import Libraryes

In [2]:
```
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
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

Load the dataset

In [5]:
```
data = r"C:\Users\Hello\Desktop\AmesHousing.csv"
data = pd.read_csv(data)
```

Handle Missing Values

In [11]:
```
#Handle Missing Value
# Numerical columns: Fill with mean
num_cols = data.select_dtypes(include=['float64', 'int64']).columns
for col in num_cols:
    if data[col].isnull().sum() > 0:
        data[col].fillna(data[col].mean(), inplace=True)
```

```
C:\Users\Hello\AppData\Local\Temp\ipykernel_5892\2708537806.py:6: FutureWarning: A value is trying to be set on a cop
y of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which
we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or d
f[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data[col].fillna(data[col].mean(), inplace=True)
```

In [13]:
```
# Categorical columns: Fill with mode
cat_cols = data.select_dtypes(include=['object']).columns
for col in cat_cols:
```

```
    if data[col].isnull().sum() > 0:
        data[col].fillna(data[col].mode()[0], inplace=True)
```

```
C:\Users\Hello\AppData\Local\Temp\ipykernel_5892\2302232173.py:5: FutureWarning: A value is trying to be set on a cop
y of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which
we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or d
f[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  data[col].fillna(data[col].mode()[0], inplace=True)
```

Remove duplicates

In [15]:
```
# Remove duplicates
data = data.drop_duplicates()
```

 # Convert categorical variables to numerical

In [19]:
```
# Convert categorical variables to numerical
data = pd.get_dummies(data, drop_first=True)
```

In [21]:
```
# Define features and target
X = data[['Lot Area']]  # Example: Selecting a single feature
y = data['SalePrice']
```

In [23]:
```
# Step 3: Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [25]:
```
# Step 4: Build and train the linear regression model
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[25]:

▾ LinearRegression

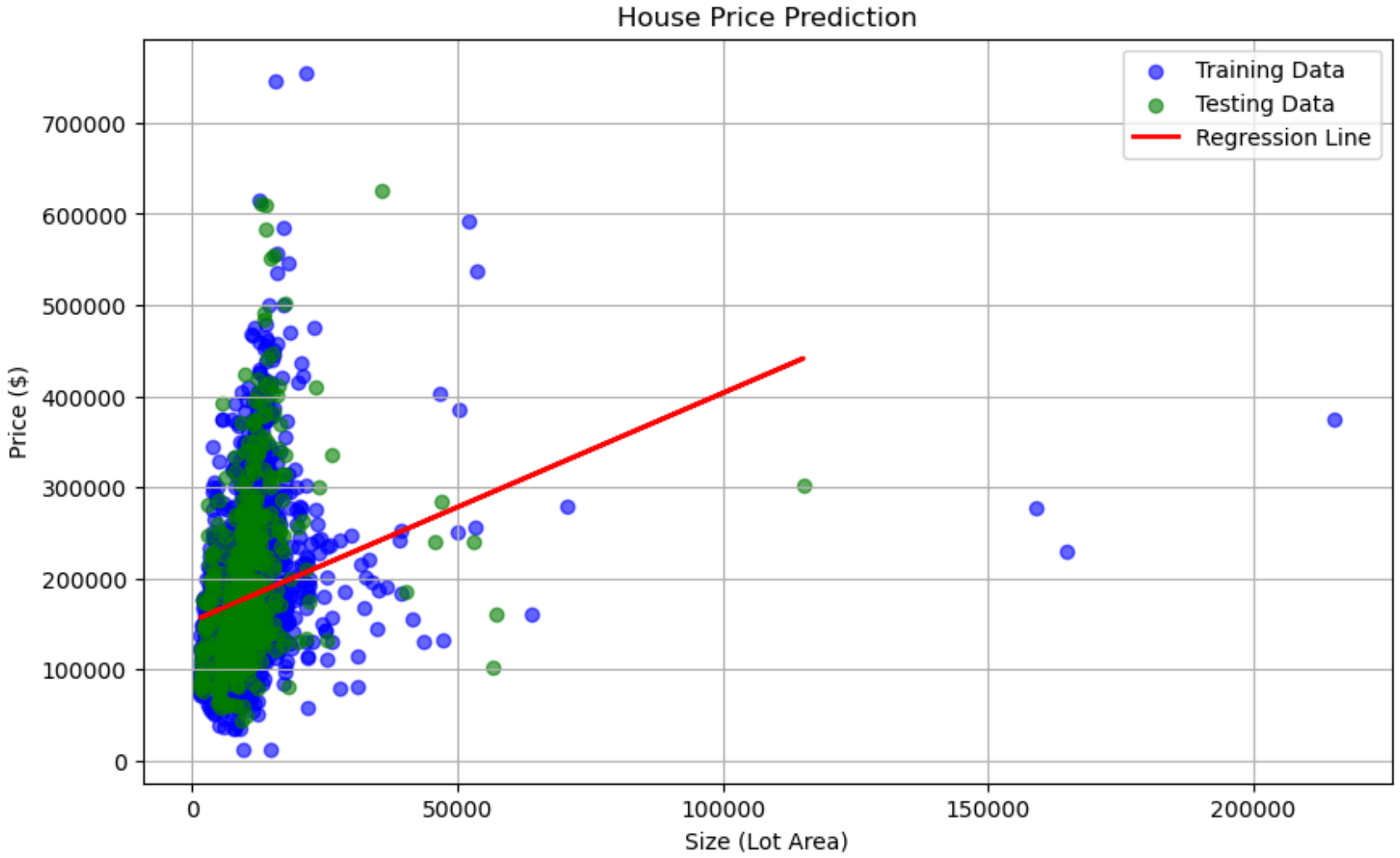
LinearRegression()

In [28]:
```
# Step 5: Make predictions on the test set
predictions = model.predict(X_test)
```

In [30]:
```
# Step 6: Evaluate the model
print("\nModel Performance:")
print("MAE:", mean_absolute_error(y_test, predictions))
print("MSE:", mean_squared_error(y_test, predictions))
print("R-squared:", r2_score(y_test, predictions))
```

```
Model Performance:
MAE: 62056.86000101161
MSE: 7509189795.222837
R-squared: 0.06340568713349304
```

In [32]:
```
# Step 7: Visualize the results
plt.figure(figsize=(10, 6))
# Plot training data
plt.scatter(X_train, y_train, color='blue', label='Training Data', alpha=0.6)
# Plot testing data
plt.scatter(X_test, y_test, color='green', label='Testing Data', alpha=0.6)
# Plot regression line
plt.plot(X_test, predictions, color='red', linewidth=2, label='Regression Line')
plt.title('House Price Prediction')
plt.xlabel('Size (Lot Area)')
plt.ylabel('Price ($)')
plt.legend()
plt.grid()
plt.show()
```


House Price Prediction
Training Data
Testing Data
Regression Line
Price ($)
Size (Lot Area)
0
100000
200000
300000
400000
500000
600000
700000
0
50000
100000
150000
200000

In [ ]: