


Dataset used:  Arjun\_Bowlers\_dataset

## Outliers found:

```
5]: values = ['UK Group 1 Bowler 2', 'London Academy: Bowler 26', ' London Academy: Bowler 53',  
            'Non HD 1', 'HD 3', 'HD 11', 'London Academy: Bowler 8', 'London Academy: Bowler 36']
```

```
5]: train_df = train_df[train_df.athlete_name.isin(values) == False]
```

*Values* is an array of all the outliers identified by the names of the bowlers in the dataset. I found outliers in each graph by comparing each weight with bowling speed.

## Removed Null Values:

```
[91]: train_df['Bowling Speed (Kmph)'] = np.log1p(train_df['Bowling Speed (Kmph)'])
```

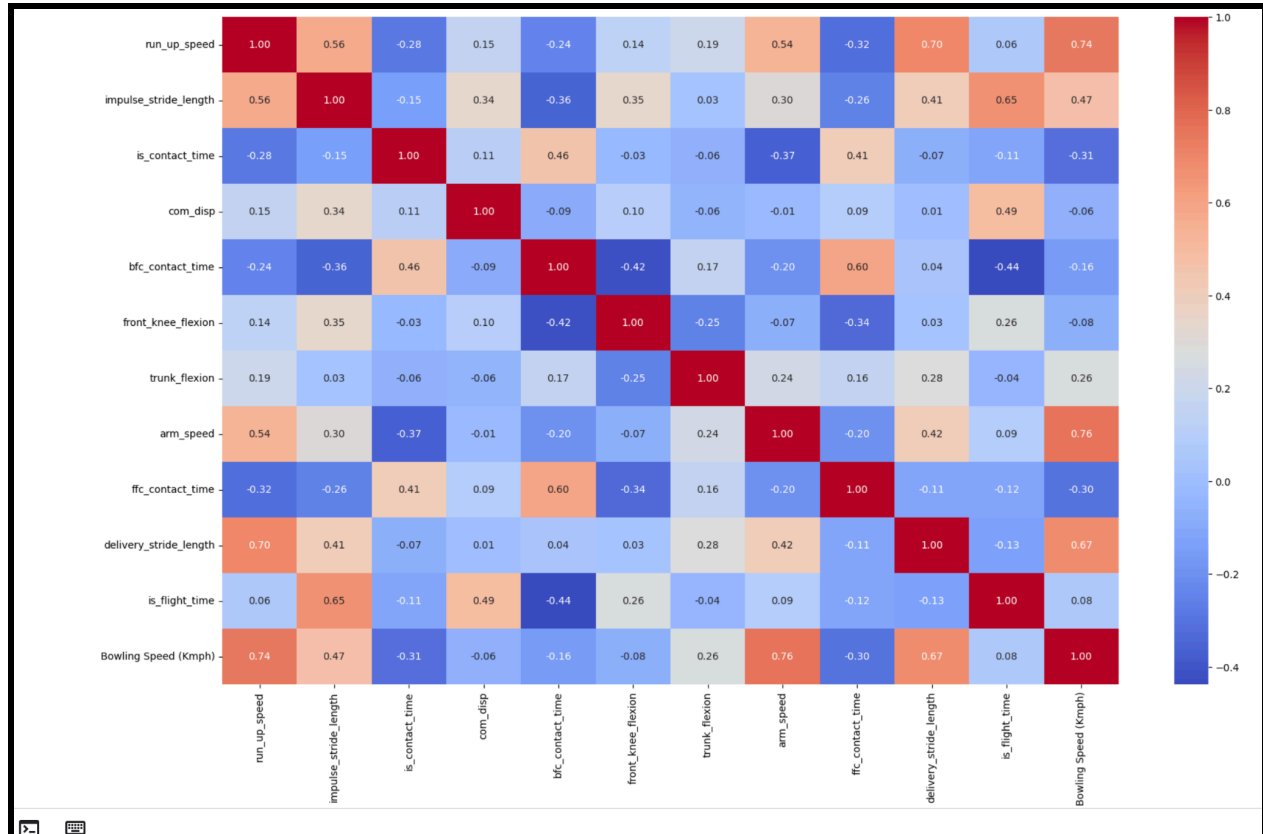
This line of code removes all rows with at least one feature having no value.

```
[96]: X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
[96... ((130, 11), (33, 11), (130,), (33,))
```

Out of 180 total rows, 17 of them were removed, 9 of them being Null rows and 8 of them being obvious outliers.

*Reference Correlations (NOT THE ACTUAL CORRELATION VALUES)*



The figure above represents a correlation heatmap of the entire dataset towards each metric. As shown above, our prediction correlations should be a bit similar to these heatmap findings, as we will use this heatmap as a reference comparing it to each model's predictions.

This map will be referenced during the Regression Analysis process below.

## Individual Patterns (each variable vs Bowling Speed):

## Model Testing:

### 1. Linear Regression:

```
#LINEAR REGRESSION MODEL
lr = LinearRegression()
lr.fit(X_train, y_train)
```

```
LinearRegression
```

```
y_pred_lr = lr.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred_lr))
```

0.07929179628145665

R<sup>2</sup> Value (Coefficient of Determination): 0.712, 71.23%  
RMSE Value (Root Mean Squared Error): 0.0793

Linear Regression Equation:

$$y = 3.461 + (0.015)*run\_up\_speed + (0.000)*impulse\_stride\_length + (0.000)*is\_contact\_time + (-0.006)*com\_disp + (0.000)*bfc\_contact\_time + (-0.001)*front\_knee\_flexion + (0.000)*trunk\_flexion + (0.002)*arm\_speed + (-0.000)*ffc\_contact\_time + (0.002)*delivery\_stride\_length + (0.001)*is\_flight\_time$$

```
intercept_lr = lr.intercept_
coefficients_lr = lr.coef_

# Generate the equation
features = X_train.columns # Feature names (assuming X_train is a DataFrame)
equation_lr = f"y = {intercept_lr:.3f}" # Start with the intercept

for feature, coef in zip(features, coefficients_lr):
    equation_lr += f" + ({coef:.3f})*{feature}"

print("Linear Regression Equation:")
print(equation_lr)
```

Linear Regression Equation:  
y = 3.461 + (0.015)\*run\_up\_speed + (0.000)\*impulse\_stride\_length + (0.000)\*is\_contact\_time + (-0.006)\*com\_disp + (0.000)\*bfc\_contact\_time + (-0.001)\*front\_knee\_flexion + (0.000)\*trunk\_flexion + (0.002)\*arm\_speed + (-0.000)\*ffc\_contact\_time + (0.002)\*delivery\_stride\_length + (0.001)\*is\_flight\_time

+ Code + Markdown

## 2. Random Forest Regression:

```
[78]: #Random Forest Regressor
RFR = RandomForestRegressor(random_state=13)
param_grid_RFR = {
    'max_depth': [7, 14, 21],
    'n_estimators': [120, 270, 520],
    'min_samples_split': [3, 5, 10]
}

[79]: rfr_cv = GridSearchCV(RFR, param_grid_RFR, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
rfr_cv.fit(X_train, y_train)

[79]: > GridSearchCV
> estimator: RandomForestRegressor
> RandomForestRegressor

[80]: y_pred_rfr = rfr_cv.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred_rfr))

[80]: 0.09314766149467063

[81]: r2_score(y_test, y_pred_rfr)

[81]: 0.6029948851424121
```

R<sup>2</sup> Value: 0.603, 60.30%  
RMSE Value: 0.093  
Approximate Random Forest Regression Equation:  
y ≈ (0.214)\*run\_up\_speed + (0.033)\*impulse\_stride\_length + (0.014)\*is\_contact\_time + (0.016)\*com\_disp + (0.016)\*bfc\_contact\_time + (0.016)\*front\_knee\_flexion + (0.016)\*trunk\_flexion + (0.476)\*arm\_speed + (0.030)\*ffc\_contact\_time + (0.151)\*delivery\_stride\_length + (0.018)\*is\_flight\_time

Although I could not generate a proper equation for Random Forest, as the model does not allow coefficients, I was able to find the feature importances instead. This helped me determine the key features better.

Details: The parameters I used were Max Depth, N estimators, and minimum samples split. After research, the bigger the maximum depth the more complex the RFR tree becomes, which would possibly lead to overfitting. Any depths above [7, 14, 21] cause a bigger RMSE value, and thus might cause overfitting if any larger. The same goes for the n estimators. I initially started with lower values and increased to the values that give the lowest RMSE values.

Based on this model, the top 5 coefficients are Arm Speed, IS Length, Run-up Speed, Delivery Stride Length, and FFC Contact Time.

### 3. XGB Regressor:

```
1: #XGB Regressor
XGB = XGBRegressor(random_state=13)
param_grid_XGB = {
    'learning_rate': [0.05, 0.1, 0.2],
    'n_estimators': [230], #between 220 and 230
    'max_depth': [4], #4 is good
    'min_child_weight': [1,2,3], # matters whether you remove some of the elements
    'gamma': [0,1,2],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0],
}
xgb_cv = GridSearchCV(XGB, param_grid_XGB, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)

2: xgb_cv.fit(X_train, y_train)

3: GridSearchCV
  estimator: XGBRegressor
    XGBRegressor

4: y_pred_xgb = xgb_cv.predict(X_test)
   np.sqrt(mean_squared_error(y_test, y_pred_xgb))

0.08881841463306206
```

R<sup>2</sup> Value: 0.639, 63.90%  
 RMSE Value: 0.8882  
 Approximate XGB Regression Equation:  

$$y \approx (0.311) * \text{run\_up\_speed} + (0.040) * \text{impulse\_stride\_length} + (0.029) * \text{is\_contact\_time} + (0.037) * \text{com\_disp} + (0.016) * \text{bfc\_contact\_time} + (0.032) * \text{front\_knee\_flexion} + (0.021) * \text{trunk\_flexion} + (0.309) * \text{arm\_speed} + (0.039) * \text{ffc\_contact\_time} + (0.150) * \text{delivery\_stride\_length} + (0.015) * \text{is\_flight\_time}$$

```
5: r2_score(y_test, y_pred_xgb)

[17...] 0.6390407101942326
```

Based on my research, XGB Regressor would be a better version of Random Forest Regression. XGBoost deals with underrepresented areas and correlations better than the Random Forest Regression does, and it can be seen in the equation. Each correlation value is more balanced and makes more sense compared to the initial training data correlations. I finetuned all of the hyperparameters to make the RMSE value from this model the best possible.

I learned from XGBoost hyperparameter tuning that the Learning Rate should typically be 0.3 or less, going progressively lower until the data is the best fit. I put Gamma as 0 because ideally, we would want to split all of the data points and not leave any behind. Since Gamma represents the minimum loss reduction to make a split in data, having it set to 0 would make sure that all of the data should get split, adding complexity to the model.

Based on my findings for XGBoost, the top 5 key features are Run Up Speed (31.1%), Arm Speed (30.9%), Delivery Stride Length (15%), Impulse Stride Length (4%), FFC Contact Time (3.9%). This is very similar to the initial correlation results from the heatmap representing the entire dataset.

#### 4. Ridge Regression

```
param_grid_ridge = {  
    'alpha': [955],  
    'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag']  
}  
ridge_cv = GridSearchCV(ridge, param_grid_ridge, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)  
ridge_cv.fit(X_train, y_train)
```

R<sup>2</sup> Value: 0.734, 73.45%

RMSE Value: 0.0762

```
> GridSearchCV  
  estimator: Ridge  
    Ridge
```

```
y_pred_ridge = ridge_cv.predict(X_test)  
np.sqrt(mean_squared_error(y_test, y_pred_ridge))
```

0.07617948539895607

```
r2_score(y_test, y_pred_ridge)
```

0.734461034590099

Ridge Regression Equation:

$$y = 3.585 + (0.00480) \cdot \text{run\_up\_speed} + (0.00055) \cdot \text{impulse\_stride\_length} + (0.00012) \cdot \text{is\_contact\_time} + (-0.00294) \cdot \text{com\_disp} + (0.00017) \cdot \text{bfc\_contact\_time} + (-0.00119) \cdot \text{front\_knee\_flexion} + (0.00065) \cdot \text{trunk\_flexion} + (0.00228) \cdot \text{arm\_speed} + (-0.00034) \cdot \text{ffc\_contact\_time} + (0.00263) \cdot \text{delivery\_stride\_length} + (0.00023) \cdot \text{is\_flight\_time}$$

The alpha value was initially set to 0, but as I kept increasing it, it was most optimal at around 955, just below 1000. Normally, such a high alpha value is not recommended but perhaps it would be such that the fit would be more flexible and not overfitting. The only con with this model is that if the high alpha value would lead to underfitting by any chance (however it isn't seen compared to the high r<sup>2</sup> score and RMSE value).

Based on this equation, the top 5 key features are Run Up Speed (by a landslide), COM Displacement, Delivery Stride Length, Arm Speed, and Front Knee Flexion.

## Feature Importances (Ridge):

	Feature	Coefficient	Importance
0	run_up_speed	0.004795	0.301229
3	com_disp	-0.002944	0.184954
9	delivery_stride_length	0.002631	0.165296
7	arm_speed	0.002285	0.143508
5	front_knee_flexion	-0.001192	0.074854
6	trunk_flexion	0.000651	0.040918
1	impulse_stride_length	0.000555	0.034850
8	ffc_contact_time	-0.000341	0.021414
10	is_flight_time	0.000234	0.014723
4	bfc_contact_time	0.000169	0.010623
2	is_contact_time	0.000121	0.007631

## 5. Lasso Regression

```
param_grid_lasso = {
    'alpha': [0.05]
}
```

```
lasso_cv = GridSearchCV(lasso, param_grid_lasso, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
```

```
lasso_cv.fit(X_train, y_train)
```

```
> GridSearchCV
> estimator: Lasso
  > Lasso
```

```
y_pred_lasso = lasso_cv.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred_lasso))
```

```
0.0763662057015352
```

```
r2_score(y_test, y_pred_lasso)
```

```
0.7331577367460049
```

R<sup>2</sup> Value: 0.7332, 73.32%

RMSE Value: 0.07637

Lasso Regression Equation:

$$y = 3.670 + (0.00044) \cdot \text{run\_up\_speed} + (0.00072) \cdot \text{impulse\_stride\_length} + (0.00000) \cdot \text{is\_contact\_time} + (-0.00085) \cdot \text{com\_disp} + (0.00007) \cdot \text{bfc\_contact\_time} + (-0.00108) \cdot \text{front\_knee\_flexion} + (0.00012) \cdot \text{trunk\_flexion} + (0.00246) \cdot \text{arm\_speed} + (-0.00033) \cdot \text{ffc\_contact\_time} + (0.00275) \cdot \text{delivery\_stride\_length} + (0.00000) \cdot \text{is\_flight\_time}$$

I found this to be the opposite of Ridge, in that the model was best functioning with a very low alpha value which is typically not a

recommended parameter value (just like Ridge in general). If it went any higher or lower than 0.05 for alpha, then the RMSE and R<sup>2</sup> values would drastically change. Both Ridge and Lasso models have their strengths and weaknesses but the very high alpha value in Ridge and the very low value for Lasso make it a slightly weak model despite the high RMSE and R<sup>2</sup> values in each model.

Based on this equation, the top 5 key features are Delivery Stride Length, Arm Speed, Front Knee Flexion, COM Displacement, and Impulse Stride Length.

## Feature Importances (Lasso):

	Feature	Coefficient	Importance
9	delivery_stride_length	0.002750	0.312148
7	arm_speed	0.002457	0.278886
5	front_knee_flexion	-0.001076	0.122143
3	com_disp	-0.000847	0.096171
1	impulse_stride_length	0.000724	0.082209
0	run_up_speed	0.000439	0.049797
8	ffc_contact_time	-0.000330	0.037501
6	trunk_flexion	0.000118	0.013408
4	bfc_contact_time	0.000068	0.007737
2	is_contact_time	0.000000	0.000000
10	is_flight_time	0.000000	0.000000

## 6. Gradient Boosting Regressor

```
param_grid_GBR = {
    'max_depth': [12, 15, 20],
    'n_estimators': [230],
    'min_samples_leaf': [10, 25, 50],
    'learning_rate': [0.001, 0.01, 0.1],
    'max_features': [0.01, 0.1, 0.7]
}
```

```
GBR_cv = GridSearchCV(GBR, param_grid_GBR, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
```

```
GBR_cv.fit(X_train, y_train)
```

```
> GridSearchCV
> estimator: GradientBoostingRegressor
  + GradientBoostingRegressor
```

```
y_pred_GBR = GBR_cv.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred_GBR))
```

```
0.09202284338754116
```

```
r2_score(y_test, y_pred_GBR)
```

```
0.6125251791604647
```

R<sup>2</sup> Value: 0.6125, 61.25%

RMSE Value: 0.0920

Approximate Gradient Boosting  
Regression Equation:

$$y \approx (0.21309) \cdot \text{run\_up\_speed} + (0.02883) \cdot \text{impulse\_stride\_length} + (0.01072) \cdot \text{is\_contact\_time} + (0.01773) \cdot \text{com\_disp} + (0.01616) \cdot \text{bfc\_contact\_time} + (0.02332) \cdot \text{front\_knee\_flexion} + (0.01096) \cdot \text{trunk\_flexion} + (0.45795) \cdot \text{arm\_speed} + (0.03386) \cdot \text{ffc\_contact\_time} + (0.17393) \cdot \text{delivery\_stride\_length} + (0.01345) \cdot \text{is\_flight\_time}$$

With Gradient Boosting Regressor, there was a bit of randomness added to each time I ran this model, thus with the same parameters there is more than one possible RMSE value as there is a randomizing layer. This made it a bit difficult to adjust parameters for the better. However, since I could not get anything below 0.0920 for my RMSE value, for now, I kept these parameters to represent the best possible GBR model. I would say XGBoost Regression would work better in the future, as it naturally functions better with bigger datasets.

Based on this equation, the top 5 key features are Arm Speed (45.80%), Run Up Speed (21.31%), Delivery Stride Length (17.39%), FFC Contact Time (3.39%), Impulse Stride Length (2.88%)

## 7. Cat Boost Regressor:

```
cat_cv = GridSearchCV(catboost, param_grid_cat, cv=3, scoring='neg_mean_squared_error', n_jobs=-1)
```

```
cat_cv.fit(X_train, y_train)
```

```
> GridSearchCV
> estimator: CatBoostRegressor
  > CatBoostRegressor
```

```
y_pred_cat = cat_cv.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred_cat))
```

```
0.0853711403719028
```

```
r2_score(y_test, y_pred_cat)
```

```
0.6665164968821647
```

R<sup>2</sup> Value: 0.6665, 66.65%

RMSE Value: 0.0854

Approximate Catboost Regressor

Equation:

$$y \approx (0.204) * \text{run\_up\_speed} + (0.044) * \text{impulse\_stride\_length} + (0.021) * \text{is\_contact\_time} + (0.046) * \text{com\_disp} + (0.023) * \text{bfc\_contact\_time} + (0.048) * \text{front\_knee\_flexion} + (0.025) * \text{trunk\_flexion} + (0.312) * \text{arm\_speed} + (0.056) * \text{ffc\_contact\_time} + (0.198) * \text{delivery\_stride\_length} + (0.023) * \text{is\_flight\_time}$$

This model performs well and all of the hyperparameters make sense. Although it might not be the best RMSE and R<sup>2</sup> value compared to two of the others (Ridge and Lasso), the hyperparameters make sense to the general rules and trends of the CatBoost Regressor.

I learned from this experience that using GridSearchCV would automatically optimize the hyperparameters if you put more values in each parameter. Hence, Grid Search Cross Validation would allow you to put in multiple values for each parameter and select which would work the best, essentially incorporating cross-validation into the model, which is very beneficial.

```
param_grid_cat = {
    'iterations': [100, 500, 1000],
    'depth': [4, 6, 8, 10],
    'learning_rate': [0.01, 0.05, 0.1, 0.3]
}
```

The top 5 key features of the Cat Boost Regressor are Arm Speed (31.2%), Run-Up Speed (20.4%), Delivery Stride Length (19.8%), FFC Contact time (5.6%), and Front Knee Flexion (4.8%).

## 8. Elastic Net Regression:

```
> param_grid_en = {
    'alpha': [0.1, 1.0, 10.0, 100],
    'l1_ratio': [0.1, 0.5, 0.7, 0.9, 1.0]
}

en = ElasticNet()
en_cv = GridSearchCV(estimator=en, param_grid=param_grid_en, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
en_cv.fit(X_train, y_train)
```

```
> GridSearchCV
> estimator: ElasticNet
```



R<sup>2</sup> Value: 0.7274, 72.74%

RMSE Value: 0.07718

Elastic Net Regression Equation:

$y = 3.511 + (0.012) \cdot \text{run\_up\_speed} + (0.000) \cdot \text{impulse\_stride\_length} + (0.000) \cdot \text{is\_contact\_time} + (-0.005) \cdot \text{com\_disp} + (0.000) \cdot \text{bfc\_contact\_time} + (-0.001) \cdot \text{front\_knee\_flexion} + (0.000) \cdot \text{trunk\_flexion} + (0.002) \cdot \text{arm\_speed} + (-0.000) \cdot \text{ffc\_contact\_time} + (0.002) \cdot \text{delivery\_stride\_length} + (0.000) \cdot \text{is\_flight\_time}$

**Feature Importances (Elastic Net):**

	Feature	Coefficient	Importance
0	run_up_speed	0.012103	0.503399
3	com_disp	-0.004551	0.189275
9	delivery_stride_length	0.002346	0.097557
7	arm_speed	0.002080	0.086492
5	front_knee_flexion	-0.001215	0.050545
10	is_flight_time	0.000473	0.019690
6	trunk_flexion	0.000335	0.013926
8	ffc_contact_time	-0.000302	0.012564
4	bfc_contact_time	0.000246	0.010244
1	impulse_stride_length	0.000236	0.009803
2	is_contact_time	0.000156	0.006506

**REASON FOR LOW COEFFICIENT VALUES:** In comparison to the heatmap, all of the coefficients from the models are significantly lower because I inverse-logged the entire output of Bowling Speed to make it so that multiple regression is easier to perform. This is completely normal and was expected throughout this process.

## OVERALL ANALYSIS:

Models from Highest to Lowest  $R^2$ :

1. Ridge (73.45%)
2. Lasso (73.32%)
3. Elastic Net (72.74%)
4. Linear (71.23%)
5. Catboost (66.65%)
6. XGBoost (63.90%)
7. GBR (61.25%)
8. Random Forest Regression (60.30%)

Above are the models ranging from highest to lowest RMSE values. That being said, what I believe to be the model with the most sensible top 5 key features is XGBoost. Catboost is a close 2nd.