



Machine Learning Certification Training Report

On

## **SPOTIFY SONG RECOMMENDATION SYSTEM**

Submitted by

**RAMESH KUMAR CHOUHAN**

**PREM KUMAR PAUL**

**ARBIND KUMAR**

**DEBOLINA DAS**

**ANWESHA DEY**

Under the Guidance of

**INDRANIL DAS**

Department of Computer Science & Engineering and Information Technology

ASANSOL ENGINEERING COLLEGE, ASANSOL

## DECLARATION

I hereby declare that I have completed my four weeks summer training at webskitters(one of the India's leading online certification training providers) from 5th april, 2021 to 10th may, 2021 under the guidance of **INDRANIL DAS**. I have declared that I have worked with full dedication during these four weeks of training and my learning outcomes fulfill the requirements of training for the award of degree of Bachelor of Technology (B.Tech.) in CSE and IT, Asansol Engineering College, Asansol.

.....

(Signature of Student)

Date:.....

# ACKNOWLEDGEMENT

The success and final outcome of learning Machine Learning required a lot of guidance and assistance from many people and I am extremely privileged to have got this all along the completion of my course and few of the projects. All that I have done is only due to such supervision and assistance and I would not forget to thank them.

I respect and thank Webskitters, for providing me an opportunity to do the course and project work and giving me all support and guidance, which made me complete the course duly. I am extremely thankful to the course advisor **Mr. INDRANIL DAS**

I am thankful to and fortunate enough to get constant encouragement, support and guidance from all Teaching staffs of Webskitters which helped us in successfully completing my course and project work.

.....

(Signature of Student)

Date:.....

# Table of contents

## **Python**

- Scope of python
- What can we do with Python
- Who uses Python today
- Why do people use Python
- Python operator
- Tuples
- List
- Loop
- Conditional statements

## **1. Introduction of Machine Learning**

- 1.1. Taste of machine learning
- 1.2. Relation to data mining
- 1.3. Relation to Optimization
- 1.4. Relation to statistics
- 1.5. Future of Machine Learning

## **2. Technology Learnt**

- 2.1. Introduction to AI and Machine Learning
  - 2.1.1 Definition of Artificial Intelligence

2.1.2. Definition of Machine Learning

2.1.3. Machine Learning Algorithms

2.1.4. Applications of machine learning

- **Linear Regression**

- **Logistic Regression**

- **Polynomial Regression**

- **Decision Tree Terminology**

- **Random Forest Algorithms**

- **K-means clustering**

- **Introduction to recommendation system**

- **Why Recommender Systems**

- **Problem statement**

- **Objective**

- **Methodology**

- **Import Libraries**

- **Music over time**

- **Characteristics of different genres**

- **Clustering genres with K-means**

- **Visualising the genre clusters with t-SNE**

- **Clustering Songs with K-means**

- **Visualizing the Song Clusters with PCA**

- **Register your app**

- **Installing Spotipy**

- **Generating song recommendations**

- Conclusion

- Future Work

# PYTHON

## SCOPE OF PYTHON

### 1 - Science

- Bioinformatics

### 2 - System Administration

- Unix
- Web logic
- Web sphere

### 3 - Web Application Development

#### **What Can We do With Python?**

1 - System programming

2 - Graphical User Interface

Programming3 - Internet

Scripting

4 - Component

Integration5 -

Database

Programming

6 - Gaming, Images, XML , Robot and more

## WHO USES PYTHON TODAY?

- Python is being applied in real revenue-generating products by real companies.
- Google makes extensive use of Python in its web search system, and employs Python's creator.
- Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, and IBM use Python for hardware testing.
- ESRI uses Python as an end-user customization tool for its popular GIS mapping products.

## WHY DO PEOPLE USE PYTHON?

- The YouTube video sharing service is largely written in Python.
- Python is object-oriented
  - Structure supports such concepts as polymorphism, operation overloading, and multiple inheritance.
- Indentation
  - Indentation is one of the greatest feature in Python.
- It's free (open source)
  - Downloading and installing Python is free and easy
  - Source code is easily accessible
- It's powerful
  - Dynamic typing
  - Built-in types and tools
  - Library utilities
    - Third party utilities (e.g. Numeric, NumPy, SciPy)
    - Automatic memory management
- It's portable
  - Python runs virtually every major platform used today
  - As long as you have a compatible Python interpreter installed, Python programs will run in exactly the same manner, irrespective of platform.

### **PYTHON:-**

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C++ or Java. The language provides constructs intended to enable clear programs on both a small and large scale. Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library. Python interpreters are available for installation on many operating systems, allowing Python code execution on a wide variety of systems.

### **DATA TYPE:-**

(this is called dynamic typing). Data types determine whether an object can do something, or whether it just would not make sense. Other programming languages often determine whether an operation makes sense for an object by making sure the object can never be stored somewhere where the operation will be performed on the object (this type system is called static typing). Python does not do that. Instead it stores the type of an object with the object, and checks when the operation is performed whether that operation makes sense for that object. Python has many native data types. Here are the important ones: Booleans are either True or False. Numbers can be integers (1 and 2), floats (1.1 and 1.2), fractions (1/2 and 2/3), or even complex numbers. Strings are sequences of Unicode characters, e.g. an HTML document. Bytes and byte arrays, e.g. a JPEG image file. Lists are ordered sequences of values. Tuples are ordered, immutable sequences of values. Sets are unordered bags of values.



**Booleans** are either True or False.

**Numbers** can be integers (1 and 2), floats (1.1 and 1.2), fractions (1/2 and 2/3), or even complex numbers.

**Strings** are sequences of Unicode characters, e.g. an HTML document.

**Bytes and byte arrays**, e.g. a JPEG image file.

**Lists** are ordered sequences of values.

**Tuples** are ordered, immutable sequences of values.

**Sets** are unordered bags of values.

### **Variables:-**

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Ex: counter = 100 # An integer

assignment miles = 1000.0 # A floating

point name = "John" # A string

### **String :-**

"hello"+"world" "helloworld" # concatenation

"hello"\*3 "hellohellohello" # repetition

"hello"[0] "h" # indexing

"hello"[-1] "o" # (from end)

"hello"[1:4] "ell" # slicing

len("hello") 5 # size

"hello" < "jello" 1 # comparison

"e" in "hello" 1 # search

## **Python Operator**

### **Arithmetic Operator**

| Operator | Meaning  | Example                  |
|----------|--|--------------------------|
| +        | Add two operands or unary plus   | x + y<br>+2              |
| -        | Subtract right operand from the left or unary minus  | x - y<br>-2              |
| *        | Multiply two operands  | x * y                    |
| /        | Divide left operand by the right one (always results into float)                                 | x / y                    |
| %        | Modulus - remainder of the division of left operand by the right                                 | x % y (remainder of x/y) |
| //       | Floor division - division that results into whole number adjusted to the left in the number line | x // y                   |
| **       | Exponent - left operand raised to the power of right   | x**y (x to the power y)  |

### Comparison Operator

|   |   |       |
|---|---|-------|
| > | Greater than - True if left operand is greater than the right | x > y |
|---|---|-------|

|    |   |          |
|----|---|----------|
| <  | Less than - True if left operand is less than the right                               | $x < y$  |
| == | Equal to - True if both operands are equal  | $x == y$ |
| != | Not equal to - True if operands are not equal   | $x != y$ |
| >= | Greater than or equal to - True if left operand is greater than or equal to the right | $x >= y$ |
| <= | Less than or equal to - True if left operand is less than or equal to the right       | $x <= y$ |

# Tuples

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses.

## Accessing Values in Tuples:

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example – `tup1 = ('physics', 'chemistry', 1997, 2000); tup2 = (1, 2, 3, 4, 5, 6, 7 ); print "tup1[0]: ", tup1[0] print "tup2[1:5]: ", tup2[1:5]`

When the above code is executed, it produces the following result – `tup1[0]:`

`physicstup2[1:5]: [2, 3, 4, 5]`

## Basic Tuples Operations:

Tuples respond to the + and \* operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string. In fact, tuples respond to all of the general sequence operations we used on strings in the prior chapter –

| Python Expression                         | Results                                   | Description   |
|---|---|---------------|
| <code>len((1, 2, 3))</code>               | 3   | Length        |
| <code>(1, 2, 3) + (4, 5, 6)</code>        | <code>(1, 2, 3, 4, 5, 6)</code>           | Concatenation |
| <code>('Hi!') * 4</code>                  | <code>('Hi!', 'Hi!', 'Hi!', 'Hi!')</code> | Repetition    |
| <code>3 in (1, 2, 3)</code>               | True                                      | Membership    |
| <code>for x in (1, 2, 3): print x,</code> | 1 2 3                                     | Iteration     |

## Built-in Tuple Functions:

Python includes the following tuple functions –

| SN | Function with Description   |
|----|---|
| 1  | <a href="#"><u>cmp(tuple1, tuple2)</u></a> Compares elements of both tuples.  |
| 2  | <a href="#"><u>len(tuple)</u></a> Gives the total length of the tuple.        |
| 3  | <a href="#"><u>max(tuple)</u></a> Returns item from the tuple with max value. |
| 4  | <a href="#"><u>min(tuple)</u></a> Returns item from the tuple with min value. |
| 5  | <a href="#"><u>tuple(seq)</u></a> Converts a list into tuple.                 |

## List

The list is a most versatile datatype available in Python which can be written as a list of comma- separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between squarebrackets. For example – list1 = ['physics', 'chemistry', 1997, 2000]; list2 = [1, 2, 3, 4, 5 ]; list3 = ["a", "b", "c", "d"];

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

### Accessing Values in Lists:

To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example – list1 = ['physics', 'chemistry', 1997, 2000]; list2 = [1, 2,3, 4, 5, 6, 7 ]; print "list1[0]: ", list1[0] print "list2[1:5]: ", list2[1:5]

**Output:** list1[0]: physics

list2[1:5]: [2, 3, 4, 5]

**Update:** list = ['physics', 'chemistry', 1997, 2000];  
print "Value available at index 2 : " print list[2] list[2] =  
2001; print "New value available at index 2 : " print  
list[2]

**Output:** Value available at index 2 :  
1997New value available at index 2 :  
2001

**Delete:** list1 = ['physics', 'chemistry', 1997, 2000]; print  
list1 del list1[2]; print "After deleting value at index 2 : "  
print list1

['physics', 'chemistry', 1997, 2000]

**Output:** After deleting value at index  
2 :['physics', 'chemistry', 2000]

Basic List Operation

| Python Expression            | Results                      | Description   |
|------------------------------|------------------------------|---------------|
| len([1, 2, 3])               | 3                            | Length        |
| [1, 2, 3] + [4, 5, 6]        | [1, 2, 3, 4, 5, 6]           | Concatenation |
| ['Hi!'] * 4                  | ['Hi!', 'Hi!', 'Hi!', 'Hi!'] | Repetition    |
| 3 in [1, 2, 3]               | True                         | Membership    |
| for x in [1, 2, 3]: print x, | 1 2 3                        | Iteration     |

### Built-in List Functions & Methods:

| SN | Function with Description   |
|----|---|
| 1  | <a href="#"><u>cmp(list1, list2)</u></a> Compares elements of both lists.   |
| 2  | <a href="#"><u>len(list)</u></a> Gives the total length of the list.        |
| 3  | <a href="#"><u>max(list)</u></a> Returns item from the list with max value. |
| 4  | <a href="#"><u>min(list)</u></a> Returns item from the list with min value. |
| 5  | <a href="#"><u>list(seq)</u></a> Converts a tuple into list.                |

Python includes following list methods

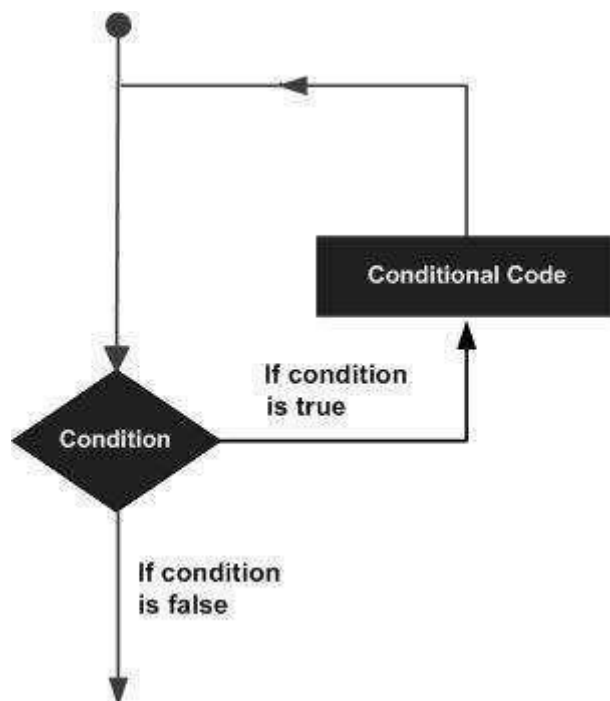
| SN | Methods with Description   |
|----|--|
| 1  | <a href="#"><u>list.append(obj)</u></a> Appends object obj to list                             |
| 2  | <a href="#"><u>list.count(obj)</u></a> Returns count of how many times obj occurs in list      |
| 3  | <a href="#"><u>list.extend(seq)</u></a> Appends the contents of seq to list                    |
| 4  | <a href="#"><u>list.index(obj)</u></a> Returns the lowest index in list that obj appears       |
| 5  | <a href="#"><u>list.insert(index, obj)</u></a> Inserts object obj into list at offset index    |
| 6  | <a href="#"><u>list.pop(obj=list[-1])</u></a> Removes and returns last object or obj from list |

|   |   |
|---|---|
| 7 | <a href="#"><u>list.remove(obj)</u></a> Removes object obj from list                      |
| 8 | <a href="#"><u>list.reverse()</u></a> Reverses objects of list in place                   |
| 9 | <a href="#"><u>list.sort([func])</u></a> Sorts objects of list, use compare func if given |

### Loop definition

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times. The following diagram illustrates a loop statement –



Python programming language provides following types of loops to handle looping requirements.

| Loop Type | Description |
|-----------|-------------|
|           |             |



|                                     |  |
|-------------------------------------|--|
| <a href="#"><u>while loop</u></a>   | Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body. |
| <a href="#"><u>for loop</u></a>     | Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.                          |
| <a href="#"><u>nested loops</u></a> | You can use one or more loop inside any another while, for or do..while loop.  |

#### *Loop Example:*

For Loop:

```
>>> for mynum in [1, 2, 3, 4,
5]:print ("Hello", mynum )
```

Hello 1

Hello 2

Hello 3

Hello 4

Hello 5

While Loop:

```
>>> count = 0 >>while(count< 4):
```

```
print 'The count is:', count
count = count + 1
```

The count is: 0

The count is:

1 The count

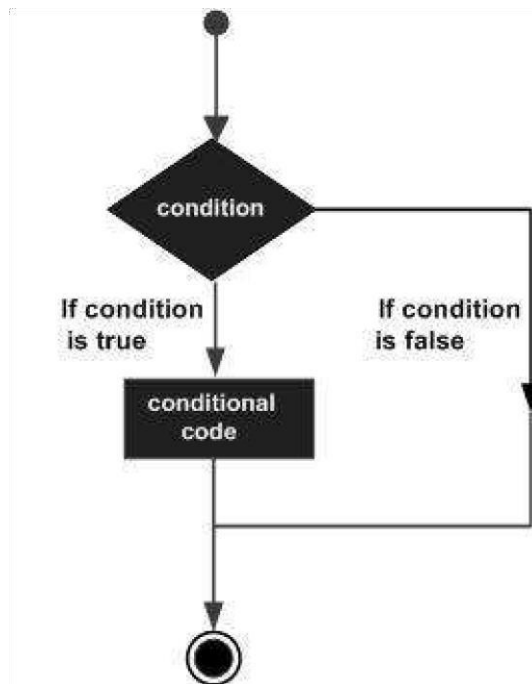
is: 2 The

count is: 3

## Conditional Statements:

Decision making is anticipation of conditions occurring while execution of the program and specifying actions taken according to the conditions.

Decision structures evaluate multiple expressions which produce TRUE or FALSE as outcome. You need to determine which action to take and which statements to execute if outcome is TRUE or FALSE otherwise.



Python programming language provides following types of decision making statements. Click the following links to check their detail.

| Statement                            | Description  |
|--------------------------------------|--|
| <a href="#">if statements</a>        | An <b>if statement</b> consists of a boolean expression followed by one or more statements.  |
| <a href="#">if...else statements</a> | An <b>if statement</b> can be followed by an optional <b>else statement</b> , which executes when the boolean expression is FALSE. |
| <a href="#">nested if statements</a> | You can use one <b>if</b> or <b>else if</b> statement inside another <b>if</b> or <b>else if</b> statement(s).                     |

Example:

If Statement:

a=

33

b=

20

0

If b>a:

print("b")

If...Else Statement:

a=

20

0

b=

33

if b>a:

print("b is greater than

a")else:

print("a is greater than b")

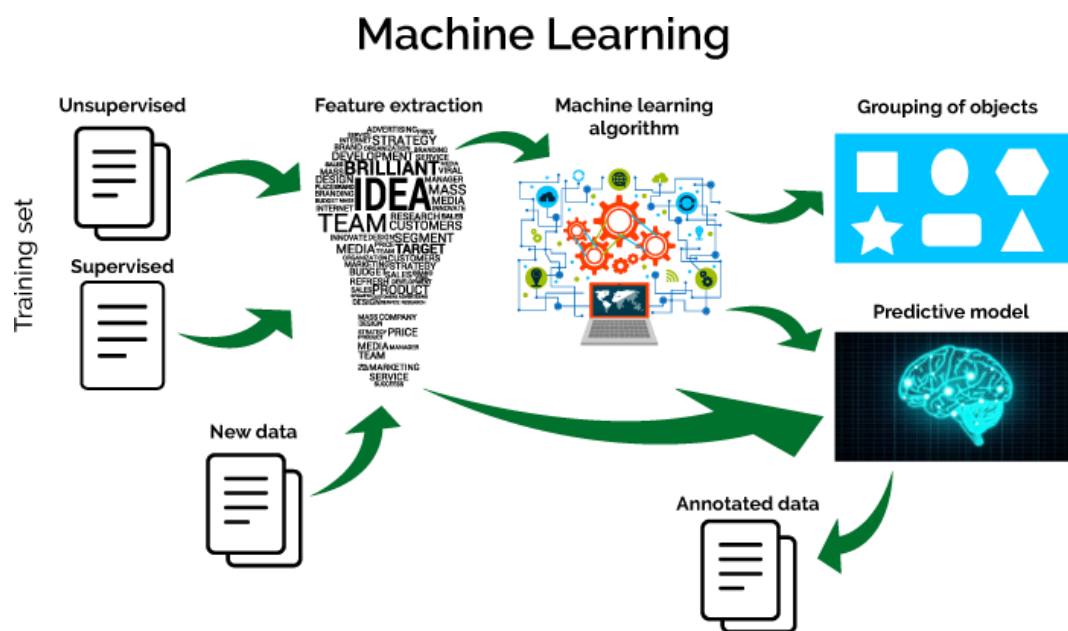
# Introduction of Machine Learning

## 1.1.A Taste of Machine Learning

- ✓ Arthur Samuel, an American pioneer in the field of computer gaming and artificial intelligence, coined the term "Machine Learning" in 1959.
- ✓ Over the past two decades Machine Learning has become one of the mainstays of information technology.

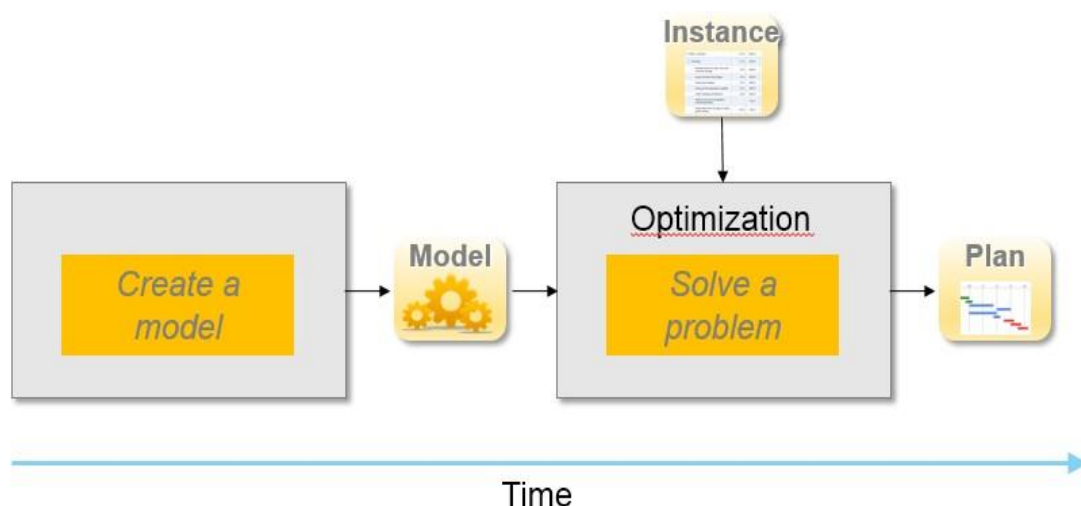
With the ever-increasing amounts of data becoming available there is good reason to believe that smart data analysis will become even more pervasive as a necessary ingredient for technological progress

## 1.2. Relation to Data Mining



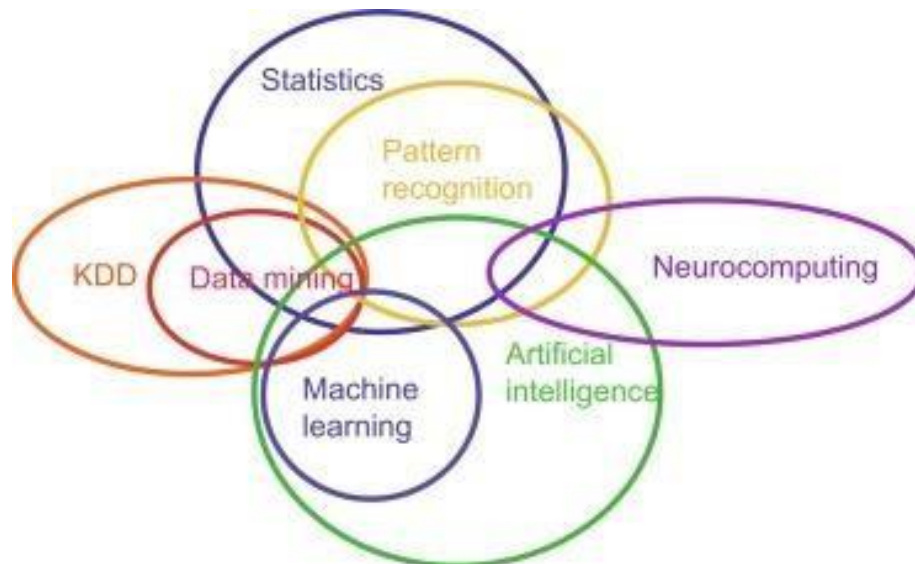
- Data mining uses many machine learning methods, but with different goals; on the other hand, machine learning also employs data mining methods as "unsupervised learning" or as a preprocessing step to improve learner accuracy.

## 1.3. Relation to Optimization



- Machine learning also has intimate ties to optimization: many learning problems are formulated as minimization of some loss function on a training set of examples.
- Loss functions express the discrepancy between the predictions of the model being trained and the actual problem instances.

## 1.4. Relation to Statistics



- Michael I. Jordan suggested the term data science as a placeholder to call the overall field.
- Leo Breiman distinguished two statistical modelling paradigms: data model and algorithmic model, wherein "algorithmic model" means more or less the machine learning algorithms like Random forest.

## 1.5. Future of Machine Learning

- Machine Learning can be a competitive advantage to any company be it a top MNC or a startup as things that are currently being done manually will be done tomorrow by machines.
- Machine Learning revolution will stay with us for long and so will be the future of Machine Learning.

# 1. Technology Learnt

## 1.1. Introduction to AI & Machine Learning

### 1.1.1. Definition of Artificial Intelligence

#### ❖ Data Economy

- ✓ World is witnessing real time flow of all types structured and unstructured data from social media, communication, transportation, sensors, and devices.
- ✓ **International Data Corporation (IDC)** forecasts that 180 zettabytes of data will be generated by 2025.



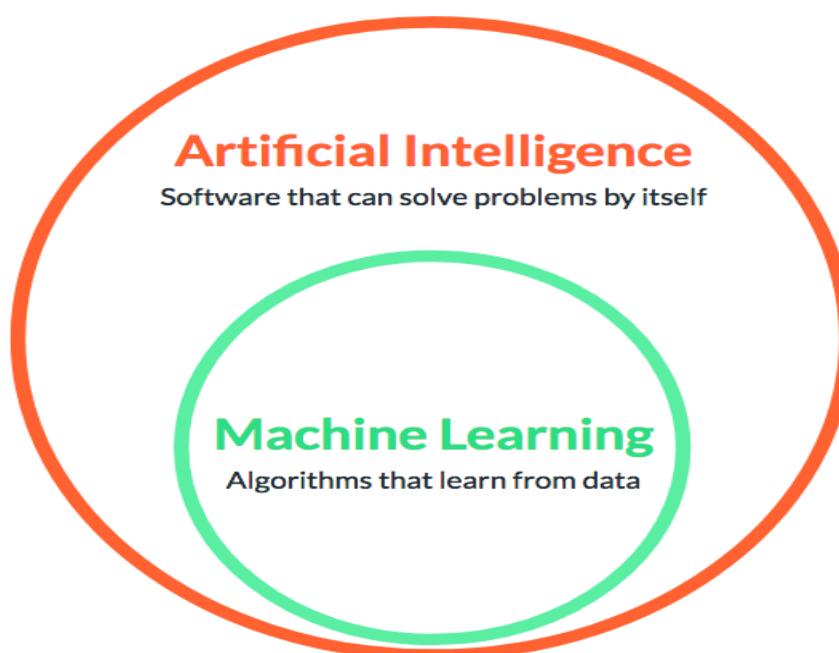
- ✓ This explosion of data has given rise to a new economy known as the **Data Economy**.
- ✓ Data is the new oil that is precious but useful only when cleaned and processed.
- ✓ There is a constant battle for ownership of data between enterprises to derive benefits from it.

❖ *Define Artificial Intelligence*

Artificial intelligence refers to the simulation of human intelligence in machines that are programmed to think like humans and mimic their actions. The term may also be applied to any machine that exhibits traits associated with a human mind such as learning and problem-solving.

### 1.1.2. Definition of Machine Learning

❖ *Relationship between AI and ML*



Machine Learning is an approach or subset of Artificial Intelligence that is based on the idea that machines can be given access to data along with the ability to learn from it.

### ❖ Define Machine Learning

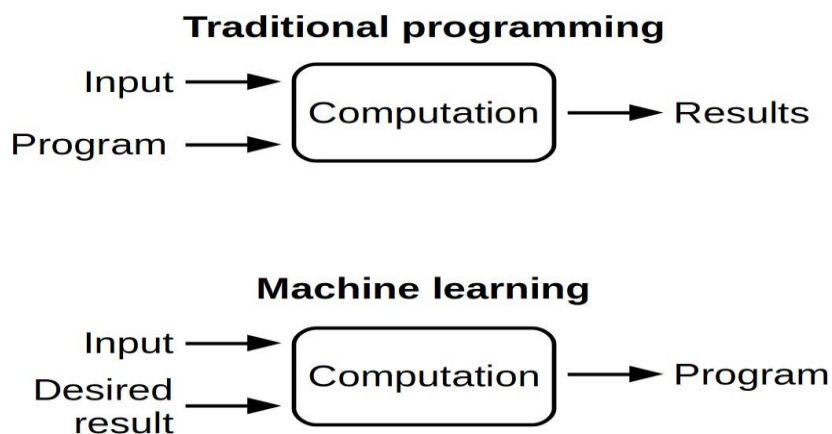
Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

### ❖ Features of Machine Learning

- ✓ Machine Learning is computing-intensive and generally requires a large amount of training data.
- ✓ It involves repetitive training to improve the learning and decision making of algorithms.
- ✓ As more data gets added, Machine Learning training can be automated for learning new data patterns and adapting its algorithm.

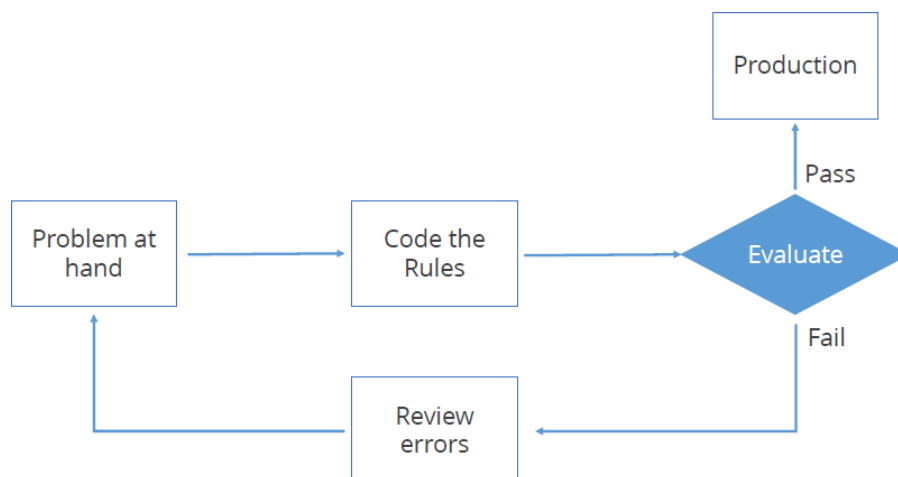
### 1.1.3. Machine Learning Algorithms

### ❖ Traditional Programming vs. Machine Learning Approach



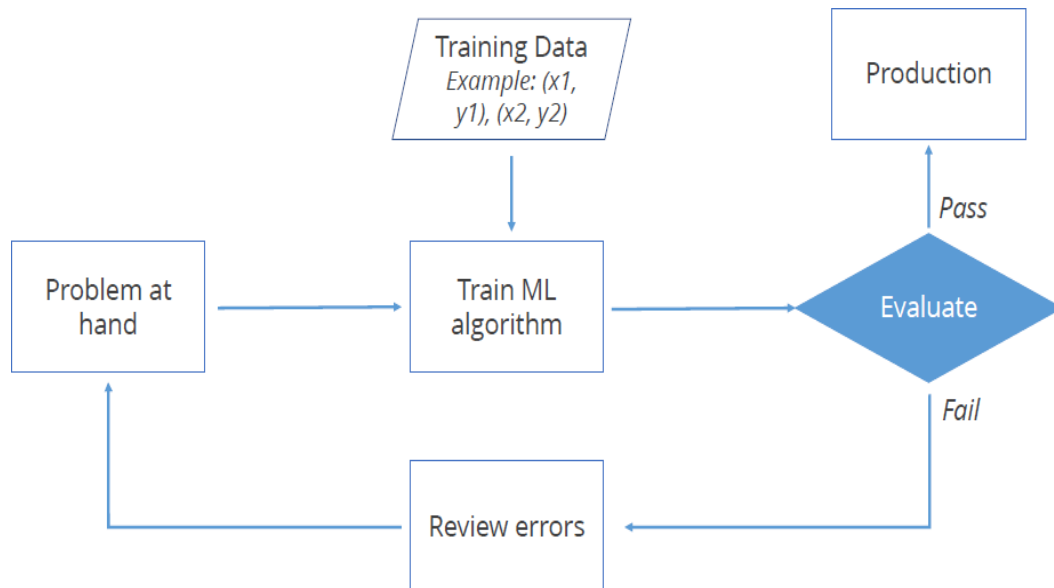
### ❖ Traditional Approach

Traditional programming relies on **hard-coded rules**.



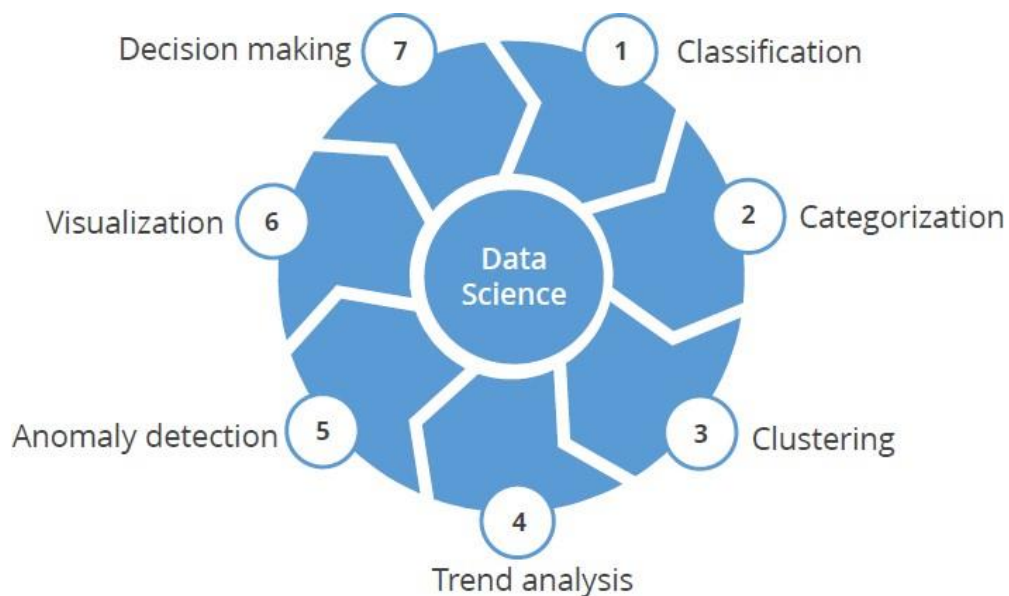
### ❖ Machine Learning Approach

Machine Learning relies on learning patterns based on sample data.



### ❖ Machine Learning Techniques

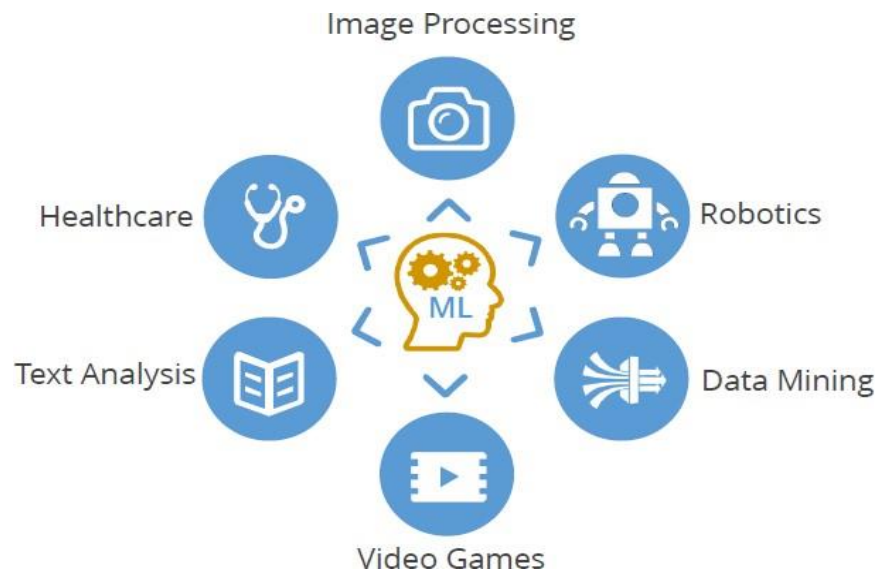
- ✓ Machine Learning uses a number of theories and techniques from Data Science.



- ✓ Machine Learning can learn from **labelled data** (known as supervised learning) or **unlabelled data** (known as unsupervised learning).

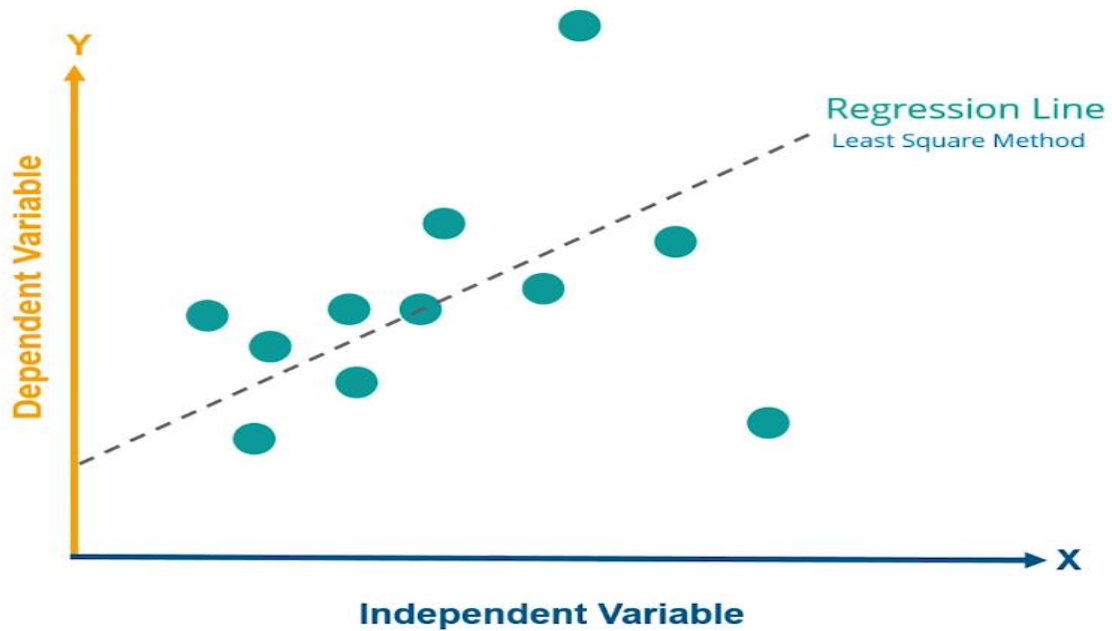
#### 1.1.4. Applications of Machine Learning





- ❖ *Image Processing*
  - ✓ Optical Character Recognition (OCR)
  - ✓ Self-driving cars
  - ✓ Image tagging and recognition
- ❖ *Robotics*
  - ✓ Industrial robotics
  - ✓ Human simulation
- ❖ *Data Mining*
  - ✓ Association rules
  - ✓ Anomaly detection
  - ✓ Grouping and Predictions
- ❖ *Video games*
  - ✓ Pokémon
  - ✓ PUBG
- ❖ *Text Analysis*
  - ✓ Spam Filtering
  - ✓ Information Extraction
  - ✓ Sentiment Analysis
- ❖ *Healthcare*
  - ✓ Emergency Room & Surgery
  - ✓ Research
  - ✓ Medical Imaging & Diagnostics

# Linear Regression



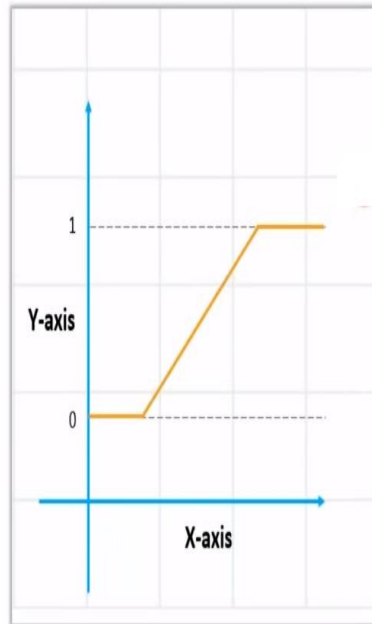
- Linear regression is a linear approach for modeling the relationship between a scalar dependent variable  $y$  and an independent variable  $x$ .
- where  $x$ ,  $y$ ,  $w$  are vectors of real numbers and  $w$  is a vector of weight parameters.
- The equation is written as:

$$\mathbf{y} = \mathbf{wx} + \mathbf{b}$$

- where  $b$  is the bias or the value of output for zero input

# Logistic Regression

$$P = \frac{e^{a+bX}}{1 + e^{a+bX}}$$

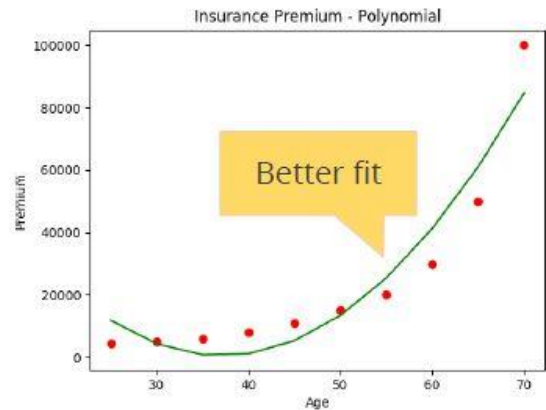
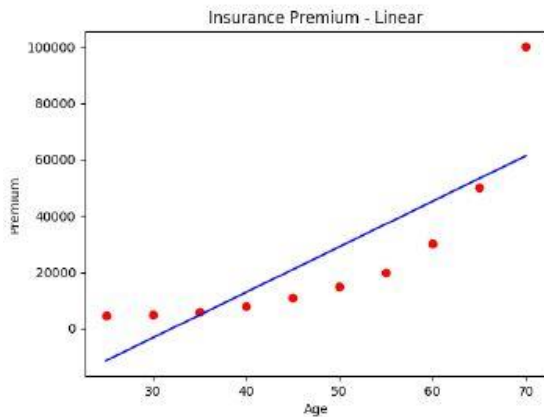


With this, our resulting curve cannot be formulated into a single formula. Hence we came up with **Logistic**!

# Polynomial Regression

## Example: Quadratic features

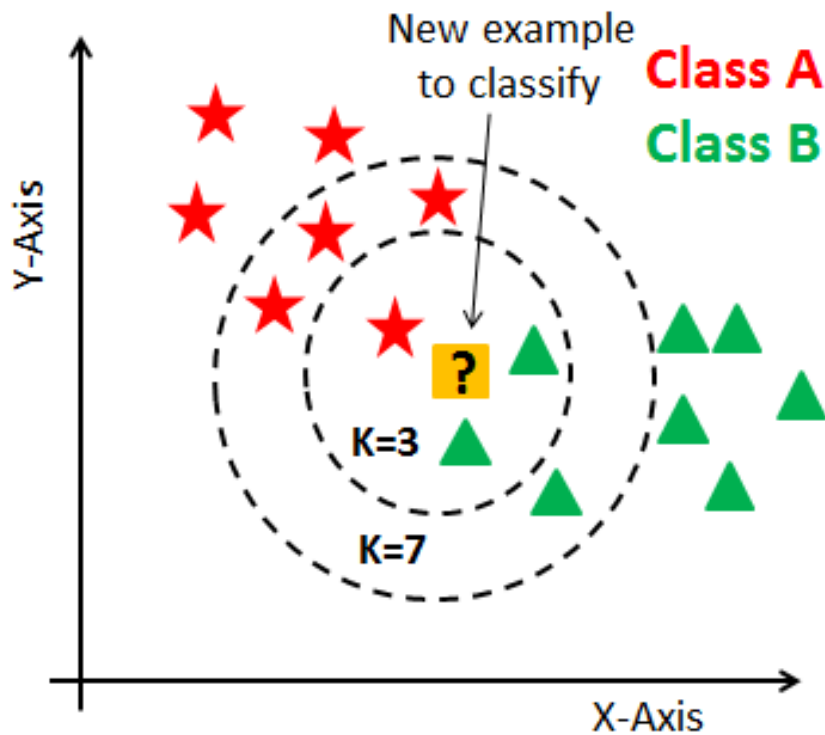
$$\begin{aligned}x_2' &= x_2^2 \\ y &= w_1x_1 + w_2x_2^2 + 6 \\ &= w_1x_1 + w_2x_2' + 6\end{aligned}$$



- Polynomial regression is applied when data is not formed in a straight line.
- It is used to fit a linear model to non-linear data by creating new features from powers of non-linear features.

$$\begin{array}{ccccccc} & & \text{Population} & \text{Population} & \text{Independent} & & \text{Random} \\ & & \text{Y intercept} & \text{Slope} & \text{Variable} & & \text{Error} \\ & & & \text{Coefficient} & & & \text{term} \\ \text{Dependent} & & & & & & \\ \text{Variable} & \rightarrow & Y_i = & \beta_0 + & \beta_1 X_i & + & \epsilon_i \\ & & & \underbrace{\beta_0 + \beta_1 X_i}_{\text{Linear component}} & & \underbrace{+ \epsilon_i}_{\text{Random Error component}} & \end{array}$$

# K-Nearest Neighbors (KNN) Classification



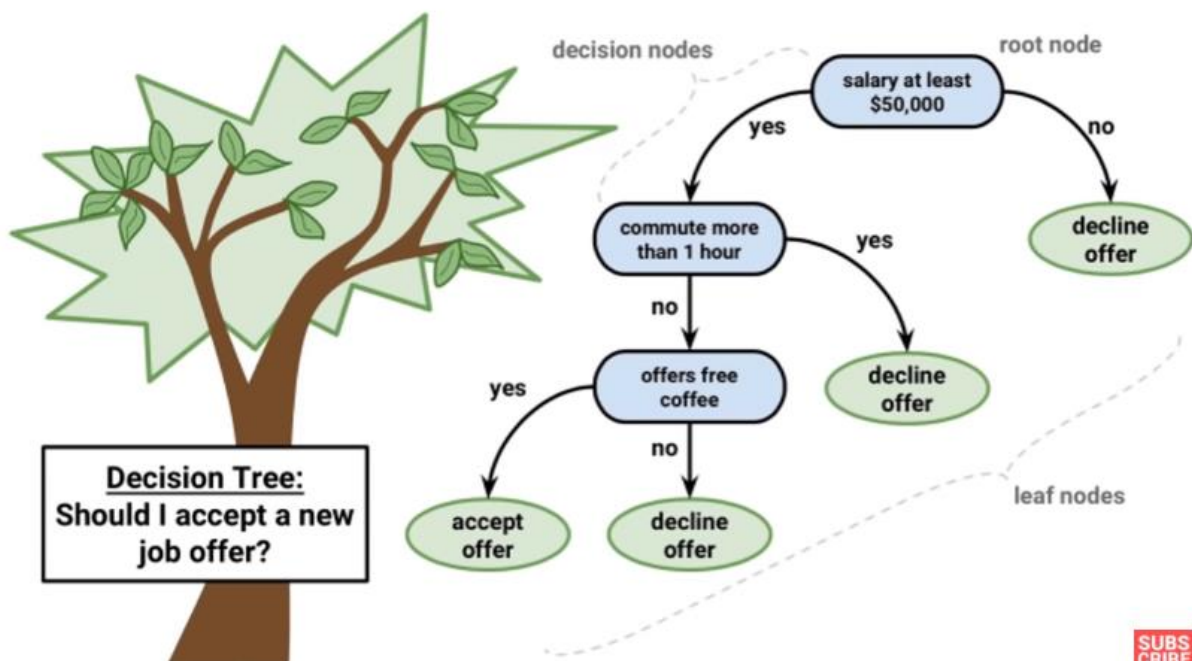
K-nearest Neighbors algorithm is used to assign a data point to clusters based on similarity measurement

A new input point is classified in the category such that it has the **greatest number of neighbors** from that category

$$d(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

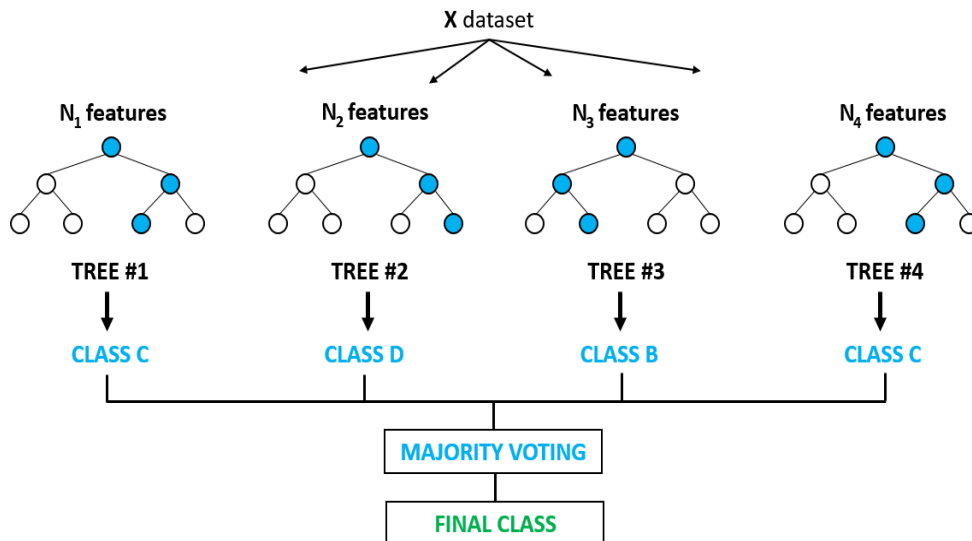
# Decision Tree Terminology

- A decision tree is a graphical representation of all the possible solutions to a decision based on a few conditions.
- Decision Trees are non-parametric models, which means that the number of parameters is not determined prior to training. Such models will normally overfit data.
- In contrast, a parametric model (such as a linear model) has a predetermined number of parameters, thereby reducing its degrees of freedom. This in turn prevents overfitting.



# Random Forest Algorithms

- Ensemble Learning uses the same algorithm multiple times or a group of different algorithms together to improve the prediction of a model.
- Random Forests use an ensemble of decision trees to perform regression tasks.
- 



# K-Means Clustering

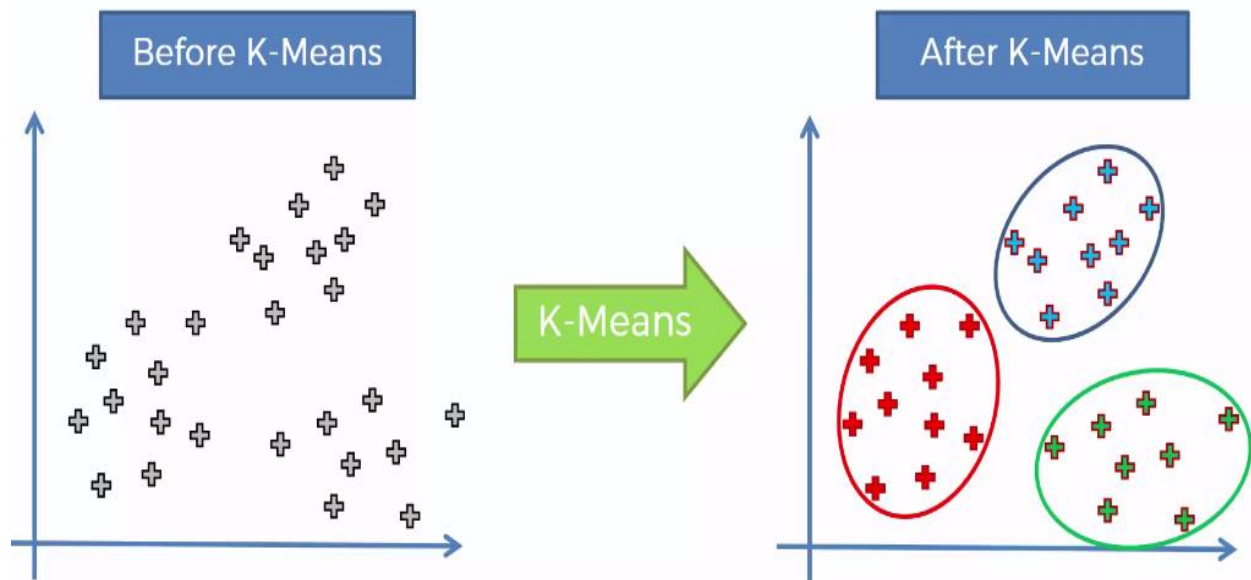
Step 1: randomly pick k centroids

Step 2: assign each point to the nearest centroid

Step 3: move each centroid to the center of the respective cluster Step 4: calculate the distance of the centroids from each point again

Step 5: move points across clusters and re-calculate the distance from the centroid

Step 6: keep moving the points across clusters until the Euclidean distance is minimized





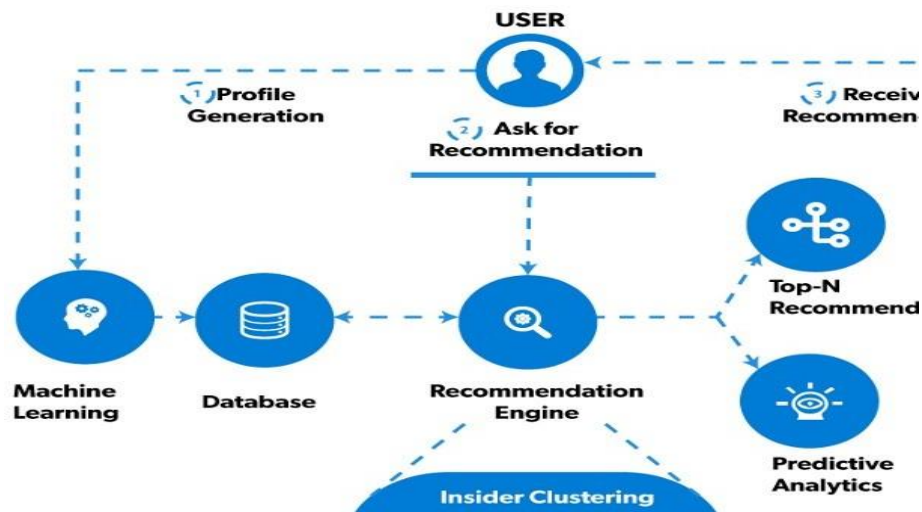
## **INTRODUCTION TO SONG RECOMENDATION SYSTEM**

Nowadays lots of music industries like amazon music, wink music, gaana.com are using recommender systems and the old fashioned way of selling music has changed to a totally different cloud based .Now all the music resources are present in their cloud and users can listen to the songs directly from the cloud. But the issue is there are lot of songs present in the cloud system. so we need to classify all the songs based on different genres ,artists locations , age groups, languages and the main goal is to classify these set of songs in accordance to the taste of the user. Because user expects valuable return after the investment of time as well as money thereby we can attract a lot of customers by providing various valuable services of their interests For this project we are using various machine learning algorithms as well as data mining techniques. We have implemented various algorithms and compared the results with one another to find the effective algorithm that suits our model.

### **Recommender System**

With the promotion of the Internet and the advancement of E-trade, the Ecommerce destinations offer a great many items available for purchasing. Picking among such a large number of choices is challenging for buyers. So clients typically lose all sense of direction in the huge space of ware data and can't discover the products they truly need. Recommender frameworks have risen in light of this issue. A recommender framework for an E-trade site prescribes items that are probably going to meet client's requirements. Suggestion frameworks have rapidly changed the way in which the life less sites can now interact and speak with their clients and users. As opposed to provide a constant involvement in which clients look for and conceivably purchase items, recommender frameworks increment cooperation to give a more extravagant or deal. Recommender architecture is used by the E-business objective to propose and suggest items and services which are similar to their clients. There are many constraints and parameters on which an internet service provider recommends a user certain choices and options depending upon some restrained set of parameters. These parameters can include language , age , nationality, history, likes, ratings, purchase and many more. The items can be prescribed and suggested dependent on the best generally speaking and interacting vender on a particular site, in the presence of the social and economic constraints of the client, or dependent on an examination of the past purchasing conduct of the client as an expectation for future purchasing conduct. These parameters enable the service providers to analyse and determine a set of choices and preferences. Moreover, these strategies have been used extensively and are a piece of personalization on a site, since this enable the site to adjust to every client in accordance to the client or user .Moreover these famous and successful online service providers use personalization as a key component while recommendations because generalization cannot be more accurate under defined parameters. Recommender frameworks mechanize personal services and

platforms on the internet, which facilitates and empowers singular personalization for every client



## Why Recommender Systems?

As the web moved from a proprietor model to an open publicly supporting model and enabled individuals to contribute unreservedly, it saw an exponential ascent in the measure of substance accessible, which was something to be thankful for. Be that as it may, this prompted two noteworthy issues:

- i. Aggregation: The measure of data turned out to be large to the point that it inspired extreme to oversee it while as yet having the capacity to run a web benefit that was reachable to all parts of the world. This issue was tackled by building overall substance conveyance and dissemination systems, helped by the ascent of NoSQL Database frameworks and diminishing stockpiling costs.
- ii. Searching: The second significant issue was the means by which to guarantee that the data is inside the scope of the client and that the client does not become mixed up in the immense information dumps accessible. This turned out to be a significantly more concerning issue than accumulation since the information troves are tremendous and every client carries alongside him/her a remarkable point of view and consequently a one of a kind pursuit design. We are as yet attempting to take care of this issue today and are a long way from accomplishing an ideal answer for it. This is the place recommender frameworks become possibly the most important factor.

iii. More or less, a recommender framework is a framework that predicts client reaction to an assortment of choices. Anticipating what the client may get next is the basic point of a recommender framework. There is a broad class of web applications that include foreseeing the client's reaction to choices. Such an office is known as a recommender framework.



Generalized Recommender

- iv. In today's services like Netflix, Spotify, youtube offers their customers with plenty of choices. A person using such a service is known as client or user and to assist user, services can use information filtering to recommend items to users. Items can be several things such as books, music, movies, news etc.
- v. Recommender systems need information for functioning, data about a particular user. This particular data can be fetched directly or indirectly. Directly collecting data means that user of a particular service gives feedback and review of the item. Indirectly means that system will analyse the users interaction with the particular service consisting of history and present services.

## Problem statement

The main goal is suggesting best set of options to the user. For a specific user we had their song history frequency list liked songs. From all this information we had to predict what songs user might like then the question comes: how can we use all this

information to achieve our goal. As it is not a straightforward task to find the relevance between various songs it might be possible that one song which looks similar to another may be completely different and users may dislike that song or may be that song is not of user's taste. There are lots of users around the world and lots of songs so making a relevance between songs and users is a tedious task.

## **Objective**

### **Learning Objective**

Learning objective of doing this project was to first learn about machine learning and its key concepts and various data mining techniques and algorithms. Other goal was to learn a lot of machine learning algorithms and how to use them. Just learning algorithms doesn't make you an engineer; the real task is to learn which is the right algorithm to apply for a specific project.

### **Outcome objective**

The main object in terms of outcome was to create a framework for users which can help them suggest the right songs for them. This project aims to find the correlation and similarity between different music lovers' tastes and various songs so that if a user's taste is similar to the other one we can recommend the songs of one to another on the basis of similar taste. Or if a song is similar to the other we can suggest that song to the user that listens the first one. One of the objects of this project is to reduce the time that user generally wastes on looking for the right song.

# **Methodology**

## **Functional**

### **Requirement**

The functional requirement specification of the project are mainly categorized as user requirements, security requirements, and device requirement each of which are explained in detail below:

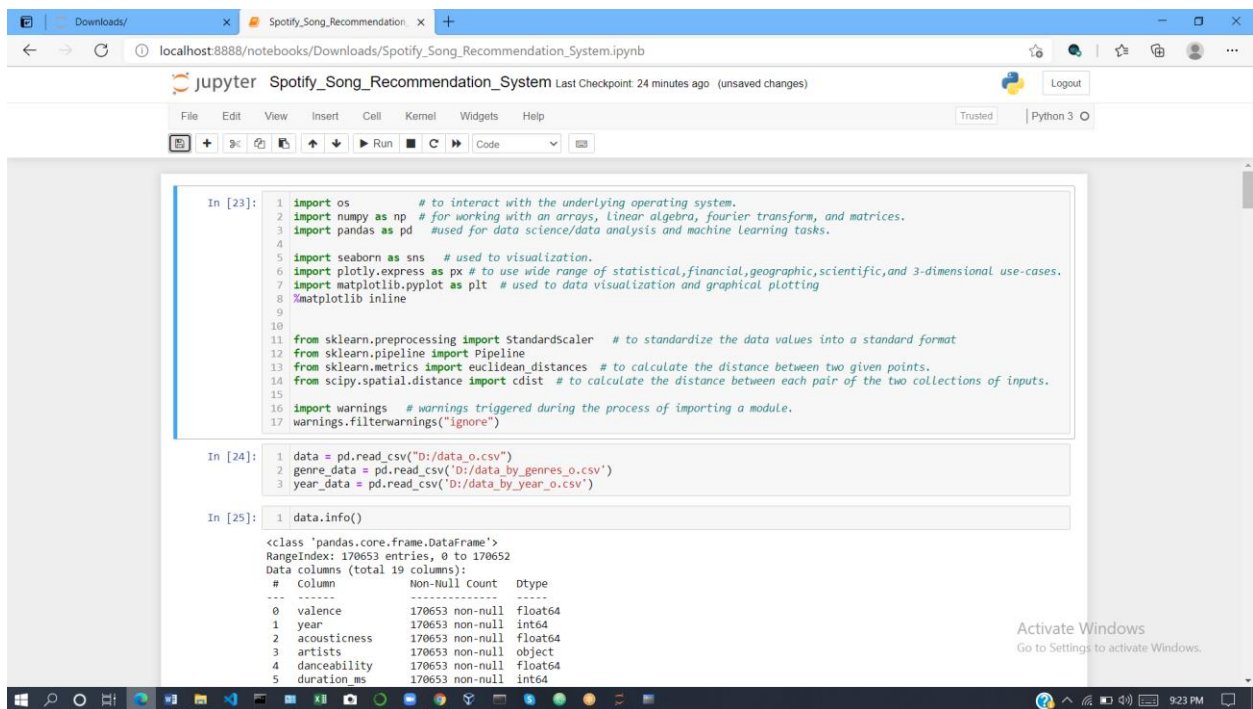
i. User Requirement: User ought to have account on framework and client must have somewhere around one song listened to investigate the identity for the music suggestion.

### **Non-functional Requirement**

- i. Performance: The framework will have a speedy, exact and dependable outcomes.
- ii. Capacity and Scalability: The framework will have the capacity to store identity registered by the framework into the database.
- iii. Availability: The framework will be accessible to client whenever at whatever point there is an Internet association.
- iv. Recovery: if there should arise an occurrence of breaking down or inaccessibility of server, the framework ought to have the capacity to recuperate and keep any information misfortune or excess.
- v. Flexibility and Portability: System will be available whenever from any areas

# Import Libraries

In the code below, I imported some other basic libraries for data manipulation and visualization.



The screenshot shows a Jupyter Notebook titled "Spotify\_Song\_Recommendation\_System" running on a local host. The notebook contains three code cells. The first cell imports various libraries: `os` for file operations, `numpy` as `np` for arrays, `pandas` as `pd` for data manipulation, `seaborn` as `sns` for visualization, `plotly.express` as `px` for interactive plots, `matplotlib.pyplot` as `plt` for standard plotting, `StandardScaler` from `sklearn.preprocessing` for data normalization, `Pipeline` from `sklearn.pipeline` for model building, `euclidean_distances` from `sklearn.metrics` for distance calculations, `cdist` from `scipy.spatial.distance` for pairwise distance calculations, and `warnings` to suppress warnings. The second cell reads three CSV files: `data_o.csv`, `data_by_genres_o.csv`, and `data_by_year_o.csv`. The third cell displays the output of `data.info()`, showing the data structure and column types.

```
In [23]: 1 import os # to interact with the underlying operating system.
2 import numpy as np # for working with an arrays, linear algebra, fourier transform, and matrices.
3 import pandas as pd #used for data science/data analysis and machine learning tasks.
4
5 import seaborn as sns # used to visualization.
6 import plotly.express as px # to use wide range of statistical,financial,geographic,scientific,and 3-dimensional use-cases.
7 import matplotlib.pyplot as plt # used to data visualization and graphical plotting
8 %matplotlib inline
9
10
11 from sklearn.preprocessing import StandardScaler # to standardize the data values into a standard format
12 from sklearn.pipeline import Pipeline
13 from sklearn.metrics import euclidean_distances # to calculate the distance between two given points.
14 from scipy.spatial.distance import cdist # to calculate the distance between each pair of the two collections of inputs.
15
16 import warnings # warnings triggered during the process of importing a module.
17 warnings.filterwarnings("ignore")

In [24]: 1 data = pd.read_csv("D:/data_o.csv")
2 genre_data = pd.read_csv("D:/data_by_genres_o.csv")
3 year_data = pd.read_csv("D:/data_by_year_o.csv")

In [25]: 1 data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170653 entries, 0 to 170652
Data columns (total 19 columns):
 #   column              Non-Null Count  Dtype
---  -
0   valence              170653 non-null float64
1   year                170653 non-null int64
2   acousticness        170653 non-null float64
3   artists             170653 non-null object
4   danceability         170653 non-null float64
5   duration_ms         170653 non-null int64
```

I have included the column metadata below that was generated by calling the Pandas **info** function for each data frame.

```
Spotify_Song_Recommendation_System

In [24]: 1 data = pd.read_csv("D:/data_o.csv")
          2 genre_data = pd.read_csv("D:/data_by_genres_o.csv")
          3 year_data = pd.read_csv("D:/data_by_year_o.csv")

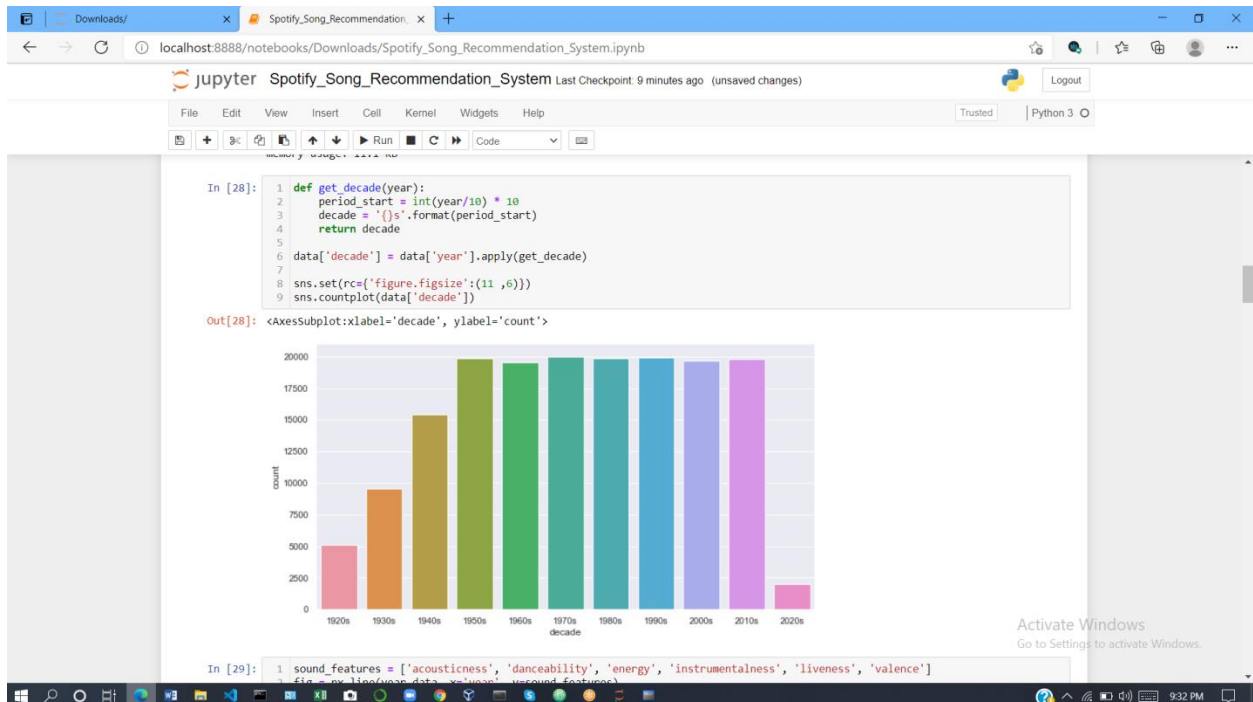
In [25]: 1 data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 170653 entries, 0 to 170652
Data columns (total 19 columns):
 #   Column              Non-Null Count  Dtype
---  -
 0   valence              170653 non-null float64
 1   year                 170653 non-null int64
 2   acousticness         170653 non-null float64
 3   artists              170653 non-null object
 4   danceability         170653 non-null float64
 5   duration_ms          170653 non-null int64
 6   energy               170653 non-null float64
 7   explicit             170653 non-null int64
 8   id                   170653 non-null object
 9   instrumentalness      170653 non-null float64
10   key                  170653 non-null int64
11   liveness              170653 non-null float64
12   loudness              170653 non-null float64
13   mode                 170653 non-null int64
14   name                 170653 non-null object
15   popularity            170653 non-null int64
16   release_date          170653 non-null object
17   speechiness           170653 non-null float64
18   tempo                 170653 non-null float64
dtypes: float64(9), int64(6), object(4)
memory usage: 24.7+ MB

In [26]: 1 genre_data.info()

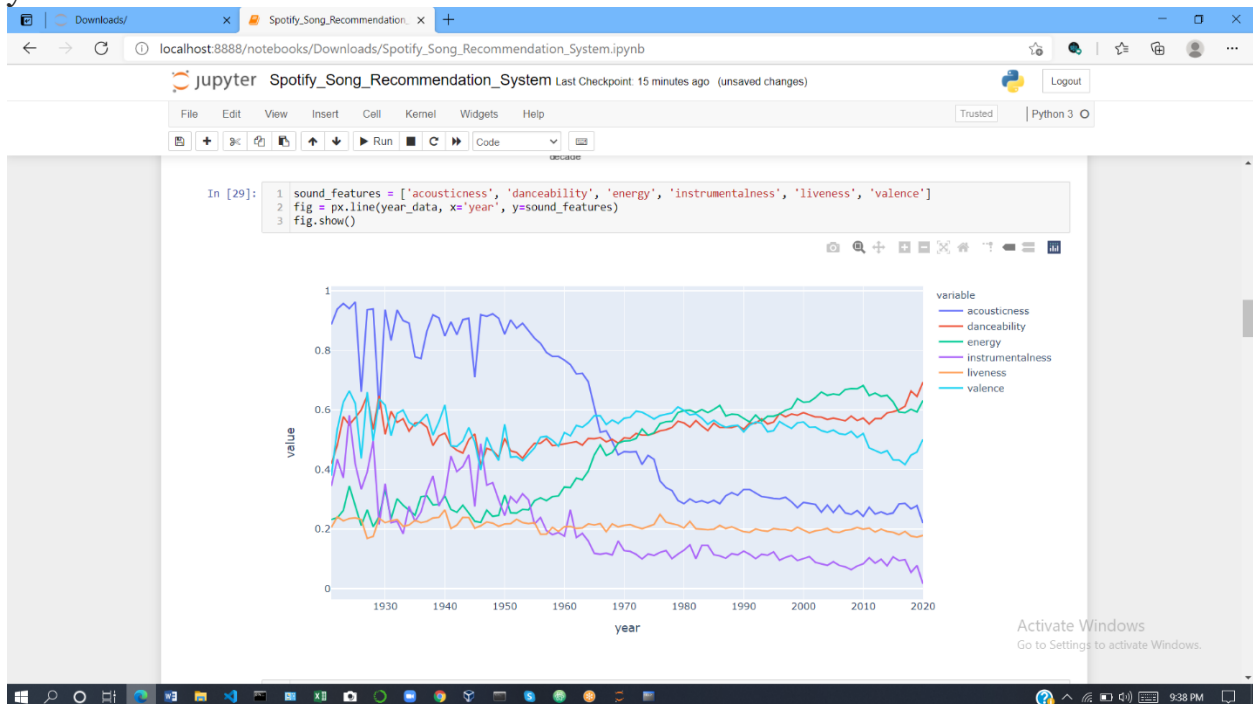
<class 'pandas.core.frame.DataFrame'>
Index: 2973 entries, 0 to 2972
```

## Music Over Time



Using the data grouped by year, we can understand how the overall sound of music has changed from 1921 to 2020.

Using the data grouped by year, we can understand how the overall sound of music has changed from 1921 to 2020. In the code below, I used Plotly to visualize the values of different audio features for songs over the past 100 years.

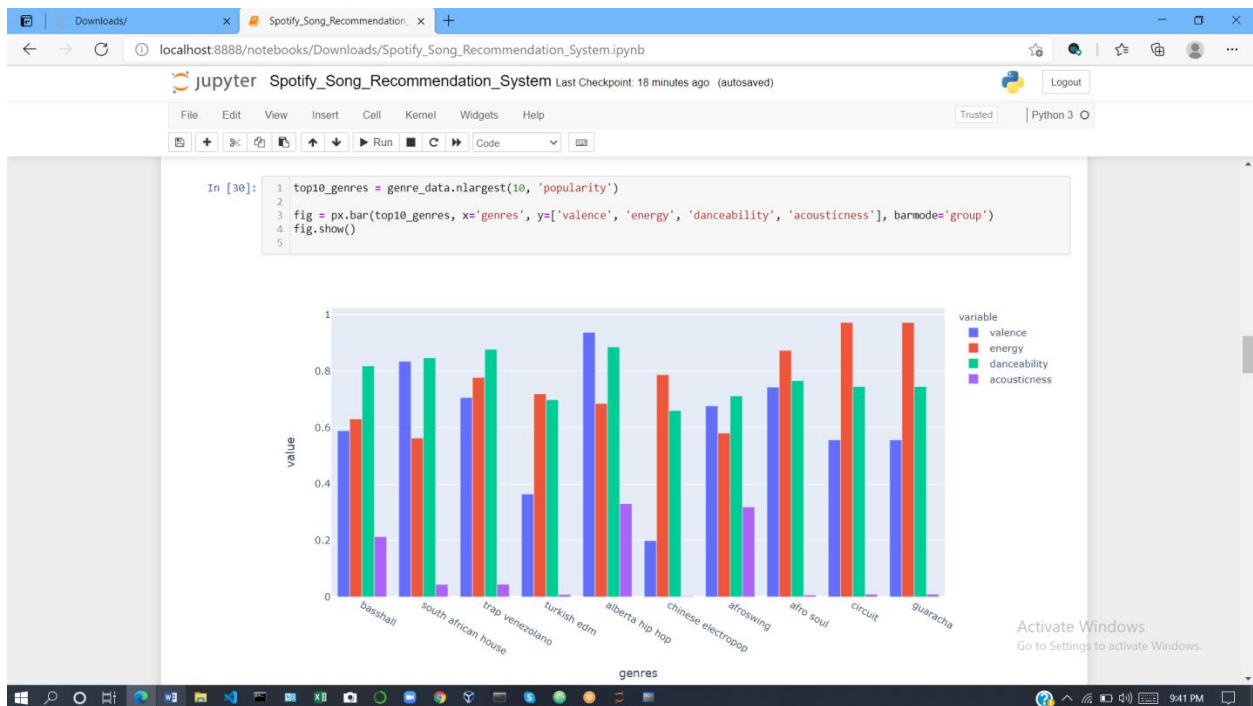


Based on the plot above, we can see that music has transitioned from the more acoustic and instrumental sound of the early 1900s to the more danceable and energetic sound of the 2000s. The majority of the tracks from the 1920s were likely instrumental pieces from classical and jazz genres. The music of the 2000s sounds very different due to the advent of computers and advanced audio engineering technology that allows us to create electronic music with a wide range of effects and beats.



## Characteristics of Different Genres

This dataset contains the audio features for different songs along with the audio features for different genres. We can use this information to compare different genres and understand their unique differences in sound. In the code below, I selected the ten most popular genres from the dataset and visualized audio features for each of them.



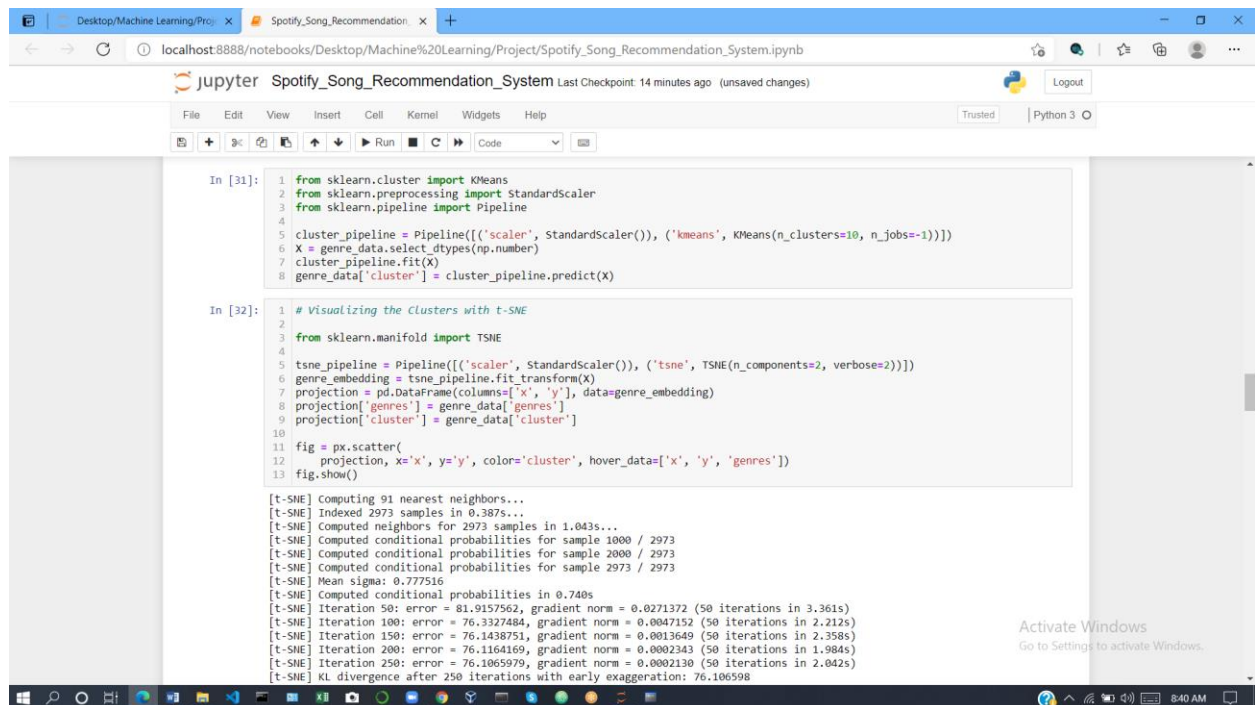
Many of the genres above, such as Chinese electropop are extremely specific and likely belong to one or more broad genres such as pop or electronic music. We can take these highly specific genres and understand how similar they are to other genres by clustering them based on their audio features.

# Clustering Genres with K-Means

Here, I used simple K-means clustering algorithm to divide the genres in this dataset into ten clusters based on the numerical audio features of each genres.

## Visualizing the Genre Clusters with t-SNE

There are many audio features for each genre and it is difficult to visualize clusters in a high-dimensional space. However, we can use a dimensionality reduction technique known as [t-Distributed Stochastic Neighbor Embedding](#) to compress the data into a two-dimensional space as demonstrated in the code below.

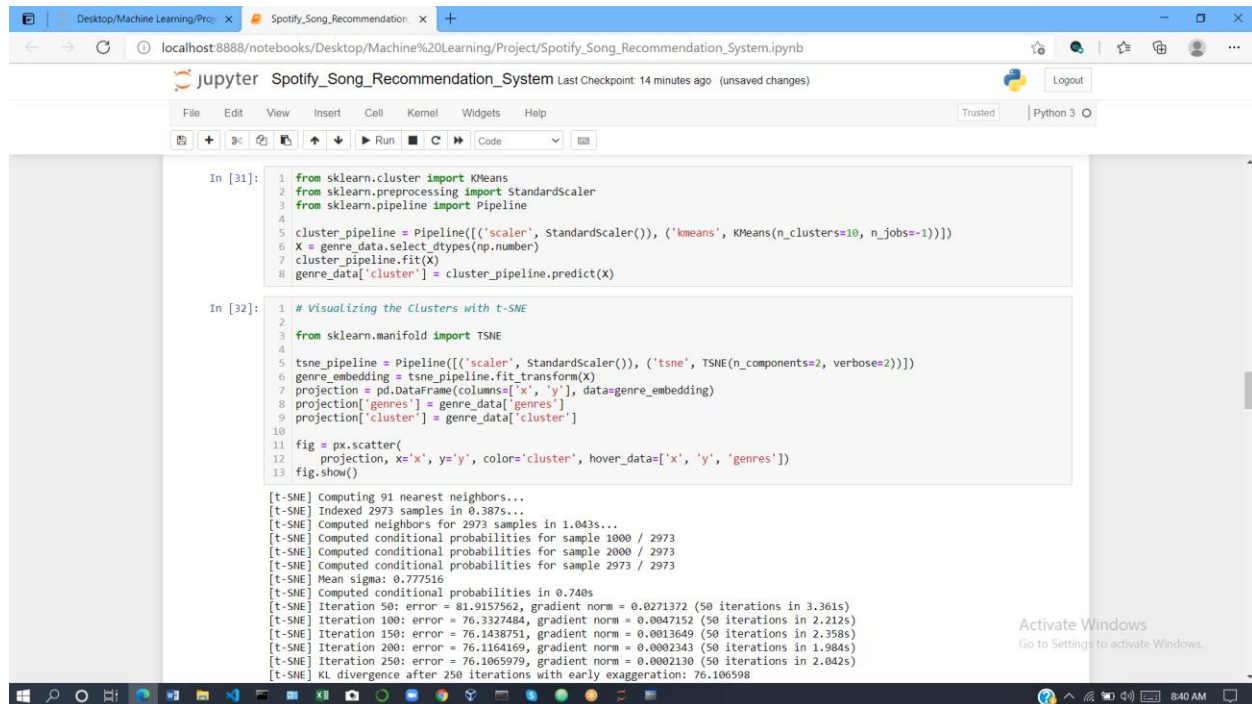


```
In [31]: 1 from sklearn.cluster import KMeans
2 from sklearn.preprocessing import StandardScaler
3 from sklearn.pipeline import Pipeline
4
5 cluster_pipeline = Pipeline([('scaler', StandardScaler()), ('kmeans', KMeans(n_clusters=10, n_jobs=-1))])
6 X = genre_data.select_dtypes(np.number)
7 cluster_pipeline.fit(X)
8 genre_data['cluster'] = cluster_pipeline.predict(X)

In [32]: 1 # Visualizing the Clusters with t-SNE
2
3 from sklearn.manifold import TSNE
4
5 tsne_pipeline = Pipeline([('scaler', StandardScaler()), ('tsne', TSNE(n_components=2, verbose=2))])
6 genre_embedding = tsne_pipeline.fit_transform(X)
7 projection = pd.DataFrame(columns=['x', 'y'], data=genre_embedding)
8 projection['genres'] = genre_data['genres']
9 projection['cluster'] = genre_data['cluster']
10
11 fig = px.scatter(
12     projection, x='x', y='y', color='cluster', hover_data=['x', 'y', 'genres'])
13 fig.show()

[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 2973 samples in 0.387s...
[t-SNE] Computed neighbors for 2973 samples in 1.043s...
[t-SNE] Computed conditional probabilities for sample 1000 / 2973
[t-SNE] Computed conditional probabilities for sample 2000 / 2973
[t-SNE] Computed conditional probabilities for sample 2973 / 2973
[t-SNE] Mean sigma: 0.777516
[t-SNE] Computed conditional probabilities in 0.740s
[t-SNE] Iteration 50: error = 81.9157562, gradient norm = 0.0271372 (50 iterations in 3.361s)
[t-SNE] Iteration 100: error = 76.3327484, gradient norm = 0.0047152 (50 iterations in 2.212s)
[t-SNE] Iteration 150: error = 76.1438751, gradient norm = 0.0013649 (50 iterations in 2.358s)
[t-SNE] Iteration 200: error = 76.1164169, gradient norm = 0.0002343 (50 iterations in 1.984s)
[t-SNE] Iteration 250: error = 76.1065979, gradient norm = 0.0002130 (50 iterations in 2.042s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 76.106598
```

Now, we can easily visualize the genre clusters in a two-dimensional coordinate plane by using Plotly's scatter function.



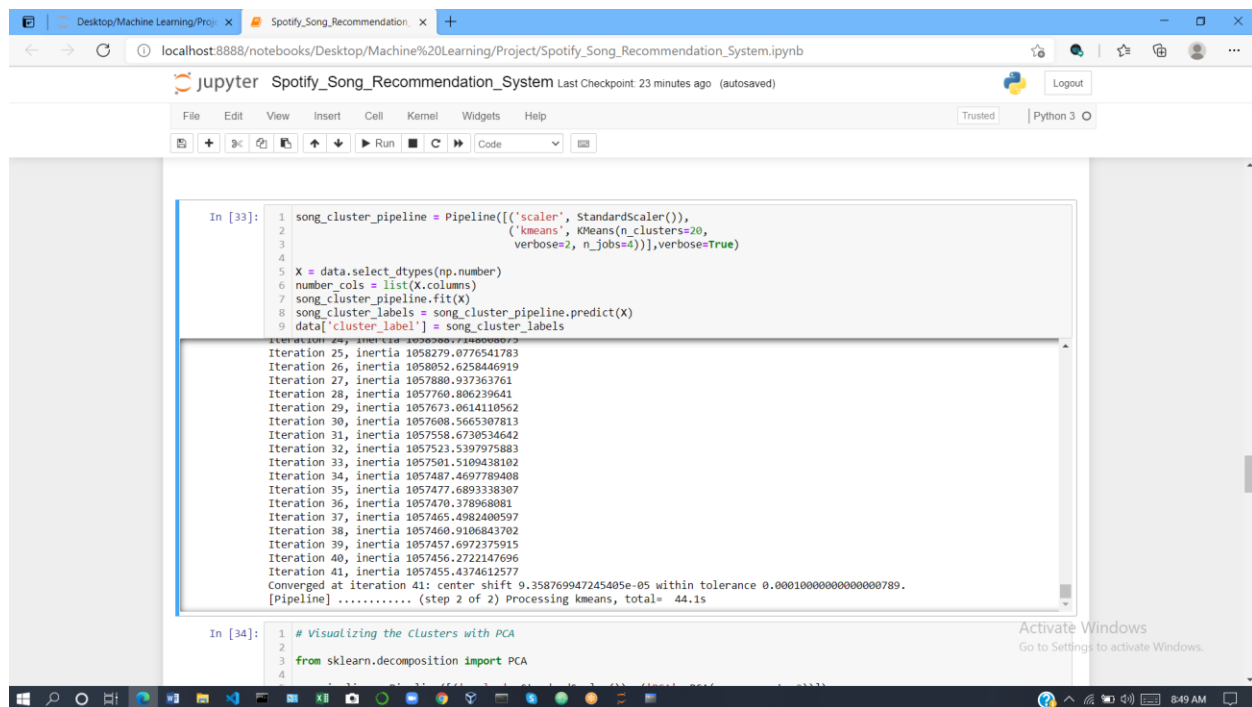
```
In [31]: 1 from sklearn.cluster import KMeans
2 from sklearn.preprocessing import StandardScaler
3 from sklearn.pipeline import Pipeline
4
5 cluster_pipeline = Pipeline([('scaler', StandardScaler()), ('kmeans', KMeans(n_clusters=10, n_jobs=-1))])
6 X = genre_data.select_dtypes(np.number)
7 cluster_pipeline.fit(X)
8 genre_data['cluster'] = cluster_pipeline.predict(X)

In [32]: 1 # Visualizing the Clusters with t-SNE
2
3 from sklearn.manifold import TSNE
4
5 tsne_pipeline = Pipeline([('scaler', StandardScaler()), ('tsne', TSNE(n_components=2, verbose=2))])
6 genre_embedding = tsne_pipeline.fit_transform(X)
7 projection = pd.DataFrame(columns=['x', 'y'], data=genre_embedding)
8 projection['genres'] = genre_data['genres']
9 projection['cluster'] = genre_data['cluster']
10
11 fig = px.scatter(
12     projection, x='x', y='y', color='cluster', hover_data=['x', 'y', 'genres'])
13 fig.show()
```

[t-SNE] Computing 91 nearest neighbors...  
[t-SNE] Indexed 2973 samples in 0.387s...  
[t-SNE] Computed neighbors for 2973 samples in 1.043s...  
[t-SNE] Computed conditional probabilities for sample 1000 / 2973  
[t-SNE] Computed conditional probabilities for sample 2000 / 2973  
[t-SNE] Computed conditional probabilities for sample 2973 / 2973  
[t-SNE] Mean sigma: 0.777516  
[t-SNE] Computed conditional probabilities in 0.740s  
[t-SNE] Iteration 50: error = 81.9157562, gradient norm = 0.0271372 (50 iterations in 3.361s)  
[t-SNE] Iteration 100: error = 76.3327484, gradient norm = 0.0047152 (50 iterations in 2.212s)  
[t-SNE] Iteration 150: error = 76.1438751, gradient norm = 0.0013649 (50 iterations in 2.358s)  
[t-SNE] Iteration 200: error = 76.1164169, gradient norm = 0.0002343 (50 iterations in 1.984s)  
[t-SNE] Iteration 250: error = 76.1065979, gradient norm = 0.0002130 (50 iterations in 2.042s)  
[t-SNE] KL divergence after 250 iterations with early exaggeration: 76.106598

## Clustering Songs with K-Means

We can also cluster the songs using K-means as demonstrated below in order to understand how to build a better recommendation system.



The screenshot shows a Jupyter Notebook titled "Spotify\_Song\_Recommendation\_System" running on a local host. The notebook contains two code cells. The first cell, labeled "In [33]:", defines a KMeans clustering pipeline using scikit-learn's Pipeline, StandardScaler, and KMeans. It then fits the pipeline to the data and predicts cluster labels. The output of this cell shows the progress of the KMeans algorithm, including iterations 25 through 41, with inertia values and a convergence message. The second cell, labeled "In [34]:", imports PCA from sklearn.decomposition. The notebook interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom showing the time as 8:49 AM. An "Activate Windows" watermark is visible in the bottom right corner.

```
In [33]: 1 song_cluster_pipeline = Pipeline([('scaler', StandardScaler()),
2                                         ('kmeans', KMeans(n_clusters=20,
3                                         verbose=2, n_jobs=4))], verbose=True)
4
5 X = data.select_dtypes(np.number)
6 number_cols = list(X.columns)
7 song_cluster_pipeline.fit(X)
8 song_cluster_labels = song_cluster_pipeline.predict(X)
9 data['cluster_label'] = song_cluster_labels

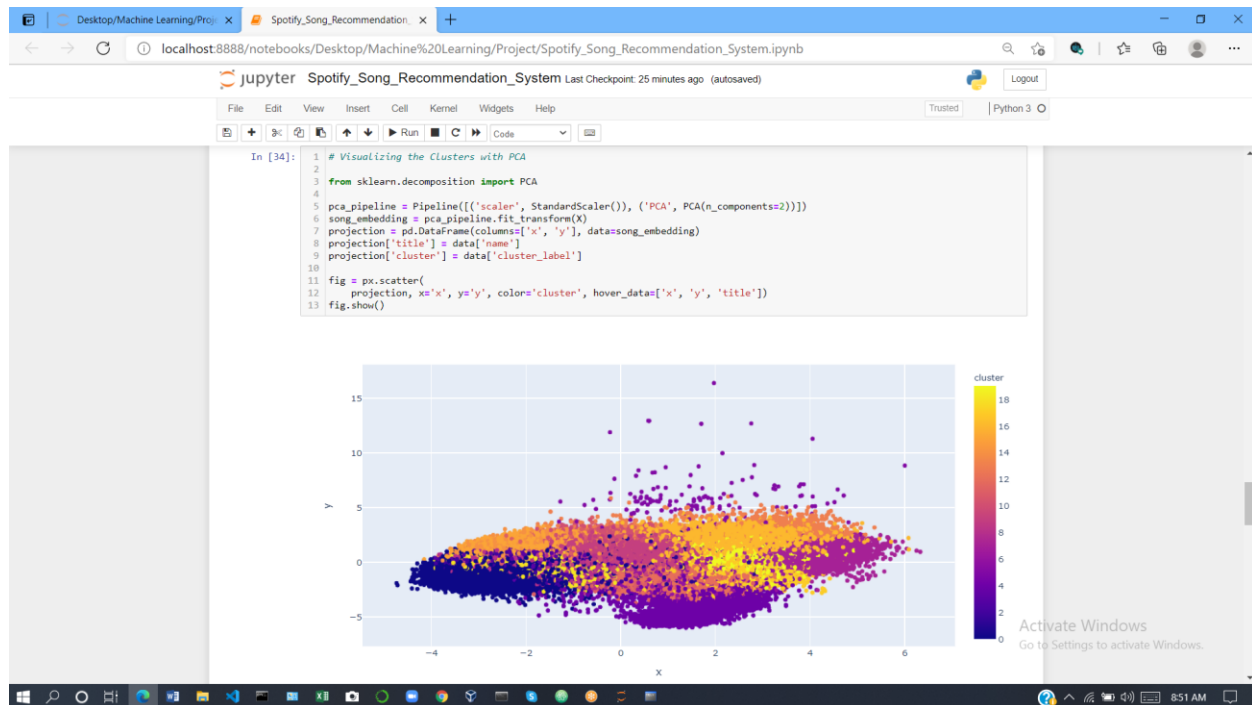
Iteration 25, inertia 1058279.0776541783
Iteration 26, inertia 1058052.6258446919
Iteration 27, inertia 1057880.937363761
Iteration 28, inertia 1057760.806239641
Iteration 29, inertia 1057673.0614110562
Iteration 30, inertia 1057608.5665307813
Iteration 31, inertia 1057558.6730534642
Iteration 32, inertia 1057523.5397975883
Iteration 33, inertia 1057501.5109438102
Iteration 34, inertia 1057487.4697789408
Iteration 35, inertia 1057477.6893338307
Iteration 36, inertia 1057470.378968081
Iteration 37, inertia 1057465.4982400597
Iteration 38, inertia 1057460.9186843702
Iteration 39, inertia 1057457.6972375915
Iteration 40, inertia 1057456.2722147696
Iteration 41, inertia 1057455.4374612577
Converged at iteration 41: center shift 9.358769947245405e-05 within tolerance 0.000100000000000000789.
[Pipeline] ..... (step 2 of 2) Processing kmeans, total= 44.1s

In [34]: 1 # Visualizing the Clusters with PCA
2
3 from sklearn.decomposition import PCA
4
```

## Visualizing the Song Clusters with PCA

The song data frame is much larger than the genre data frame so I decided to use PCA for dimensionality reduction rather than t-SNE because it runs significantly faster.

Now, we can visualize the song cluster in a two-dimensional space using the code

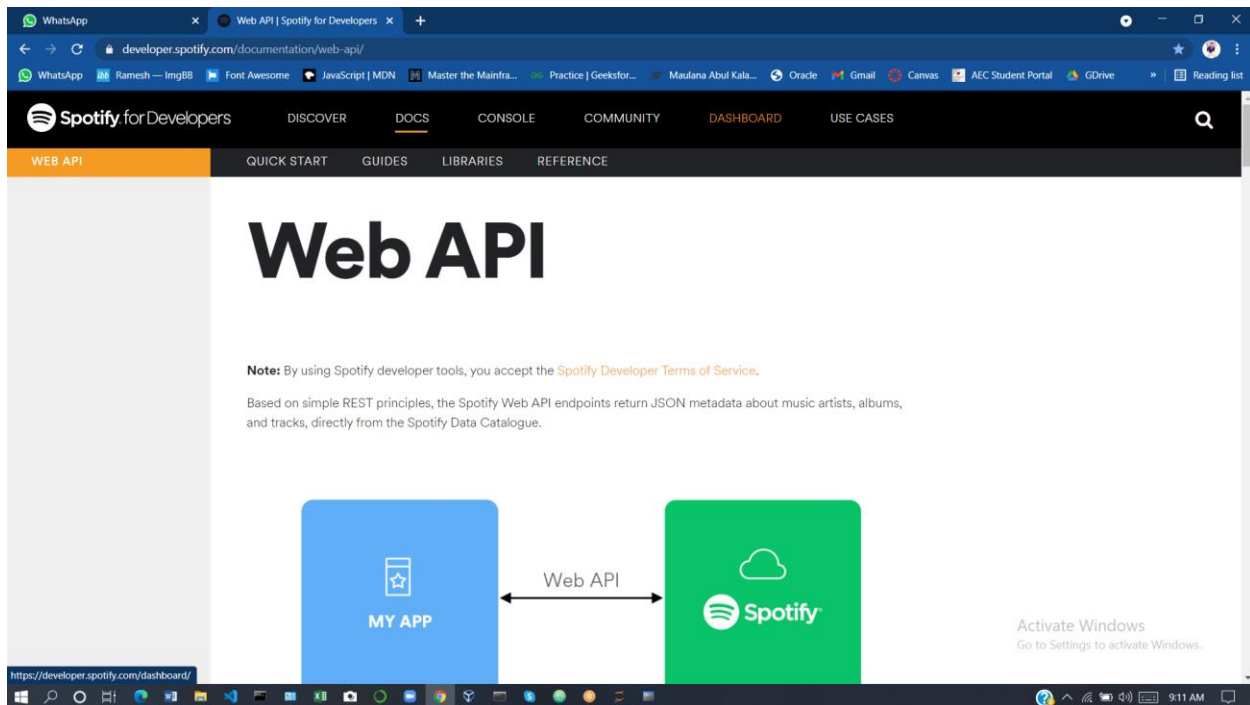


The plot above is interactive, so you can see the title of each song when you hover over the points. If you spend some time exploring the plot above you'll find that similar songs tend to be located close to each other and songs within clusters tend to be at least somewhat similar. This observation is the key idea behind the content-based recommendation system that I created in the next section.

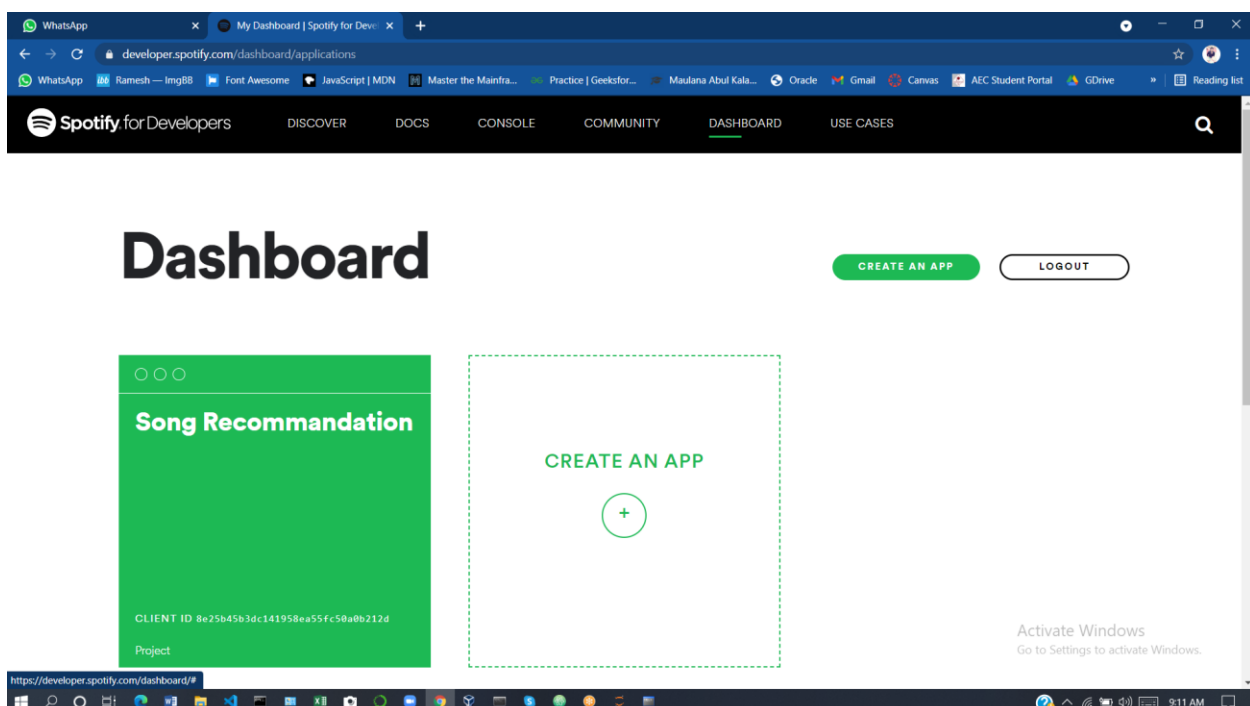
## Register Your App:

[Developer.spotify.com/documentation/web-api/](https://developer.spotify.com/documentation/web-api/)

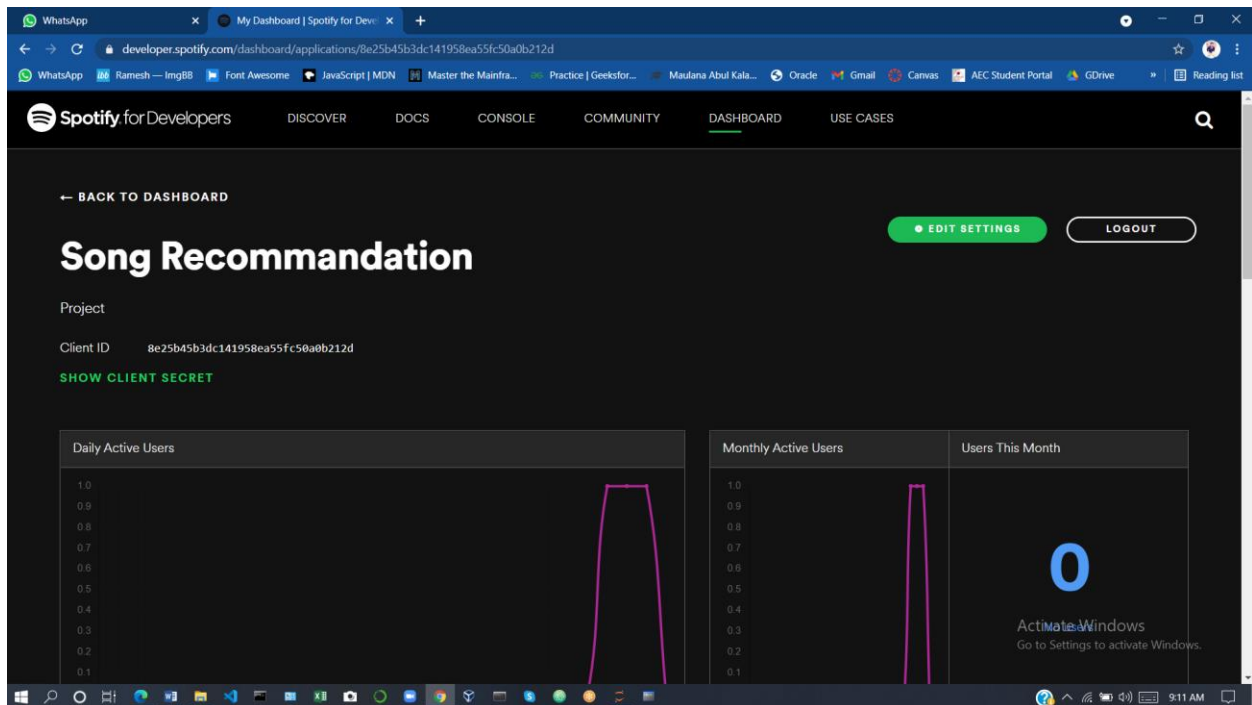
1. On your Dashboard click **CREATE** a **CLIENT ID**.



2. Enter Application Name and Application Description and then click **CREATE**. Your application is registered, and the app view opens.



After creating an app on the [Spotify Developer's page](#) and save your Client ID and secret ID.



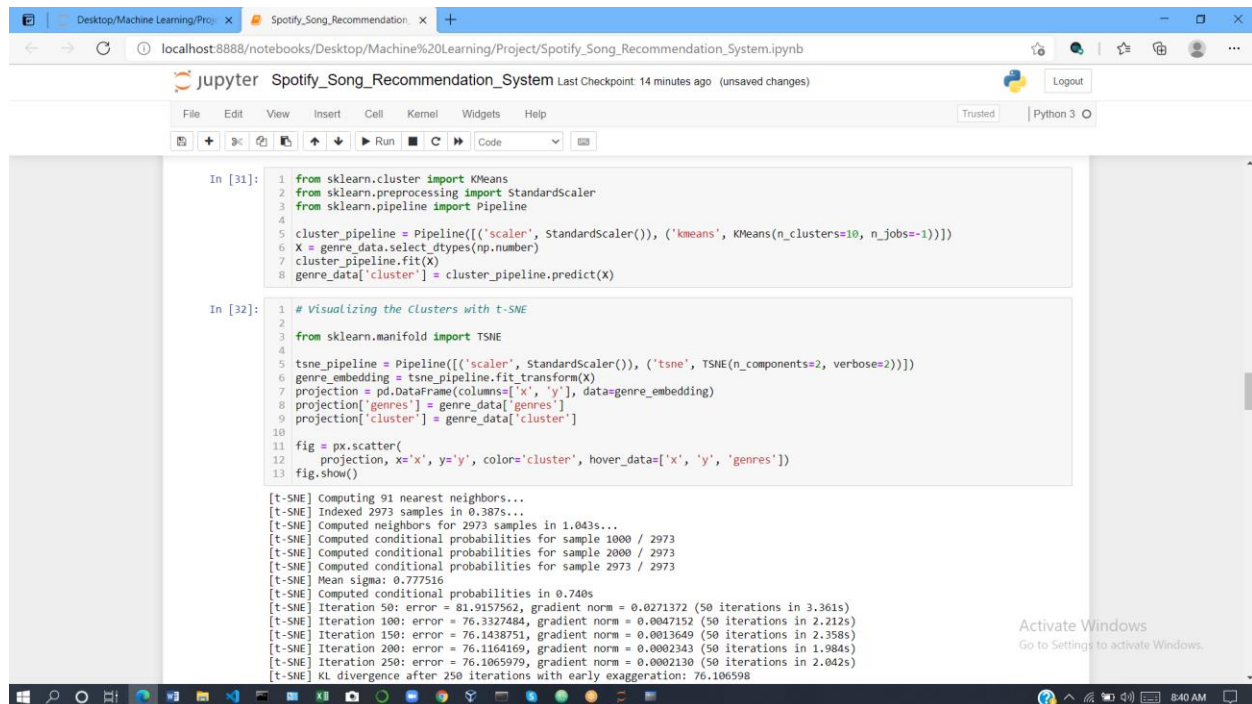
## Installing Spotipy

[Spotipy](#) is a Python client for the Spotify Web API that makes it easy for developers to fetch data and query Spotify's catalog for songs.

You can install Spotipy with pip using the command below.

```
!pip install spotipy
```





```
In [31]: 1 from sklearn.cluster import KMeans
2 from sklearn.preprocessing import StandardScaler
3 from sklearn.pipeline import Pipeline
4
5 cluster_pipeline = Pipeline([('scaler', StandardScaler()), ('kmeans', KMeans(n_clusters=10, n_jobs=-1))])
6 X = genre_data.select_dtypes(np.number)
7 cluster_pipeline.fit(X)
8 genre_data['cluster'] = cluster_pipeline.predict(X)

In [32]: 1 # Visualizing the Clusters with t-SNE
2
3 from sklearn.manifold import TSNE
4
5 tsne_pipeline = Pipeline([('scaler', StandardScaler()), ('tsne', TSNE(n_components=2, verbose=2))])
6 genre_embedding = tsne_pipeline.fit_transform(X)
7 projection = pd.DataFrame(columns=['x', 'y'], data=genre_embedding)
8 projection['genres'] = genre_data['genres']
9 projection['cluster'] = genre_data['cluster']
10
11 fig = px.scatter(
12     projection, x='x', y='y', color='cluster', hover_data=['x', 'y', 'genres'])
13 fig.show()

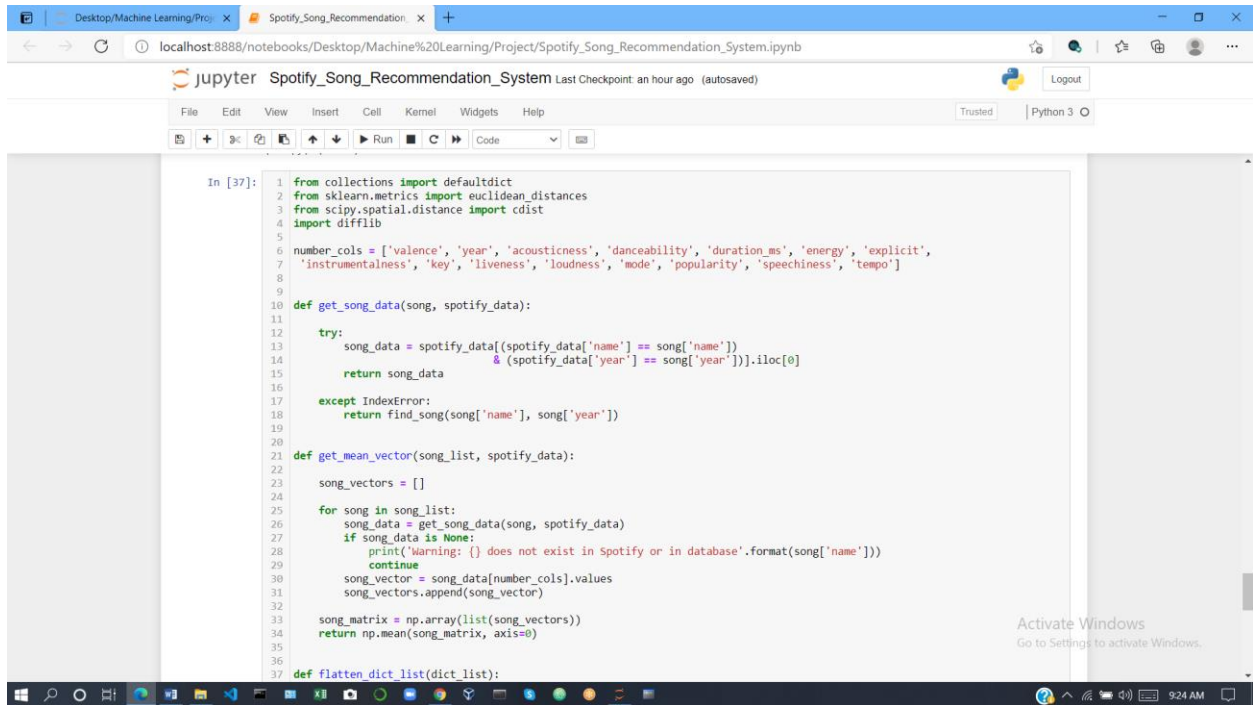
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 2973 samples in 0.3875...
[t-SNE] Computed neighbors for 2973 samples in 1.043s...
[t-SNE] Computed conditional probabilities for sample 1000 / 2973
[t-SNE] Computed conditional probabilities for sample 2000 / 2973
[t-SNE] Computed conditional probabilities for sample 2973 / 2973
[t-SNE] Mean sigma: 0.777516
[t-SNE] Computed conditional probabilities in 0.740s
[t-SNE] Iteration 50: error = 81.9157562, gradient norm = 0.0271372 (50 iterations in 3.361s)
[t-SNE] Iteration 100: error = 76.3327484, gradient norm = 0.0047152 (50 iterations in 2.212s)
[t-SNE] Iteration 150: error = 76.1438751, gradient norm = 0.0013649 (50 iterations in 2.358s)
[t-SNE] Iteration 200: error = 76.1164169, gradient norm = 0.0002343 (50 iterations in 1.984s)
[t-SNE] Iteration 250: error = 76.1065979, gradient norm = 0.0002130 (50 iterations in 2.042s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 76.106598
```

## Generating song recommendations

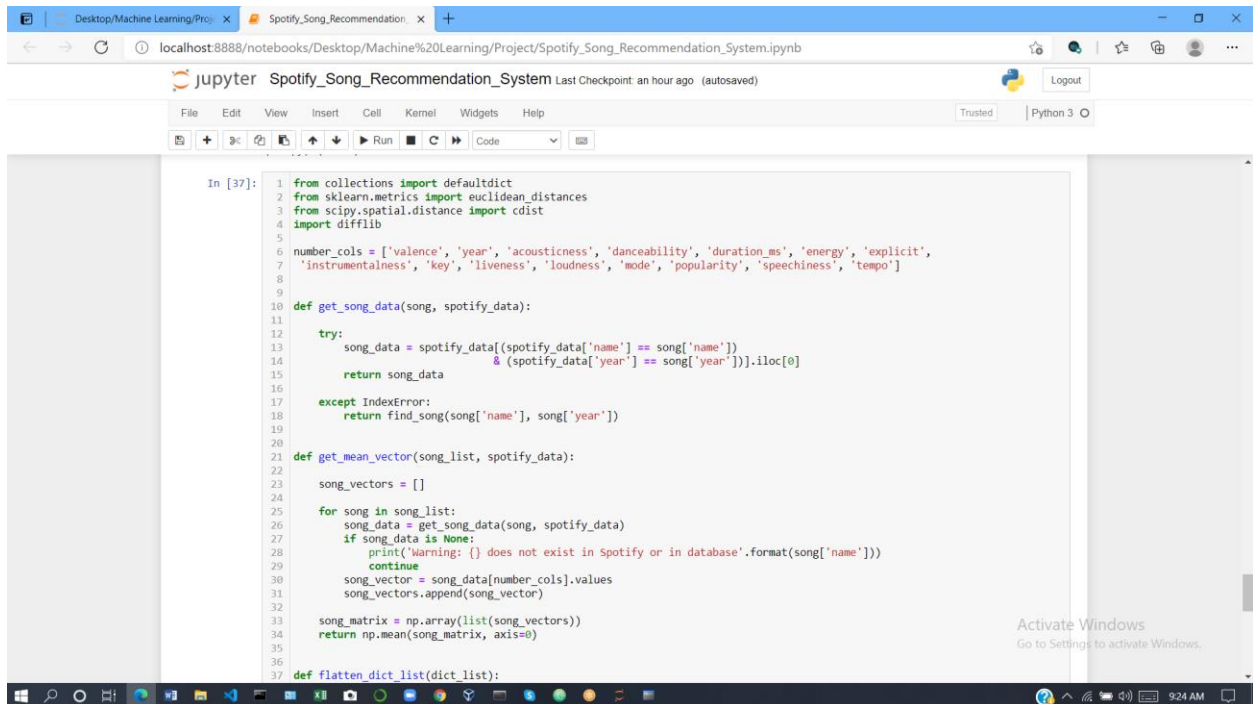
Now we can finally build the music recommendation system. The recommendation algorithm I used to follows three steps:

1. **Compute the average vector of the audio and metadata features for each song the user has listened to.**
2. **Find the  $n$ -closest data points in the dataset to this average vector.**
3. **Take these  $n$  points and recommend the songs corresponding to them**



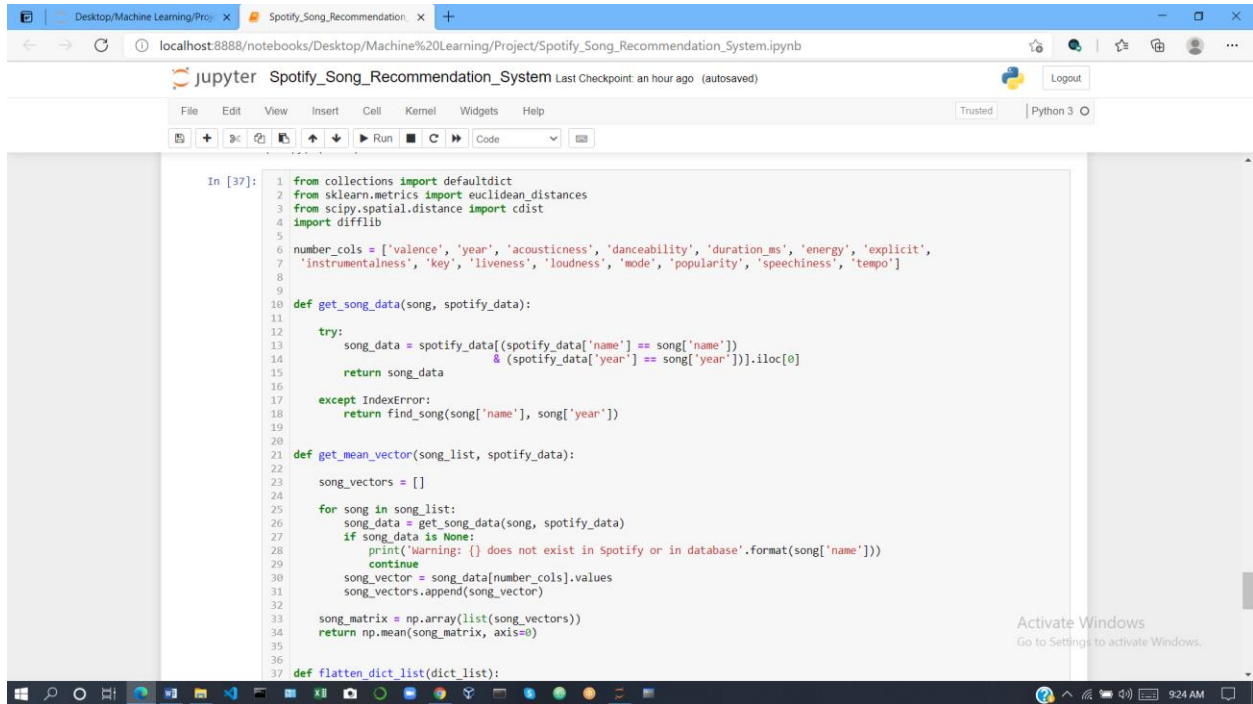


```
In [37]: 1 from collections import defaultdict
2 from sklearn.metrics import euclidean_distances
3 from scipy.spatial.distance import cdist
4 import difflib
5
6 number_cols = ['valence', 'year', 'acousticness', 'danceability', 'duration_ms', 'energy', 'explicit',
7 'instrumentalness', 'key', 'liveness', 'loudness', 'mode', 'popularity', 'speechiness', 'tempo']
8
9
10 def get_song_data(song, spotify_data):
11
12     try:
13         song_data = spotify_data[(spotify_data['name'] == song['name'])
14                                 & (spotify_data['year'] == song['year'])].iloc[0]
15         return song_data
16
17     except IndexError:
18         return find_song(song['name'], song['year'])
19
20
21 def get_mean_vector(song_list, spotify_data):
22
23     song_vectors = []
24
25     for song in song_list:
26         song_data = get_song_data(song, spotify_data)
27         if song_data is None:
28             print('Warning: {} does not exist in Spotify or in database'.format(song['name']))
29             continue
30         song_vector = song_data[number_cols].values
31         song_vectors.append(song_vector)
32
33     song_matrix = np.array(list(song_vectors))
34     return np.mean(song_matrix, axis=0)
35
36
37 def flatten_dict_list(dict_list):
```



```
In [37]: 1 from collections import defaultdict
2 from sklearn.metrics import euclidean_distances
3 from scipy.spatial.distance import cdist
4 import difflib
5
6 number_cols = ['valence', 'year', 'acousticness', 'danceability', 'duration_ms', 'energy', 'explicit',
7 'instrumentalness', 'key', 'liveness', 'loudness', 'mode', 'popularity', 'speechiness', 'tempo']
8
9
10 def get_song_data(song, spotify_data):
11
12     try:
13         song_data = spotify_data[(spotify_data['name'] == song['name'])
14                                 & (spotify_data['year'] == song['year'])].iloc[0]
15         return song_data
16
17     except IndexError:
18         return find_song(song['name'], song['year'])
19
20
21 def get_mean_vector(song_list, spotify_data):
22
23     song_vectors = []
24
25     for song in song_list:
26         song_data = get_song_data(song, spotify_data)
27         if song_data is None:
28             print('Warning: {} does not exist in Spotify or in database'.format(song['name']))
29             continue
30         song_vector = song_data[number_cols].values
31         song_vectors.append(song_vector)
32
33     song_matrix = np.array(list(song_vectors))
34     return np.mean(song_matrix, axis=0)
35
36
37 def flatten_dict_list(dict_list):
```

**recommend\_songs** function to specify their listening history and generate recommendations as shown below.



The screenshot shows a Jupyter Notebook titled "Spotify\_Song\_Recommendation\_System" running on a local host. The code defines several functions for song recommendation:

```
In [37]: 1 from collections import defaultdict
2 from sklearn.metrics import euclidean_distances
3 from scipy.spatial.distance import cdist
4 import difflib
5
6 number_cols = ['valence', 'year', 'acousticness', 'danceability', 'duration_ms', 'energy', 'explicit',
7 'instrumentalness', 'key', 'liveness', 'loudness', 'mode', 'popularity', 'speechiness', 'tempo']
8
9
10 def get_song_data(song, spotify_data):
11
12     try:
13         song_data = spotify_data[(spotify_data['name'] == song['name'])
14                                 & (spotify_data['year'] == song['year'])].iloc[0]
15         return song_data
16
17     except IndexError:
18         return find_song(song['name'], song['year'])
19
20
21 def get_mean_vector(song_list, spotify_data):
22
23     song_vectors = []
24
25     for song in song_list:
26         song_data = get_song_data(song, spotify_data)
27         if song_data is None:
28             print('Warning: {} does not exist in Spotify or in database'.format(song['name']))
29             continue
30         song_vector = song_data[number_cols].values
31         song_vectors.append(song_vector)
32
33     song_matrix = np.array(list(song_vectors))
34     return np.mean(song_matrix, axis=0)
35
36
37 def flatten_dict_list(dict_list):
```

# CONCLUSION

We had a great learning experience doing this project. we have learn about data mining and data cleaning .this is the very first task of a machine learning model to remove all the problem creating objects from the dataset. data cleaning and data exploration were very useful to making dataset algorithm ready .we have learnt to create machine learning model ,train the model and then doing testing on it .most common algorithms used before by others were content and collaborative but the hybrid of the two gave extraordinary results. Best suited algorithm for this project was random forest algorithm.

## Future work

Due to lack of time we couldn't make a model using singular value decomposition and support vector machine .Popularity based models are also good at recommendations so we'll try to implement this also to predict top-N songs to the users which are most popular at a given time.

There are of the recommender system is vast and covers various parameters. It is developing and emerging in modern day generation of e services and commerce. But simultaneously there is a need to develop and optimize the working and output of recommender system. Several service providers facilitate the users with a list of items . But this is not sufficient because customers have different preferences and choices which may mainly depend upon various factors and constraints. Also in many cases it may not be possible to recommend specific items to particular users. Therefore there is a scope for incorporating the concept of multiple dimensions in music recommender system particularly.

Most of the products and services provided by the various e commerce sites are expensive and therefore less used by customers. This leads to the inability to rate an item or set of items correctly and specifically. Thus traditional recommender system techniques are not optimum. This lays a way towards further work and development in building an optimized recommender system which also takes into considerations the feedback and all other constraints related to recommendation.