Predict Houses Price : Linear regression

Predict house prices based on various features such as the size of the house (in square feet), number of bedrooms, age of the house, and so on.

Tasks :

1- Import the Necessary Libraries:

Import pandas, numpy, train_test_split from sklearn.model_selection, LinearRegression from sklearn.linear_model, and mean_squared_error from sklearn.metrics.

2- Create the Dataset:

Create a dictionary with the following keys: Size (sq ft), Number of Bedrooms, Age of the House (years), Price ($). Or download House price Dataset from Kaggle.

Convert the dictionary into a pandas DataFrame.

3- Prepare the Data for Training:

Separate the DataFrame into features (X) and target (y). Split the data into training and testing sets using train_test_split.

4- Train the Linear Regression Model:

Create an instance of LinearRegression. Fit the model to the training data.

5- Make Predictions:

Use the trained model to make predictions on the test set.

6 - Evaluate the Model:

Calculate the mean squared error between the predicted prices and the actual prices on the test set. Print the mean squared error.

7 - Additional Suggestions

Visualize the Data: Create scatter plots to visualize the relationships between the features and the target variable.

Feature Scaling: Apply feature scaling if necessary.

More Features: If available, add more features such as the number of bathrooms, location, etc.

Model Evaluation: Use other metrics like the R^2 score to evaluate the model's performance.

In [59]:
```
#import librairies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler

#Dataset downloaded from Kaggle, prepared in CSV file

data = pd.read_csv("house_price.csv",
                   delimiter=',',
                   dtype={'size(sqft)': 'int', 'nbr_of__bedrooms': 'int', 'h
                   na_values=['NA', 'n/a'],
                   encoding='utf-8' )


# Handle missing values by filling with a default value
data.fillna(0, inplace=True)

data.head(20)

#Prepare data for training

X = data[['size(sqft)', 'nbr_of__bedrooms', 'house_age(years)']]
y = data['price($)']

#Split data into training and testing

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

#Train the linear regression model

model = LinearRegression()
model.fit(X_train, y_train)

#Make prediction

y_pred = model.predict(X_test)

#Evaluate the model

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

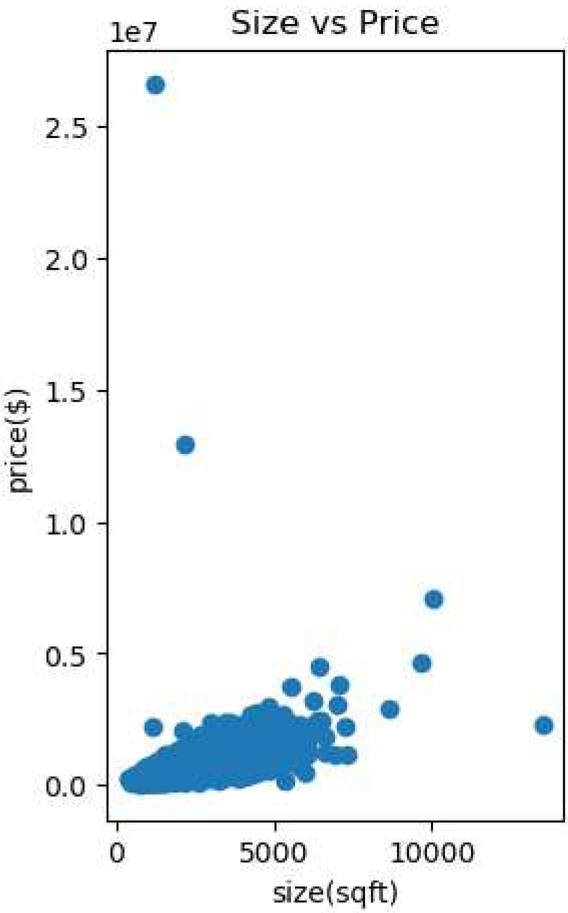
```
Mean Squared Error: 58156934686.6303
```

In [63]:
```
y
```

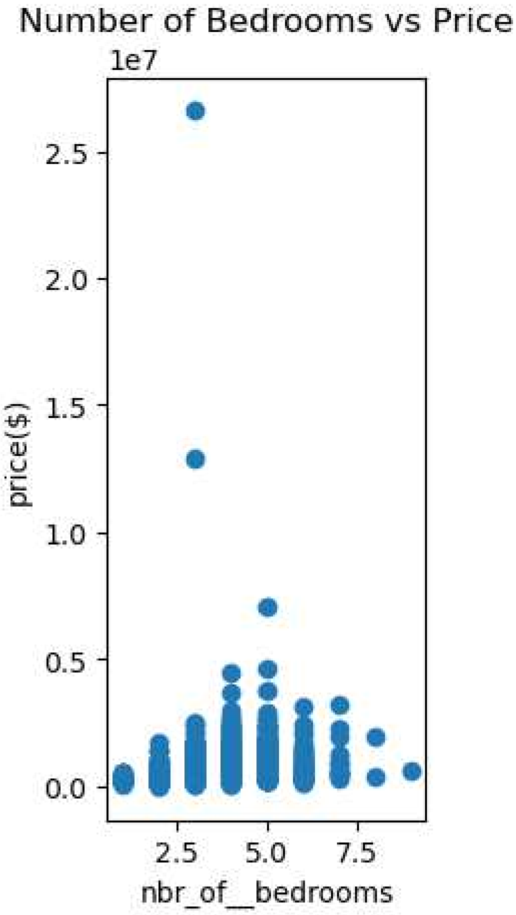
Out[63]:
```
0        276000.0
1        245000.0
2        280000.0
3         80000.0
4        150000.0
           ...
4541    1135250.0
4542    2888000.0
4543    4668000.0
4544    7062500.0
4545    2280000.0
Name: price($), Length: 4546, dtype: float64
```

In [21]:
```
# Scatter plot for Size vs Price
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.scatter(data['size(sqft)'], data['price($)'])
plt.title('Size vs Price')
plt.xlabel('size(sqft)')
plt.ylabel('price($)')
```

Out[21]:
```
Text(0, 0.5, 'price($)')
```



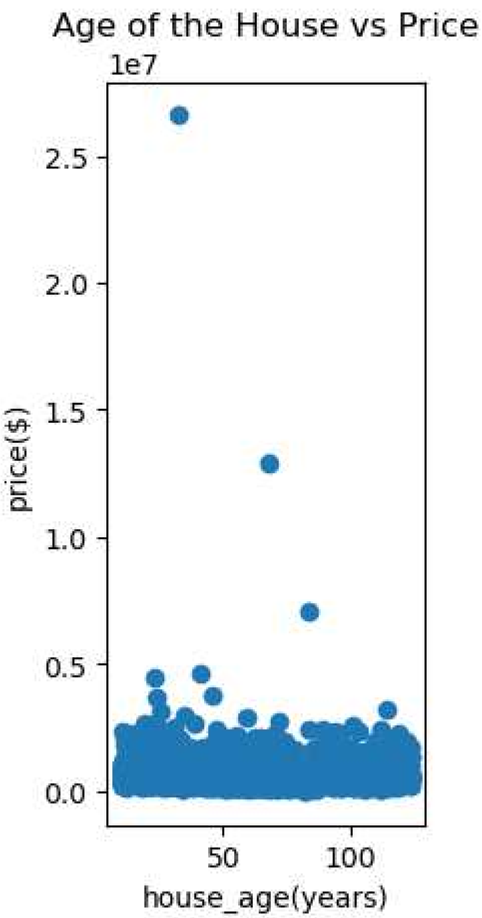
In [25]:
```
# Scatter plot for Number of Bedrooms vs Price
plt.subplot(1, 3, 2)
plt.scatter(data['nbr_of__bedrooms'], data['price($)'])
plt.title('Number of Bedrooms vs Price')
plt.xlabel('nbr_of__bedrooms')
plt.ylabel('price($)')
plt.tight_layout()
plt.show()
```



In [ ]:

In [24]:

```
# Scatter plot for Age of the House vs Price
plt.subplot(1, 3, 3)
plt.scatter(data['house_age(years)'], data['price($)'])
plt.title('Age of the House vs Price')
plt.xlabel('house_age(years)')
plt.ylabel('price($)')
plt.tight_layout()
plt.show()
```



In [31]:

```
#use the R² score to evaluate the model’s performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Squared Error with more features: {mse}")
print(f"R² Score with more features: {r2}")
```

```
Mean Squared Error with more features: 58156934686.6303
R² Score with more features: 0.5535337811431654
```

R^2 Score of 0.55: This means that your model explains approximately 55% of the variance in the house prices. This is a moderate level of explanatory power, but it indicates that there is still a substantial amount of variability (45%) in the house prices that the model does not capture.

The high MSE value suggests that the model's predictions are not very close to the actual values. This further supports the need for model improvement.

Model improvement

By implementing model improvements, we may be able to increase the R^2 score and decrease the MSE, leading to a better performing model.

In [33]:
```
#Improve the model Adding Polynomial Features

from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Create polynomial features
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

# Train the model with polynomial features
model_poly = LinearRegression()
model_poly.fit(X_poly, y)

# Make predictions and evaluate
y_pred_poly = model_poly.predict(poly.transform(X_test))
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)

print(f"Mean Squared Error with polynomial features: {mse_poly}")
print(f"R² Score with polynomial features: {r2_poly}")
```

```
Mean Squared Error with polynomial features: 65527339014.63083
R² Score with polynomial features: 0.4969517661263917
```

In [34]:
```
#improve thw model Using Ridge Regression

from sklearn.linear_model import Ridge

# Train the model with Ridge regression
model_ridge = Ridge(alpha=1.0)
model_ridge.fit(X_train, y_train)

# Make predictions and evaluate
y_pred_ridge = model_ridge.predict(X_test)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print(f"Mean Squared Error with Ridge regression: {mse_ridge}")
print(f"R² Score with Ridge regression: {r2_ridge}")
```

```
Mean Squared Error with Ridge regression: 58155754043.27591
R² Score with Ridge regression: 0.5535428448494473
```

In [53]:
```
#Model improvement by Feature Engineering:
#Implemeting additional features like location,property condition...

#Dataset downloaded from Kaggle, prepared in CSV file

data_featured = pd.read_csv("house-price-featured.csv",
                    delimiter=',',
                    dtype={'size(sqft)': 'int', 'nbr_of__bedrooms': 'int', 'h
                    na_values=['NA', 'n/a'],
                    encoding='utf-8' )
```

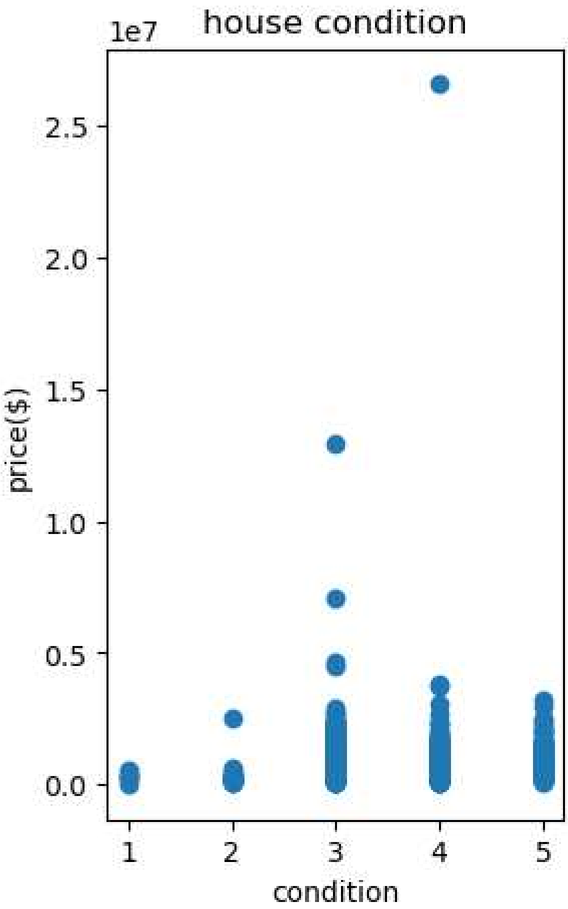
In [54]:
```
data_featured.head(20)
```

Out[54]:

	size(sqft)	nbr_of_bedrooms	house_age(years)	price($)	condition	city
0	2163	4	18	248000.0000	3	Algona
1	2020	4	22	262000.0000	3	Algona
2	1560	3	32	196440.0000	3	Algona
3	1390	3	64	230000.0000	4	Algona
4	910	2	68	100000.0000	3	Algona
5	2009	4	10	303210.0000	3	Auburn
6	2481	5	10	309000.0000	3	Auburn
7	2242	3	10	309780.0000	3	Auburn
8	2250	4	10	333490.0000	3	Auburn
9	2570	3	10	339990.0000	3	Auburn
10	2701	4	10	399895.0000	3	Auburn
11	2658	4	10	411605.0000	3	Auburn
12	2656	4	10	495000.0000	3	Auburn
13	2538	3	11	289373.3077	3	Auburn
14	2303	4	11	329995.0000	3	Auburn
15	1481	3	12	240000.0000	3	Auburn
16	1584	3	12	267000.0000	3	Auburn
17	1769	3	12	334888.0000	3	Auburn
18	1550	3	13	259000.0000	3	Auburn
19	2437	3	13	285000.0000	3	Auburn

In [58]:
```
# Scatter plot for Size vs Price
plt.figure(figsize=(10, 5))
plt.subplot(1, 3, 1)
plt.scatter(data_featured['condition'], data_featured['price($)'])
plt.title('house condition')
plt.xlabel('condition')
plt.ylabel('price($)')
```

Out[58]:
```
Text(0, 0.5, 'price($)')
```



In [48]:

```
#Prepare data for training

X = data_featured[['size(sqft)', 'nbr_of_bedrooms', 'house_age(years)', 'coı
y = data_featured['price($)']

#Split data into training and testing

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

#Train the linear regression model

model = LinearRegression()
model.fit(X_train, y_train)

#Make prediction

y_pred = model.predict(X_test)

#Evaluate the model

mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

#use the R² score to evaluate the model's performance

r2 = r2_score(y_test, y_pred)
print(f"R² Score with more features: {r2}")
```

```
Mean Squared Error: 230524089546.39798
R² Score with more features: 0.22052304188153138
```

In [49]:

```
#improve thw model Using Ridge Regression

from sklearn.linear_model import Ridge

# Train the model with Ridge regression
model_ridge = Ridge(alpha=1.0)
model_ridge.fit(X_train, y_train)

# Make predictions and evaluate
y_pred_ridge = model_ridge.predict(X_test)
mse_ridge = mean_squared_error(y_test, y_pred_ridge)
r2_ridge = r2_score(y_test, y_pred_ridge)

print(f"Mean Squared Error with Ridge regression: {mse_ridge}")
print(f"R² Score with Ridge regression: {r2_ridge}")
```

```
Mean Squared Error with Ridge regression: 230523385429.0518
R² Score with Ridge regression: 0.22052542273139508
```

In [50]:
```python
#Improve the model Adding Polynomial Features

from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline

# Create polynomial features
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X)

# Train the model with polynomial features
model_poly = LinearRegression()
model_poly.fit(X_poly, y)

# Make predictions and evaluate
y_pred_poly = model_poly.predict(poly.transform(X_test))
mse_poly = mean_squared_error(y_test, y_pred_poly)
r2_poly = r2_score(y_test, y_pred_poly)

print(f"Mean Squared Error with polynomial features: {mse_poly}")
print(f"R² Score with polynomial features: {r2_poly}")
```

```
Mean Squared Error with polynomial features: 229233034709.81216
R² Score with polynomial features: 0.2248885183866821
```

Adding more features has improved the MSE but decreased the r2!