Лабораторная 5. Деревья решений

Импорт библиотек

In [400…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
from math import sqrt, ceil, log2
```

Выбор датасетов: Студенты с **нечетным** порядковым номером в группе должны использовать датасет **с данными про оценки студентов инженерного и педагогического факультетов**.

In [401…
```
seed = 421337
```

In [402…
```
data = pd.read_csv('data/higher_education_students_performance_evaluation.cs
data
```

Out[402…

	STUDENT ID	1	2	3	4	5	6	7	8	9	…	23	24	25	26	27	28	2
0	STUDENT1	2	2	3	3	1	2	2	1	1	…	1	1	3	2	1	2	
1	STUDENT2	2	2	3	3	1	2	2	1	1	…	1	1	3	2	3	2	
2	STUDENT3	2	2	2	3	2	2	2	2	4	…	1	1	2	2	1	1	
3	STUDENT4	1	1	1	3	1	2	1	2	1	…	1	2	3	2	2	1	
4	STUDENT5	2	2	1	3	2	2	1	3	1	…	2	1	2	2	2	1	
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
140	STUDENT141	2	1	2	3	1	1	2	1	1	…	1	1	2	1	2	1	
141	STUDENT142	1	1	2	4	2	2	2	1	4	…	1	1	3	2	2	1	
142	STUDENT143	1	1	1	4	2	2	2	1	1	…	1	1	3	3	2	1	
143	STUDENT144	2	1	2	4	1	1	1	5	2	…	2	1	2	1	2	1	
144	STUDENT145	1	1	1	5	2	2	2	3	1	…	2	1	3	2	3	1	

145 rows × 33 columns

In [403…
```
data.describe()
```

Out[403…

	1	2	3	4	5	6
count	145.000000	145.000000	145.000000	145.000000	145.000000	145.000000
mean	1.620690	1.600000	1.944828	3.572414	1.662069	1.600000
std	0.613154	0.491596	0.537216	0.805750	0.474644	0.491596
min	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000
25%	1.000000	1.000000	2.000000	3.000000	1.000000	1.000000
50%	2.000000	2.000000	2.000000	3.000000	2.000000	2.000000
75%	2.000000	2.000000	2.000000	4.000000	2.000000	2.000000
max	3.000000	2.000000	3.000000	5.000000	2.000000	2.000000

8 rows × 32 columns

In [404…

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 145 entries, 0 to 144
Data columns (total 33 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   STUDENT ID  145 non-null    object
 1   1           145 non-null    int64 
 2   2           145 non-null    int64 
 3   3           145 non-null    int64 
 4   4           145 non-null    int64 
 5   5           145 non-null    int64 
 6   6           145 non-null    int64 
 7   7           145 non-null    int64 
 8   8           145 non-null    int64 
 9   9           145 non-null    int64 
 10  10          145 non-null    int64 
 11  11          145 non-null    int64 
 12  12          145 non-null    int64 
 13  13          145 non-null    int64 
 14  14          145 non-null    int64 
 15  15          145 non-null    int64 
 16  16          145 non-null    int64 
 17  17          145 non-null    int64 
 18  18          145 non-null    int64 
 19  19          145 non-null    int64 
 20  20          145 non-null    int64 
 21  21          145 non-null    int64 
 22  22          145 non-null    int64 
 23  23          145 non-null    int64 
 24  24          145 non-null    int64 
 25  25          145 non-null    int64 
 26  26          145 non-null    int64 
 27  27          145 non-null    int64 
 28  28          145 non-null    int64 
 29  29          145 non-null    int64 
 30  30          145 non-null    int64 
 31  COURSE ID   145 non-null    int64 
 32  GRADE       145 non-null    int64 
dtypes: int64(32), object(1)
memory usage: 37.5+ KB
```

In [405…

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

Out[405…
```
STUDENT ID    0
1             0
2             0
3             0
4             0
5             0
6             0
7             0
8             0
9             0
10            0
11            0
12            0
13            0
14            0
15            0
16            0
17            0
18            0
19            0
20            0
21            0
22            0
23            0
24            0
25            0
26            0
27            0
28            0
29            0
30            0
COURSE ID     0
GRADE         0
dtype: int64
```

In [406…
```
X = data.drop(['GRADE', 'STUDENT ID'], axis=1)
y = data['GRADE']
```

Визуализация

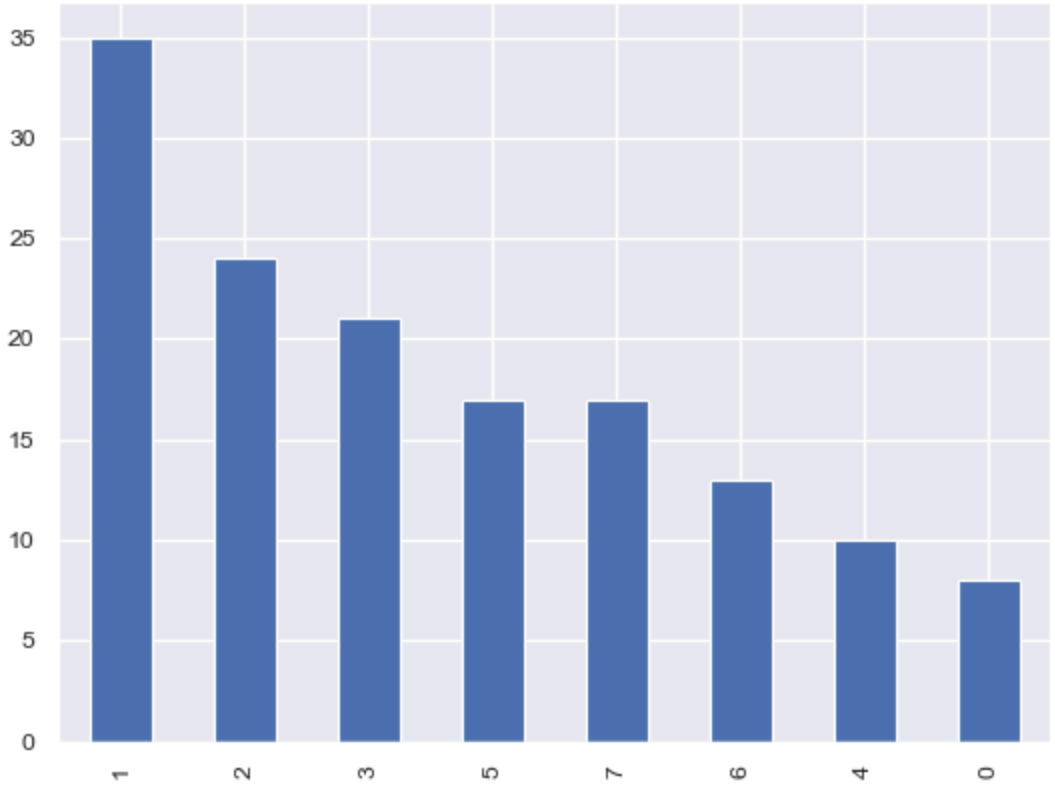
In [407…
```
fig, axs = plt.subplots(ceil(sqrt(len(X.columns))), ceil(sqrt(len(X.columns)
for ax, col in zip(axs.flatten(), X.columns):
    X[col].value_counts().plot(kind="bar", ax=ax).set_title(col)
plt.show()
```

In [408…

```
y.value_counts().plot(kind="bar")
```

Out[408…

```
<Axes: >
```

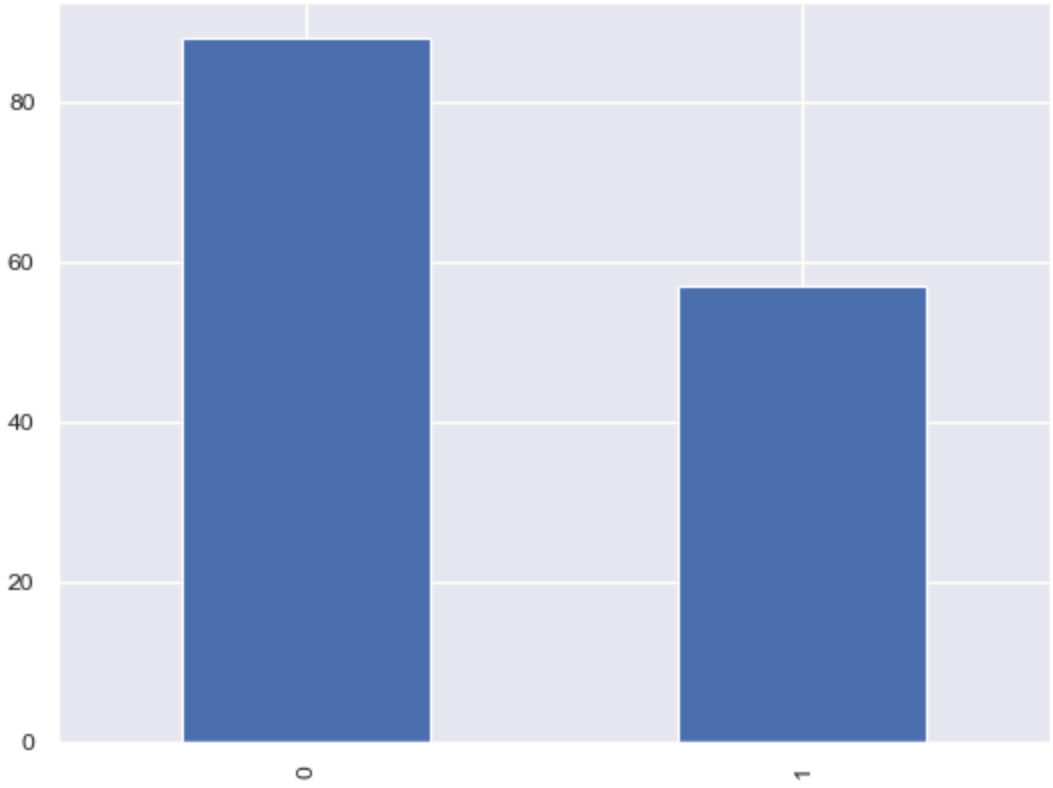


По условию нужно ввести для данного датасета метрику: студент **успешный/неуспешный на основании грейда**

In [409…
```
# Определение порогового значения для успешности
threshold = 4

y = pd.Series([1 if i >= threshold else 0 for i in y])
y.value_counts().plot(kind="bar")
```

Out[409… <Axes: >



Успешным будем считать студента, набравшего >= 4 баллов.

Отобрать случайным образом `sqrt(n)` признаков

In [410…
```
columns = X.columns
columns = np.random.choice(columns, ceil(sqrt(len(columns))), replace=False)
X = X[columns]
X.head()
```

Out[410…

	8	COURSE ID	15	18	30	10
0	1	1	2	2	1	1
1	1	1	2	2	3	1
2	2	1	2	1	2	2
3	2	1	2	1	2	2
4	3	1	2	1	2	4

In [411…
```
columns
```

Out[411…
```
array(['8', 'COURSE ID', '15', '18', '30', '10'], dtype=object)
```

Разделение данных на обучающий и тестовый наборы

In [412…

```
def _train_test_split(X, Y, seed, test_percent=0.2):
  random.seed(seed)
  random.shuffle(list(range(len(X))))

  test_size = int(len(X) * test_percent)

  x_train = X[test_size:]
  x_test = X[:test_size]
  y_train = Y[test_size:]
  y_test = Y[:test_size]

  return x_train, x_test, y_train, y_test

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, rar

# x_train, x_test, y_train, y_test = _train_test_split(X, y, seed, 0.2)
x_train
```

Out[412…

	8	COURSE ID	15	18	30	10
73	1	3	2	1	3	2
77	2	4	4	2	1	2
83	3	5	1	2	2	1
94	1	6	2	2	4	1
136	1	9	3	2	2	3
...	...	...	...	...	...	...
29	1	1	4	2	3	2
18	3	1	2	3	3	1
139	2	9	5	1	2	3
41	2	1	2	2	3	2
127	1	9	3	1	2	3

116 rows × 6 columns

Дерево решений

In [413…

```
class Node:
  def __init__(self, feature, value):
    self.feature = feature
    self.value = value

  def pred(self, x):
    feature_value = x[self.feature]
    # Если значение признака отсутствует, возвращаем None
    if feature_value not in self.value:
```

```
            return None
        # Рекурсивно вызываем предсказание для следующего узла
        return self.value[feature_value].pred(x)

    def pred_proba(self, x):
        feature_value = x[self.feature]
        # Если значение признака отсутствует, возвращаем 0
        if feature_value not in self.value:
            return 0, 0
        # Рекурсивно вызываем предсказание с вероятностью для следующего узла
        return self.value[feature_value].pred_proba(x)

    def print_node(self, indent=0):
        # Выводим информацию о текущем узле и рекурсивно вызываем вывод для доче
        for value, node in self.value.items():
            print(' ' * indent + f'{self.feature} == {value}:')
            node.print_node(indent + 2)
```

In [414…

```
class Leaf(Node):
    def __init__(self, leaf_value, proba):
        # Листовой узел не имеет feature и value
        super().__init__("", {})
        self.leaf_value = leaf_value
        self.proba = proba

    def pred(self, x):
        # Возвращаем значение листового узла
        return self.leaf_value

    def pred_proba(self, x):
        # Возвращаем значение листового узла и его вероятность
        return self.leaf_value, self.proba

    def print_node(self, indent=0):
        # Выводим информацию о листовом узле
        print(' ' * indent + f'-> {self.leaf_value} ({self.proba})')
```

In [415…

```
class DecisionTree():
    def __init__(self, columns: list[str]):
        self.columns = columns
        self.root = None

    def fit(self, x, y):
        # Построение дерева начинается с корня
        self.root = self.build_node(x[self.columns], y, DecisionTree.entropy(y))
        return self.root

    def pred(self, x):
        # Возвращаем предсказания для каждой строки входных данных
        return [self.root.pred(i) for i in x[self.columns].to_records()]

    def pred_proba(self, x):
        # Возвращаем предсказания с вероятностями для каждой строки входных данн
        return [self.root.pred_proba(i) for i in x[self.columns].to_records()]
```

```
  def print_tree(self):
    # Выводим дерево, начиная с корня
    self.root.print_node()

  @staticmethod
  def entropy(y):
    # Вычисляем энтропию для меток классов
    class_n = y.unique()
    res = 0
    for i in class_n:
      res -= (y.value_counts()[i] / len(y)) * log2(y.value_counts()[i] / ler
    return res

  def build_node(self, x, y, parent_entropy: float) -> Node:
    # Если в узле осталась метка только одного класса, создаем листовой узел
    if len(y.unique()) == 1:
      return Leaf(y.unique()[0], 1)

    # best_gain, best_gain_info, best_gain_col
    max_gain = 0
    gain_info = 0
    max_gain_column = ''

    # Поиск лучшего разбиения
    for column in self.columns:
        features_names =  pd.unique(x[column]) # x[column].unique()
        info = sum(x[column].value_counts()[feature] / len(x) * DecisionTree

        # Обновляем параметры лучшего разбиения
        if parent_entropy - info > max_gain:
            max_gain = parent_entropy - info
            gain_info = info
            max_gain_column = column

    # Если разделить не удается, возвращаем листовой узел с самым частым кла
    if not max_gain_column:
        mode = y.mode()[0]
        return Leaf(mode, y.value_counts()[mode] / len(y))

    values = {}

    features_names = x[max_gain_column].unique()
    for feature in features_names:
        values[feature] = self.build_node(x[x[max_gain_column] == feature],

    return Node(max_gain_column, values)
```

Training

In [416…

```
model = DecisionTree(columns)
model.fit(x_train, y_train)
y_pred = model.pred(x_test)
```

```
print("Используются признаки:", *columns)
model.print_tree()
```

```
Используются признаки: 8 COURSE ID 15 18 30 10
COURSE ID == 3:
  -> 1 (1)
COURSE ID == 4:
  -> 1 (1)
COURSE ID == 5:
  -> 1 (1)
COURSE ID == 6:
  -> 1 (1)
COURSE ID == 9:
  10 == 3:
    -> 0 (1)
  10 == 1:
    -> 0 (1)
  10 == 2:
    30 == 4:
      -> 0 (1)
    30 == 3:
      -> 1 (1)
    30 == 2:
      -> 0 (1)
COURSE ID == 1:
  -> 0 (0.7818181818181819)
COURSE ID == 8:
  -> 0 (1)
COURSE ID == 7:
  30 == 4:
    -> 1 (1)
  30 == 3:
    -> 1 (1)
  30 == 2:
    -> 1 (1)
  30 == 1:
    -> 0 (1)
COURSE ID == 2:
  8 == 1:
    -> 1 (1)
  8 == 3:
    -> 0 (1)
```

In [417…

```
def accuracy(y_test, y_pred):
  return np.sum(y_test == y_pred) / len(y_test)
accuracy(y_test, y_pred)
```

Out[417…

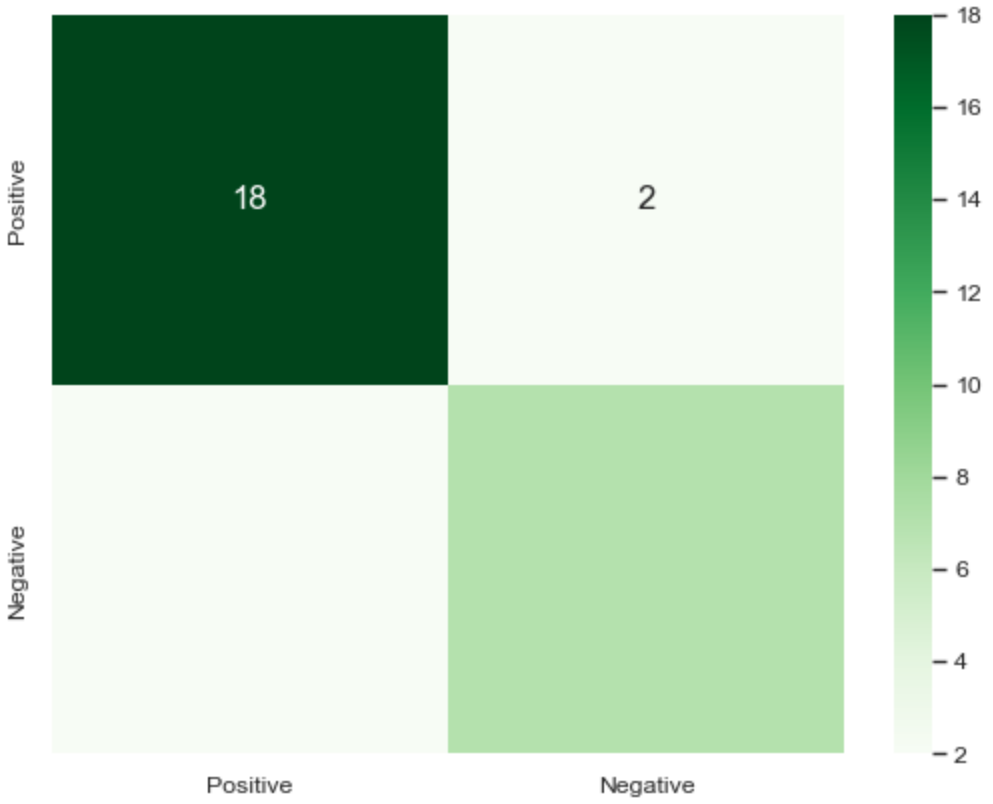
```
0.8620689655172413
```

In [418…

```
import seaborn as sn
def confusion_matrix(pred_y, true_y):
    matrix = np.zeros((2, 2))
    for pred, true in zip(pred_y, true_y):
        pred = 1 if pred == 1 else 0
        true = 1 if true == 1 else 0
        matrix[pred][true] += 1
```

```
    return matrix
```

In [419…
```
cm_indeces = ['Positive', 'Negative']
df_cm = pd.DataFrame(confusion_matrix(y_pred, y_test), index = cm_indeces, c
sn.set_context("paper", rc={"font.size":12,"axes.titlesize":8,"axes.labelsiz
sn.heatmap(df_cm, annot=True, fmt='.0f', cmap="Greens")
plt.show()
```



In [422…
```
def accuracy(conf):
    return (conf[1][1] + conf[0][0]) / (conf[1][1] + conf[0][0] + conf[1][0]

def precision(conf):
    return conf[1][1] / (conf[1][1] + conf[1][0])

def recall(conf):
    return conf[1][1] / (conf[1][1] + conf[0][1])

def tpr(conf):
    return recall(conf)

def fpr(conf):
    return conf[1][0] / (conf[1][0] + conf[0][0])

print("Accuracy: ", accuracy(confusion_matrix(y_pred, y_test)))
print("Precision: ", precision(confusion_matrix(y_pred, y_test)))
print("Recall: ", recall(confusion_matrix(y_pred, y_test)))
```

```
Accuracy:  0.8620689655172413
Precision:  0.7777777777777778
Recall:  0.7777777777777778
```

In [423…
```
y_pred_proba = model.pred_proba(x_test)
```

In [424…
```
def confusion_matrix_proba(y_pred_proba, true_y, threshold):
    matrix = np.zeros((2, 2))
    for pred_prob, true in zip(y_pred_proba, true_y):
        pred = 1 if pred_prob >= threshold else 0
        true = 1 if true == 1 else 0
        matrix[pred][true] += 1

    return matrix
```

In [425…
```
def probas(y_pred_proba):
  return [prob if pred == 1 else 1 - prob for pred, prob in y_pred_proba]
```

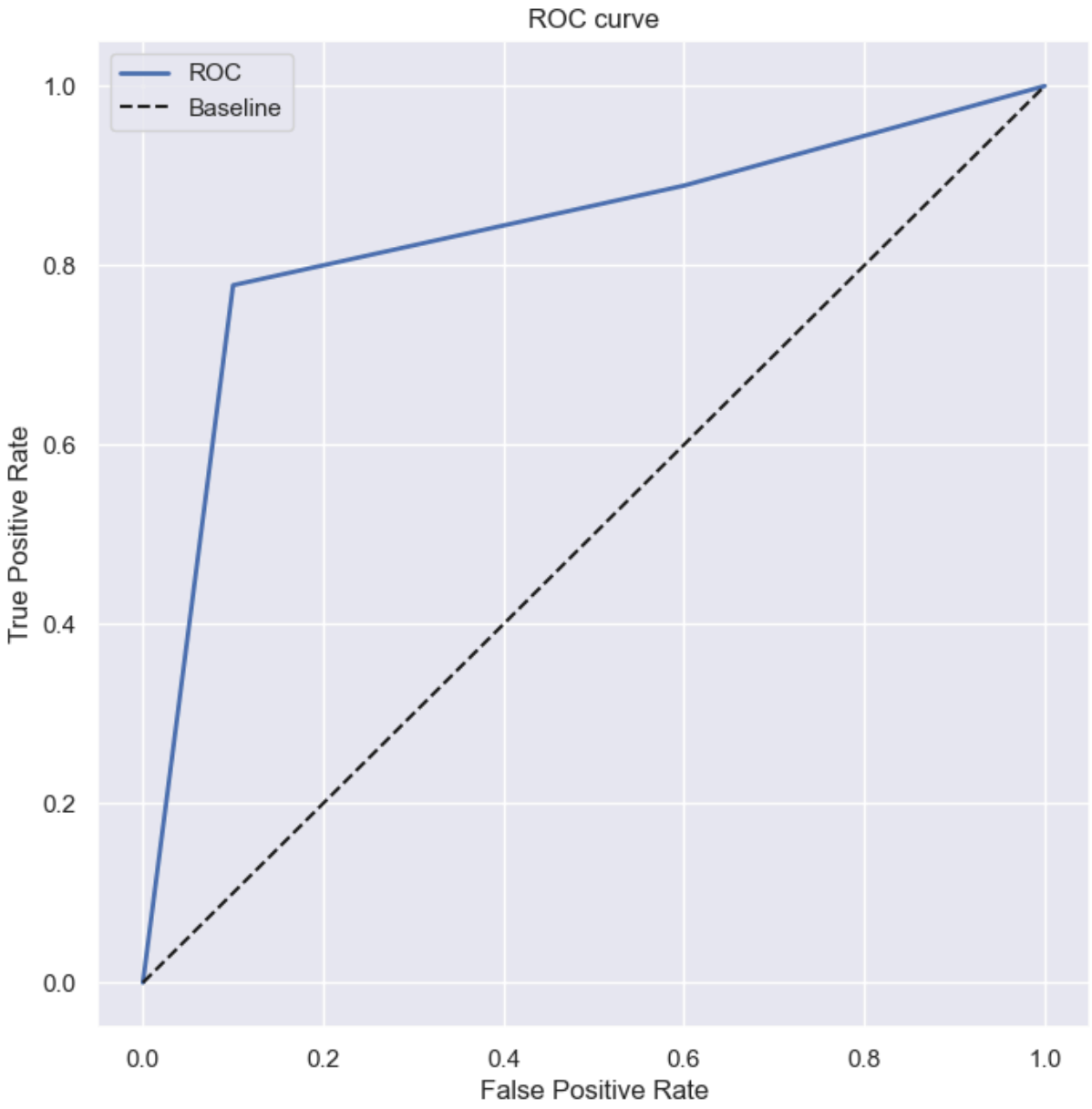
In [426…
```
import seaborn as sns
def auc_roc_plot(y_pred_proba):
    sns.set(font_scale=1)
    sns.set_color_codes("muted")
    plt.figure(figsize=(8, 8))
    tpr_arr = []
    fpr_arr = []
    for th in np.arange(1, 0, -0.01):
        conf = confusion_matrix_proba(probas(y_pred_proba), y_test, th)
        tpr_arr.append(tpr(conf))
        fpr_arr.append(fpr(conf))
    display(pd.DataFrame({'tpr': tpr_arr, 'fpr': fpr_arr}))

    plt.plot([0] + fpr_arr + [1], [0] + tpr_arr + [1], lw=2, label='ROC')
    plt.plot(np.linspace(0, 1, 100), np.linspace(0, 1, 100), 'k--', label='E
    plt.title('ROC curve')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend()
    plt.show()
```

In [427…
```
auc_roc_plot(y_pred_proba)
```

	tpr	fpr
0	0.777778	0.1
1	0.777778	0.1
2	0.777778	0.1
3	0.777778	0.1
4	0.777778	0.1
...	...	...
95	0.888889	0.6
96	0.888889	0.6
97	0.888889	0.6
98	0.888889	0.6
99	0.888889	0.6

100 rows × 2 columns

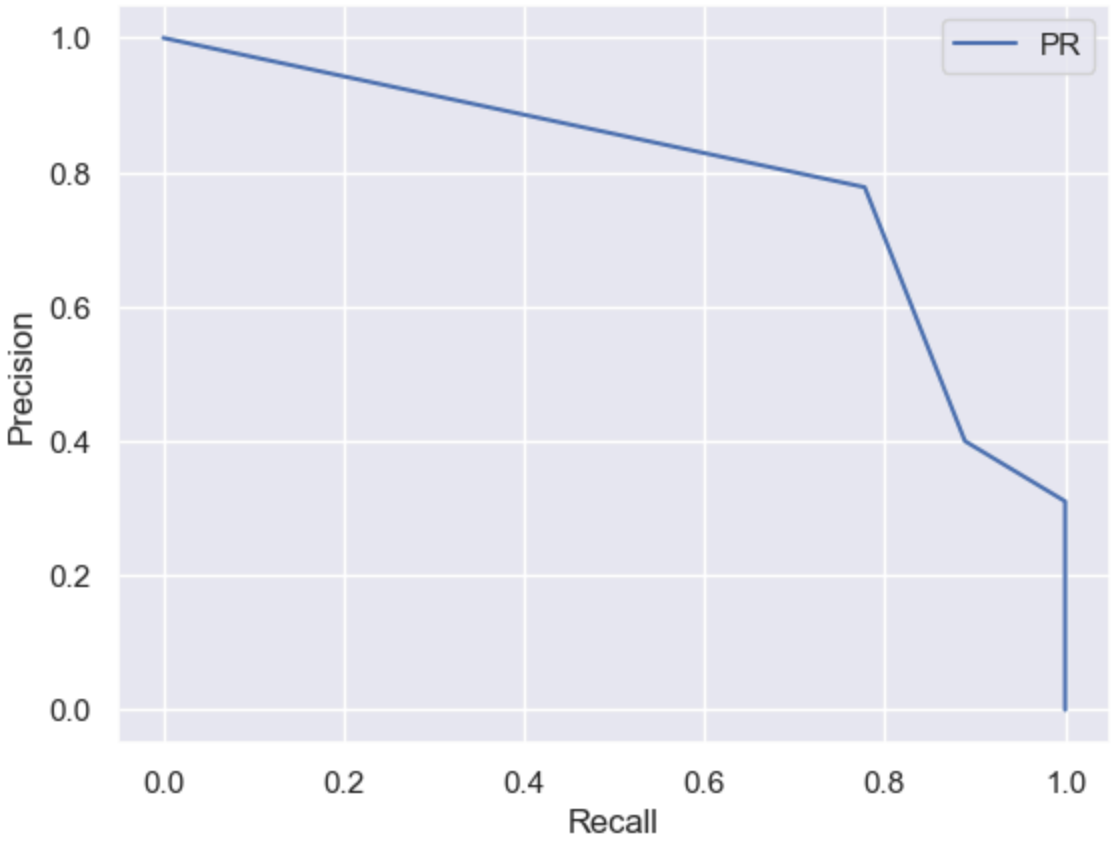


In [428…
```
def auc_pr_plot(y_pred_proba):
  p = []
  r = []

  for th in np.arange(0, 1, 0.01):
    conf = confusion_matrix_proba(probas(y_pred_proba), y_test, th)
    p.append(precision(conf))
    r.append(recall(conf))

  plt.plot([1] + r + [0], [0] + p + [1], label='PR')
  plt.xlabel('Recall')
  plt.ylabel('Precision')
  plt.legend()
```

In [429…
```
auc_pr_plot(y_pred_proba)
```



In [ ]: