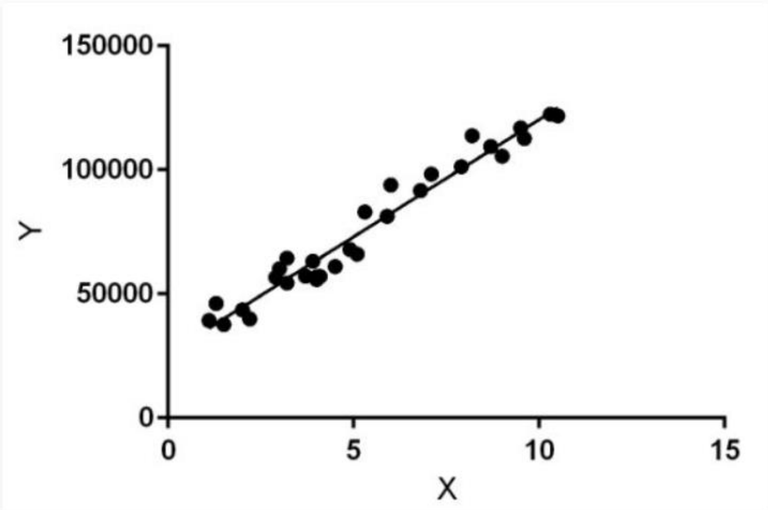
# Linear Regression

**Linear Regression** is a machine learning algorithm based on **supervised learning**. It performs a **regression task**. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting. Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used.




Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
In the figure above, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.

### Hypothesis function for Linear Regression :

$$\mathbf{y = \theta_1 + \theta_2.x}$$

While training the model we are given :
**x:** input training data (univariate – one input variable(parameter))
**y:** labels to data (supervised learning)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best $\theta_1$ and $\theta_2$ values.
**$\theta_1$:** intercept
**$\theta_2$:** coefficient of x
Once we find the best $\theta_1$ and $\theta_2$ values, we get the best fit line. So when we are finally using our model for prediction, it will predict the value of y for the input value of x.
**How to update $\theta_1$ and $\theta_2$ values to get the best fit line ?**

**Cost Function (J):**

By achieving the best-fit regression line, the model aims to predict y value such that the error difference between predicted value and true value is minimum. So, it is very important to update the $\theta_1$ and $\theta_2$ values, to reach the best value that minimize the error between predicted y value (pred) and true y value (y).

$$minimize \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

$$J = \frac{1}{n} \sum_{i=1}^{n} (pred_i - y_i)^2$$

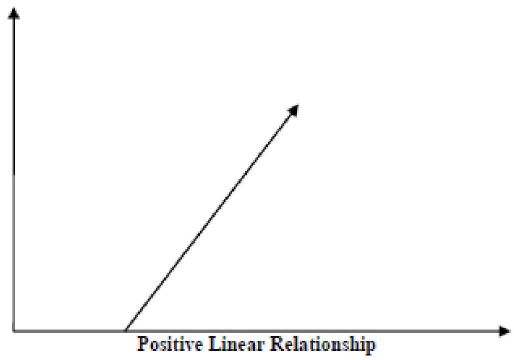
Cost function(J) of Linear Regression is the **Root Mean Squared Error (RMSE)** between predicted y value (pred) and true y value (y).

**Gradient Descent:**

To update $\theta_1$ and $\theta_2$ values in order to reduce Cost function (minimizing RMSE value) and achieving the best fit line the model uses Gradient Descent. The idea is to start with random $\theta_1$ and $\theta_2$ values and then iteratively updating the values, reaching minimum cost.

Positive Linear Relationship

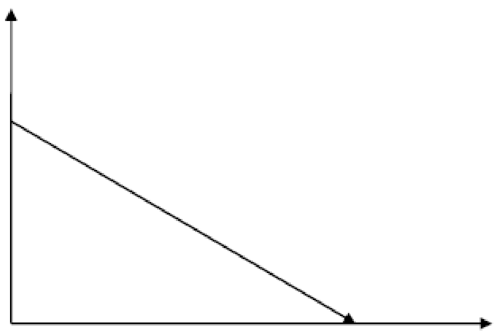
A linear relationship will be called positive if both independent and dependent variable increases. It can be understood with the help of following graph −



Positive Linear Relationship

Negative Linear relationship

A linear relationship will be called positive if independent increases and dependent variable decreases. It can be understood with the help of following graph −



Negative Linear Relationship

**Types of Linear Regression**

Linear regression is of the following two types −

- Simple Linear Regression
- Multiple Linear Regression

**Assumptions**

The following are some assumptions about dataset that is made by Linear Regression model −

**Multi-collinearity −** Linear regression model assumes that there is very little or no multi-collinearity in the data. Basically, multi-collinearity occurs when the independent variables or features have dependency in them.

**Auto-correlation −** Another assumption Linear regression model assumes is that there is very little or no auto-correlation in the data. Basically, auto-correlation occurs when there is dependency between residual errors.

**Relationship between variables −** Linear regression model assumes that the relationship between response and feature variables must be linear.

## Linear Regression with Python

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 is sold at
- 'Address': Address for the house

## 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

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

## Check out the Data

```
USAhousing = pd.read_csv('USA_Housing.csv')
```

```
USAhousing.head()
```

| | 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 Michael Ferry Apt. 674\nLau |
| 1 | 79248.642455 | 6.002900 | 6.730821 | 3.09 | 40173.072174 | 1.505891e+06 | 188 Johnson Views Suite Kath |
| 2 | 61287.067179 | 5.865890 | 8.512727 | 5.13 | 36882.159400 | 1.058988e+06 | 9127 Elizabeth Stravenue\nDani |
| 3 | 63345.240046 | 7.188236 | 5.586729 | 3.26 | 34310.242831 | 1.260617e+06 | USS Barnett\nFPC |
| 4 | 59982.197226 | 5.040555 | 7.839388 | 4.23 | 26354.109472 | 6.309435e+05 | USNS Raymond\nFPC |

```
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
```

```
USAhousing.describe()
```

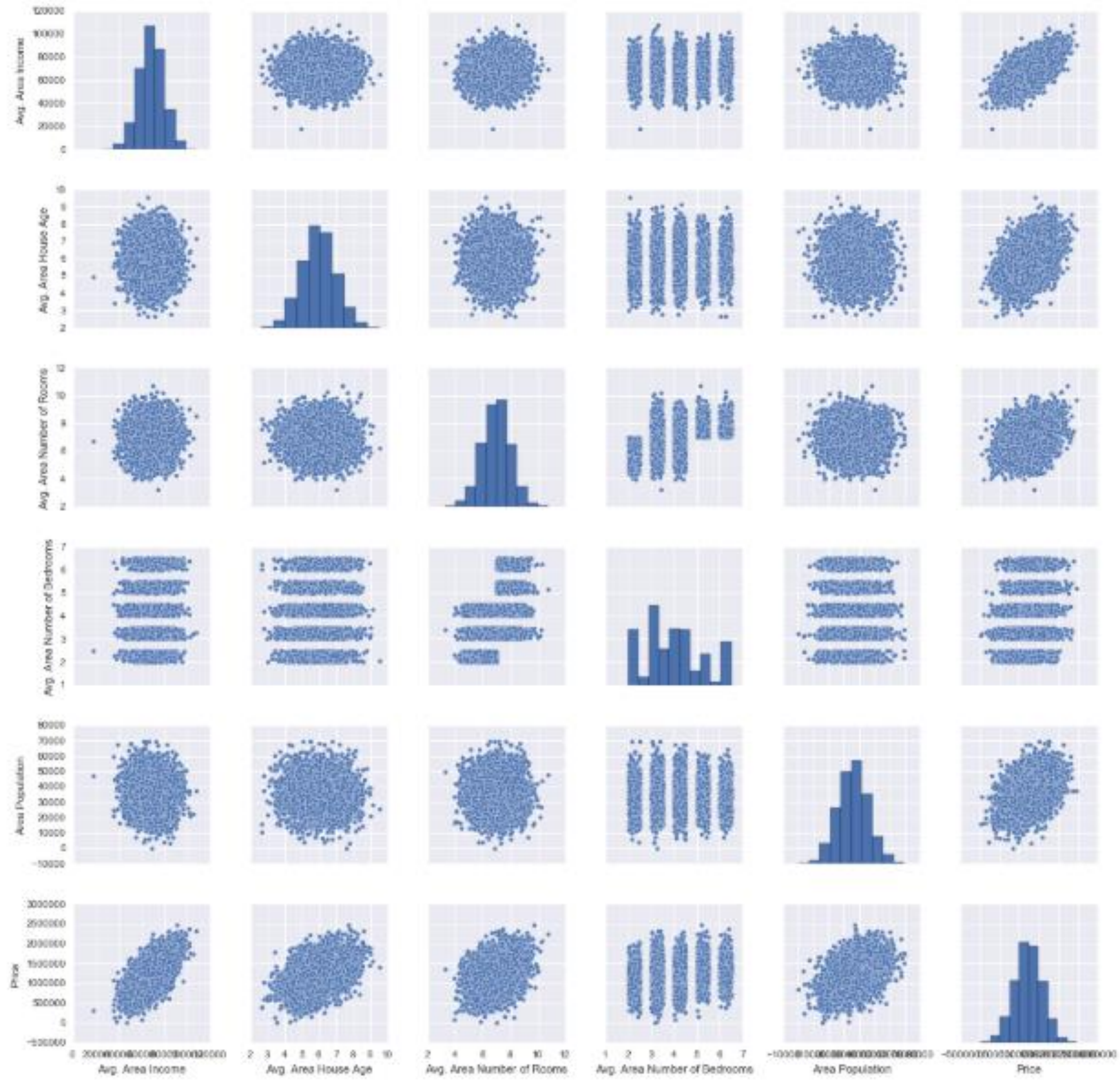
| | Avg. Area Income | Avg. Area House Age | Avg. Area Number of Rooms | Avg. Area Number of Bedrooms | Area Population | Price |
|---|---|---|---|---|---|---|
| count | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5000.000000 | 5.000000e+03 |
| mean | 68583.108984 | 5.977222 | 6.987792 | 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 |

## Exploratory Data Analysis

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

```
sns.pairplot(USAhousing)
```

```
<seaborn.axisgrid.PairGrid at 0x13e898358>
```




### 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.

## 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

```
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.

```
from sklearn.model_selection import train_test_split
```

```
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

```
from sklearn.linear_model import LinearRegression
```

```
lm = LinearRegression()
```

```
lm.fit(X_train,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

## Model Evaluation

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

```
# print the intercept
print(lm.intercept_)
```

```
-2640159.79685
```

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

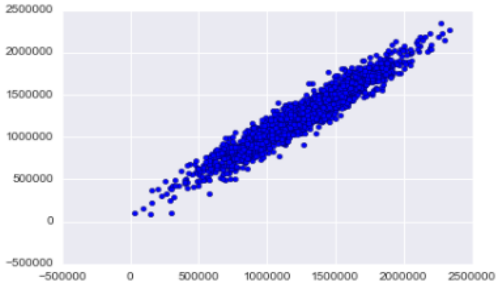
| | Coefficient |
|---|---|
| **Avg. Area Income** | 21.528276 |
| **Avg. Area House Age** | 164883.282027 |
| **Avg. Area Number of Rooms** | 122368.678027 |
| **Avg. Area Number of Bedrooms** | 2233.801864 |
| **Area Population** | 15.150420 |

## Predictions for the Model

```
predictions = lm.predict(X_test)
```

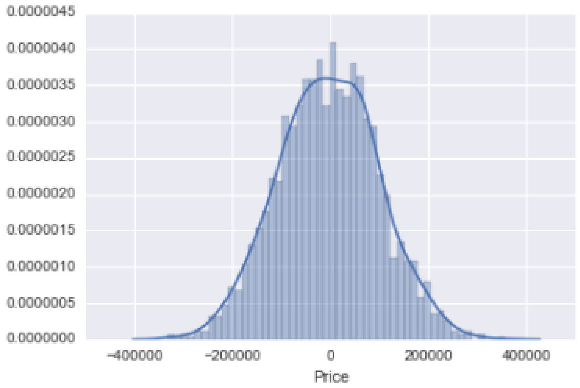
```
plt.scatter(y_test,predictions)
```

```
<matplotlib.collections.PathCollection at 0x142622c88>
```



**Residual Histogram**

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




# 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.

```
from sklearn import metrics
```

```
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: 82288.2225191
MSE: 10460958907.2
RMSE: 102278.829223
```