1. Модели и оценка их эффективности

Цель – получить навыки работы с библиотеками Python для задач анализа данных.

Задачи:

1. Загрузить в рабочую зону предоставленный набор данных.
2. Восстановить пропущенные значения.
3. Построить модель классификации и настроить её гиперпараметры.
4. Отобрать информативные признаки и оценить влияние этого процесса на работу модели.

Критерии оценки:

Каждый пункт задания оценивается в 2 балла.

Итого за выполнение можно получить 8 баллов.

Для положительной оценки необходимо получить не менее 5 баллов.

Загрузка набора данных и восстановление пропущенных значений

В первую очередь необходимо подключить необходимые библиотеки. Это можно сделать с помощью набора команд, изображенных на рисунке 1.1

```
import numpy as np
import pandas as pd
from sklearn.model_selection import KFold
from sklearn.metrics import f1_score
```

Рисунок 1.1 – Подключение библиотек

Обычно загрузка всех необходимых библиотек совершают в начале файла, добавляя их по мере необходимости. В данном случае первые две библиотеки уже вам знакомы, они нужны для работы с непосредственно набором данных. Следующие две библиотеки мы не подключаем полностью, а берем из них только две функции: Kfold и f1_score. Первая позволит воспользоваться методом перекрестной проверки для оценки

качества модели, а вторая использовать метрику качества модели под название F1-score.

Теперь приступим к загрузке набора данных. Это можно сделать с помощью команды read_csv из библиотеки pandas (рисунок 1.2).

```
data = pd.read_csv("train_new.csv")
```

Рисунок 1.2 – Загрузка набора данных

Следующим шагом всегда следует осмотр загруженного набора данных. Это делают с помощью команд head и shape (рисунок 1.3).

```
data.head()
```

	PassengerId	Survived	Pclass	Surname	Title	Sex	Age	FsizeD	Fsize	SibSp	Parch	Ticket	Fare	Deck	Embarked
0	1	0	3	Braund	Mr	male	22.0	small	2	1	0	A5	7.2500	X	S
1	2	1	1	Cumings	Mrs	female	38.0	small	2	1	0	PC	71.2833	C	C
2	3	1	3	Heikkinen	Miss	female	26.0	singleton	1	0	0	STONO2	7.9250	X	S
3	4	1	1	Futrelle	Mrs	female	35.0	small	2	1	0	X	53.1000	C	S
4	5	0	3	Allen	Mr	male	35.0	singleton	1	0	0	X	8.0500	X	S

```
data.shape
```

```
(891, 15)
```

Рисунок 1.3 – Осмотр набора данных

Целевой переменной является столбец под название Survived. Целевая переменная принимает значение 0 или 1, а значит перед нами случай бинарной классификации (переменная имеет всего две градации).

Частой проблемой при анализе данных и построение моделей в частности является наличие пропущенных значений в наборе. Проверить наличие пропущенных значений можно с помощью команды указанной на рисунке 1.4.

```
data.isna().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Surname          0
Title            0
Sex              0
Age            177
FsizeD           0
Fsize            0
SibSp            0
Parch            0
Ticket           0
Fare             0
Deck             0
Embarked         2
dtype: int64
```

Рисунок 1.4 – Проверка наличия пропущенных значений

Как и ожидалось были обнаружены пропущенные значения в столбцах Age и Embared. Существует множество способов борьбы с пропущенными значениями, но в данной работе мы рассмотрим самые простые из них, на рисунке 1.5 представлено подключение необходимой функции из библиотеки sklearn.

```
from sklearn.impute import SimpleImputer
```

Рисунок 1.5 – Подключение функции для обработки пропущенных значений

Стоит отметить что столбец Age является количественным признаком, в то время как Embared категориальный. В связи с этим для каждого и них придется использовать разные методы восстановления пропущенных значений. Пропущенные значения количественных признаков замещаются медианой, вычисленной по имеющимся значениям (рисунок 1.6).

```
imr = SimpleImputer(missing_values = np.nan, strategy='median')
imr = imr.fit(data[['Age']])
data['Age'] = imr.transform(data[['Age']])
```

Рисунок 1.6 – Замещение пропусков в количественном признаке

В свою очередь пропущенные значения категориальных признаков замещаются модой, то есть самой популярной категорией и имеющихся (рисунок 1.7).

```
imr = SimpleImputer(missing_values = np.nan, strategy = 'most_frequent')
imr = imr.fit(data[['Embarked']])
data['Embarked'] = imr.transform(data[['Embarked']])
```

Рисунок 1.7 – Замещение пропусков в категориальном признаке

Обязательно проверьте что теперь в наборе данных нет пропущенных значений ранее используемой командой.

Построение модели классификации и настройка гиперпараметров

Настало время перейти непосредственно к построению модели классификации. При работе в Python целевую переменную отделяют от остальных признаков. Как это сделать показано на рисунке 1.8.

```
X = data.drop("Survived", axis = 1)
y = data.Survived
```

Рисунок 1.8 – Отделение целевой переменной

Следующим шагом будет установка необходимой библиотеки. Дело в том, что метод Catboost, который мы будем использовать в данной работе не поставляется в комплекте Anaconda. Сделать это можно следующей командой (рисунок 1.9).

```
! pip install catboost
```

Рисунок 1.9 – Установка библиотеки

Теперь можно подключать библиотеку (рисунок 1.10).

```
from catboost import CatBoostClassifier, Pool
```

Рисунок 1.10 – Подключение библиотеки

Теперь задаём объект для проведения перекрестной проверки с помощью функции Kfold, которую мы подключили в начале работы. Для успешной работы Catboost необходимо явно передать номера

категориальных признаков. Эти номера были выписаны для вас заранее, проверьте нет ли в них ошибки (рисунок 1.11).

```
skf = KFold(n_splits = 5)
cat_feature = [2, 3, 4, 6, 10, 12, 13]
```

Рисунок 1.11 – Создание необходимых объектов

Теперь настало время инициализировать модель (рисунок 1.12). Объявленный пустой список под названием f1 в скором времени нам понадобится.

```
model = CatBoostClassifier(iterations = 100, depth = 2, learning_rate = 0.03, verbose = False)
f1 = []
```

Рисунок 1.12 – Инициализация модели

Далее представлен хоть и большой, но очень простой код. Метод перекрестной проверки работает таким образом: строки набора данных разделятся на заранее объявленное количество частей (в нашем случае пять), после чего каждая из пяти частей последовательно становится тестовой выборкой, а остальные четыре становятся обучающей выборкой. В связи с этим мы получаем возможность обучить и проверить эффективность модели несколько раз. Именно это и происходит во фрагменте кода представленном на рисунке 1.13. Результат оценки качества классификации записывается в список f1.

```
%%time
for train_index, test_index in skf.split(y):
    X_train = X.iloc[train_index]
    X_test = X.iloc[test_index]
    y_train = y[train_index]
    y_test = y[test_index]

    train_pool = Pool(X_train, y_train, cat_features = cat_feature)
    test_pool = Pool(X_test, cat_features = cat_feature)
    model.fit(train_pool)
    preds = model.predict(test_pool, prediction_type = 'Class')
    f1.append(f1_score(y_test, preds.flatten()))
```

```
Wall time: 2.02 s
```

Рисунок 1.13 – Оценка модели с помощью перекрестной проверки

Далее всё что нам осталось сделать это посмотреть среднюю точность классификации исходя из значений, полученных на каждом шаге перекрестной проверки (рисунок 1.14).

```
np.mean(f1)
```

```
0.7453192553576455
```

Рисунок 1.14 – Вычисление средней точности классификации

А какое значение получилось у вас? Попробуйте поменять значение гиперпараметров (iterations, depth и learning_rate), так что бы максимально повысить среднее качество классификации.

Отбор информативных признаков

Не все признаки являются одинаково полезными для построения моделей. Чаще всего среди них попадаются полностью бесполезные. Catboost позволяет оценить информативность признаков, для этого необходимо воспользоваться методом get_feature_importance (рисунок 1.15).

```
score_feature = model.get_feature_importance(Pool(X, y, cat_features = cat_feature))
```

Рисунок 1.15 – Применения метода оценки информативности признаков

На рисунке 1.16 представлен код позволяющий представить результат работы метода get_feature_importance в удобоваримом виде.

```
data_score = pd.DataFrame({"feature_name": X.columns, "score": score_feature})
data_score = data_score.sort_values(by = ['score'], ascending = False)
```

Рисунок 1.16 – Формирование таблицы с оценками информативности признаков

На рисунке 1.17 представлена итоговая таблица с оценкой информативности признаков отсортированной по убыванию.

```
data_score
```

	feature_name	score
3	Title	57.009346
4	Sex	13.694774
1	Pclass	10.192859
12	Deck	8.520197
6	FsizeD	6.553524
11	Fare	1.741405
7	Fsize	1.001980
5	Age	0.422160
13	Embarked	0.391605
8	SibSp	0.373225
10	Ticket	0.098926
0	PassengerId	0.000000
2	Surname	0.000000
9	Parch	0.000000

Рисунок 1.17 – Оценка информативности признаков

Как мы видим часть признаков получили не очень высокие оценки информативности, возможно если их исключить, это повысит качество модели. На рисунках 1.18 и 1.19 представлен процесс формирования нового набора данных, с учетом исключения признаков, не соответствующих выставленным условиям.

```
new_columns = data_score.feature_name[data_score.score > 0.2]
```

Рисунок 1.18 – Выбор информативных признаков

```
X_new_columns = X[new_columns]
X_new_columns
```

Рисунок 1.19 – Формирование нового набора данных

На рисунке 1.20 представлен код автоматически формирующих список категориальных признаков для успешной работы Catboost. Он будет нам полезен для выполнения последнего задания.

```
cat_feature = [X_new_columns.columns.get_loc(i) for i in X_new_columns.dtypes[X_new_columns.dtypes == "object"].index]
```

Рисунок 1.20 – Создание списка категориальных переменных

Последующий код практически повторяет рисунок 1.13, за одним исключение теперь используется набор данных с отобранными информативными признаками (рисунок 1.21).

```
f1_new = []
```

```
%%time
for train_index, test_index in skf.split(y):
    X_train = X_new_columns.iloc[train_index]
    X_test = X_new_columns.iloc[test_index]
    y_train = y[train_index]
    y_test = y[test_index]

    train_pool = Pool(X_train, y_train, cat_features = cat_feature)
    test_pool = Pool(X_test, cat_features = cat_feature)
    model.fit(train_pool)
    preds = model.predict(test_pool, prediction_type = 'Class')
    f1_new.append(f1_score(y_test, preds.flatten()))
```

```
Wall time: 2.19 s
```

```
np.mean(f1_new)
```

```
0.7462963124708141
```

Рисунок 1.21 – Оценка качества модели на новом наборе данных

Попробуйте изменить условие в фрагменте кода на рисунке 1.18, и проверьте как меняется средняя точность классификации, если мы будем оставлять различное количество исходных признаков. Не забудьте после каждого изменения условия запускать код на рисунке 1.20, так как признаки в наборе данных будут каждый раз меняться, будут меняться и номера категориальных признаков, а значит их каждый раз нужно вычислять заново.

Обязательно запишите в отчет сколько и каких признаков нужно оставить для оптимальной работы модели.

2. Исследовательский анализ данных

Цель работы:

получить навыки работы с библиотеками **Pandas, Numpy.**

Задачи:

- Установить необходимые библиотеки
- Импортировать библиотеки
- Загрузить набор данных
- Изучить существующие функции и проделать агрегации

Ответ на задание необходимо предоставить в виде файла формата в формате **Word** или **PDF.** В файле снимками экрана с текстовыми пояснениями должны быть зафиксированы основные результаты работы **(**созданы по инструкции и самостоятельно**).**

Критерии оценки:

1) Выполнение задания**:**

- Выполнено меньше половины задания **0** баллов
- Большая часть выполнена **1** балл
- Выполнено полностью **2** балла

2) Качество отчета**:**

- Сильные огрехи в оформлении или удовлетворяет менее половины требований к отчету **0** баллов
- Хорошее оформление**.** Удовлетворяет большей части требований **1** балл
- Соответствует всем требованиям **2** балла

3) Программный код**:**

- Код нерабочий или непонятный**. 0** баллов
- Программный код рабочий**.** Нет комментариев или есть незначительные огрехи к его написанию **1** балл
- Полностью понятный код**.** Содержит комментарии к коду **2** балла

4) Выводы по работе**:**

- Выводов нет или они не соответствуют полученным результатам **0** баллов
- Выполнены не по всем разделам работы или частично объясняют результат **1** балл
- Выполнены по всем разделам работы и полностью объясняют результат **2** балла

Итого за выполнение лабораторной работы можно получить **8** баллов**.**

Ход работы:

Импортируем библиотеку для работы с датафреймами

In [1]:

```
import pandas as pd
```

Используем функцию **read_csv** для считывания данных из файла **csv**

In [2]:

```
data = pd.read_csv('data.csv').drop(columns=['Unnamed: 0', 'id', 'key']).set_index(['art
```

```
ists', 'name'])
data.head()
```

Out[2]:

artists	name	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	liveness	loudness	mo
['Dennis Day']	Clancy Lowered the Boom	0.732	0.819	180533	0.341	0	0.000000	0.160	-12.441	
['Sergei Rachmaninoff', 'James Levine', 'Berliner Philharmoniker']	Piano Concerto No. 3 in D Minor, Op. 30: III. Finale. Alla breve	0.982	0.279	831667	0.211	0	0.878000	0.665	-20.096	
['John McCormack']	The Wearing of the Green	0.996	0.518	159507	0.203	0	0.000000	0.115	-10.589	
['Sergei Rachmaninoff', 'James Levine', 'Berliner Philharmoniker']	Piano Concerto No. 3 in D Minor, Op. 30: III. Finale. Alla breve	0.982	0.279	831667	0.211	0	0.878000	0.665	-20.096	
['Phil Regan']	When Irish Eyes Are Smiling	0.957	0.418	166693	0.193	0	0.000002	0.229	-10.096	

Вывести первые **n** записей датафрейма

In [3]:

```
data.head(3)
```

Out[3]:

artists	name	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	liveness	loudness	mo
['Dennis Day']	Clancy Lowered the Boom	0.732	0.819	180533	0.341	0	0.000	0.160	-12.441	
['Sergei Rachmaninoff', 'James Levine', 'Berliner Philharmoniker']	Piano Concerto No. 3 in D Minor, Op. 30: III. Finale. Alla breve	0.982	0.279	831667	0.211	0	0.878	0.665	-20.096	
['John McCormack']	The Wearing of the Green	0.996	0.518	159507	0.203	0	0.000	0.115	-10.589	

Вывести последние **5** записей датафрейма

In [4]:

```
data.tail()
```

Out[4]:

artists	name	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	liveness	loudness	mode	p
['Kelly Clarkson']	Born to Die	0.6430	0.481	205787	0.3680	0	0.000000	0.125	-8.310	0	
['JoJo']	Man	0.6700	0.661	173760	0.5800	1	0.000055	0.117	-7.718	1	
['S.J Morgan']	Rivers	0.9790	0.502	160125	0.0355	0	0.867000	0.106	-26.940	1	
['Childish Gambino']	0.00	0.6720	0.174	179387	0.0466	0	0.196000	0.420	-18.458	1	
['Alina Baraz', '6LACK']	Morocco (feat. 6LACK)	0.0686	0.615	141941	0.7130	0	0.001440	0.154	-3.539	1	

Вывести статистические характеристики для каждого численного признака

In [5]:

```
data.describe()
```

Out[5]:

	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	liveness	loud
count	168592.000000	168592.000000	1.685920e+05	168592.000000	168592.000000	168592.000000	168592.000000	168592.0
mean	0.501360	0.533648	2.327016e+05	0.488577	0.071516	0.169476	0.205151	-11.3
std	0.377993	0.175919	1.223921e+05	0.267346	0.257685	0.315383	0.175896	5.6
min	0.000000	0.000000	5.108000e+03	0.000000	0.000000	0.000000	0.000000	-60.0
25%	0.097800	0.412000	1.721600e+05	0.265000	0.000000	0.000000	0.098200	-14.3
50%	0.515000	0.543000	2.091330e+05	0.480000	0.000000	0.000264	0.134000	-10.4
75%	0.896000	0.662000	2.637070e+05	0.709000	0.000000	0.111000	0.259000	-7.1
max	0.996000	0.988000	5.403500e+06	1.000000	1.000000	1.000000	1.000000	3.8

Вывести типы данных для каждого признака

In [6]:

```
data.dtypes
```

Out[6]:

```
acousticness        float64
danceability        float64
duration_ms           int64
energy              float64
explicit              int64
instrumentalness    float64
liveness            float64
loudness            float64
mode                  int64
popularity            int64
```

```
popularity            int64
release_date         object
speechiness         float64
tempo               float64
valence             float64
year                  int64
dtype: object
```

Вывести размеры датафрейма

In [7]:

```
data.shape
```

Out[7]:

```
(168592, 15)
```

Вывести количество значений в датафрейме **(**количество признаков * количество записей**)**

In [8]:

```
data.size
```

Out[8]:

```
2528880
```

Использование агрегации средним **('mean')** по году **('year')** колонки **'duration_ms'** и вывод первых пяти записей **(head)**

In [9]:

```
data.groupby('year').aggregate({'duration_ms':'mean'}).head()
```

Out[9]:

year	duration_ms
1921	**229911.914062**
1922	**167904.541667**
1923	**178356.301775**
1924	**184891.512712**
1925	**184130.699620**

Использование **.iloc** для вывода конкретных записей и столбцов датафрейма

In [10]:

```
data.iloc[0:5,0:3]
```

Out[10]:

		acousticness	danceability	duration_ms
artists	**name**			
['Dennis Day']	**Clancy Lowered the Boom**	**0.732**	**0.819**	**180533**
['Sergei Rachmaninoff', 'James Levine', 'Berliner Philharmoniker']	**Piano Concerto No. 3 in D Minor, Op. 30: III. Finale. Alla breve**	**0.982**	**0.279**	**831667**
['John McCormack']	**The Wearing of the Green**	**0.996**	**0.518**	**159507**
['Sergei Rachmaninoff', 'James Levine', 'Berliner Philharmoniker']	**Piano Concerto No. 3 in D Minor, Op. 30: III. Finale. Alla breve**	**0.982**	**0.279**	**831667**
['Phil Regan']	**When Irish Eyes Are Smiling**	**0.957**	**0.418**	**166693**

		acousticness	danceability	duration_ms

In [11]:

```
data.iloc[[0, 3, 5], [0, 3, -1]]
```

Out[11]:

artists	name	acousticness	energy	year
['Dennis Day']	Clancy Lowered the Boom	0.732	0.341	1921
['Sergei Rachmaninoff', 'James Levine', 'Berliner Philharmoniker']	Piano Concerto No. 3 in D Minor, Op. 30: III. Finale. Alla breve	0.982	0.211	1921
['Phil Regan']	Come Back To Erin	0.957	0.212	1921

Отрицательные индексы для нумерации с конца.

-1 - последний элемент

In [12]:

```
data.iloc[:,-3:-1].head()
```

Out[12]:

artists	name	tempo	valence
['Dennis Day']	Clancy Lowered the Boom	60.936	0.9630
['Sergei Rachmaninoff', 'James Levine', 'Berliner Philharmoniker']	Piano Concerto No. 3 in D Minor, Op. 30: III. Finale. Alla breve	80.954	0.0594
['John McCormack']	The Wearing of the Green	66.221	0.4060
['Sergei Rachmaninoff', 'James Levine', 'Berliner Philharmoniker']	Piano Concerto No. 3 in D Minor, Op. 30: III. Finale. Alla breve	80.954	0.0594
['Phil Regan']	When Irish Eyes Are Smiling	101.665	0.2530

Тип объекта определяется с помощью метода **type()**

Выбирая несколько колонок датафрейма **-** получаем датафрейм **(pandas.core.frame.DataFrame)**

In [13]:

```
type(data.iloc[:,-3:-1])
```

Out[13]:

```
pandas.core.frame.DataFrame
```

Выбирая одну колонку **-** получаем серию **(pandas.core.series.Series)**

In [14]:

```
data.iloc[:,0].head()
```

Out[14]:

```
artists                                                 name

['Dennis Day']                                          Clancy Lowered the B
oom                                             0.732
['Sergei Rachmaninoff', 'James Levine', 'Berliner Philharmoniker']  Piano Concerto No. 3
in D Minor, Op. 30: III. Finale. Alla breve     0.982
['John McCormack']                                      The Wearing of the G
reen                                            0.996
['Sergei Rachmaninoff', 'James Levine', 'Berliner Philharmoniker']  Piano Concerto No. 3
```

```
[ 'Sergei Rachmaninoff' , 'James Levine' , 'Berliner Philharmoniker' ]  Piano Concerto No. 3
in D Minor, Op. 30: III. Finale. Alla breve    0.982
['Phil Regan']                                                          When Irish Eyes Are
Smiling                                         0.957
Name: acousticness, dtype: float64
```

In [15]:

```
type(data.iloc[:,0])
```

Out[15]:

```
pandas.core.series.Series
```

Создадим серию

In [16]:

```
pd.Series([1,2,3])
```

Out[16]:

```
0    1
1    2
2    3
dtype: int64
```

Создадим две серии, запишем их в переменные.

In [17]:

```
my_series_1 = pd.Series([1,2,3], index=['Tom', 'Tim', 'Sam'])
my_series_1
```

Out[17]:

```
Tom    1
Tim    2
Sam    3
dtype: int64
```

In [18]:

```
my_series_2 = pd.Series([4,5,6], index=['Tom', 'Tim', 'Sam'])
my_series_2
```

Out[18]:

```
Tom    4
Tim    5
Sam    6
dtype: int64
```

Соеденим две серии в датафрейм

In [19]:

```
pd.DataFrame({'col_1':my_series_1, 'col_2':my_series_2})
```

Out[19]:

	col_1	col_2
Tom	1	4
Tim	2	5
Sam	3	6

Серия или датафрейм

Есть возможность выбрать одну колонку получив не серию, а датафрейм

Есть возможность выбрать одну колонку получив не серию, а датафрейм

In [20]:

```
type(data['year'])
```

Out[20]:

```
pandas.core.series.Series
```

In [21]:

```
type(data[['year']])
```

Out[21]:

```
pandas.core.frame.DataFrame
```

Работа с признаками

- Преобразование колонки **release_date** к типу **datetime.**
- Оставляем из даты только год

In [22]:

```
data.release_date = pd.to_datetime(data.release_date, yearfirst=True)
data.release_date = data.release_date.dt.year
```

In [23]:

```
data.head(3)
```

Out[23]:

		acousticness	danceability	duration_ms	energy	explicit	instrumentalness	liveness	loudness	mo
artists	name									
['Dennis Day']	Clancy Lowered the Boom	0.732	0.819	180533	0.341	0	0.000	0.160	-12.441	
['Sergei Rachmaninoff', 'James Levine', 'Berliner Philharmoniker']	Piano Concerto No. 3 in D Minor, Op. 30: III. Finale. Alla breve	0.982	0.279	831667	0.211	0	0.878	0.665	-20.096	
['John McCormack']	The Wearing of the Green	0.996	0.518	159507	0.203	0	0.000	0.115	-10.589	

In [ ]:

```

```

3. Построение системы ИИ с помощью инструментов **Scikit-Learn** для решения задачи классификации

Цель работы:

научиться строить модели классификации.

Задачи:

- Научиться строить классификаторы
- Научиться оценивать его качество
- Изучить вклад используемых переменных в решения классификатора
- Научиться визуализировать классификатор (дерево решений)

Ответ на задание необходимо предоставить в виде файла формата в формате **Word** или **PDF.** В файле снимками экрана с текстовыми пояснениями должны быть зафиксированы основные результаты работы (созданы по инструкции и самостоятельно).

Критерии оценки:

1) Выполнение задания:

- Выполнено меньше половины задания **0** баллов
- Большая часть выполнена **1** балл
- Выполнено полностью **2** балла

2) Качество отчета:

- Сильные огрехи в оформлении или удовлетворяет менее половины требований к отчету **0** баллов
- Хорошее оформление. Удовлетворяет большей части требований **1** балл
- Соответствует всем требованиям **2** балла

3) Программный код:

- Код нерабочий или непонятный. **0** баллов
- Программный код рабочий. Нет комментариев или есть незначительные огрехи к его написанию **1** балл
- Полностью понятный код. Содержит комментарии к коду **2** балла

4) Выводы по работе:

- Выводов нет или они не соответствуют полученным результатам **0** баллов
- Выполнены не по всем разделам работы или частично объясняют результат **1** балл
- Выполнены по всем разделам работы и полностью объясняют результат **2** балла

Итого за выполнение лабораторной работы можно получить **8** баллов.

Алгоритм выполнения

- Разделить набор данных на входные данные для модели и целевую переменную
- Построить классификатор
- Изучить вклад переменных (важность переменных)
- Визуализировать дерево решений
- Сформировать отчет о проделанной работе

Импортируем библиотеки

In [1]:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import graphviz
from sklearn.tree import export_graphviz
from IPython.display import SVG
```

Используем функцию **read_csv** для считывания данных из файла **csv**

In [2]:

```
data = pd.read_csv('data.csv').drop(columns=['Unnamed: 0', 'id', 'key']).set_index(['art
ists', 'name'])
```

Изменим тип признака **release_date** на **datetime** и извлечем только год**.**

In [3]:

```
data.release_date = pd.to_datetime(data.release_date, yearfirst=True)
data.release_date = data.release_date.dt.year
```

Выберем целевой переменной **- popularity,** убрав ее из набора данных для обучения **X.**

In [4]:

```
X = data.drop(columns=['popularity'])
y = data.popularity
```

Создадим классификатор **-** решающее дерево**,** ограничим его глубиной **3.**

In [5]:

```
clf = DecisionTreeClassifier(random_state=0, max_depth=3, min_samples_leaf=5)
```

Обучим классификатор

In [6]:

```
clf.fit(X,y)
```

Out[6]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=5, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

Узнаем вклад переменных в решения классификатора

In [7]:

```
clf.feature_importances_
```

Out[7]:

```
array([0.01841238, 0.        , 0.00499254, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.00514254, 0.        , 0.        , 0.97145253])
```

Для более удобного воспринятия информации о важности переменных построим датафрейм с названиями переменных и их важностью**.**

In [9]:

```
pd.DataFrame(data = clf.feature_importances_, index=X.columns, columns=['feature_importa
```

```
nces'])
```

Out[9]:

	feature_importances
acousticness	0.018412
danceability	0.000000
duration_ms	0.004993
energy	0.000000
explicit	0.000000
instrumentalness	0.000000
liveness	0.000000
loudness	0.000000
mode	0.000000
release_date	0.000000
speechiness	0.005143
tempo	0.000000
valence	0.000000
year	0.971453

Визуализируем получившееся дерево

In [ ]:

```
dot_data = export_graphviz(clf, out_file=None, filled=True)
graph = graphviz.Source(dot_data)
display(SVG(graph.pipe(format='svg')))
```