In [226…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [227…
```
df=pd.read_csv("C:\\Users\\PCCV\\Downloads\\California_Houses.csv")
```

In [228…
```
df
```

Out[228…

	Median_House_Value	Median_Income	Median_Age	Tot_Rooms	Tot_Bedrooms	Pop
0	452600.0	8.3252	41	880	129	
1	358500.0	8.3014	21	7099	1106	
2	352100.0	7.2574	52	1467	190	
3	341300.0	5.6431	52	1274	235	
4	342200.0	3.8462	52	1627	280	
…	…	…	…	…	…	
20635	78100.0	1.5603	25	1665	374	
20636	77100.0	2.5568	18	697	150	
20637	92300.0	1.7000	17	2254	485	
20638	84700.0	1.8672	18	1860	409	
20639	89400.0	2.3886	16	2785	616	

20640 rows × 14 columns

In [229…
```
df.columns
```

Out[229…
```
Index(['Median_House_Value', 'Median_Income', 'Median_Age', 'Tot_Rooms',
       'Tot_Bedrooms', 'Population', 'Households', 'Latitude', 'Longitude',
       'Distance_to_coast', 'Distance_to_LA', 'Distance_to_SanDiego',
       'Distance_to_SanJose', 'Distance_to_SanFrancisco'],
      dtype='object')
```

In [230…
```
df.describe()
```

Out[230…

	Median_House_Value	Median_Income	Median_Age	Tot_Rooms	Tot_Bedrooms	
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	2
mean	206855.816909	3.870671	28.639486	2635.763081	537.898014	
std	115395.615874	1.899822	12.585558	2181.615252	421.247906	
min	14999.000000	0.499900	1.000000	2.000000	1.000000	
25%	119600.000000	2.563400	18.000000	1447.750000	295.000000	
50%	179700.000000	3.534800	29.000000	2127.000000	435.000000	
75%	264725.000000	4.743250	37.000000	3148.000000	647.000000	
max	500001.000000	15.000100	52.000000	39320.000000	6445.000000	3

In [231…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Median_House_Value       20640 non-null  float64
 1   Median_Income            20640 non-null  float64
 2   Median_Age               20640 non-null  int64  
 3   Tot_Rooms                20640 non-null  int64  
 4   Tot_Bedrooms             20640 non-null  int64  
 5   Population               20640 non-null  int64  
 6   Households               20640 non-null  int64  
 7   Latitude                 20640 non-null  float64
 8   Longitude                20640 non-null  float64
 9   Distance_to_coast        20640 non-null  float64
 10  Distance_to_LA           20640 non-null  float64
 11  Distance_to_SanDiego     20640 non-null  float64
 12  Distance_to_SanJose      20640 non-null  float64
 13  Distance_to_SanFrancisco  20640 non-null  float64
dtypes: float64(9), int64(5)
memory usage: 2.2 MB
```

In [232…
```
df.isnull().sum()
```

Out[232…
```
Median_House_Value          0
Median_Income               0
Median_Age                  0
Tot_Rooms                   0
Tot_Bedrooms                0
Population                  0
Households                  0
Latitude                    0
Longitude                   0
Distance_to_coast           0
Distance_to_LA              0
Distance_to_SanDiego        0
Distance_to_SanJose         0
Distance_to_SanFrancisco    0
dtype: int64
```

In [233…
```
# Count total duplicate rows
df.duplicated().sum()
```

Out[233…
```
np.int64(0)
```

In [234…
```
# Detect outliers using the IQR method
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# A row is considered an outlier if any feature is outside 1.5 * IQR
outliers = ((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR)))

# Count outliers per column
outlier_counts = outliers.sum().sort_values(ascending=False)
print(outlier_counts)
```

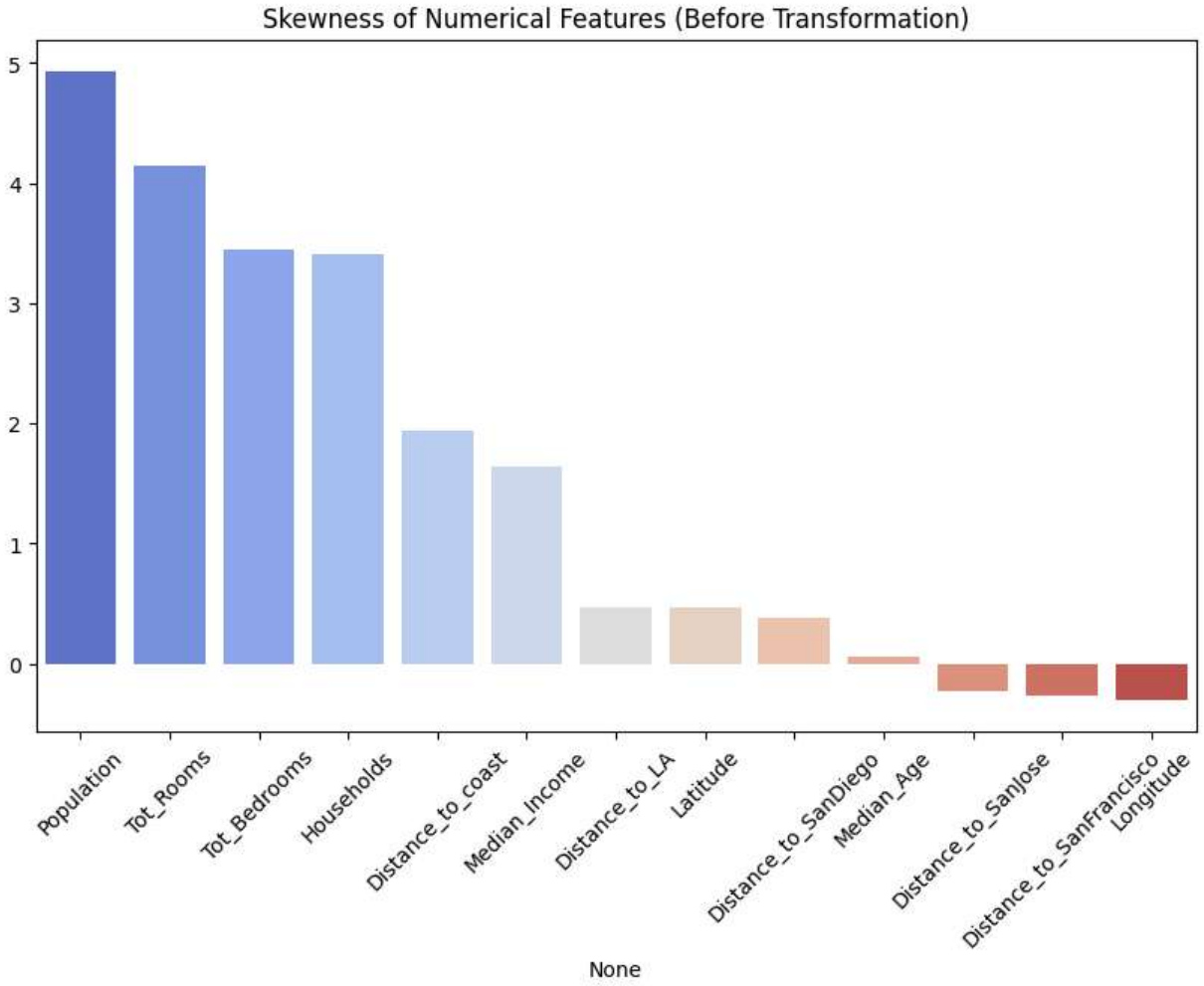
```
Distance_to_coast          2376
Tot_Rooms                  1287
Tot_Bedrooms               1282
Households                 1220
Population                 1196
Median_House_Value         1071
Median_Income               681
Median_Age                    0
Latitude                      0
Longitude                     0
Distance_to_LA                0
Distance_to_SanDiego          0
Distance_to_SanJose           0
Distance_to_SanFrancisco      0
dtype: int64
```

In [235…
```
# ----------------------------------------
# Step 1: Skewness Check and Transform (features only)
# ----------------------------------------
from sklearn.preprocessing import PowerTransformer
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

```
import joblib  # ✅ added to allow saving transformers

# Exclude target
num_cols = df.drop(columns=['Median_House_Value']).select_dtypes(include=np.number)
skew_values = num_cols.skew().sort_values(ascending=False)
print("Skewness before transformation:\n", skew_values)

plt.figure(figsize=(10,6))
sns.barplot(x=skew_values.index, y=skew_values.values, palette="coolwarm")
plt.title('Skewness of Numerical Features (Before Transformation)')
plt.xticks(rotation=45)
plt.show()

# Identify highly skewed columns
skewed_cols = skew_values[abs(skew_values) > 1].index.tolist()
print(f"\nHighly skewed columns (|skew| > 1): {skewed_cols}")

if not skewed_cols:
    print("✅ No highly skewed numeric columns found — skipping transformation.")
else:
    # --- Apply Yeo-Johnson PowerTransformer ---
    pt = PowerTransformer(method='yeo-johnson')
    df[skewed_cols] = pt.fit_transform(df[skewed_cols])
    print(f"✅ Applied Yeo-Johnson transformation to: {skewed_cols}")

    # --- 💾 Save PowerTransformer and skewed column list for later use ---
    joblib.dump(pt, "power_transformer.pkl")
    joblib.dump(skewed_cols, "skewed_cols_list.pkl")
    print("💾 Saved PowerTransformer and skewed column list for later use.")

    # --- Show post-transform skewness ---
    new_skew = df[skewed_cols].skew().sort_values(ascending=False)
    print("\nSkewness after transformation:\n", new_skew)

    plt.figure(figsize=(10,6))
    sns.barplot(x=new_skew.index, y=new_skew.values, palette="viridis")
    plt.title('Skewness of Numerical Features (After Transformation)')
    plt.xticks(rotation=45)
    plt.show()
```

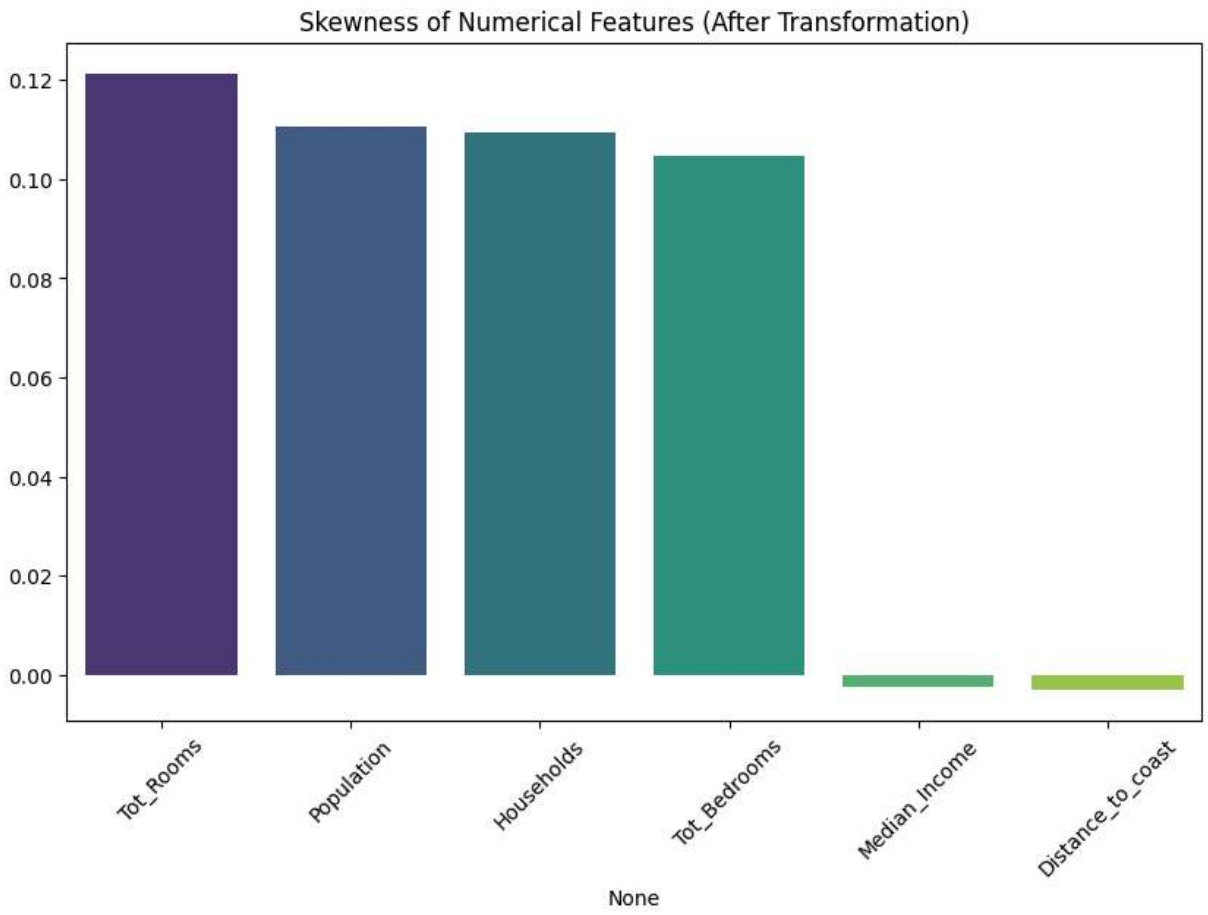
```
Skewness before transformation:
 Population                  4.935858
Tot_Rooms                   4.147343
Tot_Bedrooms                3.453073
Households                  3.410438
Distance_to_coast           1.938709
Median_Income               1.646657
Distance_to_LA              0.466536
Latitude                    0.465953
Distance_to_SanDiego        0.384936
Median_Age                  0.060331
Distance_to_SanJose        -0.223069
Distance_to_SanFrancisco   -0.263499
Longitude                  -0.297801
dtype: float64
```

```
C:\Users\PCCV\AppData\Local\Temp\ipykernel_23832\500157445.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=skew_values.index, y=skew_values.values, palette="coolwarm")
```




```
Highly skewed columns (|skew| > 1): ['Population', 'Tot_Rooms', 'Tot_Bedrooms', 'Hou
seholds', 'Distance_to_coast', 'Median_Income']
✅ Applied Yeo-Johnson transformation to: ['Population', 'Tot_Rooms', 'Tot_Bedroom
s', 'Households', 'Distance_to_coast', 'Median_Income']
💾 Saved PowerTransformer and skewed column list for later use.

Skewness after transformation:
 Tot_Rooms            0.121378
Population           0.110641
Households           0.109520
Tot_Bedrooms         0.104735
Median_Income       -0.002538
Distance_to_coast   -0.002899
dtype: float64
```

```
C:\Users\PCCV\AppData\Local\Temp\ipykernel_23832\500157445.py:43: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=new_skew.index, y=new_skew.values, palette="viridis")
```



In [236…

```
# ------------------------------------------
# Step 2: Split Features and Target (no target transform here)
# ------------------------------------------
from sklearn.model_selection import train_test_split
import joblib
import numpy as np

X = df.drop('Median_House_Value', axis=1)
y = df['Median_House_Value']

# 70% train, 15% val, 15% test
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_sta
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, rand

# reset indices (avoid alignment problems later)
for obj in (X_train, X_val, X_test, y_train, y_val, y_test):
    obj.reset_index(drop=True, inplace=True)

print("Shapes:")
print("X_train:", X_train.shape)
print("X_val:  ", X_val.shape)
print("X_test: ", X_test.shape)
```

```
Shapes:
X_train: (14448, 13)
X_val:   (3096, 13)
X_test:  (3096, 13)
```

In [237…
```
# ----------------------------------------
# Step 3: Scaling and LOG target transform (log1p)
# ----------------------------------------
from sklearn.preprocessing import StandardScaler
import numpy as np
import joblib

# Feature scaling (fit on training features only)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled   = scaler.transform(X_val)
X_test_scaled  = scaler.transform(X_test)

# Target transform: use log1p for positivity and reduced skew
# Keep variable names consistent with your notebook
y_train_transformed = np.log1p(y_train)   # log(1 + y)
y_val_transformed   = np.log1p(y_val)
y_test_transformed  = np.log1p(y_test)

print("✅ Scaling complete and target transformed using log1p.")
```

✅ Scaling complete and target transformed using log1p.

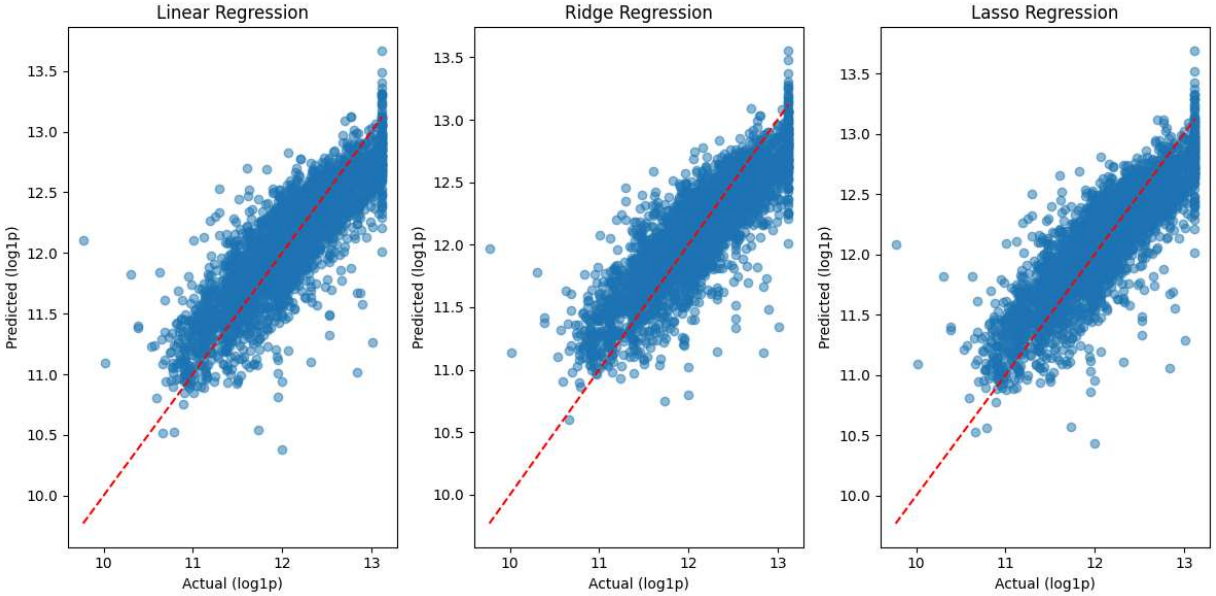
In [238…
```
# ----------------------------
# Step 4 (Fixed): Train Ridge with log1p target transform
# ----------------------------
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Apply log1p transform to target
y_train_log = np.log1p(y_train)
y_val_log = np.log1p(y_val)
y_test_log = np.log1p(y_test)

ridge_model = Ridge(alpha=1000, random_state=42)
ridge_model.fit(X_train_scaled, y_train_log)

# Validation performance
y_val_pred_log = ridge_model.predict(X_val_scaled)
val_rmse_log = np.sqrt(mean_squared_error(y_val_log, y_val_pred_log))
val_r2_log = r2_score(y_val_log, y_val_pred_log)
print(f"RMSE (val, log1p): {val_rmse_log:.4f}, R² (log1p): {val_r2_log:.4f}")

# Back-transform predictions for validation
y_val_pred_orig = np.expm1(y_val_pred_log)
val_rmse_orig = np.sqrt(mean_squared_error(y_val, y_val_pred_orig))
val_r2_orig = r2_score(y_val, y_val_pred_orig)
print(f"RMSE (val, original $): ${val_rmse_orig:,.2f}, R² (orig): {val_r2_orig:.4f}
```

```
RMSE (val, log1p): 0.3294, R² (log1p): 0.6645
RMSE (val, original $): $69,299.95, R² (orig): 0.6314
```

In [239…
```
# ----------------------------------------
# Step 5: Train & Evaluate Linear, Ridge, Lasso (log1p target)
# ----------------------------------------
```

```
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import matplotlib.pyplot as plt
import numpy as np

linear_model = LinearRegression()
ridge_model  = Ridge(alpha=1000, random_state=42)
lasso_model  = Lasso(alpha=0.001, random_state=42, max_iter=10000)

# Fit on log-transformed target
linear_model.fit(X_train_scaled, y_train_transformed)
ridge_model.fit(X_train_scaled, y_train_transformed)
lasso_model.fit(X_train_scaled, y_train_transformed)

def evaluate_model_logspace(model, X, y_true_log):
    y_pred_log = model.predict(X)
    mse = mean_squared_error(y_true_log, y_pred_log)
    mae = mean_absolute_error(y_true_log, y_pred_log)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_true_log, y_pred_log)
    return mse, mae, rmse, r2, y_pred_log

models = {
    'Linear Regression': linear_model,
    'Ridge Regression': ridge_model,
    'Lasso Regression': lasso_model
}

results = []
for name, model in models.items():
    mse, mae, rmse, r2, y_pred_log = evaluate_model_logspace(model, X_test_scaled,
    results.append([name, mse, mae, rmse, r2])
    print(f"{name} → RMSE (log): {rmse:.4f}, R² (log): {r2:.4f}")
    models[name] = (model, y_pred_log)

# Cross-validation on training data (log-space MAE)
for name, (model, _) in models.items():
    cv_mae = -cross_val_score(model, X_train_scaled, y_train_transformed, cv=5,
                              scoring='neg_mean_absolute_error').mean()
    print(f"Cross-Validation MAE ({name}): {cv_mae:.4f}")

# Plot Actual vs Predicted in log-space
plt.figure(figsize=(12,6))
for i, (name, (_, y_pred_log)) in enumerate(models.items(), 1):
    plt.subplot(1, 3, i)
    plt.scatter(y_test_transformed, y_pred_log, alpha=0.5)
    plt.plot([y_test_transformed.min(), y_test_transformed.max()],
             [y_test_transformed.min(), y_test_transformed.max()], 'r--')
    plt.xlabel("Actual (log1p)")
    plt.ylabel("Predicted (log1p)")
    plt.title(name)
plt.tight_layout()
plt.show()
```

```
Linear Regression → RMSE (log): 0.3014, R² (log): 0.7200
Ridge Regression → RMSE (log): 0.3104, R² (log): 0.7031
Lasso Regression → RMSE (log): 0.3018, R² (log): 0.7192
Cross-Validation MAE (Linear Regression): 0.2325
Cross-Validation MAE (Ridge Regression): 0.2453
Cross-Validation MAE (Lasso Regression): 0.2334
```



In [240…

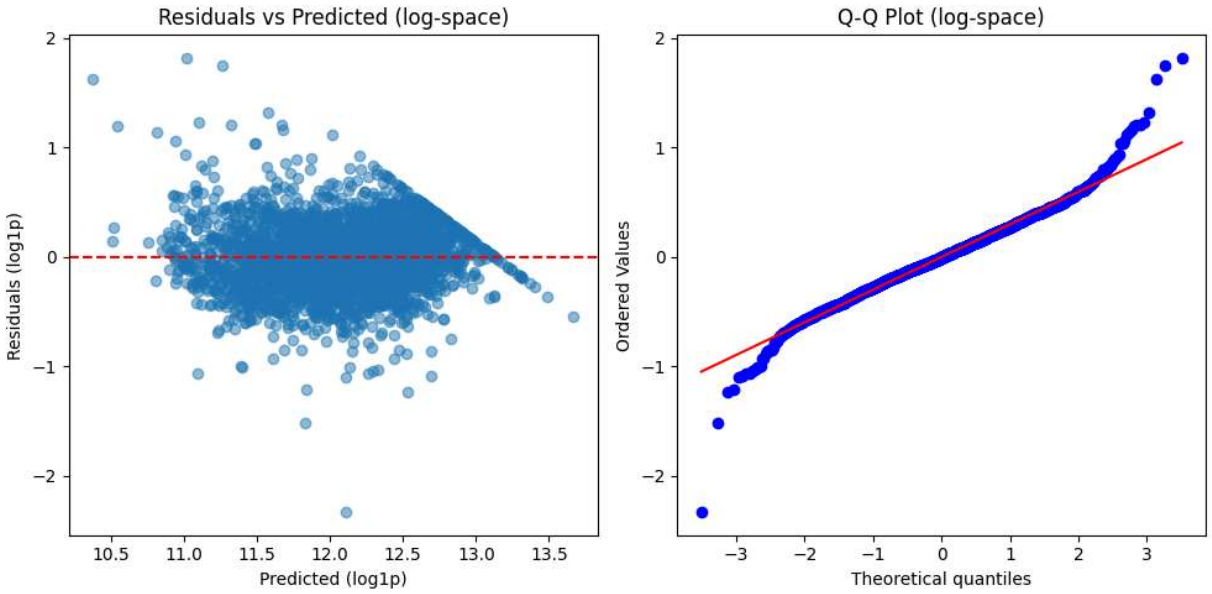
```python
# ----------------------------------------
# Step 6: Residual Diagnostics & Summary (log1p space)
# ----------------------------------------
import scipy.stats as stats
import statsmodels.api as sm
from statsmodels.stats.diagnostic import het_breuschpagan
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

def adjusted_r2(r2, n, p):
    return 1 - (1 - r2) * (n - 1) / (n - p - 1)

results_df = pd.DataFrame(columns=['Model','MSE','MAE','RMSE','R²','Adjusted R²'])
for name, (model, y_pred_log) in models.items():
    mse = mean_squared_error(y_test_transformed, y_pred_log)
    mae = mean_absolute_error(y_test_transformed, y_pred_log)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test_transformed, y_pred_log)
    adj = adjusted_r2(r2, X_test_scaled.shape[0], X_test_scaled.shape[1])
    results_df.loc[len(results_df)] = [name, mse, mae, rmse, r2, adj]

print("\n📊 Model Comparison (log-space):")
print(results_df)

best_model_name = results_df.loc[results_df['RMSE'].idxmin(), 'Model']
best_model = models[best_model_name][0]
print(f"\n🏆 Best model (log-space): {best_model_name}")

# Residuals (log-space)
```

```
y_pred_best_log = best_model.predict(X_test_scaled)
residuals_log = y_test_transformed - y_pred_best_log

plt.figure(figsize=(10,5))
plt.subplot(1,2,1)
plt.scatter(y_pred_best_log, residuals_log, alpha=0.5)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel("Predicted (log1p)")
plt.ylabel("Residuals (log1p)")
plt.title("Residuals vs Predicted (log-space)")

plt.subplot(1,2,2)
stats.probplot(residuals_log, dist="norm", plot=plt)
plt.title("Q-Q Plot (log-space)")
plt.tight_layout()
plt.show()

# BP test (log-space)
lm, lm_pval, fval, f_pval = het_breuschpagan(residuals_log, sm.add_constant(X_test_
print(f"\nBreusch-Pagan Lagrange Multiplier p-value: {lm_pval:.4f}")

# OPTIONAL: show RMSE in original dollars for best model
y_pred_best_orig = np.expm1(y_pred_best_log)
rmse_orig = np.sqrt(mean_squared_error(y_test, y_pred_best_orig))
print(f"\nBest model RMSE in original $: ${rmse_orig:,.2f}")
```

📊 Model Comparison (log-space):

	Model	MSE	MAE	RMSE	R²	Adjusted R²
0	Linear Regression	0.090831	0.228939	0.301382	0.720016	0.718835
1	Ridge Regression	0.096319	0.239621	0.310353	0.703099	0.701847
2	Lasso Regression	0.091108	0.229683	0.301841	0.719164	0.717979

🏆 Best model (log-space): Linear Regression

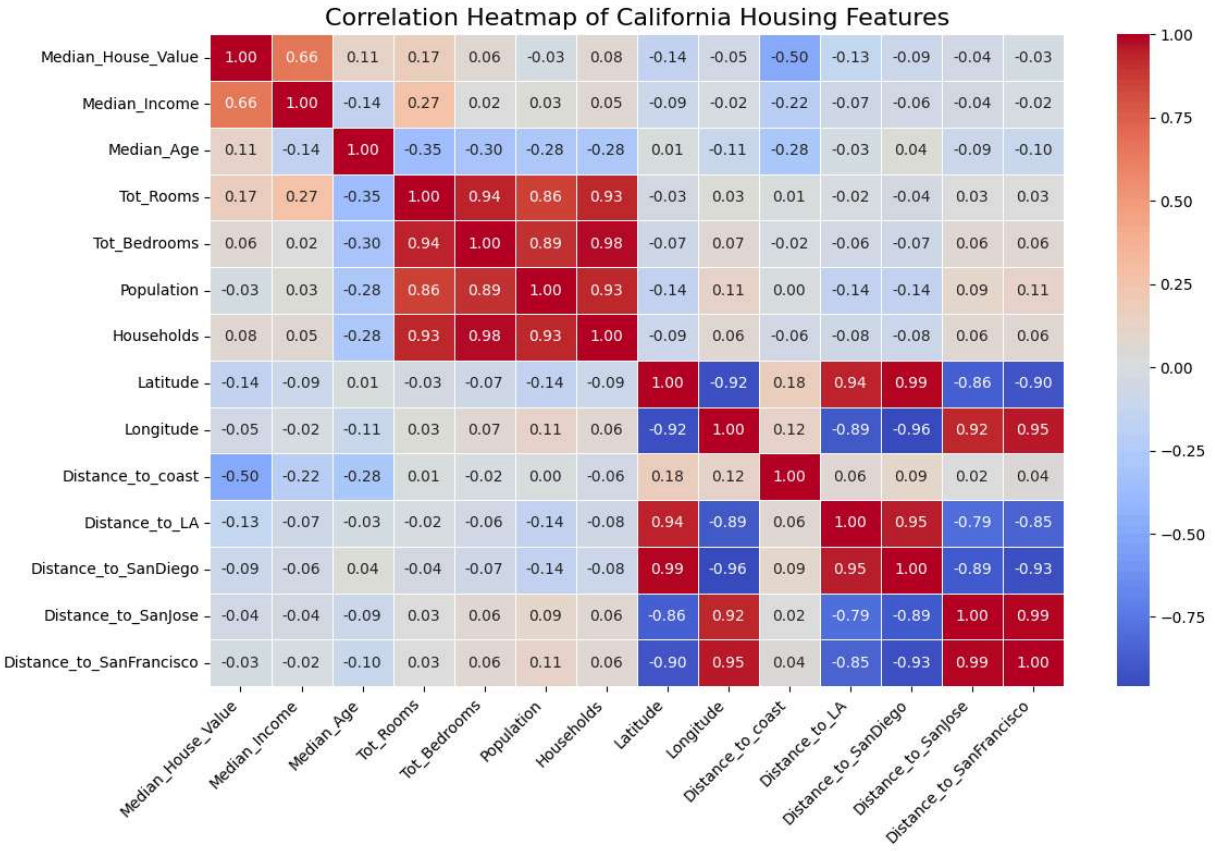


Breusch-Pagan Lagrange Multiplier p-value: 0.0000

Best model RMSE in original $: $63,023.21

In [241…

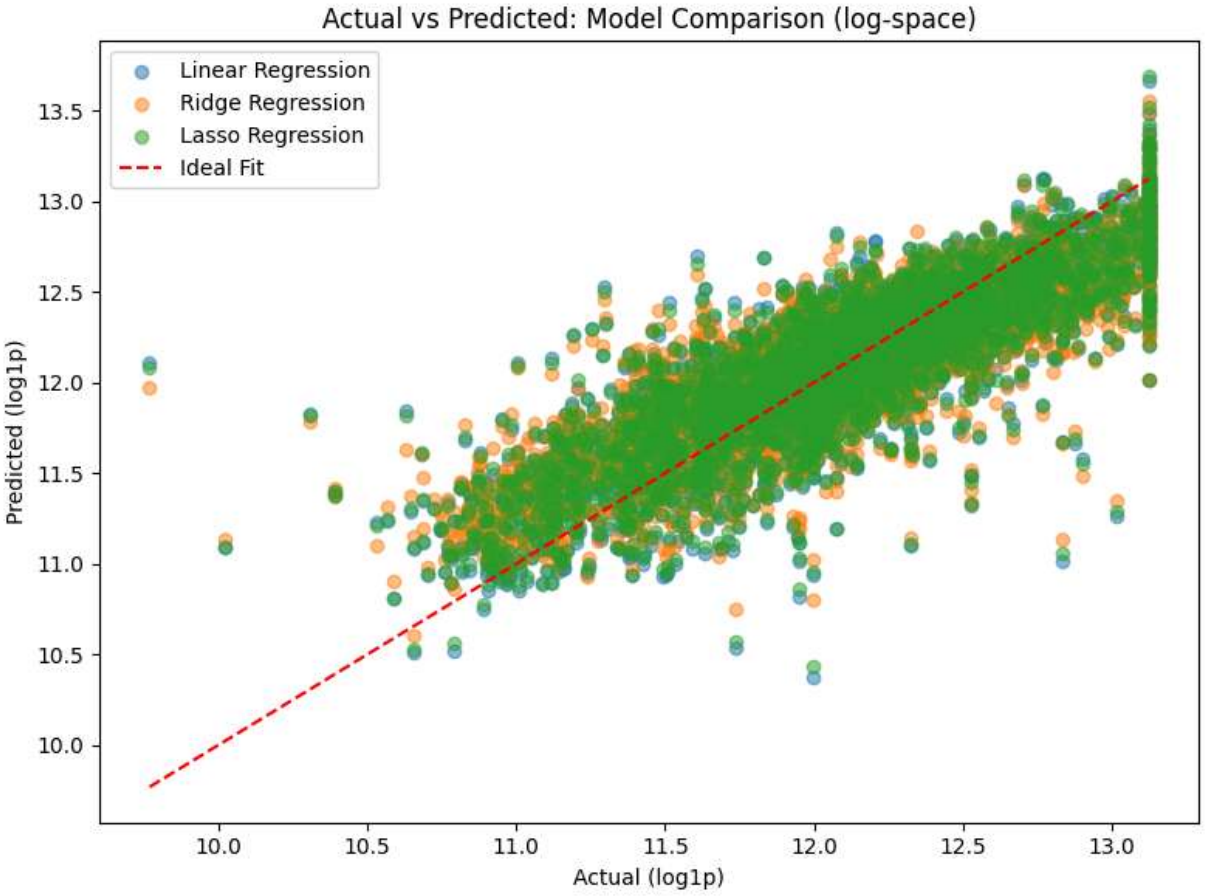
```
# ----------------------------------------
# Step 7: Correlation Heatmap
# ----------------------------------------
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12,8))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt=".2f", lin
plt.title('Correlation Heatmap of California Housing Features', fontsize=16)
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



In [242…

```
# ----------------------------------------
# Step 8: Combined Actual vs Predicted (log-space)
# ----------------------------------------
import matplotlib.pyplot as plt
plt.figure(figsize=(8,6))

y_pred_lr    = models['Linear Regression'][1]
y_pred_ridge = models['Ridge Regression'][1]
y_pred_lasso = models['Lasso Regression'][1]

plt.scatter(y_test_transformed, y_pred_lr, alpha=0.5, label='Linear Regression')
plt.scatter(y_test_transformed, y_pred_ridge, alpha=0.5, label='Ridge Regression')
plt.scatter(y_test_transformed, y_pred_lasso, alpha=0.5, label='Lasso Regression')

plt.plot([y_test_transformed.min(), y_test_transformed.max()],
         [y_test_transformed.min(), y_test_transformed.max()], 'r--', label='Ideal
```

```
plt.xlabel('Actual (log1p)')
plt.ylabel('Predicted (log1p)')
plt.title('Actual vs Predicted: Model Comparison (log-space)')
plt.legend()
plt.tight_layout()
plt.show()
```



In [243…

```
# ------------------------------------------
# Step 9: Save Model, Scaler and Test Prediction (log1p handling)
# ------------------------------------------
import joblib
import numpy as np
import pandas as pd
from sklearn.metrics import mean_squared_error, r2_score

# Evaluate on validation set (original-dollar RMSE shown too)
y_val_pred_log = ridge_model.predict(X_val_scaled)
val_rmse_log = np.sqrt(mean_squared_error(y_val_transformed, y_val_pred_log))
val_r2_log = r2_score(y_val_transformed, y_val_pred_log)
print(f"RMSE (val, log1p): {val_rmse_log:.4f}, R² (log1p): {val_r2_log:.4f}")

# Also show validation performance on original scale
y_val_pred_orig = np.expm1(y_val_pred_log)
val_rmse_orig = np.sqrt(mean_squared_error(y_val, y_val_pred_orig))
val_r2_orig = r2_score(y_val, y_val_pred_orig)
print(f"RMSE (val, original $): ${val_rmse_orig:,.2f}, R² (orig): {val_r2_orig:.4f}
```

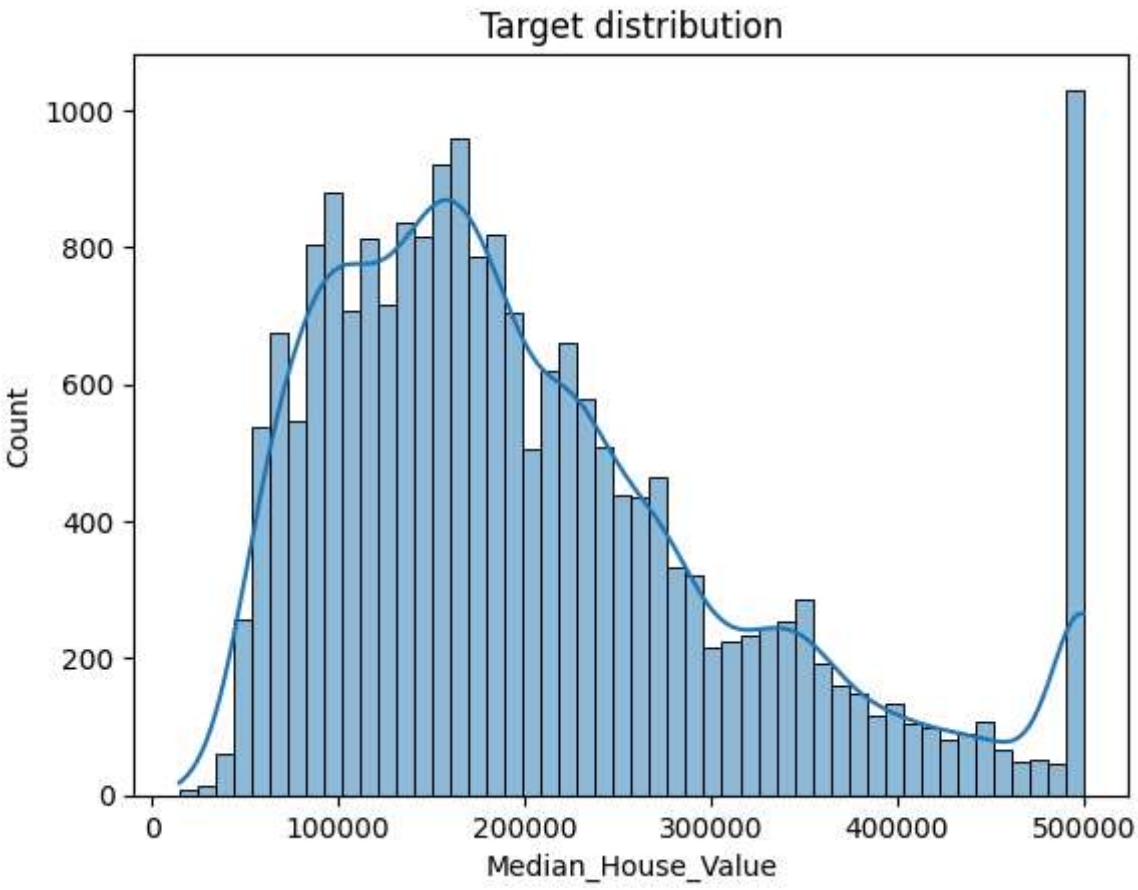
```
# Save artifacts: model + scaler + small metadata indicating log-transform
joblib.dump(ridge_model, "ridge_model_california.pkl")
joblib.dump(scaler, "feature_scaler_california.pkl")
joblib.dump({'target_transform': 'log1p'}, "target_transform_info.pkl")

print("\n💾 Model and Scaler saved successfully! (target transform: log1p)")

# Confirm feature order
model_features = X_train.columns.tolist()
print("\n✅ Feature order confirmed:")
print(model_features)

# --- 🆕 Load PowerTransformer and skewed column list ---
pt = joblib.load("power_transformer.pkl")
skewed_cols = joblib.load("skewed_cols_list.pkl")
print("\n🔄 Loaded PowerTransformer and skewed column list.")

# --- Step 9: Test predictions with realistic input ranges ---
urban_LA = np.array([[5.0, 30.0, 1500.0, 300.0, 800.0, 280.0,
                      34.05, -118.25, 20.0, 0.0, 180.0, 500.0, 550.0]])

coastal_SD = np.array([[6.0, 25.0, 1800.0, 350.0, 1000.0, 350.0,
                        32.72, -117.16, 5.0, 180.0, 0.0, 720.0, 750.0]])

inland_valley = np.array([[4.0, 35.0, 1200.0, 250.0, 600.0, 200.0,
                           36.33, -119.30, 150.0, 300.0, 480.0, 320.0, 450.0]])

samples = {
    "🌆 Urban LA": urban_LA,
    "🌊 Coastal San Diego": coastal_SD,
    "🌾 Inland Valley": inland_valley
}

# --- ✅ FIXED PREDICTION LOOP ---
for name, arr in samples.items():
    df_test = pd.DataFrame(arr, columns=model_features)

    # 1️⃣ Apply PowerTransformer to skewed columns (same as training)
    if skewed_cols:
        df_test[skewed_cols] = pt.transform(df_test[skewed_cols])

    # 2️⃣ Then scale everything
    df_scaled = scaler.transform(df_test)

    # 3️⃣ Predict
    pred_log = ridge_model.predict(df_scaled)[0]

    # 4️⃣ Convert back from log1p scale
    pred_orig = np.expm1(pred_log)

    # 5️⃣ Output
    if pred_orig < 1000:
        print(f"{name} → ⚠️ Predicted unusually low (${pred_orig:,.2f}) [log={pred_
    else:
        print(f"{name} → Predicted Median House Value: ${pred_orig:,.2f} [log={pred
```

```
RMSE (val, log1p): 0.3294, R² (log1p): 0.6645
RMSE (val, original $): $69,299.95, R² (orig): 0.6314

💾 Model and Scaler saved successfully! (target transform: log1p)

✅ Feature order confirmed:
['Median_Income', 'Median_Age', 'Tot_Rooms', 'Tot_Bedrooms', 'Population', 'Household
ds', 'Latitude', 'Longitude', 'Distance_to_coast', 'Distance_to_LA', 'Distance_to_Sa
nDiego', 'Distance_to_SanJose', 'Distance_to_SanFrancisco']

🔄 Loaded PowerTransformer and skewed column list.
🌆 Urban LA → Predicted Median House Value: $805,915.25 [log=13.600]
🌊 Coastal San Diego → Predicted Median House Value: $1,042,348.95 [log=13.857]
🌾 Inland Valley → Predicted Median House Value: $530,064.96 [log=13.181]
```

In [244…

```python
# Check if any y <= 0
print("min target:", y.min(), "max target:", y.max())

# Look at distribution
import seaborn as sns, matplotlib.pyplot as plt
sns.histplot(y, bins=50, kde=True); plt.title("Target distribution"); plt.show()
```

```
min target: 14999.0 max target: 500001.0
```



In [245…

```python
test_df = pd.DataFrame(urban_LA, columns=model_features)
test_scaled = scaler.transform(test_df)
pd.DataFrame(test_scaled, columns=model_features)
```

Out[245...

	Median_Income	Median_Age	Tot_Rooms	Tot_Bedrooms	Population	Households	Lat
0	4.988988	0.112947	1499.240374	299.767819	800.589779	279.568506	-0.7