In [42]:
```
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_e
from sklearn.linear_model import LinearRegression
```

In [43]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
import numpy as np
```

In [44]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_e
```

In [45]:
```
#A) DATA PROCESSING:

# 1) Load train & test data
train = pd.read_csv("california_housing_train.csv")
test = pd.read_csv("california_housing_test.csv")

print("Train shape:", train.shape)
print("Test shape:", test.shape)
X_train = train.drop(columns=['median_house_value'])
y_train = train['median_house_value']

X_test = test.drop(columns=['median_house_value'])
y_test = test['median_house_value']

# 2) Standardize features (zero mean, unit variance)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)


# 3) Add intercept (bias) column of ones
X_train_scaled.insert(0, 'intercept', 1)
X_test_scaled.insert(0, 'intercept', 1)

print("Processed Train Features Shape:", X_train_scaled.shape)
print("Processed Test Features Shape:", X_test_scaled.shape)
```

```
Train shape: (17000, 9)
Test shape: (3000, 9)
Processed Train Features Shape: (17000, 9)
Processed Test Features Shape: (3000, 9)
```

In [46]:
```
train.head()
```

Out[46]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	populati
0	-114.31	34.19	15.0	5612.0	1283.0	101
1	-114.47	34.40	19.0	7650.0	1901.0	112
2	-114.56	33.69	17.0	720.0	174.0	33
3	-114.57	33.64	14.0	1501.0	337.0	51
4	-114.57	33.57	20.0	1454.0	326.0	62

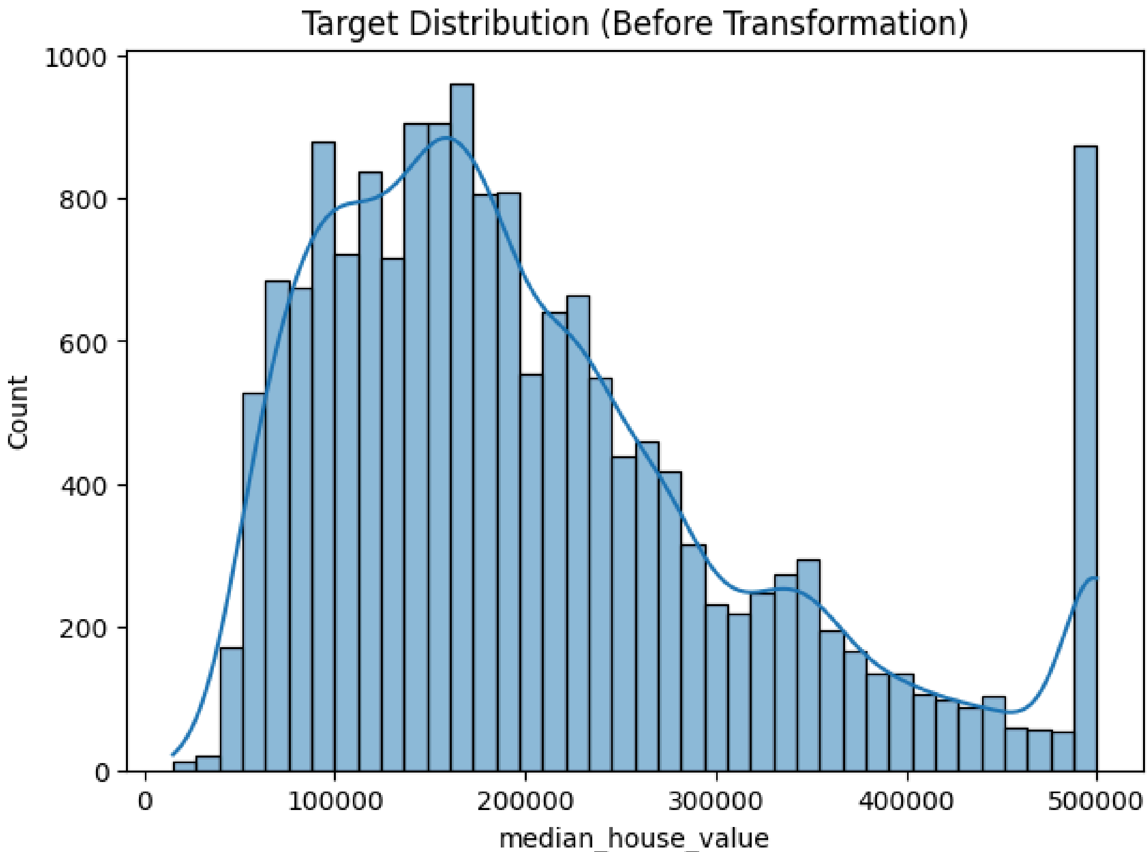
In [47]:

```
# === Dataset Information ===
print("Info:")
print(train.info())
print("\nMissing Values:\n", train.isnull().sum())
print("\nSummary Stats:\n", train.describe())
```

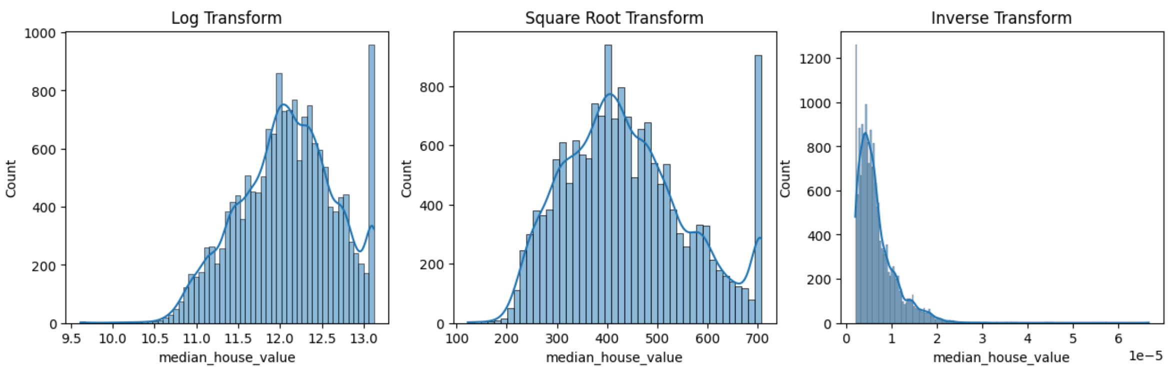
```
Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17000 entries, 0 to 16999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           17000 non-null  float64
 1   latitude            17000 non-null  float64
 2   housing_median_age  17000 non-null  float64
 3   total_rooms         17000 non-null  float64
 4   total_bedrooms      17000 non-null  float64
 5   population          17000 non-null  float64
 6   households          17000 non-null  float64
 7   median_income       17000 non-null  float64
 8   median_house_value  17000 non-null  float64
dtypes: float64(9)
memory usage: 1.2 MB
None

Missing Values:
 longitude             0
latitude              0
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
dtype: int64

Summary Stats:
           longitude      latitude  housing_median_age   total_rooms  \
count  17000.000000  17000.000000        17000.000000  17000.000000   
mean    -119.562108     35.625225           28.589353   2643.664412   
std        2.005166      2.137340           12.586937   2179.947071   
min     -124.350000     32.540000            1.000000      2.000000   
25%     -121.790000     33.930000           18.000000   1462.000000   
50%     -118.490000     34.250000           29.000000   2127.000000   
75%     -118.000000     37.720000           37.000000   3151.250000   
max     -114.310000     41.950000           52.000000  37937.000000   

       total_bedrooms    population    households  median_income  \
count    17000.000000  17000.000000  17000.000000   17000.000000   
mean       539.410824   1429.573941    501.221941       3.883578   
std        421.499452   1147.852959    384.520841       1.908157   
min          1.000000      3.000000      1.000000       0.499900   
25%        297.000000    790.000000    282.000000       2.566375   
50%        434.000000   1167.000000    409.000000       3.544600   
75%        648.250000   1721.000000    605.250000       4.767000   
max       6445.000000  35682.000000   6082.000000      15.000100   

       median_house_value  
count        17000.000000  
mean        207300.912353  
std         115983.764387  
min          14999.000000  
25%         119400.000000  
50%         180400.000000  
```

```
75%          265000.000000
max          500001.000000
```

In [48]:
```
# === Target Distribution ===
TARGET_NAME = train.columns[-1]  # assume last col is target

plt.figure(figsize=(7,5))
sns.histplot(train[TARGET_NAME], kde=True, bins=40)
plt.title("Target Distribution (Before Transformation)")
plt.show()
```

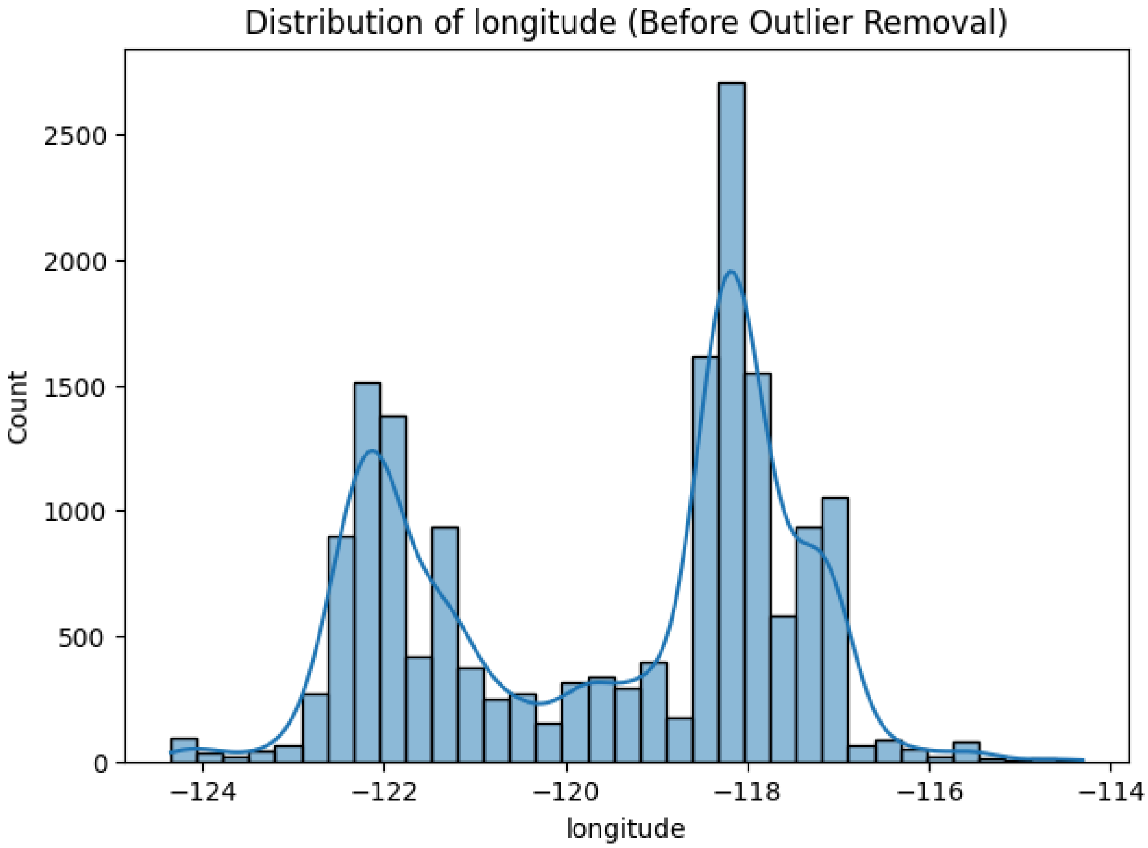


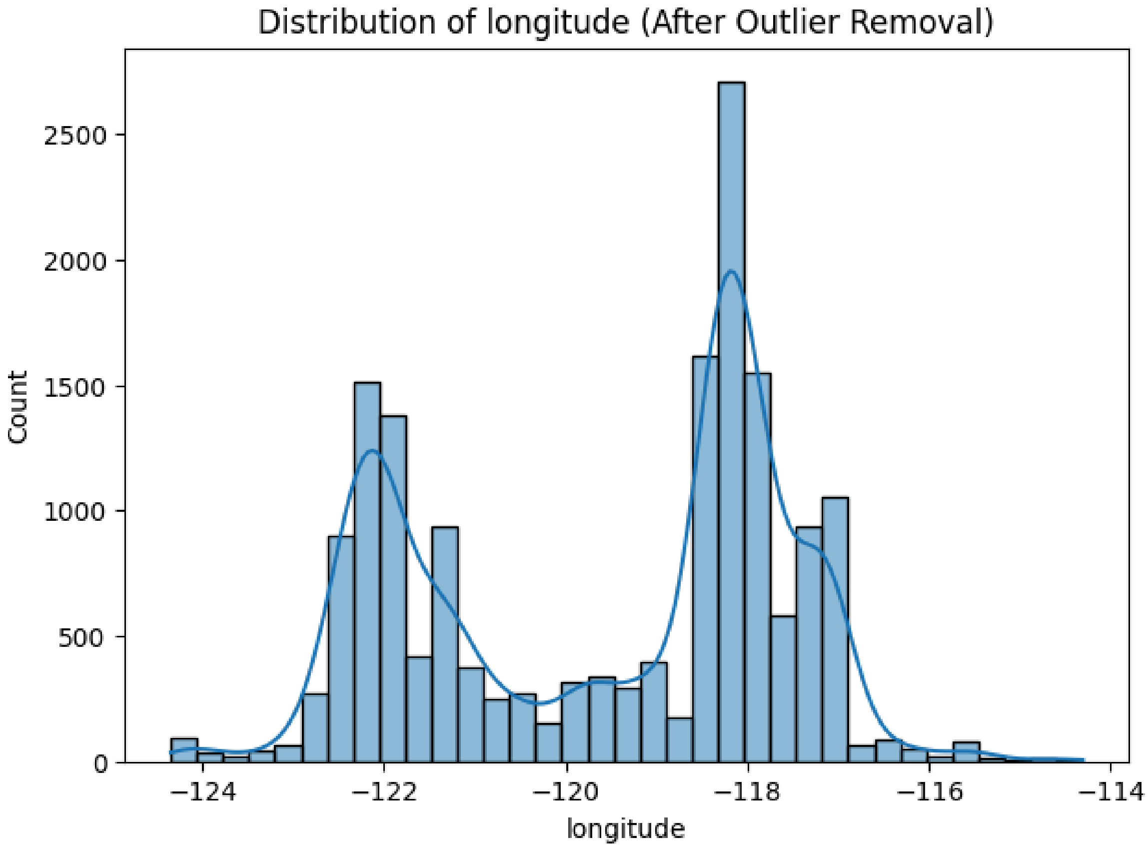
In [49]:
```
# === Resolve Right Skewness ===
fig, axes = plt.subplots(1,3, figsize=(15,4))
sns.histplot(np.log1p(train[TARGET_NAME]), kde=True, ax=axes[0]); axes[0]
sns.histplot(np.sqrt(train[TARGET_NAME]), kde=True, ax=axes[1]); axes[1].
sns.histplot(1/(train[TARGET_NAME]+1e-6), kde=True, ax=axes[2]); axes[2].
plt.show()
```



In [50]:
```
# === Predictor Distribution + Outlier Removal ===
col = train.columns[0]  # just pick first feature for demo, adjust as nee

plt.figure(figsize=(7,5))
```

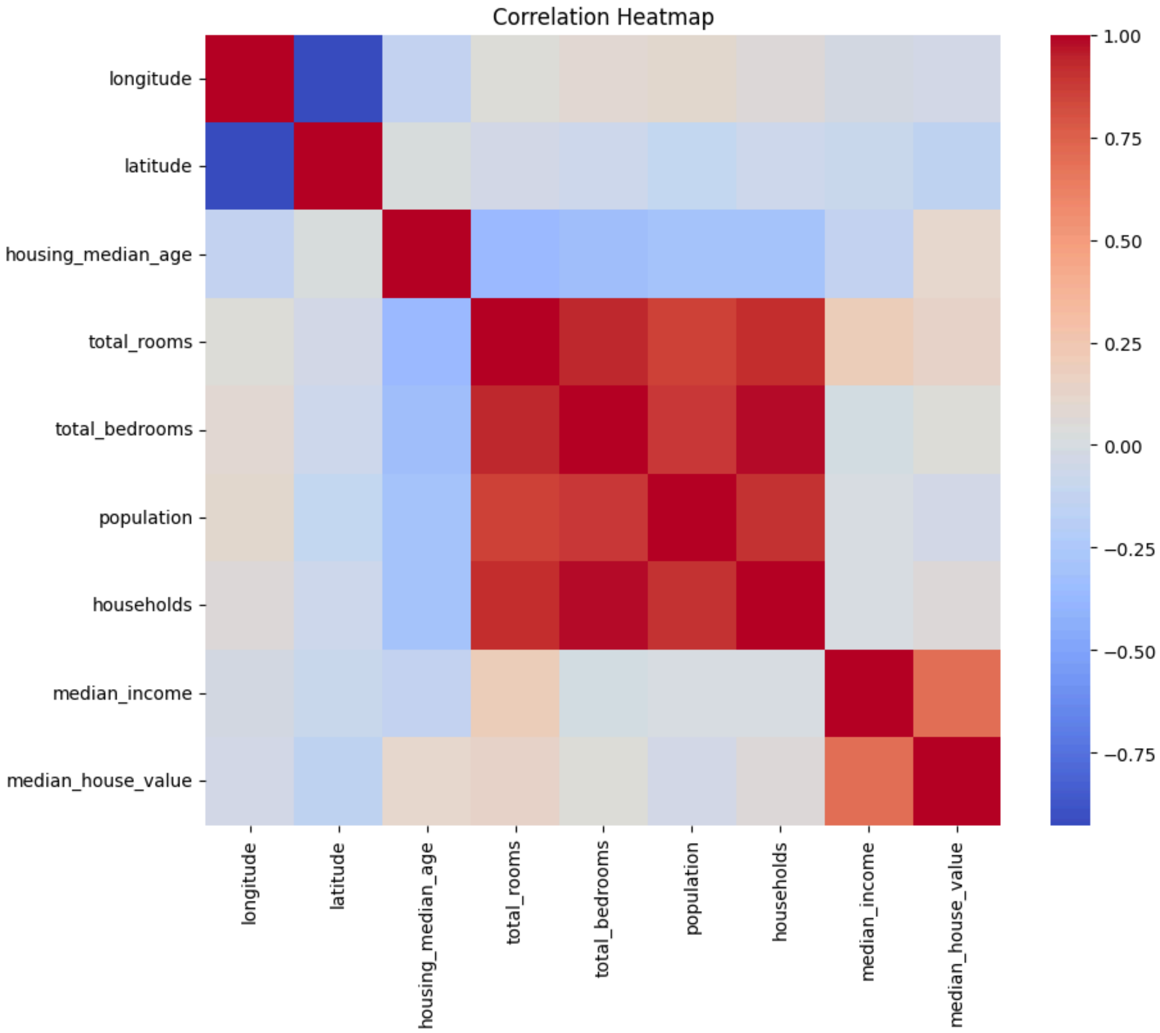
```
sns.histplot(train[col], kde=True)
plt.title(f"Distribution of {col} (Before Outlier Removal)")
plt.show()

# IQR method
Q1, Q3 = train[col].quantile([0.25,0.75])
IQR = Q3 - Q1
low, high = Q1 - 1.5*IQR, Q3 + 1.5*IQR
train = train[(train[col] >= low) & (train[col] <= high)]

plt.figure(figsize=(7,5))
sns.histplot(train[col], kde=True)
plt.title(f"Distribution of {col} (After Outlier Removal)")
plt.show()
```





In [51]:

```python
# === Correlation Heatmap ===
plt.figure(figsize=(10,8))
sns.heatmap(train.corr(), cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```



In [52]:
```
#B) IMPLEMENT NORMAL EQUATION:
X = X_train_scaled.values
y = y_train.values.reshape(-1, 1)

theta = np.linalg.pinv(X) @ y
y_pred = X_test_scaled.values @ theta
```

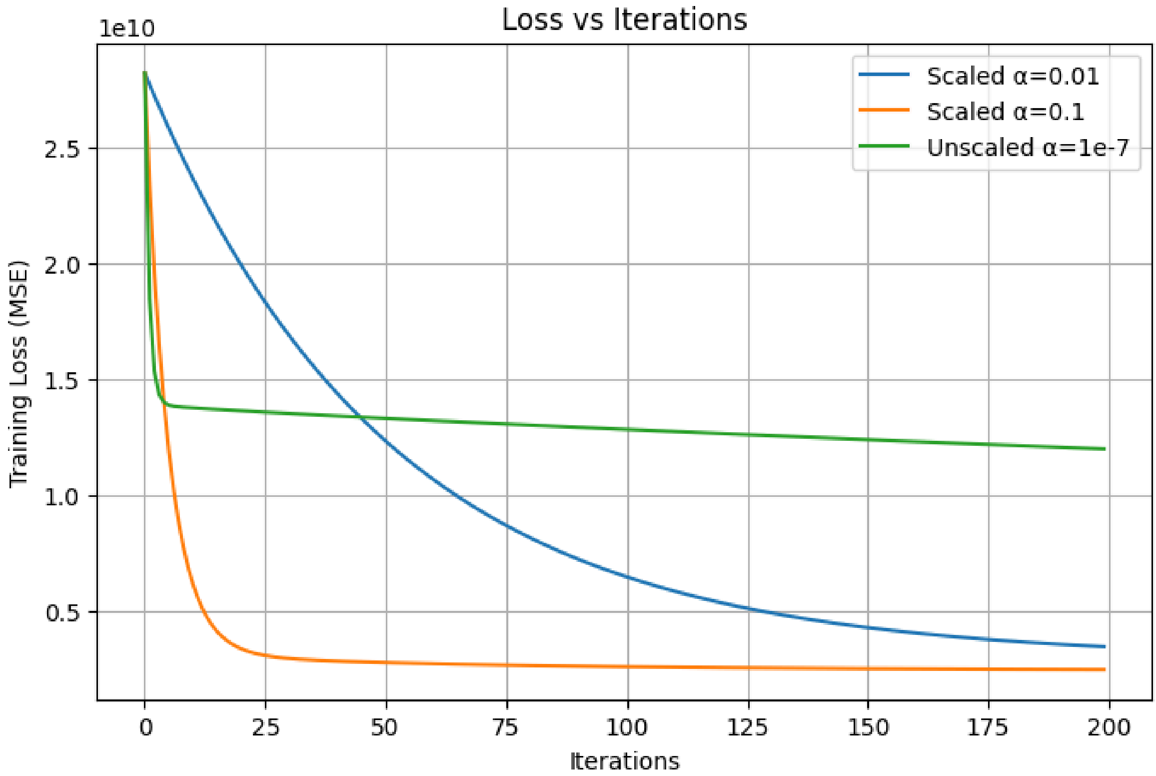
In [53]:
```
#C) Implement Batch Gradient Descent (iterative):


def batch_gradient_descent(X, y, alpha=0.01, num_iters=1000, tol=None):
    n, d = X.shape
    theta = np.zeros((d, 1))
    losses = []

    for i in range(num_iters):
        y_pred = X @ theta
        error = y_pred - y
        loss = (1/(2*n)) * np.sum(error**2)
        losses.append(loss)

        grad = (1/n) * (X.T @ error)
        theta -= alpha * grad

        if tol and i > 0 and abs(losses[-2] - losses[-1]) < tol:
            break

    return theta, losses
```

```python
X = X_train_scaled.values
y = y_train.values.reshape(-1, 1)

theta_gd, losses = batch_gradient_descent(X, y, alpha=0.01, num_iters=100
```

In [54]:
```python
#D) Comparisons with scikit-learn

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

y_pred_gd = X_test_scaled.values @ theta_gd
mse_gd = mean_squared_error(y_test, y_pred_gd)
r2_gd = r2_score(y_test, y_pred_gd)

lr = LinearRegression(fit_intercept=False)
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)

print("Gradient Descent -> MSE:", mse_gd, "R²:", r2_gd)
print("Sklearn LinearRegression -> MSE:", mse_lr, "R²:", r2_lr)
```

```
Gradient Descent -> MSE: 5149828415.410274 R²: 0.5974116946603373
Sklearn LinearRegression -> MSE: 4867205486.9288645 R²: 0.6195057678312001
```

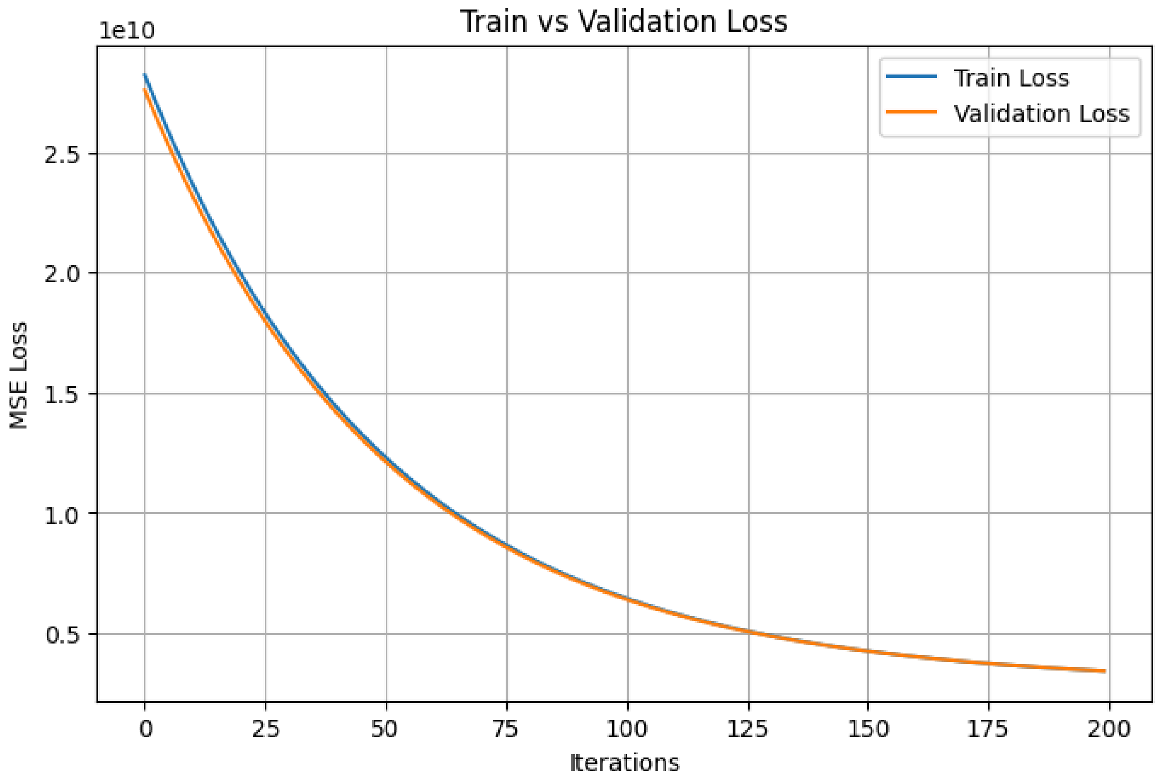
In [55]:
```python
#E) Visualizations

#1) Loss vs Iterations (different learning rates + scaling effect)
import matplotlib.pyplot as plt

def run_gd(alpha, X, y, label):
    _, losses = batch_gradient_descent(X, y, alpha=alpha, num_iters=200)
    plt.plot(losses, label=label)

plt.figure(figsize=(8,5))

# scaled
run_gd(0.01, X_train_scaled.values, y_train.values.reshape(-1,1), "Scaled
run_gd(0.1, X_train_scaled.values, y_train.values.reshape(-1,1), "Scaled

# unscaled
run_gd(0.0000001, X_train.values, y_train.values.reshape(-1,1), "Unscaled

plt.xlabel("Iterations")
plt.ylabel("Training Loss (MSE)")
plt.title("Loss vs Iterations")
plt.legend()
plt.grid(True)
plt.savefig("loss_vs_iterations.png", dpi=300)
plt.show()
```



In [56]:

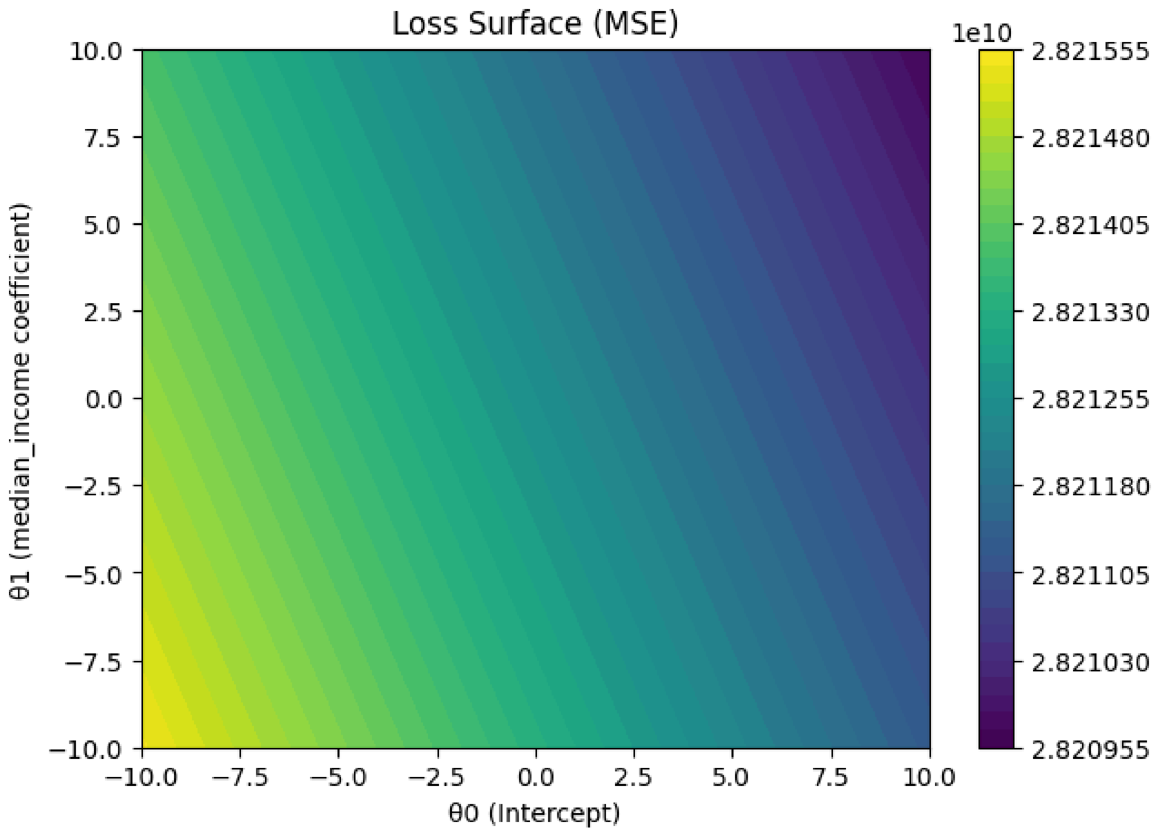
```
#2) Validation Loss vs Iterations

def batch_gd_with_val(X_train, y_train, X_val, y_val, alpha=0.01, num_ite
    n, d = X_train.shape
    theta = np.zeros((d, 1))
    train_losses, val_losses = [], []

    for i in range(num_iters):
        y_pred = X_train @ theta
        error = y_pred - y_train
        train_loss = (1/(2*n)) * np.sum(error**2)
        train_losses.append(train_loss)

        val_pred = X_val @ theta
        val_loss = (1/(2*len(y_val))) * np.sum((val_pred - y_val)**2)
        val_losses.append(val_loss)

        grad = (1/n) * (X_train.T @ error)
        theta -= alpha * grad

    return theta, train_losses, val_losses

theta_val, train_losses, val_losses = batch_gd_with_val(
    X_train_scaled.values, y_train.values.reshape(-1,1),
    X_test_scaled.values, y_test.values.reshape(-1,1),
    alpha=0.01, num_iters=200
)

plt.figure(figsize=(8,5))
plt.plot(train_losses, label="Train Loss")
plt.plot(val_losses, label="Validation Loss")
plt.xlabel("Iterations")
plt.ylabel("MSE Loss")
plt.title("Train vs Validation Loss")
plt.legend()
```

```
plt.grid(True)
plt.savefig("train_val_loss.png", dpi=300)
plt.show()
```



In [57]:

```
#3) Gradient/Loss Surface Visualization

# Take only 1 feature + intercept for visualization
X_vis = X_train_scaled[['intercept', 'median_income']].values
y_vis = y_train.values.reshape(-1,1)

theta0_vals = np.linspace(-10, 10, 100)
theta1_vals = np.linspace(-10, 10, 100)
J_vals = np.zeros((len(theta0_vals), len(theta1_vals)))

for i, t0 in enumerate(theta0_vals):
    for j, t1 in enumerate(theta1_vals):
        theta_try = np.array([[t0],[t1]])
        errors = X_vis @ theta_try - y_vis
        J_vals[i,j] = (1/(2*len(y_vis))) * np.sum(errors**2)

T0, T1 = np.meshgrid(theta0_vals, theta1_vals)

plt.figure(figsize=(7,5))
cp = plt.contourf(T0, T1, J_vals.T, 50, cmap='viridis')
plt.colorbar(cp)
plt.xlabel("θ0 (Intercept)")
plt.ylabel("θ1 (median_income coefficient)")
plt.title("Loss Surface (MSE)")
plt.savefig("loss_surface.png", dpi=300)
plt.show()
```

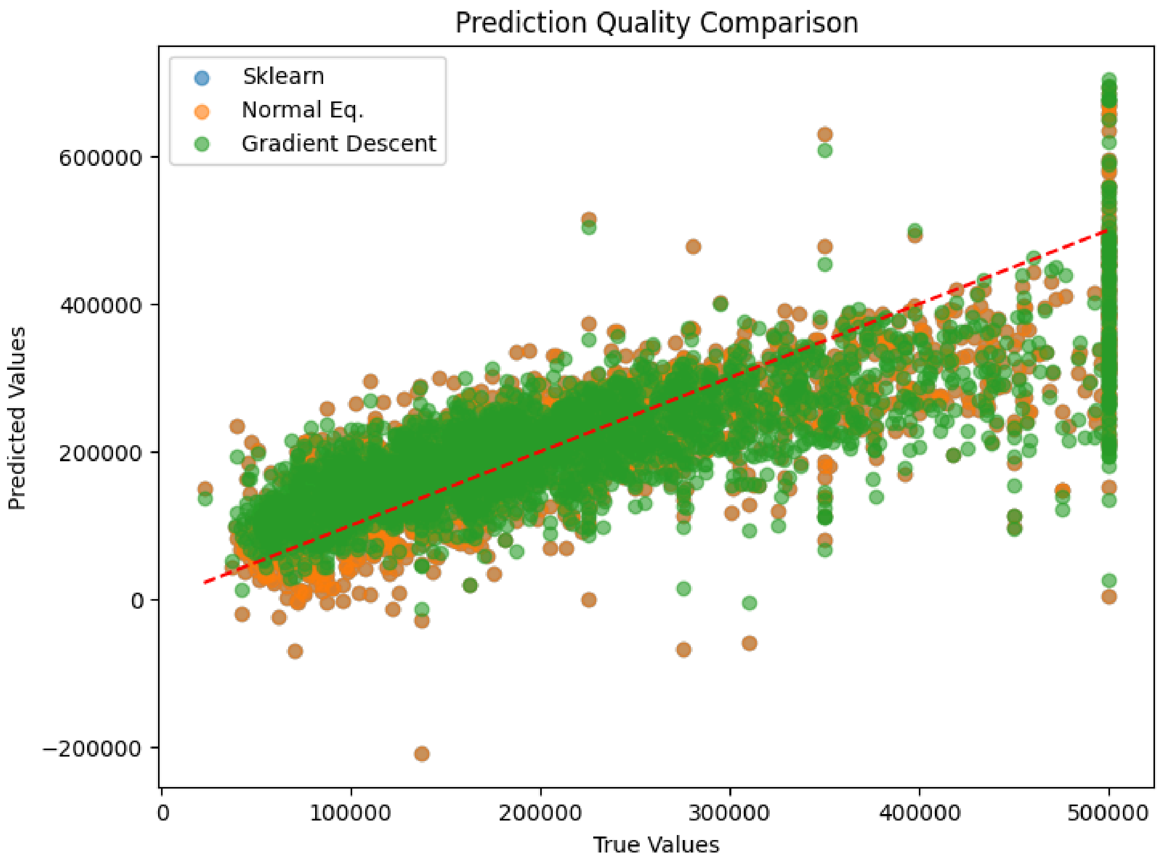


In [58]:
```
#F) Evaluation metrics

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_e
import numpy as np
import pandas as pd

def evaluate_model(theta, X_test, y_test):
    y_pred = X_test @ theta
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    return mse, rmse, r2, mae

# Prepare arrays
X_train_arr = X_train_scaled.values
y_train_arr = y_train.values.reshape(-1,1)
X_test_arr = X_test_scaled.values
y_test_arr = y_test.values.reshape(-1,1)

# --- Normal Equation ---
theta_ne = np.linalg.pinv(X_train_arr) @ y_train_arr
mse_ne, rmse_ne, r2_ne, mae_ne = evaluate_model(theta_ne, X_test_arr, y_t

# --- Gradient Descent ---
theta_gd, _ = batch_gradient_descent(X_train_arr, y_train_arr, alpha=0.01
mse_gd, rmse_gd, r2_gd, mae_gd = evaluate_model(theta_gd, X_test_arr, y_t

# --- Sklearn ---
from sklearn.linear_model import LinearRegression
lr = LinearRegression(fit_intercept=False)
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)
mse_lr = mean_squared_error(y_test, y_pred_lr)
```

```
rmse_lr = np.sqrt(mse_lr)
r2_lr = r2_score(y_test, y_pred_lr)
mae_lr = mean_absolute_error(y_test, y_pred_lr)

# Collect results in a DataFrame
results = pd.DataFrame({
    "MSE": [mse_ne, mse_gd, mse_lr],
    "RMSE": [rmse_ne, rmse_gd, rmse_lr],
    "R2": [r2_ne, r2_gd, r2_lr],
    "MAE": [mae_ne, mae_gd, mae_lr]
}, index=["Normal Equation", "Gradient Descent", "Sklearn LinearRegressio

print(results)
```

```
                                 MSE          RMSE        R2            M
AE
Normal Equation          4.867205e+09  69765.360222  0.619506  50352.2282
58
Gradient Descent         5.499150e+09  74156.254531  0.570103  53891.3125
22
Sklearn LinearRegression  4.867205e+09  69765.360222  0.619506  50352.2282
58
```

In [59]:
```
import matplotlib.pyplot as plt

# Predictions from each model
y_pred_normal = X_test_arr @ theta_ne
y_pred_gd = X_test_arr @ theta_gd
y_pred_sklearn = y_pred_lr  # already computed

plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred_sklearn, label="Sklearn", alpha=0.6)
plt.scatter(y_test, y_pred_normal, label="Normal Eq.", alpha=0.6)
plt.scatter(y_test, y_pred_gd, label="Gradient Descent", alpha=0.6)

# Ideal line
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--

plt.xlabel("True Values")
plt.ylabel("Predicted Values")
plt.title("Prediction Quality Comparison")
plt.legend()
plt.show()
```



In [60]:
```
# === CONFIG & DATA LOADING (robust) ===
import os, pandas as pd, numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression

# You can override these if needed:
TRAIN_PATH = "california_housing_train.csv"
TEST_PATH  = "california_housing_test.csv"
# If None, will auto-detect using common names or assume last column is t
TARGET_NAME = None

# Load
train = pd.read_csv(TRAIN_PATH)
test  = pd.read_csv(TEST_PATH)

# Auto-detect target
if TARGET_NAME is None:
    common_targets = ["target","Target","y","label","price","Price","medi
    found = [c for c in train.columns if c in common_targets]
    if found:
        TARGET_NAME = found[0]
    else:
        TARGET_NAME = train.columns[-1]  # fall back to last column

# Split X/y
y_train = train[TARGET_NAME].values.reshape(-1,1)
X_train = train.drop(columns=[TARGET_NAME]).values
y_test  = test[TARGET_NAME].values.reshape(-1,1) if TARGET_NAME in test.c
X_test  = test.drop(columns=[TARGET_NAME]).values if TARGET_NAME in test.

print("Detected target:", TARGET_NAME)
print("Shapes -> X_train:", X_train.shape, "y_train:", y_train.shape, "X_
```

```
Detected target: median_house_value
Shapes -> X_train: (17000, 8) y_train: (17000, 1) X_test: (3000, 8) y_tes
t: (3000, 1)
```

In [61]:

```
# === SCALING & BIAS COLUMN ===
scaler = StandardScaler()
X_train_scaled_core = scaler.fit_transform(X_train)
X_test_scaled_core  = scaler.transform(X_test)

# Add bias column
X_train_scaled = np.hstack([np.ones((X_train_scaled_core.shape[0],1)), X_
X_test_scaled  = np.hstack([np.ones((X_test_scaled_core.shape[0],1)),  X_

# Unscaled with bias (to show scaling effect)
X_train_unscaled = np.hstack([np.ones((X_train.shape[0],1)), X_train])
X_test_unscaled  = np.hstack([np.ones((X_test.shape[0],1)),  X_test])

print("Processed Train Features Shape (scaled):", X_train_scaled.shape)
print("Processed Test Features Shape  (scaled):", X_test_scaled.shape)
```

```
Processed Train Features Shape (scaled): (17000, 9)
Processed Test Features Shape  (scaled): (3000, 9)
```

In [62]:

```
# === NORMAL EQUATION ===
def normal_eq_theta(X, y):
    # Use pseudo-inverse for numerical stability
    return np.linalg.pinv(X) @ y

theta_normal = normal_eq_theta(X_train_scaled, y_train)
y_pred_normal = X_test_scaled @ theta_normal
```

In [63]:

```
# === BATCH GRADIENT DESCENT (train loss) ===
def mse_loss(X, y, theta):
    n = X.shape[0]
    err = X @ theta - y
    return (1/(2*n)) * np.sum(err**2)

def batch_gradient_descent(X, y, alpha=0.01, num_iters=1000, tol=1e-8):
    n, d = X.shape
    theta = np.zeros((d,1))
    losses = []
    prev = None
    for i in range(num_iters):
        err = X @ theta - y
        grad = (1/n) * (X.T @ err)
        theta = theta - alpha * grad
        loss = (1/(2*n)) * np.sum(err**2)
        losses.append(loss)
        if prev is not None and abs(prev - loss) < tol:
            break
        prev = loss
    return theta, np.array(losses)

theta_gd_scaled, train_losses_scaled = batch_gradient_descent(X_train_sca
y_pred_gd_scaled = X_test_scaled @ theta_gd_scaled
```

In [64]:

```
# === GD WITH VALIDATION (track train & validation loss) ===
def batch_gd_with_val(X_train, y_train, X_val, y_val, alpha=0.05, num_ite
    n, d = X_train.shape
    theta = np.zeros((d,1))
```

```
    train_losses, val_losses = [], []
    prev = None
    for i in range(num_iters):
        err_train = X_train @ theta - y_train
        grad = (1/n) * (X_train.T @ err_train)
        theta = theta - alpha * grad
        # record
        tl = (1/(2*n)) * np.sum(err_train**2)
        train_losses.append(tl)
        vl = (1/(2*X_val.shape[0])) * np.sum((X_val @ theta - y_val)**2)
        val_losses.append(vl)
        if prev is not None and abs(prev - tl) < tol:
            break
        prev = tl
    return theta, np.array(train_losses), np.array(val_losses)

theta_gd_val, train_losses_val, val_losses = batch_gd_with_val(X_train_sc
```

In [65]:

```
# === SKLEARN LINEAR REGRESSION ===
lr = LinearRegression(fit_intercept=False)  # bias already added
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)
```

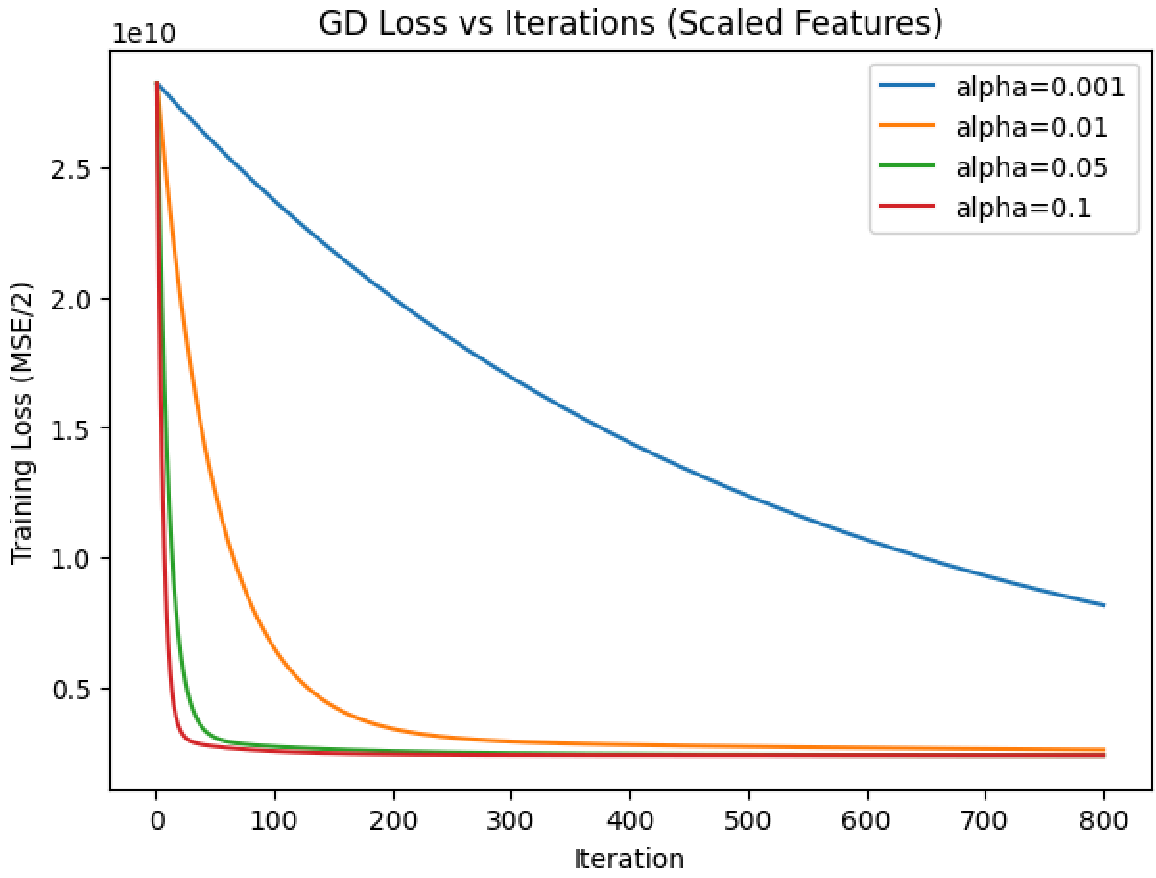
In [66]:

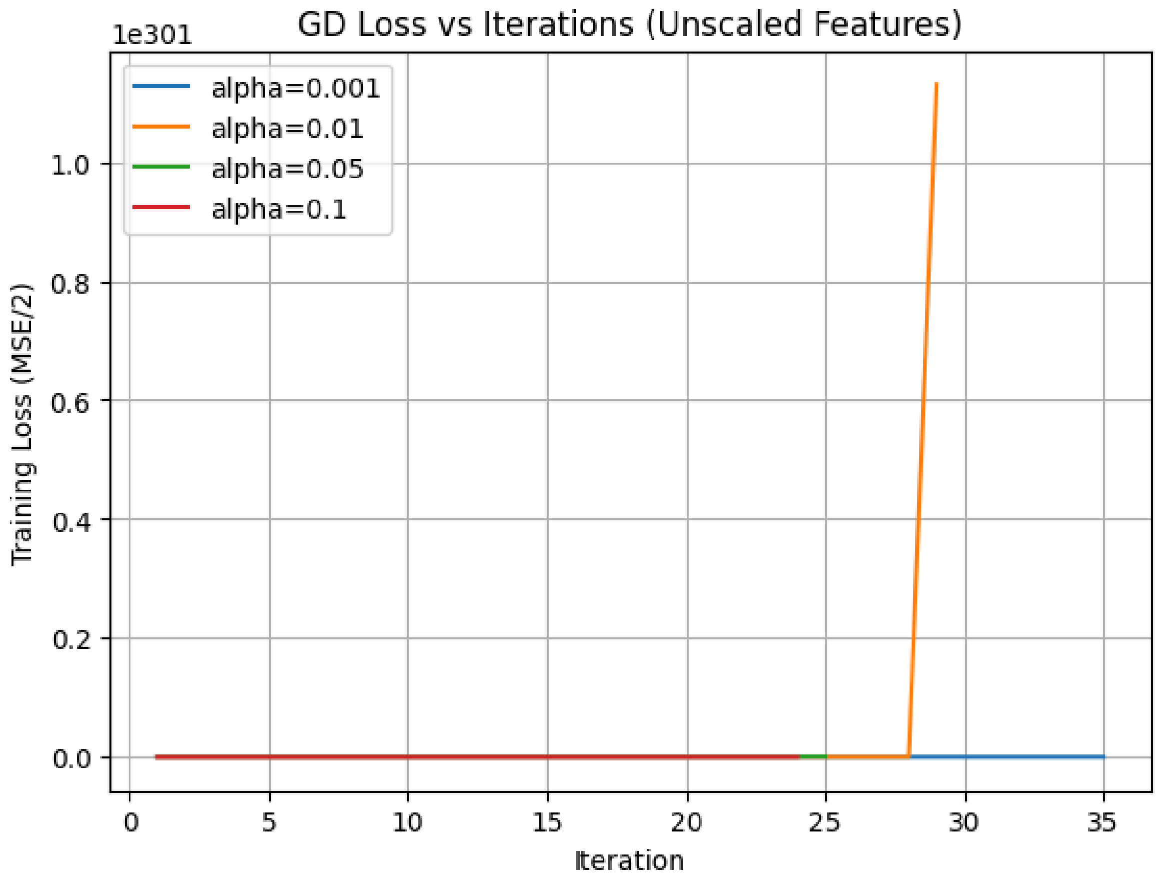
```
# === METRICS (MSE, RMSE, R2, MAE) ===
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_e

def evaluate_preds(y_true, y_hat):
    mse = mean_squared_error(y_true, y_hat)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true, y_hat)
    mae = mean_absolute_error(y_true, y_hat)
    return mse, rmse, r2, mae

# y_test might be None if test lacks target; handle by using train as pro
y_true_eval = y_test if y_test is not None else y_train

metrics = []
labels = ["Normal Equation","Gradient Descent","Sklearn LinearRegression"
preds   = [y_pred_normal, y_pred_gd_scaled, y_pred_lr]

for lab, yp in zip(labels, preds):
    mse, rmse, r2, mae = evaluate_preds(y_true_eval, yp)
    metrics.append({"Model": lab, "MSE": mse, "RMSE": rmse, "R2": r2, "MA

metrics_df = pd.DataFrame(metrics)
print(metrics_df)
metrics_path = "metrics_comparison.csv"
metrics_df.to_csv(metrics_path, index=False)
print("Saved metrics to:", metrics_path)
```

```
                       Model           MSE          RMSE        R2  \
0            Normal Equation  4.867205e+09  69765.360222  0.619506
1           Gradient Descent  4.866230e+09  69758.366060  0.619582
2  Sklearn LinearRegression  4.867205e+09  69765.360222  0.619506

            MAE
0  50352.228258
1  50341.803968
2  50352.228258
Saved metrics to: metrics_comparison.csv
```

In [70]:
```python
# === PLOTS: LOSS VS ITERATIONS (alphas + scaling effect) ===
import matplotlib.pyplot as plt

def collect_losses_for_alphas(X, y, alphas, num_iters=800):
    curves = {}
    for a in alphas:
        _, losses = batch_gradient_descent(X, y, alpha=a, num_iters=num_i
        curves[a] = losses
    return curves

alphas = [0.001, 0.01, 0.05, 0.1]
curves_scaled = collect_losses_for_alphas(X_train_scaled, y_train, alphas
curves_unscaled = collect_losses_for_alphas(X_train_unscaled, y_train, al

# Plot scaled
plt.figure(figsize=(7,5))
for a, losses in curves_scaled.items():
    plt.plot(range(1, len(losses)+1), losses, label=f"alpha={a}")
plt.xlabel("Iteration")
plt.ylabel("Training Loss (MSE/2)")
plt.title("GD Loss vs Iterations (Scaled Features)")
plt.legend()

plt.savefig("loss_vs_iterations_scaled.png", dpi=300, bbox_inches="tight"
plt.show()

# Plot unscaled
plt.figure(figsize=(7,5))
for a, losses in curves_unscaled.items():
    plt.plot(range(1, len(losses)+1), losses, label=f"alpha={a}")
plt.xlabel("Iteration")
plt.ylabel("Training Loss (MSE/2)")
plt.title("GD Loss vs Iterations (Unscaled Features)")
plt.legend()
plt.grid(True)
plt.savefig("loss_vs_iterations_unscaled.png", dpi=300, bbox_inches="tigh
plt.show()
```

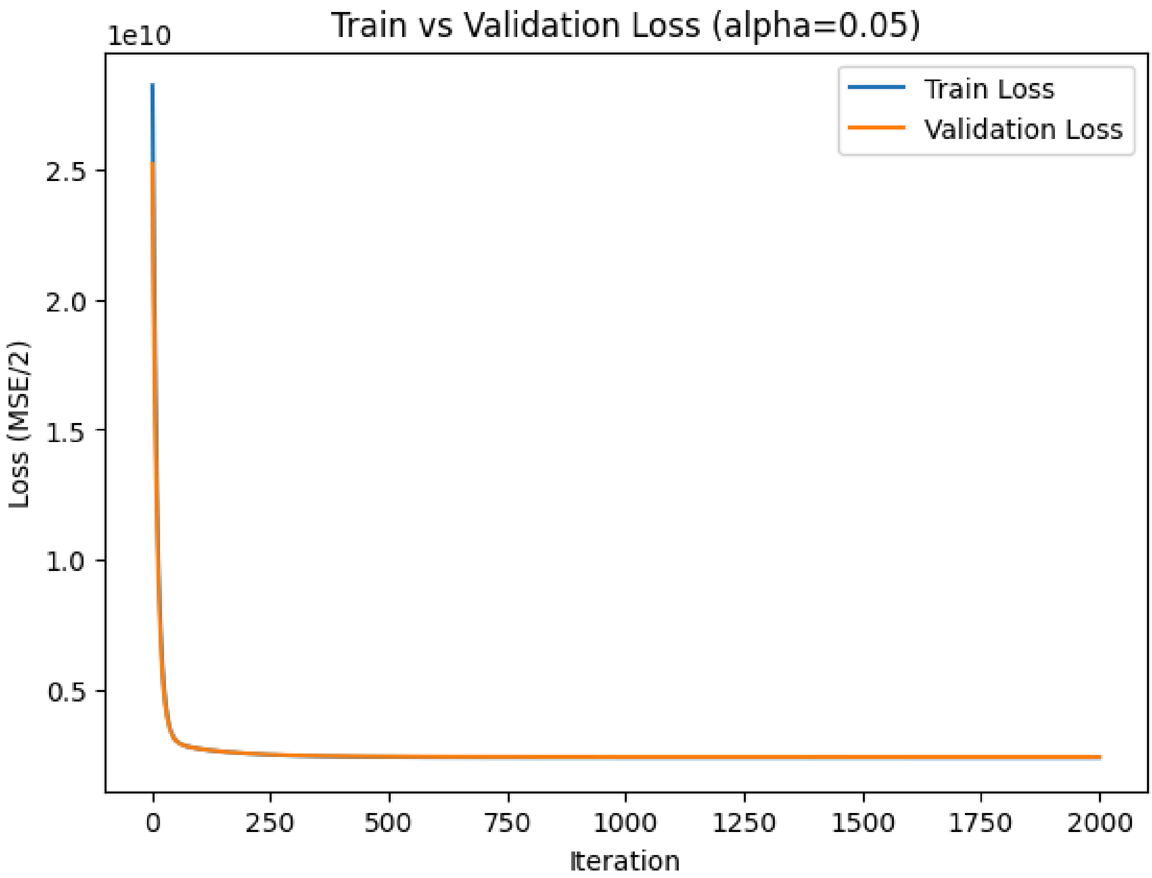
```
/var/folders/2j/6bvnywcd4k76ggrbfxbv4ckc0000gn/T/ipykernel_20103/25345853
8.py:16: RuntimeWarning: overflow encountered in square
  loss = (1/(2*n)) * np.sum(err**2)
/var/folders/2j/6bvnywcd4k76ggrbfxbv4ckc0000gn/T/ipykernel_20103/25345853
8.py:18: RuntimeWarning: invalid value encountered in scalar subtract
  if prev is not None and abs(prev - loss) < tol:
/var/folders/2j/6bvnywcd4k76ggrbfxbv4ckc0000gn/T/ipykernel_20103/25345853
8.py:15: RuntimeWarning: invalid value encountered in subtract
  theta = theta - alpha * grad
```





In [71]:
```
# === PLOT: TRAIN vs VALIDATION LOSS OVER ITERATIONS ===
plt.figure(figsize=(7,5))
plt.plot(range(1, len(train_losses_val)+1), train_losses_val, label="Trai
plt.plot(range(1, len(val_losses)+1), val_losses, label="Validation Loss"
plt.xlabel("Iteration")
plt.ylabel("Loss (MSE/2)")
```

```
plt.title("Train vs Validation Loss (alpha=0.05)")
plt.legend()

plt.savefig("train_vs_validation_loss.png", dpi=300, bbox_inches="tight")
plt.show()
```



In [69]:
```
# === LOSS SURFACE VISUALIZATION (single best feature) ===
# Pick the feature with highest absolute correlation to y (on train)
X_core = X_train_scaled[:,1:]  # drop bias
yvec = y_train.flatten()
if X_core.shape[1] >= 1:
    # compute correlations
    cors = [abs(np.corrcoef(X_core[:,j], yvec)[0,1]) for j in range(X_cor
    jbest = int(np.nanargmax(cors))
    x1 = X_core[:, jbest][:,None]
    X_vis = np.hstack([np.ones((x1.shape[0],1)), x1])

    # Grid for theta0 (bias) and theta1
    t0 = np.linspace(-2, 2, 80)
    t1 = np.linspace(-2, 2, 80)
    T0, T1 = np.meshgrid(t0, t1)
    J = np.zeros_like(T0)

    for i in range(T0.shape[0]):
        for k in range(T0.shape[1]):
            th = np.array([[T0[i,k]], [T1[i,k]]])
            err = X_vis @ th - y_train
            J[i,k] = (1/(2*X_vis.shape[0])) * np.sum(err**2)

    # Contour plot
    plt.figure(figsize=(7,5))
    CS = plt.contour(T0, T1, J, levels=30)
    plt.clabel(CS, inline=True, fontsize=8)
```

```
    plt.xlabel(r"$\theta_0$ (bias)")
    plt.ylabel(r"$\theta_1$ (for best-correlated feature)")
    plt.title("Loss Surface Contours (MSE/2)")
    plt.savefig("loss_surface_contour.png", dpi=300, bbox_inches="tight")
    plt.show()
else:
    print("Not enough features to plot a loss surface.")
```