Introduction to Python

INTRODUCTION

Computational tools for data science/analytics

Menu based

- MS Excel
- Tableau
- MS Power BI
- JMP

Programming languages

- Python
- R
- SQL
- Matlab

INTRODUCTION

Best programming languages for data science/analytics

INTRODUCTION

Why?

- Both include a large number of libraries for data analytics, data visualization, ML/SL, web scraping, text analytics, deep learning.
- With these libraries, applications can be developed in a very efficient way

Python

Python is a computer language suitable for

- Data Analysis
- Data Visualization
- Machine learning
- App development
- Game development
- Web development
- Front/Back/Full stack
- Artificial intelligence
- scripting

Python

Pros

- Open-source languages
- Thousands of libraries
- Available for Linux and MS Windows OS

Cons

- Developers may update the libraries without notice

INTRODUCTION

We will introduce the Python language and will develop predictive analytics applications with different libraries

INTRODUCTION – The Python ecosystem

A collection of

- Python language
- User interfaces
- libraries

INTRODUCTION – The Python ecosystem

A collection of

- Python language –version 3.6+
- User interfaces (Jupyter Notebook)
- libraries

INTRODUCTION – The Python ecosystem

A collection of

- Python language –version 3.6+
- User interfaces (Jupyter Notebook)
- libraries

Use a bundled Python distribution: Anaconda

Anaconda includes Python, Jupyter Notebook, and most libraries we will need

INTRODUCTION – Installing Python

Anaconda comes with over 100 Python libraries

Search for Anaconda download or visit

https://docs.anaconda.com/anaconda/install/

INTRODUCTION – Installing Python

Visit https://docs.anaconda.com/anaconda/install/

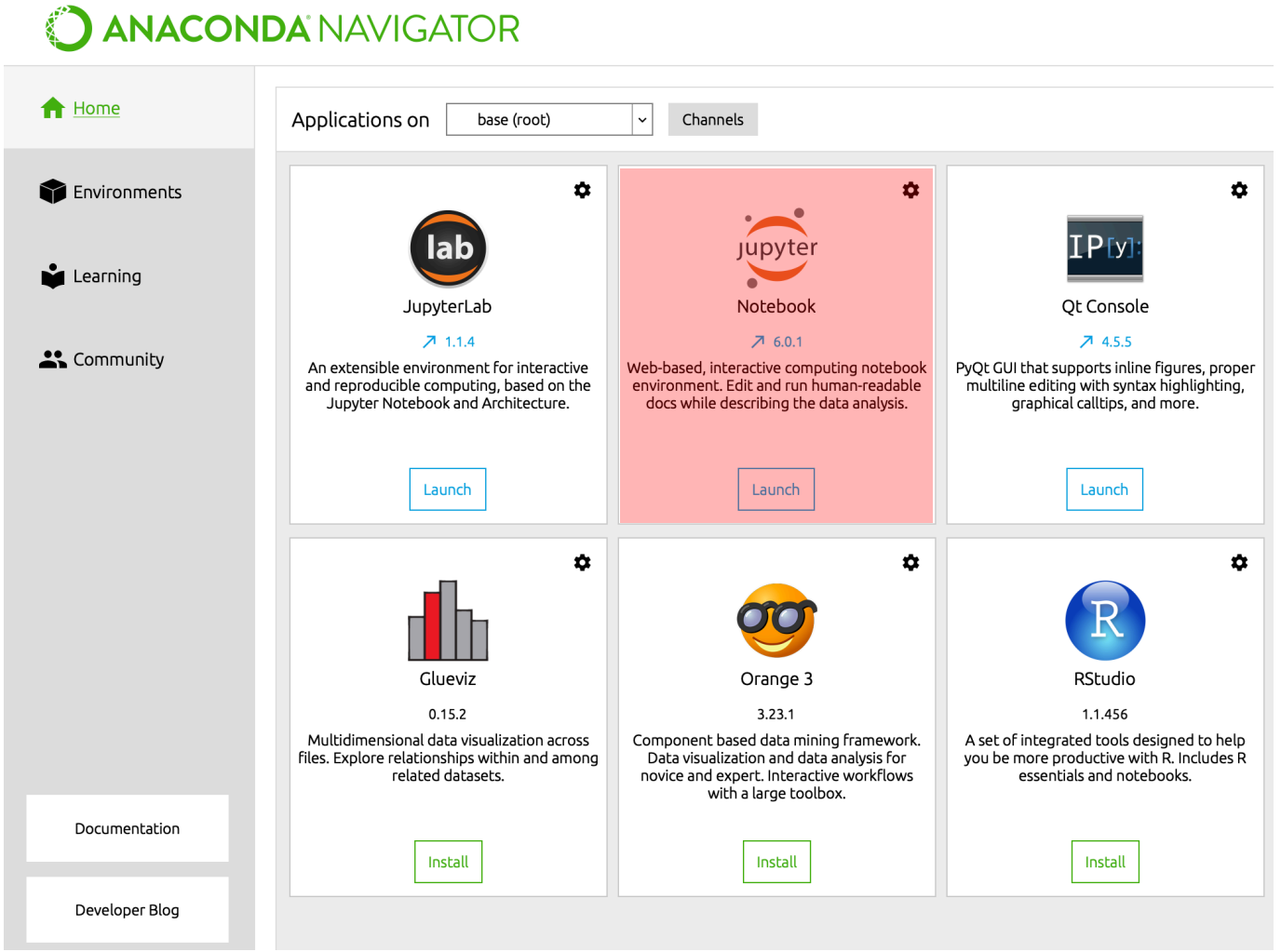
- Select the link for your OS (Mac or Windows)
- Open installer and follow the steps

If you have trouble

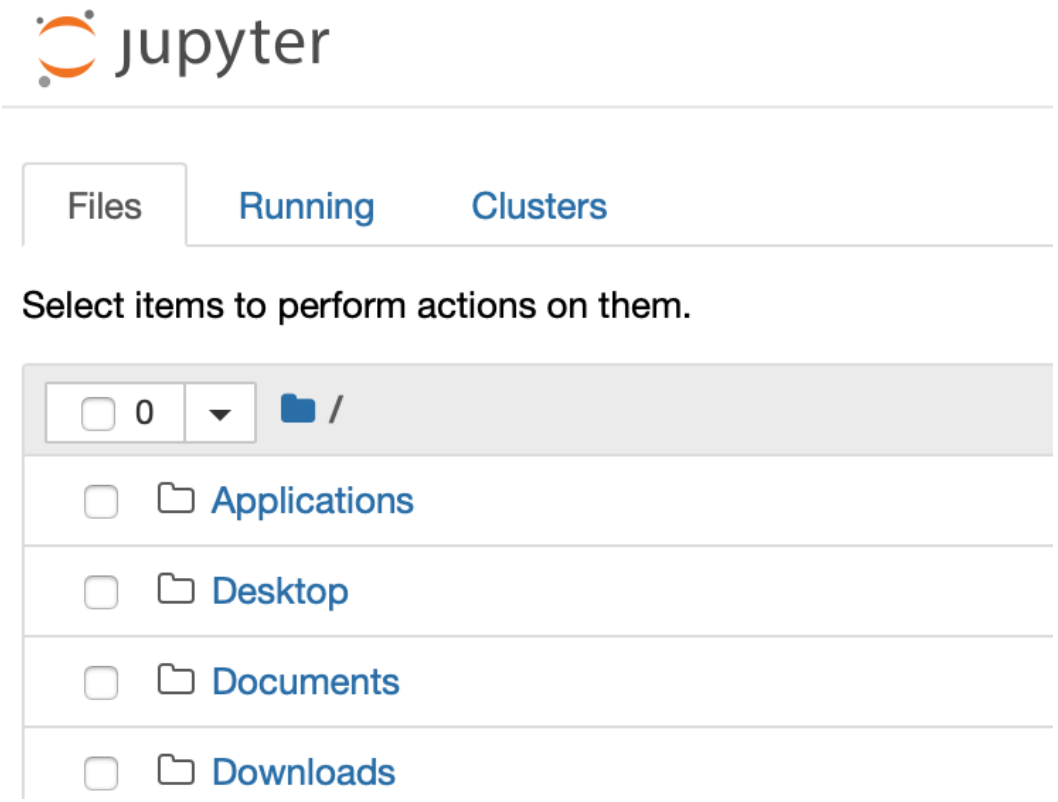
https://docs.anaconda.com/anaconda/user-guide/troubleshooting/

ANACONDA


Other
Anaconda-Navigator


ANACONDA NAVIGATOR
Home
Environments
Learning
Community
Documentation
Developer Blog
Applications on
base (root)
Channels
JupyterLab
1.1.4
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch
Notebook
6.0.1
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
Qt Console
4.5.5
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Launch
Glueviz
0.15.2
Multidimensional data visualization across files. Explore relationships within and among related datasets.
Install
Orange 3
3.23.1
Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.
Install
RStudio
1.1.456
A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.
Install

ANACONDA


jupyter
Files
Running
Clusters
Select items to perform actions on them.
0
/
Applications
Desktop
Documents
Downloads

Jupyter Notebook

INTRODUCTION – Jupyter notebook

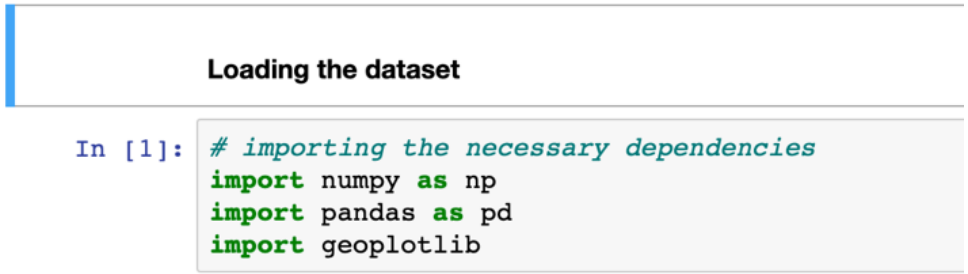
- A web application for writing code, get the results in-line, add Markdown text
- Text in the file is used to document the code
- file extension is .*ipynb*
- file extension for plain python code is .*py*

INTRODUCTION – Jupyter notebook

INTRODUCTION – Jupyter notebook

Keyboard Input modes

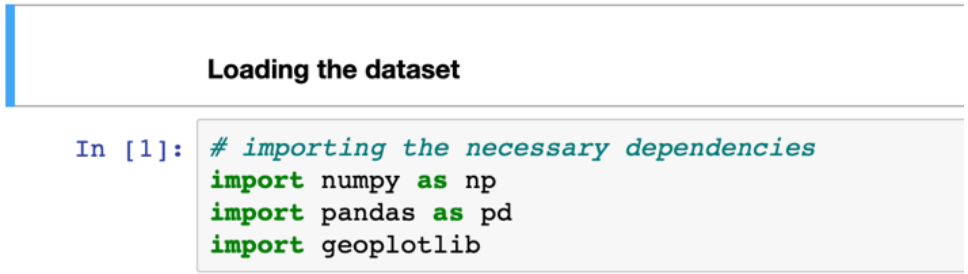
- Edit mode (green ribbon)
- Command mode (blue ribbon)



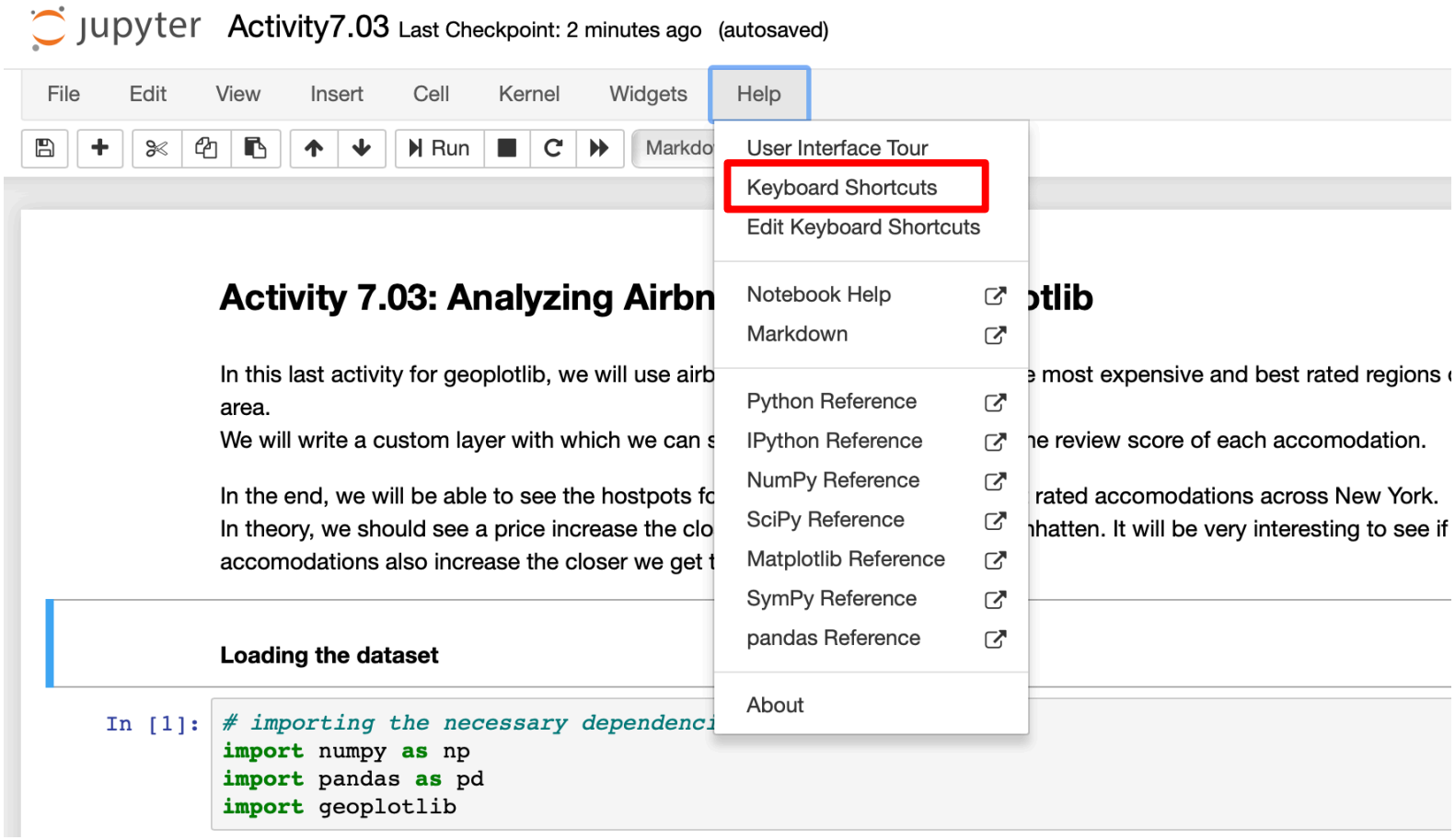
INTRODUCTION – Jupyter notebook

Keyboard Input modes

- Edit mode (Esc to go to command mode)
- Command mode (↵ to go to edit mode)



INTRODUCTION – Jupyter notebook



see JN shortcuts.pdf

INTRODUCTION – Jupyter notebook

Cross Tabulation

markdown text

```
# number of cars by DriveTrain
```

comment

```
pd.value_counts(df.DriveTrain)
```

Python command

```
Front    63
Rear     14
4WD       5
Name: DriveTrain, dtype: int64
```

Output

```
# number of cars by DriveTrain and Airbags
```

```
pd.crosstab(df.DriveTrain,df.AirBags)
```

AirBags DriveTrain	Driver & Passenger	Driver only	None
4WD	0	2	3
Front	11	28	24
Rear	5	8	1

Python libraries

INTRODUCTION – Keyboard symbols

:	colon
;	semicolon
~	tilde
&	ampersand
-	dash
_	underscore
\	backslash

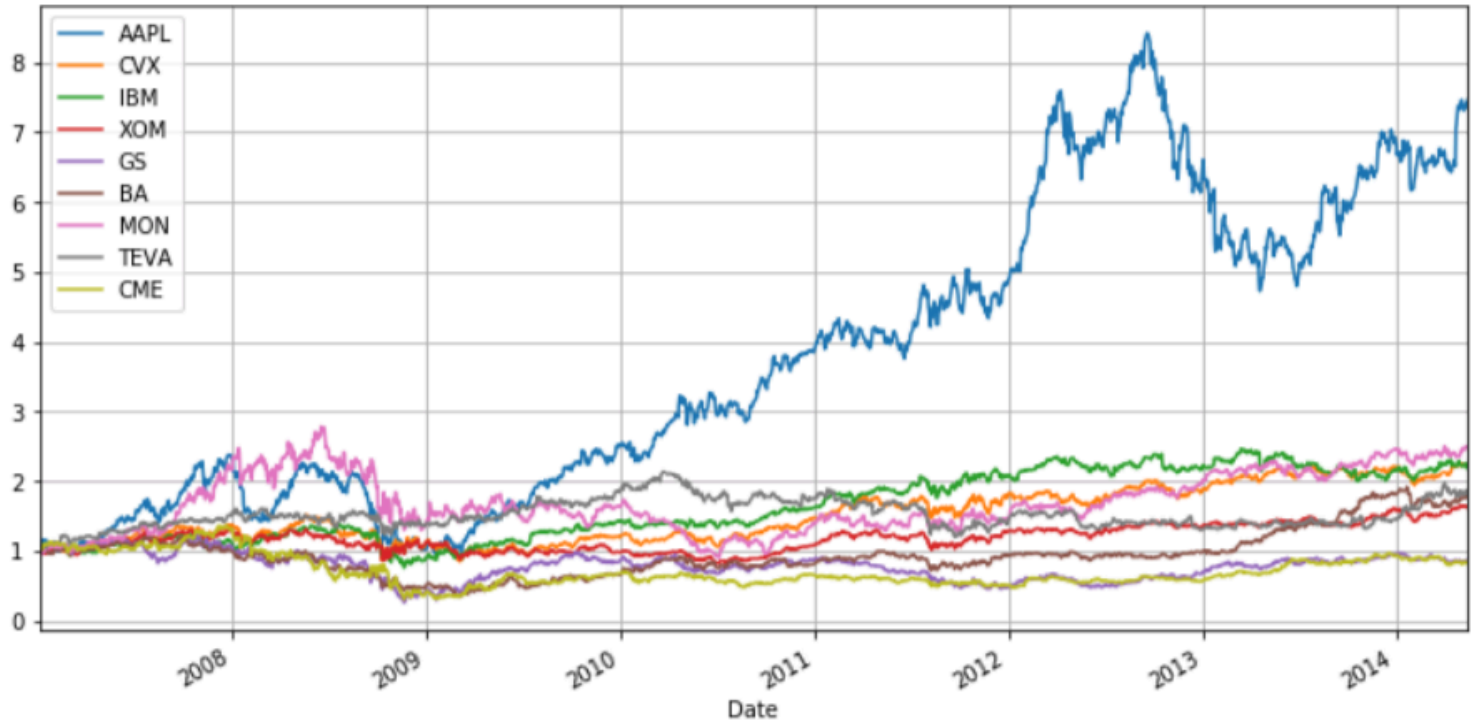
INTRODUCTION – Python libraries

Why do we need libraries?

Libraries allow users
to develop applications
without having to code low-level details

INTRODUCTION

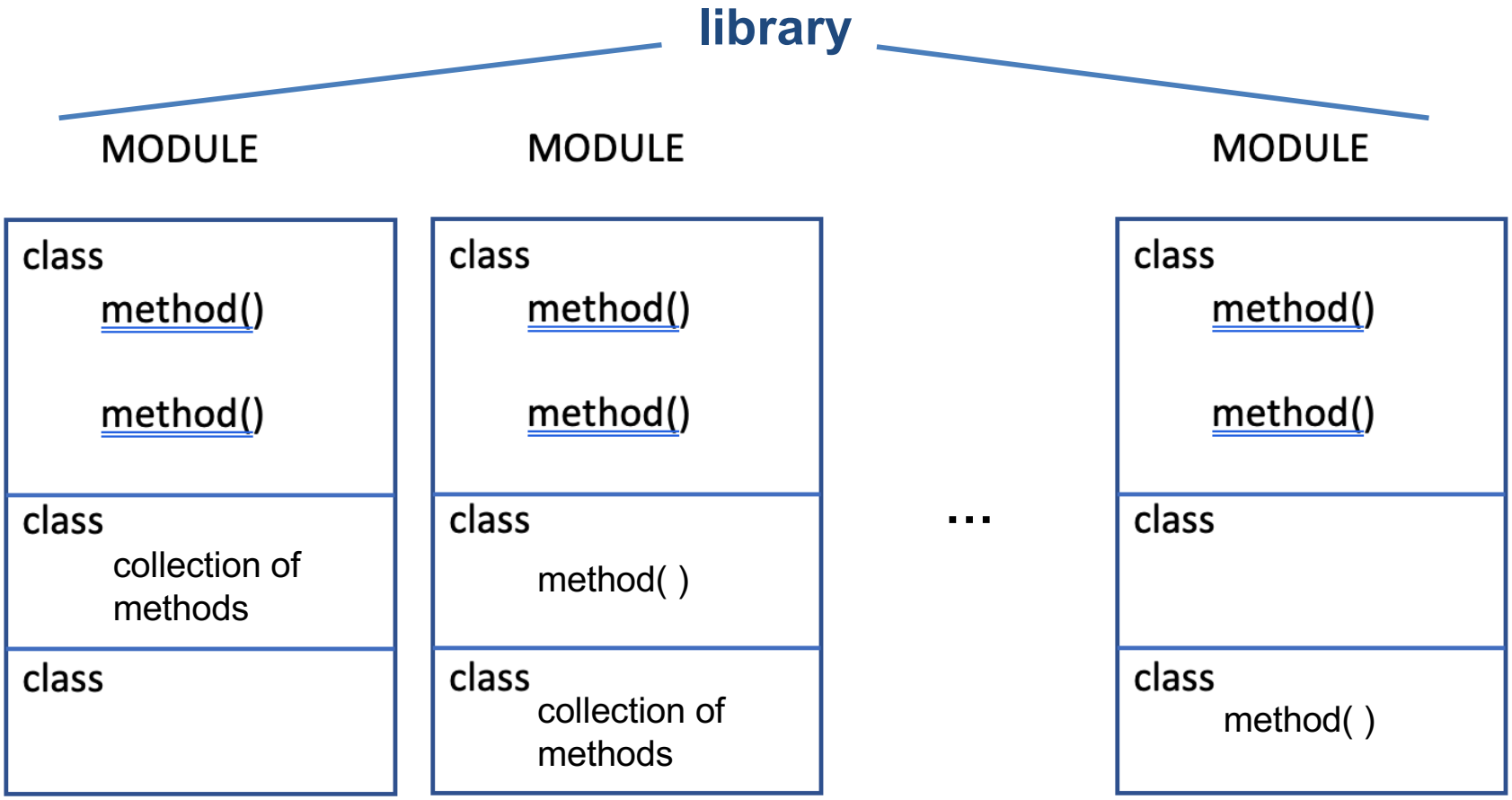
In [36]:
```
gross_returns.plot(figsize=(12,6))
plt.grid()
```



INTRODUCTION – library

- A collection of high-level functions
- Used to perform data operations without the need to write detailed code
- A library is a collection of *Modules*
- *Modules* are made of *Classes*
- *Classes* include *Methods* (functions)

INTRODUCTION – a Python library



INTRODUCTION – Python library notation

- library
- library.module
- library.module.class
- library.module.class.method()

Python Objects

- Python is an object-oriented programming (OOP) language
- In Python, everything is an object
- Every object has *attributes*
- *Methods* can be applied to an object via the dot syntax

INTRODUCTION – Python libraries

Python library

- numpy
- pandas
- matplotlib
- statsmodels
- scikit-learn

INTRODUCTION – Python libraries

Python library	functions
• numpy	for working with arrays
• pandas	for working with data sets
• matplotlib	for plotting
• statsmodels	for statistical modeling
• scikit-learn	for machine learning

INTRODUCTION – importing a library

```
import numpy as np
```

library name

alias

Import all modules from the library

INTRODUCTION – importing a module

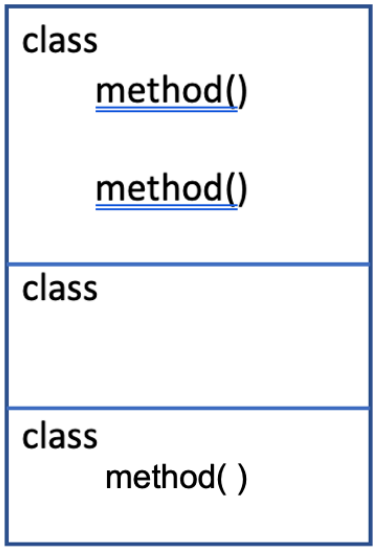
```
import numpy as np
```

library name alias

```
import matplotlib.pyplot as plt
```

library name module alias

Import just one Module



INTRODUCTION – importing a class

```
from sklearn.model_selection  import kfold
```

library name — module — class

INTRODUCTION – importing and using a class

```
from sklearn.model_selection  import kfold
```

library name — module — class

```
kfold.                        ( ..., ..., ...)
```

method — arguments

INTRODUCTION – Python example

```
In [1]: import numpy as np

In [2]: import statsmodels.api as sm

In [3]: import statsmodels.formula.api as smf

# Load data
In [4]: dat = sm.datasets.get_rdataset("Guerry", "HistData").data

# Fit regression model (using the natural log of one of the regressors)
In [5]: results = smf.ols('Lottery ~ Literacy + np.log(Pop1831)', data=dat).fit()

# Inspect the results
In [6]: print(results.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                Lottery   R-squared:                       0.348
Model:                            OLS   Adj. R-squared:                  0.333
Method:                 Least Squares   F-statistic:                     22.20
Date:                Fri, 21 Feb 2020   Prob (F-statistic):           1.90e-08
Time:                        13:59:15   Log-Likelihood:                -379.82
No. Observations:                  86   AIC:                             765.6
```

INTRODUCTION – Python example

```
In [1]: import numpy as np

In [2]: import statsmodels.api as sm

In [3]: import statsmodels.formula.api as smf

# Load data
In [4]: dat = sm.datasets.get_rdataset("Guerry", "HistData").data

# Fit regression model (using the natural log of one of the regressors)
In [5]: results = smf.ols('Lottery ~ Literacy + np.log(Pop1831)', data=dat).fit()

# Inspect the results
In [6]: print(results.summary())
                            OLS Regression Results
==============================================================================
Dep. Variable:                Lottery   R-squared:                       0.348
Model:                            OLS   Adj. R-squared:                  0.333
Method:                 Least Squares   F-statistic:                     22.20
Date:                Fri, 21 Feb 2020   Prob (F-statistic):           1.90e-08
Time:                        13:59:15   Log-Likelihood:                -379.82
No. Observations:                  86   AIC:                             765.6
```

Python Data Structures

INTRODUCTION

Data Structures p30

- List [a,b,a,c] collection of items
- set {a,b,c} collection unique items
- tuple (a,b,c) immutable collection
- dictionary {key_1:val_1, key_2:val_2, …} pairs

LIST

A list is an *ordered* collection of objects

- x = [1, 7, 8, 3, 7]
- y = [7, 1, 3, 7, 8]

Objects can be extracted by their positional index

The index starts at position 0

- x[0] = 1
- x[1] = 7
- y[0] = 7
- y[2] = 3

INTRODUCTION – slicing a list

```
x = [1,3,5,8,2,4]
```

```
x
```

```
[1, 3, 5, 8, 2, 4]
```

show first 4

```
x[:4]
```

```
[1, 3, 5, 8]
```

show all beyond the first 4

```
x[4:]
```

```
[2, 4]
```

INTRODUCTION – slicing a list

```
x = [1,3,5,8,2,4]
```

```
x
```

```
[1, 3, 5, 8, 2, 4]
```

show first 4

```
x[:4]
```

```
[1, 3, 5, 8]
```

show all beyond the first 4

```
x[4:]
```

```
[2, 4]
```

```
x[1:3]
```

```
[3, 5]
```

show items with index 1 and 2

```
x[-1]
```

```
4
```

show last item

```
x[:-1]
```

```
[1, 3, 5, 8, 2]
```

show all but not the last one

INTRODUCTION – functions for lists

`append(x)` adds `x` to the end of the list

`count(x)` counts how many times `x` appears in the list

`extend(L)` adds the elements in list `L` to the end of the original list

`index(x)` returns the index of the first element of the list to match `x`

`insert(i, x)` inserts element `x` at location `i` in the list, moving everything else along

`pop(i)` removes the item at index `i`

`remove(x)` deletes the first element that matches `x`

`reverse()` reverses the order of the list

`sort()` we've already seen

All these functions work: in-place

INTRODUCTION – lists

```
x = [1,3,5,8,2,4]
```

```
x.append(9)
x
```

```
[1, 3, 5, 8, 2, 4, 9]
```

```
L = [0,5,8]
```

```
x.extend(L)
x
```

```
[1, 3, 5, 8, 2, 4, 9, 0, 5, 8]
```

```
x.count(5)
```

```
2
```

INTRODUCTION – lists

```
x = [1,3,5,8,2,4]
```

```
x.append(9)
x
```

```
[1, 3, 5, 8, 2, 4, 9]
```

```
L = [0,5,8]
```

```
x.extend(L)
x
```

```
[1, 3, 5, 8, 2, 4, 9, 0, 5, 8]
```

```
x.count(5)
```

```
2
```

```
# index of 8 (first time)
```

```
x.index(8)
```

```
3
```

```
# insert 6 in position 3
```

```
x.insert(3,6)
x
```

```
[1, 3, 5, 6, 8, 2, 4, 9, 0, 5, 8]
```

```
# deletes item in position 3
```

```
x.pop(3)
x
```

```
[1, 3, 5, 8, 2, 4, 9, 0, 5, 8]
```

INTRODUCTION – lists

```
x
```

```
[1, 3, 5, 8, 2, 4, 9, 0, 5, 8]
```

```
# remove 8 (first time only)
```

```
x.remove(8)
x
```

```
[1, 3, 5, 2, 4, 9, 0, 5, 8]
```

```
# reverse list x
```

```
x.reverse()
x
```

```
[8, 5, 0, 9, 4, 2, 5, 3, 1]
```

INTRODUCTION – lists

```
x
```

```
[1, 3, 5, 8, 2, 4, 9, 0, 5, 8]
```

```
# remove 8 (first time only)
```

```
x.remove(8)
x
```

```
[1, 3, 5, 2, 4, 9, 0, 5, 8]
```

```
# reverse list x
```

```
x.reverse()
x
```

```
[8, 5, 0, 9, 4, 2, 5, 3, 1]
```

```
# duplicate list x
```

```
y = x.copy()
y
```

```
[8, 5, 0, 9, 4, 2, 5, 3, 1]
```

```
x.sort()
x
```

```
[0, 1, 2, 3, 4, 5, 5, 8, 9]
```

INTRODUCTION

Python constructs p55-56

- iterator
- enumerate
- zip

INTRODUCTION – Creating lists

```
In [3]: # range is an iterator (no elements in it)

In [4]: range(10)
Out[4]: range(0, 10)

In [5]: range(0,10)
Out[5]: range(0, 10)

In [6]: # create a list using iterator

In [7]: list(range(0,10))
Out[7]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

INTRODUCTION – Creating lists

```
In [6]: # create a list using iterator

In [7]: list(range(0,10))
Out[7]: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [3]: a = list(range(3,9))
        a
Out[3]: [3, 4, 5, 6, 7, 8]

In [5]: a[1]
Out[5]: 4

In [6]: # index starts at 0
```

INTRODUCTION – Creating lists

using for loop

```
L = []    # empty list
```

```
for n in range(12):
    L.append(n**2)
```

```
L
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121]
```

INTRODUCTION – Creating lists

```
L = []   # empty list
```

using for loop

```
for n in range(12):
    L.append(n**2)
```

```
L
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121]
```

using list comprehension

```
L=[i**2 for i in range(12)]
```

```
L
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121]
```

INTRODUCTION – Creating lists

using for loop

```
for n in range(12):
    L.append(n**2)
```

using list comprehension

```
L=[i**2 for i in range(12)]
```

INTRODUCTION – Creating lists

```
a = [i for i in range(20)]
a
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
# multiples of 3
```

```
a = [i for i in range(20) if i%3 == 0]
a
```

```
[0, 3, 6, 9, 12, 15, 18]
```

INTRODUCTION – Creating lists

```
a = [i for i in range(20)]
a
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

```
# multiples of 3
```

reminder

```
a = [i for i in range(20) if i%3 == 0]
a
```

```
[0, 3, 6, 9, 12, 15, 18]
```

INTRODUCTION – Arithmetic Operations

Operator		Description
+	add	Addition (e.g., 1 + 1 = 2)
-	subtract	Subtraction (e.g., 3 - 2 = 1)
*	multiply	Multiplication (e.g., 2 * 3 = 6)
/	divide	Division (e.g., 3 / 2 = 1.5)
//	floor_divide	Floor division (e.g., 3 // 2 = 1)
**	power	Exponentiation (e.g., 2 ** 3 = 8)
%	mod	Modulus/remainder (e.g., 9 % 4 = 1)

INTRODUCTION – Creating lists

```
list with 5 random integers
```

```
import random
```

```
random.seed(0)
```

```
x = [random.randint (0, 100) for i in range(5)]
x
```

```
[49, 97, 53, 5, 33]
```

FUNCTION

```
def f(x):
    return x**2
```

$y = x^2$

```
f(2.5)
```

```
6.25
```

LAMBDA FUNCTION

```
def f(x):
    return x**2
```

$y = x^2$

```
f(2.5)
```

```
6.25
```

```
# lambda function
```

```
g = lambda x:x**2
```

$y = x^2$

```
g(2.5)
```

```
6.25
```

INTRODUCTION – Apply a function to many values

```
def f(x):
    return x**2
```

```
f(2.5)
```

```
6.25
```

```
# lambda function
```

```
g = lambda x:x**2
```

```
g(2.5)
```

```
6.25
```

map

```
f = lambda x:2*x
```

y = 2x

```
f(2.5)
```

```
5.0
```

```
list(map(f,range(10)))
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

SET

A set is an *unordered* collection of unique objects

```
x = {1,2,4,5,2,5}
x
```

```
{1, 2, 4, 5}
```

(no duplicates)

```
type(x)
```

```
set
```

sets do not support indexing or slicing