Stratégie avancée de paris sportifs

TIPE 2023-2024

THÈME : JEUX ET SPORTS

N° CANDIDAT : 16866

Contexte/Introduction aux paris sportifs

- Paris sportifs
- Côtes
- Bookmakers (plateformes de paris)
- Gains potentiels
- Type de paris

Côtes proposées par « Betlclic » pour la finale de coupe du monde 2018

V F : 2.50	NUL : 3.10	V C : 2.80

$$\frac{1}{2.5} + \frac{1}{3.10} + \frac{1}{2.80} = 107.9\%$$

7.9% de marge

Objectifs

- Prédire le score via la loi de poisson
- Stratégie de mise : Le critère de Kelly
- Simuler les performances de trois types de joueurs :
  - Le joueur à mise fixe
  - Le joueur matheux
  - Le joueur compulsif

Prédiction des issues par une loi de Poisson

- Détermination du paramètre $\lambda_{équipe}$:
- Simulation de 1000 matchs entre 20 équipes différentes
- Moyenne globale de buts marqués et encaissés
- Facteur d'attaque et de défense de chaque équipe.

Calcul du paramètre λ

Pour l'équipe A à domicile contre l'équipe B:

$$\lambda = Moyenne\ des\ buts\ \text{à}\ domicile * Facteur\ d'attaque\ de\ l'\text{équipe}\ A * Facteur\ de\ d\text{é}fense\ de\ l'\text{équipe}(B)$$

Équipe à domicile	Équipe à l'extérieur	Buts à domicile	Buts à l'extérieur
10	8	1	2
3	20	0	0
20	7	0	1
20	12	3	0
12	19	2	0
1	15	0	1
1	14	2	1
10	16	0	2
18	14	0	2
15	3	2	1
2	18	2	1
17	6	0	0
8	17	2	2
9	14	3	2
20	14	1	1

Extrait du tableau des scores parmi les 1000 matchs simulés

	EquipeDomicile	EquipeExterieur	LambdaDomicile	LambdaExterieur
280	15	1	1.1648535258826516	0.8038443011233773
275	14	16	1.4617019882227515	0.9631647932379207
279	14	20	1.4654403564790757	0.9668381906186909
41	3	2	1.9385826312209307	1.1018277523069417
35	2	16	1.5149461156288038	0.8561464828781516
378	20	3	1.6425538288143837	1.0615176533661292
58	3	19	1.8701000926180174	0.9085536188438053
276	14	17	1.5444398366127186	0.839142580536977
327	17	8	1.624507958819851	0.9949323337868204
177	9	18	1.86883784989853	1.0969951726379175

Paramètres lambdas obtenus parmi les 380 rencontres possibles

Probabilités des issues

$$\mathbb{P}(X = k) = \frac{e^{-\lambda}\lambda^{k}}{k!}$$

Probabilité du nombre de buts k

Buts	Proba_eq15	Proba_eq1
0	12.7%	38.5%
1	26.7%	36.2%
2	28.1%	17.0%
3	19.4%	5.3%
4	9.7%	1.3%
5	3.2%	0.3%
6	1.0%	0.0%

Probabilités du nombre de buts lors de la rencontre 15vs1

Stratégie de mise

- John Larry Kelly « *Une nouvelle interprétation du taux d'information* » 1956

- p : probabilité de gagner le pari
- q = 1 – p
- c : cote du bookmaker

La mise dépend de son signe

$$f^* = \frac{cp - q}{c} = p - \frac{q}{c}$$

Exemples de fraction de Kelly

Rencontres	V Domicile	Nul	V Ext	Prédiction	Fraction de Kelly
1 contre 8	2.21	3.82	1.38	V Dom	0.012
11 contre 7	1.91	2.92	2.53	V Ext	0.007
2 contre 9	2.19	2.53	1.72	V Ext	0.018
16 contre 14	1.67	2.87	3.65	V Dom	0.004
19 contre 12	3.27	1.68	1.56	V Ext	0.006
9 contre 16	2.53	3.56	1.76	V Dom	0.013
17 contre 20	2.75	1.29	1.43	V Ext	0.003
10 contre 6	1.74	3.54	3.56	V Ext	0.019
15 contre 5	2.27	1.78	2.28	V Dom	0.011

Quelques valeurs de la fraction de Kelly avec la méthode de prédiction

Analyse de nos performances

- Historique réel d'un joueur :
- 1521 paris
- profit de 0,76% (11,61 €)

Esperance de gain

- $\mathbb{E}(G) = \frac{cote\ proposée}{cote\ estimée} - 1$ (en pourcentage)

Match	Pari sélectionné	Cote proposée	Cote estimée	Esperance de gain
2 contre 9	9	1.75	1.61	8.69%
12 contre 7	7	2.0	1.95	2.56%
19 contre 12	12	5.2	5.1	1.96%
17 contre 8	17	4.3	4.16	3.36%
12 contre 3	12	7.0	6.48	8.02%
1 contre 7	1	1.65	1.58	4.43%

Exemples concrets d'espérance de gain

Attentes contre réalité

- Moyenne obtenue : 4.04%
- Esperance de bénéfice de 61.45€ pour 1521€ pariés
- Gain réel : 11.61€

Monte Carlo

- Méthodologie utilisée :
  - Déterminer la probabilité la plus favorable.
  - Cotes des bookmakers aléatoires (variable)
  - Issues aléatoires.
  - 100 000 itérations
  - Trois mises différentes.

Capital initial : 1521€

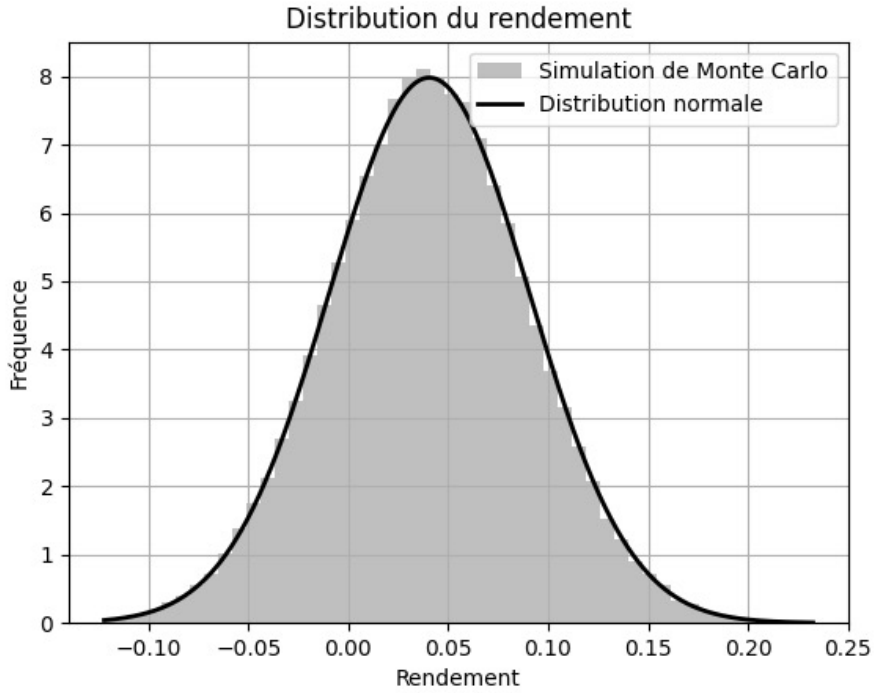
Résultats de la méthode de Kelly

Match	Pari sélectionné	Cote proposée	Cote estimée	Chiffre aléatoire	Montant misé	Gain
2 contre 9	9	1.75	1.61	0.46	0.35€	0,61 €
12 contre 7	7	2.0	1.95	0.15	0.48€	0.95€
19 contre 12	12	5.2	5.1	0.81	1,00 €	-1,00 €
17 contre 8	17	4.3	4.16	0.70	0.13€	0.58€
12 contre 3	12	7.0	6.48	0.52	1.6€	-1.6€
1 contre 7	1	1.65	1.58	0.53	0.98€	-0.98€

Gains et pertes obtenus avec la méthode de Kelly

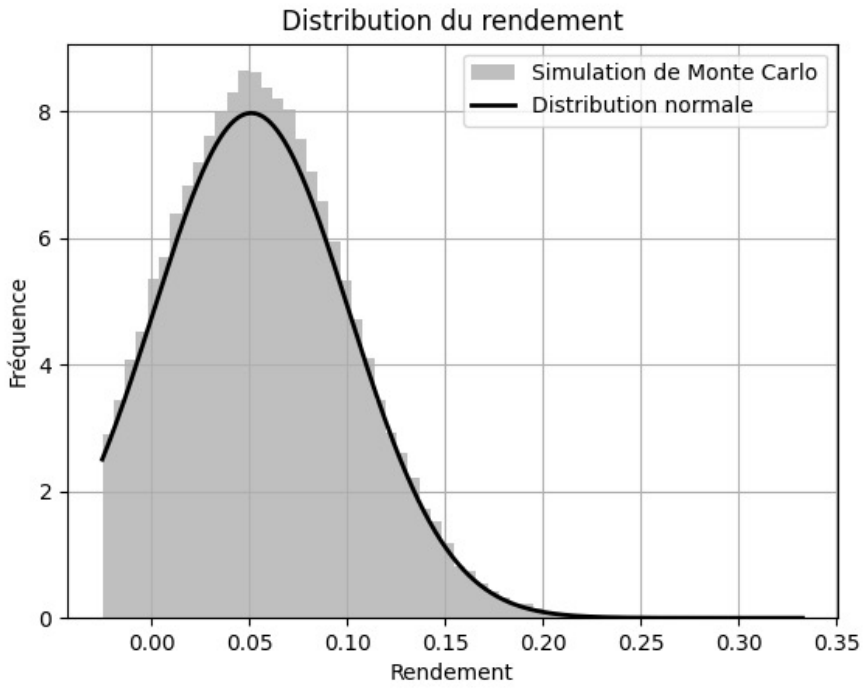
Résultats de la simulation à mise fixe

Gain moyen	Meilleure performance	Pire performance	Fréquence pertes	Fréquence gain> 0.76%
3.65%	23.26%	-12.23%	17.21%	78 %



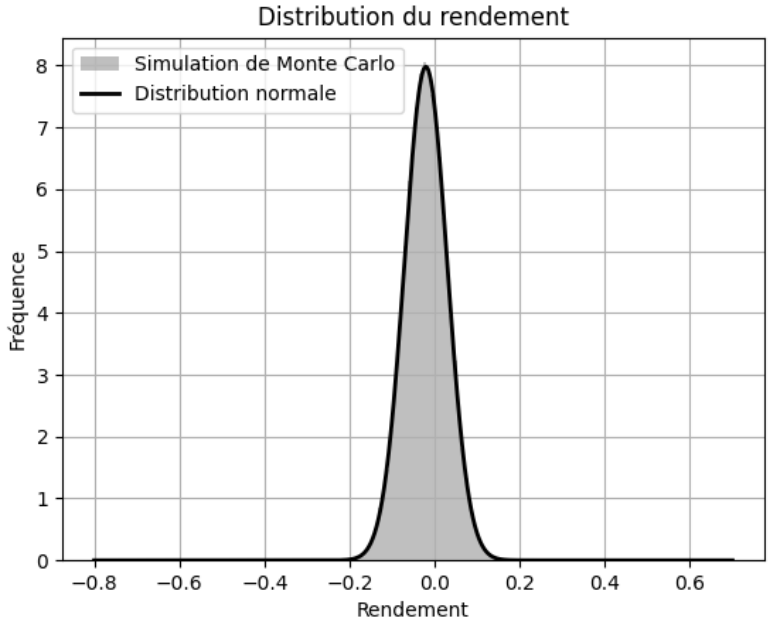
Résultats pour la méthode de Kelly

Gain moyen	Meilleure performance	Pire performance	Fréquence pertes	Fréquence gain> 0.76%
4.12%	33.21%	-2.51%	12.91%	79.98 %



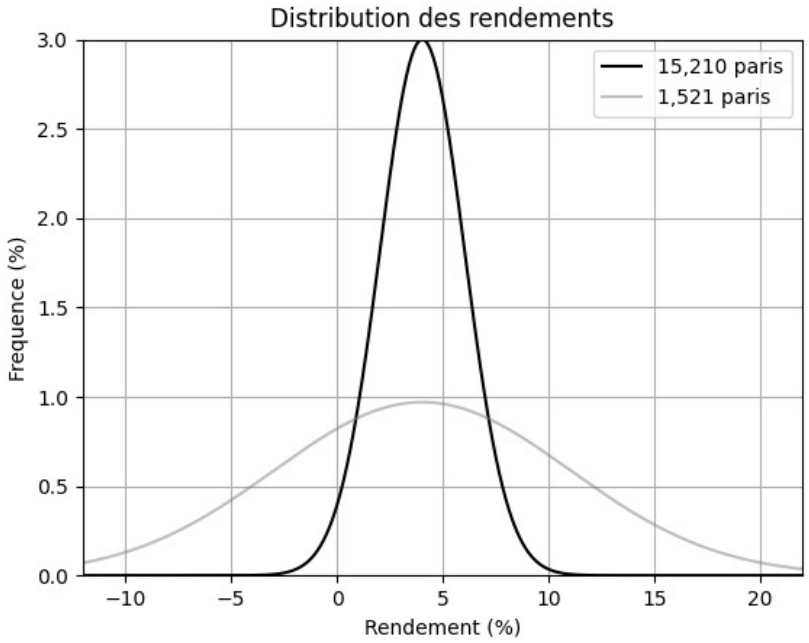
Résultats pour le parieur compulsif

Gain moyen	Meilleure performance	Pire performance	Fréquence pertes	Fréquence gain> 0.76%
-2.06%	71.9%	-81.2%	51.1%	41.12%



Système de pari faillible ou malchance ?

Nb paris	Gain moyen	Meilleure performance	Pire performance	Rendement de probabilité < 0%	Rendement de probabilité > 0.76%
15210	4.04%	10.17%	-1.21%	0.10%	99.3 %
1521	4.12%	33.21%	-2.51%	12.91%	79.98%



Comparaison des méthodes

- Avantages de la mise de Kelly:
  - Gains moyens plus élevés
  - Perte maximale moins importante
  - Gestion prudente des risques
  - Fréquence de gains élevée

Améliorations possibles

- Affiner les paramètres de la loi de Poisson
- Seuils de gain et de perte
- Diversifier les paris

Conclusion

Je ne suis toujours pas millionnaire mais j'ai appris des choses… Et ça ! Ça vaut tout l'or du Monde.

Création de la date base

datamatch.py > ...

```python
import pandas as pd
import numpy as np

# Nombre d'équipes et de matchs
nombre_equipes = 20
nombre_matchs = 1000

# Génération aléatoire des matchs
np.random.seed(42)  # Pour la reproductibilité
equipes_domicile = np.random.randint(1, nombre_equipes + 1, size=nombre_matchs)
equipes_exterieur = np.random.randint(1, nombre_equipes + 1, size=nombre_matchs)

# Éviter les matchs où une équipe joue contre elle-même
while any(equipes_domicile == equipes_exterieur):
    for i in range(nombre_matchs):
        if equipes_domicile[i] == equipes_exterieur[i]:
            equipes_exterieur[i] = np.random.randint(1, nombre_equipes + 1)

# Génération des scores simulés (simplifiée)
moyenne_buts_domicile = 1.5  # Moyenne des buts à domicile
moyenne_buts_exterieur = 1.0  # Moyenne des buts à l'extérieur

buts_domicile = np.random.poisson(moyenne_buts_domicile, nombre_matchs)
buts_exterieur = np.random.poisson(moyenne_buts_exterieur, nombre_matchs)

# Création de la DataFrame
matchs = pd.DataFrame({
    'EquipeDomicile': equipes_domicile,
    'EquipeExterieur': equipes_exterieur,
    'ButsDomicile': buts_domicile,
    'ButsExterieur': buts_exterieur
})
```

Calcul de λ 1

```python
# Calculer les moyennes globales
moyenne_buts_domicile_global = matchs['ButsDomicile'].mean()
moyenne_buts_exterieur_global = matchs['ButsExterieur'].mean()

# Calculer les facteurs d'attaque et de défense pour chaque équipe
facteur_attaque_domicile = matchs.groupby('EquipeDomicile')['ButsDomicile'].mean() / moyenne_buts_domicile_global
facteur_attaque_exterieur = matchs.groupby('EquipeExterieur')['ButsExterieur'].mean() / moyenne_buts_exterieur_global

facteur_defense_domicile = matchs.groupby('EquipeDomicile')['ButsExterieur'].mean() / moyenne_buts_exterieur_global
facteur_defense_exterieur = matchs.groupby('EquipeExterieur')['ButsDomicile'].mean() / moyenne_buts_domicile_global

# Créer des DataFrames pour les facteurs d'attaque et de défense
facteur_attaque = pd.DataFrame({
    'AttaqueDomicile': facteur_attaque_domicile,
    'AttaqueExterieur': facteur_attaque_exterieur
}).fillna(1)

facteur_defense = pd.DataFrame({
    'DefenseDomicile': facteur_defense_domicile,
    'DefenseExterieur': facteur_defense_exterieur
}).fillna(1)

print(facteur_attaque)
print(facteur_defense)

# Générer les 380 matchs possibles entre les 10 équipes
equipes_domicile = np.repeat(np.arange(1, nombre_equipes + 1), nombre_equipes)
equipes_exterieur = np.tile(np.arange(1, nombre_equipes + 1), nombre_equipes)

# Éviter les matchs où une équipe joue contre elle-même
mask = equipes_domicile != equipes_exterieur
equipes_domicile = equipes_domicile[mask]
equipes_exterieur = equipes_exterieur[mask]

```

Calcul de λ et de la probabilité du nombre de buts

```python
# Calculer les lambdas pour les matchs possibles
lambda_domicile = moyenne_buts_domicile * facteur_attaque.loc[equipes_domicile, 'AttaqueDomicile'].values * facteur_defense.loc[equipes_exterieur, 'DefenseExterieur'].values
lambda_exterieur = moyenne_buts_exterieur * facteur_attaque.loc[equipes_exterieur, 'AttaqueExterieur'].values * facteur_defense.loc[equipes_domicile, 'DefenseDomicile'].values

# Créer la DataFrame des matchs possibles avec les lambdas
matchs_possibles = pd.DataFrame({
    'EquipeDomicile': equipes_domicile,
    'EquipeExterieur': equipes_exterieur,
    'LambdaDomicile': lambda_domicile,
    'LambdaExterieur': lambda_exterieur
})

print(matchs_possibles.head())
```

```python
84  # Calcul des probabilités du nombre de buts (de 0 à 6) via la loi de Poisson
85  max_buts = 6
86  probabilites_domicile = [poisson.pmf(k, matchs_possibles['LambdaDomicile']) for k in range(max_buts + 1)]
87  probabilites_exterieur = [poisson.pmf(k, matchs_possibles['LambdaExterieur']) for k in range(max_buts + 1)]
88
89  # Ajouter les probabilités dans le DataFrame
90  for k in range(max_buts + 1):
91      matchs_possibles[f'Proba_Domicile_{k}_Buts'] = probabilites_domicile[k]
92      matchs_possibles[f'Proba_Exterieur_{k}_Buts'] = probabilites_exterieur[k]
93
94  # Afficher les premiers enregistrements
95  print(matchs_possibles.head())
96
```

Calcul de la fraction de Kelly et processus de sélection

```python
# Définir une fonction pour calculer la probabilité de victoire
def prob_victoire(prob_domicile, prob_exterieur):
    prob_victoire_domicile = np.sum(prob_domicile[np.tril_indices_from(prob_domicile, k=-1)])
    prob_victoire_exterieur = np.sum(prob_exterieur[np.triu_indices_from(prob_exterieur, k=1)])
    return prob_victoire_domicile, prob_victoire_exterieur

# Exemple de probabilité de buts pour un match spécifique
prob_domicile = np.array([matchs_possibles.iloc[0][f'Proba_Domicile_{k}_Buts'] for k in range(7)])
prob_exterieur = np.array([matchs_possibles.iloc[0][f'Proba_Exterieur_{k}_Buts'] for k in range(7)])

# Matrices des probabilités de score pour domicile et extérieur
prob_domicile_matrix = np.outer(prob_domicile, np.ones(7))
prob_exterieur_matrix = np.outer(np.ones(7), prob_exterieur)

# Calcul des probabilités de victoire
prob_victoire_domicile, prob_victoire_exterieur = prob_victoire(prob_domicile_matrix, prob_exterieur_matrix)
prob_egalite = 1 - (prob_victoire_domicile + prob_victoire_exterieur)

# Affichage des probabilités de victoire et d'égalité
print(f"Probabilité de victoire domicile : {prob_victoire_domicile:.4f}")
print(f"Probabilité de victoire extérieur : {prob_victoire_exterieur:.4f}")
print(f"Probabilité d'égalité : {prob_egalite:.4f}")

# Cotes offertes par le bookmaker (exemple fictif)
cote_domicile = 2.0
cote_exterieur = 3.5
cote_egalite = 3.0

# Calculer la fraction de Kelly
def calculer_fraction_kelly(cote, prob):
    q = 1 - prob
    f = (cote * prob - q) / cote
    return f

# Déterminer l'issue la plus probable et calculer la fraction de Kelly
if prob_victoire_domicile > prob_victoire_exterieur and prob_victoire_domicile > prob_egalite:
    fraction_kelly = calculer_fraction_kelly(cote_domicile, prob_victoire_domicile)
    print(f"Issue choisie : Victoire domicile")
elif prob_victoire_exterieur > prob_victoire_domicile and prob_victoire_exterieur > prob_egalite:
    fraction_kelly = calculer_fraction_kelly(cote_exterieur, prob_victoire_exterieur)
    print(f"Issue choisie : Victoire extérieur")
```

```
else:
    fraction_kelly = calculer_fraction_kelly(cote_egalite, prob_egalite)
    print(f"Issue choisie : Égalité")

print(f"Fraction de Kelly : {fraction_kelly:.4f}")
```

Monte Carlo

```python
# Paramètres de la simulation
nombre_simulations = 10000
nombre_paris = 500
capital_initial = 2000

# Générer des cotes fictives et des probabilités de victoire
np.random.seed(42)
cotes_domicile = np.random.uniform(1.5, 5.0, nombre_paris)
cotes_exterieur = np.random.uniform(1.5, 5.0, nombre_paris)
prob_victoire_domicile = np.random.uniform(0.1, 0.9, nombre_paris)
prob_victoire_exterieur = 1 - prob_victoire_domicile

# Fonction de pari avec méthode de Kelly
def parier_kelly(capital, cote, prob):
    q = 1 - prob
    f = (cote * prob - q) / cote
    f = max(f, 0)  # Assurer que f est non-négatif
    mise = capital * f
    gain = mise * (cote - 1) if np.random.rand() < prob else -mise
    return capital + gain
```

```python
# Simulation de Monte Carlo
def simulation_monte_carlo(nombre_simulations, nombre_paris, cotes_domicile, cotes_exterieur, prob_victoire_domicile, prob_victoire_exterieur):
    gains_kelly = np.zeros(nombre_simulations)
    gains_fixe_1 = np.zeros(nombre_simulations)
    gains_fixe_10 = np.zeros(nombre_simulations)

    for i in range(nombre_simulations):
        capital_kelly = capital_initial
        capital_fixe_1 = capital_initial
        capital_fixe_10 = capital_initial

        for j in range(nombre_paris):
            # Déterminer les cotes et probabilités pour le pari actuel
            cote_d = cotes_domicile[j]
            cote_e = cotes_exterieur[j]
            prob_d = prob_victoire_domicile[j]
            prob_e = prob_victoire_exterieur[j]

            # Pari méthode Kelly
            if prob_d > prob_e:
                capital_kelly = parier_kelly(capital_kelly, cote_d, prob_d)
            else:
                capital_kelly = parier_kelly(capital_kelly, cote_e, prob_e)

            # Pari mise fixe (1 unité)
            if prob_d > prob_e:
                gain = 1 * (cote_d - 1) if np.random.rand() < prob_d else -1
                capital_fixe_1 += gain
            else:
                gain = 1 * (cote_e - 1) if np.random.rand() < prob_e else -1
                capital_fixe_1 += gain

            # Pari mise agressive (10 unités)
            if prob_d > prob_e:
                gain = 10 * (cote_d - 1) if np.random.rand() < prob_d else -10
                capital_fixe_10 += gain
            else:
                gain = 10 * (cote_e - 1) if np.random.rand() < prob_e else -10
                capital_fixe_10 += gain
```

Tableau de performances

```python
# Effectuer la simulation
gains_kelly, gains_fixe_1, gains_fixe_10 = simulation_monte_carlo(nombre_simulations, nombre_paris, cotes_domicile, cotes_exterieur, prob_victoire_domicile, prob_victoire_exterieur)

# Calculer les statistiques
def calculer_statistiques(gains):
    gain_moyen = np.mean(gains)
    meilleure_performance = np.max(gains)
    pire_performance = np.min(gains)
    freq_perte = np.sum(gains < 0) / len(gains) * 100
    freq_gain = np.sum(gains > 0) / len(gains) * 100
    return gain_moyen, meilleure_performance, pire_performance, freq_perte, freq_gain

# Statistiques pour chaque stratégie
stats_kelly = calculer_statistiques(gains_kelly)
stats_fixe_1 = calculer_statistiques(gains_fixe_1)
stats_fixe_10 = calculer_statistiques(gains_fixe_10)

print("Statistiques Méthode Kelly:")
print(f"Gain moyen: {stats_kelly[0]:.2f} €")
print(f"Meilleure performance: {stats_kelly[1]:.2f} €")
print(f"Pire performance: {stats_kelly[2]:.2f} €")
print(f"Fréquence de perte: {stats_kelly[3]:.2f} %")
print(f"Fréquence de gain: {stats_kelly[4]:.2f} %")

print("\nStatistiques Mise Fixe (1 unité):")
print(f"Gain moyen: {stats_fixe_1[0]:.2f} €")
print(f"Meilleure performance: {stats_fixe_1[1]:.2f} €")
print(f"Pire performance: {stats_fixe_1[2]:.2f} €")
print(f"Fréquence de perte: {stats_fixe_1[3]:.2f} %")
print(f"Fréquence de gain: {stats_fixe_1[4]:.2f} %")

print("\nStatistiques Mise Fixe (10 unités):")
print(f"Gain moyen: {stats_fixe_10[0]:.2f} €")
print(f"Meilleure performance: {stats_fixe_10[1]:.2f} €")
print(f"Pire performance: {stats_fixe_10[2]:.2f} €")
print(f"Fréquence de perte: {stats_fixe_10[3]:.2f} %")
print(f"Fréquence de gain: {stats_fixe_10[4]:.2f} %")
```

Visualiser les distributions de gains

```
# Visualiser les distributions de gains
plt.figure(figsize=(14, 7))
plt.hist(gains_kelly, bins=50, alpha=0.5, label='Méthode Kelly')
plt.hist(gains_fixe_1, bins=50, alpha=0.5, label='Mise Fixe (1 unité)')
plt.hist(gains_fixe_10, bins=50, alpha=0.5, label='Mise Fixe (10 unités)')
plt.xlabel('Gains (€)')
plt.ylabel('Fréquence')
plt.title('Distribution des gains pour différentes stratégies de mise')
plt.legend()
plt.grid(True)
plt.show()
```