МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра вычислительной техники

Отчет по лабораторной работе №1
по дисциплине «Машинное обучение»

Студент гр. 0306 Семенов М.Д.
Преподаватель Ручкан А.Д.

Санкт-Петербург
2023

Оглавление

Цель работы:

Получение и закрепление навыков предобработки данных и применения методов машинного обучения для решения задач регрессии.

Набор данных берется из лабораторной работы №1.

В ходе выполнения лабораторной работы должны быть выполнены следующие этапы:

1. Предварительная обработка данных
    a. Визуализация значимых признаков (диаграммы рассеяния, ящики с усами, гистограммы)
    b. Очистка данных (удаление пропусков, нормализация, удаление дубликатов)
2. Обучение моделей и подбор параметров (где применимо):
    a. Линейная регрессия
    b. LASSO
    c. Ридж-регрессия
3. Оценка моделей
    a. Вывод метрик
    b. Построение графиков

В рамках защиты лабораторной работы необходимо продемонстрировать jupyter-notebook с кодом, быть готовым пояснить выполненные действия, продемонстрировать базовое понимание работы используемых в работе методов.

Теория

Начнем с основ статистики.

Выборка – это анализ подмножества данных с целью выявить значимую информацию в большем наборе данных.

Выборку можно разделить на:

Простая случайная выборка

Механическая (систематическая) выборка - упорядочены по признаку (дата, алфавит и т.д.)

Стратифицированная выборка - генеральная совокупность разбивается на группы (страты). В каждой страте отбор осуществляется случайным или механическим образом.

Групповая выборка - единицами отбора выступают не сами объекты, а группы. В широком смысле стандартизация/нормализация данных представляет собой этап их предобработки с целью приведения к определённому формату и представлению, которые обеспечивают их корректное применение в многомерном анализе, совместных исследованиях, сложных технологиях аналитической обработки.

Цель: обеспечение возможности корректного сравнения значений наблюдений, собранных одними и теми же методами, но в различных условиях.

Стандартизация/нормализация - это не совсем равнозначные термины!

●Стандартизация: среднее значение 0, стандартное отклонение 1, нет верхней/нижней границы для максимальных/минимальных значений , чтобы узнать, как значения в выборке отклоняются от среднего.

●Нормализация: все значения от 0 до 1 // чтобы построить одинаковый диапазон для различных переменных, имеющих разные масштабы.

Диаграмма рассеивания - позволяет определить тип зависимости 2х величин.

Положительная корреляция: значения переменных увеличиваются вместе(прямая зависимость)

Отрицательная корреляция: одна переменная увеличивается, в то же времядругая уменьшается (обратная зависимость)

Степень корреляции определяется плотностью точек на диаграмме

Точки, которые значительно отдаляются от главных кластеров, называют выбросами.

Гистограмма - определение выбросов, распределения

Ящик с усами - такой вид диаграммы в удобной форме показывает медиану (или, если нужно, среднее), нижний и верхний квартили, минимальное и максимальное значение выборки и выбросы. Несколько таких ящиков можно нарисовать бок о бок, чтобы визуально сравнивать одно распределение с другим; их можно располагать как горизонтально, так и вертикально.

Расстояния между различными частями ящика позволяют определить степень разброса (дисперсии) и асимметрии данных и выявить выбросы.

Выполнение программы:

Загрузим наш датасет для дальнейшей работы,используя метод библиотеки pandas - read_csv().Затем выведем результат для просмотра.

```
import pandas as pd
ds = pd.read_csv("train.csv")
ds
```

	Unnamed: 0	track_id	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	...	loudness	mode
0	0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	73	230666	False	0.676	0.4610	...	-6.746	0
1	1	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	55	149610	False	0.420	0.1660	...	-17.235	1
2	2	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	57	210826	False	0.438	0.3590	...	-9.734	1
3	3	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Sou...	Can't Help Falling In Love	71	201933	False	0.266	0.0596	...	-18.515	1
4	4	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	82	198853	False	0.618	0.4430	...	-9.681	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
113995	113995	2C3TZjDRiAzdyViavDJ217	Rainy Lullaby	#mindfulness - Soft Rain for Mindful Meditatio...	Sleep My Little Boy	21	384999	False	0.172	0.2350	...	-16.393	1
113996	113996	1hIz5L4lB9hN3WRYPOCGPw	Rainy Lullaby	#mindfulness - Soft Rain for Mindful Meditatio...	Water Into Light	22	385000	False	0.174	0.1170	...	-18.318	0
113997	113997	6x8ZfSoqDjuNa5SVP5QjvX	Cesária Evora	Best Of	Miss Perfumado	22	271466	False	0.629	0.3290	...	-10.895	0
113998	113998	2e6sXL2bYv4bSz6VTdnfLs	Michael W. Smith	Change Your World	Friends	41	283893	False	0.587	0.5060	...	-10.889	1
113999	113999	2hETkH7cOfqmz3LqZDHZf5	Cesária Evora	Miss Perfumado	Barbincor	22	241826	False	0.526	0.4870	...	-10.204	0

114000 rows × 21 columns

Рис.1.Загрузка датасета

Удалим из датасета не нужные нам поля.

```
notUse = ['track_id','artists', 'album_name',  'track_name','explicit','track_genre']
ds.drop(notUse, axis=1, inplace=True)
```

Рис.2.Удаление ненужных столбцов

Получив снова информацию о датасете, видим, что в некоторых столбцах уникальных значений меньше, чем строк в данном датасете, что говорит о пропусках в них. Поэтому удалим их, а также и дубликаты.

```
dupls = ds[ds.duplicated()]
print('number of duplicate rows: ', dupls.shape)
ds = ds.drop_duplicates()
dupls = ds[ds.duplicated()]
print('number of duplicate rows: ', dupls.shape)
```

```
number of duplicate rows:  (0, 21)
number of duplicate rows:  (0, 21)
```

```
ds = ds.dropna()
ds.count()
print(ds.isnull().sum())
```

```
Unnamed: 0          0
track_id            0
artists             0
album_name          0
track_name          0
popularity          0
duration_ms         0
explicit            0
danceability        0
energy              0
key                 0
loudness            0
mode                0
speechiness         0
acousticness        0
instrumentalness    0
liveness            0
valence             0
tempo               0
time_signature      0
track_genre         0
dtype: int64
```

Рис.3. Удаление пропусков и дубликатов

Теперь построим несколько диаграмм рассеивания для того, чтобы проанализировать как некоторые величины соотносятся друг с другом. Например,как свзаны популярность и танцевальность

```
fig,ax = plt.subplots(figsize=(7,5))
ax.scatter(ds['popularity'],ds['danceability'],color = 'blue')
ax.set_xlabel('popularity')
ax.set_ylabel('danceability')
plt.show()
```

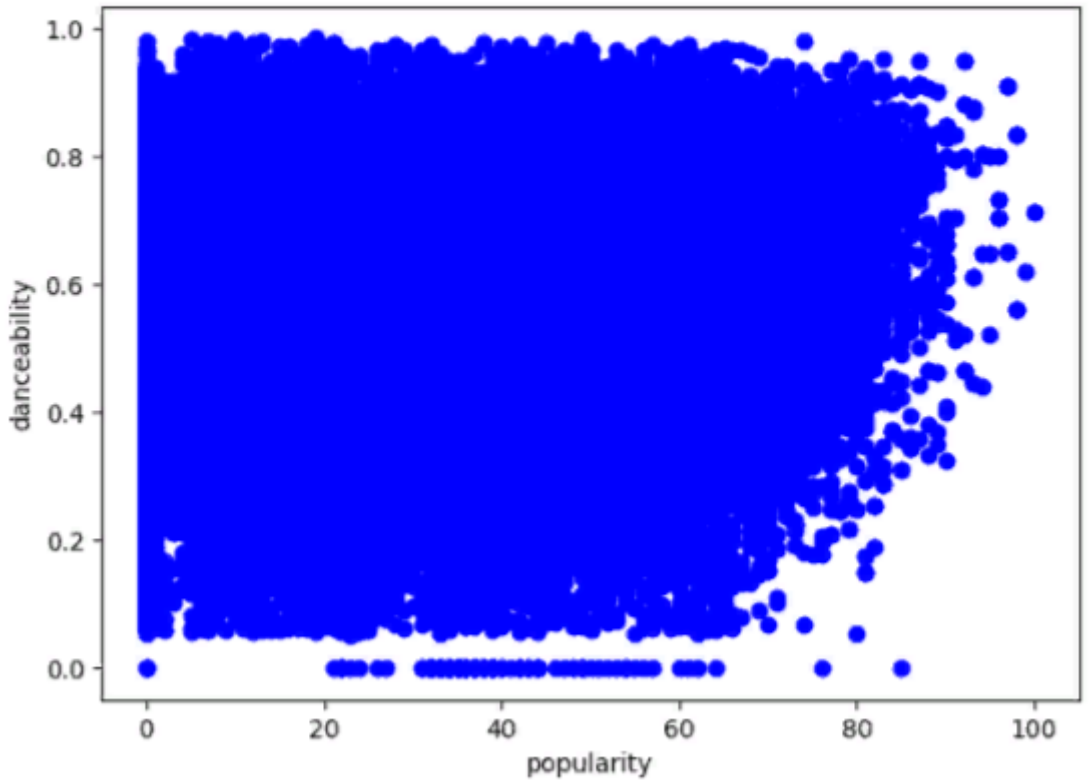


Рис.5.Диаграмма рассеивания популярности от танцевальности

Посмотрим как связаны танцевальность и энергичность

```
fig,ax = plt.subplots(figsize=(7,5))
ax.scatter(ds['energy'],ds['loudness'],color = 'blue')
ax.set_xlabel('energy')
ax.set_ylabel('loudness')
plt.show()
```

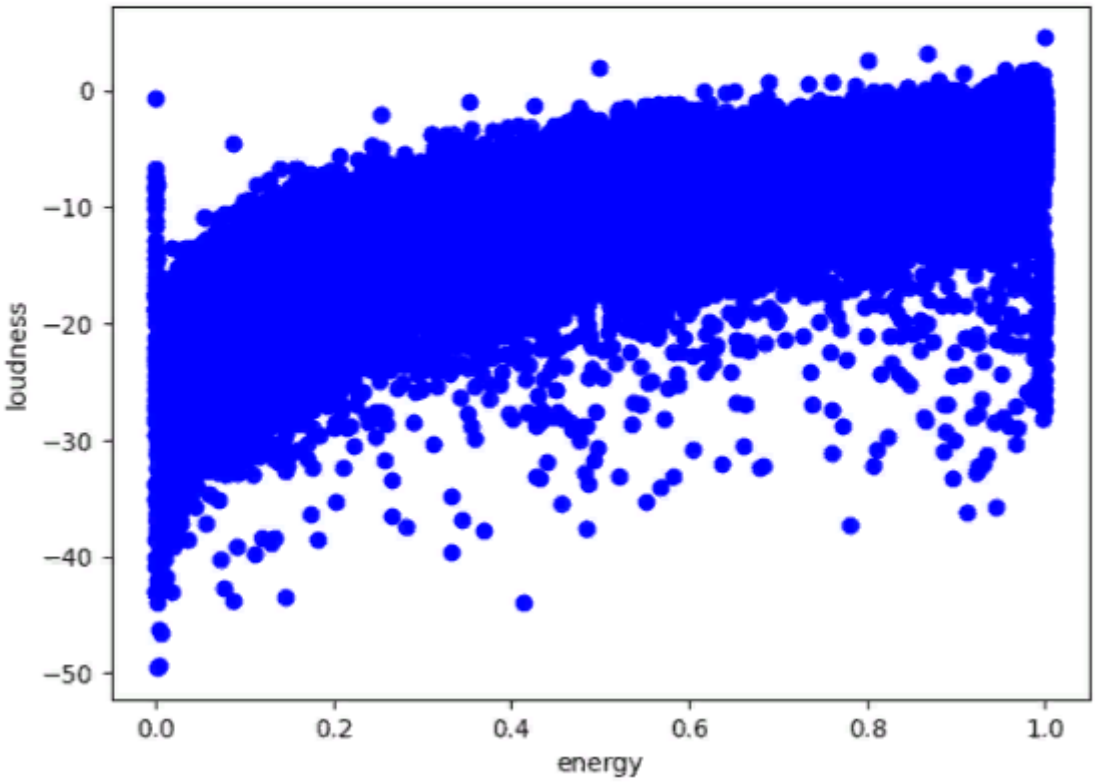


Рис.6.Диаграмма рассеивания танцевальности от энергии

Посмотрим как связаны акустичность и речь

```
fig,ax = plt.subplots(figsize=(7,5))
ax.scatter(ds['acousticness'],ds['speechiness'],color = 'blue')
ax.set_xlabel('acousticness')
ax.set_ylabel('speechiness')
plt.show()
```

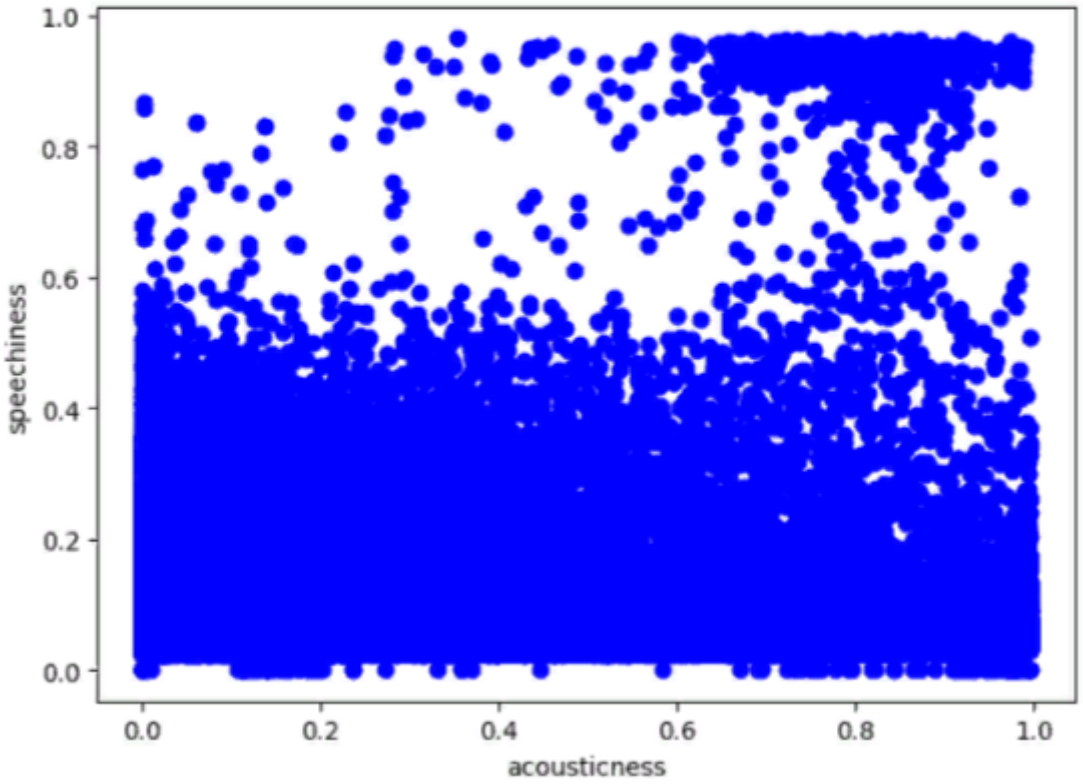


Рис.7.Диаграмма рассеивания речи от акустичности

Дальше были построены гистограммы

```
ds.hist(figsize = (20, 20), bins = 12, legend = False, grid = True);
```

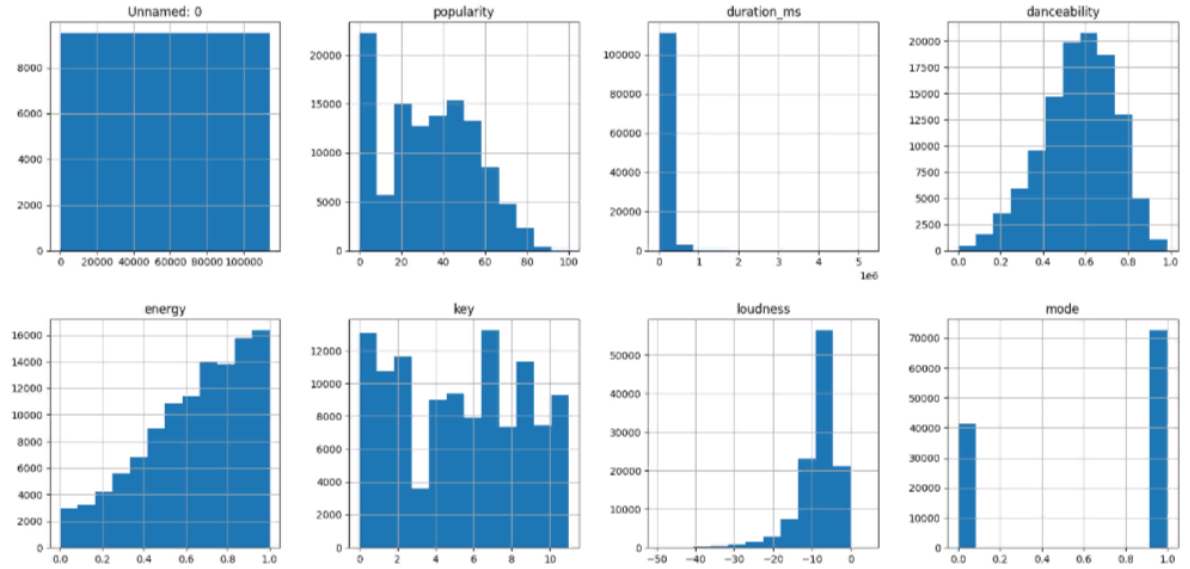


Рис.8.Гистограммы(1)

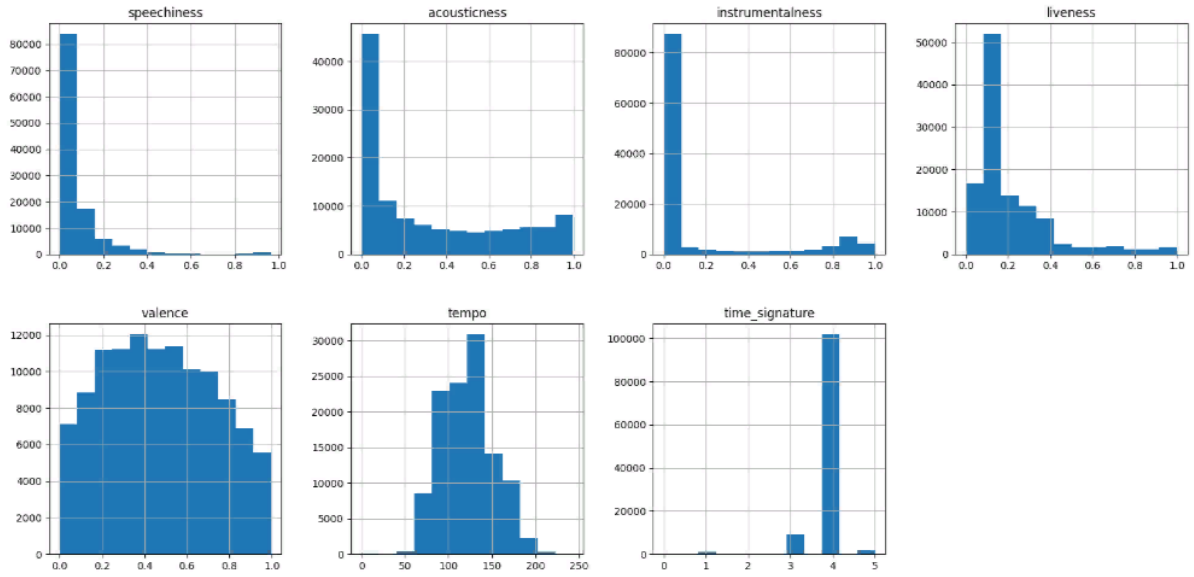


Рис.9.Гистограммы(2)

Дальше мною был построен ящик с усами параметра «danceability».

```
sns.boxplot(x=ds['danceability'],color = 'blue')
```

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

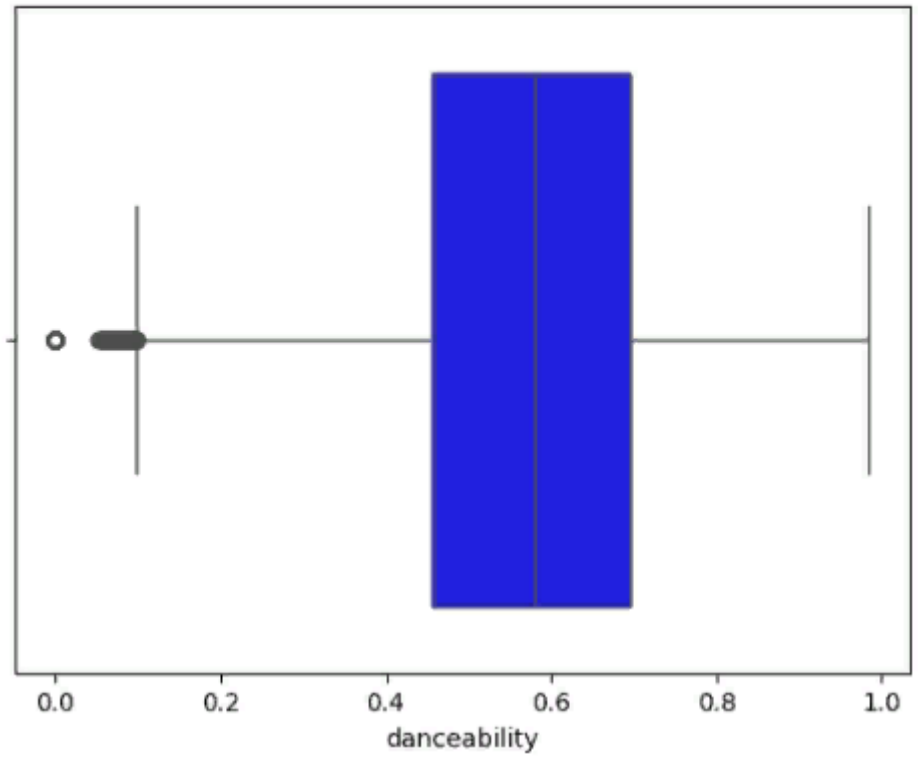


Рис.10.Ящик с усами параметра ‘danceability’

В нашем датасете много выбросов, которые мешают правильной работе методов. Они присутствуют во многих столбцах. Очистим их.

```
z = np.abs(stats.zscore(ds))
clean_ds = ds[(z<2.3).all(axis=1)]
```

Рис.11.Удаление выбросов

Затем была проведена так называемая «нормализация» датасета. На рисунке 12 виден положительный результат ее проведения.

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
```

```
#стандартизация
scaler = StandardScaler()
features = scaler.fit_transform(clean_ds)
standart_ds = pd.DataFrame(features,columns=['Unnamed: 0','popularity','duration_ms',
'danceability','energy','key','loudness','speechiness','acousticness',
'instrumentalness','liveness','valence','tempo','time_signature'
])
standart_ds.describe()
```

	Unnamed: 0	popularity	duration_ms	danceability	energy	key	loudness	speechiness	acousticness	instrumentalness	
count	8.951600e+04	8.951600e+04	8.951600e+04	8.951600e+04	8.951600e+04	8.951600e+04	8.951600e+04	8.951600e+04	8.951600e+04	8.951600e+04	8.951(
mean	6.096081e-17	9.271124e-17	2.540034e-17	-1.460519e-16	5.257870e-16	4.079929e-17	4.216456e-16	3.810051e-18	-6.731090e-17	-5.588075e-17	-1.028
std	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000(
min	-1.762109e+00	-1.484164e+00	-3.265350e+00	-2.698562e+00	-2.689775e+00	-1.496158e+00	-3.706330e+00	-8.381319e-01	-9.121967e-01	-3.918514e-01	-1.401
25%	-8.403985e-01	-7.378992e-01	-6.586624e-01	-6.683423e-01	-7.153250e-01	-9.346699e-01	-5.549857e-01	-6.154606e-01	-8.631536e-01	-3.918514e-01	-6.898
50%	1.351050e-03	9.616146e-02	-1.229860e-01	5.122908e-02	1.491375e-01	-9.243768e-02	1.976665e-01	-3.945289e-01	-4.544619e-01	-3.917696e-01	-4.214
75%	8.715727e-01	7.546304e-01	5.148532e-01	7.386768e-01	8.487283e-01	7.497946e-01	7.290372e-01	1.551909e-01	7.519961e-01	-3.650500e-01	5.360
max	1.736454e+00	2.203262e+00	3.740007e+00	2.421960e+00	1.485935e+00	1.592027e+00	2.672184e+00	4.465968e+00	2.344259e+00	3.668118e+00	3.850

Рис.12.Стандартизация данных

```
#нормализация
normal = MinMaxScaler()
features = normal.fit_transform(standart_ds)
norm_ds = pd.DataFrame(features,columns=[
    'Unnamed: 0','popularity','duration_ms','danceability','energy',
'key','loudness','speechiness','acousticness','instrumentalness',
'liveness','valence','tempo','time_signature'
])
#norm_ds = norm_ds.drop('hearing(left)',axis=1)
#norm_ds = norm_ds.drop('hearing(right)',axis=1)
norm_ds.describe()
```

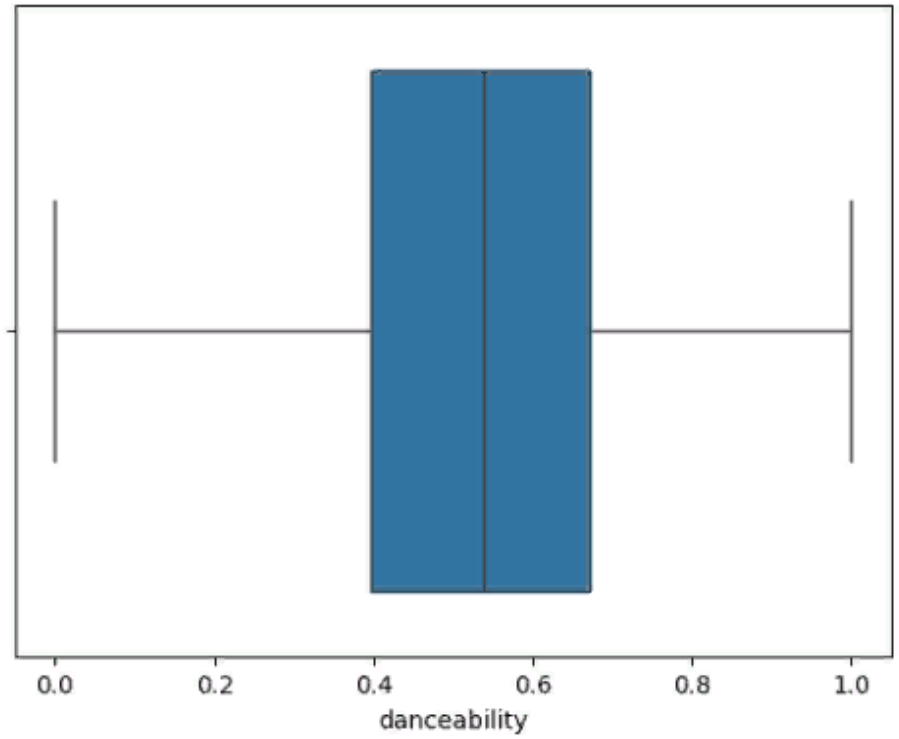
	Unnamed: 0	popularity	duration_ms	danceability	energy	key	loudness	speechiness	acousticness	instrumentalness	liveness
count	89516.000000	89516.000000	89516.000000	89516.000000	89516.000000	89516.000000	89516.000000	89516.000000	89516.000000	89516.000000	89516.000000
mean	0.503666	0.402493	0.466122	0.527009	0.644148	0.484478	0.581065	0.158016	0.280119	0.096516	0.266841
std	0.285833	0.271193	0.142749	0.195294	0.239482	0.323817	0.156777	0.188534	0.307084	0.246309	0.190403
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.263454	0.202381	0.372099	0.396487	0.472842	0.181818	0.494056	0.041981	0.015060	0.000000	0.135489
50%	0.504053	0.428571	0.448566	0.537014	0.679863	0.454545	0.612054	0.083634	0.140562	0.000020	0.186599
75%	0.752789	0.607143	0.539616	0.671267	0.847402	0.727273	0.695361	0.187275	0.511044	0.006601	0.368913
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Рис.13.Нормализация данных

Далее нас интересуют выбросы, проверяем их отсутствие:

```
sns.boxplot(x = norm_ds['danceability'])
```

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




```
sns.boxplot(x=norm_ds['energy'])
```

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

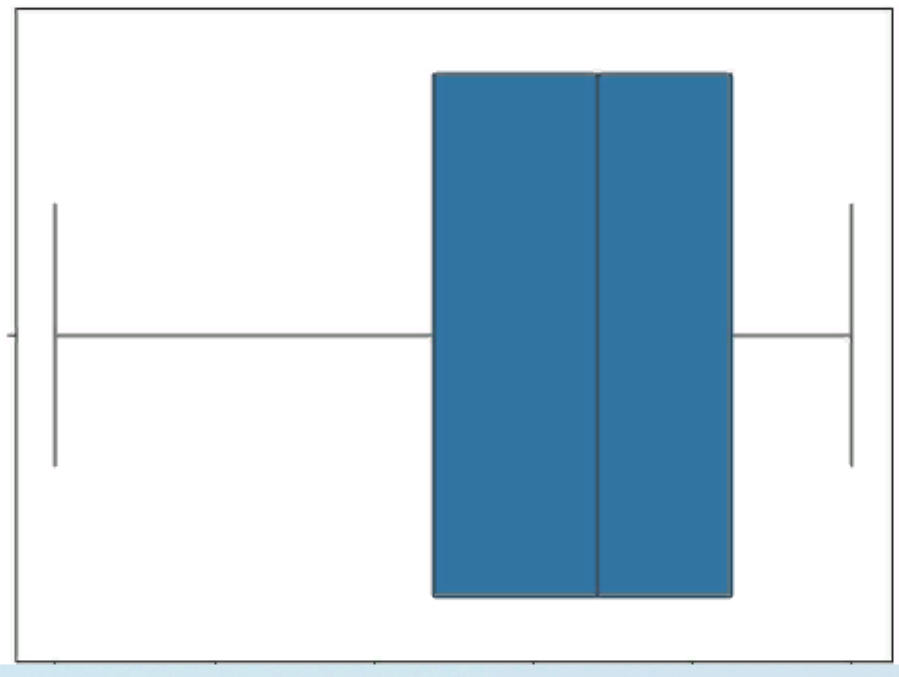


Рис.14.Проверка отсутствия выбросов

В конце построим матрицу корреляций, чтобы посмотреть насколько сильно величины соотносятся между собой.

```
ds_cor = norm_ds.assign(profit=target)
plt.figure(figsize = [22, 12], clear = True, facecolor = "white")
sns.heatmap(ds_cor.corr(), annot = True, square = False, linewidths = 3,
            linecolor = "#422e13", cmap = "terrain_r");
```

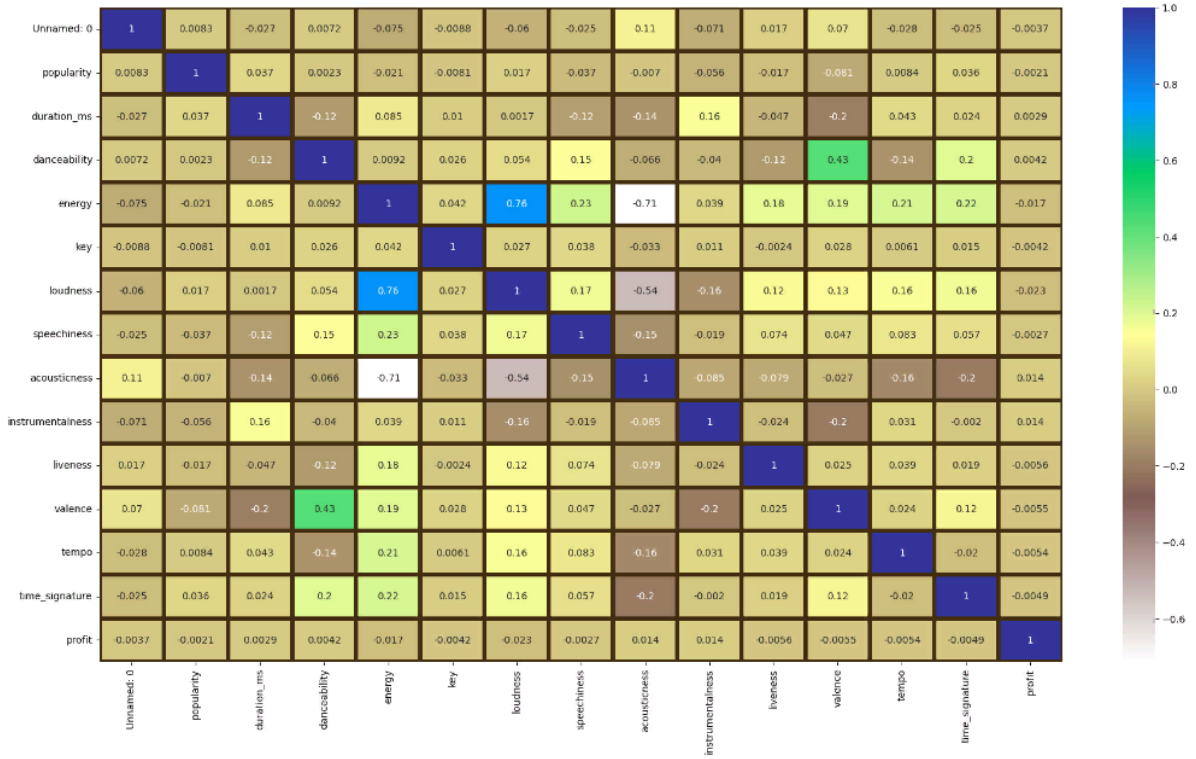


Рис.15.Матрица корреляций

Программа:

Программа находится в файле lab_1_mo.ipynb

Вывод:

Я получил и закрепил навыки предобработки данных для дальнейшего применения методов машинного обучения при решении задач, научилась визуализировать данные с помощью разных средств, таких как гистограммы, ящики с усами, диаграммы рассеивания, научилась обрабатывать данные (удалять дубликаты и пропуски, а также нормализировать их). Помимо этого изучил матрицу корреляций.

Цель работы

Получение и закрепления навыков предобработки данных и применения методов машинного обучения для решения задач классификации.

Задание

Набор данных берется из лабораторной работы №1.

В ходе выполнения лабораторной работы должны быть выполнены следующие этапы:

1. Обучение моделей и подбор параметров:
    a. K-ближайших соседей (KNN)
    b. Машина опорных векторов (SVM)
    c. Дерево решений ИЛИ Случайный лес
2. Оценка моделей
    a. Визуализация предсказанных значений
    b. Оценка качества прогноза (precision/recall/f1-score/ROC-AUC)
    c. Визуализация дерева решений ИЛИ Визуализация Feature Importance для случайного

Выполнение работы.

1. Подготовка данных для тренировки. Подготовку данных мы уже сделали в предыдущей работе. Для текущей работы нам потребуется определить target-переменную. Я выбрал такой переменной столбец «mode». Это дискретная величина.

2. Разобьем данные на тренировочные и проверочные.

Применяем функцию train_test_split() из библиотеки sklearn (рис.1).

```
X_train, X_test, Y_train, Y_test = train_test_split(norm_ds, target, test_size=0.2, random_state=35)
```

Рис.1.train_test_split()

3. k Nearest Neighbors (kNN).

Данный алгоритм классификации находит k ближайших соседей к выбранным данным. На основе своих k соседей, объект получает свой класс. То есть если все похожие объекты были класса 2, то и наш объект будет иметь 2-ой класс. Здесь нам необходимо для начала подобрать гиперпараметр k = количество ближайших соседей, вектора которых будут анализироваться. Мы подберем данный параметр перебором. Заметим, что k mod2 = 1, так как алгоритму нужно выбрать одну из ближайших точек(чтобы не было неопределенности). Как же нам выбрать, какой из параметров лучше? Все просто! Необходимо посмотреть на некоторые метрики работы алгоритма, и выяснить,какой из отчетов по обучению моделей был лучшим. В нашем случае, лучше всего модель обучилась при k = 3. Данный алгоритм, как видно из отчета, имеет точность 84%. (рис.2)

```
print(classification_report(y_pred,Y_test))
```

	precision	recall	f1-score	support
0	0.74	0.81	0.77	1696
1	0.90	0.85	0.87	3323
accuracy			0.84	5019
macro avg	0.82	0.83	0.82	5019
weighted avg	0.84	0.84	0.84	5019

Рис.2. kNN алгоритм

ROC-кривая представлена на рисунке 3.

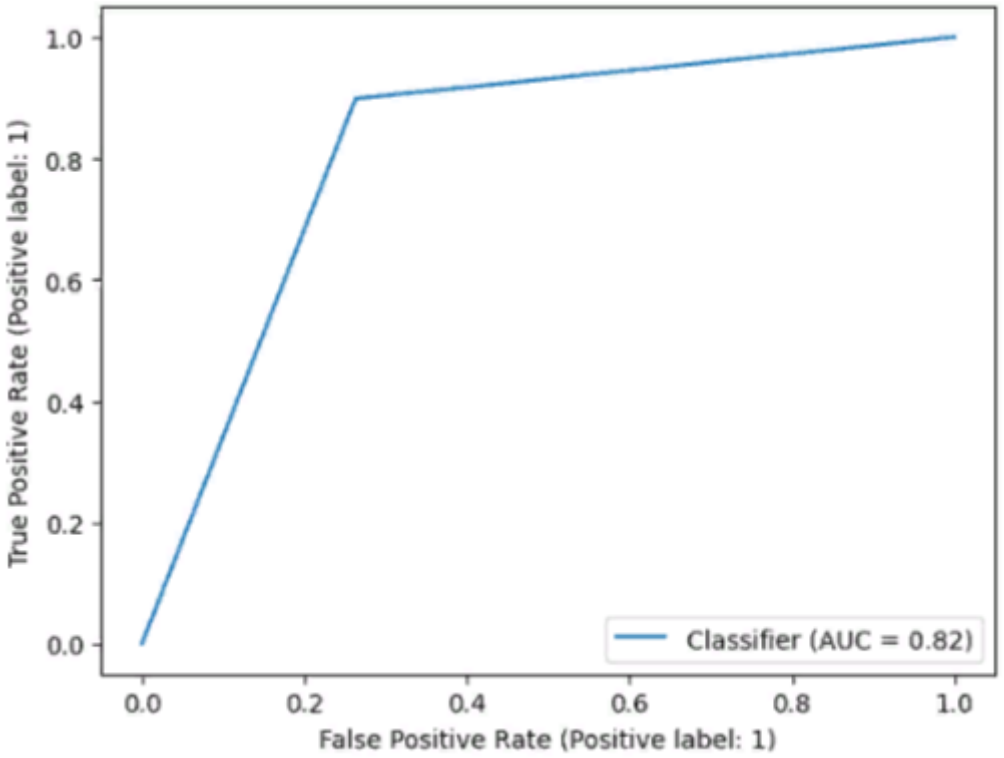


Рис.3.ROC-AUC

4. Метод опорных векторов SVM (SVC).

Данный метод работает по принципу логистической регрессии. То есть мы хотим найти разделяющую гиперплоскость для наших данных. Важно, что этот алгоритм требует дополнительной обработки датасета при несбалансированных классах. Наш случай как раз такой. Для этого существует достаточно методов, которые позволяют дополнить датасет данными меньшего класса. Самое простое — это просто удалить часть данных, относящихся к большему классу, но это уменьшает объем доступной информации, что может привести к более низким результатам. Можно также добавить дубликаты, которые не внесут новой информации, но хотя бы уравновесят классы.

```
from sklearn.svm import SVC
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix ,classification_report
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
model2 = SVC(kernel = "poly", verbose=True, cache_size = 8192)
X_train_scaled = scaler.fit_transform(X_train)
```

```
%%time
model2.fit(X_train_scaled, Y_train.ravel())
```

```
[LibSVM]CPU times: total: 1min 42s
Wall time: 1min 54s
```

```
SVC
SVC(cache_size=8192, kernel='poly', verbose=True)
```

```
%%time
y_pred = model2.predict(scaler.transform(X_test))
```

```
CPU times: total: 4.58 s
Wall time: 4.65 s
```

Рис.4.Обучение моделей

```
print(classification_report(y_pred,Y_test))
```

	precision	recall	f1-score	support
0	0.16	0.61	0.25	474
1	0.94	0.65	0.77	4545
accuracy			0.65	5019
macro avg	0.55	0.63	0.51	5019
weighted avg	0.87	0.65	0.72	5019

Рис.5. Вывод метрик

Заметно, что плохо определяется класс 1, то есть алгоритм плохо его отличает. Это происходит как раз потому, что классы в дисбалансе.

Посмотрим матрицы ошибок:

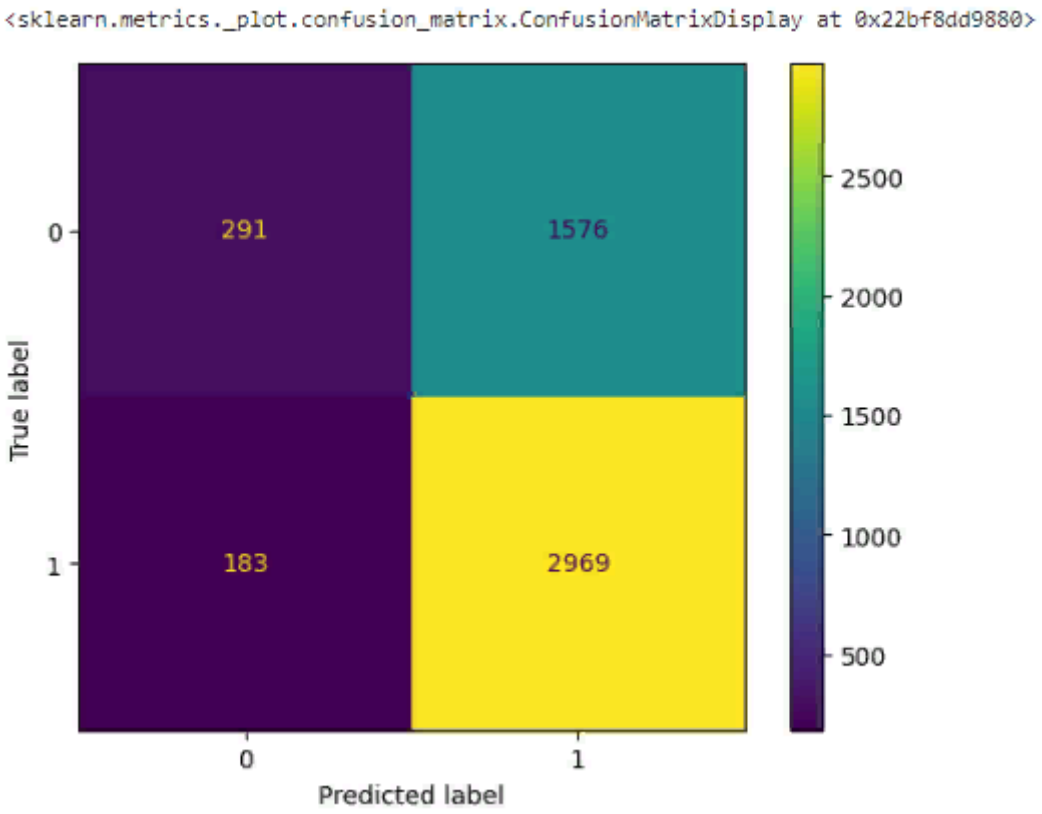


Рис.6.Матрицы ошибок

Прослеживается та же проблема.

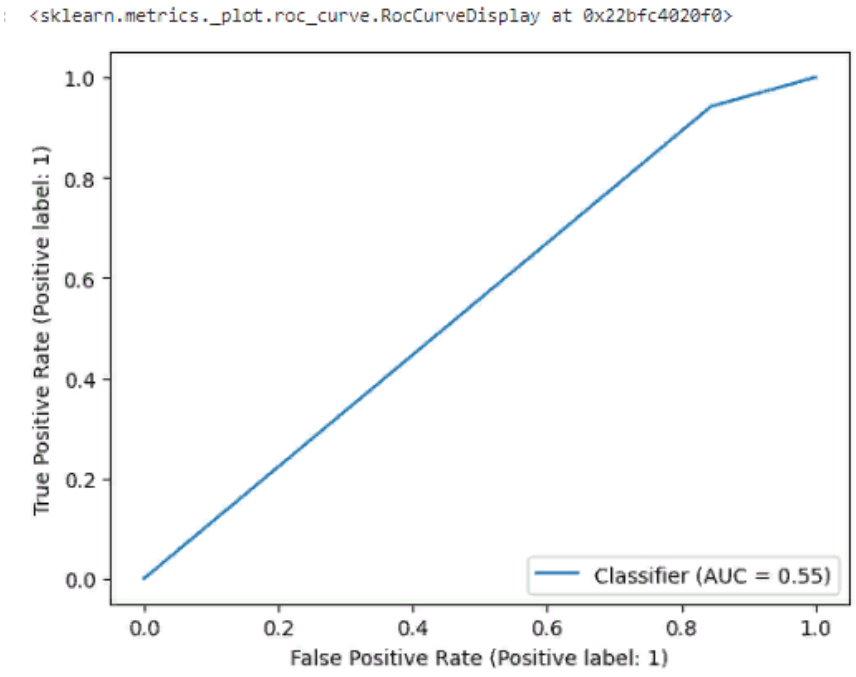


Рис.7.Кривая ROC-AUC

5. Дерево решений (Decision Tree)

Работает по принципу деления признаков на несколько частей. То есть берем, например, feature_1 и делаем, что при feature_1 <1 000 дальше пойдем в левый лист, при feature_1> =1 000 в правый. И так далее. В конечном итоге получим большое дерево, которое может предсказать класс по признакам.Сделаем классификатор, обучим, найдем accuracy и нарисуем дерево(рис.9-10). Точность составила 70%

.

```
Y_tree=target
X_tree=norm_ds
X_train_tree, X_test_tree, Y_train_tree, Y_test_tree = train_test_split(X_tree, Y_tree, test_size=0.2, random_state=35)

from sklearn.tree import DecisionTreeClassifier
model3 = DecisionTreeClassifier(max_features = 5)
```

```
model3.fit(X_train_tree, Y_train_tree)
```

```
DecisionTreeClassifier(max_features=5)
```

```
Y_pred_tree = model3.predict(X_test_tree)
```

```
print(classification_report(Y_pred_tree,Y_test_tree))
```

```
              precision    recall  f1-score   support

           0       0.59      0.59      0.59      1863
           1       0.76      0.76      0.76      3156

    accuracy                           0.70      5019
   macro avg       0.67      0.67      0.67      5019
weighted avg       0.70      0.70      0.70      5019
```

Рис.8.Tree

<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x22b85207080>

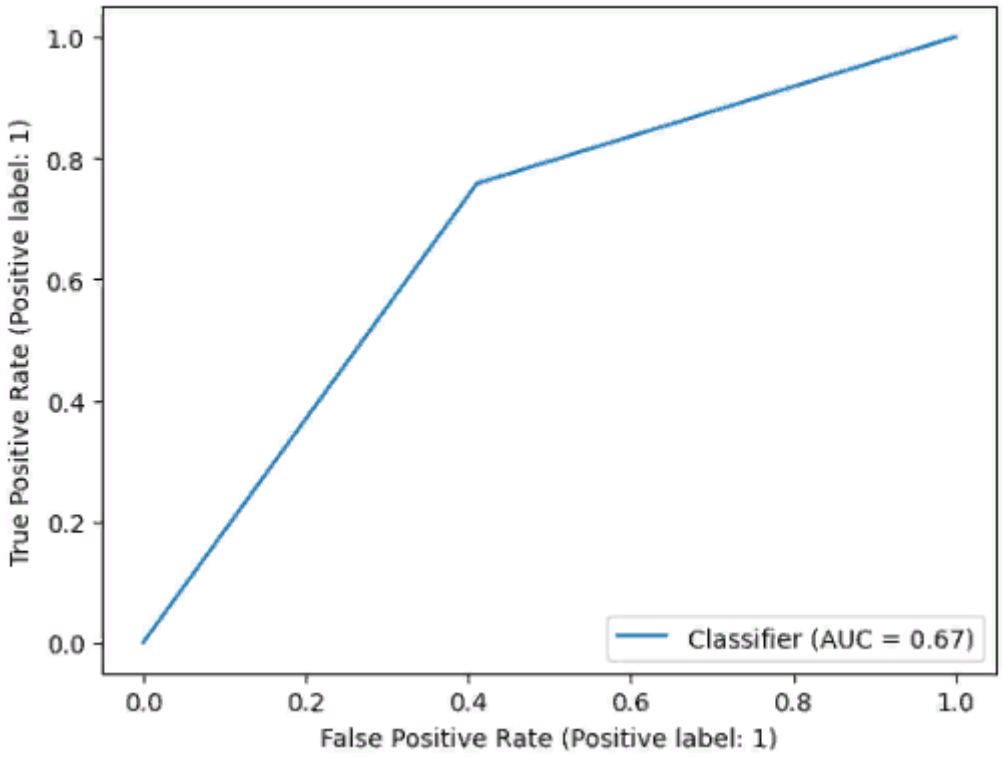


Рис.9.ROC-AUC

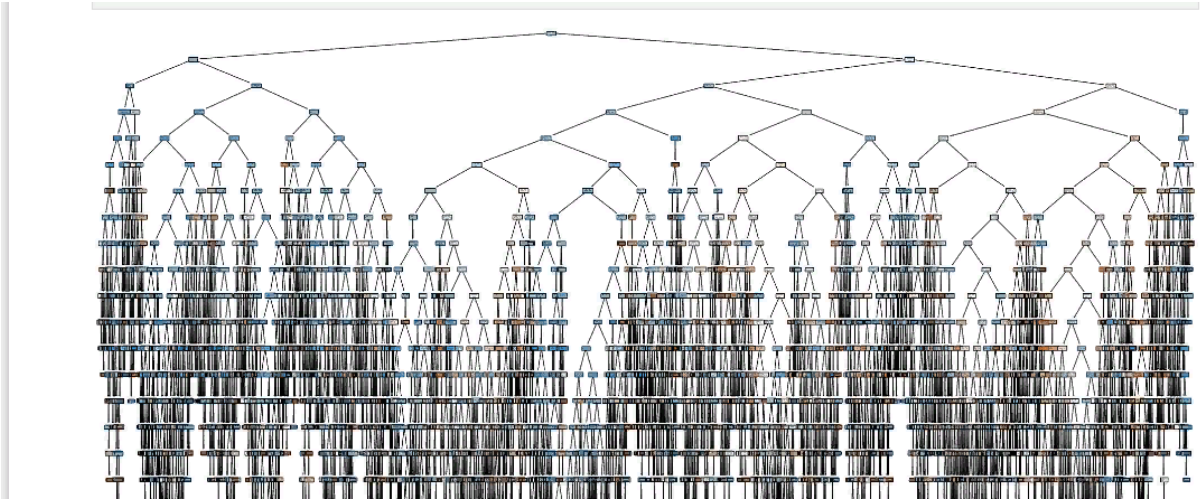


Рис 10. Рисунок дерева решений.

Вывод:

В ходе выполнения данной работы были приобретены и закреплены навыки обучения моделей методами: K-ближайших соседей, SVM, Tree. Наиболее худшей моделью оказалась модель, обученная методом SVM. Наиболее лучшей оказалась модель KNN.

Цель работы.

Служит для получения и закрепления навыков предобработки данных и применения методов машинного обучения для решения задач кластеризации.

Задание.

Набор данных берется из лабораторной работы No1. В ходе выполнения лабораторной работы должны быть выполнены следующие этапы:

1. Обучение моделей и подбор параметров (где применимо):
    a. метод К-средних
    b. DBSCAN
    c. Иерархическая кластеризация
2. Оценка моделей
    a. Экспертная оценка
    b. Сравнение разбиения на классы с помощью кластеризации с реальными.
    c. Визуализация предсказанных значений

Выполнение работы.

Подготовка данных для тренировки.

Прежде чем начать работу, нам потребуется стандартизировать и нормализировать наши данные

```
#стандартизация
scaler = StandardScaler()
features = scaler.fit_transform(clean_ds)
standart_ds = pd.DataFrame(features,columns=['Unnamed: 0','popularity','duration_ms',
'danceability','energy','key','loudness','speechiness','acousticness',
'instrumentalness','liveness','valence','tempo','time_signature'
])
standart_ds.describe()
```

```
#нормализация
normal = MinMaxScaler()
features = normal.fit_transform(standart_ds)
norm_ds = pd.DataFrame(features,columns=[
    'Unnamed: 0','popularity','duration_ms','danceability','energy',
'key','loudness','speechiness','acousticness','instrumentalness',
'liveness','valence','tempo','time_signature'
])
norm_ds.describe()
```

	Unnamed: 0	popularity	duration_ms	danceability	energy	key	loudness	speechiness	acousticness	instrumentalness	liveness
count	25092.000000	25092.000000	25092.000000	25092.000000	25092.000000	25092.000000	25092.000000	25092.000000	25092.000000	25092.000000	25092.000000
mean	0.502220	0.393870	0.400792	0.526565	0.642369	0.486421	0.580098	0.119512	0.268558	0.192728	0.253890
std	0.294821	0.268337	0.158060	0.202328	0.244964	0.325750	0.168154	0.158009	0.315558	0.329868	0.183186
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.251625	0.185185	0.300064	0.390875	0.466598	0.181818	0.493650	0.030037	0.008351	0.000000	0.131613
50%	0.487816	0.382716	0.377343	0.538841	0.682030	0.545455	0.614206	0.059381	0.109438	0.000283	0.177621
75%	0.767767	0.617284	0.473354	0.679408	0.850854	0.818182	0.699991	0.130545	0.495984	0.236181	0.337929
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Рис.1.Стандартизация и нормализация

PCA-model

Мною было решено применить алгоритм PCA для того, чтобы уменьшить количество параметров. Был составлен график зависимости совокупной накопленной дисперсии от количества параметров. Для точности дальнейшей кластеризации было выбрано число параметров, у которого значение совокупной накопленной дисперсии больше 0.95. n_components = 11.

```
for i, exp_var in enumerate(evr.cumsum()):
    if exp_var >=0.95:
        n_comps = i + 1
        break
print("Number of components:", n_comps)
```

```
Number of components: 11
```

```
pca = PCA(n_components=n_comps)
ds_PCA = pca.fit_transform(norm_ds)
ds_PCA = pd.DataFrame(ds_PCA, index=norm_ds.index,
                      columns=['PC1','PC2','PC3','PC4','PC5','PC6','PC7','PC8','PC9','PC10','PC11'])#,'PC12'
ds_PCA
```

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11
0	0.105893	-0.455367	-0.350907	0.153662	-0.424554	-0.443529	0.193928	-0.203278	0.283461	-0.001758	0.297242
1	1.122462	-0.157221	-0.294893	-0.005958	-0.250449	-0.242174	-0.238728	-0.151426	0.048398	0.066104	-0.053547
2	0.447202	-0.103174	-0.464633	-0.110879	-0.538609	-0.226921	-0.210174	-0.395502	-0.025664	-0.005516	0.059939
3	1.337552	0.078779	-0.427951	-0.391376	-0.547152	0.328350	0.583906	0.274046	-0.275767	-0.015332	0.242809
4	0.257210	-0.414075	0.123095	0.136796	-0.303726	-0.327172	0.084202	-0.144730	0.051283	0.040830	0.122277
...	...	...	...	...	...	...	...	...	...	...	...
25087	-0.558279	0.499764	-0.471453	-0.056744	0.207321	-0.151879	0.187944	0.166240	0.040428	0.019801	-0.178080
25088	-0.276043	-0.081098	0.279864	-0.444282	0.340482	0.049201	-0.089718	-0.079276	-0.000730	-0.082575	-0.191629
25089	-0.461866	0.724804	0.007556	-0.104725	0.282639	-0.285236	0.131616	-0.068791	-0.115837	-0.085685	0.013224
25090	-0.648854	0.311731	0.395287	-0.276434	0.320604	0.054245	0.029116	0.202108	0.267544	0.280210	0.008405
25091	-0.036576	0.071531	0.040712	-0.407005	0.449318	0.145320	-0.322118	-0.459527	0.034665	0.435901	0.082528

25092 rows × 11 columns

Рис.2.Выбор n_components

K-Means

Прежде, чем применить алгоритм средних, я решила использовать метод локтя для определения оптимального количества кластеров, на которые можно разделить наши данные.

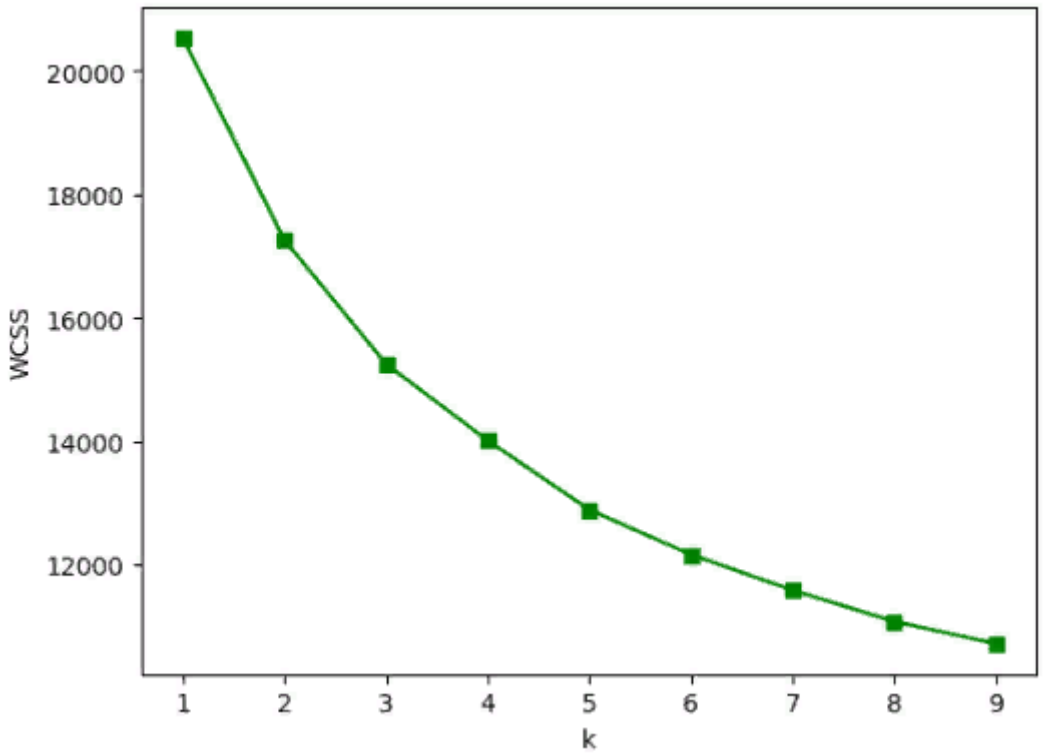


Рис.3.Метод локтя

В моменте, где график преломляется (как сгиб локтя) – оптимальное k кластеров.K = 3.

Дальше применим k-means.

```
k = 3
KM = KMeans(init='random', n_clusters = k, n_init = 50)
CLUSTERS = KM.fit_predict(ds_PCA.values)
```

Рис.4.K-means

Результаты представлены на рисунке 5.

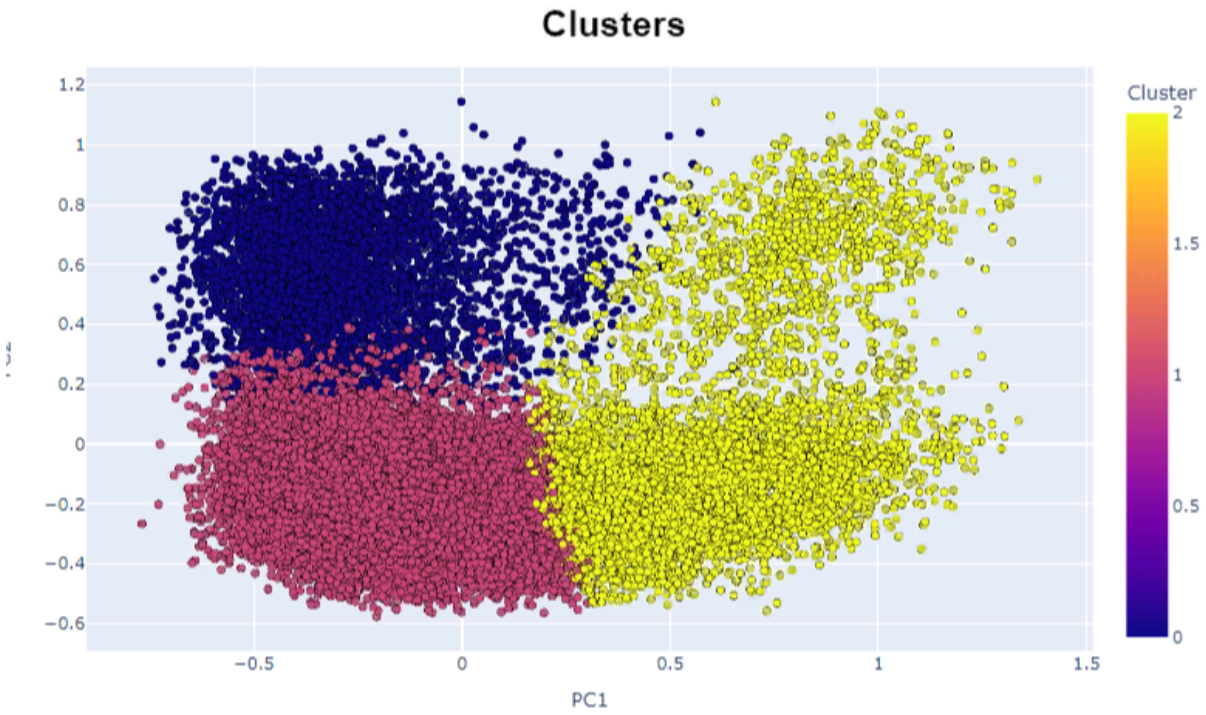


Рис.5.Результаты K-means

DBSCAN

Данный метод основан на плотности. Если дан набор точек в некотором пространстве, алгоритм группирует вместе точки, которые тесно расположены (точки со многими близкими соседями), помечая как выбросы точки, которые находятся одиноко в областях с малой плотностью (ближайшие соседи которых лежат далеко). Данный метод использует две важнейшие характеристики, такие как min_samples- количество выборок в окрестности точки, чтобы рассматривать ее как главную точку и eps – максимальное расстояние между двумя точками, чтобы рассматривать одну, как соседа второй. Для корректной работы алгоритма min_samples >= 2*D, где D – размерность данных. То есть в нашем случае, D = 11.

Для того, чтобы корректно определить эти два параметра был применен метод ближайших соседей (метод k-ближайших соседей). Это алгоритм для автоматической классификации объектов или регрессии. В случае использования метода для классификации объект присваивается тому классу, который является наиболее распространённым среди k соседей данного элемента, классы которых уже известны.

Основной параметр – n_neighbors, который по правилам равен min_samples -1. Благодаря данному алгоритму мы сможем найти eps параметр для нашего алгоритма DBSCAN. Результат представлен на рисунке 7.

```
arr = ds_PCA.values
kNN = NearestNeighbors(n_neighbors = 21)
new_model = kNN.fit(arr)
dist, indices = kNN.kneighbors(arr)
dist = np.sort(dist, axis=0)
dist=dist[:,1]
plt.grid()
plt.plot(dist)
```

```
[<matplotlib.lines.Line2D at 0x1da15df48f0>]
```

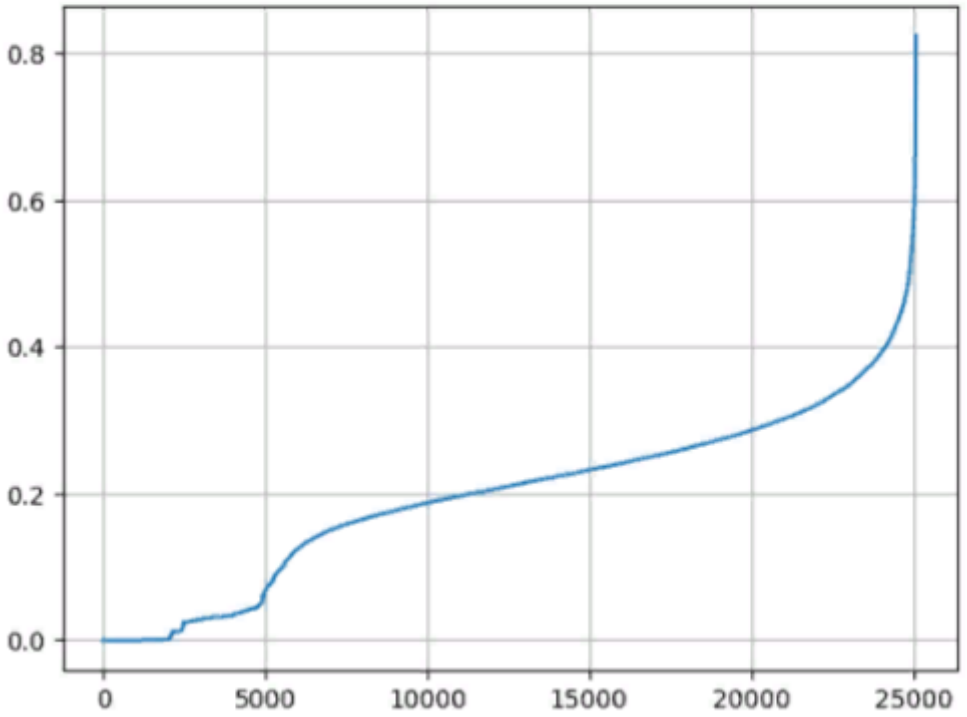


Рис.6.Метод K ближайших соседей

Параметр следует брать из области, где начинается «сгиб». Выберем eps = 0.6 и применим наш DBSCAN(рис.7).

```
db = DBSCAN(eps = 0.8, min_samples = 22).fit(arr)
csm = np.zeros_like(db.labels_,dtype=bool)
csm[db.core_sample_indices_]=True
labels = db.labels_
```

```
n_clust = len(set(labels)) - (1 if -1 in labels else 0)
n_noise = list(labels).count(-1)
print('Number of Clusters: ', n_clust)
print('Number of Outliers: ', n_noise)
```

```
Number of Clusters:  6
Number of Outliers:  3
```

Рис.7.DBSCAN

Наш алгоритм разделил данные на два кластера с 3 выбросами, то есть точками,которые не попали ни в один кластер.

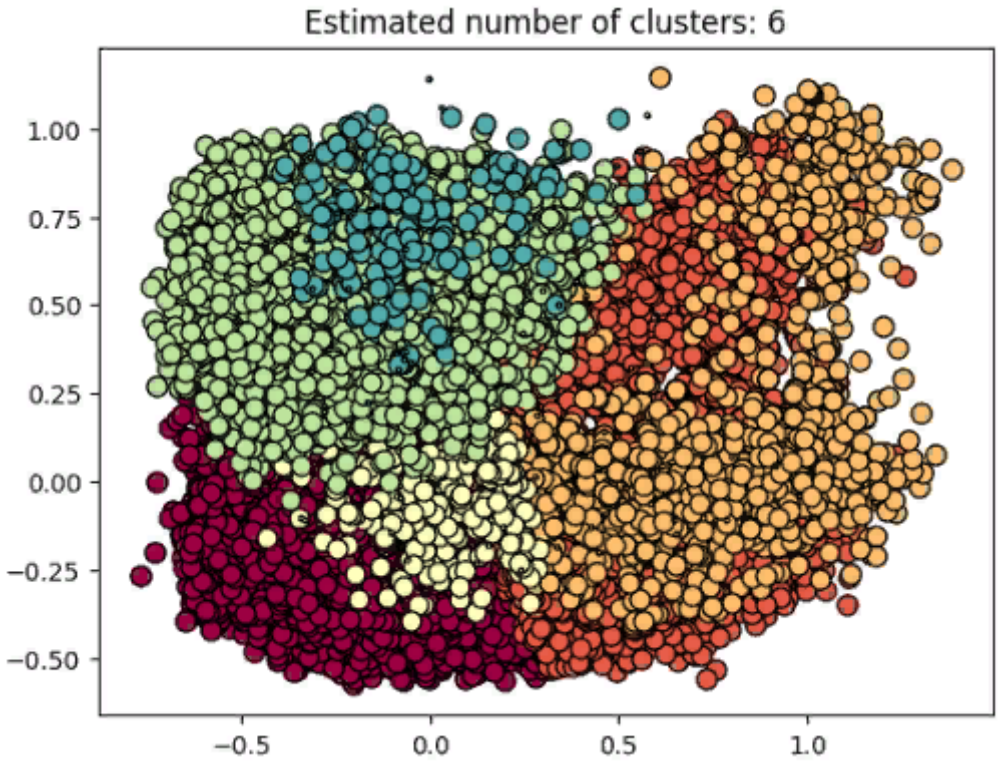


Рис 8. Визуализация DBSCAN.

Hierarchy.

Иерархическая модель данных — это модель данных, где используется представление базы данных в виде древовидной структуры, состоящей из объектов различных уровней. Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Для начала мною была построена дендрограмма (рис.9)

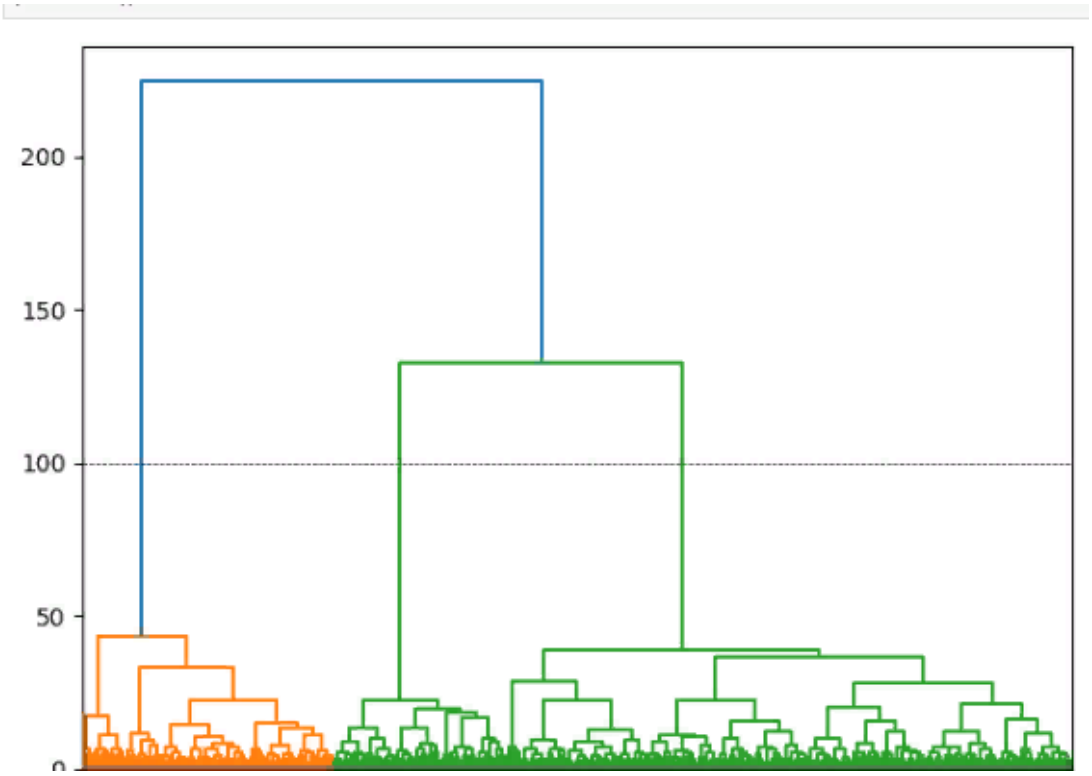


Рис.10. Дендрограмма

По ней можно заметить, что 3 ветви на верхнем уровне объединяются в одну, соответственно, выбранное количество кластеров равно 3-ем. Применим наш алгоритм иерархической кластеризации и посмотрим на результаты 11.

```
from scipy.cluster.hierarchy import fcluster
k = 3
cluster_labels = fcluster(Z, k, criterion='maxclust')
ds_PCA['Cluster_dendro'] = cluster_labels
```

```
ds_PCA['Cluster_dendro'].describe()
```

```
count    25092.000000
mean         2.318030
std          0.851356
min          1.000000
25%          1.000000
50%          3.000000
75%          3.000000
max          3.000000
Name: Cluster_dendro, dtype: float64
```

Рис.11.Применение иерархической кластеризации

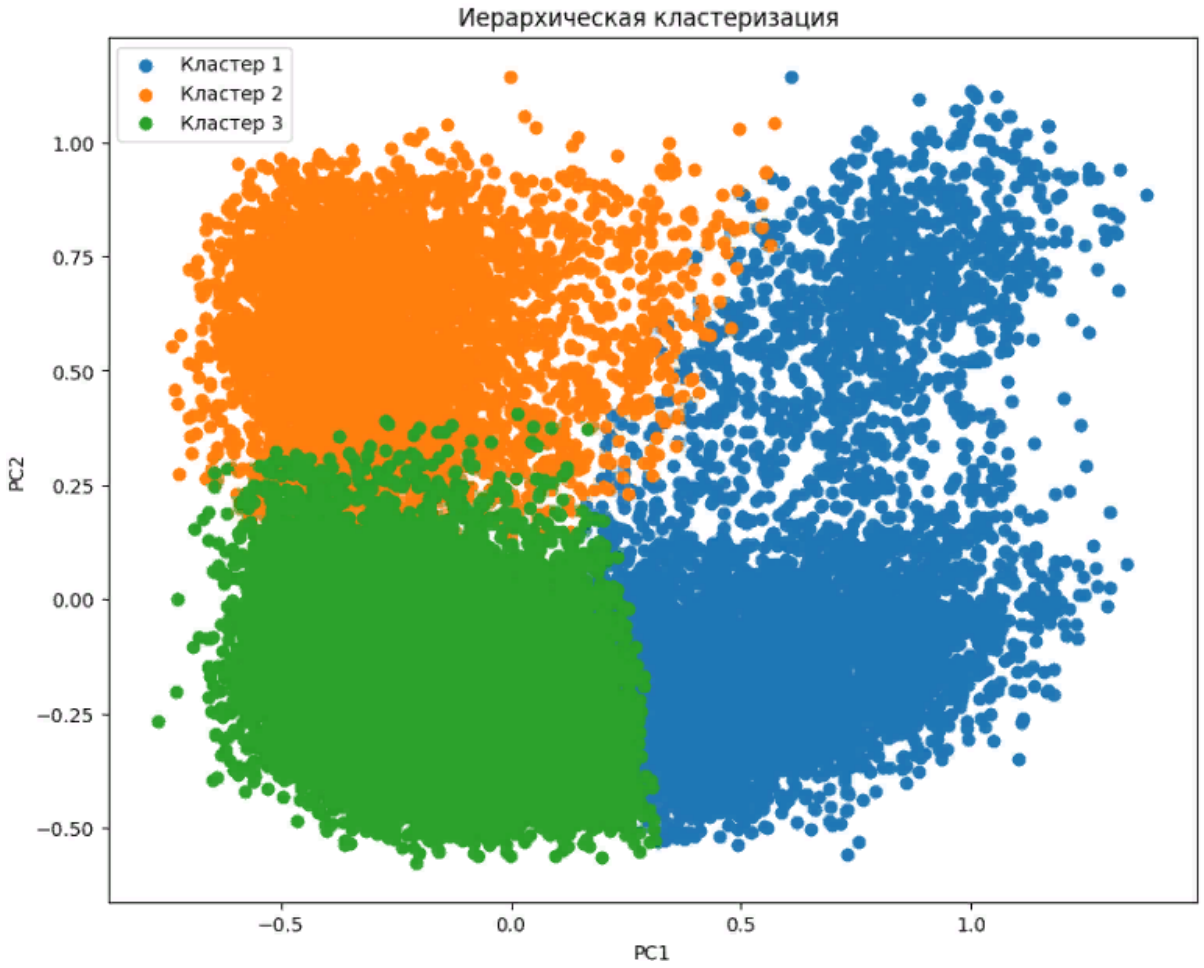


Рис.12. Результаты иерархической кластеризации.

Оценка производительности кластеризации

Оценка производительности алгоритма не так тривиальна, как подсчет количества ошибок или точности и полноты контролируемого алгоритма классификации. Если основные истинные метки неизвестны, то оценка должна выполняться с помощью самой модели. Так как в нашей работе мы не знаем эти самые метки, то можем ограничиваться лишь несколькими методами оценки кластеризации, а именно «силуэтная» метка, критерий отношения дисперсии и индекс Дэвиса-Болдина. Первые два критерия оцениваются по принципу: чем больше, тем лучше алгоритм, последний же индекс наоборот: более низкий индекс Дэвиса-Болдина относится к модели с лучшим разделением между кластерами. Результаты этих метрик представлены на рисунке 13.

```
-----------------------------------------K-means-----------------------------------------------
Silhouette Coefficient: 0.380
Calinski_Harabasz_Score: 16477.045
Davies_Bouldin_Score: 1.156
-----------------------------------------DBSCAN------------------------------------------------
Silhouette Coefficient: 0.303
Calinski_Harabasz_Score: 6572.975
Davies_Bouldin_Score: 1.520
-----------------------------------------Hierarchy---------------------------------------------
Silhouette Coefficient: 0.380
Calinski_Harabasz_Score: 16477.045
Davies_Bouldin_Score: 1.156
```

Рис.13.Результаты метрик

Вывод:

Я получил и закрепил навыки предобработки данных и применения методов машинного обучения для решения задач кластеризации.

Выполнение работы.

Подготовка данных для тренировки.

Загружаем данные и обрабатываем их

```
ds = pd.read_csv("train.csv")
```

```
notUse = ['track_id','artists', 'album_name',   'track_name','explicit','track_genre']
ds.drop(notUse, axis=1, inplace=True)
```

```
ds = ds[:30000]
ds
```

	Unnamed: 0	track_id	artists	album_name	track_name	popularity	duration_ms	explicit	danceability	energy	...	loudness	mode
0	0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	73	230666	False	0.676	0.4610	...	-6.746	0
1	1	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	55	149610	False	0.420	0.1660	...	-17.235	1
2	2	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	57	210826	False	0.438	0.3590	...	-9.734	1
3	3	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Sou...	Can't Help Falling In Love	71	201933	False	0.266	0.0596	...	-18.515	1
4	4	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	82	198853	False	0.618	0.4430	...	-9.681	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
29995	29995	7beAh9FEDhV8VUQ39fwDyf	Ray Volpe;Soltan	Elbow Grease	Elbow Grease	39	213600	False	0.594	0.9920	...	-1.800	1
29996	29996	3pnCeZNsoRXi2b5DbsOpqV	Jason Ross;RUNN	Atlas	More To Life (feat. RUNN)	36	224181	False	0.551	0.8120	...	-5.186	1
29997	29997	2p4Y8c7O0Tm48Zk6IAWcTo	SLANDER	Thrive	Second Life	41	376615	False	0.706	0.8570	...	-7.419	0
29998	29998	5gQJ9ccczcIMrKfPnfxpkD	Dirt Monkey	Primatology	High Grade	22	115201	False	0.709	0.9750	...	-1.587	0
29999	29999	5ruT2dWSHtML4esT7doTzt	Borgore	Ruined Dubstep - EP, Pt. 1	Cry Me a River	21	278851	True	0.536	0.4160	...	-10.004	0

Рис.1.Загрузка и обработка датасета

Теперь мы построим несколько диаграмм рассеивания, гистограмм и ящик с усами для параметра «danceability».

```
fig,ax = plt.subplots(figsize=(7,5))
ax.scatter(ds['danceability'],ds['acousticness'],color = 'blue')
ax.set_xlabel('danceability')
ax.set_ylabel('acousticness')
plt.show()
```

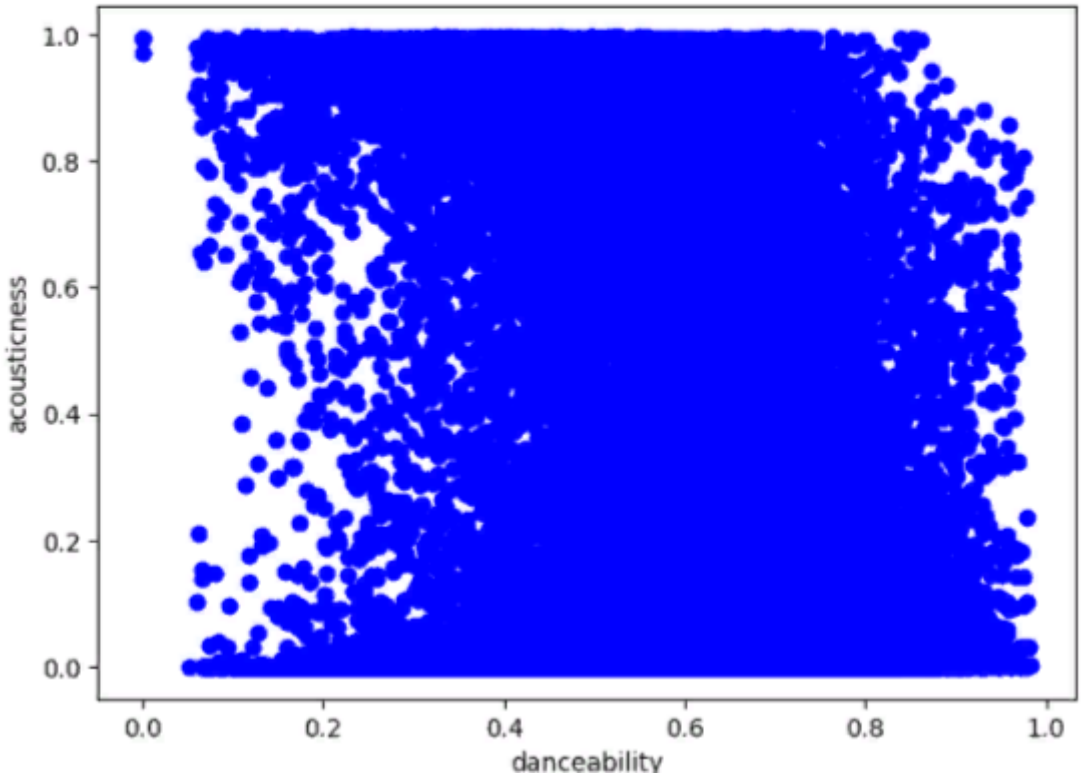


Рис.9.Диаграмма рассеивания accousticness от danceability

```
clean_ds.hist(figsize = (20, 20), bins = 12, legend = False, grid = True);
```

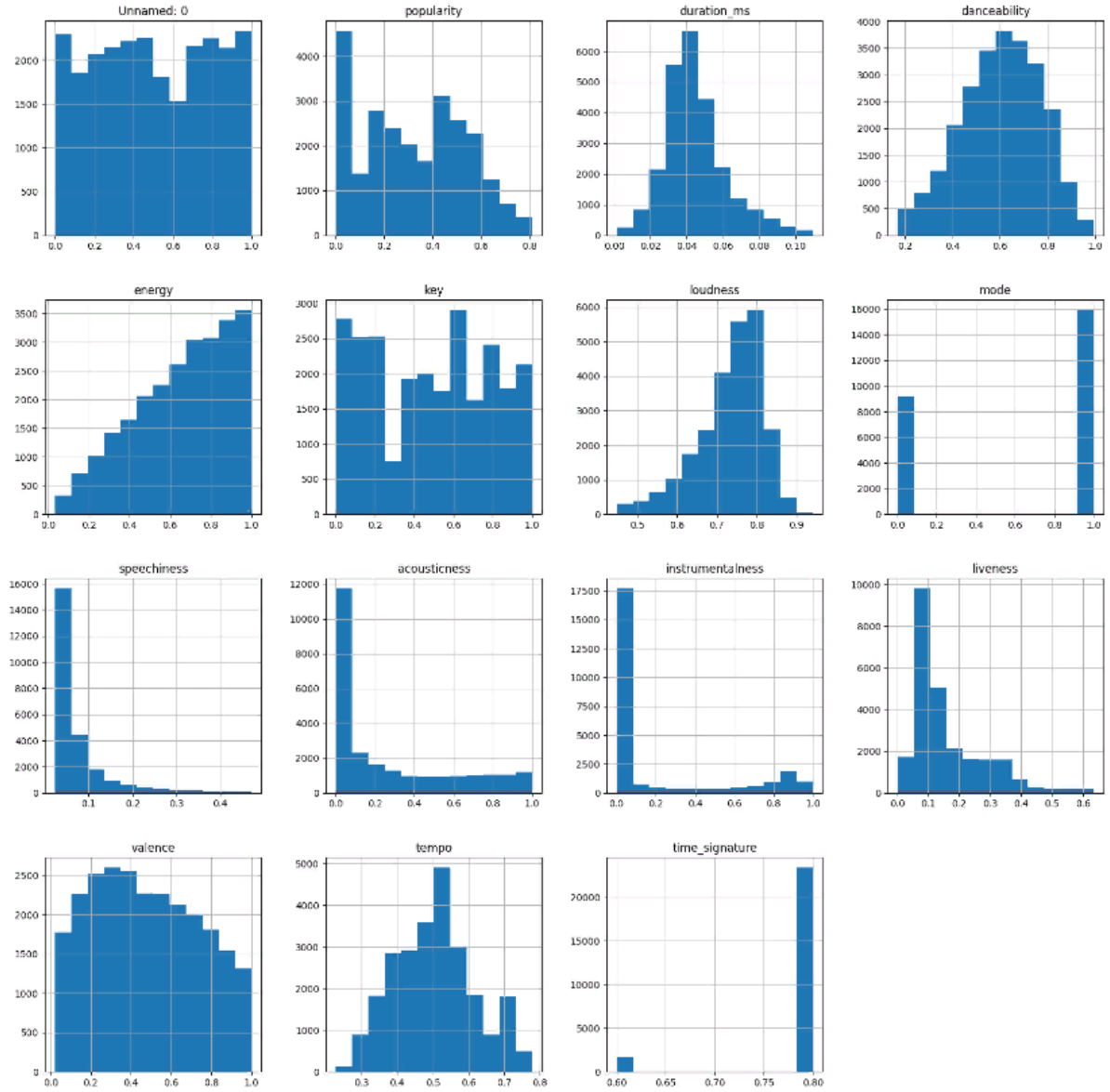


Рис.3.Гистограммы данных

```
sns.boxplot(x=clean_ds['danceability'],color = 'blue')
```

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

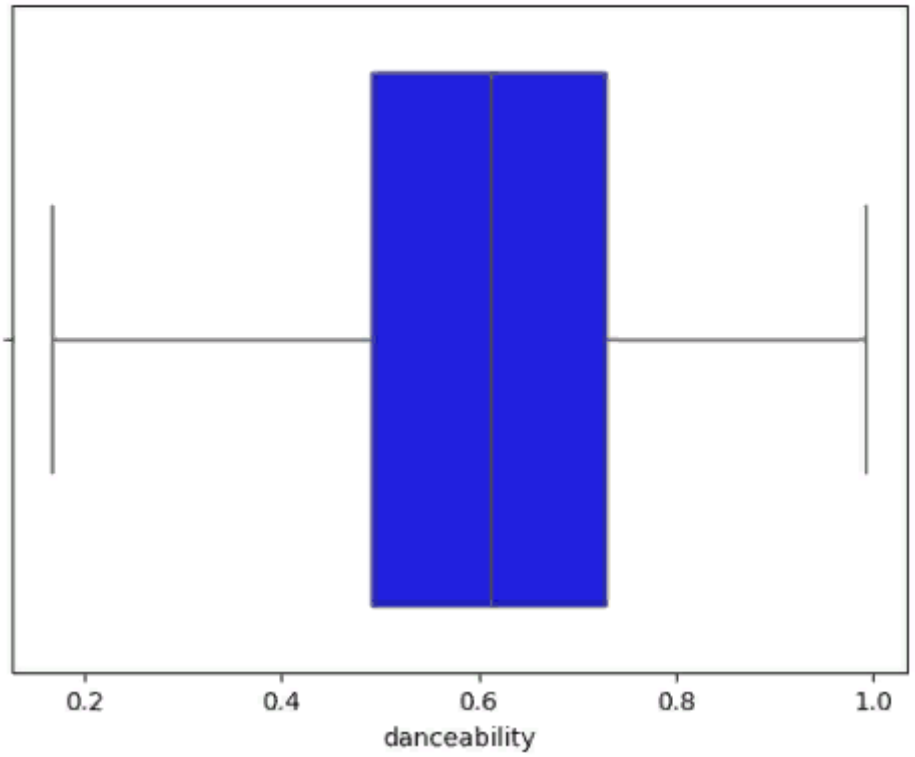


Рис.4.Ящик с усами параметра danceability

Нормализация

Была проведена нормализация данных с помощью метода MinMaxScaler()

```
from sklearn.preprocessing import StandardScaler
#стандартизация
scaler = StandardScaler()
features = scaler.fit_transform(ds)
standart_ds = pd.DataFrame(features,columns=['Unnamed: 0','popularity','duration_ms',
'danceability','energy','key','loudness','mode','speechiness','acousticness',
'instrumentalness','liveness','valence','tempo','time_signature'
])
standart_ds.describe()
#нормализация
normal = MinMaxScaler()
features = normal.fit_transform(standart_ds)
norm_ds = pd.DataFrame(features,columns=[
    'Unnamed: 0','popularity','duration_ms','danceability','energy',
'key','loudness','mode','speechiness','acousticness','instrumentalness',
'liveness','valence','tempo','time_signature'
])
norm_ds.describe()
```

	Unnamed: 0	popularity	duration_ms	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness
count	30000.00000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000
mean	0.50000	0.314746	0.045865	0.580996	0.632542	0.482918	0.714317	0.640600	0.103237	0.312877	0.217444
std	0.28869	0.215579	0.027704	0.179802	0.260355	0.323932	0.115849	0.479833	0.160380	0.345410	0.349961
min	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.25000	0.160000	0.033053	0.464903	0.454588	0.181818	0.672626	0.000000	0.037306	0.010442	0.000000
50%	0.50000	0.290000	0.041892	0.594100	0.679758	0.454545	0.744670	1.000000	0.050984	0.143072	0.000376
75%	0.75000	0.490000	0.053240	0.715158	0.853890	0.727273	0.790187	1.000000	0.089223	0.619478	0.383920
max	1.00000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Рис 5.Нормализация данных

Линейная регрессия

Я буду предсказывать параметр «Танцевальность» на основе других параметров. С помощью метода train_test_split, я разделил данные на две категории (train = 90% датасета).

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.1, random_state = 42)
```

Рис.6.train_test_split

Дальше берем метод линейной регрессии и обучаем и предсказываем ее на тестовой выборке. Получаем коэффициенты.

```
model = LinearRegression()
```

```
model.fit(X_train, Y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
Y_pred = model.predict(X_test)
```

```
coef = pd.DataFrame([X_train.columns, model.coef_]).T
coef = coef.rename (columns = {0:"Attribute", 1 : "Coefficient"})
print(coef)
```

```
           Attribute Coefficient
0         popularity   -0.000623
1        duration_ms    0.084876
2             energy    -0.21881
3                key    0.003376
4           loudness    0.179331
5               mode   -0.016957
6        speechiness     0.33993
7       acousticness    -0.12453
8   instrumentalness    0.034423
9           liveness   -0.156282
10           valence    0.312922
11             tempo   -0.177464
12    time_signature    0.475171
```

Рис.7.Линейная регрессия

Результат нашей модели составил:

```
train_score_lr = model.score(X_train, Y_train)
print("The train score for lr model is {}".format(train_score_lr))
test_score_lr = model.score(X_test, Y_test)
print("The test score for lr model is {}".format(test_score_lr))
```

```
The train score for lr model is 0.33709124056750805
The test score for lr model is 0.33849574458021714
```

Рис.8.Результат линейной регрессии

Визуализация предсказаний:

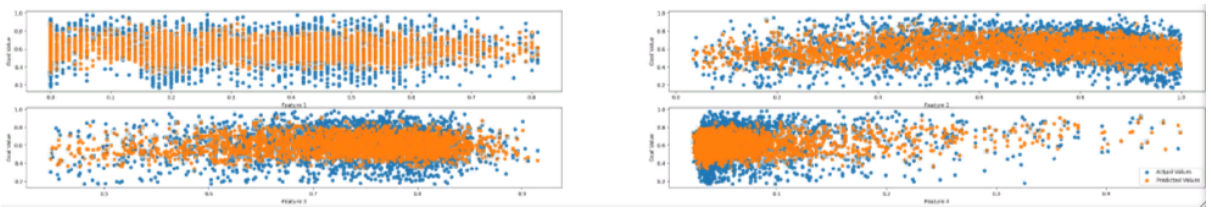


Рис.9.Визуализация предсказаний

Закидываем тестовые значения и получаем метрики:

```
R2_score = metrics.r2_score(Y_test, Y_pred)
print("LINEAR REGRESSION METRICS")
print("R^2: ", R2_score)
print("MAE: ", metrics.mean_absolute_error(Y_test, Y_pred))
print("MSE: ", metrics.mean_squared_error(Y_test, Y_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y_test, Y_pred)))
```

```
LINEAR REGRESSION METRICS
R^2:  0.33849574458021714
MAE:  0.11104335936178163
MSE:  0.0190818507315697
RMSE:  0.13813707225639937
```

R2 указывает долю точек данных, которые лежат внутри линии, созданной уравнением регрессии. Желательно более высокое значение R2, поскольку оно указывает на лучшие результаты.

MAE - это степень несоответствия между фактическими и прогнозируемыми значениями.

MSE : метрика, которая сообщает нам среднеквадратичную разницу между прогнозируемыми значениями и фактическими значениями в наборе данных. Чем ниже MSE, тем лучше модель соответствует набору данных.

Некоторая визуализация для нескольких признаков

Рис.10.Обучение модели на тестовых значениях и метрики

LASSO

Метод регрессии лассо (LASSO, Least Absolute Shrinkage and Selection Operator, оператор наименьшего абсолютного сжатия и выбора). Накладывает штраф на L1 – норму вектора «Постарайтесь достичь наилучшей производительности, но, если некоторые коэффициенты бесполезны, то их нужно отбросить» Мы переберем различные коэффициента регуляризации и для каждого создадим модель ЛАССО. Лучший коэффициент – имеет наименьшую квадратичную ошибку.

```
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
lambda1_values = [0.000001, 0.0001, 0.001, 0.005, 0.01, 0.05,  0.1, 0.2, 0.3, 0.4, 0.5, 1, 2, 5, 10, 100]# список коэффициента регуляризации 1
for lambda_val in lambda1_values:
    # для каждого коэффициента регуляризации создаем модель Lasso
    lasso_reg = Lasso(lambda_val)
    # обучение на тренировочной выборке
    lasso_reg.fit(X_train, Y_train)
    # прогноз
    Y_pred__ = lasso_reg.predict(X_test)
    # средняя квадратичная ошибка
    mse_lasso = mean_squared_error(Y_pred__, Y_test)
    print(("Lasso MSE with Lambda={} is {}").format(lambda_val, mse_lasso))
```

```
Lasso MSE with Lambda=1e-06 is 0.0190816413090519
Lasso MSE with Lambda=0.0001 is 0.019084663115014423
Lasso MSE with Lambda=0.001 is 0.019858786464601808
Lasso MSE with Lambda=0.005 is 0.021949302558867484
Lasso MSE with Lambda=0.01 is 0.02330137245725043
Lasso MSE with Lambda=0.05 is 0.028846241887399728
Lasso MSE with Lambda=0.1 is 0.028846241887399728
Lasso MSE with Lambda=0.2 is 0.028846241887399728
Lasso MSE with Lambda=0.3 is 0.028846241887399728
Lasso MSE with Lambda=0.4 is 0.028846241887399728
Lasso MSE with Lambda=0.5 is 0.028846241887399728
Lasso MSE with Lambda=1 is 0.028846241887399728
Lasso MSE with Lambda=2 is 0.028846241887399728
Lasso MSE with Lambda=5 is 0.028846241887399728
Lasso MSE with Lambda=10 is 0.028846241887399728
Lasso MSE with Lambda=100 is 0.028846241887399728
```

Рис.11.Метод Lasso и ошибки при разных коэффицентах

Наименьшая ошибка при коэффициенте, равном 0.000001.Обучим модель на тренировочных данных и предскажем на тестовых. Выведем коэффициенты.

```
lasso_reg = Lasso(0.000001)
Y_pred_LAS = lasso_reg.fit(X_train, Y_train)
Y_pred_LAS = lasso_reg.predict(X_test)
```

```
#выведем полученные коэффициенты для наших признаков
lasso_coef = pd.DataFrame([X_train.columns, lasso_reg.coef_]).T
lasso_coef = lasso_coef.rename (columns = {0:"Attribute", 1 : "Coefficient"})
print(lasso_coef)
```

	Attribute	Coefficient
0	popularity	-0.00062
1	duration_ms	0.080607
2	energy	-0.21852
3	key	0.003369
4	loudness	0.178587
5	mode	-0.01696
6	speechiness	0.339601
7	acousticness	-0.124546
8	instrumentalness	0.034387
9	liveness	-0.156258
10	valence	0.312852
11	tempo	-0.177372
12	time_signature	0.474825

Рис.12.Метод Lasso при 0.000001

Визуализация предсказаний:

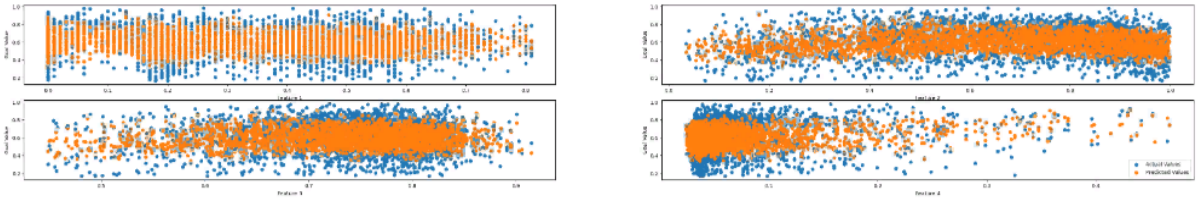


Рис.13.Визуализация предсказаний

Выводим метрики:

```
#оценка модели
R2_score = metrics.r2_score(Y_test, Y_pred_LAS)
print("LASSO REGRESSION METRICS")
print("R^2: ", R2_score)
print("MAE: ", metrics.mean_absolute_error(Y_test, Y_pred_LAS))
print("MSE: ", metrics.mean_squared_error(Y_test, Y_pred_LAS))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y_test, Y_pred_LAS)))
```

```
LASSO REGRESSION METRICS
R^2:  0.33850300456189586
MAE:  0.11104482803717194
MSE:  0.0190816413090519
RMSE:  0.1381363142300094
```

Рис.14.Вывод метрик

RIDGE

Модель, которая накладывает штраф на L2-норму вектора. «Постарайтесь добиться наилучшей производительности, но ни один из

коэффициентов не должен достичь экстремального значения» Мы переберем различные коэффициента регуляризации и для каждого создадим модель RIDGE. Лучший коэффициент – имеет наименьшую квадратичную ошибку.

```
from sklearn.linear_model import Ridge

lambda2_values = [0.000001, 0.0001, 0.001, 0.005, 0.01, 0.05,  0.1, 0.2, 0.3, 0.4, 0.5, 1, 2, 2.13, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 10, 100]# список коэффи


for lambda_val in lambda2_values:
    # для каждого коэффициента регуляризации создаем модель Ridge
    ridge_reg = Ridge(lambda_val)

    ridge_reg.fit(X_train, Y_train)
    y_pred = ridge_reg.predict(X_test)
    mse_ridge = mean_squared_error(y_pred, Y_test)
    print(("Lasso MSE with Lambda={} is {}").format(lambda_val, mse_ridge))
```

```
Lasso MSE with Lambda=1e-06 is 0.019081850729748533
Lasso MSE with Lambda=0.0001 is 0.019081850549457796
Lasso MSE with Lambda=0.001 is 0.01908184891085819
Lasso MSE with Lambda=0.005 is 0.01908184163706289
Lasso MSE with Lambda=0.01 is 0.0190818325651611
Lasso MSE with Lambda=0.05 is 0.019081760799408883
Lasso MSE with Lambda=0.1 is 0.019081673095761434
Lasso MSE with Lambda=0.2 is 0.019081504237840812
Lasso MSE with Lambda=0.3 is 0.019081343898112398
Lasso MSE with Lambda=0.4 is 0.019081191835635013
Lasso MSE with Lambda=0.5 is 0.019081047826318677
Lasso MSE with Lambda=1 is 0.019080441717522986
Lasso MSE with Lambda=2 is 0.019079737966386043
Lasso MSE with Lambda=2.13 is 0.019079691422289366
Lasso MSE with Lambda=2.4 is 0.019079625348537514
Lasso MSE with Lambda=2.5 is 0.019079611101435735
Lasso MSE with Lambda=2.6 is 0.01907960225739643
Lasso MSE with Lambda=2.7 is 0.01907959874111771
Lasso MSE with Lambda=2.8 is 0.019079600479671624
Lasso MSE with Lambda=2.9 is 0.01907960740235324
Lasso MSE with Lambda=10 is 0.01909030204713572
Lasso MSE with Lambda=100 is 0.019502370799670547
```

Рис.15.Ridge и ошибки при разных коэффициентах

Наименьшая ошибка при коэффициенте, равном 2.7. Обучим модель на тренировочных данных и предскажем на тестовых. Выведем коэффициенты.

```
ridge_reg = Ridge(2.7)
Y_pred_Ridge = ridge_reg.fit(X_train, Y_train)
Y_pred_Ridge = ridge_reg.predict(X_test)
#выведем полученные коэффициенты для наших признаков
ridge_coef = pd.DataFrame([X_train.columns, ridge_reg.coef_]).T
ridge_coef = ridge_coef.rename (columns = {0:"Attribute", 1 : "Coefficient"})
print(ridge_coef)
```

```
           Attribute Coefficient
0         popularity   -0.000454
1        duration_ms    0.053276
2             energy   -0.212919
3                key    0.003369
4           loudness    0.166422
5               mode   -0.017073
6        speechiness    0.330642
7       acousticness   -0.124345
8   instrumentalness    0.033412
9           liveness   -0.155395
10           valence    0.312093
11             tempo   -0.176311
12    time_signature    0.452999
```

Рис.16.RIDGE при L2 = 2

Визуализация данных:

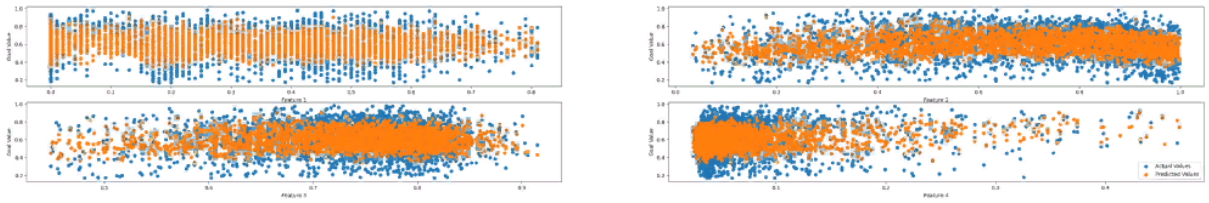


Рис.17.Визуализация данных

Выводим метрики:

```
#оценка модели
R2_score = metrics.r2_score(Y_test, Y_pred_Ridge)
print("RIDGE REGRESSION METRICS")
print("R^2: ", R2_score)
print("MAE: ", metrics.mean_absolute_error(Y_test, Y_pred_Ridge))
print("MSE: ", metrics.mean_squared_error(Y_test, Y_pred_Ridge))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y_test, Y_pred_Ridge)))
```

```
RIDGE REGRESSION METRICS
R^2:  0.3385738135939682
MAE:  0.111072322986149
MSE:  0.01907959874111771
RMSE:  0.13812892072668095
```

Рис.18.Вывод метрик

Метрики:

```python
#Вывод всех оценок моделей
R2_score = metrics.r2_score(Y_test, Y_pred)
print("LINEAR REGRESSION METRICS")
print("R^2: ", R2_score)
print("MAE: ", metrics.mean_absolute_error(Y_test, Y_pred))
print("MSE: ", metrics.mean_squared_error(Y_test, Y_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y_test, Y_pred)))

R2_score = metrics.r2_score(Y_test, Y_pred_LAS)
print("LASSO REGRESSION METRICS")
print("R^2: ", R2_score)
print("MAE: ", metrics.mean_absolute_error(Y_test, Y_pred_LAS))
print("MSE: ", metrics.mean_squared_error(Y_test, Y_pred_LAS))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y_test, Y_pred_LAS)))

R2_score = metrics.r2_score(Y_test, Y_pred_Ridge)
print("RIDGE REGRESSION METRICS")
print("R^2: ", R2_score)
print("MAE: ", metrics.mean_absolute_error(Y_test, Y_pred_Ridge))
print("MSE: ", metrics.mean_squared_error(Y_test, Y_pred_Ridge))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y_test, Y_pred_Ridge)))
```

```
LINEAR REGRESSION METRICS
R^2:  0.33849574458021714
MAE:  0.11104335936178163
MSE:  0.0190818507315697
RMSE:  0.13813707225639937
LASSO REGRESSION METRICS
R^2:  0.33850300456189586
MAE:  0.11104482803717194
MSE:  0.0190816413090519
RMSE:  0.1381363142300094
RIDGE REGRESSION METRICS
R^2:  0.3385738135939682
MAE:  0.111072322986149
MSE:  0.01907959874111771
RMSE:  0.13812892072668095
```

Рис.19.Все метрики

Подбор параметров

Попытаемся найти «лучшие» параметры для RIDGE и LASSO моделей.

```
from sklearn.linear_model import RidgeCV

#Lasso Cross validation
ridge_cv = RidgeCV(alphas = [0.000001, 0.0001, 0.001, 0.005, 0.01, 0.05,  0.1, 0.2, 0.3, 0.4, 0.5, 1, 2, 3, 100]).fit(X_train, Y_train)
y_pred_ridgeCV = ridge_cv.predict(X_test)
#Выведем результаты
reg_score_lr = ridge_cv.score(X_test, Y_test)
print("The test score for ridge model is {}".format(reg_score_lr))
```

```
The test score for ridge model is 0.3385235786327183
```

```
from sklearn.linear_model import LassoCV

#Lasso Cross validation
lasso_cv = LassoCV(alphas = [0.000000000000000001, 0.000000000001, 0.00000000001, 0.000000001, 0.0000001, 0.00001, 0.000001, 0.0001, 0.001, 0.0015, 0.
y_pred_lassoCV = lasso_cv.predict(X_test)
#Выведем результаты
reg_score_lasso = lasso_cv.score(X_test, Y_test)
print("The test score for lasso model is {}".format(reg_score_lasso))
```

```
The test score for lasso model is 0.33850300456189586
```

Рис.20.Перебор параметров для RIDGE и LASSO

По итогу подобрать параметры лучше не получилось.

Вывод: Я получил и закрепил навыки предобработки данных и применения методов машинного обучения для решения задач регрессии.