TP1 : configurer son environnement

1- QU'EST-CE QUE ANACONDA?

Anaconda est une distribution open source Python pour l'analyse de données à grande échelle (fournie par Continuum Analytics, Inc.). Il vous fournit de nombreux outils nécessaires à l'analyse de grands ensembles de données. Une fois installé, Anaconda comprend :

- Le langage Python de base (vous pouvez utiliser quelle version)
- Plus de 1000 packages de science des données
- Gestion de colis avec conda
- IPython
- Beaucoup plus

- Telecharger et installer anaconda
  - Telecherger anaconda : https://www.anaconda.com/distribution/
  - Suivre les instructions

2- QU'EST-CE QUE MINICONDA?

Miniconda est une version allégée d'Anaconda. Le téléchargement d'Anaconda est volumineux (quelques giga-octets) et peut prendre un certain temps à télécharger et à installer. Miniconda, en revanche, est une alternative plus petite. Il ne comprend que les exigences de base et vous permet d'installer des packages de données scientifiques à la demande, réduisant ainsi la taille et la durée du téléchargement.

- Telecharger miniconda
  - Telecherger miniconda : https://docs.conda.io/projects/conda/en/latest/user-guide/install/windows.html
  - Conda pour data scientist: https://kaust-vislab.github.io/introduction-to-conda-for-data-scientists/

3- Installation de l'environnement de travail sous Windows

Créez un conda environment en Python 3.6. (pour plus d'information à se sujet consultez la documentation concernant la gestion d'environnements)

https://conda.io/docs/user-guide/tasks/manage-environments.html

Un conda environment est un répertoire contenant une collection spécifique de packages que vous aurez décidé d'installer. Certains packages ne supportant pas la dernière version de Python, il peut être pertinent de maintenir un environnement pour une version spécifique de Python.

E1 : créer son conda environnement

```
conda create --name py36 python=3.6
```

E2 : lister les environnements

```
conda info --envs
```

E3 : activer votre environnement au sein de la console

```
Conda activate ML
```

E4 : désactiver votre environnement au sein de la console

```
Conda deactivate
```

E5 : afficher la version de python installer

```
Python --version
```

Pour les besoins du Machine Learning, on a généralement besoin des librairies suivantes : NumPy, SciPy, Matplotlib, Pandas, Scikit-learn, Statsmodels, seaborn, bokeh, …

E5 : installer spyder

```
conda install spyder
```

E5 : lister les package de son environnement

```
conda list
```

4- Prise en main de python

https://github.com/justmarkham/python-reference

5- Numpy, Pandas et Matplotlib

- **Numpy: tableaux et matrices**

NumPy est une extension du langage de programmation Python, permettant la prise en charge de grands formats multidimensionnels (numériques), des tableaux et des matrices, ainsi qu'une grande bibliothèque de fonctions mathématiques de haut niveau pour opérer sur ces derniers.

https://github.com/Kyubyong/numpy_exercises

https://github.com/rougier/numpy-100/blob/master/100_Numpy_exercises.md

https://github.com/Kyubyong/numpy_exercises/blob/master/9_Mathematical_functions_solutions.ipynb

 # créer des arrays a partir de lists

```
from __future__ import print_function
import numpy as np
data1 = [1, 2, 3, 4, 5]
arr1 = np.array(data1)
data2 = [range(1, 5), range(5, 9)]
arr2 = np.array(data2)
```

 # examiner les arrays

```
arr1.dtype
arr2.dtype
arr2.ndim
arr2.shape
arr2.size
len(arr2)
```

 # Utilisation de arange : arange est comme range, excepté qu'ilrenvoie un array et non une liste

```
int_array = np.arange(5)
float_array = int_array.astype(float)
```

 # Reshaping

```
matrix = np.arange(10, dtype=float).reshape((2, 5))
print(matrix.shape)
print(matrix.reshape(5, 2))
```

 # Selection

```
arr1[0]
arr2[0, 3]
arr2[0][3]
```

 # Utilisation des booleen

```
arr = np.arange(10)
arr[5:8] = 12
arr_view = arr[5:8]
arr_view[:] = 13
arr_copy = arr[5:8].copy()
arr[arr > 5]
arr[arr > 5] = 0
```

```
names = np.array(['Stan', 'Diarra', 'Medar', 'Pat'])
names[(names == 'Stan') | (names=='Pat')]
names[(names != 'Stan')]
names[names != 'Stan'] = 'Joe'
np.unique(names)
```

 # Distance euclidienne

```
vec1 = np.random.randn(10)
vec2 = np.random.randn(10)
dist = np.sqrt(np.sum((vec1 - vec2) ** 2))
```

 # Math et stat

```
rnd = np.random.randn(4, 2)
rnd.mean()
rnd.std()
rnd.sum()
rnd.sum(axis=0)
rnd.sum(axis=1)
(rnd > 0).sum()
(rnd > 0).any()
(rnd > 0).all()
nums = np.arange(32).reshape(8, 4)
nums.T
nums.flatten()
```

from __future__ import print_function

https://github.com/justmarkham

NumPy est une extension du langage de programmation Python, permettant la prise en charge de grands formats multidimensionnels (numériques), des tableaux et des matrices, ainsi qu'une grande bibliothèque de fonctions mathématiques de haut niveau pour opérer sur ces derniers.

- **Pandas: manipulation des données**

On dit souvent que 80% de l'analyse des données est consacrée au nettoyage et à la préparation des données. Pour résoudre le problème, ce chapitre se concentre sur un petit aspect mais important de la manipulation et du nettoyage des données avec les pandas.

https://github.com/justmarkham

https://pandas.pydata.org/pandas-docs/stable/user_guide/reshaping.html

```
from __future__ import print_function
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

 # Créer un DataFrame

```
columns = ['name', 'age', 'gender', 'job']
user1 = pd.DataFrame([['alice', 19, "F", "student"],
 ['john', 26, "M", "student"]],
columns=columns)

user2 = pd.DataFrame([['eric', 22, "M", "student"],
['paul', 58, "F", "manager"]],
columns=columns)

user3 = pd.DataFrame(dict(name=['peter', 'julie'],
age=[33, 44], gender=['M', 'F'],
job=['engineer', 'scientist']))
```

 # Concatener des DataFrames

```
user1.append(user2)
users = pd.concat([user1, user2, user3])
```

 # Intersection et union de DataFrames

```
user4 = pd.DataFrame(dict(name=['alice', 'john', 'eric', 'julie'],
  height=[165, 180, 175, 171]))
merge_inter = pd.merge(users, user4, on="name")
users = pd.merge(users, user4, on="name", how='outer')
```

 # résumer un DataFrames

```
users
users.head()
users.tail()
type(users)
users.info()
users.index
users.shape
users.columns
users.values
users.describe()
users.describe(include='all')
users.describe(include=['object'])
users.describe(include=['int64','float64'])
```

 # Sélectionner des colonnes sur un DataFrame

```
users['gender']
users.gender
users[['age', 'gender']]
my_cols = ['age', 'gender']
users[my_cols]
```

 # Sélectionner des lignes sur un DataFrame

```
df = users.copy()
## iloc : acces via des index entiers
df.iloc[0]
df.iloc[0, 0]
## ix ou loc : acces mixte
df.ix[0]
```

```
df.ix[0, "age"]
df.loc[0]
df.loc[0, "age"]
```

 # Sélectionner/filtrer des lignes sur un DataFrame

```
users[users.age < 20]
young_bool = users.age < 20
users[young_bool]
users[users.age < 20].job
users[users.age < 20][['job']]
users[users.age < 20][['age', 'job']]
users[(users.age > 20) & (users.gender=='M')]
users[users.job.isin(['student', 'engineer'])]
```

 # Tri sur un DataFrame

```
df.age.sort_values()
df.height.sort_values()
df.sort_values(by='age')
df.sort_values(by='age', ascending=False)
df.sort_values(by=['job', 'age'])
```

 # Remodeler en pivotant

Découpe un DataFrame du format large et long (empilé)

```
staked = pd.melt(users, id_vars="name", var_name="variable",
value_name="value")
```

"Pivote" un DataFrame du format long (empilé) au format large

```
unstaked = staked.pivot(index='name', columns='variable', values='value')
```

 # Contrôle Qualité : Renommer des valeurs

```
df = users.copy()
df.columns = ['age', 'genre', 'travail', 'nom', 'taille']
df.travail = df.travail.map({ 'student':'etudiant', 'manager':'manager',
 'engineer':'ingenieur', 'scientist':'scientific'})
```

 # Contrôle Qualité : gestion des duplicats

```
df = users.append(df.iloc[0], ignore_index=True)
df.duplicated()
df[df.duplicated()]
df.age.duplicated()
df.duplicated(['age', 'gender']).sum()
df = df.drop_duplicates()
df.duplicated()
```

 # Contrôle Qualité : gestion des valeurs manquantes

```
df = users.copy()
df.describe(include='all')
df.isnull()
df.isnull().sum()
df.height.isnull()
df.height.isnull().sum()
df.height.notnull()
df[df.height.notnull()]
```

Stratégie 1 : suppression des valeurs manquantes

```
df.dropna()
df.dropna(how='all')
```

Stratégie 2 : remplacer les valeurs manquantes par la moyenne de la colonne

```
df = users.copy()
df.height.mean()
df.ix[df.height.isnull(), "height"] = df["height"].mean()
```

 # Contrôle Qualité : Faire face aux valeurs aberrantes

size = pd.Series(np.random.normal(loc=175, size=20, scale=10))

size[:3] += 500

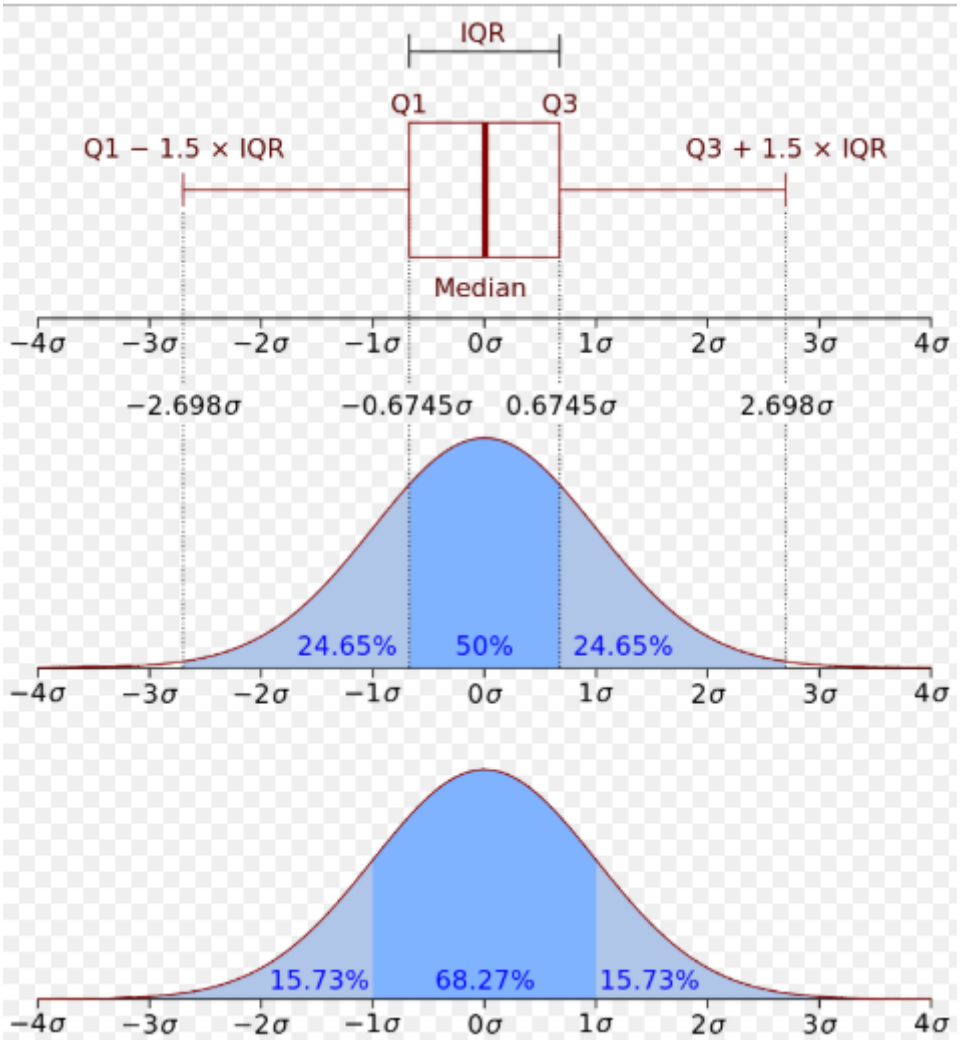
Methode 1 : Basé sur des statistiques paramétriques ; utilisez la moyenne

Supposer que la variable aléatoire suit la distribution normale. Exclure les données en dehors de 3 écarts-types :

- Probabilité qu'un échantillon se situe dans un délai de 1 sd: 68,27%

- Probabilité qu'un échantillon se situe dans un délai de 3 sd : 99,73% (68,27 + 2 * 15,73)

https://fr.wikipedia.org/wiki/Loi_normale#/media/File:Boxplot_vs_PDF.svg




```
3 * size.std()
```

```
(size - size.mean()).abs()
size_outlr_mean = size.copy()
size_outlr_mean[((size - size.mean()).abs() > 3 * size.std())] = size.mean()
```

Methode 2 : Basé sur des statistiques non paramétriques ; utiliser la médiane

L'écart absolu médian (MAD), basé sur la médiane, est une statistique non paramétrique robuste.

https://en.wikipedia.org/wiki/Median_absolute_deviation

```
mad = 1.4826 * np.median(np.abs(size - size.median()))
size_outlr_mad = size.copy()
print(size_outlr_mad.mean(), size_outlr_mad.median())
print(size_outlr_mad.mean(), size_outlr_mad.median())
```

 # Group by

```
for grp, data in users.groupby("job"):
print(grp, data)
```

 # Fichier I / O

read from csv

```
import tempfile, os.path
wdir=' C:\Users\M Stan\ML '
os.chdir(wdir)
csv_filename = os.path.join(wdir, "users.csv")
users.to_csv(csv_filename, index=False)
other = pd.read_csv(csv_filename)
```

read from url

```
url =
'https://raw.github.com/neurospin/pystatsml/master/data/salary_table.csv'
salary = pd.read_csv(url)
```

read from excel

(Voir importation des fichiers excels)

- **Matplotlib: Graphismes**

https://github.com/rougier/matplotlib-tutorial

https://www.edureka.co/blog/python-matplotlib-tutorial/

Matplotlib est probablement le package Python le plus utilisé pour les graphiques 2D. Il fournit à la fois un moyen très rapide de visualiser les données de Python et des chiffres de qualité publication dans de nombreux formats. Nous allons explorer matplotlib en mode interactif couvrant les cas les plus courants.

plot simple

EX 1

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 10, 50)
sinus = np.sin(x)
```

```
plt.plot(x, sinus)
plt.plot(x, sinus, "o")
plt.show()
```

EX 2

```
from matplotlib import pyplot as plt

x = [5,2,7]
y = [2,16,4]
plt.plot(x,y)
plt.title('Info')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.show()
```

EX 3 : Ajouter de la couleur et des style

```
from matplotlib import pyplot as plt
from matplotlib import style

style.use('ggplot')
x = [5,8,10]
y = [12,16,6]
x2 = [6,9,11]
y2 = [6,15,7]
plt.plot(x,y,'g',label='line one', linewidth=5)
plt.plot(x2,y2,'c',label='line two',linewidth=5)
plt.title('Epic Info')
plt.ylabel('Y axis')
plt.xlabel('X axis')
plt.legend()
plt.grid(True,color='k')
plt.show()
```

Graphique à barres

Un graphique à barres utilise des barres pour comparer les données entre différentes catégories. Il convient bien lorsque vous souhaitez mesurer les changements sur une période donnée.

EX : Comparaison entre la distance parcourue par deux voitures BMW et Audi sur une période de 5 jours.

```
from matplotlib import pyplot as plt

plt.bar([0.25,1.25,2.25,3.25,4.25],[50,40,70,80,20],
label="BMW",color='b',width=.5)
plt.bar([.75,1.75,2.75,3.75,4.75],[80,20,20,50,60],
label="Audi", color='r',width=.5)
plt.legend()
plt.xlabel('Days')
plt.ylabel('Distance (kms)')
plt.title('Information')
plt.show()
```

Histogramme

Les histogrammes sont utilisés pour montrer une distribution alors qu'un graphique à barres est utilisé pour comparer différentes entités. Les histogrammes sont utiles lorsque vous avez des tableaux ou une très longue liste.

EX : Tracer l'âge de la population par rapport à la plage de valeurs divisée en séries d'intervalles.

```
import matplotlib.pyplot as plt
population_age =
[22,55,62,45,21,22,34,42,42,4,2,102,95,85,55,110,120,70,65,55,111,115,80,75,
65,54,44,43,42,48]
bins = [0,10,20,30,40,50,60,70,80,90,100]
plt.hist(population_age, bins, histtype='bar', rwidth=0.9)
plt.xlabel(' groupes d’age ')
plt.ylabel(Nombre de personnes)
plt.title('Histogramme')
plt.show()
```

Nuage de points

Nous avons besoin de diagrammes de dispersion pour comparer des variables, par exemple, dans quelle mesure une variable est affectée par une autre variable pour créer une relation à partir de celle-ci. Les données sont affichées sous forme d'un ensemble de points, chacun ayant la valeur d'une variable qui détermine la position sur l'axe horizontal et la valeur d'une autre variable détermine la position sur l'axe vertical.

```
import matplotlib.pyplot as plt
x = [1,1.5,2,2.5,3,3.5,3.6]
y = [7.5,8,8.5,9,9.5,10,10.5]

x1=[8,8.5,9,9.5,10,10.5,11]
y1=[3,3.5,3.7,4,4.5,5,5.2]

plt.scatter(x,y, label='revenu eleve faible epargne',color='r')
plt.scatter(x1,y1,label='Faible revenu epargne eleve',color='b')
plt.xlabel(epargne*100')
plt.ylabel(revenu*1000')
plt.title('Scatter Plot')
plt.legend()
plt.show()
```

Parcelles empilées

Ces graphiques peuvent être utilisés pour suivre les changements dans le temps pour deux groupes liés ou plus qui composent une catégorie entière.

EX : regroupons le travail effectué au cours d’une journée en catégories, par exemple, dormir, manger, travailler et jouer.

```
import matplotlib.pyplot as plt
days = [1,2,3,4,5]

sleeping =[7,8,6,11,7]
eating = [2,3,4,3,2]
working =[7,8,7,2,2]
playing = [8,5,7,8,13]

plt.plot([],[],color='m', label='Sleeping', linewidth=5)
plt.plot([],[],color='c', label='Eating', linewidth=5)
plt.plot([],[],color='r', label='Working', linewidth=5)
```

```
plt.plot([],[],color='k', label='Playing', linewidth=5)

plt.stackplot(days, sleeping,eating,working,playing,
colors=['m','c','r','k'])

plt.xlabel('x')
plt.ylabel('y')
plt.title('Stack Plot')
plt.legend()
plt.show()
```

Comme nous pouvons le voir dans l'image ci-dessus, nous avons du temps passé en fonction des catégories. Par conséquent, le diagramme de surface ou le diagramme de pile est utilisé pour afficher les tendances au fil du temps, parmi différents attributs.

Camembert

Un graphique à secteurs se réfère à un graphique circulaire qui est divisé en segments, à savoir des tranches de camembert. Il est essentiellement utilisé pour montrer le pourcentage ou les données proportionnelles où chaque tranche de camembert représente une catégorie.

```
import matplotlib.pyplot as plt

days = [1,2,3,4,5]

sleeping =[7,8,6,11,7]
eating = [2,3,4,3,2]
working =[7,8,7,2,2]
playing = [8,5,7,8,13]
slices = [7,2,2,13]
activities = ['sleeping','eating','working','playing']
cols = ['c','m','r','b']

plt.pie(slices,
  labels=activities,
  colors=cols,
  startangle=90,
  shadow= True,
  explode=(0,0.1,0,0),
  autopct='%1.1f%%')

plt.title('Pie Plot')
plt.show()
```

Travailler avec plusieurs parcelles

```
import numpy as np
import matplotlib.pyplot as plt

def f(t):
    return np.exp(-t) * np.cos(2*np.pi*t)
t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)
plt.subplot(221)
plt.plot(t1, f(t1), 'bo', t2, f(t2))
plt.subplot(222)
plt.plot(t2, np.cos(2*np.pi*t2))
plt.show()
```

Le code est à peu près similaire aux exemples précédents que vous avez vus, mais il y a un nouveau concept ici, à savoir la sous-parcelle. La commande subplot () spécifie numrow, numcol, fignum, qui va de 1 à numrows * numcols. Les virgules dans cette commande sont facultatifs si numrows * numcols <10. La sous-parcelle (221) est donc identique à la sous-parcelle (2,2,1). Par conséquent, les sous-graphiques nous aident à tracer plusieurs graphiques dans lesquels vous pouvez le définir en alignant verticalement ou horizontalement.

sauvegarder un graphique

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 10, 50)
plt.plot(x, sinus)
plt.savefig("sinus.png")
plt.close()
```

 # autres exemples

Step by step multiplot

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 10, 50)
sinus = np.sin(x)
cosinus = np.cos(x)
plt.xlabel('this is x!')
plt.ylabel('this is y!')
plt.title('My First Plot')
plt.show()
```

 # Scatter (2D) plots

```
import pandas as pd
url = 'https://raw.github.com/duchesnay/pylearn-
doc/master/data/salary_table.csv'
salary = pd.read_csv(url)
```

 # Simple scatter plot simple avec couleur

```
colors = colors_edu = {'Bachelor':'r', 'Master':'g', 'Ph.D':'blue'}
df= salary
plt.scatter(df['experience'], df['salary'], c=df['education'].apply(lambda
x: colors[x]))
plt.show()
```

plt.figure(figsize=(6,5))

- **Seaborn**
  http://stanford.edu/~mwaskom/software/seaborn

Seaborn est une bibliothèque de visualisation de données Python basée sur matplotlib . Il fournit une interface de haut niveau pour dessiner des graphiques statistiques attractifs et informatifs.

 # Boxplot

Les boîtes à moustaches ne sont pas paramétriques : elles affichent une variation dans les échantillons d'une population statistique sans aucune hypothèse de la distribution statistique sous-jacente.

```
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
url = 'https://raw.github.com/duchesnay/pylearn-
doc/master/data/salary_table.csv'
salary = pd.read_csv(url)
sns.boxplot(x="education", y="salary", hue="management", data=salary,
palette="PRGn")
plt.show()
```

sns.boxplot(x="management", y="salary", hue="education", data=salary, palette="PRGn")

plt.show()

https://www.dataschool.io/learn/