```
import numpy as np

import pandas as pd

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.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

import statsmodels.api as sm

from statsmodels.stats.outliers_influence import variance_inflation_factor


# Set seed for reproducibility

np.random.seed(42)


# Load dataset

housing = fetch_california_housing()

X = housing.data

y = housing.target

feature_names = housing.feature_names


# Create a DataFrame for easier handling

df = pd.DataFrame(X, columns=feature_names)

df['target'] = y


print("Dataset Information:")

print(f"Number of samples: {X.shape[0]}")

print(f"Number of features: {X.shape[1]}")

print(f"Features: {feature_names}")

print("\nData Preview:")

print(df.head())

print("\nStatistical Summary:")

print(df.describe())


# Check for missing values

print("\nMissing Values:")

print(df.isnull().sum())


# Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Standardize features

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)


# Build and train the multiple linear regression model

model = LinearRegression()

model.fit(X_train_scaled, y_train)


# Make predictions

y_pred = model.predict(X_test_scaled)


# Evaluate the model

mse = mean_squared_error(y_test, y_pred)

rmse = np.sqrt(mse)

mae = mean_absolute_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


print("\nModel Evaluation:")

print(f"Mean Squared Error: {mse:.4f}")

print(f"Root Mean Squared Error: {rmse:.4f}")

print(f"Mean Absolute Error: {mae:.4f}")

print(f"R-squared: {r2:.4f}")


# Display model coefficients

coefficients = pd.DataFrame({

'Feature': feature_names,

'Coefficient': model.coef_

}).sort_values('Coefficient', ascending=False)


print("\nModel Coefficients:")

print(coefficients)

X_train_sm = sm.add_constant(X_train_scaled) # Adding intercept term

sm_model = sm.OLS(y_train, X_train_sm).fit()


print("\nDetailed Statistical Summary:")

print(sm_model.summary())

def calculate_vif(X):

vif = pd.DataFrame()

vif["Variable"] = X.columns

vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

return vif


X_train_df = pd.DataFrame(X_train, columns=feature_names)

vif_data = calculate_vif(X_train_df)

print("\nVariance Inflation Factors:")

print(vif_data)
```

```
# Visualizations

# 1. Actual vs Predicted values

plt.figure(figsize=(10, 6))

plt.scatter(y_test, y_pred, alpha=0.6)

plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)

plt.xlabel('Actual Values')

plt.ylabel('Predicted Values')

plt.title('Actual vs Predicted Values')

plt.tight_layout()


# 2. Residuals visualization

residuals = y_test - y_pred

plt.figure(figsize=(10, 6))

plt.scatter(y_pred, residuals, alpha=0.6)

plt.hlines(y=0, xmin=min(y_pred), xmax=max(y_pred), colors='r', linestyles='--')

plt.xlabel('Predicted Values')

plt.ylabel('Residuals')

plt.title('Residual Plot')

plt.tight_layout()


# 3. Residuals distribution

plt.figure(figsize=(10, 6))

sns.histplot(residuals, kde=True)

plt.xlabel('Residuals')

plt.ylabel('Frequency')

plt.title('Distribution of Residuals')

plt.tight_layout()


# 4. Feature coefficients visualization

plt.figure(figsize=(12, 8))

sns.barplot(x='Coefficient', y='Feature', data=coefficients)

plt.title('Feature Coefficients')

plt.tight_layout()


# 5. Correlation heatmap

plt.figure(figsize=(12, 10))

correlation_matrix = df.corr()

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Correlation Matrix')

plt.tight_layout()


# 6. Partial regression plots to visualize the relationship between each feature and the target

fig, axes = plt.subplots(4, 2, figsize=(15, 20))

axes = axes.flatten()
```

```
for i, feature in enumerate(feature_names):

if i < len(axes):

sns.regplot(x=df[feature], y=df['target'], ax=axes[i])

axes[i].set_title(f'Partial Regression Plot: {feature}')


plt.tight_layout()


# Implementing Multiple Linear Regression from scratch

class CustomMultipleLinearRegression:

def __init__(self, learning_rate=0.01, num_iterations=1000):

self.learning_rate = learning_rate

self.num_iterations = num_iterations

self.weights = None

self.bias = None

self.cost_history = []

def fit(self, X, y):

# Initialize parameters

num_samples, num_features = X.shape

self.weights = np.zeros(num_features)

self.bias = 0

self.cost_history = []

# Gradient descent

for i in range(self.num_iterations):

# Linear model

y_predicted = np.dot(X, self.weights) + self.bias

# Compute cost (Mean Squared Error)

cost = (1 / (2 * num_samples)) * np.sum((y_predicted - y) ** 2)

self.cost_history.append(cost)

# Compute gradients

dw = (1 / num_samples) * np.dot(X.T, (y_predicted - y))

db = (1 / num_samples) * np.sum(y_predicted - y)

# Update parameters

self.weights -= self.learning_rate * dw

self.bias -= self.learning_rate * db

def predict(self, X):

return np.dot(X, self.weights) + self.bias

def get_cost_history(self):

return self.cost_history


# Train and evaluate the custom model

custom_model = CustomMultipleLinearRegression(learning_rate=0.01, num_iterations=1000)

custom_model.fit(X_train_scaled, y_train)

custom_y_pred = custom_model.predict(X_test_scaled)
```

```python
# Evaluate custom model

custom_mse = mean_squared_error(y_test, custom_y_pred)

custom_rmse = np.sqrt(custom_mse)

custom_r2 = r2_score(y_test, custom_y_pred)


print("\nCustom Multiple Linear Regression Model:")

print(f"Mean Squared Error: {custom_mse:.4f}")

print(f"Root Mean Squared Error: {custom_rmse:.4f}")

print(f"R-squared: {custom_r2:.4f}")


# Plot cost history

plt.figure(figsize=(10, 6))

plt.plot(custom_model.get_cost_history())

plt.xlabel('Iteration')

plt.ylabel('Cost')

plt.title('Cost History During Training')

plt.tight_layout()
```

```
^[[Aishadpande@Argos:~/Documents/dev/practicals/PS$ /bin/python /home/ishadpande/Documents/dev/practicals/PS/6.py
Dataset Information:
Number of samples: 20640
Number of features: 8
Features: ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude']

Data Preview:
   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

Statistical Summary:
             MedInc      HouseAge      AveRooms     AveBedrms    Population      AveOccup      Latitude     Longitude        target
count  20640.000000  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      2.068558
std        1.899822     12.585558      2.474173      0.473911   1132.462122     10.386050      2.135952      2.003532      1.153956
min        0.499900      1.000000      0.846154      0.333333      3.000000      0.692308     32.540000   -124.350000      0.149990
25%        2.563400     18.000000      4.440716      1.006079    787.000000      2.429741     33.930000   -121.800000      1.196000
50%        3.534800     29.000000      5.229129      1.048780   1166.000000      2.818116     34.260000   -118.490000      1.797000
75%        4.743250     37.000000      6.052381      1.099526   1725.000000      3.282261     37.710000   -118.010000      2.647250
max       15.000100     52.000000    141.909091     34.066667  35682.000000   1243.333333     41.950000   -114.310000      5.000010

Missing Values:
```

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

Model Evaluation:
Mean Squared Error: 0.5559
Root Mean Squared Error: 0.7456
Mean Absolute Error: 0.5332
R-squared: 0.5758

Model Coefficients:
      Feature  Coefficient
0      MedInc     0.854383
3   AveBedrms     0.339259
1    HouseAge     0.122546
4  Population    -0.002308
5    AveOccup    -0.040829
2    AveRooms    -0.294410
7   Longitude    -0.869842
6    Latitude    -0.896929
```

```
Variance Inflation Factors:
     Variable         VIF
0      MedInc   11.831609
1    HouseAge    7.155405
2    AveRooms   46.792373
3   AveBedrms   48.332634
4  Population    2.915730
5    AveOccup    1.080609
6    Latitude  560.583263
7   Longitude  641.224254

Custom Multiple Linear Regression Model:
Mean Squared Error: 0.5672
Root Mean Squared Error: 0.7531
R-squared: 0.5672
```