Experiment 7: Build a Multiple Linear Regression Model and Predict

the Dependent Variable using the Gradient Descent Method

ANJALI K - 24MCAR0186

In [ ]:

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

In [8]:

```
# ======================================
# Step 1: Load and Preprocess Dataset
# ======================================

# Load California Housing dataset
california = fetch_california_housing()
data = pd.DataFrame(california.data, columns=california.feature_names)
data['PRICE'] = california.target

print("Sample Data:")
print(data.head())

# Handle missing values (if any)
data = data.dropna()

# Select features and target
```

```python
X = data[['MedInc', 'AveRooms', 'HouseAge', 'AveOccup']]  # selecting a few features for simplicity
y = data['PRICE']

# Normalize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```
Sample Data:
   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  PRICE  
0    -122.23  4.526  
1    -122.22  3.585  
2    -122.24  3.521  
3    -122.25  3.413  
4    -122.25  3.422  
```

In [9]:

```python
# ======================================
# Step 2: Implement Gradient Descent for MLR
# ======================================

def gradient_descent(X, y, learning_rate=0.01, iterations=1000):
    m, n = X.shape
    X_b = np.c_[np.ones((m, 1)), X]  # Add intercept term
    theta = np.zeros((n + 1, 1))     # Initialize coefficients
    y = y.values.reshape(-1, 1)
    cost_history = []

    for i in range(iterations):
        y_pred = X_b.dot(theta)
        error = y_pred - y
        gradients = (1/m) * X_b.T.dot(error)
        theta -= learning_rate * gradients
```

```python
        cost = (1/(2*m)) * np.sum(error ** 2)
        cost_history.append(cost)

    return theta, cost_history

# Train model
theta, cost_history = gradient_descent(X_train, y_train, learning_rate=0.05, iterations=2000)
```

In [10]:

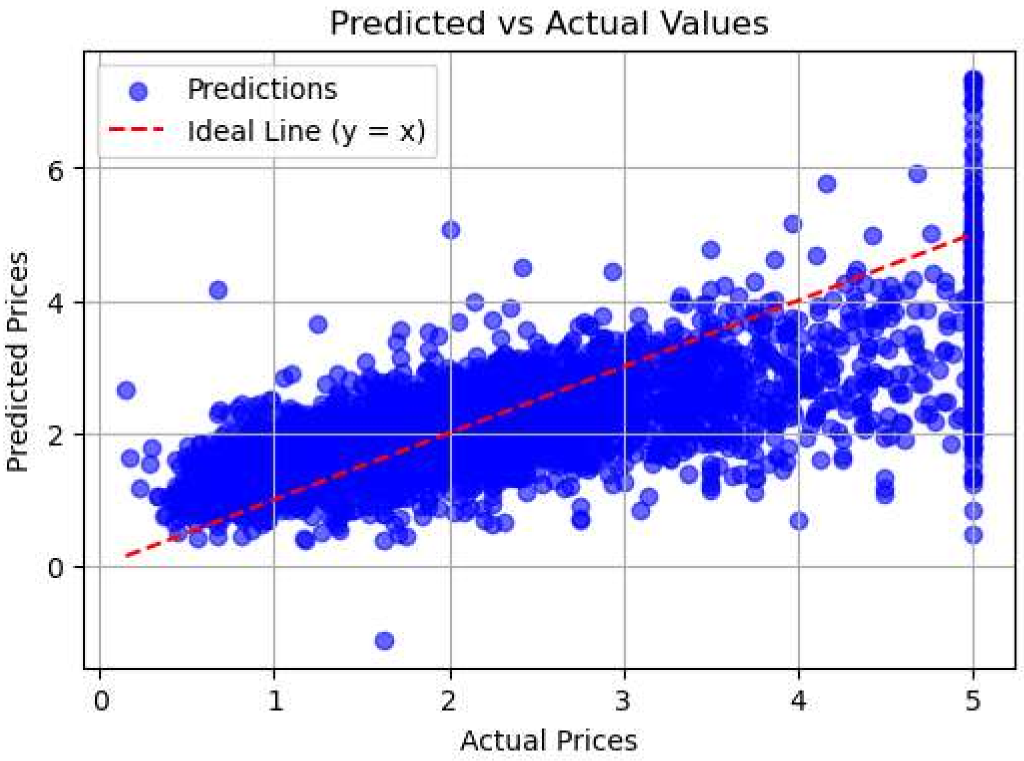
```python
# ======================================
# Step 3: Model Evaluation
# ======================================

# Predictions
X_test_b = np.c_[np.ones((X_test.shape[0], 1)), X_test]
y_pred = X_test_b.dot(theta)

# Compute metrics
r2 = r2_score(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print("\nModel Coefficients:")
print(f"Intercept: {theta[0][0]:.4f}")
for i, col in enumerate(X.columns):
    print(f"{col}: {theta[i+1][0]:.4f}")

print(f"\nR² Score: {r2:.4f}")
print(f"MAE: {mae:.4f}")
print(f"RMSE: {rmse:.4f}")
```

```
Model Coefficients:
Intercept: 2.0683
MedInc: 0.8463
AveRooms: -0.0702
HouseAge: 0.2127
AveOccup: -0.0430

R² Score: 0.4983
MAE: 0.6025
RMSE: 0.8108
```

In [11]:
```python
# ======================================
# Step 4: Plots
# ======================================

# Cost vs Iterations
plt.figure(figsize=(6,4))
plt.plot(range(len(cost_history)), cost_history, color='blue')
plt.title("Cost (Loss) vs Iterations")
plt.xlabel("Iterations")
plt.ylabel("Cost")
plt.grid(True)
plt.show()
```

In [12]:
```python
# Predicted vs Actual
plt.figure(figsize=(6,4))
plt.scatter(y_test, y_pred, alpha=0.6, color='blue', label='Predictions')
```

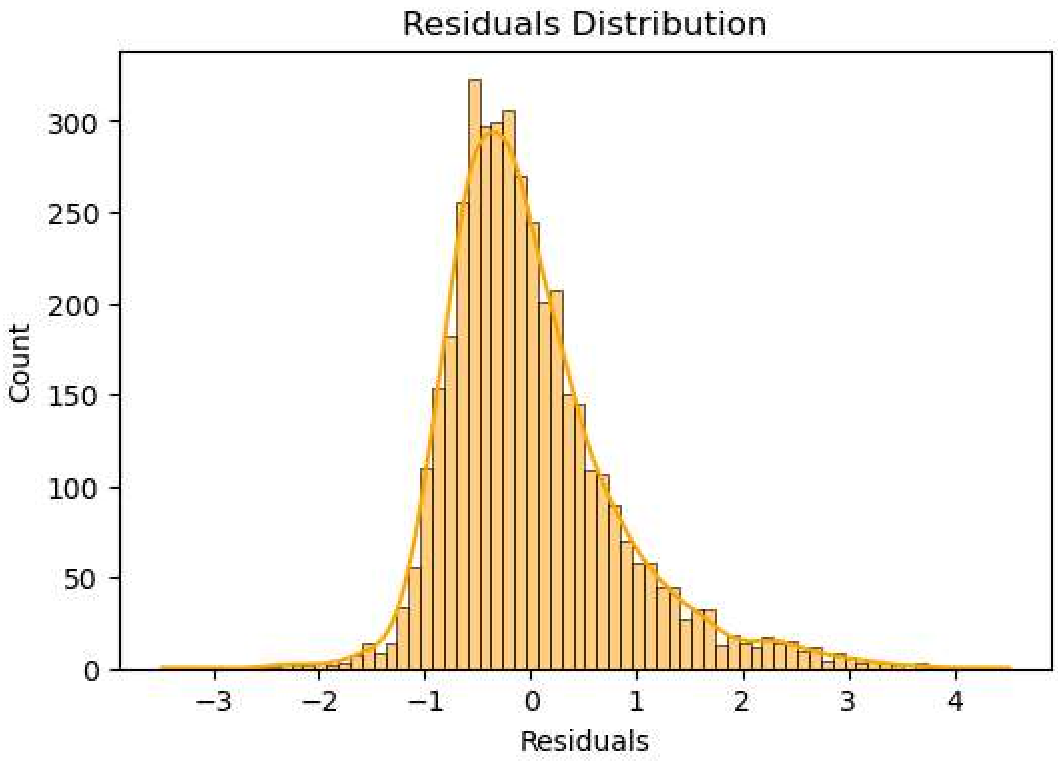
```
plt.plot([y_test.min(), y_test.max()],
         [y_test.min(), y_test.max()],
         color='red', linestyle='--', label='Ideal Line (y = x)')
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Predicted vs Actual Values")
plt.legend()
plt.grid(True)
plt.show()
```



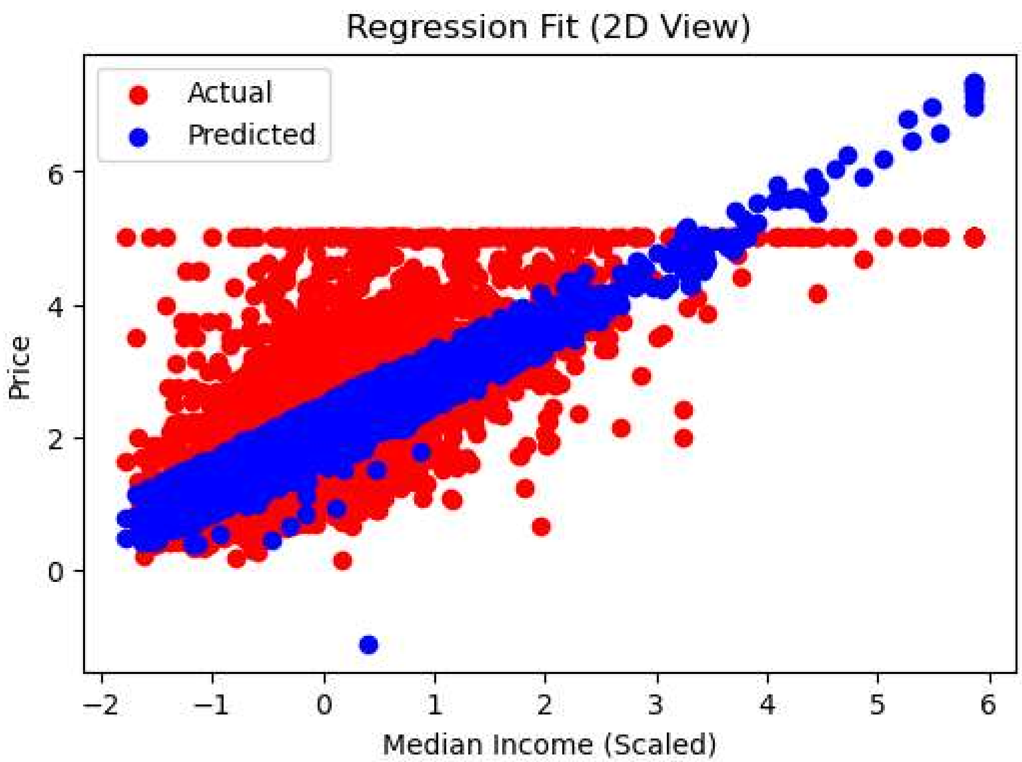
In [15]:

```
# Residual Plot
residuals = y_test - y_pred.flatten()
plt.figure(figsize=(6,4))
sns.histplot(residuals, kde=True, color='orange')
plt.title("Residuals Distribution")
```

```
plt.xlabel("Residuals")
plt.show()
```



In [14]:
```
# Optional: Scatter 2D (using 1 feature for visualization)
plt.figure(figsize=(6,4))
plt.scatter(X_test[:, 0], y_test, color='red', label='Actual')
plt.scatter(X_test[:, 0], y_pred, color='blue', label='Predicted')
plt.xlabel("Median Income (Scaled)")
plt.ylabel("Price")
plt.title("Regression Fit (2D View)")
plt.legend()
plt.show()
```


Regression Fit (2D View)
Actual
Predicted
Price
Median Income (Scaled)
6
4
2
0
−2
−1
0
1
2
3
4
5
6