Workshop_IALab_1-Introduction_to_python

November 10, 2019

1 Python cheat sheet

1.0.1 IALab - GarageISEP

Author: Lucas Mercier

- Python basics
  - Operators
  - Le typage dynamique
  - Les intéractions utilisateur
  - Les conditions
  - Les boucles
  - Les fonctions
- Data manipulation with Numpy and Pandas
  - Numpy arrays
  - Pandas DataFrame
  - Manipulate and extract data
  - Load data from files
- Visualize data with Matplotlib and Seaborn

Formating values

```
In [ ]: import numpy as np # change
```

Formating data

```
In [9]: pi=3.1415
        print("I am a rounded value {0:0.2f}.".format(pi))
        print("You can print multiple values like this {0:0.2f} and {1:0.4f} .".format(pi,pi))
```

```
I am a rounded value 3.14.
You can print multiple values like this 3.14 and 3.1415 .
```

Loops

```
In [50]: # one line loop vs 'classic' loop
         def one_line_square(n):
             return [i**2 for i in range(n)]

         def square(n):
             res=[]
             for i in range(n):
                 res.append(i**2)
             return res

         print(one_line_square(3))
         print(square(3))
```

```
[0, 1, 4]
[0, 1, 4]
```

Data manipulation with Numpy and Pandas

1.0.2 Numpy arrays

```
In [8]: import numpy as np

        vector1=np.array([5,8,9,10,11,12,13,14,16])
        vector2=np.array([10,12,48,23,24,48,9,7,13])
        matrix=np.array([vector1,vector2])
        print("A vector:")
        print(vector1)
        print("")
        print("A matrix")
        print(matrix)

        zeros_vector = np.zeros([10])
        zeros_matrix = np.zeros([10,10])
```

```
A vector:
[ 5  8  9 10 11 12 13 14 16]

A matrix
[[ 5  8  9 10 11 12 13 14 16]
 [10 12 48 23 24 48  9  7 13]]
```

1.0.3 Pandas DataFrame

- Pandas is data manipulation library based on numpy
- Pandas DataFrames are a way to present data and add labels to columns and indexes

```
In [21]: import pandas as pd

         human = np.array([["M",190,90],["M",170,65],["F",165,55]])
         df = pd.DataFrame(human,columns=["Sex","Size","Weigth"])
         print(human)
         df
```

```
[['M' '190' '90']
 ['M' '170' '65']
 ['F' '165' '55']]
```

```
Out[21]:   Sex Size Weigth
         0   M  190     90
         1   M  170     65
         2   F  165     55
```

You can easily access to a dataframe's value with: df.values

```
In [22]: df.values
```

```
Out[22]: array([['M', '190', '90'],
                ['M', '170', '65'],
                ['F', '165', '55']], dtype=object)
```

1.0.4 Load data from files

- "Comma-separated values" or .CSV is the most common file type for data analysis. It can be easily read thanks to pandas.
- When loading data from files, we have to know if the file contains a header or not and are they separated by ";" or ","

```
In [34]: iris = pd.read_csv("iris.csv",sep=";")
         iris.head(3)
```

```
Out[34]:    SepalLength  SepalWidth  PetalLength  PetalWidth   Class
         0          5.1         3.5          1.4         0.2  setosa
         1          4.9         3.0          1.4         0.2  setosa
         2          4.7         3.2          1.3         0.2  setosa
```

1.0.5 Manipulate and extract data

```
In [35]: iris.describe()
```

```
Out[35]:        SepalLength  SepalWidth  PetalLength  PetalWidth
         count   150.000000  150.000000   150.000000  150.000000
         mean      5.843333    3.054000     3.758667    1.198667
         std       0.828066    0.433594     1.764420    0.763161
         min       4.300000    2.000000     1.000000    0.100000
         25%       5.100000    2.800000     1.600000    0.300000
```

```
         50%       5.800000    3.000000     4.350000    1.300000
         75%       6.400000    3.300000     5.100000    1.800000
         max       7.900000    4.400000     6.900000    2.500000
```

```
In [62]: iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
SepalLength    150 non-null float64
SepalWidth     150 non-null float64
PetalLength    150 non-null float64
PetalWidth     150 non-null float64
Class          150 non-null object
dtypes: float64(4), object(1)
memory usage: 5.9+ KB
```

```
In [64]: iris.columns
```

```
Out[64]: Index(['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth', 'Class'], dtype='obj
```

```
In [66]: iris.sort_values(by='SepalLength').head()
```

```
Out[66]:     SepalLength  SepalWidth  PetalLength  PetalWidth   Class
         13          4.3         3.0          1.1         0.1  setosa
         42          4.4         3.2          1.3         0.2  setosa
         38          4.4         3.0          1.3         0.2  setosa
         8           4.4         2.9          1.4         0.2  setosa
         41          4.5         2.3          1.3         0.3  setosa
```

```
In [70]: iris.loc[iris['Class']=="setosa"].head()
```

```
Out[70]:    SepalLength  SepalWidth  PetalLength  PetalWidth   Class
         0          5.1         3.5          1.4         0.2  setosa
         1          4.9         3.0          1.4         0.2  setosa
         2          4.7         3.2          1.3         0.2  setosa
         3          4.6         3.1          1.5         0.2  setosa
         4          5.0         3.6          1.4         0.2  setosa
```

```
In [80]: (iris['Class']=="setosa").value_counts()
         ((iris['SepalLength']>5.0) & (iris['Class']=="setosa")).value_counts()
```

```
Out[80]: False    128
         True      22
         dtype: int64
```

When training machine learning models, we often need to separate values and labels. A simple way to achieve it is by creating two new dataframes.

```
In [123]: X = iris[['SepalLength','SepalWidth','PetalLength','PetalWidth']].copy()
          Y = iris[['Class']].copy()
          Y.head(5)

Out[123]:     Class
          0  setosa
          1  setosa
          2  setosa
          3  setosa
          4  setosa
```

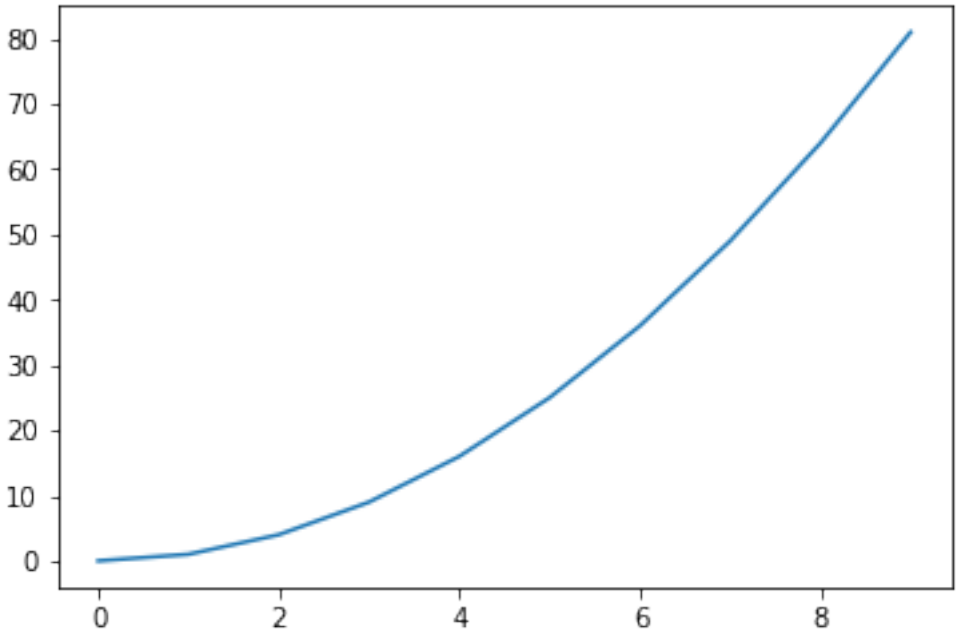
For classification problems you will often face a common issue: labels. For a lot of algorithm, you can't deal with string values, thus you will have to convert each label in an integer. For example for iris dataset: - Setosa => 0 - Virginica => 1 - Versicolor => 2

```
In [125]: Y = iris[['Class']].copy()
          map_dict = {"setosa":0,'versicolor':1,'virginica':2}
          Y["Class"] = Y["Class"].map(map_dict)
          Y.head()

Out[125]:    Class
          0      0
          1      0
          2      0
          3      0
          4      0
```

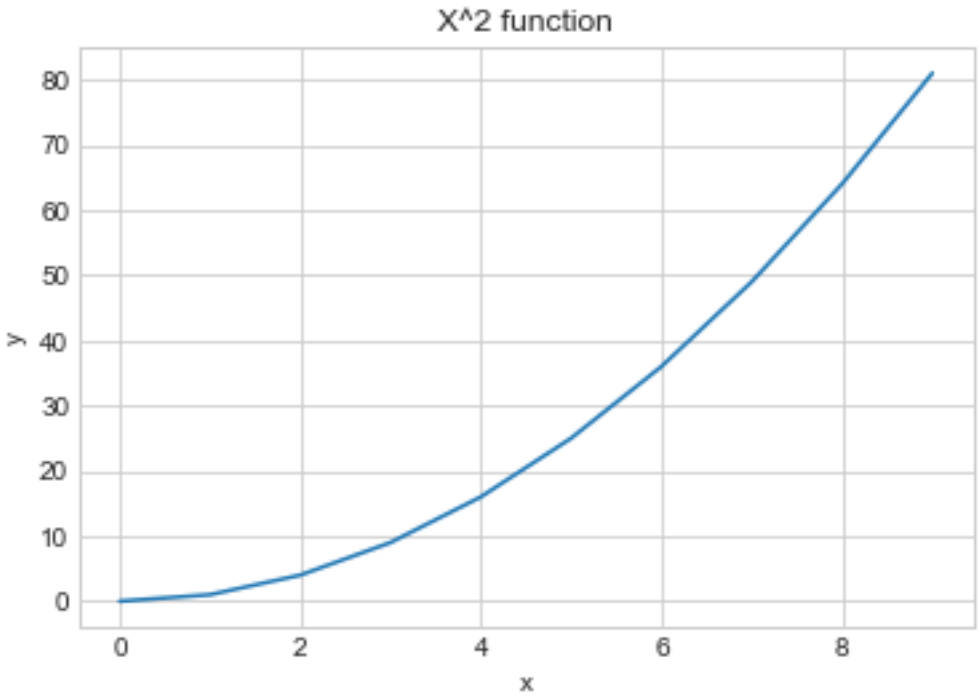
Visualize data with Matplotlib and Seaborn

1.0.6 Simple plot

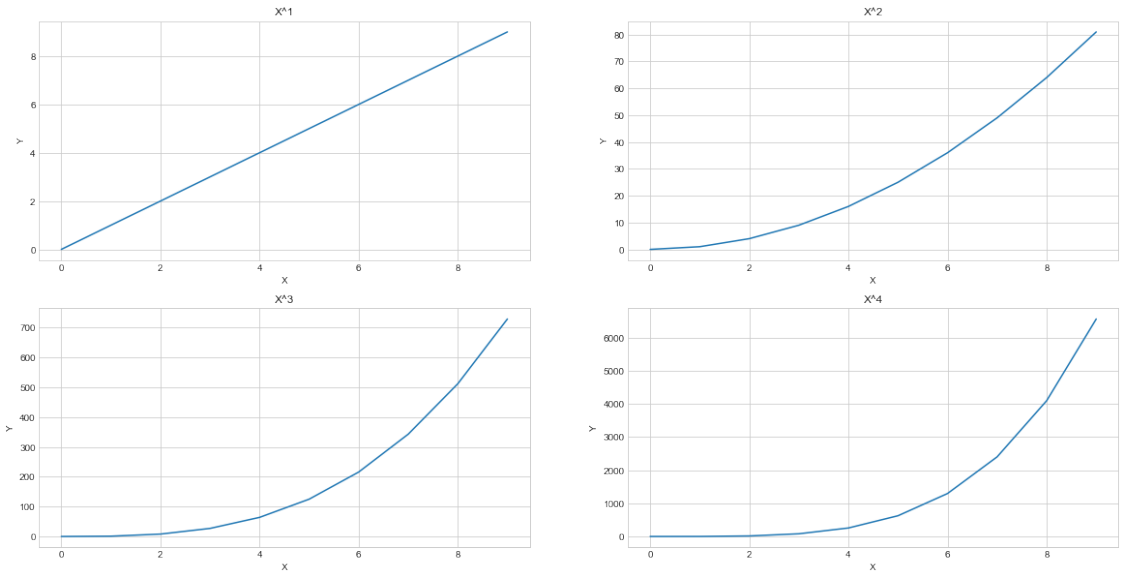
```
In [107]: import matplotlib.pyplot as plt
          import seaborn as sns

          x = np.arange(10)
          plt.plot(x,x**2)
          plt.show()
```



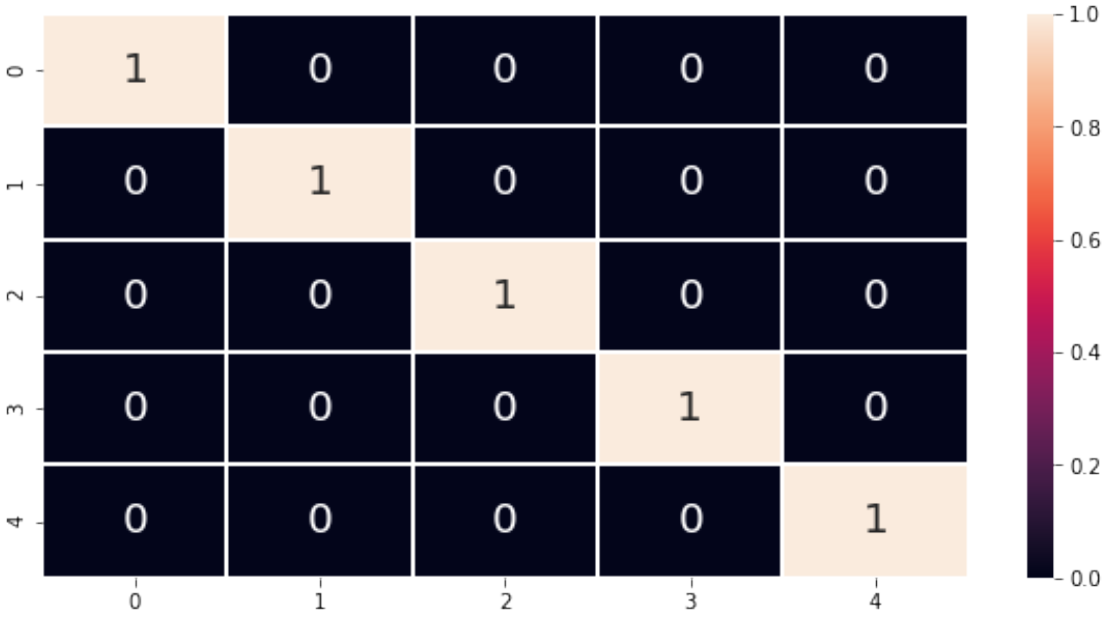
In [111]:
```
plt.style.use('seaborn-whitegrid')

x = np.arange(10)
plt.plot(x,x**2)
plt.title("X^2 function")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



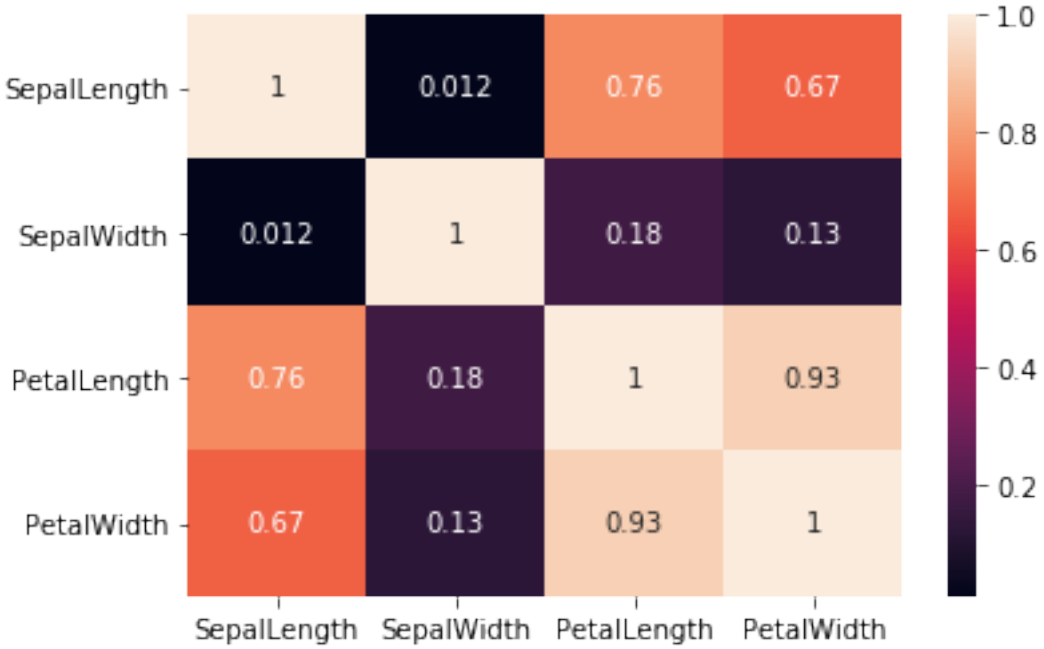
In [122]:
```
%matplotlib inline

ax=plt.figure(figsize=(20,10))

x = np.arange(10)

for i in range(0,4):
    plt.subplot(2,2,(i+1))
    plt.title("X^{}".format(i+1))
    plt.xlabel("X")
    plt.ylabel("Y")
    plt.plot(x,x**(i+1))
plt.show()
```



In [94]:
```
ax=plt.figure(figsize=(10,5))
sns.heatmap(np.identity(5),annot=True,linewidth=1,annot_kws={"fontsize":20})
plt.show()
```



In [95]:
```
sns.heatmap(iris.corr()**2,annot=True)
plt.show()
```


SepalLength
SepalWidth
PetalLength
PetalWidth
1
0.012
0.76
0.67
0.012
1
0.18
0.13
0.76
0.18
1
0.93
0.67
0.13
0.93
1
SepalLength
SepalWidth
PetalLength
PetalWidth
1.0
0.8
0.6
0.4
0.2

In [ ]: