

## 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)

## Import Required Library

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pickle
from scipy.stats import zscore
import scikitplot as skplt
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler, power_transform
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
import xgboost as xgb
from sklearn.metrics import classification_report, accuracy_score,roc_auc_score,
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

## Reading Data

```
In [2]: df = pd.read_csv(r"C:\Users\Kushal Arya\Desktop\csv file\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

## Check no of row and column

```
In [3]: print('No of Rows and Columns ----->', df.shape )
```

No of Rows and Columns -----> (30, 17)

## Checking for Null values

```
In [4]: print('-----\n')
print(df.isnull().sum())
print('\n-----')
```

```
-----  
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  
-----
```

**There is no null value**

## Information about dataset

```
In [5]: print('-----\n')
print(df.info())
print('\n-----')
```

```
<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  SH0      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
None
```

**All are int and float value**

## Statistical Analysis of Dataset

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

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

All data is normal distributed

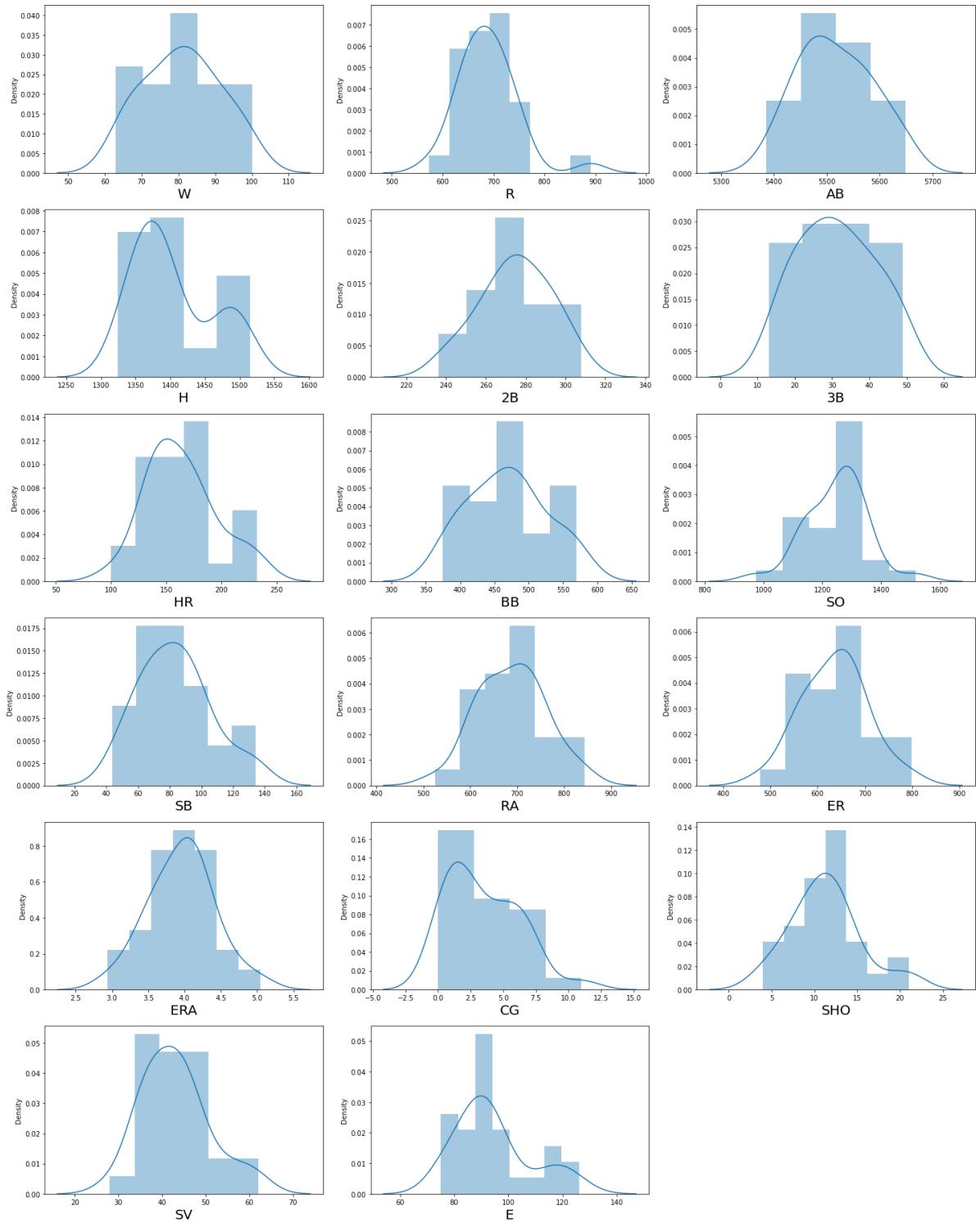
## Checking Outliers

```
In [7]: print('-----')
print('Distribution Plot :- ')
print('-----')

plt.figure(figsize = (20,25))
plotnumber = 1

for column in df:
    if plotnumber <=18:
        ax = plt.subplot(6,3, plotnumber)
        sns.distplot(df[column])
        plt.xlabel(column, fontsize = 20)
    plotnumber +=1
plt.tight_layout()
```

```
-----
Distribution Plot :-
```



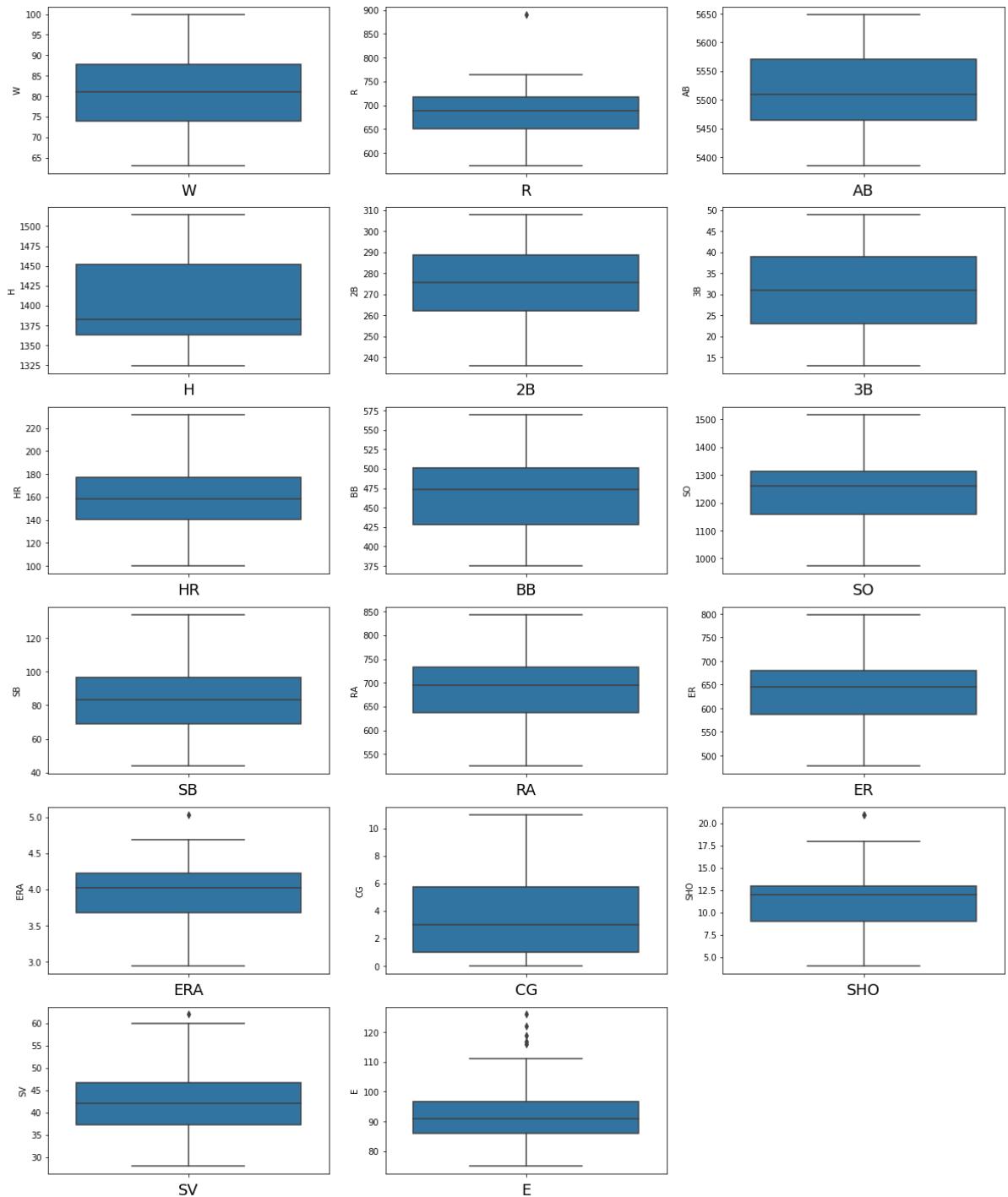
**Features are little skewed**

```
In [8]: # Visualize the outliers using boxplot
print('-----')
print('Box Plot :-')
print('-----')

plt.figure(figsize = (20,25))
graph = 1

for column in df:
    if graph <=18:
        ax = plt.subplot(6,3, graph)
        sns.boxplot(y=df[column]) # It is the axis for vertical set as y
        plt.xlabel(column, fontsize = 18)
    graph +=1
plt.show()
```

```
-----
Box Plot :-
```



**There are some outlier present in our dataset**

## Removing Outliers using Zscore

```
In [10]: # with std 3 Lets see the stats
```

```
z_score = zscore(df[['SV', 'ERA', 'SHO', 'R', 'E']]) # use only continous data
abs_z_score = np.abs(z_score)

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

df = df[filtering_entry]
```

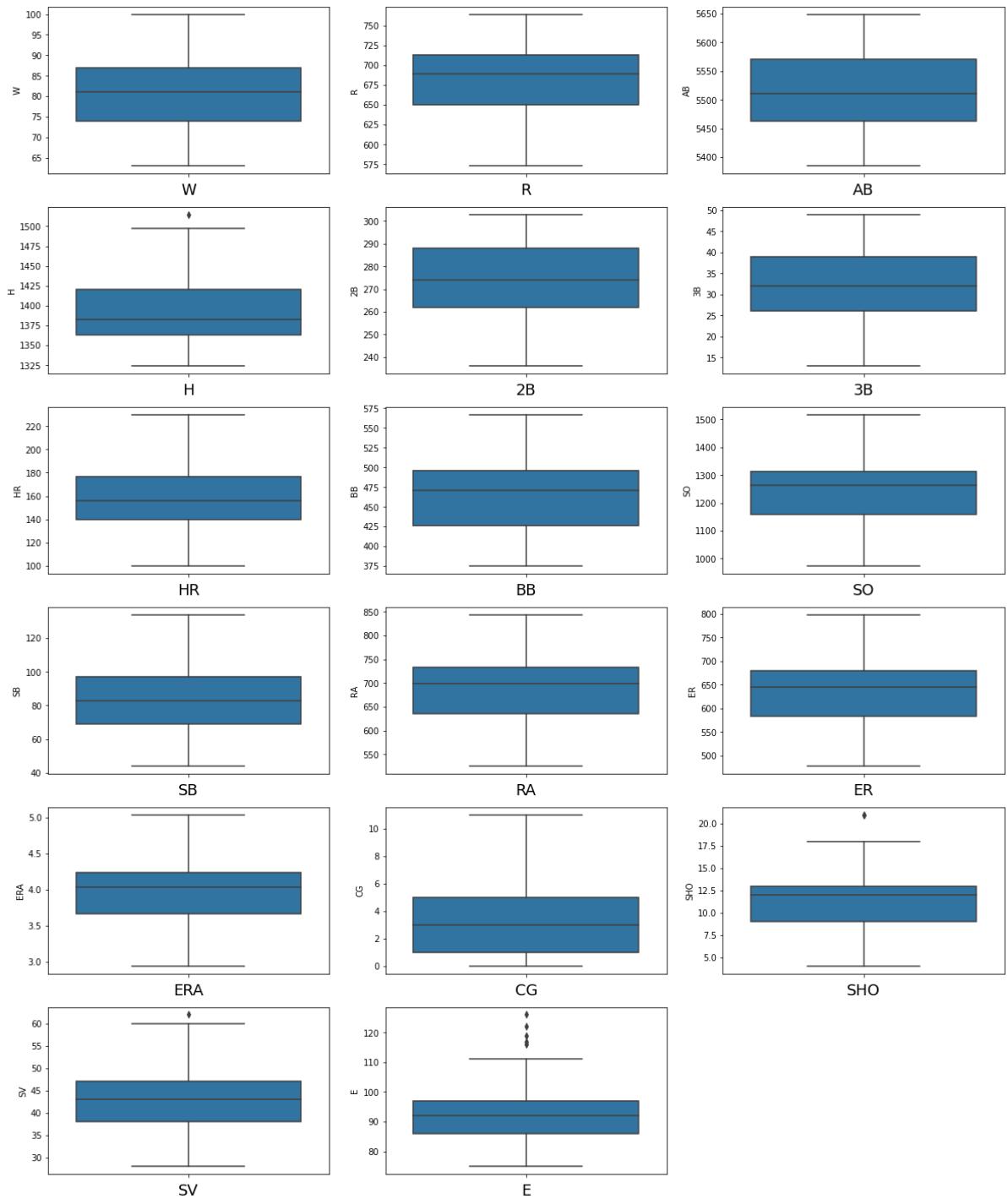
## Checking Outlier Remove or Not

```
In [11]: # Visualize the outliers using boxplot
print('-----')
print('Box Plot :-')
print('-----')

plt.figure(figsize = (20,25))
graph = 1

for column in df:
    if graph <=18:
        ax = plt.subplot(6,3, graph)
        sns.boxplot(y=df[column]) # It is the axis for vertical set as y
        plt.xlabel(column, fontsize = 18)
    graph +=1
plt.show()
```

-----  
Box Plot :-  
-----



**Outliers are Removed**

**Checking for Imbalance Label**

```
In [12]: print('-----')
print('No of wins matches :')
print('-----')
print(df['W'].value_counts())
print('-----')
```

```
-----
No of wins matches :
-----
68      3
74      2
76      2
81      2
83      2
64      1
84      1
100     1
92      1
90      1
88      1
87      1
86      1
85      1
80      1
97      1
79      1
78      1
63      1
71      1
67      1
98      1
95      1
Name: W, dtype: int64
-----
```

**Label are balanced**

## Checking Skewness in Dataset

```
In [13]: print('=====')  
print('Skewness in data :')  
print('=====')  
print(df.skew())  
print('=====')
```

```
=====  
Skewness in data :  
=====  
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  
=====
```

Some features are little skewed

Corelation of Feature vs Label using Heat map

```
In [14]: print('=====')
```

```
print('Heat Map :')
```

```
print('=====')
```

```
df_corr = df.corr().abs()
```

```
plt.figure(figsize = (22,16))
```

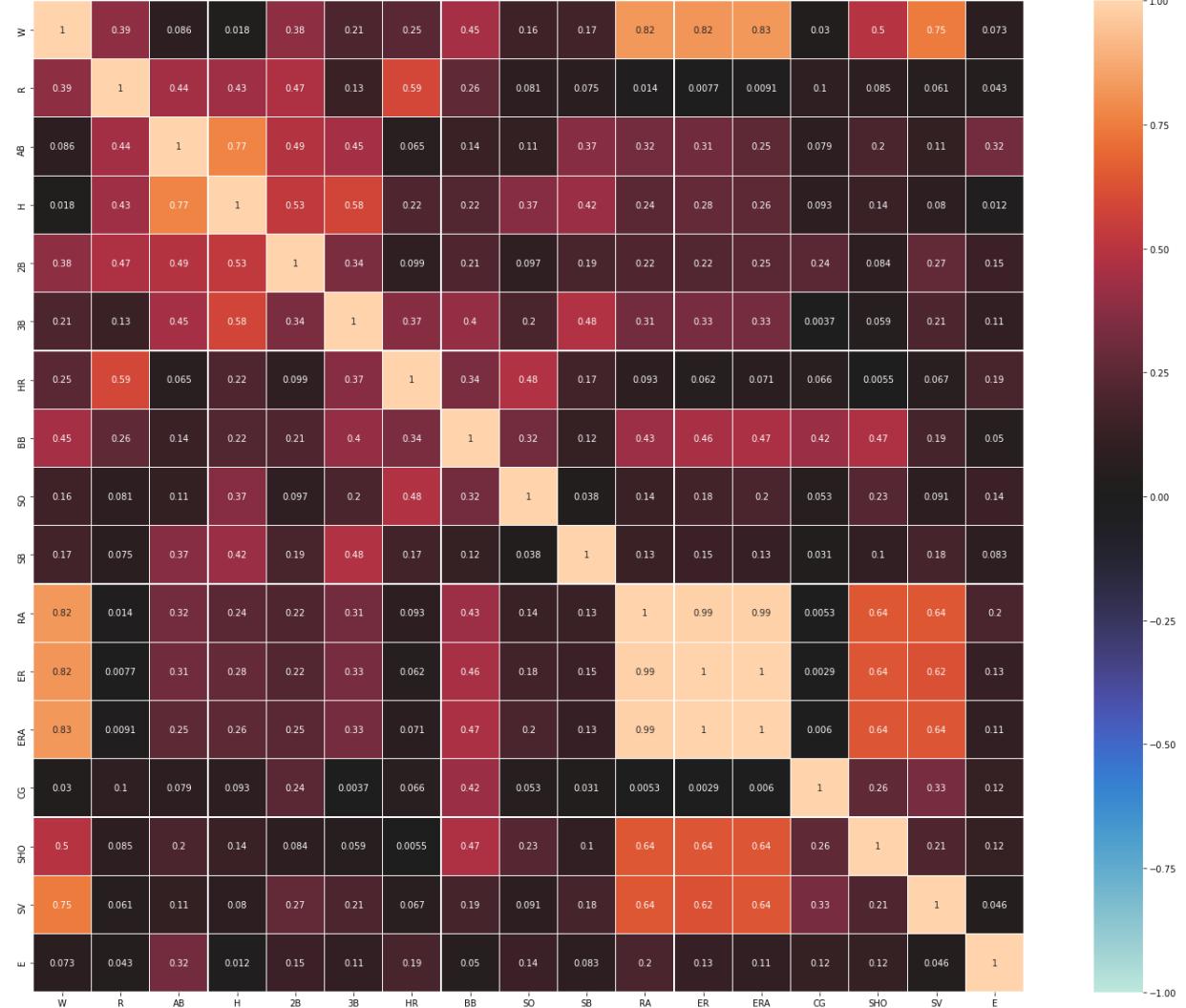
```
sns.heatmap(df_corr, vmin = -1, annot = True, square = True, center = 0, fmt = '.2f')
```

```
plt.tight_layout()
```

```
=====
```

```
Heat Map :
```

```
=====
```



**ERA and ER are show higest corelation and RA show least corelation with label**

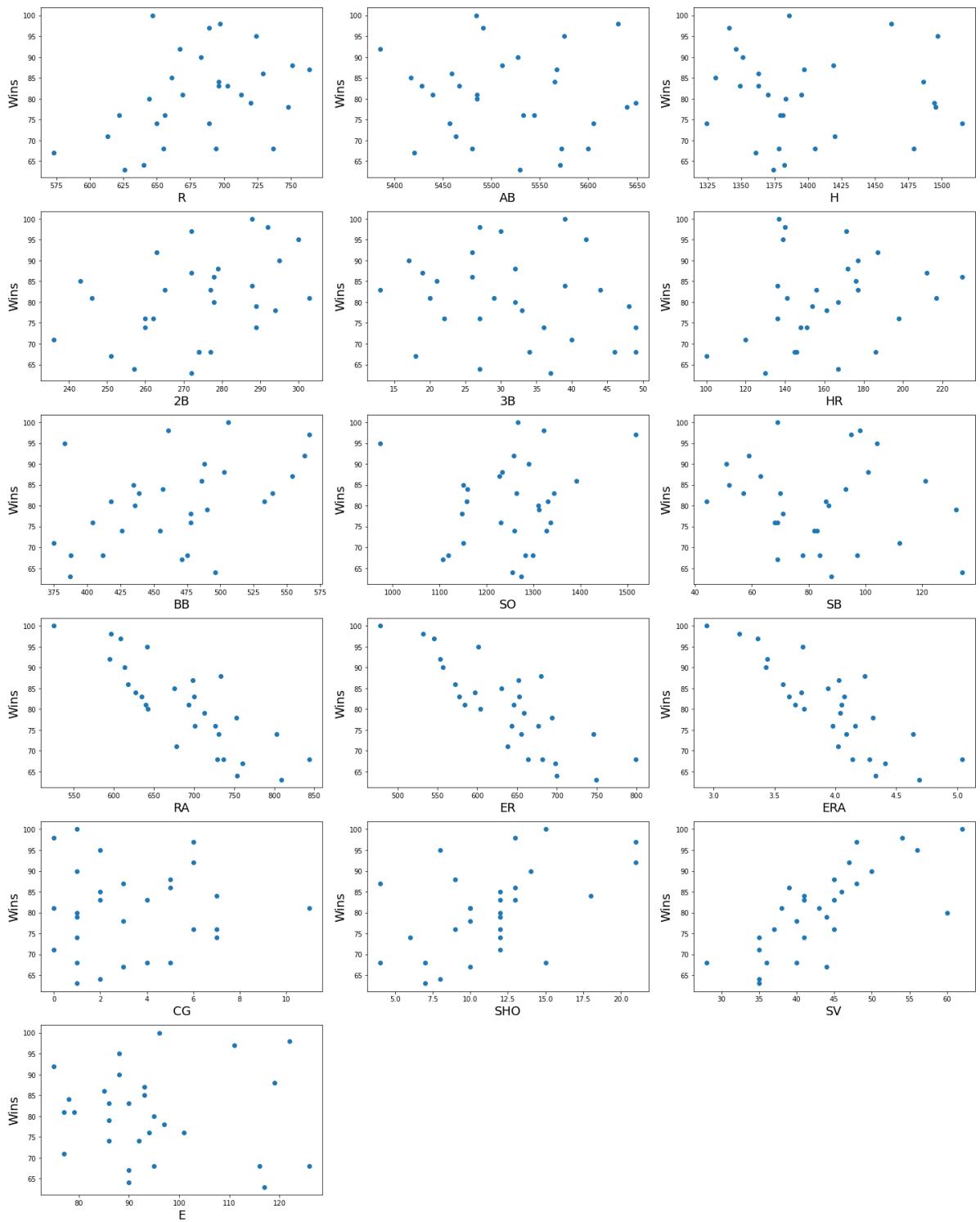
## **Spliting Dataset into features and labels**

```
In [15]: x = df.drop('W', axis = 1)
y = df.W
print('Data has been splited')
```

```
Data has been splited
```

```
In [16]: # Let's see relation between features and labels.  
print('-----')  
print('Distribution Plot :-')  
print('-----')  
  
plt.figure(figsize = (20,25), facecolor = 'white')  
plotnumber = 1  
for column in x:  
    if plotnumber <=18:  
        ax = plt.subplot(6,3, plotnumber)  
        plt.scatter(x[column],y)  
        plt.xlabel(column, fontsize = 18)  
        plt.ylabel('Wins', fontsize = 18)  
    plotnumber += 1  
plt.tight_layout()
```

-----  
Distribution Plot :-  
-----



Above scatter shows positive relation between features and target

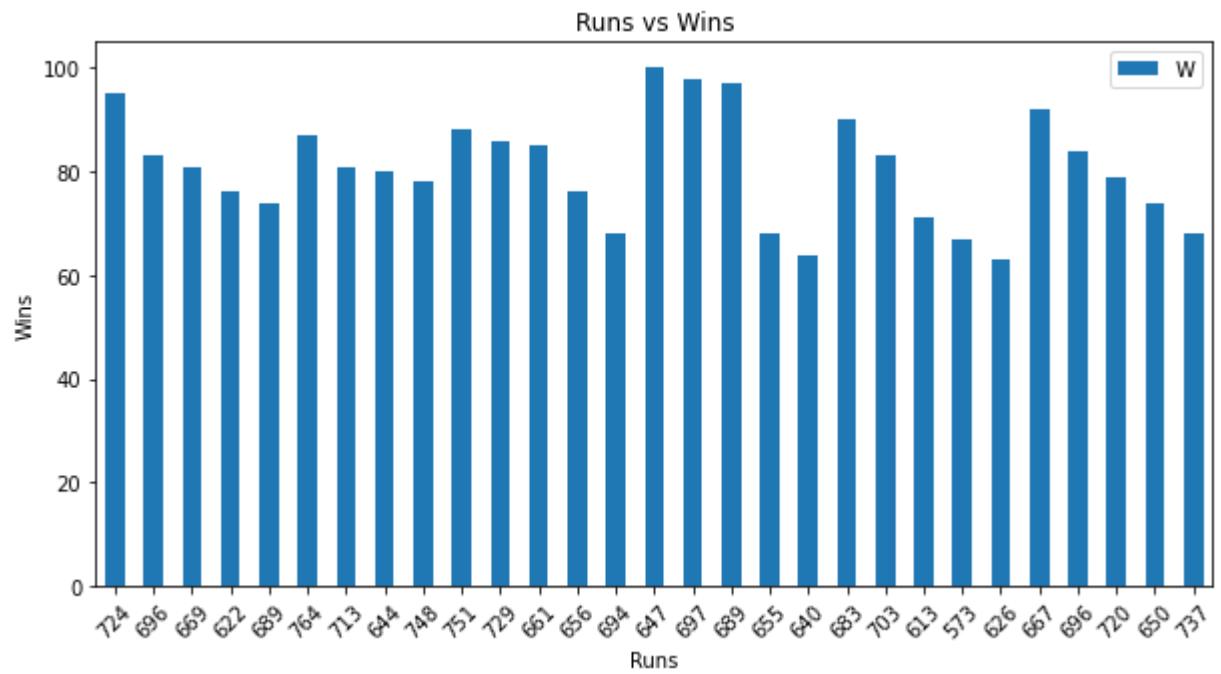
## Analysis of Data

```
In [17]: run = df.groupby('W')[['R','H']].sum()
run
```

Out[17]:

	R	H
W		
<b>63</b>	626	1374
<b>64</b>	640	1382
<b>67</b>	573	1361
<b>68</b>	2086	4262
<b>71</b>	613	1420
<b>74</b>	1339	2839
<b>76</b>	1278	2760
<b>78</b>	748	1495
<b>79</b>	720	1494
<b>80</b>	644	1383
<b>81</b>	1382	2765
<b>83</b>	1399	2712
<b>84</b>	696	1486
<b>85</b>	661	1331
<b>86</b>	729	1363
<b>87</b>	764	1397
<b>88</b>	751	1419
<b>90</b>	683	1351
<b>92</b>	667	1346
<b>95</b>	724	1497
<b>97</b>	689	1341
<b>98</b>	697	1462
<b>100</b>	647	1386

```
In [18]: df.plot.bar( x = 'R', y = 'W', figsize = (10,5), rot = 45)
plt.xlabel('Runs')
plt.ylabel('Wins')
plt.title('Runs vs Wins')
plt.show()
```

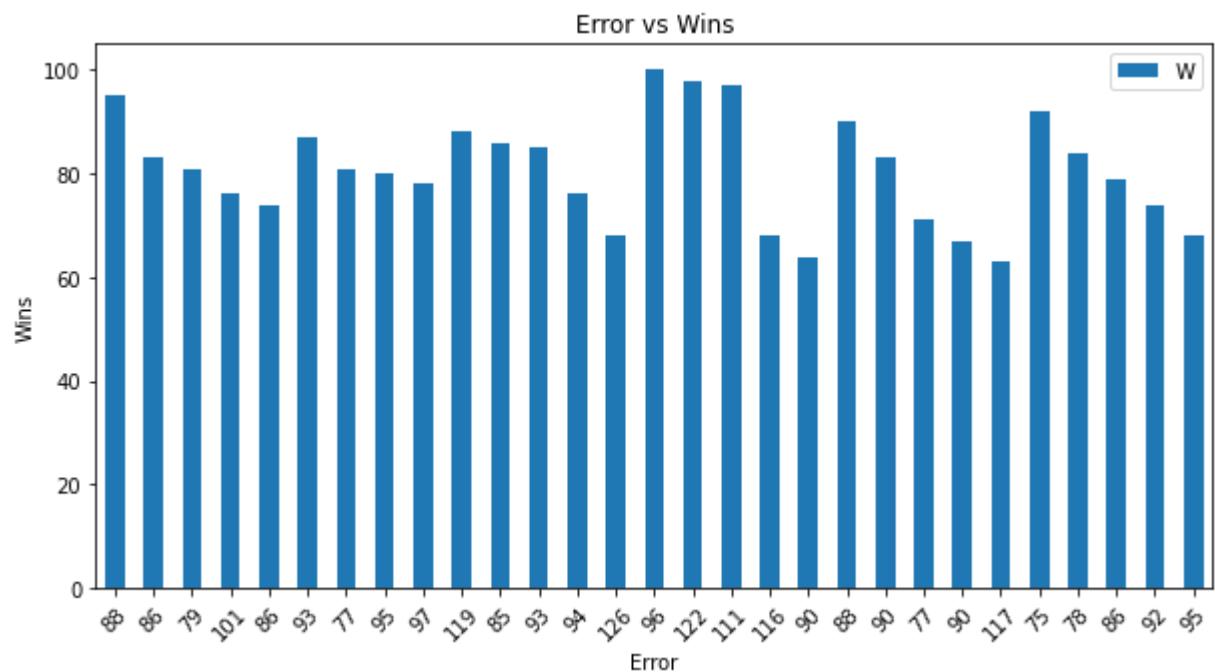


Maximum matches win when team score between 640 to 730 . A maximum Hits run when team score 2000+ run.

```
In [19]: hr = df.groupby('W')['E'].sum()  
hr
```

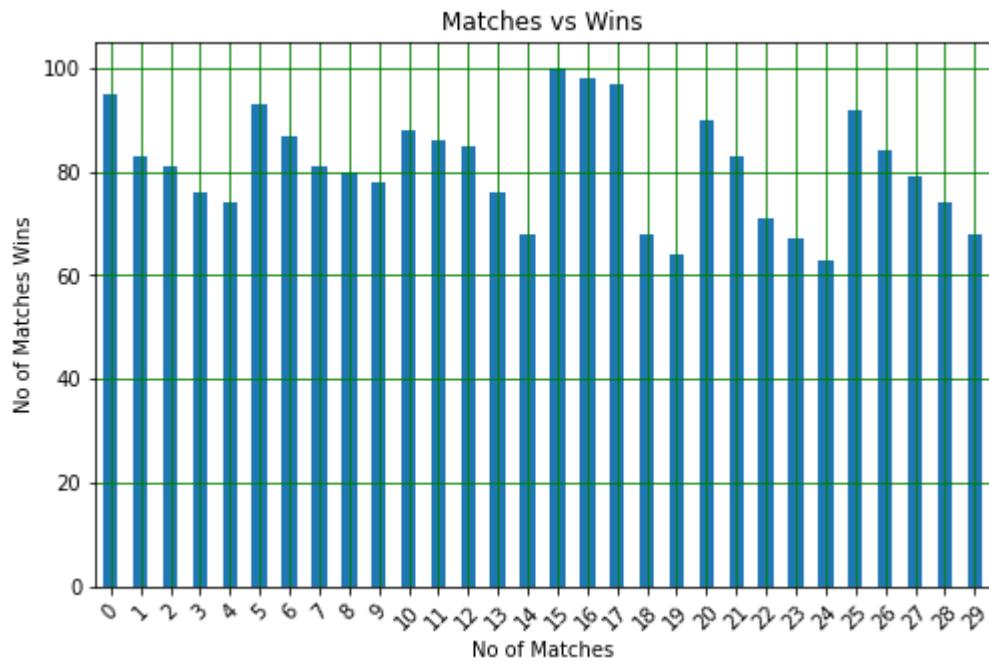
```
Out[19]: W  
63      117  
64       90  
67       90  
68     337  
71       77  
74     178  
76     195  
78       97  
79       86  
80       95  
81     156  
83     176  
84       78  
85       93  
86       85  
87       93  
88     119  
90       88  
92       75  
95       88  
97     111  
98     122  
100      96  
Name: E, dtype: int64
```

```
In [20]: df.plot.bar( x = 'E', y = 'W', figsize = (10,5), rot = 45)  
plt.xlabel('Error')  
plt.ylabel('Wins')  
plt.title('Error vs Wins')  
plt.show()
```



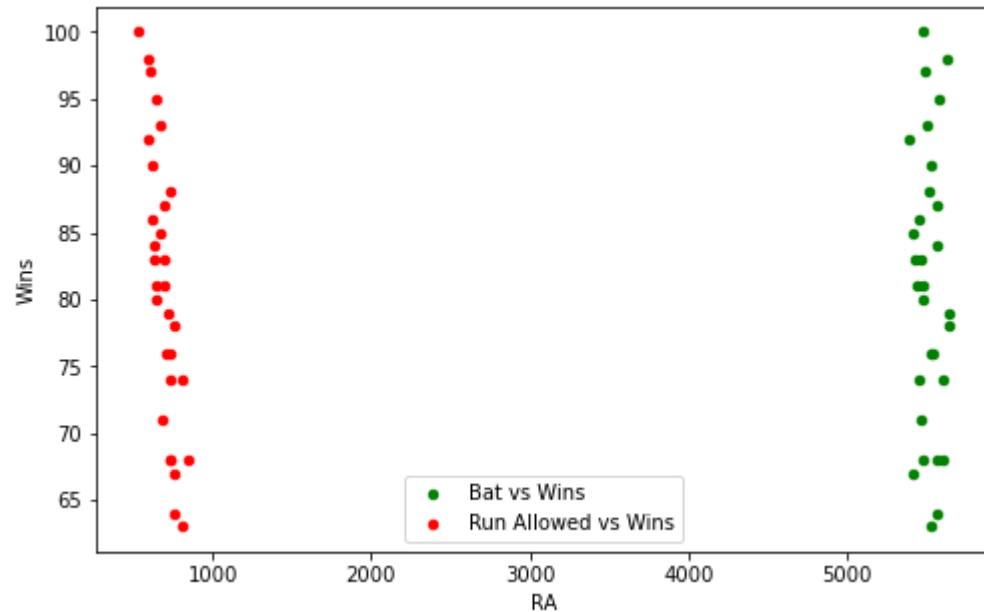
Highest Error in range of 90 to 113.

```
In [26]: df['W'].plot.bar(figsize = (8, 5), rot = 45)
plt.grid(which = 'major', c = 'g')
plt.xlabel('No of Matches')
plt.ylabel('No of Matches Wins')
plt.title('Matches vs Wins')
plt.show()
```



Range 15 to 17 higest matches wins

```
In [33]: ax = df.plot.scatter(x = 'AB', y = 'W' , label = 'Bat vs Wins', color = 'g')
df.plot.scatter(x = 'RA', y = 'W' , label = 'Run Allowed vs Wins', ax = ax, color
plt.ylabel('Wins')
plt.show()
```



It shows positive relation

## Power Transform

```
In [21]: x = power_transform(x)
```

```
In [22]: x
```

```
Out[22]: array([[ 9.62543504e-01,  0.00000000e+00,  0.00000000e+00,
   1.68518793e+00,  1.00615029e+00, -7.41927000e-01,
  -1.60519802e+00, -2.55061247e+00,  9.36131648e-01,
  -6.60978697e-01, -5.08052227e-01, -5.09292146e-01,
  -3.07098204e-01, -7.87002186e-01,  1.53275292e+00,
  -3.48265262e-01],
 [ 2.98863300e-01,  0.00000000e+00,  0.00000000e+00,
  1.38197902e-01,  1.18522654e+00, -1.09958425e-01,
  -4.62095966e-01,  9.36832915e-02, -5.16377335e-01,
  1.60225829e-01,  2.35800484e-01,  2.41440214e-01,
  -3.07098204e-01,  2.36736538e-01,  3.12020186e-01,
  -5.40819806e-01],
 [-3.12105130e-01,  0.00000000e+00,  0.00000000e+00,
  1.90738550e+00, -2.28819392e-01, -6.64354121e-01,
  1.23209786e+00, -9.35611465e-01,  2.25038365e-01,
  -6.74967476e-01, -7.52213883e-01, -6.42097599e-01,
  2.01131531e+00, -2.52844176e-01, -6.64136739e-01,
  -1.32612477e+00],
 [-1.30829774e+00,  0.00000000e+00,  0.00000000e+00,
```

## Handling Class Imbalance

```
In [23]: from imblearn.over_sampling import RandomOverSampler  
ros = RandomOverSampler()  
x_over, y_over = ros.fit_resample(x, y)
```

```
In [24]: print('-----')  
print('Class are balanced :-')  
print('-----')  
print(y_over.value_counts())  
print('-----')
```

```
-----  
Class are balanced :-  
-----  
63      3  
84      3  
98      3  
97      3  
95      3  
92      3  
90      3  
88      3  
87      3  
86      3  
85      3  
83      3  
64      3  
81      3  
80      3  
79      3  
78      3  
76      3  
74      3  
71      3  
68      3  
67      3  
100     3  
Name: W, dtype: int64  
-----
```

**Class are balanced**

## Data Scaling

```
In [25]: scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)
x_scaled
```

```
Out[25]: array([[ 9.62543504e-01,  0.00000000e+00,  0.00000000e+00,
   1.68518793e+00,  1.00615029e+00, -7.41927000e-01,
  -1.60519802e+00, -2.55061247e+00,  9.36131648e-01,
  -6.60978697e-01, -5.08052227e-01, -5.09292146e-01,
  -3.07098204e-01, -7.87002186e-01,  1.53275292e+00,
  -3.48265262e-01],
 [ 2.98863300e-01,  0.00000000e+00,  0.00000000e+00,
  1.38197902e-01,  1.18522654e+00, -1.09958425e-01,
  -4.62095966e-01,  9.36832915e-02, -5.16377335e-01,
  1.60225829e-01,  2.35800484e-01,  2.41440214e-01,
  -3.07098204e-01,  2.36736538e-01,  3.12020186e-01,
  -5.40819806e-01],
 [-3.12105130e-01,  0.00000000e+00,  0.00000000e+00,
  1.90738550e+00, -2.28819392e-01, -6.64354121e-01,
  1.23209786e+00, -9.35611465e-01,  2.25038365e-01,
  -6.74967476e-01, -7.52213883e-01, -6.42097599e-01,
  2.01131531e+00, -2.52844176e-01, -6.64136739e-01,
  -1.32612477e+00],
 [-1.30829774e+00,  0.00000000e+00,  0.00000000e+00,
  2.37664772e-01, -1.32227007e-01, -2.60020212e-01]]
```

Data has been scaled

**Split data into train and test. Model will be bulit on training data and tested on test data**

```
In [26]: x_train, x_test, y_train, y_test = train_test_split(x_over, y_over, test_size = 0.2)
print('Data has been splited.')
```

Data has been splited.

## Model Building

**Decision Tree model instantiaing, training and evaluating**

```
In [27]: DT = DecisionTreeClassifier()
DT.fit(x_train, y_train)
y_pred = DT.predict(x_test)
```

```
In [28]: print('-----\n')
print('\nClassification Report:')
print(classification_report(y_test, y_pred, digits = 2))
print('-----\n')
```

Classification Report:				
	precision	recall	f1-score	support
63	0.00	0.00	0.00	0
67	1.00	1.00	1.00	1
68	0.00	0.00	0.00	1
71	1.00	1.00	1.00	2
74	0.00	0.00	0.00	0
76	0.00	0.00	0.00	2
78	1.00	1.00	1.00	2
79	1.00	1.00	1.00	1
80	1.00	1.00	1.00	1
81	1.00	1.00	1.00	1
83	1.00	1.00	1.00	1
84	1.00	1.00	1.00	2
85	1.00	1.00	1.00	1
92	1.00	1.00	1.00	1
95	1.00	1.00	1.00	1
100	1.00	1.00	1.00	1
accuracy			0.83	18
macro avg	0.75	0.75	0.75	18
weighted avg	0.83	0.83	0.83	18

Conclusion : Decision Tree model has 83% score

## Cross Validation score to check if the model is overfitting

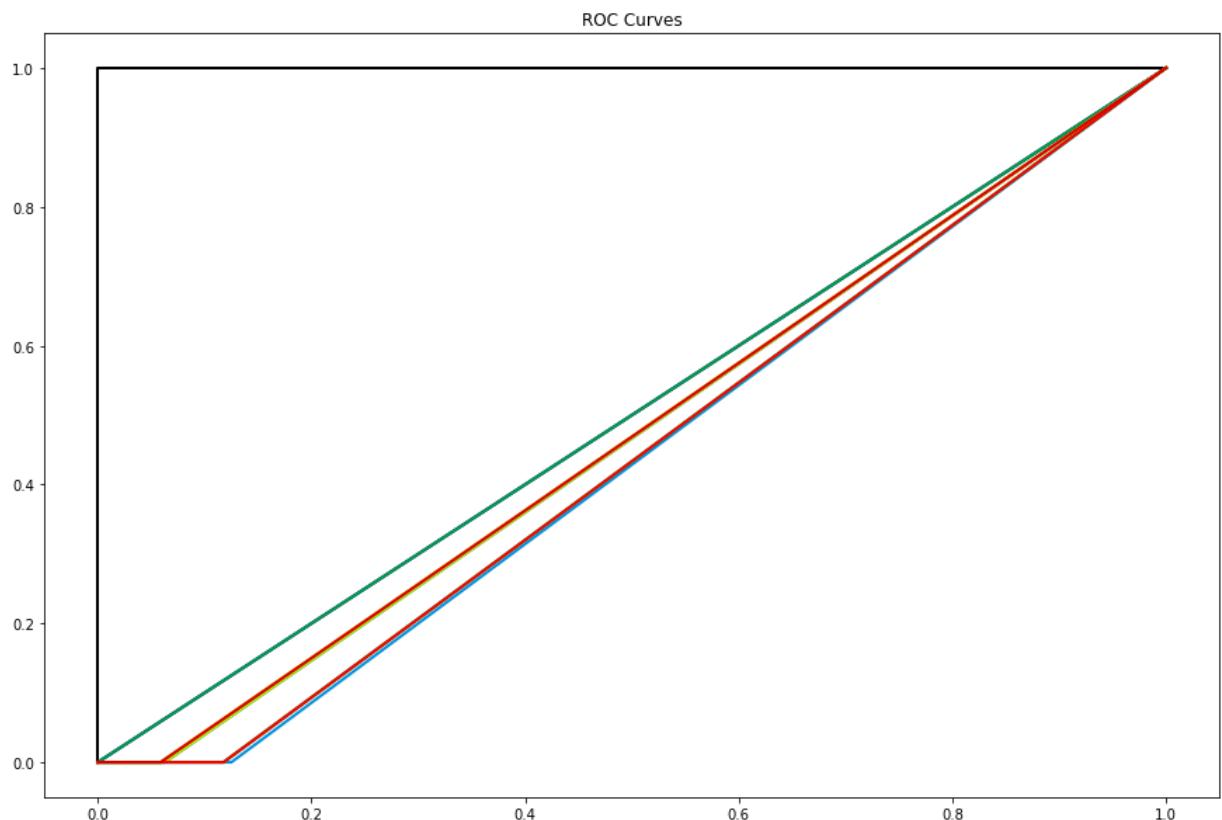
```
In [54]: cv = cross_val_score(DT, x, y, cv = 3)
print('Cross Validation score of Decision Tree model --->', cv.mean())
```

Cross Validation score of Decision Tree model ---> 0.037037037037037035

Conclusion : Decision Tree model has 3% Cross Validation score

## ROC, AUC Curve

```
In [30]: try:  
    prob = DT.predict_proba(x_test) # calculating probability  
    skplt.metrics.plot_roc(y_pred,prob, figsize = (15,10))  
    plt.show()  
except ValueError:  
    pass
```



## Knn model instantiaing, training and evaluating

```
In [31]: Knn = KNeighborsClassifier()  
Knn.fit(x_train, y_train)  
y_pred = Knn.predict(x_test)
```

```
In [32]: print('-----\n')
print('\nClassification Report:')
print(classification_report(y_test, y_pred, digits = 2))
print('-----\n')
```

Classification Report:				
	precision	recall	f1-score	support
63	0.00	0.00	0.00	0
64	0.00	0.00	0.00	0
67	1.00	1.00	1.00	1
68	0.00	0.00	0.00	1
71	0.00	0.00	0.00	2
74	0.00	0.00	0.00	0
76	0.00	0.00	0.00	2
78	0.00	0.00	0.00	2
79	0.33	1.00	0.50	1
80	1.00	1.00	1.00	1
81	0.00	0.00	0.00	1
83	0.00	0.00	0.00	1
84	0.00	0.00	0.00	2
85	1.00	1.00	1.00	1
88	0.00	0.00	0.00	0
90	0.00	0.00	0.00	0
92	0.00	0.00	0.00	1
95	1.00	1.00	1.00	1
98	0.00	0.00	0.00	0
100	0.00	0.00	0.00	1
accuracy			0.28	18
macro avg	0.22	0.25	0.23	18
weighted avg	0.24	0.28	0.25	18

Conclusion : Knn model has 28% score

## Cross Validation score to check if the model is overfitting

```
In [58]: cv = cross_val_score(Knn, x, y, cv = 3)
print('Cross Validation score of Knn model --->', cv.mean())
```

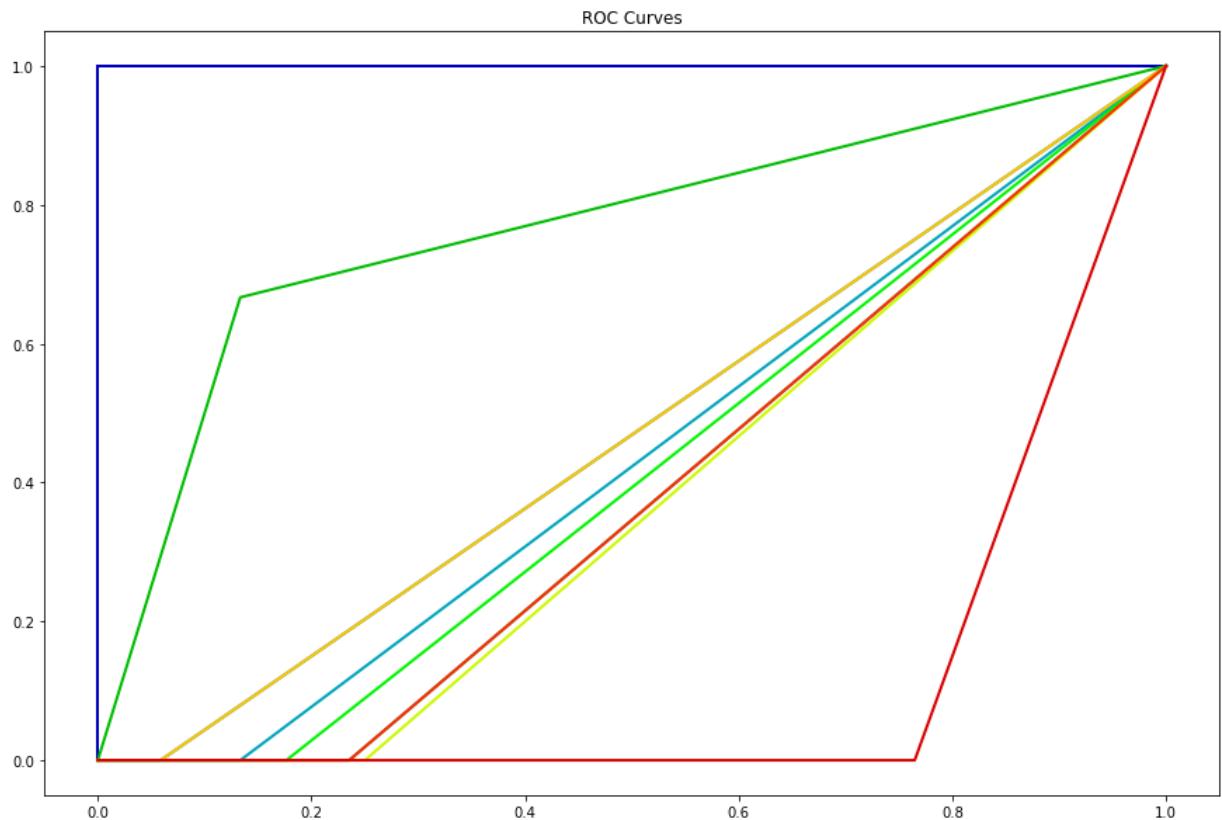
Cross Validation score of Knn model ---> 0.0

Conclusion : Knn model has 0% Cross Validation score

## ROC, AUC Curve

```
In [34]:
```

```
try:  
    prob = Knn.predict_proba(x_test) # calculating probability  
    skplt.metrics.plot_roc(y_pred,prob, figsize = (15,10))  
    plt.show()  
except ValueError:  
    pass
```



## Random Forest model instantiating, training and evaluating

```
In [35]:
```

```
Rn = RandomForestClassifier()  
Rn.fit(x_train, y_train)  
y_pred = Rn.predict(x_test)
```

```
In [36]: print('-----\n')
print('\nClassification Report:')
print(classification_report(y_test, y_pred, digits = 2))
print('-----\n')
```

Classification Report:				
	precision	recall	f1-score	support
63	0.00	0.00	0.00	0
67	1.00	1.00	1.00	1
68	0.00	0.00	0.00	1
71	1.00	1.00	1.00	2
74	0.00	0.00	0.00	0
76	0.00	0.00	0.00	2
78	1.00	1.00	1.00	2
79	1.00	1.00	1.00	1
80	1.00	1.00	1.00	1
81	1.00	1.00	1.00	1
83	1.00	1.00	1.00	1
84	1.00	1.00	1.00	2
85	1.00	1.00	1.00	1
92	1.00	1.00	1.00	1
95	1.00	1.00	1.00	1
100	1.00	1.00	1.00	1
accuracy			0.83	18
macro avg	0.75	0.75	0.75	18
weighted avg	0.83	0.83	0.83	18

Conclusion : Random Forest model has 83% score

## Cross Validation score to check if the model is overfitting

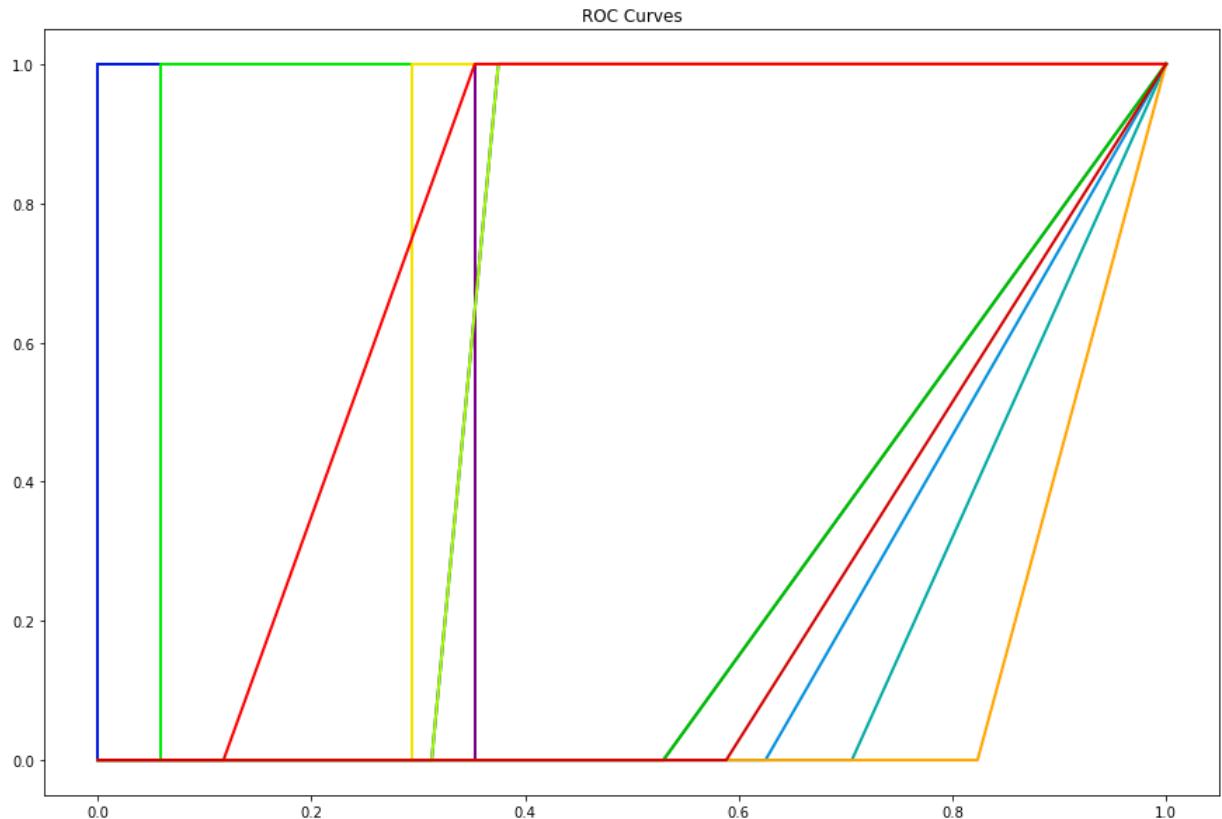
```
In [61]: cv = cross_val_score(Rn, x, y, cv = 3)
print('Cross Validation score of Random Forest model --->', cv.mean())
```

Cross Validation score of Random Forest model ---> 0.0

Conclusion : Random Forest model has 3% Cross Validation score

## ROC, AUC Curve

```
In [38]: try:  
    prob = Rn.predict_proba(x_test) # calculating probability  
    skplt.metrics.plot_roc(y_pred,prob, figsize = (15,10))  
    plt.show()  
except ValueError:  
    pass
```



## SVM model instantiating, training and evaluating

```
In [39]: svm = SVC(probability=True)  
svm.fit(x_train, y_train)  
y_pred = svm.predict(x_test)
```

```
In [40]: print('-----\n')
print('\nClassification Report:')
print(classification_report(y_test, y_pred, digits = 2))
print('-----\n')
```

	precision	recall	f1-score	support
67	1.00	1.00	1.00	1
68	0.00	0.00	0.00	1
71	1.00	1.00	1.00	2
74	0.00	0.00	0.00	0
76	0.00	0.00	0.00	2
78	0.00	0.00	0.00	2
79	1.00	1.00	1.00	1
80	1.00	1.00	1.00	1
81	0.33	1.00	0.50	1
83	0.00	0.00	0.00	1
84	0.00	0.00	0.00	2
85	1.00	1.00	1.00	1
88	0.00	0.00	0.00	0
92	1.00	1.00	1.00	1
95	1.00	1.00	1.00	1
100	1.00	1.00	1.00	1
accuracy			0.56	18
macro avg	0.52	0.56	0.53	18
weighted avg	0.52	0.56	0.53	18

Conclusion : SVM model has 56% score

## Cross Validation score to check if the model is overfitting

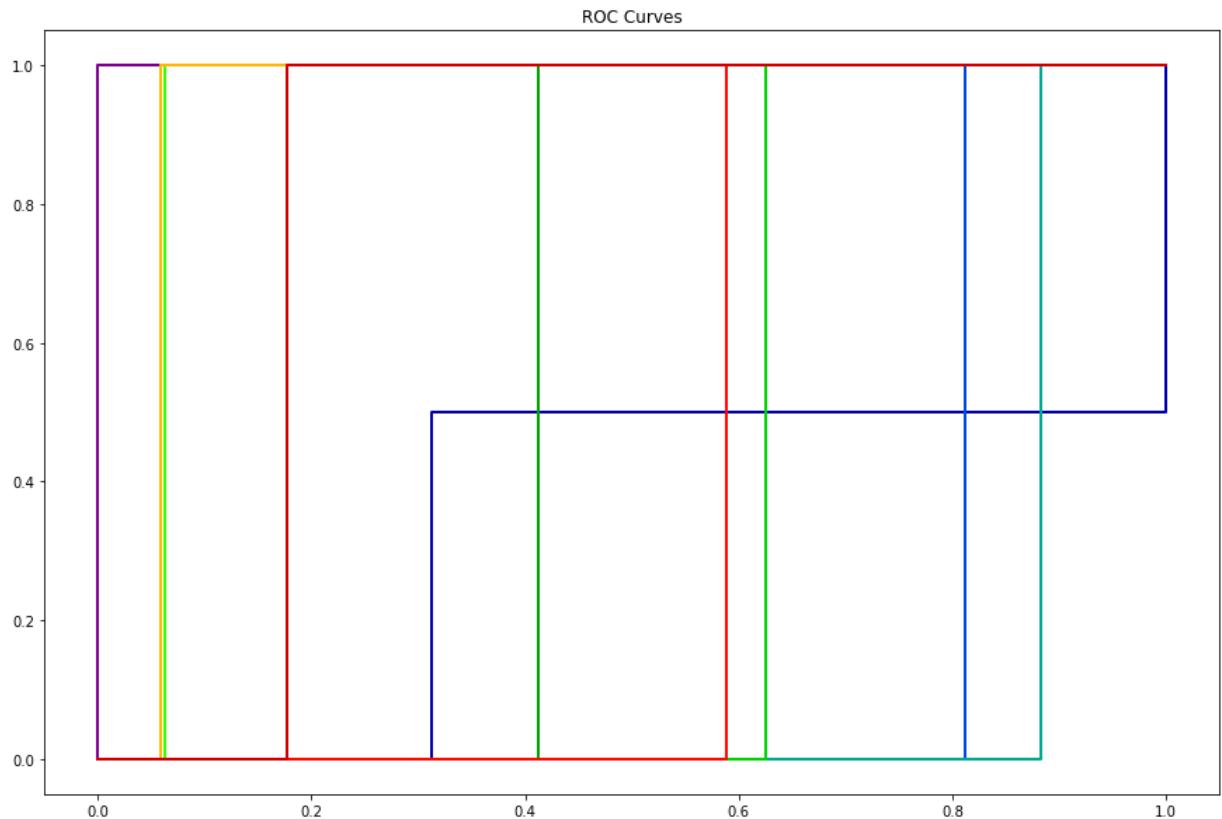
```
In [41]: cv = cross_val_score(svm, x, y, cv = 3)
print('Cross Validation score of Knn model --->', cv.mean())
```

Cross Validation score of Knn model ---> 0.037037037037037035

Conclusion : Knn model has 3% Cross Validation score

## ROC, AUC Curve

```
In [71]: try:  
    prob = svm.predict_proba(x_test) # calculating probability  
    skplt.metrics.plot_roc(y_pred,prob, figsize = (15,10))  
    plt.show()  
except ValueError:  
    pass
```



## XGBoost model instantiaing, training and evaluating

```
In [43]: xgb = xgb.XGBClassifier(eval_metric='mlogloss')  
xgb.fit(x_train, y_train)  
y_pred = xgb.predict(x_test)
```

```
In [44]: print('-----\n')
print('\nClassification Report:')
print(classification_report(y_test, y_pred, digits = 2))
print('-----\n')
```

Classification Report:				
	precision	recall	f1-score	support
63	0.00	0.00	0.00	0
67	1.00	1.00	1.00	1
68	0.00	0.00	0.00	1
71	1.00	1.00	1.00	2
76	0.00	0.00	0.00	2
78	1.00	1.00	1.00	2
79	1.00	1.00	1.00	1
80	1.00	1.00	1.00	1
81	1.00	1.00	1.00	1
83	0.33	1.00	0.50	1
84	1.00	1.00	1.00	2
85	1.00	1.00	1.00	1
92	1.00	1.00	1.00	1
95	1.00	1.00	1.00	1
100	1.00	1.00	1.00	1
accuracy			0.83	18
macro avg	0.76	0.80	0.77	18
weighted avg	0.80	0.83	0.81	18

Conclusion : XGB model has 83% score

## Cross Validation score to check if the model is overfitting

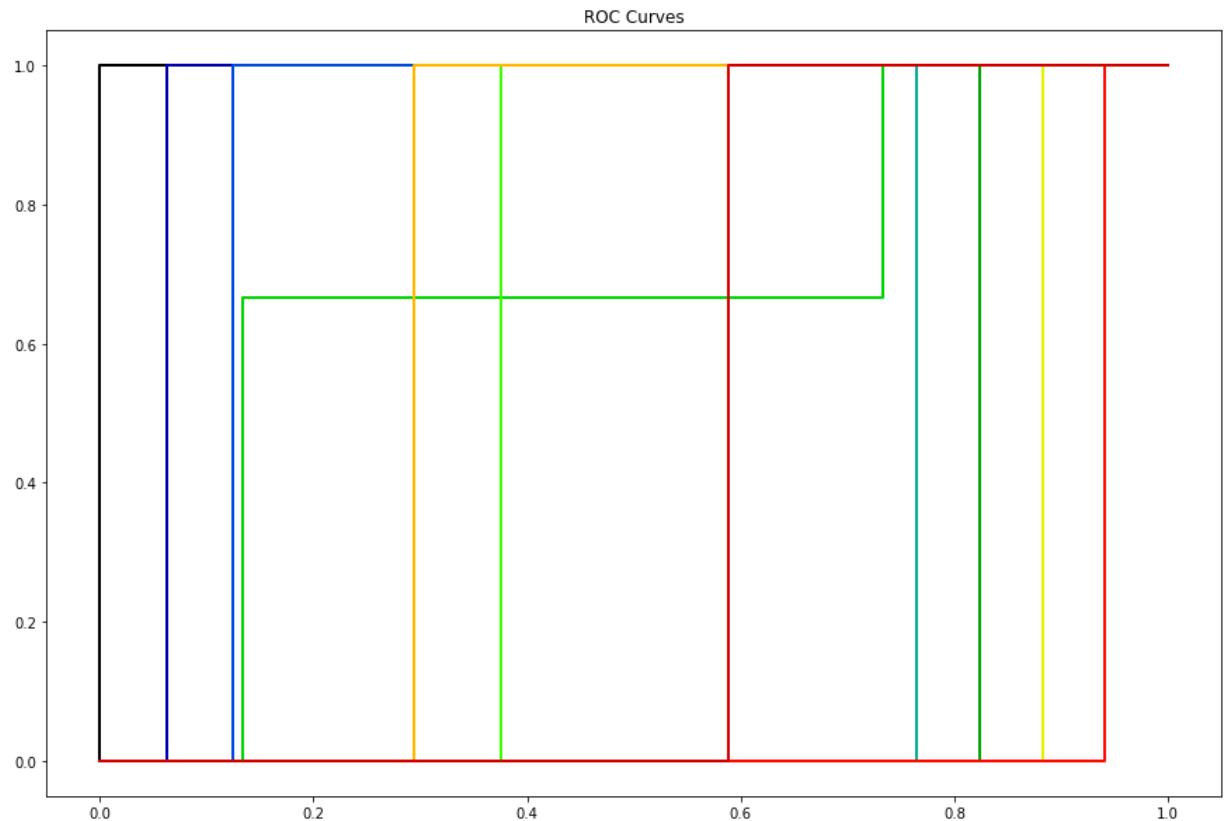
```
In [45]: cv = cross_val_score(xgb, x, y, cv = 3)
print('Cross Validation score of XGB model --->', cv.mean())
```

Cross Validation score of XGB model ---> 0.0

Conclusion : XGB model has 0% Cross Validation score

## ROC, AUC Curve

```
In [46]: try:  
    prob = xgb.predict_proba(x_test) # calculating probability  
    skplt.metrics.plot_roc(y_pred,prob, figsize = (15,10))  
    plt.show()  
except ValueError:  
    pass
```



## Logistic Regression model instantiating, training and evaluating

```
In [47]: Lg = LogisticRegression()  
Lg.fit(x_train, y_train)  
y_pred = Lg.predict(x_test)
```

```
In [48]: print('-----\n')
print('\nClassification Report:')
print(classification_report(y_test, y_pred, digits = 2))
print('-----\n')
```

	precision	recall	f1-score	support
67	1.00	1.00	1.00	1
68	0.00	0.00	0.00	1
71	1.00	1.00	1.00	2
74	0.00	0.00	0.00	0
76	0.00	0.00	0.00	2
78	1.00	1.00	1.00	2
79	1.00	1.00	1.00	1
80	1.00	1.00	1.00	1
81	1.00	1.00	1.00	1
83	0.00	0.00	0.00	1
84	1.00	1.00	1.00	2
85	1.00	1.00	1.00	1
92	1.00	1.00	1.00	1
95	1.00	1.00	1.00	1
100	1.00	1.00	1.00	1
accuracy			0.78	18
macro avg	0.73	0.73	0.73	18
weighted avg	0.78	0.78	0.78	18

Conclusion : Logistic Regression model has 78% score

## Cross Validation score to check if the model is overfitting

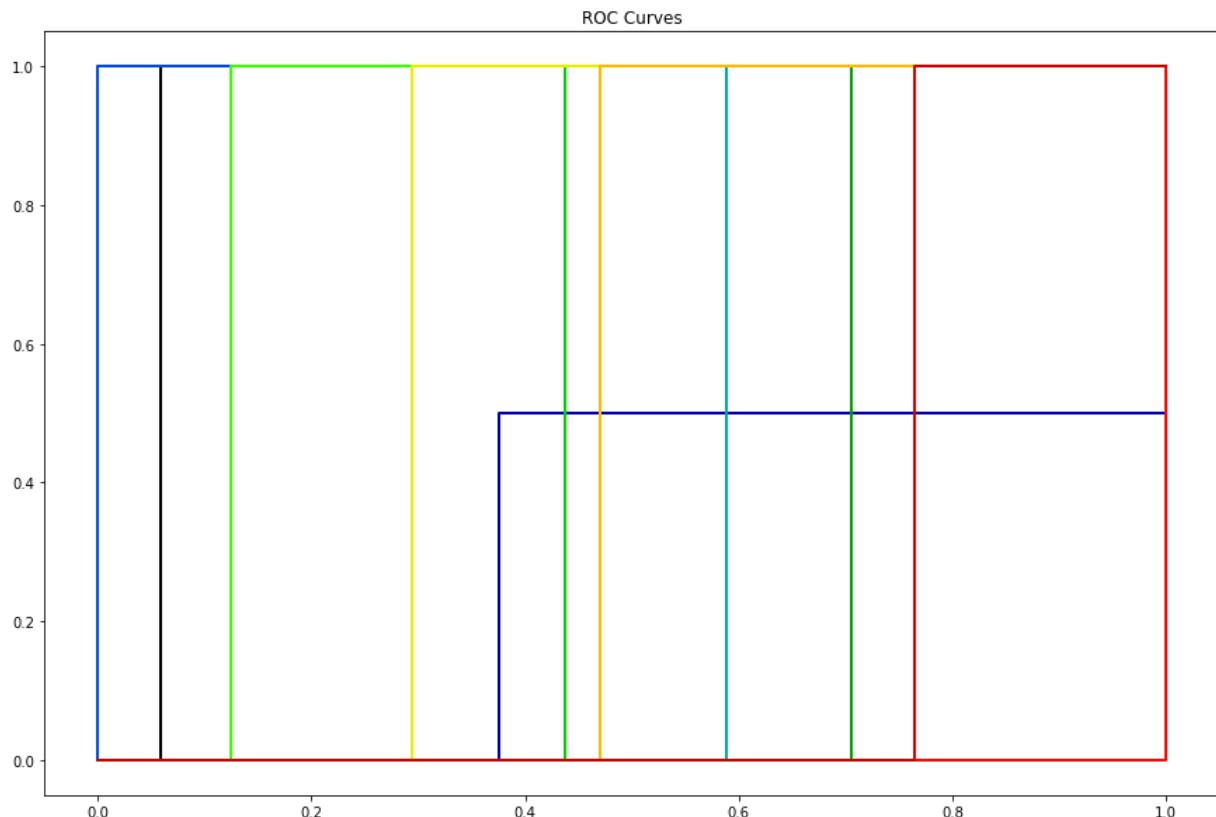
```
In [49]: cv = cross_val_score(Lg, x, y, cv = 3)
print('Cross Validation score of Logistic Regression model --->', cv.mean())
```

Cross Validation score of Logistic Regression model ---> 0.0

Conclusion : Logistic Regression model has 0% Cross Validation score

## ROC, AUC Curve

```
In [50]: try:  
    prob = Lg.predict_proba(x_test) # calculating probability  
    skplt.metrics.plot_roc(y_pred,prob, figsize = (15,10))  
    plt.show()  
except ValueError:  
    pass
```



## Looking CV score we select KNN for Hyperparameter Tuning

```
In [62]: grid_param = {'leaf_size' : [1,3,5], 'n_neighbors': [3],  
                   'p':[1,2,3,4]}
```

```
In [63]: grid_search = GridSearchCV(estimator = Knn, param_grid = grid_param, cv = 3 , n_
```

```
In [64]: grid_search.fit(x_train, y_train)
```

```
Out[64]: GridSearchCV(cv=3, estimator=KNeighborsClassifier(), n_jobs=-1,  
                      param_grid={'leaf_size': [1, 3, 5], 'n_neighbors': [3],  
                                  'p': [1, 2, 3, 4]})
```

```
In [65]: best_parameters = grid_search.best_params_
print(best_parameters)

{'leaf_size': 1, 'n_neighbors': 3, 'p': 2}
```

```
In [66]: hkn = KNeighborsClassifier(leaf_size = 1, n_neighbors = 3, p = 2)
hkn.fit(x_train, y_train)
hkn.score(x_test, y_test)
```

```
Out[66]: 0.3888888888888889
```

```
In [67]: print('-----')
print('\nClassification Report:')
print(classification_report(y_test, y_pred, digits = 2))
print('-----')
```

```
-----  
Classification Report:  
precision    recall   f1-score   support  
  
      67       1.00     1.00     1.00      1  
      68       0.00     0.00     0.00      1  
      71       1.00     1.00     1.00      2  
      74       0.00     0.00     0.00      0  
      76       0.00     0.00     0.00      2  
      78       1.00     1.00     1.00      2  
      79       1.00     1.00     1.00      1  
      80       1.00     1.00     1.00      1  
      81       1.00     1.00     1.00      1  
      83       0.00     0.00     0.00      1  
      84       1.00     1.00     1.00      2  
      85       1.00     1.00     1.00      1  
      92       1.00     1.00     1.00      1  
      95       1.00     1.00     1.00      1  
     100       1.00     1.00     1.00      1  
  
          accuracy                           0.78      18  
     macro avg       0.73     0.73     0.73      18  
weighted avg       0.78     0.78     0.78      18  
-----
```

After Hyperparameter Tuning model accuracy score increase to 78% .

## Saving The Model

```
In [69]: # saving the model to the Local file system
filename = 'Base ball project.pickle'
pickle.dump(hkn, open(filename, 'wb'))
```

**Final Conclusion : Knn is our best model.**

In [ ]: