8.1-Linear Regression

KeytoDataScience.com

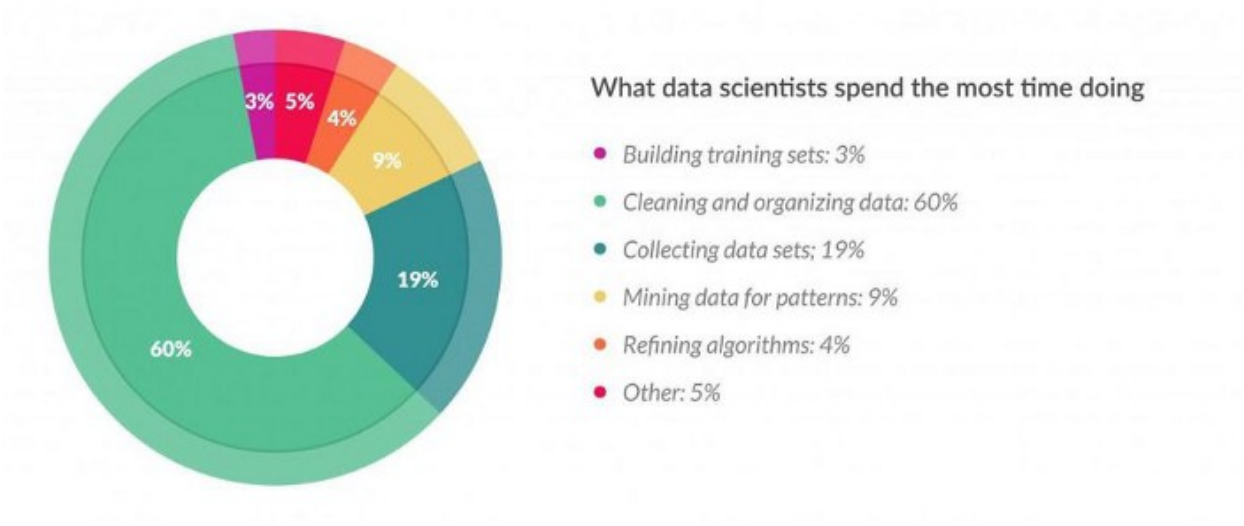
For any Data Science project, we will follow the approach of Data Science Project Lifecycle:

💡💡💡💡💡 **Data Science Project Life Cycle** 💡💡💡💡💡

---

1. **Understand Problem/Objective**
2. **Data Collection**
3. **Data Preparation**
    - 3.1 Cleaning Data
    - 3.2 EDA
    - 3.3 Train/Validation/Test Split
    - 3.4 Feature Engineering
    - 3.5 Feature Selection
4. **Modeling**
    - Classification
    - Regression
5. **Evaluation**
    - Classification
        - Confusion Matrix, AUC-ROC, Lift Chart, Gain Chart, KS Statistic, F1 Score, etc.
    - Regression
        - RMSE, RSE, MAE, RAE, Coefficient of Determination (R2)
6. **Model Deployment**
    - Model Deployment in pipeline or tool

According to a survey in Forbes, data scientists spend 80% of their time on data preparation:



KeytoDataScience - DataScience Project Life Cycle

Table of Contents

1. Understand Objective

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

↥ back to top

Let's get started!

2. Data Collection

We've been already provided with the dataset for housing prices as a csv file. So, data collection part is already done.

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
```

Load 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

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

Check out the Data

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')
```

↥ back to top

3. Data Preparation

3.1 Cleaning Data

Check null values

In [7]:
```
USAhousing.isnull().sum()
```

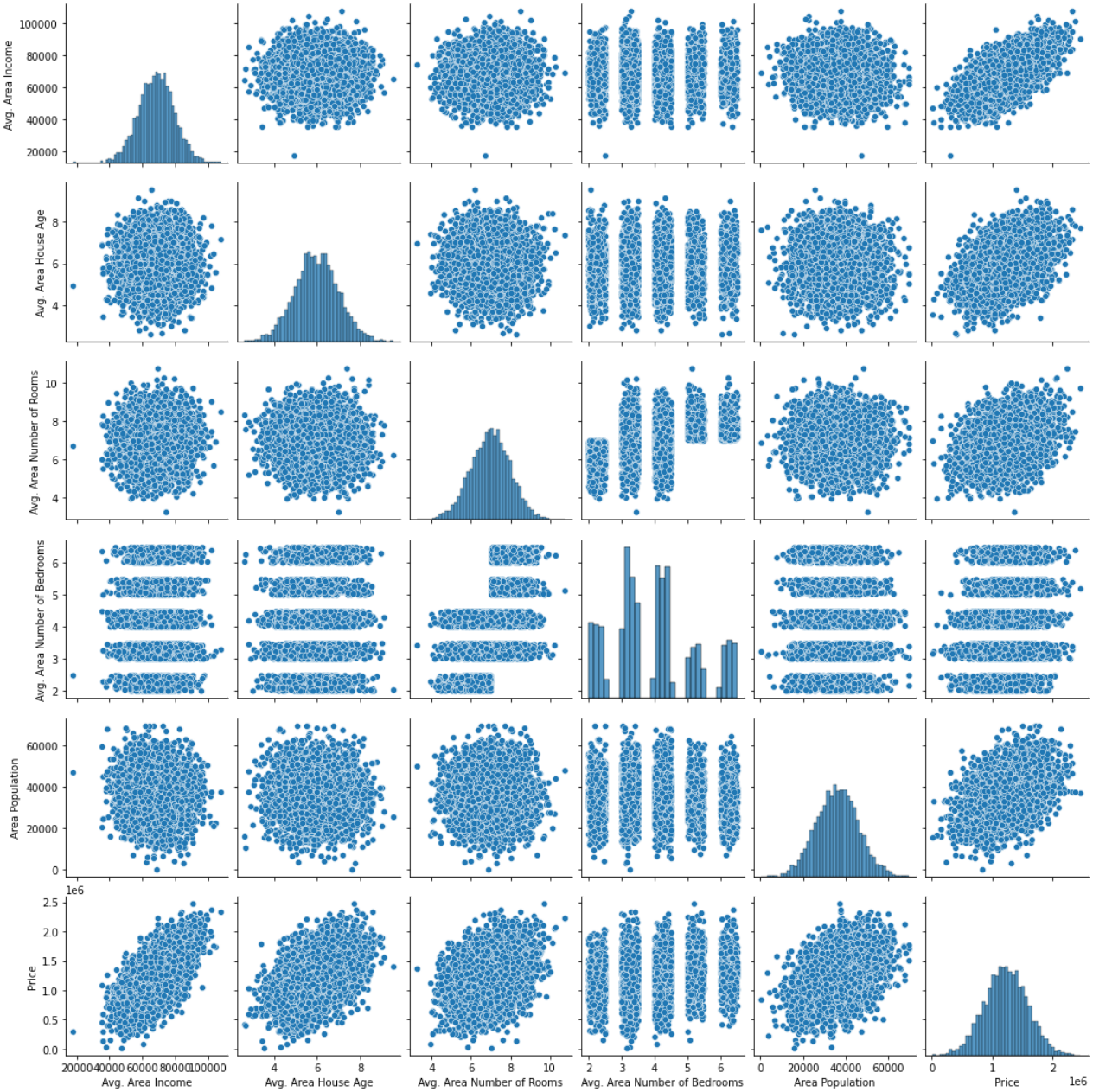
Out[7]:
```
Avg. Area Income              0
Avg. Area House Age           0
Avg. Area Number of Rooms     0
Avg. Area Number of Bedrooms  0
Area Population               0
Price                         0
Address                       0
dtype: int64
```

3.2 EDA

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

In [8]:
```
sns.pairplot(USAhousing)
```

Out[8]:
```
<seaborn.axisgrid.PairGrid at 0x25978cb0640>
```



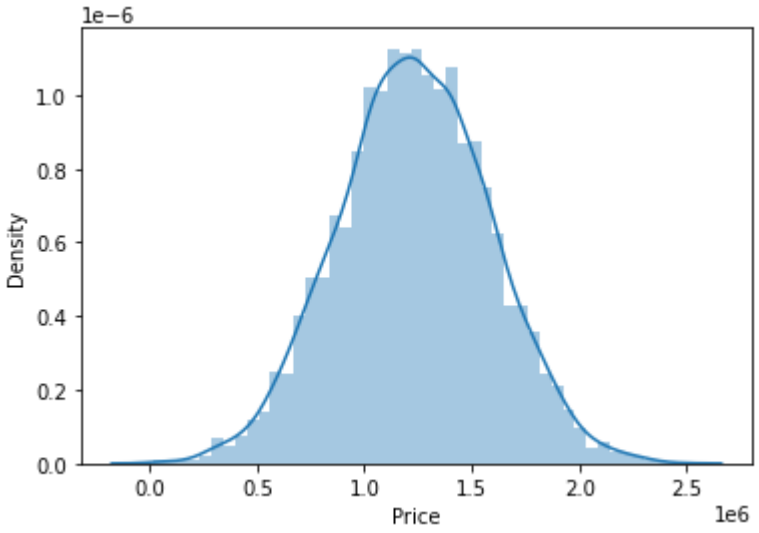
In [9]:

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

```
C:\Users\Prateek\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\distri
butions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a figure-level func
tion with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

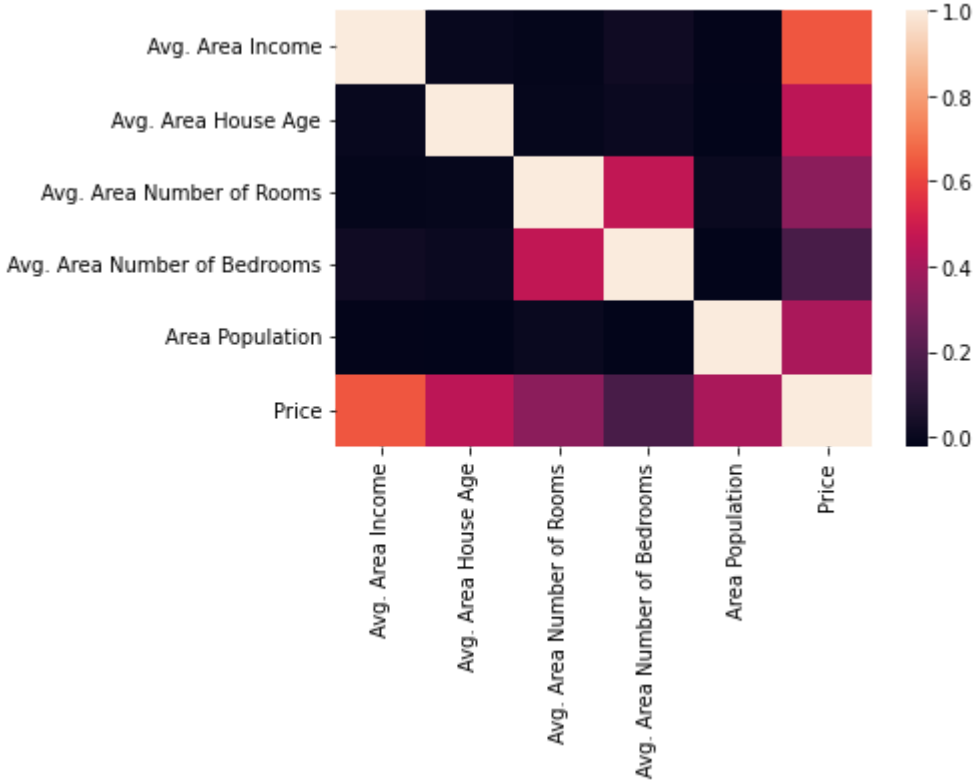
Out[9]:

```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```



In [10]:
```
sns.heatmap(USAhousing.corr())
```

Out[10]: <AxesSubplot:>



Let's assign **Independent Variables(X) and Dependent Variable(y)**

Q. What is an independent variable?

An independent variable changes in the experiment and has effects on the dependent variable.

Q. What is the dependent variable?

A dependent variable is the variable being tested and measured in a scientific experiment.

Example1

In the current case study, variables like `Avg. Area Income`, `Avg. Area House Age`, `Avg. Area Number of Rooms`, etc are independent variables. These all independent variabales have effect on `Price` which will be a dependent variable.

Example 2

For example, we have a function `y = x1 + x2 + x3`. In this `x1`, `x2` and `x3` are independent variables and `y` is dependent variable.

NOTE

- Independent variable: Features
- Dependent variables: Labels, Target variable

Let's 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 drop the Address column for now because it only has text info that the linear regression model can't use. Linear Regression can take only numerical variables.

In [11]:
```
X = USAhousing[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
                'Avg. Area Number of Bedrooms', 'Area Population']]

y = USAhousing['Price']
```

3.3 Train/Validation/Test Split

Why we split dataset into Train, Validation and Test Set?

1. **Train Set** (60% of the original data set): This is used to create our prediction algorithm. Our algorithm tries to tune itself according to the training data set. In this phase, we usually create multiple algorithms in order to compare their performances during the Validation Phase (or Cross-Validation Phase).

2. **Validation Set** (20% of the original data set): In Training we must have trained multiple algorithms. Now we must pick one algorithm. That's why we have a validation set. Used in tuning model hyperparameters. Most people choose the algorithm that performs best on the validation set (and that's ok). But, we must measure our choosen top-performing algorithm's error rate on the test set also. If we just go with the error rate on the validation set, then we have gravely mistaken the "best possible scenario" for the "most likely scenario." That's a recipe for disaster.

3. **Test Set**(20% of the original data set): This is the Out-of-time test set (OOT test set) which model has never seen. Model performance on this test dataset is most closest to unseen real-world data. We just have to make prediction using our choosen model and check if it holds the same performance or it is way off.

Info: For the sake of simplicity (as we are not tuning any hyperparameters), we will split the data

only into train and test sets.

In [12]:
```
from sklearn.model_selection import train_test_split
```

In the `train_test_split` parameter `test_size` we can use the percentage of test dateset.

Let's choose `test_size` = 0.3 or 30%, that mean train set size is 70% of original data set.

In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
```

3.4 Feature Engineering

Think of any new feature or any insight that can be generated out of existing data.

List of Techniques used:

1. Imputation
2. Handling Outliers
3. Binning
4. Log Transform
5. One-Hot Encoding
6. Grouping Operations
7. Feature Split
8. Scaling
9. Extracting Date

3.5 Feature Selection

📌 **Correlation Check among independent variables**

In [14]:
```
def correlation(dataset, threshold):
    col_corr = set()  # Set of all the names of correlated columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i, j]) > threshold: # we are interested in absolute
                colname = corr_matrix.columns[i]  # getting the name of column
                col_corr.add(colname)
    return col_corr
```

Below code will drop the features(columns) having correlation above 80%, by choosing `threshold = 0.8`

In [15]:
```
corr_features = correlation(X_train, 0.8)
print('correlated features: ', (set(corr_features)) )
```

```
correlated features:  set()
```

No correlation detected among the independent training variables.

📌 **Variance Inflation Factor (VIF) Check**

We can use Variance Inflation Factor (VIF) analysis also. We will cover this in future sessions.

↥ back to top

4. Modeling

Let's now begin to train out regression model!

Creating the Linear Regression Model

In [16]:
```
from sklearn.linear_model import LinearRegression
```

In [17]:
```
# We create an Linear Regression algorithm instance.
lr = LinearRegression()
```

Training the Linear Regression Model

The `fit()` function is used to train our algorithm. It takes the input dataframe `X_train` and tries to fit it to the expected output.

In [18]:
```
lr.fit(X_train,y_train)
```

Out[18]:
```
LinearRegression()
```

↥ back to top

Model Evaluation

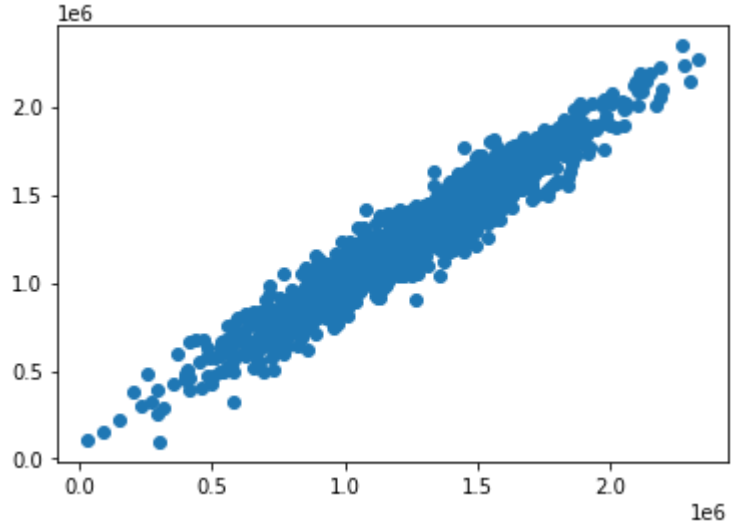
5.1 Model Prediction

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

In [19]:
```
predictions = lr.predict(X_test)
```

In [20]:
```
plt.scatter(y_test,predictions)
```

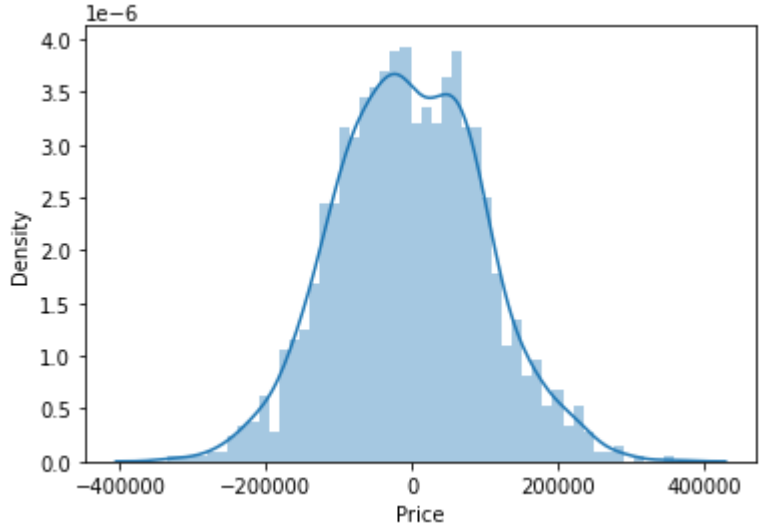
Out[20]:
```
<matplotlib.collections.PathCollection at 0x2597fd625b0>
```



Residual Histogram

In [21]:
```
sns.distplot((y_test-predictions),bins=50);
```

```
C:\Users\Prateek\AppData\Local\Programs\Python\Python39\lib\site-packages\seaborn\distri
butions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed
in a future version. Please adapt your code to use either `displot` (a figure-level func
tion with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



5.2 Model Evaluation

Training Set Model Evaluation

Evaluation metric: **R2 (R-squared)**

R-squared (R2) is a statistical measure that represents the proportion of the variance for a dependent variable that's explained by an independent variable (or variables) in a regression model.

In [22]:
```
from sklearn.metrics import r2_score

y_train_predict = lr.predict(X_train)
r2 = r2_score(y_train, y_train_predict)
```

```
print("The model performance for training set")
print("--------------------------------------")
print(f'R2 score is {round(r2,3)}')
```

```
The model performance for training set
--------------------------------------
R2 score is 0.918
```

Test Set Model Evaluation

In [23]:
```
y_test_predict = lr.predict(X_test)
r2 = r2_score(y_test, y_test_predict)

print("The model performance for testing set")
print("--------------------------------------")
print(f'R2 score is {round(r2,3)}')
```

```
The model performance for testing set
--------------------------------------
R2 score is 0.919
```

📌 Advance Analysis

Let's evaluate the model by checking out coefficients and interpreting them.

In [24]:
```
# print the intercept
print(lr.intercept_)
```

```
-2641372.6673014304
```

In [25]:
```
coeff_df = pd.DataFrame(lr.coef_,X.columns,columns=['Coefficient'])
coeff_df
```

Out[25]:

	Coefficient
Avg. Area Income	21.617635
Avg. Area House Age	165221.119872
Avg. Area Number of Rooms	121405.376596
Avg. Area Number of Bedrooms	1318.718783
Area Population	15.225196

📌 Interpreting the coefficients:

- Holding all other features fixed, a 1 unit increase in **Avg. Area Income** is associated with an **increase of $21.62**.
- Holding all other features fixed, a 1 unit increase in **Avg. Area House Age** is associated with an **increase of $165221.12**.
- Holding all other features fixed, a 1 unit increase in **Avg. Area Number of Rooms** is associated with an **increase of $121405.38**.
- Holding all other features fixed, a 1 unit increase in **Avg. Area Number of Bedrooms** is associated with an **increase of $1318.72**.

- Holding all other features fixed, a 1 unit increase in **Area Population** is associated with an **increase of $15.22**.

Does this make sense? Probably not, as this is a made up this data. If you want real data to repeat this sort of analysis, check out the boston dataset:

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

📌 **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 [26]:
```
from sklearn import metrics
```

In [27]:
```
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: 81257.55795855941
MSE: 10169125565.897606
RMSE: 100842.08231635048
```

This was your first real Machine Learning Project! Congratulations on helping your neighbor out!

We'll let this end here for now, but go ahead and explore the Boston Dataset mentioned earlier, if this particular data set was interesting to you!

↥ back to top

Great Job!
