Data Science and Generative AI

by SATISH @

https://sathyatech.com

Machine Learning

Data Set

In the mind of a computer, a data set is any collection of data. It can be anything from an array to a complete database.

Example of an array:

[99,86,87,88,111,86,103]

Example of a database:

Carname	Color	Age	Speed	AutoPass
BMW	red	5	99	Y
Volvo	black	7	86	Y
Ford	white	2	111	Y
Tesla	red	2	103	Y

By looking at the array, we can guess that the average value is probably around 80 or 90, and we are also able to determine the highest value and the lowest value, but what else can we do?

And by looking at the database we can see that the most popular color is white, and the oldest car is 17 years, but what if we could predict if a car had an AutoPass, just by looking at the other values?

That is what Machine Learning is for! Analyzing data and predicting the outcome!

In Machine Learning it is common to work with very large data sets. In this we will try to make it as easy as possible to understand the different concepts of machine learning, and we will work with small easy-to-understand data sets.

Data Types

To analyze data, it is important to know what type of data we are dealing with.

We can split the data types into three main categories:

- **Numerical**
- **Categorical**
- **Ordinal**

Numerical data are numbers, and can be split into two numerical categories:

- **Discrete Data**
  - counted data that are limited to integers. **Example**: The number of cars passing by.
- **Continuous Data**
  - measured data that can be any number. **Example**: The price of an item, or the size of an item

Categorical data are values that cannot be measured up against each other.

Example: a color value, or any yes/no values.

Ordinal data are like categorical data, but can be measured up against each other.

Example: school grades where A is better than B and so on.

By knowing the data type of your data source, you will be able to know what technique to use when analyzing them.

You will learn more about statistics and analyzing data in the next chapters.

Machine Learning - Mean Median Mode

Mean, Median, and Mode

What can we learn from looking at a group of numbers?

In Machine Learning (and in mathematics) there are often three values that interests us:

- **Mean** - The average value
- **Median** - The mid point value
- **Mode** - The most common value

Example: We have registered the speed of 13 cars:

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

What is the average, the middle, or the most common speed value?

Mean

The mean value is the average value.

To calculate the mean, find the sum of all values, and divide the sum by the number of values:

(99+86+87+88+111+86+103+87+94+78+77+85+86) / 13 = 89.77

The NumPy module has a method for this. Learn about the NumPy module

Example

Use the NumPy mean() method to find the average speed:

```
import numpy

speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]

x = numpy.mean(speed)

print(x)
```

Machine Learning - Standard Deviation

What is Standard Deviation?

Standard deviation is a number that describes how spread out the values are.

A **low standard deviation** means that most of the numbers are **close to the mean (average)** value.

A **high standard deviation** means that the values are **spread out over a wider range**.

Example: This time we have registered the speed of 7 cars:

speed = [86,87,88,86,87,85,86]

mean = (86+87+88+86+87+85+86)/7 = 86.4

Mean differene Value: [0.4, -0.6, -1.6, 0.4, -0.6, 1.4, -0.4]

Squre differene Value: [0.16, 0.36, 2.56, 0.16, 0.36, 1.96, 0.16]

Variance: (0.16+0.36+2.56+0.16+0.36+1.96+0.16)/7 = 5.72 / 7 = 0.817142

Standard deviation: sqare root of variance (0.817142) = 0.9

The standard deviation is:

0.9

Meaning that most of the values are within the range of 0.9 from the mean value, which is 86.4.

Let us do the same with a selection of numbers with a wider range:

Ex-2:

speed = [32,111,138,28,59,77,97]

mean = (32+111+138+28+59+77+97)/7= 77.4

Mean differene Value: [45.4, -33.57, -60.5, 49.4, 18.4, 0.4, -19.5]

Squre differene Value: [2061.16, 1126.9, 3660.2, 2440.3 , 338.56, 0.16, 380.2]

Variance: (2061.16+1126.9+3660.2+2440.3+338.56+0.16+380.2)/7 = 10007.48 / 7 = 1429.64

Standard deviation: sqare root of variance (1429.64) = 37.8

The standard deviation is:

37.85

Meaning that most of the values are within the range of 37.85 from the mean value, which is 77.4.

As you can see, a higher standard deviation indicates that the values are spread out over a wider range.

The NumPy module has a method to calculate the standard deviation:

Example

Use the NumPy std() method to find the standard deviation:

```
import numpy

speed = [86,87,88,86,87,85,86]
x = numpy.std(speed)
print(x)
```

Example

```
import numpy

speed = [32,111,138,28,59,77,97]

x = numpy.std(speed)

print(x)
```

Machine Learning - Percentiles

What are Percentiles?

Percentiles are used in statistics to give you a number that describes the value that a given percent of the values are lower than.

Example: Let's say we have an array of the ages of all the people that live in a street.

Ex:

ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]

What is the 75. percentile? The answer is 43, meaning that 75% of the people are 43 or younger.

The NumPy module has a method for finding the specified percentile:

Example

Use the NumPy percentile() method to find the percentiles:

```
import numpy

ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]

x = numpy.percentile(ages, 75)

print(x)
```

Example

What is the age that 90% of the people are younger than?

```
import numpy

ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]

x = numpy.percentile(ages, 90)

print(x)
```

Machine Learning - Data Distribution

Data Distribution

Earlier in this we have worked with very small amounts of data in our examples, just to understand the different concepts.

In the real world, the data sets are much bigger, but it can be difficult to gather real world data, at least at an early stage of a project.

How Can we Get Big Data Sets?

To create big data sets for testing, we use the Python module NumPy, which comes with a number of methods to create random data sets, of any size.

Example

Create an array containing 250 random floats between 0 and 5:

```
import numpy

x = numpy.random.uniform(0.0, 5.0, 250)

print(x)
```

Histogram

To visualize the data set we can draw a histogram with the data we collected.

We will use the Python module Matplotlib to draw a histogram.

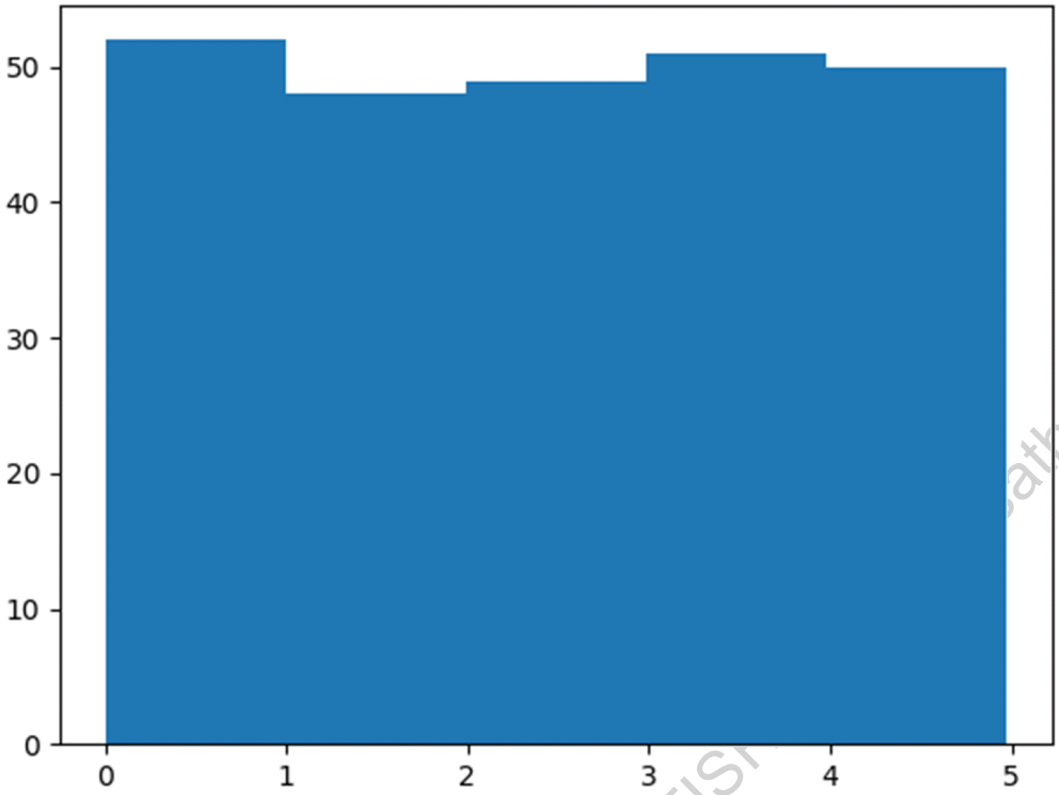
Example

Draw a histogram:

```
import numpy
import matplotlib.pyplot as plt
```

```
x = numpy.random.uniform(0.0, 5.0, 250)

plt.hist(x, 5)
plt.show()
```

Result:



Histogram Explained

We use the array from the example above to draw a histogram with 5 bars.

The first bar represents how many values in the array are between 0 and 1.

The second bar represents how many values are between 1 and 2.

Etc.

Which gives us this result:

- 52 values are between 0 and 1
- 48 values are between 1 and 2

- 49 values are between 2 and 3
- 51 values are between 3 and 4
- 50 values are between 4 and 5

Note: The array values are random numbers and will not show the exact same result on your computer.

Big Data Distributions

An array containing 250 values is not considered very big, but now you know how to create a random set of values, and by changing the parameters, you can create the data set as big as you want.

Example

Create an array with 100000 random numbers, and display them using a histogram with 100 bars:

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.uniform(0.0, 5.0, 100000)

plt.hist(x, 100)
plt.show()
```

Machine Learning - Normal Data Distribution

Normal Data Distribution

In probability theory this kind of data distribution is known as the ***normal data distribution***, or the ***Gaussian data distribution***, after the mathematician Carl Friedrich Gauss who came up with the formula of this data distribution.

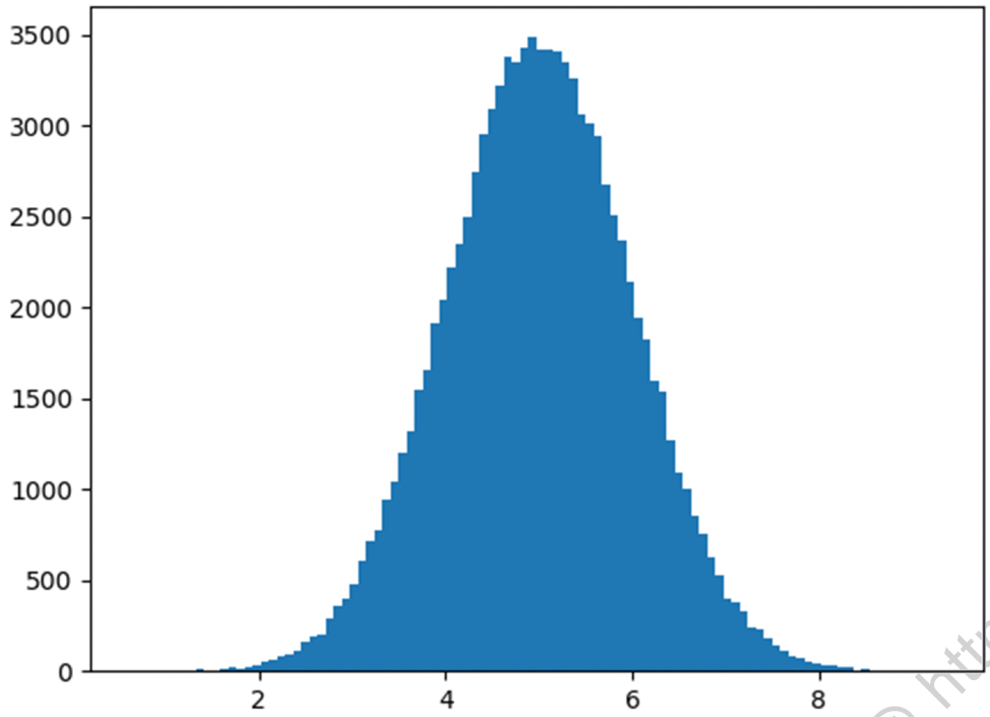
Example

A typical normal data distribution:

```
import numpy
import matplotlib.pyplot as plt
```

```
x = numpy.random.normal(5.0, 1.0, 100000)
plt.hist(x, 100)
plt.show()
```

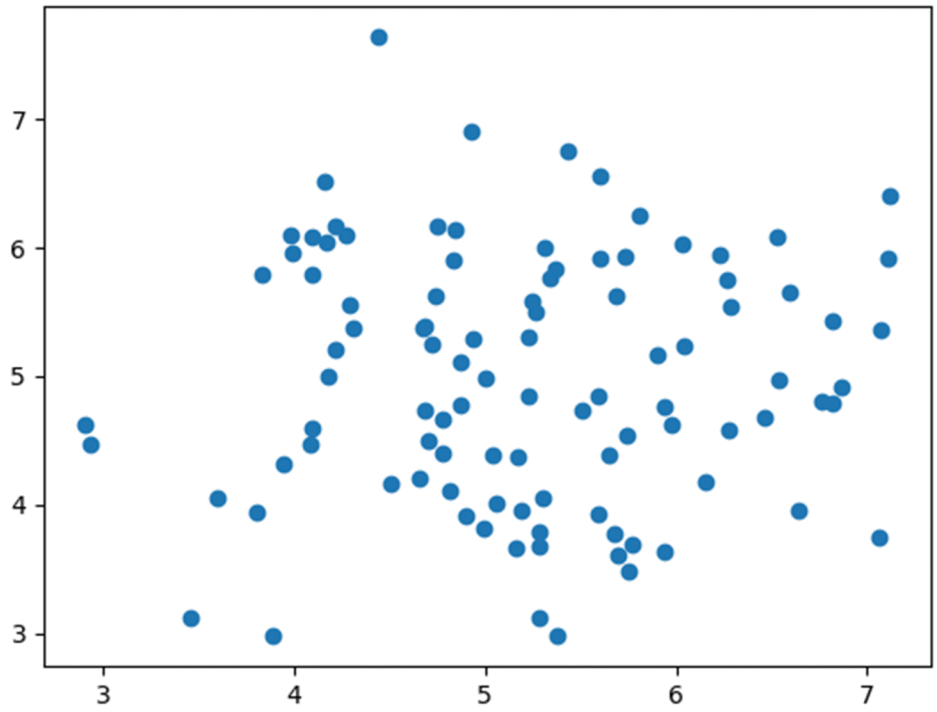
Result:



Machine Learning - Scatter Plot

Scatter Plot

A scatter plot is a diagram where each value in the data set is represented by a dot.



The Matplotlib module has a method for drawing scatter plots, it needs two arrays of the same length, one for the values of the x-axis, and one for the values of the y-axis:

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]

y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

The x array represents the age of each car.

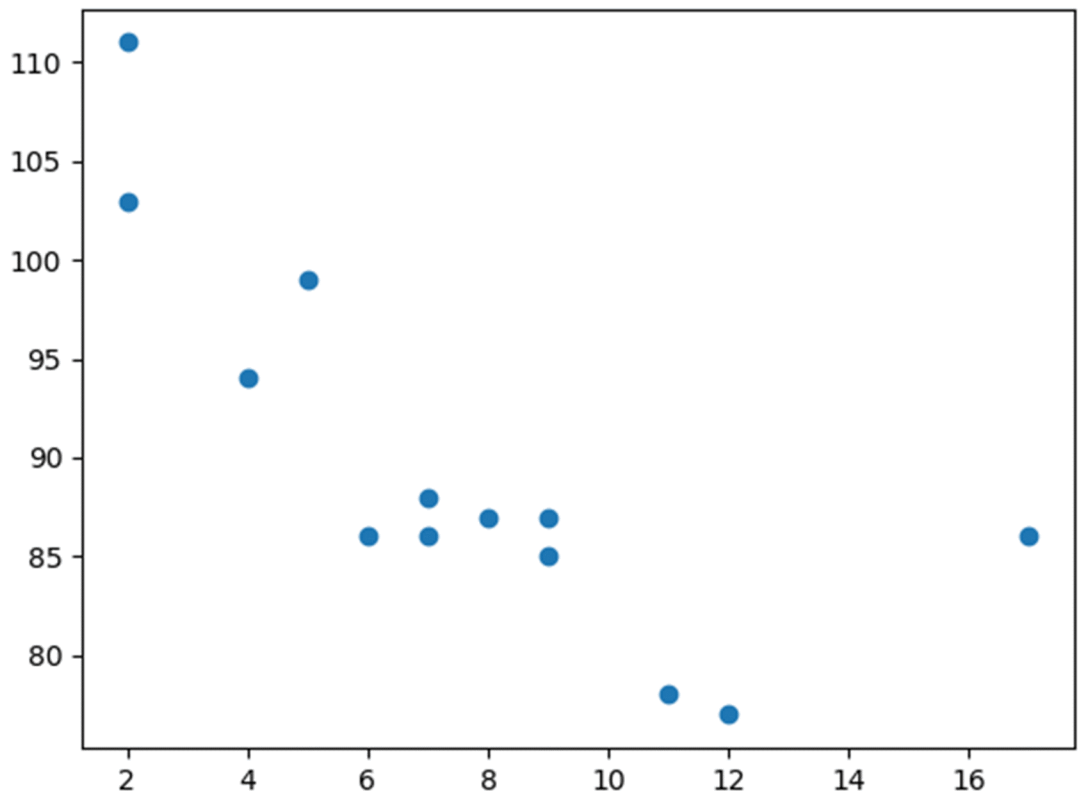
The y array represents the speed of each car.

Example

Use the scatter() method to draw a scatter plot diagram:

```
import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

plt.scatter(x, y)
plt.show()
```

Result:



Scatter Plot Explained

The x-axis represents ages, and the y-axis represents speeds.

What we can read from the diagram is that the two fastest cars were both 2 years old, and the slowest car was 12 years old.

Note: It seems that the newer the car, the faster it drives, but that could be a coincidence, after all we only registered 13 cars.

Random Data Distributions

In Machine Learning the data sets can contain thousands-, or even millions, of values.

You might not have real world data when you are testing an algorithm, you might have to use randomly generated values.

Let us create two arrays that are both filled with 1000 random numbers from a normal data distribution.

The first array will have the mean set to 5.0 with a standard deviation of 1.0.

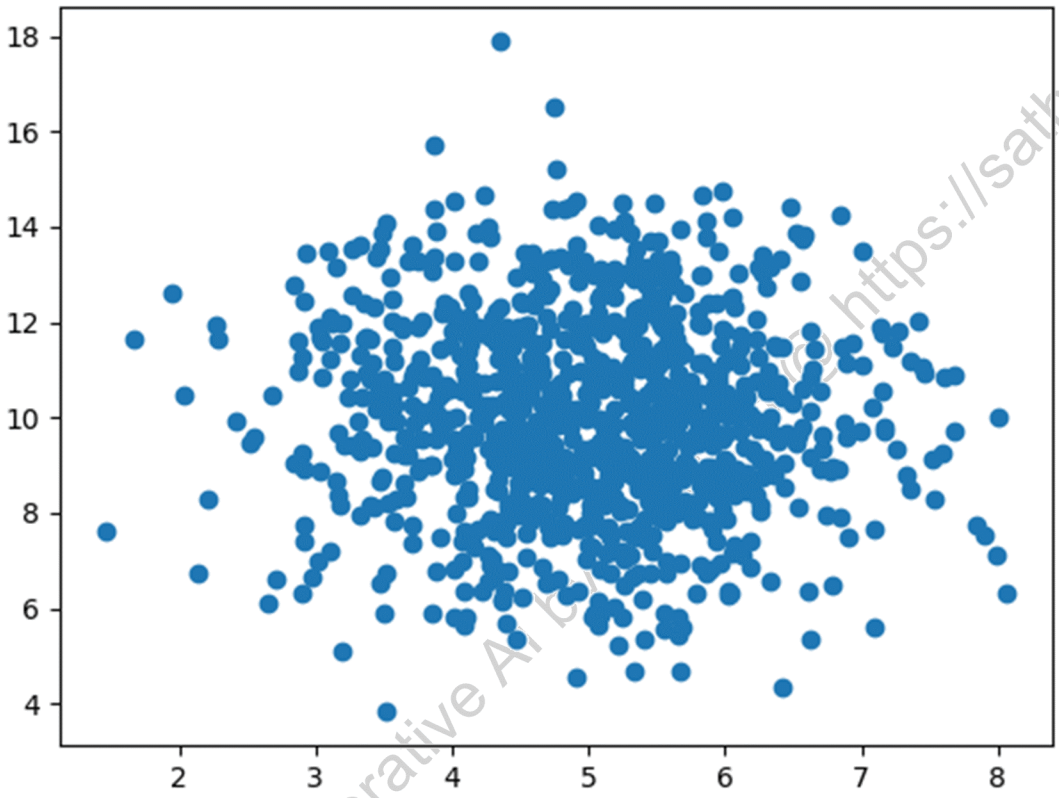
The second array will have the mean set to 10.0 with a standard deviation of 2.0:

Example

A scatter plot with 1000 dots:

```
import numpy
import matplotlib.pyplot as plt

x = numpy.random.normal(5.0, 1.0, 1000)
y = numpy.random.normal(10.0, 2.0, 1000)

plt.scatter(x, y)
plt.show()
```

Result:



Scatter Plot Explained

We can see that the dots are concentrated around the value 5 on the x-axis, and 10 on the y-axis.

We can also see that the spread is wider on the y-axis than on the x-axis.

Machine Learning - Linear Regression

Regression

The term regression is used when you try to find the relationship between variables.

In Machine Learning, and in statistical modeling, that relationship is used to predict the outcome of future events.

Syntax:

---

linregress(*x*, *y*=*None*, *alternative*='*two-sided*')

Calculate a linear least-squares regression for two sets of measurements.

Parameters:

x, y*array_like*

Two sets of measurements. Both arrays should have the same length N. If only *x* is given (and y=None), then it must be a two-dimensional array where one dimension has length 2. The two sets of measurements are then found by splitting the array along the length-2 dimension. In the case where y=None and *x* is a 2xN array, linregress(x) is equivalent to linregress(x[0], x[1]).

Deprecated since version 1.14.0: Inference of the two sets of measurements from a single argument *x* is deprecated will result in an error in SciPy 1.16.0; the sets must be specified separately as *x* and *y*.

alternative*{'two-sided', 'less', 'greater'}, optional*

Defines the alternative hypothesis. Default is 'two-sided'. The following options are available:

- 'two-sided': the slope of the regression line is nonzero
- 'less': the slope of the regression line is less than zero
- 'greater': the slope of the regression line is greater than zero

Added in version 1.7.0.

Returns:

Result ***LinregressResult instance***

The return value is an object with the following attributes:

Slope ***float***

Slope of the regression line.

Intercept ***float***

Intercept of the regression line.

r-value ***float***

The Pearson correlation coefficient. The square of rvalue is equal to the coefficient of determination.

p-value ***float***

The p-value for a hypothesis test whose null hypothesis is that the slope is zero, using Wald Test with t-distribution of the test statistic. See *alternative* above for alternative hypotheses.

Stderr ***float***

Standard error of the estimated slope (gradient), under the assumption of residual normality.

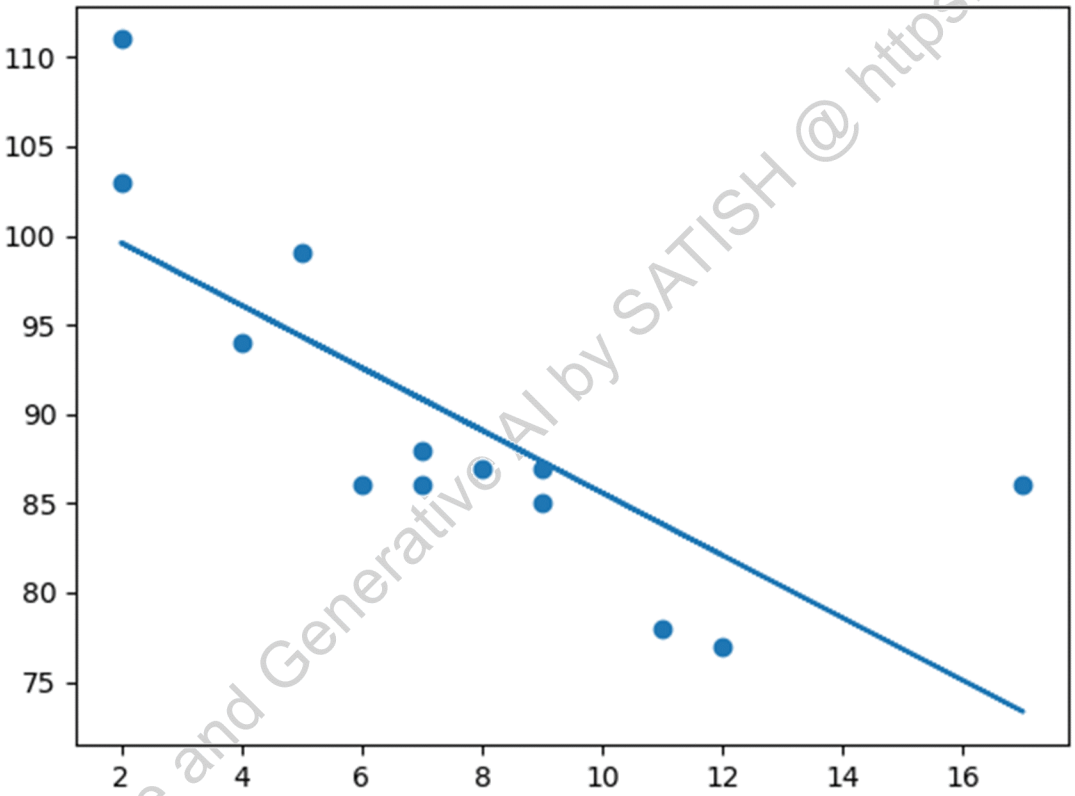
intercept_stderr*float*

Standard error of the estimated intercept, under the assumption of residual normality.

Linear Regression

Linear regression uses the relationship between the data-points to draw a straight line through all them.

This line can be used to predict future values.



How Does it Work?

Python has methods for finding a relationship between data-points and to draw a line of linear regression. We will show you how to use these methods instead of going through the mathematic formula.

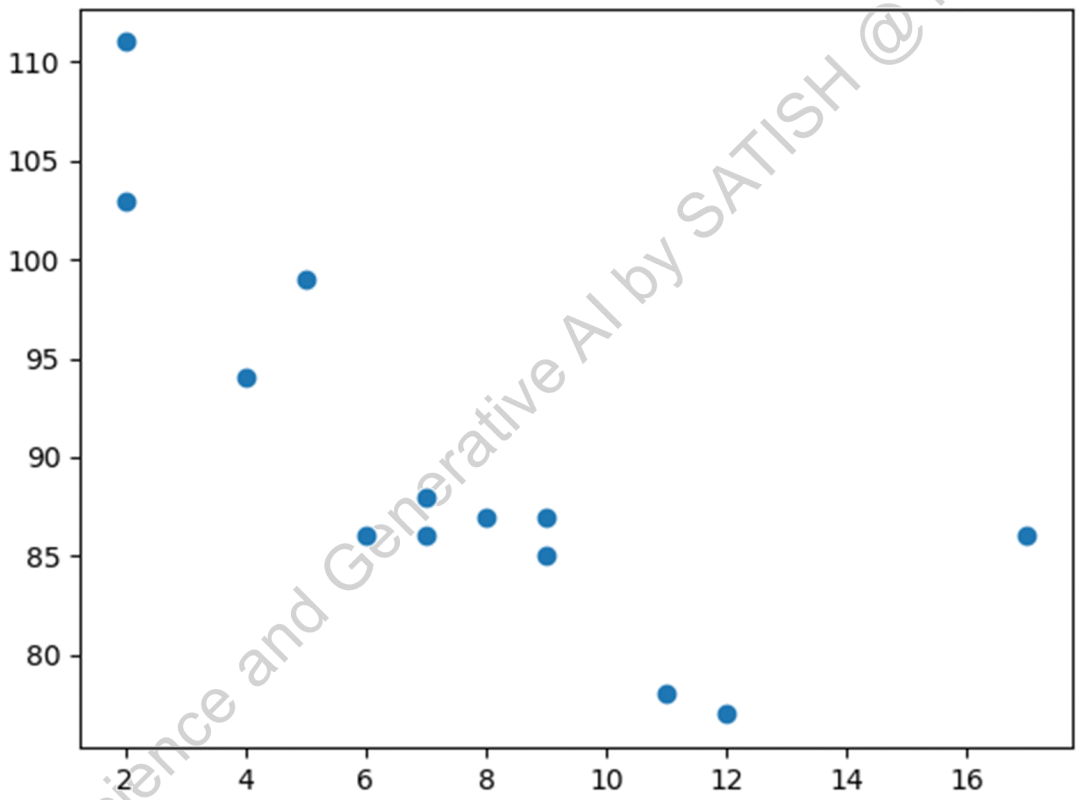
In the example below, the x-axis represents age, and the y-axis represents speed. We have registered the age and speed of 13 cars as they were passing a tollbooth. Let us see if the data we collected could be used in a linear regression:

Example

Start by drawing a scatter plot:

```
import matplotlib.pyplot as plt

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
plt.scatter(x, y)
plt.show()
```

Result:

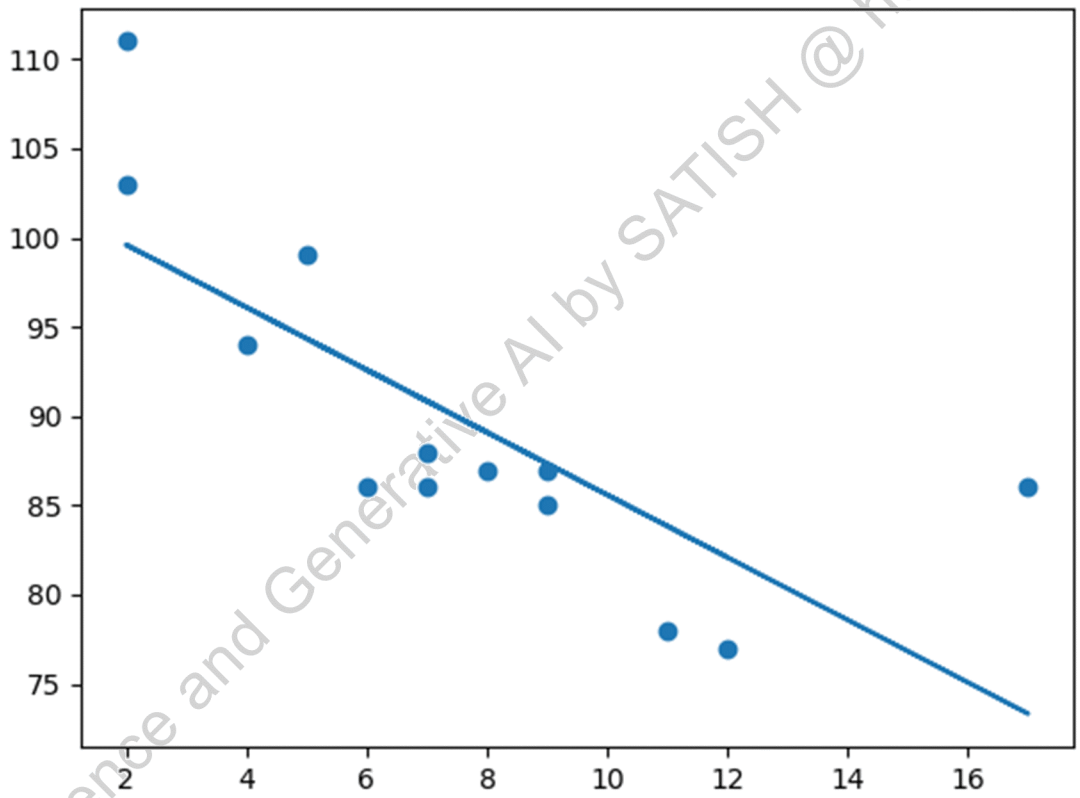


Example

Import scipy and draw the line of Linear Regression:

```
import matplotlib.pyplot as plt
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

Result:



Example Explained

Import the modules you need.

```
import matplotlib.pyplot as plt
from scipy import stats
```

Create the arrays that represent the values of the x and y axis:

```
x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

Execute a method that returns some important key values of Linear Regression:

```
slope, intercept, r, p, std_err = stats.linregress(x, y)
```

Create a function that uses the slope and intercept values to return a new value. This new value represents where on the y-axis the corresponding x value will be placed:

```
def myfunc(x):
  return slope * x + intercept
```

Run each value of the x array through the function. This will result in a new array with new values for the y-axis:

```
mymodel = list(map(myfunc, x))
```

Draw the original scatter plot:

```
plt.scatter(x, y)
```

Draw the line of linear regression:

```
plt.plot(x, mymodel)
```

Display the diagram:

```
plt.show()
```

R for Relationship

It is important to know how the relationship between the values of the x-axis and the values of the y-axis is, if there are no relationship the linear regression can not be used to predict anything.

This relationship - the coefficient of correlation - is called r.

The r value ranges from -1 to 1, where **0 means no relationship**, and 1 (and -1) means 100% related.

Python and the Scipy module will compute this value for you, all you have to do is feed it with the x and y values.

Example

How well does my data fit in a linear regression?

```
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

print(r)
```

Note: The result -0.76 shows that there is a relationship, not perfect, but it indicates that we could use linear regression in future predictions.

Predict Future Values

Now we can use the information we have gathered to predict future values.

Example: Let us try to predict the speed of a 10 years old car.

To do so, we need the same myfunc() function from the example above:

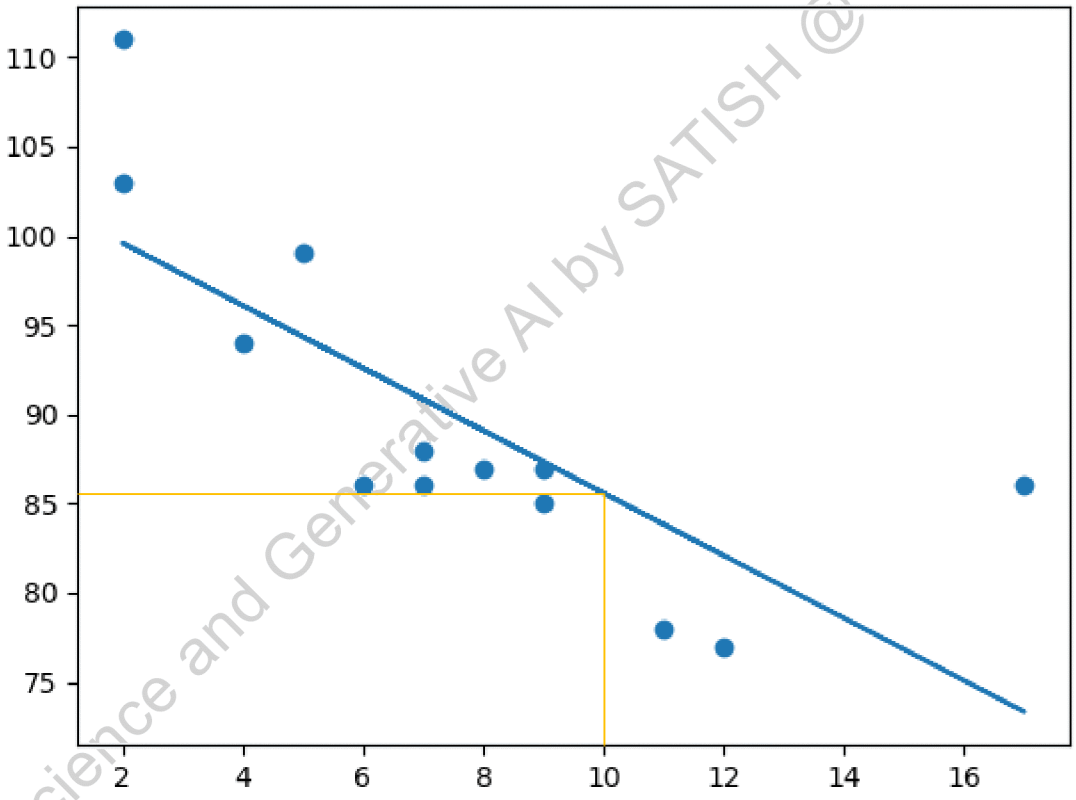
```
def myfunc(x):
  return slope * x + intercept
```

Example

Predict the speed of a 10 years old car:

```
from scipy import stats

x = [5,7,8,7,2,17,2,9,4,11,12,9,6]
y = [99,86,87,88,111,86,103,87,94,78,77,85,86]

slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept

speed = myfunc(10)

print(speed)
```

The example predicted a speed at **85.6**, which we also could read from the diagram:



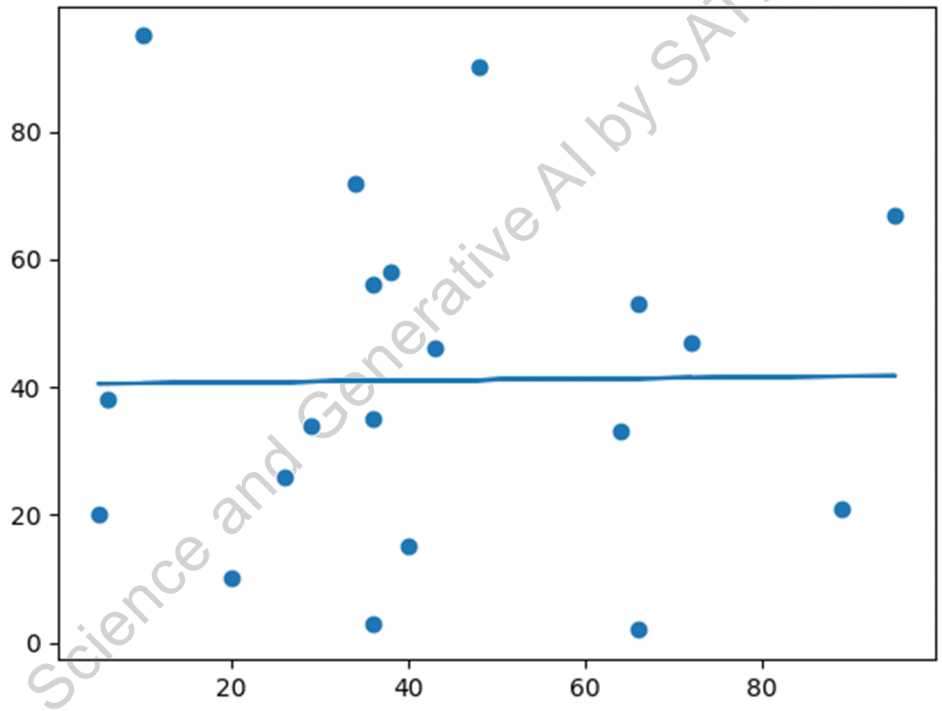
Bad Fit?

Let us create an example where linear regression would not be the best method to predict future values.

Example: These values for the x- and y-axis should result in a very bad fit for linear regression:

```
import matplotlib.pyplot as plt
from scipy import stats

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
slope, intercept, r, p, std_err = stats.linregress(x, y)

def myfunc(x):
  return slope * x + intercept
mymodel = list(map(myfunc, x))

plt.scatter(x, y)
plt.plot(x, mymodel)
plt.show()
```

Result:



And the r for relationship?

Example

You should get a very low r value.

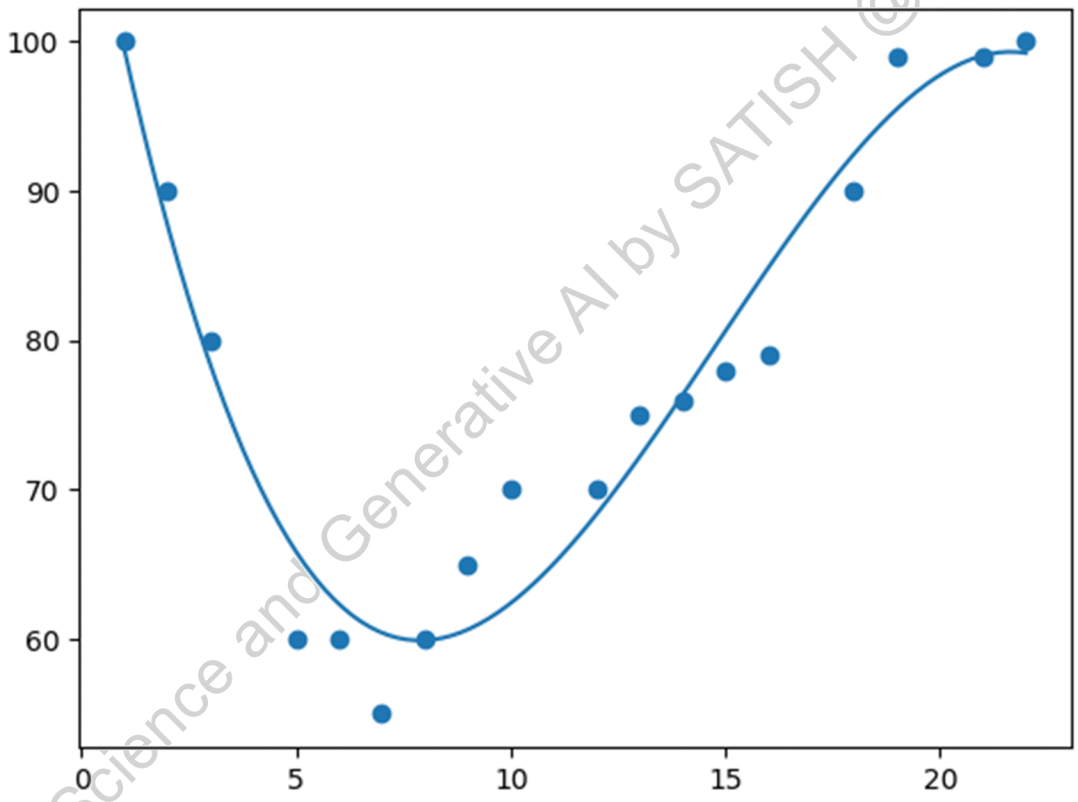
```
import numpy
from scipy import stats

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]
slope, intercept, r, p, std_err = stats.linregress(x, y)
print(r)
```

Machine Learning - Polynomial Regression

Polynomial Regression

If your data points clearly will not fit a linear regression (a straight line through all data points), it might be ideal for polynomial regression.

Polynomial regression, like linear regression, uses the relationship between the variables x and y to find the best way to draw a line through the data points.



How Does it Work?

Python has methods for finding a relationship between data-points and to draw a line of polynomial regression. We will show you how to use these methods instead of going through the mathematic formula.

In the example below, we have registered 18 cars as they were passing a certain tollbooth. We have registered the car's speed, and the time of day (hour) the passing occurred.

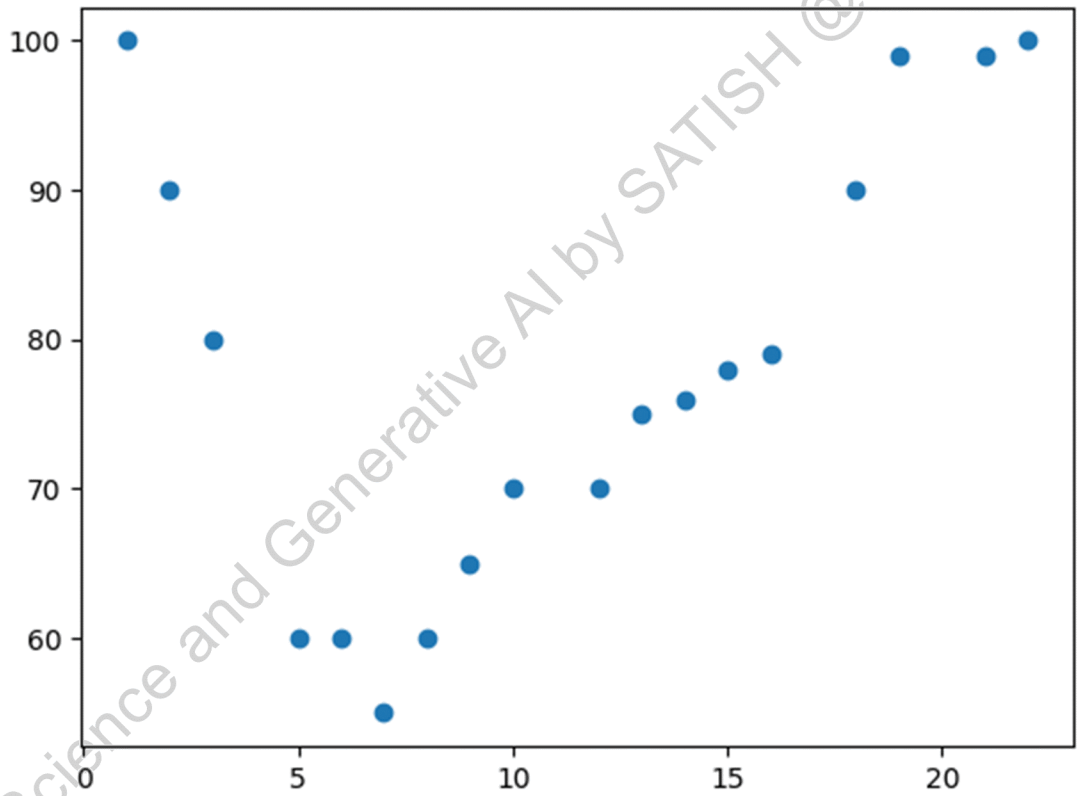
The x-axis represents the hours of the day and the y-axis represents the speed:

Example

Start by drawing a scatter plot:

```
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
plt.scatter(x, y)
plt.show()
```

Result:

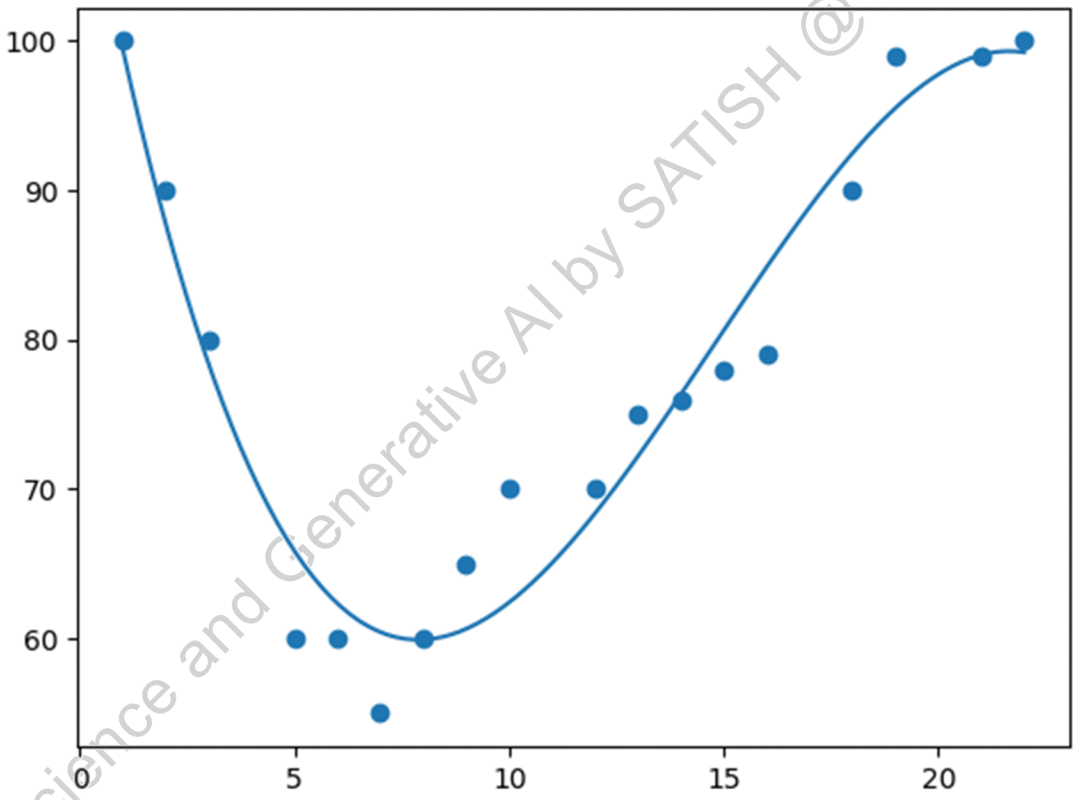


Example

Import numpy and matplotlib then draw the line of Polynomial Regression:

```
import numpy
import matplotlib.pyplot as plt

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

myline = numpy.linspace(1, 22, 100)

plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Result:



Example Explained

Import the modules you need.

You can learn about the NumPy module in our NumPy

You can learn about the SciPy module in our SciPy

```
import numpy
import matplotlib.pyplot as plt
```

Create the arrays that represent the values of the x and y axis:

```
x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

NumPy has a method that lets us make a polynomial model:

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

Then specify how the line will display, we start at position 1, and end at position 22:

```
myline = numpy.linspace(1, 22, 100)
```

Draw the original scatter plot:

```
plt.scatter(x, y)
```

Draw the line of polynomial regression:

```
plt.plot(myline, mymodel(myline))
```

Display the diagram:

```
plt.show()
```

R-Squared

It is important to know how well the relationship between the values of the x- and y-axis is, if there are no relationship the polynomial regression can not be used to predict anything.

The relationship is measured with a value called the r-squared.

The r-squared value ranges from 0 to 1, where 0 means no relationship, and 1 means 100% related.

Python and the Sklearn module will compute this value for you, all you have to do is feed it with the x and y arrays:

Example

How well does my data fit in a polynomial regression?

```
import numpy
from sklearn.metrics import r2_score

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

print(r2_score(y, mymodel(x)))
```

Note: The result 0.94 shows that there is a very good relationship, and we can use polynomial regression in future predictions.

Predict Future Values

Now we can use the information we have gathered to predict future values.

Example: Let us try to predict the speed of a car that passes the tollbooth at around the time 17:00:

To do so, we need the same mymodel array from the example above:

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))
```

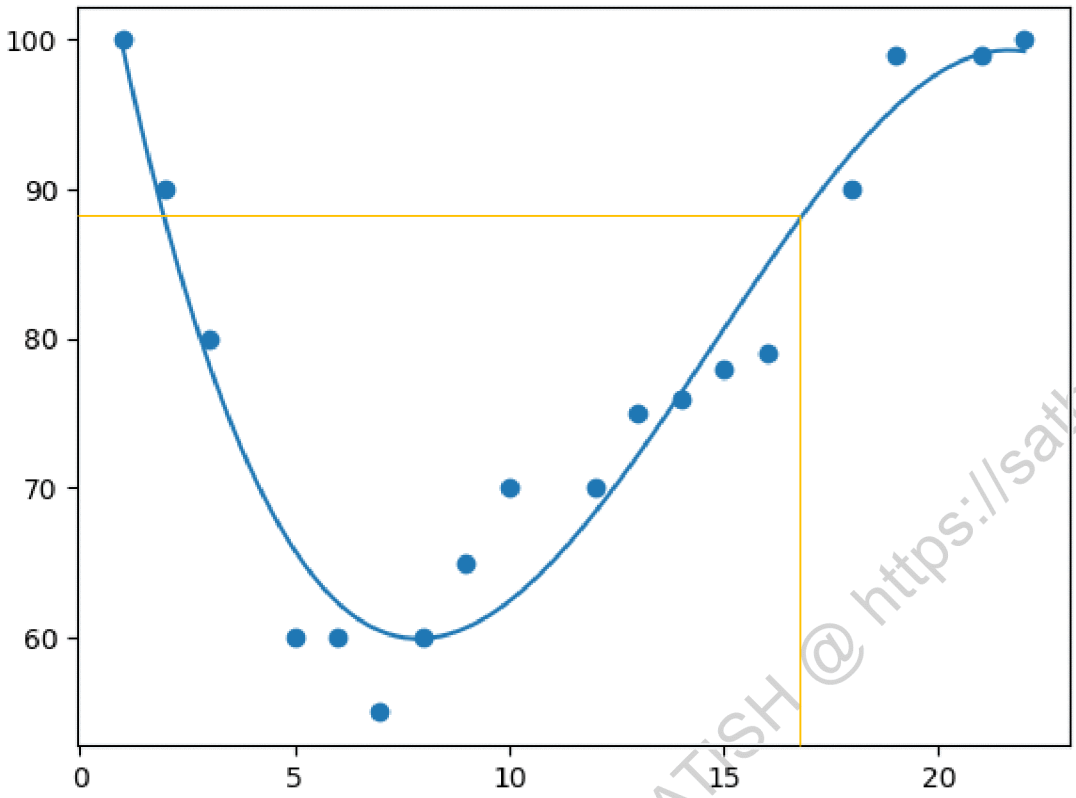
Example

Predict the speed of a car passing at 17:00:

```
import numpy
from sklearn.metrics import r2_score

x = [1,2,3,5,6,7,8,9,10,12,13,14,15,16,18,19,21,22]
y = [100,90,80,60,60,55,60,65,70,70,75,76,78,79,90,99,99,100]
```

```
mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

speed = mymodel(17)
print(speed)
```

The example predicted a speed to be 88.87, which we also could read from the diagram:



Bad Fit?

Let us create an example where polynomial regression would not be the best method to predict future values.

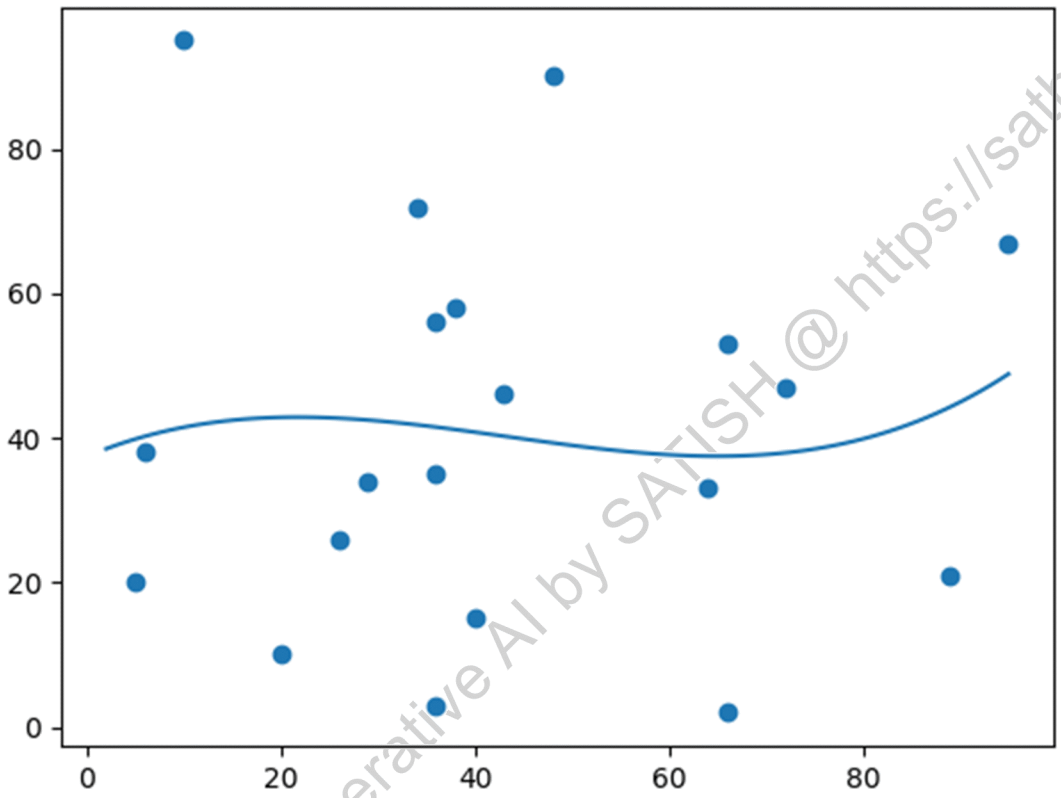
Example

These values for the x- and y-axis should result in a very bad fit for polynomial regression:

```
import numpy
import matplotlib.pyplot as plt
```

```
x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

myline = numpy.linspace(2, 95, 100)

plt.scatter(x, y)
plt.plot(myline, mymodel(myline))
plt.show()
```

Result:



And the r-squared value?

Example

You should get a very low r-squared value.

```
import numpy
from sklearn.metrics import r2_score

x = [89,43,36,36,95,10,66,34,38,20,26,29,48,64,6,5,36,66,72,40]
y = [21,46,3,35,67,95,53,72,58,10,26,34,90,33,38,20,56,2,47,15]

mymodel = numpy.poly1d(numpy.polyfit(x, y, 3))

print(r2_score(y, mymodel(x)))
```

Machine Learning - Multiple Regression

Multiple Regression

Multiple regression is like linear regression, but with more than one independent value, meaning that we try to predict a value based on **two or more** variables.

Take a look at the data set below, it contains some information about cars.

Car	Model	Volume	Weight	CO2
Toyota	Aygo	1000	790	99
Mitsubishi	Space Star	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Cooper	1500	1140	105
VW	Up!	1000	929	105
Skoda	Fabia	1400	1109	90
Mercedes	A-Class	1500	1365	92

Ford	Fiesta	1500	1112	98
Audi	A1	1600	1150	99
Hyundai	I20	1100	980	99
Suzuki	Swift	1300	990	101
Ford	Fiesta	1000	1112	99
Honda	Civic	1600	1252	94
Hundai	I30	1600	1326	97
Opel	Astra	1600	1330	97
BMW	1	1600	1365	99
Mazda	3	2200	1280	104
Skoda	Rapid	1600	1119	104
Ford	Focus	2000	1328	105
Ford	Mondeo	1600	1584	94
Opel	Insignia	2000	1428	99
Mercedes	C-Class	2100	1365	99
Skoda	Octavia	1600	1415	99
Volvo	S60	2000	1415	99
Mercedes	CLA	1500	1465	102
Audi	A4	2000	1490	104
Audi	A6	2000	1725	114
Volvo	V70	1600	1523	109
BMW	5	2000	1705	114

Mercedes	E-Class	2100 1605 115
Volvo	XC70	2000 1746 117
Ford	B-Max	1600 1235 104
BMW	2	1600 1390 108
Opel	Zafira	1600 1405 109
Mercedes	SLK	2500 1395 120

We can predict the CO2 emission of a car based on the size of the engine, but with multiple regression we can throw in more variables, like the weight of the car, to make the prediction more accurate.

How Does it Work?

In Python we have modules that will do the work for us. Start by importing the Pandas module.

```
import pandas
```

Learn about the Pandas module in our Pandas

The Pandas module allows us to read csv files and return a DataFrame object.

The file is meant for testing purposes only, you can download it here: data.csv

```
df = pandas.read_csv("data.csv")
```

Then make a list of the independent values and call this variable X.

Put the dependent values in a variable called y.

```
X = df[['Weight', 'Volume']]
y = df['CO2']
```

Tip: It is common to name the list of independent values with a upper case X, and the list of dependent values with a lower case y.

We will use some methods from the sklearn module, so we will have to import that module as well:

from sklearn import linear_model

From the sklearn module we will use the LinearRegression() method to create a linear regression object.

This object has a method called fit() that takes the independent and dependent values as parameters and fills the regression object with data that describes the relationship:

```
regr = linear_model.LinearRegression()
regr.fit(X, y)
```

Now we have a regression object that are ready to predict CO2 values based on a car's weight and volume:

```
#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300]])
```

Example

See the whole example in action:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

#predict the CO2 emission of a car where the weight is 2300kg, and the volume is 1300cm3:
predictedCO2 = regr.predict([[2300, 1300]])

print(predictedCO2)
```

Result:

[107.2087328]

We have predicted that a car with 1.3 liter engine, and a weight of 2300 kg, will release approximately 107 grams of CO2 for every kilometer it drives.

Coefficient

The coefficient is a factor that describes the relationship with an unknown variable.

Example: if x is a variable, then 2x is x two times. x is the unknown variable, and the number 2 is the coefficient.

In this case, we can ask for the coefficient value of weight against CO2, and for volume against CO2. The answer(s) we get tells us what would happen if we increase, or decrease, one of the independent values.

Example

Print the coefficient values of the regression object:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

print(regr.coef_)
```

Result:

```
[0.00755095 0.00780526]
```

Result Explained

The result array represents the coefficient values of weight and volume.

Weight: 0.00755095
Volume: 0.00780526

These values tell us that if the weight increase by 1kg, the CO2 emission increases by 0.00755095g.

And if the engine size (Volume) increases by 1 cm^3, the CO2 emission increases by 0.00780526 g.

I think that is a fair guess, but let test it!

We have already predicted that if a car with a 1300cm^3 engine weighs 2300kg, the CO2 emission will be approximately 107g.

What if we increase the weight with 1000kg?

Example

Copy the example from before, but change the weight from 2300 to 3300:

```
import pandas
from sklearn import linear_model

df = pandas.read_csv("data.csv")

X = df[['Weight', 'Volume']]
y = df['CO2']

regr = linear_model.LinearRegression()
regr.fit(X, y)

predictedCO2 = regr.predict([[3300, 1300]])

print(predictedCO2)
```

Result:

```
[114.75968007]
```

We have predicted that a car with 1.3 liter engine, and a weight of 3300 kg, will release approximately 115 grams of CO2 for every kilometer it drives.

Which shows that the coefficient of 0.00755095 is correct:

107.2087328 + (1000 * 0.00755095) = 114.75968