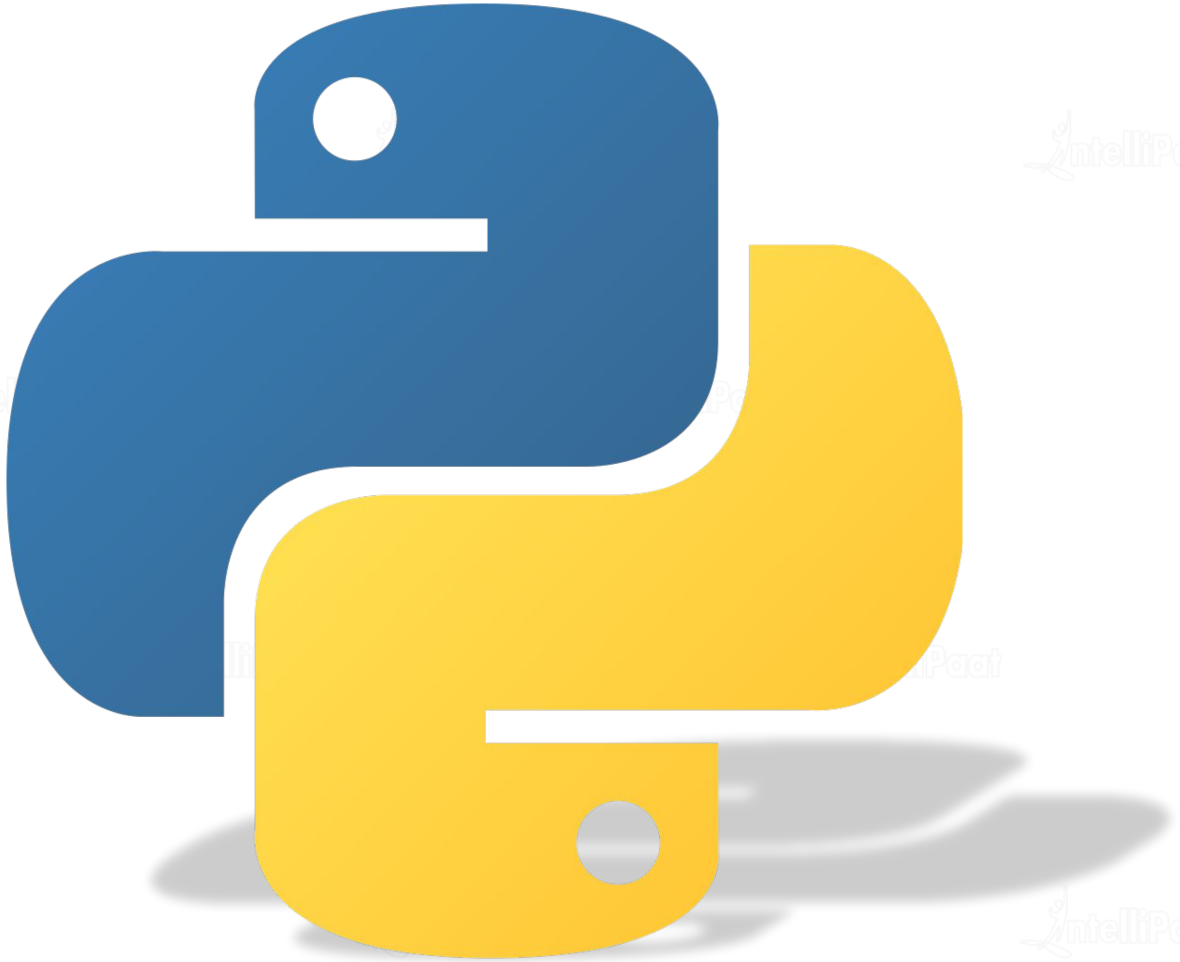
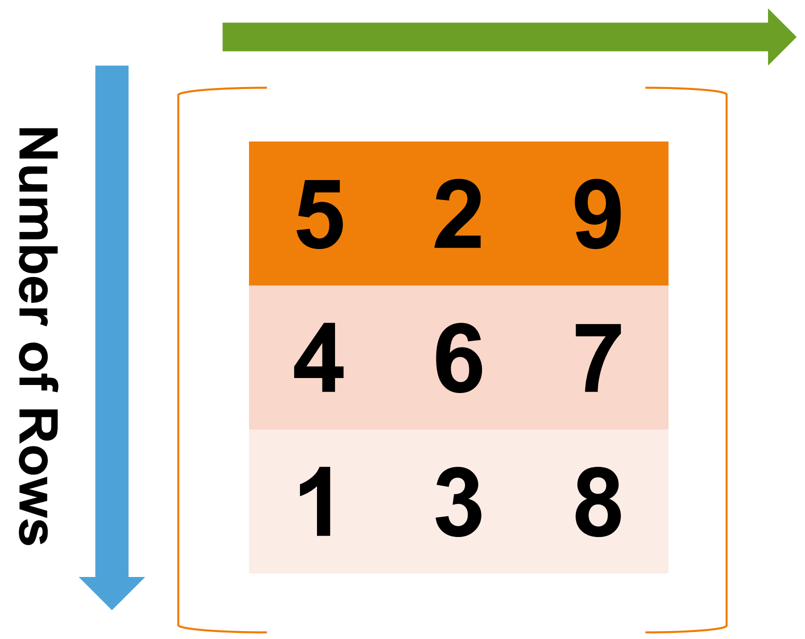
Linear Algebra For Machine Learning



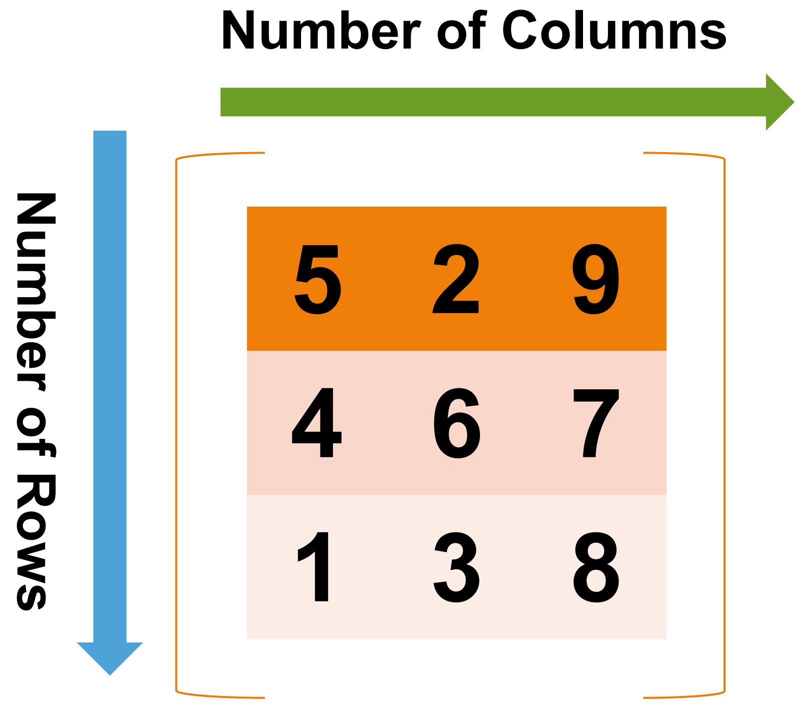
Agenda



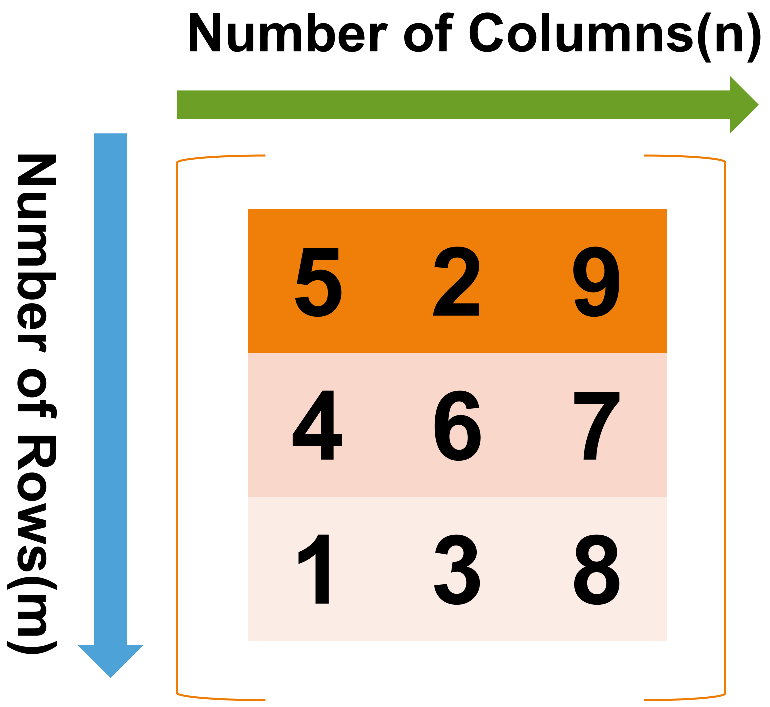
What is a Matrix?

What is a Matrix?

A Matrix is a combination of arrays, numbers or expressions that are represented in a rectangular format in the form of rows and columns.



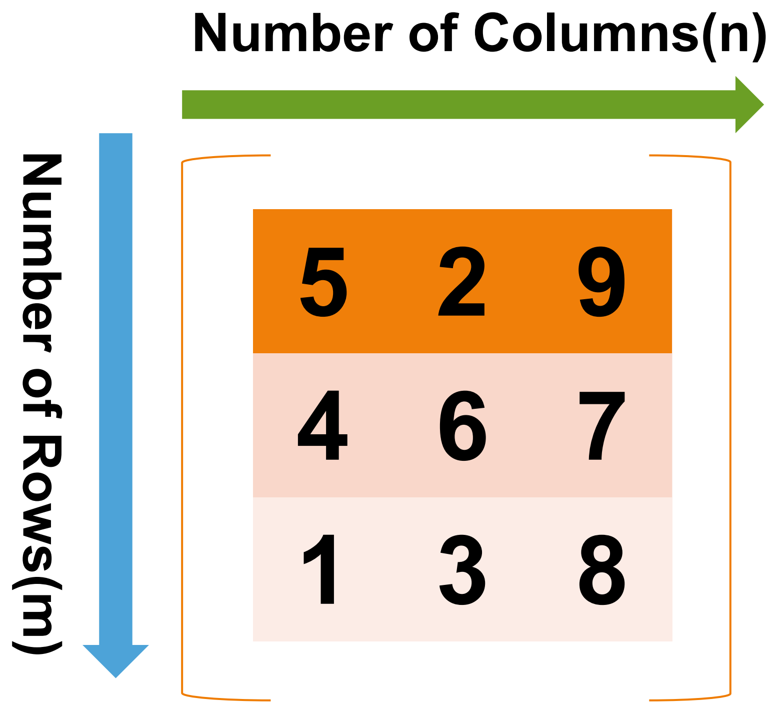
What is a Matrix?



$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix}$$

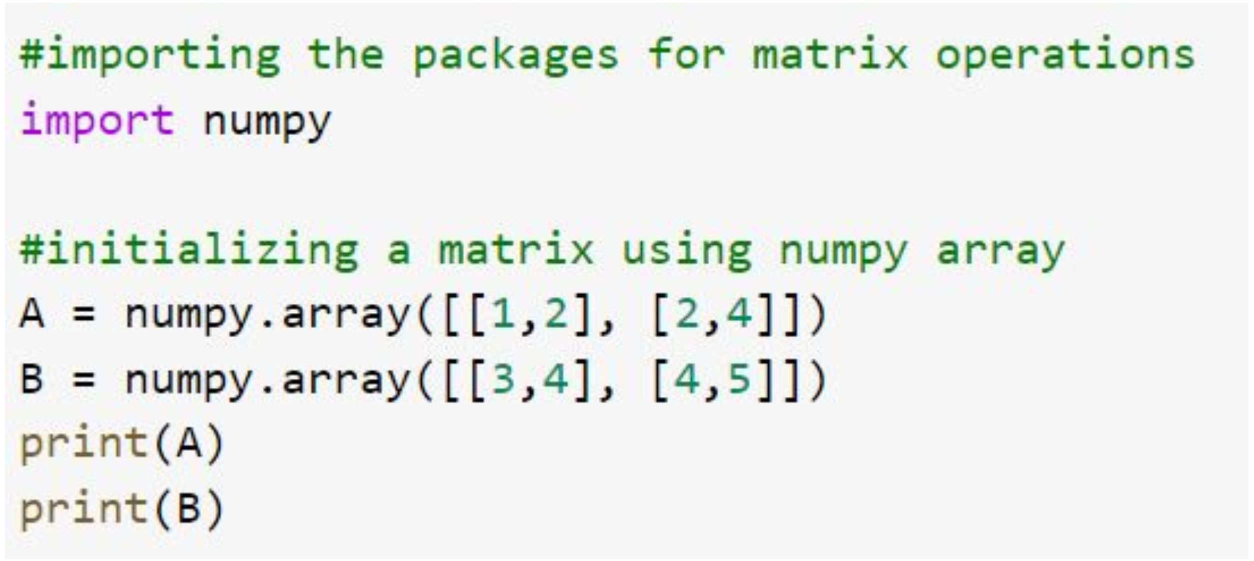
The rows and columns of a matrix are associated with the dimensions of a matrix and immensely help while working with linear expressions. The matrix with m rows and n columns will have the m x n dimensions.

What is a Matrix?



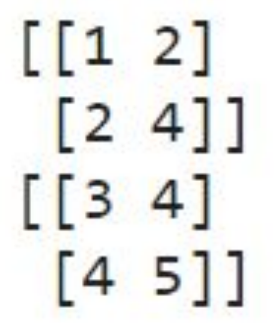
Let us go through some of the Matrix operations like addition, subtraction, multiplication, transpose, inverse of a matrix, etc.

Hands on - Python



```
#importing the packages for matrix operations
import numpy

#initializing a matrix using numpy array
A = numpy.array([[1,2], [2,4]])
B = numpy.array([[3,4], [4,5]])
print(A)
print(B)
```

Output



```
[[1 2]
 [2 4]]
[[3 4]
 [4 5]]
```

In Python, we use the list of lists in a numpy array to declare a list. As seen in the example we have declared A and B matrices with dimensions 2x2.

Matrix Operations

Addition

$$\begin{bmatrix} \boxed{5} & 2 & 2 \\ \boxed{4} & 6 & 7 \\ \boxed{1} & 3 & 2 \end{bmatrix} + \begin{bmatrix} \boxed{4} & 5 & 4 \\ \boxed{2} & 3 & 2 \\ \boxed{1} & 2 & 4 \end{bmatrix} = \begin{bmatrix} \boxed{9} & 7 & 6 \\ \boxed{6} & 9 & 9 \\ \boxed{2} & 5 & 6 \end{bmatrix}$$

In the addition operation, the corresponding values of the elements are added to the resultant matrix like shown in the example.

Addition

$$\begin{bmatrix} \boxed{5} & 2 & 2 \\ \boxed{4} & 6 & 7 \\ \boxed{1} & 3 & 2 \end{bmatrix} + \begin{bmatrix} \boxed{4} & 5 & 4 \\ \boxed{2} & 3 & 2 \\ \boxed{1} & 2 & 4 \end{bmatrix} = \begin{bmatrix} \boxed{9} & 7 & 6 \\ \boxed{6} & 9 & 9 \\ \boxed{2} & 5 & 6 \end{bmatrix}$$

In the addition operation, it is mandatory that the dimensions of the matrices are equivalent to each other.

Subtraction

$$\begin{bmatrix} \boxed{5} & 2 & 2 \\ \boxed{4} & 6 & 7 \\ \boxed{1} & 3 & 2 \end{bmatrix} - \begin{bmatrix} \boxed{4} & 5 & 4 \\ \boxed{2} & 3 & 2 \\ \boxed{1} & 2 & 4 \end{bmatrix} = \begin{bmatrix} \boxed{1} & -3 & -2 \\ \boxed{2} & 3 & 5 \\ \boxed{0} & 1 & -2 \end{bmatrix}$$

Similar to the addition operation, in subtraction we will subtract the corresponding element in both the matrices to get the resultant matrix.

Subtraction

$$\begin{bmatrix} \boxed{5} & 2 & 2 \\ \boxed{4} & 6 & 7 \\ \boxed{1} & 3 & 2 \end{bmatrix} - \begin{bmatrix} \boxed{4} & 5 & 4 \\ \boxed{2} & 3 & 2 \\ \boxed{1} & 2 & 4 \end{bmatrix} = \begin{bmatrix} \boxed{1} & -3 & -2 \\ \boxed{2} & 3 & 5 \\ \boxed{0} & 1 & -2 \end{bmatrix}$$

In the subtraction operation, it is mandatory for the matrices to be of equivalent dimensions as well.

Multiplication

$$\begin{bmatrix} \boxed{5} & 2 & 9 \\ \boxed{4} & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \boxed{2} = \begin{bmatrix} \boxed{10} & 4 & 18 \\ \boxed{8} & 12 & 14 \\ 2 & 6 & 16 \end{bmatrix}$$

Multiplying a matrix with a scalar will yield the resultant matrix that will have each element multiplied with the scalar.

Multiplication

$$\begin{bmatrix} \boxed{5} & 2 & 9 \\ \boxed{4} & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \boxed{2}$$

Scalar multiplication will not only have the same dimensions, but will show the properties like distributivity, associativity, additive inverse, etc.

Vector Multiplication

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix} =$$

While multiplying a matrix with a column vector, we must figure out the dimensions of the resultant matrix. For this, the rows of the first matrix and the columns of the second matrix will be the dimensions of the resultant matrix.

Vector Multiplication

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix} =$$

The dimensions of the first matrix is 3x3 and that of the second matrix is 3x1. Therefore, the dimensions of the resultant matrix will be 3x1.

Vector Multiplication

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

For finding the elements of the resultant matrix, we multiply the rows of the first matrix to the column of the second matrix.

Vector Multiplication

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 45 \\ 44 \\ 39 \end{bmatrix}$$

X = (5x1) + (2x2) + (9x4) = 5+4+36 = 45

Y = (4x1) + (6x2) + (7x4) = 4+12+28 = 44

Z = (1x1) + (3x2) + (8x4) = 1+6+32 = 39

Hands on - Python

```
C = numpy.array([[5,2,9],[4,6,7],[1,3,8]])
D = numpy.array([[1],[2],[4]])

result = numpy.dot(C,D)
print(result)
```

Output

```
[[45]
 [44]
 [39]]
```

In this example, we have used the numpy.dot() method to multiply the two matrices.

Matrix Multiplication

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \begin{pmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 2 \end{pmatrix} =$$

Similar to the vector multiplication, we will determine the dimension of the resultant matrix from these two matrices.

Matrix Multiplication

$$\begin{bmatrix}5 & 2 & 9\\4 & 6 & 7\\1 & 3 & 8\end{bmatrix} \times \begin{pmatrix}1 & 2\\2 & 1\\1 & 2\end{pmatrix} =$$

The dimension of the first matrix is 3x3 and that of the second matrix is 3x2. Therefore the dimension of the resultant matrix will be 3x2.

Matrix Multiplication

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} x1 & x2 \\ y1 & y2 \\ z1 & z2 \end{bmatrix}$$

Matrix Multiplication

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 2 \end{bmatrix}$$

X1 = (5x1) + (2x2) + (9x1) = 5+4+9 = 18

X2 = (5x2) + (2x1) + (9x2) = 10+2+18 = 30

X1 = (4x1) + (6x2) + (7x1) = 4+12+7 = 23

X2 = (4x2) + (6x1) + (7x2) = 8+6+14 = 28

X1 = (1x1) + (3x2) + (8x1) = 1+6+8 = 15

X2 = (1x2) + (3x1) + (8x2) = 2+3+16 = 21

Matrix Multiplication

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 18 & 30 \\ 23 & 28 \\ 15 & 21 \end{bmatrix}$$

There is a **limitation** with matrix multiplication that states that the **columns of the first matrix should always be equal to the rows of the second matrix**. For example – if the dimensions of the above matrices were 3x**3** and **2**x2, the multiplication would not have been possible.

Hands on - Python

```
C = numpy.array([[5,2,9],[4,6,7],[1,3,8]])
D = numpy.array([[1,2],[2,1],[1,2]])

result = numpy.dot(C,D)
print(result)
```

Output

```
[[18 30]
 [23 28]
 [15 21]]
```

In this example, we have used the numpy.dot() method to multiply the two matrices.

Matrix Multiplication

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} =$$

The dimension of the first matrix is 3x3 and that of the vector is 3x3. Therefore, the dimension of the resultant matrix will be 3x3.

Matrix Multiplication

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \\ 3 & 1 & 2 \\ 4 & 2 & 1 \end{bmatrix} =$$

The approach would be same here, we will multiply the rows of the first matrix to the column of the second matrix.

Cross Product

$$\begin{bmatrix} 3 \\ 2 \\ 8 \end{bmatrix} \times \begin{bmatrix} 2 \\ 9 \\ 7 \end{bmatrix} = \begin{bmatrix} i & j & k \\ 3 & 2 & 8 \\ 2 & 9 & 7 \end{bmatrix}$$

$$= i \begin{vmatrix} 2 & 8 \\ 9 & 7 \end{vmatrix} - j \begin{vmatrix} 3 & 8 \\ 2 & 7 \end{vmatrix} + k \begin{vmatrix} 3 & 2 \\ 2 & 9 \end{vmatrix}$$

Matrix Multiplication

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \\ 3 & 1 & 2 \\ 4 & 2 & 1 \end{bmatrix} = \begin{bmatrix} \boxed{47} & \boxed{30} & \\ & & \\ & & \end{bmatrix}$$

$\boxed{X1 = (5x1) + (2x3) + (9x4) = 5+6+36 = 47}$

$\boxed{X2 = (5x2) + (2x1) + (9x2) = 10+2+18 = 30}$

Matrix Multiplication

$$\begin{bmatrix} \boxed{5 \quad 2 \quad 9} \\ 4 \quad 6 \quad 7 \\ 1 \quad 3 \quad 8 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \\ 3 & 1 & 2 \\ 4 & 2 & 1 \end{bmatrix} = \begin{bmatrix} \boxed{47} & \boxed{30} & ? \\ ? & ? & ? \\ ? & ? & ? \end{bmatrix}$$

Similarly, you can find the rest of the elements of the resultant matrix.

Matrix Multiplication

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \\ 3 & 1 & 2 \\ 4 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 47 & 30 & 18 \\ 50 & 28 & 23 \\ 42 & 21 & 15 \end{bmatrix}$$

Hands on - Python

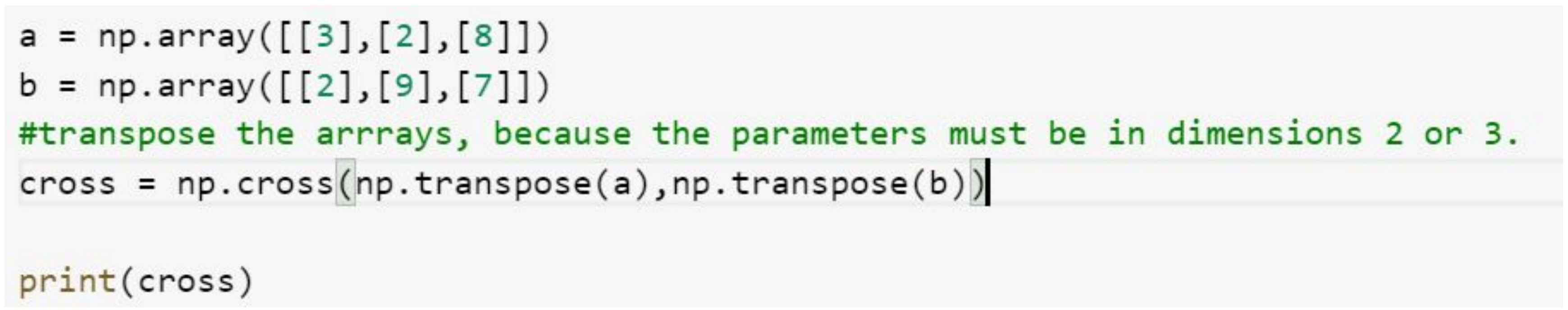
```python
C = numpy.array([[5,2,9],[4,6,7],[1,3,8]])
D = numpy.array([[1,2,1],[3,1,2],[4,2,1]])

result = numpy.dot(C,D)
print(result)
```

Output

```
[[47 30 18]
 [50 28 23]
 [42 21 15]]
```

In this example, we have used the numpy.dot() method to multiply the two matrices.

Hands on - Python



```
a = np.array([[3],[2],[8]])
b = np.array([[2],[9],[7]])
#transpose the arrays, because the parameters must be in dimensions 2 or 3.
cross = np.cross(np.transpose(a),np.transpose(b))

print(cross)
```

Output

```
[[-58  -5  23]]
```

In this example, we have used the numpy.cross() method to find the cross product of the two matrices.

Transpose

$$\begin{bmatrix} 1 & 1 & 0 \\ 2 & 1 & 2 \end{bmatrix}^T = \begin{bmatrix} 1 & 2 \\ 1 & 1 \\ 0 & 2 \end{bmatrix}$$

In the transpose of a matrix, the columns and rows are interchanged like shown in the example.

Properties of Transpose of a Matrix

Transpose of a Transposed matrix is the original matrix
$(A')' = A$

Additive property of transpose of a matrix
$(A+B)' = A' + B'$

Multiplication by a Constant
$K \times A' = KA'$, Where K is a constant

Multiplication property of the transpose:
$(AB)' = B'A'$

Hands on - Python

```
E = numpy.array([[1,1,0],[2,1,2]])
transpose = numpy.transpose(E)
print(transpose)
```

Output

```
[[1 2]
 [1 1]
 [0 2]]
```

In this example, we have used the numpy.transpose() method to obtain the transpose of the matrix E.

Inverse

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix}^{-1} =$$

To understand the inverse of a matrix, there are several concepts that you must be familiar with like determinant of a matrix, adjoint of a matrix, etc.

Inverse – Determinant of a Matrix

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} =$$

We can follow the laplacian expansion method to find the determinant of the matrix. It helps us in finding the inverse of the matrix.

Inverse – Determinant of a 2x2 Matrix

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \boxed{|Matrix| = a \times d - b \times c}$$

$$\begin{bmatrix} 1 & 4 \\ 6 & 9 \end{bmatrix} = \boxed{|Matrix| = 1 \times 9 - 4 \times 6 = 9 - 24 = -15}$$

Inverse – Determinant of a 3x3 Matrix

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} =$$

|Matrix| = a(e x i – f x h) – b(d x i – f x g) + c(d x h – e x g)

$$\begin{bmatrix} 2 & 2 & -3 \\ 1 & 2 & 2 \\ 1 & 1 & 2 \end{bmatrix}$$

|Matrix| = 2(2 x 2 – 2 x 1) – 2(1 x 2 – 2 x 1) – 3(1 x 1 – 2 x 1)

= 2(2) – 2(0) – 3(-1)
= 4 – 0 – (-3)
= 4 + 3 = 7

Inverse – Adjoint of a Matrix

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} =$$

To find the adjoint of a given matrix, we have to find the transpose of the cofactors of the elements in the given matrix. It is also known as adjugate of a matrix.

Inverse – Adjoint of a Matrix

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} = \begin{bmatrix} m11 & m12 & m13 \\ m21 & m22 & m23 \\ m31 & m32 & m33 \end{bmatrix}$$

The minor (m11, m12,… mij) are the determinants of the square matrix that is formed after eliminating the ith row and jth column of the matrix. We use the minors to calculate the cofactors of the given matrix.

Inverse – Adjoint of a Matrix

$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} = \begin{bmatrix} m11 & m12 & m13 \\ m21 & m22 & m23 \\ m31 & m32 & m33 \end{bmatrix}$$

Formulae to find the cofactors = $Cij = (-1)^{i+j}|mij|$

We will use the above formulae to calculate the cofactors of the given matrix.

Inverse – Adjoint of a Matrix

m11 = 48 – 21 = 27

m22 = 40 – 9 = 31

m12 = 32 – 7 = 25

m23 = 15 -2 = 13

m13 = 12 – 6 = 6

m31 = 14 – 54 = -40

m21 = 16 – 27 = -11

m32 = 35 – 36 = -1

m33 = 30 – 8 = 22

Inverse – Adjoint of a Matrix

$$\begin{bmatrix} 27 & -25 & 6 \\ 11 & 31 & -13 \\ -40 & 1 & 22 \end{bmatrix}^T = \begin{bmatrix} 27 & 11 & -40 \\ -25 & 31 & 1 \\ 6 & -13 & 22 \end{bmatrix}$$

Let’s take a look at the formulae to find the inverse of the matrix.

Inverse – Formulae

$$\begin{bmatrix} 2 & 1 \\ 4 & 3 \end{bmatrix}^{-1} =$$

The formulae to calculate the inverse of a matrix is as follows:

$$\mathbf{A^{-1} = 1/|A|. Adj\ A}$$

Inverse – Formulae

$$\begin{bmatrix} 2 & 1 \\ 4 & 3 \end{bmatrix}^{-1} =$$

Now, to calculate the inverse, we will find the determinant and adjoint of the given matrix.

Inverse – Formulae

Adjoint of the given matrix is $=$ $\begin{bmatrix} 3 & -1 \\ -4 & 2 \end{bmatrix}$

Determinant of the given matrix is $=$ => 2x3 – 4x1 = 2

Hands on - Python

```
F = numpy.array([[2,1],[4,3]])
inverse = numpy.linalg.inv(F)
print(inverse)

G = numpy.array([[1,1,2],[2,1,2],[2,3,1]])
inverse3x3 = numpy.linalg.inv(G)
print(inverse3x3)
```

Output

```
[[ 1.5 -0.5]
 [-2.   1. ]]
[[-1.   1.   0. ]
 [ 0.4 -0.6  0.4]
 [ 0.8 -0.2 -0.2]]
```

In this example, we have used the numpy.linalg.inv() method to calculate the inverse of a 2x2 and a 3x3 matrices F and G respectively.

Rank of a Matrix

$$\rho(A) \rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

The number of linearly independent columns in a matrix is known as the rank of the matrix. There are several ways we can use to calculate the rank of the matrix. One such method is to reduce the matrix to its row-echelon form.

Rank of a Matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

Things to Remember:

1. If the matrix A is of the order mxm, and if the determinant of |A| is not equal to zero, then the rank of the matrix will be m.
2. If the determinant is equal to zero, the rank will be less than m.

Rank of a Matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

Row Echelon Form

1. The pivots go from top-left to bottom right. Pivots are the first non-zero element in the row.
2. There must be a zero below the pivots in the row echelon form.
3. The rows will all zero elements must be at the end of the matrix.

Rank of a Matrix

$$\begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 4 \\ 2 & 4 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 2 \\ 0 & 1 & 4 \\ 0 & 0 & 0 \end{bmatrix}$$

We can use the gaussian elimination method where we interchange, multiply and add the rows to reach the row echelon form. Here, we have used the following operation to get the row echolen form.

1. R3 = R3 – 2R1

Since, the non-zero rows are 2, hence the rank of the matrix is **2**.

Hands on - Python

```
H = numpy.array([[1,2,2],[0,1,4],[2,4,4]])
rank = numpy.linalg.matrix_rank(H)
print("rank of the matrix H is: " + str(rank))
```

Output

```
rank of the matrix H is: 2
```

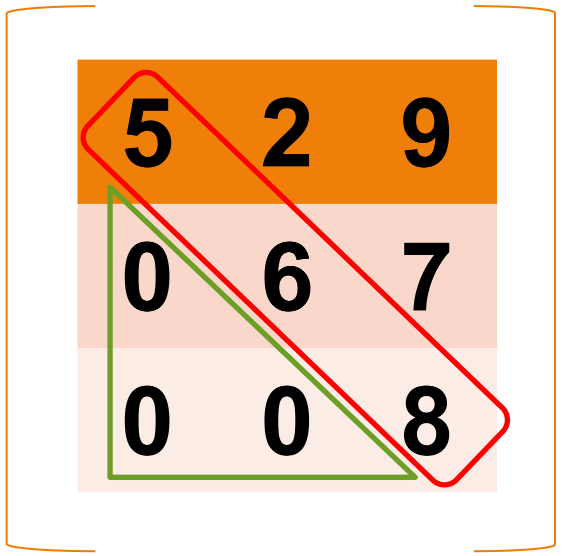
In this example, we have used the numpy.linalg.matrix_rank() method to calculate the rank of the given matrix.

Matrix Factorization

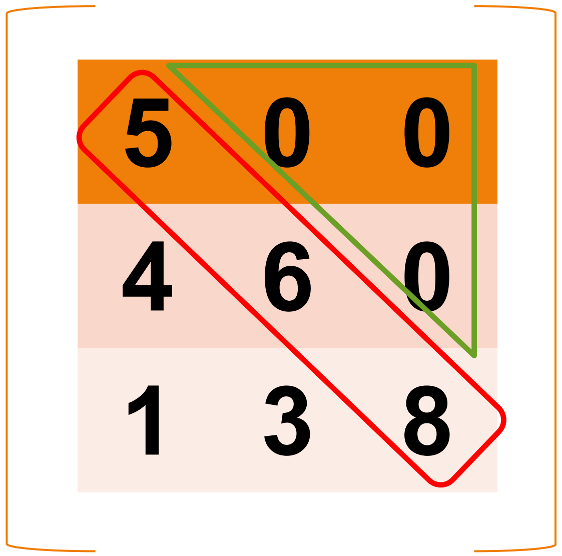
$$\begin{bmatrix} 5 & 2 & 9 \\ 4 & 6 & 7 \\ 1 & 3 & 8 \end{bmatrix} =$$

Matrix factorization, also known as decomposition is a process where a matrix is reduced to various components/factors that can be used for complex operations. There are various ways to decompose a matrix, let’s take a look at a few ways to perform matrix Factorization.

Matrix Decomposition



$$\begin{bmatrix} 5 & 2 & 9 \\ 0 & 6 & 7 \\ 0 & 0 & 8 \end{bmatrix}$$



$$\begin{bmatrix} 5 & 0 & 0 \\ 4 & 6 & 0 \\ 1 & 3 & 8 \end{bmatrix}$$

Upper Triangle Matrix

In the upper triangle matrix all the elements below the diagonal are zero.

Lower Triangle Matrix

In the lower triangle matrix, all the elements above the diagonal are zero.

LU Decomposition

LU decomposition is the method in which the matrix A is decomposed in the following manner:

A = L.U

Where L is the lower triangle matrix and U is an Upper triangle matrix.

$$\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix} = \begin{bmatrix} L11 & 0 & 0 \\ L21 & L22 & 0 \\ L31 & L32 & L33 \end{bmatrix} \times \begin{bmatrix} U11 & U12 & U13 \\ 0 & U22 & U23 \\ 0 & 0 & U33 \end{bmatrix}$$

Hands on - Python

```
from scipy.linalg import lu
I = numpy.array([[1,2,1],[3,4,1],[1,3,2]])
P,L,U = lu(I)
print(P)
print(L)
print(U)
result = P.dot(L).dot(U)
print(result)
```

Output

```
[[0. 0. 1.]
 [1. 0. 0.]
 [0. 1. 0.]]
[[1.         0.         0.        ]
 [0.33333333 1.         0.        ]
 [0.33333333 0.4        1.        ]]
[[3.         4.         1.        ]
 [0.         1.66666667 1.66666667]
 [0.         0.         0.        ]]
[[1. 2. 1.]
 [3. 4. 1.]
 [1. 3. 2.]]
```

In this example, we have used the **lu** module from **scipy.linalg.** Where P is for partial pivoting for stable calculations.

QR Decomposition

QR decomposition deals with the matrices that have dimensions in the form of **mxn**. QR decomposition is the method in which the matrix A is decomposed in the following manner:

A = Q.R

Where Q is a matrix with dimensions mxm and R is an upper triangle matrix with dimensions mxn.

$$\begin{bmatrix} a11 & a21 \\ a21 & a22 \\ a31 & a32 \end{bmatrix} = \begin{bmatrix} Q11 & Q12 & Q13 \\ Q21 & Q22 & Q23 \\ Q31 & Q32 & Q33 \end{bmatrix} \times \begin{bmatrix} r11 & r12 \\ 0 & r22 \\ 0 & 0 \end{bmatrix}$$

Hands on - Python

```
from numpy.linalg import qr

J = numpy.array([[1,2],[4,1],[5,1]])
Q,R = qr(J)
print(Q)
print(R)
result = Q.dot(R)
print(result)
```

Output

```
[[-0.15430335  0.98415288]
 [-0.6172134  -0.02696309]
 [-0.77151675 -0.1752601 ]]
[[-6.4807407  -1.69733685]
 [ 0.          1.76608256]]
[[1. 2.]
 [4. 1.]
 [5. 1.]]
```

In this example, we have used the **qr** module from **numpy.linalg.** Where there is another parameter **mode,** which is optional.

Cholesky Decomposition

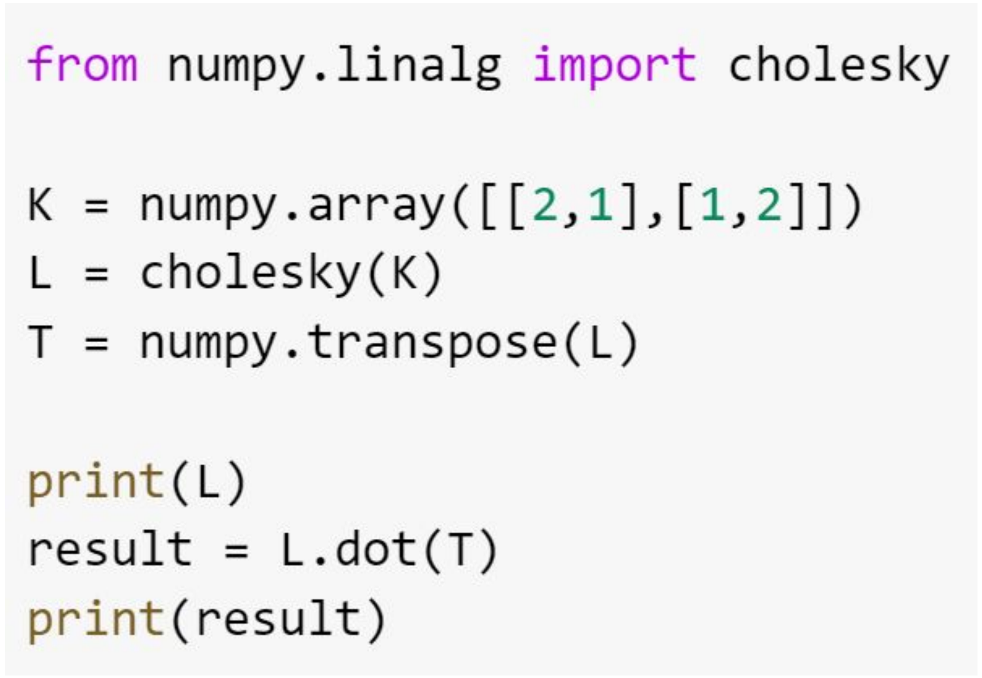
Cholesky decomposition method deals with the square symmetrical matrices that have positive Eigen values. The matrix A is decomposed in the following manner:

$$A = L.A^T$$

Where L is the lower triangle matrix and is multiplied with the transpose of the matrix A.

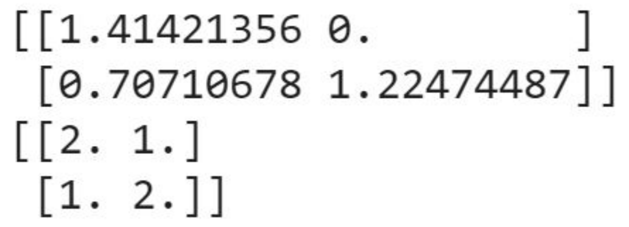
$$\begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix} = \begin{bmatrix} L11 & 0 & 0 \\ L21 & L22 & 0 \\ L31 & L32 & L33 \end{bmatrix} \times \begin{bmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{bmatrix}^T$$

Hands on - Python



```python
from numpy.linalg import cholesky

K = numpy.array([[2,1],[1,2]])
L = cholesky(K)
T = numpy.transpose(L)

print(L)
result = L.dot(T)
print(result)
```

Output



```
[[1.41421356 0.        ]
 [0.70710678 1.22474487]]
[[2. 1.]
 [1. 2.]]
```

In this example, we have used the cholesky module from numpy.linalg. Also, we have to make sure the matrix is positive definite matrix in order to get the efficient results.

Eigen Values And Eigen Vectors

Eigen Value

Eigen values are the roots of the characteristic polynomial of the matrix A. Let’s understand this with a simple example.

$$\begin{bmatrix} 2 & 1 \\ 4 & 3 \end{bmatrix}$$

To calculate the Eigen values of the given matrix, we will first compute the characteristic polynomial and then derive the Eigen values by finding the roots of the Characteristic polynomial.

Eigen Value

$$\begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix}$$

To find the characteristic polynomial, we will calculate the determinant of $(A - \lambda I_n)$.

The computation will get the resultant characteristic polynomial as:

$$f(\lambda) = \lambda^2 - 6\lambda + 1$$

Now, to find the eigen values, we will calculate the roots of the characteristic polynomial.

Eigen Values

$$\begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix}$$

$$f(\lambda) = \lambda^2 - 6\lambda + 1$$

$$\rightarrow \lambda^2 - 6\lambda + 1 = 0$$

Using the quadratic formula, we can get the roots of the equation as **$3 + 2\sqrt{2}$ and $3 - 2\sqrt{2}$**.

Therefore, the Eigen values for the given matrix are **$3 + 2\sqrt{2}$ and $3 - 2\sqrt{2}$**

Eigen Vectors

$$\begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix}$$

Now to calculate the Eigen vectors, we will do the following:

1. We will use the equation, **Av = λv** where v is the eigen vector.

$$\begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix} \begin{matrix} X \\ Y \end{matrix} = \lambda \cdot \begin{bmatrix} X \\ Y \end{bmatrix}$$

Eigen Vectors

$$\begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix} = \lambda \cdot \begin{bmatrix} X \\ Y \end{bmatrix}$$

Using the equations that we get by solving the above equation for both the values of **λ, we will get the Eigen Vectors of the given matrix.**

Hands on - Python

```
M = numpy.array([[5,2],[2,1]])
Eigen_value, Eigen_vector = numpy.linalg.eig(M)
print(Eigen_value)
print(Eigen_vector)
```

Output

```
[5.82842712 0.17157288]
[[ 0.92387953 -0.38268343]
 [ 0.38268343  0.92387953]]
```

In this example, we have used the numpy.linalg.eig() method to calculate the eigen values and eigen vectors of the given matrix M.

Correlation Matrix

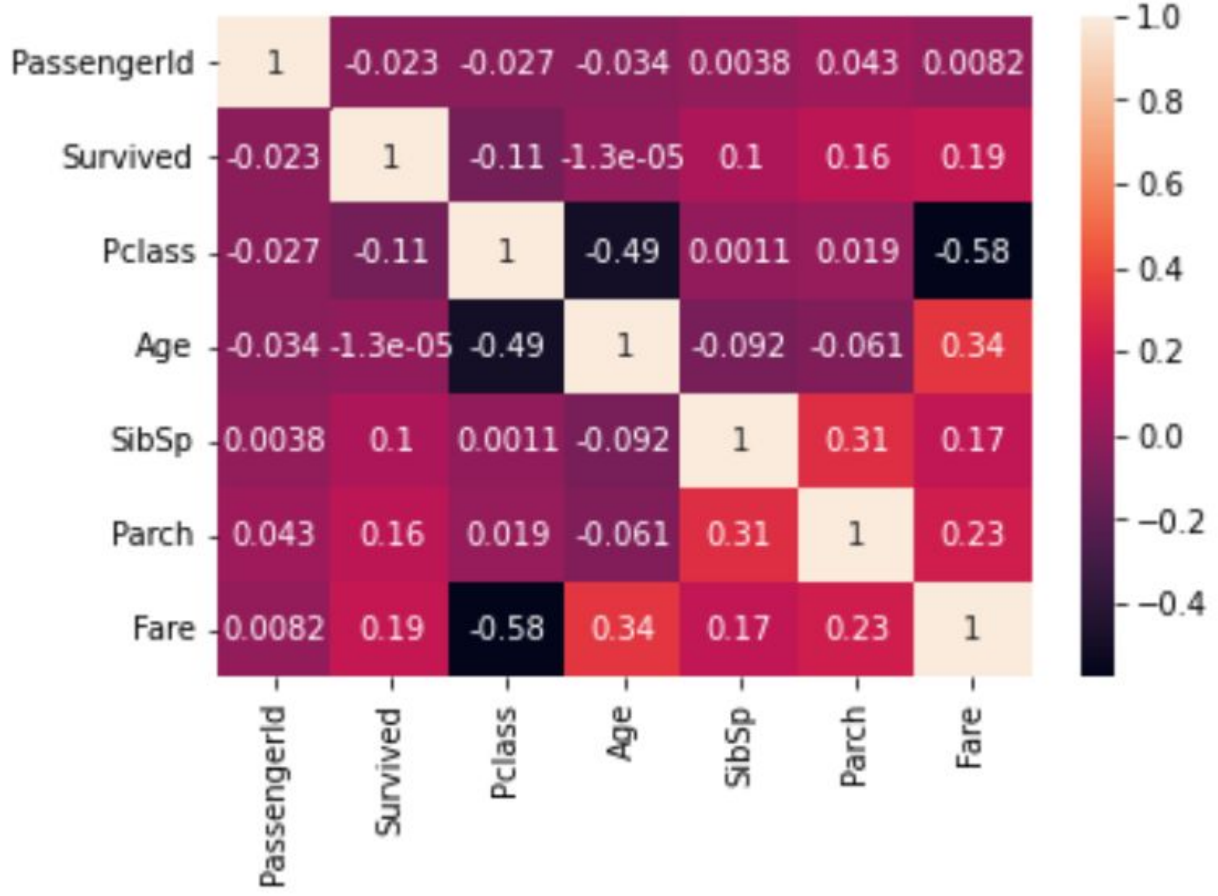
Collinearity is a concept in statistics that resonates with the linear relationship between the predictor variables(independent variable) in a regression model. If the independent variables are correlated, they won’t be able to independently predict the value of the dependent variable in the regression model.

A correlation matrix is a table that has the correlated coefficients between the variables. It is mostly used to deduce the patterns in a large dataset.

Hands on - Python

```python
import pandas as pd
import seaborn as sns
data = pd.read_csv("titanic.csv")
sns.heatmap(data.corr(), annot=True)
```

In this example, we have plotted the correlation coefficients of the titanic data on a heatmap using seaborn in python. By the value of the correlation coefficient, we can figure out the collinearity of the predictor variables.



India: +91-7847955955

US: 1-800-216-8930 (TOLL FREE)

support@intellipaat.com

24/7 Chat with Our Course Advisor