import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.linear\_model import LinearRegression, Ridge, Lasso

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

import warnings

# Step 2: Loading the Dataset

# ------------------------------

file\_path = 'synthetic\_house\_prices.csv'

data = pd.read\_csv(file\_path)

# Display the first five rows

print(data.head())

# Summary statistics

print(data.describe())

# Check for missing values

print(data.isnull().sum())

# Info of the dataset

print(data.info())

# Check for non-numeric columns

non\_numeric\_columns = data.select\_dtypes(exclude=[np.number]).columns

print(f"Non-numeric columns: {non\_numeric\_columns}")

# If there are non-numeric columns, drop them for correlation

if len(non\_numeric\_columns) > 0:

data\_numeric = data.drop(columns=non\_numeric\_columns)

else:

data\_numeric = data

# Generate the correlation matrix

corr\_matrix = data\_numeric.corr()

# Check for NaN values

print(f"NaN values in correlation matrix: {corr\_matrix.isnull().sum().sum()}")

# Plot the heatmap

plt.figure(figsize=(12, 8))

sns.heatmap(corr\_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Correlation Heatmap')

plt.show()

# Pair Plot for Numerical Features

sns.pairplot(data[['LotArea', 'GrLivArea', 'OverallQual', 'SalePrice']])

plt.show()

# Distribution of SalePrice

plt.figure(figsize=(8, 5))

sns.histplot(data['SalePrice'], kde=True, color='green')

plt.title('Distribution of Sale Price')

plt.show()

# Bar Plot for Neighborhood

plt.figure(figsize=(10, 5))

sns.barplot(x=data['Neighborhood'].value\_counts().index, y=data['Neighborhood'].value\_counts().values, palette='viridis')

plt.title('Number of Houses by Neighborhood')

plt.xticks(rotation=45)

plt.show()

""

# ------------------------------

# Step 4: Data Preprocessing

# ------------------------------

# 1. Encoding categorical features

data = pd.get\_dummies(data, columns=['Neighborhood'], drop\_first=True)

# 2. Splitting features and target variable

X = data.drop('SalePrice', axis=1)

y = data['SalePrice']

# 3. Train-Test Split

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

# Display shapes of training and testing data

print(f"Training data shape: {X\_train.shape}")

print(f"Testing data shape: {X\_test.shape}")

# Ddefine models to train

models = {

'Linear Regression': LinearRegression(),

'Ridge Regression': Ridge(),

'Lasso Regression': Lasso(),

'Random Forest': RandomForestRegressor(),

'Gradient Boosting': GradientBoostingRegressor()

}

# Train and evaluate each model

for name, model in models.items():

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

print(f"{name} - MAE: {mae:.2f}, MSE: {mse:.2f}, RMSE: {rmse:.2f}, R2: {r2:.2f}")

# Example: Hyperparameter tuning for Random Forest

param\_grid = {

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

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

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

}

grid\_search = GridSearchCV(RandomForestRegressor(), param\_grid, cv=3, scoring='neg\_mean\_squared\_error')

grid\_search.fit(X\_train, y\_train)

print(f"Best parameters for Random Forest: {grid\_search.best\_params\_}")

print(f"Best cross-validation score: {grid\_search.best\_score\_:.2f}")

# Ensure grid\_search is fitted before accessing the best estimator

if grid\_search.best\_estimator\_ is not None:

best\_rf = grid\_search.best\_estimator\_

else:

print("Error: GridSearchCV wasn't fitted successfully.")

# Train the best model (using Random Forest in this case) on the entire dataset

best\_rf.fit(X\_train, y\_train)

# Make final predictions on the test data

final\_predictions = best\_rf.predict(X\_test)

# Evaluate final model performance

final\_mae = mean\_absolute\_error(y\_test, final\_predictions)

final\_mse = mean\_squared\_error(y\_test, final\_predictions)

final\_rmse = np.sqrt(final\_mse)

final\_r2 = r2\_score(y\_test, final\_predictions)

print(f"Final Model - MAE: {final\_mae:.2f}, MSE: {final\_mse:.2f}, RMSE: {final\_rmse:.2f}, R2: {final\_r2:.2f}")

# ------------------------------

# Step 8: Conclusion

# ------------------------------

# Check if final\_predictions are available and valid

if len(final\_predictions) > 0:

final\_mae = mean\_absolute\_error(y\_test, final\_predictions)

final\_mse = mean\_squared\_error(y\_test, final\_predictions)

final\_rmse = np.sqrt(final\_mse)

final\_r2 = r2\_score(y\_test, final\_predictions)

# Display final performance metrics

print(f"Final Model - MAE: {final\_mae:.2f}, MSE: {final\_mse:.2f}, RMSE: {final\_rmse:.2f}, R2: {final\_r2:.2f}")

else:

print("Error: Final predictions are empty or not available.")

# Optionally, save the final model

import joblib

try:

joblib.dump(best\_rf, 'house\_price\_forecasting\_model.pkl')

print("Model saved as 'house\_price\_forecasting\_model.pkl'")

except Exception as e:

print(f"Error saving model: {e}")

# Step 9: Model Comparison & Reporting

# ------------------------------

# Store model names and their corresponding evaluation metrics

model\_comparison = []

# Train and evaluate each model

for name, model in models.items():

model.fit(X\_train, y\_train)

y\_pred = model.predict(X\_test)

# Compute performance metrics

mae = mean\_absolute\_error(y\_test, y\_pred)

mse = mean\_squared\_error(y\_test, y\_pred)

rmse = np.sqrt(mse)

r2 = r2\_score(y\_test, y\_pred)

# Store the metrics in a list

model\_comparison.append({

'Model': name,

'MAE': mae,

'MSE': mse,

'RMSE': rmse,

'R2': r2

})

# Convert the list into a DataFrame for easier visualization

comparison\_df = pd.DataFrame(model\_comparison)

# Sort models by R2 score for better readability

comparison\_df = comparison\_df.sort\_values(by='R2', ascending=False)

# Display the model comparison table

print("Model Comparison Report:")

print(comparison\_df)

# Optionally, visualize model performance using a bar plot for R2 score

plt.figure(figsize=(10, 5))

sns.barplot(x='R2', y='Model', data=comparison\_df, palette='viridis')

plt.title('Model Comparison (R2 Score)')

plt.show()

# Example: Saving Model Comparison to CSV

comparison\_df.to\_csv('model\_comparison.csv', index=False)

# Saving the best model (e.g., Random Forest)

import joblib

joblib.dump(best\_rf, 'house\_price\_forecasting\_model.pkl')