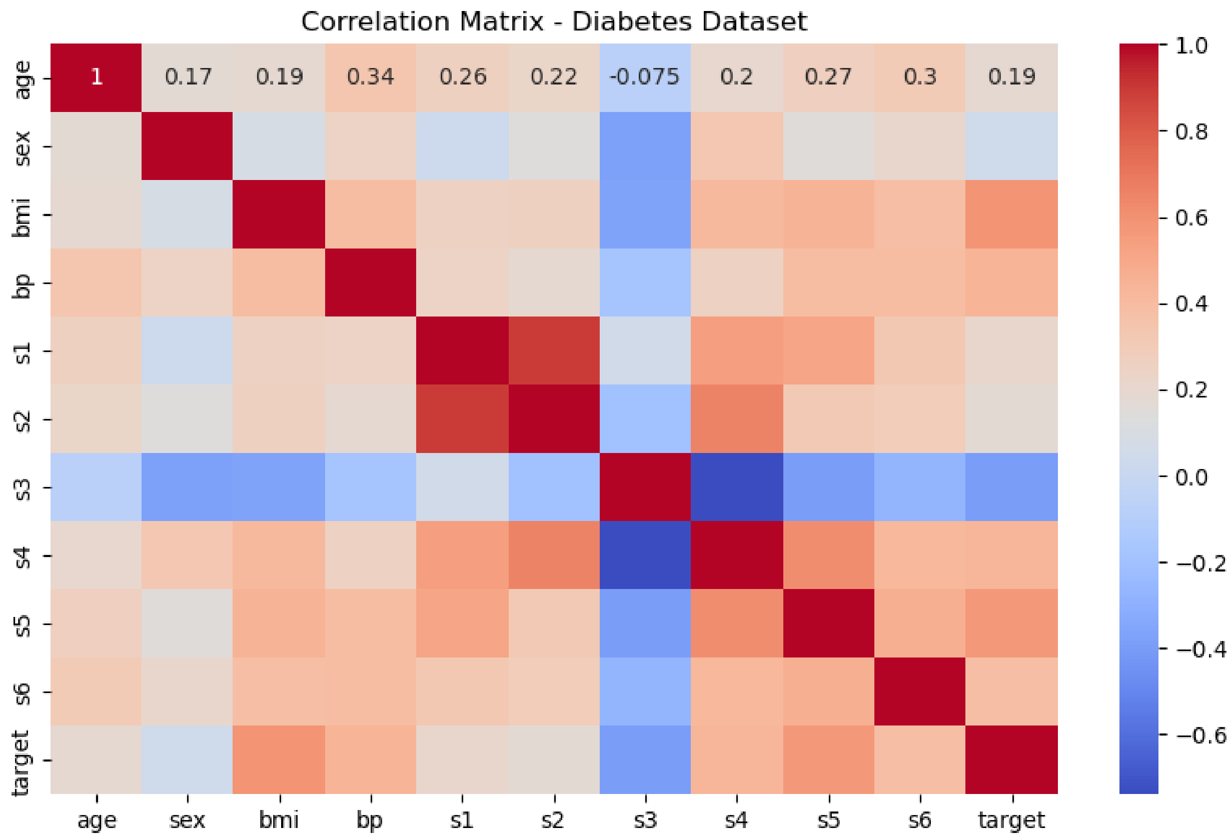
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 442 entries, 0 to 441
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   age     442 non-null    float64
 1   sex     442 non-null    float64
 2   bmi     442 non-null    float64
 3   bp      442 non-null    float64
 4   s1      442 non-null    float64
 5   s2      442 non-null    float64
 6   s3      442 non-null    float64
 7   s4      442 non-null    float64
 8   s5      442 non-null    float64
 9   s6      442 non-null    float64
 10  target  442 non-null    float64
dtypes: float64(11)
memory usage: 38.1 KB
None
                age           sex           bmi            bp
s1  \
count  4.420000e+02  4.420000e+02  4.420000e+02  4.420000e+02
4.420000e+02
mean  -2.511817e-19  1.230790e-17 -2.245564e-16 -4.797570e-17 -
1.381499e-17
std    4.761905e-02  4.761905e-02  4.761905e-02  4.761905e-02
4.761905e-02
min   -1.072256e-01 -4.464164e-02 -9.027530e-02 -1.123988e-01 -
1.267807e-01
25%   -3.729927e-02 -4.464164e-02 -3.422907e-02 -3.665608e-02 -
3.424784e-02
50%    5.383060e-03 -4.464164e-02 -7.283766e-03 -5.670422e-03 -
4.320866e-03
75%    3.807591e-02  5.068012e-02  3.124802e-02  3.564379e-02
2.835801e-02
max    1.107267e-01  5.068012e-02  1.705552e-01  1.320436e-01
1.539137e-01
```

```
                 s2            s3            s4            s5
s6  \
count  4.420000e+02  4.420000e+02  4.420000e+02  4.420000e+02
4.420000e+02
mean   3.918434e-17 -5.777179e-18 -9.042540e-18  9.293722e-17
1.130318e-17
std    4.761905e-02  4.761905e-02  4.761905e-02  4.761905e-02
4.761905e-02
min   -1.156131e-01 -1.023071e-01 -7.639450e-02 -1.260971e-01 -
1.377672e-01
25%   -3.035840e-02 -3.511716e-02 -3.949338e-02 -3.324559e-02 -
3.317903e-02
50%   -3.819065e-03 -6.584468e-03 -2.592262e-03 -1.947171e-03 -
1.077698e-03
75%    2.984439e-02  2.931150e-02  3.430886e-02  3.243232e-02
2.791705e-02
max    1.987880e-01  1.811791e-01  1.852344e-01  1.335973e-01
1.356118e-01

           target
count  442.000000
mean   152.133484
std     77.093005
min     25.000000
25%     87.000000
50%    140.500000
75%    211.500000
max    346.000000
```

```python
plt.figure(figsize=(10, 6))
sns.heatmap(california_df.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Matrix - California Housing Dataset")
plt.show()
```

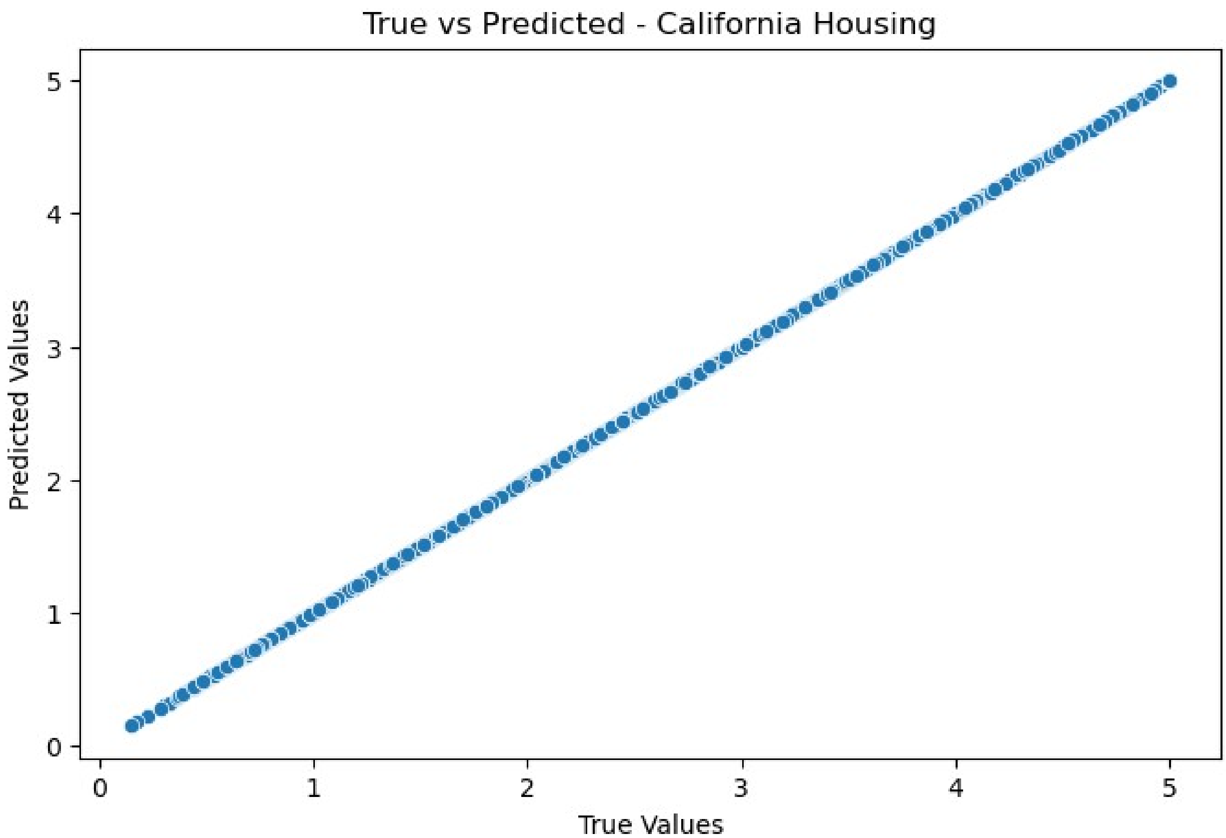



```
plt.figure(figsize=(10, 6))
sns.heatmap(diabetes_df.corr(), annot=True, cmap="coolwarm")
plt.title("Correlation Matrix - Diabetes Dataset")
plt.show()
```

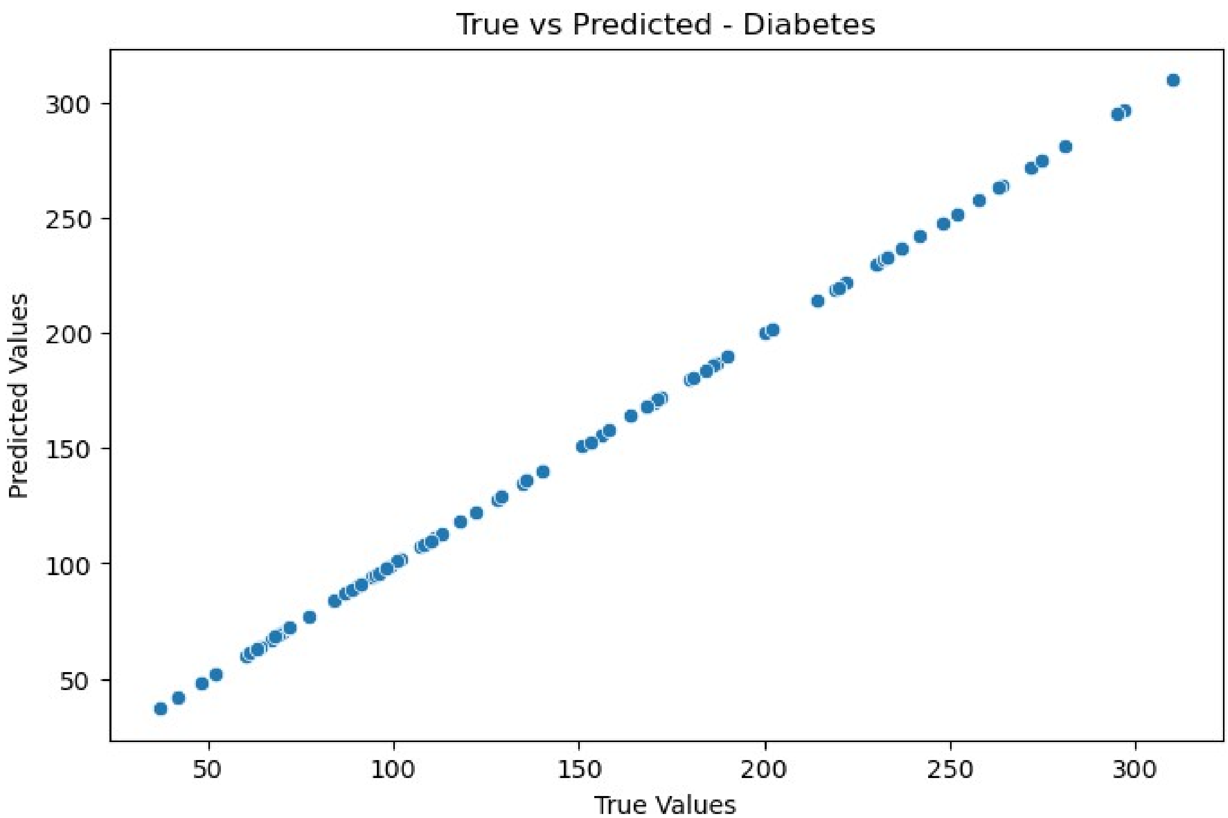



```
X_california = california_df.drop('target', axis=1)
y_california = california_df['target']

X_diabetes = diabetes_df.drop('target', axis=1)
y_diabetes = diabetes_df['target']

X_train_california, X_test_california, y_train_california,
y_test_california = train_test_split(X_california, y_california,
test_size=0.2, random_state=42)
X_train_diabetes, X_test_diabetes, y_train_diabetes, y_test_diabetes =
train_test_split(X_diabetes, y_diabetes, test_size=0.2,
random_state=42)

scaler = StandardScaler()
X_train_california = scaler.fit_transform(X_train_california)
X_test_california = scaler.transform(X_test_california)

X_train_diabetes = scaler.fit_transform(X_train_diabetes)
X_test_diabetes = scaler.transform(X_test_diabetes)

plt.figure(figsize=(8, 5))
sns.scatterplot(x=y_test_california, y=y_test_california)  # Replace
this with predicted values later
plt.title('True vs Predicted - California Housing')
plt.xlabel('True Values')
```

```
plt.ylabel('Predicted Values')
plt.show()
```




```
plt.figure(figsize=(8, 5))
sns.scatterplot(x=y_test_diabetes, y=y_test_diabetes)  # Replace this
with predicted values later
plt.title('True vs Predicted - Diabetes')
plt.xlabel('True Values')
plt.ylabel('Predicted Values')
plt.show()
```




```
model_california = LinearRegression()
model_diabetes = LinearRegression()

model_california.fit(X_train_california, y_train_california)
model_diabetes.fit(X_train_diabetes, y_train_diabetes)

LinearRegression()

y_pred_california = model_california.predict(X_test_california)
y_pred_diabetes = model_diabetes.predict(X_test_diabetes)

param_grid = {
    'fit_intercept': [True, False],
    'copy_X': [True, False],
    'positive': [True, False]
}

grid_search_california = GridSearchCV(LinearRegression(), param_grid,
cv=5, scoring='neg_mean_squared_error')
grid_search_california.fit(X_train_california, y_train_california)
print(f"Best hyperparameters for California dataset:
{grid_search_california.best_params_}")

Best hyperparameters for California dataset: {'copy_X': True,
'fit_intercept': True, 'positive': False}
```

```
grid_search_diabetes = GridSearchCV(LinearRegression(), param_grid,
cv=5, scoring='neg_mean_squared_error')
grid_search_diabetes.fit(X_train_diabetes, y_train_diabetes)
print(f"Best hyperparameters for Diabetes dataset:
{grid_search_diabetes.best_params_}")
```

```
Best hyperparameters for Diabetes dataset: {'copy_X': True,
'fit_intercept': True, 'positive': False}
```

```
best_model_california = grid_search_california.best_estimator_
best_model_diabetes = grid_search_diabetes.best_estimator_

y_pred_california_best =
best_model_california.predict(X_test_california)
y_pred_diabetes_best = best_model_diabetes.predict(X_test_diabetes)

mse_california = mean_squared_error(y_test_california,
y_pred_california_best)
r2_california = r2_score(y_test_california, y_pred_california_best)
print(f"California Model Performance:\nMSE: {mse_california}\nR²:
{r2_california}")
```

```
California Model Performance:
MSE: 0.5558915986952442
R²: 0.575787706032451
```

```
mse_diabetes = mean_squared_error(y_test_diabetes,
y_pred_diabetes_best)
r2_diabetes = r2_score(y_test_diabetes, y_pred_diabetes_best)
print(f"Diabetes Model Performance:\nMSE: {mse_diabetes}\nR²:
{r2_diabetes}")
```

```
Diabetes Model Performance:
MSE: 2900.1936284934823
R²: 0.4526027629719926
```

```
models = ['California Housing', 'Diabetes']
mse_values = [mse_california, mse_diabetes]
r2_values = [r2_california, r2_diabetes]

comparison_df = pd.DataFrame({
    'Model': models,
    'MSE': mse_values,
    'R²': r2_values
})

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

sns.barplot(x='Model', y='MSE', data=comparison_df, ax=axes[0],
palette='Blues')
axes[0].set_title('Mean Squared Error (MSE) Comparison')
```

```
sns.barplot(x='Model', y='R²', data=comparison_df, ax=axes[1],
palette='Greens')
axes[1].set_title('R² Score Comparison')

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