Προγραμματιστικά Εργαλεία και Τεχνολογίες για Επιστήμη Δεδομένων

Παράδοση 31/1/2019, Νίκος Παπασπύρου.

SciPy

Το SciPy είναι ένα open-source οικοσύστημα της Python για μαθηματικά, επιστημονικούς υπολογισμούς και τους κλάδους των μηχανικών. Μερικά από τα συστατικά που το αποτελούν είναι τα εξής:

- NumPy, αποδοτικοί μαθηματικοί υπολογισμοί σε N-διάστατους χώρους. Το καλύψαμε λίγο στην προηγούμενη παράδοση.
- Βιβλιοθήκη SciPy, βασική βιβλιοθήκη επιστημονικών υπολογισμών. Με αυτό θα ασχοληθούμε λίγο τώρα.
- Matplotlib, βιβλιοθήκη για διδιάστατη απεικόνιση. Με αυτό θα ασχοληθούμε λίγο στη σημερινή παράδοση.
- IPython, μια εύχρηστη διαδραστική κονσόλα για την Python.
- Sympy, βιβλιοθήκη συμβολικών μαθηματικών υπολογισμών.
- pandas, βιβλιοθήκη δομών δεδομένων και ανάλυσης δεδομένων. Με αυτό θα ασχοληθούμε στη σημερινή παράδοση.

Απλή επεξεργασία εικόνων με το SciPy

1. Στη συνέχεια θα δούμε λίγα παραδείγματα επεξεργασίας εικόνων με χρήση της βιβλιοθήκης SciPy. Θα χρησιμοποιήσουμε τη βιβλιοθήκη imageio για το διάβασμα και την αποθήκευση των εικόνων.

   ```
   >>> import imageio
   >>> cat = imageio.imread('https://courses.softlab.ntua.gr/progds/2018b/cat.jpg')
   ```

   Η συνάρτηση `imread` διαβάζει ένα αρχείο εικόνας, που όπως βλέπετε μπορεί να δίνεται τοπικά ή σε κάποιο URL. Ο τύπος του αντικειμένου `cat` μοιάζει με τον τύπο των arrays του NumPy.

   ```
   >>> type(cat)
   <class 'imageio.core.util.Array'>
   >>> print(cat.dtype)
   uint8
   >>> print(cat.shape)
   (400, 248, 3)
   >>> print(cat)
   [[[132 128 117]
     ...
     [ 64  53  31]]
    [[134 130 118]
     ...
     [ 62  51  29]]
    ...
    [[124 151  84]
     ...
     [ 86  71  50]]]
   ```

   Ο πίνακας για μία έγχρωμη εικόνα έχει τρεις διαστάσεις. Οι δύο είναι χωρικές ενώ η τρίτη αντιστοιχεί στα τρία συστατικά χρώματος (RGB — red, green, blue) που συνθέτουν το χρώμα του αντίστοιχου pixel. (Βλ. Σχήμα 1.)

2. Έστω ότι θέλουμε να μικρύνουμε την εικόνα `cat` και να φτιάξουμε ένα "thumbnail" του οποίου η μεγαλύτερη διάσταση να είναι 60 pixels. Ξεκινάμε γράφοντας μία συνάρτηση που επιστρέφει μία τριάδα "zoom factors": συντελεστών με τους οποίους θα πολλαπλασιάσουμε τις διαστάσεις της αρχικής εικόνας ώστε να γίνουν ίσες με αυτές που θέλουμε. Φροντίζουμε να έχουμε τον ίδιο συντελεστή για τις δύο χωρικές διαστάσεις, ώστε να μην παραμορφωθεί η εικόνα, και να μην αλλοιώσουμε την τρίτη διάσταση (το χρώμα):

   ```
   >>> def zoom_factors(img, m=60):
   ...   x, y, _ = img.shape
   ...   z = m / max(x, y)
   ...   return (z, z, 1)
   ...
   >>> zoom_factors(cat)
   (0.15, 0.15, 1)
   ```

Σχήμα 1: Η αρχική εικόνα 400×248 — `cat.jpg`.

Στη συνέχεια, χρησιμοποιούμε τη συνάρτηση `zoom` του πακέτου `scipy.ndimage` για να μικρύνουμε την εικόνα:

```
>>> import scipy.ndimage
>>> cat_thumb = scipy.ndimage.zoom(cat, zoom_factors(cat))
>>> cat_thumb.shape
(60, 37, 3)
```

Βλέπουμε ότι η εικόνα που προέκυψε είναι όντως 60×37 — έχει την ίδια αναλογία με την αρχική και η μεγαλύτερη διάσταση είναι 60 pixels (όσο η default παράμετρος της `zoom_factors`). Τώρα μπορούμε να αποθηκεύσουμε τη νέα εικόνα και μάλιστα σε δύο αρχεία με διαφορετικό format, ανάλογα με την κατάληξη του αρχείου:

```
>>> imageio.imwrite('cat-thumbnail.jpg', cat_thumb)
>>> imageio.imwrite('cat-thumbnail.png', cat_thumb)
```

Το αποτέλεσμα φαίνεται στο Σχήμα 2. (Ίσως προσέξετε ότι το αποτέλεσμα δεν είναι καλό, μάλλον λόγω κακής λειτουργίας της συνάρτησης `ndimage.zoom`.)

Σχήμα 2: Το thumbnail 60×37 — `cat-thumbnail.png`.

3. Απλή επεξεργασία χρωμάτων. Με χρήση του πολλαπλασιασμού με το διάνυσμα `[0.5, 1, 0.75]` (και του broadcasting του NumPy), μετασχηματίζουμε τις χρωματικές συντεταγμένες της αρχικής εικόνας πολλαπλασιάζοντας το κόκκινο και το μπλε αντίστοιχα με 0.5 και 0.75.

```
>>> cat_funny = cat * [0.5, 1, 0.75]
>>> imageio.imwrite('funny.png', cat_funny)
WARNING:root:Lossy conversion from float64 to uint8. Range [1.5, 255.0].
Convert image to uint8 prior to saving to suppress this warning.
```

Βλέπουμε όμως ότι κατά την αποθήκευση του αρχείου, το πακέτο `imageio` διαμαρτύρεται. Παρατηρούμε ότι ο τύπος δεδομένων της εικόνας έχει αλλάξει από `uint8` σε `float64`

```
>>> print(cat_funny.dtype)
float64
```

Αυτό συμβαίνει γιατί ο πολλαπλασιασμός πινάκων στο NumPy προκαλεί τη μετατροπή του τύπου δεδομένων. Για να το διορθώσουμε, μπορούμε να μετατρέψουμε την εικόνα πάλι σε τύπο δεδομένων `uint8`:

```
>>> import numpy as np
```

```
>>> cat_funny = np.uint8(cat_funny)
>>> imageio.imwrite('funny.png', cat_funny)
```

Τώρα το αποτέλεσμα είναι αυτό του Σχήματος 3.

Σχήμα 3: Με αλλαγμένα τα χρώματα — `funny.png`.

4. Εξίσου εύκολα μπορούμε να δημιουργήσουμε το "αρνητικό" της εικόνας, αφαιρώντας από το 255 κάθε τιμή χρώματος (πάλι με broadcasting):

   ```
   >>> cat_funnier = 255 - cat_funny
   >>> imageio.imwrite('funnier.png', np.uint8(cat_funnier))
   ```

   Το αρνητικό είναι αυτό του Σχήματος 4.

Σχήμα 4: Με αλλαγμένα τα χρώματα — `funnier.png`.

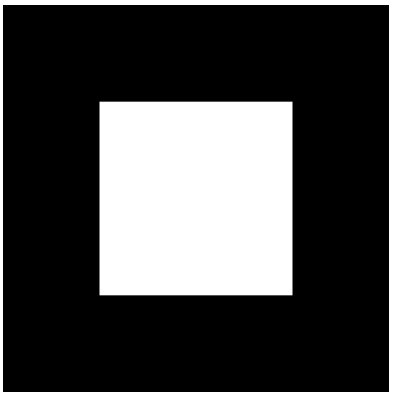
5. Τώρα θα προσπαθήσουμε να φτιάξουμε μία εικόνα από το μηδέν. Για αλλαγή, θα χρησιμοποιήσουμε πραγματικούς αριθμούς για τα συστατικά των χρωμάτων (0=μαύρο, 1=άσπρο). Ξεκινάμε με μία κενή εικόνα 256×256 με μηδενικά (δηλαδή μαύρη) και στη συνέχεια γεμίζουμε με άσσους (δηλαδή άσπρο) ένα μικρότερο τετράγωνο 128×128 στο κέντρο της:

```
>>> square = np.zeros((256, 256, 3))
>>> square[64:-64, 64:-64] = 1
>>> print(square.dtype)
float64
>>> print(type(square))
<class 'numpy.ndarray'>
```

Βλέπουμε όμως ότι ο τύπος αυτής της εικόνας δεν είναι `imageio.core.util.Array` όπως ήταν πριν η `cat`, αλλά ένας πίνακας του NumPy. Αυτό είναι λογικό, αφού χρησιμοποιήσαμε την `np.zeros` για να τη φτιάξουμε. Δε θα είναι πρόβλημα: η συνάρτηση `imageio.imwrite` δουλεύει σωστά και με πίνακες του NumPy.

```
>>> imageio.imwrite('square.png', np.uint8(square*255))
```

Το αποτέλεσμα είναι στο Σχήμα 5. Προσέξτε ότι χρειάστηκε να πολλαπλασιάσουμε με 255 πριν μετατρέψουμε σε `uint8` για να αποθηκευθούν τα χρώματα σωστά.

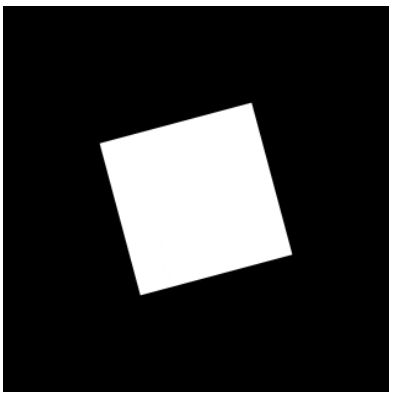


Σχήμα 5: Ένα άσπρο τετράγωνο — `square.png`.

6. Περιστρέφουμε την εικόνα κατά 15 μοίρες αριστερόστροφα.

```
>>> square = scipy.ndimage.rotate(square, 15, order=1)
>>> imageio.imwrite('square.png', np.uint8(square*255))
```

Το αποτέλεσμα είναι στο Σχήμα 6.



Σχήμα 6: Το άσπρο τετράγωνο μετά την περιστροφή — `square.png`.

7. Εφαρμόζουμε ένα φίλτρο Gauss για να θολώσουμε την εικόνα.

```
>>> square = scipy.ndimage.gaussian_filter(square, 8)
>>> imageio.imwrite('square.png', np.uint8(square*255))
```

Το αποτέλεσμα είναι στο Σχήμα 7.

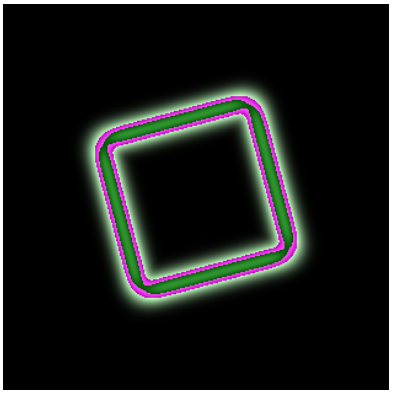
8. Τέλος, εφαρμόζουμε ένα φίλτρο Sobel για να βρούμε τις ακμές στην εικόνα που προέκυψε:

```
>>> sx = scipy.ndimage.sobel(square, axis=0, mode='constant')
>>> sy = scipy.ndimage.sobel(square, axis=1, mode='constant')
>>> edges = np.hypot(sx, sy)
```

Σχήμα 7: Το άσπρο τετράγωνο μετά το θόλωμα — `square.png`.

```
>>> imageio.imwrite('edges.png', np.uint8(edges*255))
```

Το αποτέλεσμα είναι στο Σχήμα 8.



Σχήμα 8: Εύρεση ακμών — `edges.png`.

9. Περισσότερες πληροφορίες για την επεξεργασία εικόνων με χρήση του SciPy.

Pandas

0. Ξεκινάμε φορτώνοντας δύο βασικά modules: το NumPy και το `pandas`.

```
>>> import numpy as np
>>> import pandas as pd
```

1. Βασικό είδος δεδομένων του `pandas` είναι το `Series`: ένα είδος μονοδιάστατου array τα στοιχεία του οποίου φέρουν ετικέτες (labels) και μπορούν να περιέχουν οποιονδήποτε τύπο δεδομένων (ακέραιους, πραγματικούς, συμβολοσειρές, αντικείμενα Python, κ.λπ.).

   Εδώ κατασκευάζουμε ένα τέτοιο `Series` με πέντε τυχαίους αριθμούς:

```
>>> s = pd.Series(np.random.randn(5))
>>> print(s)
0   -0.786549
1    0.581943
2   -0.150719
3   -0.485675
4    0.516488
dtype: float64
>>> print(type(s))
<class 'pandas.core.series.Series'>
```

   Τα στοιχεία αυτού του `Series` δεικτοδοτούνται με τους αριθμούς 0 έως και 4, όπως και αν είχαμε χρησιμοποιήσει ένα NumPy array για τον ίδιο σκοπό.

```
>>> print(s[2])
-0.1507186818477338
```

2. Όμως, μπορούμε να κατασκευάσουμε ένα ίδιο `Series` που να δεικτοδοτείται με ό,τι άλλες ετικέτες θέλουμε (εδώ τα γράμματα `'a'` έως και `'e'`):

```
>>> s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
>>> print(s)
a   -0.998521
b   -2.834074
c    1.334055
d   -0.001928
e   -0.480652
dtype: float64
>>> s.index
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
>>> print(s['a'])
-0.5093028206800931
>>> print(s['0'])
-0.5093028206800931
```

3. Τα `Series` είναι mutable:

```
>>> s['a'] = 42
>>> print(s['a'])
42.0
```

4. Μπορούμε να κατασκευάσουμε ένα `Series` από ένα λεξικό — σε αυτή την περίπτωση οι ετικέτες είναι τα κλειδιά του λεξικού:

```
>>> s = pd.Series({'a': 1, 'b': 2, 'c': 3})
>>> print(s)
a    1
b    2
c    3
dtype: int64
```

5. Επίσης, μπορούμε να κατασκευάσουμε ένα `Series` με broadcasting — εδώ όλα τα στοιχεία θα έχουν την ίδια τιμή:

```
>>> s = pd.Series(42, index=['a', 'b', 'c', 'd', 'e'])
>>> print(s)
a    42
b    42
c    42
d    42
e    42
dtype: int64
```

6. **Φέτες** σε `Series`: Λειτουργούν όπως περίπου και στο NumPy:

```
>>> s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
>>> print(s)
a   -0.319502
b   -1.179270
c    1.345146
d   -1.113787
e    0.783312
dtype: float64

>>> print(s[:3])
a   -0.319502
b   -1.179270
c    1.345146
dtype: float64
```

   Όπως επίσης και τα boolean slices:

```
>>> print(s.median())
-0.31950201314430705
>>> print(s[s > s.median()])
c    1.345146
e    0.783312
dtype: float64
```

7. Υπολογισμοί με `Series`:

```
>>> print(s*2 + 10)
a     9.360996
b     7.641459
c    12.690292
d     7.772425
e    11.566624
dtype: float64
```

8. Τα `Series` συμπεριφέρονται όπως οι μονοδιάστατοι πίνακες του NumPy, με εξαίρεση τη δυνατότητα χρήσης ετικετών. Ας δούμε όμως τώρα και πώς διαφέρουν. Ορίζουμε το εξής `Series`:

```
>>> s = pd.Series(np.arange(5), index=['a', 'b', 'c', 'd', 'e'])
>>> print(s)
a    0
b    1
c    2
d    3
e    4
dtype: int64
```

και απομονώνουμε δύο φέτες του:

```
>>> print(s[1:])
b    1
c    2
d    3
e    4
dtype: int64
>>> print(s[:-1])
a    0
b    1
c    2
d    3
dtype: int64
```

Αν αυτές ήταν πίνακες του NumPy και τις προσθέταμε, τότε τα στοιχεία τους θα προστίθεντο κατά μέλη: το 1ο με το 1ο, το 2ο με το 2ο, κ.ο.κ. Στο `pandas` αντίθετα:

```
>>> print(s[1:] + s[:-1])
a    NaN
b    2.0
c    4.0
d    6.0
e    NaN
dtype: float64
```

προστίθενται τα στοιχεία που έχουν την ίδια ετικέτα. Αν απουσιάζει μία ετικέτα από κάποιον προσθετέο (όπως εδώ συμβαίνει με την ετικέτα `'a'` του πρώτου και την ετικέτα `'e'` του δεύτερου), τότε το αποτέλεσμα είναι `np.nan` (που εμφανίζεται σαν 'NaN — not a number). Η τιμή αυτή αντιστοιχεί σε κενά στοιχεία ενός πίνακα. Μπορούμε να τη χρησιμοποιήσουμε και επί τούτου:

```
>>> s = pd.Series([1, 3, 5, np.nan, 6, 8])
>>> print(s)
0    1.0
1    3.0
2    5.0
3    NaN
```

```
4    6.0
5    8.0
dtype: float64
```

9. Το πιο συχνά χρησιμοποιούμενο είδος δεδομένων του `pandas` είναι το `DataFrame`: ένα είδος διδιάστατου array τα στοιχεία του οποίου φέρουν ετικέτες (labels) τόσο στις γραμμές όσο και στις στήλες και που κάθε στήλη μπορεί να περιέχει διαφορετικό τύπο δεδομένων. Τα `DataFrames` είναι σαν τα λογιστικά φύλλα του Excel ή σαν τους πίνακες στην SQL — μπορούμε επίσης να τα σκεφτόμαστε σαν λεξικά με τιμές τύπου `Series`.

   Μπορούμε να κατασκευάσουμε ένα `DataFrame` προσδιορίζοντας τα `Series` που το απαρτίζουν:

```
>>> s = pd.Series(np.arange(5), index=['a', 'b', 'c', 'd', 'e'])
>>> t = pd.Series(np.random.randn(5), index=['b', 'c', 'd', 'e', 'f'])
>>> print(s)
a    0
b    1
c    2
d    3
e    4
dtype: int64
>>> print(t)
b    0.574093
c    1.339853
d   -0.162268
e   -0.750811
f   -0.689334
dtype: float64
```

   και τώρα βάζοντας αυτά μαζί σε ένα λεξικό, από το οποίο κατασκευάζουμε το `DataFrame`:

```
>>> df = pd.DataFrame({'one': s, 'two': t})
>>> print(df)
   one       two
a  0.0       NaN
b  1.0  0.574093
c  2.0  1.339853
d  3.0 -0.162268
e  4.0 -0.750811
f  NaN -0.689334
```

10. Οι ετικέτες των γραμμών και των στηλών:

```
>>> print(df.index)
Index(['a', 'b', 'c', 'd', 'e', 'f'], dtype='object')
>>> print(df.columns)
Index(['one', 'two'], dtype='object')
```

11. Η δεικτοδότηση ενός `DataFrame` μπορεί να γίνει με τις ετικέτες των στηλών του, παίρνοντας έτσι τα αντίστοιχα `Series`:

```
>>> print(df['one'])
a    0.0
b    1.0
c    2.0
d    3.0
e    4.0
f    NaN
Name: one, dtype: float64
>>> print(type(df['one']))
<class 'pandas.core.series.Series'>
```

    Στη συνέχεια μπορεί κανείς να δεικτοδοτήσει με τις ετικέτες των γραμμών, παίρνοντας τα επί μέρους στοιχεία:

```
>>> print(df['one']['b'])
1.0
```

    Πιο κάτω θα δούμε ότι η δεικτοδότηση μπορεί να γίνει και με πολλούς άλλους τρόπους.

12. Τα `DataFrames` είναι mutable:

```
>>> df['one']['b'] = 42
>>> print(df)
    one       two
a   0.0       NaN
b  42.0  0.574093
c   2.0  1.339853
d   3.0 -0.162268
e   4.0 -0.750811
f   NaN -0.689334
```

13. Νέα `DataFrames` μπορούν να κατασκευαστούν αντιγράφοντας τμήματα άλλων `DataFrames`:

```
>>> print(pd.DataFrame(df, index=['d', 'a']))
   one       two
d  3.0 -0.162268
a  0.0       NaN
>>> print(pd.DataFrame(df, index=['d', 'a'], columns=['two']))
        two
d -0.162268
a       NaN
>>> print(pd.DataFrame(df, index=['d', 'a'], columns=['two', 'three']))
        two  three
d -0.162268    NaN
a       NaN    NaN
```

14. Παρακάτω χρησιμοποιούμε broadcasting για να κάνουμε όλα τα στοιχεία της στήλης `'two'` ίσα με 42:

```
>>> df['two'] = 42
>>> print(df)
   one  two
a  0.0   42
b  1.0   42
c  2.0   42
d  3.0   42
e  4.0   42
f  NaN   42
>>> print(df.dtypes)
one    float64
two      int64
dtype: object
```

Παρατηρούμε επίσης ότι οι τύποι δεδομένων των δύο στηλών είναι τώρα διαφορετικοί.

15. Το "ανάστροφο" ενός `DataFrame`, όπου οι στήλες έχουν γίνει γραμμές και αντίστροφα.

```
>>> print(df.T)
        a     b     c     d     e     f
one   0.0   1.0   2.0   3.0   4.0   NaN
two  42.0  42.0  42.0  42.0  42.0  42.0
```

16. Μπορούμε να χρησιμοποιήσουμε ένα range ημερομηνιών για ετικέτες:

```
>>> dates = pd.date_range('20190131', periods=7)
>>> print(dates)
DatetimeIndex(['2019-01-31', '2019-02-01', '2019-02-02', '2019-02-03',
               '2019-02-04', '2019-02-05', '2019-02-06'],
              dtype='datetime64[ns]', freq='D')
```

και στη συνέχεια να φτιάξουμε ένα `DataFrame` με αυτές τις ετικέτες για τις γραμμές του.

```
>>> df = pd.DataFrame(np.random.randn(7, 4), index=dates, columns=list('ABCD'))
>>> print(df)
                   A         B         C         D
2019-01-31 -0.463177 -0.233471 -2.115803  0.365349
2019-02-01 -0.587392 -0.685675 -0.026568  0.984521
```

```
2019-02-02 -0.475614 -0.348735 -0.755759 -1.094812
2019-02-03  2.021493 -0.138285  1.591936 -1.408307
2019-02-04  0.639041  0.405880  0.787391  0.529801
2019-02-05 -0.678607  1.411401  0.515590  0.166234
2019-02-06  0.343811 -0.161719 -1.448845  0.519112
```

17. Μπορούμε να ταξινομήσουμε τις στήλες κατά φθίνουσα σειρά ετικέτας:

```
>>> print(df.sort_index(axis=1, ascending=False))
                   D         C         B         A
2019-01-31  0.365349 -2.115803 -0.233471 -0.463177
2019-02-01  0.984521 -0.026568 -0.685675 -0.587392
2019-02-02 -1.094812 -0.755759 -0.348735 -0.475614
2019-02-03 -1.408307  1.591936 -0.138285  2.021493
2019-02-04  0.529801  0.787391  0.405880  0.639041
2019-02-05  0.166234  0.515590  1.411401 -0.678607
2019-02-06  0.519112 -1.448845 -0.161719  0.343811
```

18. Χρησιμότερο όμως είναι να ταξινομήσουμε βάσει των περιεχόμενων ενός `DataFrame` και όχι βάσει των ετικετών του. Εδώ οι γραμμές ταξινομούνται βάσει της τιμής της στήλης `'B'`:

```
>>> print(df.sort_values(by='B'))
                   A         B         C         D
2019-02-01 -0.587392 -0.685675 -0.026568  0.984521
2019-02-02 -0.475614 -0.348735 -0.755759 -1.094812
2019-01-31 -0.463177 -0.233471 -2.115803  0.365349
2019-02-06  0.343811 -0.161719 -1.448845  0.519112
2019-02-03  2.021493 -0.138285  1.591936 -1.408307
2019-02-04  0.639041  0.405880  0.787391  0.529801
2019-02-05 -0.678607  1.411401  0.515590  0.166234
```

19. Επιλέγουμε μερικές στήλες:

```
>>> print(df[['A', 'B']])
                   A         B
2019-01-31 -0.463177 -0.233471
2019-02-01 -0.587392 -0.685675
2019-02-02 -0.475614 -0.348735
2019-02-03  2.021493 -0.138285
2019-02-04  0.639041  0.405880
2019-02-05 -0.678607  1.411401
2019-02-06  0.343811 -0.161719
```

20. Επιλέγουμε μερικές γραμμές:

```
>>> print(df[0:3])
                   A         B         C         D
2019-01-31 -0.463177 -0.233471 -2.115803  0.365349
2019-02-01 -0.587392 -0.685675 -0.026568  0.984521
2019-02-02 -0.475614 -0.348735 -0.755759 -1.094812
```

21. Επιλέγουμε μία γραμμή βάσει της ετικέτας της. Προσέξτε ότι το αποτέλεσμα είναι ένα `Series` που αντιστοιχεί στη γραμμή:

```
>>> df.loc["2019-01-31"]
A   -0.463177
B   -0.233471
C   -2.115803
D    0.365349
Name: 2019-01-31 00:00:00, dtype: float64
```

22. Επιλέγουμε ένα range γραμμών βάσει των ετικετών τους και δύο στήλες:

```
>>> print(df.loc['20190201':'20190203', ['A', 'B']])
                   A         B
2019-02-01 -0.587392 -0.685675
2019-02-02 -0.475614 -0.348735
```

```
2019-02-03  2.021493 -0.138285
```

23. Επιλέγουμε ένα μεμονωμένο στοιχείο βάσει των ετικετών γραμμής και στήλης:

```
>>> print(df.loc['20190201', 'A'])
-0.5873920154160649
```

Το ίδιο μπορεί να γίνει πιο αποδοτικά με το `at`, στο οποίο όμως πρέπει να δώσουμε την ακριβή ετικέτα της γραμμής και της στήλης. Προσέξτε ότι στο παραπάνω παράδειγμα, η ετικέτα της γραμμής είναι η ημερομηνία `dates[1]` και όχι η συμβολοσειρά `'20190201'`:

```
>>> dates[1]
Timestamp('2019-02-01 00:00:00', freq='D')
>>> df.at[dates[1], 'A']
-0.5873920154160649
```

24. Αντίστοιχα, τα `iloc` και `iat` κάνουν το ίδιο αλλά με αριθμητικούς δείκτες αντί ετικετών:

```
>>> print(df.iloc[3])
A    2.021493
B   -0.138285
C    1.591936
D   -1.408307
Name: 2019-02-03 00:00:00, dtype: float64
>>> print(df.iloc[3:5, 0:2])
                   A         B
2019-02-03  2.021493 -0.138285
2019-02-04  0.639041  0.405880
>>> print(df.iat[1, 1])
-0.6856745633031265
```

25. Επιλέγουμε με boolean indices:

```
>>> print(df[df.A > 0])
                   A         B         C         D
2019-02-03  2.021493 -0.138285  1.591936 -1.408307
2019-02-04  0.639041  0.405880  0.787391  0.529801
2019-02-06  0.343811 -0.161719 -1.448845  0.519112
>>> print(df[df > 0])
                   A         B         C         D
2019-01-31       NaN       NaN       NaN  0.365349
2019-02-01       NaN       NaN       NaN  0.984521
2019-02-02       NaN       NaN       NaN       NaN
2019-02-03  2.021493       NaN  1.591936       NaN
2019-02-04  0.639041  0.405880  0.787391  0.529801
2019-02-05       NaN  1.411401  0.515590  0.166234
2019-02-06  0.343811       NaN       NaN  0.519112
```

26. Τέλος μπορούμε να αποθηκεύσουμε το αποτέλεσμα σε ένα αρχείο CSV:

```
>>> df.to_csv("pandas.csv")
```

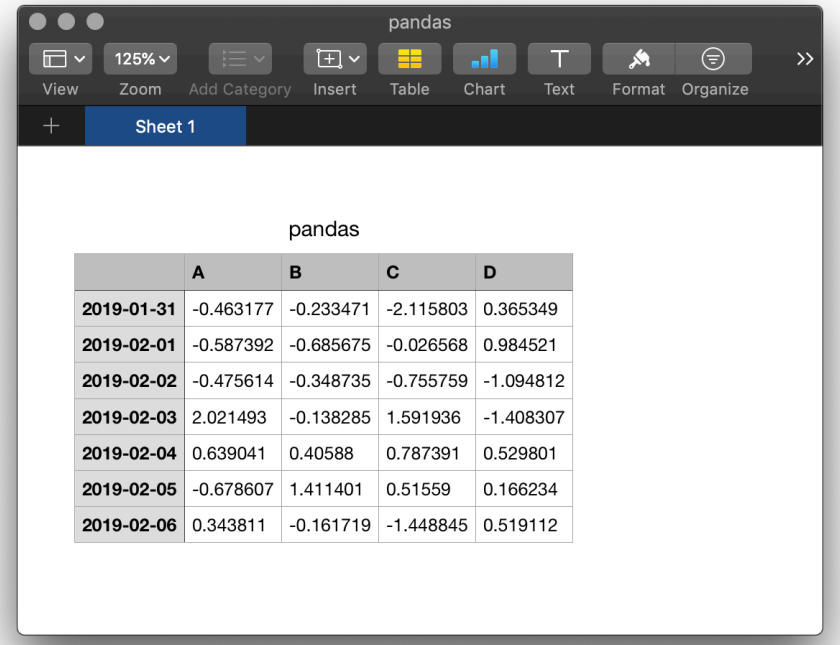
και στη συνέχεια να το ανοίξουμε π.χ. με το Excel ή κάποια άλλη αντίστοιχη εφαρμογή. (Βλ. Σχήμα 9.)

27. Επίσης μπορούμε να αποθηκεύσουμε το αποτέλεσμα σε ένα αρχείο Excel. (Βλ. Σχήμα 10. Πιθανώς θα χρειαστεί να εγκαταστήσετε το πακέτο `xlsxwriter`.)

```
>>> df.to_excel('pandas.xlsx', sheet_name='Date Example')
```

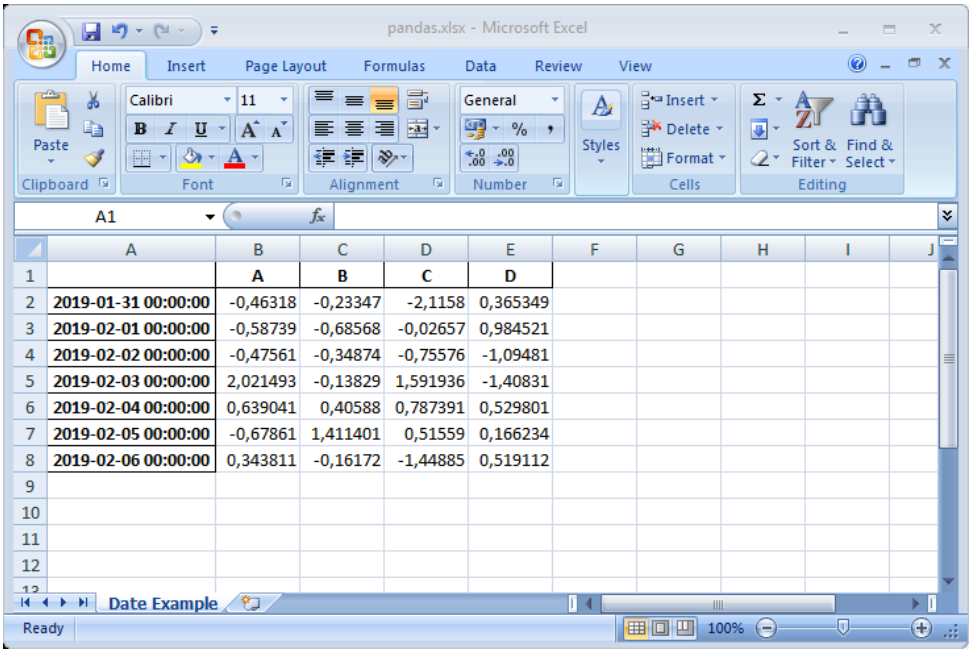
28. Διάβασμα από αρχείο CSV.

```
>>> print(pd.read_csv("pandas.csv"))
   Unnamed: 0         A         B         C         D
0  2019-01-31 -0.463177 -0.233471 -2.115803  0.365349
1  2019-02-01 -0.587392 -0.685675 -0.026568  0.984521
2  2019-02-02 -0.475614 -0.348735 -0.755759 -1.094812
3  2019-02-03  2.021493 -0.138285  1.591936 -1.408307
4  2019-02-04  0.639041  0.405880  0.787391  0.529801
```



	A	B	C	D
2019-01-31	-0.463177	-0.233471	-2.115803	0.365349
2019-02-01	-0.587392	-0.685675	-0.026568	0.984521
2019-02-02	-0.475614	-0.348735	-0.755759	-1.094812
2019-02-03	2.021493	-0.138285	1.591936	-1.408307
2019-02-04	0.639041	0.40588	0.787391	0.529801
2019-02-05	-0.678607	1.411401	0.51559	0.166234
2019-02-06	0.343811	-0.161719	-1.448845	0.519112

Σχήμα 9: Αποθήκευση σε αρχείο CSV — `pandas.csv`.



	A	B	C	D
2019-01-31 00:00:00	-0,46318	-0,23347	-2,1158	0,365349
2019-02-01 00:00:00	-0,58739	-0,68568	-0,02657	0,984521
2019-02-02 00:00:00	-0,47561	-0,34874	-0,75576	-1,09481
2019-02-03 00:00:00	2,021493	-0,13829	1,591936	-1,40831
2019-02-04 00:00:00	0,639041	0,40588	0,787391	0,529801
2019-02-05 00:00:00	-0,67861	1,411401	0,51559	0,166234
2019-02-06 00:00:00	0,343811	-0,16172	-1,44885	0,519112

Σχήμα 10: Αποθήκευση σε αρχείο Excel — `pandas.xlsx`.

```
5  2019-02-05 -0.678607  1.411401  0.515590  0.166234
6  2019-02-06  0.343811 -0.161719 -1.448845  0.519112
```

Προσέξτε ότι οι ετικέτες των γραμμών είναι οι αριθμοί 0 έως και 6, αντί των ημερομηνιών. Αν θέλουμε να επαναφέρουμε ως ετικέτες τις ημερομηνίες θα πρέπει να προσδιορίσουμε ότι η πρώτη στήλη περιέχει τις ετικέτες των γραμμών:

```
>>> print(pd.read_csv("pandas.csv", index_col=0))
                   A         B         C         D
2019-01-31 -0.463177 -0.233471 -2.115803  0.365349
2019-02-01 -0.587392 -0.685675 -0.026568  0.984521
2019-02-02 -0.475614 -0.348735 -0.755759 -1.094812
2019-02-03  2.021493 -0.138285  1.591936 -1.408307
2019-02-04  0.639041  0.405880  0.787391  0.529801
2019-02-05 -0.678607  1.411401  0.515590  0.166234
2019-02-06  0.343811 -0.161719 -1.448845  0.519112
```

29. Διάβασμα από αρχείο Excel. (Πιθανώς θα χρειαστεί να εγκαταστήσετε το πακέτο `xlrd`.)

```
>>> print(pd.read_excel('pandas.xlsx', 'Date Example', index_col=0))
                   A         B         C         D
2019-01-31 -0.463177 -0.233471 -2.115803  0.365349
2019-02-01 -0.587392 -0.685675 -0.026568  0.984521
2019-02-02 -0.475614 -0.348735 -0.755759 -1.094812
2019-02-03  2.021493 -0.138285  1.591936 -1.408307
2019-02-04  0.639041  0.405880  0.787391  0.529801
2019-02-05 -0.678607  1.411401  0.515590  0.166234
2019-02-06  0.343811 -0.161719 -1.448845  0.519112
```