Machine Learning Specific:

Machine learning is the process of making systems that learn and improve by themselves, by being specifically programmed.

The ultimate goal of machine learning is to design algorithms that automatically help a system gather data and use that data to learn more. Systems are expected to look for patterns in the data collected and use them to make vital decisions for themselves.

In general, machine learning is getting systems to think and act like humans, show human-like intelligence, and give them a brain. In the real world, there are existing machine learning models capable of tasks like:

- Separating spam from actual emails, as seen in Gmail
- Correcting grammar and spelling mistakes, as seen in autocorrect

Thanks to machine learning, the world has also seen design systems capable of exhibiting uncanny human-like thinking, which performs tasks like:

- Object and image recognition
- Detecting fake news
- Understanding written or spoken words
- Bots on websites that interact with humans, like humans
- Self-driven cars

Figure 1: Machine learning

Machine Learning Steps

1. Collecting Data:

As you know, machines initially learn from the data that you give them. It is of the utmost importance to collect reliable data so that your machine learning model can find the correct patterns. The quality of the data that you feed to the machine will determine how accurate your model is. If you have incorrect or outdated data, you will have wrong outcomes or predictions which are not relevant.

Make sure you use data from a reliable source, as it will directly affect the outcome of your model. Good data is relevant, contains very few missing and repeated values, and has a good representation of the various subcategories/classes present.

Figure 2: Collecting Data

2. Preparing the Data:

After you have your data, you have to prepare it. You can do this by :

- Putting together all the data you have and randomizing it. This helps make sure that data is evenly distributed, and the ordering does not affect the learning process.
- Cleaning the data to remove unwanted data, missing values, rows, and columns, duplicate values, data type conversion, etc. You might even have to restructure the dataset and change the rows and columns or index of rows and columns.
- Visualize the data to understand how it is structured and understand the relationship between various variables and classes present.

- Splitting the cleaned data into two sets - a training set and a testing set. The training set is the set your model learns from. A testing set is used to check the accuracy of your model after training.

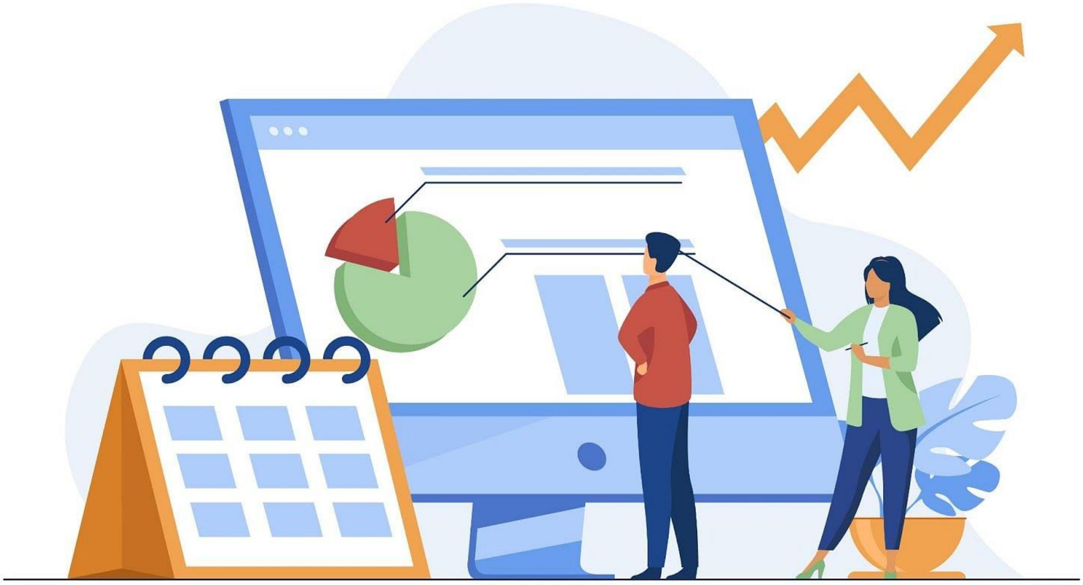


Figure 3: Cleaning and Visualizing Data

3. Choosing a Model:

A machine learning model determines the output you get after running a machine learning algorithm on the collected data. It is important to choose a model which is relevant to the task at hand. Over the years, scientists and engineers developed various models suited for different tasks like speech recognition, image recognition, prediction, etc. Apart from this, you also have to see if your model is suited for numerical or categorical data and choose accordingly.

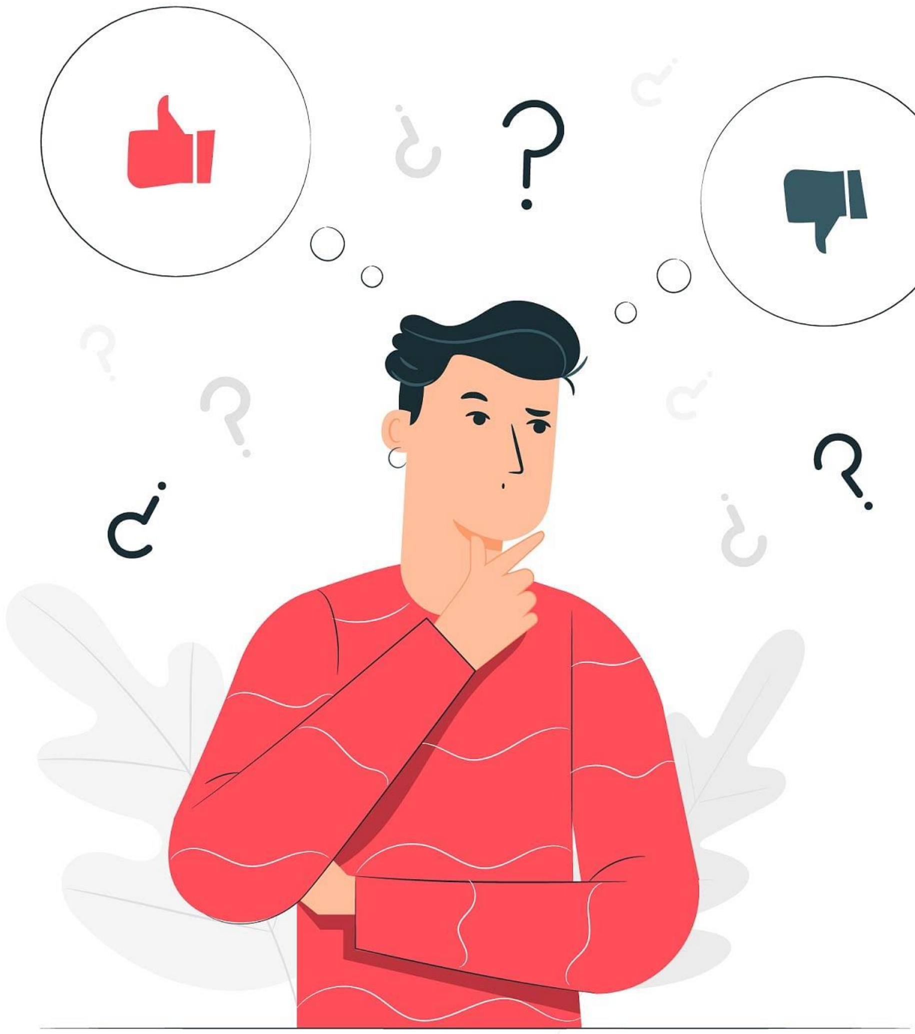


Figure 4: Choosing a model

4. Training the Model:

Training is the most important step in machine learning. In training, you pass the prepared data to your machine learning model to find patterns and make predictions. It results in the model learning from the data so that it can accomplish the task set. Over time, with training, the model gets better at predicting.

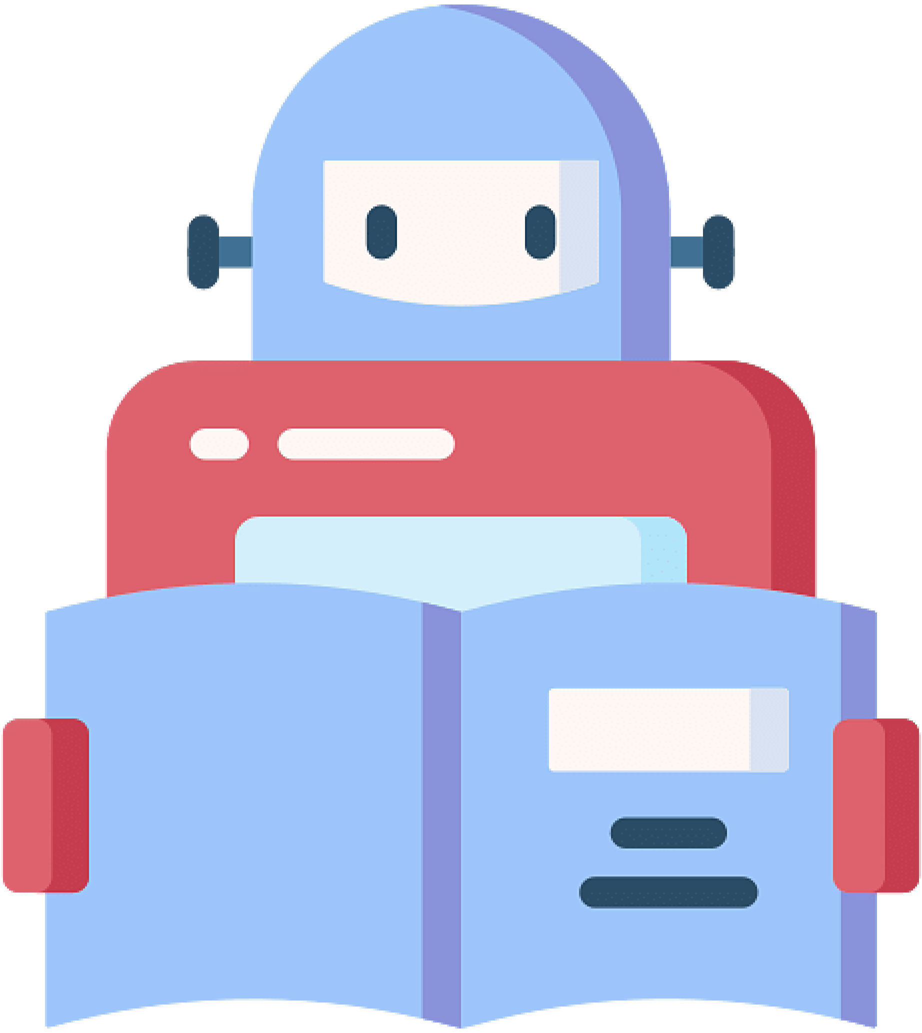


Figure 5: Training a model

5. Evaluating the Model:

After training your model, you have to check to see how it's performing. This is done by testing the performance of the model on previously unseen data. The unseen data used is the testing set that you split our data into earlier. If testing was done on the same data which is used for training, you will not get an accurate measure, as the model is already used to the data, and finds the same patterns in it, as it previously did. This will give you disproportionately high accuracy.

When used on testing data, you get an accurate measure of how your model will perform and its speed.

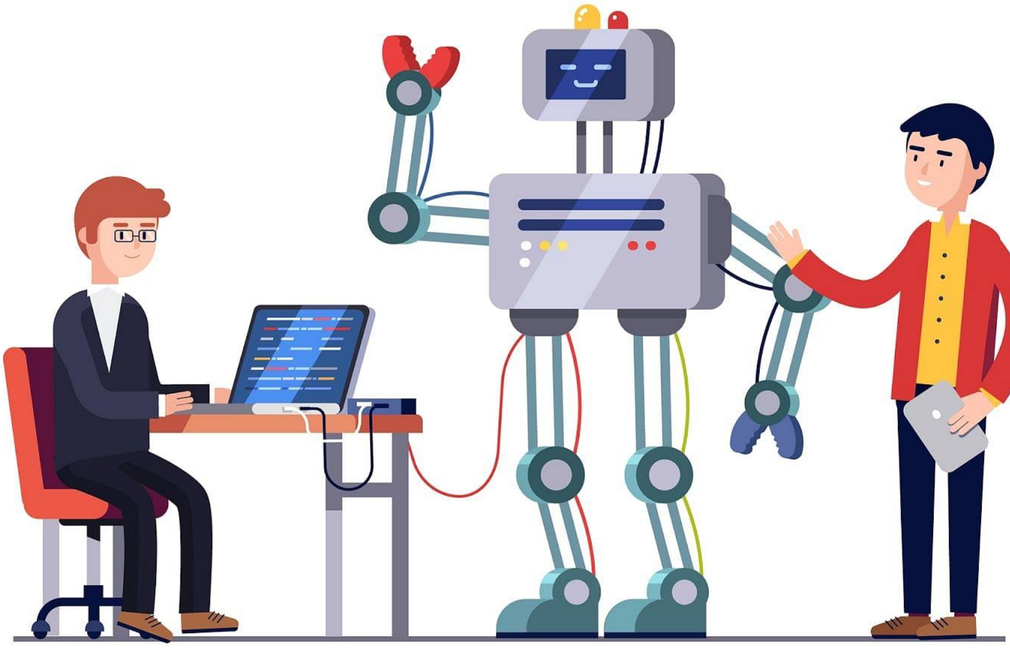


Figure 6: Evaluating a model

6. Parameter Tuning:

Once you have created and evaluated your model, see if its accuracy can be improved in any way. This is done by tuning the parameters present in your model. Parameters are the variables in the model that the programmer generally decides. At a particular value of your parameter, the accuracy will be the maximum. Parameter tuning refers to finding these values.

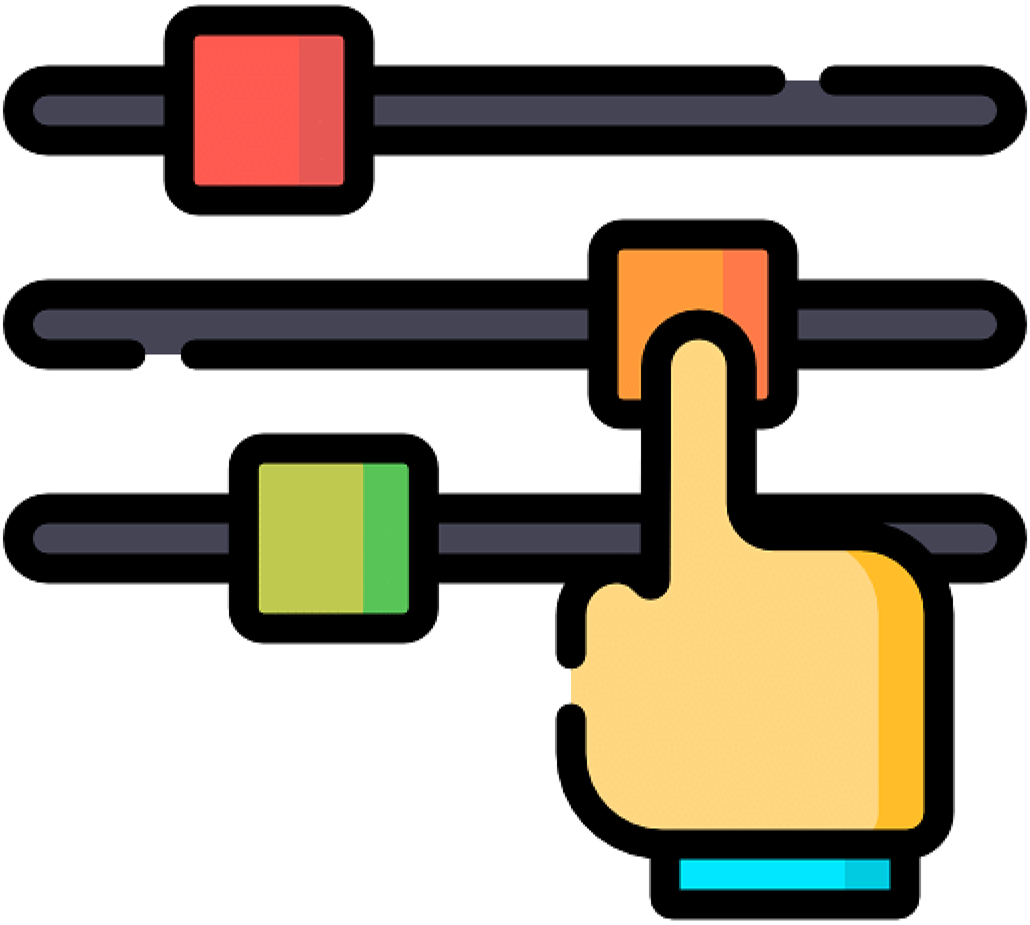


Figure 7: Parameter Tuning

7. Making Predictions

In the end, you can use your model on unseen data to make predictions accurately.

How to Implement Machine Learning Steps in Python?

You will now see how to implement a machine learning model using Python.

In this example, data collected is from an insurance company, which tells you the variables that come into play when an insurance amount is set. Using this, you will have to predict the insurance amount for a person. This data was collected from Kaggle.com, which has many reliable datasets.

You need to start by importing any necessary modules, as shown.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score
```

Figure 8: Importing necessary modules

Following this, you will import the data.

```
dataset = pd.read_csv('/Users/guess/Downloads/insurance.csv')
dataset
```

Figure 9: Importing data

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
1333	50	male	30.970	3	no	northwest	10600.54830
1334	18	female	31.920	0	no	northeast	2205.98080
1335	18	female	36.850	0	no	southeast	1629.83350
1336	21	female	25.800	0	no	southwest	2007.94500
1337	61	female	29.070	0	yes	northwest	29141.36030

1338 rows × 7 columns

Figure 10: Insurance dataset

Now, clean your data by removing duplicate values, and transforming columns into numerical values to make them easier to work with.

```
label = LabelEncoder()
label.fit(dataset.sex.drop_duplicates())
dataset.sex = label.transform(dataset.sex)
# smoker or not
label.fit(dataset.smoker.drop_duplicates())
dataset.smoker = label.transform(dataset.smoker)
#region
label.fit(dataset.region.drop_duplicates())
dataset.region = label.transform(dataset.region)

dataset
```

Figure 11: Cleaning Data

The final dataset becomes as shown.

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	3	16884.92400
1	18	1	33.770	1	0	2	1725.55230
2	28	1	33.000	3	0	2	4449.46200
3	33	1	22.705	0	0	1	21984.47061
4	32	1	28.880	0	0	1	3866.85520
...	...	...	...	...	...	...	...
1333	50	1	30.970	3	0	1	10600.54830
1334	18	0	31.920	0	0	0	2205.98080
1335	18	0	36.850	0	0	2	1629.83350
1336	21	0	25.800	0	0	3	2007.94500
1337	61	0	29.070	0	1	1	29141.36030

1338 rows × 7 columns

Figure 12: Cleaned dataset

Now, split your dataset into training and testing sets.

```python
X_lin = dataset.drop(['charges'], axis =1)
y_lin = dataset[['charges']]

X_lin_train, X_lin_test, y_lin_train, y_lin_test = train_test_split(X_lin, y_lin, test_size=0.3, random_state=42)
```

Figure 13: Splitting the dataset

As you need to predict a numeral value based on some parameters, you will have to use Linear Regression. The model needs to learn on your training set. This is done by using the '.fit' command.

```python
Linear_model = LinearRegression()
Linear_model.fit(X_lin_train, y_lin_train)
```

Figure 14: Choosing and training your model

Now, predict your testing dataset and find how accurate your predictions are.

```python
pred = Linear_model.predict(X_lin_test)
pred
```

```
       [ 3.60294190e+03],
       [ 4.40040903e+03],
       [ 1.40663345e+04],
       [ 1.16268203e+04],
       [ 8.89219642e+03],
       [ 1.21011367e+04],
       [ 5.23906853e+03],
       [ 2.84241293e+03],
       [ 3.56294259e+04],
       [ 9.27854339e+03],
       [ 1.59720792e+04],
       [ 2.34524488e+03],
       [ 1.24695907e+04],
       [ 1.45575199e+03],
       [ 1.36060478e+04],
       [ 1.27386152e+04],
       [ 4.36613796e+03],
       [ 3.22719994e+04],
       [ 1.32349447e+04],
```

```python
Linear_model.score(X_lin_test,pred)
```

```
1.0
```

Figure 15: Predicting using your model

1.0 is the highest level of accuracy you can get. Now, get your parameters.

```
for idx, col_name in enumerate(X_lin_train.columns):
    print("The coefficient for {} is {}".format(col_name, Linear_model.coef_[0][idx]))
```

```
The coefficient for age is 261.62568984274697
The coefficient for sex is 109.64719595061018
The coefficient for bmi is 344.54483065603415
The coefficient for children is 424.37016595763396
The coefficient for smoker is 23620.80252148174
The coefficient for region is -326.4626252721925
```

```
intercept = Linear_model.intercept_[0]
intercept
```

```
-12364.391322279203
```

Figure 16: Model Parameters

The above picture shows the hyperparameters which affect the various variables in your dataset.

NumPy refers to Numerical Python. It is an open-source library in Python that aids in mathematical and numerical calculations and computations; and, scientific, engineering, and data science programming. NumPy is an essential library used to perform mathematical and statistical operations. It is especially suited for multi-dimensional arrays and matrix multiplications.

Jupyter Notebook, which is an open-source web application that comes with built-in packages and enables you to run code in real-time.

Let's get started by importing our NumPy module and writing basic code.

NumPy is usually imported under the np alias.

```
#importing Numpy
import numpy as np
a = np.array([1,2,3])
print (a)
```

```
[1 2 3]
```

Fig: Basic NumPy example

You can also make more than a one-dimensional array.

```
In [10]: #creating an array
         import numpy as np
         a = np.array([1,2,3])
         print (a)
         print(type(a))

         [1 2 3]
         <class 'numpy.ndarray'>
```

Fig: 2-D NumPy array

NumPy Arrays

The array object in NumPy is called ndarray, which means an N-dimensional array.

To create ndarray in NumPy, we use the array() function.

```
In [10]: #creating an array
         import numpy as np
         a = np.array([1,2,3])
         print (a)
         print(type(a))

         [1 2 3]
         <class 'numpy.ndarray'>
```

Fig: NumPy array

NumPy Array functions

- ndim

The ndim() attribute can be used to find the dimensions of the array.

```
In [12]: a = np.array(8)
         b = np.array([1, 2, 3])
         c = np.array([[1, 2, 3], [4, 5, 6]])
         d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])

         print(a.ndim)
         print(b.ndim)
         print(c.ndim)
         print(d.ndim)

         0
         1
         2
         3
```

Fig: ndim function

- itemsize()

The itemsize() is used to calculate the byte size of each element.

```
In [13]: import numpy as np
         a = np.array([[1,2],[3,4]])
         print (a.itemsize)

         4
```

Fig: itemsize()

Each item occupies four bytes in the above array.

- dtype

The dtype attribute is used to understand the data type of the given element.

```
In [15]: import numpy as np
         a = np.array([[1,2],[3,4]])
         print (a.dtype)

         int32
```

Fig: dtype

- shape

This array attribute returns a tuple consisting of array dimensions.

```
In [16]: import numpy as np
         a = np.array([[1,2],[3,4]])
         print (a.shape)

         (2, 2)
```

Fig: shape

This means the array has two dimensions, and each dimension contains two elements.

- reshape()

The reshape() function is used to reshape the array.

```
In [17]: import numpy as np
         a = np.array([[1,2,3],[4,5,6]])
         a = a.reshape(3,2)
         print (a)

         [[1 2]
          [3 4]
          [5 6]]
```

Fig: reshape()

Now the array has three dimensions, with two elements in each dimension.

- Slicing

Slicing is used to extract a range of elements from the array.

```
In [19]: import numpy as np
         a = np.array([[1,2,3],[4,5,6]])
         print(a[0,1])

         2
```

Fig: Slicing

Zero represents the index of the array, and one indicates the element of the mentioned array.

- random.rand()

The random module's rand() method returns a random float between zero and one.

Example:

```
In [7]: import numpy as np
        a=np.random.rand(3,2)
        a

Out[7]: array([[0.85211241, 0.98646421],
               [0.08201986, 0.7948377 ],
               [0.8422548 , 0.52107172]])
```

Fig: random()

We have generated a three-dimensional array with two elements in each dimension.

- random.randint()

The randint() method takes a size parameter where you can specify the shape of the array.

Example:

```
In [3]: from numpy import random
        x = random.randint(10, size=(3, 4))
        print(x)

        [[6 7 4 8]
         [4 5 2 4]
         [2 7 1 0]]
```

Fig: random.randint()

The code above will generate a 2D array with three rows, and each row will contain four random integers between zero and 10.

- mean()

The mean() function is used to compute the arithmetic mean of the given data along the specified axis.

Example:

```
In [7]: import numpy as np
        a = np.array([[1,2], [3,4]])
        b=np.mean(a,axis=0)
        c=np.mean(a,axis=1)
        print (b)
        print (c)

        [2. 3.]
        [1.5 3.5]
```

Fig: mean()

- median()

The median() function is used to compute the arithmetic median of the given data along the specified axis.

Example:

```
In [10]: import numpy as np
         a = np.array([[1,2,3,4], [5,6,7,8],[9,10,11,12]])
         b=np.median(a,axis=0)
         c=np.median(a,axis=1)
         print (b)
         print (c)

         [5. 6. 7. 8.]
         [ 2.5  6.5 10.5]
```

Fig: median()

- std()

The std() function is used to compute the standard deviation along the specified axis.

Example:

```
In [11]: a=np.array([[1,4,7,10],[2,5,8,11]])
         b=np.std(a, axis=0)
         print (b)

         [0.5 0.5 0.5 0.5]
```

Fig: std()

- append()

The append() function is used to add new values to an existing array.

Example:

```
In [14]: import numpy as np
         a=np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
         b=np.array([[1, 1, 3], [4, 2, 6], [7, 3, 9]])
         c=np.append(a,b)
         print(c)

         [1 2 3 4 5 6 7 8 9 1 1 3 4 2 6 7 3 9]
```

Fig: append()

- insert()

The insert() function inserts the value in the input array along the mentioned axis.

Syntax: numpy.insert(arr, obj, values, axis)

Example:

```
In [17]: import numpy as np
         a = np.array([[1,2],[3,4],[5,6]])
         print ('Broadcast along axis 0:')
         print (np.insert(a,1,[4],axis = 0))
         print ('\n')

         Broadcast along axis 0:
         [[1 2]
          [4 4]
          [3 4]
          [5 6]]
```

Fig: insert()

- concatenate()

The concatenate() function is used for joining two or more arrays of the same shape along the specified axis.

Example:

```
In [12]: x=np.array([[1,6],[3,4]])
         y=np.array([[2,12]])
         z=np.concatenate((x,y))
         print(z)

[[ 1  6]
 [ 3  4]
 [ 2 12]]
```

Fig: concatenate()

NumPy Mathematical Operation

In NumPy, basic mathematical functions operate elementwise on an array. Let's look at some examples to understand this more clearly.

```
In [2]: import numpy as np

        x = np.array([[1,2],[3,4]], dtype=np.float64)
        y = np.array([[5,6],[7,8]], dtype=np.float64)

        # Elementwise sum
        print(x + y)
        print(np.add(x, y))

        # Elementwise difference
        print(x - y)
        print(np.subtract(x, y))

        # Elementwise product
        print(x * y)
        print(np.multiply(x, y))

        # Elementwise division
        print(x / y)
        print(np.divide(x, y))
```

Fig: Mathematical operations

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

Python pandas is one of the most widely-used Python libraries in data science and analytics. It provides high-performance, easy-to-use structures, and data analysis tools. Two-dimensional table objects in pandas are referred to as DataFrame, as well as Series. It is a structure that contains column names and row labels.

What is Python Pandas?

Pandas is an open-source Python library that provides high-performance, easy-to-use data structure, and data analysis tools for the Python programming language.

Python with pandas is used in a wide range of fields, including academics, retail, finance, economics, statistics, analytics, and many others.

Python pandas is well suited for different kinds of data, such as:

- Ordered and unordered time series data
- Unlabeled data

- Any other form of observational or statistical data sets

Pandas Series

Series is a one-dimensional array that can contain any type of data. You can create a series by using the following constructor:

pandas.Series(data, index, dtype, copy)

Example:

```
In [3]: #import the pandas library
        import pandas as pd
        s = pd.Series()
        print (s)

        Series([], dtype: float64)
```

Fig: importing pandas module

Basic Operations on Series

- Create a series from ndarray

```
In [5]: import pandas as pd
        import numpy as np
        data = np.array(['a','b','c'])
        x = pd.Series(data)
        print (x)

        0    a
        1    b
        2    c
        dtype: object
```

Fig: ndarray series

If you don't mention the index of the array, it begins at zero by default.

- Create a series from a dictionary

A dictionary data structure can be passed as an input in the series.

Example:

```
In [7]: import pandas as pd
        import numpy as np
        data = {'a' : 0., 'b' : 1., 'c' : 2.}
        x = pd.Series(data)
        print (x)

        a    0.0
        b    1.0
        c    2.0
        dtype: float64
```

Fig: Series from a dictionary

- Accessing data from a series

To access the data in the series, we enter the index number of the element or the label on an element.

Example:

```
In [9]: import pandas as pd
        import numpy as np
        data = np.array(['a','b','c'])
        x = pd.Series(data)
        print (x[0])

        a
```

Fig: Access data in a series

To retrieve data using labels, we enter the label value.

Example:

```
In [11]: import pandas as pd
         s = pd.Series([1,2,3],index = ['a','b','c'])
         print (s['a'])

         1
```

Fig: Retrieving data by label name

Pandas DataFrame

A DataFrame is a multi-dimensional data structure in which data is arranged in the form of rows and columns. You can create a DataFrame using the following constructor:

pandas.DataFrame(data, index, columns, dtype, copy)

Example:


```
In [13]: import pandas as pd
         x = pd.DataFrame()
         print (x)

         Empty DataFrame
         Columns: []
         Index: []
```

Fig: Empty DataFrame

Basic Operations on DataFrames

- Create a DataFrame from lists

A DataFrame can be created using a list:

```
In [14]: import pandas as pd
         data = [1,2,3,4,5]
         x = pd.DataFrame(data)
         print (x)

            0
         0  1
         1  2
         2  3
         3  4
         4  5
```

Fig: DataFrame

```
In [16]: import pandas as pd
         data = [['John',22],['Carter',25],['harold',33]]
         x = pd.DataFrame(data,columns=['Name','Age'])
         print (x)

              Name  Age
         0    John   22
         1  Carter   25
         2  harold   33
```

Fig: 2-D DataFrame

- Creating a DataFrame from a series dictionary

A series dictionary can be passed to form a DataFrame.

Example:

```
In [17]: import pandas as pd

         d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
            'two' : pd.Series([4, 5, 6, 7], index=['a', 'b', 'c', 'd'])}

         x = pd.DataFrame(d)
         print (x)

            one  two
         a  1.0    4
         b  2.0    5
         c  3.0    6
         d  NaN    7
```

Fig: DataFrame from a Series dictionary

Let us now look at the column selection, addition, and deletion, and indexing a DataFrame through an example.

- Column selection

You select a particular column by mentioning the column name.

Example:

```
In [18]: import pandas as pd

         d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
            'two' : pd.Series([4, 5, 6, 7], index=['a', 'b', 'c', 'd'])}

         x = pd.DataFrame(d)
         print (x['one'])

         a    1.0
         b    2.0
         c    3.0
         d    NaN
         Name: one, dtype: float64
```


Fig: Column selection

- Addition of a new column

The following enables users to incorporate new columns into the data provided:

```
In [20]: import pandas as pd

         d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
            'two' : pd.Series([4, 5, 6, 7], index=['a', 'b', 'c', 'd'])}
         x = pd.DataFrame(d)

         print ("Adding a new column by passing as Series:")
         x['three']=pd.Series([5,20,3],index=['a','b','c'])
         print (x)

         Adding a new column by passing as Series:
            one  two  three
         a  1.0    4    5.0
         b  2.0    5   20.0
         c  3.0    6    3.0
         d  NaN    7    NaN
```

Fig: Adding a new column

- Deleting a column

Columns can be deleted using the del or pop functions.

Example:

```
In [21]: import pandas as pd

         d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
            'two' : pd.Series([4, 5, 6, 7], index=['a', 'b', 'c', 'd']),
            'three' : pd.Series([8,9,10], index=['a','b','c'])}

         x = pd.DataFrame(d)
         print ("Our dataframe is:")
         print (x)

         # using del function
         print ("Deleting column using DEL function:")
         del x['one']
         print (x)

         # using pop function
         print ("Deleting column using POP function:")
         x.pop('two')
         print (x)
```

Fig: Deleting a column

- Indexing a DataFrame

The iloc() method is used for integer-based indexing.

Example:

```
In [23]: # import the pandas library and aliasing as pd
         import pandas as pd
         import numpy as np

         x = pd.DataFrame(np.random.randn(6, 4), columns = ['A', 'B', 'C', 'D'])

         # select all rows for a specific column
         print (x.iloc[:4])

                  A         B         C         D
         0  0.514884 -1.108968 -0.272898 -0.810167
         1 -0.201098  0.822071 -0.831389 -2.102403
         2 -0.259776 -0.524546  1.923136 -0.488985
         3 -0.236475  0.642512  0.729878 -0.581850
```

Fig: iloc()

Python Pandas Sorting

There are two types of sorting available in pandas. They are:

- By label
- By actual value

By Label

The sort_index() method is used to sort data in pandas. You pass the axis arguments and order of the sorting.

Example:

```
In [32]: import pandas as pd
         import numpy as np

         unsorted = pd.DataFrame(np.random.randn(5,2),index=[1,4,6,2,3],columns = ['col2','col1'])


         x = unsorted.sort_index()
         print (x)

                col2      col1
         1 -0.224505 -1.164272
         2 -1.147121 -0.269138
         3 -0.522339 -1.058028
         4  0.188638 -0.284197
         6 -1.380247 -1.012464
```

Fig: Sorting by label

By default, sorting is done in ascending order.

By Actual Value

The sort_values() method is used to sort the column according to values.

Example:

```
In [34]: import pandas as pd
         import numpy as np

         unsorted_df = pd.DataFrame({'col1':[4,5,7,8],'col2':[1,3,2,4]})
         sorted_df = unsorted_df.sort_values(by='col1')

         print (sorted_df)

            col1  col2
         0     4     1
         1     5     3
         2     7     2
         3     8     4
```

Fig: By actual value

Python Pandas GroupBy

The groupby function performs one of the following operations on original data. They include:

- Splitting the object
- Applying a function
- Combining the result

Let's create a DataFrame object and perform all the operations.

Example:

```
In [32]: import pandas as pd

         exam_data = {'Student': ['Jack', 'Jack', 'Denver', 'Denver', 'Koleman',
            'Koleman', 'Koleman', 'Koleman', 'Rachel', 'Ross', 'Ross', 'Rachel'],
            'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
            'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
            'Marks':[896,749,883,613,721,610,746,758,664,711,884,640]}
         x = pd.DataFrame(exam_data)

         print (x)

            Student  Rank  Year  Marks
         0     Jack     1  2014    896
         1     Jack     2  2015    749
         2   Denver     2  2014    883
         3   Denver     3  2015    613
         4  Koleman     3  2014    721
         5  Koleman     4  2015    610
         6  Koleman     1  2016    746
         7  Koleman     1  2017    758
         8   Rachel     2  2016    664
         9     Ross     4  2014    711
         10    Ross     1  2015    884
         11  Rachel     2  2017    640
```

Fig: DataFrame

Split Data by Groups

Let us see how grouping objects can be used in DataFrames.

Example:

```
In [33]: import pandas as pd

         exam_data = {'Student': ['Jack', 'Jack', 'Denver', 'Denver', 'Koleman',
            'Koleman', 'Koleman', 'Koleman', 'Rachel', 'Ross', 'Ross', 'Rachel'],
            'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
            'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
            'Marks':[896,749,883,613,721,610,746,758,664,711,884,640]}
         x = pd.DataFrame(exam_data)

         print (x.groupby('Student'))

         <pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000006C6C11FC48>
```

Fig: Splitting data into groups

View Groups

```
In [35]: import pandas as pd

         exam_data = {'Student': ['Jack', 'Jack', 'Denver', 'Denver', 'Koleman',
            'Koleman', 'Koleman', 'Koleman', 'Rachel', 'Ross', 'Ross', 'Rachel'],
            'Rank': [1, 2, 2, 3, 3,4 ,1 ,1,2 , 4,1,2],
            'Year': [2014,2015,2014,2015,2014,2015,2016,2017,2016,2014,2015,2017],
            'Marks':[896,749,883,613,723,610,746,758,664,711,884,640]}
         x = pd.DataFrame(exam_data)

         print (x.groupby('Student').groups)

{'Denver': Int64Index([2, 3], dtype='int64'), 'Jack': Int64Index([0, 1], dtype='int64'), 'Koleman': Int64Index([4, 5, 6, 7], dt
ype='int64'), 'Rachel': Int64Index([8, 11], dtype='int64'), 'Ross': Int64Index([9, 10], dtype='int64')}
```

Fig: View groups

Python Pandas: Merging

You can merge two DataFrames by including the key in the following way:

```
In [36]: import pandas as pd
         left = pd.DataFrame({
            'id':[1,2,3,4,5],
            'Name': ['Jack', 'Amy', 'Elias', 'Young', 'Smith'],
            'subject_id':['sub1','sub2','sub4','sub6','sub5']})
         right = pd.DataFrame({
            'id':[1,2,3,4,5],
            'Name': ['Billy', 'Brooks', 'Brown', 'Aurier', 'Jose'],
            'subject_id':['sub2','sub4','sub3','sub6','sub5']})
         print (pd.merge(left,right,on='id'))

            id Name_x subject_id_x  Name_y subject_id_y
         0   1   Jack         sub1   Billy         sub2
         1   2    Amy         sub2  Brooks         sub4
         2   3  Elias         sub4   Brown         sub3
         3   4  Young         sub6  Aurier         sub6
         4   5  Smith         sub5    Jose         sub5
```

Fig: Merging two DataFrames

In the above program, we used the ‘id’ column as a common key.

Python Pandas: Concatenation

The concat function is used to concatenate two DataFrames.

Example:

```
In [38]: import pandas as pd

         one = pd.DataFrame({
            'Name': ['Amber', 'Jack', 'Brown', 'Smith', 'Young'],
            'subject_id':['sub1','sub2','sub4','sub6','sub5'],
            'Marks_scored':[93,90,82,64,71]},
            index=[1,2,3,4,5])

         two = pd.DataFrame({
            'Name': ['Ben', 'Cole', 'Sam', 'Tom', 'Martial'],
            'subject_id':['sub2','sub4','sub3','sub6','sub5'],
            'Marks_scored':[96,80,73,77,81]},
            index=[1,2,3,4,5])
         print (pd.concat([one,two]))

      Name subject_id  Marks_scored
1    Amber       sub1            93
2     Jack       sub2            90
3    Brown       sub4            82
4    Smith       sub6            64
5    Young       sub5            71
1      Ben       sub2            96
2     Cole       sub4            80
3      Sam       sub3            73
4      Tom       sub6            77
5  Martial       sub5            81
```


Fig: Concatenation

Bernoulli Naive Bayes

Introduction

To understand Bernoulli Naive Bayes algorithm, it is essential to understand Naive Bayes.

Naive Bayes is a supervised machine learning algorithm to predict the probability of different classes based on numerous attributes. It indicates the likelihood of occurrence of an event. Naive Bayes is also known as conditional probability.

Naive Bayes is based on the Bayes Theorem.

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)}$$

where:-

A: event 1
B: event 2
P(A|B): Probability of A being true given B is true - posterior probability
P(B|A): Probability of B being true given A is true - the likelihood
P(A): Probability of A being true - prior
P(B): Probability of B being true - marginalization
However, in the case of the Naive Bayes classifier, we are concerned only with the maximum posterior probability, so we ignore the denominator, i.e., the marginal probability.

The Naive Bayes classifier is based on two essential assumptions:-

(i) **Conditional Independence** - All features are independent of each other. This implies that one feature does not affect the performance of the other. This is the sole reason behind the 'Naive' in 'Naive Bayes.'

(ii) **Feature Importance** - All features are equally important. It is essential to know all the features to make good predictions and get the most accurate results.

Naive Bayes is classified into three main types: Multinomial Naive Bayes, Bernoulli Naive Bayes, and Gaussian Bayes.

Before going ahead, let us have a look at the **Bernoulli** Distribution:-

Let there be a random variable 'X' and let the probability of success be denoted by 'p' and the likelihood of failure be represented by 'q.'

Success: p
Failure: q

q = 1 - (probability of Sucesss)
q = 1 - p

$$p(x) = P[X = x] = \begin{cases} q = 1 - p & x = 0 \\ p & x = 1 \end{cases}$$

$$X = \begin{cases} 1 & \text{Bernoulli trial} = \mathbf{S} \\ 0 & \text{Bernoulli trial} = \mathbf{F} \end{cases}$$

As we notice above, x can take only two values (binary values), i.e., 0 or 1.

Bernoulli Naive Bayes is a part of the Naive Bayes family. It is based on the Bernoulli Distribution and accepts only binary values, i.e., 0 or 1. If the features of the dataset are binary, then we can assume that Bernoulli Naive Bayes is the algorithm to be used.

Example:

(i) Bernoulli Naive Bayes classifier can be used to detect whether a person has a disease or not based on the data given. This would be a binary classification problem so that Bernoulli Naive Bayes would work well in this case.

(ii) Bernoulli Naive Bayes classifier can also be used in text classification to determine whether an SMS is 'spam' or 'not spam.'

Mathematics Behind

Let us consider the example below to understand Bernoulli Naive Bayes:-

Adult	Gender	Fever	Disease
Yes	Female	No	False
Yes	Female	Yes	True

No	Male	Yes	False
No	Male	No	True
Yes	Male	Yes	True

In the above dataset, we are trying to predict whether a person has a disease or not based on their age, gender, and fever. Here, ‘Disease’ is the target, and the rest are the features.
All values are binary.

We wish to classify an instance ‘X’ where Adult=’Yes’, Gender= ‘Male’, and Fever=’Yes’.

Firstly, we calculate the class probability, probability of disease or not.

P(Disease = True) = $\frac{3}{5}$

P(Disease = False) = $\frac{2}{5}$

Secondly, we calculate the individual probabilities for each feature.

P(Adult= Yes | Disease = True) = $\frac{2}{3}$
P(Gender= Male | Disease = True) = $\frac{2}{3}$
P(Fever= Yes | Disease = True) = $\frac{2}{3}$

P(Adult= Yes | Disease = False) = $\frac{1}{2}$
P(Gender= Male | Disease = False) = $\frac{1}{2}$
P(Fever = Yes | Disease = False) = $\frac{1}{2}$

Now, we need to find out two probabilities:-
(i) P(Disease= True | X) = (P(X | Disease= True) * P(Disease=True))/ P(X)
(ii) P(Disease = False | X) = (P(X | Disease = False) * P(Disease= False))/P(X)

P(Disease = True | X) = (($\frac{2}{3}$ *$\frac{2}{3}$ * $\frac{2}{3}$) * ($\frac{3}{5}$))/P(X) = (8/27 * $\frac{3}{5}$) / P(X) = 0.17/P(X)

P(Disease = False | X) = [($\frac{1}{2}$ * $\frac{1}{2}$ * $\frac{1}{2}$) * ($\frac{2}{5}$)] / P(X) = [$\frac{1}{8}$ * $\frac{2}{5}$] / P(X) = 0.05/ P(X)

Now, we calculate estimator probability:-

P(X) = P(Adult= Yes) * P(Gender = Male) * P(Fever = Yes)
= $\frac{3}{5}$ * $\frac{3}{5}$ * $\frac{3}{5}$ = 27/125 = 0.21

So we get finally:-

P(Disease = True | X) = 0.17 / P(X)
= 0.17 / 0.21
= 0.80 - (1)

P(Disease = False | X) = 0.05 / P(X)
= 0.05 / 0.21
= 0.23 - (2)

Now, we notice that (1) > (2), the result of instance ‘X’ is ‘True’, i.e., the person has the disease.

Practice Example: BNB Classifier

[Dataset Name: spam.csv][Uploaded to dashboard]

The dataset we're using will be helpful in SMS Message Spam Detection.

(i) message - text message, categorical feature.

(ii) class - target variable, binary feature - ‘spam’ or ‘ham’ .

Implementation
For self-implementation, we will have to create three functions, one for estimating prior probability, one for estimating conditional probability, and one for prediction.

For simplicity, we would be using the already existing sklearn library for Bernoulli Naive Bayes implementation.

Importing Necessary Libraries
Firstly, we will load some basic libraries:-
(i) Numpy - for linear algebra.
(ii) Pandas - for data analysis.
(iii) Seaborn - for data visualization.
(iv) Matplotlib - for data visualisation.
(v) BernoulliNB - for Bernoulli Naive Bayes implementation.
(vi) CountVectorizer - for sparse matrix representation.

```
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.naive_bayes import BernoulliNB
from sklearn.feature_extraction.text import CountVectorizer
```

Loading Data

```
#loading                                                          dataset
df = pd.read_csv('spam.csv', encoding= 'latin-1')
```

Visualization
We visualize the dataset by printing the first ten rows of the data frame. We use the head() function for the same.

```
#visualizing                                                      dataset
df.head(n=10)
```

Output

	class	message	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN
5	spam	FreeMsg Hey there darling it's been 3 week's n...	NaN	NaN	NaN
6	ham	Even my brother is not like to speak with me. ...	NaN	NaN	NaN
7	ham	As per your request 'Melle Melle (Oru Minnamin...	NaN	NaN	NaN
8	spam	WINNER!! As a valued network customer you have...	NaN	NaN	NaN
9	spam	Had your mobile 11 months or more? U R entitle...	NaN	NaN	NaN

Above, we observe all the features and the target variable 'class.' Also, we notice that three columns 'Unnamed:2', 'Unnamed:3' and 'Unnamed:4' contain many NaN or missing values. We will be handling the same in the next section.

Now, we use the shape function to get an idea about the dimensions of the dataset.

```
df.shape
```

Output

```
(5572, 5)
```

From the above, we observe there are 5572 examples and five columns.

Preprocessing
1. **<u>Data imputation</u>**

We drop 'Unnamed:2', 'Unnamed:3' and 'Unnamed:4' as they contain too many missing values. Also, these features are unknown, so there is no point in retaining them.

```
#dropping columns with too many NaN values
df= df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], axis=1)
```

```
df.shape
```

Output

```
(5572, 2)
```

We notice that three features have been dropped, and now our data contains just two columns, one representing the 'message' feature and one representing the 'class' target variable.

2. **Binarization**

To test Bernoulli Naive Bayes on the dataset, we need to ensure that all values are binary.

So, firstly, we check if our target variable values are binary or not.

```
#checking if target variable is binary or not
np.unique(df['class'])
#2 unique values, hence it is binary
```

Output

```
array(['ham', 'spam'], dtype=object)
```

We notice that our target variable has binary values, 'ham' or 'spam.'

Secondly, we check if our ‘message’ feature values are binary or not.

```
#checking if 'message' feature is binary or not
np.unique(df['message'])
# > 2 unique values , hence it is not binary
```

Output

```
array([' &lt;#&gt;  in mca. But not conform.',
       ' &lt;#&gt;  mins but i had to stop somewhere first.',
       ' &lt;DECIMAL&gt; m but its not a common car here so its better to buy from china or asia. Or if i find it les
s expensive. I.ll holla',
       ..., 'ÌÏ thk of wat to eat tonight.', 'ÌÏ v ma fan...',
       'ÌÏ wait 4 me in sch i finish ard 5..'], dtype=object)
```

We notice that our ‘message’ feature is not binary. So we will use CountVectorizer to fix this.

3. **Vectorization**

Now, we will use CountVectorizer() to fix our 'message' feature by creating a sparse matrix.

```
#creating sparse matrix using CountVectorizer

#converting df columns to individual array
x =df["message"].values
y = df["class"].values
# creating count vectorizer object
cv = CountVectorizer()
#tranforming values
x = cv.fit_transform(x)
v= x.toarray()
#printing sparse matrix
print(v)
```

Output

```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

4. **Data arrangement**

Now, we will just arrange our dataset such that our target variable is the last column. This will make training easier.

```
#shifting target column to the end
first_col = df.pop('message')
df.insert(0, 'message', first_col)
df
```

Output

	message	class
0	Go until jurong point, crazy.. Available only ...	ham
1	Ok lar... Joking wif u oni...	ham
2	Free entry in 2 a wkly comp to win FA Cup fina...	spam
3	U dun say so early hor... U c already then say...	ham
4	Nah I don't think he goes to usf, he lives aro...	ham
...	...	...
5567	This is the 2nd time we have tried 2 contact u...	spam
5568	Will Ì_ b going to esplanade fr home?	ham
5569	Pity, * was in mood for that. So...any other s...	ham
5570	The guy did some bitching but I acted like i'd...	ham
5571	Rofl. Its true to its name	ham

5572 rows × 2 columns

5. **Train-Test Split**

Now, we will divide our data into training data and testing data. We will have a 3:1 train test split. This would imply that our training data will have 4179 examples, whereas our testing data will have 1393 examples.

```
#train test split = 3:1

train_x = x[:4179]
train_y = y[:4179]

test_x = x[4179:]
test_y = y[4179:]
```

Training

We will build our Bernoulli Naive Bayes model using the sklearn library and then train it.

```
bnb = BernoulliNB(binarize=0.0)
model = bnb.fit(train_x, train_y)
y_pred_train= bnb.predict(train_x)
y_pred_test = bnb.predict(test_x)
```

We have passed 'binarize' as a parameter for binarizing the values of the dataset. We have also generated the prediction, and now we will move on to the results.

Results

Now, we analyze our model and generate the results.

```
print(bnb.score(train_x, train_y)*100)
print(bnb.score(test_x, test_y)*100)
```

```
98.73175400813592
98.20531227566404
```

We notice that we get good results on both training and testing sets. The training set gives us a score of 98.73, whereas the testing set gives us a score of 98.20.

Now, we will also generate classification reports for training and testing sets.

For training set:-

```
#for training set
from sklearn.metrics import classification_report
print(classification_report(train_y, y_pred_train))
```

Output

	precision	recall	f1-score	support
ham	0.99	1.00	0.99	3614
spam	0.99	0.91	0.95	565
accuracy			0.99	4179
macro avg	0.99	0.96	0.97	4179
weighted avg	0.99	0.99	0.99	4179

For testing set:-

```
#for testing set
```

```
from sklearn.metrics import classification_report
print(classification_report(test_y, y_pred_test))
```

```
              precision    recall  f1-score   support

         ham       0.98      1.00      0.99      1211
        spam       0.99      0.87      0.93       182

    accuracy                           0.98      1393
   macro avg       0.99      0.93      0.96      1393
weighted avg       0.98      0.98      0.98      1393
```

As visible from the above, we have been able to get good results. Finally, we are done studying Bernoulli Naive Bayes.

Frequently Asked Questions

1. **How many types of Naive Bayes Classifiers are there?**
   Naive Bayes can be classified into three types:-
   (i) Multinomial Naive Bayes- suitable for discrete features.
   (ii) Bernoulli Naive Bayes - suitable for binary features.
   (iii) Gaussian Naive Bayes - suitable for continuous features.

2. **What is the limitation of the Naive Bayes Classifier?**
   The main limitation of the Naive Bayes classifier is its assumption of conditional independence - all features are independent of each other. In reality, this is highly improbable.

3. **What is the advantage of the Naive Bayes Classifier?**
   The main advantage of the Naive Bayes classifier is that it is really fast in the case of multi-class predictions. When it comes to classification, it performs better than other models such as Logistic Regression.

What does describe () do in Python?

The describe() method **computes and displays summary statistics for a Python dataframe**. (It also operates on dataframe columns and Pandas series objects.)

Pandas DataFrameisnull() Method
The isnull() method **returns a DataFrame object where all the values are replaced with a Boolean value True for NULL values, and otherwise False**.

sum() **returns the number of missing values in the dataset**.

NLTK is **a toolkit build for working with NLP in Python**. It provides us various text processing libraries with a lot of test datasets. A variety of tasks can be performed using NLTK such as tokenizing, parse tree visualization, etc

Python has a module named re to work with RegEx. Here's an example:

```
import re

pattern = '^a...s$'
test_string = 'abyss'
result = re.match(pattern, test_string)

if result:
print("Search successful.")
else:
print("Search unsuccessful.")
```

Here, we used re.match() function to search pattern within the test_string. The method returns a match object if the search is successful. If not, it returns None.

What is from NLTK corpus import Stopwords?

By default, NLTK (Natural Language Toolkit) includes a list of 40 stop words, including: “a”, “an”, “the”, “of”, “in”, etc. The stopwords in nltk are the most common words in data. They are **words that you do not want to use to describe the topic of your content**. They are pre-defined and cannot be removed.

Python String module contains some constants, utility function, and classes for string manipulation.

Python String Module

It's a built-in module and we have to import it before using any of its constants and classes.

String Module Constants

Let's look at the constants defined in the string module.

```
import string


# string module constants

print(string.ascii_letters)

print(string.ascii_lowercase)

print(string.ascii_uppercase)

print(string.digits)

print(string.hexdigits)

print(string.whitespace)  # ' \t\n\r\x0b\x0c'

print(string.punctuation)
```

Output:

```
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz

ABCDEFGHIJKLMNOPQRSTUVWXYZ

0123456789

0123456789abcdefABCDEF



!"#$%&'()*+,-./:;?@[\]^_`{|}~
```

string capwords() function

Python string module contains a single utility function - capwords(s, sep=None). This function split the specified string into words using str.split(). Then it capitalizes each word using str.capitalize() function. Finally, it joins the capitalized words using str.join(). If the optional argument sep is not provided or None, then leading and trailing whitespaces are removed and words are separated with single whitespace. If it's provided then the separator is used to split and join the words.

Python String Module Classes

Python string module contains two classes - Formatter and Template.

Formatter

It behaves exactly same as str.format() function. This class become useful if you want to subclass it and define your own format string syntax.

Snowball Stemmer – NLP

Snowball Stemmer: It is a stemming algorithm which is also known as the Porter2 stemming algorithm as it is a better version of the Porter Stemmer since some issues of it were fixed in this stemmer.

First, let's look at what is stemming-

Stemming: It is the process of reducing the word to its word stem that affixes to suffixes and prefixes or to roots of words known as a lemma. In simple words stemming is reducing a word to its base word or stem in such a way that the words of similar kind lie under a common stem. For example – The words care, cared and caring lie under the same stem 'care'. Stemming is important in natural language processing(NLP).

Some few common **rules of Snowball stemming** are:

Few Rules:
ILY -----> ILI
LY ----->Nill
SS -----> SS
S ----->Nill
ED ----->E,Nill

- *Nill* means the suffix is replaced with nothing and is just removed.
- There may be cases where these rules vary depending on the words. As in the case of the suffix '*ed*' if the words are 'cared' and 'bumped' they will be stemmed as '*care*' and '*bump*'. Hence, here

in cared the suffix is considered as '*d*' only and not '*ed*'. One more interesting thing is in the word '*stemmed*' it is replaced with the word '*stem*' and not '*stemmed*'. Therefore, the suffix depends on the word.

Let's see a few examples:-

WordStem
cared care
university univers
fairly fair
easily easili
singing sing
sings sing
sung sung
singer singer
sportingly sport

Code: Python code implementation of Snowball Stemmer using NLTK library

```
import nltk
from nltk.stem.snowball import SnowballStemmer


#the stemmer requires a language parameter
snow_stemmer = SnowballStemmer(language='english')


#list of tokenized words
words = ['cared','university','fairly','easily','singing',
    'sings','sung','singer','sportingly']


#stem's of each word
stem_words = []
for w in words:
    x = snow_stemmer.stem(w)
    stem_words.append(x)


#print stemming results
for e1,e2 in zip(words,stem_words):
```

```
    print(e1+' ----> '+e2)
```

Output:

cared ----> care

university ---->univers

fairly ----> fair

easily ---->easili

singing ----> sing

sings ----> sing

sung ----> sung

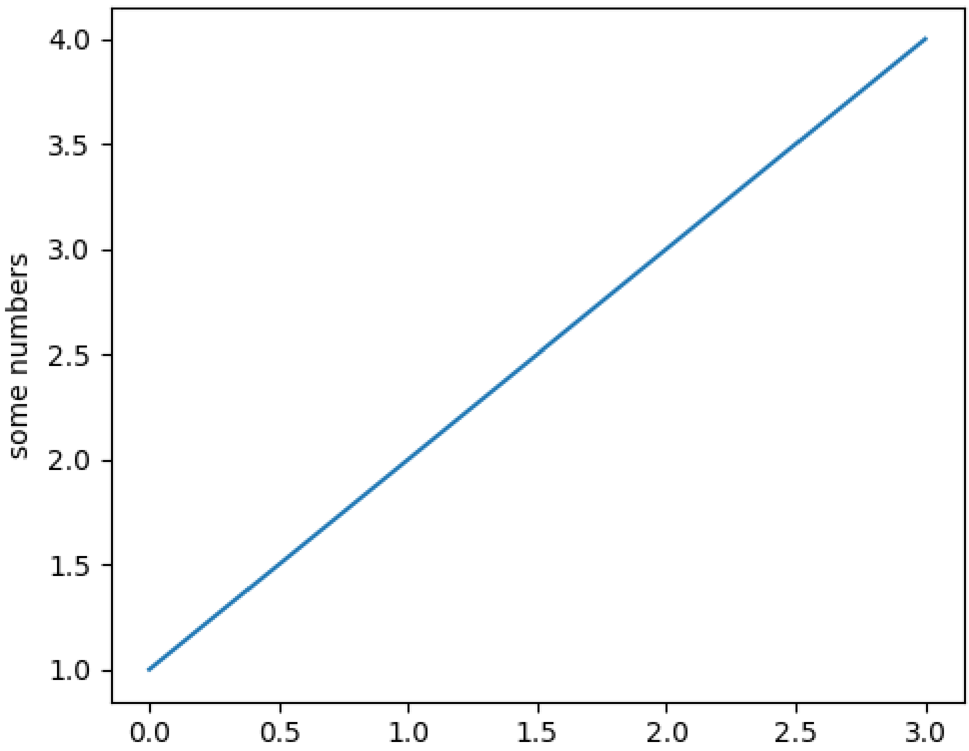
singer ----> singer

sportingly ----> sport

Pyplot

matplotlib.pyplot is a collection of command style functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the *axes* part of a figure and not the strict mathematical term for more than one axis).

```
importmatplotlib.pyplotasplt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```



You may be wondering why the x-axis ranges from 0-3 and the y-axis from 1-4. If you provide a single list or array to the plot() command, matplotlib assumes it is a sequence of y values, and automatically generates the x values for you. Since python ranges start with 0, the default x vector has the same length as y but starts with 0. Hence the x data are [0,1,2,3].

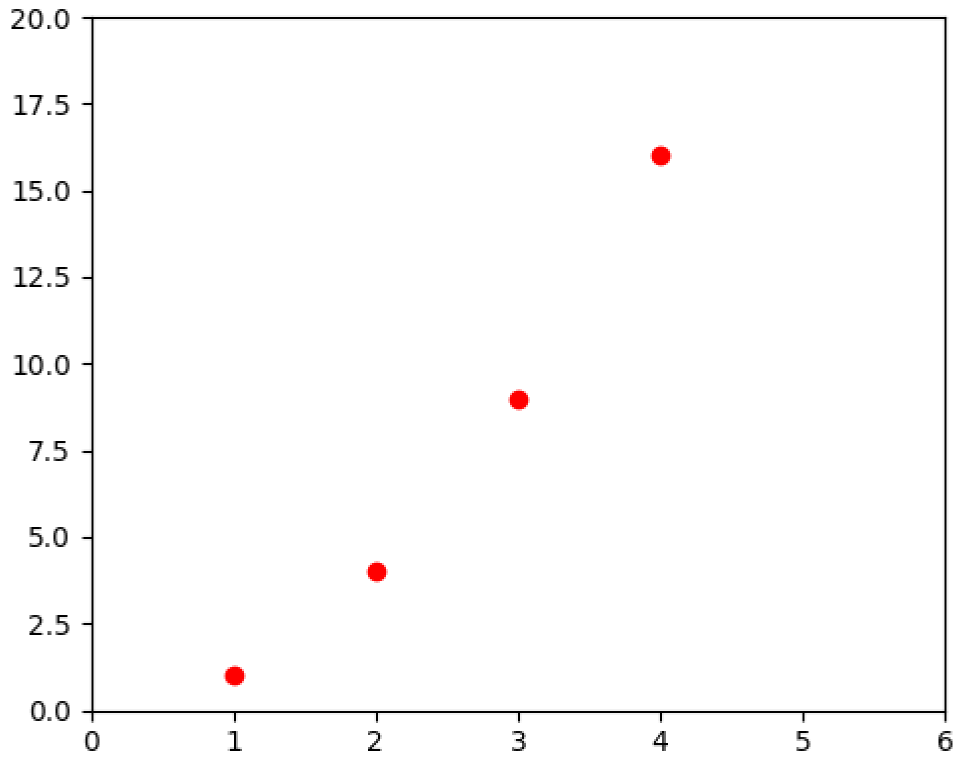
plot() is a versatile command, and will take an arbitrary number of arguments. For example, to plot x versus y, you can issue the command:

```
plt.plot([1,2,3,4],[1,4,9,16])
```

For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot. The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string. The default format string is 'b-', which is a solid blue line. For example, to plot the above with red circles, you would issue

```
importmatplotlib.pyplotasplt
plt.plot([1,2,3,4],[1,4,9,16],'ro')
plt.axis([0,6,0,20])
```

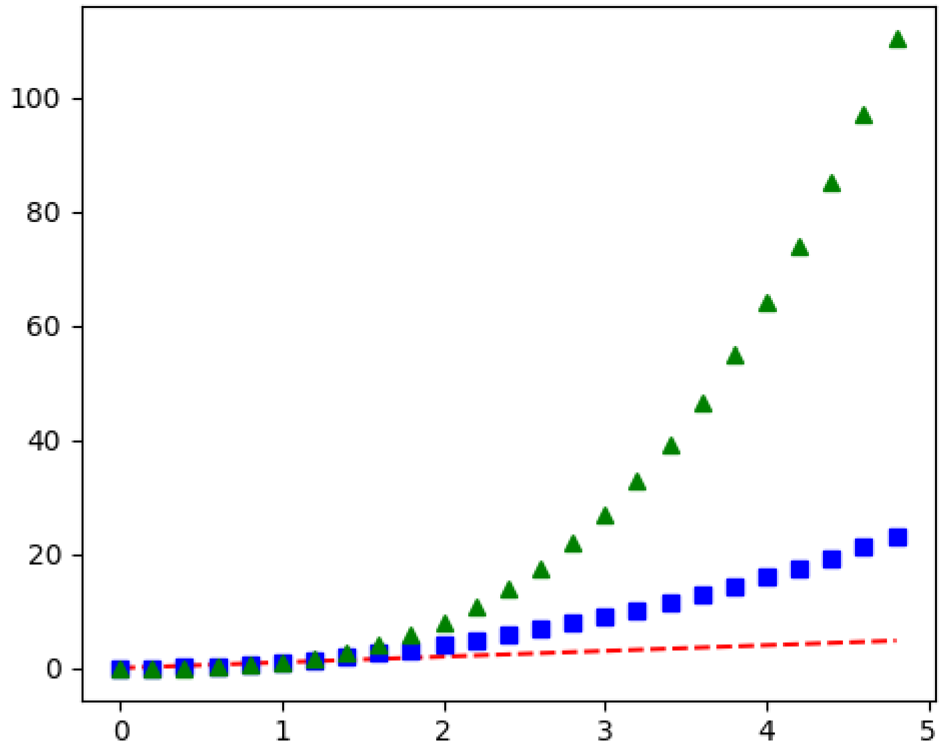
```
plt.show()
```



See the plot() documentation for a complete list of line styles and format strings. The axis() command in the example above takes a list of [xmin, xmax, ymin, ymax] and specifies the viewport of the axes.

If matplotlib were limited to working with lists, it would be fairly useless for numeric processing. Generally, you will use numpy arrays. In fact, all sequences are converted to numpy arrays internally. The example below illustrates a plotting several lines with different format styles in one command using arrays.

```
importnumpyasnp
importmatplotlib.pyplotasplt

# evenly sampled time at 200ms intervals
t=np.arange(0.,5.,0.2)

# red dashes, blue squares and green triangles
plt.plot(t,t,'r--',t,t**2,'bs',t,t**3,'g^')
plt.show()
```



Controlling line properties

Lines have many attributes that you can set: linewidth, dash style, antialiased, etc; see matplotlib.lines.Line2D. There are several ways to set line properties

- Use keyword args:

```
plt.plot(x,y,linewidth=2.0)
```

- Use the setter methods of a Line2D instance. plot returns a list of Line2D objects; e.g., line1, line2 = plot(x1, y1, x2, y2). In the code below we will suppose that we have only one line so that the list returned is of length 1. We use tuple unpacking with line, to get the first element of that list:

```
line,=plt.plot(x,y,'-')
line.set_antialiased(False)# turn off antialising
```

- Use the setp() command. The example below uses a MATLAB-style command to set multiple properties on a list of lines. setp works transparently with a list of objects or a single object. You can either use python keyword arguments or MATLAB-style string/value pairs:

- lines=plt.plot(x1,y1,x2,y2)
- *# use keyword args*
- plt.setp(lines,color='r',linewidth=2.0)
- *# or MATLAB style string value pairs*
- plt.setp(lines,'color','r','linewidth',2.0)

Here are the available Line2D properties.

Property	**Value Type**
alpha	Float
animated	[True | False]
antialiased or aa	[True | False]
clip_box	a matplotlib.transform.Bbox instance
clip_on	[True | False]
clip_path	a Path instance and a Transform instance, a Patch
color or c	any matplotlib color
contains	the hit testing function
dash_capstyle	['butt' | 'round' | 'projecting']
dash_joinstyle	['miter' | 'round' | 'bevel']
dashes	sequence of on/off ink in points
data	(np.arrayxdata, np.arrayydata)
figure	a matplotlib.figure.Figure instance
label	any string
linestyle or ls	['-' | '--' | '-.' | ':' | 'steps' | ...]
linewidth or lw	float value in points
lod	[True | False]

<table>
<tr><th>Property</th><th>Value Type</th></tr>
<tr><td>marker</td><td>['+' | ',' | '.' | '1' | '2' | '3' | '4']</td></tr>
<tr><td>markeredgecolor or mec</td><td>any matplotlib color</td></tr>
<tr><td>markeredgewidth or mew</td><td>float value in points</td></tr>
<tr><td>markerfacecolor or mfc</td><td>any matplotlib color</td></tr>
<tr><td>markersize or ms</td><td>Float</td></tr>
<tr><td>markevery</td><td>[None | integer | (startind, stride)]</td></tr>
<tr><td>picker</td><td>used in interactive line selection</td></tr>
<tr><td>pickradius</td><td>the line pick selection radius</td></tr>
<tr><td>solid_capstyle</td><td>['butt' | 'round' | 'projecting']</td></tr>
<tr><td>solid_joinstyle</td><td>['miter' | 'round' | 'bevel']</td></tr>
<tr><td>transform</td><td>a matplotlib.transforms.Transform instance</td></tr>
<tr><td>visible</td><td>[True | False]</td></tr>
<tr><td>xdata</td><td>np.array</td></tr>
<tr><td>ydata</td><td>np.array</td></tr>
<tr><td>zorder</td><td>any number</td></tr>
</table>

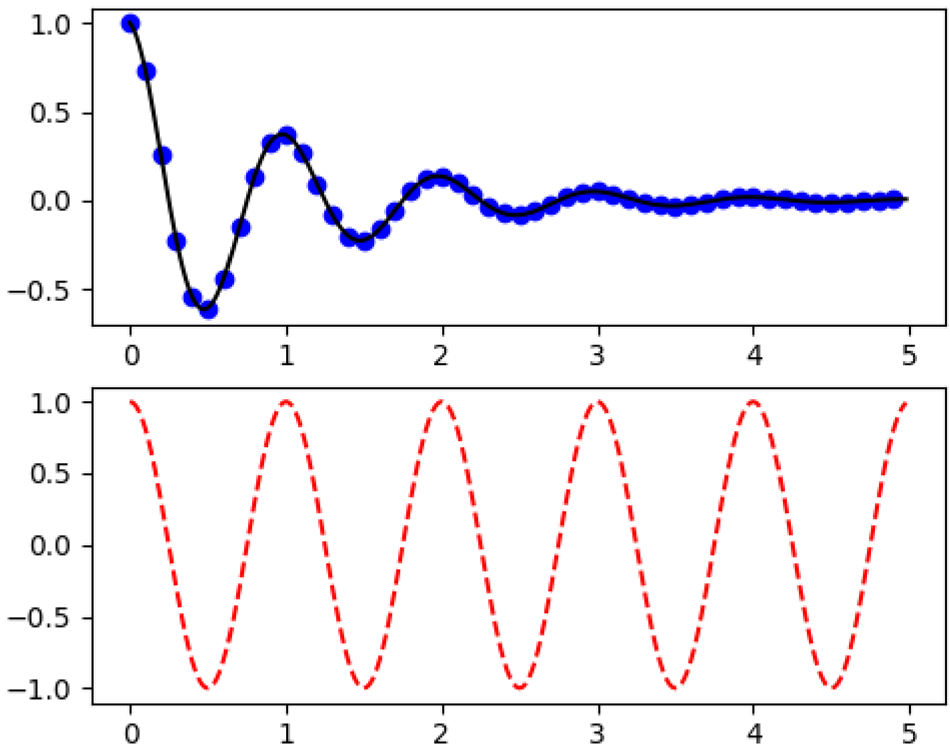
To get a list of settable line properties, call the setp() function with a line or lines as argument

```
In [69]: lines=plt.plot([1,2,3])

In [70]: plt.setp(lines)
  alpha: float
  animated: [True | False]
antialiased or aa: [True | False]
  ...snip
```

Working with multiple figures and axes

MATLAB, and pyplot, have the concept of the current figure and the current axes. All plotting commands apply to the current axes. The function gca() returns the current axes (a matplotlib.axes.Axes instance), and gcf() returns the current figure (matplotlib.figure.Figure instance). Normally, you don’t have to worry about this, because it is all taken care of behind the scenes. Below is a script to create two subplots.

```
importnumpyasnp
importmatplotlib.pyplotasplt

deff(t):
returnnp.exp(-t)*np.cos(2*np.pi*t)

t1=np.arange(0.0,5.0,0.1)
t2=np.arange(0.0,5.0,0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1,f(t1),'bo',t2,f(t2),'k')

plt.subplot(212)
plt.plot(t2,np.cos(2*np.pi*t2),'r--')
plt.show()
```



The figure() command here is optional because figure(1) will be created by default, just as a subplot(111) will be created by default if you don't manually specify any axes. The subplot() command specifies numrows, numcols, fignum where fignum ranges from 1 to numrows*numcols. The commas in the subplot command are optional if numrows*numcols<10. So subplot(211) is identical to subplot(2, 1, 1). You can create an arbitrary number of subplots and axes. If you want to place an axes manually, i.e., not on a rectangular grid, use the axes() command, which allows you to specify the location as axes([left, bottom, width, height]) where all values are in fractional (0 to 1) coordinates. See pylab_examples example code: axes_demo.py for an example of placing axes manually and pylab_examples example code: subplots_demo.py for an example with lots of subplots.

You can create multiple figures by using multiple figure() calls with an increasing figure number. Of course, each figure can contain as many axes and subplots as your heart desires:

```
importmatplotlib.pyplotasplt
plt.figure(1)# the first figure
plt.subplot(211)# the first subplot in the first figure
plt.plot([1,2,3])
plt.subplot(212)# the second subplot in the first figure
plt.plot([4,5,6])


plt.figure(2)# a second figure
plt.plot([4,5,6])# creates a subplot(111) by default

plt.figure(1)# figure 1 current; subplot(212) still current
plt.subplot(211)# make subplot(211) in figure1 current
plt.title('Easy as 1, 2, 3')# subplot 211 title
```

You can clear the current figure with clf() and the current axes with cla().

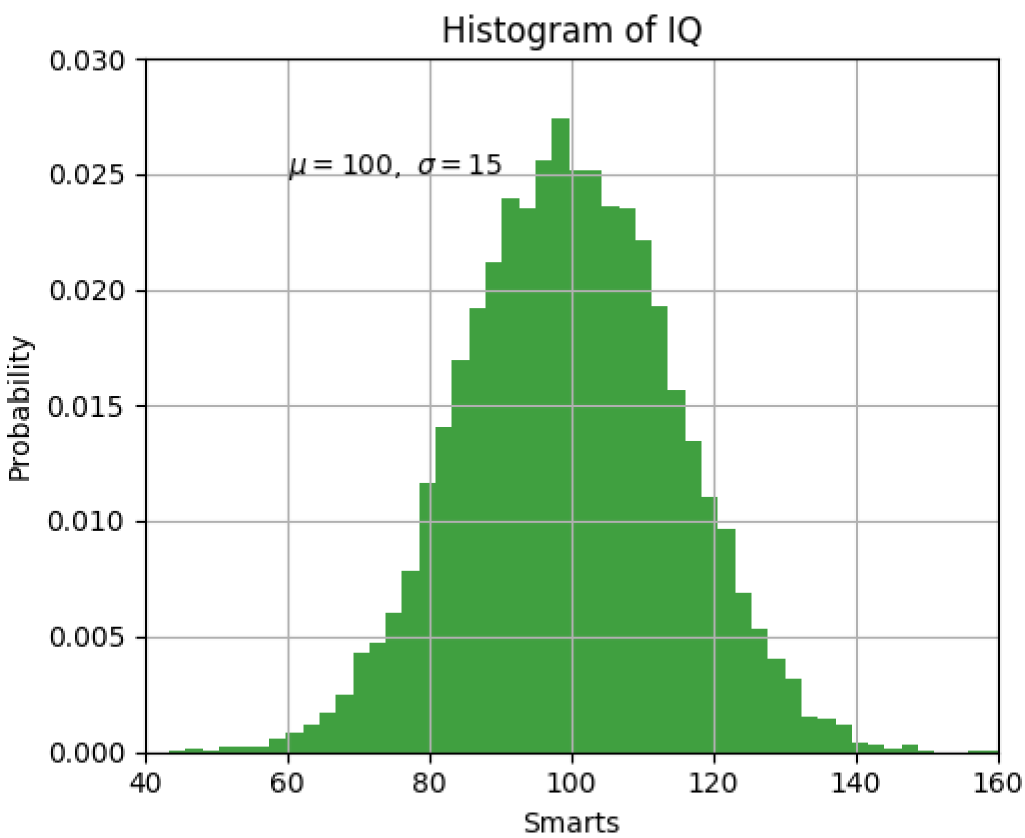
If you are making lots of figures, you need to be aware of one more thing: the memory required for a figure is not completely released until the figure is explicitly closed with close(). Deleting all references to the figure, and/or using the window manager to kill the window in which the figure appears on the screen, is not enough, because pyplot maintains internal references until close() is called.

Working with text

The text() command can be used to add text in an arbitrary location, and the xlabel(), ylabel() and title() are used to add text in the indicated locations

```
importnumpyasnp
importmatplotlib.pyplotasplt

# Fixing random state for reproducibility
np.random.seed(19680801)

mu,sigma=100,15
x=mu+sigma*np.random.randn(10000)

# the histogram of the data
n,bins,patches=plt.hist(x,50,normed=1,facecolor='g',alpha=0.75)


plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title('Histogram of IQ')
plt.text(60,.025,r'$\mu=100,\ \sigma=15$')
plt.axis([40,160,0,0.03])
plt.grid(True)
plt.show()
```

(Source code, png, pdf)



All of the text() commands return an matplotlib.text.Text instance. Just as with with lines above, you can customize the properties by passing keyword arguments into the text functions or using setp():

```
t=plt.xlabel('my data',fontsize=14,color='red')
```

Using mathematical expressions in text

matplotlib accepts TeX equation expressions in any text expression. For example to write the expression $\sigma_i = 15$ in the title, you can write a TeX expression surrounded by dollar signs:

```
plt.title(r'$\sigma_i=15$')
```

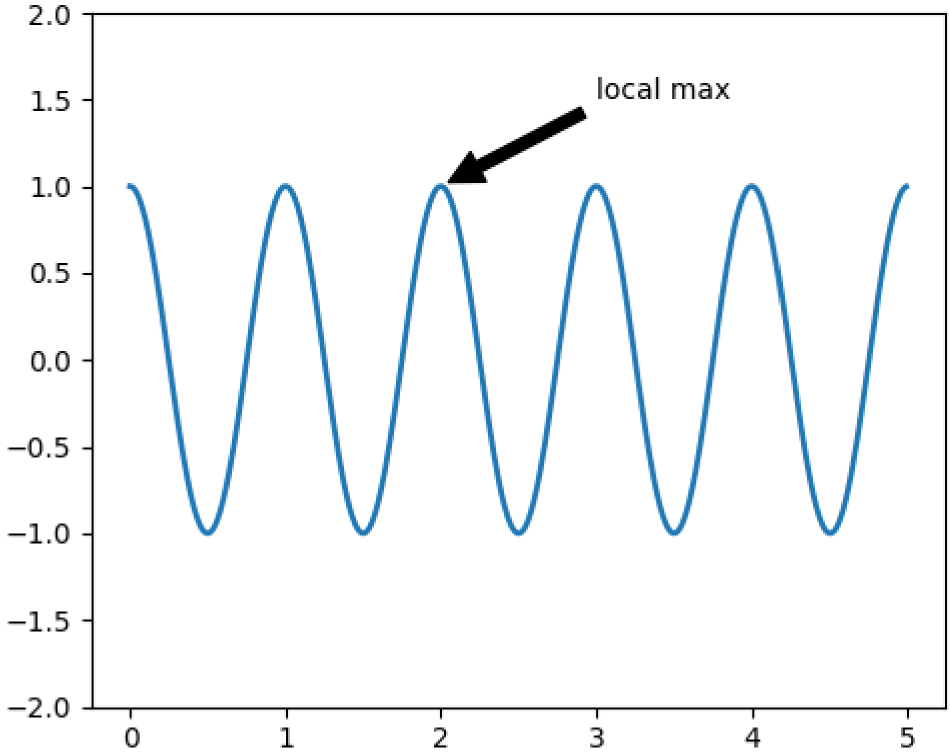
The r preceding the title string is important – it signifies that the string is a *raw* string and not to treat backslashes as python escapes. matplotlib has a built-in TeX expression parser and layout engine, and ships its own math fonts – for details see Writing mathematical expressions. Thus you can use mathematical text across platforms without requiring a TeX installation. For those who have LaTeX and dvipng installed, you can also use LaTeX to format your text and incorporate the output directly into your display figures or saved postscript – see Text rendering With LaTeX.

Annotating text

The uses of the basic text() command above place text at an arbitrary position on the Axes. A common use for text is to annotate some feature of the plot, and the annotate() method provides helper functionality to make annotations easy. In an annotation, there are two points to consider: the location being annotated represented by the argument xy and the location of the text xytext. Both of these arguments are (x,y) tuples.

```
importnumpyasnp
importmatplotlib.pyplotasplt

ax=plt.subplot(111)

t=np.arange(0.0,5.0,0.01)
s=np.cos(2*np.pi*t)
line,=plt.plot(t,s,lw=2)

plt.annotate('local max',xy=(2,1),xytext=(3,1.5),
arrowprops=dict(facecolor='black',shrink=0.05),
)
```

```
plt.ylim(-2,2)
plt.show()
```



In this basic example, both the xy (arrow tip) and xytext locations (text location) are in data coordinates.

Logarithmic and other nonlinear axes

matplotlib.pyplot supports not only linear axis scales, but also logarithmic and logit scales. This is commonly used if data spans many orders of magnitude. Changing the scale of an axis is easy:

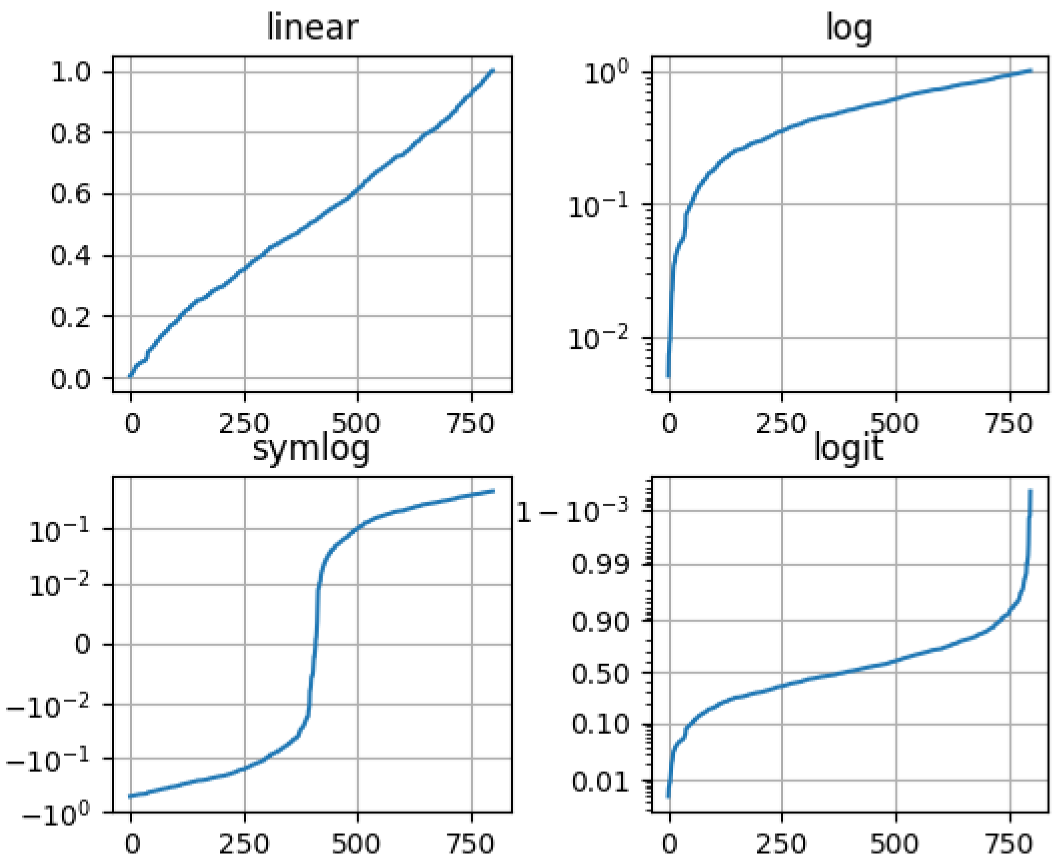
plt.xscale('log')

An example of four plots with the same data and different scales for the y axis is shown below.

```
importnumpyasnp
importmatplotlib.pyplotasplt

frommatplotlib.tickerimportNullFormatter# useful for `logit` scale
```

```
# Fixing random state for reproducibility
np.random.seed(19680801)

# make up some data in the interval ]0, 1[
y=np.random.normal(loc=0.5,scale=0.4,size=1000)
y=y[(y>0)&(y<1)]
y.sort()
x=np.arange(len(y))

# plot with various axes scales
plt.figure(1)

# linear
plt.subplot(221)
plt.plot(x,y)
plt.yscale('linear')
plt.title('linear')
plt.grid(True)


# log
plt.subplot(222)
plt.plot(x,y)
plt.yscale('log')
plt.title('log')
plt.grid(True)


# symmetric log
plt.subplot(223)
plt.plot(x,y-y.mean())
plt.yscale('symlog',linthreshy=0.01)
plt.title('symlog')
plt.grid(True)

# logit
plt.subplot(224)
plt.plot(x,y)
plt.yscale('logit')
plt.title('logit')
plt.grid(True)
# Format the minor tick labels of the y-axis into empty strings with
# `NullFormatter`, to avoid cumbering the axis with too many labels.
plt.gca().yaxis.set_minor_formatter(NullFormatter())
# Adjust the subplot layout, because the logit one may take more space
# than usual, due to y-tick labels like "1 - 10^{-3}"
```

```
plt.subplots_adjust(top=0.92,bottom=0.08,left=0.10,right=0.95,hspace=0.25,
wspace=0.35)

plt.show()
```



What is WordCloud?

Many times you might have seen a cloud filled with lots of words in different sizes, which represent the frequency or the importance of each word. This is called **Tag Cloud** or WordCloud.

How to create a basic wordcloud from one to several text documents

- Adjust color, size and number of text inside your wordcloud
- Mask your wordcloud into any shape of your choice
- Mask your wordcloud into any color pattern of your choice

Prerequisites

You will need to install some packages below:

- **numpy**
- **pandas**
- **matplotlib**
- **pillow**
- **wordcloud**

The numpy library is one of the most popular and helpful libraries that is used for handling multi-dimensional arrays and matrices. It is also used in combination with Pandas library to perform data analysis.

The Python os module is a built-in library, so you don't have to install it.

For visualization, matplotlib is a basic library that enables many other libraries to run and plot on its base including **seaborn** or wordcloud that you will use in this tutorial. The pillow library is a package that enables image reading. Pillow is a wrapper for PIL - Python Imaging Library. You will need this library to read in image as the mask for the wordcloud.

wordcloud can be a little tricky to install. If you only need it for plotting a basic wordcloud, then pip install wordcloud or conda install -c conda-forge wordcloud would be sufficient.

Dataset:

This tutorial uses the **wine review dataset** from **Kaggle**. This collection is a great dataset for learning with no missing values (which will take time to handle) and a lot of text (wine reviews), categorical, and numerical data.

Now let's get started!

First thing first, you load all the necessary libraries:

```
# Start with loading all necessary libraries

importnumpyas np

import pandas as pd
```

```
fromosimport path

from PIL import Image

fromwordcloudimportWordCloud, STOPWORDS,ImageColorGenerator



importmatplotlib.pyplotasplt

%matplotlib inline

import warnings

warnings.filterwarnings("ignore")
```

If you have more than 10 libraries, organize them by sections (such as basic libs, visualization, models, etc.) using comments in the code will make your code clean and easy to follow.
Now, using pandas `read_csv` to load in the dataframe. Notice the use of `index_col=0` meaning we don't read in row name (index) as a separated column.

```
# Load in the dataframe

df=pd.read_csv("data/winemag-data-130k-v2.csv",index_col=0)

# Looking at first 5 rows of the dataset

df.head()
```

	country	description	designation	points	price	province	region_1	region_2	taster_name	taster_twitter_handle	title	variety	winery
0	Italy	Aromas include tropical fruit, broom, brimston...	Vulkà Bianco	87	NaN	Sicily & Sardinia	Etna	NaN	Kerin O'Keefe	@kerinokeefe	Nicosia 2013 Vulkà Bianco (Etna)	White Blend	Nicosia
1	Portugal	This is ripe and fruity, a wine that is smooth...	Avidagos	87	15.0	Douro	NaN	NaN	Roger Voss	@vossroger	Quinta dos Avidagos 2011 Avidagos Red (Douro)	Portuguese Red	Quinta dos Avidagos
2	US	Tart and snappy, the flavors of lime flesh and...	NaN	87	14.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Rainstorm 2013 Pinot Gris (Willamette Valley)	Pinot Gris	Rainstorm
3	US	Pineapple rind, lemon pith and orange blossom ...	Reserve Late Harvest	87	13.0	Michigan	Lake Michigan Shore	NaN	Alexander Peartree	NaN	St. Julian 2013 Reserve Late Harvest Riesling ...	Riesling	St. Julian
4	US	Much like the regular bottling from 2012, this...	Vintner's Reserve Wild Child Block	87	65.0	Oregon	Willamette Valley	Willamette Valley	Paul Gregutt	@paulgwine	Sweet Cheeks 2012 Vintner's Reserve Wild Child...	Pinot Noir	Sweet Cheeks

You can printout some basic information about the dataset using print() combined with .format() to have a nice printout.

```
print("There are {} observations and {} features in this dataset. \n".format(df.shape[0],df.shape[1]))

print("There are {} types of wine in this dataset such as {}... \n".format(len(df.variety.unique()),

", ".join(df.variety.unique()[0:5])))

print("There are {} countries producing wine in this dataset such as {}... \n".format(len(df.country.unique()),

", ".join(df.country.unique()[0:5])))
```

```
There are 129971 observations and 13 features inthis dataset.

There are 708 types of wine inthis dataset such as White Blend, Portuguese Red, Pinot Gris, Riesling, Pinot Noir...
```

There are 44 countries producing wine inthis dataset such as Italy, Portugal,US, Spain, France...

```
df[["country","description","points"]].head()
```

	country	description	Points
0	Italy	Aromas include tropical fruit, broom, brimston...	87
1	Portugal	This is ripe and fruity, a wine that is smooth...	87
2	US	Tart and snappy, the flavors of lime flesh and...	87
3	US	Pineapple rind, lemon pith and orange blossom ...	87

	country	description	Points
4	US	Much like the regular bottling from 2012, this...	87

To make comparisons between groups of a feature, you can use groupby() and compute summary statistics.

With the wine dataset, you can group by country and look at either the summary statistics for all countries' points and price or select the most popular and expensive ones.

```
# Groupby by country

country =df.groupby("country")



# Summary statistic of all countries

country.describe().head()
```

	points								price							
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max
country																
Argentina	3800.0	86.710263	3.179627	80.0	84.00	87.0	89.00	97.0	3756.0	24.510117	23.430122	4.0	12.00	17.0	25.00	230.0
Armenia	2.0	87.500000	0.707107	87.0	87.25	87.5	87.75	88.0	2.0	14.500000	0.707107	14.0	14.25	14.5	14.75	15.0
Australia	2329.0	88.580507	2.989900	80.0	87.00	89.0	91.00	100.0	2294.0	35.437663	49.049458	5.0	15.00	21.0	38.00	850.0
Austria	3345.0	90.101345	2.499799	82.0	88.00	90.0	92.00	98.0	2799.0	30.762772	27.224797	7.0	18.00	25.0	36.50	1100.0
Bosnia and Herzegovina	2.0	86.500000	2.121320	85.0	85.75	86.5	87.25	88.0	2.0	12.500000	0.707107	12.0	12.25	12.5	12.75	13.0

This selects the top 5 highest average points among all 44 countries:

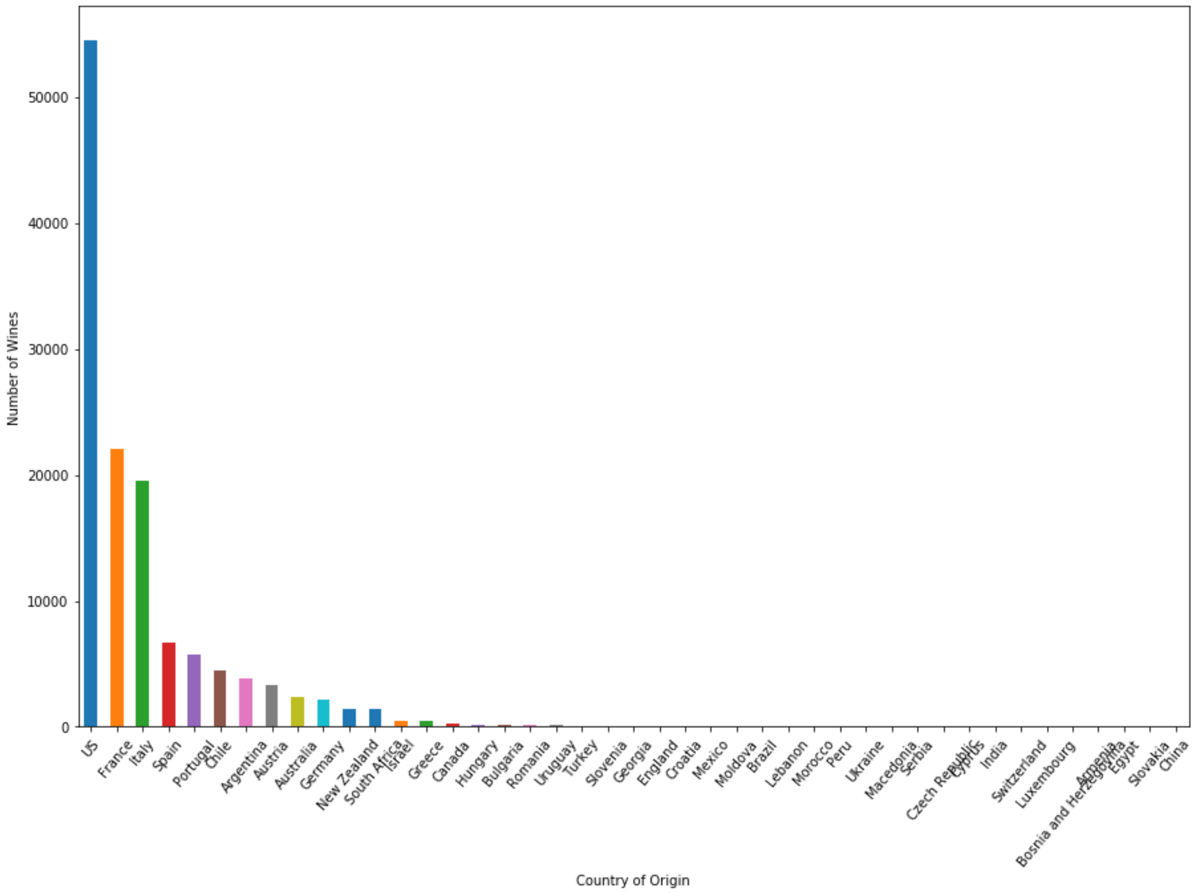
```
country.mean().sort_values(by="points",ascending=False).head()
```

	Points	Price

Country		
England	91.581081	51.681159
India	90.222222	13.333333
Austria	90.101345	30.762772
Germany	89.851732	42.257547
Canada	89.369650	35.712598

You can plot the number of wines by country using the plot method of Pandas DataFrame and Matplotlib.

```
plt.figure(figsize=(15,10))

country.size().sort_values(ascending=False).plot.bar()

plt.xticks(rotation=50)

plt.xlabel("Country of Origin")

plt.ylabel("Number of Wines")

plt.show()
```

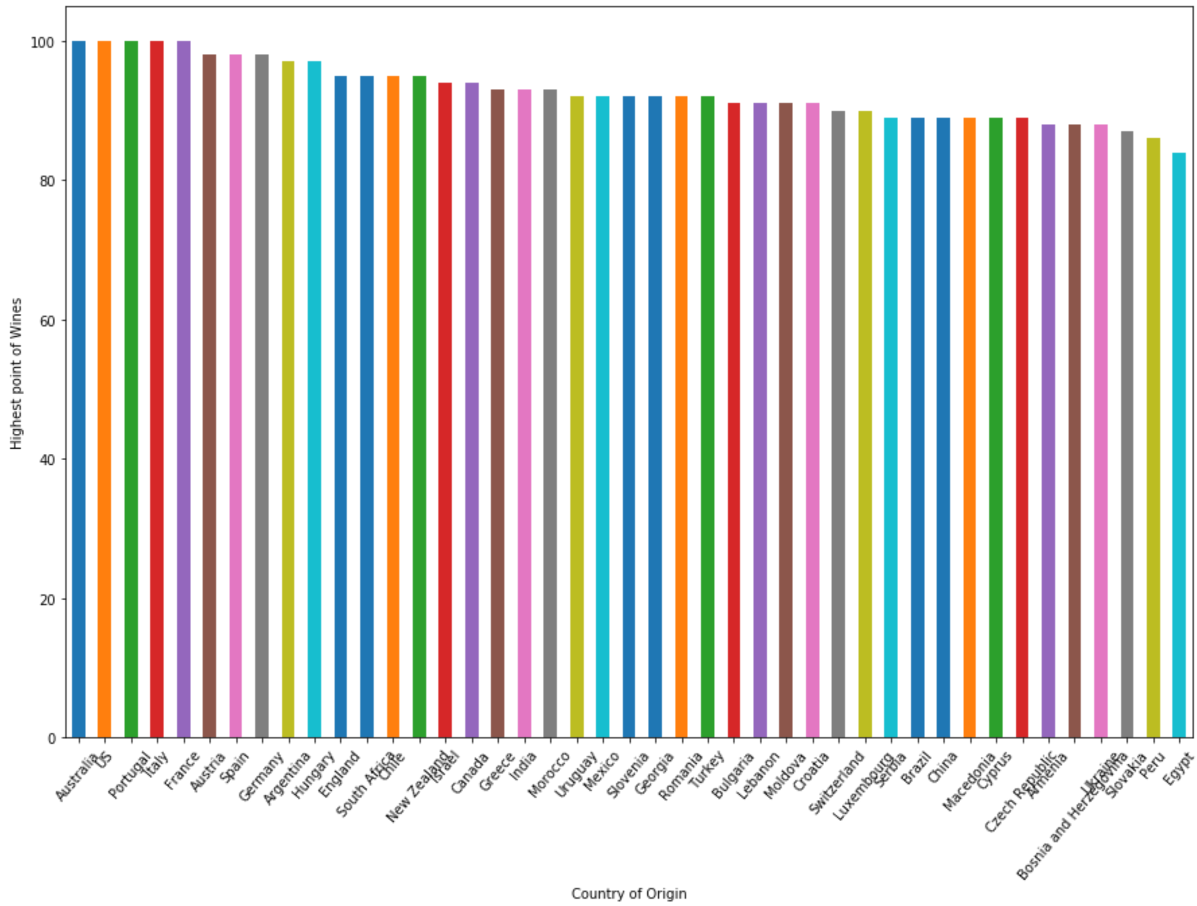


Among 44 countries producing wine, US has more than 50,000 types of wine in the wine review dataset, twice as much as the next one in the rank: France - the country famous for its wine. Italy also produces a lot of quality wine, having nearly 20,000 wines open to review.

Does quantity over quality?

Let's now take a look at the plot of all 44 countries by its highest rated wine, using the same plotting technique as above:

```
plt.figure(figsize=(15,10))

country.max().sort_values(by="points",ascending=False)["points"].plot.bar()

plt.xticks(rotation=50)

plt.xlabel("Country of Origin")

plt.ylabel("Highest point of Wines")

plt.show()
```



Australia, US, Portugal, Italy, and France all have 100 points wine. If you notice, Portugal ranks 5th and Australia ranks 9th in the number of wines produces in the dataset, and both countries have less than 8000 types of wine.

CountVectorizer is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text.

The CountVectorizer will select the words/features/terms which occur the most frequently. It takes absolute values so if you set the 'max_features = 3', it will select the 3 most common words in the data. By setting 'binary = True', the

CountVectorizer no more takes into consideration the frequency of the term/word.

CountVectorizer is a great tool provided by the scikit-learn library in Python. It is used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. This is helpful when we have multiple such texts, and we wish to convert each word in each text into vectors

How do you implement Bernoulli naive Bayes?

$$p(x) = P[X = x] = \begin{cases} q = 1 - p & x = 0 \\ p & x = 1 \end{cases}$$

$$X = \begin{cases} 1 & \text{Bernoulli trial} = \mathbf{S} \\ 0 & \text{Bernoulli trial} = \mathbf{F} \end{cases}$$

To test Bernoulli Naive Bayes on the dataset, we need to **ensure that all values are binary**. So, firstly, we check if our target variable values are binary or not.

What is Bernoullinb?

The Bernoulli Naive Bayes is one of the variations of the Naive Bayes algorithm in machine learning and it is very useful to use in a binary distribution where the output label may be present or absent.

Bernoulli Naive Bayes

Bernoulli Naive Bayes is one of the variants of the Naive Bayes algorithm in machine learning. It is very useful to be used when the dataset is in a binary distribution where the output label is present or absent. The main advantage of this algorithm is that it only accepts features in the form of binary values such as:

1. True or False
2. Spam or Ham
3. Yes or No
4. 0 or 1

Here are some other advantages of using this algorithm for binary classification:

1. It is very fast compared to other classification algorithms.

2. Sometimes machine learning algorithms do not work well if the dataset is small, but this is not the case with this algorithm because it gives more accurate results compared to other classification algorithms in the case of a small dataset.
3. It's fast and can also handle irrelevant features easily.

Bernoulli Naive Bayes using Python

To implement this algorithm using Python, I will be using the scikit-learn library. I will first start by importing the necessary Python libraries and the dataset that we need to implement this algorithm:

```
importpandasaspd

importnumpyasnp

fromsklearn.feature_extraction.textimportCountVectorizer

fromsklearn.model_selectionimporttrain_test_split

fromsklearn.naive_bayesimportBernoulliNB



data=pd.read_csv("spam.csv", encoding='latin-1')

data=data[["class", "message"]]
```

This algorithm expects binary feature vectors although the BernoulliNB class from the scikit-learn library has a binarize parameter that allows us to specify a threshold value that will be used to transform the features. So here is how to implement this algorithm using Python:

```
x=np.array(data["message"])

y=np.array(data["class"])



cv=CountVectorizer()

x=cv.fit_transform(x)

xtrain, xtest, ytrain, ytest=train_test_split(x, y, test_size=0.33, random_state=42)
```

```
model=BernoulliNB(binarize=0.0)

model.fit(xtrain, ytrain)

print(model.score(xtest, ytest))
```

0.9782490483958673

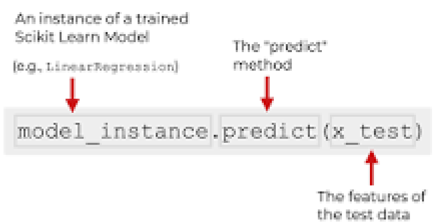
Summary

Bernoulli Naive Bayes is one of the variants of the Naive Bayes algorithm in machine learning. It is very useful to be used when the dataset is in a binary distribution where the output label is either present or absent.

What does model fit do in sklearn?

The fit() method **takes the training data as arguments**, which can be one array in the case of unsupervised learning, or two arrays in the case of supervised learning. Note that the model is fitted using X and y , but the object holds no reference to X and y

What is predict () sklearn?



The Sklearn 'Predict' Method **Predicts an Output**

That being the case, it provides a set of tools for doing things like training and evaluating machine learning models. And it also has tools to predict an output value, once the model is trained (for ML techniques that actually make predictions).