Projet 8

Prédire l'issue des combats de MMA

Predicting outcome of MMA fights

Notre objectif

Les paris sportifs sont une industrie de plusieurs milliards de dollars. Sa popularité, tant sur le marché régional que mondial, en fait une activité très attrayante.

Les combats de MMA se classent parmi les meilleurs de l'industrie. Nous aimerions construire un modèle qui nous permettrait de prédire l'issue des combats de MMA, sur la base de données historiques. Le résultat idéal est un modèle qui peut à la fois prédire le vainqueur et donner une probabilité suffisamment élevée (>65-70%) pour ce résultat. De cette façon, nous pourrions également tirer parti des facteurs de risque lorsque nous décidons de parier selon le modèle ou contre lui.

Notre jeu de données

Nos données :

- Enregistrements historiques des combats de MMA provenant de kaggle et du site de l'UFC.
- De 1993 à 2019
- contenant divers paramètres statistiques pour chaque combattant et chaque combat.

	R_fighter	B_fighter	Referee	date	location	Winner	title_bout	weight_class	no_of_rounds	B_current_lose_streak	B_current_win_streak
0	Henry Cejudo	Marlon Moraes	Marc Goddard	2019-06-08	Chicago, Illinois, USA	Red	True	Bantamweight	5	0	4
1	Valentina Shevchenko	Jessica Eye	Robert Madrigal	2019-06-08	Chicago, Illinois, USA	Red	True	Women's Flyweight	5	0	3
2	Tony Ferguson	Donald Cerrone	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Red	False	Lightweight	3	0	3
3	Jimmie Rivera	Petr Yan	Kevin MacDonald	2019-06-08	Chicago, Illinois, USA	Blue	False	Bantamweight	3	0	4
4	Tai Tuivasa	Blagoy Ivanov	Dan Miragliotta	2019-06-08	Chicago, Illinois, USA	Blue	False	Heavyweight	3	0	1

Notre jeu de données

- Le jeu de données contient plus de 5144 combats et un grand nombre de variables, 145.
- Cela pourrait compliquer la construction du modèle et la prédiction, donc l'une des premières choses que nous devons envisager est de réduire le nombre de colonnes.
- L'ensemble de données décrit ~26 ans de combats, de 1993 à 2019.

La vérification des valeurs manquantes nous montre que :
- 25 % des attributs des chasseurs bleus sont manquants
- 13% des attributs des combattants rouges sont manquants
- le nombre d'autres valeurs manquantes est faible

```
print(df.date.min())
print(df.date.max())
```

```
1993-11-12
2019-06-08
```

```
df.shape
```

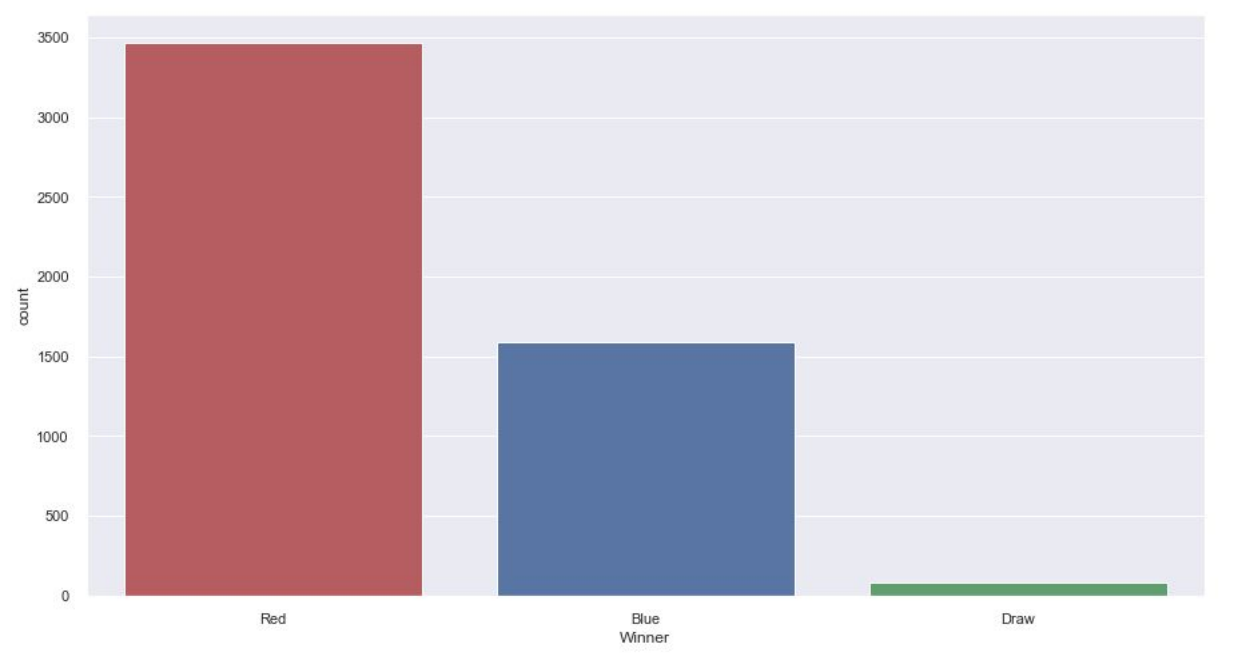
```
(5144, 145)
```

```
print((df.isna().sum()/df.shape[0]*100).round(2))
print(' ')
print(df.isna().sum())
```

```
R_fighter                   0.00
B_fighter                   0.00
Referee                     0.45
date                        0.00
location                    0.00
Winner                      0.00
title_bout                  0.00
weight_class                0.00
no_of_rounds                0.00
B_current_lose_streak       0.00
B_current_win_streak        0.00
B_draw                      0.00
B_avg_BODY_att             24.59
B_avg_BODY_landed          24.59
B_avg_CLINCH_att           24.59
B_avg_CLINCH_landed        24.59
B_avg_DISTANCE_att         24.59
B_avg_DISTANCE_landed      24.59
```

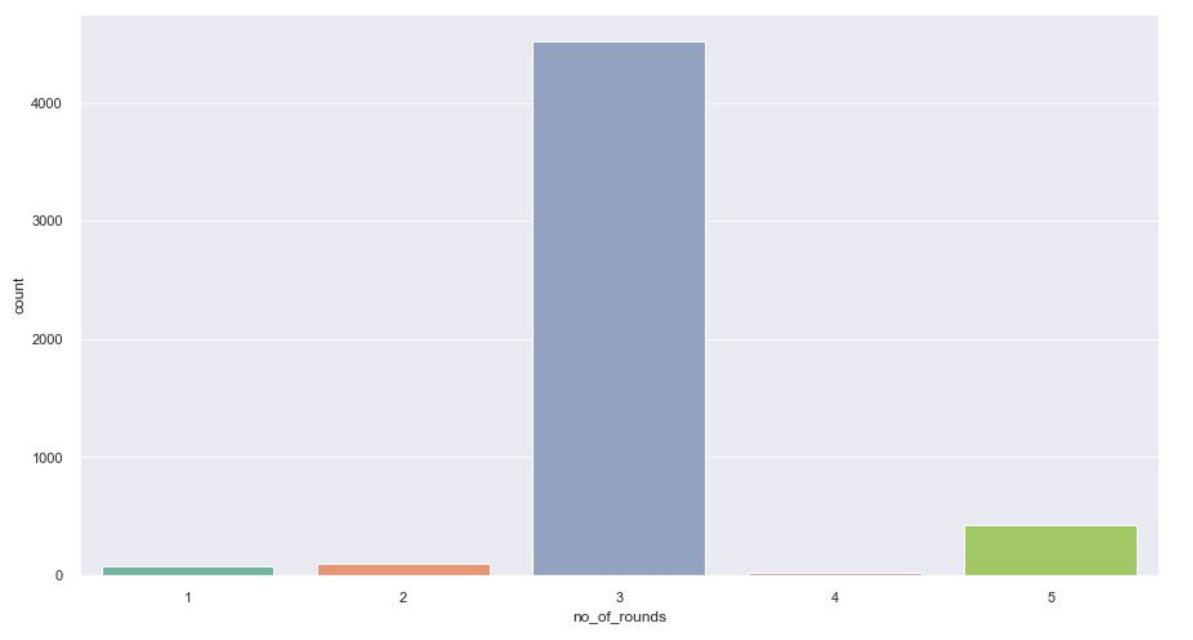
Notre jeu de données

Notre variable dépendante souhaitée est fortement déséquilibrée.
~70% des victoires sont pour les combattants rouges, et seulement 30% pour les bleus. Le nombre de tirages au sort est minuscule. Envisagez une distribution égale des gagnants pour les données du train (stratify=y pour sklearn ou class_weight).



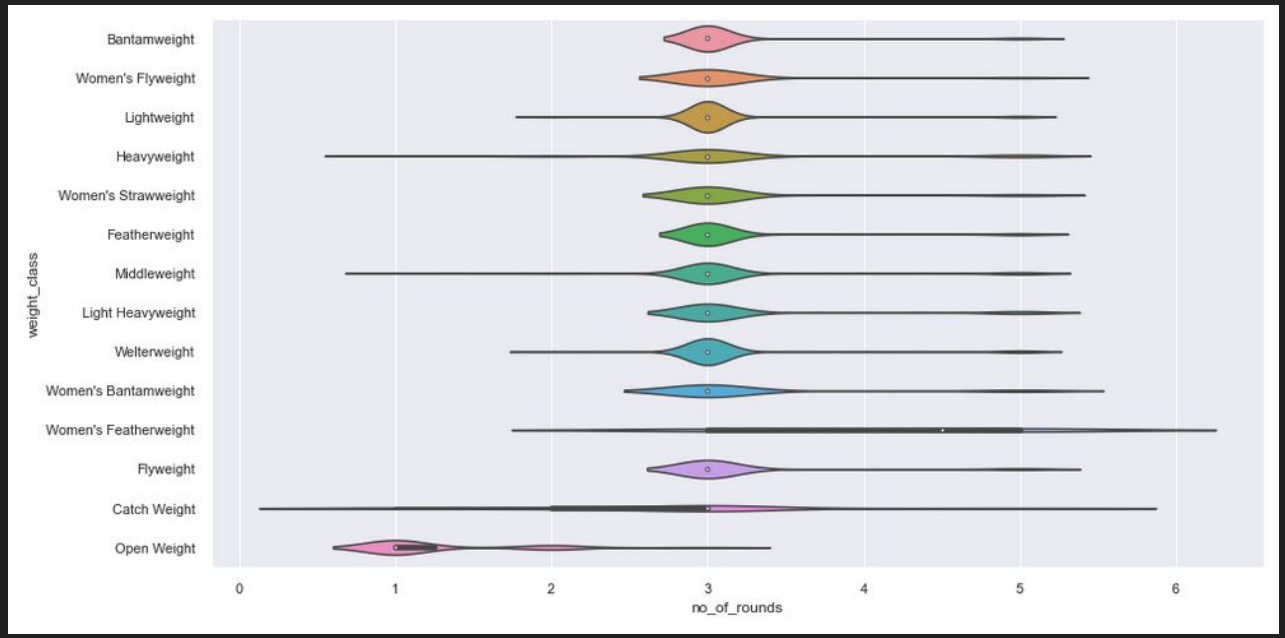
Notre jeu de données

La majorité absolue des combats duraient trois rounds.
La deuxième valeur est celle des combats de 5 rounds, tandis que les combats de 1, 2 et 4 rounds sont très peu représentés dans l'ensemble de données.



Notre jeu de données

Si nous répartissons la durée des combats par catégorie de poids, nous constatons que la tendance aux trois rounds se maintient. Les exceptions sont les poids ouverts (inférieurs à la moyenne) et les poids plumes féminins (supérieurs à la moyenne).



EDA et le nettoyage des données

Nous avons perdu ~2000 lignes en supprimant les NaNs. Nous avons réduit manuellement le nombre de colonnes (caractéristiques pour la modélisation) de 145 à 32. Les colonnes conservées correspondent aux données historiques que nous pouvons avoir avant que le combat n'ait lieu.

```python
# supprimer tous les résultats de tirage de l'ensemble de données
# dropping all draw outcomes from the dataset
df.drop(df.loc[df['Winner']=='Draw'].index, inplace=True)

# Suppression des colonnes non pertinentes
# dropping irrelevant columns
df = df.drop(columns=['date'], axis=1)

# Suppression de toutes les lignes qui contiennent des NaN.
# drop all rows that have any NaN
df = df.dropna(axis=0, how='any')

# Réduction du nombre de colonnes de 145 à 32
# reduce amount of columns from 145 to 32
df = df[['Winner', 'title_bout', 'weight_class', 'Referee',
         'B_fighter', 'B_current_lose_streak', 'B_current_win_streak',
         'B_losses', 'B_total_rounds_fought', 'B_total_time_fought(seconds)', 'B_total_title_bouts',
         'B_wins', 'B_Stance', 'B_Height_cms', 'B_Reach_cms', 'B_Weight_lbs', 'B_age',
         'R_fighter', 'R_current_lose_streak', 'R_current_win_streak',
         'R_losses', 'R_total_rounds_fought', 'R_total_time_fought(seconds)',
         'R_total_title_bouts', 'R_wins', 'R_Stance', 'R_Height_cms', 'R_Reach_cms',
         'R_Weight_lbs', 'R_age']]

df.shape
```

```
(3151, 30)
```

EDA et le nettoyage des données

Nous appliquons LabelEncoder à nos colonnes de chaînes de caractères, afin de les rendre accessibles à nos futurs modèles. Nous supprimons également 10 combats aléatoires de notre ensemble de données, afin de disposer de données nouvelles et inédites pour que les modèles puissent vérifier leurs performances.

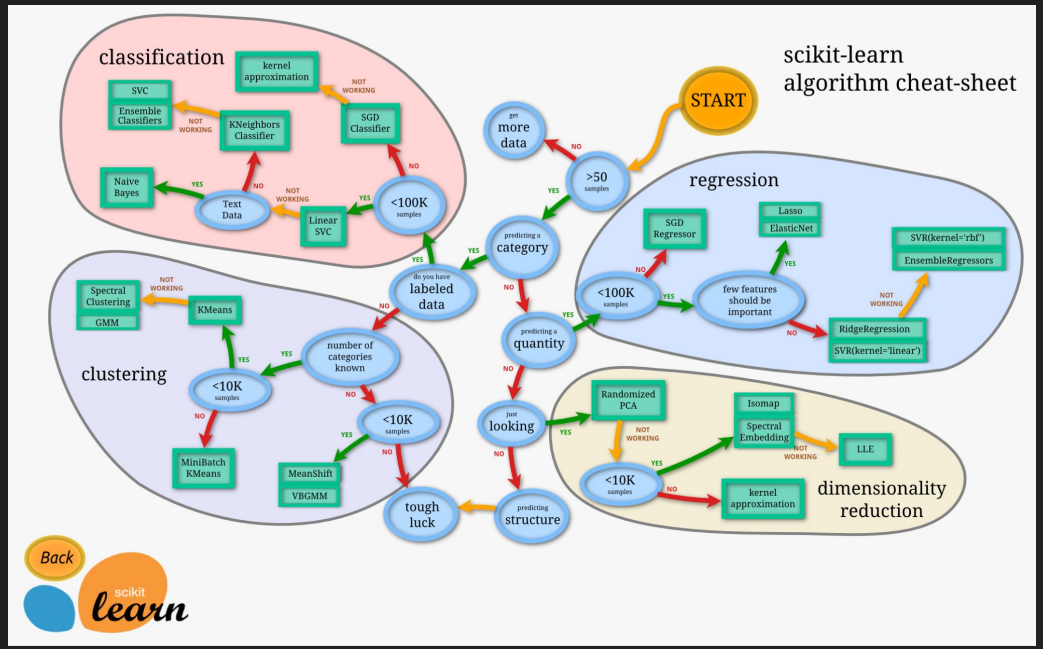
```
print(df.Winner.value_counts())
df['Winner'] = le.fit_transform(df['Winner']) # 1 RED, 0 BLUE
print(df.Winner.value_counts())
```

```
Red     2016
Blue    1135
Name: Winner, dtype: Int64
1    2016
0    1135
Name: Winner, dtype: int64
```

	Winner	title_bout	weight_class	Referee	B_fighter	B_current_lose_streak	B_current_win_streak	B_losses	B_total_rounds_fought
1707	1	0	0	47	894	2	0	7	25
4132	1	0	8	107	1035	0	1	0	1
2558	0	0	6	165	319	2	0	2	9
4604	1	0	8	107	756	0	1	0	3
1725	1	0	7	107	221	0	1	7	39
4047	1	0	7	131	253	2	0	3	17
4151	1	1	7	56	780	0	4	4	18
3728	1	0	8	56	333	2	0	4	16
2018	1	0	6	73	9	0	1	1	4
619	0	0	8	5	690	1	0	2	7

Choisir le type de modèle

Afin de trouver le(s) meilleur(s) modèle(s) pour notre objectif, nous nous tournons vers la documentation de scikit-learn:



Selon ce modèle, notre jeu de données et notre objectif exigent des modèles de classification comme étant les plus optimaux.

Modèles de classification :

- Naive Bayes gaussien
- KNN (K Nearest Neighbors classifier)
- SVC
- Classificateur Random Forest
- Classificateur Decision Trees

Nous pouvons également essayer d'appliquer d'autres types de modèles, à des fins d'expérimentation.

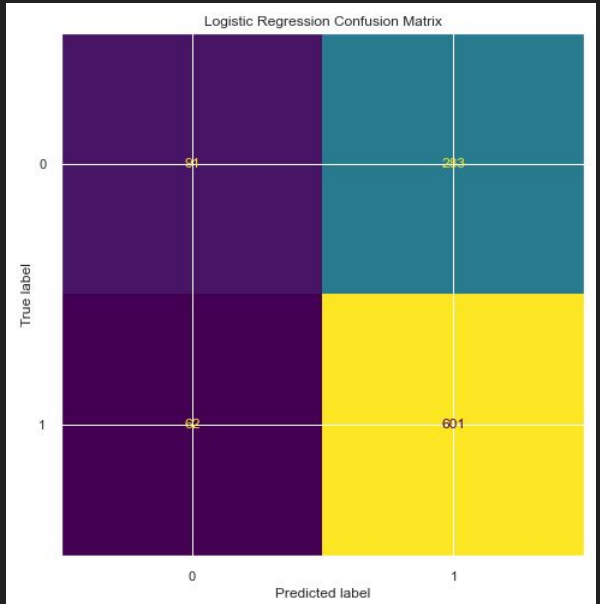
Construire des modèles avec des données standardisées

Pour notre première tentative, nous décidons d'entraîner des modèles en utilisant des données standardisées. Nous utilisons train_test_split de scikit-learn.

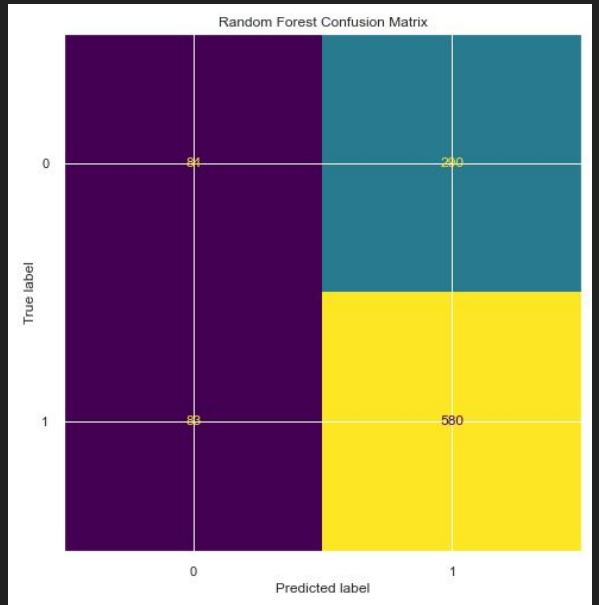
```
# diviser le cadre de données en données de formation et de test
# split dataframe into train and test data

X = df.drop('Winner', axis=1)
y = df['Winner']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=TEST_SIZE, stratify=y, shuffle=True,

# Standardize X data. We leave Y as it is since it is a boolean variable
sc = preprocessing.StandardScaler()
X_train_norm = sc.fit_transform(X_train)
X_test_norm = sc.transform(X_test)
```

Construire des modèles avec des données standardisées

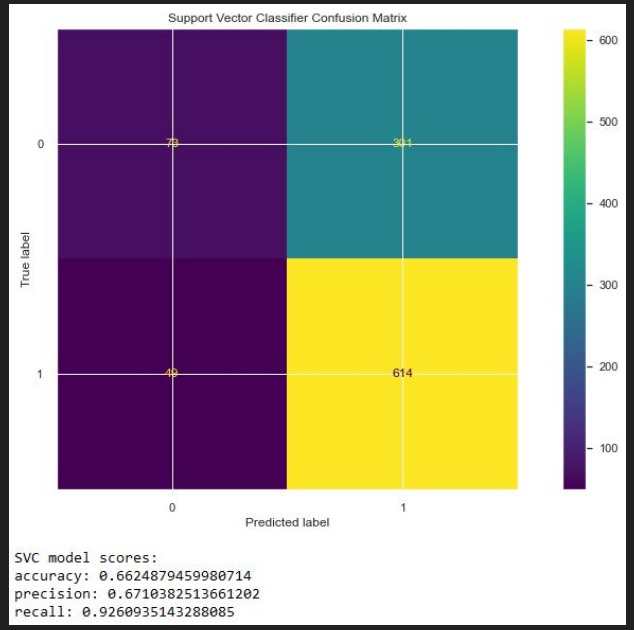
Parmi ces trois modèles, la régression logistique est le modèle le plus performant.



Logistic Regression model scores:
accuracy: 0.6673095467695275
precision: 0.6798642533936652
recall: 0.9064856711915535



Random Forest model scores:
accuracy: 0.6403085824493732
precision: 0.6666666666666666
recall: 0.8748114630467572



SVC model scores:
accuracy: 0.6624879459980714
precision: 0.6710382513661202
recall: 0.9260935143288085

Construire des modèles avec des données standardisées

Mais lorsque nous montrons à nos modèles des données absolument nouvelles, quelque chose d'étrange se produit.

```
print('Logistic Regression predicts the following winners:')
print(lr_model.predict(df_new))
print(validation_sample.Winner.values)
print('Actual fight results')
```

```
Logistic Regression predicts the following winners:
[0 1 0 0 0 0 1 0 0 1]
[1 1 0 1 1 1 1 1 1 0]
Actual fight results
```

```
print('Accuracy score for Logistic Regression on new data is', accuracy_
print('Precision score for Logistic Regression on new data is', precisio
print('Recall score for Logistic Regression on new data is', recall_scor
```

```
Accuracy score for Logistic Regression on new data is 0.3
Precision score for Logistic Regression on new data is 0.6666666666666666
Recall score for Logistic Regression on new data is 0.25
```

Construire des modèles avec des données standardisées

Mais lorsque nous montrons à nos modèles des données absolument nouvelles, quelque chose d'étrange se produit.

```
print('RandomForest predicts the following winners:')
print(rf_model.predict(df_new))
print(validation_sample.Winner.values)
print('Actual fight results')
```

```
RandomForest predicts the following winners:
[1 0 0 0 1 1 1 1 1 0]
[1 1 0 1 1 1 1 1 1 0]
Actual fight results
```

```
print('Accuracy score for RandomForest on new data is', ac
print('Precision score for RandomForest on new data is', p
print('Recall score for RandomForest on new data is', reca
```

```
Accuracy score for RandomForest on new data is 0.8
Precision score for RandomForest on new data is 1.0
Recall score for RandomForest on new data is 0.75
```

Construire des modèles avec des données standardisées

Mais lorsque nous montrons à nos modèles des données absolument nouvelles, quelque chose d'étrange se produit.

```
1 print('SVC predicts the following winners:')
2 print(svc_model.predict(df_new))
3 print(validation_sample.Winner.values)
4 print('Actual fight results')
```

```
SVC predicts the following winners:
[1 1 1 1 1 1 1 1 1 1]
[1 1 0 1 1 1 1 1 1 0]
Actual fight results
```

```
1 print('Accuracy score for SVC on new data is', accu
2 print('Precision score for SVC on new data is', pre
3 print('Recall score for SVC on new data is', recall
```

```
Accuracy score for SVC on new data is 0.8
Precision score for SVC on new data is 0.8
Recall score for SVC on new data is 1.0
```

Conclusions sur les modèles avec des données standardisées

- aucun des 3 modèles les plus performants ne semble bon.
- les 3 sont inconsistants, de 0,0 à 0,9 de précision. La moyenne se situe à 0,5.
- les modèles actuels ne nous permettent pas de les utiliser pour prédire les combats. Nous devrions essayer d'entraîner les mêmes modèles sur des données originales, non standardisées.

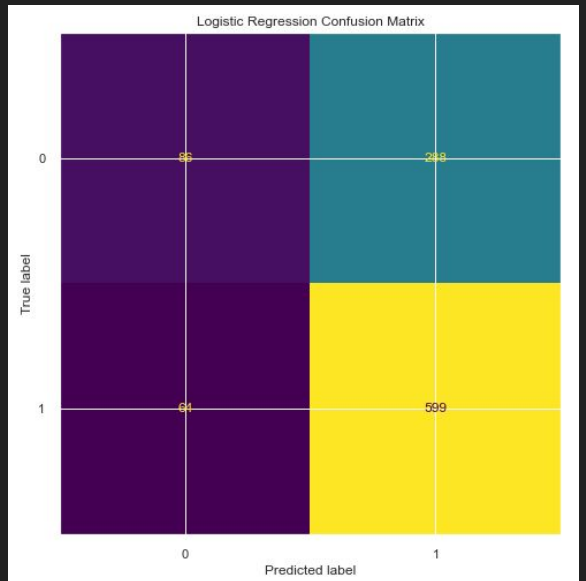
Construire des modèles avec des données originales

Ensuite, nous décidons d'entraîner nos modèles sur des données originales, non standardisées.

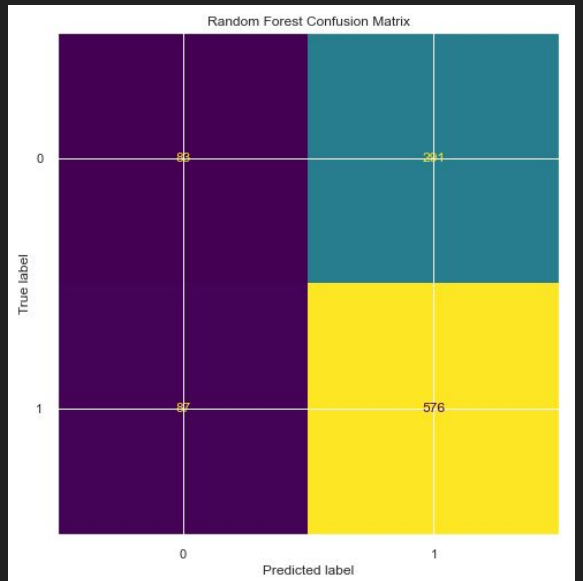
```
# diviser le cadre de données en données de formation et de test
# split dataframe into train and test data

X = df.drop('Winner', axis=1)
y = df['Winner']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=TEST_SIZE, stratify=y,
```

Construire des modèles avec des données originales

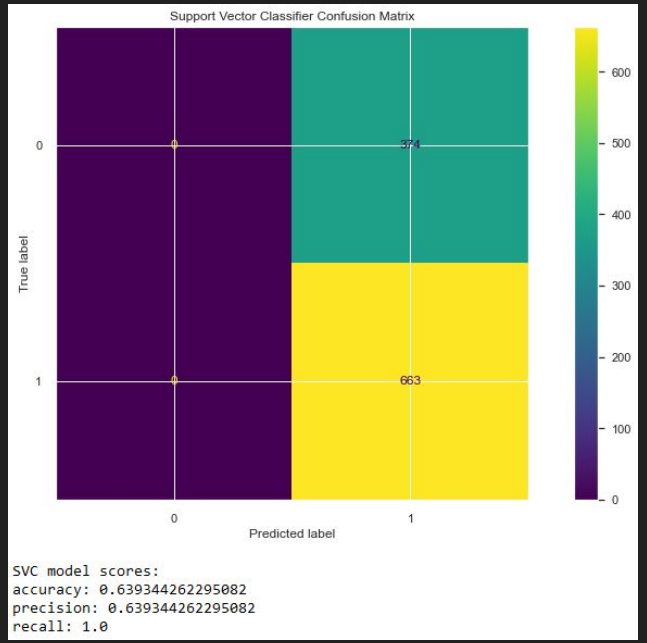
Une fois encore, les modèles les plus performants sont la régression logistique, la forêt aléatoire et le SVC.



Logistic Regression model scores:
accuracy: 0.660559305689489
precision: 0.6753100338218715
recall: 0.9034690799396682



Random Forest model scores:
accuracy: 0.6354869816779171
precision: 0.6643598615916955
recall: 0.8687782805429864



SVC model scores:
accuracy: 0.639344262295082
precision: 0.639344262295082
recall: 1.0

Construire des modèles avec des données originales

Une fois encore, les modèles les plus performants sont la régression logistique, la forêt aléatoire et le SVC.

```
print('Logistic Regression predicts the following winners:')
print(lr_model.predict(df_new))
print(validation_sample.Winner.values)
print('Actual fight results')
```

```
Logistic Regression predicts the following winners:
[1 1 0 1 1 1 1 1 1 1]
[1 1 0 1 1 1 1 1 1 0]
Actual fight results
```

```
print('Accuracy score for Logistic Regression on new data is', accurac
print('Precision score for Logistic Regression on new data is', precis
print('Recall score for Logistic Regression on new data is', recall_sc
```

```
Accuracy score for Logistic Regression on new data is 0.9
Precision score for Logistic Regression on new data is 0.8888888888888888
Recall score for Logistic Regression on new data is 1.0
```

Construire des modèles avec des données originales

Une fois encore, les modèles les plus performants sont la régression logistique, la forêt aléatoire et le SVC.

```
1 print('RandomForest predicts the following winners:')
2 print(rf_model.predict(df_new))
3 print(validation_sample.Winner.values)
4 print('Actual fight results')
```

```
RandomForest predicts the following winners:
[1 1 0 1 1 1 1 1 1 1]
[1 1 0 1 1 1 1 1 1 0]
Actual fight results
```

```
1 print('Accuracy score for RandomForest on new data is', accurac
2 print('Precision score for RandomForest on new data is', precis
3 print('Recall score for RandomForest on new data is', recall_sc
```

```
Accuracy score for RandomForest on new data is 0.9
Precision score for RandomForest on new data is 0.8888888888888888
Recall score for RandomForest on new data is 1.0
```

Construire des modèles avec des données originales

Une fois encore, les modèles les plus performants sont la régression logistique, la forêt aléatoire et le SVC.

```
1 print('SVC predicts the following winners:')
2 print(svc_model.predict(df_new))
3 print(validation_sample.Winner.values)
4 print('Actual fight results')
```

```
SVC predicts the following winners:
[1 1 1 1 1 1 1 1 1 1]
[1 1 0 1 1 1 1 1 1 0]
Actual fight results
```

```
1 print('Accuracy score for SVC on new data is', accuracy_sc
2 print('Precision score for SVC on new data is', precision_
3 print('Recall score for SVC on new data is', recall_score(
```

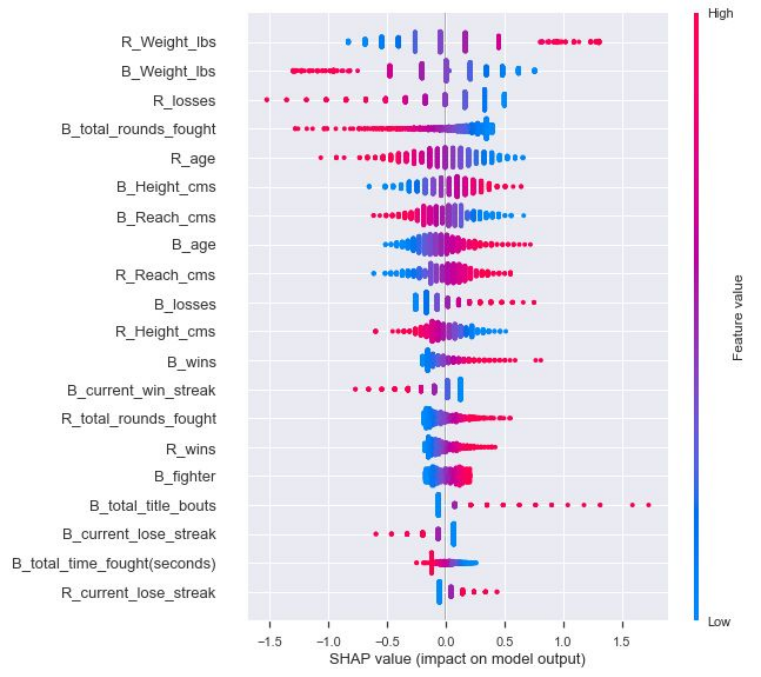
```
Accuracy score for SVC on new data is 0.8
Precision score for SVC on new data is 0.8
Recall score for SVC on new data is 1.0
```

Conclusions sur les modèles avec des données non standardisées

- la différence entre les scores est très faible, mais aussi incohérente. Les scores des modèles basés sur les données originales sont parfois légèrement supérieurs, parfois légèrement inférieurs.
- Il semble que l'entraînement sur les données originales donne des prédictions plus cohérentes.
- nous pourrions avoir besoin d'évaluer la capacité à voir quelle caractéristique (colonne) a plus de valeur pour le modèle, par rapport aux autres.

Poids des caractéristiques individuelles

Afin d'essayer de trouver quelles caractéristiques sont plus importantes que d'autres pour notre modèle, nous allons utiliser la bibliothèque SHAP.



Les caractéristiques sont triées en fonction de leur impact, du plus fort au plus faible. C'est l'axe Y.
L'axe X correspond aux valeurs SHAP. Chaque point est une observation distincte.
Le code couleur aide à comprendre la signification de la caractéristique correspondante. Le bleu est faible, le rouge est élevé (ceci n'est pas du tout lié à notre variable dépendante du gagnant rouge-bleu).

Par exemple :

- si la caractéristique B_total_rounds_fought est plus élevée, il est plus probable que le combattant bleu gagne.
- si R_total_title_bouts est plus élevé, le combattant rouge a de fortes chances de gagner.
- si B_reach_cms devient élevé, le combattant bleu a plus de chances de gagner ; s'il devient faible, le combattant rouge a plus de chances de gagner.

Poids des caractéristiques individuelles

Nous pouvons comparer les valeurs de la bibliothèque SHAP et les coefficients d'importance du modèle lui-même.

```
# créer un tableau de coefficients de caractéristiques à partir de LogReg lui-même
# create a table of feature coefficients from LogReg itself

feature_imp = pd.DataFrame()
feature_imp['feature'] = X.columns
feature_imp['importance'] = lr.coef_[0]
feature_imp = feature_imp.sort_values(by='importance', ascending=False)

feature_imp.head(20)
```

	feature	importance
0	title_bout	0.256582
9	B_total_title_bouts	0.137469
24	R_Stance	0.133655
17	R_current_lose_streak	0.097682
6	B_losses	0.091556
15	B_age	0.047394
10	B_wins	0.043731
22	R_total_title_bouts	0.037863
18	R_current_win_streak	0.029962
23	R_wins	0.029913

Conclusions générales après l'analyse effectuée

Étonnamment, le modèle le plus performant en termes de score s'est avéré être la régression logistique. Il est suivi de peu par Random Forest et SVC. Cependant, les résultats des trois modèles les plus performants sont incohérents. Cela ne répond pas à notre objectif d'avoir une précision de 65-70%. En gardant cela à l'esprit, afin d'utiliser notre (nos) modèle(s), nous devrions travailler davantage au perfectionnement de nos modèles. Sinon, l'utilisation de nos modèles existants serait très risquée.

Améliorations possibles pour ce projet :

- envisager de donner plus de poids aux données récentes, car elles devraient mieux refléter les combats d'aujourd'hui que les données de 1993.
- essayer d'utiliser toutes les colonnes/fonctionnalités existantes. Nous pouvons essayer de déduire divers paramètres historiques pour chaque combattant qui pourraient aider aux prédictions.
- égaliser l'échantillon de validation pour avoir 50% de gagnants rouges et 50% de gagnants bleus.
- essayer d'autres types de modèles (XGBoost, etc).
- construire une meilleure représentation des prédictions (code de couleur, nom du combattant, etc.).
- construire une meilleure façon de montrer la confiance du modèle dans une prédiction spécifique.
- penser à présenter à l'utilisateur un ensemble de facteurs les plus précieux pour la prédiction.
- créer une interface conviviale (déploiement sur le web).

Conclusions générales après l'analyse effectuée

Surprisingly, the best performing model by score turned out to be Logistic Regression. Only slightly behind it is Random Forest and SVC. However, the results of all 3 top performing models are inconsistent. This does not meet our goal of having 65-70% accuracy. With this in mind, in order to use our model(s), we should put in more work towards perfecting our models. Otherwise using our existing models would be very risky.

Possible improvements for this project:

- look into giving more weight to recent data, since it should reflect modern day fights better than data from 1993.
- try and make use of all existing columns/features. We can try and deduce various historical parameters for each fighter that might help with predictions.
- equalize validation sample to have 50% Red and 50% Blue winners.
- try other types of models (XGBoost, etc).
- build a better representation of predictions (color coding, name of the fighter, etc).
- build a better way of showing how confident the model is in one specific prediction.
- think about presenting a user with a set of most valuable factors for prediction.
- create a user-friendly interface (web-deployment).