

BİLİŞİM SİSTEMLERİ MÜHENDİSLİĞİ TASARIMI

Rapor 1

Konu : Python for Data Analysis

Recep Aydoğdu

B171200017

İçindekiler

Chapter 1	2
1. Preliminaries	2
1.1 What Is This Book About?	2
What Kinds of Data?	2
1.2 Why Python for Data Analysis?	2
1.3 Essential Python Libraries	3
NumPy	3
Pandas	3
Matplotlib	3
IPython and Jupyter	3
SciPy	4
Scikit-learn	4
Statsmodels	4
Chapter 2	5
2. Python Language Basics, IPython, and Jupyter Notebooks	5
2.1 The Python Interpreter	5
2.2 IPython Basics	6
Running the IPython Shell	6
Running the Jupyter Notebook	7
Tab Completion	10
Introspection	12
The %run Command	13
Interrupting Running Code	14
Executing Code from the Clipboard	14
Terminal Keyboard Shortcuts	15
About Magic Commands	16
Matplotlib Integration	18
2.3 Python Language Basics	19

Chapter 1

1. Preliminaries

1.1 What Is This Book About?

Bu kitap, Python'da verilerin manipüle edilmesi (manipulation), işlenmesi (preprocessing), temizlenmesi (cleaning) ve ezilmesiyle (crunching) ilgili ayrıntılar ile ilgilidir. Amacımız, Python programlama dilinin bölümleri ve veri odaklı kütüphane ekosistemi ve sizi etkili bir veri analisti olmamızı sağlayacak araçlar için bir rehber sunmaktır. Kitabın başlığında “veri analizi” yer alırken, veri analizi metodolojisinin aksine odak noktası özellikle Python programlama, kitaplıklar ve araçlar üzerinedir. Bu, veri analizi için ihtiyaç duyduğunuz Python programlamasıdır.

What Kinds of Data?

"Veri" dediğimizde tam olarak neyi kastediyoruz? Birincil odak noktası, aşağıdakiler gibi birçok farklı ortak veri biçimini kapsayan kasıtlı olarak belirsiz bir terim olan yapılandırılmış (**structured**) veriler üzerinedir:

- Her sütunun farklı bir türde (string, numeric, date veya başka türlü) olabileceği tablo veya elektronik tablo benzeri veriler. Bu, genellikle ilişkisel veritabanlarında veya sekme veya virgülle sınırlanmış metin dosyalarında depolanan çoğu veri türünü içerir.
- Çok boyutlu diziler (matrisler).
- Anahtar sütunlarla (bir SQL kullanıcısı için primary veya foreign key'ler olabilir) birbiriyle ilişkili birden çok veri tablosu.
- Eşit veya düzensiz aralıklı zaman serileri.

Bu kesinlikle tam bir liste değildir. Her zaman açık olmasa da, veri kümelerinin büyük bir yüzdesi, analiz ve modelleme için daha uygun olan yapılandırılmış bir forma dönüştürülebilir.

Bir veri kümesindeki özellikleri yapılandırılmış bir forma çıkarmak mümkün olabilir. Örnek olarak, bir haber makaleleri koleksiyonu, daha sonra duygu analizi yapmak için kullanılabilen bir kelime sıklığı tablosuna işlenebilir.

1.2 Why Python for Data Analysis?

Çoğu insan için Python programlama dili güçlü bir çekiciliğe sahiptir. 1991'deki ilk ortaya çıkışından bu yana, Python Perl, Ruby ve diğerleri ile birlikte en popüler yorumlanmış programlama dillerinden biri haline geldi. Python ve Ruby, 2005 yılından beri Rails (Ruby) ve Django (Python) gibi çok sayıda web framework'ünü kullanarak web siteleri oluşturmak için özellikle popüler hale geldi. Bu tür diller, küçük programları veya diğer görevleri otomatikleştirmek için komut dosyalarını hızlı bir şekilde yazmak için kullanılabildikleri için genellikle komut dosyası dilleri olarak adlandırılır. Ciddi bir yazılım oluşturmak için kullanılamayacakları çağrışımını taşıdığı için "komut dosyası dili" terimini sevmiyorum. Python, çeşitli tarihsel ve kültürel nedenlerle yorumlanan diller arasında geniş ve aktif bir bilimsel hesaplama ve veri analizi topluluğu geliştirmiştir. Son 10 yılda, Python bir kanama noktasından veya "riski size ait olmak üzere" bilimsel hesaplama dilinden, akademi ve endüstride veri bilimi, makine öğrenimi ve genel yazılım geliştirme için en önemli dillerden birine geçti.

Veri analizi ve etkileşimli hesaplama ve veri görselleştirme için Python kaçınılmaz olarak diğer açık kaynak ve ticari programlama dilleri ve R, MATLAB, SAS, Stata ve diğerleri gibi geniş

kullanımda olan araçlarla karşılaştırmalar yapacaktır. Son yıllarda, Python'un kitaplıklar için geliştirilmiş desteği (pandas ve scikit-learn gibi), onu veri analizi görevleri için popüler bir seçim haline getirmiştir. Python'un genel amaçlı yazılım mühendisliği için genel gücüyle birleştiğinde, veri uygulamaları oluşturmak için birincil dil olarak mükemmel bir seçenektir.

1.3 Essential Python Libraries

NumPy

Numerical Python'un kısaltması olan NumPy, uzun zamandır Python'da sayısal hesaplamanın temel taşı olmuştur. Python'da sayısal verileri içeren çoğu bilimsel uygulama için gereken veri yapılarını ve algoritmaları sağlar. NumPy, diğer şeylerin yanı sıra şunları içerir:

- Hızlı ve verimli çok boyutlu bir dizi nesnesi *ndarray*
- Dizilerle eleman bazlı hesaplamalar veya diziler arasında matematiksel işlemler gerçekleştirmek için işlevler.
- Dizi tabanlı veri kümelerini okumak ve diske yazmak için araçlar.
- Doğrusal cebir işlemleri, Fourier dönüşümü ve rastgele sayı üretimi
- Python uzantılarının ve yerel C veya C ++ kodunun NumPy'nin veri yapılarına ve hesaplama tesislerine erişmesini sağlayan uygun bir C API

NumPy'nin Python'a eklediği hızlı dizi işleme yeteneklerinin ötesinde, veri analizindeki birincil kullanımlarından biri, algoritmalar ve kitaplıklar arasında veri aktarımı için bir container'dır. Sayısal veriler için NumPy dizileri, verileri depolamak ve işlemek için diğer yerleşik Python veri yapılarına göre daha etkilidir. Ayrıca, C veya Fortran gibi daha düşük seviyeli bir dilde yazılmış kitaplıklar, verileri başka bir bellek temsiline kopyalamadan NumPy dizisinde depolanan veriler üzerinde çalışabilir. Bu nedenle, Python için birçok sayısal hesaplama aracı, NumPy dizilerini birincil veri yapısı olarak varsayar veya NumPy ile sorunsuz birlikte çalışabilirliği hedefler.

Pandas

Pandas, yapılandırılmış veya tablo şeklindeki verilerle çalışmayı hızlı, kolay ve anlamlı hale getirmek için tasarlanmış üst düzey veri yapıları ve işlevleri sağlar. 2010'daki ortaya çıkışından bu yana, Python'un güçlü ve verimli bir veri analizi ortamı olmasına yardımcı oldu. Bu kitapta kullanılacak pandas'daki birincil nesneler, hem satır hem de sütun etiketlerine sahip tablo şeklinde, sütun yönelimli bir veri yapısı olan DataFrame ve tek boyutlu etiketli bir dizi nesnesi olan Series'dir.

Pandas, NumPy'nin yüksek performanslı, dizi hesaplama fikirlerini elektronik tabloların ve ilişkisel veritabanlarının (SQL gibi) esnek veri işleme yetenekleriyle harmanlamaktadır.

Matplotlib

Matplotlib, grafikler ve diğer iki boyutlu veri görselleştirmeleri üretmek için en popüler Python kitaplığıdır.

IPython and Jupyter

IPython projesi, 2001 yılında Fernando Pérez'in daha iyi etkileşimli bir Python yorumlayıcısı yapma yan projesi olarak başladı. Sonraki 16 yılda, modern Python veri yığınınındaki en önemli araçlardan biri haline geldi. Kendi başına herhangi bir hesaplama veya veri analitik aracı sağlamazken, IPython hem etkileşimli hesaplamada hem de yazılım geliştirmede

retkenliĐinizi en st dzeye ıkarmak iin sıfırdan tasarlanmıřtır. DiĐer birok programlama dilinin tipik dzenleme-derleyici alıřma iř akıřı yerine yrtme-keřfetme iř akıřını teřvik eder. Ayrıca iřletim sisteminizin kabuĐuna ve dosya sistemine kolay eriřim saĐlar. Veri analizi kodlamasının oĐu keřif, deneme yanılma ve yineleme ierdiĐinden, IPython iři daha hızlı tamamlamanıza yardımcı olabilir.

2014'te Fernando ve IPython ekibi, dilden baĐımsız etkileřimli bilgi iřlem aralarını tasarlamak iin daha geniř bir giriřim olan Jupyter projesini duyurdu. IPython web notebook'u, řimdi 40'tan fazla programlama dilini destekleyen Jupyter notebook oldu. IPython sistemi artık Python'u Jupyter ile kullanmak iin bir kernel (bir programlama dili modu) olarak kullanılabilir.

SciPy

SciPy, bilimsel hesaplamada bir dizi farklı standart problem alanını ele alan bir paketler koleksiyonudur.

Scikit-learn

2010 yılında projenin bařlangıcından bu yana scikit-learn, Python programcıları iin nde gelen genel amalı makine Đrenimi ara seti haline geldi. Yalnızca yedi yıl iinde, dnyanın drt bir yanından 1.500'den fazla katılımcısı oldu. Bu tr modeller iin alt modller ierir:

- Classification: SVM, nearest neighbors, random forest, logistic regression, etc.
- Regression: Lasso, ridge regression, etc.
- Clustering: k-means, spectral clustering, etc.
- Dimensionality reduction: PCA, feature selection, matrix factorization, etc.
- Model selection: Grid search, cross-validation, metrics
- Preprocessing: Feature extraction, normalization

Statsmodels

statsmodels, R programlama dilinde popler bir dizi regresyon analizi modelini uygulayan Stanford niversitesi istatistik profesr Jonathan Taylor'ın alıřmasıyla tohumlanan bir istatistiksel analiz paketidir.

Chapter 2

2. Python Language Basics, IPython, and Jupyter Notebooks

2.1 The Python Interpreter

Python yorumlanmış bir dildir. Python yorumlayıcısı, her seferinde bir ifadeyi çalıştırarak bir programı çalıştırır. Standart etkileşimli Python yorumlayıcısı, python komutuyla komut satırından çağrılabilir:

```
$ python
Python 3.6.0 | packaged by conda-forge | (default, Jan 13 2017, 23:17:12)
[GCC 4.8.2 20140120 (Red Hat 4.8.2-15)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 5
>>> print(a)
5
```

Gördüğünüz >>>, kod ifadelerini yazacağınız istemdir. Python yorumlayıcısından çıkmak ve komut istemine dönmek için **exit ()** yazabilir veya Ctrl-D tuşlarına basabilirsiniz.

Python programlarını çalıştırmak, ilk argümanı olarak bir .py dosyasıyla python'u çağırmak kadar basittir. Şu içeriklerle *hello_world.py* oluşturduğumuzu varsayalım:

```
print('Hello world')
```

Aşağıdaki komutu çalıştırarak çalıştırabilirsiniz (*hello_world.py* dosyası, geçerli çalışan terminal dizininizde olmalıdır):

```
$ python hello_world.py
Hello world
```

Bazı Python programcıları tüm Python kodlarını bu şekilde çalıştırırken, veri analizi veya bilimsel hesaplama yapanlar IPython, gelişmiş bir Python yorumlayıcısı veya orijinal olarak IPython projesi içinde oluşturulan web tabanlı kod defterleri olan Jupyter not defterlerini kullanırlar.

`%run` komutunu kullandığınızda, IPython belirtilen dosyadaki kodu aynı işlemde çalıştırarak keşfetmenizi sağlar tamamlandığında etkileşimli olarak sonuçlar:

```
$ ipython
Python 3.6.0 | packaged by conda-forge | (default, Jan 13 2017, 23:17:12)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- An enhanced Interactive Python.
?      -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help    -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

In [1]: %run hello_world.py
Hello world

In [2]:
```

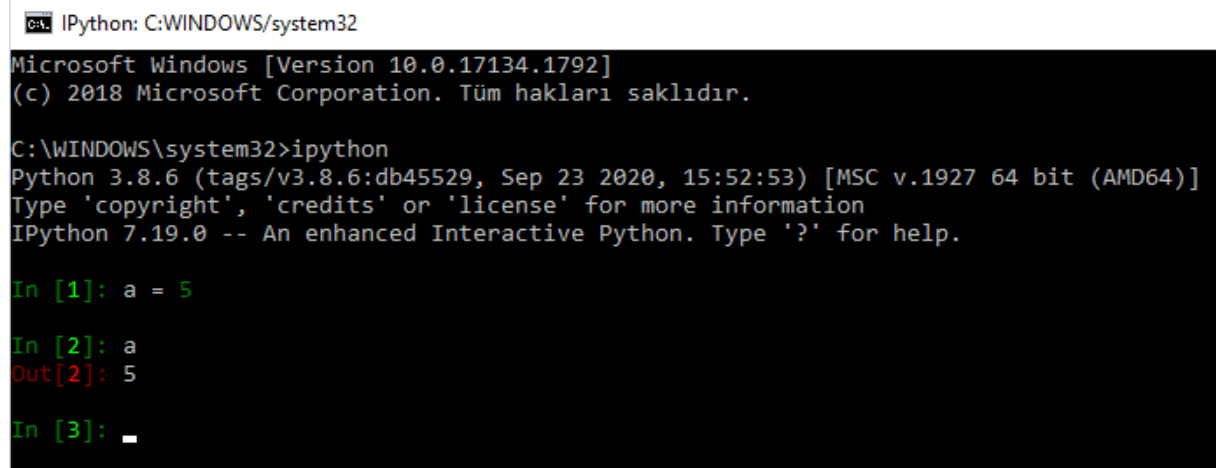
Varsayılan IPython komut istemi, numaralandırılmış [2]: stilini standart `>>>` komut istemiyle karşılaştırır.

2.2 IPython Basics

Bu bölümde, IPython shell', ve Jupyter Notebook'a hazırlanıp çalışacağız ve bazı temel kavramları size tanıyacağız.

Running the IPython Shell

IPython kabuğunu, ipython komutu haricinde normal Python yorumlayıcısını başlatır gibi komut satırında başlatabiliriz:



```

C:\WINDOWS\system32>ipython
Microsoft Windows [Version 10.0.17134.1792]
(c) 2018 Microsoft Corporation. Tüm hakları saklıdır.

C:\WINDOWS\system32>ipython
Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v.1927 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.19.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: a = 5

In [2]: a
Out[2]: 5

In [3]: _
```

İsteğe bağlı Python ifadelerini yazarak ve Return (veya Enter) tuşuna basarak çalıştırabilirsiniz. IPython'a sadece bir değişken yazdığınızda, nesnenin bir string temsilini oluşturur:

```
[4]: import numpy as np
```

```
[5]: data = {i : np.random.randn() for i in range(7)}
```

```
[6]: data
```

```
[6]: {0: -0.3982976406984385,  
      1: -0.07776196283978527,  
      2: -0.11967699048385459,  
      3: 0.5902793183256904,  
      4: 2.171831990922387,  
      5: 0.1990260547753195,  
      6: -0.278082407614929}
```

İlk iki satır Python kod ifadeleridir; ikinci ifade, yeni oluşturulan bir Python sözlüğüne başvuran `data` adlı bir değişken oluşturur. Son satır, konsoldaki verilerin değerini yazdırır.

Birçok Python nesnesi türü, normal `print` ile yazdırmaktan farklı olarak, daha okunabilir veya güzel basılmış olacak şekilde biçimlendirilmiştir. Yukarıdaki veri değişkenini standart Python yorumlayıcısında yazdırırsanız, çok daha az okunabilir olacaktır:

```
[9]: from numpy.random import randn
```

```
[10]: data = {i : randn() for i in range(7)}
```

```
[11]: print(data)
```

```
{0: 0.1089264969992243, 1: 0.031039188304209056, 2: -1.3404162410677365, 3: 0.23966690069381563,  
 4: -0.41758532623374023, 5: 0.1476185674601656, 6: -0.8879869022124973}
```

```
[12]: data
```

```
[12]: {0: 0.1089264969992243,  
      1: 0.031039188304209056,  
      2: -1.3404162410677365,  
      3: 0.23966690069381563,  
      4: -0.41758532623374023,  
      5: 0.1476185674601656,  
      6: -0.8879869022124973}
```

Running the Jupyter Notebook

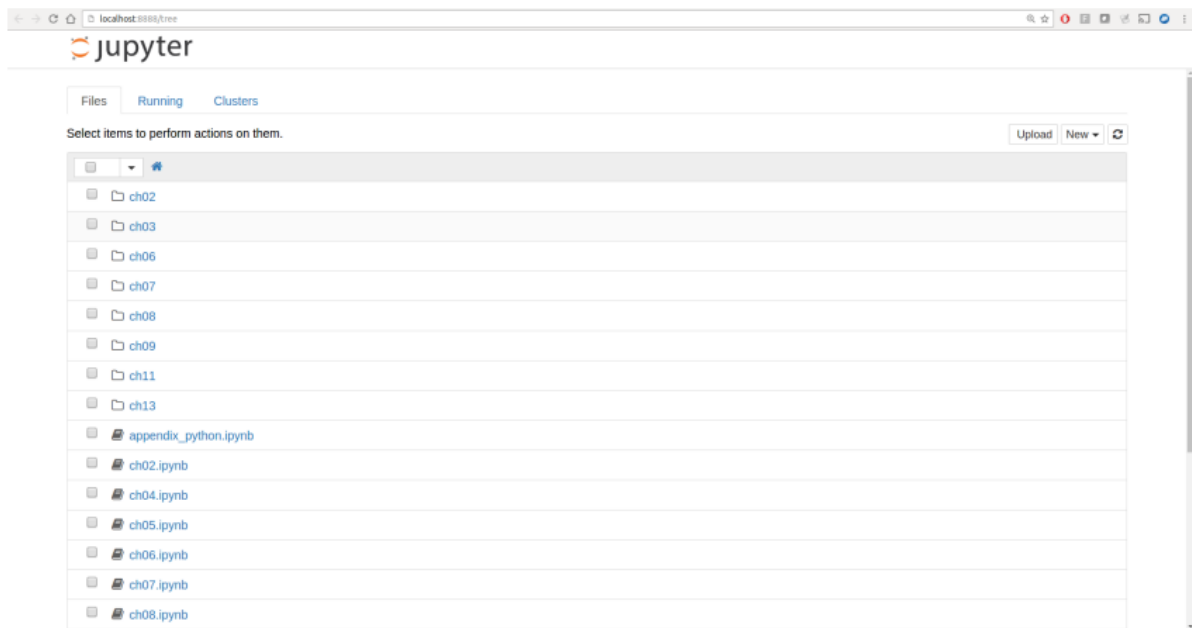
Jupyter projesinin ana bileşenlerinden biri notebook'dur, kod için bir tür etkileşimli belge, metin (işaretili veya işaretsiz), veri görselleştirmeleri ve diğer çıktılarıdır.

Jupyter notebook, Jupyter etkileşimli hesaplama protokolünün herhangi bir sayıda programlama dilinde uygulamaları olan çekirdeklerle etkileşim kurar. Python'un Jupyter çekirdeği, temel davranışı için IPython sistemini kullanır.

Jupyter'i başlatmak için, bir terminalde jupyter notebook komutunu çalıştırın:


```
$ jupyter notebook
[I 15:20:52.739 NotebookApp] Serving notebooks from local directory:
/home/wesm/code/pydata-book
[I 15:20:52.739 NotebookApp] 0 active kernels
[I 15:20:52.739 NotebookApp] The Jupyter Notebook is running at:
http://localhost:8888/
[I 15:20:52.740 NotebookApp] Use Control-C to stop this server and shut down
all kernels (twice to skip confirmation).
Created new window in existing browser session.
```

Pek çok platformda, Jupyter otomatik olarak varsayılan web tarayıcınızda açılır (--no-browser ile başlatmadığınız sürece). Aksi takdirde, not defterini başlattığınızda yazdırılan HTTP adresine gidebilirsiniz, burada <http://localhost:8888/>.



Örnek bir Jupyter Notebook dosyası görünümü;

```
[13]: %pwd

[13]: 'C:\\Users\\recep\\Desktop\\tasarım\\notebooks'

[16]: path = "datasets/bitly_usagov/example.txt"

[18]: open(path).readline()

[18]: '{ "a": "Mozilla\\5.0 (Windows NT 6.1; WOW64) AppleWebKit\\/535.11 (KHTML, like Gecko) Chrome
\\17.0.963.78 Safari\\/535.11", "c": "US", "nk": 1, "tz": "America\\/New_York", "gr": "MA",
"g": "A6qOVH", "h": "wflQtf", "l": "orofrog", "al": "en-US,en;q=0.8", "hh": "1.usa.gov", "r": "h
ttp:\\\\/\\www.facebook.com\\/1\\/7AQEFzjSi\\/1.usa.gov\\/wflQtf", "u": "http:\\\\/\\www.ncbi.nlm.
nih.gov\\/pubmed\\/22415991", "t": 1331923247, "hc": 1331822918, "cy": "Danvers", "ll": [ 42.576
698, -70.954903 ] }\\n'

[19]: import json
path = "datasets/bitly_usagov/example.txt"
records = [json.loads(line) for line in open(path)]

[20]: records[0]

[20]: {'a': 'Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.96
3.78 Safari/535.11',
'c': 'US',
'nk': 1,
'tz': 'America/New_York',
'gr': 'MA',
'g': 'A6qOVH',
'h': 'wflQtf',
'l': 'orofrog',
'al': 'en-US,en;q=0.8',
'hh': '1.usa.gov',
'r': 'http://www.facebook.com/1/7AQEFzjSi/1.usa.gov/wflQtf',
'u': 'http://www.ncbi.nlm.nih.gov/pubmed/22415991',
't': 1331923247,
'hc': 1331822918,
'cy': 'Danvers',
'll': [42.576698, -70.954903]}

[21]: records[0]['tz']

[21]: 'America/New_York'

[22]: print(records[0]['tz'])

America/New_York
```

Tab Completion

Yüzeyde, IPython kabuğu, standart uçbirim Python yorumlayıcısının (python ile çağrılan) kozmetik olarak farklı bir versiyonu gibi görünür. Standart Python kabuğu üzerindeki en büyük iyileştirmelerden biri, birçok IDE'de veya diğer etkileşimli hesaplama analiz ortamlarında bulunan Tab tamamlamadır. Kabuğa ifadeler girerken, Tab tuşuna basmak, şimdiye kadar yazdığınız karakterlerle eşleşen tüm değişkenler (objects, functions vb.) için ad alanında arama yapacaktır:

```
In [1]: an_apple = 27
```

```
In [2]: an_example = 42
```

```
In [3]: an<Tab>
an_apple    and        an_example any
```

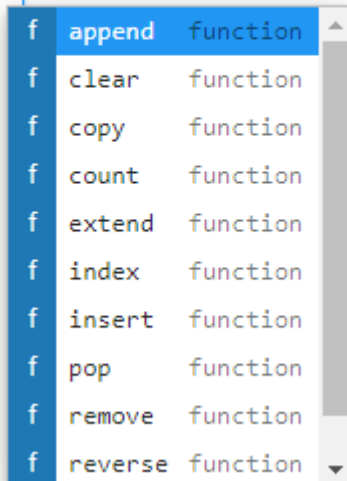
Bu örnekte, IPython'un hem tanımladığım iki değişkeni hem de Python anahtar sözcüğü *and* ve yerleşik *any* işlevini gösterdiğine dikkat edin. Doğal olarak, bir nokta yazdıktan sonra herhangi bir nesne üzerindeki yöntemleri ve nitelikleri de tamamlayabilirsiniz:

```
In [3]: b = [1, 2, 3]
```

```
In [4]: b.<Tab>
b.append  b.count  b.insert  b.reverse
b.clear   b.extend  b.pop     b.sort
b.copy    b.index   b.remove
```

```
[23]: b = [1,2,3]
```

```
[ ]: b.|
```



f	append	function
f	clear	function
f	copy	function
f	count	function
f	extend	function
f	index	function
f	insert	function
f	pop	function
f	remove	function
f	reverse	function

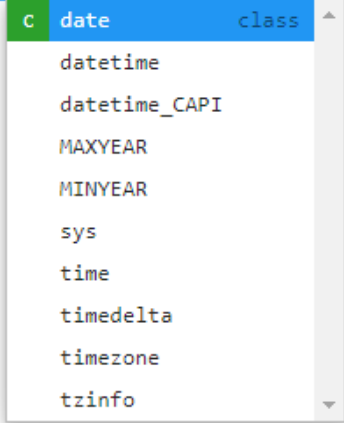
Aynı modüller için de geçerlidir:

```
In [1]: import datetime

In [2]: datetime.<Tab>
datetime.date          datetime.MAXYEAR      datetime.timedelta
datetime.datetime      datetime.MINYEAR      datetime.timezone
datetime.datetime_CAPI datetime.time          datetime.tzinfo

[24]: import datetime

[ ]: datetime.|
```

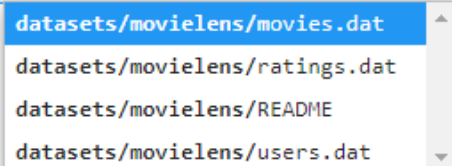


The image shows a Jupyter notebook cell with the code `import datetime` followed by a new cell starting with `datetime.`. An autocomplete dropdown menu is visible, listing attributes of the `datetime` module. The first item, `date`, is highlighted in blue and labeled as a `class`. Other items include `datetime`, `datetime_CAPI`, `MAXYEAR`, `MINYEAR`, `sys`, `time`, `timedelta`, `timezone`, and `tzinfo`.

Jupyter notebook ve IPython'un daha yeni sürümlerinde (5.0 ve üstü), otomatik tamamlamalar metin çıktısı yerine açılır bir kutuda görünür.

Sekme tamamlama, etkileşimli ad alanını arama ve nesne veya modül özelliklerini tamamlamanın dışında birçok bağlamda çalışır. Dosya yoluna benzeyen herhangi bir şey yazarken (bir Python dizisinde bile), Sekme tuşuna basmak bilgisayarınızın dosya sistemindeki her şeyi yazdıklarınızla eşleşen şekilde tamamlayacaktır:

```
[ ]: path = 'datasets/movielens/|'
```

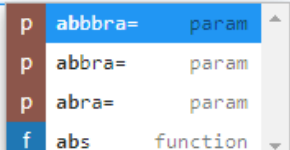


The image shows a Jupyter notebook cell with the code `path = 'datasets/movielens/|'`. An autocomplete dropdown menu is visible, listing files and directories in the `datasets/movielens/` directory. The first item, `datasets/movielens/movies.dat`, is highlighted in blue. Other items include `datasets/movielens/ratings.dat`, `datasets/movielens/README`, and `datasets/movielens/users.dat`.

Sekme tamamlamanın zaman kazandırdığı başka bir alan, anahtar kelime işlevi argümanlarının (ve = sign! dahil) tamamlanmasıdır.

```
[25]: def func_with_keywords(abra=1, abbra=2, abbbra=3):
      return abra, abbra, abbbra

[ ]: func_with_keywords(ab
```



The image shows a Jupyter notebook cell with the code `func_with_keywords(ab`. An autocomplete dropdown menu is visible, listing arguments for the `func_with_keywords` function. The first item, `abbbra=`, is highlighted in blue and labeled as a `param`. Other items include `abbra=` (labeled `param`), `abra=` (labeled `param`), and `abs` (labeled `function`).

Introspection

Bir değişkenden önce veya sonra bir soru işareti (?) Kullanmak, nesne hakkında bazı genel bilgileri görüntüler:

```
[26]: b = [1,2,3]
```

```
[27]: b?
```

```
Type:      list
String form: [1, 2, 3]
Length:    3
Docstring:
Built-in mutable sequence.
```

```
If no argument is given, the constructor creates a new empty list.
The argument must be an iterable if specified.
```

```
[28]: print?
```

```
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file: a file-like object (stream); defaults to the current sys.stdout.
sep:  string inserted between values, default a space.
end:  string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

Bu, nesne iç gözlemi(object introspection) olarak adlandırılır. Nesne bir function veya instance method ise, tanımlanmışsa, docstring de gösterilecektir. Aşağıdaki işlevi yazdığımızı varsayalım (IPython veya Jupyter'de çoğaltabilirsiniz):

```
[29]: def add_numbers(a, b):
      """
      Add two numbers together
      Returns
      -----
      the_sum : type of arguments
      """
      return a + b
```

```
[30]: add_numbers?
```

```
Signature: add_numbers(a, b)
Docstring:
Add two numbers together
Returns
-----
the_sum : type of arguments
File:    c:\users\recep\desktop\tasarım\notebooks\<ipython-input-29-5447cfd50127>
Type:    function
```

```
[31]: add_numbers??

Signature: add_numbers(a, b)
Source:
def add_numbers(a, b):
    """
    Add two numbers together
    Returns
    -----
    the_sum : type of arguments
    """
    return a + b
File:      c:\users\recep\desktop\tasarım\notebooks\<ipython-input-29-5447cfd50127>
Type:      function
```

? IPython ad alanını standart Unix veya Windows komut satırına benzer bir şekilde aramak için son bir kullanıma sahiptir. Joker karakterle (*) birleştirilen birkaç karakter, joker ifadesiyle eşleşen tüm adları gösterecektir. Örneğin, en üst düzey NumPy ad alanındaki load içeren tüm işlevlerin bir listesini alabiliriz:

```
[32]: np.*load*?

np.__loader__
np.load
np.loads
np.loadtxt
```

The %run Command

Herhangi bir dosyayı %run komutunu kullanarak IPython oturumunuzun ortamında bir Python programı olarak çalıştırabilirsiniz. *ipython_script_test.py*'de aşağıdaki basit komut dosyasının depolandığını varsayalım:

```
def f(x, y, z):
    return (x + y) / z

a = 5
b = 6
c = 7.5

result = f(a, b, c)
```

Dosya adını %run'a ileterek bunu yürütebilirsiniz.

```
[1]: %run ipython_script_test.py
```

Komut dosyası boş bir namespace'de çalıştırılır (import'lar veya diğer değişkenler tanımlanmadan), böylece davranış, programı python script.py kullanarak komut satırında çalıştırmakla aynı olmalıdır. Dosyada tanımlanan tüm değişkenler (import'lar, fonksiyon'lar ve global'ler) (varsa bir istisna oluşana kadar) IPython kabuğundan erişilebilir olacaktır:

```
[2]: c
```

```
[2]: 7.5
```

```
[3]: result
```

```
[3]: 1.4666666666666666
```

Bir Python betiği komut satırı bağımsız değişkenlerini beklerse (sys.argv'de bulunur), bunlar komut satırında çalıştırılmış gibi dosya yolundan sonra geçirilebilir.

Not: Etkileşimli IPython namespace'den önceden tanımlanmış değişkenlere bir komut dosyası erişimi vermek isterseniz, düz **%run** yerine **%run -i** kullanın.

Jupyter not defterinde, bir komut dosyasını bir kod hücresine aktaran ilgili **%load** magic işlevini de kullanabilirsiniz:

```
[4]: # %load ipython_script_test.py
def f(x, y, z):
    →return (x + y) / z

a = 5
b = 6
c = 7.5
result = f(a, b, c)
```

Interrupting Running Code

Herhangi bir kod çalışırken Ctrl-C tuşlarına basmak, ister %run üzerinden bir komut dosyası isterse uzun süreli bir komut olsun, bir KeyboardInterrupt'ın yükseltilmesine neden olur. Bu, bazı olağandışı durumlar dışında neredeyse tüm Python programlarının hemen durmasına neden olur.

Not: Bir Python kodu parçası bazı derlenmiş uzantı modüllerini çağırdığında, Ctrl-C'ye basmak her zaman program yürütülmesinin hemen durmasına neden olmaz. Bu gibi durumlarda, ya kontrolün Python yorumlayıcısına geri dönmesini beklemeniz gerekecek ya da daha kötü durumlarda Python sürecini zorla sonlandırmalısınız.

Executing Code from the Clipboard

Jupyter not defterini kullanıyorsanız, kodu herhangi bir kod hücresine kopyalayıp yapıştırabilir ve çalıştırabilirsiniz. IPython kabuğundaki panodan kod çalıştırmak da mümkündür. Başka bir uygulamada aşağıdaki koda sahip olduğunuzu varsayalım:

```
[6]: x = 5
y = 7
if x > 5:
    x += 1

y = 8
```

En kusursuz yöntemler, **%paste** ve **%cpaste** magic işlevleridir. **%paste** panodaki metni alır ve onu kabukta tek bir blok olarak yürütür:

```
In [17]: %paste
x = 5
y = 7
if x > 5:
    x += 1

    y = 8
## -- End pasted text --
```

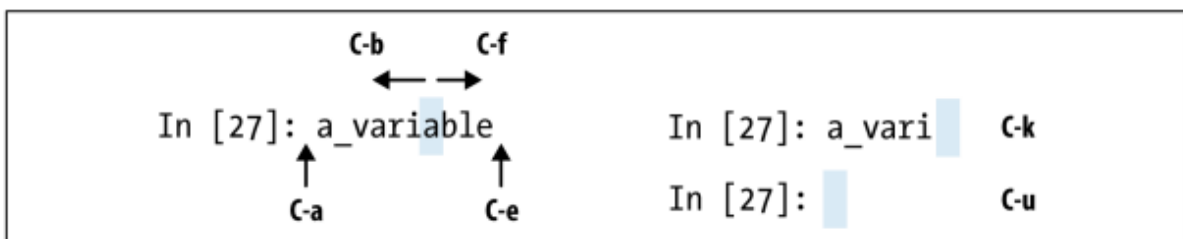
%cpaste, kodu şuraya yapıştırmanız için size özel bir uyarı vermesi dışında benzerdir:

```
In [18]: %cpaste
Pasting code; enter '--' alone on the line to stop or use Ctrl-D.
:x = 5
:y = 7
:if x > 5:
:    x += 1
:
:
:    y = 8
:--
```

Not: %paste magic fonksiyonu sadece IPython'un terminal sürümü için tanımlanmıştır, grafiksel ön uçları (yani notebook ve qtconsole) için değil, çünkü buna ihtiyaç duymazlar.

%cpaste bloğu ile, çalıştırmadan önce istediğiniz kadar çok kod yapıştırma özgürlüğüne sahipsiniz. Yapıştırılan koda, çalıştırmadan önce bakmak için %cpaste kullanmaya karar verebilirsiniz. Yanlışlıkla yanlış kodu yapıştırırsanız, Ctrl-C tuşlarına basarak %cpaste isteminden çıkabilirsiniz.

Terminal Keyboard Shortcuts



Keyboard shortcut	Description
Ctrl-P or up-arrow	Search backward in command history for commands starting with currently entered text
Ctrl-N or down-arrow	Search forward in command history for commands starting with currently entered text
Ctrl-R	Readline-style reverse history search (partial matching)
Ctrl-Shift-V	Paste text from clipboard
Ctrl-C	Interrupt currently executing code
Ctrl-A	Move cursor to beginning of line
Ctrl-E	Move cursor to end of line
Ctrl-K	Delete text from cursor until end of line
Ctrl-U	Discard all text on current line
Ctrl-F	Move cursor forward one character
Ctrl-B	Move cursor back one character
Ctrl-L	Clear screen

About Magic Commands

IPython'un özel komutları (Python'da yerleşik değildir) "magic" komutlar olarak bilinir. Bunlar, ortak görevleri kolaylaştırmak ve IPython sisteminin davranışını kolayca kontrol etmenizi sağlamak için tasarlanmıştır. Bir magic komut, % yüzde sembolü ile başlayan herhangi bir komuttur. Örneğin, matris çarpımı gibi herhangi bir Python ifadesinin yürütme zamanını **%timeit** magic işlevini kullanarak kontrol edebilirsiniz (daha sonra daha ayrıntılı olarak tartışılacaktır):

```
[9]: import numpy as np
a = np.random.randn(100, 100)

[10]: %timeit np.dot(a, a)

115 µs ± 1.19 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Magic komutlar, IPython sistemi içinde çalıştırılacak komut satırı programları olarak görülebilir. Birçoğunun ek "komut satırı" seçenekleri vardır, bunların tümü (beklediğiniz gibi) "?" kullanılarak görüntülenebilir:

```
[11]: %debug?
Docstring:
::

    %debug [--breakpoint FILE:LINE] [statement [statement ...]]

Activate the interactive debugger.

This magic command support two ways of activating debugger.
One is to activate debugger before executing code. This way, you
can set a break point, to step through the code from the point.
You can use this mode by giving statements to execute and optionally
a breakpoint.

The other one is to activate debugger in post-mortem mode. You can
activate this mode simply running %debug without any argument.
If an exception has just occurred, this lets you inspect its stack
frames interactively. Note that this will always work only on the last
traceback that occurred, so you must call this quickly after an
exception that you wish to inspect has fired, because if another one
occurs, it clobbers the previous one.

If you want IPython to automatically do this on every exception, see
the %pdb magic for more details.

.. versionchanged:: 7.3
    When running code, user variables are no longer expanded,
    the magic line is always left unmodified.

positional arguments:
  statement          Code to run in debugger. You can omit this in cell magic mode.

optional arguments:
  --breakpoint <FILE:LINE>, -b <FILE:LINE>
                        Set break point at LINE in FILE.
File: c:\users\recep\appdata\local\programs\python\python39\python.exe
```

Söz konusu magic işlevle aynı ada sahip hiçbir değişken tanımlanmadığı sürece, magic işlevler varsayılan olarak yüzde işareti olmadan kullanılabilir. Bu özelliğe automagic denir ve %automagic ile etkinleştirilebilir veya devre dışı bırakılabilir.

Bazı magic işlevler Python işlevleri gibi davranır ve çıktıları bir değişkene atanabilir:

```
[12]: %pwd
[12]: 'D:\\PC Yedek\\Belgeler\\Çalışmalar\\Tasarım-Bitirme\\notebooks'

[13]: foo = %pwd
[14]: foo
[14]: 'D:\\PC Yedek\\Belgeler\\Çalışmalar\\Tasarım-Bitirme\\notebooks'
```

Command	Description
<code>%quickref</code>	Display the IPython Quick Reference Card
<code>%magic</code>	Display detailed documentation for all of the available magic commands
<code>%debug</code>	Enter the interactive debugger at the bottom of the last exception traceback
<code>%hist</code>	Print command input (and optionally output) history
<code>%pdb</code>	Automatically enter debugger after any exception
<code>%paste</code>	Execute preformatted Python code from clipboard
<code>%cpaste</code>	Open a special prompt for manually pasting Python code to be executed
<code>%reset</code>	Delete all variables/names defined in interactive namespace
<code>%page OBJECT</code>	Pretty-print the object and display it through a pager
<code>%run script.py</code>	Run a Python script inside IPython
<code>%prun statement</code>	Execute <i>statement</i> with <code>cProfile</code> and report the profiler output
<code>%time statement</code>	Report the execution time of a single statement
<code>%timeit statement</code>	Run a statement multiple times to compute an ensemble average execution time; useful for timing code with very short execution time
<code>%who, %who_ls, %whos</code>	Display variables defined in interactive namespace, with varying levels of information/verbosity
<code>%xdel variable</code>	Delete a variable and attempt to clear any references to the object in the IPython internals

Matplotlib Integration

IPython'un analitik hesaplamadaki popüleritesinin bir nedeni, veri görselleştirme ve matplotlib gibi diğer kullanıcı arayüzü kitaplıkları ile iyi entegre olmasıdır.

`%matplotlib` magic işlevi IPython kabuğu veya Jupyter notebook ile entegrasyonunu yapılandırır. Bu önemlidir, çünkü aksi halde oluşturduğunuz grafikler görünmez (notebook) veya kapanana kadar (kabuk) oturumun kontrolünü ele geçirmez.

IPython kabuğunda, `%matplotlib`'i çalıştırmak, entegrasyonu kurar, böylece konsol oturumuna müdahale etmeden birden çok çizim penceresi oluşturabilirsiniz:

```
In [26]: %matplotlib
Using matplotlib backend: Qt4Agg
```

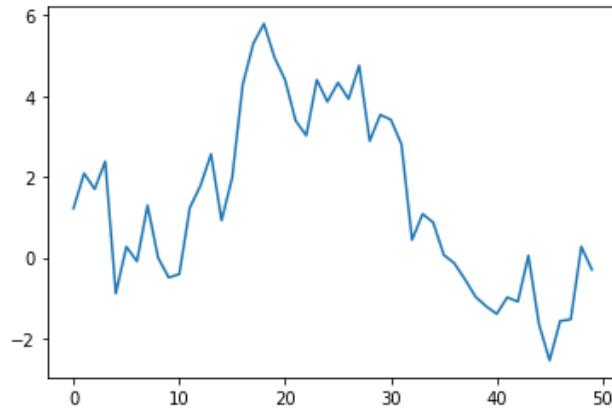
Jupyter'de komut biraz farklı:

```
[18]: %matplotlib inline
```

Matplotlib is building the font cache; this may take a moment.

```
[19]: import matplotlib.pyplot as plt  
plt.plot(np.random.randn(50).cumsum())
```

```
[19]: [<matplotlib.lines.Line2D at 0x16a40835310>]
```



2.3 Python Language Basics