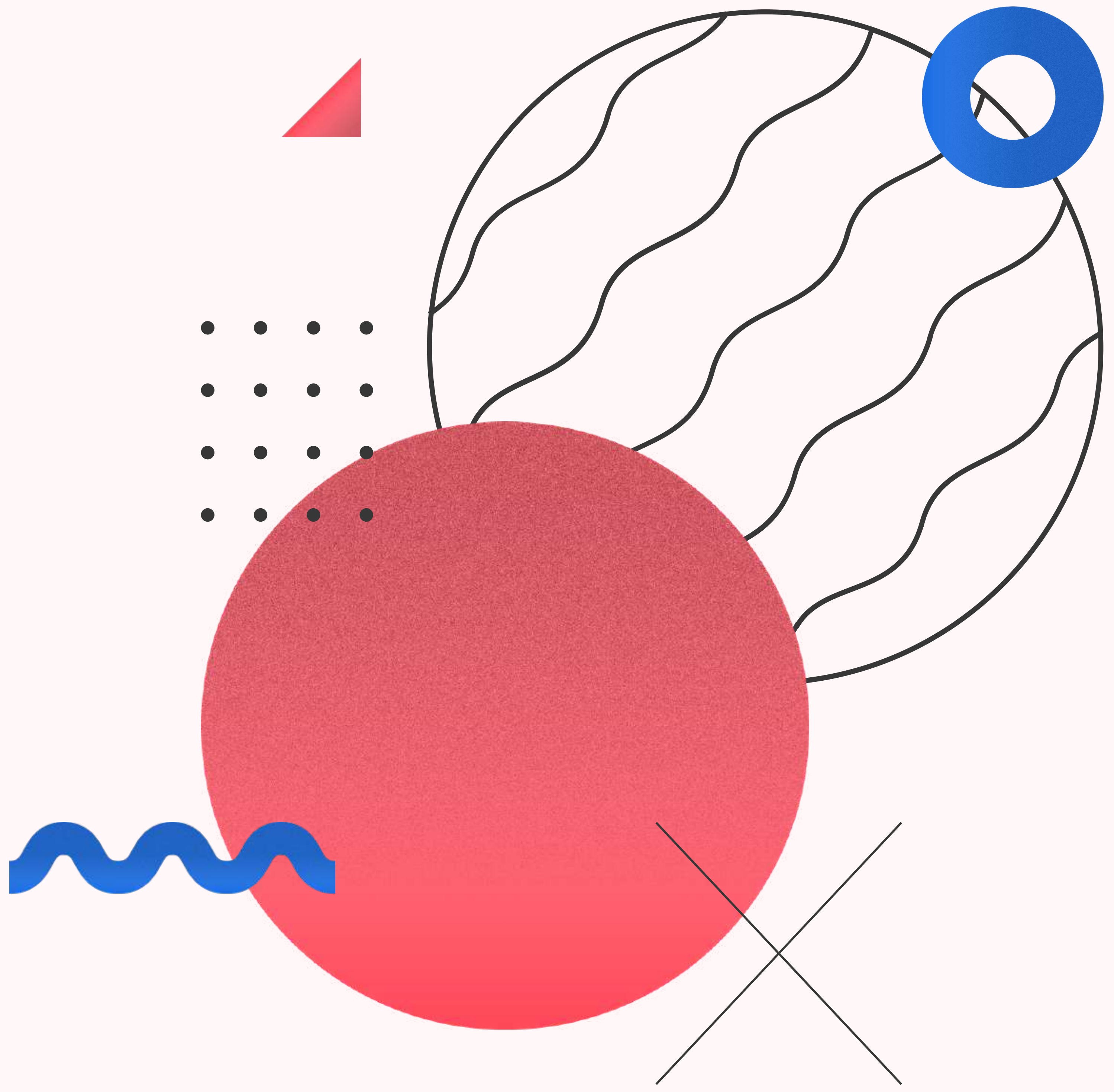
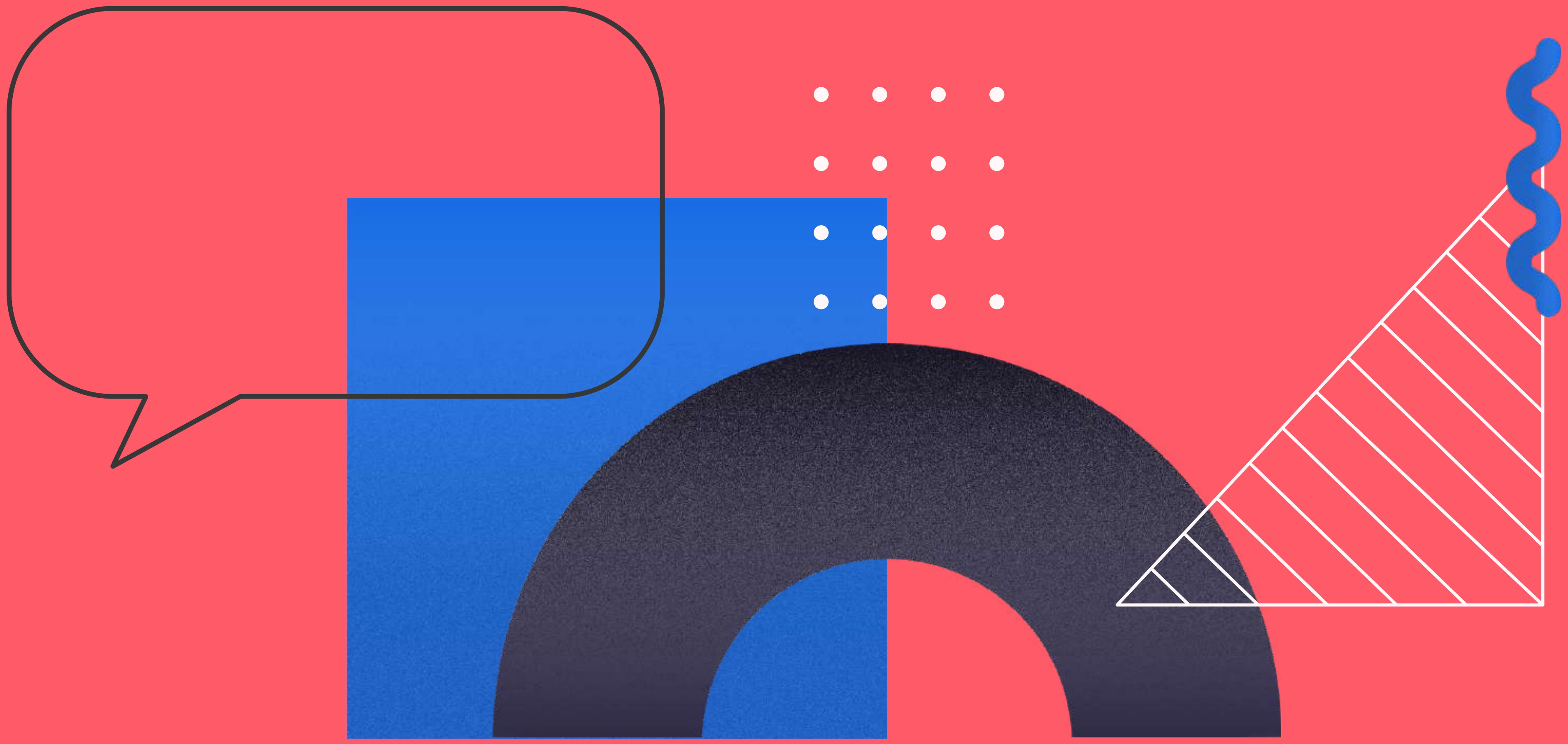
Data Preprocessing & Cleaning





Data Preprocessing is a data mining technique which is used to transform the raw data in a useful and efficient format.

Steps Involved:

Step 1: Analysis of Data:

Realizing the size and nature of data. Finding out the Target(or Dependent) Variable And the Independent Variables.

Step 2: Removal of Missing and Noise Data

Usage of drop, dropna, and fillna methods

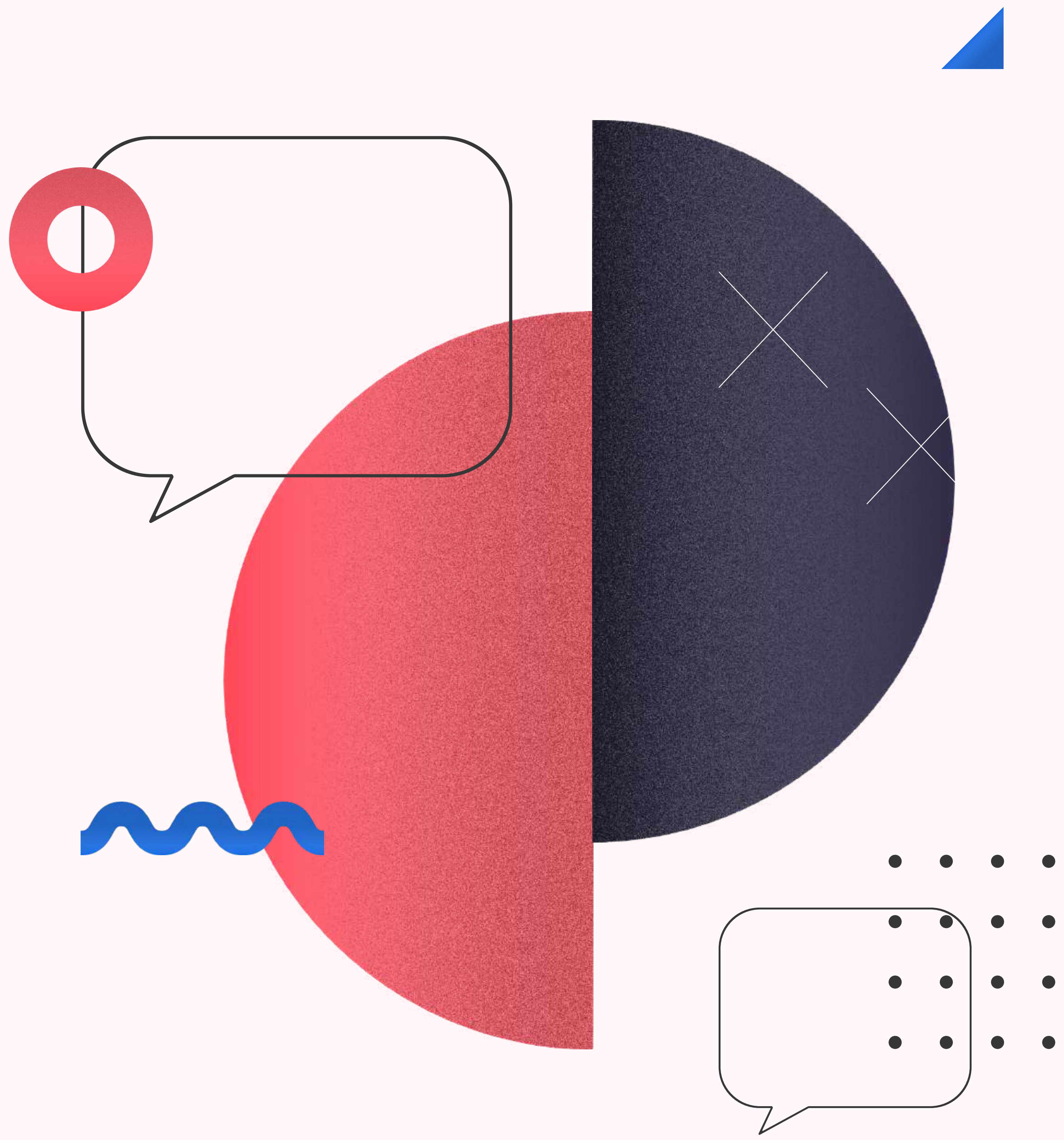
Step 3: Converting the data into a more efficient form for our model.

Encoding the categorical values, etc.

Numpy and Pandas

NumPy is a python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely

Pandas is a high-level data manipulation tool developed by Wes McKinney. It is built on the Numpy package and its key data structure is called the DataFrame. DataFrames allow you to store and manipulate tabular data in rows of observations and columns of variables



Numpy Arrays and Related Methods and Attributes

Numpy provides a data structure known as array.

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers

We can call the numpy package to our program as follows.

Syntax:

```
import numpy as np
```

Types of Numpy Arrays

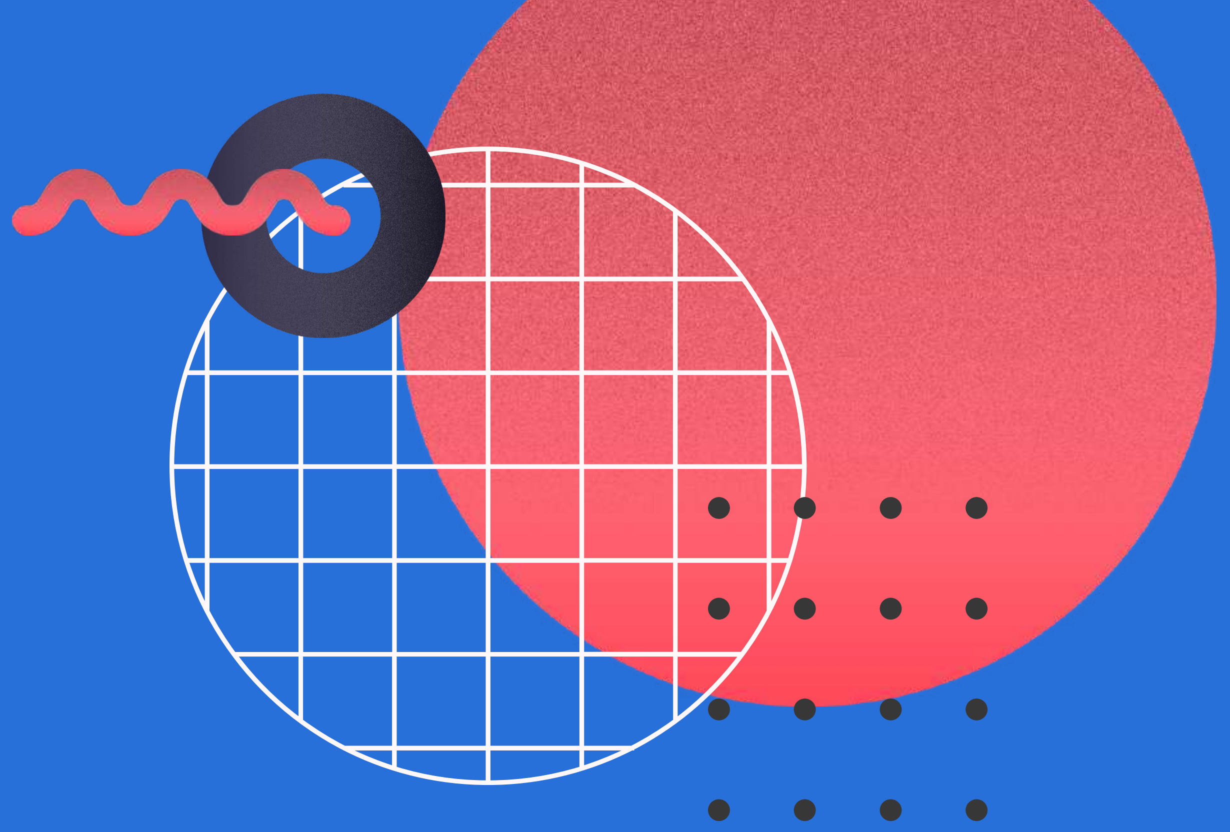
a.) VECTORS

Vectors are the single dimensional or 1D Arrays

b.) MATRICES

Matrices are the two dimensional or 2D Arrays.

Creating An Array



1.) array()

Passing a list to the above method converts it into a numpy array.

Syntax:

np.array(<list>)

2.) arange()

This function returns the array of number starting from a lower bound to an upper bound (excluding the upper bound) in steps specified by the user.

Syntax:

np.arange(<start>, <stop>, <steps>)

3.) linspace()

This function returns n equally space number between two given numbers.

Syntax:

np.linspace(<num1>,<num2>, n)

Continued....

4.) random.rand()

Returns a random value or an array of random values in range [0,1).

Syntax:

Vectors: np.random.rand(n)
Matrix: np.random.rand(a,b)

5.) random.randn()

Returns a value or an array of values from standard normal distribution.

Syntax:

Vectors: np.random.randn(n)
Matrix: np.random.randn(a,b)

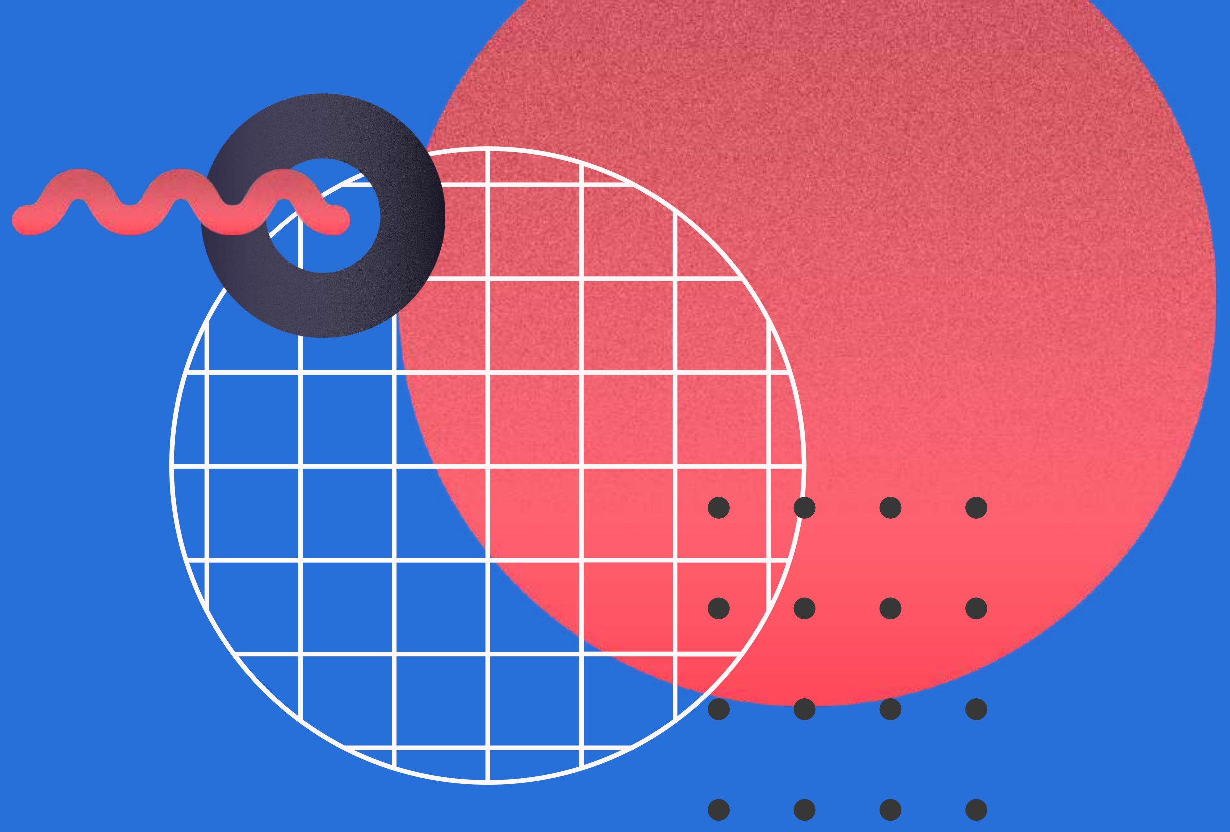
6.) random.randint()

Returns an integer or array of n integers between a lower bound and upper bound(exclusive of upper).

Syntax:

np.random.randint(<low>, <high>, n)

continued....



7.) zeros()

Returns a vector or a matrix of zeros only.

Syntax:

Vector: np.zeros(n)
Matrix: np.zeros((a,b))

8.) ones()

Returns a vector or a matrix of ones only.

Syntax:

Vector: np.ones(n)
Matrix: np.ones((a,b))

9.) eye()

Returns an identity matrix of order n.

Syntax:

np.eye(n)

shape attribute and reshape method

shape attribute returns the shape(or dimension) of an array.

Syntax:

```
<arr>.shape
```

reshape method helps us to change the shape, i.e., the dimension of an array. We can convert a matrix into a vector and vice versa using the attribute.

Syntax:

```
<arr>.reshape(<dimension>)
```

max, min, argmax, argmin

max() and min() are the methods which are used to find the maximum and minimum elements of an array.

Syntax:

```
<arr>.max(), <arr>.min()
```

argmax() and argmin() returns the index value of the maximum and minimum elements of an array.

Syntax:

```
<arr>.argmax(), <arr>.argmin()
```

Indexing and selection

To catch an element in array we do the following.

Syntax:

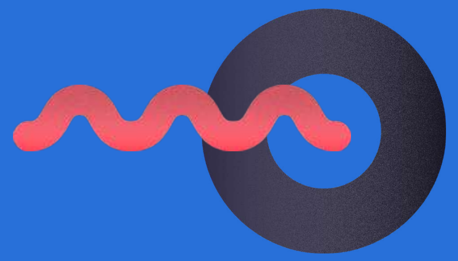
Vectors: arr[a]
Matrix: arr[i][j] or arr[i, j]

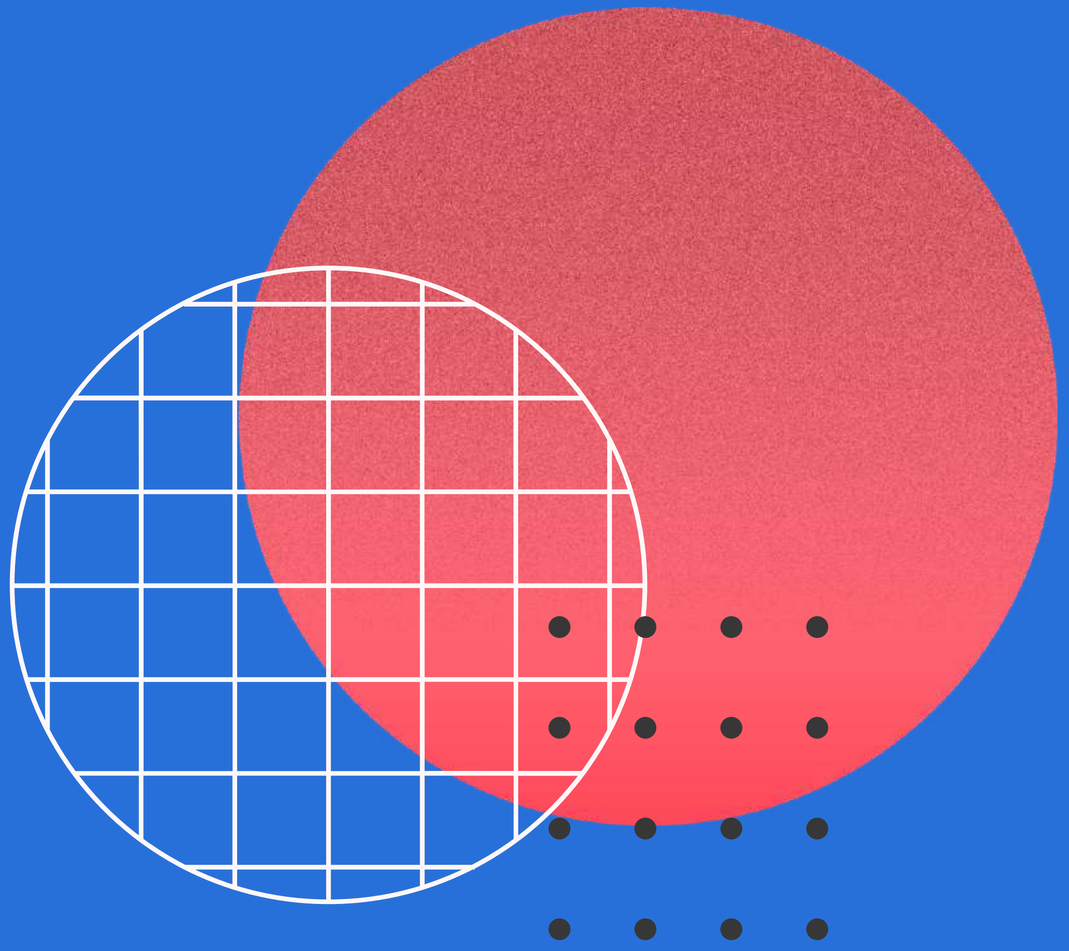
To select a part of an array we do the following.

Syntax:

arr[a:b] where a is inclusve and b is exclusive.

Note:
arr[: b] means all the elements from the start till b(exclusive)
arr[a :] means all the elements from a till the end
arr[:] means an entire array





copy()

copy() method is used to make a copy of array.

Syntax:

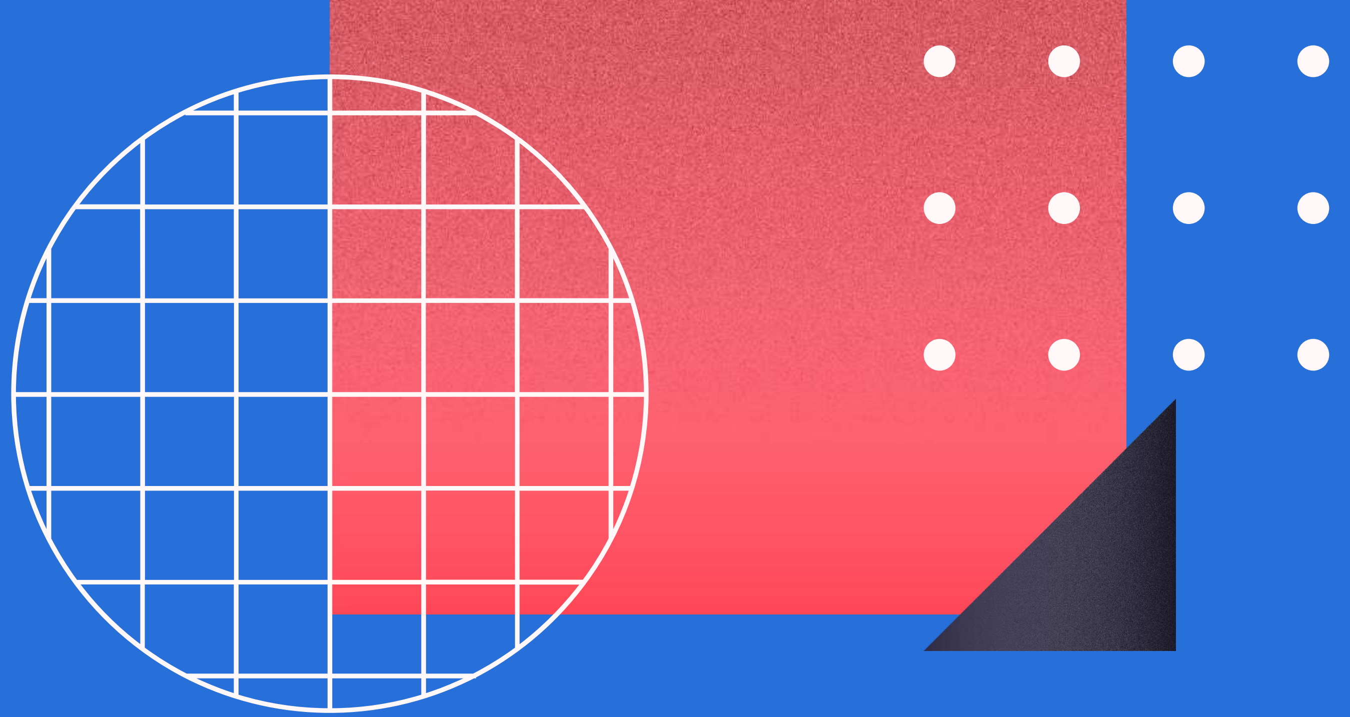
```
arr.copy()
```

Note:

We don't do arr1 = arr2 with numpy arrays as arr1 becomes the reference to the original array and any change made to arr1 will be reflected in arr2 too.

```
arr + n
arr - n
arr * n
arr / n
```

All the above statements will perform the corresponding arithmetic operation on each oh the array elements with n.

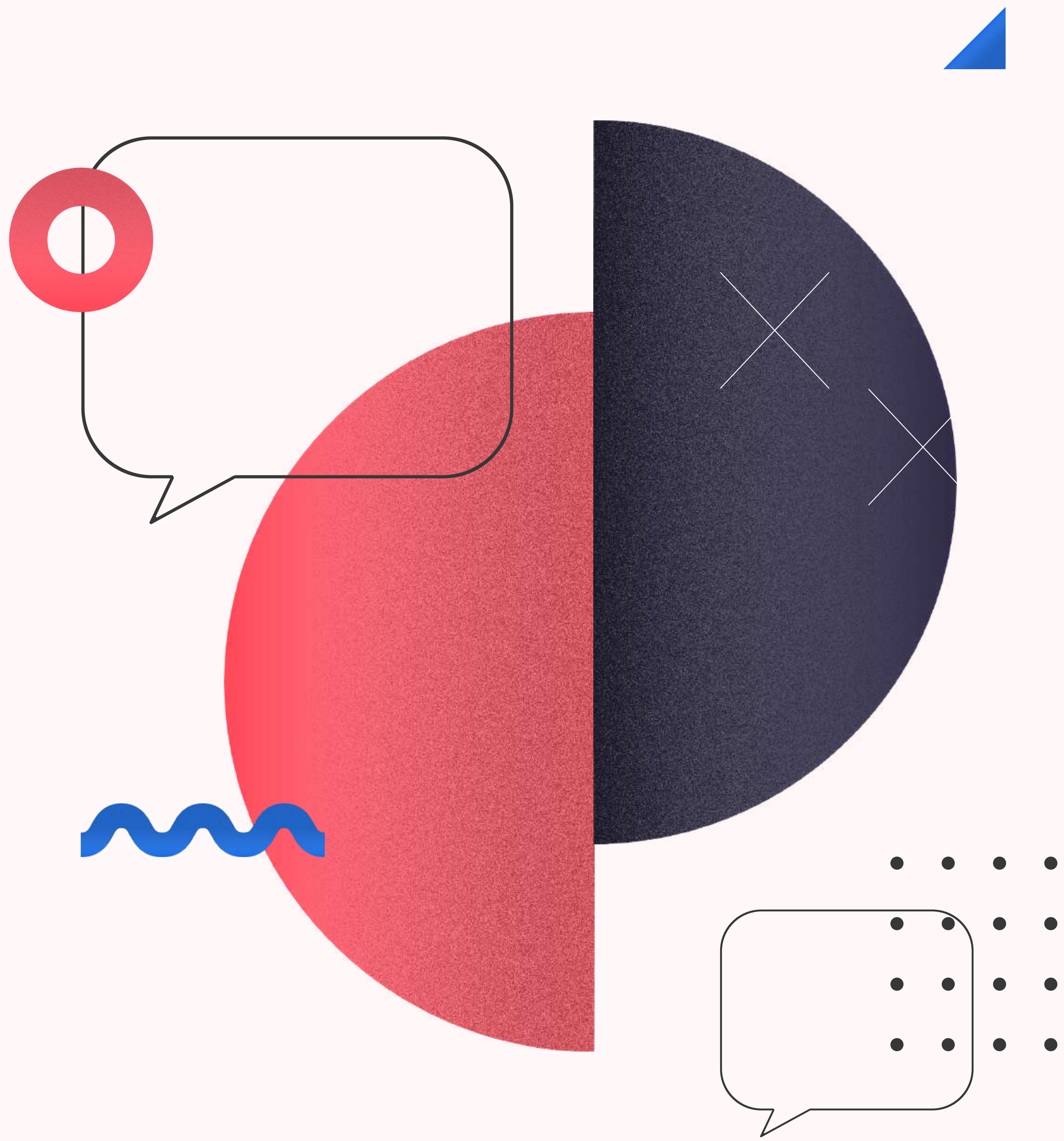


```
arr1 + arr2
arr1 - arr2
arr1 * arr2
arr1 / arr2
```

All the above statements perform the operation on each element of the first array with the corresponding elements of the second array.

Note:

Both the arrays must be of the same dimension.



Pandas Series and Dataframes and their Methods and Attributes

Pandas Series

Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called index.

We can import the package as follows.

Syntax:

```
import pandas as pd
```

Creating a pandas series

Syntax:

```
pd.Series(<data>, <labels>)
```

Data can be a list, numpy array or a dictionary.
Labels is a list of index names.
Labels are not required when we pass a dictionary as dictionary keys are considered as labels and values as data.

Indexing of a Series

Indexing is pretty much the same as in arrays or list. Only difference here is that the index can be an integer, a floating point number, a character, a string or a python object.

Syntax:

```
ser[<index>]
```

ser1 + ser2

ser1 * ser2

ser1 - ser2

ser1 / ser2

All the above statements performs the operation on elements of both the series with same indices and the rest are set to null.

Note:

Unlike arrays, the length of the two series may or may not be equal

Pandas Dataframe

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).

Creating A Dataframe

A dataframe is created as follows.

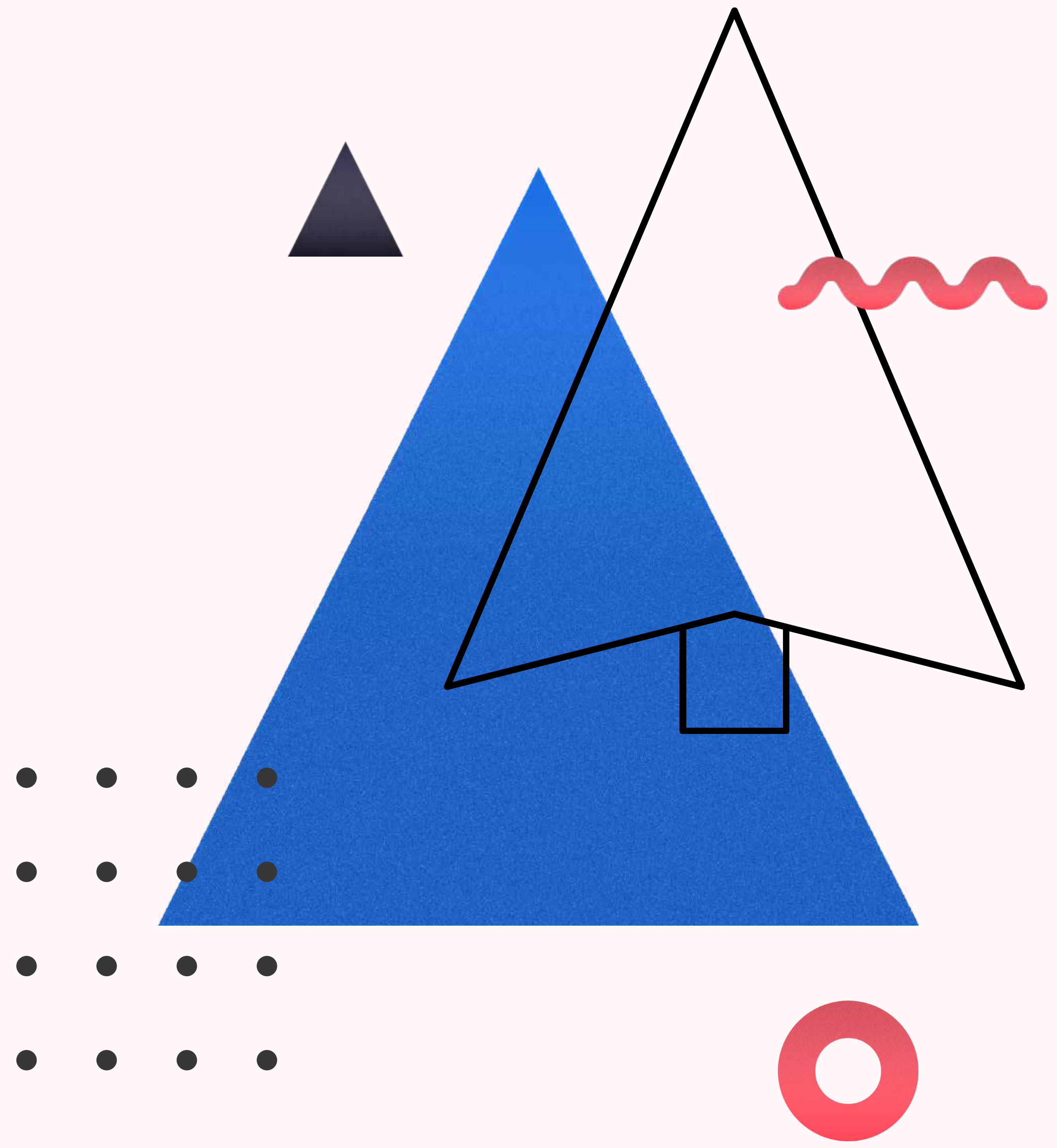
Syntax:

```
pd.DataFrame(<matrix>, <index>, <columns>)
```

```
df[<col1>] + df[<col2>]
df[<col1>] - df[<col2>]
df[<col1>] * df[<col2>]
df[<col1>] / df[<col2>]
```

all the above statements performs the operation on each element of the column one of the dataframe with each element of the second column of the dataframe.

Note: Each row of dataframe can be considered as a series.



head() and tail()

The head() method is used to print the top 5 rows of the dataframe, unless explicitly mentioned while calling the method iself. Similarly the tail() method prints 5 rows(by default) from the bottom.

Syntax:

```
<dataframe>.head(<number of rows>)

<dataframe>.tail(<number of rows>)
```

shape, index and columns attributes

shape attribute returns the shape of the data frame in the form (<rows>, <columns>)

index attribute returns a list of all the index values in the dataframe

columns attribute returns a list of colmun names in the dataframe

syntax: -

<df>.shape , <df>.index, <df>.columns

NOTE: These are attributes and not methods. While calling these we don't use parenthesis.

set_index(by, inplace = False)

sets one of the columns of the dataframe as the index of the dataframe. The parameter 'by' takes the column name and the parameter inplace takes a boolean value (false by default), which if true, makes the change in the existing dataframe itself

Syntax:

```
<df>.set_index(<column>, <inplace>)
```

sort_index(), sort_values()

sort_index() is used to sort the dataframe according to the index while sort_values() sort the dataframe according to the values of a column or a row. Both takes 'axis' (0 for row and 1 for column), 'ascending'(true or false) and 'inplace'(true or false) as parameters.

Syntax:

```
<df>.sort_index(<axis>,
<ascending>, <inplace>)

<df>.sort_values(by, <axis>,
<ascending>, <inplace>)
```

loc[] and iloc[]

To access columns and rows of a dataframe we make use of these two functions. loc[] makes use of the labels to access data while iloc[] makes use of their index values.

Syntax:

```
 <df>.loc[row, column]
<df>.iloc[row, column]
```

max(), min(), idxmax(), idxmin().

max() and min() returns the maximum and minimum value.

Syntax:

```
<df>.max()

<df>.min()
```

idxmax() and idxmin() returns the index value where the maximum or minimum value found.

Syntax:

```
<df>.idxmax()

<df>.idxmin()
```

value_counts(), isnull().

value_counts() counts how many times each unique value in a column occurs.

isnull() checks if the value is null and returns a boolean value.

Syntax:

```
<df>.value_counts()
<df>.isnull()
```

unique() and nunique()

unique() returns an array of all the unique values in a column of the dataframe.

Syntax:

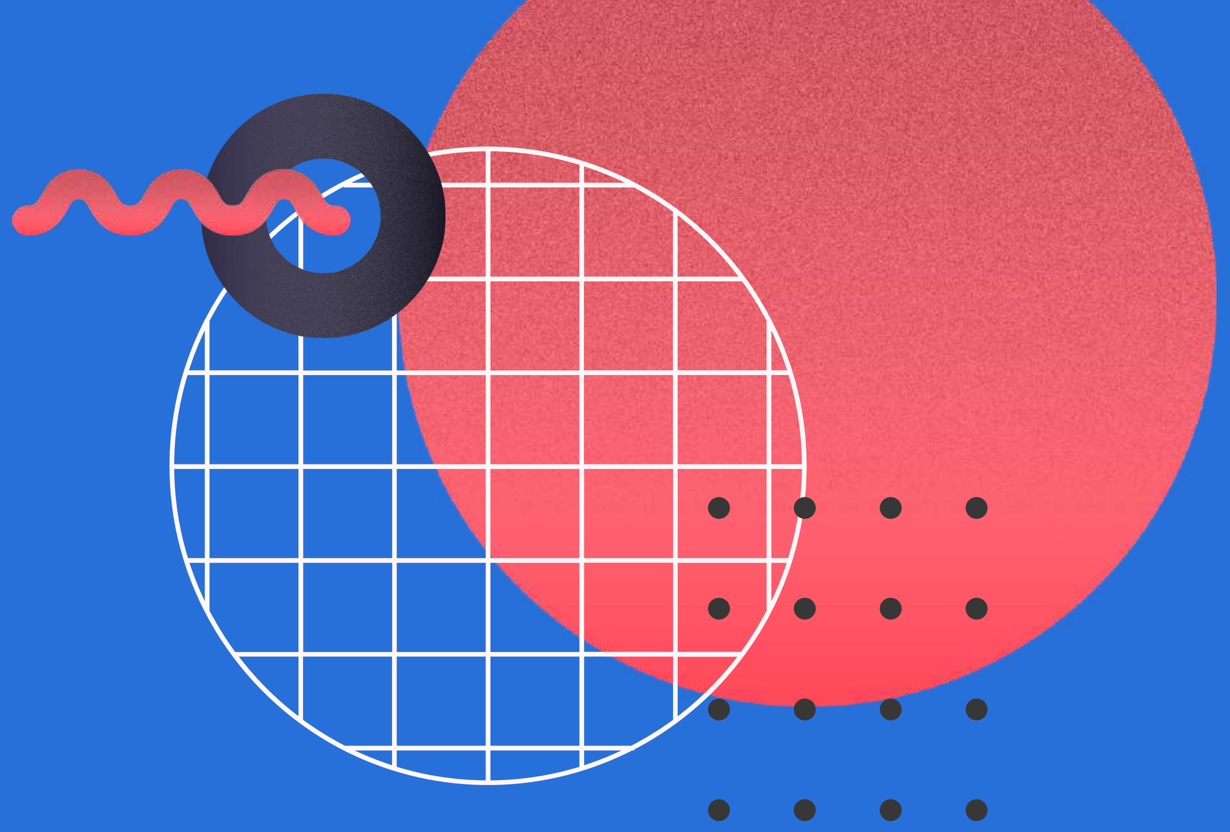
```
df[<column>].unique()
```

nunique() returns the number of unique values in a column of the dataframe.

Syntax:

```
df[<column>].nunique()
```

drop(), dropna(),fillna()



drop() is used to drop a row or a column from the dataframe. It takes paramters 'axis' and 'inplace'. 'axis' defines wether the label that we are dropping is a row or a column. It takes 0 for row and 1 for column.

Syntax:

df.drop(<columns>,<axis>, <inplace>)

dropna() drops all the rows or columns if there exists if one null value, unless specified otherwise. It takes all the parameters same as drop and one other parameter called 'thresh' which takes an integer value which defines the minimum number of non null values for the row or column to not be dropped.

Syntax:

df.dropna(<inplace>, <thresh>)

fillna() is used to replace the null value with a specified value

Syntax:

df[<column>].fillna(<value>, <inplace>)

Encoding:

When we have categorical values we encode them for the computer to understand better. We create dummy variables for each category and if a data falls into one of those, the corresponding dummy variables is set to one and the rest to 0.These dummy variables replace the original columns in the dataframe.

Note: We drop one of the dummy variables as they are higly corelated and one can be determined by the rest.

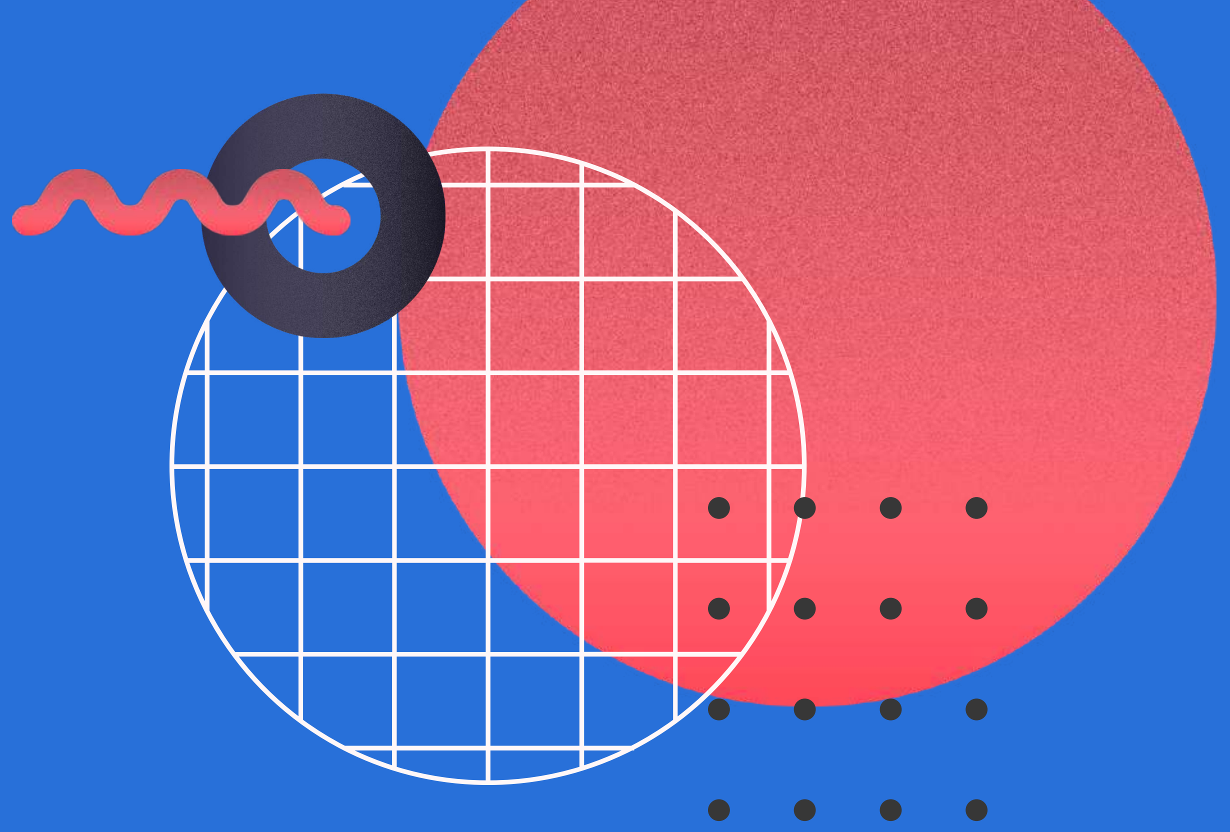
get_dummies().

Pandas provides a method to get the dummy variables called the get_dummies(). It takes the column as parameter. It also takes a parameter drop_first which takes boolean value(false by default) as input, which if true drops the first dummy column.

Syntax:

pandas.get_dummies(<column>, <drop_first>)

concat(), merge(), join()



concat() takes a list of dataframes and joins them into one either row-wise (axis = 0) or column-wise (axis = 1)

Syntax:

```
pd.concat(<list of df>, <axis>)
```

merge() takes two dataframes and merges them over a common (passed as 'on' parameter) column

Syntax:

```
pd.merge(<df1>, <df2>, <on>)
```

join() performs the same merge function but also converts the common column as the index of the new dataframe.

Syntax:

```
pd.join(<df1>, <df2>)
```

Thank You!