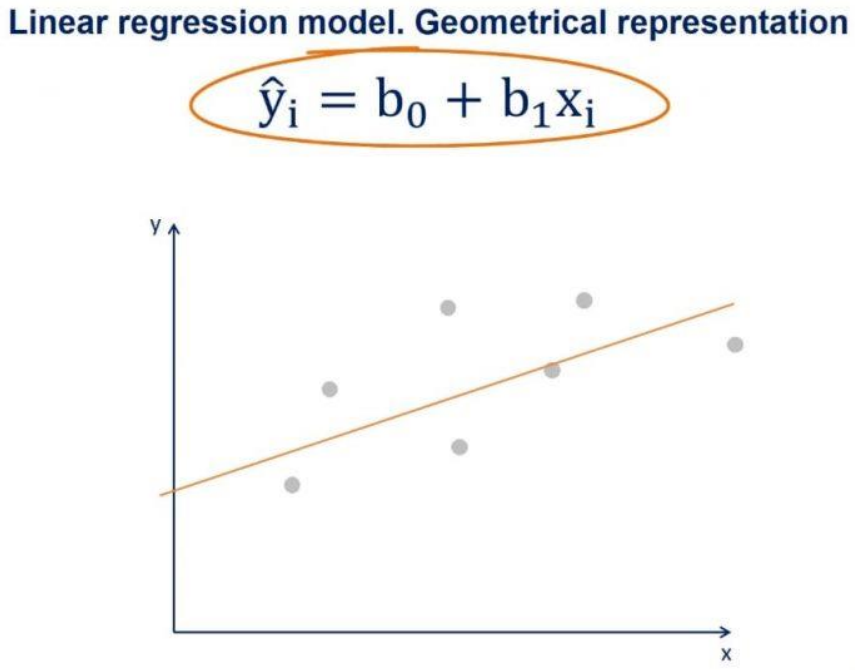
Data science and Machine learning with Python

Designed by Abdur Rahman Joy - MCSD, MCPD, MCSE, MCTS, OCJP, Sr. Technical Trainer for VFX at IDB BISW (Scholarship program), and C#.net, R, Scala, Kotlin, JAVA, Android/IOS/Windows Mobile Apps, SQL server, Azure, Oracle, SharePoint Development, AWS , CEH, KALI Linux, Python, Data Science, Machine Learning ,Software Testing, Graphics, Multimedia and Game Developer at Joy Infosys and other premises like BITM, SkillsJob, PNTL, Leads Training and New Horizon inc , Cell #: +880-1712587348, email: jspaonline@gmail.com. Web URL: http://www.joyinfosys.com/me.

Linear regression (continued)

The Regression Line

You may have heard about the **regression line**, too. When we plot the data points on an *x-y* plane, the **regression line** is the best-fitting line through the data points.

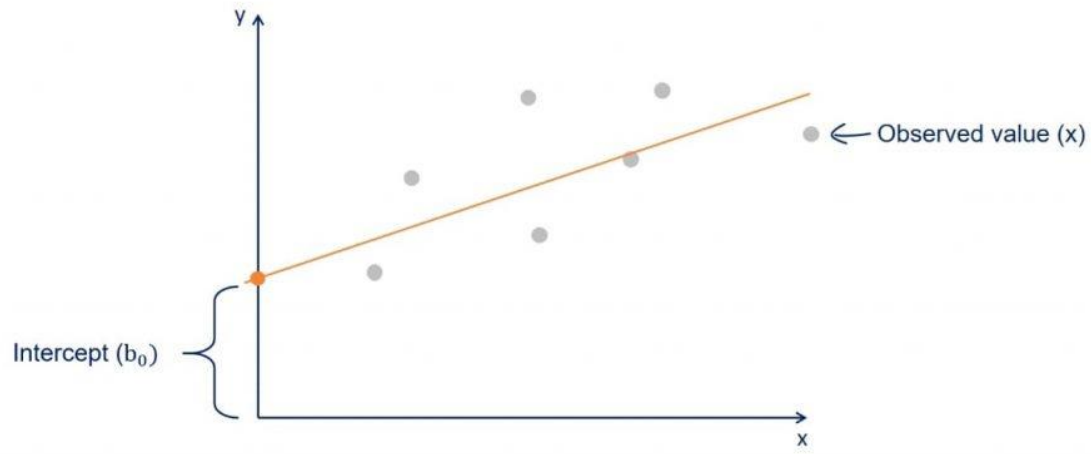


You can take a look at a plot with some data points in the picture above. We plot the line based on the **regression equation**.

The grey points that are scattered are the observed values. B_0, as we said earlier, is a *constant* and is the intercept of the **regression line** with the y-axis.

Linear regression model. Geometrical representation

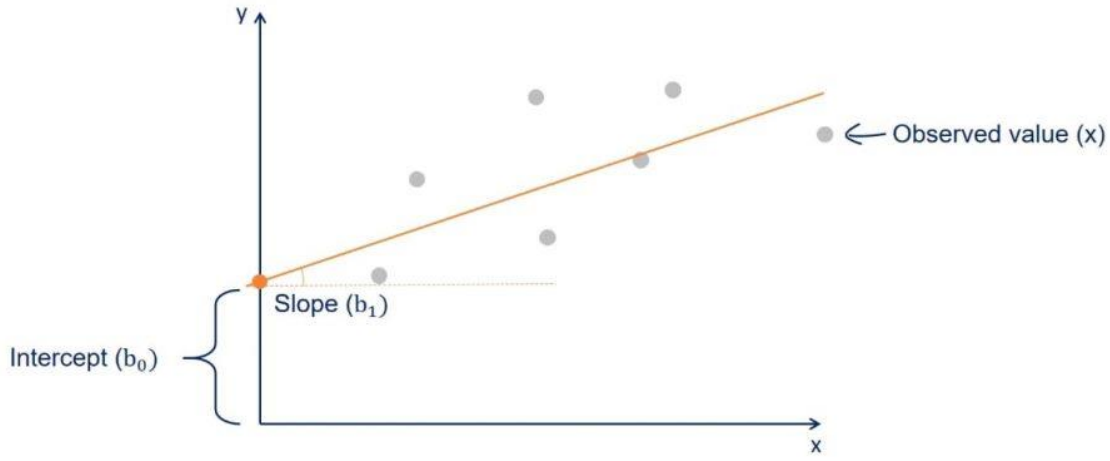
$$\hat{y}_i = b_0 + b_1 x_i$$



B_1 is the slope of the **regression line**. It shows how much *y* changes for each unit change of *x*.

Linear regression model. Geometrical representation

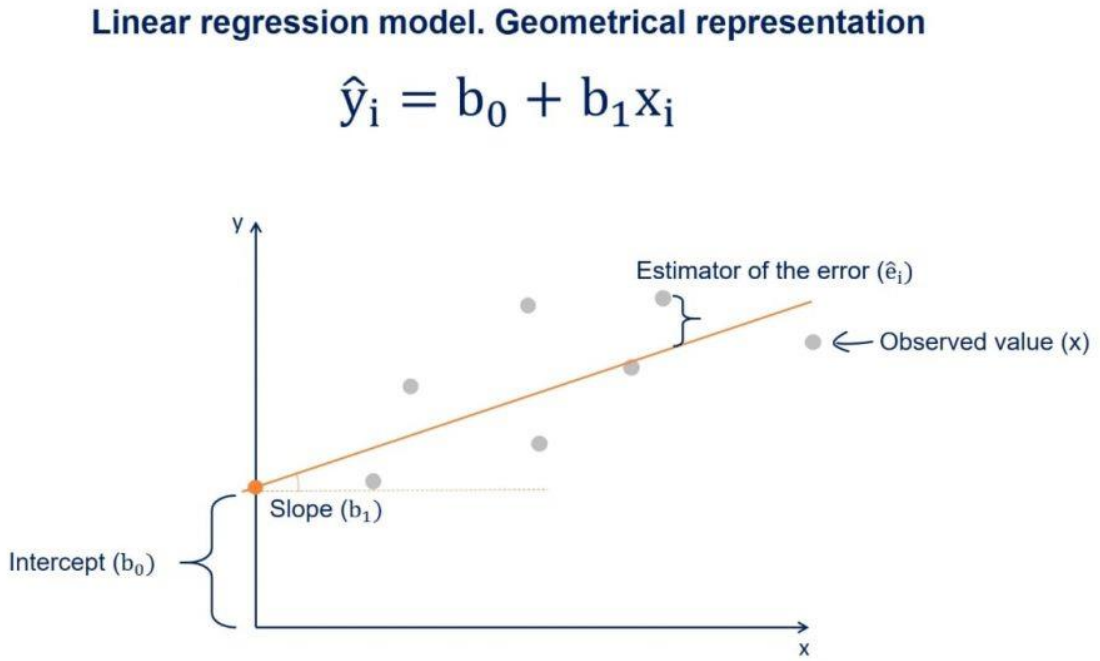
$$\hat{y}_i = b_0 + b_1 x_i$$



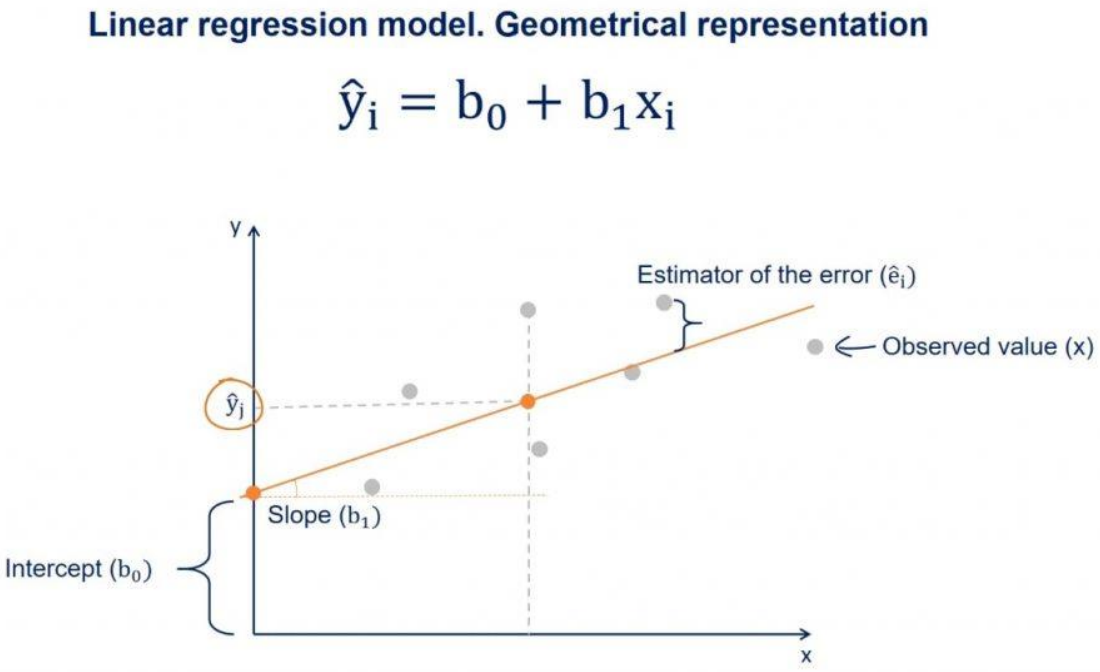
Designed by Abdur Rahman Joy - MCSD, MCPD, MCSE, MCTS, OCJP, Sr. Technical Trainer for VFX at IDB BISW (Scholarship program), and C#.net, R, Scala, Kotlin, JAVA, Android/IOS/Windows Mobile Apps, SQL server, Azure, Oracle, SharePoint Development, AWS , CEH, KALI Linux, Python, Data Science, Machine Learning ,Software Testing, Graphics, Multimedia and Game Developer at Joy Infosys and other premises like BITM, SkillsJob, PNTL, Leads Training and New Horizon inc , Cell #: +880-1712587348, email: jspaonline@gmail.com. Web URL: http://www.joyinfosys.com/me.

The Estimator of the Error

The distance between the observed values and the **regression line** is the *estimator of the error term epsilon*. Its **point estimate** is called residual.

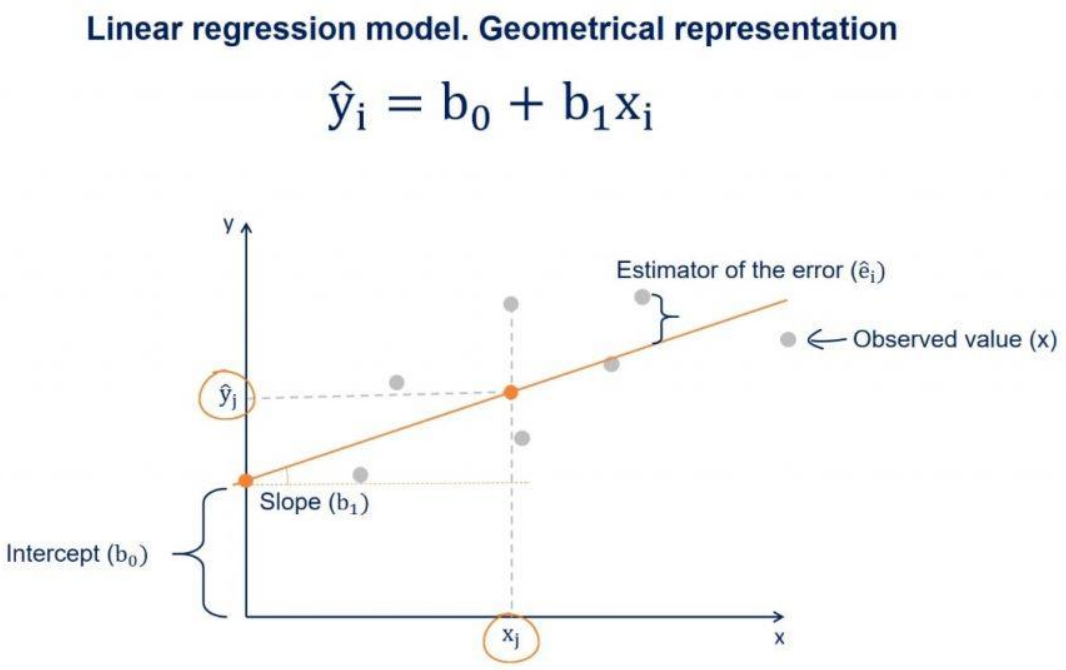


Now, suppose we draw a perpendicular from an observed point to the **regression line**. The intercept between that perpendicular and the **regression line** will be a point with a *y* value equal to $\hat{y}$.



Designed by Abdur Rahman Joy - MCSD, MCPD, MCSE, MCTS, OCJP, Sr. Technical Trainer for VFX at IDB BISW (Scholarship program), and C#.net, R, Scala, Kotlin, JAVA, Android/IOS/Windows Mobile Apps, SQL server, Azure, Oracle, SharePoint Development, AWS , CEH, KALI Linux, Python, Data Science, Machine Learning ,Software Testing, Graphics, Multimedia and Game Developer at Joy Infosys and other premises like BITM, SkillsJob, PNTL, Leads Training and New Horizon inc , Cell #: +880-1712587348, email: jspaonline@gmail.com. Web URL: http://www.joyinfosys.com/me.

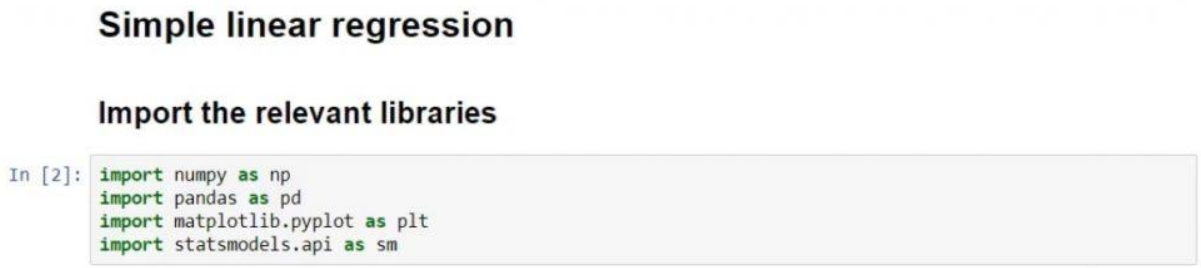
As we said earlier, given an x, $\hat{y}$ is the value predicted by the **regression line**.



Linear Regression Example in Python

Importing the Relevant Libraries

Let's import the following libraries:



The first three are pretty conventional. We won't even need numpy, but it's always good to have it there – ready to lend a helping hand for some operations. In addition, the machine learning library we will employ for this **linear regression** example is: statsmodels. So, we can basically write the following code: import numpy as np

```
import pandas as pd

import matplotlib.pyplot as plt

import statsmodels.api as sm
```

Loading the Data

The data which we will be using for our **linear regression** example is in a .csv file called: 'regression.csv'. Make sure that you save it in the folder of the user.

Now, let's load it in a new variable called: *data* using the *pandas* method: 'read_csv'. We can write the following code:

```
data = pd.read_csv('regression.csv')
```

After running it, the data from the .csv file will be loaded in the *data* variable. As we are using *pandas*, the *data* variable will be automatically converted into a data frame.

Visualizing the Data Frame

Let's see if that's true. We can write *data* and run the line. As you can see below, we have indeed displayed the data frame.

```
In [4]: data
```

Out[4]:

	SAT	GPA
0	1714	2.40
1	1664	2.52
2	1760	2.54
3	1685	2.74
4	1693	2.83
5	1670	2.91
6	1764	3.00
7	1764	3.00
8	1792	3.01
9	1850	3.01
10	1735	3.02
11	1775	3.07

There are two columns – *SAT* and *GPA*. And that's what our **linear regression** example will be all about. Let's further check

```
data.describe()
```

This is a *pandas* method which will give us the most useful descriptive statistics for each column in the data frame – number of observations, **mean**, **standard deviation**, and so on.

```
In [5]: data.describe()
```

Out[5]:

	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

In this **linear regression** example we won't put that to work just yet. However, it's good practice to use it.

The Problem

Let's explore the problem with our **linear regression** example.

So, we have a sample of 84 students, who have studied in college.

```
In [5]: data.describe()
Out[5]:
```

	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

Their total SAT scores include critical reading, mathematics, and writing. Whereas, the GPA is their Grade Point Average they had at graduation.


```
In [5]: data.describe()
Out[5]:
```

	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

SAT = Critical Reading + Mathematics + Writing

GPA = Grade point average (at graduation from university)

That's a very famous relationship. We will create a **linear regression** which predicts the GPA of a student based on their SAT score.

When you think about it, it totally makes sense.

1. You sit the SAT and get a score.
2. With this score, you apply to college.
3. The next 4 years, you attend college and graduate receiving many grades, forming your GPA.

You can see the timeline below.

```
In [5]: data.describe()
Out[5]:
```

	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

SAT = Critical Reading + Mathematics + Writing

GPA = Grade point average (at graduation from university)

We will create a linear regression which predicts GPA based on the SAT score obtained

Meaningful Regressions

Before we finish this introduction, we want to get this out of the way. Each time we create a **regression**, it should be meaningful. **Why** would we predict GPA with SAT? Well, the SAT is considered one of the best estimators of intellectual capacity and capability.

On average, if you did well on your SAT, you will do well in college and at the workplace. Furthermore, almost all colleges across the USA are using the SAT as a proxy for admission.

And last but not least, the SAT stood the test of time and established itself as the leading exam for college admission.

```
In [5]: data.describe()
Out[5]:
```

	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

Why would I predict GPA with SAT?

1. The SAT is considered one of the best estimators of intellectual capacity and capability
2. Almost all college across the USA are using the SAT as a proxy for admission
3. The SAT stood the test of time

It is safe to say our **regression** makes sense.

Creating our First Regression in Python

After we've cleared things up, we can start creating our first **regression** in Python. We will go through the code and in subsequent tutorials, we will clarify each point.

Important: Remember, the equation is:

Our *dependent variable* is GPA, so let's create a variable called *y* which will contain GPA.

Just a reminder – the *pandas'* syntax is quite simple.

This is all we need to code:

y = data['GPA']

x1 = data['SAT']

It goes like this:

1. First, we write the name of the data frame, in this case *data*
2. Then, we add in square brackets the relevant column name, which is GPA in our case.

Similarly, our *independent variable* is SAT, and we can load it in a variable x1.

Create your first regression

Define the dependent and the independent variables

```
In [7]: y = data['GPA']
        x1 = data['SAT']
```

Exploring the Data

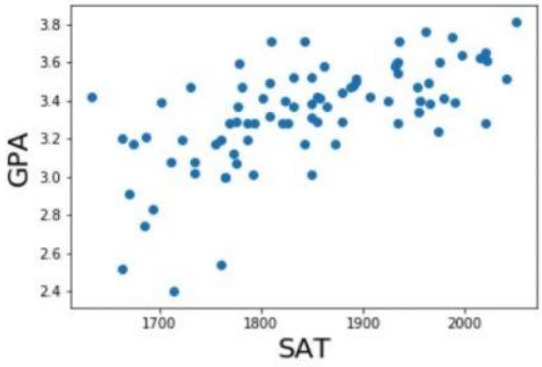
It's always useful to plot our data in order to understand it better and see if there is a relationship to be found. We will use some conventional *matplotlib* code.

```
plt.scatter(x1,y)

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

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

plt.show()
```

You can see the result we receive after running it, in the picture below.

Explore the data

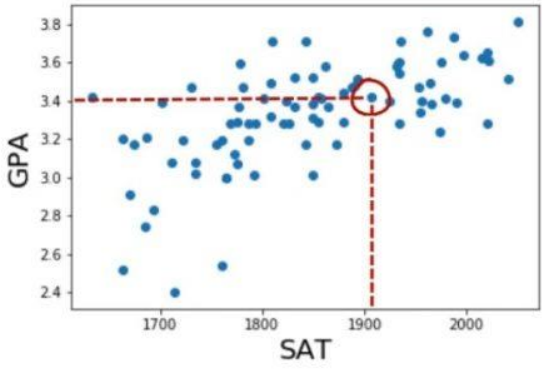
```
In [8]: plt.scatter(x1,y)
        plt.xlabel('SAT',fontsize=20)
        plt.ylabel('GPA',fontsize=20)
        plt.show()
```



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.

Explore the data

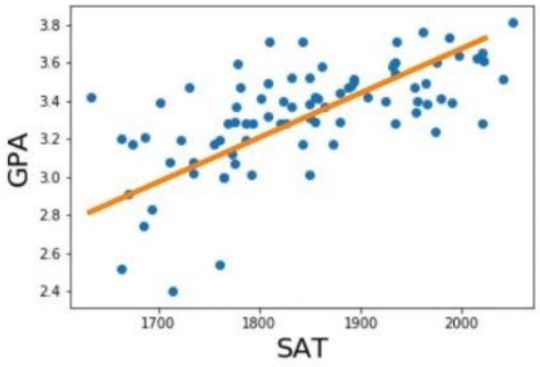
```
In [8]: plt.scatter(x1,y)
        plt.xlabel('SAT',fontsize=20)
        plt.ylabel('GPA',fontsize=20)
        plt.show()
```



Observing all data points, we can see that there is a strong relationship between SAT and GPA. In general, the higher the SAT of a student, the higher their GPA.

Explore the data

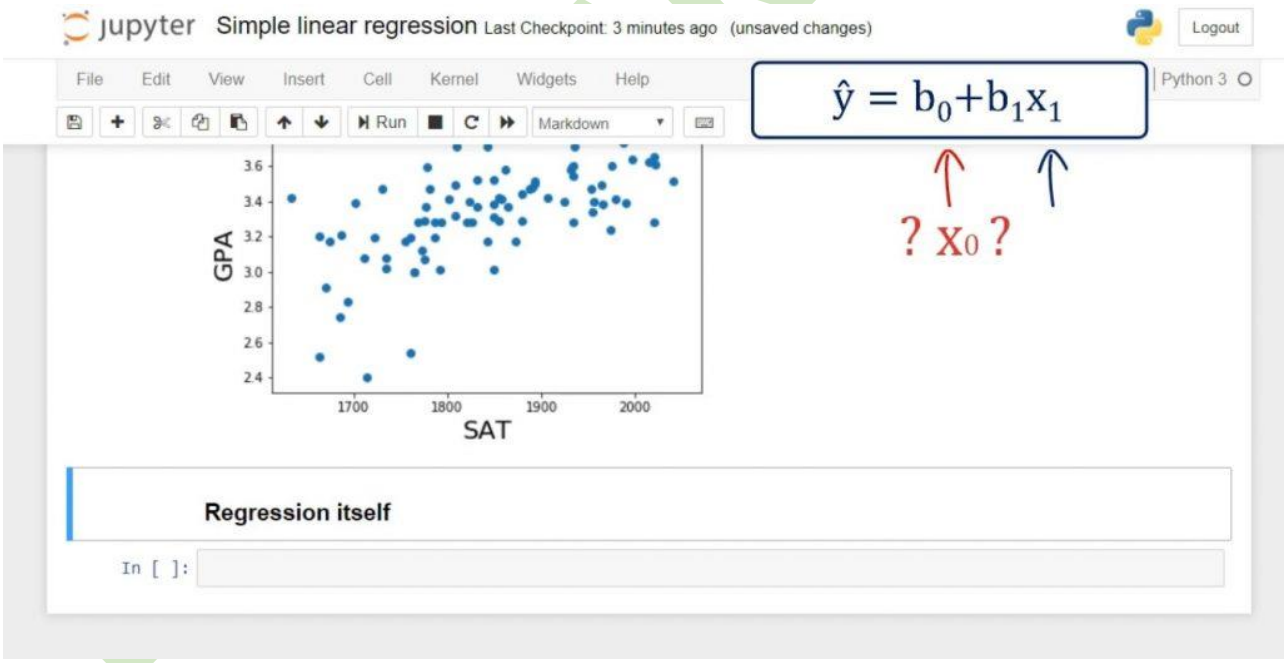
```
In [8]: plt.scatter(x1,y)
        plt.xlabel('SAT',fontsize=20)
        plt.ylabel('GPA',fontsize=20)
        plt.show()
```



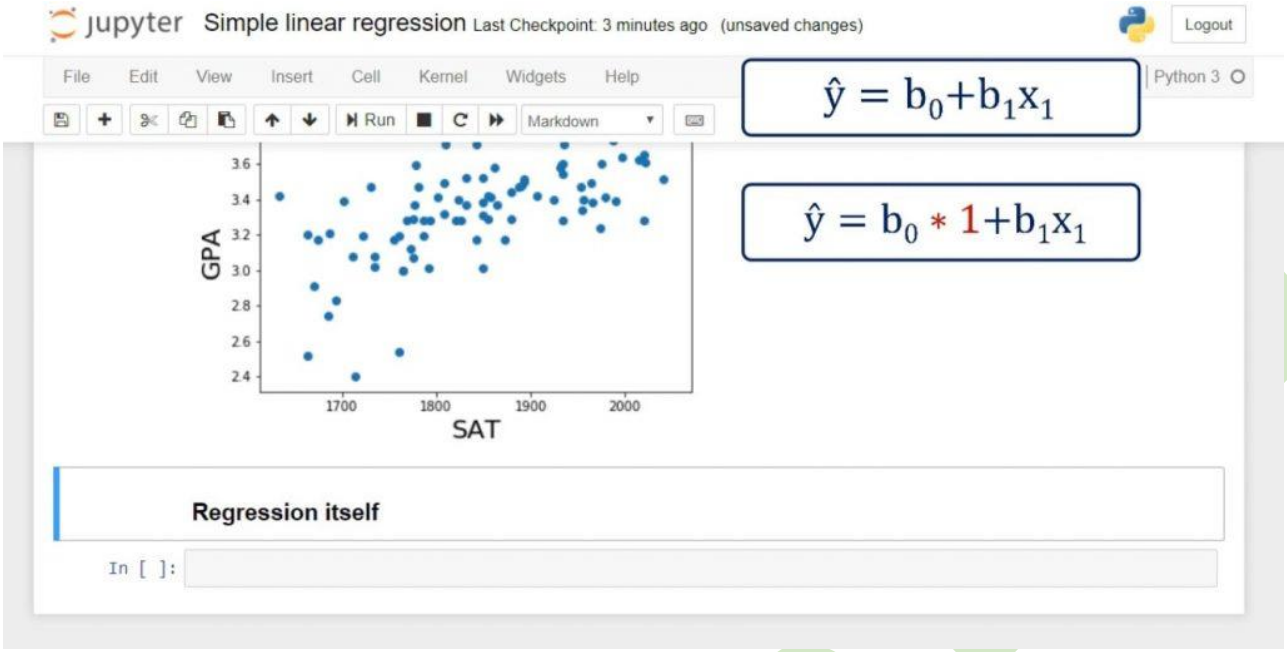
Adding a Constant

Next, we need to create a new variable, which we'll call x.

We have our x_1, but we don't have an x_0. In fact, in the **regression equation** there is no explicit x_0. The coefficient b_0 is alone.



That can be represented as: b_0 * 1. So, if there was an x_0, it would always be 1.



It is really practical for computational purposes to incorporate this notion into the equation. And that's how we estimate the intercept b_0.

In terms of code, *statsmodels* uses the method: .add_constant(). So, let's declare a new variable:

```
x = sm.add_constant(x1)
```

The Results Variable

Right after we do that, we will create another variable named *results*. It will contain the output of the *ordinary least squares* ***regression***, or **OLS**. As arguments, we must add the *dependent variable y* and the newly defined *x*. At the end, we will need the .fit() method. It is a method that applies a specific estimation technique to obtain the fit of the model.

That itself is enough to perform the **regression**.

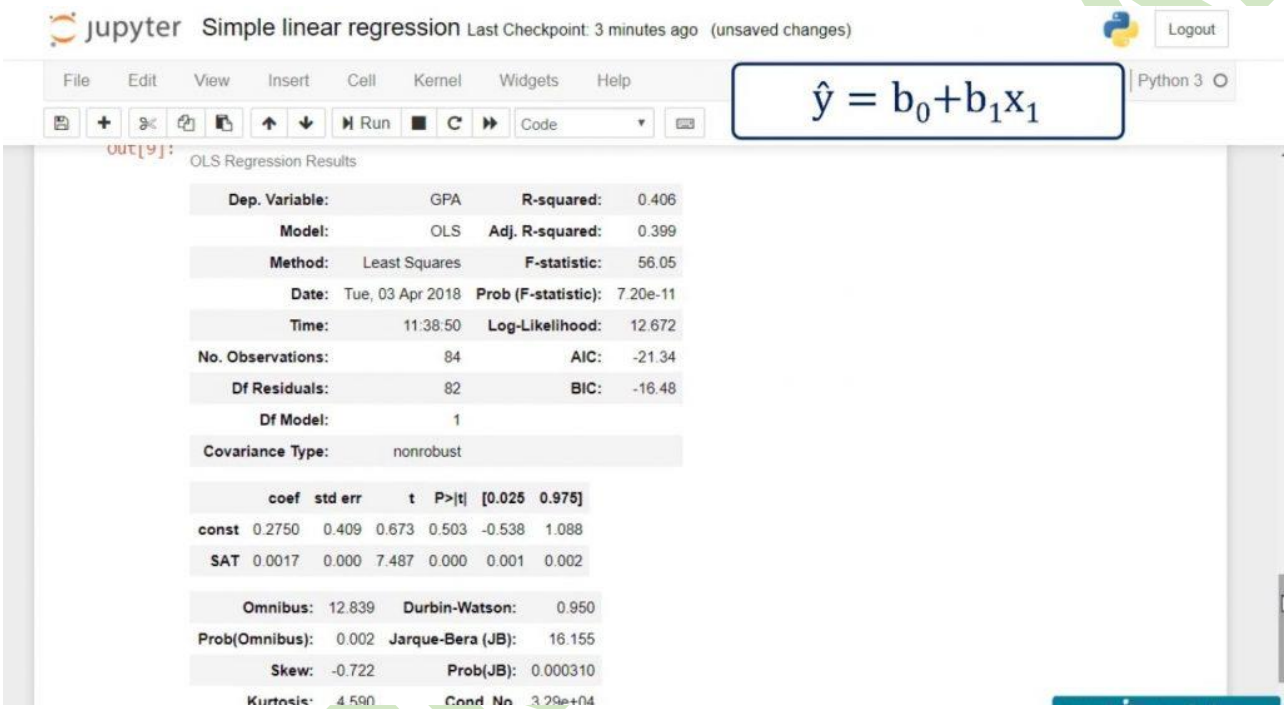
Displaying the Regression Results

In any case, results.summary() will display the **regression** results and organize them into three tables.

So, this is all the code we need to run:

```
x = sm.add_constant(x1)

results = sm.OLS(y,x).fit()

results.summary()
```

And this is what we get after running it:



As you can see, we have a lot of statistics in front of us! And we will examine it in more detail in subsequent tutorials.

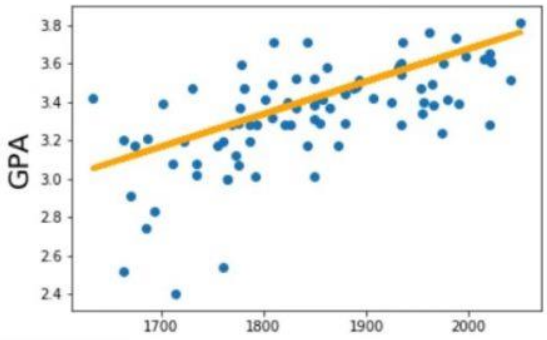
Plotting the Regression line

Let's plot the regression line on the same scatter plot. We can achieve that by writing the following:

```
plt.scatter(x1,y)

yhat = 0.0017*x1 + 0.275

fig = plt.plot(x1,yhat, lw=4, c='orange', label = 'regression line')

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

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

plt.show()
```

As you can see below, that is the best fitting line, or in other words – the line which is closest to all observations simultaneously.

```
In [9]: plt.scatter(x1,y)
        yhat = 0.0017*x1 + 0.275
        fig = plt.plot(x1,yhat, lw=4, c='orange', label ='regression line')
        plt.xlabel('SAT', fontsize = 20)
        plt.ylabel('GPA', fontsize = 20)
        plt.show()
```



So that's how you create a **simple linear regression** in Python!

Designed by Abdur Rahman Joy - MCSD, MCPD, MCSE, MCTS, OCJP, Sr. Technical Trainer for VFX at IDB BISW (Scholarship program), and C#.net, R, Scala, Kotlin, JAVA, Android/IOS/Windows Mobile Apps, SQL server, Azure, Oracle, SharePoint Development, AWS , CEH, KALI Linux, Python, Data Science, Machine Learning ,Software Testing, Graphics, Multimedia and Game Developer at Joy Infosys and other premises like BITM, SkillsJob, PNTL, Leads Training and New Horizon inc , Cell #: +880-1712587348, email: jspaonline@gmail.com. Web URL: http://www.joyinfosys.com/me.