```
import pandas as pd
import numpy as np

import kagglehub
import os

# Download dataset
path = kagglehub.dataset_download("camnugent/california-housing-prices")
print("Dataset path:", path)

# Load CSV
housing_data_path = os.path.join(path, "housing.csv")
df = pd.read_csv(housing_data_path)

df.head()
```

```
Using Colab cache for faster access to the 'california-housing-prices' dataset.
Dataset path: /kaggle/input/california-housing-prices
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_valu
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200

Next steps: Generate code with df | New interactive sheet

```
df = pd.get_dummies(df, drop_first=True)
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_valu
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200

Next steps: Generate code with df | New interactive sheet

```
df= df.dropna()
print(df.head())
```

```
   longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0    -122.23     37.88                41.0        880.0           129.0
1    -122.22     37.86                21.0       7099.0          1106.0
2    -122.24     37.85                52.0       1467.0           190.0
3    -122.25     37.85                52.0       1274.0           235.0
4    -122.25     37.85                52.0       1627.0           280.0

   population  households  median_income  median_house_value  \
0       322.0       126.0         8.3252            452600.0
1      2401.0      1138.0         8.3014            358500.0
2       496.0       177.0         7.2574            352100.0
3       558.0       219.0         5.6431            341300.0
4       565.0       259.0         3.8462            342200.0

   ocean_proximity_INLAND  ocean_proximity_ISLAND  ocean_proximity_NEAR BAY  \
0                   False                   False                      True
1                   False                   False                      True
2                   False                   False                      True
3                   False                   False                      True
4                   False                   False                      True

   ocean_proximity_NEAR OCEAN
```

```
0                     False
1                     False
2                     False
3                     False
4                     False
```

```
X = df.drop("median_house_value", axis=1)
Y = df["median_house_value"]
```

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error


X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.2, random_state=42
)

print("Training set size:", X_train.shape)
print("Test set size:", X_test.shape)
```

```
Training set size: (16346, 12)
Test set size: (4087, 12)
```

```
from sklearn . preprocessing import StandardScaler
scaled_x= StandardScaler()
X_train_scaled =  scaled_x.fit_transform(X_train)
X_test_scaled= scaled_x.transform(X_test)
```

```
scaled_Y= StandardScaler()
Y_train_scaled = scaled_Y.fit_transform(Y_train.values.reshape(-1, 1)).ravel() # StandardScaler expects a 2D array but y_train
Y_test_scaled = scaled_Y.transform(Y_test.values.reshape(-1, 1)).ravel()  # ravel. Converts 2D array back to 1D
scaled= scaled_Y.transform(Y_test.values.reshape(-1,1)).ravel()
```

```
#classification task
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
linear_model= LinearRegression()
linear_model.fit(X_train_scaled, Y_train_scaled)

y_train_pred= linear_model.predict(X_train_scaled)
y_test_pred= linear_model.predict(X_test_scaled)

train_mse= mean_squared_error(Y_train_scaled, y_train_pred)
print("train_mse",train_mse)
test_mse= mean_squared_error(Y_test_scaled, y_test_pred)
print("test_mse",test_mse)

#observe coefficient
coefficient_df= pd.DataFrame({
    'Feature': X.columns,
    'Coefficient': linear_model.coef_
})

print(coefficient_df.sort_values('Coefficient',key=abs, ascending=False))
```

```
train_mse 0.3543517602353898
test_mse 0.36278746237466286
                      Feature  Coefficient
7               median_income     0.647872
1                    latitude    -0.476378
0                   longitude    -0.472620
4              total_bedrooms     0.373744
5                  population    -0.357397
8      ocean_proximity_INLAND    -0.158489
6                  households     0.141739
3                 total_rooms    -0.118322
2          housing_median_age     0.118209
9      ocean_proximity_ISLAND     0.025155
10    ocean_proximity_NEAR BAY    -0.017129
11  ocean_proximity_NEAR OCEAN     0.009235
```

```
print(f"\nTraining set size: {X_train_scaled.shape[0]}")
print(f"Test set size: {X_test_scaled.shape[0]}")
```

```
Training set size: 16346
Test set size: 4087
```

```
print("\n" + "="*70) #"\n" → inserts a blank line before the output and  "=" * 70 → repeats the = character 70 times
print("STEP 2: HYPERPARAMETER TUNING WITH GRIDSEARCHCV") #Clearly labels the next phase of the workflow → Model optimization
print("="*70) #This creates a boxed header effect:
#Visually separates sections in the output and Makes console output easier to scan
```

```
======================================================================
STEP 2: HYPERPARAMETER TUNING WITH GRIDSEARCHCV
======================================================================
```

```
alpha_grid={
    'alpha':[0.001,0.01,0.1,10,1000,10000]
}
print("alpha values are ",alpha_grid['alpha'])
```

```
alpha values are  [0.001, 0.01, 0.1, 10, 1000, 10000]
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge, Lasso
#_____________________________________________________________
print("for ridge")
#_____________________________________________________________
#creating ridge model
ridge= Ridge()
#performing gridsearch cv with k fold =5
ridge_grid_search= GridSearchCV(
    estimator=ridge,
    param_grid=alpha_grid,
    cv= 5,
    scoring='neg_mean_squared_error',
    n_jobs=-1,
    verbose=1
)
print("trainning data")
ridge_grid_search.fit(X_train_scaled,Y_train_scaled)
```

```
for ridge
trainning data
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

▸ **GridSearchCV** ⓘ ?

▸ **best_estimator_: Ridge**

▸ Ridge ?

```
# Get the best parameters
best_ridge_alpha = ridge_grid_search.best_params_['alpha']
print(f"Best alpha for Ridge: {best_ridge_alpha}")

# Get cross-validation results
ridge_cv_results = pd.DataFrame(ridge_grid_search.cv_results_)
ridge_cv_results['mean_mse'] = -ridge_cv_results['mean_test_score']
ridge_cv_results['std_mse'] = ridge_cv_results['std_test_score']

print("\nCross-Validation Results for Ridge:")
print(ridge_cv_results[['param_alpha', 'mean_mse', 'std_mse']].to_string(index=False))
```

```
Best alpha for Ridge: 10

Cross-Validation Results for Ridge:
 param_alpha  mean_mse  std_mse
       0.001  0.357540 0.029032
       0.010  0.357540 0.029032
```

```
       0.100  0.357540 0.029031
      10.000  0.357493 0.028856
    1000.000  0.369099 0.022582
   10000.000  0.476420 0.019969
```

```
#trainning the best ridge model
best_ridge_model= Ridge(alpha=best_ridge_alpha)
best_ridge_model.fit(X_train_scaled,Y_train_scaled)

#make prediction from best model
rigid_train_pred= best_ridge_model.predict(X_train_scaled)
rigid_test_pred= best_ridge_model.predict(X_test_scaled)

rigid_train_mse= mean_squared_error(Y_train_scaled,rigid_train_pred )
rigid_test_mse= mean_squared_error(Y_test_scaled,rigid_test_pred )

print(f"trainning mse {rigid_train_mse:.4f}")
print(f"testing mse {rigid_test_mse:.4f}")
```

```
trainning mse 0.3544
testing mse 0.3628
```

```
#for lasso
lasso= Lasso(max_iter=10000)
lasso_grid_search= GridSearchCV(
    estimator=lasso,
    param_grid=alpha_grid,
    cv=5,
    scoring='neg_mean_squared_error',
    n_jobs= -1,
    verbose =1
)
print("trainnig the lasso model")
lasso_grid_search.fit(X_train_scaled, Y_train_scaled)
```

```
trainnig the lasso model
Fitting 5 folds for each of 6 candidates, totalling 30 fits
```

▸ **GridSearchCV**

▸ **best_estimator_: Lasso**

▸ Lasso

```
best_laso_alpha= lasso_grid_search.best_params_['alpha']
print("best alpha for lasso is",best_laso_alpha)


lasso_cv_results = pd.DataFrame(lasso_grid_search.cv_results_)
lasso_cv_results['mean_mse'] = -lasso_cv_results['mean_test_score']
lasso_cv_results['std_mse'] = lasso_cv_results['std_test_score']

print("\nCross-Validation Results for Lasso:")
print(lasso_cv_results[['param_alpha', 'mean_mse', 'std_mse']].to_string(index=False))
```

```
best alpha for lasso is 0.001

Cross-Validation Results for Lasso:
 param_alpha  mean_mse  std_mse
       0.001  0.357523 0.028480
       0.010  0.364981 0.024084
       0.100  0.431730 0.021229
      10.000  1.000163 0.036360
    1000.000  1.000163 0.036360
   10000.000  1.000163 0.036360
```

```
#training the model for lasso
best_lasso_model= Lasso(alpha=best_laso_alpha, max_iter=10000)
best_lasso_model.fit(X_train_scaled, Y_train_scaled)

# making predictions of lasso
lasso_train_pred= best_lasso_model.predict(X_train_scaled)
lasso_test_pred= best_lasso_model.predict(X_test_scaled)
```

```
#calculating mse
lasso_train_mse= mean_squared_error(Y_train_scaled, lasso_train_pred)
lasso_test_mse= mean_squared_error(Y_test_scaled, lasso_test_pred)

print("lasso models errors")
print(f"trainning mse is {lasso_train_mse:.4f}")
print(f"testing mse is {lasso_test_mse:.4f}")
```

```
lasso models errors
trainning mse is 0.3545
testing mse is 0.3629
```

```
# Load Breast Cancer Dataset
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

# Load the dataset
cancer_data = load_breast_cancer()
X_cancer = cancer_data.data
y_cancer = cancer_data.target
feature_names_cancer = cancer_data.feature_names
target_names_cancer = cancer_data.target_names

print(f"Dataset loaded: {X_cancer.shape[0]} samples, {X_cancer.shape[1]} features")
print(f"Target classes: {target_names_cancer} (0: {target_names_cancer[0]}, 1: {target_names_cancer[1]})")
print(f"Class distribution: {np.bincount(y_cancer)}")

# Split into training (80%) and test (20%) sets
X_train_cancer, X_test_cancer, y_train_cancer, y_test_cancer = train_test_split(
    X_cancer, y_cancer, test_size=0.2, random_state=42, stratify=y_cancer
)

print(f"\nTraining set size: {X_train_cancer.shape[0]} samples")
print(f"Test set size: {X_test_cancer.shape[0]} samples")
print(f"Training class distribution: {np.bincount(y_train_cancer)}")
print(f"Test class distribution: {np.bincount(y_test_cancer)}")

# Scale features for classification
from sklearn.preprocessing import StandardScaler

scaler_cancer = StandardScaler()
X_train_cancer_scaled = scaler_cancer.fit_transform(X_train_cancer)
X_test_cancer_scaled = scaler_cancer.transform(X_test_cancer)
```

```
Dataset loaded: 569 samples, 30 features
Target classes: ['malignant' 'benign'] (0: malignant, 1: benign)
Class distribution: [212 357]

Training set size: 455 samples
Test set size: 114 samples
Training class distribution: [170 285]
Test class distribution: [42 72]
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
# Train baseline logistic regression (default parameters - usually L2 with C=1.0)
baseline_logistic = LogisticRegression(random_state=42, max_iter=1000)
baseline_logistic.fit(X_train_cancer_scaled, y_train_cancer)

# Make predictions
y_train_pred_baseline = baseline_logistic.predict(X_train_cancer_scaled)
y_test_pred_baseline = baseline_logistic.predict(X_test_cancer_scaled)

# Calculate accuracy
train_acc_baseline = accuracy_score(y_train_cancer, y_train_pred_baseline)
test_acc_baseline = accuracy_score(y_test_cancer, y_test_pred_baseline)

print(f"Baseline Logistic Regression Accuracy:")
print(f"  Training Accuracy: {train_acc_baseline:.4f}")
print(f"  Test Accuracy:     {test_acc_baseline:.4f}")

# Observe coefficients
coefficients_baseline = pd.DataFrame({
    'Feature': feature_names_cancer,
    'Coefficient': baseline_logistic.coef_[0],
    'Absolute_Coefficient': np.abs(baseline_logistic.coef_[0])
})

print("\nTop 10 Features by Absolute Coefficient Value (Baseline Model):")
print(coefficients_baseline.sort_values('Absolute_Coefficient', ascending=False).head(10).to_string(index=False))

# Display classification report
print("\nClassification Report (Test Set - Baseline):")
print(classification_report(y_test_cancer, y_test_pred_baseline, target_names=target_names_cancer))
```

```
Baseline Logistic Regression Accuracy:
  Training Accuracy: 0.9890
  Test Accuracy:     0.9825

Top 10 Features by Absolute Coefficient Value (Baseline Model):
             Feature  Coefficient  Absolute_Coefficient
       worst texture    -1.255088              1.255088
        radius error    -1.082965              1.082965
worst concave points    -0.953686              0.953686
          worst area    -0.947756              0.947756
        worst radius    -0.947616              0.947616
      worst symmetry    -0.939181              0.939181
          area error    -0.929104              0.929104
     worst concavity    -0.823151              0.823151
     worst perimeter    -0.763220              0.763220
    worst smoothness    -0.746625              0.746625

Classification Report (Test Set - Baseline):
              precision    recall  f1-score   support

   malignant       0.98      0.98      0.98        42
      benign       0.99      0.99      0.99        72

    accuracy                           0.98       114
   macro avg       0.98      0.98      0.98       114
weighted avg       0.98      0.98      0.98       114
```

```
from sklearn.model_selection import GridSearchCV

# Define parameter grid for Logistic Regression
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],  # Inverse of regularization strength
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga']  # Solvers that support both L1 and L2
}

# Create and train GridSearchCV
```

```
logistic_grid_search = GridSearchCV(
    estimator=LogisticRegression(random_state=42, max_iter=5000),
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)

print("Running GridSearchCV for Logistic Regression...")
logistic_grid_search.fit(X_train_cancer_scaled, y_train_cancer)

# Get best parameters
best_params = logistic_grid_search.best_params_
best_score = logistic_grid_search.best_score_
print(f"\nBest Parameters: {best_params}")
print(f"Best Cross-Validation Accuracy: {best_score:.4f}")

# Display grid search results
cv_results_df = pd.DataFrame(logistic_grid_search.cv_results_)
print("\nTop 5 Parameter Combinations:")
print(cv_results_df[['param_C', 'param_penalty', 'mean_test_score', 'std_test_score']]
      .sort_values('mean_test_score', ascending=False)
      .head(5).to_string(index=False))
```

```
Running GridSearchCV for Logistic Regression...
Fitting 5 folds for each of 28 candidates, totalling 140 fits

Best Parameters: {'C': 0.1, 'penalty': 'l2', 'solver': 'saga'}
Best Cross-Validation Accuracy: 0.9802

Top 5 Parameter Combinations:
 param_C param_penalty  mean_test_score  std_test_score
     0.1            l2         0.980220        0.016150
     1.0            l2         0.980220        0.012815
     1.0            l2         0.980220        0.012815
     0.1            l2         0.978022        0.013900
     1.0            l1         0.975824        0.012815
```

```
# Train L1 (Lasso-like) and L2 (Ridge-like) models with best C values
# Let's extract best C for each penalty type from grid search
l1_results = cv_results_df[cv_results_df['param_penalty'] == 'l1']
l2_results = cv_results_df[cv_results_df['param_penalty'] == 'l2']

best_l1_idx = l1_results['mean_test_score'].idxmax()
best_l2_idx = l2_results['mean_test_score'].idxmax()

best_l1_params = {
    'C': cv_results_df.loc[best_l1_idx, 'param_C'],
    'penalty': 'l1',
    'solver': 'liblinear'
}

best_l2_params = {
    'C': cv_results_df.loc[best_l2_idx, 'param_C'],
    'penalty': 'l2',
    'solver': 'lbfgs'  # Efficient for L2
}

print(f"Best L1 Parameters: {best_l1_params}")
print(f"Best L2 Parameters: {best_l2_params}")
```

```
# Train L1 model
l1_model = LogisticRegression(
    penalty='l1',
    C=best_l1_params['C'],
    solver='liblinear',
    random_state=42,
    max_iter=5000
)
l1_model.fit(X_train_cancer_scaled, y_train_cancer)

# Train L2 model
l2_model = LogisticRegression(
    penalty='l2',
    C=best_l2_params['C'],
    solver='lbfgs',
    random_state=42,
    max_iter=5000
)
l2_model.fit(X_train_cancer_scaled, y_train_cancer)

# Make predictions
y_train_pred_l1 = l1_model.predict(X_train_cancer_scaled)
y_test_pred_l1 = l1_model.predict(X_test_cancer_scaled)

y_train_pred_l2 = l2_model.predict(X_train_cancer_scaled)
y_test_pred_l2 = l2_model.predict(X_test_cancer_scaled)

# Calculate accuracy
train_acc_l1 = accuracy_score(y_train_cancer, y_train_pred_l1)
test_acc_l1 = accuracy_score(y_test_cancer, y_test_pred_l1)

train_acc_l2 = accuracy_score(y_train_cancer, y_train_pred_l2)
test_acc_l2 = accuracy_score(y_test_cancer, y_test_pred_l2)

# Create comparison table
comparison_df = pd.DataFrame({
    'Model': ['Baseline (L2, C=1)', f'L1 (C={best_l1_params["C"]})', f'L2 (C={best_l2_params["C"]})'],
    'Train Accuracy': [train_acc_baseline, train_acc_l1, train_acc_l2],
    'Test Accuracy': [test_acc_baseline, test_acc_l1, test_acc_l2],
    'Num Non-Zero Coefficients': [
        np.sum(baseline_logistic.coef_[0] != 0),
        np.sum(l1_model.coef_[0] != 0),
        np.sum(l2_model.coef_[0] != 0)
    ]
})

print("\nModel Comparison:")
print(comparison_df.to_string(index=False))

# Compare coefficients
coefficients_l1 = pd.DataFrame({
    'Feature': feature_names_cancer,
    'L1_Coefficient': l1_model.coef_[0],
    'L2_Coefficient': l2_model.coef_[0]
})

coefficients_l1['L1_Abs'] = np.abs(coefficients_l1['L1_Coefficient'])
coefficients_l1['L2_Abs'] = np.abs(coefficients_l1['L2_Coefficient'])

print("\nSparsity Analysis:")
print(f"L1 Model: {np.sum(l1_model.coef_[0] == 0)} zero coefficients out of {len(l1_model.coef_[0])}")
print(f"L2 Model: {np.sum(l2_model.coef_[0] == 0)} zero coefficients out of {len(l2_model.coef_[0])}")

print("\nTop 10 Features by L1 Coefficient (Absolute Value):")
print(coefficients_l1.sort_values('L1_Abs', ascending=False).head(10)[['Feature', 'L1_Coefficient', 'L2_Coefficient']].to_string

print("\nFeatures with Zero Coefficients in L1 Model:")
zero_features = coefficients_l1[coefficients_l1['L1_Coefficient'] == 0]
if len(zero_features) > 0:
    print(zero_features[['Feature']].to_string(index=False))
else:
    print("No features with exactly zero coefficients")
```

```
Best L1 Parameters: {'C': np.float64(1.0), 'penalty': 'l1', 'solver': 'liblinear'}
Best L2 Parameters: {'C': np.float64(0.1), 'penalty': 'l2', 'solver': 'lbfgs'}

Model Comparison:
```

```
                Model  Train Accuracy  Test Accuracy  Num Non-Zero Coefficients
Baseline (L2, C=1)        0.989011       0.982456                         30
        L1 (C=1.0)        0.989011       0.991228                         16
        L2 (C=0.1)        0.986813       0.973684                         30

Sparsity Analysis:
L1 Model: 14 zero coefficients out of 30
L2 Model: 0 zero coefficients out of 30

Top 10 Features by L1 Coefficient (Absolute Value):
             Feature  L1_Coefficient  L2_Coefficient
          worst area       -3.529897       -0.460311
        radius error       -2.167842       -0.439074
worst concave points       -1.705453       -0.507791
        worst radius       -1.248721       -0.509038
       worst texture       -1.205456       -0.563530
   compactness error        0.698341        0.227123
     worst concavity       -0.677940       -0.357845
      worst symmetry       -0.663777       -0.412812
    worst smoothness       -0.609725       -0.376370
 mean concave points       -0.489200       -0.403913

Features with Zero Coefficients in L1 Model:
                Feature
            mean radius
         mean perimeter
              mean area
        mean smoothness
       mean compactness
         mean concavity
          mean symmetry
        perimeter error
             area error
        concavity error
         symmetry error
        worst perimeter
      worst compactness
worst fractal dimension
```