Lecture 3: OOP, classes, scripts and some other issues

- Everything is an object, so... everything!
- OOP
- Classes
- Some brief notes on other language features
- Sets and choice of data structures
- Scripting
- Presentation by Anders Märak Leffler
- Attribution: slightly extends work by Johan Falkenjack.
- License: CC-BY-SA 4.0 (https://creativecommons.org/licenses/by-sa/4.0/)

OOP in principle

- Paradigm. Structuring code.
  - Abstractions.
  - Rough resemblance between objects in the world, and the code.
  - Keeping related data and behaviour together.
    - Roughly: a bunch of data and the way they interact with the world.

Objects have state (and groups related data together)

In [ ]:

```
# Most extreme (somewhat strawmannish) contrast: entirely ungrouped values.
car_1_maker = "Volvo"
car_1_model = 240
car_1_colour = "black"
car_1_cost = 90000

car_2_maker = "Toyota"
car_2_model = "Camry"
car_2_colour = "red"
car_2_cost = 100000

# ...

# How do we pass a list of cars to a function which calculates eg if Toyotas are more e
xpensive?
```

In [3]:

```
# Grouping data together.

class Car:
    pass   # More later!

car_1 = Car()
car_1.maker = "Volvo"
car_1.model = 240
car_1.colour = "black"
car_1.cost = 90000

car_2 = Car()
car_2.maker = "Volvo"
car_2.model = "Camry"
car_2.colour = "red"
car_2.cost = 10000

cars = [car_1, car_2]
cars
cars[0].colour
car_1.cost = 9999999999999
cars[0].cost
```

Out[3]:

```
9999999999999
```

Note: not only grouping, but now we can change the cost of a *specific* car.

In [4]:

```
# Grouping data together, somewhat more structured fashion.
# Homespun version to avoid the code repetition above. (not part of
# Python)
def bad_init(maker, model, cost, colour):
    res = Car()
    res.maker = maker
    res.model = model
    res.cost = cost
    res.colour = colour
    return res

car_test = bad_init("volvo", 240, 12313, "white")

# Creating a "blueprint" for what a car (in our application) _is_.

class Car:
    # Somehow defining that a car _is_ something which has all these properties.
    # (in our application)

    def __init__(self, maker, model, cost, colour = "black"):
        self.maker = maker
        self.model = model
        self.cost = cost
        self.colour = colour


car_1 = Car(maker = "Volvo", model = 240, cost = 90000)  # black is the default colour
car_2 = Car(maker = "Toyota", model = "Camry", colour = "red", cost = 10000)
cars = [car_1, car_2]
cars
```

Out[4]:

```
[<__main__.Car at 0x7f32fd139e10>, <__main__.Car at 0x7f32fd139f28>]
```

In [10]:

```
cars = [Car(maker = "Volvo", model = 240, colour = "black", cost = 90000),
       Car(maker = "Toyota", model = "Camry", colour = "red", cost = 10000)]
cars
```

Out[10]:

```
[<__main__.Car at 0x7fc1f80d3e80>, <__main__.Car at 0x7fc1f80d3e10>]
```

In [ ]:

```
# With a data source such as an API connection which gives us data in a certain format,
a CSV file handle,
# we can automate this.

# Nothing specific to OOP so far, but noteworthy feature.

some_magic_data_iterable = [("Volvo", "black", 240, 90000), ("Toyota", "red", "Camry",
10000)]

cars = [Car(maker = maker, colour = colour, model = model, cost = cost)
        for maker, colour, model, cost in some_magic_data_iterable]
```

Notes:

- Couldn't we do this via some other abstraction (namedtuple, or tuples + a function which picks out the right parts, such as `get_maker(car) == car[0]`)...?
- Now we have objects with state. What about "behaviours"?
  - (Philosophical question for those so inclined: structs vs objects.)

Objects act on messages (and the concept of interface)

- (Public) interface: What messages does an object accept, and what does it do or return?

In [13]:

```
my_seq = [1,2,3]   # State before: my_seq is a list with these three elements.
other_seq = [4,5]
my_seq.append(999)  # What does this mean?
# Intuitive here.
# Formally: tell my_seq to change itself.
help(my_seq.append)
```

```
Help on built-in function append:

append(...) method of builtins.list instance
    L.append(object) -> None -- append object to end
```

- Objects carry their own behaviours "wrapped in" [not technically].
  - Ex: `classifier_1.classify(image)` uses classifier_1's `classify` method.
    `classifier_2.classify(image)` might use some entirely different procedure.

    (Contrast: some external function `classify(classifier_1, image)` , where the behaviour is not carried with the object).

Encapsulation. Objects should be isolated, and hide implementation

- Carry their own data, or references to where to get it.
- Carry their own behaviours.
  - (Corollary) Avoids dependence on other objects' implementation.

Corollary: other parts of the program shouldn't need to know a lot about *how* your object does things. And if you change how it does things, their code shouldn't break.

With great power comes great responsibility

- Default: Python will let you.

Classes in Python

- We use `class` to create classes.
  - `class Cat(): ...` which defines what it means to be a `Cat` object).
- We call the classes to create *instances*. The `__init__` method is called.
  - `Cat(name = "Alonzo")` to create a cat instance. This is just a value, a single cat.
  - Usually bind this to save value for later (`alonzo = Cat(name="Alonzo")`).

Design note: a particular cat is an instance, what is *common to all cats* belongs in the class.

Creating simple classes and instances

In [17]:

```
# Defining what is common to all cats.

class Cat:
    # More data goes here!

    # Initialiser
    def __init__(self, name = "Shere Khan"):
        self.name = name
        self.description = "adorable"

    def greet(self):
        return "Hi, my name is " + self.name

    # and it should have a "greet" method that returns a greeting string.

# Creating some instances.
alonzo = Cat(name = "Alonzo")
zeno = Cat(name = "Zeno")

# Calling a method (function attribute).

alonzo.greet()
#zeno.greet()
```

Out[17]:

```
'Hi, my name is Alonzo'
```

- Classes as namespaces. What attributes does `alonzo` have above?

In [18]:

```
dir(alonzo)
```

Out[18]:

```
['__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'description',
 'greet',
 'name']
```

In [19]:

```
help(dir)
```

```
Help on built-in function dir in module builtins:

dir(...)
    dir([object]) -> list of strings

    If called without an argument, return the names in the current scope.
    Else, return an alphabetized list of names comprising (some of) the at
tributes
    of the given object, and of attributes reachable from it.
    If the object supplies a method named __dir__, it will be used; otherw
ise
    the default dir() logic is used and returns:
      for a module object: the module's attributes.
      for a class object:  its attributes, and recursively the attributes
        of its bases.
      for any other object: its attributes, its class's attributes, and
        recursively the attributes of its class's base classes.
```

- What is the `self` ? Why do we pass it along?

In [20]:

```
alonzo.greet()
# corresponding to...
Cat.greet(alonzo)   # "self" is the alonzo object
```

Out[20]:

```
'Hi, my name is Alonzo'
```

In [28]:

```
# Added live after question about generating objects.

myobj = type("mynewtype", (object,), { "meth" : id}) # some function
myobj.meth(2)
```

Out[28]:

```
10910432
```

In [22]:

```
# Added live after question about adding functions.

Cat.meow = lambda self: "Meow"
alonzo.meow()
```

Out[22]:

```
'Meow'
```

In [23]:

```
alonzo.__dict__   # What's in the instance? Is there a meow?
```

Out[23]:

```
{'name': 'Alonzo', 'description': 'adorable'}
```

[C++? Think of it somewhat like a this pointer.]

- Can we access and check for the presence of attributes in other ways than trying with .-access as above?

In [29]:

```
help(getattr)
```

```
Help on built-in function getattr in module builtins:

getattr(...)
    getattr(object, name[, default]) -> value

    Get a named attribute from an object; getattr(x, 'y') is equivalent to
x.y.
    When a default argument is given, it is returned when the attribute do
esn't
    exist; without it, an exception is raised in that case.
```

In [ ]:

```
help(setattr)
```

In [ ]:

```
help(hasattr)
```

[More to come.]

- We can generate "more of the same". A note on `type` .

In [31]:

```
type(alonzo) #  gives us the class/constructor!
```

Out[31]:

```
<__main__.Cat at 0x7fc1f808d8d0>
```

In [32]:

```
# Accessing the class of an object in a different way.
alonzo.__class__
```

Out[32]:

```
__main__.Cat
```

In [35]:

```
# Using it.
CatConstructor = type(alonzo)
fcat = CatConstructor("Forex")
fcat.name
```

Out[35]:

```
'Forex'
```

- Bonus: we can generate classes as well.

In [ ]:

```
help(type)
```

In [38]:

```
# Bonus: creating using type.

CC = type("CoolCat", (Cat,), { "a" : 5, "zoo" : lambda self : self.a })
CC
tomomalley = CC()
tomomalley.zoo()
```

Out[38]:

```
5
```

(Can be used to create types on the fly.)

- Can we have several initialisers (like several constructors in C++)?

In [ ]:

```
# Not possible to have this type of polymorph. in Python!
def __init__(self, maker, model, ...):
    pass

def __init__(self, readymadecar):
    pass

# Only the last definition survives!
```

- What about destructors? (For those used to C++).
  - Main takeaway: **Garbage collected language! No "need" for delete, delete[], free,... due to memory.**.
  - But sometimes other kinds of cleanup is useful.

In [40]:

```
# Slightly contrived example.
# Check in detail on your own time.

class DataSource:
    def __init__(self, addr = None):
        self.api_connector = APIConnection(addr)   # Connect to the data source via the
internet.
        # ...

    def sign_off(self):
        """Perform an ordered sign-off from the data server."""
        self.api_connector.sign_off()
        self.api_connector = None

    def __del__(self):
        if self.api_connector is not None:
            print("Someone should've sent a proper signoff ")
            self.api_connector.signal_abnormal_use()   # "normal" deletion of the conne
ctor might send other signals

        # Note that api_connector:s destructor will normally be run when the GC passes
 by, and
        # there are no references to it. So you should really think about who's respons
ible
        # for an object, if you are to do this! Also, it might not be needed.

        print("The DataSource object destructor has finished. The GC will clean up the
 api_connector...")

class APIConnection:
    """Dummy class."""
    def __init__(self, address):
        pass

    def sign_off(self):
        pass

    def signal_abnormal_use(self):
        pass

def f():
    print("Entered f.")
    src = DataSource(addr = "api.scb.se")
    #src.sign_off()
    print("Exiting f now. src should go out of scope and be deleted soon:ish.")
f()
```

```
Entered f.
Exiting f now. src should go out of scope and be deleted soon:ish.
Someone should've sent a proper signoff
The DataSource object destructor has finished. The GC will clean up the ap
i_connector...
```

Concept: composition (has-a)

- Adding capabilities by generating objects of their own.
- Above: every `DataSource` instance has an `APIConnection` instance of its own.
  - Design choice: don't make giant DataSource class which has all the APIConnection methods inside.
  - Delegating that part of the work to a specialised class.
- See Lutz ch 31.

Concept: Inheritance (is-a)

- Conceptually: *every X is also a Y* (and can do everything it can).
  - Adding capabilities.
  - Specialisation.
- Operationally: add properties to the objects at the right level.
- Issue: how do we handle conflicts?

In [1]:

```
class PrimordialSoupObject:
    pass

class Animal(PrimordialSoupObject):
    def __init__(self):
        self.description = "Hello, this is Dog."
        self.x = "animal"

    def make_sound(self):
        print("Running Animal's make_sound.")
        print("Grrrr.")
        print("I believe that description is:", self.description)

class SuperHero(PrimordialSoupObject):
    def __init__(self, superpower = "Flight"):
        self.description = "Na"*50 + "Batman"
        self.x = "animal"
        self.superpower = superpower

    def make_sound(self):
        print("Running SuperHero's make_sound.")
        print("Truth, justice and the american way")
        print("I believe that description is:", self.description)

    def fight(self):
        print("POW!")

class Cat(Animal, SuperHero):
    def __init__(self, name = "Shere Khan"):

        # Run parent classes' inits (early on)
        SuperHero.__init__(self)  # added later
        Animal.__init__(self)


        self.description = "cat."
        self.name = name
        self.x = "animal"


    def sdfsdfsdfsdfmake_sound(self):
        print("Running Cat's make_sound.")
        print("Meow.")
        print("I believe that description is:", self.description)

# What will this print? What will it yield? Why?
alonzo = Cat(name = "Alonzo")
alonzo.make_sound()  # comes from Animal (since the name change)
alonzo.fight()  # only in SuperHero
```

```
Running Animal's make_sound.
Grrrr.
I believe that description is: cat.
POW!
```

...

- Method resolution order: [reference (https://www.python.org/download/releases/2.3/mro/)]

In [50]:

```
alonzo.__class__.__mro__   # In which order will I look for attributes?
```

Out[50]:

```
(__main__.Cat,
 __main__.SuperHero,
 __main__.Animal,
 __main__.PrimordialSoupObject,
 object)
```

In [48]:

```
import inspect
inspect.getmro(Cat)
```

Out[48]:

```
(__main__.Cat,
 __main__.Animal,
 __main__.SuperHero,
 __main__.PrimordialSoupObject,
 object)
```

- Initialisers

In [55]:

```
# Trying to access the superpower (see SuperHero initialiser).
# Crashes unless we run the SuperHero.__init__ somehow.
alonzo.superpower
```

Out[55]:

```
'Flight'
```

In [58]:

```
alonzo.make_sound()
```

```
Running SuperHero's make_sound.
Truth, justice and the american way
I believe that description is: cat.
```

In [ ]:

```
# Attempt I (super)
# super().__init__(self) in the initialiser.
```

In [51]:

```
# Attempt II (explicit)
# Animal.__init__ etc
```

Why care? Specialisation!

Finding out what kind of objects we're working with

Test if it is a direct instance.

In [60]:

```
# Can we test if alonzo is a Cat?
type(alonzo) is Cat
```

Out[60]:

```
True
```

Says something about the hierarchy.

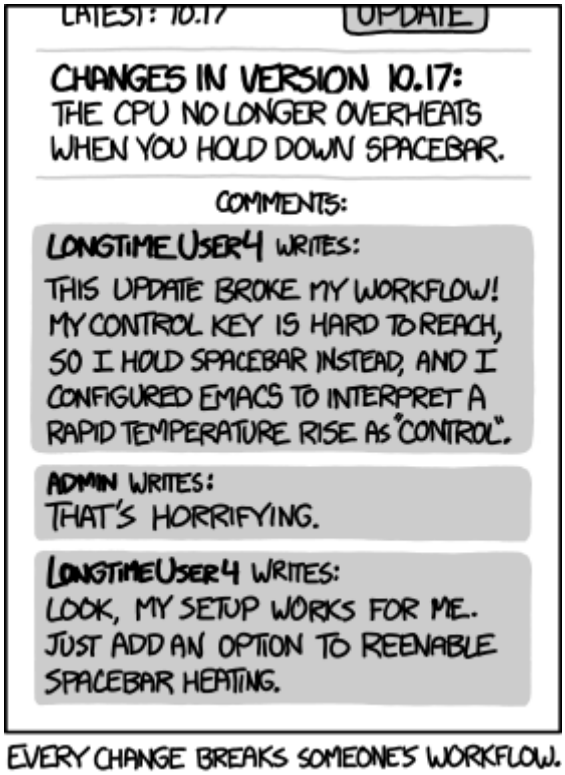
In [62]:

```
# Can we test if alonzo is an Animal?
isinstance(alonzo, Animal)  # Also check the inheritances!
```

Out[62]:

```
True
```

Conventions for "hiding" data



Dependencies, in [xkcd 1172 (https://xkcd.com/1172/)](https://xkcd.com/1172/).

- Many languages have public/private member distinctions.
- Pessimistic note above. (Hyrum's law (http://www.hyrumslaw.com/)).
- ...but let's try to hide things.
- By default everything in Python is public.

In [66]:

```
class SecretKeeper():
    def __init__(self, hidden):
        # self.hidden = hidden    # Initial code.
        self.__hidden = hidden    # Mangle name to "~hide"


c1 = SecretKeeper(hidden = "supersecret")
c1.hidden
```

```
---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
<ipython-input-66-9248b0c5a70f> in <module>
      6
      7 c1 = SecretKeeper(hidden = "supersecret")
----> 8 c1.hidden

AttributeError: 'SecretKeeper' object has no attribute 'hidden'
```

Can we hide it away somewhat?

In [67]:

```
dir(c1)
```

Out[67]:

```
['_SecretKeeper__hidden',
 '__class__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__le__',
 '__lt__',
 '__module__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__']
```

Can we find it anyway?

In [ ]:

```
# Yes, if you know name mangling. But take it as a signal from the programmer that you
 shouldn't.

# Left as exercise.
```

In [ ]:

```
# And yes, you can reach the instance variables even if you don't know name mangling.
# Still something to avoid.
```

Conclusion: follow the guidelines. **Assume that non-hidden attributes are public.**

- ...but we can tailor access by `__getattribute__` and `__setattr__` .

In [69]:

```
# Bonus task (outside the scope of this course): using custom methods to steer access.

class Mirror:
    def __getattribute__(self, name):

        # This function might do something special, eg pass the message on
        # over the internet, to an object on a remote server.

        return "You wanted {}, you say?".format(name)

val = Mirror()
print(val.asdasdasdfasdfasdf)
dir(val)  # not a lot here
```

```
You wanted asdasdasdfasdfasdf, you say?
```

Out[69]:

```
[]
```

Most decisions are made by the objects

In [74]:

```
# Somewhat contrived example!

class Snake():
    def __add__(self, other):
        return "Snake!"

class Ladder():
    def __add__(self, other):
        return "Hello, this is Ladder."

# Apart from possible issues with inheriting from int, will it commute?
p1 = Snake()
p2 = Ladder()
p2 + p1
```

Out[74]:

```
'Hello, this is Ladder.'
```

Why useful to know?

- Python methods carried by objects do the heavy lifting, even when there is no `.method()` in the call.
- Following conventions is good. If you implement your own classes, you might want `p1 + p2 == p2 + p1` to hold, even if Python doesn't force you.
- Builtins are hard to avoid. Shows why your nice `my_clever_vector * 5` might be different from `5 * my_clever_vector`.

Getters? Setters? Interesting feature: properties

- Controlling getting, assignment, deletion of value.
- What happens if we write `myobj.x = 5` ? Can be controlled via functions, like in eg C#.

In [82]:

```
class GetterTestClass():
    def __init__(self, n = 0):
        self.__n = n

    def __get_n(self):
        return self.__n

    def __set_n(self, new_val):
        if new_val >= 0:
            self.__n = new_val
        else:
            raise ValueError("n must be non-negative!")

    n = property(fget=__get_n, fset = __set_n)


    # Left out: fdel. When someone deletes an attribute.

    # When someone tries to get n, __get_n will be called. (Etc)


c1 = GetterTestClass(n = 100)
c1.n
c1.n = 3
#c1.n
```

Note: `@property` -syntax also available. You'll do this in lab 3A.

- Slightly weird. Sidesteps the usual `=` always meaning changing labels for class attributes.
- **Know that it's there, don't (ab)use**.

Useful pointer: dataclasses

- We might structure data by "dummy" classes.
- Predictably: initialisers with the relevant attributes, etc.
  - Prime example: the `Car` above.
- Since Python 3.7, we have dataclasses (https://docs.python.org/3/library/dataclasses.html) in the standard library. (Also pip-installed in the currently-Python 3.6.3 lab environment if you want to try it.)

Concepts to explore on your own

- Promises that a class should implement some behaviour (eg "this class should support sequence methods"), that can be checked by the system.

Abstract Base Classes (ABC:s) (https://docs.python.org/3/glossary.html#term-abstract-base-class).

In [ ]:

```
import collections
isinstance([1,2,3], collections.abc.Sequence)
```

See also the `abc` module in standard library.

- Mixins. Adding a capability to your class (note: here we include implementation). See Lutz.

In [83]:

```
# We can stick the capability to return five (or something more useful) into
# a class by just making it a subclass of ReturnFiveMixin.

class ReturnFiveMixin:
    def return_five(self):
        return 5

class MyClass(ReturnFiveMixin):
    # No code here!
    pass

obj = MyClass()
obj.return_five()
```

Out[83]:

```
5
```

- Class and instance attribute conventions.

In [86]:

```
class MyVal:
    common_to_all_myvals = 99
    def __init__(self, val):
        self.val = val


one_val = MyVal(val = 1)
two_val = MyVal(val = 2)
one_val.val # in the instance
one_val.common_to_all_myvals # in the class
```

Out[86]:

```
99
```

In [88]:

```
one_val.common_to_all_myvals = "new value. Is this only in one_val?"
two_val.common_to_all_myvals # Actually, just a local name in one_val.
```

Out[88]:

```
99
```

Note: you may have class methods which are reached via `<Class>.method()`, as we might reach `MyVal.common_to_all_myvals` above.

Other interesting *language* details

- Decorators. Transforming Python functions. (Telltale sign in code: the `@` sign. `@something`, such as `@property`, `@dataclass`.)
- Annotations. PEP3107 (https://www.python.org/dev/peps/pep-3107/)

In [89]:

```
def sq(n : int,
       otherarg : "annotations can be anything" = ""):
    """Return the value n^2."""
    return n*n

sq.__annotations__   # can be used by some code analysis tools, documentation tools etc.
```

Out[89]:

```
{'n': int, 'otherarg': 'annotations can be anything'}
```

- Consequence: type information might be useful eg
  - for code analysis, See eg mypy (http://www.mypy-lang.org/).
  - IDE:s.

Sets (and built-in types)

- Why do we care?
- So far:
  - `list`. Mutable, quick access by index. Finding in general slow.
  - `tuple`. Immutable. Fast, efficient, quick access by index. But: finding in general slow.
  - `dict`. Mapping, based on hash tables. Very fast access by key, finding by value slow. Membership test very fast. Slight memory overhead. Requires values to be hashable.
    - Errata: insertion-order iteration since Python 3.7 (not *un*ordered).
- Yet another useful type: `set`. Based on hash tables, with **very fast membership tests**. Mathematical-set methods.

In [96]:

```
import profile, random

N = 9999999
vals_list = list(range(N))
vals_set  = set(range(N))

# Pick some random element, just for demonstration purposes.
needle = random.randint(0, N)

print("--- vals_list")
profile.run(str(needle) + " in vals_list")

print("--- vals_set")
profile.run(str(needle) + " in vals_set")
```

```
--- vals_list
         4 function calls in 0.079 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.079    0.079 :0(exec)
        1    0.000    0.000    0.000    0.000 :0(setprofile)
        1    0.079    0.079    0.079    0.079 <string>:1(<module>)
        1    0.000    0.000    0.079    0.079 profile:0(7884751 in vals_li
st)
        0    0.000             0.000          profile:0(profiler)


--- vals_set
         4 function calls in 0.000 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 :0(exec)
        1    0.000    0.000    0.000    0.000 :0(setprofile)
        1    0.000    0.000    0.000    0.000 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 profile:0(7884751 in vals_se
t)
        0    0.000             0.000          profile:0(profiler)
```

Conclusion: if you need to perform lots of lookups based on the keys, sets, dicts etc might be useful.

In [98]:

```
words = set(["cat", "snape", "doge"])
animals = set(["cat"])

# What are the commonalities?
words.intersection(animals)
```

Out[98]:

```
{'cat'}
```

In [99]:

```
# Which words are not in animals?
words.difference(animals)
```

Out[99]:

```
{'doge', 'snape'}
```

In [103]:

```
# How would we add a word?
animals.add("giraffe")
animals.add("giraffe")  # Redundant - sets contain no duplicates.
animals
```

Out[103]:

```
{'cat', 'giraffe'}
```

- Immutable version: `frozenset` . Can be used as a key in a dictionary.

In [ ]:

```

```

Aside: modules

- Those `import math` , `import my_own_module` ...
- Where do they come from? [reference (https://docs.python.org/3.7/tutorial/modules.html#the-module-search-path)]
  - Beginner's note: some installation system usually takes care of this for you. But useful to know if something breaks.
- When you write `my_module.py` you can do `import my_module` at least from the same directory (see search path above).
- You will see `__init__.py` around. This concerns packages (https://docs.python.org/3.7/tutorial/modules.html#packages).
- Namespaces.

In [93]:

```
# Example package: sklearn, and its datasets
import sklearn
import sklearn.datasets
help(sklearn.datasets)
```

```
Help on package sklearn.datasets in sklearn:

NAME
    sklearn.datasets

DESCRIPTION
    The :mod:`sklearn.datasets` module includes utilities to load dataset
s,
    including methods to load and fetch popular reference datasets. It als
o
    features some artificial data generators.

PACKAGE CONTENTS
    _svmlight_format
    base
    california_housing
    covtype
    kddcup99
    lfw
    mlcomp
    mldata
    olivetti_faces
    openml
    rcv1
    samples_generator
    setup
    species_distributions
    svmlight_format
    tests (package)
    twenty_newsgroups

FUNCTIONS
    clear_data_home(data_home=None)
        Delete all the content of the data home cache.

        Parameters
        ----------
        data_home : str | None
            The path to scikit-learn data dir.

    dump_svmlight_file(X, y, f, zero_based=True, comment=None, query_id=No
ne, multilabel=False)
        Dump the dataset in svmlight / libsvm file format.

        This format is a text-based format, with one sample per line. It d
oes
        not store zero valued features hence is suitable for sparse datase
t.

        The first element of each line can be used to store a target varia
ble
        to predict.

        Parameters
        ----------
        X : {array-like, sparse matrix}, shape = [n_samples, n_features]
            Training vectors, where n_samples is the number of samples and
            n_features is the number of features.

        y : {array-like, sparse matrix}, shape = [n_samples (, n_labels)]
            Target values. Class labels must be an
```

```
            integer or float, or array-like objects of integer or float fo
r
            multilabel classifications.

        f : string or file-like in binary mode
            If string, specifies the path that will contain the data.
            If file-like, data will be written to f. f should be opened in
binary
            mode.

        zero_based : boolean, optional
            Whether column indices should be written zero-based (True) or
one-based
            (False).

        comment : string, optional
            Comment to insert at the top of the file. This should be eithe
r a
            Unicode string, which will be encoded as UTF-8, or an ASCII by
te
            string.
            If a comment is given, then it will be preceded by one that id
entifies
            the file as having been dumped by scikit-learn. Note that not
all
            tools grok comments in SVMlight files.

        query_id : array-like, shape = [n_samples]
            Array containing pairwise preference constraints (qid in svmli
ght
            format).

        multilabel : boolean, optional
            Samples may have several labels each (see
            https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multil
abel.html)

            .. versionadded:: 0.17
               parameter *multilabel* to support multilabel datasets.

    fetch_20newsgroups(data_home=None, subset='train', categories=None, sh
uffle=True, random_state=42, remove=(), download_if_missing=True)
        Load the filenames and data from the 20 newsgroups dataset (classi
fication).

        Download it if necessary.

        =================   ==========
        Classes                     20
        Samples total            18846
        Dimensionality               1
        Features                  text
        =================   ==========

        Read more in the :ref:`User Guide <20newsgroups_dataset>`.

        Parameters
        ----------
        data_home : optional, default: None
            Specify a download and cache folder for the datasets. If None,
            all scikit-learn data is stored in '~/scikit_learn_data' subfo
```

```
lders.

        subset : 'train' or 'test', 'all', optional
            Select the dataset to load: 'train' for the training set, 'tes
t'
            for the test set, 'all' for both, with shuffled ordering.

        categories : None or collection of string or unicode
            If None (default), load all the categories.
            If not None, list of category names to load (other categories
            ignored).

        shuffle : bool, optional
            Whether or not to shuffle the data: might be important for mod
els that
            make the assumption that the samples are independent and ident
ically
            distributed (i.i.d.), such as stochastic gradient descent.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset shuffling. Pas
s an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        remove : tuple
            May contain any subset of ('headers', 'footers', 'quotes'). Ea
ch of
            these are kinds of text that will be detected and removed from
the
            newsgroup posts, preventing classifiers from overfitting on
            metadata.

            'headers' removes newsgroup headers, 'footers' removes blocks
at the
            ends of posts that look like signatures, and 'quotes' removes
lines
            that appear to be quoting another post.

            'headers' follows an exact standard; the other filters are not
always
            correct.

        download_if_missing : optional, True by default
            If False, raise an IOError if the data is not locally availabl
e
            instead of trying to download the data from the source site.

        Returns
        -------
        bunch : Bunch object
            bunch.data: list, length [n_samples]
            bunch.target: array, shape [n_samples]
            bunch.filenames: list, length [n_classes]
            bunch.DESCR: a description of the dataset.

    fetch_20newsgroups_vectorized(subset='train', remove=(), data_home=Non
e, download_if_missing=True, return_X_y=False)
        Load the 20 newsgroups dataset and vectorize it into token counts
(classification).
```

```
        Download it if necessary.

        This is a convenience function; the transformation is done using t
he
        default settings for
        :class:`sklearn.feature_extraction.text.CountVectorizer`. For more
        advanced usage (stopword filtering, n-gram extraction, etc.), comb
ine
        fetch_20newsgroups with a custom
        :class:`sklearn.feature_extraction.text.CountVectorizer`,
        :class:`sklearn.feature_extraction.text.HashingVectorizer`,
        :class:`sklearn.feature_extraction.text.TfidfTransformer` or
        :class:`sklearn.feature_extraction.text.TfidfVectorizer`.

        =================   ==========
        Classes                     20
        Samples total            18846
        Dimensionality          130107
        Features                  real
        =================   ==========

        Read more in the :ref:`User Guide <20newsgroups_dataset>`.

        Parameters
        ----------
        subset : 'train' or 'test', 'all', optional
            Select the dataset to load: 'train' for the training set, 'tes
t'
            for the test set, 'all' for both, with shuffled ordering.

        remove : tuple
            May contain any subset of ('headers', 'footers', 'quotes'). Ea
ch of
            these are kinds of text that will be detected and removed from
the
            newsgroup posts, preventing classifiers from overfitting on
            metadata.

            'headers' removes newsgroup headers, 'footers' removes blocks
at the
            ends of posts that look like signatures, and 'quotes' removes
lines
            that appear to be quoting another post.

        data_home : optional, default: None
            Specify an download and cache folder for the datasets. If Non
e,
            all scikit-learn data is stored in '~/scikit_learn_data' subfo
lders.

        download_if_missing : optional, True by default
            If False, raise an IOError if the data is not locally availabl
e
            instead of trying to download the data from the source site.

        return_X_y : boolean, default=False.
            If True, returns ``(data.data, data.target)`` instead of a Bun
ch
            object.

            .. versionadded:: 0.20
```

```
        Returns
        -------
        bunch : Bunch object
            bunch.data: sparse matrix, shape [n_samples, n_features]
            bunch.target: array, shape [n_samples]
            bunch.target_names: list, length [n_classes]
            bunch.DESCR: a description of the dataset.

        (data, target) : tuple if ``return_X_y`` is True

            .. versionadded:: 0.20

    fetch_california_housing(data_home=None, download_if_missing=True, ret
urn_X_y=False)
        Load the California housing dataset (regression).

        ==============     ==============
        Samples total               20640
        Dimensionality                  8
        Features                     real
        Target             real 0.15 - 5.
        ==============     ==============

        Read more in the :ref:`User Guide <california_housing_dataset>`.

        Parameters
        ----------
        data_home : optional, default: None
            Specify another download and cache folder for the datasets. By
default
            all scikit-learn data is stored in '~/scikit_learn_data' subfo
lders.

        download_if_missing : optional, default=True
            If False, raise a IOError if the data is not locally available
            instead of trying to download the data from the source site.


        return_X_y : boolean, default=False.
            If True, returns ``(data.data, data.target)`` instead of a Bun
ch
            object.

            .. versionadded:: 0.20

        Returns
        -------
        dataset : dict-like object with the following attributes:

        dataset.data : ndarray, shape [20640, 8]
            Each row corresponding to the 8 feature values in order.

        dataset.target : numpy array of shape (20640,)
            Each value corresponds to the average house value in units of
100,000.

        dataset.feature_names : array of length 8
            Array of ordered feature names used in the dataset.

        dataset.DESCR : string
```

```
            Description of the California housing dataset.

        (data, target) : tuple if ``return_X_y`` is True

            .. versionadded:: 0.20

        Notes
        ------

        This dataset consists of 20,640 samples and 9 features.

    fetch_covtype(data_home=None, download_if_missing=True, random_state=N
one, shuffle=False, return_X_y=False)
        Load the covertype dataset (classification).

        Download it if necessary.

        =================   ============
        Classes                        7
        Samples total             581012
        Dimensionality                54
        Features                     int
        =================   ============

        Read more in the :ref:`User Guide <covtype_dataset>`.

        Parameters
        ----------
        data_home : string, optional
            Specify another download and cache folder for the datasets. By
default
            all scikit-learn data is stored in '~/scikit_learn_data' subfo
lders.

        download_if_missing : boolean, default=True
            If False, raise a IOError if the data is not locally available
            instead of trying to download the data from the source site.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset shuffling. Pas
s an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        shuffle : bool, default=False
            Whether to shuffle dataset.

        return_X_y : boolean, default=False.
            If True, returns ``(data.data, data.target)`` instead of a Bun
ch
            object.

            .. versionadded:: 0.20

        Returns
        -------
        dataset : dict-like object with the following attributes:

        dataset.data : numpy array of shape (581012, 54)
            Each row corresponds to the 54 features in the dataset.
```

```
        dataset.target : numpy array of shape (581012,)
            Each value corresponds to one of the 7 forest covertypes with
values
            ranging between 1 to 7.

        dataset.DESCR : string
            Description of the forest covertype dataset.

        (data, target) : tuple if ``return_X_y`` is True

            .. versionadded:: 0.20

    fetch_kddcup99(subset=None, data_home=None, shuffle=False, random_stat
e=None, percent10=True, download_if_missing=True, return_X_y=False)
        Load the kddcup99 dataset (classification).

        Download it if necessary.

        =================   ====================================
        Classes                                             23
        Samples total                                  4898431
        Dimensionality                                      41
        Features            discrete (int) or continuous (float)
        =================   ====================================

        Read more in the :ref:`User Guide <kddcup99_dataset>`.

        .. versionadded:: 0.18

        Parameters
        ----------
        subset : None, 'SA', 'SF', 'http', 'smtp'
            To return the corresponding classical subsets of kddcup 99.
            If None, return the entire kddcup 99 dataset.

        data_home : string, optional
            Specify another download and cache folder for the datasets. By
default
            all scikit-learn data is stored in '~/scikit_learn_data' subfo
lders.
            .. versionadded:: 0.19

        shuffle : bool, default=False
            Whether to shuffle dataset.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset shuffling and
for
            selection of abnormal samples if `subset='SA'`. Pass an int fo
r
            reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        percent10 : bool, default=True
            Whether to load only 10 percent of the data.

        download_if_missing : bool, default=True
            If False, raise a IOError if the data is not locally available
            instead of trying to download the data from the source site.

        return_X_y : boolean, default=False.
```

```
            If True, returns ``(data, target)`` instead of a Bunch object.
    See
            below for more information about the `data` and `target` objec
    t.

            .. versionadded:: 0.20

        Returns
        -------
        data : Bunch
            Dictionary-like object, the interesting attributes are:
             - 'data', the data to learn.
             - 'target', the regression target for each sample.
             - 'DESCR', a description of the dataset.

        (data, target) : tuple if ``return_X_y`` is True

            .. versionadded:: 0.20

    fetch_lfw_pairs(subset='train', data_home=None, funneled=True, resize=
    0.5, color=False, slice_=(slice(70, 195, None), slice(78, 172, None)), dow
    nload_if_missing=True)
        Load the Labeled Faces in the Wild (LFW) pairs dataset (classifica
    tion).

        Download it if necessary.

        =================   =======================
        Classes                                5749
        Samples total                         13233
        Dimensionality                         5828
        Features            real, between 0 and 255
        =================   =======================

        In the official `README.txt`_ this task is described as the
        "Restricted" task.  As I am not sure as to implement the
        "Unrestricted" variant correctly, I left it as unsupported for no
    w.

          .. _`README.txt`: http://vis-www.cs.umass.edu/lfw/README.txt

        The original images are 250 x 250 pixels, but the default slice an
    d resize
        arguments reduce them to 62 x 47.

        Read more in the :ref:`User Guide <labeled_faces_in_the_wild_datas
    et>`.

        Parameters
        ----------
        subset : optional, default: 'train'
            Select the dataset to load: 'train' for the development traini
    ng
            set, 'test' for the development test set, and '10_folds' for t
    he
            official evaluation set that is meant to be used with a 10-fol
    ds
            cross validation.

        data_home : optional, default: None
            Specify another download and cache folder for the datasets. By
```

```
            default all scikit-learn data is stored in '~/scikit_learn_dat
a'
            subfolders.

        funneled : boolean, optional, default: True
            Download and use the funneled variant of the dataset.

        resize : float, optional, default 0.5
            Ratio used to resize the each face picture.

        color : boolean, optional, default False
            Keep the 3 RGB channels instead of averaging them to a single
            gray level channel. If color is True the shape of the data has
            one more dimension than the shape with color = False.

        slice_ : optional
            Provide a custom 2D slice (height, width) to extract the
            'interesting' part of the jpeg files and avoid use statistical
            correlation from the background

        download_if_missing : optional, True by default
            If False, raise a IOError if the data is not locally available
            instead of trying to download the data from the source site.

        Returns
        -------
        The data is returned as a Bunch object with the following attribut
es:

        data : numpy array of shape (2200, 5828). Shape depends on ``subse
t``.
            Each row corresponds to 2 ravel'd face images of original size
62 x 47
            pixels. Changing the ``slice_``, ``resize`` or ``subset`` para
meters
            will change the shape of the output.

        pairs : numpy array of shape (2200, 2, 62, 47). Shape depends on `
`subset``
            Each row has 2 face images corresponding to same or different
person
            from the dataset containing 5749 people. Changing the ``slice_
``,
            ``resize`` or ``subset`` parameters will change the shape of t
he
            output.

        target : numpy array of shape (2200,). Shape depends on ``subset`
`.
            Labels associated to each pair of images. The two label values
being
            different persons or the same person.

        DESCR : string
            Description of the Labeled Faces in the Wild (LFW) dataset.

    fetch_lfw_people(data_home=None, funneled=True, resize=0.5, min_faces_
per_person=0, color=False, slice_=(slice(70, 195, None), slice(78, 172, No
ne)), download_if_missing=True, return_X_y=False)
        Load the Labeled Faces in the Wild (LFW) people dataset (classific
ation).
```

```
        Download it if necessary.

        =================   ======================
        Classes                                5749
        Samples total                         13233
        Dimensionality                         5828
        Features            real, between 0 and 255
        =================   ======================

        Read more in the :ref:`User Guide <labeled_faces_in_the_wild_datas
et>`.

        Parameters
        ----------
        data_home : optional, default: None
            Specify another download and cache folder for the datasets. By
default
            all scikit-learn data is stored in '~/scikit_learn_data' subfo
lders.

        funneled : boolean, optional, default: True
            Download and use the funneled variant of the dataset.

        resize : float, optional, default 0.5
            Ratio used to resize the each face picture.

        min_faces_per_person : int, optional, default None
            The extracted dataset will only retain pictures of people that
have at
            least `min_faces_per_person` different pictures.

        color : boolean, optional, default False
            Keep the 3 RGB channels instead of averaging them to a single
            gray level channel. If color is True the shape of the data has
            one more dimension than the shape with color = False.

        slice_ : optional
            Provide a custom 2D slice (height, width) to extract the
            'interesting' part of the jpeg files and avoid use statistical
            correlation from the background

        download_if_missing : optional, True by default
            If False, raise a IOError if the data is not locally available
            instead of trying to download the data from the source site.

        return_X_y : boolean, default=False.
            If True, returns ``(dataset.data, dataset.target)`` instead of
a Bunch
            object. See below for more information about the `dataset.data
` and
            `dataset.target` object.

            .. versionadded:: 0.20

        Returns
        -------
        dataset : dict-like object with the following attributes:

        dataset.data : numpy array of shape (13233, 2914)
            Each row corresponds to a ravelled face image of original size
```

```
62 x 47
            pixels. Changing the ``slice_`` or resize parameters will chan
ge the
            shape of the output.

        dataset.images : numpy array of shape (13233, 62, 47)
            Each row is a face image corresponding to one of the 5749 peop
le in
            the dataset. Changing the ``slice_`` or resize parameters will
change
            the shape of the output.

        dataset.target : numpy array of shape (13233,)
            Labels associated to each face image. Those labels range from
 0-5748
            and correspond to the person IDs.

        dataset.DESCR : string
            Description of the Labeled Faces in the Wild (LFW) dataset.

        (data, target) : tuple if ``return_X_y`` is True

            .. versionadded:: 0.20

    fetch_mldata(dataname, target_name='label', data_name='data', transpos
e_data=True, data_home=None)
        DEPRECATED: fetch_mldata was deprecated in version 0.20 and will b
e removed in version 0.22

        Fetch an mldata.org data set

            mldata.org is no longer operational.

            If the file does not exist yet, it is downloaded from mldata.o
rg .

            mldata.org does not have an enforced convention for storing da
ta or
            naming the columns in a data set. The default behavior of this
function
            works well with the most common cases:

              1) data values are stored in the column 'data', and target v
alues in the
                 column 'label'
              2) alternatively, the first column stores target values, and
the second
                 data values
              3) the data array is stored as `n_features x n_samples` , an
d thus needs
                 to be transposed to match the `sklearn` standard

            Keyword arguments allow to adapt these defaults to specific da
ta sets
            (see parameters `target_name`, `data_name`, `transpose_data`,
 and
            the examples below).

            mldata.org data sets may have multiple columns, which are stor
ed in the
            Bunch object with their original name.
```

```
            .. deprecated:: 0.20
                Will be removed in version 0.22

            Parameters
            ----------

            dataname : str
                Name of the data set on mldata.org,
                e.g.: "leukemia", "Whistler Daily Snowfall", etc.
                The raw name is automatically converted to a mldata.org UR
L .

            target_name : optional, default: 'label'
                Name or index of the column containing the target values.

            data_name : optional, default: 'data'
                Name or index of the column containing the data.

            transpose_data : optional, default: True
                If True, transpose the downloaded data array.

            data_home : optional, default: None
                Specify another download and cache folder for the data set
s. By default
                all scikit-learn data is stored in '~/scikit_learn_data' s
ubfolders.

            Returns
            -------

            data : Bunch
                Dictionary-like object, the interesting attributes are:
                'data', the data to learn, 'target', the classification la
bels,
                'DESCR', the full description of the dataset, and
                'COL_NAMES', the original names of the dataset columns.

    fetch_olivetti_faces(data_home=None, shuffle=False, random_state=0, do
wnload_if_missing=True)
        Load the Olivetti faces data-set from AT&T (classification).

        Download it if necessary.

        =================   ====================
        Classes                               40
        Samples total                        400
        Dimensionality                      4096
        Features            real, between 0 and 1
        =================   ====================

        Read more in the :ref:`User Guide <olivetti_faces_dataset>`.

        Parameters
        ----------
        data_home : optional, default: None
            Specify another download and cache folder for the datasets. By
default
            all scikit-learn data is stored in '~/scikit_learn_data' subfo
lders.
```

```
        shuffle : boolean, optional
            If True the order of the dataset is shuffled to avoid having
            images of the same person grouped.

        random_state : int, RandomState instance or None (default=0)
            Determines random number generation for dataset shuffling. Pas
s an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        download_if_missing : optional, True by default
            If False, raise a IOError if the data is not locally available
            instead of trying to download the data from the source site.

        Returns
        -------
        An object with the following attributes:

        data : numpy array of shape (400, 4096)
            Each row corresponds to a ravelled face image of original size
            64 x 64 pixels.

        images : numpy array of shape (400, 64, 64)
            Each row is a face image corresponding to one of the 40 subjec
ts
            of the dataset.

        target : numpy array of shape (400, )
            Labels associated to each face image. Those labels are ranging
from
            0-39 and correspond to the Subject IDs.

        DESCR : string
            Description of the modified Olivetti Faces Dataset.

    fetch_openml(name=None, version='active', data_id=None, data_home=Non
e, target_column='default-target', cache=True, return_X_y=False)
        Fetch dataset from openml by name or dataset id.

        Datasets are uniquely identified by either an integer ID or by a
        combination of name and version (i.e. there might be multiple
        versions of the 'iris' dataset). Please give either name or data_i
d
        (not both). In case a name is given, a version can also be
        provided.

        Read more in the :ref:`User Guide <openml>`.

        .. note:: EXPERIMENTAL

            The API is experimental in version 0.20 (particularly the retu
rn value
            structure), and might have small backward-incompatible changes
in
            future releases.

        Parameters
        ----------
        name : str or None
            String identifier of the dataset. Note that OpenML can have mu
ltiple
```

```
            datasets with the same name.

        version : integer or 'active', default='active'
            Version of the dataset. Can only be provided if also ``name``
 is given.
            If 'active' the oldest version that's still active is used. Si
nce
            there may be more than one active version of a dataset, and th
ose
            versions may fundamentally be different from one another, sett
ing an
            exact version is highly recommended.

        data_id : int or None
            OpenML ID of the dataset. The most specific way of retrieving
 a
            dataset. If data_id is not given, name (and potential version)
are
            used to obtain a dataset.

        data_home : string or None, default None
            Specify another download and cache folder for the data sets. B
y default
            all scikit-learn data is stored in '~/scikit_learn_data' subfo
lders.

        target_column : string, list or None, default 'default-target'
            Specify the column name in the data to use as target. If
            'default-target', the standard target column a stored on the s
erver
            is used. If ``None``, all columns are returned as data and the
            target is ``None``. If list (of strings), all columns with the
se names
            are returned as multi-target (Note: not all scikit-learn class
ifiers
            can handle all types of multi-output combinations)

        cache : boolean, default=True
            Whether to cache downloaded datasets using joblib.

        return_X_y : boolean, default=False.
            If True, returns ``(data, target)`` instead of a Bunch object.
See
            below for more information about the `data` and `target` objec
ts.

        Returns
        -------

        data : Bunch
            Dictionary-like object, with attributes:

            data : np.array or scipy.sparse.csr_matrix of floats
                The feature matrix. Categorical features are encoded as or
dinals.
            target : np.array
                The regression target or classification labels, if applica
ble.
                Dtype is float if numeric, and object if categorical.
            DESCR : str
                The full description of the dataset
```

```
            feature_names : list
                The names of the dataset columns
            categories : dict
                Maps each categorical feature name to a list of values, su
ch
                that the value encoded as i is ith in the list.
            details : dict
                More metadata from OpenML

        (data, target) : tuple if ``return_X_y`` is True

            .. note:: EXPERIMENTAL

                This interface is **experimental** as at version 0.20 and
                subsequent releases may change attributes without notice
                (although there should only be minor changes to ``data``
                and ``target``).

            Missing values in the 'data' are represented as NaN's. Missing
values
            in 'target' are represented as NaN's (numerical target) or Non
e
            (categorical target)

    fetch_rcv1(data_home=None, subset='all', download_if_missing=True, ran
dom_state=None, shuffle=False, return_X_y=False)
        Load the RCV1 multilabel dataset (classification).

        Download it if necessary.

        Version: RCV1-v2, vectors, full sets, topics multilabels.

        =================   ====================
        Classes                               103
        Samples total                      804414
        Dimensionality                      47236
        Features            real, between 0 and 1
        =================   ====================

        Read more in the :ref:`User Guide <rcv1_dataset>`.

        .. versionadded:: 0.17

        Parameters
        ----------
        data_home : string, optional
            Specify another download and cache folder for the datasets. By
default
            all scikit-learn data is stored in '~/scikit_learn_data' subfo
lders.

        subset : string, 'train', 'test', or 'all', default='all'
            Select the dataset to load: 'train' for the training set
            (23149 samples), 'test' for the test set (781265 samples),
            'all' for both, with the training samples first if shuffle is
 False.
            This follows the official LYRL2004 chronological split.

        download_if_missing : boolean, default=True
            If False, raise a IOError if the data is not locally available
            instead of trying to download the data from the source site.
```

```
        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset shuffling. Pas
s an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        shuffle : bool, default=False
            Whether to shuffle dataset.

        return_X_y : boolean, default=False.
            If True, returns ``(dataset.data, dataset.target)`` instead of
a Bunch
            object. See below for more information about the `dataset.data
` and
            `dataset.target` object.

            .. versionadded:: 0.20

        Returns
        -------
        dataset : dict-like object with the following attributes:

        dataset.data : scipy csr array, dtype np.float64, shape (804414, 4
7236)
            The array has 0.16% of non zero values.

        dataset.target : scipy csr array, dtype np.uint8, shape (804414, 1
03)
            Each sample has a value of 1 in its categories, and 0 in other
s.
            The array has 3.15% of non zero values.

        dataset.sample_id : numpy array, dtype np.uint32, shape (804414,)
            Identification number of each sample, as ordered in dataset.da
ta.

        dataset.target_names : numpy array, dtype object, length (103)
            Names of each target (RCV1 topics), as ordered in dataset.targ
et.

        dataset.DESCR : string
            Description of the RCV1 dataset.

        (data, target) : tuple if ``return_X_y`` is True

            .. versionadded:: 0.20

    fetch_species_distributions(data_home=None, download_if_missing=True)
        Loader for species distribution dataset from Phillips et. al. (200
6)

        Read more in the :ref:`User Guide <datasets>`.

        Parameters
        ----------
        data_home : optional, default: None
            Specify another download and cache folder for the datasets. By
default
            all scikit-learn data is stored in '~/scikit_learn_data' subfo
lders.
```

```
        download_if_missing : optional, True by default
            If False, raise a IOError if the data is not locally available
            instead of trying to download the data from the source site.

        Returns
        --------
        The data is returned as a Bunch object with the following attribut
es:

        coverages : array, shape = [14, 1592, 1212]
            These represent the 14 features measured at each point of the
 map grid.
            The latitude/longitude values for the grid are discussed belo
w.
            Missing data is represented by the value -9999.

        train : record array, shape = (1624,)
            The training points for the data.  Each point has three field
s:

            - train['species'] is the species name
            - train['dd long'] is the longitude, in degrees
            - train['dd lat'] is the latitude, in degrees

        test : record array, shape = (620,)
            The test points for the data.  Same format as the training dat
a.

        Nx, Ny : integers
            The number of longitudes (x) and latitudes (y) in the grid

        x_left_lower_corner, y_left_lower_corner : floats
            The (x,y) position of the lower-left corner, in degrees

        grid_size : float
            The spacing between points of the grid, in degrees

        References
        ----------

        * `"Maximum entropy modeling of species geographic distributions"
          <http://rob.schapire.net/papers/ecolmod.pdf>`_
          S. J. Phillips, R. P. Anderson, R. E. Schapire - Ecological Mode
lling,
          190:231-259, 2006.

        Notes
        -----

        This dataset represents the geographic distribution of species.
        The dataset is provided by Phillips et. al. (2006).

        The two species are:

        - `"Bradypus variegatus"
          <http://www.iucnredlist.org/details/3038/0>`_ ,
          the Brown-throated Sloth.

        - `"Microryzomys minutus"
          <http://www.iucnredlist.org/details/13408/0>`_ ,
```

```
          also known as the Forest Small Rice Rat, a rodent that lives in
 Peru,
          Colombia, Ecuador, Peru, and Venezuela.

        - For an example of using this dataset with scikit-learn, see
          :ref:`examples/applications/plot_species_distribution_modeling.p
y
          <sphx_glr_auto_examples_applications_plot_species_distribution_m
odeling.py>`.

    get_data_home(data_home=None)
        Return the path of the scikit-learn data dir.

        This folder is used by some large dataset loaders to avoid downloa
ding the
        data several times.

        By default the data dir is set to a folder named 'scikit_learn_dat
a' in the
        user home folder.

        Alternatively, it can be set by the 'SCIKIT_LEARN_DATA' environmen
t
        variable or programmatically by giving an explicit folder path. Th
e '~'
        symbol is expanded to the user home folder.

        If the folder does not already exist, it is automatically created.

        Parameters
        ----------
        data_home : str | None
            The path to scikit-learn data dir.

    load_boston(return_X_y=False)
        Load and return the boston house-prices dataset (regression).

        ==============     ==============
        Samples total                506
        Dimensionality                13
        Features           real, positive
        Targets             real 5. - 50.
        ==============     ==============

        Read more in the :ref:`User Guide <boston_dataset>`.

        Parameters
        ----------
        return_X_y : boolean, default=False.
            If True, returns ``(data, target)`` instead of a Bunch object.
            See below for more information about the `data` and `target` o
bject.

            .. versionadded:: 0.18

        Returns
        -------
        data : Bunch
            Dictionary-like object, the interesting attributes are:
            'data', the data to learn, 'target', the regression targets,
            'DESCR', the full description of the dataset,
```

```
            and 'filename', the physical location of boston
            csv dataset (added in version `0.20`).

        (data, target) : tuple if ``return_X_y`` is True

            .. versionadded:: 0.18

        Notes
        -----
            .. versionchanged:: 0.20
                Fixed a wrong data point at [445, 0].

        Examples
        --------
        >>> from sklearn.datasets import load_boston
        >>> boston = load_boston()
        >>> print(boston.data.shape)
        (506, 13)

    load_breast_cancer(return_X_y=False)
        Load and return the breast cancer wisconsin dataset (classificatio
n).

        The breast cancer dataset is a classic and very easy binary classi
fication
        dataset.

        =================   ==============
        Classes                          2
        Samples per class    212(M),357(B)
        Samples total                  569
        Dimensionality                  30
        Features            real, positive
        =================   ==============

        Read more in the :ref:`User Guide <breast_cancer_dataset>`.

        Parameters
        ----------
        return_X_y : boolean, default=False
            If True, returns ``(data, target)`` instead of a Bunch object.
            See below for more information about the `data` and `target` o
bject.

            .. versionadded:: 0.18

        Returns
        -------
        data : Bunch
            Dictionary-like object, the interesting attributes are:
            'data', the data to learn, 'target', the classification label
s,
            'target_names', the meaning of the labels, 'feature_names', th
e
            meaning of the features, and 'DESCR', the full description of
            the dataset, 'filename', the physical location of
            breast cancer csv dataset (added in version `0.20`).

        (data, target) : tuple if ``return_X_y`` is True

            .. versionadded:: 0.18
```

```
        The copy of UCI ML Breast Cancer Wisconsin (Diagnostic) dataset is
        downloaded from:
        https://goo.gl/U2Uwz2

        Examples
        --------
        Let's say you are interested in the samples 10, 50, and 85, and wa
nt to
        know their class name.

        >>> from sklearn.datasets import load_breast_cancer
        >>> data = load_breast_cancer()
        >>> data.target[[10, 50, 85]]
        array([0, 1, 0])
        >>> list(data.target_names)
        ['malignant', 'benign']

    load_diabetes(return_X_y=False)
        Load and return the diabetes dataset (regression).

        ==============      ==================
        Samples total       442
        Dimensionality      10
        Features            real, -.2 < x < .2
        Targets             integer 25 - 346
        ==============      ==================

        Read more in the :ref:`User Guide <diabetes_dataset>`.

        Parameters
        ----------
        return_X_y : boolean, default=False.
            If True, returns ``(data, target)`` instead of a Bunch object.
            See below for more information about the `data` and `target` o
bject.

            .. versionadded:: 0.18

        Returns
        -------
        data : Bunch
            Dictionary-like object, the interesting attributes are:
            'data', the data to learn, 'target', the regression target for
each
            sample, 'data_filename', the physical location
            of diabetes data csv dataset, and 'target_filename', the physi
cal
            location of diabetes targets csv datataset (added in version `
0.20`).

        (data, target) : tuple if ``return_X_y`` is True

            .. versionadded:: 0.18

    load_digits(n_class=10, return_X_y=False)
        Load and return the digits dataset (classification).

        Each datapoint is a 8x8 image of a digit.

        =================   ==============
```

```
        Classes                          10
        Samples per class             ~180
        Samples total                 1797
        Dimensionality                  64
        Features             integers 0-16
        =================   ==============

        Read more in the :ref:`User Guide <digits_dataset>`.

        Parameters
        ----------
        n_class : integer, between 0 and 10, optional (default=10)
            The number of classes to return.

        return_X_y : boolean, default=False.
            If True, returns ``(data, target)`` instead of a Bunch object.
            See below for more information about the `data` and `target` o
bject.

            .. versionadded:: 0.18

        Returns
        -------
        data : Bunch
            Dictionary-like object, the interesting attributes are:
            'data', the data to learn, 'images', the images corresponding
            to each sample, 'target', the classification labels for each
            sample, 'target_names', the meaning of the labels, and 'DESC
R',
            the full description of the dataset.

        (data, target) : tuple if ``return_X_y`` is True

            .. versionadded:: 0.18

        This is a copy of the test set of the UCI ML hand-written digits d
atasets
        http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Hand
written+Digits

        Examples
        --------
        To load the data and visualize the images::

            >>> from sklearn.datasets import load_digits
            >>> digits = load_digits()
            >>> print(digits.data.shape)
            (1797, 64)
            >>> import matplotlib.pyplot as plt #doctest: +SKIP
            >>> plt.gray() #doctest: +SKIP
            >>> plt.matshow(digits.images[0]) #doctest: +SKIP
            >>> plt.show() #doctest: +SKIP

    load_files(container_path, description=None, categories=None, load_con
tent=True, shuffle=True, encoding=None, decode_error='strict', random_stat
e=0)
        Load text files with categories as subfolder names.

        Individual samples are assumed to be files stored a two levels fol
der
        structure such as the following:
```

```
            container_folder/
                category_1_folder/
                    file_1.txt
                    file_2.txt
                    ...
                    file_42.txt
                category_2_folder/
                    file_43.txt
                    file_44.txt
                    ...

        The folder names are used as supervised signal label names. The in
dividual
        file names are not important.

        This function does not try to extract features into a numpy array
 or scipy
        sparse matrix. In addition, if load_content is false it does not t
ry to
        load the files in memory.

        To use text files in a scikit-learn classification or clustering a
lgorithm,
        you will need to use the `sklearn.feature_extraction.text` module
 to build
        a feature extraction transformer that suits your problem.

        If you set load_content=True, you should also specify the encoding
of the
        text using the 'encoding' parameter. For many modern text files,
 'utf-8'
        will be the correct encoding. If you leave encoding equal to None,
then the
        content will be made of bytes instead of Unicode, and you will not
be able
        to use most functions in `sklearn.feature_extraction.text`.

        Similar feature extractors should be built for other kind of unstr
uctured
        data input such as images, audio, video, ...

        Read more in the :ref:`User Guide <datasets>`.

        Parameters
        ----------
        container_path : string or unicode
            Path to the main folder holding one subfolder per category

        description : string or unicode, optional (default=None)
            A paragraph describing the characteristic of the dataset: its
 source,
            reference, etc.

        categories : A collection of strings or None, optional (default=No
ne)
            If None (default), load all the categories. If not None, list
 of
            category names to load (other categories ignored).

        load_content : boolean, optional (default=True)
```

```
            Whether to load or not the content of the different files. If
 true a
            'data' attribute containing the text information is present in
the data
            structure returned. If not, a filenames attribute gives the pa
th to the
            files.

        shuffle : bool, optional (default=True)
            Whether or not to shuffle the data: might be important for mod
els that
            make the assumption that the samples are independent and ident
ically
            distributed (i.i.d.), such as stochastic gradient descent.

        encoding : string or None (default is None)
            If None, do not try to decode the content of the files (e.g. f
or images
            or other non-text content). If not None, encoding to use to de
code text
            files to Unicode if load_content is True.

        decode_error : {'strict', 'ignore', 'replace'}, optional
            Instruction on what to do if a byte sequence is given to analy
ze that
            contains characters not of the given `encoding`. Passed as key
word
            argument 'errors' to bytes.decode.

        random_state : int, RandomState instance or None (default=0)
            Determines random number generation for dataset shuffling. Pas
s an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        data : Bunch
            Dictionary-like object, the interesting attributes are: either
            data, the raw text data to learn, or 'filenames', the files
            holding it, 'target', the classification labels (integer inde
x),
            'target_names', the meaning of the labels, and 'DESCR', the fu
ll
            description of the dataset.

    load_iris(return_X_y=False)
        Load and return the iris dataset (classification).

        The iris dataset is a classic and very easy multi-class classifica
tion
        dataset.

        =================   ==============
        Classes                          3
        Samples per class               50
        Samples total                  150
        Dimensionality                   4
        Features            real, positive
        =================   ==============
```

```
        Read more in the :ref:`User Guide <iris_dataset>`.

        Parameters
        ----------
        return_X_y : boolean, default=False.
            If True, returns ``(data, target)`` instead of a Bunch object.
See
            below for more information about the `data` and `target` objec
t.

            .. versionadded:: 0.18

        Returns
        -------
        data : Bunch
            Dictionary-like object, the interesting attributes are:
            'data', the data to learn, 'target', the classification label
s,
            'target_names', the meaning of the labels, 'feature_names', th
e
            meaning of the features, 'DESCR', the full description of
            the dataset, 'filename', the physical location of
            iris csv dataset (added in version `0.20`).

        (data, target) : tuple if ``return_X_y`` is True

            .. versionadded:: 0.18

        Notes
        -----
            .. versionchanged:: 0.20
                Fixed two wrong data points according to Fisher's paper.
                The new version is the same as in R, but not as in the UCI
                Machine Learning Repository.

        Examples
        --------
        Let's say you are interested in the samples 10, 25, and 50, and wa
nt to
        know their class name.

        >>> from sklearn.datasets import load_iris
        >>> data = load_iris()
        >>> data.target[[10, 25, 50]]
        array([0, 0, 1])
        >>> list(data.target_names)
        ['setosa', 'versicolor', 'virginica']

    load_linnerud(return_X_y=False)
        Load and return the linnerud dataset (multivariate regression).

        ==============    ============================
        Samples total     20
        Dimensionality    3 (for both data and target)
        Features          integer
        Targets           integer
        ==============    ============================

        Read more in the :ref:`User Guide <linnerrud_dataset>`.

        Parameters
```

```
        ----------
        return_X_y : boolean, default=False.
            If True, returns ``(data, target)`` instead of a Bunch object.
            See below for more information about the `data` and `target` o
bject.

            .. versionadded:: 0.18

        Returns
        -------
        data : Bunch
            Dictionary-like object, the interesting attributes are: 'data'
and
            'targets', the two multivariate datasets, with 'data' correspo
nding to
            the exercise and 'targets' corresponding to the physiological
            measurements, as well as 'feature_names' and 'target_names'.
            In addition, you will also have access to 'data_filename',
            the physical location of linnerud data csv dataset, and
            'target_filename', the physical location of
            linnerud targets csv datataset (added in version `0.20`).

        (data, target) : tuple if ``return_X_y`` is True

            .. versionadded:: 0.18

    load_mlcomp(name_or_id, set_='raw', mlcomp_root=None, **kwargs)
        DEPRECATED: since the http://mlcomp.org/ website will shut down in
March 2017, the load_mlcomp function was deprecated in version 0.19 and wi
ll be removed in 0.21.

        Load a datasets as downloaded from http://mlcomp.org

            Read more in the :ref:`User Guide <datasets>`.

            Parameters
            ----------

            name_or_id : int or str
                The integer id or the string name metadata of the MLComp
                dataset to load

            set\_ : str, default='raw'
                Select the portion to load: 'train', 'test' or 'raw'

            mlcomp_root : str, optional
                The filesystem path to the root folder where MLComp datase
ts
                are stored, if mlcomp_root is None, the MLCOMP_DATASETS_HO
ME
                environment variable is looked up instead.

            **kwargs : domain specific kwargs to be passed to the dataset
 loader.

            Returns
            -------

            data : Bunch
                Dictionary-like object, the interesting attributes are:
                'filenames', the files holding the raw to learn, 'target',
```

```
   the
                classification labels (integer index), 'target_names',
                the meaning of the labels, and 'DESCR', the full descripti
   on of the
                dataset.

            Note on the lookup process: depending on the type of name_or_i
   d,
            will choose between integer id lookup or metadata name lookup
    by
            looking at the unzipped archives and metadata file.

            TODO: implement zip dataset loading too

        load_sample_image(image_name)
            Load the numpy array of a single sample image

            Read more in the :ref:`User Guide <sample_images>`.

            Parameters
            -----------
            image_name : {`china.jpg`, `flower.jpg`}
                The name of the sample image loaded

            Returns
            -------
            img : 3D array
                The image as a numpy array: height x width x color

            Examples
            ---------

            >>> from sklearn.datasets import load_sample_image
            >>> china = load_sample_image('china.jpg')   # doctest: +SKIP
            >>> china.dtype                              # doctest: +SKIP
            dtype('uint8')
            >>> china.shape                              # doctest: +SKIP
            (427, 640, 3)
            >>> flower = load_sample_image('flower.jpg') # doctest: +SKIP
            >>> flower.dtype                             # doctest: +SKIP
            dtype('uint8')
            >>> flower.shape                             # doctest: +SKIP
            (427, 640, 3)

        load_sample_images()
            Load sample images for image manipulation.

            Loads both, ``china`` and ``flower``.

            Read more in the :ref:`User Guide <sample_images>`.

            Returns
            -------
            data : Bunch
                Dictionary-like object with the following attributes : 'image
   s', the
                two sample images, 'filenames', the file names for the images,
   and
                'DESCR' the full description of the dataset.

            Examples
```

```
        --------
        To load the data and visualize the images:

        >>> from sklearn.datasets import load_sample_images
        >>> dataset = load_sample_images()     #doctest: +SKIP
        >>> len(dataset.images)                #doctest: +SKIP
        2
        >>> first_img_data = dataset.images[0] #doctest: +SKIP
        >>> first_img_data.shape               #doctest: +SKIP
        (427, 640, 3)
        >>> first_img_data.dtype               #doctest: +SKIP
        dtype('uint8')

    load_svmlight_file(f, n_features=None, dtype=<class 'numpy.float64'>,
 multilabel=False, zero_based='auto', query_id=False, offset=0, length=-1)
        Load datasets in the svmlight / libsvm format into sparse CSR matr
ix

        This format is a text-based format, with one sample per line. It d
oes
        not store zero valued features hence is suitable for sparse datase
t.

        The first element of each line can be used to store a target varia
ble
        to predict.

        This format is used as the default format for both svmlight and th
e
        libsvm command line programs.

        Parsing a text based source can be expensive. When working on
        repeatedly on the same dataset, it is recommended to wrap this
        loader with joblib.Memory.cache to store a memmapped backup of the
        CSR results of the first call and benefit from the near instantane
ous
        loading of memmapped structures for the subsequent calls.

        In case the file contains a pairwise preference constraint (known
        as "qid" in the svmlight format) these are ignored unless the
        query_id parameter is set to True. These pairwise preference
        constraints can be used to constraint the combination of samples
        when using pairwise loss functions (as is the case in some
        learning to rank problems) so that only pairs with the same
        query_id value are considered.

        This implementation is written in Cython and is reasonably fast.
        However, a faster API-compatible loader is also available at:

          https://github.com/mblondel/svmlight-loader

        Parameters
        ----------
        f : {str, file-like, int}
            (Path to) a file to load. If a path ends in ".gz" or ".bz2", i
t will
            be uncompressed on the fly. If an integer is passed, it is ass
umed to
            be a file descriptor. A file-like or file descriptor will not
 be closed
            by this function. A file-like object must be opened in binary
```

```
 mode.

        n_features : int or None
            The number of features to use. If None, it will be inferred. T
his
            argument is useful to load several files that are subsets of a
            bigger sliced dataset: each subset might not have examples of
            every feature, hence the inferred shape might vary from one
            slice to another.
            n_features is only required if ``offset`` or ``length`` are pa
ssed a
            non-default value.

        dtype : numpy data type, default np.float64
            Data type of dataset to be loaded. This will be the data type
 of the
            output numpy arrays ``X`` and ``y``.

        multilabel : boolean, optional, default False
            Samples may have several labels each (see
            https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multil
abel.html)

        zero_based : boolean or "auto", optional, default "auto"
            Whether column indices in f are zero-based (True) or one-based
            (False). If column indices are one-based, they are transformed
to
            zero-based to match Python/NumPy conventions.
            If set to "auto", a heuristic check is applied to determine th
is from
            the file contents. Both kinds of files occur "in the wild", bu
t they
            are unfortunately not self-identifying. Using "auto" or True s
hould
            always be safe when no ``offset`` or ``length`` is passed.
            If ``offset`` or ``length`` are passed, the "auto" mode falls
 back
            to ``zero_based=True`` to avoid having the heuristic check yie
ld
            inconsistent results on different segments of the file.

        query_id : boolean, default False
            If True, will return the query_id array for each file.

        offset : integer, optional, default 0
            Ignore the offset first bytes by seeking forward, then
            discarding the following bytes up until the next new line
            character.

        length : integer, optional, default -1
            If strictly positive, stop reading any new line of data once t
he
            position in the file has reached the (offset + length) bytes t
hreshold.

        Returns
        -------
        X : scipy.sparse matrix of shape (n_samples, n_features)

        y : ndarray of shape (n_samples,), or, in the multilabel a list of
            tuples of length n_samples.
```

```
        query_id : array of shape (n_samples,)
           query_id for each sample. Only returned when query_id is set to
           True.

        See also
        --------
        load_svmlight_files: similar function for loading multiple files i
n this
        format, enforcing the same number of features/columns on all of th
em.

        Examples
        --------
        To use joblib.Memory to cache the svmlight file::

            from joblib import Memory
            from sklearn.datasets import load_svmlight_file
            mem = Memory("./mycache")

            @mem.cache
            def get_data():
                data = load_svmlight_file("mysvmlightfile")
                return data[0], data[1]

            X, y = get_data()

    load_svmlight_files(files, n_features=None, dtype=<class 'numpy.float6
4'>, multilabel=False, zero_based='auto', query_id=False, offset=0, length
=-1)
        Load dataset from multiple files in SVMlight format

        This function is equivalent to mapping load_svmlight_file over a l
ist of
        files, except that the results are concatenated into a single, fla
t list
        and the samples vectors are constrained to all have the same numbe
r of
        features.

        In case the file contains a pairwise preference constraint (known
        as "qid" in the svmlight format) these are ignored unless the
        query_id parameter is set to True. These pairwise preference
        constraints can be used to constraint the combination of samples
        when using pairwise loss functions (as is the case in some
        learning to rank problems) so that only pairs with the same
        query_id value are considered.

        Parameters
        ----------
        files : iterable over {str, file-like, int}
            (Paths of) files to load. If a path ends in ".gz" or ".bz2", i
t will
            be uncompressed on the fly. If an integer is passed, it is ass
umed to
            be a file descriptor. File-likes and file descriptors will not
be
            closed by this function. File-like objects must be opened in b
inary
            mode.
```

```
        n_features : int or None
            The number of features to use. If None, it will be inferred fr
om the
            maximum column index occurring in any of the files.

            This can be set to a higher value than the actual number of fe
atures
            in any of the input files, but setting it to a lower value wil
l cause
            an exception to be raised.

        dtype : numpy data type, default np.float64
            Data type of dataset to be loaded. This will be the data type
 of the
            output numpy arrays ``X`` and ``y``.

        multilabel : boolean, optional
            Samples may have several labels each (see
            https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multil
abel.html)

        zero_based : boolean or "auto", optional
            Whether column indices in f are zero-based (True) or one-based
            (False). If column indices are one-based, they are transformed
to
            zero-based to match Python/NumPy conventions.
            If set to "auto", a heuristic check is applied to determine th
is from
            the file contents. Both kinds of files occur "in the wild", bu
t they
            are unfortunately not self-identifying. Using "auto" or True s
hould
            always be safe when no offset or length is passed.
            If offset or length are passed, the "auto" mode falls back
            to zero_based=True to avoid having the heuristic check yield
            inconsistent results on different segments of the file.

        query_id : boolean, defaults to False
            If True, will return the query_id array for each file.

        offset : integer, optional, default 0
            Ignore the offset first bytes by seeking forward, then
            discarding the following bytes up until the next new line
            character.

        length : integer, optional, default -1
            If strictly positive, stop reading any new line of data once t
he
            position in the file has reached the (offset + length) bytes t
hreshold.

        Returns
        -------
        [X1, y1, ..., Xn, yn]
        where each (Xi, yi) pair is the result from load_svmlight_file(fil
es[i]).

        If query_id is set to True, this will return instead [X1, y1, q1,
        ..., Xn, yn, qn] where (Xi, yi, qi) is the result from
        load_svmlight_file(files[i])
```

```
        Notes
        -----
        When fitting a model to a matrix X_train and evaluating it against
a
        matrix X_test, it is essential that X_train and X_test have the sa
me
        number of features (X_train.shape[1] == X_test.shape[1]). This may
not
        be the case if you load the files individually with load_svmlight_
file.

        See also
        --------
        load_svmlight_file

    load_wine(return_X_y=False)
        Load and return the wine dataset (classification).

        .. versionadded:: 0.18

        The wine dataset is a classic and very easy multi-class classifica
tion
        dataset.

        =================   ==============
        Classes                          3
        Samples per class        [59,71,48]
        Samples total                   178
        Dimensionality                   13
        Features            real, positive
        =================   ==============

        Read more in the :ref:`User Guide <wine_dataset>`.

        Parameters
        ----------
        return_X_y : boolean, default=False.
            If True, returns ``(data, target)`` instead of a Bunch object.
            See below for more information about the `data` and `target` o
bject.

        Returns
        -------
        data : Bunch
            Dictionary-like object, the interesting attributes are: 'dat
a', the
            data to learn, 'target', the classification labels, 'target_na
mes', the
            meaning of the labels, 'feature_names', the meaning of the fea
tures,
            and 'DESCR', the full description of the dataset.

        (data, target) : tuple if ``return_X_y`` is True

        The copy of UCI ML Wine Data Set dataset is downloaded and modifie
d to fit
        standard format from:
        https://archive.ics.uci.edu/ml/machine-learning-databases/wine/win
e.data

        Examples
```

```
        --------
        Let's say you are interested in the samples 10, 80, and 140, and w
ant to
        know their class name.

        >>> from sklearn.datasets import load_wine
        >>> data = load_wine()
        >>> data.target[[10, 80, 140]]
        array([0, 1, 2])
        >>> list(data.target_names)
        ['class_0', 'class_1', 'class_2']

    make_biclusters(shape, n_clusters, noise=0.0, minval=10, maxval=100, s
huffle=True, random_state=None)
        Generate an array with constant block diagonal structure for
        biclustering.

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        shape : iterable (n_rows, n_cols)
            The shape of the result.

        n_clusters : integer
            The number of biclusters.

        noise : float, optional (default=0.0)
            The standard deviation of the gaussian noise.

        minval : int, optional (default=10)
            Minimum value of a bicluster.

        maxval : int, optional (default=100)
            Maximum value of a bicluster.

        shuffle : boolean, optional (default=True)
            Shuffle the samples.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape `shape`
            The generated array.

        rows : array of shape (n_clusters, X.shape[0],)
            The indicators for cluster membership of each row.

        cols : array of shape (n_clusters, X.shape[1],)
            The indicators for cluster membership of each column.

        References
        ----------

        .. [1] Dhillon, I. S. (2001, August). Co-clustering documents and
            words using bipartite spectral graph partitioning. In Proceedi
```

```
ngs
            of the seventh ACM SIGKDD international conference on Knowledg
e
            discovery and data mining (pp. 269-274). ACM.

        See also
        --------
        make_checkerboard

    make_blobs(n_samples=100, n_features=2, centers=None, cluster_std=1.0,
center_box=(-10.0, 10.0), shuffle=True, random_state=None)
        Generate isotropic Gaussian blobs for clustering.

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        n_samples : int or array-like, optional (default=100)
            If int, it is the total number of points equally divided among
            clusters.
            If array-like, each element of the sequence indicates
            the number of samples per cluster.

        n_features : int, optional (default=2)
            The number of features for each sample.

        centers : int or array of shape [n_centers, n_features], optional
            (default=None)
            The number of centers to generate, or the fixed center locatio
ns.
            If n_samples is an int and centers is None, 3 centers are gene
rated.
            If n_samples is array-like, centers must be
            either None or an array of length equal to the length of n_sam
ples.

        cluster_std : float or sequence of floats, optional (default=1.0)
            The standard deviation of the clusters.

        center_box : pair of floats (min, max), optional (default=(-10.0,
 10.0))
            The bounding box for each cluster center when centers are
            generated at random.

        shuffle : boolean, optional (default=True)
            Shuffle the samples.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_samples, n_features]
            The generated samples.

        y : array of shape [n_samples]
            The integer labels for cluster membership of each sample.
```

```
        Examples
        --------
        >>> from sklearn.datasets.samples_generator import make_blobs
        >>> X, y = make_blobs(n_samples=10, centers=3, n_features=2,
        ...                   random_state=0)
        >>> print(X.shape)
        (10, 2)
        >>> y
        array([0, 0, 1, 0, 2, 2, 2, 1, 1, 0])
        >>> X, y = make_blobs(n_samples=[3, 3, 4], centers=None, n_feature
s=2,
        ...                   random_state=0)
        >>> print(X.shape)
        (10, 2)
        >>> y
        array([0, 1, 2, 0, 2, 2, 2, 1, 1, 0])

        See also
        --------
        make_classification: a more intricate variant

    make_checkerboard(shape, n_clusters, noise=0.0, minval=10, maxval=100,
shuffle=True, random_state=None)
        Generate an array with block checkerboard structure for
        biclustering.

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        shape : iterable (n_rows, n_cols)
            The shape of the result.

        n_clusters : integer or iterable (n_row_clusters, n_column_cluster
s)
            The number of row and column clusters.

        noise : float, optional (default=0.0)
            The standard deviation of the gaussian noise.

        minval : int, optional (default=10)
            Minimum value of a bicluster.

        maxval : int, optional (default=100)
            Maximum value of a bicluster.

        shuffle : boolean, optional (default=True)
            Shuffle the samples.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape `shape`
            The generated array.

        rows : array of shape (n_clusters, X.shape[0],)
```

```
            The indicators for cluster membership of each row.

        cols : array of shape (n_clusters, X.shape[1],)
            The indicators for cluster membership of each column.


        References
        ----------

        .. [1] Kluger, Y., Basri, R., Chang, J. T., & Gerstein, M. (2003).
            Spectral biclustering of microarray data: coclustering genes
            and conditions. Genome research, 13(4), 703-716.

        See also
        --------
        make_biclusters

    make_circles(n_samples=100, shuffle=True, noise=None, random_state=Non
e, factor=0.8)
        Make a large circle containing a smaller circle in 2d.

        A simple toy dataset to visualize clustering and classification
        algorithms.

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        n_samples : int, optional (default=100)
            The total number of points generated. If odd, the inner circle
will
            have one point more than the outer circle.

        shuffle : bool, optional (default=True)
            Whether to shuffle the samples.

        noise : double or None (default=None)
            Standard deviation of Gaussian noise added to the data.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset shuffling and
 noise.
            Pass an int for reproducible output across multiple function c
alls.
            See :term:`Glossary <random_state>`.

        factor : 0 < double < 1 (default=.8)
            Scale factor between inner and outer circle.

        Returns
        -------
        X : array of shape [n_samples, 2]
            The generated samples.

        y : array of shape [n_samples]
            The integer labels (0 or 1) for class membership of each sampl
e.

    make_classification(n_samples=100, n_features=20, n_informative=2, n_r
edundant=2, n_repeated=0, n_classes=2, n_clusters_per_class=2, weights=Non
e, flip_y=0.01, class_sep=1.0, hypercube=True, shift=0.0, scale=1.0, shuff
```

```
le=True, random_state=None)
        Generate a random n-class classification problem.

        This initially creates clusters of points normally distributed (st
d=1)
        about vertices of an ``n_informative``-dimensional hypercube with
 sides of
        length ``2*class_sep`` and assigns an equal number of clusters to
 each
        class. It introduces interdependence between these features and ad
ds
        various types of further noise to the data.

        Without shuffling, ``X`` horizontally stacks features in the follo
wing
        order: the primary ``n_informative`` features, followed by ``n_red
undant``
        linear combinations of the informative features, followed by ``n_r
epeated``
        duplicates, drawn randomly with replacement from the informative a
nd
        redundant features. The remaining features are filled with random
 noise.
        Thus, without shuffling, all useful features are contained in the
 columns
        ``X[:, :n_informative + n_redundant + n_repeated]``.

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        n_samples : int, optional (default=100)
            The number of samples.

        n_features : int, optional (default=20)
            The total number of features. These comprise ``n_informative``
            informative features, ``n_redundant`` redundant features,
            ``n_repeated`` duplicated features and
            ``n_features-n_informative-n_redundant-n_repeated`` useless fe
atures
            drawn at random.

        n_informative : int, optional (default=2)
            The number of informative features. Each class is composed of
 a number
            of gaussian clusters each located around the vertices of a hyp
ercube
            in a subspace of dimension ``n_informative``. For each cluste
r,
            informative features are drawn independently from  N(0, 1) and
then
            randomly linearly combined within each cluster in order to add
            covariance. The clusters are then placed on the vertices of th
e
            hypercube.

        n_redundant : int, optional (default=2)
            The number of redundant features. These features are generated
as
            random linear combinations of the informative features.
```

```
        n_repeated : int, optional (default=0)
            The number of duplicated features, drawn randomly from the inf
ormative
            and the redundant features.

        n_classes : int, optional (default=2)
            The number of classes (or labels) of the classification proble
m.

        n_clusters_per_class : int, optional (default=2)
            The number of clusters per class.

        weights : list of floats or None (default=None)
            The proportions of samples assigned to each class. If None, th
en
            classes are balanced. Note that if ``len(weights) == n_classes
- 1``,
            then the last class weight is automatically inferred.
            More than ``n_samples`` samples may be returned if the sum of
            ``weights`` exceeds 1.

        flip_y : float, optional (default=0.01)
            The fraction of samples whose class are randomly exchanged. La
rger
            values introduce noise in the labels and make the classificati
on
            task harder.

        class_sep : float, optional (default=1.0)
            The factor multiplying the hypercube size.  Larger values spre
ad
            out the clusters/classes and make the classification task easi
er.

        hypercube : boolean, optional (default=True)
            If True, the clusters are put on the vertices of a hypercube.
 If
            False, the clusters are put on the vertices of a random polyto
pe.

        shift : float, array of shape [n_features] or None, optional (defa
ult=0.0)
            Shift features by the specified value. If None, then features
            are shifted by a random value drawn in [-class_sep, class_se
p].

        scale : float, array of shape [n_features] or None, optional (defa
ult=1.0)
            Multiply features by the specified value. If None, then featur
es
            are scaled by a random value drawn in [1, 100]. Note that scal
ing
            happens after shifting.

        shuffle : boolean, optional (default=True)
            Shuffle the samples and the features.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
```

```
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_samples, n_features]
            The generated samples.

        y : array of shape [n_samples]
            The integer labels for class membership of each sample.

        Notes
        -----
        The algorithm is adapted from Guyon [1] and was designed to genera
te
        the "Madelon" dataset.

        References
        ----------
        .. [1] I. Guyon, "Design of experiments for the NIPS 2003 variable
               selection benchmark", 2003.

        See also
        --------
        make_blobs: simplified variant
        make_multilabel_classification: unrelated generator for multilabel
tasks

    make_friedman1(n_samples=100, n_features=10, noise=0.0, random_state=N
one)
        Generate the "Friedman #1" regression problem

        This dataset is described in Friedman [1] and Breiman [2].

        Inputs `X` are independent features uniformly distributed on the i
nterval
        [0, 1]. The output `y` is created according to the formula::

            y(X) = 10 * sin(pi * X[:, 0] * X[:, 1]) + 20 * (X[:, 2] - 0.5)
** 2 + 10 * X[:, 3] + 5 * X[:, 4] + noise * N(0, 1).

        Out of the `n_features` features, only 5 are actually used to comp
ute
        `y`. The remaining features are independent of `y`.

        The number of features has to be >= 5.

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        n_samples : int, optional (default=100)
            The number of samples.

        n_features : int, optional (default=10)
            The number of features. Should be at least 5.

        noise : float, optional (default=0.0)
            The standard deviation of the gaussian noise applied to the ou
tput.

        random_state : int, RandomState instance or None (default)
```

```
            Determines random number generation for dataset noise. Pass an
int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_samples, n_features]
            The input samples.

        y : array of shape [n_samples]
            The output values.

        References
        ----------
        .. [1] J. Friedman, "Multivariate adaptive regression splines", Th
e Annals
               of Statistics 19 (1), pages 1-67, 1991.

        .. [2] L. Breiman, "Bagging predictors", Machine Learning 24,
               pages 123-140, 1996.

    make_friedman2(n_samples=100, noise=0.0, random_state=None)
        Generate the "Friedman #2" regression problem

        This dataset is described in Friedman [1] and Breiman [2].

        Inputs `X` are 4 independent features uniformly distributed on the
        intervals::

            0 <= X[:, 0] <= 100,
            40 * pi <= X[:, 1] <= 560 * pi,
            0 <= X[:, 2] <= 1,
            1 <= X[:, 3] <= 11.

        The output `y` is created according to the formula::

            y(X) = (X[:, 0] ** 2 + (X[:, 1] * X[:, 2]  - 1 / (X[:, 1] * X
[:, 3])) ** 2) ** 0.5 + noise * N(0, 1).

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        n_samples : int, optional (default=100)
            The number of samples.

        noise : float, optional (default=0.0)
            The standard deviation of the gaussian noise applied to the ou
tput.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset noise. Pass an
int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_samples, 4]
            The input samples.
```

```
        y : array of shape [n_samples]
            The output values.

        References
        ----------
        .. [1] J. Friedman, "Multivariate adaptive regression splines", Th
e Annals
               of Statistics 19 (1), pages 1-67, 1991.

        .. [2] L. Breiman, "Bagging predictors", Machine Learning 24,
               pages 123-140, 1996.

    make_friedman3(n_samples=100, noise=0.0, random_state=None)
        Generate the "Friedman #3" regression problem

        This dataset is described in Friedman [1] and Breiman [2].

        Inputs `X` are 4 independent features uniformly distributed on the
        intervals::

            0 <= X[:, 0] <= 100,
            40 * pi <= X[:, 1] <= 560 * pi,
            0 <= X[:, 2] <= 1,
            1 <= X[:, 3] <= 11.

        The output `y` is created according to the formula::

            y(X) = arctan((X[:, 1] * X[:, 2] - 1 / (X[:, 1] * X[:, 3])) /
 X[:, 0]) + noise * N(0, 1).

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        n_samples : int, optional (default=100)
            The number of samples.

        noise : float, optional (default=0.0)
            The standard deviation of the gaussian noise applied to the ou
tput.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset noise. Pass an
int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_samples, 4]
            The input samples.

        y : array of shape [n_samples]
            The output values.

        References
        ----------
        .. [1] J. Friedman, "Multivariate adaptive regression splines", Th
e Annals
               of Statistics 19 (1), pages 1-67, 1991.
```

```
        .. [2] L. Breiman, "Bagging predictors", Machine Learning 24,
               pages 123-140, 1996.

    make_gaussian_quantiles(mean=None, cov=1.0, n_samples=100, n_features=
2, n_classes=3, shuffle=True, random_state=None)
        Generate isotropic Gaussian and label samples by quantile

        This classification dataset is constructed by taking a multi-dimen
sional
        standard normal distribution and defining classes separated by nes
ted
        concentric multi-dimensional spheres such that roughly equal numbe
rs of
        samples are in each class (quantiles of the :math:`\chi^2` distrib
ution).

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        mean : array of shape [n_features], optional (default=None)
            The mean of the multi-dimensional normal distribution.
            If None then use the origin (0, 0, ...).

        cov : float, optional (default=1.)
            The covariance matrix will be this value times the unit matri
x. This
            dataset only produces symmetric normal distributions.

        n_samples : int, optional (default=100)
            The total number of points equally divided among classes.

        n_features : int, optional (default=2)
            The number of features for each sample.

        n_classes : int, optional (default=3)
            The number of classes

        shuffle : boolean, optional (default=True)
            Shuffle the samples.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_samples, n_features]
            The generated samples.

        y : array of shape [n_samples]
            The integer labels for quantile membership of each sample.

        Notes
        -----
        The dataset is from Zhu et al [1].

        References
```

```
        ----------
        .. [1] J. Zhu, H. Zou, S. Rosset, T. Hastie, "Multi-class AdaBoos
t", 2009.

    make_hastie_10_2(n_samples=12000, random_state=None)
        Generates data for binary classification used in
        Hastie et al. 2009, Example 10.2.

        The ten features are standard independent Gaussian and
        the target ``y`` is defined by::

          y[i] = 1 if np.sum(X[i] ** 2) > 9.34 else -1

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        n_samples : int, optional (default=12000)
            The number of samples.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_samples, 10]
            The input samples.

        y : array of shape [n_samples]
            The output values.

        References
        ----------
        .. [1] T. Hastie, R. Tibshirani and J. Friedman, "Elements of Stat
istical
               Learning Ed. 2", Springer, 2009.

        See also
        --------
        make_gaussian_quantiles: a generalization of this dataset approach

    make_low_rank_matrix(n_samples=100, n_features=100, effective_rank=10,
tail_strength=0.5, random_state=None)
        Generate a mostly low rank matrix with bell-shaped singular values

        Most of the variance can be explained by a bell-shaped curve of wi
dth
        effective_rank: the low rank part of the singular values profile i
s::

            (1 - tail_strength) * exp(-1.0 * (i / effective_rank) ** 2)

        The remaining singular values' tail is fat, decreasing as::

            tail_strength * exp(-0.1 * i / effective_rank).

        The low rank part of the profile can be considered the structured
        signal part of the data while the tail can be considered the noisy
```

```
        part of the data that cannot be summarized by a low number of line
ar
        components (singular vectors).

        This kind of singular profiles is often seen in practice, for inst
ance:
         - gray level pictures of faces
         - TF-IDF vectors of text documents crawled from the web

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        n_samples : int, optional (default=100)
            The number of samples.

        n_features : int, optional (default=100)
            The number of features.

        effective_rank : int, optional (default=10)
            The approximate number of singular vectors required to explain
most of
            the data by linear combinations.

        tail_strength : float between 0.0 and 1.0, optional (default=0.5)
            The relative importance of the fat noisy tail of the singular
 values
            profile.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_samples, n_features]
            The matrix.

    make_moons(n_samples=100, shuffle=True, noise=None, random_state=None)
        Make two interleaving half circles

        A simple toy dataset to visualize clustering and classification
        algorithms. Read more in the :ref:`User Guide <sample_generators>
`.

        Parameters
        ----------
        n_samples : int, optional (default=100)
            The total number of points generated.

        shuffle : bool, optional (default=True)
            Whether to shuffle the samples.

        noise : double or None (default=None)
            Standard deviation of Gaussian noise added to the data.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset shuffling and
 noise.
```

```
            Pass an int for reproducible output across multiple function c
alls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_samples, 2]
            The generated samples.

        y : array of shape [n_samples]
            The integer labels (0 or 1) for class membership of each sampl
e.

    make_multilabel_classification(n_samples=100, n_features=20, n_classes
=5, n_labels=2, length=50, allow_unlabeled=True, sparse=False, return_indi
cator='dense', return_distributions=False, random_state=None)
        Generate a random multilabel classification problem.

        For each sample, the generative process is:
            - pick the number of labels: n ~ Poisson(n_labels)
            - n times, choose a class c: c ~ Multinomial(theta)
            - pick the document length: k ~ Poisson(length)
            - k times, choose a word: w ~ Multinomial(theta_c)

        In the above process, rejection sampling is used to make sure that
        n is never zero or more than `n_classes`, and that the document le
ngth
        is never zero. Likewise, we reject classes which have already been
chosen.

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        n_samples : int, optional (default=100)
            The number of samples.

        n_features : int, optional (default=20)
            The total number of features.

        n_classes : int, optional (default=5)
            The number of classes of the classification problem.

        n_labels : int, optional (default=2)
            The average number of labels per instance. More precisely, the
number
            of labels per sample is drawn from a Poisson distribution with
            ``n_labels`` as its expected value, but samples are bounded (u
sing
            rejection sampling) by ``n_classes``, and must be nonzero if
            ``allow_unlabeled`` is False.

        length : int, optional (default=50)
            The sum of the features (number of words if documents) is draw
n from
            a Poisson distribution with this expected value.

        allow_unlabeled : bool, optional (default=True)
            If ``True``, some instances might not belong to any class.

        sparse : bool, optional (default=False)
```

```
            If ``True``, return a sparse feature matrix

            .. versionadded:: 0.17
               parameter to allow *sparse* output.

        return_indicator : 'dense' (default) | 'sparse' | False
            If ``dense`` return ``Y`` in the dense binary indicator forma
t. If
            ``'sparse'`` return ``Y`` in the sparse binary indicator forma
t.
            ``False`` returns a list of lists of labels.

        return_distributions : bool, optional (default=False)
            If ``True``, return the prior class probability and conditiona
l
            probabilities of features given classes, from which the data w
as
            drawn.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_samples, n_features]
            The generated samples.

        Y : array or sparse CSR matrix of shape [n_samples, n_classes]
            The label sets.

        p_c : array, shape [n_classes]
            The probability of each class being drawn. Only returned if
            ``return_distributions=True``.

        p_w_c : array, shape [n_features, n_classes]
            The probability of each feature being drawn given each class.
            Only returned if ``return_distributions=True``.

    make_regression(n_samples=100, n_features=100, n_informative=10, n_tar
gets=1, bias=0.0, effective_rank=None, tail_strength=0.5, noise=0.0, shuff
le=True, coef=False, random_state=None)
        Generate a random regression problem.

        The input set can either be well conditioned (by default) or have
 a low
        rank-fat tail singular profile. See :func:`make_low_rank_matrix` f
or
        more details.

        The output is generated by applying a (potentially biased) random
 linear
        regression model with `n_informative` nonzero regressors to the pr
eviously
        generated input and some gaussian centered noise with some adjusta
ble
        scale.

        Read more in the :ref:`User Guide <sample_generators>`.
```

```
        Parameters
        ----------
        n_samples : int, optional (default=100)
            The number of samples.

        n_features : int, optional (default=100)
            The number of features.

        n_informative : int, optional (default=10)
            The number of informative features, i.e., the number of featur
es used
            to build the linear model used to generate the output.

        n_targets : int, optional (default=1)
            The number of regression targets, i.e., the dimension of the y
output
            vector associated with a sample. By default, the output is a s
calar.

        bias : float, optional (default=0.0)
            The bias term in the underlying linear model.

        effective_rank : int or None, optional (default=None)
            if not None:
                The approximate number of singular vectors required to exp
lain most
                of the input data by linear combinations. Using this kind
 of
                singular spectrum in the input allows the generator to rep
roduce
                the correlations often observed in practice.
            if None:
                The input set is well conditioned, centered and gaussian w
ith
                unit variance.

        tail_strength : float between 0.0 and 1.0, optional (default=0.5)
            The relative importance of the fat noisy tail of the singular
 values
            profile if `effective_rank` is not None.

        noise : float, optional (default=0.0)
            The standard deviation of the gaussian noise applied to the ou
tput.

        shuffle : boolean, optional (default=True)
            Shuffle the samples and the features.

        coef : boolean, optional (default=False)
            If True, the coefficients of the underlying linear model are r
eturned.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
```

```
        X : array of shape [n_samples, n_features]
            The input samples.

        y : array of shape [n_samples] or [n_samples, n_targets]
            The output values.

        coef : array of shape [n_features] or [n_features, n_targets], opt
ional
            The coefficient of the underlying linear model. It is returned
only if
            coef is True.

    make_s_curve(n_samples=100, noise=0.0, random_state=None)
        Generate an S curve dataset.

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        n_samples : int, optional (default=100)
            The number of sample points on the S curve.

        noise : float, optional (default=0.0)
            The standard deviation of the gaussian noise.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_samples, 3]
            The points.

        t : array of shape [n_samples]
            The univariate position of the sample according to the main di
mension
            of the points in the manifold.

    make_sparse_coded_signal(n_samples, n_components, n_features, n_nonzer
o_coefs, random_state=None)
        Generate a signal as a sparse combination of dictionary elements.

        Returns a matrix Y = DX, such as D is (n_features, n_components),
        X is (n_components, n_samples) and each column of X has exactly
        n_nonzero_coefs non-zero elements.

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        n_samples : int
            number of samples to generate

        n_components :  int,
            number of components in the dictionary

        n_features : int
            number of features of the dataset to generate
```

```
        n_nonzero_coefs : int
            number of active (non-zero) coefficients in each sample

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        data : array of shape [n_features, n_samples]
            The encoded signal (Y).

        dictionary : array of shape [n_features, n_components]
            The dictionary with normalized components (D).

        code : array of shape [n_components, n_samples]
            The sparse code such that each column of this matrix has exact
ly
            n_nonzero_coefs non-zero items (X).

    make_sparse_spd_matrix(dim=1, alpha=0.95, norm_diag=False, smallest_co
ef=0.1, largest_coef=0.9, random_state=None)
        Generate a sparse symmetric definite positive matrix.

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        dim : integer, optional (default=1)
            The size of the random matrix to generate.

        alpha : float between 0 and 1, optional (default=0.95)
            The probability that a coefficient is zero (see notes). Larger
values
            enforce more sparsity.

        norm_diag : boolean, optional (default=False)
            Whether to normalize the output matrix to make the leading dia
gonal
            elements all 1

        smallest_coef : float between 0 and 1, optional (default=0.1)
            The value of the smallest coefficient.

        largest_coef : float between 0 and 1, optional (default=0.9)
            The value of the largest coefficient.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        prec : sparse matrix of shape (dim, dim)
            The generated matrix.
```

```
        Notes
        -----
        The sparsity is actually imposed on the cholesky factor of the mat
rix.
        Thus alpha does not translate directly into the filling fraction o
f
        the matrix itself.

        See also
        --------
        make_spd_matrix

    make_sparse_uncorrelated(n_samples=100, n_features=10, random_state=No
ne)
        Generate a random regression problem with sparse uncorrelated desi
gn

        This dataset is described in Celeux et al [1]. as::

            X ~ N(0, 1)
            y(X) = X[:, 0] + 2 * X[:, 1] - 2 * X[:, 2] - 1.5 * X[:, 3]

        Only the first 4 features are informative. The remaining features
 are
        useless.

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        n_samples : int, optional (default=100)
            The number of samples.

        n_features : int, optional (default=10)
            The number of features.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_samples, n_features]
            The input samples.

        y : array of shape [n_samples]
            The output values.

        References
        ----------
        .. [1] G. Celeux, M. El Anbari, J.-M. Marin, C. P. Robert,
               "Regularization in regression: comparing Bayesian and frequ
entist
               methods in a poorly informative situation", 2009.

    make_spd_matrix(n_dim, random_state=None)
        Generate a random symmetric, positive-definite matrix.

        Read more in the :ref:`User Guide <sample_generators>`.
```

```
        Parameters
        ----------
        n_dim : int
            The matrix dimension.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_dim, n_dim]
            The random symmetric, positive-definite matrix.

        See also
        --------
        make_sparse_spd_matrix

    make_swiss_roll(n_samples=100, noise=0.0, random_state=None)
        Generate a swiss roll dataset.

        Read more in the :ref:`User Guide <sample_generators>`.

        Parameters
        ----------
        n_samples : int, optional (default=100)
            The number of sample points on the S curve.

        noise : float, optional (default=0.0)
            The standard deviation of the gaussian noise.

        random_state : int, RandomState instance or None (default)
            Determines random number generation for dataset creation. Pass
an int
            for reproducible output across multiple function calls.
            See :term:`Glossary <random_state>`.

        Returns
        -------
        X : array of shape [n_samples, 3]
            The points.

        t : array of shape [n_samples]
            The univariate position of the sample according to the main di
mension
            of the points in the manifold.

        Notes
        -----
        The algorithm is from Marsland [1].

        References
        ----------
        .. [1] S. Marsland, "Machine Learning: An Algorithmic Perspectiv
e",
               Chapter 10, 2009.
               http://seat.massey.ac.nz/personal/s.r.marsland/Code/10/lle.
py
```

```
    mldata_filename(dataname)
        DEPRECATED: mldata_filename was deprecated in version 0.20 and wil
l be removed in version 0.22

        Convert a raw name for a data set in a mldata.org filename.

            .. deprecated:: 0.20
                Will be removed in version 0.22

            Parameters
            ----------
            dataname : str
                Name of dataset

            Returns
            -------
            fname : str
                The converted dataname.

DATA
    __all__ = ['clear_data_home', 'dump_svmlight_file', 'fetch_20newsgrou
p...

FILE
    /home/a/lectenv/lib/python3.6/site-packages/sklearn/datasets/__init__.
py
```

Scripts

- Back to Python as a glue.
- Take us outside Notebooks.

In [ ]:

```
#!/usr/bin/env python3

# Why do we want the comment above?
# Tell us the interpreter.

if __name__ == "__main__":
    print("Running as main.")
else:
    print("I was imported!")
```

After lectuer: During demo in class, I made the file executable by `chmod +x foo.py` (as this was the name of my script).

- Several useful modules to check out. For instance `sys`, `subprocess` (OS commands).

Pointers to noteworthy data science packages

- matplotlib (https://matplotlib.org/) (home page with tutorials)
- scikit-learn (https://scikit-learn.org/stable/index.html) (import as sklearn)
- numpy (http://www.numpy.org/). Arrays!
- scipy (https://www.scipy.org/) Note: sparse matrices.
- pandas (https://pandas.pydata.org/)

- Entire courses.

Wrapping up