

Support Vector Machine (Regréssion)

Importation des bibliothèques

```
In [2]: import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

Chargement de DataSet

```
In [3]: data = pd.read_csv("gym_members_exercise_tracking.csv")
data
```

Out[3]:

	Age	Gender	Weight (kg)	Height (m)	Max_BPM	Avg_BPM	Resting_BPM	Session_Duration (hours)	Calories_Burned	Workout_Type	Fat_Percentage
0	56	Male	88.3	1.71	180	157	60	1.69	1313.0	Yoga	12.6
1	46	Female	74.9	1.53	179	151	66	1.30	883.0	HIIT	33.9
2	32	Female	68.1	1.66	167	122	54	1.11	677.0	Cardio	33.4
3	25	Male	53.2	1.70	190	164	56	0.59	532.0	Strength	28.8
4	38	Male	46.1	1.79	188	158	68	0.64	556.0	Strength	29.2
...
968	24	Male	87.1	1.74	187	158	67	1.57	1364.0	Strength	10.0
969	25	Male	66.6	1.61	184	166	56	1.38	1260.0	Strength	25.0
970	59	Female	60.4	1.76	194	120	53	1.72	929.0	Cardio	18.8
971	32	Male	126.4	1.83	198	146	62	1.10	883.0	HIIT	28.2
972	46	Male	88.7	1.63	166	146	66	0.75	542.0	Strength	28.8

973 rows × 15 columns



Préparation des données

In [4]:

```
#Vérification des valeurs manquantes
data.isnull().sum()
```

```
Out[4]: Age          0  
Gender        0  
Weight (kg)    0  
Height (m)     0  
Max_BPM        0  
Avg_BPM        0  
Resting_BPM    0  
Session_Duration (hours) 0  
Calories_Burned 0  
Workout_Type    0  
Fat_Percentage 0  
Water_Intake (liters) 0  
Workout_Frequency (days/week) 0  
Experience_Level 0  
BMI            0  
dtype: int64
```

```
In [88]: #Vérification des valeurs aberrantes  
col = ['Age', 'Weight (kg)', 'Height (m)', 'Max_BPM', 'Avg_BPM', 'Resting_BPM', 'Session_Duration (hours)',  
       'Calories_Burned', 'Fat_Percentage', 'Water_Intake (liters)', 'Workout_Frequency (days/week)', 'Experience_Level', 'BMI']  
df = data.copy()  
for col in col:  
    Q1 = df[col].quantile(0.25)  
    Q3 = df[col].quantile(0.75)  
    IQR = Q3 - Q1  
  
    lower_bound = Q1 - 1.5 * IQR  
    upper_bound = Q3 + 1.5 * IQR  
  
    df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]  
print("Taille avant :", len(data))  
print("Taille après :" , len(df))
```

Taille avant : 973

Taille après : 931

```
In [89]: #variable cible  
df['Calories_Burned'].value_counts()
```

```
Out[89]: Calories_Burned  
883.0      6  
1025.0     6  
832.0      5  
1046.0     4  
1150.0     4  
..  
862.0      1  
723.0      1  
1127.0     1  
1113.0     1  
812.0      1  
Name: count, Length: 598, dtype: int64
```

Séparation des données à la variable cible

```
In [90]: X = df.drop(columns=['Calories_Burned'])  
y = df['Calories_Burned']
```

Division des données

```
In [91]: # Division des données (données pour l'entraînement et données pour le test)  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
In [92]: X_train.shape
```

```
Out[92]: (698, 14)
```

```
In [93]: y_train.shape
```

```
Out[93]: (698,)
```

```
In [94]: X_test.shape
```

```
Out[94]: (233, 14)
```

```
In [95]: y_test.shape
```

Out[95]: (233,)

Séparation des colonnes numériques aux colonnes catégorielles

```
In [96]: #Identification des colonnes catégorielles et numériques  
cat_cols = X.select_dtypes(include=['object', 'category']).columns  
num_cols = X.select_dtypes(include=['int64', 'float64']).columns
```

Encodage des variables catégorielles avec LabelEncoder

```
In [97]: #Encodage des colonnes catégorielles  
le = LabelEncoder()  
for col in cat_cols:  
    X_train[col] = le.fit_transform(X_train[col])  
    X_test[col] = le.transform(X_test[col])
```

Standardisation des colonnes numériques avec StandardScaler

```
In [98]: #Standardisation des colonnes numériques  
scaler = StandardScaler()  
X_train[num_cols] = scaler.fit_transform(X_train[num_cols])  
X_test[num_cols] = scaler.transform(X_test[num_cols])
```

Création du model Support Vector Machine (Regréssion)

```
In [99]: #Création du model SVM (Linear)  
model_SVR_Linear = SVR(kernel='linear')  
model_SVR_Linear.fit(X_train, y_train)
```

Out[99]:



Entraînement du model

```
In [100...]
# Prédiction
y_pred = model_SVR_Linear.predict(X_test)
```

Evaluation du model

```
In [101...]
# Evaluation du model
print("Résultats SVR Regression :")
print("MSE:\n ", mean_squared_error(y_test, y_pred))
print("RMSE:\n ", np.sqrt(mean_squared_error(y_test, y_pred)))
print("MAE:\n ", mean_absolute_error(y_test, y_pred))
print("R2:\n ", r2_score(y_test, y_pred))
```

Résultats SVR Regression :

MSE:

1978.5657813139555

RMSE:

44.481072169114306

MAE:

33.557195305813224

R2:

0.9717520784496703

R² = 0.9717, le modèle explique environ 97% de la variance de Calories_Burned, il capture donc très bien la relation entre les variables explicatives et la cible.

Le modèle SVR linéaire est très performant, avec des erreurs faibles et une capacité à expliquer presque toute la variance de la variable cible.

```
In [102...]
import joblib
"""
importe la bibliothèque Joblib, spécialisée dans la sérialisation (sauvegarde/chargement)
d'objets Python volumineux, notamment les modeles scikit-learn.

il est Optimisee pour les objets numpy et scikit-learn, plus rapide et compacte que pickle pour ce cas d'usage.
"""
# Sauvegarder Le modèle entraîné sous le nom "SVR_model.pkl"
#joblib.dump() sérialise L'objet et l'écrit sur disque
#il retourne une liste contenant le chemin du fichier écrit
```

```
joblib.dump(model_SVR_Linear, 'SVR_model.pkl')
print("Modele sauvegarde sous le nom 'SVR_model.pkl'")
```

Modele sauvegarde sous le nom 'SVR_model.pkl'