In [1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
import time
%matplotlib inline
```

In [2]:
```
from nba_py.team import TeamYearOverYearSplits, TeamList
team_list = TeamList().info().head(30)
```

In [3]:
```
rockets = TeamYearOverYearSplits(1610612745).by_year()
rockets.head(1)
```

Out[3]:

	GROUP_SET	GROUP_VALUE	GP	W	L	W_PCT	MIN	FGM	FGA	FG_PCT	...	TOV_RANK	S
0	By Year	2017-18	82	65	17	0.793	48.2	38.7	84.2	0.46	...	3	

1 rows × 56 columns

In [ ]:
```

```

In [6]:
```
season_team = {}
for team in team_list['TEAM_ID']:
    df = TeamYearOverYearSplits(team,season_type='Playoffs').by_year()
    for index, row in df.iterrows():
        season_data =  season_team.get(row['GROUP_VALUE'])
        if season_data:
            if team not in season_team[row['GROUP_VALUE']]:
                season_team[row['GROUP_VALUE']].append(team)
        else:
            season_team[row['GROUP_VALUE']] = [team]
    time.sleep(2)
```

In [4]:
```
def playoff_team(team_id, season):
    if team_id in season_team[season]:
        return 1
    return 0
```

In [7]:
```
all_team_data = pd.DataFrame()
for team in team_list['TEAM_ID']:
    team_data = TeamYearOverYearSplits(team,measure_type = 'Advanced').h
    team_data['PLAYOFFS'] = team_data.apply(lambda row: playoff_team(tea
    team_data['TEAM_ID'] = team
    all_team_data = pd.concat([all_team_data,team_data])
    time.sleep(2)
```

In [11]:
```
all_team_data['TEAM_ID'].value_counts()
```

Out[11]:
```
1610612751    22
1610612765    22
1610612738    22
1610612739    22
1610612741    22
1610612742    22
1610612743    22
1610612744    22
1610612745    22
1610612746    22
1610612747    22
1610612748    22
1610612749    22
1610612750    22
1610612737    22
1610612752    22
1610612753    22
1610612754    22
1610612755    22
1610612756    22
1610612757    22
1610612758    22
1610612759    22
1610612760    22
1610612761    22
1610612762    22
1610612763    22
1610612764    22
1610612766    20
1610612740    16
Name: TEAM_ID, dtype: int64
```

In [16]:
```
teams = TeamList().info().head(30)
```

In [16]:
```
all_team_data.to_csv('playoff_adv_with_ids.csv')
```

In [5]:
```
playoff_adv = pd.read_csv('playoff_adv_with_ids.csv')
```

In [34]:
```
playoff_reg = pd.read_csv('playoff_with_ids.csv')
```

In [17]:
```
teams.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 5 columns):
LEAGUE_ID       30 non-null object
TEAM_ID         30 non-null int64
MIN_YEAR        30 non-null object
MAX_YEAR        30 non-null object
ABBREVIATION    30 non-null object
dtypes: int64(1), object(4)
memory usage: 1.2+ KB
```

In [7]:
```
1 playoff_adv.head()
```

Out[7]:

	Unnamed: 0	GROUP_SET	GROUP_VALUE	GP	W	L	W_PCT	MIN	OFF_RATING	DEF_RATIN
0	0	By Year	2017-18	82	24	58	0.293	3941.0	102.4	108.
1	1	By Year	2016-17	82	43	39	0.524	3976.0	102.3	103.
2	2	By Year	2015-16	82	48	34	0.585	3966.0	103.0	98.
3	3	By Year	2014-15	82	60	22	0.732	3946.0	106.2	100.
4	4	By Year	2013-14	82	38	44	0.463	3966.0	103.4	104.

5 rows × 45 columns

In [35]:
```
1 idx = np.searchsorted(teams.TEAM_ID.values,playoff_reg.TEAM_ID.values)
```

In [36]:
```
1 playoff_reg['TEAM_NAME'] = teams.ABBREVIATION.values[idx]
```

In [37]:
```
1 playoff_reg['TEAM_NAME'].value_counts()
```

Out[37]:
```
DAL    22
PHX    22
HOU    22
CHI    22
BKN    22
MIA    22
LAC    22
MEM    22
DET    22
SAS    22
NYK    22
UTA    22
SAC    22
GSW    22
WAS    22
DEN    22
IND    22
POR    22
BOS    22
CLE    22
PHI    22
MIN    22
ORL    22
MIL    22
TOR    22
ATL    22
LAL    22
OKC    22
CHA    20
NOP    16
Name: TEAM_NAME, dtype: int64
```

In [20]:
```
1 playoff_adv['TEAM_NAME'] = teams.ABBREVIATION.values[idx]
```

In [21]:
```
1 playoff_adv['TEAM_NAME'].value_counts()
```

Out[21]:
```
DAL    22
PHX    22
HOU    22
CHI    22
BKN    22
MIA    22
LAC    22
MEM    22
DET    22
SAS    22
NYK    22
UTA    22
SAC    22
GSW    22
WAS    22
DEN    22
IND    22
POR    22
BOS    22
CLE    22
PHI    22
MIN    22
ORL    22
MIL    22
TOR    22
ATL    22
LAL    22
OKC    22
CHA    20
NOP    16
Name: TEAM_NAME, dtype: int64
```

In [22]:
```
1 east_teams = ['CHI','BKN','MIA','DET','NYK','WAS','IND','BOS','CLE','PHI
```

In [38]:
```
1 playoff_reg['EASTERN'] = playoff_reg.apply(lambda x: True if x.TEAM_NAME
```

In [27]:
```
1 playoff_adv['EASTERN'] = playoff_adv.apply(lambda x: True if x.TEAM_NAME
```

In [39]:
```
1 playoff_reg['EASTERN'].value_counts()
```

Out[39]:
```
True     328
False    324
Name: EASTERN, dtype: int64
```

In [40]:
```
1 playoff_reg_east = playoff_reg[playoff_reg['EASTERN'] == True]
```

In [41]:
```
1 playoff_reg_west = playoff_reg[playoff_reg['EASTERN'] == False]
```

In [42]:
```
1 playoff_reg_east.to_csv('playoff_reg_east.csv')
```

In [43]:
```
1 playoff_reg_west.to_csv('playoff_reg_west.csv')
```

In [29]:
```
playoff_adv_east = playoff_adv[playoff_adv['EASTERN'] == True]
```

In [30]:
```
playoff_adv_west = playoff_adv[playoff_adv['EASTERN'] == False]
```

In [32]:
```
playoff_adv_east.to_csv('playoff_adv_east.csv')
```

In [33]:
```
playoff_adv_west.to_csv('playoff_adv_west.csv')
```

In [3]:
```
playoff_adv_west = pd.read_csv('playoff_adv_west.csv')
```

In [4]:
```
playoff_adv_east = pd.read_csv('playoff_adv_east.csv')
```

In [5]:
```
playoff_reg_west = pd.read_csv('playoff_reg_west.csv')
```

In [6]:
```
playoff_reg_east = pd.read_csv('playoff_reg_east.csv')
```

In [7]:
```
1 playoff_adv_east.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 328 entries, 0 to 327
Data columns (total 48 columns):
Unnamed: 0          328 non-null int64
Unnamed: 0.1        328 non-null int64
GROUP_SET           328 non-null object
GROUP_VALUE         328 non-null object
GP                  328 non-null int64
W                   328 non-null int64
L                   328 non-null int64
W_PCT               328 non-null float64
MIN                 328 non-null float64
OFF_RATING          328 non-null float64
DEF_RATING          328 non-null float64
NET_RATING          328 non-null float64
AST_PCT             328 non-null float64
AST_TO              328 non-null float64
AST_RATIO           328 non-null float64
OREB_PCT            328 non-null float64
DREB_PCT            328 non-null float64
REB_PCT             328 non-null float64
TM_TOV_PCT          328 non-null float64
EFG_PCT             328 non-null float64
TS_PCT              328 non-null float64
PACE                328 non-null float64
PIE                 328 non-null float64
GP_RANK             328 non-null int64
W_RANK              328 non-null int64
L_RANK              328 non-null int64
W_PCT_RANK          328 non-null int64
MIN_RANK            328 non-null int64
OFF_RATING_RANK     328 non-null int64
DEF_RATING_RANK     328 non-null int64
NET_RATING_RANK     328 non-null int64
AST_PCT_RANK        328 non-null int64
AST_TO_RANK         328 non-null int64
AST_RATIO_RANK      328 non-null int64
OREB_PCT_RANK       328 non-null int64
DREB_PCT_RANK       328 non-null int64
REB_PCT_RANK        328 non-null int64
TM_TOV_PCT_RANK     328 non-null int64
EFG_PCT_RANK        328 non-null int64
TS_PCT_RANK         328 non-null int64
PACE_RANK           328 non-null int64
PIE_RANK            328 non-null int64
CFID                328 non-null int64
CFPARAMS            328 non-null object
PLAYOFFS            328 non-null int64
TEAM_ID             328 non-null int64
TEAM_NAME           328 non-null object
EASTERN             328 non-null bool
dtypes: bool(1), float64(16), int64(27), object(4)
memory usage: 120.8+ KB
```

In [14]:
```
1 teams['TEAM_ID']
```

Out[14]:
```
0     1610612737
1     1610612738
2     1610612739
3     1610612740
4     1610612741
5     1610612742
6     1610612743
7     1610612744
8     1610612745
9     1610612746
10    1610612747
11    1610612748
12    1610612749
13    1610612750
14    1610612751
15    1610612752
16    1610612753
17    1610612754
18    1610612755
19    1610612756
20    1610612757
21    1610612758
22    1610612759
23    1610612760
24    1610612761
25    1610612762
26    1610612763
27    1610612764
28    1610612765
29    1610612766
Name: TEAM_ID, dtype: int64
```

In [ ]:
```
1 all_team_data = pd.DataFrame()
2 for team in team_list['TEAM_ID']:
3     team_data = TeamYearOverYearSplits(team).by_year()
4     team_data['PLAYOFFS'] = team_data.apply(lambda row: playoff_team(tea
5     team_data['TEAM_ID'] = team
6     all_team_data = pd.concat([all_team_data,team_data])
7     time.sleep(2)
```

In [25]:
```
1 playoff_adv_west.head()
```

Out[25]:

	Unnamed: 0	Unnamed: 0.1	GROUP_SET	GROUP_VALUE	GP	W	L	W_PCT	MIN	OFF_RATING
0	66	0	By Year	2017-18	82	48	34	0.585	3991.0	107.7
1	67	1	By Year	2016-17	82	34	48	0.415	3981.0	103.3
2	68	2	By Year	2015-16	82	30	52	0.366	3956.0	103.2
3	69	3	By Year	2014-15	82	45	37	0.549	3956.0	105.4
4	70	4	By Year	2013-14	82	34	48	0.415	3971.0	104.7

5 rows × 48 columns

In [3]:
```
1 regular_stats = pd.read_csv('all_team_playoffs.csv')
2 advs_stats = pd.read_csv('all_team_playoffs_adv.csv')
```

In [7]:
```
1 all_stats_west = pd.concat([playoff_adv_west,playoff_reg_west],axis = 1)
```

In [8]:
```
1 all_stats_west.to_csv('all_stats_playoffs_west.csv')
```

In [8]:
```
1 all_stats_east = pd.concat([playoff_adv_east,playoff_reg_east],axis = 1)
```

In [13]:
```
1 all_stats_east.to_csv('all_stats_playoffs_east.csv')
```

In [60]:
```
1 all_stats = pd.read_csv('all_stats_playoffs.csv')
2
```

In [9]:
```
1 regular_features = playoff_reg_west[['FGM', 'FGA', 'FG_PCT', 'FG3M', 'FG
2        'FT_PCT', 'OREB', 'DREB', 'REB', 'AST', 'TOV', 'STL', 'BLK', 'BLK
3        'PF', 'PFD', 'PTS', 'PLUS_MINUS']]
```

In [10]:
```
1 advs_features = playoff_adv_west[['NET_RATING', 'AST_PCT', 'AST_TO',
2        'AST_RATIO', 'OREB_PCT', 'DREB_PCT', 'REB_PCT', 'TM_TOV_PCT', 'EF
3        'TS_PCT', 'PACE', 'PIE']]
```

In [12]:
```
1 all_features = all_stats_west[['FGM', 'FGA', 'FG_PCT', 'FG3M', 'FG3A',
2        'FT_PCT', 'OREB', 'DREB', 'REB', 'AST', 'TOV', 'STL', 'BLK', 'BLK
3        'PF', 'PFD', 'PTS','AST_PCT', 'AST_TO',
4        'AST_RATIO', 'OREB_PCT', 'DREB_PCT', 'REB_PCT', 'TM_TOV_PCT', 'EF
5        'TS_PCT', 'PACE', 'PIE']]
```

In [13]:
```
1 feature_names = ['FGM', 'FGA', 'FG_PCT', 'FG3M', 'FG3A', 'FG3_PCT', 'FTM
2        'FT_PCT', 'OREB', 'DREB', 'REB', 'AST', 'TOV', 'STL', 'BLK', 'BLK
3        'PF', 'PFD', 'PTS','AST_PCT', 'AST_TO',
4        'AST_RATIO', 'OREB_PCT', 'DREB_PCT', 'REB_PCT', 'TM_TOV_PCT', 'EF
5        'TS_PCT', 'PACE', 'PIE']
```

Train test split for adv stats

In [14]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(advs_features, playo
```

Train test split for boxscore stats

In [16]:
```
X_train_all, X_test_all, y_train_all, y_test_all = train_test_split(all_
```

In [17]:
```
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(regu
```

In [18]:
```
from sklearn.preprocessing import StandardScaler
```

In [12]:
```
scaler = StandardScaler()
```

In [13]:
```
scaler.fit(regular_features)
```

Out[13]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [14]:
```
scaler.fit(advs_features)
```

Out[14]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [15]:
```
scaled_reg_features = scaler.fit_transform(regular_features)
```

In [16]:
```
scaled_features = scaler.fit_transform(advs_features)
```

In [17]:
```
df_advs_feat = pd.DataFrame(scaled_features,columns = advs_features.colu
df_advs_feat.head()
```

Out[17]:

	NET_RATING	AST_PCT	AST_TO	AST_RATIO	OREB_PCT	DREB_PCT	REB_PCT	TM_TOV_PCT
0	0.288789	0.787895	1.491675	1.694466	-2.367891	1.356987	-0.483617	-0.738536
1	-0.403058	-0.215469	1.335452	0.137441	-2.847748	1.390944	-1.692364	-2.093133
2	-0.870523	-0.326953	0.710559	-0.096113	-1.984004	2.070066	-0.738090	-1.455676
3	0.027009	-0.237766	0.762633	0.137441	-0.096564	0.813689	0.597894	-0.977582
4	-0.590044	-0.639111	0.189815	-0.329666	-0.160545	0.372260	-0.101907	-0.658854

In [18]:
```
df_reg_feat = pd.DataFrame(scaled_reg_features,columns = regular_feature
df_reg_feat.head()
```

Out[18]:

	FGM	FGA	FG_PCT	FG3M	FG3A	FG3_PCT	FTM	FTA	FT_PCT	-
0	2.763072	2.007603	1.722705	1.691085	1.747216	0.343119	-1.207623	-1.342131	0.518628	-
1	0.925645	1.598269	-0.320463	1.319840	1.497990	-0.202280	-0.909285	-0.827732	-0.177695	-
2	0.619407	1.251910	-0.444292	0.948594	0.963936	0.252219	-0.610948	-0.864475	0.645233	-
3	0.313169	0.307292	0.112936	0.252510	0.162854	0.706718	-1.058454	-1.011446	-0.146044	-
4	0.262130	0.181343	0.236764	-0.304358	-0.442408	0.843067	-0.163441	-0.350077	0.423675	-

5 rows × 21 columns

In [85]:
```
X_adv = df_advs_feat
```

In [86]:
```
X_reg = df_reg_feat
```

In [87]:
```
y_adv = playoff_adv_east['PLAYOFFS']
```

In [88]:
```
y_reg = playoff_reg_east['PLAYOFFS']
```

In [89]:
```
X_adv = X_adv.as_matrix()
y_adv = y_adv.as_matrix()
```

In [ ]:
```

```

In [ ]:
```

```

In [25]:
```
#import tensorflow.contrib.learn.python as learn
```

In [ ]:
```

```

In [18]:
```
from sklearn.metrics import classification_report
```

In [19]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [20]:
```
rfc = RandomForestClassifier(n_estimators = 75)
```

In [21]:
```
rfc_reg = RandomForestClassifier(n_estimators=75)
```

In [22]:
```
1 rfc.fit(X_train_all,y_train_all)
```

Out[22]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=75, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [23]:
```
1 rfc_reg.fit(X_train_reg,y_train_reg)
```

Out[23]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=75, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [24]:
```
1 rfc_pred_reg = rfc_reg.predict(X_test_reg)
```

In [25]:
```
1 rfc_pred = rfc.predict(X_test_all)
```

In [26]:
```
1 importances = rfc.feature_importances_
```

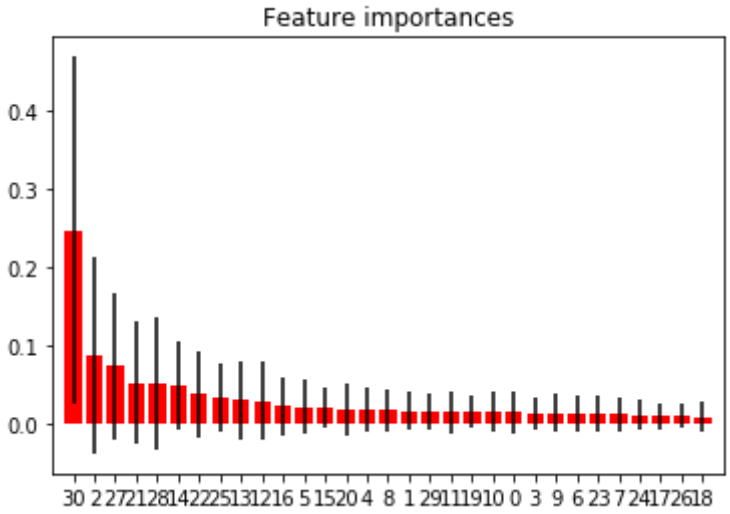
In [27]:
```
1 importances
```

Out[27]:
```
array([0.01419667, 0.01660842, 0.08733524, 0.01363501, 0.01752271,
       0.02124341, 0.0130184 , 0.01152973, 0.01676331, 0.0133985 ,
       0.01476988, 0.01506027, 0.02841407, 0.02983615, 0.04836608,
       0.02043337, 0.02221032, 0.0101104 , 0.00842054, 0.0148117 ,
       0.01814206, 0.0522356 , 0.03775021, 0.012816  , 0.01070785,
       0.03377444, 0.00965721, 0.07309017, 0.05203006, 0.01540433,
       0.24670787])
```

In [33]:
```
std = np.std([tree.feature_importances_ for tree in rfc.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X_train_all.shape[1]):
    print("%d. %s (%f)" % (f + 1, feature_names[indices[f]], importances

# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X_train_all.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X_train_all.shape[1]), indices)
plt.xlim([-1, X_train_all.shape[1]])

```

```
Feature ranking:
1. PIE (0.246708)
2. FG_PCT (0.087335)
3. EFG_PCT (0.073090)
4. AST_TO (0.052236)
5. TS_PCT (0.052030)
6. STL (0.048366)
7. AST_RATIO (0.037750)
8. REB_PCT (0.033774)
9. TOV (0.029836)
10. AST (0.028414)
11. BLKA (0.022210)
12. FG3_PCT (0.021243)
13. BLK (0.020433)
14. AST_PCT (0.018142)
15. FG3A (0.017523)
16. FT_PCT (0.016763)
17. FGA (0.016608)
18. PACE (0.015404)
19. REB (0.015060)
20. PTS (0.014812)
21. DREB (0.014770)
22. FGM (0.014197)
23. FG3M (0.013635)
24. OREB (0.013398)
25. FTM (0.013018)
26. OREB_PCT (0.012816)
27. FTA (0.011530)
28. DREB_PCT (0.010708)
29. PF (0.010110)
30. TM_TOV_PCT (0.009657)
31. PFD (0.008421)
```

Out[33]: (-1, 31)



In [ ]:
```
1
```

Predictions of Western conference playoff teams merged stats with random forest classifier

In [29]:
```
1 print(classification_report(y_test_all,rfc_pred))
```

```
             precision    recall  f1-score   support

          0       0.90      0.88      0.89        59
          1       0.90      0.88      0.89        59

avg / total       0.90      0.88      0.89       118
```

Predictions of Western conference playoff teams (advanced stats) with random forest classifier

In [26]:
```
1 print(classification_report(y_test,rfc_pred))
```

```
             precision    recall  f1-score   support

          0       0.92      0.92      0.92        39
          1       0.95      0.95      0.95        59

avg / total       0.94      0.94      0.94        98
```

Predictions of Western conference playoff teams (boxscorestats) with random forest classifier

In [27]:
```
print(classification_report(y_test,rfc_pred_reg))
```

```
             precision    recall  f1-score   support

          0       0.89      0.87      0.88        39
          1       0.92      0.93      0.92        59

avg / total       0.91      0.91      0.91        98
```

Predictions of Eastern conference playoff teams (boxscore stats) with random forest classifier

In [70]:
```
print(classification_report(y_test,rfc_pred))
```

```
             precision    recall  f1-score   support

          0       0.95      0.89      0.92        45
          1       0.91      0.96      0.94        54

avg / total       0.93      0.93      0.93        99
```

Predictions of Eastern conference playoff teams (advanced stats) with random forest classifier

In [33]:
```
print(classification_report(y_test,rfc_pred))
```

```
             precision    recall  f1-score   support

          0       0.91      0.89      0.90        45
          1       0.91      0.93      0.92        54

avg / total       0.91      0.91      0.91        99
```

Predictions of western conference playoff teams (advanced stats) with random forest classifier

In [28]:
```
#rfc with advanced stats  fo
print(classification_report(y_test,rfc_pred))
```

```
             precision    recall  f1-score   support

          0       0.96      0.92      0.94        53
          1       0.91      0.96      0.93        45

avg / total       0.94      0.94      0.94        98
```

Predictions of Eastern conference playoff teams (advanced stats) with random forest classifier

In [44]:
```
#rfc with regular stats
print(classification_report(y_test,rfc_pred))
```

```
             precision    recall  f1-score   support

          0       0.95      0.86      0.90        49
          1       0.87      0.96      0.91        50

avg / total       0.91      0.91      0.91        99
```

In [ ]:
```

```

In [42]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [43]:
```
dtree = DecisionTreeClassifier()
```

In [41]:
```
dtree_adv = DecisionTreeClassifier()
```

In [45]:
```
dtree.fit(X_train_all,y_train_all)
```

Out[45]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [43]:
```
dtree_pred = dtree.predict(X_test_reg)
```

In [44]:
```
dtree_adv.fit(X_train,y_train)

```

Out[44]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [45]:
```
dtree_pred_adv = dtree_adv.predict(X_test)
```

In [46]:
```
dtree_pred = dtree.predict(X_test_all)
```

Decision Tree Classifier with merged stats

In [47]:
```
print(classification_report(y_test_all,dtree_pred))
```

```
             precision    recall  f1-score   support

          0       0.89      0.81      0.85        59
          1       0.89      0.81      0.85        59

avg / total       0.89      0.81      0.85       118
```

Decision Tree Classifier with advanced stats

In [47]:
```
print(classification_report(y_test,dtree_pred_adv))
```

```
             precision    recall  f1-score   support

          0       0.85      0.85      0.85        39
          1       0.90      0.90      0.90        59

avg / total       0.88      0.88      0.88        98
```

Decsion tree on Western conference teams with box score stats

In [48]:
```
print(classification_report(y_test,dtree_pred))
```

```
             precision    recall  f1-score   support

          0       0.87      0.87      0.87        39
          1       0.92      0.92      0.92        59

avg / total       0.90      0.90      0.90        98
```

Decision tree advanced stats predictions for eastern conference with advanced stats

In [44]:
```
#dtree with advanced stats
print(classification_report(y_test,dtree_pred))
```

```
             precision    recall  f1-score   support

          0       0.80      0.80      0.80        45
          1       0.83      0.83      0.83        54

avg / total       0.82      0.82      0.82        99
```

decision tree boxscore stats predictions for eastern conference with box score stats

In [57]:
```
1 #dtree with reuglar stats
2 print(classification_report(y_test_reg,dtree_pred))
```

```
             precision    recall  f1-score   support

          0       0.80      0.82      0.81        45
          1       0.85      0.83      0.84        54

avg / total       0.83      0.83      0.83        99
```

In [48]:
```
1 from sklearn.svm import SVC
```

In [64]:
```
1 svc_model = SVC()
```

In [72]:
```
1 svc_model_reg = SVC()
```

In [55]:
```
1 X_train = np.asarray(X_train_all)
```

In [56]:
```
1 y_train = np.asarray(y_train_all)
```

In [65]:
```
1 svc_model.fit(X_train_all,y_train_all)
```

Out[65]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [54]:
```
1 pred_reg = svc_model_reg.predict(X_test_reg)
```

In [66]:
```
1 predictions = svc_model.predict(X_test_all)
```

SVM predictions for playoff west with box score

In [57]:
```
1 print(classification_report(y_test,pred_reg))
```

```
             precision    recall  f1-score   support

          0       0.89      0.82      0.85        39
          1       0.89      0.93      0.91        59

avg / total       0.89      0.89      0.89        98
```

SVM predictions for western conference with adv stats

In [56]:
```
print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.95      0.90      0.92        39
          1       0.93      0.97      0.95        59

avg / total       0.94      0.94      0.94        98
```

SVM predictions for playoff east with adv stats

In [56]:
```
print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.91      0.89      0.90        45
          1       0.91      0.93      0.92        54

avg / total       0.91      0.91      0.91        99
```

SVM predictions for playoff east with box score

In [58]:
```
print(classification_report(y_test,pred_reg))
```

```
             precision    recall  f1-score   support

          0       0.86      0.80      0.83        45
          1       0.84      0.89      0.86        54

avg / total       0.85      0.85      0.85        99
```

Predictions for playoffs with merged stats

In [68]:
```
print(classification_report(y_test_all,predictions))
```

```
             precision    recall  f1-score   support

          0       0.73      0.77      0.75        87
          1       0.81      0.77      0.79       109

avg / total       0.77      0.77      0.77       196
```

In [58]:
```
from sklearn.linear_model import LogisticRegression
```

playoff predictions for Western conference with boxscore stats using logistic regression

In [71]:
```
1
2 regular_model = LogisticRegression()
3 regular_model.fit(X_train_reg, y_train_reg)
4 predictions = regular_model.predict(X_test_reg)
5
6 print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.90      0.95      0.92        39
          1       0.96      0.93      0.95        59

avg / total       0.94      0.94      0.94        98
```

playoff predictions for Western conference with advanced stats using logistic regression

In [73]:
```
1 adv_model = LogisticRegression()
2 adv_model.fit(X_train, y_train)
3 predictions = adv_model.predict(X_test)
4
5 print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.95      0.95      0.95        39
          1       0.97      0.97      0.97        59

avg / total       0.96      0.96      0.96        98
```

playoff predictions for Eastern conference with boxscore stats using logistic regression

In [78]:
```
1
2 regular_model = LogisticRegression()
3 regular_model.fit(X_train_reg, y_train_reg)
4 predictions = regular_model.predict(X_test_reg)
5
6 print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.90      0.95      0.92        39
          1       0.96      0.93      0.95        59

avg / total       0.94      0.94      0.94        98
```

playoff predictions for Eastern conference with advanced stats using logistic regression

In [79]:
```
adv_model = LogisticRegression()
adv_model.fit(X_train, y_train)
predictions = adv_model.predict(X_test)

print(classification_report(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.95      0.95      0.95        39
          1       0.97      0.97      0.97        59

avg / total       0.96      0.96      0.96        98
```

Prdictions for playoffs based on merged stats

In [69]:
```
adv_model = LogisticRegression()
adv_model.fit(X_train_all, y_train_all)
predictions = adv_model.predict(X_test_all)

print(classification_report(y_test_all,predictions))
```

```
             precision    recall  f1-score   support

          0       0.76      0.86      0.81        87
          1       0.88      0.78      0.83       109

avg / total       0.82      0.82      0.82       196
```

In [77]:
```
test_team =  TeamYearOverYearSplits(1610612740).by_year()[['FGM', 'FGA',
       'FT_PCT', 'OREB', 'DREB', 'REB', 'AST', 'TOV', 'STL', 'BLK', 'BLK
       'PF', 'PFD', 'PTS', 'PLUS_MINUS']]
regular_model.predict(test_team.head(1))
```

Out[77]:
```
array([1], dtype=int64)
```

In [35]:
```
current_predictions_norm = {}
current_predictions_adv = {}
for index, row in team_list.iterrows():
    current = TeamYearOverYearSplits(row['TEAM_ID']).by_year()[['FGM',
       'FT_PCT', 'OREB', 'DREB', 'REB', 'AST', 'TOV', 'STL', 'BLK', 'BLK
       'PF', 'PFD', 'PTS']]
    current_adv = TeamYearOverYearSplits(row['TEAM_ID'],measure_type =
       'AST_RATIO', 'OREB_PCT', 'DREB_PCT', 'REB_PCT', 'TM_TOV_PCT', 'EF
       'TS_PCT', 'PACE', 'PIE']]
    current = pd.concat([current,current_adv],axis = 1)
    current_predictions_norm[row['ABBREVIATION']] = rfc.predict_proba(cu

```

In [49]:
```
for norm in (current_predictions_norm.keys()):
    print(norm)
    print("Probability of missing playoffs::  "+ str(current_predictions
    print("Probability of making playoffs::  "+ str(current_predictions_
    print("------------------------------")
```

```
ATL
Probability of missing playoffs::  65.33333333333333%
Probability of making playoffs::  34.66666666666667%
------------------------------
BOS
Probability of missing playoffs::  20.0%
Probability of making playoffs::  80.0%
------------------------------
CLE
Probability of missing playoffs::  52.0%
Probability of making playoffs::  48.0%
------------------------------
NOP
Probability of missing playoffs::  16.0%
Probability of making playoffs::  84.0%
------------------------------
CHI
Probability of missing playoffs::  76.0%
Probability of making playoffs::  24.0%
```

In [72]:
```
current_predictions_norm = {}
current_predictions_adv = {}
for index, row in team_list.iterrows():
    current = TeamYearOverYearSplits(row['TEAM_ID']).by_year()[['FGM',
       'FT_PCT', 'OREB', 'DREB', 'REB', 'AST', 'TOV', 'STL', 'BLK', 'BLK
       'PF', 'PFD', 'PTS', 'PLUS_MINUS']]
    current_adv = TeamYearOverYearSplits(row['TEAM_ID'],measure_type =
       'AST_RATIO', 'OREB_PCT', 'DREB_PCT', 'REB_PCT', 'TM_TOV_PCT', 'EF
       'TS_PCT', 'PACE', 'PIE']]

    current_predictions_norm[row['ABBREVIATION']] = regular_model.predic
    current_predictions_adv[row['ABBREVIATION']] = adv_model.predict(cur
```

```
---------------------------------------------------------------------------
--
KeyError                                  Traceback (most recent call las
t)
<ipython-input-72-2bf5b27a8148> in <module>()
      9         'PF', 'PFD', 'PTS','AST_PCT', 'AST_TO',
     10         'AST_RATIO', 'OREB_PCT', 'DREB_PCT', 'REB_PCT', 'TM_TOV_PC
T', 'EFG_PCT',
---> 11         'TS_PCT', 'PACE', 'PIE']]
     12     #urrent_predictions_norm[row['ABBREVIATION']] = regular_mode
l.predict(current.head(1))
     13     current_predictions_adv[row['ABBREVIATION']] = adv_model.pred
ict(current_adv.head(1))

c:\users\tkauk\appdata\local\programs\python\python36\lib\site-packages\p
andas\core\frame.py in __getitem__(self, key)
   2131         if isinstance(key, (Series, np.ndarray, Index, list)):
   2132             # either boolean or fancy integer index
-> 2133             return self._getitem_array(key)
   2134         elif isinstance(key, DataFrame):
   2135             return self._getitem_frame(key)

c:\users\tkauk\appdata\local\programs\python\python36\lib\site-packages\p
andas\core\frame.py in _getitem_array(self, key)
   2175             return self._take(indexer, axis=0, convert=False)
   2176         else:
-> 2177             indexer = self.loc._convert_to_indexer(key, axis=1)
   2178             return self._take(indexer, axis=1, convert=True)
   2179

c:\users\tkauk\appdata\local\programs\python\python36\lib\site-packages\p
andas\core\indexing.py in _convert_to_indexer(self, obj, axis, is_setter)
   1267                 if mask.any():
   1268                     raise KeyError('{mask} not in index'
-> 1269                                    .format(mask=objarr[mask]))
   1270
   1271                 return _values_from_object(indexer)

KeyError: "['FGM' 'FGA' 'FG_PCT' 'FG3M' 'FG3A' 'FG3_PCT' 'FTM' 'FTA' 'FT_
PCT' 'OREB'\n 'DREB' 'REB' 'AST' 'TOV' 'STL' 'BLK' 'BLKA' 'PF' 'PFD' 'PT
S'] not in index"
```

In [80]:
```
for norm, adv in zip(current_predictions_norm.keys(),current_predictions
    print("Normal Prediciton: "+norm+" "+str(current_predictions_norm[no
    print("Advanced Prediciton: "+adv+" "+str(current_predictions_adv[ad
    print("------------------------------")
```

```
Normal Prediciton: ATL [0]
Advanced Prediciton: ATL [0]
------------------------------
Normal Prediciton: BOS [1]
Advanced Prediciton: BOS [1]
------------------------------
Normal Prediciton: CLE [1]
Advanced Prediciton: CLE [1]
------------------------------
Normal Prediciton: NOP [1]
Advanced Prediciton: NOP [1]
------------------------------
Normal Prediciton: CHI [0]
Advanced Prediciton: CHI [0]
------------------------------
Normal Prediciton: DAL [0]
Advanced Prediciton: DAL [0]
------------------------------
Normal Prediciton: DEN [1]
Advanced Prediciton: DEN [1]
------------------------------
Normal Prediciton: GSW [1]
Advanced Prediciton: GSW [1]
------------------------------
Normal Prediciton: HOU [1]
Advanced Prediciton: HOU [1]
------------------------------
Normal Prediciton: LAC [0]
Advanced Prediciton: LAC [0]
------------------------------
Normal Prediciton: LAL [0]
Advanced Prediciton: LAL [0]
------------------------------
Normal Prediciton: MIA [1]
Advanced Prediciton: MIA [1]
------------------------------
Normal Prediciton: MIL [1]
Advanced Prediciton: MIL [1]
------------------------------
Normal Prediciton: MIN [1]
Advanced Prediciton: MIN [1]
------------------------------
Normal Prediciton: BKN [0]
Advanced Prediciton: BKN [0]
------------------------------
Normal Prediciton: NYK [0]
Advanced Prediciton: NYK [0]
------------------------------
Normal Prediciton: ORL [0]
Advanced Prediciton: ORL [0]
------------------------------
Normal Prediciton: IND [1]
```

```
Advanced Prediciton: IND [1]
------------------------------
Normal Prediciton: PHI [1]
Advanced Prediciton: PHI [1]
------------------------------
Normal Prediciton: PHX [0]
Advanced Prediciton: PHX [0]
------------------------------
Normal Prediciton: POR [1]
Advanced Prediciton: POR [1]
------------------------------
Normal Prediciton: SAC [0]
Advanced Prediciton: SAC [0]
------------------------------
Normal Prediciton: SAS [1]
Advanced Prediciton: SAS [1]
------------------------------
Normal Prediciton: OKC [1]
Advanced Prediciton: OKC [1]
------------------------------
Normal Prediciton: TOR [1]
Advanced Prediciton: TOR [1]
------------------------------
Normal Prediciton: UTA [1]
Advanced Prediciton: UTA [1]
------------------------------
Normal Prediciton: MEM [0]
Advanced Prediciton: MEM [0]
------------------------------
Normal Prediciton: WAS [1]
Advanced Prediciton: WAS [1]
------------------------------
Normal Prediciton: DET [0]
Advanced Prediciton: DET [0]
------------------------------
Normal Prediciton: CHA [0]
Advanced Prediciton: CHA [0]
------------------------------
```

In [ ]:
```
1
```

Predictions for this year's playoff teams based on eastern conference models for regular and advanced stats

In [72]:
```
for norm, adv in zip(current_predictions_norm.keys(),current_predictions
    print("Normal Prediciton: "+norm+" "+str(current_predictions_norm[no
    print("Advanced Prediciton: "+adv+" "+str(current_predictions_adv[ad
    print("------------------------------")
```

```
Normal Prediciton: ATL [0]
Advanced Prediciton: ATL [0]
------------------------------
Normal Prediciton: BOS [1]
Advanced Prediciton: BOS [1]
------------------------------
Normal Prediciton: CLE [1]
Advanced Prediciton: CLE [1]
------------------------------
Normal Prediciton: NOP [1]
Advanced Prediciton: NOP [1]
------------------------------
Normal Prediciton: CHI [0]
Advanced Prediciton: CHI [0]
------------------------------
Normal Prediciton: DAL [0]
Advanced Prediciton: DAL [0]
------------------------------
Normal Prediciton: DEN [1]
Advanced Prediciton: DEN [1]
------------------------------
Normal Prediciton: GSW [1]
Advanced Prediciton: GSW [1]
------------------------------
Normal Prediciton: HOU [1]
Advanced Prediciton: HOU [1]
------------------------------
Normal Prediciton: LAC [1]
Advanced Prediciton: LAC [1]
------------------------------
Normal Prediciton: LAL [0]
Advanced Prediciton: LAL [0]
------------------------------
Normal Prediciton: MIA [1]
Advanced Prediciton: MIA [1]
------------------------------
Normal Prediciton: MIL [1]
Advanced Prediciton: MIL [1]
------------------------------
Normal Prediciton: MIN [1]
Advanced Prediciton: MIN [1]
------------------------------
Normal Prediciton: BKN [0]
Advanced Prediciton: BKN [0]
------------------------------
Normal Prediciton: NYK [0]
Advanced Prediciton: NYK [0]
------------------------------
Normal Prediciton: ORL [0]
Advanced Prediciton: ORL [0]
------------------------------
Normal Prediciton: IND [1]
```

```
Advanced Prediciton: IND [1]
------------------------------
Normal Prediciton: PHI [1]
Advanced Prediciton: PHI [1]
------------------------------
Normal Prediciton: PHX [0]
Advanced Prediciton: PHX [0]
------------------------------
Normal Prediciton: POR [1]
Advanced Prediciton: POR [1]
------------------------------
Normal Prediciton: SAC [0]
Advanced Prediciton: SAC [0]
------------------------------
Normal Prediciton: SAS [1]
Advanced Prediciton: SAS [1]
------------------------------
Normal Prediciton: OKC [1]
Advanced Prediciton: OKC [1]
------------------------------
Normal Prediciton: TOR [1]
Advanced Prediciton: TOR [1]
------------------------------
Normal Prediciton: UTA [1]
Advanced Prediciton: UTA [1]
------------------------------
Normal Prediciton: MEM [0]
Advanced Prediciton: MEM [0]
------------------------------
Normal Prediciton: WAS [1]
Advanced Prediciton: WAS [1]
------------------------------
Normal Prediciton: DET [0]
Advanced Prediciton: DET [0]
------------------------------
Normal Prediciton: CHA [1]
Advanced Prediciton: CHA [0]
------------------------------
```

In [15]:
```
from sklearn.linear_model import LinearRegression
```

In [16]:
```
lm_all = LinearRegression()
```

In [17]:
```
lm_reg = LinearRegression()
```

In [18]:
```
lm = LinearRegression()
```

In [19]:
```
lm_reg.fit(X_train_reg,y_train_reg)
```

Out[19]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fal
se)
```

In [20]:
```
1 lm_all.fit(X_train_all,y_train_all)
```

Out[20]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fal
se)
```

In [21]:
```
1 lm.fit(X_train,y_train)
```

Out[21]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fal
se)
```

In [17]:
```
1 print('Coefficients: \n', lm.coef_)
```

```
Coefficients:
 [ 2.32480251e-02  5.77254111e-01 -1.45140801e-01 -8.94242769e-03
 -2.04116074e-01 -5.38752156e-01  1.25100309e+00 -2.29979850e+00
  1.32022526e+00 -9.63187393e-01 -8.52798954e-04  7.34191404e-01]
```

In [22]:
```
1 pred_reg = lm_reg.predict(X_test_reg)
```

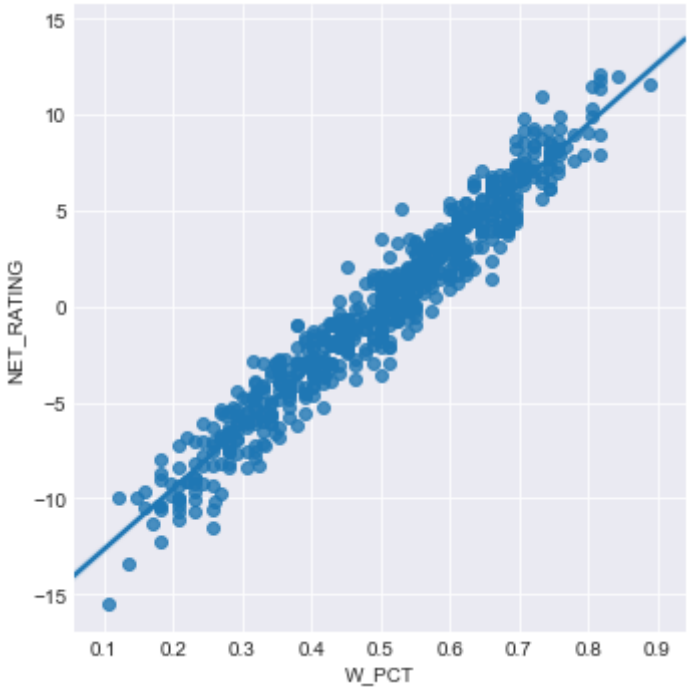
In [23]:
```
1 pred_all = lm_all.predict(X_test_all)
```

In [24]:
```
1 predictions = lm.predict( X_test)
```

In [25]:
```
1 sns.set_style('darkgrid')
```

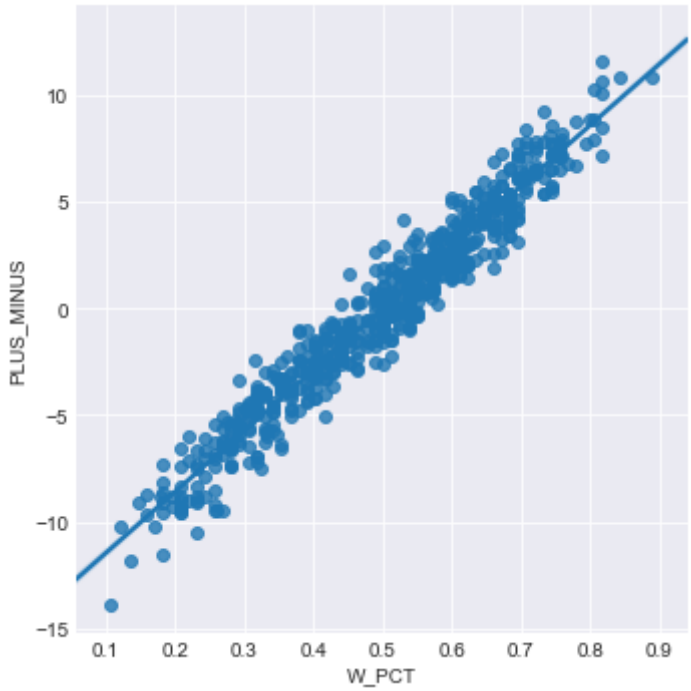
In [24]:
```
1 sns.lmplot(x="W_PCT", y="NET_RATING", data=advs_stats)
```

Out[24]:
```
<seaborn.axisgrid.FacetGrid at 0x19c52a07518>
```



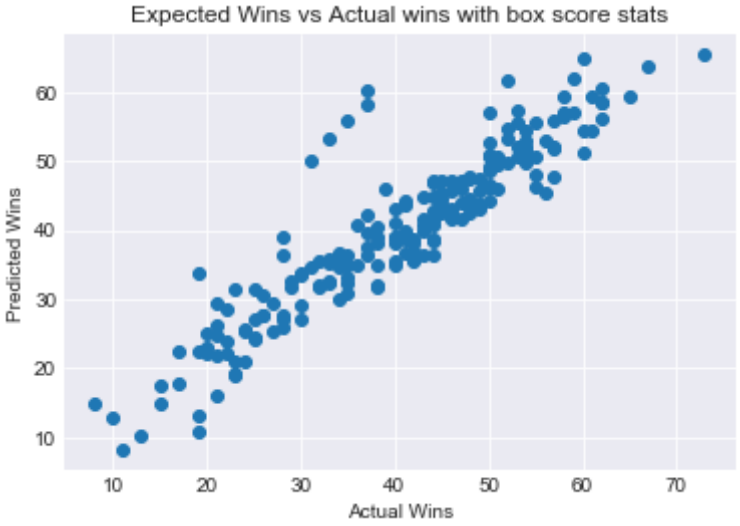
In [25]:
```
1 sns.lmplot(x= "W_PCT", y="PLUS_MINUS", data=regular_stats)
```

Out[25]: <seaborn.axisgrid.FacetGrid at 0x19c52a2ad30>



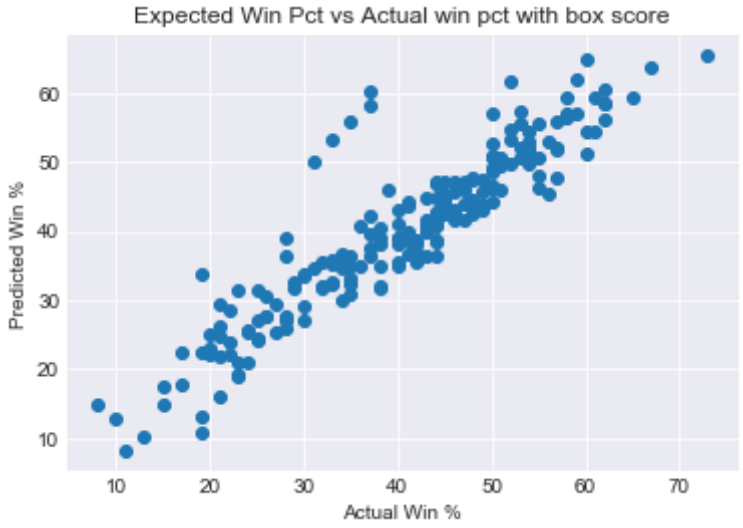
In [78]:
```
1 plt.scatter(y_test,pred_reg)
2 plt.xlabel('Actual Wins')
3 plt.ylabel('Predicted Wins')
4 plt.title('Expected Wins vs Actual wins with box score stats')
```

Out[78]: Text(0.5,1,'Expected Wins vs Actual wins with box score stats')



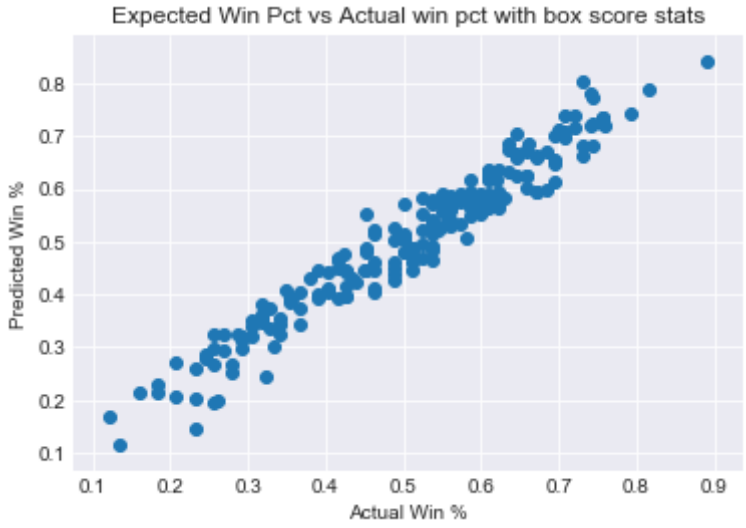
In [79]:
```
1 plt.scatter(y_test,pred_reg)
2 plt.xlabel('Actual Win %')
3 plt.ylabel('Predicted Win %')
4 plt.title('Expected Win Pct vs Actual win pct with box score')
```

Out[79]: Text(0.5,1,'Expected Win Pct vs Actual win pct with box score')



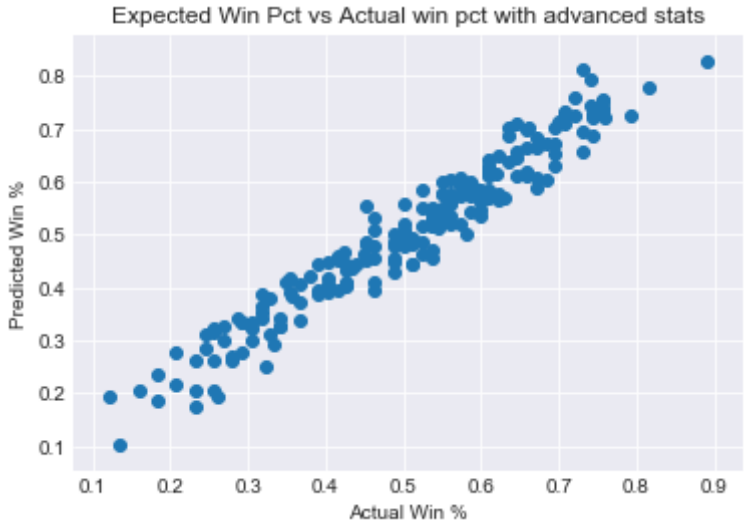
In [26]:
```
1 plt.scatter(y_test,pred_reg)
2 plt.xlabel('Actual Win %')
3 plt.ylabel('Predicted Win %')
4 plt.title('Expected Win Pct vs Actual win pct with box score stats')
```

Out[26]: Text(0.5,1,'Expected Win Pct vs Actual win pct with box score stats')



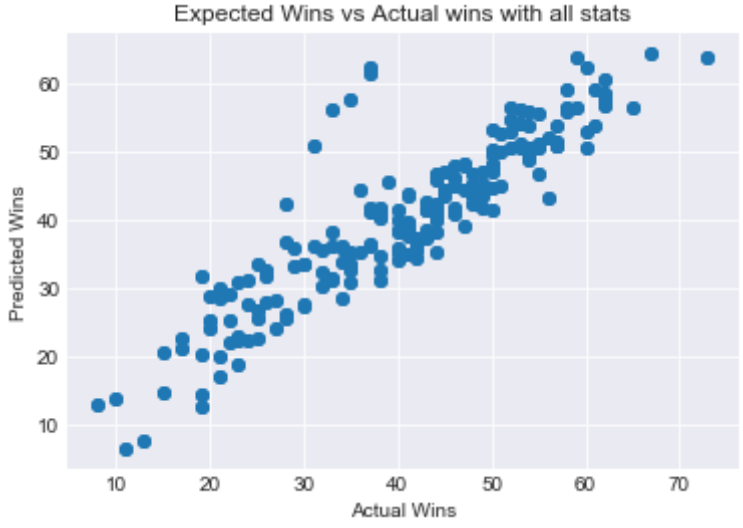
In [33]:
```
plt.scatter(y_test,predictions)
plt.xlabel('Actual Win %')
plt.ylabel('Predicted Win %')
plt.title('Expected Win Pct vs Actual win pct with advanced stats')
```

Out[33]: Text(0.5,1,'Expected Win Pct vs Actual win pct with advanced stats')



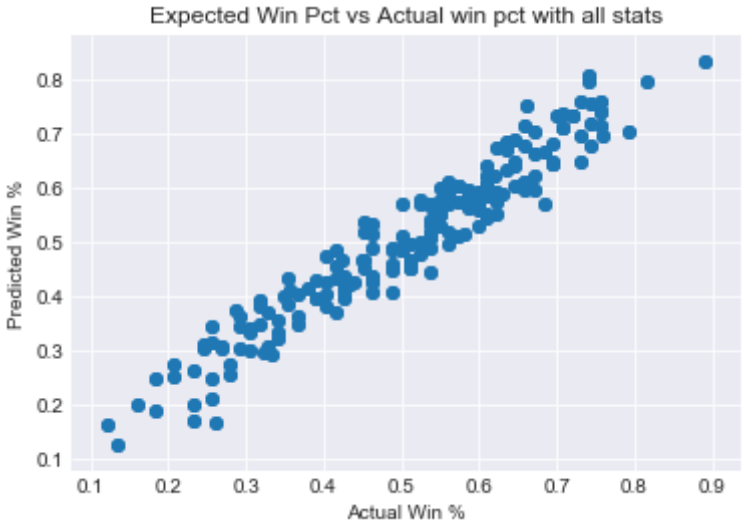
In [80]:
```
plt.scatter(y_test_all,pred_all)
plt.xlabel('Actual Wins')
plt.ylabel('Predicted Wins')
plt.title('Expected Wins vs Actual wins with all stats')
```

Out[80]: Text(0.5,1,'Expected Wins vs Actual wins with all stats')



In [34]:
```
plt.scatter(y_test_all,pred_all)
plt.xlabel('Actual Win %')
plt.ylabel('Predicted Win %')
plt.title('Expected Win Pct vs Actual win pct with all stats')
```

Out[34]: Text(0.5,1,'Expected Win Pct vs Actual win pct with all stats')



MSE for wins regression

advs stats mse

In [81]:
```
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 3.7411290740333767
MSE: 27.67795944216655
RMSE: 5.260984645688158
```

box score mse

In [82]:
```
print('MAE:', metrics.mean_absolute_error(y_test, pred_reg))
print('MSE:', metrics.mean_squared_error(y_test, pred_reg))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, pred_reg)))
```

```
MAE: 3.741133217914493
MSE: 27.746762462532327
RMSE: 5.267519574005618
```

all stats MSE

In [83]:
```
print('MAE:', metrics.mean_absolute_error(y_test_all, pred_all))
print('MSE:', metrics.mean_squared_error(y_test_all, pred_all))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test_all, pred_all)))
```

```
MAE: 4.195726497460205
MSE: 33.97683231065662
RMSE: 5.828964943337421
```

MSE for win percentage regression

In [35]:
```

from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 0.03203795913740231
MSE: 0.0015574397095453154
RMSE: 0.03946441067018885
```

In [36]:
```
print('MAE:', metrics.mean_absolute_error(y_test, pred_reg))
print('MSE:', metrics.mean_squared_error(y_test, pred_reg))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, pred_reg)))
```

```
MAE: 0.031722021453383945
MSE: 0.001456724925432775
RMSE: 0.03816706597883541
```

In [38]:
```
print('MAE:', metrics.mean_absolute_error(y_test_all, pred_all))
print('MSE:', metrics.mean_squared_error(y_test_all, pred_all))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test_all, pred_all)))
```

```
MAE: 0.03499071223680994
MSE: 0.0018304025418126647
RMSE: 0.04278320396852794
```

In [64]:
```
coeffecients = pd.DataFrame(lm.coef_,advs_features.columns)
coeffecients.columns = ['Coeffecient']
coeffecients
```

Out[64]:

	Coeffecient
NET_RATING	0.023248
AST_PCT	0.577254
AST_TO	-0.145141
AST_RATIO	-0.008942
OREB_PCT	-0.204116
DREB_PCT	-0.538752
REB_PCT	1.251003
TM_TOV_PCT	-2.299798
EFG_PCT	1.320225
TS_PCT	-0.963187
PACE	-0.000853
PIE	0.734191

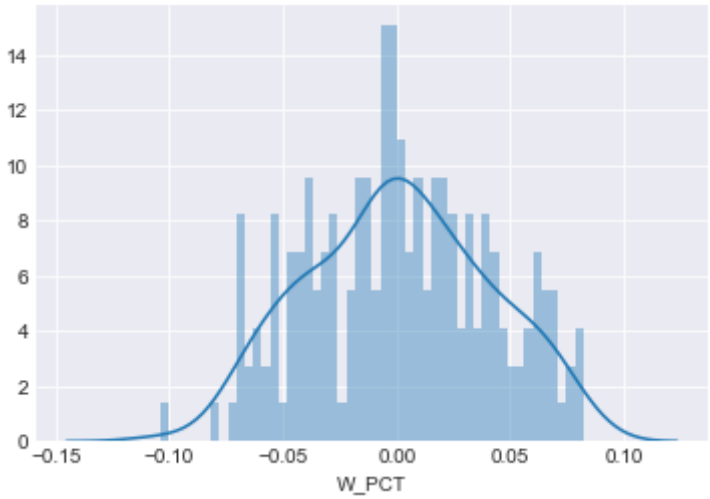
In [65]:
```
coeffecients = pd.DataFrame(lm_reg.coef_,regular_features.columns)
coeffecients.columns = ['Coeffecient']
coeffecients
```

Out[65]:

	Coeffecient
FGM	-0.021721
FGA	-0.014718
FG_PCT	-1.715030
FG3M	-0.040687
FG3A	0.006924
FG3_PCT	0.222312
FTM	-0.038845
FTA	0.011833
FT_PCT	0.403679
OREB	-0.010031
DREB	-0.010700
REB	0.015196
AST	0.001641
TOV	-0.004372
STL	0.001471
BLK	0.003483
BLKA	-0.005350
PF	0.000035
PFD	-0.000163
PTS	0.022706
PLUS_MINUS	0.030324

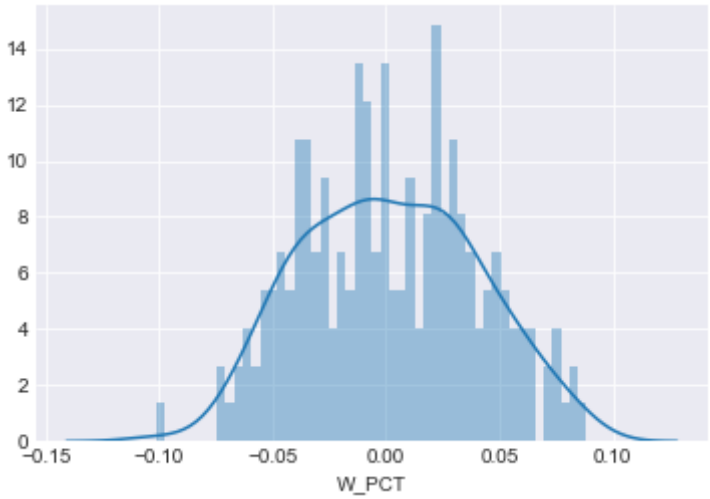
In [66]:
```
1 sns.distplot((y_test-predictions),bins=50);
```

```
c:\users\tkauk\appdata\local\programs\python\python36\lib\site-packages\m
atplotlib\axes\_axes.py:6462: UserWarning: The 'normed' kwarg is deprecat
ed, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



In [67]:
```
1 sns.distplot((y_test-pred_reg),bins=50);
```

```
c:\users\tkauk\appdata\local\programs\python\python36\lib\site-packages\m
atplotlib\axes\_axes.py:6462: UserWarning: The 'normed' kwarg is deprecat
ed, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



In [ ]:
```
1
```