Курс “Python для DataScience”

Практическое задание

Тема “Обучение с учителем”

Задание 1

- Импортируйте библиотеки pandas и numpy.
- Загрузите "Boston House Prices dataset" из встроенных наборов данных библиотеки sklearn.
- Создайте датафреймы X и Y из этих данных.
- Разбейте эти датафреймы н а тренировочные (X_train, y_train) и тестовые (X_test, y_test) с помощью функции train_test_split так, чтобы размер тестовой выборки составлял 30% от всех данных, при этом аргумент random_state должен быть равен 42.
- Создайте модель линейной регрессии под названием lr с помощью класса LinearRegression из модуля sklearn.linear_model.
- Обучите модель на тренировочных данных (используйте все признаки) и сделайте предсказание на тестовых.
- Вычислите R2 полученных предказаний с помощью r2_score из модуля sklearn.metrics.

Задание 2

- Создайте модель под названием model с помощью RandomForestRegressor из модуля sklearn.ensemble.
- Сделайте агрумент n_estimators равным 1000, max_depth должен быть равен 12 и random_state сделайте равным 42.
- Обучите модель на тренировочных данных аналогично тому, как вы обучали модель LinearRegression, но при этом в метод fit вместо датафрейма y_train поставьте y_train.values[:, 0], чтобы получить из датафрейма одномерный массив Numpy, так как для класса RandomForestRegressor в данном методе для аргумента y предпочтительно применение массивов вместо датафрейма.
- Сделайте предсказание на тестовых данных и посчитайте R2. Сравните с результатом из предыдущего задания.
- Напишите в комментариях к коду, какая модель в данном случае работает лучше.

*Задание 3

- Вызовите документацию для класса RandomForestRegressor,
- найдите информацию об атрибуте feature_importances_.
- С помощью этого атрибута найдите сумму всех показателей важности,
- установите, какие два признака показывают наибольшую важность.

*Задание 4

В этом задании мы будем работать с датасетом, с которым мы уже знакомы по домашнему заданию по библиотеке Matplotlib, это датасет Credit Card Fraud Detection.Для этого датасета мы будем решать задачу классификации - будем определять,какие из транзакциции по кредитной карте являются мошенническими. Данный датасет сильно несбалансирован (так как случаи мошенничества относительно редки),так что применение метрики accuracy не принесет пользы и не поможет выбрать лучшую модель.Мы будем вычислять AUC, то есть площадь под кривой ROC.

Импортируйте из соответствующих модулей RandomForestClassifier, GridSearchCV и train_test_split.

Загрузите датасет creditcard.csv и создайте датафрейм df.

С помощью метода value_counts с аргументом normalize=True убедитесь в том, что выборка несбалансирована.Используя метод info, проверьте, все ли столбцы содержат числовые данные и нет ли в них пропусков.Примените следующую настройку, чтобы можно было просматривать все столбцы датафрейма:

- pd.options.display.max_columns = 100.
- Просмотрите первые 10 строк датафрейма df.
- Создайте датафрейм X из датафрейма df, исключив столбец Class.
- Создайте объект Series под названием y из столбца Class.

Разбейте X и y на тренировочный и тестовый наборы данных при помощи функции train_test_split, используя аргументы: test_size=0.3, random_state=100, stratify=y.

У вас должны получиться объекты X_train, X_test, y_train и y_test.

Просмотрите информацию о их форме.

Для поиска по сетке параметров задайте такие параметры:

- parameters = [{'n_estimators': [10, 15],
- 'max_features': np.arange(3, 5),
- 'max_depth': np.arange(4, 7)}]

Создайте модель GridSearchCV со следующими аргументами:

- estimator=RandomForestClassifier(random_state=100),
- param_grid=parameters,
- scoring='roc_auc',
- cv=3.

Обучите модель на тренировочном наборе данных (может занять несколько минут).

Просмотрите параметры лучшей модели с помощью атрибута best_params_.

Предскажите вероятности классов с помощью полученнной модели и метода predict_proba.

Из полученного результата (массив Numpy) выберите столбец с индексом 1 (вероятность класса 1) и запишите в массив y_pred_proba. Из модуля sklearn.metrics импортируйте метрику roc_auc_score.

Вычислите AUC на тестовых данных и сравните с результатом,полученным на тренировочных данных, используя в качестве аргументовмассивы y_test и y_pred_proba.

**Дополнительные задания:

1. Загрузите датасет Wine из встроенных датасетов sklearn.datasets с помощью функции load_wine в переменную data.
2. Полученный датасет не является датафреймом. Это структура данных, имеющая ключи аналогично словарю. Просмотрите тип данных этой структуры данных и создайте список data_keys, содержащий ее ключи.
3. Просмотрите данные, описание и названия признаков в датасете. Описание нужно вывести в виде привычного, аккуратно оформленного текста, без обозначений переноса строки, но с самими переносами и т.д.
4. Сколько классов содержит целевая переменная датасета? Выведите названия классов.
5. На основе данных датасета (они содержатся в двумерном массиве Numpy) и названий признаков создайте датафрейм под названием X.
6. Выясните размер датафрейма X и установитпе, имеются ли в нем пропущенные значения.
7. Добавьте в датафрейм поле с классами вин в виде чисел, имеющих тип данных numpy.int64. Название поля - 'target'.
8. Постройте матрицу корреляций для всех полей X. Дайте полученному датафрейму названиеX_corr.
9. Создайте список high_corr из признаков, корреляция которых с полем target по абсолютному значению превышает 0.5 (причем, само поле target не должно входить в этот список).
10. Удалите из датафрейма X поле с целевой переменной. Для всех признаков, названия которых содержатся в списке high_corr, вычислите квадрат их значений и добавьте в датафрейм X соответствующие поля с суффиксом '_2', добавленного к первоначальному названию признака. Итоговый датафрейм должен содержать все поля, которые, были в нем изначально, а также поля с признаками из списка high_corr, возведенными в квадрат. Выведите описание полей датафрейма X с помощью метода describe.

Задание 1

1.1 Импортируйте библиотеки pandas и numpy.

B [1]:
```
import warnings

warnings.filterwarnings('ignore')
```

B [2]:
```
import numpy as np
import pandas as pd
```

1.2 Загрузите "Boston House Prices dataset" из встроенных наборов данных библиотеки sklearn.

B [3]:
```
from sklearn.datasets import load_boston
```

B [4]:
```
boston = load_boston()

boston.keys()  # Смотрим содержимое dataseta
```

Out[4]:
```
dict_keys(['data', 'target', 'feature_names', 'DESCR', 'filename'])
```

1.3 Создайте датафреймы X и Y из этих данных.

B [5]:
```
data = boston["data"]  # Данные о недвижимости (хранятся в массиве по ключу "data").
```

B [6]:
```
data.shape
```

Out[6]:
```
(506, 13)
```

B [7]:
```
data
```

Out[7]:
```
array([[6.3200e-03, 1.8000e+01, 2.3100e+00, ..., 1.5300e+01, 3.9690e+02,
        4.9800e+00],
       [2.7310e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9690e+02,
        9.1400e+00],
       [2.7290e-02, 0.0000e+00, 7.0700e+00, ..., 1.7800e+01, 3.9283e+02,
        4.0300e+00],
       ...,
       [6.0760e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        5.6400e+00],
       [1.0959e-01, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9345e+02,
        6.4800e+00],
       [4.7410e-02, 0.0000e+00, 1.1930e+01, ..., 2.1000e+01, 3.9690e+02,
        7.8800e+00]])
```

B [8]:
```
feature_names = boston["feature_names"] # Название признаков

feature_names[12]
```

Out[8]:
```
'LSTAT'
```

```
- CRIM     уровень преступности на душу населения в разбивке по городам
- ZN       доля жилой земли, зонированной под участки площадью более 25 000 кв.фт.
- INDUS    доля торговых акров, не относящихся к розничной торговле, в расчете на один город
- CHAS     Фиктивная переменная Чарльза Ривера (= 1, если район граничит с рекой; 0 в противном случае)
- NOX      концентрация оксидов азота (части на 10 миллионов)
- RM       среднее количество комнат на одно жилище
- AGE      доля занятых владельцами единиц, построенных до 1940 года
```

- AGE   доля занятых владельцами единиц, построенных до 1940 года
- DIS   взвешенные расстояния до пяти бостонских центров занятости
- RAD   индекс доступности радиальных магистралей
- TAX   ставка налога на полную стоимость имущества за 10 000 долл. США
- PTRATIO соотношение между учениками и учителями в разбивке по городам
- B   1000 (Bk - 0,63) ^ 2, где Bk - доля чернокожих по городам
- LSTAT   Доля населения с более низким статусом (социальным?)
- MEDV   Медианная стоимость жилья, занимаемого владельцами, в тысячах долларов

B [9]:
```
print(boston["DESCR"]) # Описание датасета
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attribute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/ (https://archive.ics.uci.edu/ml/machine-learning-databases/housing/)


This dataset was taken from the StatLib library which is maintained at Carnegie Mellon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that address regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proceedings on the Tenth International Conference of Machine Learning, 236-243, University of Massachu
setts, Amherst. Morgan Kaufmann.
```

B [10]:
```
target = boston["target"] # Массив с целевыми значениями (цены на недвижимость)

target[:10]
```

Out[10]:
```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9])
```

B [11]:
```
X = pd.DataFrame(data, columns=feature_names)

X.head()
```

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

B [12]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    float64
 9   TAX      506 non-null    float64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
dtypes: float64(13)
memory usage: 51.5 KB
```

B [13]:
```
y = pd.DataFrame(target, columns=["price"])

y.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   price   506 non-null    float64
dtypes: float64(1)
memory usage: 4.1 KB
```

In [14]:
```
y.head()
```

Out[14]:

	price
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2

1.4 Разбейте эти датафреймы н а тренировочные (X_train, y_train) и тестовые (X_test, y_test) с помощью функции train_test_split так, чтобы размер тестовой выборки составлял 30% от всех данных, при этом аргумент random_state должен быть равен 42.

In [15]:
```
from sklearn.model_selection import train_test_split
```

In [16]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

1.5 Создайте модель линейной регрессии под названием lr с помощью класса LinearRegression из модуля sklearn.linear_model.

In [17]:
```
from sklearn.linear_model import LinearRegression  # Загружаем модель линейной регрессии
```

In [18]:
```
lr = LinearRegression()  # исходная модель
```

1.6 Обучите модель на тренировочных данных (используйте все признаки) и сделайте предсказание на тестовых.

In [19]:
```
lr.fit(X_train, y_train)  # обучаем модель используя тренировочную выборку
```

Out[19]:
```
LinearRegression()
```

In [20]:
```
y_pred = lr.predict(X_test)  # делаем предсказание на тестовых данных
y_pred.shape
```

Out[20]:
```
(152, 1)
```

In [21]:
```
# y_pred
```

In [22]:
```
# Сопоставляем реальные значения с предсказанными.
# Переведём массив y_pred в одномерный, используя метод .flatten.

check_test = pd.DataFrame({"y_test": y_test["price"],
                           "y_pred": y_pred.flatten()},
                          columns=['y_test', 'y_pred'])
```

B [23]:
```
check_test.head(10)
```

Out[23]:

	y_test	y_pred
173	23.6	28.648960
274	32.4	36.495014
491	13.6	15.411193
72	22.8	25.403213
452	16.1	18.855280
76	20.0	23.146689
316	17.8	17.392124
140	14.0	14.078599
471	19.6	23.036927
500	16.8	20.599433

1.7 Вычислите R2 (коэффициент детерминации) полученных предказаний с помощью r2_score из модуля sklearn.metrics.

B [24]:
```
from sklearn.metrics import r2_score
r2_score_1 = r2_score(check_test["y_test"], check_test["y_pred"])
r2_score_1
```

Out[24]:
```
0.711226005748496
```

Задание 2

2.1 Создайте модель под названием model с помощью RandomForestRegressor из модуля sklearn.ensemble. Сделайте агрумент n_estimators равным 1000, max_depth должен быть равен 12 и random_state сделайте равным 42.

B [25]:
```
# Случайный лес (Random Forest)
from sklearn.ensemble import RandomForestRegressor
```

В [26]:
```
# Создаём модель RandomForestRegressor
model = RandomForestRegressor(n_estimators = 1000,  # Число деревьев
                              max_depth = 12,  # Максимальная глубина деревьев
                              random_state = 42 # Предустановка генератора псевдослучайных чисел
                             )

# n_estimators - Число деревьев
# max_depth - Максимальная глубина деревьев
# max_features - Число признаков для выбора расщепления
# min_samples_split - Минимальное число объектов, при котором выполняется расщепление
# min_samples_leaf - Ограничение на число объектов в листьях
# criterion - Критерий расщепления
#
# По умолчанию в sklearn-овских методах n_jobs=1, т.е. случайный лес строится на одном процессоре
# Если Вы хотите существенно ускорить построение, используйте n_jobs=-1 (строить на максимально возможном числе процессоров)
# Построение воспроизводимых экспериментов используйтся предустановка генератора псевдослучайных чисел: random_state.
#
# Случайный лес (Random Forest) - https://dyakonov.org/2016/11/14/случайный-лес-random-forest/
```

2.3 Обучите модель на тренировочных данных аналогично тому, как вы обучали модель LinearRegression, но при этом в метод fit вместо датафрейма y_train поставьте y_train.values[:, 0],чтобы получить из датафрейма одномерный массив Numpy, так как для класса RandomForestRegressor в данном методе для аргумента y предпочтительно применение массивов вместо датафрейма.

В [27]:
```
model.fit(X_train, y_train.values[:, 0])
```

Out[27]:
```
RandomForestRegressor(max_depth=12, n_estimators=1000, random_state=42)
```

2.4 Сделайте предсказание на тестовых данных и посчитайте R2. Сравните с результатом из предыдущего задания.

В [28]:
```
y_pred = model.predict(X_test)
y_pred.shape
```

Out[28]:
```
(152,)
```

В [29]:
```
check_test = pd.DataFrame({
 "y_test": y_test["price"],
 "y_pred": y_pred.flatten(),
})
check_test.head(10)
```

Out[29]:

	y_test	y_pred
173	23.6	22.806412
274	32.4	31.131464
491	13.6	16.339125
72	22.8	23.810726
452	16.1	17.139521
76	20.0	21.832284
316	17.8	19.895747
140	14.0	14.754118
471	19.6	21.240835
500	16.8	20.898658

In [30]:
```
# Вычисляем  R2 коэффициент детерминации полученных предказаний

r2_score_2 = r2_score(check_test["y_pred"], check_test["y_test"])

print("r2_score_2 = ",r2_score_2)
```

```
r2_score_2 =  0.8479049999699443
```

In [31]:
```
# Сравниваем с результатом из предыдущего задания.

r2_score_1 < r2_score_2
```

Out[31]:
```
True
```

2.5 Напишите в комментариях к коду, какая модель в данном случае работает лучше.

In [32]:
```
# Вывод: модель RandomForestRegressor работает лучше, чем модель LinearRegression,
#
# так как r2_score_2 > r2_score_1,  где
#
# r2_score_2 - коэффициент детерминации R2 модели RandomForestRegressor,
# r2_score_1 - коэффициент детерминации R2 модели LinearRegression.
#
# r2_score_2 =  0.8479049999699443
# r2_score_1 =  0.711226005748496
#

print("r2_score_2 = ",r2_score_2)
print("r2_score_1 = ",r2_score_1)
```

```
r2_score_2 =  0.8479049999699443
r2_score_1 =  0.711226005748496
```

*Задание 3

3.1 Вызовите документацию для класса RandomForestRegressor,

In [33]:
```
?RandomForestRegressor
#??RandomForestRegressor
```

- найдите информацию об атрибуте feature_importances_.

In [34]:
```
#feature_importances_ : ndarray of shape (n_features,)
#    The impurity-based feature importances.
#    The higher, the more important the feature.
#    The importance of a feature is computed as the (normalized)
#    total reduction of the criterion brought by that feature.  It is also
#    known as the Gini importance.

# feature_importances_ возвращает вектор "важностей" признаков.
# Индекс элемента в этом векторе соответствует индексу признака в данных.
# Значение элемента отражает "важность" признака относительно остальных:
#      чем больше значение, тем больше важность.
```

3.2 С помощью этого атрибута найдите сумму всех показателей важности,

В [35]:
```
print(model.feature_importances_)
```

```
[0.03167574 0.00154252 0.00713813 0.00123624 0.01426897 0.40268179
 0.01429864 0.06397257 0.00528122 0.01152493 0.01808108 0.01245085
 0.41584732]
```

В [36]:
```
model.feature_importances_.sum()
```

Out[36]:
```
1.0
```

- установите, какие два признака показывают наибольшую важность.

В [37]:
```
feature_names
```

Out[37]:
```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

В [38]:
```
max_value_idx_1=model.feature_importances_.argmax()
print('1-й признак: max_value_idx_1 = ', max_value_idx_1)
# 12 - LSTAT (% более низкий статус населения)
```

```
1-й признак: max_value_idx_1 =  12
```

В [39]:
```
print('model.n_features_ = ', model.n_features_)
max_value_idx_2 = 0
max_value = model.feature_importances_[max_value_idx_2]
for i in range(model.n_features_):
    if max_value < model.feature_importances_[i] and i != max_value_idx_1:
        max_value = model.feature_importances_[i]
        max_value_idx_2 = i

print('\n2-й признак: max_value_idx_2 = ', max_value_idx_2)
# 5 - RM (среднее количество комнат на одно жилище)
```

```
model.n_features_ =  13

2-й признак: max_value_idx_2 =  5
```

*Задание 4

В этом задании мы будем работать с датасетом, с которым мы уже знакомы по домашнему заданию по библиотеке Matplotlib, это датасет Credit Card Fraud Detection. Для этого датасета мы будем решать задачу классификации - будем определять,какие из транзакциии по кредитной карте являются мошенническими. Данный датасет сильно несбалансирован (так как случаи мошенничества относительно редки),так что применение метрики accuracy не принесет пользы и не поможет выбрать лучшую модель. Мы будем вычислять AUC, то есть площадь под кривой ROC.

Импортируйте из соответствующих модулей RandomForestClassifier, GridSearchCV и train_test_split.

В [40]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, train_test_split
```

Загрузите датасет creditcard.csv и создайте датафрейм df.

В [41]:
```
DATASET_PATH = './creditcard.csv' # Путь до датасета, можно взять на kaggle.com (California Housing Prices)
PREPARED_DATASET_PATH = './creditcard.csv_prepared.csv' # Путь до предобработанного датасета
DATASET_PATH
```

Out[41]:
```
'./creditcard.csv'
```

В [42]:
```
df = pd.read_csv(DATASET_PATH, sep=',')  # Читаем DataSet
df.head()
```

Out[42]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

5 rows × 31 columns

С помощью метода value_counts с аргументом ***normalize=True*** убедитесь в том, что выборка несбалансирована.

В [43]:
```
df["Class"].value_counts(normalize=True)
```

Out[43]:
```
0    0.998273
1    0.001727
Name: Class, dtype: float64
```

В [44]:
```
vc_class = df.value_counts('Class', normalize=True)
vc_class
```

Out[44]:
```
Class
0    0.998273
1    0.001727
dtype: float64
```

Используя метод ***info***, проверьте, все ли столбцы содержат числовые данные

B [45]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

и нет ли в них пропусков.

B [46]:
```
#df.isnull().astype(np.int).sum().sum().astype(np.int)
df.isnull().astype(np.int).sum()
```

Out[46]:
```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

Примените следующую настройку, чтобы можно было просматривать все столбцы датафрейма:

- pd.options.display.max_columns = 100.
- Просмотрите первые 10 строк датафрейма df.

B [47]:
```
pd.options.display.max_columns = 100
df.head(10)
```

Out[47]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169	1.468177	-0.470401	0.207971	0.025791	0.403993	0.251412
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772	0.635558	0.463917	-0.114805	-0.183361	-0.145783	-0.069083
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946	2.345865	-2.890083	1.109969	-0.121359	-2.261857	0.524980
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924	-0.631418	-1.059647	-0.684093	1.965775	-1.232622	-0.208038
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670	0.175121	-0.451449	-0.237033	-0.038195	0.803487	0.408542
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	-0.371407	1.341262	0.359894	-0.358091	-0.137134	0.517617	0.401726	-0.058133	0.068653	-0.033194	0.084968
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	-0.099254	-1.416907	-0.153826	-0.751063	0.167372	0.050144	-0.443587	0.002821	-0.611987	-0.045575	-0.219633
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	1.249376	-0.619468	0.291474	1.757964	-1.323865	0.686133	-0.076127	-1.222127	-0.358222	0.324505	-0.156742
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	-0.410430	-0.705117	-0.110452	-0.286254	0.074355	-0.328783	-0.210077	-0.499768	0.118765	0.570328	0.052736
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	-0.366846	1.017614	0.836390	1.006844	-0.443523	0.150219	0.739453	-0.540980	0.476677	0.451773	0.203711

- Создайте датафрейм X из датафрейма df, исключив столбец Class.

B [48]:
```
X = df.drop("Class", axis=1)
X.head()
```

Out[48]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16	V17	V18	V19	V20
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600	-0.617801	-0.991390	-0.311169	1.468177	-0.470401	0.207971	0.025791	0.403993	0.251412
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612727	1.065235	0.489095	-0.143772	0.635558	0.463917	-0.114805	-0.183361	-0.145783	-0.069083
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624501	0.066084	0.717293	-0.165946	2.345865	-2.890083	1.109969	-0.121359	-2.261857	0.524980
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226487	0.178228	0.507757	-0.287924	-0.631418	-1.059647	-0.684093	1.965775	-1.232622	-0.208038
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822843	0.538196	1.345852	-1.119670	0.175121	-0.451449	-0.237033	-0.038195	0.803487	0.408542

- Создайте объект Series под названием y из столбца Class.

В [49]:
```
#y = pd.Series(df["Class"])
y = df["Class"]
print(type(y))  # Tun
print(y.unique())  # уникальные значения
y.head()
```

```
<class 'pandas.core.series.Series'>
[0 1]
```

Out[49]:
```
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int64
```

В [50]:
```
# pd.Series({c: df[c].unique() for c in df}) # уникальные значения
```

Разбейте X и y на тренировочный и тестовый наборы данных при помощи функции train_test_split, используя аргументы:

- test_size=0.3,
- random_state=100,
- stratify=y.

У вас должны получиться объекты X_train, X_test, y_train и y_test.

В [51]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=100, stratify=y)
```

Просмотрите информацию о их форме.

В [52]:
```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[52]:
```
((199364, 30), (85443, 30), (199364,), (85443,))
```

Для поиска по сетке параметров задайте такие параметры:

- parameters = [{'n_estimators': [10, 15],
- 'max_features': np.arange(3, 5),
- 'max_depth': np.arange(4, 7)}]

В [53]:
```
parameters = [{'n_estimators': [10, 15],
               'max_features': np.arange(3, 5),
               'max_depth': np.arange(4, 7)}]
```

Создайте модель GridSearchCV со следующими аргументами:

- estimator=RandomForestClassifier(random_state=100),
- param_grid=parameters,
- scoring='roc_auc',
- cv=3.

В [54]:
```
clf = GridSearchCV(
    estimator=RandomForestClassifier(random_state=100),
    param_grid=parameters,
    scoring='roc_auc',
    cv=3,
)
```

Обучите модель на тренировочном наборе данных (может занять несколько минут).

В [55]:
```
clf.fit(X_train, y_train)
```

Out[55]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=100),
             param_grid=[{'max_depth': array([4, 5, 6]),
                          'max_features': array([3, 4]),
                          'n_estimators': [10, 15]}],
             scoring='roc_auc')
```

Просмотрите параметры лучшей модели с помощью атрибута best_params_.

В [56]:
```
clf.best_params_
```

Out[56]:
```
{'max_depth': 6, 'max_features': 3, 'n_estimators': 15}
```

Предскажите вероятности классов с помощью полученнной модели и метода predict_proba.

В [57]:
```
# model.predict_proba() - выдать «степень уверенности» в ответе (вероятность) - для некоторых моделей
?clf.predict_proba
```

В [58]:
```
# в 1-м столбце вероятность класса 0, во 2-м столбце вероятность класса 1.
y_pred_proba = clf.predict_proba(X_test)
print(y_pred_proba[:10])
```

```
[[9.99070828e-01 9.29171738e-04]
 [9.99704794e-01 2.95206364e-04]
 [9.99717846e-01 2.82154033e-04]
 [9.99717846e-01 2.82154033e-04]
 [9.99717846e-01 2.82154033e-04]
 [9.99717846e-01 2.82154033e-04]
 [9.99717846e-01 2.82154033e-04]
 [9.99717846e-01 2.82154033e-04]
 [9.99717846e-01 2.82154033e-04]
 [9.99717846e-01 2.82154033e-04]]
```

В [59]:
```
print(y_pred_proba[:, 0] + y_pred_proba[:, 1])
print(y_pred_proba[1,0])
print(y_pred_proba[0,1])
```

```
[1. 1. 1. ... 1. 1. 1.]
0.9997047936359688
0.0009291717381760071
```

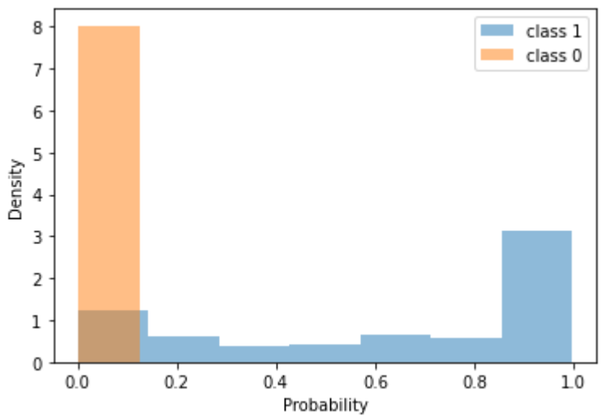
Из полученного результата (массив Numpy) выберите столбец с индексом 1 (вероятность класса 1) и запишите в массив y_pred_proba.

B [60]:
```
y_pred_proba = y_pred_proba[:, 1]
print(y_pred_proba[:10])
```

```
[0.00092917 0.00029521 0.00028215 0.00028215 0.00028215 0.00028215
 0.00028215 0.00028215 0.00028215 0.00028215]
```

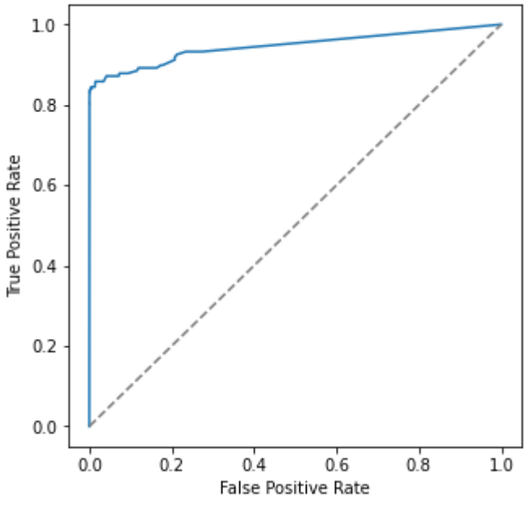
B [61]:
```
from matplotlib import pyplot as plt
plt.hist(y_pred_proba[y_test == 1], bins=7, density=True, label='class 1', alpha=0.5)
plt.hist(y_pred_proba[y_test == 0], bins=7, density=True, label='class 0', alpha=0.5)

plt.xlabel("Probability")
plt.ylabel("Density")

plt.legend()
```

Out[61]: <matplotlib.legend.Legend at 0x69a4a2fd60>



B [62]:
```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba, pos_label=1)

thresholds

th = thresholds[3]
th

print(fpr[3])
print(tpr[3])

# Теперь ROC-кривая просто строится по точкам, которые получаются, если FPR откладывать по оси x, а TPR - по оси y.
# Чем больше площадь под этой кривой, тем больше разделяющаяы способность этих моделей.:

plt.rcParams['figure.figsize'] = 5, 5

plt.plot(fpr, tpr)
plt.plot([0, 1], [0, 1], color='grey', linestyle='dashed')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve')

plt.show()
```

```
0.0
0.02702702702702703
```



Из модуля sklearn.metrics импортируйте метрику roc_auc_score.

B [63]:
```
from sklearn.metrics import roc_auc_score
```

Вычислите AUC на тестовых данных и сравните с результатом,полученным на тренировочных данных, используя в качестве аргументовмассивы y_test и y_pred_proba.

В [64]:
```
# Часто используемой метрикой является метрика AUC или Area Under Curve (площадь под кривой).
# Часто в качестве кривой используется кривая ROC.
# Мтрика ROC AUC - это площадь под кривой ROC.
# Для константного классификатора эта метрика равна  0.5 , поэтому для хороших классификаторов она должна быть между  0.5  и  1 .

roc_auc_score(y_test, y_pred_proba)
```

Out[64]:
```
0.9462664156037156
```

**Дополнительные задания:

1. Загрузите датасет Wine из встроенных датасетов sklearn.datasets с помощью функции load_wine в переменную data.

В [65]:
```
from sklearn.datasets import load_wine
data = load_wine()
data.keys()  # Смотрим содержимое dataseta
```

Out[65]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])
```

2. Полученный датасет не является датафреймом. Это структура данных, имеющая ключи аналогично словарю. Просмотрите тип данных этой структуры данных и создайте список data_keys, содержащий ее ключи.

В [66]:
```
print(type(data))
#print(data["DESCR"])
data.keys()
```

```
<class 'sklearn.utils.Bunch'>
```

Out[66]:
```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names'])
```

В [67]:
```
data_keys = data["feature_names"]
data_keys
```

Out[67]:
```
['alcohol',
 'malic_acid',
 'ash',
 'alcalinity_of_ash',
 'magnesium',
 'total_phenols',
 'flavanoids',
 'nonflavanoid_phenols',
 'proanthocyanins',
 'color_intensity',
 'hue',
 'od280/od315_of_diluted_wines',
 'proline']
```

3. Просмотрите данные, описание и названия признаков в датасете. Описание нужно вывести в виде привычного, аккуратно оформленного текста, без обозначений переноса строки, но с самими переносами и т.д.

In [68]:
```
#Просмотрите данные
data.data
```

Out[68]:
```
array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00, 3.920e+00,
        1.065e+03],
       [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
        1.050e+03],
       [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
        1.185e+03],
       ...,
       [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
        8.350e+02],
       [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
        8.400e+02],
       [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
        5.600e+02]])
```

In [69]:
```
data.data.shape
```

Out[69]:
```
(178, 13)
```

In [70]:
```
#описание и названия признаков в датасете. Описание нужно вывести в виде привычного, аккуратно
#оформленного текста, без обозначений переноса строки, но с самими переносами и т. д.
for line in data.DESCR.split('\n'):
    print(line)
```

```
.. _wine_dataset:

Wine recognition dataset
------------------------

**Data Set Characteristics:**

    :Number of Instances: 178 (50 in each of three classes)
    :Number of Attributes: 13 numeric, predictive attributes and the class
    :Attribute Information:
                - Alcohol
                - Malic acid
                - Ash
                - Alcalinity of ash
                - Magnesium
                - Total phenols
                - Flavanoids
                - Nonflavanoid phenols
                - Proanthocyanins
                - Color intensity
                - Hue
                - OD280/OD315 of diluted wines
                - Proline

    - class:
            - class_0
            - class_1
            - class_2

:Summary Statistics:

============================== ==== ===== ======= =====
                                Min   Max   Mean     SD
============================== ==== ===== ======= =====
Alcohol:                      11.0  14.8    13.0   0.8
Malic Acid:                   0.74  5.80    2.34  1.12
Ash:                          1.36  3.23    2.36  0.27
Alcalinity of Ash:            10.6  30.0    19.5   3.3
Magnesium:                    70.0 162.0    99.7  14.3
Total Phenols:                0.98  3.88    2.29  0.63
Flavanoids:                   0.34  5.08    2.03  1.00
Nonflavanoid Phenols:         0.13  0.66    0.36  0.12
Proanthocyanins:              0.41  3.58    1.59  0.57
Colour Intensity:              1.3  13.0     5.1   2.3
Hue:                          0.48  1.71    0.96  0.23
OD280/OD315 of diluted wines: 1.27  4.00    2.61  0.71
Proline:                       278  1680     746   315
============================== ==== ===== ======= =====

:Missing Attribute Values: None
:Class Distribution: class_0 (59), class_1 (71), class_2 (48)
:Creator: R.A. Fisher
:Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
:Date: July, 1988
```

This is a copy of UCI ML Wine recognition datasets.
https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data (https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data)

The data is the results of a chemical analysis of wines grown in the same
region in Italy by three different cultivators. There are thirteen different
measurements taken for different constituents found in the three types of
wine.

Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository
[https://archive.ics.uci.edu/ml]. Irvine, CA: University of California,
School of Information and Computer Science.

.. topic:: References

  (1) S. Aeberhard, D. Coomans and O. de Vel,
  Comparison of Classifiers in High Dimensional Settings,
  Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Technometrics).

  The data was used with many others for comparing various
  classifiers. The classes are separable, though only RDA
has achieved 100% correct classification.
(RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
(All results using the leave-one-out technique)

(2) S. Aeberhard, D. Coomans and O. de Vel,
"THE CLASSIFICATION PERFORMANCE OF RDA"
Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
Mathematics and Statistics, James Cook University of North Queensland.
(Also submitted to Journal of Chemometrics).

B [71]:
```
print(data["DESCR"])
```

```
.. _wine_dataset:

Wine recognition dataset
------------------------

**Data Set Characteristics:**

    :Number of Instances: 178 (50 in each of three classes)
    :Number of Attributes: 13 numeric, predictive attributes and the class
    :Attribute Information:
                - Alcohol
                - Malic acid
                - Ash
                - Alcalinity of ash
                - Magnesium
                - Total phenols
                - Flavanoids
                - Nonflavanoid phenols
                - Proanthocyanins
                - Color intensity
                - Hue
                - OD280/OD315 of diluted wines
                - Proline

    - class:
            - class_0
            - class_1
            - class_2

    :Summary Statistics:

    ============================== ==== ===== ======= =====
                                    Min   Max   Mean     SD
    ============================== ==== ===== ======= =====
    Alcohol:                      11.0  14.8    13.0   0.8
    Malic Acid:                   0.74  5.80    2.34  1.12
    Ash:                          1.36  3.23    2.36  0.27
    Alcalinity of Ash:            10.6  30.0    19.5   3.3
    Magnesium:                    70.0 162.0    99.7  14.3
    Total Phenols:                0.98  3.88    2.29  0.63
    Flavanoids:                   0.34  5.08    2.03  1.00
    Nonflavanoid Phenols:         0.13  0.66    0.36  0.12
    Proanthocyanins:              0.41  3.58    1.59  0.57
    Colour Intensity:              1.3  13.0     5.1   2.3
    Hue:                          0.48  1.71    0.96  0.23
    OD280/OD315 of diluted wines: 1.27  4.00    2.61  0.71
    Proline:                       278  1680     746   315
    ============================== ==== ===== ======= =====

    :Missing Attribute Values: None
    :Class Distribution: class_0 (59), class_1 (71), class_2 (48)
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML Wine recognition datasets.
https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data (https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data)
```

```
The data is the results of a chemical analysis of wines grown in the same
region in Italy by three different cultivators. There are thirteen different
measurements taken for different constituents found in the three types of
wine.

Original Owners:

Forina, M. et al, PARVUS -
An Extendible Package for Data Exploration, Classification and Correlation.
Institute of Pharmaceutical and Food Analysis and Technologies,
Via Brigata Salerno, 16147 Genoa, Italy.

Citation:

Lichman, M. (2013). UCI Machine Learning Repository
[https://archive.ics.uci.edu/ml]. Irvine, CA: University of California,
School of Information and Computer Science.

.. topic:: References

  (1) S. Aeberhard, D. Coomans and O. de Vel,
  Comparison of Classifiers in High Dimensional Settings,
  Tech. Rep. no. 92-02, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Technometrics).

  The data was used with many others for comparing various
  classifiers. The classes are separable, though only RDA
  has achieved 100% correct classification.
  (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN 96.1% (z-transformed data))
  (All results using the leave-one-out technique)

  (2) S. Aeberhard, D. Coomans and O. de Vel,
  "THE CLASSIFICATION PERFORMANCE OF RDA"
  Tech. Rep. no. 92-01, (1992), Dept. of Computer Science and Dept. of
  Mathematics and Statistics, James Cook University of North Queensland.
  (Also submitted to Journal of Chemometrics).
```

4. Сколько классов содержит целевая переменная датасета? Выведите названия классов.

B [72]:
```
print(np.unique(data["target"]).shape[0])
data_unique_rows = np.unique(data["target"], axis=0)
data_unique_rows
```

```
3
```

Out[72]:
```
array([0, 1, 2])
```

B [73]:
```
#Выведите названия классов.
data["target_names"]
```

Out[73]:
```
array(['class_0', 'class_1', 'class_2'], dtype='<U7')
```

5. На основе данных датасета (они содержатся в двумерном массиве Numpy) и названий признаков создайте датафрейм под названием X.

B [74]:
```
X = pd.DataFrame(data.data, columns=feature_names)

X.head()
```

Out[74]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0

6. Выясните размер датафрейма X и установитпе, имеются ли в нем пропущенные значения.

B [75]:
```
X.shape
```

Out[75]:
```
(178, 13)
```

B [76]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 13 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     178 non-null    float64
 1   ZN       178 non-null    float64
 2   INDUS    178 non-null    float64
 3   CHAS     178 non-null    float64
 4   NOX      178 non-null    float64
 5   RM       178 non-null    float64
 6   AGE      178 non-null    float64
 7   DIS      178 non-null    float64
 8   RAD      178 non-null    float64
 9   TAX      178 non-null    float64
 10  PTRATIO  178 non-null    float64
 11  B        178 non-null    float64
 12  LSTAT    178 non-null    float64
dtypes: float64(13)
memory usage: 18.2 KB
```

B [77]:
```
X.isnull().astype("int").sum()
```

Out[77]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
dtype: int64
```

7. Добавьте в датафрейм поле с классами вин в виде чисел, имеющих тип данных numpy.int64. Название поля - 'target'.

B [78]:
```
X["target"]=data["target"].astype(np.int64)
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     178 non-null    float64
 1   ZN       178 non-null    float64
 2   INDUS    178 non-null    float64
 3   CHAS     178 non-null    float64
 4   NOX      178 non-null    float64
 5   RM       178 non-null    float64
 6   AGE      178 non-null    float64
 7   DIS      178 non-null    float64
 8   RAD      178 non-null    float64
 9   TAX      178 non-null    float64
 10  PTRATIO  178 non-null    float64
 11  B        178 non-null    float64
 12  LSTAT    178 non-null    float64
 13  target   178 non-null    int64  
dtypes: float64(13), int64(1)
memory usage: 19.6 KB
```

B [79]:
```
X.head()
```

Out[79]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	target
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0

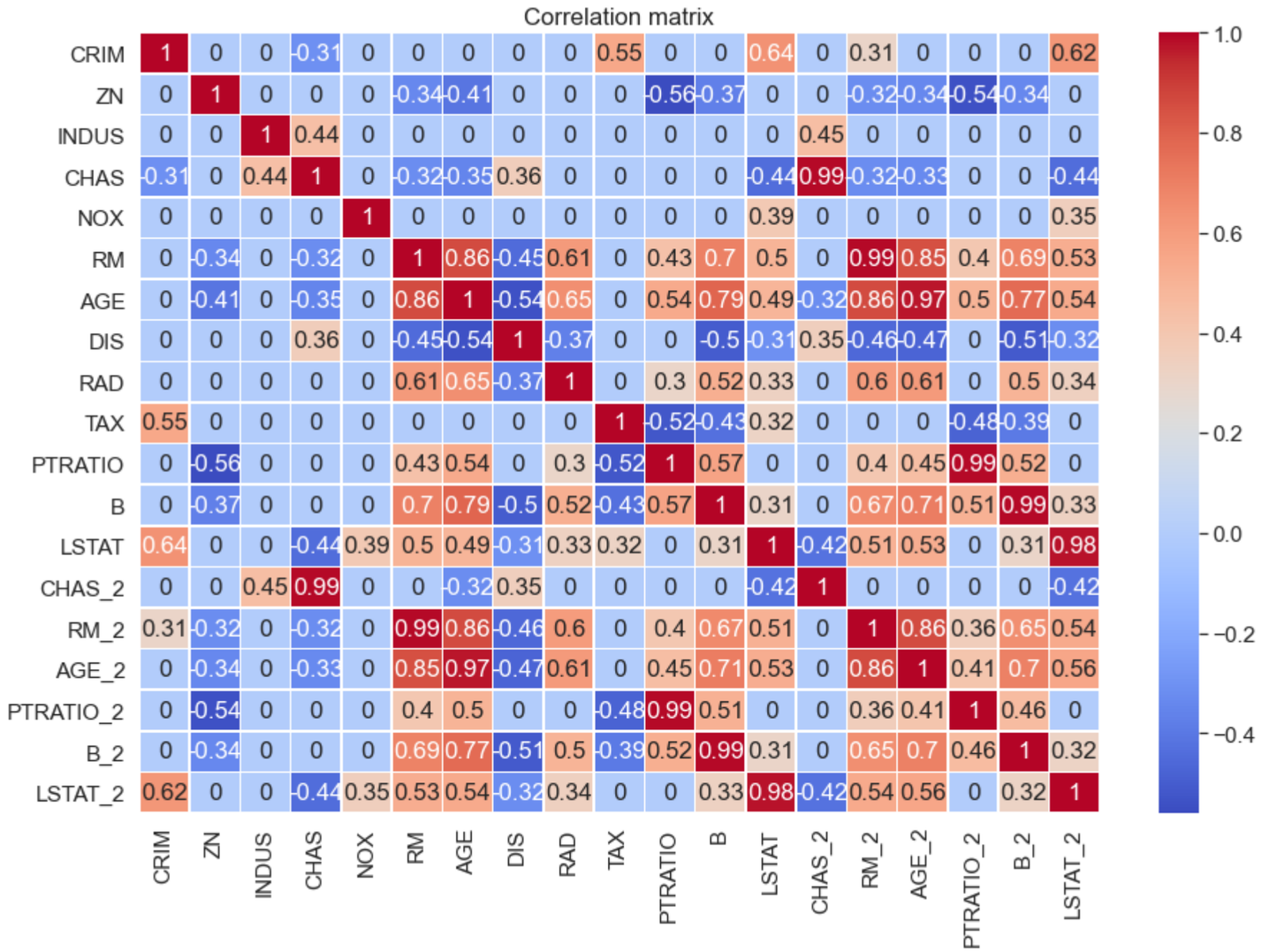
8. Постройте матрицу корреляций для всех полей X. Дайте полученному датафрейму названиеX_corr.

B [88]:
```
# Матрица корреляций
# Показывает линейную связь между переменными
# Изменяется от -1 до 1
# Коррелиция - мера только линейной связи

import matplotlib.pyplot as plt
import seaborn as sns # Основан на matplotlib но графики более красивые

X_corr=X.corr()
X_corr
```

Out[88]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	CHAS_2	RM_2	AGE_2	PTRATIO_2	B_2	LSTAT_2
CRIM	1.000000	0.094397	0.211545	-0.310235	0.270798	0.289101	0.236815	-0.155929	0.136698	0.546364	-0.071747	0.072343	0.643720	-0.292040	0.308655	0.288629	-0.088187	0.095814	0.621819
ZN	0.094397	1.000000	0.164045	0.288500	-0.054575	-0.335167	-0.411007	0.292977	-0.220746	0.248985	-0.561296	-0.368710	-0.192011	0.270947	-0.318857	-0.342771	-0.536999	-0.339016	-0.212272
INDUS	0.211545	0.164045	1.000000	0.443367	0.286587	0.128980	0.115077	0.186230	0.009652	0.258887	-0.074667	0.003911	0.223626	0.454174	0.129180	0.167245	-0.072469	0.018279	0.204804
CHAS	-0.310235	0.288500	0.443367	1.000000	-0.083333	-0.321113	-0.351370	0.361922	-0.197327	0.018732	-0.273955	-0.276769	-0.440597	0.991672	-0.323454	-0.333522	-0.239637	-0.269570	-0.436734
NOX	0.270798	-0.054575	0.286587	-0.083333	1.000000	0.214401	0.195784	-0.256294	0.236441	0.199950	0.055398	0.066004	0.393351	-0.064474	0.227581	0.213365	0.043488	0.084705	0.346545
RM	0.289101	-0.335167	0.128980	-0.321113	0.214401	1.000000	0.864564	-0.449935	0.612413	-0.055136	0.433681	0.699949	0.498115	-0.287657	0.989802	0.849072	0.396113	0.685587	0.525916
AGE	0.236815	-0.411007	0.115077	-0.351370	0.195784	0.864564	1.000000	-0.537900	0.652692	-0.172379	0.543479	0.787194	0.494193	-0.318738	0.855460	0.968530	0.496377	0.771087	0.535204
DIS	-0.155929	0.292977	0.186230	0.361922	-0.256294	-0.449935	-0.537900	1.000000	-0.365845	0.139057	-0.262640	-0.503270	-0.311385	0.350483	-0.456062	-0.469998	-0.227167	-0.512948	-0.319794
RAD	0.136698	-0.220746	0.009652	-0.197327	0.236441	0.612413	0.652692	-0.365845	1.000000	-0.025250	0.295544	0.519067	0.330417	-0.176397	0.600684	0.614290	0.282531	0.501980	0.344608
TAX	0.546364	0.248985	0.258887	0.018732	0.199950	-0.055136	-0.172379	0.139057	-0.025250	1.000000	-0.521813	-0.428815	0.316100	0.019552	-0.015870	-0.054448	-0.478239	-0.391859	0.287844
PTRATIO	-0.071747	-0.561296	-0.074667	-0.273955	0.055398	0.433681	0.543479	-0.262640	0.295544	-0.521813	1.000000	0.565468	0.236183	-0.251933	0.395089	0.452204	0.988694	0.516669	0.273273
B	0.072343	-0.368710	0.003911	-0.276769	0.066004	0.699949	0.787194	-0.503270	0.519067	-0.428815	0.565468	1.000000	0.312761	-0.256762	0.666172	0.708518	0.506046	0.991928	0.332799
LSTAT	0.643720	-0.192011	0.223626	-0.440597	0.393351	0.498115	0.494193	-0.311385	0.330417	0.316100	0.236183	0.312761	1.000000	-0.420321	0.512220	0.525429	0.201429	0.310554	0.981708
CHAS_2	-0.292040	0.270947	0.454174	0.991672	-0.064474	-0.287657	-0.318738	0.350483	-0.176397	0.019552	-0.251933	-0.256762	-0.420321	1.000000	-0.289922	-0.294305	-0.219307	-0.248779	-0.416590
RM_2	0.308655	-0.318857	0.129180	-0.323454	0.227581	0.989802	0.855460	-0.456062	0.600684	-0.015870	0.395089	0.666172	0.512220	-0.289922	1.000000	0.856830	0.358317	0.654578	0.541399
AGE_2	0.288629	-0.342771	0.167245	-0.333522	0.213365	0.849072	0.968530	-0.469998	0.614290	-0.054448	0.452204	0.708518	0.525429	-0.294305	0.856830	1.000000	0.408174	0.699305	0.564023
PTRATIO_2	-0.088187	-0.536999	-0.072469	-0.239637	0.043488	0.396113	0.496377	-0.227167	0.282531	-0.478239	0.988694	0.506046	0.201429	-0.219307	0.358317	0.408174	1.000000	0.459163	0.240910
B_2	0.095814	-0.339016	0.018279	-0.269570	0.084705	0.685587	0.771087	-0.512948	0.501980	-0.391859	0.516669	0.991928	0.310554	-0.248779	0.654578	0.699305	0.459163	1.000000	0.324805
LSTAT_2	0.621819	-0.212272	0.204804	-0.436734	0.346545	0.525916	0.535204	-0.319794	0.344608	0.287844	0.273273	0.332799	0.981708	-0.416590	0.541399	0.564023	0.240910	0.324805	1.000000

B [89]:
```
plt.figure(figsize = (15,10))

sns.set(font_scale=1.4)

X_corr = X.corr()
X_corr = np.round(X_corr, 2)
X_corr[np.abs(X_corr) < 0.3] = 0

sns.heatmap(X_corr, annot=True, linewidths=.5, cmap='coolwarm')

plt.title('Correlation matrix')
plt.show()
```



9. Создайте список high_corr из признаков, корреляция которых с полем target по абсолютному значению превышает 0.5 (причем, само поле target не должно входить в этот список).

B [81]:
```
high_corr=X_corr["target"]
high_corr=high_corr[np.abs(high_corr)>0.5].drop("target", axis=0)
high_corr=list(high_corr.index)
high_corr
```

Out[81]:
```
['CHAS', 'RM', 'AGE', 'PTRATIO', 'B', 'LSTAT']
```

10. Удалите из датафрейма X поле с целевой переменной.

B [82]:
```
X = X.drop("target", axis=1)
X.head()
```

Out[82]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0

Для всех признаков, названия которых содержатся в списке high_corr, вычислите квадрат их значений и добавьте в датафрейм X соответствующие поля с суффиксом '_2', добавленного к первоначальному названию признака. Итоговый датафрейм должен содержать все поля, которые, были в нем изначально, а также поля с признаками из списка high_corr, возведенными в квадрат.

B [86]:
```
for i in high_corr:
    print(i + "_2")
    X[i + "_2"] = X[i]**2
X.head()
```

```
CHAS_2
RM_2
AGE_2
PTRATIO_2
B_2
LSTAT_2
```

Out[86]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	CHAS_2	RM_2	AGE_2	PTRATIO_2	B_2	LSTAT_2
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	243.36	7.8400	9.3636	1.0816	15.3664	1134225.0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	125.44	7.0225	7.6176	1.1025	11.5600	1102500.0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	345.96	7.8400	10.4976	1.0609	10.0489	1404225.0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	282.24	14.8225	12.1801	0.7396	11.9025	2190400.0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	441.00	7.8400	7.2361	1.0816	8.5849	540225.0

Выведите описание полей датафрейма X с помощью метода describe.

B [87]:
```
X.describe()
```

Out[87]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	CHAS_2	RM_2	AGE_2	PTRATIO_2	
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.00(
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	5.058090	0.957449	2.611685	746.893258	391.142865	5.657030	5.110049	0.968661	7.32
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.572359	2.318286	0.228572	0.709990	314.907474	133.671775	2.936294	4.211441	0.443798	3.58
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.280000	0.480000	1.270000	278.000000	112.360000	0.960400	0.115600	0.230400	1.61
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.250000	3.220000	0.782500	1.937500	500.500000	295.840000	3.036325	1.452100	0.612325	3.75
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.555000	4.690000	0.965000	2.780000	673.500000	380.250000	5.546050	4.558250	0.931250	7.72
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	6.200000	1.120000	3.170000	985.000000	462.250000	7.840000	8.265700	1.254400	10.04
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000	4.000000	1680.000000	900.000000	15.054400	25.806400	2.924100	16.00

B [ ]: