```
from google.colab import files

uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

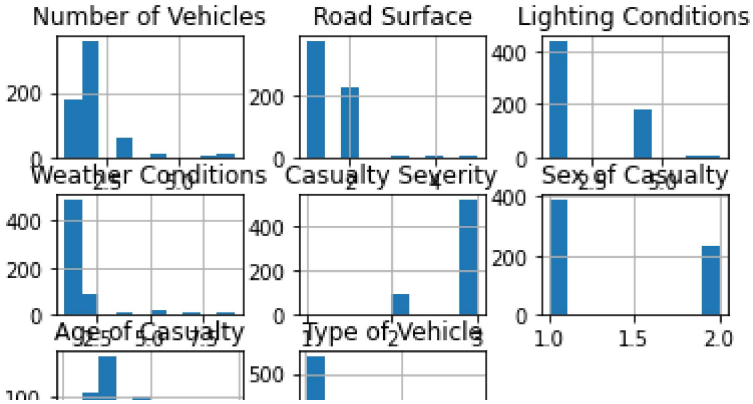
Saving accident-2014-4.csv to accident-2014-4.csv

```
import pandas as pd
import io
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
%matplotlib inline
import warnings
from sklearn.preprocessing import LabelEncoder
warnings.filterwarnings('ignore')

df = pd.read_csv(io.BytesIO(uploaded['accident-2014-4.csv']))
print(df.head().to_markdown())
```

	Reference Number	Grid Ref: Easting	Grid Ref: Northing	Number of Veh
0	11S0281	407491	428929	
1	11S1147	412912	429039	
2	11S1147	412912	429039	
3	11U0285	407179	424776	
4	11U0285	407179	424776	

```
#Drop the columns with the non numerical values.
df1 = df
num_df =df1.drop(['Grid Ref: Easting', 'Grid Ref: Northing','Reference Number','Accident Date
num_df.columns
num_df.hist()

# Bring the class attribute as the last column
# The casulaty Severity here has 3 values.. 1.Fatal, 2.Seriuos . 3.Slight

temp_series = num_df.pop('Casualty Severity')
num_df['Casualty Severity'] = temp_series
num_df.columns
```

```
Index(['Number of Vehicles', 'Road Surface', 'Lighting Conditions',
       'Weather Conditions', 'Sex of Casualty', 'Age of Casualty',
       'Type of Vehicle', 'Casualty Severity'],
      dtype='object')
```




```
df.nunique()
```

```
Reference Number       427
Grid Ref: Easting      410
Grid Ref: Northing     409
Number of Vehicles       7
Accident Date          253
Time (24hr)            261
1st Road Class           4
1st Road Class & No     24
Road Surface             5
Lighting Conditions      5
Weather Conditions       8
Local Authority          1
Casualty Class           3
Casualty Severity        3
Sex of Casualty          2
Age of Casualty         88
Vehicle Number           7
Type of Vehicle         12
dtype: int64
```

```
# importing the sklearn library
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_auc_score


# getting the x(dataset excluding the class attribute) and y(class attribute) dataset from cl
x = num_df.drop('Casualty Severity',axis=1)
y = num_df['Casualty Severity']

# Get dummy variable
y = pd.get_dummies(y).values
```

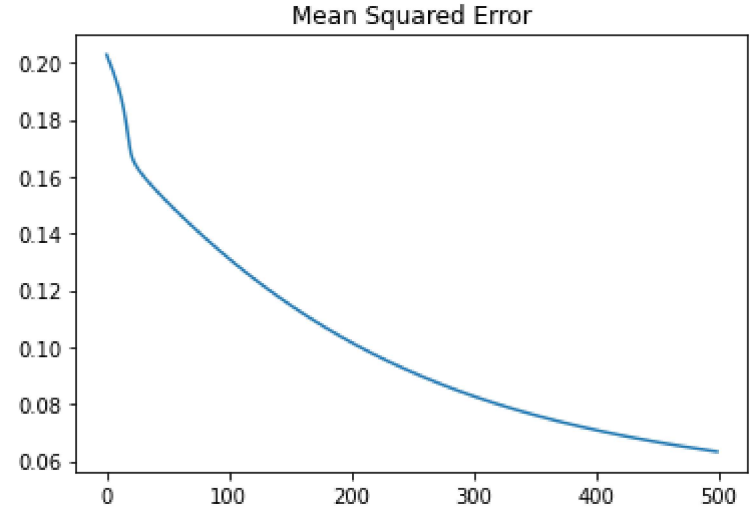
```
X_train, X_test, y_train, y_test= train_test_split(x, y, test_size= 0.25)

clf = MLPClassifier(hidden_layer_sizes=(100,),random_state=1, max_iter=300).fit(X_train, y_tr
clf.fit(X_train,y_train)
y_pred = clf.predict_proba(X_train)
print('Overall AUC:', roc_auc_score(y_train, clf.predict_proba(X_train),multi_class="ovr"))
```

```
Overall AUC: 0.7795678187519647
```

```
# y_pred = clf.predict(X_test)
# cm = confusion_matrix(y_test, y_pred)
# cm


# Initialize variables
learning_rate = 0.1
iterations = 500
N = y_train.size

# number of input features
input_size = 7

# number of hidden layers neurons
hidden_size = 2

# number of neurons at the output layer
output_size = 3

results = pd.DataFrame(columns=["mse", "accuracy"])


# Initialize weights
np.random.seed(10)

# initializing weight for the hidden layer
W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))

# initializing weight for the output layer
W2 = np.random.normal(scale=0.5, size=(hidden_size , output_size))


def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def mean_squared_error(y_pred, y_true):
    return ((y_pred - y_true)**2).sum() / (2*y_pred.size)

def accuracy(y_pred, y_true):
    acc = y_pred.argmax(axis=1) == y_true.argmax(axis=1)
    return acc.mean()
```

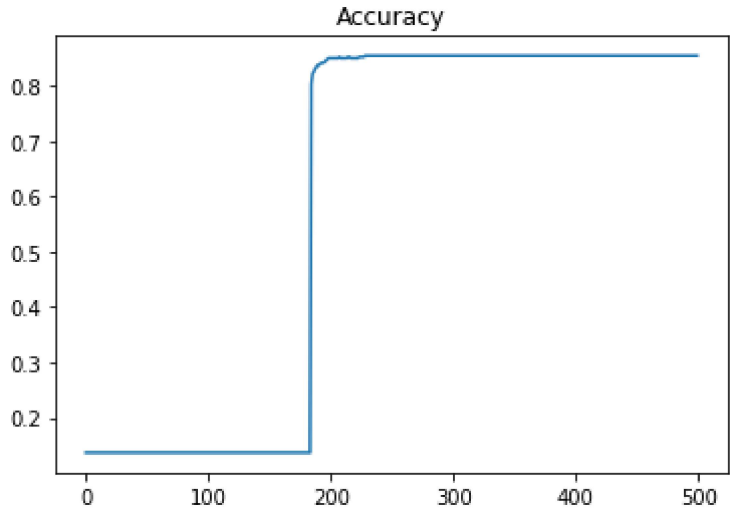
```
for itr in range(iterations):

    # feedforward propagation
    # on hidden layer
    Z1 = np.dot(X_train, W1)
    A1 = sigmoid(Z1)

    # on output layer
    Z2 = np.dot(A1, W2)
    A2 = sigmoid(Z2)


    # Calculating error
    mse = mean_squared_error(A2, y_train)
    acc = accuracy(A2, y_train)
    results=results.append({"mse":mse, "accuracy":acc},ignore_index=True )

    # backpropagation
    E1 = A2 - y_train
    dW1 = E1 * A2 * (1 - A2)

    E2 = np.dot(dW1, W2.T)
    dW2 = E2 * A1 * (1 - A1)


    # weight updates
    W2_update = np.dot(A1.T, dW1) / N
    W1_update = np.dot(X_train.T, dW2) / N

    W2 = W2 - learning_rate * W2_update
    W1 = W1 - learning_rate * W1_update


results.mse.plot(title="Mean Squared Error")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f32f813b2d0>
```



```
results.accuracy.plot(title="Accuracy")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f32f80a4c90>
```




```
y_pred = clf.predict_proba(X_train)
print('Overall AUC:', roc_auc_score(y_train,y_pred,multi_class="ovr"))
```

```
Overall AUC: 0.7795678187519647
```

```
import pandas as pd

df=pd.read_csv("rockmusic_2147239.csv")
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5484 entries, 0 to 5483
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   index             5484 non-null   int64  
 1   name              5484 non-null   object 
 2   artist            5484 non-null   object 
 3   release_date      5484 non-null   int64  
 4   length            5484 non-null   float64
 5   popularity        5484 non-null   int64  
 6   danceability      5484 non-null   float64
 7   acousticness      5484 non-null   float64
 8   danceability.1    5484 non-null   float64
 9   energy            5484 non-null   float64
 10  instrumentalness  5484 non-null   float64
 11  key               5484 non-null   int64  
 12  liveness          5484 non-null   float64
 13  loudness          5484 non-null   float64
 14  speechiness       5484 non-null   float64
 15  tempo             5484 non-null   float64
 16  time_signature    5484 non-null   int64  
 17  valence           5484 non-null   float64
dtypes: float64(11), int64(5), object(2)
memory usage: 771.3+ KB
```

```
a=df.corr()
a.style.text_gradient(cmap="Dark2")
```

	index	name	artist	release_date	length	popularity	dan
index	1.000000	-0.001033	-0.020613	0.165672	-0.047728	-0.386874	
name	-0.001033	1.000000	0.000921	0.010811	0.013942	0.006225	
artist	-0.020613	0.000921	1.000000	0.028908	-0.061020	-0.059490	
release_date	0.165672	0.010811	0.028908	1.000000	0.056641	-0.282614	
length	-0.047728	0.013942	-0.061020	0.056641	1.000000	0.023665	
popularity	-0.386874	0.006225	-0.059490	-0.282614	0.023665	1.000000	
danceability	-0.056157	-0.005092	-0.015624	-0.083073	-0.116905	0.114804	
acousticness	-0.073477	-0.011339	-0.011812	-0.360184	-0.055241	0.088253	
danceability.1	-0.056157	-0.005092	-0.015624	-0.083073	-0.116905	0.114804	
energy	0.070377	-0.001250	0.014250	0.288541	-0.045376	-0.081577	
instrumentalness	0.001021	-0.003752	0.041001	0.023215	0.228365	-0.091788	
key	0.022695	0.023421	-0.032017	0.030631	0.011435	0.032483	
liveness	0.021272	0.003252	0.007299	0.084054	0.010616	-0.115119	
loudness	0.050855	0.002196	-0.006799	0.414275	-0.091621	-0.012989	

```
df.head()
```

	index	name	artist	release_date	length	popularity	danceab
0	0	3931	942	35	3827	74	
1	1	4069	772	15	4639	78	
2	2	610	1045	19	4316	74	
3	3	2201	704	15	995	77	

```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()

cols = ['index','name','artist', 'release_date','length','popularity','danceability','acousti
df[cols] = df[cols].apply(LabelEncoder().fit_transform)
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
```

```
df['artist']=labelencoder.fit_transform(df['artist'])
df.head(1)
x= df.iloc[:, [9,13]].values
y= df.iloc[:, 5].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
import numpy as nm
#features = nm.array([[4,5]])
features = nm.array([[752,4557]])
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)

# using inputs to predict the output
classifier.fit(x_train, y_train)
prediction = classifier.predict(features)

print("Popularity: {}".format(prediction))
```

```
Popularity: [36]
```

```
x= df.iloc[:,[4]].values
y= df.iloc[:,[5]].values
```

```
x.shape
```

```
(5484, 1)
```

```
y.shape
```

```
(5484, 1)
```

```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
a_train, a_test, b_train, b_test= train_test_split(x,y, test_size= 0.50, random_state=42)
```

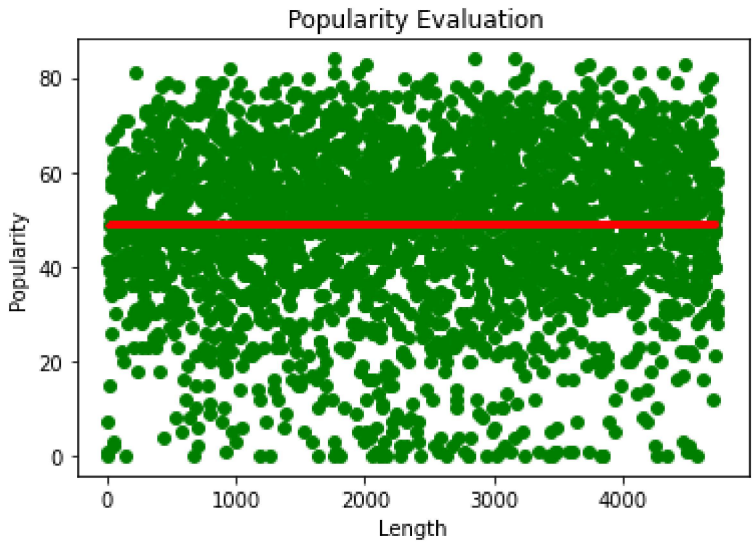
```
#Fitting the Simple Linear Regression model to the training dataset
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
```

```
regressor.fit(a_train, b_train)
```

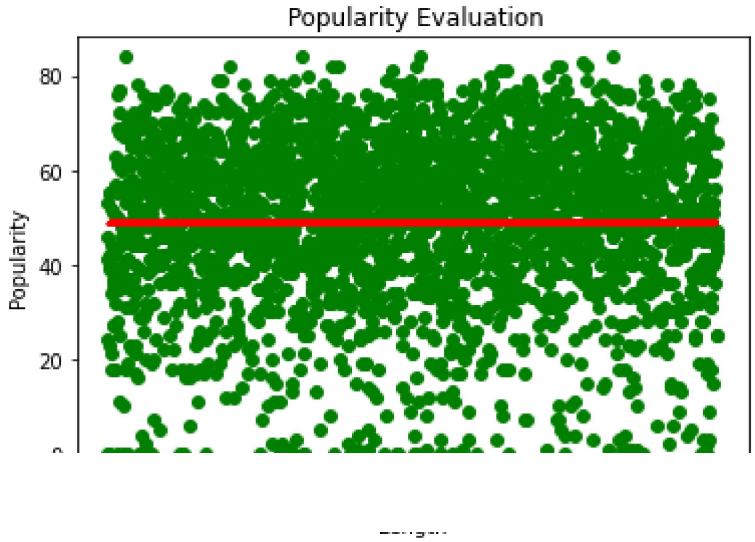
```
LinearRegression()
```

```
#Prediction of Test and Training set result
a_pred= regressor.predict(a_test)
b_pred= regressor.predict(a_train)
```

```
import numpy as nm
import matplotlib.pyplot as mtp

mtp.scatter(a_train, b_train, color="green")
mtp.plot(a_train, a_pred, color="red")
mtp.title(" Popularity Evaluation")
mtp.xlabel("Length")
mtp.ylabel("Popularity")
mtp.show()
```




```
import numpy as nm
import matplotlib.pyplot as mtp

mtp.scatter(a_test, b_test, color="green")
mtp.plot(a_train, a_pred, color="red")
mtp.title(" Popularity Evaluation")
mtp.xlabel("Length")
mtp.ylabel("Popularity")
mtp.show()
```



```
#LOGISTIC REGRESSION for HCV

#Logistic regression is an example of supervised learning. It is used to calculate or predict


from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell

```
import pandas as pd
import io
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings
from sklearn.preprocessing import LabelEncoder

%matplotlib inline

warnings.filterwarnings('ignore')



df=pd.read_csv('hcvdat0.csv', index_col=0)
df.head()
```

	Category	Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	CR
1	0=Blood Donor	32	m	38.5	52.5	7.7	22.1	7.5	6.93	3.23	10(
2	0=Blood Donor	32	m	38.5	70.3	18.0	24.7	3.9	11.17	4.80	7.

```
#Category is dependent variable

#Age    Sex ALB ALP ALT AST BIL CHE CHOL    CREA    GGT PRO are independent variable


# # Drop column of index using DataFrame.iloc[] and drop() methods.
# df2 = df.drop(df.iloc[:, 0:1],axis = 1)
```

```
# print(df2)
# df2.shape
```

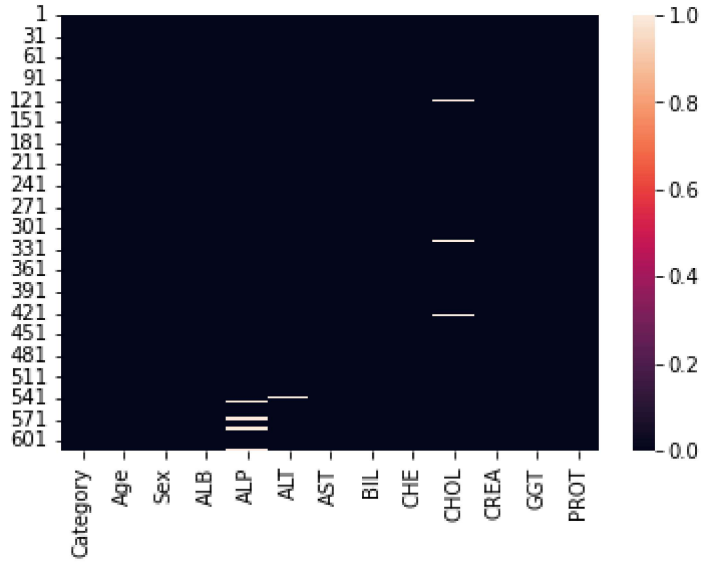
```
df.nunique()
```

```
Category      5
Age          49
Sex           2
ALB         189
ALP         414
ALT         341
AST         297
BIL         188
CHE         407
CHOL        313
CREA        117
GGT         358
PROT        198
dtype: int64
```

```
#some null values are shown here
```

```
sns.heatmap(df.isnull())
```

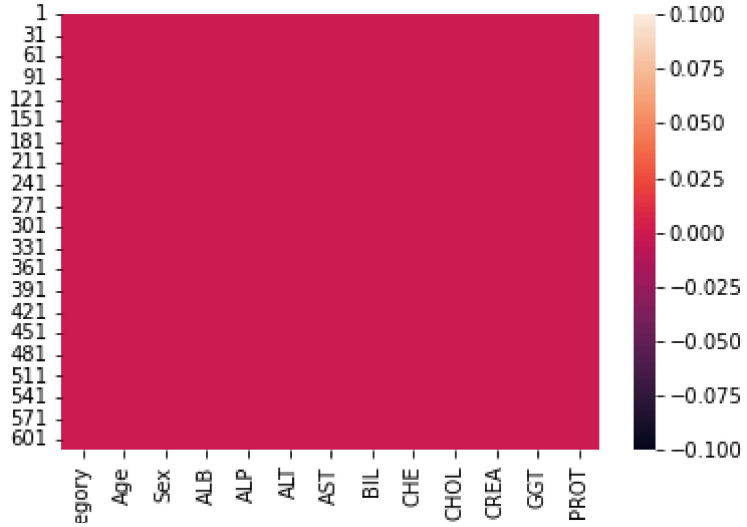
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5c4ecd0790>
```




```
df.fillna(0, inplace = True)
```

```
sns.heatmap(df.isnull())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5c4b323d90>
```




```
#According to heat map - ALP CHOL AND PROT has some null values hence should be replaced by a
```

```
df.head()
```

	Category	Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	CR
1	0=Blood Donor	32	m	38.5	52.5	7.7	22.1	7.5	6.93	3.23	10(
2	0=Blood Donor	32	m	38.5	70.3	18.0	24.7	3.9	11.17	4.80	7⁴

```
labelencoder = LabelEncoder()
new_df=df
new_df['Category']=labelencoder.fit_transform(df['Category'])
new_df['Sex']=labelencoder.fit_transform(df['Sex'])
```

```
new_df.head(2)
```

	Category	Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	CRI
1	0	32	1	38.5	52.5	7.7	22.1	7.5	6.93	3.23	106

```
x= new_df.iloc[:, 1:]
print(x)
```

```
     Age  Sex   ALB    ALP   ALT    AST   BIL    CHE  CHOL   CREA   GGT  \
1     32    1  38.5   52.5   7.7   22.1   7.5   6.93  3.23  106.0  12.1
```

```
2     32    1  38.5   70.3   18.0   24.7   3.9  11.17  4.80   74.0   15.6
3     32    1  46.9   74.7   36.2   52.6   6.1   8.84  5.20   86.0   33.2
4     32    1  43.2   52.0   30.6   22.6  18.9   7.33  4.74   80.0   33.8
5     32    1  39.2   74.1   32.6   24.8   9.6   9.15  4.32   76.0   29.9
..   ...  ...   ...    ...    ...    ...   ...    ...   ...    ...    ...
611   62    0  32.0  416.6    5.9  110.3  50.0   5.57  6.30   55.7  650.9
612   64    0  24.0  102.8    2.9   44.4  20.0   1.54  3.02   63.0   35.9
613   64    0  29.0   87.3    3.5   99.0  48.0   1.66  3.63   66.7   64.2
614   46    0  33.0    0.0   39.0   62.0  20.0   3.56  4.20   52.0   50.0
615   59    0  36.0    0.0  100.0   80.0  12.0   9.07  5.30   67.0   34.0

     PROT
1    69.0
2    76.5
3    79.3
4    75.7
5    68.7
..    ...
611  68.5
612  71.3
613  82.0
614  71.0
615  68.0

[615 rows x 12 columns]
```

```
y= new_df.iloc[:, 0]
print(y)
```

```
1      0
2      0
3      0
4      0
5      0
      ..
611    4
612    4
613    4
614    4
615    4
Name: Category, Length: 615, dtype: int64
```

```
#Extracting Independent and dependent Variable
#all rows from  1 column to last,  except column 0 that is Category
#all rows from  0 column ie Category
x= new_df.iloc[:, 1:].values
y= new_df.iloc[:, 0].values


# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

```
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)


#Fitting Logistic Regression to the training set
from sklearn.linear_model import LogisticRegression
classifier= LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
```

```
LogisticRegression(random_state=0)
```

```
#Predicting the test set result
y_pred= classifier.predict(x_test)


#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
cm
```

```
array([[135,   0,   0,   0,   0],
       [  2,   1,   0,   0,   1],
       [  1,   0,   0,   2,   1],
       [  1,   0,   0,   2,   1],
       [  0,   0,   0,   1,   6]])
```

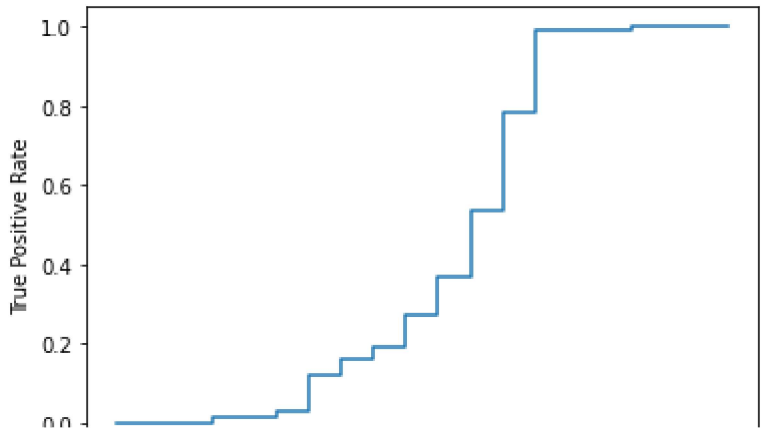
```
print('Accuracy of logistic regression classifier on test set: {}'.format(classifier.score(x_

print('Accuracy of logistic regression classifier on test set: {}'.format(classifier.score(x_
```

```
Accuracy of logistic regression classifier on test set: 0.935064935064935
Accuracy of logistic regression classifier on test set: 0.935064935064935
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.97      1.00      0.99       135
               1       1.00      0.25      0.40         4
               2       0.00      0.00      0.00         4
               3       0.40      0.50      0.44         4
               4       0.67      0.86      0.75         7

        accuracy                           0.94       154
       macro avg       0.61      0.52      0.52       154
    weighted avg       0.92      0.94      0.92       154
```

```
tested_x = classifier.predict(x_test)


from sklearn import metrics
log_regression = LogisticRegression()

log_regression.fit(x_train,y_train)
#define metrics
y_pred_proba = log_regression.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba, pos_label=0)

#create ROC curve
plt.plot(fpr,tpr )
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Decision tree for HCV dataset

Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree.

```
# importing the sklearn library
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.tree import DecisionTreeClassifier
```

Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified.

```
# getting the x(dataset excluding the class attribute) and y(class attribute) dataset from cl
df.drop(df.iloc[:, 0:1],axis = 1)
xdf=df
t = xdf.iloc[:, 1:].values
v = new_df.iloc[:, 0].values

print(v)

t_train, t_test, v_train, v_test= train_test_split(t, v, test_size= 0.25)


st_x= StandardScaler()
t_train= st_x.fit_transform(t_train)
t_test= st_x.fit_transform(t_test)

clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)

# fit the model
clf_gini.fit(t_train, v_train)
```

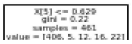
```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
DecisionTreeClassifier(max_depth=3, random_state=0)
```

```
v_pred_gini = clf_gini.predict(t_test)

print('Model accuracy score with criterion gini index: {0:0.4f}'. format(accuracy_score(v_tes

v_pred_train_gini = clf_gini.predict(t_train)

v_pred_train_gini
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(v_train, v_pred_train_gi
```

```
Model accuracy score with criterion gini index: 0.8831
Training-set accuracy score: 0.9523
```

```
plt.figure(figsize=(12,8))

from sklearn import tree

tree.plot_tree(clf_gini.fit(t_train, v_train))
```

```
[Text(0.5, 0.875, 'X[5] <= 0.629\ngini = 0.22\nsamples =
461\nvalue = [406, 5, 12, 16, 22]'),
 Text(0.25, 0.625, 'X[4] <= -0.685\ngini = 0.079\nsamples =
420\nvalue = [403, 2, 7, 5, 3]'),
 Text(0.125, 0.375, 'X[3] <= -0.514\ngini = 0.754\nsamples =
17\nvalue = [6, 1, 4, 3, 3]'),
 Text(0.0625, 0.125, 'gini = 0.594\nsamples = 8\nvalue = [0, 1,
4, 3, 0]'),
 Text(0.1875, 0.125, 'gini = 0.444\nsamples = 9\nvalue = [6, 0,
0, 0, 3]'),
 Text(0.375, 0.375, 'X[3] <= -1.85\ngini = 0.029\nsamples =
403\nvalue = [397, 1, 3, 2, 0]'),
 Text(0.3125, 0.125, 'gini = 0.0\nsamples = 2\nvalue = [0, 0, 0,
2, 0]'),
 Text(0.4375, 0.125, 'gini = 0.02\nsamples = 401\nvalue = [397,
1, 3, 0, 0]'),
 Text(0.75, 0.625, 'X[7] <= -0.81\ngini = 0.688\nsamples =
41\nvalue = [3, 3, 5, 11, 19]'),
 Text(0.625, 0.375, 'X[6] <= 0.272\ngini = 0.374\nsamples =
23\nvalue = [1, 1, 2, 1, 18]'),
 Text(0.5625, 0.125, 'gini = 0.75\nsamples = 8\nvalue = [1, 1, 2,
1, 3]'),
 Text(0.6875, 0.125, 'gini = 0.0\nsamples = 15\nvalue = [0, 0, 0,
0, 15]'),
 Text(0.875, 0.375, 'X[3] <= -0.36\ngini = 0.636\nsamples =
18\nvalue = [2, 2, 3, 10, 1]'),
 Text(0.8125, 0.125, 'gini = 0.439\nsamples = 14\nvalue = [0, 0,
3, 10, 1]'),
 Text(0.9375, 0.125, 'gini = 0.5\nsamples = 4\nvalue = [2, 2, 0,
0, 0]')]
```



Logistic regression for my data

```
from google.colab import files
upload = files.upload()
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell

```
import pandas as pd
import io
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings
from sklearn.preprocessing import LabelEncoder
labelencoder = LabelEncoder()

%matplotlib inline

warnings.filterwarnings('ignore')


df=pd.read_csv('/content/student-mat.csv', delimiter=";")
df.head()
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famre
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	
3	GP	F	15	U	GT3	T	4	2	health	services	...	
4	GP	F	16	U	GT3	T	3	3	other	other	...	

5 rows × 33 columns

```
#Drop the columns with the non numerical values.
num_df=df
num_df.drop(df.iloc[1: , :],axis = 1)
print(num_df)
# num_df =df1.drop(['Grid Ref: Easting', 'Grid Ref: Northing','Reference Number','Accident Da
num_df.columns
```

```
    school sex  age address famsize Pstatus  Medu  Fedu      Mjob      Fjob  \
0       GP   F   18       U     GT3       A     4     4   at_home   teacher
1       GP   F   17       U     GT3       T     1     1   at_home     other
2       GP   F   15       U     LE3       T     1     1   at_home     other
3       GP   F   15       U     GT3       T     4     2    health  services
4       GP   F   16       U     GT3       T     3     3     other     other
..     ...  ..  ...     ...     ...     ...   ...   ...       ...       ...
390     MS   M   20       U     LE3       A     2     2  services  services
```

```
391     MS   M   17       U     LE3       T     3     1  services  services
392     MS   M   21       R     GT3       T     1     1     other     other
393     MS   M   18       R     LE3       T     3     2  services     other
394     MS   M   19       U     LE3       T     1     1     other   at_home

     ... famrel freetime  goout  Dalc  Walc health absences  G1  G2  G3
0    ...      4        3      4     1     1      3        6   5   6   6
1    ...      5        3      3     1     1      3        4   5   5   6
2    ...      4        3      2     2     3      3       10   7   8  10
3    ...      3        2      2     1     1      5        2  15  14  15
4    ...      4        3      2     1     2      5        4   6  10  10
..   ...    ...      ...    ...   ...   ...    ...      ...  ..  ..  ..
390  ...      5        5      4     4     5      4       11   9   9   9
391  ...      2        4      5     3     4      2        3  14  16  16
392  ...      5        5      3     3     3      3        3  10   8   7
393  ...      4        4      1     3     4      5        0  11  12  10
394  ...      3        2      3     3     3      5        5   8   9   9

[395 rows x 33 columns]
Index(['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu',
       'Mjob', 'Fjob', 'reason', 'guardian', 'traveltime', 'studytime',
       'failures', 'schoolsup', 'famsup', 'paid', 'activities', 'nursery',
       'higher', 'internet', 'romantic', 'famrel', 'freetime', 'goout', 'Dalc',
       'Walc', 'health', 'absences', 'G1', 'G2', 'G3'],
      dtype='object')
```

```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()

cols = ['address','famsize','school', 'sex','Mjob','Fjob','schoolsup','famsup','paid','Pstatu
df[cols] = df[cols].apply(LabelEncoder().fit_transform)


#x= num_df.iloc[:, 0:].values y= num_df.iloc[:, 0].values

x= num_df.iloc[:, 1:32].values
y= num_df.iloc[:, 0].values


# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)


#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)


#Fitting Logistic Regression to the training set
from sklearn.linear_model import LogisticRegression
```

```
classifier= LogisticRegression(random_state=0)
classifier.fit(x_train, y_train)
```

```
LogisticRegression(random_state=0)
```

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
cm
```

```
array([[89,  3],
       [ 4,  3]])
```

```
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(classifier.scor
```

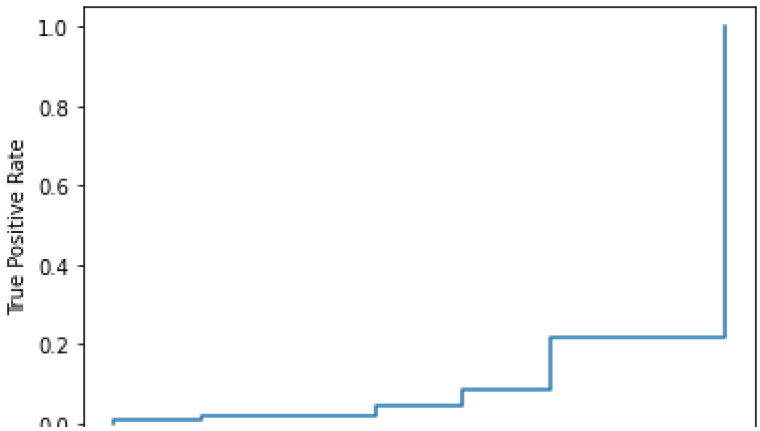
```
Accuracy of logistic regression classifier on test set: 0.93
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
                   precision    recall  f1-score   support

               0       0.96      0.97      0.96        92
               1       0.50      0.43      0.46         7

        accuracy                           0.93        99
       macro avg       0.73      0.70      0.71        99
    weighted avg       0.92      0.93      0.93        99
```

```
tested_x = classifier.predict(x_test)
```

```
from sklearn import metrics
log_regression = LogisticRegression()

log_regression.fit(x_train,y_train)
#define metrics
y_pred_proba = log_regression.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba, pos_label=0)

#create ROC curve
plt.plot(fpr,tpr)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



LAB 09

Exploring mlp classifier

```
from google.colab import files

uploaded = files.upload()
```

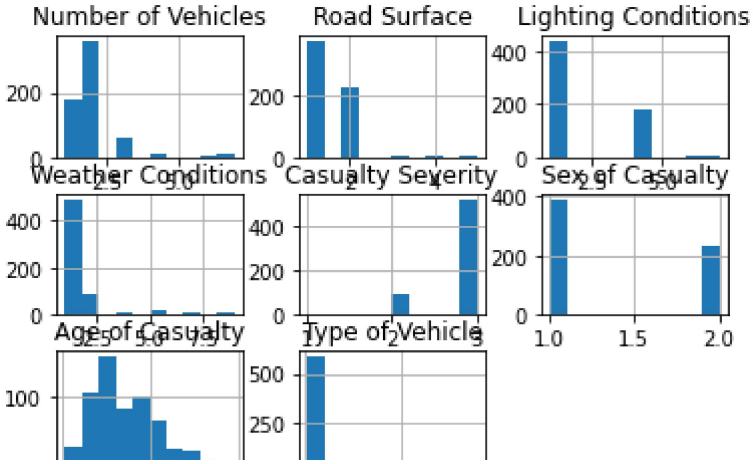
```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving accident-2014-4.csv to accident-2014-4.csv
```

```
import pandas as pd
import io
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans
%matplotlib inline
import warnings
from sklearn.preprocessing import LabelEncoder
warnings.filterwarnings('ignore')

df = pd.read_csv(io.BytesIO(uploaded['accident-2014-4.csv']))
print(df.head().to_markdown())
```

```
|    | Reference Number   |   Grid Ref: Easting |   Grid Ref: Northing |   Number of Veh
|---:|:-------------------|--------------------:|---------------------:|----------------
|  0 | 11S0281            |              407491 |               428929 |
|  1 | 11S1147            |              412912 |               429039 |
|  2 | 11S1147            |              412912 |               429039 |
|  3 | 11U0285            |              407179 |               424776 |
|  4 | 11U0285            |              407179 |               424776 |
```

```
#Drop the columns with the non numerical values.
df1 = df
num_df =df1.drop(['Grid Ref: Easting', 'Grid Ref: Northing','Reference Number','Accident Date
num_df.columns
num_df.hist()

# Bring the class attribute as the last column
# The casulaty Severity here has 3 values.. 1.Fatal, 2.Seriuos . 3.Slight

temp_series = num_df.pop('Casualty Severity')
num_df['Casualty Severity'] = temp_series
num_df.columns
```

```
Index(['Number of Vehicles', 'Road Surface', 'Lighting Conditions',
       'Weather Conditions', 'Sex of Casualty', 'Age of Casualty',
       'Type of Vehicle', 'Casualty Severity'],
      dtype='object')
```




```
# importing the sklearn library
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import roc_auc_score


# getting the x(dataset excluding the class attribute) and y(class attribute) dataset from cl
x = num_df.drop('Casualty Severity',axis=1)
y = num_df['Casualty Severity']

X_train, X_test, y_train, y_test= train_test_split(x, y, test_size= 0.25)


clf = MLPClassifier(hidden_layer_sizes=(100,),random_state=1, max_iter=300).fit(X_train, y_tr
clf.fit(X_train,y_train)
y_pred = clf.predict_proba(X_train)
print('Overall AUC:', roc_auc_score(y_train, clf.predict_proba(X_train),multi_class="ovr"))
```

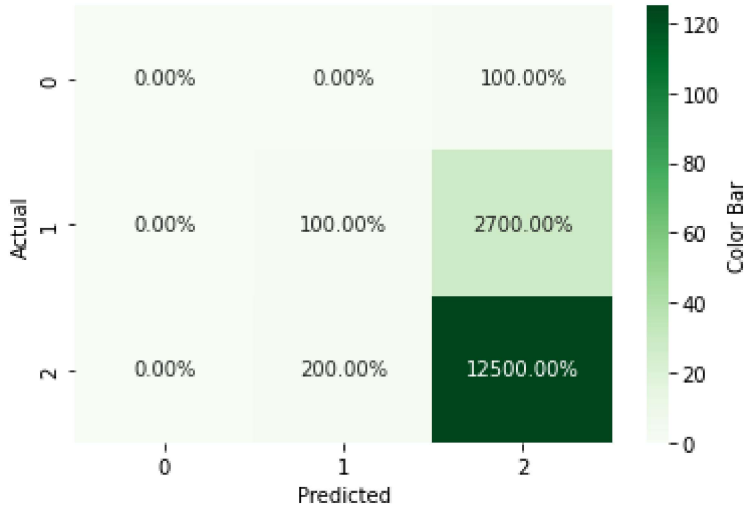
```
Overall AUC: 0.8485003047881352
```

```
y_pred = clf.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
cm
```

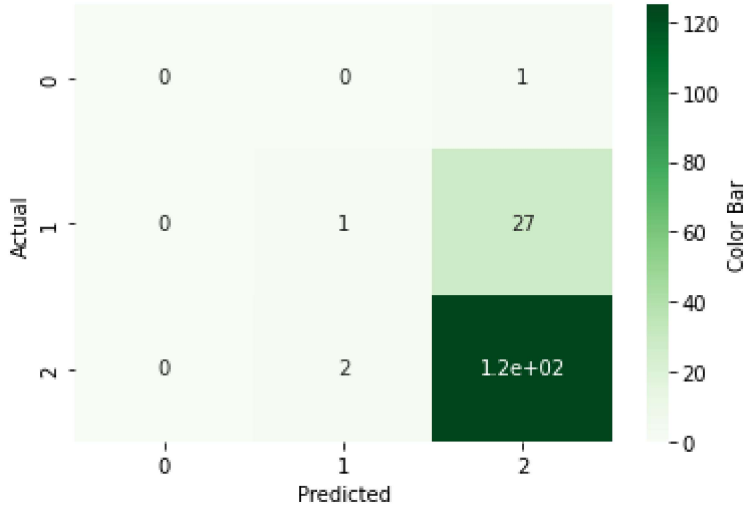
```
array([[  0,   0,   1],
       [  0,   1,  27],
       [  0,   2, 125]])
```

```
import seaborn as seaborn
import matplotlib.pyplot as plt
```

```
sns.heatmap(cm, cmap="Greens", annot=True,fmt='.2%',
            cbar_kws={'orientation':'vertical', 'label':'Color Bar'})
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```




```
sns.heatmap(cm, cmap="Greens", annot=True,
            cbar_kws={'orientation':'vertical', 'label':'Color Bar'})
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```




```
clf.predict_proba(X_test)
print('')
```

```
clf.score(X_test, y_test)
```

```
0.8589743589743589
```

```
score = clf.score(X_train,y_train)
```

```
score
```

```
0.867237687366167
```

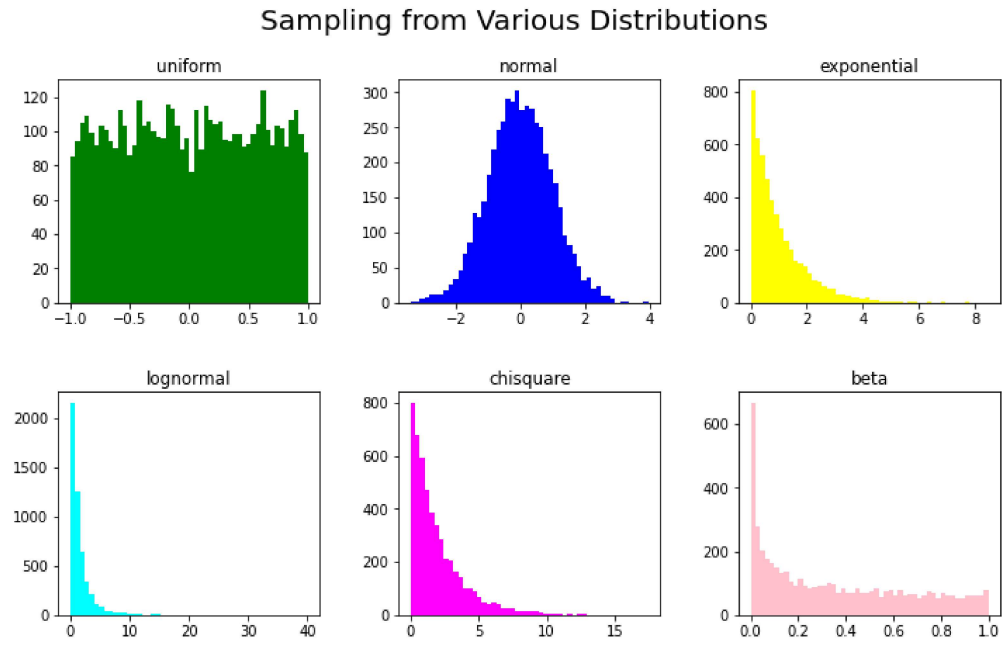
```
activationList = ["relu", "identity", "logistic", "tanh"]
for i in range(0,4):
  clf = MLPClassifier(activation = activationList[i]);
  clf.fit(X_train, y_train);
  tempscore = clf.score(X_train, y_train)
  print("Activation function -",activationList[i],"- Accuracy : ",tempscore)
```

```
Activation function - relu - Accuracy :  0.8586723768736617
Activation function - identity - Accuracy :  0.8522483940042827
Activation function - logistic - Accuracy :  0.8522483940042827
Activation function - tanh - Accuracy :  0.860813704496788
```

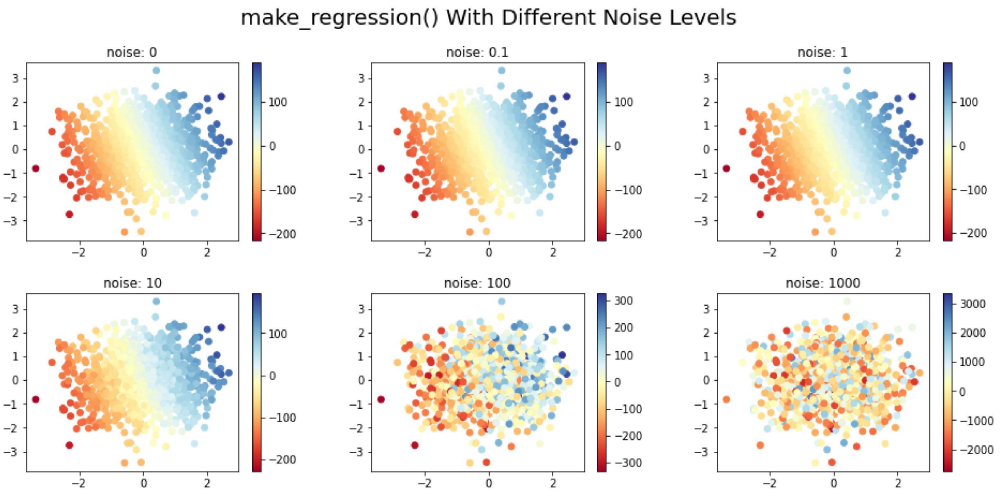
```
import numpy as np

# Needed for plotting
import matplotlib.colors
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Needed for generating classification, regression and clustering datasets
import sklearn.datasets as dt

# Needed for generating data from an existing dataset
from sklearn.neighbors import KernelDensity
from sklearn.model_selection import GridSearchCV



# Define the seed so that results can be reproduced
seed = 11
rand_state = 11

# Define the color maps for plots
color_map = plt.cm.get_cmap('RdYlBu')
color_map_discrete = matplotlib.colors.LinearSegmentedColormap.from_list("", ["red","cyan","m
```



```
rand = np.random.RandomState(seed)

dist_list = ['uniform','normal','exponential','lognormal','chisquare','beta']
param_list = ['-1,1','0,1','1','0,1','2','0.5,0.9']
colors_list = ['green','blue','yellow','cyan','magenta','pink']


fig,ax = plt.subplots(nrows=2, ncols=3,figsize=(12,7))
plt_ind_list = np.arange(6)+231

for dist, plt_ind, param, colors in zip(dist_list, plt_ind_list, param_list, colors_list):
    x = eval('rand.'+dist+'('+param+',5000)')

    plt.subplot(plt_ind)
    plt.hist(x,bins=50,color=colors)
    plt.title(dist)

fig.subplots_adjust(hspace=0.4,wspace=.3)
plt.suptitle('Sampling from Various Distributions',fontsize=20)
plt.show()
```



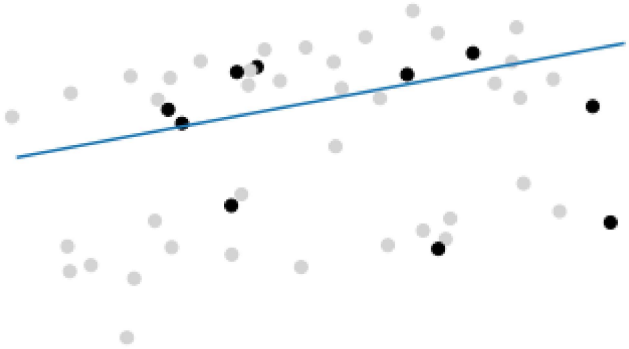

```
map_colors = plt.cm.get_cmap('RdYlBu')
fig,ax = plt.subplots(nrows=2, ncols=3,figsize=(16,7))
plt_ind_list = np.arange(6)+231

for noise,plt_ind in zip([0,0.1,1,10,100,1000],plt_ind_list):
    x,y = dt.make_regression(n_samples=1000,
                             n_features=2,
                             noise=noise,
                             random_state=rand_state)

    plt.subplot(plt_ind)
    my_scatter_plot = plt.scatter(x[:,0],
                                  x[:,1],
                                  c=y,
                                  vmin=min(y),
                                  vmax=max(y),
                                  s=35,
                                  cmap=color_map)

    plt.title('noise: '+str(noise))
    plt.colorbar(my_scatter_plot)

fig.subplots_adjust(hspace=0.3,wspace=.3)
plt.suptitle('make_regression() With Different Noise Levels',fontsize=20)
plt.show()
```



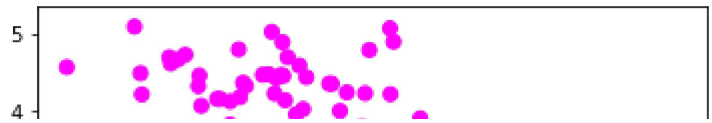

```
#SVM : Support vector machines : Supervised Machine Learning Algorithm used for classificatio


import numpy as np
from sklearn.datasets import make_classification
from sklearn import svm
from sklearn.model_selection import train_test_split

classes = 4
X,t= make_classification(100, 5, n_classes = classes, random_state= 40, n_informative = 2, n_
#%%
X_train, X_test, y_train, y_test= train_test_split(X, t , test_size=0.50)
#%%
model = svm.SVC(kernel = 'linear', random_state = 0, C=1.0)
#%%
model.fit(X_train, y_train)
#%%
y=model.predict(X_test)
y2=model.predict(X_train)
#%%
from sklearn.metrics import accuracy_score
score =accuracy_score(y, y_test)
print(score)
score2 =accuracy_score(y2, y_train)
print(score2)
#%%
import matplotlib.pyplot as plt
color = ['black' if c == 0 else 'lightgrey' for c in y]
plt.scatter(X_train[:,0], X_train[:,1], c=color)
```

```
# Create the hyperplane
w = model.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(-2.5, 2.5)
yy = a * xx - (model.intercept_[0]) / w[1]

# Plot the hyperplane
plt.plot(xx, yy)
plt.axis("off"), plt.show();
```

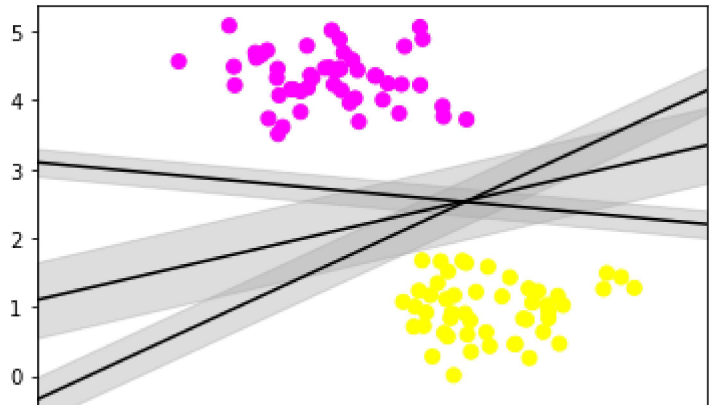
```
0.9
0.96
```



```
import seaborn as sns
# importing scikit learn with make_blobs
from sklearn.datasets import make_blobs

# creating datasets X containing n_samples
# Y containing two classes
X, Y = make_blobs(n_samples=100, centers=2,
                  random_state=0, cluster_std=0.40)
import matplotlib.pyplot as plt
# plotting scatters
plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring');
plt.show()
```

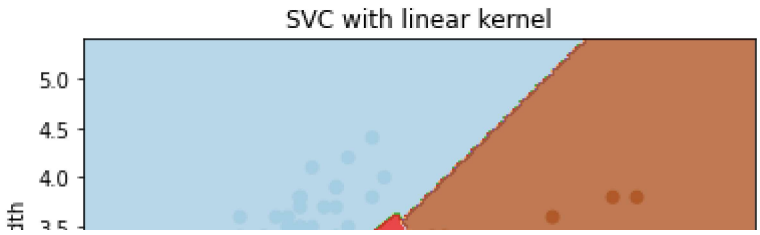



```
# creating linspace between -1 to 3.5
xfit = np.linspace(-1, 3.5)

# plotting scatter
plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring')

# plot a line between the different sets of data
for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
    yfit = m * xfit + b
    plt.plot(xfit, yfit, '-k')
    plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
    color='#AAAAAA', alpha=0.4)

plt.xlim(-1, 3.5);
plt.show()
```

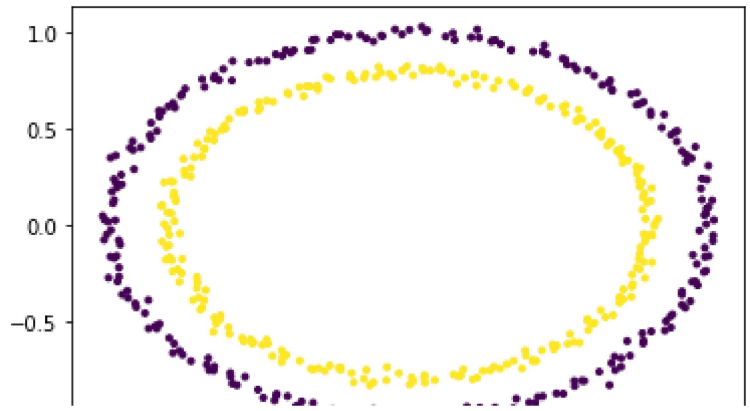



```
#Import standards library
import numpy as np
import matplotlib.pyplot as plt
#Import datest and model selection
from sklearn import svm
from sklearn.datasets import make_blobs
#Creation 40 sepearable points
X, y = make_blobs(n_samples=100, centers=2, random_state=20)


#linear dataset- apllications : text Classification, as each alphabet is a new feature.

# Import the Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
```

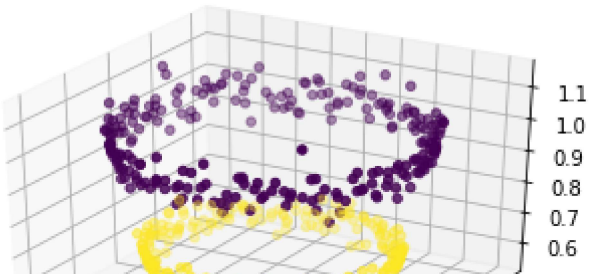
```
# Import some Data from the iris Data Set
iris = datasets.load_iris()

# Take only the first two features of Data.
# To avoid the slicing, Two-Dim Dataset can be used

X = iris.data[:, :2]
y = iris.target

# C is the SVM regularization parameter
C = 1.0

# Create an Instance of SVM and Fit out the data.
# Data is not scaled so as to be able to plot the support vectors
svc = svm.SVC(kernel ='linear', C = 1).fit(X, y)

# create a mesh to plot
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
        np.arange(y_min, y_max, h))

# Plot the data for Proper Visual Representation
plt.subplot(1, 1, 1)

# Predict the result by giving Data to the model
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap = plt.cm.Paired, alpha = 0.8)

plt.scatter(X[:, 0], X[:, 1], c = y, cmap = plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')

# Output the Plot
plt.show()
```



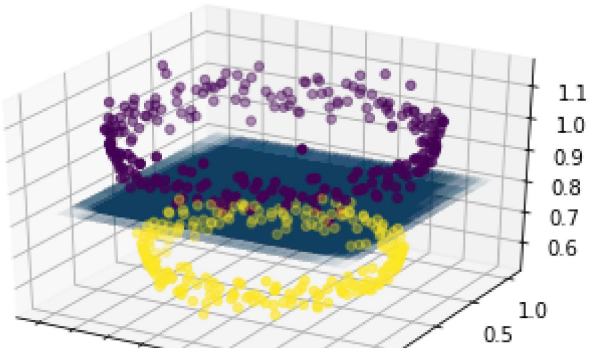

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
from mpl_toolkits.mplot3d import Axes3D

# generating data
X, Y = make_circles(n_samples = 500, noise = 0.02)

# visualizing data
plt.scatter(X[:, 0], X[:, 1], c = Y, marker = '.')
plt.show()
```




```
# adding a new dimension to X
X1 = X[:, 0].reshape((-1, 1))
X2 = X[:, 1].reshape((-1, 1))
X3 = (X1**2 + X2**2)
X = np.hstack((X, X3))

# visualizing data in higher dimension
fig = plt.figure()
axes = fig.add_subplot(111, projection = '3d')
axes.scatter(X1, X2, X1**2 + X2**2, c = Y, depthshade = True)
plt.show()
```



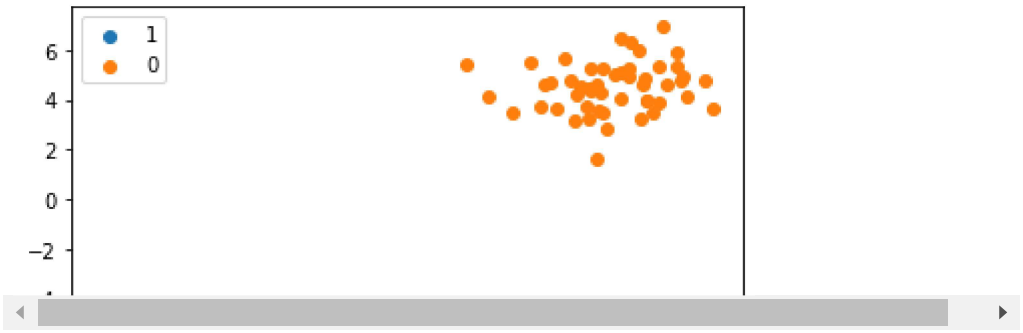

```
# create support vector classifier using a linear kernel
from sklearn import svm

svc = svm.SVC(kernel = 'linear')
svc.fit(X, Y)
w = svc.coef_
b = svc.intercept_

# plotting the separating hyperplane
x1 = X[:, 0].reshape((-1, 1))
x2 = X[:, 1].reshape((-1, 1))
x1, x2 = np.meshgrid(x1, x2)
x3 = -(w[0][0]*x1 + w[0][1]*x2 + b) / w[0][2]

fig = plt.figure()
axes2 = fig.add_subplot(111, projection = '3d')
axes2.scatter(X1, X2, X1**2 + X2**2, c = Y, depthshade = True)
axes1 = fig.gca(projection = '3d')
axes1.plot_surface(x1, x2, x3, alpha = 0.01)
plt.show()
```




```
#advantages Training a SVM with a Linear Kernel is Faster than with any other Kernel.


# example of binary classification task
from numpy import where
from collections import Counter
from sklearn.datasets import make_blobs
```

```
from matplotlib import pyplot
# define dataset
#n_features=3 by default the feature =2
X, y = make_blobs(n_samples=100, centers=2,n_features=5, random_state=1)
# summarize dataset shape
print(X.shape, y.shape)
# summarize observations by class label
counter = Counter(y)
#print(counter)
# summarize first few examples

for i in range(10):
    print(X[i], y[i])
# plot the dataset and color the by class label
for label, _ in counter.items():
    row_ix = where(y == label)[0]
    pyplot.scatter(X[row_ix, 0], X[row_ix, 1], label=str(label))
pyplot.legend()
pyplot.show()
```

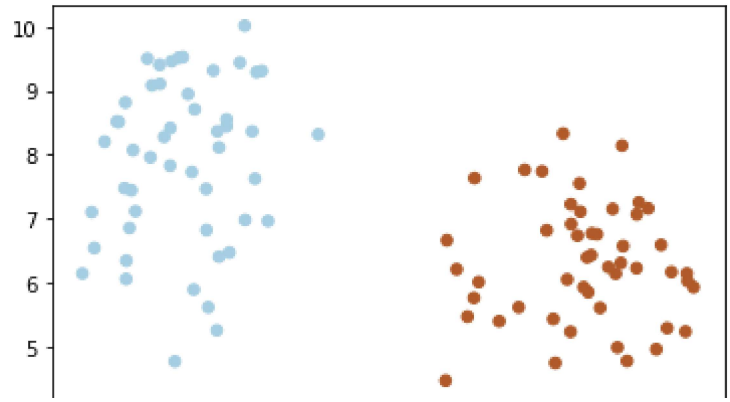
```
(100, 5) (100,)
[-8.27199928 -6.47379396 -1.22231408 -2.48358841  0.29714976] 1
[-6.38526816 -6.75016865 -2.61117528 -3.08653646  1.57086292] 1
[-1.13898357  3.26214848 -9.19585147 -3.90678125 -7.25145196] 0
[-2.16402577  4.56652694 -9.12154358 -3.6377136  -9.0870834 ] 0
[-10.29569483  -7.10496464  -2.63716951  -0.96047619   0.49459841
[-2.33022219  4.78405366 -9.87589123 -2.82386464 -5.8659643 ] 0
[-7.52303243 -6.68964267 -2.63683942 -3.6438068  -0.0522933 ] 1
[-6.60986899 -5.51599011 -2.20387664 -2.94193203 -0.09145254] 1
[ -0.91900345   3.45278927 -10.26393101  -3.920734    -8.4379995
[-1.45772973  5.06751016 -8.20555429 -4.07381312 -8.29800292] 0
```




```
#Import standards library
import numpy as np
import matplotlib.pyplot as plt
#Import datest and model selection
from sklearn import svm
from sklearn.datasets import make_blobs
```

```
#Creation 40 sepearable points
X, y = make_blobs(n_samples=100, centers=2, random_state=20)

#Fit the model
clf = svm.SVC(kernel='linear', C=1000)
clf.fit(X,y)
plt.scatter(X[:,0],X[:,1], c=y, s=30, cmap=plt.cm.Paired)
#plt.show()
#Predict unknown data
newData = [[3,4],[5,6]]
print(clf.predict(newData))
```

```
[0 1]
```




```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(X, y, test_size= 0.50, random_state=0)
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)


from sklearn.svm import SVC # "Support vector classifier"
classifier = SVC(kernel='linear', random_state=0)
classifier.fit(x_train, y_train)
```

```
SVC(kernel='linear', random_state=0)
```

```
#Predicting the test set result
y_pred= classifier.predict(x_test)


#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
```

```
cm= confusion_matrix(y_test, y_pred)
```

```
cm
```

```
array([[21,  0],
       [ 0, 29]])
```

```
from sklearn.metrics import accuracy_score
acc= accuracy_score(y_test, y_pred)
```

```
acc*100
```

```
100.0
```

```
from sklearn.metrics import classification_report
report=classification_report(y_test, y_pred)
```

```
print(report)
```

```
                  precision    recall  f1-score   support

               0       1.00      1.00      1.00        21
               1       1.00      1.00      1.00        29

        accuracy                           1.00        50
       macro avg       1.00      1.00      1.00        50
    weighted avg       1.00      1.00      1.00        50
```

```
import pandas as pd
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
import seaborn as sb
df = pd.read_csv ('rockmusic_2147239.csv')
print (df)
data_top = df.head()
data_top

data_down = df.tail()
data_down
```

```
      index                                          name  \
0         0                       Smells Like Teen Spirit
1         1                  Stairway to Heaven - Remaster
2         2            Bohemian Rhapsody - Remastered 2011
3         3                        Imagine - Remastered 2010
4         4  (I Can't Get No) Satisfaction - Mono Version
...     ...                                           ...
5479   5479                              I'm In Your Mind
5480   5480                                   Cellophane
5481   5481                                    Hot Water
5482   5482            Vitamin C - 2004 Remastered Version
5483   5483                                             ~

                              artist  release_date    length  popularity  \
0                            Nirvana          1991  5.032000          74
1                        Led Zeppelin          1971  8.047167          78
2                              Queen          1975  5.905333          74
3                         John Lennon          1971  3.131100          77
4                   The Rolling Stones          1965  3.713550          77
...                                ...           ...       ...         ...
5479  King Gizzard & The Lizard Wizard          2014  3.559833          47
5480  King Gizzard & The Lizard Wizard          2014  3.179750          44
5481  King Gizzard & The Lizard Wizard          2014  3.396450          40
5482                               CAN          1972  3.567767          52
```

```
df.head()
```

	index	name	artist	release_date	length	popularity	danceability	acousti
0	0	Smells Like Teen Spirit	Nirvana	1991	5.032000	74	0.502	0.0
1	1	Stairway to Heaven - Remaster	Led Zeppelin	1971	8.047167	78	0.338	0.5
2	2	Bohemian Rhapsody - Remastered 2011	Queen	1975	5.905333	74	0.392	0.2
3	3	Imagine - Remastered 2010	John Lennon	1971	3.131100	77	0.547	0.9
4	4	(I Can't Get No) Satisfaction - Mono Version	The Rolling Stones	1965	3.713550	77	0.723	0.0

```
df.describe()
```

	index	release_date	length	popularity	danceability	acousticness	c
count	5484.000000	5484.000000	5484.000000	5484.000000	5484.000000	5484.000000	
mean	2741.500000	1991.196389	4.148302	49.413202	0.511047	0.173019	
std	1583.238769	15.331628	1.496269	17.317263	0.147916	0.242596	
min	0.000000	1956.000000	0.162533	0.000000	0.000000	0.000001	
25%	1370.750000	1978.000000	3.302100	40.000000	0.413000	0.003658	
50%	2741.500000	1993.000000	3.945442	52.000000	0.515000	0.048400	
75%	4112.250000	2004.000000	4.680271	62.000000	0.611000	0.260250	
max	5483.000000	2020.000000	24.091767	84.000000	0.987000	0.995000	

```
df.isnull().sum().sum()
#checking null values throyghout the dataset
```

```
0
```

```
df.dropna(inplace=True)
#drop all the null values and make inplace as true permanently
```

```
df.isnull().sum()
#checking null values in each column
```

```
index              0
name               0
artist             0
release_date       0
length             0
popularity         0
danceability       0
acousticness       0
danceability.1     0
energy             0
instrumentalness   0
key                0
liveness           0
loudness           0
```

```
speechiness         0
tempo               0
time_signature      0
valence             0
dtype: int64
```

```
a=df.corr()
a
#The linear/pairwise relationship between two variables.
```

	index	release_date	length	popularity	danceability	acousticne
index	1.000000	0.165672	-0.053774	-0.386874	-0.056993	-0.0635
release_date	0.165672	1.000000	0.003281	-0.282614	-0.082780	-0.2776
length	-0.053774	0.003281	1.000000	0.000198	-0.134880	-0.0817
popularity	-0.386874	-0.282614	0.000198	1.000000	0.115690	0.0476
danceability	-0.056993	-0.082780	-0.134880	0.115690	1.000000	0.0995
acousticness	-0.063588	-0.277685	-0.081761	0.047653	0.099566	1.0000
danceability.1	-0.056993	-0.082780	-0.134880	0.115690	1.000000	0.0995
energy	0.068924	0.285391	-0.063400	-0.079499	-0.134408	-0.5941
instrumentalness	0.049552	0.077449	0.125650	-0.115815	-0.098507	-0.0162
key	0.022695	0.030631	0.014821	0.032483	0.021297	-0.0396
liveness	0.014186	0.091113	0.048147	-0.135936	-0.165255	-0.0323
loudness	0.037623	0.382334	-0.101349	-0.008251	-0.124727	-0.4527
speechiness	0.034534	0.064358	-0.050859	-0.055595	-0.091101	-0.0728
tempo	0.022279	0.027662	-0.047545	-0.030013	-0.261845	-0.1380
time_signature	-0.013936	0.026903	0.010123	0.015511	0.110368	-0.1072
valence	-0.039388	-0.215245	-0.246810	0.030984	0.501211	-0.0094

Double-click (or enter) to edit

```
a.style.background_gradient(cmap='binary_r')
# Displaying dataframe as a heatmap
# with diverging colourmap as coolwarm
```

	index	release_date	length	popularity	danceability	acousticne
index	1.000000	0.165672	-0.053774	-0.386874	-0.056993	-0.0635
release_date	0.165672	1.000000	0.003281	-0.282614	-0.082780	-0.2776
length	-0.053774	0.003281	1.000000	0.000198	-0.134880	-0.0817
popularity	-0.386874	-0.282614	0.000198	1.000000	0.115690	0.0476
danceability	-0.056993	-0.082780	-0.134880	0.115690	1.000000	0.0995
acousticness	-0.063588	-0.277685	-0.081761	0.047653	0.099566	1.0000
danceability.1	-0.056993	-0.082780	-0.134880	0.115690	1.000000	0.0995
energy	0.068924	0.285391	-0.063400	-0.079499	-0.134408	-0.5941
instrumentalness	0.049552	0.077449	0.125650	-0.115815	-0.098507	-0.0162
key	0.022695	0.030631	0.014821	0.032483	0.021297	-0.0396
liveness	0.014186	0.091113	0.048147	-0.135936	-0.165255	-0.0323
loudness	0.037623	0.382334	-0.101349	-0.008251	-0.124727	-0.4527
speechiness	0.034534	0.064358	-0.050859	-0.055595	-0.091101	-0.0728
tempo	0.022279	0.027662	-0.047545	-0.030013	-0.261845	-0.1380
time_signature	-0.013936	0.026903	0.010123	0.015511	0.110368	-0.1072
valence	-0.039388	-0.215245	-0.246810	0.030984	0.501211	-0.0094

Double-click (or enter) to edit

```
df.drop(['valence'],axis=1)
```

	index	name	artist	release_date	length	popularity	danceability	acou
0	0	Smells Like Teen Spirit	Nirvana	1991	5.032000	74	0.502	
1	1	Stairway to Heaven - Remaster	Led Zeppelin	1971	8.047167	78	0.338	
2	2	Bohemian Rhapsody - Remastered 2011	Queen	1975	5.905333	74	0.392	
3	3	Imagine - Remastered 2010	John Lennon	1971	3.131100	77	0.547	
4	4	(I Can't Get No) Satisfaction - Mono Version	The Rolling Stones	1965	3.713550	77	0.723	
...	...	...	...	...	...	...	...	...
5479	5479	I'm In Your Mind	King Gizzard & The Lizard Wizard	2014	3.559833	47	0.296	
5480	5480	Cellophane	King Gizzard & The Lizard Wizard	2014	3.179750	44	0.432	
			King Gizzard					

Double-click (or enter) to edit

Wizard

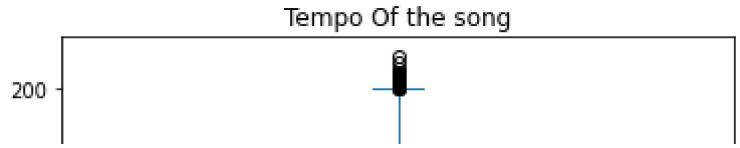
```
df.head()
```

df

	index	name	artist	release_date	length	popularity	danceability	acousti
0	0	Smells Like Teen Spirit	Nirvana	1991	5.032000	74	0.502	0.0
1	1	Stairway to Heaven - Remaster	Led Zeppelin	1971	8.047167	78	0.338	0.5
2	2	Bohemian Rhapsody - Remastered 2011	Queen	1975	5.905333	74	0.392	0.2
3	3	Imagine - Remastered	John	1971	3.131100	77	0.547	0.9

	index	name	artist	release_date	length	popularity	danceability	acou
0	0	Smells Like Teen Spirit	Nirvana	1991	5.032000	74	0.502	
1	1	Stairway to Heaven - Remaster	Led Zeppelin	1971	8.047167	78	0.338	
2	2	Bohemian Rhapsody - Remastered 2011	Queen	1975	5.905333	74	0.392	
3	3	Imagine - Remastered 2010	John Lennon	1971	3.131100	77	0.547	
		(I Can't Get						
		Mono	Stones					

```
df['tempo'].plot(kind='box', title='Tempo Of the song')
plt.show()
df['loudness'].plot(kind='box', title='Loudness Of the song')
plt.show()
#box #whiskers line-median val #outliers-dots #505 of data #interquartile range #mean dotted
```




```
newname = df.rename(columns = {'name': 'song_name', 'artist':'artist_name'} , inplace = False

print(newname)
#renaming a function
```

```
      index                                    song_name  \
0         0                        Smells Like Teen Spirit
1         1                    Stairway to Heaven - Remaster
2         2              Bohemian Rhapsody - Remastered 2011
3         3                           Imagine - Remastered 2010
4         4  (I Can't Get No) Satisfaction - Mono Version
...     ...                                          ...
5479   5479                               I'm In Your Mind
5480   5480                                    Cellophane
5481   5481                                     Hot Water
5482   5482             Vitamin C - 2004 Remastered Version
5483   5483                                             ~

                          artist_name  release_date    length  popularity  \
0                             Nirvana          1991  5.032000          74
1                         Led Zeppelin          1971  8.047167          78
2                                Queen          1975  5.905333          74
3                           John Lennon          1971  3.131100          77
4                    The Rolling Stones          1965  3.713550          77
...                                 ...           ...       ...         ...
5479  King Gizzard & The Lizard Wizard          2014  3.559833          47
5480  King Gizzard & The Lizard Wizard          2014  3.179750          44
5481  King Gizzard & The Lizard Wizard          2014  3.396450          40
5482                               CAN          1972  3.567767          52
5483                     Touché Amoré          2011  1.485100           0

      danceability  acousticness  danceability.1  energy  instrumentalness  \
0            0.502      0.000025           0.502   0.912          0.000173
1            0.338      0.580000           0.338   0.340          0.003200
2            0.392      0.288000           0.392   0.402          0.000000
3            0.547      0.907000           0.547   0.257          0.183000
4            0.723      0.038300           0.723   0.863          0.031700
...            ...           ...             ...     ...               ...
5479         0.296      0.005910           0.296   0.776          0.801000
5480         0.432      0.002130           0.432   0.887          0.916000
5481         0.627      0.860000           0.627   0.609          0.890000
5482         0.643      0.006690           0.643   0.644          0.673000
5483         0.222      0.000258           0.222   0.959          0.000275

      key  liveness  loudness  speechiness    tempo  time_signature  valence
0       1    0.1060    -4.556       0.0564  116.761               4    0.720
1       9    0.1160   -12.049       0.0339   82.433               4    0.197
2       0    0.2430    -9.961       0.0536  143.883               4    0.228
3       0    0.0935   -12.358       0.0252   75.752               4    0.169
4       2    0.1280    -7.890       0.0338  136.302               4    0.931
```

```
...   ...       ...       ...          ...      ...              ...      ...
5479    6    0.5970    -5.630       0.0597   93.481                4    0.406
5480    7    0.1200    -6.175       0.1230   92.965                4    0.357
5481    9    0.1160    -9.387       0.0332   86.861                4    0.734
5482    4    0.1620   -12.615       0.0462  117.225                4    0.853
5483    6    0.1540    -5.134       0.2080  161.693                4    0.128

[5484 rows x 18 columns]
```

```
df.groupby(['name', 'artist'])
```

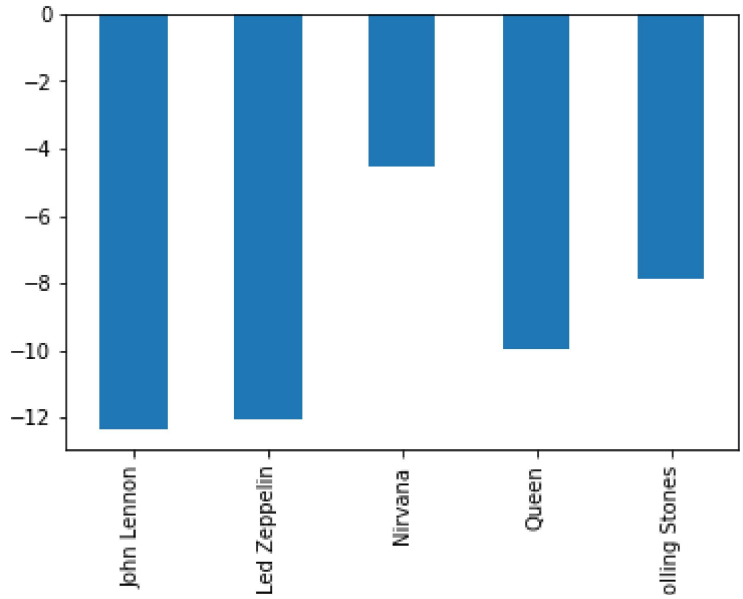
```
print(df.groupby(['name', 'artist']).groups)
```

```
{('"45"', 'The Gaslight Anthem'): [5329], ('"Get A Shot Of The Refrigerator"', 'Stereola
```

```
ch = df.head()
ch
```

	index	name	artist	release_date	length	popularity	danceability	acousti
0	0	Smells Like Teen Spirit	Nirvana	1991	5.032000	74	0.502	0.0
1	1	Stairway to Heaven - Remaster	Led Zeppelin	1971	8.047167	78	0.338	0.5
2	2	Bohemian Rhapsody - Remastered 2011	Queen	1975	5.905333	74	0.392	0.2
3	3	Imagine - Remastered 2010	John Lennon	1971	3.131100	77	0.547	0.9
4	4	(I Can't Get No) Satisfaction - Mono Version	The Rolling Stones	1965	3.713550	77	0.723	0.0

```
bargr=ch['loudness'].groupby(df['artist']).mean().plot(kind='bar')
```

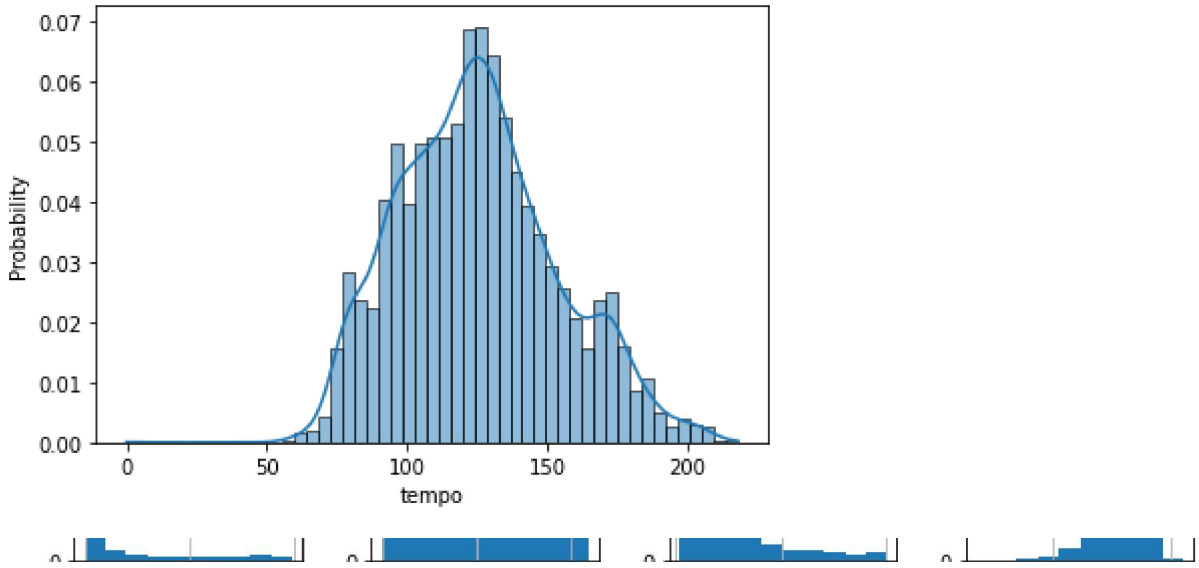



```
df.hist(figsize=(10,10),bins=10)
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f79b6d00090>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f79b6d1ab50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f79b6d9ce10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f79b72f16d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f79bc83ded0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f79bc7edd50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f79b6ddcd90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f79b6770650>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f79b6776890>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f79b6cfba10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f79b6729490>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f79b66d6a90>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f79b6648f50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f79b667f6d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f79b65f2cd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f79b65b3310>]],
      dtype=object)
```

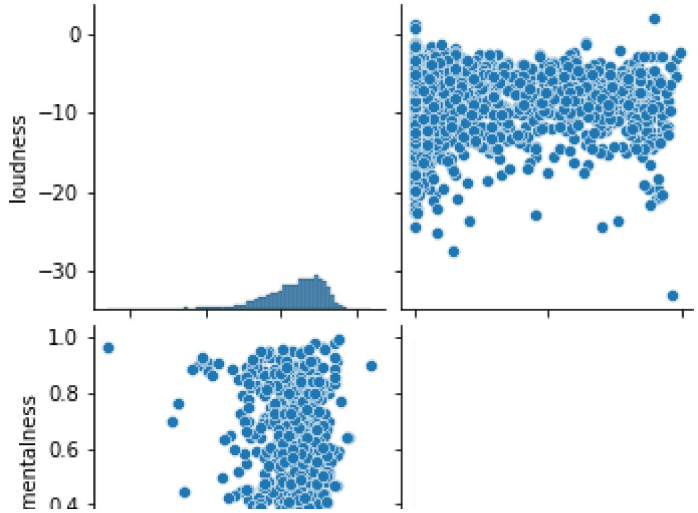
```
num_var = df['tempo']
num_var = pd.Series(num_var, name = "tempo")

# Plot histogram
sns.histplot(data = num_var, kde = True,stat = "probability")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f79b6235090>
```



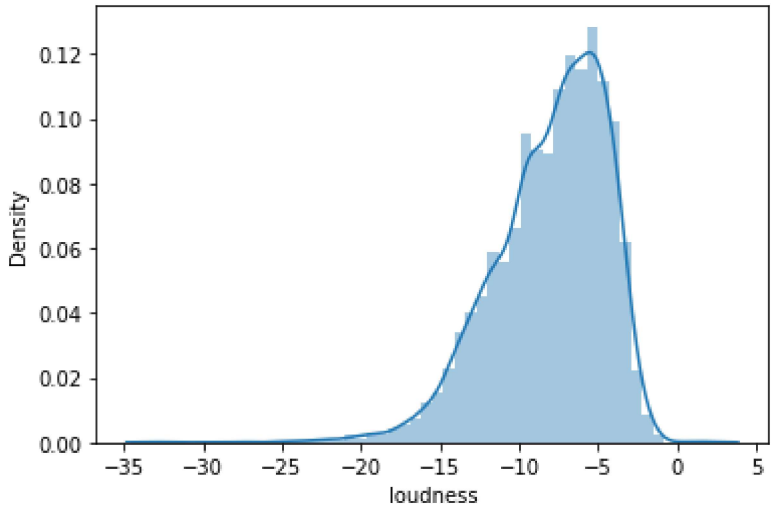

```
import seaborn as sb #seaborn helps to analyse multiple features
col=['loudness','instrumentalness']
sb.pairplot(df[col])
#more than 2 independent features

#scatterplot
```

```
<seaborn.axisgrid.PairGrid at 0x7f79b60ed350>
```




```
a = df['loudness']
sns.distplot(a)
plt.show()
#check distribution of column
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
```




```
colors = ['pink', 'silver', 'steelblue']
```

```
df.groupby(['name']).sum().plot(
    kind='pie', y='tempo', autopct='%1.0f%%',
  colors=colors )
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-80-c47da6eaaf56> in <module>()
      4 df.groupby(['name']).sum().plot(
      5     kind='pie', y='tempo', autopct='%1.0f%%',
----> 6   colors=colors )
```

9 frames

```
<__array_function__ internals> in linspace(*args, **kwargs)

/usr/local/lib/python3.7/dist-packages/numpy/core/function_base.py in linspace(start,
stop, num, endpoint, retstep, dtype, axis)
    133
    134     delta = stop - start
--> 135     y = _nx.arange(0, num, dtype=dt).reshape((-1,) + (1,) * ndim(delta))
    136     # In-place multiplication y *= delta/div is faster, but prevents the
multiplicant
    137     # from overriding what class is produced, and thus prevents, e.g. use of
Quantities,

KeyboardInterrupt:
```

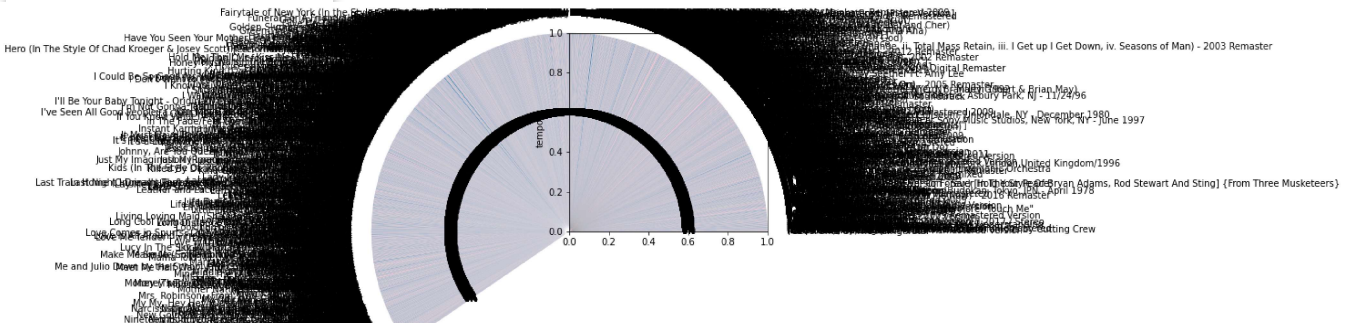
SEARCH STACK OVERFLOW

```
Error in callback <function install_repl_displayhook.<locals>.post_execute at 0x7f79d545
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/matplotlib/pyplot.py in post_execute()
    107             def post_execute():
    108                 if matplotlib.is_interactive():
--> 109                     draw_all()
    110
    111             # IPython >= 2
```

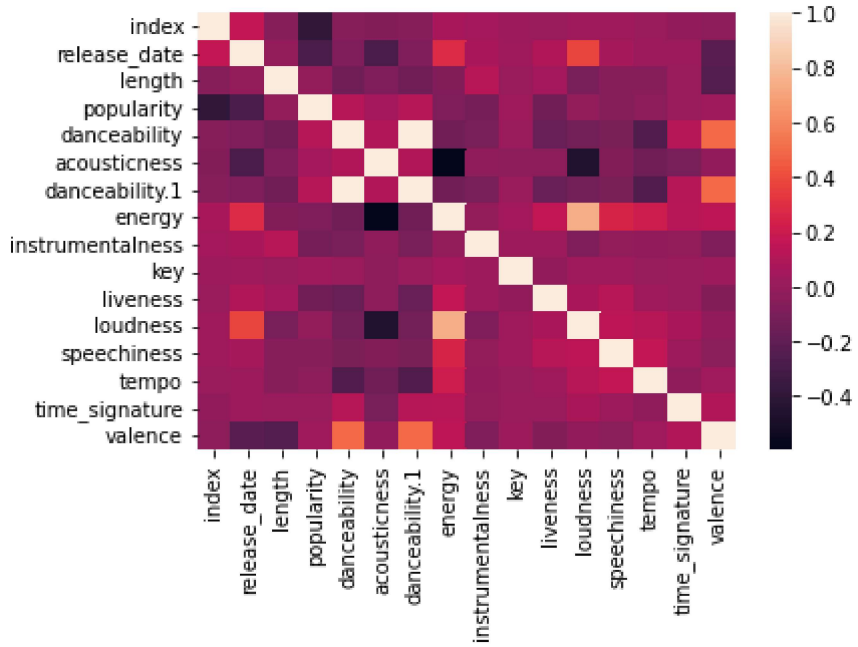
12 frames

```
/usr/local/lib/python3.7/dist-packages/matplotlib/backends/backend_agg.py in
get_text_width_height_descent(self, s, prop, ismath)
    212         flags = get_hinting_flag()
    213         font = self._get_agg_font(prop)
--> 214         font.set_text(s, 0.0, flags=flags)
    215         w, h = font.get_width_height()  # width and height of unrotated string
    216         d = font.get_descent()

KeyboardInterrupt:
```

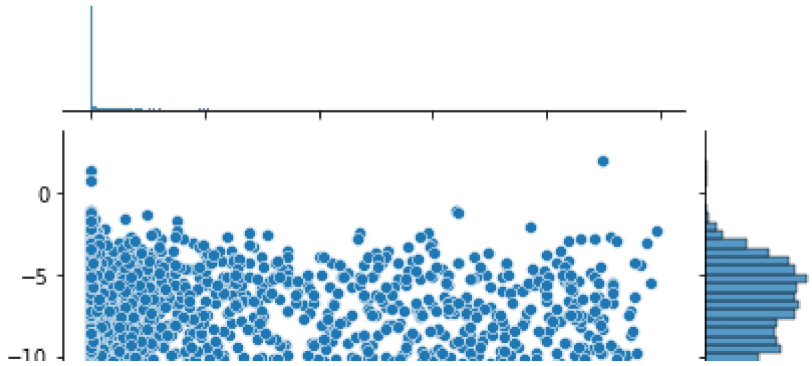
SEARCH STACK OVERFLOW



```
import seaborn as sb
dataplot=sb.heatmap(df.corr())
plt.show()
```



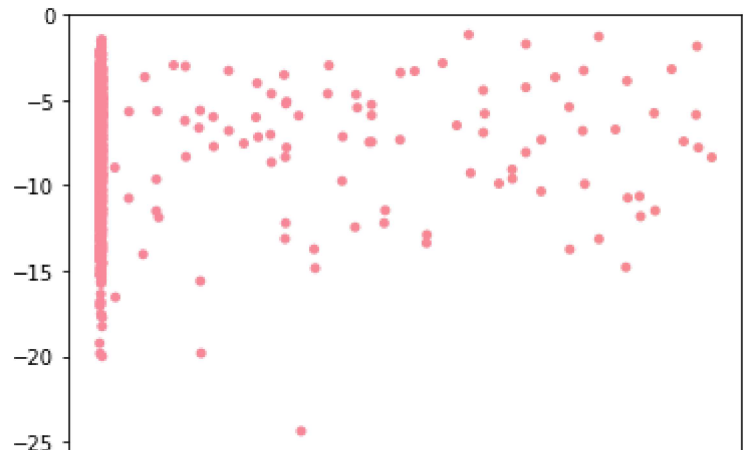

```
# importing required packages
import seaborn as sns
import matplotlib.pyplot as plt
sns.jointplot(x = "instrumentalness", y = "loudness",kind = "scatter", data = df)
# show the plot
plt.show()

#relationship btw 2 numeric variables
#reg #hex
```



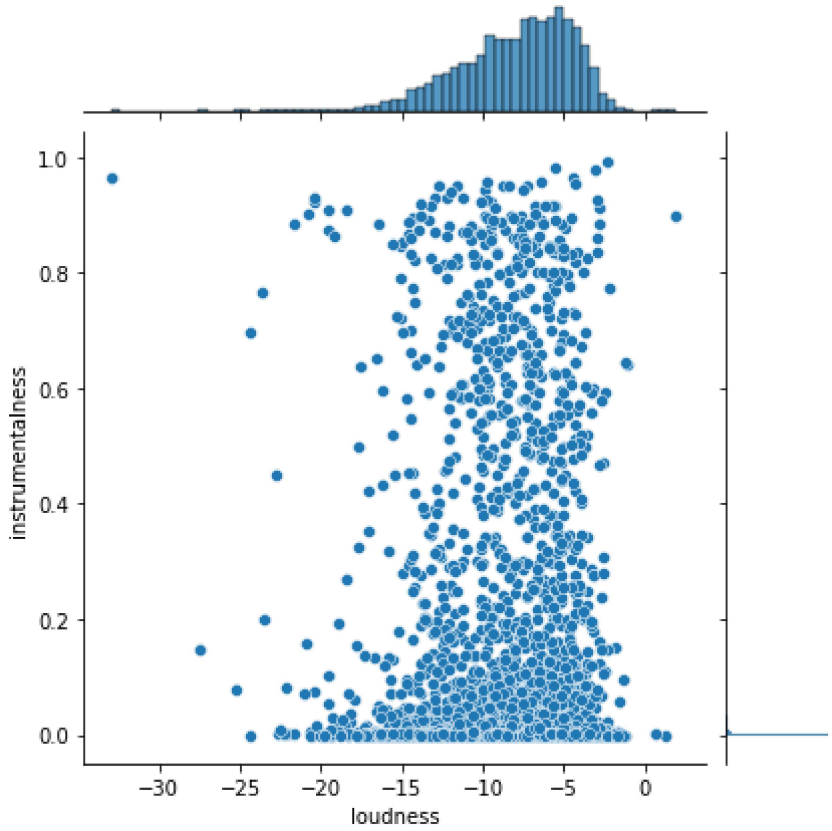

```
sb.stripplot(x="instrumentalness", y="loudness", data=df)

plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/pandas/core/series.py in __setitem__(self, key,
value)
   1061         try:
-> 1062             self._set_with_engine(key, value)
   1063         except (KeyError, ValueError):

                                 17 frames

TypeError: 'slice(None, None, None)' is an invalid key

During handling of the above exception, another exception occurred:

KeyboardInterrupt                         Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/pandas/core/dtypes/generic.py in _check(cls,
inst)
     41     # https://github.com/python/mypy/issues/1006
     42     # error: 'classmethod' used with a non-method
---> 43     @classmethod  # type: ignore[misc]
     44     def _check(cls, inst) -> bool:
     45         return getattr(inst, attr, "_typ") in comp

KeyboardInterrupt:
```

SEARCH STACK OVERFLOW



```
# importing required packages
import seaborn as sns
import matplotlib.pyplot as plt
sns.jointplot(x = "loudness", y = "instrumentalness",kind = "scatter", data = df)
# show the plot
plt.show()
```




```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report


#Extracting Independent and dependent Variable
x= df.iloc[:, [2,3]].values
y= df.iloc[:, 4].values

# Splitting the dataset into training and test set.
X_train, X_test, Y_train, Y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

```
x
```

```
array([['Nirvana', 1991],
       ['Led Zeppelin', 1971],
       ['Queen', 1975],
       ...,
       ['King Gizzard & The Lizard Wizard', 2014],
       ['CAN', 1972],
       ['Touché Amoré', 2011]], dtype=object)
```

```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
#converting string into int using label encounter

cols = ['name','artist']



df[cols] = df[cols].apply(LabelEncoder().fit_transform)
#label encoder



df.info()
#info of data after conversion
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5484 entries, 0 to 5483
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   index             5484 non-null   int64  
 1   name              5484 non-null   int64  
 2   artist            5484 non-null   int64  
 3   release_date      5484 non-null   int64  
 4   length            5484 non-null   float64
 5   popularity        5484 non-null   int64  
 6   danceability      5484 non-null   float64
 7   acousticness      5484 non-null   float64
 8   danceability.1    5484 non-null   float64
 9   energy            5484 non-null   float64
 10  instrumentalness  5484 non-null   float64
 11  key               5484 non-null   int64  
 12  liveness          5484 non-null   float64
 13  loudness          5484 non-null   float64
 14  speechiness       5484 non-null   float64
 15  tempo             5484 non-null   float64
 16  time_signature    5484 non-null   int64  
 17  valence           5484 non-null   float64
dtypes: float64(11), int64(7)
memory usage: 943.1 KB
```

```
df
```

	index	name	artist	release_date	length	popularity	danceability	acousticnes
0	0	3931	942	1991	5.032000	74	0.502	0.00002
1	1	4069	772	1971	8.047167	78	0.338	0.58000
2	2	610	1045	1975	5.905333	74	0.392	0.28800
3	3	2201	704	1971	3.131100	77	0.547	0.90700
4	4	8	1516	1965	3.713550	77	0.723	0.03830
...	...	...	...	...	...	...	...	
5479	5479	2146	751	2014	3.559833	47	0.296	0.00591
5480	5480	789	751	2014	3.179750	44	0.432	0.00213
5481	5481	1932	751	2014	3.396450	40	0.627	0.86000
5482	5482	4884	239	1972	3.567767	52	0.643	0.00669
5483	5483	5315	1627	2011	1.485100	0	0.222	0.00025

5484 rows × 18 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5484 entries, 0 to 5483
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   index             5484 non-null   int64  
 1   name              5484 non-null   int64  
 2   artist            5484 non-null   int64  
 3   release_date      5484 non-null   int64  
 4   length            5484 non-null   float64
 5   popularity        5484 non-null   int64  
 6   danceability      5484 non-null   float64
 7   acousticness      5484 non-null   float64
 8   danceability.1    5484 non-null   float64
 9   energy            5484 non-null   float64
 10  instrumentalness  5484 non-null   float64
 11  key               5484 non-null   int64  
 12  liveness          5484 non-null   float64
 13  loudness          5484 non-null   float64
 14  speechiness       5484 non-null   float64
 15  tempo             5484 non-null   float64
 16  time_signature    5484 non-null   int64  
 17  valence           5484 non-null   float64
dtypes: float64(11), int64(7)
memory usage: 943.1 KB
```

```
#Extracting Independent and dependent Variable
x= df.iloc[:, [2,5]]
y= df.iloc[:, 4]

# Splitting the dataset into training and test set.
X_train, X_test, Y_train, Y_test= train_test_split(x, y, test_size= 0.25, random_state=0)
```

```
X_test
```

	artist	popularity
1540	1373	60
3488	35	48
348	1453	73
4264	126	51
15	819	71
...	...	...
1946	860	41
57	772	72
2464	436	42
2608	570	34
5012	235	36

1371 rows × 2 columns

```
X_train
```

	artist	popularity
246	574	77
3873	1461	38
2311	348	53
2337	1160	52

```
Y_train
```

```
246     4.760883
3873    1.989550
2311    8.155550
2337    3.846667
2447    4.864667
          ...
4931    3.639767
3264    3.373917
1653    3.073767
2607    3.719767
2732    6.054433
Name: length, Length: 4113, dtype: float64
```

```
df.dtypes
```

```
index                int64
name                 int64
artist               int64
release_date         int64
length             float64
popularity           int64
danceability       float64
acousticness       float64
danceability.1     float64
energy             float64
instrumentalness   float64
key                  int64
liveness           float64
loudness           float64
speechiness        float64
tempo              float64
time_signature       int64
valence            float64
dtype: object
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5484 entries, 0 to 5483
Data columns (total 18 columns):
```

```
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   index             5484 non-null   int64  
 1   name              5484 non-null   int64  
 2   artist            5484 non-null   int64  
 3   release_date      5484 non-null   int64  
 4   length            5484 non-null   float64
 5   popularity        5484 non-null   int64  
 6   danceability      5484 non-null   float64
 7   acousticness      5484 non-null   float64
 8   danceability.1    5484 non-null   float64
 9   energy            5484 non-null   float64
 10  instrumentalness  5484 non-null   float64
 11  key               5484 non-null   int64  
 12  liveness          5484 non-null   float64
 13  loudness          5484 non-null   float64
 14  speechiness       5484 non-null   float64
 15  tempo             5484 non-null   float64
 16  time_signature    5484 non-null   int64  
 17  valence           5484 non-null   float64
dtypes: float64(11), int64(7)
memory usage: 943.1 KB
```

```
#Extracting Independent and dependent Variable
x= df.iloc[:, [2,3]].values
y= df.iloc[:, 2].values

# Splitting the dataset into training and test set.
X_train, X_test, Y_train, Y_test= train_test_split(x, y, test_size= 0.25, random_state=0)



#p=1=Minkowski=Mannahattan p=2=Minkowski=Euchlidean p=3=Chebyshev
classifier= KNeighborsClassifier(n_neighbors=3,p=2)
classifier.fit(X_train, Y_train)
classifier2= KNeighborsClassifier(n_neighbors=3,p=1)
classifier2.fit(X_train, Y_train)
```

```
KNeighborsClassifier(n_neighbors=3, p=1)
```

```
print("{}".format(classifier))
print("{}".format(classifier2))
```

```
KNeighborsClassifier(n_neighbors=3)
KNeighborsClassifier(n_neighbors=3, p=1)
```

```
#Predicting the test set result
Y_pred= classifier.predict(X_test)
Y_pred2= classifier2.predict(X_test)


from sklearn.metrics import confusion_matrix
```

```
cm= confusion_matrix(Y_test, Y_pred)
```

```
cm
```

```
array([[1, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 1, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 1, 0],
       [0, 0, 0, ..., 0, 1, 0],
       [0, 0, 0, ..., 0, 1, 0]])
```

```
acc=accuracy_score(Y_test, Y_pred)
acc
```

```
0.5463165572574763
```

```
cl=classification_report(Y_test, Y_pred)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undefine
  _warn_prf(average, modifier, msg_start, len(result))
```

```
cl
```

```
'              precision    recall  f1-score   support\n\n           1       1.00
1.00      1.00         1\n           2       0.00      0.00      0.00         0\n
5       1.00      0.50      0.67         2\n           6       0.33      0.50      0.40
2\n           7       0.00      0.00      0.00         0\n           8       0.00
0.00      0.00         0\n           9       0.00      0.00      0.00         1\n
10       0.00      0.00      0.00         2\n          16       1.00      1.00      1.0
```

```
from matplotlib.colors import ListedColormap
import numpy as nm
import matplotlib.pyplot as mtp
x_set, y_set = X_train, Y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, s
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shap
```

```
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
    c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('loudness')
mtp.ylabel('')
mtp.legend()
mtp.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-724b6e52b869> in <module>()
      2 import numpy as nm
      3 import matplotlib.pyplot as mtp
----> 4 x_set, y_set = X_train, Y_train
      5 x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:,
0].max() + 1, step  = 0.01),
      6 nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step =
0.01))

NameError: name 'X_train' is not defined
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
import warnings

 #IMPORTING NECESSARY LIBRARIES
```

```
df.head()
```

```
df.tail()
```

```
df.shape
#viewing number of rows and columns
```

```
df.isnull().sum()
#Data is clean and can continue to the Explorary Data Analysis stage


#Univariate analysis Type (Target features).
sns.countplot(df['loudness'], color='red')


#Extracting Independent and dependent Variable
x= df.iloc[:, [4,5]]
y= df.iloc[:, 2]

# Splitting the dataset into training and test set.
X_train, X_test, Y_train, Y_test= train_test_split(x, y, test_size= 0.25, random_state=0)



df.info()


from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()




classifier.fit(X_train, Y_train)


    # Predicting the Test set results
Y_pred = classifier.predict(X_test)


    # Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)
cm


# Visualising the Training set results
import matplotlib.pyplot as mtp
import numpy as nm
from matplotlib.colors import ListedColormap
x_set, y_set = X_train, Y_train
X1, X2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, s
mtp.contourf(X1, X2, classifier.predict(nm.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shap
mtp.xlim(X1.min(), X1.max())
mtp.ylim(X2.min(), X2.max())
for i, j in enumerate(nm.unique(y_set)):
   mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
                c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Naive Bayes (Training set)')
```

```
mtp.xlabel('loudness')
mtp.ylabel('')
mtp.legend()
mtp.show()
```

K MEANS CLUSTERING

```
df.head()
```

	index	name	artist	release_date	length	popularity	danceability	acousti
0	0	Smells Like Teen Spirit	Nirvana	1991	5.032000	74	0.502	0.0
1	1	Stairway to Heaven - Remaster	Led Zeppelin	1971	8.047167	78	0.338	0.5
2	2	Bohemian Rhapsody - Remastered 2011	Queen	1975	5.905333	74	0.392	0.2
3	3	Imagine - Remastered 2010	John Lennon	1971	3.131100	77	0.547	0.9
4	4	(I Can't Get No) Satisfaction - Mono Version	The Rolling Stones	1965	3.713550	77	0.723	0.0

```
df. columns
```

```
Index(['index', 'name', 'artist', 'release_date', 'length', 'popularity',
       'danceability', 'acousticness', 'danceability.1', 'energy',
       'instrumentalness', 'key', 'liveness', 'loudness', 'speechiness',
       'tempo', 'time_signature', 'valence'],
      dtype='object')
```

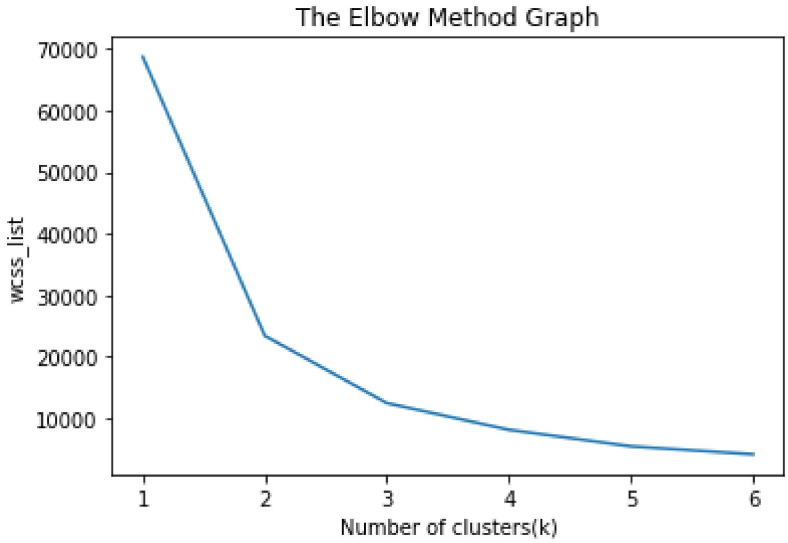
```
df.info
```

```
<bound method DataFrame.info of       index
name  \
```

```
0         0                       Smells Like Teen Spirit
1         1                 Stairway to Heaven - Remaster
2         2           Bohemian Rhapsody - Remastered 2011
3         3                      Imagine - Remastered 2010
4         4  (I Can't Get No) Satisfaction - Mono Version
...     ...                                           ...
5479   5479                           I'm In Your Mind
5480   5480                                 Cellophane
5481   5481                                  Hot Water
5482   5482           Vitamin C - 2004 Remastered Version
5483   5483                                          ~

                              artist  release_date    length  popularity  \
0                            Nirvana          1991  5.032000          74
1                       Led Zeppelin          1971  8.047167          78
2                              Queen          1975  5.905333          74
3                        John Lennon          1971  3.131100          77
4                 The Rolling Stones          1965  3.713550          77
...                              ...           ...       ...         ...
5479  King Gizzard & The Lizard Wizard          2014  3.559833          47
5480  King Gizzard & The Lizard Wizard          2014  3.179750          44
5481  King Gizzard & The Lizard Wizard          2014  3.396450          40
5482                               CAN          1972  3.567767          52
5483                     Touché Amoré          2011  1.485100           0

      danceability  acousticness  danceability.1  energy  instrumentalness  \
0            0.502      0.000025           0.502   0.912          0.000173
1            0.338      0.580000           0.338   0.340          0.003200
2            0.392      0.288000           0.392   0.402          0.000000
3            0.547      0.907000           0.547   0.257          0.183000
4            0.723      0.038300           0.723   0.863          0.031700
...            ...           ...             ...     ...               ...
5479         0.296      0.005910           0.296   0.776          0.801000
5480         0.432      0.002130           0.432   0.887          0.916000
5481         0.627      0.860000           0.627   0.609          0.890000
5482         0.643      0.006690           0.643   0.644          0.673000
5483         0.222      0.000258           0.222   0.959          0.000275

      key  liveness  loudness  speechiness    tempo  time_signature  valence
0       1    0.1060    -4.556       0.0564  116.761               4    0.720
1       9    0.1160   -12.049       0.0339   82.433               4    0.197
2       0    0.2430    -9.961       0.0536  143.883               4    0.228
3       0    0.0935   -12.358       0.0252   75.752               4    0.169
4       2    0.1280    -7.890       0.0338  136.302               4    0.931
...   ...       ...       ...          ...      ...             ...      ...
5479    6    0.5970    -5.630       0.0597   93.481               4    0.406
5480    7    0.1200    -6.175       0.1230   92.965               4    0.357
5481    9    0.1160    -9.387       0.0332   86.861               4    0.734
5482    4    0.1620   -12.615       0.0462  117.225               4    0.853
5483    6    0.1540    -5.134       0.2080  161.693               4    0.128

[5484 rows x 18 columns]>
```

```
x = df.iloc[:, [9,13]].values   #energy #loudness
```

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as mtp
wcss_list= []  #Initializing the list for the values of a.uynty

#Using for loop for iterations from 1 to 6.
for i in range(1, 7):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)   # measures how well a dataset is clustered by k means
mtp.plot(range(1, 7), wcss_list)
mtp.title('The Elbow Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```



Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

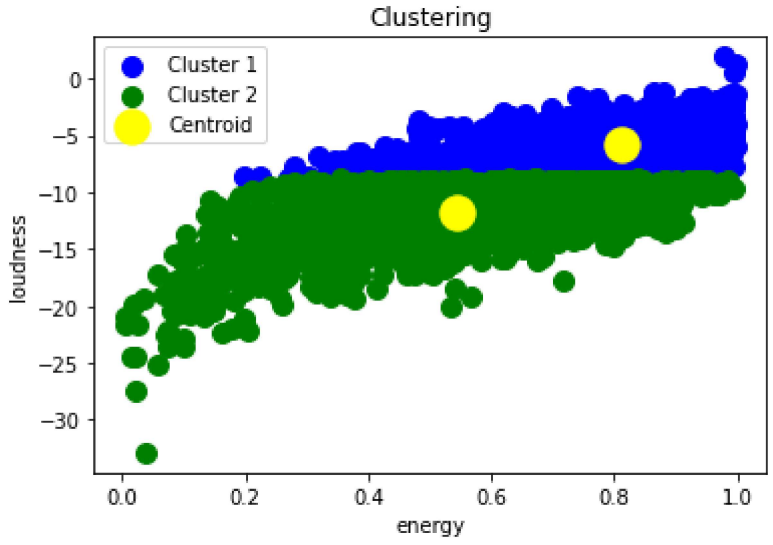
```
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=2, init='k-means++', random_state= 0)
y_predict= kmeans.fit_predict(x)
```

Double-click (or enter) to edit

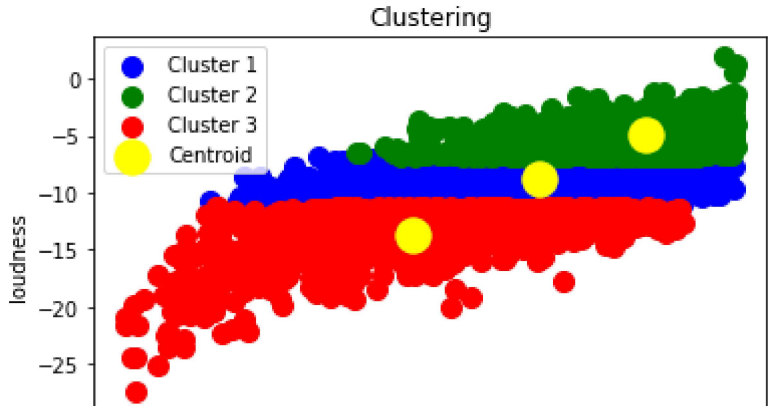
```
y_predict
```

```
array([0, 1, 1, ..., 1, 1, 0], dtype=int32)
```

```
#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluste
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yello
mtp.title('Clustering')
mtp.xlabel('energy')
mtp.ylabel('loudness')
mtp.legend()
mtp.show()
```




```
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=3, init='k-means++', random_state= 0)
y_predict= kmeans.fit_predict(x)
#visulaizing the clusters
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluste
mtp.scatter(x[y_predict == 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yello
mtp.title('Clustering')
mtp.xlabel('energy')
mtp.ylabel('loudness')
mtp.legend()
mtp.show()
```




```
x= df.iloc[:,:-1].values #extracting independent variable
```

```
y= df.iloc[:,3].values  #extracting depending variables
```

Double-click (or enter) to edit

```
df.head()
```

	index	name	artist	release_date	length	popularity	danceability	acousti
0	0	Smells Like Teen Spirit	Nirvana	1991	5.032000	74	0.502	0.0
1	1	Stairway to Heaven - Remaster	Led Zeppelin	1971	8.047167	78	0.338	0.5
2	2	Bohemian Rhapsody - Remastered 2011	Queen	1975	5.905333	74	0.392	0.2
3	3	Imagine - Remastered 2010	John Lennon	1971	3.131100	77	0.547	0.9
4	4	(I Can't Get No) Satisfaction - Mono Version	The Rolling Stones	1965	3.713550	77	0.723	0.0

```
import pandas as pd
df2 = pd.read_csv ('/content/rockmusic_2147239.csv')
```

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5484 entries, 0 to 5483
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   index             5484 non-null   int64  
 1   name              5484 non-null   object 
 2   artist            5484 non-null   object 
 3   release_date      5484 non-null   int64  
 4   length            5484 non-null   float64
 5   popularity        5484 non-null   int64  
 6   danceability      5484 non-null   float64
 7   acousticness      5484 non-null   float64
 8   danceability.1    5484 non-null   float64
 9   energy            5484 non-null   float64
 10  instrumentalness  5484 non-null   float64
 11  key               5484 non-null   int64  
 12  liveness          5484 non-null   float64
 13  loudness          5484 non-null   float64
 14  speechiness       5484 non-null   float64
 15  tempo             5484 non-null   float64
 16  time_signature    5484 non-null   int64  
 17  valence           5484 non-null   float64
dtypes: float64(11), int64(5), object(2)
memory usage: 771.3+ KB
```

```
df2.tail()
```

```
df2.isnull().sum()
```

```
index               0
name                0
artist              0
release_date        0
length              0
popularity          0
danceability        0
acousticness        0
danceability.1      0
energy              0
instrumentalness    0
key                 0
liveness            0
loudness            0
speechiness         0
tempo               0
time_signature      0
valence             0
dtype: int64
```

```
from sklearn.preprocessing import LabelEncoder
labelencoder=LabelEncoder()
#converting string into int using label encounter

cols = ['name','artist']

df2[cols] = df2[cols].apply(LabelEncoder().fit_transform)
#label encoder


df2.info
```

```
<bound method DataFrame.info of       index  name  artist  release_date    length
popularity  danceability  \
0         0  3931     942          1991  5.032000          74         0.502
1         1  4069     772          1971  8.047167          78         0.338
2         2   610    1045          1975  5.905333          74         0.392
3         3  2201     704          1971  3.131100          77         0.547
4         4     8    1516          1965  3.713550          77         0.723
...     ...   ...     ...           ...       ...         ...           ...
5479   5479  2146     751          2014  3.559833          47         0.296
5480   5480   789     751          2014  3.179750          44         0.432
5481   5481  1932     751          2014  3.396450          40         0.627
5482   5482  4884     239          1972  3.567767          52         0.643
5483   5483  5315    1627          2011  1.485100           0         0.222

      acousticness  danceability.1  energy  instrumentalness  key  liveness  \
0         0.000025           0.502   0.912          0.000173    1    0.1060
1         0.580000           0.338   0.340          0.003200    9    0.1160
2         0.288000           0.392   0.402          0.000000    0    0.2430
3         0.907000           0.547   0.257          0.183000    0    0.0935
4         0.038300           0.723   0.863          0.031700    2    0.1280
...            ...             ...     ...               ...  ...       ...
5479      0.005910           0.296   0.776          0.801000    6    0.5970
5480      0.002130           0.432   0.887          0.916000    7    0.1200
5481      0.860000           0.627   0.609          0.890000    9    0.1160
5482      0.006690           0.643   0.644          0.673000    4    0.1620
5483      0.000258           0.222   0.959          0.000275    6    0.1540

      loudness  speechiness    tempo  time_signature  valence
0       -4.556       0.0564  116.761               4    0.720
1      -12.049       0.0339   82.433               4    0.197
2       -9.961       0.0536  143.883               4    0.228
3      -12.358       0.0252   75.752               4    0.169
4       -7.890       0.0338  136.302               4    0.931
...        ...          ...      ...             ...      ...
5479    -5.630       0.0597   93.481               4    0.406
5480    -6.175       0.1230   92.965               4    0.357
5481    -9.387       0.0332   86.861               4    0.734
5482   -12.615       0.0462  117.225               4    0.853
5483    -5.134       0.2080  161.693               4    0.128

[5484 rows x 18 columns]>
```

```
#Extracting Independent and dependent Variable
x= df2.iloc[:, [2,3]].values
y= df2.iloc[:, 4].values

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)


x_train
```

```
array([[1293, 2010],
       [1346, 1993],
       [1565, 2014],
       ...,
       [ 256, 1970],
       [1539, 2011],
       [ 907, 2001]])
```

```
from sklearn.preprocessing import StandardScaler


df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5484 entries, 0 to 5483
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   index             5484 non-null   int64  
 1   name              5484 non-null   int64  
 2   artist            5484 non-null   int64  
 3   release_date      5484 non-null   int64  
 4   length            5484 non-null   float64
 5   popularity        5484 non-null   int64  
 6   danceability      5484 non-null   float64
 7   acousticness      5484 non-null   float64
 8   danceability.1    5484 non-null   float64
 9   energy            5484 non-null   float64
 10  instrumentalness  5484 non-null   float64
 11  key               5484 non-null   int64  
 12  liveness          5484 non-null   float64
 13  loudness          5484 non-null   float64
 14  speechiness       5484 non-null   float64
 15  tempo             5484 non-null   float64
 16  time_signature    5484 non-null   int64  
 17  valence           5484 non-null   float64
dtypes: float64(11), int64(7)
memory usage: 771.3 KB
```

```
df2.name = df2.name.astype('int64')
```

```
#function which return return of min-max eq
def norm(item):
    return (item - item.min())/(item.max() - item.min())
#apply norm function to each item in dataframe
df2 = df2.apply(norm)
df2
```

	index	name	artist	release_date	length	popularity	danceability	aco
0	0.000000	0.739049	0.542314	0.546875	0.203494	0.880952	0.508612	
1	0.000182	0.764993	0.444444	0.234375	0.329498	0.928571	0.342452	
2	0.000365	0.114683	0.601612	0.296875	0.239991	0.880952	0.397163	
3	0.000547	0.413800	0.405296	0.234375	0.124056	0.916667	0.554205	
4	0.000730	0.001504	0.872769	0.140625	0.148397	0.916667	0.732523	
...	...	...	...	...	...	...	...	
5479	0.999270	0.403459	0.432355	0.906250	0.141973	0.559524	0.299899	
5480	0.999453	0.148336	0.432355	0.906250	0.126089	0.523810	0.437690	
5481	0.999635	0.363226	0.432355	0.906250	0.135145	0.476190	0.635258	
5482	0.999818	0.918218	0.137594	0.250000	0.142304	0.619048	0.651469	
5483	1.000000	0.999248	0.936672	0.859375	0.055270	0.000000	0.224924	

5484 rows × 18 columns

```
df2.to_csv(r"aleena_prepro.csv")


#lab5
import matplotlib.pyplot as plt
import pandas as pd


data=pd.read_csv('rockmusic_2147239.csv')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5484 entries, 0 to 5483
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   index             5484 non-null   int64
 1   name              5484 non-null   object
 2   artist            5484 non-null   object
 3   release_date      5484 non-null   int64
```

```
 4   length            5484 non-null   float64
 5   popularity        5484 non-null   int64
 6   danceability      5484 non-null   float64
 7   acousticness      5484 non-null   float64
 8   danceability.1    5484 non-null   float64
 9   energy            5484 non-null   float64
 10  instrumentalness  5484 non-null   float64
 11  key               5484 non-null   int64
 12  liveness          5484 non-null   float64
 13  loudness          5484 non-null   float64
 14  speechiness       5484 non-null   float64
 15  tempo             5484 non-null   float64
 16  time_signature    5484 non-null   int64
 17  valence           5484 non-null   float64
dtypes: float64(11), int64(5), object(2)
memory usage: 771.3+ KB
```

```
data.isnull().sum()
```

```
index              0
name               0
artist             0
release_date       0
length             0
popularity         0
danceability       0
acousticness       0
danceability.1     0
energy             0
instrumentalness   0
key                0
liveness           0
loudness           0
speechiness        0
tempo              0
time_signature     0
valence            0
dtype: int64
```

```
df_reduced = data.fillna(data.mean())
df_reduced
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning: Dropping
  """Entry point for launching an IPython kernel.
```

	index	name	artist	release_date	length	popularity	danceability	acou
0	0	Smells Like Teen Spirit	Nirvana	1991	5.032000	74	0.502	
1	1	Stairway to Heaven - Remaster	Led Zeppelin	1971	8.047167	78	0.338	
2	2	Bohemian Rhapsody - Remastered 2011	Queen	1975	5.905333	74	0.392	
3	3	Imagine - Remastered 2010	John Lennon	1971	3.131100	77	0.547	
4	4	(I Can't Get No) Satisfaction - Mono Version	The Rolling Stones	1965	3.713550	77	0.723	
...	...	...	...	...	...	...	...	
5479	5479	I'm In Your Mind	King Gizzard & The Lizard Wizard	2014	3.559833	47	0.296	
5480	5480	Cellophane	King Gizzard & The Lizard Wizard	2014	3.179750	44	0.432	
5481	5481	Hot Water	King Gizzard & The Lizard Wizard	2014	3.396450	40	0.627	
		Vitamin C -						

```
df_clean = df_reduced.fillna("Unknown")
df_clean
```

	index	name	artist	release_date	length	popularity	danceability	acou
0	0	Smells Like Teen Spirit	Nirvana	1991	5.032000	74	0.502	
1	1	Stairway to Heaven - Remaster	Led Zeppelin	1971	8.047167	78	0.338	
2	2	Bohemian Rhapsody - Remastered 2011	Queen	1975	5.905333	74	0.392	
3	3	Imagine - Remastered 2010	John Lennon	1971	3.131100	77	0.547	
4	4	(I Can't Get No) Satisfaction - Mono Version	The Rolling Stones	1965	3.713550	77	0.723	
...	...	...	...	...	...	...	...	
5479	5479	I'm In Your Mind	King Gizzard & The Lizard Wizard	2014	3.559833	47	0.296	
5480	5480	Cellophane	King Gizzard & The Lizard Wizard	2014	3.179750	44	0.432	
5481	5481	Hot Water	King Gizzard & The Lizard Wizard	2014	3.396450	40	0.627	
5482	5482	Vitamin C - 2004 Remastered Version	CAN	1972	3.567767	52	0.643	
5483	5483	~	Touché Amoré	2011	1.485100	0	0.222	

```
df_clean.isnull().sum()
```

```
index           0
name            0
artist          0
release_date    0
```

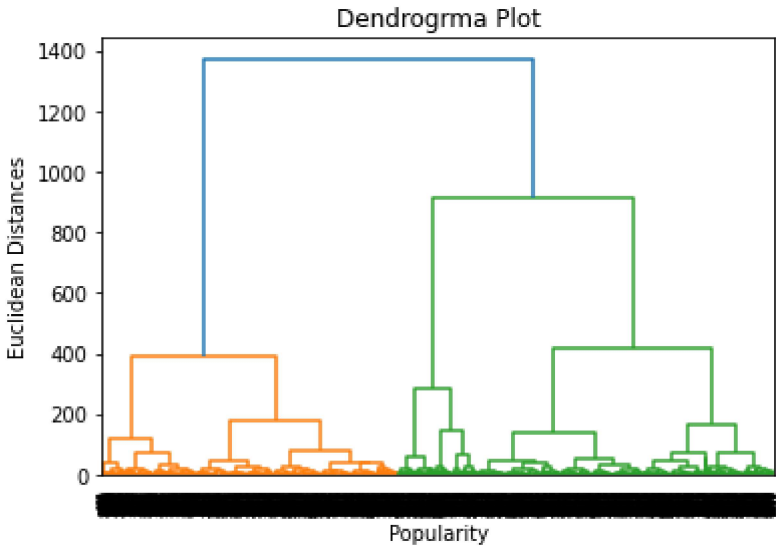
```
length              0
popularity          0
danceability        0
acousticness        0
danceability.1      0
energy              0
instrumentalness    0
key                 0
liveness            0
loudness            0
speechiness         0
tempo               0
time_signature      0
valence             0
dtype: int64
```

```
df_clean.columns
```

```
Index(['index', 'name', 'artist', 'release_date', 'length', 'popularity',
       'danceability', 'acousticness', 'danceability.1', 'energy',
       'instrumentalness', 'key', 'liveness', 'loudness', 'speechiness',
       'tempo', 'time_signature', 'valence'],
      dtype='object')
```

```
df_imp=df_clean[['popularity','length']]
```

```
import scipy.cluster.hierarchy as shc
dendro = shc.dendrogram(shc.linkage(df_imp, method="ward"))
plt.title("Dendrogrma Plot")
plt.ylabel("Euclidean Distances")
plt.xlabel("Popularity")
plt.show()
```




```
x = df_imp
```

```
#training the hierarchical model on dataset
from sklearn.cluster import AgglomerativeClustering
hc= AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
y_pred= hc.fit_predict(x)


#visulaizing the clusters
import numpy as np
x = np.array(x)
plt.scatter(x[y_pred == 0, 0], x[y_pred == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')
plt.scatter(x[y_pred == 1, 0], x[y_pred == 1, 1], s = 100, c = 'green', label = 'Cluster 2')
plt.scatter(x[y_pred== 2, 0], x[y_pred == 2, 1], s = 100, c = 'red', label = 'Cluster 3')
#plt.scatter(x[y_pred == 3, 0], x[y_pred == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')
#plt.scatter(x[y_pred == 4, 0], x[y_pred == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5
plt.legend()
plt.show()
```