

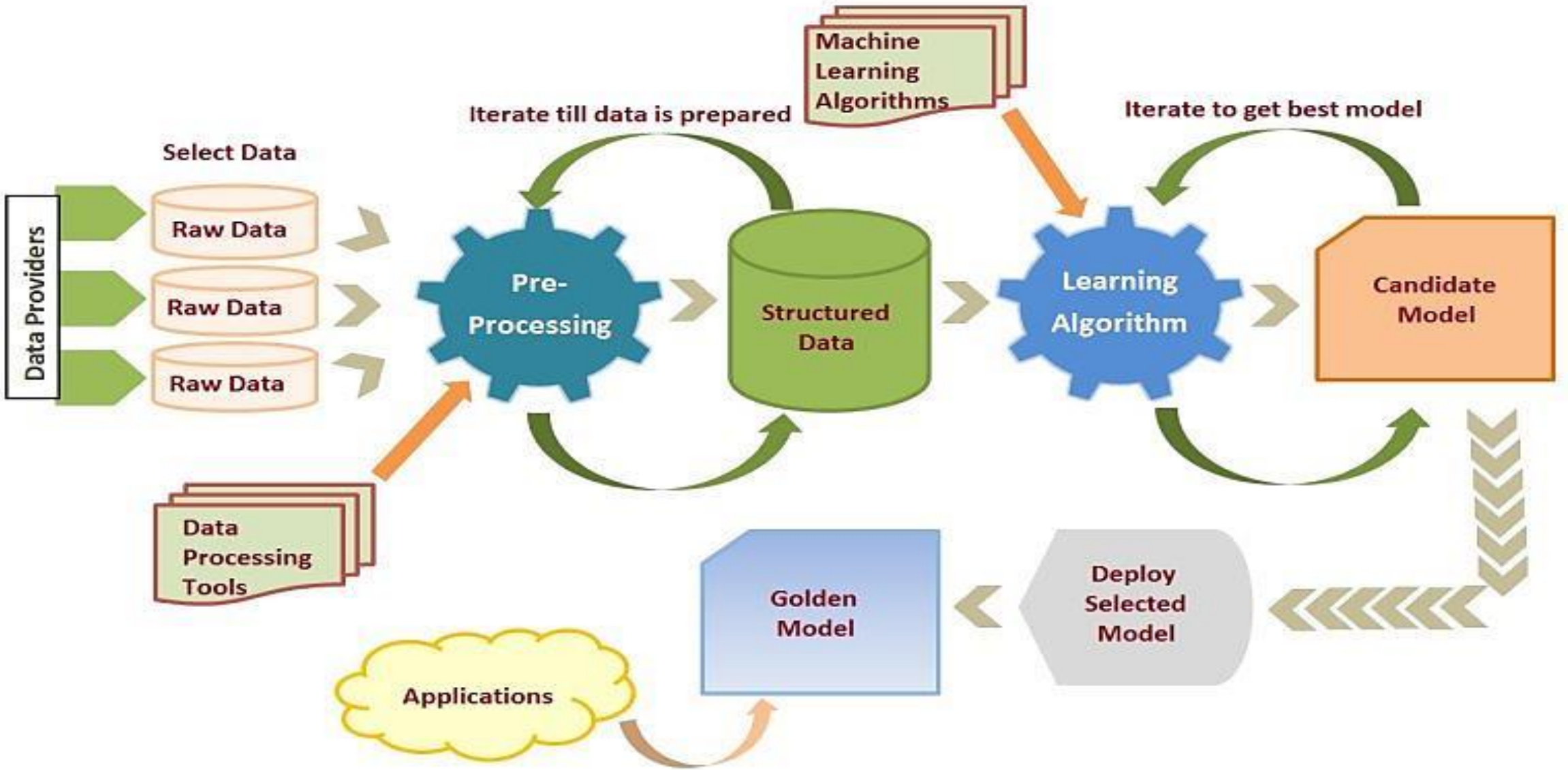
Tools Implementasi Machine Learning & Deep Learning dan **Cheat Sheets for AI**

Achmad Benny Mutiara

2021

APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Proses Machine Learning



Tool Implementasi: Matlab

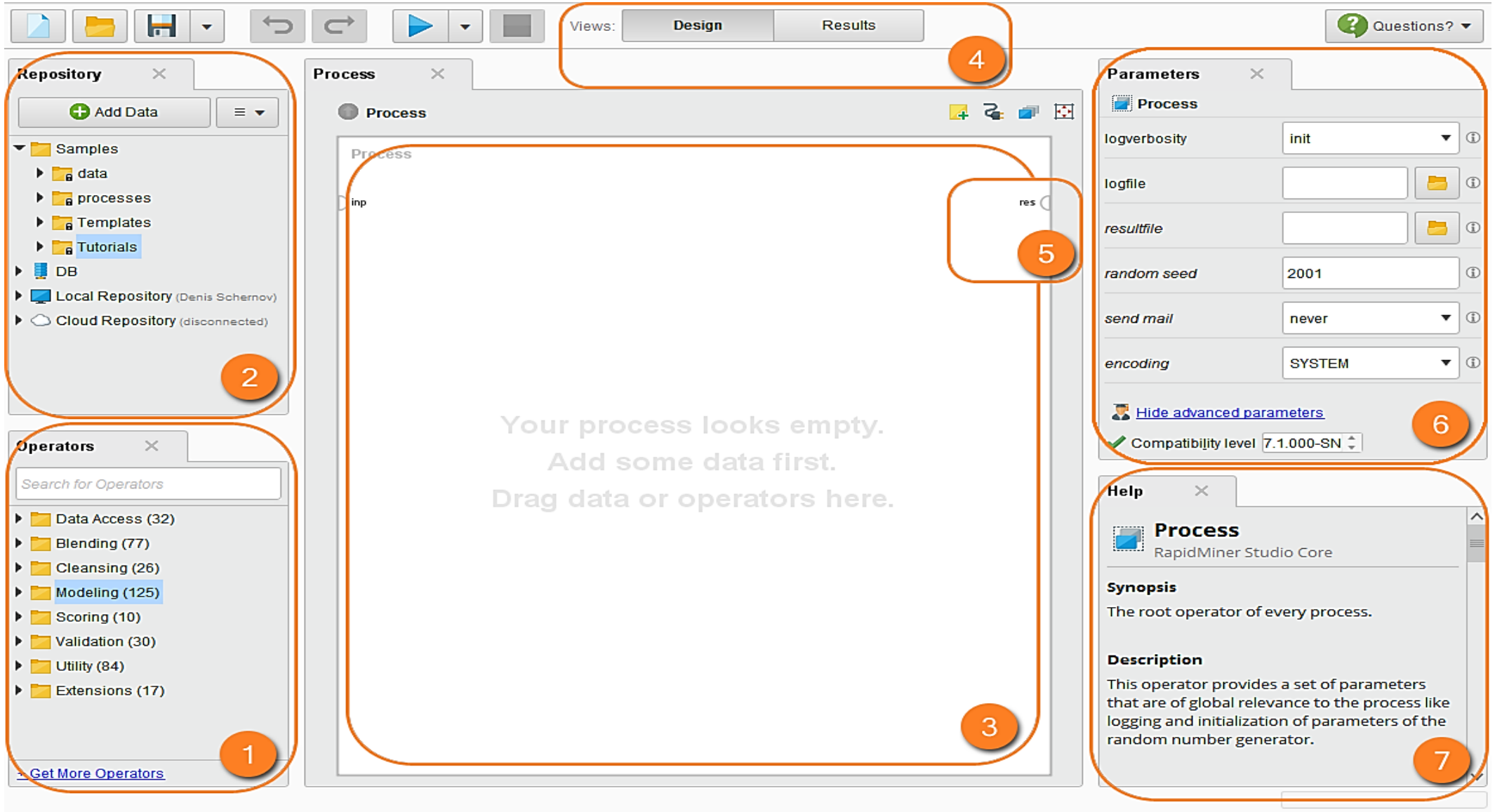
- Matlab https://www.mathworks.com/products/matlab.html
  - Komersial versi terakhir R2020a
  - Tersedia Toolbox: **AI, Data Science, and Statistics**
    - Statistics and Machine Learning Toolbox
    - Deep Learning Toolbox
    - Reinforcement Learning Toolbox
    - Text Analytics Toolbox
    - Predictive Maintenance Toolbox

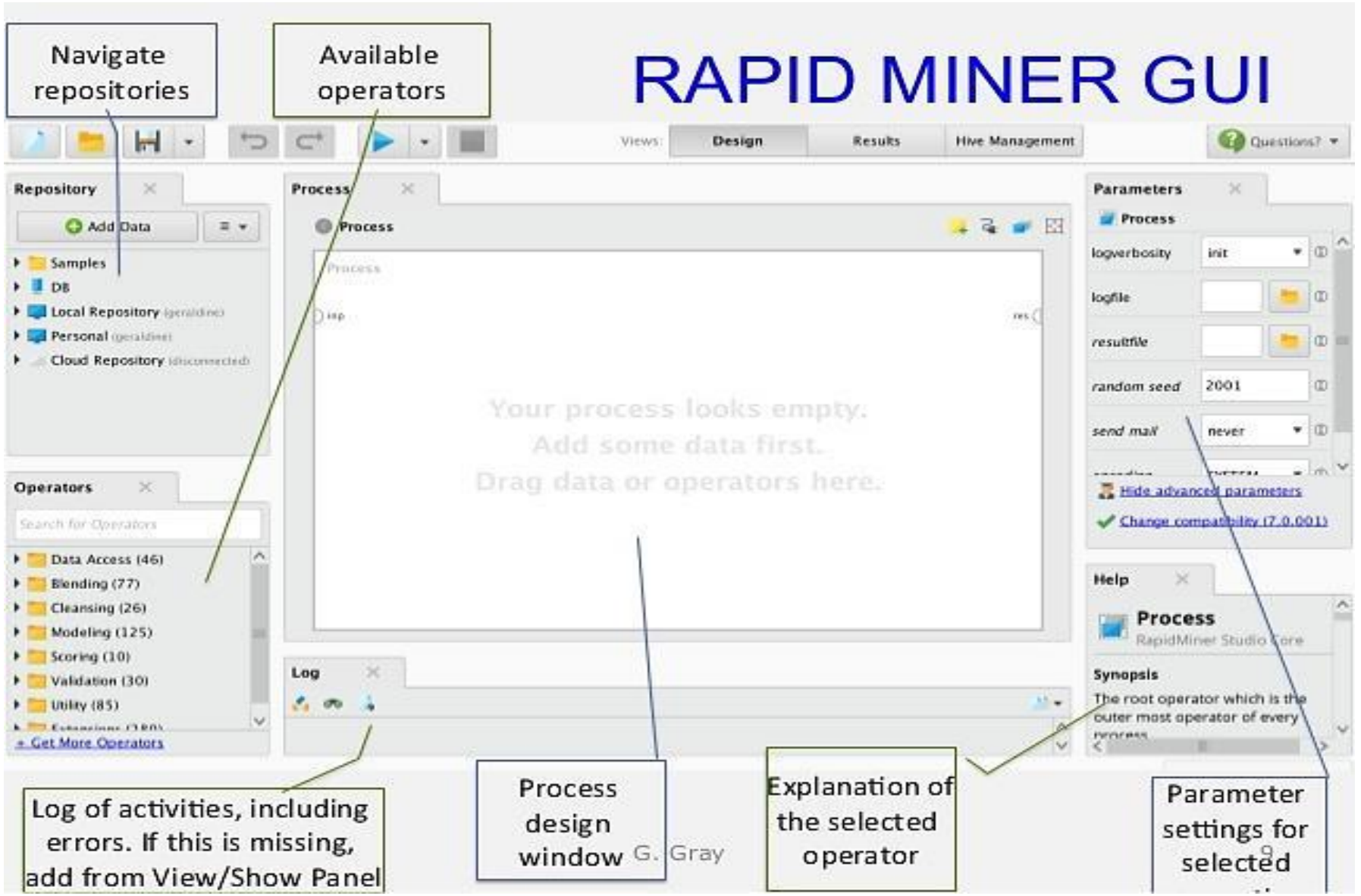


- Link buku Matlab: https://drive.google.com/drive/folders/1qHLqc2kYrI7REC2UClijIZhrzICmm8AF?usp=sharing
- Link buku Deep Learning with Matlab: https://drive.google.com/drive/folders/1QuU9tAMPF-XPwM4WmSBRiSYQoj8aA9Wg?usp=sharing

Tool Implementasi: RapidMiner

- RapidMiner https://rapidminer.com/
  - platform perangkat lunak data science
  - yang dikembangkan oleh perusahaan bernama sama dengan yang menyediakan lingkungan terintegrasi untuk **data preparation, machine learning, deep learning, text mining, and predictive analytics.**
  - Digunakan untuk bisnis dan komersial, juga untuk penelitian, pendidikan, pelatihan, *rapid prototyping*, dan pengembangan aplikasi serta mendukung semua langkah dalam proses **machine learning termasuk data preparation, results visualization, model validation and optimization**.
  - RapidMiner dikembangkan pada **open core model**. Dengan RapidMiner Studio **Free Edition**, yang terbatas untuk 1 prosesor logika dan 10.000 baris data, tersedia di bawah lisensi AGPL. **RapidMiner Studio 9.7 (https://my.rapidminer.com/nexus/account/index.html#downloads)** Harga komersial dimulai dari $2.500 dan tersedia dari pengembang.
- Link buku RapidMiner: https://drive.google.com/drive/folders/1In2R4ryr2qj_Iwbk-ZZT_T9wTyvpuhaN?usp=sharing


Views:
Design
Results
Questions?
Repository
Add Data
Samples
data
processes
Templates
Tutorials
DB
Local Repository (Denis Schernov)
Cloud Repository (disconnected)
Operators
Search for Operators
Data Access (32)
Blending (77)
Cleansing (26)
Modeling (125)
Scoring (10)
Validation (30)
Utility (84)
Extensions (17)
Get More Operators
Process
Process
Process
inp
res
Your process looks empty.
Add some data first.
Drag data or operators here.
Parameters
Process
logverbosity
init
logfile
resultfile
random seed
2001
send mail
never
encoding
SYSTEM
Hide advanced parameters
Compatibility level
7.1.000-SN
Help
Process
RapidMiner Studio Core
Synopsis
The root operator of every process.
Description
This operator provides a set of parameters that are of global relevance to the process like logging and initialization of parameters of the random number generator.
1
2
3
4
5
6
7

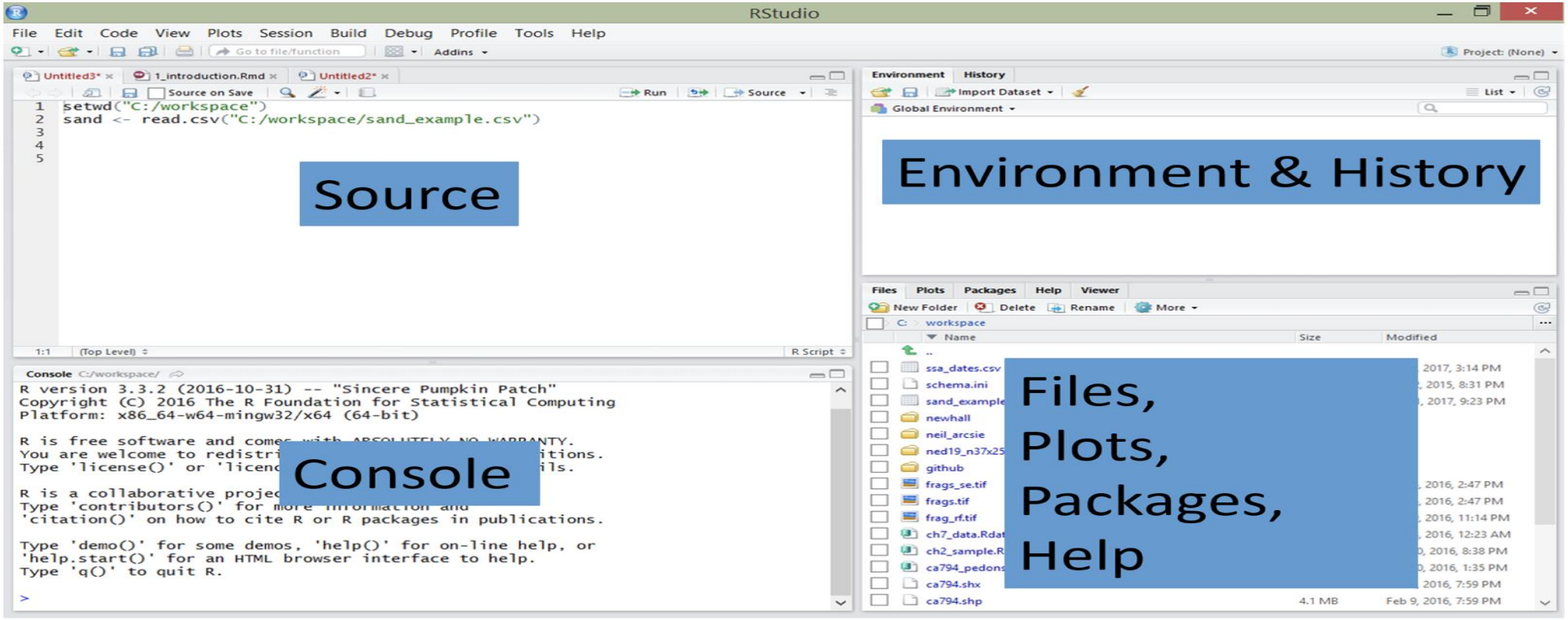


APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Tool Implementasi: R-Studio

4 RStudio: An Integrated Development Environment (IDE) for R

RStudio is an integrated development environment (IDE) that allows you to interact with R more readily. RStudio is similar to the standard RGui, but is considerably more user friendly. It has more drop-down menus, windows with multiple tabs, and many customization options. The first time you open RStudio, you will see three windows. A forth window is hidden by default, but can be opened by clicking the **File** drop-down menu, then **New File,** and then **R Script.**. Detailed information on using RStudio can be found at at RStudio's Website.

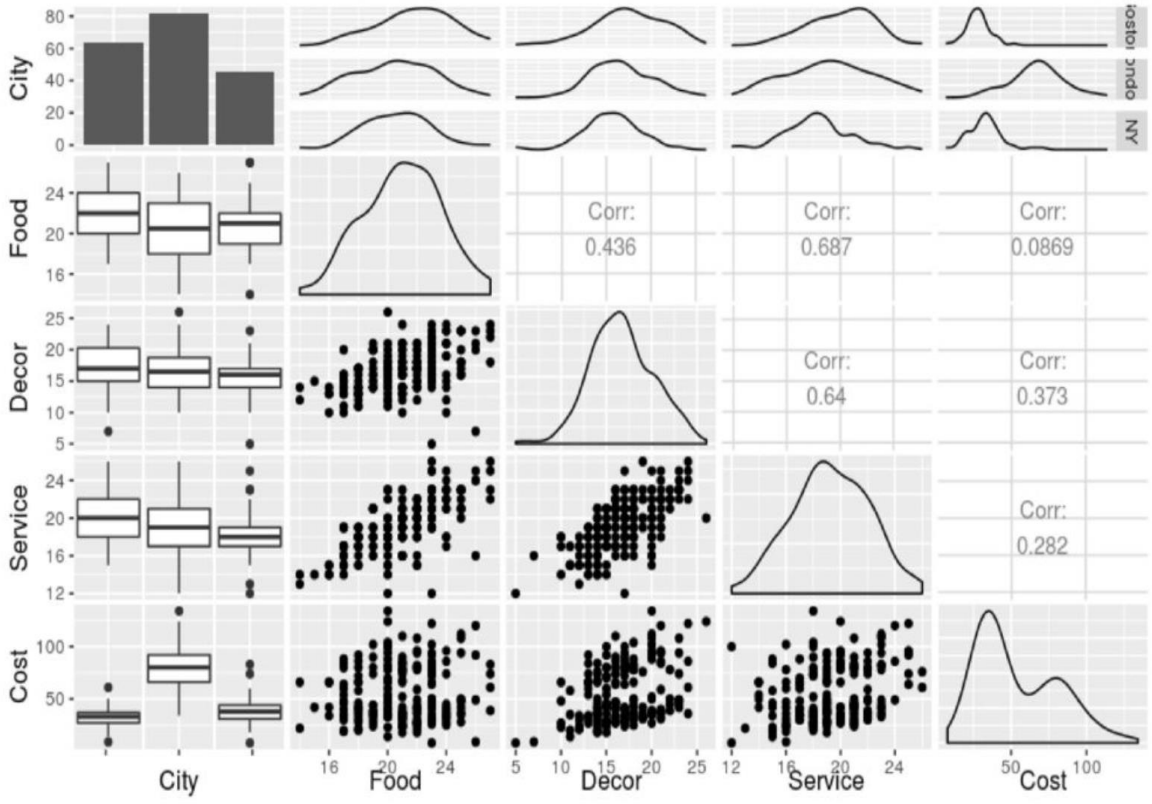


RStudio Windows / Tabs	Location	Description

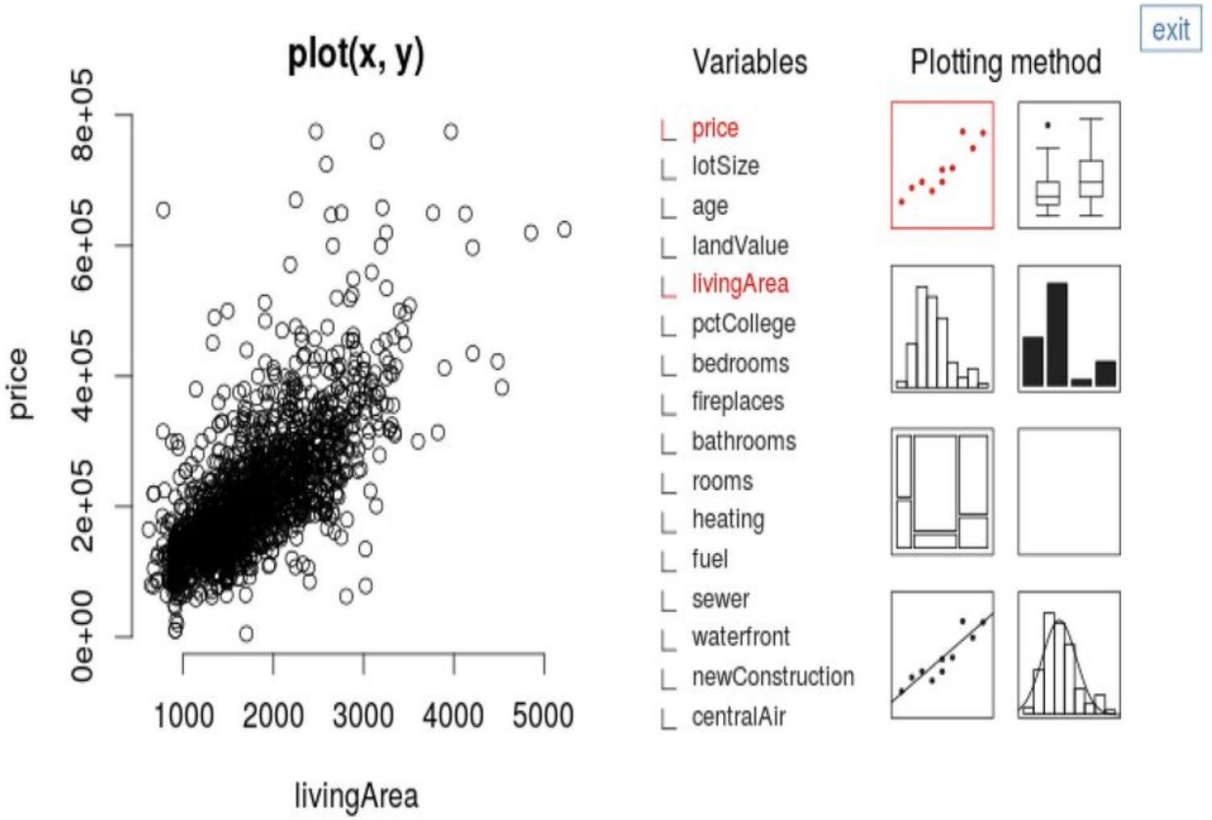
Mengapa Pakai R Language ?

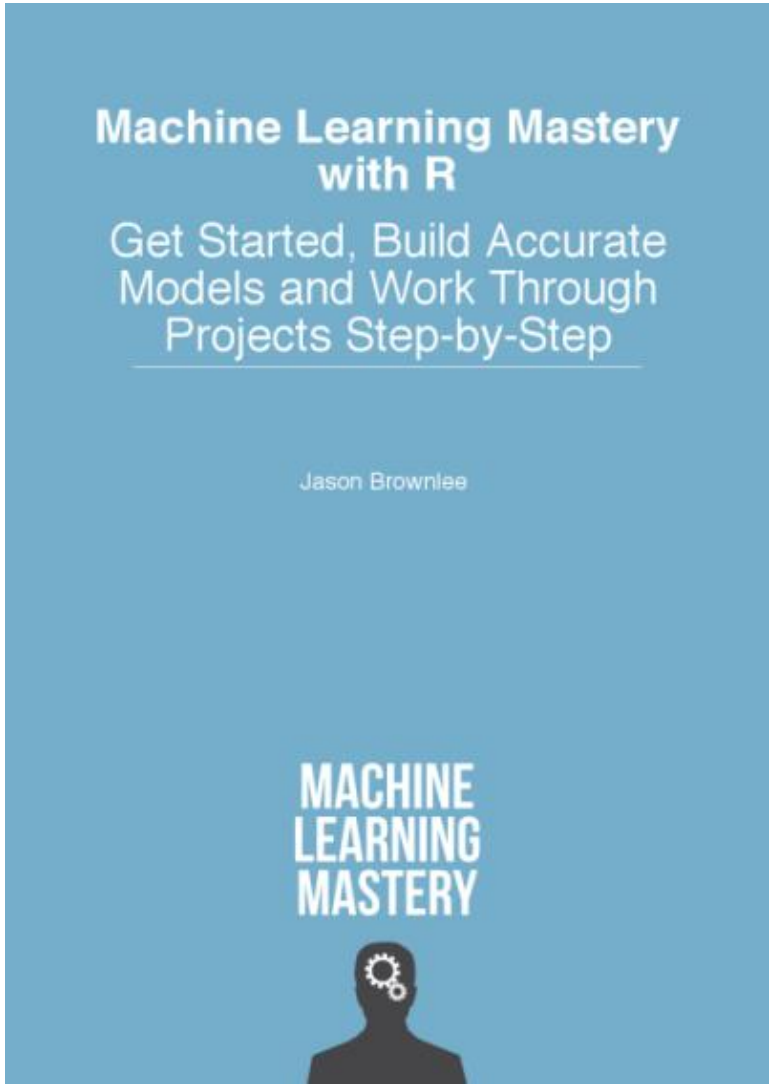
- R is a free, open-source software and programming language developed in 1995 at the University of Auckland as an environment for statistical computing and graphics (Ikaha and Gentleman, 1996).
- Since then R has become one of the dominant software environments for **data analysis** and is used by a variety of scientific disiplines, including soil science, ecology, and geoinformatics (Envirometrics CRAN Task View; Spatial CRAN Task View).
- R is particularly popular for its graphical capabilities, but it is also prized for it's GIS capabilities which make it relatively easy to generate raster-based models.
- More recently, R has also gained several packages which are designed specifically for analyzing soil data.

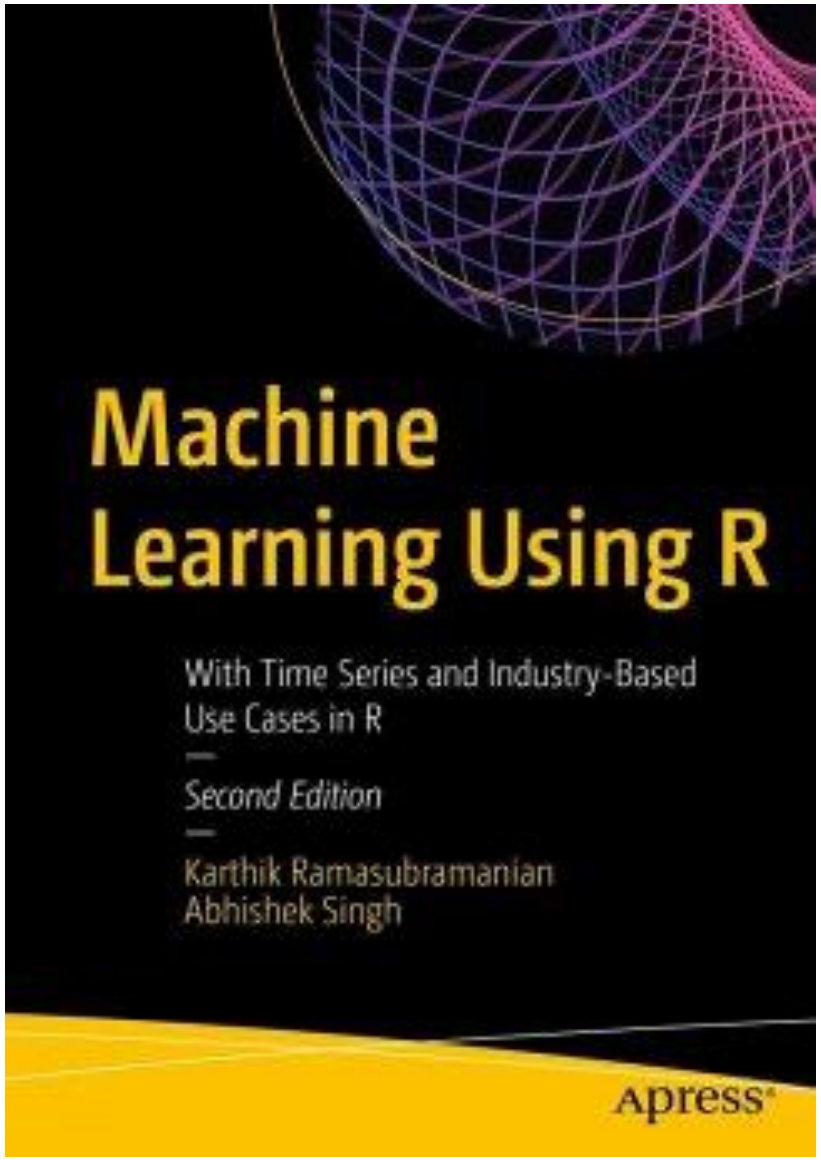
An example of graphing using the GGally package in R

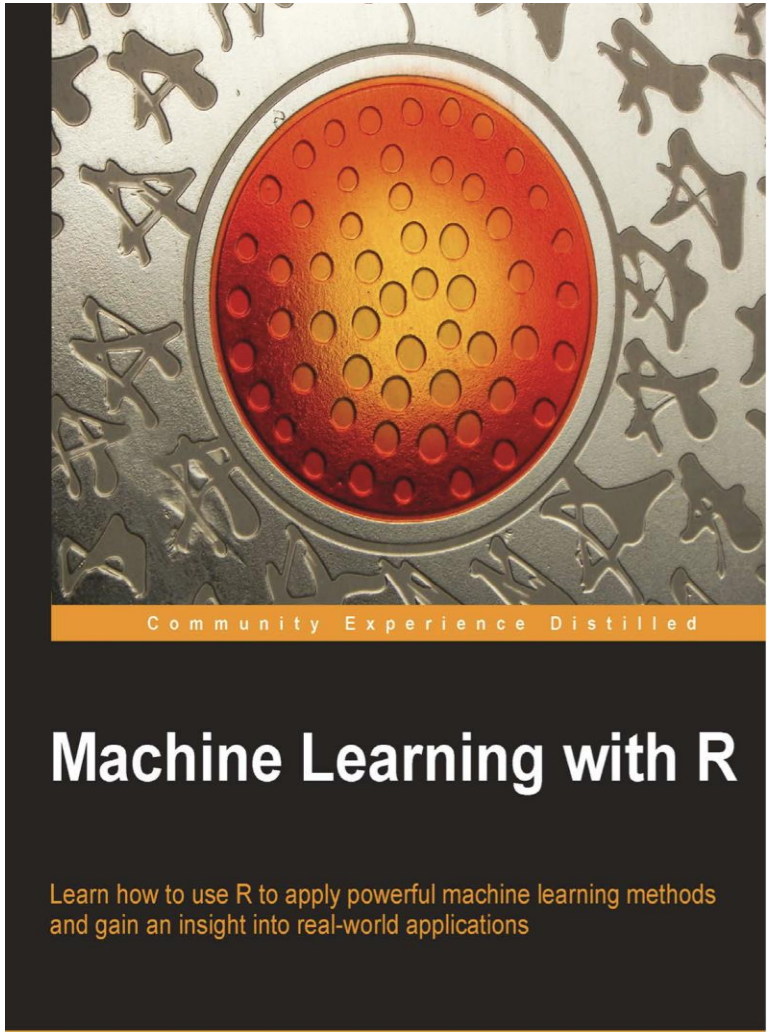


The openintro package edaPlot() command makes exploring data graphically easy to do. edaPlot(SaratogaHouses)




Machine Learning Mastery with R
Get Started, Build Accurate Models and Work Through Projects Step-by-Step
Jason Brownlee
MACHINE LEARNING MASTERY

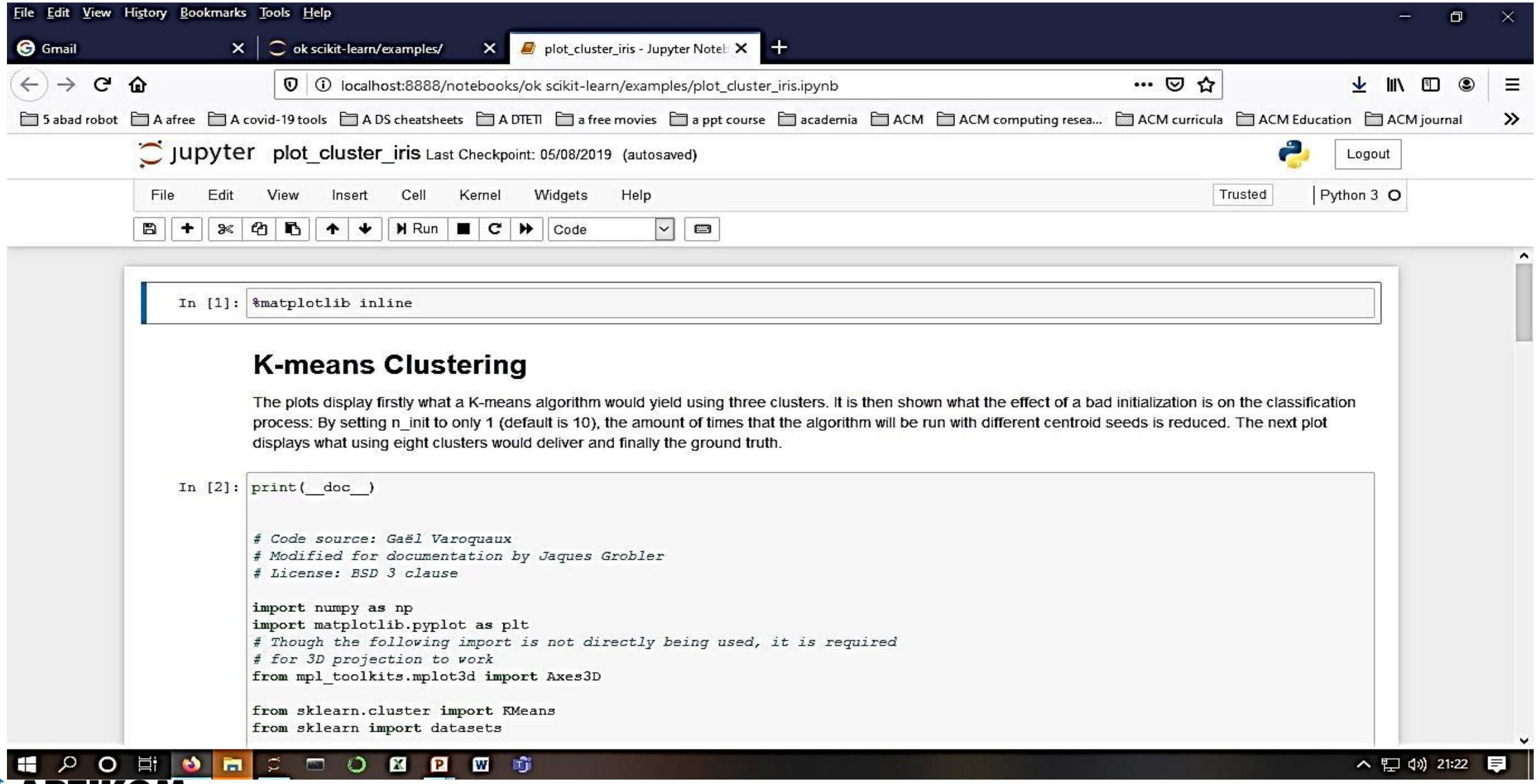

Machine Learning Using R
With Time Series and Industry-Based Use Cases in R
Second Edition
Karthik Ramasubramanian
Abhishek Singh
Apress


Community Experience Distilled
Machine Learning with R
Learn how to use R to apply powerful machine learning methods and gain an insight into real-world applications

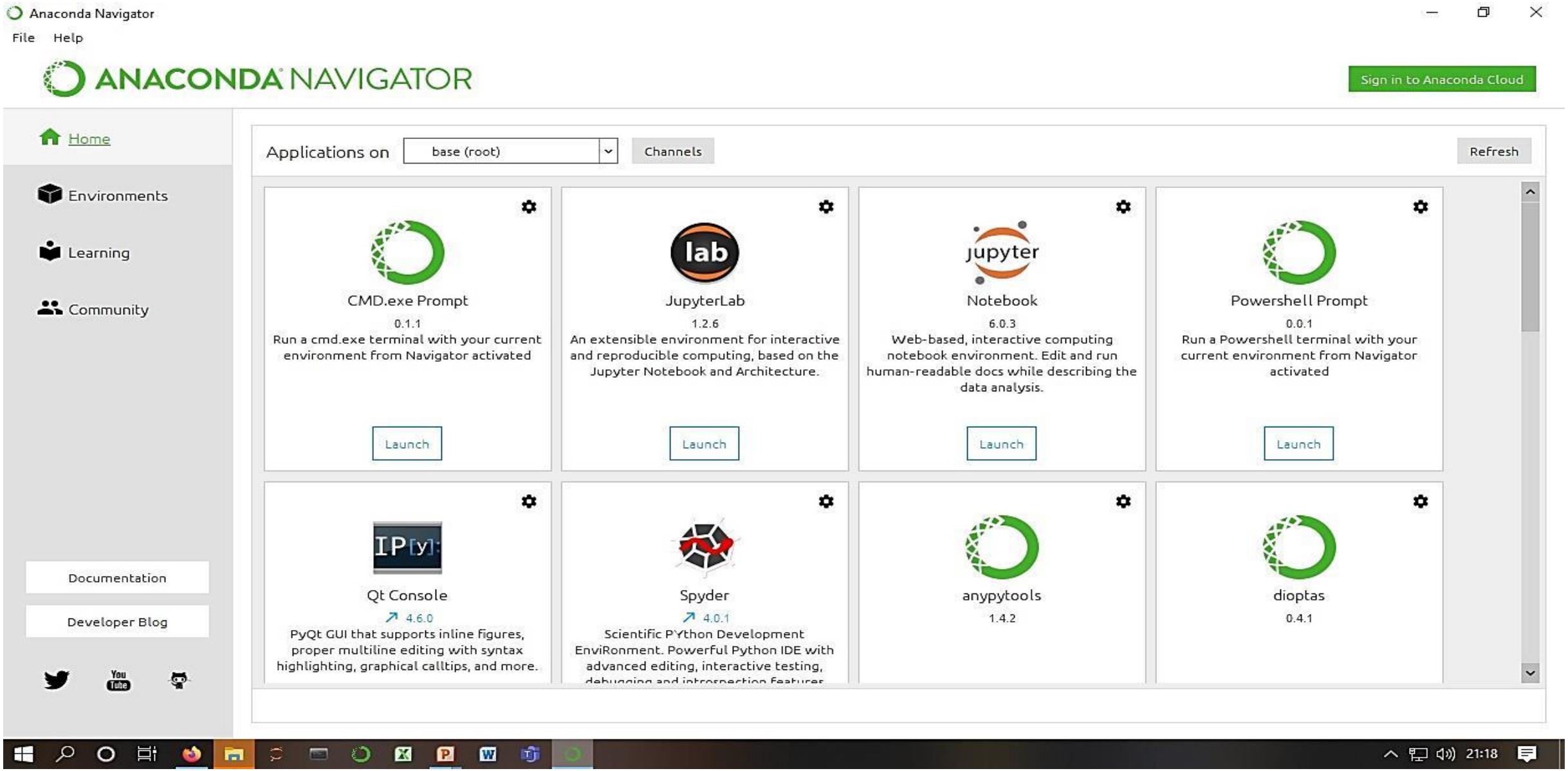
Tool Implementasi: Python, Jupyter, Anaconda

- Python
  - Versi 3.8.X
  - Tersedia IDE: **Spyder** https://www.spyder-ide.org/
  - Tool interactive: **Jupyter** *(Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.)* https://jupyter.org/
  - Toolkit: **Anaconda** (*the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries*) https://www.anaconda.com/
  - **Google Colab** Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with
    - Zero configuration required
    - Free access to GPUs
    - Easy sharing
    - Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier https://colab.research.google.com/notebooks/intro.ipynb

Jupyter


File Edit View History Bookmarks Tools Help
Gmail
ok scikit-learn/examples/
plot_cluster_iris - Jupyter Noteb
localhost:8888/notebooks/ok scikit-learn/examples/plot_cluster_iris.ipynb
5 abad robot
A afree
A covid-19 tools
A DS cheatsheets
A DTETI
a free movies
a ppt course
academia
ACM
ACM computing resea...
ACM curricula
ACM Education
ACM journal
jupyter plot_cluster_iris Last Checkpoint: 05/08/2019 (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
In [1]: %matplotlib inline
K-means Clustering
The plots display firstly what a K-means algorithm would yield using three clusters. It is then shown what the effect of a bad initialization is on the classification process: By setting n_init to only 1 (default is 10), the amount of times that the algorithm will be run with different centroid seeds is reduced. The next plot displays what using eight clusters would deliver and finally the ground truth.
In [2]: print(__doc__)
# Code source: Gaël Varoquaux
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause
import numpy as np
import matplotlib.pyplot as plt
# Though the following import is not directly being used, it is required
# for 3D projection to work
from mpl_toolkits.mplot3d import Axes3D
from sklearn.cluster import KMeans
from sklearn import datasets
21:22

Anaconda


Anaconda Navigator
File Help
ANACONDA NAVIGATOR
Sign in to Anaconda Cloud
Home
Environments
Learning
Community
Documentation
Developer Blog
Applications on
base (root)
Channels
Refresh
CMD.exe Prompt
0.1.1
Run a cmd.exe terminal with your current environment from Navigator activated
Launch
JupyterLab
1.2.6
An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture.
Launch
Notebook
6.0.3
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch
Powershell Prompt
0.0.1
Run a Powershell terminal with your current environment from Navigator activated
Launch
Qt Console
4.6.0
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.
Spyder
4.0.1
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing,
anypytools
1.4.2
dioptas
0.4.1
21:18

Cheat-Sheet for AI



Table of Content

Neural Networks

03 Neural Networks Basics

04 Neural Network Graphs

Machine Learning

06 Machine Learning Basics

07 Scikit Learn with Python

08 Scikit Learn Algorithm

09 Choosing ML Algorithm

Data Science with Python

11 Tensor Flow

12 Python Basics

13 PySpark Basics

14 Numpy Basics

15 Bokeh

16 Karas

17 Pandas

18 Data Wrangling with Pandas

19 Data Wrangling with dplyr & tidyr

20 SciPi

21 MatPlotLib

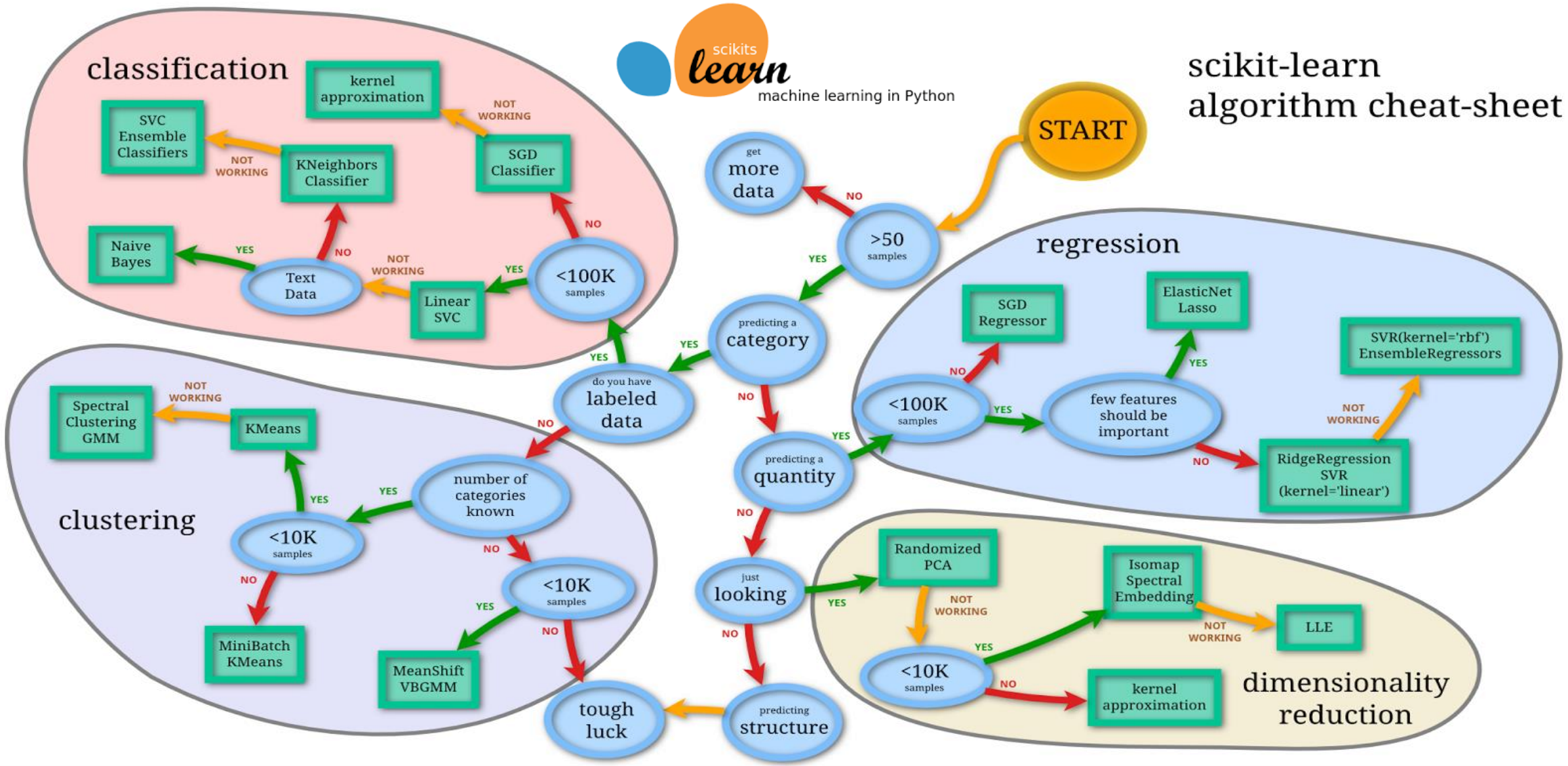
22 Data Visualization with ggplot

23 Big-O

APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Download: https://drive.google.com/drive/folders/1o3t1Ceg_0fL6lfXaNtfq6MzNLgWKqLKH?usp=sharing

Library: scikit-learn



SCIKIT LEARN : https://scikit-learn.org/stable/

Python For Data Science *Cheat Sheet*

Scikit-Learn

Learn Python for data science **Interactively** at www.DataCamp.com

Scikit-learn

Scikit-learn is an open source Python library that implements a range of machine learning, preprocessing, cross-validation and visualization algorithms using a unified interface.

A Basic Example

```
>>> from sklearn import neighbors, datasets, preprocessing
>>> from sklearn.cross_validation import train_test_split
>>> from sklearn.metrics import accuracy_score
>>> iris = datasets.load_iris()
>>> X, y = iris.data[:, :2], iris.target
>>> X_train, X_test, y_train, y_test= train_test_split(X, y, random_state=33)
>>> scaler = preprocessing.StandardScaler().fit(X_train)
>>> X_train = scaler.transform(X_train)
>>> X_test = scaler.transform(X_test)
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
>>> knn.fit(X_train, y_train)
>>> y_pred = knn.predict(X_test)
>>> accuracy_score(y_test, y_pred)
```

Loading The Data

Also see NumPy & Pandas

Your data needs to be numeric and stored as NumPy arrays or SciPy sparse matrices. Other types that are convertible to numeric arrays, such as Pandas DataFrame, are also acceptable.

```
>>> import numpy as np
>>> X = np.random.random((10,5))
>>> y = np.array(['M','M','F','F','M','F','M','M','F','F','F'])
>>> X[X < 0.7] = 0
```

Training And Test Data

```
>>> from sklearn.cross_validation import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        random_state=0)
```

Create Your Model

Supervised Learning Estimators

Linear Regression

```
>>> from sklearn.linear_model import LinearRegression
>>> lr = LinearRegression(normalize=True)
```

Support Vector Machines (SVM)

```
>>> from sklearn.svm import SVC
>>> svc = SVC(kernel='linear')
```

Naive Bayes

```
>>> from sklearn.naive_bayes import GaussianNB
>>> gnb = GaussianNB()
```

KNN

```
>>> from sklearn import neighbors
>>> knn = neighbors.KNeighborsClassifier(n_neighbors=5)
```

Unsupervised Learning Estimators

Principal Component Analysis (PCA)

```
>>> from sklearn.decomposition import PCA
>>> pca = PCA(n_components=0.95)
```

K Means

```
>>> from sklearn.cluster import KMeans
>>> k_means = KMeans(n_clusters=3, random_state=0)
```

Model Fitting

Supervised learning	
`>>> lr.fit(X, y)`	Fit the model to the data
`>>> knn.fit(X_train, y_train)`	
`>>> svc.fit(X_train, y_train)`	
Unsupervised Learning	
`>>> k_means.fit(X_train)`	Fit the model to the data
`>>> pca_model = pca.fit_transform(X_train)`	Fit to data, then transform it

Prediction

Supervised Estimators	
`>>> y_pred = svc.predict(np.random.random((2,5)))`	Predict labels
`>>> y_pred = lr.predict(X_test)`	Predict labels
`>>> y_pred = knn.predict_proba(X_test)`	Estimate probability of a label
Unsupervised Estimators	
`>>> y_pred = k_means.predict(X_test)`	Predict labels in clustering algos

Preprocessing The Data

Standardization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(X_train)
>>> standardized_X = scaler.transform(X_train)
>>> standardized_X_test = scaler.transform(X_test)
```

Normalization

```
>>> from sklearn.preprocessing import Normalizer
>>> scaler = Normalizer().fit(X_train)
>>> normalized_X = scaler.transform(X_train)
>>> normalized_X_test = scaler.transform(X_test)
```

Binarization

```
>>> from sklearn.preprocessing import Binarizer
>>> binarizer = Binarizer(threshold=0.0).fit(X)
>>> binary_X = binarizer.transform(X)
```

Encoding Categorical Features

```
>>> from sklearn.preprocessing import LabelEncoder
>>> enc = LabelEncoder()
>>> y = enc.fit_transform(y)
```

Imputing Missing Values

```
>>> from sklearn.preprocessing import Imputer
>>> imp = Imputer(missing_values=0, strategy='mean', axis=0)
>>> imp.fit_transform(X_train)
```

Generating Polynomial Features

```
>>> from sklearn.preprocessing import PolynomialFeatures
>>> poly = PolynomialFeatures(5)
>>> poly.fit_transform(X)
```

Evaluate Your Model's Performance

Classification Metrics

Accuracy Score — Estimator score method

```
>>> knn.score(X_test, y_test)
```

Metric scoring functions

```
>>> from sklearn.metrics import accuracy_score
>>> accuracy_score(y_test, y_pred)
```

Classification Report — Precision, recall, f1-score and support

```
>>> from sklearn.metrics import classification_report
>>> print(classification_report(y_test, y_pred))
```

Confusion Matrix

```
>>> from sklearn.metrics import confusion_matrix
>>> print(confusion_matrix(y_test, y_pred))
```

Regression Metrics

Mean Absolute Error

```
>>> from sklearn.metrics import mean_absolute_error
>>> y_true = [3, -0.5, 2]
>>> mean_absolute_error(y_true, y_pred)
```

Mean Squared Error

```
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, y_pred)
```

R^2 Score

```
>>> from sklearn.metrics import r2_score
>>> r2_score(y_true, y_pred)
```

Clustering Metrics

Adjusted Rand Index

```
>>> from sklearn.metrics import adjusted_rand_score
>>> adjusted_rand_score(y_true, y_pred)
```

Homogeneity

```
>>> from sklearn.metrics import homogeneity_score
>>> homogeneity_score(y_true, y_pred)
```

V-measure

```
>>> from sklearn.metrics import v_measure_score
>>> metrics.v_measure_score(y_true, y_pred)
```

Cross-Validation

```
>>> from sklearn.cross_validation import cross_val_score
>>> print(cross_val_score(knn, X_train, y_train, cv=4))
>>> print(cross_val_score(lr, X, y, cv=2))
```

Tune Your Model

Grid Search

```
>>> from sklearn.grid_search import GridSearchCV
>>> params = {"n_neighbors": np.arange(1,3),
              "metric": ["euclidean", "cityblock"]}
>>> grid = GridSearchCV(estimator=knn,
                        param_grid=params)
>>> grid.fit(X_train, y_train)
>>> print(grid.best_score_)
>>> print(grid.best_estimator_.n_neighbors)
```

Randomized Parameter Optimization

```
>>> from sklearn.grid_search import RandomizedSearchCV
>>> params = {"n_neighbors": range(1,5),
              "weights": ["uniform", "distance"]}
>>> rsearch = RandomizedSearchCV(estimator=knn,
                                 param_distributions=params,
                                 cv=4,
                                 n_iter=8,
                                 random_state=5)
>>> rsearch.fit(X_train, y_train)
>>> print(rsearch.best_score_)
```

DataCamp
Learn Python for Data Science Interactively

APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com



Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
 5
```

Calculations With Variables

```
>>> x+2
 7
>>> x-2
 3
>>> x*2
 10
>>> x**2
 25
>>> x%2
 1
>>> x/float(2)
 2.5
```

Sum of two variables
Subtraction of two variables
Multiplication of two variables
Exponentiation of a variable
Remainder of a variable
Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset

```
>>> my_list[1]
>>> my_list[-3]
```

Select item at index 1
Select 3rd last item

Slice

```
>>> my_list[1:3]
>>> my_list[1:]
>>> my_list[:3]
>>> my_list[:]
```

Select items at index 1 and 2
Select items after index 0
Select items before index 3
Copy my_list

Subset Lists of Lists

```
>>> my_list2[1][0]
>>> my_list2[1][:2]
```

my_list[list][itemOfList]

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

```
>>> my_list.index(a)
>>> my_list.count(a)
>>> my_list.append('!')
>>> my_list.remove('!')
>>> del(my_list[0:1])
>>> my_list.reverse()
>>> my_list.extend('!')
>>> my_list.pop(-1)
>>> my_list.insert(0,'!')
>>> my_list.sort()
```

Get the index of an item
Count an item
Append an item at a time
Remove an item
Remove an item
Reverse the list
Append an item
Remove an item
Insert an item
Sort the list

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
 'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
 'thisStringIsAwesomeInnit'
>>> 'm' in my_string
 True
```

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

```
>>> my_string.upper()
>>> my_string.lower()
>>> my_string.count('w')
>>> my_string.replace('e', 'i')
>>> my_string.strip()
```

String to uppercase
String to lowercase
Count String elements
Replace String elements
Strip whitespace from ends

Libraries

Import libraries

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

Selective import

```
>>> from math import pi
```

pandas — Data analysis
Machine learning
NumPy — Scientific computing
matplotlib — 2D plotting

Install Python

ANACONDA — Leading open data science platform powered by Python

spyder — free IDE that is included with Anaconda

jupyter — Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset

```
>>> my_array[1]
  2
```

Select item at index 1

Slice

```
>>> my_array[0:2]
  array([1, 2])
```

Select items at index 0 and 1

Subset 2D Numpy arrays

```
>>> my_2darray[:,0]
  array([1, 4])
```

my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
 array([False, False, False,  True], dtype=bool)
>>> my_array * 2
 array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
 array([6, 8, 10, 12])
```

Numpy Array Functions

```
>>> my_array.shape
>>> np.append(other_array)
>>> np.insert(my_array, 1, 5)
>>> np.delete(my_array,[1])
>>> np.mean(my_array)
>>> np.median(my_array)
>>> my_array.corrcoef()
>>> np.std(my_array)
```

Get the dimensions of the array
Append items to an array
Insert items in an array
Delete items in an array
Mean of the array
Median of the array
Correlation coefficient
Standard deviation

DataCamp
Learn Python for Data Science Interactively



APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Python For Data Science *Cheat Sheet*

NumPy Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

NumPy

The **NumPy** library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

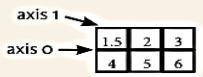
```
>>> import numpy as np
```

NumPy Arrays

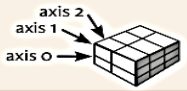
1D array



2D array



3D array



Creating Arrays

```
>>> a = np.array([1,2,3])
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]],
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))                  Create an array of zeros
>>> np.ones((2,3,4),dtype=np.int16)  Create an array of ones
>>> d = np.arange(10,25,5)           Create an array of evenly
                                     spaced values (step value)
>>> np.linspace(0,2,9)               Create an array of evenly
                                     spaced values (number of samples)
>>> e = np.full((2,2),7)             Create a constant array
>>> f = np.eye(2)                    Create a 2X2 identity matrix
>>> np.random.random((2,2))          Create an array with random values
>>> np.empty((3,2))                  Create an empty array
```

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)
>>> np.savez('array.npz', a, b)
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt("myfile.txt")
>>> np.genfromtxt("my_file.csv", delimiter=',')
>>> np.savetxt("myarray.txt", a, delimiter=" ")
```

Data Types

```
>>> np.int64       Signed 64-bit integer types
>>> np.float32     Standard double-precision floating point
>>> np.complex     Complex numbers represented by 128 floats
>>> np.bool        Boolean type storing TRUE and FALSE values
>>> np.object      Python object type
>>> np.string_     Fixed-length string type
>>> np.unicode_    Fixed-length unicode type
```

Inspecting Your Array

```
>>> a.shape             Array dimensions
>>> len(a)              Length of array
>>> b.ndim              Number of array dimensions
>>> e.size              Number of array elements
>>> b.dtype             Data type of array elements
>>> b.dtype.name        Name of data type
>>> b.astype(int)       Convert an array to a different type
```

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b                        Subtraction
  array([[-0.5,  0. ,  0. ],
         [-3. , -3. , -3. ]])
>>> np.subtract(a,b)                 Subtraction
>>> b + a                            Addition
  array([[ 2.5,  4. ,  6. ],
         [ 5. ,  7. ,  9. ]])
>>> np.add(b,a)                      Addition
>>> a / b                            Division
  array([[ 0.66666667,  1.        ,  1.        ],
         [ 0.25      ,  0.4       ,  0.5       ]])
>>> np.divide(a,b)                   Division
>>> a * b                            Multiplication
  array([[  1.5,   4. ,   9. ],
         [  4. ,  10. ,  18. ]])
>>> np.multiply(a,b)                 Multiplication
>>> np.exp(b)                        Exponentiation
>>> np.sqrt(b)                       Square root
>>> np.sin(a)                        Print sines of an array
>>> np.cos(b)                        Element-wise cosine
>>> np.log(a)                        Element-wise natural logarithm
>>> e.dot(f)                         Dot product
  array([[ 7.,  7.],
         [ 7.,  7.]])
```

Comparison

```
>>> a == b                           Element-wise comparison
  array([[False,  True,  True],
         [False, False, False]], dtype=bool)
>>> a < 2                            Element-wise comparison
  array([True, False, False], dtype=bool)
>>> np.array_equal(a, b)             Array-wise comparison
```

Aggregate Functions

```
>>> a.sum()             Array-wise sum
>>> a.min()             Array-wise minimum value
>>> b.max(axis=0)       Maximum value of an array row
>>> b.cumsum(axis=1)    Cumulative sum of the elements
>>> a.mean()            Mean
>>> b.median()          Median
>>> a.corrcoef()        Correlation coefficient
>>> np.std(b)           Standard deviation
```

Copying Arrays

```
>>> h = a.view()        Create a view of the array with the same data
>>> np.copy(a)          Create a copy of the array
>>> h = a.copy()        Create a deep copy of the array
```

Sorting Arrays

```
>>> a.sort()            Sort an array
>>> c.sort(axis=0)      Sort the elements of an array's axis
```

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]                Select the element at the 2nd index
  3
>>> b[1,2]              Select the element at row 0 column 2
  6.0                   (equivalent to b[1][2])
```

Slicing

```
>>> a[0:2]              Select items at index 0 and 1
  array([1, 2])
>>> b[0:2,1]            Select items at rows 0 and 1 in column 1
  array([ 2.,  5.])
>>> b[:1]               Select all items at row 0
  array([[1.5, 2., 3.]])  (equivalent to b[0:1, :])
>>> c[1,...]            Same as [1,:,:]
  array([[[ 3.,  2.,  1.],
          [ 4.,  5.,  6.]]])
>>> a[ : :-1]           Reversed array a
  array([3, 2, 1])
```

Boolean Indexing

```
>>> a[a<2]              Select elements from a less than 2
  array([1])
```

Fancy Indexing

```
>>> b[[1, 0, 1, 0],[0, 1, 2, 0]]   Select elements (1,0),(0,1),(1,2) and (0,0)
  array([ 4. ,  2. ,  6. ,  1.5])
>>> b[[1, 0, 1, 0]][:,[0,1,2,0]]   Select a subset of the matrix's rows
  array([[ 4. ,  5. ,  6. ,  4. ],  and columns
         [ 1.5,  2. ,  3. ,  1.5],
         [ 4. ,  5. ,  6. ,  4. ],
         [ 1.5,  2. ,  3. ,  1.5]])
```

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)     Permute array dimensions
>>> i.T                     Permute array dimensions
```

Changing Array Shape

```
>>> b.ravel()               Flatten the array
>>> g.reshape(3,-2)         Reshape, but don't change data
```

Adding/Removing Elements

```
>>> h.resize((2,6))         Return a new array with shape (2,6)
>>> np.append(h,g)          Append items to an array
>>> np.insert(a, 1, 5)      Insert items in an array
>>> np.delete(a,[1])        Delete items from an array
```

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)   Concatenate arrays
  array([ 1,  2,  3, 10, 15, 20])
>>> np.vstack((a,b))               Stack arrays vertically (row-wise)
  array([[ 1. ,  2. ,  3. ],
         [ 1.5,  2. ,  3. ],
         [ 4. ,  5. ,  6. ]])
>>> np.r_[e,f]                     Stack arrays vertically (row-wise)
>>> np.hstack((e,f))               Stack arrays horizontally (column-wise)
  array([[ 7.,  7.,  1.,  0.],
         [ 7.,  7.,  0.,  1.]])
>>> np.column_stack((a,d))         Create stacked column-wise arrays
  array([[ 1, 10],
         [ 2, 15],
         [ 3, 20]])
>>> np.c_[a,d]                     Create stacked column-wise arrays
```

Splitting Arrays

```
>>> np.hsplit(a,3)          Split the array horizontally at the 3rd
  [array([1]),array([2]),array([3])]  index
>>> np.vsplit(c,2)          Split the array vertically at the 2nd index
[array([[[ 1.5,  2. ,  1. ],
         [ 4. ,  5. ,  6. ]]]),
 array([[[ 3.,  2.,  3.],
         [ 4.,  5.,  6.]]])]
```

DataCamp
Learn Python for Data Science Interactively

APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Python For Data Science *Cheat Sheet*

Pandas Basics

Learn Python for Data Science **Interactively** at www.DataCamp.com

Pandas

The **Pandas** library is built on NumPy and provides easy-to-use **data structures** and **data analysis** tools for the Python programming language.

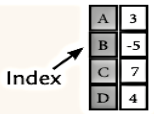
Use the following import convention:

```
>>> import pandas as pd
```

Pandas Data Structures

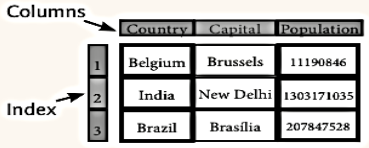
Series

A **one-dimensional** labeled array capable of holding any data type




```
>>> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame



A **two-dimensional** labeled data structure with columns of potentially different types

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}

>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

Asking For Help

```
>>> help(pd.Series.loc)
```

Selection

Also see NumPy Arrays

Getting

```
>>> s['b']
  -5

>>> df[1:]
    Country     Capital  Population
  1   India  New Delhi  1303171035
  2  Brazil   Brasília   207847528
```

Get one element

Get subset of a DataFrame

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]]
 'Belgium'
>>> df.iat[[0],[0]]
 'Belgium'
```

Select single value by row & column

By Label

```
>>> df.loc[[0], ['Country']]
 'Belgium'
>>> df.at[[0], ['Country']]
 'Belgium'
```

Select single value by row & column labels

By Label/Position

```
>>> df.ix[2]
 Country      Brazil
 Capital    Brasília
 Population  207847528
```

Select single row of subset of rows

```
>>> df.ix[:,'Capital']
 0      Brussels
 1     New Delhi
 2      Brasília
```

Select a single column of subset of columns

```
>>> df.ix[1,'Capital']
 'New Delhi'
```

Select rows and columns

Boolean Indexing

```
>>> s[~(s > 1)]
>>> s[(s < -1) | (s > 2)]
>>> df[df['Population']>1200000000]
```

Series `s` where value is not >1
`s` where value is <-1 or >2
Use filter to adjust DataFrame

Setting

```
>>> s['a'] = 6
```

Set index `a` of Series `s` to 6

Dropping

```
>>> s.drop(['a', 'c'])
>>> df.drop('Country', axis=1)
```

Drop values from rows (axis=0)
Drop values from columns(axis=1)

Sort & Rank

```
>>> df.sort_index(by='Country')
>>> s.order()
>>> df.rank()
```

Sort by row or column index
Sort a series by its values
Assign ranks to entries

Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape
>>> df.index
>>> df.columns
>>> df.info()
>>> df.count()
```

(rows,columns)
Describe index
Describe DataFrame columns
Info on DataFrame
Number of non-NA values

Summary

```
>>> df.sum()
>>> df.cumsum()
>>> df.min()/df.max()
>>> df.idmin()/df.idmax()
>>> df.describe()
>>> df.mean()
>>> df.median()
```

Sum of values
Cummulative sum of values
Minimum/maximum values
Minimum/Maximum index value
Summary statistics
Mean of values
Median of values

Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f)
>>> df.applymap(f)
```

Apply function
Apply function element-wise

Data Alignment

Internal Data Alignment

NA values are introduced in the indices that don't overlap:

```
>>> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
>>> s + s3
  a     10.0
  b      NaN
  c      5.0
  d      7.0
```

Arithmetic Operations with Fill Methods

You can also do the internal data alignment yourself with the help of the fill methods:

```
>>> s.add(s3, fill_value=0)
  a    10.0
  b    -5.0
  c    5.0
  d    7.0
>>> s.sub(s3, fill_value=2)
>>> s.div(s3, fill_value=4)
>>> s.mul(s3, fill_value=3)
```

I/O

Read and Write to CSV

```
>>> pd.read_csv('file.csv', header=None, nrows=5)
>>> pd.to_csv('myDataFrame.csv')
```

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
```

Read multiple sheets from the same file

```
>>> xlsx = pd.ExcelFile('file.xls')
>>> df = pd.read_excel(xlsx, 'Sheet1')
```

Read and Write to SQL Query or Database Table

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite:///:memory:')
>>> pd.read_sql("SELECT * FROM my_table;", engine)
>>> pd.read_sql_table('my_table', engine)
>>> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

`read_sql()` is a convenience wrapper around `read_sql_table()` and `read_sql_query()`

```
>>> pd.to_sql('myDf', engine)
```

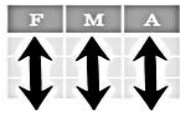

APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



&



Each **variable** is saved in its own **column**

Each **observation** is saved in its own **row**

Tidy data complements pandas's vectorized operations. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.



M * A

Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
        index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
         names=['n','v'])))
```

Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change the layout of a data set



`pd.melt(df)`
Gather columns into rows.



`df.pivot(columns='var', values='val')`
Spread rows into columns.



`pd.concat([df1,df2])`
Append rows of DataFrames



`pd.concat([df1,df2], axis=1)`
Append columns of DataFrames

`df.sort_values('mpg')`
Order rows by values of a column (low to high).

`df.sort_values('mpg',ascending=False)`
Order rows by values of a column (high to low).

`df.rename(columns = {'y':'year'})`
Rename the columns of a DataFrame

`df.sort_index()`
Sort the index of a DataFrame

`df.reset_index()`
Reset index of DataFrame to row numbers, moving index to columns.

`df.drop(['Length','Height'], axis=1)`
Drop columns from DataFrame

Subset Observations (Rows)



`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

`df.iloc[10:20]`
Select rows by position.

`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	`df.column.isin(values)`	Group membership
==	Equals	`pd.isnull(obj)`	Is NaN
<=	Less than or equals	`pd.notnull(obj)`	Is not NaN
>=	Greater than or equals	`&,|,~,^,df.any(),df.all()`	Logical and, or, not, xor, any, all

Subset Variables (Columns)



`df[['width','length','species']]`
Select multiple columns with specific names.

`df['width']` *or* `df.width`
Select single column with specific name.

`df.filter(regex='regex')`
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples

`'\.'`	Matches strings containing a period '.'
`'Length$'`	Matches strings ending with word 'Length'
`'^Sepal'`	Matches strings beginning with the word 'Sepal'
`'^x[1-5]$'`	Matches strings beginning with 'x' and ending with 1,2,3,4,5
`''^(?!Species$).*'`	Matches strings except the string 'Species'

`df.loc[:,'x2':'x4']`
Select all columns between x2 and x4 (inclusive).

`df.iloc[:,[1,2,5]]`
Select columns in positions 1, 2 and 5 (first column is 0).

`df.loc[df['a'] > 10, ['a','c']]`
Select rows meeting logical condition, and only the specific columns .

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive statistics for each column (or GroupBy)



pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

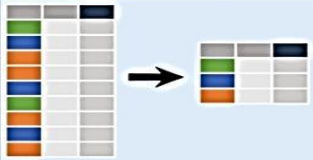
`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Group Data



`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)`
Copy with values shifted by 1.

`rank(method='dense')`
Ranks with no gaps.

`rank(method='min')`
Ranks. Ties get min rank.

`rank(pct=True)`
Ranks rescaled to interval [0, 1].

`rank(method='first')`
Ranks. Ties go to first value.

`shift(-1)`
Copy with values lagged by 1.

`cumsum()`
Cumulative sum.

`cummax()`
Cumulative max.

`cummin()`
Cumulative min.

`cumprod()`
Cumulative product.

Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns



`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.



pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)`
Element-wise max.

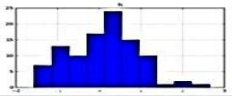
`clip(lower=-10,upper=10)`
Trim values at input thresholds

`min(axis=1)`
Element-wise min.

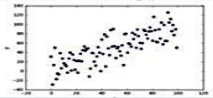
`abs()`
Absolute value.

Plotting

`df.plot.hist()`
Histogram for each column



`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points



Combine Data Sets

adf

x1	x2
A	1
B	2
C	3

+

bdf

x1	x3
A	T
B	F
D	T

=

Standard Joins

x1	x2	x3
A	1	T
B	2	F
C	3	NaN

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

x1	x2	x3
A	1.0	T
B	2.0	F
D	NaN	T

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

x1	x2	x3
A	1	T
B	2	F

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

x1	x2	x3
A	1	T
B	2	F
C	3	NaN
D	NaN	T

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

Filtering Joins

x1	x2
A	1
B	2

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

x1	x2
C	3

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf

x1	x2
A	1
B	2
C	3

+

zdf

x1	x2
B	2
C	3
D	4

=

Set-like Operations

x1	x2
B	2
C	3

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

x1	x2
A	1
B	2
C	3
D	4

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

x1	x2
A	1

```
pd.merge(ydf, zdf, how='outer',
         indicator=True)
.query('_merge == "left_only"')
.drop(['_merge'],axis=1)
```
Rows that appear in ydf but not zdf (Setdiff).

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Python For Data Science *Cheat Sheet*

SciPy - Linear Algebra

Learn More Python for Data Science Interactively at www.datacamp.com

SciPy

The SciPy library is one of the core packages for scientific computing that provides mathematical algorithms and convenience functions built on the NumPy extension of Python.

Interacting With NumPy

Also see NumPy

```
>>> import numpy as np
>>> a = np.array([1,2,3])
>>> b = np.array([(1+5j,2j,3j), (4j,5j,6j)])
>>> c = np.array([[(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)]])
```

Index Tricks

```
>>> np.mgrid[0:5,0:5]          Create a dense meshgrid
>>> np.ogrid[0:2,0:2]          Create an open meshgrid
>>> np.r_[3,[0]*5,-1:1:10j]    Stack arrays vertically (row-wise)
>>> np.c_[b,c]                 Create stacked column-wise arrays
```

Shape Manipulation

```
>>> np.transpose(b)            Permute array dimensions
>>> b.flatten()                Flatten the array
>>> np.hstack((b,c))           Stack arrays horizontally (column-wise)
>>> np.vstack((a,b))           Stack arrays vertically (row-wise)
>>> np.hsplit(c,2)             Split the array horizontally at the 2nd index
>>> np.vpsplit(d,2)            Split the array vertically at the 2nd index
```

Polynomials

```
>>> from numpy import poly1d
>>> p = poly1d([3,4,5])        Create a polynomial object
```

Vectorizing Functions

```
>>> def myfunc(a):
        if a < 0:
          return a*2
        else:
          return a/2
>>> np.vectorize(myfunc)       Vectorize functions
```

Type Handling

```
>>> np.real(b)                 Return the real part of the array elements
>>> np.imag(b)                 Return the imaginary part of the array elements
>>> np.real_if_close(c,tol=1000)  Return a real array if complex parts close to 0
>>> np.cast['f'](np.pi)        Cast object to a data type
```

Other Useful Functions

```
>>> np.angle(b,deg=True)       Return the angle of the complex argument
>>> g = np.linspace(0,np.pi,num=5)  Create an array of evenly spaced values
                                    (number of samples)
>>> g [3:] += np.pi
>>> np.unwrap(g)               Unwrap
>>> np.logspace(0,10,3)        Create an array of evenly spaced values (log scale)
>>> np.select([c<4],[c*2])     Return values from a list of arrays depending on
                               conditions
>>> misc.factorial(a)          Factorial
>>> misc.comb(10,3,exact=True) Combine N things taken at k time
>>> misc.central_diff_weights(3)  Weights for Np-point central derivative
>>> misc.derivative(myfunc,1.0)   Find the n-th derivative of a function at a point
```

Linear Algebra

Also see NumPy

You'll use the `linalg` and `sparse` modules. Note that `scipy.linalg` contains and expands on `numpy.linalg`.

```
>>> from scipy import linalg, sparse
```

Creating Matrices

```
>>> A = np.matrix(np.random.random((2,2)))
>>> B = np.asmatrix(b)
>>> C = np.mat(np.random.random((10,5)))
>>> D = np.mat([[3,4], [5,6]])
```

Basic Matrix Routines

Inverse

```
>>> A.I                        Inverse
>>> linalg.inv(A)              Inverse
```

Transposition

```
>>> A.T                        Tranpose matrix
>>> A.H                        Conjugate transposition
```

Trace

```
>>> np.trace(A)                Trace
```

Norm

```
>>> linalg.norm(A)             Frobenius norm
>>> linalg.norm(A,1)           L1 norm (max column sum)
>>> linalg.norm(A,np.inf)      L inf norm (max row sum)
```

Rank

```
>>> np.linalg.matrix_rank(C)   Matrix rank
```

Determinant

```
>>> linalg.det(A)              Determinant
```

Solving linear problems

```
>>> linalg.solve(A,b)          Solver for dense matrices
>>> E = np.mat(a).T            Solver for dense matrices
>>> linalg.lstsq(F,E)          Least-squares solution to linear matrix
                               equation
```

Generalized inverse

```
>>> linalg.pinv(C)             Compute the pseudo-inverse of a matrix
                               (least-squares solver)
>>> linalg.pinv2(C)            Compute the pseudo-inverse of a matrix
                               (SVD)
```

Creating Sparse Matrices

```
>>> F = np.eye(3, k=1)         Create a 2X2 identity matrix
>>> G = np.mat(np.identity(2)) Create a 2x2 identity matrix
>>> C[C > 0.5] = 0
>>> H = sparse.csr_matrix(C)   Compressed Sparse Row matrix
>>> I = sparse.csc_matrix(D)   Compressed Sparse Column matrix
>>> J = sparse.dok_matrix(A)   Dictionary Of Keys matrix
>>> E.todense()                Sparse matrix to full matrix
>>> sparse.isspmatrix_csc(A)   Identify sparse matrix
```

Sparse Matrix Routines

Inverse

```
>>> sparse.linalg.inv(I)       Inverse
```

Norm

```
>>> sparse.linalg.norm(I)      Norm
```

Solving linear problems

```
>>> sparse.linalg.spsolve(H,I) Solver for sparse matrices
```

Sparse Matrix Functions

```
>>> sparse.linalg.expm(I)      Sparse matrix exponential
```

Matrix Functions

Addition

```
>>> np.add(A,D)                Addition
```

Subtraction

```
>>> np.subtract(A,D)           Subtraction
```

Division

```
>>> np.divide(A,D)             Division
```

Multiplication

```
>>> A @ D                      Multiplication operator (Python 3)
>>> np.multiply(D,A)           Multiplication
>>> np.dot(A,D)                Dot product
>>> np.vdot(A,D)               Vector dot product
>>> np.inner(A,D)              Inner product
>>> np.outer(A,D)              Outer product
>>> np.tensordot(A,D)          Tensor dot product
>>> np.kron(A,D)               Kronecker product
```

Exponential Functions

```
>>> linalg.expm(A)             Matrix exponential
>>> linalg.expm2(A)            Matrix exponential (Taylor Series)
>>> linalg.expm3(D)            Matrix exponential (eigenvalue decomposition)
```

Logarithm Function

```
>>> linalg.logm(A)             Matrix logarithm
```

Trigonometric Functions

```
>>> linalg.sinm(D)             Matrix sine
>>> linalg.cosm(D)             Matrix cosine
>>> linalg.tanm(A)             Matrix tangent
```

Hyperbolic Trigonometric Functions

```
>>> linalg.sinhm(D)            Hyperbolic matrix sine
>>> linalg.coshm(D)            Hyperbolic matrix cosine
>>> linalg.tanhm(A)            Hyperbolic matrix tangent
```

Matrix Sign Function

```
>>> np.signm(A)                Matrix sign function
```

Matrix Square Root

```
>>> linalg.sqrtm(A)            Matrix square root
```

Arbitrary Functions

```
>>> linalg.funm(A, lambda x: x*x)  Evaluate matrix function
```

Decompositions

Eigenvalues and Eigenvectors

```
>>> la, v = linalg.eig(A)      Solve ordinary or generalized
                               eigenvalue problem for square matrix
>>> l1, l2 = la                Unpack eigenvalues
>>> v[:,0]                     First eigenvector
>>> v[:,1]                     Second eigenvector
>>> linalg.eigvals(A)          Unpack eigenvalues
```

Singular Value Decomposition

```
>>> U,s,Vh = linalg.svd(B)     Singular Value Decomposition (SVD)
>>> M,N = B.shape
>>> Sig = linalg.diagsvd(s,M,N)  Construct sigma matrix in SVD
```

LU Decomposition

```
>>> P,L,U = linalg.lu(C)       LU Decomposition
```

Sparse Matrix Decompositions

```
>>> la, v = sparse.linalg.eigs(F,1)  Eigenvalues and eigenvectors
>>> sparse.linalg.svds(H, 2)         SVD
```

Asking For Help

```
>>> help(scipy.linalg.diagsvd)
>>> np.info(np.matrix)
```

DataCamp
Learn Python for Data Science Interactively

APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

Python For Data Science *Cheat Sheet*

Matplotlib

Learn Python Interactively at www.DataCamp.com

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication-quality figures in a variety of hardcopy formats and interactive environments across platforms.

matplotlib

1 Prepare The Data

Also see Lists & NumPy

1D Data

```
>>> import numpy as np
>>> x = np.linspace(0, 10, 100)
>>> y = np.cos(x)
>>> z = np.sin(x)
```

2D Data or Images

```
>>> data = 2 * np.random.random((10, 10))
>>> data2 = 3 * np.random.random((10, 10))
>>> Y, X = np.mgrid[-3:3:100j, -3:3:100j]
>>> U = -1 - X**2 + Y
>>> V = 1 + X - Y**2
>>> from matplotlib.cbook import get_sample_data
>>> img = np.load(get_sample_data('axes_grid/bivariate_normal.npy'))
```

2 Create Plot

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

Figure

```
>>> fig = plt.figure()
>>> fig2 = plt.figure(figsize=plt.figaspect(2.0))
```

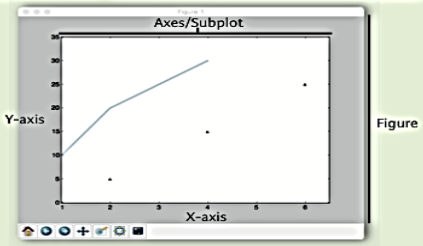
Axes

All plotting is done with respect to an `Axes`. In most cases, a subplot will fit your needs. A subplot is an axes on a grid system.

```
>>> fig.add_axes()
>>> ax1 = fig.add_subplot(221) # row-col-num
>>> ax3 = fig.add_subplot(212)
>>> fig3, axes = plt.subplots(nrows=2,ncols=2)
>>> fig4, axes2 = plt.subplots(ncols=3)
```

Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare data 2 Create plot 3 Plot 4 Customize plot 5 Save plot 6 Show plot

```
>>> import matplotlib.pyplot as plt
>>> x = [1,2,3,4]                          Step 1
>>> y = [10,20,25,30]
>>> fig = plt.figure()                     Step 2
>>> ax = fig.add_subplot(111)              Step 3
>>> ax.plot(x, y, color='lightblue', linewidth=3)   Step 3, 4
>>> ax.scatter([2,4,6],
               [5,15,25],
               color='darkgreen',
               marker='^')
>>> ax.set_xlim(1, 6.5)
>>> plt.savefig('foo.png')
>>> plt.show()                             Step 6
```

4 Customize Plot

Colors, Color Bars & Color Maps

```
>>> plt.plot(x, x, x, x**2, x, x**3)
>>> ax.plot(x, y, alpha = 0.4)
>>> ax.plot(x, y, c='k')
>>> fig.colorbar(im, orientation='horizontal')
>>> im = ax.imshow(img,
                   cmap='seismic')
```

Markers

```
>>> fig, ax = plt.subplots()
>>> ax.scatter(x,y,marker=".")
>>> ax.plot(x,y,marker="o")
```

Linestyles

```
>>> plt.plot(x,y,linewidth=4.0)
>>> plt.plot(x,y,ls='solid')
>>> plt.plot(x,y,ls='--')
>>> plt.plot(x,y,'--',x**2,y**2,'-.')
>>> plt.setp(lines,color='r',linewidth=4.0)
```

Text & Annotations

```
>>> ax.text(1,
            -2.1,
            'Example Graph',
            style='italic')
>>> ax.annotate("Sine",
                xy=(8, 0),
                xycoords='data',
                xytext=(10.5, 0),
                textcoords='data',
                arrowprops=dict(arrowstyle="->",
                                connectionstyle="arc3"),)
```

Mathtext

```
>>> plt.title(r'$sigma_i=15$', fontsize=20)
```

Limits, Legends & Layouts

Limits & Autoscaling

```
>>> ax.margins(x=0.0,y=0.1)          # Add padding to a plot
>>> ax.axis('equal')                 # Set the aspect ratio of the plot to 1
>>> ax.set(xlim=[0,10.5],ylim=[-1.5,1.5])   # Set limits for x-and y-axis
>>> ax.set_xlim(0,10.5)              # Set limits for x-axis
```

Legends

```
>>> ax.set(title='An Example Axes',          # Set a title and x-and y-axis labels
           ylabel='Y-Axis',
           xlabel='X-Axis')
>>> ax.legend(loc='best')            # No overlapping plot elements
```

Ticks

```
>>> ax.xaxis.set(ticks=range(1,5),           # Manually set x-ticks
                 ticklabels=[3,100,-12,"foo"])
>>> ax.tick_params(axis='y',                 # Make y-ticks longer and go in and out
                   direction='inout',
                   length=10)
```

Subplot Spacing

```
>>> fig3.subplots_adjust(wspace=0.5,         # Adjust the spacing between subplots
                         hspace=0.3,
                         left=0.125,
                         right=0.9,
                         top=0.9,
                         bottom=0.1)
>>> fig.tight_layout()               # Fit subplot(s) in to the figure area
```

Axis Spines

```
>>> ax1.spines['top'].set_visible(False)     # Make the top axis line for a plot invisible
>>> ax1.spines['bottom'].set_position(('outward',10))   # Move the bottom axis line outward
```

3 Plotting Routines

1D Data

```
>>> fig, ax = plt.subplots()
>>> lines = ax.plot(x,y)          # Draw points with lines or markers connecting them
>>> ax.scatter(x,y)               # Draw unconnected points, scaled or colored
>>> axes[0,0].bar([1,2,3],[3,4,5])          # Plot vertical rectangles (constant width)
>>> axes[1,0].barh([0.5,1,2.5],[0,1,2])     # Plot horiontal rectangles (constant height)
>>> axes[1,1].axhline(0.45)       # Draw a horizontal line across axes
>>> axes[0,1].axvline(0.65)       # Draw a vertical line across axes
>>> ax.fill(x,y,color='blue')     # Draw filled polygons
>>> ax.fill_between(x,y,color='yellow')     # Fill between y-values and o
```

Vector Fields

```
>>> axes[0,1].arrow(0,0,0.5,0.5)    # Add an arrow to the axes
>>> axes[1,1].quiver(y,z)           # Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V)   # Plot a 2D field of arrows
```

Data Distributions

```
>>> ax1.hist(y)                     # Plot a histogram
>>> ax3.boxplot(y)                  # Make a box and whisker plot
>>> ax3.violinplot(z)               # Make a violin plot
```

2D Data or Images

```
>>> fig, ax = plt.subplots()
>>> im = ax.imshow(img,                     # Colormapped or RGB arrays
                   cmap='gist_earth',
                   interpolation='nearest',
                   vmin=-2,
                   vmax=2)
```

```
>>> axes2[0].pcolor(data2)          # Pseudocolor plot of 2D array
>>> axes2[0].pcolormesh(data)       # Pseudocolor plot of 2D array
>>> CS = plt.contour(Y,X,U)         # Plot contours
>>> axes2[2].contourf(data1)        # Plot filled contours
>>> axes2[2]= ax.clabel(CS)         # Label a contour plot
```

5 Save Plot

Save figures

```
>>> plt.savefig('foo.png')
```

Save transparent figures

```
>>> plt.savefig('foo.png', transparent=True)
```

6 Show Plot

```
>>> plt.show()
```

Close & Clear

```
>>> plt.cla()          # Clear an axis
>>> plt.clf()          # Clear the entire figure
>>> plt.close()        # Close a window
```

DataCamp
Learn Python for Data Science Interactively

APTIKOM
ASOSIASI PERGURUAN TINGGI INFORMATIKA DAN KOMPUTER

- Link buku-buku
  - **Big-data dan Data Science**: https://drive.google.com/drive/folders/18jbNHjUWsRor8W64oNDxggOd_yWqMzHs?usp=sharing
  - **Deep Learning** dan **Machine Learning**: https://drive.google.com/drive/folders/1hJ-E5OJhg35R7LC7_bHy99ccoY7CJ3nO?usp=sharing
  - **Python:** https://drive.google.com/drive/folders/1zqr5GPjQhP96XqKcWMxcmeWiMAAZ1iVx?usp=sharing

scikit-learn

- scikit-learn user guide, Mar 01, 2019 : https://drive.google.com/drive/folders/1rRsU6WdnPUlT3d9NcsTuk2f6N92PLZkc?usp=sharing