In [1]:
```
import pandas as pd
import warnings
import os
warnings.filterwarnings('ignore')
os.chdir("E:\Ginu_StudyMaterials\Sem2\MachineLearning")
```

In [2]:
```
# reading and printing dataset
dataset = pd.read_csv("wnba_team_elo_ratings.csv", encoding = 'unicode_escape')
dataset
```

Out[2]:

	season	date	team1	team2	name1	name2	neutral	playoff	score1	score2	elo1_pre	elo2_pre	elo1_post	elo2_post	p
0	2019	10/10/2019	WAS	CON	Washington Mystics	Connecticut Sun	0	1	89	78	1684	1634	1692	1627	
1	2019	10/10/2019	CON	WAS	Connecticut Sun	Washington Mystics	0	1	78	89	1634	1684	1627	1692	
2	2019	10/8/2019	WAS	CON	Washington Mystics	Connecticut Sun	0	1	86	90	1693	1626	1684	1634	
3	2019	10/8/2019	CON	WAS	Connecticut Sun	Washington Mystics	0	1	90	86	1626	1693	1634	1684	
4	2019	10/6/2019	WAS	CON	Washington Mystics	Connecticut Sun	0	1	94	81	1671	1648	1693	1626	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10483	1997	6/21/1997	SAC	LVA	Sacramento Monarchs	Utah Starzz	0	0	73	61	1500	1500	1521	1479	
10484	1997	6/21/1997	NYL	LAS	New York Liberty	Los Angeles Sparks	0	0	67	57	1500	1500	1519	1481	
10485	1997	6/21/1997	LAS	NYL	Los Angeles Sparks	New York Liberty	0	0	57	67	1500	1500	1481	1519	
10486	1997	6/21/1997	LVA	SAC	Utah Starzz	Sacramento Monarchs	0	0	61	73	1500	1500	1479	1521	
10487	1997	6/21/1997	CLE	HOU	Cleveland Rockers	Houston Comets	0	0	56	76	1500	1500	1470	1530	

10488 rows × 16 columns

Dataset description

The dataset `wnba-team-elo-ratings.csv` contains Elo Ratings for every team in WNBA history on a game-by-game basis.

The ratings were developed by FiveThirtyEight's Jay Boice, similar to the basic ratings for the NBA.

The ratings change after every game based on the winner's pregame win probability, with more unexpected wins resulting in more points shifting from the loser's rating to the winner's.

Category	Decription	Type
season	Year of game	discrete numerical
date	Date of game	date type(categorical)
team1	First team listed's ID	Nominal categorical
team2	Second team listed's ID	Nominal categorical
name1	Team1's full name	Nominal categorical
name2	Team2's full name	Nominal categorical
neutral	Was game at a neutral site?	discrete numerical
playoff	Was game in playoffs? (1=yes,0=no)	discrete numerical
score1	Team1's points in game	discrete numerical
score2	Team2's points in game	discrete numerical
elo1_pre	Team1's pregame Elo rating	discrete numerical
elo2_pre	Team2's pregame Elo rating	discrete numerical
elo1_post	Team1's postgame Elo rating	discrete numerical
elo2_post	Team2's postgame Elo rating	discrete numerical
prob1	Team1's pregame odds of winning	Continuous numerical
is_home1	Was Team1 the home team?(1=yes,0=no)	discrete numerical

1.Perform EDA and choose any one automated feature selection method.

In [3]:
```
# printing first 5 rows
dataset.head(5)
```

Out[3]:

	season	date	team1	team2	name1	name2	neutral	playoff	score1	score2	elo1_pre	elo2_pre	elo1_post	elo2_post	prob1
0	2019	10/10/2019	WAS	CON	Washington Mystics	Connecticut Sun	0	1	89	78	1684	1634	1692	1627	0.718
1	2019	10/10/2019	CON	WAS	Connecticut Sun	Washington Mystics	0	1	78	89	1634	1684	1627	1692	0.282
2	2019	10/8/2019	WAS	CON	Washington Mystics	Connecticut Sun	0	1	86	90	1693	1626	1684	1634	0.476
3	2019	10/8/2019	CON	WAS	Connecticut Sun	Washington Mystics	0	1	90	86	1626	1693	1634	1684	0.524
4	2019	10/6/2019	WAS	CON	Washington Mystics	Connecticut Sun	0	1	94	81	1671	1648	1693	1626	0.399

In [4]:
```
# shape of the dataset
dataset.shape
```

Out[4]:
```
(10488, 16)
```

There are 10488 rows and 16 columns in the dataset.So the dataset is balanced.

In [5]:
```
# to know the type of variables
types=dataset.dtypes
types
```

Out[5]:
```
season         int64
date          object
team1         object
team2         object
name1         object
name2         object
neutral        int64
playoff        int64
score1         int64
score2         int64
elo1_pre       int64
elo2_pre       int64
elo1_post      int64
elo2_post      int64
prob1        float64
is_home1       int64
dtype: object
```

In [6]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10488 entries, 0 to 10487
Data columns (total 16 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   season     10488 non-null  int64  
 1   date       10488 non-null  object 
 2   team1      10488 non-null  object 
 3   team2      10488 non-null  object 
 4   name1      10488 non-null  object 
 5   name2      10488 non-null  object 
 6   neutral    10488 non-null  int64  
 7   playoff    10488 non-null  int64  
 8   score1     10488 non-null  int64  
 9   score2     10488 non-null  int64  
 10  elo1_pre   10488 non-null  int64  
 11  elo2_pre   10488 non-null  int64  
 12  elo1_post  10488 non-null  int64  
 13  elo2_post  10488 non-null  int64  
 14  prob1      10488 non-null  float64
 15  is_home1   10488 non-null  int64  
dtypes: float64(1), int64(10), object(5)
memory usage: 1.3+ MB
```

The datatypes of the variables are correct. All decimal varibales are in float and integer values are in integer type. String varibales are in object type.

In [7]:
```
# data description to understand the summary statistics of data
dataset.describe()
```

Out[7]:

	season	neutral	playoff	score1	score2	elo1_pre	elo2_pre	elo1_post	elo2_post	prob1
count	10488.000000	10488.0	10488.000000	10488.000000	10488.000000	10488.000000	10488.000000	10488.000000	10488.000000	10488.000000
mean	2008.094775	0.0	0.073227	74.171815	74.171815	1493.718440	1493.718440	1493.717868	1493.717868	0.500000
std	6.361955	0.0	0.260520	12.409629	12.409629	87.037547	87.037547	88.109043	88.109043	0.184767
min	1997.000000	0.0	0.000000	0.000000	0.000000	1183.000000	1183.000000	1168.000000	1168.000000	0.049000
25%	2003.000000	0.0	0.000000	66.000000	66.000000	1442.000000	1442.000000	1441.000000	1441.000000	0.358000
50%	2008.000000	0.0	0.000000	74.000000	74.000000	1498.000000	1498.000000	1497.000000	1497.000000	0.500000
75%	2014.000000	0.0	0.000000	82.000000	82.000000	1549.000000	1549.000000	1549.000000	1549.000000	0.642000
max	2019.000000	0.0	1.000000	127.000000	127.000000	1741.000000	1741.000000	1743.000000	1743.000000	0.951000

All the summary statistics are almost same for team1 and team2. The mean, median and std are almost same.

In [8]:
```
# setting the precisin to limit the decimals
pd.set_option('precision', 2)
dataset.describe()
```

Out[8]:

	season	neutral	playoff	score1	score2	elo1_pre	elo2_pre	elo1_post	elo2_post	prob1	is_home1
count	10488.00	10488.0	10488.00	10488.00	10488.00	10488.00	10488.00	10488.00	10488.00	10488.00	10488.0
mean	2008.09	0.0	0.07	74.17	74.17	1493.72	1493.72	1493.72	1493.72	0.50	0.5
std	6.36	0.0	0.26	12.41	12.41	87.04	87.04	88.11	88.11	0.18	0.5
min	1997.00	0.0	0.00	0.00	0.00	1183.00	1183.00	1168.00	1168.00	0.05	0.0
25%	2003.00	0.0	0.00	66.00	66.00	1442.00	1442.00	1441.00	1441.00	0.36	0.0
50%	2008.00	0.0	0.00	74.00	74.00	1498.00	1498.00	1497.00	1497.00	0.50	0.5
75%	2014.00	0.0	0.00	82.00	82.00	1549.00	1549.00	1549.00	1549.00	0.64	1.0
max	2019.00	0.0	1.00	127.00	127.00	1741.00	1741.00	1743.00	1743.00	0.95	1.0

In [9]:
```
# checking if there is any null values present
dataset.isnull().sum()
```

Out[9]:
```
season       0
date         0
team1        0
team2        0
name1        0
name2        0
neutral      0
playoff      0
score1       0
score2       0
elo1_pre     0
elo2_pre     0
elo1_post    0
elo2_post    0
prob1        0
is_home1     0
dtype: int64
```

There are no null values in the dataset.

In [10]:
```
# checking for duplicates
dataset.duplicated().sum()
```

Out[10]:
```
0
```

There are no duplicate values in the dataset

In [11]:
```
# correlation between variables
Correlations=dataset.corr()
Correlations
```

Out[11]:

	season	neutral	playoff	score1	score2	elo1_pre	elo2_pre	elo1_post	elo2_post	prob1	is_home1
season	1.00e+00	NaN	2.87e-02	0.37	0.37	1.55e-01	1.55e-01	1.53e-01	1.53e-01	-2.11e-15	-2.63e-15
neutral	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
playoff	2.87e-02	NaN	1.00e+00	0.02	0.02	2.53e-01	2.53e-01	2.50e-01	2.50e-01	-1.69e-17	-1.21e-16
score1	3.73e-01	NaN	1.57e-02	1.00	0.45	2.07e-01	-4.64e-02	2.76e-01	-1.18e-01	2.23e-01	1.31e-01
score2	3.73e-01	NaN	1.57e-02	0.45	1.00	-4.64e-02	2.07e-01	-1.18e-01	2.76e-01	-2.23e-01	-1.31e-01
elo1_pre	1.55e-01	NaN	2.53e-01	0.21	-0.05	1.00e+00	5.13e-02	9.87e-01	5.19e-02	5.63e-01	9.32e-03
elo2_pre	1.55e-01	NaN	2.53e-01	-0.05	0.21	5.13e-02	1.00e+00	5.19e-02	9.87e-01	-5.63e-01	-9.32e-03
elo1_post	1.53e-01	NaN	2.50e-01	0.28	-0.12	9.87e-01	5.19e-02	1.00e+00	2.57e-02	5.54e-01	8.89e-03
elo2_post	1.53e-01	NaN	2.50e-01	-0.12	0.28	5.19e-02	9.87e-01	2.57e-02	1.00e+00	-5.54e-01	-8.89e-03
prob1	-2.11e-15	NaN	-1.69e-17	0.22	-0.22	5.63e-01	-5.63e-01	5.54e-01	-5.54e-01	1.00e+00	5.77e-01
is_home1	-2.63e-15	NaN	-1.21e-16	0.13	-0.13	9.32e-03	-9.32e-03	8.89e-03	-8.89e-03	5.77e-01	1.00e+00

The neutral varaibale can be discarded since it is all zero values. There is negative correalation exists between the pregame Elo ratings and the score of the teams.

There is high positive correlation between the playoff and the scores.

The corerlation is in medium level between the scores ie. score1 and score2.

In [12]:
```
# identify the skewness of the data
dataset.skew()
```

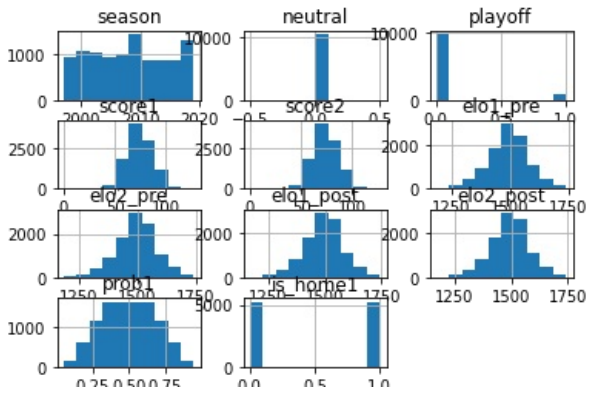
Out[12]:
```
season       0.07
neutral      0.00
playoff      3.28
score1       0.14
score2       0.14
elo1_pre    -0.24
elo2_pre    -0.24
elo1_post   -0.23
elo2_post   -0.23
prob1        0.00
is_home1     0.00
dtype: float64
```

There is negative skewness in the variables elo1_pre, elo2_pre, elo1_post and elo2_post. And high poitive skewness in the playoff variable.

Visualizations

In [13]:
```
from matplotlib import pyplot
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
```

In [14]:
```
histogram=dataset.hist()
pyplot.show()
```



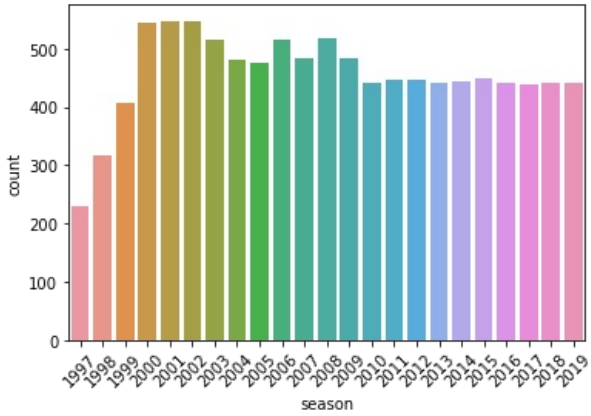
Histogram helps to identify the skewness and outliers in the data. It helps to identify the distribution is Gaussian or not.

Here the variables score1, score2, elo1_pre, elo2_pre, elo1_post, elo2_post and prob1 are following somewhat Gaussian distribution.

Univariate analysis

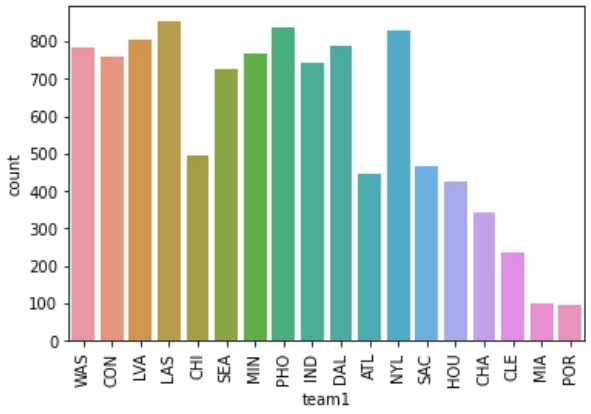
plotting each variable will help to understand the details more clearly.

In [15]:
```
# season
chart = sns.countplot(x='season', data=dataset)
chart.set_xticklabels(chart.get_xticklabels(), rotation=45)
plt.show()
```



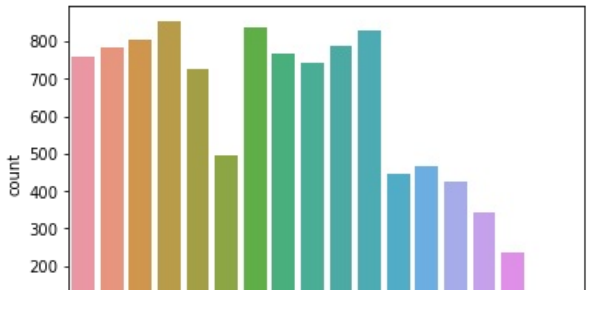
The years 2000, 2001 and 2002 has the most games.

In [16]:
```
# team1
chart = sns.countplot(x='team1', data=dataset)
chart.set_xticklabels(chart.get_xticklabels(), rotation=90)
plt.show()
```



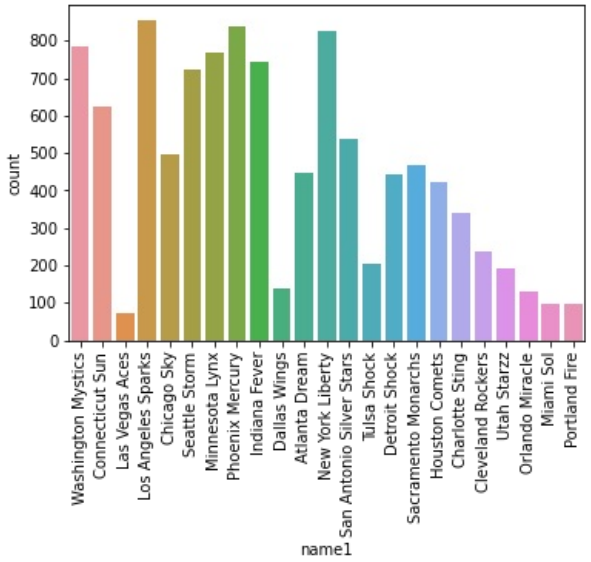
The team LAS has played the most games as team1 and MIA and POR played least games.

In [17]:
```
# team2
chart = sns.countplot(x='team2', data=dataset)
chart.set_xticklabels(chart.get_xticklabels(), rotation=90)
plt.show()
```

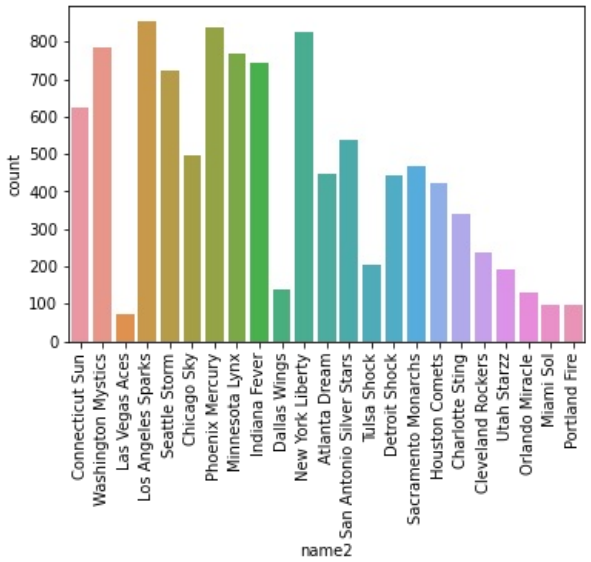


The team LAS has played the most games as team2 and MIA and POR played least games.

In [18]:
```
# name1
chart = sns.countplot(x='name1', data=dataset)
chart.set_xticklabels(chart.get_xticklabels(), rotation=90)
plt.show()
```

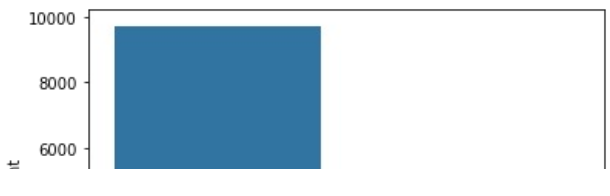


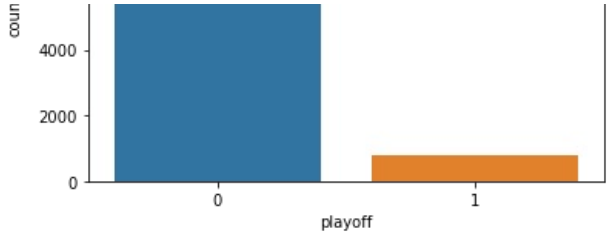
In [19]:
```
# name2
chart = sns.countplot(x='name2', data=dataset)
chart.set_xticklabels(chart.get_xticklabels(), rotation=90)
plt.show()
```



The team Los Angeles Sparks has played the most games and Las Vegas Aces played least games.

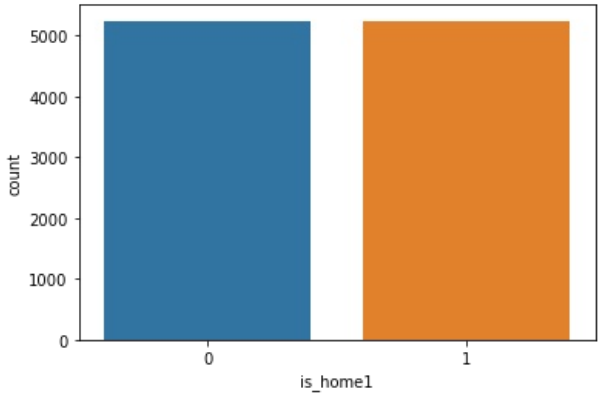
In [20]:
```
# playoff
chart = sns.countplot(x='playoff', data=dataset)
plt.show()
```





Most of the games were not in playoffs.

In [21]:
```
# is_home1
sns.countplot(x='is_home1', data=dataset)
plt.show()
```



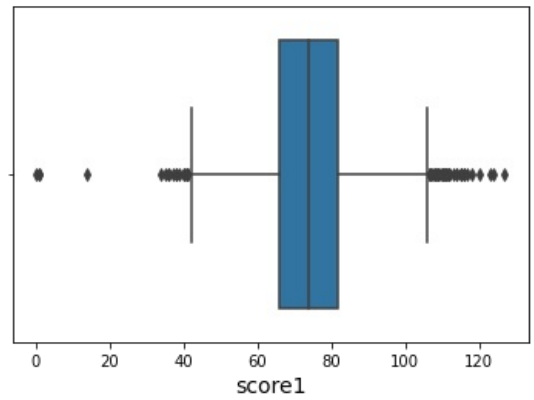
The team1 was equally home team and away team in games. It has equal number of games as away and home team.

Boxplots

Box plots can be used to visualize the continuous numerical variables. It gives the imformation about the outliers more efficiently. The interquartile range, variance, minimum and maximum value can be understood from the box plot.

In [22]:
```
# SCORE1
boxplot = sns.boxplot(x="score1", data =dataset)
boxplot.set_xlabel("score1", fontsize=14)
```

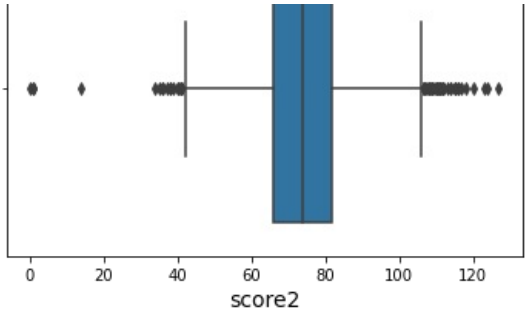
Out[22]:
```
Text(0.5, 0, 'score1')
```



It has outliers. The mean and median are somewhat equal and the whiskers are of almost equal length.

In [23]:
```
#score2
boxplot = sns.boxplot(x="score2", data =dataset)
boxplot.set_xlabel("score2", fontsize=14)
```

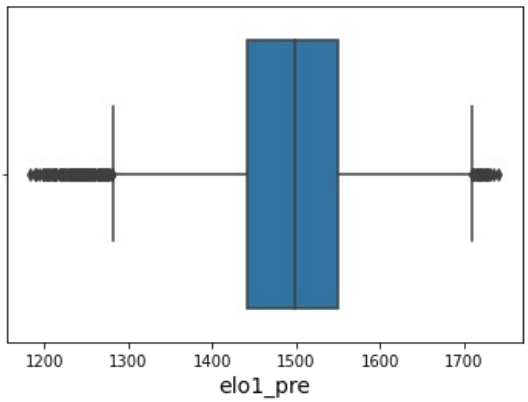
Out[23]:
```
Text(0.5, 0, 'score2')
```



It has outliers. The mean and median are somewhat equal and the whiskers are of almost equal length.

In [24]:
```
#elo1_pre
boxplot = sns.boxplot(x="elo1_pre", data =dataset)
boxplot.set_xlabel("elo1_pre", fontsize=14)
```

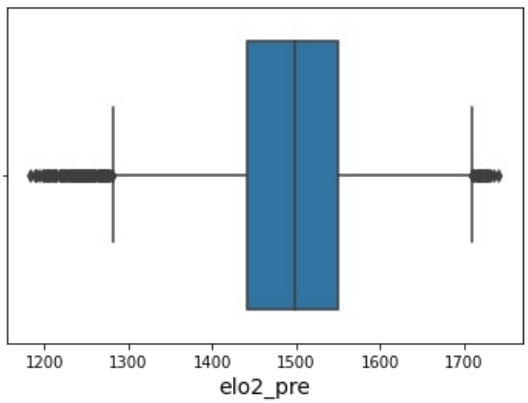
Out[24]:
```
Text(0.5, 0, 'elo1_pre')
```



It has outliers. The mean and median are somewhat equal and the whiskers are of almost equal length.

In [25]:
```
# elo2_pre
boxplot = sns.boxplot(x="elo2_pre", data =dataset)
boxplot.set_xlabel("elo2_pre", fontsize=14)
```

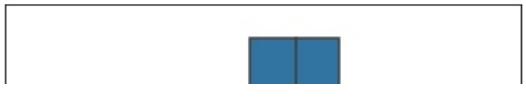
Out[25]:
```
Text(0.5, 0, 'elo2_pre')
```

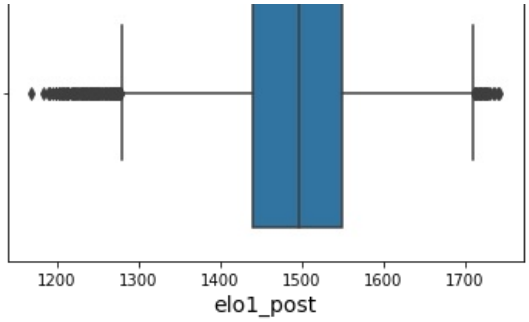


It has outliers. The mean and median are somewhat equal and the whiskers are of almost equal length.

In [26]:
```
#elo1_post
boxplot = sns.boxplot(x="elo1_post", data =dataset)
boxplot.set_xlabel("elo1_post", fontsize=14)
```

Out[26]:
```
Text(0.5, 0, 'elo1_post')
```

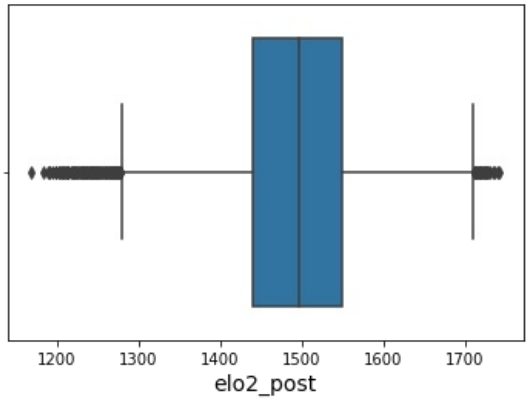




It has outliers. The mean and median are somewhat equal and the whiskers are of almost equal length.

In [27]:
```
#elo2_post
boxplot = sns.boxplot(x="elo2_post", data =dataset)
boxplot.set_xlabel("elo2_post", fontsize=14)
```

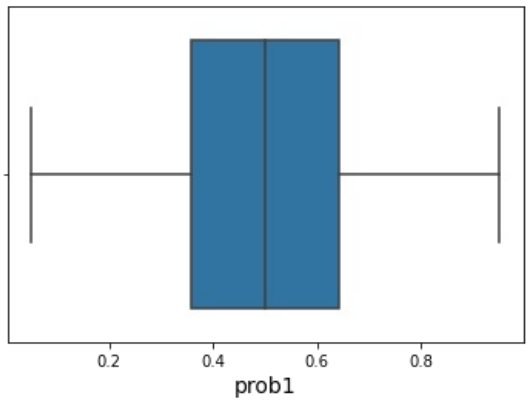
Out[27]:
```
Text(0.5, 0, 'elo2_post')
```



It has outliers. The mean and median are somewhat equal and the whiskers are of almost equal length.

In [28]:
```
# prob1
boxplot = sns.boxplot(x="prob1", data =dataset)
boxplot.set_xlabel("prob1", fontsize=14)
```

Out[28]:
```
Text(0.5, 0, 'prob1')
```



The probability variable is symmetrical with equally distributed whiskers. The mean and median is almost equal.

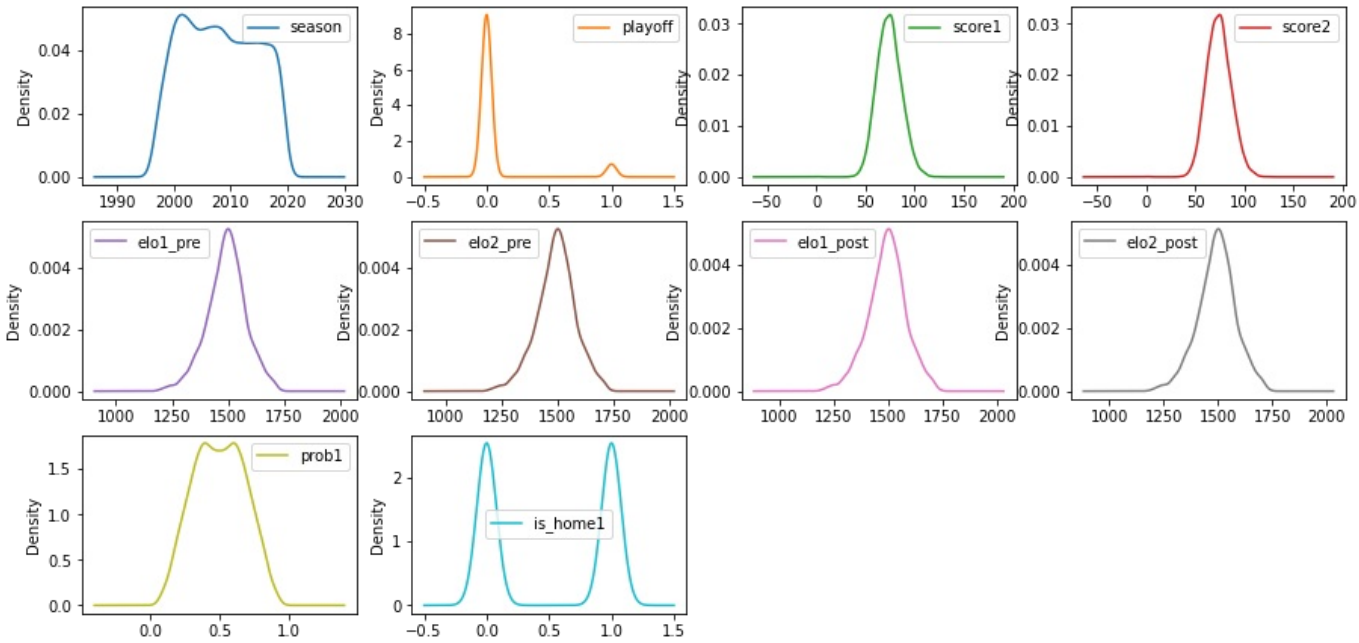
In [29]:
```
# printing dataset columns
dataset.columns
```

Out[29]:
```
Index(['season', 'date', 'team1', 'team2', 'name1', 'name2', 'neutral',
       'playoff', 'score1', 'score2', 'elo1_pre', 'elo2_pre', 'elo1_post',
       'elo2_post', 'prob1', 'is_home1'],
      dtype='object')
```

In [30]:
```
# dropping 'neutral' variable since it contains only zeros.
density = dataset.copy()
density.drop(columns=['neutral'],inplace=True)
```

density plot

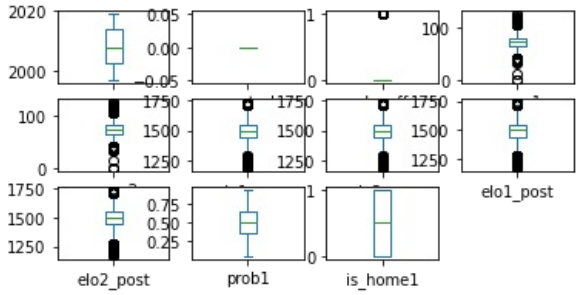
In [31]:
```
K= density.plot(kind='density',subplots=True,sharex=False,layout=(4,4),sharey=False, figsize=[15,10])
```



Density plot can be used to get an overall idea about the data to identify whether it follows gaussian distribution or not. The score1, score2, elo1_pre, elo2_pre, elo1_post and elo2_post seems to have gaussian distribution among the attributes.

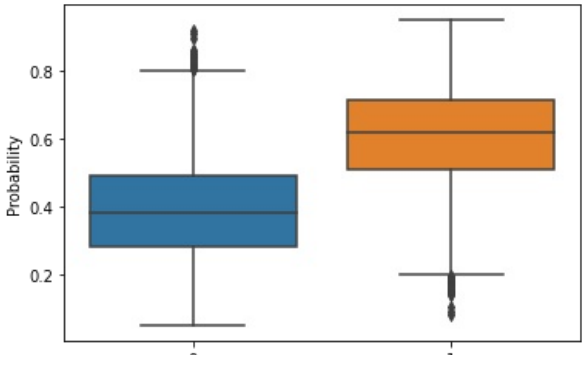
Box and whisker plots

In [32]:
```
boxplot = dataset.plot(kind='box',subplots=True, layout =(4,4), sharex= False, sharey= False)
```



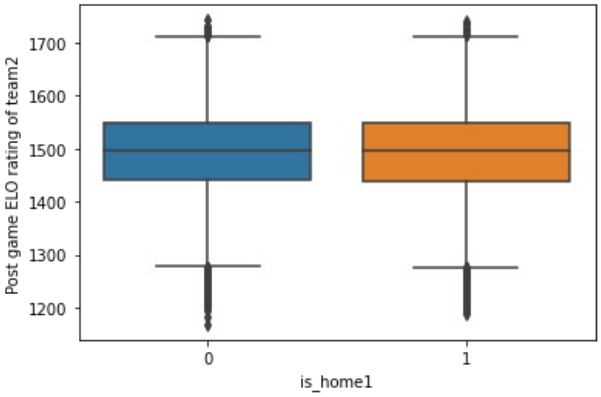
Bivariate plots

In [33]:
```
sns.boxplot(x="is_home1", y="prob1", data=dataset)
plt.ylabel("Probability")
plt.show()
```



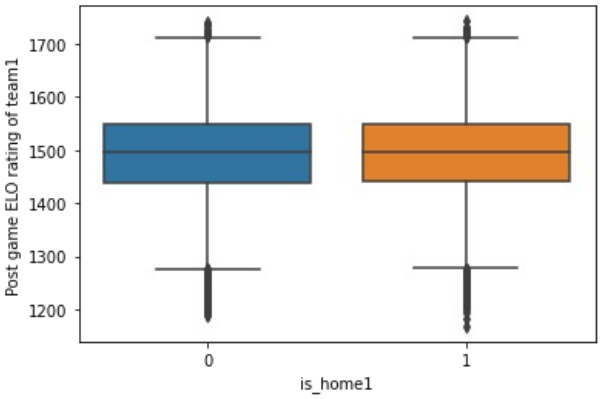
There are outliers present in both the groups. The IQR is not overlapped.

In [34]:
```
sns.boxplot(x="is_home1", y="elo2_post", data=dataset)
plt.ylabel("Post game ELO rating of team2")
plt.show()
```



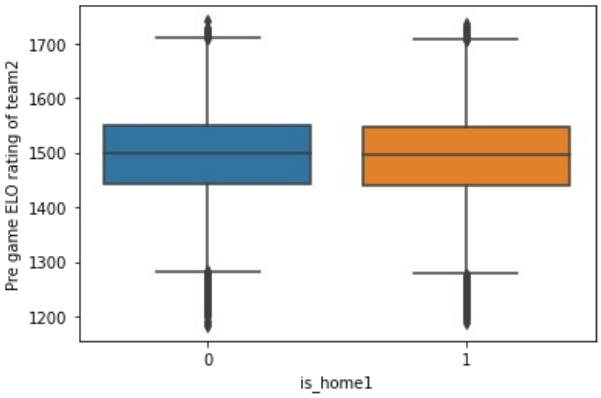
There are outliers present in both the groups. The IQR is overlapped. Both groups have equal variance.

In [35]:
```
sns.boxplot(x="is_home1", y="elo1_post", data=dataset)
plt.ylabel("Post game ELO rating of team1")
plt.show()
```



There are outliers present in both the groups. The IQR is overlapped. Both groups have equal variance.

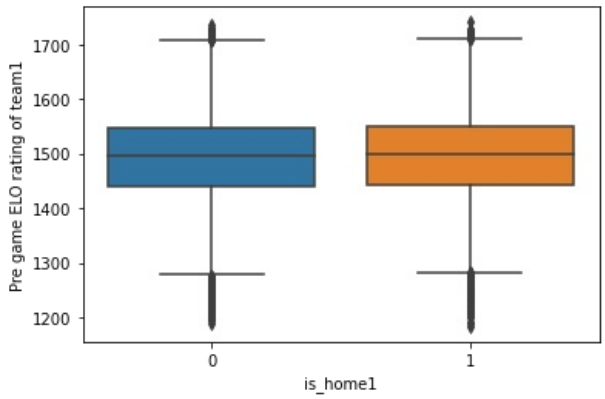
In [36]:
```
sns.boxplot(x="is_home1", y="elo2_pre", data=dataset)
plt.ylabel("Pre game ELO rating of team2")
plt.show()
```



There are outliers present in both the groups. The IQR is overlapped. Both groups have equal variance.

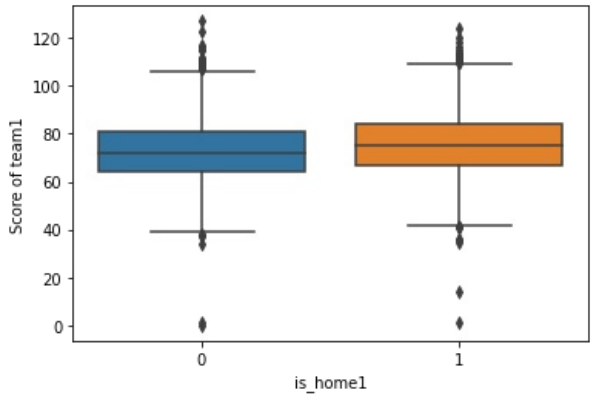
In [37]:
```
sns.boxplot(x="is_home1", y="elo1_pre", data=dataset)
```

```
plt.ylabel("Pre game ELO rating of team1")
plt.show()
```



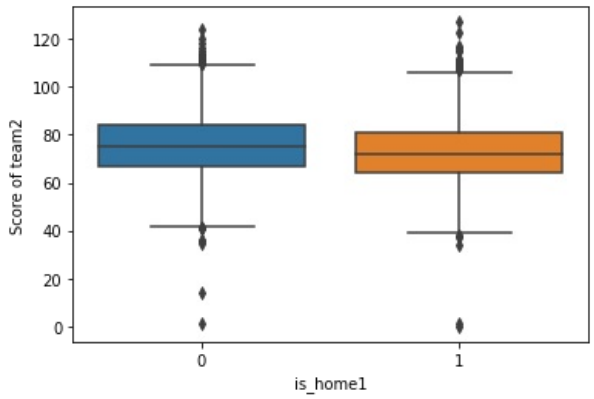
There are outliers present in both the groups. The IQR is overlapped. Both groups have equal variance.

In [38]:
```
sns.boxplot(x="is_home1", y="score1", data=dataset)
plt.ylabel("Score of team1")
plt.show()
```



There are outliers present in both the groups. The IQR is overlapped. Both groups have equal variance.

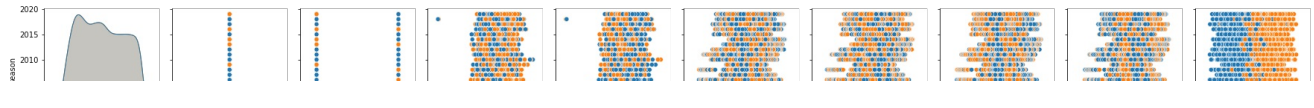
In [39]:
```
sns.boxplot(x="is_home1", y="score2", data=dataset)
plt.ylabel("Score of team2")
plt.show()
```

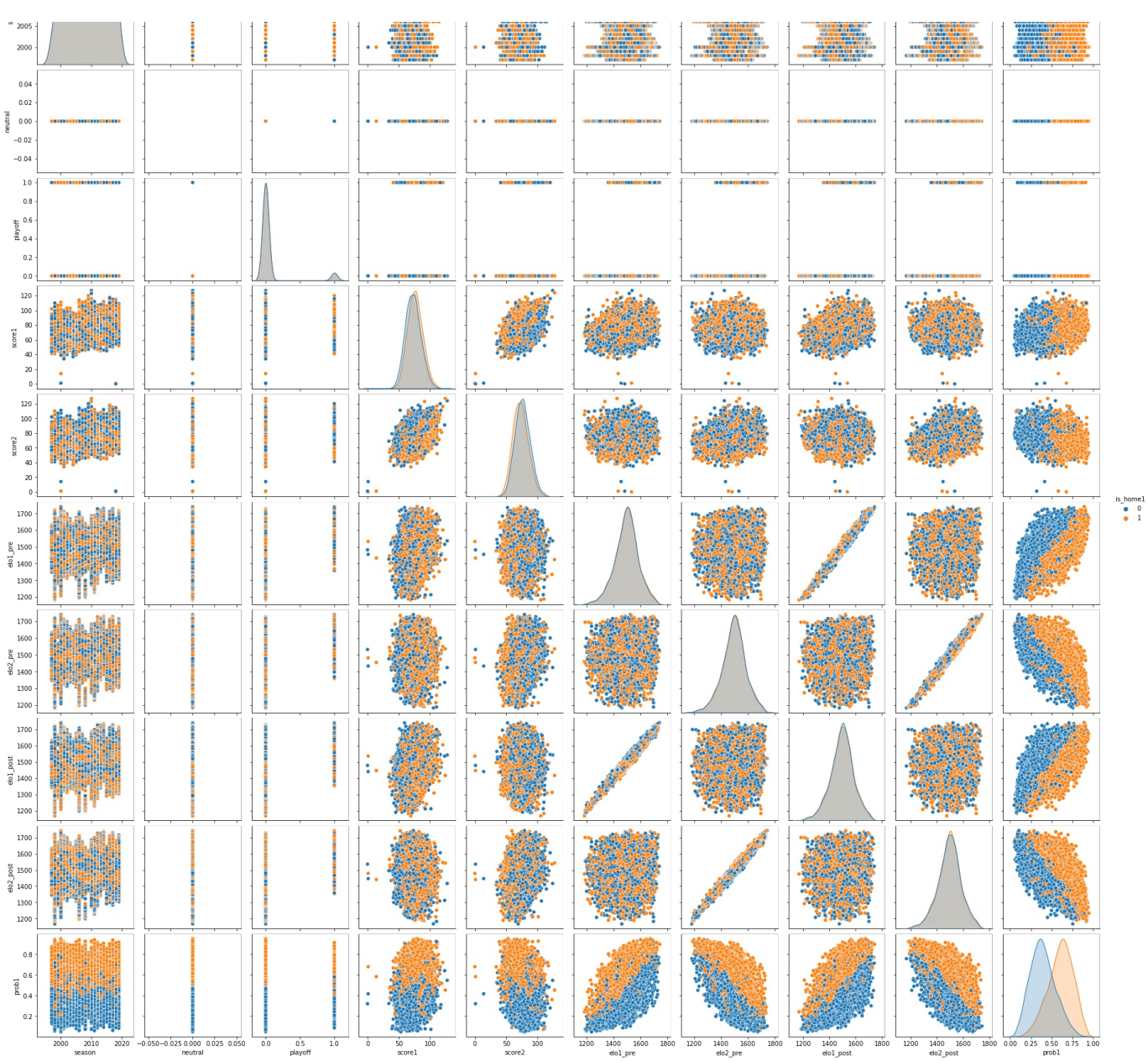


There are outliers present in both the groups. The IQR is overlapped.

Multivariate plots

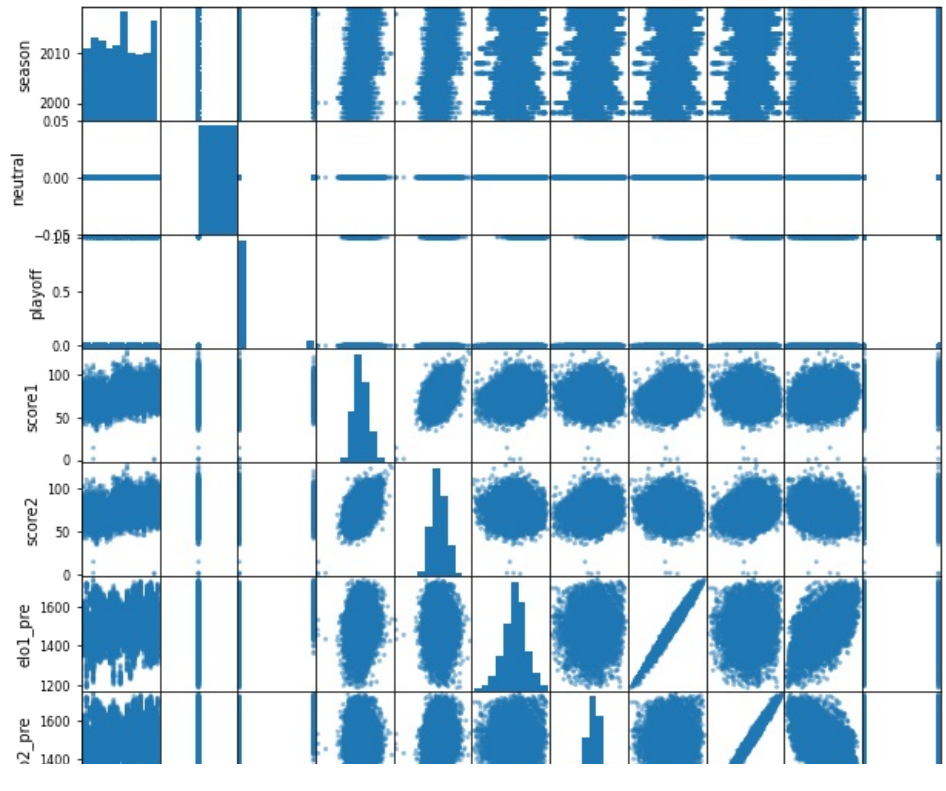
In [40]:
```
sns.pairplot(dataset,hue="is_home1")
plt.show()
```

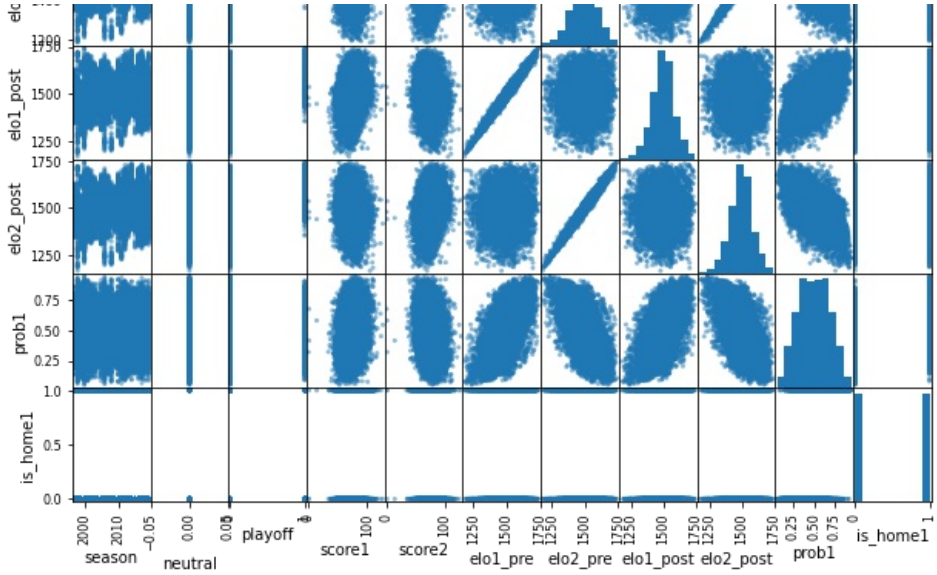




In [41]:

```
# scatter plots
from pandas.plotting import scatter_matrix
data = scatter_matrix(dataset,figsize=[10,15])
```





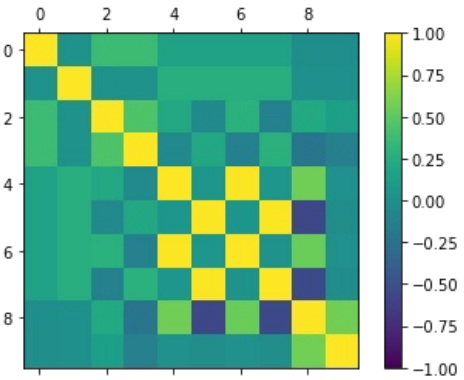
The scatter plot found to be not suitable to understand the relationship using this data. Most values are spread without any relation and are almost clustered.

Correlation Matrix Plot

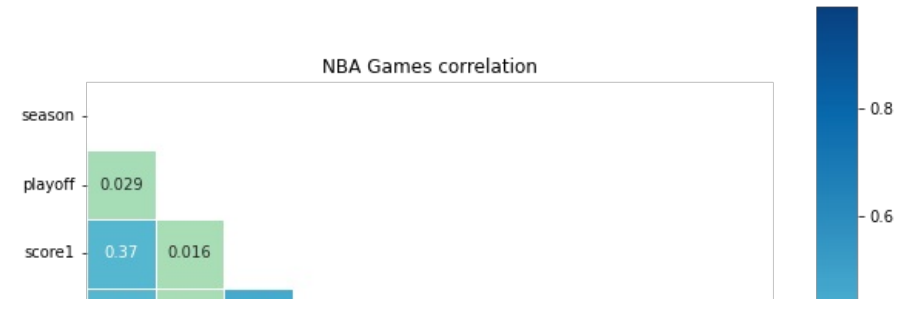
Correlation is an indication of how related the variables are. There is inversely and direct proportion in correlation.

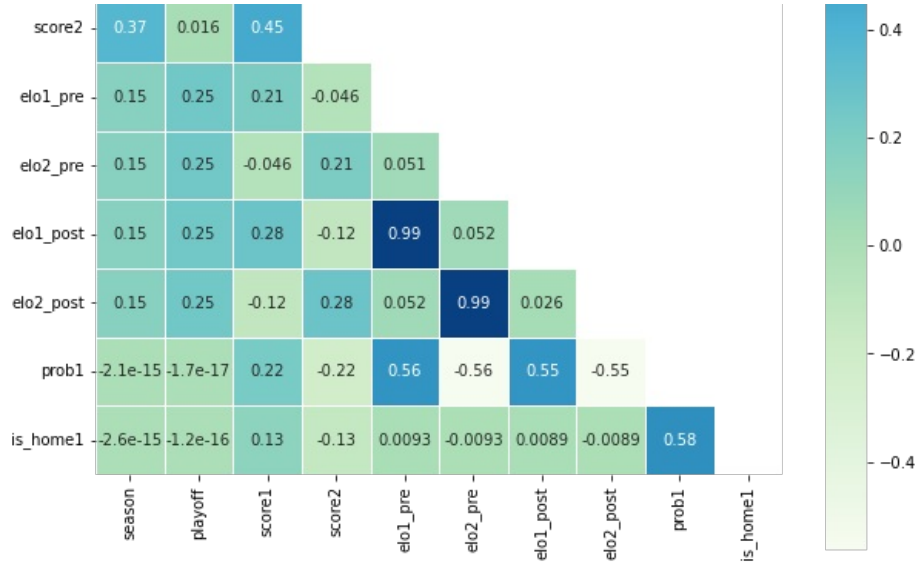
In [42]:
```
# removing variables that are not much important
corr1 = dataset.copy()
corr1.drop(columns=['neutral', 'date'],inplace=True)
```

In [43]:
```
#Correlation Matrix
correlations = corr1.corr()
fig = pyplot.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
pyplot.show()
```



In [44]:
```
plt.figure(figsize=(10,10))
mask=np.zeros_like(corr1.corr(),dtype=np.bool)
mask[np.triu_indices_from(mask)]=True
sns.heatmap(data=corr1.corr(),annot=True,square=True,mask=mask,cmap="GnBu",linewidths=1,linecolor="white")
plt.title("NBA Games correlation")
plt.show()
```





The ELO ratings of pre games and post games are having high correlation. The score and the ELO ratings are having low correlation with negative values.

The highly correlated values can be removed for better modelling.

Preparing the dataset for ML

Dropping neutral column since it is all zero values.And also removing date column.

In [45]:
```
data = dataset.copy()
data.drop(columns=['neutral', 'date'],inplace=True)
```

converting categorical columns to dummies for analysis

In [46]:
```
data = pd.get_dummies(data, columns=['team1', 'team2', 'name1', 'name2'])
```

In [47]:
```
data
```

Out[47]:

	season	playoff	score1	score2	elo1_pre	elo2_pre	elo1_post	elo2_post	prob1	is_home1	...	name2_New York Liberty	name2_Orlando Miracle	name2_Ph M
0	2019	1	89	78	1684	1634	1692	1627	0.72	1	...	0	0	
1	2019	1	78	89	1634	1684	1627	1692	0.28	0	...	0	0	
2	2019	1	86	90	1693	1626	1684	1634	0.48	0	...	0	0	
3	2019	1	90	86	1626	1693	1634	1684	0.52	1	...	0	0	
4	2019	1	94	81	1671	1648	1693	1626	0.40	0	...	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
10483	1997	0	73	61	1500	1500	1521	1479	0.39	0	...	0	0	
10484	1997	0	67	57	1500	1500	1519	1481	0.39	0	...	0	0	
10485	1997	0	57	67	1500	1500	1481	1519	0.61	1	...	1	0	
10486	1997	0	61	73	1500	1500	1479	1521	0.61	1	...	0	0	
10487	1997	0	56	76	1500	1500	1470	1530	0.61	1	...	0	0	

10488 rows × 92 columns

automated feature selection method.

In [48]:
```
# Feature Extraction with PCA
import numpy
from sklearn.decomposition import PCA
# 'is_home1' is selected as the response variable for modelling and analysis.

X= data.copy()
X.drop(columns=['is_home1'], inplace=True)
```

```
X = X.values
Y= data['is_home1']
Y=Y.values
# feature extraction with 2 components
pca2 = PCA(n_components=2)
fit = pca2.fit(X)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_)
print(fit.components_)
```

```
Explained Variance: [0.5134581 0.4693092]
[[-1.08192066e-02 -7.20533886e-04 -1.10469572e-02 -1.10469572e-02
  -4.99924068e-01 -4.99924068e-01 -4.99894538e-01 -4.99894538e-01
  -1.45075456e-18  3.09609347e-06  1.22691630e-04  7.02053212e-05
   6.72089609e-05 -6.11796572e-05  1.54800853e-04 -1.49263556e-04
  -6.67724638e-06 -2.96051316e-04  1.53377357e-04  7.19367950e-05
  -1.71885096e-04  4.23701526e-05 -1.12686396e-04  7.34054230e-05
  -6.96480470e-05 -5.61056650e-05  1.64404394e-04  3.09609347e-06
   1.22691630e-04  7.02053212e-05  6.72089609e-05 -6.11796572e-05
   1.54800853e-04 -1.49263556e-04 -6.67724638e-06 -2.96051316e-04
   1.53377357e-04  7.19367950e-05 -1.71885096e-04  4.23701526e-05
  -1.12686396e-04  7.34054230e-05 -6.96480470e-05 -5.61056650e-05
   1.64404394e-04  3.09609347e-06  1.22691630e-04  7.02053212e-05
   6.72089609e-05 -1.51961842e-04  1.94571539e-05  4.25912555e-05
  -1.49263556e-04 -6.67724638e-06 -7.44350032e-06 -2.96051316e-04
   7.19367950e-05 -1.71885096e-04  4.23701526e-05  9.07821844e-05
  -1.12686396e-04  7.34054230e-05 -6.96480470e-05  9.46292339e-05
  -5.61056650e-05  9.27524436e-05  6.61916237e-05  1.64404394e-04
   3.09609347e-06  1.22691630e-04  7.02053212e-05  6.72089609e-05
  -1.51961842e-04  1.94571539e-05  4.25912555e-05 -1.49263556e-04
  -6.67724638e-06 -7.44350032e-06 -2.96051316e-04  7.19367950e-05
  -1.71885096e-04  4.23701526e-05  9.07821844e-05 -1.12686396e-04
   7.34054230e-05 -6.96480470e-05  9.46292339e-05 -5.61056650e-05
   9.27524436e-05  6.61916237e-05  1.64404394e-04]
 [ 1.11022302e-16  1.11022302e-16  2.43968143e-02 -2.43968143e-02
   4.92974184e-01 -4.92974184e-01  5.06339627e-01 -5.06339627e-01
   1.24126424e-03 -7.74410557e-05 -5.30129811e-05 -1.39078668e-04
   2.81594955e-05  9.78714049e-05 -1.67190132e-04  2.07842057e-04
  -1.10927830e-05  3.03732505e-04 -2.44680124e-04 -2.53828746e-05
   1.49586867e-04 -1.71076564e-05  8.62583400e-05 -5.27775367e-05
   7.31581003e-05  3.29744461e-05 -1.91819404e-04  7.74410557e-05
   5.30129811e-05  1.39078668e-04 -2.81594955e-05 -9.78714049e-05
   1.67190132e-04 -2.07842057e-04  1.10927830e-05 -3.03732505e-04
   2.44680124e-04  2.53828746e-05 -1.49586867e-04  1.71076564e-05
  -8.62583400e-05  5.27775367e-05 -7.31581003e-05 -3.29744461e-05
   1.91819404e-04 -7.74410557e-05 -5.30129811e-05 -1.39078668e-04
   2.81594955e-05  1.29486776e-04 -4.67451980e-05  2.73766119e-05
   2.07842057e-04 -1.10927830e-05 -7.71424938e-06  3.03732505e-04
  -2.53828746e-05  1.49586867e-04 -1.71076564e-05 -3.16153716e-05
   8.62583400e-05 -5.27775367e-05  7.31581003e-05 -1.90275362e-04
   3.29744461e-05 -1.47821546e-04 -4.66905126e-05 -1.91819404e-04
   7.74410557e-05  5.30129811e-05  1.39078668e-04 -2.81594955e-05
  -1.29486776e-04  4.67451980e-05 -2.73766119e-05 -2.07842057e-04
   1.10927830e-05  7.71424938e-06 -3.03732505e-04  2.53828746e-05
  -1.49586867e-04  1.71076564e-05  3.16153716e-05 -8.62583400e-05
   5.27775367e-05 -7.31581003e-05  1.90275362e-04 -3.29744461e-05
   1.47821546e-04  4.66905126e-05  1.91819404e-04]]
```

In [49]:
```
fit_2 = pca2.fit_transform(X)
```

In [50]:
```
pca2.explained_variance_ratio_
```

Out[50]:
```
array([0.5134581, 0.4693092])
```

51 % of the variability in the dataset is accounted by the first principal component. 46% of the variability in the dataset is accounted by the second principal component.

In [51]:
```
pca4 = PCA(n_components=4)
fit = pca4.fit_transform(X)
pca4.explained_variance_ratio_
```

Out[51]:
```
array([0.5134581 , 0.4693092 , 0.00843954, 0.0073353 ])
```

Only .8 and 0.7% of the variability in the dataset is accounted by the third and fourth principal component. So it is better to select only two principal components.

2. Perform meaningful observations through visualisation on the obtained k-means and KNN

3.Clusters graph to visualise and use elbow method to select number of centroids.

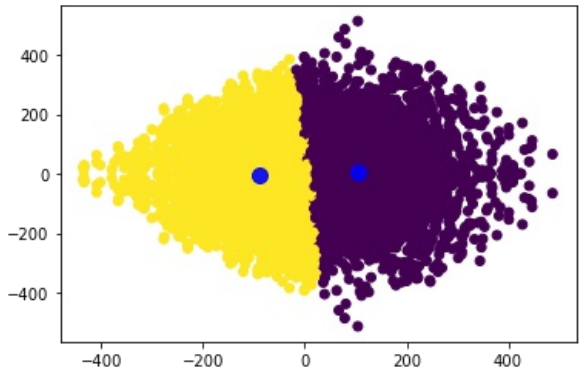
4. Visualise and use elbow method to select number of K (KNN) clustering

In [53]:
```
## kmeans clustering with 2
from sklearn.cluster import KMeans
Kmean = KMeans(n_clusters=2)
Kmean.fit(fit_2)
```

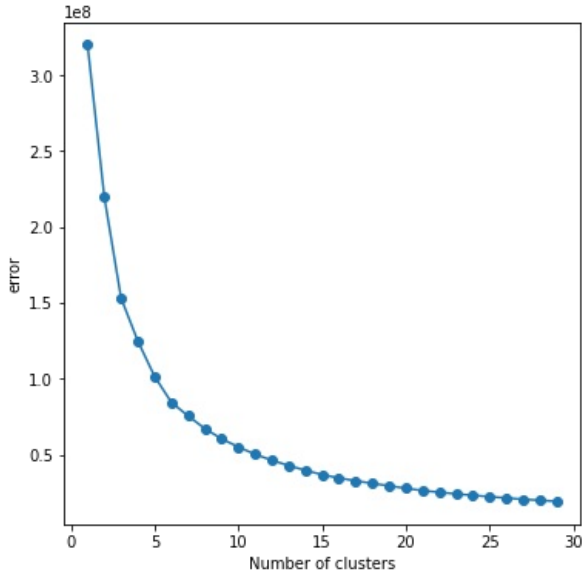
Out[53]:
```
KMeans(n_clusters=2)
```

In [54]:
```
Kmeans = Kmean.predict(fit_2)
```

In [55]:
```
# plotting centroids with 2 clusters
plt.scatter(fit_2[:, 0], fit_2[:, 1], c=Kmeans)
centers = Kmean.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='blue', s=100, alpha=0.9);
plt.show()
```

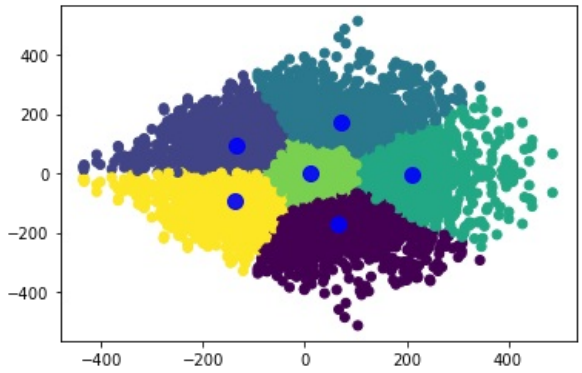


In [56]:
```
# plotting elbow graph
vals=[]
list_k= list(range(1,30))

for k in list_k:
    km = KMeans(n_clusters=k)
    km.fit(fit_2)
    vals.append(km.inertia_)

# Plot sse against k
plt.figure(figsize=(6, 6))
plt.plot(list_k, vals, '-o')
plt.xlabel('Number of clusters')
plt.ylabel('error')
plt.show()
```



From the elbow graph it is starting to show the curve after 6 clusters.

In [57]:
```
## kmeans clustering with 6 clusters from elbow graph
from sklearn.cluster import KMeans
Kmean = KMeans(n_clusters=6)
Kmean.fit(fit_2)
Kmeans = Kmean.predict(fit_2)

## plotting 6 centroids
plt.scatter(fit_2[:, 0], fit_2[:, 1], c=Kmeans)
centers = Kmean.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='blue', s=100, alpha=0.9);
plt.show()
```



In [58]:
```
# saving dataset to csv file to be used in different notebook for KNN
data.to_csv("data.csv", index=False)
```

In [ ]:
```

```