Term: L2-ElnI

Semester: 4

AY: 2020-2021

Abdelbacet Mhamdi

Dr.-Ing. in Electrical Engineering

Senior Lecturer at ISET Bizerte

abdelbacet.mhamdi@bizerte.r-iset.tn

Machine Learning

LAB MANUAL

Institut Supérieur des Études Technologiques de Bizerte

Available at https://github.com/a-mhamdi/isetbz/

Honor code

The University of North Carolina at Chapel Hill
Department of Physics and Astronomy
http://physics.unc.edu/undergraduate-program/labs/general-info/

"During this course, you will be working with one or more partners with whom you may discuss any points concerning laboratory work. However, you must write your lab report, in your own words.

Lab reports that contain identical language are not acceptable, so do not copy your lab partner's writing.

If there is a problem with your data, include an explanation in your report. Recognition of a mistake and a well-reasoned explanation is more important than having high-quality data, and will be rewarded accordingly by your instructor. A lab report containing data that is inconsistent with the original data sheet will be considered a violation of the Honor Code.

Falsification of data or plagiarism of a report will result in prosecution of the offender(s) under the University Honor Code.

On your first lab report you must write out the entire honor pledge:

The work presented in this report is my own, and the data was obtained by my lab partner and me during the lab period.

On future reports, you may simply write "*Laboratory Honor Pledge*" and sign your name."

Contents

In order to activate the virtual environment and launch **Jupyter Notebook**, we recommend you to proceed as follow

① Press simultaneously the keys  &  on the keyboard. This will open the dialog box **Run**;

② Then enter **cmd** in the command line and confirm with  key on the keyboard;

③ Type the instruction **ml.bat** in the console prompt line;

④ Finally press the key.

Leave the system console active.

1 | Getting Started with `Python`

Student's name	. . . . .	. . . . .	. . . . .
	. . . . .	. . . . .	. . . . .
	. . . . .	. . . . .	. . . . .
Score /20	. . . . .	. . . . .	. . . . .

Detailed Credits

Anticipation *(4 points)*	. . . . .	. . . . .	. . . . .
Management *(2 points)*	. . . . .	. . . . .	. . . . .
Testing *(7 points)*	. . . . .	. . . . .	. . . . .
Data Logging *(3 points)*	. . . . .	. . . . .	. . . . .
Interpretation *(4 points)*	. . . . .	. . . . .	. . . . .

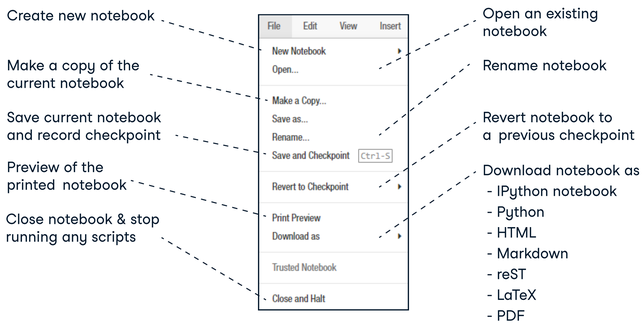
A full list of Cheat Sheets is accessible at https://www.datacamp.com/community/data-science-cheatsheets.

datacamp

Python For Data Science
Jupyter Cheat Sheet

Learn Jupyter online at www.DataCamp.com

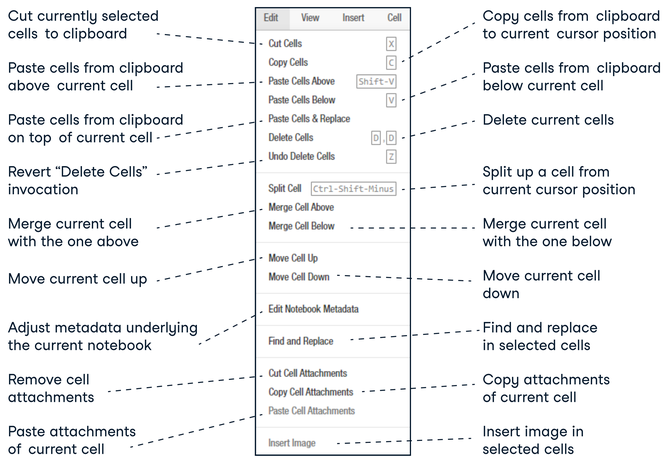
> Saving/Loading Notebooks



> Writing Code And Text

Code and text are encapsulated by 3 basic cell types: markdown cells, code cells, and raw NBConvert cells

Edit Cells



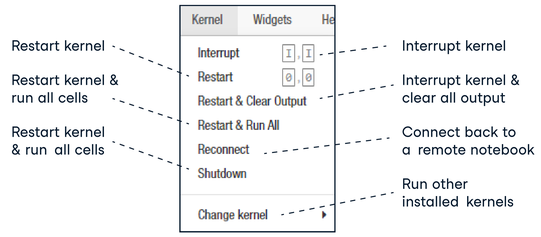
Insert Cells

> Working with Different Programming Languages

Kernels provide computation and communication with front-end interfaces like the notebooks. There are three main kernels:

IPython IRkernel IJulia

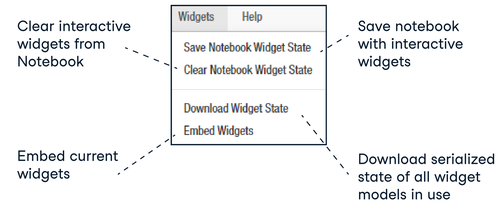
Installing Jupyter Notebook will automatically install the IPython kernel.



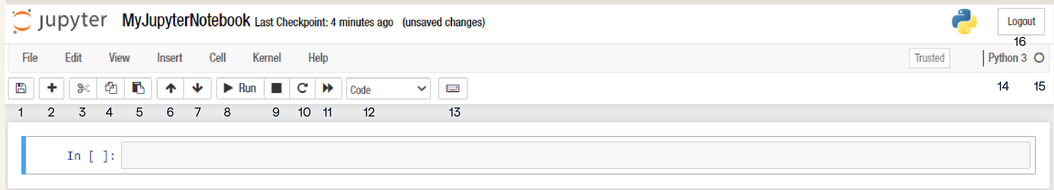
> Widgets

Notebook widgets provide the ability to visualize and control changes in your data, often as a control like a slider, textbox, etc.

You can use them to build interactive GUIs for your notebooks or to synchronize stateful and stateless information between Python and JavaScript.

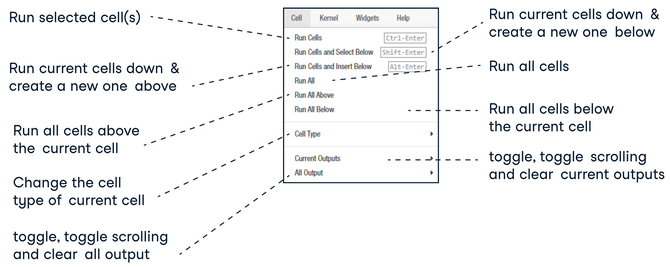


Command Mode:

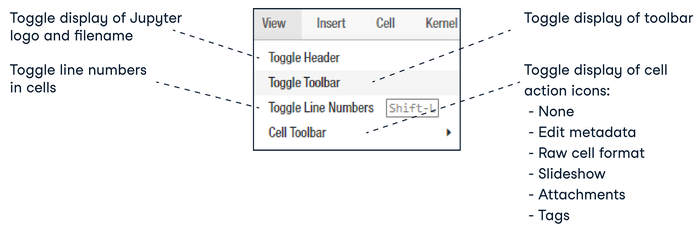


Edit Mode:

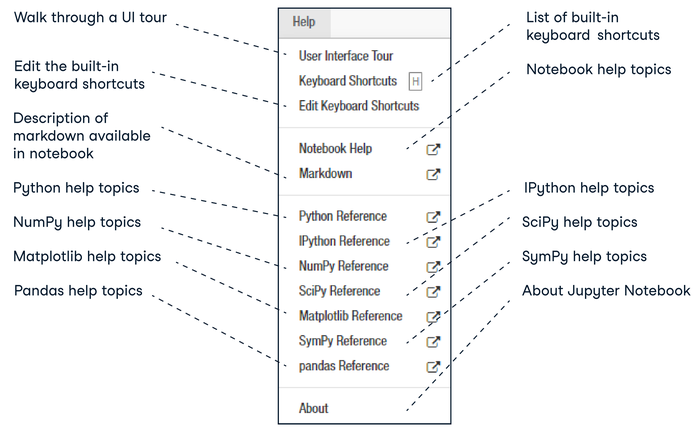
Executing Cells



View Cells



Asking For Help



datacamp

Python For Data Science

python™ Basics Cheat Sheet

Learn Python Basics online at www.DataCamp.com

> Variables and Data Types

Variable Assignment

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

Calculations With Variables

```
>>> x+2 #Sum of two variables
 7
>>> x-2 #Subtraction of two variables
 3
>>> x*2 #Multiplication of two variables
 10
>>> x**2 #Exponentiation of a variable
 25
>>> x%2 #Remainder of a variable
 1
>>> x/float(2) #Division of a variable
 2.5
```

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
```

> Libraries

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

Import Libraries

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

Selective import

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

> Strings

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

String Operations

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

String Indexing

Index starts at 0

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

String Methods

```
>>> my_string.upper() #String to uppercase
>>> my_string.lower() #String to lowercase
>>> my_string.count('w') #Count String elements
>>> my_string.replace('e', 'i') #Replace String elements
>>> my_string.strip() #Strip whitespaces
```

> 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] #Select item at index 1
 2
```

Slice

```
>>> my_array[0:2] #Select items at index 0 and 1
 array([1, 2])
```

Subset 2D Numpy arrays

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

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 #Get the dimensions of the array
>>> np.append(other_array) #Append items to an array
>>> np.insert(my_array, 1, 5) #Insert items in an array
>>> np.delete(my_array,[1]) #Delete items in an array
>>> np.mean(my_array) #Mean of the array
>>> np.median(my_array) #Median of the array
>>> my_array.corrcoef() #Correlation coefficient
>>> np.std(my_array) #Standard deviation
```

> 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] #Select item at index 1
>>> my_list[-3] #Select 3rd last item
```

Slice

```
>>> my_list[1:3] #Select items at index 1 and 2
>>> my_list[1:] #Select items after index 0
>>> my_list[:3] #Select items before index 3
>>> my_list[:] #Copy my_list
```

Subset Lists of Lists

```
>>> my_list2[1][0] #my_list[list][itemOfList]
>>> my_list2[1][:2]
```

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) #Get the index of an item
>>> my_list.count(a) #Count an item
>>> my_list.append('!') #Append an item at a time
>>> my_list.remove('!') #Remove an item
>>> del(my_list[0:1]) #Remove an item
>>> my_list.reverse() #Reverse the list
>>> my_list.extend('!') #Append an item
>>> my_list.pop(-1) #Remove an item
>>> my_list.insert(0,'!') #Insert an item
>>> my_list.sort() #Sort the list
```

> Python IDEs (Integrated Development Environment)

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

> Asking For Help

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

datacamp

Python For Data Science
NumPy Cheat Sheet

Learn NumPy online at www.DataCamp.com

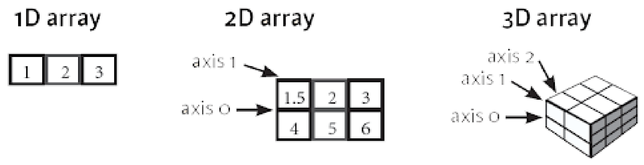
Numpy

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



> 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=" ")
```

> Asking For Help

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

> 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
```

> 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
```

> 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

Subsetting

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

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 (equivalent to b[0:1, :])
 array([[1.5, 2., 3.]])
>>> 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 and columns
 array([[ 4. ,5. , 6. , 4. ],
        [ 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 index
 [array([1]),array([2]),array([3])]
>>> 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

Python For Data Science

Matplotlib Cheat Sheet

Learn Matplotlib online 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.

> Prepare The Data

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'))
```

> 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)
```

> Save Plot

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

> Show Plot

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

> 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 0
```

2D Data

```
>>> 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
```

Vector Fields

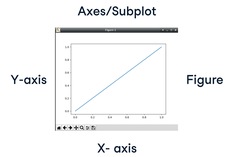
```
>>> axes[0,1].arrow(0,0,0.5,0.5) #Add an arrow to the axes
>>> axes[1,1].quiver(y,z) #Plot a 2D field of arrows
>>> axes[0,1].streamplot(X,Y,U,V) #Plot a 2D field of arrows
```

Data Distributions

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

> Plot Anatomy & Workflow

Plot Anatomy



Workflow

The basic steps to creating plots with matplotlib are:

1 Prepare Data 2 Create Plot 3 Plot 4 Customized 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') #Step 5
>>> plt.show() #Step 6
```

> Close and Clear

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

> Plotting Cutomize 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 and 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
```

datacamp

Python For Data Science

Pandas Basics Cheat Sheet

Learn Pandas Basics online 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

Index →

a	3
b	-5
c	7
d	4

```
>>> 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

Columns →

Index →	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasilia	207847528

```
>>> data = {'Country': ['Belgium', 'India', 'Brazil'],
            'Capital': ['Brussels', 'New Delhi', 'Brasília'],
            'Population': [11190846, 1303171035, 207847528]}
>>> df = pd.DataFrame(data,
                      columns=['Country', 'Capital', 'Population'])
```

> Dropping

```
>>> s.drop(['a', 'c']) #Drop values from rows (axis=0)
>>> df.drop('Country', axis=1) #Drop values from columns(axis=1)
```

> Asking For Help

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

> Sort & Rank

```
>>> df.sort_index() #Sort by labels along an axis
>>> df.sort_values(by='Country') #Sort by the values along an axis
>>> df.rank() #Assign ranks to entries
```

> I/O

Read and Write to CSV

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

Read and Write to Excel

```
>>> pd.read_excel('file.xlsx')
>>> df.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()

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

> Selection

Also see NumPy Arrays

Getting

```
>>> s['b'] #Get one element
 -5
>>> df[1:] #Get subset of a DataFrame
   Country Capital Population
 1 India New Delhi 1303171035
 2 Brazil Brasília 207847528
```

Selecting, Boolean Indexing & Setting

By Position

```
>>> df.iloc[[0],[0]] #Select single value by row & column
 'Belgium'
>>> df.iat([0],[0])
 'Belgium'
```

By Label

```
>>> df.loc[[0], ['Country']] #Select single value by row & column labels
 'Belgium'
>>> df.at([0], ['Country'])
 'Belgium'
```

By Label/Position

```
>>> df.ix[2] #Select single row of subset of rows
 Country Brazil
 Capital Brasília
 Population 207847528
>>> df.ix[:,'Capital'] #Select a single column of subset of columns
 0 Brussels
 1 New Delhi
 2 Brasília
>>> df.ix[1,'Capital'] #Select rows and columns
 'New Delhi'
```

Boolean Indexing

```
>>> s[~(s > 1)] #Series s where value is not >1
>>> s[(s < -1) | (s > 2)] #s where value is <-1 or >2
>>> df[df['Population']>1200000000] #Use filter to adjust DataFrame
```

Setting

```
>>> s['a'] = 6 #Set index a of Series s to 6
```

> Retrieving Series/DataFrame Information

Basic Information

```
>>> df.shape #(rows,columns)
>>> df.index #Describe index
>>> df.columns #Describe DataFrame columns
>>> df.info() #Info on DataFrame
>>> df.count() #Number of non-NA values
```

Summary

```
>>> df.sum() #Sum of values
>>> df.cumsum() #Cummulative sum of values
>>> df.min()/df.max() #Minimum/maximum values
>>> df.idxmin()/df.idxmax() #Minimum/Maximum index value
>>> df.describe() #Summary statistics
>>> df.mean() #Mean of values
>>> df.median() #Median of values
```

> Applying Functions

```
>>> f = lambda x: x*2
>>> df.apply(f) #Apply function
>>> df.applymap(f) #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_values=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)
```

datacamp

Python For Data Science
Scikit-Learn Cheat Sheet

Learn Scikit-Learn online 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.model_selection 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.model_selection import train_test_split
>>> X_train, X_test, y_train, y_test = train_test_split(X,
                                                        y,
                                                        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)
```

> 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)
```

> Evaluate Your Model's Performance

Classification Metrics

Accuracy Score

```
>>> knn.score(X_test, y_test) #Estimator score method
>>> from sklearn.metrics import accuracy_score #Metric scoring functions
>>> accuracy_score(y_test, y_pred)
```

Classification Report

```
>>> from sklearn.metrics import classification_report #Precision, recall, f1-score and support
>>> 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

Python For Data Science
Keras Cheat Sheet

Learn Keras online at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library forTheano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

> Data

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing, mnist, cifar10, imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data =
np.loadtxt(urlopen("http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

> Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

> Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

> Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X, y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

> Inspect Model

```
>>> model.output_shape #Model output shape
>>> model.summary() #Model summary representation
>>> model.get_config() #Model configuration
>>> model.get_weights() #List all weight tensors in the model
```

> Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

> Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

> Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

> Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

> Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```

2 | Linear Regression

Student's name	.....	.....	.....
	.....	.....	.....
	.....	.....	.....
Score /20	.....	.....	.....

Detailed Credits

Anticipation *(4 points)*	.....	.....	.....
Management *(2 points)*	.....	.....	.....
Testing *(7 points)*	.....	.....	.....
Data Logging *(3 points)*	.....	.....	.....
Interpretation *(4 points)*	.....	.....	.....

The notebook is available at https://github.com/a-mhamdi/cosnip/ → Python → ml → linear-regression.ipynb

Simple Linear Regression using artificial data

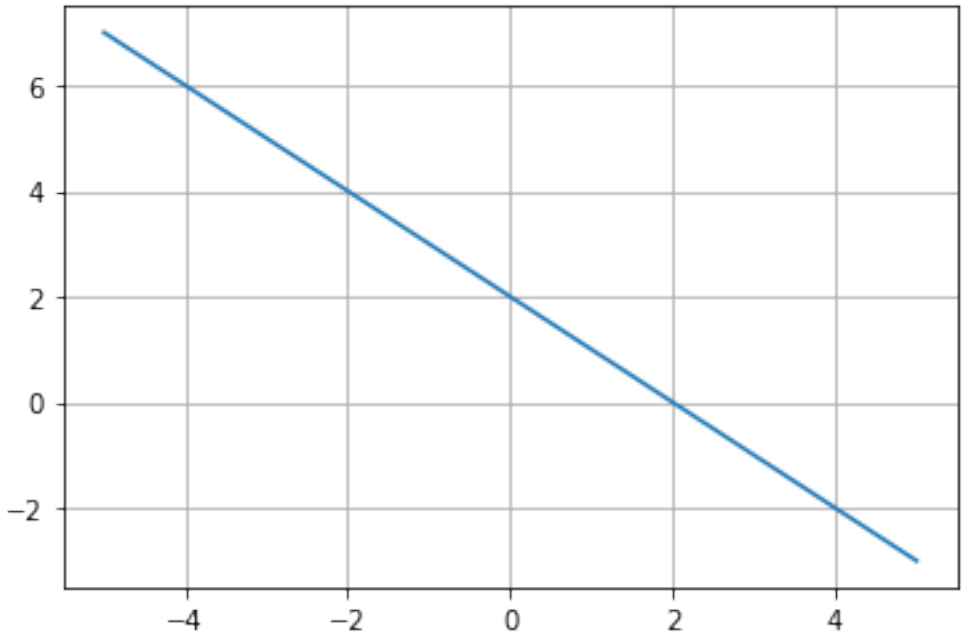
[1]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
```

Consider the following equation:

$$y = -x + 2 \quad \longrightarrow \begin{cases} a = -1 \\ b = 2 \end{cases}$$

[2]:
```
x = np.linspace(-5,5,100)
y = -x + 2
```

[3]:
```
plt.plot(x, y)
plt.grid()
```



A full description of the available methods can be retrieved from the official website of scikit-learn.

Syntax	Description
`fit(X, y[, sample_weight])`	Fit linear model.
`get_params([deep])`	Get parameters for this estimator.
`predict(X)`	Predict using the linear model.
`score(X, y[, sample_weight])`	Return the coefficient of determination of the prediction.
`set_params(**params)`	Set the parameters of this estimator.

[4]:
```
x = x.reshape(-1,1)
linear_reg = LinearRegression().fit(x, y)
```

[5]:
```
linear_reg.score(x, y)
```

[5]:
```
1.0
```

Coefficient a

[6]:
```
linear_reg.coef_
```

[6]:
```
array([-1.])
```

Intercept b

[7]:
```
linear_reg.intercept_
```

[7]:
```
2.0
```

Let's check for $x = 5$.

[8]:
```
linear_reg.predict(np.array([5]).reshape(-1,1))
```

[8]:
```
array([-3.])
```

Simple Linear Regression using dataset

[9]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
```

Load the datasets.

[10]:
```
dataset = pd.read_csv("./datasets/Weight_Height.csv")
```

Check the dataset.

[11]:
```
dataset.head()
```

[11]:

	Gender	Height	Weight
0	Male	73.847017	241.893563
1	Male	68.781904	162.310473
2	Male	74.110105	212.740856
3	Male	71.730978	220.042470
4	Male	69.881796	206.349801

Check the dimensions of the loaded dataset.

[12]:
```
dataset.shape
```

[12]:
```
(10000, 3)
```

Check if there are null values in the dataset.

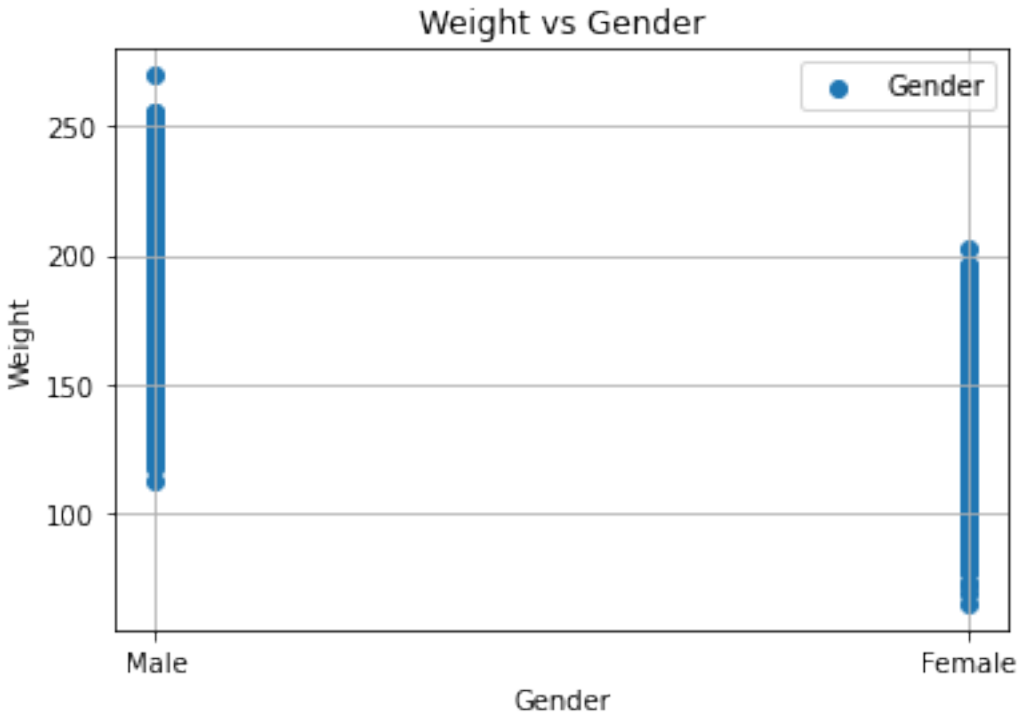
[13]:
```
dataset.isnull().sum()
```

[13]:
```
Gender    0
Height    0
Weight    0
dtype: int64
```

Plot *Gender* vs *Weight*.

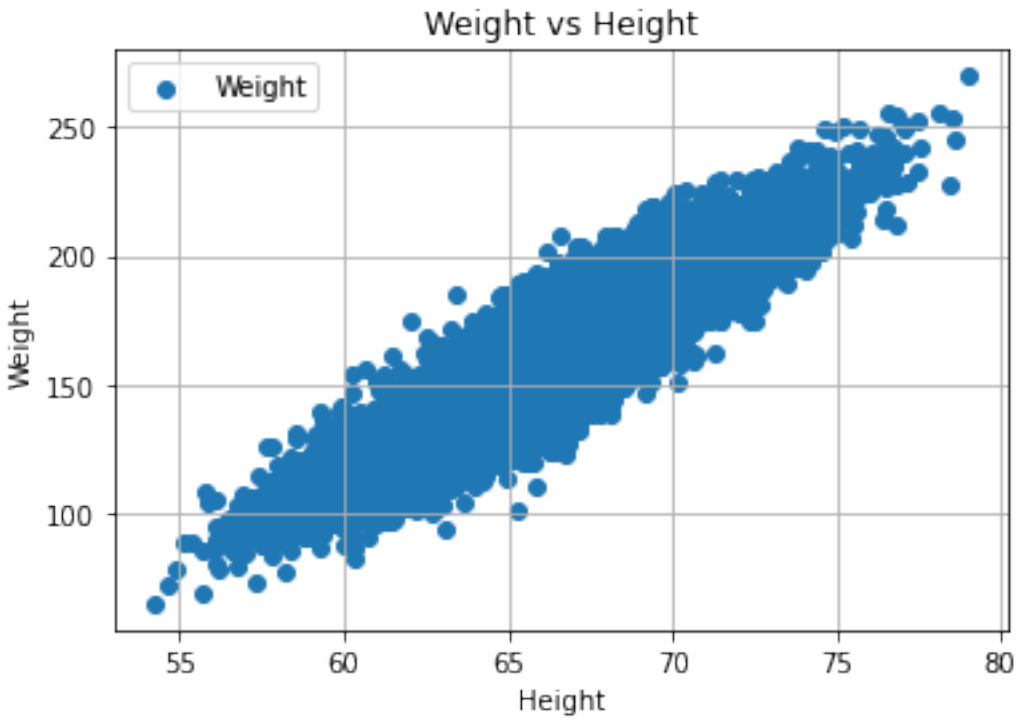
[14]:
```
x1 = dataset.iloc[:, 0].values
y1 = dataset.iloc[:, 2].values
plt.scatter(x1, y1, label='Gender')
plt.xlabel('Gender')
plt.ylabel('Weight')
plt.title('Weight vs Gender')
```

```
plt.grid()
plt.legend()
```



Plot *Height* vs *Weight*.

[15]:
```
x2 = dataset.iloc[:, 1].values
y2 = dataset.iloc[:, 2].values
plt.scatter(x2,y2,label='Weight')
plt.xlabel('Height')
plt.ylabel('Weight')
plt.title('Weight vs Height')
plt.grid()
plt.legend()
```



[16]:
```
X = dataset.iloc[:, 1].values
```

Target values y

[17]:
```
y = dataset.iloc[:, 2].values
```

[18]:
```
X_train, X_test, y_train, y_test = train_test_split(X.reshape(-1,1), y,␣
↪test_size=0.2, random_state=123)
```

Create linear regression model.

[19]:
```
w_h_regressor = LinearRegression()
w_h_regressor.fit(X_train, y_train)
```

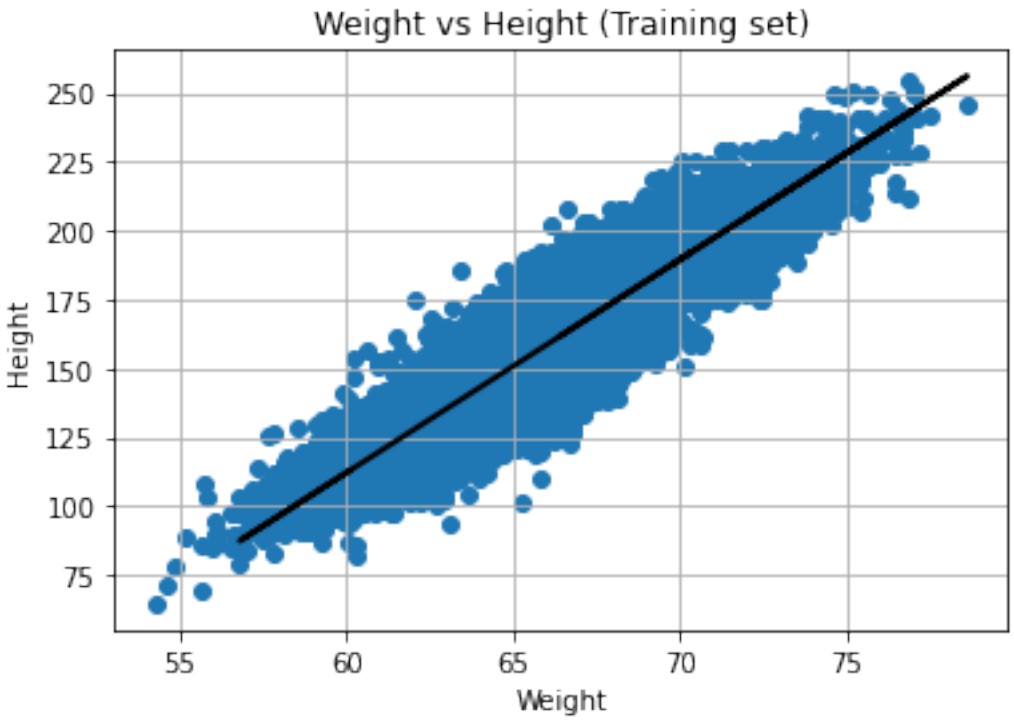
[19]:
```
LinearRegression()
```

Predict the training set.

[20]:
```
y_pred = w_h_regressor.predict(X_train)
```

Display the training set results

[21]:
```
plt.scatter(X_train, y_train)
plt.plot(X_train, y_pred, color='black', linewidth=2)
plt.title('Weight vs Height (Training set)')
plt.xlabel('Weight')
plt.ylabel('Height')
plt.grid()
```

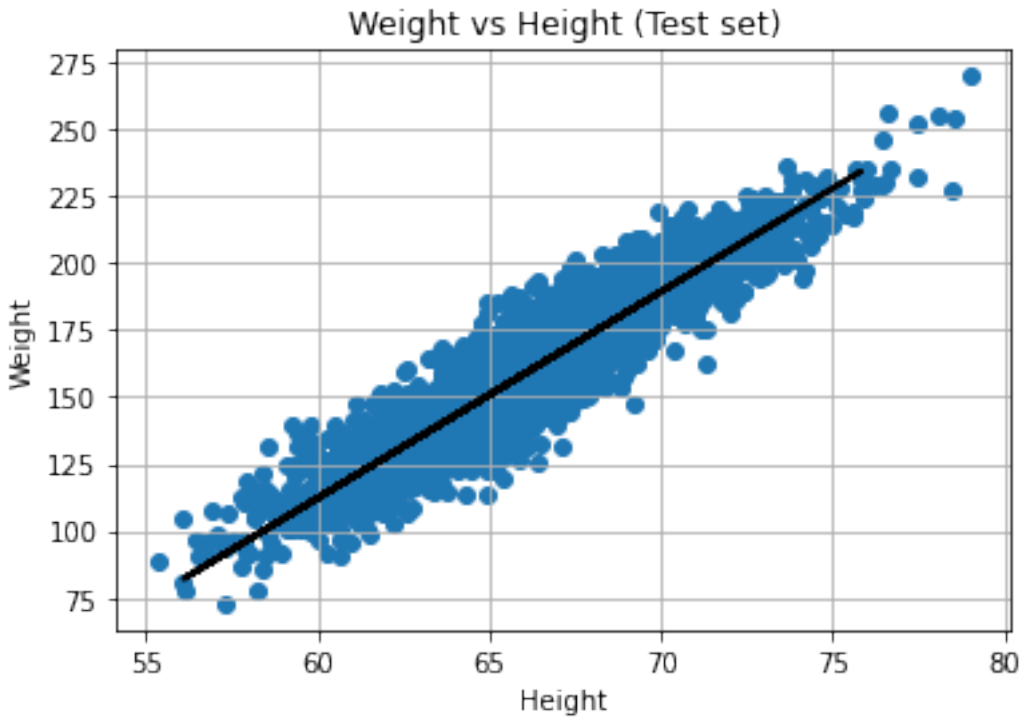


Predict the test set.

[22]:
```
y_pred = w_h_regressor.predict(X_test)
```

Display the test set results.

[23]:
```
plt.scatter(X_test, y_test)
plt.plot(X_test, y_pred, color='black', linewidth=2)
plt.title('Weight vs Height (Test set)')
plt.xlabel('Height')
plt.ylabel('Weight')
plt.grid()
```



Overall evaluation of the model

[24]:
```
print('Coefficients: ', w_h_regressor.coef_)
```

```
Coefficients:  [7.72896259]
```

The mean squared error

[25]:
```
print('Mean squared error is {}.'.format(np.mean((y_pred - y_test)** 2)))
```

```
Mean squared error is 143.22556010111649.
```

The more variance score approaches to 1, the more perfect is the prediction.

[26]:
```
print('Variance score is {}.'.format(w_h_regressor.score(X_test, y_test)))
```

```
Variance score is 0.8649031737206692.
```

3 KNN for Classification

Student's name	....	....	....
Score /20	.....	.....	.....

Detailed Credits

Anticipation *(4 points)*	.....	.....	.....
Management *(2 points)*	.....	.....	.....
Testing *(7 points)*	.....	.....	.....
Data Logging *(3 points)*	.....	.....	.....
Interpretation *(4 points)*	.....	.....	.....

The notebook is available at https://github.com/a-mhamdi/cosnip/ → Python → ml → clf-knn.ipynb

Load the necessary python modules

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Load the datasets

[2]:
```
df = pd.read_csv('./datasets/Diabetes.csv')
```

Print the first 5 rows of the dataframe.

[3]:
```
df.head()
```

[3]:
```
   Pregnancies  Glucose  Diastolic  Triceps  Insulin   BMI    DPF  Age  \
0            6      148         72       35        0  33.6  0.627   50
1            1       85         66       29        0  26.6  0.351   31
2            8      183         64        0        0  23.3  0.672   32
3            1       89         66       23       94  28.1  0.167   21
4            0      137         40       35      168  43.1  2.288   33

   Diabetes
0         1
1         0
```

```
2          1
3          0
4          1
```

Let's observe the shape of the dataframe.

[4]:
```
df.shape
```

[4]:
```
(768, 9)
```

Let's extract the features and target as numpy arrays.

[5]:
```
X = df.drop('Diabetes',axis=1).values
y = df['Diabetes'].values
```

Split the data into two sets: train and test. We begin by importing the `train_test_split` from `sklearn` module.

[6]:
```
from sklearn.model_selection import train_test_split
```

[7]:
```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
↪4,random_state=42, stratify=y)
```

It is time now to create a classifier using k-Nearest Neighbors algorithm. At first, the class `KNeighborsClassifier` has to be loaded.

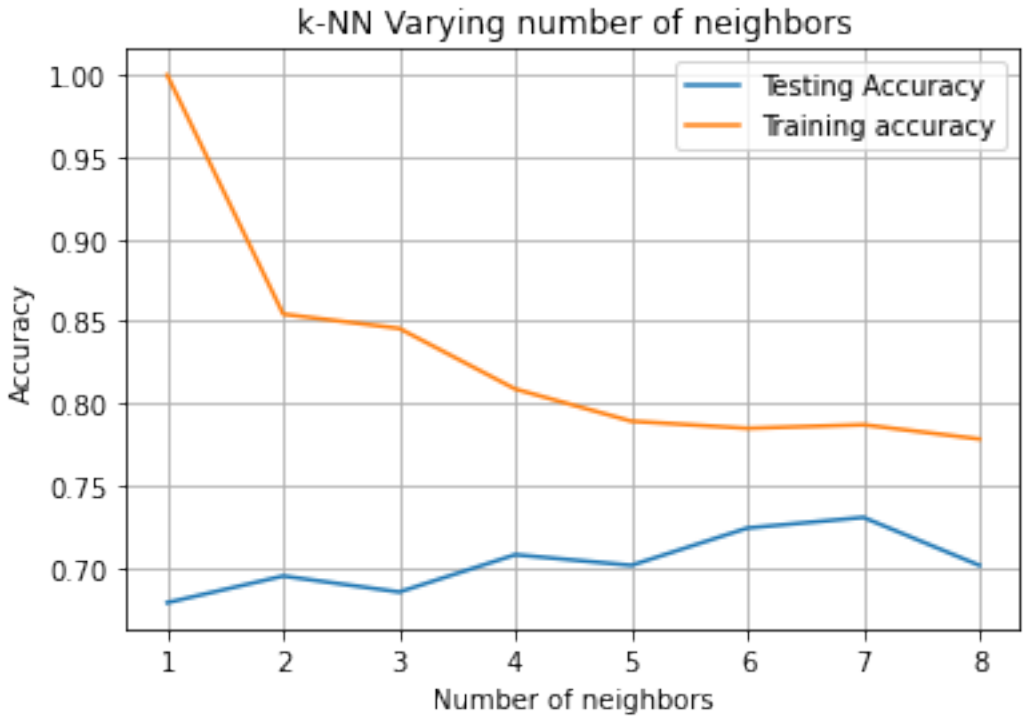
[8]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[9]:
```
# Setup arrays to store training and test accuracies
neighbors = np.arange(1,9)
train_accuracy =np.empty(len(neighbors))
test_accuracy = np.empty(len(neighbors))

for i,k in enumerate(neighbors):
    # Setup a knn classifier with k neighbors
    knn = KNeighborsClassifier(n_neighbors=k)

    # Fit the model
    knn.fit(X_train, y_train)

    # Compute accuracy on the training set
    train_accuracy[i] = knn.score(X_train, y_train)

    # Compute accuracy on the test set
    test_accuracy[i] = knn.score(X_test, y_test)
```

Generate some plots to see which number of neighbors is well suited for the classification task.

[10]:
```
plt.title('k-NN Varying number of neighbors')
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend()
plt.xlabel('Number of neighbors')
plt.ylabel('Accuracy')
```

```
plt.grid()
```



As shown above, 7 neighbors seem to be a suitable choice. So, let's setup a knn classifier with only $k = 7$ neighbors.

[11]:
```
knn = KNeighborsClassifier(n_neighbors=7)
```

Fit the model.

[12]:
```
knn.fit(X_train,y_train)
```

[12]:
```
KNeighborsClassifier(n_neighbors=7)
```

It is always a good manner to gather some score metrics.

[13]:
```
knn.score(X_test,y_test)
```

[13]:
```
0.7305194805194806
```

Import `confusion_matrix`

[14]:
```
from sklearn.metrics import confusion_matrix
```

Let's make some predictions using the classifier we built earlier.

[15]:
```
y_pred = knn.predict(X_test)
```

[16]:
```
confusion_matrix(y_test,y_pred)
```

[16]:
```
array([[165,  36],
       [ 47,  60]])
```

A fancy way to display the confusion matrix, is to use the `crosstab` method.

[17]:
```
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'],␣
↪margins=True)
```

[17]:

Predicted	0	1	All
True			
0	165	36	201
1	47	60	107
All	212	96	308

By importing `classification_report`, we can get some insights on how the model behaves.

[18]:
```
from sklearn.metrics import classification_report
```

As a reminder, **F1-Score**, **Accuracy**, **Recall** and **Precision** are calculated as follow:

$$f1-\text{score} = \frac{2}{\frac{1}{\text{Recall}} + \frac{1}{\text{Precision}}}$$

$f1-$score denotes the *Harmonic Mean of Recall & Precision*

$$\text{Accuracy} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{FP}+\text{TN}+\text{FN}}$$

It denotes the ratio of how much we got right over all cases. Recall, on the other hand, designates the ratio of how much positives do we got right over all actual positive cases.

$$\text{Recall} = \frac{\text{TP}}{\text{TP}+\text{FN}}$$

Precision, at last, is how much positives we got right over all positive predictions. It is given by:

$$\text{Precision} = \frac{\text{TP}}{\text{TP}+\text{FP}}$$

[19]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.78      0.82      0.80       201
           1       0.62      0.56      0.59       107

    accuracy                           0.73       308
   macro avg       0.70      0.69      0.70       308
weighted avg       0.73      0.73      0.73       308
```

4 | Support Vector Machine Classifier

Student's name	....	....	....
Score /20	.....	.....	.....

Detailed Credits

Anticipation *(4 points)*	.....	.....	.....
Management *(2 points)*	.....	.....	.....
Testing *(7 points)*	.....	.....	.....
Data Logging *(3 points)*	.....	.....	.....
Interpretation *(4 points)*	.....	.....	.....

The notebook is available at https://github.com/a-mhamdi/cosnip/ → Python → ml → clf-svm.ipynb

Support vector machine or **SVM** for short is also known as a discriminator classifier. The concepts are relatively simple. Using a hyperplane with the largest possible margin, the classifier separates data points. The **SVM** classifier finds an optimal hyperplane which allows classifying new data points much more accurately compared to other classifiers such as logistic regression, knn, etc.

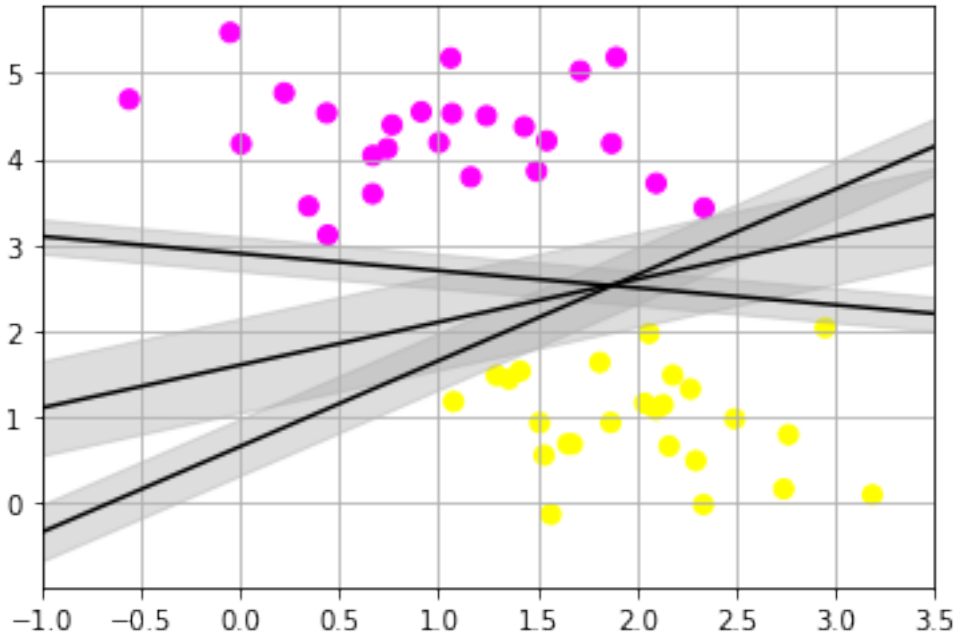
[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

Load the datasets `make_blobs` from the `datasets` module of the `sklearn` library.

[2]:
```
from sklearn.datasets import make_blobs
```

[3]:
```
X, y = make_blobs(n_samples=50, centers=2, random_state=0, cluster_std=0.60)

xfit = np.linspace(-1, 3.5)
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='spring')

# Draw three lines that couple separate the data
for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
    yfit = m * xfit + b
    plt.plot(xfit, yfit, '-k')
    plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',␣
 ↪color='#AAAAAA', alpha=0.4)
```

```
plt.xlim(-1, 3.5);
plt.grid()
```



Import the **Support Vector Machine Classifier**.

[4]:
```
from sklearn.svm import SVC
```

A full description of the available methods can be retrieved from the official website of scikit-learn.

Syntax	Description
`decision_function(X)`	Evaluates the decision function for the samples in X.
`fit(X, y[, sample_weight])`	Fit the SVM model according to the given training data.
`get_params([deep])`	Get parameters for this estimator.
`predict(X)`	Perform classification on samples in X.
`score(X, y[, sample_weight])`	Return the mean accuracy on the given test data and labels.
`set_params(**params)`	Set the parameters of this estimator.

We will choose a linear kernel as the distribution of the points can be linearly separated.

[5]:
```
clf = SVC(kernel='linear')
```

Fit to data the **SVM** classifier, denoted here by `clf`.

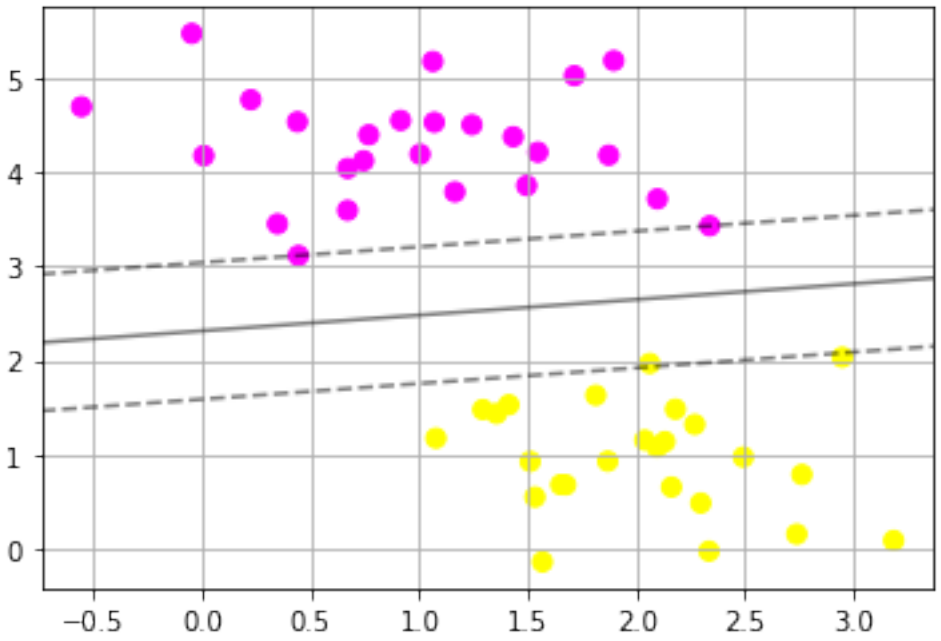
[6]:
```
clf.fit(X, y)
```

[6]:
```
SVC(kernel='linear')
```

Define a function `plot_svc_decision_function` that will plot **SVM** decision boundaries.

[7]:
```
def plot_svc_decision_function(clf, ax=None):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    x = np.linspace(plt.xlim()[0], plt.xlim()[1], 30)
    y = np.linspace(plt.ylim()[0], plt.ylim()[1], 30)
    Y, X = np.meshgrid(y, x)
    P = np.zeros_like(X)
    for i, xi in enumerate(x):
        for j, yj in enumerate(y):
            P[i, j] = clf.decision_function([[xi, yj]])
    # plot the margins
    ax.contour(X, Y, P, colors='k', levels=[-1, 0, 1], alpha=0.5,␣
 ↪linestyles=['--', '-', '--'])
```

[8]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='spring')
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1], s=200,␣
 ↪facecolors='none');
plt.grid()
```



5 | K-Means for Clustering

Student's name	.....	.....	.....
	.....	.....	.....
	.....	.....	.....
Score /20	.....	.....	.....

Detailed Credits

Anticipation *(4 points)*	.....	.....	.....
Management *(2 points)*	.....	.....	.....
Testing *(7 points)*	.....	.....	.....
Data Logging *(3 points)*	.....	.....	.....
Interpretation *(4 points)*	.....	.....	.....

The notebook is available at https://github.com/a-mhamdi/cosnip/ → Python → ml → clu-k-means.ipynb

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Let's begin by loading the *Mall_Customers* datasets.

[2]:
```
df = pd.read_csv('./datasets/Mall_Customers.csv')
```

[3]:
```
df.head()
```

[3]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Renaming some columns is very handy for further data manipulation.

[4]:
```
df.rename(columns={'Annual Income (k$)': 'Income', 'Spending Score (1-100)':␣
↪'Spending Score'}, inplace=True)
```

[5]:
```
df.head()
```

[5]:

	CustomerID	Genre	Age	Income	Spending Score
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

`df.describe()` allows to get useful insights from data.

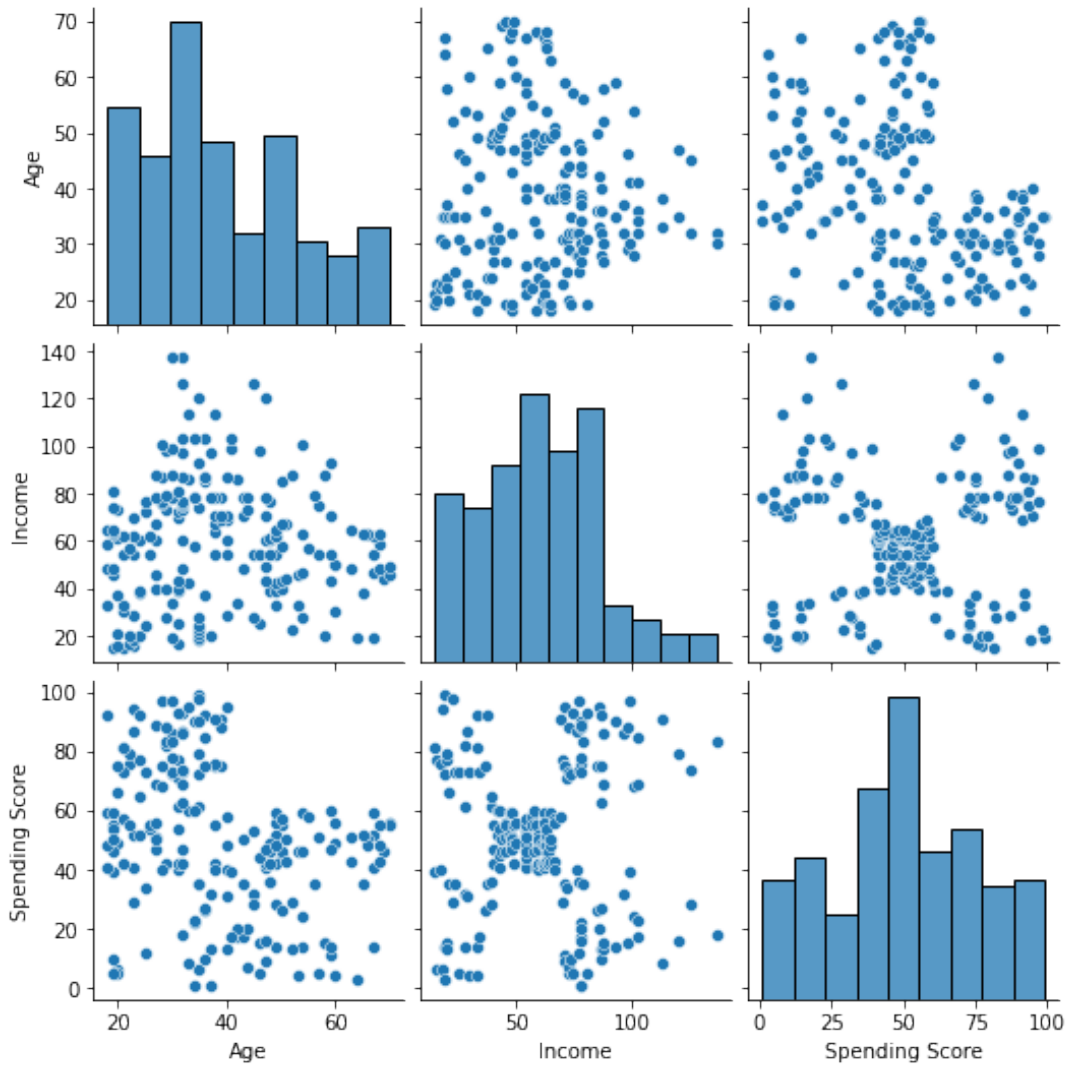
[6]:
```
df.describe()
```

[6]:

	CustomerID	Age	Income	Spending Score
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

Check the correlation between *Age*, *Income* and *Spending Score*.

[7]:
```
sns.pairplot(df[['Age','Income', 'Spending Score']])
```



[8]:
```
from sklearn import cluster
```

We will perform **K-Means** Clustering with 5 clusters using only 2 Variables.

[9]:
```
clu_k = cluster.KMeans(n_clusters=5 ,init="k-means++")
```

[10]:
```
clu_k = clu_k.fit(df[['Spending Score','Income']])
```

Coordinates of the centers.

[11]:
```
clu_k.cluster_centers_
```

[11]:
```
array([[49.51851852, 55.2962963 ],
       [82.12820513, 86.53846154],
       [17.11428571, 88.2       ],
       [79.36363636, 25.72727273],
       [20.91304348, 26.30434783]])
```

[12]:
```
df['Clusters'] = clu_k.labels_
```

[13]:
```
df.head()
```

[13]:

	CustomerID	Genre	Age	Income	Spending Score	Clusters
0	1	Male	19	15	39	4
1	2	Male	21	15	81	3
2	3	Female	20	16	6	4
3	4	Female	23	16	77	3
4	5	Female	31	17	40	4

[14]:
```
df['Clusters'].value_counts()
```

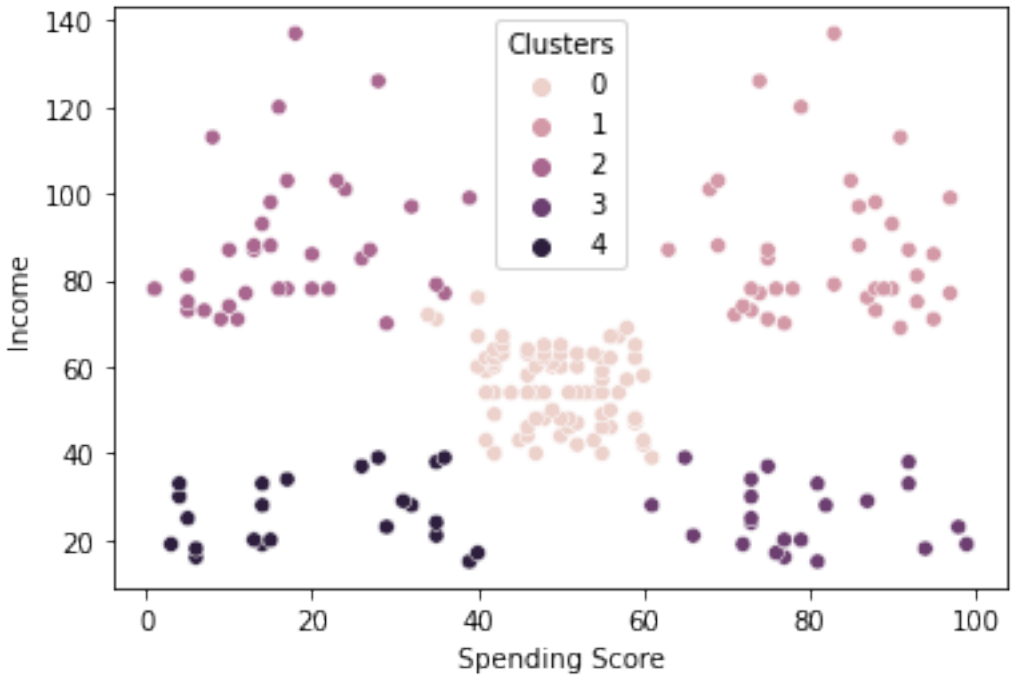
[14]:
```
0    81
1    39
2    35
4    23
3    22
Name: Clusters, dtype: int64
```

Let's save the new data frame to a new file.

[15]:
```
df.to_csv('./datasets/Mall_Clusters.csv', index = False)
```

Plot the 5 clusters on a chart.

[16]:
```
sns.scatterplot(x="Spending Score", y="Income", hue='Clusters',  data=df)
```



6 Binary Classifier using ANN

Student's name	....	....	....
Score /20	.....	.....	.....

Detailed Credits

Anticipation *(4 points)*	.....	.....	.....
Management *(2 points)*	.....	.....	.....
Testing *(7 points)*	.....	.....	.....
Data Logging *(3 points)*	.....	.....	.....
Interpretation *(4 points)*	.....	.....	.....

The notebook is available at https://github.com/a-mhamdi/cosnip/ → Python → ml → clf-ann.ipynb

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Import `sklearn`.

[2]:
```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

Import `keras`.

[3]:
```
from keras.models import Sequential
from keras.layers import Dense
```

```
Using TensorFlow backend.
```

Load the data using `pandas`.

[4]:
```
df = pd.read_csv('./datasets/Churn_Modelling.csv')
```

[5]:
```
df.head(3)
```

[5]:
```
   RowNumber  CustomerId   Surname  CreditScore Geography  Gender  Age  \
0          1    15634602  Hargrave          619    France  Female   42
1          2    15647311      Hill          608     Spain  Female   41
2          3    15619304      Onio          502    France  Female   42

   Tenure    Balance  NumOfProducts  HasCrCard  IsActiveMember  \
0       2       0.00              1          1               1
1       1   83807.86              1          0               1
2       8  159660.80              3          1               0

   EstimatedSalary  Exited
0        101348.88       1
1        112542.58       0
2        113931.57       1
```

[6]:
```
X = df.iloc[:, 3:13].values
y = df.iloc[:, 13].values

label_encoder_X_country = LabelEncoder()
label_encoder_X_gender = LabelEncoder()

X[:, 1] = label_encoder_X_country.fit_transform(X[:, 1])
X[:, 2] = label_encoder_X_gender.fit_transform(X[:, 2])

one_hot_encoder = ColumnTransformer([("Geography", OneHotEncoder(), [1])],␣
 ↪remainder = 'passthrough')

X = one_hot_encoder.fit_transform(X)
X = np.array(X, dtype=float)
X = X[:, 1:]
```

Scale the features.

[7]:
```
sc = StandardScaler()
X = sc.fit_transform(X)
```

Split the datasets into training & testing sets.

[8]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=0)
```

Define the artificial neural network architecture.

[9]:
```
clf_ann = Sequential()
```

Input layer & first hidden layer

[10]:
```
num_features = X_train.shape[1]
clf_ann.add(Dense(6, input_shape = (num_features, ), activation = 'relu'))
```

Second hidden layer

[11]:
```
clf_ann.add(Dense(6, activation = 'relu'))
```

Output layer

[12]:
```
num_classes = 1
clf_ann.add(Dense(num_classes, activation = 'sigmoid'))
```

[13]:
```
clf_ann.compile('Adam', loss = 'binary_crossentropy', metrics=['accuracy'])
```

An overall description of the neural network architecture.

[14]:
```
clf_ann.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 6)                 72
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 42
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 7
=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
_________________________________________________________________
```

Fit the classifier.

[15]:
```
clf_ann.fit(x=X_train, y=y_train, batch_size=200, epochs=20, verbose=1)
```

```
Epoch 1/20
8000/8000 [==============================] - 0s 21us/step - loss: 0.6352 -
accuracy: 0.6734
Epoch 2/20
8000/8000 [==============================] - 0s 7us/step - loss: 0.5697 -
accuracy: 0.7575
Epoch 3/20
8000/8000 [==============================] - 0s 7us/step - loss: 0.5312 -
accuracy: 0.7881
Epoch 4/20
8000/8000 [==============================] - 0s 6us/step - loss: 0.5058 -
accuracy: 0.7961
Epoch 5/20
8000/8000 [==============================] - 0s 8us/step - loss: 0.4867 -
accuracy: 0.8001
Epoch 6/20
8000/8000 [==============================] - 0s 7us/step - loss: 0.4722 -
accuracy: 0.8020
Epoch 7/20
8000/8000 [==============================] - 0s 6us/step - loss: 0.4607 -
accuracy: 0.8037
Epoch 8/20
8000/8000 [==============================] - 0s 8us/step - loss: 0.4520 -
```

```
accuracy: 0.8065
Epoch 9/20
8000/8000 [==============================] - 0s 9us/step - loss: 0.4454 -
accuracy: 0.8100
Epoch 10/20
8000/8000 [==============================] - 0s 9us/step - loss: 0.4405 -
accuracy: 0.8135
Epoch 11/20
8000/8000 [==============================] - 0s 7us/step - loss: 0.4366 -
accuracy: 0.8149
Epoch 12/20
8000/8000 [==============================] - 0s 7us/step - loss: 0.4336 -
accuracy: 0.8160
Epoch 13/20
8000/8000 [==============================] - 0s 7us/step - loss: 0.4310 -
accuracy: 0.8173
Epoch 14/20
8000/8000 [==============================] - 0s 5us/step - loss: 0.4287 -
accuracy: 0.8171
Epoch 15/20
8000/8000 [==============================] - 0s 5us/step - loss: 0.4270 -
accuracy: 0.8174
Epoch 16/20
8000/8000 [==============================] - 0s 5us/step - loss: 0.4255 -
accuracy: 0.8179
Epoch 17/20
8000/8000 [==============================] - 0s 5us/step - loss: 0.4240 -
accuracy: 0.8199
Epoch 18/20
8000/8000 [==============================] - 0s 5us/step - loss: 0.4229 -
accuracy: 0.8199
Epoch 19/20
8000/8000 [==============================] - 0s 5us/step - loss: 0.4215 -
accuracy: 0.8205
Epoch 20/20
8000/8000 [==============================] - 0s 6us/step - loss: 0.4202 -
accuracy: 0.8219
```

[15]:
```
<keras.callbacks.callbacks.History at 0x7f46f71cefa0>
```

Evaluate the model.

[16]:
```
scores = clf_ann.evaluate(x=X_test, y=y_test, batch_size=100, verbose=1)
```

```
2000/2000 [==============================] - 0s 10us/step
```

Let's predict an output.

[17]:
```
y_pred = clf_ann.predict(X_test)
y_pred = (y_pred > 0.5)
```

Define the confusion matrix.

[18]:
```
cm = confusion_matrix(y_test, y_pred)
tp, fp, fn, tn = cm.ravel()
```

[19]:
```
print('Accuracy is about {}%.' .format(100*(tp+tn)/sum((sum(cm)))))
```

```
Accuracy is about 83.2%.
```

[20]:
```
print('\
The loss value is: {}.\n\n\
The accuracy percentage is: {}%. '.format(scores[0], 100*scores[1]))
```

```
The loss value is: 0.4155806913971901.

The accuracy percentage is: 83.20000171661377%.
```

7 | Handwritten Digit Recognition

Student's name	.....	.....	.....
	.....	.....	.....
	.....	.....	.....
Score /20	.....	.....	.....

Detailed Credits

Anticipation *(4 points)*	.....	.....	.....
Management *(2 points)*	.....	.....	.....
Testing *(7 points)*	.....	.....	.....
Data Logging *(3 points)*	.....	.....	.....
Interpretation *(4 points)*	.....	.....	.....

The notebook is available at https://github.com/a-mhamdi/cosnip/ → Python → ml → mnist-cnn.ipynb

Let's begin with importing all the required librairies.

[1]:
```
import os

import numpy as np
from matplotlib import pyplot as plt

import tensorflow as tf

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.utils import np_utils
from keras.models import model_from_json

from PIL import Image
```

```
Using TensorFlow backend.
```

[2]:
```
import keras.backend.tensorflow_backend as tfback
from keras import backend as K
```

Check the current installed version of `tensorflow` and `keras`.

[3]:
```
print("tf.__version__ is", tf.__version__)
print("tf.keras.__version__ is:", tf.keras.__version__)
```

```
tf.__version__ is 2.2.0
tf.keras.__version__ is: 2.3.0-tf
```

The following method allows to get a list of available **GPU** devices, formatted as strings.

[4]:
```
def _get_available_gpus():
    #global _LOCAL_DEVICES
    if tfback._LOCAL_DEVICES is None:
        devices = tf.config.list_logical_devices()
        tfback._LOCAL_DEVICES = [x.name for x in devices]
    return [x for x in tfback._LOCAL_DEVICES if 'device:gpu' in x.lower()]
```

[5]:
```
tfback._get_available_gpus = _get_available_gpus
```

[6]:
```
# K.image_data_format() == 'channels_first'
# K.set_image_dim_ordering('tf')
K.set_image_data_format('channels_last') # tf: TensorFlow, th: Theano
```

Fix random seed for reproducibility.

[7]:
```
np.random.seed(0)
```

Load and normalize the data.

[8]:
```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

[9]:
```
'''
img_idx = np.random.randint(0, high=X_test.shape[0])
plt.imshow(X_train[img_idx, :, :], cmap=plt.cm.gray_r,␣
 ↪interpolation="nearest")
plt.show()
print("The output is {}.".format(y_train[img_idx]))
'''
```

[9]:
```
'\nimg_idx = np.random.randint(0,
high=X_test.shape[0])\nplt.imshow(X_train[img_idx, :, :], cmap=plt.cm.gray_r,
interpolation="nearest")\nplt.show()\nprint("The output is
{}.".format(y_train[img_idx]))\n'
```

[10]:
```
num_samples_train = np.random.randint(0, high=X_train.shape[0], size=20000)
X_train = X_train[num_samples_train, :, :]
y_train =y_train[num_samples_train]
```

[11]:
```
num_samples_test = np.random.randint(0, high=X_test.shape[0], size=4000)
X_test = X_test[num_samples_test, :, :]
y_test =y_test[num_samples_test]
```

Reshape the inputs.

[12]:
```
X_train = X_train.reshape(X_train.shape[0], 28, 28, 1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')
```

Normalize the inputs from 0 ⟶ 255 to 0 ⟶ 1.

[13]:
```
X_train = X_train/255
X_test = X_test/255
```

One hot encode the outputs.

[14]:
```
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

Number of classes is 10.

[15]:
```
num_classes = y_test.shape[1]
```

It is time now to define and build the model.

[16]:
```
my_model = Sequential()
my_model.add(Conv2D(16, (5,5), input_shape=(28,28,1), activation='relu'))
my_model.add(MaxPooling2D(pool_size=(2,2)))
my_model.add(Conv2D(32, (3,3), activation='relu'))
my_model.add(MaxPooling2D(pool_size=(2,2)))
my_model.add(Dropout(0.2))
my_model.add(Flatten())
# Fully Connected NN
my_model.add(Dense(128, activation='relu'))
my_model.add(Dense(50, activation='relu'))
my_model.add(Dense(num_classes, activation='softmax'))
```

[17]:
```
my_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 16)        416
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 32)        4640
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 32)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 5, 5, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 800)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               102528
_________________________________________________________________
dense_2 (Dense)              (None, 50)                6450
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
```

```
=================================================================
Total params: 114,544
Trainable params: 114,544
Non-trainable params: 0
_________________________________________________________________
```

- List of losses
- List of optimizers
- List of metrics

[18]:
```
my_model.compile(loss='categorical_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])
```

Fit the model.

[19]:
```
r = my_model.fit(x=X_train, y=y_train, validation_data=(X_test, y_test),␣
↪epochs=5, batch_size=100)

print("Returned:", r)
```

```
Train on 20000 samples, validate on 4000 samples
Epoch 1/5
20000/20000 [==============================] - 4s 222us/step - loss: 0.5177 -
accuracy: 0.8479 - val_loss: 0.1281 - val_accuracy: 0.9657
Epoch 2/5
20000/20000 [==============================] - 4s 186us/step - loss: 0.1278 -
accuracy: 0.9622 - val_loss: 0.0762 - val_accuracy: 0.9753
Epoch 3/5
20000/20000 [==============================] - 4s 195us/step - loss: 0.0855 -
accuracy: 0.9735 - val_loss: 0.0746 - val_accuracy: 0.9735
Epoch 4/5
20000/20000 [==============================] - 4s 193us/step - loss: 0.0656 -
accuracy: 0.9794 - val_loss: 0.0619 - val_accuracy: 0.9812
Epoch 5/5
20000/20000 [==============================] - 4s 195us/step - loss: 0.0528 -
accuracy: 0.9837 - val_loss: 0.0526 - val_accuracy: 0.9810
Returned: <keras.callbacks.callbacks.History object at 0x7f89f0259d30>
```

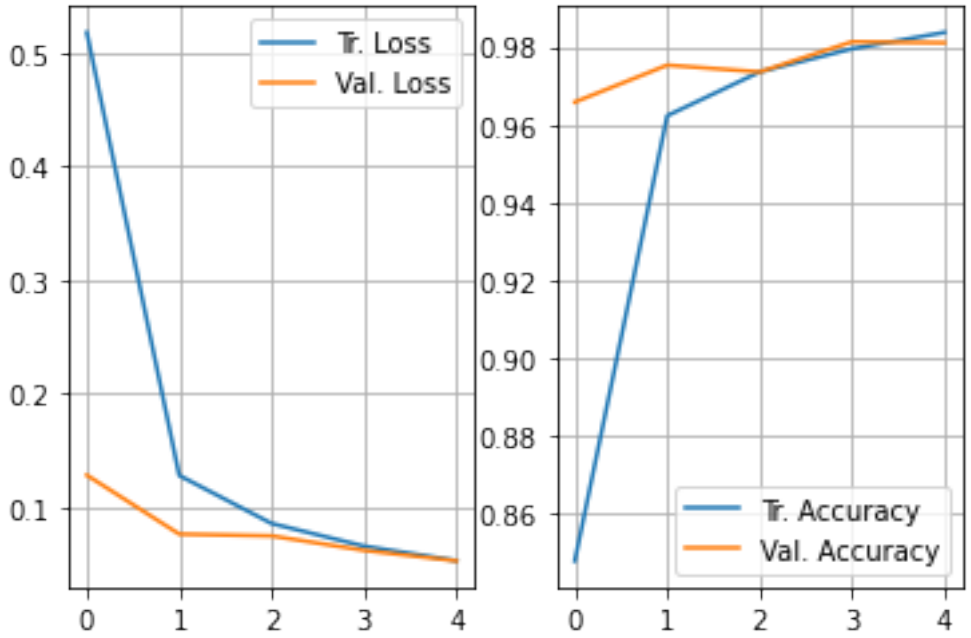
By printing the available keys, we should see: `dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])`.

[20]:
```
print(r.history.keys())
```

```
dict_keys(['val_loss', 'val_accuracy', 'loss', 'accuracy'])
```

[21]:
```
# Losses
plt.subplot(1, 2, 1)
plt.plot(r.history['loss'], label='Tr. Loss')
plt.plot(r.history['val_loss'], label='Val. Loss')
plt.grid()
plt.legend()

# Accuracies
plt.subplot(1, 2, 2)
```

```
plt.plot(r.history['accuracy'], label='Tr. Accuracy')
plt.plot(r.history['val_accuracy'], label='Val. Accuracy')
plt.grid()
plt.legend()
```



Evaluate the model.

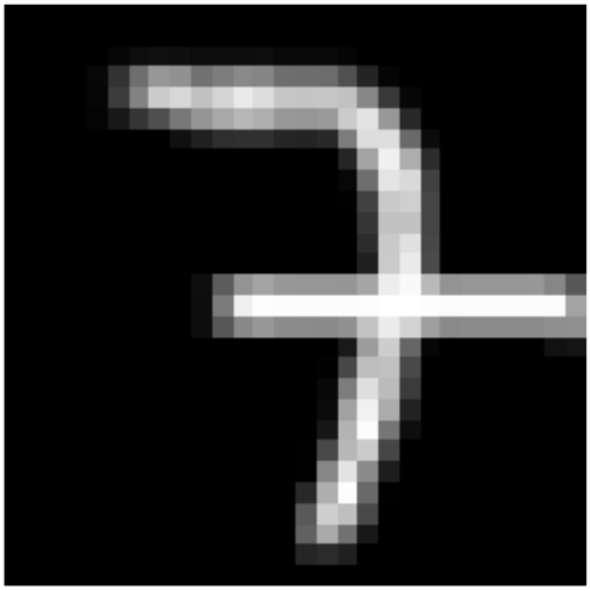
[22]:
```
scores = my_model.evaluate(X_test, y_test, verbose=0)
print("CNN error: {}%".format(100*(1-scores[1])))
```

```
CNN error: 1.8999993801116943%
```

Test the model.

[23]:
```
img = Image.open('to-test/7.png');
img = img.convert('L')
img = img.resize((28, 28))

array_img = (np.array(img))/255
in_data = array_img.reshape((1, 28, 28, 1)).astype('float32')

plt.imshow(array_img*255, cmap=plt.cm.gray_r, interpolation="nearest")
plt.axis('Off')
```

[23]:
```
(-0.5, 27.5, 27.5, -0.5)
```



[24]:
```
y_pred = my_model.predict(in_data)
_, idx = np.where(y_pred == np.max(y_pred))
print("Result is {}. Probability is {}%.".format(int(idx), 100*y_pred[0,␣
 ↪int(idx)]))
```

```
Result is 7. Probability is 59.57772135734558%.
```

The overall scope of this manual is to introduce **Machine Learning**, through some numeric simulations, to the students at the department of **Electrical Engineering**.

The topics discussed in this manuscript are as follow:

① Getting started with `Python`

② Linear Regression

③ Classification

④ Clustering

⑤ CNN

`Python`; `Jupyter`; `NumPy`; `Matplotlib`; `scikit-learn`; machine learning; linear regression; classification; clustering; deep learning.