import pandas as pd import numpy as np import matplotlib.pyplot as plt import seabornassnsfromsklearn.model\_selectionimporttrain\_test\_split,GridSearchCV from sklearn.preprocessing import StandardScaler, PolynomialFeatures from sklearn.linear\_model import LinearRegression, Ridge, Lasso from sklearn.tree import DecisionTreeRegressor from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor from xgboost import XGBRegressor from sklearn.svm import SVR from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score from sklearn.pipeline import make\_pipelineprint("\n=== LoadingData===")#Loaddataset(replacewithyour dataset)

url = "https://raw.githubusercontent.com/ageron/handsonml2/ master/datasets/housing/housing.csv" data = pd.read\_csv(url) print(f"\nData Shape: {data.shape}")

print("\nFirst5Rows:")print(data.head())#BasicEDA Visualizations

plt.figure(figsize=(15, 10)) # Distribution of house prices plt.subplot(2, 2, 1) sns.histplot(data['median\_house\_value'], kde=True, bins=30) plt.title('House Price Distribution')

#Correlationheatmap plt.subplot(2, 2, 2)

#Selectonlynumericcolumnsnumeric\_data= data.select\_dtypes(include=['number'])

#Computecorrelationmatrixcorr=numeric\_data.corr()# Plot heatmap sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".1f")

plt.title('FeatureCorrelation')# Price vs. median income plt.subplot(2, 2, 3)

sns.scatterplot(x='median\_income',y='median\_house\_value',data=data,alpha=0.3) plt.title('Price vs. Income') # Price by ocean proximity

plt.subplot(2,2,4)

sns.boxplot(x='ocean\_proximity',y='median\_house\_value',data=data) plt.xticks(rotation=45) plt.title('Price by Location') plt.tight\_layout() plt.show() print("\n=== Preprocessing Data ===")

#Handlemissingvaluesdata.fillna(data.select\_dtypes(include='number').median(), inplace=True)

# Feature engineering data['rooms\_per\_household'] = data['total\_rooms']/data['households'] data['bedrooms\_per\_room'] = data['total\_bedrooms']/data['total\_rooms'] # Convert categorical to numericaldata=pd.get\_dummies(data,columns=['ocean\_proximity'])

#Selectfeaturesand target

X=data.drop('median\_house\_value',axis=1)y

=data['median\_house\_value'] # Train-test split

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2,random\_state=42) # Feature scaling scaler

= StandardScaler()

X\_train\_scaled=scaler.fit\_transform(X\_train) X\_test\_scaled = scaler.transform(X\_test) print("\n===TrainingModels===")models=

{

"LinearRegression":LinearRegression(), "Ridge Regression": Ridge(alpha=1.0), "Lasso Regression": Lasso(alpha=0.1),

"DecisionTree":DecisionTreeRegressor(max\_depth=5),

"Random Forest": RandomForestRegressor(n\_estimators=100, random\_state=42), "GradientBoosting":GradientBoostingRegressor(n\_estimators=100,random\_state=42), "XGBoost": XGBRegressor(n\_estimators=100, random\_state=42),

"SVR":SVR(kernel='rbf')

}results={}forname,modelin models.items():

print(f"Training {name}...") model.fit(X\_train\_scaled[:1000], y\_train[:1000]) y\_pred

= model.predict(X\_test\_scaled) results[name]={

"MAE": mean\_absolute\_error(y\_test, y\_pred), "RMSE":np.sqrt(mean\_squared\_error(y\_test,y\_pred)),

"R2":r2\_score(y\_test,y\_pred)

}

# Display results results\_df = pd.DataFrame(results).T print("\n=== Model Performance ===") print(results\_df.sort\_values(by='RMSE')) print("\n=== Optimizing Best Model

===")

# Let's optimize Random Forest as it typically performs well fromsklearn.model\_selectionimportRandomizedSearchCV# Smaller parameter grid or use RandomizedSearchCV param\_dist = {

'n\_estimators':[50,100,200],

'max\_depth':[None,10,20],

'min\_samples\_split':[2,5,10]

} rf = RandomForestRegressor(random\_state=42)

random\_search=RandomizedSearchCV(rf,param\_distributions=param\_dist,n\_iter=5, cv=2, scoring='neg\_mean\_squared\_error', n\_jobs=- 1,verbose=1,random\_state=42,error\_score='raise')random\_search.fit(X\_train\_scaled,y\_train) best\_model = random\_search.best\_estimator\_

# Evaluate optimized model y\_pred = best\_model.predict(X\_test\_scaled)print("\nOptimized Model Performance:") print(f"MAE:

{mean\_absolute\_error(y\_test, y\_pred):.2f}") print(f"RMSE: {np.sqrt(mean\_squared\_error(y\_test, y\_pred)):.2f}") print(f"R2 Score: {r2\_score(y\_test,

y\_pred):.4f}")print("\n===GeneratingVisualizations

===")

#FeatureImportanceplt.figure(figsize=(10, 6))

importances = best\_model.feature\_importances\_ features = X.columns indices = np.argsort(importances)[-10:]# Top 10 features plt.title('Feature Importances') plt.barh(range(len(indices)), importances[indices], color='b', align='center') plt.yticks(range(len(indices)), [features[i] for i in indices]) plt.xlabel('Relative Importance') plt.show() # Actual vs Predicted plt.figure(figsize=(10, 6)) plt.scatter(y\_test, y\_pred, alpha=0.3) plt.plot([y\_test.min(),y\_test.max()],[y\_test.min(),y\_test.max()],'k--',lw=2) plt.xlabel('Actual Prices') plt.ylabel('Predicted Prices') plt.title('Actual vs Predicted House Prices') plt.show() # Residual Plot residuals = y\_test - y\_pred plt.figure(figsize=(10, 6)) plt.scatter(y\_pred, residuals, alpha=0.3) plt.axhline(y=0, color='r', linestyle='--') plt.xlabel('Predicted Prices') plt.ylabel('Residuals') plt.title('Residual Plot')

plt.show()print("\n===ProgramExecution Complete ===")

OUTPUT: