```
# Upload 'housing_price_dataset.csv' to Colab Files before running.
import os, math, numpy as np, pandas as pd, matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import Ridge, LassoCV
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

```
def make_onehot():
    # compatible with different sklearn versions
    try:
        return OneHotEncoder(handle_unknown='ignore', sparse_output=False)
    except TypeError:
        return OneHotEncoder(handle_unknown='ignore', sparse=False)

def metrics(y, yp, name):
    r2 = r2_score(y, yp)
    mae = mean_absolute_error(y, yp)
    rmse = math.sqrt(mean_squared_error(y, yp))
    print(f"{name} -> R²: {r2:.6f}  MAE: {mae:.2f}  RMSE: {rmse:.2f}")
    return {"r2": r2, "mae": mae, "rmse": rmse}

def plot_ap(y, yp, title):
    plt.scatter(y, yp, alpha=0.6, s=18)
    mn, mx = min(y.min(), np.min(yp)), max(y.max(), np.max(yp))
    plt.plot([mn,mx],[mn,mx],'k--'); plt.xlabel("Actual"); plt.ylabel("Predicted"); plt.title(title); plt.grid(True)
```

```
path = "housing_price_dataset.csv"
if not os.path.exists(path):
    raise FileNotFoundError(f"Upload '{path}' to Colab Files first.")
df = pd.read_csv(path)
print("Loaded:", df.shape)
```

```
Loaded: (5000, 7)
```

```
targets = [c for c in df.columns if 'price' in c.lower()]
if not targets:
    raise ValueError("No column containing 'price' found in dataset.")
target = targets[0]
df = df.dropna(subset=[target]).copy()
X = df.drop(columns=[target])
y = df[target].astype(float)
```

```
for c in ['id','Id','ID','description','address','Address']:
    if c in X.columns: X = X.drop(columns=[c])
```

```
num_cols = X.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = X.select_dtypes(exclude=[np.number]).columns.tolist()
ohe = make_onehot()
pre = ColumnTransformer([('num', StandardScaler(), num_cols),
                         ('cat', ohe, cat_cols)], remainder='drop')
X_proc = pre.fit_transform(X)
print("Transformed features shape:", getattr(X_proc, "shape", "unknown"))
```

```
Transformed features shape: (5000, 5)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_proc, y, test_size=0.2, random_state=42)
print("Train:", X_train.shape, "Test:", X_test.shape)
```

```
Train: (4000, 5) Test: (1000, 5)
```

```
ridge = Ridge()
grid = {'alpha':[0.001,0.01,0.1,1.0,10.0]}
gs = GridSearchCV(ridge, grid, cv=5, scoring='r2', n_jobs=1, verbose=0)
gs.fit(X_train, y_train)
best_ridge = gs.best_estimator_
y_pred_r = best_ridge.predict(X_test)
print("\nRidge best alpha:", gs.best_params_['alpha'])
metrics_r = metrics(y_test, y_pred_r, "Ridge")
```

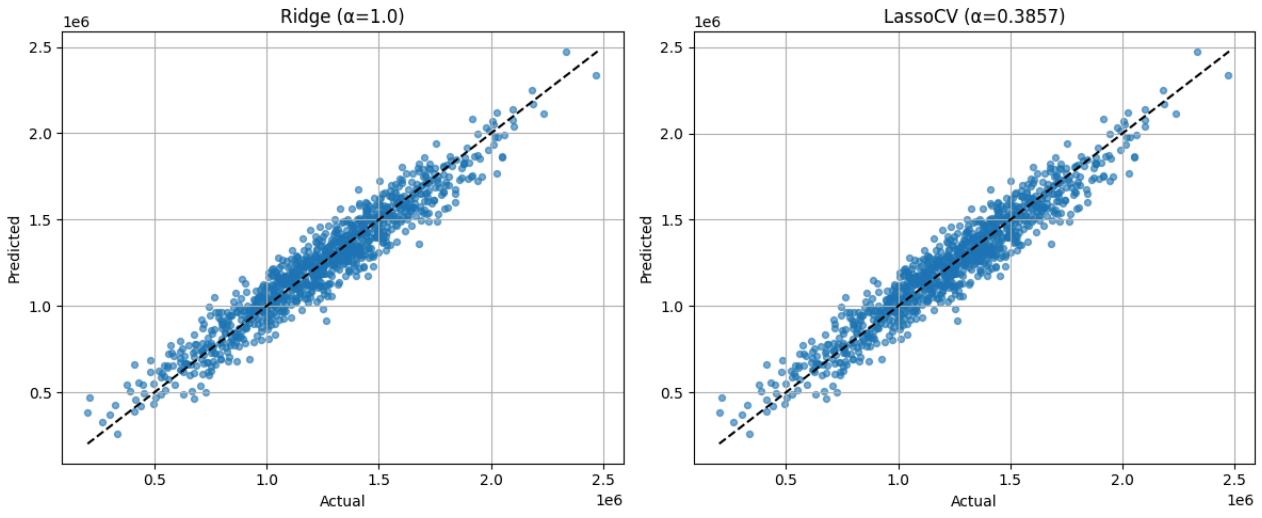
```

Ridge best alpha: 1.0
Ridge -> R²: 0.917997  MAE: 80877.79  RMSE: 100444.04
```

```
alphas = np.logspace(-4, 0, 30)
lassocv = LassoCV(alphas=alphas, cv=5, max_iter=5000, n_jobs=1, random_state=42)
```

```
lassocv.fit(X_train, y_train)
y_pred_l = lassocv.predict(X_test)
print("\nLassoCV best alpha:", lassocv.alpha_)
metrics_l = metrics(y_test, y_pred_l, "Lasso (LassoCV)")
```

```
LassoCV best alpha: 0.38566204211634725
Lasso (LassoCV) -> R²: 0.917997  MAE: 80879.07  RMSE: 100444.02
```

```
plt.figure(figsize=(12,5))
plt.subplot(1,2,1); plot_ap(y_test.values, y_pred_r, f"Ridge (α={gs.best_params_['alpha']})")
plt.subplot(1,2,2); plot_ap(y_test.values, y_pred_l, f"LassoCV (α={lassocv.alpha_:.4g})")
plt.tight_layout(); plt.show()
```



```
better = "Ridge" if metrics_r['r2'] >= metrics_l['r2'] else "Lasso"
print(f"\nBetter model by R²: {better} (Ridge R²={metrics_r['r2']:.6f}, Lasso R²={metrics_l['r2']:.6f})")
```

```
Better model by R²: Lasso (Ridge R²=0.917997, Lasso R²=0.917997)
```