Linear Regression with Python

** This is mostly just code for reference. Please learn the concept for more info behind all of this code.**

Your neighbor is a real estate agent and wants some help predicting housing prices for regions in the USA. It would be great if you could somehow create a model for her that allows her to put in a few features of a house and returns back an estimate of what the house would sell for.

She has asked you if you could help her out with your new data science skills. You say yes, and decide that Linear Regression might be a good path to solve this problem!

Your neighbor then gives you some information about a bunch of houses in regions of the United States,it is all in the data set: USA_Housing.csv.

The data contains the following columns:

- 'Avg. Area Income': Avg. Income of residents of the city house is located in.
- 'Avg. Area House Age': Avg Age of Houses in same city
- 'Avg. Area Number of Rooms': Avg Number of Rooms for Houses in same city
- 'Avg. Area Number of Bedrooms': Avg Number of Bedrooms for Houses in same city
- 'Area Population': Population of city house is located in
- 'Price': Price that the house sold at
- 'Address': Address for the house

Let's get started!

Check out the data

We've been able to get some data from your neighbor for housing prices as a csv set, let's get our environment ready with the libraries we'll need and then import the data!

Import Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Check out the Data

In [2]:
```
USAhousing = pd.read_csv('USA_Housing.csv')
```

In [3]:
```
USAhousing.head()
```

Out[3]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

In [4]:
```
USAhousing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

In [5]:
```
USAhousing.describe()
```

Out[5]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

In [6]:
```
USAhousing.columns
```

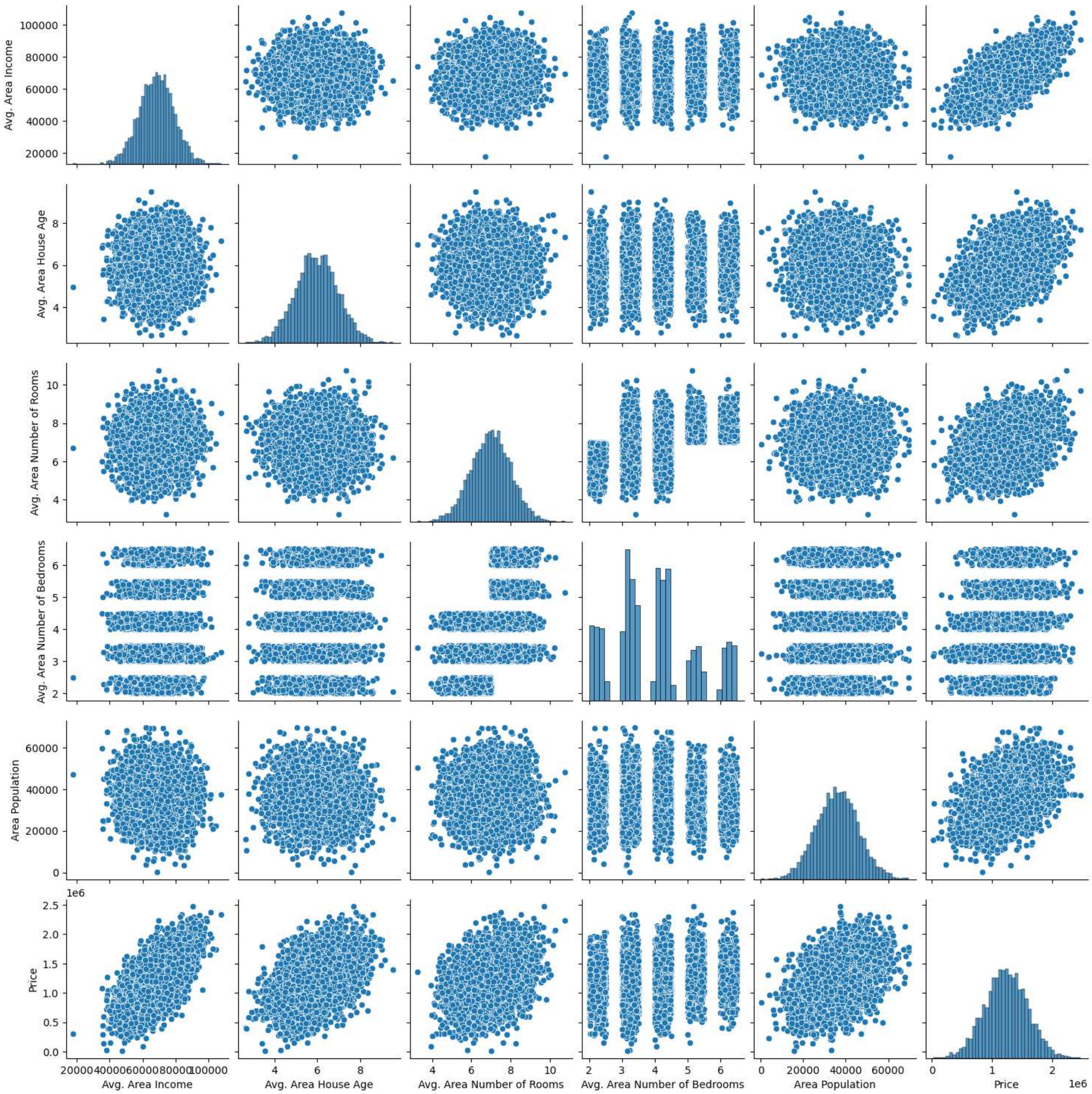
Out[6]:
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

Exploratory data analysis (EDA)

Let's create some simple plots to check out the data!

In [7]:
```
sns.pairplot(USAhousing)
```

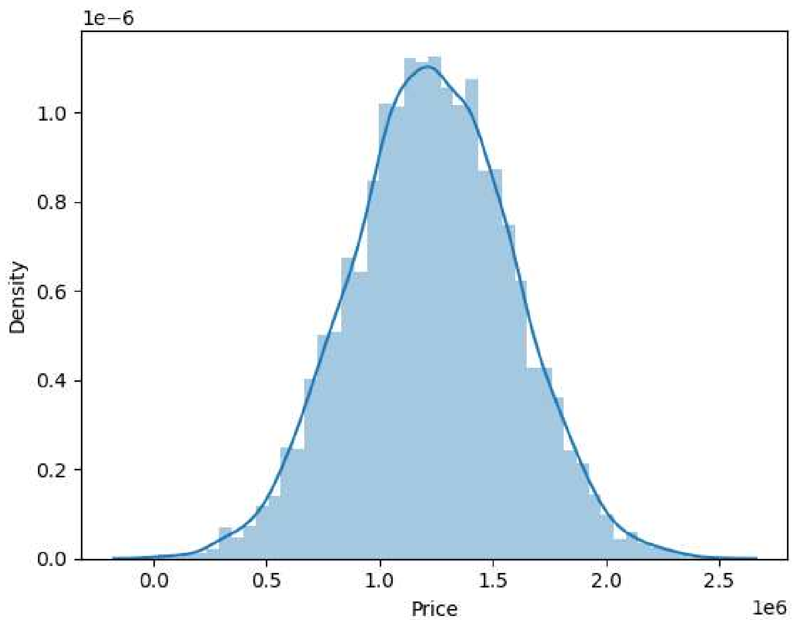
Out[7]: <seaborn.axisgrid.PairGrid at 0x2ada5802da0>



In [23]:
```
sns.distplot(USAhousing['Price'])
```

```
C:\Users\mahmud\AppData\Local\Temp\ipykernel_4576\812483608.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad63
72750bbe5751)

  sns.distplot(USAhousing['Price'])
```

Out[23]:
```
<Axes: xlabel='Price', ylabel='Density'>
```



In [28]:
```
USAhousing.columns
```

Out[28]:
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

In [29]:
```
house=USAhousing[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
                        'Avg. Area Number of Bedrooms', 'Area Population', 'Price']]
```

In [30]:
```
house
```

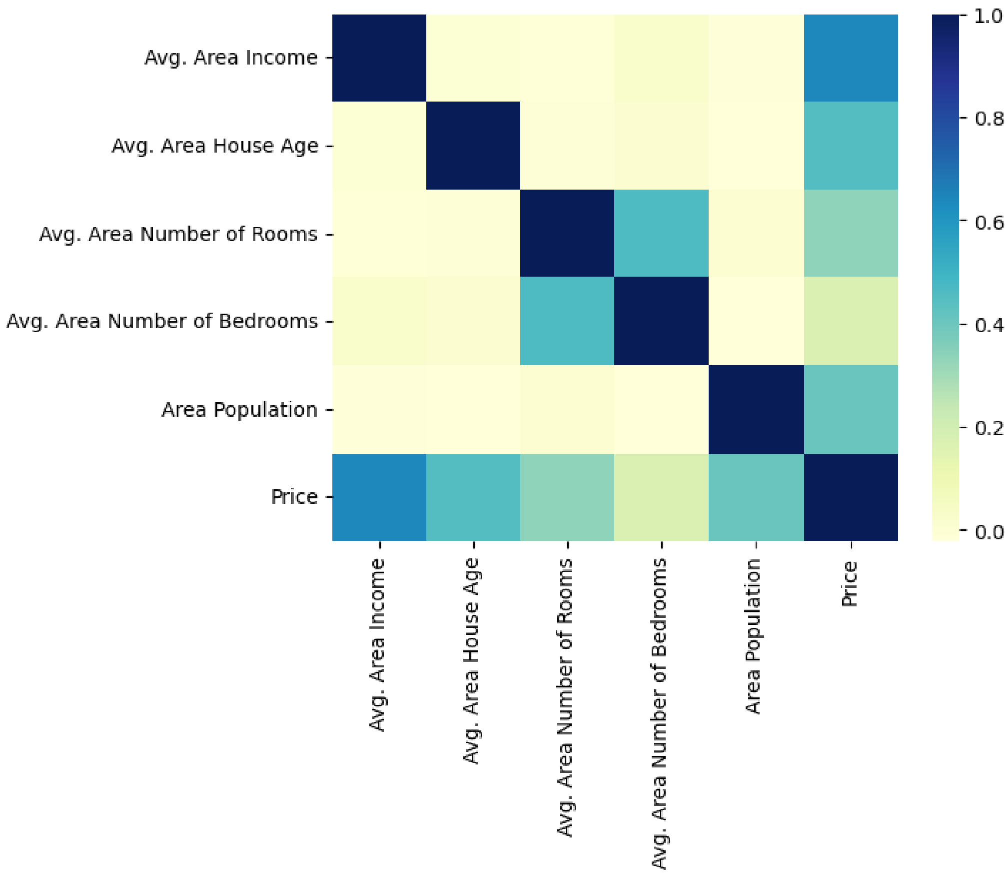
Out[30]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05
...	...	...	...	...	...	...
4995	60567.944140	7.830362	6.137356	3.46	22837.361035	1.060194e+06
4996	78491.275435	6.999135	6.576763	4.02	25616.115489	1.482618e+06
4997	63390.686886	7.250591	4.805081	2.13	33266.145490	1.030730e+06
4998	68001.331235	5.534388	7.130144	5.44	42625.620156	1.198657e+06
4999	65510.581804	5.992305	6.792336	4.07	46501.283803	1.298950e+06

5000 rows × 6 columns

In [33]:
```
sns.heatmap(house.corr(),cmap="YlGnBu")
```

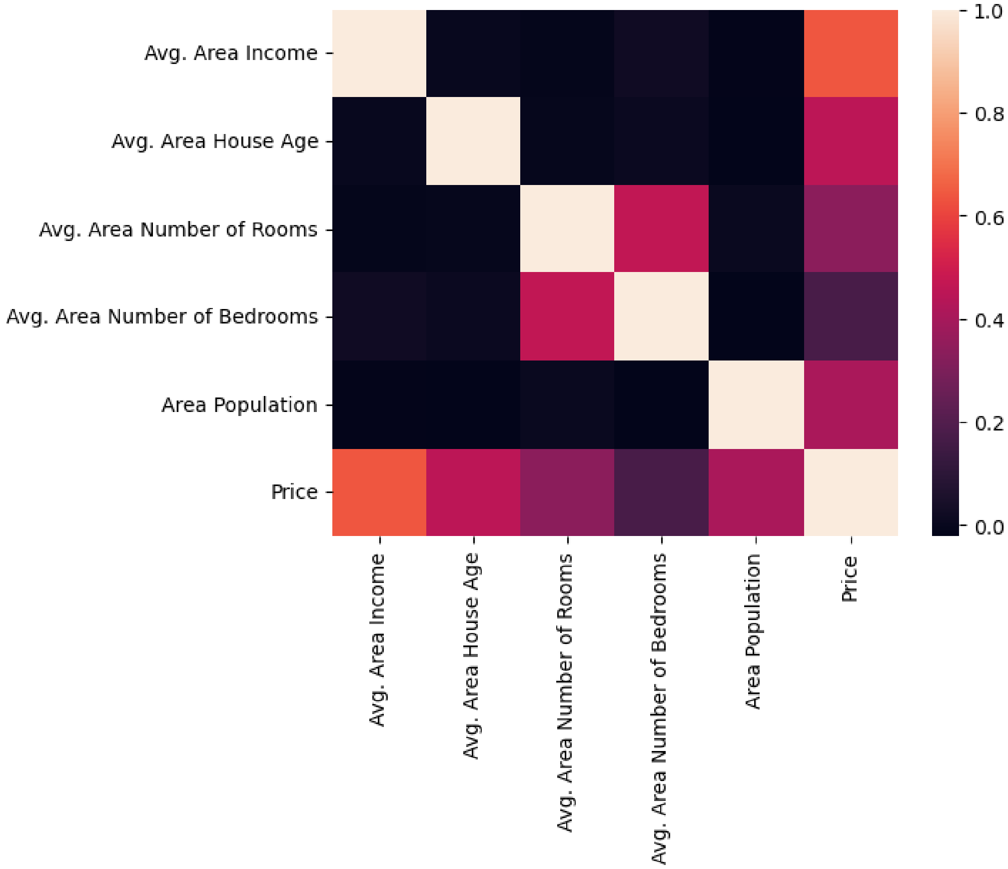
Out[33]: <Axes: >



In [9]:
```
sns.heatmap(USAhousing.corr())
```

```
C:\Users\mahmud\AppData\Local\Temp\ipykernel_4576\437206318.py:1: FutureWarning: The default value of numeric_only in
DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the
value of numeric_only to silence this warning.
  sns.heatmap(USAhousing.corr())
```

Out[9]: <Axes: >



Training a Linear Regression Model

Let's now begin to train out regression model! We will need to first split up our data into an X array that contains the features to train on, and a y array with the target variable, in this case the Price column. We will toss out the Address column because it only has text info that the linear regression model can't use.

X and y arrays

In [34]:
```
X = USAhousing[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
               'Avg. Area Number of Bedrooms', 'Area Population']]
y = USAhousing['Price']
```

Train Test Split

Now let's split the data into a training set and a testing set. We will train out model on the training set and then use the test set to evaluate the model.

In [35]:
```
from sklearn.model_selection import train_test_split
```

In [36]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=101)
```

Creating and Training the Model

In [37]:
```
from sklearn.linear_model import LinearRegression
```

In [38]:
```
lm = LinearRegression()
```

In [39]:
```
lm.fit(X_train,y_train)
```

Out[39]:
```
▾ LinearRegression
LinearRegression()
```

Model Evaluation

Let's evaluate the model by checking out it's coefficients and how we can interpret them.

In [40]:
```
# print the intercept
print(lm.intercept_)
```

```
-2640159.796851625
```

In [41]:
```
coeff_df = pd.DataFrame(lm.coef_,X.columns,columns=['Coefficient'])
coeff_df
```

Out[41]:

	Coefficient
Avg. Area Income	21.528276
Avg. Area House Age	164883.282027
Avg. Area Number of Rooms	122368.678027
Avg. Area Number of Bedrooms	2233.801864
Area Population	15.150420

Interpreting the coefficients:

- Holding all other features fixed, a 1 unit increase in **Avg. Area Income** is associated with an **increase of $21.52 **.
- Holding all other features fixed, a 1 unit increase in **Avg. Area House Age** is associated with an **increase of $164883.28 **.
- Holding all other features fixed, a 1 unit increase in **Avg. Area Number of Rooms** is associated with an **increase of $122368.67 **.
- Holding all other features fixed, a 1 unit increase in **Avg. Area Number of Bedrooms** is associated with an **increase of $2233.80 **.
- Holding all other features fixed, a 1 unit increase in **Area Population** is associated with an **increase of $15.15 **.

Does this make sense? Probably not because I made up this data. If you want real data to repeat this sort of analysis, check out the [boston dataset (http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html)](http://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_boston.html):

```
from sklearn.datasets import load_boston
boston = load_boston()
print(boston.DESCR)
boston_df = boston.data
```

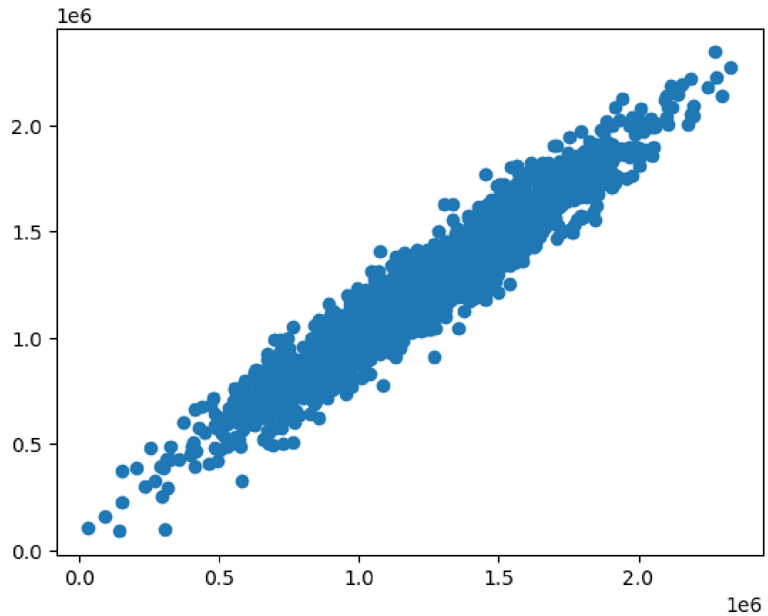
Predictions from our Model

Let's grab predictions off our test set and see how well it did!

In [42]:
```
predictions = lm.predict(X_test)
```

In [43]:
```
plt.scatter(y_test,predictions)
```

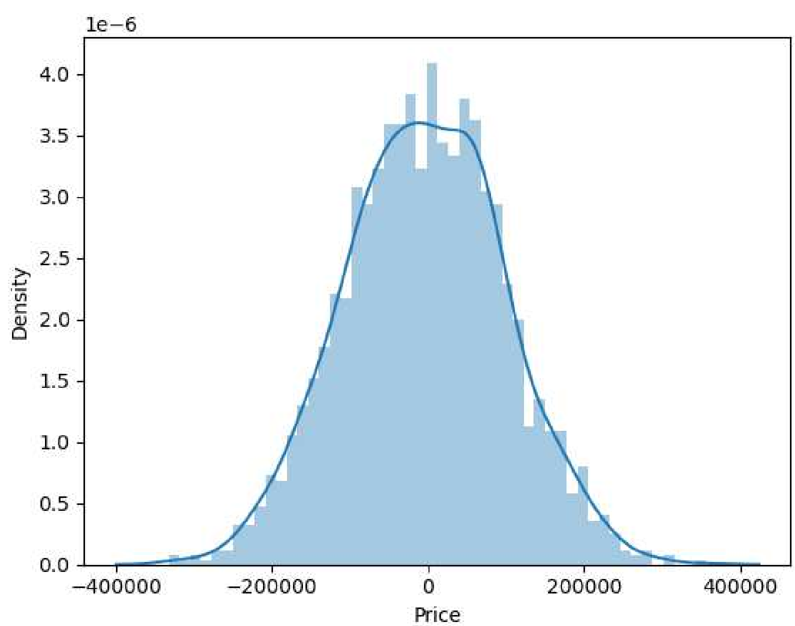
Out[43]: <matplotlib.collections.PathCollection at 0x2adab85e800>



Residual Histogram

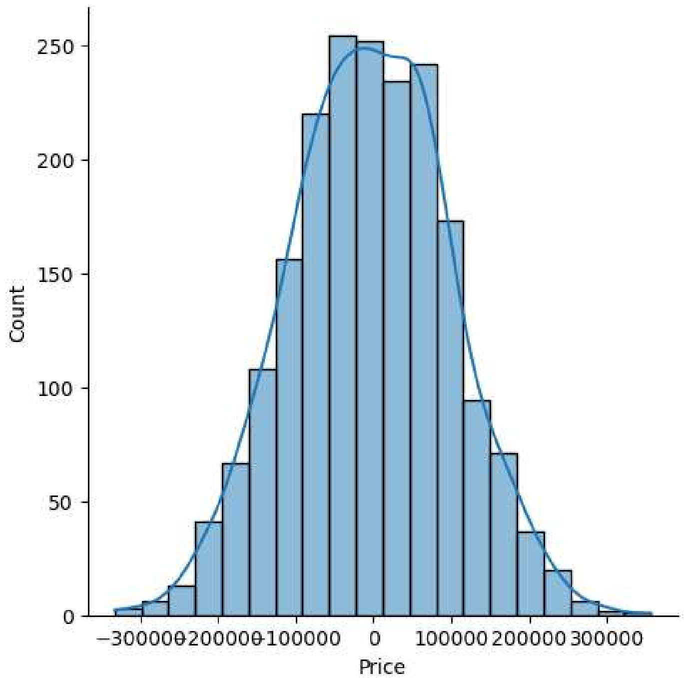
In [48]:
```
sns.distplot((y_test-predictions),bins=50);
```

```
C:\Users\mahmud\AppData\Local\Temp\ipykernel_4576\1326397652.py:1: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwaskom/de44147ed2974457ad63
72750bbe5751)

  sns.distplot((y_test-predictions),bins=50);
```



In [62]:
```
# if you choose bins equal to 50 then your input will be divided into 50 intervals or bins if possible
# kernel density estimate line, by passing kde=True

sns.displot((y_test-predictions),kde=True,bins=20)
```

Out[62]: <seaborn.axisgrid.FacetGrid at 0x2adaa738280>



Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

Mean Absolute Error (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Root Mean Squared Error (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

In [63]:
```
from sklearn import metrics
```

In [64]:
```
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 82288.2225191496
MSE: 10460958907.209692
RMSE: 102278.82922291246
```