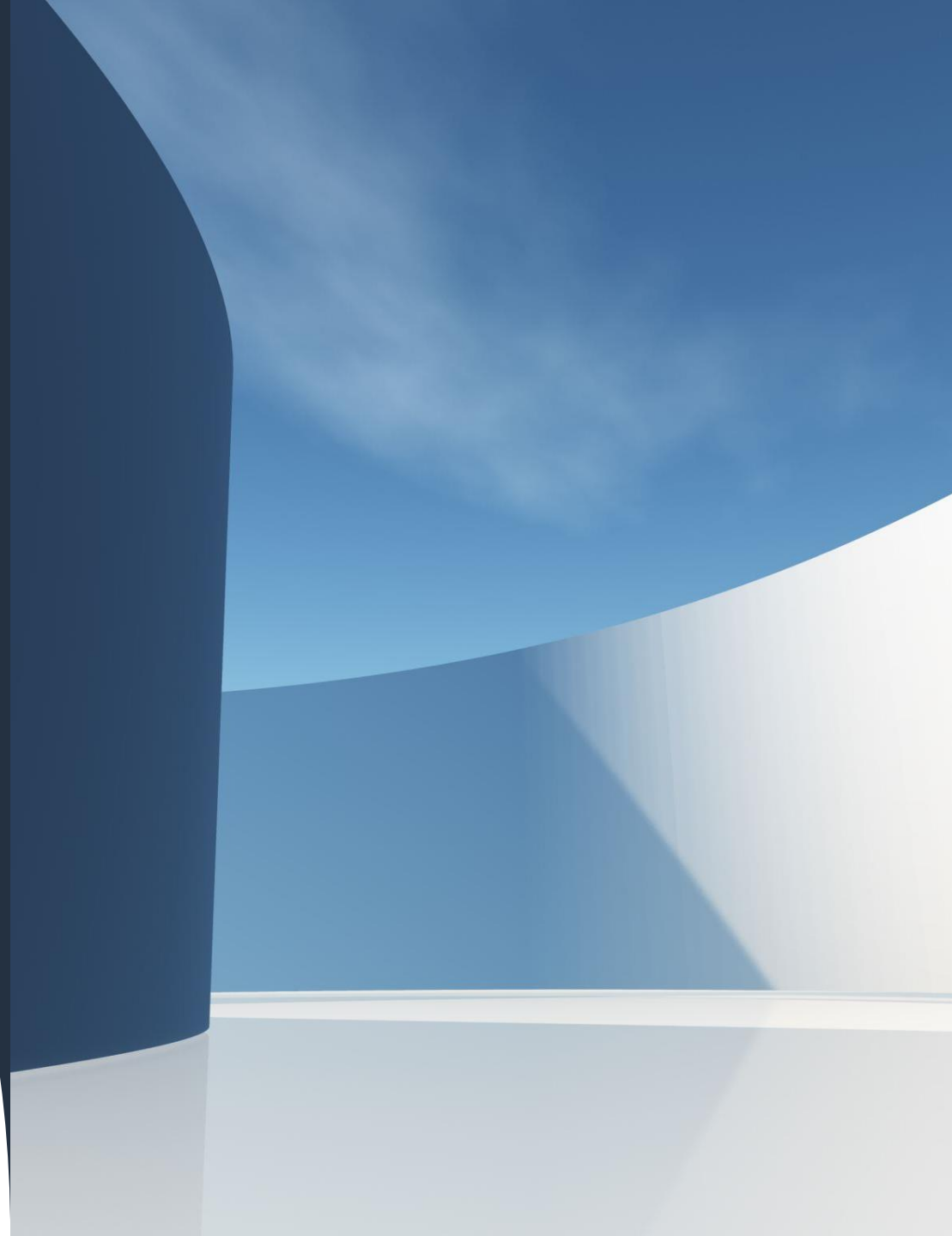
Artificial Intelligence with Python

Chapter 4 - Regression



REGRESSION IN ARTIFICIAL INTELLIGENCE

Regression in artificial intelligence (AI) refers to a supervised learning technique used for predicting continuous numerical values based on input features. In other words, regression models analyze the relationship between independent variables (features) and a dependent variable (target) to make predictions.

REGRESSION IN ARTIFICIAL INTELLIGENCE

Supervised Learning: Regression is a form of supervised learning, meaning that it requires labeled training data where both the input features and the corresponding target values are provided.

REGRESSION IN ARTIFICIAL INTELLIGENCE

Continuous Target Variable: Unlike classification, where the target variable is categorical, regression deals with continuous numerical values. For example, predicting house prices, stock prices, temperature, or sales revenue are common regression tasks.

REGRESSION IN ARTIFICIAL INTELLIGENCE

Modeling Relationships: The goal of regression is to model the relationship between input features and the target variable. The model learns from the training data to capture patterns and associations, enabling it to make predictions on new, unseen data.

REGRESSION IN ARTIFICIAL INTELLIGENCE

Linear and Non-linear Relationships: Regression models can capture both linear and non-linear relationships between features and the target variable. Simple linear regression models assume a linear relationship between the input features and the target, while polynomial regression, ridge regression, and other techniques can handle more complex relationships.

REGRESSION IN ARTIFICIAL INTELLIGENCE

Evaluation: Regression models are evaluated based on their ability to accurately predict the target variable. Common evaluation metrics include R-squared (coefficient of determination), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE), among others.

REGRESSION IN ARTIFICIAL INTELLIGENCE

Applications: Regression analysis is widely used across various domains, including finance, economics, healthcare, marketing, and engineering. It helps in making informed decisions, forecasting future trends, and understanding the relationships between variables.

Primer on lines and parabolas

A straight line in cartesian coordinate system (x , y) is represented as

$y = a+bx,$

where a and b (called coefficients) have some concrete values. Let us consider, for example,

the line $y = 1 + 2 x$. This means that for any value of x we can compute the corresponding value for y according to this rule. As a result, we get a table of (x , y) values such as

x	y	(x,y)
0	1+2*0 = 1	(0,1)
1	1+2*1 = 3	(1,3)
2	1+2*2 = 5	(2,5)
3	1+2*3 = 7	(3,7)
4	1+2*4 = 9	(4,9)

Primer on lines and parabolas

These points can be plotted in a coordinate system to reveal the graph of the line. It should be noted here that one can use any value for x, positive or negative, and get the value for y so there are infinitely many points (x , y) that make up the line's graph.

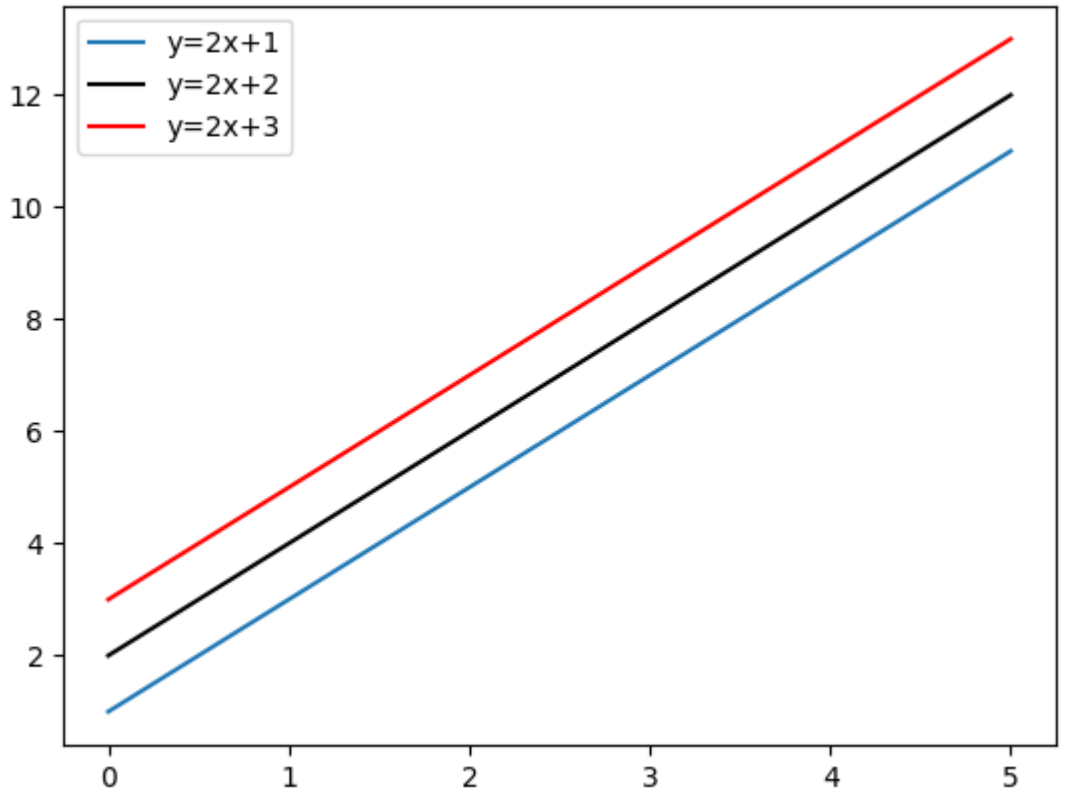
Having the values at hand they can be plotted using numpy and matplotlib as follows:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([0,1,2,3,4])
y = np.array([1,3,5,7,9])
plt.scatter(x,y)
plt.plot(x,y)

plt.show()
```

Primer on lines and parabolas

The figure below shows the graphs of three lines plotted with quite dense set of x values but the basic idea remains the same.



Primer on lines and parabolas

We can see that the lines have the same slope (steepness). But they intersect the y-axis at different points. For the red line this y-intercept is 3, for the black line it is 2 and for the blue line it is 1.
For each of the lines we see that the value of the coefficient a equals the y-intercept.
This follows by putting x = 0 in the formula of the line.
Recall that the slope of any line is defined as the ratio between the y-distance and x-distance
(measured at any point of the line). Here the slope equals 2 for each line and it is given by the coefficient b.

In the figure below we have also three lines but now the slope b varies from 2 to 4.
The y-intercept a remains the same at 1. Note the different behaviour of the lines.

Primer on lines and parabolas

A line is also a polynomial of degree 1 since it contains only terms involving first powers of x.
It is also called the linear function because its graph is a straight line.

Polynomial of degree 2 might contain second powers of x (squares of x) and it is also called the quadratic polynomial.
Cubic polynomial is of degree 3 and it contains cubes of x i.e. terms such as x^3.

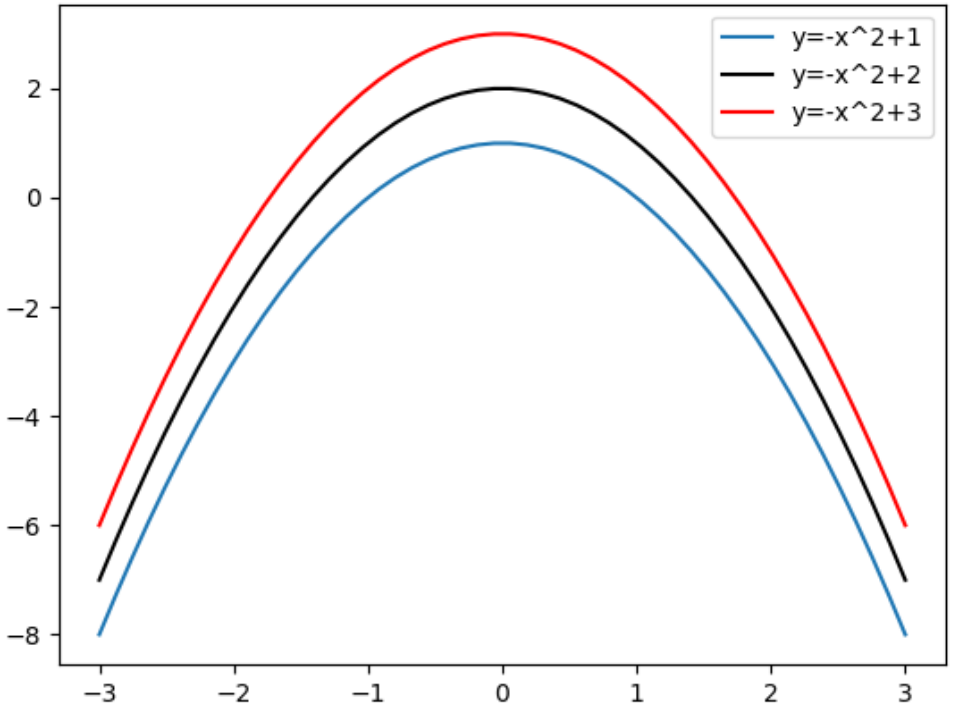
Primer on lines and parabolas

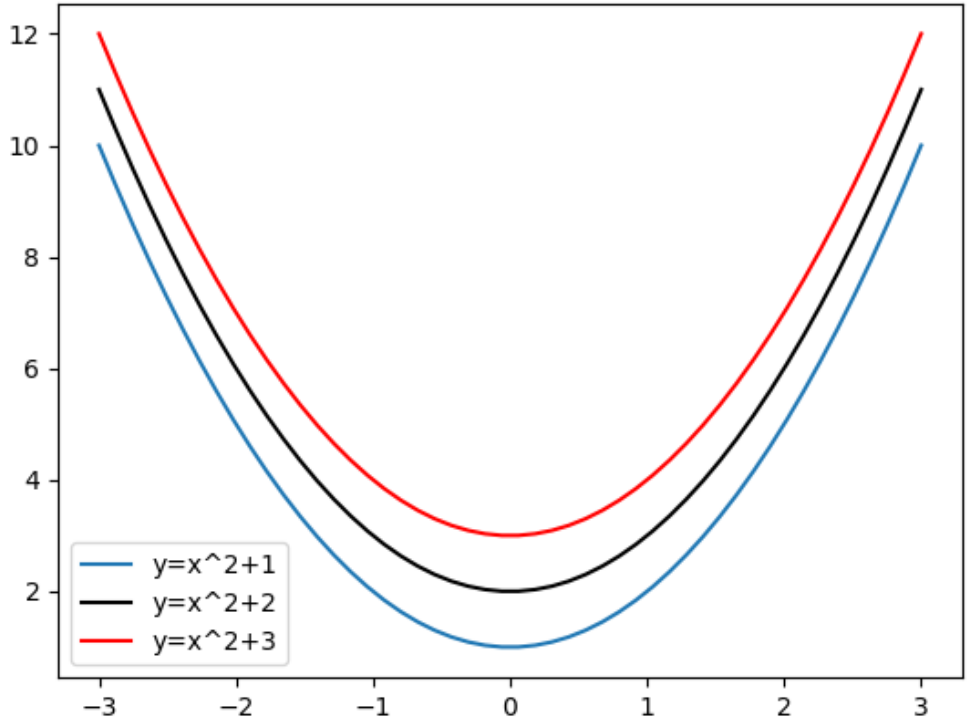
Next, consider parabolas given by the quadratic polynomial (of degree 2) as

$y = ax^2+bx+c.$

Also here we talk about y-intercept, which is now the coefficient c. Parabolas with $a>0$ open upward while $a<0$ means that the parabola open downward. Some examples:

Primer on lines and parabolas



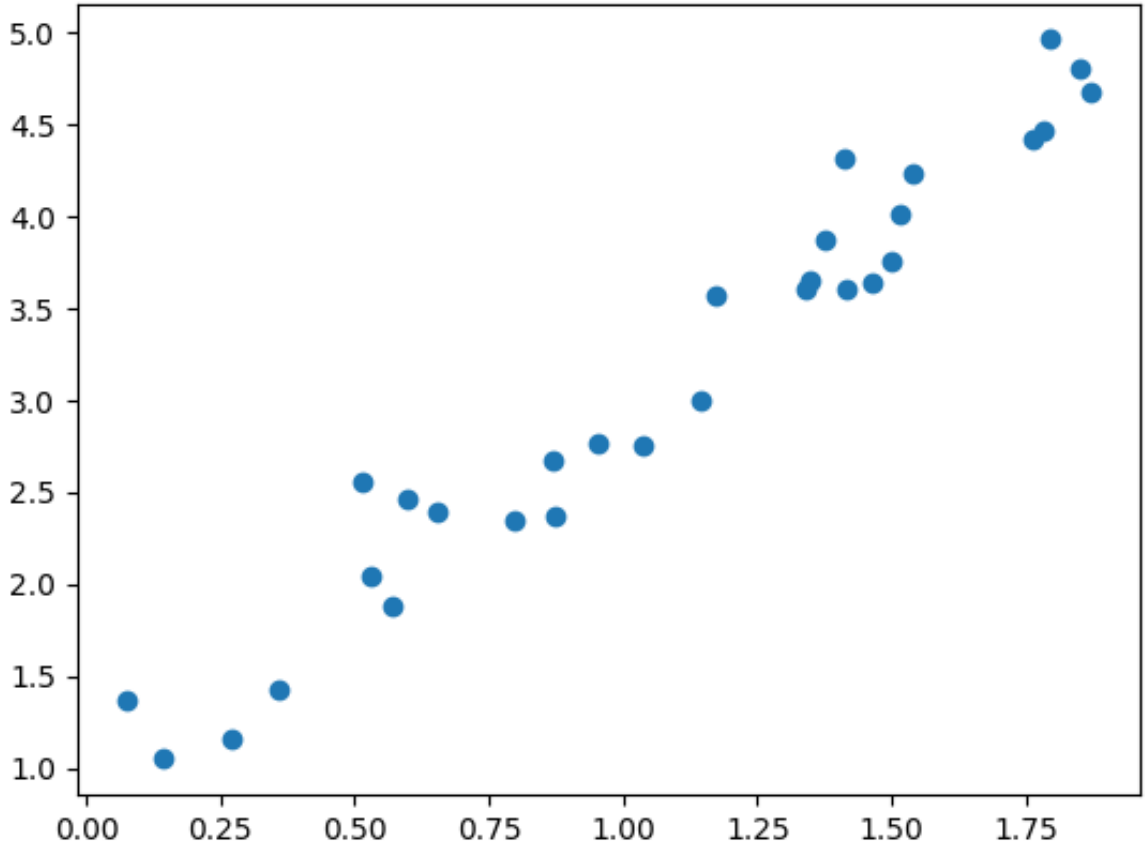


REGRESSION IN ARTIFICIAL INTELLIGENCE

- In simple linear regression the goal is to investigate the relationship between an independent variable (call it x) and a dependent variable (call it y).
- Independent variables are also called features, predictors, explanatory variables or inputs. Dependent variables are also called target variable, response variable or output.

REGRESSION IN ARTIFICIAL INTELLIGENCE

- Suppose we are given a set observations (x,y) in tabular format. Plotting them in Cartesian coordinate system might reveal the following illustration



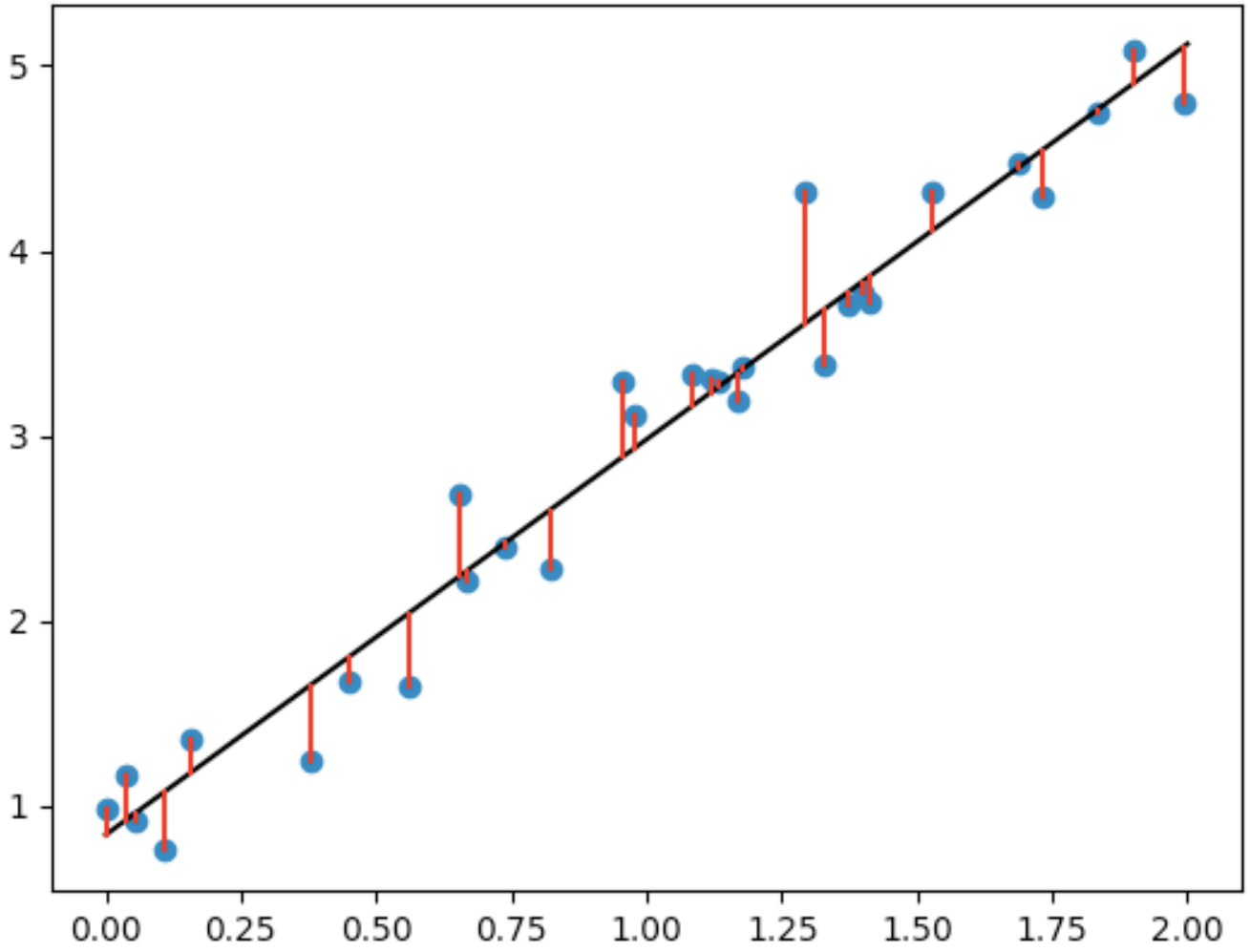
REGRESSION IN ARTIFICIAL INTELLIGENCE

It appears that the x-values and y-values have a linear dependence. Recalling that linear dependence is written as the formula $y = a + b x$, where a signifies the y-intercept of the line and b is the slope of the line. Now, a human being is quite good at drawing a line which best follows the observations. With careful work she could also find a and b quite accurately.

REGRESSION IN ARTIFICIAL INTELLIGENCE

But for a machine to do the same the situation is much more difficult. Systematically trying various values for a and b might be one solution but both have infinite continuum of possible values. How to evaluate the accuracy of the line? Note that a machine has no visual sense at all, it only sees raw numbers x , y , a and b. A popular choice to evaluate the accuracy is to look at the total error that we make when we represent the dataset with a straight line. In other words, we compute the difference $y_i - (a + bx_i)$ for all values of i. Pictorially this difference is depicted in red vertical lines below for each data point.

REGRESSION IN ARTIFICIAL INTELLIGENCE



REGRESSION IN ARTIFICIAL INTELLIGENCE

This difference is then squared to make it non-negative (we don't care about the direction of the error, and to allow differentiation, and to penalize more on big errors). Finally, the squares are summed together so the end result is

$$\sum (y_i - a - bx_i)^2$$

REGRESSION IN ARTIFICIAL INTELLIGENCE

This single value indicates how accurate the line is across all data points.
Starting from next page we make all this more precise...

Here i is an index which is used to index the tabulated values of x and y i.e.

i	x_i	y_i
1	x_i	y_1
2	x_2	y_2
3	x_3	y_3
4	x_4	y_4
n	x_n	y_n

It does not matter whether indexing starts from 0 or 1.

REGRESSION IN ARTIFICIAL INTELLIGENCE MATHS FORMULA

In simple linear regression we assume that the underlying model is the linear model.

$$\hat{y} = a + bx.$$

Here the notation $\hat{y}$ signifies the predicted value of y to distinguish it from the actual observed y. We form the residual sum of squares as

$$RSS = \sum (y_i - \hat{y}_i)^2 = \sum (y_i - a - bx_i)^2 .$$

We would like to make this value as small as possible in order to have the smallest total error.

Formula for a and b value

Solving this for b results in

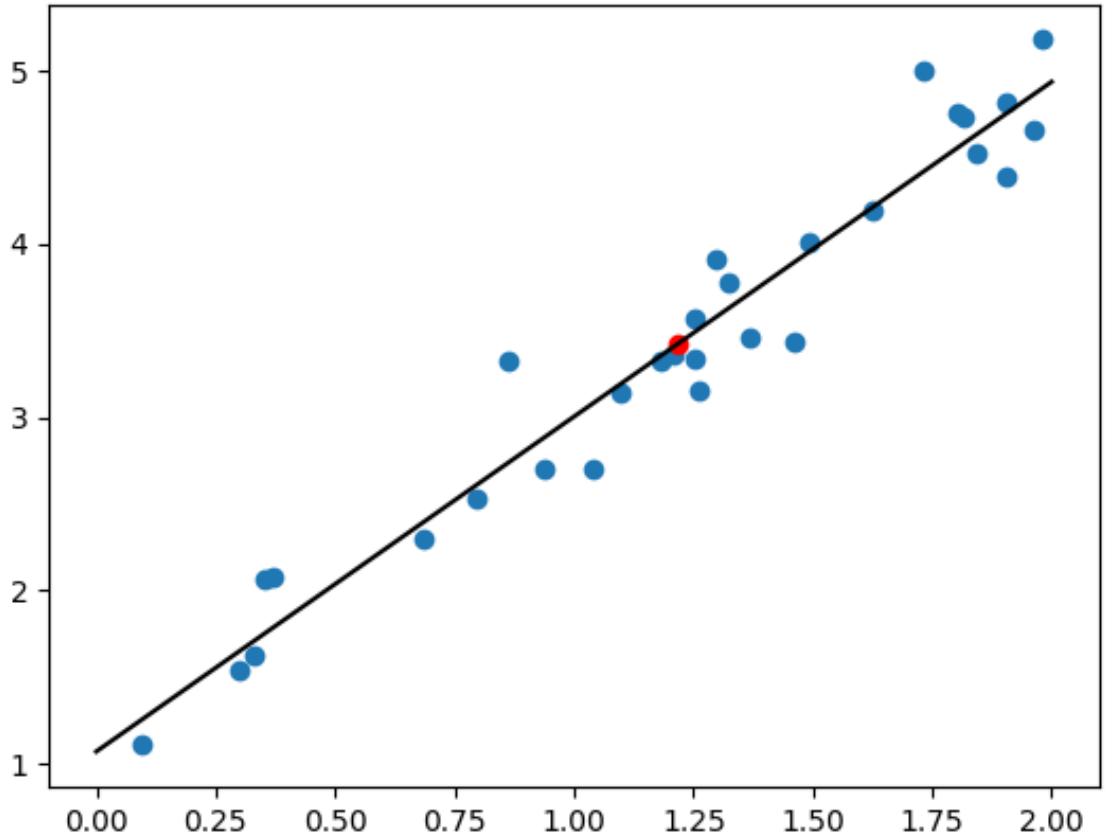
$$b = \frac{\sum(x_i y_i - x_i \overline{y})}{\sum(x_i^2 - x_i \overline{x})} = \frac{\sum x_i y_i - n\overline{xy}}{\sum x_i^2 - n\overline{x}^2}$$

$$a + b\overline{x} = \cdots = \overline{y}$$

Regression in simplest term

This means that the regression line y = a + b x passes through the point $(\overline{x},\overline{y})$ showing how linear regression is "regression to the mean" in its purest form.

Using the regression line we can predict what is the corresponding value for any x value



MODEL EVALUATION

Once the model has been trained it is essential to be able to tell if the end result is any good. To this end, the accuracy of the model can be evaluated by computing the quantity

$$RMSE = \sqrt{\frac{1}{n}\sum(y_i - \hat{y}_i)^2}$$

which is called the root mean square error. The reason for taking the square root here is that it makes the error quantity to be in same units as the target variable y. RMSE measures how much the predictions differ from observed values in average.

Note that

$$RMSE = \frac{1}{\sqrt{n}}\sqrt{RSS}.$$

So, since we have mimised RSS when doing linear regression we have also obtained the minimal RMSE. In other words, this is the best that our linear model can do in this regard.

There are also other error quantities such as the mean absolute error

$$MAE = \frac{1}{n}\sum |y_i - \hat{y}_i|$$

and mean squared error

$$MSE = \frac{1}{n}\sum (y_i - \hat{y}_i)^2.$$

MAE also measures the average deviation between predicted and observed values but in different scale than RMSE and MSE.

Another standard measure of goodness of fit is the coefficient of determination or R2 value defined by

$$R^2 = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \overline{y})^2}.$$

This value is between 0 and 1 (or 0% and 100%). In the best fit case we have $R^2 = 1$ since all predicted values match actual values perfectly. In the baseline case we would always predict $overl \in ey$ which would correspond to value $R^2 = 0$.

Residual Sum of Squares (RSS):

RSS is the sum of the squared differences between the predicted values and the actual values.

It measures the total error of the model by summing the squares of the residuals.

Root Mean Squared Error (RMSE):

- RMSE is the square root of the mean squared error (MSE).
- It represents the average magnitude of the errors between predicted and actual values.
- RMSE is particularly useful because it's in the same units as the target variable, making it easier to interpret.

Mean Absolute Error (MAE):

- MAE is the mean of the absolute differences between the predicted values and the actual values.
- It measures the average magnitude of errors without considering their direction.
- MAE is less sensitive to outliers compared to RMSE.

Mean Squared Error (MSE):

- MSE is the mean of the squared differences between the predicted values and the actual values.
- It measures the average squared error of the model.
- MSE is widely used in optimization algorithms due to its differentiable and convex nature.

Use of pandas Library: The pandas library in Python is a powerful tool for data manipulation and analysis. It provides easy-to-use data structures and functions designed to work with structured or tabular data, making it an essential tool for data scientists, analysts, and researchers.

sklearn

scikit-learn, often abbreviated as sklearn, is a popular Python library for machine learning. It provides a wide range of tools for various machine learning tasks such as classification, regression, clustering, dimensionality reduction, and model selection.

Supervised Learning: For tasks where you have labeled data.

Classification: Predicting categories or classes.

Regression: Predicting continuous values.

SIMPLE LINEAR REGRESSION IN PYTHON

Here we get ourselves familiar with doing simple linear regression in Python using the fundamental mathematical formulas discussed earlier.
The first step is import data from datafile to python using pandas and inspect the data in raw numbers.

```
import pandas as pd
import matplotlib.pyplot as plt
data = pd.read_csv("linreg_data.csv",skiprows=0,names=["x","y"])
head(data)
```

It is also a good idea to form visual understanding of the data by plotting it

```
xpd = data["x"]
ypd = data["y"]
n = xpd.size
plt.scatter(xpd,ypd)
plt.show()
```

The computation of the means of x and y using numpy is as easy as

```
xbar = np.mean(xpd)
ybar = np.mean(ypd)
```

The computation of the means of x and y using numpy is as easy as

```
xbar = np.mean(xpd)
ybar = np.mean(ypd)
```

The term $\sum x_i y_i$ in b is also easy to find in numpy using elementwise operations for arrays

```
term1 = np.sum(xpd*ypd)
```

Alternatively one could loop over indices and compute the sum directly but this is not recommended. Instead one should learn to use the efficient array operations in numpy.

Similarly, $\sum x_i^2$ is computed as

```
term2 = np.sum(xpd**2)
```

Now, b and a are easy to get

```
b = (term1-n*xbar*ybar)/(term2-n*xbar*xbar)
a = ybar - b*xbar
```

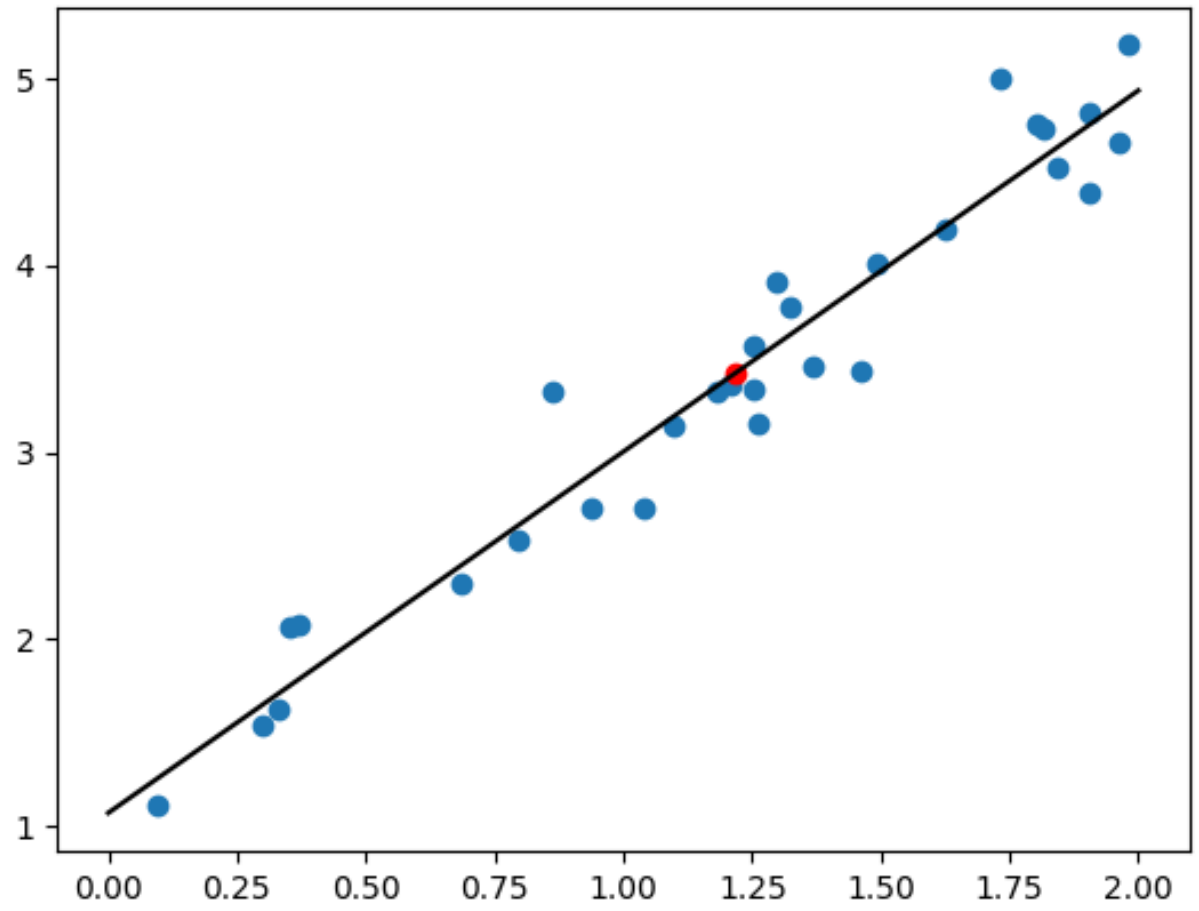
We obtain the values

a=0.8403987072579042

b= 2.138775903173365

This allows us to plot the results i.e. the regression line in the same graph with data points. For verification, we also plot the single point $(overl \in e\{x\}, overl \in e\{y\})$ in red.

```
x = np.linspace(0,2,100)
y = a+b*x
plt.plot(x,y,color="black")
plt.scatter(xpd,ypd)
plt.scatter(xbar,ybar,color="red")
plt.show()
```


5
4
3
2
1
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Using the regression line we can predict what is the corresponding value for any x value

```
xval = 0.50
yval = a+b*xval
```

and we get the value

yval = 1.9097866588445866

Better yet, using numpy we can use several values for x and predict them all as

```
xval = np.array([0.5,0.75,0.90])
yval = a+b*xval
print(yval)
```

The model evaluation or the computation of error indicators can also be done now

```
yhat = a+b*xpd
RSS = np.sum((ypd-yhat)**2)
print("RSS =",RSS)
RMSE = np.sqrt(np.sum((ypd-yhat)**2)/n)
print("RMSE=",RMSE)
MAE = np.sum(np.abs(ypd-yhat))/n
print("MAE =",MAE)
MSE = np.sum((ypd-yhat)**2)/n
print("MSE =",MSE)
R2 = 1-np.sum((ypd-yhat)**2)/np.sum((ypd-ybar)**2)
print("R2  =",R2)
```

This gives the values

RSS = 1.9762720606619886
RMSE= 0.2566626878909352
MAE = 0.19969075618017426
MSE = 0.06587573535539962
R2 = 0.9588798319617569

The R2 value shows very nice agreement between the data and the model. The other quantities here must be interpreted with more care since they depend on the scale of y values.

In addition to low-level approach of doing simple linear regression (see pt. 1) there is also a possibility to use high-level functions from module sklearn. It allows for streamlined treatment of solving the regression problem and also introduces us the module for other, more advanced uses. Moreover, sklearn is very convenient in evaluating the performance of machine learning models by using the training and test sets separately as we will see.

The first thing to do is import modules. For sklearn we import only the linear model

```
import numpy as np
from sklearn import linear_model
```

Next, we read in data and convert it into 2D arrays

```
my_data = np.genfromtxt('linreg_data.csv', delimiter=',')
xp = my_data[:,0]
yp = my_data[:,1]
xp = xp.reshape(-1,1)
yp = yp.reshape(-1,1)
```

Model is created and trained in just two lines of code

```
regr = linear_model.LinearRegression()
regr.fit(xp, yp) # fitting the model=training the model
```

The coefficieints a and b are now attributes of regr object

```
print(regr.coef_,regr.intercept_)
```

This gives us the same exact results as our low-level approach above.

Making predictions is done as follows

```
xval = np.full((1,1),0.5)

yval = regr.predict(xval)

print(yval)
```

Also this gives the same results as above.

Plotting of the regression line can be done by first predicting y-values for some appropriate x-values:

```
xval = np.linspace(-1,2,20).reshape(-1,1)

yval = regr.predict(xval)

plt.plot(xval,yval) # this plots the line

plt.scatter(xp,yp,color="red")

plt.show()
```

Regarding the metrics of accuracy introduced earlier they are readily available in sklearn as from sklearn import metrics

```
yhat = regr.predict(xp)

print('Mean Absolute Error:', metrics.mean_absolute_error(yp, yhat))

print('Mean Squared Error:', metrics.mean_squared_error(yp, yhat))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(yp, yhat)))

print('R2 value:', metrics.r2_score(yp, yhat))
```

Here the key takeaway is that sklearn streamlines and simplifies the whole process of training a model and using it to make predictions. These steps apply also to other models than just simple linear regression in sklearn.

Polynomial regression

If the data points do not see to follow a linear trend then it might be useful to try higher order polynomials as model. Quadratic model (degree 2) is of the form

$$\hat{y} = ax^2 + bx + c$$

Cubic model (degree 3) is

$$\hat{y} = ax^3 + bx^2 + cx + d$$

and so on. One can find out the mathematical formulas as in the linear case but the derivation becomes much more involved. The end result for quadratic regression is

$$a = \frac{S_{x^2y}S_{xx} - S_{xy}S_{xx^2}}{S_{xx}S_{x^2x^2} - S_{xx^2}^2}$$

$$b = \frac{S_{xy}S_{x^2x^2} - S_{x^2y}S_{xx^2}}{S_{xx}S_{x^2x^2} - S_{xx^2}^2}$$

$$c = \overline{y} - b\overline{x} - a(\sum x_i^2)/n$$

where

$$S_{xx} = \sum x_i^2 - n\overline{x}^2$$

$$S_{xy} = \sum x_i y_i - n\bar{x}\bar{y}$$

$$S_{xx^2} = \sum x_i^3 - \bar{x}\sum x_i^2$$

$$S_{x^2y} = \sum x_i^2 y_i - \bar{y}\sum x_i^2$$

$$S_{x^2x^2} = \sum x_i^4 - (\sum x_i^2)^2/n$$

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

data_pd = pd.read_csv("quadreg_data.csv",skiprows=0,names=["x","y"])
print(data_pd)

xpd = np.array(data_pd[["x"]])
ypd = np.array(data_pd[["y"]])
n = xpd.size

xbar = np.mean(xpd)
ybar = np.mean(ypd)

xpd = xpd.reshape((n,))
ypd = ypd.reshape((n,))
```

Next we compute the five sums appearing above

```
Sxx = np.sum(xpd**2)-n*xbar**2
Sxy = np.dot(xpd,ypd)-n*xbar*ybar
Sxx2 = np.sum(xpd**3)-xbar*np.sum(xpd**2)
Sx2y = np.sum(xpd**2*(ypd))-ybar*np.sum(xpd**2)
Sx2x2 = np.sum(xpd**4)-(np.sum(xpd**2)**2)/n
```

This allows us to obtain the coefficients a,b,c

a = (Sx2y*Sxx-Sxy*Sxx2)/(Sxx*Sx2x2-Sxx2**2)

b = (Sxy*Sx2x2-Sx2y*Sxx2)/(Sxx*Sx2x2-Sxx2**2)

c = ybar-b*xbar-a*np.sum(xpd**2)/n

We get the values

a= 1.0158122820554032 b= -0.010091535235750588 c= 0.99140532143387

Using these numbers the fitted parabola can be plotted as

```
x = np.linspace(-1,1,100)

y = a*x**2 + b*x +c

plt.plot(x,y)

plt.scatter(xpd,ypd,color="black")

plt.show()
```

Also, we may use the formula of this parabola to predict y for any given value(s) x.

The performance of the model is quantified using RMSE and R^2 value. They become

```
yhat = a*xpd**2 + b*xpd +c

RMSE = np.sqrt(np.sum((ypd-yhat)**2)/n)

print(RMSE)

R2 = 1-np.sum((ypd-yhat)**2)/np.sum((ypd-ybar)**2)

print(R2)
```

Polynomial regression with sklearn

Again, we may resort to just using the high level sklearn module to do polynomial regression.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

Read data in as before

```
data_pd = pd.read_csv("quadreg_data.csv",skiprows=0,names=["x","y"])
print(data_pd)

xpd = np.array(data_pd[["x"]])
ypd = np.array(data_pd[["y"]])
xpd = xpd.reshape(-1,1)
ypd = ypd.reshape(-1,1)
```

Set up the polynomial regression model of appropriate degree

```
poly_reg = PolynomialFeatures(degree=2)
X_poly = poly_reg.fit_transform(xpd)
pol_reg = LinearRegression()
pol_reg.fit(X_poly, ypd)
```

Here X_poly is an array which contains powers of x as columns. The powers are increasing from 0 to 2 and they are needed in order to use sklearn for quadratic regression.

Same procedure for x is needed when predicting values of qudratic model. We see it here when we plot the results

```
plt.scatter(xpd, ypd, color='red')
xval = np.linspace(-1,1,10).reshape(-1,1)
plt.plot(xval, pol_reg.predict(poly_reg.fit_transform(xval)), color='blue')
plt.show()
```

We see that we get the same results as above for the coefficients a,b and c

```
print(pol_reg.coef_)
print("c=",pol_reg.intercept_)
```

THANKS