

```

# =====
# Task 2: House Price Prediction (Regression)
# Dataset: California Housing (housing.csv)
# =====

import argparse
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.impute import SimpleImputer

from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error

def rmse(y_true, y_pred):
    return np.sqrt(mean_squared_error(y_true, y_pred))

def build_preprocessor(X: pd.DataFrame):
    # Detect numeric and categorical columns
    numeric_features = X.select_dtypes(include=["int64", "float64"]).columns.tolist()
    categorical_features = X.select_dtypes(include=["object"]).columns.tolist()

    numeric_transformer = Pipeline(steps=[
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler", StandardScaler()) # scaling needed for Ridge/Lasso
    ])

    categorical_transformer = Pipeline(steps=[
        ("imputer", SimpleImputer(strategy="most_frequent")),
        ("onehot", OneHotEncoder(handle_unknown="ignore"))
    ])

    preprocessor = ColumnTransformer(
        transformers=[
            ("num", numeric_transformer, numeric_features),
            ("cat", categorical_transformer, categorical_features),
        ],
        remainder="drop"
    )
    return preprocessor, numeric_features, categorical_features

def train_and_evaluate(X_train, X_test, y_train, y_test, model, model_name):
    preprocessor, _, _ = build_preprocessor(X_train)

    pipeline = Pipeline(steps=[
        ("preprocess", preprocessor),
        ("model", model)
    ])

    pipeline.fit(X_train, y_train)

    # Predictions
    pred_train = pipeline.predict(X_train)
    pred_test = pipeline.predict(X_test)

    # Metrics
    train_rmse = rmse(y_train, pred_train)
    test_rmse = rmse(y_test, pred_test)
    test_mae = mean_absolute_error(y_test, pred_test)

    return {
        "Model": model_name,
        "RMSE (Train)": train_rmse,
        "RMSE (Test)": test_rmse,
        "MAE (Test)": test_mae
    }

# ===== Load Dataset =====

```

```

data_path = "housing.csv" # in colab, upload housing.csv and keep this name
df = pd.read_csv(data_path)

# Target column in this dataset
target = "median_house_value"

# Split features & target
X = df.drop(columns=[target])
y = df[target]

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# ===== Models (at least 3) =====
results = []

# 1) Linear Regression baseline
results.append(train_and_evaluate(
    X_train, X_test, y_train, y_test,
    model=LinearRegression(),
    model_name="Linear Regression"
))



# 2) Ridge Regression (regularized)
results.append(train_and_evaluate(
    X_train, X_test, y_train, y_test,
    model=Ridge(alpha=1.0, random_state=42),
    model_name="Ridge (alpha=1.0)"
))

# 3) Decision Tree - show both underfit and overfit behavior by changing depth
# For table, keep one tree model. But we will also demonstrate under/overfit below.

results.append(train_and_evaluate(
    X_train, X_test, y_train, y_test,
    model=DecisionTreeRegressor(max_depth=5, random_state=42),
    model_name="Decision Tree (max_depth=5)"
))

# ===== Comparison Table =====
comparison = pd.DataFrame(results)
comparison = comparison.sort_values(by="RMSE (Test)")
comparison

```

	Model	RMSE (Train)	RMSE (Test)	MAE (Test)	
0	Linear Regression	68433.937367	70059.193339	50670.489236	
1	Ridge (alpha=1.0)	68434.995896	70066.021121	50676.922171	
2	Decision Tree (max_depth=5)	69127.806177	71452.881461	50295.915499	

Next steps: [Generate code with comparison](#) [New interactive sheet](#)

```
!find /content -name "housing.csv"
```


```

# Underfitting vs Overfitting using Decision Tree depth
depths = [2, 5, 10, 20, None] # None = unlimited depth (usually overfits)

tree_rows = []
for d in depths:
    name = f"Tree depth={d}" if d is not None else "Tree depth=None (unlimited)"
    row = train_and_evaluate(
        X_train, X_test, y_train, y_test,
        model=DecisionTreeRegressor(max_depth=d, random_state=42),
        model_name=name
    )
    tree_rows.append(row)

pd.DataFrame(tree_rows)

```

	Model	RMSE (Train)	RMSE (Test)	MAE (Test)	
0	Tree depth=2	81918.085075	83027.754093	61644.403638	
1	Tree depth=5	69127.806177	71452.881461	50295.915499	
2	Tree depth=10	48589.563780	61279.694223	40556.375765	
3	Tree depth=20	8371.593617	68496.950256	43281.775760	
4	Tree depth=None (unlimited)	0.000000	69175.769189	43604.014293	

✓ Brief Note (Underfitting vs Overfitting + real-world issue)

- **Underfitting (high bias)** happens when the model is too simple (e.g., Decision Tree with very small depth like 2). In that case, **Train RMSE and Test RMSE both stay high**, so the model cannot learn enough.
- A ****balanced**

```
!ls -l /content
```

```
total 404
-rw-r--r-- 1 root root 409382 Jan 21 15:46 housing.csv.zip
drwxr-xr-x 1 root root  4096 Jan 16 14:24 sample_data
```

```
!unzip housing.csv.zip
```

```
Archive:  housing.csv.zip
  inflating: housing.csv
```

```
!ls -l /content
```

```
total 1796
-rw-r--r-- 1 root root 1423529 Sep 22  2019 housing.csv
-rw-r--r-- 1 root root  409382 Jan 21 15:46 housing.csv.zip
drwxr-xr-x 1 root root  4096 Jan 16 14:24 sample_data
```