The Art of Regression with Linear Regression

I know the title is HUGE but it's true. I'm Rishat and let's learn Linear Regression together.

- LinkedIn
- YouTube
- gtihub
- Gmail
- discord

A little Recap

In the previous article, I gave you small introduction of `machine learning` and what steps every machine learning model goes through and in this article we will go through the steps to build a `linear regression` model.

The steps are:

1. Get the `DATA`.
   - Define the `features` and the `target`.
2. Prepare/clean the `data`.
3. Split the `data` into `train` and `test` sets.
4. Train the `model`.
5. Evaluate the `model`.
6. Make `predictions`.

So, let's get started.

Get the Data

Now, the first step is to get the data and as this is the `first algorithm` of `machine learning`, I want to keep it simple and talk more about the algorithm and less about cleaning the data.

So, I want to introduce you to a website called kaggle which is a great place to get data for `machine learning` and `data science` projects.

Here you can find real, non-real, artificial, non-aritificial, man made, woman made, child made, computer made data for `machine learning` and `data science` projects. Also, you can do competitions and make friend and also code with your friends.

This website is amazing and the more you spend time on this site the more you get amazed how much a website can do.

So, I found a dataset called USA Housing. I don't know if it's real or not. But we can practice some model training with this dataset. You can either download the data from my github repo or you can download it from kaggle yourself(I recommend you download it from here).

So, let's see the data.

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
data = pd.read_csv('./USA_Housing.csv')
data.head()
```

Out[2]:

	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	208 Mic 674\
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 S
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	Stravenu
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Ba
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS F

We have `features` like `Avg. Area Income`, `Avg. Area House Age`, `Avg. Area Number of Rooms`, `Avg. Area Number of Bedrooms`, `Area Population`. So, linear regression is a model that predicts `numeric` values so, we should see which features are numeric and which features are categorical.

In [3]:
```
data.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
```

That's weird...

Every single column is a `float64` data type. But that's okay. Because it makes our work easier. There is no missing data in any of the columns and only one column is `object` data type which is the `address` column. Which I'll drop for this example.

> address is both usefull and useless. Depends on the context. Here a house price is effected by the location of the house, not the address of the house. We can process the address column to get the location of the house. But for now, as we want to focus on learnign the algorithms we will ignore the address column.

So, let's predict some `house price` using `linear regression`.

Prepare the Data

First, let's drop the `address` column.

In [4]:
```python
data.drop(
    columns=['Address'], inplace=True
)
```

In [5]:
```python
data.head()
```

Out[5]:

	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

It's kinda wierd that the number of rooms is a `float64` data type. Rooms and bedrooms should be whole numbers right? But as you can see, the column is not showing the room number of a house. It's showing the average number of rooms in a house in a particular area. So, I will keep it like this. But I know one thing for sure area population should be a whole number so, let's change it to `int64` data type.

In [6]:
```python
data['Area Population'] = data['Area Population'].astype('int64')
```

In [7]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 6 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   int64
 5   Price                         5000 non-null   float64
dtypes: float64(5), int64(1)
memory usage: 234.5 KB
```

Now, the data should look like this.

In [8]:
```python
data.head()
```

Out[8]:

	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	1.059034e+06
1	79248.642455	6.002900	6.730821	3.09	40173	1.505891e+06
2	61287.067179	5.865890	8.512727	5.13	36882	1.058988e+06
3	63345.240046	7.188236	5.586729	3.26	34310	1.260617e+06
4	59982.197226	5.040555	7.839388	4.23	26354	6.309435e+05

Great!

Now, let's do some exploratory data analysis and see if there is any `anomalies` in the data.

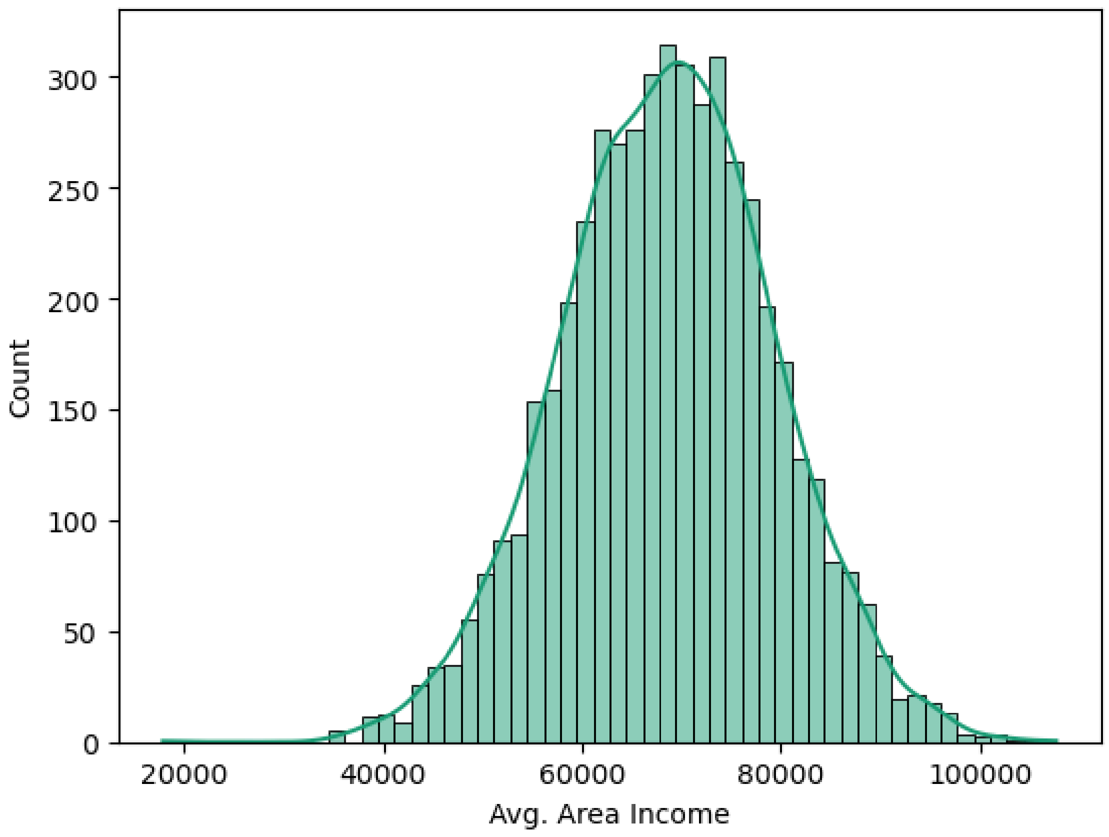
Let's see the distribution of `average area income`.

In [9]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_palette('Dark2')

sns.histplot(data=data,x='Avg. Area Income', kde=True)
```

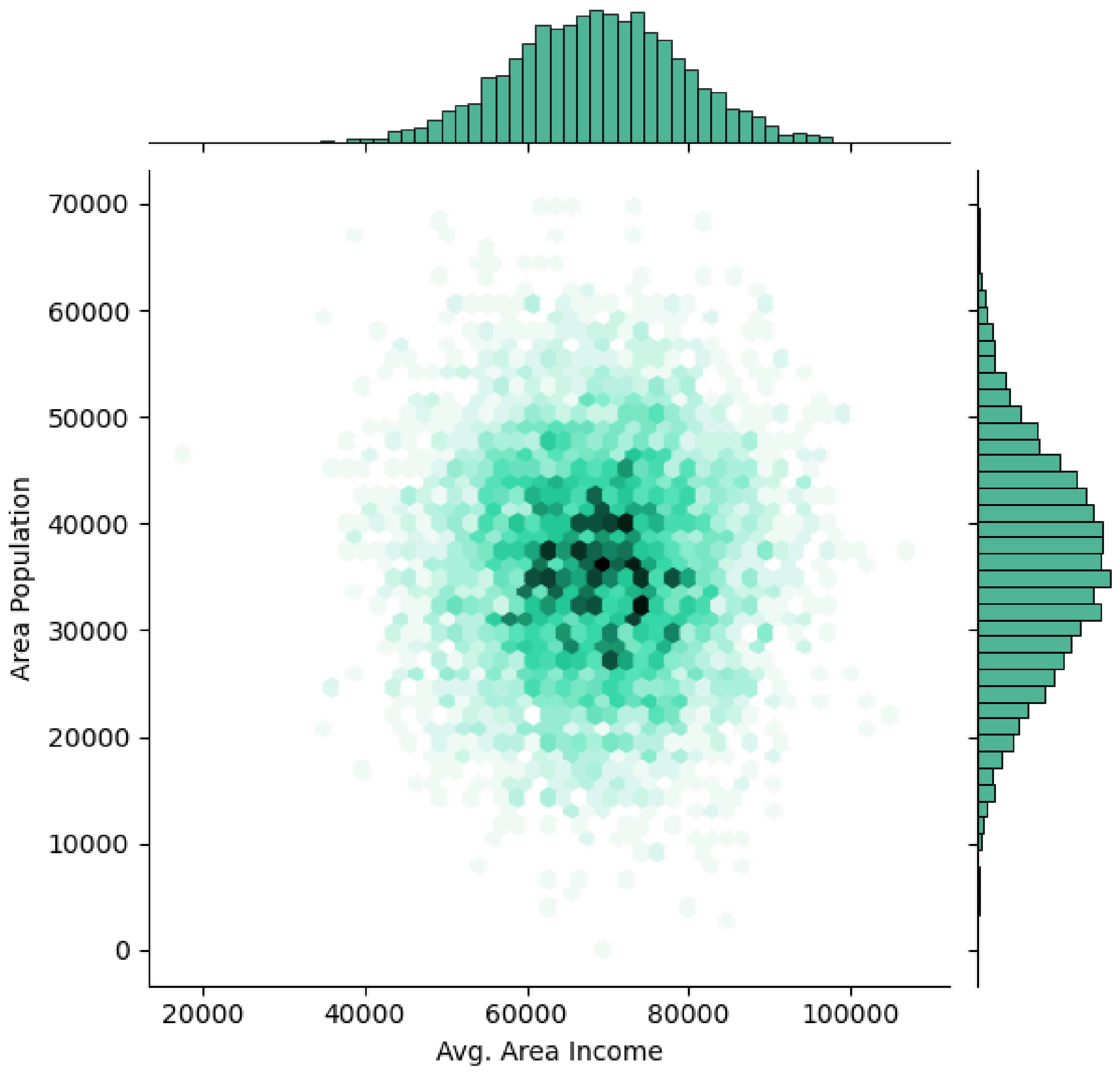
Out[9]:
```
<Axes: xlabel='Avg. Area Income', ylabel='Count'>
```



Well, it looks normal to me. I don't see any anomalies in the distribution of `average area income` but I want see `average area house age vs area population` distribution.

```
In [10]: sns.jointplot(
             data=data,
             x='Avg. Area Income',
             y='Area Population',
             kind='hex'
         )
```

```
Out[10]: <seaborn.axisgrid.JointGrid at 0x759de0dbfc50>
```



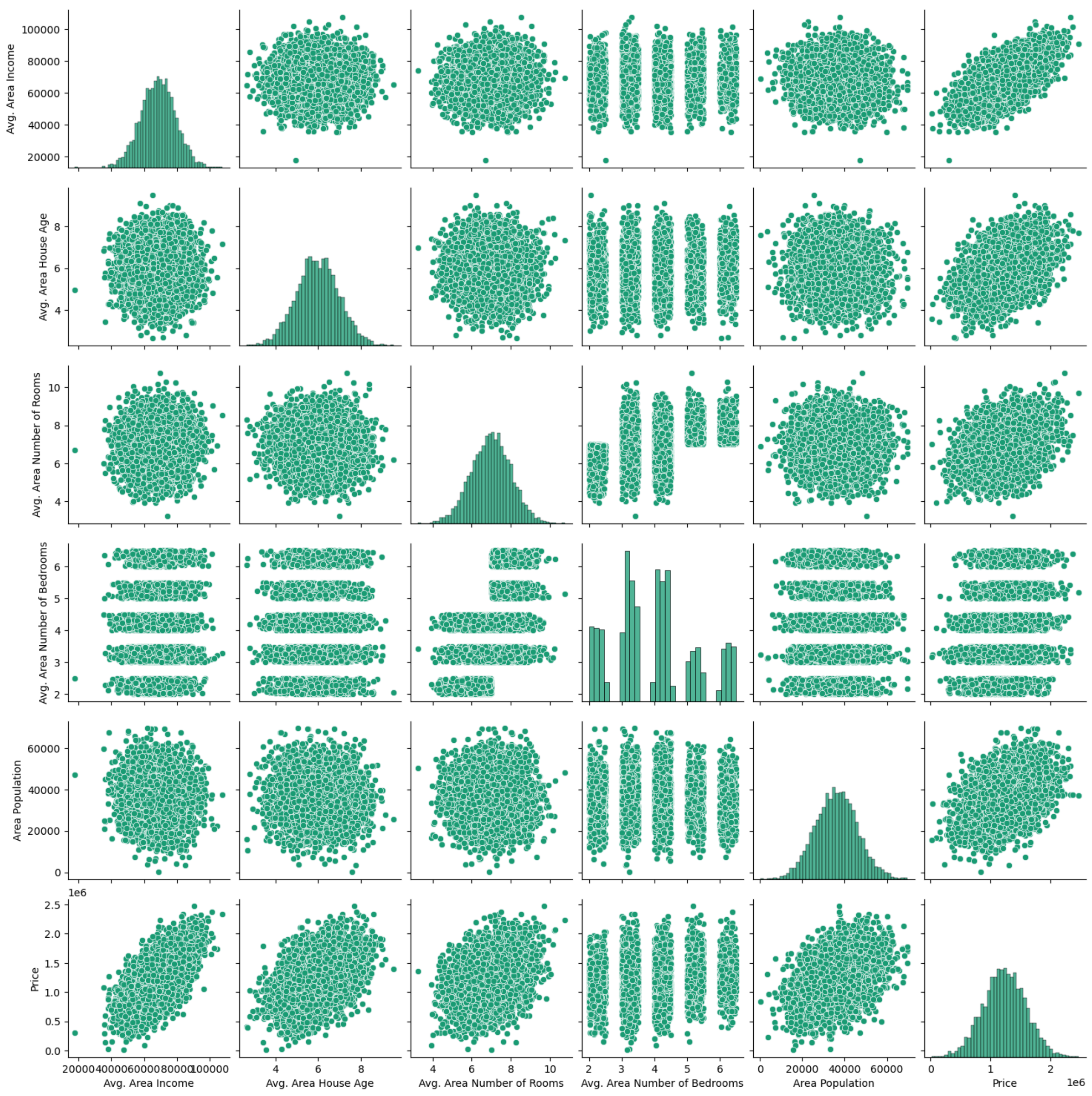
Well, that confirms it. The where the population is more `concentrated` the `average area income` is also more concentrated.

That area with 30000-40000 people has a higher `average area income` than the area with 10000-20000 people.

Finally I want to see the whole datasets distribution.

```
In [11]: sns.pairplot(data=data)
```

```
Out[11]: <seaborn.axisgrid.PairGrid at 0x759dd9017b50>
```



Everything looks good and normally distributed and I don't see any abnormalities in the data.

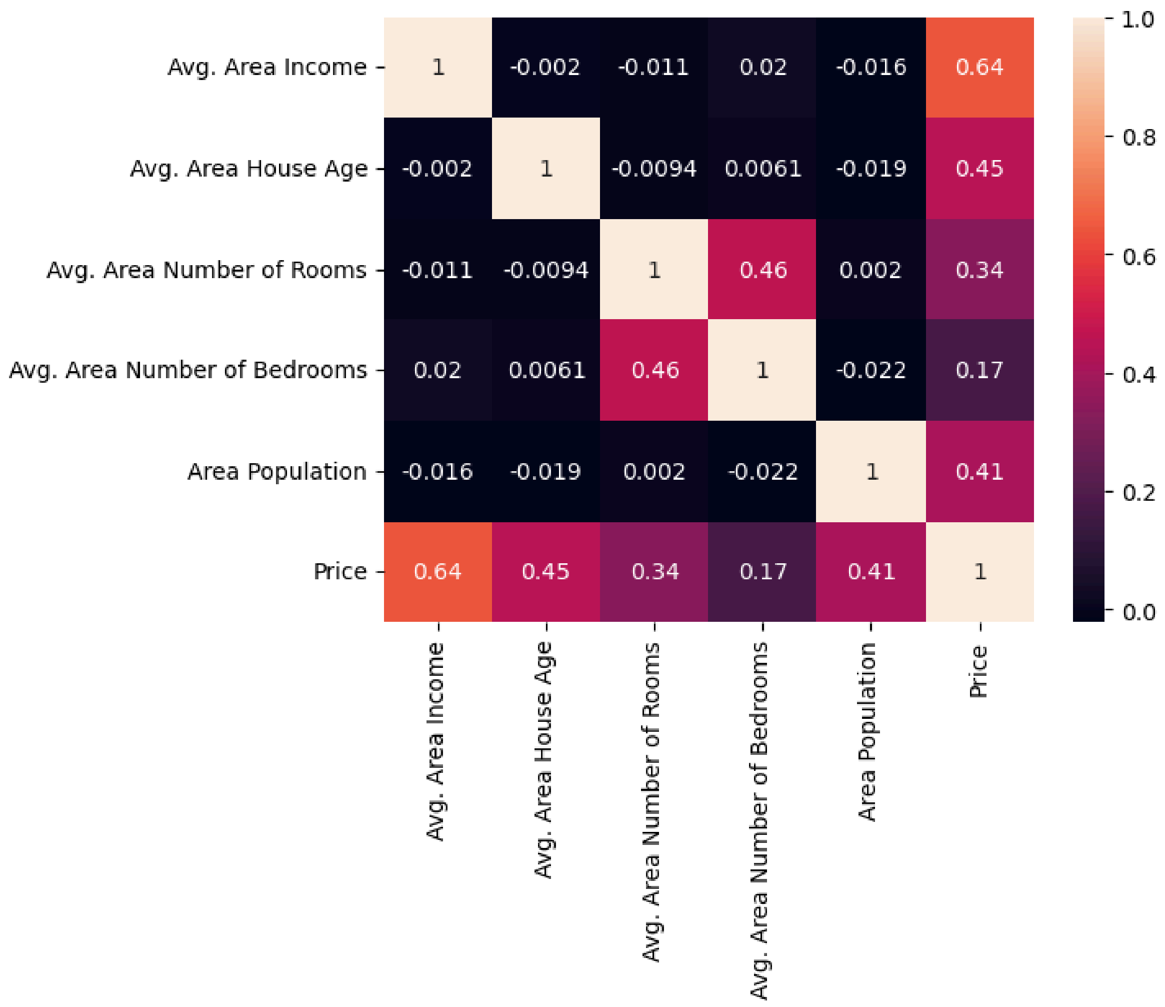
So, before I go into making the model I just want to confirm my hypothesis that every column in this dataset is `positively correlated` with the `target`.

We can simply visualize that using a `heatmap`.

In [12]:
```
corr = data.corr()

sns.heatmap(corr, annot=True)
```

Out[12]:
```
<Axes: >
```



YES! It looks like every column in this dataset is `positively correlated` with the `target`. So, I think I can use `linear regression` to predict the `target` from the `features`.

> You can see that most of the columns are `negetively correlated` and it's because every column is containing average values of a particular area. So We have to think about the relation betweem two columns with respect to the area. So, even though the distributions look normal the colums can be negetively correlated. And these numbers are so small that they are comparable to zero. But the main thing is the column are effecting the target.

Now, let's get into the fun.

Split the Data

Now, the last step before we train the model is to split the data into `train` and `test` sets.

So, how do we do that?

There is a library called `scikit-learn`, maybe the best the and the only library that you might need for almost every `machine learning` model.

> Scikit-Leanr is a Machine learning library with a cast collection of methods to prepare, train, test and deploy models. It is built with Python and NumPy and can seamlessly integrate with other libraries and frameworks like pandas.

We will use this library to do all the heavy lifting for us.

So, let's install the library.

```
conda install scikit-learn
```

You can also install it by running the following command:

```
pip install scikit-learn
```

And let's import the library.

Now, as I said earlier this library has a lot of functions that we can use and one of them is `train_test_split` which is used to split the data into `train` and `test` sets.

> train_test_split is a function that takes the `features` and the `target` as arguments and returns four values: `X_train`, `X_test`, `y_train` and `y_test`.

> `X` represents the `features` and `y` represents the `target`. `X_train` and `X_test` are the training and testing `features` respectively and `y_train` and `y_test` are the training and testing `target` respectively.

So, first let's make two variables `X` and `y` which will contain the `features` and the `target` respectively.

In [13]:
```python
X = data.drop(
    columns=['Price']
)
y = data['Price']
```

And we now we can use the `train_test_split` function to split the data into `train` and `test` sets.

In [14]:
```python
# train test split is a function of the model selection module
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X,y, train_size=0.8, s

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(4000, 5)
(1000, 5)
(4000,)
(1000,)
```

And we have our `train` and `test` sets.

Before we go into training the model, let's understand the `train_test_split` function a little bit more.

> train_test_split does somethings before actually splitting the data into `train` and `test` sets.

1. It `shuffles` the data.

Now, why do we need to shuffle the data?

The simple answer is to prevent `bias` in the model. If the data is directly split into `train` and `test` sets, there is a high possibility that a model will be biased towards the data that is in the `train` set.

Let's say you have a target column like this [1,1,1,1,2,2,2,2,3,3]. And you split the data into `train` and `test` set with train_size=0.8. That means the first 80% of the data will be in the training set. So, in this case the last 20% of the data which is just `3` will be left out and the model will trained on `1 and 2`.

> `train_size` is used to define the size of the split. The default value is 0.75 but if you want to change it you can do it by passing it as an argument to the `train_test_split` function. So, if you pass .7 as the `train_size` then the first 70% of the data will be in the `train` set and the last 30% will be in the `test` set.

So, Now you see why suffle is important.

2. Every time you run the `train_test_split` function, you will get different `train` and `test` sets.

This is because of the `random_state` argument. This is used to suffle the data and every time it changes it's value and that's it is a good practice to set a value for the `random_state` argument so that you can recreate the model.

And now that we have our `train` and `test` sets, let's learn about `linear regression`.

Linear Regression

Our data is prepared for training and now let's learn about the model that will be used to train the data.

Linear Regression is a very very very simple(mot at all, don't search linear regressing books on google) model that will be used to train the data.

Let's say you have a data set like this:

Feature 1	Feature 2	Target
1	2	3

Feature 1	Feature 2	Target
2	3	4
3	4	5
4	5	6
5	6	7
6	7	8
7	8	9
8	9	10
9	10	11
10	11	12

Now, what can we do to predict the `target` from the `features`?

If we plot the target versus the features, let's see what we see.

```
In [15]: example_df = pd.DataFrame(
             {
                 'Feature 1': np.arange(1, 11),
                 'Feature 2': np.arange(2, 12),
                 'Target' : np.arange(3, 13)
             }
         )

         example_df
```

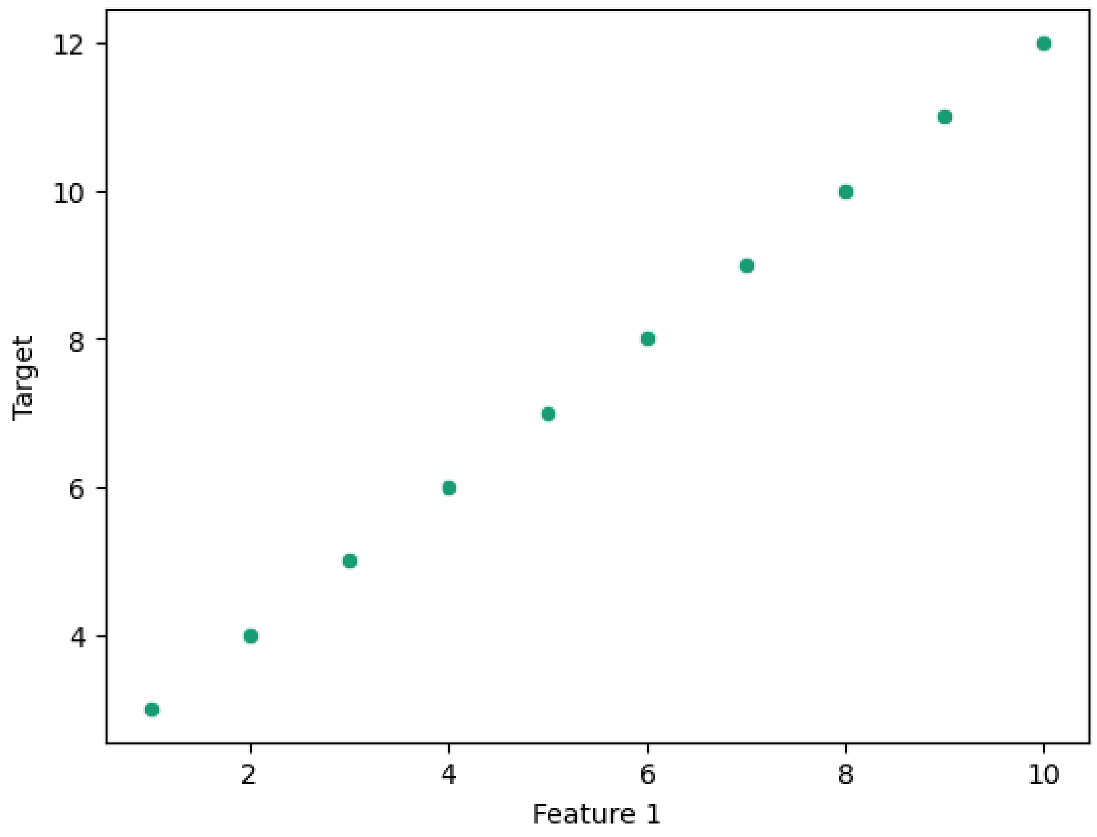
Out[15]:

	Feature 1	Feature 2	Target
0	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6
4	5	6	7
5	6	7	8
6	7	8	9
7	8	9	10
8	9	10	11
9	10	11	12

I just made the same data set as above but now I want to plot it.

```
In [16]: sns.scatterplot(
             data=example_df,
             x='Feature 1',
             y='Target',
         )
```

```
Out[16]: <Axes: xlabel='Feature 1', ylabel='Target'>
```



As you can see the `target vs feature 1` is a straight line.

So, next values should be in that straing line right?

This is what linear regression does.

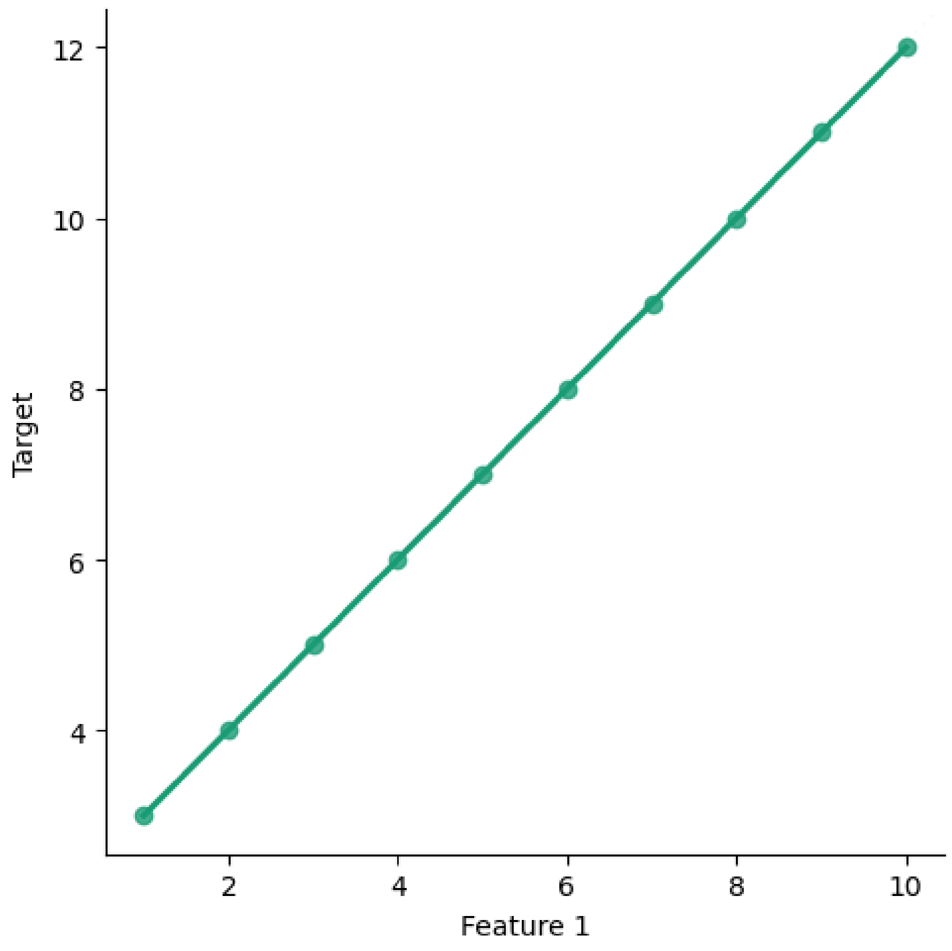
Linear regressing algorithm `finds a straight line` that best `fits` the data.

> Whena ml algorithm finds the pattern of the data, it's called `fitting` the data.

We can also, see the regressing like of `target vs feature 1` with seaborn.

```
In [17]: sns.lmplot(
             data=example_df,
             x='Feature 1',
             y='Target'
         )
```

```
Out[17]: <seaborn.axisgrid.FacetGrid at 0x759dc1e66450>
```



This line is called the `regression line` and the algorithms main task is to find a straing line that has the minimum `error` or the `residuals` between the data and the regression line.

So, for example, I'll adjust our dataset for more realistic view.

In [18]:
```
new_target_value = example_df['Target'].copy()
np.random.shuffle(new_target_value)

example_df['Target'] = new_target_value
example_df
```

```
/tmp/ipykernel_467970/3472451870.py:2: UserWarning: you are shuffling a 'Se
ries' object which is not a subclass of 'Sequence'; `shuffle` is not guaran
teed to behave correctly. E.g., non-numpy array/tensor objects with view se
mantics may contain duplicates after shuffling.
  np.random.shuffle(new_target_value)
```

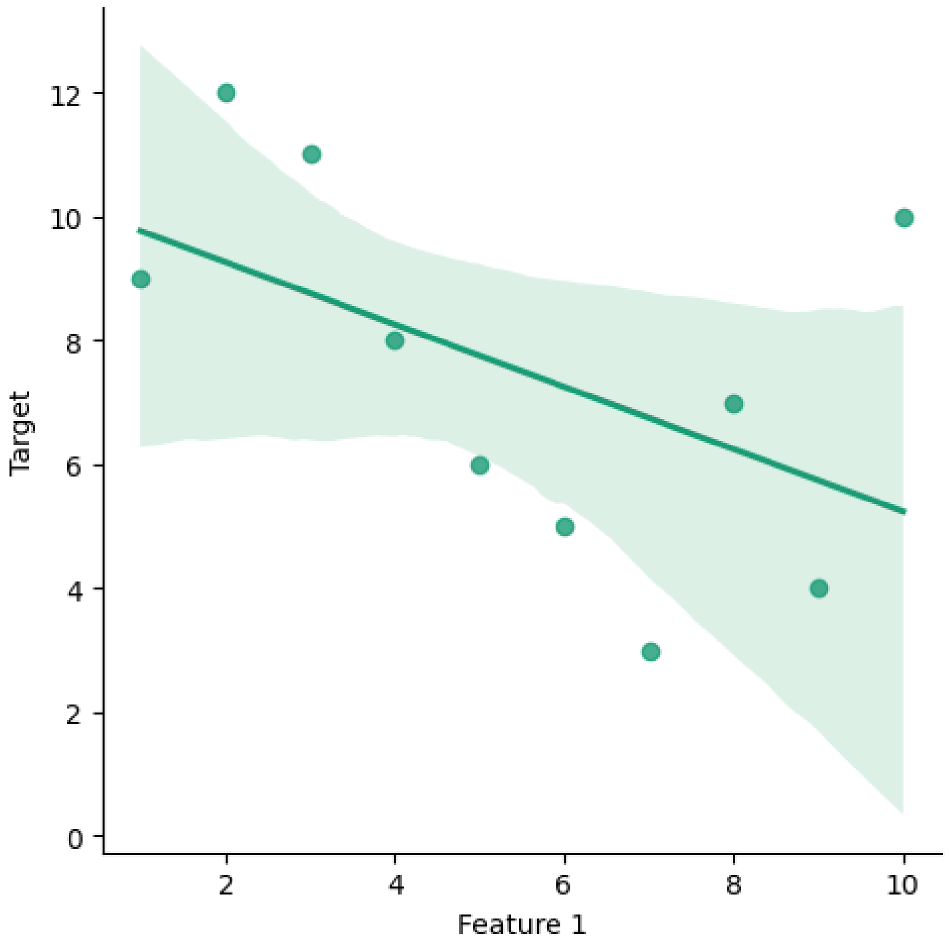
```
Out[18]:
```

	Feature 1	Feature 2	Target
0	1	2	9
1	2	3	12
2	3	4	11
3	4	5	8
4	5	6	6
5	6	7	5
6	7	8	3
7	8	9	7
8	9	10	4
9	10	11	10

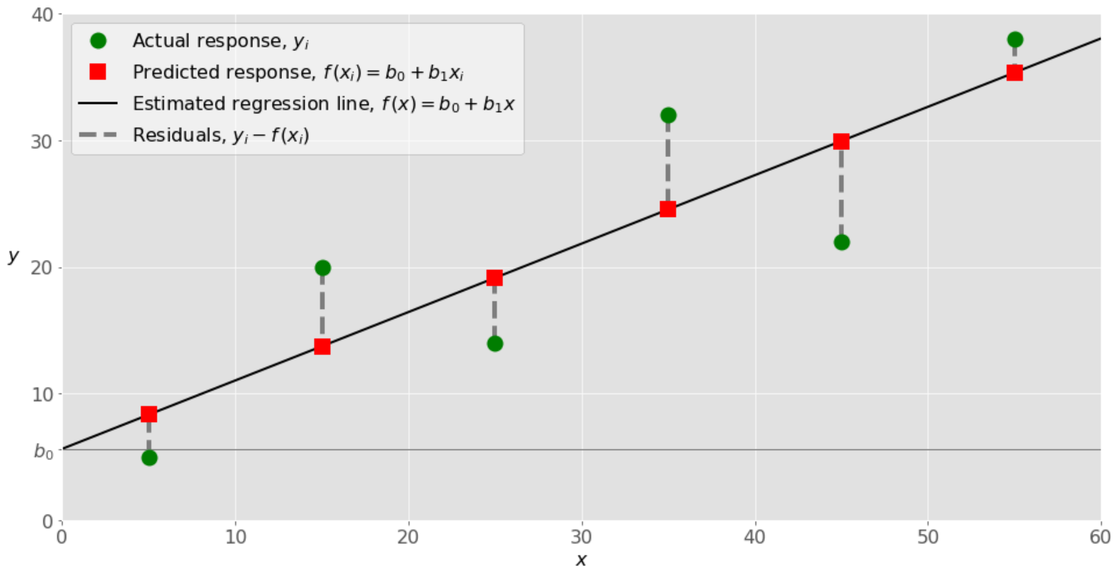
I suffled the target values and now I want to plot it.

```
In [21]: sns.lmplot(
             data=example_df,
             x='Feature 1',
             y='Target'
         )
```

```
Out[21]: <seaborn.axisgrid.FacetGrid at 0x759dd8377290>
```



Now, you have a better view of what is going on.



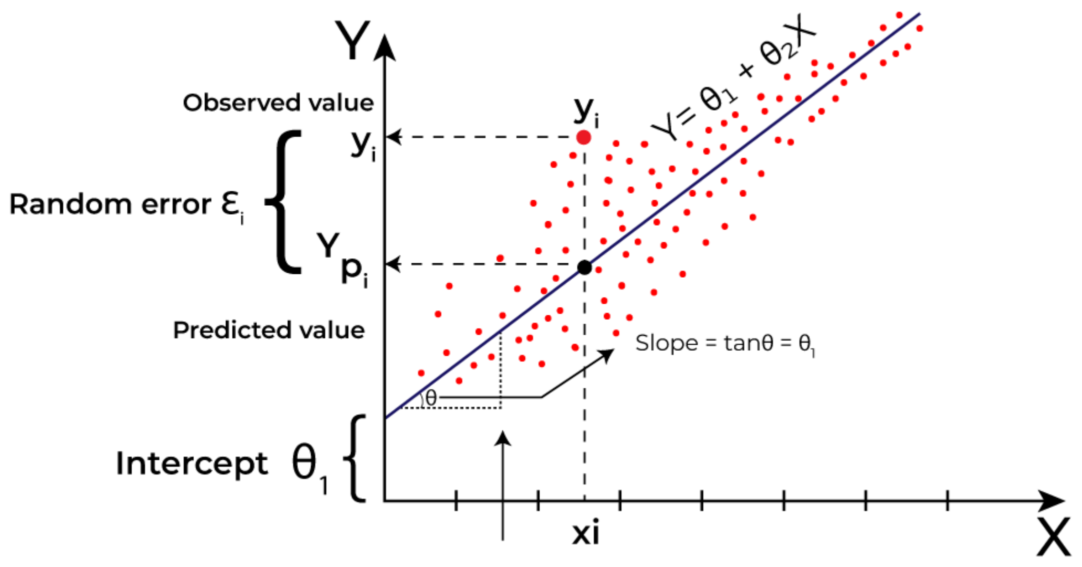
The algorithm has one task, to find a straing line that has the minimum `error` from every data point.

And the equation to that line is:

$$y = \beta_0 + \beta_1 x$$

Just a fancy way to write `y = mx + b` where `m` is the slope and `b` is the intercept.

The algorithm will try to find the `slope` and the `intercept` of the regressing line that will be at minimum error distance from every data point.



Y
Observed value
y_i
Random error ε_i
Y_{p_i}
Predicted value
Intercept θ_1
y_i
$Y = \theta_1 + \theta_2 X$
Slope = tanθ = θ_1
θ
xi
X

`Linear Regression` is the statistical method that finds a straight-lien relationship between inputs and outputs.

Linear regression tries to draw the line that best describes the relationship between the inputs and the outputs.

I hope you got the gist of it.

Now let's train the model.

Training the Model

We already have the train test sets, let's train the model.

Scikit-learn already has a function called `linear regression` that we can use to train the model.

In [22]:
```python
from sklearn.linear_model import LinearRegression

model = LinearRegression()
```

And we have our model in hand and ready to train.

In [23]:
```python
model.fit(X_train, y_train)
```

Out[23]:
```
▾ LinearRegression
▶ Parameters
```

We have to use the `fit()` method and inside the mathod first we pass the `features` and then the `target` values.

For us, that's the `X_train` and `y_train` variables.

And just like that the model has been trained.

And As I explained earlier the `linear regression` model has one task, to find a straing line that has the minimum `error` from every data point.

To describe that line we need the `slope` and the `intercept` values.

We, can now use `coef_` and `intercept_` attributes to get those values.

In [24]:
```python
model.coef_
```

Out[24]:
```
array([2.16522070e+01, 1.64666422e+05, 1.19624079e+05, 2.44034240e+03,
       1.52703170e+01])
```

And it'll returns a numpy array with the `slope` of the line for each `feature`.

We can easily format this into a datafram to get a better view.

In [26]:
```python
pd.DataFrame(
    index=X_train.columns,
    data=model.coef_,
    columns=['Coefficients']
)
```

Out[26]:

	Coefficients
Avg. Area Income	21.652207
Avg. Area House Age	164666.421931
Avg. Area Number of Rooms	119624.079051
Avg. Area Number of Bedrooms	2440.342401
Area Population	15.270317

In [25]:
```
model.intercept_
```

Out[25]:
```
np.float64(-2635065.488492687)
```

And the `intercept_` attribute returns the `intercept` of the line.

Now, as the model is trained we can use it to make predictions.

Making Predictions

We can use the `predict()` method to make predictions.

In [27]:
```
predictions = model.predict(X_test)
predictions
```

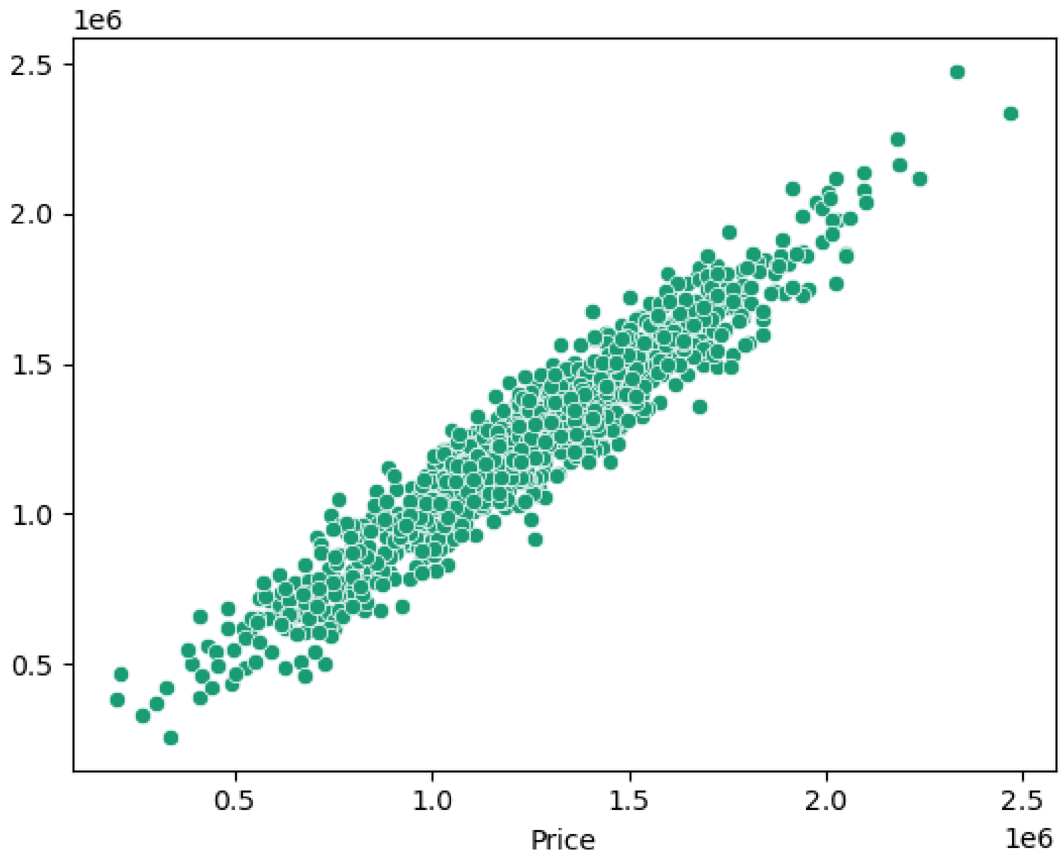
```
Out[27]:  array([1308581.01071068, 1237042.56863874, 1243434.84263746,
                 1228904.57446275, 1063321.17014634, 1544053.51357563,
                 1094770.94545506,  833283.10748247,  788410.79652598,
                 1469709.83960968,  671735.80497133, 1606819.1526406 ,
                 1004169.54794027, 1796802.69031365, 1288561.01354532,
                 1087784.17519401, 1423071.07868596, 1078181.39608634,
                  802281.66179687,  930766.72405962, 1134823.1052443 ,
                  916392.20953954, 1489976.294751  , 1284582.46134275,
                 1582072.99980339, 1132520.43029572, 1089884.66029928,
                  974517.98968995,  924052.81272256, 1740767.01023142,
                 1286475.06419415, 1621288.83354309, 1435258.03072299,
                 1234017.81013581, 1485430.66757187, 1718332.50964031,
                 1538952.65158026,  777112.6383753 , 1765196.3317792 ,
                 1175964.84898443, 1553712.23269422,  897710.56326408,
                 1371047.3777589 ,  845285.15158192, 1201028.7454289 ,
                 1133292.79503028, 1363121.8913513 , 1449818.91097458,
                 1574368.95430037, 1233578.83460129, 1484457.88293163,
                 1295272.54563773, 1222138.42591282,  990119.30339255,
                 1693822.91847773, 1823784.59702897, 1136491.07037318,
                 1282158.56096771, 1327294.39567783, 1353353.20730476,
                  966259.19842934,  661903.52917827, 1533755.25484484,
                 1002476.97853958,  995792.51103471, 1567356.95458191,
                 1500807.2507903 , 1090085.45934402, 1820960.70617976,
                 1479858.80495159,  902781.89025955, 1494546.6817349 ,
                 1378851.93991989,  962608.69821399,  712805.83371747,
                 1565650.40313274, 1149220.45216469,  931309.8944281 ,
                 1600924.08467906,  506874.35859692, 1592927.53972733,
                 1292016.24572658,  681259.18909599,  432970.63346695,
                 1395329.2615222 ,  696834.24151819,  663610.15720625,
                 1030072.89421631, 1485131.23924234, 1320068.45104793,
                 1271269.91271035, 1422269.56110886,  671604.89954099,
                 1149993.40978679, 1261306.32019986,  784693.98327182,
                 1189918.76176926, 1014894.33932548, 1405378.60633654,
                 1539081.94742648,  593299.63740971, 1391799.9467623 ,
                 1262735.66739109, 1875126.82474762, 2336893.62440627,
                  946298.03703466, 1315655.81524373,  953109.52419477,
                 1831764.09855762, 1625887.731429  , 1639242.62213433,
                 1267017.26206814, 1804843.39449279, 1220830.33955207,
                  850665.95925936, 1584182.04772669,  761909.93359567,
                 1360030.32693734, 1186063.98573178, 1559465.96604592,
                 1770325.78996845, 1608248.59240786, 1573659.13726333,
                 1531210.85874081, 1812540.43158777, 1124960.14630155,
                  669522.93815769,  978250.10257177, 1316295.11881401,
                 1627782.292125  , 1343334.33656253, 1088746.97361483,
                 1516820.09352218, 1333387.67059875, 1421957.08611747,
                 1527963.31211271, 1674055.50927624, 1301124.27170522,
                  843898.84598065, 1439398.76248495, 1857459.38904647,
                 1090381.94531756, 1845464.64624998, 1272017.2171664 ,
                 2036876.65859175,  882683.35426329,  954148.2551347 ,
                 1116677.80490366, 1395122.81096676, 1510718.59096897,
                  952944.5595919 ,  969884.62479323, 1320443.8586243 ,
                 1494548.95341234, 1311796.34209843,  764595.4621232 ,
                 1729945.05984152,  966394.24102227, 1980007.3872757 ,
                 1253541.19891789, 1378036.56065094, 1465703.21613905,
                 1041635.95497762, 2016180.21728441, 1425282.94546678,
                  783767.60330936,  826320.31657151, 1753900.37794331,
                 1077086.51345206, 2073698.02671367, 1735962.49799414,
                 1049436.1199503 , 1829293.65993091, 1128900.79402297,
                 1629028.56086363, 1634468.1894356 , 1524792.95893327,
                  722150.32108788,  740337.0673647 ,  547623.89017983,
                  828931.56614342, 1067367.20718149,  969249.18506174,
                 1227289.17703269,  871757.9195051 ,  504208.61491476,
                 1200036.83656484, 1062307.87109483, 1519497.9541355 ,
                 1329225.20392706, 1386537.10159415, 1320930.96036365,
                 1204353.48396435, 1240622.06312555, 1365631.27868269,
                 1769954.73478347, 1329716.99779058, 1376329.15408579,
                 1298985.46134512,  834746.76376906, 1272072.61109437,
                 1451499.86751249, 1289196.51333037,  896870.29745021,
                  990438.11896038, 1357840.68201398, 1649359.64110801,
                  779322.16843413, 1005586.28708533, 1467827.22278215,
                 1124911.50194063, 1142766.07503705, 1250202.95642428,
                 1105424.52605892,  622301.68371393, 1841658.06983886,
                 1182734.59674902,  609156.05795652, 1410668.86610042,
                 1263534.39535245, 1119671.76193417, 1145393.91015934,
                  500453.55946276, 1109883.31686439, 1626974.62750287,
                  797668.57288231, 1407085.68156612, 1560741.30872322,
                 1698893.67969362, 1714324.11280835, 1216157.71026073,
                 1114780.594886  , 1001072.85447345, 1065265.33837014,
                 1625180.70554491, 1593969.52245977, 1526219.60201411,
                 1129979.32454835,  735892.31903317, 1593874.51856686,
                 1732249.79852678, 1264947.63324111, 1216705.69331077,
                 1539829.82555749, 1748280.13766086,  544483.66536666,
                 1177812.22391365, 1404016.74691019, 1125399.42382239,
                  545092.98858538, 1290744.82520905, 1658367.14499065,
                 1487208.31664549, 1291141.32973776, 2050449.95917829,
                  768872.32226966, 1863639.45620355,  929814.22314663,
                 1761087.18189625, 1659004.58747588, 1247441.12869489,
                  713429.95236865, 1778761.62980127,  461931.28057743,
                 2167300.73391807,  664670.78825139, 1801833.4326536 ,
                 1265105.90799668, 1139224.12909256, 1173982.64906289,
                 1748194.27745856, 1147791.10139137, 1029926.04106132,
                  947128.10983002, 1182462.59653952,  927595.18070498,
                 1566516.47882417, 1175820.29206998,  388094.11927464,
                 1288960.94409961,  986663.15210731, 1340020.02019213,
                 1044425.55014554, 1286335.58641182, 1530997.98945277,
                 1667764.2211924 , 1084803.49787574,  720961.54851085,
                  812003.2659476 , 1328870.13004114, 1519503.6905801 ,
                 1379415.14654505, 1457743.32697671,  998978.22889249,
                 2252172.60489164, 1491963.38054912, 1349006.13926073,
                 1042664.13196654,  486555.6537881 , 1792551.4893881 ,
                 1193460.24178056, 1247208.79732018, 1122697.572441  ,
                 1004015.33005585, 1699114.25942742, 1504247.54274655,
                 1432828.44463205, 1342789.49626315, 1601418.50659127,
                  821509.78205102, 1126576.24792585, 1291475.40949343,
                 1698971.83146382, 1232605.74454872, 1489641.06031053,
                 1233257.83411663, 1135050.24010371, 1732120.42705952,
                 1348168.87307708,  621079.76256944, 1501389.43581605,
                 1201749.57140421, 1326722.92039169, 1097545.83576204,
                 1221286.31723297,  788260.44616358,  935555.83142105,
                 1287407.80101334, 1473895.28064428,  609892.93553621,
                 1183184.30105367, 1035479.21747176,  809282.93611181,
                  655566.07572267,  983751.48179365,  986625.80803874,
                 1320590.54154183, 1196956.28380219,  961007.4321371 ,
                 1275840.98256586, 1425244.43538439, 1654322.63386602,
                 1521722.10008156, 1396980.97698705, 1413831.17009271,
                 1219884.40844028, 1232370.5655212 , 1654014.2281145 ,
                 1328318.09568335, 1321083.9942136 , 1299214.36041252,
                 1347307.2475923 , 1688688.81060367, 1574512.27134922,
                 1045332.16939677, 1356268.79010283, 1439507.70288776,
                 1504155.44167535, 1138142.71962412, 1500050.23333208,
                 1486905.98434776,  385023.97538742, 1216217.72992042,
                 1726336.73257065, 1207804.05820543,  894597.82789422,
                 1240637.62410745, 1308445.9009465 , 1437359.25248367,
                  893241.39349376, 1792784.98144512, 2076936.76075378,
                  922560.99765556, 1458344.12817568, 1249779.56019092,
                 1370807.01158326, 1121252.61165083,  963225.89435908,
                 1327242.19507344, 1187322.36223222,  943711.48456294,
                  424847.1151748 , 1223722.70126977, 1411761.15625795,
                 1236009.05694498, 1449014.14253641, 1411711.21824702,
                  926885.2944173 , 1006635.37318849, 1325144.48817817,
                  470011.48701371, 1568406.94375275, 1071686.11545938,
                 1261710.39207403,  983177.84456041, 1086379.05409567,
                  726357.67401386,  814171.80700115,  803021.19446626,
                 1619610.48631597,  459333.98487767, 1300393.95933789,
                  961516.45162381, 1025135.96958988, 1384785.33969711,
                 1235855.38774921,  769795.40370396, 1026171.89768433,
                 1405631.79129328, 1756296.84736763, 1280494.50892522,
                  622594.19623631, 1449918.267239  , 1198421.44622153,
                 1356963.14868894, 1079278.97648232, 1352327.56470791,
                 1057109.34172446, 1290456.37582617,  705162.80475145,
                 1200361.58413726, 1507694.24708834,  962341.01050516,
                 1204225.76594838, 1571861.7484577 , 1209469.56292863,
                 1549860.0300921 ,  769640.19215114, 1404680.16252551,
                 1240640.13012263, 1029495.6416624 , 1335463.13117284,
                 1088599.66292317, 1047145.56110216, 1388976.83854214,
                  918929.84199865, 1096413.34605799, 1406973.46525757,
                  806955.65132452, 1196190.9671857 , 1010135.50768945,
                 1394710.44829982, 1257528.36745232, 1299687.6344453 ,
                 1294974.85586116, 1723109.75376483, 1711249.4845776 ,
                 1174496.65142106,  897736.56759106, 1356642.19121245,
                 1219845.93794713, 1366676.29708746, 1644466.64434907,
                  724994.53402221, 1570998.99373315,  924993.47469208,
                  673281.08242669, 1589687.30854429, 1373581.45612314,
                 1236758.16270476, 1939041.16719484, 1248344.80371858,
                 1393191.03259161, 1703471.57749901, 1059234.40915152,
                 1268669.07577026, 1304696.44782603,  892458.72751602,
                  964860.14201581,  818818.3996247 , 1401576.45850305,
                 1180880.69814464, 1259247.01744007, 1702904.81087928,
                 1649090.10954887, 1731294.69318813, 1125719.70382678,
                  865175.51804904,  968236.57792643,  784706.65933984,
                  781817.5745268 , 1209734.98925661,  972637.32148938,
                 1576924.06621565, 1060075.6893438 ,  694196.95948355,
                 1610581.10213516, 1309665.21603173, 1141468.65461386,
                 1205905.29015935, 1281341.342866  , 1171886.09791572,
                 1567899.44724601, 1322655.0255603 , 1166508.70322732,
                  789694.97494937, 1143724.13323814,  648254.30508641,
                 1071133.21842649, 1428572.15573739, 1424324.53833225,
                 1057495.79487824, 1430509.46005003, 1324084.72732751,
                 1292015.62974065, 1071124.34805955, 1287171.39870564,
                 1019161.96726287, 1037474.04332331, 1049933.97294885,
                  731785.58810169,  782566.03717371, 1002201.60573441,
                 1067326.49197049, 1124199.7249757 , 1457959.80198987,
                 1108351.30232344, 1598776.20659554, 1176016.1822219 ,
                 1096986.92154031, 1247378.52524596, 1363561.81099204,
                  773366.15292978,  822335.29713731,  993391.83736991,
                 1543917.63821831, 1493181.39850698, 1175892.57159071,
                 1253692.51836031, 1521006.88186051, 1627807.52159164,
                 1686079.9619657 , 1980131.78088498,  977460.45136747,
                 1109686.0129652 , 1113006.37868996, 1211539.23484322,
                 1407483.44899157,  789313.78474761, 1234898.25209406,
                 1209073.79987605, 1803376.96515268, 2114893.51143151,
                 1118851.45459001, 1206709.68185611, 1352853.05975386,
                 1396182.17011285, 1546377.68301134,  666836.70574042,
                 1481419.72679828, 1021473.18030345,  968492.90703082,
                 1424085.71483141, 1914790.10414847,  914203.94760764,
                 1489854.94770978,  693919.68906281, 1031060.50218563,
                 1352124.06191107, 1373506.81927043, 1049112.58314425,
                 1486518.99707197, 1310517.58501236, 1647942.37226118,
                 1283580.25728884, 1452985.41897418, 1429415.86348985,
                 1034087.88059825, 1134636.30377596, 1403560.97293617,
                 1584889.50647722, 1746825.46364176, 1466280.74074845,
                 1294751.54649215, 1051687.40641588, 1101554.45579158,
                 1465162.90268305, 1382732.41590952,  939222.18219687,
                 1337344.34451952, 1120979.12848168, 1319827.77155948,
                  558966.00861593, 1401607.61729606, 1354541.62845116,
                 1306336.90926512, 2086265.57290708, 1904135.2587414 ,
                  539892.4390333 , 1525526.79411014, 1235215.9845961 ,
                  750638.01529775, 1127964.45479133, 1471493.80908123,
                  619420.34081604, 1115003.31822814, 1061461.72225454,
                 1013599.244599  , 1581431.39274964, 1079495.23205197,
                 1055056.17236798, 1034583.97687614, 1224312.01422426,
                  887273.1581298 , 1516250.88858906, 1865328.42601406,
                 1027373.17633005, 1490057.59189778, 1337282.87577961,
                 1119420.41949067,  979085.16890818,  679559.04977052,
                 1325910.50111608, 1171592.89770042, 1567496.81272073,
                 1143860.63447489, 1126155.22430333,  864149.7457481 ,
                  490141.13328141, 1291066.39463294, 1038391.53250718,
                 1662681.0638133 , 1160620.9673178 , 2037286.36968723,
                 1516256.93095763, 1229069.53637782, 1547215.49224586,
                 1434747.90949711, 1709129.8647322 , 1281179.54307014,
                 1524051.30786209,  968009.11933986, 1293672.58095111,
                 1106479.85519732, 1431111.7008398 , 1712065.93503162,
                 1160247.539837  ,  655742.33863297, 1226710.64065609,
                  927582.25942869, 1267636.62416374,  891673.69553707,
                 1234373.98599634,  921266.25172545,  973621.15187904,
                 1211116.14464685, 1152861.00434455, 1769234.72346776,
                 1185566.01923969,  982629.90540206, 1332527.5119026 ,
                 1642474.28227428,  965389.27814784, 1281974.69876199,
                 1406340.35309863,  677672.47209188, 1209554.67899545,
                  732923.34730017, 1682128.42393425,  872614.33645627,
                 1136137.45408401,  510264.26000149, 1288107.05284592,
                 1371034.63622594, 1310120.39873694, 1436852.93243778,
                 1051226.46808388,  684829.83876346, 1101023.24861434,
                 1567989.00486549, 1018010.00497483, 1050701.07055259,
                  825823.64760017, 1173624.71838145, 1591979.19449699,
                 1595723.74394152, 1360886.57532561, 1192751.98386928,
                  655831.76278592, 1560105.53815402,  813166.82668698,
                 1132148.52807398, 1564590.8141896 , 1118674.58854321,
                 1569768.02704115, 1383014.67891972, 1609534.43723762,
                 1045542.44582631, 1174356.21100261, 1447874.16513214,
                 1824784.34490009, 1639846.41678174, 1488883.01380091,
                 1427511.02463218, 1231678.98336781, 1368008.09215282,
                 1424922.91758982, 1551125.75000156,  914696.73771341,
                 1249843.66248736,  258040.09145085, 1230971.77935951,
                 1126749.62638171,  858786.08177544,  739046.92433749,
                 1044216.68094539, 1112382.41811254, 1223247.97427211,
                 1447903.03475979, 1328623.21378636, 1049310.57303083,
                  769594.59999525, 1751852.12863847, 1463026.30490511,
                 2139923.91111539, 1718405.65885256, 1115994.29683168,
                 1995823.7747327 , 1193383.76486488,  660186.83078965,
                  660682.92082385, 1798650.10193204,  470124.42551905,
                 1364095.14014282,  889318.09050635,  706545.84004527,
                 1399884.94970267, 1416306.64282398,  967235.18621019,
                 1729650.66588014,  869955.25340374, 1437821.4168205 ,
                 1122486.69806344, 1673256.70500514, 1440044.39570527,
                 1500537.15824992,  971586.46809392, 1043185.96688548,
                 1317537.98344872, 1007625.88277157, 1323347.84179497,
                 1135661.13695857, 1510355.51557515,  993059.53606753,
                 1208826.31500061, 1988661.96137133, 1230194.93906559,
                 1390407.52839748, 1262919.65670276, 1582324.85325915,
                 1028419.38733988, 1672114.67397285, 1265750.9669541 ,
                 1236605.53636191,  329710.26642032,  717958.9624125 ,
                 1103734.7291001 , 1188713.26320902, 1261800.45976552,
                 1193843.4171663 , 1303384.36960533, 1076899.79856105,
                 1169748.6190393 , 1458476.07205043, 1190651.21541096,
                 1360609.18705453, 1497401.9111102 ,  916160.45915039,
                 1311288.87198511, 1579842.76386999, 1155922.77228989,
                 1425751.7932932 , 1287289.05036771, 1153856.11783055,
                 1825983.93713141, 1049735.91817595,  909548.37932324,
                 1253931.76052363, 1467213.99803936, 1639984.26039641,
                 1521773.40760562, 1019776.54583934,  652303.78559464,
                  643372.74718255, 1142031.46842531, 1263241.53405144,
                 1258081.24553498,  589604.67323532, 1187013.25590901,
                 1121745.61543524, 1808967.41911429, 1800049.12136652,
                 1123521.28787107, 1425294.72857977, 1510531.81157226,
                 1323067.65058099, 1443107.63202997, 1296650.73396469,
                 1551950.92483251, 1237294.92793299,  665441.43265824,
                 1262713.75508237, 1048880.33247589, 1867302.76399464,
                 1562899.2071779 ,  731317.44074269, 1239101.34446745,
                  909235.33078041, 1752100.33093334, 1602829.56929417,
                 1504850.42190153, 1411561.51702467, 1394414.87983431,
                  370716.08193928, 1672306.51024747, 2120152.00742177,
                  421201.82704517, 1486649.6744485 , 1200111.58302753,
                 1531545.07962133, 1031786.84483134,  842915.62115281,
                  838957.6377278 ,  596824.45992752,  855567.20218524,
                 1040347.40451108, 1499201.31588399, 1353189.45899057,
                 1167133.25743745,  980456.71734486, 1615645.09466486,
                 1192261.56595699, 1280275.70163576,  542775.85125051,
                 1121556.50745582, 1430527.17875358, 1187716.20659465,
                 1293446.36973834, 1344673.5196317 , 1183421.71553631,
                 1508086.29963044, 1265160.43406317, 1342600.22525461,
                 1277578.77838212,  733239.95186418, 1553396.2306774 ,
                 1398167.59582418, 1187015.08566453, 1371020.46053747,
                 1108980.58218564,  975277.60486906, 1202027.68787441,
                 1933561.33071922,  753141.69771829,  972289.39235218,
                 1448912.11516862, 1859397.68668329,  914395.53680156,
                 1004530.70384351,  924304.35157821,  964224.45249439,
                 1220870.34120182, 1266521.13419771, 1619403.50212775,
                 1582516.26606419,  764721.09396426, 1739057.51532454,
                 1593278.18863009, 1677324.48373688,  982949.88184499,
                 1370595.91984723,  906583.90312637, 1631023.63859897,
                  690987.58086862, 1573207.25184479, 1466491.78146789,
                 1650260.87363899, 1182472.04903333, 1173491.49208081,
                 1333496.35064533, 1256984.15209279, 1859913.70357027,
                  758941.2151325 , 1392549.88878182, 1711867.89971001,
                 1561615.25651861,  886778.50046686, 1312022.90392266,
                 1174659.26161451, 1071959.79108047, 1381321.17297063,
                 1106586.85128273, 1170607.12208347,  992968.08818805,
                 1471291.28961579, 1531566.8018789 , 1609335.47334044,
                 1374687.54609447,  963512.14991626, 1516660.93971866,
                 1161809.95604154,  874671.46332176, 1168470.19994823,
                  608245.76618667, 1499900.67963604, 1020501.88806917,
                 1657909.68711951,  870372.68681214, 1469293.49569801,
                 1585005.13019098, 2474726.65996497, 1312115.89932008,
                  927777.36358046, 1291394.70280262, 1591845.26571793,
                  601184.74855153, 1595043.37674665, 1186555.75966639,
                 1569916.33268986,  775295.8198003 , 1498586.42588644,
                  731221.50177431, 1504697.89072472, 1046683.24440096,
                 1305409.43886119, 1275949.85694532,  997330.26436095,
                  617842.58364156, 1591120.352187  , 1400756.53396251,
                 1311862.99423235, 1197042.35099079, 1041877.71945026,
                 1530063.24350204,  967933.38900735, 1427002.20311465,
                 1165153.87836653,  869042.91173668, 1541096.16725804,
                  952860.14216882, 1649276.45583689, 1262906.85046186,
                  630788.02150028,  576876.16332836, 1709541.0365753 ,
                 1627859.56309498,  838662.71661703,  878771.36037099,
                 1299351.85461091, 1160410.59238338,  749974.78526665,
                  983978.60878279, 1153311.58787989,  943185.53507664,
                 1123567.83660636, 1547451.66433711, 1298895.46177214,
                  694676.31125373, 1042417.81212078,  858374.27723384,
                 1238166.63959701, 1069912.24423643, 1032800.56479861,
                 1215822.60031076, 1344359.93309328, 1128690.00037688,
                 1451586.71140471, 1094468.06445285, 1242819.39988085,
                 1312627.09312665, 1501590.57621021, 1577867.11060771,
                 1263870.37887385,  966357.62784311, 1304934.46343134,
                  948726.47350216, 1116938.46705941, 1177295.42553225,
                  988224.34927559, 1035263.82612742, 1114921.40413045,
                 1691525.44345393,  493937.12409347, 1819475.60440268,
                 1387274.94885001,  632700.94371593, 1224354.30867164,
                 1180743.31215851, 1036235.54810382,  921237.99151615,
                 1754055.2495626 , 1583333.71708174, 1563762.61417528,
                 1342872.08854068, 1173600.89635467, 1042899.40107588,
                 1318538.0250079 , 1870008.2727134 , 1663622.55355978,
                 1045092.85028269])
```

AWell, that's a lot of predictions.

We cannot get any usefull information from them. Let's try to visualize them.

In [30]:
```
sns.scatterplot(
    x=y_test,
    y=predictions,
)
```

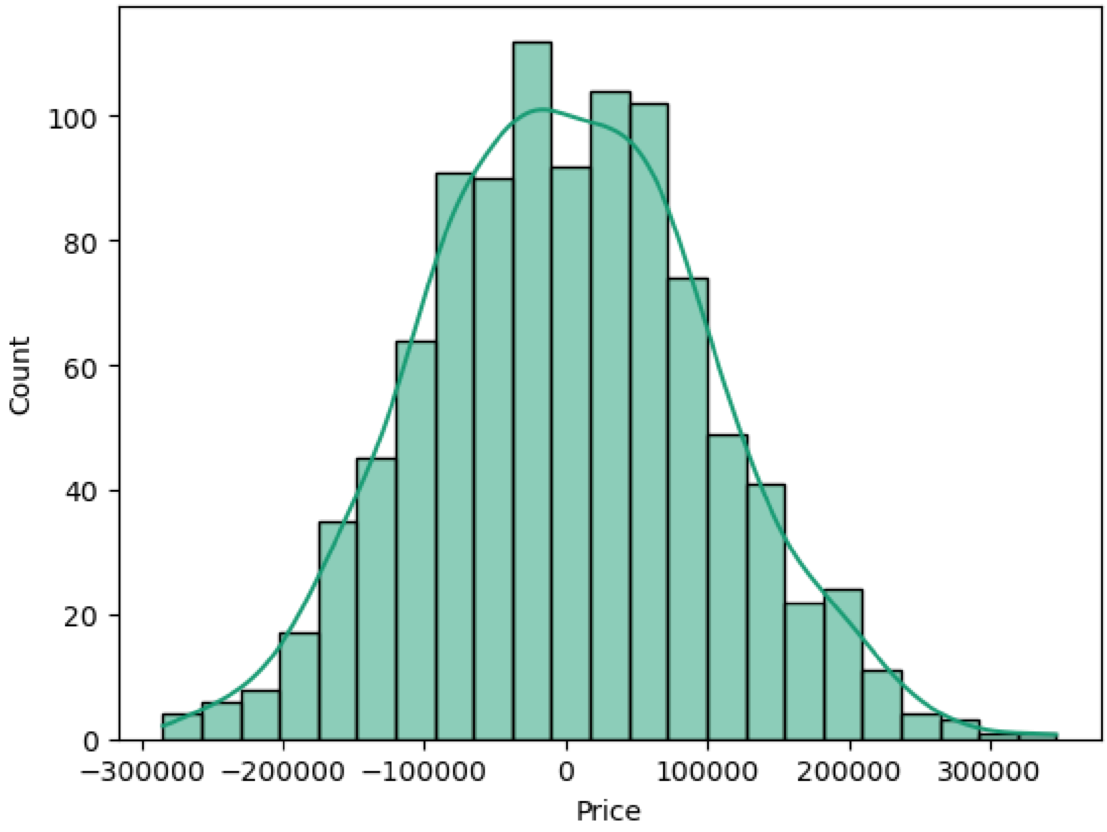
Out[30]:
```
<Axes: xlabel='Price'>
```



That't good we see that the predictions are close to a straight line. That is what we want and we to confirm this I'll see the `error/residuals` distribution of the predictions.

```
In [32]: sns.histplot(
             data=(y_test - predictions),
             kde=True
         )
```

```
Out[32]: <Axes: xlabel='Price', ylabel='Count'>
```



And this is what you want to see.

WHen you train a model after prediction the models `error` distribution should be normally distributed. If there is a abnormality in the `residual` distribution then that's a sign that the model is overfitting.

So, this also means that `linear regression` is the correct model for this dataset.

So, let's evaluate.

Evaluation

We will use `mean absoulute error`, `mean squared error` and `root mean squared error` to evaluate the model.

In [34]:
```
from sklearn import metrics

MAE = metrics.mean_absolute_error(y_test, predictions)
MSE = metrics.mean_squared_error(y_test, predictions)
RMSE = np.sqrt(MSE) # just square root of MSE
```

In [35]:
```
print(f"MAE: {MAE}")
print(f"MSE: {MSE}")
print(f"RMSE: {RMSE}")
```

```
MAE: 80878.79115229756
MSE: 10088963923.461655
RMSE: 100443.83467123134
```

As you can see the `RMSE` and the `MAE` are close to each other, not much but this will do. These matrics are also refered to as `loss functions` and `error functions`.

The loss is `80,878` in this context of house price that is in `millions` the prediction is not that far off. I guess this is a good model.

And that's it for linear regression.

> Explaining ML models are a challenge without visualization or simulations.
> I tried my best to explain linear regression in a simple way. I'm not perfect but I hope you got the gist of it.

Final Words

Mathematics is the key to be a good `ML engineer`. So, if you are passionate about machine learning, you should open up you mathematics book and god through the `matrix, linear algebra and statistics` NOW!

I hope you like this article. If you did that leave a like and share it with your friends.