Python Programming for Data Science

Week 41, Friday

External modules:

- Scipy
- Scikit Learn

Last words on Python: nice-to-haves

- the zip function
- list-comprehension
- lambda functions

External modules: Scipy

Scipy

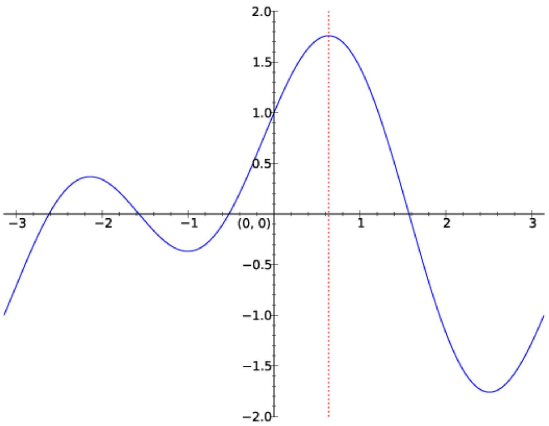
SciPy is a collection of scientific tools in Python. It contains modules for:

- optimization
- numerical integration
- special functions
- statistics
- interpolation
- signal processing
- linear algebra
- image processing
- ...

We'll just cover a very few basic examples here...

Scipy - optimization

Challenge: numerically find the optimal value of this function:
$f(x) = cos(x) + sin(2x)$
Import module




```
import scipy.optimize
```

Define function (note negative sign for minimization)

```
def f(x):
    return -(np.cos(x)+np.sin(2*x))
```

Minimize

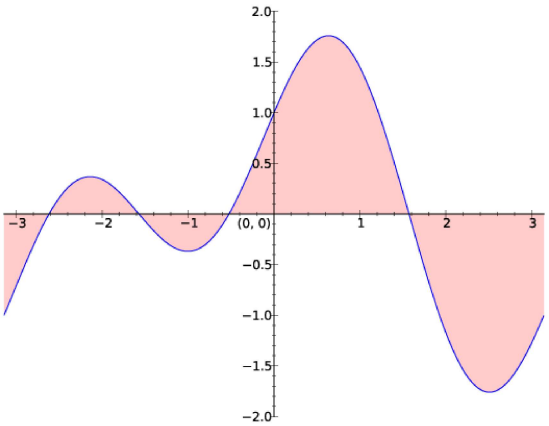
```
res = scipy.optimize.minimize(f, 0.0,
method="powell")
print(res.x)
```

output

```
[
0.6348
67]
```

Scipy - integration

Challenge: numerically find the area under the curve of:
$f(x) = cos(x) + sin(2x)$
Import module




```
import scipy.integrate
```

Define function

```
def f(x):
    return (np.cos(x)+np.sin(2*x))
```

Integrate from a to b

```
res, error = scipy.integrate.quad(f, a=-
np.pi, b=np.pi)
print(res)
```

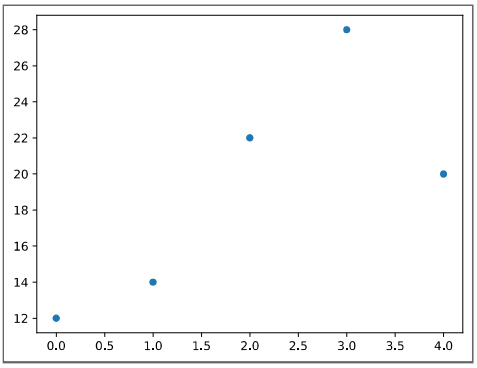
output

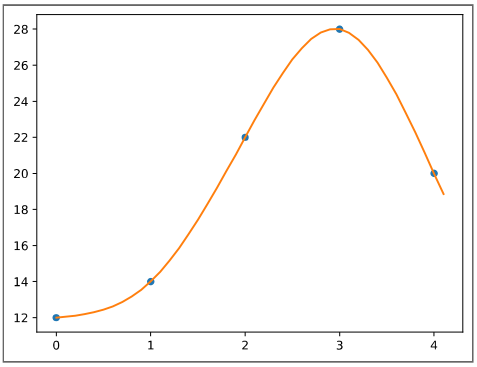
```
-6.975
7e-16
```

Interpolation

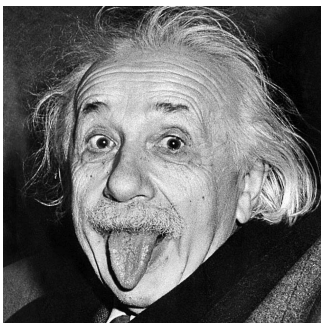
Challenge: interpolate between points

```
import numpy as np
from scipy.interpolate import
CubicSpline
import matplotlib.pyplot as plt

# Define original points
x = np.array([0, 1, 2, 3, 4])
y = np.array([12,14,22,28,20])

# Define spline
cs =
CubicSpline(x,y,bc_type='natural'
)

# Evaluate spline at new points
x2 = np.arange(0, 4.2, 0.1)
y2 = cs(x2)

plt.plot(x,y, '.', markersize=10)
plt.plot(x2,y2)
```





Scipy - image manipulation

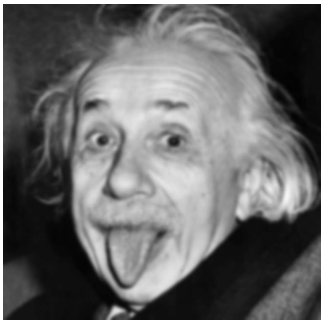


Challenge: blur an image

Import module

```
import imageio as iio
import scipy.ndimage
```

Load image (into a numpy array)



```
albert = iio.imread("albert.png")
print(albert.shape)
```

```
output
(940, 940)
```

Apply blur filter

```
blurred_albert =
scipy.ndimage.gaussian_filter(albert, sigma=5)
iio.imsave("blurred_albert.png", blurred_albert)
```

Scipy - Exercise

1. Write some code that finds the maximum of $cos(x) + sin(2 * x)$, like the example on the slides.
2. Try using -2.0 as a starting value. What happens?

Scipy - Solution

```
import scipy.optimize
```

1....like the examples on the slides.

```
def f(x):
    return -(np.cos(x)+np.sin(2*x))
res = scipy.optimize.minimize(f, x0=0.0, method="powell")
print(res.x)
```

```
output
0.634867833686
```

2. Try using -2.0 as a starting value. What happens?

```
res = scipy.optimize.minimize(f, x0=-2.0, method="powell")
print(res.x)
```

```
output
-2.13862566719
```

Conclusion: optimization is hard. Initial guesses are important.

3. Alternative for scalars (using brent optimization as default)

```
res = scipy.optimize.minimize_scalar(f)
print(res.x)
```

```
output
0.634867833686
```

External modules: scikit-learn

Scikit-learn

"Simple and efficient tools for data mining and data analysis"

- Classification
- Regression
- Clustering
- Model Selection
- Preprocessing

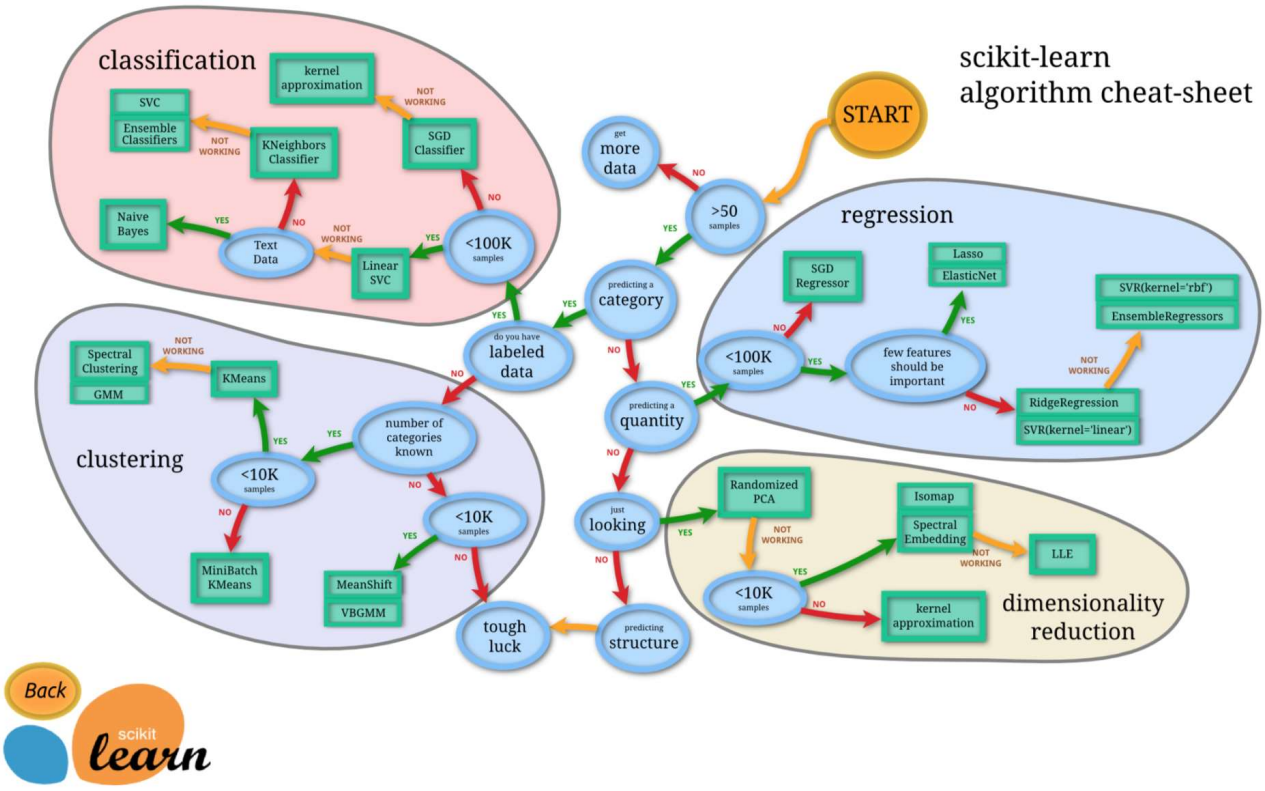
scikit-learn: importing

The module name of scikit learn is `sklearn`:

```
import sklearn
```

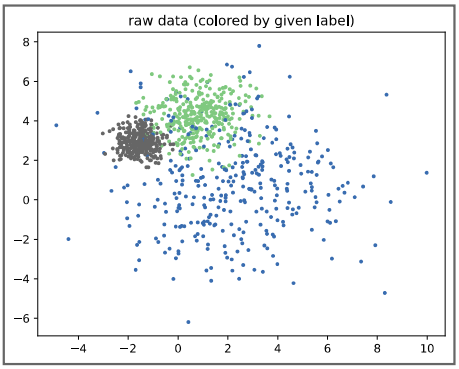
But most functionality is available through sub-modules:

```
import sklearn.cluster
import sklearn.linear_model
import sklearn.datasets
...
```

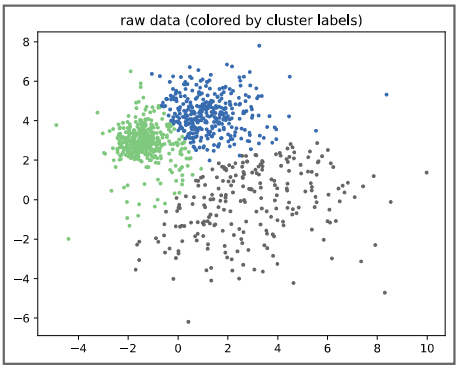


Example: clustering

```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import datasets,
cluster

# create a dataset: blobs with
varied variances
x, y = datasets.make_blobs(
    n_samples=1000,
    cluster_std=[1.0, 2.5, 0.5],
    random_state=0)

plt.scatter(x[:,0], x[:,1], c=y,
            s=5.0,
cmap=cm.get_cmap('Accent'))
plt.title('raw data (colored by
given label)')
```

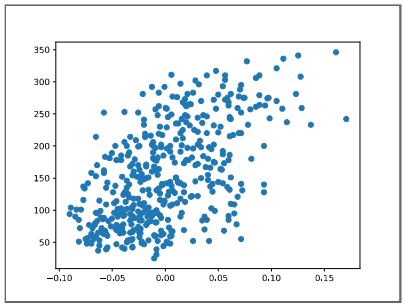



```
# Create and fit model
model =
cluster.KMeans(n_clusters=3)
model.fit(x)

plt.scatter(x[:,0], x[:,1],
c=model.labels_,
            s=5.0,
cmap=cm.get_cmap('Accent'))
plt.title('raw data (colored by
cluster labels)')
```

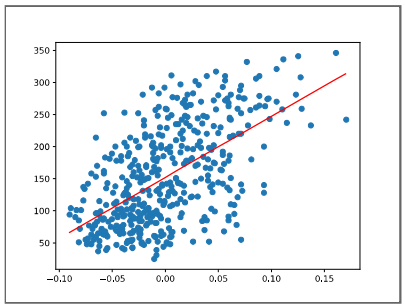


Example: regression

```
import matplotlib.pyplot as plt
from sklearn import datasets, linear_model
import numpy as np

x, y = datasets.load_diabetes(return_X_y=True)
x = x[:,2].reshape(-1,1)  # reshape to 2D

plt.scatter(x,y)
```

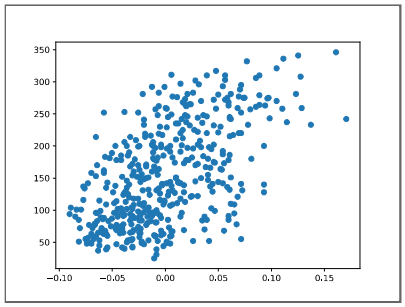



```
# Create and fit model
model = linear_model.LinearRegression()
model.fit(x,y)

# Generate x-values
x_predict = np.arange(min(x), max(x),
                      0.01).reshape(-1,1)
# reshape to 2D

# Calculate predicted values
y_predict = model.predict(x_predict)

plt.plot(x_predict, y_predict, c='red')
```

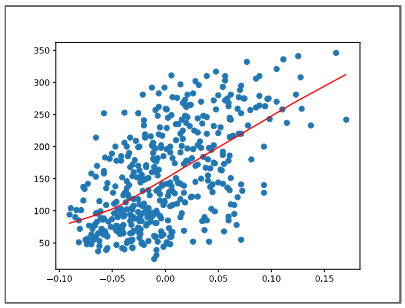


Example: regression with neural network

```
import matplotlib.pyplot as plt
from sklearn import datasets, neural_network
import numpy as np

x, y = datasets.load_diabetes(return_X_y=True)
x = x[:,2].reshape(-1,1)  # reshape to 2D

plt.scatter(x,y)
```




```
# Create and fit model
model = neural_network.MLPRegressor(hidden_layer_sizes=(100, 100, 100),
                    max_iter=200000)
model.fit(x,y)

# Generate x-values
x_predict = np.arange(min(x), max(x),
0.01).reshape(-1,1)  # reshape to 2D

# Calculate predicted values
y_predict = model.predict(x_predict)

plt.plot(x_predict, y_predict, c='red')
```



Note that I only needed to change two lines to make this work

Sklearn: exercise

1. Read through the code on the next slide and see if you can figure out what it does.
2. Run the code. What do the left and right plots represent? What is wrong?
3. See if you can fix this by switching to a different clustering algorithm: **https://scikit-learn.org/stable/modules/clustering.html**

```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import datasets, cluster

# Generate dataset
x,y = datasets.make_moons(n_samples=1000, noise=.05)

# Create and fit model
model = cluster.KMeans(n_clusters=2)
model.fit(x)

# Create 1x2 side-by-side plot
fig, axs = plt.subplots(1,2)

# Plot in the two figures
axs[0].scatter(x[:,0], x[:,1], s=5.0, c=y,
               cmap=cm.get_cmap('Accent'))
axs[1].scatter(x[:,0], x[:,1], c=model.labels_,
               s=5.0, cmap=cm.get_cmap('Accent'))
plt.show()
```

Sklearn: exercise - solution

1. Read through the code on the next slide and see if you can figure out what it does.
   Similar to clustering example from the slides, but on a different dataset. We use subplots to create a side-by-side plot, with the true labeling on the left, and the predicted on the right.
2. Run the code. What do the left and right plots represent? What is wrong?
   The clustering algorithm is not working very well

Sklearn: exercise - solution

3. See if you can fix this...
   One solution is to use the DBscan method. Requires some tinkering with the `eps` parameter.

```
...

# Create and fit model
model = cluster.DBSCAN(eps=.2)
model.fit(x)

...
```

Last words in Python: nice-to-haves

Nice to haves in the core Python language

We have now covered several external modules which make life easier.
There are also some tools in the core Python language that we originally skipped - but which can also be great time-savers.

The zip function

The `zip` function takes two lists and combines them into one. The new list will consist of tuples containing the values from the original lists.
As usual in python3 - `zip` will return an iterable, so we need to convert it to a list to see it:

```
x_values = [1.0, 3.0, 2.0]
y_values = [5.0, 6.0, 7.0]
xy_values = zip(x_values, y_values)
print(list(xy_values))
```

```
output

[(1.0, 5.0), (3.0, 6.0), (2.0, 7.0)]
```

Note that we can reproduce the behavior of `enumerate` this way:

```
my_list = ["how", "are", "you"]
print(list(zip(range(len(my_list)),
               my_list)))
```

```
output

[(0, 'how'), (1, 'are'), (2, 'you')]
```

List comprehension

In numpy and pandas, we have gotten used to doing element-wise operations.
Often, when iterating over a list in Python, you also do element-wise operation, and end up with a new list.
For cases like this, Python allows you to write your loop for compactly, using *list comprehension*:

```
my_list = ["how", "are", "you"]
new_list = [my_word.upper() for my_word
in my_list]
print(new_list)
```

```
output

['HOW', 'ARE',
'YOU']
```

Notice that list comprehension is just a shorthand for a loop:

```
# List comprehension version
new_list = [my_word.upper() for
my_word in my_list]
```

```
# Loop version
new_list = []
for my_word in
my_list:

new_list.append(my_wo
rd.upper())
```

List comprehension with a condition

List comprehensions allow you to specify a condition for when elements should be included:

```
my_list = [4,3,2,1]
print([x for x in my_list if x>1])
```

```
output

[4, 3, 2]
```

Nested list comprehension

You can use list comprehensions for lists of lists as well

```
xy_values = [[1.0, 5.0], [3.0, 6.0], [2.0, 7.0]]
xy_sq_values = [[value*value for value in xy_value] for
xy_value in xy_values]
print(xy_sq_values)
```

```
output
[[1.0, 25.0], [9.0, 36.0], [4.0, 49.0]]
```

Or, if you want to convert it into a single 1D list:

```
xy_sq_values = [value*value for xy_value in xy_values for
value in xy_value]
print(xy_sq_values)
```

```
output
[1.0, 25.0, 9.0, 36.0, 4.0, 49.0]
```

As you can see, it quickly becomes difficult to read. For nested lists, it is therefore often better to write an actual for-loop.

Dictionary comprehension

You can do the same with dictionaries. You just need to specify key:val pairs:

```
my_dict = {'mon':'monday', 'tue':'tuesday',
'wed':'wednesday'}
new_dict = {key:val.upper() for key,val in
my_dict.items()}
print(new_dict)
```

```
output
{'mon': 'MONDAY', 'tue': 'TUESDAY', 'wed': 'WEDNESDAY'}
```

Another example, where we build a dictionary from scratch:

```
{x:x**2 for x in range(4)}
```

```
output
{0: 0, 1: 1, 2: 4, 3:
9}
```

Lambda functions

Sometimes, it is convenient to specify a simple function as a one-liner.
Lambda functions allow you to do this - by defining an *anonymous* function:

```
lambda arguments: expression
```

Example:

```
# Define a function that squares its
input.
# Save it in variable my_function
my_function = lambda x: x*x
print(my_function(3.0))
print((lambda x: x*x)(3.0)) # or
without variable
```

```
output


9.0
9.0
```

Lambda functions - example of use case

Some python functions allow you to change their behavior by providing a function as argument. One example is `sorted()`

Per default, if we call `sorted` a list of tuples, it will sort by the first element in each tuple:

```
my_list = [(1, 'Love'), (0, 'Python'), (2, 'I')]
print(sorted(my_list))
```

```
output
[(0, 'Python'), (1, 'Love'), (2, 'I')]
```

Can we change this? From the documentation of `sorted()`:

```
Signature: sorted(iterable, /, *, key=None, reverse=False)
Docstring:
Return a new list containing all items from the iterable in ascending order.

A custom key function can be supplied to customize the sort order, and the
  reverse flag can be set to request the result in descending order.
```

Lambda functions - example of use case (2)

In the previous slide, we learned that we can give `sorted` a `key` argument, which specifies how it should select the value to sort by.

```
def my_key_function(tuple_value):
    '''Helper function for sorted - selects the second
value in each tuple'''
    return tuple_value[1]

my_list = [(1, 'Love'), (0, 'Python'), (2, 'I')]
# Pass my_key_function as arg to sorted:
print(sorted(my_list, key=my_key_function))
```

```
output
# list os now sorted alphabetically by the words
[(2, 'I'), (1, 'Love'), (0, 'Python')]
```

If we only use it once, it seems a bit silly to create a whole function for this. Instead, we can use a lambda function:

```
print(sorted(my_list, key=lambda
tuple_value:tuple_value[1]))
```

Exercise

Save this string in a variable called `my_str`:
"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it. - Brian Kernighan"

1. Write a one-liner that splits the string into a list of words and sorts this list by their length.
2. Write a one-liner that removes all words shorter than 9 characters from this string.

Exercise - solution

1. Write a one-liner that splits the string into a list of words and sorts this list by their length.

```
sorted(my_str.split(), key=len)
```

2. Write a one-liner that removes all words shorter than 9 characters from this string.

```
" ".join([word for word in my_str.split() if len(word)
> 8])
```

Speaker notes