Title: F1 Race Winner Predictor

Introduction

This is a classification problem where we will predict the position/rank/winner in the race based on certain input features.
We will start off with a relatively simpler 'Logistic Regression' model to set up a baseline, we will introduce increasingly complex models and compare the accuracy with the previous models to decide the best model/algorithm for this.

Phase 1 Review:

In phase 1, we cleaned the data and performed EDA to explore, understand and analyse the data. In phase 1, we found that Starting Position of the driver plays a crucial role in predicting the winner. We also observed that one particular driver generally dominates in a season(year) with wins. In phase we are using this information and applying classification algorithms to predict the winner of the race.

Why GridSearchCV ?

- It systematically tests the combinations of the hyperparameters values to identify the best performing model.
- It automates the tunning, uses k fold cross validation to assess the model performance on training data reducing the risk of overfitting.
- Optimizes for a specific metric to focus on the desired performance.

Algorithms:

Baseline: A dummy classifier that randomly decides a winner has an accuracy of 1/25 = 0.04 = 4%

We decide a new baseline using a weaker model, and then use much more complex models to fit our data and capture non-linear relations.
The baseline is the one against which we compare our results to justify the new model's performance and whether investing in deploying the model is worthwhile.

1. **Logistic Regression**

We first load the data from the csv file and sort it by date.

We select Driver POS Standings, Drive Race Points, Laps, Team Race Points, Team Champ Points, Driver Champ Points and Driver Champ Positions as the features. We kept last 2 years of data for testing and used the remaining data for training. Means we will use the previous year data and predict the winners of 2021 and 2022. We scale the features to make sure all features are on a similar scale.

In the end we print the classification report and winners for 2021 and 2022 sorted according to race number.

```
Model Accuracy: 0.96
Recall for True cases: 0.61

Predicted Winners for the last two years (sorted by y
      season  raceNumber  Winner Code
8321    2021           1          830
8320    2021           1            1
8281    2021           2            1
8361    2021           3          830
8362    2021           3          822
8360    2021           3            1
8381    2021           4          830
8380    2021           4            1
8401    2021           5          830
8460    2021           7          830
8461    2021           7            1
8441    2021           8            1
8440    2021           8          830
8480    2021           9          830
8500    2021          10            1
```

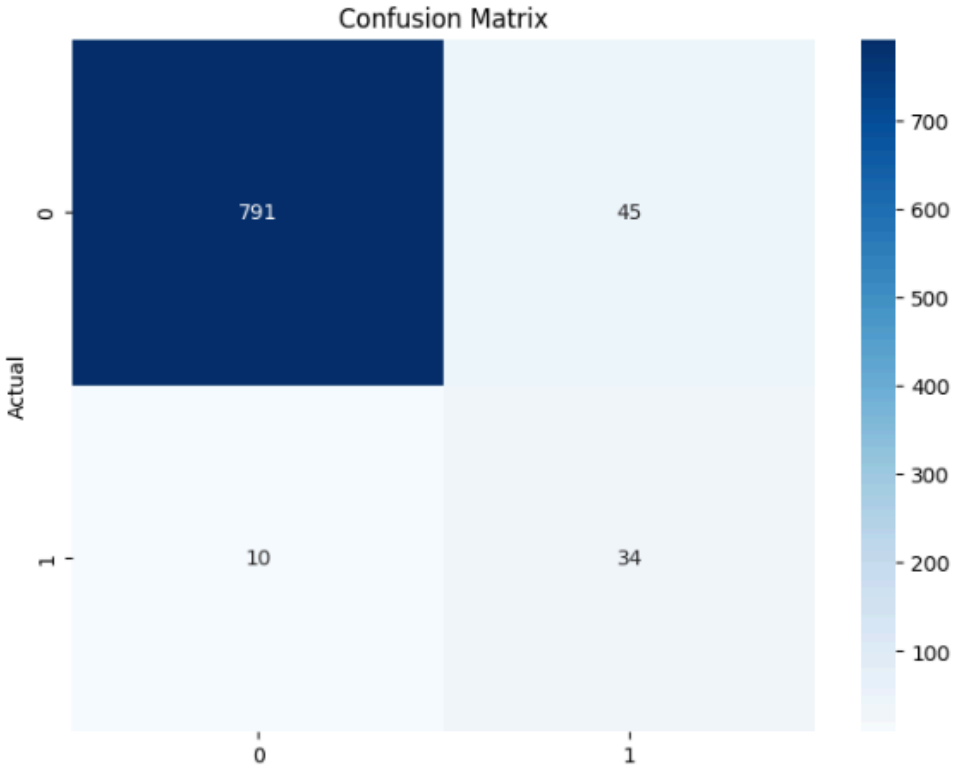
As we can see the model has high accuracy, but the recall is 0.61 means the model performs good at identifying actual winners. Also, in the predictions we can see that for certain races we have multiple winners this is because these racers have probability above the threshold.

So, we do hyperparameter tuning to check if we can achieve a better recall.

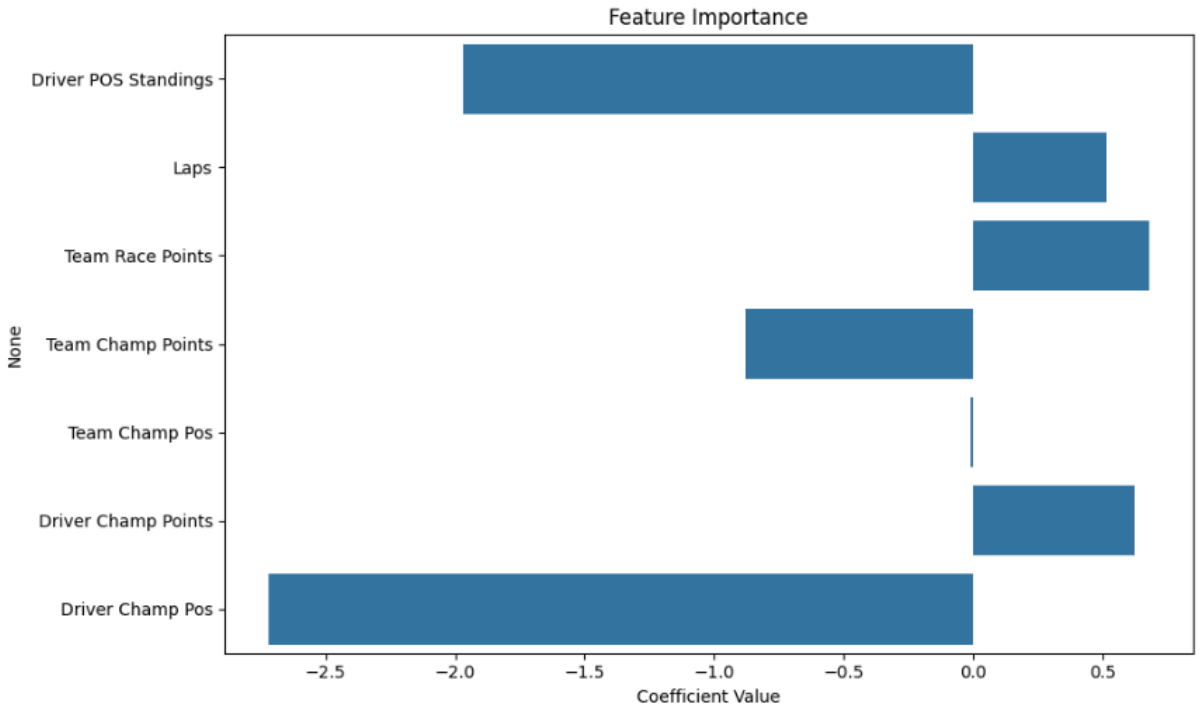
After hyperparameter tuning the recall has improved from 0.61 to 0.77 meaning that our model has improved at identifying winners.

```
Model Accuracy: 0.94
Recall for True cases: 0.77

Best Parameters:
{'classifier__C': 0.1, 'classifier__class_weight': {Tr

Predicted Winners for the last two years (sorted by ye
      season  raceNumber  Winner Code
8321    2021           1          830
8322    2021           1          822
8324    2021           1          815
8320    2021           1            1
8281    2021           2            1
...      ...         ...          ...
9125    2022          21          830
9123    2022          21          844
9121    2022          21            1
9120    2022          21          847
9140    2022          22          830
```

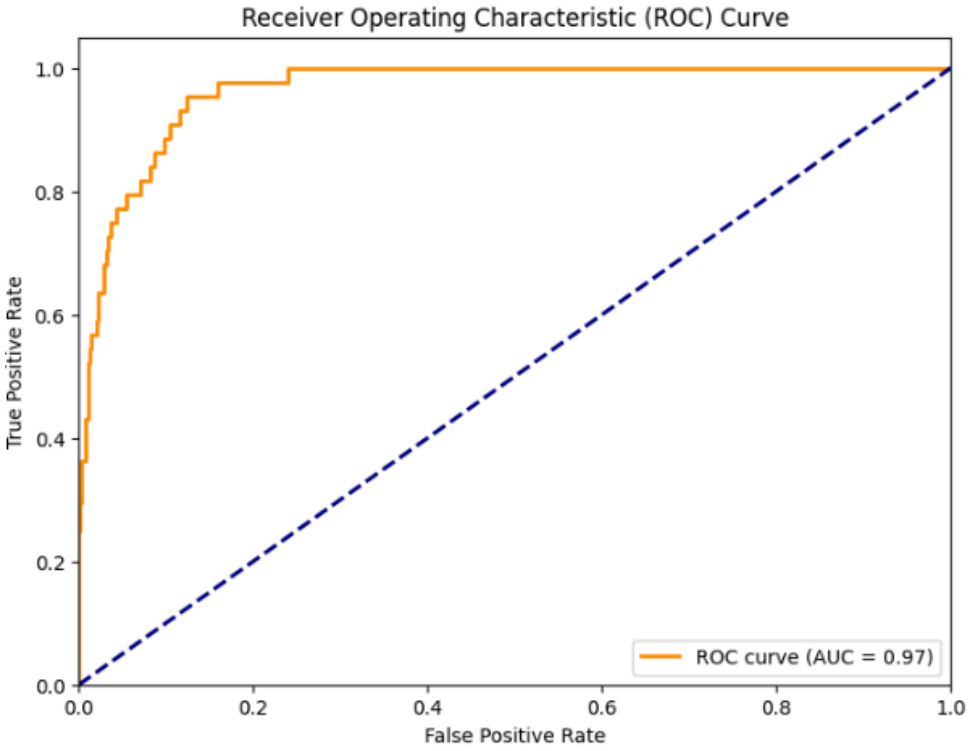
Usually in a race there's just a single winner per race so we have an imbalance in data as there are nearly 20-25 losers per race and just a single winner. The improvement in recall without affecting accuracy indicates that the model has become better at handling imbalance.



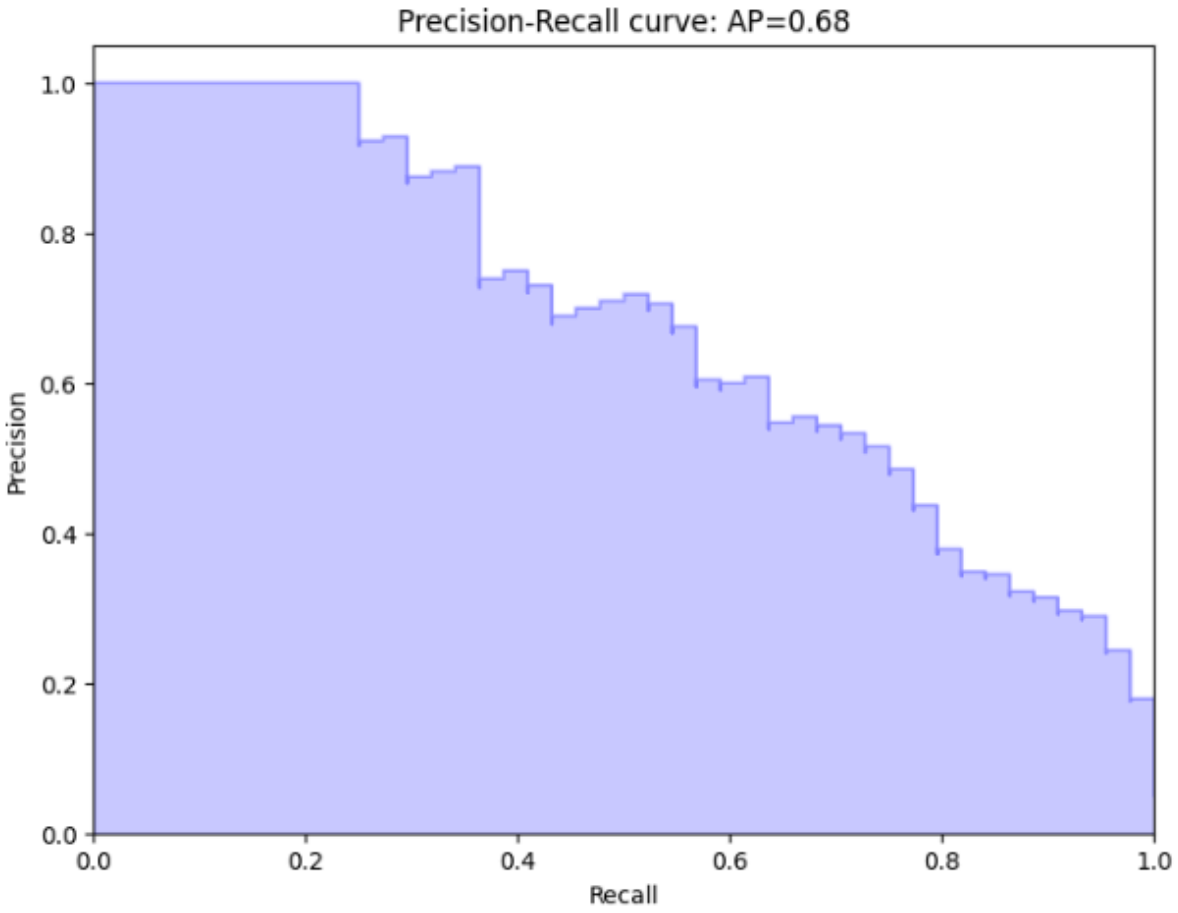
As you can see the model is good at predicting true positives and true negatives.



In this we can see that drive race points have the highest positive coefficient value meaning that the driver's points contribute significantly to winning. Which is true as the driver with the highest points wins.



The ROC curve is close to the top-left corner indicating excellent performance.



The AP value of 0.68 indicates a reasonably good performance.

2. KNN

The KNN model was chosen because its characteristics align well with the requirements and nature of data. As in KNN there are no assumptions about data distribution, as it can take a dataset of any size and can handle high dimensionality data.

We start with loading the dataset (initial_cleaned_winner_data.csv).
Target variable: Winner

```
Index(['raceId', 'season', 'raceNumber', 'Grand Prix', 'Race Date',
       'Driver Code', 'Team Code', 'Driver POS Standings',
       'Driver Race Points', 'Laps', 'Team Race Points', 'Nationality',
       'Team Name', 'Team Country', 'Team Champ Points', 'Team Champ Pos',
       'Team Champ Wins', 'Driver Champ Points', 'Driver Champ Pos',
       'Driver Champ Wins', 'Circuit Name', 'Race Location', 'Race Country',
       'Winner'],
      dtype='object')
```

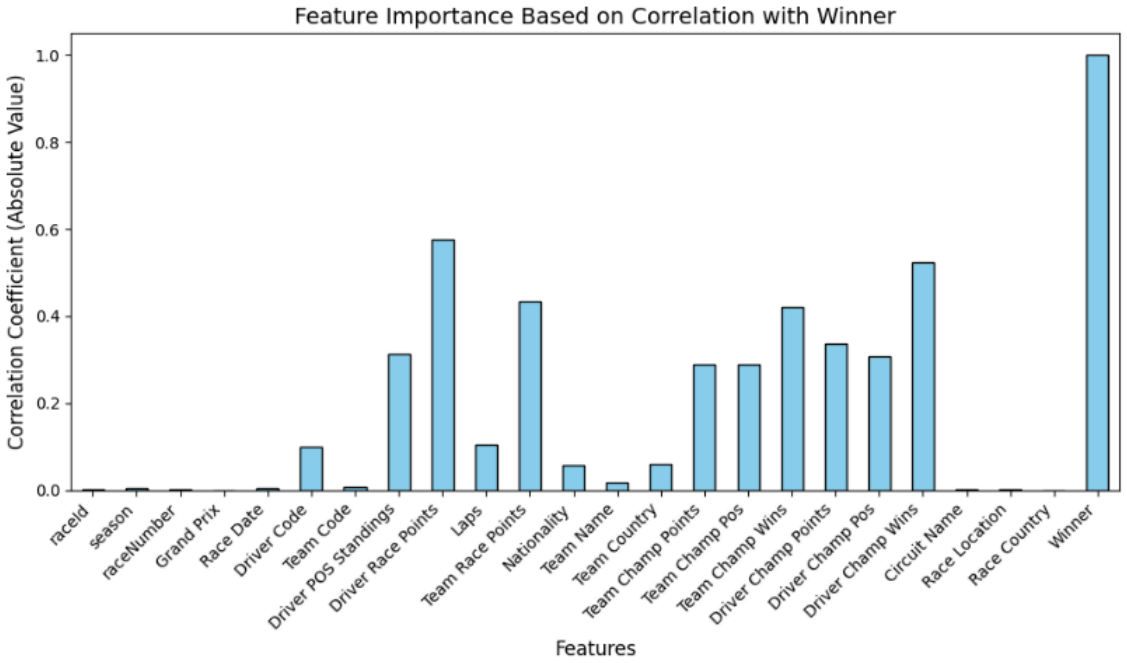
Above are the available features out of which we have used only the important features using the correlation analysis.

Correlation Analysis:
Here we have computed the correlation matrix to get the most relevant features to target "Winner" which will train the model to get more accurate predictions.
`selected_features = correlation_target[correlation_target > 0.4].index.tolist()`

- We have set the threshold here to 0.4, as it belongs to the moderate threshold range which calculates moderately correlated features.
- When we use a low threshold there is risk of using irrelevant features which may lead to more wrong predictions.
- And when we use a high threshold we may exclude potential but weakly correlated features.

KNN relies on distance calculations, so irrelevant or weak features can affect the effect of meaningful features. A moderate threshold will be useful to get relevant predictors removing noises.

Below is the bar graph showing the importance of each feature on correlation with the target Winner.



So from above we get,
Final Selected Features
['Driver POS Standings', 'Team Race Points', 'Team Champ Wins', 'Driver Champ Points', 'Driver Champ Pos', 'Driver Champ Wins', 'Winner']

Later on we split the data into training and test set.
Training set : 80%
Test set : 20%

Hyperparameter Tuning:
Before training the model we are doing hyperparameter tuning, as it maximizes the model performance by systematically testing parameters combinations.

In the code we have used GridSearchCV to tune the hyperparameters.
Below are the Key hyperparameters tuned:
n_neighbors: The number of neighbors considered when making predictions
weights: We have “weights = distance ”, which indicates that closer neighbors have more influence and “weights = uniform” when all neighbors contribute equally.
metric: Distance metric for calculating the similarity.
E.g. Euclidean, Manhattan, Minkowski

Later we get best parameters,
Best Parameters: {'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}

Using the above parameters we do the predictions and then calculate recall and accuracy of the mode.

Before hyperparameter tuning we get below results:

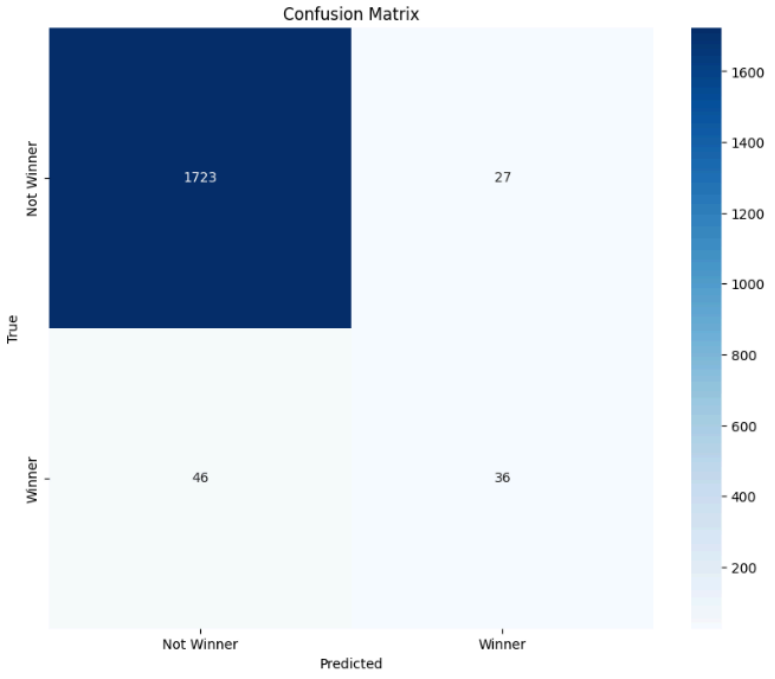
```
Recall (Winner Class): 0.5
Accuracy: 0.9683406113537117
```

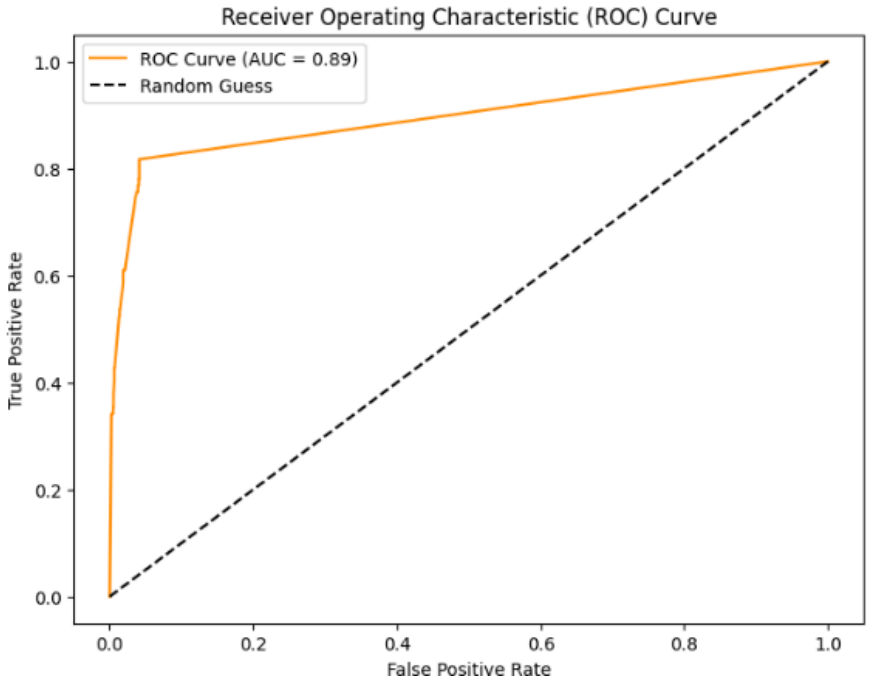
And after hyperparameter tuning we get,

```
Recall (Winner Class): 0.5365853658536586
Accuracy: 0.9656113537117904
```

Hence after parameter tuning recall has improved from 0.5 to 0.53 meaning that our model has improved at identifying winners.

Then we plot the confusion matrix. The confusion matrix visualizes true positives, false positives, false negatives, and true negatives.





The ROC curve plot evaluates the performance of a K-Nearest Neighbors (KNN) classifier. It shows the relation between the true positive rate and false positive rate.

The area under the curve 0.89 which indicates the strong ability of the model to distinguish between the classes.

3. **XGBoost:**

XGBoost was chosen because it is a boosting model which is considered to be one of the most powerful models.
It provides parallel boosting and is the leading machine learning library/algorithm for most regression, classification problems.

Using a simple XGBoost model (with default values) gives us the following results:

```
Initial Model Classification Report:
              precision    recall  f1-score   support

       False       0.99      0.99      0.99      1750
        True       0.72      0.72      0.72        82

    accuracy                           0.97      1832
   macro avg       0.85      0.85      0.85      1832
weighted avg       0.97      0.97      0.97      1832
```

A recall of 0.72 which is better than most algorithms (even after hyperparameter tuning).

Optimizing hyperparameters:

- **n_estimators**: The number of trees (boosting rounds) in the model; higher values increase complexity but may risk overfitting.
- **max_depth**: The maximum depth of each tree
- **learning_rate**: The step size at each iteration to reduce errors; smaller values make training slower but more precise.
- **subsample**: The fraction of the training data used for each boosting round; prevents overfitting by introducing randomness.
- **colsample_bytree**: The fraction of features randomly sampled for each tree; controls model diversity and reduces overfitting.
-

Running GridSearch on these variables gives us the following result:

```
Best Model Classification Report:
              precision    recall  f1-score

       False       0.99      0.99      0.99
        True       0.81      0.78      0.80

    accuracy                           0.98
   macro avg       0.90      0.89      0.89
weighted avg       0.98      0.98      0.98
```

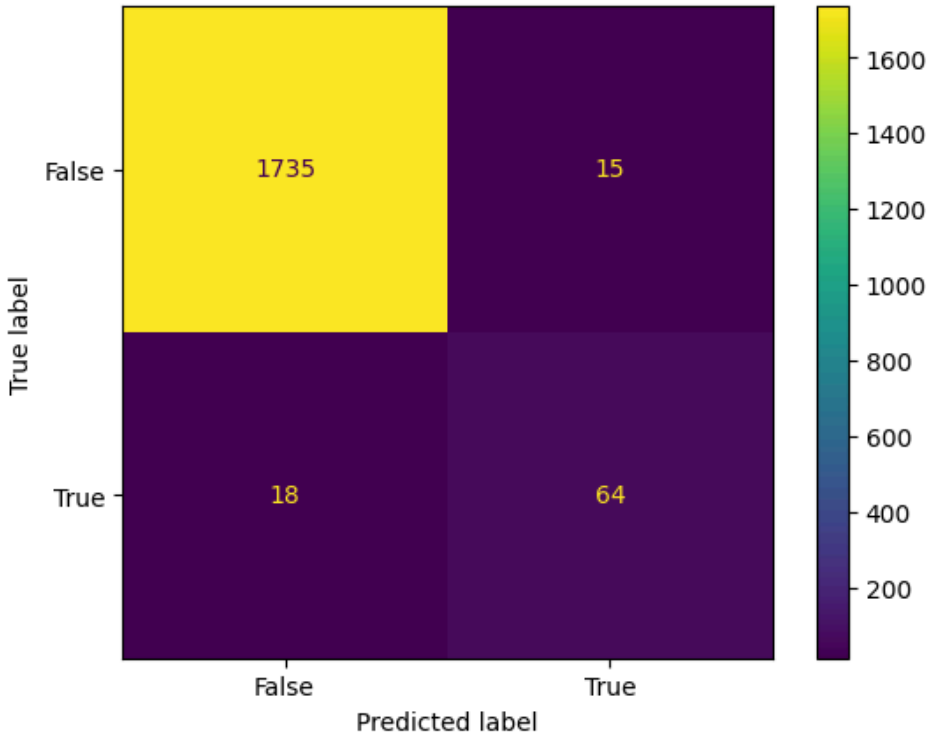
<u>i.e. a Recall of 0.78</u>

With these hyperparameters:

'colsample_bytree': 1.0

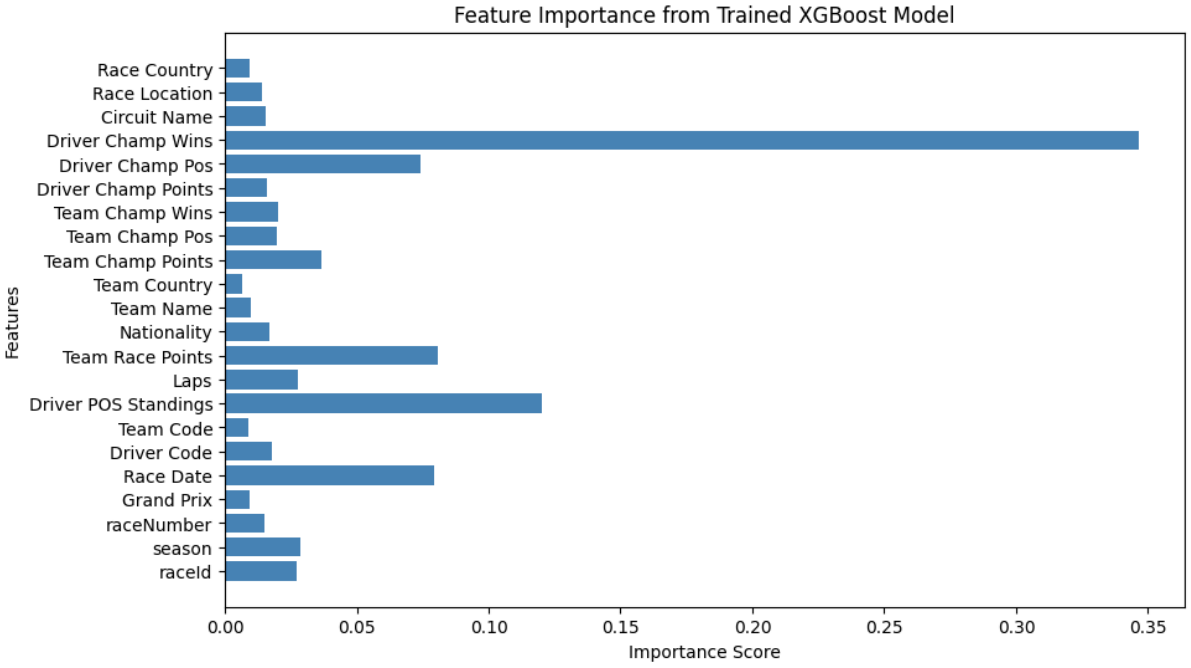
'learning_rate': 0.05
'max_depth': 3
'n_estimators': 300
'subsample': 1.0

The confusion matrix of the best model can be seen below:



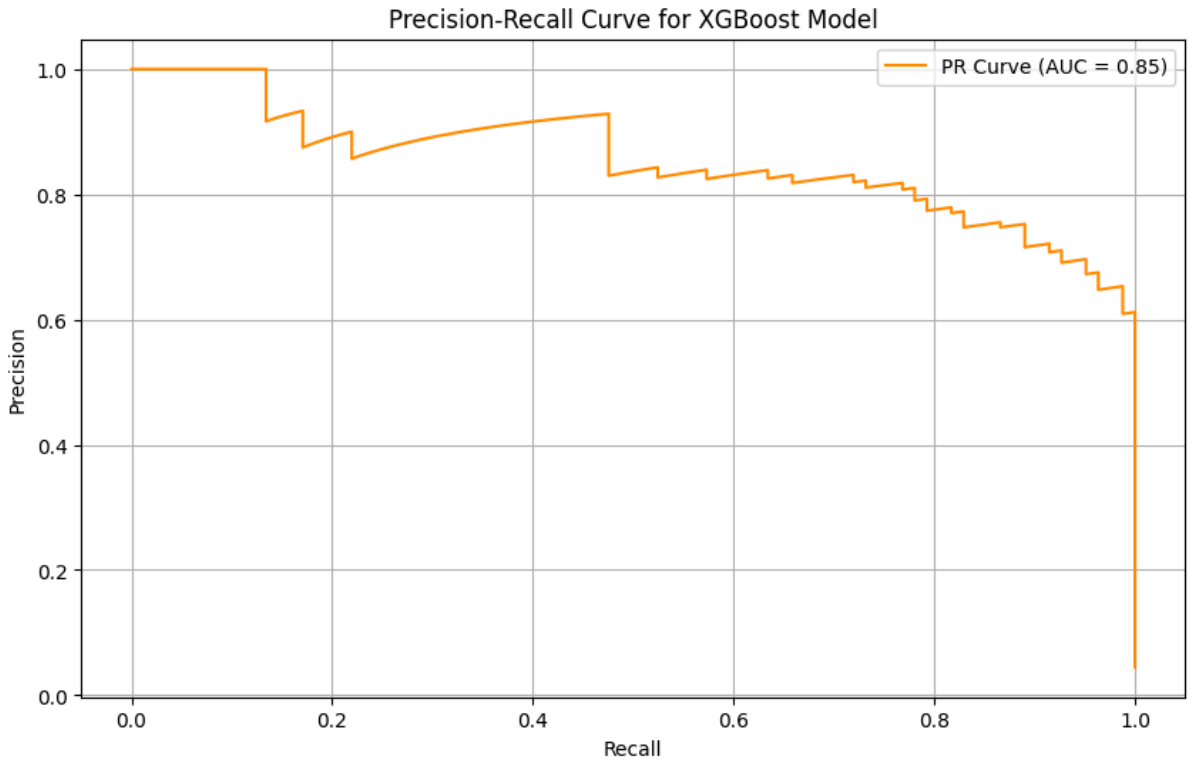
with minimum number of misclassifications

The feature importance graph shows that ‘Driver Champ Wins’ is the most important feature:

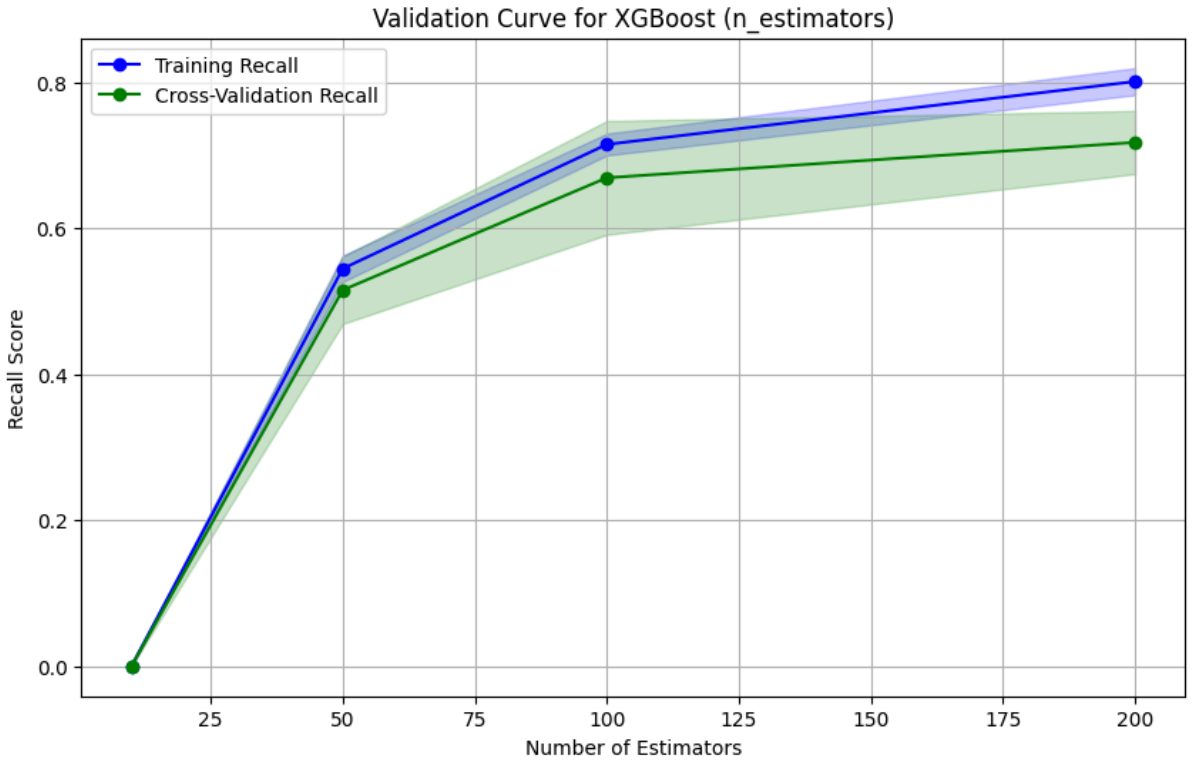


whereas ‘Team Country’ is the least

The Precision-Recall Curve shows that the model is a good fit (with AUC = 0.85):



We also plot a n_estimators vs Recall score graph to verify that more estimators increases the recall needed:



4. **Neural Networks**

Class 0 -> Not Winner

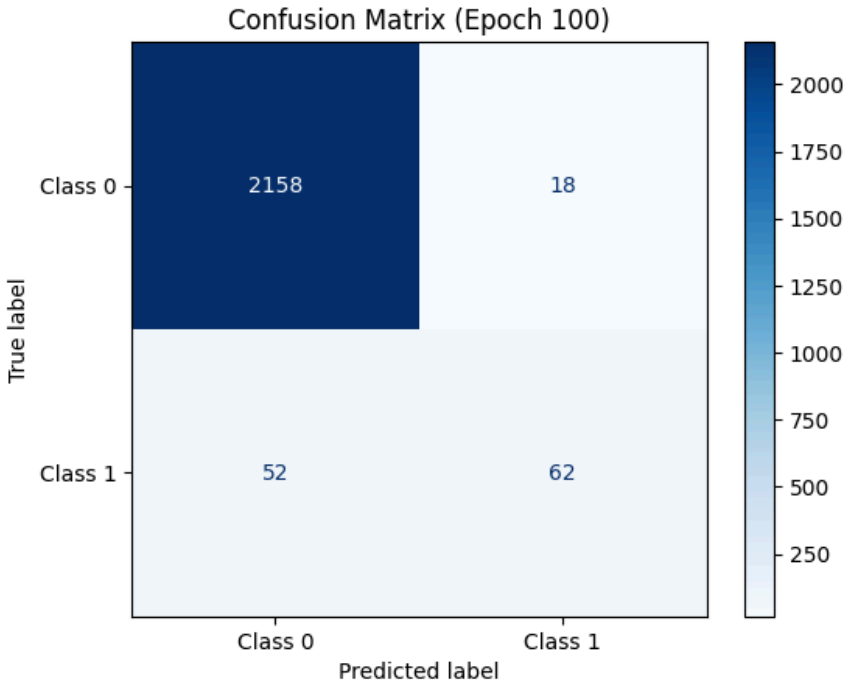
Class 1 -> Winner

a. Reason for Selection

- We have selected this algorithm to train our model due to their ability to model complex, non-linear relationships in the data.
- They are suitable for high-dimensional datasets and can automatically learn relevant patterns, making them ideal for the Winner prediction task.

b. Baseline Accuracy/ Recall

- Using a simple model with 2 hidden layers we get a recall of 54.39%



c. Adam optimizer

- The Adam optimizer was selected due to its ability to adjust the learning rate which enables the model to converge quicker and demonstrate good performance on high dimensional data.

d. Hyperparameter Tuning

- Since the previous network is a simple 2-hidden layer network, we add more hidden layers to enable it to capture deeper connections.

e. Best Accuracy

- Using 5 hidden layers allows us to increase the recall from 54% to 69% for the same number of iterations. However the recall plateaus there and doesnt seem to increase any more.

f. Final findings

- Even though the accuracy and recall are far better than our original baselines, a neural network with BCELoss is not the best model (even with deep connections) for this problems

5. **Random Forest:**

Next model we used is the Random Forest Classification model. We chose a random forest model because it works by combining the predictions of multiple decision trees to make more stable and accurate predictions. It is also useful as it avoids overfitting and provides Feature Importance.

We selected ‘season','Driver Code','Team Code', 'Driver POS Standings', 'Driver Champ Points', 'Team Champ Points' features to train the data.

We also converted the ‘Winner’ Column (target) to binary format (0 or 1) to declutter the target field and simplify the prediction.

While splitting data, data was stratified according to ‘seasons’.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42,
                                                    stratify=data['season'])
```

To find the best hyperparameters we use RandomSearchCV() method and set the cross validation to 5, scoring metric as recall and number of iteration to 200.

```
# Find best hyperparameters
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=param_grid, n_iter=200,
                               cv=5, verbose=2, random_state=42, n_jobs=-1, scoring='recall')


rf_random.fit(X_train, y_train)

#training model on best hyper parameyters
best_params = rf_random.best_params_
model = RandomForestClassifier(random_state=42, class_weight='balanced', **best_params)
model.fit(X_train, y_train)

y_pred1 = model.predict(X_test)
y_pred_binary = (y_pred1 > 0.5).astype(int)  # convert predictions to binary
```

We retrieved the best params from the hyperparameters and trained the RandomForestClassifier with them and set class_weight=balanced to control the imbalance in the dataset. Since there is only 1 winner per race and 19-24 losers, the ratio is highly unbalanced.

After performing the prediction, to convert it into binary, we set a threshold of 0.5. If the prediction is above 0.5, we set it to 1. (shown in above screenshot)

Below is the result - Best Parameters, Recall and Accuracy:
Recall: 0.36
Accuracy: 0.95

```
Fitting 5 folds for each of 200 candidates, totalling 1000 fits
Best hyperparameters:  {'n_estimators': 400, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 20}
Recall for True cases: 0.36
```

```
Accuracy:  0.9536026200873362
```

Even though the model's accuracy is high, the recall is 0.36 which is poor. Thus, the hyper parameters require tuning.

We figured out that the reason for the model poor performance in True Recall is that the data has been stratified by season.

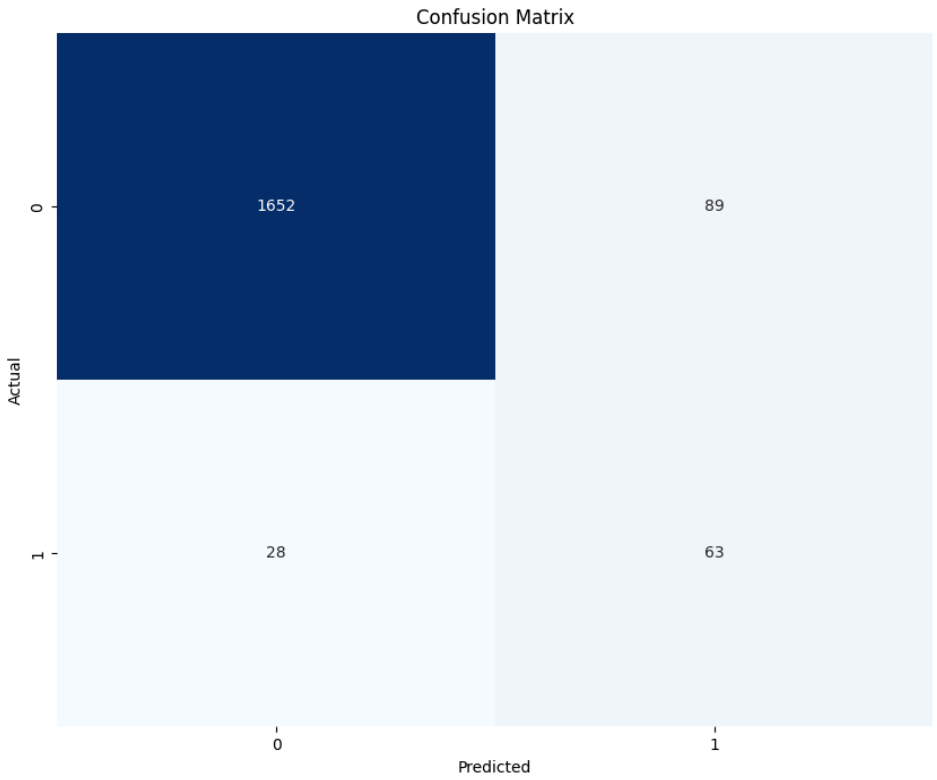
```
# split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42,
                                                    shuffle=False)
```

```
[52]
#setting hyper parameters
param_grid = {'n_estimators': [100, 200],
              'max_features': ['sqrt', 'log2'],
              'max_depth': [10, 20, None],
              'min_samples_split': [ 5, 10],
              'min_samples_leaf': [2, 4]}

rf = RandomForestClassifier(random_state=42)
```

After tuning the data split by not shuffling it and hyper parameters, the output of the code is below:
Recall: 0.69
Accuracy: 0.93

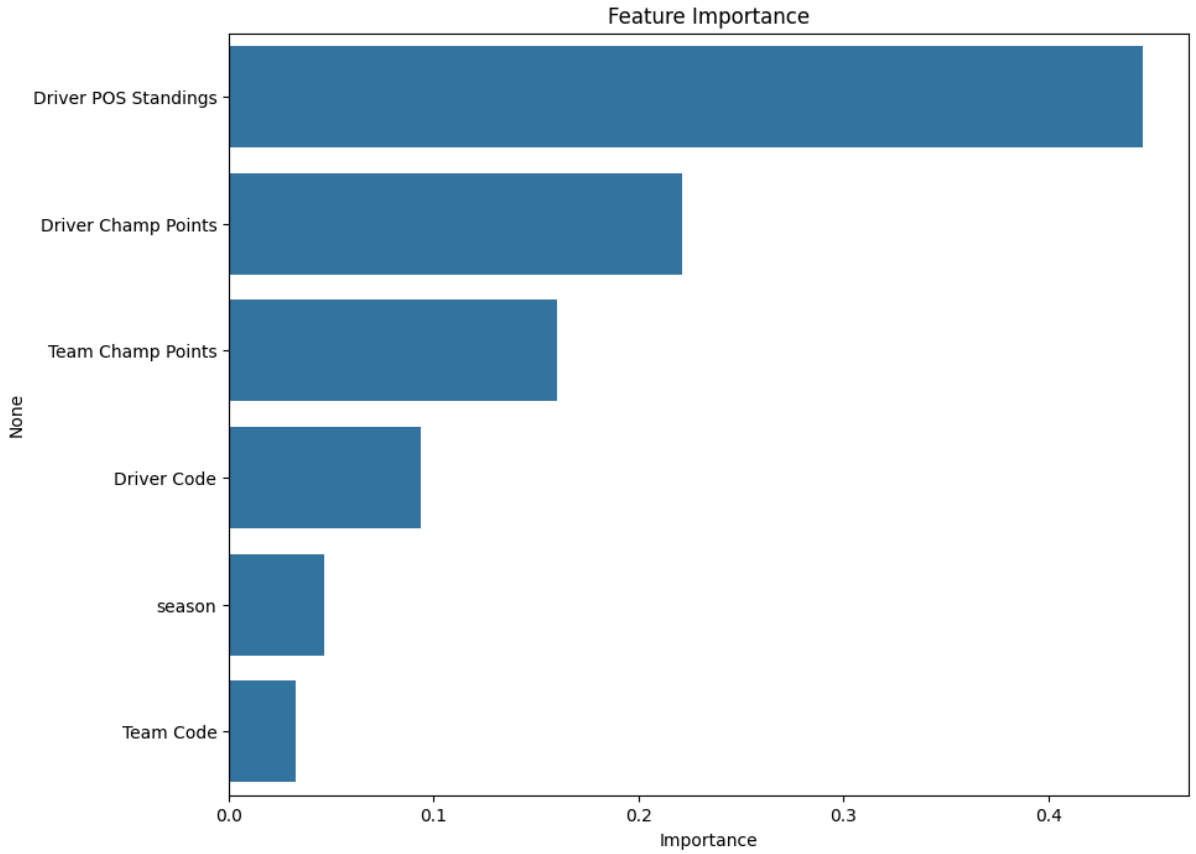
```
Best hyperparameters:  {'n_estimators': 100, 'min_samples_split': 5, 'min_samples_leaf': 4, 'max_features': 'sqrt', 'max_depth': 10}
Recall for True cases: 0.69
Accuracy:  0.9361353711790393
```

Confusion Matrix;



There are 63 correct winners predicted by the model.

The feature importance graph shows that ‘Driver POS Standing’ i.e Drivers starting grid position’ is the most important feature followed by ‘Driver’s Championship Points’



Findings:
Best Model: XGBoost with .78 recall.
Worst Model: KNN with .53 recall.