Baseball Case Study Problem Statement: This dataset utilizes data from 2014 Major League Baseball seasons in order to develop an algorithm that predicts the number of wins for a given team in the 2015 season based on several different indicators of success. 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: Runs, At Bats, Hits, Doubles, Triples, Homeruns, Walks, Strikeouts, Stolen Bases, Runs Allowed, Earned Runs, Earned Run Average (ERA), Shutouts, Saves, Complete Games and Errors

-- Output: Number of predicted wins (W)

To understand the columns meaning, follow the link given below to understand the baseball statistics: https://en.wikipedia.org/wiki/Baseball_statistics (https://en.wikipedia.org/wiki/Baseball_statistics)

In [13]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
import seaborn as sns
warnings.filterwarnings('ignore')
```

In [2]:
```
df=pd.read_csv("baseball.csv")
```

In [3]:
```
df.head()
```

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

In [4]:
```
df.dtypes
```

Out[4]:
```
W        int64
R        int64
AB       int64
H        int64
2B       int64
3B       int64
HR       int64
BB       int64
SO       int64
SB       int64
RA       int64
ER       int64
ERA    float64
CG       int64
SHO      int64
SV       int64
E        int64
dtype: object
```

Exploratory DataAnalysis

In [5]:
```
df.shape
```

Out[5]:
```
(30, 17)
```

In [6]:
```
df.dtypes
```

Out[6]:
```
W        int64
R        int64
AB       int64
H        int64
2B       int64
3B       int64
HR       int64
BB       int64
SO       int64
SB       int64
RA       int64
ER       int64
ERA    float64
CG       int64
SHO      int64
SV       int64
E        int64
dtype: object
```

In [8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 17 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   W       30 non-null     int64  
 1   R       30 non-null     int64  
 2   AB      30 non-null     int64  
 3   H       30 non-null     int64  
 4   2B      30 non-null     int64  
 5   3B      30 non-null     int64  
 6   HR      30 non-null     int64  
 7   BB      30 non-null     int64  
 8   SO      30 non-null     int64  
 9   SB      30 non-null     int64  
 10  RA      30 non-null     int64  
 11  ER      30 non-null     int64  
 12  ERA     30 non-null     float64
 13  CG      30 non-null     int64  
 14  SHO     30 non-null     int64  
 15  SV      30 non-null     int64  
 16  E       30 non-null     int64  
dtypes: float64(1), int64(16)
memory usage: 4.1 KB
```

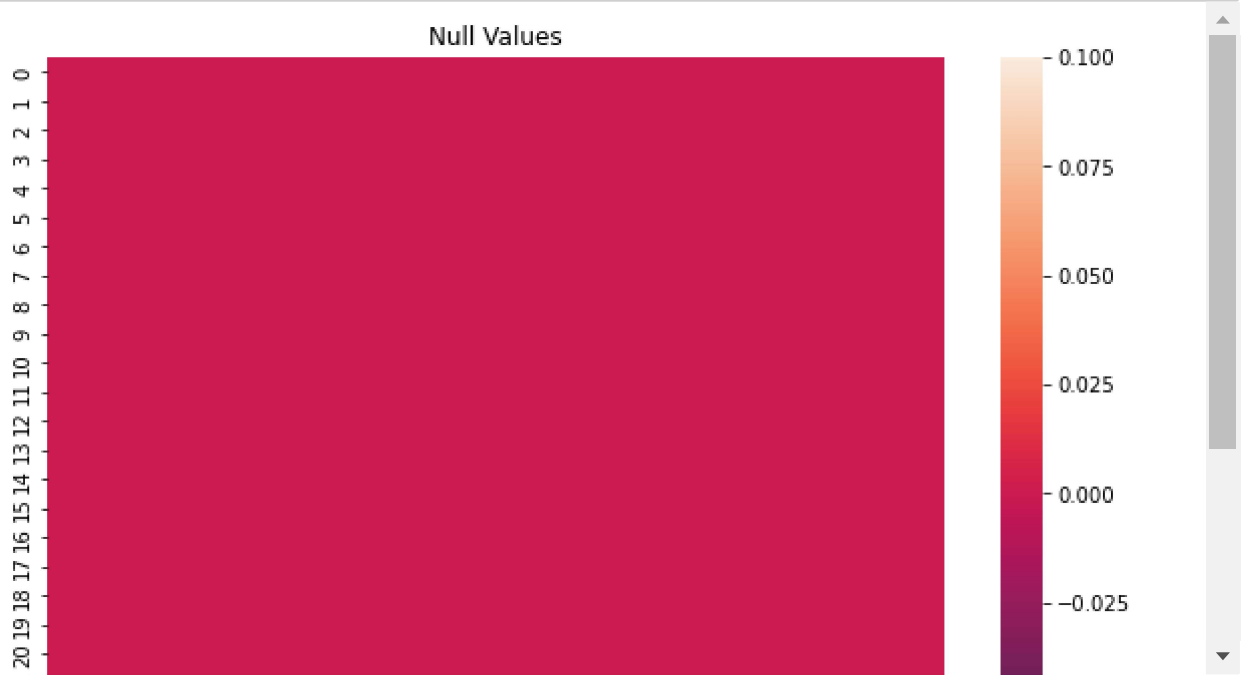
In [9]:
```
df.nunique()
```

Out[9]:
```
W      24
R      28
AB     29
H      29
2B     22
3B     23
HR     27
BB     29
SO     29
SB     27
RA     30
ER     30
ERA    30
CG      9
SHO    12
SV     20
E      21
dtype: int64
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
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
```

In [14]:
```
plt.figure(figsize=[10,8])
sns.heatmap(df.isnull())
plt.title('Null Values')
plt.show()
```



In [15]:
```
df['W'].value_counts
```

Out[15]:
```
<bound method IndexOpsMixin.value_counts of 0      95
1      83
2      81
3      76
4      74
5      93
6      87
7      81
8      80
9      78
10     88
11     86
12     85
13     76
14     68
15    100
16     98
17     97
18     68
19     64
20     90
21     83
22     71
23     67
24     63
25     92
26     84
27     79
28     74
29     68
Name: W, dtype: int64>
```

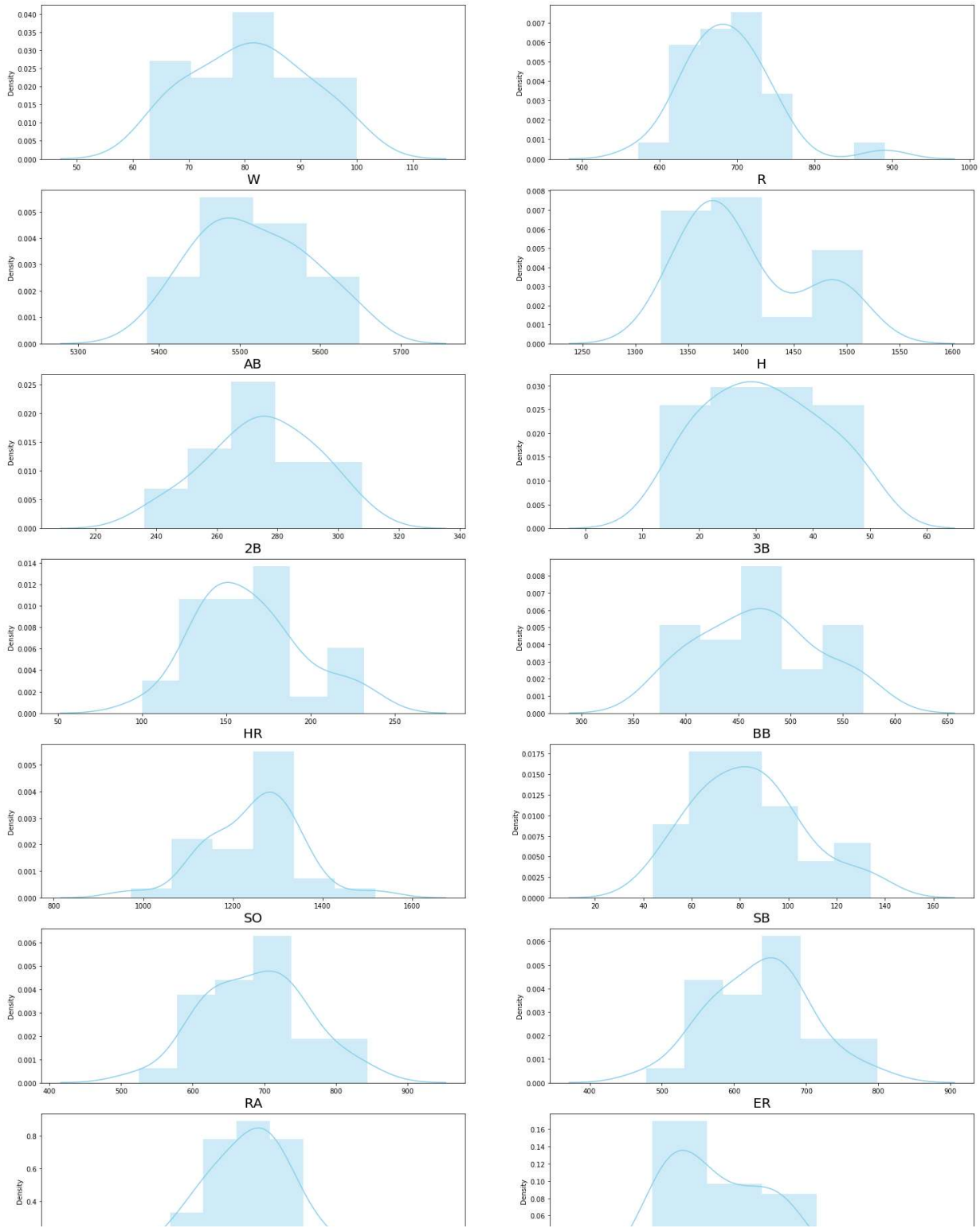
In [16]:
```
df.describe()
```

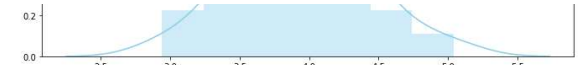
Out[16]:

	W	R	AB	H	2B	3B	HR	
count	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.000000	30.0
mean	80.966667	688.233333	5516.266667	1403.533333	274.733333	31.300000	163.633333	469.1
std	10.453455	58.761754	70.467372	57.140923	18.095405	10.452355	31.823309	57.0
min	63.000000	573.000000	5385.000000	1324.000000	236.000000	13.000000	100.000000	375.0
25%	74.000000	651.250000	5464.000000	1363.000000	262.250000	23.000000	140.250000	428.2
50%	81.000000	689.000000	5510.000000	1382.500000	275.500000	31.000000	158.500000	473.0
75%	87.750000	718.250000	5570.000000	1451.500000	288.750000	39.000000	177.000000	501.2
max	100.000000	891.000000	5649.000000	1515.000000	308.000000	49.000000	232.000000	570.0

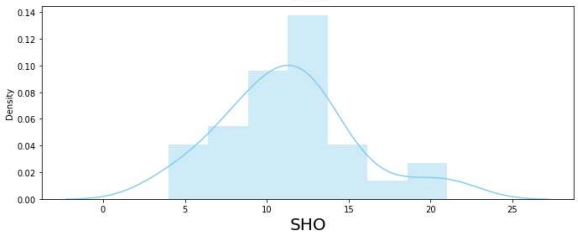
Data visualisation

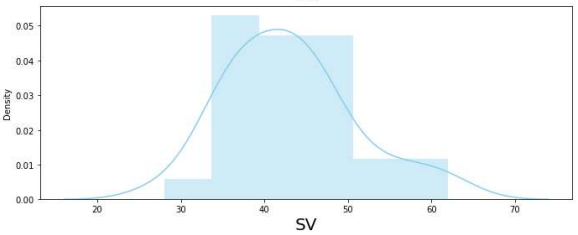
In [17]:
```
plt.figure(figsize = (25,50), facecolor = 'white')
plotnumber = 1

for column in df:
    if plotnumber <=17:
        ax = plt.subplot(10,2,plotnumber)
        sns.distplot(df[column], color = 'skyblue')
        plt.xlabel(column,fontsize=20)

    plotnumber +=1
plt.show()
```

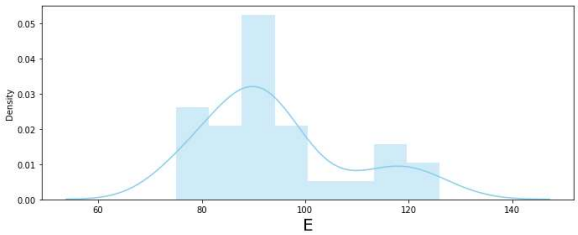



0.2
0.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
ERA

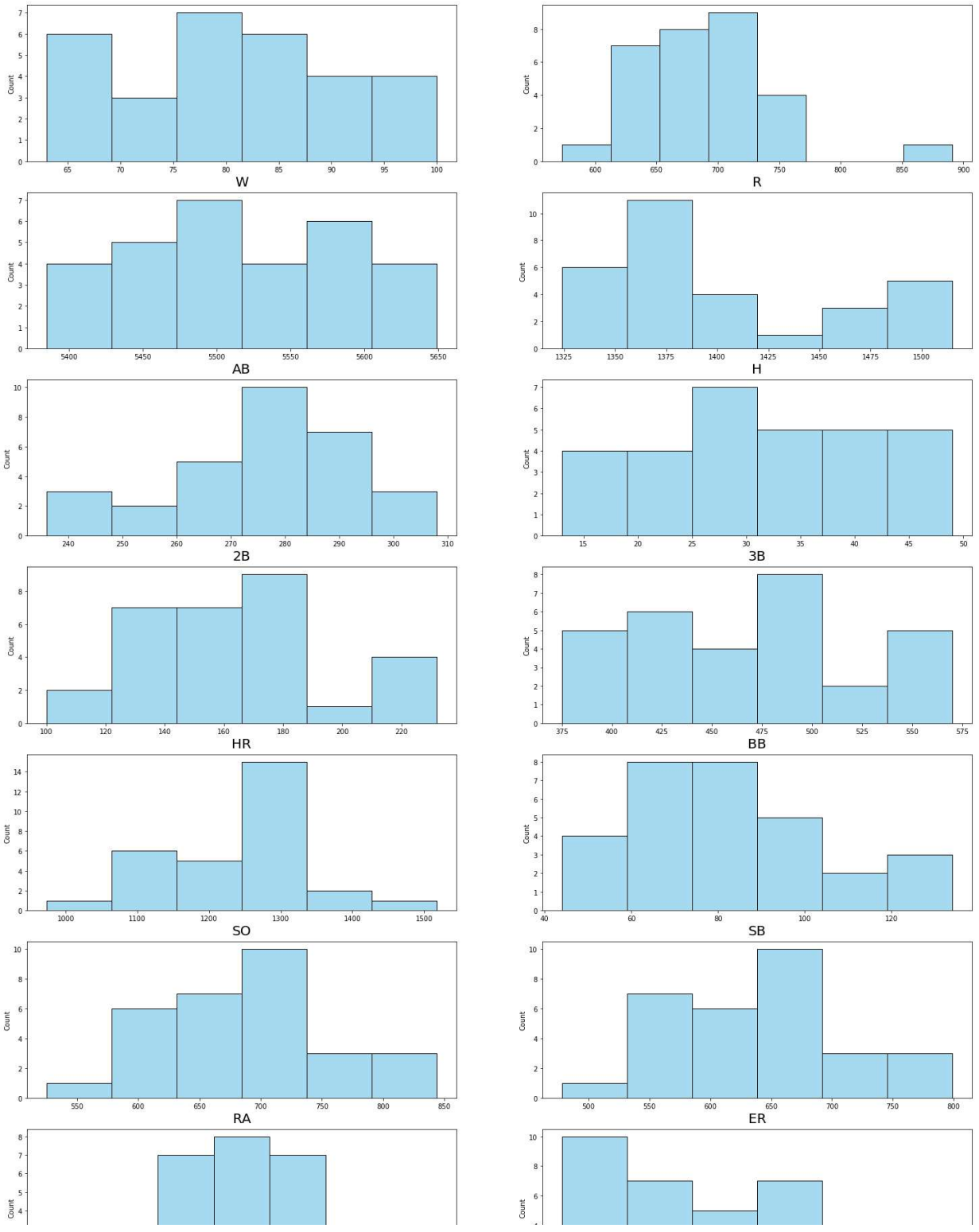

0.04
0.02
0.00
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
12.5
15.0
CG

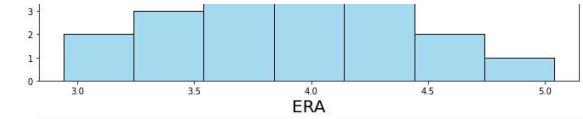

Density
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
5
10
15
20
25
SHO

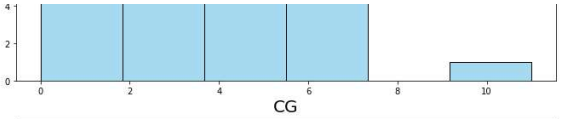

Density
0.05
0.04
0.03
0.02
0.01
0.00
20
30
40
50
60
70
SV

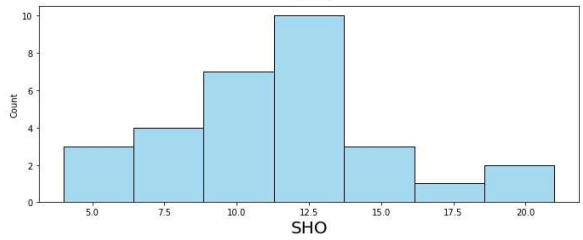

Density
0.05
0.04
0.03
0.02
0.01
0.00
60
80
100
120
140
E

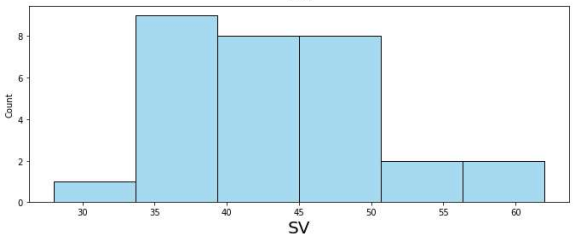
In [18]:
```
plt.figure(figsize = (25,50), facecolor = 'white')
plotnumber = 1

for column in df:
    if plotnumber <=17:
        ax = plt.subplot(10,2,plotnumber)
        sns.histplot(df[column], color = 'skyblue')
        plt.xlabel(column,fontsize=20)

    plotnumber += 1
plt.show()
```

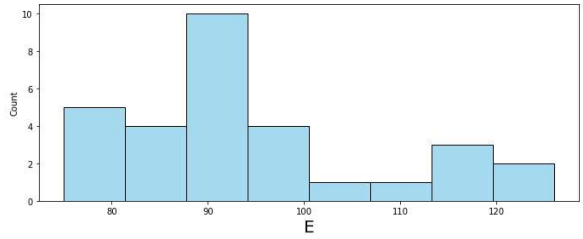



ERA

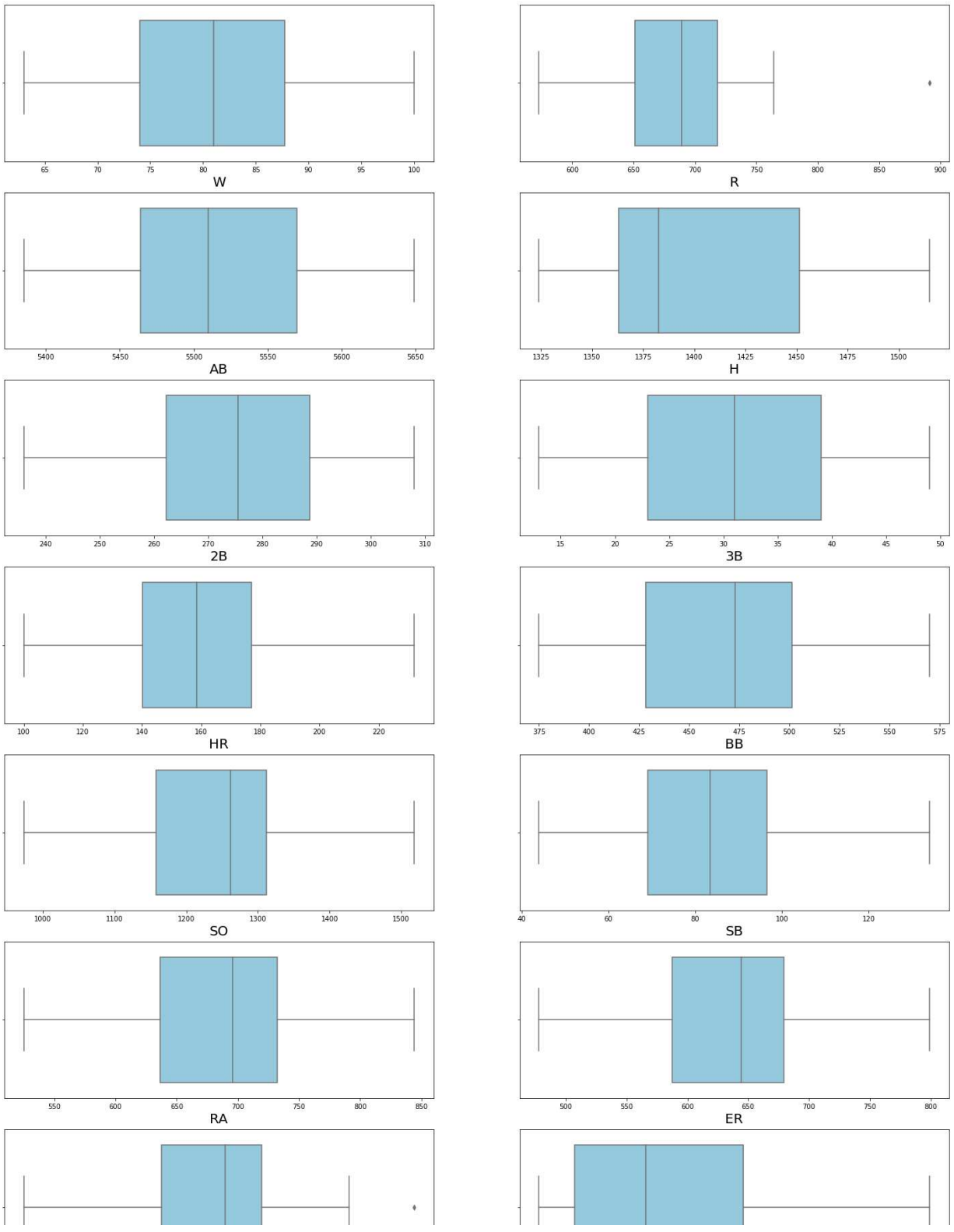

CG

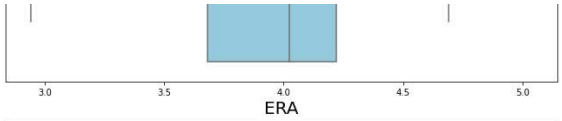

Count
SHO

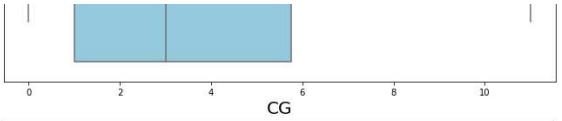

Count
SV


Count
E

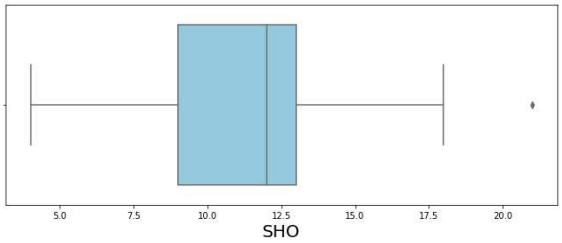
In [19]:
```
plt.figure(figsize = (25,50), facecolor = 'white')
plotnumber = 1
for column in df:
    if plotnumber <=17:
        ax = plt.subplot(10,2,plotnumber)
        sns.boxplot(df[column], color = 'skyblue')
        plt.xlabel(column,fontsize=20)

    plotnumber +=1
plt.show()
```



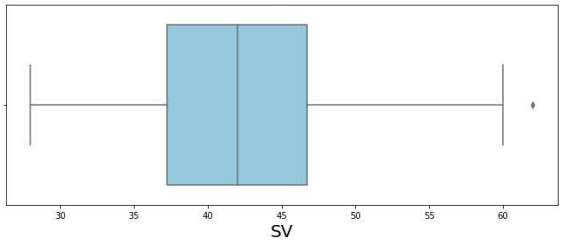




ERA

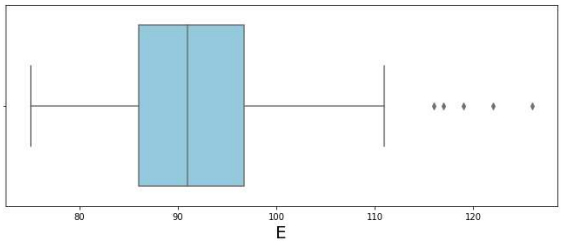


CG



SHO

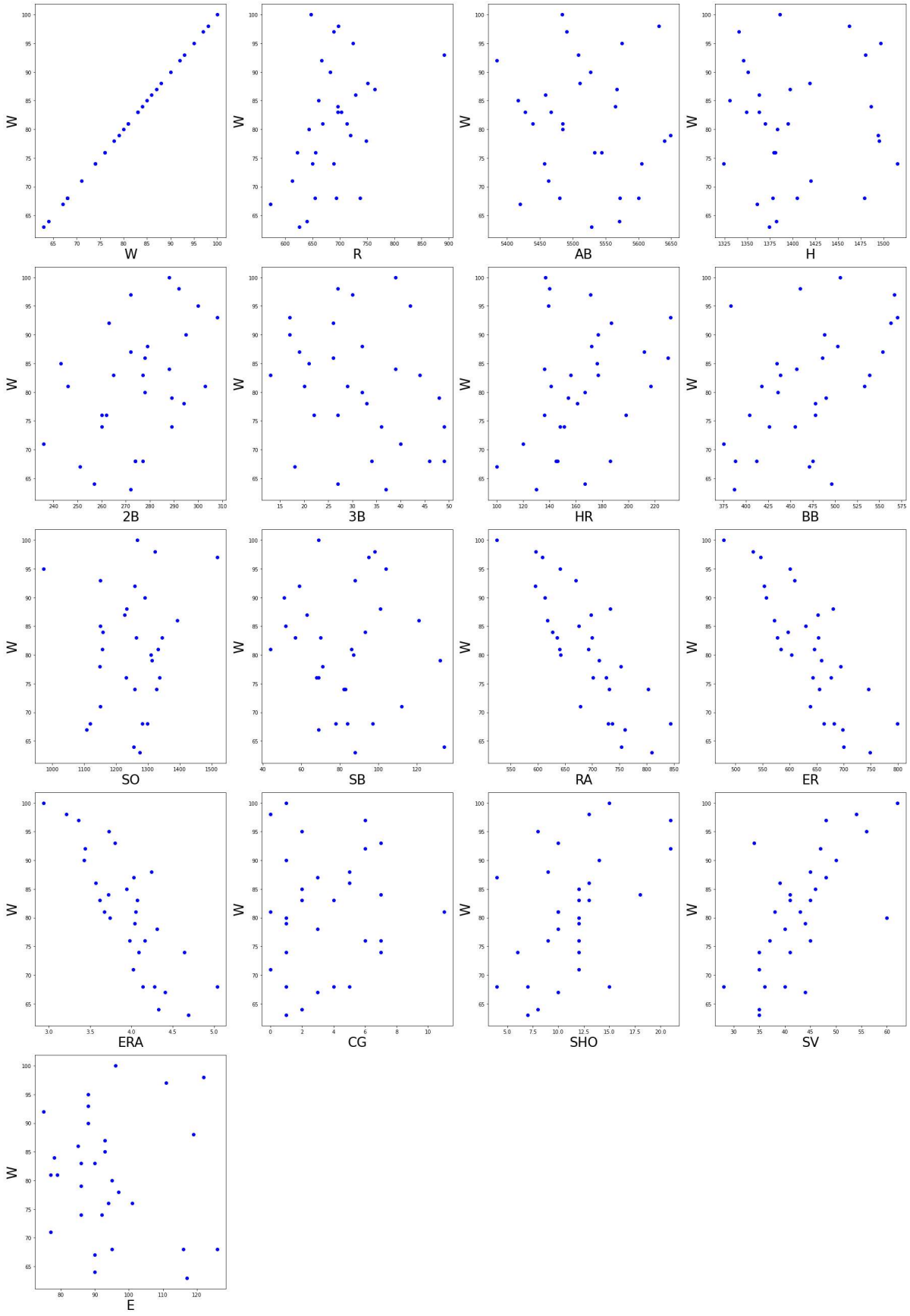
SV



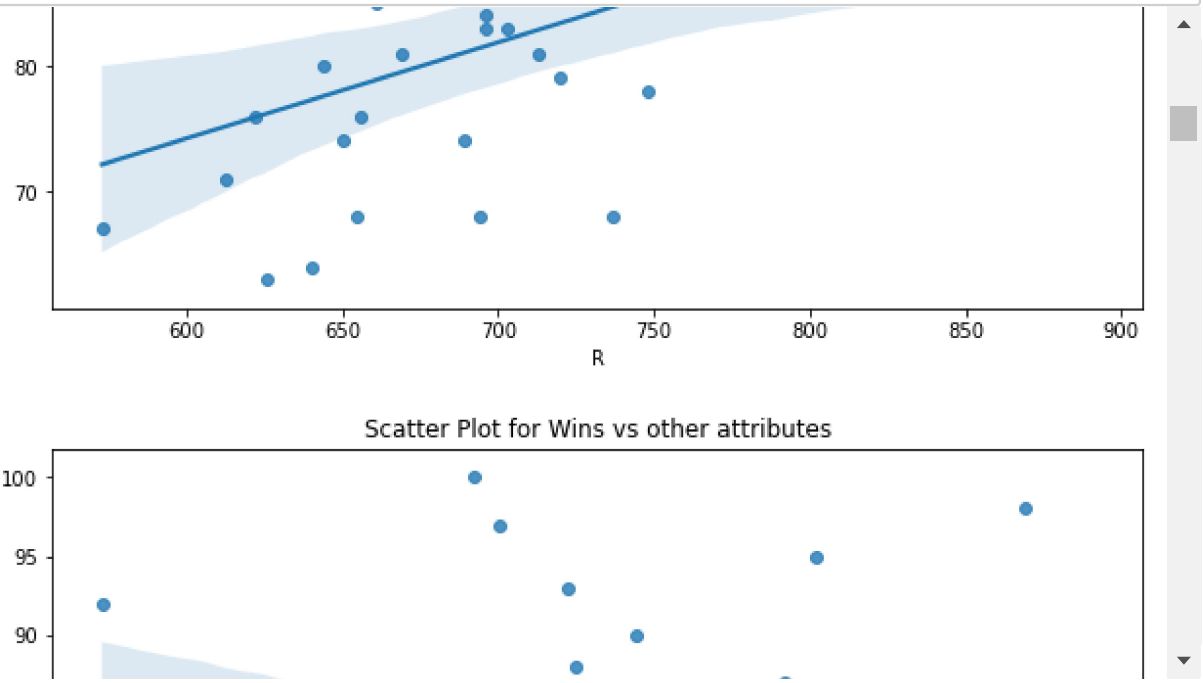
E

Bivariate Analysis

In [20]:
```
plt.figure(figsize = (25,50), facecolor = 'white')
plotnumber = 1

for column in df:
    if plotnumber <=20:
        ax = plt.subplot(7,4,plotnumber)
        plt.scatter(df[column],df['W'], color='b')
        plt.xlabel(column,fontsize=26)
        plt.ylabel('W', fontsize=26)
    plotnumber += 1
plt.tight_layout()
```

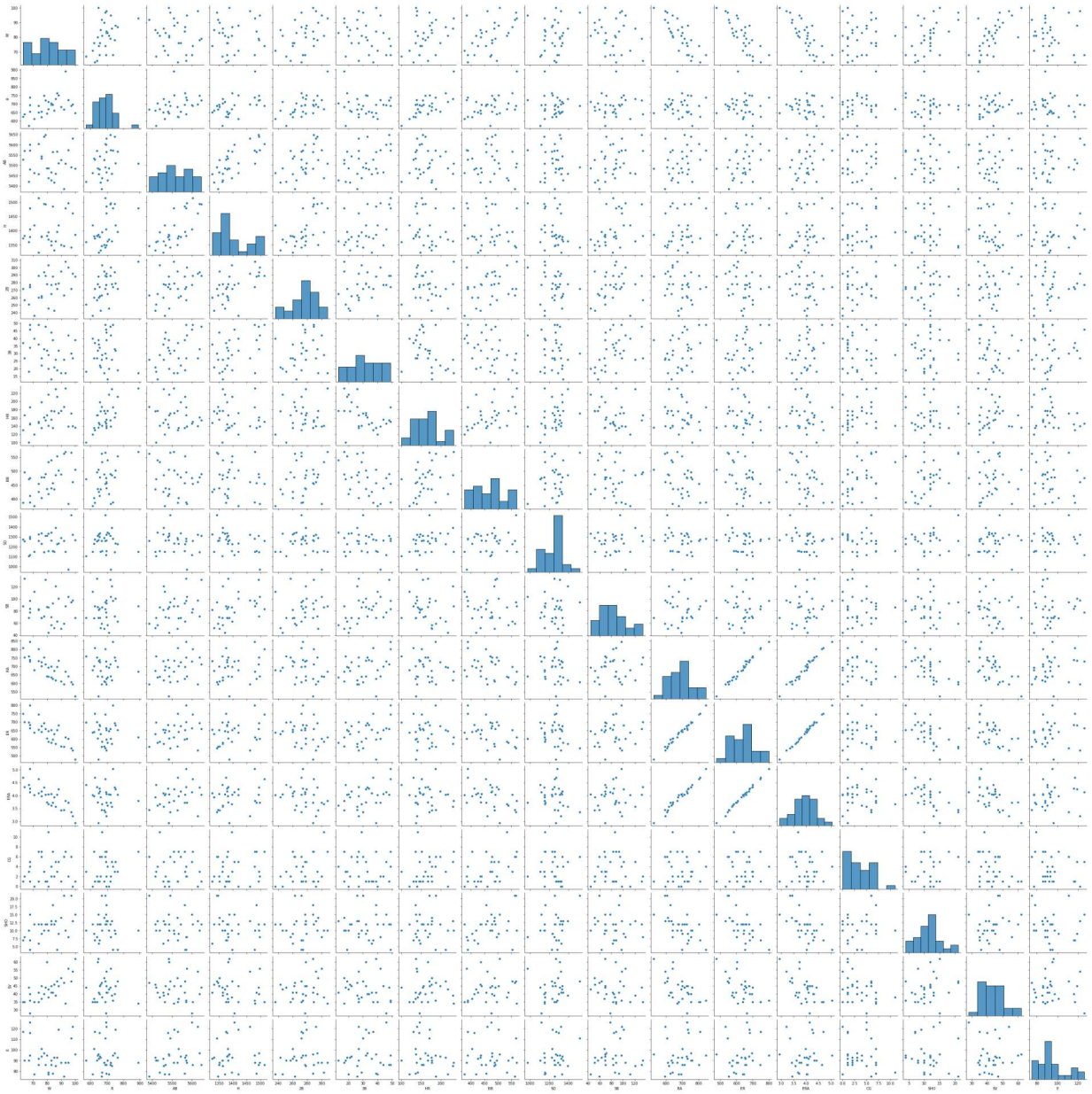

W
R
AB
H
2B
3B
HR
BB
SO
SB
RA
ER
ERA
CG
SHO
SV
E

In [21]:
```
plt.figure(figsize = (25,20), facecolor = 'white')
plotnumber = 1

for column in df:
    if plotnumber <=20:
        plt.figure(figsize = (10,6))
        sns.regplot(df[column], df['W'])
        plt.title('Scatter Plot for Wins vs other attributes')
        plt.xlabel(column)
        plt.ylabel('W')

plt.tight_layout()
```



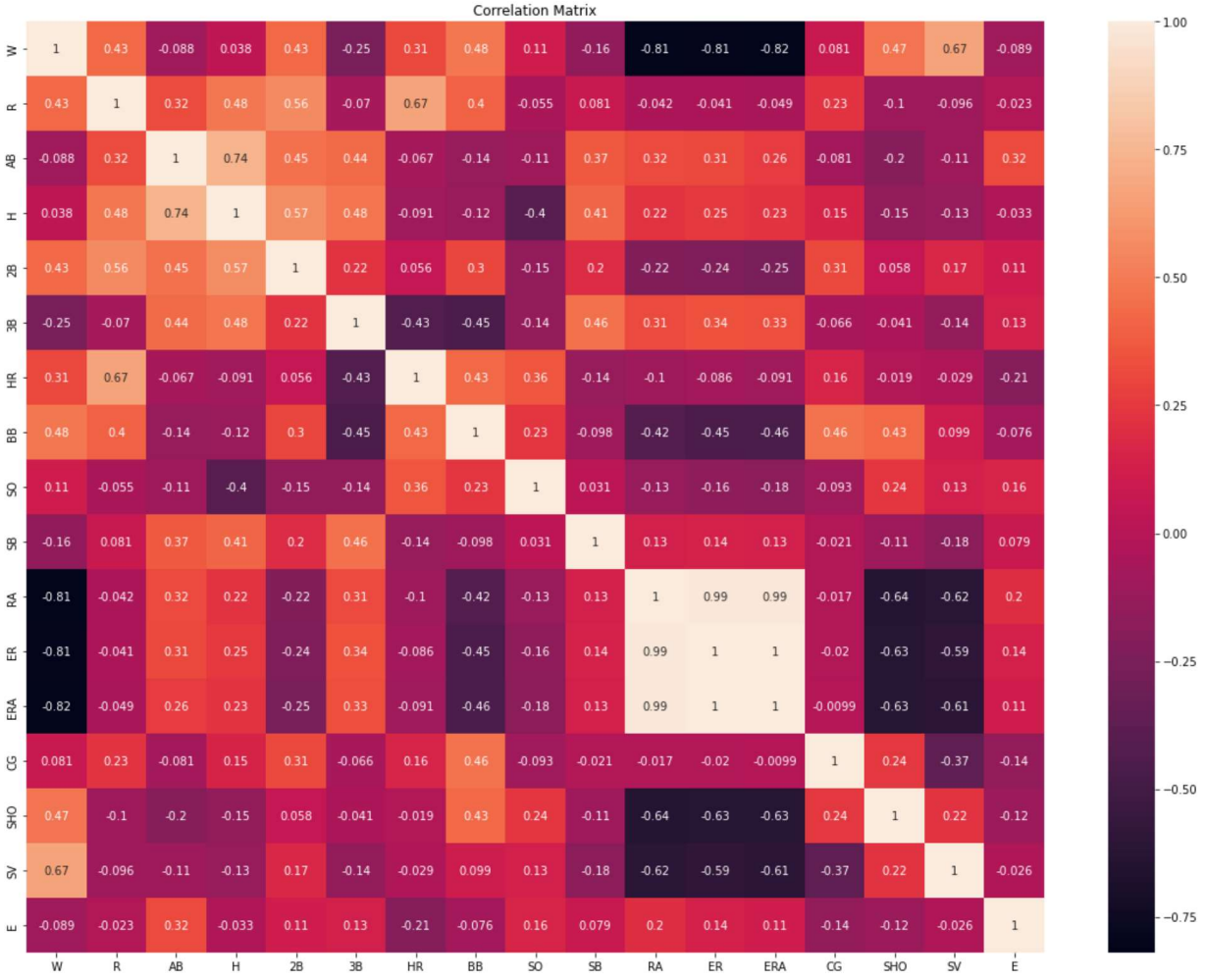
Multivariate Analysis

In [22]:
```
sns.pairplot(df)
plt.show()
```

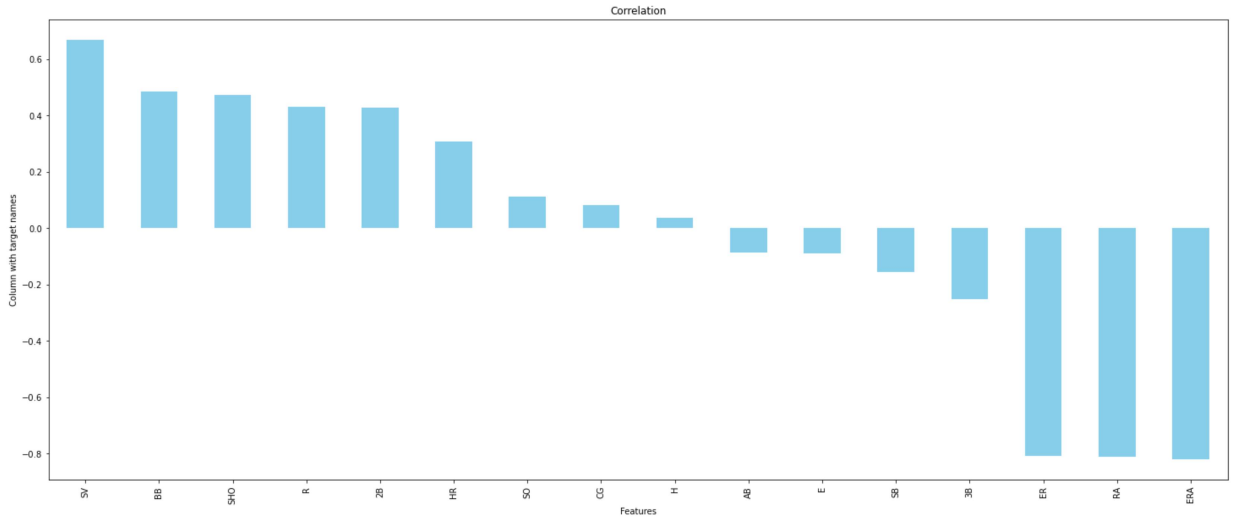


Heatmap

In [23]:
```
plt.figure(figsize = [20, 15])
sns.heatmap(df.corr(), annot = True)
plt.title('Correlation Matrix')
plt.show()
```



In [24]:
```
plt.figure(figsize =(25, 10))
df.corr()['W'].sort_values(ascending = False).drop(['W']).plot(kind ='bar', color
plt.xlabel('Features')
plt.ylabel('Column with target names')
plt.title('Correlation')
plt.show()
```



Data Cleaning

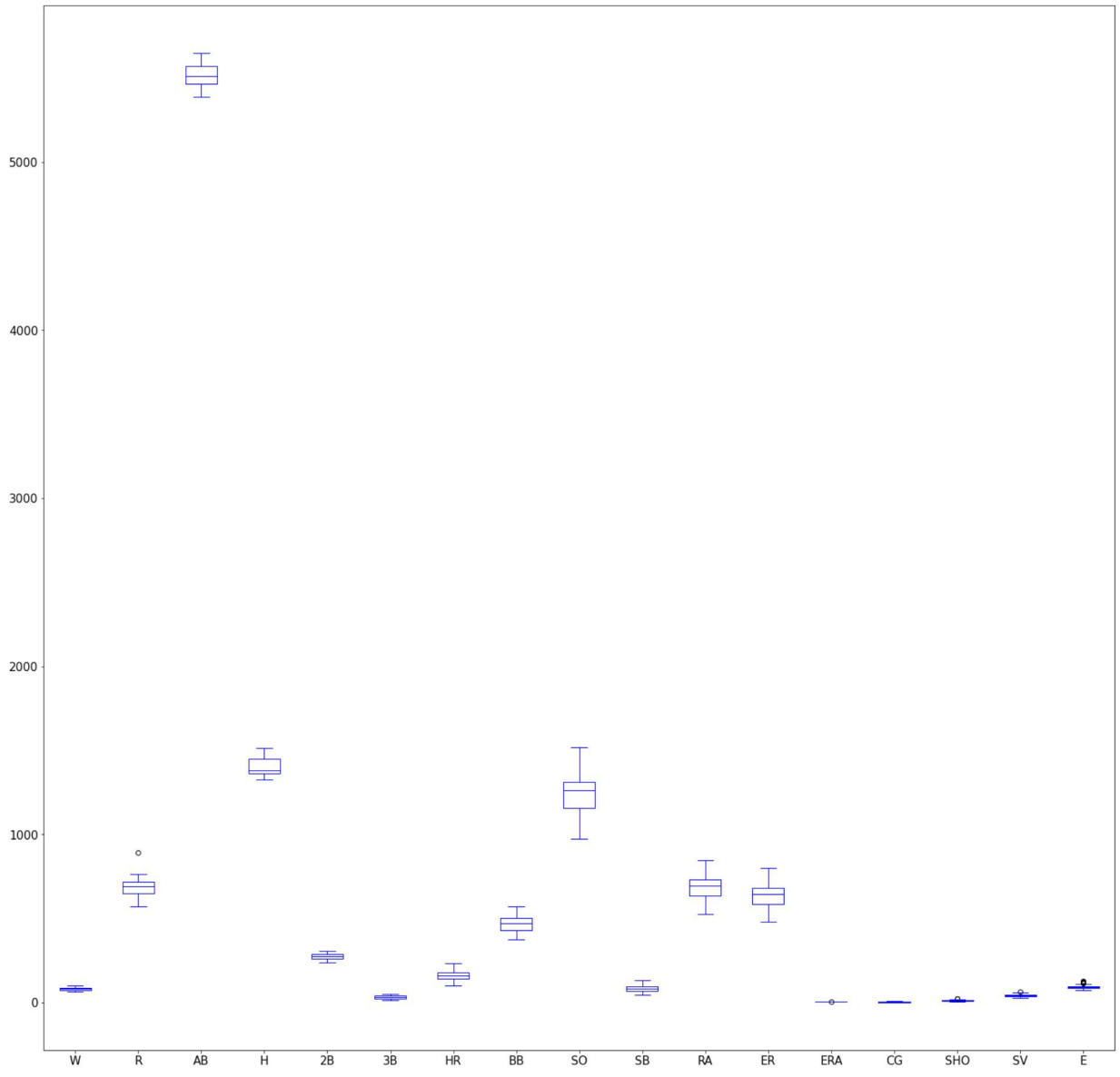
In [25]:
```
df.isnull().sum()
```

Out[25]:
```
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
```

Checking for Outliers

In [26]:
```
df.plot(kind='box',fontsize=15,figsize=(25,25),color='b')
```

Out[26]: <AxesSubplot:>



Removing Outliers

In [27]:
```
features=df[['R','ERA','SHO','SV','E']]
```

In [28]:
```
from scipy.stats import zscore
z=np.abs(zscore(features))
df_new=df[(z<3).all(axis=1)]
df_new
```

Out[28]:

	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
6	87	764	5567	1397	272	19	212	554	1227	63	698	652	4.03	3	4	48	93
7	81	713	5485	1370	246	20	217	418	1331	44	693	646	4.05	0	10	43	77
8	80	644	5485	1383	278	32	167	436	1310	87	642	604	3.74	1	12	60	95
9	78	748	5640	1495	294	33	161	478	1148	71	753	694	4.31	3	10	40	97
10	88	751	5511	1419	279	32	172	503	1233	101	733	680	4.24	5	9	45	119
11	86	729	5459	1363	278	26	230	486	1392	121	618	572	3.57	5	13	39	85
12	85	661	5417	1331	243	21	176	435	1150	52	675	630	3.94	2	12	46	93
13	76	656	5544	1379	262	22	198	478	1336	69	726	677	4.16	6	12	45	94
14	68	694	5600	1405	277	46	146	475	1119	78	729	664	4.14	5	15	28	126
15	100	647	5484	1386	288	39	137	506	1267	69	525	478	2.94	1	15	62	96
16	98	697	5631	1462	292	27	140	461	1322	98	596	532	3.21	0	13	54	122
17	97	689	5491	1341	272	30	171	567	1518	95	608	546	3.36	6	21	48	111
18	68	655	5480	1378	274	34	145	412	1299	84	737	682	4.28	1	7	40	116
19	64	640	5571	1382	257	27	167	496	1255	134	754	700	4.33	2	8	35	90
20	90	683	5527	1351	295	17	177	488	1290	51	613	557	3.43	1	14	50	88
21	83	703	5428	1363	265	13	177	539	1344	57	635	577	3.62	4	13	41	90
22	71	613	5463	1420	236	40	120	375	1150	112	678	638	4.02	0	12	35	77
23	67	573	5420	1361	251	18	100	471	1107	69	760	698	4.41	3	10	44	90
24	63	626	5529	1374	272	37	130	387	1274	88	809	749	4.69	1	7	35	117
25	92	667	5385	1346	263	26	187	563	1258	59	595	553	3.44	6	21	47	75
26	84	696	5565	1486	288	39	136	457	1159	93	627	597	3.72	7	18	41	78
27	79	720	5649	1494	289	48	154	490	1312	132	713	659	4.04	1	12	44	86
28	74	650	5457	1324	260	36	148	426	1327	82	731	655	4.09	1	6	41	92
29	68	737	5572	1479	274	49	186	388	1283	97	844	799	5.04	4	4	36	95

In [29]:
```
df_new.shape
```

Out[29]:
```
(29, 17)
```

In [30]:
```
df.shape
```

Out[30]:
```
(30, 17)
```

In [31]:
```
Dataloss = (((30-29)/30)*100)
```

In [32]:
```
Dataloss
```

Out[32]:
```
3.3333333333333335
```

Checking for Skewness

In [33]:
```
df_new.skew()
```

Out[33]:
```
W      0.119013
R     -0.215364
AB     0.169573
H      0.783772
2B    -0.335304
3B     0.090124
HR     0.450862
BB     0.151193
SO    -0.233815
SB     0.494966
RA     0.018155
ER     0.018461
ERA    0.016693
CG     0.854980
SHO    0.526943
SV     0.627480
E      0.840271
dtype: float64
```

Removing Skewness

In [34]:
```
skew_features=["H","CG","SHO","SV","E"]
```

In [35]:
```
df_new["H"]=np.log1p(df_new["H"])
```

In [36]:
```
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer(method='yeo-johnson')
'''
parameters:
method = 'box_cox' or 'yeo-johnson'
'''
```

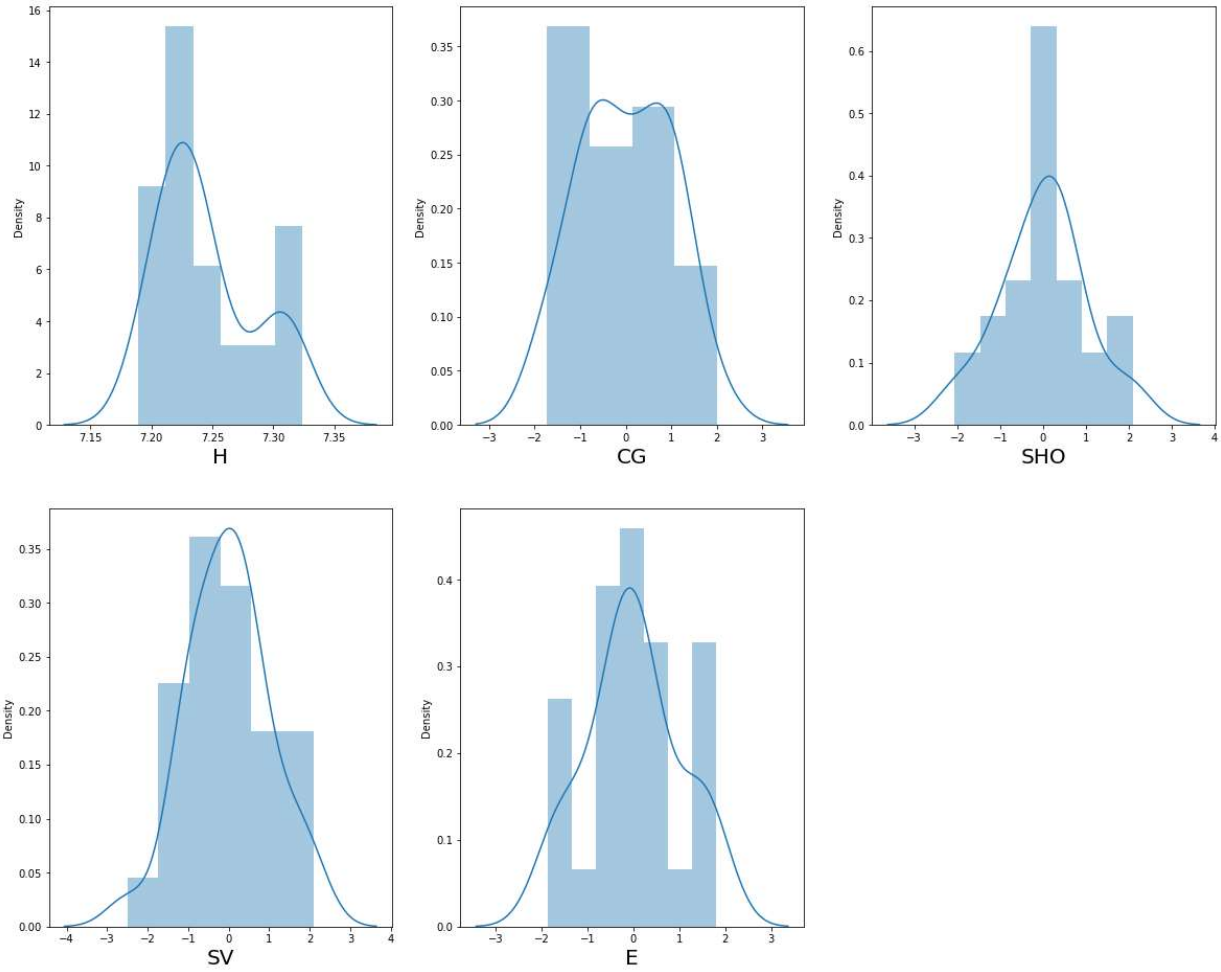
Out[36]:
```
"\nparameters:\nmethod = 'box_cox' or 'yeo-johnson'\n"
```

In [37]:
```
df_new[['E','SHO','SV','CG']] = scaler.fit_transform(df_new[['E','SHO','SV','CG']
```

In [38]:
```
df_new.skew()
```

Out[38]:
```
W      0.119013
R     -0.215364
AB     0.169573
H      0.738455
2B    -0.335304
3B     0.090124
HR     0.450862
BB     0.151193
SO    -0.233815
SB     0.494966
RA     0.018155
ER     0.018461
ERA    0.016693
CG    -0.045947
SHO    0.000529
SV    -0.000925
E      0.065585
dtype: float64
```

In [39]:
```
plt.figure(figsize=(20,25), facecolor='white')
plotnumber = 1

for column in df_new[skew_features]:
    if plotnumber<=9:
        ax = plt.subplot(3,3,plotnumber)
        sns.distplot(df_new[column])
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.show()
```



Splitting the columns

In [40]:
```
x = df_new.drop("W",axis=1)
y = df_new["W"]
```

Scaling Data using StandardScaler

In [41]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
```

In [42]:
```
ssc=StandardScaler()
X = pd.DataFrame(ssc.fit_transform(x), columns=x.columns)
X.head()
```

Out[42]:

	R	AB	H	2B	3B	HR	BB	SO	SB
0	0.959398	0.830084	1.723726	1.556538	1.010845	-0.765863	-1.536359	-2.727623	0.905953
1	0.331147	-0.702831	-0.949846	0.201171	1.208917	-0.181389	-0.495021	0.121896	-0.585315
2	-0.274666	-1.100253	-0.088700	1.733325	-0.276617	-0.697101	1.252941	-0.925866	0.116458
3	-1.329231	0.233951	-0.347747	-0.800621	-0.474688	-0.869006	-1.145857	-0.201246	-0.673037
4	0.174084	1.255894	2.030708	0.908319	1.704094	-0.353293	-0.197495	0.072935	-0.015124

Variance inflation factor

In [43]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(X.values, i) for i in range(X.shap
vif["Features"]=X.columns
vif
```

Out[43]:

	vif_Features	Features
0	7.112342	R
1	20.237195	AB
2	10.114607	H
3	3.528059	2B
4	3.227808	3B
5	8.736960	HR
6	3.324550	BB
7	2.811546	SO
8	1.955254	SB
9	200.826177	RA
10	2079.420586	ER
11	1614.529449	ERA
12	2.912414	CG
13	3.274913	SHO
14	5.914692	SV
15	2.341746	E

In [44]:
```
X = X.drop(["ER"],axis=1)
```

In [45]:
```
X.head(10)
```

Out[45]:

	R	AB	H	2B	3B	HR	BB	SO	SB
0	0.959398	0.830084	1.723726	1.556538	1.010845	-0.765863	-1.536359	-2.727623	0.905953
1	0.331147	-0.702831	-0.949846	0.201171	1.208917	-0.181389	-0.495021	0.121896	-0.585315
2	-0.274666	-1.100253	-0.088700	1.733325	-0.276617	-0.697101	1.252941	-0.925866	0.116458
3	-1.329231	0.233951	-0.347747	-0.800621	-0.474688	-0.869006	-1.145857	-0.201246	-0.673037
4	0.174084	1.255894	2.030708	0.908319	1.704094	-0.353293	-0.197495	0.072935	-0.015124
5	1.856900	0.716535	-0.051906	-0.093474	-1.266972	1.743939	1.643442	-0.240414	-0.892341
6	0.712585	-0.447345	-0.553131	-1.625627	-1.167936	1.915843	-0.885522	0.777970	-1.725697
7	-0.835605	-0.447345	-0.310580	0.260100	0.020490	0.196801	-0.550807	0.572335	0.160319
8	1.497899	1.752672	1.689389	1.202964	0.119526	-0.009484	0.230197	-1.013995	-0.541454
9	1.565212	-0.078310	0.349394	0.319029	0.020490	0.368705	0.695081	-0.181661	0.774370

In [46]:
```
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(X.values, i) for i in range(X.shap
vif["Features"]=X.columns
vif
```

Out[46]:

	vif_Features	Features
0	6.069116	R
1	7.791456	AB
2	9.340727	H
3	3.227862	2B
4	3.125190	3B
5	7.474108	HR
6	3.323583	BB
7	2.786186	SO
8	1.787493	SB
9	142.535647	RA
10	141.460079	ERA
11	2.536866	CG
12	3.136960	SHO
13	2.630293	SV
14	2.272337	E

In [47]:
```
X = X.drop(["ERA"],axis=1)
```

In [48]:
```
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(X.values, i) for i in range(X.shap
vif["Features"]=X.columns
vif
```

Out[48]:

	vif_Features	Features
0	6.019438	R
1	5.210230	AB
2	7.728111	H
3	3.165732	2B
4	2.981242	3B
5	6.168639	HR
6	3.053363	BB
7	2.287734	SO
8	1.773119	SB
9	4.759240	RA
10	2.522962	CG
11	3.121855	SHO
12	2.606434	SV
13	2.239918	E

Splitting the data for training and testing

In [49]:
```
from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
```

Best random state

In [50]:
```
from sklearn.linear_model import  LinearRegression

LR=LinearRegression()

for i in range(0,100):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.22,random_stat
    LR.fit(x_train,y_train)
    LR_predict_train=LR.predict(x_train)
    LR_predict_test=LR.predict(x_test)
    print(f'At random state{i}, The training accuracy is :-{r2_score(y_train,LR_p
    print(f'At random state{i}, The test accuracy is :-{r2_score(y_test,LR_predic
    print('\n')
```

```
At random state63, The training accuracy is :-0.9820000911158874
At random state63, The test accuracy is :-0.5977506629877276


At random state64, The training accuracy is :-0.9570117243236277
At random state64, The test accuracy is :-0.7484439773830265


At random state65, The training accuracy is :-0.9785915941789562
At random state65, The test accuracy is :-0.38882625237926727


At random state66, The training accuracy is :-0.9556742188288497
At random state66, The test accuracy is :-0.6270406623518233


At random state67, The training accuracy is :-0.9598461778868997
```

In [51]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.22,random_state=99
x_train.shape
```

Out[51]: (22, 16)

In [52]:
```
y_train.shape
```

Out[52]: (22,)

In [53]:
```
x_test.shape
```

Out[53]: (7, 16)

In [54]:
```
y_test.shape
```

Out[54]: (7,)

Regression Algorithm

In [55]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor as KNN
from sklearn.linear_model import SGDRegressor
from sklearn.metrics import classification_report
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import BaggingRegressor
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Lasso,Ridge
```

Linear Regression

In [56]:
```
from sklearn.linear_model import LinearRegression

LR=LinearRegression()
LR.fit(x_train,y_train)
print(LR.score(x_train,y_train))
LR_predict=LR.predict(x_test)
```

```
0.9633011594527373
```

In [57]:
```
print('MSE:',mean_squared_error(LR_predict,y_test))
print('MAE:',mean_absolute_error(LR_predict,y_test))
print('r2_score:',r2_score(LR_predict,y_test))
```

```
MSE: 27.077878720280413
MAE: 4.999072397568483
r2_score: 0.8702723798743279
```

SGD

In [58]:
```
from sklearn.linear_model import SGDRegressor
sgd=SGDRegressor()
sgd.fit(x_train,y_train)
pred=sgd.predict(x_test)
print('R2_score:',r2_score(y_test,pred))
print('mae:',metrics.mean_squared_error(y_test,pred))
print('mae:',metrics.mean_absolute_error(y_test,pred))
print('rmse:',np.sqrt(metrics.mean_squared_error(y_test,pred)))
```

```
R2_score: -2.688080491262262e+29
mae: 3.6470120624309225e+31
mae: 6021693940014786.0
rmse: 6039049645789412.0
```

Lasso Regressor

In [59]:
```
parameters={'alpha':[0.0001,0.001,0.01,0.1,1,10],'random_state':list(range(0,100)
ls=Lasso()
Z=GridSearchCV(ls,parameters)
Z.fit(x_train,y_train)
print(Z.best_params_)
```

```
{'alpha': 10, 'random_state': 0}
```

In [60]:
```
ls=Lasso(alpha=1,random_state=0)
ls.fit(x_train,y_train)
pred=ls.predict(x_test)
print('R2_score:',r2_score(y_test,pred))
print('mse:',metrics.mean_squared_error(y_test,pred))
print('mae:',metrics.mean_absolute_error(y_test,pred))
print('rmse:',np.sqrt(metrics.mean_squared_error(y_test,pred)))
```

```
R2_score: 0.8162585359689938
mse: 24.92884189547204
mae: 4.824282278813722
rmse: 4.992879118852372
```

Ridge Regressor

In [61]:
```
parameters={'alpha':[0.001,0.01,0.1,1,10],'solver':['auto','svd','cholesky','lsqr
rd=Ridge()
Z1=GridSearchCV(rd,parameters)
Z1.fit(x_train,y_train)
print(Z1.best_params_)
```

```
{'alpha': 10, 'solver': 'saga'}
```

In [62]:
```
rd=Ridge(alpha=10,solver='lsqr')
rd.fit(x_train,y_train)
pred=rd.predict(x_test)
print('R2_score:',r2_score(y_test,pred))
print('mse:',metrics.mean_squared_error(y_test,pred))
print('mae:',metrics.mean_absolute_error(y_test,pred))
print('rmse:',np.sqrt(metrics.mean_squared_error(y_test,pred)))
```

```
R2_score: 0.8412327486974005
mse: 21.54050380938125
mae: 4.382292280473268
rmse: 4.6411748307278025
```

Cross Validation

In [63]:
```
from sklearn.model_selection import cross_val_score

np.random.seed(10)
def rmse_cv(model,x,y):
    rmse =-(cross_val_score(model,x,y, scoring='neg_mean_squared_error', cv=10))
    return(rmse)

models = [LinearRegression(),
          Ridge(),
          SVR(kernel='linear'),
          SVR(kernel='poly'),
          SVR(kernel='rbf'),
          RandomForestRegressor(),
          DecisionTreeRegressor(),
          GradientBoostingRegressor(),]

names = ['LR','R','svr','svr_p','svr_r','RF','DTR','GBR']

for model,name in zip(models,names):
    score = rmse_cv(model,x,y)
    print("{}   : {:.6f}, {:4f}".format(name,score.mean(),score.std()))
```

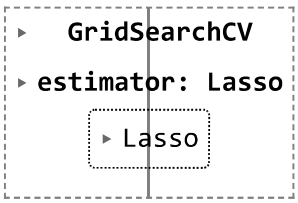
```
LR   : 33.227142, 22.206678
R   : 32.549049, 32.609656
svr   : 58.165093, 34.299128
svr_p   : 111.891643, 79.768934
svr_r   : 112.275451, 79.777668
RF   : 39.864027, 24.023091
DTR   : 70.216667, 45.490539
GBR   : 47.569624, 31.099844
```

Hyper Parameter Tuning

In [64]:
```
from sklearn.model_selection import GridSearchCV
```

In [65]:
```
parameter = {'alpha':[0.01,0.1,1.0,10.26],
             'selection':['cyclic','random'],
             'max_iter':[2,4,6,8,10],
             'tol':[100,1000],
             'fit_intercept':['bool']}
GVC=GridSearchCV(Lasso(),parameter,cv=5)
GVC.fit(x_train,y_train)
```

Out[65]:



In [66]:
```
GVC.best_params_
```

Out[66]:
```
{'alpha': 0.1,
 'fit_intercept': 'bool',
 'max_iter': 2,
 'selection': 'random',
 'tol': 1000}
```

In [67]:
```
Final_mod=Lasso(alpha=1.0, fit_intercept="bool", max_iter=4, selection="random",
Final_mod.fit(x_train,y_train)
pred=Final_mod.predict(x_test)
print('R2_Score:',r2_score(y_test,pred)*100)
print('mean_squared_error:',metrics.mean_squared_error(y_test,pred))
print('mean_absolute_error:',metrics.mean_absolute_error(y_test,pred))
print("RMSE value:",np.sqrt(metrics.mean_squared_error(y_test,pred)))
```

```
R2_Score: 79.48387716584048
mean_squared_error: 27.83493563295764
mean_absolute_error: 4.3354552220416105
RMSE value: 5.275882450638721
```

Saving the model

In [68]:
```
import joblib
joblib.dump(Final_mod,"BaseBallCaseStudy.pkl")
```

Out[68]:
```
['BaseBallCaseStudy.pkl']
```

Prediction

In [69]:
```
model=joblib.load("BaseballCaseStudy.pkl")


prediction = model.predict(x_test)
prediction
```

Out[69]:
```
array([69.58640733, 67.11525668, 85.45311838, 79.89227939, 85.81429738,
       88.93161129, 83.67532456])
```

In [70]:
```
pd.DataFrame([model.predict(x_test)[:],y_test[:]],index=["Predicted","Original"])
```

Out[70]:

	0	1	2	3	4	5	6
Predicted	69.586407	67.115257	85.453118	79.892279	85.814297	88.931611	83.675325
Original	67.000000	63.000000	97.000000	76.000000	83.000000	92.000000	86.000000

In [ ]:
```

```