In [1]:

```
#1. Импортируйте библиотеки pandas и numpy.
import numpy as np
import pandas as pd
```

In [2]:

```
#Загрузите "Boston House Prices dataset" из встроенных наборов данных библиотеки sklearn.
from sklearn.datasets import load_boston
boston = load_boston()
boston.keys()
```

Out[2]:

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

In [3]:

```
#Создайте датафреймы X и y из этих данных.
data = boston["data"]
data.shape
```

Out[3]:

```
(506, 13)
```

In [4]:

```
feature_names = boston["feature_names"]
feature_names
```

Out[4]:

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

In [5]:

```
target = boston["target"]
target[:10]
```

Out[5]:

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

In [6]:

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

Out[6]:

	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	
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	
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	
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	
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	

In [7]:

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 13 columns):
CRIM       506 non-null float64
ZN         506 non-null float64
INDUS      506 non-null float64
CHAS       506 non-null float64
NOX        506 non-null float64
RM         506 non-null float64
AGE        506 non-null float64
DIS        506 non-null float64
RAD        506 non-null float64
TAX        506 non-null float64
PTRATIO    506 non-null float64
B          506 non-null float64
LSTAT      506 non-null float64
dtypes: float64(13)
memory usage: 51.5 KB
```

In [8]:

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 1 columns):
price    506 non-null float64
dtypes: float64(1)
memory usage: 4.0 KB
```

In [9]:

```
#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.
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)
```

In [10]:

```
#Создайте модель линейной регрессии под названием lr с помощью класса
#LinearRegression из модуля sklearn.linear_model.
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
```

In [11]:

```
#Обучите модель на тренировочных данных (используйте все признаки) и сделайте предсказание
lr.fit(X_train, y_train)
```

Out[11]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [12]:

```
y_pred = lr.predict(X_test)
y_pred.shape
```

Out[12]:

```
(152, 1)
```

In [13]:

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

check_test.head(10)
```

Out[13]:

	y_test	y_pred
233	48.3	36.046222
235	24.0	25.669899
305	28.4	31.023508
78	21.2	21.111576
322	20.4	23.090008
499	17.5	19.053526
398	5.0	6.572040
87	22.2	26.167542
301	22.0	29.112336
232	41.7	36.925940

In [14]:

```
#Вычислите R2 полученных предказаний с помощью r2_score из модуля sklearn.metrics.
check_test["error"] = check_test["y_pred"] - check_test["y_test"]
check_test.head()
```

Out[14]:

	y_test	y_pred	error
233	48.3	36.046222	-12.253778
235	24.0	25.669899	1.669899
305	28.4	31.023508	2.623508
78	21.2	21.111576	-0.088424
322	20.4	23.090008	2.690008

In [15]:

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

Out[15]:

```
0.6606552063780124
```

In [16]:

```
#2. Создайте модель под названием model с помощью RandomForestRegressor из модуля sklearn.e
#Сделайте агрумент n_estimators равным 1000, max_depth должен быть равен 12 и random_state
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor(n_estimators=1000, max_depth=12, random_state=42)
```

In [17]:

```
#Обучите модель на тренировочных данных аналогично тому, как вы обучали модель LinearRegres
#но при этом в метод fit вместо датафрейма y_train поставьте y_train.values[:, 0], чтобы по
#из датафрейма одномерный массив Numpy, так как для класса RandomForestRegressor в данном м
#для аргумента y предпочтительно применение массивов вместо датафрейма.
model.fit(X_train, y_train.values[:, 0])
```

Out[17]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=12,
           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=1000, n_jobs=None,
           oob_score=False, random_state=42, verbose=0, warm_start=False)
```

In [18]:

```
#Сделайте предсказание на тестовых данных и посчитайте R2.
y_pred = model.predict(X_test)
y_pred.shape
```

Out[18]:

```
(152,)
```

In [19]:

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

check_test.head(10)
```

Out[19]:

	y_test	y_pred
233	48.3	44.402100
235	24.0	22.933879
305	28.4	26.063353
78	21.2	20.603344
322	20.4	21.485415
499	17.5	19.129334
398	5.0	6.873510
87	22.2	21.860189
301	22.0	24.913114
232	41.7	44.547800

In [20]:

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

Out[20]:

```
0.8663837866131361
```

In [21]:

```
#Сравните с результатом из предыдущего задания.
#Напишите в комментариях к коду, какая модель в данном случае работает лучше.
r2_score_1<r2_score_2
```

Out[21]:

```
True
```

In [22]:

```
#Вывод: мдель RandomForestRegressor работает лучше, чем модель LinearRegression
```

In [23]:

```
#3. Вызовите документацию для класса RandomForestRegressor,
?RandomForestRegressor
```

In [24]:

```
#найдите информацию об атрибуте feature_importances.
#feature_importances_ : array of shape = [n_features]
#    The feature importances (the higher, the more important the feature).
```

In [25]:

```
#С помощью этого атрибута найдите сумму всех показателей важности,
print(model.feature_importances_)
```

```
[0.05536014 0.00145701 0.00485976 0.00134209 0.02539987 0.25424547
 0.01744122 0.06553827 0.00423607 0.0081908  0.01881172 0.0140924
 0.52902519]
```

In [26]:

```
model.feature_importances_.sum()
```

Out[26]:

```
1.0000000000000004
```

In [27]:

```
#установите, какие два признака показывают наибольшую важность.
max_value_idx1=model.feature_importances_.argmax()
max_value_idx1
```

Out[27]:

```
12
```

In [28]:

```
max_value_idx2=0
max_value=model.feature_importances_[max_value_idx2]
for i in range(model.n_features_):
    if max_value<model.feature_importances_[i] and i!=max_value_idx1:
        max_value=model.feature_importances_[i]
        max_value_idx2=i
print(max_value_idx2)
```

```
5
```

In [29]:

```
#4. В этом задании мы будем работать с датасетом, с которым мы уже знакомы по домашнему зад
#по библиотеке Matplotlib, это датасет Credit Card Fraud Detection. Для этого датасета мы б
#решать задачу классификации - определять, какие из транзакциции по кредитной карте являютс
#мошенническими. Данный датасет несбалансирован (так как случаи мошенничества относительно
#так что применение метрики accuracy не принесет пользы и не поможет выбрать лучшую модель.
#вычислять AUC, то есть площадь под кривой ROC. Импортируйте из соответствующих модулей
#RandomForestClassifier, GridSearchCV и train_test_split.
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
```

In [30]:

```
#Загрузите датасет creditcard.csv и создайте датафрейм df.
df = pd.read_csv('creditcard.csv')
df.head()
```

Out[30]:

	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 rows × 31 columns

In [31]:

```
#С помощью метода value_counts с аргументом normalize=True убедитесь в том, что выборка нес
df["Class"].value_counts(normalize=True)
```

Out[31]:

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

In [32]:

```
#Используя метод info, проверьте, все ли столбцы содержат числовые данные
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
Time      284807 non-null float64
V1        284807 non-null float64
V2        284807 non-null float64
V3        284807 non-null float64
V4        284807 non-null float64
V5        284807 non-null float64
V6        284807 non-null float64
V7        284807 non-null float64
V8        284807 non-null float64
V9        284807 non-null float64
V10       284807 non-null float64
V11       284807 non-null float64
V12       284807 non-null float64
V13       284807 non-null float64
V14       284807 non-null float64
V15       284807 non-null float64
V16       284807 non-null float64
V17       284807 non-null float64
V18       284807 non-null float64
V19       284807 non-null float64
V20       284807 non-null float64
V21       284807 non-null float64
V22       284807 non-null float64
V23       284807 non-null float64
V24       284807 non-null float64
V25       284807 non-null float64
V26       284807 non-null float64
V27       284807 non-null float64
V28       284807 non-null float64
Amount    284807 non-null float64
Class     284807 non-null int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

In [33]:

```
#и нет ли в них пропусков
df.isnull().astype(np.int).sum().astype(np.int)
```

Out[33]:

```
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: int32
```

In [34]:

```
#Примените следующую настройку, чтобы можно было просматривать все столбцы датафрейма:
#pd.options.display.max_columns = 100.
pd.options.display.max_columns = 100
```

In [35]:

```
#Просмотрите первые 10 строк датафрейма df.
df.head(10)
```

Out[35]:

	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

In [36]:

```
#Создайте датафрейм X из датафрейма df, исключив столбец Class.
X = df.drop("Class", axis=1)
X.head()
```

Out[36]:

	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

In [37]:

```
#Создайте объект Series под названием y из столбца Class.
y = df["Class"]
y.head()
```

Out[37]:

```
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int64
```

In [38]:

```
#Разбейте X и y на тренировочный и тестовый наборы данных при помощи функции train_test_spl
#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.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=100,
```

In [39]:

```
#Просмотрите информацию о их форме.
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[39]:

```
((199364, 30), (85443, 30), (199364,), (85443,))
```

In [40]:

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

In [41]:

```
#Создайте модель GridSearchCV со следующими аргументами:
#estimator=RandomForestClassifier(random_state=100),
#param_grid=parameters,
#scoring='roc_auc',
#cv=3.
clf = GridSearchCV(
    estimator=RandomForestClassifier(random_state=100),
    param_grid=parameters,
    scoring='roc_auc',
    cv=3,
)
```

In [42]:

```
#Обучите модель на тренировочном наборе данных (может занять несколько минут).
clf.fit(X_train, y_train)
```

Out[42]:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, c
riterion='gini',
            max_depth=None, 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=None,
            oob_score=False, random_state=100, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid=[{'n_estimators': [10, 15], 'max_features': array([3, 4]),
'max_depth': array([4, 5, 6])}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=0)
```

In [43]:

```
#Просмотрите параметры лучшей модели с помощью атрибута best_params_.
clf.best_params_
```

Out[43]:

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

In [44]:

```
#Предскажите вероятности классов с помощью полученнной модели и метода predict_proba.
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]]
```

In [45]:

```
#Из полученного результата (массив Numpy) выберите столбец с индексом 1 (вероятность класса
#и запишите в массив y_pred_proba.
y_pred_proba = y_pred_proba[:, 1]
print(y_pred_proba[:5])
```

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

In [46]:

```
#Из модуля sklearn.metrics импортируйте метрику roc_auc_score.
from sklearn.metrics import roc_auc_score
```

In [47]:

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

Out[47]:

```
0.9462664156037156
```

####Дополнительные задания:

In [48]:

```
#Загрузите датасет Wine из встроенных датасетов sklearn.datasets с помощью функции load_win
#Полученный датасет не является датафреймом. Это структура данных, имеющая ключи аналогично
from sklearn.datasets import load_wine
data = load_wine()
data.keys()
```

Out[48]:

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

In [49]:

```
#Просмотрите тип данных этой структуры данных
type(data)
```

Out[49]:

```
sklearn.utils.Bunch
```

In [50]:

```
#и создайте список data_keys, содержащий ее ключи.
data_keys=data["feature_names"]
data_keys
```

Out[50]:

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

In [51]:

```
#Просмотрите данные
data.data
```

Out[51]:

```
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 [52]:

```
data.data.shape
```

Out[52]:

```
(178, 13)
```

In [53]:

```
#описание и названия признаков в датасете. Описание нужно вывести в виде привычного, аккура
#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 (ht
tps://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
[http://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 [54]:

```
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 (ht
```

```
tps://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
[http://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 [55]:

```
#Сколько классов содержит целевая переменная датасета?
np.unique(data["target"]).shape
```

Out[55]:

```
(3,)
```

In [56]:

```
#Выведите названия классов.
data["target_names"]
```

Out[56]:

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

In [57]:

```
#На основе данных датасета (они содержатся в двумерном массиве Numpy)
#и названий признаков создайте датафрейм под названием X.
X = pd.DataFrame(data.data, columns=feature_names)
X.head()
```

Out[57]:

	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

In [58]:

```
#Выясните размер датафрейма X и установите, имеются ли в нем пропущенные значения.
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 13 columns):
CRIM       178 non-null float64
ZN         178 non-null float64
INDUS      178 non-null float64
CHAS       178 non-null float64
NOX        178 non-null float64
RM         178 non-null float64
AGE        178 non-null float64
DIS        178 non-null float64
RAD        178 non-null float64
TAX        178 non-null float64
PTRATIO    178 non-null float64
B          178 non-null float64
LSTAT      178 non-null float64
dtypes: float64(13)
memory usage: 18.2 KB
```

In [59]:

```
X.shape
```

Out[59]:

```
(178, 13)
```

In [60]:

```
X.isnull().astype("int").sum()
```

Out[60]:

```
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
```

In [61]:

```
#Добавьте в датафрейм поле с классами вин в виде чисел, имеющих тип данных numpy.int64. Наз
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):
CRIM       178 non-null float64
ZN         178 non-null float64
INDUS      178 non-null float64
CHAS       178 non-null float64
NOX        178 non-null float64
RM         178 non-null float64
AGE        178 non-null float64
DIS        178 non-null float64
RAD        178 non-null float64
TAX        178 non-null float64
PTRATIO    178 non-null float64
B          178 non-null float64
LSTAT      178 non-null float64
target     178 non-null int64
dtypes: float64(13), int64(1)
memory usage: 19.5 KB
```

In [62]:

```
X.head()
```

Out[62]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	targ
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]:

```
#Постройте матрицу корреляций для всех полей X. Дайте полученному датафрейму название X_cor
X_corr=X.corr()
X_corr
```

Out[63]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS
CRIM	1.000000	0.094397	0.211545	-0.310235	0.270798	0.289101	0.236815	-0.155929
ZN	0.094397	1.000000	0.164045	0.288500	-0.054575	-0.335167	-0.411007	0.292977
INDUS	0.211545	0.164045	1.000000	0.443367	0.286587	0.128980	0.115077	0.186230
CHAS	-0.310235	0.288500	0.443367	1.000000	-0.083333	-0.321113	-0.351370	0.361922
NOX	0.270798	-0.054575	0.286587	-0.083333	1.000000	0.214401	0.195784	-0.256294
RM	0.289101	-0.335167	0.128980	-0.321113	0.214401	1.000000	0.864564	-0.449935
AGE	0.236815	-0.411007	0.115077	-0.351370	0.195784	0.864564	1.000000	-0.537900
DIS	-0.155929	0.292977	0.186230	0.361922	-0.256294	-0.449935	-0.537900	1.000000
RAD	0.136698	-0.220746	0.009652	-0.197327	0.236441	0.612413	0.652692	-0.365845
TAX	0.546364	0.248985	0.258887	0.018732	0.199950	-0.055136	-0.172379	0.139057
PTRATIO	-0.071747	-0.561296	-0.074667	-0.273955	0.055398	0.433681	0.543479	-0.262640
B	0.072343	-0.368710	0.003911	-0.276769	0.066004	0.699949	0.787194	-0.503270
LSTAT	0.643720	-0.192011	0.223626	-0.440597	0.393351	0.498115	0.494193	-0.311385
target	-0.328222	0.437776	-0.049643	0.517859	-0.209179	-0.719163	-0.847498	0.489109

In [64]:

```
#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).
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[64]:

```
['CHAS', 'RM', 'AGE', 'PTRATIO', 'B', 'LSTAT']
```

In [65]:

```
#Удалите из датафрейма X поле с целевой переменной.
X=X.drop("target", axis=1)
X.head()
```

Out[65]:

	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

In [66]:

```
#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 квадр
for i in high_corr:
    X[i+"_2"]=X[i]**2
X.head()
```

Out[66]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	CH
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	2
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	1
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
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	2
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	4

In [67]:

```
#Выведите описание полей датафрейма X с помощью метода describe.
X.describe()
```

Out[67]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	17
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	

In [ ]:

```

```