Coefficient of Correlation: It denotes the relationship between independent and dependent(target) variable.

It is denoted by R.

R value is from +1 to -1.

In [1]:
```
#First lets understand variance:
```

Formula

$$S^2 = \frac{\sum (x_i - \bar{x})^2}{n - 1}$$

S^2 = sample variance

x_i = the value of the one observation

$\bar{x}$ = the mean value of all observations

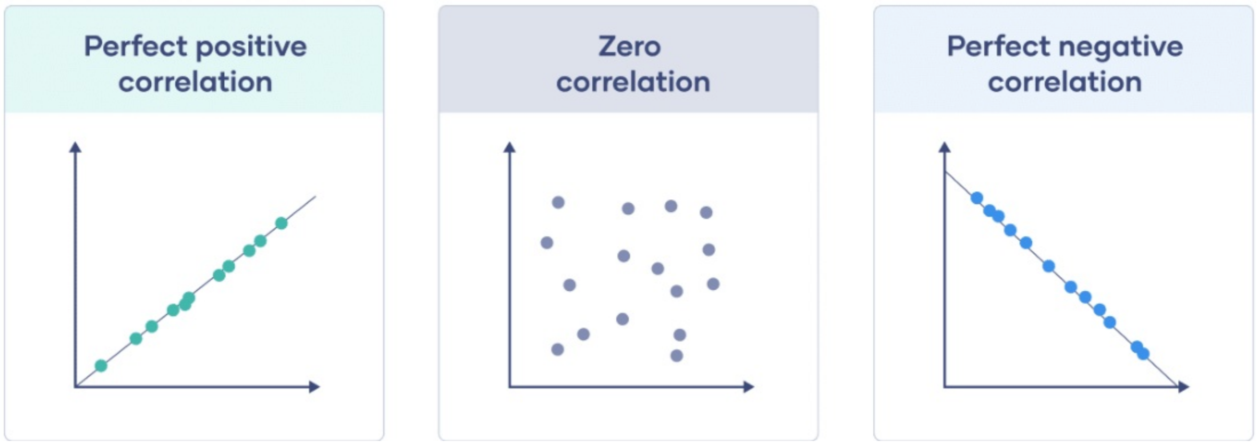
n = the number of observations

The term variance refers to a statistical measurement of the spread between numbers in a data set. More specifically, variance measures how far each number in the set is from the mean (average)

Covariance: Note: Independent variable and dependent variable should be covariant but data inside independent variable should not be covariant.

So, what is coefficient of Correlation?

Correlation coefficient value	Correlation type	Meaning
1	Perfect positive correlation	When one variable changes, the other variables change in the same direction.
0	Zero correlation	There is no relationship between the variables.
-1	Perfect negative correlation	When one variable changes, the other variables change in the opposite direction.



In [ ]:
```
import numpy as np
```

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

```
In [2]: df=pd.read_csv(r"C:\Users\USER\Downloads\placement.csv") #lets used same file which we have used in tutorial 1
        df.head()
```

Out[2]:

	cgpa	package
0	6.89	3.26
1	5.12	1.98
2	7.82	3.25
3	7.42	3.67
4	6.94	3.57

Finding R value - cofficient of correlation

```
In [3]: R = np.corrcoef(df['cgpa'], df['package'])[0,1]
        print(R)
```

```
0.8806924693700491
```

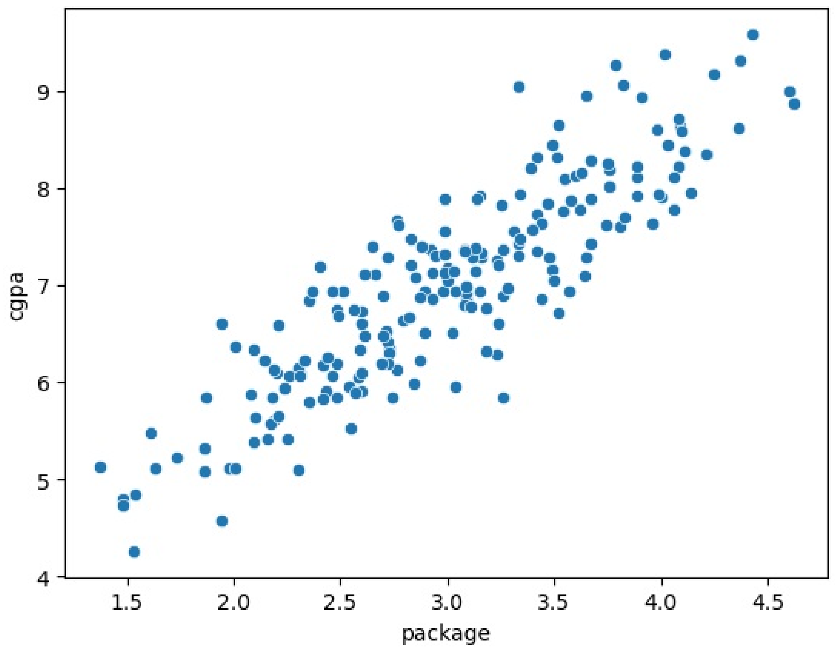
- R value of 0.88 indicates a strong positive correlation between the two variables. This means as one variable increases, the other variable tends to increase as well.
- Knowing there is a strong correlation is very useful before creating a linear regression model between the variables:

1. It indicates that a linear model is appropriate. Linear regression assumes there is a linear relationship between variables. The high R value supports this assumption.
2. It suggests the linear model is likely to fit the data well and be meaningful. A strong correlation means much of the variability is shared between variables. This shared variance will be modeled by regression.
3. It tells how much of the variability can be modeled through linear regression. R^2 represents explained variance. So for an R value of 0.88, the R^2 is 0.88^2 = 0.7744. This means a linear model with our variables can explain about 77% of variance in our data.

```
In [4]: import seaborn as sns
```

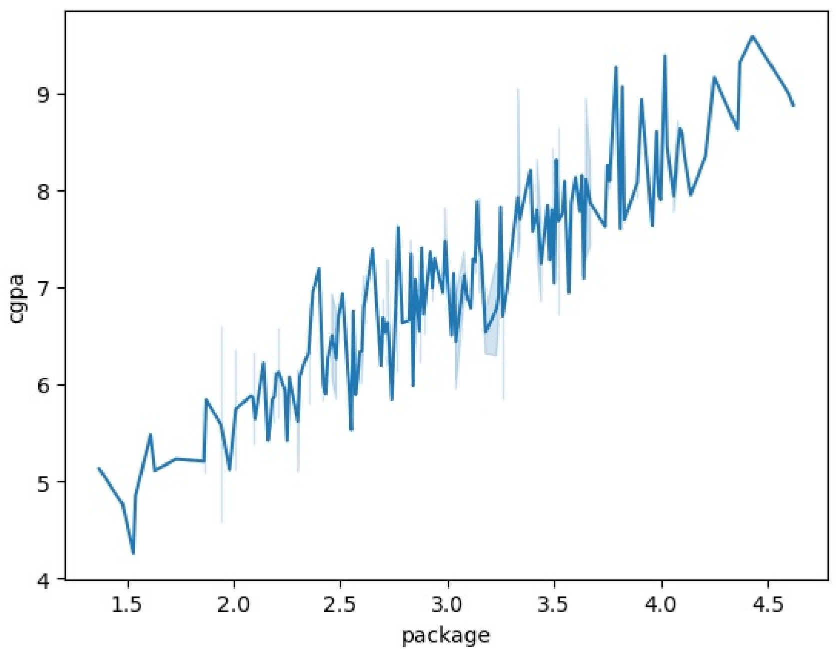
```
In [12]: sns.scatterplot(data=df, x="package", y="cgpa")
```

```
Out[12]: <Axes: xlabel='package', ylabel='cgpa'>
```



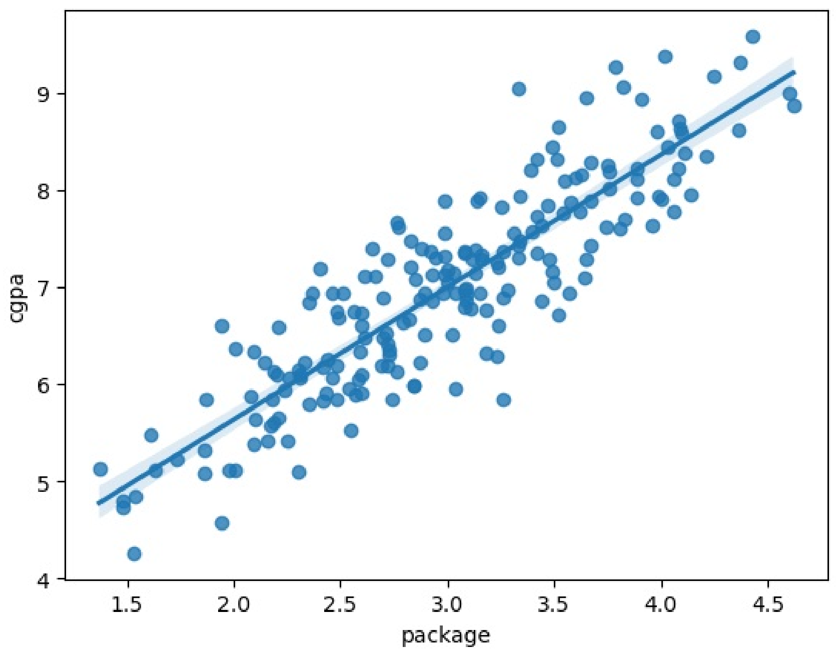
```
In [13]: sns.lineplot(data=df, x="package", y="cgpa")
```

```
Out[13]: <Axes: xlabel='package', ylabel='cgpa'>
```



```
In [15]: sns.regplot(data=df, x="package", y="cgpa")
```

```
Out[15]: <Axes: xlabel='package', ylabel='cgpa'>
```



I was just checking all graph to confirm that data is almost linear. Because perfect linear cannot be in real set data.

NOTE: Best fit line is prediction value, the dot which we saw in regplot is actual value. So the dot which is far from best fit line is known as error but, not in wrong term. And Error is nothing but variance.

HOW to find m and c or b __ y=mx+b

Two types : Closed form and non closed form. Closed form is just a mathematical expression with well known function.

From closed form, we can use direct formula 'ols' - odinary least squares. Non closed form or technique is Gradient Descent.

m= (y-b)/x

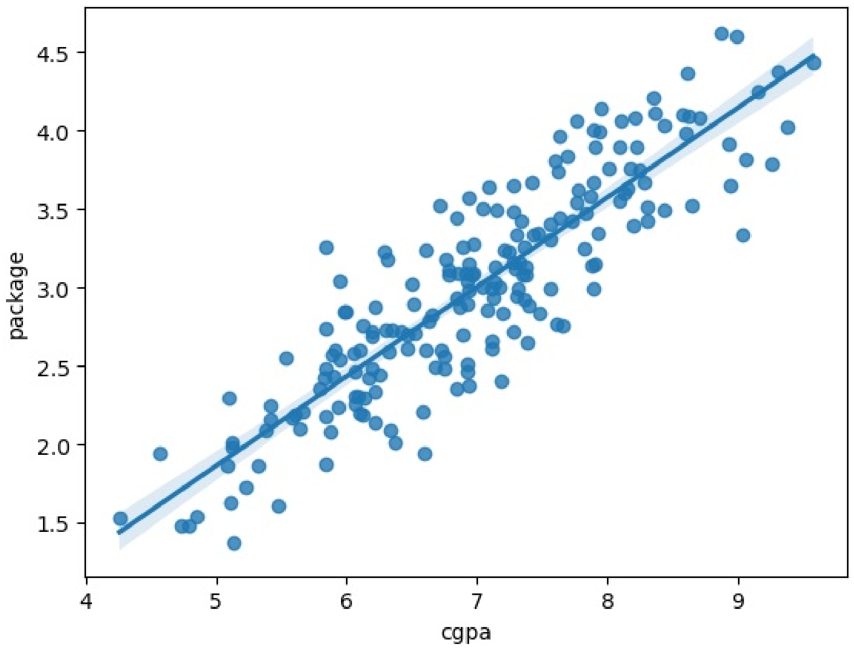
Now, how to find m?

$$b_1 = \frac{\sum_{i=1}^{n}((x_i - \bar{x})(y_i - \bar{y}))}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Lets go in depth mathematically

```
In [17]: # So, now think this below is our line
         sns.regplot(data=df, x="cgpa", y="package")
```

```
Out[17]: <Axes: xlabel='cgpa', ylabel='package'>
```



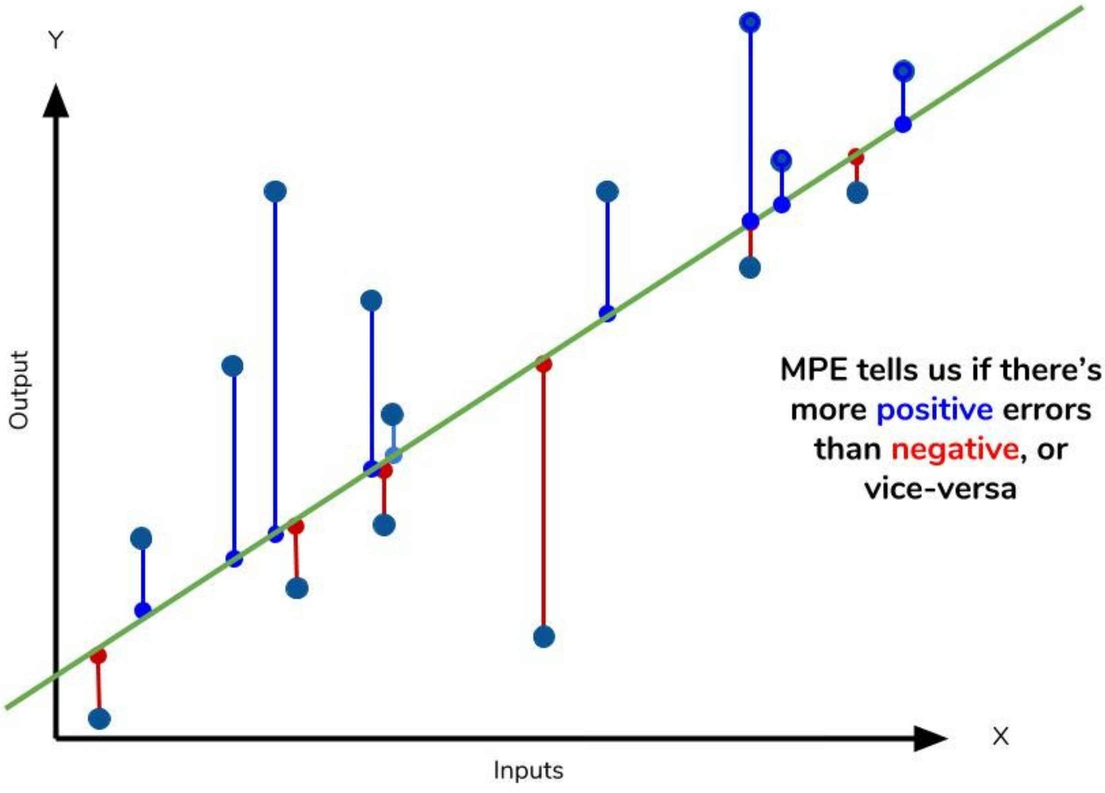
So, straight line is best fit line, and circle we can see around line is actual data, If we get this graph in machine learning prediction. Then my machine is predicting that the line is actual prediction, where for this much cgpa, this much should be package according to line but actual data may be slightly up and down of line as denoting by circle, so the distance between each specefic dot and line is error.

Now think: This error function mainly depend on m & b, so we need to look for that value of m & b which minimize the error and distance between line and dot should be minimum.

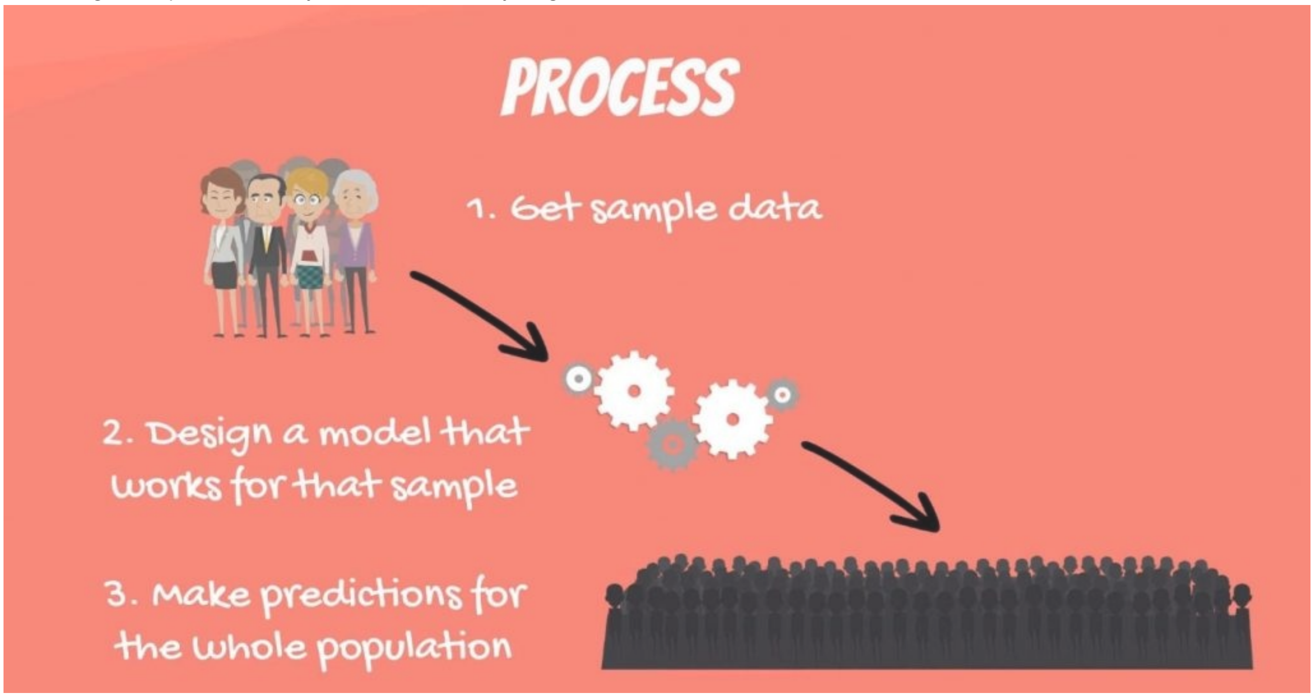
What is formula of error function in regression? This formula is showing average error, if we need total error, remove n from formula:

$$\text{Error}_{(m,b)} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (mx_i + b))^2$$

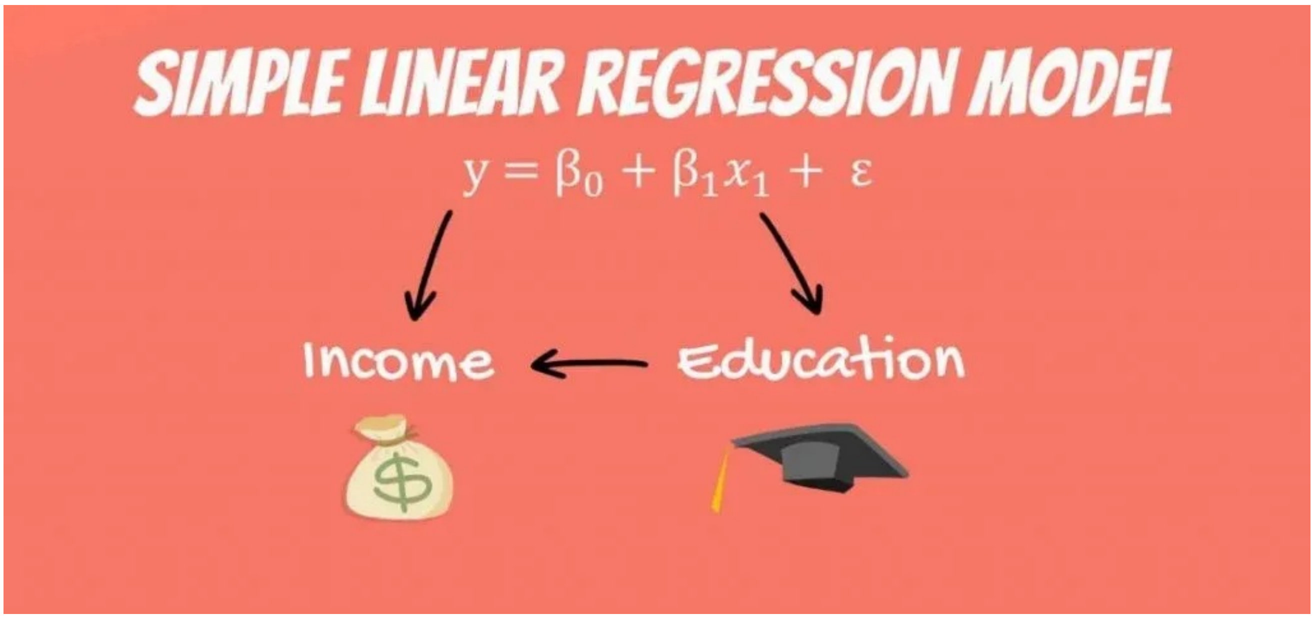
Just example:


Y
Output
X
Inputs
MPE tells us if there's more positive errors than negative, or vice-versa

Now lets get sample data and try to understand everything:


PROCESS
1. Get sample data
2. Design a model that works for that sample
3. Make predictions for the whole population

Think we are predicting income (Y) based on education (E)

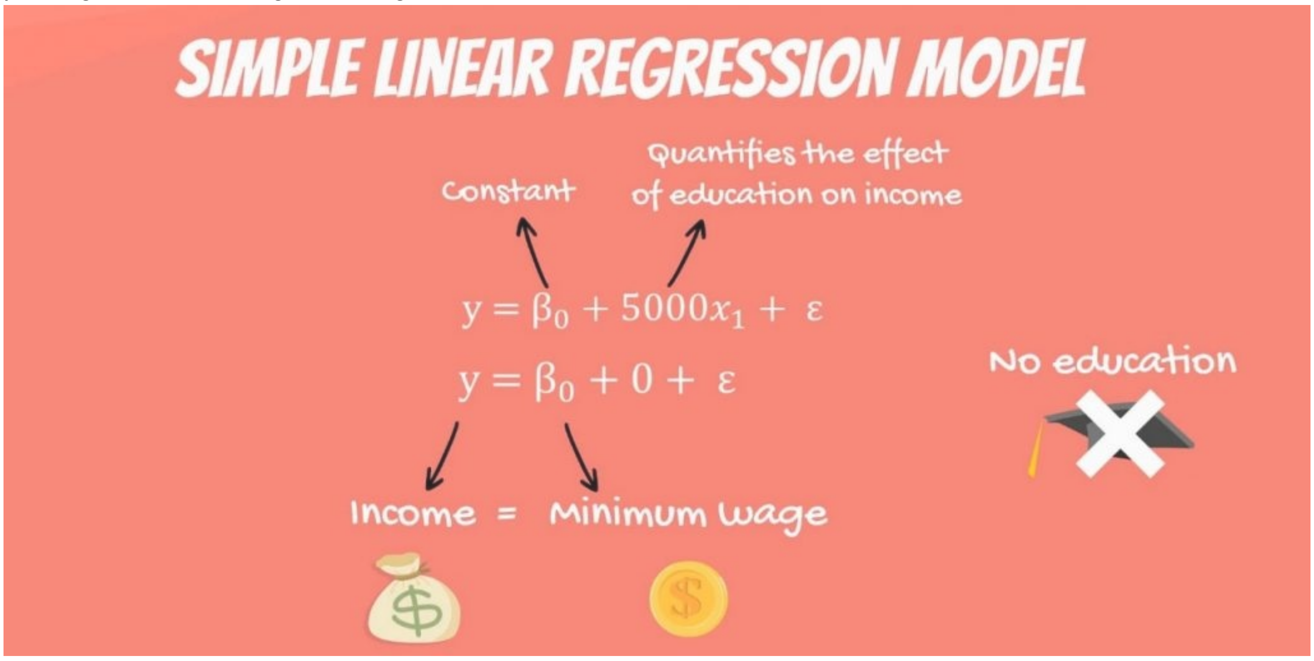

SIMPLE LINEAR REGRESSION MODEL
$y = \beta_0 + \beta_1 x_1 + \varepsilon$
Income
Education

So, what is B0, B1 and E:

- β1 is the coefficient that stands before the independent variable. It quantifies the effect of education on income.

  example: If β1 is 50, then for each additional year of education, your income would grow by $50.

In this linear regression example, you can think of the constant β0 as the minimum wage. No matter your education, if you have a job, you will get the minimum wage. This is a guaranteed amount.


SIMPLE LINEAR REGRESSION MODEL
Constant
Quantifies the effect of education on income
$y = \beta_0 + 5000x_1 + \varepsilon$
$y = \beta_0 + 0 + \varepsilon$
No education
Income = Minimum wage

Epsilon/Error

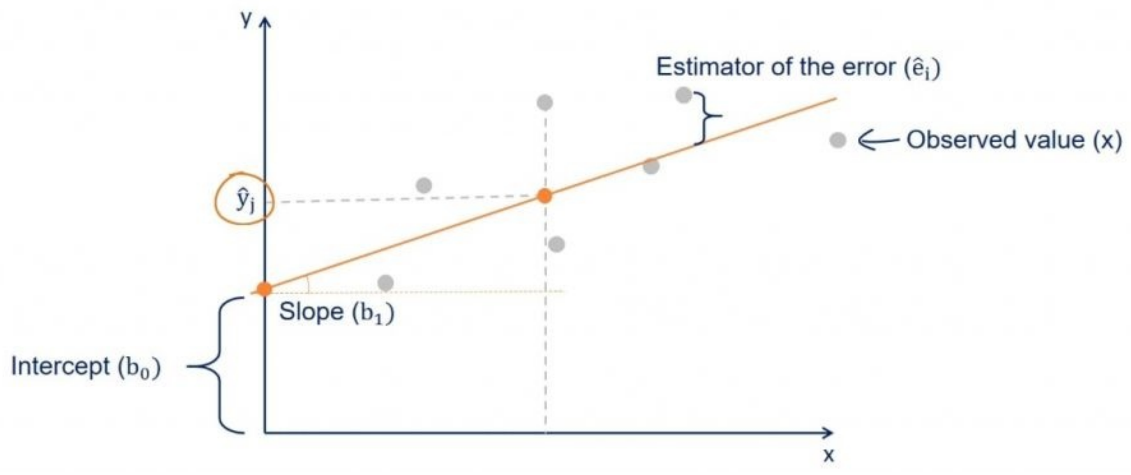
The last term is the epsilon(ε). This represents the error of estimation. The error is the actual difference between the observed income and the income the regression predicted.

Formula is same

What we have seen above or y=mx+b or below one:

Linear regression model. Geometrical representation

$$\hat{y}_i = b_0 + b_1 x_i$$



Lets Dive into Regression using Python

In [19]:
```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import statsmodels.formula.api as smf
```

Loading data using pandas

In [2]:
```
df=pd.read_csv(r"C:\Users\USER\Downloads\1.01. Simple linear regression.csv")
df.head()
```

Out[2]:

	SAT	GPA
0	1714	2.40
1	1664	2.52
2	1760	2.54
3	1685	2.74
4	1693	2.83

In [3]:
```
df.describe()
```

Out[3]:

	SAT	GPA
count	84.000000	84.000000
mean	1845.273810	3.330238
std	104.530661	0.271617
min	1634.000000	2.400000
25%	1772.000000	3.190000
50%	1846.000000	3.380000
75%	1934.000000	3.502500
max	2050.000000	3.810000

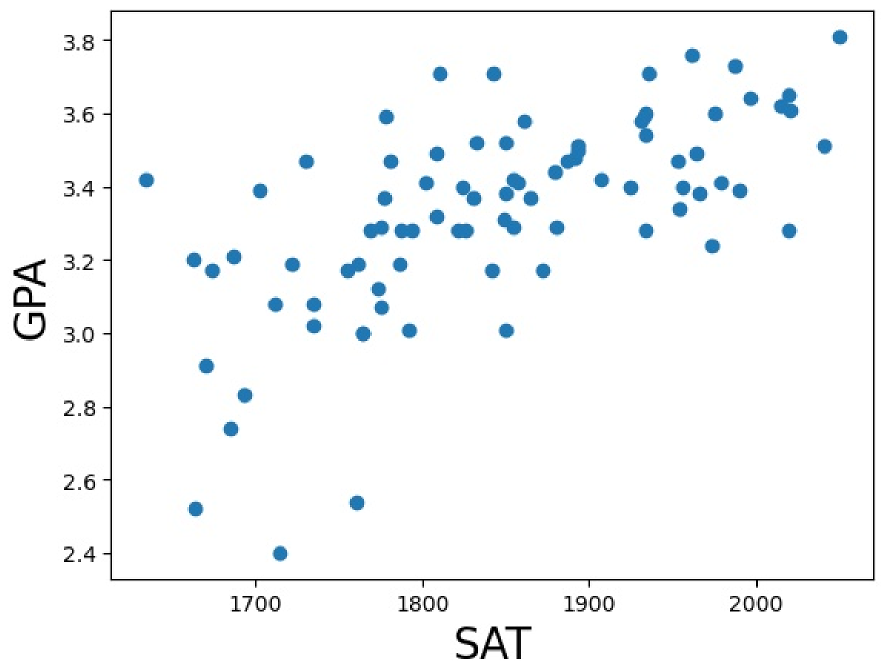
So, here we are predicting GPA after Grduation based on SAT score during student admission so we can approximate making prediction of student GPA when he/she get admission on college or school or University.

Before starting regression lets keep SAT and GPA in x and y, GPA is depend on SAT score so lets keep gpa as dependent variable in x and SAT as y.

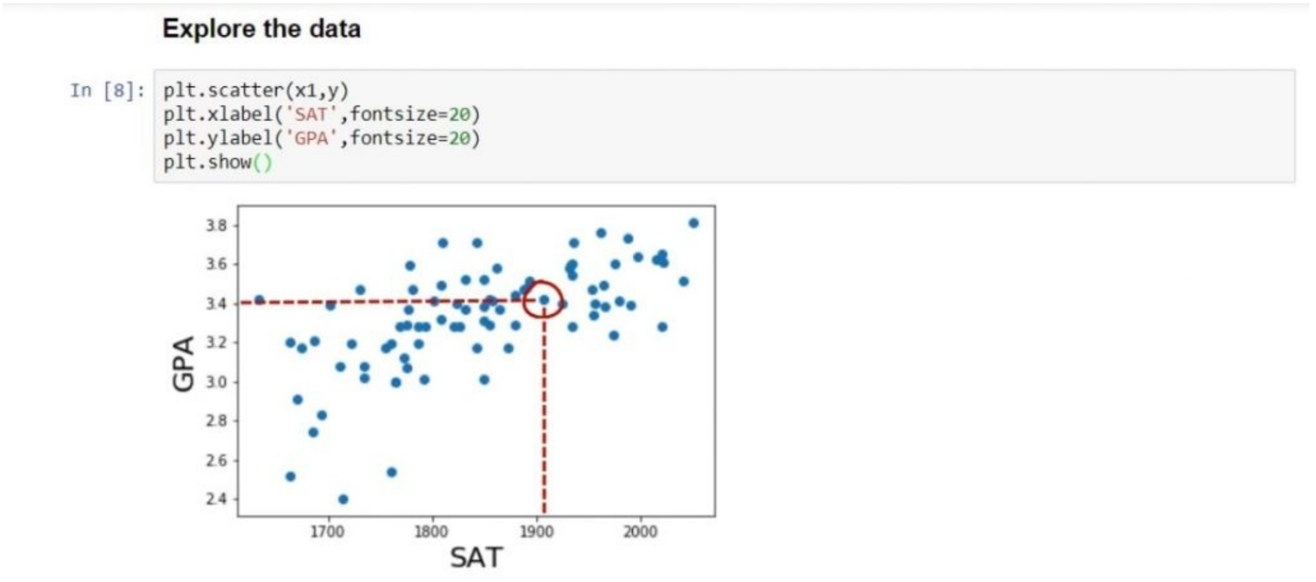
In [24]:
```
GPA=df.iloc[:,-1]
SAT=df.iloc[:,0:1]
```

Lets explore data with some graphs

In [25]:
```
plt.scatter(SAT,GPA)   # most of the time in scatter, we put y first but labeling should be as dependent and in

plt.xlabel('SAT', fontsize = 20)

plt.ylabel('GPA', fontsize = 20)

plt.show()
```



Each point on the graph represents a different student. For instance, the highlighted point below is a student who scored around 1900 on the SAT and graduated with a 3.4 GPA. (BELOW IS JUST A EXAMPLE -not related with calculation)

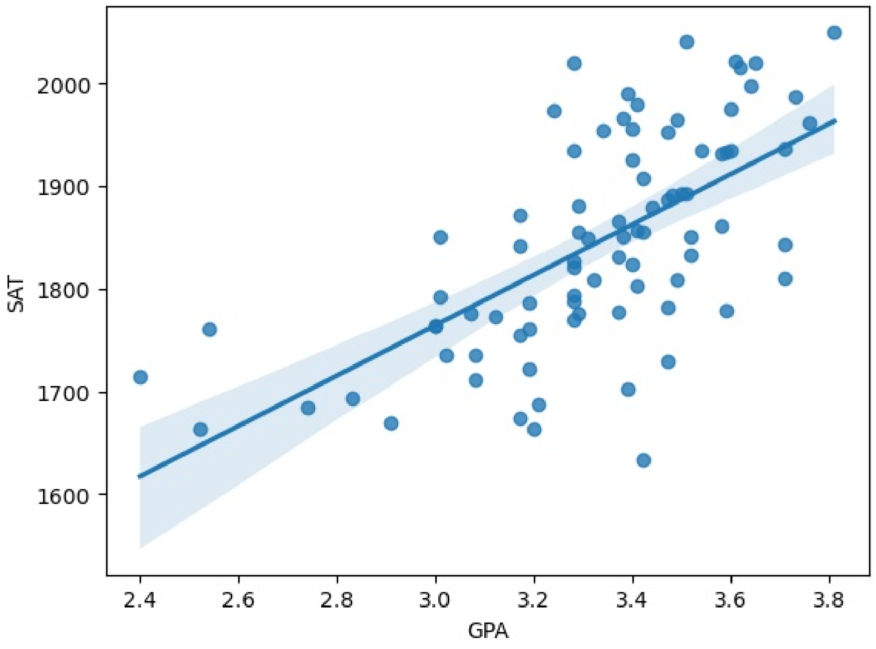


Explore the data

In [8]:
```
plt.scatter(x1,y)
plt.xlabel('SAT',fontsize=20)
plt.ylabel('GPA',fontsize=20)
plt.show()
```

In [26]:
```
import seaborn as sns
sns.regplot(x=GPA,y=SAT)
```

Out[26]:
```
<Axes: xlabel='GPA', ylabel='SAT'>
```



In [27]:
```
results = smf.ols('GPA ~ SAT', data=df).fit()
```

In [28]:
```
print(results.summary())
```

```
                            OLS Regression Results                            
==============================================================================
Dep. Variable:                    GPA   R-squared:                       0.406
Model:                            OLS   Adj. R-squared:                  0.399
Method:                 Least Squares   F-statistic:                     56.05
Date:                Sun, 03 Dec 2023   Prob (F-statistic):           7.20e-11
Time:                        13:58:43   Log-Likelihood:                 12.672
No. Observations:                  84   AIC:                            -21.34
Df Residuals:                      82   BIC:                            -16.48
Df Model:                           1                                         
Covariance Type:            nonrobust                                         
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.2750      0.409      0.673      0.503      -0.538       1.088
SAT            0.0017      0.000      7.487      0.000       0.001       0.002
==============================================================================
Omnibus:                       12.839   Durbin-Watson:                   0.950
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               16.155
Skew:                          -0.722   Prob(JB):                     0.000310
Kurtosis:                       4.590   Cond. No.                     3.29e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.29e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Therefore,

ŷ= 0.275 + 0.0017 * x

means:

y=mx+b or y=mx+c

In [29]:
```
GPA=0.275+0.0017*1750          #suppose if one student enter with score 1750
```

```
GPA=0.275+0.0017*1750          #suppose if one student enter with score 1750
```

In [30]:
```
print(GPA)
```

```
3.2499999999999996
```

The coefficient in a linear regression model represents the slope of the regression line. Specifically, it tells you how much the dependent variable is expected to increase (if the coefficient is positive) or decrease (if it's negative) when the independent variable increases by one unit.

Standard error from above table

The standard errors show the accuracy of prediction for each variable.

The lower the standard error, the better the estimate! means if data gets increase or sample value less standard error will be there.

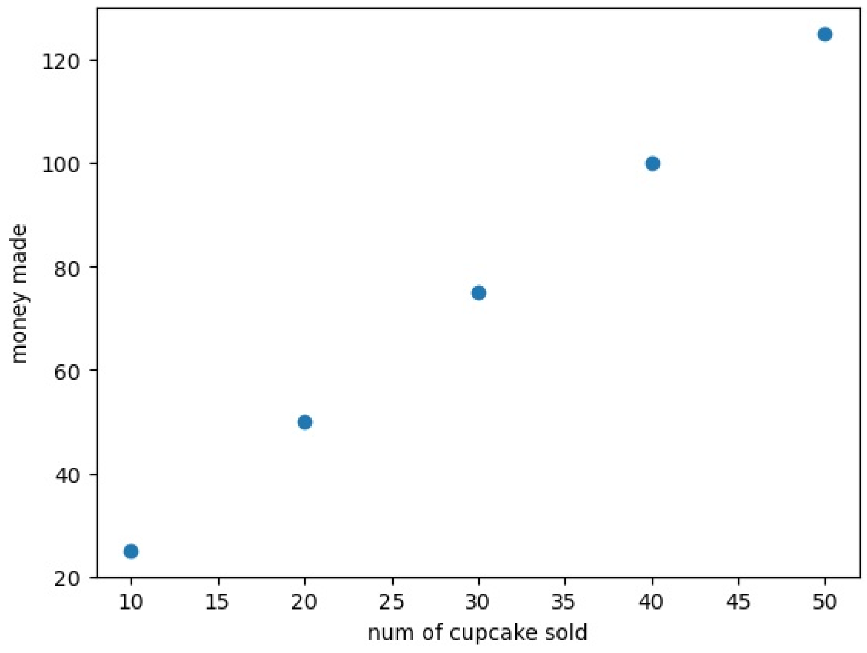
Trying Machine Learning Regression model with example

In [37]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
import pandas as pd

#sample data
num_cupcakes = [10, 20, 30, 40, 50]
money_made = [25, 50, 75, 100, 125]
```

In [38]:
```
x=pd.DataFrame(num_cupcakes)
y=pd.DataFrame(money_made)
```

In [40]:
```
plot1=plt.scatter(x,y)
plt.xlabel("num of cupcake sold")
plt.ylabel("money made")
```

Out[40]:
```
Text(0, 0.5, 'money made')
```



In [41]:
```
model=LinearRegression()
```

In [42]:
```
model.fit(x,y)
```

Out[42]:
```
▾ LinearRegression
LinearRegression()
```

In [43]:
```
new_money_made = model.predict([[25]])
```

In [44]:
```
print(new_money_made)
```

```
[[62.5]]
```

If we fit a linear regression model on a dataset without explicitly splitting it into training and test sets, like just above basic example, by default the model will use 100% of the data for training.

In that case, there is no separate test set that is held out for evaluating model performance. The model will overfit and perform very well on the training data used to fit it, but may not generalize well to new unseen data.

Typically in machine learning you want to:

- Split the original dataset into training and test sets (e.g. 80% training, 20% test)
- Fit the model on the training data only
- Evaluate model performance on the test data, which the model has not seen before