Predict NBA player wage with multiple regression model

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
from scipy import stats
import statsmodels.api as sm
import statsmodels.formula.api as smf
import wooldridge as woo
from simple_colors import *
```

Data Description and Motivation

In [2]:
```
# import data
df = woo.dataWoo('nbasal')
df_missing = woo.dataWoo('nbasal')
```

Motivation:

Our goal is to study what would influence NBA players' salaries. The outcome would be helpful for players to improve themselves and for teams to determine how much they should pay based on players' performance.

Data Description:

We found this data from Wooldridge Dataset. This data was collected by Christopher Torrente, who did for school team project while he was in MSU. This is the salary data and the career statistics from The Complete Handbook of Pro Basketball, 1995, edited by Zander Hollander. New York: Signet. The demographic information (marital status, number of children, and so on) was obtained from the teams' 1994-1995 media guides.

Variable Explaination:

marr: =1 if married

wage: annual salary, thousands $

exper: years as professional player

age: age in years

coll: years played in college

games: average games per year

minutes: average minutes per year

guard: =1 if guard

forward: =1 if forward

center: =1 if center

points: points per game

rebounds: rebounds per game

assists: assists per game

draft: draft number

allstar: =1 if ever all star

avgmin: minutes per game

lwage: log(wage)

black: =1 if black

children: =1 if has children

expersq: exper^2

agesq: age^2

marrblck: marr*black

Variable Exploration

In [3]:
```
# explore the data
print(df.info())
print(df.describe())
print(df.isnull().sum())


# We find there are two response variables, wage and lwage, therefore we drop lwage
df.drop('lwage',axis = 1,inplace = True)
total = df.isnull().sum().sort_values(ascending = False)
percent = (df.isnull().sum()/df.count()).sort_values(ascending = False)
missing_data = pd.concat([total,percent],axis = 1,keys = ['total','percent'])
print(missing_data)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 269 entries, 0 to 268
Data columns (total 22 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   marr      269 non-null    int64  
 1   wage      269 non-null    float64
 2   exper     269 non-null    int64  
 3   age       269 non-null    int64  
 4   coll      269 non-null    int64  
 5   games     269 non-null    int64  
 6   minutes   269 non-null    int64  
 7   guard     269 non-null    int64  
 8   forward   269 non-null    int64  
 9   center    269 non-null    int64  
 10  points    269 non-null    float64
 11  rebounds  269 non-null    float64
 12  assists   269 non-null    float64
```

```
 13  draft     240 non-null    float64
 14  allstar   269 non-null    int64
 15  avgmin    269 non-null    float64
 16  lwage     269 non-null    float64
 17  black     269 non-null    int64
 18  children  269 non-null    int64
 19  expersq   269 non-null    int64
 20  agesq     269 non-null    int64
 21  marrblck  269 non-null    int64
dtypes: float64(7), int64(15)
memory usage: 46.4 KB
None
             marr         wage       exper         age        coll  \
count  269.000000   269.000000  269.000000  269.000000  269.000000
mean     0.442379  1423.827511    5.118959   27.394052    3.717472
std      0.497595   999.774073    3.400062    3.391292    0.754410
min      0.000000   150.000000    1.000000   21.000000    0.000000
25%      0.000000   650.000000    2.000000   25.000000    4.000000
50%      0.000000  1186.000000    4.000000   27.000000    4.000000
75%      1.000000  2014.500000    7.000000   30.000000    4.000000
max      1.000000  5740.000000   18.000000   41.000000    4.000000

            games      minutes       guard     forward      center  ...  \
count  269.000000   269.000000  269.000000  269.000000  269.000000  ...
mean    65.724907  1682.193309    0.420074    0.408922    0.171004  ...
std     18.851110   893.327771    0.494491    0.492551    0.377214  ...
min      3.000000    33.000000    0.000000    0.000000    0.000000  ...
25%     57.000000   983.000000    0.000000    0.000000    0.000000  ...
50%     74.000000  1690.000000    0.000000    0.000000    0.000000  ...
75%     79.000000  2438.000000    1.000000    1.000000    0.000000  ...
max     82.000000  3533.000000    1.000000    1.000000    1.000000  ...

          assists      draft     allstar      avgmin       lwage       black  \
count  269.000000  240.00000  269.000000  269.000000  269.000000  269.000000
mean     2.408922   20.20000    0.115242   23.979254    6.952296    0.806691
std      2.092986   18.73582    0.319909    9.731177    0.881376    0.395629
min      0.000000    1.00000    0.000000    2.888889    5.010635    0.000000
25%      0.900000    7.00000    0.000000   16.731340    6.476973    1.000000
50%      1.900000   14.50000    0.000000   24.816900    7.078341    1.000000
75%      3.400000   28.25000    0.000000   33.256100    7.608126    1.000000
max     12.600000  139.00000    1.000000   43.085369    8.655214    1.000000

         children     expersq        agesq   marrblck
count  269.000000  269.000000   269.000000  269.00000
mean     0.345725   37.721190   761.892193    0.33829
std      0.476491   46.537021   195.149406    0.47401
min      0.000000    1.000000   441.000000    0.00000
25%      0.000000    4.000000   625.000000    0.00000
50%      0.000000   16.000000   729.000000    0.00000
75%      1.000000   49.000000   900.000000    1.00000
max      1.000000  324.000000  1681.000000    1.00000

[8 rows x 22 columns]
marr         0
wage         0
exper        0
age          0
coll         0
games        0
minutes      0
guard        0
forward      0
center       0
points       0
rebounds     0
assists      0
draft       29
allstar      0
avgmin       0
```

```
lwage        0
black        0
children     0
expersq      0
agesq        0
marrblck     0
dtype: int64
          total   percent
draft        29  0.120833
marr          0  0.000000
rebounds      0  0.000000
agesq         0  0.000000
expersq       0  0.000000
children      0  0.000000
black         0  0.000000
avgmin        0  0.000000
allstar       0  0.000000
assists       0  0.000000
points        0  0.000000
wage          0  0.000000
center        0  0.000000
forward       0  0.000000
guard         0  0.000000
minutes       0  0.000000
games         0  0.000000
coll          0  0.000000
age           0  0.000000
exper         0  0.000000
marrblck      0  0.000000
```

From the above analysis, we can see that the total number of variables in this dataset is 21. We have 269 observations, where 12% observations miss the value of draft.

In [4]:
```
for i in df.columns:
    print('The unique number of comlumn ' + str(i),'is',len(df[i].unique()))
```

```
The unique number of comlumn marr is 2
The unique number of comlumn wage is 180
The unique number of comlumn exper is 15
The unique number of comlumn age is 18
The unique number of comlumn coll is 5
The unique number of comlumn games is 58
The unique number of comlumn minutes is 254
The unique number of comlumn guard is 2
The unique number of comlumn forward is 2
The unique number of comlumn center is 2
The unique number of comlumn points is 159
The unique number of comlumn rebounds is 91
The unique number of comlumn assists is 72
The unique number of comlumn draft is 60
The unique number of comlumn allstar is 2
The unique number of comlumn avgmin is 265
The unique number of comlumn black is 2
The unique number of comlumn children is 2
The unique number of comlumn expersq is 15
The unique number of comlumn agesq is 18
The unique number of comlumn marrblck is 2
```

In order to use Boruta Algorithm and Mallows Cp, we will need to handle missing values first. We removed obeservations who does not have values for draft. We will explain the reasoning in Question 2.

In [5]:
```
# drop observations where draft is NA
df.dropna(axis = 0,inplace = True)
```

Q1: Variable Selection

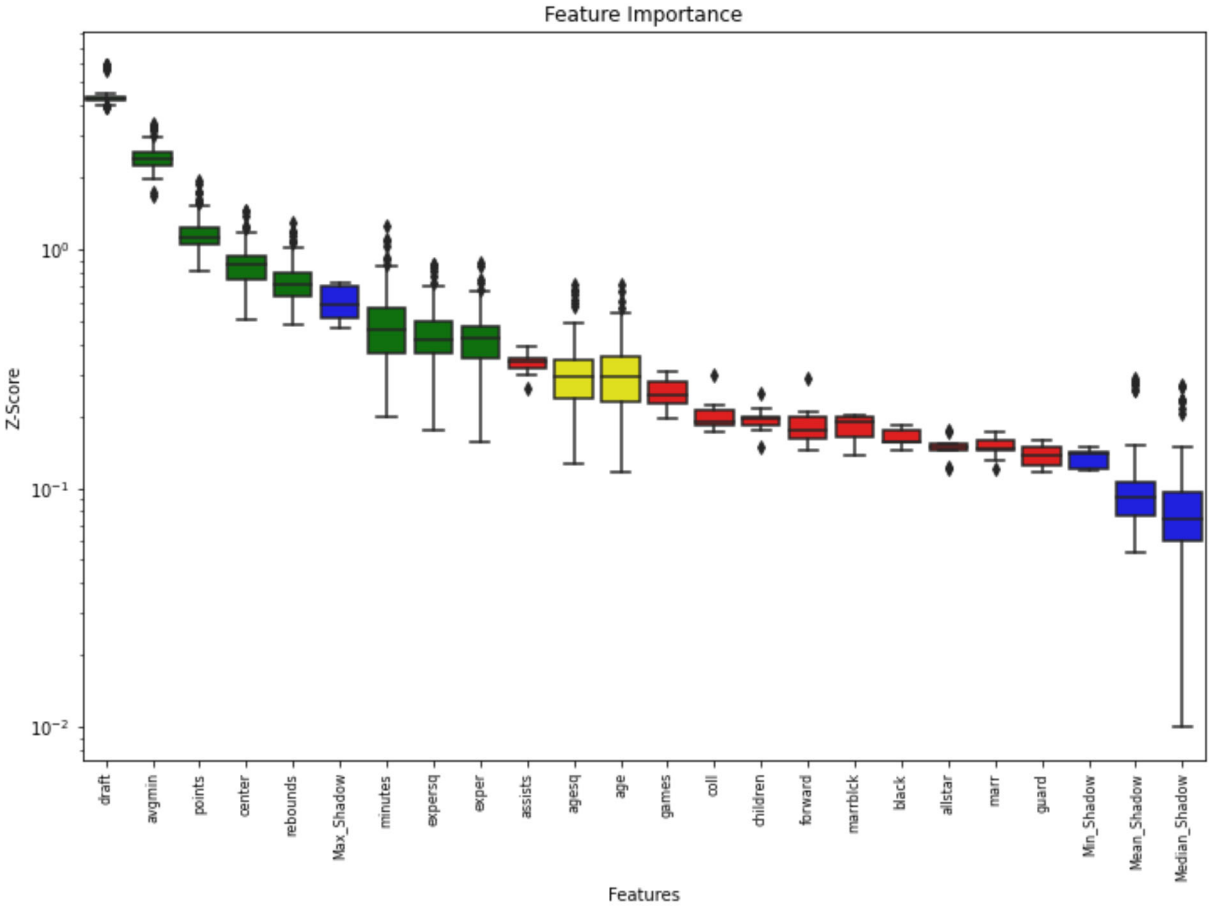
(a) Using the Boruta Algorithm identify the top 10 predictors

In [6]:
```
# perform boruta to select potnetial predictor
from BorutaShap import BorutaShap
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
```

In [7]:
```
x = df.loc[:,(df.columns != 'wage')]
y = df['wage']
```

In [10]:
```
Feature_Selector = BorutaShap(importance_measure='shap', classification=False)
Feature_Selector.fit(X=x, y=y, n_trials=100, random_state=0)
Feature_Selector.plot(which_features='all')
```

```
8 attributes confirmed important: ['exper', 'center', 'expersq', 'rebounds', 'point
s', 'avgmin', 'draft', 'minutes']
10 attributes confirmed unimportant: ['forward', 'children', 'marr', 'assists', 'mar
rblck', 'games', 'guard', 'black', 'allstar', 'coll']
2 tentative attributes remains: ['agesq', 'age']
```



Based on the result from Boruta Algorithm, we will choose the 8 confirmed important variables and 2 tentative varaibles:

draft, avgmin, points,center, rebounds,minutes, expersq, exper, age, agesq (from high to low according to Z-score)

(b) Using Mallows Cp identify the top 10 predictors

In [87]:
```
# list of all variables
potential_list = ['center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'exper

# create combinations of 10 variables
import itertools
potential_list = ['center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'exper
variable_list = []
for subset in itertools.combinations(potential_list,10):
    variable_list.append(subset)


for i in variable_list[:20]:
    print(i)

print("Total number of possible combinations is", len(variable_list))
```

```
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'marr', 'forward')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'marr', 'guard')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'marr', 'allstar')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'marr', 'coll')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'marr', 'children')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'marr', 'assists')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'marr', 'games')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'marr', 'black')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'marr', 'marrblck')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'marr', 'age')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'marr', 'agesq')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'forward', 'guard')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'forward', 'allstar')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'forward', 'coll')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'forward', 'children')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'forward', 'assists')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'forward', 'games')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'forward', 'black')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'forward', 'marrblck')
('center', 'exper', 'rebounds', 'draft', 'points', 'avgmin', 'expersq', 'minutes',
'forward', 'age')
Total number of possible combinations is 184756
```

In [56]:
```
# Mallows Cp
from RegscorePy import *
y = df['wage']
x_original = df.loc[:,potential_list]
full_model = sm.OLS(y,sm.add_constant(x_original))
```

```
full_model_fit = full_model.fit()
y_pred = np.array(full_model_fit.fittedvalues)
mallow_select = pd.DataFrame()
for i in variable_list:
    x = df.loc[:,i]
    Mallow_CP_mod = sm.OLS(y,sm.add_constant(x))
    Mallow_CP_fit = Mallow_CP_mod.fit()
    y_sub= np.array(Mallow_CP_fit.fittedvalues)
    k = 21
    p = 11
    cp = mallow.mallow(y,y_pred,y_sub,k,p)
    mallow_select = mallow_select.append({'combination' :str(i), 'mallow' :cp},ignor
```

In [60]:

```
# calculate distance
mallow_select["distance"] = np.abs(11-mallow_select.mallow)
```

In [61]:

```
mallow_select
```

Out[61]:

	combination	mallow	distance
0	('center', 'exper', 'rebounds', 'draft', 'poin...	12.494124	1.494124
1	('center', 'exper', 'rebounds', 'draft', 'poin...	12.494124	1.494124
2	('center', 'exper', 'rebounds', 'draft', 'poin...	12.912346	1.912346
3	('center', 'exper', 'rebounds', 'draft', 'poin...	11.253221	0.253221
4	('center', 'exper', 'rebounds', 'draft', 'poin...	12.831555	1.831555
...	...	...	...
184751	('forward', 'guard', 'allstar', 'coll', 'assis...	117.081912	106.081912
184752	('forward', 'guard', 'allstar', 'children', 'a...	122.178118	111.178118
184753	('forward', 'guard', 'coll', 'children', 'assi...	144.389133	133.389133
184754	('forward', 'allstar', 'coll', 'children', 'as...	140.339847	129.339847
184755	('guard', 'allstar', 'coll', 'children', 'assi...	120.945829	109.945829

184756 rows × 3 columns

In [98]:

```
mallow_select.sort_values(["distance", "mallow"], axis = 0)
```

Out[98]:

	combination	mallow	distance
70820	('center', 'draft', 'points', 'avgmin', 'forwa...	10.998546	0.001454
70946	('center', 'draft', 'points', 'avgmin', 'guard...	10.998546	0.001454
168495	('draft', 'points', 'avgmin', 'forward', 'guar...	10.998546	0.001454
68351	('center', 'draft', 'points', 'avgmin', 'exper...	11.002223	0.002223
165720	('draft', 'points', 'avgmin', 'expersq', 'minu...	11.002223	0.002223
...	...	...	...
91272	('center', 'expersq', 'marr', 'forward', 'guar...	212.826782	201.826782
92228	('center', 'marr', 'forward', 'guard', 'coll',...	213.979441	202.979441

	combination	mallow	distance
91254	('center', 'expersq', 'marr', 'forward', 'guar...	215.158817	204.158817
91255	('center', 'expersq', 'marr', 'forward', 'guar...	216.902133	205.902133
91287	('center', 'expersq', 'marr', 'forward', 'guar...	223.153688	212.153688

184756 rows × 3 columns

In [100…
```
mallow_select.iloc[70820].combination
```

Out[100…
```
"('center', 'draft', 'points', 'avgmin', 'forward', 'coll', 'children', 'games', 'ag
e', 'agesq')"
```

In [120…
```
mallow_select.iloc[70946].combination
```

Out[120…
```
"('center', 'draft', 'points', 'avgmin', 'guard', 'coll', 'children', 'games', 'ag
e', 'agesq')"
```

In [121…
```
mallow_select.iloc[168495].combination
```

Out[121…
```
"('draft', 'points', 'avgmin', 'forward', 'guard', 'coll', 'children', 'games', 'ag
e', 'agesq')"
```

In [15]:
```
print(red('From above mallow cp result : ',['bold']))
print('We achieve three combinations with same mallow-cp values. They have 8 identic
```

```
From above mallow cp result :
We achieve three combinations with same mallow-cp values. They have 8 identical pred
ictors.
We suspect that there is perfect multicollinearity between the rest three variables
 : guard, forward, center
```

In [8]:
```
df_test = df.copy()
df_test['sum_up'] = df_test['center']+df_test['forward']+df_test['guard']
df_test.sum_up.value_counts()
```

Out[8]:
```
1    240
Name: sum_up, dtype: int64
```

In [16]:
```
print('The made up columns proved our hypothesis, so we only need two variables from
```

```
The made up columns proved our hypothesis, so we only need two variables from guard,
forward, and center
```

Based on the result from Mallows Cp, we will choose the following varaibles:

draft, points, avgmin, forward, center, coll, children, games, age, agesq

(c) Based on your findings from parts (a) and (b) above, select your preferred choice of predictors

In [6]:
```
list_brt = ['draft','avgmin','points', 'center','rebounds', 'minutes','expersq', 'ex
list_cp = ['center', 'draft', 'points', 'avgmin', 'forward', 'coll', 'children', 'ga
variable_intersect = np.intersect1d(np.array(list_brt), np.array(list_cp))
variable_union = np.union1d(np.array(list_brt), np.array(list_cp))
print(blue('Intersection',['bold']))
```

```
print(variable_intersect,'with length of ',len(variable_intersect))
print(blue('Union',['bold']))
print(variable_union,'with length of ',len(variable_union))
```

```
Intersection
['age' 'agesq' 'avgmin' 'center' 'draft' 'points'] with length of  6
Union
['age' 'agesq' 'avgmin' 'center' 'children' 'coll' 'draft' 'exper'
 'expersq' 'forward' 'games' 'minutes' 'points' 'rebounds'] with length of  14
```

In [7]:

```
# avgmin = (minutes/games)
df2 = df.copy()
df2['minperg'] = df2['minutes']/df2["games"]
df2[['minperg', "avgmin"]]
```

Out[7]:

	minperg	avgmin
0	37.233766	37.233761
1	35.756410	35.756409
2	15.527027	15.527030
3	25.063830	25.063829
4	25.560976	25.560980
...	...	...
264	33.392405	33.392410
265	14.453333	14.453330
266	17.865672	17.865669
267	27.089744	27.089741
268	9.400000	9.400000

240 rows × 2 columns

Based on the result from (a) and (b), we will choose the following varaibles:

In [8]:

```
print(blue('age, avgmin, draft, points, coll, forward, center, rebounds',['bold']))
```

```
age, avgmin, draft, points, coll, forward, center, rebounds
```

Reasons:

> 1. We conduct two predictor selection process, we will keep the intersection of predictor selected by two process
> 2. After that, we will delete any high order term of existing predictor as we will conduct linearity check afterwards
> 3. Then we deal with multicollinearty issues that can be easily found: minutes/games = avgmin, guard + center + forward = 1

Q2: Descriptive Analysis

(a) Descriptive analysis of the variables

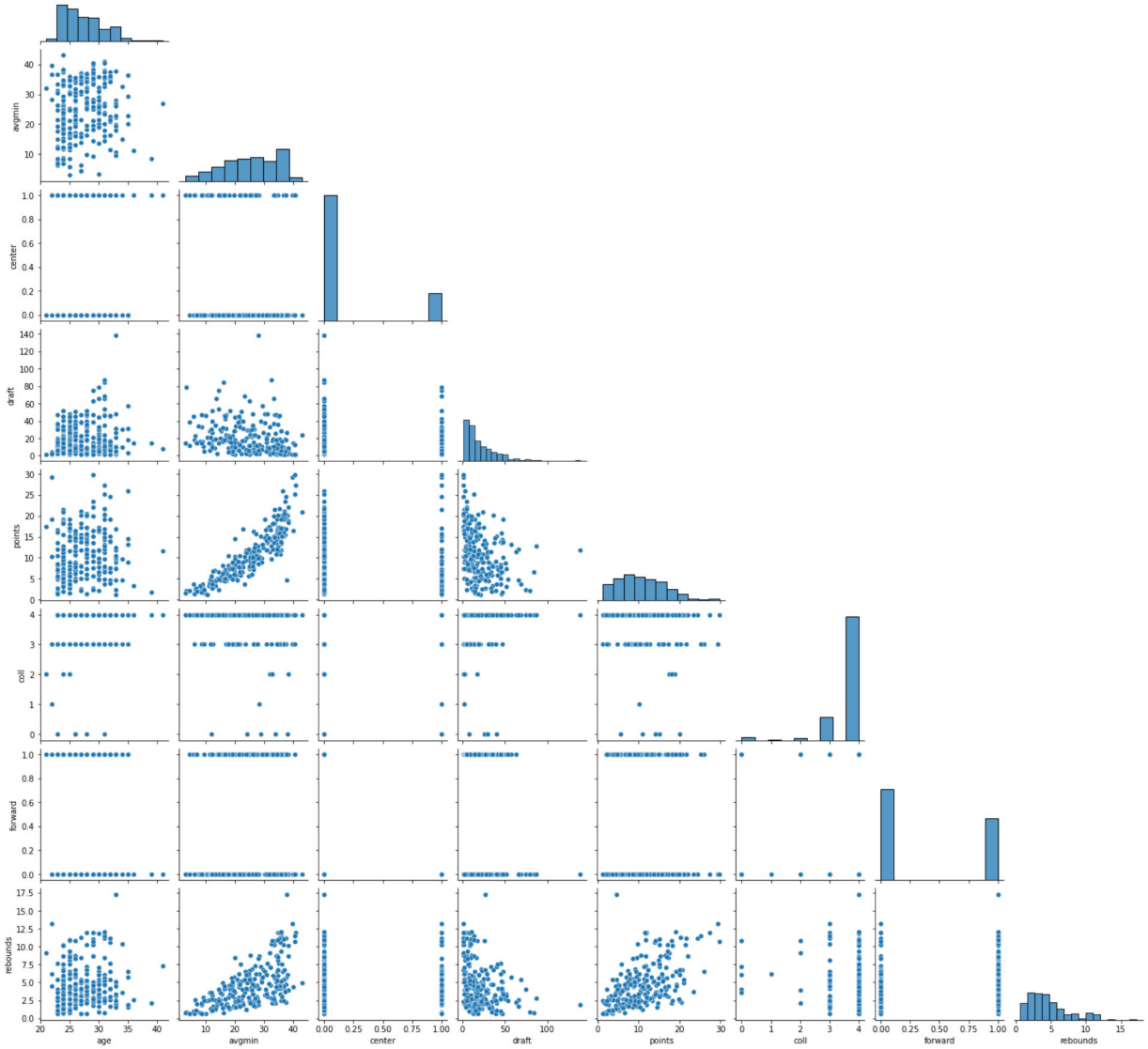
In [9]:
```
variables = df[['age','avgmin','center','draft','points','coll','forward','rebounds'
print(variables.describe())
print(variables.info())
```

```
              age      avgmin      center      draft      points        coll  \
count  240.000000  240.000000  240.000000  240.00000  240.000000  240.000000
mean    27.491667   25.080930    0.179167   20.20000   10.795833    3.720833
std      3.476149    9.392200    0.384293   18.73582    5.842220    0.715331
min     21.000000    2.888889    0.000000    1.00000    1.200000    0.000000
25%     24.000000   18.313507    0.000000    7.00000    6.200000    4.000000
50%     27.000000   25.773495    0.000000   14.50000   10.250000    4.000000
75%     30.000000   33.696756    0.000000   28.25000   14.850000    4.000000
max     41.000000   43.085369    1.000000  139.00000   29.799999    4.000000

          forward    rebounds         wage
count  240.000000  240.000000   240.000000
mean     0.404167    4.612083  1532.652500
std      0.491756    2.955298   996.567088
min      0.000000    0.600000   150.000000
25%      0.000000    2.500000   783.500000
50%      0.000000    4.000000  1265.500000
75%      1.000000    5.825000  2114.000000
max      1.000000   17.299999  5740.000000
<class 'pandas.core.frame.DataFrame'>
Int64Index: 240 entries, 0 to 268
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       240 non-null    int64
 1   avgmin    240 non-null    float64
 2   center    240 non-null    int64
 3   draft     240 non-null    float64
 4   points    240 non-null    float64
 5   coll      240 non-null    int64
 6   forward   240 non-null    int64
 7   rebounds  240 non-null    float64
 8   wage      240 non-null    float64
dtypes: float64(5), int64(4)
memory usage: 18.8 KB
None
```

We used pairplot to interpret and determine the correlation between different predictors.

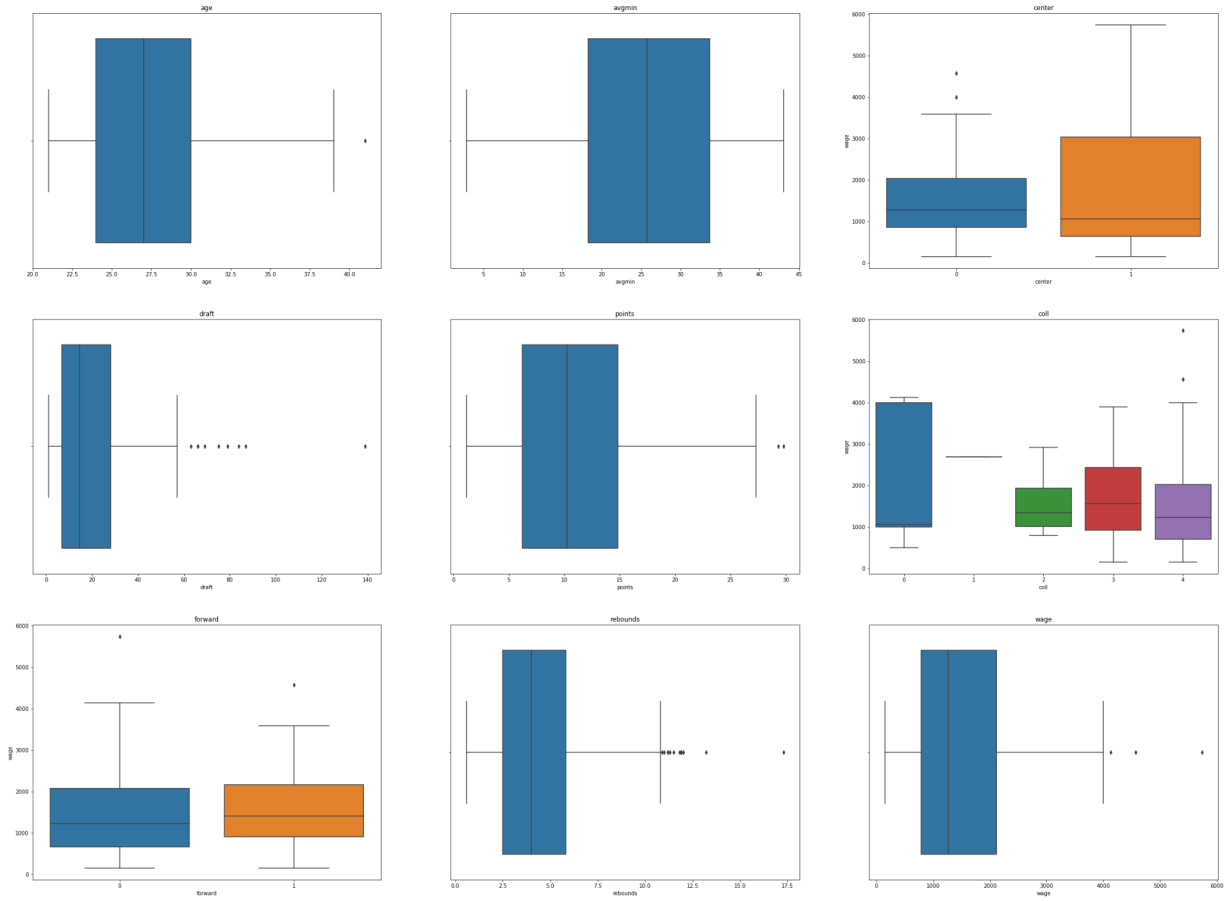
In [10]:
```
sns.pairplot(data = variables,vars = ['age','avgmin','center','draft','points','coll
```

Out[10]:
```
<seaborn.axisgrid.PairGrid at 0x2406b9c25e0>
```



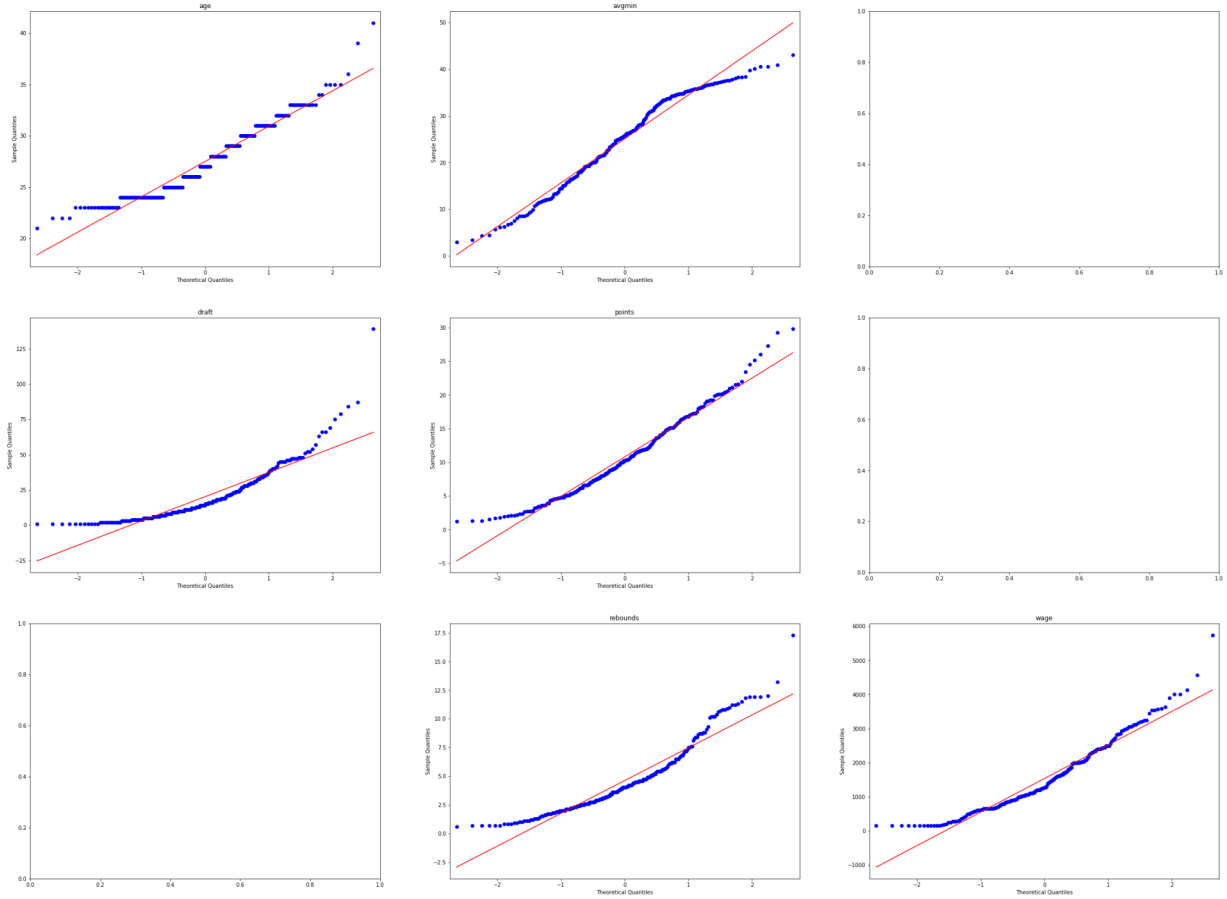
In [11]:

```
# boxplot:
var = ['age','avgmin','center','draft','points','coll','forward','rebounds','wage']
fig,axes = plt.subplots(3,3,figsize = (40,30))
axes = axes.ravel()
for i in range(9):
    if len(variables[var[i]].unique()) <=5:
        sns.boxplot(ax = axes[i],x = var[i],y = 'wage',data = variables)
    else:
        sns.boxplot(ax = axes[i],x = var[i],data = variables)
    axes[i].set_title(str(var[i]))
```



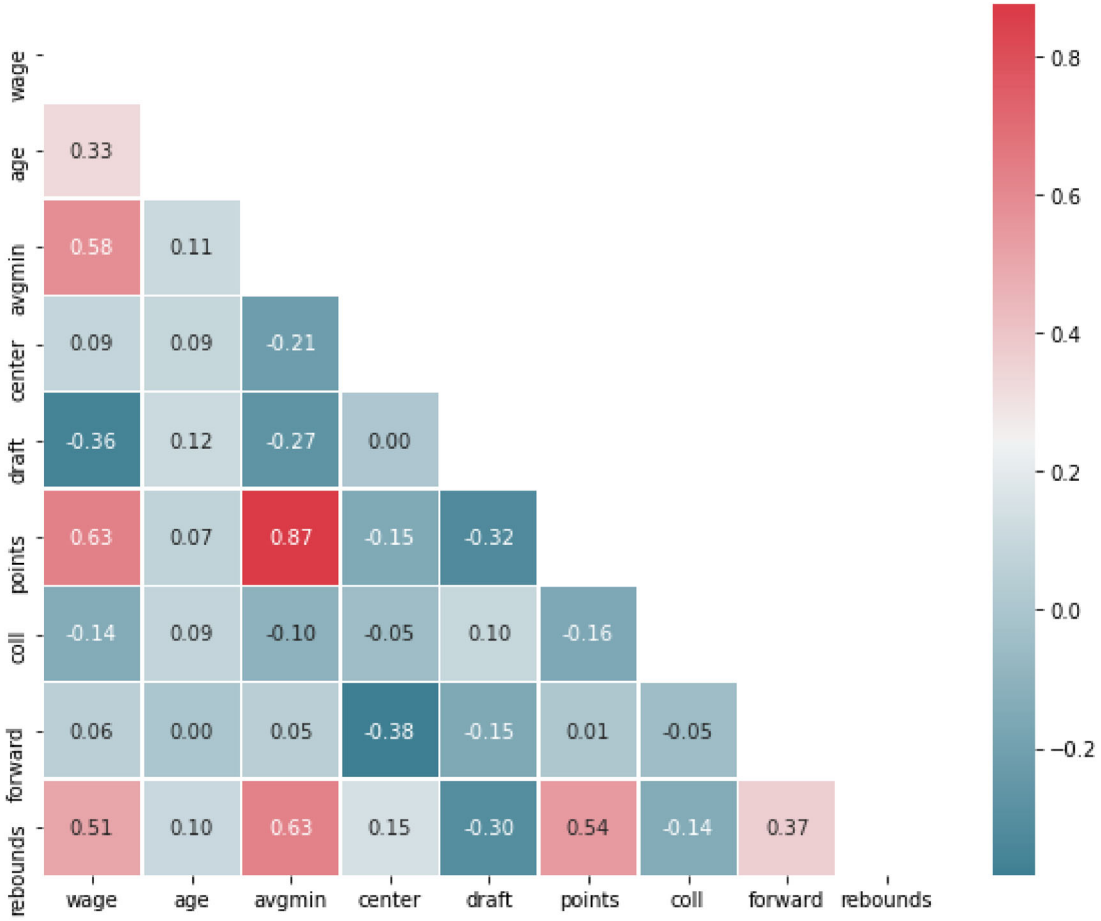
In [12]:

```
# QQ plot
var = ['age','avgmin','center','draft','points','coll','forward','rebounds','wage']
fig,axes = plt.subplots(3,3,figsize = (40,30))
axes = axes.ravel()
for i in range(9):
    if len(variables[var[i]].unique()) <=5:
        continue
    else:
        sm.qqplot(ax = axes[i],data = variables[var[i]],line = 'r')
    axes[i].set_title(str(var[i]))
```



In [15]:

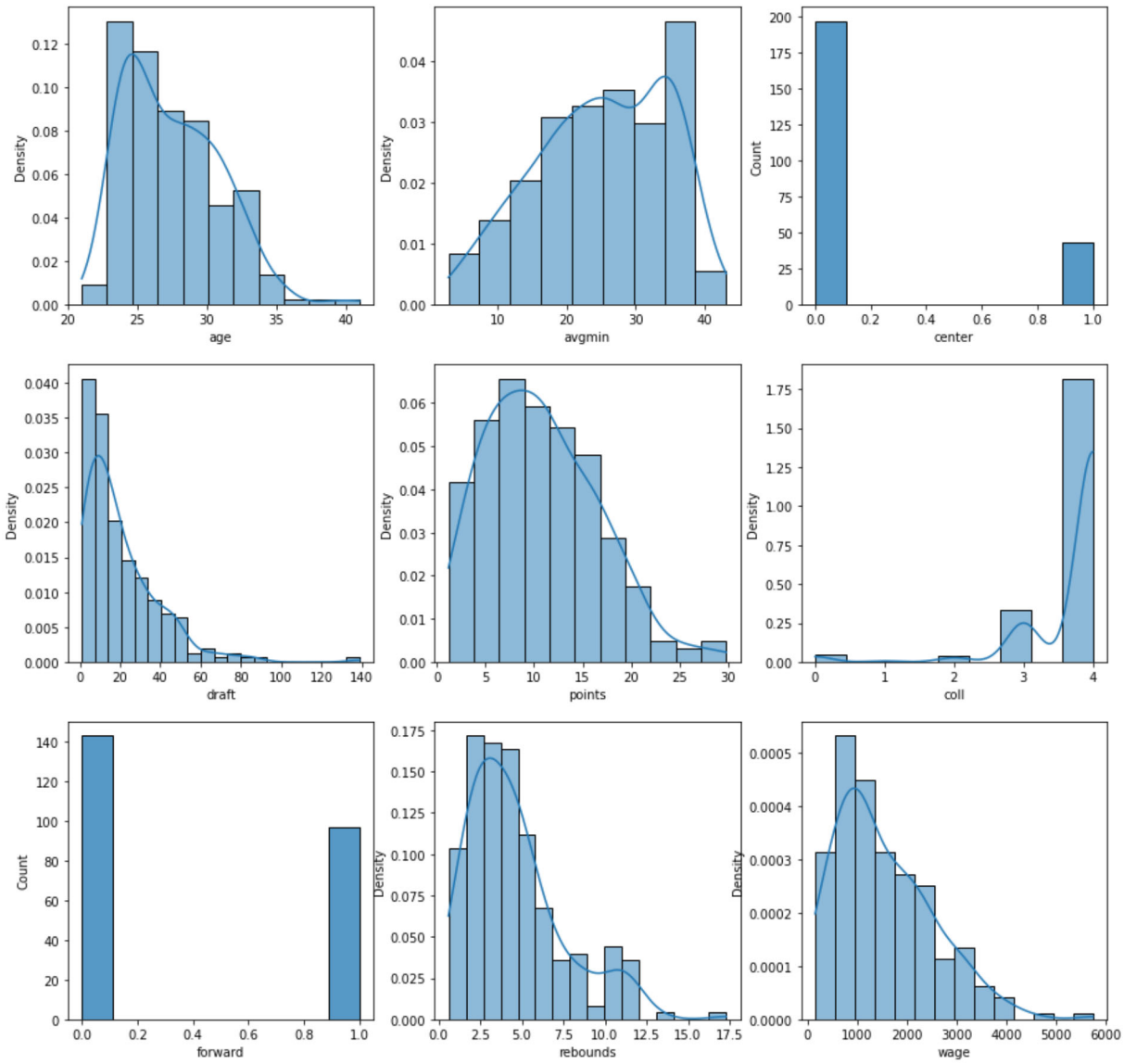
```
# correlation plot
sub_df = variables[['wage','age','avgmin','center','draft','points','coll','forward'
corr = sub_df.corr()
fig, ax = plt.subplots(figsize=(10, 8))
colormap = sns.diverging_palette(220, 10, as_cmap = True)
dropvals = np.zeros_like(corr)
dropvals[np.triu_indices_from(dropvals)] = True
sns.heatmap(corr, cmap = colormap, linewidths = .5, annot = True, fmt = ".2f", mask
plt.show()
```



(b) Density plots and fitted distributions.

In [16]:

```
# plot histogram and corresponding kde
fig,axes = plt.subplots(3,3,figsize = (15,15))
axes = axes.ravel()
for i in range(9):
    if len(variables[var[i]].unique()) == 2:
        sns.histplot(data=variables, x=var[i],kde = False,ax = axes[i])
    else:
        sns.histplot(data=variables, x=var[i],kde = True,stat = 'density',ax = axes[
```



In [6]:

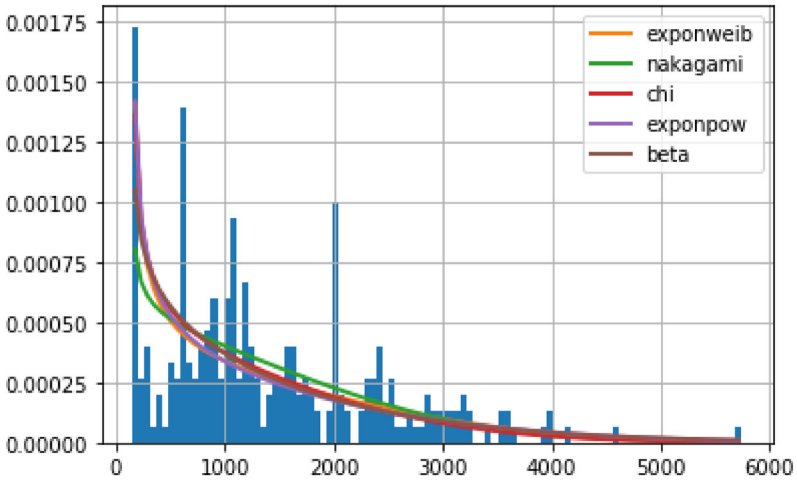
```
# We explore more about response variable and use cullen-frey graph to see the distr
from fitter import Fitter
f = Fitter(df.wage)
f.fit()
f.summary()
```

```
WARNING:root:SKIPPED kstwo distribution (taking more than 30 seconds)
C:\Software\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py:1715: I
ntegrationWarning: The integral is probably divergent, or slowly convergent.
  return integrate.quad(self._mom_integ1, 0, 1, args=(m,)+args)[0]
WARNING:root:SKIPPED rv_continuous distribution (taking more than 30 seconds)
WARNING:root:SKIPPED rv_histogram distribution (taking more than 30 seconds)
C:\Software\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py:1715: I
ntegrationWarning: The algorithm does not converge.  Roundoff error is detected
  in the extrapolation table.  It is assumed that the requested tolerance
  cannot be achieved, and that the returned result (if full_output = 1) is
  the best which can be obtained.
  return integrate.quad(self._mom_integ1, 0, 1, args=(m,)+args)[0]
WARNING:root:SKIPPED genexpon distribution (taking more than 30 seconds)
C:\Software\Anaconda3\lib\site-packages\scipy\stats\_distn_infrastructure.py:1715: I
ntegrationWarning: The maximum number of subdivisions (50) has been achieved.
  If increasing the limit yields no improvement it is advised to analyze
  the integrand in order to determine the difficulties.  If the position of a
  local difficulty can be determined (singularity, discontinuity) one will
  probably gain from splitting up the interval and calling the integrator
  on the subranges.  Perhaps a special-purpose integrator should be used.
  return integrate.quad(self._mom_integ1, 0, 1, args=(m,)+args)[0]
WARNING:root:SKIPPED kappa4 distribution (taking more than 30 seconds)
```

```
WARNING:root:SKIPPED kstwobign distribution (taking more than 30 seconds)
WARNING:root:SKIPPED levy_stable distribution (taking more than 30 seconds)
WARNING:root:SKIPPED recipinvgauss distribution (taking more than 30 seconds)
WARNING:root:SKIPPED vonmises distribution (taking more than 30 seconds)
WARNING:root:SKIPPED vonmises_line distribution (taking more than 30 seconds)
```

Out[6]:

	sumsquare_error	aic	bic	kl_div
exponweib	0.000004	1899.852774	-4823.945413	inf
nakagami	0.000004	1913.351638	-4820.042568	inf
chi	0.000004	1939.770924	-4819.509377	inf
exponpow	0.000004	1874.842826	-4817.184895	inf
beta	0.000004	1871.110985	-4799.586176	inf



We may use transformations on 'wage'.

(c) Identify non-linearities within the variables

We run a simple linear regression to see if there are any non-linearities:

In [17]:

```
# first perform ols regression
y = variables['wage']
x = variables[['age','avgmin','center','draft','points','coll','forward','rebounds']
reg_r = smf.ols(formula = 'wage ~ age + avgmin + center + draft + points + coll + fo
result_r = reg_r.fit()
```

In [18]:

```
result_r.summary()
```

Out[18]:

OLS Regression Results

Dep. Variable:	wage	**R-squared:**	0.560
Model:	OLS	**Adj. R-squared:**	0.545
Method:	Least Squares	**F-statistic:**	36.82
Date:	Tue, 09 Nov 2021	**Prob (F-statistic):**	2.81e-37
Time:	17:37:45	**Log-Likelihood:**	-1898.4
No. Observations:	240	**AIC:**	3815.
Df Residuals:	231	**BIC:**	3846.
Df Model:	8		
Covariance Type:	nonrobust		

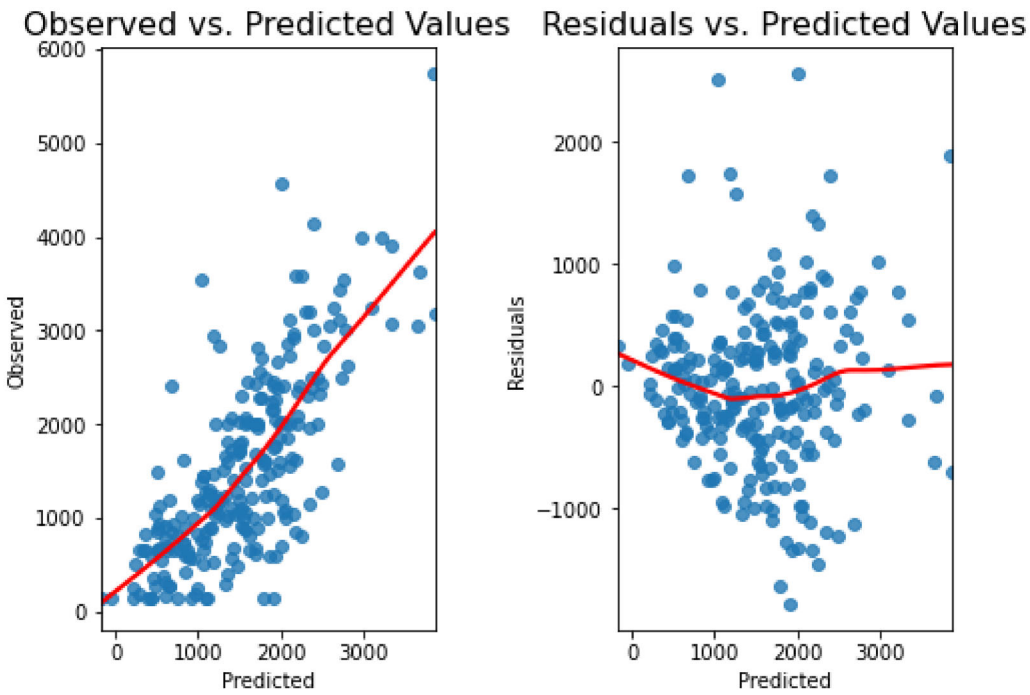
	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-1582.0468	420.850	-3.759	0.000	-2411.242	-752.852
age	84.4998	12.945	6.528	0.000	58.995	110.004
avgmin	9.3790	11.677	0.803	0.423	-13.627	32.385
center	373.4870	166.036	2.249	0.025	46.348	700.626
draft	-10.4124	2.544	-4.092	0.000	-15.425	-5.399
points	74.3322	15.883	4.680	0.000	43.038	105.626
coll	-71.2023	62.420	-1.141	0.255	-194.188	51.784
forward	76.8712	129.579	0.593	0.554	-178.437	332.180
rebounds	28.4522	27.413	1.038	0.300	-25.560	82.464

Omnibus:	20.038	**Durbin-Watson:**	1.931
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	34.476
Skew:	0.476	**Prob(JB):**	3.26e-08
Kurtosis:	4.594	**Cond. No.**	443.

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

In [19]:
```
# plot predict y V.S residual plot
fig, ax = plt.subplots(1,2,figsize=(8, 6))
fig.tight_layout(pad=6.0)
sns.regplot(x=result_r.fittedvalues, y=variables['wage'], lowess=True, ax=ax[0], lin
ax[0].set_title('Observed vs. Predicted Values', fontsize=16)
ax[0].set(xlabel='Predicted', ylabel='Observed')

sns.regplot(x=result_r.fittedvalues, y=result_r.resid, lowess=True, ax=ax[1], line_k
ax[1].set_title('Residuals vs. Predicted Values', fontsize=16)
ax[1].set(xlabel='Predicted', ylabel='Residuals')
```

Out[19]:
```
[Text(0.5, 33.0, 'Predicted'), Text(288.49090909090904, 0.5, 'Residuals')]
```



In [20]:

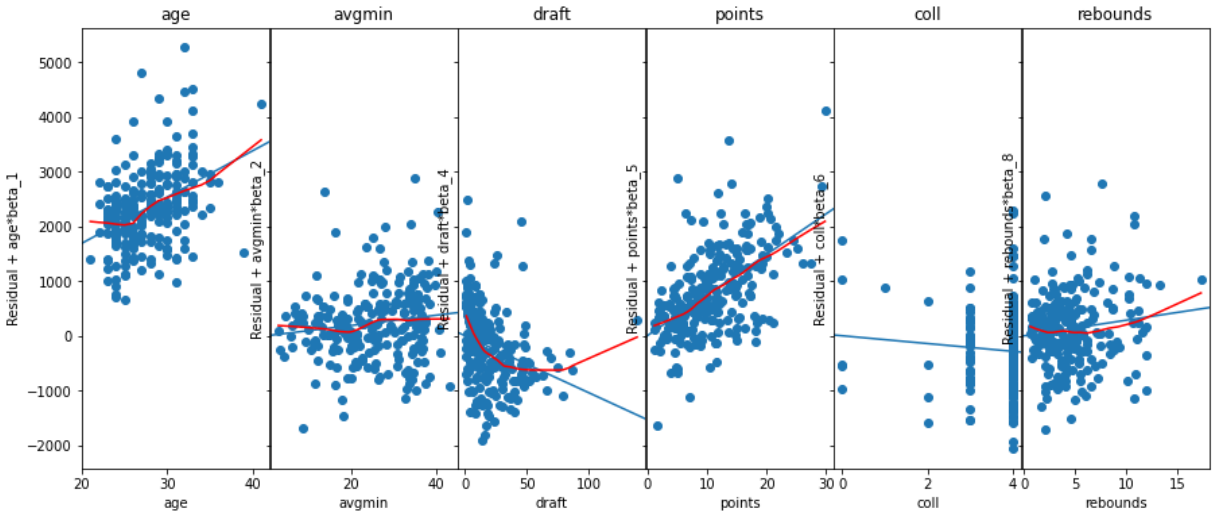
```
import statsmodels.regression.linear_model as rg
import statsmodels.stats.diagnostic as dg

test =  dg.linear_reset(result_r, power=2,  test_type='fitted', use_f = True)

print(blue("Ramsey-RESET:",['bold']))
print(test)
```

```
Ramsey-RESET:
<F test: F=array([[6.88085555]]), p=0.009294989188064835, df_denom=230, df_num=1>
```

According to P-value which is 0.03047, higher order of some of the variables may be added to the model.

In [21]:

```
# ccpr plot
from statsmodels.graphics.regressionplots import add_lowess
fig, axs = plt.subplots(1,6, figsize=(15, 6), facecolor='w', edgecolor='k',sharey =
fig.subplots_adjust(hspace = .5, wspace=.001)
axs = axs.ravel()
predictor_category = ['age','avgmin','draft','points','coll','rebounds']
for i in range(6):
    sm.graphics.plot_ccpr(result_r, predictor_category[i],ax = axs[i])
    axs[i].set_title(str(predictor_category[i]))
    add_lowess(axs[i],frac = 0.5)
```



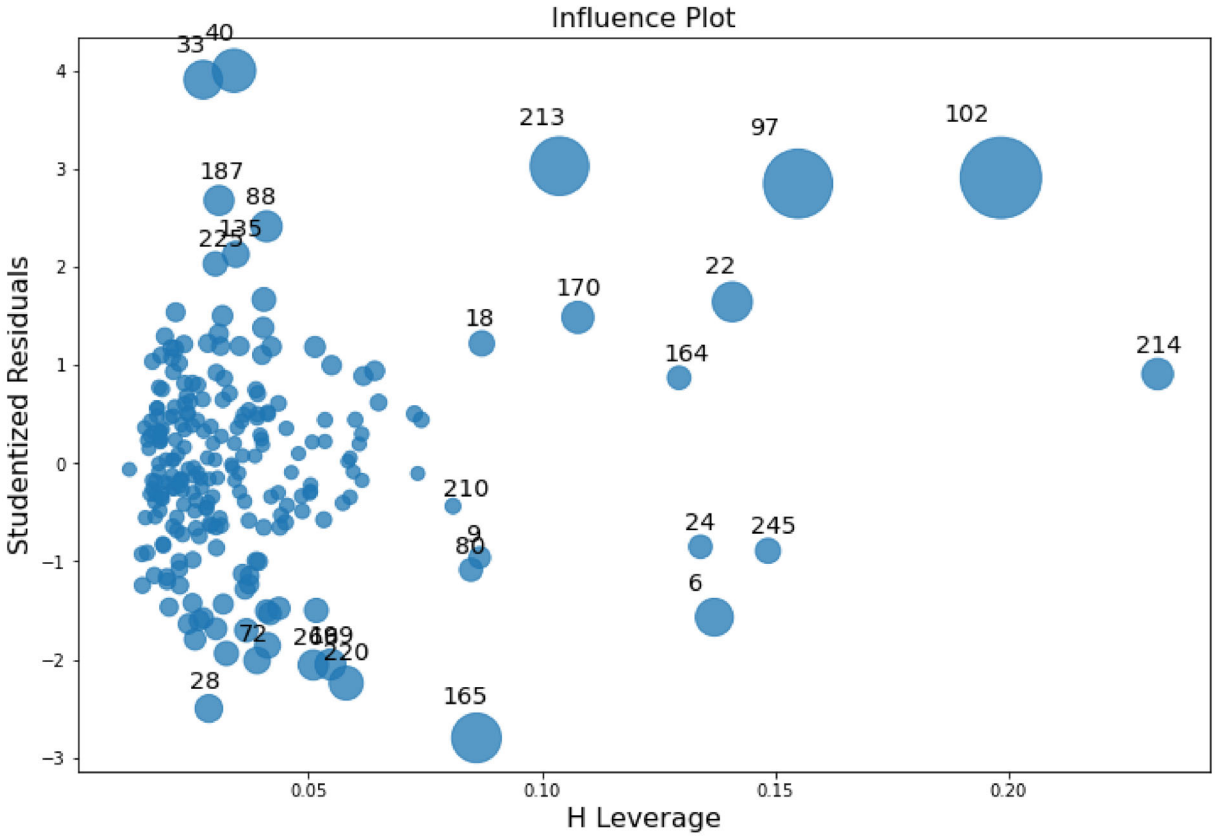
Based on the ccpr plot, we found that draft can be changed to quadratic form.

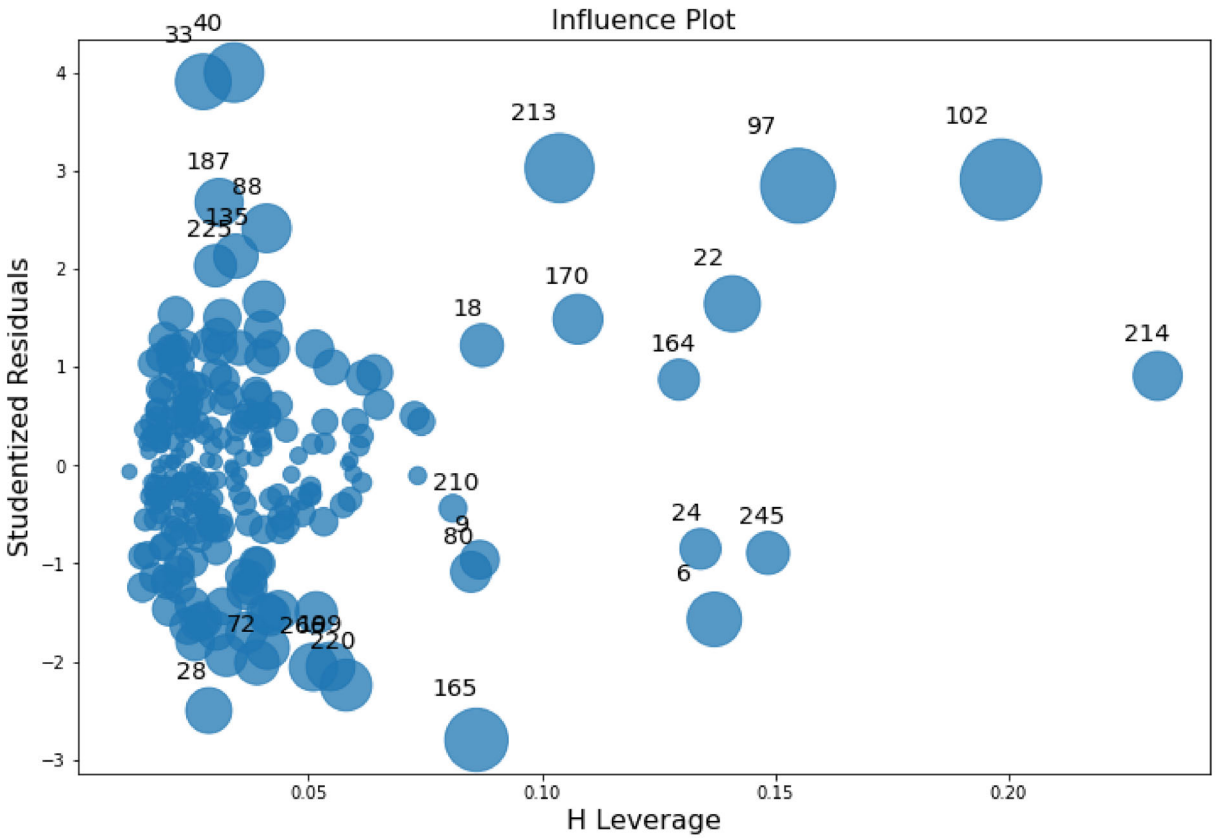
Age and rebounds may also need transformations, but they do not have a strong trend. It becomes unnecessary after we apply log transformation on wage.

If we do not transform the variables and simply add those non-linear variables, we will break the linearity assumption of the multiple regression model. Our parameter estimates would be biased, and our model would have poor performance.

(d) Outliers

In [22]:
```
# use influence plot to identify outlier
fig,ax = plt.subplots(figsize = (12,8))
fig = sm.graphics.influence_plot(result_r,alpha = 0.05,ax = ax,criterion = 'cooks')
fig,ax = plt.subplots(figsize = (12,8))
fig = sm.graphics.influence_plot(result_r,alpha = 0.05,ax = ax,criterion = 'DFFITS')
```





In [51]:
```
variables.loc[102,:]
```

Out[51]:
```
age          33.000000
avgmin       28.123461
center        0.000000
draft       139.000000
points       11.900000
coll          4.000000
forward       0.000000
rebounds      1.900000
wage       2400.000000
Name: 102, dtype: float64
```

In [52]:
```
variables.loc[213,:]
```

Out[52]:
```
age          29.000000
avgmin       40.512501
center        1.000000
draft         1.000000
points       29.799999
coll          4.000000
forward       0.000000
rebounds     10.700000
wage       5740.000000
Name: 213, dtype: float64
```

In [53]:
```
variables.loc[165,:]
```

Out[53]:
```
age          39.000000
avgmin        8.533334
center        1.000000
draft        14.000000
points        1.700000
coll          4.000000
forward       0.000000
rebounds      2.100000
```

```
wage        150.000000
Name: 165, dtype: float64
```

Our model's goal is to predict NBA players' salary by players' performance such as points per game, rebounds per game and average minutes played per game. In this case, every influencial point in the influence plots above represents an active player in that season. We selected some outlier points and search for the players corresponding to these points in order to see if its existence is reasonable.

Point 213 represents a player named David Robinson.During that season, he was the most valuable player, scoring champion with highest points per game, also, he has the highest win share, which means he was the most capable player to lead the team to victory that season. In response, David Robinson's salary is higher than any other players in the league.

Point 102 represents Sedale Threatt, the 139th draft pick in 1983,he was just an average substitute player until 1990. He was traded to LA Lakers to play as a backup guard, but Laker's starting point guard abruptly retired due to HIV. Therefore, he was assigned to be the starting point guard and did so well that the team offered him a lucrative contract in return.

Point 165 represents Tree Rollins, one of the oldest player at that season and meanwhile had the lowest salary of 150 thousand dollars. It's not uncommon for players to be paid the lowest wage, and more than a tenth of the players that season were paid as much as he was.

These players might be outliers in the graph, but in reality, their situation is not uncommon and are even fairly representative. David Robinson, the MVP player in that season, represents the best players in the NBA. Sedale Threatt, who was a below-average player until he took his chance, represents every regular player who has a dream and fights for it. Tree Rollins represents a player at the end of his career with a veteran's minimum salary.

In conclusion, we cannot simply take any of them off based on the influential plot.

Reference:

https://www.basketball-reference.com/leagues/NBA_1994_per_game.html#per_game_stats::

https://www.basketball-reference.com/players/r/robinda01.html

https://en.wikipedia.org/wiki/Sedale_Threatt

https://www.basketball-reference.com/players/r/rollitr01.html

(e) NAs

We only have NAs in "draft".

In [23]:
```
# first we observe the index of missing values
miss = df_missing[df_missing['draft'].isnull()]
miss_index = miss['draft'].index.tolist()
miss_index
```

Out[23]:
```
[8,
```

```
 21,
 35,
 41,
 52,
 68,
 74,
 76,
 77,
 81,
 85,
 93,
 98,
 106,
 120,
 126,
 138,
 141,
 143,
 145,
 151,
 155,
 171,
 175,
 228,
 229,
 230,
 239,
 249]
```

In [24]:
```
# add new a column is_null_draft where 1 means null, 0 means not null
df_test_corr = df_missing.copy()
df_test_corr['is_null_draft'] = 1
df_test_corr['is_null_draft'].where(df_test_corr['draft'].isnull(),0,inplace = True)
```

In [25]:
```
df_test_corr['is_null_draft'].value_counts()
```

Out[25]:
```
0    240
1     29
Name: is_null_draft, dtype: int64
```

We run a logit regression to see if any variables could affect the probability of having a missing 'draft'.

In [26]:
```
y = df_test_corr['is_null_draft']
x = df_test_corr.drop(['wage','draft','is_null_draft','lwage'],axis = 1)
result = sm.Logit(y, sm.add_constant(x)).fit()
print(result.summary())
```

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.236411
         Iterations: 35
                           Logit Regression Results
==============================================================================
Dep. Variable:          is_null_draft   No. Observations:                  269
Model:                          Logit   Df Residuals:                      250
Method:                           MLE   Df Model:                           18
Date:                Tue, 09 Nov 2021   Pseudo R-squ.:                 0.3085
Time:                        17:38:09   Log-Likelihood:                -63.595
converged:                      False   LL-Null:                       -91.972
Covariance Type:            nonrobust   LLR p-value:                 6.768e-06
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const        -47.6345        nan        nan        nan         nan         nan
marr          -0.0606      1.113     -0.054      0.957      -2.242       2.120
```

```
exper         -0.7194      0.537     -1.341      0.180      -1.771       0.332
age            4.2066      2.882      1.460      0.144      -1.441       9.854
coll           0.1917      0.384      0.500      0.617      -0.560       0.944
games          0.0053      0.028      0.192      0.848      -0.049       0.060
minutes       -0.0021      0.002     -1.318      0.187      -0.005       0.001
guard        -14.9451        nan        nan        nan         nan         nan
forward      -15.6567        nan        nan        nan         nan         nan
center       -17.0327        nan        nan        nan         nan         nan
points        -0.0742      0.157     -0.473      0.636      -0.381       0.233
rebounds      -0.0621      0.295     -0.211      0.833      -0.640       0.515
assists       -0.4392      0.378     -1.162      0.245      -1.180       0.301
allstar       -7.9795    164.095     -0.049      0.961    -329.599     313.640
avgmin         0.0968      0.125      0.773      0.439      -0.149       0.342
black          0.2587      0.777      0.333      0.739      -1.264       1.782
children       0.5841      0.577      1.013      0.311      -0.546       1.715
expersq        0.0217      0.047      0.460      0.646      -0.071       0.114
agesq         -0.0666      0.052     -1.275      0.202      -0.169       0.036
marrblck      -0.1667      1.231     -0.135      0.892      -2.580       2.247
==============================================================================

Possibly complete quasi-separation: A fraction 0.12 of observations can be
perfectly predicted. This might indicate that there is complete
quasi-separation. In this case some parameters will not be identified.
```

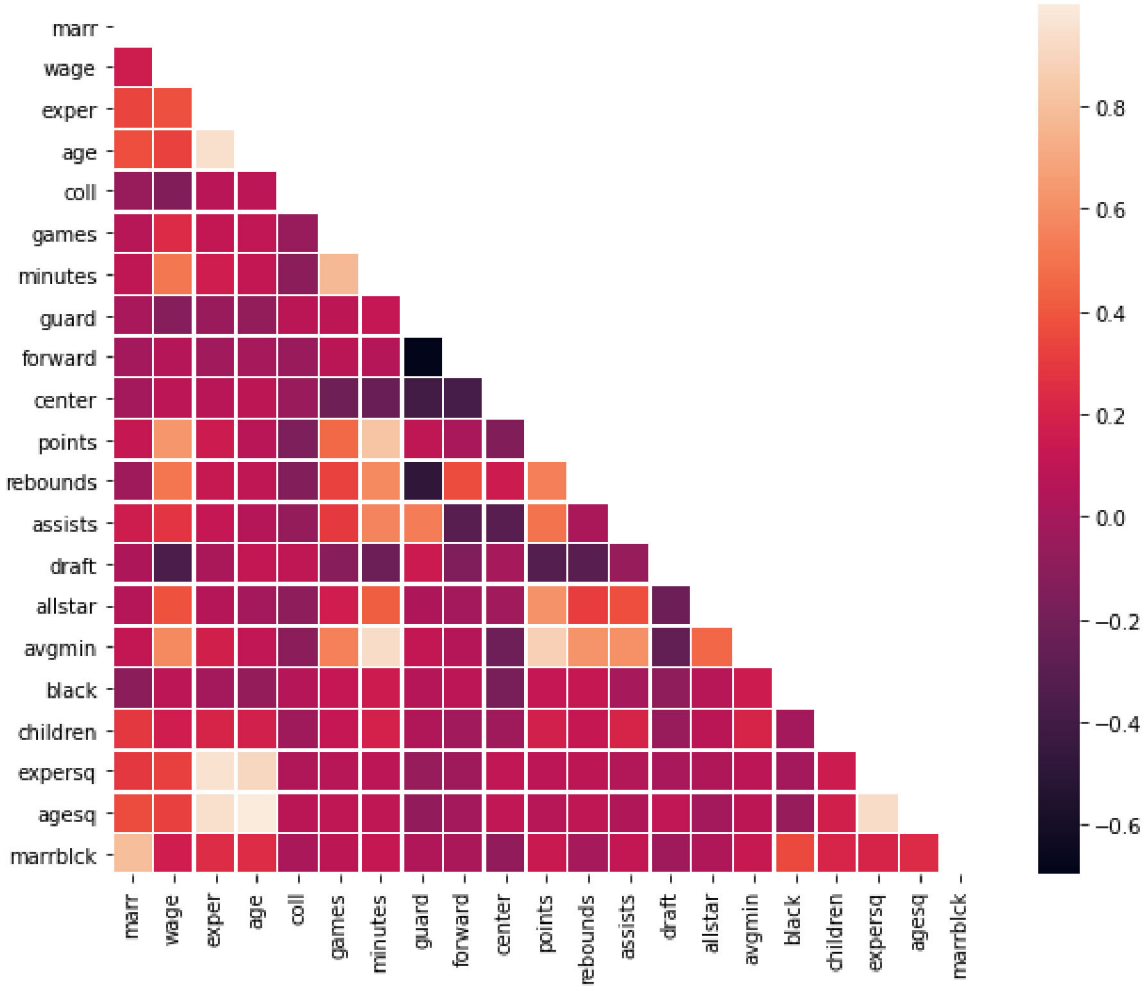
No variable has significant effect. Missing values are not likely to be caused by certain characteristics of the data.

In [27]:

```
# we will then draw the correlation plot to see if draft is important
corr = df.corr()
fig,ax = plt.subplots(figsize = (10,8))
mask = np.triu(df.corr())
sns.heatmap(corr,linewidths = 0.5,mask = mask,square = True)
```

Out[27]:

```
<AxesSubplot:>
```



In [28]:
```
corr['wage'].sort_values(ascending = False)[1:]
```

Out[28]:
```
points      0.631578
avgmin      0.582897
minutes     0.514826
rebounds    0.509130
allstar     0.383320
exper       0.380704
age         0.331491
agesq       0.327822
expersq     0.322339
assists     0.279039
games       0.242037
children    0.167958
marrblck    0.165852
marr        0.161330
center      0.088145
black       0.085361
forward     0.055290
guard      -0.123599
coll       -0.143489
draft      -0.362531
Name: wage, dtype: float64
```

From above correlation, we found out the variable 'draft' is likely to have a significance effect on wage, which could cause really large impact if we remove the variable 'draft'.

Thus, we cannot drop the column of 'draft'.

As shown above, the variable 'draft' has 29 missing values, but we could not tell if the missing values are due to failure in draft or no observations on draft. What's more, the Draft is a ordinal

variable, so we couldn't simply replace the missing values with either mean, median, or mode. Therefore, we decided to drop the 29 observations with missing values.

In addition, we want to use KNN to see approximately missing value of draft will belong to which group

https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

```
combine with cross validation to build a knn model to predict the missing
value
https://towardsdatascience.com/complete-guide-to-pythons-cross-validation-
with-examples-a9676b5cac12
from sklearn.model_selection import KFold
kf5 = KFold(n_splits = 5, shuffle = False)
kf3 = KFold(n_splits = 3,shuffle = False)
```

In [3]:
```
df = woo.dataWoo('nbasal')
```

we need to first perform PCA to reduce dimension

In [4]:
```
# first pick the data that don't contain null value for draft stored as train_df, th
from sklearn.preprocessing import StandardScaler
train_df = df[~df['draft'].isnull()]
train_set = train_df.iloc[:,df.columns != 'draft']
train_label = train_df.loc[:,df.columns == 'draft']
# then create prediction df, which only contains null value for draft stored as pred
pred_df = df[df['draft'].isnull()]
pred_set = pred_df.loc[:,df.columns != 'draft']
```

In [5]:
```
# before perform PCA, we need to centerdized our data, so that sum(column i) = 0
train_set_norm = StandardScaler().fit_transform(train_set)
pred_set_norm = StandardScaler().fit_transform(pred_set)
```

In [6]:
```
# we can visualize the centerdized data first for train_set_norm
feature = ['feature'+str(i) for i in range(train_set_norm.shape[1])]
standardized_train_set = pd.DataFrame(train_set_norm,columns = feature,index = train
standardized_train_set.head()
```

Out[6]:

	feature0	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8	
0	1.114895	-0.533091	-0.379069	-0.141736	0.391078	0.535373	1.256727	1.183216	-0.823603	-
1	1.114895	0.500104	-0.088409	0.146540	0.391078	0.593121	1.165874	1.183216	-0.823603	-
2	-0.896946	-0.887544	-1.251050	-0.718287	0.391078	0.362129	-0.744384	-0.845154	-0.823603	
3	-0.896946	0.500104	-0.088409	0.146540	0.391078	-1.197070	-0.710605	-0.845154	1.214177	-
4	-0.896946	-0.781962	-0.669730	-1.006563	0.391078	0.824114	0.358673	1.183216	-0.823603	-

5 rows × 21 columns

In [7]:
```
feature = ['feature'+str(i) for i in range(pred_set_norm.shape[1])]
standardized_pred_set = pd.DataFrame(pred_set_norm,columns = feature,index = pred_se
standardized_pred_set.head()
```

Out[7]:

	feature0	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8
8	1.190238	-0.986285	-1.035098	-0.649370	0.304082	0.488683	-0.409966	1.109400	-0.901388
21	-0.840168	-0.986285	-0.234622	-0.649370	0.304082	-0.591074	-0.600997	1.109400	-0.901388
35	1.190238	-0.127398	-1.035098	-0.649370	0.304082	-1.220932	-1.020335	1.109400	-0.901388
41	-0.840168	1.788581	0.966092	0.169401	0.304082	-1.535860	-0.807560	-0.901388	1.109400
52	1.190238	0.202943	0.565854	0.169401	0.304082	1.118541	0.110324	-0.901388	1.109400

5 rows × 21 columns

In [8]:

```
# next, perform PCA to reduce irrelevant dimension and get pca_df
from sklearn.decomposition import PCA
pca = PCA(n_components = 12) # we choose to keep 10 most important feature
features = ['feature'+str(i) for i in range(12)]
pca_train_set = pd.DataFrame(pca.fit_transform(train_set_norm),columns = features,in
pca_train_set.head()
```

Out[8]:

	feature0	feature1	feature2	feature3	feature4	feature5	feature6	feature7	feature8
0	1.051041	-1.226427	2.115160	-1.195376	0.113924	-0.418244	-0.681317	0.818447	0.925224
1	2.303245	-0.945495	3.035934	-1.072930	0.626745	0.146552	-0.410637	-0.225881	-0.945513
2	-2.996266	-0.289783	-0.983257	0.939946	0.666079	-1.167257	-1.386932	1.069106	0.051153
3	-0.778477	0.217941	-1.692294	0.067825	-1.003424	-0.739469	1.089752	-0.962444	-0.347633
4	-1.275316	-1.860036	1.087968	0.457798	-1.078139	-0.520803	-0.588438	0.798341	0.014021

In [9]:

```
# do the same thing for prediction set
pca_pred_set= pd.DataFrame(pca.fit_transform(pred_set_norm),index = pred_set.index)
pca_pred_set.head()
```

Out[9]:

	0	1	2	3	4	5	6	7	8
8	-1.741095	-1.976665	2.079022	1.082324	0.599997	-0.937162	0.008521	0.422266	0.138091
21	-2.183053	-0.976768	0.668381	-0.198981	-1.270008	-0.808505	0.414811	-0.617794	0.146286
35	-1.877525	-0.597677	2.166444	0.994902	1.432599	-1.343286	0.466486	-1.119531	-0.755648
41	1.143879	2.544899	-1.802928	-0.840113	0.094383	-1.095663	0.488714	-1.329687	-1.156212
52	0.850500	0.832824	0.451951	0.461917	1.135971	-1.184825	-1.128681	1.344954	-0.377321

From now, we have two dataset that can be used in the next step: cross validation and train true model.

1.pca_train_set train_label

2.pca_pred_set

In [11]:

```
from sklearn.model_selection import cross_validate
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5)
knn_result = cross_validate(knn,pca_train_set,train_label,cv = 10,scoring = ('neg_me
```

```
                                                                 'neg_me
                                                                 'neg_me
                                                                 'neg_ro
```

```
D:\Anaconda\lib\site-packages\sklearn\model_selection\_split.py:666: UserWarning: Th
e least populated class in y has only 1 members, which is less than n_splits=10.
  warnings.warn(("The least populated class in y has only %d"
D:\Anaconda\lib\site-packages\sklearn\neighbors\_classification.py:179: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
D:\Anaconda\lib\site-packages\sklearn\neighbors\_classification.py:179: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
D:\Anaconda\lib\site-packages\sklearn\neighbors\_classification.py:179: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
D:\Anaconda\lib\site-packages\sklearn\neighbors\_classification.py:179: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
D:\Anaconda\lib\site-packages\sklearn\neighbors\_classification.py:179: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
D:\Anaconda\lib\site-packages\sklearn\neighbors\_classification.py:179: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
D:\Anaconda\lib\site-packages\sklearn\neighbors\_classification.py:179: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
D:\Anaconda\lib\site-packages\sklearn\neighbors\_classification.py:179: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
D:\Anaconda\lib\site-packages\sklearn\neighbors\_classification.py:179: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
D:\Anaconda\lib\site-packages\sklearn\neighbors\_classification.py:179: DataConversi
onWarning: A column-vector y was passed when a 1d array was expected. Please change
the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

In [12]:

```
# use regression to predict
from sklearn.linear_model import LinearRegression
reg = LinearRegression()
reg_result = cross_validate(reg,pca_train_set,train_label,cv = 10,scoring = ('neg_me
                                                                 'neg_me
                                                                 'neg_me
                                                                 'neg_ro
```

In [13]:

```
reg_score = pd.DataFrame()
for i in sorted(reg_result.keys()):
    reg_score = reg_score.append({'method': i,'score':(np.mean(reg_result[i])*-1)},i
knn_score = pd.DataFrame()
for i in sorted(knn_result.keys()):
    knn_score = knn_score.append({'method': i,'score':(np.mean(knn_result[i])*-1)},i
```

In [15]:

```
print(blue('prediction error from regression model'))
```

```
print(reg_score)
print(blue('prediction error from knn model'))
print(knn_score)
```

```
prediction error from regression model
                                  method       score
0                                fit_time   -0.019554
1                             score_time   -0.006876
2           test_neg_mean_absolute_error   11.817716
3  test_neg_mean_absolute_percentage_error    1.371025
4            test_neg_mean_squared_error  302.152235
5       test_neg_root_mean_squared_error   16.544796
prediction error from knn model
                                  method       score
0                                fit_time   -0.004691
1                             score_time   -0.007078
2           test_neg_mean_absolute_error   13.775000
3  test_neg_mean_absolute_percentage_error    0.691329
4            test_neg_mean_squared_error  480.291667
5       test_neg_root_mean_squared_error   21.345832
```

From above prediction model, the prediction error are too large. So instead of trying to impute missing value, we we will throw observations with NAs away.

Q3: Model Building

In [7]:
```
final = df[['age','avgmin','draft','points','coll','forward','center','rebounds','wa
final
```

Out[7]:

	age	avgmin	draft	points	coll	forward	center	rebounds	wage
0	27	37.233761	19.0	15.5	4	0	0	3.9	1002.500000
1	28	35.756409	28.0	13.3	4	0	0	2.5	2030.000000
2	25	15.527030	19.0	5.5	4	0	1	3.3	650.000000
3	28	25.063829	1.0	7.3	4	1	0	5.1	2030.000000
4	24	25.560980	24.0	10.8	4	0	0	4.3	755.000000
...	...	...	...	...	...	...	...	...	...
264	29	33.392410	11.0	19.9	4	0	0	2.7	3210.000000
265	31	14.453330	54.0	4.8	4	1	0	2.5	715.000122
266	33	17.865669	4.0	10.4	3	0	0	1.6	600.000000
267	28	27.089741	2.0	15.7	4	0	1	6.2	2500.000000
268	33	9.400000	5.0	2.3	3	1	0	2.5	2000.000000

240 rows × 9 columns

We have 3 models which involved quadratic term, log, and linear functions.

Model 1

From the above observations in our baseline model, we find higher order effects of draft may be appropriate. So we decided to add $draft^2$ in the linear regression model, the outcome is shown below.

In [30]:
```
# model1:

y = final['wage']
x = final[['age','avgmin','draft','points','coll','forward','center','rebounds']]
x["draft**2"] = x["draft"]**2
x = sm.add_constant(x)
model1 = sm.OLS(y,x)
result1 = model1.fit()
result1.summary()
```

Out[30]:

OLS Regression Results

Dep. Variable:	wage	**R-squared:**	0.596
Model:	OLS	**Adj. R-squared:**	0.580
Method:	Least Squares	**F-statistic:**	37.64
Date:	Tue, 09 Nov 2021	**Prob (F-statistic):**	1.51e-40
Time:	17:38:35	**Log-Likelihood:**	-1888.4
No. Observations:	240	**AIC:**	3797.
Df Residuals:	230	**BIC:**	3832.
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1051.6700	421.611	-2.494	0.013	-1882.384	-220.956
age	79.1210	12.502	6.329	0.000	54.488	103.753
avgmin	9.1343	11.225	0.814	0.417	-12.983	31.251
draft	-31.3496	5.285	-5.932	0.000	-41.763	-20.936
points	62.6424	15.491	4.044	0.000	32.120	93.164
coll	-76.7517	60.018	-1.279	0.202	-195.006	41.503
forward	26.8640	125.066	0.215	0.830	-219.558	273.286
center	311.1943	160.218	1.942	0.053	-4.489	626.878
rebounds	35.4631	26.399	1.343	0.180	-16.552	87.478
draft2**	0.2540	0.057	4.469	0.000	0.142	0.366

Omnibus:	29.554	**Durbin-Watson:**	1.877
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	70.134
Skew:	0.565	**Prob(JB):**	5.90e-16
Kurtosis:	5.395	**Cond. No.**	1.86e+04

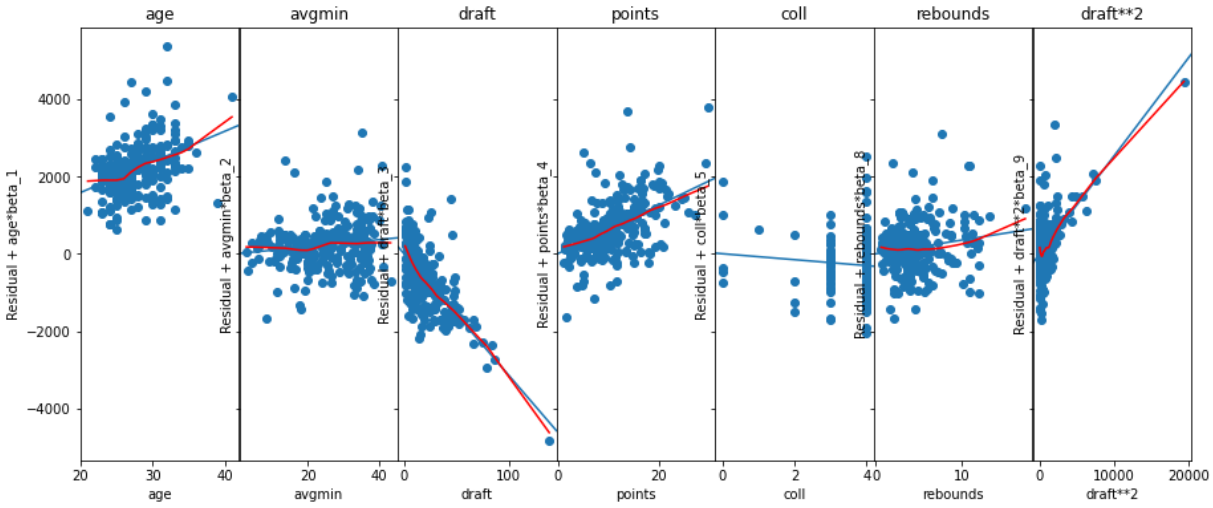
Notes:

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

[2] The condition number is large, 1.86e+04. This might indicate that there are strong multicollinearity or other numerical problems.

Comparing to the simple linear regression model, adding a higher order variable slightly imporved our model, the R^2 increased from 0.560 to 0.596, which means the quadratic regression model can better explain the changes in wage rather than the linear regression model. The value of AIC and BIC are relatively lower than before, which also means the model has been improved.

In [31]:
```
# Evaluate transformations of variables
from statsmodels.graphics.regressionplots import add_lowess
fig, axs = plt.subplots(1,7, figsize=(15, 6), facecolor='w', edgecolor='k',sharey =
fig.subplots_adjust(hspace = .5, wspace=.001)
axs = axs.ravel()
predictor_category = ['age','avgmin','draft','points','coll','rebounds','draft**2']
for i in range(7):
    sm.graphics.plot_ccpr(result1, predictor_category[i],ax = axs[i])
    axs[i].set_title(str(predictor_category[i]))
    add_lowess(axs[i],frac = 0.5)
```



In [32]:
```
# Test for multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
# VIF dataframe
pd.Series([variance_inflation_factor(x.values, i)
 for i in range(x.shape[1])],
 index=x.columns)
```

Out[32]:
```
const       102.224640
age           1.081570
avgmin        6.365379
draft         5.614923
points        4.690496
coll          1.055578
forward       2.166194
center        2.171040
rebounds      3.485804
draft**2      5.131260
dtype: float64
```

The outcome of variance inflation factor indicated that there is a multicollinearity relationship between minutes played per game, draft, $draft^2$ and points per game. Theoretically, dropping these variables would make our model better.

However, we decided not to drop any of them because draft, avgmin and points are important references for people to evaluate a player in reality, also, draft and $draft^2$ would naturally have some kind of correlation.

In [33]:
```
# Test for heteroskedasticity
import statsmodels.stats.api as sms
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
test = sms.het_breuschpagan(result1.resid, result1.model.exog)
print(blue("BP Results:",['bold']))
print(list(zip(name, test)))
```

```
BP Results:
[('Lagrange multiplier statistic', 20.01913797070582), ('p-value', 0.017794663815773
84), ('f-value', 2.325657731610197), ('f p-value', 0.015959074757628304)]
```

From above BP test, we see that p value is small, we reject the null hypothesis: variance = constant, therefore, this model shows heteroskedasticity.

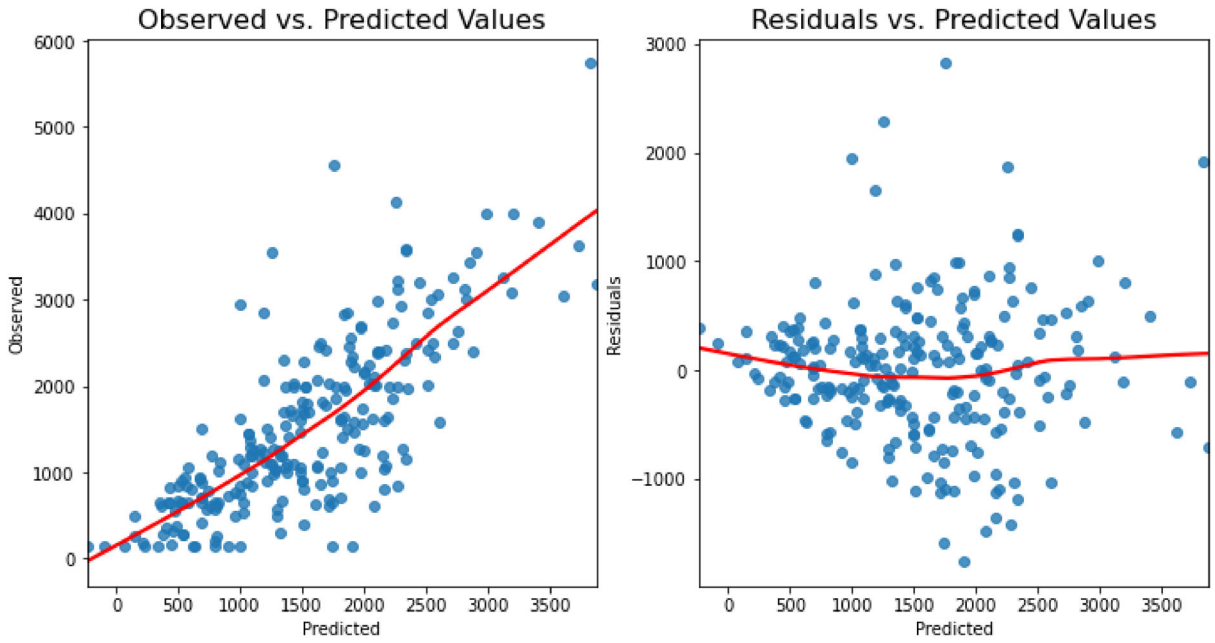
In [34]:
```
# Test for model misspecification
import statsmodels.regression.linear_model as rg
import statsmodels.stats.diagnostic as dg

test =  dg.linear_reset(result1, power=2,  test_type='fitted', use_f = True)

print(blue("Ramsey-RESET:",['bold']))
print(test)
```

```
Ramsey-RESET:
<F test: F=array([[5.46352411]]), p=0.02028017467603946, df_denom=229, df_num=1>
```

From above Ramsey RESET test, for 5% level of confidence, we see that p value is small that we reject the null hypothesis and conclude that this model is not good enough. We should consider improving this model by including quadratic terms or interactions.

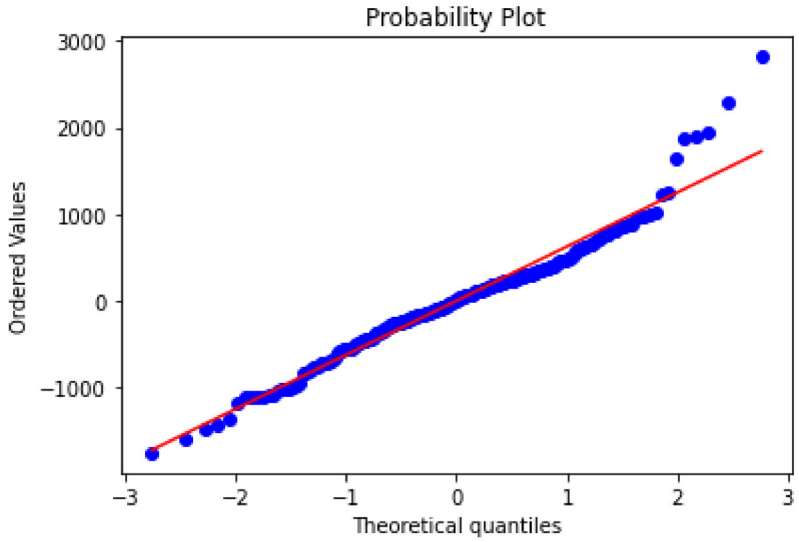
In [35]:
```
# Cook's distance Plot, Residuals Plot, QQ-Plot

fig, ax = plt.subplots(1,2,figsize=(12, 6))
sns.regplot(x=result1.fittedvalues, y=final['wage'], lowess=True, ax=ax[0], line_kws
ax[0].set_title('Observed vs. Predicted Values', fontsize=16)
ax[0].set(xlabel='Predicted', ylabel='Observed')

sns.regplot(x=result1.fittedvalues, y=result1.resid, lowess=True, ax=ax[1], line_kws
ax[1].set_title('Residuals vs. Predicted Values', fontsize=16)
ax[1].set(xlabel='Predicted', ylabel='Residuals')
```

Out[35]:
```
[Text(0.5, 0, 'Predicted'), Text(0, 0.5, 'Residuals')]
```



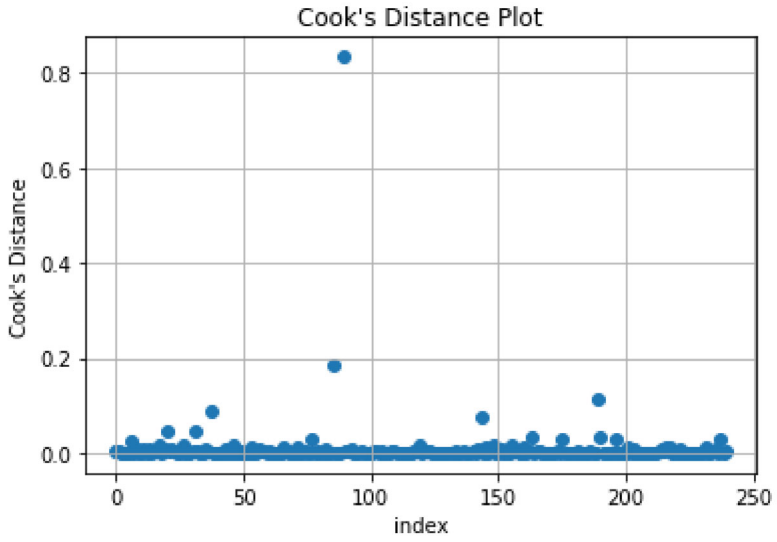
From above, we can see that residuals are not random distributed and the variance is not constant.

In [36]:
```
import scipy as sp
figA, axA = plt.subplots(figsize=(6,4))
_, (__, ___, r) = sp.stats.probplot(result1.resid, plot = axA, fit=True)
```



From above, we can see that large proportion of the residuals of model 1 follow the red line, yet also have some outliers on the top right of the plot.

In [37]:
```
cooks = result1.get_influence().cooks_distance[0]
plt.title("Cook's Distance Plot")
plt.ylabel("Cook's Distance")
plt.xlabel("index")
plt.scatter(range(len(cooks)), cooks)
plt.grid()
```



From above, we can see one outlier, we have discuessed outliers in Q2 and we decided to keep them because remove those are representative points. By including those points, our model shows more accuracy.

In [38]:

```
# bootstrap
# resample with replacement each row
boot_age = []
boot_avgmin = []
boot_center = []
boot_draft = []
boot_points = []
boot_coll = []
boot_forward = []
boot_rebounds = []
boot_draft2 = []
boot_interc = []
boot_adjR2 = []
n_boots = 1000
n_points = df.shape[0]
plt.figure()
for _ in range(n_boots):
 # sample the rows, same size, with replacement
    sample_df = final.sample(n=n_points, replace=True)
 # fit a linear regression
    y = sample_df['wage']
    x = sample_df[['age','avgmin','draft','points','coll','forward','center','reboun
    x["draft**2"] = x["draft"]**2
    x = sm.add_constant(x)
    ols_model_temp = sm.OLS(y,x)
    results_temp = ols_model_temp.fit()


 # append coefficients
    boot_interc.append(results_temp.params[0])
    boot_age.append(results_temp.params[1])
    boot_avgmin.append(results_temp.params[2])
    boot_draft.append(results_temp.params[3])
    boot_points.append(results_temp.params[4])
    boot_coll.append(results_temp.params[5])
    boot_forward.append(results_temp.params[6])
    boot_center.append(results_temp.params[7])
    boot_rebounds.append(results_temp.params[8])
    boot_draft2.append(results_temp.params[9])
```

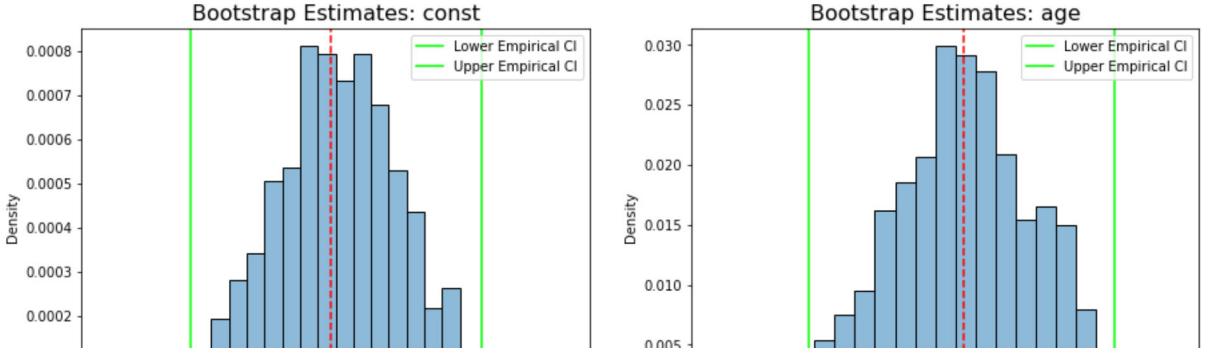
```
    boot_adjR2.append(results_temp.rsquared_adj)
```

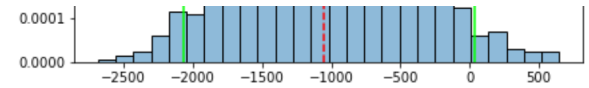
```
<Figure size 432x288 with 0 Axes>
```

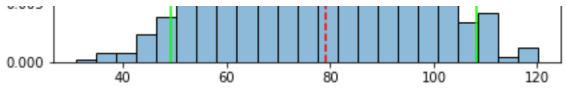
In [39]:
```
def empirical_ci(sample_median, boot_medians, confidence):
    boot_medians = np.array(boot_medians)
    differences = boot_medians - sample_median
    alpha = (1-confidence)/2
    # calculate the lower percentile confidence
    lower = np.percentile(differences, 100*alpha)
    # calculate the upper percentile of confidence
    upper = np.percentile(differences, (1-alpha)*100)
    # Return results
    return(lower+sample_median, upper+sample_median)
```

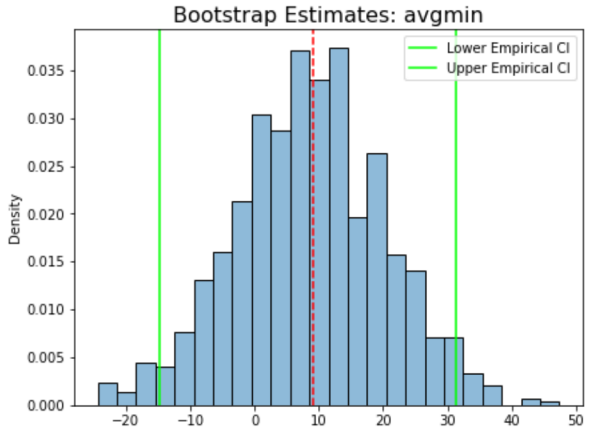
In [43]:
```
results = [boot_interc,boot_age,boot_avgmin,boot_draft,boot_points,boot_coll,boot_fo
name = ['const','age','avgmin','draft','points','coll','forward','center','rebounds'
fig, axes = plt.subplots(5,2, figsize=(15, 30))
axes = axes.ravel()
for i in range(10):

    sns.histplot(results[i], alpha = 0.5, stat="density", ax = axes[i])
    title = 'Bootstrap Estimates: ' + name[i]
    axes[i].set_title(title, fontsize=16)
    axes[i].axvline(x=result1.params[i], color='red', linestyle='--')
    low, up = empirical_ci(result1.params[i], results[i], .95)
    axes[i].axvline(low, color = "lime", label='Lower Empirical CI')
    axes[i].axvline(up, color = "lime", label='Upper Empirical CI')
    axes[i].legend(loc='upper right')
    print('We can expect the coefficient of variable '+ name[i] +' to fall within',s
       'and',str(up),'for 95% of the time.')
```

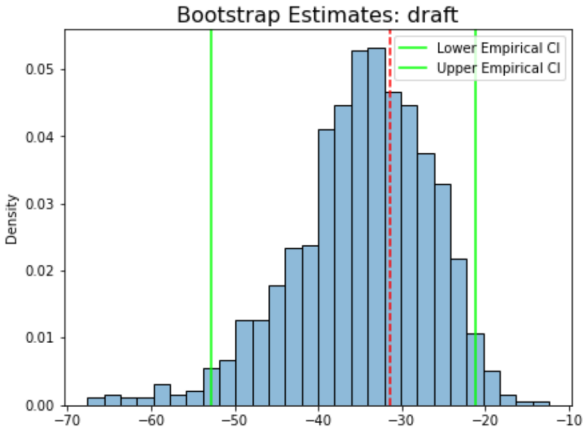
```
We can expect the coefficient of variable const to fall within -2066.223830861866 an
d 34.49213205325486 for 95% of the time.
We can expect the coefficient of variable age to fall within 49.14411294414774 and 1
08.1713156554096 for 95% of the time.
We can expect the coefficient of variable avgmin to fall within -14.857285378505358
and 31.225586610232725 for 95% of the time.
We can expect the coefficient of variable draft to fall within -52.79395344563742 an
d -21.25066990975295 for 95% of the time.
We can expect the coefficient of variable points to fall within 26.39525010632338 an
d 99.20554364066012 for 95% of the time.
We can expect the coefficient of variable coll to fall within -240.17409819459462 an
d 86.74097583533049 for 95% of the time.
We can expect the coefficient of variable forward to fall within -197.10591807489496
and 256.8972682026157 for 95% of the time.
We can expect the coefficient of variable center to fall within -26.9104816982196 an
d 656.3799849763011 for 95% of the time.
We can expect the coefficient of variable rebounds to fall within -21.85569570317303
3 and 83.16194036546693 for 95% of the time.
We can expect the coefficient of variable draft^2 to fall within 0.17240772204363608
and 0.595113798263102 for 95% of the time.
```

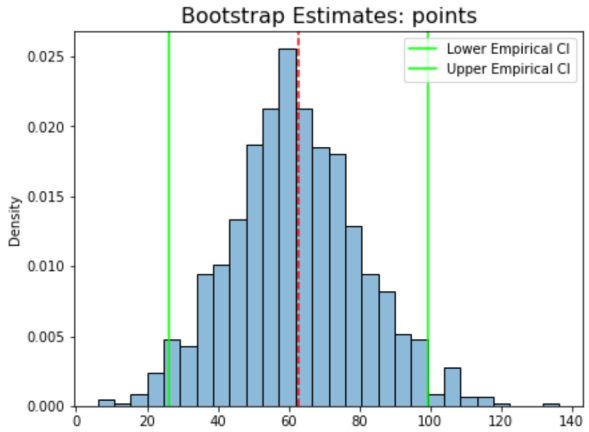


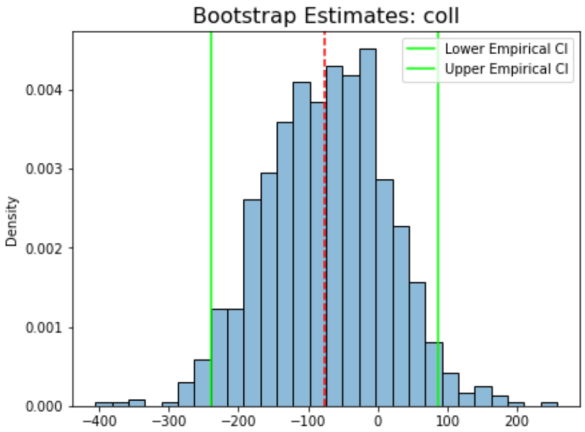


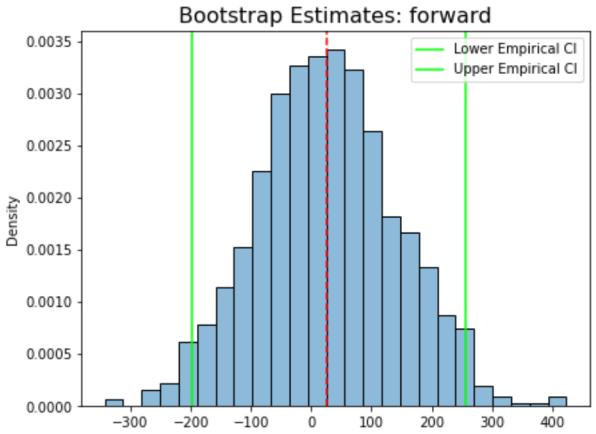


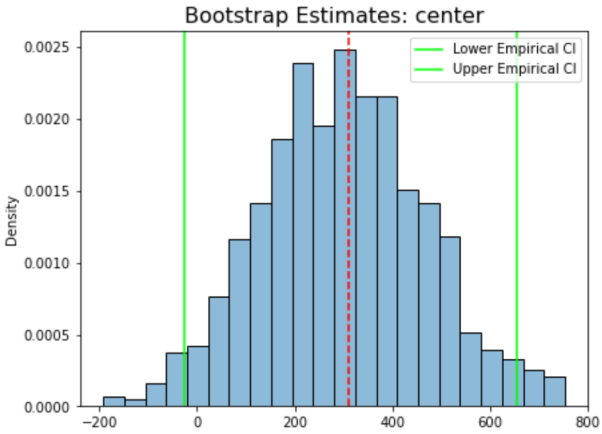


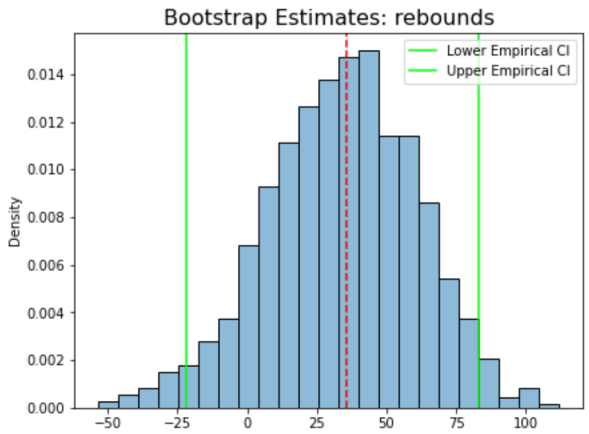


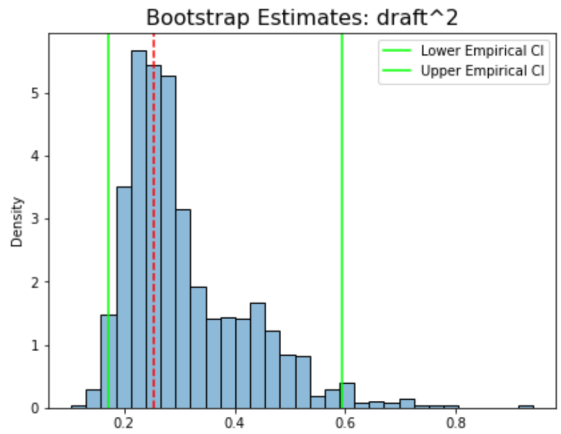












In [45]:

```
# testing set performance and cross-validation performance

from sklearn.model_selection import train_test_split
from sklearn import linear_model
```

```
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_score


y = final['wage']
x = final[['age','avgmin','draft','points','coll','forward','center','rebounds']]
x["draft**2"] = x["draft"]**2

# Perform an OLS fit using all the data
regr = LinearRegression()
model = regr.fit(x,y)
regr.coef_
regr.intercept_

# Split the data into train  (70%)/test(30%) samples:
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_stat

# Train the model:
regr = LinearRegression()
regr.fit(x_train, y_train)

# Make predictions based on the test sample
y_pred = regr.predict(x_test)

# Evaluate Performance

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

# Perform a 5-fold CV
# Use MSE as the scoring function (there are other options as shown here:
# https://scikit-learn.org/stable/modules/model_evaluation.html

regr = linear_model.LinearRegression()
scores = cross_val_score(regr, x, y, cv=5, scoring='neg_root_mean_squared_error')
print('5-Fold CV RMSE Scores:', scores)
```

```
MAE: 477.58230280409384
MSE: 494559.21188411757
RMSE: 703.2490397320977
5-Fold CV RMSE Scores: [-800.29022105 -681.80474776 -556.38124986 -671.99301441 -61
6.15583876]
```

We can see the error in this model is really large.

Model 2

In [47]:

```
# model2:

y = np.log(final['wage'])
x = final[['age','avgmin','draft','points','coll','forward','center','rebounds']]
x = sm.add_constant(x)
model2 = sm.OLS(y,x)
result2 = model2.fit()
result2.summary()
```

Out[47]:

OLS Regression Results

Dep. Variable:	wage	**R-squared:**	0.513
Model:	OLS	**Adj. R-squared:**	0.496

Method:	Least Squares	**F-statistic:**	30.39
Date:	Tue, 09 Nov 2021	**Prob (F-statistic):**	3.21e-32
Time:	17:47:32	**Log-Likelihood:**	-203.08
No. Observations:	240	**AIC:**	424.2
Df Residuals:	231	**BIC:**	455.5
Df Model:	8		
Covariance Type:	nonrobust		

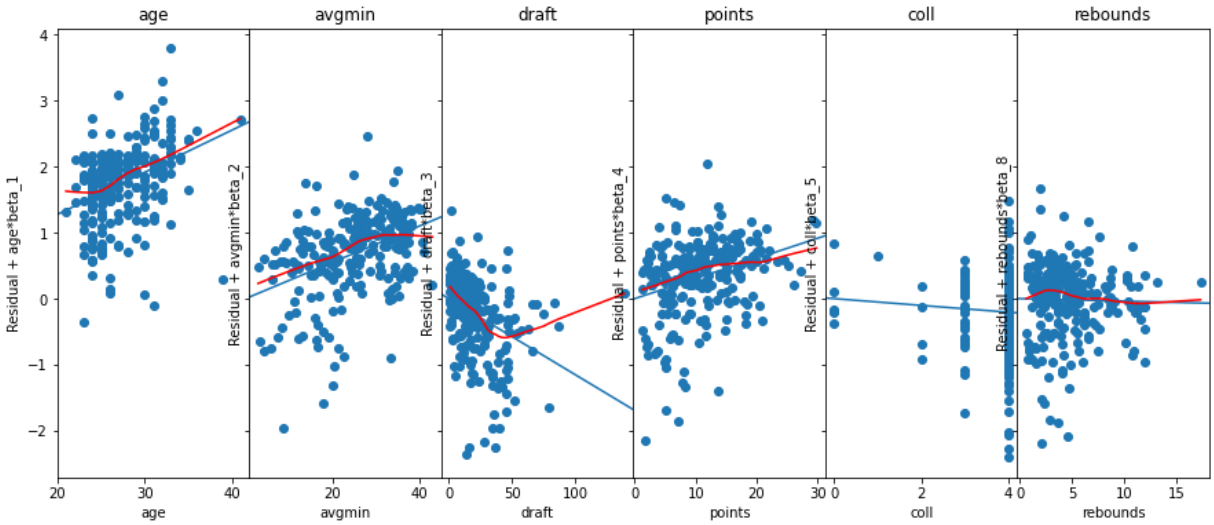
	coef	std err	t	P>|t|	[0.025	0.975]
const	4.6748	0.360	12.986	0.000	3.966	5.384
age	0.0638	0.011	5.758	0.000	0.042	0.086
avgmin	0.0274	0.010	2.747	0.006	0.008	0.047
draft	-0.0115	0.002	-5.290	0.000	-0.016	-0.007
points	0.0305	0.014	2.245	0.026	0.004	0.057
coll	-0.0514	0.053	-0.963	0.336	-0.157	0.054
forward	0.1138	0.111	1.026	0.306	-0.105	0.332
center	0.1311	0.142	0.923	0.357	-0.149	0.411
rebounds	-0.0038	0.023	-0.163	0.871	-0.050	0.042

Omnibus:	40.196	**Durbin-Watson:**	2.108
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	70.489
Skew:	-0.897	**Prob(JB):**	4.94e-16
Kurtosis:	4.957	**Cond. No.**	443.

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

Comparing to model 1, althought the log-linear regression model has a lower R^2 value of 0.513, the outcome of AIC and BIC has significantly decreased. AIC has decreased from 3797 to 424.2, and BIC for model 2 is 455.5 comparing to 3932 in model 1.

In [48]:
```
# Evaluate transformations of variables
from statsmodels.graphics.regressionplots import add_lowess
fig, axs = plt.subplots(1,6, figsize=(15, 6), facecolor='w', edgecolor='k',sharey =
fig.subplots_adjust(hspace = .5, wspace=.001)
axs = axs.ravel()
predictor_category = ['age','avgmin','draft','points','coll','rebounds']
for i in range(6):
    sm.graphics.plot_ccpr(result2, predictor_category[i],ax = axs[i])
    axs[i].set_title(str(predictor_category[i]))
    add_lowess(axs[i],frac = 0.5)
```



From above plots, we can see that plots show more linearity than in model 1, which agree with us on the decision of making dependent variable log. However, we can still add a quadratic term for draft.

In [49]:
```
# Test for multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
# VIF dataframe
pd.Series([variance_inflation_factor(x.values, i)
 for i in range(x.shape[1])],
 index=x.columns)
```

Out[49]:
```
const       94.125041
age          1.071545
avgmin       6.365227
draft        1.202632
points       4.556759
coll         1.055126
forward      2.148854
center       2.154609
rebounds     3.473494
dtype: float64
```

Comparing to model 1, the multicollinearity issue has been improved. As we discussed in model 1, we decided not to delete avgmin or points since they are important parameters to evaluate wages.

In [50]:
```
# Test for heteroskedasticity
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
test = sms.het_breuschpagan(result2.resid, result2.model.exog)
print(blue("BP Results:",['bold']))
print(list(zip(name, test)))
```

```
BP Results:
[('Lagrange multiplier statistic', 23.91130038540748), ('p-value', 0.002371588893294
2493), ('f-value', 3.195163837165393), ('f p-value', 0.0018490086591841793)]
```

From above BP test, we see that p value is small that we reject the null hypothesis: variance = constant. Therefore, this model shows heteroskedasticity.

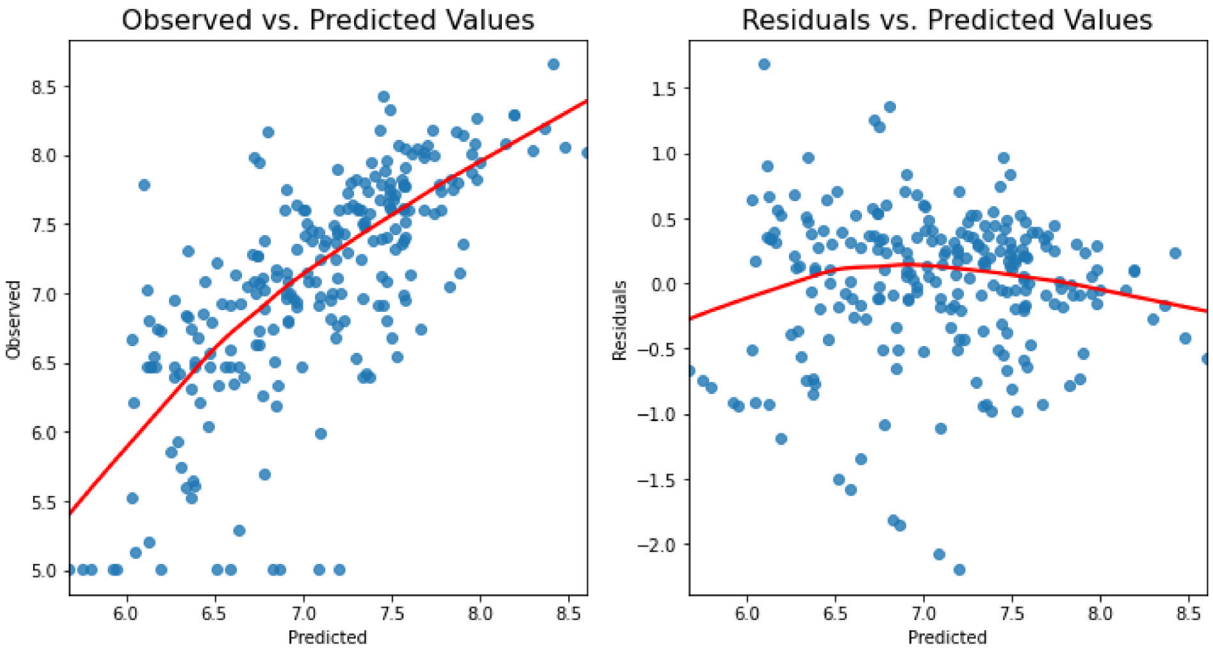
In [51]:
```
# Test for model misspecification
test =  dg.linear_reset(result2, power=2,  test_type='fitted', use_f = True)

print(blue("Ramsey-RESET:",['bold']))
print(test)
```

Ramsey-RESET:
<F test: F=array([[3.89389642]]), p=0.049659206228830015, df_denom=230, df_num=1>

From above Ramsey RESET test, although the p value is larger than p value of model 1 for 5% level of confidence, we see that p value is small than 0.05. We reject the null hypothesis and conclude that this model is not good enough.

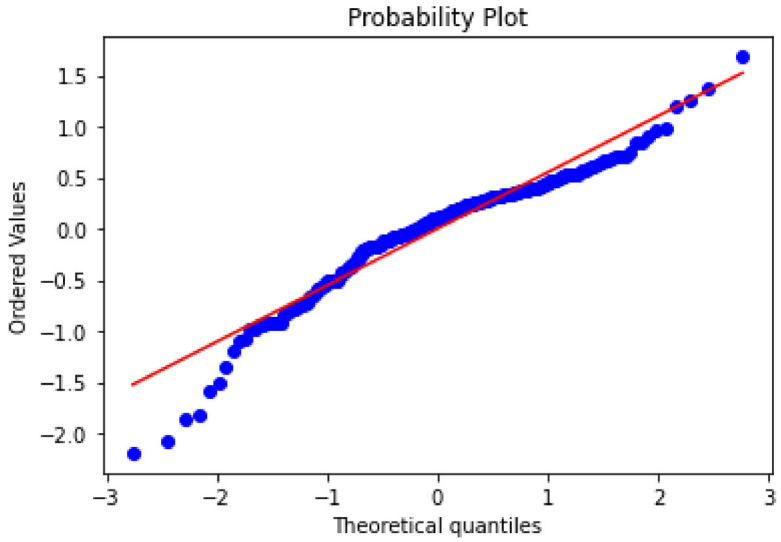
In [52]:
```
# Cook's distance Plot, Residuals Plot, QQ-Plot

fig, ax = plt.subplots(1,2,figsize=(12, 6))
sns.regplot(x=result2.fittedvalues, y=np.log(final['wage']), lowess=True, ax=ax[0],
ax[0].set_title('Observed vs. Predicted Values', fontsize=16)
ax[0].set(xlabel='Predicted', ylabel='Observed')

sns.regplot(x=result2.fittedvalues, y=result2.resid, lowess=True, ax=ax[1], line_kws
ax[1].set_title('Residuals vs. Predicted Values', fontsize=16)
ax[1].set(xlabel='Predicted', ylabel='Residuals')
```

Out[52]: [Text(0.5, 0, 'Predicted'), Text(0, 0.5, 'Residuals')]



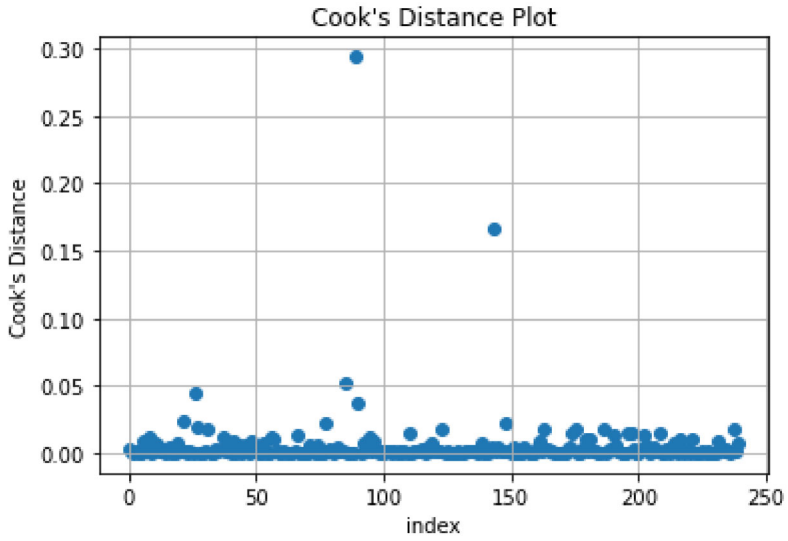
From above, we can see that residuals are not random distributed and the variance is not constant.

In [53]:
```
import scipy as sp
figA, axA = plt.subplots(figsize=(6,4))
_, (__, ___, r) = sp.stats.probplot(result2.resid, plot = axA, fit=True)
```



From above, we can see that middle part of the residuals of model 2 follow the red line, wwe can also see some outliers on the top right and down left of the plot. Comparing the qq plot of model 1 and model 2, the distribution of model 1 seems more normally distributed.

In [54]:
```
cooks = result2.get_influence().cooks_distance[0]
plt.title("Cook's Distance Plot")
plt.ylabel("Cook's Distance")
plt.xlabel("index")
plt.scatter(range(len(cooks)), cooks)
plt.grid()
```



From above, we can see two outliers, we have discuessed outliers in Q2 and we decided to keep them because remove those are representative points. By including those points, our model shows more accuracy.

In [55]:
```
# bootstrap
# resample with replacement each row
boot_age = []
boot_avgmin = []
boot_center = []
boot_draft = []
boot_points = []
boot_coll = []
boot_forward = []
boot_rebounds = []
```

```
#boot_draft2 = []
boot_interc = []
boot_adjR2 = []
n_boots = 1000
n_points = df.shape[0]
plt.figure()
for _ in range(n_boots):
 # sample the rows, same size, with replacement
    sample_df = final.sample(n=n_points, replace=True)
 # fit a linear regression
    y = np.log(sample_df['wage'])
    x = sample_df[['age','avgmin','draft','points','coll','forward','center','reboun
    #x["draft**2"] = x["draft"]**2
    x = sm.add_constant(x)
    ols_model_temp = sm.OLS(y,x)
    results_temp = ols_model_temp.fit()


 # append coefficients
    boot_interc.append(results_temp.params[0])
    boot_age.append(results_temp.params[1])
    boot_avgmin.append(results_temp.params[2])
    boot_draft.append(results_temp.params[3])
    boot_points.append(results_temp.params[4])
    boot_coll.append(results_temp.params[5])
    boot_forward.append(results_temp.params[6])
    boot_center.append(results_temp.params[7])
    boot_rebounds.append(results_temp.params[8])
    #boot_draft2.append(results_temp.params[9])
    boot_adjR2.append(results_temp.rsquared_adj)
```

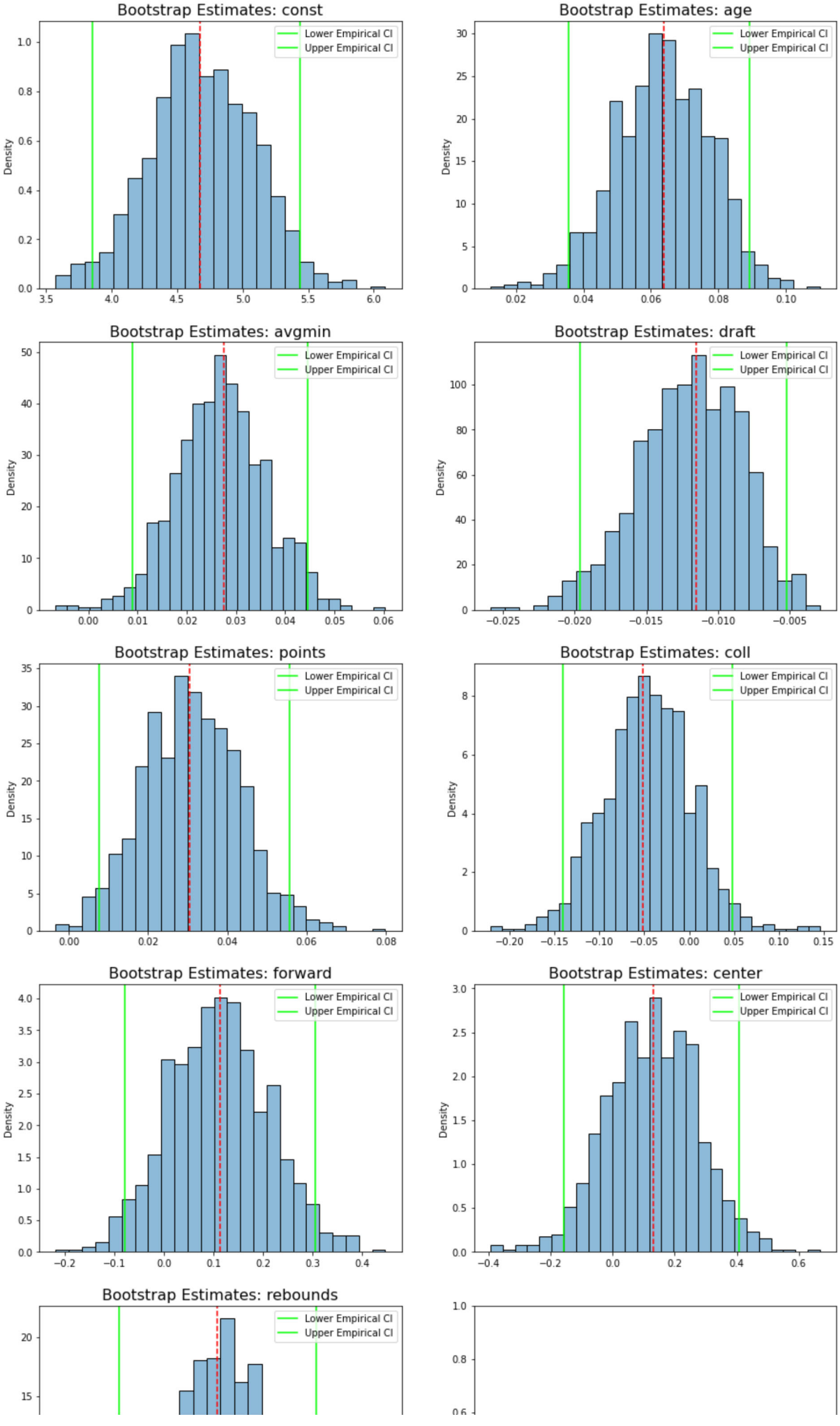
```
<Figure size 432x288 with 0 Axes>
```

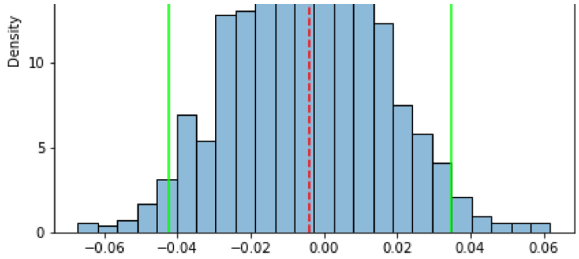
In [56]:

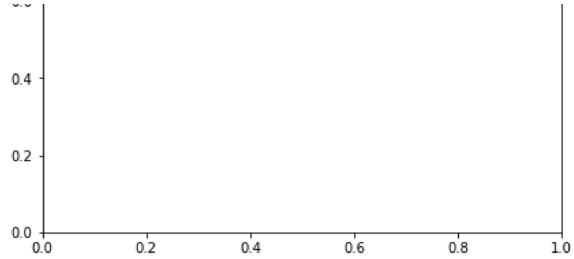
```
results = [boot_interc,boot_age,boot_avgmin,boot_draft,boot_points,boot_coll,boot_fo
name = ['const','age','avgmin','draft','points','coll','forward','center','rebounds'
fig, axes = plt.subplots(5,2, figsize=(15, 30))
axes = axes.ravel()
for i in range(9):
    sns.histplot(results[i], alpha = 0.5, stat="density", ax = axes[i])
    title = 'Bootstrap Estimates: ' + name[i]
    axes[i].set_title(title, fontsize=16)
    axes[i].axvline(x=result2.params[i], color='red', linestyle='--')
    low, up = empirical_ci(result2.params[i], results[i], .95)
    axes[i].axvline(low, color = "lime", label='Lower Empirical CI')
    axes[i].axvline(up, color = "lime", label='Upper Empirical CI')
    axes[i].legend(loc='upper right')
    print('We can expect the coefficient of variable '+ name[i] +' to fall within',s
```

```
We can expect the coefficient of variable const to fall within 3.853056202959648 and
5.436637140340842 for 95% of the time.
We can expect the coefficient of variable age to fall within 0.03571674440031277 and
0.08918577138615706 for 95% of the time.
We can expect the coefficient of variable avgmin to fall within 0.008979304818731278
and 0.04447721157076362 for 95% of the time.
We can expect the coefficient of variable draft to fall within -0.019636322525928153
and -0.0052204476673587226 for 95% of the time.
We can expect the coefficient of variable points to fall within 0.007724159701869716
and 0.055783088281828014 for 95% of the time.
We can expect the coefficient of variable coll to fall within -0.14118876791464308 a
nd 0.04858151908552204 for 95% of the time.
We can expect the coefficient of variable forward to fall within -0.0791105718139859
2 and 0.30522006088705333 for 95% of the time.
We can expect the coefficient of variable center to fall within -0.15616263669723335
and 0.40726464097114484 for 95% of the time.
```

```
We can expect the coefficient of variable rebounds to fall within -0.042552829476918
8 and 0.0345983522457395 for 95% of the time.
```







In [125…

```
# testing set performance and cross-validation performance

from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_score


y = np.log(final['wage'])
x = final[['age','avgmin','draft','points','coll','forward','center','rebounds']]

# Perform an OLS fit using all the data
regr = LinearRegression()
model = regr.fit(x,y)
regr.coef_
regr.intercept_

# Split the data into train  (70%)/test(30%) samples:
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_stat

# Train the model:
regr = LinearRegression()
regr.fit(x_train, y_train)

# Make predictions based on the test sample
y_pred = regr.predict(x_test)

# Evaluate Performance

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

# Perform a 5-fold CV
# Use MSE as the scoring function (there are other options as shown here:
# https://scikit-learn.org/stable/modules/model_evaluation.html

regr = linear_model.LinearRegression()
scores = cross_val_score(regr, x, y, cv=5, scoring='neg_root_mean_squared_error')
print('5-Fold CV RMSE Scores:', scores)
```

```
MAE: 0.40371722355496187
MSE: 0.32042814858940544
RMSE: 0.5660637319148838
5-Fold CV RMSE Scores: [-0.64020699 -0.65140213 -0.5581169  -0.51736431 -0.55125551]
```

Although the log-linear regression model has a significant lower value of AIC and BIC, the outcome of Ramsey-RESET test and Lagrange Multiplier test still suggest that our model can be somehow improved. Therefore, we decided to try out a log-quadratic regression form in model 3.

Model 3

In [57]:
```
# model3:

y = np.log(final['wage'])
x = final[['age','avgmin','draft','points','coll','forward','center','rebounds']]
x["draft**2"] = x["draft"]**2
x = sm.add_constant(x)
model3 = sm.OLS(y,x)
result3 = model3.fit()
result3.summary()
```

Out[57]:

OLS Regression Results

Dep. Variable:	wage	**R-squared:**	0.564
Model:	OLS	**Adj. R-squared:**	0.547
Method:	Least Squares	**F-statistic:**	33.06
Date:	Tue, 09 Nov 2021	**Prob (F-statistic):**	7.19e-37
Time:	17:47:53	**Log-Likelihood:**	-189.74
No. Observations:	240	**AIC:**	399.5
Df Residuals:	230	**BIC:**	434.3
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.1953	0.356	14.608	0.000	4.495	5.896
age	0.0585	0.011	5.545	0.000	0.038	0.079
avgmin	0.0272	0.009	2.873	0.004	0.009	0.046
draft	-0.0321	0.004	-7.191	0.000	-0.041	-0.023
points	0.0190	0.013	1.456	0.147	-0.007	0.045
coll	-0.0569	0.051	-1.124	0.262	-0.157	0.043
forward	0.0647	0.106	0.613	0.540	-0.143	0.273
center	0.0699	0.135	0.517	0.605	-0.196	0.336
rebounds	0.0031	0.022	0.138	0.891	-0.041	0.047
draft2**	0.0002	4.79e-05	5.199	0.000	0.000	0.000

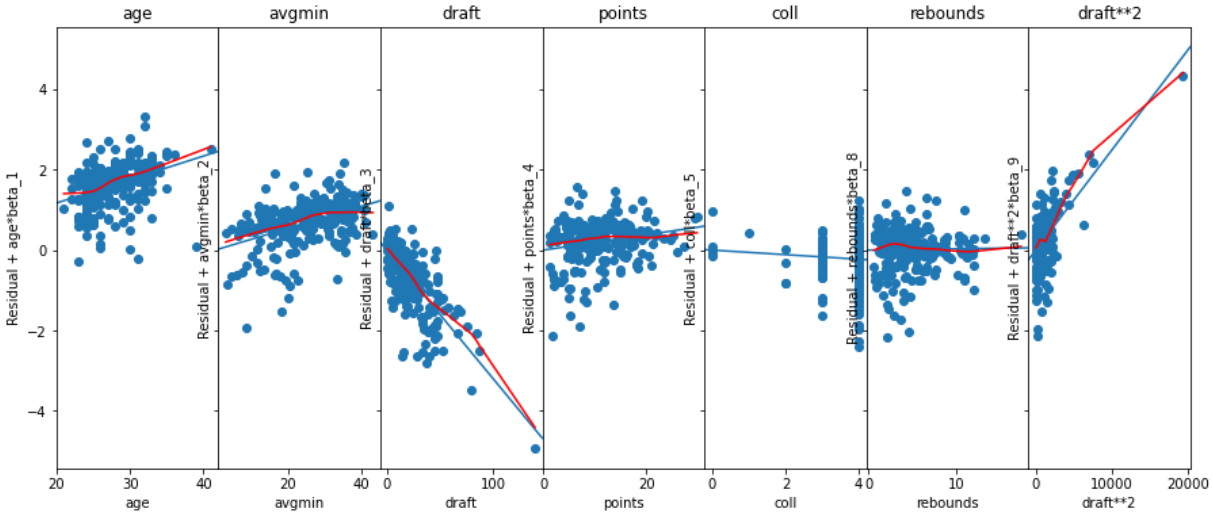
Omnibus:	42.383	**Durbin-Watson:**	1.980
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	77.298
Skew:	-0.924	**Prob(JB):**	1.64e-17
Kurtosis:	5.078	**Cond. No.**	1.86e+04

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

[2] The condition number is large, 1.86e+04. This might indicate that there are strong multicollinearity or other numerical problems.

Comparing to model 2, the outcome of model 3 has higher R^2 value of 0.564 and lower AIC and BIC value.

In [58]:
```
# Evaluate transformations of variables
from statsmodels.graphics.regressionplots import add_lowess
fig, axs = plt.subplots(1,7, figsize=(15, 6), facecolor='w', edgecolor='k',sharey =
fig.subplots_adjust(hspace = .5, wspace=.001)
axs = axs.ravel()
predictor_category = ['age','avgmin','draft','points','coll','rebounds','draft**2']
for i in range(7):
    sm.graphics.plot_ccpr(result3, predictor_category[i],ax = axs[i])
    axs[i].set_title(str(predictor_category[i]))
    add_lowess(axs[i],frac = 0.5)
```



From above plots, we can see that plots show even more linearity than in model 2, which shows log(wage) and draft^ 2 improves the model.

In [59]:
```
# Test for multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor
# VIF dataframe
pd.Series([variance_inflation_factor(x.values, i)
 for i in range(x.shape[1])],
 index=x.columns)
```

Out[59]:
```
const       102.224640
age           1.081570
avgmin        6.365379
draft         5.614923
points        4.690496
coll          1.055578
forward       2.166194
center        2.171040
rebounds      3.485804
draft**2      5.131260
dtype: float64
```

As we stated in model 1, although adding $draft^2$ would increase multicollinearity, we decided to delete neither of them since they both have significant correlation with the dependent variable.

In [60]:
```
# Test for heteroskedasticity
```

```
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
test = sms.het_breuschpagan(result3.resid, result3.model.exog)
print(blue("BP Results:",['bold']))
print(list(zip(name, test)))
```

BP Results:

```
[('Lagrange multiplier statistic', 22.798481897442837), ('p-value', 0.00666511490112
1574), ('f-value', 2.682430011549577), ('f p-value', 0.005552257910645854)]
```

From above BP test, we see that p value is small that we reject the null hypothesis: variance = constant. Therefore, this model shows heteroskedasticity.

In [61]:

```
# Test for model misspecification
test =  dg.linear_reset(result3, power=2,  test_type='fitted', use_f = True)

print(blue("Ramsey-RESET:",['bold']))
print(test)
```

Ramsey-RESET:

```
<F test: F=array([[13.05096209]]), p=0.0003725515717707902, df_denom=229, df_num=1>
```

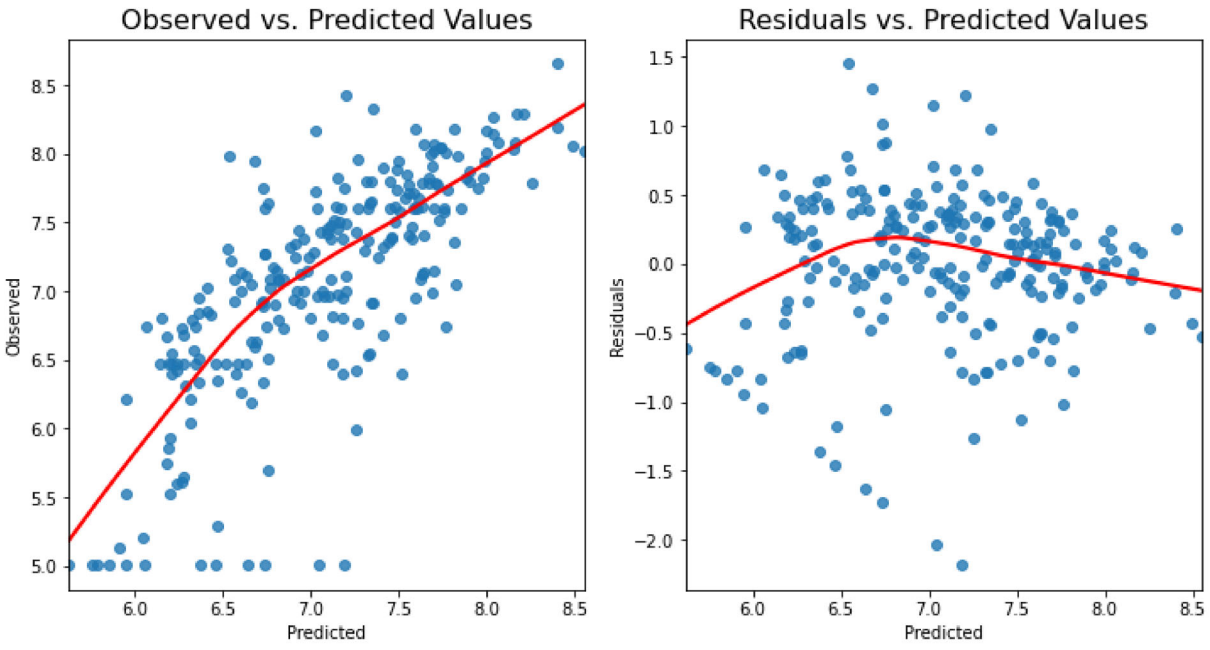
From above Ramsey RESET test, for 5% level of confidence, we can see that p value is so small that we reject tha null hypothesis. We should consider improving this model by including quadratic terms or interactions.

In [62]:

```
# Cook's distance Plot, Residuals Plot, QQ-Plot

fig, ax = plt.subplots(1,2,figsize=(12, 6))
sns.regplot(x=result3.fittedvalues, y=np.log(final['wage']), lowess=True, ax=ax[0],
ax[0].set_title('Observed vs. Predicted Values', fontsize=16)
ax[0].set(xlabel='Predicted', ylabel='Observed')

sns.regplot(x=result3.fittedvalues, y=result3.resid, lowess=True, ax=ax[1], line_kws
ax[1].set_title('Residuals vs. Predicted Values', fontsize=16)
ax[1].set(xlabel='Predicted', ylabel='Residuals')
```

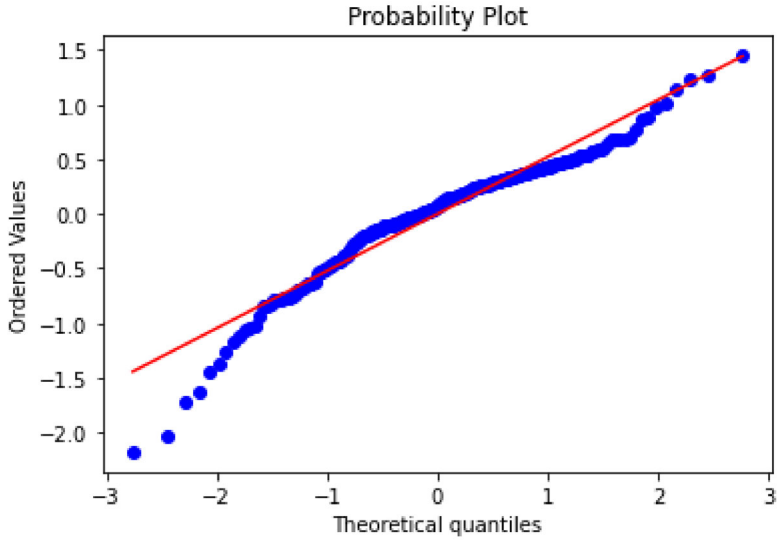
Out[62]:

```
[Text(0.5, 0, 'Predicted'), Text(0, 0.5, 'Residuals')]
```



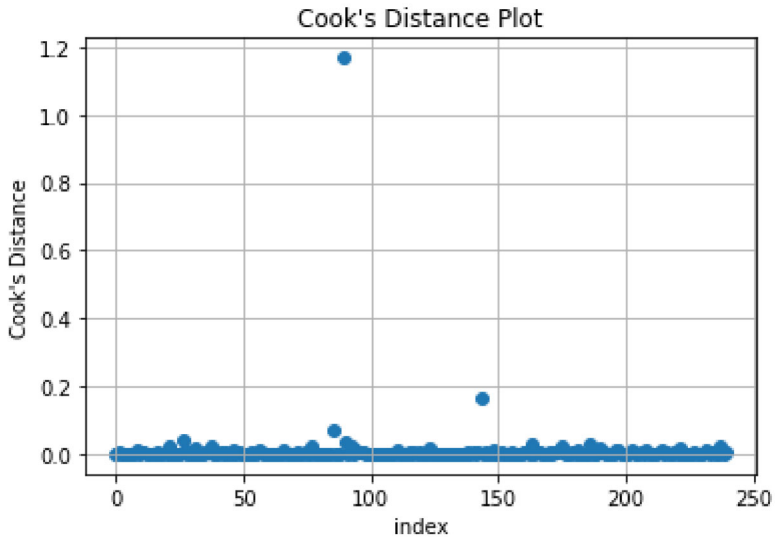
From above, we can see that residuals are not random distributed and the variance is not constant.

In [63]:
```
import scipy as sp
figA, axA = plt.subplots(figsize=(6,4))
_, (__, ___, r) = sp.stats.probplot(result3.resid, plot = axA, fit=True)
```



From above, we can see that majority of the residuals of model 3 follow the red line, we can also see some outliers on the down left of the plot.

In [64]:
```
cooks = result3.get_influence().cooks_distance[0]
plt.title("Cook's Distance Plot")
plt.ylabel("Cook's Distance")
plt.xlabel("index")
plt.scatter(range(len(cooks)), cooks)
plt.grid()
```



From above, we can see one outlier, we have discuessed outliers in Q2 and we decided to keep them because remove those are representative points.

In [65]:
```
# bootstrap
# resample with replacement each row
boot_age = []
boot_avgmin = []
boot_center = []
boot_draft = []
boot_points = []
boot_coll = []
```

```
boot_forward = []
boot_rebounds = []
boot_draft2 = []
boot_interc = []
boot_adjR2 = []
n_boots = 1000
n_points = df.shape[0]
plt.figure()
for _ in range(n_boots):
 # sample the rows, same size, with replacement
    sample_df = final.sample(n=n_points, replace=True)
 # fit a linear regression
    y = np.log(sample_df['wage'])
    x = sample_df[['age','avgmin','draft','points','coll','forward','center','reboun
    x["draft**2"] = x["draft"]**2
    x = sm.add_constant(x)
    ols_model_temp = sm.OLS(y,x)
    results_temp = ols_model_temp.fit()


 # append coefficients
    boot_interc.append(results_temp.params[0])
    boot_age.append(results_temp.params[1])
    boot_avgmin.append(results_temp.params[2])
    boot_draft.append(results_temp.params[3])
    boot_points.append(results_temp.params[4])
    boot_coll.append(results_temp.params[5])
    boot_forward.append(results_temp.params[6])
    boot_center.append(results_temp.params[7])
    boot_rebounds.append(results_temp.params[8])
    boot_draft2.append(results_temp.params[9])
    boot_adjR2.append(results_temp.rsquared_adj)
```

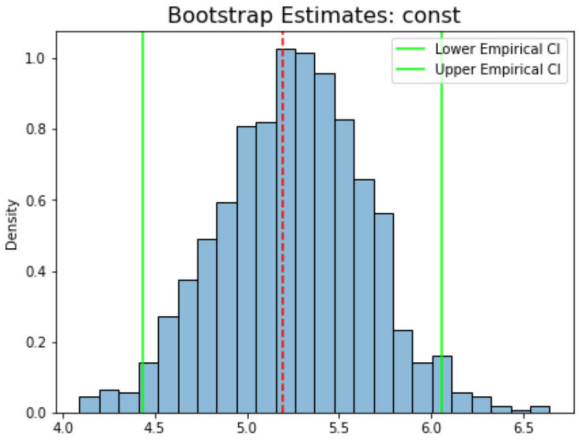
```
<Figure size 432x288 with 0 Axes>
```

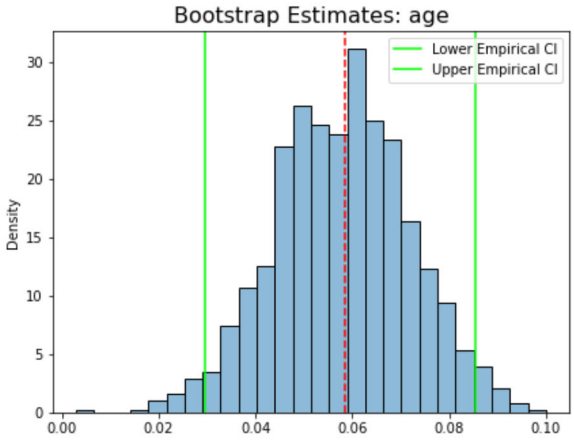
In [67]:

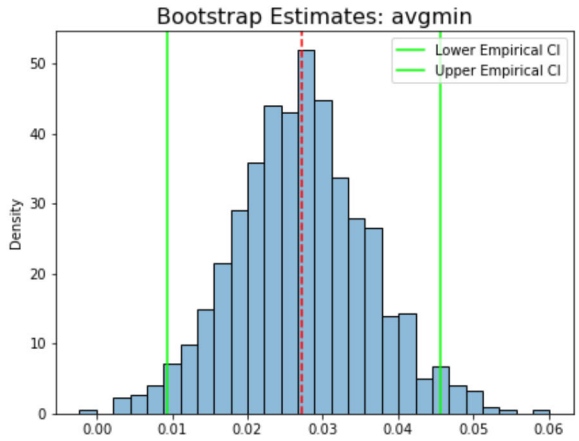
```
results = [boot_interc,boot_age,boot_avgmin,boot_draft,boot_points,boot_coll,boot_fo
name = ['const','age','avgmin','draft','points','coll','forward','center','rebounds'
fig, axes = plt.subplots(5,2, figsize=(15, 30))
axes = axes.ravel()
for i in range(10):
    sns.histplot(results[i], alpha = 0.5, stat="density", ax = axes[i])
    title = 'Bootstrap Estimates: ' + name[i]
    axes[i].set_title(title, fontsize=16)
    axes[i].axvline(x=result3.params[i], color='red', linestyle='--')
    low, up = empirical_ci(result3.params[i], results[i], .95)
    axes[i].axvline(low, color = "lime", label='Lower Empirical CI')
    axes[i].axvline(up, color = "lime", label='Upper Empirical CI')
    axes[i].legend(loc='upper right')
    print('We can expect the coefficient of variable '+ name[i] +' to fall within',s
```

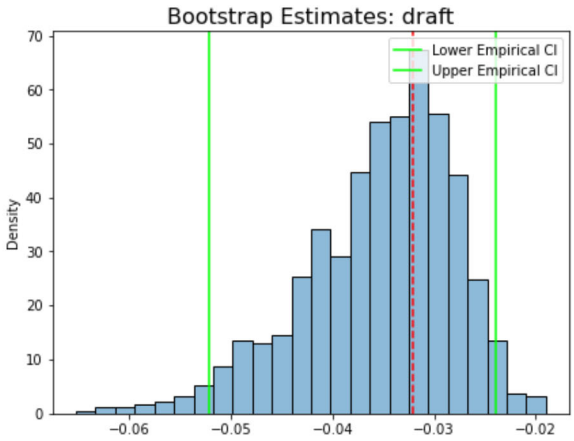
```
We can expect the coefficient of variable const to fall within 4.43622109460328 and
6.0595447796404365 for 95% of the time.
We can expect the coefficient of variable age to fall within 0.029666340219637335 an
d 0.0853512521344814 for 95% of the time.
We can expect the coefficient of variable avgmin to fall within 0.009384629829435737
and 0.0456756439403166 for 95% of the time.
We can expect the coefficient of variable draft to fall within -0.05211109415324658
and -0.023906168597257638 for 95% of the time.
We can expect the coefficient of variable points to fall within -0.00670449294769158
4 and 0.04175060742357946 for 95% of the time.
We can expect the coefficient of variable coll to fall within -0.14231744491188175 a
nd 0.03253556026376159 for 95% of the time.
We can expect the coefficient of variable forward to fall within -0.1090860036666175
3 and 0.24693043983562765 for 95% of the time.
```

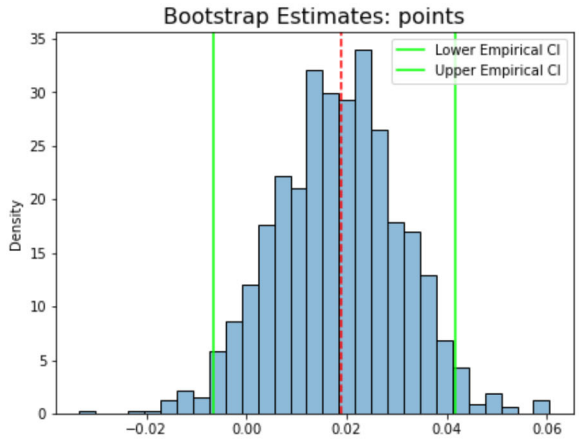
```
We can expect the coefficient of variable center to fall within -0.22673290527208684
and 0.32001780696575916 for 95% of the time.
We can expect the coefficient of variable rebounds to fall within -0.033708752420357
396 and 0.03904313339012437 for 95% of the time.
We can expect the coefficient of variable draft^2 to fall within 0.00018034901438406
436 and 0.0005559620058501159 for 95% of the time.
```

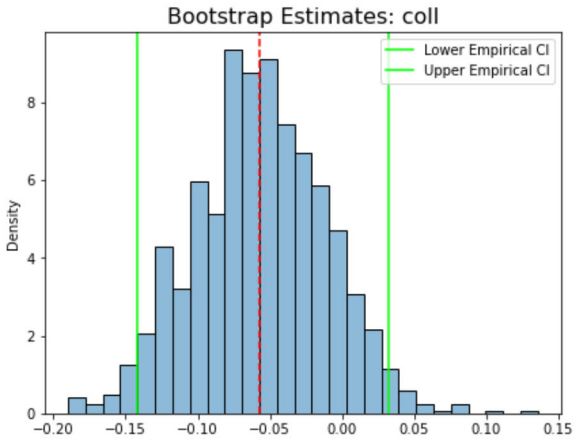


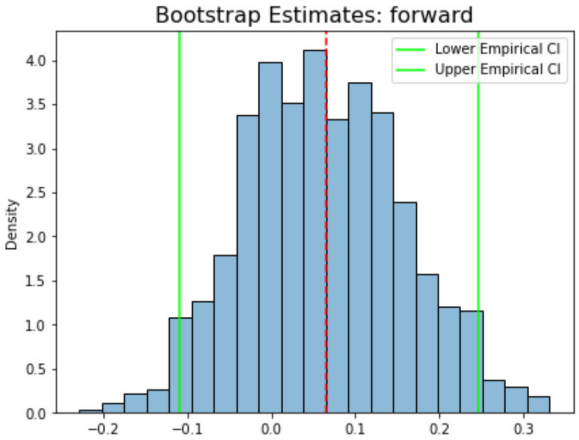


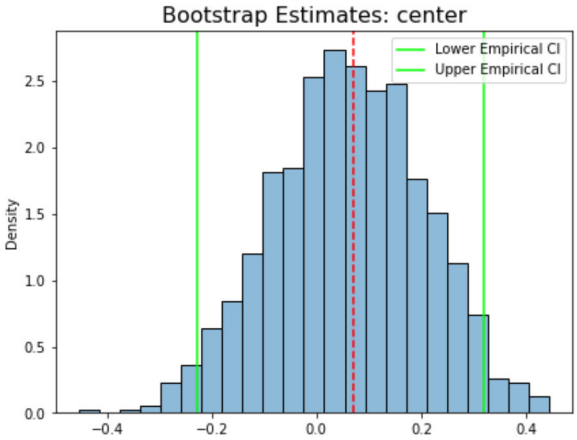


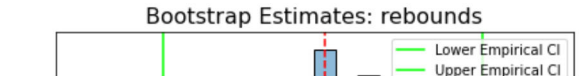


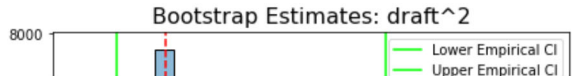


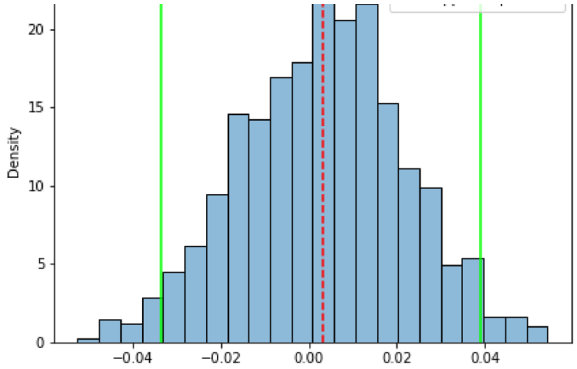


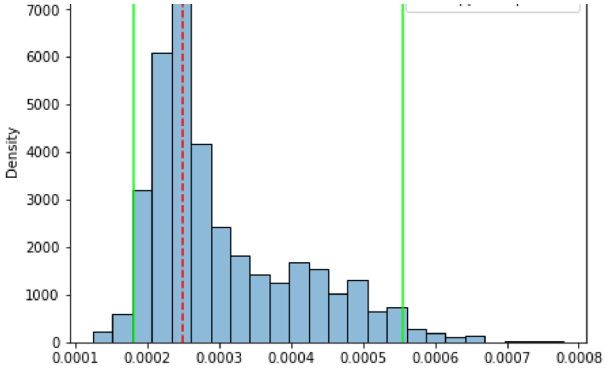












In [136…

```
# testing set performance and cross-validation performance

from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn.model_selection import cross_val_score


y = np.log(final['wage'])
x = final[['age','avgmin','draft','points','coll','forward','center','rebounds']]
x["draft**2"] = x["draft"]**2

# Perform an OLS fit using all the data
regr = LinearRegression()
model = regr.fit(x,y)
regr.coef_
regr.intercept_

# Split the data into train  (70%)/test(30%) samples:
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_stat

# Train the model:
regr = LinearRegression()
regr.fit(x_train, y_train)

# Make predictions based on the test sample
y_pred = regr.predict(x_test)

# Evaluate Performance

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

# Perform a 5-fold CV
# Use MSE as the scoring function (there are other options as shown here:
# https://scikit-learn.org/stable/modules/model_evaluation.html

regr = linear_model.LinearRegression()
scores = cross_val_score(regr, x, y, cv=5, scoring='neg_root_mean_squared_error')
print('5-Fold CV RMSE Scores:', scores)
```

```
MAE: 0.39858366364135633
MSE: 0.32348850710865285
RMSE: 0.5687605006579244
5-Fold CV RMSE Scores: [-0.60341233 -0.62677291 -0.53191821 -0.51843378 -0.52743471]
```

We can see that model 3 has the relatively lowest error out of 3 models shown above.

In conclusion

We decided to use Model 3 as our final regression model since it has the lowest AIC and BIC values.

According to the outcome of Model 3, NBA players' wages has a significant positive relationship with their age and average minutes played per game. This indicates that players with more time on court will receive higher salaries. Also, model 3 suggests that wage is significantly correlated with drafts in a negative way. In other words, Players with lower draft picks are more likely to receive higher salaries. Last but not least, older players tend to receive higher salaries than younger players(In part I, we found there is high multicollinearity between age and experience, so we decided to delete $exper$. Therefore, the coefficient of age could be influenced by experience).

The coefficient on points per game is insignificant, which is due to the fact that we have included average minutes played per game who takes over most of the effect on wage. The coefficient on years played in college, forward, center, and rebounds are also insignificant, meaning these factors are not going to affect salary.

It is worth mentioning that we have tried other models by adding possible quadratic terms or interaction variables with actual meanings, but we failed to get a p value that is larger than .05 for every Ramsey-RESET test. Nonetheless, the Lagrange Multiplier test result for all those potential models also implies heteroskedasticity exists in those models. By this time, we believe that is due to the dataset we chose is not good enough, it may lack some important variables which could explain other factors that are correlated to players' wage. In fact, when NBA tries to select the most valuable player of a season, the association looks at various categories of statistics.Common sense tells us that a player's salary will never only be determined by age, average time per game and his draft postion. For instance, this dataset from Wooldridge did not collect the data on blocks or steals, which are very important when we evaluate a player's performance. What's more, players in different positions will focus on different fields, guards tend to have higher steals than forwards and centers, and centers will obviously have more rebounds and blocks than guards. All these factors will potentially affect our model's outcome, but we are constrained by the data and cannot further improve it.

What's more, the rules of the NBA draft also changed in 1989. Before 1989,NBA draft every year has 7 rounds, each has 20 picks, which gives 140 picks per year. After 1989, It has narrowed to 2 rounds, 30 picks per round, 60 drafts in total. Because of this, the distribution of Draft in our dataset is strongly skewed, and it certainly will influence the correlation between draft and wages.

Overall, based on our model, in order to be better paid, we would recommend NBA players to:

> 1. be better prepared before the draft begins so that they can get picked eariler(smaller draft number) since every draft position ahead will increase annual salaries, and smaller draft has an increasing effect on percentage change in wage.
> 2. try to play as long as possible in every game. For every additional minute of average time per game, a player's annual salary will increase by 27.2 percent.
> 3. retire as late as possible since each younger player will earn 58.5 percent per year less than a player who is one year older.