Peut-on prédire les cours des cryptomonnaies ?

Aubin Ollivier

nº 38461

Qu’est ce qu’une cryptomonnaie ?

- Devise décentralisée
- Transactions inscrites sur la « blockchain »
- Cours très volatile et spéculatif

Bitcoin   *Ethereum*

Présentation du projet

- Création d'une base de données
- Reconnaître des tendances,

  formuler des lois empiriques

  → Optimisation des gains
- Implémentation des modèles

Création des bases de données

- **Première base de données :**
  Cours de six devises de 2017 à 2022
  → plus de 10 000 000 de valeurs

- Importation grâce au module Python CCXT

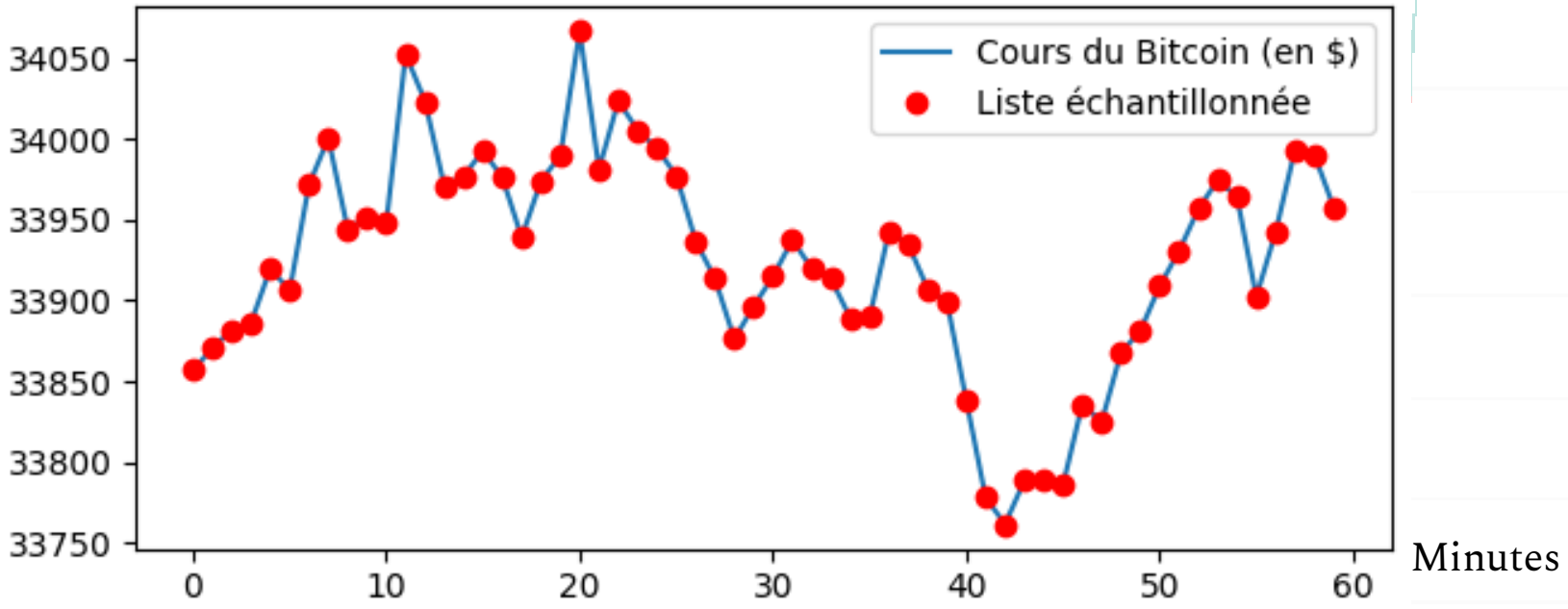
BINANCE

- Stockage dans des fichiers texte

- **Deuxième base de données** indépendante pour évaluer les modèles

Organisation des données

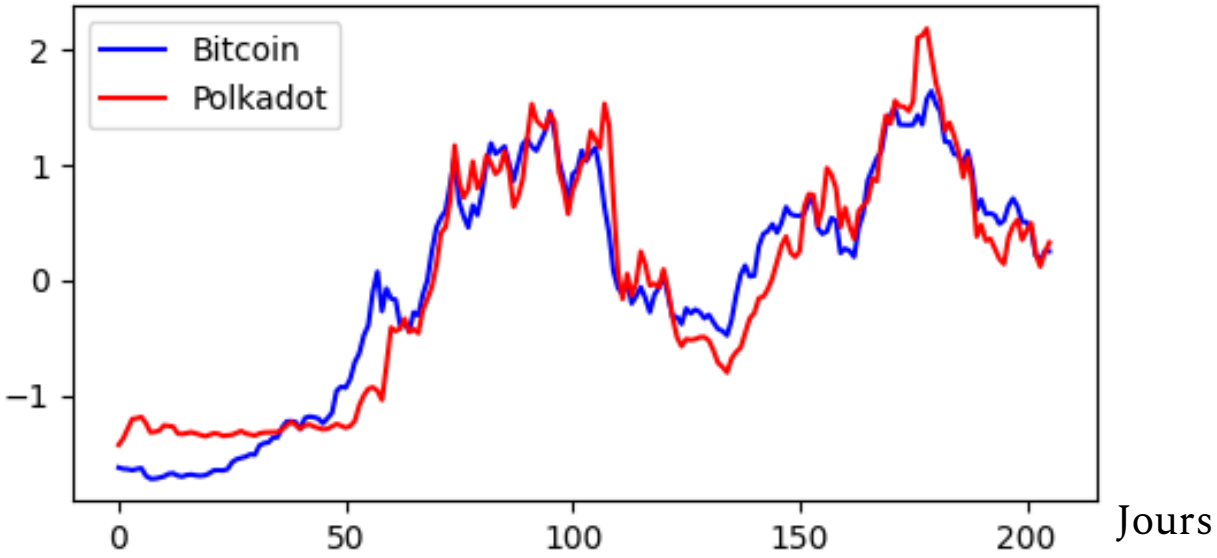
- Listes de 60 valeurs (par minutes) successives
- Chaque liste correspond à une plage d'une heure
- 3600 listes par jour



Exemple de liste

Étude de la corrélation des cours

- Coefficient de corrélation moyen de **0,82**



Cours du **Bitcoin** et du **Polkadot** d'août 2022 à janvier 2022

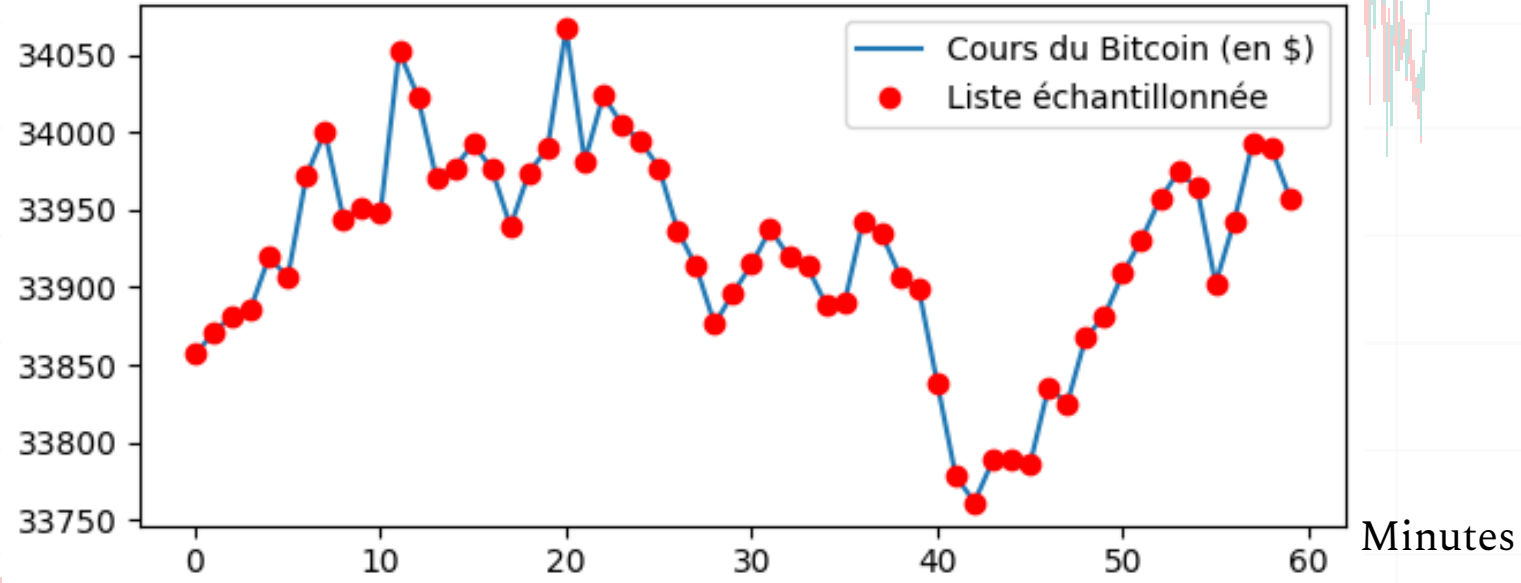
r = 0,95

Première modélisation : **Codage sur un bit**

- Chaque valeur est codée sur un bit

  (0 si le cours est en baisse, 1 s'il est en hausse)

- Listes de 59 bits




```
11111011010100110011101000000111000110000000110101111100110
```

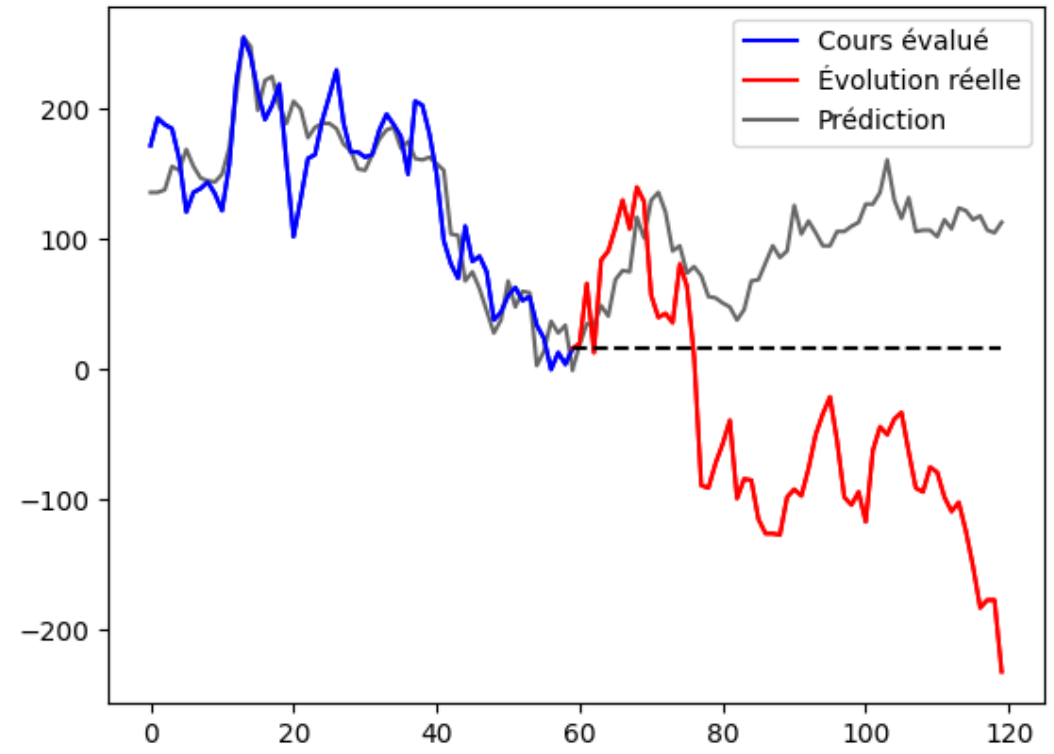
Comparaison des listes : **Distance de Hamming**

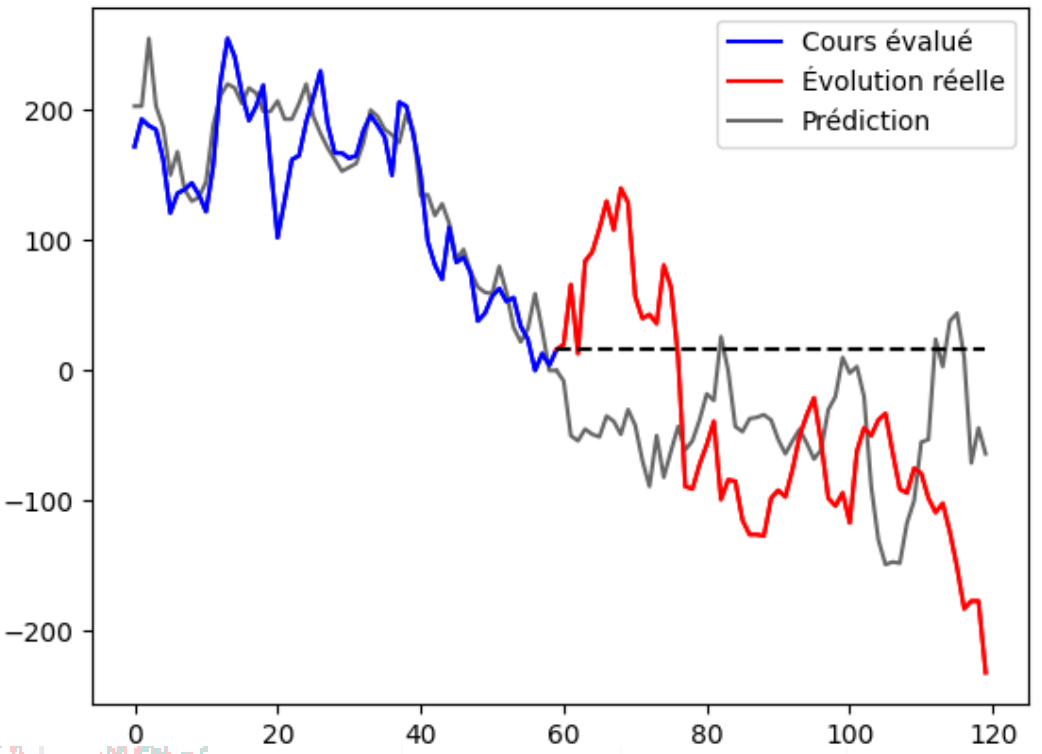
$$d(000111, 110101) = 3$$

- Recherche des dix listes les plus proches au sens de la distance de Hamming dans la base de données
- Tendance moyenne sur l'heure suivante

Définition d'un « succès »

« La tendance prédite se réalise »

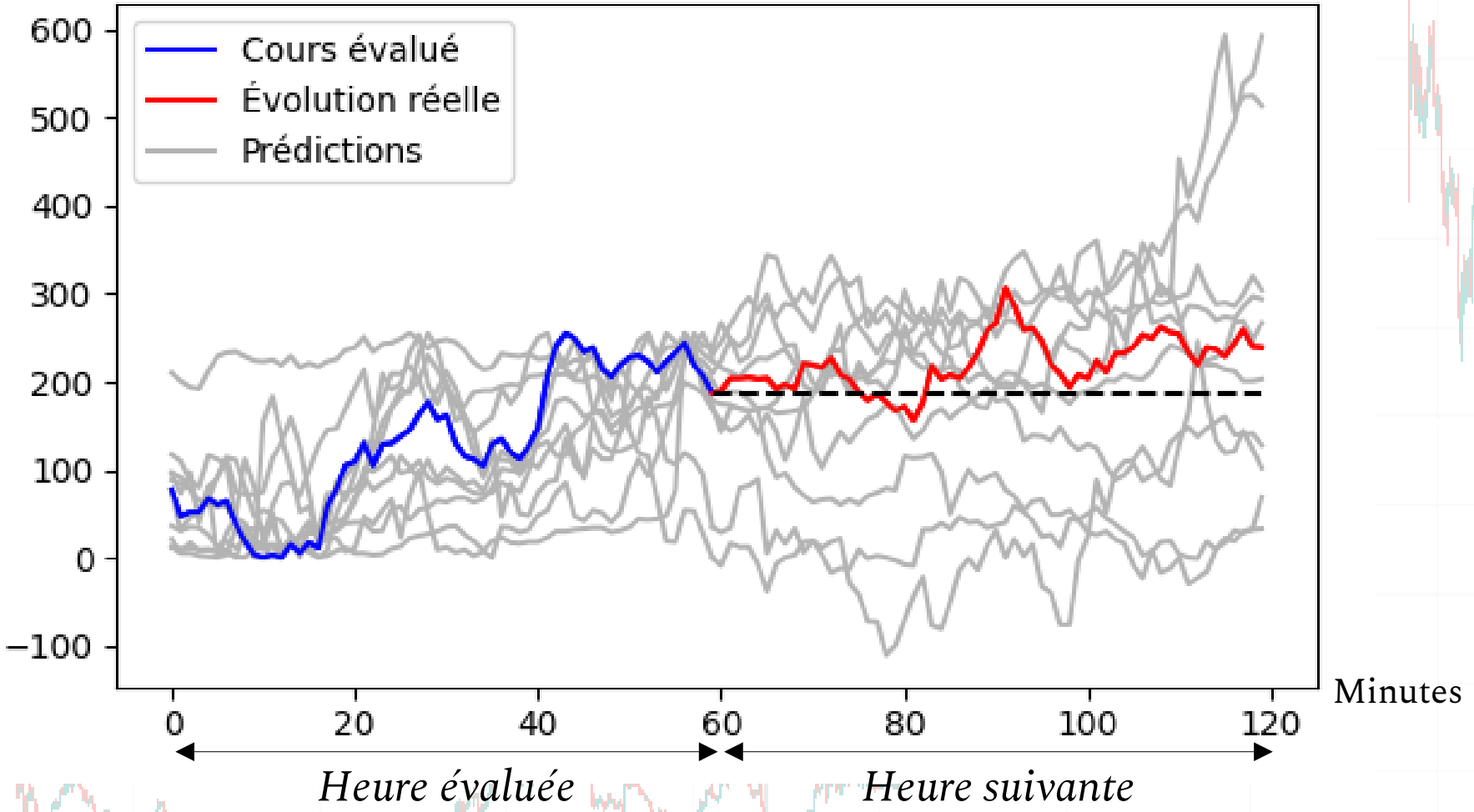




Échec

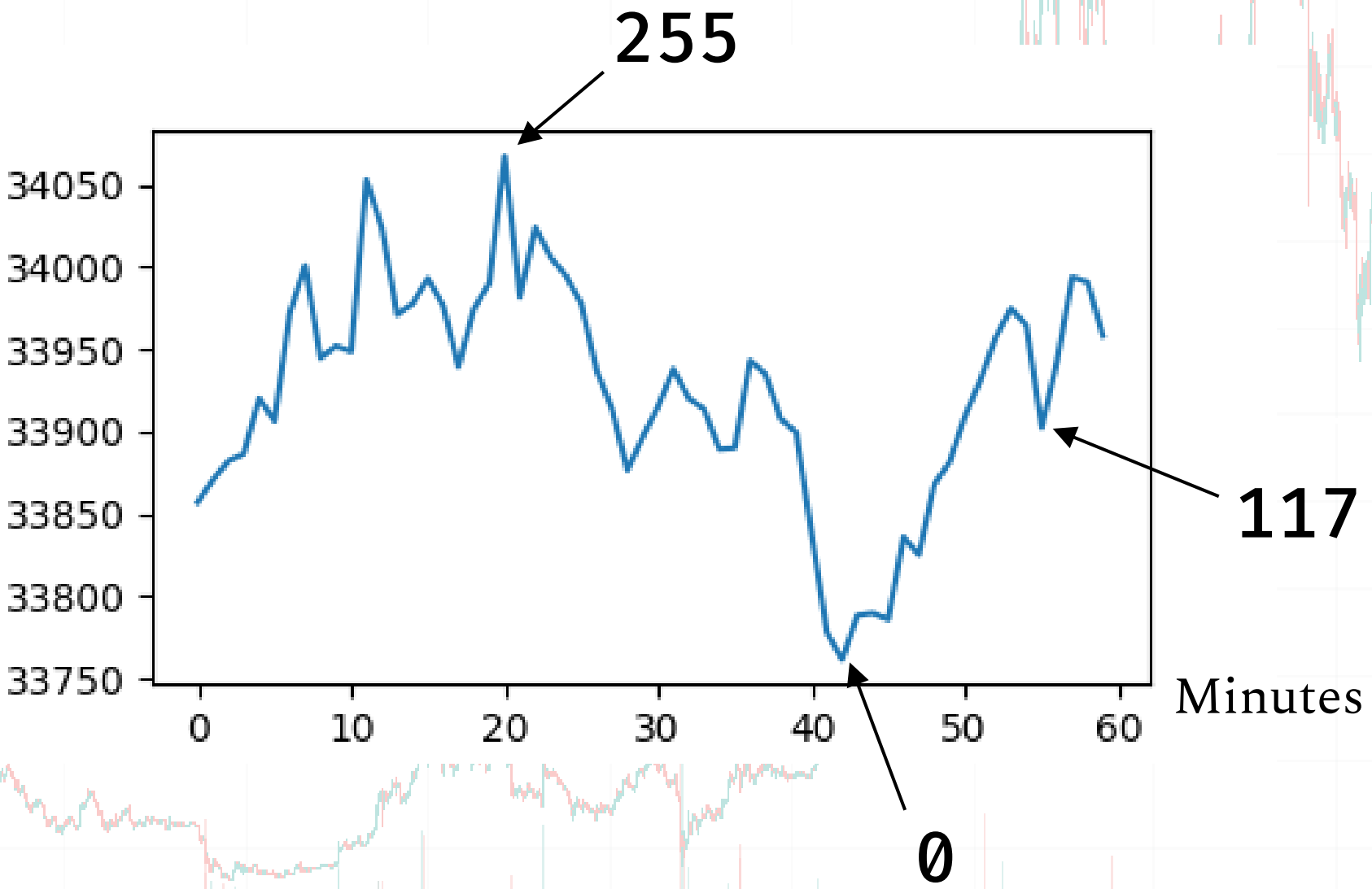
Succès

Résultats de la première modélisation

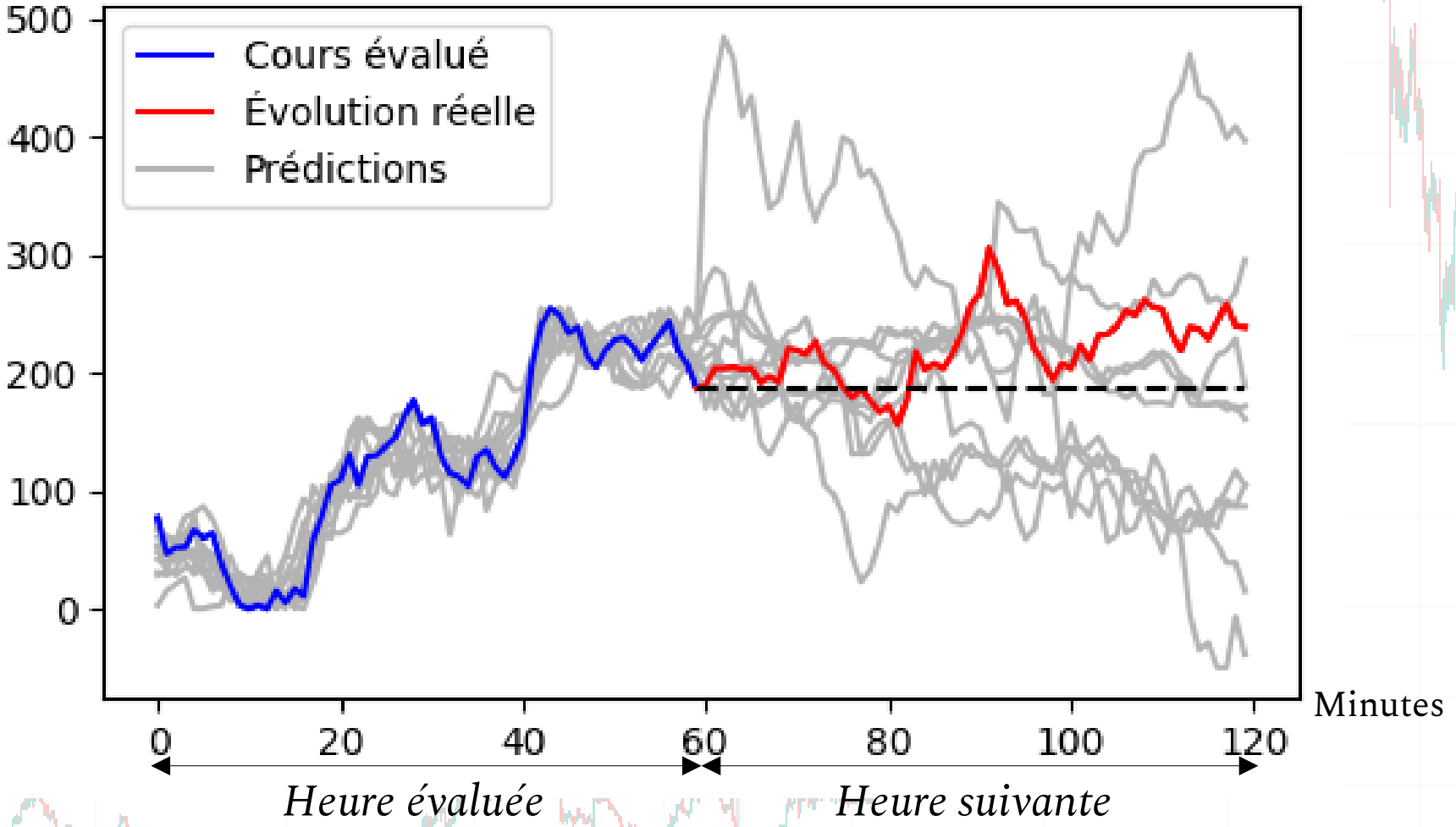


Taux de succès : **49,5 %**

Deuxième modélisation : **Codage sur 8 bits**



Résultats de la deuxième modélisation



Taux de succès : **51 %**

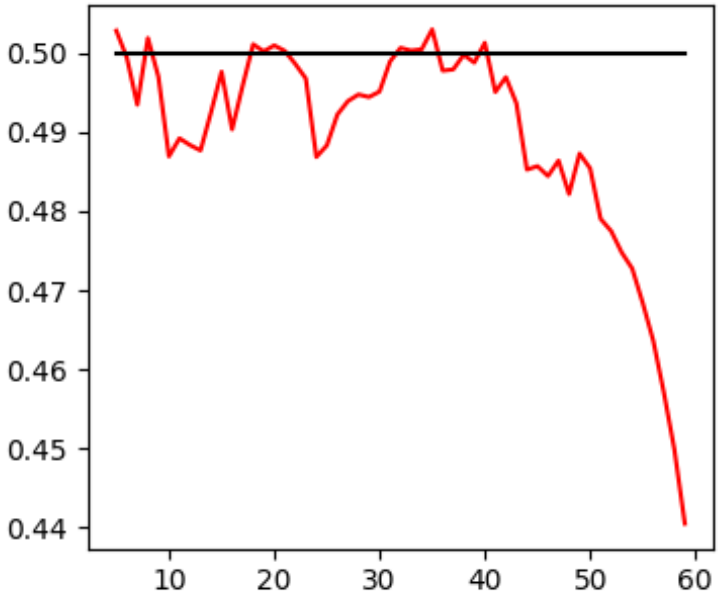
Un résultat intéressant

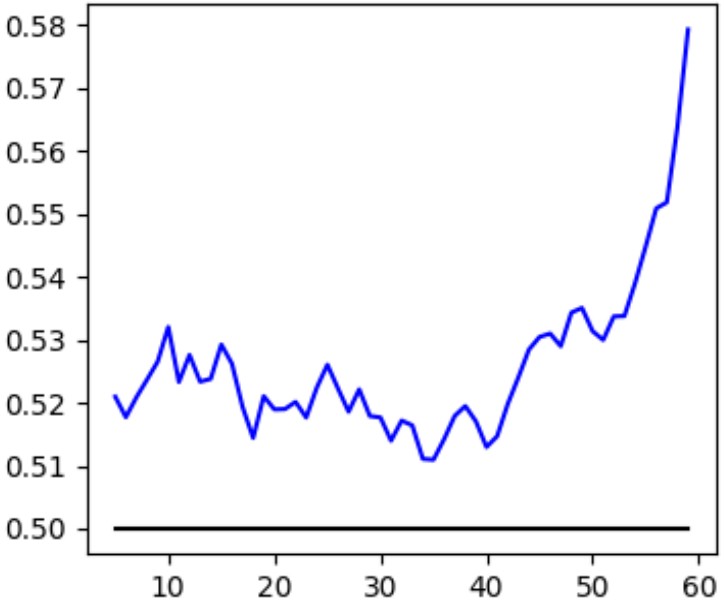
Tendance de la liste / Tendance de la liste suivante	Haussière	Baissière
Haussière	48,12 %	**53,72 %**
Baissière	**51,88 %**	46,28 %

Troisième modélisation

- Tendance (haussière ou baissière) de la liste
- Localisation du maximum et du minimum

Pour une **tendance haussière**

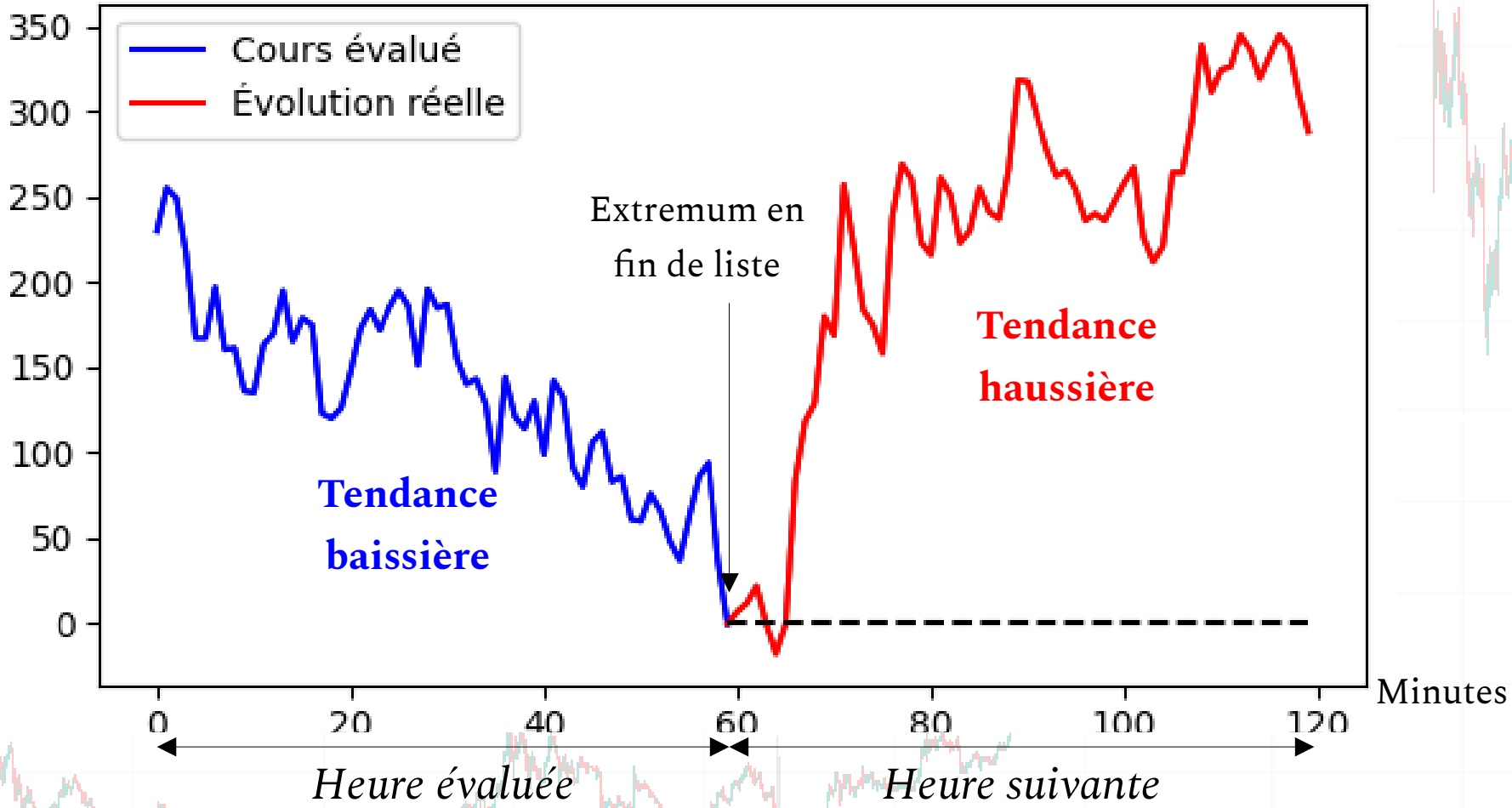




Pour une **tendance baissière**

Probabilité que la tendance suivante soit haussière en fonction de la position du maximum

Résultats de la troisième modélisation

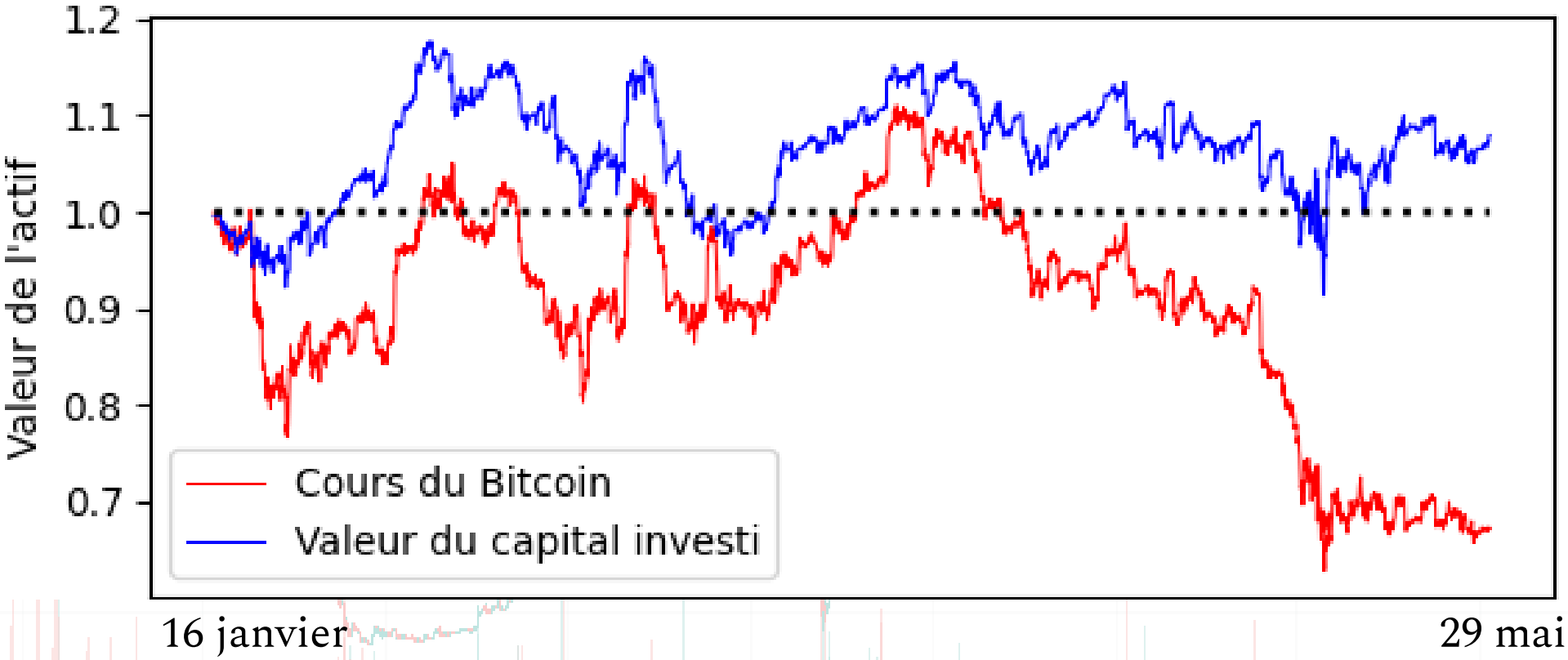


Taux de succès : **53 %**

Implémentation dans le cas réel

- Le Bitcoin s'est **effondré de 33 %** du 16 janvier au 29 mai
- Capital **accru de 8 % en 4 mois**

  → retour sur investissement moyen de **26 % par an**




Merci de votre écoute

Annexes

19 Importation des données

20 Établissement des listes

21 Établissement des listes (première modélisation)

22 Établissement des listes (deuxième modélisation)

23 Comparaison des listes (première modélisation)

24 Comparaison des listes (deuxième modélisation)

25 Établissement de la loi (troisième modélisation)

26 Comparaison des listes (troisième modélisation)

27 Calcul des gains et de la rentabilité

Importation des données

```
def fichier(m):
    btc=open(text_btc, 'w')
    unix=1642374000000
    i=0
    y=[]
    while unix<1653827000000:
        print(unix)
        l = exchange.fetch_ohlcv(m, since=unix, limit=1000)
        for n in l:
            btc.write(str(n[0])+' '+str(n[1])+' '+str(n[2])+'
'+str(n[3])+' '+str(n[4])+' '+str(n[5])+'\n')
        unix+=60000000
        y.append(n[1])
    btc.close()
    x=list(range(len(y)))
    figure()
    plot(x,y)
    show()
```

Établissement des listes

```
def etablissement_liste_heures():
    lh=open(text_liste_heures_comparaison,'w')
    valeurs=open(text_valeurs_comparaison,'r')
    v=valeurs.read().splitlines()
    c=len(v)
    for i in range(c-59):
        if i%10000==0: print(i)
        a=''
        for j in range(60):
            a+=v[i+j]
            a+=' '
        lh.write(a+'\n')
    valeurs.close()
    lh.close()
```

Établissement des listes (première modélisation)

```
def etablissement_liste_hamming():
    lham=open(text_liste_hamming,'w')
    for fichier in liste_fichiers_heures:
        print(0)
        f=open(fichier,'r')
        c=f.readlines()
        for ligne in c:
            s=''
            l=ligne.split(' ')
            for i in range(59):
                a,b=l[i],l[i+1]
                if a<b: s+='2'
                elif a>b: s+='0'
                else: s+='1'
            lham.write(s+'\n')
        f.close()
    lham.close()
```

Établissement des listes (deuxième modélisation)

```
def etablissement_liste_octet():
    lo=open(text_liste_octet_2,'w')
    f=open(text_liste_heures_comparaison,'r')
    c=f.readlines()
    for i in range(len(c)):
        ligne=c[i]
        if i%10000==0: print(i)
        l=ligne.split(' ')[:-1]
        m,s=[],''
        for x in l: m.append(float(x))
        a,b=min(m),max(m)
        for fl in m:
            if a==b: d=0
            else: d=int(255*(fl-a)/(b-a))
            s+=str(d)+' '
        lo.write(s+'\n')
    f.close()
    lo.close()
```

Comparaison des listes (première modélisation)

```
def hamming(a,b):
    d=0
    for i in range(len(a)):
        d+=abs(int(a[i])-int(b[i]))
    return d

def ordre(L): return L[1]

def comparaison_hamming(s):
    lh=open(text_liste_hamming_btc,'r')
    L=[]
    c=lh.read().splitlines()
    for i in range(len(c)-60):
        if i%100000==0:print(i//100000)
        l=c[i]
        L.append((l,hamming(s,l),i))
    M=sorted(L,key=ordre)
    N=[]
    for i in range(10):
        N.append(M[i][2])
    return N
```

Comparaison des listes (deuxième modélisation)

```
def distance_octet(l,m):
    L=l.split(' ')[:-1]
    M=m.split(' ')[:-1]
    d=0
    for i in range(len(L)):
        d+=abs(int(L[i])-int(M[i]))
    return d

def ordre(L): return L[1]

def comparaison_octet(s):
    lh=open(text_liste_octet_1,'r')
    L=[]
    c=lh.read().splitlines()
    for i in range(len(c)-60):
        if i%100000==0:print(i//100000)
        l=c[i]
        L.append((l,distance_octet(s,l),i))
    M=sorted(L,key=ordre)
    N=[]
    for i in range(10):
        N.append(M[i][2])
    return N
```

Établissement de la loi (troisième modélisation)

```
def stats_3_h():
    tc=open(text_liste_octet_3,'r')
    c=tc.read().splitlines()
    tc.close()
    C=[]
    for i in
range(60):C.append([0,0])
    for ligne in c:
        l=ligne.split(' ')
a1,a2,a3=int(l[0]),int(l[59]),int(l[
119])
        k=[]
        for i in range(60):
            k.append(int(l[i]))
        m=max(k)
        if a1<a2:
            pos=k.index(m)
            C[pos][0]+=1
            if a2<a3:
                C[pos][1]+=1
    return C
```

```
def stats_3_b():
    tc=open(text_liste_octet_3,'r')
    c=tc.read().splitlines()
    tc.close()
    C=[]
    for i in
range(60):C.append([0,0])
    for ligne in c:
        l=ligne.split(' ')
a1,a2,a3=int(l[0]),int(l[59]),int(l[
119])
        k=[]
        for i in range(60):
            k.append(int(l[i]))
        m=min(k)
        if a1>a2:
            pos=k.index(m)
            C[pos][0]+=1
            if a2<a3:
                C[pos][1]+=1
    return C
```

Comparaison des listes (troisième modélisation)

```
def comparaison_extremum():
    Ch=stats_3_h()
    Cb=stats_3_b()
    tc=open(text_liste_octet_4,'r')
    c=tc.read().splitlines()
    tc.close()
    d,v,f=0,0,0
    for ligne in c:
        d+=1
        l=ligne.split(' ')
        a1,a2,a3=int(l[0]),int(l[59]),int(l[119])
        if a1⩽a2:
            k=[]
            for i in range(60):
                k.append(int(l[i]))
            m=max(k)
            pos=k.index(m)
            b1=Ch[pos][1]>0.5*Ch[pos][0]
            b2=a2⩽a3
        else:
            k=[]
            for i in range(60):
                k.append(int(l[i]))
            m=min(k)
            pos=k.index(m)
            b1=Cb[pos][1]>0.5*Cb[pos][0]
            b2=a2<a3
        if b1==b2:
            v+=1
        else: f+=1
        if d%1000==0:
            print(d,v,f)
    print(d,v,f)
```

Calcul des gains et de la rentabilité

```
if d%60==0:
    lh1=h[d].split(' ')
    lh2=h[d+60].split(' ')
    l=ligne.split(' ')
    a1,a2,a3,a4,a5=int(l[0]),int(l[59]),int(l[119]),float(lh1[59]),float(lh2[59])
    if a1<=a2:
        k=[]
        for i in range(60):
            k.append(int(l[i]))
        m=max(k)
        pos=k.index(m)
        b1=Ch[pos][1]>0.5*Ch[pos][0]
        if b1:
            s*=a5/a4
        else:
            s*=1+(a4-a5)/a4
    else:
        k=[]
        for i in range(60):
            k.append(int(l[i]))
        m=min(k)
        pos=k.index(m)
        b1=Cb[pos][1]>0.5*Cb[pos][0]
        if b1:
            s*=a5/a4
        else:
            s*=1+(a4-a5)/a4
```