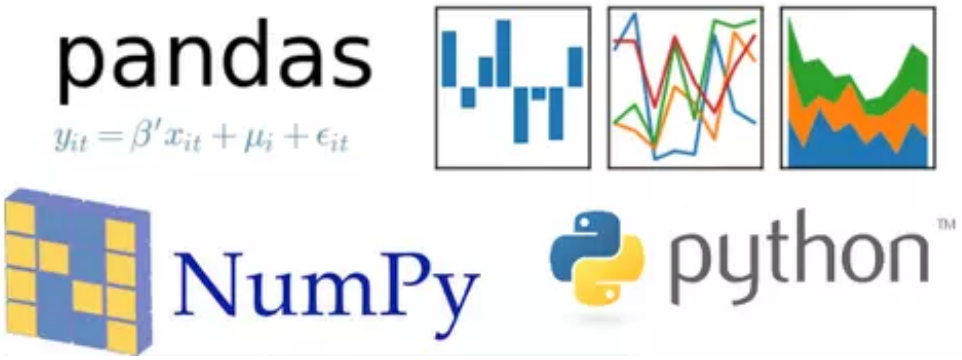
Python : pour aller plus loin que la calculatrice



SOMMAIRE

PRÉSENTATION

- **Objectif : utiliser Python pour générer et traiter des données statistiques**
- **Le contexte : un lycée du Val de Marne**
  - les élèves reçoivent un ordinateur portable à leur entrée en Seconde;
  - toutes les salles sont équipées du Wifi.
- **Le cadre :**
  - les programmes de mathématiques du lycée général et technologique;
  - un niveau de technicité Python raisonnable
- **L'outil: l'application Capytale**
  - disponible sur l'ENT;
  - permet de générer, partager, récupérer des Notebook Jupyter;
  - particulièrement adapté pour conserver une trace propre des sorties .

I Comparaison des temps d'exécution de différentes fonctions

- **Import des bibliothèques utiles**

[32]:
```
from math import * # pour acceder a la fonction racine carree
from time import * # pour mesurer le temps
import matplotlib.pyplot as plt # pour faire des graphiques
import numpy as np # pour faire des stats
```

1.1 Partie 1 : tests de primalité

1.1.1 A - Niveau Seconde

- **Définition de fonctions test**

On définit 2 fonctions testant la primalité des entiers naturels **impairs** supérieurs ou égaux à 3.

[2]:
```
def premier1(n):
    '''teste la divisibilité de n par tous les impairs de 3 à  n - 1'''
    k = 3
    while k <= n - 1 :
        if n % k == 0:
            return 0
        k = k + 2
    return 1
```

[3]:
```
def premier2(n):
    '''teste la divisibilité par les entiers impairs de 3 à rac(n)'''
    k = 3
    while k <= sqrt(n):
        if n % k == 0:
            return 0
        k = k + 2
    return 1
```

- **Comparaison des performances des 2 fonctions**

On appelle chacune des fonctions 100 fois pour exécuter une même tache : tester la primalité des nombres impairs de 3 à 15000.

Pour mesure le temps d'exécution, on utilise pour cela la fonction `perf_counter` (ou `process_time`) de la bibliothèque `time`.

[4]:
```
temps1 = [] # liste qui va contenir les temps d'exécution de la fonction premier1

for k in range(100):
    start = perf_counter()
    for n in range(3, 15000, 2):
        premier1(n)
    end = perf_counter()
```

```
    t = end - start
    temps1.append(t) #ajoute la valeur de la variable t à la liste temps1
```

[5]:
```
m1 = np.mean(temps1)
s1 = np.std(temps1)
print(m1, s1)
```

```
1.8317432610000015 0.14305204722220957
```

[6]:
```
temps2 = [] # liste qui va contenir les temps d'exécution de la fonction premier2
for k in range(100):
    start = perf_counter()
    for n in range(3, 15000, 2):
        premier2(n)
    end = perf_counter()
    t = end - start
    temps2.append(t) #ajoute la valeur de la variable t à la liste temps2
```

[7]:
```
m2 = np.mean(temps2)
s2 = np.std(temps2)
print(m2, s2)
```

```
0.0418152050000026 0.003060836116794697
```

- **Comparaison des moyennes et écarts type**

[8]:
```
print(m1/m2)
```

```
43.80567453872072
```

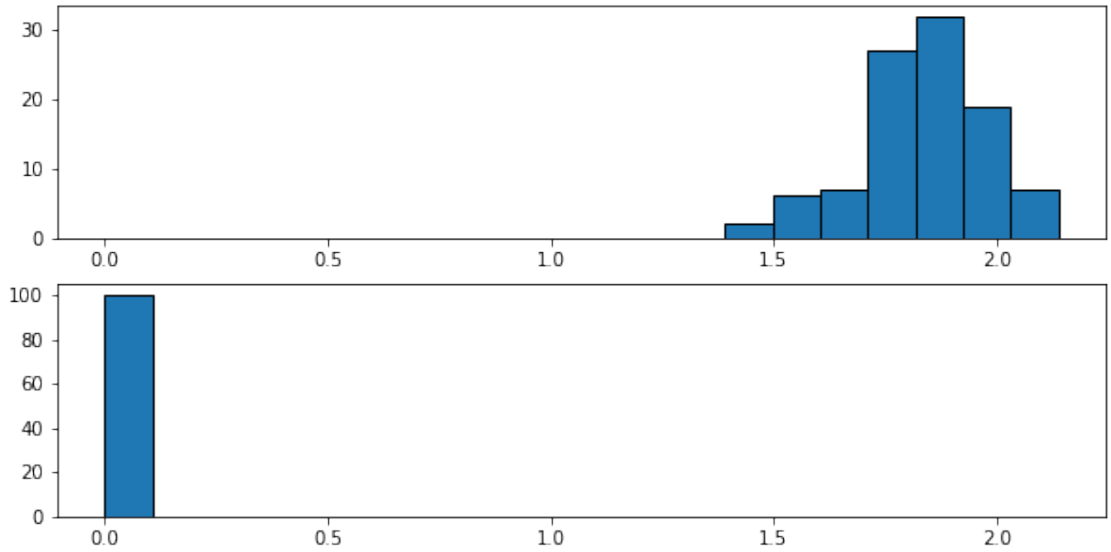
[9]:
```
print(s1/s2)
```

```
46.73626478637264
```

- **Représentation graphique avec le même axe des abscisses**

[10]:
```
plt.clf()
plt.figure(figsize=(10, 5))
plt.subplot(211)
plt.hist(temps1, range = (0, max(temps1)), bins = 20, edgecolor = 'black')
plt.subplot(212)
plt.hist(temps2, range = (0, max(temps1)), bins = 20, edgecolor = 'black')
plt.show()
```

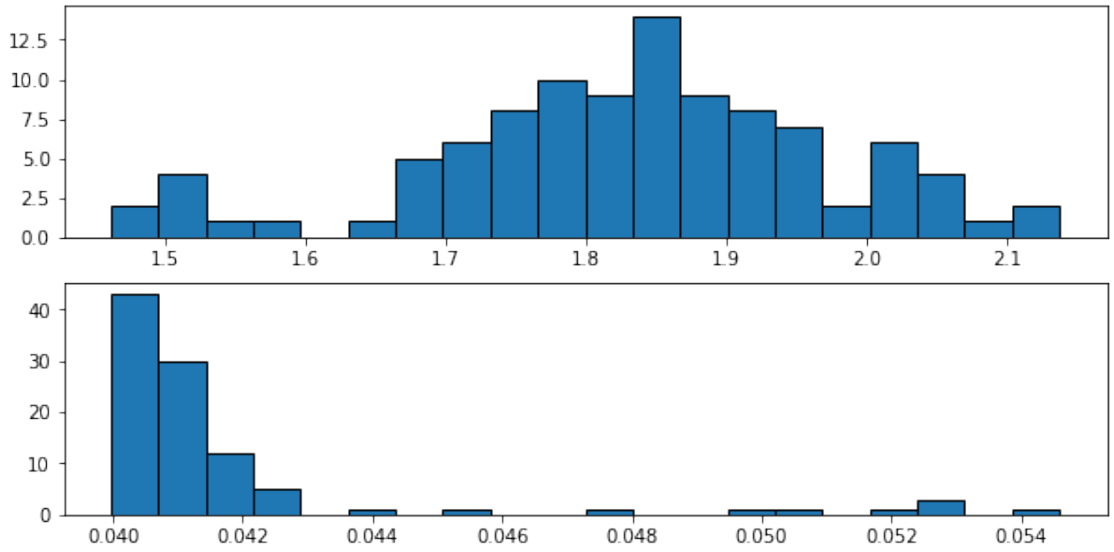
```
<Figure size 432x288 with 0 Axes>
```



Avec un axe des abscisses adapté à chacune des fonctions

[11]:
```
plt.clf()
plt.figure(figsize=(10,5))
plt.subplot(211)
plt.hist(temps1, bins = 20, edgecolor = 'black')
plt.subplot(212)
plt.hist(temps2, bins = 20, edgecolor = 'black')
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



- **Intervalle 2σ**

[12]:
```
c1 = 0
for t in temps1:
    if m1 - 2 * s1 <= t <= m1 + 2 * s1 :
        c1 = c1 + 1
print(c1 / len(temps1))
```

```
0.91
```

[13]:
```
c2 = 0
for t in temps2:
    if m2 - 2 * s2 <= t <= m2 + 2 * s2 :
        c2 = c2 + 1
print(c2 / len(temps2))
```

```
0.93
```

1.1.2 B - Niveau Maths expertes : test de Lucas-Lehmer et nombres de Mersenne

Soit p un nombre premier supérieur ou égal à 3 et $M_p = 2^p - 1$ le nombre de Mersenne correspondant.

Soit (a_n) la suite définie par $a_0 = 4$ et, pour tout entier naturel n, a_{n+1} est le reste de la division euclidienne de $a_n^2 - 2$ par M_p.

Alors M_p est premier si et seulement si $a_{p-2} = 0$.

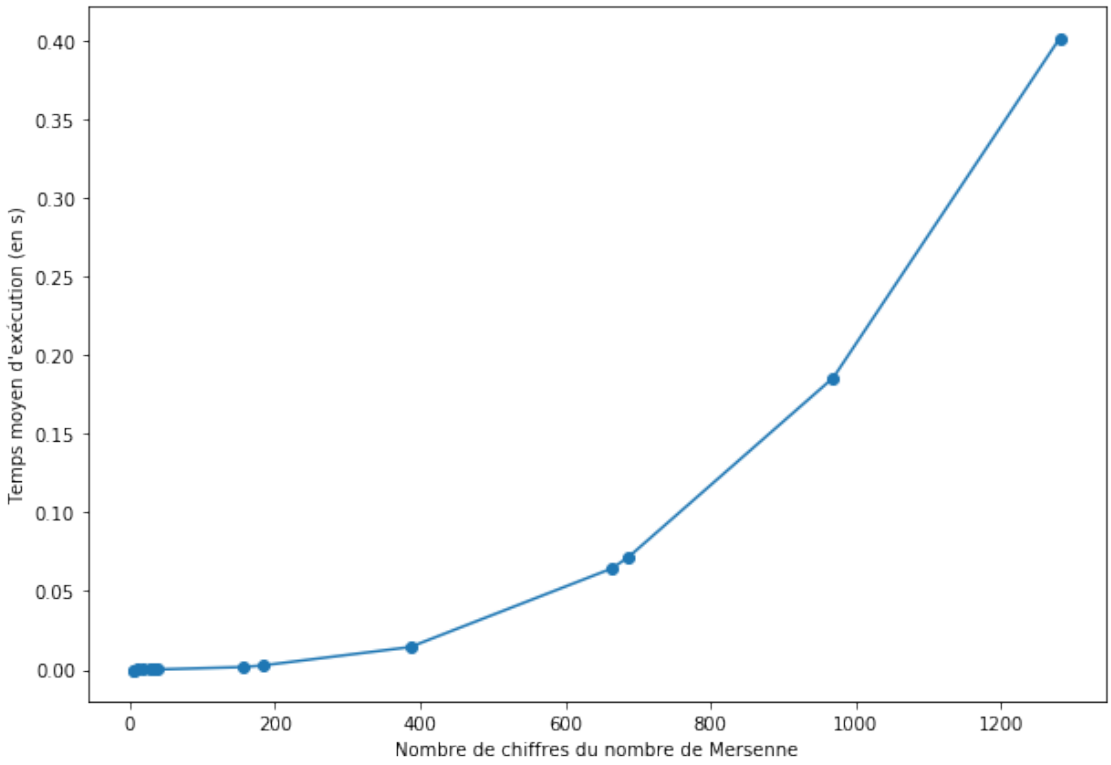
[14]:
```
def lucas(p):
    M = 2 ** p - 1
    a = 4
    for k in range(1, p - 1):
        a = (a ** 2 - 2) % M
    if a  == 0:
        return 1
    else:
        return 0
```

- **Temps d'exécution en fonction du nombre de chiffres des nombre de Mersenne premiers**

[15]:
```
p_list = [17, 19, 31, 61, 89, 107, 127, 521, 607, 1279, 2203, 2281, 3217, 4253]
chif_list=[]
moy_list = []
for p in p_list :
    n = 2 ** p - 1
    long = len(str(n))
    chif_list.append(long)
    temps4 = []
```

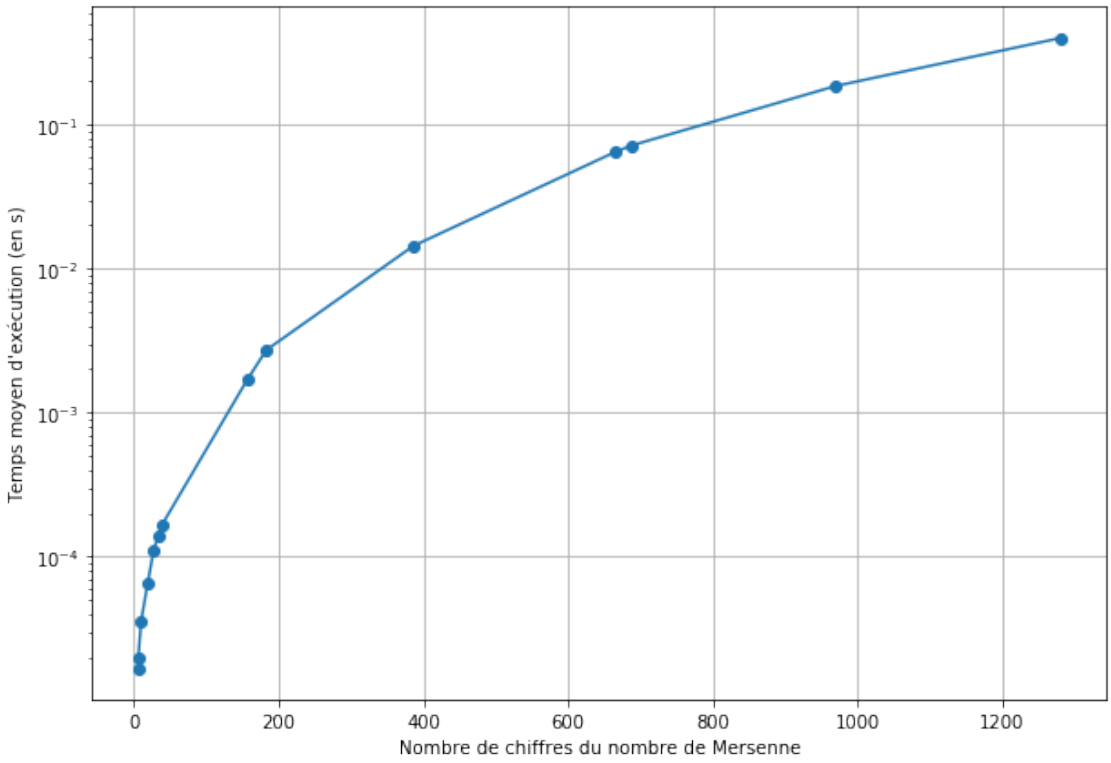
```
    for k in range(100):
        start = perf_counter()
        lucas(p)
        end = perf_counter()
        t = end - start
        temps4.append(t)
    moy_list.append(np.mean(temps4))
```

[16]:
```
plt.figure(figsize=(10,7))
plt.plot(chif_list, moy_list, marker='o')
plt.xlabel('Nombre de chiffres du nombre de Mersenne')
plt.ylabel("Temps moyen d'exécution (en s)")
plt.show()
```



- **Temps d'exécution en fonction du nombre de chiffres dans un repère semi-log**

[17]:
```
plt.figure(figsize=(10,7))
plt.plot(chif_list, moy_list, marker='o')
plt.yscale('log')
plt.xlabel('Nombre de chiffres du nombre de Mersenne')
plt.ylabel("Temps moyen d'exécution (en s)")
plt.grid()
plt.show()
```



1.2 Partie 2 : Complexité

- Soit f la fonction définie par $f(x) = 5x^3 - 3x^2 + 2x + 1$.

On étudie l'influence de l'écriture de l'expression de la fonction f sur le temps de calcul des images des entiers de 1 à 200000.

[18]:
```
def f1(x) :
    y = 5 * x ** 3  - 3 * x ** 2 + 2 * x + 1
    return y

def f2(x) :
    y = 5 * x * x * x  - 3 * x * x + 2 * x + 1
    return y

def f3(x) :
    y = x  * (x * (5 * x - 3) + 2) + 1
    return y
```

[19]:
```
temps5 = []
for k in range(100):
    start = perf_counter()
    for x in range(1, 200001):
```

```
        f1(x)
    end = perf_counter()
    t = end - start
    temps5.append(t)
```

[20]:
```
m5 = np.mean(temps5)
s5 = np.std(temps5)
print(m5, s5)
```

```
0.4229342630000002 0.0287977058730589
```

[21]:
```
temps6 = []
for k in range(100):
    start = perf_counter()
    for x in range(1, 200001):
        f2(x)
    end = perf_counter()
    t = end - start
    temps6.append(t)
```

[22]:
```
m6 = np.mean(temps6)
s6 = np.std(temps6)
print(m6, s6)
```

```
0.18386876500000254 0.012729721749414873
```

[23]:
```
temps7 = []
for k in range(100):
    start = perf_counter()
    for x in range(1, 200001):
        f3(x)
    end = perf_counter()
    t = end - start
    temps7.append(t)
```

[24]:
```
m7 = np.mean(temps7)
s7 = np.std(temps7)
print(m7, s7)
```

```
0.129814796 0.009765535148706744
```

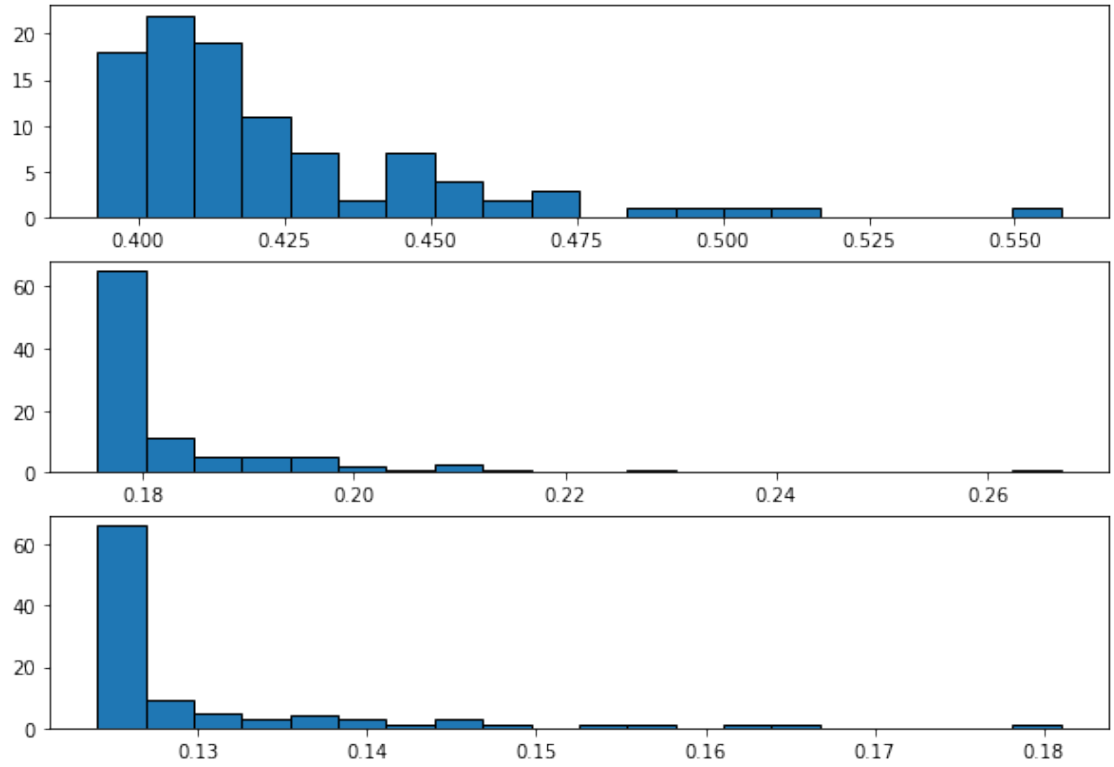
[25]:
```
print(m5/m6)
print(m6/m7)
```

```
2.3001963547206854
1.416392974187646
```

[26]:
```
plt.clf()
plt.figure(figsize=(10,7))
```

```
plt.subplot(311)
plt.hist(temps5, bins = 20, edgecolor = 'black')
plt.subplot(312)
plt.hist(temps6, bins = 20, edgecolor = 'black')
plt.subplot(313)
plt.hist(temps7, bins = 20, edgecolor = 'black')
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



- **Import des bibliothèques utiles**

[33]:
```
from math import *
from random import * # pour simuler le hasard
import matplotlib.pyplot as plt # pour faire des graphiques
import numpy as np # pour faire des stats
```

Un professeur rend aléatoirement les copies aux élèves d'une classe. On note X la variable aléatoire donnant le nombre d'élèves de la classe ayant reçu leur copie. On cherche à estimer $E(X)$.

Ce problème est connu sous différents noms: * le problème des chapeaux; * le jeu du Treize (Rémond de Montmort en 1708) * permutations à points fixes

2.1 Partie 1 : loi de X pour une taille de classe fixée

Pour simuler la variable X, on utilise la fonction `shuffle`de la bibliothèque `random`.

[2]:
```
def simul(n):
    eleves = list(range(n)) # liste des entiers de 0 à n - 1
    shuffle(eleves)
    x = 0
    for k in range(n):
        if eleves[k] == k:
            x = x + 1
    return x
```

[3]:
```
simul(34)
```

[3]:
```
1
```

- On simule $n = 1000$ réalisations de la variable X.

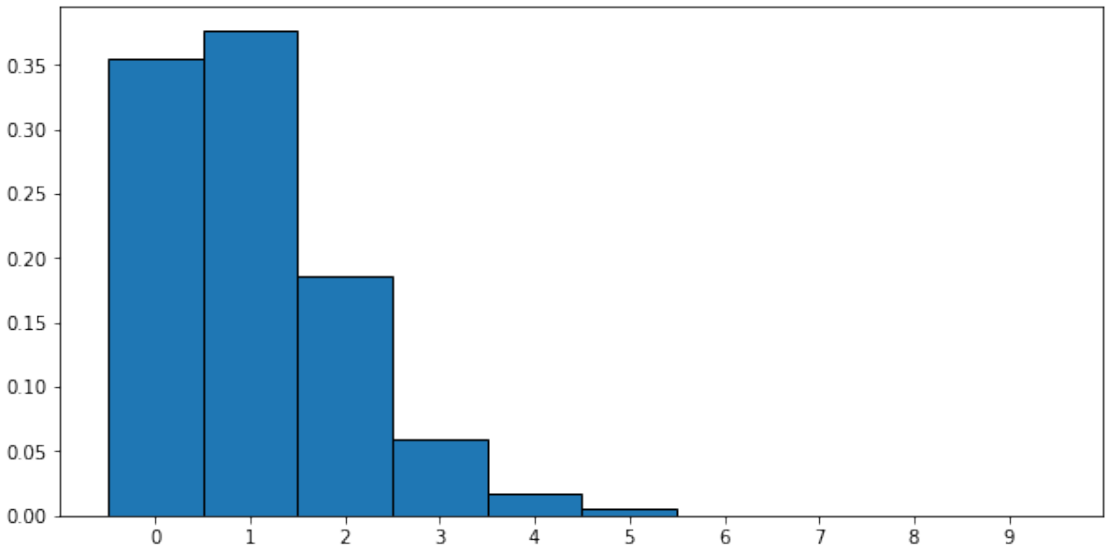
[4]:
```
n = 1000
```

[5]:
```
echantillon = [simul(34) for k in range(n)]
```

- **Représentation graphique de la distribution des n valeurs simulées de X**

[6]:
```
plt.clf()
plt.figure(figsize=(10, 5))
centres = [k + 0.5 for k in range(-1, 10)]
plt.hist(echantillon, bins = centres, edgecolor = 'black', density = True)
plt.xticks([k for k in range(10)])
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



- **Estimation de l'espérance de X**

[7]:
```
m = np.mean(echantillon)
m
```

[7]: 1.022

[8]:
```
sigma = np.std(echantillon)
sigma
```

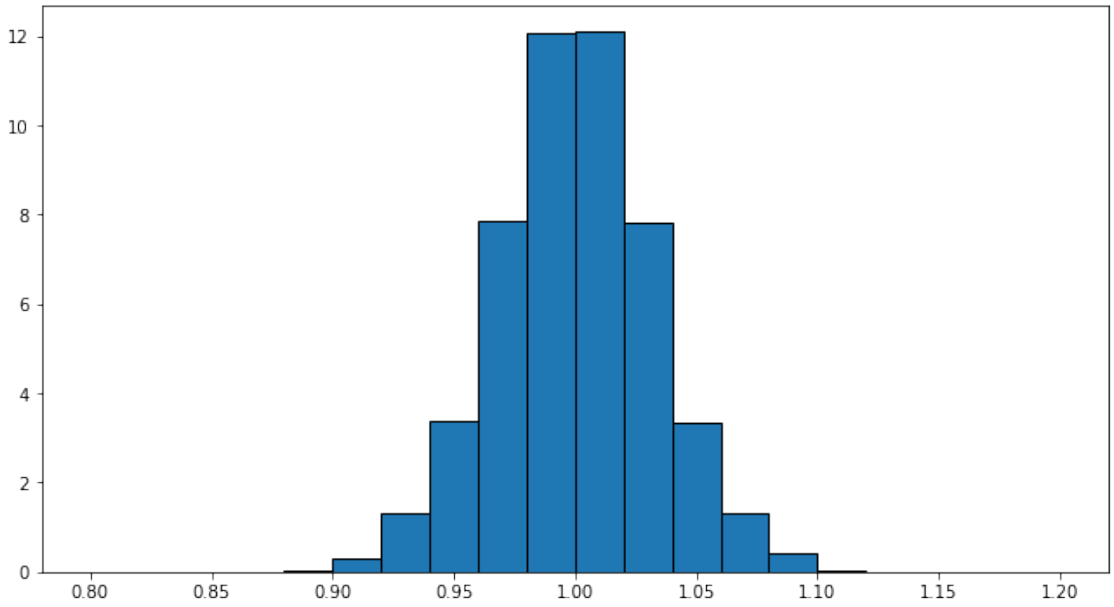
[8]: 1.0067353177474208

- **2000 estimations de l'espérance**

[9]:
```
liste_moy = []
for p in range(2000):
    echantillon = [simul(34) for k in range(n)]
    m = np.mean(echantillon)
    liste_moy.append(m)
```

[15]:
```
plt.clf()
plt.figure(figsize=(11,6))
plt.hist(liste_moy, bins = 20, range =(0.8, 1.2), edgecolor = 'black', density =␣
 ↪True)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



- **Intervalle** 2σ

[11]:
```
moy = np.mean(liste_moy)
s = np.std(liste_moy)
c = 0
for m in liste_moy:
    if moy - 2 * s <= m <= moy + 2 * s:
        c = c + 1
print(c/len(liste_moy))
```

```
0.951
```

- **Ecart type de** X **vs écart type** $\overline{X}$**, moyenne sur des échantillons de taille** $n = 1000$

Écart type de X estimé sur $n = 1000$ réalisations de X

[16]:
```
echantillon = [simul(34) for k in range(n)]
sigma = np.std(echantillon)
print(sigma)
```

```
0.9911609354691093
```

Écart type $\overline{X}$ estimé sur 2000 réalisations de $\overline{X}$, moyenne sur des échantillons de taille $n = 1000$.

[19]:
```
print(s)
```

```
0.031629794672112556
```

- **Ecart type de** X **vs écart type** $\overline{X}$**, moyenne sur des échantillons de taille** $n = 1000$

Comparaison des écarts type et propriété $\sigma(\overline{X}) = \dfrac{\sigma(X)}{\sqrt{n}}$

[20]:
```
print(sigma/s, sqrt(1000))
```

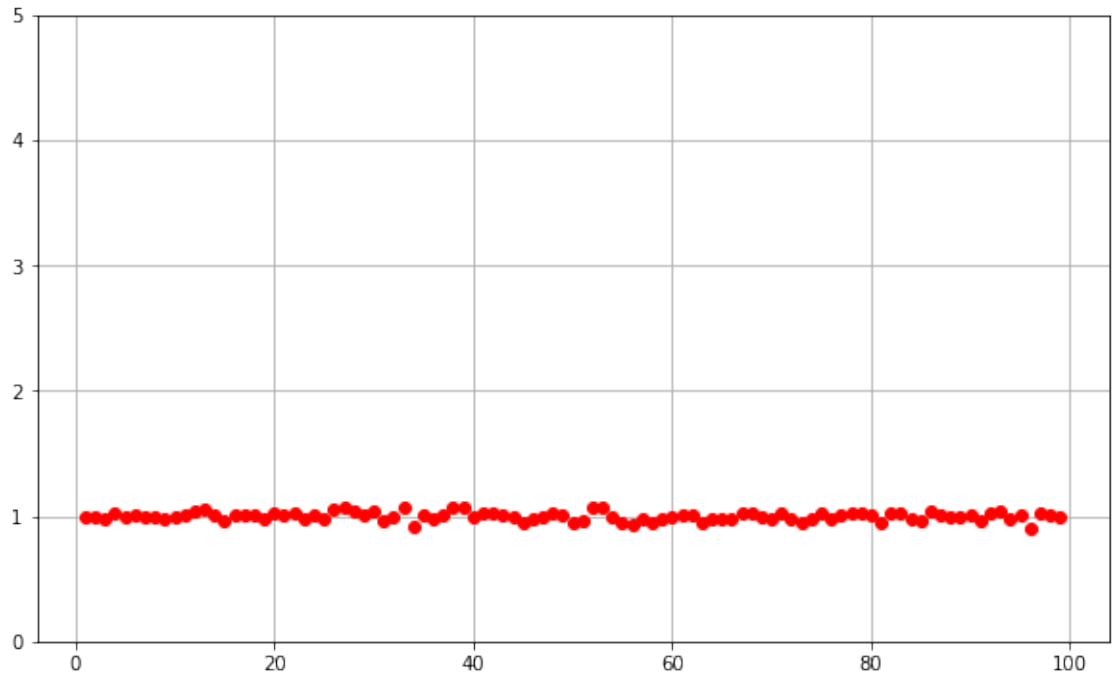
```
31.3363063448211 31.622776601683793
```

2.2 Partie 2 : Espérance X en fonction de la taille de la classe

[13]:
```
liste_moy2 = []
for taille in range(1, 100):
    echantillon = [simul(taille) for k in range(1000)]
    moy2 = np.mean(echantillon)
    liste_moy2.append(moy2)
```

[14]:
```
plt.clf()
plt.figure(figsize=(10,6))
plt.plot(range(1, 100), liste_moy2, 'ro')
plt.ylim(0, 5)
plt.grid(True)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



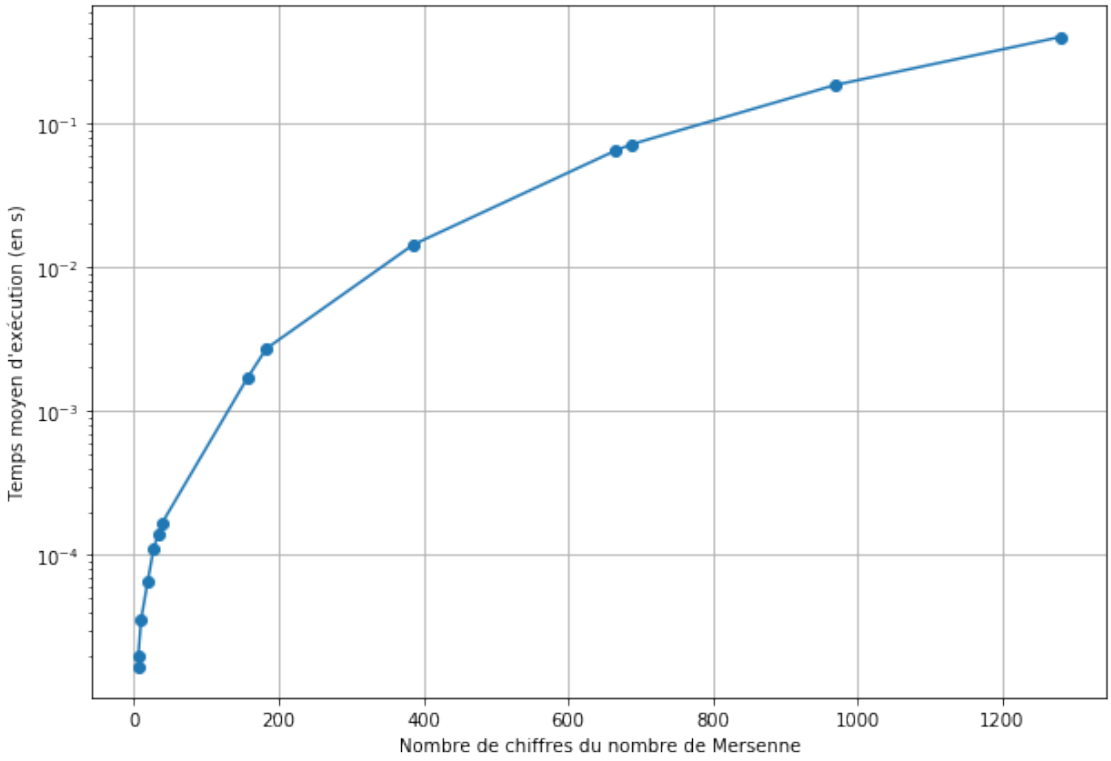
- On peut conjecturer que, quelle que soit la taille de la classe, l'espérance de X est égale à 1.

- Ce résultat peut se démontrer en Spécialité Terminale, dans le chapitre *Somme de variables aléatoires*
- **Prolongement : on peut estimer la probabilité $P(X = 0)$ d'obtenir un dérangement.**

Celle-ci tend vers $\frac{1}{e}$ lorsque la taille de la classe tend vers $+\infty$

[40]:
```
liste_derang = []
for taille in range(1, 1000):
    echantillon = [simul(taille) for k in range(1000)]
    derang = echantillon.count(0)
    liste_derang.append(derang / 1000)
```

[39]:
```
plt.figure(figsize=(10,6))
plt.plot(range(1, 1000), liste_derang, linestyle ='none', color= 'blue', marker␣
↪= 'o')
plt.plot((1, 1000), (1 / exp(1), 1 / exp(1)), linestyle ='--', color = 'red')
plt.show()
```



III ÉTUDE DE DONNÉES PROVENANT D'UN FICHIER EXTERNE

- **Import des bibliothèques utiles**

[1]:
```
import pandas # pour lire et étudier un fichier de donnees
import matplotlib.pyplot as plt # pour faire des graphiques
```

3.1 Activité 1 : température corporelle aux États-Unis

Selon une étude conduite par des chercheurs de l'université de Stanford publiée le 7 janvier 2020, aux États-Unis, la température corporelle aurait diminué depuis la révolution industrielle. Les chercheurs de cette université ont comparé trois vastes cohortes : * 23 710 vétérans de la guerre civile américaine, suivis entre 1862 et 1930 ; * 15 301 personnes enrôlées dans une étude de nutrition entre 1971 et 1975 ; * 150 280 personnes intégrées dans une cohorte suivie par Stanford entre 2007 et 2017.

Soit au total 677423 mesures de température à analyser.

Source : Le Monde, 15 Janvier 2020, Hervé Morin

- **Lecture du fichier de données**

La bibliothèque `pandas` (contraction de *panel data*) permet de lire et d'étudier les données contenues dans le fichier `temperatures2.csv`.

[2]:
```
donnees = pandas.read_csv('temperatures2.csv', sep=';')
```

Les données sont stockées dans un tableau de données (un *data frame* dans le vocabulaire `pandas`) nommé `donnees`.

- **Visualisation du tableau de données**

[3]:
```
donnees.shape
```

[3]:
```
(677423, 3)
```

[4]:
```
donnees.head()
```

[4]:
```
   exam_year sex    temp
0     1891.0   m  36.727
1     1901.0   m  36.894
2     1902.0   m  37.061
3     1890.0   m  36.838
4     1891.0   m  36.838
```

[5]:
```
donnees.tail()
```

[5]:
```
        exam_year sex    temp
677418     2009.0   m  36.500
677419     2017.0   m  36.833
677420     2015.0   f  36.666
```

```
677421     2015.0   f  36.500
677422     2015.0   f  37.055
```

- **Températures des vétérans de la guerre civile : un exemple guidé**

On extrait les lignes correspondant à une année d'examen antérieure à 1930 et on nomme `veterans` ce nouveau sous-tableau.

[6]:
```
veterans = donnees.query('exam_year <= 1930')
```

[7]:
```
veterans.tail()
```

[7]:
```
       exam_year sex    temp
83895     1898.0   m  35.783
83896     1892.0   m  36.061
83897     1898.0   m  36.616
83898     1900.0   m  36.616
83899     1898.0   m  36.616
```

[8]:
```
veterans.shape
```

[8]:
```
(83780, 3)
```

On récupère ensuite la colonne des températures (objet de type `Series`dans le vocabulaire `pandas`).

[9]:
```
veterans_temp = veterans['temp']
```

[10]:
```
veterans_temp
```

[10]:
```
0        36.727
1        36.894
2        37.061
3        36.838
4        36.838
          ...
83895    35.783
83896    36.061
83897    36.616
83898    36.616
83899    36.616
Name: temp, Length: 83780, dtype: float64
```

On analyse les données de cette série statistique grâce à la méthode `describe`

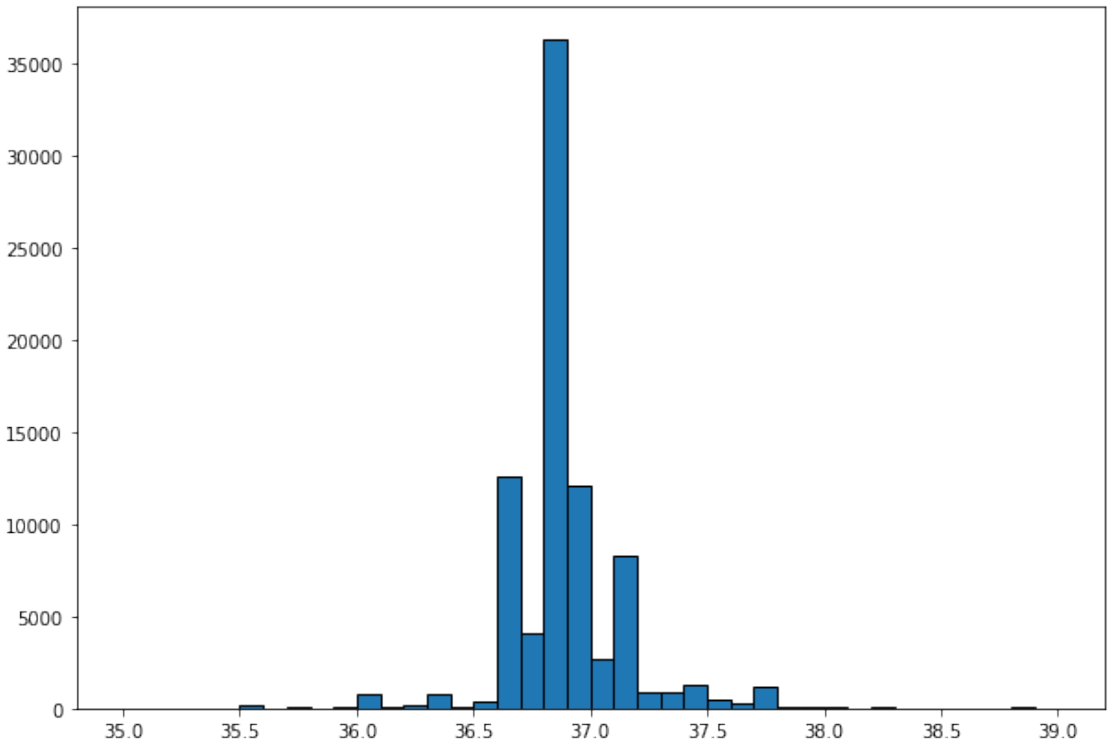
[11]:
```
veterans_temp.describe()
```

[11]:
```
count    83780.000000
mean        36.897912
std          0.263062
min         35.116000
```

```
25%          36.838000
50%          36.894000
75%          36.949000
max          38.949000
Name: temp, dtype: float64
```

On représente graphiquement les données par un histogramme.

[12]:
```
bornes = [35 + 0.1 * k for k in range(0, 41)]
plt.figure(figsize = (10,7))
plt.hist(veterans_temp, bins = bornes, edgecolor = 'black')
plt.show()
```



On détermine la proportion des températures dans la plage 2σ.

[13]:
```
n1 = veterans_temp.count()
m1 = veterans_temp.mean()
s1 = veterans_temp.std()
c1 = 0 # compteur
for t in veterans_temp :
    if m1 - 2 * s1 <= t <= m1 + 2 * s1:
        c1 = c1 + 1
print(c1 / n1)
```

```
0.9347576987347815
```

- **Bilan des instructions utilisées**

Action	Instruction
Lire un fichier de données	`donnees = pandas.read_csv('nom.csv', sep=';')`
Taille du tableau	`donnees.shape`
Premières lignes du tableau	`donnees.head()`
Dernières lignes du tableau	`donnees.tail()`
Extraire des données sous contraintes	`donnees.query('condition')`
Récupérer une colonne	`donnees['nom_colonne']`
Paramètres statistiques	méthode `describe()`

- **Étude des températures des deux autres cohortes : en autonomie**

[14]:
```
nutrition = donnees.query('exam_year <= 2022')
nutrition.shape
```

[14]:
```
(677303, 3)
```

[15]:
```
nutrition_temp = nutrition['temp']
nutrition_temp.describe()
```

[15]:
```
count    677303.000000
mean         36.692953
std           0.366009
min          35.055000
25%          36.500000
50%          36.722000
75%          36.894000
max          38.949000
Name: temp, dtype: float64
```

[16]:
```
stanford = donnees.query('exam_year >= 2007')
stanford.shape
```

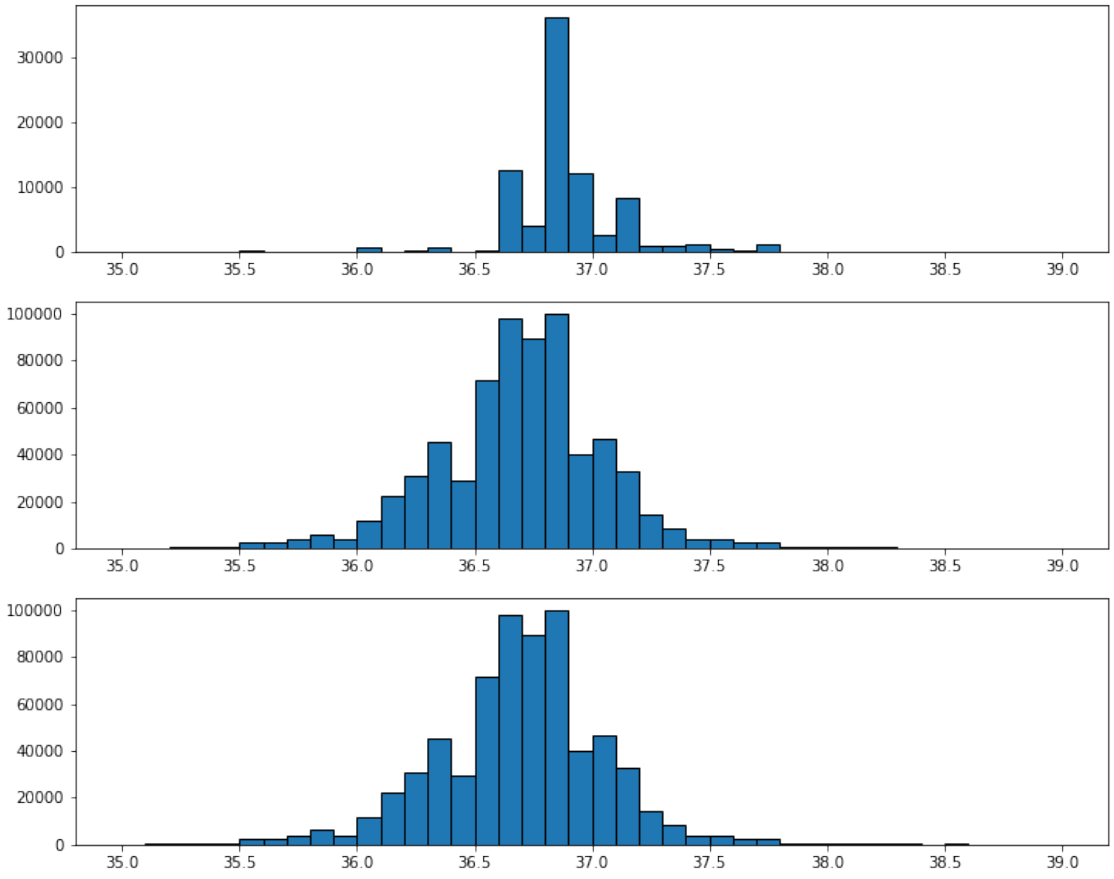
[16]:
```
(578222, 3)
```

[17]:
```
stanford_temp = nutrition['temp']
stanford_temp.describe()
```

[17]:
```
count    677303.000000
mean         36.692953
std           0.366009
min          35.055000
25%          36.500000
50%          36.722000
75%          36.894000
```

```
max           38.949000
Name: temp, dtype: float64
```

[18]:
```
bornes = [35 + 0.1 * k for k in range(0, 41)]
plt.figure(figsize = (12,10))
plt.subplot(3, 1, 1)
plt.hist(veterans_temp, bins = bornes, edgecolor = 'black')
plt.subplot(3, 1, 2)
plt.hist(nutrition_temp , bins = bornes, edgecolor = 'black')
plt.subplot(3, 1, 3)
plt.hist(stanford_temp , bins = bornes, edgecolor = 'black')
plt.show()
```



- **Évolution des températures hommes/femmes : devoir à la maison**

[34]:
```
homme_nutrition = nutrition.query('sex=="m"')
homme_nutrition_temp = homme_nutrition['temp']
homme_stanford = stanford.query('sex=="m"')
homme_stanford_temp = homme_stanford['temp']
```

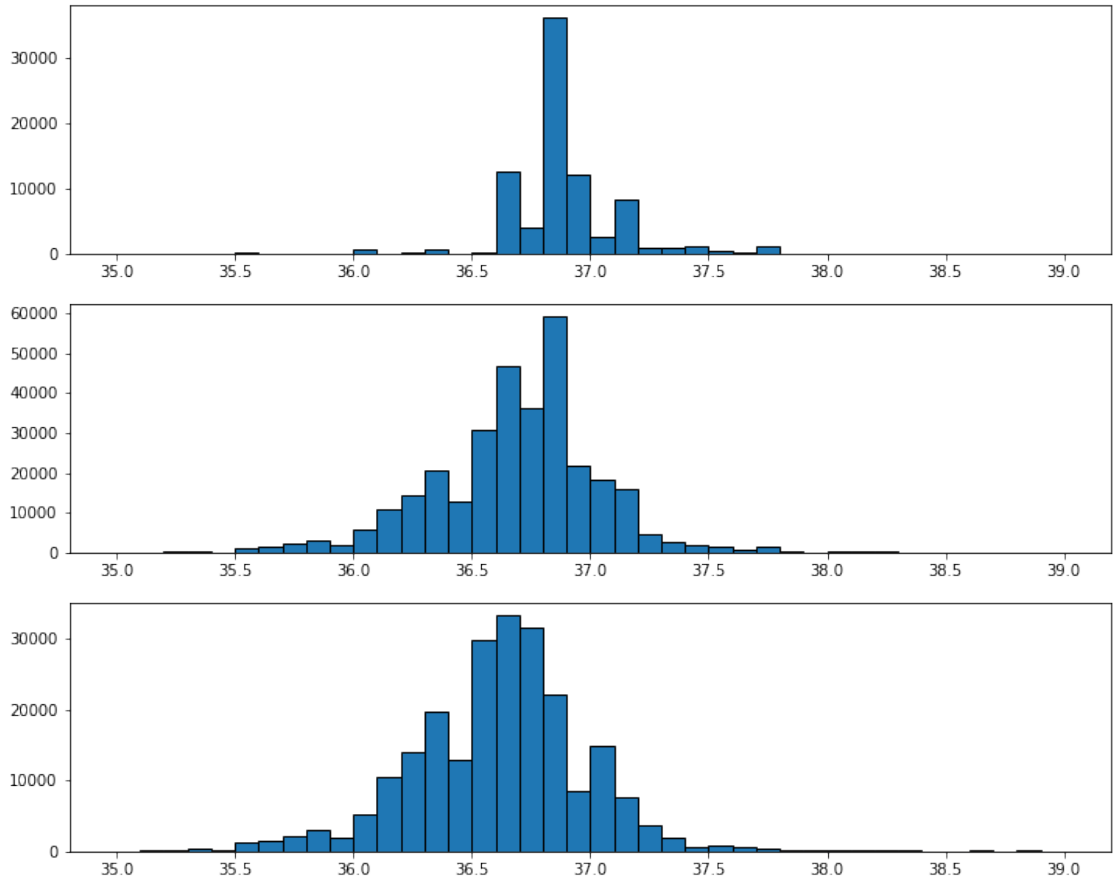
[35]:
```
homme_nutrition_temp.describe()
```

[35]:
```
count    319994.000000
mean         36.691111
std           0.367766
min          35.055000
25%          36.500000
50%          36.722000
75%          36.894000
max          38.949000
Name: temp, dtype: float64
```

[36]:
```
homme_stanford_temp.describe()
```

[36]:
```
count    230216.000000
mean         36.614153
std           0.372890
min          35.055000
25%          36.388000
50%          36.611000
75%          36.833000
max          38.944000
Name: temp, dtype: float64
```

[37]:
```
bornes = [35 + 0.1 * k for k in range(0, 41)]
plt.figure(figsize = (12,10))
plt.subplot(3, 1, 1)
plt.hist(veterans_temp, bins = bornes, edgecolor = 'black')
plt.subplot(3, 1, 2)
plt.hist(homme_nutrition_temp , bins = bornes, edgecolor = 'black')
plt.subplot(3, 1, 3)
plt.hist(homme_stanford_temp , bins = bornes, edgecolor = 'black')
plt.show()
```



[ ]:
```
manquant = 0
for k in range(677423):
    if donnees.loc[donnees.index[k],'exam_year'] not in list(range(1862, 2018)):
        #print(k, donnees.loc[donnees.index[k],'exam_year'])
        manquant = manquant + 1
print(manquant)
```

3.2 Activité 2 : Titanic et tableaux croisés

[45]:
```
titanic = pandas.read_csv('titanic.csv', sep = ';')
titanic.shape
```

[45]:
```
(1309, 6)
```

[48]:
```
titanic.head()
```

[48]:
```
   pclass  survived                                             name     sex  \
0       1         1                    Allen, Miss. Elisabeth Walton  female
```

```
1       1         1                    Allison, Master. Hudson Trevor    male
2       1         0                      Allison, Miss. Helen Loraine  female
3       1         0             Allison, Mr. Hudson Joshua Creighton    male
4       1         0  Allison, Mrs. Hudson J C (Bessie Waldo Daniels)  female

    age   fare
0  29.0  211.0
1   1.0  152.0
2   2.0  152.0
3  30.0  152.0
4  25.0  152.0
```

[49]:
```
titanic.tail()
```

[49]:
```
      pclass  survived                        name     sex   age  fare
1304       3         0       Zabour, Miss. Hileni  female  15.0  14.0
1305       3         0     Zabour, Miss. Thamine  female   NaN  14.0
1306       3         0  Zakarian, Mr. Mapriededer    male  27.0   7.0
1307       3         0        Zakarian, Mr. Ortin    male  27.0   7.0
1308       3         0         Zimmerman, Mr. Leo    male  29.0   8.0
```

- **Tableaux croisés d'effectifs**

[50]:
```
pandas.crosstab(titanic['sex'], titanic['survived'])
```

[50]:
```
survived    0    1
sex
female    127  339
male      682  161
```

[51]:
```
pandas.crosstab(titanic['pclass'], titanic['survived'])
```

[51]:
```
survived    0    1
pclass
1         123  200
2         158  119
3         528  181
```

- **Tableaux croisés et fréquences conditionnelles**

[54]:
```
pandas.crosstab(titanic['sex'], titanic['survived'], normalize = 'columns')
```

[54]:
```
survived         0      1
sex
female    0.156984  0.678
male      0.843016  0.322
```

[55]:
```
pandas.crosstab(titanic['sex'], titanic['survived'], normalize = 'index')
```

[55]:
```
survived         0         1
sex
female    0.272532  0.727468
male      0.809015  0.190985
```

[56]:
```
pandas.crosstab(titanic['pclass'], titanic['survived'], normalize = 'columns')
```

[56]:
```
survived         0      1
pclass
1         0.152040  0.400
2         0.195303  0.238
3         0.652658  0.362
```

[57]:
```
pandas.crosstab(titanic['pclass'], titanic['survived'], normalize = 'index')
```

[57]:
```
survived         0         1
pclass
1         0.380805  0.619195
2         0.570397  0.429603
3         0.744711  0.255289
```