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)

Importing all the required libraries:

```
In [1]: import pandas as pd
        import numpy as np
        import warnings
        warnings.filterwarnings('ignore')
        %matplotlib inline
```

Importing the data:

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

Out[2]:

	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

Data Analysis:

Checking if there are any null values present in the data

Checking for null values

```
In [3]: df.isnull().sum()

Out[3]: 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
```

Seems there are no null points present in the data

To check the shape that is to find the number of columns and rows in the data

```
In [4]: df.columns

Out[4]: Index(['W', 'R', 'AB', 'H', '2B', '3B', 'HR', 'BB', 'SO', 'SB', 'RA', 'ER',
               'ERA', 'CG', 'SHO', 'SV', 'E'],
              dtype='object')
```

To know the data in detailed we use describe funtion

```
In [5]: df.shape
Out[5]: (30, 17)
```

```
In [6]: df.describe()
Out[6]:
```

	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

To check how many unique values are present in each column

```
In [7]: df.nunique()
Out[7]: 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
```

To check the datatype of each column present in the data

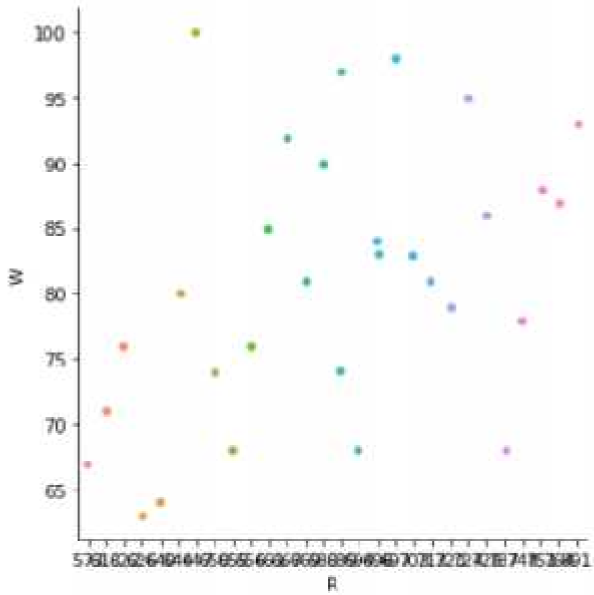
```
In [8]: df.dtypes

Out[8]: 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
```

Below are the catplot graphs where each column is compared with the WIN column

In [9]:
```
import seaborn as sns
```

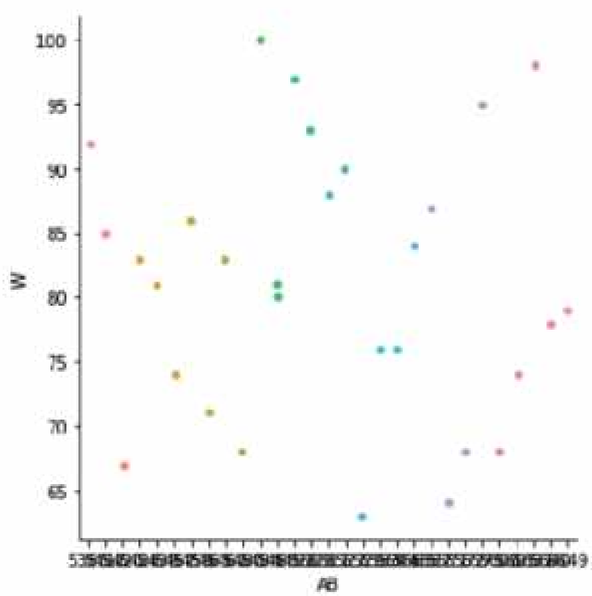
In [10]:
```
sns.catplot(x='R',y='W',data=df)
```

Out[10]: <seaborn.axisgrid.FacetGrid at 0x238ffd3a430>



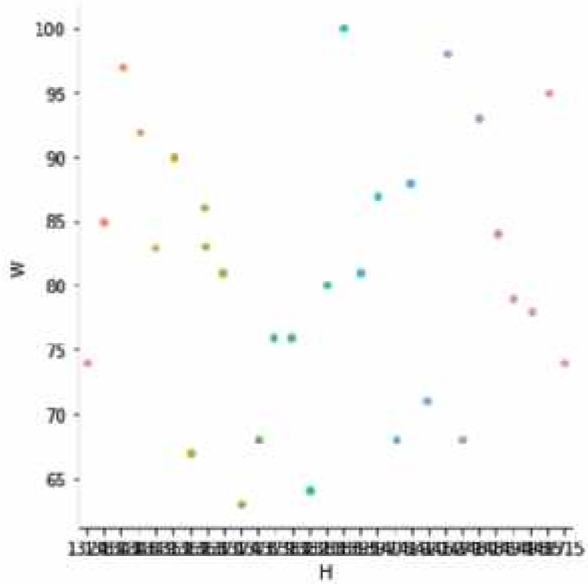
In [11]:
```
sns.catplot(x='AB',y='W',data=df)
```

Out[11]: <seaborn.axisgrid.FacetGrid at 0x238ffe79220>



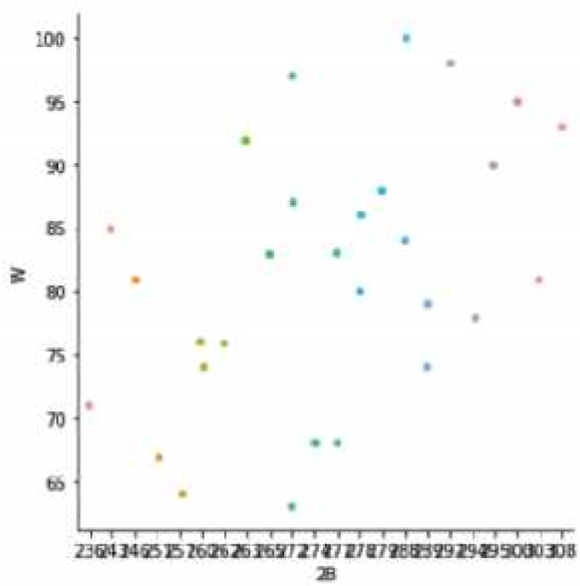
```
In [12]: sns.catplot(x='H',y='W',data=df)
```

```
Out[12]: <seaborn.axisgrid.FacetGrid at 0x238c0038be0>
```



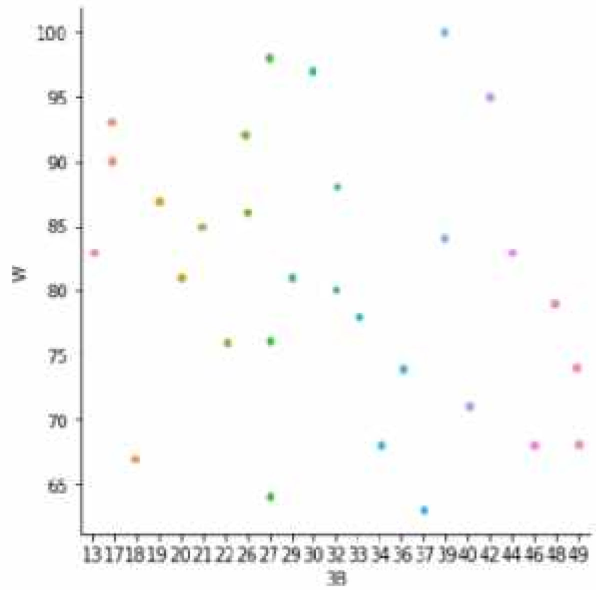

```
In [13]: sns.catplot(x='2B',y='W',data=df)
```

```
Out[13]: <seaborn.axisgrid.FacetGrid at 0x238ffe03730>
```



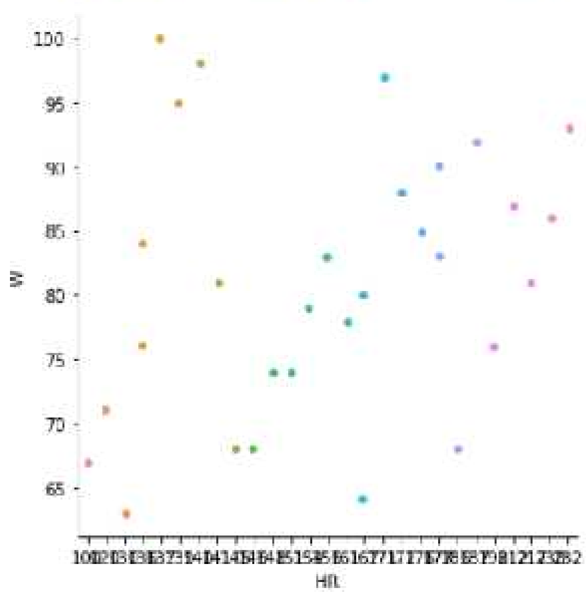
```
In [14]: sns.catplot(x='3B',y='W',data=df)
```

```
Out[14]: <seaborn.axisgrid.FacetGrid at 0x238c0132ca0>
```



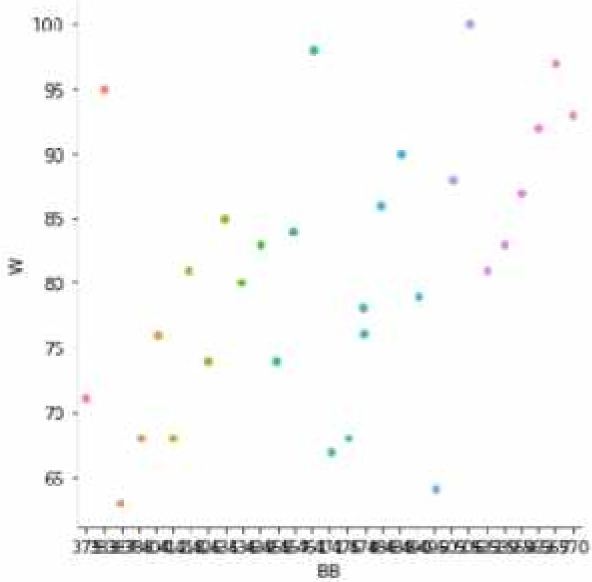

```
In [15]: sns.catplot(x='HR',y='W',data=df)
```

```
Out[15]: <seaborn.axisgrid.FacetGrid at 0x238c02ae730>
```



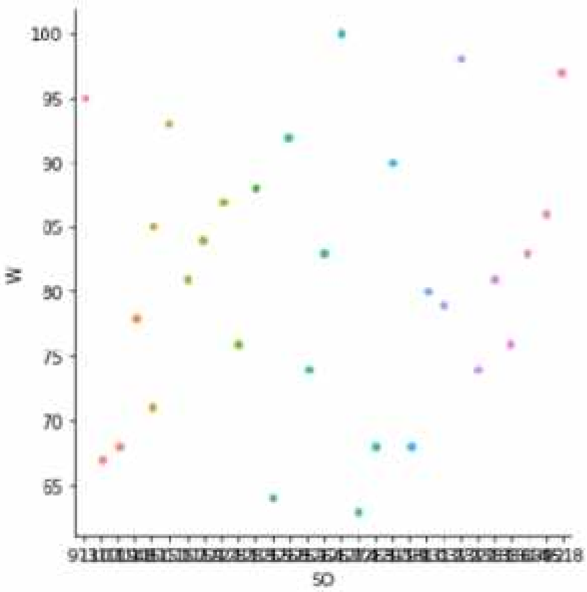
```
In [16]: sns.catplot(x='BB',y='W',data=df)
```

Out[16]: <seaborn.axisgrid.FacetGrid at 0x238c02f08e0>



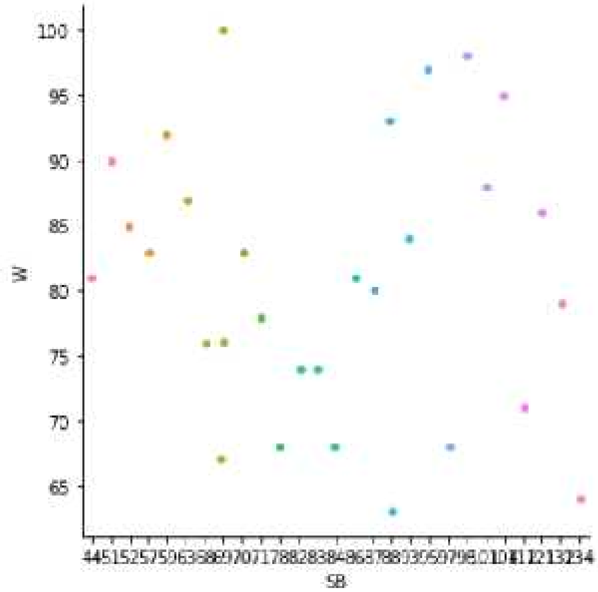

```
In [17]: sns.catplot(x='SO',y='W',data=df)
```

Out[17]: <seaborn.axisgrid.FacetGrid at 0x238c02a2a90>



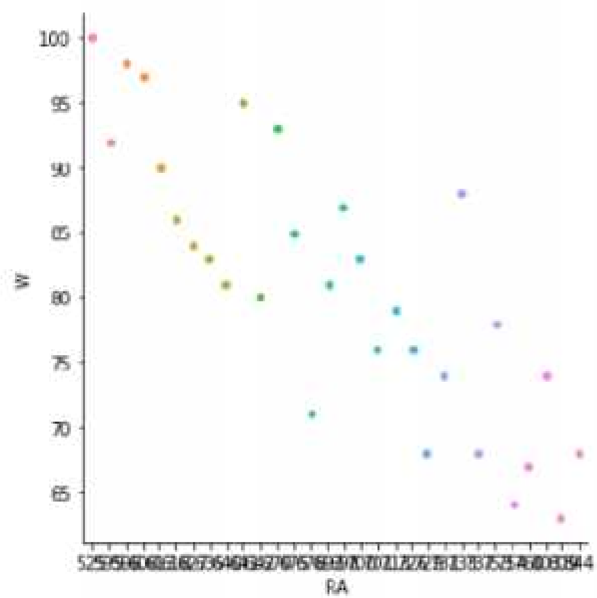
```
In [18]: sns.catplot(x='SB',y='W',data=df)
```

```
Out[18]: <seaborn.axisgrid.FacetGrid at 0x238c042d7f0>
```



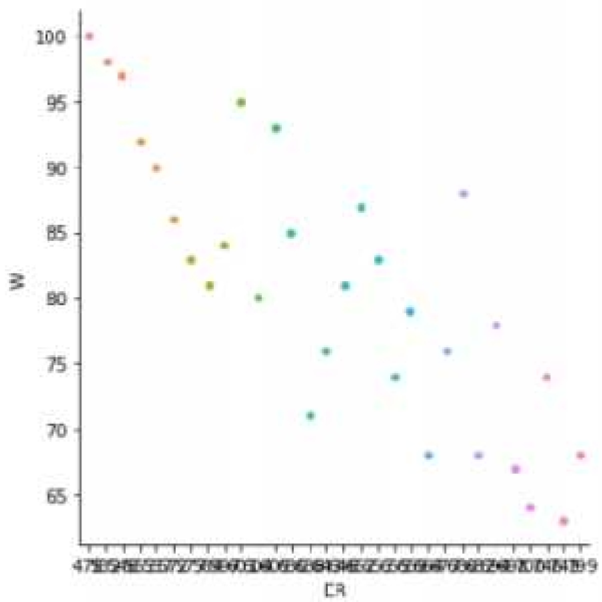
```
In [19]: sns.catplot(x='RA',y='W',data=df)
```

```
Out[19]: <seaborn.axisgrid.FacetGrid at 0x238c05498e0>
```



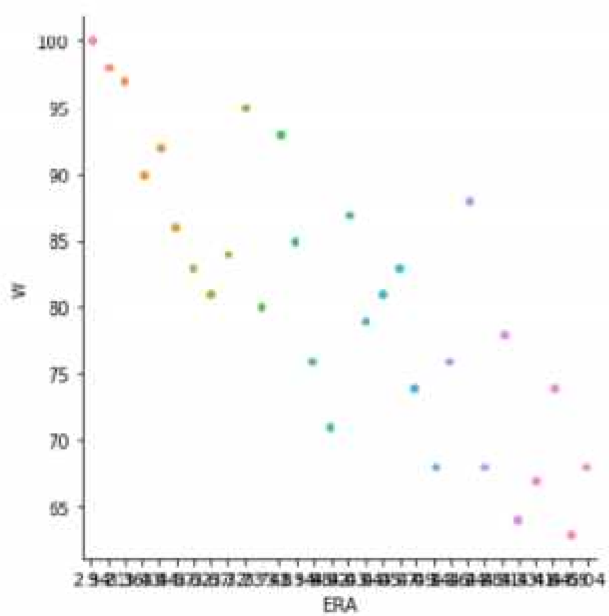
In [20]:
```
sns.catplot(x='ER',y='W',data=df)
```

Out[20]:
```
<seaborn.axisgrid.FacetGrid at 0x238c1680fa0>
```



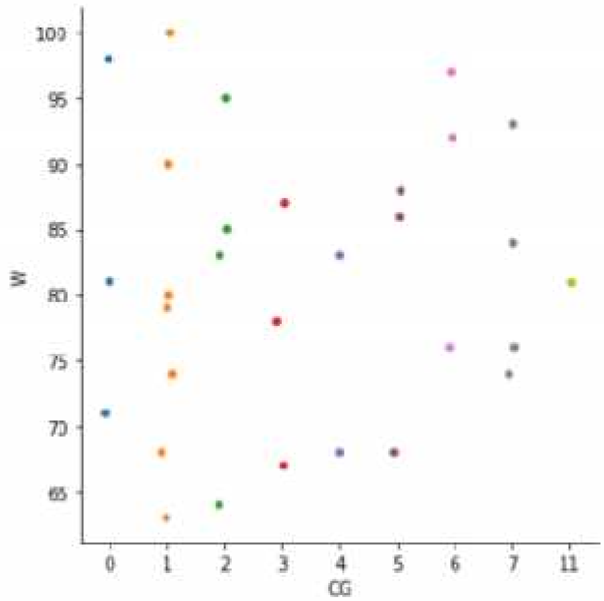
In [21]:
```
sns.catplot(x='ERA',y='W',data=df)
```

Out[21]:
```
<seaborn.axisgrid.FacetGrid at 0x238c175bdf0>
```



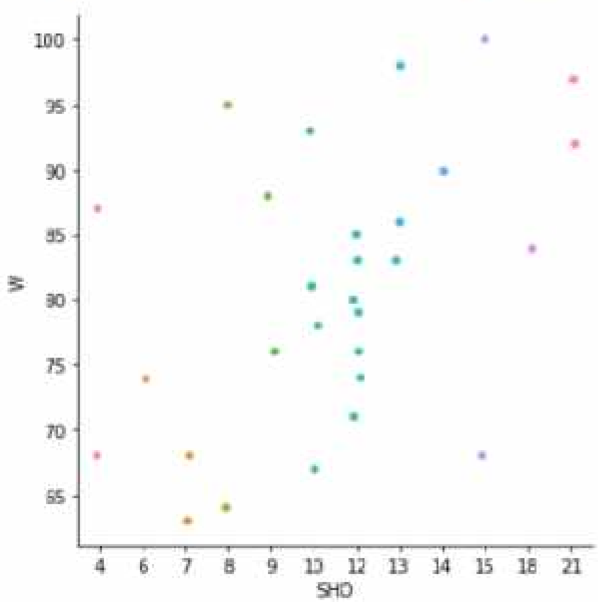
In [22]:
```python
sns.catplot(x='CG',y='W',data=df)
```

Out[22]: <seaborn.axisgrid.FacetGrid at 0x238c183d3d0>



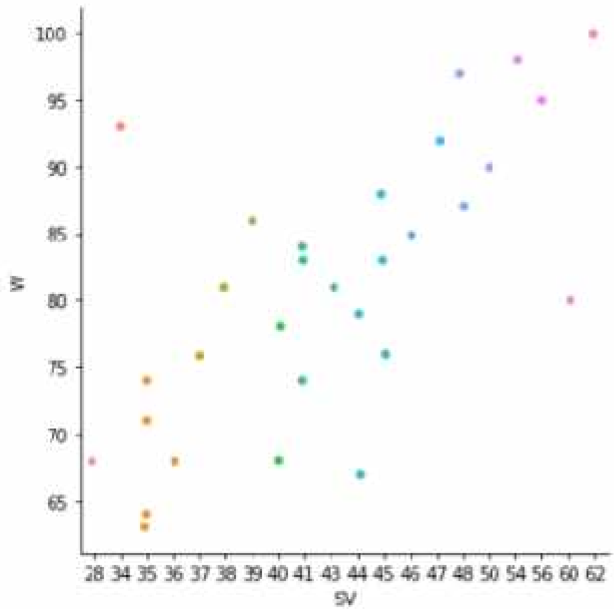
In [23]:
```python
sns.catplot(x='SHO',y='W',data=df)
```

Out[23]: <seaborn.axisgrid.FacetGrid at 0x238ffe8a250>



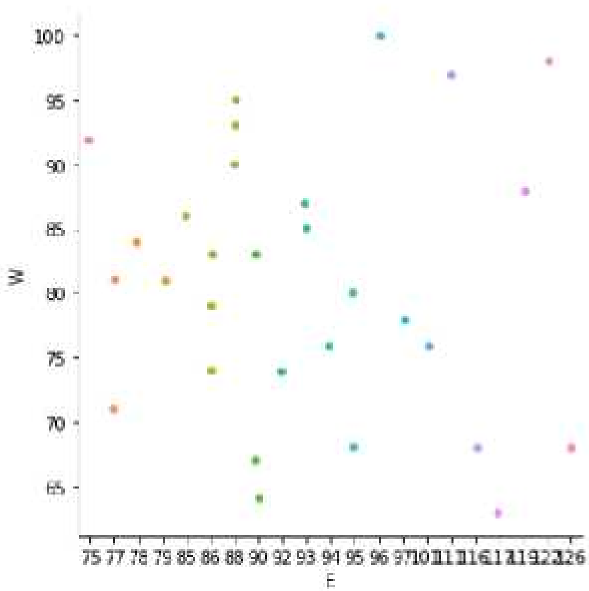
In [24]:
```
sns.catplot(x='SV',y='W',data=df)
```

Out[24]:
```
<seaborn.axisgrid.FacetGrid at 0x238c15e94c0>
```



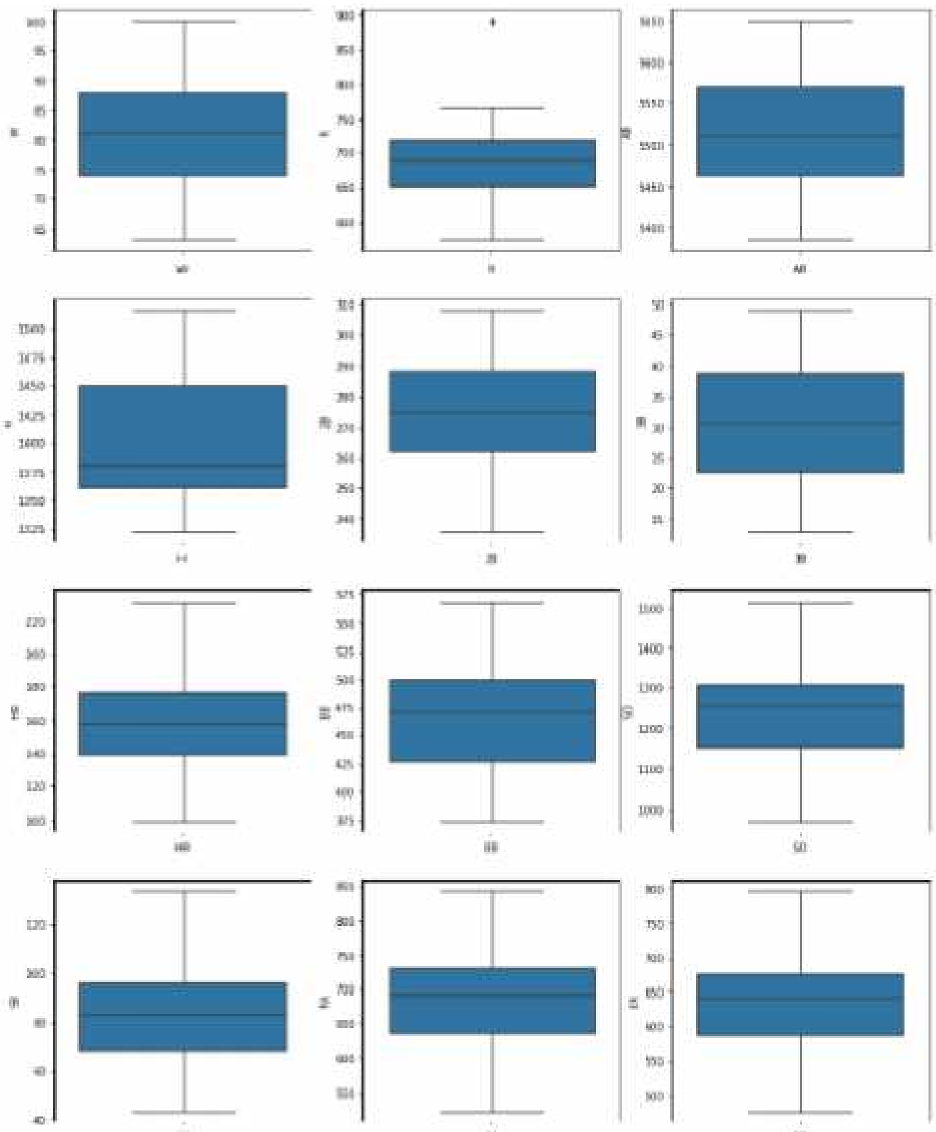
In [25]:
```
sns.catplot(x='E',y='W',data=df)
```

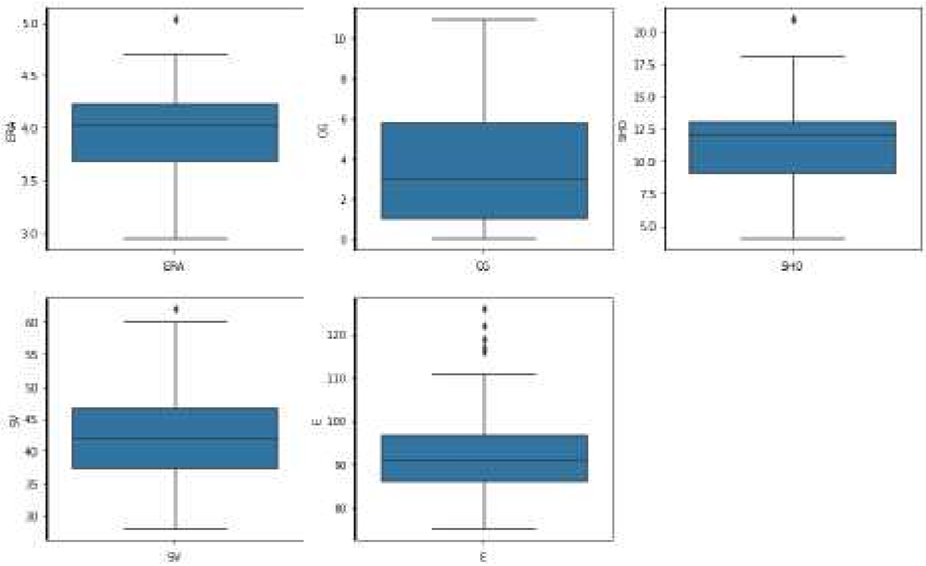
Out[25]:
```
<seaborn.axisgrid.FacetGrid at 0x238c19abc40>
```



Visualizing the outliers using the boxplot:

In [26]:
```
#Visualize the outliers using boxplot
import matplotlib.pyplot as plt
plt.figure(figsize=(15,50))
graph=1
for column in df:
    if graph<=30:
        ax=plt.subplot(10,3,graph)
        sns.boxplot(df[column],orient='v')
        plt.xlabel(column,fontsize=10)
    graph+=1
plt.show()
```





Removing the outliers by using z-score

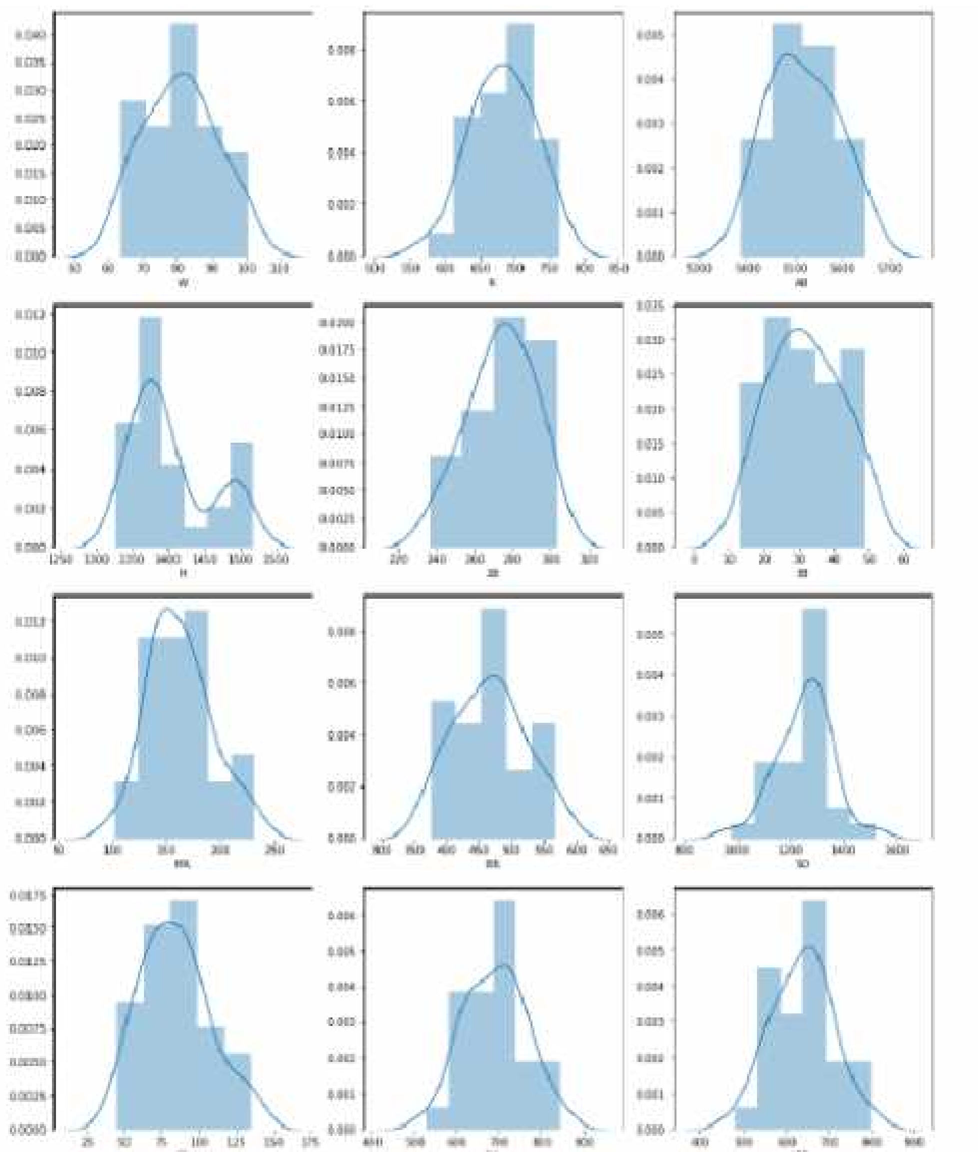
```
In [27]: #for removing outliers implementing zscore
         from scipy.stats import zscore
         z_score=zscore(df[['R','ERA','SHO','SV','E']])
         abs_z_score=np.abs(z_score)

         filtering_entry=(abs_z_score<3).all(axis=1)

         df=df[filtering_entry]
```

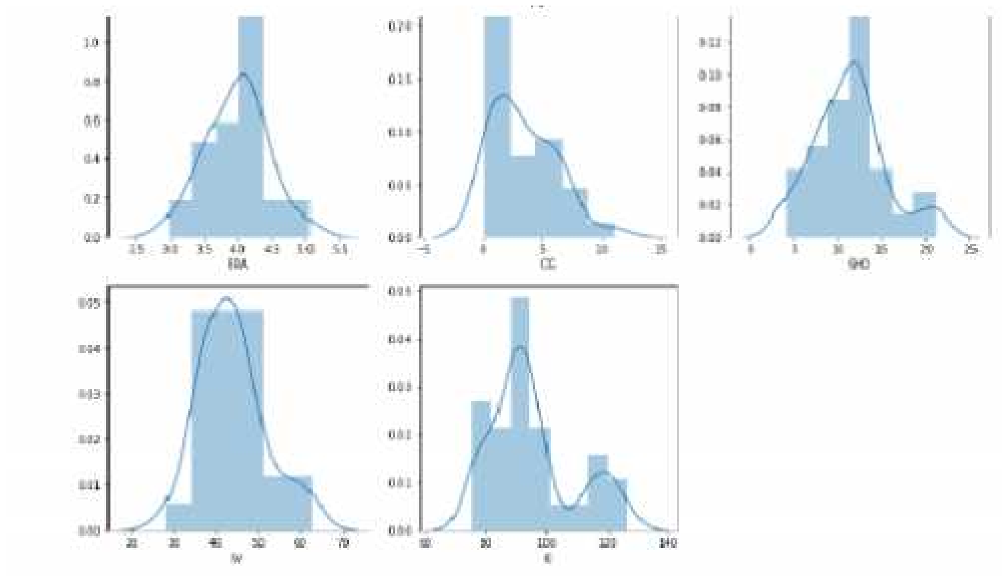
After removing the outliers the data ends up with 29 rows and 17 columns

```
In [31]: df.shape
Out[31]: (29, 17)
```

After removing the outliers, plotted the dist plot to check whether all columns have normal distribution

In [32]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(15,50))
graph=1
for column in df:
    if graph<=30:
        ax=plt.subplot(10,3,graph)
        sns.distplot(df[column])
        plt.xlabel(column,fontsize=10)
    graph+=1
plt.show()
```





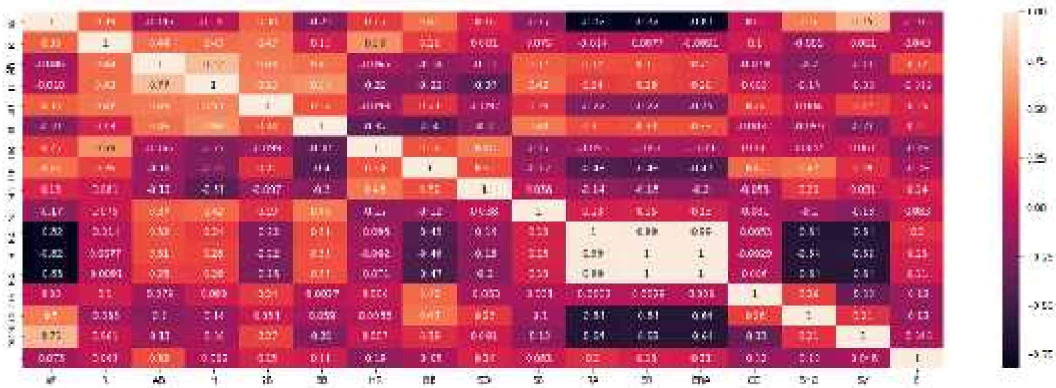
Now plotting the correlation graph to check the correlation of the data

```
In [33]: df.corr()
```

Out[33]:

	W	R	AB	H	2B	3B	HR	BB	
W	1.000000	0.390451	-0.085780	-0.018360	0.384886	-0.206737	0.245697	0.447513	0.1564
R	0.390451	1.000000	0.438022	0.433525	0.469293	0.134204	0.586894	0.258450	0.0811
AB	-0.085780	0.438022	1.000000	0.769159	0.490752	0.445604	-0.064653	-0.137850	-0.1112
H	-0.018360	0.433525	0.769159	1.000000	0.528016	0.582024	-0.218711	-0.222271	-0.3718
2B	0.384886	0.469293	0.490752	0.528016	1.000000	0.342419	-0.098695	0.211243	-0.0967
3B	-0.206737	0.134204	0.445604	0.582024	0.342419	1.000000	-0.369299	-0.404852	-0.1965
HR	0.245697	0.586894	-0.064653	-0.218711	-0.098695	-0.369299	1.000000	0.336814	0.4799
BB	0.447513	0.258450	-0.137850	-0.222271	0.211243	-0.404852	0.336814	1.000000	0.3155
SO	0.156469	0.081158	-0.111243	-0.371861	-0.096772	-0.196586	0.479914	0.315566	1.0000
SB	-0.169503	0.075323	0.373674	0.417877	0.194308	0.483818	-0.166072	-0.117622	0.0381
RA	-0.823176	-0.013858	0.315499	0.244606	-0.215196	0.312750	-0.092586	-0.425381	-0.1405
ER	-0.815308	0.007727	0.309146	0.260571	-0.224993	0.333731	-0.062094	-0.455832	-0.1776
ERA	-0.826952	-0.009122	0.254872	0.256458	-0.248212	0.325863	-0.070756	-0.465794	-0.1953
CG	0.029594	0.101438	-0.078511	0.092577	0.244856	-0.003733	0.065978	0.417437	-0.0530
SHO	0.497526	-0.065108	-0.198872	-0.135116	0.084060	-0.058896	0.005546	0.473922	0.2312
SV	0.749290	0.061381	-0.113342	-0.079814	0.269999	-0.210627	0.066984	0.187101	0.0913
E	-0.072858	0.043123	0.316297	-0.011945	0.145032	0.108610	-0.189790	-0.050114	0.1427

```
In [34]: #Now lets plot heatmap based on the correlation
         plt.figure(figsize=(22,7))
         sns.heatmap(df.corr(),annot=True)

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x238c1a5e760>
```



EDA Concluding Remarks:

- Seems there are no null values present in the data
- Every column has their own unique values
- By observing the cat plots we could see that there is quite good relation between every column to the Win column.
- There are outliers in 'R','ERA','SHO','SV','E.So performed the z score on only those columns
- And coming to the dist plot every column have the normal distributed data present in it
- Coming to the correlation graph, The number of runs, home runs, doubles, saves, shutouts and walks are highly positively linearly correlated.
- Stolen bases, runs allowed, earned runs are highly negative linearly correlated
- the remaining features have less to no linear correlation with no of wins

Preprocessing Pipe-lines:

Dividing the data into features and labels:

Here x is assigned to the input features of the model whereas y is assigned to the target.

```
In [35]: #Diving data set into features and labels
         y=df['W']
         x=df.drop(columns=['W'])
```

ANOVA:

Analysis of variance (ANOVA) is an analysis tool used in statistics that splits an observed aggregate variability found inside a data set into two parts: systematic factors and random factors

```
In [36]: from sklearn.feature_selection import SelectKBest
         from sklearn.feature_selection import f_classif
         s=SelectKBest(f_classif,k=15)
         s.fit(x,y)
         anova=pd.DataFrame([s.scores_,s.pvalues_],columns=x.columns).T.sort_values(by=0)
```

```
In [37]: #ANOVA is used to determine the influence that independent variables have on the
         anova
```

Out[37]:

	0	1
CG	0.361597	0.962964
H	0.729450	0.730204
2B	0.799063	0.680704
3B	0.811129	0.672290
HR	0.818974	0.666351
BB	0.943327	0.584607
SHO	1.253358	0.418187
SO	1.519309	0.316179
AB	1.622566	0.284961
ER	1.636442	0.281042
ERA	1.732208	0.255692
R	2.485509	0.130541
E	2.492758	0.129775
RA	2.524616	0.126473
SV	2.941436	0.091839
SB	3.283197	0.072161

DATA SCALING:

Data scaling is a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step.

Data Scaling

```
In [38]: from sklearn.preprocessing import StandardScaler
         scaler=StandardScaler()
         x_scaled=scaler.fit_transform(x)
```

Building Machine Learning Models:

In the below piece of code, there is a function called classify where the data splitting is done , model fitting is done and accuracy of the model is calculated.

LogisticRegression , LinearRegression , DecisionTreeClassifier ,RandomForestClassifier , ExtraTreeClassifier , XGBClassifier are implemented by calling the classify function. After fitting with everything mentioned above,The highest accuracy is shown by LinearRegression

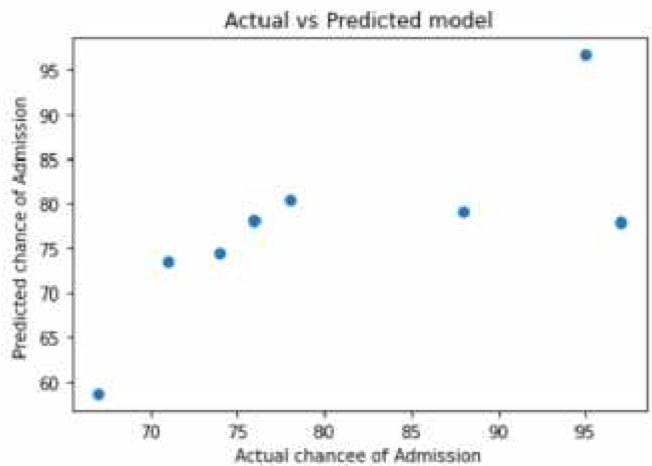
```
In [40]: #Model training
         from sklearn.model_selection import train_test_split,cross_val_score
         def classify (model):
             x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_stat
             model.fit(x_train,y_train)
             print("Accuracy:",model.score(x_test,y_test))
```

```
In [42]: # Linear Regression model
         from sklearn.linear_model import LinearRegression
         model=LinearRegression()
         classify(model)

         Accuracy: 0.8906756651558634
```

The below graph is the predicted vs actual outcome graph

```
In [47]:  x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=

In [48]: model=LinearRegression()
         model.fit(x_train, y_train)

Out[48]: LinearRegression()

In [49]: y_predicted = model.predict(x_test)
         y_predicted

Out[49]: array([74.52025213, 77.84775591, 78.02499087, 58.71913825, 80.44146018,
                78.96373701, 73.3939036 , 96.67416429])
```

```
In [50]: plt.scatter(y_test,y_predicted)
         plt.xlabel('Actual chancee of Admission')
         plt.ylabel('Predicted chance of Admission')
         plt.title('Actual vs Predicted model')
         plt.show()
```



Cross Validation and GridSearchCV is done

Cross-validation is a technique in which we train our model using the subset of the data-set and then evaluate using the complementary subset of the data-set.

```
In [51]: lm = LinearRegression()
         from sklearn.model_selection import RepeatedKFold
         rkf = RepeatedKFold(n_splits=4,n_repeats=2,random_state=True)

         from sklearn.model_selection import cross_val_score
         scores = cross_val_score(lm,x,y,cv=rkf)
         print(scores)
         print("Average score %.2f" % scores.mean())

         [ 0.39669142  0.51602666  0.74669849 -0.29314478 -0.38619091 -0.63323544
           0.34375239  0.84459241]
         Average score 0.19
```

GridSearchCV is a useful tool to fine tune the parameters of your model.

```
In [52]: from sklearn.model_selection import GridSearchCV
         from sklearn.linear_model import Ridge
         ridge_params={'alpha':[1,2,3,4,5,6,7,8,9,10]}
         xg_grid=GridSearchCV(Ridge(),ridge_params,cv=3)
         xg_grid.fit(x_train,y_train)
         print('Best Score:',xg_grid.best_score_)
         print('Best Score:',xg_grid.best_params_)
         print('Best Score:',xg_grid.best_estimator_)

         Best Score: -1.0858097103295599
         Best Score: {'alpha': 10}
         Best Score: Ridge(alpha=10)
```

Below is the piece of code to save the best model as a pickle file

```
In [53]: import pickle
         model=LinearRegression()
         model.fit(x_train, y_train)
         pickle.dump(model, open('baseball.pkl','wb'))

In [54]: # Loading model to compare the results
         loaded = pickle.load(open('baseball.pkl','rb'))

In [55]: y_predicted = model.predict(x_test)
         y_predicted

Out[55]: array([74.52025213, 77.84775591, 78.02499087, 58.71913825, 80.44146018,
                78.96373701, 73.3939036 , 96.67416429])
```

Concluding Remarks:

As discussed in EDA and storytelling the randomness of the dataset should be reduced by reducing the number of variables. Which was done by Cross Validation and GridSearchCV.