Importing Libraries

In [2]:
```
import pandas as pd
import pandas as pd
import statsmodels.api as sm
from sklearn import datasets, linear_model
from sklearn.metrics import r2_score
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
```

Defining Functions

Create NBA Team List

In [3]:
```
def get_team_list(year):
    df_yr=df_s.loc[df_s['Season']==year].reset_index()
    team_list = df_yr.Tm.unique().tolist()
    return (team_list)

#print (get_team_list(17))
```

Get Current Season's Total Wins and Conference Information

In [4]:
```
def get_stand_conf(year):
    df_yr = df_s.loc[df_s['Season']==year].reset_index()
    df_yr.set_index('Tm', inplace = True)
    stand_dict = {}
    conf_dict={}
    last_y_dict={}
    for team in get_team_list(year):
        conf_dict[team+str(year)] = df_yr.loc[team]['Conf']
        stand_dict[team+str(year)] = df_yr.loc[team]['W']
    return conf_dict, stand_dict

#print (get_stand_conf(17))
```

Get Last Season's Total Wins

In [5]:
```
def get_last_s_win(year):
    df_yr = df_s.loc[df_s['Season']==year-1].reset_index()
    df_yr.set_index('Tm', inplace = True)
    last_y_dict={}
    for team in get_team_list(year):
        last_y_dict[team+str(year)] = df_yr.loc[team]['W']
    return last_y_dict

#print (get_last_s_win(17))
```

Get Current Year NBA Team Rosters at Team

In [6]:
```
def get_team_roster(year, team):
    roster_list = list()
    df_yr=df.loc[df['Season']==year].reset_index()
    for i in range(0, len(df_yr.index)):
        if df_yr.loc[i, 'Tm'] == team:
            roster_list.append(df_yr.loc[i, 'Player'])
    return (roster_list)

#print (get_team_roster(17, 'LAL'))
```

Get Current Season's NBA Team Rosters at Team

In [7]:
```
def get_all_team_roster(year):
    team_dict = {}
    for team in get_team_list(year):
        team_dict[team] = get_team_roster(year, team)
    return (team_dict)

#print (get_all_team_roster(17))
```

Get Last Season's Player Advanced Stats

In [8]:
```
def get_T_minus1 (year, team_list, team):
    pd.options.mode.chained_assignment = None
    df_yr=df.loc[df['Season']==year-1]
    df_team = pd.DataFrame()
    for player in team_list:
        if player in df_yr.values:
            df_yr.loc[df_yr['Player']==player, 'Tm'] = str(team) +str(year)
            df_team = df_team.append(df_yr.loc[df_yr['Player']==player])
    return (df_team)

#for team in get_team_list(17):
#   print (get_T_minus1(17, get_all_team_roster(17)[team], team))
```

Get Last Season's Player Traditional Stats

In [9]:
```
def get_traditional (year, team_list, team):
    df_yr=df_g.loc[df_g['Season']==year-1]
    df_team = pd.DataFrame()
    for player in team_list:
        if player in df_yr.values:
            df_yr.loc[df_yr['Player']==player, 'Tm'] = str(team) +str(year)
            df_team = df_team.append(df_yr.loc[df_yr['Player']==player])
    return (df_team)

#print (get_traditional (16, ['Stephen Curry', 'Lou Williams']))
```

Get Team Level Data by Year

In [10]:
```
def get_df_team_master (year):
    print ('Getting {} data..'.format(year))
    df_team = pd.DataFrame()
    df_team_master = pd.DataFrame()
    team_sum_list = list()
    conf_dict, stand_dict = get_stand_conf(year)
    last_y_dict = get_last_s_win(year)
    for team in get_team_list(year):
        df_team = get_T_minus1(year, get_all_team_roster(year)[team], team)
        sum_list = list()
        sum_list.append(str(team)+str(year))
        for column in df_team.iloc[:,7:27]:
            sum_list.append(df_team[column].sum())
        df_trd = get_traditional (year, get_all_team_roster(year)[team], team)
        for column in df_trd.iloc[:,8:30]:
            sum_list.append(df_trd[column].sum())
        sum_list.append(last_y_dict[team+str(year)])
        sum_list.append(conf_dict[team+str(year)])
        sum_list.append(stand_dict[team+str(year)])
        team_sum_list.append(sum_list)
    df_team_master = df_team_master.append(team_sum_list)
    df_team_master.columns = ['Team', 'PER', 'TS%','3PAr','FTr','ORB%','DRB%','TRB%','AST%','S
'T-1 W', 'Conf', 'Wins']
    return (df_team_master)
```

Normalize Data

In [11]:
```
def normalize_data (year_back):
    df_all = pd.DataFrame()
    for i in range (0,year_back):
        df = get_df_team_master(17-i)
        df_all = df_all.append(df)
    print ('Normalizing data')
    for column in df_all.iloc[:,1:44]:
        df_all[column] = (df_all[column]-df_all[column].mean())/df_all[column].std()
    df_all.to_csv('nba_norm5.csv')
    return ('File nba_norm.csv has been created')
```

Run Regression

In [12]:
```
def run_regression (ind, dep):
    X = df_result[ind]
    X = sm.add_constant(X)
    Y = df_result[dep]
    model = sm.OLS(Y,X)
    result = model.fit()
    print (result.summary())
    return (run_vif(ind))
```

Calculate VIF to Check Collinearity

In [13]:
```
def run_vif (var):
    vif_list =list()
    for x in var:
        X = df_result[var].drop(x, axis =1)
        X = sm.add_constant(X)
        Y = df_result[x]
        model = sm.OLS(Y,X)
        result = model.fit()
        vif_list.append('VIF for {}: '.format(x) + str(1/(1-result.rsquared)))
    return (vif_list)
```

Test the Model

In [50]:
```
def run_test (ind, dep, year):
    df_test = df_result.loc[df_result['Team'].str.contains(str(year))]
    X = df_result[ind]
    lm = linear_model.LinearRegression()
    Y = df_result[dep]
    model = lm.fit(X, Y)
    predictions = model.predict(df_test[ind])
    plt.scatter(df_test['Wins'], predictions)
    plt.xlabel('True Values')
    plt.ylabel('Predictions')
    x = np.random.rand(100)
    y = np.random.rand(100)
    colors = np.random.rand(100)
    plt.plot( [15,70],[15,70] )
    plt.show()
    df_test['Pred'] = predictions
    df_test['Residual'] = df_test['Pred'] - df_test['Wins']
    plt.hist(df_test['Residual'], bins = 10)
    plt.xlabel('Residual Error')
    plt.ylabel('Frequency')
    plt.show()
    plt.scatter(df_test['ID'], df_test['Residual'])
    plt.xlabel('Observations across Teams')
    plt.ylabel('Residual Error')
    plt.show()
    stats.probplot(df_test['Residual'], plot = plt)
    plt.show()
    return ()
```

Plot Residual

In [15]:
```
def plot_residual (ind, dep):
    X = df_result[ind]
    lm = linear_model.LinearRegression()
    Y = df_result[dep]
    model = lm.fit(X, Y)
    predictions = model.predict(df_result[ind])
    df_result['Pred'] = predictions
    df_result['Residual'] = df_result['Pred'] - df_result['Wins']
    plt.hist(df_result['Residual'])
    plt.xlabel('Residual Error')
    plt.ylabel('Frequency')
    plt.show()
    plt.scatter(df_result['ID'], df_result['Residual'])
    plt.xlabel('Observations across Seasons')
    plt.ylabel('Residual Error')
    plt.show()
    stats.probplot(df_result['Residual'], plot = plt)
    plt.show()
    plt.scatter(df_result['Pred'], df_result['Residual'])
    plt.xlabel('Prediction')
    plt.ylabel('Residual Error')
    plt.show()
    return ()
```

Predict the Total Wins in 2017-2018 Season and Compare with ESPN's Prediction

In [16]:
```
def predict_18 (ind, dep):
    df_18 = pd.read_csv('nba_18.csv')
    X = df_result[ind]
    lm = linear_model.LinearRegression()
    Y = df_result[dep]
    model = lm.fit(X, Y)
    predictions = model.predict(df_18[ind])
    df_18['Prediction'] = predictions
    df_18['espn'] = df_espn['1718']
    df_18['+/-'] = df_18['Prediction'] - df_18['espn']
    plt.scatter(df_18['Prediction'], df_18['espn'])
    plt.xlabel('Our Prediction')
    plt.ylabel('ESPN Prediction')
    x = np.random.rand(100)
    y = np.random.rand(100)
    colors = np.random.rand(100)
    plt.plot( [20,70],[20,70] )
    plt.show()
    print ('R-squared socre with ESPN prediction: ', r2_score(df_18['Prediction'], df_18['espn
    return (df_18[['Team', 'Prediction', 'espn', '+/-']])
```

Reading Data Sources and Creating Datasets

Data source: NBA offical stats for the players from NBA Reference website https://www.basketball-reference.com/ (https://www.basketball-reference.com/)

In [211]:
```
df = pd.read_csv('player.csv')
df_s = pd.read_csv('standing.csv')
df_g = pd.read_csv('1617pg.csv')

normalize_data(6)
```

```
Getting 17 data..
Getting 16 data..
Getting 15 data..
Getting 14 data..
Getting 13 data..
Getting 12 data..
Normalizing data
```

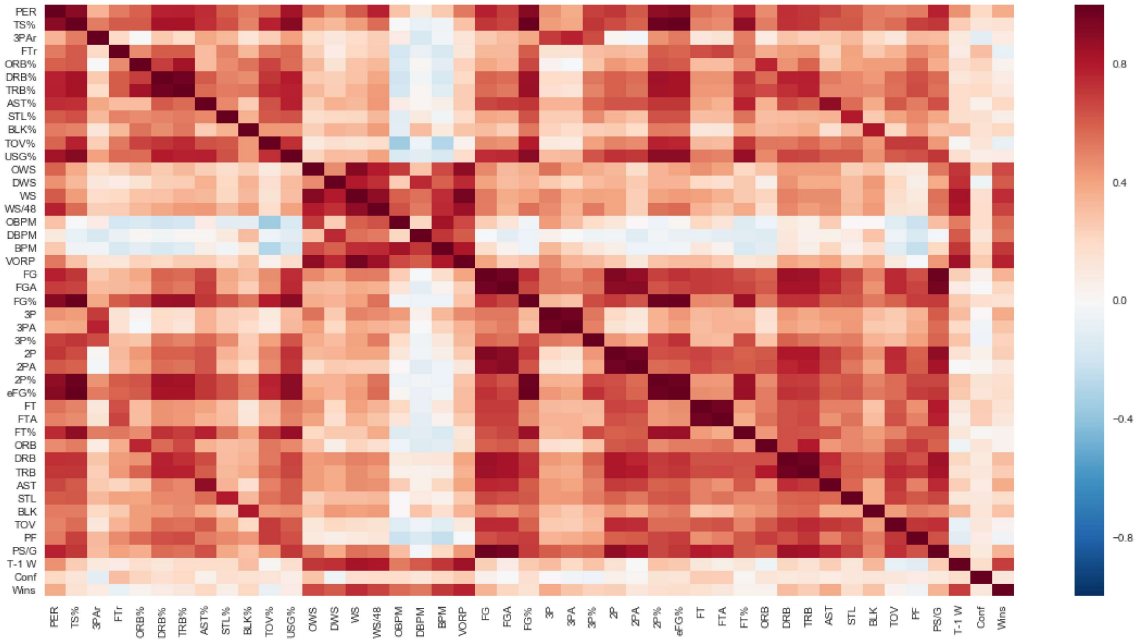
Out[211]:
```
'File nba_norm.csv has been created'
```

Running Regression

In [17]:
```
df_result = pd.read_csv('nba_norm.csv')
```

In [20]:
```
import seaborn as sns
%matplotlib inline
corr = df_result.iloc[:,2:47].corr()
fig, ax = plt.subplots(figsize=(20,10))
# plot the heatmap
sns.heatmap(corr)
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x265efae1860>



Model with All Variables

In [21]:
```
run_regression(df_result.iloc[:,2:46].columns.values.tolist(),'Wins')
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Wins   R-squared:                       0.765
Model:                            OLS   Adj. R-squared:                  0.536
Method:                 Least Squares   F-statistic:                     3.334
Date:                Mon, 11 Dec 2017   Prob (F-statistic):           4.97e-05
Time:                        21:11:22   Log-Likelihood:                -291.09
No. Observations:                  90   AIC:                             672.2
Df Residuals:                      45   BIC:                             784.7
Df Model:                          44
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         40.9654      1.755     23.339      0.000      37.430      44.501
PER           -0.4653     16.150     -0.029      0.977     -32.993      32.062
TS%          -25.7965     35.228     -0.732      0.468     -96.750      45.157
3PAr           3.7820      7.488      0.505      0.616     -11.300      18.864
FTr           -2.4151      3.271     -0.738      0.464      -9.004       4.174
ORB%          -9.1435     14.730     -0.621      0.538     -38.811      20.524
DRB%          -4.4712     33.446     -0.134      0.894     -71.835      62.892
TRB%          12.2870     42.650      0.288      0.775     -73.614      98.188
AST%          -8.4392      5.105     -1.653      0.105     -18.721       1.843
STL%           0.7296      4.325      0.169      0.867      -7.981       9.440
BLK%          -0.7886      3.566     -0.221      0.826      -7.971       6.394
TOV%           7.1157      5.282      1.347      0.185      -3.522      17.753
USG%           5.9313      8.213      0.722      0.474     -10.611      22.474
OWS           -0.5157     55.983     -0.009      0.993    -113.271     112.239
DWS            5.6862     38.139      0.149      0.882     -71.129      82.502
WS            -7.8469     81.669     -0.096      0.924    -172.337     156.643
WS/48          6.9364      9.774      0.710      0.482     -12.750      26.623
OBPM         -60.8419     63.579     -0.957      0.344    -188.897      67.213
DBPM         -44.7353     49.262     -0.908      0.369    -143.954      54.483
BPM           83.5189     88.973      0.939      0.353     -95.682     262.720
VORP           2.7278      6.838      0.399      0.692     -11.044      16.500
FG           -39.4625     70.768     -0.558      0.580    -181.997     103.072
FGA          119.6732     78.312      1.528      0.133     -38.055     277.402
FG%           17.8514     50.546      0.353      0.726     -83.954     119.657
3P           -17.8876     23.107     -0.774      0.443     -64.428      28.653
3PA          -57.8736     34.025     -1.701      0.096    -126.404      10.656
3P%           -1.2784      2.655     -0.481      0.633      -6.627       4.070
2P           -14.5768     43.770     -0.333      0.741    -102.735      73.581
2PA         -119.3206     69.988     -1.705      0.095    -260.284      21.643
2P%           -1.6719      9.854     -0.170      0.866     -21.518      18.174
eFG%          -3.3773     57.076     -0.059      0.953    -118.333     111.579
FT           -16.3366     20.094     -0.813      0.421     -56.809      24.135
FTA           -1.1892      8.471     -0.140      0.889     -18.252      15.873
FT%           -0.3882      4.400     -0.088      0.930      -9.250       8.473
ORB           16.6793     13.893      1.201      0.236     -11.303      44.662
DRB           24.0009     38.265      0.627      0.534     -53.068     101.070
TRB          -35.9989     48.973     -0.735      0.466    -134.636      62.638
AST           12.5797      5.042      2.495      0.016       2.425      22.734
STL            0.0207      4.427      0.005      0.996      -8.896       8.938
BLK            0.1494      3.444      0.043      0.966      -6.788       7.086
TOV          -13.2262      4.998     -2.646      0.011     -23.292      -3.160
PF             1.0670      3.182      0.335      0.739      -5.342       7.476
PS/G         102.9054     78.559      1.310      0.197     -55.320     261.131
T-1 W         -0.8433      2.805     -0.301      0.765      -6.492       4.806
Conf           0.0693      2.995      0.023      0.982      -5.963       6.101
==============================================================================
Omnibus:                        0.900   Durbin-Watson:                   1.963
Prob(Omnibus):                  0.638   Jarque-Bera (JB):                0.905
Skew:                           0.070   Prob(JB):                        0.636
Kurtosis:                       2.529   Cond. No.                         637.
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Out[21]:
```
['VIF for PER: 307.535247858',
 'VIF for TS%: 1463.3464948',
 'VIF for 3PAr: 66.1163598531',
 'VIF for FTr: 12.6191072077',
```

```
'VIF for ORB%: 255.832666682',
'VIF for DRB%: 1319.02647644',
'VIF for TRB%: 2144.86525408',
'VIF for AST%: 30.7299012223',
'VIF for STL%: 22.0548022192',
'VIF for BLK%: 14.9942573191',
'VIF for TOV%: 32.8917085615',
'VIF for USG%: 79.543507908',
'VIF for OWS: 3695.49846119',
'VIF for DWS: 1715.14163427',
'VIF for WS: 7864.69226981',
'VIF for WS/48: 112.651738758',
'VIF for OBPM: 4766.45387303',
'VIF for DBPM: 2861.45162068',
'VIF for BPM: 9334.29328434',
'VIF for VORP: 55.130929234',
'VIF for FG: 5905.27025891',
'VIF for FGA: 7231.41425541',
'VIF for FG%: 3012.63076961',
'VIF for 3P: 629.603046246',
'VIF for 3PA: 1365.10107486',
'VIF for 3P%: 8.31464711428',
'VIF for 2P: 2259.04869274',
'VIF for 2PA: 5775.87271959',
'VIF for 2P%: 114.489328664',
'VIF for eFG%: 3841.18502738',
'VIF for FT: 476.115119622',
'VIF for FTA: 84.622083089',
'VIF for FT%: 22.8259508064',
'VIF for ORB: 227.600525816',
'VIF for DRB: 1726.46909075',
'VIF for TRB: 2828.00985672',
'VIF for AST: 29.9721433508',
'VIF for STL: 23.1123949283',
'VIF for BLK: 13.9878518676',
'VIF for TOV: 29.4537150468',
'VIF for PF: 11.9403377388',
'VIF for PS/G: 7277.04331578',
'VIF for T-1 W: 9.27611848733',
'VIF for Conf: 2.67362776099']
```

Reduced Model (Our Final Model)

In [22]:
```
run_regression(['DBPM', 'VORP', 'FT%', 'AST', 'TOV', 'PS/G'], 'Wins')
```

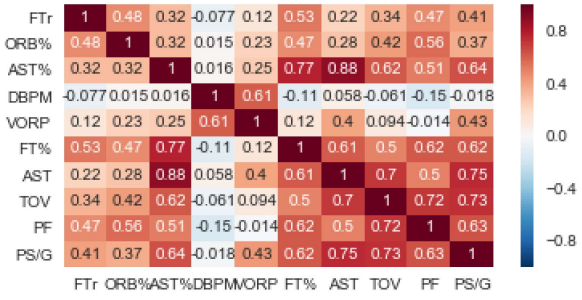
```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Wins   R-squared:                       0.659
Model:                            OLS   Adj. R-squared:                  0.634
Method:                 Least Squares   F-statistic:                     26.71
Date:                Mon, 11 Dec 2017   Prob (F-statistic):           1.71e-17
Time:                        21:11:28   Log-Likelihood:                -307.92
No. Observations:                  90   AIC:                             629.8
Df Residuals:                      83   BIC:                             647.3
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         41.0000      0.813     50.433      0.000      39.383      42.617
DBPM           2.6978      1.171      2.305      0.024       0.369       5.026
VORP           5.5309      1.489      3.714      0.000       2.569       8.493
FT%           -2.5697      1.149     -2.236      0.028      -4.856      -0.284
AST            4.4094      1.470      2.999      0.004       1.485       7.334
TOV           -4.5313      1.504     -3.012      0.003      -7.524      -1.539
PS/G           3.7191      1.779      2.091      0.040       0.181       7.257
==============================================================================
Omnibus:                        0.023   Durbin-Watson:                   2.121
Prob(Omnibus):                  0.989   Jarque-Bera (JB):                0.045
Skew:                           0.023   Prob(JB):                        0.978
Kurtosis:                       2.901   Cond. No.                         5.44
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Out[22]:
```
['VIF for DBPM: 2.0505590773',
 'VIF for VORP: 3.31824388531',
 'VIF for FT%: 1.97663675827',
 'VIF for AST: 3.23480254967',
 'VIF for TOV: 3.38658178022',
 'VIF for PS/G: 4.73396559484']
```

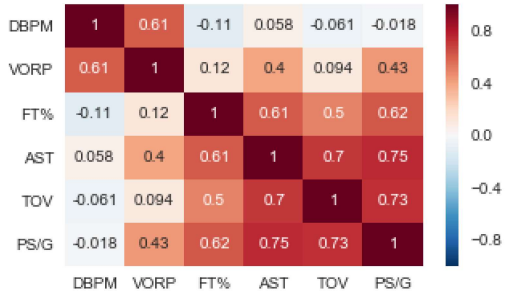
In [25]:
```
fig, ax = plt.subplots(figsize=(6,3))
corrf = df_result[['FTr', 'ORB%', 'AST%', 'DBPM', 'VORP', 'FT%', 'AST', 'TOV', 'PF', 'PS/G']].
# plot the heatmap
sns.heatmap(corrf, annot = True)
```

Out[25]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x265eff33860>
```



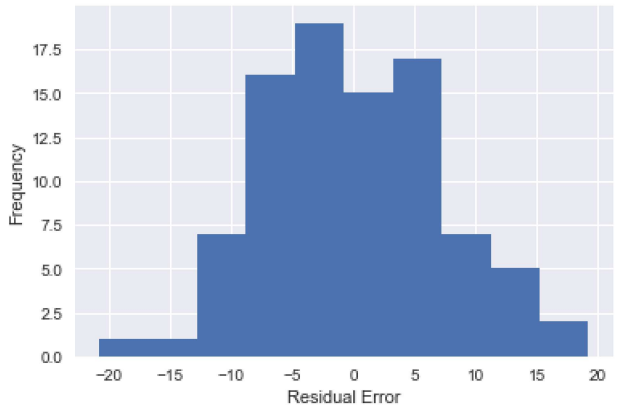
In [24]:
```
fig, ax = plt.subplots(figsize=(5,3))
corrf = df_result[['DBPM', 'VORP', 'FT%', 'AST', 'TOV', 'PS/G']].corr()
# plot the heatmap
sns.heatmap(corrf, annot = True)
```

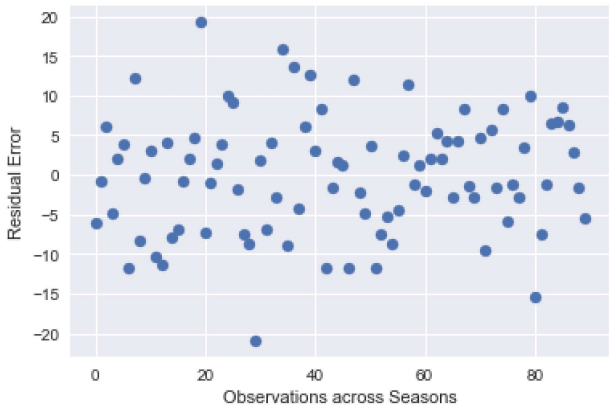
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x265efe38cf8>

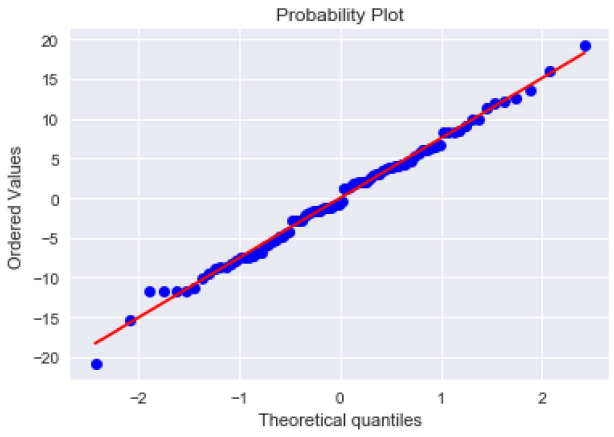


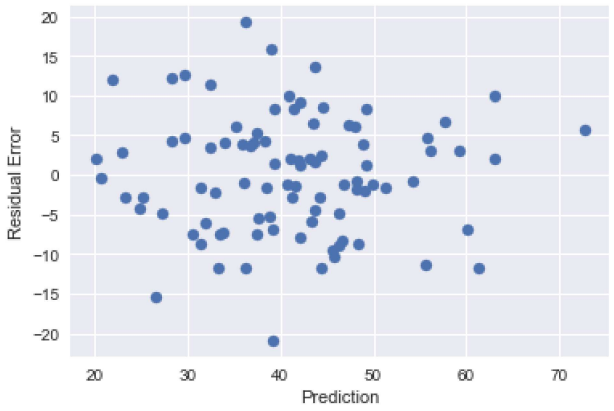
Residual Plot

In [55]:
```
plot_residual (['DBPM', 'VORP', 'FT%', 'AST', 'TOV', 'PS/G'], 'Wins')
```





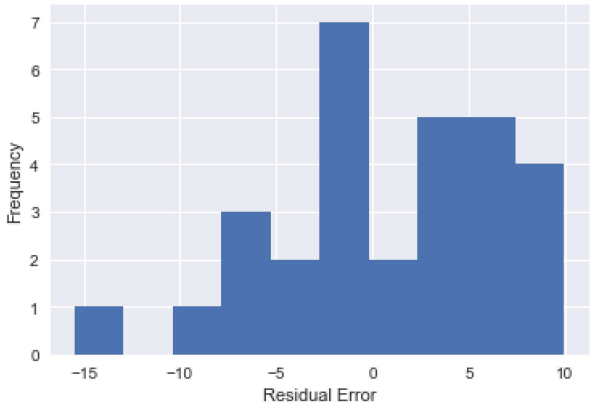


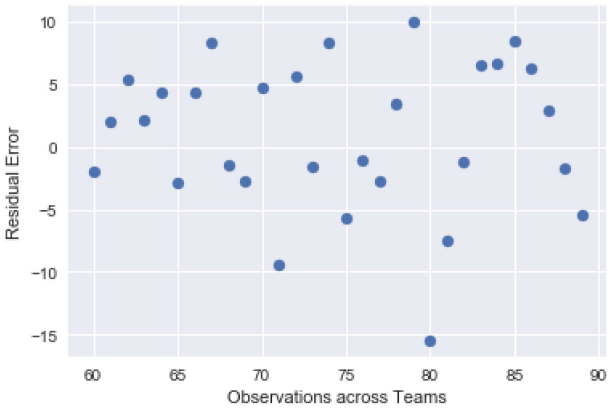


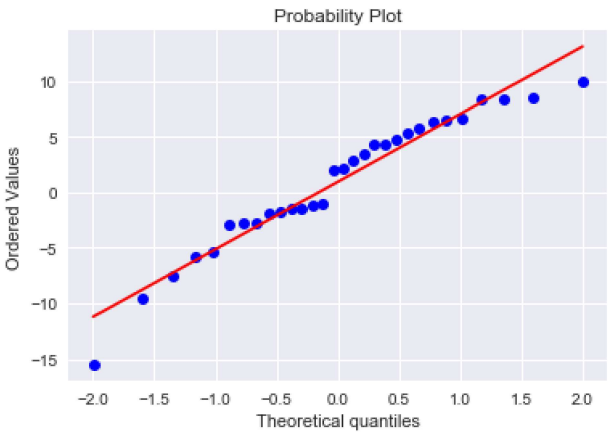
Out[55]: ()

Testing the Model and Compare with ESPN's Prediction

In [52]:
```
run_test(['DBPM', 'VORP', 'FT%', 'AST', 'TOV', 'PS/G'], 'Wins', 17)
df_espn = pd.read_csv('espnPred.csv')
print ('R-Squared Score of our Prediction: ', r2_score(df_espn['1617R'], df_espn['Model1617']))
print ('R-Squared Score of ESPN Prediction: ', r2_score(df_espn['1617R'], df_espn['E1617']))
```

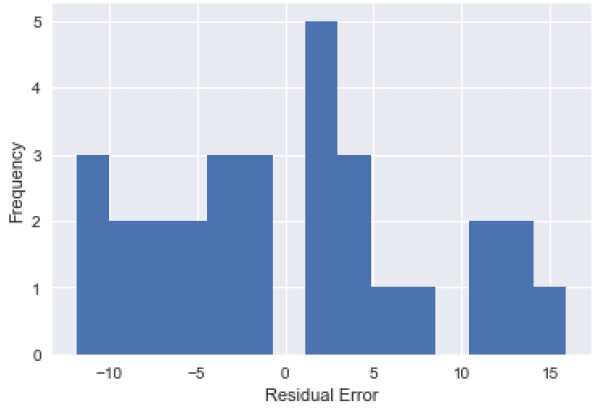


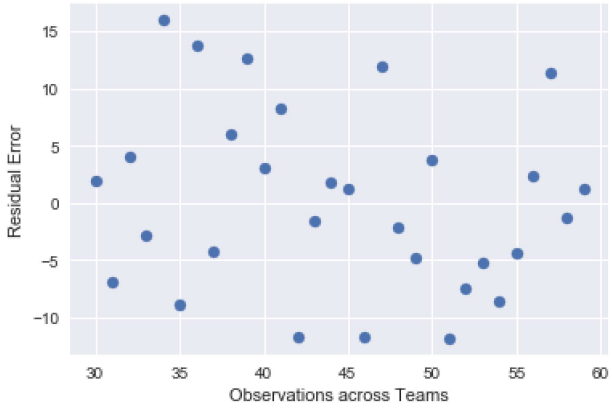


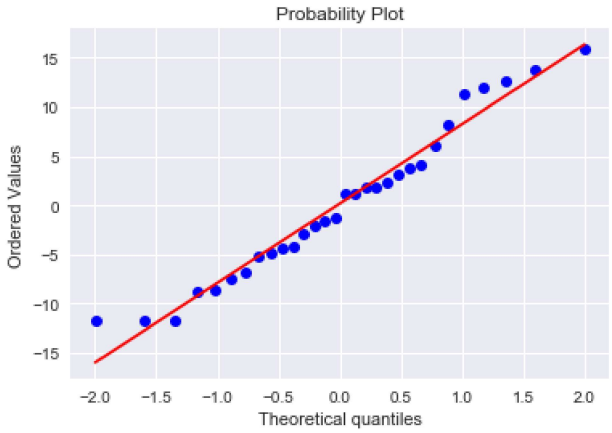



```
R-Squared Score of our Prediction:  0.704392809917
R-Squared Score of ESPN Prediction:  0.805785123967
```

In [48]:
```
run_test(['DBPM', 'VORP', 'FT%', 'AST', 'TOV', 'PS/G'], 'Wins', 16)
print ('R-Squared Score of our Prediction: ', r2_score(df_espn['1516R'], df_espn['Model1516']))
print ('R-Squared Score of ESPN Prediction: ', r2_score(df_espn['1516R'], df_espn['E1516']))
```

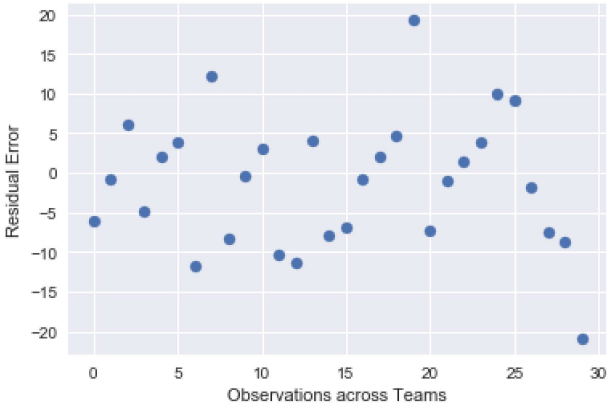


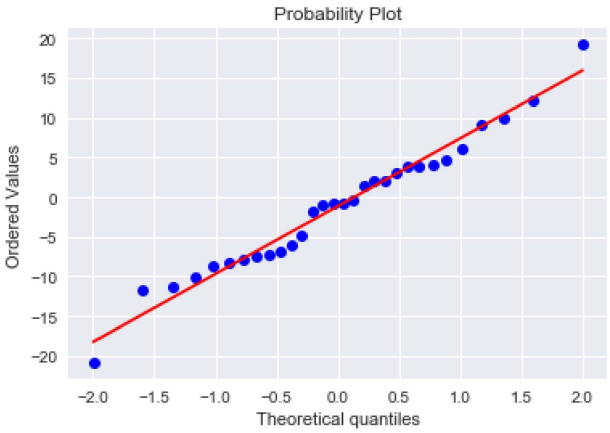





```
R-Squared Score of our Prediction:  0.676810178891
R-Squared Score of ESPN Prediction:  0.606976744186
```

In [54]:
```
run_test(['DBPM', 'VORP', 'FT%', 'AST', 'TOV', 'PS/G'], 'Wins', 15)
print ('R-Squared Score of our Prediction: ', r2_score(df_espn['1415R'], df_espn['Model1415']))
print ('R-Squared Score of ESPN Prediction: ', r2_score(df_espn['1415R'], df_espn['E1415']))
```






```
R-Squared Score of our Prediction:  0.608020171429
R-Squared Score of ESPN Prediction:  0.530666666667
```