Практическое задание 3

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 на 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 предсказаний с помощью r2_score из модуля sklearn.metrics.

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_boston
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
from sklearn.metrics import roc_auc_score
import seaborn as sns
pd.options.display.max_columns = 100
```

```
C:\ProgramData\Anaconda3\lib\site-packages\statsmodels\tools\_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the pu
blic API at pandas.testing instead.
  import pandas.util.testing as tm
```

In [2]:

```
boston = load_boston()
boston.keys()
```

Out[2]:

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

In [3]:

```
X = pd.DataFrame(boston.data, columns = boston.feature_names)
X.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
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
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
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
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
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

In [4]:

```
y = pd.DataFrame(boston.target, columns = ["price"])
y.head()
```

Out[4]:

	price
0	24.0
1	21.6
2	34.7
3	33.4
4	36.2

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=4
```

In [6]:

```
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
```

В документации указано, что y_pred, идет вторым аргументов - sklearn.metrics.r2_score(y_true, y_pred, *, sample_weight=None, multioutput='uniform_average')

In [7]:

```
r2_score(y_test, y_pred)
```

Out[7]:

```
0.7112260057484918
```

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.

In [8]:

```
model = RandomForestRegressor(n_estimators = 1000, max_depth=12, random_state=42)
model.fit(X_train, y_train.values[:, 0])
y_pred = model.predict(X_test)
```

In [9]:

```
r2_score(y_test, y_pred)
```

Out[9]:

```
0.8749965273218174
```

Вывод: RandomForestRegressor с данным датасетом работает лучше чем LinearRegression

***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.

In [10]:

```
importances = model.feature_importances_
importances
```

Out[10]:

```
array([0.03211748, 0.00154999, 0.0070941 , 0.0011488 , 0.01436832,
       0.40270459, 0.01424477, 0.06403265, 0.00496762, 0.01169177,
       0.01808961, 0.0123114 , 0.41567892])
```

In [11]:

```
importances = pd.DataFrame(importances, index = boston.feature_names, columns = ["featu
importances
```

Out[11]:

	feature_importances
CRIM	0.032117
ZN	0.001550
INDUS	0.007094
CHAS	0.001149
NOX	0.014368
RM	0.402705
AGE	0.014245
DIS	0.064033
RAD	0.004968
TAX	0.011692
PTRATIO	0.018090
B	0.012311
LSTAT	0.415679

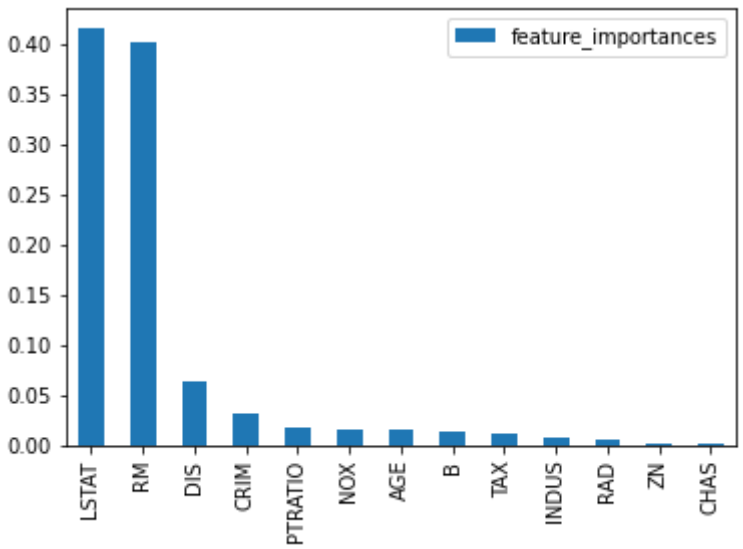
In [12]:

```
print("Сумма всех показателей важности = ", np.sum(importances)[0])
```

```
Сумма всех показателей важности =  0.9999999999999999
```

In [13]:

```
importances.sort_values("feature_importances", ascending = False).plot(kind = "bar")
plt.show()
```



Два признака, показывающие наибольшую важность:

LSTAT - % более низкиого статуса населения
RM - среднее количество комнат в одном жилом помещении

****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.

In [14]:

```
# Все библиотеки импортируются вначале задания
df = pd.read_csv("creditcard.csv")
df
```

Out[14]:

	Time	V1	V2	V3	V4	V5	V6	V7
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941
...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

284807 rows × 31 columns

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

In [15]:

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

Out[15]:

```
0    0.998273
1    0.001727
Name: Class, dtype: float64
```

Видим, что доля класса "1" составляет менее 0,02%

Используя метод info, проверьте, все ли столбцы содержат числовые данные и нет ли в них пропусков.

In [16]:

```
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
```

Видим, что пропусков нет ни в одной из 284807 строк

Примените следующую настройку, чтобы можно было просматривать все столбцы датафрейма: pd.options.display.max_columns = 100. Просмотрите первые 10 строк датафрейма df.

In [17]:

```
# Настройку применил в начале задания
df.head(10)
```

Out[17]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539

Создайте датафрейм X из датафрейма df, исключив столбец Class.

In [18]:

```
X = df.drop("Class", axis = 1)
X
```

Out[18]:

	Time	V1	V2	V3	V4	V5	V6	V7
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941
...	...	...	...	...	...	...	...	...
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006

284807 rows × 30 columns

Создайте объект Series под названием y из столбца Class.

In [19]:

```
y = df["Class"]
y
```

Out[19]:

```
0         0
1         0
2         0
3         0
4         0
         ..
284802    0
284803    0
284804    0
284805    0
284806    0
Name: Class, Length: 284807, dtype: int64
```

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.

In [20]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
```

Просмотрите информацию о их форме.

In [21]:

```
for set, name in zip([X_train, X_test, y_train, y_test], ["X_train", "X_test", "y_trai
    print(name, set.shape)
```

```
X_train (199364, 30)
X_test (85443, 30)
y_train (199364,)
y_test (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)}]
Создайте модель GridSearchCV со следующими аргументами:
estimator=RandomForestClassifier(random_state=100),
param_grid=parameters,
scoring='roc_auc',
cv=3.

In [22]:

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

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).

In [23]:

```
1 clf.fit(X_train, y_train)
```

Out[23]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=No
ne,
                                              criterion='gini', max_depth=Non
e,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=Non
e,
                                              oob_score=False, random_state=1
00,
                                              verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': array([4, 5, 6]),
                         'max_features': array([3, 4]),
                         'n_estimators': [10, 15]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

In [24]:

```
clf
```

Out[24]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=No
ne,
                                              criterion='gini', max_depth=Non
e,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=Non
e,
                                              oob_score=False, random_state=1
00,
                                              verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': array([4, 5, 6]),
                         'max_features': array([3, 4]),
                         'n_estimators': [10, 15]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

In [25]:

```
cv_results = pd.DataFrame(clf.cv_results_)
cv_results.columns
```

Out[25]:

```
Index(['mean_fit_time', 'std_fit_time', 'mean_score_time', 'std_score_time',
       'param_max_depth', 'param_max_features', 'param_n_estimators', 'param
s',
       'split0_test_score', 'split1_test_score', 'split2_test_score',
       'mean_test_score', 'std_test_score', 'rank_test_score'],
      dtype='object')
```

In [26]:

```
param_columns = [
    column
    for column in cv_results.columns
    if column.startswith('param_')
]

score_columns = ['mean_test_score']

cv_results = (cv_results[param_columns + score_columns]
              .sort_values(by=score_columns, ascending=False))

cv_results.head(10)
```

Out[26]:

	param_max_depth	param_max_features	param_n_estimators	mean_test_score
9	6	3	15	0.965969
7	5	4	15	0.962598
6	5	4	10	0.960201
11	6	4	15	0.959975
10	6	4	10	0.956352
3	4	4	15	0.955851
8	6	3	10	0.954698
5	5	3	15	0.949443
4	5	3	10	0.949299
2	4	4	10	0.941082

Просмотрите параметры лучшей модели с помощью атрибута best_params_.

In [27]:

```
clf.best_params_
```

Out[27]:

```
{'max_depth': 6, 'max_features': 3, 'n_estimators': 15}
```

Предскажите вероятности классов с помощью полученнной модели и метода predict_proba.

In [28]:

```
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]]
```

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

In [29]:

```
y_pred_proba = y_pred_proba[:, 1]
```

Из модуля sklearn.metrics импортируйте метрику roc_auc_score.

roc_auc_score импортировал вначале задания

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

In [30]:

```
roc_auc_score(y_test, y_pred_proba)
```

Out[30]:

```
0.9462664156037156
```

In [31]:

```
y_pred_proba_train = clf.predict_proba(X_train)
```

In [32]:

```
roc_auc_score(y_train, y_pred_proba_train[:, 1])
```

Out[32]:

```
0.9703527882554751
```

Видим, что на тренировочных данных roc_auc_score несколько выше

***Дополнительные задания:**

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

In [33]:

```
data = load_wine()
```

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.

In [34]:

```
type(data)
```

Out[34]:

```
sklearn.utils.Bunch
```

In [35]:

```
data.keys()
```

Out[35]:

```
dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names'])
```

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 [36]:

```
# Данные
data.data
```

Out[36]:

```
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 [37]:

```
# Описание
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).

In [38]:

```
# Названия признаков в датасете
feature_names = data["feature_names"]
feature_names
```

Out[38]:

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

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

In [39]:

```
np.unique(data["target"])
```

Out[39]:

```
array([0, 1, 2])
```

Целевая переменная датасета содержит три класса - 0, 1 и 2

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

In [40]:

```
X = pd.DataFrame(data.data, columns = data.feature_names)
X.head()
```

Out[40]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavan
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	

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

In [41]:

```
1 X.shape
```

Out[41]:

```
(178, 13)
```

In [42]:

```
1 X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 13 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
dtypes: float64(13)
memory usage: 18.2 KB
```

Пропусков нет

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

In [43]:

```
1 X['target'] = np.int64(data.target)
```

In [44]:

```
1 X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
 11  od280/od315_of_diluted_wines  178 non-null    float64
 12  proline                       178 non-null    float64
 13  target                        178 non-null    int64  
dtypes: float64(13), int64(1)
memory usage: 19.6 KB
```

8). Постройте матрицу корреляций для всех полей X. Дайте полученному датафрейму название X_corr.

In [45]:

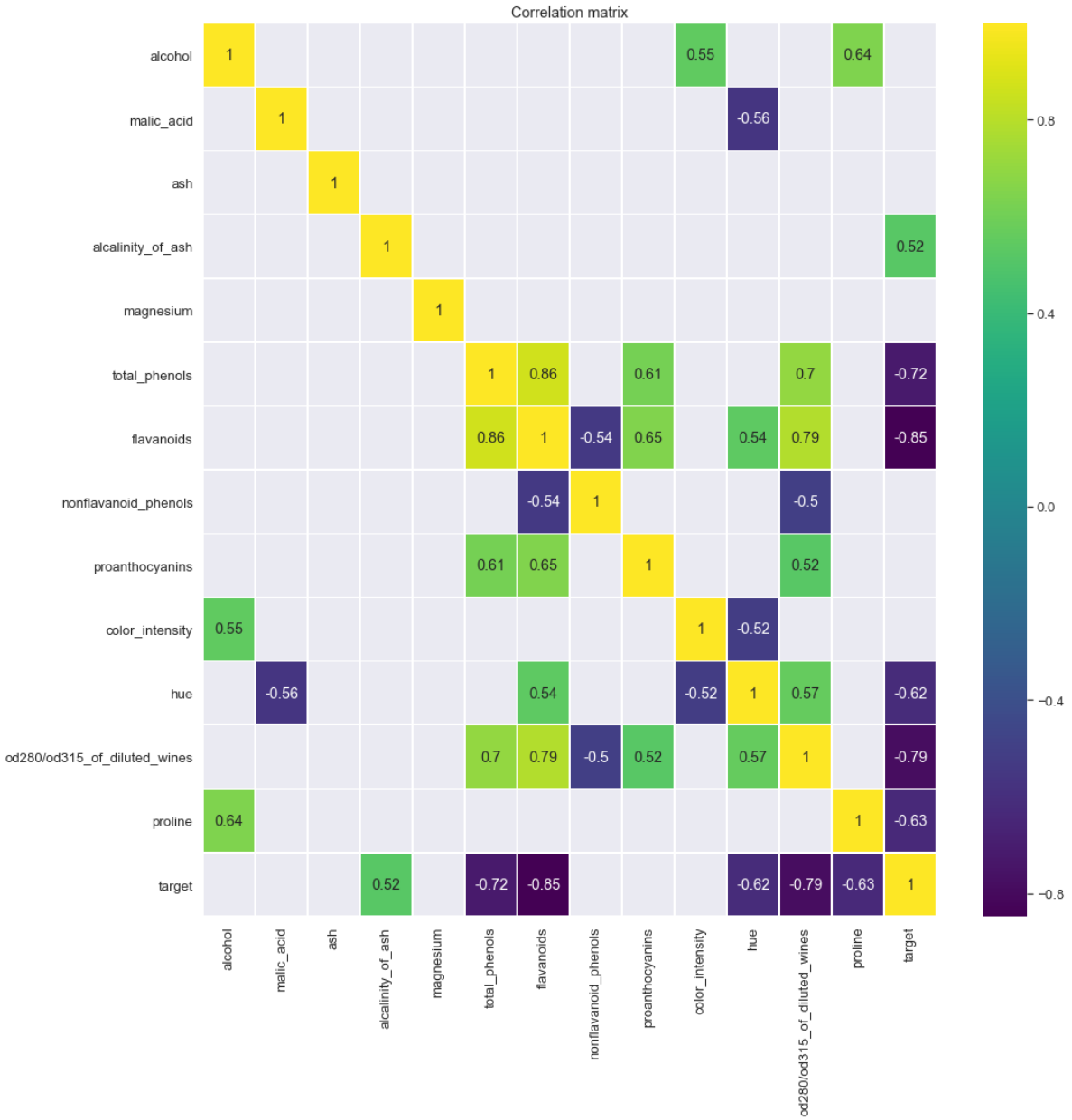
```
1 X_corr = X.corr()
```

In [46]:

```
plt.figure(figsize = (16,16))
sns.set(font_scale=1.2)

corr_matrix = X_corr
sns.heatmap(corr_matrix[abs(corr_matrix) > 0.5], annot=True, linewidths=.5, cmap='virid

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



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).

In [47]:

```
high_corr = [feature for feature in list(X_corr["target"][abs(X_corr["target"]) > 0.5]
high_corr
```

Out[47]:

```
['alcalinity_of_ash',
 'total_phenols',
 'flavanoids',
 'hue',
 'od280/od315_of_diluted_wines',
 'proline']
```

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.

In [48]:

```
X = X.drop("target", axis = 1)
X
```

Out[48]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflav
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	
...	...	...	...	...	...	...	...	
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	

178 rows × 13 columns

In [49]:

```
for col in high_corr:
    X[f"{col}_2"] = X[col]**2
```

In [50]:

```
X.describe()
```

Out[50]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavan
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080