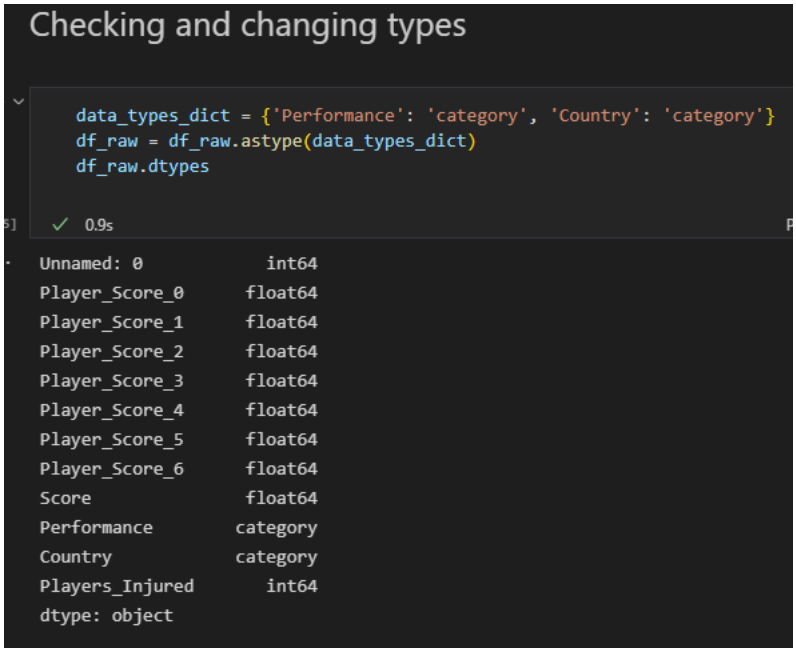
Dataset Analysis

Supervised Learning by Oleg Lastocichin

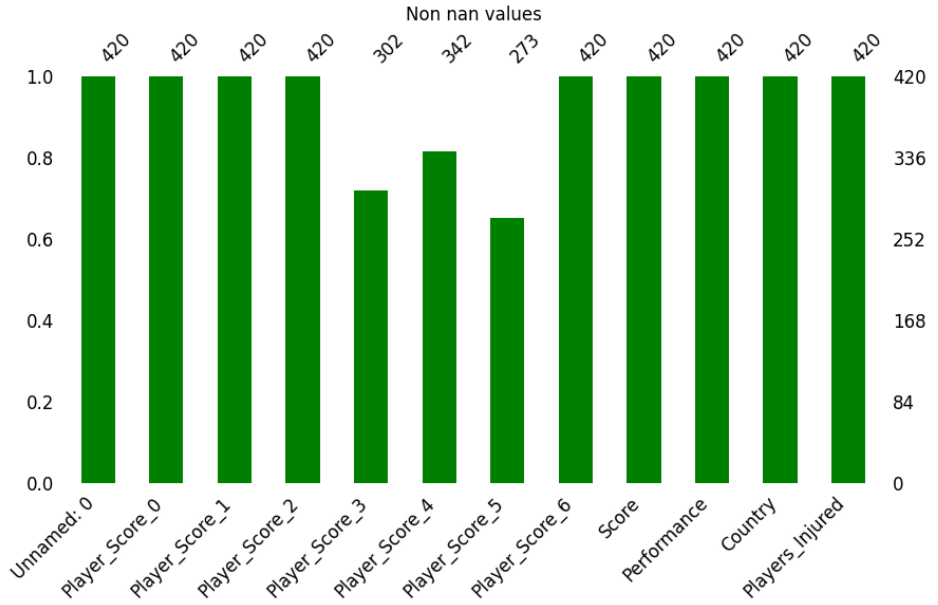
27/11/2022

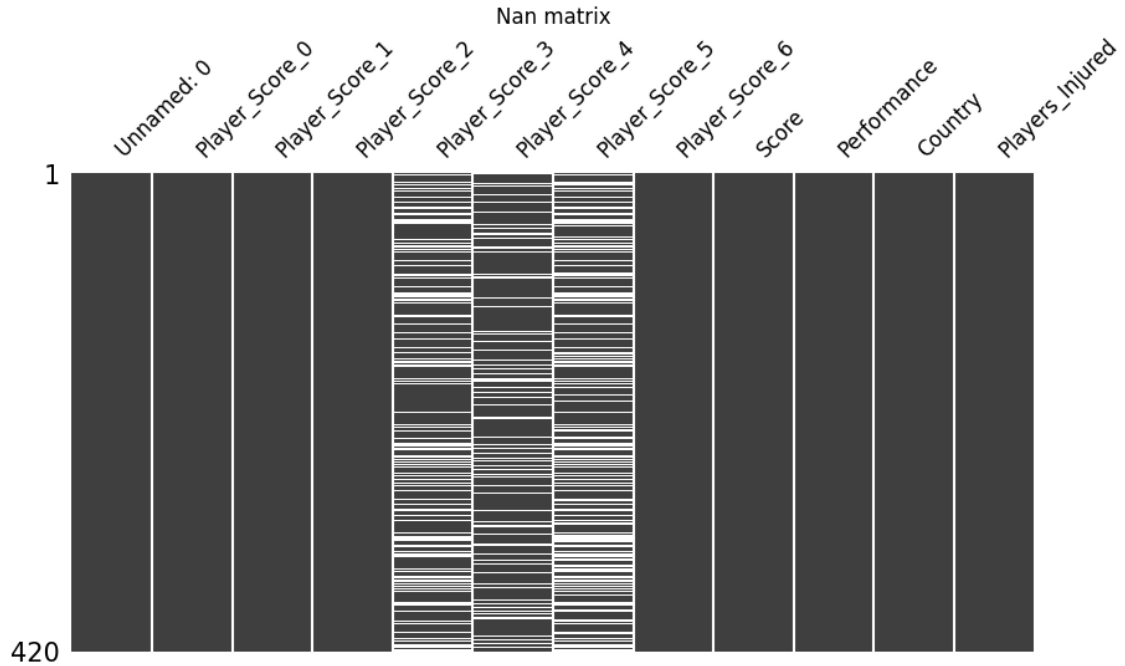
Questions and Tasks

1. Load and explore the dataset
    a. There are 9 numerical features (Player Score 0 to 6, Score, Players Injured), there are 2 categorical features (Performance, Country)

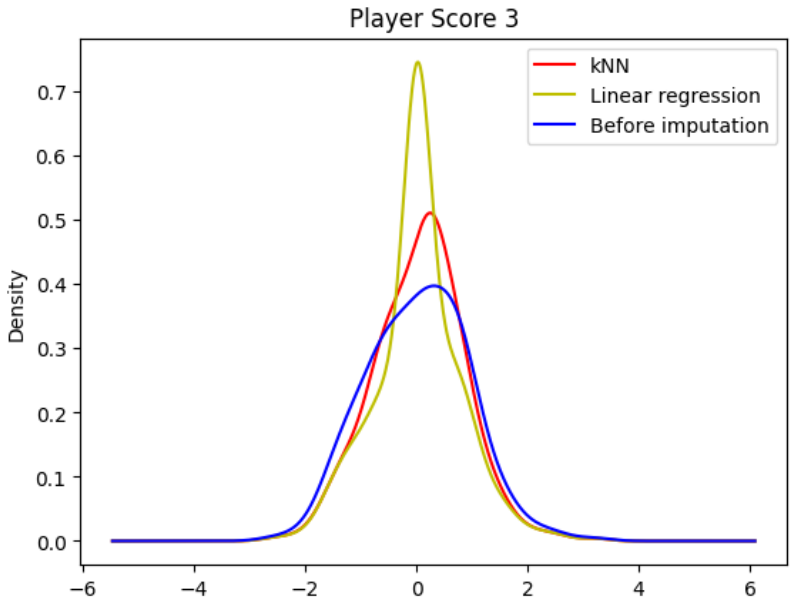


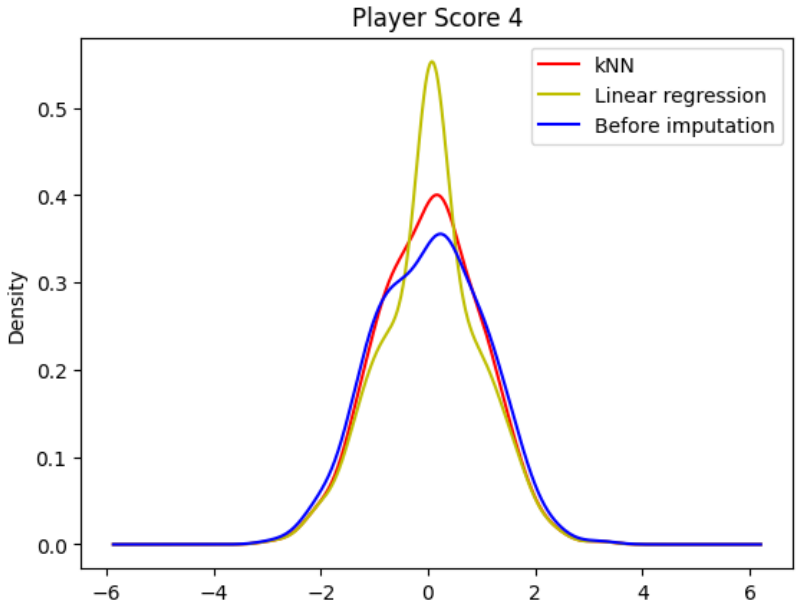
b. There are missing values in the dataset

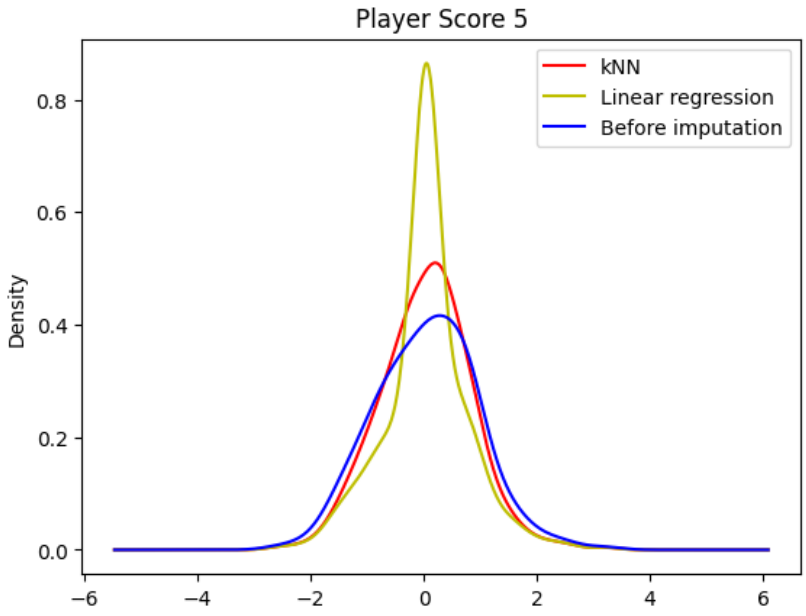




Handling the missing values in the dataset:
Since the missing values don't go over 70%, I won't drop the columns instead I will try to impute them with Linear Regression or kNN


Player Score 3
kNN
Linear regression
Before imputation
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
−6
−4
−2
0
2
4
6

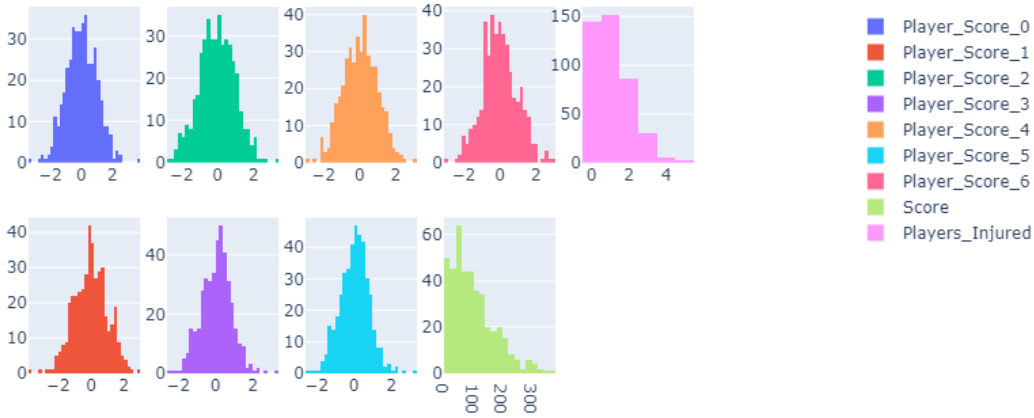

Player Score 4
kNN
Linear regression
Before imputation
Density
0.0
0.1
0.2
0.3
0.4
0.5
−6
−4
−2
0
2
4
6




```
Player Score 3 lr distance: 0.15804035497765384
Player Score 4 lr distance: 0.12638514073792098
Player Score 5 lr distance: 0.19037973174084133
Player Score 3 knn distance: 0.0645205517780904
Player Score 4 knn distance: 0.04416420089753926
Player Score 5 knn distance: 0.06357975816122968
```

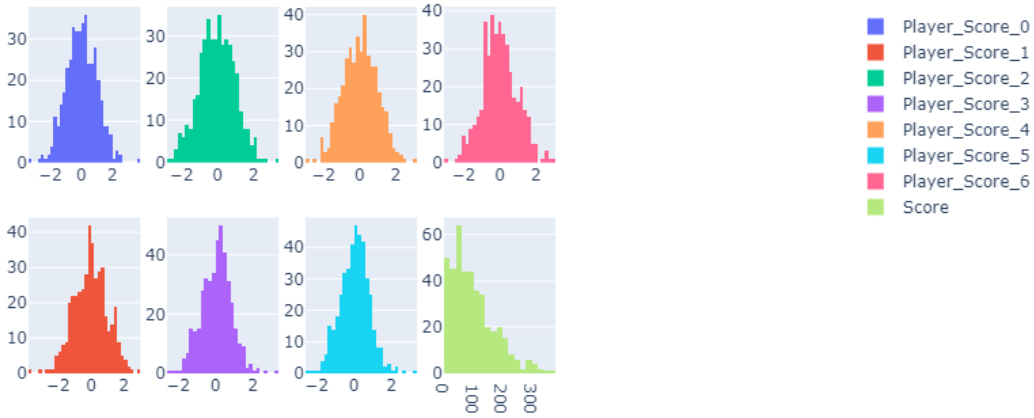
c. Since the Shannon distance is smaller with kNN I will use that to fill my dataset's Nan values and it's clearly visible from the plots too.

2. Prepare the dataset for a Linear Regression task.



a.

b. I will use a Robust scaler such that I can manage the outliers too, and I can represent the data for a better Linear Regression process



distribution after scaling

c. Since robust scaling handles outliers too this process should be skipped but I did a IQR-score outlier detection to be sure, and as

expected nothing changed

```
for i in df_scaled.columns:
    # discovering outliers with IQR-score
    Q1 = df_scaled[i].quantile(0.05)
    Q3 = df_scaled[i].quantile(0.95)
    IQR = Q3 - Q1
    print(IQR)

    # DROP
    logical_index_not_outliers = (df_scaled[i] > (
        Q1 - 1.5 * IQR)) & (df_scaled[i] < (Q3 + 1.5 * IQR))
    df_scaled = df_scaled[logical_index_not_outliers]
    # CAP
    df_scaled.loc[(df_scaled[i] < Q1), i] = Q1
    df_scaled.loc[(df_scaled[i] > Q3), i] = Q3
df_scaled.shape
```

✓ 0.1s

```
3.2376199999999975
3.3141299999999987
3.300774999999999
2.6930316666666663
3.040801666666666
2.595353333333331
3.2383649999999995
235.73340999999996

(420, 8)
```

d. I think it's necessary if I want to use the dataset for a Linear Regression and use this features too, then I used label encoding using dataframe.replace from pandas

```
# Replace a categorical value with a specific numeric one
dictionary = {"Performance": {'below_average': 0, 'neutral': 1,
                              'average': 2, 'above_average': 3, 'extraordinary': 4}}
df_scaled.replace(dictionary, inplace=True)
# Replace a categorical value with a specific numeric one
dictionary = {'France': 0, 'Finland': 1, 'Germany': 2,
              'Norway': 3, 'Switzerland': 4, 'The_Netherlands': 5, 'Italy': 6}
df_scaled.replace(dictionary, inplace=True)

display(df_scaled)
```

✓ 0.7s Python

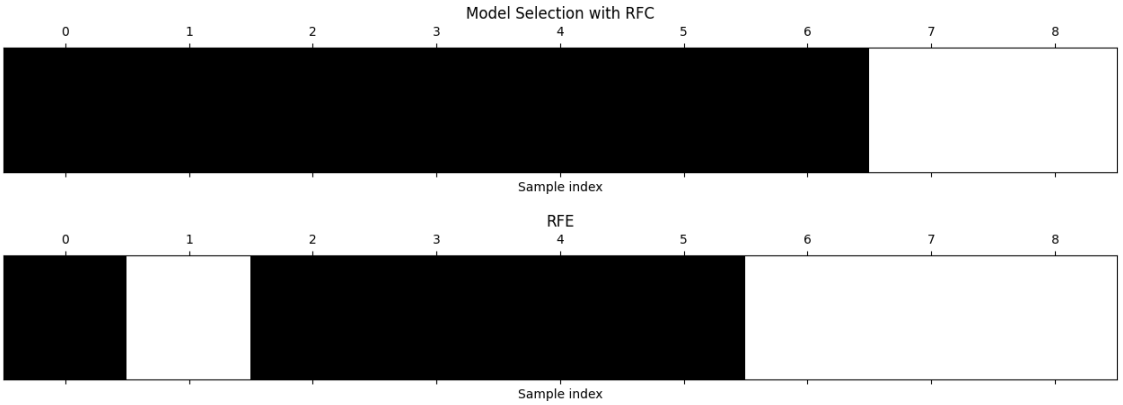
ore_1	Player_Score_2	Player_Score_3	Player_Score_4	Player_Score_5	Player_Score_6	Score	Country	Performance
8710	0.83170	0.206000	0.914267	0.206000	-0.242400	82.9055	0	0
6590	0.55470	0.123400	0.624000	0.123400	1.176200	27.3858	1	0
9945	0.83320	-0.071500	0.190500	-0.071500	-0.796300	78.9628	2	1
3040	-1.07530	-0.074400	-1.117500	-0.074400	-1.622150	80.2715	1	0
3540	1.61592	-0.314200	-0.791300	-0.314200	-0.084600	102.4383	3	2
...	...	...	...	...	...	...	...	...
1950	1.61592	-0.138500	0.971600	-0.138500	-0.212800	114.0887	4	0
6000	-0.14490	-0.138433	-0.103067	-0.138433	1.616215	21.9961	1	3
6080	-0.98240	0.921100	1.517600	0.921100	-0.113900	50.7298	6	1
9130	0.03390	-0.333967	0.758600	-0.333967	0.076700	11.3839	2	4
7370	-0.88290	0.172200	-0.512000	0.172200	0.355500	19.0997	4	1

e. I increased the dimensionality of the dataset using PolynomialFeatuers from sklearn

f. I think I did all the useful/necessary transformation for the dataset so I will proceed with the next task.

3. Features Selection
    a. I performed an ANOVA test between Country and Score and ended up accepting H0 meaning that it doesn't exist variance between the groups, I won't include this feature for model training.

b. I tried with all the methodologies for feature selection, whereas some were not made for this dataset
I ended up making a choice between Model selection with random forest classifier and RFE, since I think those features would be more helpful for the Linear Regression task, my choice in the end was Model Selection



4. Linear Regression
   a. I trained the model using the transformed features including Performance and Country and not using the polynomials
   test RMSE=66.78679659791713
   b. test R2=-0.033550616190128
   c. train RMSE=66.4131367390314
   d. train R2=0.04970599069771997

```
df_train = df_model.copy(deep=True)
df_train.drop(columns='Score', inplace=True)
y = df_model['Score'].copy(deep=True)

X_train, X_test, y_train, y_test = train_test_split(
    df_train, y, random_state=24)
```

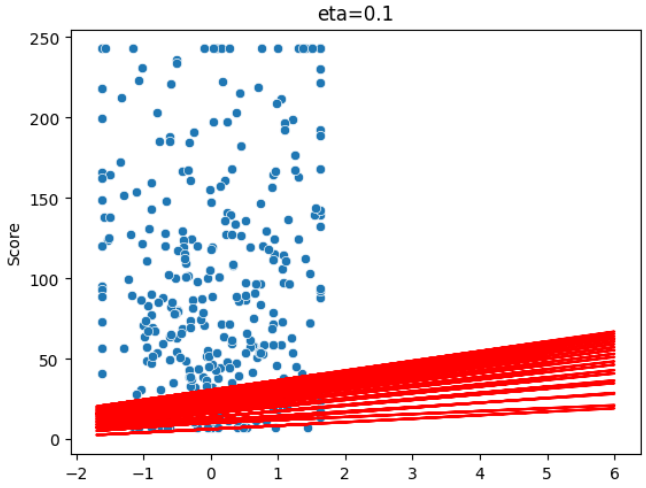
✓ 0.3s

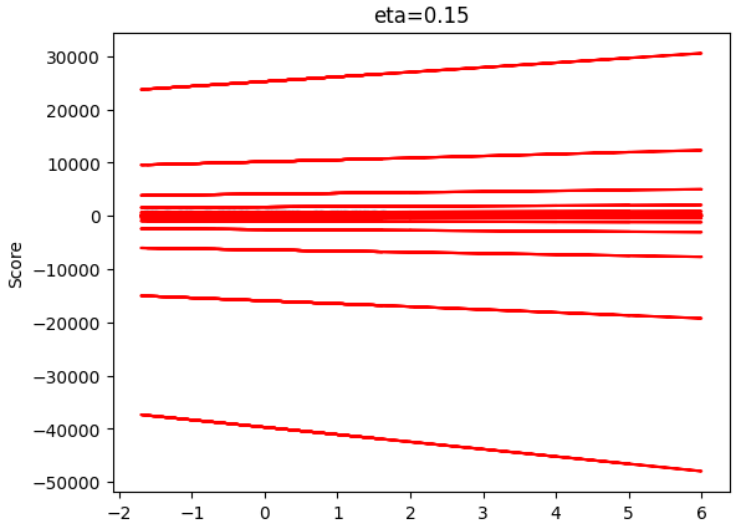
```
lin_reg = LinearRegression()

lin_reg.fit(X_train, y_train)
print("Intercept={}, Slope={}".format(lin_reg.intercept_, lin_reg.coef_))
y_predict = lin_reg.predict(X_test)
```

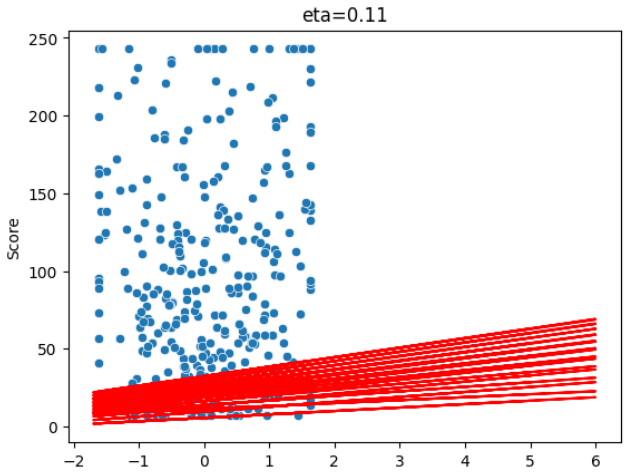
✓ 0.6s

```
Intercept=87.69643208217602, Slope=[  2.84579472  -2.07489303   3.95147882  34.26342482  -4.28547712
 -42.34416263   1.59470542   1.61554065   3.01421527]
```

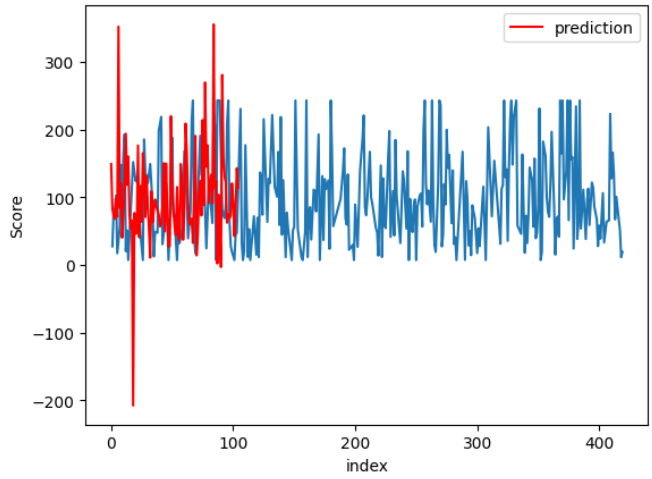
e. I used eta 0.1, 0.15 and 0.11 where the best one that wasn't overshooting was 0.11



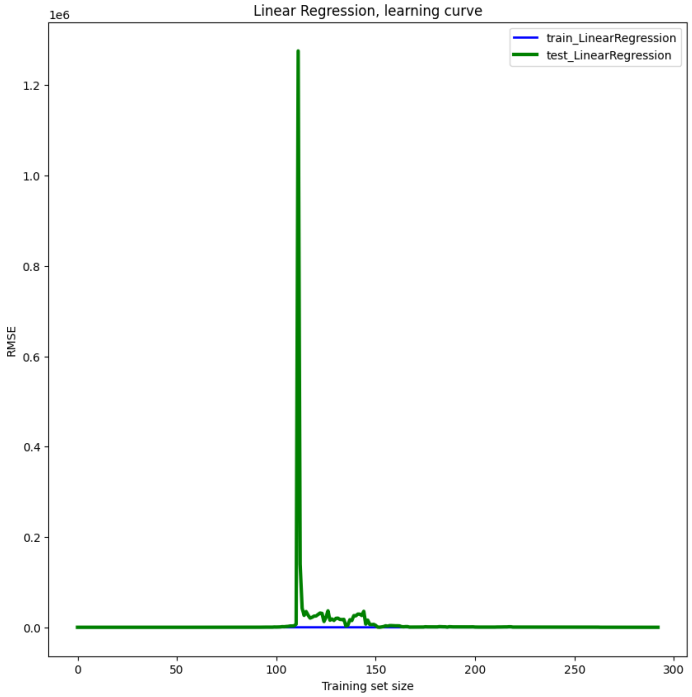




f. This linear regression felt more accurate from the predictions only

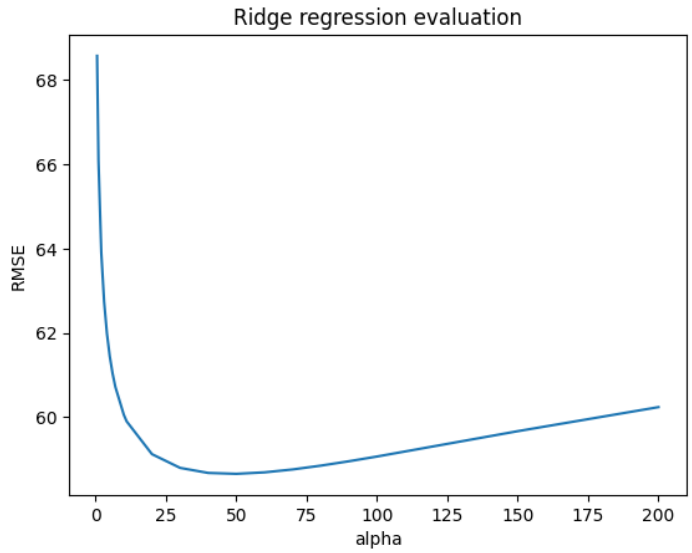


And in fact from the learning curve we can see that the RMSE is almost from the start 0 except from 1 strange spike at around 100 to 150 for the test set.



g. I performed the Ridge regression (I used all the features and the polynomial features) and evaluated the RMSE and I noticed that it

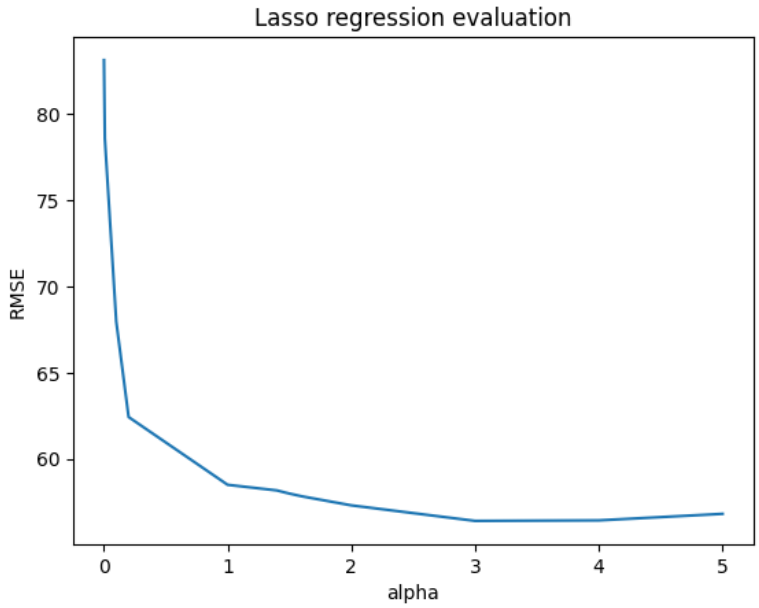
needed a really large alpha to reduce the RMSE



And the RMSE didn't shift that drastically either (while using big differences of alpha)

```
Alpha = 0.5, RMSE = 68.57355765624578
Alpha = 1, RMSE = 66.08425343185684
Alpha = 2, RMSE = 63.89800465881835
Alpha = 3, RMSE = 62.74148271985468
Alpha = 4, RMSE = 61.98714340426305
Alpha = 5, RMSE = 61.44688729410152
Alpha = 6, RMSE = 61.03751387174634
Alpha = 7, RMSE = 60.71506907685875
Alpha = 10, RMSE = 60.05489028113115
Alpha = 11, RMSE = 59.89901017830868
Alpha = 20, RMSE = 59.11985325118686
Alpha = 30, RMSE = 58.79375568047981
Alpha = 40, RMSE = 58.67351735210657
Alpha = 50, RMSE = 58.65254173109622
Alpha = 60, RMSE = 58.68747579419384
Alpha = 70, RMSE = 58.756424154661374
Alpha = 80, RMSE = 58.84683479110648
Alpha = 90, RMSE = 58.950916180405734
Alpha = 100, RMSE = 59.063572803085776
Alpha = 150, RMSE = 59.6651973264918
Alpha = 200, RMSE = 60.2360032374881
Minimum test-RMSE = 58.65254173109622
```

h. Performing the Lasso regression felt better since I didn't need such large alpha numbers (Even though I think Alpha = 3 is still kind of big) and RMSE was smallest at alpha = 3

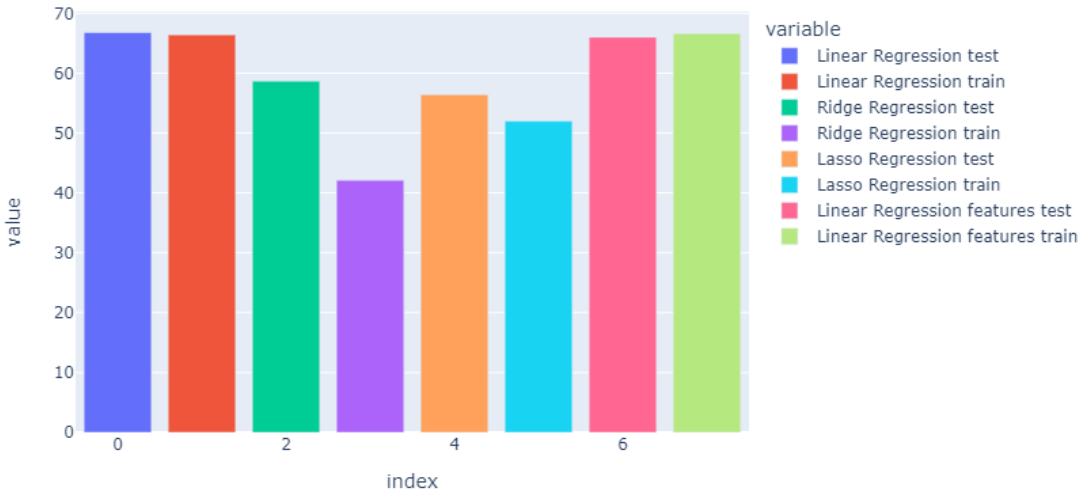


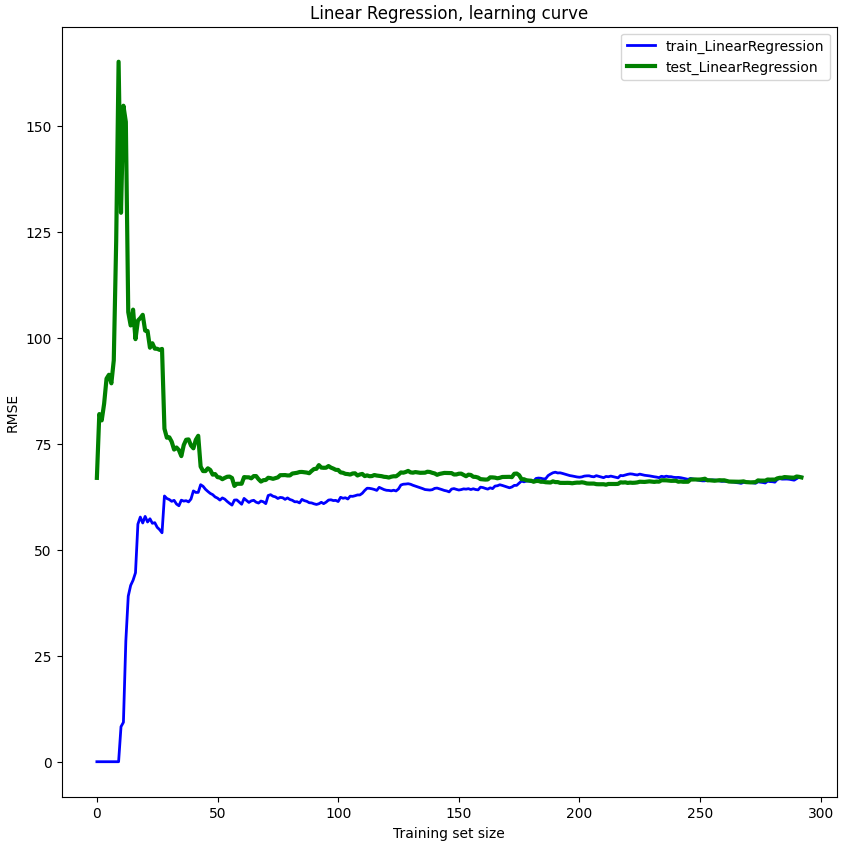

```
Alpha = 0.001, RMSE = 83.14460015202002
Alpha = 0.002, RMSE = 82.4688697498599
Alpha = 0.003, RMSE = 81.81210277966377
Alpha = 0.004, RMSE = 81.16805670963163
Alpha = 0.005, RMSE = 80.52912997267649
Alpha = 0.006, RMSE = 79.89589917287894
Alpha = 0.008, RMSE = 78.59684653506544
Alpha = 0.1, RMSE = 67.9762882887414
Alpha = 0.2, RMSE = 62.42722535216395
Alpha = 1, RMSE = 58.493782000869274
Alpha = 1.4, RMSE = 58.16557357345882
Alpha = 1.45, RMSE = 58.070995524105975
Alpha = 1.5, RMSE = 57.98369728972258
Alpha = 1.6, RMSE = 57.82541862758639
Alpha = 2, RMSE = 57.300048988152156
Alpha = 3, RMSE = 56.398550739245366
Alpha = 4, RMSE = 56.42844534647608
Alpha = 5, RMSE = 56.80789289772092
Minimum test-RMSE = 56.398550739245366
```

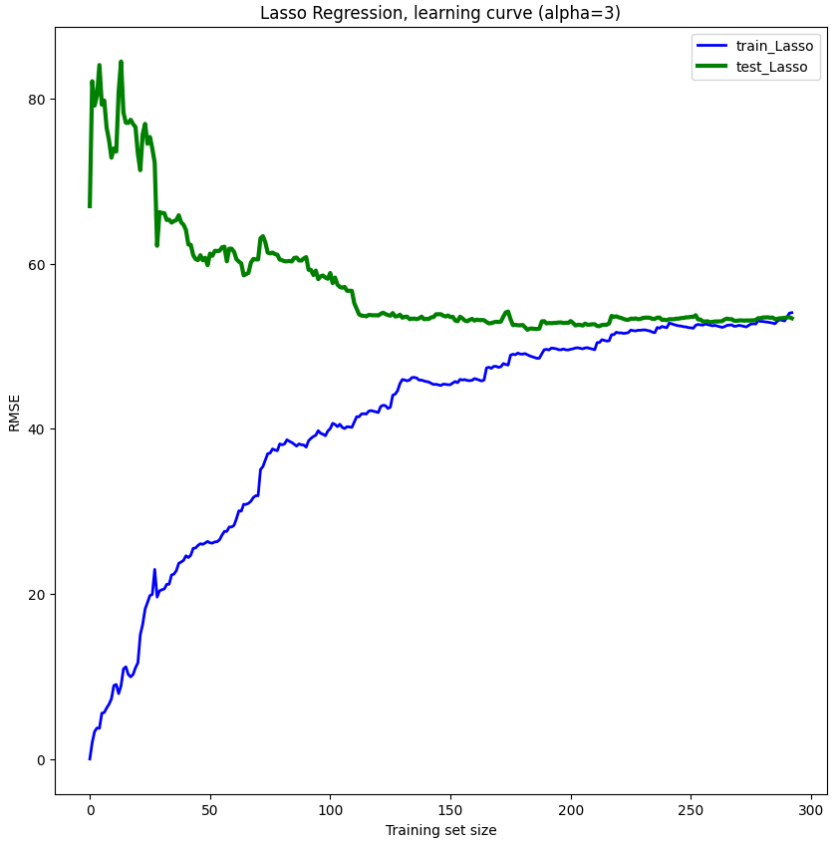
i. After the feature selection and using only the features I thought were going to represent a better regression I ended up with similar RMSE results just slightly better ones I suppose
test RMSE=66.0050336150932
test R2=-0.009496080255134487
train RMSE=66.61741320391856
train R2=0.04385108740699217

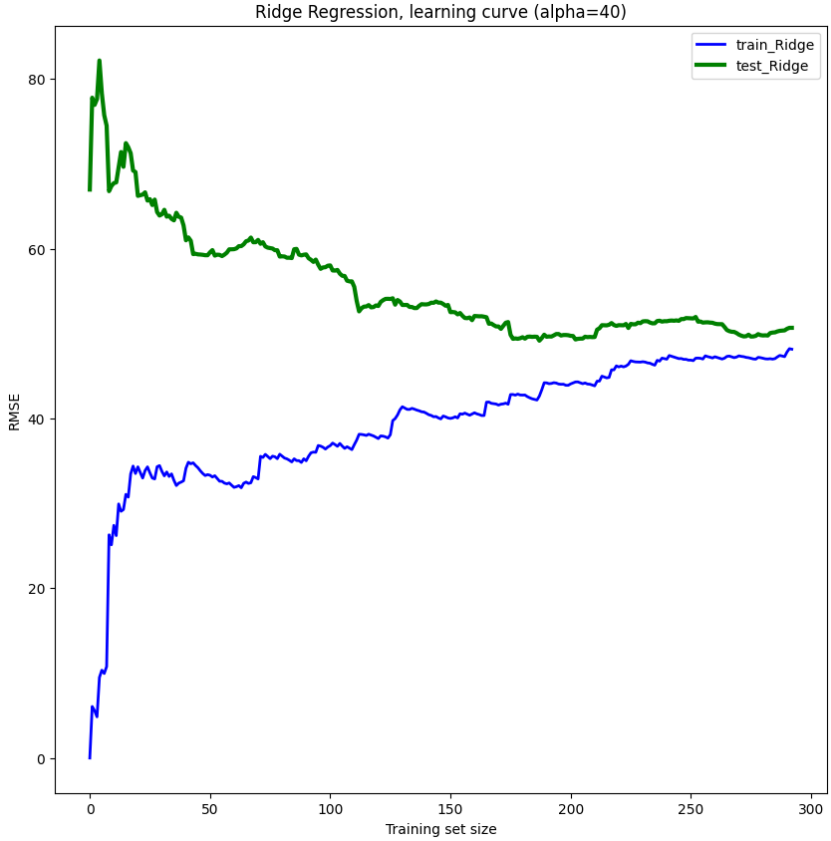
g. I computed the learning curves and a table for the models where evaluation was requested, I noticed that the Linear Regressions always converge quickly but usually have a larger RMSE, instead the Lasso

converges slightly slower but provides a smaller RMSE, and lastly the Ridge regression I think is the one who converges the slowest but has a smaller RMSE overall on the train set and Lasso had the best RMSE for the test set.






Lasso Regression, learning curve (alpha=3)
train_Lasso
test_Lasso
RMSE
Training set size
0
20
40
60
80
0
50
100
150
200
250
300


Ridge Regression, learning curve (alpha=40)
train_Ridge
test_Ridge
RMSE
Training set size
0
20
40
60
80
0
50
100
150
200
250
300

h.
Since the data was synthetic kNN imputation worked better instead of Linear Regression for this kind of dataset even though realistically Linear Regression should perform better since we shouldn’t be able to evaluate a team’s capabilities from the “Neighbouring” teams.
From the results I obtained above I think the best regression is Lasso Regression since it delivered the smallest RMSE for the test adding the fact that it converged faster than the Ridge regression, instead I think that the Batch Gradient could converge with eta = 0.1 and it was the best suited since it was converging to the linear regression results.
The dataset was obviously underfitting across all the models since I had a real high value for RMSE and R2 meaning that the complexity was too low to fit the data.
I don’t think any model is good to evaluate a team’s Score and a better job on fitting the data could’ve been done.