TP1 : configurer son environnement – Prise en main, statistique descriptive et visualisation

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'il renvoie 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

```
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'])
```

Filter un data frame

 # 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

 # 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

 # 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

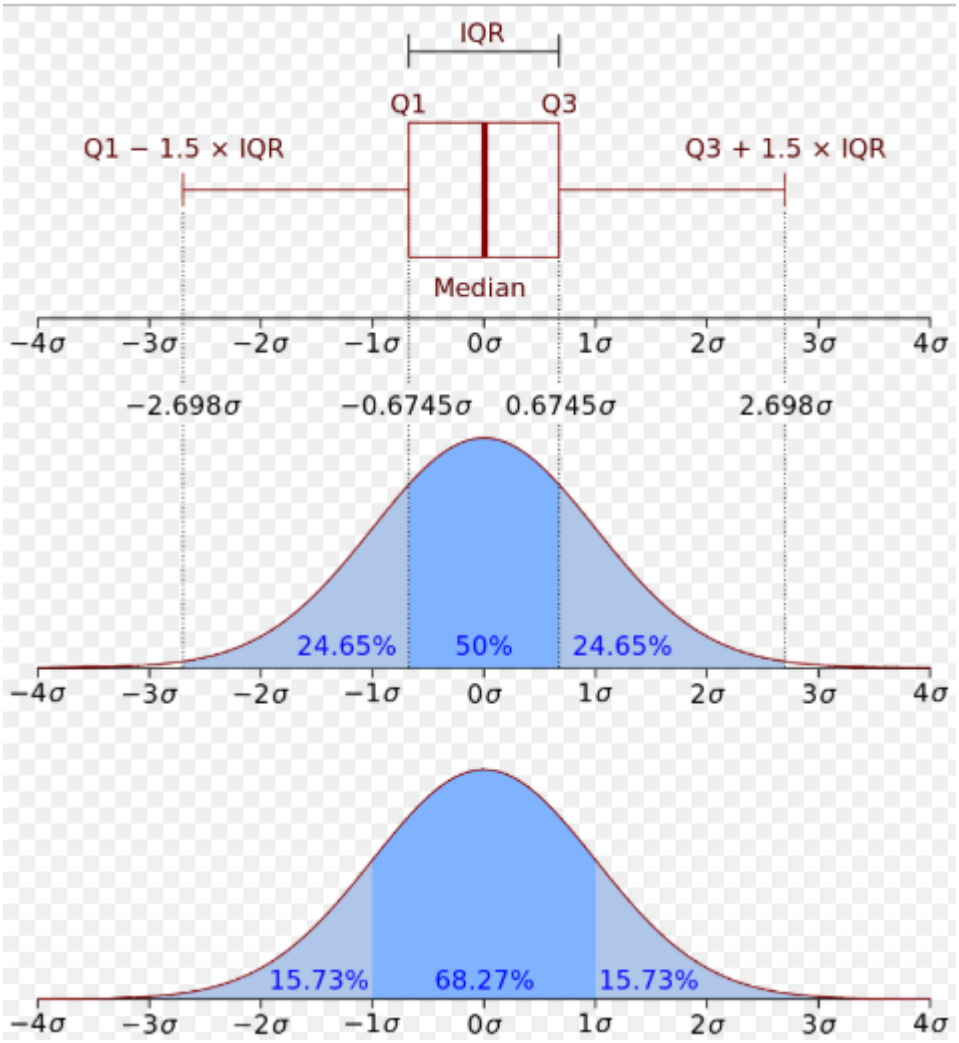
Methode 1 : Basé sur des statistiques paramétriques ; utilisez la moyenne

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())
```

Créer des sous-groupes

 # Group by

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

 # Fichier I / O

Statistiques descriptive

Dans cet étude on utilisera les fichiers de données « Comorbid.xlsx»

```
# install xlrd
Conda install –y xlrd
## read from csv
import pandas as pd
import numpy as np

# importer son jeu de doonnées
df = pd.read_excel("Comorbid.xlsx")
# afficher les deux premieres lignes du jeu de données
df.head(2)
# afficher les colonnes
df.columns

## parametres de position
# la moyenne
df.Age.mean()
#la mediane
df.Age.median()
#le mode
df.DiabetesMellitus.mode()
df.DiabetesMellitus.value_counts()
df.DiabetesMellitus.value_counts(normalize = True)*100
# les quantiles
df.Age.quantile([0, 0.25,0.5, 0.75, 0.9, 1])
np.percentile(df.Age.values,[0, 25,50, 75, 90, 100])
# la moyenne des ages des patients de grade A
df[df.Grade=='A']
df[df.Grade=='A'].Age
df[df.Grade=='A'].Age.mean()
# le,maximum et le minimum
df[df.Grade=='A'].Age.min()
df[df.Grade=='A'].Age.max()
## parametres de dispersion
df.Age.std()
df.Age.var()
df.Age.var()

# l'intervalle interquartle
from scipy import stats
stats.iqr(df.Age)

# statistique descriptive avec describe
df.Age.describe()
```

Représentations graphiques

Dans cet étude on utilisera les fichiers de données « Salary_Data.csv »

```
# Importer les librairies
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

## Representations graphiques
#########
```

```
# lien
# http://www.python-simple.com/python-matplotlib/scatterplot.php
# https://matplotlib.org/3.1.1/gallery/index.html

### scarter plot ( nuage de point)
# Importer le dataset
dataset = pd.read_csv('Salary_Data.csv')
# afficher les deux premieres lignes du jeu de données
dataset.head(4)
# afficher les colonnes
dataset.columns

#######
# le scarter plot
#######

x = dataset['YearsExperience']
y = dataset['Salary']

plt.scatter(x, y)   # , edgecolors='r'
plt.xlabel('Années d\'experience')
plt.ylabel('salaire')
plt.title('Evolution du salaire')
plt.show()

#######
# Boxplot
#######

box_plot_data=[x]
plt.boxplot(box_plot_data)
plt.show()

box_plot_data=[x]
plt.boxplot(box_plot_data,labels=['Années d\'experience'])
plt.show()

box_plot_data=[x]
plt.boxplot(box_plot_data,patch_artist=True,labels=['Années d\'experience'])
plt.show()

box_plot_data=[x]
plt.boxplot(box_plot_data,patch_artist=True,labels=['Années
d\'experience'],vert=0)
plt.show()

value1 = dataset[dataset.grade=='A'].Salary
value2=dataset[dataset.grade=='B'].Salary

box_plot_data=[value1,value2]
plt.boxplot(box_plot_data,)
plt.show()

#######
# Histogrammes
#######

y = dataset['Salary']
plt.hist(y,5, histtype='bar',
```

```
align='mid', color='c', label='Salaire',edgecolor='black')
plt.legend()
plt.title('Histogramme des salairs')
plt.show()

y = dataset['Salary']
plt.hist(y,5,
align='mid', color='c', label='Salaire',edgecolor='black', orientation =
'horizontal')
plt.legend()
plt.ylabel('Salaire')
plt.xlabel('Frequences')
plt.title('Histogramme des salairs')
plt.show()

## histogrammes supperposés des ages dans les groupes A et B
value1 = dataset[dataset.grade=='A'].Salary
value2=dataset[dataset.grade=='B'].Salary

plt.hist([value1, value1], bins = 5, color = ['yellow', 'green'],
            edgecolor = 'red', hatch = '/', label = ['A', '.B'],
            histtype = 'barstacked')
plt.ylabel('valeurs')
plt.xlabel('nombres')
plt.title('2 series superposees')
plt.legend()
plt.show()

#######
# bar plot
#######

### Barplot simple
# Creer fake dataset:
haut = [3, 12, 5, 18, 45]
bars = ('A', 'B', 'C', 'D', 'E')
y_pos = np.arange(len(bars))
# Creer les barres
plt.bar(y_pos, haut)
# ajouter des nons a l'axe des abcisses
plt.xticks(y_pos, bars)
# afficher le graphe
plt.show()

# ou

plt.bar(range(5), [3, 12, 5, 18, 45], width = 0.6, color = 'yellow',
  edgecolor = 'blue', linewidth = 2,
  ecolor = 'magenta', capsize = 10)
plt.xticks(range(5), ['A', 'B', 'C', 'D', 'E'])
# afficher le graphe
plt.show()

### Barplot double
data = [[5., 25., 50., 20.],[4., 23., 51., 17.]]
X = np.arange(4)
plt.bar(X + 0.00, data[0], color = 'b', width = 0.25)
plt.bar(X + 0.25, data[1], color = 'g', width = 0.25)
plt.show()
```

```
#######
# Pie / camembert
#######

# pie plot simple

x = [1, 2, 3, 4, 10]
plt.pie(x, labels = ['A', 'B', 'C', 'D', 'E'])
plt.show()

# pie plot simple somme des valeurs inferieur a 1
plt.figure(figsize = (8, 8))    # agrandir l'affichage
x = [0.1, 0.2, 0.3, 0.1]
plt.pie(x, labels = ['A', 'B', 'C', 'D'])
plt.legend()
plt.show()

# exemple complet
plt.figure(figsize = (8, 8))
x = [1, 2, 3, 4, 10]
plt.pie(x, labels = ['A', 'B', 'C', 'D', 'E'],
           colors = ['red', 'green', 'yellow'],
           explode = [0, 0.2, 0, 0, 0],
           autopct = lambda x: str(round(x, 2)) + '%',
           pctdistance = 0.7, labeldistance = 1.4,
           shadow = True)
plt.legend()
plt.show()

# NB: nous pouvez utiliser ploty express pour vos representations graphiques
# https://plot.ly/python/bar-charts/
```