In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.metrics import adjusted_rand_score
from sklearn.cluster import KMeans
```

Загрузка датасета

Генерарация изотропных гауссовых "сгущений" данных для кластеризации.

sklearn.datasets.make_blobs
(
n_samples=100, # общее количество элементов, поровну разделенное между кластерами.
# Если это массив, каждый элемент последовательности указывает количество выборок на кластер.
n_features=2, # Количество параметров для каждого образца
*,
centers=None, # Количество создаваемых центров или фиксированные местоположения центров. Если n_samples — это целое число, а центры — None, генерируются 3 центра.
Если n_samples массив, центры должны быть либо None, либо массив длиной, равной длине n_samples.
cluster_std=1.0, # Стандартное отклонение кластеров
center_box=(-10.0, 10.0), # Ограничивающая рамка для каждого центра кластера, когда центры генерируется случайным образом.
shuffle=True, # Перемешать
random_state=None, # Определяет генерацию случайных чисел для создания набора данных.
return_centers=False # Вернуть центры каждого кластера
)
Возвращает:

X: ndarray формы (n_samples, n_features) # Сгенерированные образцы.

y: ndarray формы (n_samples,) # Целочисленные метки членства в кластере каждого образца.

In [2]:
```
X1, y1 = make_blobs(n_samples=250, n_features=2, centers=8, random_state=0)
print(X1.shape); print(X1)
print(y1.shape); print(y1)
```

```
(250, 2)
[[  2.43169305  -0.20173713]
 [ -1.10630261   7.5161316 ]
 [ -9.01846836  -8.07607558]
 [ -9.09679788  -9.23624387]
 [  2.18247693   8.51722541]
 [ -8.92672949  -8.83868248]
 [  3.54351972   2.79355284]
 [ -2.19270203   7.42541032]
 [ -8.11614851  -7.97831824]
 [-10.21191336  -9.82518173]
 [  3.2460247    2.84942165]
 [ -1.71185175   8.31694149]
 [  0.9845149    1.95211539]
 [  2.3535057    2.22404956]
 [ -0.55671702   8.53020916]
 [ -1.26527619   8.21461175]
 [  4.07861018   1.02883286]
 [  1.38093486   0.92949422]
 [ -8.6249828   -8.03690635]
 [  9.93963829  -2.79188941]
 [  0.58894326   4.00148458]
 [  3.93841822  -0.4500954 ]
 [  1.62011397   2.74692739]
 [ -1.38113635   7.53766914]
 [  9.54777157  -3.22208471]
 [ -8.19654641  -8.29165629]
 [  0.1631238    2.57750473]
 [  1.1312175    4.68194985]
 [  2.16145603   8.59019294]
 [ -7.4789945   -6.95939203]
 [ -0.73000011   6.25456272]
 [  0.46546494   3.12315514]
 [  6.11494247  -0.41522522]
 [  6.41302226   0.92755285]
 [ -7.45068477  -7.76141306]
 [ -2.63128735   2.97004734]
 [  3.23404709   0.71773882]
 [  2.45127423  -0.19539785]
 [  1.17583755   7.70428428]
 [  5.81881865   0.73882656]
 [  9.11558819  -0.07444613]
 [ -9.2378318   -8.77164797]
 [  1.64164854   0.15020885]
 [ -2.84281142   2.45629766]
 [  2.63185834   0.6893649 ]
 [  2.2635425    1.8743027 ]
 [  8.64516765  -2.81219674]
 [ -0.33807687   8.15267823]
 [  3.09361241   9.19643387]
 [  7.24257074  -0.26667676]
 [ -8.37997864  -8.30801755]
 [ -1.79111725   8.25151006]
 [ -0.65392827   4.76656958]
 [  0.63633194   4.25441021]
 [ -2.17052242   0.69447911]
```

```
[  0.56400993   1.33705536]
[-10.59568546  -8.79686864]
[ -2.27165884   2.09144372]
[  9.21162881  -2.4384749 ]
[  1.42013331   4.63746165]
[  0.30380963   3.94423417]
[  0.08848433   2.32299086]
[  2.66934689   1.81987033]
[  7.96672836  -0.67303894]
[  2.23672398   8.3968253 ]
[  4.84899002  -0.89393661]
[  1.37861172   3.61897724]
[  8.87380618  -1.96111374]
[  2.1275544    8.86822558]
[ -0.08564348   9.31223072]
[  8.1464294   -3.06184738]
[  9.37098018  -1.74821594]
[  3.92282648   1.80370832]
[  9.79714623  -2.24274754]
[ -0.42724442   3.57314599]
[ -2.40443821   8.61665812]
[  2.18217961   1.29965302]
[  1.74371499   0.953829  ]
[  5.15048986   2.23744919]
[  9.96307337  -1.02932339]
[  2.30393731   5.7722556 ]
[  8.55365082  -3.14416261]
[  2.52092996  -0.63858003]
[ -0.57748321   3.0054335 ]
[  5.62759709   1.45807731]
[  0.34194798   3.94104616]
[ -9.75843677  -6.95598593]
[  2.82746994   9.36448471]
[ -7.71668282 -10.9130331 ]
[ -0.09592421   8.91507861]
[ -0.63762777   4.09104705]
[  3.97820955   2.37817845]
[ -2.43972624   4.03489855]
[  3.17532852   1.18421792]
[  2.06576754   2.68353415]
[  5.56676722  -0.55011294]
[ -1.84456981   7.78289272]
[ -2.67231668   7.34214013]
[ -0.46192781   7.36904092]
[  1.19404184   2.80772861]
[  5.46980722   0.73460225]
[  9.67230156  -5.10376238]
[  1.84070628   3.56162231]
[  8.62084663  -2.722123  ]
[ -5.88305478  -8.33133867]
[  2.21881515   9.65303463]
[ -0.33887422   3.23482487]
[ -2.75233953   3.76224524]
[ -2.26646701   4.46089686]
[  8.58120536  -0.79479257]
[  2.22985471   8.78880467]
```

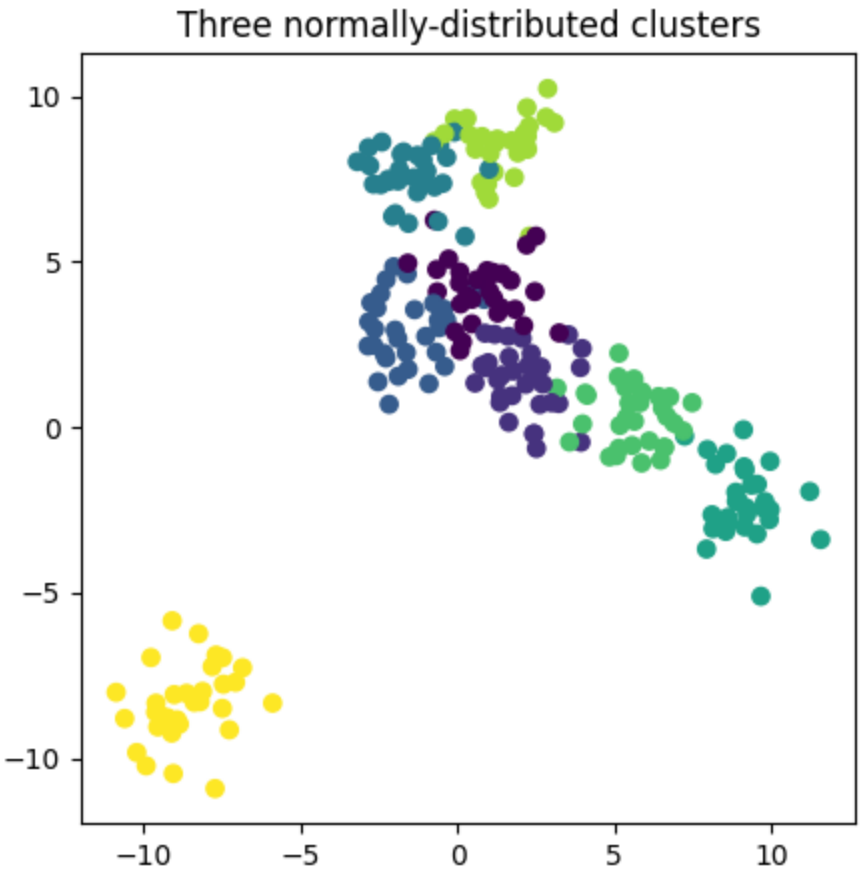
```
[ -0.75511346   3.74138642]
[  5.35484495   1.19825669]
[  0.78478252   1.86706037]
[ -0.73371185   8.63565468]
[  8.96236904  -2.23376946]
[  0.94808785   4.7321192 ]
[  1.81830683   7.54732075]
[  5.07035684  -0.85989308]
[  0.08080352   4.69068983]
[ -0.07228289   2.88376939]
[  0.76223728   7.39603578]
[ -3.18453558   8.02423861]
[  7.19903261  -0.11155079]
[  2.72756228   1.3051255 ]
[  2.29039633   9.09415736]
[  0.06897171   4.35573272]
[ -0.40026809   1.83795075]
[  5.18220716   0.05670908]
[  6.5816891   -0.61104656]
[  0.79157917   8.78183712]
[ -1.84557184   7.59753829]
[ -1.2725819    7.09742911]
[  1.78726415   1.70012006]
[  6.60775374  -0.60598225]
[  9.13955365  -1.25342582]
[  5.64384727   0.18304888]
[  1.0427873    4.60625923]
[ -1.62535654   2.25440397]
[ -2.56114686   3.59947678]
[ -2.78905279   7.89872201]
[  0.62835793   4.4601363 ]
[  9.13730551  -1.19427826]
[ -9.48804208  -8.68173163]
[  6.53295791   0.58166928]
[  8.8883754   -2.23681803]
[ -0.6700734    2.26685667]
[  0.10547293   3.72493766]
[ -1.36023052   3.5529137 ]
[  2.88088608  10.23152207]
[ -1.88089792   1.54293097]
[  3.99143121   0.09992439]
[ -2.67437267   2.48006222]
[  2.41163392   1.60423683]
[  8.11589995  -2.64346187]
[  0.89011768   1.79849015]
[  7.48263569   0.74212615]
[  2.47034915   4.09862906]
[  9.15509116  -3.01134783]
[  0.38978665   8.82674997]
[ -8.85494937  -8.96714197]
[  5.14666315  -0.63617901]
[  3.57893653  -0.44460845]
[ -9.59732071  -8.33526876]
[  1.9281815    8.28925767]
[ -2.52711936   1.37311116]
[  0.85624076   3.86236175]
```

```
[  0.24622877   5.76547499]
[  1.06905386   7.75044055]
[  1.28535145   1.43691285]
[ -7.26014196  -9.13983282]
[ -1.09174924   8.06764105]
[  9.76699699  -2.44727356]
[ -9.04814302 -10.45885529]
[ -2.33031368   2.22833248]
[  1.73171622   8.65399457]
[  2.17541104   8.57085836]
[  8.22800184  -1.12002433]
[ -2.02493646   4.84741432]
[ -0.90167256   1.31582461]
[  1.70536064   4.43277024]
[  6.47781523  -0.99272501]
[  7.93899674  -3.67788713]
[ 11.22916752  -1.9410763 ]
[  0.57826207   8.40154347]
[ -2.43711503   7.32864366]
[  1.41942144   1.57409695]
[ -7.80787289  -7.22797518]
[  1.00745947   6.89545858]
[ -7.48293199  -8.49162981]
[ -8.24037471  -6.23637044]
[  2.11567076   3.06896151]
[  9.5595989   -1.72232579]
[  5.39357813   0.2975429 ]
[ -6.84040616  -7.26301961]
[ -1.98243652   2.93536142]
[  0.89404568   7.09502665]
[  0.96566224   7.35251225]
[ -1.89608585   2.67850308]
[  5.87313131  -1.07881671]
[ -1.55726874   6.15945621]
[ -2.06162003   6.36903569]
[ -1.59514562   4.63122498]
[ -7.06595075  -7.70428194]
[  1.36069966   0.74802912]
[-10.86789888  -8.00592959]
[  1.23078427   8.605886  ]
[ -0.40764723   8.86741456]
[ 11.57717191  -3.39118545]
[ -1.8856928    7.4381882 ]
[  4.13639494   0.96517887]
[ -9.91149049 -10.2260387 ]
[  1.27496045   8.7062257 ]
[  1.01105318   7.79320286]
[  6.67613203   0.32843981]
[  2.50904929   5.7731461 ]
[ -0.7271909    7.25967205]
[ -2.20420078   7.48947824]
[  5.88399574   1.07173517]
[  6.90301016   0.12451259]
[ -0.60604519   3.23660991]
[  0.4666179    3.86571303]
[  1.05177678   8.2784661 ]
```

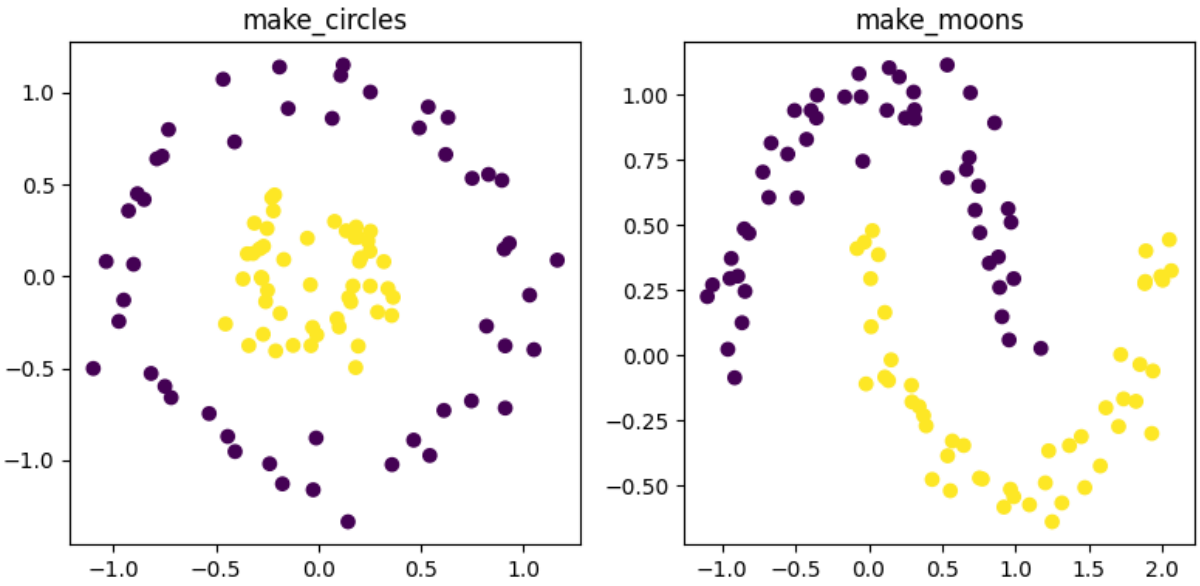
```
 [ -1.97385315   6.45209606]
 [  9.16271455  -1.31099691]
 [  1.28933778   3.44969159]
 [  5.12980049   1.52115912]
 [ -2.83119417   8.44583939]
 [  6.76634914   0.91786338]
 [ -1.00362735   2.74633593]
 [ -1.57671974   4.95740592]
 [  0.30626276   9.3321806 ]
 [  9.23108376  -2.61805682]
 [ -0.27652528   5.08127768]
 [  3.00251949   0.74265357]
 [  1.65209057   2.12010873]
 [ -2.81976092   3.18493313]
 [ -1.56618683   1.74978876]
 [  1.0220286    4.11660348]
 [  0.88214412   2.84128485]
 [ -0.60812425   6.21850397]
 [ -7.68401856  -6.88244994]
 [  1.15369622   3.90200639]
 [ -9.08209554  -5.84496033]
 [  2.20656076   5.50616718]
 [ -0.82199704   8.51236805]
 [  9.96702836  -2.49074306]
 [ -9.60921412  -8.60735737]
 [ -0.96833118   7.73730963]
 [ -9.53978322  -9.05053137]]
(250,)
[1 3 7 7 6 7 1 3 7 7 0 3 1 1 3 3 5 1 7 4 0 1 1 3 4 7 0 0 6 7 0 0 5 5 7 2 1
 1 6 5 4 7 1 2 1 1 4 3 6 4 7 3 0 2 2 1 7 2 4 0 0 0 1 4 6 5 0 4 6 6 4 4 1 4
 2 3 1 1 5 4 6 4 1 2 5 0 7 6 7 3 0 1 2 5 1 5 3 3 3 1 5 4 0 4 7 6 2 2 2 4 6
 2 5 1 6 4 0 6 5 0 0 6 3 5 1 6 0 2 5 5 6 3 3 1 5 4 5 0 2 2 3 0 4 7 5 4 2 0
 2 6 2 5 2 1 4 1 5 0 4 6 7 5 5 7 6 2 2 3 6 1 7 3 4 7 2 6 6 4 2 2 0 5 4 4 6
 3 1 7 6 7 7 0 4 5 7 2 6 6 2 5 3 3 2 7 1 7 6 6 4 3 5 7 6 3 5 0 3 3 5 5 2 0
 6 3 4 0 5 3 5 2 0 6 4 0 1 1 2 2 0 1 3 7 0 7 0 3 4 7 3 7]
```

In [3]:

```
plt.figure(figsize=(5,5))
plt.scatter(X1[:, 0], X1[:, 1], c=y1)
plt.title("Three normally-distributed clusters")
plt.show()
```



In [4]:
```
from sklearn.datasets import make_circles, make_moons

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))

X, Y = make_circles(noise=0.1, factor=0.3, random_state=0)
ax1.scatter(X[:, 0], X[:, 1], c=Y)
ax1.set_title("make_circles")

X, Y = make_moons(noise=0.1, random_state=0)
ax2.scatter(X[:, 0], X[:, 1], c=Y)
ax2.set_title("make_moons")

plt.tight_layout()
plt.show()
```



KMeans (scikit-learn)

sklearn.cluster.k_means(*X*, *n_clusters*, *, *sample_weight=None*, *init='k-means++'*, *n_init='auto'*, *max_iter=300*, *verbose=False*, *tol=0.0001*, *random_state=None*, *copy_x=True*, *algorithm='lloyd'*, *return_n_iter=False*)

X : {array-like, sparse matrix} of shape (n_samples, n_features)

Наблюдения для кластеризации. Необходимо отметить, что данные будет преобразован в порядок C, что приведет к копированию памяти если данные данные не являются C-непрерывными.

n_clusters : int

Количество образуемых кластеров, а также количество центроиды для генерации.

sample_weight : array-like of shape (n_samples,), default=None

Веса для каждого наблюдения в `X` . Если `None` , все наблюдения имеют равный вес. `sample_weight` не используется во время инициализация, если `init` является вызываемым или предоставленным пользователем массивом.

init : {'k-means++', 'random'}, callable or array-like of shape (n_clusters, n_features), default='k-means++'

Метод инициализации:

- `'k-means++'` : выбирает начальные центры кластеров для k-среднего разумная кластеризация для ускорения конвергенции. См. раздел Примечания в k_init для более подробной информации.
- `'random'` : выбирать `n_clusters` наблюдения (строки) случайным образом из данных для начальных центроидов.

- Если передается массив, он должен иметь форму `(n_clusters, n_features)` и дает начальные центры.
- Если передается вызываемый объект, он должен принимать аргументы `X`, `n_clusters` и случайное состояние и вернуть инициализацию.

n_init : ‘auto’ or int, default=”auto”

Сколько раз алгоритм k-средних будет выполняться с разными центроидные семена. Окончательные результаты будут лучшим результатом n_init последовательных запусков по инерции. Когда `n_init='auto'`, количество запусков зависит от значения init: 10 при использовании `init='random'` или `init` является вызываемым; 1 при использовании `init='k-means++'` или `init` является массивом.

max_iter : int, default=300

Максимальное количество итераций алгоритма k-средних для запуска.

verbose : bool, default=False

"Отладочный" режим.

tol : float, default=1e-4

Относительная толерантность к норме Фробениуса разницы в центрах кластеров двух последовательных итераций объявить конвергенция.

random_state : int, RandomState instance or None, default=None

Определяет генерацию случайных чисел для инициализации центроида. Использовать int, чтобы сделать случайность детерминированной.

copy_x : bool, default=True

При предварительном вычислении расстояний более точно центрировать данные в первую очередь. Если `copy_x` имеет значение True (по умолчанию), то исходные данные не модифицирован. Если значение равно False, исходные данные изменяются и возвращаются обратно. до того, как функция вернется, но могут быть небольшие численные различия. вводится путем вычитания, а затем добавления среднего значения данных. Обратите внимание, что если исходные данные не являются C-непрерывными, копия будет сделана, даже если `copy_x` неверно. Если исходные данные разрежены, но не в формате CSR, копия будет сделана, даже если `copy_x` неверно.

algorithm : {“lloyd”, “elkan”}, default=”lloyd”

Используемый алгоритм K-средних. Классический алгоритм в стиле EM: `"lloyd"`. `"elkan"` вариация может быть более эффективной для некоторых наборов данных с четко определенные кластеры, используя неравенство треугольника. Однако это более

требовательный к памяти из-за выделения дополнительного массива формы `(n_samples, n_clusters)` .

return_n_iter : bool, default=False

Возвращать или нет количество итераций.

Returns:

centroid : ndarray of shape (n_clusters, n_features)

Центроиды найдены на последней итерации k-средних.

label : ndarray of shape (n_samples,)

The `label[i]` это код или индекс центроида мое наблюдение наиболее близко к этому.

inertia : float

Окончательное значение критерия инерции (суммы квадратов расстояний до ближайший центроид для всех наблюдений в обучающем наборе).

best_n_iter : int

Количество итераций, соответствующих лучшим результатам. Возвращается только в том случае, если `return_n_iter` установлено значение `True` .

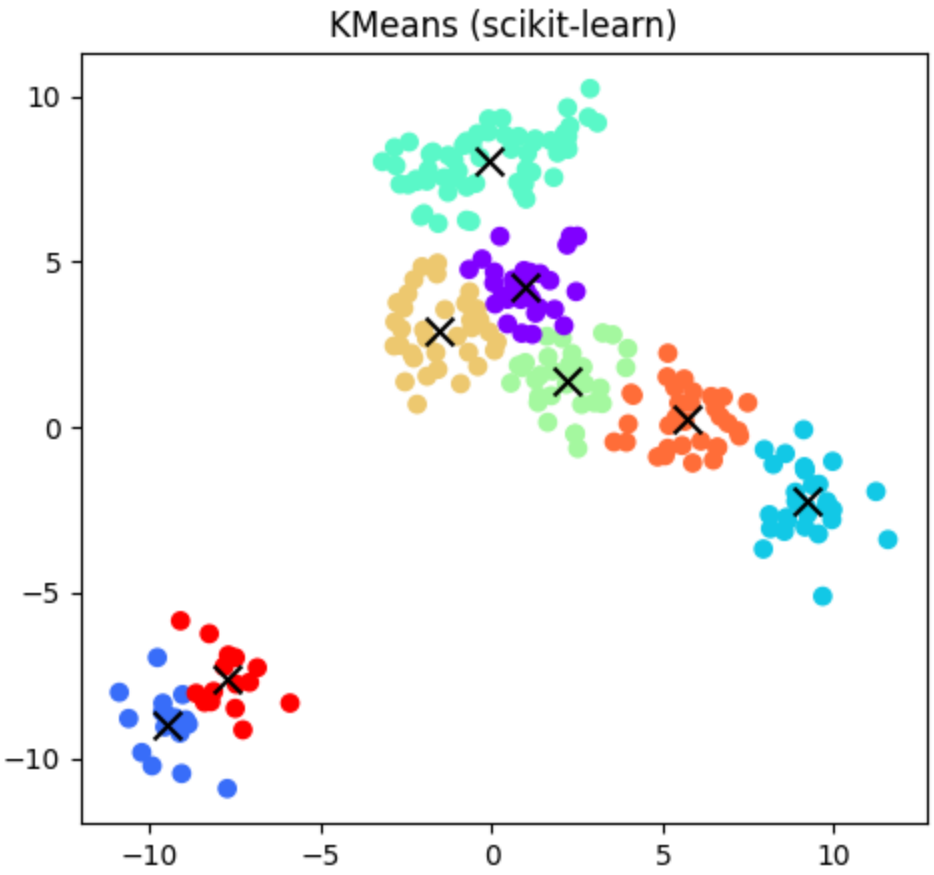
In [5]:
```
# n_clusters=8,      - количество кластеров
# n_init='auto',     - количество запусков зависит от значения init. Для 'k-means++'
# random_state=0     - seed
sk_kmeans = KMeans(n_clusters=8, n_init='auto', random_state=0)
# Проводим кластеризацию
sk_kmeans.fit(X1)
# Определяем какой элемент попадает в какой кластер
sk_kmeans_pred_res = sk_kmeans.predict(X1)
# Индекс Rand вычисляет меру сходства между двумя кластеризациями.
# рассматривая все пары образцов и подсчитывая пары, которые назначены
# в одном или разных кластерах в прогнозируемых и истинные кластеризации.
sk_kmeans_ari = adjusted_rand_score(y1, sk_kmeans_pred_res)
sk_kmeans_centroinds = sk_kmeans.cluster_centers_
print(f'Индекс Rand: {sk_kmeans_ari}', '', sep='\n')
print("Центры",sk_kmeans_centroinds, '', sep='\n')
print('Результат', sk_kmeans_pred_res, sep='\n')
```

```
Индекс Rand: 0.7098033382986403

Центры
[[ 0.98298252  4.24015671]
 [-9.46746174 -8.98416087]
 [ 9.20217726 -2.23709633]
 [-0.05947239  8.06260956]
 [ 2.1911753   1.42736587]
 [-1.52747726  2.92117212]
 [ 5.68383116  0.22802058]
 [-7.70627886 -7.63050415]]

Результат
[4 3 1 1 3 1 4 3 7 1 4 3 4 4 3 3 6 4 7 2 0 6 4 3 2 7 5 0 3 7 3 0 6 6 7 5 4
 4 3 6 2 1 4 5 4 4 2 3 3 6 7 3 0 0 5 4 1 5 2 0 0 5 4 2 3 6 0 2 3 3 2 2 4 2
 5 3 4 4 6 2 0 2 4 5 6 0 1 3 1 3 5 4 5 4 4 6 3 3 3 0 6 2 0 2 7 3 5 5 5 2 3
 5 6 4 3 2 0 3 6 0 5 3 3 6 4 3 0 5 6 6 3 3 3 4 6 2 6 0 5 5 3 0 2 1 6 2 5 0
 5 3 5 6 5 4 2 4 6 0 2 3 1 6 6 1 3 5 0 0 3 4 7 3 2 1 5 3 3 2 5 5 0 6 2 2 3
 3 4 7 3 7 7 0 2 6 7 5 3 3 5 6 3 3 5 7 4 1 3 3 2 3 6 1 3 3 6 0 3 3 6 6 5 0
 3 3 2 0 6 3 6 5 5 3 2 0 4 4 5 5 0 0 3 7 0 7 0 3 2 1 3 1]
```

Визуализация прогнозов

In [6]:

```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 2)
plt.scatter(X1[:, 0], X1[:, 1], c=sk_kmeans_pred_res, cmap="rainbow")
plt.scatter(sk_kmeans_centroinds[:, 0], sk_kmeans_centroinds[:, 1], marker="x", col
plt.title("KMeans (scikit-learn)")

plt.show()
```



Демонстрация кластеризации методом K-средних на основе данных рукописных цифр

Загрузка

Загрузка набора данных `digits` . Этот набор данных содержит рукописные цифры от 0 до 9.

In [7]:
```
from sklearn.datasets import load_digits

data, labels = load_digits(return_X_y=True)
(n_samples, n_features), n_digits = data.shape, np.unique(labels).size

print(f"# Цифр: {n_digits}; # Строк данных: {n_samples}; # Записей на каждую цифру
```

```
# Цифр: 10; # Строк данных: 1797; # Записей на каждую цифру 64
```

Визуализируйте результаты на данных, уменьшенных с помощью PCA

:class: `~sklearn.decomposition.PCA` позволяет проецировать данные из исходного

64-мерного пространства в пространство с меньшей размерностью. Впоследствии мы можем использовать :class: `~sklearn.decomposition.PCA` проецировать данные в двумерное пространство и отображать данные и кластеры в этом новом пространстве.

In [8]:
```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

reduced_data = PCA(n_components=2).fit_transform(data)
kmeans = KMeans(init="k-means++", n_clusters=n_digits, n_init=4)
kmeans.fit(reduced_data)

# Размер шага сетки. Можно уменьшить, чтобы повысить качество VQ.
h = 0.02  # точка в сетке [x_min, x_max]x[y_min, y_max].

# Построим границу принятия решения. Для этого мы назначим цвет каждому
x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Получить метки для каждой точки в сетке. Использовать последнюю обученную модель.
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])


# Помещаем результат в цветовую диаграмму.
Z = Z.reshape(xx.shape)
plt.figure(1)
plt.clf()
plt.imshow(
    Z,
    interpolation="nearest",
    extent=(xx.min(), xx.max(), yy.min(), yy.max()),
    cmap=plt.cm.Paired,
    aspect="auto",
    origin="lower",
)

plt.plot(reduced_data[:, 0], reduced_data[:, 1], "k.", markersize=2)
# Plot the centroids as a white X
centroids = kmeans.cluster_centers_
plt.scatter(
    centroids[:, 0],
    centroids[:, 1],
    marker="x",
    s=169,
    linewidths=3,
    color="w",
    zorder=10,
)
plt.title(
    "Кластеризация k-средних на наборе данных цифр\n(данные, уменьшенные PCA)\n"
    "Центроиды отмечены белым крестом"
)
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
```

```
plt.yticks(())
plt.show()
```

Сравнение алгоритмов кластеризации K-Means и MiniBatchKMeans

Сравнение производительности MiniBatchKMeans и KMeans: MiniBatchKMeans быстрее, но дает немного разные результаты

Кластеризация сначала набор данных с помощью KMeans, а затем с помощью MiniBatchKMeans, и отобразим результаты на графике. Мы также отобразим точки, которые помечены по-разному в двух алгоритмах.

Генерация данных

Начинаем с генерации блоков данных, которые необходимо кластеризовать.

In [9]:
```
import numpy as np

from sklearn.datasets import make_blobs

np.random.seed(0)
```

```
batch_size = 45
centers = [[1, 1], [-1, -1], [1, -1]]
n_clusters = len(centers)
X, labels_true = make_blobs(n_samples=3000, centers=centers, cluster_std=0.7)
```

Кластеризация с помощью KMeans

In [10]:
```
import time

from sklearn.cluster import KMeans

k_means = KMeans(init="k-means++", n_clusters=3, n_init=10)
t0 = time.time()
k_means.fit(X)
t_batch = time.time() - t0
```

Кластеризация с помощью MiniBatchKMeans

In [11]:
```
from sklearn.cluster import MiniBatchKMeans

mbk = MiniBatchKMeans(
    init="k-means++",
    n_clusters=3,
    batch_size=batch_size,
    n_init=10,
    max_no_improvement=10,
    verbose=0,
)
t0 = time.time()
mbk.fit(X)
t_mini_batch = time.time() - t0
```

Установление паритета между кластерами

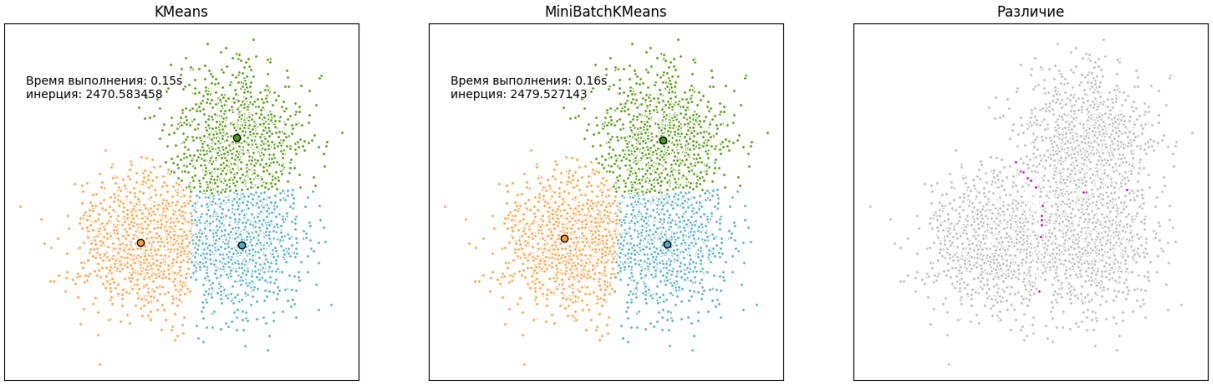
Нужен одинаковый цвет для одного и того же кластера из алгоритма MiniBatchKMeans и KMeans. Соединим центры кластеров по ближайшему.

In [12]:
```
from sklearn.metrics.pairwise import pairwise_distances_argmin

k_means_cluster_centers = k_means.cluster_centers_
order = pairwise_distances_argmin(k_means.cluster_centers_, mbk.cluster_centers_)
mbk_means_cluster_centers = mbk.cluster_centers_[order]

k_means_labels = pairwise_distances_argmin(X, k_means_cluster_centers)
mbk_means_labels = pairwise_distances_argmin(X, mbk_means_cluster_centers)
```

Plotting the results

In [13]:
```
fig = plt.figure(figsize=(15, 5))
fig.subplots_adjust(left=0.02, right=0.98, bottom=0.05, top=0.9)
colors = ["#4EACC5", "#FF9C34", "#4E9A06"]

# KMeans
ax = fig.add_subplot(1, 3, 1)
for k, col in zip(range(n_clusters), colors):
    my_members = k_means_labels == k
    cluster_center = k_means_cluster_centers[k]
    ax.plot(X[my_members, 0], X[my_members, 1], "w", markerfacecolor=col, marker=".
    ax.plot(
        cluster_center[0],
        cluster_center[1],
        "o",
        markerfacecolor=col,
        markeredgecolor="k",
        markersize=6,
    )
ax.set_title("KMeans")
ax.set_xticks(())
ax.set_yticks(())
plt.text(-3.5, 1.8, "Время выполнения: %.2fs\пинерция: %f" % (t_batch, k_means.iner

# MiniBatchKMeans
ax = fig.add_subplot(1, 3, 2)
for k, col in zip(range(n_clusters), colors):
    my_members = mbk_means_labels == k
    cluster_center = mbk_means_cluster_centers[k]
    ax.plot(X[my_members, 0], X[my_members, 1], "w", markerfacecolor=col, marker=".
    ax.plot(
        cluster_center[0],
        cluster_center[1],
        "o",
        markerfacecolor=col,
        markeredgecolor="k",
        markersize=6,
    )
ax.set_title("MiniBatchKMeans")
ax.set_xticks(())
ax.set_yticks(())
plt.text(-3.5, 1.8, "Время выполнения: %.2fs\пинерция: %f" % (t_mini_batch, mbk.ine

# Initialize the different array to all False
different = mbk_means_labels == 4
ax = fig.add_subplot(1, 3, 3)

for k in range(n_clusters):
    different += (k_means_labels == k) != (mbk_means_labels == k)

identical = np.logical_not(different)
ax.plot(X[identical, 0], X[identical, 1], "w", markerfacecolor="#bbbbbb", marker=".
ax.plot(X[different, 0], X[different, 1], "w", markerfacecolor="m", marker=".")
ax.set_title("Различие")
ax.set_xticks(())
ax.set_yticks(())
```

```
plt.show()
```



Ирисы

Variable Name	Role	Type	Description	Units	Missing Values
sepal length	Feature	Continuous		cm	no
sepal width	Feature	Continuous		cm	no
petal length	Feature	Continuous		cm	no
petal width	Feature	Continuous		cm	no
class	Target	Categorical	class of iris plant: Iris Setosa, Iris Versicolour, or Iris Virginica		no

sepal - чашелистик

petal - лепесток

In [15]:
```
import pandas as pd
iris_data = pd.read_csv('iris.data',header=None)
iris_data
```

Out[15]:

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [16]:
```
iris_no_labeled_data = iris_data.drop(4,axis=1)
iris_no_labeled_data
```

Out[16]:

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2
...	...	...	...	...
145	6.7	3.0	5.2	2.3
146	6.3	2.5	5.0	1.9
147	6.5	3.0	5.2	2.0
148	6.2	3.4	5.4	2.3
149	5.9	3.0	5.1	1.8

150 rows × 4 columns

In [17]:
```
iris_k_means = KMeans(init='k-means++', n_clusters=3, n_init=15)
iris_k_means.fit(iris_no_labeled_data)
```

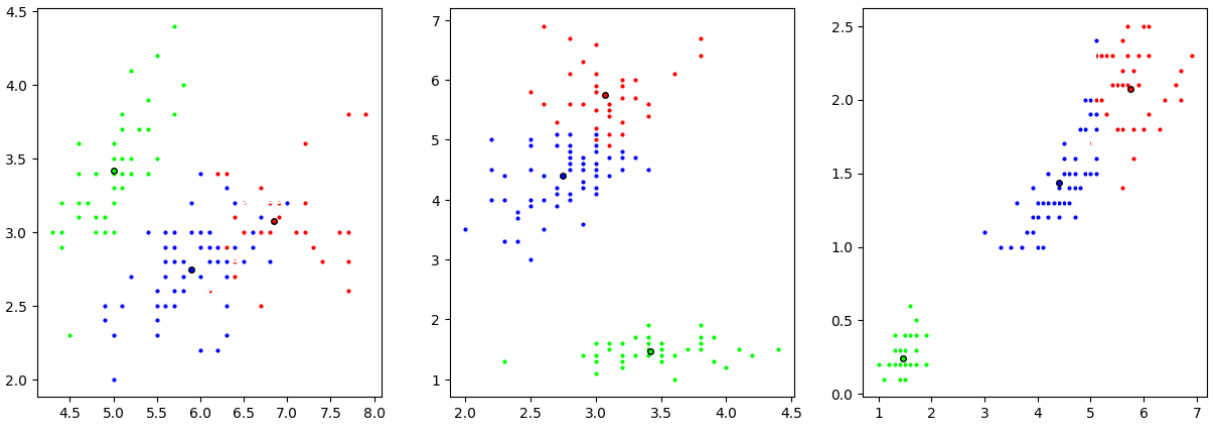
```
# X имеет 4 признака, но KMeans ожидает на входе 2 признака.
# 4-х мерное пространтсво
#iris_kmeans_predict = sk_kmeans.predict(iris_no_labeled_data)
iris_k_mean_cc = iris_k_means.cluster_centers_
iris_k_mean_cc
```

Out[17]:
```
array([[6.85      , 3.07368421, 5.74210526, 2.07105263],
       [5.006     , 3.418     , 1.464     , 0.244     ],
       [5.9016129 , 2.7483871 , 4.39354839, 1.43387097]])
```

In [18]:
```
from sklearn.metrics.pairwise import pairwise_distances_argmin
iris_labels = pairwise_distances_argmin(iris_no_labeled_data,iris_k_means.cluster_c
iris_labels
```

Out[18]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 0, 2, 0, 0, 0, 0, 2, 0,
       0, 0, 0, 2, 2, 0, 0, 0, 0, 2, 0, 2, 0, 2, 0, 0, 2, 2, 0, 0,
       0, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 0, 2, 0, 0, 2])
```

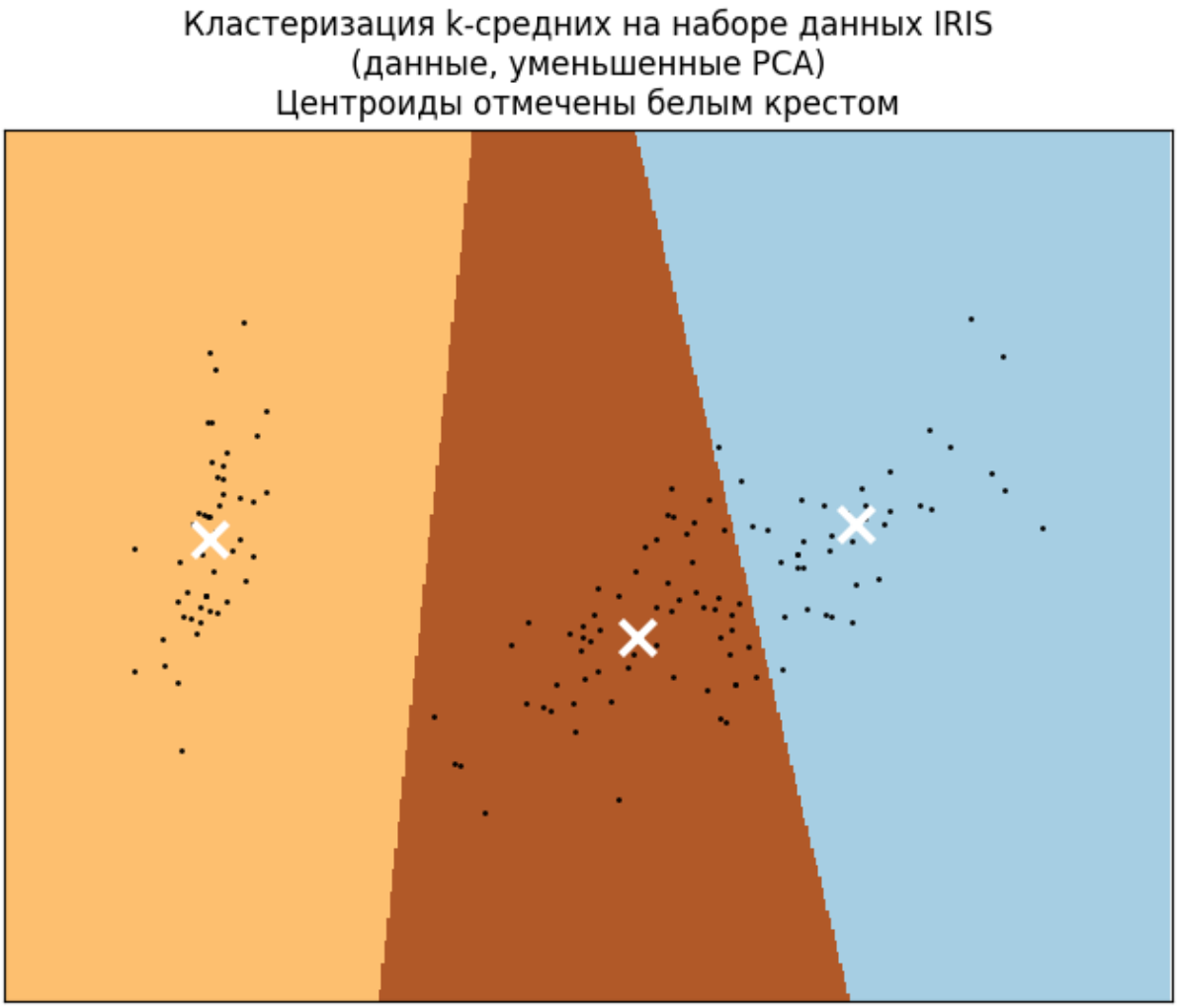
In [19]:
```
plt.rcParams['figure.figsize'] = (15,5)
f,ax=plt.subplots(1,3)
colors = ['#FF0000','#00FF00','#0000FF']

for i in range(3):
    nn = iris_labels == i
    cc=iris_k_mean_cc[i]
    for j in range(3):
        ax[j].plot(
            iris_no_labeled_data[nn][j],
            iris_no_labeled_data[nn][j+1],
            'w',
            markerfacecolor = colors[i],
            marker = 'o',
            markersize=4
        )
        ax[j].plot(
            cc[j],
            cc[j+1],
            'o',
            markerfacecolor = colors[i],
            markeredgecolor = 'k',
            markersize=4
        )
plt.show()
```



In [20]:

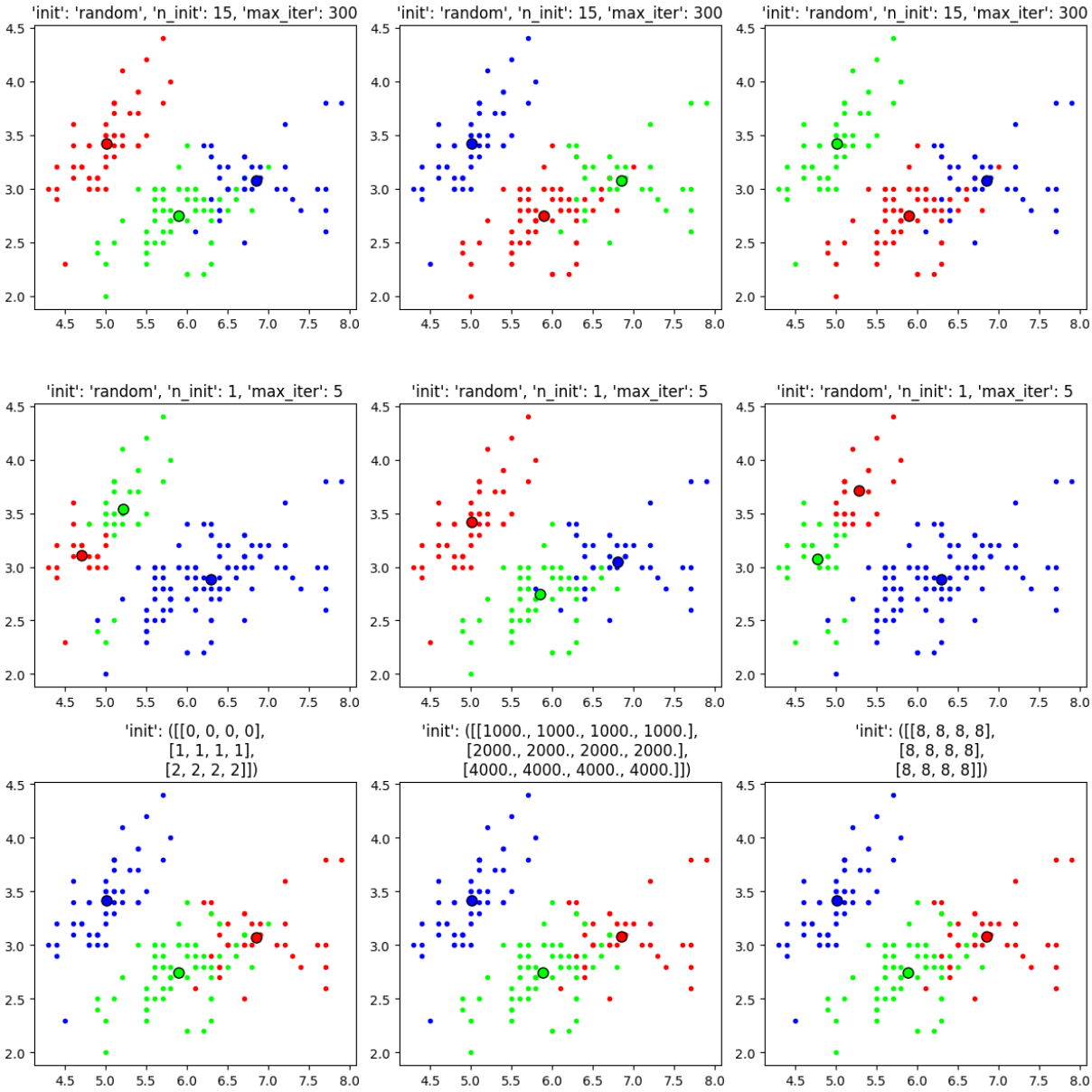
```
reduced_data = PCA(n_components=2).fit_transform(iris_no_labeled_data)
kmeans = KMeans(init="k-means++", n_clusters=3, n_init=4)
kmeans.fit(reduced_data)

# Размер шага сетки. Можно уменьшить, чтобы повысить качество VQ.
h = 0.02  # точка в сетке [x_min, x_max]x[y_min, y_max].

# Построим границу принятия решения. Для этого мы назначим цвет каждому
x_min, x_max = reduced_data[:, 0].min() - 1, reduced_data[:, 0].max() + 1
y_min, y_max = reduced_data[:, 1].min() - 1, reduced_data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

# Получить метки для каждой точки в сетке. Использовать последнюю обученную модель.
Z = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])


# Помещаем результат в цветовую диаграмму.
Z = Z.reshape(xx.shape)
plt.figure(1,figsize=(8,6))
plt.clf()
plt.imshow(
    Z,
    interpolation="nearest",
    extent=(xx.min(), xx.max(), yy.min(), yy.max()),
    cmap=plt.cm.Paired,
    aspect="auto",
    origin="lower",
)

plt.plot(reduced_data[:, 0], reduced_data[:, 1], "k.", markersize=2)
# Plot the centroids as a white X
centroids = kmeans.cluster_centers_
plt.scatter(
    centroids[:, 0],
    centroids[:, 1],
    marker="x",
    s=169,
    linewidths=3,
    color="w",
    zorder=10,
)
plt.title(
```

```
    "Кластеризация k-средних на наборе данных IRIS\n(данные, уменьшенные PCA)\n"
    "Центроиды отмечены белым крестом"
)
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.xticks(())
plt.yticks(())
plt.show()
```

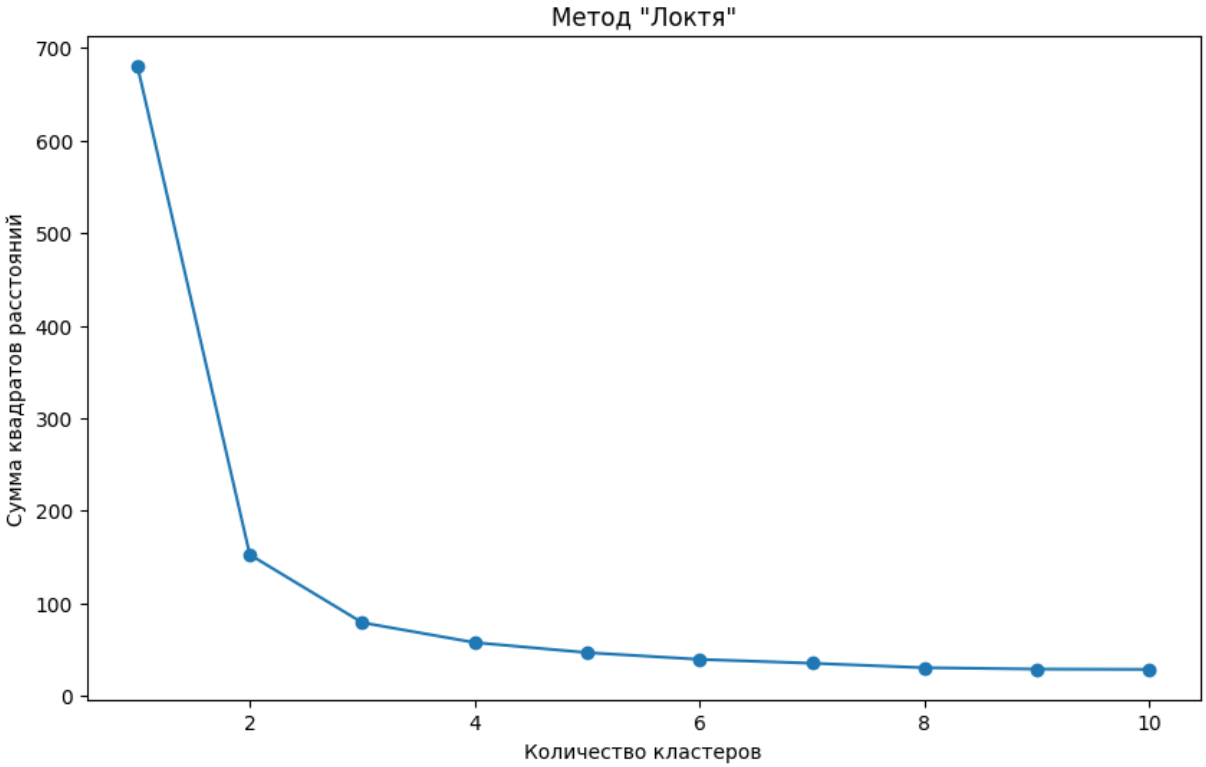


In [21]:
```
Params = np.array([
[{'init':'random','n_init':15,'max_iter':300}]*3,
[{'init':'random','n_init':1,'max_iter':5}]*3,
[
    {'init':np.array([[0,0,0,0],[1,1,1,1],[2,2,2,2]])},
    {'init':np.array([[1e3,1e3,1e3,1e3],[2e3,2e3,2e3,2e3],[4e3,4e3,4e3,4e3]])},
    {'init':np.array([[8,8,8,8],[8,8,8,8],[8,8,8,8]])}
]
])

fig, axes = plt.subplots(*Params.shape, figsize=(12, 12))

index = 0

for x in range(Params.shape[0]):
    for y in range(Params.shape[1]):
```

```
        km = KMeans(n_clusters=3, **Params[x][y])
        km.fit(iris_no_labeled_data)
        centers = km.cluster_centers_
        labels = pairwise_distances_argmin(iris_no_labeled_data, centers)
        axes[x][y].set_title(str(Params[x][y]).replace('{', '').replace('}', '').re
        for i in range(3):
            nn = labels == i
            center = centers[i]
            axes[x][y].plot(
                iris_no_labeled_data.loc[nn][index],
                iris_no_labeled_data[nn][index + 1],
                markerfacecolor=colors[i],
                markeredgecolor=colors[i],
                marker='o',
                markersize=3,
                lw=0
            )

            axes[x][y].plot(
                center[index],
                center[index + 1],
                'o',
                markerfacecolor=colors[i],
                markeredgecolor='k',
                markersize=8
            )
fig.tight_layout()
```



In [22]:
```
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(iris_no_labeled_data)
    sse.append(kmeans.inertia_)

plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), sse, marker='o')
plt.title('Метод "Локтя"')
plt.xlabel('Количество кластеров')
plt.ylabel('Сумма квадратов расстояний')
plt.show()
```



MiniBatchKMeans

The MiniBatchKMeans представляет собой вариант KMeans алгоритм который использует мини-пакеты для сокращения времени вычислений, но при этом пытается оптимизировать одну и ту же целевую функцию. Мини-пакеты — это подмножества входных данных. данные, случайно выбранные на каждой итерации обучения. Эти мини-партии радикально сократить объем вычислений, необходимых для сходимости к локальному решение. В отличие от других алгоритмов, сокращающих время сходимости k-средние, мини-пакетное использование k-средних дает результаты, которые, как правило, лишь незначительно хуже стандартного алгоритма.

Алгоритм выполняет итерации между двумя основными шагами, подобно ванильным k-средним. На первом этапе

выборки выбираются случайным образом из набора данных, чтобы сформировать мини-партия. Затем они присваиваются ближайшему центроиду. Во втором шаг, центроиды обновляются. В отличие от k-средних, это делается на на выборку. Для каждого образца в мини-партии назначенный центроид обновляется путем взятия потокового среднего значения выборки и всех предыдущих образцы, назначенные этому центроиду. Это приводит к уменьшению скорость изменения центроида с течением времени. Эти действия выполняются до тех пор, пока достигается сходимость или заданное количество итераций.

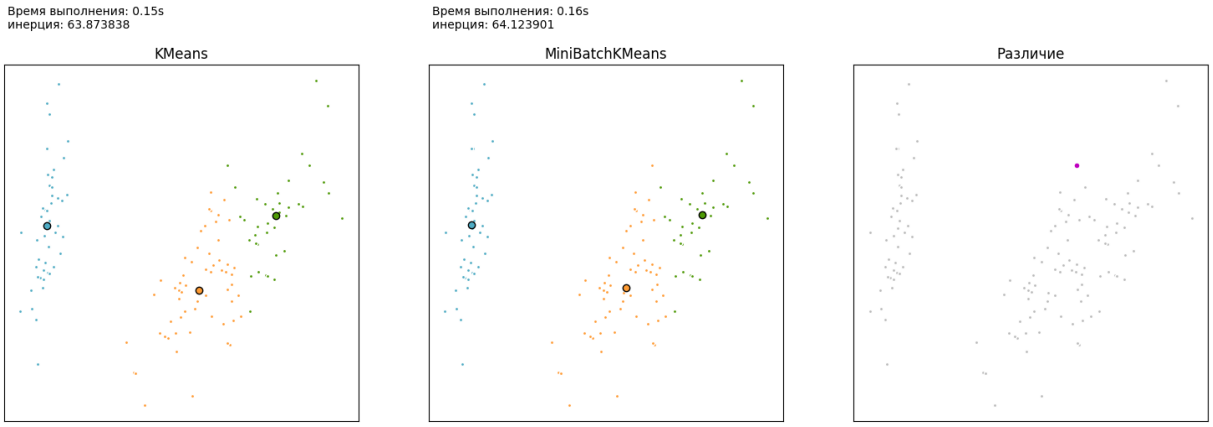
MiniBatchKMeans сходится быстрее, чем KMeans, но качество результатов снижается. На

практике эта разница в качестве может быть весьма существенной. небольшой, как показано в примере и цитируемой ссылке.

In [23]:
```
X = PCA(n_components=2).fit_transform(iris_no_labeled_data)
k_means = KMeans(init="k-means++", n_clusters=3, n_init=10)
k_means.fit(X)
k_means_cluster_centers = k_means.cluster_centers_
k_means_labels = pairwise_distances_argmin(X, k_means_cluster_centers)

mbk = MiniBatchKMeans(
    init="k-means++",
    n_clusters=3,
    batch_size=batch_size,
    n_init=10,
    max_no_improvement=10,
    verbose=0,
)
mbk.fit(X)
order = pairwise_distances_argmin(k_means.cluster_centers_, mbk.cluster_centers_)
mbk_means_cluster_centers = mbk.cluster_centers_[order]
mbk_means_labels = pairwise_distances_argmin(X, mbk_means_cluster_centers)

colors = ["#4EACC5", "#FF9C34", "#4E9A06"]
```

In [24]:
```
fig = plt.figure(figsize=(15, 5))
fig.subplots_adjust(left=0.02, right=0.98, bottom=0.05, top=0.9)

# KMeans
ax = fig.add_subplot(1, 3, 1)
for k, col in zip(range(n_clusters), colors):
    my_members = k_means_labels == k
    cluster_center = k_means_cluster_centers[k]
    ax.plot(X[my_members, 0], X[my_members, 1], "w", markerfacecolor=col, marker=".
    ax.plot(
        cluster_center[0],
        cluster_center[1],
        "o",
        markerfacecolor=col,
        markeredgecolor="k",
        markersize=6,
    )
ax.set_title("KMeans")
ax.set_xticks(())
ax.set_yticks(())
plt.text(-3.5, 1.8, "Время выполнения: %.2fs\пинерция: %f" % (t_batch, k_means.iner

# MiniBatchKMeans
ax = fig.add_subplot(1, 3, 2)
for k, col in zip(range(n_clusters), colors):
    my_members = mbk_means_labels == k
    cluster_center = mbk_means_cluster_centers[k]
    ax.plot(X[my_members, 0], X[my_members, 1], "w", markerfacecolor=col, marker=".
    ax.plot(
        cluster_center[0],
        cluster_center[1],
```

```
        "o",
        markerfacecolor=col,
        markeredgecolor="k",
        markersize=6,
    )
ax.set_title("MiniBatchKMeans")
ax.set_xticks(())
ax.set_yticks(())
plt.text(-3.5, 1.8, "Время выполнения: %.2fs\пинерция: %f" % (t_mini_batch, mbk.ine

# Initialize the different array to all False
different = mbk_means_labels == 4
ax = fig.add_subplot(1, 3, 3)

for k in range(n_clusters):
    different += (k_means_labels == k) != (mbk_means_labels == k)

identical = np.logical_not(different)
ax.plot(X[identical, 0], X[identical, 1], "w", markerfacecolor="#bbbbbb", marker=".
ax.plot(X[different, 0], X[different, 1], "w", markerfacecolor="m", marker="o", mar
ax.set_title("Различие")
ax.set_xticks(())
ax.set_yticks(())

plt.show()
```



Реализация на Python с нуля

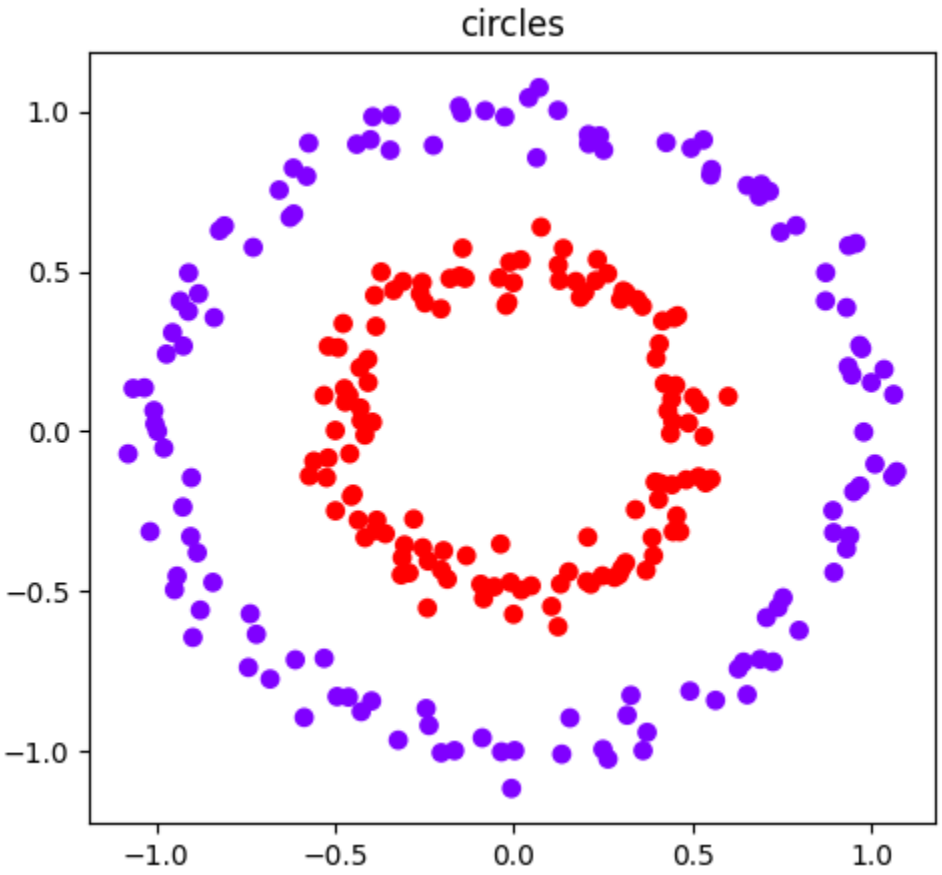
In [25]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_circles
from sklearn.metrics import adjusted_rand_score
from sklearn.preprocessing import StandardScaler
```

Загрузка датасета

In [26]:
```
X4, y4 = make_circles(n_samples=250, noise=0.05, factor=0.5, random_state=0)
```

```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 2)
plt.scatter(X4[:, 0], X4[:, 1], c=y4, cmap='rainbow')
plt.title('circles')

X4 = StandardScaler().fit_transform(X4)
print(y4)
```

```
[1 0 0 1 1 1 1 0 0 1 1 1 0 1 0 0 1 1 1 0 1 0 1 0 1 0 0 0 1 0 1 0 0 1 1 1 1
 0 0 1 0 0 0 1 0 0 1 0 0 1 0 1 1 0 0 0 1 0 1 0 1 0 0 1 1 1 0 1 1 1 1 1 1 0
 1 0 1 0 1 0 0 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 1 1 0 1 1 0 1 0 0 0 1 0 1 0 0
 1 1 1 0 0 0 0 1 0 1 0 1 1 0 1 1 0 1 1 1 1 0 1 1 1 0 1 0 1 0 1 1 0 0 0 0 0
 0 0 0 1 1 0 1 1 0 0 1 1 1 0 0 0 0 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0 0 1 0 0 1
 1 0 0 1 0 1 1 1 0 0 1 0 1 1 0 1 1 1 1 1 0 1 1 1 1 0 1 0 0 1 1 0 1 1 0 1 1
 0 0 0 1 0 1 1 1 0 1 1 0 1 0 1 0 0 0 0 1 0 0 1 0 1 0 0 1]
```



DBSCAN (scikit-learn)

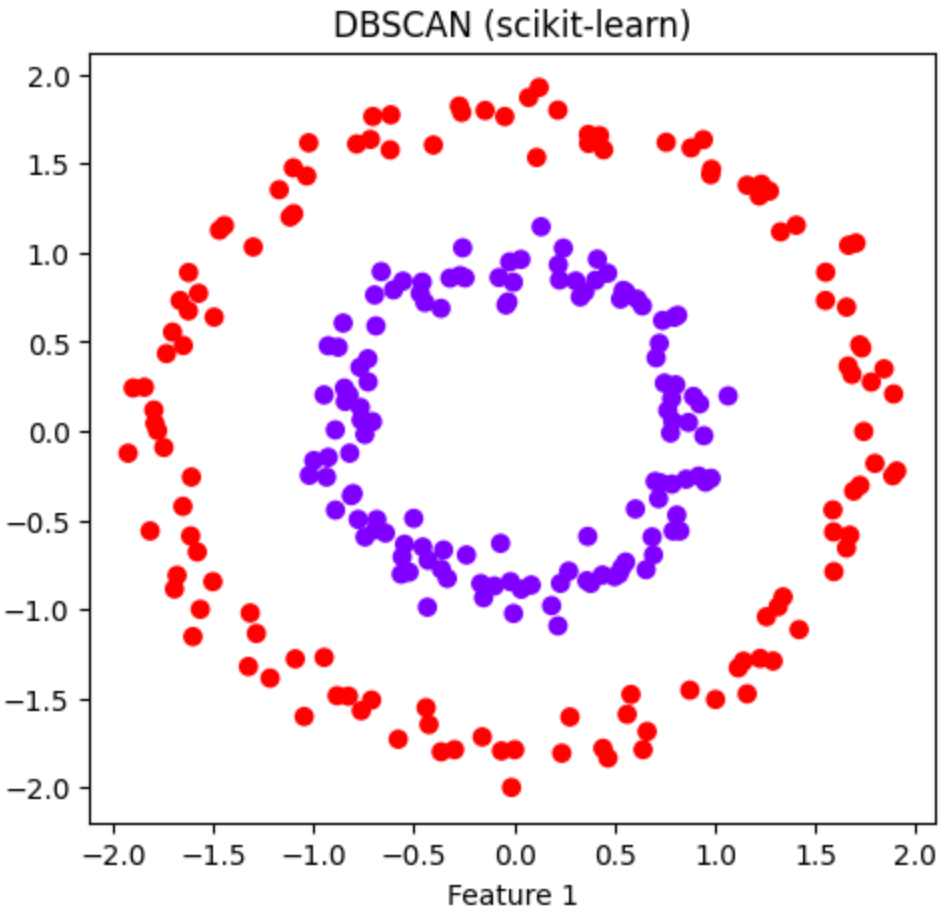
In [27]:
```
sk_dbscan = DBSCAN(
    eps=0.3, # Максимальное расстояние между двумя образцами, при котором один из н
             # Это не максимальная граница расстояний между точками внутри кластера
             # который следует выбрать в соответствии с вашим набором данных и функ
    min_samples=3 # Количество выборок (или общий вес) в окрестности точки, которая
                  # Это включает в себя саму точку. Если для min_samples установлен
                  # DBSCAN будет искать более плотные кластеры, тогда как если для
                  # низкое значение, найденные кластеры будут более разреженными.
    )
# Вычисляем кластеры на основе матрицы данных (или расстояний) и прогнозируем метки
```

```
sk_dbscan_pred_res = sk_dbscan.fit_predict(X4)
# Индекс Рэнда с поправкой на случайность.
# Индекс Рэнда вычисляет меру сходства между двумя кластеризациями,
# рассматривая все пары выборок и подсчитывая пары, которые относятся к
# одному и тому же или разным кластерам в предсказанной и истинной кластеризации.
sk_dbscan_ari = adjusted_rand_score(y4, sk_dbscan_pred_res)

print(f'Adjusted Rand Score for sk DBSCAN: {sk_dbscan_ari}', '', sep='\n')
print('prediction', sk_dbscan_pred_res, sep='\n')
```

```
Adjusted Rand Score for sk DBSCAN: 1.0

prediction
[0 1 1 0 0 0 0 1 1 0 0 0 1 0 1 1 0 0 0 1 0 1 0 1 0 1 1 1 0 1 0 1 1 0 0 0 0
 1 1 0 1 1 1 0 1 1 0 1 1 0 1 0 0 1 1 1 0 1 0 1 0 1 1 0 0 0 1 0 0 0 0 0 0 1
 0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 1 0 1 1 0 0 1 0 0 1 0 1 1 1 0 1 0 1 1
 0 0 0 1 1 1 1 0 1 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 0 0 1 1 1 1 1
 1 1 1 0 0 1 0 0 1 1 0 0 0 1 1 1 1 0 0 0 0 0 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0
 0 1 1 0 1 0 0 0 1 1 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 1 0 1 1 0 0 1 0 0 1 0 0
 1 1 1 0 1 0 0 0 1 0 0 1 0 1 0 1 1 1 1 0 1 1 0 1 0 1 1 0]
```

In [28]:

```
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 2)
plt.scatter(X4[:, 0], X4[:, 1], c=sk_dbscan_pred_res, cmap='rainbow')
plt.title('DBSCAN (scikit-learn)')
plt.xlabel("Feature 1")

plt.show()
```



Обучение моделей и оценка полученных результатов

Как можно заметить, DBSCAN также отлично справился с кластеризацией данных сложной формы, однако выбор оптимальных eps и min_samples на практике может оказаться довольно трудной задачей, поскольку данные параметры существенно влияют на результаты кластеризации.

Частично данную проблему можно решить с использованием HDBSCAN — модификации DBSCAN, которая автоматически находит подходящее значение eps для каждого кластера, используя иерархический подход, что позволяет определять кластеры разной плотности и повысить устойчивость к выбросам.

OPTICS

Ещё одной интересной модификацией, похожей на HDBSCAN, является OPTICS (Ordering Points To Identify the Clustering Structure), где используется граф достижимости, который определяет достижимое расстояние для каждой точки, которая в дальнейшем будет относиться к ближайшему кластеру. Такой подход позволяет ещё лучше определять кластеры разной плотности, особенно если они расположены близко друг к другу, однако это увеличивает время работы алгоритма.