Обучение с учителем и без учителя

Большинство моделей машинного обучения делятся на:

- модели обучения с учителем (контролируемого)
- без учителя (неконтролируемого)

Модели обучения с учителем ¶

Цель моделей обучения с учителем - делать предсказания. Вы обучаете их с помощью помеченных данных, чтобы они могли принимать последующие входные данные и предсказывать, какие метки будут получены.

Иными словами, чтобы алгоритм относился к обучению с учителем, он должен работать с примерами, которые содержат не только независимые переменные (атрибуты, признаки), но и значение, которое должна выдавать модель после обучения (такое значение называется целевым). Разность между целевым и фактическим выходами модели называется ошибкой обучения (остатками), которая минимизируется в процессе обучения и выступает в качестве «учителя». Значение выходной ошибки затем используется для вычисления коррекций параметров модели на каждой итерации обучения. Алгоритмами обучения с учителем для решения задачи регрессии являются: линейная регрессия, логистическая регрессия и т.д.

Модели обучения без учителя

Модели обучения без учителя, напротив, не требуют маркированных данных. Их цель - дать представление о существующих данных или сгруппировать данные по категориям и соответствующим образом классифицировать будущие данные.

В оучении без учителя для коррекции параметров обучаемой модели не используется целевая функция. Иными словами, в обучающих примерах при обучении без учителя не нужно иметь заранее заданные выходы модели. Классический пример — сегментация пользователей. Алгоритм k-средних (k-means). Буква k в названии отвечает за количество выделенных центроидов — точек, которые формируют вокруг себя кластер. Удобнее, если они совпадают с точками выборки данных, но необязательно.

К примеру, возьмем значение k, равное 5. Дальше для каждой точки датасета посчитаем расстояние до каждого из пяти кластеров. Наименьшее расстояние от точки до центроида позволит модели предположить, что объект принадлежит к этому кластеру.

Обучение без учителя с помощью кластеризации k средних

В обучении без учителя часто используется техника, называемая *кластеризацией*. Цель кластеризации - сгруппировать данные по сходству. Наиболее популярным алгоритмом кластеризации является кластеризация по методу k средних (k-means clustering), при которой берут n образцов данных и группируют их в m кластеров, где m - заданное число.

Группировка осуществляется с помощью итерационного процесса, в ходе которого для каждого кластера вычисляется центроид, и образцы распределяются по кластерам в зависимости от их близости к центроидам кластеров. Например, если расстояние от конкретного образца до центроида первого кластера равно 2,0, а расстояние от того же образца до центра второго кластера равно 3,0, то образец относится к первому кластеру.

Модель обучения без учителя с использованием **кластеризации k средних** построим с *Scikit-Learn*.

Начните с импорта Matplotlib и Seaborn

In [2]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

Для генерации случайного набора пар координат x и y используем функцию `make_blobs` (https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_blobs.html) и функцию scatter из Matplotlib для их отображения на графике

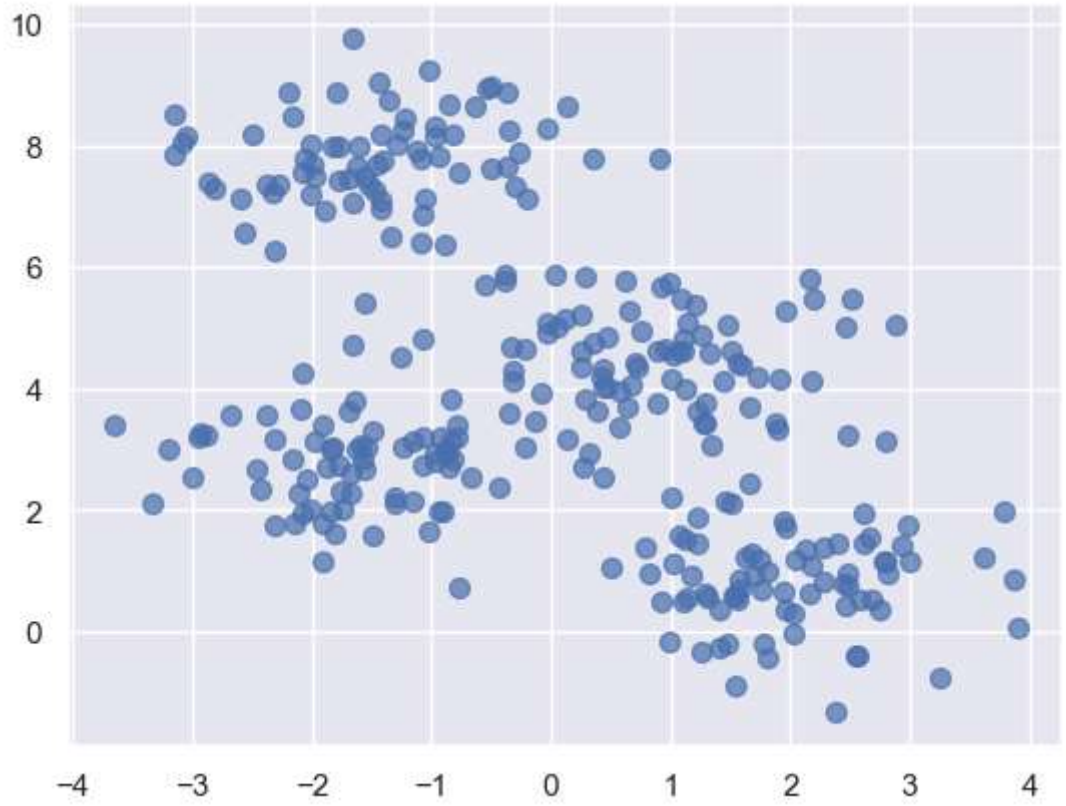
In [3]:
```
from sklearn.datasets import make_blobs

points, cluster_indexes = make_blobs(n_samples=300, centers=4, cluster_std=0.8, random_state=0)

x = points[:, 0]
y = points[:, 1]

plt.scatter(x, y, s=50, alpha=0.7)
```

Out[3]:
```
<matplotlib.collections.PathCollection at 0x12e9413f7b8>
```



Ваш результат должен быть похож этому благодаря параметру random_state, который инициализирует генератор случайных чисел, используемый make_blobs.

Далее с помощью *кластеризации k средних* разделите пары координат на три группы. Затем выделите центральные точки кластеров красным цветом и распределите точки данных по кластерам. После того, как к классу KMeans предоставляются пары координат, можно получить местоположение центроидов из атрибута cluster_centers_ класса KMeans.

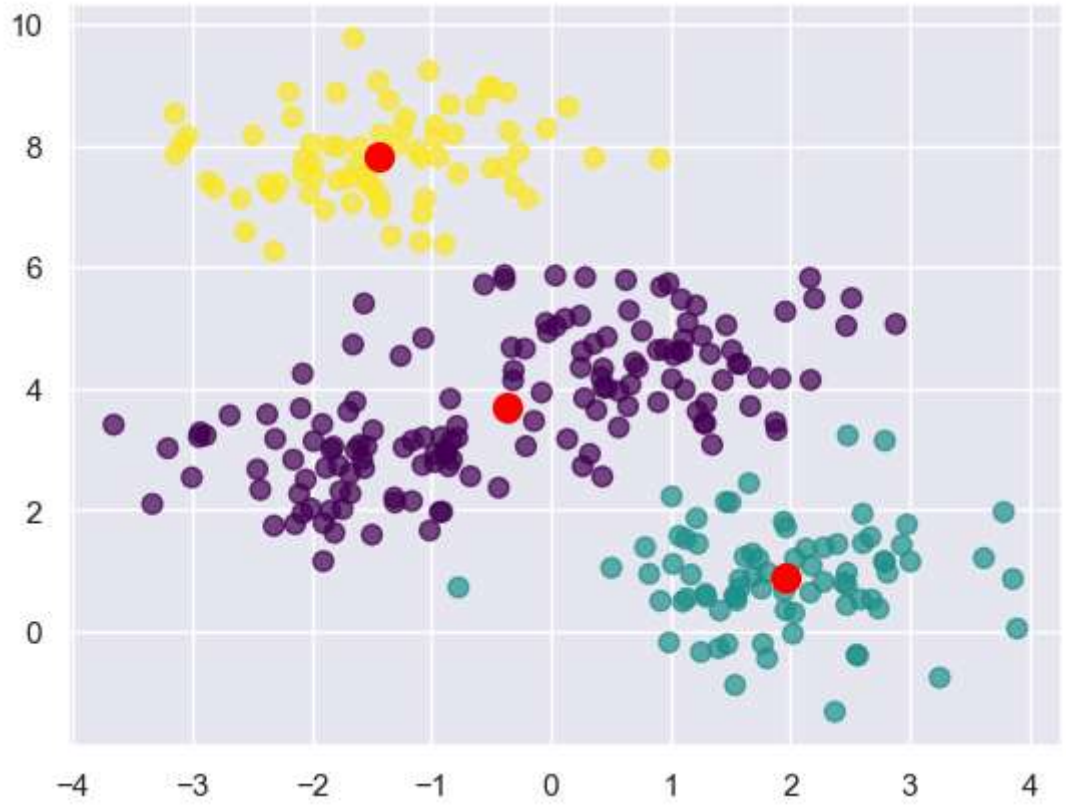
In [4]:
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=0)
kmeans.fit(points)
predicted_cluster_indexes = kmeans.predict(points)

plt.scatter(x, y, c=predicted_cluster_indexes, s=50, alpha=0.7, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)
```

Out[4]:
```
<matplotlib.collections.PathCollection at 0x12e9677a4e0>
```



А как это выглядело бы, если использовать четыре кластера?

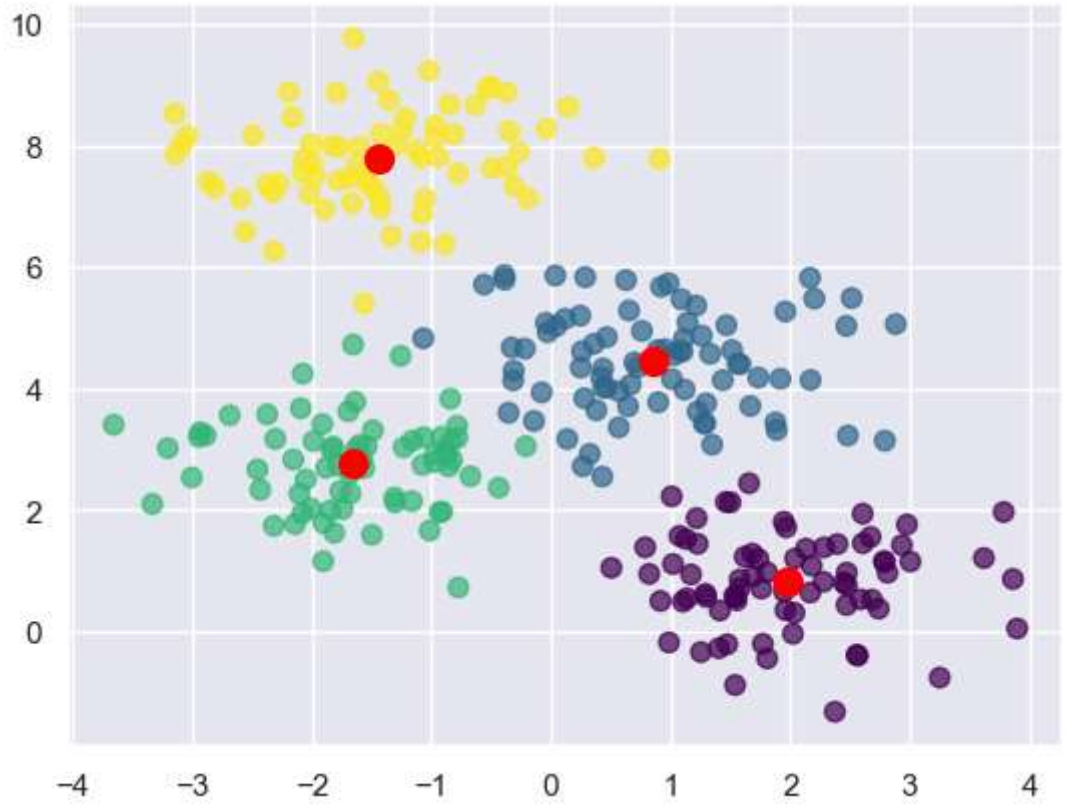
In [5]:
```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=4, random_state=0)
kmeans.fit(points)
predicted_cluster_indexes = kmeans.predict(points)

plt.scatter(x, y, c=predicted_cluster_indexes, s=50, alpha=0.7, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)
```

Out[5]:
```
<matplotlib.collections.PathCollection at 0x12e967c7550>
```



Попробуйте задать для n_clusters другие значения, например 5, чтобы посмотреть, как группируются точки при разном количестве кластеров. Возникает вопрос: *как определить правильное количество кластеров?*

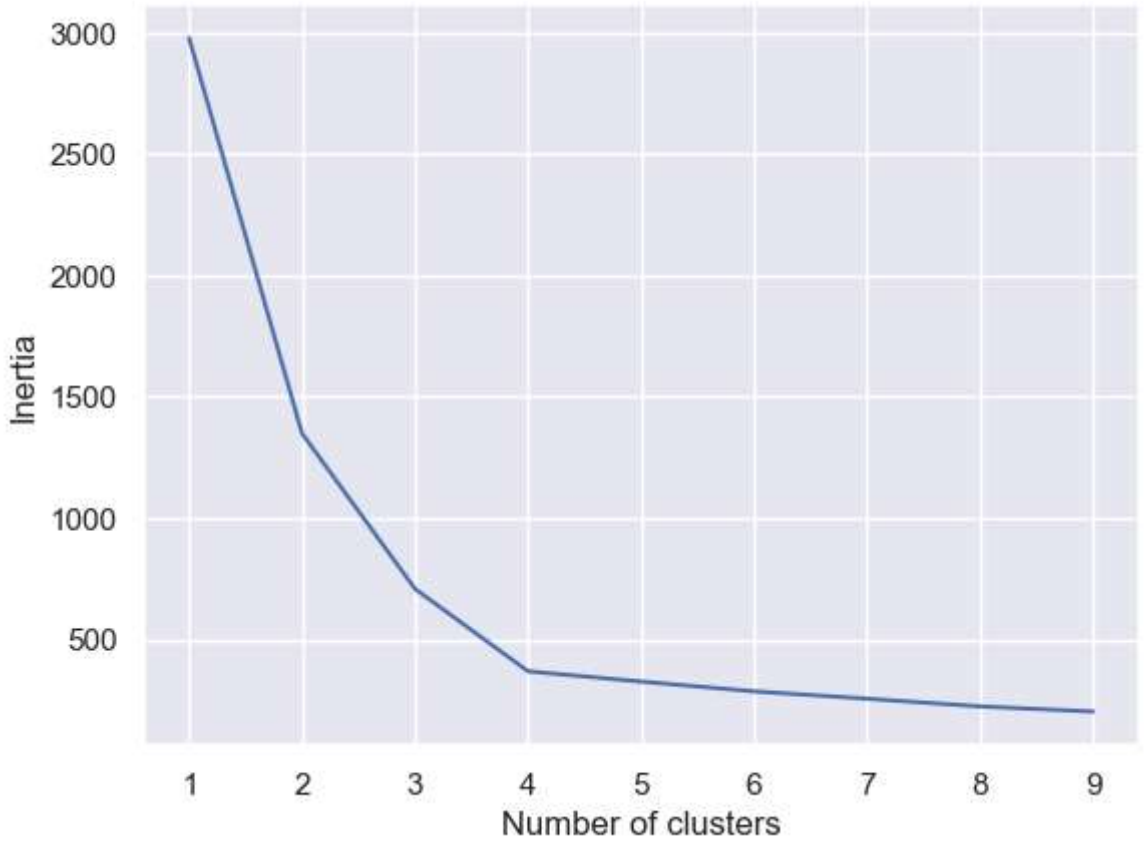
Одним из способов выбора нужного числа кластеров является, метод локтя, который строит графики «инерции» (сумму квадратов расстояний точек данных до ближайшего центра кластера), полученную из KMeans.inertia_

Постройте инерции и найдите наиболее острый локоть кривой:

In [8]:
```
inertias = []

for i in range(1, 10):
    kmeans = KMeans(n_clusters=i, random_state=0)
    kmeans.fit(points)
    inertias.append(kmeans.inertia_)
plt.plot(range(1, 10), inertias)
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

Out[8]:
```
Text(0, 0.5, 'Inertia')
```



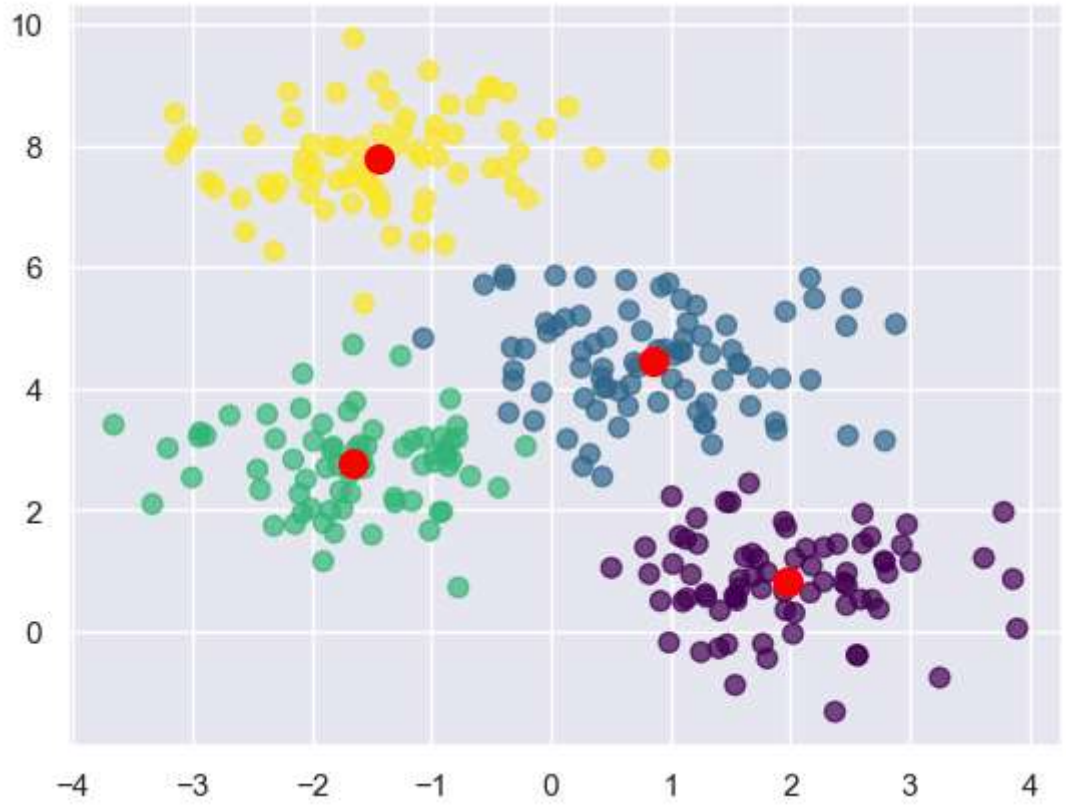
В данном примере, видимо, 4 кластера являются правильным выбором

In [10]:
```
kmeans = KMeans(n_clusters=4, random_state=0)
kmeans.fit(points)
predicted_cluster_indexes = kmeans.predict(points)

plt.scatter(x, y, c=predicted_cluster_indexes, s=50, alpha=0.7, cmap='viridis')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)
```

Out[10]:
```
<matplotlib.collections.PathCollection at 0x12e9990f6a0>
```



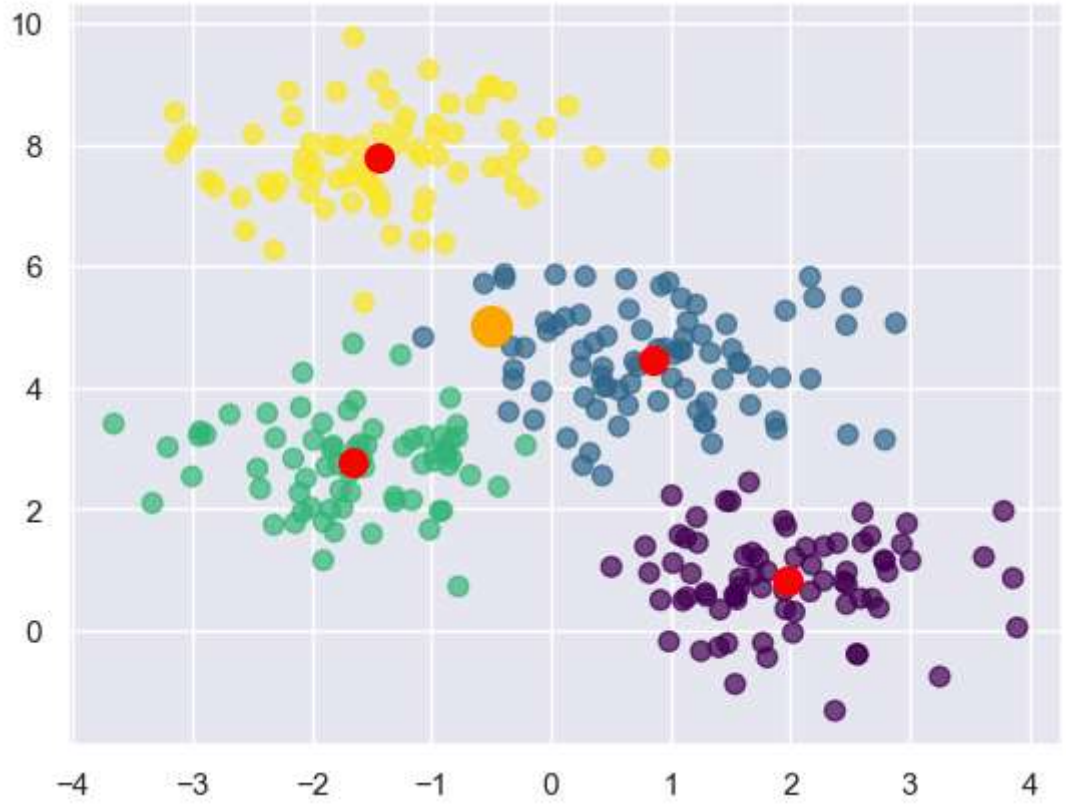
Теперь добавим к этому новую точку данных и прогнозируем к какому кластеру оно относится.

In [13]:
```
import numpy as np

point = np.array([[-0.5, 5]])
px = point[:, 0]
py = point[:, 1]

plt.scatter(x, y, c=predicted_cluster_indexes, s=50, alpha=0.7, cmap='viridis')
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)
plt.scatter(px, py, c='orange', s=200)
```

Out[13]:
```
<matplotlib.collections.PathCollection at 0x12e99aef550>
```



In [14]:
```
cluster = kmeans.predict(point)[0]
print(cluster)
```

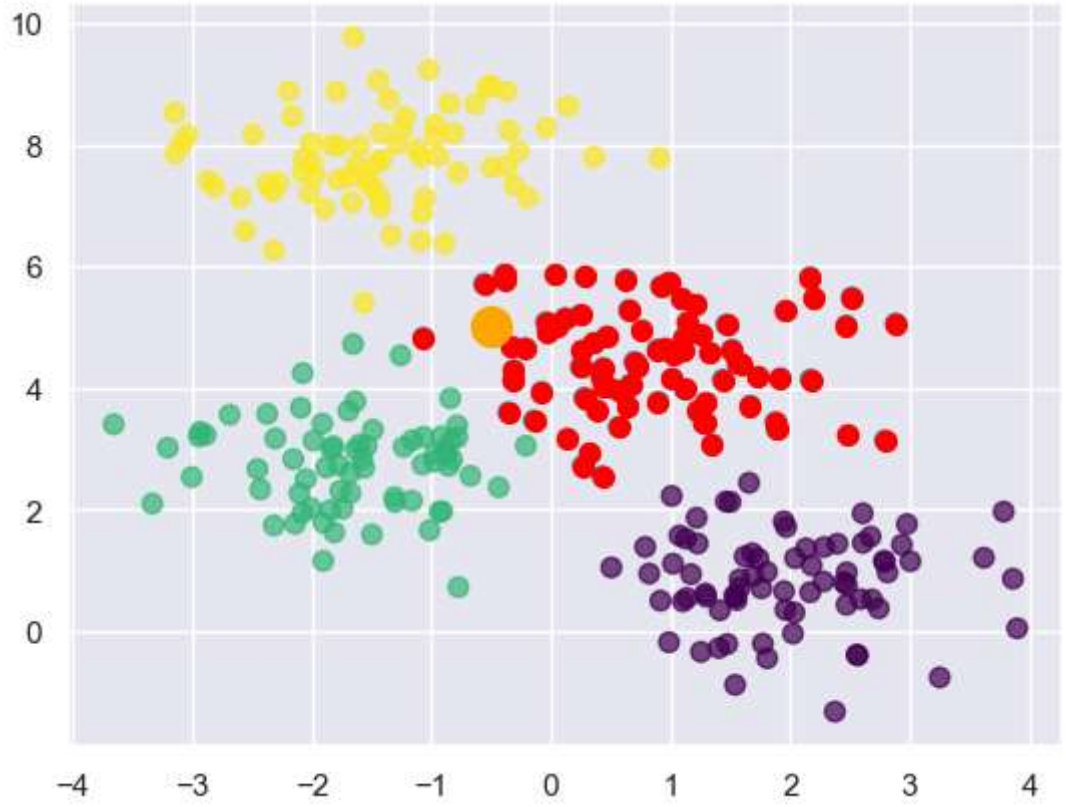
```
1
```

Какой это кластер? Чтобы ответить на этот вопрос, окрасьте все точки в соответствующем кластере в красный цвет.

In [15]:
```
r = points[predicted_cluster_indexes==cluster]

plt.scatter(x, y, c=predicted_cluster_indexes, s=50, alpha=0.7, cmap='viridis')
plt.scatter(r[:, 0], r[:, 1], c='red', s=50)
plt.scatter(px, py, c='orange', s=200)
```

Out[15]:
```
<matplotlib.collections.PathCollection at 0x12e99b34550>
```



Теперь определим расстояние от данной точки до каждого центра.

In [16]:
```
d = []
for i in range(len(centers)):
    d.append(np.sqrt((px - centers[i][0])**2 + (py - centers[i][1])**2))

print(d)
```

```
[array([4.8470332]), array([1.46570934]), array([2.52147763]), array([2.93542029])]
```

Вопрос: Имеет ли теперь смысл, что KMeans отнесла точку к кластеру, чей индекс равен 1?

Применение кластеризации k средних для сегментации клиентов по двум признакам

Давайте применим кластеризацию k средних для решения реальной задачи: сегментирования клиентов с целью выявления тех, на кого следует направить рекламную акцию для повышения их покупательской активности. В качестве набора данных будет использован файл с Kaggle customers.csv.

In [17]:
```
# загрузка набора данных в DataFrame и отображзить первые пять строк
import pandas as pd

customers = pd.read_csv('Data/customers.csv')
# customers.shape
# customers.info()
customers.head()
```

Out[17]:

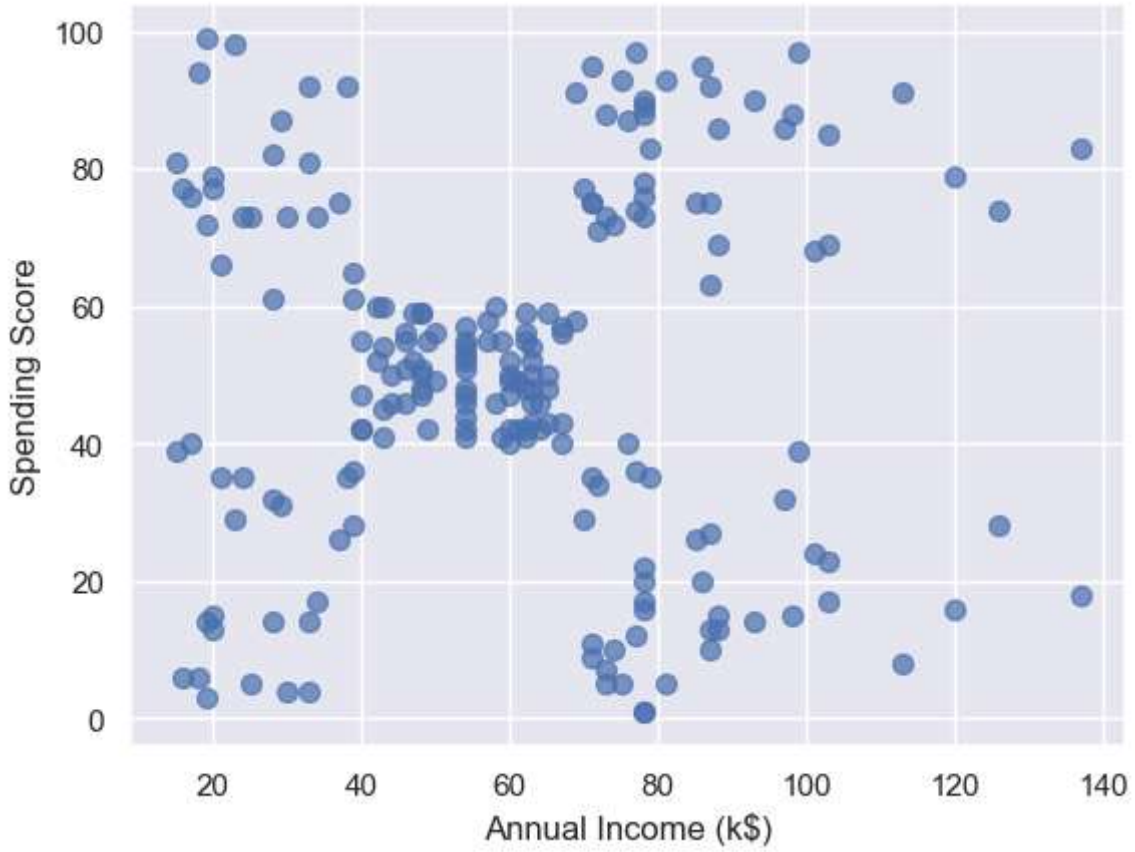
	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Как видно из результата набор данных содержит пять столбцов, два из которых описывают годовой доход клиента и его рейтинг расходов (баллы от 0-100). Чем больше число, тем существеннее клиент потратился.

Теперь постройте график годовых доходов и расходов:

In [24]:
```
# считываем столбцы 3 и 4
points = customers.iloc[:, 3:5].values
x = points[:, 0]
y = points[:, 1]
# Ось X представляет годовой доход клиента, а ось Y — рейтинг расходов
# s-размер маркера в пунктах**2 (типографский размер — 1/72 дюйма)
# alpha-смешивание от 0 (прозрачный) до 1 (непрозрачный).
plt.scatter(x, y, s=50, alpha=0.7)
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score')
```

Out[24]:
```
Text(0, 0.5, 'Spending Score')
```

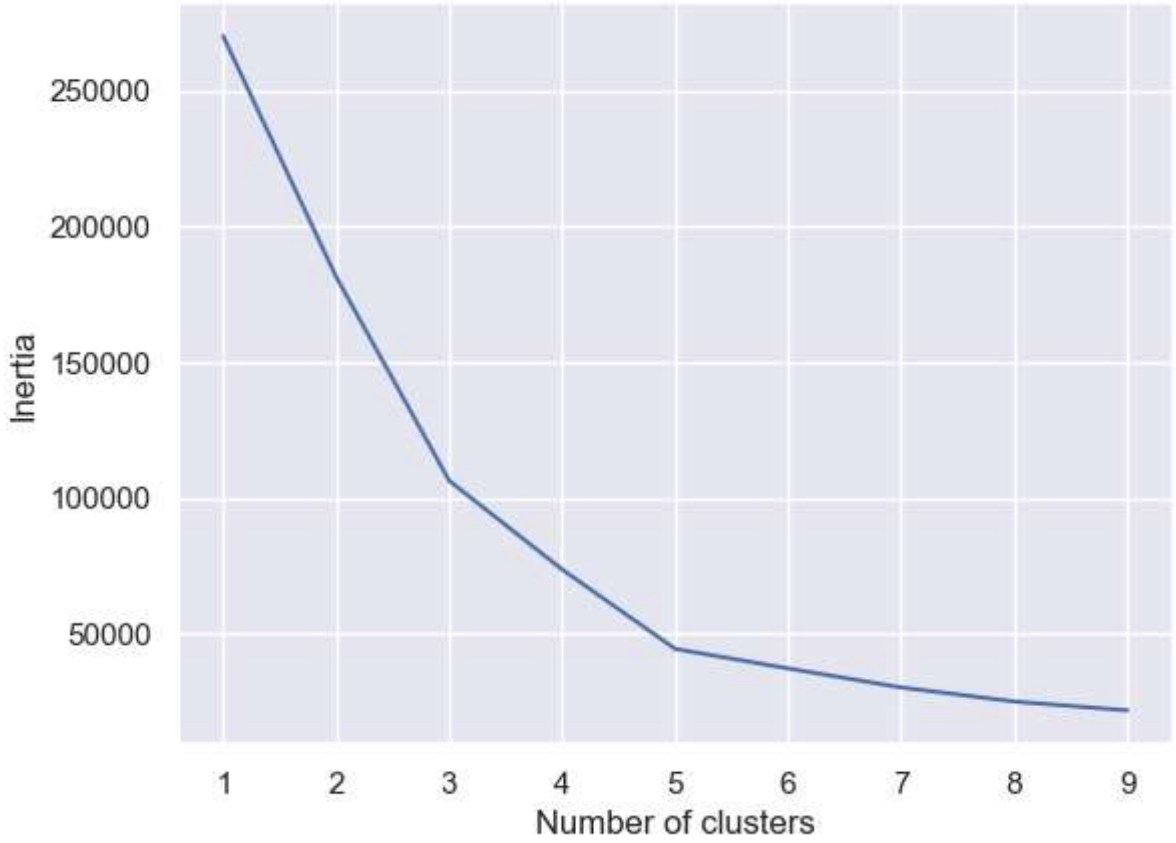


Используйте метод локтя для определения оптимального количества кластеров.

In [25]:
```
inertias = []

for i in range(1, 10):
    kmeans = KMeans(n_clusters=i, random_state=0)
    kmeans.fit(points)
    inertias.append(kmeans.inertia_)
plt.plot(range(1, 10), inertias)
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
```

Out[25]:
```
Text(0, 0.5, 'Inertia')
```



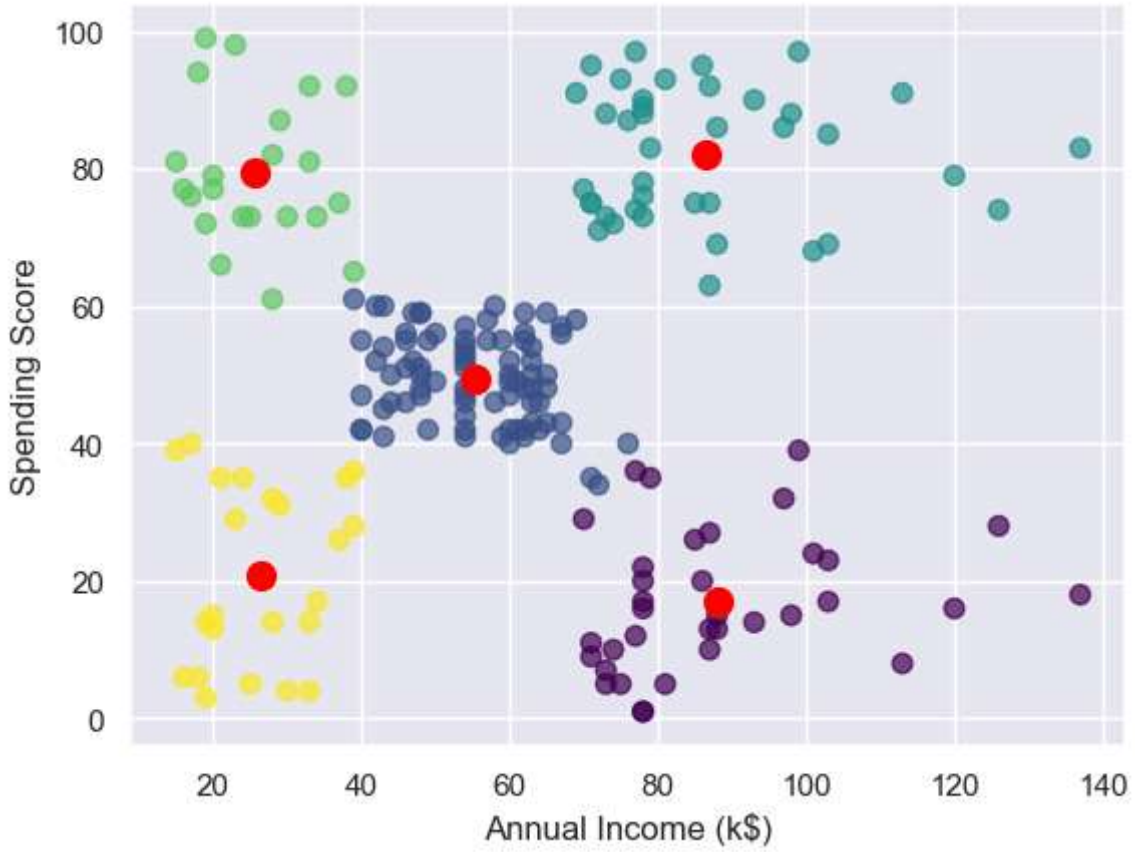
Из полученного результата следует, что оптимальное количество сегментов примерно пять

In [26]:
```
kmeans = KMeans(n_clusters=5, random_state=0)
kmeans.fit(points)
predicted_cluster_indexes = kmeans.predict(points)

plt.scatter(x, y, c=predicted_cluster_indexes, s=50, alpha=0.7, cmap='viridis')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score')

centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red', s=100)
```

Out[26]:
```
<matplotlib.collections.PathCollection at 0x12e9af609b0>
```



Клиенты, с которыми ассоциированы точки в правом нижней части диаграммы, могут стать хорошей целью для проведения рекламной акции, направленной на увеличение их расходов. Потому что у них высокие доходы, и низкие показатели расходов.

Создайте копию DataFrame и добавьте столбец Cluster, содержащий кластерные индексы

In [28]:
```
df = customers.copy()
df['Cluster'] = kmeans.predict(points)
df.head()
```

Out[28]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
0	1	Male	19	15	39	4
1	2	Male	21	15	81	3
2	3	Female	20	16	6	4
3	4	Female	23	16	77	3
4	5	Female	31	17	40	4

Теперь с помощью следующего кода выведите идентификаторы клиентов с высокими доходами, но низкими показателями расходов:

In [29]:
```
# Get the cluster index for a customer with a high income and low spending score
cluster = kmeans.predict(np.array([[120, 20]]))[0]

# Filter the DataFrame to include only customers in that cluster
clustered_df = df[df['Cluster'] == cluster]

# Show the customer IDs
clustered_df['CustomerID'].values
```

Out[29]:
```
array([125, 129, 131, 135, 137, 139, 141, 145, 147, 149, 151, 153, 155,
       157, 159, 161, 163, 165, 167, 169, 171, 173, 175, 177, 179, 181,
       183, 185, 187, 189, 191, 193, 195, 197, 199], dtype=int64)
```

Полученные идентификаторы клиентов можно использовать для извлечения имен и адресов электронной почты из базы данных клиентов

Ключевым моментом здесь является то, что применялась *кластеризация* для группировки клиентов по годовому доходу и рейтингу расходов.

Сегментирование клиентов по более чем двум показателям

Предыдущем примере использовались всего две переменные: годовой доход и рейтинг расходов. Теперь давайте снова сегментируем клиентов, используя все показатели, кроме идентификаторов клиентов.

Начнем с замены строк Male (мужчина) и Female (женщина) в столбце Gender (пол) на 1 и О. Данный процесс называется кодированием меток. Это необходимо, поскольку машинное обучение может работать только с числовыми данными.

In [30]:
```
from sklearn.preprocessing import LabelEncoder

df = customers.copy()
encoder = LabelEncoder()
df['Gender'] = encoder.fit_transform(df['Gender'])
df.head()
```

Out[30]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)
0	1	1	19	15	39
1	2	1	21	15	81
2	3	0	20	16	6
3	4	0	23	16	77
4	5	0	31	17	40

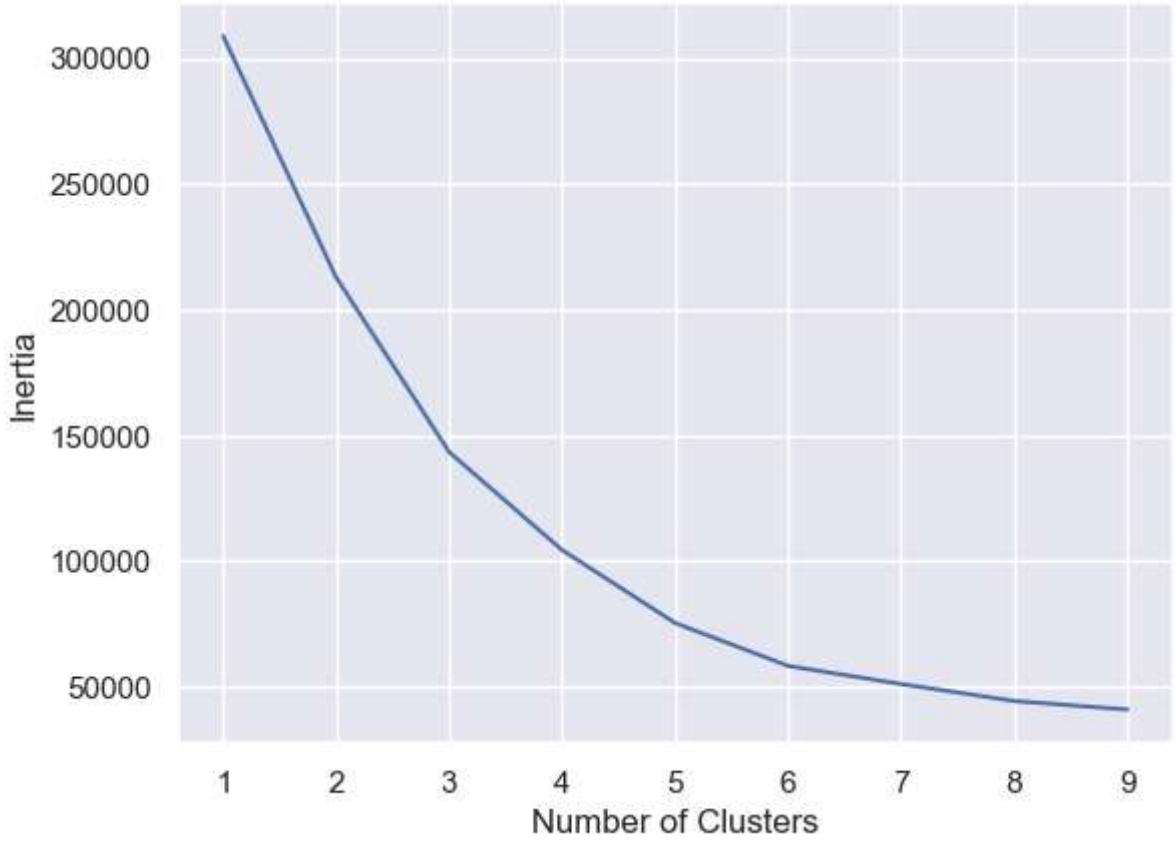
Извлечь столбцы Gender, Age, Annual Incorie и Spending Score. Затем с помощью метода локтя определим оптимальное количество кластеров на основе этих признаков

In [31]:
```
inertias = []
points = df.iloc[:, 1:5].values

for i in range(1, 10):
    kmeans = KMeans(n_clusters=i, random_state=0)
    kmeans.fit(points)
    inertias.append(kmeans.inertia_)

plt.plot(range(1, 10), inertias)
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
```

Out[31]:
```
Text(0, 0.5, 'Inertia')
```



На этот раз локоть менее отчетлив, но 5 является вполне приемлемым значением. Поэтому разделите клиентов на пять кластеров и добавьте столбец с именем Cluster

In [32]:
```
kmeans = KMeans(n_clusters=5, random_state=0)
kmeans.fit(points)

df['Cluster'] = kmeans.predict(points)
df.head()
```

Out[32]:

	CustomerID	Gender	Age	Annual Income (k$)	Spending Score (1-100)	Cluster
0	1	1	19	15	39	0
1	2	1	21	15	81	4
2	3	0	20	16	6	0
3	4	0	23	16	77	4
4	5	0	31	17	40	0

У вас есть значение кластера для каждого клиента. Вы не можете отразить пол, возраст, годовой доход и рейтинг расходов на двумерной диаграмме так же, как вы строили годовой доход и рейтинг расходов в предыдущем примере. Но можно получить среднее значение этих величин для каждого кластера по центроидам кластеров.

Поэтому создайте новый DataFrame со столбцами для среднего возраста, среднего дохода и т. д., а затем отобразите результаты в виде таблицы

In [33]:
```
results = pd.DataFrame(columns = ['Cluster', 'Average Age', 'Average Income', 'Average Spending Index', 'Number of Fem

for i, center in enumerate(kmeans.cluster_centers_):
    age = center[1]    # Average age for current cluster
    income = center[2] # Average income for current cluster
    spend = center[3]  # Average spending score for current cluster

    gdf = df[df['Cluster'] == i]
    females = gdf[gdf['Gender'] == 0].shape[0]
    males = gdf[gdf['Gender'] == 1].shape[0]

    results.loc[i] = ([i, age, income, spend, females, males])

results.head()
```

Out[33]:

	Cluster	Average Age	Average Income	Average Spending Index	Number of Females	Number of Males
0	0.0	45.217391	26.304348	20.913043	14.0	9.0
1	1.0	32.692308	86.538462	82.128205	21.0	18.0
2	2.0	43.088608	55.291139	49.569620	46.0	33.0
3	3.0	40.666667	87.750000	17.583333	17.0	19.0
4	4.0	25.521739	26.304348	78.565217	14.0	9.0

Д/з. Исходя из этого, если бы вы собирались ориентироваться на покупателей с высоким доходом, но низким рейтингом расходов, какую группу покупателей (какой кластер) вы бы выбрали для проведения рекламной акции? Будет ли иметь значение, на кого ориентироваться - на мужчин или женщин?

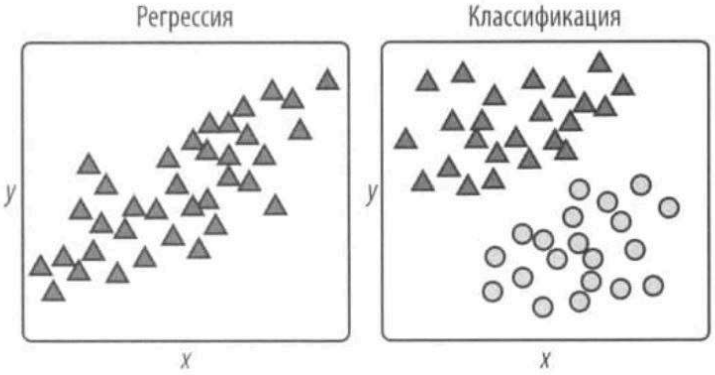
Например, больше всех тратят молодые люди (средний возраст которых составляет 25,5 лет) со скромными доходами. Среди этих покупателей больше женщин, чем мужчин.

Таким образом, мы увидели пример, где компания использует машинное обучение для улучшения своих бизнес-процессов.

Обучение с учителем

Модели контролируемого обучения делают прогнозы . Например, они предсказывают, является ли операция по кредитной карте мошеннической или прилетит ли рейс вовремя. Они также обучаются на данных с метками. Модели контролируемого обучения бывают двух видов: *регрессионные* и *классификационные*. Цель *регрессионной модели* - предсказать числовой результат, например цену, по которой будет продан дом, или возраст человека на фотографии. *Классификационные модели*, напротив, предсказывают класс или категорию из конечного набора классов, определенных в обучающих данных. В качестве примера можно привести вопрос о том, является ли операция по кредитной карте законной или мошеннической, а также о том, какое число представляет собой написанная от руки цифра. Первая модель является *бинарной классификацией*, поскольку существует только два возможных исхода: транзакция является законной или нет. Вторая модель - пример множественной классификацuu. Поскольку в западной арабской системе счисления имеется десять цифр (0-9), значит существует десять возможных классов, которые может представлять написанная от руки цифра.

Ниже показаны два типа моделей контролируемого обучения. В модели слева задача состоит в том, чтобы ввести x и предсказать, каким будет y. Справа - ввести x и y и предсказать, какому классу соответствует точка : треугольнику или эллипсу. В обоих случаях целью применения машинного обучения к задаче является построение модели для прогнозирования. Вместо того чтобы строить эту модель



Для этих наборов данных можно легко построить математические модели, не прибегая к машинному обучению.

Для регрессионной модели можно провести линию через точки данных и использовать уравнение этой линии для прогнозирования значения y по значению x.

Для классификационной модели можно провести линию, которая четко отделяет треугольники от эллипсов, и предсказать, к какому классу относится новая точка, определив, находится ли она выше или ниже этой линии. Точка, находящаяся выше линии, будет треугольником, а точка, находящаяся ниже линии, будет классифицироваться как эллипс.

В реальном мире наборы данных редко бывают настолько упорядоченными. Обычно в них нет ни одной линии, которую можно было бы провести, чтобы соотнести значения x и y или четко разделить классы. Поэтому задача состоит в том, чтобы построить наилучшую модель. Это означает выбор алгоритма обучения, который дает наиболее точную модель.

Существует множество алгоритмов контролируемого обучения. Например, линейная регрессия, случайный лес, градиентный бустинг, метод опорных векторов и т.д.

Метод k ближайших соседей

Одним из самых простых алгоритмов контролируемого обучения является алгоритм k ближайших соседей (k-nearest neighbors). Суть его заключается в том, что при заданном наборе данных можно предсказать метку для новой точки, изучив ближайшие к ней точки .

Для простой задачи регрессии, в которой каждая точка данных характеризуется координатами x и y, это означает, что при заданном x можно предсказать y, найдя n точек, ближайших по x, и усреднив их y.

Для задачи классификации необходимо найти n точек, ближайших к точке, класс которой требуется предсказать, и выбрать класс с наибольшим числом встречаемости.

Использование k ближайших соседей для классификации цветов

В состав Scikit-Leam входят классы KNeighborsRegressor и KNeighborsClassifier, помогающие обучать регрессионные и классификационные модели с использованием алгоритма обучения k ближайших соседей.

Рассмотрим пример, в котором KNeighborsClassifier используется для классификации цветов на основе известного набора данных Iris. Этот набор данных включает 150 образцов, каждый из которых представляет один из трех видов ирисов. Каждая строка содержит четыре параметра: длина чашелистика, ширина чашелистика, длина лепестка и ширина лепестка, - все в сантиметрах, а также метку: 0 - setosa, 1 - versicolor и 2 - virginica. (petal - лепесток, sepal - чашелистик)

Набор данных Iris является одним из нескольких примеров наборов данных, входящих в состав Scikit. Поэтому его можно загрузить, вызвав функцию load_iris, а не читать из внешнего файла.

iris setosa

iris versicolor

iris virginica

In [97]:
```
# Подготовка модели машинного обучения, позволяющей различать
# виды ирисов по размерам чашелистиков и лепестков
import pandas as pd
from sklearn.datasets import load_iris

iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['class'] = iris.target
df['class name'] = iris.target_names[iris['target']]
df.head()
```

Out[97]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	class	class name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

Функция train_test_split в Scikit позволяет легко разделить набор данных, 80% строк отводится для обучения, а 20% - для тестирования

In [98]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_state=0)
```

In [99]:
```
# обучение модели
# В Scikit модель машинного обучения создается путем вызова класса, содержащего выбранный алгоритм обучения,
# в данном случае KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier()
model.fit(x_train, y_train)
```

Out[99]:
```
KNeighborsClassifier()
```

In [100]:
```
model.score(x_test, y_test)
```

Out[100]:
```
0.9666666666666667
```

score возвращает значение 0,966667, что означает точность 97%, после того, как модель предсказала признаки в x_test и сравнила предсказанные метки с реальными метками в y_test.

В Scikit предсказание осуществляется путем вызова метода predict модели. Необходимо определить вид ириса, длина чашелистика которого составляет 5,6 см, ширина чашелистика - 4,4 см, длина лепестка - 1,2 см, а ширина лепестка - 0,4 см

In [101]:
```
predicted_class = model.predict([[5.6, 4.4, 1.2, 0.4]])
print(predicted_class)
```

```
[0]
```

Метод predict может говорить несколько предсказаний за один вызов. Именно поэтому ему передается список списков, а не просто список. Метод возвращает список, длина которого равна количеству переданных списков.

В данном примере предсказанный класс равен О, т. е. модель предсказала, что ирис, длина чашелистика которого составляет 5,6 см, ширина чашелистика - 4,4 см, длина лепестка - 1,2 см, а ширина лепестка - 0,4 см, скорее всего, является ирисом Setosa.

При создании KNeighborsClassifier без указания количества соседей по умолчанию используется значение 5. Вы можете указать

Д/з Попробуйте выполнить настройку и оценить модель еще раз, используя n_neighbors=10. Меняется ли оценка модели? По-прежнему ли модель предсказывает класс O? Экспериментируйте с другими значениями n_neighbors.

In [ ]: