Spotify Song Hit Prediction

A PROJECT REPORT

for

SOFTWARE METRICS (SWE2020)

in

M.Tech (Software Engineering)

by

Dev Swaika (19MIS0411)

N.Madhu Hasitha (20MIS0203)

Papareddy Anuhya (20MIS0427)

Fall Semester, 2022-23

Under the Guidance of
Prof. Krithika L.B

Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology and Engineering

Nov, 2022

DECLARATION BY THE CANDIDATE

We here by declare that the project report entitled **“Spotify Song Hit Prediction”** submitted by us to Vellore Institute of Technology University, Vellore in partial fulfillment of the requirement for the award of the course **Software Metrics (SWE2020)** is a record of bonafide project work carried out by us under the guidance of **Prof. Krithika L.B.** We further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other course.

Place: Vellore   Signature

Date: 10th November, 2022

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Information Technology & Engineering [SITE]

CERTIFICATE

This is to certify that the project report entitled **"Spotify Song Hit Prediction"** submitted by **Dev Swaika (19MIS0411), N.Madhu Hasitha (20MIS0203) and Papareddy Anuhya (20MIS0427)** to Vellore Institute of Technology University, Vellore in partial fulfillment of the requirement for the award of the course **Software Metrics (SWE2020)** is a record of bonafide work carried out by them under my guidance.

Prof. Krithika L.B

GUIDE

Spotify Song Hit Prediction

Dev Swaika[1], N.Madhu Hasitha[2], Papareddy Anuhya[3]

[1,2,3] Department of Information Technology, VIT University, Vellore, Tamil Nadu, India

Abstract

Music is such an important aspect in our lives. For most people, it is part of their daily routine. In this work, we attempt to predict the Spotify Hit Song list by using machine learning models, which aim to predict which songs will become chart-topping hits. We constructed a dataset with approximately 10 decades from 1969 to 2010 of hit and non-hit songs and extracted their audio features using the Spotify Web API. We test eight models on our dataset. Our best model was random forest, which was able to predict Billboard song success with 88% accuracy.

Introduction

It was found that there are elements beyond technical data points that could predict a song being hit or not. Music prediction is yet not a data science activity. This research uses two types of data points related to a song -audio features of a track and the sentimental analysis of the lyrics. Data from various sources like Spotify, Billboard, open data lyrics were aggregated and subjected to machine learning algorithms to identify if a formula for hit song exists. It was found that the accuracies reported in previous studies with posteriori properties cannot be attained with Apriorism, and it might be a tough task to get a magic formula when the prediction of a hit song is considered **[1]**. Prediction model for determining whether a piece of music is popular in the mainstream and using machine learning to classify songs based on their popularity. The researcher utilized the PCA (Principal Component Analyzer) module from SK-learn to complete the system. PCA determines only two variables as principal components, allowing the data to be refined for Clustering once again. The scatter plot data based on their clusters led to their designation as variables. Following the decomposition and pre-processing, the researcher utilizes k-means to cluster the data **[2]**. Mathematical model of the hit phenomenon is also applicable to the prediction reputation of musicians in the world, the box office of the concert. The total number of the postings would not be enough for the estimation of the popularity of the entertainers and concerts. If an entertainers or concerts is very excellent, the number of SNS posting does not decrease rapidly after a certain his/her event and their news. In addition, there may be differences in the reaction depending on the type of SNS to use of people. But Many people will post their opinion for the event, and they reply to the responses to other persons on SNS. Thus, change of the day for each of the number of postings is important to estimate the popularity of concerts **[3]**. Machine learning models were used to develop a good prediction model. Six models, i.e., Random Forest Classifier, SVM, Decision Tree Classifier, K-Nearest Neighbours, Logistic Regression and Naïve Bayes were tried on the dataset. In addition to sentiment analysis, profanity analysis was also carried out to gain a better understand of whether profanity affects the popularity of songs. For this, an array containing around 300 of the most used slang words was created and a counter was incremented for each occurrence of these in the lyrics of a song. It was evident that majority of the tracks had 0 profanity, and that most of the popular songs also contained a low profanity value. A threshold was set for the value of track popularity. Songs having a popularity greater than 66.5 were assigned 1 and the others 0 **[4].**

I. Problem Identified:

Spotify is a popular music streaming platform which has millions of songs. Not every song which is there on Spotify is a hit song. There are several factors which determine whether is song can be deemed as hit or not. As a popular music streaming platform Spotify must analyze those several factors and predict whether the song will be a hit. That way they can also improve their recommendation system.

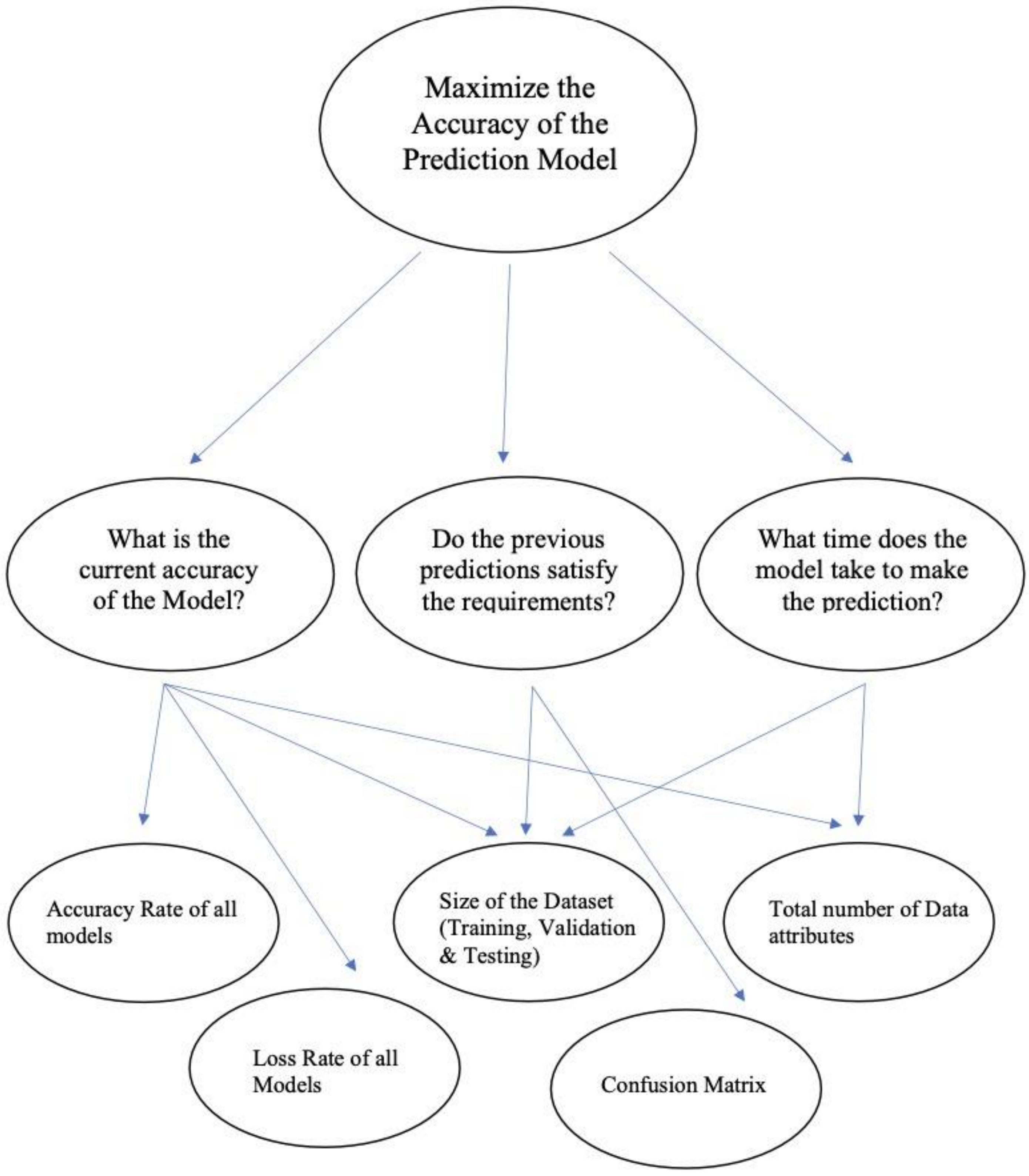
II. Problem to be Investigated:

The dataset used in our system contains details of every song from the 1960s to 2010s that was released on Spotify. It contains attributes like Danceability, Loudness, Energy, Mode, etc. With the help of these attributes, we can train models using ML and Neural Network Concepts to predict if the song released will be a hit or a flop. Hence accuracy of the model is very much necessary to be able to predict well.

III. Goal-Question-Metric (GQM):

The “Goal-Question-Metric” (GQM) approach is a proven method for driving goal-oriented measures throughout a software organization. With GQM, we start by defining the goals we are trying to achieve, then clarifying the questions we are trying to answer with the data we collect. By mapping business outcomes and goals to data-driven metrics, we can form a holistic picture of the Agile environment and clearly articulate how we are doing across the span of the enterprise.

Goal-Question-Metric (GQM)


Maximize the Accuracy of the Prediction Model
What is the current accuracy of the Model?
Do the previous predictions satisfy the requirements?
What time does the model take to make the prediction?
Accuracy Rate of all models
Loss Rate of all Models
Size of the Dataset (Training, Validation & Testing)
Confusion Matrix
Total number of Data attributes

IV. Radon Tool: What's Radon?

Radon is a Python tool which computes various code metrics. Radon will run from **Python 2.7** to **Python 3.8** (except Python versions from 3.0 to 3.3) with a single code base and without the need of tools like 2to3 or six. It can also run on **PyPy** without any problems

Radon depends on as few packages as possible. Currently only mando is strictly required (for the CLI interface). colorama is also listed as a dependency but if Radon cannot import it, the output simply will not be colored. Radon has a set of functions and classes that you can call from within your program to analyze files. Radon's API is composed of three layers: at the very bottom (the lowest level) there are the Visitors: with these classes one can build an AST out of the code and get basic metrics.

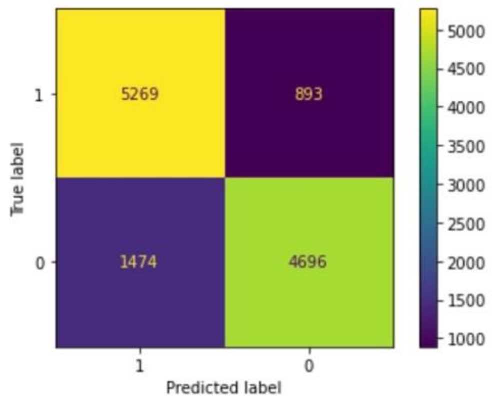
Radon Supported metrics are:

- **raw metrics**: SLOC, comment lines, blank lines, &c.
- **Cyclomatic Complexity** (i.e. McCabe's Complexity)
- **Halstead metrics** (all of them)
- **the Maintainability Index**

Radon can be used either from the command line or programmatically through its API.

V. Metric Analysis:

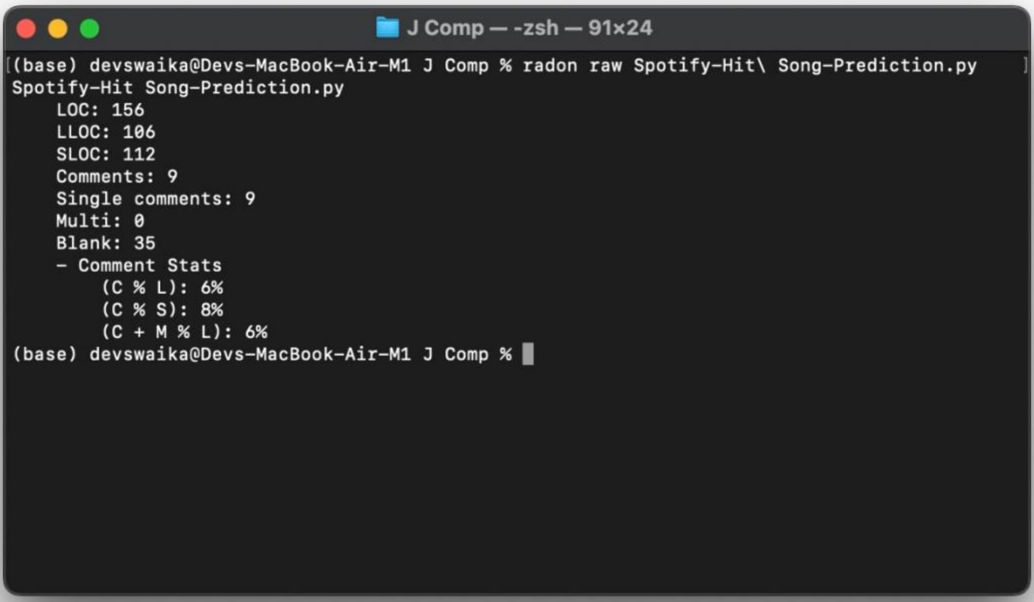
1. Confusion Matrix: A **Confusion matrix** is an ***N x N matrix*** used for evaluating the **performance of a classification model**, where **N** is the number of ***target classes***. The matrix compares the actual target values with those predicted by the machine learning model**.** t is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values (TP, FP, FN, TN).



- **True Positives (TP)**: when the actual value is Positive and predicted is also Positive.
- **True negatives (TN)**: when the actual value is Negative and prediction is also Negative.
- **False positives (FP)**: When the actual is negative but prediction is Positive. Also known as the **Type 1 error**
- **False negatives (FN)**: When the actual is Positive but the prediction is Negative. Also known as the **Type 2 error**

2. Raw metrics:

- **LOC**: The total number of lines of code. It does not necessarily correspond to the number of lines in the file.
- **LLOC**: The number of logical lines of code. Every logical line of code contains exactly one statement.
- **SLOC**: The number of source lines of code - not necessarily corresponding to the **LLOC**.
- Comments: The number of comment lines. Multi-line strings are not counted as comment since, to the Python interpreter, they are just strings.
- Multi: The number of lines which represent multi-line strings.
- Blanks: The number of blank lines (or whitespace-only ones).

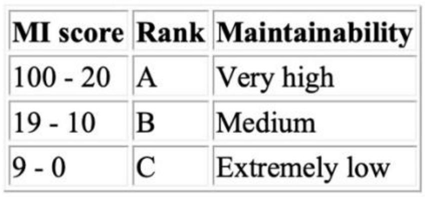


```
(base) devswaika@Devs-MacBook-Air-M1 J Comp % radon raw Spotify-Hit\ Song-Prediction.py
Spotify-Hit Song-Prediction.py
    LOC: 156
    LLOC: 106
    SLOC: 112
    Comments: 9
    Single comments: 9
    Multi: 0
    Blank: 35
    - Comment Stats
        (C % L): 6%
        (C % S): 8%
        (C + M % L): 6%
(base) devswaika@Devs-MacBook-Air-M1 J Comp %
```

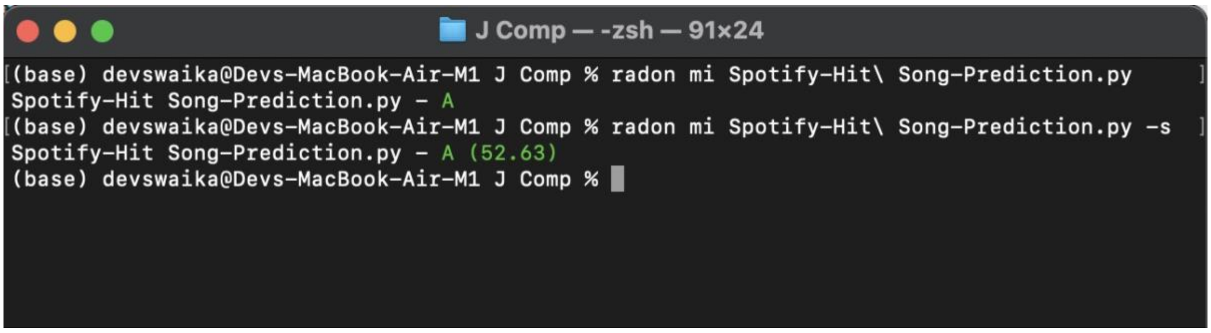
3. Maintainability Index: A software metric which measures how maintainable (easy to support and change) the source code is. The maintainability index is calculated as a factored formula consisting of SLOC (Source Lines Of Code), Cyclomatic Complexity and Halstead volume.

$$MI = \max\left[0, 100\,\frac{171 - 5.2\ln V - 0.23G - 16.2\ln L + 50\sin(\sqrt{2.4C})}{171}\right]$$

   a. `V` is the Halstead Volume (see below).
   b. `G` is the total Cyclomatic Complexity.
   c. `L` is the number of Source Lines of Code (SLOC).
   d. `C` is the percent of comment lines (important: converted to radians).



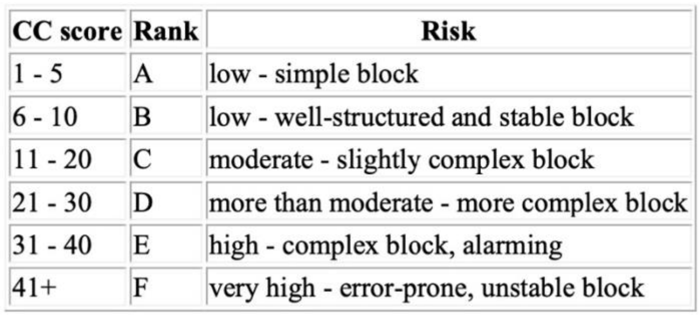
MI score	Rank	Maintainability
100 - 20	A	Very high
19 - 10	B	Medium
9 - 0	C	Extremely low



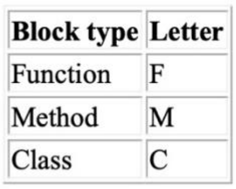
```
[(base) devswaika@Devs-MacBook-Air-M1 J Comp % radon mi Spotify-Hit\ Song-Prediction.py      ]
 Spotify-Hit Song-Prediction.py - A
[(base) devswaika@Devs-MacBook-Air-M1 J Comp % radon mi Spotify-Hit\ Song-Prediction.py -s   ]
 Spotify-Hit Song-Prediction.py - A (52.63)
 (base) devswaika@Devs-MacBook-Air-M1 J Comp %
```

4. Cyclomatic Complexity:

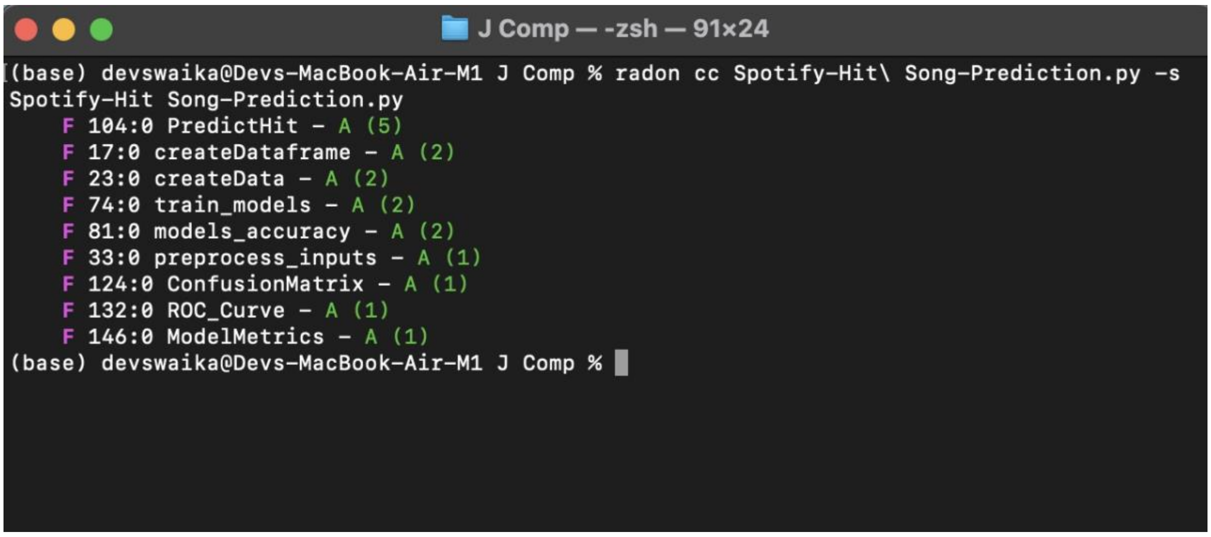
   Cyclomatic Complexity corresponds to the number of decisions a block of code contains plus 1. This number (also called McCabe number) is equal to the number of linearly independent paths through the code. This number can be used as a guide when testing conditional logic in blocks.



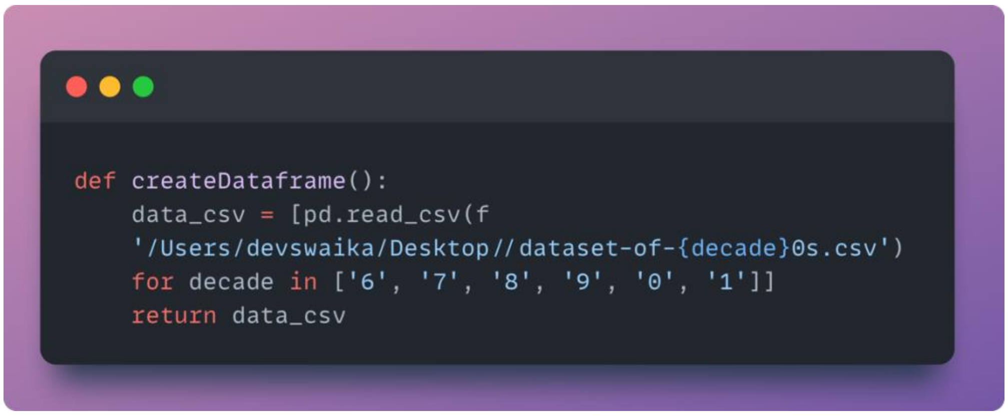
CC score	Rank	Risk
1 - 5	A	low - simple block
6 - 10	B	low - well-structured and stable block
11 - 20	C	moderate - slightly complex block
21 - 30	D	more than moderate - more complex block
31 - 40	E	high - complex block, alarming
41+	F	very high - error-prone, unstable block



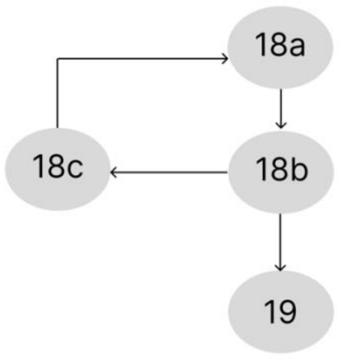
Block type	Letter
Function	F
Method	M
Class	C



```
(base) devswaika@Devs-MacBook-Air-M1 J Comp % radon cc Spotify-Hit\ Song-Prediction.py -s
Spotify-Hit Song-Prediction.py
    F 104:0 PredictHit - A (5)
    F 17:0 createDataframe - A (2)
    F 23:0 createData - A (2)
    F 74:0 train_models - A (2)
    F 81:0 models_accuracy - A (2)
    F 33:0 preprocess_inputs - A (1)
    F 124:0 ConfusionMatrix - A (1)
    F 132:0 ROC_Curve - A (1)
    F 146:0 ModelMetrics - A (1)
(base) devswaika@Devs-MacBook-Air-M1 J Comp %
```



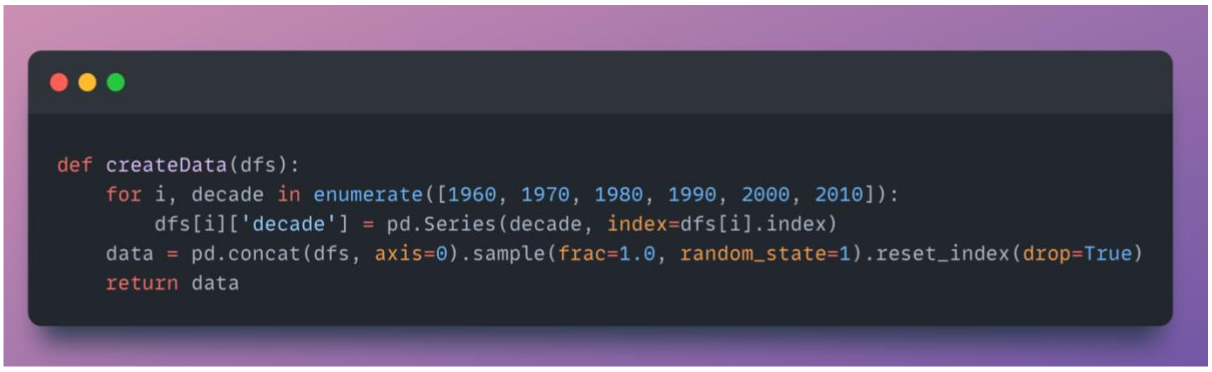
```python
def createDataframe():
    data_csv = [pd.read_csv(f
    '/Users/devswaika/Desktop//dataset-of-{decade}0s.csv')
    for decade in ['6', '7', '8', '9', '0', '1']]
    return data_csv
```



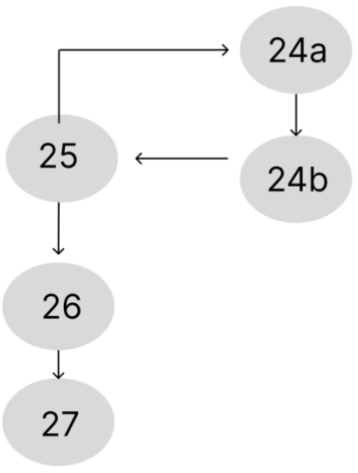
Method 1:
CC = Closed Region + 1 = 1+1 = 2

Method 2:
CC = E-N+2 = 4-4+2 = 2

Method 3:
CC = Predicate+1 = 1+1 = 2



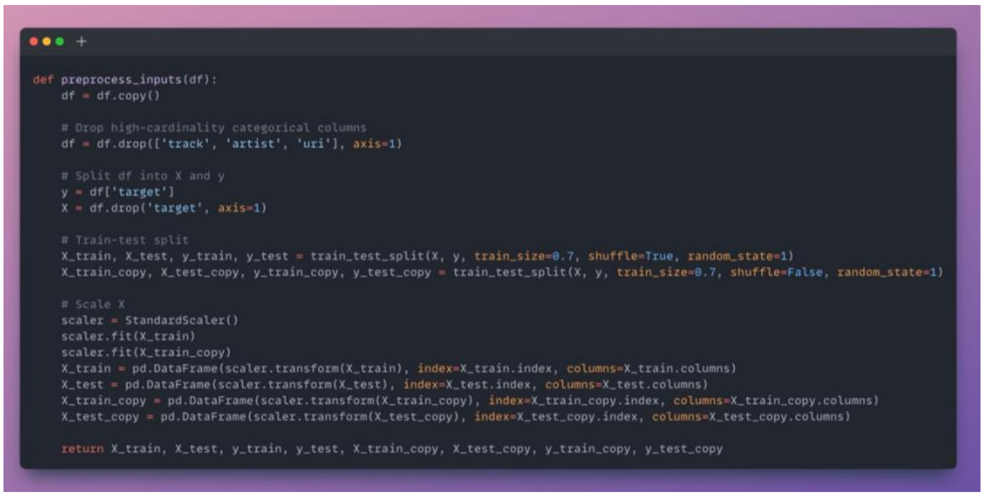
```python
def createData(dfs):
    for i, decade in enumerate([1960, 1970, 1980, 1990, 2000, 2010]):
        dfs[i]['decade'] = pd.Series(decade, index=dfs[i].index)
    data = pd.concat(dfs, axis=0).sample(frac=1.0, random_state=1).reset_index(drop=True)
    return data
```



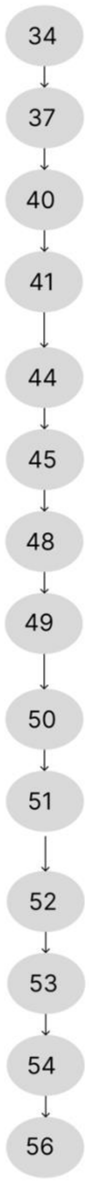
Method 1:
CC = Closed Region + 1 = 1+1 = 2

Method 2:
CC = E-N+2 = 5-5+2 = 2

Method 3:
CC = Predicate+1 = 1+1 = 2



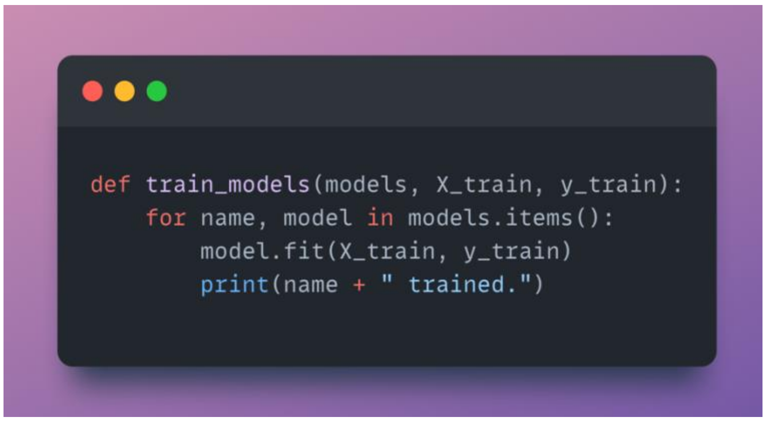
```python
def preprocess_inputs(df):
    df = df.copy()

    # Drop high-cardinality categorical columns
    df = df.drop(['track', 'artist', 'uri'], axis=1)

    # Split df into X and y
    y = df['target']
    X = df.drop('target', axis=1)

    # Train-test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, shuffle=True, random_state=1)
    X_train_copy, X_test_copy, y_train_copy, y_test_copy = train_test_split(X, y, train_size=0.7, shuffle=False, random_state=1)

    # Scale X
    scaler = StandardScaler()
    scaler.fit(X_train)
    scaler.fit(X_train_copy)
    X_train = pd.DataFrame(scaler.transform(X_train), index=X_train.index, columns=X_train.columns)
    X_test = pd.DataFrame(scaler.transform(X_test), index=X_test.index, columns=X_test.columns)
    X_train_copy = pd.DataFrame(scaler.transform(X_train_copy), index=X_train_copy.index, columns=X_train_copy.columns)
    X_test_copy = pd.DataFrame(scaler.transform(X_test_copy), index=X_test_copy.index, columns=X_test_copy.columns)

    return X_train, X_test, y_train, y_test, X_train_copy, X_test_copy, y_train_copy, y_test_copy
```



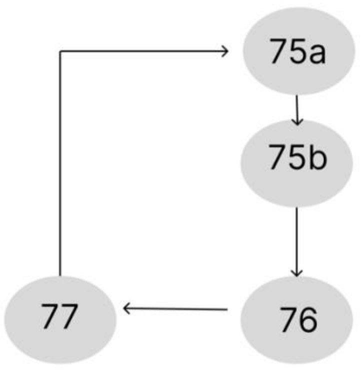
Method 1:
CC = Closed Region + 1 = 0 + 1 = 1

Method 2:
CC = E-N+2 = 13 – 14 + 2 = 1

Method 3:
CC = Predicate + 1 = 0 + 1 = 1



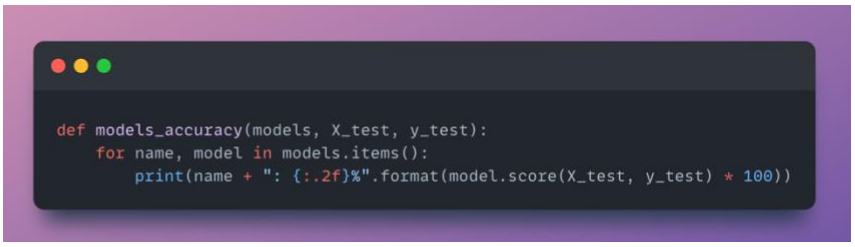
```
def train_models(models, X_train, y_train):
    for name, model in models.items():
        model.fit(X_train, y_train)
        print(name + " trained.")
```



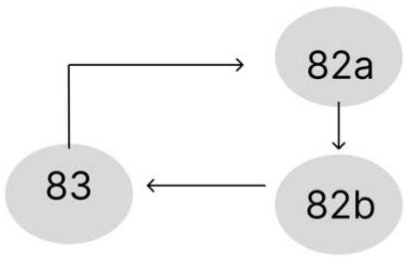
Method 1:
CC = Closed Region + 1 = 1 + 1 = 2

Method 2:
CC = E-N+2 = 4 – 4 + 2 = 2

Method 3:
CC = Predicate + 1 = 1 + 1 = 2



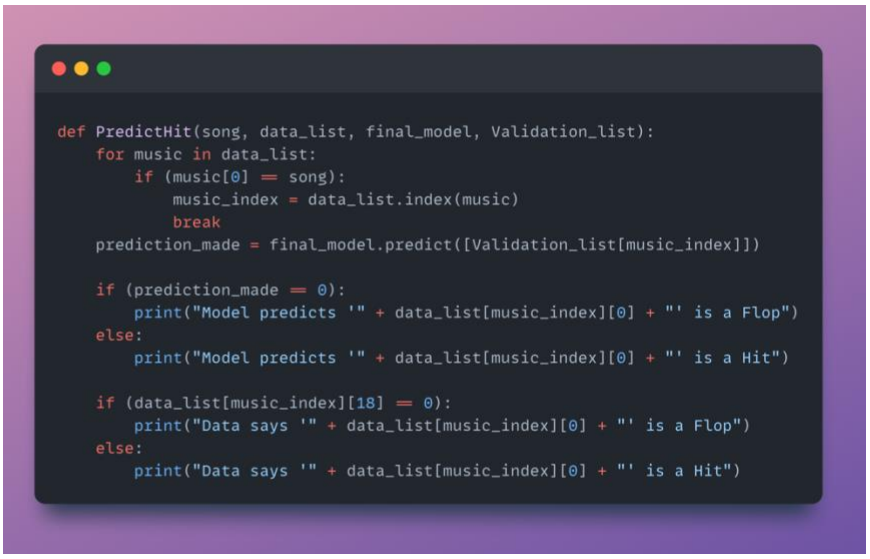
```
def models_accuracy(models, X_test, y_test):
    for name, model in models.items():
        print(name + ": {:.2f}%".format(model.score(X_test, y_test) * 100))
```



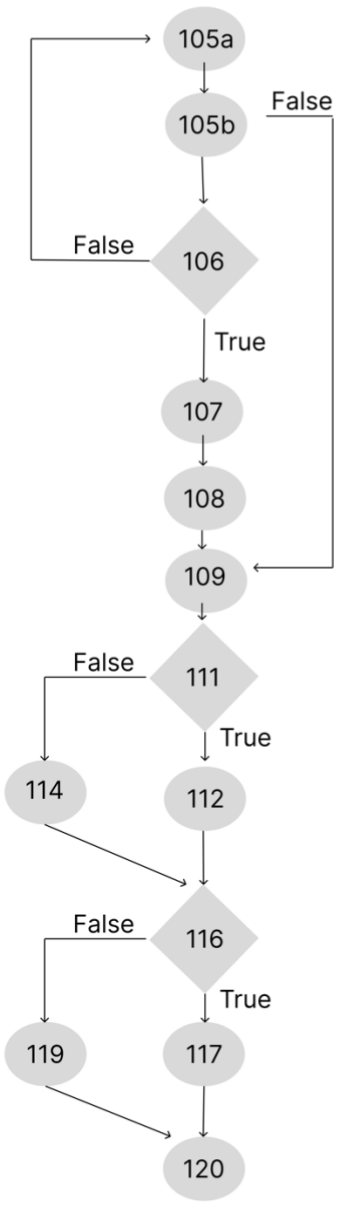
Method 1:
CC = Closed Region + 1 = 1 + 1 = 2

Method 2:
CC = E-N+2 = 3 – 3 + 2 = 2

Method 3:
CC = Predicate + 1 = 1 + 1 = 2



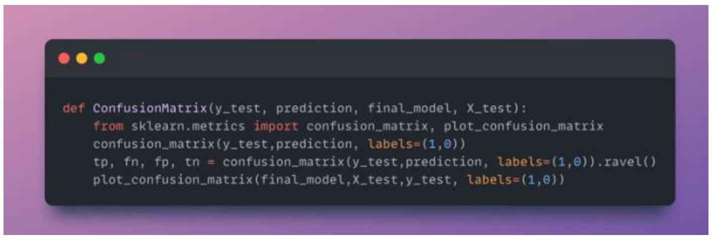
```
def PredictHit(song, data_list, final_model, Validation_list):
    for music in data_list:
        if (music[0] == song):
            music_index = data_list.index(music)
            break
    prediction_made = final_model.predict([Validation_list[music_index]])

    if (prediction_made == 0):
        print("Model predicts '" + data_list[music_index][0] + "' is a Flop")
    else:
        print("Model predicts '" + data_list[music_index][0] + "' is a Hit")

    if (data_list[music_index][18] == 0):
        print("Data says '" + data_list[music_index][0] + "' is a Flop")
    else:
        print("Data says '" + data_list[music_index][0] + "' is a Hit")
```



Method 1:
CC = Closed Region + 1 = 4 + 1 = 5

Method 2:
CC = E-N+2 = 16 – 13 + 2 = 5

Method 3:
CC = Predicate + 1 = 4 + 1 = 5



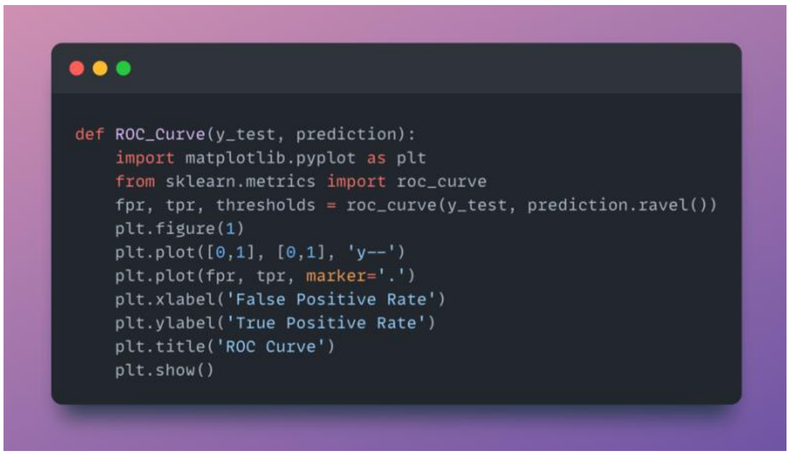
```python
def ConfusionMatrix(y_test, prediction, final_model, X_test):
    from sklearn.metrics import confusion_matrix, plot_confusion_matrix
    confusion_matrix(y_test,prediction, labels=(1,0))
    tp, fn, fp, tn = confusion_matrix(y_test,prediction, labels=(1,0)).ravel()
    plot_confusion_matrix(final_model,X_test,y_test, labels=(1,0))
```

$$\begin{tikzcd} 125 \arrow[d] \\ 126 \arrow[d] \\ 127 \arrow[d] \\ 128 \end{tikzcd}$$

Method 1:
CC = Closed Region + 1 = 0 + 1 = 1

Method 2:
CC = E-N+2 = 3 – 4 + 2 = 1

Method 3:
CC = Predicate + 1 = 0 + 1 = 1



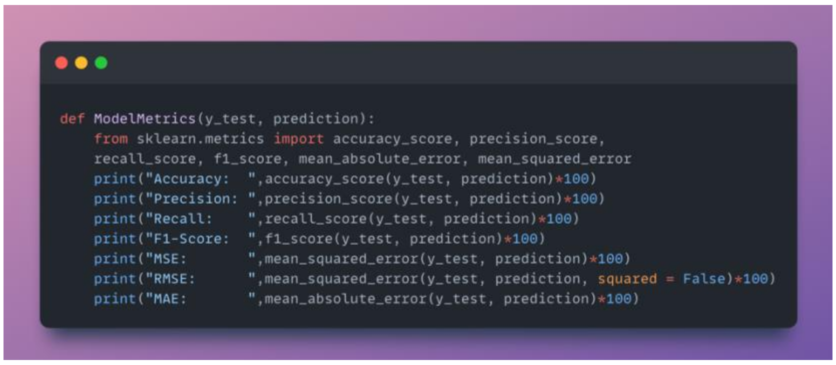
```python
def ROC_Curve(y_test, prediction):
    import matplotlib.pyplot as plt
    from sklearn.metrics import roc_curve
    fpr, tpr, thresholds = roc_curve(y_test, prediction.ravel())
    plt.figure(1)
    plt.plot([0,1], [0,1], 'y--')
    plt.plot(fpr, tpr, marker='.')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.show()
```

$$\begin{tikzcd} 133 \arrow[d] \\ 134 \arrow[d] \\ 135 \arrow[d] \\ 136 \arrow[d] \\ 137 \arrow[d] \\ 138 \arrow[d] \\ 139 \arrow[d] \\ 140 \arrow[d] \\ 141 \arrow[d] \\ 142 \end{tikzcd}$$

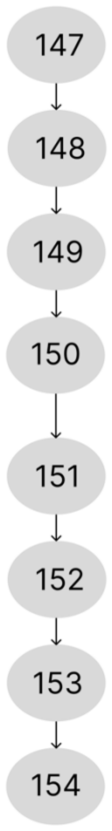
Method 1:
CC = Closed Region + 1 = 0 + 1 = 1

Method 2:
CC = E-N+2 = 9 – 10 + 2 = 1

Method 3:
CC = Predicate + 1 = 0 + 1 = 1



```python
def ModelMetrics(y_test, prediction):
    from sklearn.metrics import accuracy_score, precision_score,
    recall_score, f1_score, mean_absolute_error, mean_squared_error
    print("Accuracy:  ",accuracy_score(y_test, prediction)*100)
    print("Precision: ",precision_score(y_test, prediction)*100)
    print("Recall:    ",recall_score(y_test, prediction)*100)
    print("F1-Score:  ",f1_score(y_test, prediction)*100)
    print("MSE:       ",mean_squared_error(y_test, prediction)*100)
    print("RMSE:      ",mean_squared_error(y_test, prediction, squared = False)*100)
    print("MAE:       ",mean_absolute_error(y_test, prediction)*100)
```



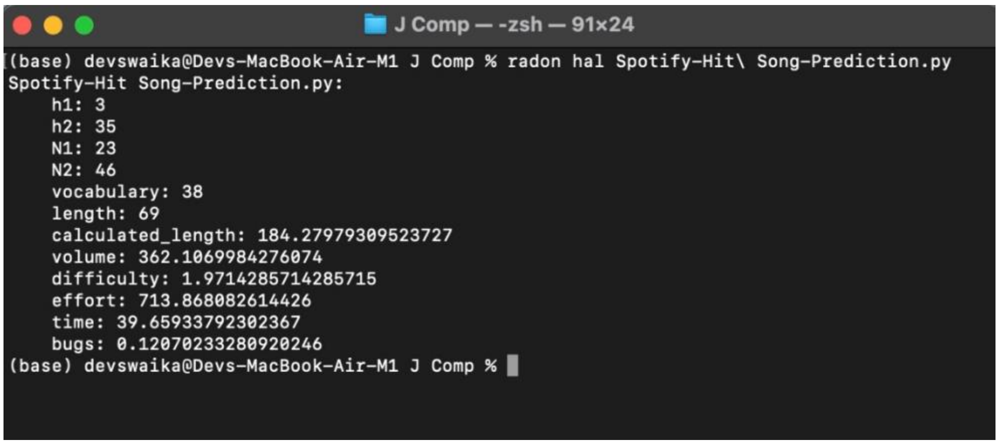
Method 1:
CC = Closed Region + 1 = 0 + 1 = 1

Method 2:
CC = E-N+2 = 7 – 8 + 2 = 1

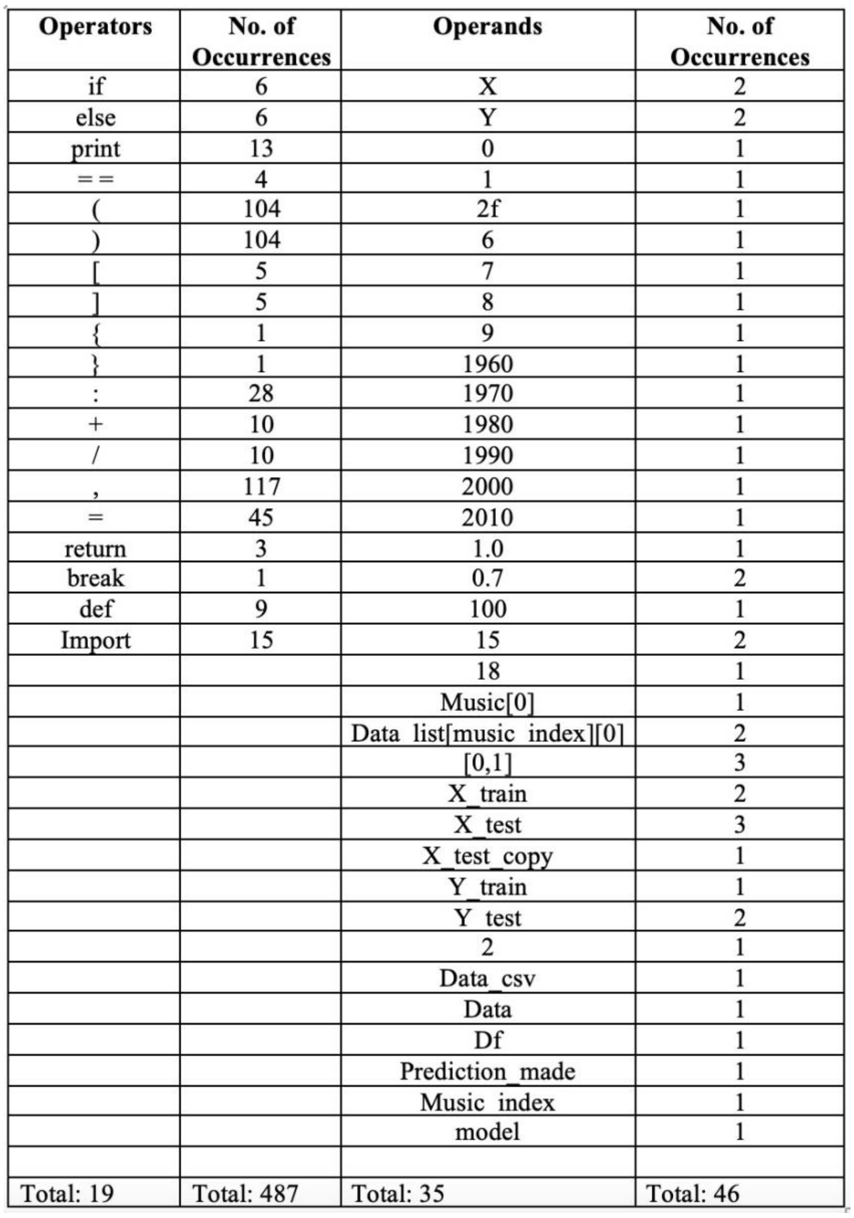
Method 3:
CC = Predicate + 1 = 0 + 1 = 1

5. Halstead Metrics:

Halstead's goal was to identify measurable properties of software, and the relations between them. These numbers are statically computed from the source code:



```
(base) devswaika@Devs-MacBook-Air-M1 J Comp % radon hal Spotify-Hit\ Song-Prediction.py
Spotify-Hit Song-Prediction.py:
    h1: 3
    h2: 35
    N1: 23
    N2: 46
    vocabulary: 38
    length: 69
    calculated_length: 184.27979309523727
    volume: 362.1069984276074
    difficulty: 1.9714285714285715
    effort: 713.868082614426
    time: 39.65933792302367
    bugs: 0.12070233280920246
(base) devswaika@Devs-MacBook-Air-M1 J Comp %
```



Operators	No. of Occurrences	Operands	No. of Occurrences
if	6	X	2
else	6	Y	2
print	13	0	1
= =	4	1	1
(	104	2f	1
)	104	6	1
[	5	7	1
]	5	8	1
{	1	9	1
}	1	1960	1
:	28	1970	1
+	10	1980	1
/	10	1990	1
,	117	2000	1
=	45	2010	1
return	3	1.0	1
break	1	0.7	2
def	9	100	1
Import	15	15	2
		18	1
		Music[0]	1
		Data_list[music_index][0]	2
		[0,1]	3
		X_train	2
		X_test	3
		X_test_copy	1
		Y_train	1
		Y_test	2
		2	1
		Data_csv	1
		Data	1
		Df	1
		Prediction_made	1
		Music_index	1
		model	1
Total: 19	Total: 487	Total: 35	Total: 46

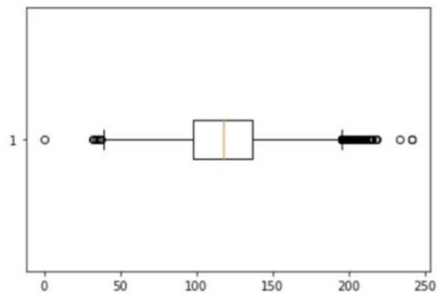
5. Box Plot:

In descriptive statistics, a box plot or boxplot is a method for graphically demonstrating the locality, spread and skewness groups of numerical data through their quartiles.
A boxplot is a standardized way of displaying the dataset based on the five-number summary: the minimum, the maximum, the sample median, and the first and third quartiles.

- Minimum (Q_0 or 0th percentile): the lowest data point in the data set excluding any outliers
- Maximum (Q_4 or 100th percentile): the highest data point in the data set excluding any outliers
- Median (Q_2 or 50th percentile): the middle value in the data set
- First quartile (Q_1 or 25th percentile): also known as the *lower quartile* $q_n(0.25)$, it is the median of the lower half of the dataset.
- Third quartile (Q_3 or 75th percentile): also known as the *upper quartile* $q_n(0.75)$, it is the median of the upper half of the dataset.

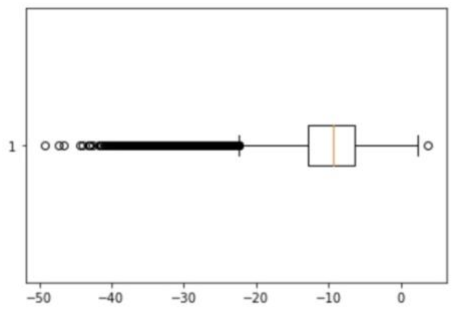
```
plt.boxplot(data['tempo'], meanline=True, vert=False)
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f952e965a30>,
  <matplotlib.lines.Line2D at 0x7f952e965d00>],
 'caps': [<matplotlib.lines.Line2D at 0x7f952e996be0>,
  <matplotlib.lines.Line2D at 0x7f952e996eb0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f952e965760>],
 'medians': [<matplotlib.lines.Line2D at 0x7f952e9a11c0>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f952e9a1490>],
 'means': []}
```



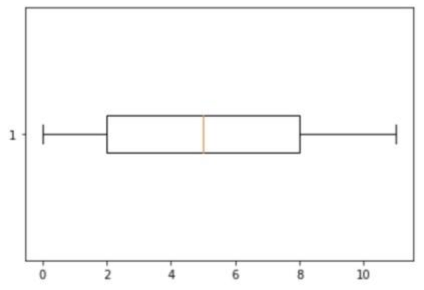
```
plt.boxplot(data['loudness'], meanline=True, vert=False)
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f950904f190>,
  <matplotlib.lines.Line2D at 0x7f950904f460>],
 'caps': [<matplotlib.lines.Line2D at 0x7f950904f730>,
  <matplotlib.lines.Line2D at 0x7f950904fa00>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f952f18de80>],
 'medians': [<matplotlib.lines.Line2D at 0x7f950904fcd0>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f950904ffa0>],
 'means': []}
```



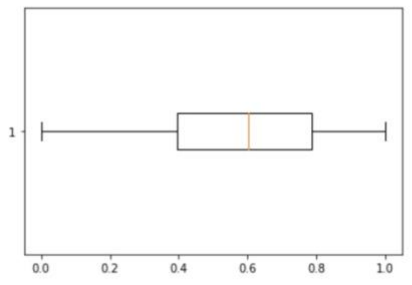
```
plt.boxplot(data['key'], meanline=True, vert=False)
```

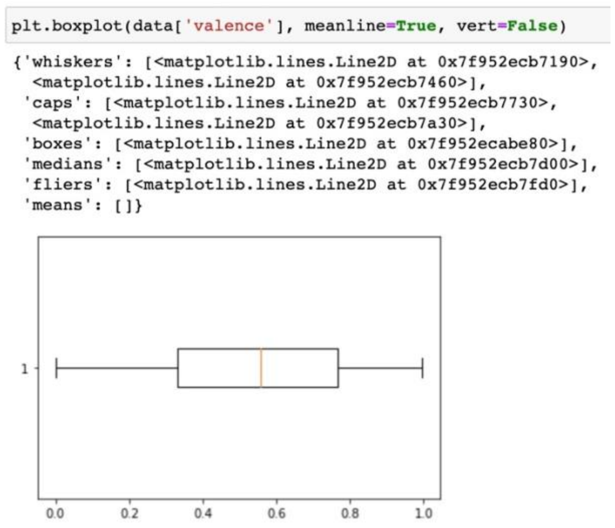
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f95090433a0>,
  <matplotlib.lines.Line2D at 0x7f9509043670>],
 'caps': [<matplotlib.lines.Line2D at 0x7f9509043940>,
  <matplotlib.lines.Line2D at 0x7f9509043c10>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f95090430d0>],
 'medians': [<matplotlib.lines.Line2D at 0x7f9509043ee0>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f952f1581f0>],
 'means': []}
```



```
plt.boxplot(data['energy'], meanline=True, vert=False)
```

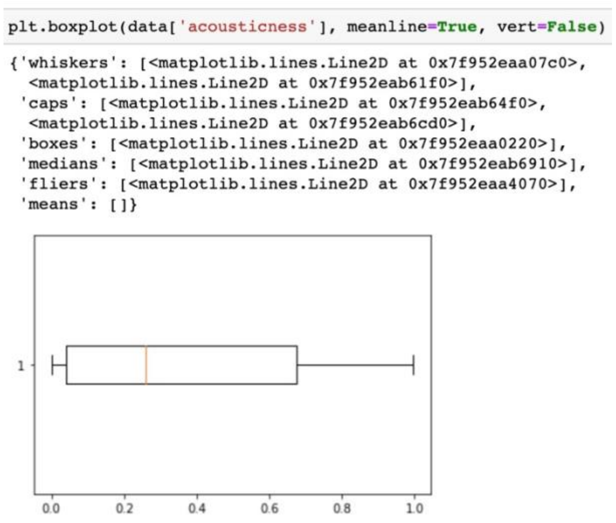
```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f952f2cf6d0>,
  <matplotlib.lines.Line2D at 0x7f952f2cf9a0>],
 'caps': [<matplotlib.lines.Line2D at 0x7f952f2cfc70>,
  <matplotlib.lines.Line2D at 0x7f952f2cff40>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f952f2cf400>],
 'medians': [<matplotlib.lines.Line2D at 0x7f952f2c6250>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f952f2c6520>],
 'means': []}
```





```
plt.boxplot(data['valence'], meanline=True, vert=False)
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f952ecb7190>,
  <matplotlib.lines.Line2D at 0x7f952ecb7460>],
 'caps': [<matplotlib.lines.Line2D at 0x7f952ecb7730>,
  <matplotlib.lines.Line2D at 0x7f952ecb7a30>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f952ecabe80>],
 'medians': [<matplotlib.lines.Line2D at 0x7f952ecb7d00>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f952ecb7fd0>],
 'means': []}
```

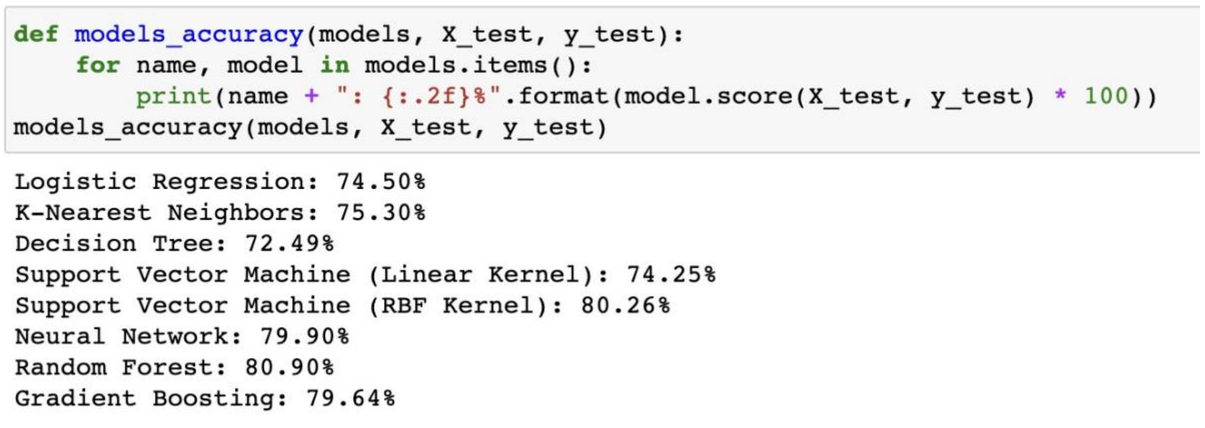


```
plt.boxplot(data['acousticness'], meanline=True, vert=False)
```

```
{'whiskers': [<matplotlib.lines.Line2D at 0x7f952eaa07c0>,
  <matplotlib.lines.Line2D at 0x7f952eab61f0>],
 'caps': [<matplotlib.lines.Line2D at 0x7f952eab64f0>,
  <matplotlib.lines.Line2D at 0x7f952eab6cd0>],
 'boxes': [<matplotlib.lines.Line2D at 0x7f952eaa0220>],
 'medians': [<matplotlib.lines.Line2D at 0x7f952eab6910>],
 'fliers': [<matplotlib.lines.Line2D at 0x7f952eaa4070>],
 'means': []}
```

Attribute	Median	1st Quartile	3rd Quartile	Outliers
Accouticness	0.25	0.05	0.7	None
Valence	0.55	0.35	0.8	None
Energy	0.6	0.4	0.8	None
Key	5	2	8	None
Loudness	-9	-13	-7	Many
Tempo	120	100	140	Many

Machine learning models



```
def models_accuracy(models, X_test, y_test):
    for name, model in models.items():
        print(name + ": {:.2f}%".format(model.score(X_test, y_test) * 100))
models_accuracy(models, X_test, y_test)
```

```
Logistic Regression: 74.50%
K-Nearest Neighbors: 75.30%
Decision Tree: 72.49%
Support Vector Machine (Linear Kernel): 74.25%
Support Vector Machine (RBF Kernel): 80.26%
Neural Network: 79.90%
Random Forest: 80.90%
Gradient Boosting: 79.64%
```

- Random Forest: Random Forest is a commonly used machine learning algorithm trademarked by Leo Breiman and Adele Cutler, which combines the output of multiple decision trees to reach a single result. Its ease of use and flexibility have fuelled its adoption, as it handles both classification and regression problems. Accuracy achieved by this model was 80.9%

- Linear Regression Model: Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Accuracy achieved by this model was 74.5%

- K-nearest neighbour's: K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories. Accuracy achieved by this model was 75.3%

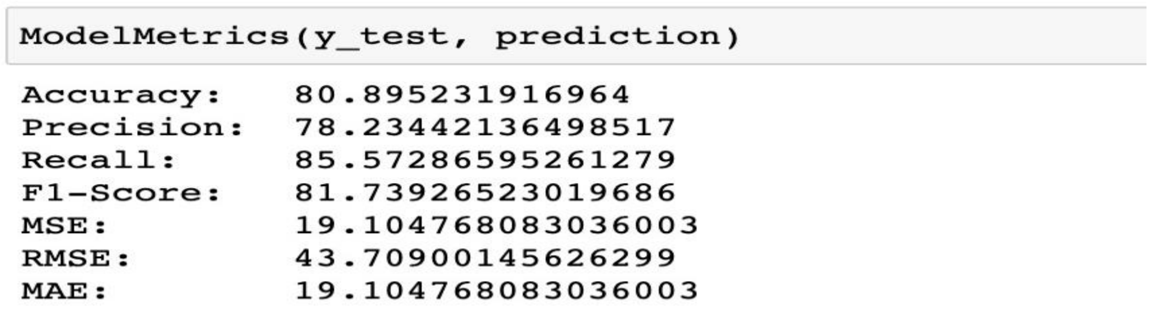
- Decision tree model: Decision Tree is a tree-like graph where sorting starts from the root node to the leaf node until the target is achieved. It is the most popular one for decision and classification based on supervised algorithms. Accuracy achieved by this model was 72.5%

- Support Vector Machine (SVM) Linear Kernel: Linear Kernel is used when the data is Linearly separable, that is, it can be separated using a single Line. It is one of the most common kernels to be used. It is mostly used when there are a Large number of Features in a particular Data Set. Accuracy achieved by this model was 74.25%

- Support Vector Machine (SVM) RBF Kernel: In machine learning, the radial basis function kernel, or RBF kernel, is a popular kernel function used in various kernelized learning algorithms. In particular, it is commonly used in support vector machine classification. Accuracy achieved by this model was 80.26%

- Neural Network: It is a series of algorithms that endeavours to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Accuracy achieved by this model was 79.9%

- Gradient boosting: is a type of machine learning boosting. It relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model to minimize the error. Accuracy achieved by this model was 79.64%



```
ModelMetrics(y_test, prediction)
```

```
Accuracy:    80.895231916964
Precision:   78.23442136498517
Recall:      85.57286595261279
F1-Score:    81.73926523019686
MSE:         19.104768083036003
RMSE:        43.70900145626299
MAE:         19.104768083036003
```

- Accuracy: Accuracy is defined as the correctly classified points by a total no of points on the test set.
- Precision : Precision is defined as the ratio of correctly classified positive samples (True Positive) to a total number of classified positive samples
- Recall: The recall is calculated as the ratio between the number of Positive samples correctly classified as Positive to the total number of Positive samples. The recall measures the model's ability to detect Positive samples. The higher the recall, the more positive samples detected.
- F1-Score: The F1 score is defined as the harmonic mean of precision and recall. As a short reminder, the harmonic mean is an alternative metric for the more common arithmetic mean. It is often useful when computing an average rate.
- Mean squared error: The mean squared error (MSE) determines the distance between the set of points and the regression line by taking the distances from the set of points to the regression line and then swapping them.
- Root mean square error: Root mean square error or root mean square deviation is one of the most commonly used measures for evaluating the quality of predictions. It shows how far predictions fall from measured true values using Euclidean distance.
- Mean absolute error: MAE (Mean Absolute Error) is the average absolute error between actual and predicted values.