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

In [2]:
```
plt.rcParams['figure.figsize'] = [19, 8]
```

In [3]:
```
import warnings
warnings.filterwarnings('ignore')
```

Variables

A variable is nothing but data. In a dataframe the columns are called variables.

The variables are of 2 types:

- Dependent Variable
- Independent Variable

Dependent Variable:

The variable whose value is to be forecasted or predicted is the dependent variable.

The value of the dependent variable is dependent on the values of the independent variables.

Dependent variables are also called response variables or target variables.

Dependent variables are generally represented by the letter y.

Independent Variables

Independent variables are used to explain the dependent variable.

Independent variables do not depend on any other variable.

Independent variables are also called feature variables or regressors.

Independent variables are represented by the letter x.

Regression

Regression is a statistical approach to analyze the relationship between a dependent variable and one or more independent variables.

Regression is used to determine the most suitable function that characterizes the relationship between dependent and independent variables.

Linear Regression

Linear regression is a machine learning algorithm that depends on the linear relationship between a dependent variable and one or more independent variables.

Two variables are in linear relationship when their values can be represented using a straight line.

Simple Linear Regression

A simple linear regression model has one independent variable that has a linear relationship with the dependent variable.

For more than one independent variables, it is called Multiple Linear regression.

The Linear Equation

In mathematics, a straight line is represented by the equation: y = mx + c, where m is slope and c is a constant.

Using the above equation, the value of y (dependent variable) can be calculated using x (independent variable).

In statistics, the equation is written as: $y = \beta_0 + \beta_1 x + \epsilon$.

Here β_0 is the intercept, β_1 is the slope and ϵ is the error component.

Slope $\beta_1 = \delta y / \delta x$.

Intercept β_0 = Distance on y axis when value of x is 0.

The error component (term), ϵ, represents the distance of the actual value from the value predicted by the regression line.

When the relationship between the independent variable (x) and the dependent variable (y) is linear, Linear Regression model is applied to analyze the data.

Example

In [4]:
```
data = {'voltage': [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
       'current': [0, 1, 2, 2.99, 4, 5, 6, 6.99, 8, 9, 10]
       }
```

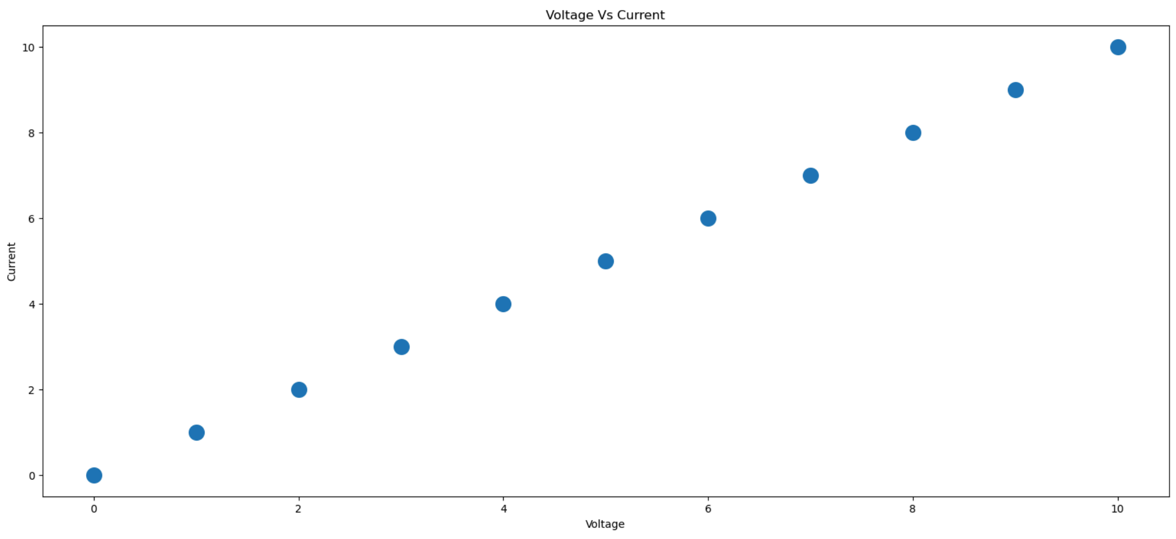
In [5]:
```
df = pd.DataFrame(data)
```

In [6]:
```
df
```

```
Out[6]:
```

	voltage	current
0	0	0.00
1	1	1.00
2	2	2.00
3	3	2.99
4	4	4.00
5	5	5.00
6	6	6.00
7	7	6.99
8	8	8.00
9	9	9.00
10	10	10.00

```
In [7]: plt.scatter(data=df, x="voltage", y="current", s=200)
        plt.title("Voltage Vs Current")
        plt.xlabel("Voltage")
        plt.ylabel("Current")
        plt.show()
```



Example

```
In [8]: homeprice = {'area': [2600, 3000, 3200, 3600, 4000], 'price': [550000, 565000, 6
```

```
In [9]: df = pd.DataFrame(homeprice)
```

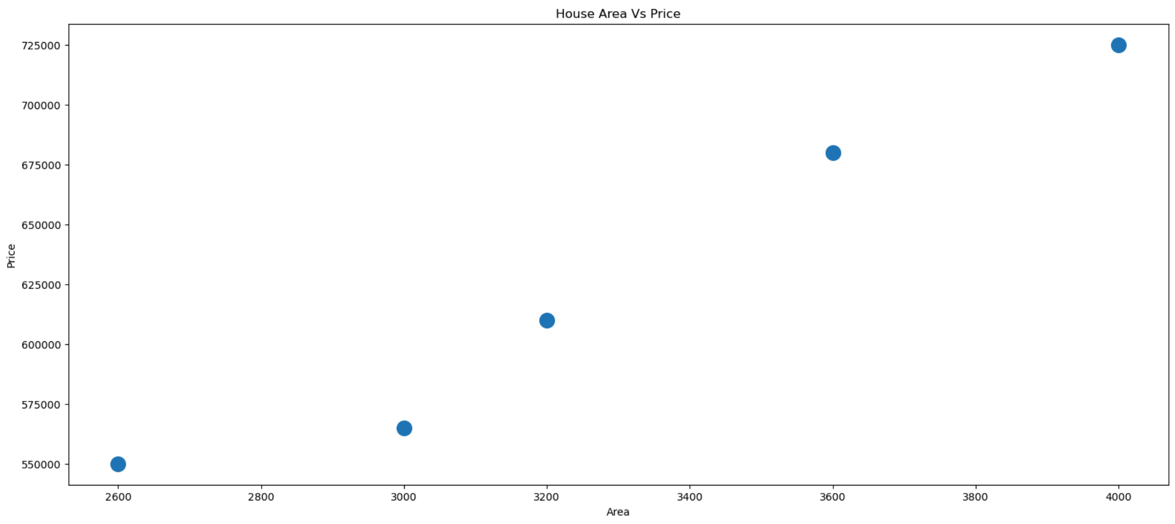
```
In [10]: df
```

Out[10]:

	area	price
0	2600	550000
1	3000	565000
2	3200	610000
3	3600	680000
4	4000	725000

In [11]:

```python
plt.scatter(data=df, x='area', y='price', s=200)
plt.title("House Area Vs Price")
plt.xlabel("Area")
plt.ylabel("Price")
plt.show()
```



From the above plot, the data points can be connected more or less using a straight line. Hence we can apply Simple Linear regression machine learning model on the above data.

Ordinary Least Square Method

The regression line that best explains the trend in the data is the best fit line.

The line may pass through all of the points or non of the points.

The method aims at minimizing the sum of squares of the error terms i.e., it determines the values of β_0 and β_1 at which the error terms are minimum.

$\beta_0 = \bar{y} - \beta_1 \bar{x}$

$\beta_1 = Cov(x, y)/Var(x)$

$\beta_1 = \sum((x - \bar{x})(y - \bar{y}))/\sum(x - \bar{x})^2$

Measures of Variation

Sum of Squares Total (SST)

The sum of squares of total is the sum of squared difference between the target variable and its mean value.

It is also known as Total Sum of Squares (TSS).

SST = $\sum(y - \bar{y})^2$

Sum of Squares Regression (SSR)

The sum of squares regression is the sum of squared difference between the predicted value and the mean of the target variable.

It is also known as Regression Sum of Squares (RSS).

SSR = $\sum(y_{predict} - \bar{y})^2$

Sum of Squares of Error (SSE)

The sum of squares of error is the sum of squared difference between target variable and its predicted value.

It is also known as Error Sum of Square.

SSE = $\sum(y - y_{predict})^2$

Total Variation

SST = SSR + SSE

Coefficient of Determination R^2

Gives the total percentage of variation that is explained by the target variable.

$R^2 = SSR/SST$

$R^2 = 1 - (SSE/SST)$

Note: $0 \leq R^2 \leq 1$

Case Study

Predict the Salary of an employee depending on the Years of Experience

Read the Data

In [12]:
```
salary_df = pd.read_csv("./datasets/Salary_Data.csv")
```

Exploratory Data Analysis

In [13]:
```
salary_df.shape
```

Out[13]:
```
(30, 2)
```

In [14]:
```
salary_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     int64  
dtypes: float64(1), int64(1)
memory usage: 612.0 bytes
```

In [15]:
```
salary_df.head()
```

Out[15]:

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891

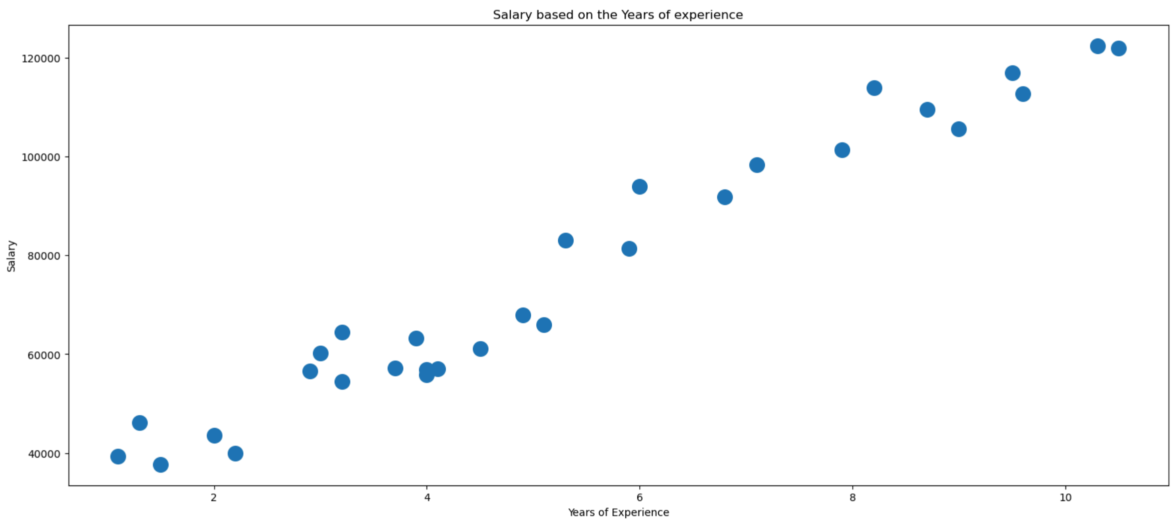
In [16]:
```
salary_df.describe()
```

Out[16]:

	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

Data Visualization

In [17]:
```
plt.scatter(data=salary_df, x='YearsExperience', y='Salary', s=200)
plt.title("Salary based on the Years of experience")
plt.xlabel("Years of Experience")
plt.ylabel("Salary")
plt.show()
```



The data shows linear relationship between YearsofExperience and Salary. Hence we can use Simple Linear Regression.

Independent Variable: YearsExperience

Dependent Variable: Salary

In [18]:
```
X = salary_df.loc[:, 'YearsExperience'].values

y = salary_df.loc[:, 'Salary'].values
```

Split the Data into Training and Test Data

The data will be divided into 2 parts: for training the model and for testing purpose.

Generally 70% of the rows are used for training and reamaining 30% for testing the model.

In [19]:
```
from sklearn.model_selection import train_test_split
```

In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [21]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[21]:
```
((24,), (6,), (24,), (6,))
```

In [22]:
```
X_train.reshape(-1, 1).shape
```

Out[22]:
```
(24, 1)
```

Train the model

In [23]:
```
from sklearn.linear_model import LinearRegression
```

In [24]:
```
reg_model = LinearRegression()
```

In [25]:
```
reg_model.fit(X_train.reshape(-1, 1), y_train.reshape(-1, 1))
```

```
Out[25]:  ▾ LinearRegression
          LinearRegression()
```

Find the coefficient m i.e., slope

```
In [26]: reg_model.coef_
```

```
Out[26]: array([[9312.57512673]])
```

Find the intercept i.e., c

```
In [27]: reg_model.intercept_
```

```
Out[27]: array([26780.09915063])
```

Predict the Salary for the test data

```
In [28]: y_predicted = reg_model.predict(X_test.reshape(-1, 1))
```

```
In [29]: y_predicted
```

```
Out[29]: array([[ 40748.96184072],
                [122699.62295594],
                [ 64961.65717022],
                [ 63099.14214487],
                [115249.56285456],
                [107799.50275317]])
```

```
In [30]: y_test
```

```
Out[30]: array([ 37731, 122391,  57081,  63218, 116969, 109431], dtype=int64)
```

Model Evaluation

```
In [31]: from sklearn.metrics import r2_score
```

The R Squared Value/R2 Score

```
In [32]: r_square = r2_score(y_test.reshape(-1, 1), y_predicted)
```

```
In [33]: r_square
```

```
Out[33]: 0.988169515729126
```

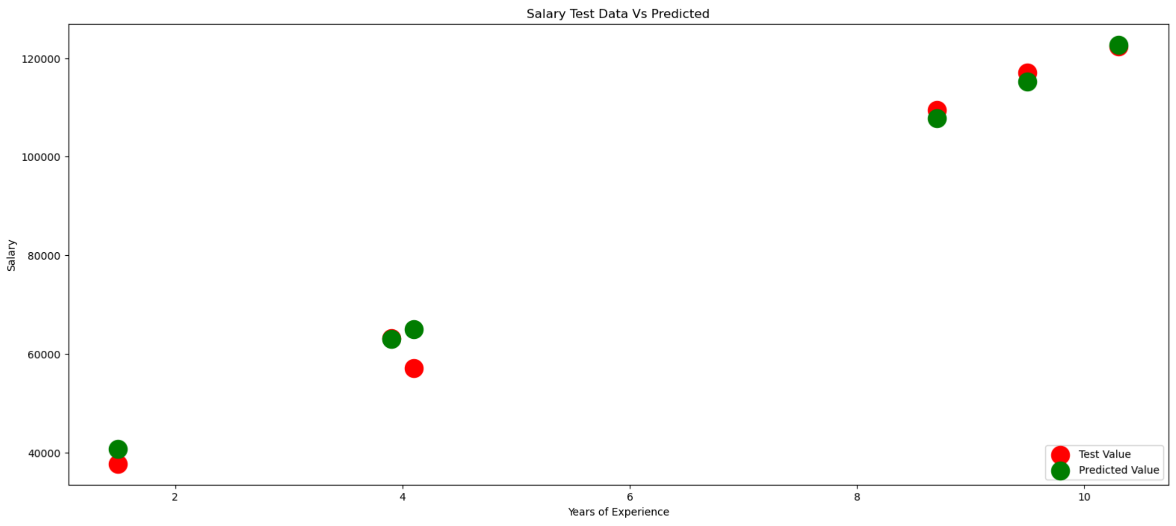
```
In [34]: print(f"Accuracy = {r_square:.2%}")
```

```
Accuracy = 98.82%
```

Data Visualization

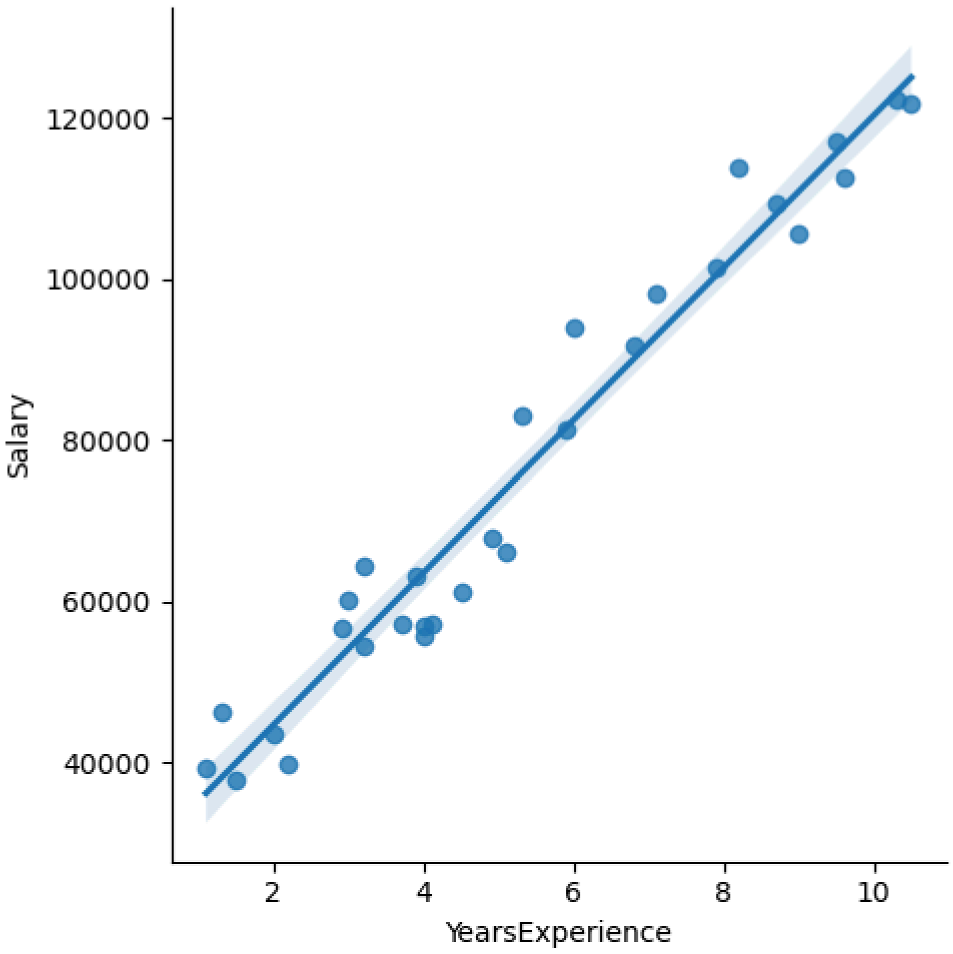
```
In [35]: plt.scatter(x=X_test, y=y_test, color='red', s=300)
         plt.scatter(x=X_test, y=y_predicted, color='green', s=300)
         plt.title("Salary Test Data Vs Predicted")
         plt.xlabel("Years of Experience")
         plt.ylabel("Salary")
         plt.legend(["Test Value", "Predicted Value"], loc="lower right")

         plt.show()
```



In [36]:
```
sns.lmplot(data=salary_df, x='YearsExperience', y='Salary')

plt.show()
```



Multiple Linear Regression

Simple Linear Regression model takes only one independent variable to predict the dependent variable value.

Multiple Linear Regression model uses multiple independent variable to predict the dependent variable value.

The equation of a Multiple Linear Regression model will be of the form:

$$y = m_1x_1 + m_2x_2 + m_3x_3 + \ldots + b$$

Here y is the dependent or target variable, x_1, x_2, x_3 are independent variables, m_1, m_2, m_3 are coefficients of the independent variables and b is the intercept.

The relationship between y and x_1, x_2, x_3 should be linear.

Case Study

Home price data prediction using area, no. of bedrooms, and age

Read the data

In [37]:
```
homeprice_df = pd.read_csv('./datasets/homeprices.csv')
```

Exploratory Data Analysis

In [38]:
```
homeprice_df.shape
```

Out[38]:
```
(6, 4)
```

In [39]:
```
homeprice_df.head()
```

Out[39]:

	area	bedrooms	age	price
0	2600	3.0	20	550000
1	3000	4.0	15	565000
2	3200	NaN	18	610000
3	3600	3.0	30	595000
4	4000	5.0	8	760000

In [40]:
```
homeprice_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   area      6 non-null      int64  
 1   bedrooms  5 non-null      float64
 2   age       6 non-null      int64  
 3   price     6 non-null      int64  
dtypes: float64(1), int64(3)
memory usage: 324.0 bytes
```

In [41]:
```
homeprice_df.duplicated().sum()
```

```
Out[41]:  0
```

```
In [42]:  homeprice_df['bedrooms'].skew()
```

```
Out[42]:  0.0
```

```
In [43]:  homeprice_df['bedrooms'].fillna(homeprice_df['bedrooms'].mean(), inplace=True)
```

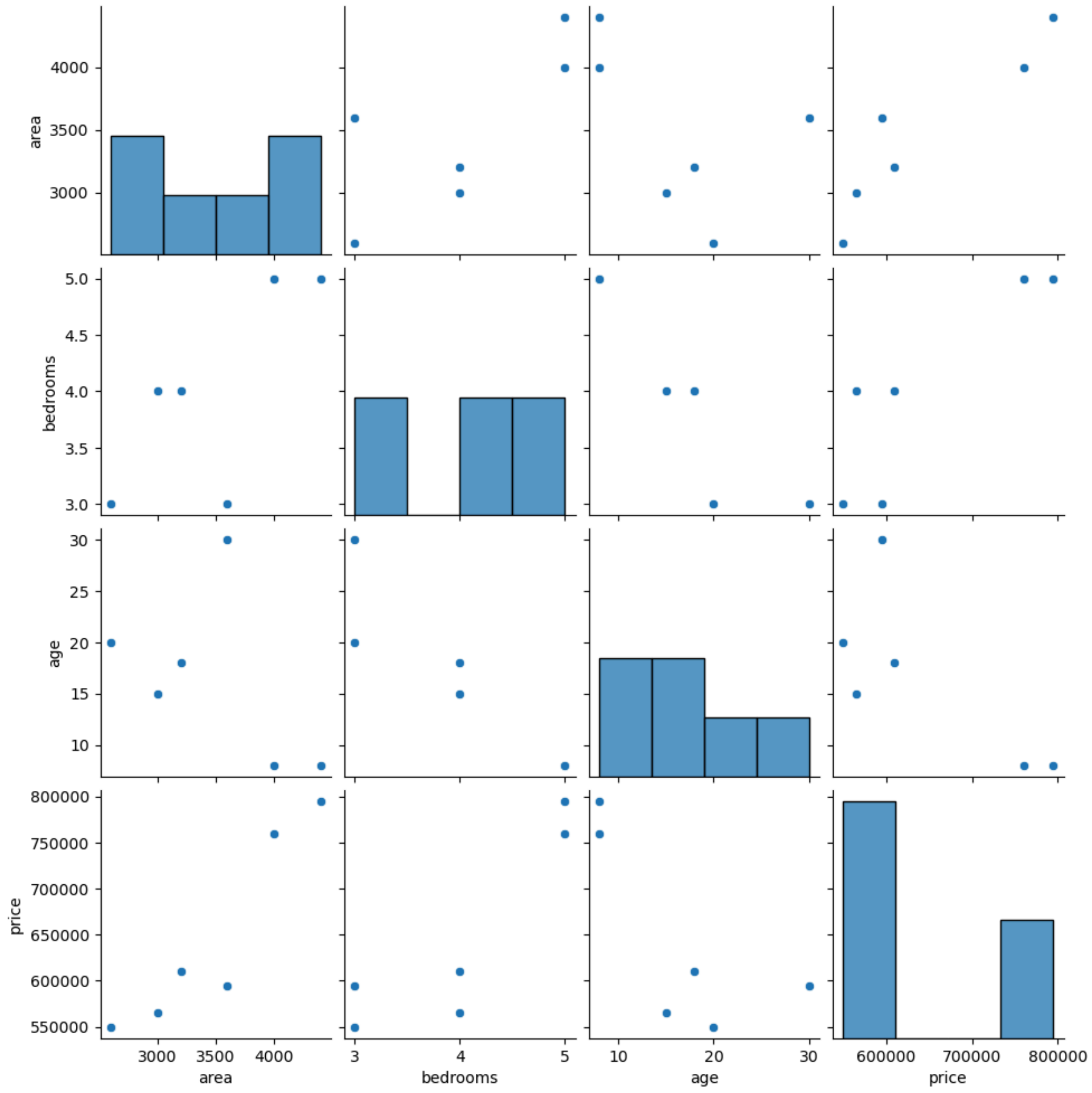
```
In [44]:  homeprice_df.isnull().sum()
```

```
Out[44]:  area        0
          bedrooms    0
          age         0
          price       0
          dtype: int64
```

Data Visualization

```
In [45]:  sns.pairplot(data=homeprice_df)

           plt.show()
```



Model Training

```
In [46]: X = homeprice_df.loc[:, 'area':'age'].values

         y = homeprice_df['price'].values
```

```
In [47]: from sklearn.linear_model import LinearRegression
```

```
In [48]: linear_model = LinearRegression()
```

```
In [49]: linear_model.fit(X, y)
```

```
Out[49]: ▾ LinearRegression

         LinearRegression()
```

```
In [50]: linear_model.coef_
```

```
Out[50]: array([   142.895644  , -48591.66405516,  -8529.30115951])
```

```
In [51]: linear_model.intercept_
```

```
Out[51]: 485561.8928233979
```

```
In [52]: y_predict = linear_model.predict(X)

         y_predict
```

```
Out[52]: array([540729.55186458, 591942.6512065 , 594933.87652772, 598332.18426826,
                745951.73926671, 803109.99686623])
```

Model Evaluation

```
In [53]: from sklearn.metrics import r2_score
```

```
In [54]: r2_score(y, y_predict)
```

```
Out[54]: 0.9760698937818199
```

```
In [55]: linear_model.score(X, y)
```

```
Out[55]: 0.9760698937818199
```

Polynomial Linear Regression

Simple and Multiple Linear Regression models are used only when the data points are in a straight line.

It is not always possible to expect the data points to be in the straight line. They may form a curved line.

Polynomial Linear Regression is used on non-linear data.

Polynomial Linear Regression is a special case of Multiple Linear Regression.

In Multiple Linear Regression, each term is without any power value but in Polynomial Linear Regression, each term is raised to a power.

The first term has power 1, the second term has power 2, and the nth term has the power n.

The equation of a Multiple Linear Regression model will be of the form:

$$y = m_1x_1 + m_2{x_2}^2 + m_3{x_3}^3 + \ldots + b$$

Here y is the dependent or target variable, x_1, x_2, x_3 are independent variables, m_1, m_2, m_3 are coefficients of the independent variables and b is the intercept.

Case Study

Bias and Variance

While building a machine learning model, the data is divided into 2 parts:

- training data, to train the model
- test data, to test the model

A machine learning model should be able to yield correct results with both training and test data. Then only the model is accurate.

While using a machine learning model, the training data points would have a lot of deviations from the regression line. This shows the inability of the model to accurately represent the relationship between the data points. This is called **bias**.

Bias represents the deviations from the original data and predicted values by the model.

On the other hand, the model may fit the test data accurately on the regression line.

The difference in fits between the datasets is called **variance**.

When the model works well on train data, it represents low bias.

When the model failed on test (new) data, it represents high variance.

Bias and variance are inversely proportional to each other. What is needed is the low bias and low variance.

If the model shows high bias and low variance, it is called **underfitting**.

If the model shows low bias and high variance, it is called **overfitting**.

Sum of Squared Residual

The Simple Linear Regression model uses a straight line to fit the data.

There are chances of some deviations from the straight line to the actual data. These deviations must be minimized.

To minimize the deviations, calculate the sum of squares of deviations and add that to the equation of the line.

The equation of the linear regression line is given by:

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

$$\epsilon = y - \beta_0 - \beta_1 x$$

$$\epsilon = y - (\beta_0 + \beta_1 x)$$

$$\epsilon = y_{actual} - y_{predict}$$

The error term exist for every observation in the data.

Squared Error:

$$\epsilon_i^2 = (y_{actual} - y_{predict})^2$$

Sum of Squared Error:

Sum of Squared Error = $\sum \epsilon_i^2$

Why to take the square of the deviations?

When deviations are taken alone, the positive and negative deviations may cancel out. But when the deviations are squared, the negative deviation becomes positive and add to the overall deviation.

Regularization

Regularization is a technique to minimize the variance or overfitting to achieve better accuracy.

When regularization is used in Linear Regression model, there would be other variations like:

- Ridge Regression
- Lasso Regression
- ElasticNET Regression

The above 3 models are developed to minimize the variance.

Ridge Regression

To control the impact on bias and variance, a regularization parameter by name λ is used.

When λ value is increased, it reduces the variance. But if the value of λ is increased too much, it will increase bias also. Hence it is importance to tune λ to the correct value to keep both the bias and the variance low.

The regularization pameter λ is also known as the penalty parameter.

Ridge Regression = $\beta_0 + \beta_1 x + \sum \epsilon_i^2 + \lambda \beta_1^2$

The term λ X square of the slope is called L2 Penalty term.

λ value can be anything from 0 to positive infinity.

The addition of penalty term is called Regularization.

When λ is 0, the Ridge Regression will be equal to Simple Linear Regression. Hence Simple Linear Regression and Ridge Regression can be used on any (train or test) of the datasets.

Ridge regularization is used when there is a lot of features in the dataset and all features have small coefficients.

Lasso Regression

LASSO stands for Least Absolute Shrinkage and Selection Operator.

Lasso Regression is same as Ridge Regression except that the penalty term is calculated by: λ X Modulus of slope.

Lasso Regression equation = $\beta_0 + \beta_1 x + \sum \epsilon_i^2 + \lambda |\beta_1|$

Lasso Regression has L1 Penalty term. L1 indicates $\lambda |\beta_1|$.

Lasso Regression offers some bias but very low variance. Hence predictions will be more accurate than Ridge Regression.

Ridge Regression considers all the columns in the dataset to predict the output. But Lasso Regression selects only the columns that influence maximum on the output. Hence Lassor Regression is generally used in feature selection.

ElasticNET Regression

ElasticNET Regression is used when a dataset has large number of columns and large volume of data.

ElasticNET Regression is a combination of Ridge and Lasso Regressions.

ElasticNET Regression equation = $\beta_0 + \beta_1 x + \sum \epsilon_i^2 + + \lambda_1 \beta_1^2 + \lambda_2 |\beta_1|$

Case Study

```
In [56]: houseprice_df = pd.read_csv("./datasets/Melbourne_housing_FULL.csv")
```

```
In [57]: houseprice_df.shape
```

```
Out[57]: (34857, 21)
```

```
In [58]: houseprice_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34857 entries, 0 to 34856
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Suburb         34857 non-null  object 
 1   Address        34857 non-null  object 
 2   Rooms          34857 non-null  int64  
 3   Type           34857 non-null  object 
 4   Price          27247 non-null  float64
 5   Method         34857 non-null  object 
 6   SellerG        34857 non-null  object 
 7   Date           34857 non-null  object 
 8   Distance       34856 non-null  float64
 9   Postcode       34856 non-null  float64
 10  Bedroom2       26640 non-null  float64
 11  Bathroom       26631 non-null  float64
 12  Car            26129 non-null  float64
 13  Landsize       23047 non-null  float64
 14  BuildingArea   13742 non-null  float64
 15  YearBuilt      15551 non-null  float64
 16  CouncilArea    34854 non-null  object 
 17  Lattitude      26881 non-null  float64
 18  Longtitude     26881 non-null  float64
 19  Regionname     34854 non-null  object 
 20  Propertycount  34854 non-null  float64
dtypes: float64(12), int64(1), object(8)
memory usage: 5.6+ MB
```

```
In [59]: houseprice_df.head()
```

Out[59]:

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distar
0	Abbotsford	68 Studley St	2	h	NaN	SS	Jellis	3/09/2016	
1	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	
2	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	
3	Abbotsford	18/659 Victoria St	3	u	NaN	VB	Rounds	4/02/2016	
4	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	

5 rows × 21 columns

In [60]:
```
houseprice_df = houseprice_df.loc[:, ['Suburb', 'Rooms', 'Type', 'Method', 'Sell
                                       'Bedroom2', 'Bathroom', 'Car', 'Landsize',
                                       'Regionname', 'Propertycount',
                                       'Price']]
```

In [61]:
```
houseprice_df.shape
```

Out[61]:
```
(34857, 15)
```

In [62]:
```
houseprice_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34857 entries, 0 to 34856
Data columns (total 15 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Suburb         34857 non-null  object 
 1   Rooms          34857 non-null  int64  
 2   Type           34857 non-null  object 
 3   Method         34857 non-null  object 
 4   SellerG        34857 non-null  object 
 5   Distance       34856 non-null  float64
 6   Bedroom2       26640 non-null  float64
 7   Bathroom       26631 non-null  float64
 8   Car            26129 non-null  float64
 9   Landsize       23047 non-null  float64
 10  BuildingArea   13742 non-null  float64
 11  CouncilArea    34854 non-null  object 
 12  Regionname     34854 non-null  object 
 13  Propertycount  34854 non-null  float64
 14  Price          27247 non-null  float64
dtypes: float64(8), int64(1), object(6)
memory usage: 4.0+ MB
```

In [63]:
```
houseprice_df['Distance'].fillna(0, inplace=True)

houseprice_df['Bedroom2'].fillna(0, inplace=True)
```

```
houseprice_df['Bathroom'].fillna(0, inplace=True)

houseprice_df['Car'].fillna(0, inplace=True)

houseprice_df['Propertycount'].fillna(0, inplace=True)
```

In [64]:
```
houseprice_df['Landsize'].fillna(houseprice_df['Landsize'].mean(), inplace=True)

houseprice_df['BuildingArea'].fillna(houseprice_df['BuildingArea'].mean(), inpla
```

In [65]:
```
houseprice_df.dropna(inplace=True)
```

In [66]:
```
houseprice_df.isnull().sum()
```

Out[66]:
```
Suburb           0
Rooms            0
Type             0
Method           0
SellerG          0
Distance         0
Bedroom2         0
Bathroom         0
Car              0
Landsize         0
BuildingArea     0
CouncilArea      0
Regionname       0
Propertycount    0
Price            0
dtype: int64
```

In [67]:
```
houseprice_df.duplicated().sum()
```

Out[67]:
```
50
```

In [68]:
```
houseprice_df = pd.get_dummies(data=houseprice_df, drop_first=True, dtype=np.int
```

In [69]:
```
houseprice_df.head()
```

Out[69]:

	Rooms	Distance	Bedroom2	Bathroom	Car	Landsize	BuildingArea	Propertycount
1	2	2.5	2.0	1.0	1.0	202.0	160.2564	4019.0
2	2	2.5	2.0	1.0	0.0	156.0	79.0000	4019.0
4	3	2.5	3.0	2.0	0.0	134.0	150.0000	4019.0
5	3	2.5	3.0	2.0	1.0	94.0	160.2564	4019.0
6	4	2.5	3.0	1.0	2.0	120.0	142.0000	4019.0

5 rows × 745 columns

In [70]:
```
y = houseprice_df['Price']
```

```
X = houseprice_df.drop('Price', axis=1)
```

In [71]:
```
from sklearn.model_selection import train_test_split
```

In [72]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
```

Simple Linear Regression

In [73]:
```
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
```

In [74]:
```
linear_model = LinearRegression()
```

In [75]:
```
linear_model.fit(X_train, y_train)
```

Out[75]:
```
▾ LinearRegression

LinearRegression()
```

In [76]:
```
linear_model.score(X_train, y_train)
```

Out[76]:
```
0.6827792395792723
```

In [77]:
```
linear_model.score(X_test, y_test)
```

Out[77]:
```
0.13853683161540487
```

Ridge Regression

In [78]:
```
ridge_model = Ridge(alpha=1.0, max_iter=100)
```

In [79]:
```
ridge_model.fit(X_train, y_train)
```

Out[79]:
```
▾ Ridge

Ridge(max_iter=100)
```

In [80]:
```
ridge_model.score(X_train, y_train)
```

Out[80]:
```
0.6796668251040214
```

In [81]:
```
ridge_model.score(X_test, y_test)
```

Out[81]:
```
0.6701765758295284
```

Lasso Regression

In [82]:
```
lasso_model = Lasso(alpha=50, max_iter=100)
```

In [83]:
```
lasso_model.fit(X_train, y_train)
```

```
Out[83]:
▾ Lasso
Lasso(alpha=50, max_iter=100)
```

```
In [84]: lasso_model.score(X_train, y_train)
```

```
Out[84]: 0.6767356948457683
```

```
In [85]: lasso_model.score(X_test, y_test)
```

```
Out[85]: 0.6637669697137103
```

ElasticNET Regression

```
In [86]: elastic_model = ElasticNet(alpha=100, l1_ratio=0.5)
```

```
In [87]: elastic_model.fit(X_train, y_train)
```

```
Out[87]:
▾ ElasticNet
ElasticNet(alpha=100)
```

```
In [88]: elastic_model.score(X_train, y_train)
```

```
Out[88]: 0.06482342181996403
```

```
In [89]: elastic_model.score(X_test, y_test)
```

```
Out[89]: 0.07381459514861177
```

```
In [ ]:
```