MINISTRY OF EDUCATION AND SCIENCE OF RUSSIA

SAINT PETERSBURG STATE

ELECTRICAL ENGINEERING UNIVERSITY

"LETI" NAMED AFTER V.I. ULYANOV (LENIN)

Department of Computer Engineering

REPORT

for laboratory work #1

in the discipline "Machine Learning"

Topic: Classification

Student group 0308 ____________________ Dashkin D.Sh.

Teacher ____________________ Alhasan A.

Saint Petersburg

2024

Purpose of the work.

Acquiring and consolidating skills in data preprocessing and applying machine learning methods to solve classification problems.

Exercise.

It is necessary to process data on the following points:

a) Visualization of significant features (scatterplots, box plots, histograms)
b) Data cleaning (removing gaps, normalization, removing duplicates)
c) Data correlation (correlation matrix)

Selecting a dataset that meets the requirements:

1. The number of columns is not less than 10
2. Number of lines not less than 10000
3. Availability of passes
4. Removing gaps in data
5. Removing duplicates
6. Data normalization
7. Visualization via scatterplots
8. Boxes with whiskers
9. Histograms
10. Correlation matrix

During the laboratory work the following steps must be completed:

1. Training models and selecting parameters:

    a. K-nearest neighbors (KNN)

    b. Support Vector Machine (SVM)

    c. Decision Tree OR Random Forest

2. Evaluation of models

    a. Visualization of predicted values

    b. Prediction quality assessment (precision/recall/f1-score/ROC-AUC)

    c. Decision Tree Visualization OR Feature Importance Visualization for Random Forest

Completing the work.

1. Let's start with the basics of statistics.

Sample: is the analysis of a subset of data in order to identify meaningful information within a larger data set.

The sample can be divided into:

- Simple random sampling
- Mechanical (systematic) sampling - sorted by attribute (date, alphabet, etc.)
- Stratified sampling - the general population is divided into groups (strata). In each stratum, selection is carried out randomly or mechanically.
- Group sampling - the units of selection are not the objects themselves, but groups.

In a broad sense, data standardization/normalization is a stage of their pre-processing with the aim of bringing them to a certain format and presentation that ensure their correct use in multidimensional analysis, joint research, and complex analytical processing technologies.

Target: ensuring the possibility of correct comparison of observation values collected by the same methods but under different conditions.

These are not quite equivalent terms!

Standardization: mean 0, standard deviation 1, no upper/lower bound for max/min values // to see how the values in a sample deviate from the mean

Normalization: all values from 0 to 1 // to plot the same range for different variables that have different scales

Scatterplot: allows you to determine the type of dependence of 2 quantities

Positive correlation: the values of the variables increase together (direct dependence)

Negative correlation: one variable increases while the other decreases (inverse relationship)

The degree of correlation is determined by the density of points on the diagram.

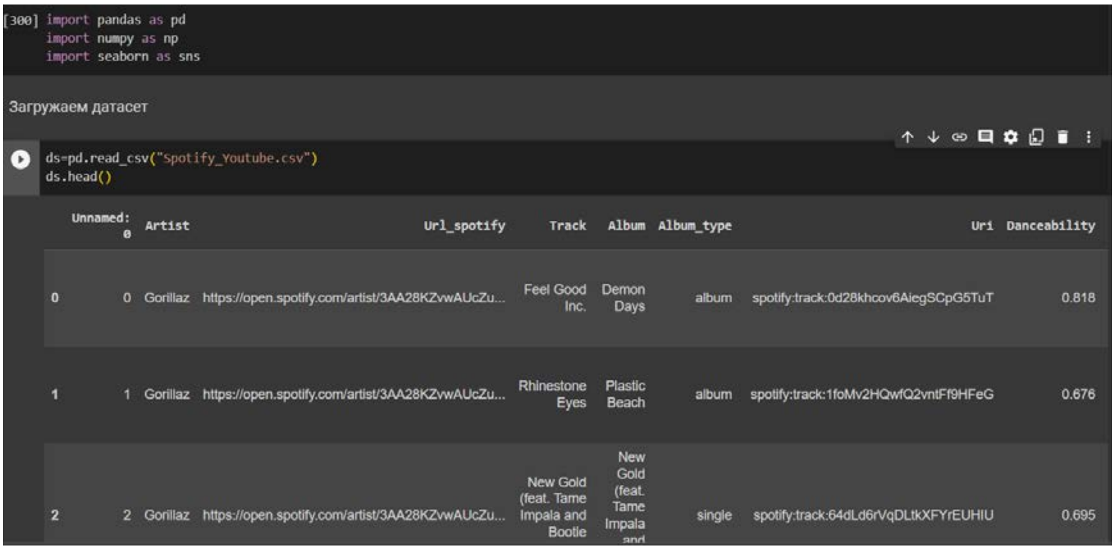
Points that are significantly away from the main clusters are called outliers.

Histogram: determination of emissions, distribution

Box with a mustache: this type of diagram conveniently shows the median (or, if necessary, the mean), lower and upper quartiles, minimum and maximum sample values, and outliers. Several such boxes can be drawn side by side to visually compare one distribution with another; they can be placed both horizontally and vertically. The distances between the different parts of the box allow you to determine the degree of dispersion and asymmetry of the data and identify outliers.

2. Preparing data for training.

First, we load the dataset:



```
[300] import pandas as pd
      import numpy as np
      import seaborn as sns
```

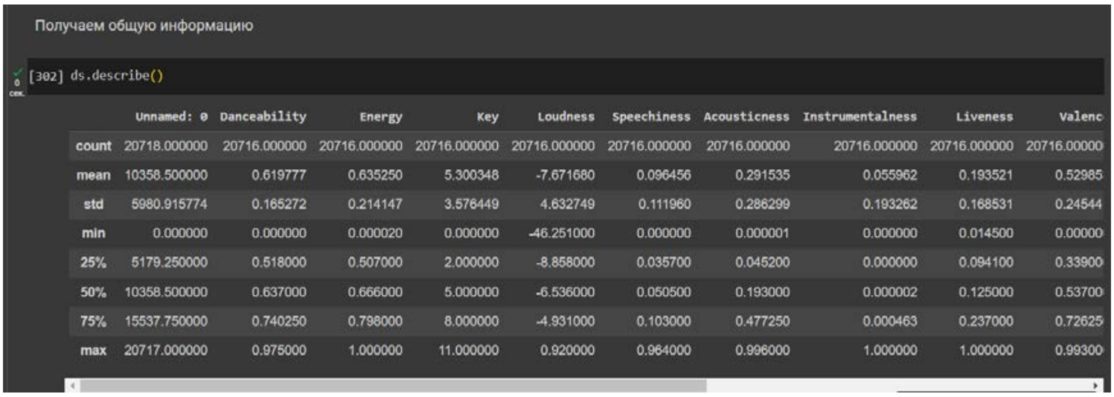
Загружаем датасет

```
ds=pd.read_csv("Spotify_Youtube.csv")
ds.head()
```

	Unnamed: 0	Artist	Url_spotify	Track	Album	Album_type	Uri	Danceability
0	0	Gorillaz	https://open.spotify.com/artist/3AA28KZvwAUcZu...	Feel Good Inc.	Demon Days	album	spotify:track:0d28khcov6AiegSCpG5TuT	0.818
1	1	Gorillaz	https://open.spotify.com/artist/3AA28KZvwAUcZu...	Rhinestone Eyes	Plastic Beach	album	spotify:track:1foMv2HQwfQ2vntFf9HFeG	0.676
2	2	Gorillaz	https://open.spotify.com/artist/3AA28KZvwAUcZu...	New Gold (feat. Tame Impala and Bootie	New Gold (feat. Tame Impala and	single	spotify:track:64dLd6rVqDLtkXFYrEUHIU	0.695

Figure 1. Loading the dataset

We receive general information:



Получаем общую информацию

```
[302] ds.describe()
```

	Unnamed: 0	Danceability	Energy	Key	Loudness	Speechiness	Acousticness	Instrumentalness	Liveness	Valenc
count	20718.000000	20716.000000	20716.000000	20716.000000	20716.000000	20716.000000	20716.000000	20716.000000	20716.000000	20716.00000
mean	10358.500000	0.619777	0.635250	5.300348	-7.671680	0.096456	0.291535	0.055962	0.193521	0.52985
std	5980.915774	0.165272	0.214147	3.576449	4.632749	0.111960	0.286299	0.193262	0.168531	0.24544
min	0.000000	0.000000	0.000020	0.000000	-46.251000	0.000000	0.000001	0.000000	0.014500	0.00000
25%	5179.250000	0.518000	0.507000	2.000000	-8.858000	0.035700	0.045200	0.000000	0.094100	0.33900
50%	10358.500000	0.637000	0.666000	5.000000	-6.536000	0.050500	0.193000	0.000002	0.125000	0.53700
75%	15537.750000	0.740250	0.798000	8.000000	-4.931000	0.103000	0.477250	0.000463	0.237000	0.72625
max	20717.000000	0.975000	1.000000	11.000000	0.920000	0.964000	0.996000	1.000000	1.000000	0.99300

Figure 2. Obtaining general information

We remove columns that do not carry significant information.

```
ds.info()
url_cols = ['Url_spotify', 'Uri', 'Url_youtube', 'Title', 'Description']
ds.drop(url_cols, axis=1, inplace=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20718 entries, 0 to 20717
Data columns (total 28 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Unnamed: 0        20718 non-null  int64
 1   Artist            20718 non-null  object
 2   Url_spotify       20718 non-null  object
 3   Track             20718 non-null  object
 4   Album             20718 non-null  object
 5   Album_type        20718 non-null  object
 6   Uri               20718 non-null  object
 7   Danceability      20716 non-null  float64
 8   Energy            20716 non-null  float64
 9   Key               20716 non-null  float64
 10  Loudness          20716 non-null  float64
 11  Speechiness       20716 non-null  float64
 12  Acousticness      20716 non-null  float64
 13  Instrumentalness  20716 non-null  float64
 14  Liveness          20716 non-null  float64
 15  Valence           20716 non-null  float64
 16  Tempo             20716 non-null  float64
 17  Duration_ms       20716 non-null  float64
 18  Url_youtube       20248 non-null  object
 19  Title             20248 non-null  object
 20  Channel           20248 non-null  object
 21  Views             20248 non-null  float64
 22  Likes             20177 non-null  float64
 23  Comments          20149 non-null  float64
 24  Description       19842 non-null  object
 25  Licensed          20248 non-null  object
 26  official_video    20248 non-null  object
 27  Stream            20142 non-null  float64
dtypes: float64(15), int64(1), object(12)
```

Figure 3. Removing unnecessary columns

Having received the dataset information again, we see that some columns have fewer unique values than rows in the dataset, which indicates gaps in them. Therefore, we will delete them, as well as duplicates.

```
ds.dropna(inplace=True)

ds.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 19549 entries, 0 to 20717
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Artist            19549 non-null  object
 1   Track             19549 non-null  object
 2   Album             19549 non-null  object
 3   Album_type        19549 non-null  object
 4   Danceability      19549 non-null  float64
 5   Energy            19549 non-null  float64
 6   Key               19549 non-null  float64
 7   Loudness          19549 non-null  float64
 8   Speechiness       19549 non-null  float64
 9   Acousticness      19549 non-null  float64
 10  Instrumentalness  19549 non-null  float64
 11  Liveness          19549 non-null  float64
 12  Valence           19549 non-null  float64
 13  Tempo             19549 non-null  float64
 14  Duration_ms       19549 non-null  float64
 15  Channel           19549 non-null  object
 16  Views             19549 non-null  float64
 17  Likes             19549 non-null  float64
 18  Comments          19549 non-null  float64
 19  Licensed          19549 non-null  object
 20  official_video    19549 non-null  object
 21  Stream            19549 non-null  float64
dtypes: float64(15), object(7)
memory usage: 3.4+ MB
```

```
ds = ds.drop_duplicates()
ds.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 19549 entries, 0 to 20717
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Artist            19549 non-null  object
 1   Track             19549 non-null  object
 2   Album             19549 non-null  object
 3   Album_type        19549 non-null  object
 4   Danceability      19549 non-null  float64
 5   Energy            19549 non-null  float64
 6   Key               19549 non-null  float64
 7   Loudness          19549 non-null  float64
 8   Speechiness       19549 non-null  float64
 9   Acousticness      19549 non-null  float64
 10  Instrumentalness  19549 non-null  float64
 11  Liveness          19549 non-null  float64
 12  Valence           19549 non-null  float64
 13  Tempo             19549 non-null  float64
 14  Duration_ms       19549 non-null  float64
 15  Channel           19549 non-null  object
 16  Views             19549 non-null  float64
 17  Likes             19549 non-null  float64
 18  Comments          19549 non-null  float64
 19  Licensed          19549 non-null  object
 20  official_video    19549 non-null  object
 21  Stream            19549 non-null  float64
dtypes: float64(15), object(7)
```

Figure 4. Removing gaps and duplicates

Now let's plot some scatterplots to analyze how some quantities relate to each other.

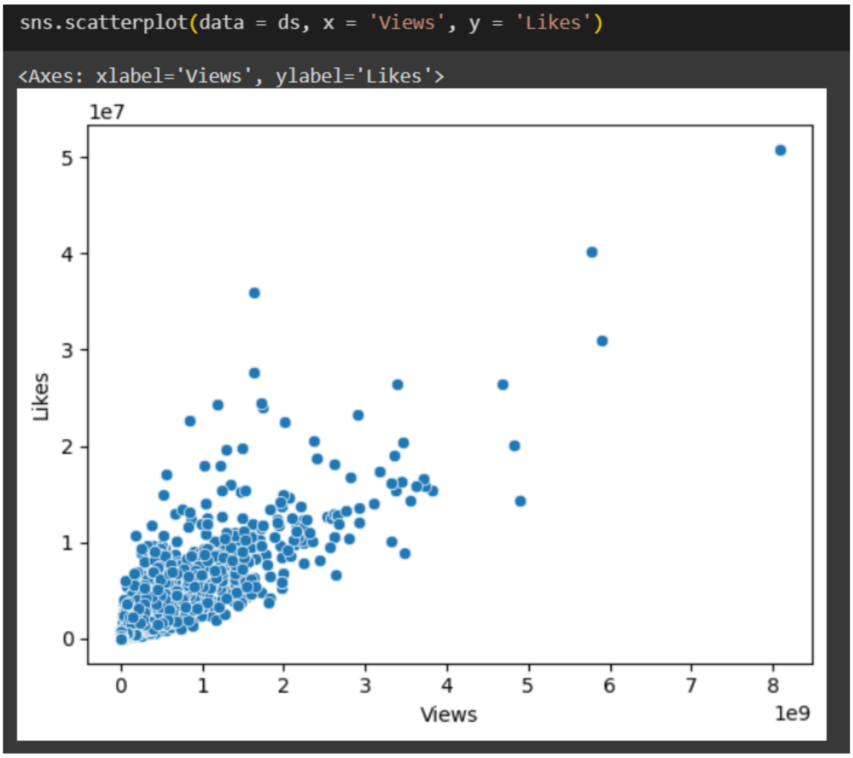


Figure 5. Scatter plot of likes from views

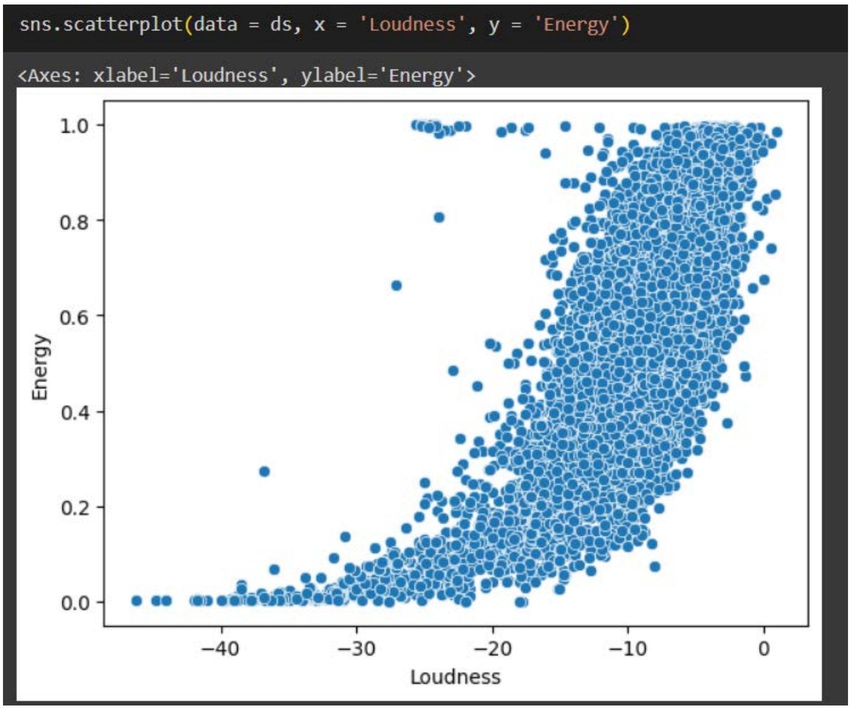


Figure 6. Energy dissipation diagram from loudness

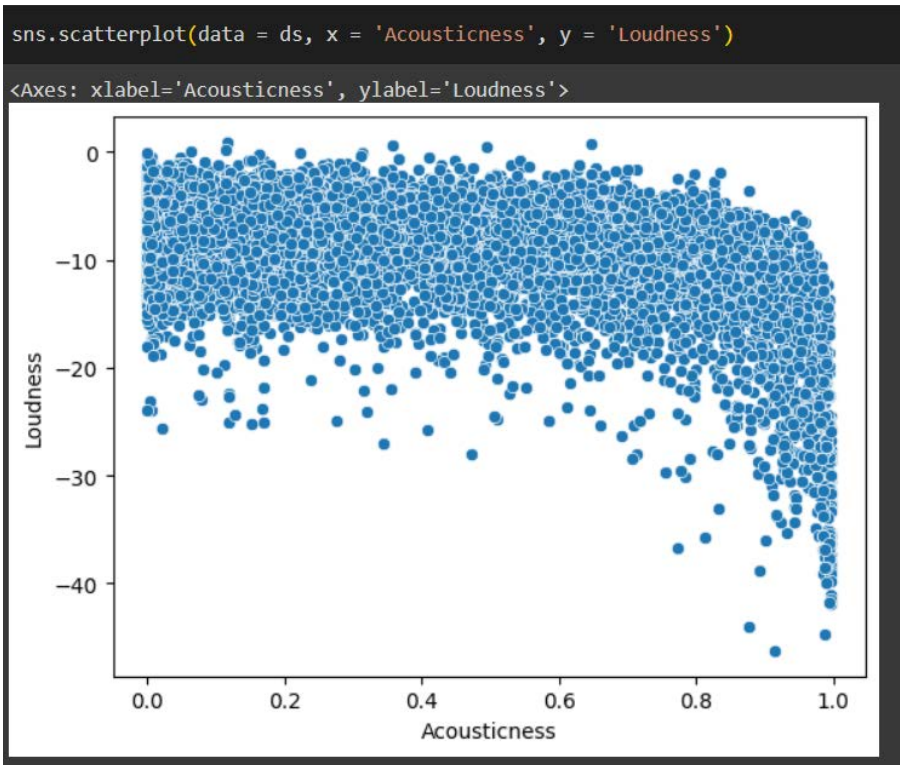


Figure 7. Loudness dispersion diagram from acoustics

Next, histograms were constructed.

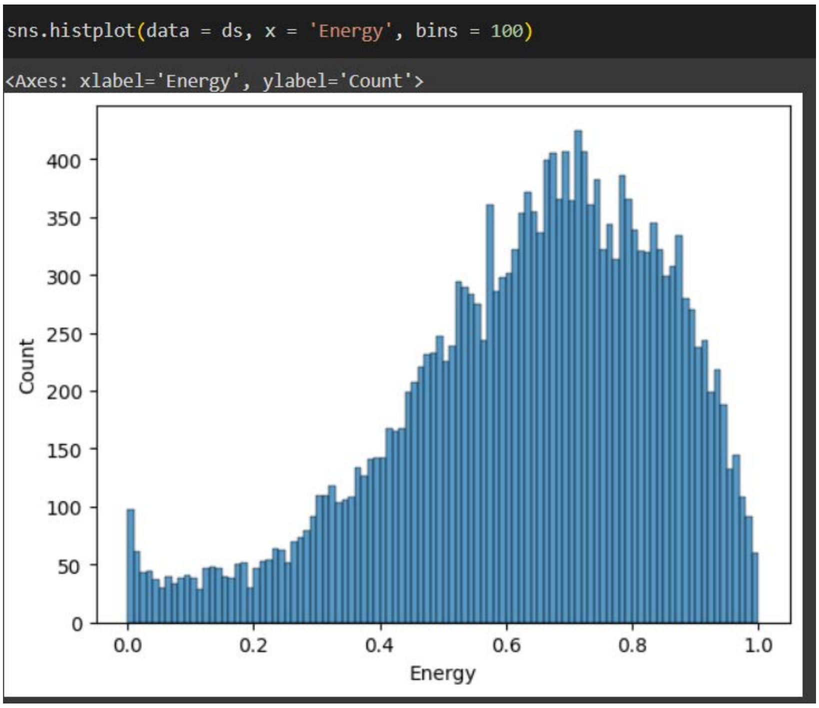


Figure 8. Histogram of the “Energy” parameter

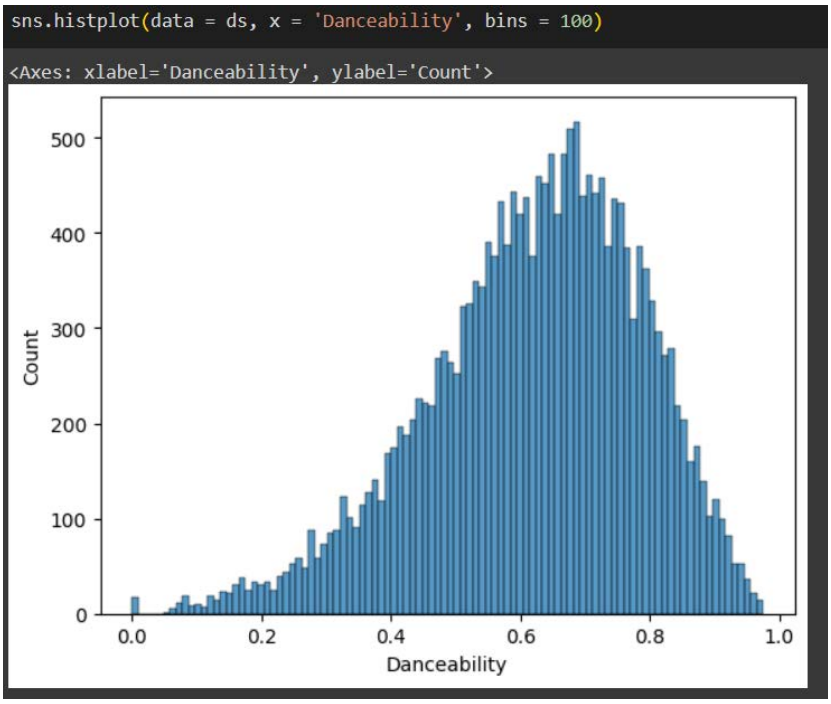


Figure 9. Histogram of the “Danceability” parameter

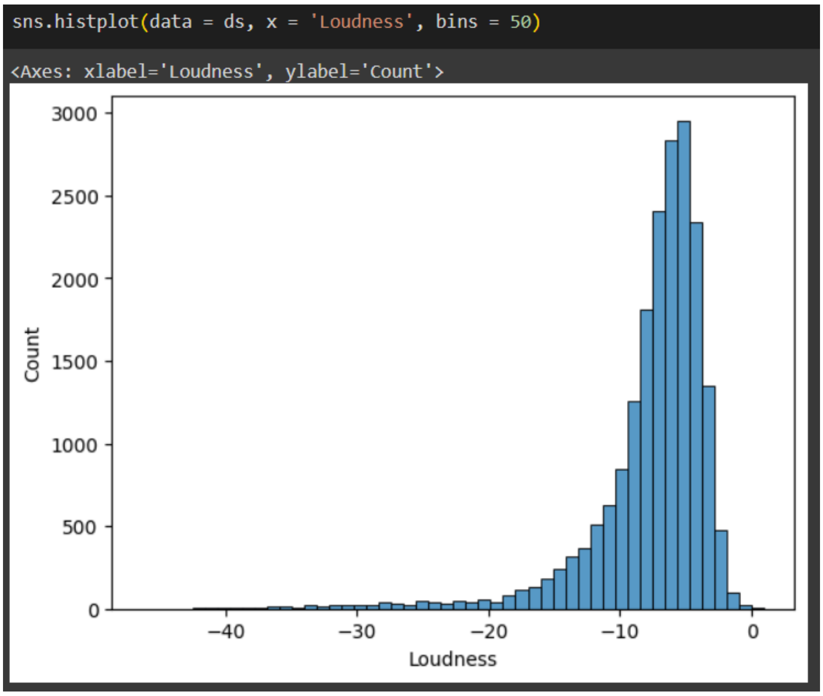


Figure 10. Histogram of the "Volume" parameter

Next, I built a box with whiskers for the “Danceability” parameter.

```python
sns.boxplot(data = ds, x ='Danceability')
```

```
<Axes: xlabel='Danceability'>
```

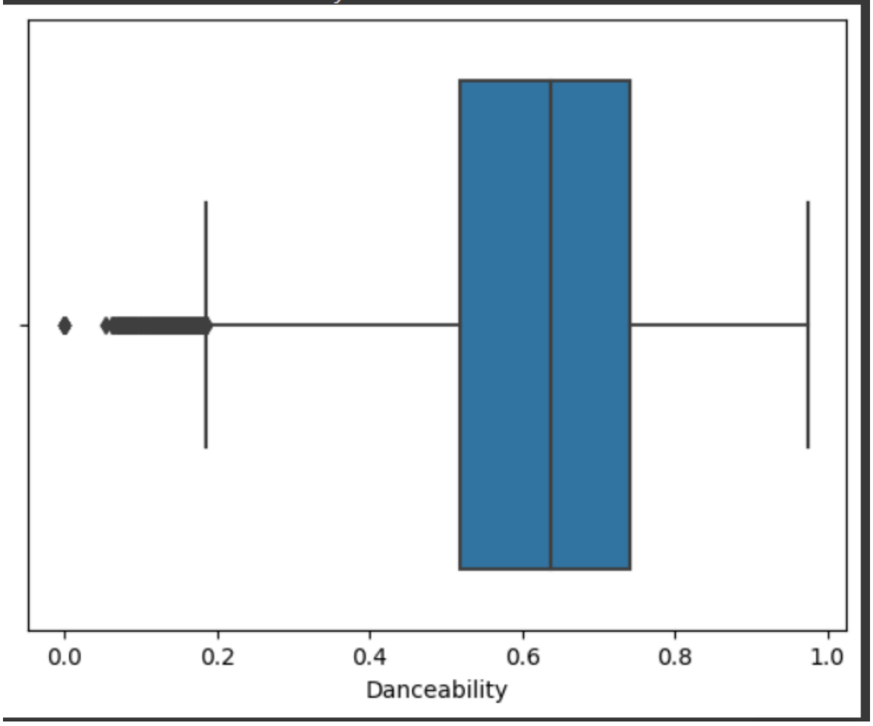


Figure 11. The "Danceability" parameter box with whiskers

Then the so-called "normalization" of the dataset was carried out. Figure 13 shows the positive result of its implementation.

```python
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import normalize
numeric_ds = ds.select_dtypes(exclude='object')
imp = SimpleImputer(strategy='mean')
new_ds = pd.DataFrame(data = imp.fit_transform(numeric_ds),columns=numeric_ds.columns, index =numeric_ds.index)
new_ds.info()
norm_ds = pd.DataFrame(data = normalize(new_ds.values),columns = new_ds.columns,index=new_ds.index)
norm_ds.describe()
del numeric_ds
del new_ds
```

```python
norm_ds.describe()
```

```python
for col in norm_ds.columns:
    ds[col] = norm_ds[col]
del norm_ds

ds.info()
```

Figure 12. Data normalization

```
norm_ds.describe()
```

	Danceability	Energy	Key	Loudness	Speechiness	Acousticness	Instrumentalness	Liveness	Valence	Temp
count	2.071800e+04	2.071800e+04	2.071800e+04	2.071800e+04	2.071800e+04	2.071800e+04	2.071800e+04	2.071800e+04	2.071800e+04	2.071800e+0
mean	5.721705e-08	5.586934e-08	4.865983e-07	-8.743316e-07	2.206691e-08	2.972039e-08	7.287886e-09	2.049823e-08	4.884645e-08	1.080595e-0
std	2.644836e-07	2.436513e-07	2.574440e-06	5.382052e-06	2.641381e-07	1.792648e-07	7.720841e-08	1.439456e-07	2.473936e-07	4.750357e-0
min	0.000000e+00	1.197772e-12	0.000000e+00	-1.797091e-04	0.000000e+00	1.151425e-14	0.000000e+00	6.749287e-12	0.000000e+00	0.000000e+0
25%	3.409665e-09	3.220051e-09	1.445980e-08	-4.450221e-07	3.283622e-10	4.247906e-10	0.000000e+00	7.777240e-10	2.446135e-09	6.752966e-0
50%	9.847569e-09	9.651288e-09	6.586741e-08	-1.150240e-07	1.062169e-09	2.752313e-09	4.968641e-14	2.492621e-09	7.855853e-09	1.999968e-0
75%	3.343086e-08	3.338173e-08	2.756983e-07	-3.520934e-08	4.359050e-09	1.476863e-08	2.681298e-11	9.695900e-09	2.803041e-08	6.680520e-0
max	6.638935e-06	7.377933e-06	8.514328e-05	1.066601e-07	9.963581e-06	6.751005e-06	3.959141e-06	6.711105e-06	7.158486e-06	1.172289e-0

Figure 13. Result of data normalization

New histogram of normalized data

```
plt.hist(ds['Danceability'], bins=100, range=(0.0,0.00000009))
plt.title('Danceability')
```

```
Text(0.5, 1.0, 'Danceability')
```

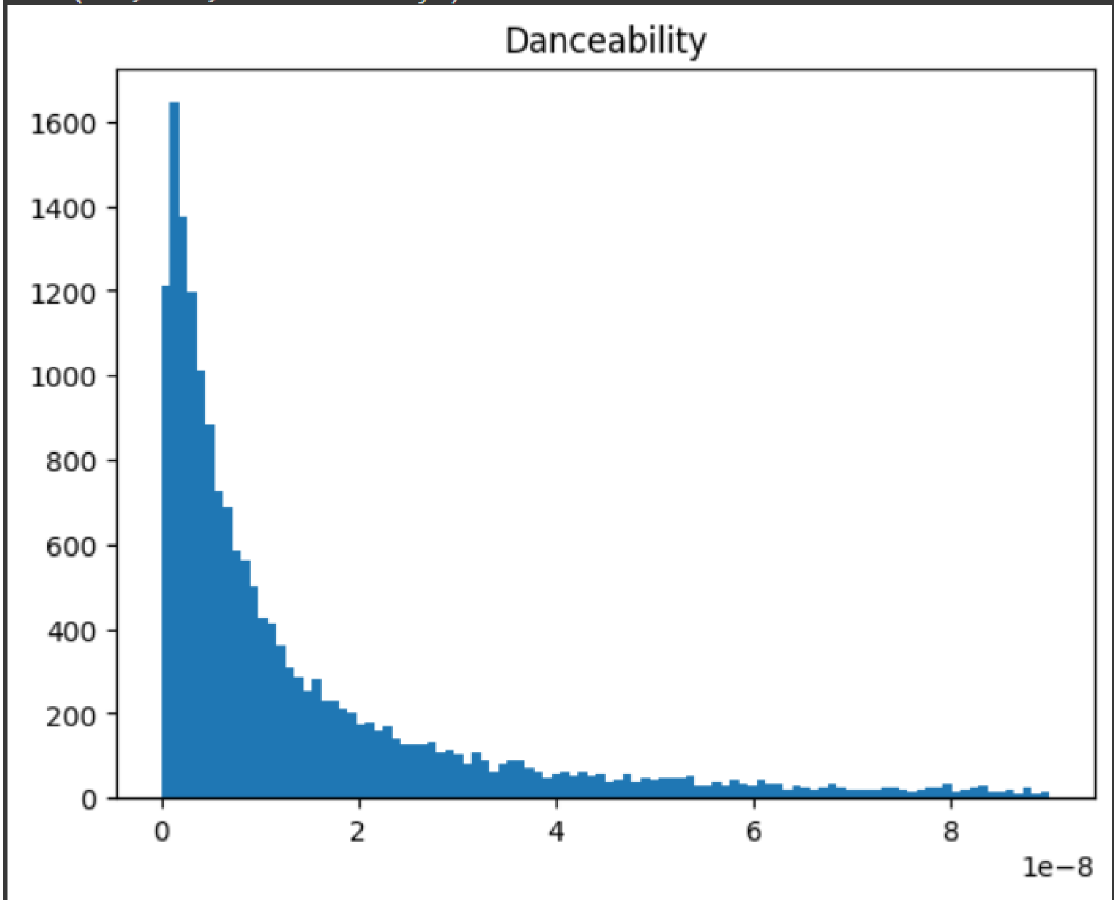


Figure 14. Histogram of the “Danceability” parameter

At the end, we will construct a correlation matrix to see how strongly the values are related to each other.

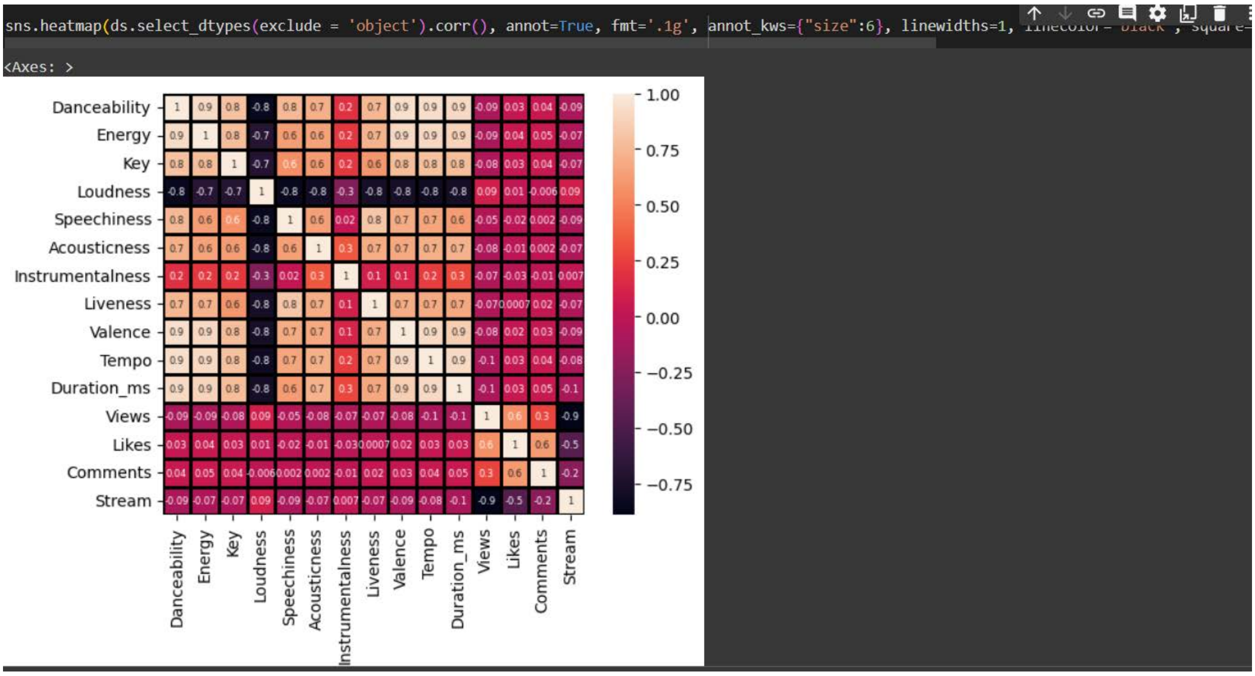


Figure 15. Correlation matrix

3. Classification

We have already prepared the data in the previous step. For classification, we need to define a target variable. I chose the "Streams" column as such a variable. Since this value is not discrete but continuous, it was necessary to come up with some method to get rid of the continuity. I came up with a function in which we divide the number of streams into three classes:

first class - streams are less than the average value of the given column

second class - streams between the mean and median values

third class - streams are greater than the median value

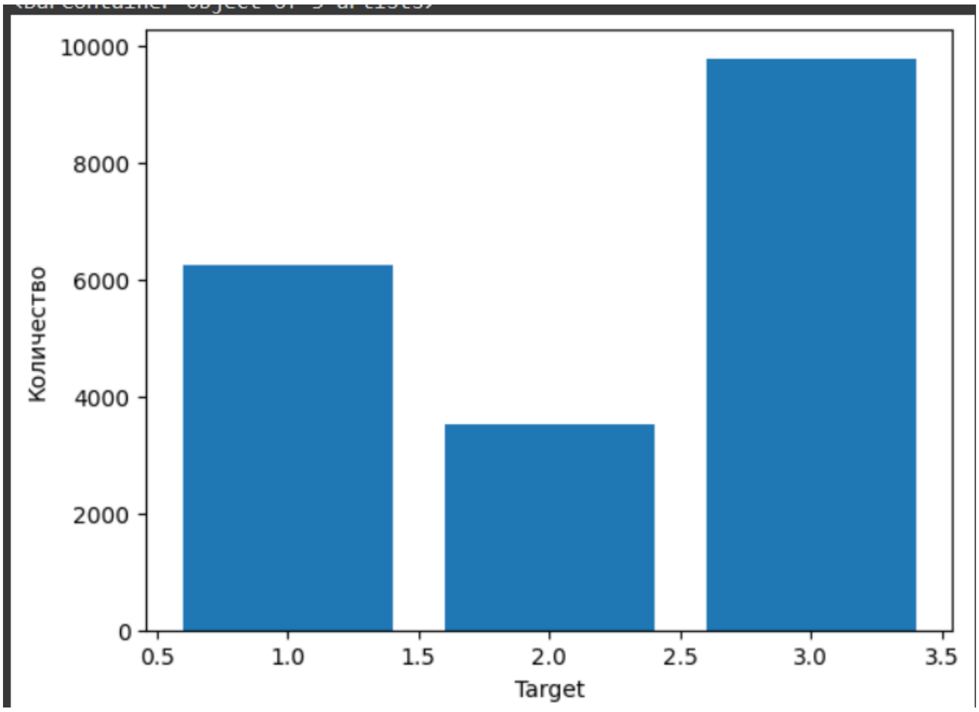


Figure 16. Distribution by classes

From the figure, it is clear that all classes are comparable to each other, so the distribution is correct, the data is not subject to deletion. The target column was added to our data frame and selected as the Y variable.

3.1. Let's split the data into training and testing.

We use the train_test_split() function from the sklearn library (Figure 17).

Теперь будем работать с X и Y

```
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=35)
```

```
[398] X_train.shape
      (15639, 16)
```

```
[399] X_test.shape
      (3910, 16)
```

Figure 17. Train-test distribution

3.2. k Nearest Neighbors (kNN).

This classification algorithm finds k nearest neighbors to the selected data. Based on its k neighbors, the object gets its class. That is, if all similar objects were class 2, then our object will also have class 2.

This algorithm, as can be seen from the report, has an accuracy of 63%. (Figure 18).

```
[ ] from sklearn.neighbors import KNeighborsClassifier

[ ] classifier_kNN = KNeighborsClassifier(n_neighbors=3)

[ ] classifier_kNN.fit(X_train, Y_train)

      KNeighborsClassifier
    KNeighborsClassifier(n_neighbors=3)

[ ] Y_pred = classifier_kNN.predict(X_test)

[ ] from sklearn.metrics import confusion_matrix ,classification_report,ConfusionMatrixDisplay

[ ] print(classification_report(Y_pred,Y_test))

                  precision    recall  f1-score   support

               1       0.69      0.56      0.62      1526
               2       0.20      0.37      0.26       392
               3       0.72      0.71      0.72      1992

        accuracy                           0.62      3910
       macro avg       0.54      0.55      0.53      3910
    weighted avg       0.66      0.62      0.63      3910
```

Figure 18. kNN algorithm

The ROC curve is shown in Figure 19.

```
from sklearn.metrics import RocCurveDisplay
RocCurveDisplay.from_predictions(Y_test, Y_pred, pos_label=1)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7b63dc4bbd60>
```

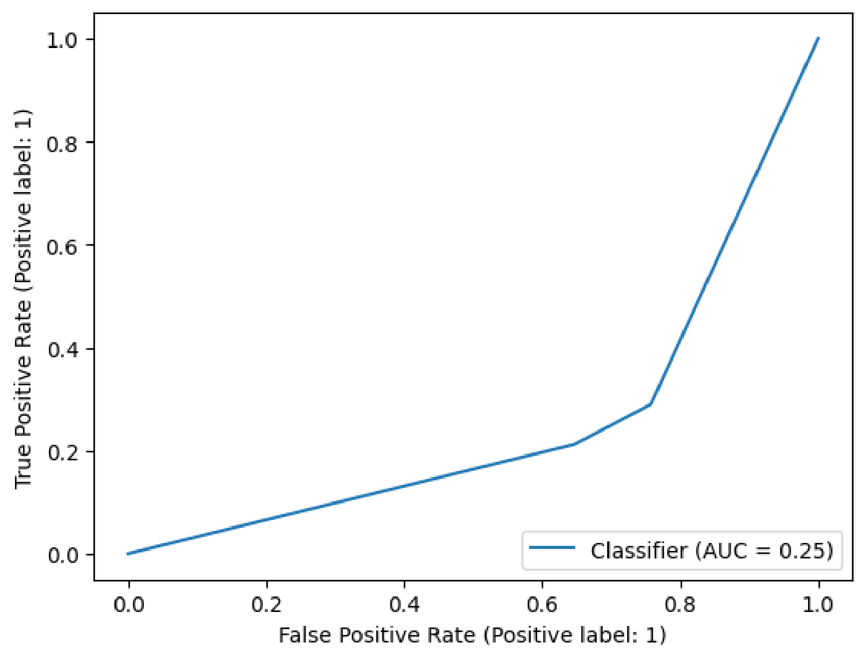


Figure 19. ROC curve

3.3. Support Vector Machine (SVM) Method (SVC).

This method works on the principle of logistic regression. That is, we want to find a separating hyperplane for our data. A visual example is shown in Figure 20.

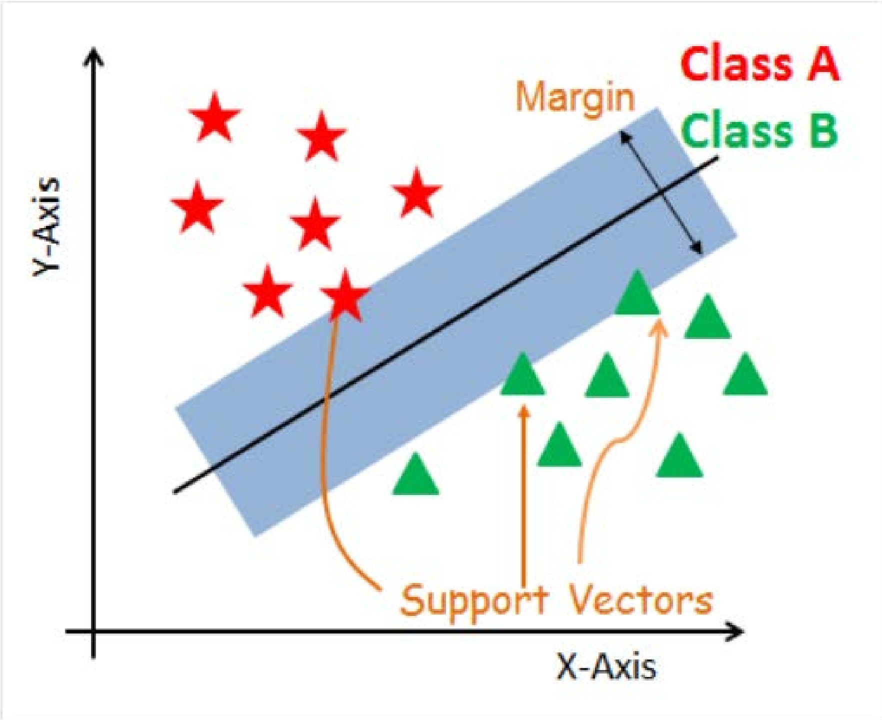


Figure 20. Visualization (2 classes).

Let's make a classifier, train it, and draw the ROC curve (Figures 21-22).

```
[408] from sklearn.pipeline import make_pipeline
      from sklearn.preprocessing import StandardScaler
      from sklearn.svm import SVC

[409] classifier_SVC = SVC(gamma='auto',probability=True)

[410] classifier_SVC.fit(X_train, Y_train)

      SVC
      SVC(gamma='auto', probability=True)

[427] Y_pred=classifier_SVC.predict(X_test)

      from sklearn.metrics import RocCurveDisplay
      RocCurveDisplay.from_predictions(Y_test, Y_pred, pos_label=1)
```

Figure 21. SVM algorithm

```
from sklearn.metrics import RocCurveDisplay
RocCurveDisplay.from_predictions(Y_test, Y_pred, pos_label=1)

<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7b63de7a6c50>
```

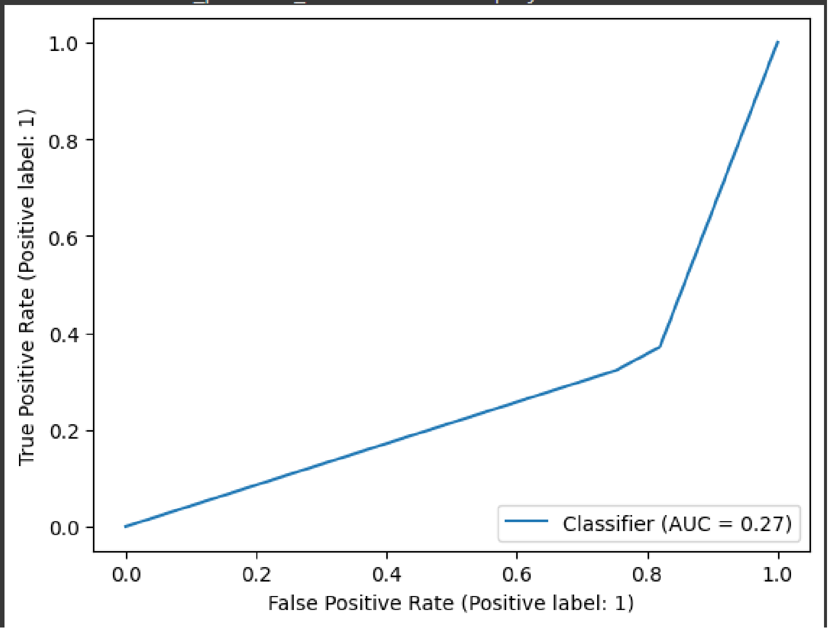


Figure 22. ROC curve of the SVM algorithm

3.4. Decision Tree.

It works on the principle of dividing features into several parts. That is, we take, for example, feature_1 and do that with feature_1 <1,000 we go further to the left leaf, with feature_1> =1,000 to the right one. And so on. In the end, we get a large tree that can predict the class by features.

Let's make a classifier, train it, find the accuracy and draw a tree (Figure 23-25). The accuracy was 78%.

```
[413] from sklearn.tree import DecisionTreeClassifier

[438] model = DecisionTreeClassifier(max_features = 1)

[439] model.fit(X_train, Y_train)

          DecisionTreeClassifier
      DecisionTreeClassifier(max_features=1)

[440] Y_pred = model.predict(X_test)

[ ] print(classification_report(Y_pred,Y_test))

              precision    recall  f1-score   support

           1       0.76      0.97      0.85       995
           2       0.75      0.45      0.56      1187
           3       0.80      0.89      0.84      1728

    accuracy                           0.78      3910
   macro avg       0.77      0.77      0.75      3910
weighted avg       0.77      0.78      0.76      3910
```

Figure 23. Report on the Decision Tree algorithm

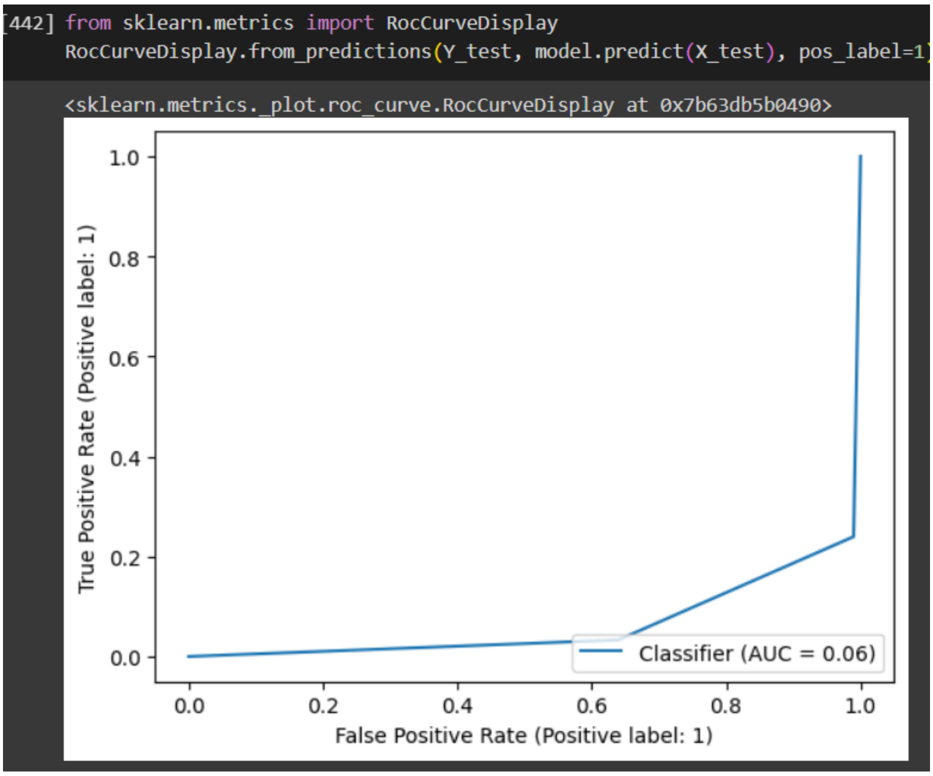


Figure 24. ROC curve of the Decision Tree algorithm

In this algorithm, I decided to change the max_features parameter and set it equal to one.

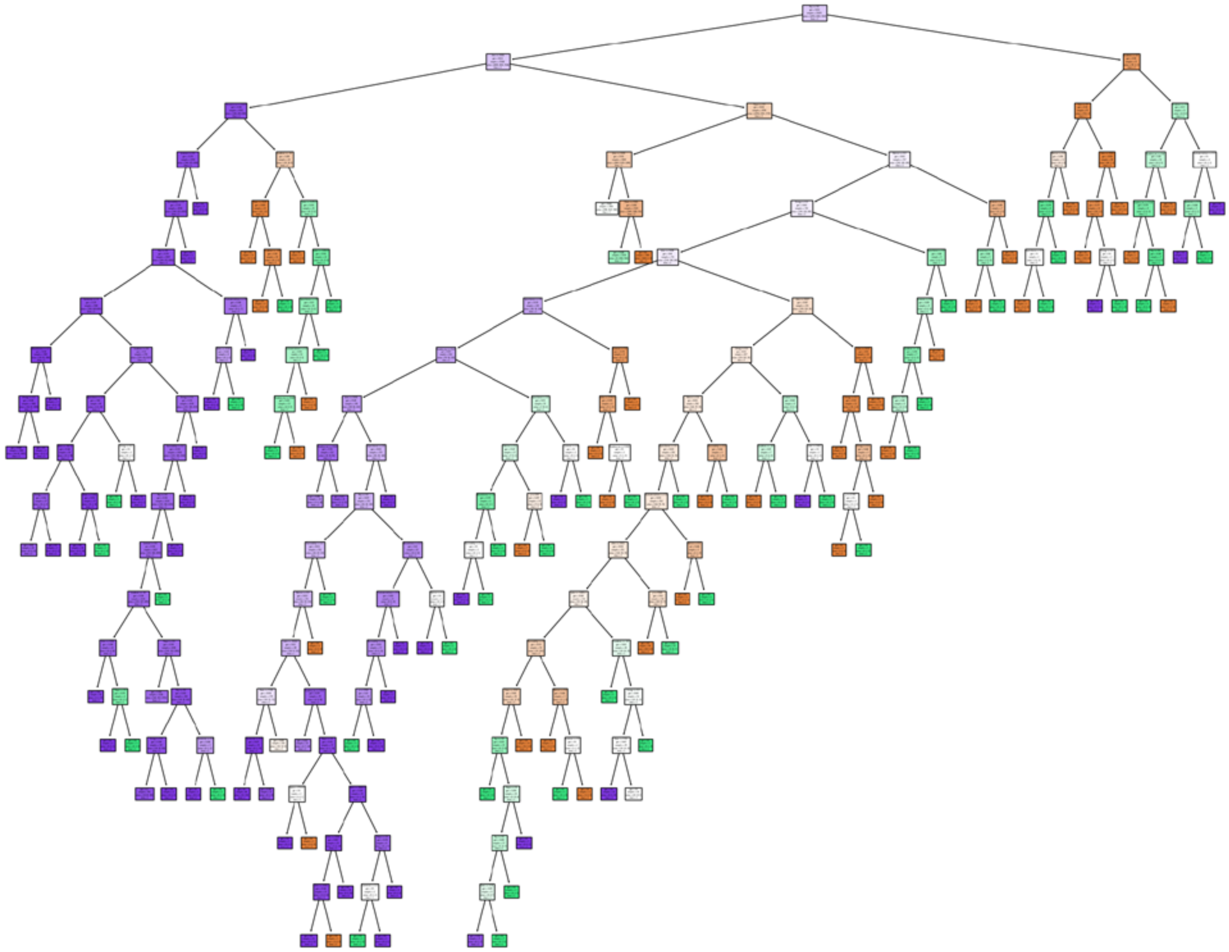


Figure 25. Drawing of a tree

3.5. Metrics

Let's create a function to calculate metrics (Figure 26) and check all metrics on test data for each classifier (Figures 27-29).

```
[ ] from sklearn.metrics import precision_score
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import recall_score
    from sklearn.metrics import f1_score
    from sklearn.metrics import roc_auc_score
```

```
def metrics(type_, Y_test, Y_pred, Y_score):
  print(type_, " Metrics: \n")
  print("Accuracy: ", accuracy_score(Y_test, Y_pred))
  print("Precision: ", precision_score(Y_test, Y_pred, average=None))
  print("Recall: ", recall_score(Y_test, Y_pred, average=None))

  print("F1 score: ", f1_score(Y_test, Y_pred, average=None))
  print("ROC-AUC score: ", roc_auc_score(Y_test, Y_score, multi_class='ovr'))
  print()
```

Figure 26. Function for outputting metrics

```
[ ] metrics("KNN classifier", Y_test, classifier_kNN.predict(X_test), classifier_kNN.predict_proba(X_test))

KNN classifier  Metrics:

Accuracy:  0.6255754475703325
Precision:  [0.58587168 0.34782609 0.71153846]
Recall:  [0.71013354 0.1915493  0.72963155]
F1 score:  [0.64204545 0.24704814 0.72047143]
ROC-AUC score:  0.720589231698141
```

Figure 27. kNN metrics

```
metrics("SVM", Y_test, classifier_SVC.predict(X_test), classifier_SVC.predict_proba(X_test))

SVM  Metrics:

Accuracy:  0.6230179028132993
Precision:  [0.62725137 0.29875519 0.65342809]
Recall:  [0.62922231 0.10140845 0.81110535]
F1 score:  [0.62823529 0.15141956 0.72377865]
ROC-AUC score:  0.7195337983677201
```

Figure 28. SVM metrics

```
metrics("Decision Tree", Y_test, model.predict(X_test), model.predict_proba(X_test))

Decision Tree  Metrics:

Accuracy:  0.7754475703324808
Precision:  [0.97286432 0.44818871 0.88657407]
Recall:  [0.76040848 0.74929577 0.79501816]
F1 score:  [0.85361552 0.56088561 0.83830369]
ROC-AUC score:  0.9143193287600555
```

Figure 29. Decision Tree Metrics

Conclusion:

I'm byacquired and consolidated skills in data preprocessing and application of machine learning methods to solve classification problems.

Link to code