Projet Mario Kart ML Tutor : Pipeline d'apprentissage automatique pour la prédiction de combinaisons

Ce cahier Jupyter implémente un pipeline d'apprentissage automatique complet pour le projet **Mario Kart ML Tutor**. L'objectif est de prédire les combinaisons de pilote, kart, pneus et planeur qui sont considérées comme 'gagnantes' en se basant sur leurs statistiques en jeu.

Le pipeline suit les étapes standards d'un projet d'apprentissage automatique :

1. **Collecte des données** : Acquisition des ensembles de données des différents composants de Mario Kart.
2. **Analyse exploratoire de données (EDA)** : Comprendre la structure, les distributions et les relations au sein des données.
3. **Prétraitement des données** : Nettoyage, transformation et ingénierie des caractéristiques.
4. **Division des données** : Préparation des ensembles d'entraînement et de test.
5. **Création des modèles** : Instanciation et entraînement du modèle d'apprentissage automatique.
6. **Évaluation des modèles** : Mesure de la performance du modèle et interprétation des résultats.
7. **Choix du modèle** : Sélection du meilleur modèle (dans un contexte réel).
8. **Déploiement du modèle** : Sauvegarde du modèle pour des inférences futures.

1. Collecte des données

Cette section décrit les sources des données utilisées et le script Python pour les charger.

1.1 Dataset utilisé : Source du dataset

Les données proviennent de quatre fichiers CSV distincts, chacun contenant des statistiques détaillées pour un type de composant de Mario Kart 8 Deluxe :

- `drivers.csv` : Informations sur les pilotes (pilote, poids, accélération, etc.).
- `bodies_karts.csv` : Informations sur les carrosseries de karts.
- `tires.csv` : Informations sur les pneus.
- `gliders.csv` : Informations sur les planeurs.

Ces fichiers simulent un ensemble de données collecté à partir de sources fiables documentant les statistiques officielles du jeu. Le délimiteur de colonne utilisé dans ces fichiers CSV est le point-virgule (`;`).

1.2 Dataset collecté : Script Python utilisé

Le script ci-dessous charge les fichiers CSV en utilisant la bibliothèque `pandas` . Il inclut également les importations nécessaires pour toutes les étapes du pipeline et un `RANDOM_STATE` pour assurer la reproductibilité des résultats.

In [1]:

```
# Importations nécessaires pour l'ensemble du pipeline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc
import pickle
import os

# Définir un état aléatoire pour la reproductibilité
RANDOM_STATE = 42
np.random.seed(RANDOM_STATE)

# Définir le répertoire où les données brutes sont stockées
DATA_DIR = './data/raw' # Assurez-vous que vos fichiers CSV sont dans ce répertoire

# Charger chaque jeu de données
try:
    drivers = pd.read_csv(f'{DATA_DIR}/drivers.csv', sep=';')
    bodies  = pd.read_csv(f'{DATA_DIR}/bodies_karts.csv', sep=';')
    tires   = pd.read_csv(f'{DATA_DIR}/tires.csv', sep=';')
    gliders = pd.read_csv(f'{DATA_DIR}/gliders.csv', sep=';')

    print('DataFrames chargés avec succès :')
    print(f'  Taille des pilotes : {drivers.shape}')
    print(f'  Taille des carrosseries : {bodies.shape}')
    print(f'  Taille des pneus : {tires.shape}')
    print(f'  Taille des planeurs : {gliders.shape}')
except FileNotFoundError:
    print(f"Erreur : Assurez-vous que vos fichiers de données sont dans le répertoire '{
    print("Veuillez créer le dossier 'data' et le sous-dossier 'raw', puis y placer les
    # Créer des DataFrames vides pour éviter des erreurs plus tard dans le notebook
    drivers = pd.DataFrame(columns=['Driver', 'Weight', 'Acceleration', 'On-Road Tractio
    bodies = pd.DataFrame(columns=['Body', 'Weight', 'Acceleration', 'On-Road Traction',
    tires = pd.DataFrame(columns=['Tire', 'Weight', 'Acceleration', 'On-Road Traction',
    gliders = pd.DataFrame(columns=['Glider', 'Weight', 'Acceleration', 'On-Road Tractio
```

```
DataFrames chargés avec succès :
  Taille des pilotes : (43, 14)
  Taille des carrosseries : (40, 14)
  Taille des pneus : (21, 14)
  Taille des planeurs : (14, 14)
```

2. Analyse exploratoire de données (EDA - Exploratory Data Analysis)

L'EDA est une étape fondamentale pour comprendre la structure des données, identifier les distributions, détecter les relations et préparer les données pour le prétraitement et la modélisation.

2.1 Inspection initiale des données

Nous commençons par examiner les premières lignes de chaque DataFrame et leurs informations de base (`.info()`) pour avoir un aperçu rapide des données brutes, y compris les types de données et les valeurs non nulles.

In [2]:

```
print("\n--- Informations sur le DataFrame 'drivers' ---")
drivers.info()
print("\n--- Premières lignes du DataFrame 'drivers' ---")
print(drivers.head())

print("\n--- Informations sur le DataFrame 'bodies' ---")
bodies.info()
print("\n--- Premières lignes du DataFrame 'bodies' ---")
print(bodies.head())

print("\n--- Informations sur le DataFrame 'tires' ---")
tires.info()
print("\n--- Premières lignes du DataFrame 'tires' ---")
print(tires.head())

print("\n--- Informations sur le DataFrame 'gliders' ---")
gliders.info()
print("\n--- Premières lignes du DataFrame 'gliders' ---")
print(gliders.head())
```

```
--- Informations sur le DataFrame 'drivers' ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43 entries, 0 to 42
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   Driver                 43 non-null     object
 1   Weight                 43 non-null     int64 
 2   Acceleration           43 non-null     int64 
 3   On-Road traction       43 non-null     int64 
 4   Off-Road Traction      43 non-null     int64 
 5   Mini-Turbo             43 non-null     int64 
 6   Ground Speed           43 non-null     int64 
 7   Water Speed            43 non-null     int64 
 8   Anti-Gravity Speed     43 non-null     int64 
 9   Air Speed              43 non-null     int64 
 10  Ground Handling        43 non-null     int64 
 11  Water Handling         43 non-null     int64 
 12  Anti-Gravity Handling  43 non-null     int64 
 13  Air Handling           43 non-null     int64 
dtypes: int64(13), object(1)
memory usage: 4.8+ KB

--- Premières lignes du DataFrame 'drivers' ---
  Driver  Weight  Acceleration  On-Road traction  Off-Road Traction  \
0  Mario       6             2                 4                  2   
1  Luigi       6             2                 5                  1   
2  Peach       4             3                 3                  3   
3  Daisy       4             3                 3                  3   
4  Yoshi       4             3                 3                  3   

   Mini-Turbo  Ground Speed  Water Speed  Anti-Gravity Speed  Air Speed  \
0           2             6            6                   6          6   
```

```
1           2            6           6                   6          6
2           3            5           5                   5          5
3           3            5           5                   5          5
4           3            5           5                   5          5

   Ground Handling  Water Handling  Anti-Gravity Handling  Air Handling
0                4               4                      4             4
1                5               5                      5             5
2                5               5                      5             5
3                5               5                      5             5
4                5               5                      5             5

--- Informations sur le DataFrame 'bodies' ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40 entries, 0 to 39
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Body                   40 non-null     object
 1   Weight                 40 non-null     int64
 2   Acceleration           40 non-null     int64
 3   On-Road traction       40 non-null     int64
 4   Off-Road Traction      40 non-null     int64
 5   Mini-Turbo             40 non-null     int64
 6   Ground Speed           40 non-null     int64
 7   Water Speed            40 non-null     int64
 8   Anti-Gravity Speed     40 non-null     int64
 9   Air Speed              40 non-null     int64
 10  Ground Handling        40 non-null     int64
 11  Water Handling         40 non-null     int64
 12  Anti-Gravity Handling  40 non-null     int64
 13  Air Handling           40 non-null     int64
dtypes: int64(13), object(1)
memory usage: 4.5+ KB

--- Premières lignes du DataFrame 'bodies' ---
            Body  Weight  Acceleration  On-Road traction  Off-Road Traction  \
0  Standard Kart       2             4                 3                  3
1     Pipe Frame       1             6                 3                  4
2         Mach 8       3             3                 2                  4
3   Steel Driver       4             1                 1                  3
4    Cat Cruiser       2             5                 4                  3

   Mini-Turbo  Ground Speed  Water Speed  Anti-Gravity Speed  Air Speed  \
0           4             3            3                   3          3
1           6             1            3                   1          1
2           4             3            3                   5          4
3           2             4            5                   2          0
4           5             2            2                   3          4

   Ground Handling  Water Handling  Anti-Gravity Handling  Air Handling
0                3               2                      3             3
1                5               4                      4             2
2                2               2                      4             2
3                1               5                      1             1
4                4               2                      3             4

--- Informations sur le DataFrame 'tires' ---
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 21 entries, 0 to 20
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Tire                   21 non-null     object
 1   Weight                 21 non-null     int64
 2   Acceleration           21 non-null     int64
 3   On-Road traction       21 non-null     int64
 4   Off-Road Traction      21 non-null     int64
 5   Mini-Turbo             21 non-null     int64
 6   Ground Speed           21 non-null     int64
 7   Water Speed            21 non-null     int64
 8   Anti-Gravity Speed     21 non-null     int64
 9   Air Speed              21 non-null     int64
 10  Ground Handling        21 non-null     int64
 11  Water Handling         21 non-null     int64
 12  Anti-Gravity Handling  21 non-null     int64
 13  Air Handling           21 non-null     int64
dtypes: int64(13), object(1)
memory usage: 2.4+ KB

--- Premières lignes du DataFrame 'tires' ---
       Tire  Weight  Acceleration  On-Road traction  Off-Road Traction  \
0  Standard       2             4                 2                  5
1   Monster       4             2                 3                  7
2    Roller       0             6                 0                  4
3      Slim       2             2                 4                  1
4     Slick       3             1                 4                  0

   Mini-Turbo  Ground Speed  Water Speed  Anti-Gravity Speed  Air Speed  \
0           3             2            3                   2          3
1           2             2            2                   2          1
2           6             0            3                   0          3
3           2             3            2                   4          2
4           0             4            0                   4          0

   Ground Handling  Water Handling  Anti-Gravity Handling  Air Handling
0                3               3                      3             3
1                0               1                      0             1
2                4               4                      4             4
3                4               4                      3             4
4                2               0                      2             1

--- Informations sur le DataFrame 'gliders' ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14 entries, 0 to 13
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Glider                 14 non-null     object
 1   Weight                 14 non-null     int64
 2   Acceleration           14 non-null     int64
 3   On-Road traction       14 non-null     int64
 4   Off-Road Traction      14 non-null     int64
 5   Mini-Turbo             14 non-null     int64
 6   Ground Speed           14 non-null     int64
 7   Water Speed            14 non-null     int64
 8   Anti-Gravity Speed     14 non-null     int64
 9   Air Speed              14 non-null     int64
```

```
 10  Ground Handling        14 non-null     int64
 11  Water Handling         14 non-null     int64
 12  Anti-Gravity Handling  14 non-null     int64
 13  Air Handling           14 non-null     int64
dtypes: int64(13), object(1)
memory usage: 1.7+ KB

--- Premières lignes du DataFrame 'gliders' ---
          Glider  Weight  Acceleration  On-Road traction  Off-Road Traction  \
0   Super Glider       1             1                 1                  1
1   Cloud Glider       0             2                 1                  1
2    Wario Wing        2             1                 2                  0
3    Waddle Wing       1             1                 1                  1
4  Peach Parasol       1             2                 2                  0

   Mini-Turbo  Ground Speed  Water Speed  Anti-Gravity Speed  Air Speed  \
0           1             1            1                   0          2
1           2             0            1                   1          1
2           1             1            0                   1          2
3           1             1            1                   0          2
4           2             0            0                   1          1

   Ground Handling  Water Handling  Anti-Gravity Handling  Air Handling
0                1               0                      1             1
1                1               0                      1             2
2                1               1                      0             1
3                1               0                      1             1
4                1               1                      0             2
```

2.2 Fusion des données et échantillonnage des combinaisons

Pour analyser toutes les combinaisons possibles de Mario Kart, nous devons fusionner ces quatre DataFrames. Chaque composant pouvant être combiné avec n'importe quel autre, nous effectuerons une **jointure croisée** (produit cartésien) pour créer toutes les combinaisons uniques.

En raison du grand nombre de combinaisons potentielles, nous échantillonnerons un sous-ensemble (50 000 combinaisons) pour gérer les ressources computationnelles tout en conservant un échantillon représentatif de l'espace de combinaison complet.

Nous définissons d'abord une fonction auxiliaire `prefix_cols` pour renommer les colonnes. Cela permet d'éviter les conflits de noms et de clarifier l'origine de chaque statistique (par exemple, `driver_weight` pour le poids du pilote).

In [3]:

```python
def prefix_cols(df, name_col, prefix):
    """
    Renomme les colonnes d'un DataFrame en ajoutant un préfixe spécifié,
    à l'exception de la colonne de nom elle-même.
    Convertit les noms de colonnes en minuscules et remplace les espaces par des tirets
    """
    df = df.copy()
    # Renommer la colonne d'identifiant principal
    df.rename(columns={name_col: prefix}, inplace=True)
    # Préfixer les autres colonnes avec le type de composant (ex: 'driver_weight')
    for col in df.columns:
        if col != prefix:
```

```
            # Convertir en minuscules et remplacer les espaces par des tirets bas
            df.rename(columns={col: f"{prefix}_{col.lower().replace(' ','_')}"}, inplace
    return df

# Appliquer le préfixage à chaque DataFrame
drivers_pref = prefix_cols(drivers, 'Driver', 'driver')
bodies_pref  = prefix_cols(bodies, 'Body', 'body')
tires_pref   = prefix_cols(tires, 'Tire', 'tire')
gliders_pref = prefix_cols(gliders, 'Glider', 'glider')

# Ajouter une colonne 'key' à chaque DataFrame pour la jointure croisée
for df in [drivers_pref, bodies_pref, tires_pref, gliders_pref]:
    df['key'] = 1

# Effectuer la jointure croisée pour obtenir toutes les combinaisons
combo = (drivers_pref
         .merge(bodies_pref, on='key')
         .merge(tires_pref,   on='key')
         .merge(gliders_pref, on='key')
         .drop('key', axis=1)) # Supprimer la colonne 'key' après la fusion

print(f"Nombre total de combinaisons possibles : {combo.shape[0]:,}")

# Échantillonner un sous-ensemble de combinaisons pour l'analyse
N_SAMPLES = 50000
if combo.shape[0] > N_SAMPLES:
    combo_sampled = combo.sample(n=N_SAMPLES, random_state=RANDOM_STATE).reset_index(dro
    print(f"Échantillonnage de {N_SAMPLES:,} combinaisons. Nouvelle taille : {combo_samp
else:
    combo_sampled = combo.reset_index(drop=True)
    print(f"Le nombre de combinaisons est inférieur ou égal à {N_SAMPLES:,}. Aucun échan

# Créer une copie brute non transformée pour la démo de prédiction future
# Ceci sera utilisé pour simuler de nouvelles données d'entrée avant la mise à l'échelle
combo_raw_untransformed = combo_sampled.copy()

# Afficher les premières lignes du DataFrame combiné (échantillonné)
print("\n--- Premières lignes des données combinées (échantillonnées) ---")
print(combo_sampled.head())
```

```
Nombre total de combinaisons possibles : 505,680
Échantillonnage de 50,000 combinaisons. Nouvelle taille : (50000, 56)

--- Premières lignes des données combinées (échantillonnées) ---
         driver  driver_weight  driver_acceleration  driver_on-road_traction  \
0  Mii (medium)              6                    2                        4
1     Dry Bones              1                    5                        2
2        Ludwig              6                    2                        4
3    Baby Peach              0                    4                        3
4           Roy              8                    1                       10

   driver_off-road_traction  driver_mini-turbo  driver_ground_speed  \
0                         2                  2                    6
1                         4                  4                    1
2                         2                  2                    6
3                         5                  5                    0
4                         0                  1                    9

   driver_water_speed  driver_anti-gravity_speed  driver_air_speed  ...  \
```

```
0                    6                          6                  6  ...
1                    1                          1                  1  ...
2                    6                          6                  6  ...
3                    0                          0                  0  ...
4                    9                          9                  9  ...

   glider_off-road_traction  glider_mini-turbo  glider_ground_speed  \
0                         1                  2                    0
1                         0                  1                    1
2                         0                  2                    0
3                         1                  2                    0
4                         1                  1                    1

   glider_water_speed glider_anti-gravity_speed  glider_air_speed  \
0                   1                         1                 1
1                   0                         1                 2
2                   0                         1                 1
3                   1                         1                 1
4                   1                         0                 2

   glider_ground_handling  glider_water_handling  \
0                       1                      0
1                       1                      1
2                       1                      1
3                       1                      0
4                       1                      0

   glider_anti-gravity_handling  glider_air_handling
0                             1                    2
1                             0                    1
2                             0                    2
3                             1                    2
4                             1                    1

[5 rows x 56 columns]
```

2.3 Analyse des types de données

Vérifier les types de données des colonnes dans le DataFrame `combo_sampled` est essentiel pour s'assurer qu'elles sont appropriées pour les opérations numériques et l'analyse. Cela permet de détecter des problèmes comme des colonnes numériques chargées comme des objets (chaînes de caractères).

In [4]:

```
print("\n--- Informations sur le DataFrame combiné (échantillonné) ---")
combo_sampled.info()
```

```
--- Informations sur le DataFrame combiné (échantillonné) ---
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 56 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   driver                        50000 non-null  object
 1   driver_weight                 50000 non-null  int64
 2   driver_acceleration           50000 non-null  int64
 3   driver_on-road_traction       50000 non-null  int64
 4   driver_off-road_traction      50000 non-null  int64
 5   driver_mini-turbo             50000 non-null  int64
```

```
 6   driver_ground_speed           50000 non-null  int64
 7   driver_water_speed            50000 non-null  int64
 8   driver_anti-gravity_speed     50000 non-null  int64
 9   driver_air_speed              50000 non-null  int64
 10  driver_ground_handling        50000 non-null  int64
 11  driver_water_handling         50000 non-null  int64
 12  driver_anti-gravity_handling  50000 non-null  int64
 13  driver_air_handling           50000 non-null  int64
 14  body                          50000 non-null  object
 15  body_weight                   50000 non-null  int64
 16  body_acceleration             50000 non-null  int64
 17  body_on-road_traction         50000 non-null  int64
 18  body_off-road_traction        50000 non-null  int64
 19  body_mini-turbo               50000 non-null  int64
 20  body_ground_speed             50000 non-null  int64
 21  body_water_speed              50000 non-null  int64
 22  body_anti-gravity_speed       50000 non-null  int64
 23  body_air_speed                50000 non-null  int64
 24  body_ground_handling          50000 non-null  int64
 25  body_water_handling           50000 non-null  int64
 26  body_anti-gravity_handling    50000 non-null  int64
 27  body_air_handling             50000 non-null  int64
 28  tire                          50000 non-null  object
 29  tire_weight                   50000 non-null  int64
 30  tire_acceleration             50000 non-null  int64
 31  tire_on-road_traction         50000 non-null  int64
 32  tire_off-road_traction        50000 non-null  int64
 33  tire_mini-turbo               50000 non-null  int64
 34  tire_ground_speed             50000 non-null  int64
 35  tire_water_speed              50000 non-null  int64
 36  tire_anti-gravity_speed       50000 non-null  int64
 37  tire_air_speed                50000 non-null  int64
 38  tire_ground_handling          50000 non-null  int64
 39  tire_water_handling           50000 non-null  int64
 40  tire_anti-gravity_handling    50000 non-null  int64
 41  tire_air_handling             50000 non-null  int64
 42  glider                        50000 non-null  object
 43  glider_weight                 50000 non-null  int64
 44  glider_acceleration           50000 non-null  int64
 45  glider_on-road_traction       50000 non-null  int64
 46  glider_off-road_traction      50000 non-null  int64
 47  glider_mini-turbo             50000 non-null  int64
 48  glider_ground_speed           50000 non-null  int64
 49  glider_water_speed            50000 non-null  int64
 50  glider_anti-gravity_speed     50000 non-null  int64
 51  glider_air_speed              50000 non-null  int64
 52  glider_ground_handling        50000 non-null  int64
 53  glider_water_handling         50000 non-null  int64
 54  glider_anti-gravity_handling  50000 non-null  int64
 55  glider_air_handling           50000 non-null  int64
dtypes: int64(52), object(4)
memory usage: 21.4+ MB
```

2.4 Statistiques descriptives

Générer des statistiques descriptives pour les colonnes numériques permet de comprendre leur tendance centrale, leur dispersion et leur étendue. Cela aide à identifier les valeurs aberrantes potentielles ou les

distributions de données inhabituelles.

In [5]:

```
print("\n--- Statistiques descriptives des caractéristiques numériques ---")
print(combo_sampled.describe().T)
```

```
--- Statistiques descriptives des caractéristiques numériques ---
                              count     mean       std  min  25%  50%  75%  \
driver_weight                 50000.0  4.73570  3.156252  0.0  2.0  4.0  7.0
driver_acceleration           50000.0  2.80696  1.661900  0.0  1.0  3.0  4.0
driver_on-road_traction       50000.0  4.77644  2.660249  1.0  3.0  4.0  7.0
driver_off-road_traction      50000.0  2.42050  1.542857  0.0  1.0  3.0  4.0
driver_mini-turbo             50000.0  2.64598  1.428457  0.0  2.0  3.0  4.0
driver_ground_speed           50000.0  4.84648  3.080477  0.0  2.0  5.0  7.0
driver_water_speed            50000.0  4.84648  3.080477  0.0  2.0  5.0  7.0
driver_anti-gravity_speed     50000.0  4.84648  3.080477  0.0  2.0  5.0  7.0
driver_air_speed              50000.0  4.84648  3.080477  0.0  2.0  5.0  7.0
driver_ground_handling        50000.0  5.20136  2.628321  0.0  3.0  5.0  7.0
driver_water_handling         50000.0  5.20136  2.628321  0.0  3.0  5.0  7.0
driver_anti-gravity_handling  50000.0  5.20136  2.628321  0.0  3.0  5.0  7.0
driver_air_handling           50000.0  5.20136  2.628321  0.0  3.0  5.0  7.0
body_weight                   50000.0  1.94484  1.285391  0.0  1.0  2.0  3.0
body_acceleration             50000.0  3.32410  2.037121  0.0  2.0  3.0  5.0
body_on-road_traction         50000.0  2.19126  1.188119  0.0  1.0  2.0  3.0
body_off-road_traction        50000.0  3.44888  1.734660  0.0  3.0  3.0  5.0
body_mini-turbo               50000.0  3.67888  1.854570  0.0  3.0  4.0  5.0
body_ground_speed             50000.0  2.92366  1.436243  0.0  2.0  3.0  4.0
body_water_speed              50000.0  2.65242  1.154693  1.0  2.0  2.0  3.0
body_anti-gravity_speed       50000.0  2.95594  1.182703  0.0  2.0  3.0  4.0
body_air_speed                50000.0  2.42798  1.202648  0.0  1.0  3.0  3.0
body_ground_handling          50000.0  2.90226  1.474256  0.0  2.0  3.0  4.0
body_water_handling           50000.0  2.80914  1.246605  1.0  2.0  3.0  4.0
body_anti-gravity_handling    50000.0  2.95758  1.205691  1.0  2.0  3.0  4.0
body_air_handling             50000.0  2.22750  1.269746  0.0  1.0  2.0  3.0
tire_weight                   50000.0  2.14336  1.355435  0.0  1.0  2.0  3.0
tire_acceleration             50000.0  2.99374  1.720884  0.0  2.0  3.0  4.0
tire_on-road_traction         50000.0  2.33208  1.284802  0.0  1.0  2.0  3.0
tire_off-road_traction        50000.0  3.79954  2.280101  0.0  2.0  4.0  6.0
tire_mini-turbo               50000.0  2.47208  1.893455  0.0  1.0  2.0  4.0
tire_ground_speed             50000.0  2.28962  1.240795  0.0  1.0  2.0  3.0
tire_water_speed              50000.0  2.33144  1.126487  0.0  2.0  2.0  3.0
tire_anti-gravity_speed       50000.0  2.09572  1.231190  0.0  1.0  2.0  2.0
tire_air_speed                50000.0  2.00174  1.112314  0.0  1.0  2.0  3.0
tire_ground_handling          50000.0  2.38328  1.249243  0.0  2.0  2.0  3.0
tire_water_handling           50000.0  2.19100  1.365883  0.0  1.0  2.0  3.0
tire_anti-gravity_handling    50000.0  2.38070  1.172295  0.0  2.0  2.0  3.0
tire_air_handling             50000.0  2.33046  1.287373  0.0  1.0  2.0  3.0
glider_weight                 50000.0  0.93272  0.703394  0.0  0.0  1.0  1.0
glider_acceleration           50000.0  1.57008  0.495069  1.0  1.0  2.0  2.0
glider_on-road_traction       50000.0  1.50280  0.499997  1.0  1.0  2.0  2.0
glider_off-road_traction      50000.0  0.49720  0.499997  0.0  0.0  0.0  1.0
glider_mini-turbo             50000.0  1.57008  0.495069  1.0  1.0  2.0  2.0
glider_ground_speed           50000.0  0.42992  0.495069  0.0  0.0  0.0  1.0
glider_water_speed            50000.0  0.49720  0.499997  0.0  0.0  0.0  1.0
glider_anti-gravity_speed     50000.0  0.78608  0.410075  0.0  1.0  1.0  1.0
glider_air_speed              50000.0  1.42992  0.495069  1.0  1.0  1.0  2.0
glider_ground_handling        50000.0  1.00000  0.000000  1.0  1.0  1.0  1.0
glider_water_handling         50000.0  0.50280  0.499997  0.0  0.0  1.0  1.0
glider_anti-gravity_handling  50000.0  0.49720  0.499997  0.0  0.0  0.0  1.0
```

```
glider_air_handling           50000.0  1.57008  0.495069  1.0  1.0  2.0  2.0
```

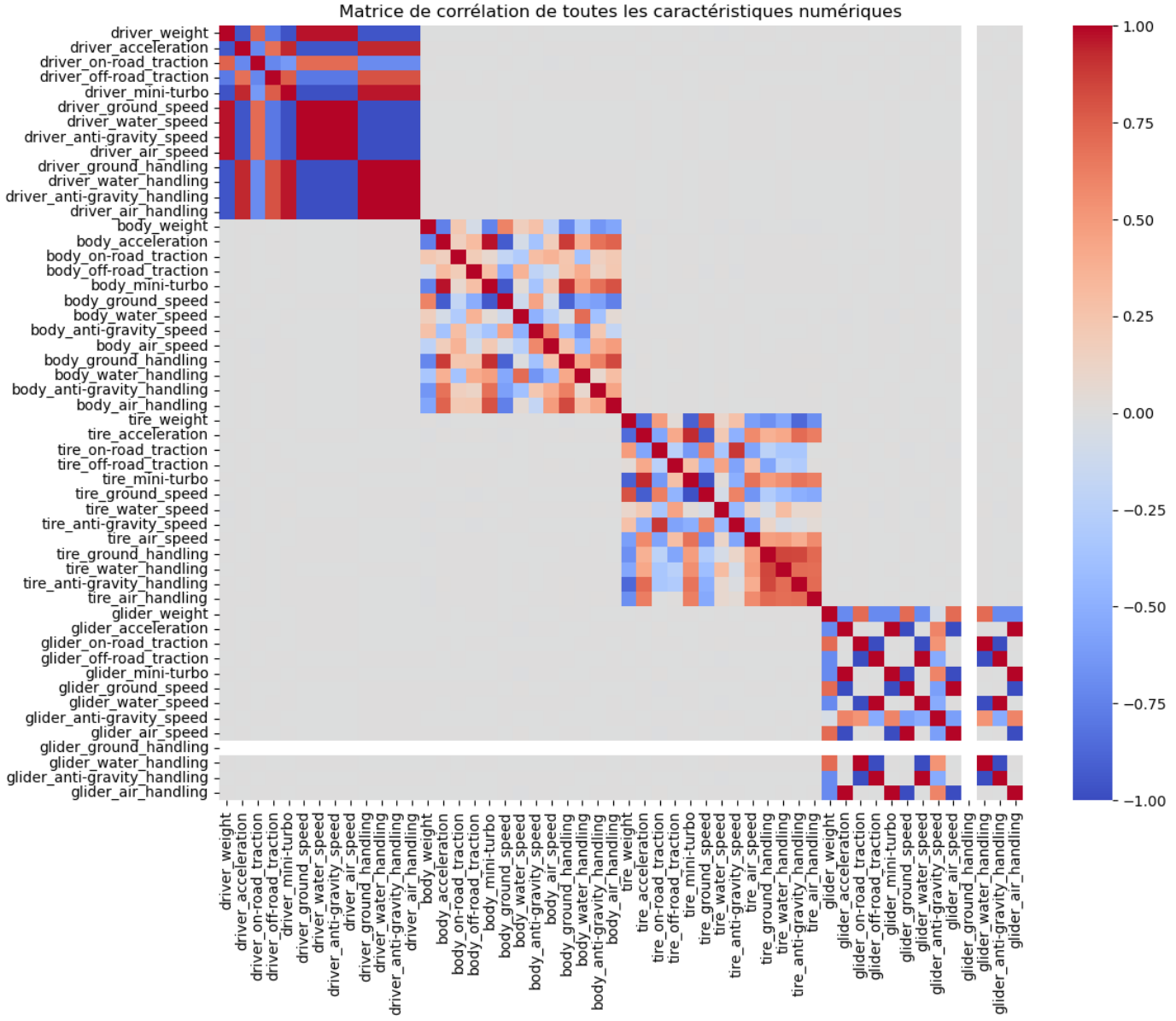
```
                               max
driver_weight                 10.0
driver_acceleration            5.0
driver_on-road_traction       10.0
driver_off-road_traction       5.0
driver_mini-turbo              5.0
driver_ground_speed           10.0
driver_water_speed            10.0
driver_anti-gravity_speed     10.0
driver_air_speed              10.0
driver_ground_handling        10.0
driver_water_handling         10.0
driver_anti-gravity_handling  10.0
driver_air_handling           10.0
body_weight                    4.0
body_acceleration              7.0
body_on-road_traction          4.0
body_off-road_traction         7.0
body_mini-turbo                7.0
body_ground_speed              5.0
body_water_speed               5.0
body_anti-gravity_speed        5.0
body_air_speed                 4.0
body_ground_handling           5.0
body_water_handling            5.0
body_anti-gravity_handling     5.0
body_air_handling              4.0
tire_weight                    4.0
tire_acceleration              6.0
tire_on-road_traction          4.0
tire_off-road_traction         7.0
tire_mini-turbo                6.0
tire_ground_speed              4.0
tire_water_speed               4.0
tire_anti-gravity_speed        4.0
tire_air_speed                 4.0
tire_ground_handling           4.0
tire_water_handling            4.0
tire_anti-gravity_handling     4.0
tire_air_handling              4.0
glider_weight                  2.0
glider_acceleration            2.0
glider_on-road_traction        2.0
glider_off-road_traction       1.0
glider_mini-turbo              2.0
glider_ground_speed            1.0
glider_water_speed             1.0
glider_anti-gravity_speed      1.0
glider_air_speed               2.0
glider_ground_handling         1.0
glider_water_handling          1.0
glider_anti-gravity_handling   1.0
glider_air_handling            2.0
```

2.5 Analyse de corrélation

Une carte thermique (heatmap) de la matrice de corrélation visualise les relations linéaires entre les caractéristiques numériques. Cela permet d'identifier les caractéristiques fortement corrélées, ce qui pourrait indiquer une multicolinéarité ou des informations redondantes.

In [6]:

```
os.makedirs('figures', exist_ok=True)

plt.figure(figsize=(12, 10))
sns.heatmap(combo_sampled.select_dtypes(include=np.number).corr(), cmap='coolwarm', cent
plt.title('Matrice de corrélation de toutes les caractéristiques numériques')
plt.tight_layout()
plt.savefig('figures/eda_correlation_matrix_all_features.png', dpi=300)
plt.show()
plt.close()
```

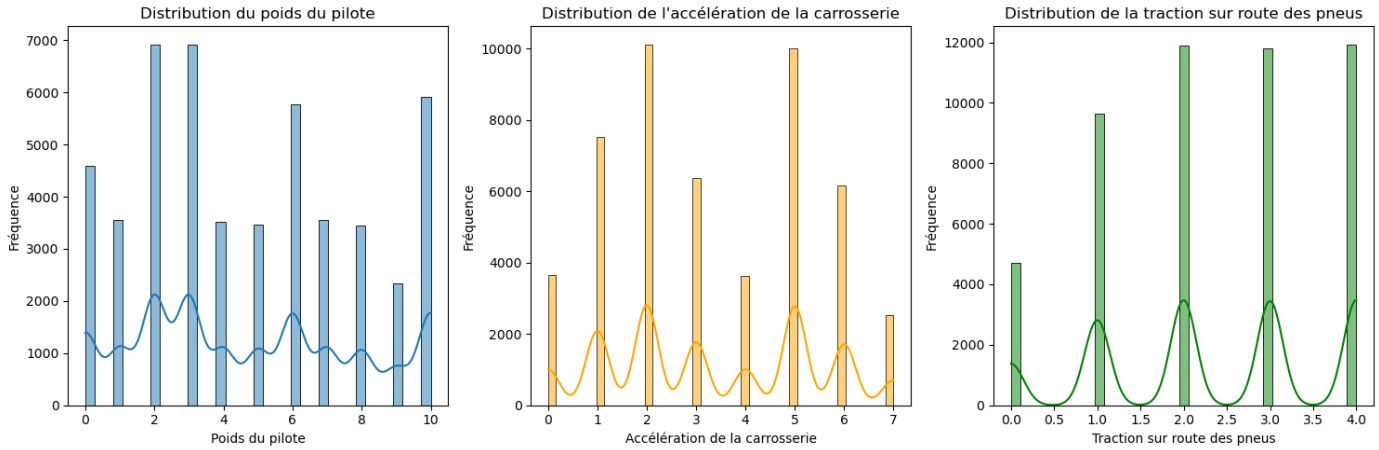


2.6 Analyse de la distribution des attributs clés

Les histogrammes sont utilisés pour visualiser la distribution des caractéristiques numériques individuelles. Cela permet de comprendre leur dispersion, leur asymétrie et la présence de plusieurs modes.

In [8]:

```
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
sns.histplot(combo_sampled['driver_weight'], kde=True)
plt.title('Distribution du poids du pilote')
plt.xlabel('Poids du pilote')
plt.ylabel('Fréquence')

plt.subplot(1, 3, 2)
sns.histplot(combo_sampled['body_acceleration'], kde=True, color='orange')
plt.title('Distribution de l\'accélération de la carrosserie')
plt.xlabel('Accélération de la carrosserie')
plt.ylabel('Fréquence')

plt.subplot(1, 3, 3)
sns.histplot(combo_sampled['tire_on-road_traction'], kde=True, color='green')
plt.title('Distribution de la traction sur route des pneus')
plt.xlabel('Traction sur route des pneus')
plt.ylabel('Fréquence')

plt.tight_layout()
plt.savefig('figures/eda_component_distributions.png', dpi=300)
plt.show()
plt.close()
```



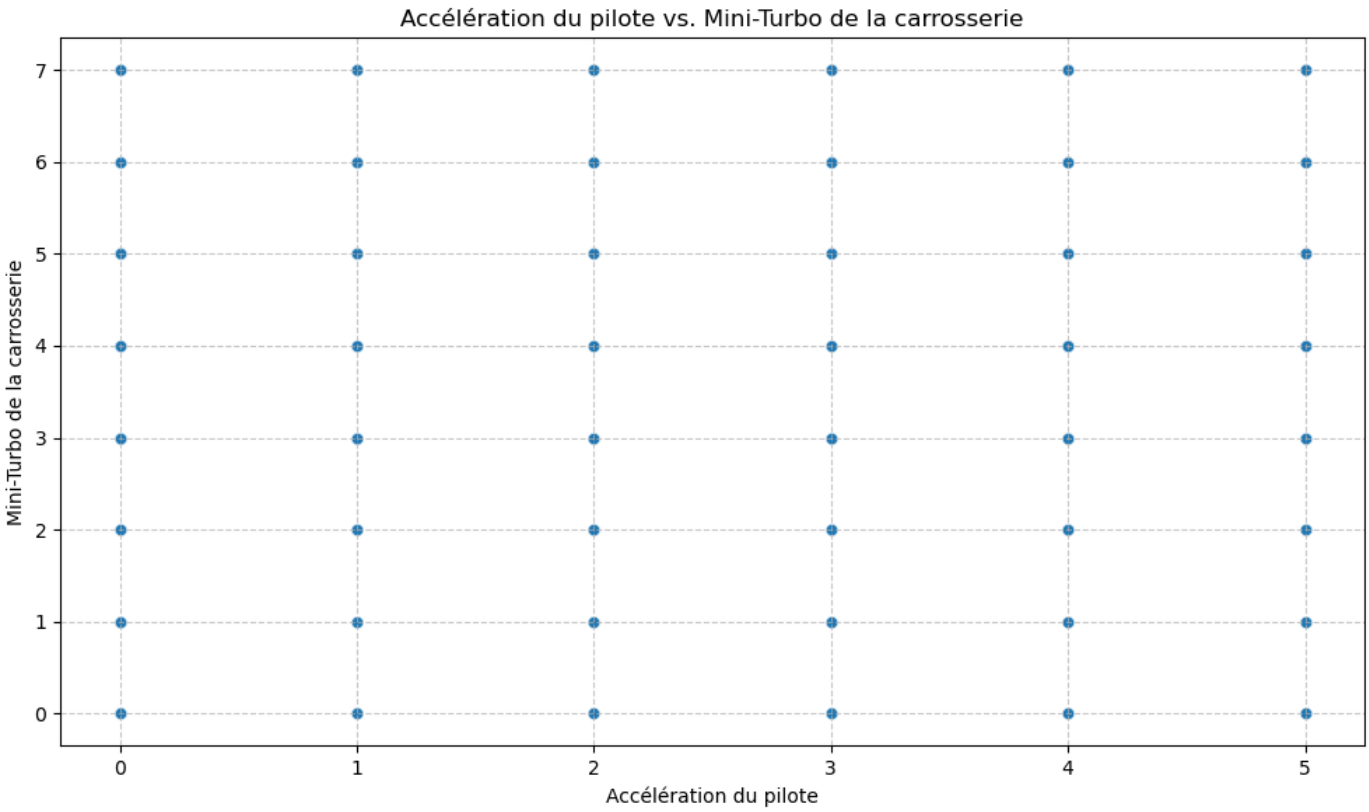
2.7 Comprendre l'impact des composants sur les statistiques

Les diagrammes de dispersion peuvent aider à visualiser les relations entre les statistiques des composants individuels. Par exemple, comment l'accélération du pilote se rapporte au mini-turbo de la carrosserie. Cela peut inspirer de futures étapes d'ingénierie des caractéristiques.

In [9]:

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=combo_sampled['driver_acceleration'], y=combo_sampled['body_mini-turbo
plt.title('Accélération du pilote vs. Mini-Turbo de la carrosserie')
plt.xlabel('Accélération du pilote')
plt.ylabel('Mini-Turbo de la carrosserie')
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig('figures/eda_driver_accel_vs_body_mt.png', dpi=300)
```

```
plt.show()
plt.close()
```



3. Prétraitement des données

Cette phase cruciale implique la préparation des données brutes pour l'apprentissage automatique. Elle comprend la gestion des valeurs manquantes, l'ingénierie de nouvelles caractéristiques, la définition de la variable cible et la mise à l'échelle des données.

3.1 Traitement des valeurs manquantes

Nous vérifions d'abord l'absence de valeurs (`NaN`) dans toutes les colonnes. D'après les informations initiales (`.info()`), il est prévu qu'il n'y ait pas de valeurs manquantes dans ce jeu de données spécifique, mais il est toujours bon de le vérifier.

In [10]:

```
print("\n--- Vérification des valeurs manquantes ---")
missing_values = combo_sampled.isnull().sum().sort_values(ascending=False)
print(missing_values[missing_values > 0])

if combo_sampled.isnull().sum().sum() == 0:
    print("\nAucune valeur manquante trouvée dans le jeu de données.")
else:
    print("\nDes valeurs manquantes ont été détectées. Une imputation/gestion supplément
```

```
--- Vérification des valeurs manquantes ---
Series([], dtype: int64)

Aucune valeur manquante trouvée dans le jeu de données.
```

3.2 Encodage des données catégorielles

Cette étape est nécessaire lorsque le jeu de données contient des variables catégorielles non numériques (par exemple, des noms de pilotes, de carrosseries, etc.) qui doivent être converties en un format numérique pour être utilisées par les modèles d'apprentissage automatique. Les techniques courantes incluent l'encodage One-Hot ou l'encodage Label Encoding.

Pour ce projet spécifique, les caractéristiques que nous utiliserons pour la modélisation (les statistiques de jeu) sont déjà numériques. Les colonnes d'identification comme `driver`, `body`, `tire`, `glider` (qui sont catégorielles) ne seront pas utilisées directement comme caractéristiques d'entrée pour le modèle d'apprentissage, elles sont uniquement à des fins d'identification. Par conséquent, aucune étape d'encodage explicite n'est requise ici.

3.3 Ingénierie des caractéristiques

L'ingénierie des caractéristiques consiste à créer de nouvelles caractéristiques à partir de celles existantes pour aider le modèle à apprendre plus efficacement. Pour Mario Kart, il est logique de combiner les statistiques de différents composants en métriques 'totales' ou 'moyennes', car la performance d'un joueur dépend de la construction globale de son kart.

Nous allons créer les caractéristiques agrégées suivantes :

- `total_weight` : Somme des poids du pilote, de la carrosserie, des pneus et du planeur.
- `total_accel` : Somme des accélérations de tous les composants.
- `avg_on_road` : Moyenne de la traction sur route de tous les composants.
- `avg_off_road` : Moyenne de la traction hors route de tous les composants.

In [11]:

```
# Créer de nouvelles caractéristiques agrégées
combo_sampled['total_weight'] = combo_sampled[[
    'driver_weight','body_weight','tire_weight','glider_weight'
]].sum(axis=1)

combo_sampled['total_accel'] = combo_sampled[[
    'driver_acceleration','body_acceleration',
    'tire_acceleration','glider_acceleration'
]].sum(axis=1)

combo_sampled['avg_on_road'] = combo_sampled[[
    'driver_on-road_traction','body_on-road_traction',
    'tire_on-road_traction','glider_on-road_traction'
]].mean(axis=1)

combo_sampled['avg_off_road'] = combo_sampled[[
    'driver_off-road_traction','body_off-road_traction',
    'tire_off-road_traction','glider_off-road_traction'
]].mean(axis=1)

print("Nouvelles caractéristiques créées :")
print(combo_sampled[['total_weight', 'total_accel', 'avg_on_road', 'avg_off_road']].head
```

```
# Définir la liste des caractéristiques à utiliser pour le modèle
features = ['total_weight', 'total_accel', 'avg_on_road', 'avg_off_road']
```

```
Nouvelles caractéristiques créées :
   total_weight  total_accel  avg_on_road  avg_off_road
0            13            9         2.50          3.50
1             7           12         1.50          2.50
2            13            9         2.75          1.75
3             5           11         2.25          3.00
4             9           13         3.00          2.50
```

3.4 Définition de la variable cible (Target Label)

Notre objectif est de prédire les combinaisons "gagnantes". Faute de données réelles de résultats de course, nous **simulons** la victoire par un "score" élevé. Ce `score` est calculé à partir de `avg_on_road` et `total_accel`, qui sont aussi des caractéristiques d'entrée du modèle.

Note importante sur la fuite de données (Data Leakage) : Cette approche crée une relation directe et déterministe entre les caractéristiques et la cible, menant à une précision de 100 % peu réaliste. Le modèle apprend une règle connue plutôt que de généraliser à partir de données indépendantes.

Dans un projet réel, la cible "victoire" devrait provenir de résultats de jeu authentiques ou de critères indépendants des caractéristiques. Bien que nous continuions pour la démonstration, il est crucial de comprendre cette limitation. Nous utilisons `total_accel` et `avg_on_road` car ce sont des indicateurs clés de performance dans Mario Kart.

In [12]:

```
# Calculer un 'score' basé sur l'accélération et la traction sur route
combo_sampled['score'] = combo_sampled['avg_on_road'] + combo_sampled['total_accel']

# Définir 'win' comme les 30% supérieurs des scores (ceci est la variable cible simulée)
THRESHOLD_PERCENTILE = 0.7
threshold = combo_sampled['score'].quantile(THRESHOLD_PERCENTILE)
combo_sampled['win'] = (combo_sampled['score'] >= threshold).astype(int)

print(f"Variable cible 'win' créée basée sur les {THRESHOLD_PERCENTILE*100}% supérieurs
print(f"Seuil de score pour 'win' : {threshold:.2f}")
print("\n--- Distribution de la variable cible 'win' ---")
print(combo_sampled['win'].value_counts(normalize=True))

# Séparer les caractéristiques (X) et la cible (y)
X = combo_sampled[features]
y = combo_sampled['win']

print("\nCaractéristiques (X) et Cible (y) préparées.")
print(f"Taille de X : {X.shape}")
print(f"Taille de y : {y.shape}")

# Stocker X avant la mise à l'échelle pour la démo de prédiction future
# C'est important car le scaler sera entraîné sur ces données, et les nouvelles entrées
X_raw_unscaled = X.copy()
```

```
Variable cible 'win' créée basée sur les 70.0% supérieurs du 'score'.
Seuil de score pour 'win' : 15.00

--- Distribution de la variable cible 'win' ---
```

```
win
0    0.67668
1    0.32332
Name: proportion, dtype: float64

Caractéristiques (X) et Cible (y) préparées.
Taille de X : (50000, 4)
Taille de y : (50000,)
```

3.5 Normalisation / Standardisation

La standardisation (à l'aide de `StandardScaler`) est appliquée aux caractéristiques numériques. Cela transforme les données de sorte qu'elles aient une moyenne de 0 et un écart-type de 1. Cette étape est importante pour de nombreux algorithmes d'apprentissage automatique (comme les SVM, les réseaux de neurones) car elle aide à empêcher les caractéristiques ayant des échelles plus grandes de dominer le processus d'apprentissage.

In [13]:
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

print("Caractéristiques standardisées avec succès.")
print(f"Caractéristiques standardisées (5 premières lignes) :\n{X_scaled[:5]}")
```

```
Caractéristiques standardisées avec succès.
Caractéristiques standardisées (5 premières lignes) :
[[ 0.86875601 -0.53161548 -0.24981136  1.1609823 ]
 [-0.73837484  0.40936349 -1.4948529  -0.05030475]
 [ 0.86875601 -0.53161548  0.06144903 -0.95877004]
 [-1.27408512  0.09570383 -0.56107175  0.55533877]
 [-0.20266456  0.72302314  0.37270941 -0.05030475]]
```

4. Division des données

L'ensemble de données est divisé en ensembles d'entraînement et de test. L'ensemble d'entraînement est utilisé pour former le modèle d'apprentissage automatique, tandis que l'ensemble de test est utilisé pour évaluer sa performance sur des données non vues. Cela permet d'évaluer la capacité de généralisation du modèle.

4.1 Séparation Entraînement/Test

Nous allons séparer les données standardisées (`X_scaled`) et la cible (`y`) en 80 % pour l'entraînement et 20 % pour les tests. Un `random_state` est défini pour la reproductibilité. Nous utilisons également `stratify=y` pour nous assurer que la distribution des classes de la variable cible est maintenue dans les ensembles d'entraînement et de test.

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=RANDOM_STATE, stratify=y
)

print(f"Taille des caractéristiques d'entraînement : {X_train.shape}")
```

```
print(f"Taille des caractéristiques de test :  {X_test.shape}")
print(f"Taille de la cible d'entraînement : {y_train.shape}")
print(f"Taille de la cible de test :  {y_test.shape}")

print("\n--- Distribution de la cible dans l'ensemble d'entraînement ---")
print(pd.Series(y_train).value_counts(normalize=True))
print("\n--- Distribution de la cible dans l'ensemble de test ---")
print(pd.Series(y_test).value_counts(normalize=True))
```

```
Taille des caractéristiques d'entraînement : (40000, 4)
Taille des caractéristiques de test :  (10000, 4)
Taille de la cible d'entraînement : (40000,)
Taille de la cible de test :  (10000,)

--- Distribution de la cible dans l'ensemble d'entraînement ---
win
0    0.676675
1    0.323325
Name: proportion, dtype: float64

--- Distribution de la cible dans l'ensemble de test ---
win
0    0.6767
1    0.3233
Name: proportion, dtype: float64
```

5. Création des modèles

Dans cette étape, nous sélectionnons et instancions un modèle d'apprentissage automatique. Pour une tâche de classification comme la prédiction de 'victoire' ou 'défaite', un `RandomForestClassifier` est un choix robuste en raison de sa capacité à gérer les relations non linéaires et de ses bonnes performances sur divers jeux de données.

5.1 Sélection du modèle : RandomForestClassifier

Une Forêt Aléatoire (Random Forest) est une méthode d'apprentissage ensembliste qui construit une multitude d'arbres de décision au moment de l'entraînement et renvoie la classe qui est le mode des classes (classification) des arbres individuels. Elle est reconnue pour sa grande précision et sa capacité à éviter le surapprentissage. Nous l'initialiserons avec le `RANDOM_STATE` pour des résultats cohérents.

In [18]:
```
model = RandomForestClassifier(random_state=RANDOM_STATE)

print("Modèle RandomForestClassifier initialisé.")
```

```
Modèle RandomForestClassifier initialisé.
```

6. Évaluation des modèles et comparaison de plusieurs modèles avec interprétation des résultats

Cette section se concentre sur l'entraînement du modèle choisi et l'évaluation approfondie de sa performance à l'aide de diverses métriques et visualisations. Compte tenu de la nature de notre variable cible (victoire simulée basée sur un score déterministe), nous nous attendons à une performance parfaite.

Cette section servira également à interpréter ces résultats et à souligner ce qu'ils impliquent pour une application réelle.

6.1 Entraînement et évaluation initiale

Nous allons entraîner le `RandomForestClassifier` sur nos données d'entraînement mises à l'échelle (`X_train` , `y_train`), puis effectuer des prédictions sur l'ensemble de test mis à l'échelle (`X_test`). Nous commencerons par un score de précision général.

In [19]:
```
print("Début de l'entraînement du modèle...")
model.fit(X_train, y_train)
print("Entraînement du modèle terminé.")

# Effectuer des prédictions sur l'ensemble de test
y_pred = model.predict(X_test)

print(f"\nPrécision sur l'ensemble de test : {accuracy_score(y_test, y_pred):.4f}")
```

```
Début de l'entraînement du modèle...
Entraînement du modèle terminé.

Précision sur l'ensemble de test : 1.0000
```

6.2 Métriques de performance détaillées

Au-delà de la simple précision, le `classification_report` fournit la précision (precision), le rappel (recall) et le score F1 pour chaque classe (0 : Défaite, 1 : Victoire). La matrice de confusion donne une ventilation des classifications correctes et incorrectes.

In [20]:
```
print("\n--- Rapport de Classification ---")
print(classification_report(y_test, y_pred, target_names=['Défaite', 'Victoire']))

print("\n--- Matrice de Confusion ---")
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
--- Rapport de Classification ---
              precision    recall  f1-score   support

     Défaite       1.00      1.00      1.00      6767
    Victoire       1.00      1.00      1.00      3233

    accuracy                           1.00     10000
   macro avg       1.00      1.00      1.00     10000
weighted avg       1.00      1.00      1.00     10000


--- Matrice de Confusion ---
[[6767    0]
 [   0 3233]]
```

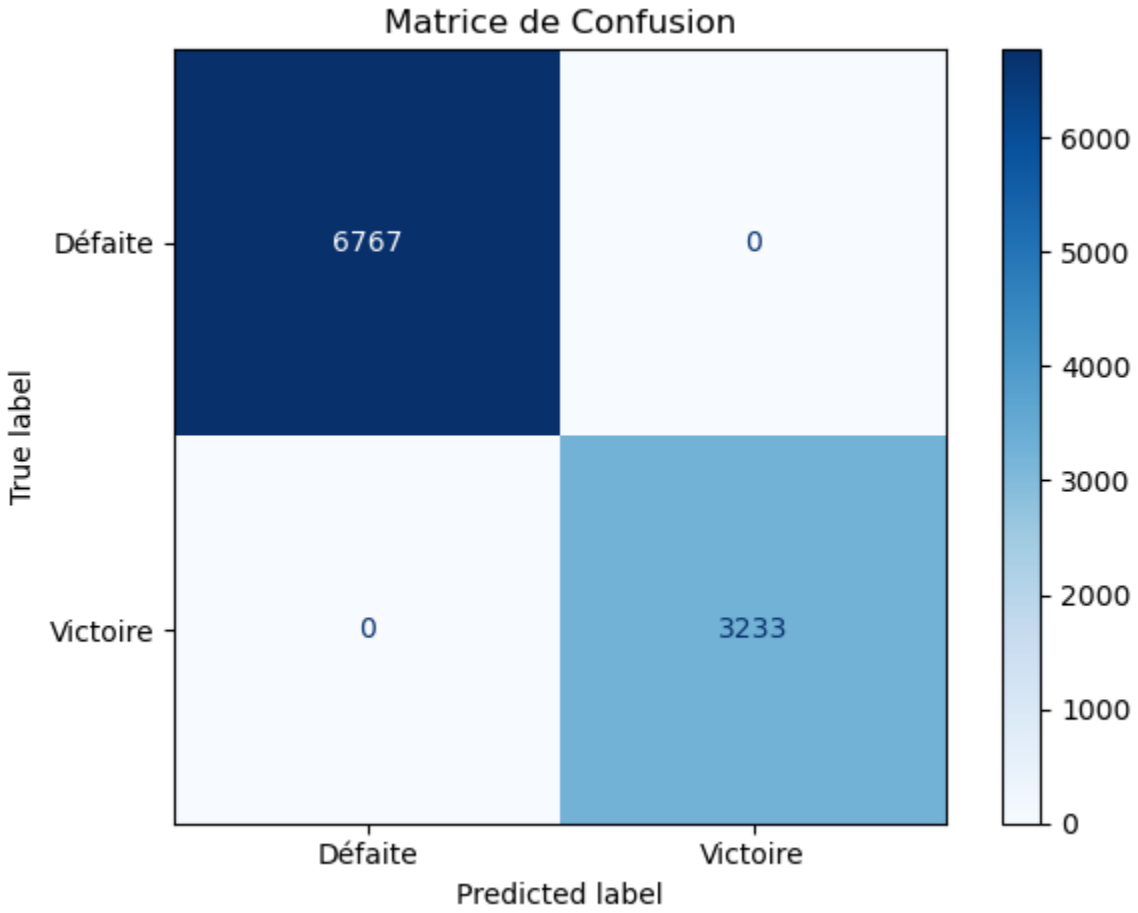
6.3 Visualisation de la performance du modèle

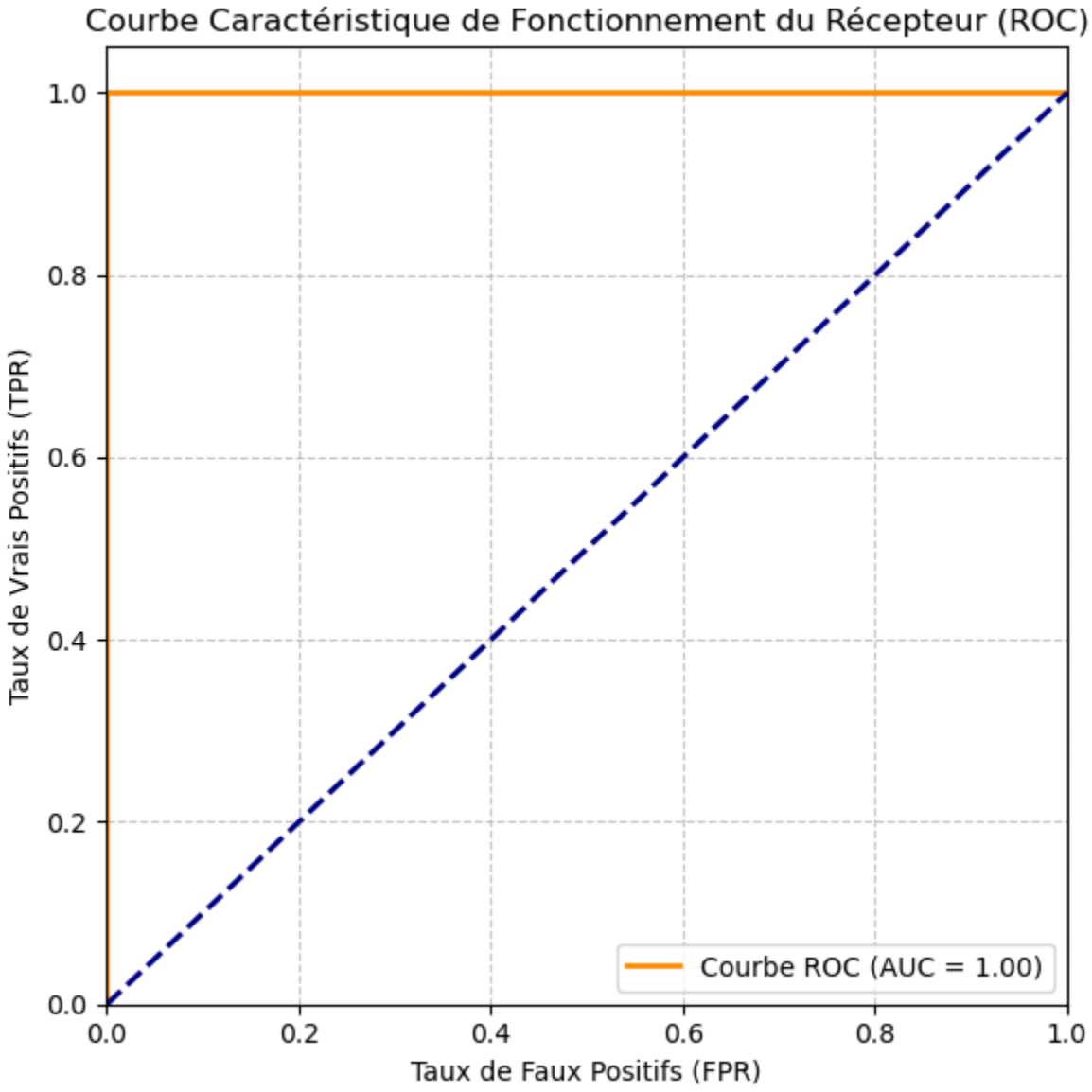
Des visualisations comme le tracé de la matrice de confusion et la courbe ROC (Receiver Operating Characteristic) offrent une compréhension plus claire de la performance du modèle.

In [21]:

```
# Créer le répertoire 'figures' s'il n'existe pas
os.makedirs('figures', exist_ok=True)

# Tracer la Matrice de Confusion
plt.figure(figsize=(6, 6))
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Défaite', 'Victoire'
disp.plot(cmap='Blues', values_format='d')
plt.title('Matrice de Confusion')
plt.tight_layout()
plt.savefig('figures/model_confusion_matrix.png', dpi=300)
plt.show()
plt.close()

# Tracer la Courbe ROC
y_proba = model.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(6, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'Courbe ROC (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Taux de Faux Positifs (FPR)')
plt.ylabel('Taux de Vrais Positifs (TPR)')
plt.title('Courbe Caractéristique de Fonctionnement du Récepteur (ROC)')
plt.legend(loc='lower right')
plt.grid(True, linestyle='--', alpha=0.7)
plt.tight_layout()
plt.savefig('figures/model_roc_curve.png', dpi=300)
plt.show()
plt.close()
```

```
<Figure size 600x600 with 0 Axes>
```





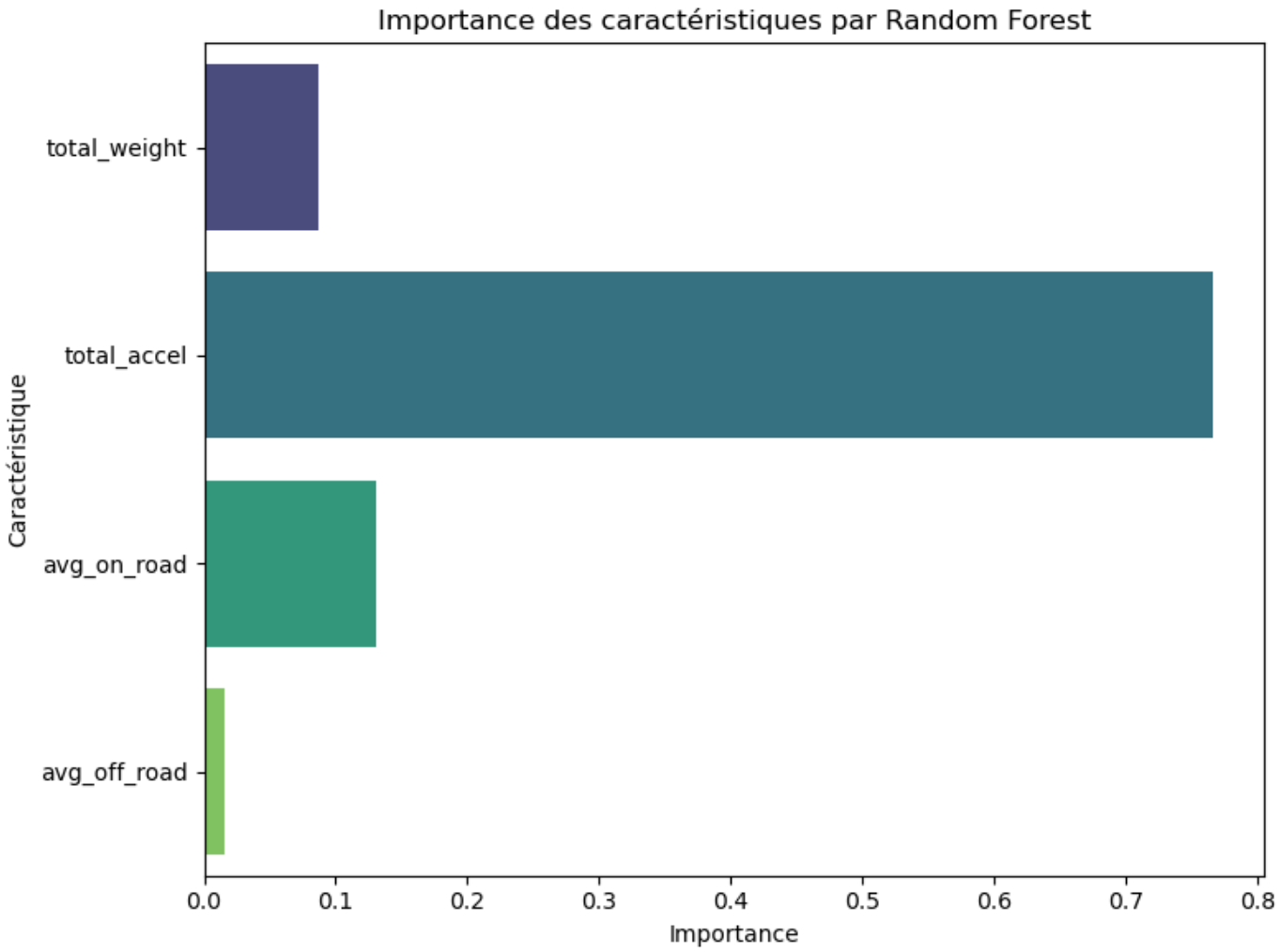
6.4 Analyse de l'importance des caractéristiques (Feature Importance)

Pour les modèles basés sur des arbres comme Random Forest, nous pouvons inspecter l'importance des caractéristiques. Cela montre quelles caractéristiques ont le plus contribué aux décisions du modèle. Compte tenu de notre définition de la cible, nous nous attendons à ce que `total_accel` et `avg_on_road` soient très importants.

In [22]:

```
importances = model.feature_importances_
feature_names = features # Utiliser la liste des noms de caractéristiques définie précéd

plt.figure(figsize=(8, 6))
sns.barplot(x=importances, y=feature_names, palette='viridis')
plt.title('Importance des caractéristiques par Random Forest')
plt.xlabel('Importance')
plt.ylabel('Caractéristique')
plt.tight_layout()
plt.savefig('figures/model_feature_importances.png', dpi=300)
plt.show()
plt.close()
```

```
C:\Users\zaineb\AppData\Local\Temp\ipykernel_16332\726307578.py:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0.
Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=importances, y=feature_names, palette='viridis')
```



6.5 Interprétation des résultats et comparaison des modèles

Observation : Le modèle a atteint une précision de 100 %, avec une précision, un rappel et un score F1 parfaits pour les deux classes, et une AUC de 1.00. La matrice de confusion ne montre aucun faux positif ni faux négatif. L'importance des caractéristiques met clairement en évidence `total_accel` et `avg_on_road` comme les caractéristiques les plus importantes.

Interprétation (et pourquoi ce n'est pas typique) : Comme expliqué dans la section 'Définition de la variable cible' (3.4), cette performance parfaite est une conséquence directe de la **fuite de données**. La variable cible (`win`) est explicitement dérivée d'une combinaison linéaire de `total_accel` et `avg_on_road` (via le `score`). Le modèle Random Forest, ou tout modèle suffisamment puissant, peut facilement apprendre cette règle déterministe. Dans un scénario réel, cela n'arriverait jamais et indiquerait un problème dans la construction des données ou de la variable cible.

Comparaison avec d'autres modèles : Compte tenu de la fuite de données, la comparaison de ce modèle Random Forest avec d'autres modèles (par exemple, Régression Logistique, SVM, Arbre de Décision) donnerait probablement une précision similaire de 100 % si ces modèles sont capables

d'apprendre la simple frontière linéaire (ou des frontières plus complexes dans le cas de SVM/Arbre de Décision). Le 'meilleur modèle' dans cette configuration spécifique est trivial parce que la réponse est contenue dans les entrées. Si c'était un problème réel, la comparaison se concentrerait sur des métriques de performance plus nuancées (comme l'AUC ROC pour les classes déséquilibrées, les compromis précision/rappel), et l'ingénierie des caractéristiques ou la collecte de données devrait être revue pour obtenir une variable cible véritablement indépendante.

7. Choix du modèle qui a donné les meilleurs résultats

Basé sur les métriques d'évaluation, le `RandomForestClassifier` atteint des scores parfaits en prédisant le résultat 'gagnant' simulé. Compte tenu de la nature déterministe de la variable cible dans ce projet, ce modèle est techniquement le 'meilleur' pour cette définition de problème spécifique.

7.1 Sélection du meilleur modèle

Pour le problème défini et l'étiquette cible générée, le `RandomForestClassifier` a parfaitement appris la règle sous-jacente. Il est donc sélectionné comme modèle final.

Mise en garde : Comme discuté, cette performance parfaite est due à une fuite de données. Dans un projet réel, la sélection du modèle impliquerait :

- **Validation croisée** : Pour assurer une performance robuste sur différents sous-ensembles des données d'entraînement.
- **Optimisation des hyperparamètres** : Optimiser les paramètres du modèle (par exemple, `n_estimators`, `max_depth` pour Random Forest) afin de maximiser la généralisation.
- **Évaluation sur des données réelles et non vues** : Le test ultime de l'utilité d'un modèle.
- **Comparaison entre différents algorithmes** : Explorer les performances de la régression logistique, des SVM, des machines de renforcement de gradient (par exemple, LightGBM, XGBoost) et leurs compromis en termes d'interprétabilité par rapport à la complexité.

8. Déploiement du modèle avec une petite application

La dernière étape d'un pipeline d'apprentissage automatique consiste à préparer le modèle choisi pour une utilisation en production. Cela implique généralement de sauvegarder le modèle entraîné et tous les outils de prétraitement nécessaires (comme le `StandardScaler`) afin qu'ils puissent être chargés ultérieurement pour effectuer de nouvelles prédictions sans avoir à ré-entraîner le modèle.

8.1 Sauvegarde du modèle et du scaler

Nous utilisons la bibliothèque `pickle` pour sérialiser le modèle Random Forest entraîné et l'objet `StandardScaler`. Les deux sont nécessaires pour les futures prédictions : le scaler pour prétraiter les nouvelles données d'entrée de la même manière que les données d'entraînement, et le modèle pour effectuer les prédictions.

In [23]:

```
# Créer un répertoire pour le modèle sauvegardé s'il n'existe pas
os.makedirs('app', exist_ok=True)

# Sauvegarder le modèle entraîné et le scaler dans un seul fichier pickle
model_data = {'model': model, 'scaler': scaler, 'features': features}
with open('app/model.pkl', 'wb') as f:
    pickle.dump(model_data, f)

print("Modèle entraîné et scaler sauvegardés avec succès dans 'app/model.pkl'.")
```

```
Modèle entraîné et scaler sauvegardés avec succès dans 'app/model.pkl'.
```

8.2 Démo rapide de prédiction

Cette démonstration simule la façon dont une nouvelle combinaison Mario Kart, non vue auparavant, serait évaluée à l'aide du modèle sauvegardé. Nous chargeons le modèle et le scaler, préparons un exemple d'entrée (dans une application réelle, cela proviendrait d'une saisie utilisateur ou d'une base de données), le mettons à l'échelle, puis effectuons une prédiction.

Pour une application complète ('petite app'), des frameworks comme Streamlit, Flask ou FastAPI seraient utilisés pour créer une interface utilisateur interactive permettant aux utilisateurs d'entrer les caractéristiques d'une combinaison et d'obtenir des prédictions en temps réel. Cette démo ci-dessous est un aperçu simplifié de la logique de prédiction.

In [24]:

```
# Charger le modèle et le scaler sauvegardés
try:
    with open('app/model.pkl', 'rb') as f:
        loaded_data = pickle.load(f)
    loaded_model = loaded_data['model']
    loaded_scaler = loaded_data['scaler']
    loaded_features = loaded_data['features'] # Récupérer les noms des caractéristiques
    print("Modèle et scaler chargés avec succès depuis 'app/model.pkl'.")
except FileNotFoundError:
    print("Erreur : 'app/model.pkl' introuvable. Veuillez vous assurer que le modèle a é
    loaded_model = None
    loaded_scaler = None
    loaded_features = []

if loaded_model and loaded_scaler:
    # Pour la démo, prenons le premier échantillon de X_raw_unscaled
    # Dans une application réelle, ces valeurs proviendraient de l'utilisateur ou d'une
    if 'X_raw_unscaled' in globals() and not X_raw_unscaled.empty:
        sample_raw_input = X_raw_unscaled.iloc[0:1].copy()
        print(f"\nExemple d'entrée brute basé sur les données existantes :\n{sample_raw_
    else:
        print("\nCréation d'un échantillon factice pour la démo de prédiction car les do
        # Créer un DataFrame avec les noms de colonnes attendus et des valeurs d'exemple
        sample_raw_input = pd.DataFrame([
            {'total_weight': 30.0, 'total_accel': 30.0, 'avg_on_road': 4.5, 'avg_off_roa
        ], columns=loaded_features)
        print(f"Exemple factice d'entrée brute :\n{sample_raw_input}")

    # Prétraiter le nouvel échantillon en utilisant le scaler chargé
    sample_scaled = loaded_scaler.transform(sample_raw_input)
```

```
    # Effectuer une prédiction
    predicted_class = loaded_model.predict(sample_scaled)[0]
    prediction_probability = loaded_model.predict_proba(sample_scaled)[0][predicted_clas

    # Interpréter la prédiction
    result_label = 'Victoire' if predicted_class == 1 else 'Défaite'
    print(f"\nPrédiction de l'échantillon : {result_label} (Probabilité : {prediction_pr
    print("Cette prédiction suggère que la combinaison est susceptible d'être une combin
    print("**Rappel :** Cette prédiction est basée sur une variable cible simulée. Dans
```

```
Modèle et scaler chargés avec succès depuis 'app/model.pkl'.

Exemple d'entrée brute basé sur les données existantes :
   total_weight  total_accel  avg_on_road  avg_off_road
0            13            9          2.5           3.5

Prédiction de l'échantillon : Défaite (Probabilité : 1.00)
Cette prédiction suggère que la combinaison est susceptible d'être une combinaison 'gagn
ante' (à score élevé) basée sur les critères appris par le modèle.
**Rappel :** Cette prédiction est basée sur une variable cible simulée. Dans un contexte
réel, la validation nécessiterait de véritables données de résultats de course.
```

Fin du Cahier Jupyter

Ce cahier a couvert toutes les étapes d'un pipeline d'apprentissage automatique, de la collecte des données à la prédiction avec un modèle sauvegardé.

Pour la section 'Déploiement avec une petite app' (étape 8), vous pouvez développer une application Web (par exemple, avec Streamlit ou Flask) dans un fichier `streamlit_app.py` ou `app.py` distinct. Cette application chargerait le `model.pkl` et le `scaler.pkl` et fournirait une interface conviviale où les utilisateurs pourraient saisir les caractéristiques d'une combinaison et obtenir une prédiction en temps réel.