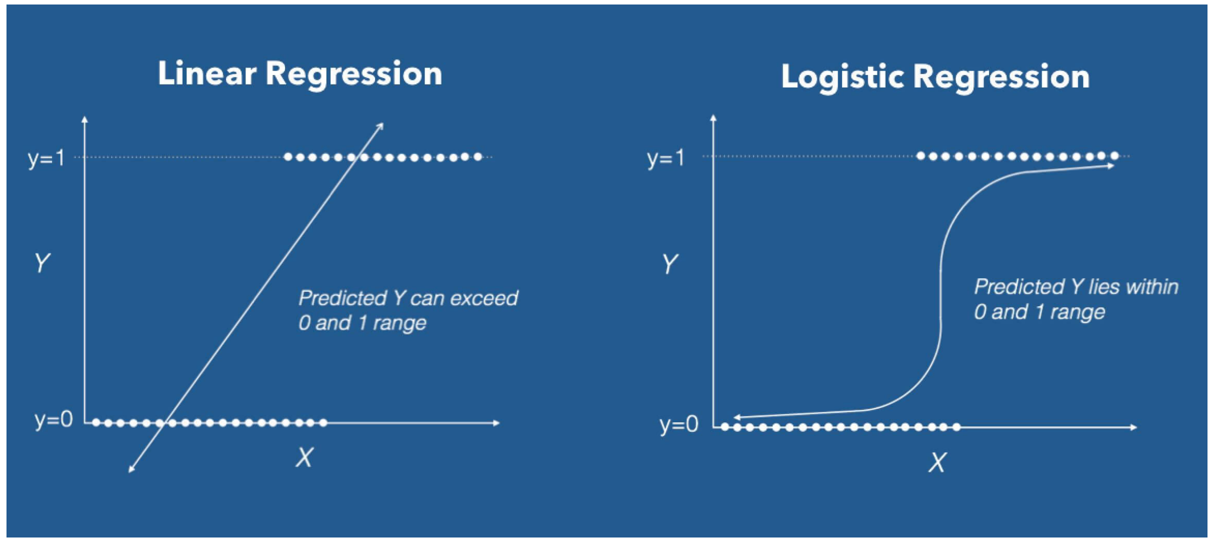
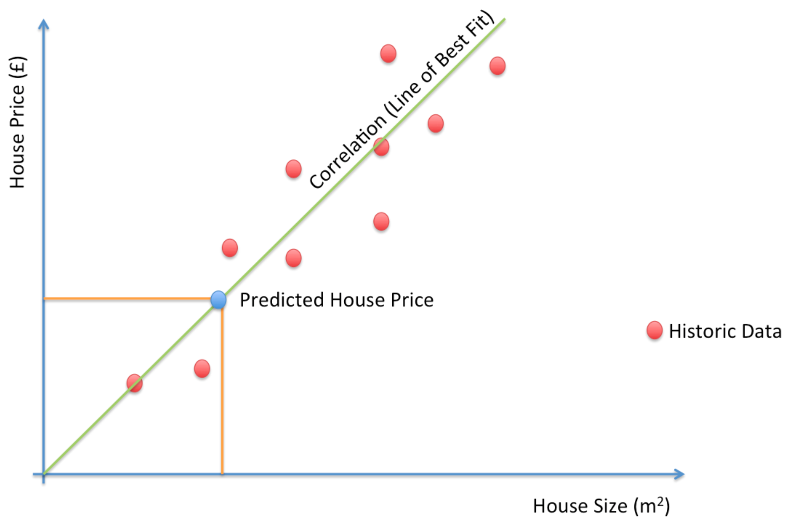
LESSON 3: LINEAR REGRESSION



This lecture was refered by [machinelearningcoban.com]

1. Linear regression introduction



With an example of **House price prediction** problem, we have 3 features of a house:

- x_1 is the size of the house (in m^2)
- x_2 is the number of bedrooms in the house (in rooms)
- x_3 is the distance from the house to the city center (in km) \

and the label price of the house y

We have to build a function to calculate the price of the house from above features $x = [x_1, x_2, x_3]$.

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

- $w = [w_0, w_1, w_2, w_3]^T$ is parameters of model
- $\hat{y}$ is a prediction of model and we expect that $\hat{y}$ and y are almost similar.

We use a **linear** function and this is **regression** problem. That's why we call **LINEAR REGRESSION**.

2. Loss function and Optimizer

Generalize our problem to n features, we have a set of features $x = [x_1, x_2, \dots, x_n]$.

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$$

Given M training samples $X = x^{(1)}, x^{(2)}, \dots, x^{(M)}$, linear regression finds w that minimizes the difference between $\hat{y}$ and y.

To minimize the difference, we have to build a **LOSS FUNCTION** and for linear regression, we use Mean Square Error (MSE).

$$\begin{aligned} MSE(w) &= \frac{1}{M}\frac{1}{2}\sum_{i=1}^{M}(\hat{y}^{(i)} - y^{(i)})^2 \\ &= \frac{1}{M}\frac{1}{2}\sum_{i=1}^{M}(x^{(i)}w - y^{(i)})^2 \\ &= \frac{1}{M}\frac{1}{2}(Xw - y)^2 \end{aligned}$$

We need to find the w to minimize the value of MSE function and this w called an ***optimal point***.

$$w^* = \arg\min_w \mathcal{L}(w)$$

To find the optimal point w^*, we solve the equation:

$$\begin{aligned} \frac{\partial MSE}{\partial w} &= \frac{1}{M}\frac{1}{2} \cdot 2 \cdot (Xw - y) \cdot X^T \\ &= \frac{1}{M} X^T \cdot (Xw - y) = 0 \end{aligned}$$

We can have the above equation because we have:

$$\frac{\partial Ax + b}{\partial x} = A^T$$

and $(Xw - y)$ is a scalar so we can use the commutative principle.

Back to the equation

$$\begin{aligned} X^T \cdot (Xw - y) &= 0 \\ X^T X w &= X^T y \\ w &= (X^T X)^{-1} X^T y \end{aligned}$$

Finally, we have $w^* = (X^T X)^{-1} X^T y$ is the solution of $\frac{\partial MSE}{\partial w} = 0$

3. Implementation example

3.1. Prepare library and data

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

```python
import numpy as np
import seaborn as sns

sns.set()
```

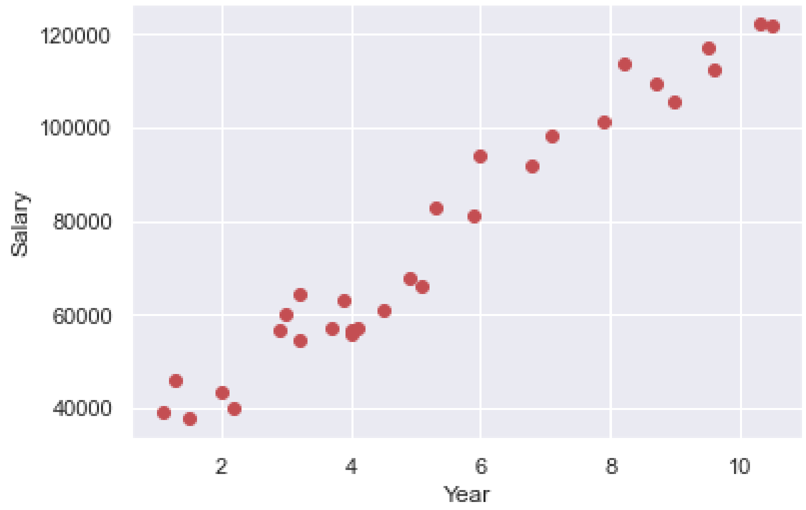
In [2]:
```python
df = pd.read_csv('../data/linear_regression_salary_data.csv')
df
```

Out[2]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0
24	8.7	109431.0
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

In [3]:
```python
year = df.YearsExperience.to_list()
salary = df.Salary.to_list()
```

In [4]:
```
plt.plot(year, salary, 'ro')
plt.xlabel('Year')
plt.ylabel('Salary')
plt.show()
```



In [5]:
```
X = np.array([year])
X.shape
```

Out[5]:
```
(1, 30)
```

In [6]:
```
y = np.array([salary])
y.shape
```

Out[6]:
```
(1, 30)
```

In [7]:
```
def prepare_X_ones(X):
    x_1 = np.ones((1, X.shape[1]))
    X = np.concatenate((x_1, X), axis=0).T
    return X
```

In [8]:
```
X
```

Out[8]:
```
array([[ 1.1,  1.3,  1.5,  2. ,  2.2,  2.9,  3. ,  3.2,  3.2,  3.7,  3.9,
         4. ,  4. ,  4.1,  4.5,  4.9,  5.1,  5.3,  5.9,  6. ,  6.8,  7.1,
         7.9,  8.2,  8.7,  9. ,  9.5,  9.6, 10.3, 10.5]])
```

In [9]:
```
X.shape
```

Out[9]:
```
(1, 30)
```

In [10]:
```
X_with_1 = prepare_X_ones(X)
```

In [11]:
```
X_with_1.shape
```

Out[11]:
```
(30, 2)
```

In [12]:
```
X_with_1
```

Out[12]:
```
array([[ 1. ,  1.1],
       [ 1. ,  1.3],
       [ 1. ,  1.5],
       [ 1. ,  2. ],
       [ 1. ,  2.2],
       [ 1. ,  2.9],
       [ 1. ,  3. ],
       [ 1. ,  3.2],
       [ 1. ,  3.2],
       [ 1. ,  3.7],
       [ 1. ,  3.9],
       [ 1. ,  4. ],
       [ 1. ,  4. ],
       [ 1. ,  4.1],
       [ 1. ,  4.5],
       [ 1. ,  4.9],
       [ 1. ,  5.1],
       [ 1. ,  5.3],
       [ 1. ,  5.9],
       [ 1. ,  6. ],
       [ 1. ,  6.8],
       [ 1. ,  7.1],
       [ 1. ,  7.9],
       [ 1. ,  8.2],
       [ 1. ,  8.7],
       [ 1. ,  9. ],
       [ 1. ,  9.5],
       [ 1. ,  9.6],
       [ 1. , 10.3],
       [ 1. , 10.5]])
```

In [13]:
```
y
```

Out[13]:
```
array([[ 39343.,  46205.,  37731.,  43525.,  39891.,  56642.,  60150.,
         54445.,  64445.,  57189.,  63218.,  55794.,  56957.,  57081.,
         61111.,  67938.,  66029.,  83088.,  81363.,  93940.,  91738.,
         98273., 101302., 113812., 109431., 105582., 116969., 112635.,
        122391., 121872.]])
```

In [14]:
```
y.T
```

Out[14]:
```
array([[ 39343.],
       [ 46205.],
       [ 37731.],
       [ 43525.],
       [ 39891.],
       [ 56642.],
       [ 60150.],
       [ 54445.],
       [ 64445.],
       [ 57189.],
       [ 63218.],
       [ 55794.],
       [ 56957.],
       [ 57081.],
       [ 61111.],
       [ 67938.],
       [ 66029.],
       [ 83088.],
       [ 81363.],
       [ 93940.],
       [ 91738.],
       [ 98273.],
       [101302.],
       [113812.],
       [109431.],
       [105582.],
       [116969.],
       [112635.],
       [122391.],
       [121872.]])
```

3.2. Implement from scratch

$w^* = (X^T X)^{-1} X^T y$

In [15]:
```
class MyLinearRegression():
    def __call__(self, X, y):
        A = np.dot(X.T, X)
        b = np.dot(X.T, y.T)
        self.w = np.dot(np.linalg.pinv(A), b)

    def display(self, X, y):
        reg_x = np.linspace(0.5, 11, 2)
        reg_y = self.w[0][0] + self.w[1][0] * reg_x

        plt.plot(X, y, 'ro')
        plt.plot(reg_x, reg_y)
        plt.xlabel('Year')
        plt.ylabel('Salary')
        plt.show()
```

In [16]:
```
my_linear_regression = MyLinearRegression()
```

In [17]:
```
my_linear_regression(X_with_1, y)
```

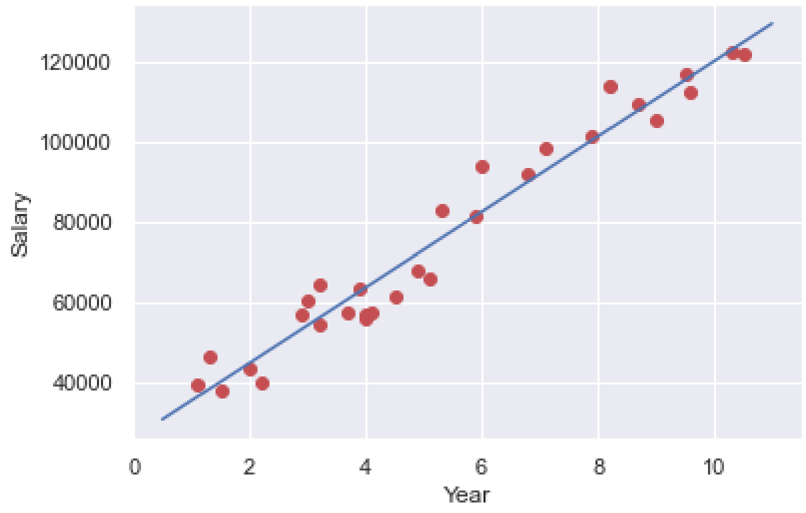
In [18]:
```
w = my_linear_regression.w
w.shape
```

Out[18]:
```
(2, 1)
```

In [19]:
```
w
```

Out[19]:
```
array([[25792.20019867],
       [ 9449.96232146]])
```

In [20]:
```
my_linear_regression.display(X, y)
```



3.3. Use `sklearn`

In [21]:
```
from sklearn.linear_model import LinearRegression
```

In [22]:
```
sklearn_linear_regression = LinearRegression(fit_intercept=False)
sklearn_linear_regression
```

Out[22]:
```
LinearRegression(fit_intercept=False)
```

In [23]:
```
sklearn_linear_regression.fit(X_with_1, y.T)
```

Out[23]:
```
LinearRegression(fit_intercept=False)
```

In [24]:
```
sklearn_linear_regression.coef_
```

Out[24]:
```
array([[25792.20019867,  9449.96232146]])
```

In [25]:
```
w
```

Out[25]:
```
array([[25792.20019867],
       [ 9449.96232146]])
```

4. Homework

4.1. Compare regression evaluation metrics: MSE and RMSE, MAE, R-Squared (Coefficient of determination)

4.2. What is regularization? Compare L1 and L2 regularization

In [ ]:
```

```