Baseball Case Study

Problem Statement:

This dataset utilizes data from 2014 Major League Baseball seasons in order to develop an algoirthm that predicts the number of wins for a given team in the 2015 season based on several different indicators of success. This model is used to select best team based on best input feature. .There are 16 different features that will be used as the inputs to the machine learning and the output will be a value that represents the number of wins.

Input features:

- R: Runs-times reached home plate legally and safely
- AB: At Bats-plate appearances, not including bases on balls, being hit by pitch, sacrifices, interference, or obstruction
- H: Hits- reaching base because of a batted, fair ball without error by the defence
- 2B: Doubles-hits on which the batter reaches second base safely without the contribution of a fielding error
- 3B: Triples- hits on which the batter reaches third base safely without the contribution of a fielding error
- HR: Homeruns
- BB: Walks-times pitching four balls, allowing the batter to take first base
- SO: Strikeouts
- SB: Stolen Bases- number of bases advanced by the runner while the ball is in the possession of the defence
- RA: Runs Allowed
- ER: Earned Runs- number of runs that did not occur as a result of errors or passed balls
- ERA: Earned Run Average (ERA)- total number of earned runs (see "ER" above), multiplied by 9, divided by innings pitched
- SO: Shutouts- number of complete games pitched with no runs allowed
- SV: Saves- number of games where the pitcher enters a game led by the pitcher's team, finishes the game without surrendering the lead, is not the winning pitcher, and either (a) the lead was three runs or fewer when the pitcher entered the game; (b) the

potential tying run was on base, at bat, or on deck; or (c) the pitcher pitched three or more innings

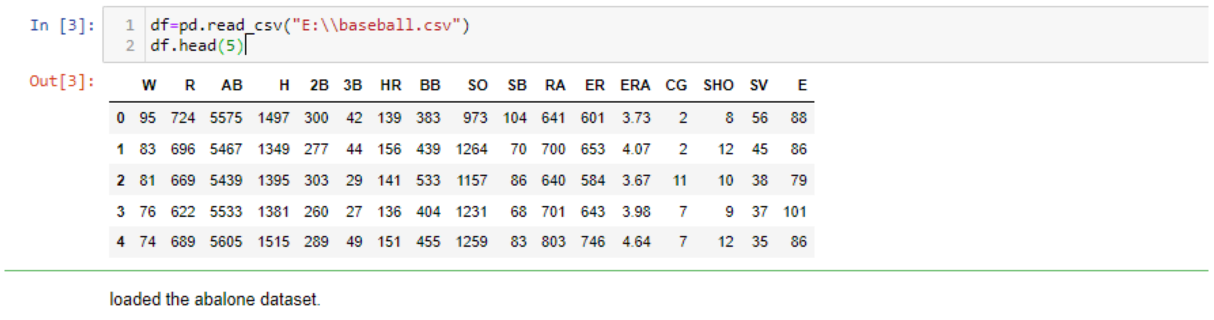
- CG: Complete Games-number of games where player was the only pitcher for their team
- E: Errors- the judgment of the official scorer, of a fielder misplaying a ball in a manner that allows a batter or base runner to advance one or more bases

Output: Number of predicted wins (W)

Importing the Libraries

import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
from matplotlib import pyplot as plt
from scipy.stats import zscore

Loading Dataset:

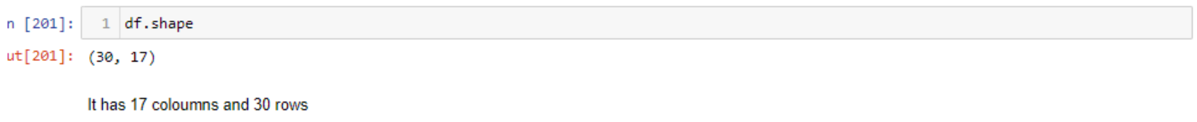


In [3]:
```
df=pd.read_csv("E:\\baseball.csv")
df.head(5)
```

Out[3]:

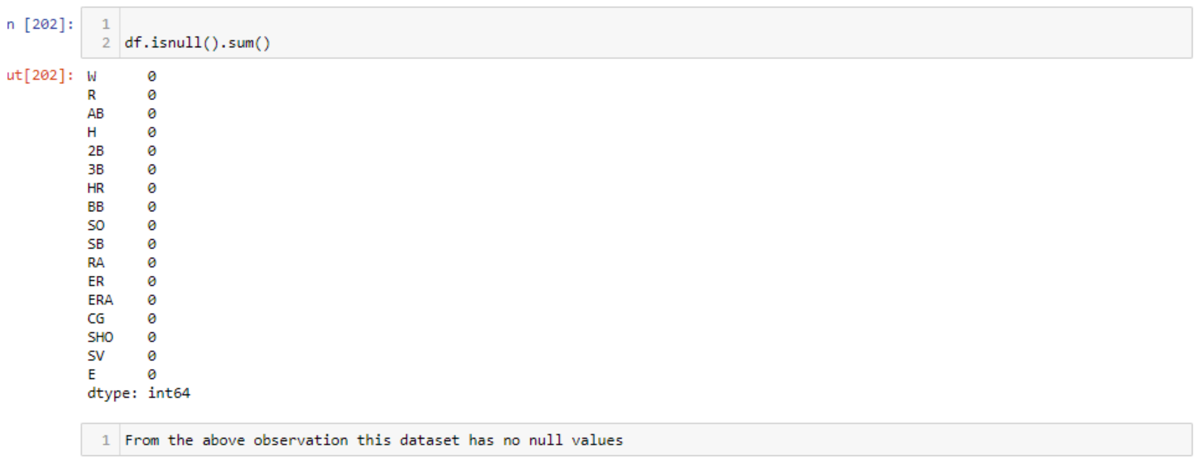
	W	R	AB	H	2B	3B	HR	BB	SO	SB	RA	ER	ERA	CG	SHO	SV	E
0	95	724	5575	1497	300	42	139	383	973	104	641	601	3.73	2	8	56	88
1	83	696	5467	1349	277	44	156	439	1264	70	700	653	4.07	2	12	45	86
2	81	669	5439	1395	303	29	141	533	1157	86	640	584	3.67	11	10	38	79
3	76	622	5533	1381	260	27	136	404	1231	68	701	643	3.98	7	9	37	101
4	74	689	5605	1515	289	49	151	455	1259	83	803	746	4.64	7	12	35	86

loaded the abalone dataset.



n [201]:
```
df.shape
```
ut[201]:
```
(30, 17)
```

It has 17 coloumns and 30 rows

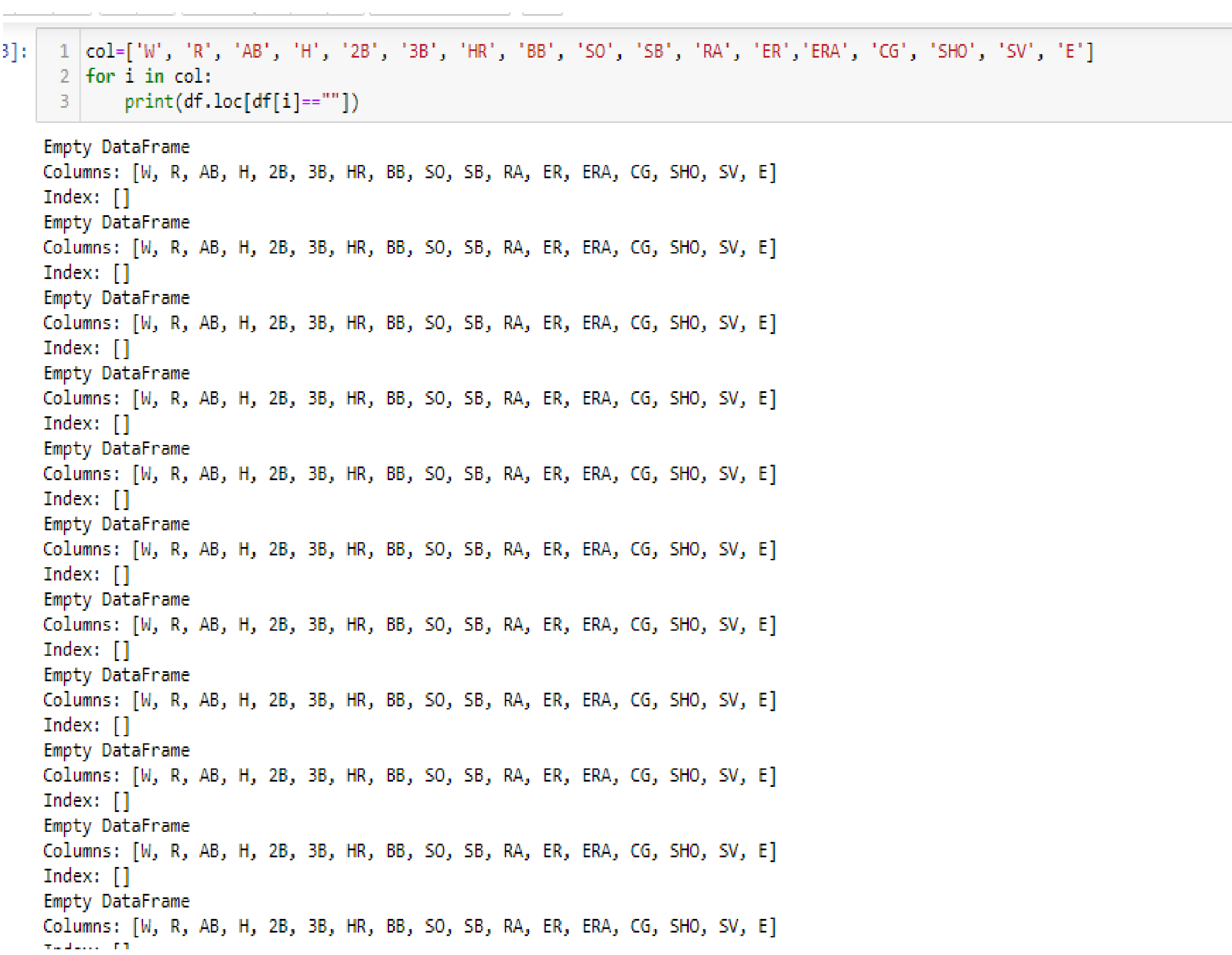


n [202]:
```

df.isnull().sum()
```
ut[202]:
```
W      0
R      0
AB     0
H      0
2B     0
3B     0
HR     0
BB     0
SO     0
SB     0
RA     0
ER     0
ERA    0
CG     0
SHO    0
SV     0
E      0
dtype: int64
```

```
From the above observation this dataset has no null values
```

This dataset has no null values and it has 30 rows and 17 columns

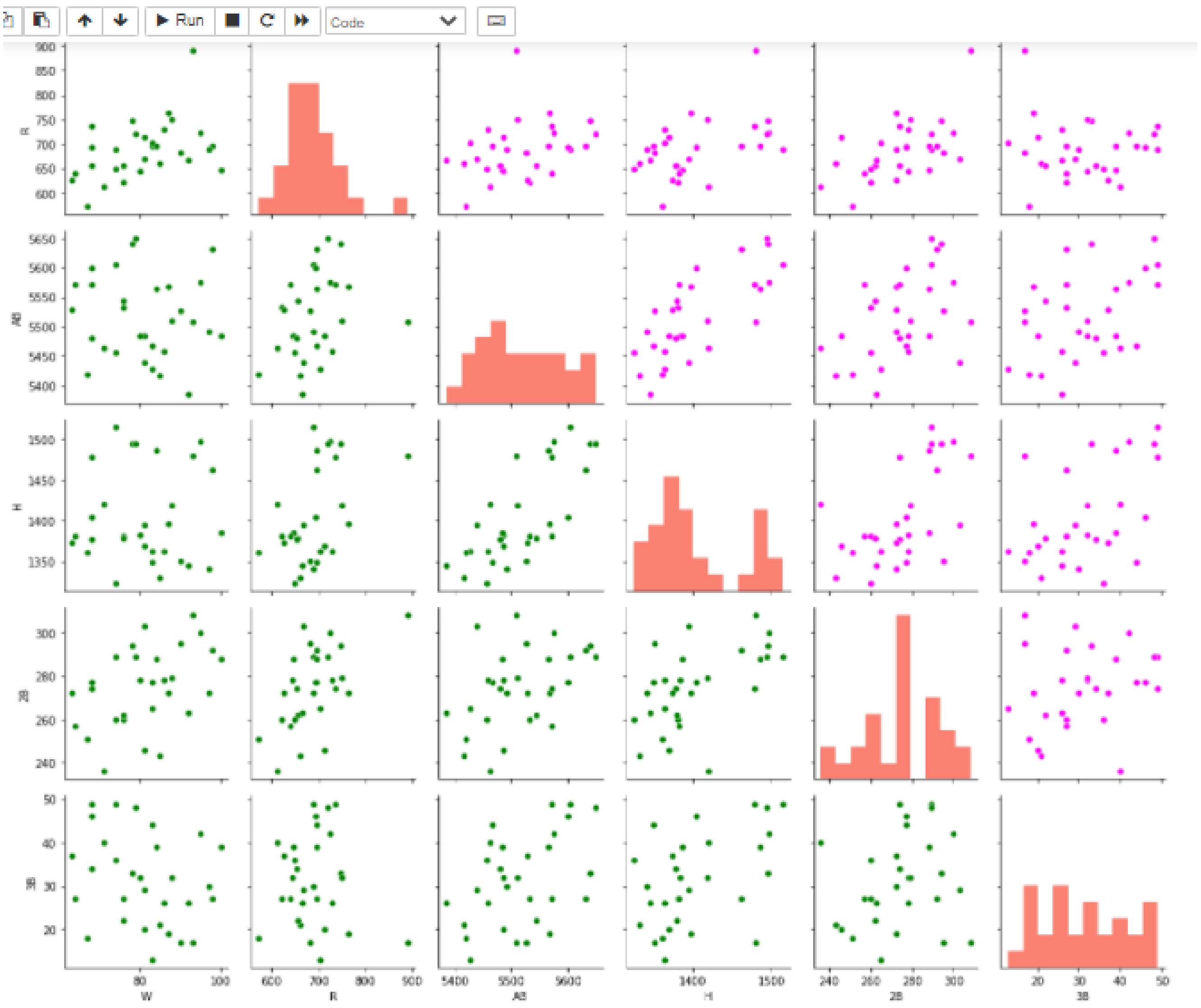
CHECKING FOR EMPTY SPACE IN DATASET



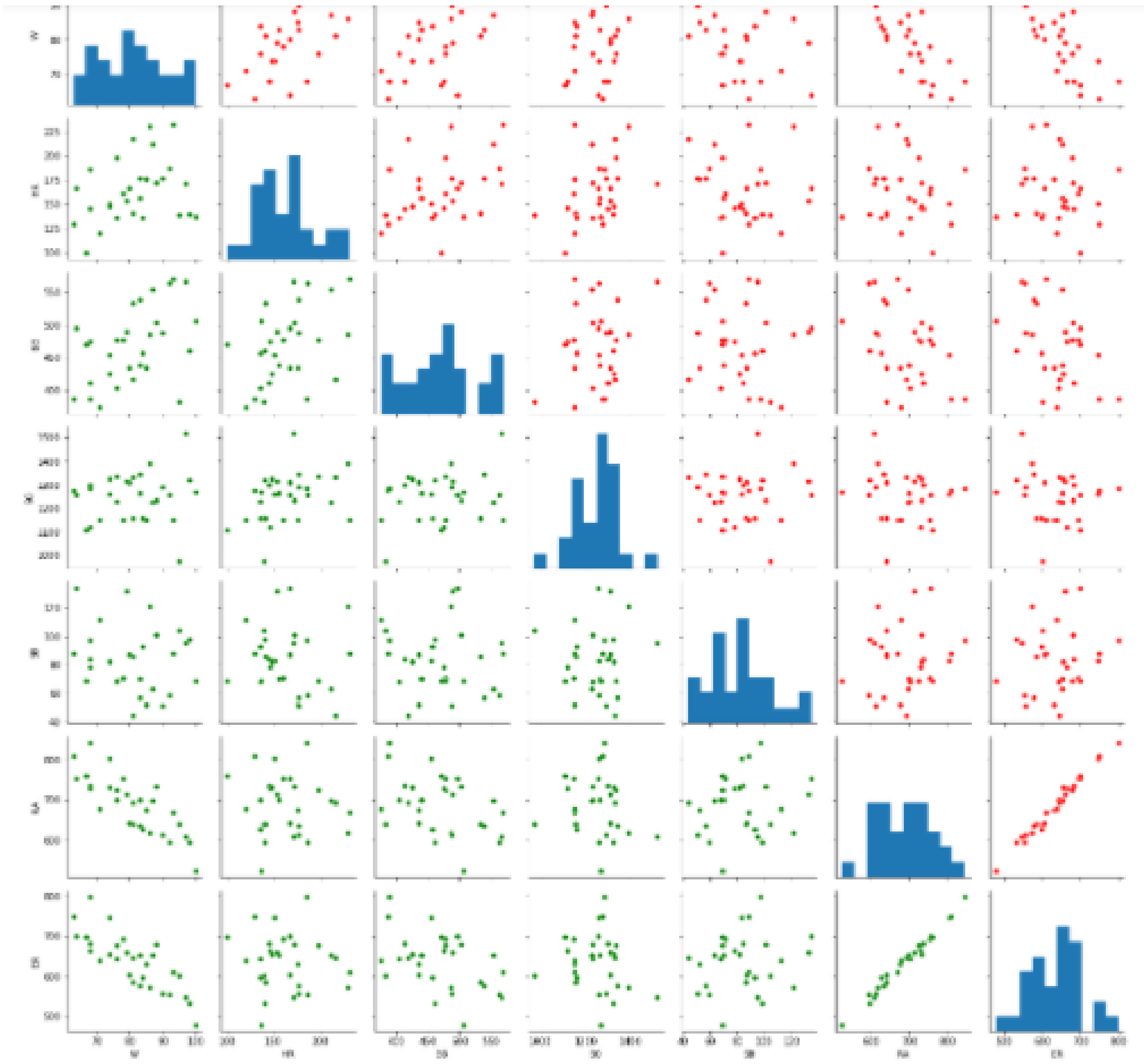
```
col=['W', 'R', 'AB', 'H', '2B', '3B', 'HR', 'BB', 'SO', 'SB', 'RA', 'ER','ERA', 'CG', 'SHO', 'SV', 'E']
for i in col:
    print(df.loc[df[i]==""])
```

```
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
Index: []
Empty DataFrame
Columns: [W, R, AB, H, 2B, 3B, HR, BB, SO, SB, RA, ER, ERA, CG, SHO, SV, E]
```

This dataset has no empty space as value



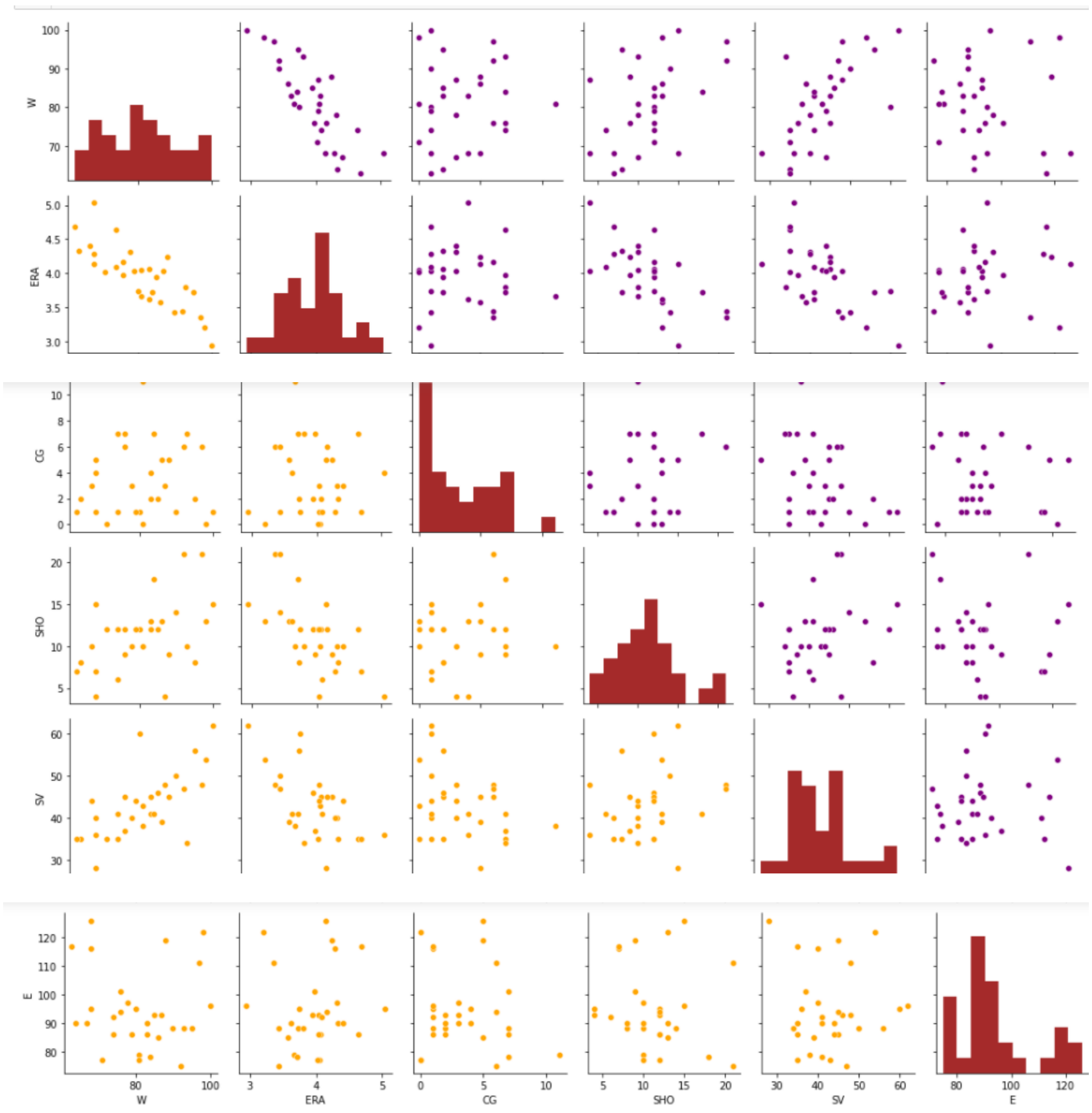
The data points of the features R and AB showing an uphill pattern as you move from left to right so it has positive relationship. The data points of the features R and w showing an uphill pattern as you move from left to right so it has positive relationship.



The data points of ER and RA showing an uphill pattern as you move from left to right so it has positive relationship.

The data points of ER and w showing an downhill pattern as you move from left to right so it has negative relationship.

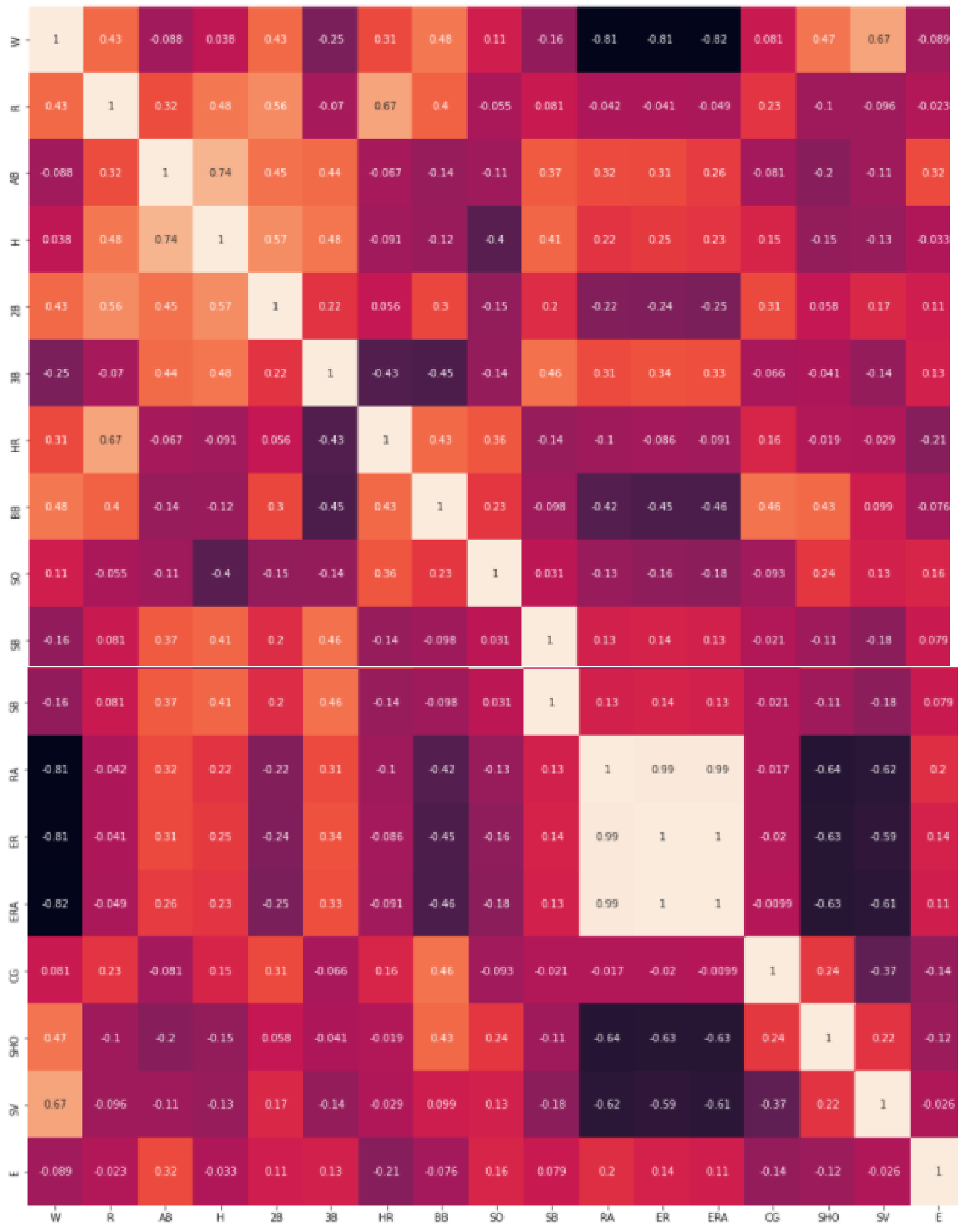
The data points of RA and w showing an downhill pattern as you move from left to right so it has negative relationship.



The data points of the features ERA and w showing an downhill pattern as you move from left to right so it has negative relationship.

The data points of the features SV and W showing an uphill pattern as you move from left to right so it has positive relationship.

CORRELATION MATRIX



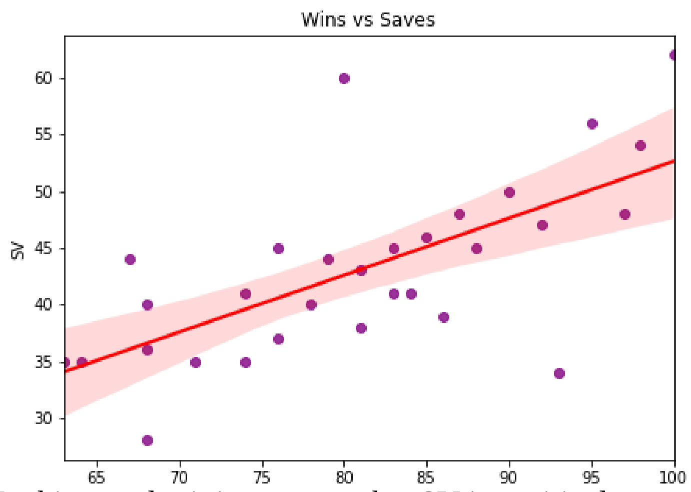
	W	R	AB	H	2B	3B	HR	BB	SO	SB	RA	ER	ERA	CG	SHO	SV	E
W	1	0.43	-0.088	0.038	0.43	-0.25	0.31	0.48	0.11	-0.16	-0.81	-0.81	-0.82	0.081	0.47	0.67	-0.089
R	0.43	1	0.32	0.48	0.56	-0.07	0.67	0.4	-0.055	0.081	-0.042	-0.041	-0.049	0.23	-0.1	-0.096	-0.023
AB	-0.088	0.32	1	0.74	0.45	0.44	-0.067	-0.14	-0.11	0.37	0.32	0.31	0.26	-0.081	-0.2	-0.11	0.32
H	0.038	0.48	0.74	1	0.57	0.48	-0.091	-0.12	-0.4	0.41	0.22	0.25	0.23	0.15	-0.15	-0.13	-0.033
2B	0.43	0.56	0.45	0.57	1	0.22	0.056	0.3	-0.15	0.2	-0.22	-0.24	-0.25	0.31	0.058	0.17	0.11
3B	-0.25	-0.07	0.44	0.48	0.22	1	-0.43	-0.45	-0.14	0.46	0.31	0.34	0.33	-0.066	-0.041	-0.14	0.13
HR	0.31	0.67	-0.067	-0.091	0.056	-0.43	1	0.43	0.36	-0.14	-0.1	-0.086	-0.091	0.16	-0.019	-0.029	-0.21
BB	0.48	0.4	-0.14	-0.12	0.3	-0.45	0.43	1	0.23	-0.098	-0.42	-0.45	-0.46	0.46	0.43	0.099	-0.076
SO	0.11	-0.055	-0.11	-0.4	-0.15	-0.14	0.36	0.23	1	0.031	-0.13	-0.16	-0.18	-0.093	0.24	0.13	0.16
SB	-0.16	0.081	0.37	0.41	0.2	0.46	-0.14	-0.098	0.031	1	0.13	0.14	0.13	-0.021	-0.11	-0.18	0.079
RA	-0.81	-0.042	0.32	0.22	-0.22	0.31	-0.1	-0.42	-0.13	0.13	1	0.99	0.99	-0.017	-0.64	-0.62	0.2
ER	-0.81	-0.041	0.31	0.25	-0.24	0.34	-0.086	-0.45	-0.16	0.14	0.99	1	1	-0.02	-0.63	-0.59	0.14
ERA	-0.82	-0.049	0.26	0.23	-0.25	0.33	-0.091	-0.46	-0.18	0.13	0.99	1	1	-0.0099	-0.63	-0.61	0.11
CG	0.081	0.23	-0.081	0.15	0.31	-0.066	0.16	0.46	-0.093	-0.021	-0.017	-0.02	-0.0099	1	0.24	-0.37	-0.14
SHO	0.47	-0.1	-0.2	-0.15	0.058	-0.041	-0.019	0.43	0.24	-0.11	-0.64	-0.63	-0.63	0.24	1	0.22	-0.12
SV	0.67	-0.096	-0.11	-0.13	0.17	-0.14	-0.029	0.099	0.13	-0.18	-0.62	-0.59	-0.61	-0.37	0.22	1	-0.026
E	-0.089	-0.023	0.32	-0.033	0.11	0.13	-0.21	-0.076	0.16	0.079	0.2	0.14	0.11	-0.14	-0.12	-0.026	1

- RA,ER,ERA features are highly negatively correlated with wins.
- RA,ER,ERA features are highly negatively correlated with BB.

- RA,ER,ERA features are highly negatively correlated with sv and sho.
- Sho,bb,2b,SV,HR are positively correlated with win
- ER andERA are posively correlated with RA
- wins feature is highly and positively correlated with saves
- RA,ER,ERA features are highly negatively correlated with saves.
- h,cg having very less correlation with winsRA,ER,ERA features are highly negatively correlated with wins.
- RA,ER,ERA features are highly negatively correlated with BB.
- RA,ER,ERA features are highly negatively correlated with sv and sho.
- Sho,bb,2b,SV,HR are positively correlated with win
- ER andERA are posively correlated with RA
- wins feature is highly and positively correlated with saves
- RA,ER,ERA features are highly negatively correlated with saves.
- h,cg having very less correlation with wins

Finding whether all features which are positively correlated with Wins feature have linear relationship:

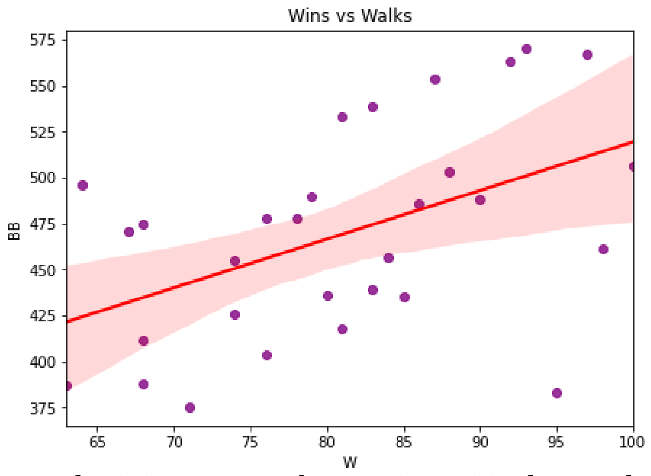
```
plt.figure(figsize=(7,5))
plt.title('Wins vs Saves')
sns.set_style="white_grid"
sns.regplot(x="W",y="SV",data=df,scatter_kws = {'color': 'purple'}, line_kws = {'color': 'r'})
plt.show()


```



In this reg plot it is apparent that SV is positively correlated with W data.most of the datapoints are almost close to best fit line it means that the feature SV and W having linear relationship

```
[282]: plt.figure(figsize=(7,5))
       plt.title('Wins vs Walks')
       sns.set_style="white_grid"
       sns.regplot(x="W",y="BB",data=df,scatter_kws = {'color': 'purple'}, line_kws = {'color': 'r'})
```

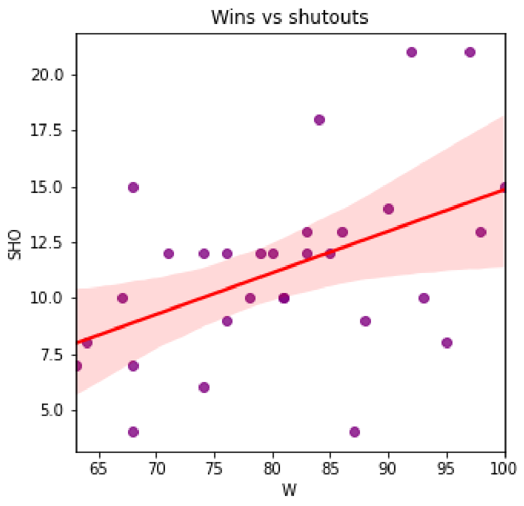
```
[282]: <AxesSubplot:title={'center':'Wins vs Walks'}, xlabel='W', ylabel='BB'>
```



In this reg plot it is apparent that BB is positively correlated with W data. Only few datapoints are close to best fit line it means that the feature BB and W are not having linear relationship. If the value of bb is between 400 to 450 there is more chance to win

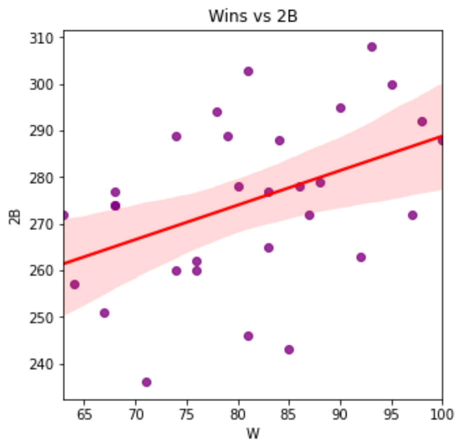
```
: plt.figure(figsize=(5,5))
  plt.title('Wins vs shutouts')
  sns.set_style="white_grid"
  sns.regplot(x="W",y="SHO",data=df,scatter_kws = {'color': 'purple'}, line_kws = {'color': 'r'})
```

```
: <AxesSubplot:title={'center':'Wins vs shutouts'}, xlabel='W', ylabel='SHO'>
```



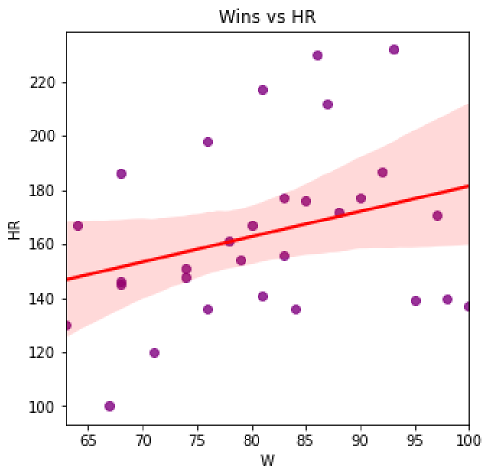
In this reg plot it is apparent that SHO is positively correlated with W data.only few datapoints are close to best fit line it means that the feature SHO and W are not having linear relationship

```
In [5]:  plt.figure(figsize=(5,5))
         plt.title('Wins vs 2B')
         sns.set_style="white_grid"
         sns.regplot(x="W",y="2B",data=df,scatter_kws = {'color': 'purple'}, line_kws = {'color': 'r'})

Out[5]: <AxesSubplot:title={'center':'Wins vs 2B'}, xlabel='W', ylabel='2B'>
```



In this reg plot it is apparent that 2B is positively correlated with W data. Only few data points are close to best fit line it means that the feature 2B and W are not having linear relationship

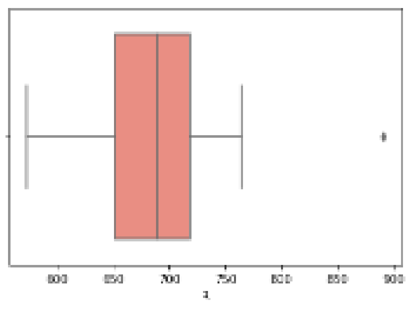
```
6]:  plt.figure(figsize=(5,5))
     plt.title('Wins vs HR')
     sns.set_style="white_grid"
     sns.regplot(x="W",y="HR",data=df,scatter_kws = {'color': 'purple'}, line_kws = {'color': 'r'})

6]: <AxesSubplot:title={'center':'Wins vs HR'}, xlabel='W', ylabel='HR'>
```



In this reg plot it is apparent that HR is positively correlated with W data. Only few datapoints are close to best fit line it means that the feature HR and W are not having linear relationship

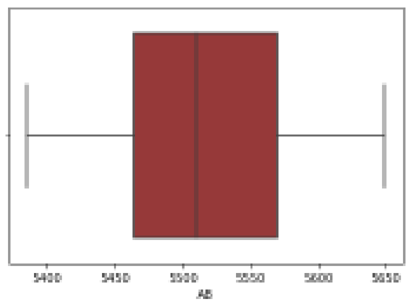
IDENTIFYING OUTLIERS:

```
In [209]:  sns.boxplot(df['R'],color="salmon")
Out[209]: <AxesSubplot:xlabel='R'>
```

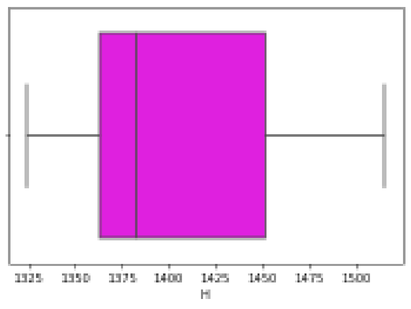


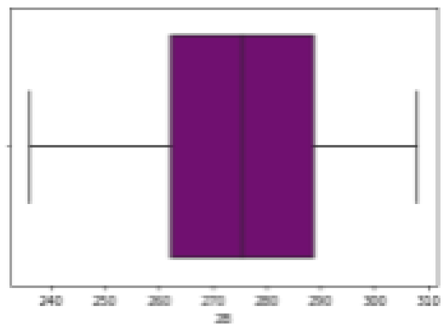
feature R has outlier

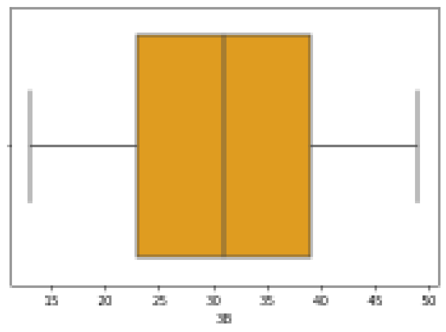
```
In [210]:  sns.boxplot(df['AB'],color="brown")
Out[210]: <AxesSubplot:xlabel='AB'>
```

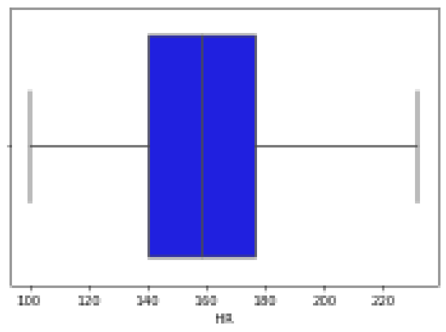


```
In [211]:  sns.boxplot(df['H'],color=(1,0,1))
Out[211]: <AxesSubplot:xlabel='H'>
```

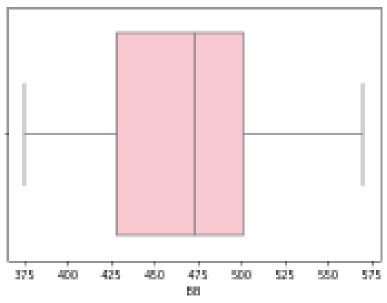


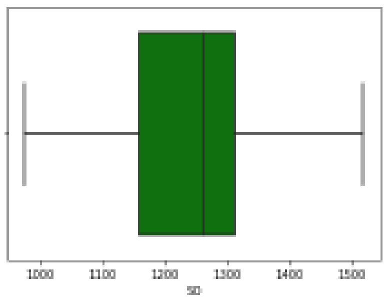
In [212]:
```
sns.boxplot(df['2B'],color="Purple")
```
Out[212]:
```
<AxesSubplot:xlabel='2B'>
```


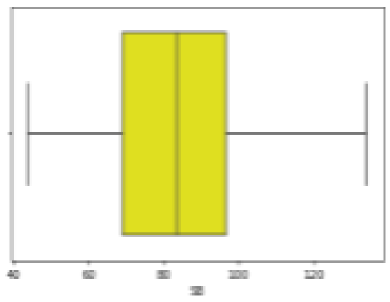
In [213]:
```
sns.boxplot(df['3B'],color="orange")
```
Out[213]:
```
<AxesSubplot:xlabel='3B'>
```


In [214]:
```
sns.boxplot(df['HR'],color=(0,0,1))
```
Out[214]:
```
<AxesSubplot:xlabel='HR'>
```


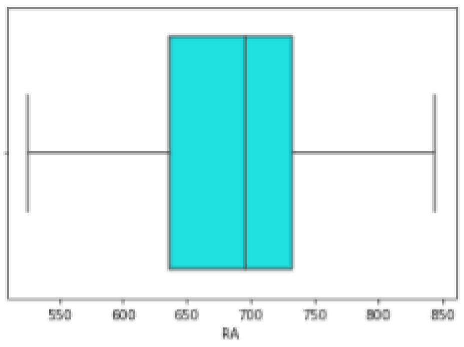
In [215]:
```
sns.boxplot(df['BB'],color="pink")
```
Out[215]:
```
<AxesSubplot:xlabel='BB'>
```

In [215]:
```
sns.boxplot(df['BB'],color="pink")
```
Out[215]:
```
<AxesSubplot:xlabel='BB'>
```


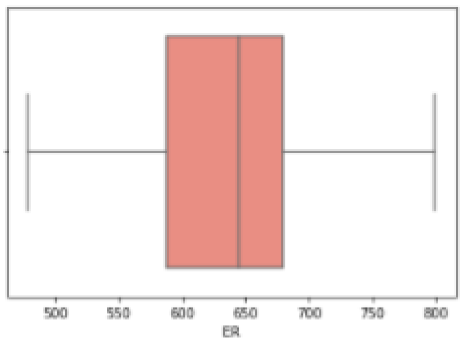
In [221]:
```
sns.boxplot(df['SO'],color="green")
```
Out[221]:
```
<AxesSubplot:xlabel='SO'>
```


In [217]:
```
sns.boxplot(df['SB'],color="yellow")
```
Out[217]:
```
<AxesSubplot:xlabel='SB'>
```


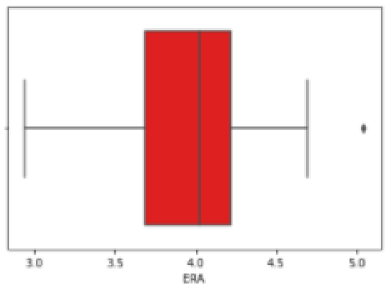
```
In [218]:  sns.boxplot(df['RA'],color=(0,1,1))
Out[218]: <AxesSubplot:xlabel='RA'>
```



```
In [219]:  sns.boxplot(df['ER'],color="salmon")
Out[219]: <AxesSubplot:xlabel='ER'>
```

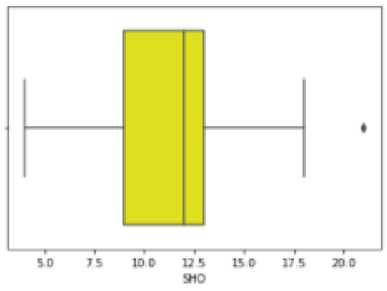


```
In [220]:  sns.boxplot(df['ERA'],color="red")
Out[220]: <AxesSubplot:xlabel='ERA'>
```



feature ERA has outlier

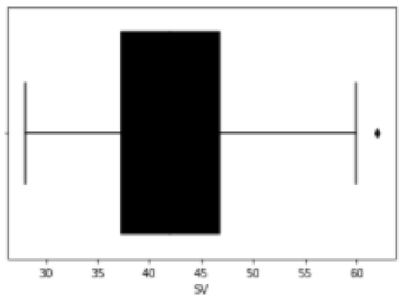
```
In [143]:  sns.boxplot(df['SHO'],color=(1,1,0))
Out[143]: <AxesSubplot:xlabel='SHO'>
```



feature SHO has outlier

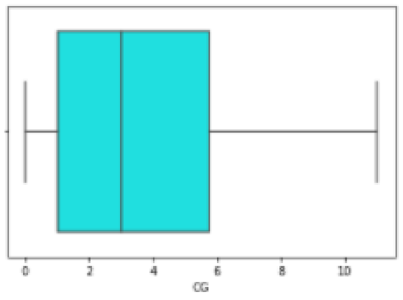
feature SHO has outlier

```
In [144]:  1 sns.boxplot(df['SV'],color=(0,0,0))
Out[144]: <AxesSubplot:xlabel='SV'>
```

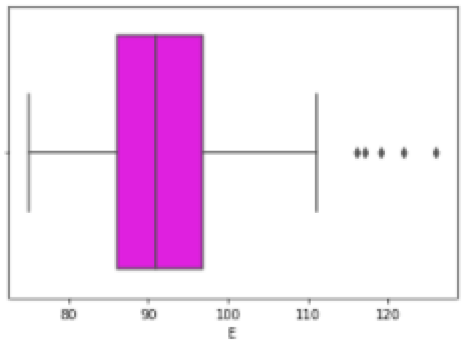


feature SV has outlier

```
In [145]:  1 sns.boxplot(df['CG'],color=(0,1,1))
Out[145]: <AxesSubplot:xlabel='CG'>
```

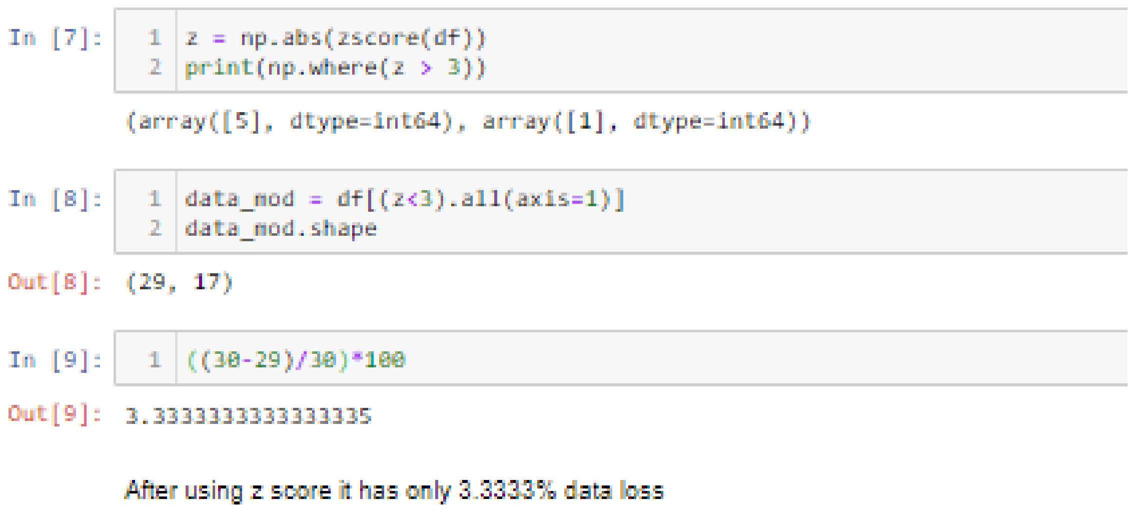


```
In [146]:  1 sns.boxplot(df['E'],color=(1,0,1))
Out[146]: <AxesSubplot:xlabel='E'>
```



The features R,ERA,SHO,SV,E are having outliers

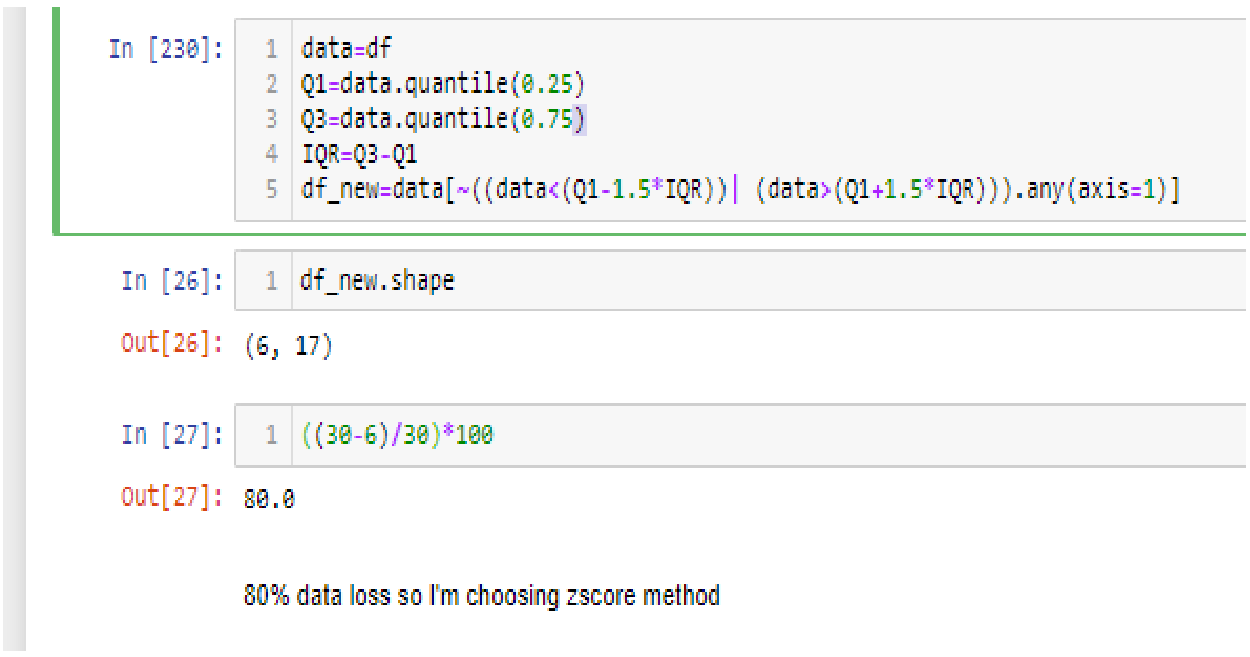
REMOVING OUTLIERS



```
In [7]:  z = np.abs(zscore(df))
         print(np.where(z > 3))

(array([5], dtype=int64), array([1], dtype=int64))

In [8]:  data_mod = df[(z<3).all(axis=1)]
         data_mod.shape

Out[8]: (29, 17)

In [9]:  ((30-29)/30)*100

Out[9]: 3.3333333333333335
```

After using z score it has only 3.3333% data loss

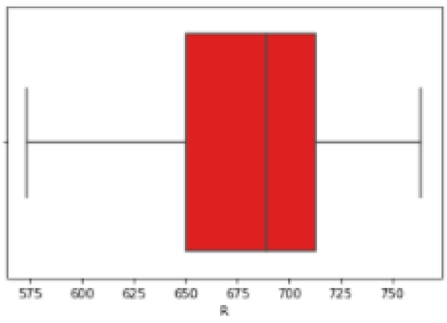
IQR TO REMOVE OUTLIERS:

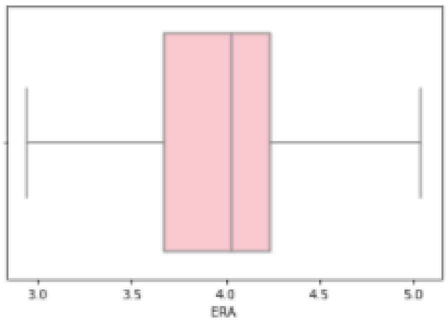


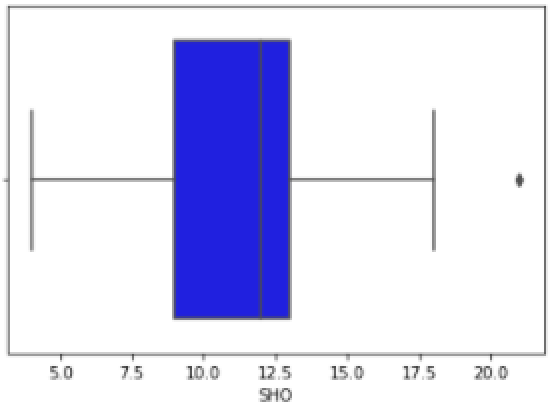
```
In [230]:  data=df
           Q1=data.quantile(0.25)
           Q3=data.quantile(0.75)
           IQR=Q3-Q1
           df_new=data[~((data<(Q1-1.5*IQR))| (data>(Q1+1.5*IQR))).any(axis=1)]

In [26]:  df_new.shape

Out[26]: (6, 17)

In [27]:  ((30-6)/30)*100

Out[27]: 80.0
```

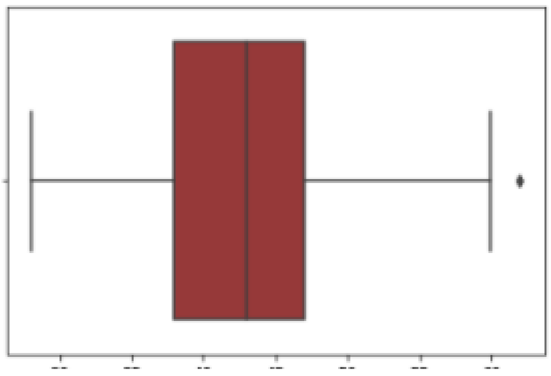
80% data loss so I'm choosing zscore method

AFTER USING ZSCORE:

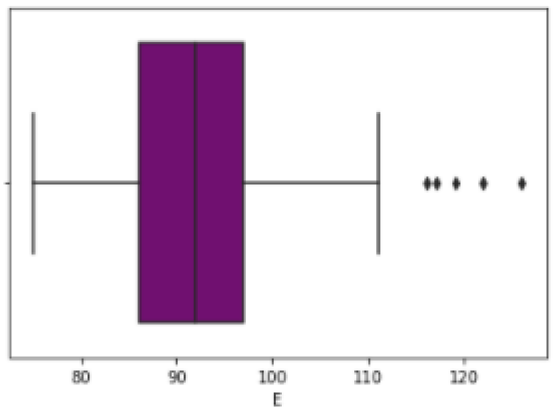
In [225]:
```
sns.boxplot(data_mod['R'],color="red")
```
Out[225]:
```
<AxesSubplot:xlabel='R'>
```


In [226]:
```
sns.boxplot(data_mod['ERA'],color="pink")
```
Out[226]:
```
<AxesSubplot:xlabel='ERA'>
```


In [227]:
```
sns.boxplot(data_mod['SHO'],color="blue")
```
Out[227]:
```
<AxesSubplot:xlabel='SHO'>
```


In [228]:
```
sns.boxplot(data_mod['SV'],color="brown")
```
Out[228]:
```
<AxesSubplot:xlabel='SV'>
```


```
In [229]:  sns.boxplot(data_mod['E'],color="purple")

Out[229]: <AxesSubplot:xlabel='E'>
```



After using z_score the outliers in the features R and ERA are removed but the features SV,SHO,E are still having outliers

HANDLING SKEWNESS:

```
In [10]:  for col in enumerate(list(data_mod.columns.values)):
              print(col[1],"=",data_mod[col[1]].skew())

W = 0.11901344569985461
R = -0.21536363420992782
AB = 0.16957316834729352
H = 0.7837722117274881
2B = -0.335303936110201
3B = 0.09012434653848651
HR = 0.45086158125803544
BB = 0.15119282971519954
SO = -0.2338149185462262
SB = 0.4949657663368456
RA = 0.018155177145956613
ER = 0.018460990156758887
ERA = 0.016693217783651695
CG = 0.8549795901105167
SHO = 0.5269430585305683
SV = 0.6274804879503074
E = 0.8402711976867623
```

the features H,CG,SHO,SV,E having skewness

To stabilize variance, make the data more normal distribution like, improve the validity of measures of association and to remove skewness I have used power transformation

```
n [18]:  col_s=['H','CG','SHO','SV','E']
         data_clean=data_mod
         from sklearn.preprocessing import power_transform
         data_clean[col_s]=power_transform (data_mod[col_s])
```

```
n [19]:  for col in enumerate(list(data_clean.columns.values)):
             print(col[1],"=",data_clean[col[1]].skew())
```

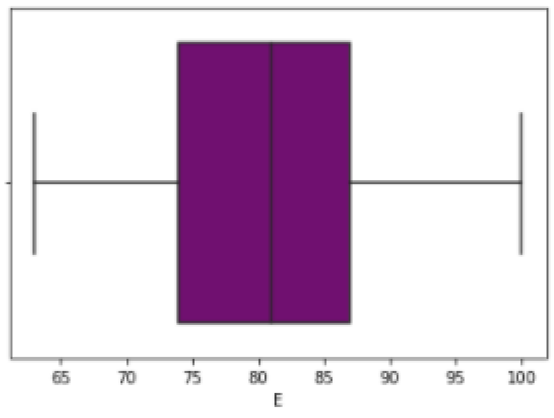
```
W = 0.11901344569985461
R = -0.21536363420992782
AB = 0.16957316834729352
H = 0
2B = -0.335303936110201
3B = 0.09012434653848651
HR = 0.45086158125803544
BB = 0.15119282971519954
SO = -0.2338149185462262
SB = 0.4949657663368456
RA = 0.018155177145956613
ER = 0.018460990156758887
ERA = 0.016693217783651695
CG = -0.045947323970913174
SHO = 0.000529365035686870 7
SV = -0.0009249344497408174
E = 0.06558547868786976
```

Skewness is completely removed

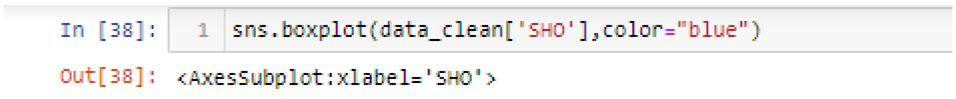
I have used Power transform on skewed data to make it symmetric, and then fit it to a symmetric distribution

```
:  sns.boxplot(data_clean['E'],color="purple")
```

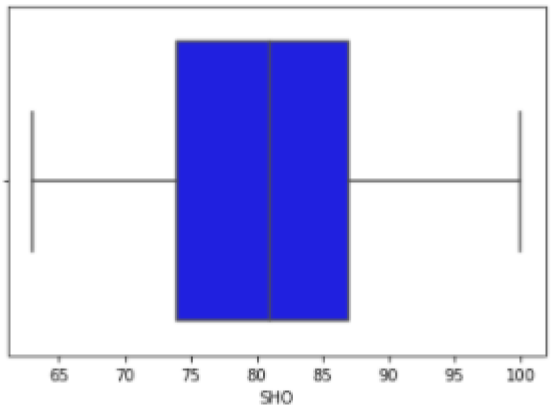
```
: <AxesSubplot:xlabel='E'>
```

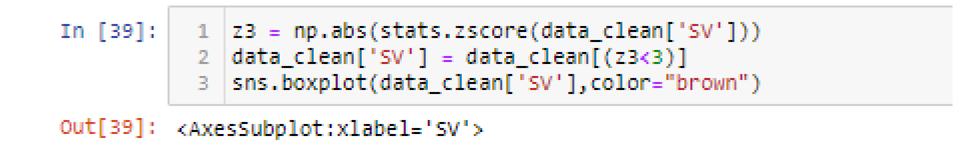


Using zscore the outliers in the feature E is completely removed and the data loss after removing outlier is 3.333333%

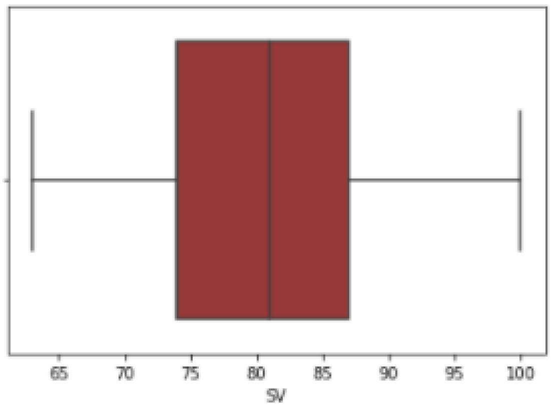


```
In [38]:  sns.boxplot(data_clean['SHO'],color="blue")
Out[38]: <AxesSubplot:xlabel='SHO'>
```





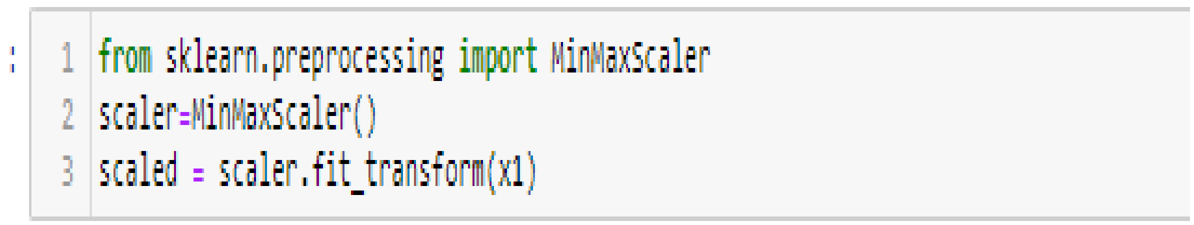
```
In [39]:  z3 = np.abs(stats.zscore(data_clean['SV']))
          data_clean['SV'] = data_clean[(z3<3)]
          sns.boxplot(data_clean['SV'],color="brown")
Out[39]: <AxesSubplot:xlabel='SV'>
```



I have used zscore to remove outliers in the features SV, SHO and E.

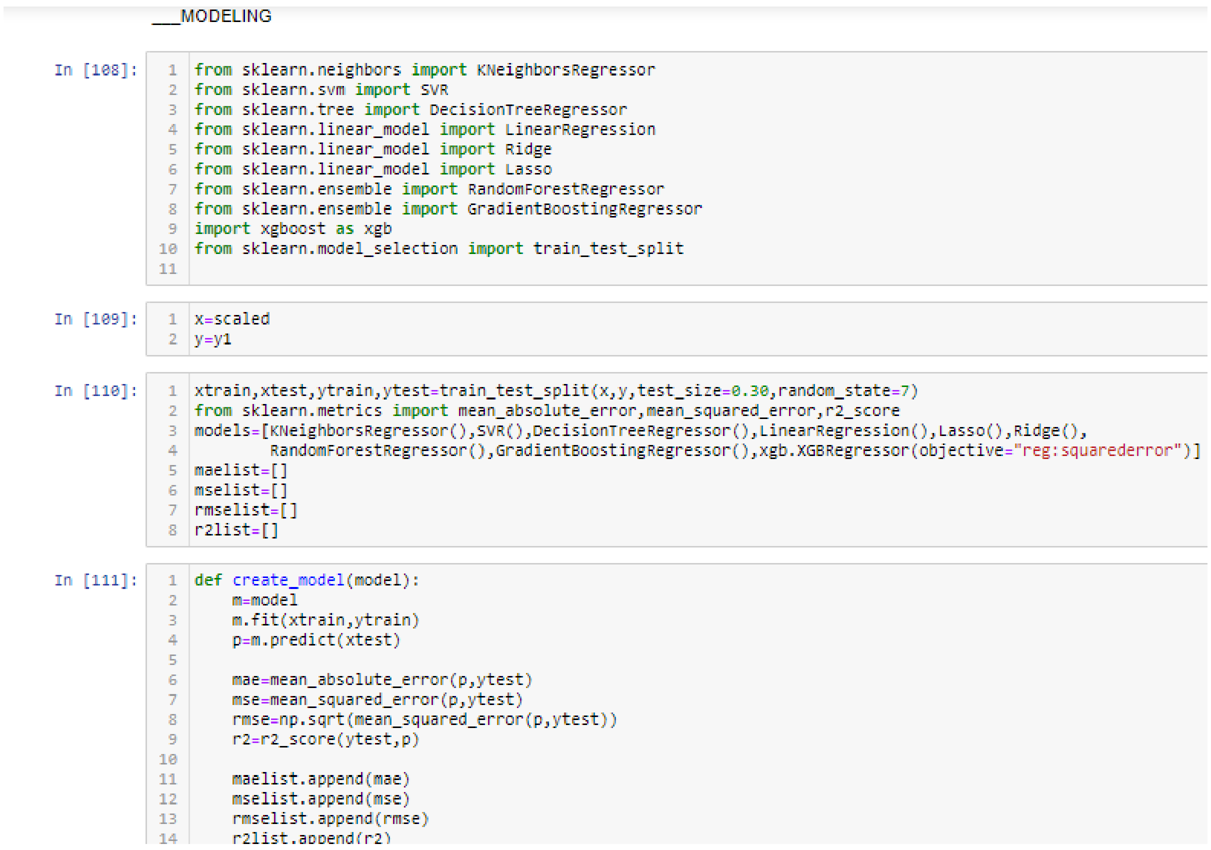
SCALING:

__feature scaling transforming un scaled data into scaled data using min max scalining technique



```
from sklearn.preprocessing import MinMaxScaler
scaler=MinMaxScaler()
scaled = scaler.fit_transform(x1)
```

MODELLING:



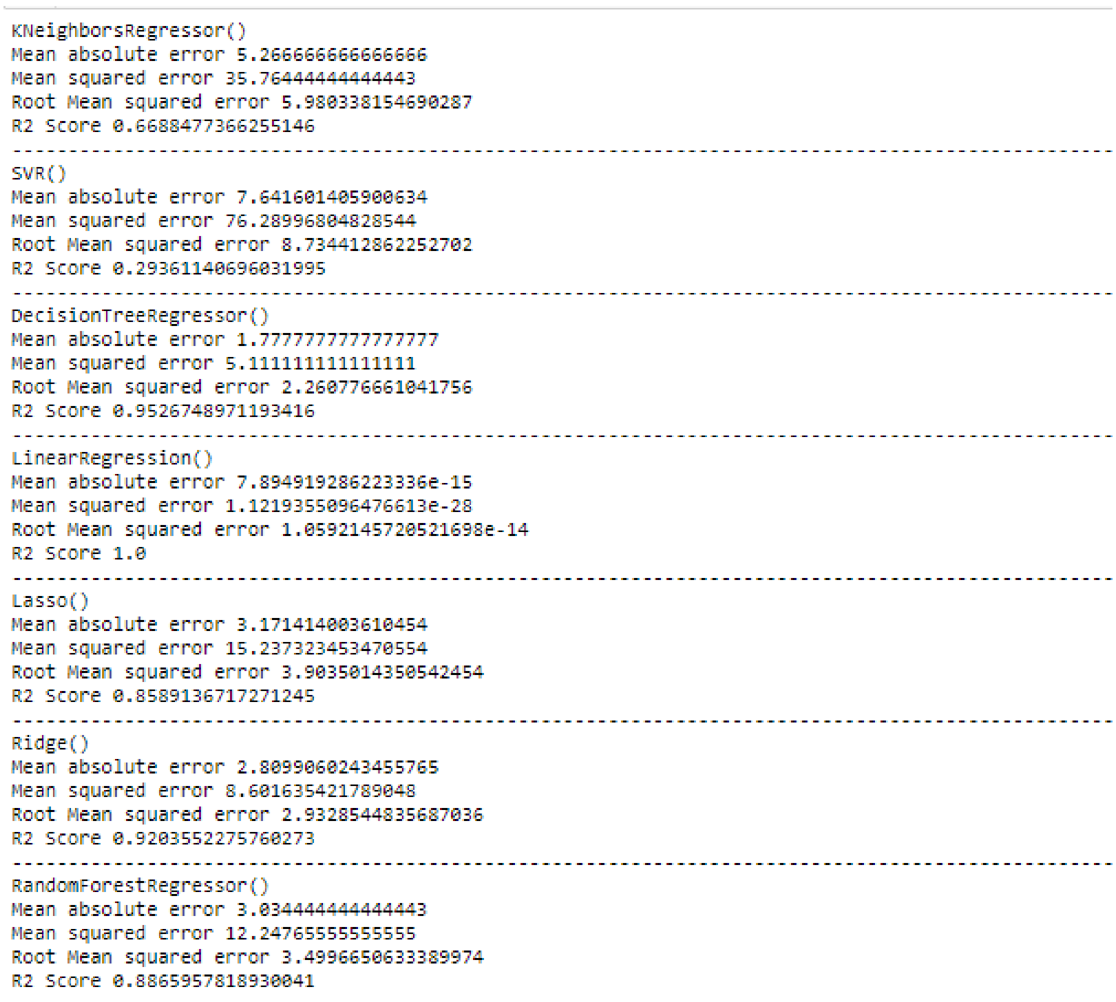
___MODELING

```
In [108]:
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
import xgboost as xgb
from sklearn.model_selection import train_test_split

```

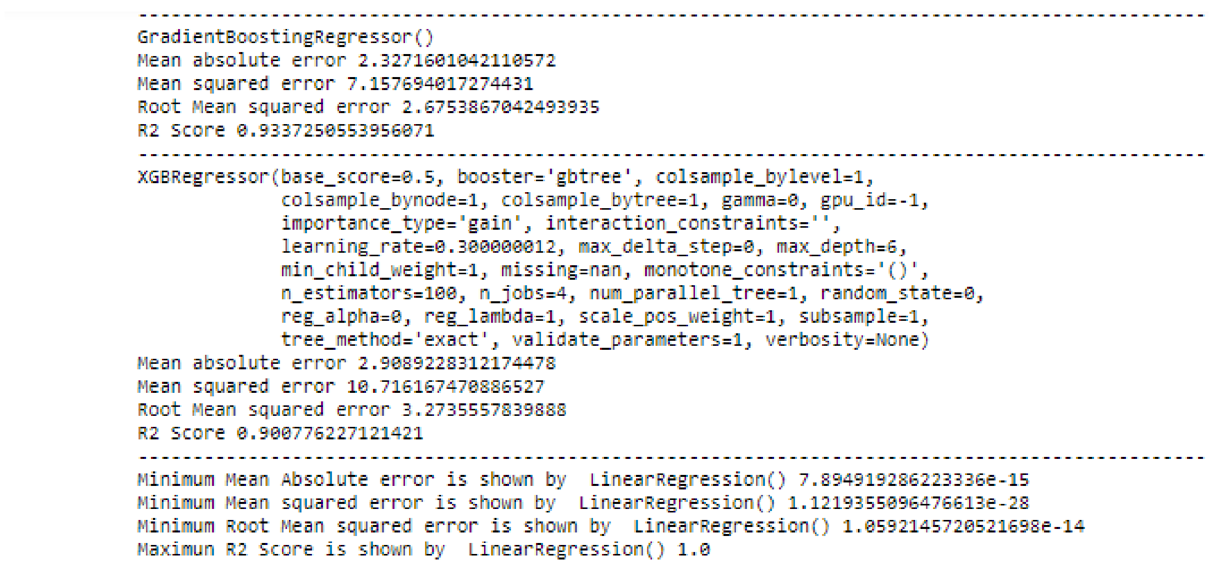
```
In [109]:
x=scaled
y=y1
```

```
In [110]:
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.30,random_state=7)
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
models=[KNeighborsRegressor(),SVR(),DecisionTreeRegressor(),LinearRegression(),Lasso(),Ridge(),
        RandomForestRegressor(),GradientBoostingRegressor(),xgb.XGBRegressor(objective="reg:squarederror")]
maelist=[]
mselist=[]
rmselist=[]
r2list=[]
```

```
In [111]:
def create_model(model):
    m=model
    m.fit(xtrain,ytrain)
    p=m.predict(xtest)

    mae=mean_absolute_error(p,ytest)
    mse=mean_squared_error(p,ytest)
    rmse=np.sqrt(mean_squared_error(p,ytest))
    r2=r2_score(ytest,p)

    maelist.append(mae)
    mselist.append(mse)
    rmselist.append(rmse)
    r2list.append(r2)
```



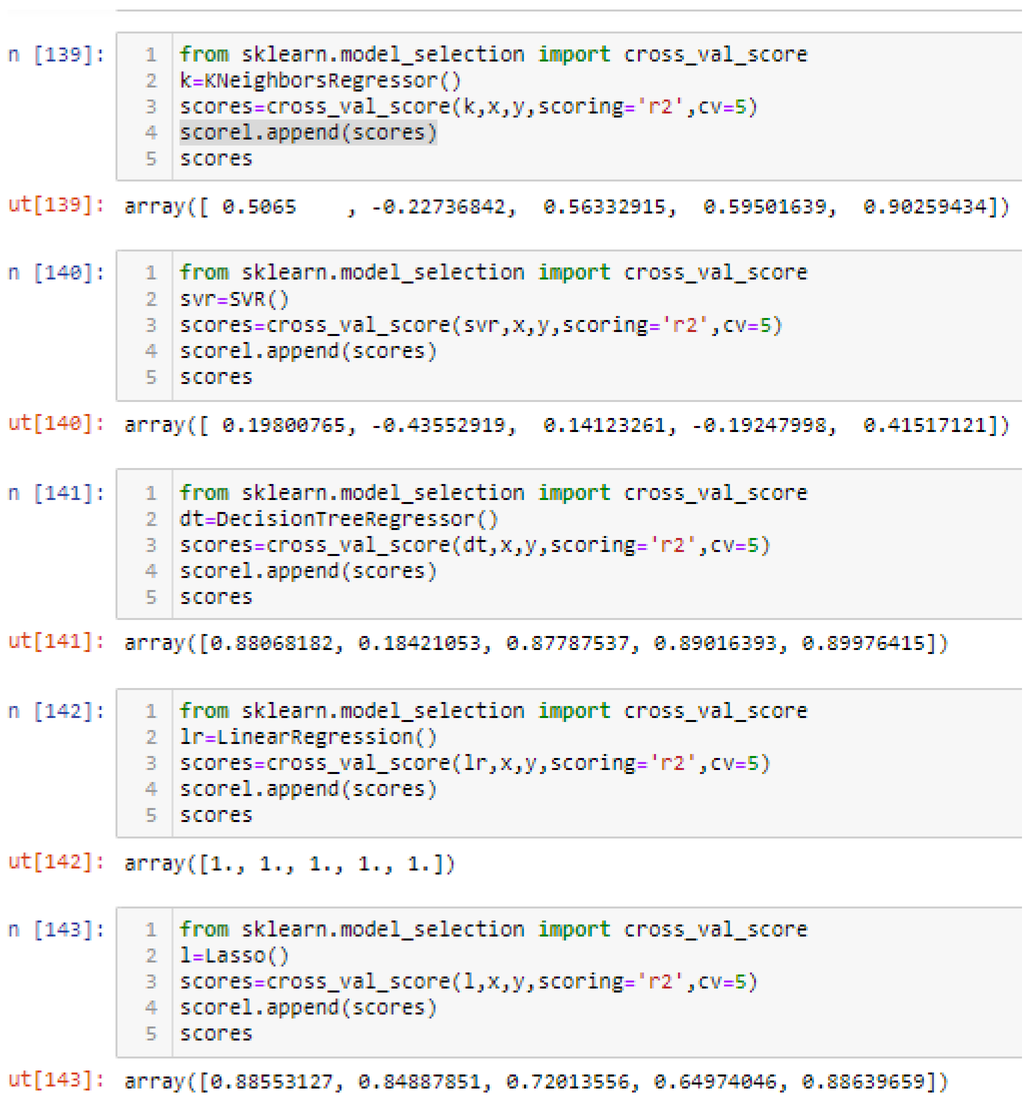
```
KNeighborsRegressor()
Mean absolute error 5.266666666666666
Mean squared error 35.76444444444443
Root Mean squared error 5.980338154690287
R2 Score 0.6688477366255146
----------------------------------------------------------------------------------------------------
SVR()
Mean absolute error 7.641601405900634
Mean squared error 76.28996804828544
Root Mean squared error 8.734412862252702
R2 Score 0.2936114069603199 5
----------------------------------------------------------------------------------------------------
DecisionTreeRegressor()
Mean absolute error 1.7777777777777777
Mean squared error 5.111111111111111
Root Mean squared error 2.260776661041756
R2 Score 0.9526748971193416
----------------------------------------------------------------------------------------------------
LinearRegression()
Mean absolute error 7.894919286223336e-15
Mean squared error 1.1219355096476613e-28
Root Mean squared error 1.0592145720521698e-14
R2 Score 1.0
----------------------------------------------------------------------------------------------------
Lasso()
Mean absolute error 3.171414003610454
Mean squared error 15.237323453470554
Root Mean squared error 3.9035014350542454
R2 Score 0.8589136717271245
----------------------------------------------------------------------------------------------------
Ridge()
Mean absolute error 2.8099060243455765
Mean squared error 8.601635421789048
Root Mean squared error 2.9328544835687036
R2 Score 0.9203552275760273
----------------------------------------------------------------------------------------------------
RandomForestRegressor()
Mean absolute error 3.034444444444443
Mean squared error 12.24765555555555
Root Mean squared error 3.4996650633389974
R2 Score 0.8865957818930041
```



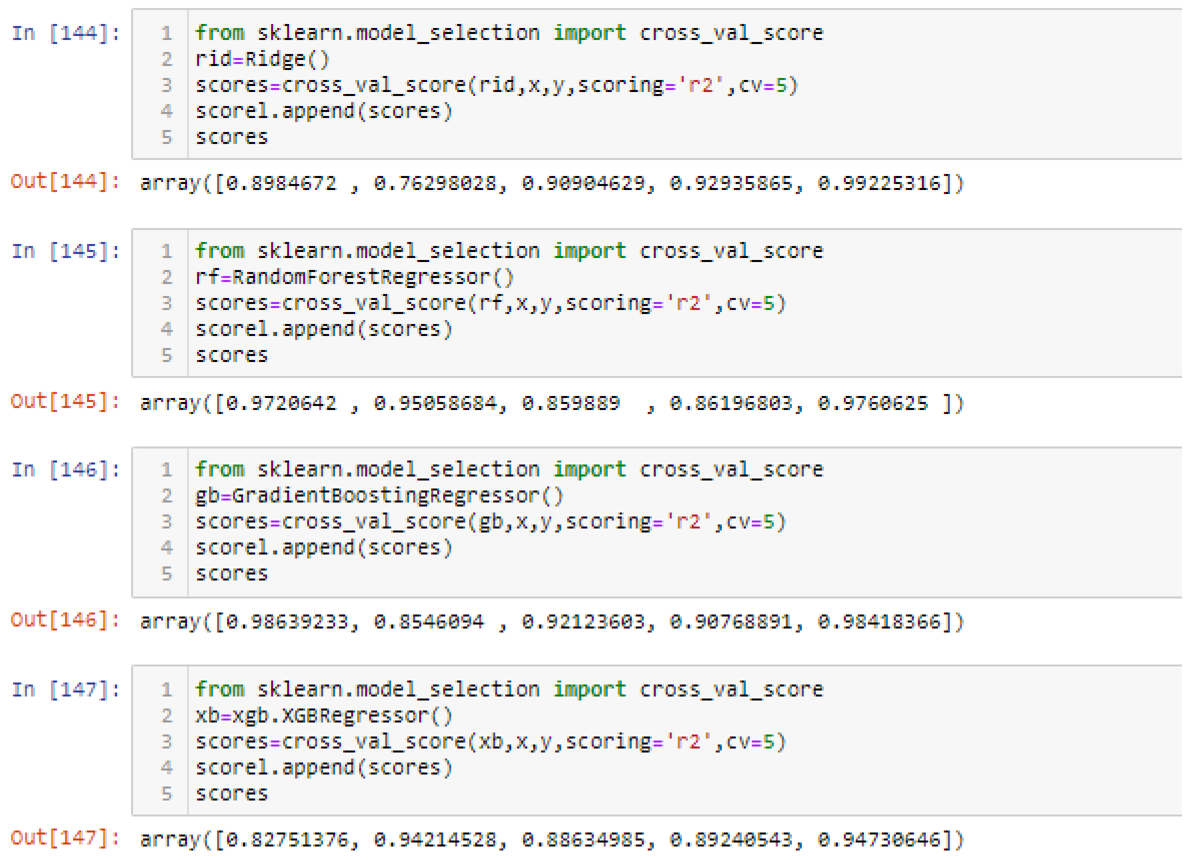
```
----------------------------------------------------------------------------------------------------
GradientBoostingRegressor()
Mean absolute error 2.3271601042110572
Mean squared error 7.157694017274431
Root Mean squared error 2.6753867042493935
R2 Score 0.9337250553956071
----------------------------------------------------------------------------------------------------
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.300000012, max_delta_step=0, max_depth=6,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=100, n_jobs=4, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
Mean absolute error 2.9089228312174478
Mean squared error 10.716167470886527
Root Mean squared error 3.2735557839888
R2 Score 0.900776227121421
----------------------------------------------------------------------------------------------------
Minimum Mean Absolute error is shown by  LinearRegression() 7.894919286223336e-15
Minimum Mean squared error is shown by  LinearRegression() 1.1219355096476613e-28
Minimum Root Mean squared error is shown by  LinearRegression() 1.0592145720521698e-14
Maximun R2 Score is shown by  LinearRegression() 1.0
```

Cross Validation:

In order to avoid over fitting, Cross-validation is used to estimate the skill of a machine learning model on unseen data.



```
n [139]: from sklearn.model_selection import cross_val_score
         k=KNeighborsRegressor()
         scores=cross_val_score(k,x,y,scoring='r2',cv=5)
         scorel.append(scores)
         scores

ut[139]: array([ 0.5065    , -0.22736842,  0.56332915,  0.59501639,  0.90259434])

n [140]: from sklearn.model_selection import cross_val_score
         svr=SVR()
         scores=cross_val_score(svr,x,y,scoring='r2',cv=5)
         scorel.append(scores)
         scores

ut[140]: array([ 0.19800765, -0.43552919,  0.14123261, -0.19247998,  0.41517121])

n [141]: from sklearn.model_selection import cross_val_score
         dt=DecisionTreeRegressor()
         scores=cross_val_score(dt,x,y,scoring='r2',cv=5)
         scorel.append(scores)
         scores

ut[141]: array([0.88068182, 0.18421053, 0.87787537, 0.89016393, 0.89976415])

n [142]: from sklearn.model_selection import cross_val_score
         lr=LinearRegression()
         scores=cross_val_score(lr,x,y,scoring='r2',cv=5)
         scorel.append(scores)
         scores

ut[142]: array([1., 1., 1., 1., 1.])

n [143]: from sklearn.model_selection import cross_val_score
         l=Lasso()
         scores=cross_val_score(l,x,y,scoring='r2',cv=5)
         scorel.append(scores)
         scores

ut[143]: array([0.88553127, 0.84887851, 0.72013556, 0.64974046, 0.88639659])
```



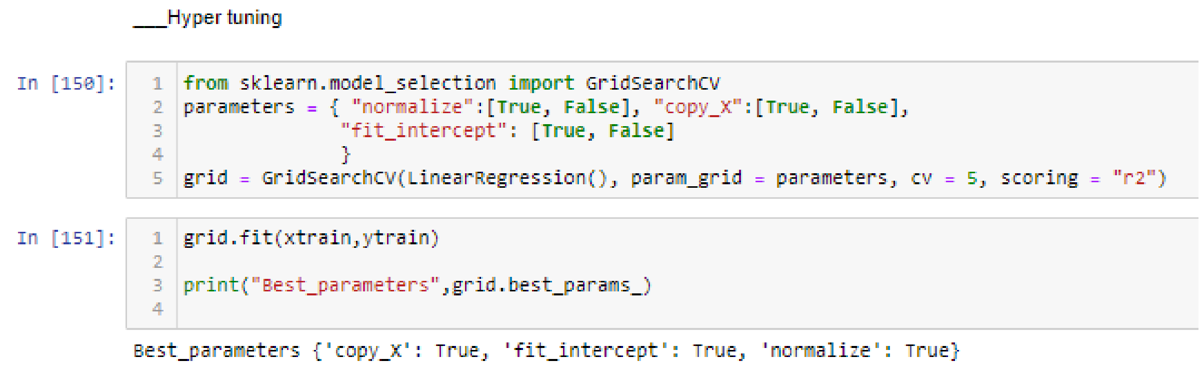
```
In [144]: from sklearn.model_selection import cross_val_score
          rid=Ridge()
          scores=cross_val_score(rid,x,y,scoring='r2',cv=5)
          scorel.append(scores)
          scores

Out[144]: array([0.8984672 , 0.76298028, 0.90904629, 0.92935865, 0.99225316])

In [145]: from sklearn.model_selection import cross_val_score
          rf=RandomForestRegressor()
          scores=cross_val_score(rf,x,y,scoring='r2',cv=5)
          scorel.append(scores)
          scores

Out[145]: array([0.9720642 , 0.95058684, 0.859889  , 0.86196803, 0.9760625 ])

In [146]: from sklearn.model_selection import cross_val_score
          gb=GradientBoostingRegressor()
          scores=cross_val_score(gb,x,y,scoring='r2',cv=5)
          scorel.append(scores)
          scores

Out[146]: array([0.98639233, 0.8546094 , 0.92123603, 0.90768891, 0.98418366])

In [147]: from sklearn.model_selection import cross_val_score
          xb=xgb.XGBRegressor()
          scores=cross_val_score(xb,x,y,scoring='r2',cv=5)
          scorel.append(scores)
          scores

Out[147]: array([0.82751376, 0.94214528, 0.88634985, 0.89240543, 0.94730646])
```

Difference of predicted model and cross validation score:

- KNeighborsRegressor - 0.2337466
- SVR()-0.12155981
- DecisionTreeRegressor()-0.05291075
- LinearRegression() -0
- Lasso - 0.02748291
- Ridge()-0.07189794
- RandomForestRegressor()-0.08946672
- GradientBoostingRegressor()-0.0504586
- XGBRegressor-0.046530

from the observation Linear regression model model has least difference so I'm selecting Linear regression as best model

Hyper Tuning:



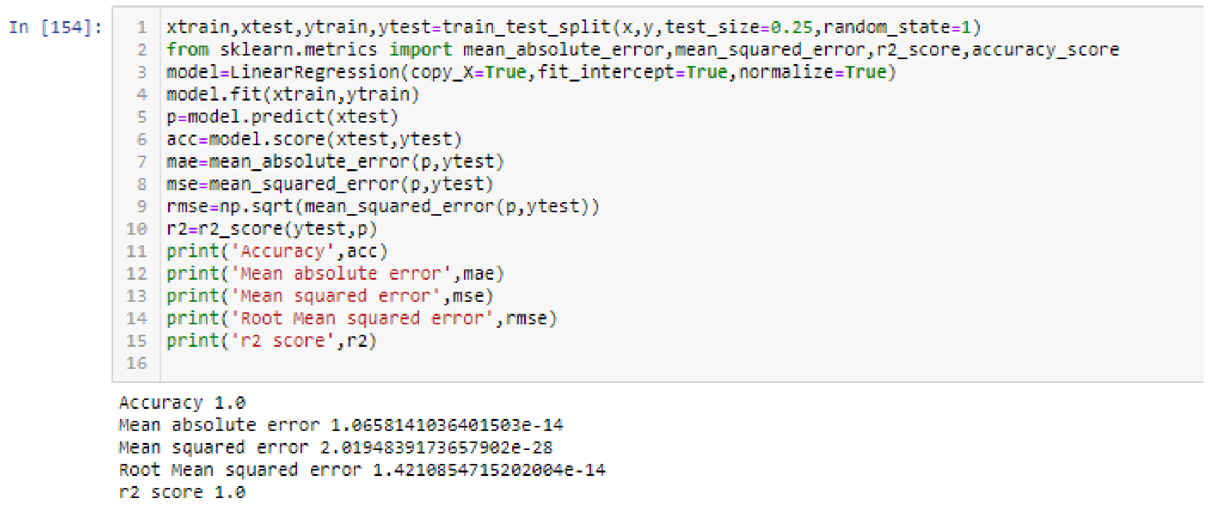
___Hyper tuning

```
In [150]:
from sklearn.model_selection import GridSearchCV
parameters = { "normalize":[True, False], "copy_X":[True, False],
               "fit_intercept": [True, False]
               }
grid = GridSearchCV(LinearRegression(), param_grid = parameters, cv = 5, scoring = "r2")
```

```
In [151]:
grid.fit(xtrain,ytrain)

print("Best_parameters",grid.best_params_)

```

```
Best_parameters {'copy_X': True, 'fit_intercept': True, 'normalize': True}
```

Best parameters: **{**'copy_X': True, 'fit_intercept': True, 'normalize': True}
Modelling using best parameter and best model:



```
In [154]:
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.25,random_state=1)
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score,accuracy_score
model=LinearRegression(copy_X=True,fit_intercept=True,normalize=True)
model.fit(xtrain,ytrain)
p=model.predict(xtest)
acc=model.score(xtest,ytest)
mae=mean_absolute_error(p,ytest)
mse=mean_squared_error(p,ytest)
rmse=np.sqrt(mean_squared_error(p,ytest))
r2=r2_score(ytest,p)
print('Accuracy',acc)
print('Mean absolute error',mae)
print('Mean squared error',mse)
print('Root Mean squared error',rmse)
print('r2 score',r2)

```

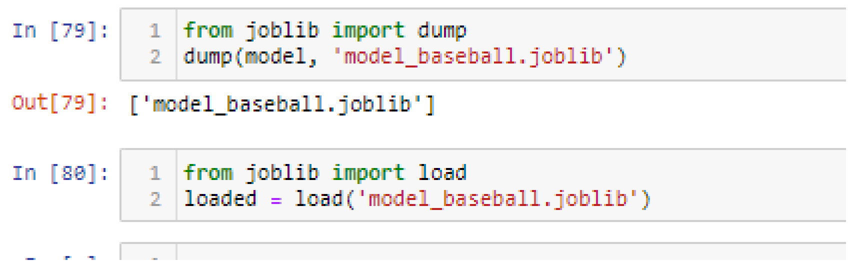
```
Accuracy 1.0
Mean absolute error 1.0658141036401503e-14
Mean squared error 2.0194839173657902e-28
Root Mean squared error 1.4210854715202004e-14
r2 score 1.0
```

Final model after hyper tuning its retaining 100%accuracy and error values got reduced

Conclusion:

I have developed a model to predict number of wins with 100% accuracy

Saving the model



```
In [79]:
from joblib import dump
dump(model, 'model_baseball.joblib')
```

```
Out[79]: ['model_baseball.joblib']
```

```
In [80]:
from joblib import load
loaded = load('model_baseball.joblib')
```