- Data cleaning
  - Are there missing values?

There are a lot of not available value in attributes. So I use the function data.isnull() to identify them and replace them by the average value of its column with the function data.PTS.fillna(data.PTS.mean(),inplace=True) .

In [4]:
```
data.isnull()
```

Out[4]:

	Unnamed: 0	Year	Player	Pos	Age	Tm	G	GS	MP	PER	...	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS
0	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False
8	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False
9	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False
10	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False

In [13]:
```
#Replace NA with the average value of the column
```

14	False	False	False	False	False	False	False	True	False	False	...	False	False	False	False	False	False	False	False	False	False

In [13]:
```
#Replace NA with the average value of the column
data.GS.fillna(data.GS.mean(),inplace=True)
data.PTS.fillna(data.PTS.mean(),inplace=True)
data.PER.fillna(data.PER.mean(),inplace=True)
data['FT%'].fillna(data['FT%'].mean(),inplace=True)
data['2P%'].fillna(data['2P%'].mean(),inplace=True)
data['3P%'].fillna(data['3P%'].mean(),inplace=True)
data['TS%'].fillna(data['TS%'].mean(),inplace=True)
```

In [14]:
```
data.isnull()
```

Out[14]:

	Unnamed: 0	Year	Player	Pos	Age	Tm	G	GS	MP	PER	...	FT%	ORB	DRB	TRB	AST	STL	BLK	TOV	PF	PTS
0	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False	False	False	False	False

- Are there inappropriate values?

By using the function data.describe(), we could see some inappropriate values, such as FT% value exceeds 100% and PTS value is less than 0.

```
In [18]: data.describe()
```

Out[18]:

	Unnamed: 0	Year	Age	G	GS	MP	PER	TS%	3PAr	FTr	...	
count	18964.000000	18927.000000	18927.000000	18927.000000	18964.000000	18927.000000	18922.000000	18851.000000	18839.000000	18839.000000	...	18964
mean	15208.500000	2000.272415	26.838326	49.639510	23.593375	1162.004649	12.395714	0.503862	0.158604	0.319590	...	0
std	5474.579588	10.691977	3.999546	26.693379	28.075092	924.026516	6.200326	0.094507	0.187495	0.230499	...	0
min	5727.000000	1980.000000	18.000000	1.000000	0.000000	0.000000	-90.600000	0.000000	0.000000	0.000000	...	0
25%	10467.750000	1992.000000	24.000000	26.000000	0.000000	313.000000	9.700000	0.473000	0.005000	0.197000	...	0
50%	15208.500000	2001.000000	26.000000	55.000000	9.000000	985.000000	12.700000	0.516000	0.064000	0.286000	...	0
75%	19949.250000	2010.000000	30.000000	75.000000	43.000000	1894.000000	15.600000	0.551000	0.288000	0.395000	...	0
max	24690.000000	2017.000000	44.000000	85.000000	83.000000	3533.000000	129.100000	1.136000	1.000000	6.000000	...	1

8 rows × 48 columns

- Remove or impute any bad data.

I remove these bad data by using function data['PER']=data['PER'].replace(data['PER'][data['PER']<0], np.nan). First, I replace bad data by null, then repeat the formal function and replace null values by the average value of its own column.

```
In [26]: #Replace all data which are less than 0 with null.
         #Replace all inappropriate data
         #The hit rate cannot exceed 100% or be less than 0
         data['PER']=data['PER'].replace(data['PER'][data['PER']<0], np.nan)
         data['2P%']=data['2P%'].replace(data['2P%'][data['2P%']>1], np.nan)
         data['TS%']=data['TS%'].replace(data['TS%'][data['TS%']>1], np.nan)
         data['3P%']=data['3P%'].replace(data['3P%'][data['3P%']>1], np.nan)
         data['2P%']=data['2P%'].replace(data['2P%'][data['2P%']<0], np.nan)
         data['TS%']=data['TS%'].replace(data['TS%'][data['TS%']<0], np.nan)
         data['3P%']=data['3P%'].replace(data['3P%'][data['3P%']<0], np.nan)
```

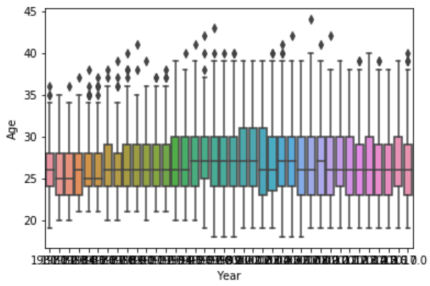
```
In [29]: #Replace new NA values with the average value of the column
         data.PER.fillna(data.PER.mean(),inplace=True)
         data['FT%'].fillna(data['FT%'].mean(),inplace=True)
         data['2P%'].fillna(data['2P%'].mean(),inplace=True)
         data['3P%'].fillna(data['3P%'].mean(),inplace=True)
```

```
In [30]: #Deal with TS data through the same method.
         data['TS%']=data['TS%'].replace(data['TS%'][data['TS%']>1], data['TS%'][data['TS%']].mean())
```

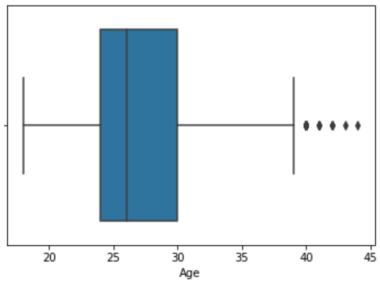
- Answer the following questions for the data in each column:
    - How is the data distributed?

By plotting box plot, displot or stripped plot and analyzing these plots and summary statistics, we could identify each column conforms to which type of distribution.

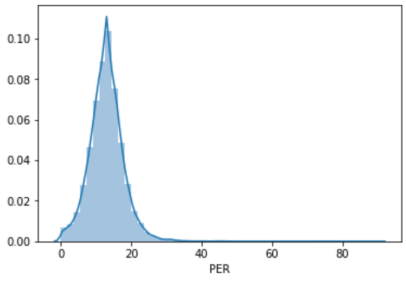
```
In [34]: sns.boxplot(x="Year", y="Age", data=data);
```




```
In [35]: #So the Age data conforms to Normal Distribution
         sns.boxplot(x=data["Age"])
Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x1a11816860>
```




```
In [38]: #PER
         #So the PER data conforms to Normal Distribution
         x = data['PER']
         sns.distplot(x)
Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1a6c21d0>
```




```
In [42]: #PTS
```

- What are the summary statistics?

I use the function data['PTS'].value_counts(), data.describe(), data.info() and others to see summary statistics of each column.

In [32]:
```
#Because there are many similar data, such as AST, STL, BLK, all they are some kinds of stroke analysis. So these kinds
#So I just choose several typical columns and analyze them.
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18964 entries, 0 to 18963
Data columns (total 51 columns):
Unnamed: 0    18964 non-null int64
Year          18927 non-null float64
Player        18927 non-null object
Pos           18927 non-null object
Age           18927 non-null float64
Tm            18927 non-null object
G             18927 non-null float64
GS            18964 non-null float64
MP            18927 non-null float64
PER           18964 non-null float64
TS%           18964 non-null float64
3PAr          18839 non-null float64
FTr           18839 non-null float64
ORB%          18922 non-null float64
DRB%          18922 non-null float64
TRB%          18922 non-null float64
AST%          18922 non-null float64
STL%          18922 non-null float64
BLK%          18922 non-null float64
TOV%          18866 non-null float64
USG%          18922 non-null float64
OWS           18927 non-null float64
DWS           18927 non-null float64
WS            18927 non-null float64
WS/48         18922 non-null float64
OBPM          18927 non-null float64
DBPM          18927 non-null float64
BPM           18927 non-null float64
VORP          18927 non-null float64
```

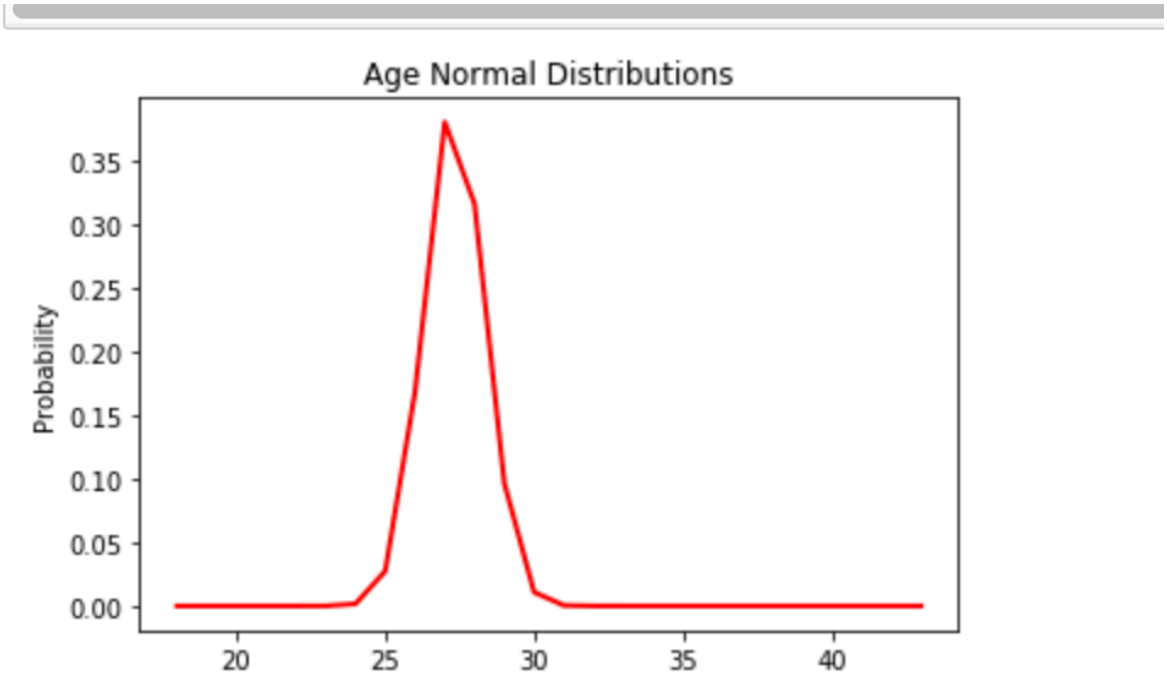
In [33]:
```
#Age
data['Age'].value_counts()
```

Out[33]:
```
24.0    2076
23.0    2001
25.0    1849
26.0    1778
27.0    1585
28.0    1371
22.0    1354
29.0    1209
30.0    1136
31.0     957
32.0     788
33.0     606
21.0     532
34.0     473
35.0     318
20.0     277
36.0     199
37.0     137
19.0     107
38.0      90
39.0      47
40.0      16
18.0      12
41.0       4
42.0       3
44.0       1
43.0       1
Name: Age, dtype: int64
```

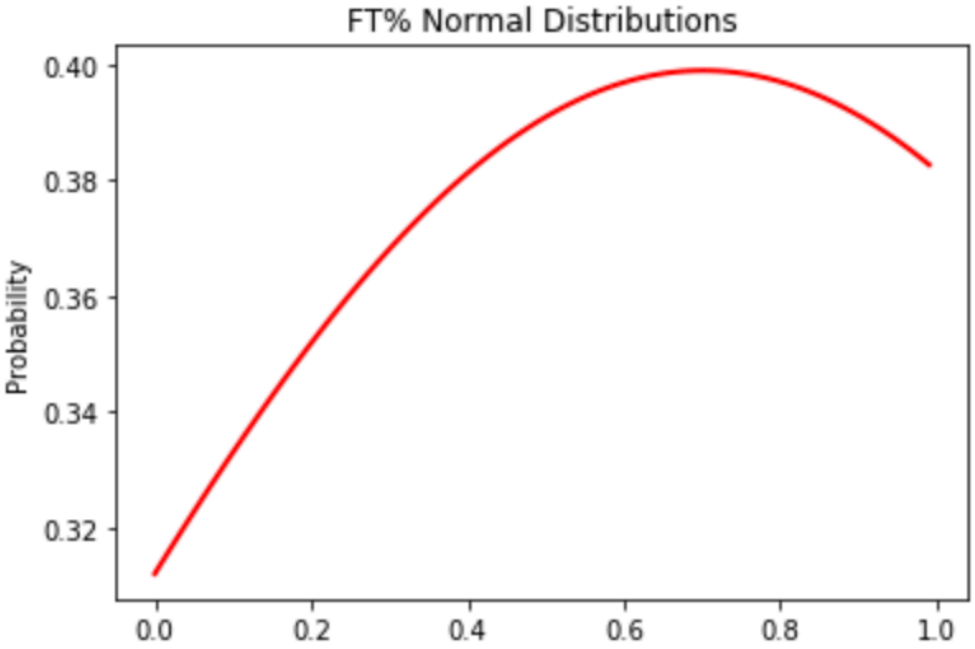
- Are there anomalies/outliers?

Through the summary statistics and plots ,we could find anomalies and outliers.

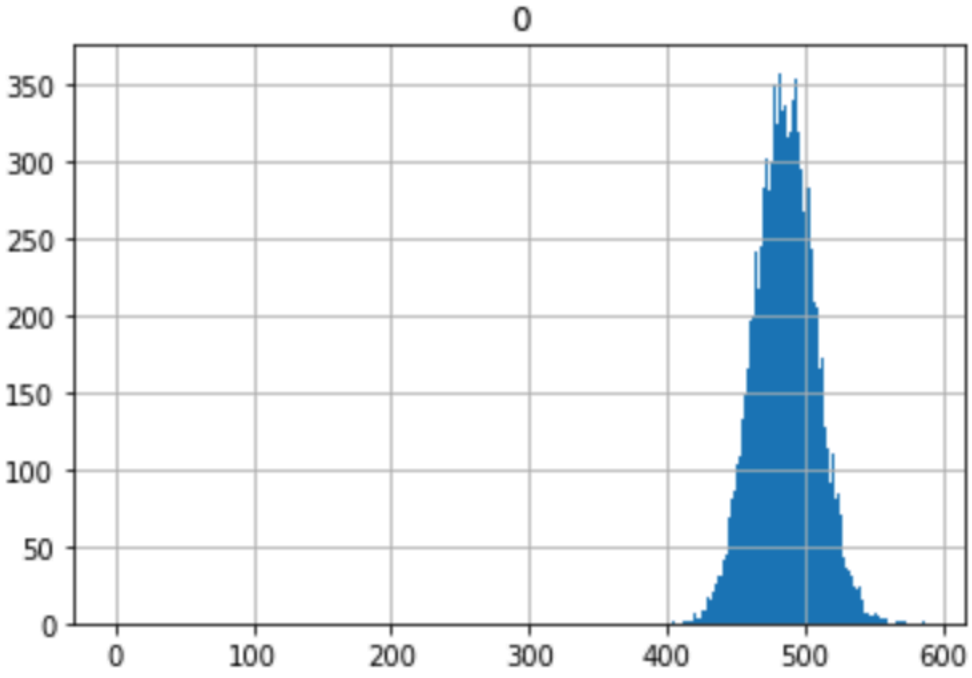
- Plot each colmun as appropriate for the data type:
  - Write a summary of what the plot tells you.



This is the Age normal distribution plot. Through the summary statistics we know that the min Age is 18 and the max is 44.
Through the plot I could know that the main group of ages are between 25 and 30. With age decreasing on the left of 25, the possibility decreases slower.
On the other hand, with age increasing on the right of 30, the possibility increases slower.



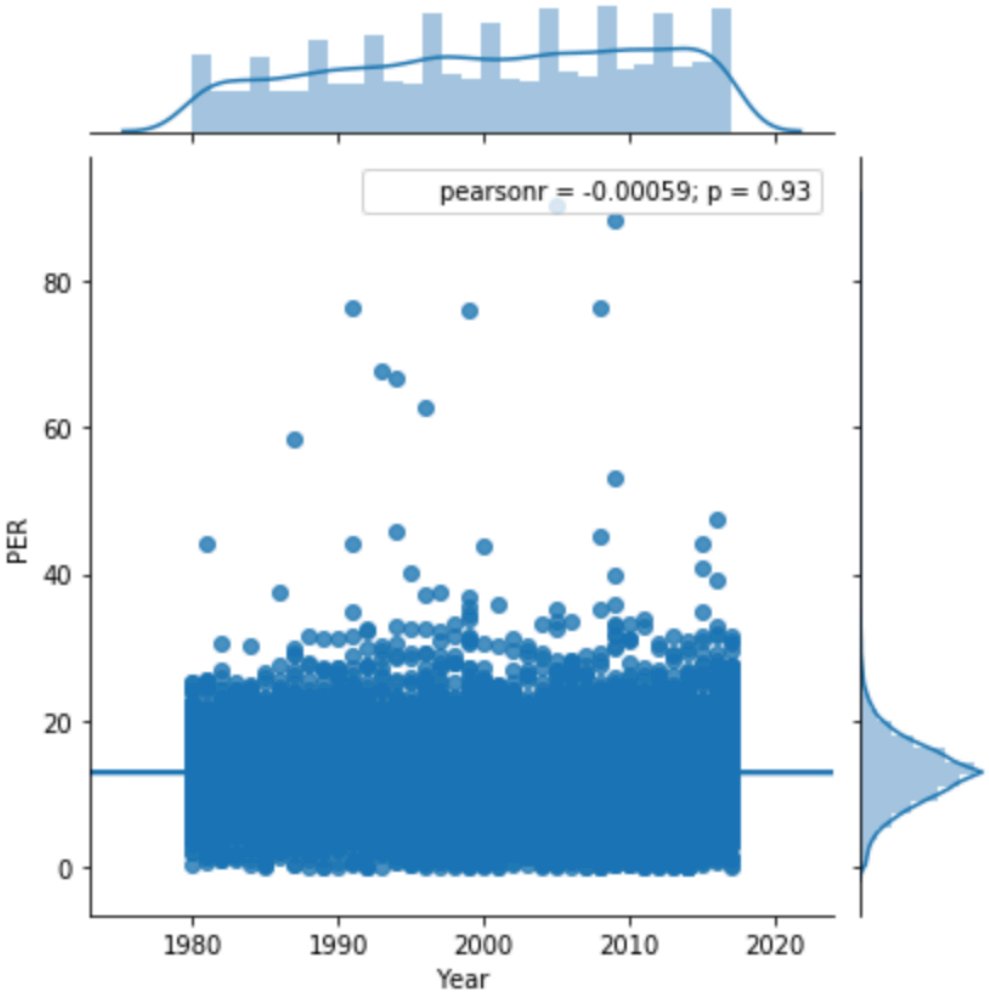
This is the FT% normal distribution plot. Through the summary statistics we know that the min FT% is 0 and the max one is 1.
Through the plot I could know that the main group of FT% are between 0.2 and 1.
Because of the particularity of FT% data, there is no possibility that FT% exceeds 1.0 or be less than 0. So this plot could only show a part of the normal distribution.



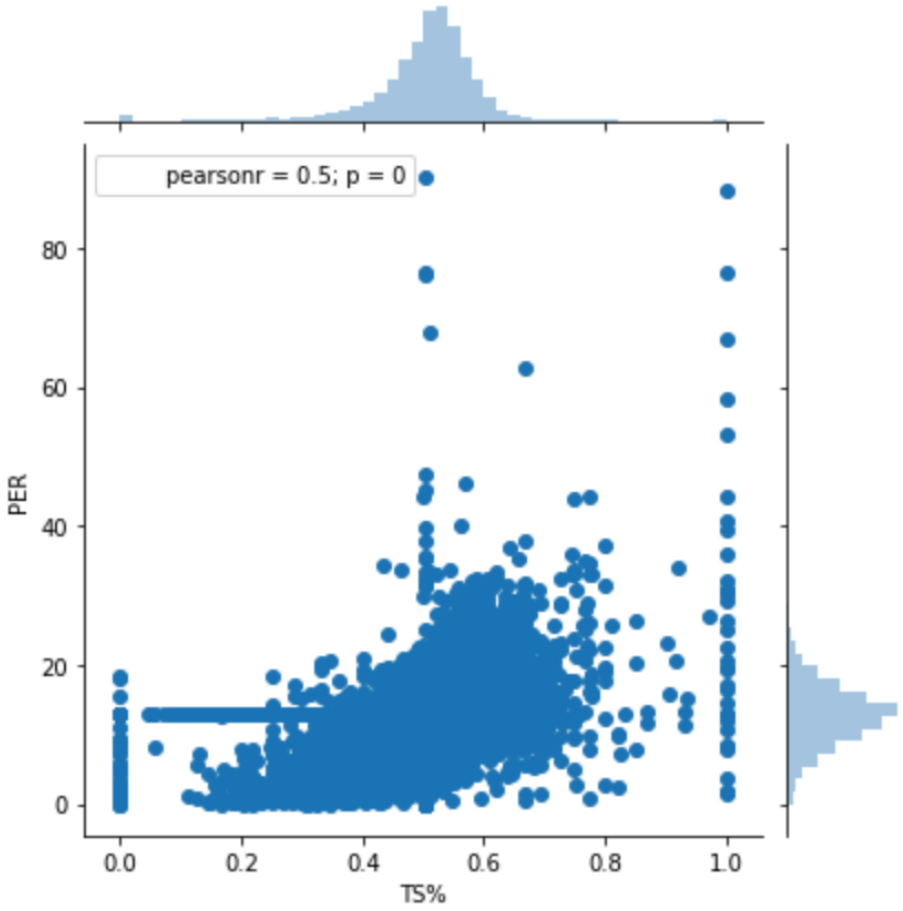
From the formal statistics we know that the count of PTS is 18964
The span of PTS data is very large.
Through the plot I could know that the main group of PTS are between 400 and 600. With PTS decreasing on the left of average point, the possibility decreases slower.
On the other hand, with PTS increasing on the right of average point, the possibility increases slower. Because I think all my plots are about normal distribution and it's similar to normal distribution, I only want to change a kind of plot here.

- Are any of the columns correlated?

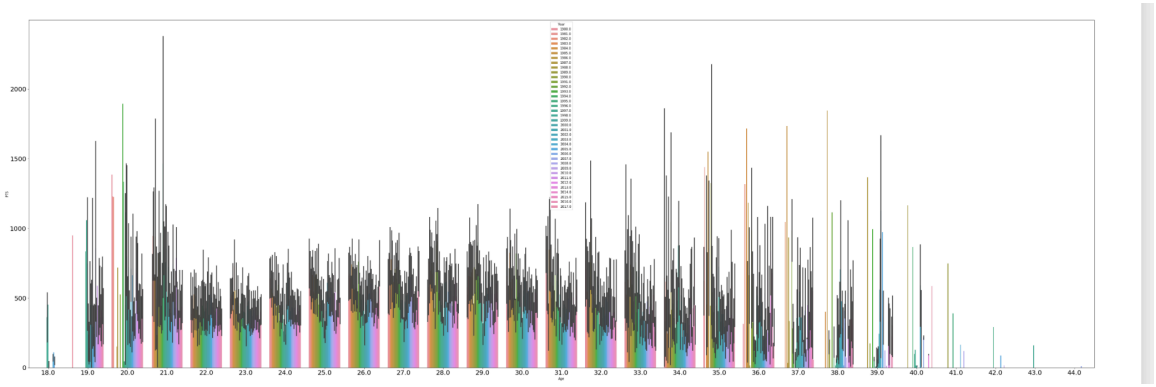
There are many columns which correlate with others.

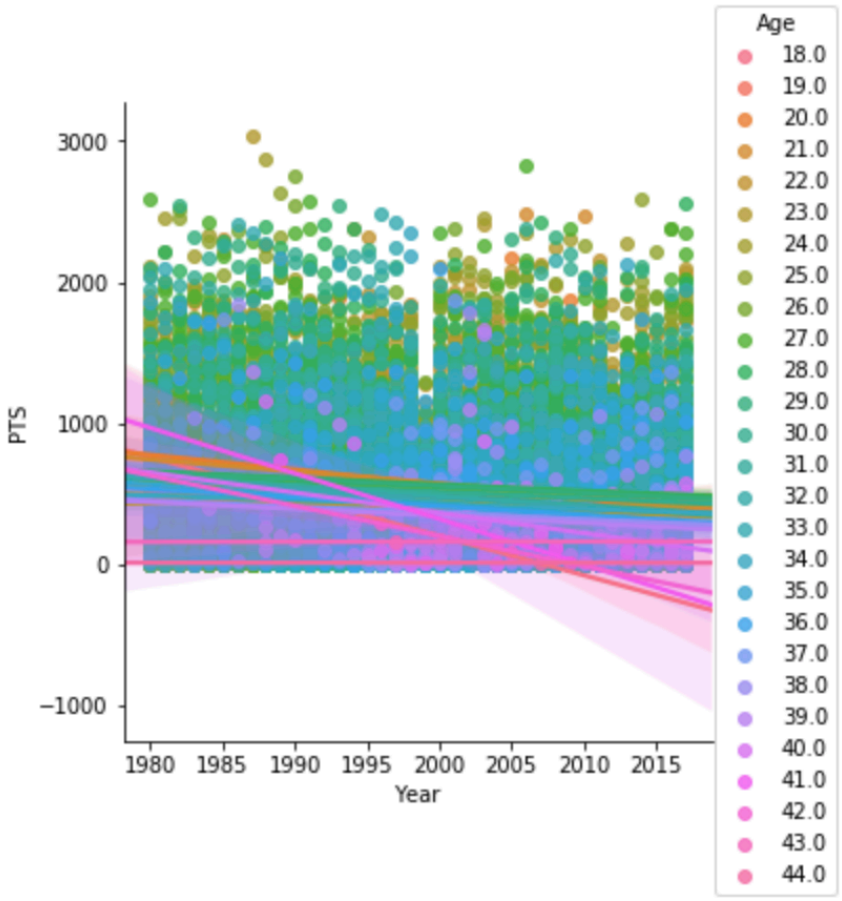


1. This is a plot from which I want to find correlation between ‘Year’ and ‘PER’. Through the plot, we could see that players now may get a bit more points than players before, but the gap is negligible.



2. This is a plot from which I want to find correlation between ‘TS%’ and ‘PER’. Through the plot, we could see that these two attributes are proportional basically. Higher TS%, higher PER. But there are also some exceptions, especially those TS% equals 1.0 or 0.0. I think they may be bad data and should be removed.





3.

There are two plots from which I want to find correlation among ‘Age’, ‘PTS’ and ‘Year’.
It seems that this plot follows to normal distribution as a whole;
Through the plot, we could see that too young or too old players cannot get a lot of points;
It seems that players between age 27 and 31 could get more points than other ages;
As for players of each age from different years, the players in green and blue are more likely to get fewer points than ones from other years.