[Обучение без учителя 2](#_Toc137022680)

[Структура курса 2](#_Toc137022681)

[Ваши цели 2](#_Toc137022682)

[Введение 2](#_Toc137022683)

[Чему вы научитесь 2](#_Toc137022684)

[Сколько времени это займёт 2](#_Toc137022685)

[Постановка задачи 2](#_Toc137022686)

[Задача кластеризации 3](#_Toc137022687)

[Алгоритм k-средних 5](#_Toc137022688)

[Задачи 6](#_Toc137022689)

[Целевая функция 8](#_Toc137022690)

[Задача 9](#_Toc137022691)

[Локальный минимум 10](#_Toc137022692)

[Визуализация 12](#_Toc137022693)

[Задача 14](#_Toc137022694)

[Оптимальное число кластеров 16](#_Toc137022695)

[Метод локтя 16](#_Toc137022696)

[Задачи 17](#_Toc137022697)

[Поиск структуры в данных 19](#_Toc137022698)

[Описание данных 19](#_Toc137022699)

[Задачи 20](#_Toc137022700)

## Обучение без учителя

Вы научитесь искать закономерности в данных без разметки.

#### Структура курса

**Обучение без учителя** (англ. unsupervised learning) — задача машинного обучения без целевого признака, когда взаимосвязи между объектами алгоритмы находят самостоятельно. Выбор алгоритма зависит от типа задачи.

Изучение типов задач вы начнёте с кластеризации, в которой нужно выявить похожие объекты и объединить их в группы. Решить эту задачу поможет алгоритм k-средних, который вы освоите.

Затем познакомитесь с задачей поиска аномалий, в которой нужно найти объекты, непохожие на большинство других. Для её решения вы научитесь применять изоляционный лес и метод ближайших соседей.

#### Ваши цели

* Разобраться, что такое кластеризация.
* Научиться применять и настраивать алгоритм k-средних.
* Освоить методы поиска аномалий в данных.

Это несложный курс: вы познакомитесь с новыми алгоритмами и научитесь их применять.

### Введение

Вы познакомитесь с задачей кластеризации.

#### Чему вы научитесь

* Разберётесь, что такое кластеризация.
* Познакомитесь с алгоритмом k-средних.
* Узнаете, как выбирать оптимальное количество кластеров для набора данных.

#### Сколько времени это займёт

11 уроков по 10 минут

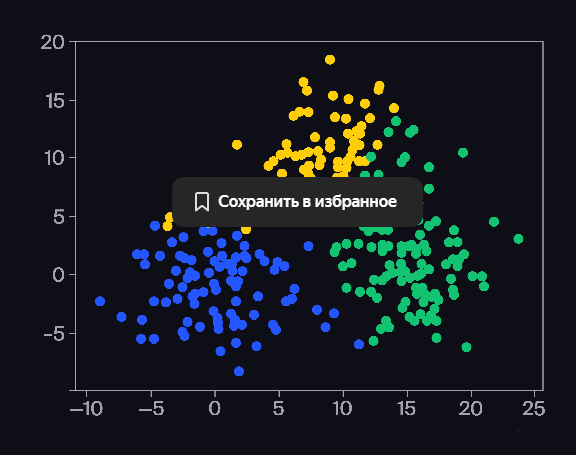
#### Постановка задачи

Освоите методы кластеризации на данных о поведении пользователей, а затем проанализируете датасет с моделями автомобилей.

### Задача кластеризации

В задаче обучения с учителем модель восстанавливает зависимость между признаками и целевым признаком, чтобы его предсказать для новых объектов. В обучении без учителя целевого признака нет, и модель ищет взаимосвязи между объектами, как, например, в задаче кластеризации.

**Кластеризация** (от англ. cluster, «гроздь») — объединение похожих объектов в группы, или кластеры. Кластеризацию также называют кластерным анализом или сегментацией.



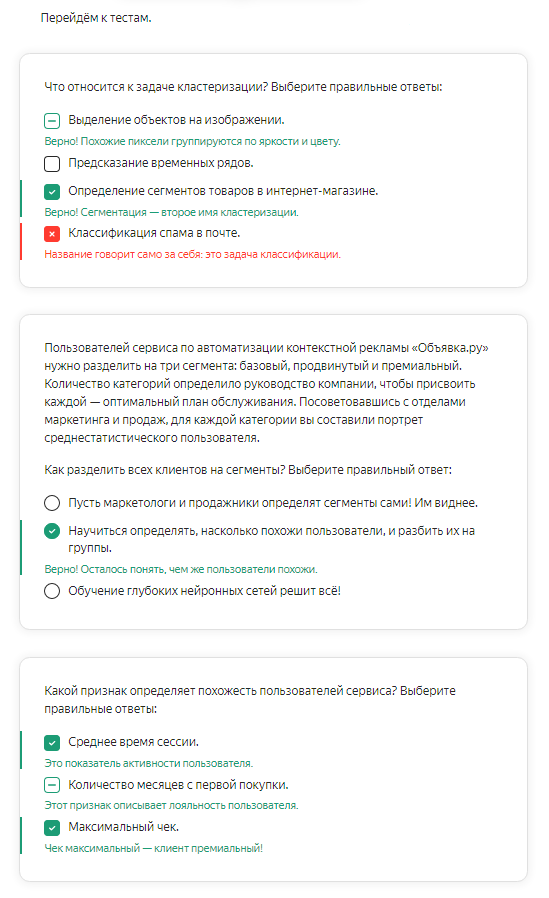
В большинстве методов кластеризации схожесть или различие объектов определяется расстоянием между ними. Чем дальше объекты друг от друга, тем меньше сходства и наоборот.

Кластеризацию не сто́ит путать с задачей классификации, в которой заранее известны классы и принадлежащие им объекты. В кластеризации они не заданы и определяются разными способами. Они зависят от того, что считать похожими объектами. Допустим, вы хотите упорядочить домашнюю библиотеку. Выбираете четыре разножанровые книги и каждую ставите на отдельную полку. Теперь осталось разобраться с жанрами других произведений и распределить по полкам. Хотя объединить книги можно было по размеру или цвету обложки.

Похожим образом составляются и плейлисты: треки группируются по стилю, году выхода релиза или лейблу. Кстати, чтобы предлагать каждому пользователю умные плейлисты, Яндекс.Музыка делит аудиосигнал на частотные составляющие.

В бизнесе кластеризация помогает:

* сегментировать пользователей или товары. Такая задача тесно связана с рекомендательными системами.
* выявлять мошенников по нетипичному поведению: например, накрутка кликов или лайков в соцсетях.



### Алгоритм k-средних

##### Познакомимся с самым популярным алгоритмом кластеризации — методом **k-средних** (англ. k-means).

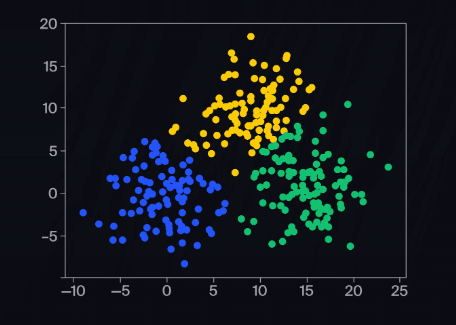
Рассмотрим ключевое понятие алгоритма — центроид (англ. centroid), или центр кластера. От степени близости к конкретному центру зависит, в какой кластер попадёт объект. У каждого кластера центроид свой, а вычисляется он как среднее арифметическое объектов кластера.

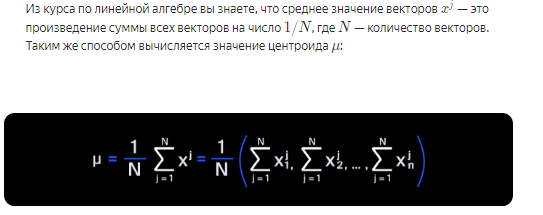
K-средних сегментирует объекты пошагово, поэтому это итеративный алгоритм. Разберём, как он работает для заданного числа кластеров k:

1. Каждому объекту алгоритм случайным образом присваивает номер кластера — от 1 до k.
2. Пока кластеры объектов не перестанут меняться, алгоритм повторяет итерацию из двух шагов:
   * вычисляет центроид каждого кластера;
   * каждому объекту присваивает номер нового кластера, центроид которого расположен ближе всего к объекту.

Другим условием остановки алгоритма может быть выполнение максимального количества итераций max\_iter. Этот параметр разберём позднее.

Проиллюстрируем работу алгоритма. В начале его запуска объекты раскрашены случайным образом. Разные цвета обозначают номера кластеров. После итерации алгоритма объекты одного цвета объединяются, а у кластеров появляются явные границы.





Вернёмся к нашей задаче. Для её решения пригодятся признаки типичного пользователя в каждом сегменте:

* Базовый: среднее время сессии (признак timespent) — 50 минут, средний чек purchase — 20 тысяч рублей, 5 месяцев с момента регистрации на сервисе — months.
* Продвинутый: среднее время сессии — 20 минут, средний чек — 30 тысяч рублей, 10 месяцев с момента регистрации.
* Премиальный: среднее время сессии — 20 минут, средний чек — 80 тысяч рублей, 8 месяцев с момента регистрации.

Признаки будем считать приближёнными: это лишь предположение отдела маркетинга. Они нужны, чтобы подсказать алгоритму, где искать кластеры. Значения этих признаков передадим на вход *k*-средних, чтобы задать **начальные центроиды** — это опциональный параметр. В задачах сравним обученную без этого параметра модель и модель с начальными центроидами.

Если на вход *k-means* передать начальные центроиды, Python во время работы выдаст предупреждение *RuntimeWarning* (англ. «цикл работы»). Чтобы этого избежать, добавим блокировку этого вывода filterwarnings() (англ. «отфильтровать предупреждения»). Вернёмся к предупреждению позже, а пока в код задачи добавим такие строки:

import warnings

warnings.filterwarnings("ignore", category=RuntimeWarning)

#### Задачи

1.

В документации sklearn найдите метод k-средних и импортируйте его. Обучите модель для трёх кластеров пользователей и заданного параметра random\_state=12345. Напечатайте на экране значения центроидов полученных кластеров.

import pandas as pd

from sklearn.cluster import KMeans

data = pd.read\_csv('/datasets/segments.csv')

#print(data.head())

# Обучение модели

model = KMeans(n\_clusters=3,random\_state=12345)

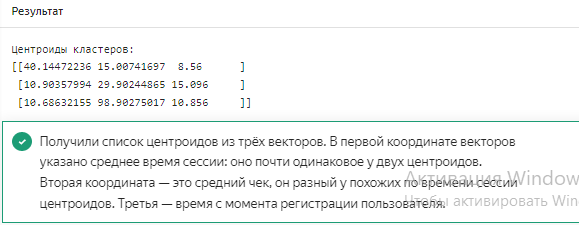
#model = # < напишите код здесь >

# < напишите код здесь >

model.fit(data[['timespent','purchase','months']])

print("Центроиды кластеров:")

print(model.cluster\_centers\_)



2.

В документации метода k-средних найдите, как модели можно передать начальные центроиды. К прекоду добавьте обучение модели с начальными центроидами, заданными в переменной centers. Выведите на экран:

* центроиды кластеров для модели из прошлого задания (уже в прекоде),
* центроиды кластеров для модели с начальными центроидами.

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

data = pd.read\_csv('/datasets/segments.csv')

centers = np.array([[20, 80, 8], [50, 20, 5], [20, 30, 10]])

model = KMeans(n\_clusters=3, random\_state=12345)

model.fit(data)

print("Центроиды кластеров:")

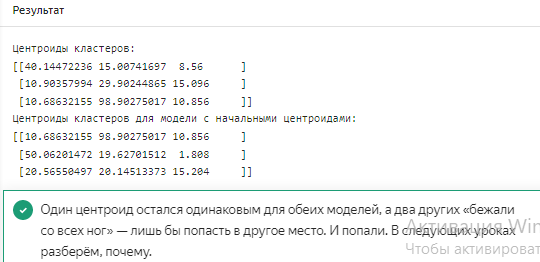
print(model.cluster\_centers\_)

model = KMeans(n\_clusters=3, init=centers, random\_state=12345)

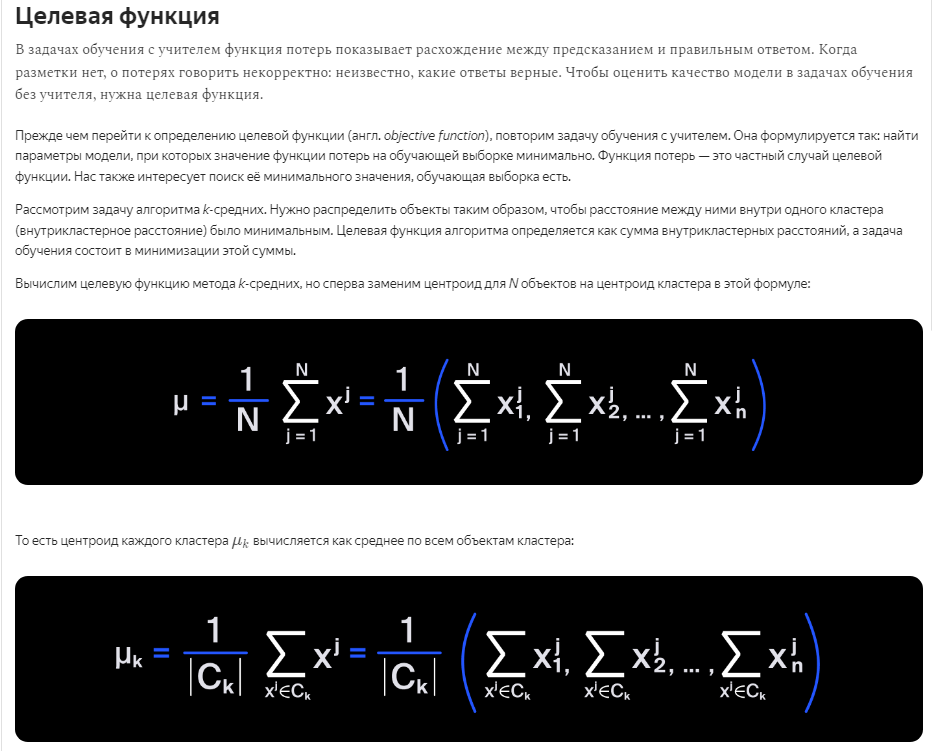
model.fit(data)

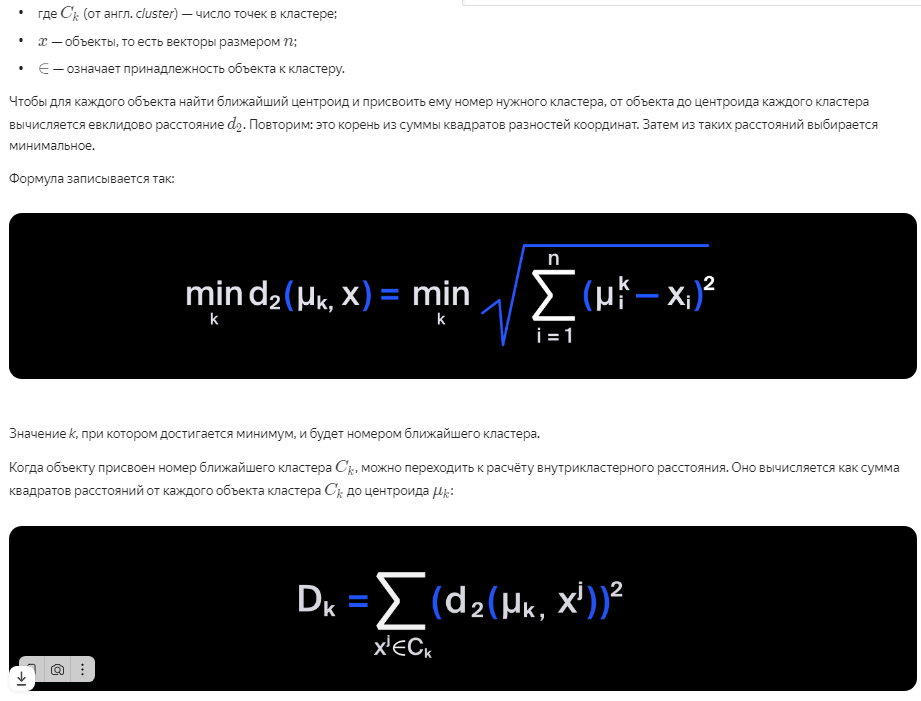
print("Центроиды кластеров для модели с начальными центроидами:")

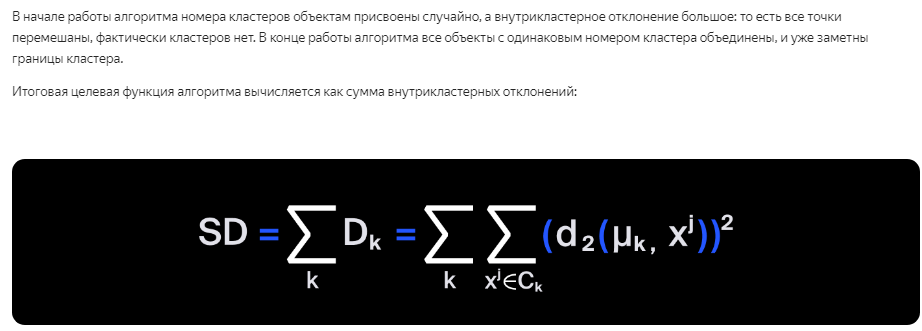
print(model.cluster\_centers\_)



### Целевая функция







#### Задача

В документации метода *sklearn.cluster.KMeans* найдите атрибут, отвечающий за целевую функцию. Добавьте к коду из предыдущего урока подсчёт этой функции для двух моделей: без начальных центроидов и с ними. Напечатайте на экране значения целевой функции для обеих моделей.

Подсказка

Значение функции потерь хранится в атрибуте *inertia\_* (англ. «инерция»).

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

data = pd.read\_csv('/datasets/segments.csv')

centers = np.array([[20, 80, 8], [50, 20, 5], [20, 30, 10]])

model = KMeans(n\_clusters=3, random\_state=12345)

model.fit(data)

print("Целевая функция:")

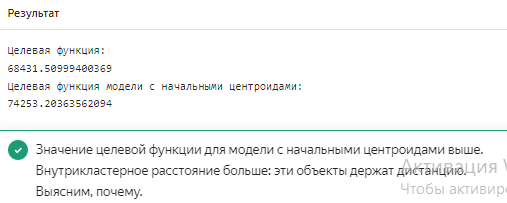
print(model.inertia\_)

model = KMeans(n\_clusters=3, init=centers, random\_state=12345)

model.fit(data)

print("Целевая функция модели с начальными центроидами:")

print(model.inertia\_)



### Локальный минимум

##### **Вы получили два значения целевой функции. Причём больше значение у модели с начальными центроидами. Разберём, почему так получилось и что с этим делать.**

Бежать к маркетологам выяснять причину бессмысленно. Может, дело в RuntimeWarning? Всё-таки блокировка предупреждения до добра не довела.

Разберём RuntimeWarning подробнее. Если убрать блокировку, обучение модели выдаст такое предупреждение:

Скопировать кодPYTHON

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

data = pd.read\_csv('/datasets/segments.csv')

centers = np.array([[20, 80, 8], [50, 20, 5], [20, 30, 10]])

model = KMeans(n\_clusters=3, random\_state=12345)

model.fit(data)

print("Целевая функция:")

print(model.inertia\_)

model = KMeans(n\_clusters=3, init=centers, random\_state=12345)

model.fit(data)

print("Целевая функция модели с начальными центроидами:")

print(model.inertia\_)

Скопировать кодPYTHON

Целевая функция:

68431.50999400369

Целевая функция модели с начальными центроидами:

74253.20363562103

/usr/local/lib/python3.6/site-packages/sklearn/cluster/k\_means\_.py:969: RuntimeWarning: Explicit initial center position passed: performing only one init in k-means instead of n\_init=10

return\_n\_iter=True)

Блокировка вывода предупреждений filterwarnings() скрывает сообщение о том, что количество запусков алгоритма n\_init с начальными центроидами стало равно единице.

Повторим: каждый запуск начинается с того, что алгоритм случайным образом присваивает объектам номер кластера. То есть получаем стартовый набор объектов определённого кластера. При следующем запуске этим же объектам присваиваются новые номера, а значение целевой функции по итогу работы алгоритма меняется.

При каждом запуске алгоритма целевая функция получается минимальной для конкретного стартового набора объектов кластера. Такой минимум называется локальным.

В нашей задаче это значение 74253.20363562103. При запуске на другом стартовом наборе значение функции может стать иным и будет новый локальный минимум.

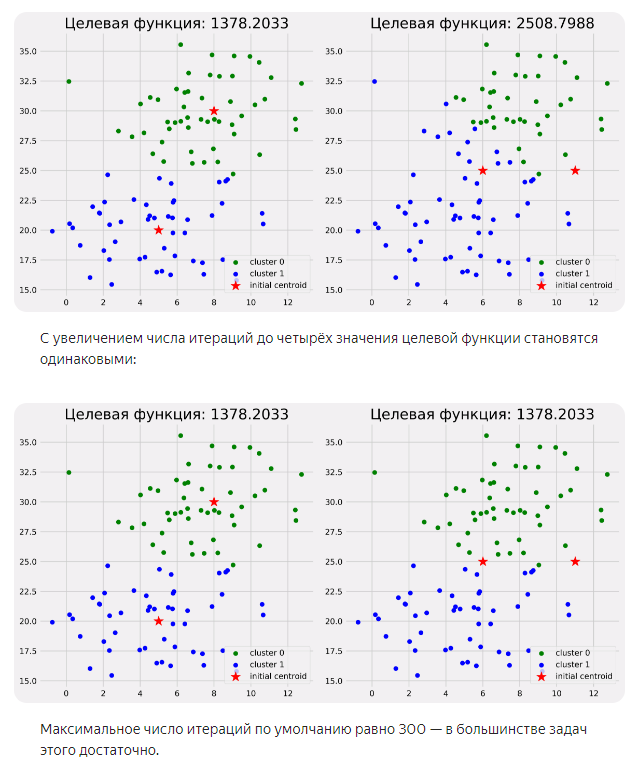
Параметр n\_init по умолчанию равен 10. Алгоритм запускается 10 раз с разными начальными кластерами. Из всех локальных минимумов выбирается наименьший. В нашей задаче это значение 68431.50999400369.

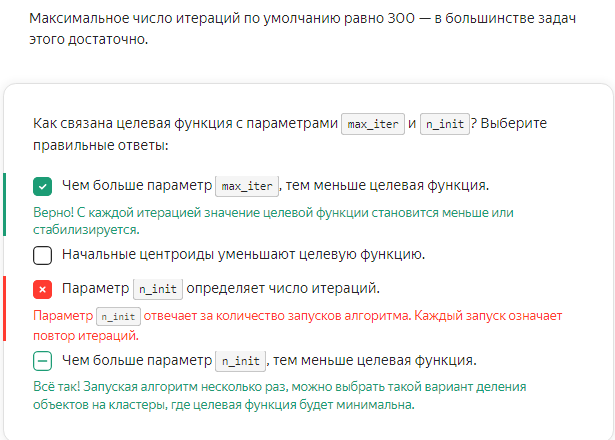
Подытожим. Целевая функция потерь с начальными центроидами получилась больше, потому что алгоритм запускался всего один раз. При 10-кратном запуске случайно удалось найти начальные центроиды лучше.

Разберём на примере ещё один параметр — max\_iter, определяющий количество итераций алгоритма. Чем их больше, тем ближе к локальному минимуму мы подойдём.

Перед вами результаты двух запусков алгоритма на синтетических данных с параметром max\_iter=1. Данные сгенерированы как два облака точек. Заранее известно, что это два кластера. На графиках отмечены начальные центроиды: они-то и повлияли на распределение точек по кластерам.

Слева начальные центроиды попали примерно в центры кластеров. Справа начальные центроиды уже смещены от кластеров, сами кластеры также сдвинулись. Целевая функция больше значения слева.





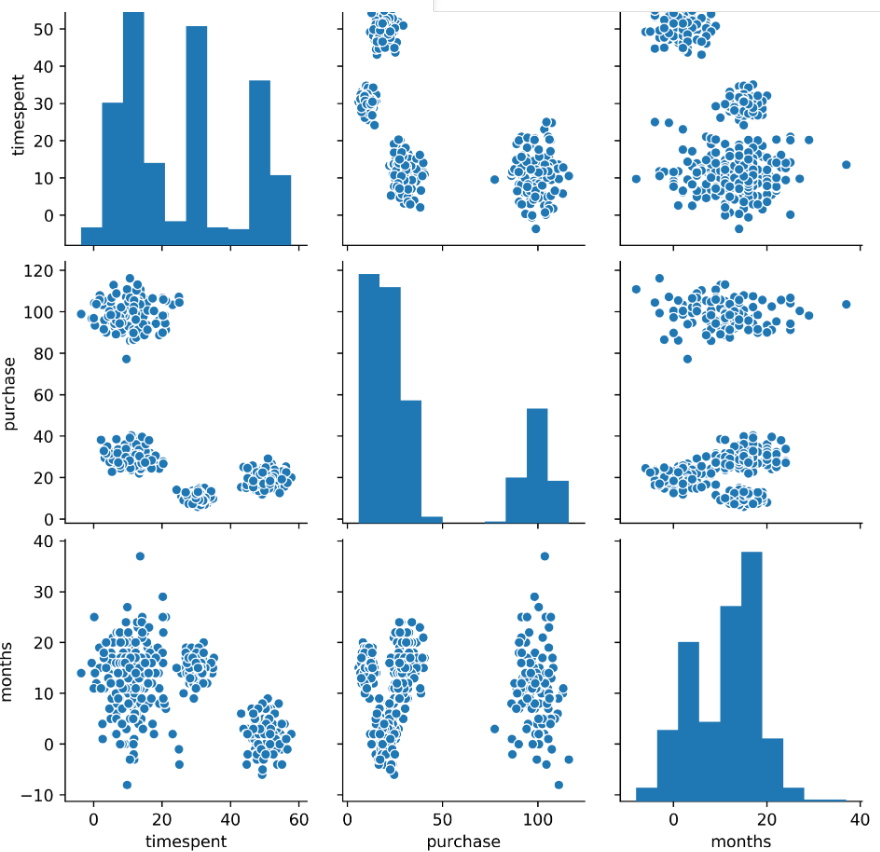
### Визуализация

##### **Для визуализации данных о пользователях сервиса построим график методом pairplot (англ. «парный график») из библиотеки seaborn.**

На диагонали находится распределение признака: purchase, timespent и months. В других ячейках — диаграммы рассеяния между всеми парами признаков. Тип графика по диагонали определяется параметром diag\_kind:

import seaborn as sns

sns.pairplot(data, diag\_kind='hist')



На графике изображены скопления точек — это будущие кластеры. Что интересного заметили?

* Чётко выделяются четыре группы объектов на проекции пары признаков: purchase (средний чек) и timespent (среднее время сессии).
* На других проекциях видно по две-три группы точек.
* Скопление объектов с высокими значениями purchase заметно отделено от остальных точек.

Добавим заливку кластеров в график модели, обученной без начальных центроидов. Правила заливки задаёт параметр hue (англ. «оттенок»). Он принимает на вход массив из строковых переменных. Поэтому в массив строк переведём номера кластеров. А чтобы добавить на график центроиды, дадим им названия. Затем все данные объединим.

import pandas as pd

from sklearn.cluster import KMeans

import seaborn as sns

data = pd.read\_csv('segments.csv')

model = KMeans(n\_clusters=3, random\_state=12345)

model.fit(data)

centroids = pd.DataFrame(model.cluster\_centers\_, columns=data.columns)

*# Добавление столбца с номером кластера*

data['label'] = model.labels\_.astype(str)

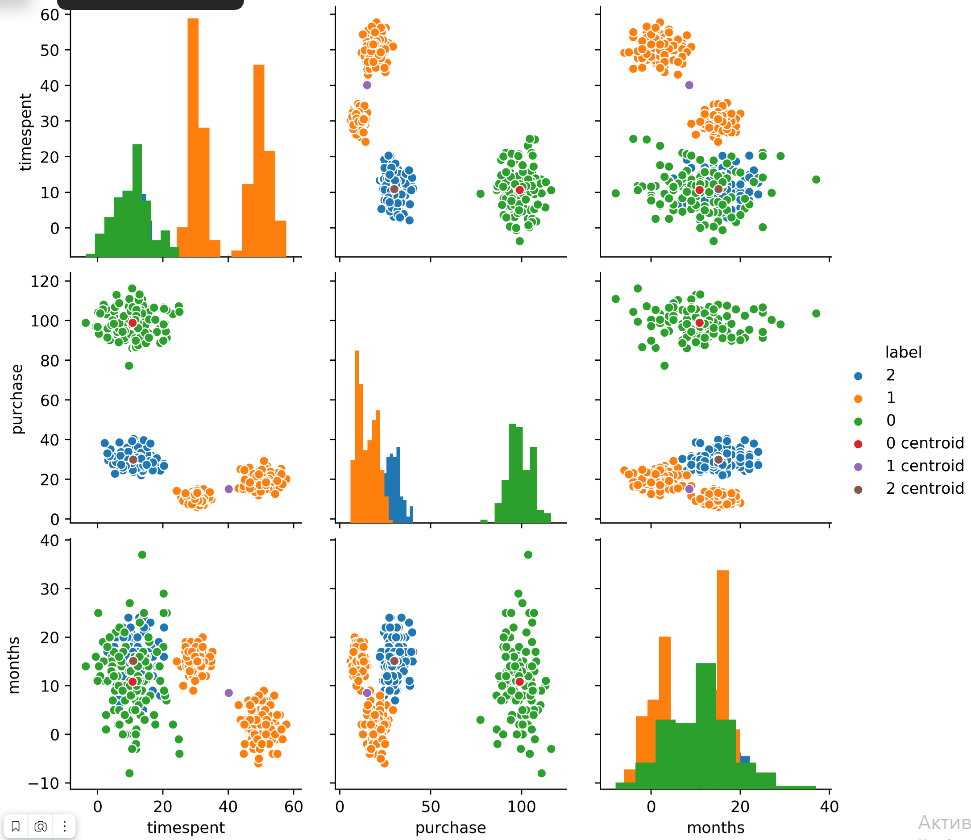
centroids['label'] = ['0 centroid', '1 centroid', '2 centroid']

*# Сброс индекса понадобится дальше*

data\_all = pd.concat([data, centroids], ignore\_index=True)

*# Построение графика*

sns.pairplot(data\_all, hue='label', diag\_kind='hist')



Наши наблюдения подтвердились: группа точек с высокими значениями признака purchase попала в отдельный кластер. Мы определили премиальный сегмент.

С базовым и продвинутым сегментами сложнее. Чтобы определить подходящие им кластеры, на график добавим начальные центроиды дополнительным слоем. Для этого сохраним соответствующий графику объект — PairGrid (от англ. «парная сетка»):

Скопировать кодPYTHON

pairgrid = sns.pairplot(data\_all, hue='label', diag\_kind='hist')

Дополнительные значения для построения графика передадим через атрибут *data*:

Скопировать кодPYTHON

centroids\_init = pd.DataFrame([[20, 80, 8], [50, 20, 5], [20, 30, 10]], \

columns=data.drop(columns=['label']).columns)

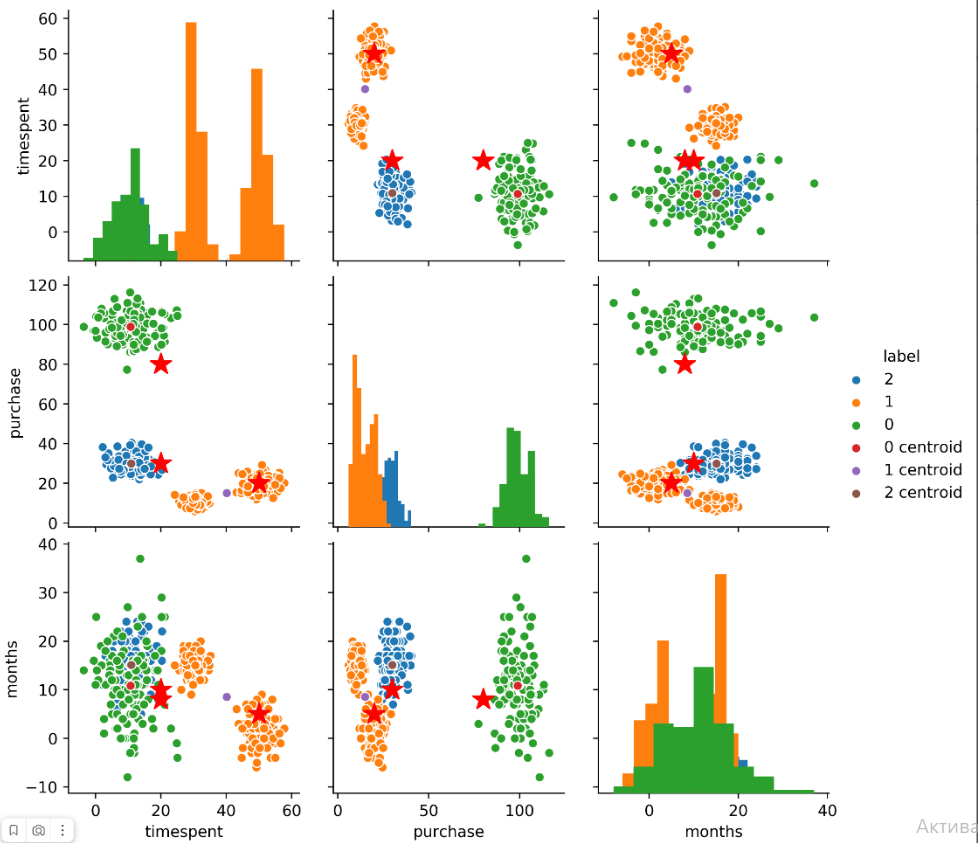
centroids\_init['label'] = 4

pairgrid.data = centroids\_init

Обратите внимание на последний столбец label. В каждой строке столбца указано значение 4. Функция этого столбца формальна: метод map\_offdiag библиотеки seaborn требует, чтобы в дополнительных данных присутствовал столбец, который передаётся параметру hue. Если не создать такой столбец, произойдёт ошибка. Добавить в дополнительный столбец label лучше такое значение, которое в основном столбце label не встречалось, например, 4.

Вызовем ещё один метод — map\_offdiag (от англ. *map off diagonal,* «график вне диагонали»). Он строит данные из pairgrid.data на проекциях вне диагоналей. Параметр func определяет тип графика, s — размер (от англ. *size*), marker — форму точек, а palette — цветовую палитру:

pairgrid.map\_offdiag(func=sns.scatterplot, s=200, marker='\*', palette='flag')



Посмотрим на проекцию purchase и timespent. У кластера с номером «2» среднее время сессии меньше — значит, это продвинутый сегмент, а кластер с номером «1» — базовый. Найдём различия между полученным графиком и результатом кластеризации с начальными центроидами.

#### Задача

Обучите модель с начальными центроидами centers. Постройте диаграмму pairplot с заливкой по кластерам и центроидами полученных кластеров. Начальные центроиды добавьте отдельным слоем без заливки.

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

import seaborn as sns

import warnings

warnings.filterwarnings("ignore", category=RuntimeWarning)

data = pd.read\_csv('https://code.s3.yandex.net/datasets/segments.csv')

centers = np.array([[20, 80, 8], [50, 20, 5], [20, 30, 10]])

model = KMeans(n\_clusters=3, init=centers, random\_state=12345)

model.fit(data)

centroids = pd.DataFrame(model.cluster\_centers\_, columns=data.columns)

data['label'] = model.labels\_.astype(str)

centroids['label'] = ['0 centroid', '1 centroid', '2 centroid']

data\_all = pd.concat([data, centroids], ignore\_index=True)

pairgrid = sns.pairplot(data\_all, hue='label', diag\_kind='hist')

centroids\_init = pd.DataFrame([[20, 80, 8], [50, 20, 5], [20, 30, 10]], \

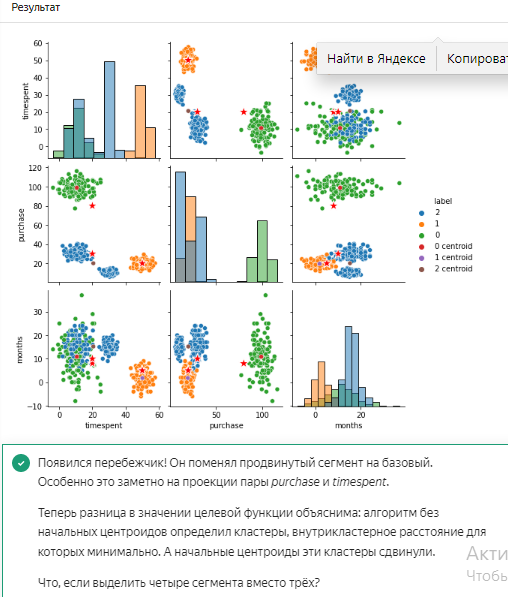
columns=data.drop(columns=['label']).columns)

centroids\_init['label'] = 4

pairgrid.data = centroids\_init

centroids\_init['label'] = 4

pairgrid.map\_offdiag(func=sns.scatterplot, s=200, marker='\*', palette='flag')



# Оптимальное число кластеров

##### **Выясним, какое количество кластеров будет оптимальным.**

Целевая функция метода k-средних уменьшается с увеличением количества кластеров. Если у каждого объекта кластер отдельный, то внутрикластерное расстояние равно нулю. Так мы минимизируем целевую функцию, но смысла в такой кластеризации нет: кластеров по сути не будет.

В нашей задаче признаков только три. Поэтому мы получили 6 парных проекций графика pairplot, дающих полное представление о данных. В других задачах признаков могут быть десятки и сотни, и графиком pairplot их охватить уже сложно. Также в наших данных изначально была структура, то есть видны скопления точек. В результате работы алгоритма они-то и стали кластерами.

### Метод локтя

Данные не всегда разделены чётко. Поэтому есть другой способ поиска числа кластеров — метод локтя (англ. elbow method). Своё название получил неслучайно: по форме его график напоминает согнутую в локте руку. Оптимальное количество кластеров определяется условным «локтем». Чтобы построить график метода, нужно составить список из значений целевой функции для разного количества кластеров: от 1 до 10 (реже 20). Для этого обучим модель несколько раз и сохраним значения целевой функции каждой модели в список distortion (англ. «искажение»):

Скопировать кодPYTHON

distortion = []

K = range(1, 8)

for k in K:

model = KMeans(n\_clusters=k, random\_state=12345)

model.fit(data)

distortion.append(model.inertia\_)

Отобразим полученный список на графике:

Скопировать кодPYTHON

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

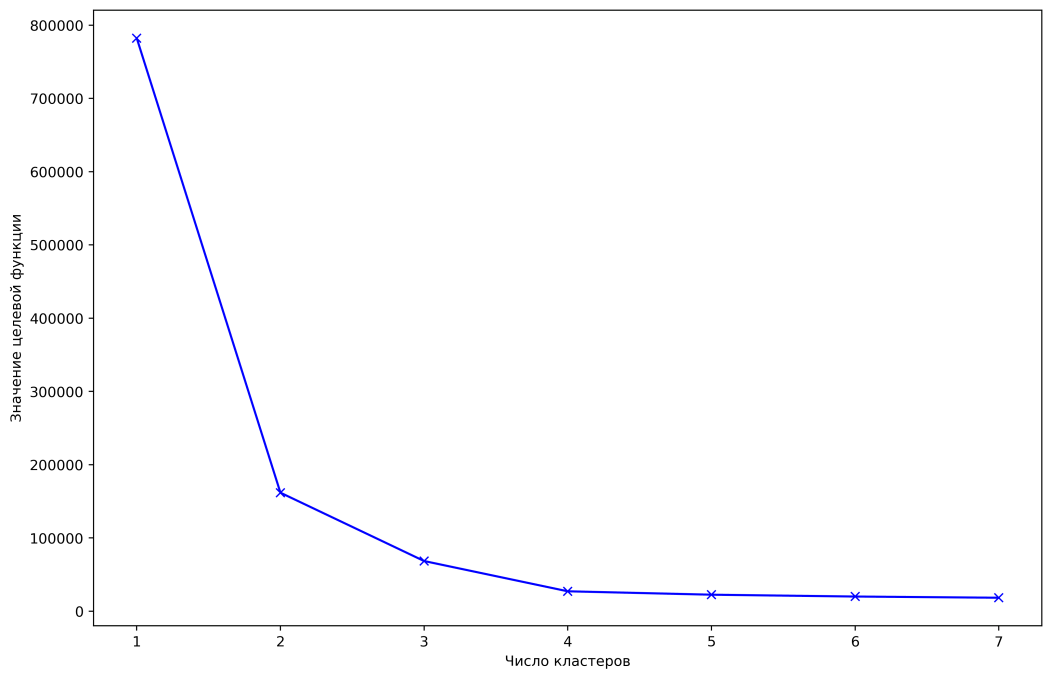
plt.figure(figsize=(12, 8))

plt.plot(K, distortion, 'bx-')

plt.xlabel('Число кластеров')

plt.ylabel('Значение целевой функции')

plt.show()



Это и есть график метода локтя: значение целевой функции сначала резко уменьшается, а затем выходит на плато. Этот момент перехода как раз отражает оптимальное количество кластеров.

На этом графике плато начинается после четвёртого кластера. Причём неплохие результаты также показывают второй и третий кластеры: после них скачок вниз целевой функции не очень большой.

Перейдём к нашей задаче и посчитаем значения целевой функции для моделей, обученных на разном количестве кластеров.

#### Задачи

1.

Выведите на экран значения целевой функции для количества кластеров — от 1 до 7. При обучении примените параметр *random\_state=12345*.

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans

data = pd.read\_csv('/datasets/segments.csv')

K = range(1, 8)

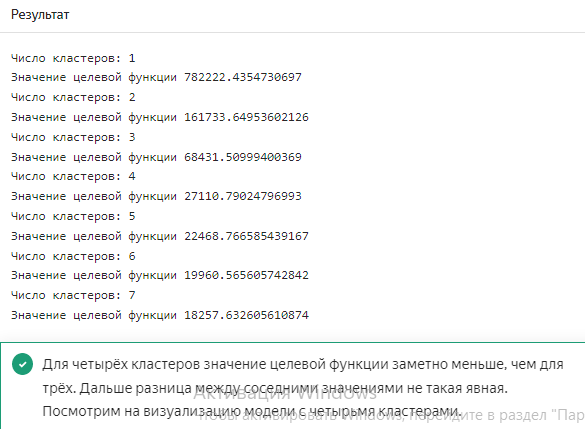
for k in K:

model = KMeans(n\_clusters=k, random\_state=12345)

model.fit(data)

print('Число кластеров:', k)

print('Значение целевой функции', model.inertia\_)



2.

Обучите модель для четырёх кластеров. Центроиды укажите так: ['0 centroid', '1 centroid', '2 centroid', '3 centroid']. Постройте диаграмму *pairplot* с полученными центроидами и заливкой для модели. При обучении примените параметр *random\_state=12345*.

import pandas as pd

import seaborn as sns

from sklearn.cluster import KMeans

data = pd.read\_csv('/datasets/segments.csv')

# Обучение модели для 4-х кластеров

model = KMeans(n\_clusters=4, random\_state=12345)

model.fit(data)

centroids = pd.DataFrame(model.cluster\_centers\_, columns=data.columns)

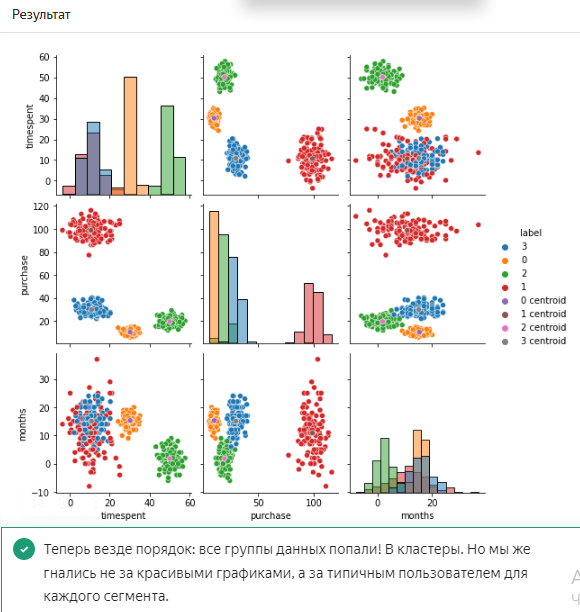
data['label'] = model.labels\_.astype(str)

centroids['label'] = ['0 centroid', '1 centroid', '2 centroid', '3 centroid']

data\_all = pd.concat([data, centroids], ignore\_index=True)

# Построение графика

sns.pairplot(data\_all, hue='label', diag\_kind='hist')



3.

Обучите модели для трёх и четырёх кластеров. Выведите на экран округлённые центроиды полученных моделей. При обучении примените параметр *random\_state=12345*.

import pandas as pd

from sklearn.cluster import KMeans

import numpy as np

data = pd.read\_csv('/datasets/segments.csv')

# Обучение модели для 3-х кластеров

model = KMeans(n\_clusters=3, random\_state=12345)

model.fit(data)

print("Типичные пользователи сегментов для 3-х кластеров:")

print(np.round(model.cluster\_centers\_))

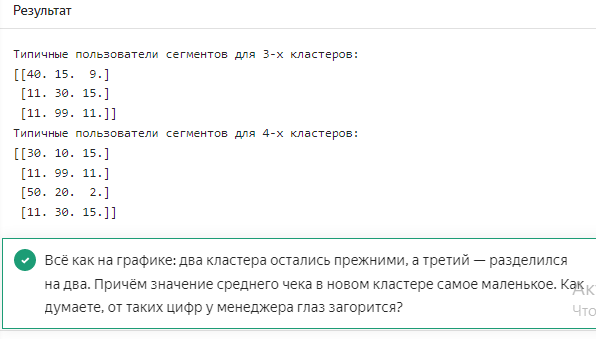
# Обучение модели для 4-х кластеров

model = KMeans(n\_clusters=4, random\_state=12345)

model.fit(data)

print("Типичные пользователи сегментов для 4-х кластеров:")

print(np.round(model.cluster\_centers\_))



### Поиск структуры в данных

##### **Кластеризацию можно применить и к задачам с разметкой. Она позволит увидеть в данных структуру и понять, какие признаки важнее.**

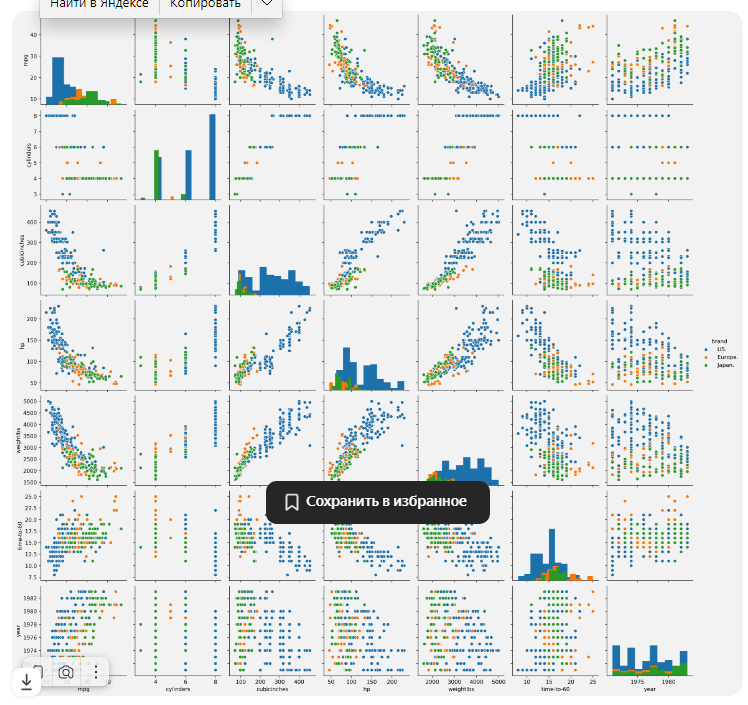
Рассмотрим новую задачу. Перед вами данные от автопроизводителей. Нужно их исследовать и разделить модели автомобилей на кластеры.

### Описание данных

Каждый объект в наборе данных — это характеристики модели автомобиля:

* mpg — потребление топлива (сколько миль проедет ваша машина на галлоне топлива),
* cylinders — количество цилиндров,
* cubicinches — объём двигателя (кубические дюймы),
* hp — мощность двигателя (лошадиные силы),
* weightlbs — вес автомобиля (фунты),
* time-to-60 — время разгона до 60 миль/час,
* year — год выпуска,
* brand — страна-производитель автомобиля.

В датасете известен целевой признак, по которому модели автомобилей делятся по кластерам. Это столбец brand, который принимает значения: [' US.', ' Europe.', ' Japan.']. Распределение автомобилей по странам выглядит так:



Опишем некоторые инсайты:

* На всех графиках для количества цилиндров выделяются три полосы точек.
* В паре признаков «количество цилиндров — расход топлива» американские автомобили заметно отделены от остальных.
* Признаки «год выпуска» и «время разгона» выглядят бесполезными: точки распределены по всей оси для каждой страны.
* На проекции признаков «мощность двигателя» и «вес автомобиля» видно, что дисперсия и среднее значение для американских автомобилей выше, а показатели японских и европейских близки.

Чтобы найти новые инсайты, обучим алгоритм *k*-средних без столбца о стране-производителе.

#### Задачи

1.

Постройте график метода локтя для количества кластеров от 1 до 10. При обучении модели примените параметр *random\_state=12345*.

Составьте список distortion значений целевой функции для количества кластеров от 1 до 10. При этом используйте параметр random\_state=12345. Для полученных значений целевой функции постройте график метода локтя размером 12 на 8. Ось X назовите «Число кластеров», а ось Y — «Значение целевой функции».

from sklearn.cluster import KMeans

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

distortion = []

data = pd.read\_csv('/datasets/cars.csv')

K = range(1, 11)

for i in K:

kmeans = KMeans(n\_clusters=i, random\_state=12345)

kmeans.fit(data)

distortion.append(kmeans.inertia\_)

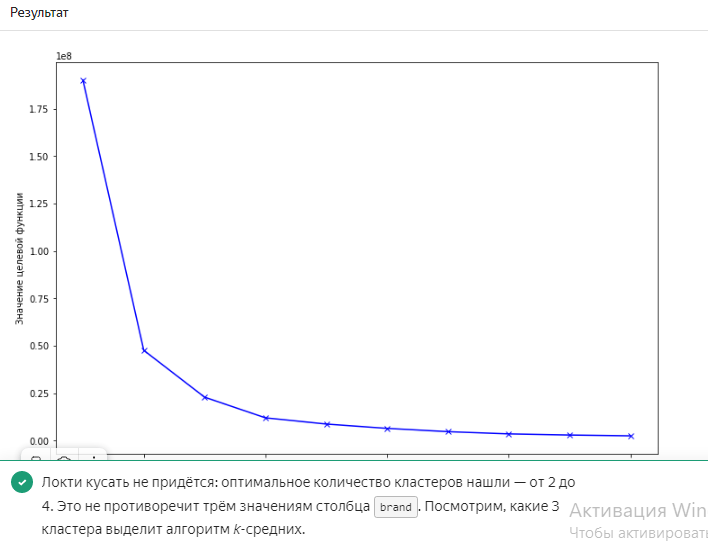
plt.figure(figsize=(12, 8))

plt.plot(K, distortion, 'bx-')

plt.xlabel('Число кластеров')

plt.ylabel('Значение целевой функции')

plt.show()



2.

Постройте диаграмму pairplot для модели с тремя кластерами без отмеченных центроидов. При обучении модели примените параметр random\_state=12345.

Из-за особенностей версий *seaborn* нужно указать список признаков в функции pairplot(): vars=data.columns[:-1]. Последний признак — это номер кластера, его отображать не надо.

import pandas as pd

from sklearn.cluster import KMeans

import seaborn as sns

data = pd.read\_csv('/datasets/cars.csv')

model = KMeans(n\_clusters=3, random\_state=12345)

model.fit(data)

centroids = pd.DataFrame(model.cluster\_centers\_, columns=data.columns)

# Добавление столбца с номером кластера

data['label'] = model.labels\_.astype(str)

#centroids['label'] = ['0 centroid', '1 centroid', '2 centroid']

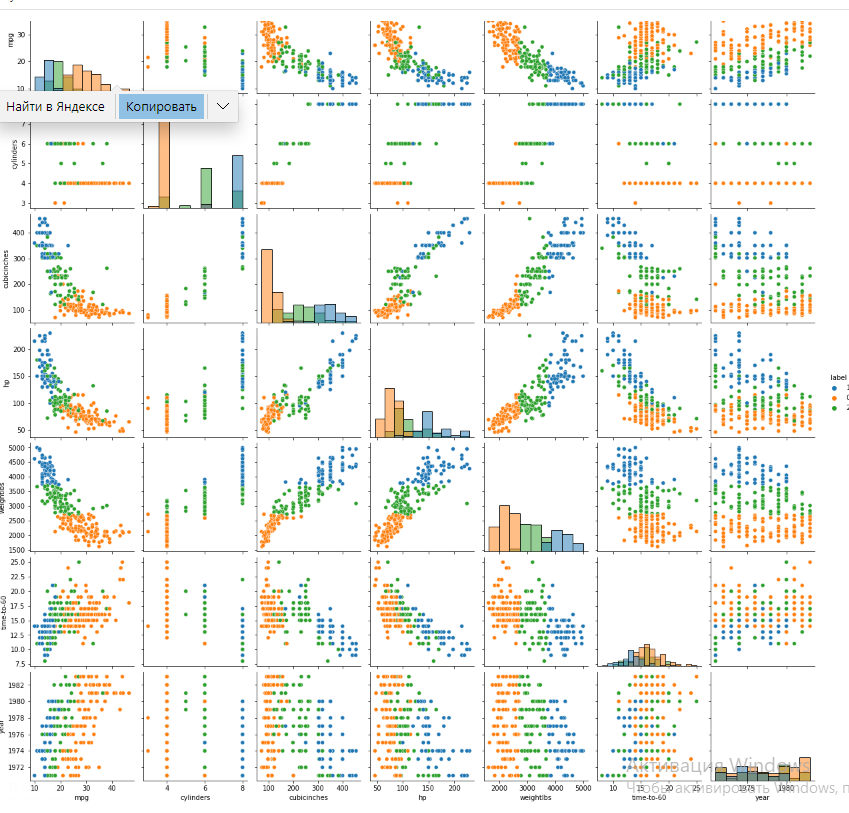
# Сброс индекса понадобится дальше

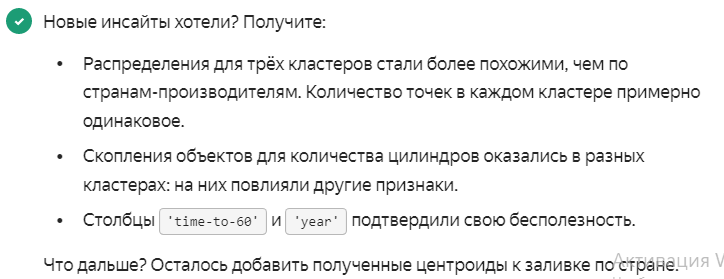
#data\_all = pd.concat([data, centroids], ignore\_index=True)

# Построение графика

sns.pairplot(data, hue='label', vars=data.columns[:-1], diag\_kind='hist')

# < напишите код здесь >





3.

Постройте диаграмму pairplot с заливкой по столбцу brand. Обучите модель с тремя кластерами на данных без столбца brand. Добавьте на график полученные центроиды. При обучении модели примените параметр random\_state=12345.

import pandas as pd

from sklearn.cluster import KMeans

import seaborn as sns

import numpy as np

data\_full = pd.read\_csv('/datasets/cars\_label.csv')

data = data\_full.drop(columns=['brand'])

# Обучение модели

model = KMeans(n\_clusters=3, random\_state=12345)

model.fit(data)

# Дополнительный слой для центроидов

centroids = pd.DataFrame(model.cluster\_centers\_, columns=data.columns)

# Сформируйте в дополнительной таблице новый столбец 'brand' в качестве заглушки

centroids['brand'] = 4

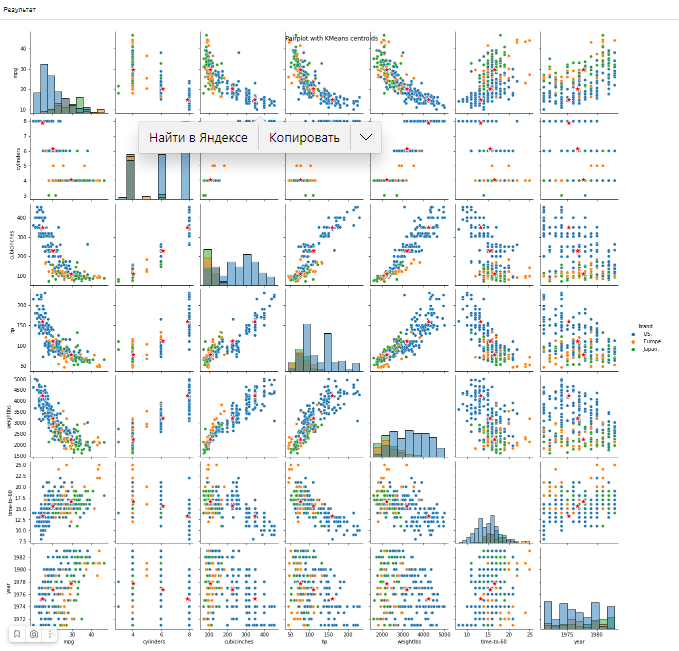
# Построение графика

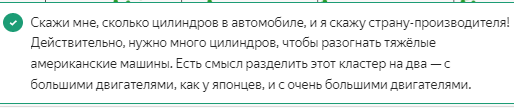
g = sns.pairplot(data\_full, hue='brand', diag\_kind='hist')

g.fig.suptitle('Pairplot with KMeans centroids')

g.data = centroids

g.map\_offdiag(func=sns.scatterplot, s=200, marker='\*', palette='flag')





## Поиск аномалий

##### **Вы научитесь находить в данных объекты, непохожие на большинство других.**

#### Чему вы научитесь

* Узнаете, что такое аномалии.
* Научитесь искать выбросы в одномерных и многомерных данных.
* Разберётесь с алгоритмом изоляционного леса.

#### Сколько времени это займёт

6 уроков по 10 минут

#### Постановка задачи

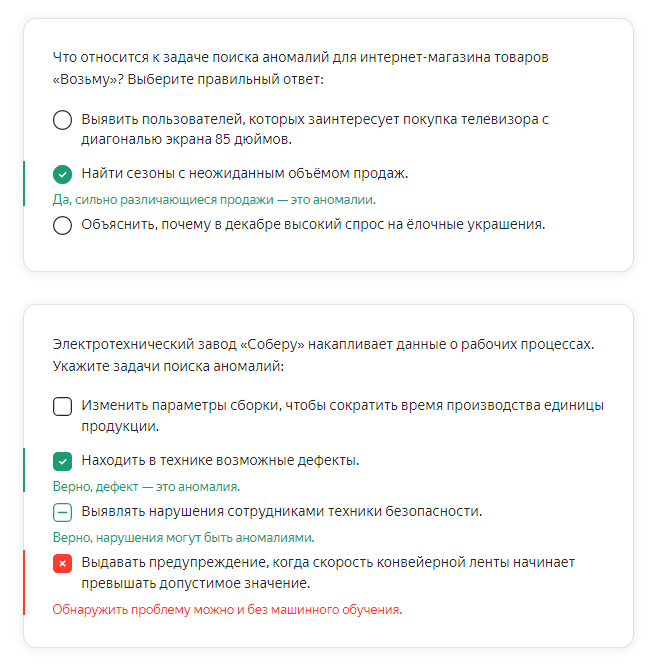
Найдёте аномалии в данных интернет-магазина офисных товаров.

### Аномалии

###### **Аномалии, или выбросы (англ. anomalies; outliers) — это объекты с «ненормальным» поведением, то есть отклоняющимся от общего тренда.**

Выбросы указывают на проблему в данных или на что-то нестандартное. Например, обнаружение подозрительных банковских операций: Иван из Москвы, радующий себя только шавермой по выходным, неожиданно «купил» очень дорогой смартфон в Ханое. Или прогнозирование природных аномалий: если можно было предсказать тёплую зиму, то Наташа сэкономила бы деньги на покупке нового пуховика. («Лучше бы пальто взяла или сразу босоножки!»)

Поскольку выбросы непредсказуемы, примеров аномальных объектов при обучении мало или вовсе нет.

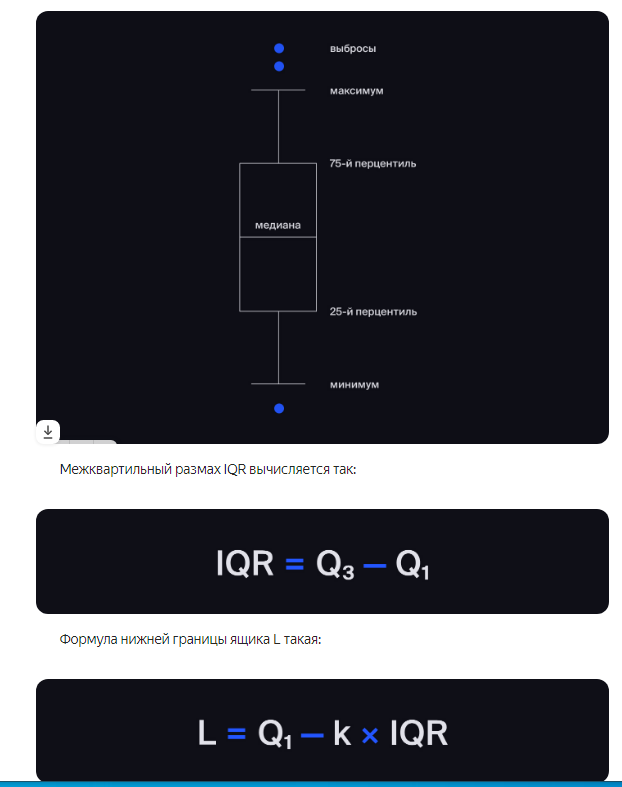


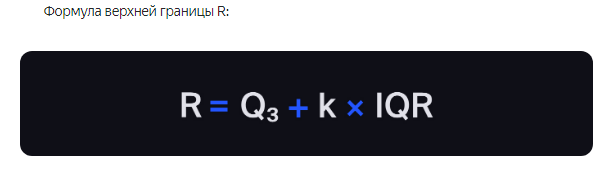
### Диаграмма размаха

##### **Узнаем, как находить выбросы в одномерных данных.**

Представим, что значения признака — это мешок чисел. Нужно найти числа, сильно отличающиеся от остальных. Для этого сравним их с медианой на **диаграмме размаха** (англ. boxplot), или «ящике с усами».

Повторим обозначения. Верхняя и нижняя границы ящика — третья и первая квартиль (75% и 25% значений). Посередине обозначена медиана (50% значений). «Усы» простираются вверх и вниз от границ ящика на расстояние, равное 1.5 **межквартильным размахам** (IQR, от англ. interquartile range). Выбросы указаны за пределами усов — максимумом и минимумом.

.



Чем больше коэффициент k, тем меньше объектов будут считаться выбросами. Обычно его указывают равным 1.5.

В этой теме будем работать с данными интернет-магазина офисных товаров. В датасете содержится информация о 9994 продажах. Построим ящик с усами для признака *'Sales'* (англ. «продажи»):

Скопировать кодPYTHON

import pandas as pd

import matplotlib.pyplot as plt

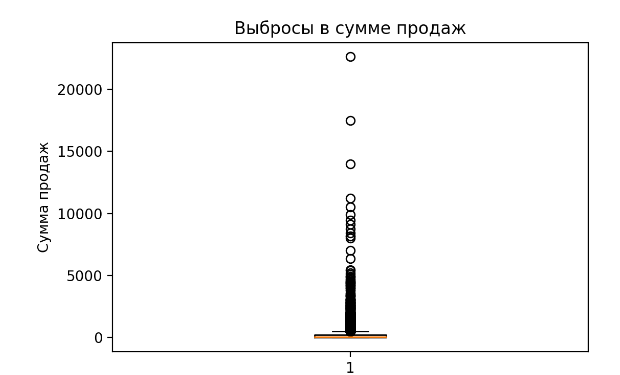
df = pd.read\_csv('/datasets/sales.csv')

plt.boxplot(df['Sales'].values)

plt.ylabel('Сумма продаж')

plt.title("Выбросы в сумме продаж")

plt.show()



Средний чек бо́льшей части покупок — 500–1000 $. Продажи от 5 до 20 тысяч долларов — аномалии (они отмечены кругами).

Диаграмма даёт информацию обо всех выбросах. Она хранится в записи "fliers" (на англ. синоним «аномалии») внутри объекта *boxplot*. Вызовом функции get\_data() из объектов получим числа. Нужные значения отделены индексами.

boxplot = plt.boxplot(df['Sales'].values)

outliers = list(boxplot["fliers"][0].get\_data()[1])

print("Выбросов в продажах: ", len(outliers))

Получили:

Выбросов в продажах: 1167

#### Задачи

Найдите в датасете аномалии по признаку *'Profit'*. Из ящика с усами возьмите список аномалий и запишите результат в переменной *outliers*.

Отфильтруйте исходный датафрейм функцией *isin()* и сохраните список объектов с аномалиями в переменной df\_outliers.

Выведите количество аномалий (уже в прекоде).

import pandas as pd

import matplotlib.pyplot as plt

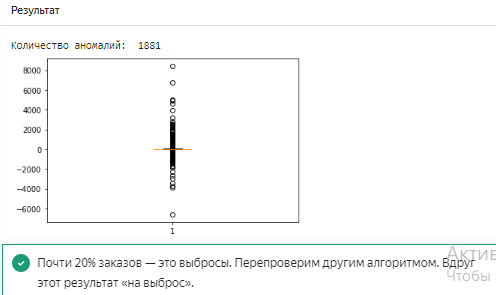
df = pd.read\_csv('/datasets/sales.csv')

boxplot = plt.boxplot(df['Profit'].values)

outliers = list(boxplot["fliers"][0].get\_data()[1])

df\_outliers = df[df.Profit.isin(outliers)]

print("Количество аномалий: ", len(df\_outliers))



### Изоляционный лес

##### **Перейдём к многомерным данным. Познакомимся с другим алгоритмом поиска аномалий — изоляционным лесом (англ. isolation forest).**

Это ансамблевый метод. Поэтому, как и в случайном лесу, его оценки строятся на усреднённых оценках множества деревьев решений. В узлах деревьев находятся **решающие правила**. Они определяют, к какой ветви отнести объект.

Изоляционный лес основан на том, что аномальные объекты можно изолировать от остальных небольшим количеством решающих правил.



Изоляционное дерево строится как решающее дерево. А вот решающие правила в нём выбираются случайным образом. Объекты на маленькой глубине, которые можно легко изолировать, считаются аномальными, а остальные — нормальными. Оценки аномальности собираются от всех деревьев и усредняются.

Посмотрим, как изоляционный лес можно обучить в библиотеке *sklearn*. Импортируем класс *IsolationForest()* из модуля *sklearn.ensemble:*

Скопировать кодPYTHON

from sklearn.ensemble import IsolationForest

Создадим модель. Пропишем количество деревьев в параметре *n\_estimators*. Чем их больше, тем точнее результаты:

Скопировать кодPYTHON

isolation\_forest = IsolationForest(n\_estimators=100)

Для поиска аномалий в одномерных данных преобразуем *df['Sales']* в двухмерный массив:

Скопировать кодPYTHON

sales = df['Sales'].values.reshape(-1, 1)

Выбор аномалий по одному признаку не даст представления обо всём датасете. Изоляционный лес найдёт выбросы по нескольким признакам.

Например, объединим данные о продажах и прибыли. Здесь дополнительные преобразования не нужны.

Скопировать кодPYTHON

data = df[['Sales', 'Profit']]

Дальнейшее обучение модели одинаково и для одномерных, и для многомерных данных.

Функцией *fit()* обучим модель на данных о продажах и прибыли:

Скопировать кодPYTHON

isolation\_forest.fit(data)

Вызовом функции **decision\_function()** (англ. «принятие решений») узнаем, как модель оценила объекты:

Скопировать кодPYTHON

anomaly\_scores = isolation\_forest.decision\_function(data)

Оценки аномальности находятся в промежутке от -0.5 до 0.5. Чем ниже оценка, тем выше вероятность того, что перед вами аномалия.

Чтобы посчитать количество аномалий, вызовом функции *predict()* классифицируем объекты на нормальные и выбросы. Если объект получит класс «1», объект нормальный; если «-1» — перед вами выброс.

Скопировать кодPYTHON

estimator = isolation\_forest.predict(data)

Если оценки аномальности не нужны, функцией *fit\_predict()* можно сразу обучить модель и получить классификацию:

Скопировать кодPYTHON

estimator = isolation\_forest.fit\_predict(data)

#### Задача

Обучите модель изоляционного леса и вычислите количество аномалий по признакам:

* продаж df['Sales'];
* прибыли df['Profit'].

Определите, какие объекты — выбросы, и запишите их в переменной outliers.

Выведите длину списка (уже в прекоде).

import pandas as pd

from sklearn.ensemble import IsolationForest

df = pd.read\_csv('/datasets/sales.csv')

data = df[['Sales', 'Profit']]

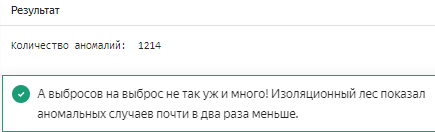
# < напишите код здесь >

isolation\_forest = IsolationForest(n\_estimators=100, random\_state=12345)

estimation = isolation\_forest.fit\_predict(data)

outliers = [e for e in estimation if e == -1]

print("Количество аномалий: ", len(outliers))



### KNN для поиска аномалий

##### **Найти аномалии в многомерных данных можно и другим способом — методом ближайших соседей.**

**Метод ближайших соседей** (англ. k-Nearest Neighbors, KNN) работает так: каждый объект датасета принимает за вектор и выбросы ищет в многомерном пространстве. Чем дальше объект от своих соседей, тем выше вероятность его аномальности.

Класс KNN() находится в библиотеке **PyOD** (англ. Python toolkit for detecting outlying objects, «инструмент для поиска аномалий в Python»). Импортируем его из модуля pyod.models.knn:

from pyod.models.knn import KNN

Вызовом функции fit() обучим модель на выборке:

model = KNN()

model.fit(data)

Когда модель обучена, можно перейти к поиску аномалий в наборе данных. Вызовем функцию predict():

predictions = model.predict(data)

Функция predict() вернёт список, где «1» означает аномалию, а «0» — её отсутствие.

#### Задача

Моделью KNN и изоляционным лесом найдите выбросы в данных с переменными *'Sales'* и *'Profit'*. Выясните, сколько аномалий совпало.

Напечатайте на экране два варианта количества выбросов и число совпавших аномалий. Формат вывода указан в прекоде.

Подсказка

Допишите код поиска совпавших выбросов:

Скопировать кодPYTHON

*# оценки модели KNN*

estimation\_knn = *# < напишите код здесь >*

*# оценки модели изоляционного леса*

estimation\_iforest = *# < напишите код здесь >*

*# совпавшие оценки*

(estimation\_knn & estimation\_iforest).sum()

import pandas as pd

from pyod.models.knn import KNN

from sklearn.ensemble import IsolationForest

RANDOM\_STATE = 42

df = pd.read\_csv('/datasets/sales.csv')

data = df[['Sales', 'Profit']]

# < напишите код здесь >

model = KNN()

model.fit(data)

estimation\_knn = model.fit\_predict(data) == 1

outliers\_knn = estimation\_knn.sum()

print("Количество аномалий (KNN): ", outliers\_knn)

# < напишите код здесь >

isolation\_forest = IsolationForest(n\_estimators=100, random\_state = RANDOM\_STATE)

isolation\_forest.fit(data)

estimation\_iforest = isolation\_forest.predict(data) == -1

outliers\_iforest = estimation\_iforest.sum()

print("Количество аномалий (изоляционный лес): ", outliers\_iforest)

print("Совпало: ", (estimation\_knn & estimation\_iforest).sum() )

