# 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 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
import matplotlib.pyplot as plt
from sklearn.datasets import load_boston
```

In [2]:
```
boston = load_boston()
data = boston['data']
target = boston['target']
features = boston['feature_names']
X = pd.DataFrame(data, columns=features)
y = pd.DataFrame(target, columns=['price'])
X.head()
```

Out[2]:

| | 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 |

In [3]:
```
y.head()
```

Out[3]:

| | price |
|---|---|
| **0** | 24.0 |
| **1** | 21.6 |
| **2** | 34.7 |
| **3** | 33.4 |
| **4** | 36.2 |

In [4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

In [5]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

In [6]:
```
lr.fit(X_train, y_train)
```

Out[6]:
```
LinearRegression()
```

In [7]:
```
y_pred = lr.predict(X_test)
y_pred.shape
```

Out[7]:
```
(152, 1)
```

In [8]:
```
from sklearn.metrics import r2_score
```

In [9]:
```
r2_score(y_test, y_pred)
```

Out[9]:
```
0.711226005748491
```

**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 у 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 [10]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [11]:
```
model = RandomForestRegressor(n_estimators=1000, max_depth=12, random_state=42)
```

In [12]:
```
model.fit(X_train, y_train.values[:, 0])
```

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

In [13]:
```
y_pred = model.predict(X_test)
y_pred.shape
```

Out[13]:
```
(152,)
```

In [14]:
```
r2_score(y_test, y_pred)
```

Out[14]:
```
0.87472606157312
```

In [15]:
```
#Модель, построенная с использованием алгоритма случайного леса, показывает лучше результат, чем модель,
#построенная при помощи линейной регрессии. На это указывает большее значение метрики r2_score.
```

**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 [16]:
```
RandomForestRegressor?
```

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. Warning: impurity-based feature importances can be misleading for high cardinality features (many unique values). See :func:
$sk \leq arn.\ \in spection.per\mu tation_imp$ or $\tan ce$ as an alternative.

In [17]:
```
model.feature_importances_
```

Out[17]:
```
array([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])
```

In [18]:
```
X.columns
```

Out[18]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT'],
      dtype='object')
```

In [19]:
```
sorted(list(zip(X.columns, model.feature_importances_)), key=lambda t: t[1], reverse=True)[:2]
```

Out[19]:
```
[('LSTAT', 0.4158473181914483), ('RM', 0.4026817857034993)]
```

In [20]:
```
#Два наиболее важных признака - это LSTAT и RM
```

**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 аргументов массивы y_test и y_pred_proba.

In [21]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
```

In [22]:
```
df = pd.read_csv('creditcard.csv')
df.head()
```

Out[22]:

| | 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

In [23]:
```
df.Class.value_counts(normalize=True)
```

Out[23]:
```
0    0.998273
1    0.001727
Name: Class, dtype: float64
```

In [24]:
```
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
```

In [25]:
```
pd.options.display.max_columns = 100
```

In [26]:
```
df.head(10)
```

Out[26]:

| | 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 | -0 |
| **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 | -0 |
| **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 | 0 |
| **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 | -0 |
| **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 | -0 |
| **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 | -0 |
| **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 | -0 |
| **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 | 1 |
| **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 | -0 |
| **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 | -0 |

In [27]:
```
X = df.loc[:, 'Time':'Amount']
```

In [28]:
```
X.head()
```

Out[28]:

| | 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 | -0. |
| **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 | -0. |
| **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 | 0. |
| **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 | -0. |
| **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 | -0. |

In [29]:
```
y = df['Class']
y
```

Out[29]:
```
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
```

In [30]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=100, stratify=y)
```

In [31]:
```
X_train.shape
```

Out[31]:
```
(199364, 30)
```

In [32]:
```
X_test.shape
```

Out[32]:
```
(85443, 30)
```

In [33]:
```
y_train.shape
```

Out[33]:
```
(199364,)
```

In [34]:
```
y_test.shape
```

Out[34]:
```
(85443,)
```

In [35]:
```
parameters = [{'n_estimators': [10, 15], 'max_features': np.arange(3, 5), 'max_depth': np.arange(4, 7)}]
```

In [36]:
```
model = GridSearchCV(estimator=RandomForestClassifier(random_state=100), param_grid=parameters, scoring='roc_auc', cv=3)
```

In [37]:
```
model.fit(X_train, y_train)
```

Out[37]:
```
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')
```

In [38]:
```
model.best_params_
```

Out[38]:
```
{'max_depth': 6, 'max_features': 3, 'n_estimators': 15}
```

In [39]:
```
model.predict_proba(X_test)
```

Out[39]:
```
array([[9.99070828e-01, 9.29171738e-04],
       [9.99704794e-01, 2.95206364e-04],
       [9.99717846e-01, 2.82154033e-04],
       ...,
       [9.99717846e-01, 2.82154033e-04],
       [9.99317795e-01, 6.82204754e-04],
       [9.87539019e-01, 1.24609813e-02]])
```

In [40]:
```
y_pred_proba = model.predict_proba(X_test)[:, 1]
```

```
from sklearn.metrics import roc_auc_score
```

In [41]:
```
roc_auc_score(y_test, y_pred_proba) #для тестовых данных
```

Out[41]:
```
0.9462664156037156
```

In [42]:
```
roc_auc_score(y_train, model.predict_proba(X_train)[:, 1])#для тренировочных данных
```

Out[42]:
```
0.9703527882554751
```

**Дополнительные задания**

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

In [43]:
```
from sklearn.datasets import load_wine
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 [44]:
```
type(data)
```

Out[44]:
```
sklearn.utils.Bunch
```

In [45]:
```
data_keys = data.keys()
data_keys
```

Out[45]:
```
dict_keys(['data', 'target', 'frame', '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 [46]:
```
data['data']
```

Out[46]:
```
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 [47]:
```
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

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 [48]:
```
data['feature_names']
```

Out[48]:
```
['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 [49]:
```
data['target_names'].size #количество классов
```

Out[49]:
```
3
```

In [50]:
```
data['target_names']
```

Out[50]:
```
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.

In [51]:
```
X = pd.DataFrame(data=data['data'], columns=data['feature_names'])
X.head()
```

Out[51]:

| | alcohol | malic_acid | ash | alcalinity_of_ash | magnesium | total_phenols | flavanoids | nonflavanoid_phenols | proanthocyanins | color_intensity | hue | od280/od315_of_diluted_wines | proline |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 установите, имеются ли в нем пропущенные значения.

In [52]:
```
X.shape
```

Out[52]:
```
(178, 13)
```

In [53]:
```
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
```

In [54]:
```
#пропущенных значений нет
```

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

In [55]:
```
X['target'] = np.asarray(data['target'], dtype='int64')
```

In [56]:
```
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
```

In [57]:
```
X.head()
```

Out[57]:

| | alcohol | malic_acid | ash | alcalinity_of_ash | magnesium | total_phenols | flavanoids | nonflavanoid_phenols | proanthocyanins | color_intensity | hue | od280/od315_of_diluted_wines | proline | 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 название X_corr.

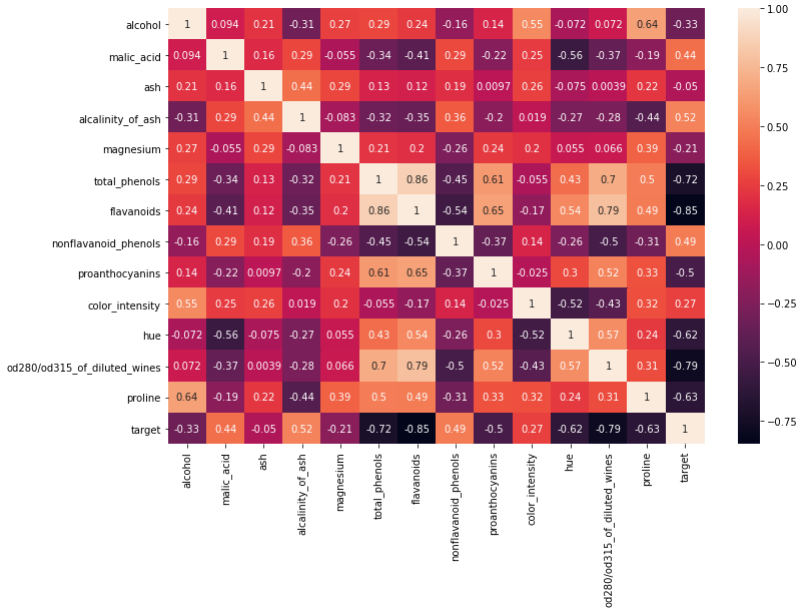
In [58]:
```
X_corr = X.corr()
X_corr
```

Out[58]:

| | alcohol | malic_acid | ash | alcalinity_of_ash | magnesium | total_phenols | flavanoids | nonflavanoid_phenols | proanthocyanins | color_intensity | hue | od280/od315_of_diluted_wines | prolin |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **alcohol** | 1.000000 | 0.094397 | 0.211545 | -0.310235 | 0.270798 | 0.289101 | 0.236815 | -0.155929 | 0.136698 | 0.546364 | -0.071747 | 0.072343 | 0.64372 |
| **malic_acid** | 0.094397 | 1.000000 | 0.164045 | 0.288500 | -0.054575 | -0.335167 | -0.411007 | 0.292977 | -0.220746 | 0.248985 | -0.561296 | -0.368710 | -0.19201 |
| **ash** | 0.211545 | 0.164045 | 1.000000 | 0.443367 | 0.286587 | 0.128980 | 0.115077 | 0.186230 | 0.009652 | 0.258887 | -0.074667 | 0.003911 | 0.22362 |
| **alcalinity_of_ash** | -0.310235 | 0.288500 | 0.443367 | 1.000000 | -0.083333 | -0.321113 | -0.351370 | 0.361922 | -0.197327 | 0.018732 | -0.273955 | -0.276769 | -0.44059 |
| **magnesium** | 0.270798 | -0.054575 | 0.286587 | -0.083333 | 1.000000 | 0.214401 | 0.195784 | -0.256294 | 0.236441 | 0.199950 | 0.055398 | 0.066004 | 0.39335 |
| **total_phenols** | 0.289101 | -0.335167 | 0.128980 | -0.321113 | 0.214401 | 1.000000 | 0.864564 | -0.449935 | 0.612413 | -0.055136 | 0.433681 | 0.699949 | 0.49811 |
| **flavanoids** | 0.236815 | -0.411007 | 0.115077 | -0.351370 | 0.195784 | 0.864564 | 1.000000 | -0.537900 | 0.652692 | -0.172379 | 0.543479 | 0.787194 | 0.49419 |
| **nonflavanoid_phenols** | -0.155929 | 0.292977 | 0.186230 | 0.361922 | -0.256294 | -0.449935 | -0.537900 | 1.000000 | -0.365845 | 0.139057 | -0.262640 | -0.503270 | -0.31138 |
| **proanthocyanins** | 0.136698 | -0.220746 | 0.009652 | -0.197327 | 0.236441 | 0.612413 | 0.652692 | -0.365845 | 1.000000 | -0.025250 | 0.295544 | 0.519067 | 0.33041 |
| **color_intensity** | 0.546364 | 0.248985 | 0.258887 | 0.018732 | 0.199950 | -0.055136 | -0.172379 | 0.139057 | -0.025250 | 1.000000 | -0.521813 | -0.428815 | 0.31610 |
| **hue** | -0.071747 | -0.561296 | -0.074667 | -0.273955 | 0.055398 | 0.433681 | 0.543479 | -0.262640 | 0.295544 | -0.521813 | 1.000000 | 0.565468 | 0.23618 |
| **od280/od315_of_diluted_wines** | 0.072343 | -0.368710 | 0.003911 | -0.276769 | 0.066004 | 0.699949 | 0.787194 | -0.503270 | 0.519067 | -0.428815 | 0.565468 | 1.000000 | 0.31276 |
| **proline** | 0.643720 | -0.192011 | 0.223626 | -0.440597 | 0.393351 | 0.498115 | 0.494193 | -0.311385 | 0.330417 | 0.316100 | 0.236183 | 0.312761 | 1.00000 |
| **target** | -0.328222 | 0.437776 | -0.049643 | 0.517859 | -0.209179 | -0.719163 | -0.847498 | 0.489109 | -0.499130 | 0.265668 | -0.617369 | -0.788230 | -0.63371 |

In [59]:
```
import seaborn as sn
plt.rcParams['figure.figsize'] = [12, 8]
sn.heatmap(X_corr, annot=True)
```

Out[59]: <AxesSubplot:>




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 [60]:
```
high_corr = np.array(X_corr[abs(X_corr['target'])>0.5].index)[:-1]
high_corr
```

Out[60]:
```
array(['alcalinity_of_ash', 'total_phenols', 'flavanoids', 'hue',
       'od280/od315_of_diluted_wines', 'proline'], dtype=object)
```

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 [61]:
```
del X['target']
```

In [62]:
```
X.head()
```

Out[62]:

| | alcohol | malic_acid | ash | alcalinity_of_ash | magnesium | total_phenols | flavanoids | nonflavanoid_phenols | proanthocyanins | color_intensity | hue | od280/od315_of_diluted_wines | proline |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 |

In [63]:
```
X[high_corr+'_2'] = X[high_corr]**2
X.head()
```

Out[63]:

| | alcohol | malic_acid | ash | alcalinity_of_ash | magnesium | total_phenols | flavanoids | nonflavanoid_phenols | proanthocyanins | color_intensity | hue | od280/od315_of_diluted_wines | proline | alcalinity_of_ash_2 | total_phenols_2 | flav |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | |
| **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 | |
| **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 | |
| **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 | |
| **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 | |

In [64]:
```
X.describe()
```

Out[64]:

| | alcohol | malic_acid | ash | alcalinity_of_ash | magnesium | total_phenols | flavanoids | nonflavanoid_phenols | proanthocyanins | color_intensity | hue | od280/od315_of_diluted_wines | proline | alcalinity_of_as |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 178.000 |
| **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 | 391.142 |
| **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 | 133.671 |
| **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 | 112.360 |
| **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 | 295.840 |
| **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 | 380.250 |
| **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 | 462.250 |
| **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 | 900.000 |

In [ ]:
```
```