```
#Import libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

```
# Load the dataset
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['Target'] = data.target  # Adding the target variable to the DataFrame

# Display the first few rows
df.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	Target
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

```
# Print the shape of the DataFrame
print("Shape of the dataset:", df.shape)
```

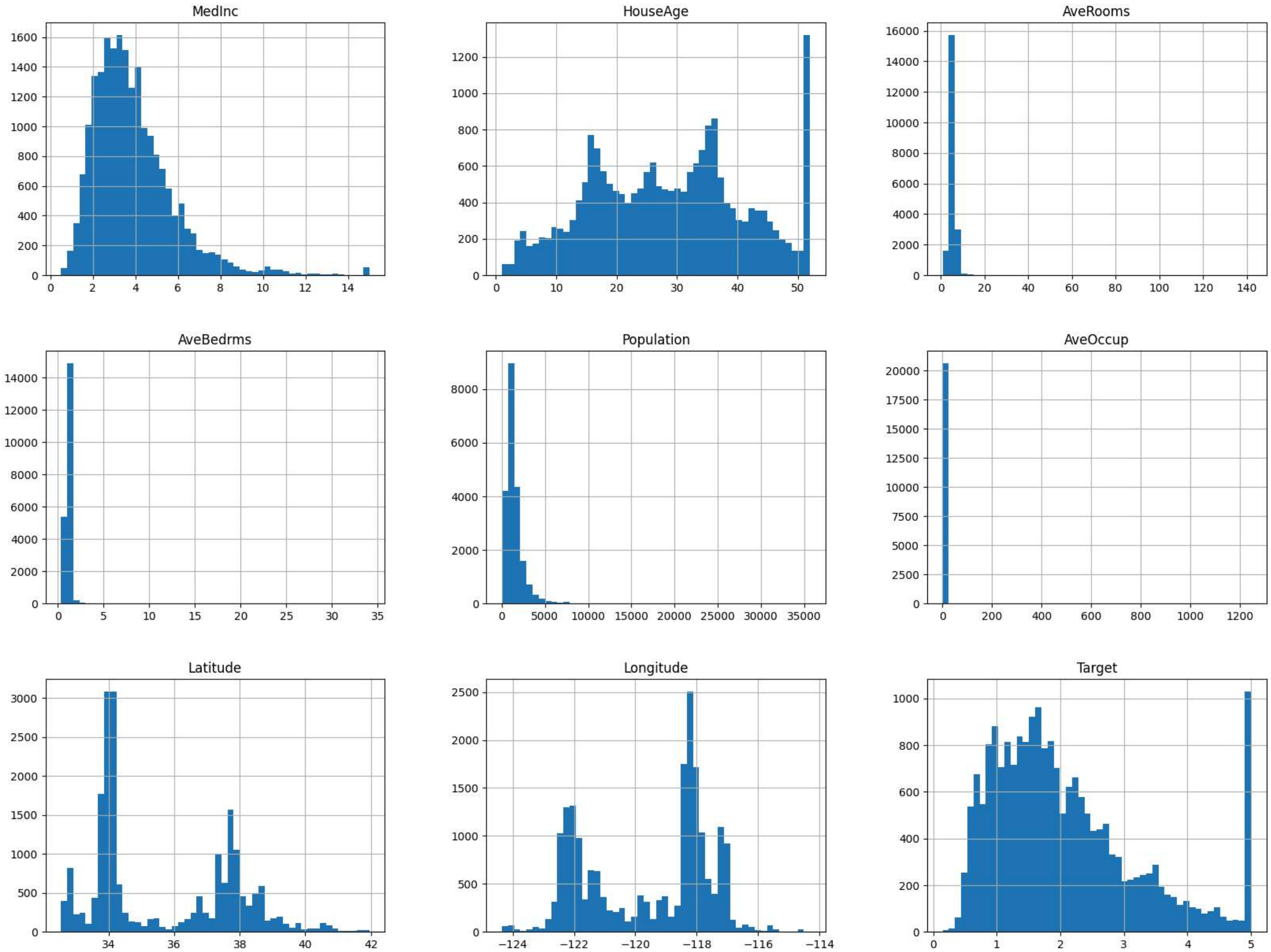
```
Shape of the dataset: (20640, 9)
```

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

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

```
# Histograms for each feature
df.hist(bins=50, figsize=(20,15))
plt.show()

# Corrected Scatter plot for geographical data
plt.figure(figsize=(10,7))
plt.scatter(df['Longitude'], df['Latitude'], alpha=0.4,
            s=df['Population']/100,  # Population as size, scaled down for better visualization
            c=df['Target'], cmap=plt.get_cmap('jet'), label='Population')
plt.colorbar(label='Target (Median House Value)')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.legend()
plt.show()
```




```
# Splitting the dataset into training and testing sets
train_set, test_set = train_test_split(df, test_size=0.2, random_state=42)

# Features and target variable
X_train = train_set.drop("Target", axis=1)
y_train = train_set["Target"]
X_test = test_set.drop("Target", axis=1)
y_test = test_set["Target"]
```

```
# Linear Regression model
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# Decision Tree Regressor
tree_reg = DecisionTreeRegressor(random_state=42)
tree_reg.fit(X_train, y_train)

# Random Forest Regressor
forest_reg = RandomForestRegressor(random_state=42)
forest_reg.fit(X_train, y_train)
```

```
▾        RandomForestRegressor
RandomForestRegressor(random_state=42)
```

```
#Evaluate models and Determine Best Model
def evaluate_model(model, X, y):
    predictions = model.predict(X)
    mse = mean_squared_error(y, predictions)
    rmse = np.sqrt(mse)
    r2 = r2_score(y, predictions)
    return rmse, r2

lin_rmse, lin_r2 = evaluate_model(lin_reg, X_test, y_test)
tree_rmse, tree_r2 = evaluate_model(tree_reg, X_test, y_test)
forest_rmse, forest_r2 = evaluate_model(forest_reg, X_test, y_test)

print(f"Linear Regression - RMSE: {lin_rmse:.2f}, R-squared: {lin_r2:.2f}")
print(f"Decision Tree - RMSE: {tree_rmse:.2f}, R-squared: {tree_r2:.2f}")
print(f"Random Forest - RMSE: {forest_rmse:.2f}, R-squared: {forest_r2:.2f}")

best_model = min(
    ('Linear Regression', lin_rmse),
    ('Decision Tree', tree_rmse),
    ('Random Forest', forest_rmse),
    key=lambda item: item[1]
)
print(f"The best model based on RMSE is: {best_model[0]} with an RMSE of {best_model[1]:.2f}")
```

```
Linear Regression - RMSE: 0.75, R-squared: 0.58
Decision Tree - RMSE: 0.70, R-squared: 0.62
Random Forest - RMSE: 0.51, R-squared: 0.81
The best model based on RMSE is: Random Forest with an RMSE of 0.51
```

```
# EXAMPLE 1 :
# Assuming 'data.feature_names' contains the correct order of your feature names
feature_names = data.feature_names

# Example new data point with all positive values (replace with actual data values)
# Feature Values:
# - MedInc: 3.0 (median income in tens of thousands)
# - HouseAge: 20 (median age of a house in years)
# - AveRooms: 5 (average number of rooms per house)
# - AveBedrms: 1.2 (average number of bedrooms per house)
# - Population: 800 (population in the area)
# - AveOccup: 3 (average number of occupants per household)
# - Latitude: 35.5 (geographic latitude of the area)
# - Longitude: -120.5 (geographic longitude of the area, typical for California)
new_data_values = [3.0, 20, 5, 1.2, 800, 3, 35.5, -120.5]  # Example feature values

# Create a DataFrame with the right column names
new_data_df = pd.DataFrame([new_data_values], columns=feature_names)

# Predict using the trained models
lin_pred = lin_reg.predict(new_data_df)
tree_pred = tree_reg.predict(new_data_df)
forest_pred = forest_reg.predict(new_data_df)

# Print predictions
print("Predictions for new data:")
print(f"Linear Regression Prediction: {lin_pred[0]:.2f}")
print(f"Decision Tree Prediction: {tree_pred[0]:.2f}")
print(f"Random Forest Prediction: {forest_pred[0]:.2f}")
```

```
Predictions for new data:
Linear Regression Prediction: 2.19
Decision Tree Prediction: 1.62
Random Forest Prediction: 1.63
```

```
# EXAMPLE 2: This example represents a typical middle-income suburban area with moderate house sizes and ages.
# Feature Values:
# - Median Income: 4.0 (moderate income)
# - House Age: 25 years (moderately old)
# - Average Rooms: 5 (average-sized homes)
# - Average Bedrooms: 2 (typical for small families)
# - Population: 1200 (suburban area population)
# - Average Occupancy: 3 (average family size)
# - Latitude: 34.05 (approximate for a location like suburban Los Angeles)
# - Longitude: -118.25 (approximate for a location like suburban Los Angeles)

# Create DataFrame for new data based on the provided features
new_data_suburban = pd.DataFrame([[4.0, 25, 5, 2, 1200, 3, 34.05, -118.25]], columns=feature_names)

# Predict using the trained models
lin_pred_suburban = lin_reg.predict(new_data_suburban)
tree_pred_suburban = tree_reg.predict(new_data_suburban)
forest_pred_suburban = forest_reg.predict(new_data_suburban)

# Output predictions for middle-income suburban area
print("Predictions for middle-income suburban area:")
print(f"Linear Regression Prediction: {lin_pred_suburban[0]:.2f}")
print(f"Decision Tree Prediction: {tree_pred_suburban[0]:.2f}")
print(f"Random Forest Prediction: {forest_pred_suburban[0]:.2f}")



# EXAMPLE 3: This example depicts a high-income urban area, often characterized by newer constructions and higher household income.
# Feature Values:
# - Median Income: 8.5 (very high income)
# - House Age: 15 years (relatively new constructions)
# - Average Rooms: 6 (larger, more spacious apartments or homes)
# - Average Bedrooms: 3 (suitable for larger families or shared accommodations)
# - Population: 2500 (high urban population density)
# - Average Occupancy: 2.5 (typical for urban areas, indicating smaller households or shared housing)
# - Latitude: 37.77 (approximate for a location like San Francisco)
# - Longitude: -122.41 (approximate for a location like San Francisco)



# Create DataFrame for new data based on the provided features
new_data_urban = pd.DataFrame([[8.5, 15, 6, 3, 2500, 2.5, 37.77, -122.41]], columns=feature_names)

# Predict using the trained models
lin_pred_urban = lin_reg.predict(new_data_urban)
tree_pred_urban = tree_reg.predict(new_data_urban)
forest_pred_urban = forest_reg.predict(new_data_urban)

# Output predictions for high-income urban area
print("Predictions for high-income urban area:")
print(f"Linear Regression Prediction: {lin_pred_urban[0]:.2f}")
print(f"Decision Tree Prediction: {tree_pred_urban[0]:.2f}")
print(f"Random Forest Prediction: {forest_pred_urban[0]:.2f}")
```

```
Predictions for high-income urban area:
Linear Regression Prediction: 5.77
Decision Tree Prediction: 3.92
Random Forest Prediction: 4.48
```