RUN PREDICTOR FOR SHAKIB AL HASAN USING *MACHINE* LEARNING MODELS

Fida Rahman

Intro to AI

University of South Florida

Date: 23rd April, 2023

Abstract

This project is about a very popular South-Asian sports called 'Cricket.' It's a sport very similar to baseball. In cricket, the different categories of players are Bowlers, Batsmen and Fielders.

In this project, I will try to use different machine learning models with my own custom-made dataset, to predict the performance of a Bangladeshi player named 'Shakib Al Hasan' who is a Batsman. A batsman performing well is determined by how much 'runs' a batsman would score. The number of runs, a batsman scores depend on few factors such as the grounds where the match is being played, the opposition, the type of bowlers the batsman faces, whether the batsman bats first or second (innings) and some other factors. There are many formats in which this game is played in, and I have chosen the ODI format. In simple words, I will try to predict the runs the batsman will score in an ODI match with some given inputs.

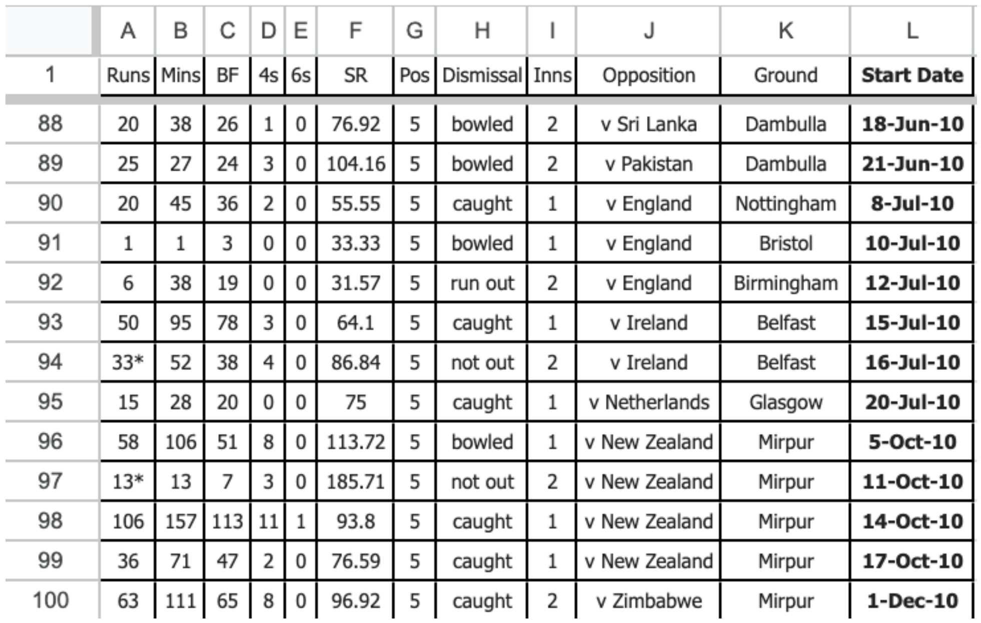
Introduction

In this project, I have tried using different machine learning models like Multiple Linear Regression, Decision Tree, and Random Forest to predict the 'batting performance' of Shakib Al Hasan with my own custom-made dataset. 'Batting performance' means that I will try predicting

As cricket is not a very popular sports worldwide, there was not any proper dataset that I could have used. Not many works have been done in the field of cricket using Artificial Intelligence. There was a work that was inspiring, and I decided to read that paper (https://ieeexplore.ieee.org/document/8628118) and take ideas and tried implementing it in my own project.

Dataset

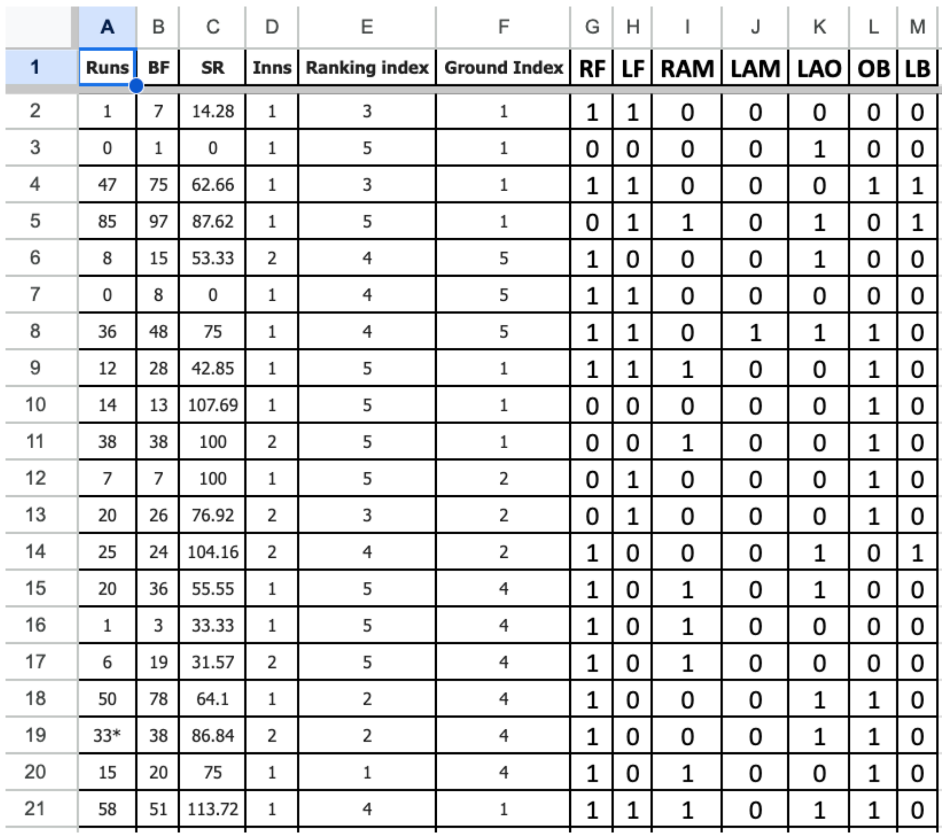
As mentioned before, collecting data was a big part of this project as there were not enough ready-made datasets that I could have used. I took data from a popular cricket website named 'ESPN CricInfo' (https://www.espncricinfo.com/). I collected all the raw data of Shakib Al Hasan for the ODI format from 2010 – 2022 and put it in an excel file. The picture below shows, how the raw data looked like.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Runs	Mins	BF	4s	6s	SR	Pos	Dismissal	Inns	Opposition	Ground	**Start Date**
88	20	38	26	1	0	76.92	5	bowled	2	v Sri Lanka	Dambulla	**18-Jun-10**
89	25	27	24	3	0	104.16	5	bowled	2	v Pakistan	Dambulla	**21-Jun-10**
90	20	45	36	2	0	55.55	5	caught	1	v England	Nottingham	**8-Jul-10**
91	1	1	3	0	0	33.33	5	bowled	1	v England	Bristol	**10-Jul-10**
92	6	38	19	0	0	31.57	5	run out	2	v England	Birmingham	**12-Jul-10**
93	50	95	78	3	0	64.1	5	caught	1	v Ireland	Belfast	**15-Jul-10**
94	33*	52	38	4	0	86.84	5	not out	2	v Ireland	Belfast	**16-Jul-10**
95	15	28	20	0	0	75	5	caught	1	v Netherlands	Glasgow	**20-Jul-10**
96	58	106	51	8	0	113.72	5	bowled	1	v New Zealand	Mirpur	**5-Oct-10**
97	13*	13	7	3	0	185.71	5	not out	2	v New Zealand	Mirpur	**11-Oct-10**
98	106	157	113	11	1	93.8	5	caught	1	v New Zealand	Mirpur	**14-Oct-10**
99	36	71	47	2	0	76.59	5	caught	1	v New Zealand	Mirpur	**17-Oct-10**
100	63	111	65	8	0	96.92	5	caught	2	v Zimbabwe	Mirpur	**1-Dec-10**

Figure: Raw Dataset

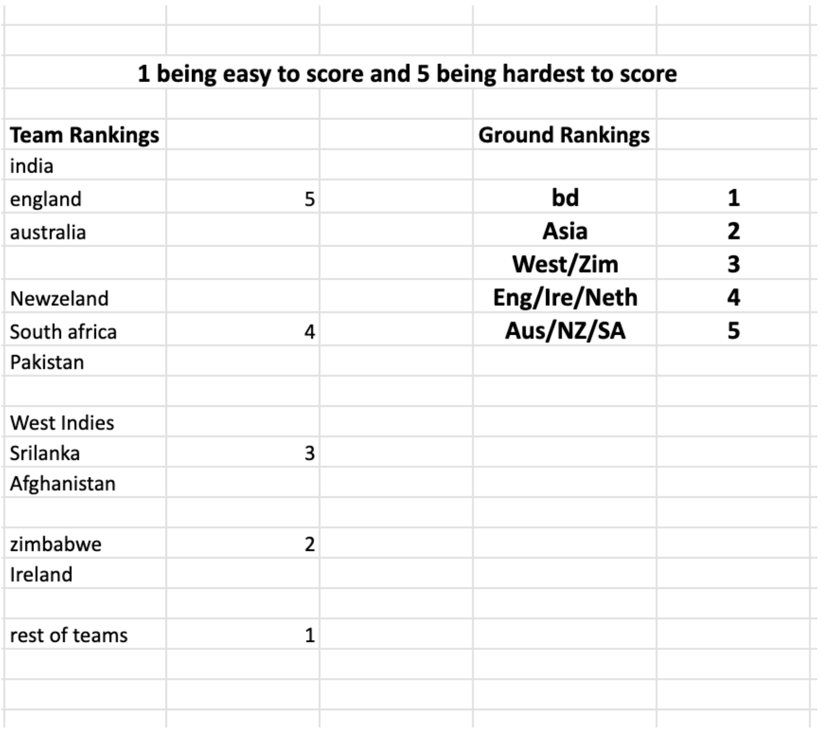
As this data was not sufficient to run my models and to predict the runs accurately, I had to go through the commentary of each of the matches from 2010-2022 to see what sort of bowlers the batsman faced to improve my dataset. The final look of my dataset was something like the picture shown below.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Runs	BF	SR	Inns	Ranking index	Ground Index	RF	LF	RAM	LAM	LAO	OB	LB
2	1	7	14.28	1	3	1	1	1	0	0	0	0	0
3	0	1	0	1	5	1	0	0	0	0	1	0	0
4	47	75	62.66	1	3	1	1	1	0	0	0	1	1
5	85	97	87.62	1	5	1	0	1	1	0	1	0	1
6	8	15	53.33	2	4	5	1	0	0	0	1	0	0
7	0	8	0	1	4	5	1	1	0	0	0	0	0
8	36	48	75	1	4	5	1	1	0	1	1	1	0
9	12	28	42.85	1	5	1	1	1	1	0	0	1	0
10	14	13	107.69	1	5	1	0	0	0	0	0	1	0
11	38	38	100	2	5	1	0	0	1	0	0	1	0
12	7	7	100	1	5	2	0	1	0	0	0	1	0
13	20	26	76.92	2	3	2	0	1	0	0	0	1	0
14	25	24	104.16	2	4	2	1	0	0	0	1	0	1
15	20	36	55.55	1	5	4	1	0	1	0	1	0	0
16	1	3	33.33	1	5	4	1	0	1	0	0	0	0
17	6	19	31.57	2	5	4	1	0	1	0	0	0	0
18	50	78	64.1	1	2	4	1	0	0	0	1	1	0
19	33*	38	86.84	2	2	4	1	0	0	0	1	1	0
20	15	20	75	1	1	4	1	0	1	0	0	1	0
21	58	51	113.72	1	4	1	1	1	1	0	1	1	0

Figure 2: Final Dataset

From the above data, you can see that I have added 9 new columns to make the dataset better to predict the number of runs scored. The columns like RF, LF, RAM, LAM, LAO, OB and LB show the type of bowlers Shakib Al Hasan (Batsman) will face. BF stands for the number of 'Ball Faced' to score runs. SR stands for strike rate which means the rate at which the batsman scored. Inns stands for innings, which means whether a batsman bats first(1) or second(2).



1 being easy to score and 5 being hardest to score

Team Rankings			**Ground Rankings**	
india				
england	5		**bd**	**1**
australia			**Asia**	**2**
			West/Zim	**3**
Newzeland			**Eng/Ire/Neth**	**4**
South africa	4		**Aus/NZ/SA**	**5**
Pakistan				
West Indies				
Srilanka	3			
Afghanistan				
zimbabwe	2			
Ireland				
rest of teams	1			

Figure 3: Index strategy

The picture above shows how the ranking index was done which shows how strong the opposition team is. The ranking index was done by taking the ranking of the different teams into consideration in the last 10-12 years. The ranking index 1 means, the opposition was the easiest and 5 meaning the toughest opposition to face. The ground index is also very similar. It shows the different types of ground where matches were played. As Shakib Al Hasan is a player from Bangladesh(bd), 1 being the easy ground and Australia(Aus) being the toughest ground to score runs.

The picture below shows a heatmap done to show the co-relation of the dataset. It shows strong correlation for BF and SR with respect to runs scored. In some cases, there was negative co-relation which might be due to the shortage of data we have.

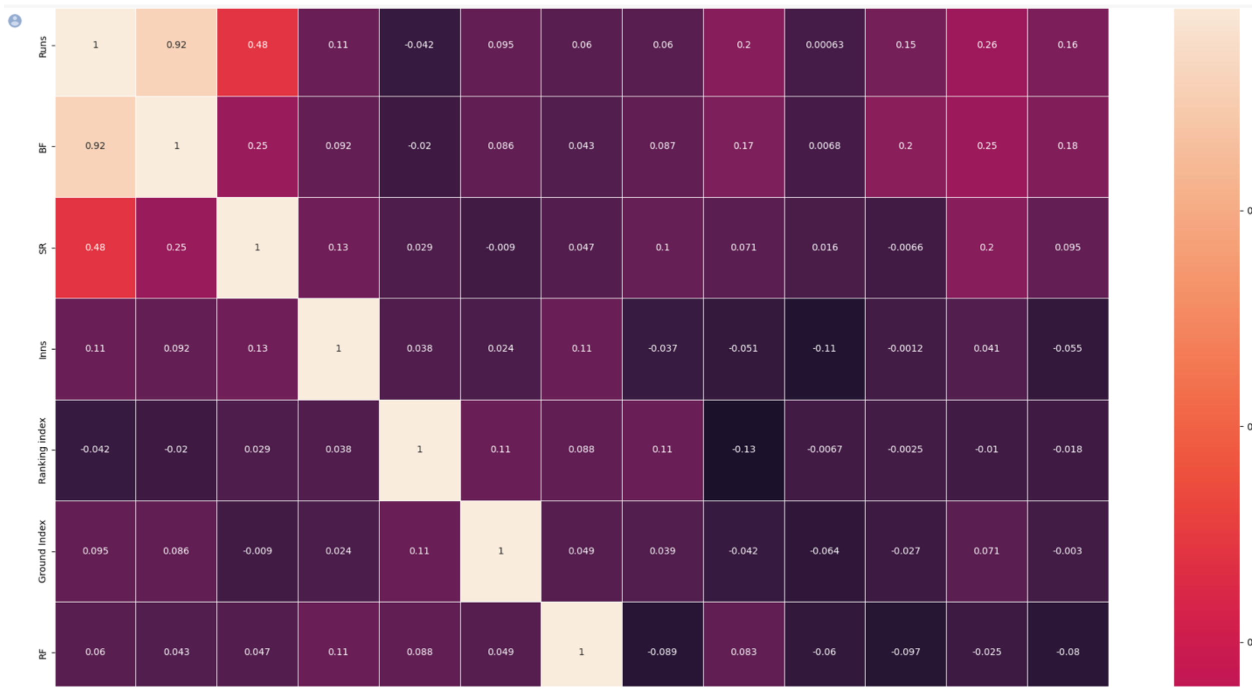
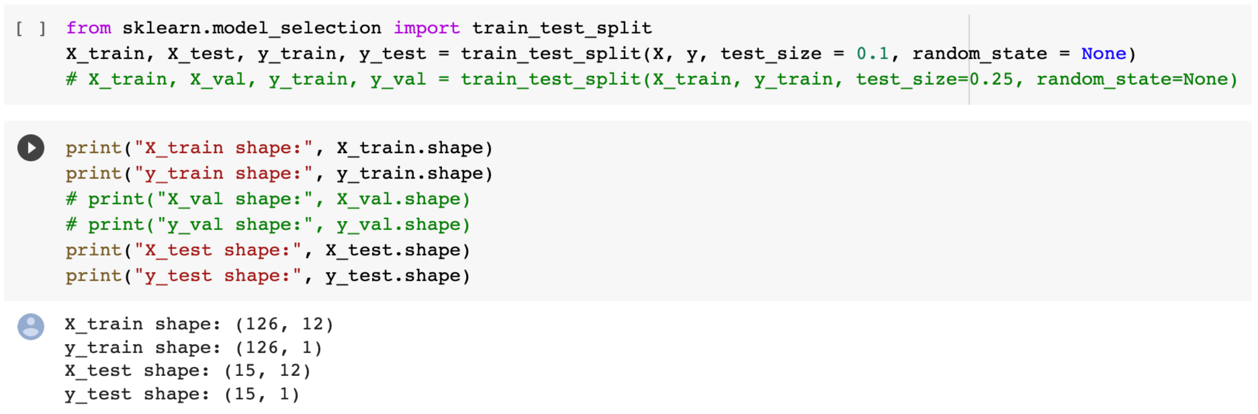


Figure 4: Heat Map

In total we had data from 141 matches played in the last 12 years. Our dependent variable was 'runs' that we tried to predict and the independent variables were BF, SR, Inns, Ranking Index, Ground Index, RF, LF, RAM, LAM, LAO, OB and LB. These were split into train and test datasets as given below.



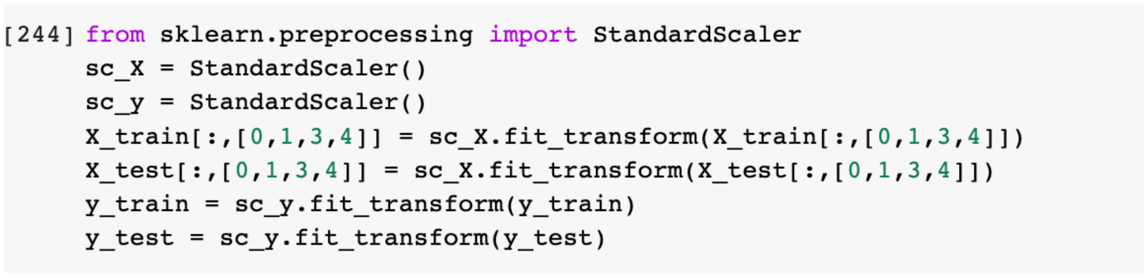
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state = None)
# X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_state=None)
```

```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
# print("X_val shape:", X_val.shape)
# print("y_val shape:", y_val.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (126, 12)
y_train shape: (126, 1)
X_test shape: (15, 12)
y_test shape: (15, 1)
```

Figure 5: Train Test Split

Then the data were normalized using standard scaling. We normalize values to bring them into a common scale, making it easier to compare and analyze data. Normalization also helps to reduce the impact of outliers and improve the accuracy and stability of statistical models. The code below shows how we normalized the data.

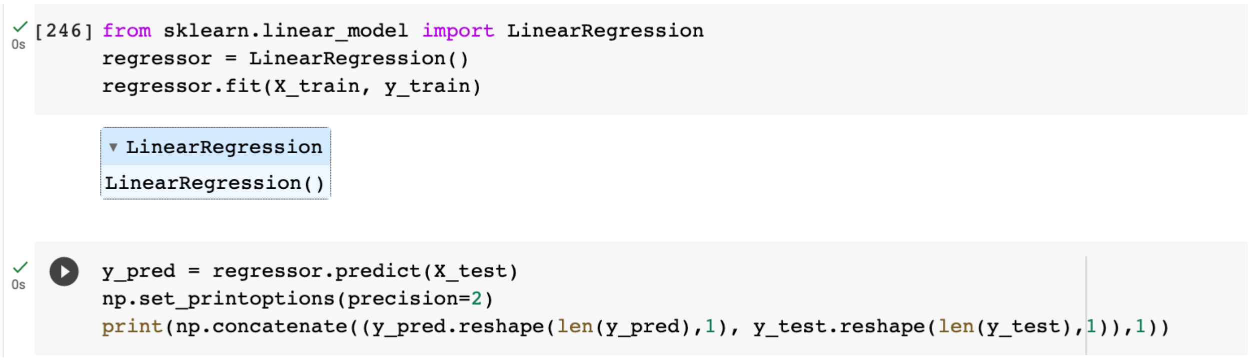


```
[244] from sklearn.preprocessing import StandardScaler
      sc_X = StandardScaler()
      sc_y = StandardScaler()
      X_train[:,[0,1,3,4]] = sc_X.fit_transform(X_train[:,[0,1,3,4]])
      X_test[:,[0,1,3,4]] = sc_X.fit_transform(X_test[:,[0,1,3,4]])
      y_train = sc_y.fit_transform(y_train)
      y_test = sc_y.fit_transform(y_test)
```

Figure 6: Standard Scaling

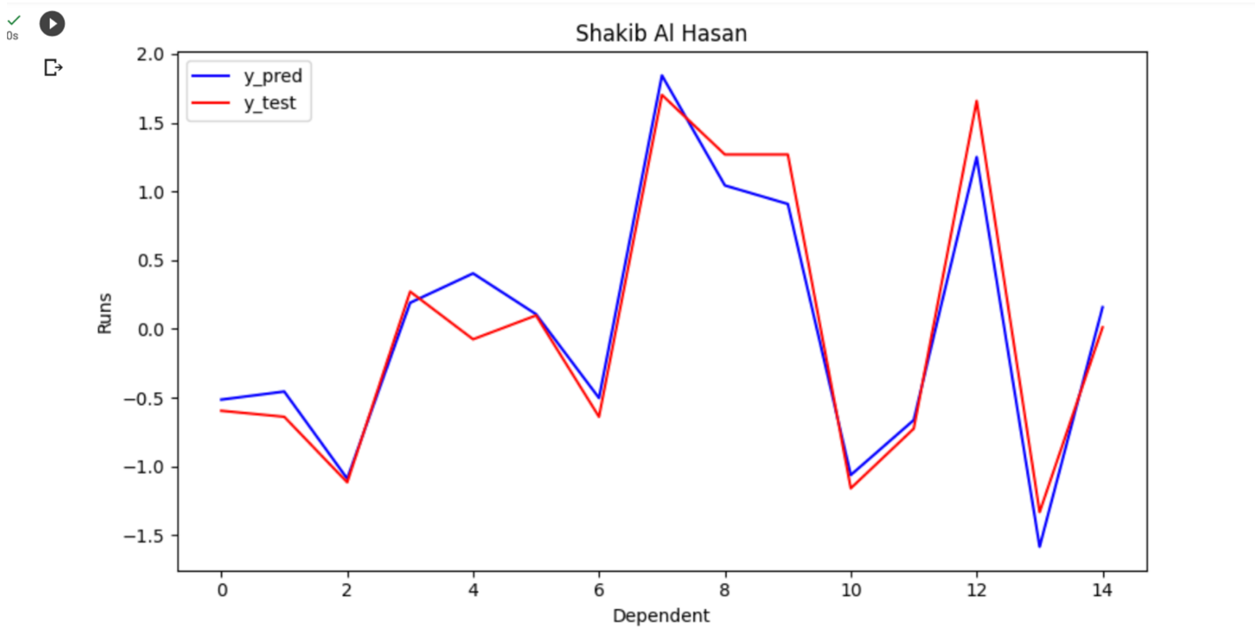
Models:

1. **Multiple Linear Regression**: a technique that uses two or more independent variables to predict the outcome of a dependent variable.



```
[246] from sklearn.linear_model import LinearRegression
      regressor = LinearRegression()
      regressor.fit(X_train, y_train)

      ▾ LinearRegression
      LinearRegression()

      y_pred = regressor.predict(X_test)
      np.set_printoptions(precision=2)
      print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

Figure 7: Code for MLR model

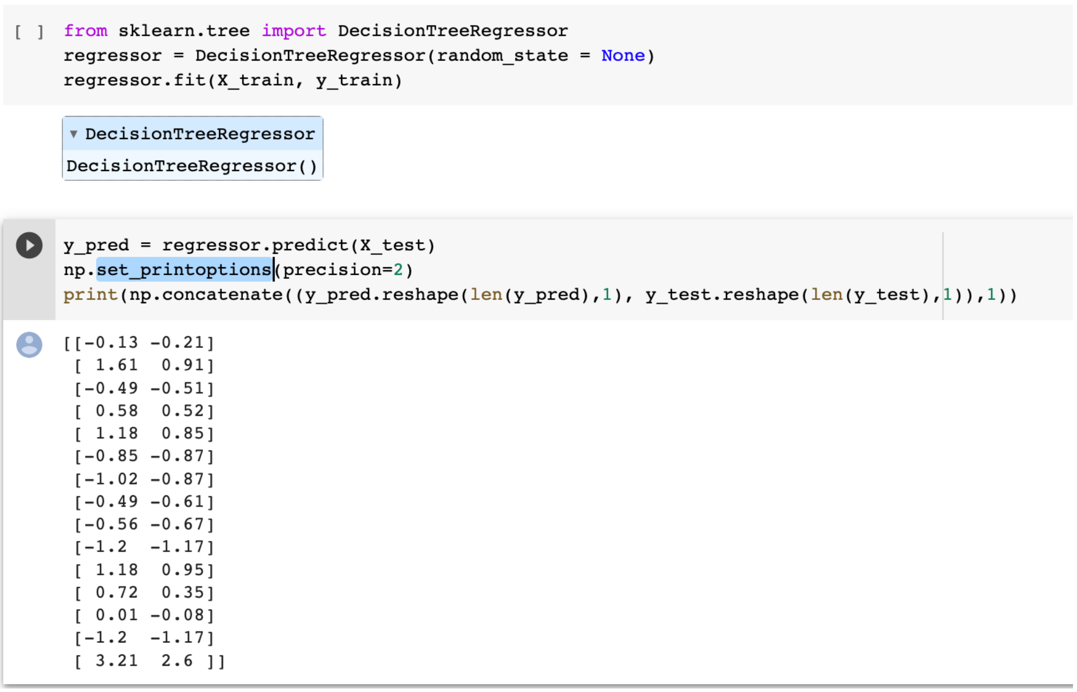


```
[344] r2_score(y_test, y_pred)*100

      94.93371200606225
```

Figure 8: Shows the graph for prediction and test values and the R2 score

R2 score is a very important metric that is used to evaluate the performance of a regression-based machine learning model. It is a statistical measure of how well the regression predictions approximate the real data points. 94% for the model was a very good R2 score.

2. **Decision Tree**: a supervised learning algorithm, which is utilized for both classification and regression tasks. It has a hierarchical, tree structure, which consists of a root node, branches, internal nodes, and leaf nodes.



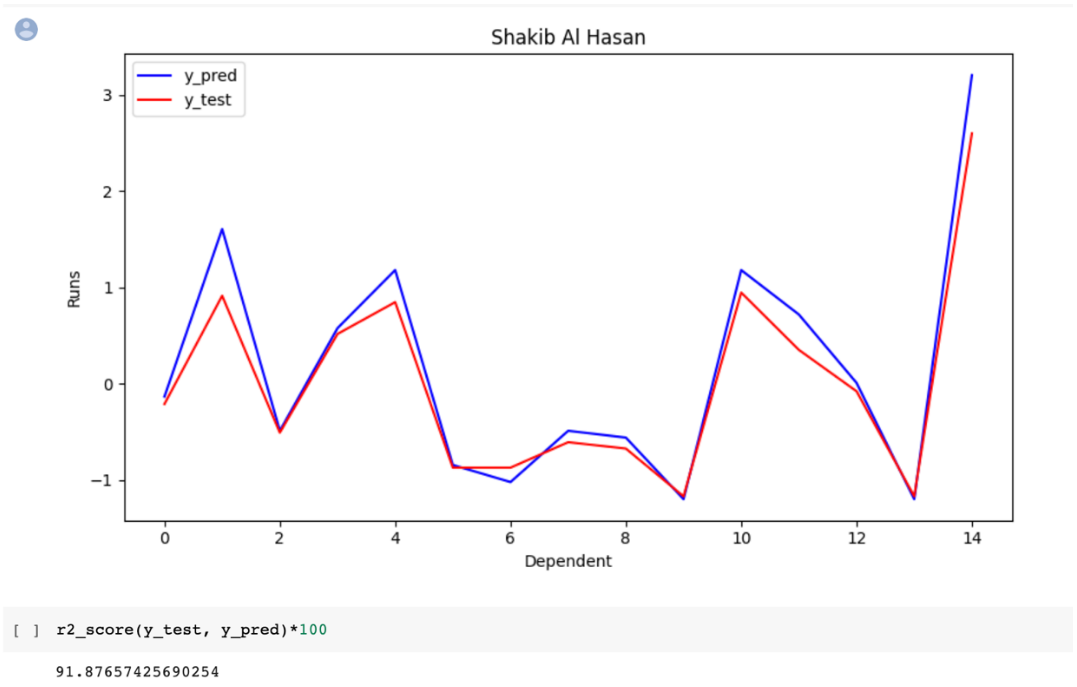
```
[ ] from sklearn.tree import DecisionTreeRegressor
    regressor = DecisionTreeRegressor(random_state = None)
    regressor.fit(X_train, y_train)
```

```
▾ DecisionTreeRegressor
DecisionTreeRegressor()
```

```
y_pred = regressor.predict(X_test)
np.set_printoptions(precision=2)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

```
[[-0.13 -0.21]
 [ 1.61  0.91]
 [-0.49 -0.51]
 [ 0.58  0.52]
 [ 1.18  0.85]
 [-0.85 -0.87]
 [-1.02 -0.87]
 [-0.49 -0.61]
 [-0.56 -0.67]
 [-1.2  -1.17]
 [ 1.18  0.95]
 [ 0.72  0.35]
 [ 0.01 -0.08]
 [-1.2  -1.17]
 [ 3.21  2.6 ]]
```

Figure 9: Code for DT model

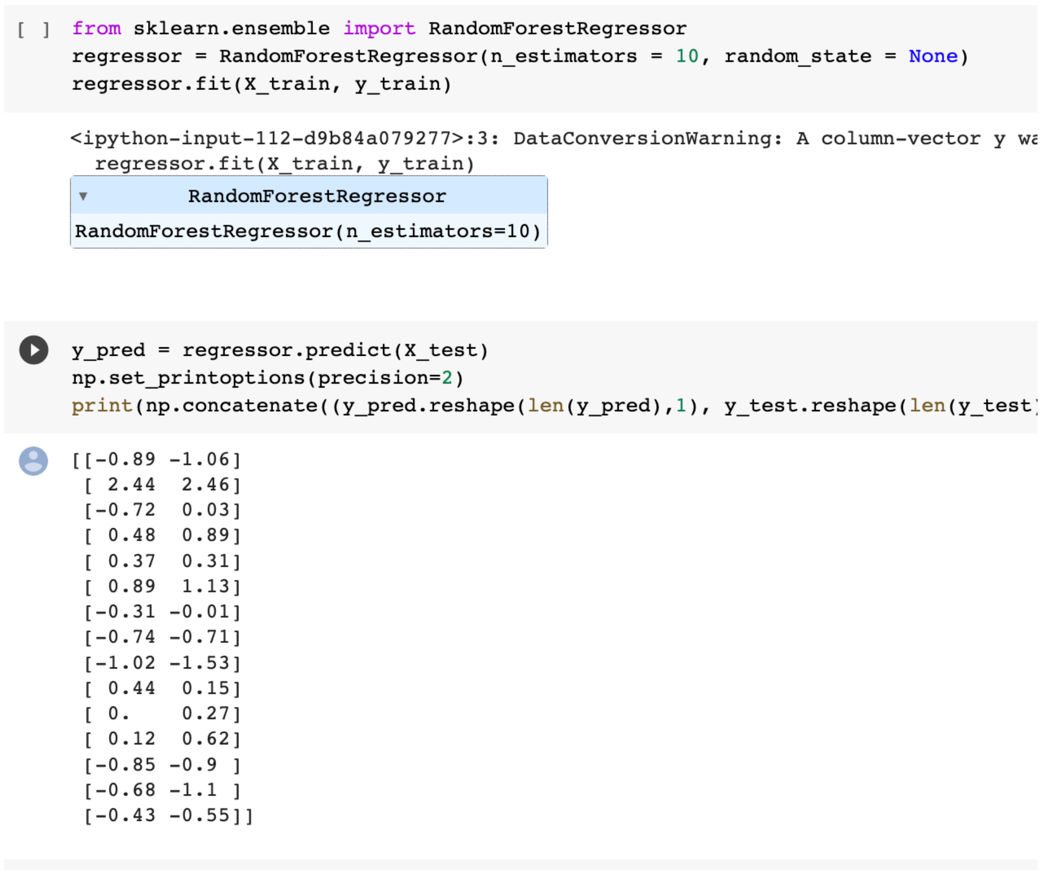


```
[ ] r2_score(y_test, y_pred)*100

    91.87657425690254
```

Figure 10: : Shows the graph for prediction and test values and the R2 score

Almost 92% for the model was a very good R2 score. Better than our previous model.

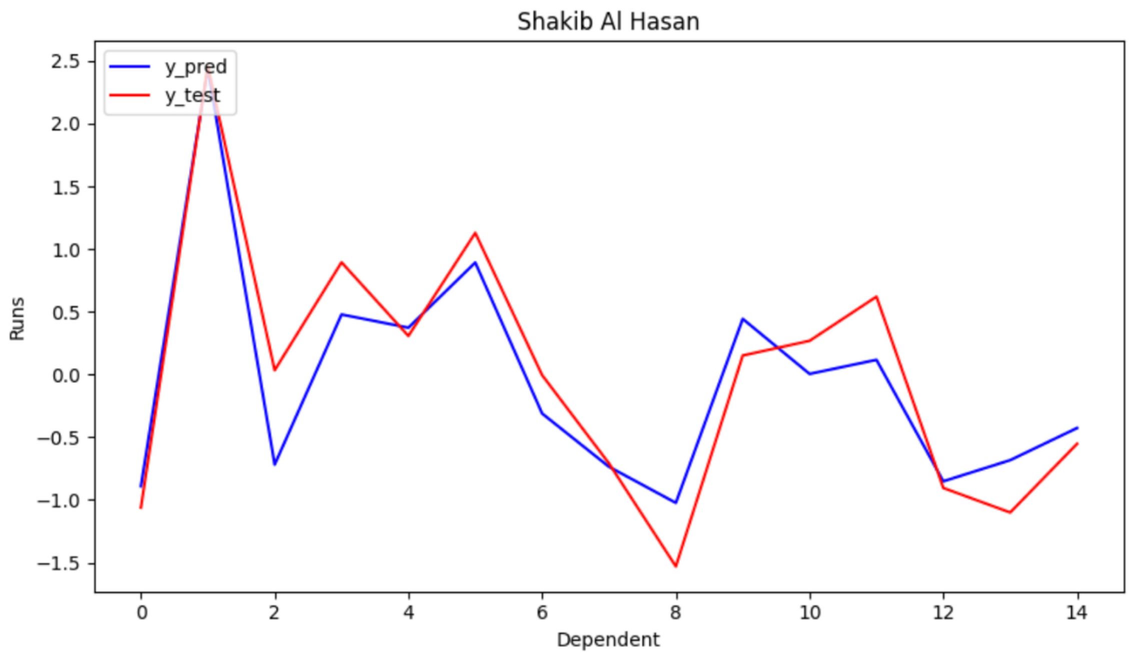
3. **Random Forest**: Random forest is a commonly used machine learning algorithm which combines the output of multiple decision trees to reach a single result. Its ease of use and makes it handles both classification and regression problems. In our case it is a regression problem as are trying to predict a value.

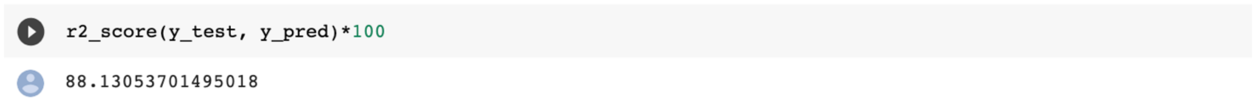


```
[ ] from sklearn.ensemble import RandomForestRegressor
    regressor = RandomForestRegressor(n_estimators = 10, random_state = None)
    regressor.fit(X_train, y_train)

    <ipython-input-112-d9b84a079277>:3: DataConversionWarning: A column-vector y wa
      regressor.fit(X_train, y_train)
    ▾          RandomForestRegressor
    RandomForestRegressor(n_estimators=10)
```

```
y_pred = regressor.predict(X_test)
np.set_printoptions(precision=2)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test)

[[-0.89 -1.06]
 [ 2.44  2.46]
 [-0.72  0.03]
 [ 0.48  0.89]
 [ 0.37  0.31]
 [ 0.89  1.13]
 [-0.31 -0.01]
 [-0.74 -0.71]
 [-1.02 -1.53]
 [ 0.44  0.15]
 [ 0.    0.27]
 [ 0.12  0.62]
 [-0.85 -0.9 ]
 [-0.68 -1.1 ]
 [-0.43 -0.55]]
```

Figure 11: Shows code for RF model





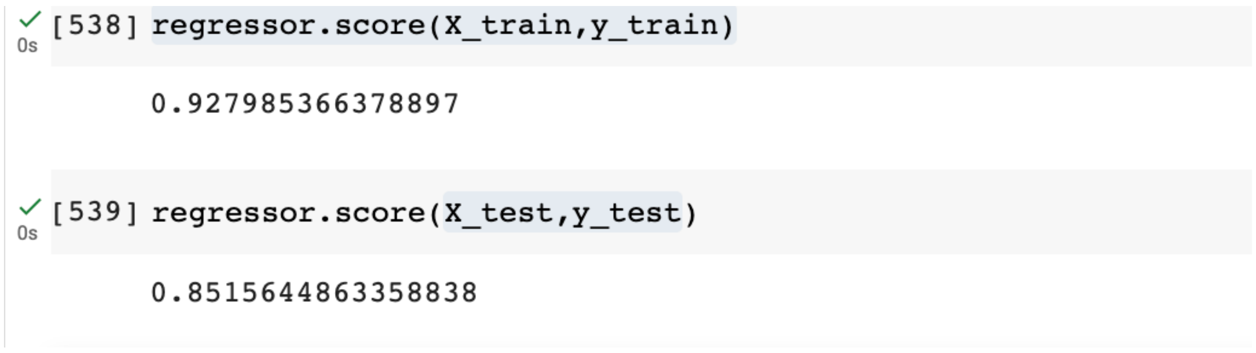
```
r2_score(y_test, y_pred)*100
```

```
88.13053701495018
```

Figure 12: Shows the graph for prediction and test values and the R2 score

88% for the model was a very good R2 score.

Now, if we compare the three different models, our decision tree model performed the best as it had the highest R2 score of 92%.



```
[538] regressor.score(X_train,y_train)

      0.927985366378897

[539] regressor.score(X_test,y_test)

      0.8515644863358838
```

Figure 13: Train and test accuracy

<u>Surprising Discoveries and Conclusion:</u>

At times, I noticed that my train accuracy was less than the test accuracy and after researching about it, I found out that this might be due to the small dataset I have. With more data in the future, this project might be a better one. Overall, I feel this was a dream project for me as I always wanted to do something in the field of cricket and finally it happened. When I tried predicting new values, the models were performing the best. I believe, with more data this problem could be solved.