Machine Learning 7 Assignment

Data Description :

A look at the data Before we dive into the algorithm, let's take a look at our data. Each row in the data contains information on how a player performed in the 2013-2014 NBA season. Download 'nba_2013.csv' file from this link: https://www.dropbox.com/s/b3nv38jjo5dxcl6/nba_2013.csv?dl=0 (https://www.dropbox.com/s/b3nv38jjo5dxcl6/nba_2013.csv?dl=0)

Here are some selected columns from the data:

- player - name of the player
- pos - the position of the player
- g - number of games the player was in
- gs - number of games the player started
- pts - total points the player scored

There are many more columns in the data, mostly containing information about average player game performance over the course of the season. See this site for an explanation of the rest of them.

In [2]:

```
import pandas as pd
import numpy as np
import math
from math import sqrt
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import  train_test_split, cross_val_score, KFold

from sklearn import metrics
import warnings
warnings.filterwarnings('ignore')

import pickle
```

In [3]:

```
pd.set_option("display.max_columns",100)
```

In [4]:

```
nba_df = pd.read_csv('nba_2013.csv')
```

In [61]:

```
nba_df.shape
```

Out[61]:

```
(481, 31)
```

In [6]:

```
columns = nba_df.columns.values
columns
```

Out[6]:

```
array(['player', 'pos', 'age', 'bref_team_id', 'g', 'gs', 'mp', 'fg',
       'fga', 'fg.', 'x3p', 'x3pa', 'x3p.', 'x2p', 'x2pa', 'x2p.', 'efg.',
       'ft', 'fta', 'ft.', 'orb', 'drb', 'trb', 'ast', 'stl', 'blk',
       'tov', 'pf', 'pts', 'season', 'season_end'], dtype=object)
```

In [7]:

```
nba_df.head()
```

Out[7]:

	player	pos	age	bref_team_id	g	gs	mp	fg	fga	fg.	x3p	x3pa	x3p.	x2p
0	Quincy Acy	SF	23	TOT	63	0	847	66	141	0.468	4	15	0.266667	62
1	Steven Adams	C	20	OKC	81	20	1197	93	185	0.503	0	0	NaN	93
2	Jeff Adrien	PF	27	TOT	53	12	961	143	275	0.520	0	0	NaN	143
3	Arron Afflalo	SG	28	ORL	73	73	2552	464	1011	0.459	128	300	0.426667	336
4	Alexis Ajinca	C	25	NOP	56	30	951	136	249	0.546	0	1	0.000000	136

In [8]:

```
nba_df.tail()
```

Out[8]:

	player	pos	age	bref_team_id	g	gs	mp	fg	fga	fg.	x3p	x3pa	x3p.
476	Tony Wroten	SG	20	PHI	72	16	1765	345	808	0.427	40	188	0.212766
477	Nick Young	SG	28	LAL	64	9	1810	387	889	0.435	135	350	0.385714
478	Thaddeus Young	PF	25	PHI	79	78	2718	582	1283	0.454	90	292	0.308219
479	Cody Zeller	C	21	CHA	82	3	1416	172	404	0.426	0	1	0.000000
480	Tyler Zeller	C	24	CLE	70	9	1049	156	290	0.538	0	1	0.000000

In [9]:

```
nba_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 481 entries, 0 to 480
Data columns (total 31 columns):
player          481 non-null object
pos             481 non-null object
age             481 non-null int64
bref_team_id    481 non-null object
g               481 non-null int64
gs              481 non-null int64
mp              481 non-null int64
fg              481 non-null int64
fga             481 non-null int64
fg.             479 non-null float64
x3p             481 non-null int64
x3pa            481 non-null int64
x3p.            414 non-null float64
x2p             481 non-null int64
x2pa            481 non-null int64
x2p.            478 non-null float64
efg.            479 non-null float64
ft              481 non-null int64
fta             481 non-null int64
ft.             461 non-null float64
orb             481 non-null int64
drb             481 non-null int64
trb             481 non-null int64
ast             481 non-null int64
stl             481 non-null int64
blk             481 non-null int64
tov             481 non-null int64
pf              481 non-null int64
pts             481 non-null int64
season          481 non-null object
season_end      481 non-null int64
dtypes: float64(5), int64(22), object(4)
memory usage: 116.6+ KB
```

In [10]:

```
nba_df.get_dtype_counts()
```

Out[10]:

```
float64     5
int64      22
object      4
dtype: int64
```

In [11]:

```
# Extrat the numerical and categorical columns list
num_cols = nba_df.select_dtypes(exclude = 'object').columns.values
cat_cols = nba_df.select_dtypes(include = 'object').columns.values
num_cols, cat_cols
```

Out[11]:

```
(array(['age', 'g', 'gs', 'mp', 'fg', 'fga', 'fg.', 'x3p', 'x3pa', 'x3p.',
        'x2p', 'x2pa', 'x2p.', 'efg.', 'ft', 'fta', 'ft.', 'orb', 'drb',
        'trb', 'ast', 'stl', 'blk', 'tov', 'pf', 'pts', 'season_end'],
       dtype=object),
 array(['player', 'pos', 'bref_team_id', 'season'], dtype=object))
```

In [12]:

```
nba_df[num_cols].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 481 entries, 0 to 480
Data columns (total 27 columns):
age           481 non-null int64
g             481 non-null int64
gs            481 non-null int64
mp            481 non-null int64
fg            481 non-null int64
fga           481 non-null int64
fg.           479 non-null float64
x3p           481 non-null int64
x3pa          481 non-null int64
x3p.          414 non-null float64
x2p           481 non-null int64
x2pa          481 non-null int64
x2p.          478 non-null float64
efg.          479 non-null float64
ft            481 non-null int64
fta           481 non-null int64
ft.           461 non-null float64
orb           481 non-null int64
drb           481 non-null int64
trb           481 non-null int64
ast           481 non-null int64
stl           481 non-null int64
blk           481 non-null int64
tov           481 non-null int64
pf            481 non-null int64
pts           481 non-null int64
season_end    481 non-null int64
dtypes: float64(5), int64(22)
memory usage: 101.5 KB
```

looking at the information columns 'fg.', 'x3p.', 'x2p.', 'efg.', 'ft.' has null values

In [13]:

```
nba_df[num_cols].nunique()
```

Out[13]:

```
age            21
g              82
gs             80
mp            433
fg            296
fga           372
fg.           212
x3p           134
x3pa          221
x3p.          272
x2p           274
x2pa          339
x2p.          408
efg.          202
ft            203
fta           231
ft.           244
orb           155
drb           267
trb           302
ast           230
stl           120
blk            90
tov           180
pf            204
pts           379
season_end      1
dtype: int64
```

Check the Null values

In [14]:

```
nba_null_values = pd.DataFrame({'total_null_values': nba_df[num_cols].isna().sum(), 'null_p
nba_null_values
```

Out[14]:

	total_null_values	null_percentage
age	0	0.000000
g	0	0.000000
gs	0	0.000000
mp	0	0.000000
fg	0	0.000000
fga	0	0.000000
fg.	2	0.415800
x3p	0	0.000000
x3pa	0	0.000000
x3p.	67	13.929314
x2p	0	0.000000
x2pa	0	0.000000
x2p.	3	0.623701
efg.	2	0.415800
ft	0	0.000000
fta	0	0.000000
ft.	20	4.158004
orb	0	0.000000
drb	0	0.000000
trb	0	0.000000
ast	0	0.000000
stl	0	0.000000
blk	0	0.000000
tov	0	0.000000
pf	0	0.000000
pts	0	0.000000
season_end	0	0.000000

In [15]:

```
nba_df.loc[(nba_df.isna()).any(axis=1),:].head()
```

Out[15]:

	player	pos	age	bref_team_id	g	gs	mp	fg	fga	fg.	x3p	x3pa	x3p.	x2p
1	Steven Adams	C	20	OKC	81	20	1197	93	185	0.503	0	0	NaN	93
2	Jeff Adrien	PF	27	TOT	53	12	961	143	275	0.520	0	0	NaN	143
5	Cole Aldrich	C	25	NYK	46	2	330	33	61	0.541	0	0	NaN	33
11	Louis Amundson	PF	31	TOT	19	0	185	16	32	0.500	0	0	NaN	16
18	Joel Anthony	C	31	TOT	33	0	186	12	32	0.375	0	0	NaN	12

In [16]:

```
nba_df.interpolate(value=np.NaN, method='nearest', axis=0, inplace=True)
```

In [17]:

```
nba_df.loc[(nba_df.isna()).any(axis=1),:].shape
```

Out[17]:

```
(0, 31)
```

In [18]:

```
nba_df.loc[(nba_df==0).all(axis=1),:].shape
```

Out[18]:

```
(0, 31)
```

No rows with all columns values == 0

In [19]:

```
nba_df.loc[(nba_df==0).any(axis=1),:].shape
```

Out[19]:

```
(222, 31)
```

In [20]:

```
nba_df.loc[(nba_df==0).any(axis=1),:].head()
```

Out[20]:

	player	pos	age	bref_team_id	g	gs	mp	fg	fga	fg.	x3p	x3pa	x3p.	x2p
0	Quincy Acy	SF	23	TOT	63	0	847	66	141	0.468	4	15	0.266667	62
1	Steven Adams	C	20	OKC	81	20	1197	93	185	0.503	0	0	0.266667	93
2	Jeff Adrien	PF	27	TOT	53	12	961	143	275	0.520	0	0	0.426667	143
4	Alexis Ajinca	C	25	NOP	56	30	951	136	249	0.546	0	1	0.000000	136
5	Cole Aldrich	C	25	NYK	46	2	330	33	61	0.541	0	0	0.000000	33

The zeroes in the dataset seem to be valid zeros. So, no cleaning is required

Clean Categorical Columns

In [21]:

```
nba_df[cat_cols].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 481 entries, 0 to 480
Data columns (total 4 columns):
player          481 non-null object
pos             481 non-null object
bref_team_id    481 non-null object
season          481 non-null object
dtypes: object(4)
memory usage: 15.1+ KB
```

There are no null values in the categorical columns

In [22]:

```
nba_df['player'].nunique()
```

Out[22]:

481

In [23]:

```
nba_df['bref_team_id'].unique()
```

Out[23]:

```
array(['TOT', 'OKC', 'ORL', 'NOP', 'NYK', 'POR', 'MIA', 'MEM', 'BRK',
       'PHI', 'MIL', 'ATL', 'WAS', 'GSW', 'DEN', 'HOU', 'SAS', 'BOS',
       'PHO', 'MIN', 'LAC', 'CLE', 'UTA', 'DET', 'CHA', 'DAL', 'CHI',
       'LAL', 'IND', 'TOR', 'SAC'], dtype=object)
```

In [24]:

```
nba_df['bref_team_id'].nunique()
```

Out[24]:

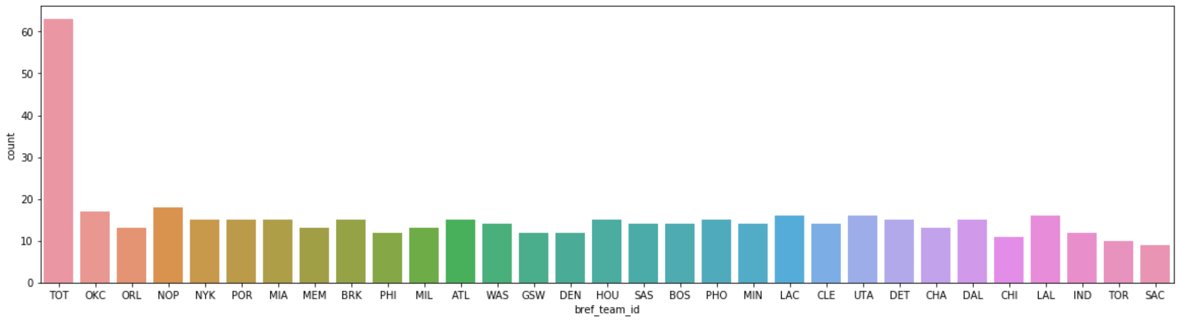
```
31
```

In [25]:

```
plt.figure(figsize = (20,5))
sns.countplot(x = nba_df['bref_team_id'], data = nba_df)
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xae9ccc0>
```



The value TOT is a valid value in this dataset

In [26]:

```
nba_df['pos'].unique()
```

Out[26]:

```
array(['SF', 'C', 'PF', 'SG', 'PG', 'G', 'F'], dtype=object)
```

In [27]:

```
nba_df['pos'].nunique()
```

Out[27]:

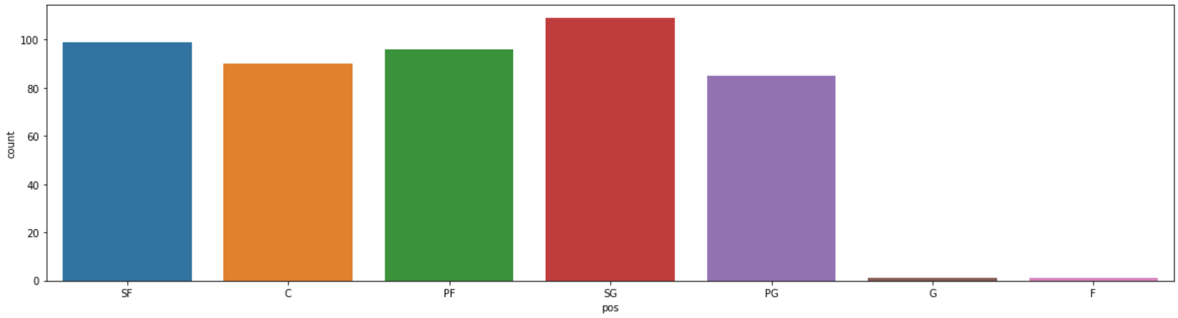
```
7
```

In [28]:

```
plt.figure(figsize = (20,5))
sns.countplot(x= nba_df['pos'], data = nba_df)
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xb305438>
```



In [29]:

```
# Identify the players for whom these invalid values G and F
nba_df.loc[(nba_df['pos'].isin(['G','F'])),['player','pos']]
```

Out[29]:

	player	pos
224	Damion James	G
356	Josh Powell	F

In [30]:

```
# Replace the invalid values with the ones mentioned above
nba_df['pos'].replace(to_replace ='G',value= 'SG',inplace=True)
nba_df['pos'].replace(to_replace ='F',value= 'PF',inplace=True)
```

In [31]:

```
# Check the players for whom these invalid values G and F
nba_df.loc[(nba_df['pos'].isin(['G','F'])),['player','pos']]
```

Out[31]:

player	pos

In [32]:

```
# Identify the players for whom these invalid values G and F
nba_df.loc[[224,356],['player','pos']]
```

Out[32]:

	player	pos
224	Damion James	SG
356	Josh Powell	PF

The replacement of the values was successful and the column is now clean

Getting Basic Statistical Information

In [33]:

```
#nba_df.describe()
print(nba_df.describe())
```

```
              age           g          gs           mp          fg  \
count  481.000000  481.000000  481.000000   481.000000  481.000000
mean    26.509356   53.253638   25.571726  1237.386694  192.881497
std      4.198265   25.322711   29.658465   897.258840  171.832793
min     19.000000    1.000000    0.000000     1.000000    0.000000
25%     23.000000   32.000000    0.000000   388.000000   47.000000
50%     26.000000   61.000000   10.000000  1141.000000  146.000000
75%     29.000000   76.000000   54.000000  2016.000000  307.000000
max     39.000000   83.000000   82.000000  3122.000000  849.000000

               fga         fg.         x3p        x3pa        x3p.  \
count   481.000000  481.000000  481.000000  481.000000  481.000000
mean    424.463617    0.436268   39.613306  110.130977    0.288133
std     368.850833    0.098509   50.855639  132.751732    0.157492
min       0.000000    0.000000    0.000000    0.000000    0.000000
25%     110.000000    0.400000    0.000000    3.000000    0.238095
50%     332.000000    0.437000   16.000000   48.000000    0.333333
75%     672.000000    0.479000   68.000000  193.000000    0.375000
max    1688.000000    1.000000  261.000000  615.000000    1.000000

              x2p         x2pa        x2p.        efg.          ft  \
count  481.000000   481.000000  481.000000  481.000000  481.000000
mean   153.268191   314.332640    0.467357    0.480661   91.205821
std    147.223161   294.174554    0.104521    0.099388  103.667725
min      0.000000     0.000000    0.000000    0.000000    0.000000
25%     31.000000    67.000000    0.434783    0.451000   16.000000
50%    110.000000   227.000000    0.474674    0.488000   53.000000
75%    230.000000   459.000000    0.513932    0.525000  126.000000
max    706.000000  1408.000000    1.000000    1.000000  703.000000

              fta         ft.         orb         drb          trb  \
count  481.000000  481.000000  481.000000  481.000000   481.000000
mean   120.642412    0.723842   55.810811  162.817048   218.627859
std    131.240639    0.158851   62.101191  145.348116   200.356507
min      0.000000    0.000000    0.000000    0.000000     0.000000
25%     22.000000    0.655000   12.000000   43.000000    55.000000
50%     73.000000    0.754000   35.000000  135.000000   168.000000
75%    179.000000    0.821000   73.000000  230.000000   310.000000
max    805.000000    1.000000  440.000000  783.000000  1114.000000

              ast         stl         blk         tov          pf  \
count  481.000000  481.000000  481.000000  481.000000  481.000000
mean   112.536383   39.280665   24.103950   71.862786  105.869023
std    131.019557   34.783590   30.875381   62.701690   71.213627
min      0.000000    0.000000    0.000000    0.000000    0.000000
25%     20.000000    9.000000    4.000000   21.000000   44.000000
50%     65.000000   32.000000   14.000000   58.000000  104.000000
75%    152.000000   60.000000   32.000000  108.000000  158.000000
max    721.000000  191.000000  219.000000  295.000000  273.000000

               pts  season_end
count   481.000000       481.0
mean    516.582121      2013.0
std     470.422228         0.0
min       0.000000      2013.0
25%     115.000000      2013.0
```

```
50%     401.000000    2013.0
75%     821.000000    2013.0
max    2593.000000    2013.0
```

Explore Data

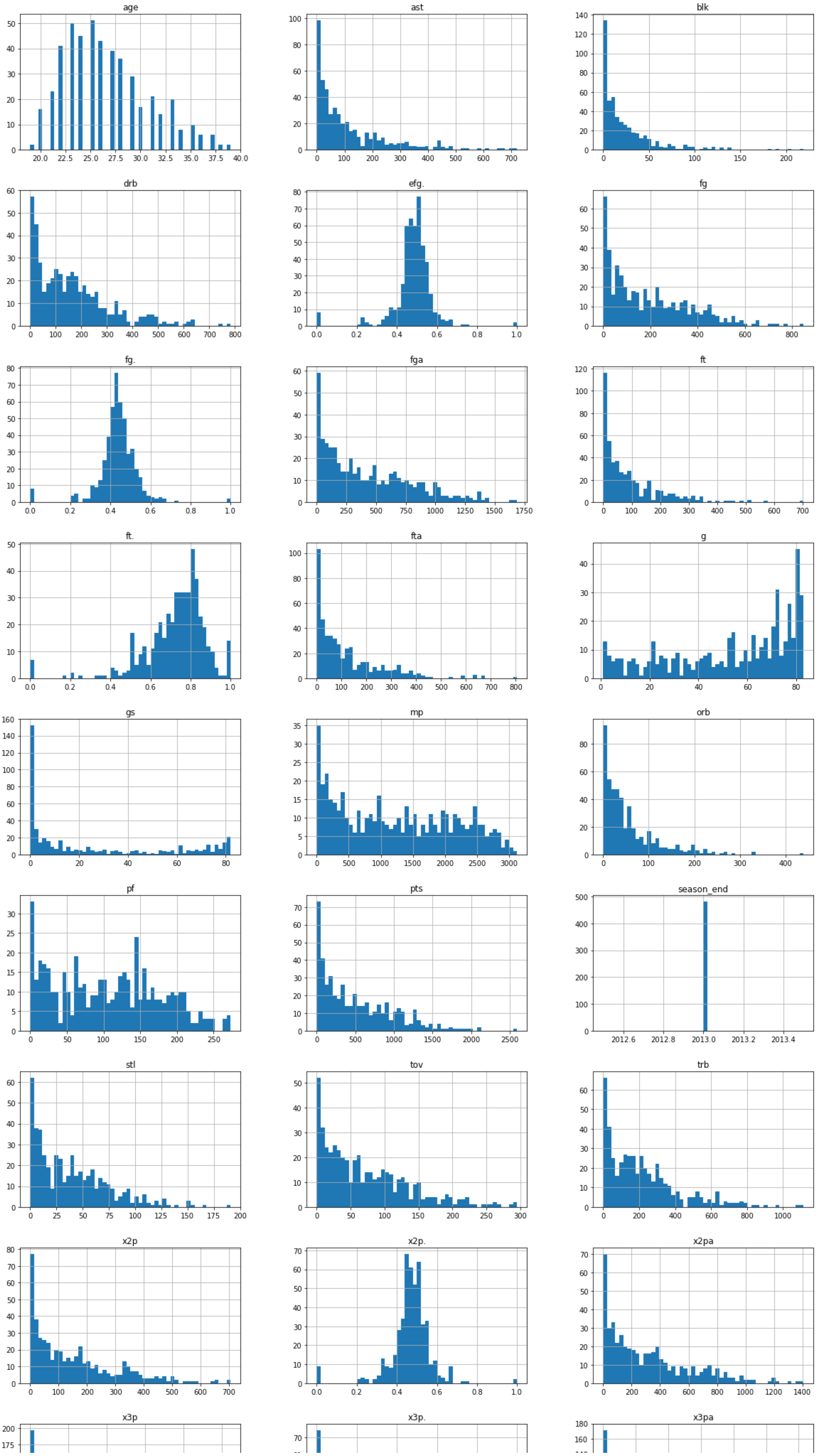
Uni-variate - Numerical columns

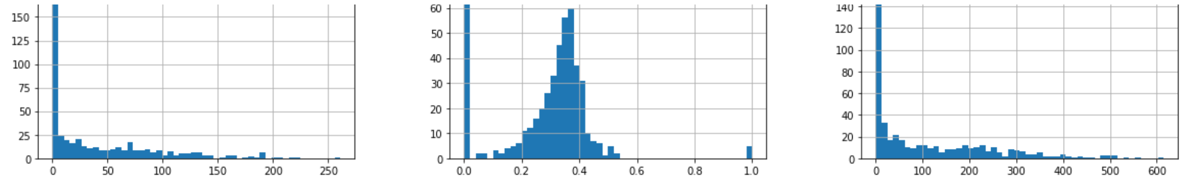
In [34]:

```
nba_df[num_cols].hist(bins=50,figsize=(20,40), layout= (9,3))
```

Out[34]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000000000B5ED780
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B61E160
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B647470
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000B66F780
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B698A90
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B698AC8
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000B8E70B8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B9103C8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B93C6D8
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000B9649E8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000B98FCF8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000BDE3048
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000BE0D358
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000BE34668
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000BE5F978
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000BE8BC88
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000BEB5F98
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000BEE82E8
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000BF115F8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000BF3B908
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000BF66C18
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000BF92F28
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000C2A5278
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000C2CF588
>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000000000C2F7898
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000C320BA8
>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000000C34CEB8
>]],
      dtype=object)
```


age
ast
blk
drb
efg.
fg
fg.
fga
ft
ft.
fta
g
gs
mp
orb
pf
pts
season_end
stl
tov
trb
x2p
x2p.
x2pa
x3p
x3p.
x3pa



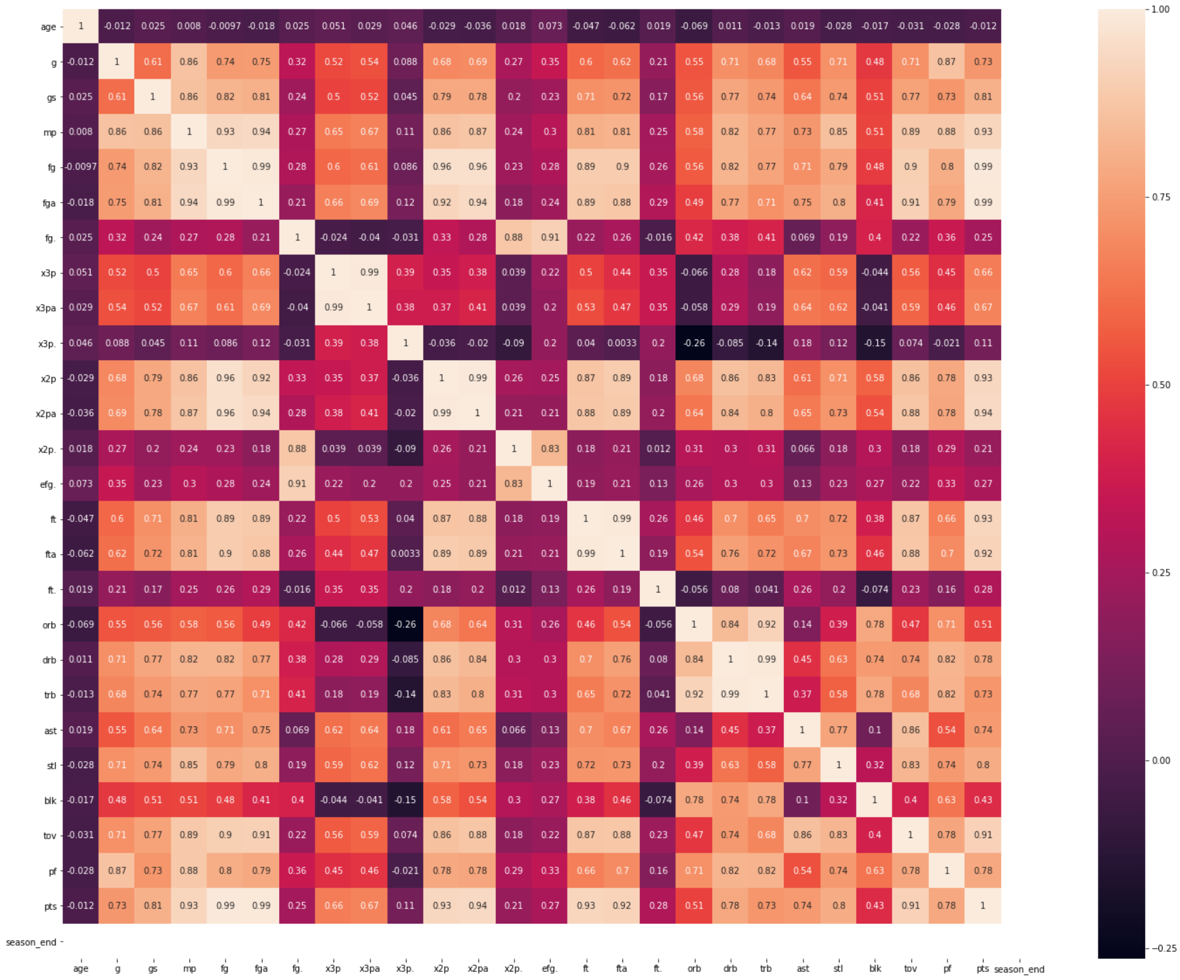
Multi-variate

In [35]:

```
plt.figure(figsize=(25,20))
sns.heatmap(nba_df.corr(), annot = True)
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xc43c748>
```



In [36]:

```
print(nba_df.corr().iloc[-2:-1,:])
```

```
          age         g        gs        mp        fg       fga       fg.  \
pts -0.01191  0.728462  0.810294  0.927464  0.992041  0.989211  0.249419   

          x3p      x3pa      x3p.       x2p      x2pa      x2p.      efg.  \
pts  0.655342  0.672076  0.113962  0.931493  0.937036  0.213475  0.269157   

           ft       fta       ft.       orb       drb      trb       ast  \
pts  0.927618  0.918979  0.281958  0.505524  0.784675  0.72593  0.738295   

          stl       blk       tov       pf  pts  season_end  
pts  0.797449  0.433549  0.912724  0.77806  1.0         NaN  
```

Engineer Features

In [37]:

```
print(num_cols)
print(cat_cols)
```

```
['age' 'g' 'gs' 'mp' 'fg' 'fga' 'fg.' 'x3p' 'x3pa' 'x3p.' 'x2p' 'x2pa'
 'x2p.' 'efg.' 'ft' 'fta' 'ft.' 'orb' 'drb' 'trb' 'ast' 'stl' 'blk' 'tov'
 'pf' 'pts' 'season_end']
['player' 'pos' 'bref_team_id' 'season']
```

Encode Categorical Columns

In [38]:

```
cols_to_encode = ['pos','bref_team_id']
prefixes = ['pos','team']
nba_encoded= pd.get_dummies(data = nba_df, prefix = prefixes, columns = cols_to_encode, pre
```

In [39]:

```
nba_encoded.columns
```

Out[39]:

```
Index(['player', 'age', 'g', 'gs', 'mp', 'fg', 'fga', 'fg.', 'x3p', 'x3pa',
       'x3p.', 'x2p', 'x2pa', 'x2p.', 'efg.', 'ft', 'fta', 'ft.', 'orb', 'dr
b',
       'trb', 'ast', 'stl', 'blk', 'tov', 'pf', 'pts', 'season', 'season_en
d',
       'pos_PF', 'pos_PG', 'pos_SF', 'pos_SG', 'team_BOS', 'team_BRK',
       'team_CHA', 'team_CHI', 'team_CLE', 'team_DAL', 'team_DEN', 'team_DE
T',
       'team_GSW', 'team_HOU', 'team_IND', 'team_LAC', 'team_LAL', 'team_ME
M',
       'team_MIA', 'team_MIL', 'team_MIN', 'team_NOP', 'team_NYK', 'team_OK
C',
       'team_ORL', 'team_PHI', 'team_PHO', 'team_POR', 'team_SAC', 'team_SA
S',
       'team_TOR', 'team_TOT', 'team_UTA', 'team_WAS'],
      dtype='object')
```

In [40]:

```
nba_encoded.drop(['player','season','season_end'], axis = 1,inplace = True)
```

In [41]:

```
nba_encoded_cols = nba_encoded.columns.values
nba_encoded_cols
```

Out[41]:

```
array(['age', 'g', 'gs', 'mp', 'fg', 'fga', 'fg.', 'x3p', 'x3pa', 'x3p.',
       'x2p', 'x2pa', 'x2p.', 'efg.', 'ft', 'fta', 'ft.', 'orb', 'drb',
       'trb', 'ast', 'stl', 'blk', 'tov', 'pf', 'pts', 'pos_PF', 'pos_PG',
       'pos_SF', 'pos_SG', 'team_BOS', 'team_BRK', 'team_CHA', 'team_CHI',
       'team_CLE', 'team_DAL', 'team_DEN', 'team_DET', 'team_GSW',
       'team_HOU', 'team_IND', 'team_LAC', 'team_LAL', 'team_MEM',
       'team_MIA', 'team_MIL', 'team_MIN', 'team_NOP', 'team_NYK',
       'team_OKC', 'team_ORL', 'team_PHI', 'team_PHO', 'team_POR',
       'team_SAC', 'team_SAS', 'team_TOR', 'team_TOT', 'team_UTA',
       'team_WAS'], dtype=object)
```

Data Preprocessing - Normalization with MinMaxScaler

In [42]:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
nba_scaled_array = scaler.fit_transform(nba_encoded)
```

In [43]:

```
df_nba_scaled = pd.DataFrame(data = nba_scaled_array, columns = nba_encoded_cols)
```

In [44]:

```
df_nba_scaled.describe()
```

Out[44]:

	age	g	gs	mp	fg	fga	fg.	
count	481.000000	481.000000	481.000000	481.000000	481.000000	481.000000	481.000000	48
mean	0.375468	0.637239	0.311850	0.396151	0.227187	0.251459	0.436268	
std	0.209913	0.308814	0.361689	0.287491	0.202394	0.218514	0.098509	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	0.200000	0.378049	0.000000	0.123999	0.055359	0.065166	0.400000	
50%	0.350000	0.731707	0.121951	0.365268	0.171967	0.196682	0.437000	
75%	0.500000	0.914634	0.658537	0.645626	0.361602	0.398104	0.479000	
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

Generating Input data X and Output Y, and Split the Data for Training and Testing

In [45]:

```
X = df_nba_scaled.drop('pts', axis = 1)
Y = df_nba_scaled['pts']
```

In [46]:

```
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, random_state = 1
```

Fit the Base Models

using Euclidean Distance as a distance metrics

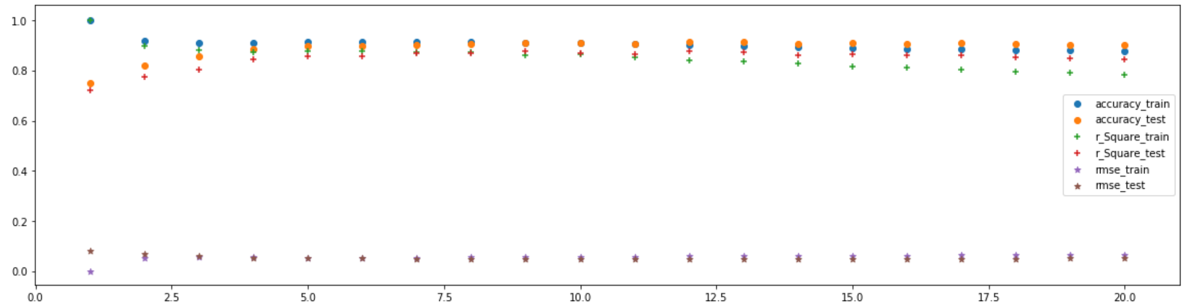
In [47]:

```
k_values =[]
r_Square_train_values =[]
r_Square_test_values =[]
rmse_train_values = []
rmse_test_values =[]
accuracy_test =[]
accuracy_train =[]

import math

for k in range(1,21):
    knn = KNeighborsRegressor(n_neighbors = k, weights='uniform', algorithm='auto')
    model = knn.fit(x_train, y_train)
    y_pred = model.predict(x_test)

    k_values.append(k)

    r_Square_train_values.append(metrics.r2_score(model.predict(x_train), y_train))
    r_Square_test_values.append(metrics.r2_score(model.predict(x_test), y_test))

    rmse_train_values.append(math.sqrt(metrics.mean_squared_error(model.predict(x_train), y
    rmse_test_values.append(math.sqrt(metrics.mean_squared_error(model.predict(x_test), y_t

    accuracy_train.append(model.score(x_train, y_train))
    accuracy_test.append(model.score(x_test, y_test))

    print("Accuracy: ", model.score(x_test, y_test),"for K-Value:",k)
```

```
Accuracy:  0.7513843147299952 for K-Value: 1
Accuracy:  0.8199725833331336 for K-Value: 2
Accuracy:  0.8580731090101001 for K-Value: 3
Accuracy:  0.886968662676202 for K-Value: 4
Accuracy:  0.8959555549179624 for K-Value: 5
Accuracy:  0.8992614554993184 for K-Value: 6
Accuracy:  0.9036493448407232 for K-Value: 7
Accuracy:  0.9072877074161129 for K-Value: 8
Accuracy:  0.9111726205321228 for K-Value: 9
Accuracy:  0.908236609041231 for K-Value: 10
Accuracy:  0.9072146074098933 for K-Value: 11
Accuracy:  0.9136840025075303 for K-Value: 12
Accuracy:  0.913021296223261 for K-Value: 13
Accuracy:  0.9059200749886736 for K-Value: 14
Accuracy:  0.9082728625112148 for K-Value: 15
Accuracy:  0.9074960092931297 for K-Value: 16
Accuracy:  0.9082081255476709 for K-Value: 17
Accuracy:  0.9038429793012481 for K-Value: 18
Accuracy:  0.9023496673490286 for K-Value: 19
Accuracy:  0.9014758950881321 for K-Value: 20
```

In [48]:

```
plt.figure(figsize = (20,5))

plt.scatter(k_values, accuracy_train, label = 'accuracy_train')
plt.scatter(k_values, accuracy_test, label = 'accuracy_test')

plt.scatter(k_values, r_Square_train_values, label = 'r_Square_train',marker='+')
plt.scatter(k_values, r_Square_test_values, label = 'r_Square_test',marker='+')

plt.scatter(k_values, rmse_train_values, label = 'rmse_train',marker='*')
plt.scatter(k_values, rmse_test_values, label = 'rmse_test',marker='*')

plt.legend()
plt.show()
```



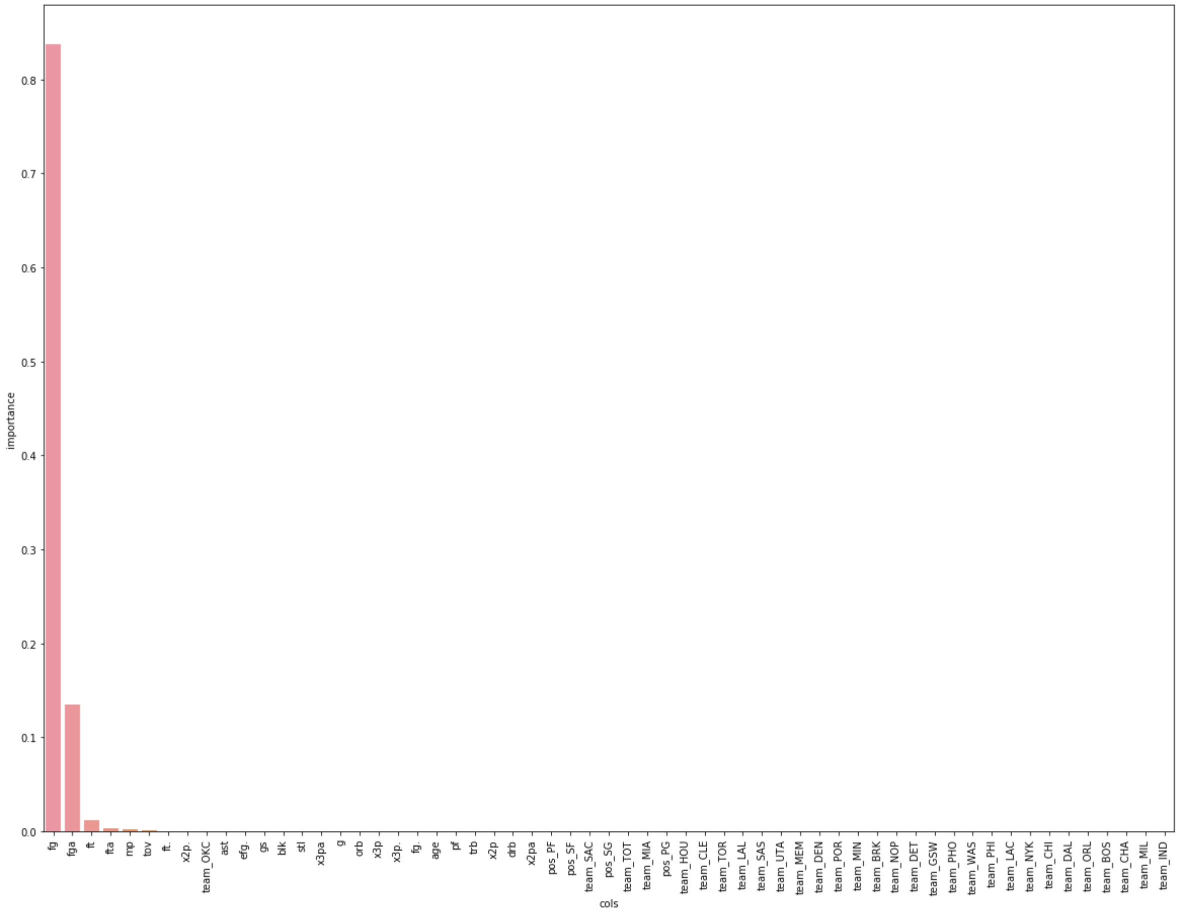
Feature selection

In [49]:

```
from sklearn.ensemble import RandomForestRegressor
rndf = RandomForestRegressor(n_estimators=150)
rndf.fit(x_train, y_train)
importance = pd.DataFrame.from_dict({'cols':x_train.columns, 'importance': rndf.feature_imp
importance = importance.sort_values(by='importance', ascending=False)
plt.figure(figsize=(20,15))
sns.barplot(importance.cols, importance.importance)
plt.xticks(rotation=90)
```

Out[49]:

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58]),
 <a list of 59 Text xticklabel objects>)
```



In [50]:

```
imp_cols = importance[importance.importance > 0.0005].cols.values
imp_cols
```

Out[50]:

```
array(['fg', 'fga', 'ft', 'fta', 'mp', 'tov', 'ft.', 'x2p.', 'team_OKC',
       'ast', 'efg.', 'gs'], dtype=object)
```

In [51]:

```
x1_train,x1_test, y1_train, y1_test = train_test_split(X[imp_cols],Y,test_size=0.2,random_s
```

Fit Features Selected Model and Collect the Metrics

Distance Metric = Euclidean Distance

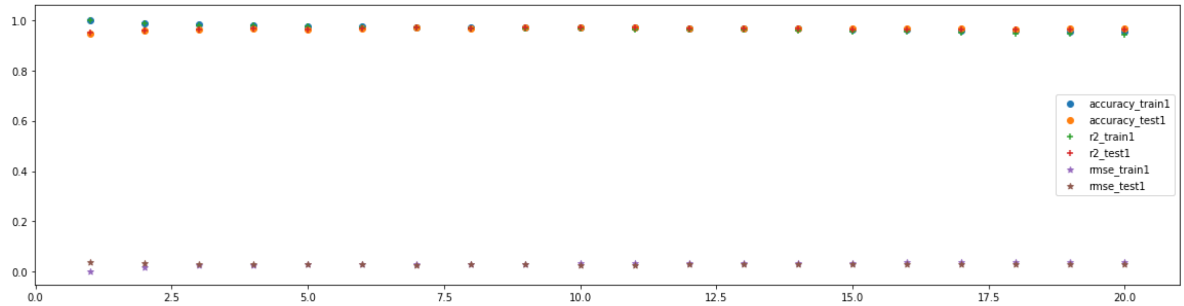
In [52]:

```
k_values =[]
r2_train1_values =[]
r2_test1_values =[]
rmse_train1_values = []
rmse_test1_values =[]
accuracy_test1 =[]
accuracy_train1 =[]


import math

for k in range(1,21):
    knn = KNeighborsRegressor(n_neighbors = k, weights='uniform', algorithm='auto')
    model = knn.fit(x1_train, y1_train)
    y_pred1 = model.predict(x1_test)
    k_values.append(k)
    r2_train1_values.append(metrics.r2_score(model.predict(x1_train), y1_train))
    r2_test1_values.append(metrics.r2_score(model.predict(x1_test), y1_test))
    rmse_train1_values.append(math.sqrt(metrics.mean_squared_error(model.predict(x1_train),
    rmse_test1_values.append(math.sqrt(metrics.mean_squared_error(model.predict(x1_test), y
    accuracy_train1.append(model.score(x1_train, y1_train))
    accuracy_test1.append(model.score(x1_test, y1_test))

    print("Accuracy: ", model.score(x1_test, y1_test),"for K-Value:",k)
```

```
Accuracy:  0.9477287435698372 for K-Value: 1
Accuracy:  0.9598869260830679 for K-Value: 2
Accuracy:  0.9649462580074877 for K-Value: 3
Accuracy:  0.9685291932590752 for K-Value: 4
Accuracy:  0.9645880837247166 for K-Value: 5
Accuracy:  0.9674408273490529 for K-Value: 6
Accuracy:  0.9723051171824373 for K-Value: 7
Accuracy:  0.9692621510389523 for K-Value: 8
Accuracy:  0.9704494424866856 for K-Value: 9
Accuracy:  0.971028298313233 for K-Value: 10
Accuracy:  0.9713848167010354 for K-Value: 11
Accuracy:  0.968143843635373 for K-Value: 12
Accuracy:  0.9680750966733992 for K-Value: 13
Accuracy:  0.9680836638359441 for K-Value: 14
Accuracy:  0.9668792148693957 for K-Value: 15
Accuracy:  0.9672120537771 for K-Value: 16
Accuracy:  0.966887364032113 for K-Value: 17
Accuracy:  0.9650130541877595 for K-Value: 18
Accuracy:  0.9660394226727712 for K-Value: 19
Accuracy:  0.9664471341205868 for K-Value: 20
```

In [53]:

```
plt.figure(figsize = (20,5))

plt.scatter(k_values, accuracy_train1, label = 'accuracy_train1')
plt.scatter(k_values, accuracy_test1, label = 'accuracy_test1')

plt.scatter(k_values, r2_train1_values, label = 'r2_train1', marker='+')
plt.scatter(k_values, r2_test1_values, label = 'r2_test1', marker='+')

plt.scatter(k_values, rmse_train1_values, label = 'rmse_train1', marker='*')
plt.scatter(k_values, rmse_test1_values, label = 'rmse_test1', marker='*')

plt.legend()
plt.show()
```



Validate Model

In [54]:

```
# Validate Base Model

cv_scores_euclid =[]
#cv_scores_manhattan =[]
#cv_scores_minkowski =[]

k_values =[]

for k in range(1, 21):

    k_values.append(k)

    knn_euclid = KNeighborsRegressor(n_neighbors = k, weights='uniform', algorithm='auto')
    scores = cross_val_score(knn_euclid, x_train, y_train, cv=10, scoring='neg_mean_squarec
    cv_scores_euclid.append(scores.mean())

    #knn_manhattan = KNeighborsRegressor(n_neighbors = k, weights='uniform', algorithm='aut
    #scores = cross_val_score(knn_manhattan, x_train, y_train, cv=10, scoring='neg_mean_squ
    #cv_scores_manhattan.append(scores.mean())

    #knn_minkowski = KNeighborsRegressor(n_neighbors = k, weights='uniform', algorithm='aut
    #scores = cross_val_score(knn_minkowski, x_train, y_train, cv=10, scoring='neg_mean_squ
    #cv_scores_minkowski.append(scores.mean())
```

The CV score are better for feature selected models

Compare Performance Metrics of Different Models - Euclidean Distance based Models

In [55]:

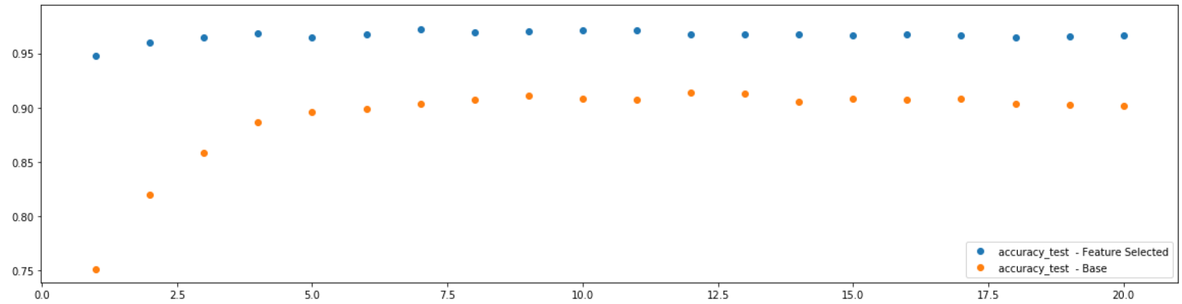
```
# Base Model

k_values =[]
r2_train_values =[]
r2_test_values =[]
rmse_train_values = []
rmse_test_values =[]
accuracy_test =[]
accuracy_train =[]

import math

for k in range(1,21):
    knn = KNeighborsRegressor(n_neighbors = k, weights='uniform', algorithm='auto')
    model = knn.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    k_values.append(k)
    r2_train_values.append(metrics.r2_score(model.predict(x_train), y_train))
    r2_test_values.append(metrics.r2_score(model.predict(x_test), y_test))
    rmse_train_values.append(math.sqrt(metrics.mean_squared_error(model.predict(x_train), y
    rmse_test_values.append(math.sqrt(metrics.mean_squared_error(model.predict(x_test), y_t
    accuracy_train.append(model.score(x_train, y_train))
    accuracy_test.append(model.score(x_test, y_test))
```

In [56]:

```
# Features Selected Model

k_values =[]
r2_train1_values =[]
r2_test1_values =[]
rmse_train1_values = []
rmse_test1_values =[]
accuracy_test1 =[]
accuracy_train1 =[]


import math

for k in range(1,21):
    knn = KNeighborsRegressor(n_neighbors = k, weights='uniform', algorithm='auto')
    model = knn.fit(x1_train, y1_train)
    y_pred1 = model.predict(x1_test)
    k_values.append(k)
    r2_train1_values.append(metrics.r2_score(model.predict(x1_train), y1_train))
    r2_test1_values.append(metrics.r2_score(model.predict(x1_test), y1_test))
    rmse_train1_values.append(math.sqrt(metrics.mean_squared_error(model.predict(x1_train),
    rmse_test1_values.append(math.sqrt(metrics.mean_squared_error(model.predict(x1_test), y
    accuracy_train1.append(model.score(x1_train, y1_train))
    accuracy_test1.append(model.score(x1_test, y1_test))
```

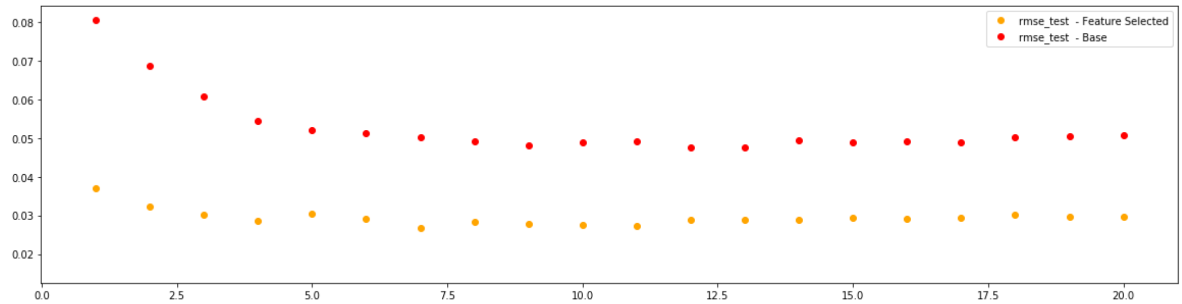
In [57]:

```
plt.figure(figsize = (20,5))

plt.scatter(k_values, accuracy_test1, label = 'accuracy_test  - Feature Selected')
plt.scatter(k_values, accuracy_test , label = 'accuracy_test  - Base')

plt.legend()
plt.show()
```



In [58]:

```
plt.figure(figsize = (20,5))

plt.scatter(k_values, rmse_test1_values, c ='orange', label = 'rmse_test  - Feature Selecte
plt.scatter(k_values, rmse_test_values,  c ='red', label = 'rmse_test  - Base')

plt.legend()
plt.show()
```



Models with k = 4,6 provide us with better metrics so we choose to them

In [59]:

```
knn_7 = KNeighborsRegressor(n_neighbors = 7, weights='uniform', algorithm='auto')
model_7 = knn_7.fit(x1_train, y1_train)
print('K = 6')
print('-'*40)
print('Accuracy: ', model_7.score(x1_test, y1_test))
print('R2-score: ', metrics.r2_score(model_7.predict(x1_test), y1_test))
print('RMSE: ', math.sqrt(metrics.mean_squared_error(model_7.predict(x1_test), y1_test)))
```

```
K = 6
----------------------------------------
Accuracy:  0.9723051171824373
R2-score:  0.9717114203231784
RMSE:  0.026891705150132596
```

The feature selected model for k = 7 performs better for better than the base model

In [ ]:

```

```