In [2]:
```python
import pandas as pd
import numpy as np
from scipy.stats import describe
from scipy.stats import ttest_ind
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```python
df = pd.read_csv("housing_price_dataset.csv")
df.head()
```

Out[3]:

	SquareFeet	Bedrooms	Bathrooms	Neighborhood	YearBuilt	Price
0	2126	4	1	Rural	1969	215355.283618
1	2459	3	2	Rural	1980	195014.221626
2	1860	2	1	Suburb	1970	306891.012076
3	2294	2	1	Urban	1996	206786.787153
4	2130	5	2	Suburb	2001	272436.239065

In [4]:
```python
df.shape
```

Out[4]: (50000, 6)

In [5]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   SquareFeet    50000 non-null  int64  
 1   Bedrooms      50000 non-null  int64  
 2   Bathrooms     50000 non-null  int64  
 3   Neighborhood  50000 non-null  object 
 4   YearBuilt     50000 non-null  int64  
 5   Price         50000 non-null  float64
dtypes: float64(1), int64(4), object(1)
memory usage: 2.3+ MB
```

In [6]:
```python
df.duplicated().sum()
```

Out[6]: 0

In [7]:
```python
df.describe()
```

Out[7]:

	SquareFeet	Bedrooms	Bathrooms	YearBuilt	Price
count	50000.000000	50000.000000	50000.000000	50000.000000	50000.000000
mean	2006.374680	3.498700	1.995420	1985.404420	224827.325151
std	575.513241	1.116326	0.815851	20.719377	76141.842966
min	1000.000000	2.000000	1.000000	1950.000000	-36588.165397
25%	1513.000000	3.000000	1.000000	1967.000000	169955.860225
50%	2007.000000	3.000000	2.000000	1985.000000	225052.141166
75%	2506.000000	4.000000	3.000000	2003.000000	279373.630052
max	2999.000000	5.000000	3.000000	2021.000000	492195.259972

In [8]:

```
#Lets drop the negative values in price
df.drop(df[df.Price <= 0].index, axis=0, inplace=True)
```

In [9]:

```
df.describe()
```

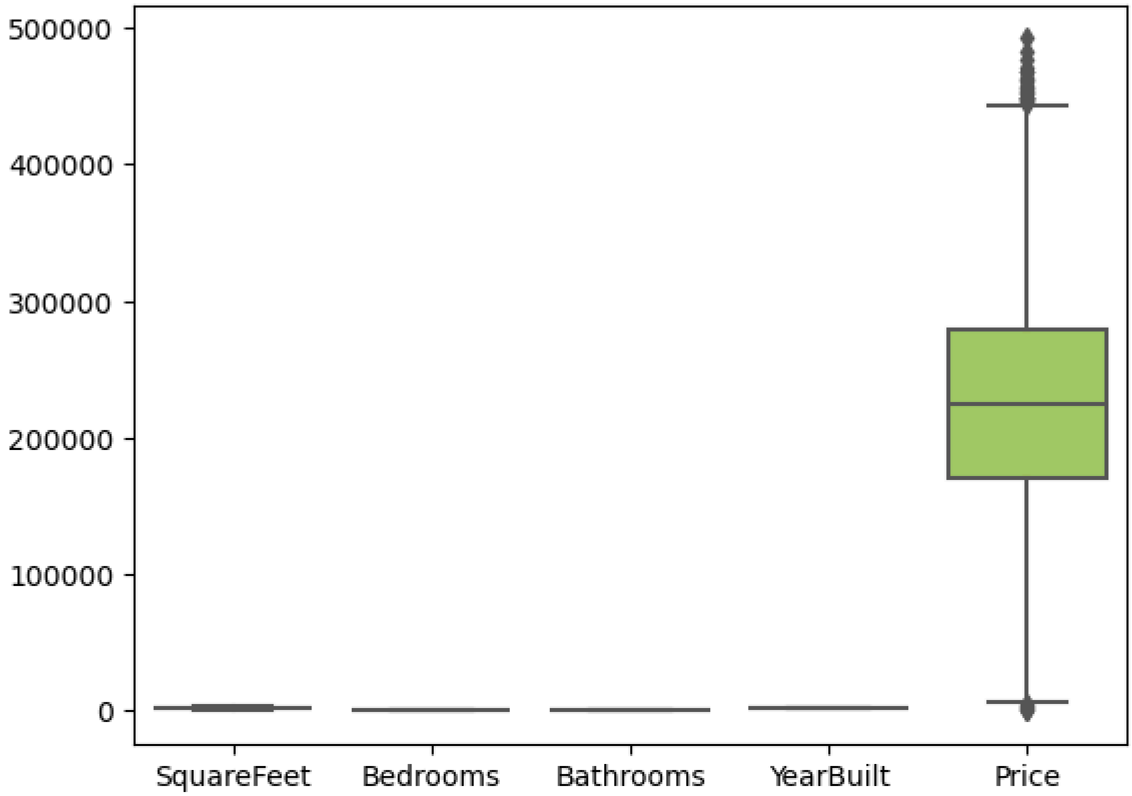
Out[9]:

	SquareFeet	Bedrooms	Bathrooms	YearBuilt	Price
count	49978.000000	49978.000000	49978.000000	49978.000000	49978.000000
mean	2006.752551	3.498659	1.995458	1985.404338	224931.667960
std	575.350298	1.116325	0.815859	20.718407	75995.682992
min	1000.000000	2.000000	1.000000	1950.000000	154.779120
25%	1514.000000	3.000000	1.000000	1967.000000	170007.487130
50%	2008.000000	3.000000	2.000000	1985.000000	225100.123857
75%	2506.000000	4.000000	3.000000	2003.000000	279395.826288
max	2999.000000	5.000000	3.000000	2021.000000	492195.259972

In [10]:

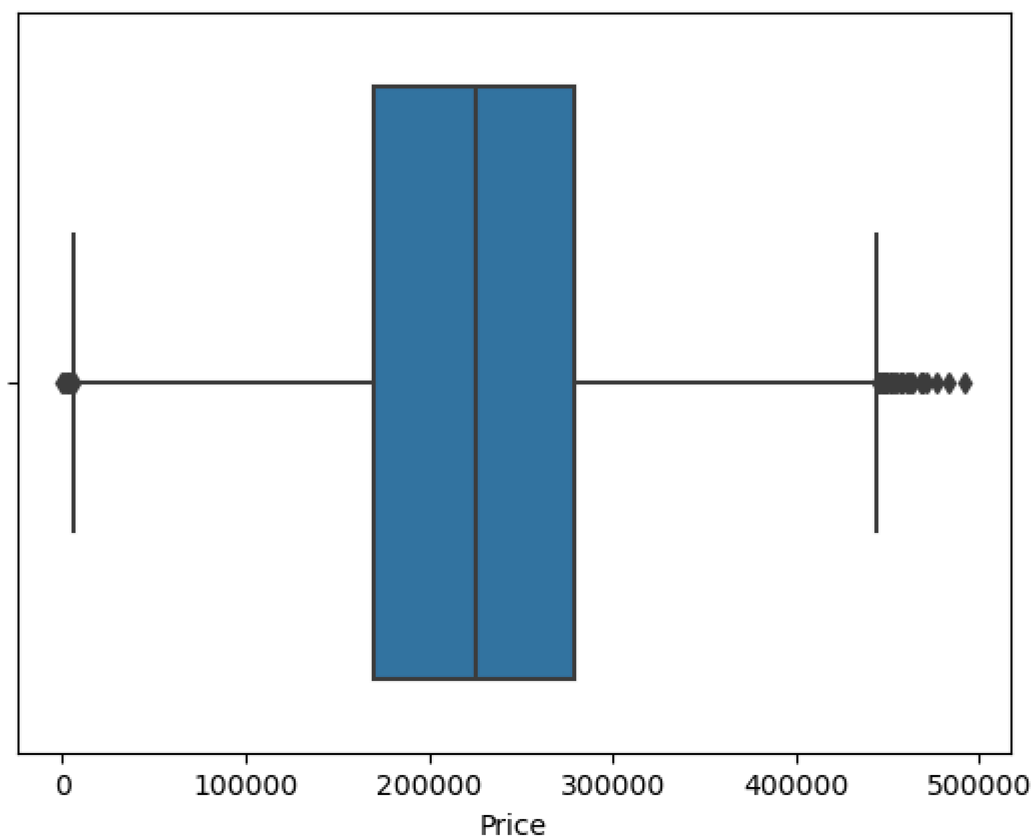
```
# Use the boxplot function to create boxplots for each column to check outliers
sns.boxplot(data=df, palette="Set2")
```

Out[10]: <Axes: >



In [11]:
```
sns.boxplot(x=df["Price"])
```

Out[11]: <Axes: xlabel='Price'>



In [12]:
```python
percentile_25 = df["Price"].quantile(0.25)
percentile_75 = df["Price"].quantile(0.75)
```

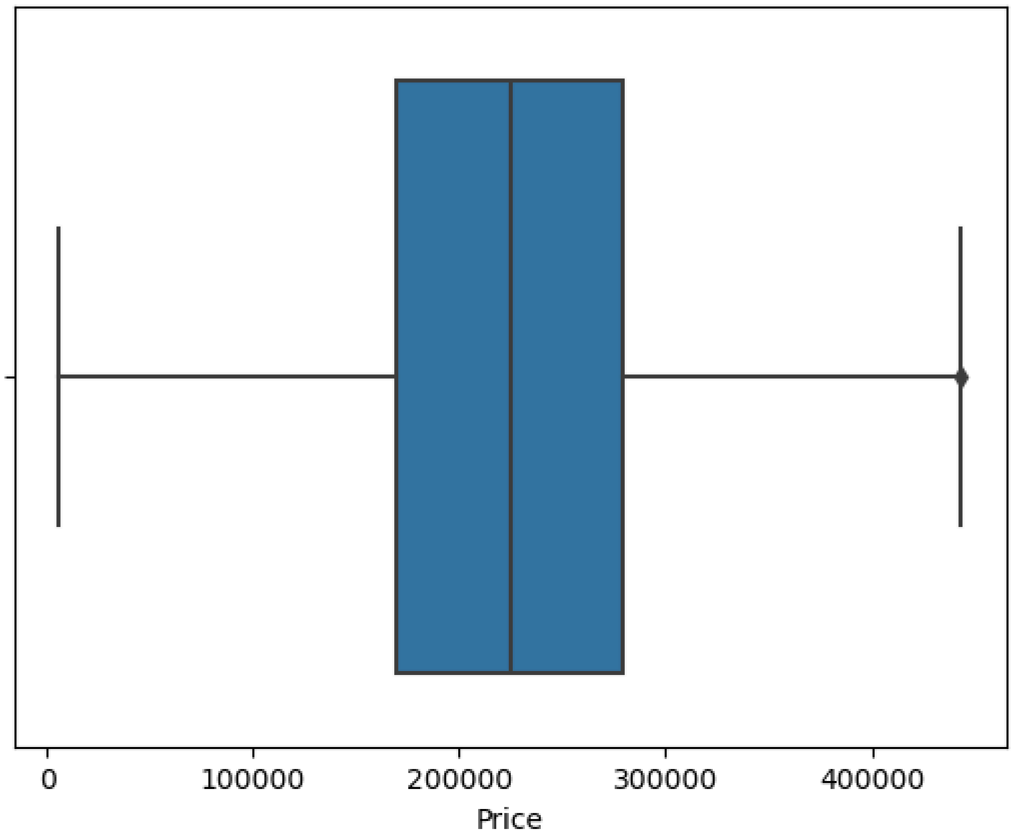
In [13]:
```python
iqr = percentile_75 - percentile_25
```

In [14]:
```python
upper_limit = percentile_75 + 1.5 * iqr
lower_limit = percentile_25 - 1.5 * iqr
```

In [15]:
```python
df_filtered = df[(df["Price"] >= lower_limit) & (df["Price"] <= upper_limit)]
```

In [16]:
```python
sns.boxplot(x=df_filtered["Price"])
```

Out[16]: <Axes: xlabel='Price'>



In [17]:
```python
df_corr = df_filtered[['SquareFeet', 'Bedrooms', 'Bathrooms', 'YearBuilt',
       'Price']].corr().style.background_gradient()
df_corr
```

Out[17]:

	SquareFeet	Bedrooms	Bathrooms	YearBuilt	Price
SquareFeet	1.000000	-0.002925	-0.003680	0.000563	0.750462
Bedrooms	-0.002925	1.000000	0.007612	0.003204	0.072500
Bathrooms	-0.003680	0.007612	1.000000	0.003882	0.027849
YearBuilt	0.000563	0.003204	0.003882	1.000000	-0.002035
Price	0.750462	0.072500	0.027849	-0.002035	1.000000

In [18]:

```
group_neigh_price = df.groupby('Neighborhood')['Price']\
.sum().reset_index(name = "total_Price").\
style.background_gradient(axis=0, cmap='YlOrRd')
group_neigh_price
```

Out[18]:

	Neighborhood	total_Price
0	Rural	3737121833.865258
1	Suburb	3732732412.854551
2	Urban	3771780654.564462

In [19]:

```
# Separate the numeric values based on the categorical column
category_A = df_filtered[df_filtered['Neighborhood'] == 'Rural']['Price']
category_B = df_filtered[df_filtered['Neighborhood'] == 'Suburb']['Price']
category_C = df_filtered[df_filtered['Neighborhood'] == 'Urban']['Price']

# Perform t-tests between pairs of categories
t_stat_AB, p_value_AB = ttest_ind(category_A, category_B)
t_stat_AC, p_value_AC = ttest_ind(category_A, category_C)
t_stat_BC, p_value_BC = ttest_ind(category_B, category_C)

# Display the results
print(f"T-test between A and B - T-statistic: {t_stat_AB}, p-value: {p_value_AB}")
print(f"T-test between A and C - T-statistic: {t_stat_AC}, p-value: {p_value_AC}")
print(f"T-test between B and C - T-statistic: {t_stat_BC}, p-value: {p_value_BC}")
```

```
T-test between A and B - T-statistic: 1.1247634796805202, p-value: 0.260697366925369
3
T-test between A and C - T-statistic: -3.6674915259655267, p-value: 0.00024532358423
37314
T-test between B and C - T-statistic: -4.790506458939784, p-value: 1.670767028181248
3e-06
```

In [20]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from statsmodels.formula.api import ols
import statsmodels.api as sm
```

In [21]:

```
df_filtered.Neighborhood = LabelEncoder().fit_transform(df_filtered.Neighborhood) #
df_filtered.head()
```

Out[21]:

	SquareFeet	Bedrooms	Bathrooms	Neighborhood	YearBuilt	Price
0	2126	4	1	0	1969	215355.283618
1	2459	3	2	0	1980	195014.221626
2	1860	2	1	1	1970	306891.012076
3	2294	2	1	2	1996	206786.787153
4	2130	5	2	1	2001	272436.239065

In [22]:

```
scale = StandardScaler()
df_filtered[['SquareFeet',
             'Bedrooms','Bathrooms',
             'Neighborhood',"YearBuilt"]]=scale.fit_transform(df_filtered[['SquareF
```

In [23]:

```
df_filtered.head()
```

Out[23]:

	SquareFeet	Bedrooms	Bathrooms	Neighborhood	YearBuilt	Price
0	0.208047	0.449164	-1.220217	-1.223784	-0.791795	215355.283618
1	0.787131	-0.446707	0.005522	-1.223784	-0.260912	195014.221626
2	-0.254524	-1.342578	-1.220217	0.001939	-0.743533	306891.012076
3	0.500197	-1.342578	-1.220217	1.227662	0.511281	206786.787153
4	0.215003	1.345036	0.005522	0.001939	0.752591	272436.239065

In [28]:

```
X = df_filtered[['SquareFeet', 'Bedrooms', 'Bathrooms','YearBuilt']]
y = df_filtered['Price']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Ordinary Least Squares (OLS) Multiple Regression using statsmodels
X_train = sm.add_constant(X_train)
model = sm.OLS(y_train, X_train).fit()


# Make predictions on the testing set
X_test = sm.add_constant(X_test)
y_pred = model.predict(X_test)

# Evaluate the model using metrics (e.g., Mean Squared Error)
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error on Test Set: {mse}')

# Normalize MSE by the range of the target variable
target_range = np.max(y) - np.min(y)
normalized_mse = mse / target_range
```

```python
#print(f'Mean Squared Error (MSE): {mse}')
print(f'Normalized MSE: {normalized_mse}')

# Print the summary to see the coefficients and statistics
print(model.summary())
```

```
Mean Squared Error on Test Set: 2468764993.7854753
Normalized MSE: 5646.615442253679
                            OLS Regression Results
==============================================================================
Dep. Variable:                  Price   R-squared:                       0.570
Model:                            OLS   Adj. R-squared:                  0.570
Method:                 Least Squares   F-statistic:                 1.326e+04
Date:                Tue, 02 Jan 2024   Prob (F-statistic):               0.00
Time:                        10:52:49   Log-Likelihood:            -4.8872e+05
No. Observations:               39952   AIC:                         9.775e+05
Df Residuals:                   39947   BIC:                         9.775e+05
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2.246e+05    248.690    903.318      0.000    2.24e+05    2.25e+05
SquareFeet  5.687e+04    248.351    228.991      0.000    5.64e+04    5.74e+04
Bedrooms    5726.9052    248.928     23.006      0.000    5239.000    6214.810
Bathrooms   2489.7879    248.777     10.008      0.000    2002.179    2977.397
YearBuilt    109.2635    248.610      0.439      0.660    -378.018     596.545
==============================================================================
Omnibus:                       11.631   Durbin-Watson:                   1.990
Prob(Omnibus):                  0.003   Jarque-Bera (JB):               10.904
Skew:                           0.011   Prob(JB):                      0.00429
Kurtosis:                       2.922   Cond. No.                         1.01
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spe
cified.
```

The output you've provided appears to be from the summary table of a linear regression model, and it seems like the model includes variables such as SquareFeet, Bedrooms, Bathrooms, and YearBuilt. Let's interpret the coefficients and associated statistics:

- Coefficients: Intercept (const):

- Coefficient (2.246e+05): This is the estimated value of the dependent variable when all other independent variables are zero (SquareFeet, Bedrooms, Bathrooms, and YearBuilt). In this context, it represents the estimated value when SquareFeet, Bedrooms, Bathrooms, and YearBuilt are all zero. Standard Error (248.690): The standard error measures the variability or uncertainty in the estimate of the coefficient.

- t-value (903.318): The t-value is the coefficient divided by its standard error. It measures how many standard deviations the coefficient is away from zero.

- p-value (0.000): The p-value tests the null hypothesis that the coefficient is equal to zero. A p-value less than the significance level (commonly 0.05) suggests that the coefficient is statistically significant. 95% Confidence Interval: [2.24e+05, 2.25e+05] This interval provides a range within which we are 95% confident that the true population parameter lies. SquareFeet:

- Coefficient (5.687e+04): For every additional square foot, the estimated value of the dependent variable increases by 5.687e+04, assuming Bedrooms, Bathrooms, and YearBuilt are held constant. Standard Error (248.351): t-value (228.991): p-value (0.000): 95% Confidence Interval: [5.64e+04, 5.74e+04] Bedrooms:

- Coefficient (5726.9052): For every additional bedroom, the estimated value of the dependent variable increases by 5726.9052, assuming SquareFeet, Bathrooms, and YearBuilt are held constant. Standard Error (248.928): t-value (23.006): p-value (0.000): 95% Confidence Interval: [5239.000, 6214.810] Bathrooms:

- Coefficient (2489.7879): For every additional bathroom, the estimated value of the dependent variable increases by 2489.7879, assuming SquareFeet, Bedrooms, and YearBuilt are held constant. Standard Error (248.777): t-value (10.008): p-value (0.000): 95% Confidence Interval: [2002.179, 2977.397] YearBuilt:

- Coefficient (109.2635): The coefficient suggests a positive effect on the dependent variable for each additional unit of the YearBuilt variable. However, the p-value (0.660) is relatively high, indicating that YearBuilt may not be statistically significant at a common significance level of 0.05.

- Summary: The intercept represents the estimated value of the dependent variable when all independent variables are zero. Interpretation of the intercept may be limited based on the nature of the variables in your model. SquareFeet, Bedrooms, and Bathrooms have statistically significant coefficients with very low p-values, suggesting a strong relationship with the dependent variable. YearBuilt has a positive coefficient but a higher p-value, indicating that its relationship with the dependent variable may not be statistically significant in this model. The 95% confidence intervals provide a range within which we are reasonably confident that the true coefficients lie.

In [25]:
```
# Create a linear regression model
model = LinearRegression()

# Fit the model
model.fit(X_train, y_train)

# Print the coefficients and intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
```

```
Coefficients: [     0.          56870.26191167   5726.90519436   2489.78790888
   109.26347004]
Intercept: 224646.29368855603
```

In [26]:
```python
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Assuming X is your design matrix (independent variables)
vif_data = pd.DataFrame()
vif_data[['SquareFeet', 'Bedrooms', 'Bathrooms','YearBuilt']] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1]
vif_data["Tolerance"] = 1 / vif_data["VIF"]

# Variance proportions
vif_data["Variance Proportion"] = 1 - vif_data["Tolerance"]

print(vif_data)
```

```
  SquareFeet  Bedrooms  Bathrooms  YearBuilt       VIF  Tolerance  \
0        NaN       NaN        NaN        NaN  1.000022   0.999978
1        NaN       NaN        NaN        NaN  1.000076   0.999924
2        NaN       NaN        NaN        NaN  1.000086   0.999914
3        NaN       NaN        NaN        NaN  1.000025   0.999975

   Variance Proportion
0             0.000022
1             0.000076
2             0.000086
3             0.000025
```

VIF Interpretation: SquareFeet:

VIF: 1.000022 Tolerance: 0.999978 Variance Proportion: 0.000022 Bedrooms:

VIF: 1.000076 Tolerance: 0.999924 Variance Proportion: 0.000076 Bathrooms:

VIF: 1.000086 Tolerance: 0.999914 Variance Proportion: 0.000086 YearBuilt:

VIF: 1.000025 Tolerance: 0.999975 Variance Proportion: 0.000025 Interpretation: VIF Values:

All VIF values are very close to 1. This suggests that there is minimal multicollinearity among the independent variables. Typically, VIF values below 5 indicate low multicollinearity. Tolerance Values:

Tolerance is the inverse of VIF. All tolerance values are very close to 1, confirming the low level of multicollinearity. Variance Proportion:

The variance proportion for each variable is very close to 0. This indicates that each variable explains a very small proportion of the variance that it does not share with the other variables. Summary: Based on the VIF values, tolerance, and variance proportion, there is no evidence of problematic multicollinearity among the independent variables (SquareFeet, Bedrooms, Bathrooms, and YearBuilt). The values are well below the commonly used threshold of 5 for VIF, suggesting that the variables are not highly correlated with each other.

This is a positive result as low multicollinearity enhances the stability and reliability of the regression coefficients. It indicates that each variable provides unique information in explaining the variance of the dependent variable without redundancy from other variables.

In summary, the VIF results suggest that the model does not suffer from multicollinearity issues among the included independent variables.

In [32]:

```
X
```

Out[32]:

	SquareFeet	Bedrooms	Bathrooms	YearBuilt
0	0.208047	0.449164	-1.220217	-0.791795
1	0.787131	-0.446707	0.005522	-0.260912
2	-0.254524	-1.342578	-1.220217	-0.743533
3	0.500197	-1.342578	-1.220217	0.511281
4	0.215003	1.345036	0.005522	0.752591
...	...	...	...	...
49995	-1.259661	1.345036	1.231262	-0.502223
49996	1.474032	-1.342578	0.005522	0.125184
49997	1.691406	1.345036	1.231262	-1.129630
49998	1.025372	1.345036	0.005522	-0.067864
49999	-0.755354	1.345036	1.231262	1.235212

49941 rows × 4 columns

In [ ]: