[1]

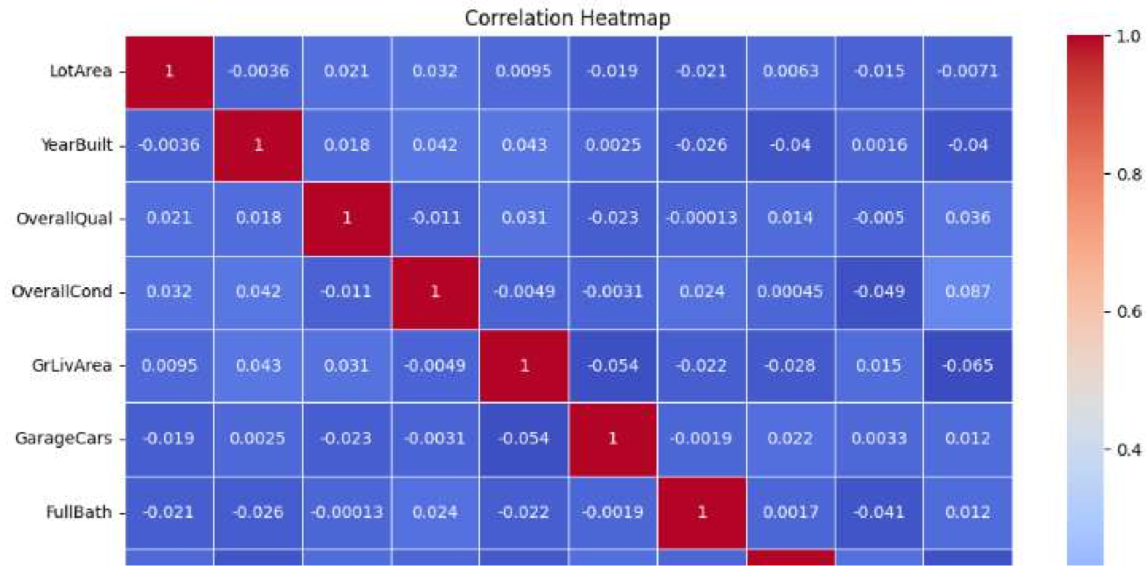
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import warnings
```

```python
# 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())
```

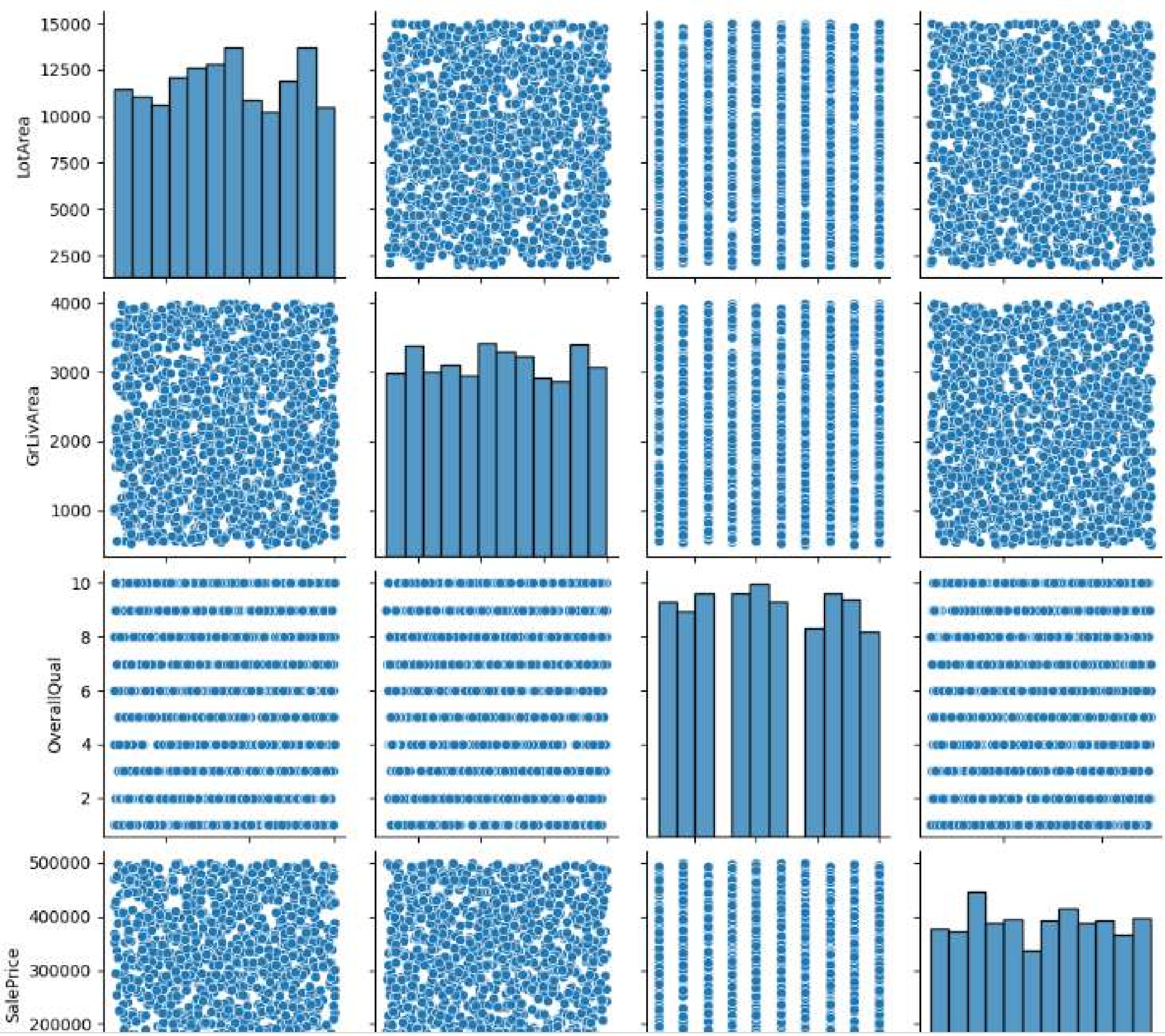
```
4    13964       1937            7            4       835           0

   FullBath  BedroomAbvGr  TotRmsAbvGrd Neighborhood  SalePrice
0         2             1            10      Mitchel     125955
1         2             5             3      CollgCr     496164
2         2             3             5      CollgCr     197748
3         3             4            10      CollgCr     144791
4         3             3             5      Crawfor     212586
            LotArea    YearBuilt  OverallQual  OverallCond    GrLivArea  \
count   1500.000000  1500.000000  1500.000000  1500.000000  1500.000000
mean    8542.969333  1961.626000     5.434667     5.546667  2247.217333
std     3678.189137    36.398828     2.837681     2.872396  1004.421250
min     2004.000000  1900.000000     1.000000     1.000000   501.000000
25%     5449.750000  1930.000000     3.000000     3.000000  1388.000000
50%     8510.000000  1961.000000     5.000000     6.000000  2237.000000
75%    11818.500000  1994.000000     8.000000     8.000000  3106.500000
max    14999.000000  2024.000000    10.000000    10.000000  3996.000000

        GarageCars     FullBath  BedroomAbvGr  TotRmsAbvGrd      SalePrice
count  1500.000000  1500.000000   1500.000000   1500.000000    1500.000000
mean      2.007333     1.488667      2.945333      8.291333  274531.491333
std       1.410179     1.112368      1.444900      3.725369  129507.074844
min       0.000000     0.000000      1.000000      2.000000   50005.000000
25%       1.000000     1.000000      2.000000      5.000000  160394.000000
50%       2.000000     1.000000      3.000000      8.000000  276643.500000
75%       3.000000     2.000000      4.000000     12.000000  386559.750000
max       4.000000     3.000000      5.000000     14.000000  499840.000000
LotArea         0
YearBuilt       0
OverallQual     0
OverallCond     0
GrLivArea       0
GarageCars      0
```

```python
# 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()
```

```
Non-numeric columns: Index(['Neighborhood'], dtype='object')
NaN values in correlation matrix: 0
```



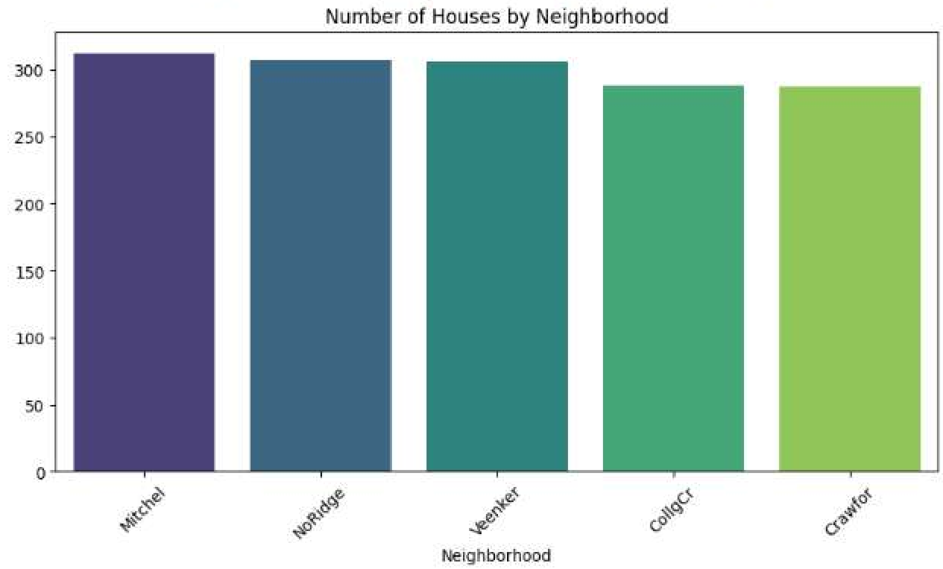
```python
# Pair Plot for Numerical Features
sns.pairplot(data[['LotArea', 'GrLivArea', 'OverallQual', 'SalePrice']])
plt.show()
```



```
[5] # 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()
```

```
<ipython-input-6-75d8882fea20>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=data['Neighborhood'].value_counts().index, y=data['Neighborhood'].value_counts().values, palette='viridis')
```



```python
[7] # ------------------------------
    # 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}")
```

```
Training data shape: (1200, 13)
Testing data shape: (300, 13)
```

```python
# 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}")
```

```
Linear Regression - MAE: 109271.12, MSE: 16306605846.49, RMSE: 127697.32, R2: -0.03
Ridge Regression - MAE: 109260.13, MSE: 16303634875.01, RMSE: 127685.69, R2: -0.03
Lasso Regression - MAE: 109269.88, MSE: 16306288572.55, RMSE: 127696.08, R2: -0.03
Random Forest - MAE: 112063.15, MSE: 17564257065.78, RMSE: 132530.21, R2: -0.11
Gradient Boosting - MAE: 110479.93, MSE: 17106530582.19, RMSE: 130791.94, R2: -0.08
```

```python
# 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}")
```

```
Best parameters for Random Forest: {'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 50}
Best cross-validation score: -17550218661.31
```

```python
# ------------------------------
# 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}")
```

```
Final Model - MAE: 111349.22, MSE: 17359478857.53, RMSE: 131755.38, R2: -0.09
Model saved as 'house_price_forecasting_model.pkl'
```