predicting_MLB_player_salary

July 15, 2018

1 Predicting Major League Baseball (MLB) Player Salaries

1.1 I. Introduction

This project builds a model using XGBoost to predict Major League Baseball (MLB) player salaries. The model developed here is for fielding players with batting statistics. Pitchers can also be modeled in a similar fashion, but the focus here is on predicting salaries using batting statistics. Several models are developed as we move from OLS regression to XGBoost. As various models are developed, using better features and better prediction methods, the adjusted R-squared value increases considerably. The baseline OLS regression model produces an R-squared value of 0.45, but with better features tops out at 0.79. Both XGBoost models are able to push the R-squared value up to around 0.9. One of these XGBoost models in particular represents an intuitive understanding of the factors that are driving player salaries.

Being able to predict MLB player salaries would help in determining a player's value as well as provide information about what factors drive value creation. This would help in salary negotiations, determining team budgets, and finding out which players may be under- or overvalued.

All data for this project come from the Lahman Baseball Database.

The client for this project is the MLB teams and their organizations.

1.2 II. Data

In [1]:
```
%matplotlib inline
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split
import math
from sklearn.metrics import explained_variance_score

import statsmodels.api as sm
from statsmodels.graphics.gofplots import ProbPlot

import plotly.plotly as py
import plotly.graph_objs as go
import plotly.figure_factory as ff
```

```
plt.style.use('seaborn') # pretty matplotlib plots
pd.set_option('display.width', 700)
pd.set_option('display.max_columns', 100)
pd.set_option('display.notebook_repr_html', True)

plt.rcParams['figure.dpi'] = 115
plt.rc('font', size=12)
plt.rc('figure', titlesize=16)
plt.rc('axes', labelsize=13)
plt.rc('axes', titlesize=16)

%config InlineBackend.figure_format = 'retina'
```

```
/Users/jeff/anaconda3/lib/python3.6/site-packages/statsmodels/compat/pandas.py:56: FutureWarnin
  from pandas.core import datetools
```

First, we import the data that we will use for this analysis. The batting, pitching, salary, and all star data come from the Lahman Baseball Database (http://www.seanlahman.com/baseball-archive/statistics/). The consumer price index data comes from the Bureau of Labor Statistics (https://www.bls.gov/cpi/).

In [2]:
```
batting = pd.read_csv('Batting.csv')
pitching = pd.read_csv('Pitching.csv')
salaries = pd.read_csv('Salaries.csv')
all_star_full = pd.read_csv('AllStarFull.csv')
cpi = pd.read_csv('CPI.csv')
```

1.2.1 Batting Data

In [3]:
```
batting.head()
```

Out[3]:

	playerID	yearID	stint	teamID	lgID	G	AB	R	H	2B	3B	HR	RBI	SB	CS
0	abercda01	1871	1	TRO	NaN	1	4	0	0	0	0	0	0.0	0.0	0.0
1	addybo01	1871	1	RC1	NaN	25	118	30	32	6	0	0	13.0	8.0	1.0
2	allisar01	1871	1	CL1	NaN	29	137	28	40	4	5	0	19.0	3.0	1.0
3	allisdo01	1871	1	WS3	NaN	27	133	28	44	10	2	2	27.0	1.0	1.0
4	ansonca01	1871	1	RC1	NaN	25	120	29	39	11	3	0	16.0	6.0	2.0

In [4]:
```
batting.describe()
```

Out[4]:

	yearID	stint	G	AB	R
count	102816.000000	102816.000000	102816.000000	102816.000000	102816.000000
mean	1964.262313	1.077838	51.343439	141.905511	18.815544
std	38.856297	0.284366	47.121658	184.654492	28.242983
min	1871.000000	1.000000	1.000000	0.000000	0.000000
25%	1934.000000	1.000000	13.000000	4.000000	0.000000
50%	1973.000000	1.000000	34.000000	49.000000	4.000000
75%	1998.000000	1.000000	80.000000	231.000000	27.000000
max	2016.000000	5.000000	165.000000	716.000000	192.000000

Batting features and their descriptions - http://m.mlb.com/glossary/

- playerID - Player ID code
- yearID - Year
- stint - player's stint: Order of appearances within a season
- teamID - Team, a factor
- lgID - League, a factor with levels AA AL FL NL PL UA
- G (Games Played) - A player is credited with having played a game if he appears in it at any point – be it as a starter or a replacement.
- AB (At-bat) - An official at-bat comes when a batter reaches base via a fielder's choice, hit or an error (not including catcher's interference) or when a batter is put out on a non-sacrifice.
- R (Run) - A player is awarded a run if he crosses the plate to score his team a run.
- H (Hit) - A hit occurs when a batter strikes the baseball into fair territory and reaches base without doing so via an error or a fielder's choice.
- 2B (Double) - A batter is credited with a double when he hits the ball into play and reaches second base without the help of an intervening error or attempt to put out another baserunner.
- 3B (Triple) - A triple occurs when a batter hits the ball into play and reaches third base without the help of an intervening error or attempt to put out another baserunner.
- HR (Home Run) - A home run occurs when a batter hits a fair ball and scores on the play without being put out or without the benefit of an error.
- RBI (Runs Batted In) - A batter is credited with an RBI in most cases where the result of his plate appearance is a run being scored.
- SB (Stolen Bases) - A stolen base occurs when a baserunner advances by taking a base to which he isn't entitled.
- CS (Caught Stealing) - A caught stealing occurs when a runner attempts to steal but is tagged out before reaching second base, third base or home plate.
- BB (Walk) - A walk occurs when a pitcher throws four pitches out of the strike zone, none of which are swung at by the hitter. The batter is awarded first base.
- SO (Strikeout) - A strikeout occurs when a pitcher throws any combination of three swinging or looking strikes to a hitter.
- IBB (Intentional Walk) - An intentional walk occurs when the defending team elects to walk a batter on purpose, putting him on first base instead of letting him try to hit.
- HBP (Hit-by-pitch) - A hit-by-pitch occurs when a batter is struck by a pitched ball without swinging at it.
- SH (Sacrifice Bunt) - A sacrifice bunt occurs when a player is successful in his attempt to advance a runner (or multiple runners) at least one base with a bunt.
- SF (Sacrifice Fly) - A sacrifice fly occurs when a batter hits a fly-ball out to the outfield or foul territory that allows a runner to score.
- GIDP (Ground Into Double Play) - A GIDP occurs when a player hits a ground ball that results in multiple outs on the bases.

1.2.2 Salary Data

```
In [5]: salaries.head()
```

```
Out[5]:    yearID teamID lgID   playerID  salary
        0    1985    ATL   NL  barkele01  870000
        1    1985    ATL   NL  bedrost01  550000
```

```
2    1985    ATL    NL   benedbr01   545000
3    1985    ATL    NL    campri01   633333
4    1985    ATL    NL   ceronri01   625000
```

In [6]:
```
salaries.describe()
```

Out[6]:

	yearID	salary
count	26428.000000	2.642800e+04
mean	2000.878727	2.085634e+06
std	8.909314	3.455348e+06
min	1985.000000	0.000000e+00
25%	1994.000000	2.947020e+05
50%	2001.000000	5.500000e+05
75%	2009.000000	2.350000e+06
max	2016.000000	3.300000e+07

1.2.3 All Star Data

In [7]:
```
all_star_full.head()
```

Out[7]:

	playerID	yearID	gameNum	gameID	teamID	lgID	GP	startingPos
0	gomezle01	1933	0	ALS193307060	NYA	AL	1.0	1.0
1	ferreri01	1933	0	ALS193307060	BOS	AL	1.0	2.0
2	gehrilo01	1933	0	ALS193307060	NYA	AL	1.0	3.0
3	gehrich01	1933	0	ALS193307060	DET	AL	1.0	4.0
4	dykesji01	1933	0	ALS193307060	CHA	AL	1.0	5.0

1.3 III. Data Wrangling

1.3.1 Step 1: Remove the pitchers from the batting data.

There are pitchers in the batting data set which need to be removed. The pitchers have very limited batting stats, so it looks like the pitchers earn a salary with out being productive at the plate. Instead, their salary is tied to pitcher productivity and not batting productivity.

In [8]:
```
pitchers = np.unique(pitching.playerID)
pitchers = pd.DataFrame(pitchers)
pitchers.columns = ['playerID']

all_df =pd.merge(batting, pitchers, how='outer', on='playerID', indicator=True)
batting_only = all_df[all_df['_merge'] == 'left_only']
batting_only.describe()
```

Out[8]:

	yearID	stint	G	AB	R	
count	53340.000000	53340.000000	53340.000000	53340.000000	53340.000000	53340.0000
mean	1962.322928	1.074634	70.698856	228.917548	30.815186	61.064
std	38.227846	0.279936	52.078948	199.304174	31.165314	57.548
min	1871.000000	1.000000	1.000000	0.000000	0.000000	0.0000
25%	1932.000000	1.000000	20.000000	43.000000	4.000000	9.0000

```
50%      1970.000000      1.000000     65.000000    174.000000     20.000000     43.0000
75%      1995.000000      1.000000    119.000000    402.000000     51.000000    106.0000
max      2016.000000      5.000000    165.000000    716.000000    192.000000    254.0000
```

1.3.2 Step 2: Remove all years before 1985 from the batting data.

We need to drop the years before 1985 because we do not have salary data before then.

```
In [9]: batting_1985 = batting_only[batting_only.yearID > 1984]
        batting_1985.describe()
```

```
Out[9]:              yearID          stint             G            AB             R
        count  19176.000000  19176.000000  19176.000000  19176.000000  19176.000000  19176.0000
        mean    2000.983156      1.076919     73.817219    234.238736     31.953796     62.2398
        std        9.087406      0.280021     51.619338    198.080972     30.986722     56.8248
        min     1985.000000      1.000000      1.000000      0.000000      0.000000      0.0000
        25%     1993.000000      1.000000     24.000000     52.000000      6.000000     11.0000
        50%     2001.000000      1.000000     69.000000    180.000000     22.000000     45.0000
        75%     2009.000000      1.000000    122.000000    401.000000     52.000000    106.0000
        max     2016.000000      4.000000    163.000000    716.000000    152.000000    240.0000
```

1.3.3 Step 3: Merge the batting data with the salary data.

Next, we merge the batting data with salary data using playerID as the common value for both data frames.

```
In [10]: df = pd.merge(batting_1985, salaries)
         df.describe()
```

```
Out[10]:              yearID          stint             G            AB             R
         count  12412.000000  12412.000000  12412.000000  12412.000000  12412.000000  12412.000
         mean    2000.336046      1.006365     96.374476    314.084837     43.299065     84.167
         std        8.804831      0.085391     46.016913    189.910777     31.432539     56.065
         min     1985.000000      1.000000      1.000000      0.000000      0.000000      0.000
         25%     1993.000000      1.000000     59.000000    144.000000     16.000000     34.000
         50%     2000.000000      1.000000    102.000000    310.000000     38.000000     79.000
         75%     2008.000000      1.000000    138.000000    484.000000     67.000000    131.000
         max     2016.000000      3.000000    163.000000    716.000000    152.000000    240.000
```

1.3.4 Step 4: Remove data where salary is below the minimum salary in 1985.

The minimum salary in 1985 was $60,000. We want to remove any salaries that are below this.

```
In [11]: df = df[df.salary >= 60000]
         df['min_salary'] = df['salary'].groupby(df['yearID']).transform('min')

         df['is_min'] = df.salary - df.min_salary

         #df = df.query('is_min > 0')
         df = df.query('AB > 0') # otherwise AVG cannot be computed
         df.describe()
```

Out[11]:

	yearID	stint	G	AB	R	
count	12395.000000	12395.000000	12395.000000	12395.000000	12395.000000	12395.000
mean	2000.337959	1.006374	96.494474	314.486244	43.352723	84.273
std	8.806370	0.085450	45.923603	189.707727	31.416071	56.022
min	1985.000000	1.000000	1.000000	1.000000	0.000000	0.000
25%	1993.000000	1.000000	59.000000	144.000000	16.000000	34.000
50%	2000.000000	1.000000	103.000000	310.000000	38.000000	79.000
75%	2008.000000	1.000000	138.000000	484.500000	67.000000	131.000
max	2016.000000	3.000000	163.000000	716.000000	152.000000	240.000

1.3.5 Step 5: Add Experience feature and All Star feature.

Creating a experience variable which represents years in the league. This will also serve as a timetrend for each player as well.

In [12]:
```
df['EXP'] = df.groupby('playerID').cumcount()+1
df.sort_values(by=['playerID', 'yearID']).head(10)
```

Out[12]:

	playerID	yearID	stint	teamID	lgID	G	AB	R	H	2B	3B	HR	RBI	SB
8929	abadan01	2006	1	CIN	NL	5	3	0	0	0	0	0	0.0	0.0
7306	abbotje01	1998	1	CHA	AL	89	244	33	68	14	1	12	41.0	3.0
7307	abbotje01	1999	1	CHA	AL	17	57	5	9	0	0	2	6.0	1.0
7308	abbotje01	2000	1	CHA	AL	80	215	31	59	15	1	3	29.0	2.0
7309	abbotje01	2001	1	FLO	NL	28	42	5	11	3	0	0	5.0	0.0
5478	abbotku01	1993	1	OAK	AL	20	61	11	15	1	0	3	9.0	2.0
5479	abbotku01	1994	1	FLO	NL	101	345	41	86	17	3	9	33.0	3.0
5480	abbotku01	1995	1	FLO	NL	120	420	60	107	18	7	17	60.0	4.0
5481	abbotku01	1996	1	FLO	NL	109	320	37	81	18	7	8	33.0	3.0
5482	abbotku01	1997	1	FLO	NL	94	252	35	69	18	2	6	30.0	3.0

Let's also create a dummy variable which represents whether a player was an all-star or not. It will be interesting to compare the salary distributions across all-star and non-all-star players. It will be also interesting to compare the differences of salary growth among these two groups. Let's first inspect the all-star data.

In [13]:
```
all_star_full['allStar'] = 1
all_star = all_star_full[['playerID', 'yearID', 'allStar']]
df = pd.merge(df, all_star, how='left', on=['playerID','yearID'])
df.head()
```

Out[13]:

	playerID	yearID	stint	teamID	lgID	G	AB	R	H	2B	3B	HR	RBI	SB	CS
0	rosepe01	1985	1	CIN	NL	119	405	60	107	12	2	2	46.0	8.0	1.0
1	rosepe01	1986	1	CIN	NL	72	237	15	52	8	2	0	25.0	3.0	0.0
2	staubru01	1985	1	NYN	NL	54	45	2	12	3	0	1	8.0	0.0	0.0
3	perezto01	1985	1	CIN	NL	72	183	25	60	8	0	6	33.0	0.0	2.0
4	perezto01	1986	1	CIN	NL	77	200	14	51	12	1	2	29.0	0.0	0.0

We can see from above that there are NaNs in the allStar column. We need to change the NaNs to zero to accuratley reflect non-all star status for a player in a given year. The ones in this column represent that a player was an all star for a given year.

```
In [14]: df=df.fillna({'allStar':0})
         df.head()
```

Out[14]:

	playerID	yearID	stint	teamID	lgID	G	AB	R	H	2B	3B	HR	RBI	SB	CS
0	rosepe01	1985	1	CIN	NL	119	405	60	107	12	2	2	46.0	8.0	1.(
1	rosepe01	1986	1	CIN	NL	72	237	15	52	8	2	0	25.0	3.0	0.(
2	staubru01	1985	1	NYN	NL	54	45	2	12	3	0	1	8.0	0.0	0.(
3	perezto01	1985	1	CIN	NL	72	183	25	60	8	0	6	33.0	0.0	2.(
4	perezto01	1986	1	CIN	NL	77	200	14	51	12	1	2	29.0	0.0	0.(

1.3.6 Step 6: Adjust salary for inflation.

Okay, now lets adjust salary for inflation. For ease of interpretation, let's use 2016 dollars. We use the consumer price index (CPI) to calculate this.

Merge the salary data and cpi data by year. Use the CPI value to adjust salary to 2016 dollars.

```
In [15]: salary_adj = pd.merge(df, cpi, how='left', on='yearID')
         salary_adj['salary2016'] = (240/salary_adj.CPI)*salary_adj.salary
         salary_adj['min_salary2016'] =(240/salary_adj.CPI)*salary_adj.min_salary
         salary_adj.head()
```

Out[15]:

	playerID	yearID	stint	teamID	lgID	G	AB	R	H	2B	3B	HR	RBI	SB	CS
0	rosepe01	1985	1	CIN	NL	119	405	60	107	12	2	2	46.0	8.0	1.(
1	rosepe01	1986	1	CIN	NL	72	237	15	52	8	2	0	25.0	3.0	0.(
2	staubru01	1985	1	NYN	NL	54	45	2	12	3	0	1	8.0	0.0	0.(
3	perezto01	1985	1	CIN	NL	72	183	25	60	8	0	6	33.0	0.0	2.(
4	perezto01	1986	1	CIN	NL	77	200	14	51	12	1	2	29.0	0.0	0.(

1.4 IV. Exploratory Data Analysis

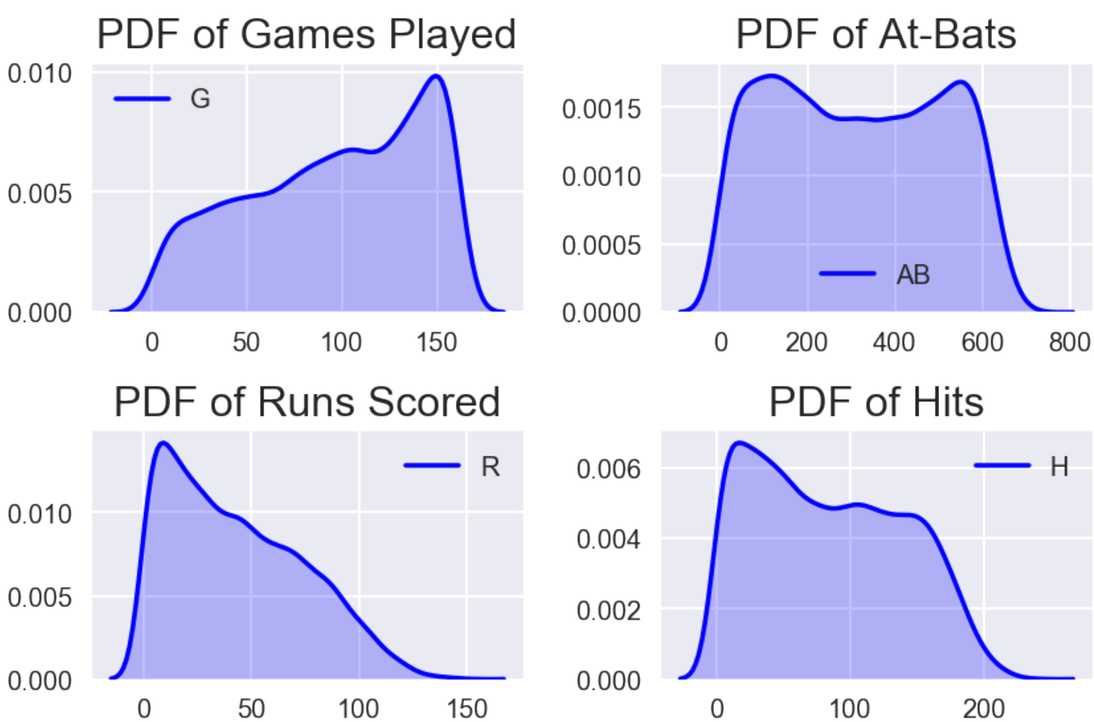
Let's take a deeper look into our data.

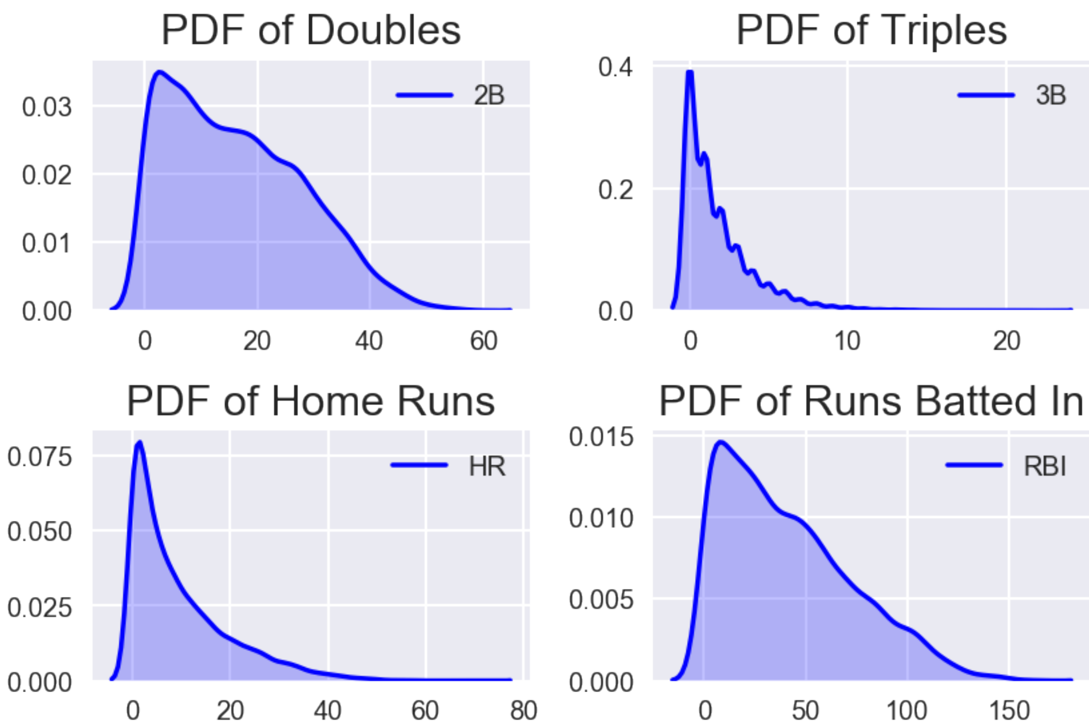
Lets look at the distributions of the target and feature variables.

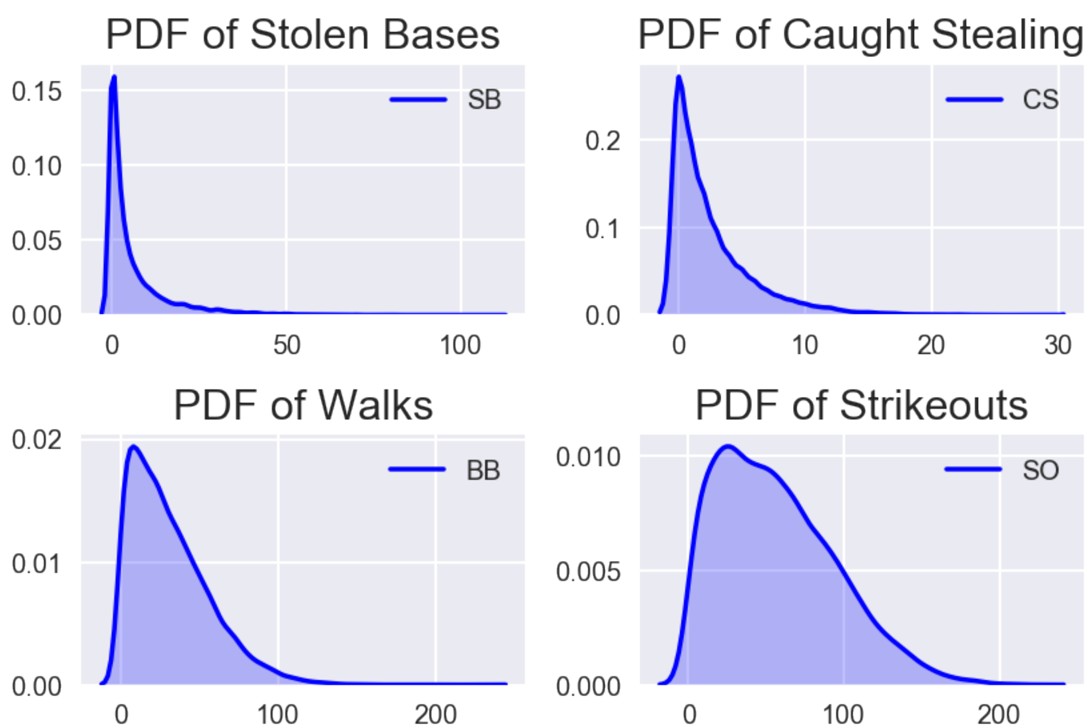
```
In [16]: plt.subplot(2,2,1)
         sns.kdeplot(df.G, shade=True, color="b")
         plt.title("PDF of Games Played")

         plt.subplot(2,2,2)
         sns.kdeplot(df.AB, shade=True, color="b")
         plt.title("PDF of At-Bats")

         plt.subplot(2,2,3)
         sns.kdeplot(df.R, shade=True, color="b")
         plt.title("PDF of Runs Scored")

         plt.subplot(2,2,4)
         sns.kdeplot(df.H, shade=True, color="b")
         plt.title("PDF of Hits")
```

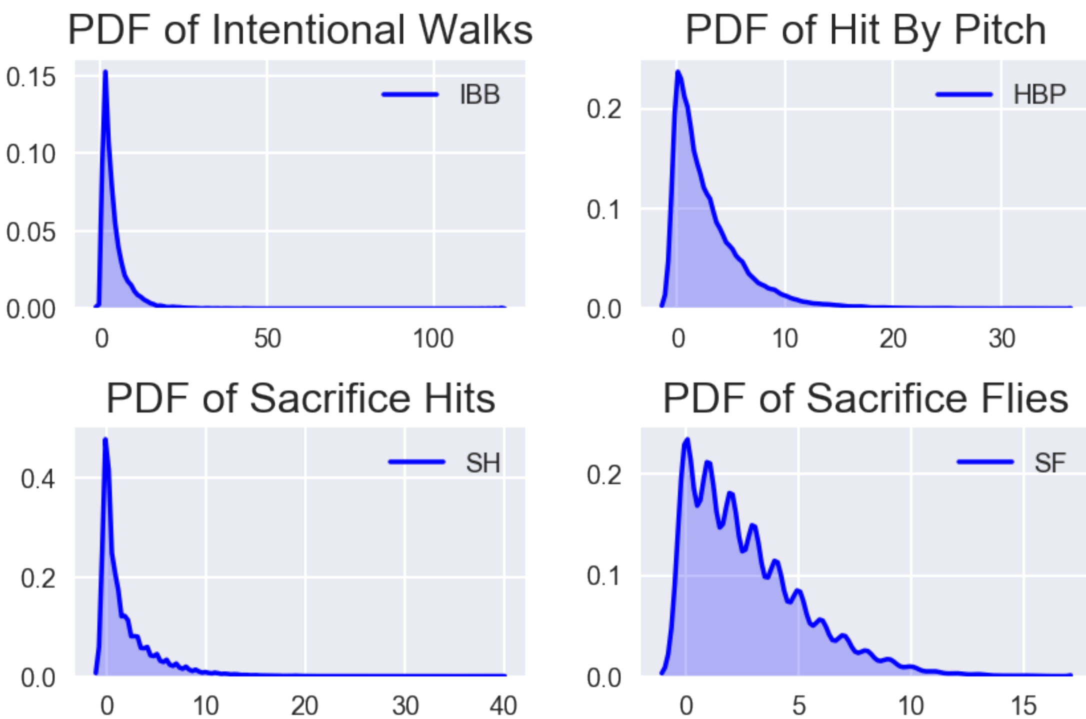
```
plt.tight_layout()
plt.show()

plt.subplot(2,2,1)
sns.kdeplot(df['2B'], shade=True, color="b")
plt.title("PDF of Doubles")

plt.subplot(2,2,2)
sns.kdeplot(df['3B'], shade=True, color="b")
plt.title("PDF of Triples")

plt.subplot(2,2,3)
sns.kdeplot(df.HR, shade=True, color="b")
plt.title("PDF of Home Runs")

plt.subplot(2,2,4)
sns.kdeplot(df.RBI, shade=True, color="b")
plt.title("PDF of Runs Batted In")

plt.tight_layout()
plt.show()

plt.subplot(2,2,1)
sns.kdeplot(df.SB, shade=True, color="b")
plt.title("PDF of Stolen Bases")

plt.subplot(2,2,2)
sns.kdeplot(df.CS, shade=True, color="b")
plt.title("PDF of Caught Stealing")

plt.subplot(2,2,3)
sns.kdeplot(df.BB, shade=True, color="b")
plt.title("PDF of Walks")

plt.subplot(2,2,4)
sns.kdeplot(df.SO, shade=True, color="b")
plt.title("PDF of Strikeouts")

plt.tight_layout()
plt.show()

plt.subplot(2,2,1)
sns.kdeplot(df.IBB, shade=True, color="b")
plt.title("PDF of Intentional Walks")

plt.subplot(2,2,2)
sns.kdeplot(df.HBP, shade=True, color="b")
plt.title("PDF of Hit By Pitch")
```

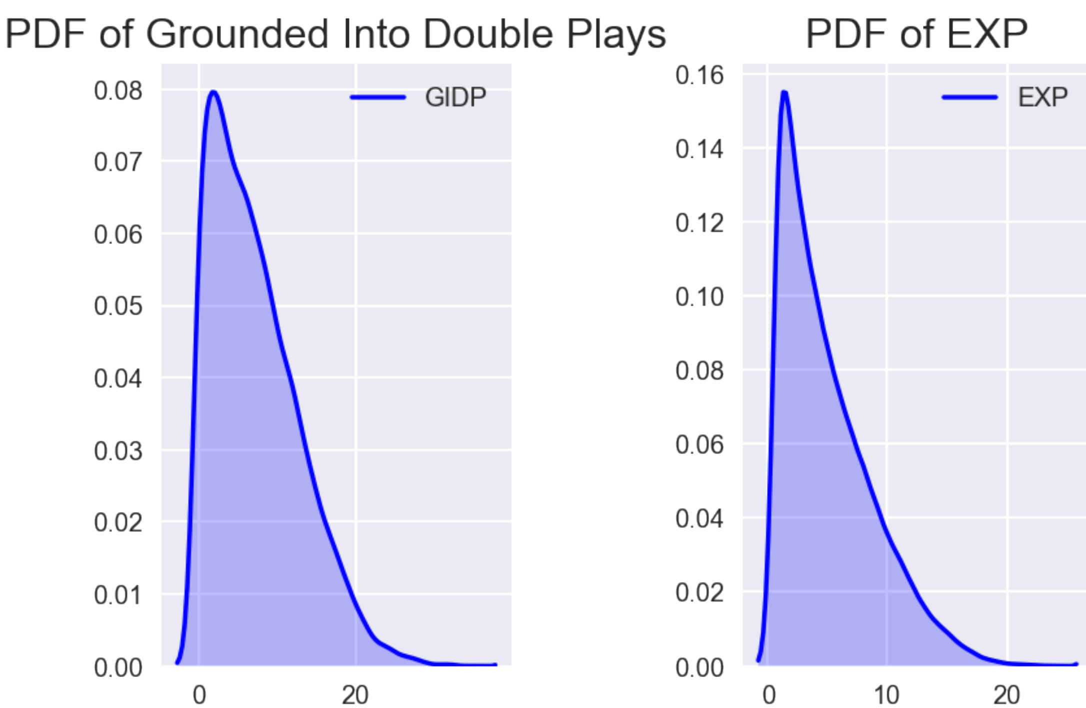
```python
plt.subplot(2,2,3)
sns.kdeplot(df.SH, shade=True, color="b")
plt.title("PDF of Sacrifice Hits")

plt.subplot(2,2,4)
sns.kdeplot(df.SF, shade=True, color="b")
plt.title("PDF of Sacrifice Flies")

plt.tight_layout()
plt.show()

plt.subplot(1,2,1)
sns.kdeplot(df.GIDP, shade=True, color="b")
plt.title("PDF of Grounded Into Double Plays")

plt.subplot(1,2,2)
sns.kdeplot(df.EXP, shade=True, color="b")
plt.title("PDF of EXP")

plt.tight_layout()
plt.show()
```




PDF of Doubles
2B
0.00
0.01
0.02
0.03
0
20
40
60
PDF of Triples
3B
0.0
0.2
0.4
0
10
20
PDF of Home Runs
HR
0.000
0.025
0.050
0.075
0
20
40
60
80
PDF of Runs Batted In
RBI
0.000
0.005
0.010
0.015
0
50
100
150

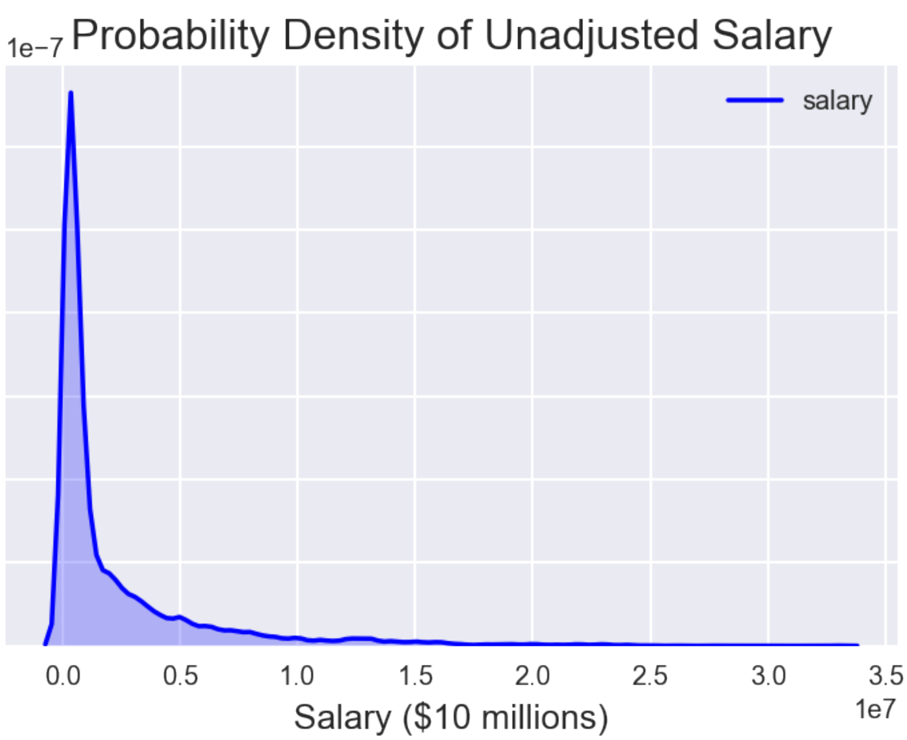

PDF of Stolen Bases
SB
0.00
0.05
0.10
0.15
0
50
100
PDF of Caught Stealing
CS
0.0
0.1
0.2
0
10
20
30
PDF of Walks
BB
0.00
0.01
0.02
0
100
200
PDF of Strikeouts
SO
0.000
0.005
0.010
0
100
200

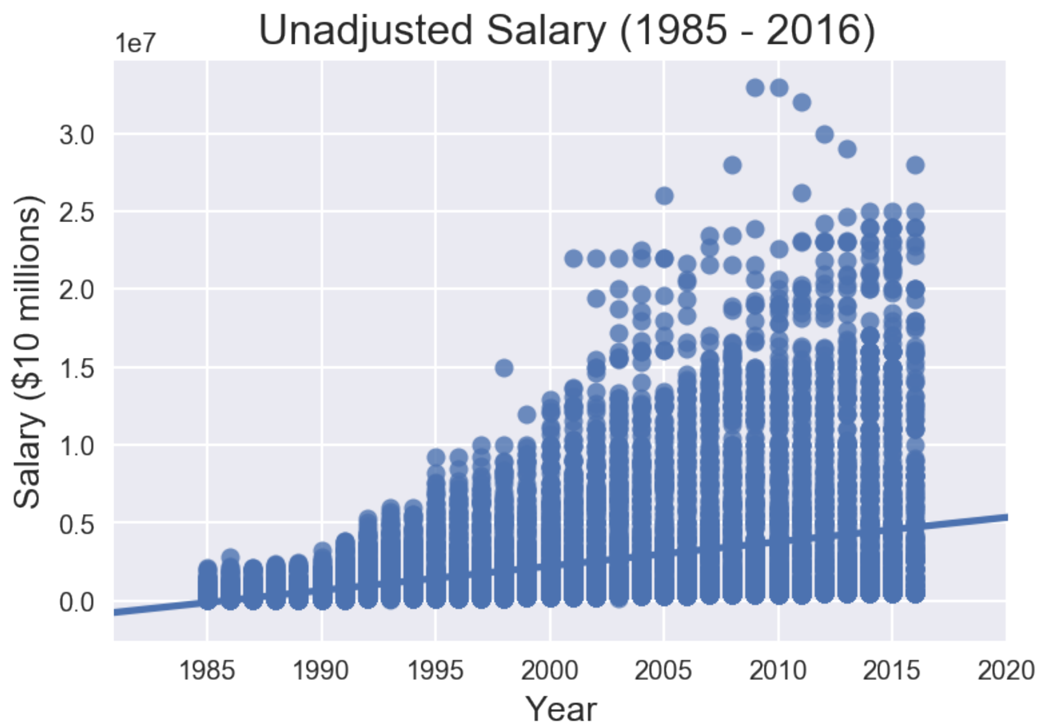

PDF of Intentional Walks
IBB
PDF of Hit By Pitch
HBP
PDF of Sacrifice Hits
SH
PDF of Sacrifice Flies
SF

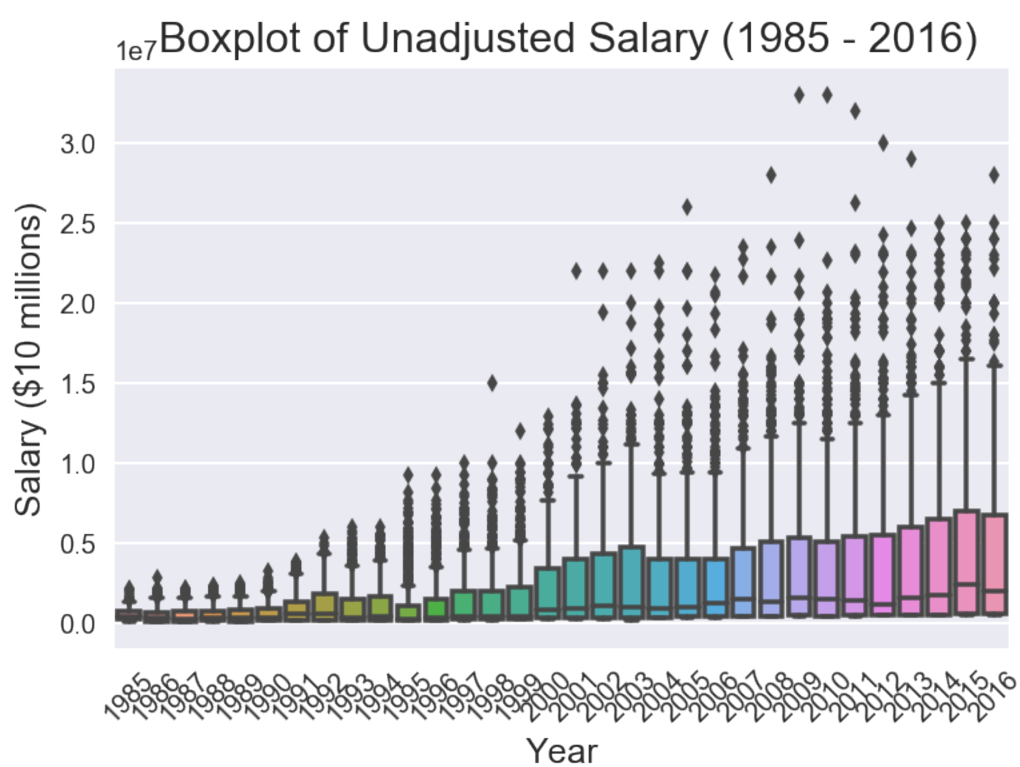

PDF of Grounded Into Double Plays
GIDP
PDF of EXP
EXP

Now let's look at the distribution of the target variable. We have already adjusted salary for inflation, but let's first look at the distibution of unadjusted salary, or salary in nominal terms. Let's also see what the growth of unadjusted salary is doing over time.

```
In [17]: sns.kdeplot(df.salary, shade=True, color="b")
         plt.title("Probability Density of Unadjusted Salary")
         plt.xlabel("Salary ($10 millions)")
         plt.show()

         sns.regplot(x='yearID',
                     y='salary',
                    data=df)
         plt.title(' Unadjusted Salary (1985 - 2016)')
         plt.xlabel('Year')
         plt.ylabel('Salary ($10 millions)')
         plt.show()

         sns.boxplot(x="yearID", y="salary", data=df)
         plt.title(' Boxplot of Unadjusted Salary (1985 - 2016)')
         plt.xlabel('Year')
         plt.ylabel('Salary ($10 millions)')
         plt.xticks(rotation=45)
         plt.show()
```

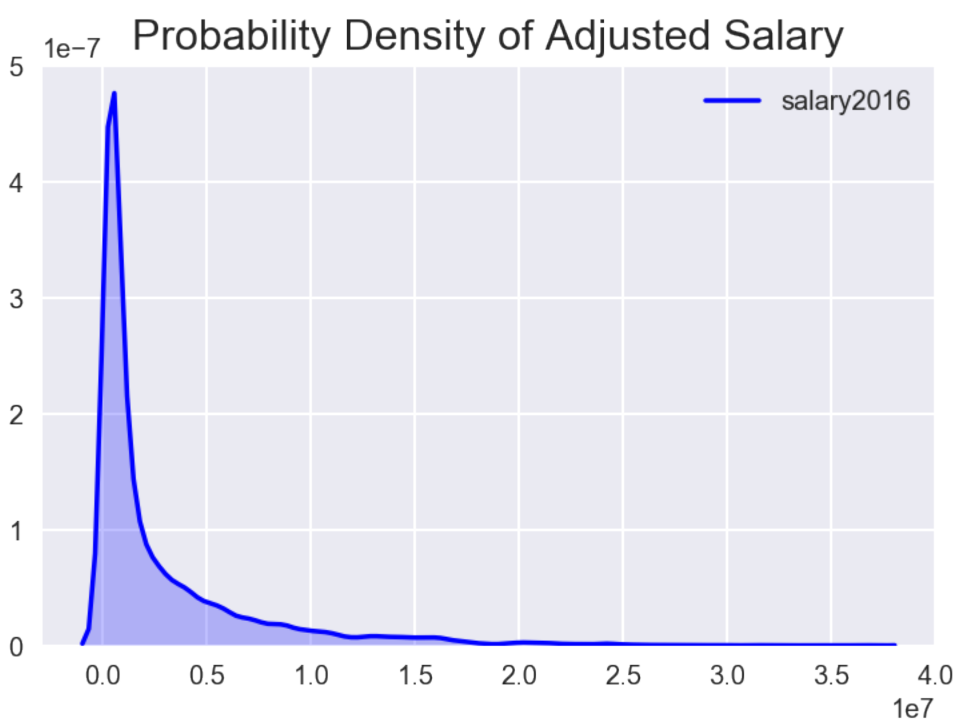


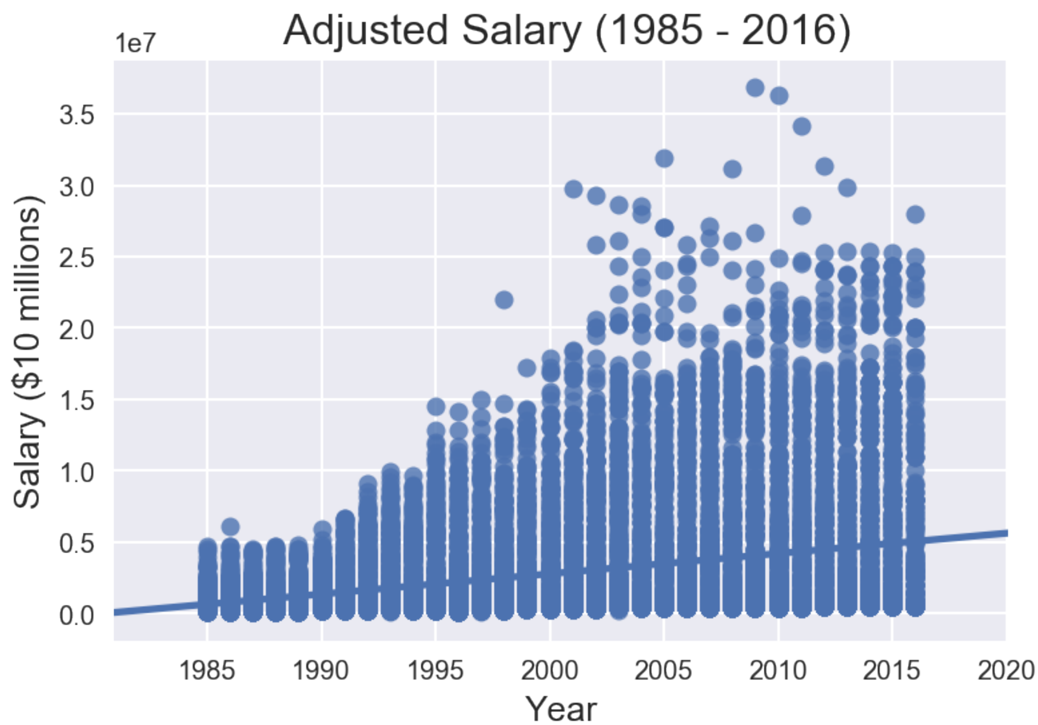

Unadjusted Salary (1985 - 2016)
1e7
Salary ($10 millions)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Year
1985
1990
1995
2000
2005
2010
2015
2020

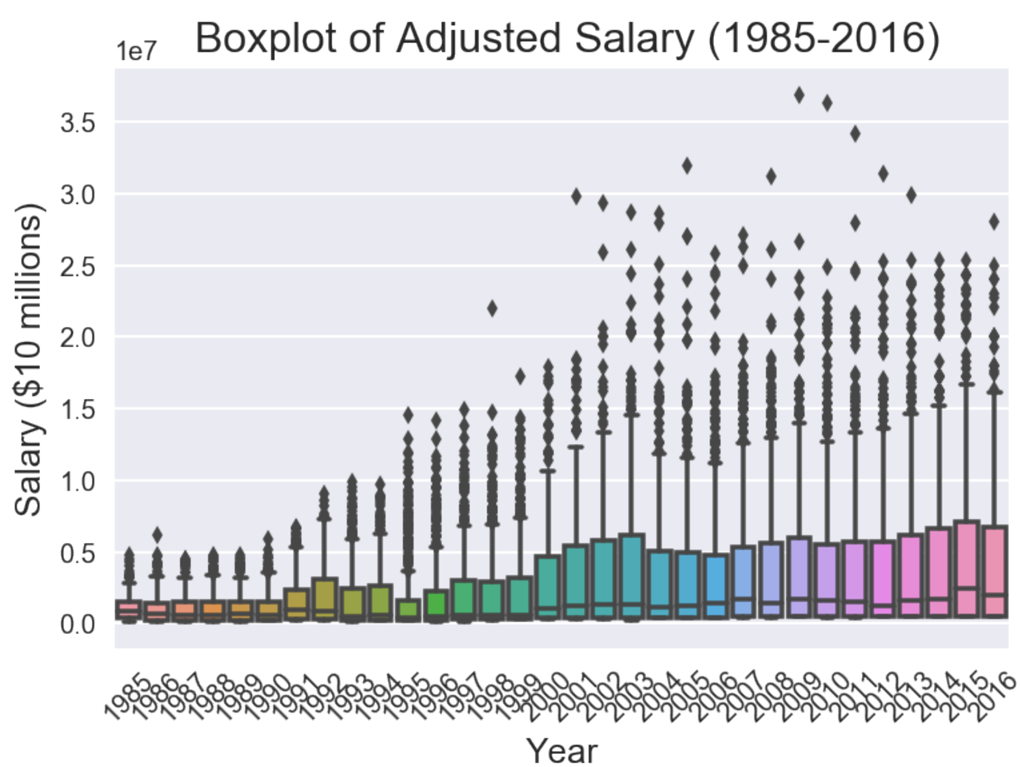

Boxplot of Unadjusted Salary (1985 - 2016)
1e7
Salary ($10 millions)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Year
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016

Now, let's look at the distribution and scatter plot over time for salary in constant 2016 dollars.

```
In [18]: sns.kdeplot(salary_adj.salary2016, shade=True, color="b")
         plt.title("Probability Density of Adjusted Salary")
         plt.xlabel("")
         plt.show()

         sns.regplot(x='yearID',
                     y='salary2016',
                    data=salary_adj)
         plt.title(' Adjusted Salary (1985 - 2016)')
         plt.xlabel('Year')
         plt.ylabel('Salary ($10 millions)')
         plt.show()

         sns.boxplot(x="yearID", y="salary2016", data=salary_adj)
         plt.title(' Boxplot of Adjusted Salary (1985-2016)')
         plt.xlabel('Year')
         plt.ylabel('Salary ($10 millions)')
         plt.xticks(rotation=45)
         plt.show()
```



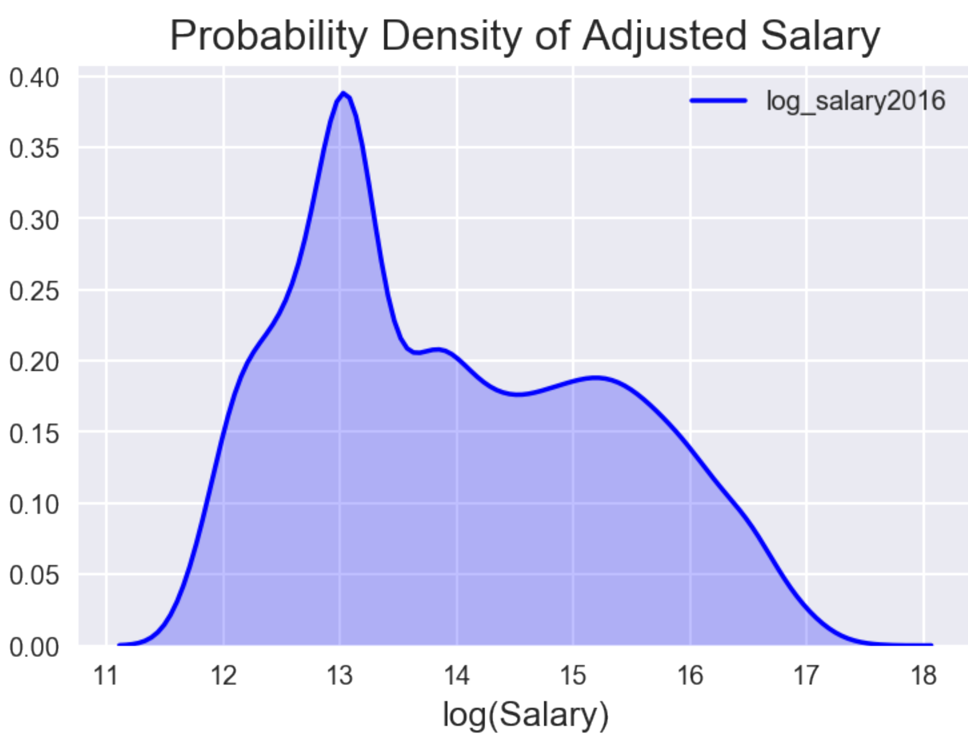

Adjusted Salary (1985 - 2016)
1e7
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Salary ($10 millions)
1985
1990
1995
2000
2005
2010
2015
2020
Year

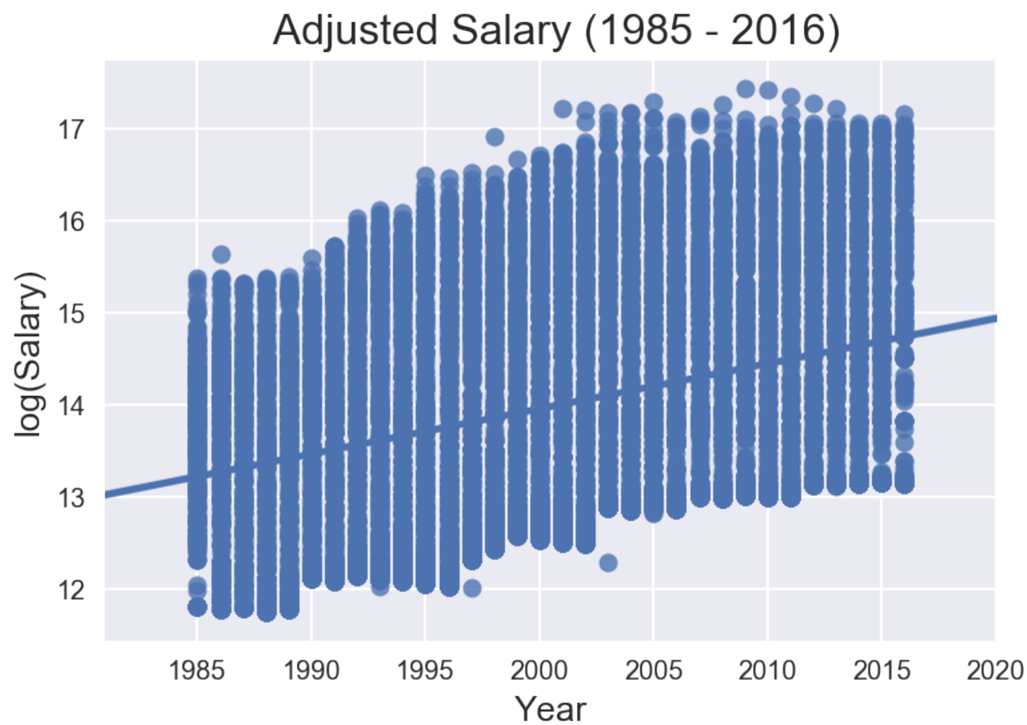

Boxplot of Adjusted Salary (1985-2016)
1e7
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Salary ($10 millions)
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
Year

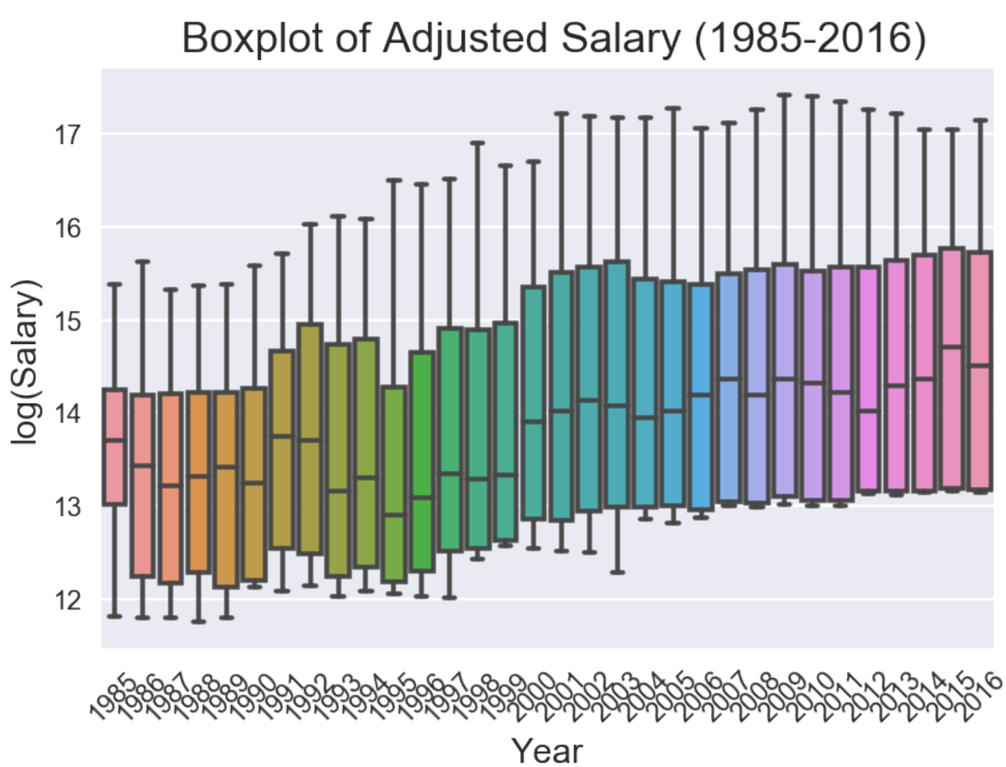
Considering the salary data is heavily skewed to the right, we will want to use the log of salary instead. Let's look at that ditribution and scatter plot.

```
In [19]: salary_adj['log_salary2016'] = np.log(salary_adj.salary2016)
```

```
In [20]: sns.kdeplot(salary_adj.log_salary2016, shade=True, color="b")
         plt.title("Probability Density of Adjusted Salary")
         plt.xlabel("log(Salary)")
         plt.show()

         sns.regplot(x='yearID',
                     y='log_salary2016',
                    data=salary_adj)
         plt.title(' Adjusted Salary (1985 - 2016)')
         plt.xlabel('Year')
         plt.ylabel('log(Salary)')
         plt.show()

         sns.boxplot(x="yearID", y="log_salary2016", data=salary_adj)
         plt.title(' Boxplot of Adjusted Salary (1985-2016)')
         plt.xlabel('Year')
         plt.ylabel('log(Salary)')
         plt.xticks(rotation=45)
         plt.show()
```




Adjusted Salary (1985 - 2016)
log(Salary)
Year
12
13
14
15
16
17
1985
1990
1995
2000
2005
2010
2015
2020


Boxplot of Adjusted Salary (1985-2016)
log(Salary)
Year
12
13
14
15
16
17
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016

Let's look at some plots between log(Salary) and what the MLB calls "standard stats". These standard stats are made up of batting average (AVG), home runs (HR), runs batted in (RBI), runs scored (R), and stolen bases (SB). First, let's create the batting average feature, which is simply a player's hits divided by his total at-bats for a number between zero (shown as .000) and one (shown as 1.000).

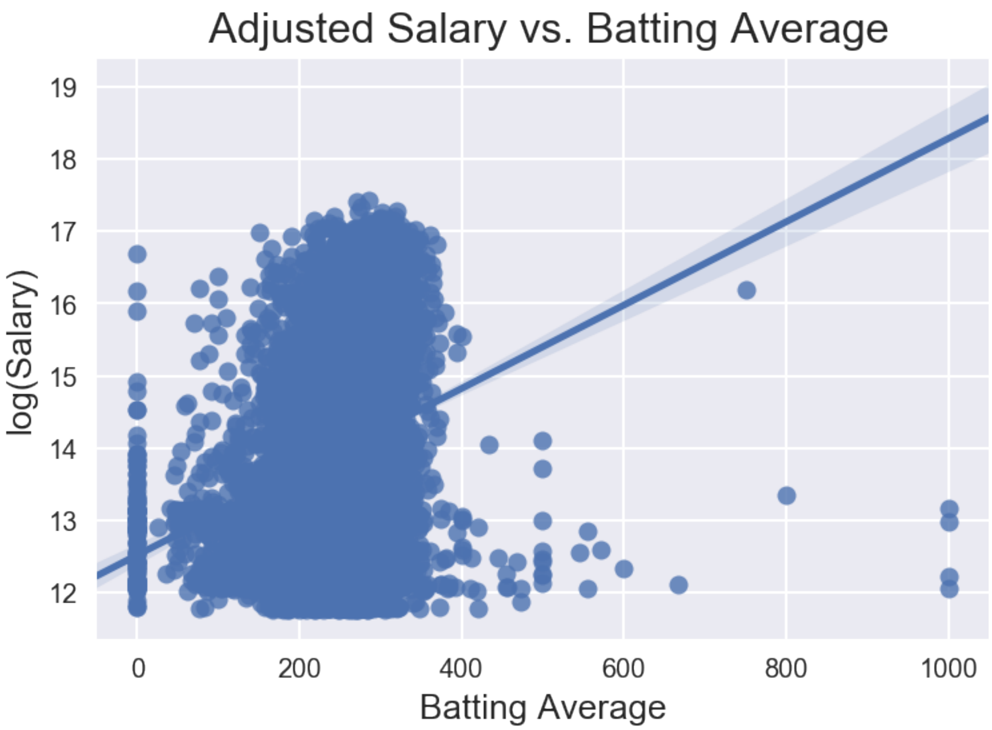
In [21]:
```
salary_adj['AVG'] = salary_adj.H / salary_adj.AB *1000
salary_adj.describe()
```

Out[21]:

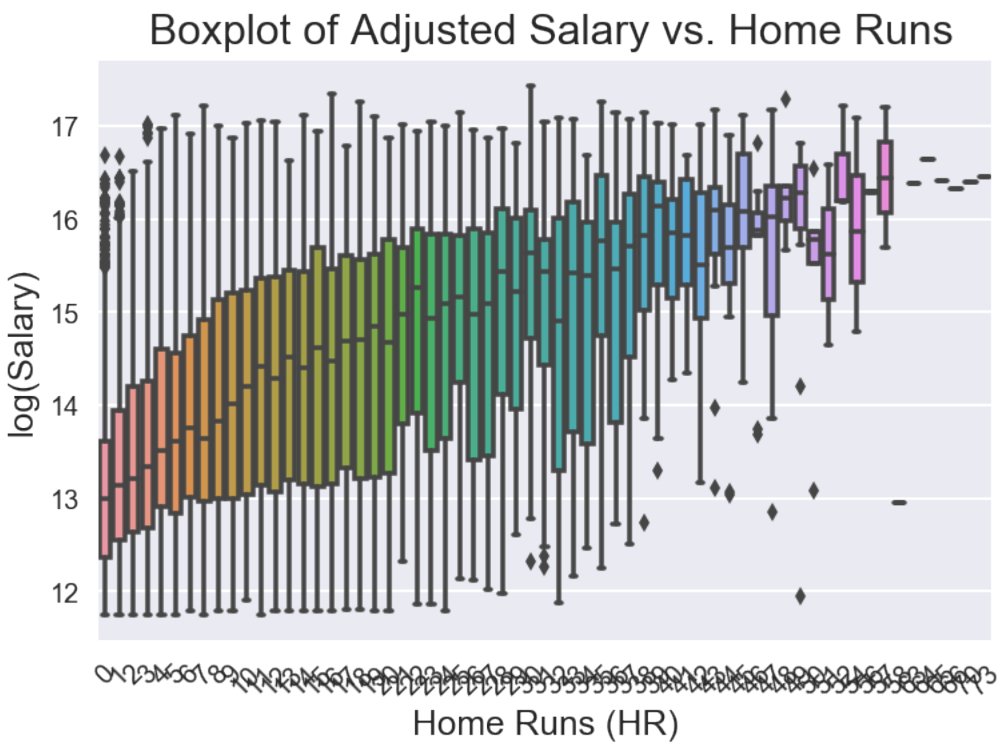
	yearID	stint	G	AB	R	
count	12395.000000	12395.000000	12395.000000	12395.000000	12395.000000	12395.00
mean	2000.337959	1.006374	96.494474	314.486244	43.352723	84.27
std	8.806370	0.085450	45.923603	189.707727	31.416071	56.02
min	1985.000000	1.000000	1.000000	1.000000	0.000000	0.00
25%	1993.000000	1.000000	59.000000	144.000000	16.000000	34.00
50%	2000.000000	1.000000	103.000000	310.000000	38.000000	79.00
75%	2008.000000	1.000000	138.000000	484.500000	67.000000	131.00
max	2016.000000	3.000000	163.000000	716.000000	152.000000	240.00

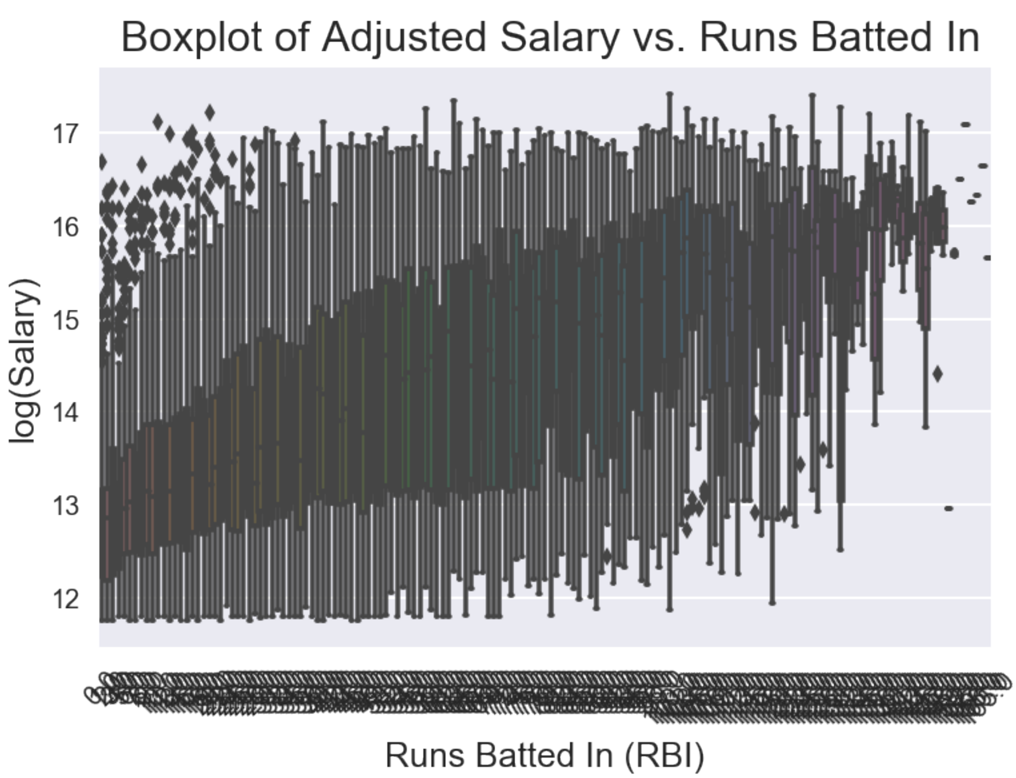
In [22]:
```
sns.regplot(x="AVG", y="log_salary2016", data=salary_adj)
plt.title(' Adjusted Salary vs. Batting Average')
plt.xlabel('Batting Average')
plt.ylabel('log(Salary)')
plt.show()

sns.boxplot(x="HR", y="log_salary2016", data=salary_adj)
plt.title(' Boxplot of Adjusted Salary vs. Home Runs')
plt.xlabel('Home Runs (HR)')
plt.ylabel('log(Salary)')
plt.xticks(rotation=45)
plt.tick_params(labelsize=10)
ax = plt.axes()
plt.show()

sns.boxplot(x="RBI", y="log_salary2016", data=salary_adj)
plt.title(' Boxplot of Adjusted Salary vs. Runs Batted In')
plt.xlabel('Runs Batted In (RBI)')
plt.ylabel('log(Salary)')
plt.xticks(rotation=45)
plt.show()

sns.boxplot(x="R", y="log_salary2016", data=salary_adj)
plt.title(' Boxplot of Adjusted Salary vs. Runs Scored')
plt.xlabel('Runs Scored (R)')
plt.ylabel('log(Salary)')
plt.xticks(rotation=45)
plt.show()
```

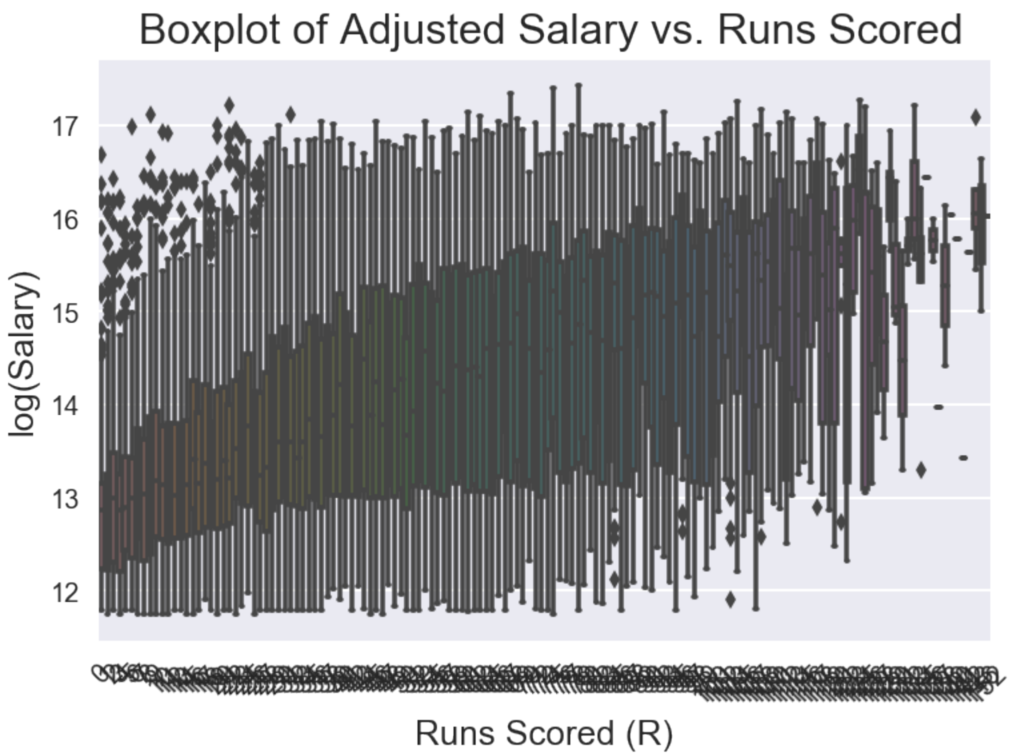
```
sns.boxplot(x="SB", y="log_salary2016", data=salary_adj)
plt.title(' Boxplot of Adjusted Salary vs. Stolen Bases')
plt.xlabel('Stolen Bases (SB)')
plt.ylabel('log(Salary)')
plt.xticks(rotation=45)
plt.show()

sns.boxplot(x="2B", y="log_salary2016", data=salary_adj)
plt.title(' Boxplot of Adjusted Salary vs. Doubles')
plt.xlabel('Doubles (DB)')
plt.ylabel('log(Salary)')
plt.xticks(rotation=45)
plt.show()
```

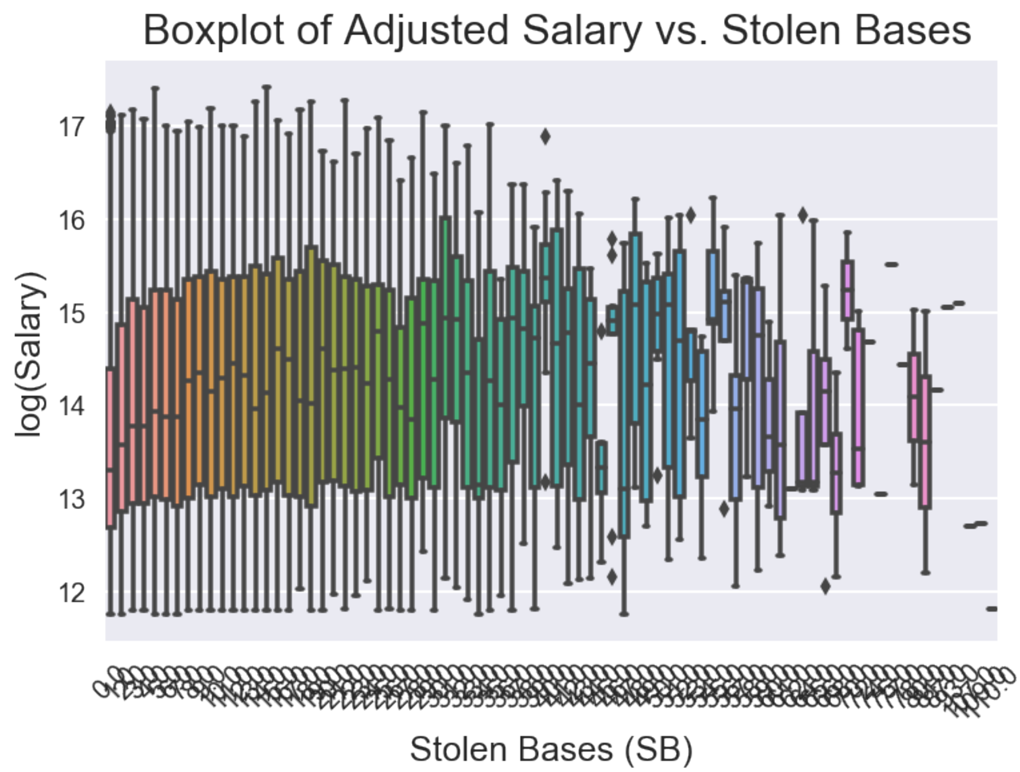


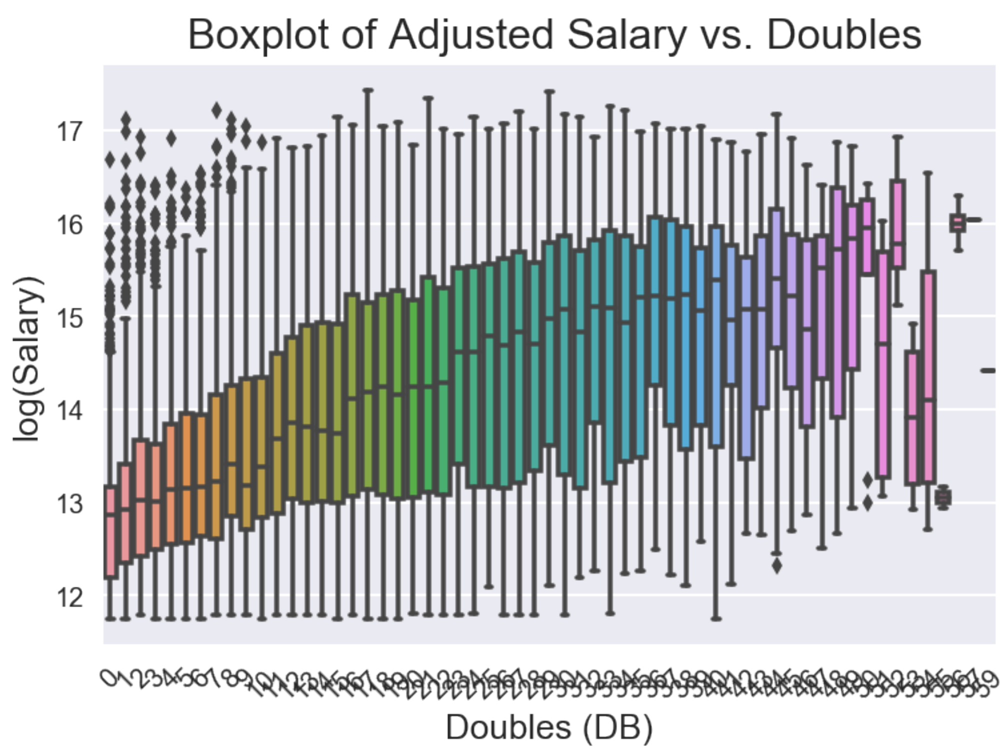

```
/Users/jeff/anaconda3/lib/python3.6/site-packages/matplotlib/cbook/deprecation.py:107: Matplotl

Adding an axes using the same arguments as a previous axes currently reuses the earlier instanc
```


Boxplot of Adjusted Salary vs. Home Runs
log(Salary)
17
16
15
14
13
12
Home Runs (HR)

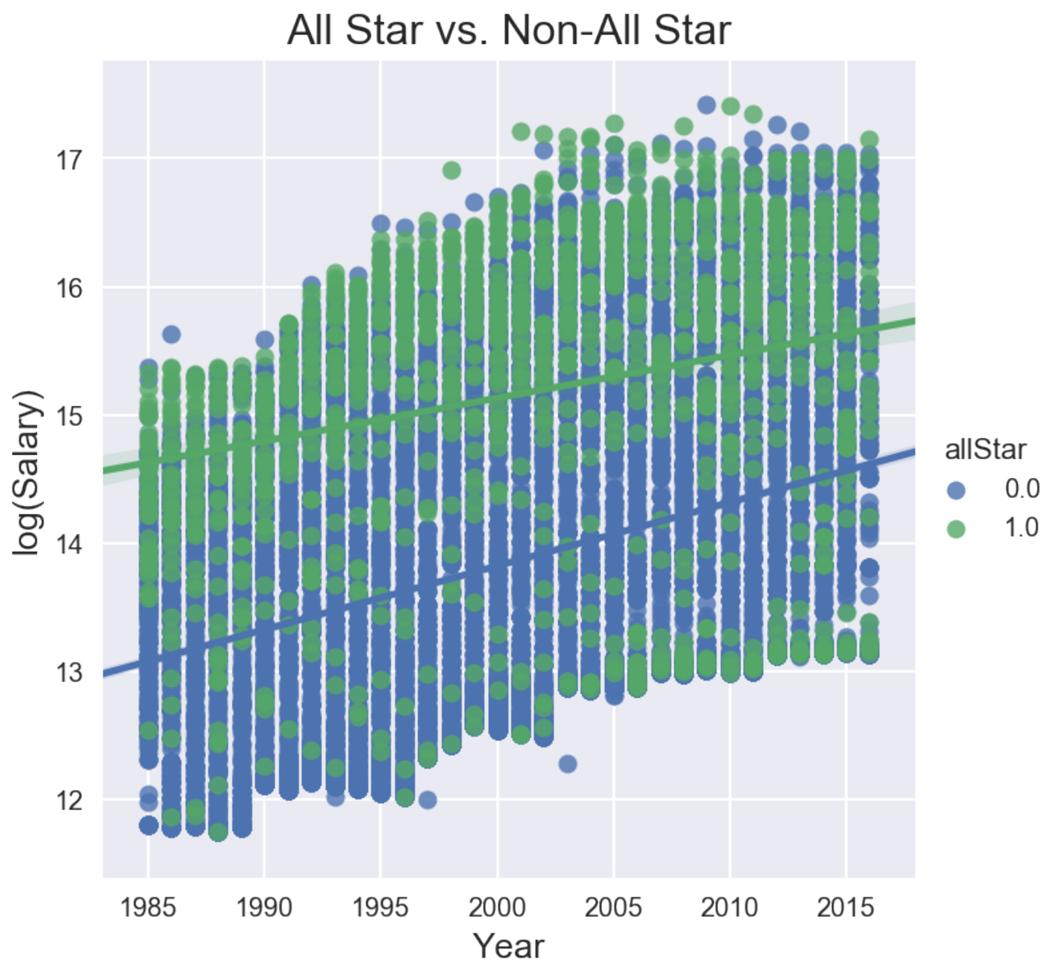

Boxplot of Adjusted Salary vs. Runs Batted In
log(Salary)
17
16
15
14
13
12
Runs Batted In (RBI)

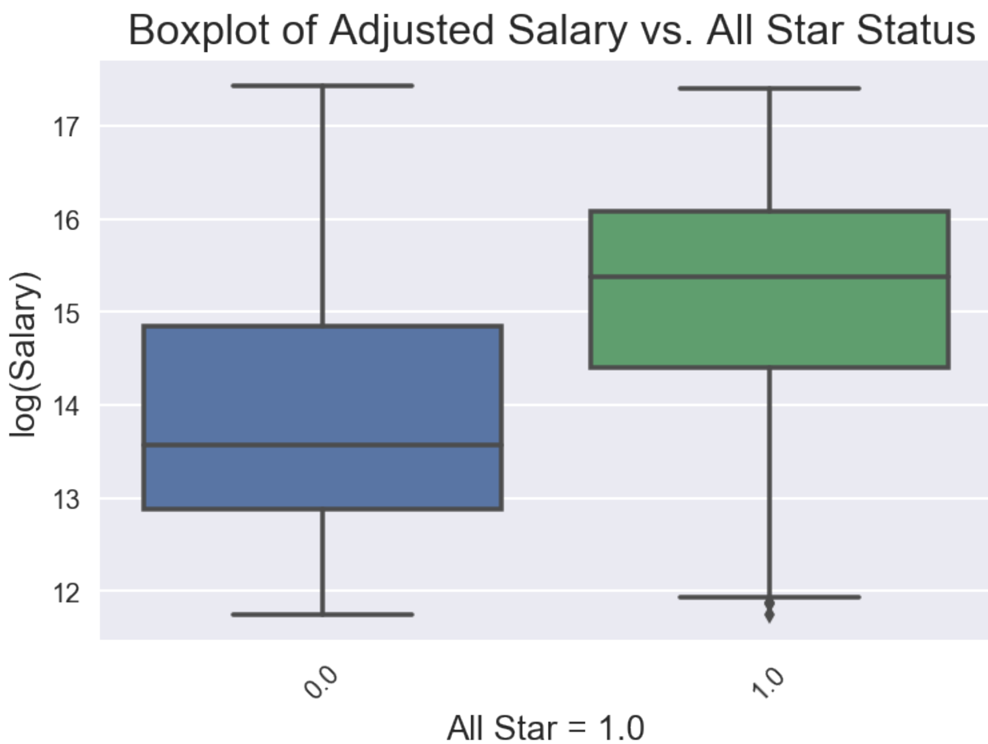

Boxplot of Adjusted Salary vs. Runs Scored
log(Salary)
17
16
15
14
13
12
Runs Scored (R)


Boxplot of Adjusted Salary vs. Stolen Bases
log(Salary)
17
16
15
14
13
12
Stolen Bases (SB)

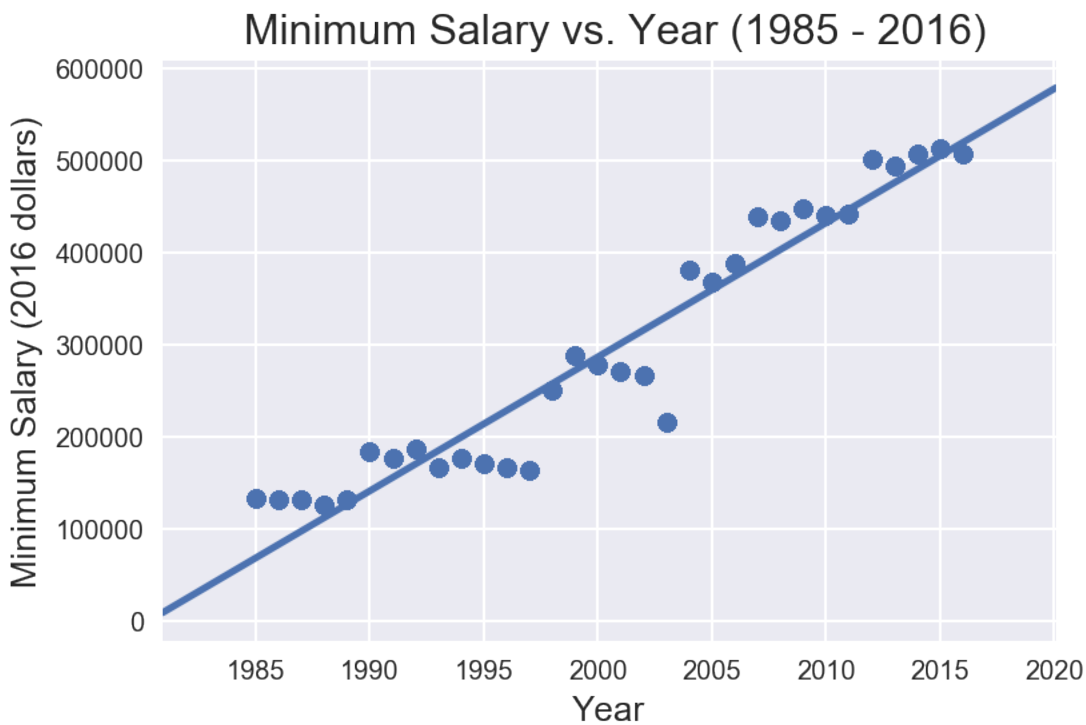

Boxplot of Adjusted Salary vs. Doubles
log(Salary)
17
16
15
14
13
12
Doubles (DB)

```
In [23]: cols = ['log_salary2016', 'G', 'AB', 'R', 'H', '2B', '3B', 'HR', 'RBI', 'SB', 'CS', 'I
                        'GIDP', 'AVG']
         corr = salary_adj[cols].corr()
         corr.style.background_gradient().set_precision(2)

Out[23]: <pandas.io.formats.style.Styler at 0x11736a2b0>

In [24]: sns.lmplot(x='yearID',
                    y='log_salary2016',
                    hue = 'allStar',
                    data=salary_adj)
         plt.title('All Star vs. Non-All Star')
         plt.ylabel('log(Salary)')
         plt.xlabel('Year')
         plt.show()

         sns.boxplot(x="allStar", y="log_salary2016", data=salary_adj)
         plt.title(' Boxplot of Adjusted Salary vs. All Star Status')
         plt.xlabel('All Star = 1.0')
         plt.ylabel('log(Salary)')
         plt.xticks(rotation=45)
         plt.show()
```


All Star vs. Non-All Star
log(Salary)
17
16
15
14
13
12
1985
1990
1995
2000
2005
2010
2015
Year
allStar
0.0
1.0



In [25]:
```
sns.regplot(x='yearID',
            y='min_salary2016',
           data=salary_adj)
plt.title(' Minimum Salary vs. Year (1985 - 2016)')
plt.xlabel('Year')
plt.ylabel('Minimum Salary (2016 dollars)')
plt.show()
```




```
In [26]: top_50_salary = salary_adj.nlargest(170, 'salary2016')
         top_50_paid = top_50_salary.playerID.unique()
         top_50_paid
```

```
Out[26]: array(['rodrial01', 'ramirma02', 'cabremi01', 'wellsve01', 'giambja01',
                'bondsba01', 'delgaca01', 'howarry01', 'jeterde01', 'pujolal01',
                'teixema01', 'mauerjo01', 'canoro01', 'fieldpr01', 'hamiljo03',
                'gonzaad01', 'ramirha01', 'vaughmo01', 'reyesjo01', 'uptonju01',
                'bagweje01', 'sheffga01', 'heltoto01', 'crawfca02', 'beltrca01',
                'kempma01', 'ellsbja01', 'werthja01', 'leeca01', 'ordonma01',
                'greensh01', 'soriaal01', 'sosasa01', 'piazzmi01', 'hunteto01',
                'wrighda03', 'tulowtr01', 'choosh01', 'vottojo01', 'jonesch06',
                'bayja01', 'braunry02', 'youngmi02', 'pencehu01', 'ramirar01',
                'ethiean01', 'beltrad01', 'martivi01', 'hollima01', 'sexsori01',
                'belleal01'], dtype=object)
```

```
In [27]: top_50_paid_players = salary_adj[salary_adj.playerID.isin(['rodrial01', 'ramirma02',
                'bondsba01', 'delgaca01', 'howarry01', 'jeterde01', 'pujolal01',
                'teixema01', 'mauerjo01', 'canoro01', 'fieldpr01', 'hamiljo03',
                'gonzaad01', 'ramirha01', 'vaughmo01', 'reyesjo01', 'uptonju01',
                'bagweje01', 'sheffga01', 'heltoto01', 'crawfca02', 'beltrca01',
                'kempma01', 'ellsbja01', 'werthja01', 'leeca01', 'ordonma01',
                'greensh01', 'soriaal01', 'sosasa01', 'piazzmi01', 'hunteto01',
                'wrighda03', 'tulowtr01', 'choosh01', 'vottojo01', 'jonesch06',
```

```
             'bayja01', 'braunry02', 'youngmi02', 'pencehu01', 'ramirar01',
             'ethiean01', 'beltrad01', 'martivi01', 'hollima01', 'sexsori01'])]
```

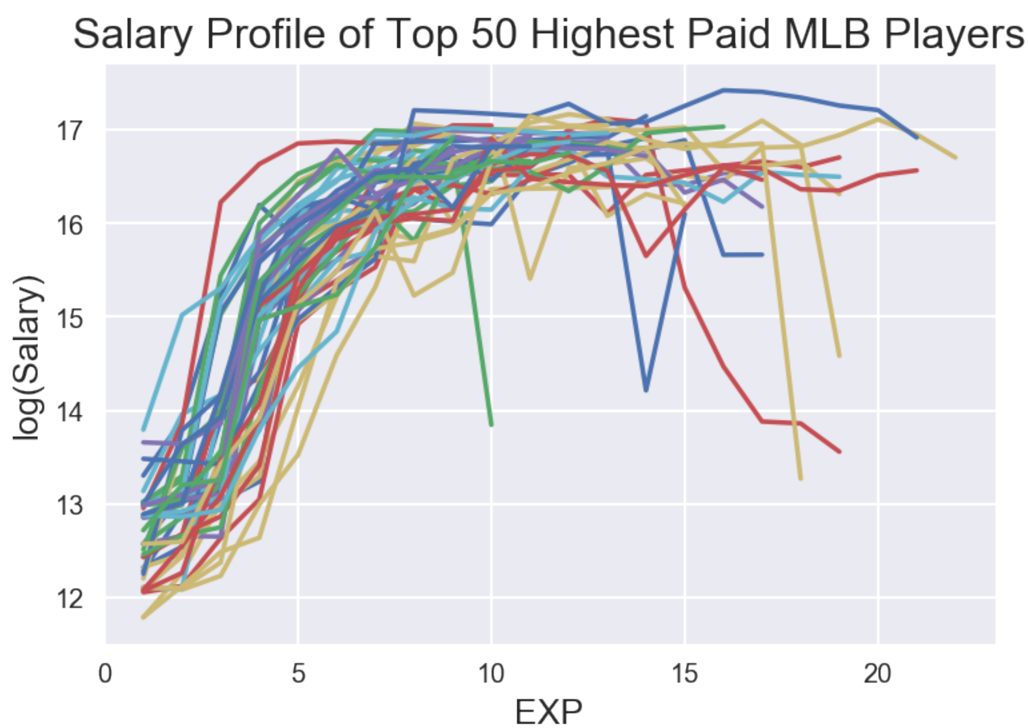
```
In [28]: top_100_salary = salary_adj.nlargest(388, 'salary2016')
         top_100_paid = top_100_salary.playerID.unique()
         top_100_paid
```

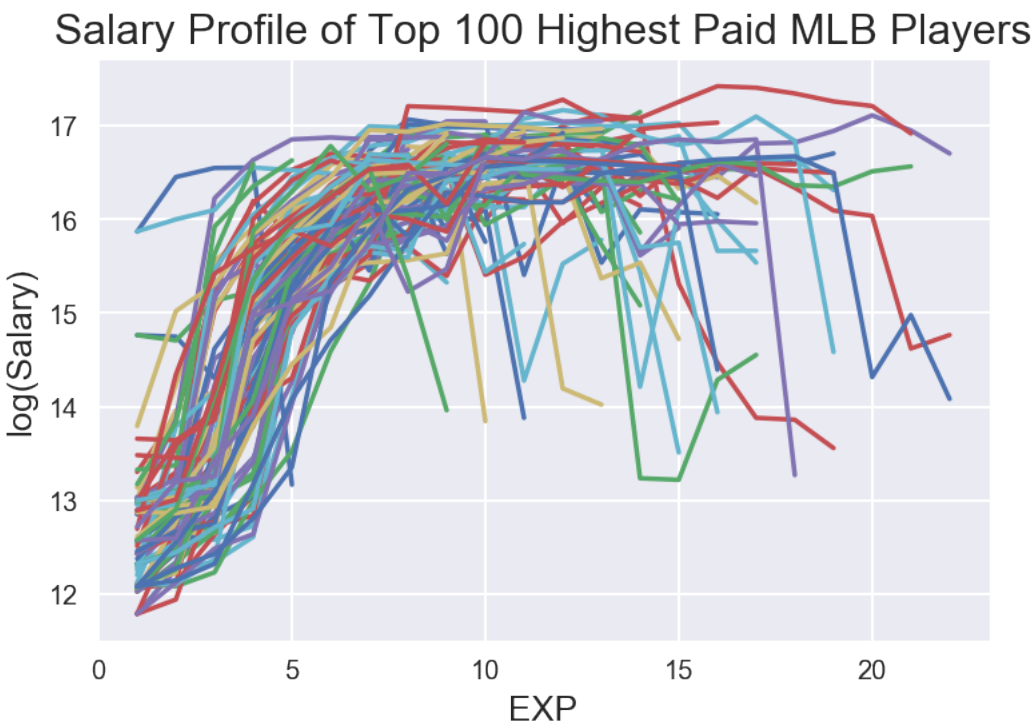
```
Out[28]: array(['rodrial01', 'ramirma02', 'cabremi01', 'wellsve01', 'giambja01',
             'bondsba01', 'delgaca01', 'howarry01', 'jeterde01', 'pujolal01',
             'teixema01', 'mauerjo01', 'canoro01', 'fieldpr01', 'hamiljo03',
             'gonzaad01', 'ramirha01', 'vaughmo01', 'reyesjo01', 'uptonju01',
             'bagweje01', 'sheffga01', 'heltoto01', 'crawfca02', 'beltrca01',
             'kempma01', 'ellsbja01', 'werthja01', 'leeca01', 'ordonma01',
             'greensh01', 'soriaal01', 'sosasa01', 'piazzmi01', 'hunteto01',
             'wrighda03', 'tulowtr01', 'chooshO1', 'vottojo01', 'jonesch06',
             'bayja01', 'braunry02', 'youngmi02', 'pencehu01', 'ramirar01',
             'ethiean01', 'beltrad01', 'martivi01', 'hollima01', 'sexsori01',
             'belleal01', 'abreubo01', 'sandopa01', 'furcara01', 'thomeji01',
             'gonzaca01', 'guerrvl01', 'berkmla01', 'mccanbr01', 'willibe02',
             'gonzaju03', 'mondera01', 'griffke02', 'walkela01', 'utleych01',
             'poseybu01', 'tejadmi01', 'morneju01', 'jonesan01', 'jonesad01',
             'kinslia01', 'napolmi01', 'drewjd01', 'ortizda01', 'grandcu01',
             'troutmi01', 'burrepa01', 'hidalri01', 'burnije01', 'markani01',
             'rasmuco01', 'wietema01', 'peraljh01', 'damonjo01', 'matsuhi01',
             'fukudko01', 'higgibo02', 'wilsopr01', 'leede02', 'andruel01',
             'dyeje01', 'molinya01', 'martiru01', 'rowanaa01', 'mcgwima01',
             'kendaja01', 'rolensc01', 'posadjo01', 'bautijo02', 'uptonbj01'],
            dtype=object)
```

```
In [29]: top_100_paid_players = salary_adj[salary_adj.playerID.isin(['rodrial01', 'ramirma02',
             'bondsba01', 'delgaca01', 'howarry01', 'jeterde01', 'pujolal01',
             'teixema01', 'mauerjo01', 'canoro01', 'fieldpr01', 'hamiljo03',
             'gonzaad01', 'ramirha01', 'vaughmo01', 'reyesjo01', 'uptonju01',
             'bagweje01', 'sheffga01', 'heltoto01', 'crawfca02', 'beltrca01',
             'kempma01', 'ellsbja01', 'werthja01', 'leeca01', 'ordonma01',
             'greensh01', 'soriaal01', 'sosasa01', 'piazzmi01', 'hunteto01',
             'wrighda03', 'tulowtr01', 'chooshO1', 'vottojo01', 'jonesch06',
             'bayja01', 'braunry02', 'youngmi02', 'pencehu01', 'ramirar01',
             'ethiean01', 'beltrad01', 'martivi01', 'hollima01', 'sexsori01',
             'belleal01', 'abreubo01', 'sandopa01', 'furcara01', 'thomeji01',
             'gonzaca01', 'guerrvl01', 'berkmla01', 'mccanbr01', 'willibe02',
             'gonzaju03', 'mondera01', 'griffke02', 'walkela01', 'utleych01',
             'poseybu01', 'tejadmi01', 'morneju01', 'jonesan01', 'jonesad01',
             'kinslia01', 'napolmi01', 'drewjd01', 'ortizda01', 'grandcu01',
             'troutmi01', 'burrepa01', 'hidalri01', 'burnije01', 'markani01',
             'rasmuco01', 'wietema01', 'peraljh01', 'damonjo01', 'matsuhi01',
             'fukudko01', 'higgibo02', 'wilsopr01', 'leede02', 'andruel01',
             'dyeje01', 'molinya01', 'martiru01', 'rowanaa01', 'mcgwima01',
```

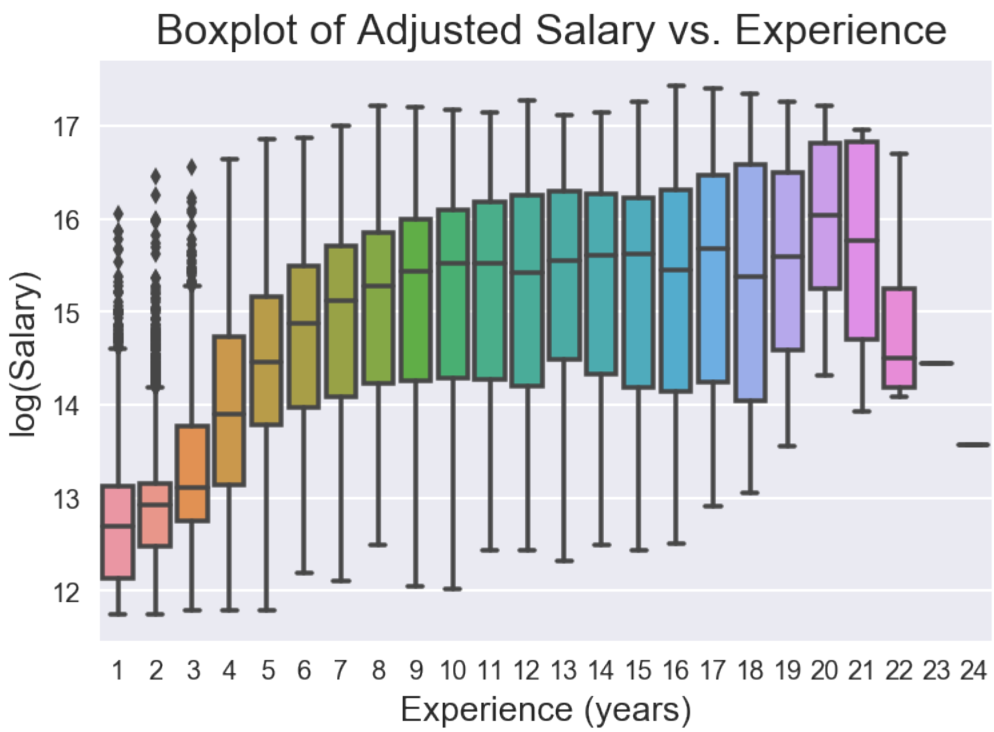
```
                'kendaja01', 'rolensc01', 'posadjo01', 'bautijo02', 'uptonbj01'
                                                                     ])]
```

```
In [30]: top_50_paid_players.set_index('EXP', inplace=True)
         top_50_paid_players.groupby('playerID')['log_salary2016'].plot(legend=False)
         plt.title("Salary Profile of Top 50 Highest Paid MLB Players")
         plt.ylabel("log(Salary)")
         plt.show()

         top_100_paid_players.set_index('EXP', inplace=True)
         top_100_paid_players.groupby('playerID')['log_salary2016'].plot(legend=False)
         plt.title("Salary Profile of Top 100 Highest Paid MLB Players")
         plt.ylabel("log(Salary)")
         plt.show()
```





In [31]:
```
sns.boxplot(x="EXP", y="log_salary2016", data=salary_adj)
plt.title(' Boxplot of Adjusted Salary vs. Experience')
plt.xlabel('Experience (years)')
plt.ylabel('log(Salary)')

#plt.xticks(rotation=90)
plt.show()
```



1.5 V. Feature Engineering

1.5.1 1. Create a quadratic term for experience (EXP-squared)

Seeing that salary seems to have a non-linear relationship with salary, let's add a quadratic term for experience to the feature set.

In [32]:
```
salary_adj['EXP_SQ']=np.square(salary_adj['EXP'])
#salary_adj.sort_values(by=['playerID', 'yearID'])
salary_adj.describe()
```

Out[32]:

	yearID	stint	G	AB	R	
count	12395.000000	12395.000000	12395.000000	12395.000000	12395.000000	12395.000
mean	2000.337959	1.006374	96.494474	314.486244	43.352723	84.27
std	8.806370	0.085450	45.923603	189.707727	31.416071	56.02
min	1985.000000	1.000000	1.000000	1.000000	0.000000	0.000
25%	1993.000000	1.000000	59.000000	144.000000	16.000000	34.000
50%	2000.000000	1.000000	103.000000	310.000000	38.000000	79.000
75%	2008.000000	1.000000	138.000000	484.500000	67.000000	131.000
max	2016.000000	3.000000	163.000000	716.000000	152.000000	240.000

1.5.2 2. Create lags of the target variable and feature set.

Let's create lagged values of the target variable and lagged values of the features. This is based on the idea that salary is based off past player performance, not current performance, since the salaries are set before a given season. Let's also use past salary as a feature as well as this is the best predictor of current salary we have.

```
In [33]: # lagged values of salary
         salary_adj['sal_t_1'] = salary_adj.groupby(['playerID'])['salary2016'].shift(1)
         salary_adj['sal_t_2'] = salary_adj.groupby(['playerID'])['salary2016'].shift(2)
         salary_adj['sal_t_3'] = salary_adj.groupby(['playerID'])['salary2016'].shift(3)

         # first difference of salary lagged one period
         salary_adj['sal_diff'] = salary_adj.salary2016 - salary_adj.sal_t_1
         salary_adj['sal_diff_t_1'] = salary_adj.groupby(['playerID'])['sal_diff'].shift(1)

         # lagged values of the features
         salary_adj['G_t_1'] = salary_adj.groupby(['playerID'])['G'].shift(1)
         salary_adj['G_t_2'] = salary_adj.groupby(['playerID'])['G'].shift(2)

         salary_adj['AB_t_1'] = salary_adj.groupby(['playerID'])['AB'].shift(1)
         salary_adj['AB_t_2'] = salary_adj.groupby(['playerID'])['AB'].shift(2)

         salary_adj['R_t_1'] = salary_adj.groupby(['playerID'])['R'].shift(1)
         salary_adj['R_t_2'] = salary_adj.groupby(['playerID'])['R'].shift(2)

         salary_adj['H_t_1'] = salary_adj.groupby(['playerID'])['H'].shift(1)
         salary_adj['H_t_2'] = salary_adj.groupby(['playerID'])['H'].shift(2)

         salary_adj['2B_t_1'] = salary_adj.groupby(['playerID'])['2B'].shift(1)
         salary_adj['2B_t_2'] = salary_adj.groupby(['playerID'])['2B'].shift(2)

         salary_adj['3B_t_1'] = salary_adj.groupby(['playerID'])['3B'].shift(1)
         salary_adj['3B_t_2'] = salary_adj.groupby(['playerID'])['3B'].shift(2)

         salary_adj['HR_t_1'] = salary_adj.groupby(['playerID'])['HR'].shift(1)
         salary_adj['HR_t_2'] = salary_adj.groupby(['playerID'])['HR'].shift(2)

         salary_adj['RBI_t_1'] = salary_adj.groupby(['playerID'])['RBI'].shift(1)
         salary_adj['RBI_t_2'] = salary_adj.groupby(['playerID'])['RBI'].shift(2)

         salary_adj['AVG_t_1'] = salary_adj.groupby(['playerID'])['AVG'].shift(1)
         salary_adj['AVG_t_2'] = salary_adj.groupby(['playerID'])['AVG'].shift(2)

         salary_adj['SB_t_1'] = salary_adj.groupby(['playerID'])['SB'].shift(1)
         salary_adj['SB_t_2'] = salary_adj.groupby(['playerID'])['SB'].shift(2)

         salary_adj['CS_t_1'] = salary_adj.groupby(['playerID'])['CS'].shift(1)
         salary_adj['CS_t_2'] = salary_adj.groupby(['playerID'])['CS'].shift(2)

         salary_adj['BB_t_1'] = salary_adj.groupby(['playerID'])['BB'].shift(1)
         salary_adj['BB_t_2'] = salary_adj.groupby(['playerID'])['BB'].shift(2)
```

```
salary_adj['SO_t_1'] = salary_adj.groupby(['playerID'])['SO'].shift(1)
salary_adj['SO_t_2'] = salary_adj.groupby(['playerID'])['SO'].shift(2)

salary_adj['IBB_t_1'] = salary_adj.groupby(['playerID'])['IBB'].shift(1)
salary_adj['IBB_t_2'] = salary_adj.groupby(['playerID'])['IBB'].shift(2)

salary_adj['HBP_t_1'] = salary_adj.groupby(['playerID'])['HBP'].shift(1)
salary_adj['HBP_t_2'] = salary_adj.groupby(['playerID'])['HBP'].shift(2)

salary_adj['SH_t_1'] = salary_adj.groupby(['playerID'])['SH'].shift(1)
salary_adj['SH_t_2'] = salary_adj.groupby(['playerID'])['SH'].shift(2)

salary_adj['SF_t_1'] = salary_adj.groupby(['playerID'])['SF'].shift(1)
salary_adj['SF_t_2'] = salary_adj.groupby(['playerID'])['SF'].shift(2)

salary_adj['GIDP_t_1'] = salary_adj.groupby(['playerID'])['GIDP'].shift(1)
salary_adj['GIDP_t_2'] = salary_adj.groupby(['playerID'])['GIDP'].shift(2)

salary_adj['allStar_t_1'] = salary_adj.groupby(['playerID'])['allStar'].shift(1)
salary_adj['allStar_t_2'] = salary_adj.groupby(['playerID'])['allStar'].shift(2)
#salary_adj.sort_values(by=['playerID', 'yearID'])
```

1.5.3 3. Calculate on base percentage (OBP).

On Base Percentage (aka OBP, On Base Average, OBA) is a measure of how often a batter reaches base. It is approximately equal to Times on Base/Plate appearances.

The full formula is OBP = (Hits + Walks + Hit by Pitch) / (At Bats + Walks + Hit by Pitch + Sacrifice Flies). Batters are not credited with reaching base on an error or fielder's choice, and they are not charged with an opportunity if they make a sacrifice bunt.

```
In [34]: salary_adj['OBP'] = 1000*(salary_adj.H + salary_adj.BB + salary_adj.HBP)/(salary_adj.
                                                                              + salary_adj.SF)

         # Create lagged value of OBP
         salary_adj['OBP_t_1'] = salary_adj.groupby(['playerID'])['OBP'].shift(1)
         salary_adj['OBP_t_2'] = salary_adj.groupby(['playerID'])['OBP'].shift(2)
```

1.5.4 4. Create interactions between certain features.

Let's experiment by interacting some of the features. For example to pick up the effect of a player getting better over time, or at least staying consistent at a high level, we could interact experience (EXP) with on base percentage (OBP). Another example would be to pick up the effect of a player that both hits a lot of home runs (HR) and gets on base a lot (OBP). We could try other interactions, but let's just stick to these two for now.

```
In [35]: salary_adj['EXP_OBP'] = salary_adj.EXP*salary_adj.OBP
         salary_adj['OBP_HR'] = salary_adj.OBP*salary_adj.HR
```

```python
# Create lag value of interactions above
salary_adj['EXP_OBP_t_1'] = salary_adj.groupby(['playerID'])['EXP_OBP'].shift(1)
salary_adj['EXP_OBP_t_2'] = salary_adj.groupby(['playerID'])['EXP_OBP'].shift(2)
salary_adj['OBP_HR_t_1'] = salary_adj.groupby(['playerID'])['OBP_HR'].shift(1)
salary_adj['OBP_HR_t_2'] = salary_adj.groupby(['playerID'])['OBP_HR'].shift(2)

salary_adj['constant'] = 1
```

In [36]:
```python
salary_adj.describe()
```

Out[36]:

	yearID	stint	G	AB	R	
count	12395.000000	12395.000000	12395.000000	12395.000000	12395.000000	12395.000
mean	2000.337959	1.006374	96.494474	314.486244	43.352723	84.273
std	8.806370	0.085450	45.923603	189.707727	31.416071	56.022
min	1985.000000	1.000000	1.000000	1.000000	0.000000	0.000
25%	1993.000000	1.000000	59.000000	144.000000	16.000000	34.000
50%	2000.000000	1.000000	103.000000	310.000000	38.000000	79.000
75%	2008.000000	1.000000	138.000000	484.500000	67.000000	131.000
max	2016.000000	3.000000	163.000000	716.000000	152.000000	240.000

	RBI_t_1	RBI_t_2	AVG_t_1	AVG_t_2	SB_t_1	SB_t_
count	10156.000000	8394.000000	10156.000000	8394.000000	10156.000000	8394.000000
mean	46.300709	49.554682	259.917214	263.467637	7.261520	7.978675
std	31.582271	31.962401	46.845609	44.749212	10.586231	11.149041
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	21.000000	24.000000	239.405459	243.123763	1.000000	1.000000
50%	42.000000	46.000000	263.433985	266.666667	3.000000	4.000000
75%	67.000000	71.000000	286.446110	288.743608	9.000000	11.000000
max	165.000000	165.000000	1000.000000	1000.000000	110.000000	110.000000

1.6 VI. Modeling and Results

In [64]:
```python
y = salary_adj.log_salary2016
x_baseline = salary_adj[['G_t_1', 'AB_t_1', 'R_t_1',
                'H_t_1', '2B_t_1', '3B_t_1', 'HR_t_1',
                'RBI_t_1', 'SB_t_1', 'CS_t_1', 'BB_t_1', 'SO_t_1',
                'IBB_t_1', 'HBP_t_1', 'SH_t_1', 'SF_t_1',
                'GIDP_t_1', 'constant']]

x_lag1 = salary_adj[['sal_t_1', 'G_t_1', 'AB_t_1', 'R_t_1',
                'H_t_1', '2B_t_1', '3B_t_1', 'HR_t_1',
                'RBI_t_1', 'SB_t_1', 'CS_t_1', 'BB_t_1', 'SO_t_1',
                'IBB_t_1', 'HBP_t_1', 'SH_t_1', 'SF_t_1',
                'GIDP_t_1', 'AVG_t_1',
                'OBP_t_1', 'EXP', 'EXP_SQ', 'allStar_t_1', 'EXP_OBP_t_1', 'OBP_HR_t_1

x = salary_adj[['sal_t_1', 'sal_t_2', 'AB_t_1', 'AB_t_2', 'R_t_1', 'R_t_2',
                'H_t_1', 'H_t_2', '2B_t_1','2B_t_2','SO_t_1', 'SO_t_2',
```

```
                         'AVG_t_1', 'AVG_t_2',
                         'OBP_t_1', 'OBP_t_2', 'EXP', 'EXP_OBP_t_1', 'EXP_OBP_t_2', 'OBP_HR_t_
```

Create the training and test splits.

```
In [65]: X_base_train, X_base_test, y_base_train, y_base_test = train_test_split(x_baseline, y
         X_train_lag1, X_test_lag1, y_train_lag1, y_test_lag1 = train_test_split(x_lag1, y, tes
         X_train, X_test, y_train, y_test = train_test_split(x,y,test_size=.25, random_state=3
```

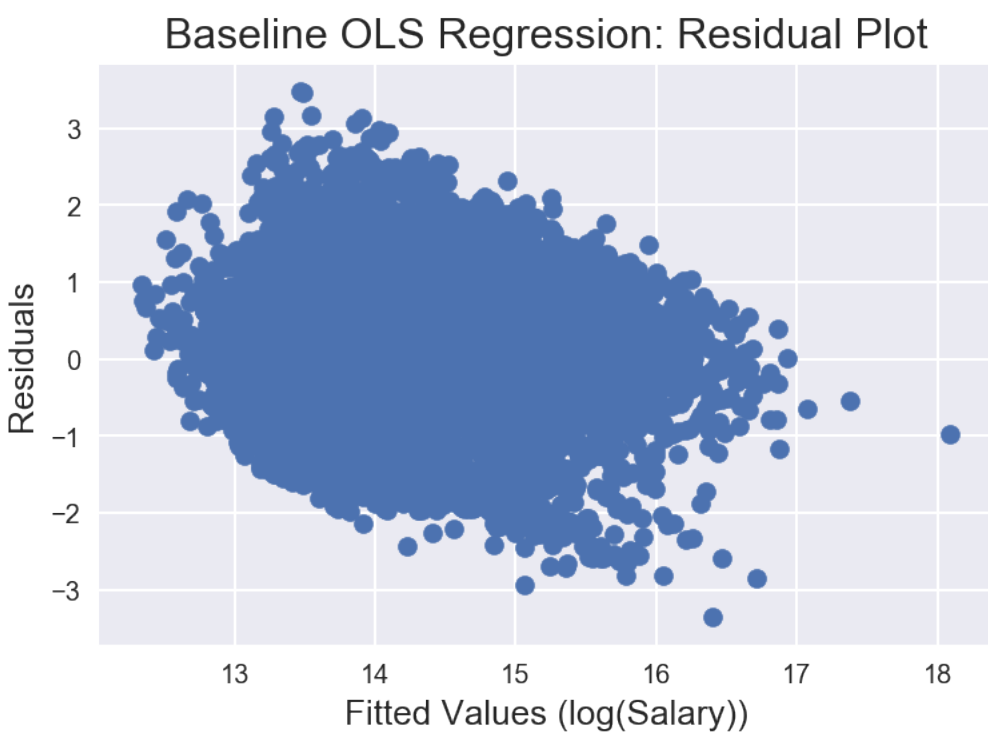
1.6.1 1. Linear Regression Models

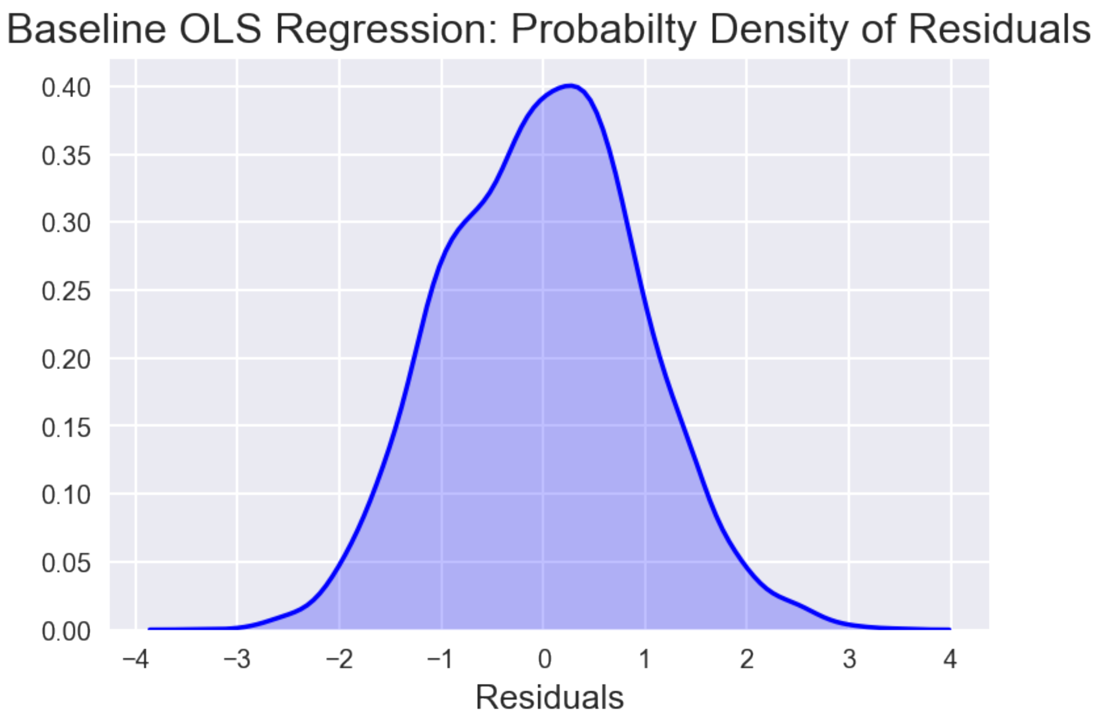
```
In [66]: ols_base = sm.OLS(y_base_train, X_base_train, missing='drop')
         results_ols_base = ols_base.fit()
         print(results_ols_base.summary())
```

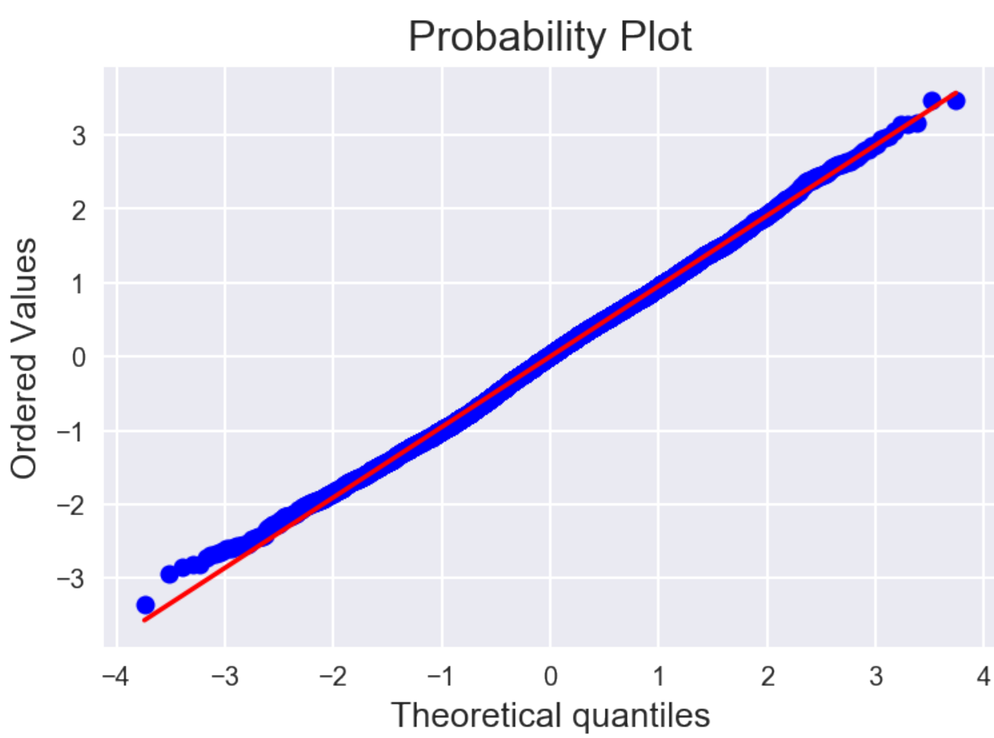
```
                            OLS Regression Results
==============================================================================
Dep. Variable:        log_salary2016   R-squared:                       0.451
Model:                           OLS   Adj. R-squared:                  0.449
Method:                Least Squares   F-statistic:                     367.6
Date:               Sun, 15 Jul 2018   Prob (F-statistic):               0.00
Time:                       20:53:43   Log-Likelihood:                -10481.
No. Observations:               7638   AIC:                         2.100e+04
Df Residuals:                   7620   BIC:                         2.112e+04
Df Model:                         17
Covariance Type:           nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
G_t_1         -0.0126      0.001    -14.611      0.000      -0.014      -0.011
AB_t_1         0.0044      0.000      9.790      0.000       0.004       0.005
R_t_1          0.0014      0.002      0.801      0.423      -0.002       0.005
H_t_1          0.0035      0.001      2.383      0.017       0.001       0.006
2B_t_1        -0.0005      0.002     -0.198      0.843      -0.005       0.004
3B_t_1        -0.0427      0.006     -6.689      0.000      -0.055      -0.030
HR_t_1         0.0113      0.004      3.183      0.001       0.004       0.018
RBI_t_1        0.0003      0.002      0.211      0.833      -0.003       0.003
SB_t_1         0.0081      0.002      4.283      0.000       0.004       0.012
CS_t_1        -0.0401      0.005     -7.662      0.000      -0.050      -0.030
BB_t_1         0.0109      0.001     11.526      0.000       0.009       0.013
SO_t_1        -0.0021      0.001     -3.330      0.001      -0.003      -0.001
IBB_t_1        0.0109      0.003      3.206      0.001       0.004       0.018
HBP_t_1        0.0167      0.004      4.709      0.000       0.010       0.024
SH_t_1        -0.0309      0.004     -6.932      0.000      -0.040      -0.022
SF_t_1         0.0029      0.006      0.461      0.645      -0.010       0.015
GIDP_t_1       0.0160      0.003      4.911      0.000       0.010       0.022
constant      13.2443      0.032    416.150      0.000      13.182      13.307
==============================================================================
Omnibus:                      14.287   Durbin-Watson:                   2.006
```

```
Prob(Omnibus):                  0.001   Jarque-Bera (JB):               12.921
Skew:                           0.059   Prob(JB):                      0.00156
Kurtosis:                       2.836   Cond. No.                     1.27e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.27e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [67]:
```python
residuals_ols_base = results_ols_base.resid
plt.scatter(results_ols_base.fittedvalues, results_ols_base.resid)
plt.title("Baseline OLS Regression: Residual Plot")
plt.xlabel("Fitted Values (log(Salary))")
plt.ylabel("Residuals")
plt.show()

sns.kdeplot(results_ols_base.resid, shade=True, color="b")
plt.title("Baseline OLS Regression: Probabilty Density of Residuals")
plt.xlabel("Residuals")
plt.show()

import pylab
import scipy.stats as scipystats
scipystats.probplot(results_ols_base.resid, dist="norm", plot=pylab)
pylab.show()
```




Baseline OLS Regression: Probabilty Density of Residuals
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−4
−3
−2
−1
0
1
2
3
4
Residuals

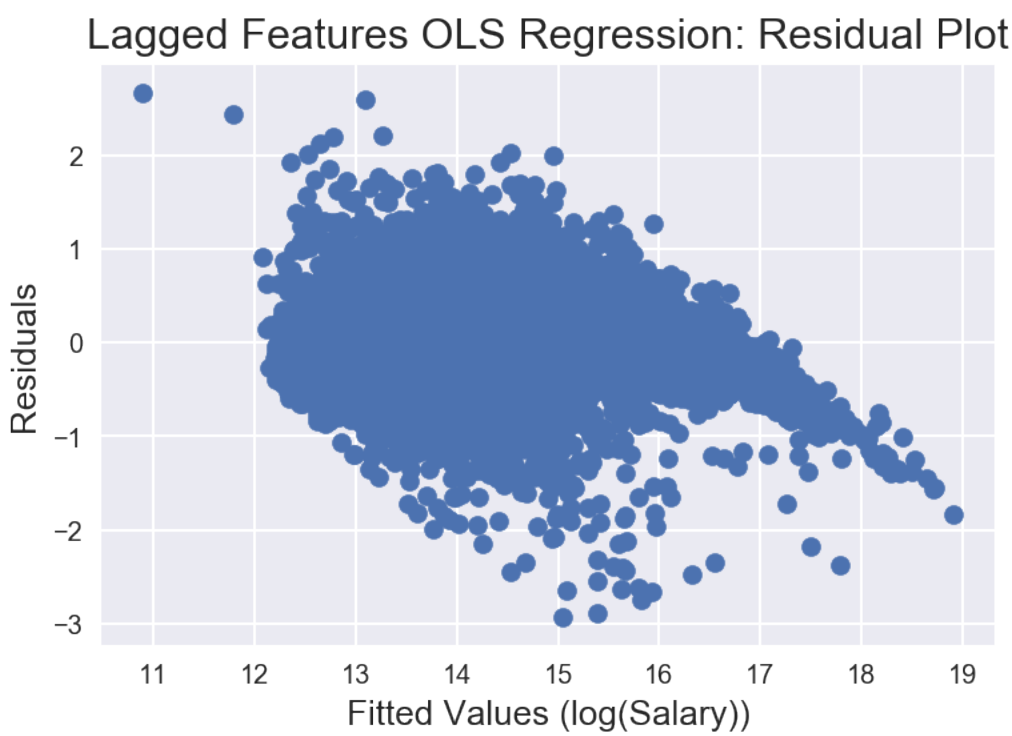

Probability Plot
Ordered Values
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
4
Theoretical quantiles

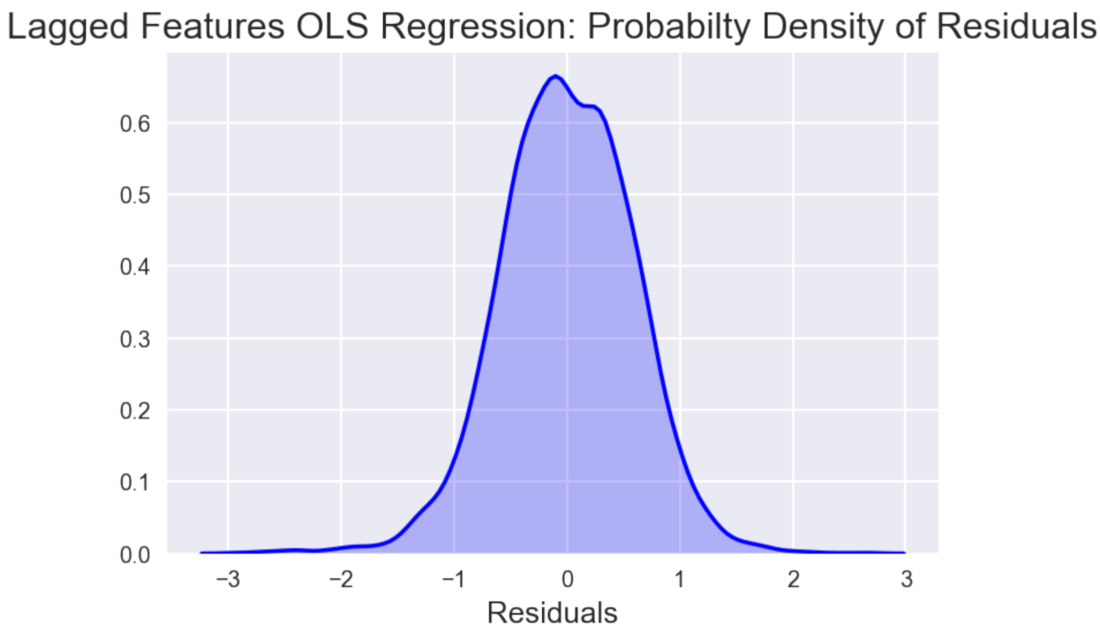
```
In [68]: ols_lag1 = sm.OLS(y_train_lag1, X_train_lag1, missing='drop')
         results_ols_lag1 = ols_lag1.fit()
         print(results_ols_lag1.summary())
```

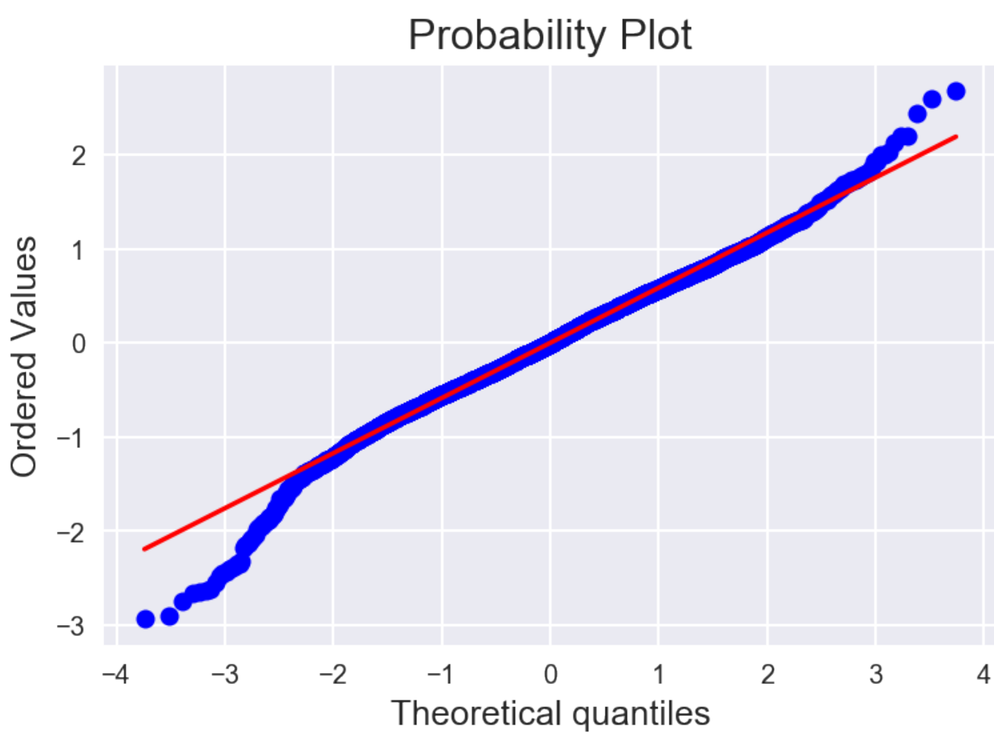
```
                            OLS Regression Results
==============================================================================
Dep. Variable:         log_salary2016   R-squared:                       0.792
Model:                            OLS   Adj. R-squared:                  0.791
Method:                 Least Squares   F-statistic:                     1113.
Date:                Sun, 15 Jul 2018   Prob (F-statistic):               0.00
Time:                        20:53:43   Log-Likelihood:                -6776.0
No. Observations:                7638   AIC:                         1.361e+04
Df Residuals:                    7611   BIC:                         1.379e+04
Df Model:                          26
Covariance Type:            nonrobust
=================================================================================
                     coef    std err          t      P>|t|      [0.025      0.975]
---------------------------------------------------------------------------------
sal_t_1          1.05e-07   2.37e-09     44.249      0.000       1e-07     1.1e-07
G_t_1             -0.0058      0.001    -10.515      0.000      -0.007      -0.005
AB_t_1             0.0022      0.000      5.219      0.000       0.001       0.003
R_t_1             -0.0015      0.001     -1.374      0.170      -0.004       0.001
H_t_1              0.0032      0.001      2.489      0.013       0.001       0.006
2B_t_1            -0.0021      0.002     -1.383      0.167      -0.005       0.001
3B_t_1            -0.0020      0.004     -0.518      0.605      -0.010       0.006
HR_t_1             0.0439      0.008      5.244      0.000       0.028       0.060
RBI_t_1            0.0019      0.001      1.910      0.056   -4.95e-05       0.004
SB_t_1             0.0025      0.001      2.116      0.034       0.000       0.005
CS_t_1            -0.0012      0.003     -0.376      0.707      -0.008       0.005
BB_t_1             0.0063      0.001      6.727      0.000       0.004       0.008
SO_t_1            -0.0018      0.000     -4.498      0.000      -0.003      -0.001
IBB_t_1            0.0085      0.002      3.740      0.000       0.004       0.013
HBP_t_1           -0.0001      0.002     -0.060      0.952      -0.005       0.004
SH_t_1            -0.0050      0.003     -1.778      0.075      -0.011       0.001
SF_t_1            -0.0038      0.004     -0.952      0.341      -0.012       0.004
GIDP_t_1           0.0020      0.002      1.012      0.312      -0.002       0.006
AVG_t_1        -4.023e-06      0.000     -0.010      0.992      -0.001       0.001
OBP_t_1            0.0005      0.000      1.350      0.177      -0.000       0.001
EXP                0.3668      0.013     28.829      0.000       0.342       0.392
EXP_SQ            -0.0187      0.000    -45.660      0.000      -0.019      -0.018
allStar_t_1        0.1101      0.026      4.262      0.000       0.059       0.161
EXP_OBP_t_1     5.635e-05   3.68e-05      1.530      0.126   -1.59e-05       0.000
OBP_HR_t_1        -0.0001   2.26e-05     -4.518      0.000      -0.000   -5.79e-05
min_salary2016  1.347e-06   5.82e-08     23.135      0.000    1.23e-06    1.46e-06
constant          11.3207      0.078    145.073      0.000      11.168      11.474
```

```
==============================================================================
Omnibus:                      181.978   Durbin-Watson:                   2.019
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              318.479
Skew:                          -0.200   Prob(JB):                     6.97e-70
Kurtosis:                       3.917   Cond. No.                     6.15e+07
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.15e+07. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [69]:
```
residuals_ols_lag1 = results_ols_lag1.resid
plt.scatter(results_ols_lag1.fittedvalues, results_ols_lag1.resid)
plt.title("Lagged Features OLS Regression: Residual Plot")
plt.xlabel("Fitted Values (log(Salary))")
plt.ylabel("Residuals")
plt.show()

sns.kdeplot(results_ols_lag1.resid, shade=True, color="b")
plt.title("Lagged Features OLS Regression: Probabilty Density of Residuals")
plt.xlabel("Residuals")
plt.show()

scipystats.probplot(results_ols_lag1.resid, dist="norm", plot=pylab)
pylab.show()
```




Lagged Features OLS Regression: Probabilty Density of Residuals
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−3
−2
−1
0
1
2
3
Residuals

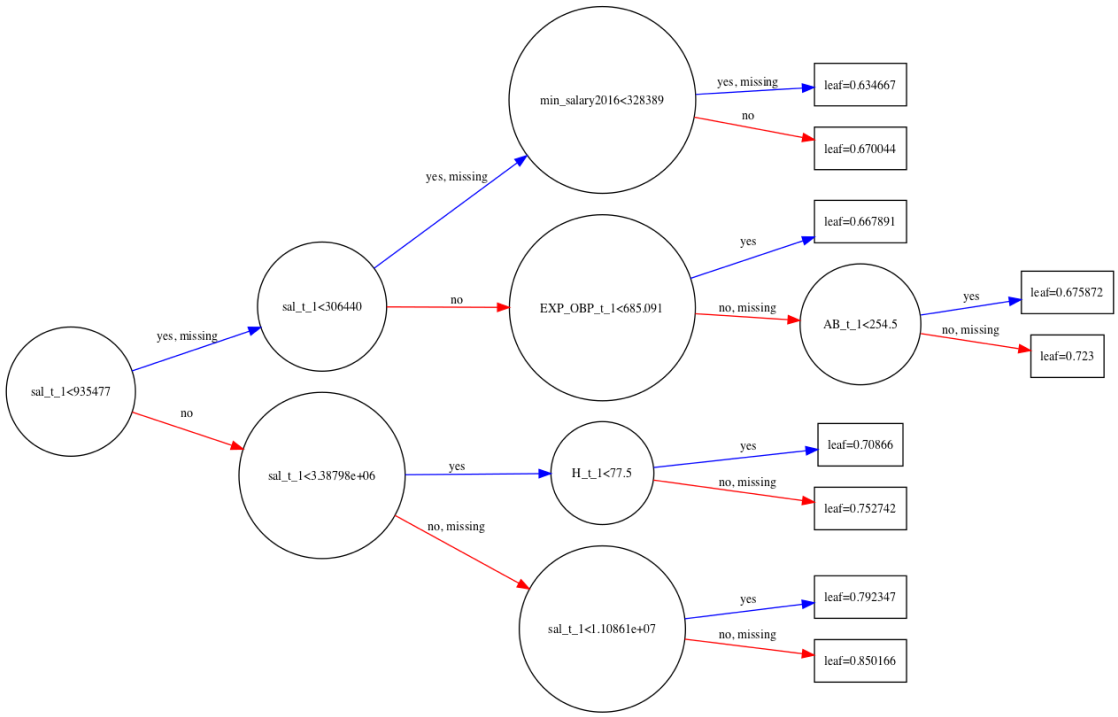

Probability Plot
Ordered Values
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
4
Theoretical quantiles

1.6.2 2. XGBoost Model

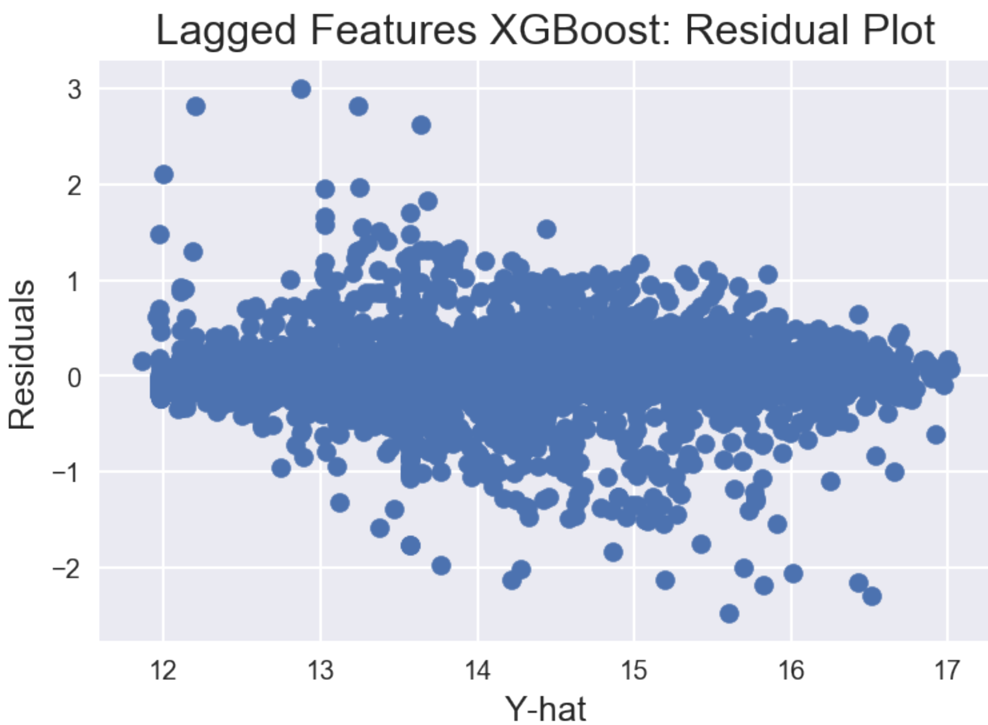
```
In [70]: import xgboost as xgb
         from xgboost import XGBRegressor
         from sklearn.model_selection import train_test_split
         from sklearn.model_selection import KFold
         from sklearn.model_selection import cross_val_score
         kfold = KFold(n_splits=5, random_state=314)

         model_xgb_lag1 = XGBRegressor(objective='reg:linear',
                                  n_estimators=400,
                                  max_depth=6,
                                  learning_rate = 0.08,
                                  colsample_bytree=1,
                                  subsample = .8,
                                  gamma = 1,
                                  min_child_weight=5,
                                  nthreads=4,
                                  seed=314,
                                  eval_metric="rmse")



         results_lag1 = cross_val_score(model_xgb_lag1, X_train_lag1, y_train_lag1, cv=kfold)

         print(results_lag1)

         model_xgb_lag1.fit(X_train_lag1, y_train_lag1)


         # make predictions for test data
         y_pred_lag1 = model_xgb_lag1.predict(X_test_lag1)

         print("Score_XGB:", model_xgb_lag1.score(X_test_lag1, y_test_lag1))
```

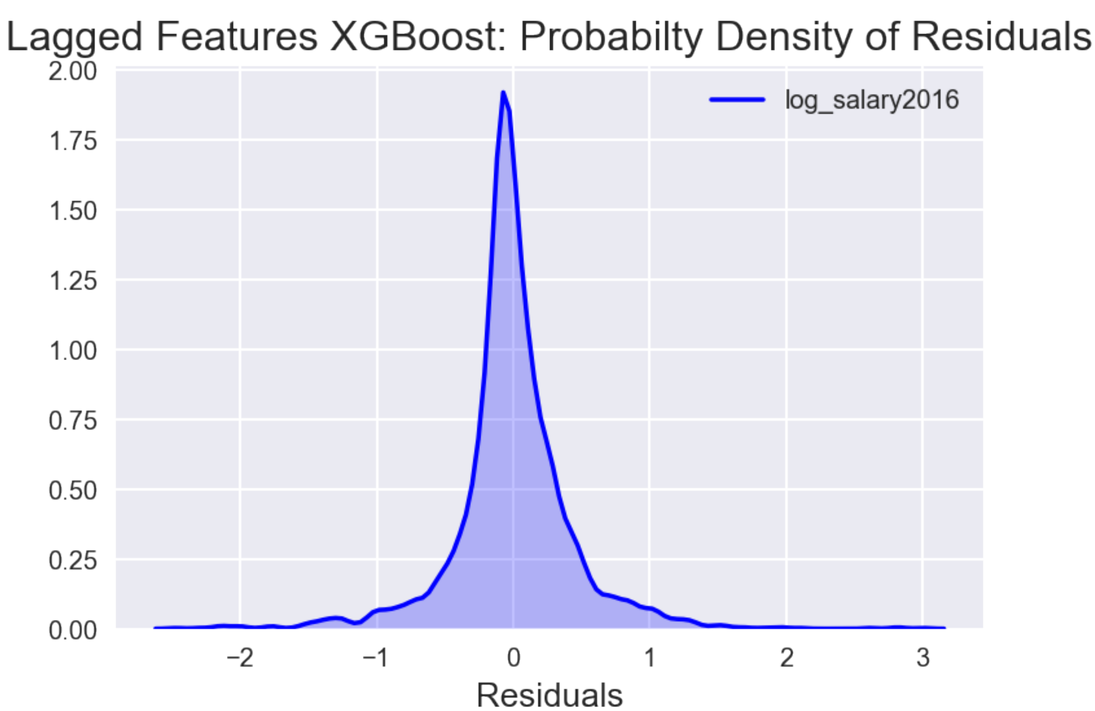
```
[0.88290786 0.89597182 0.88536109 0.8786703  0.87549786]
Score_XGB: 0.8956528121998346
```

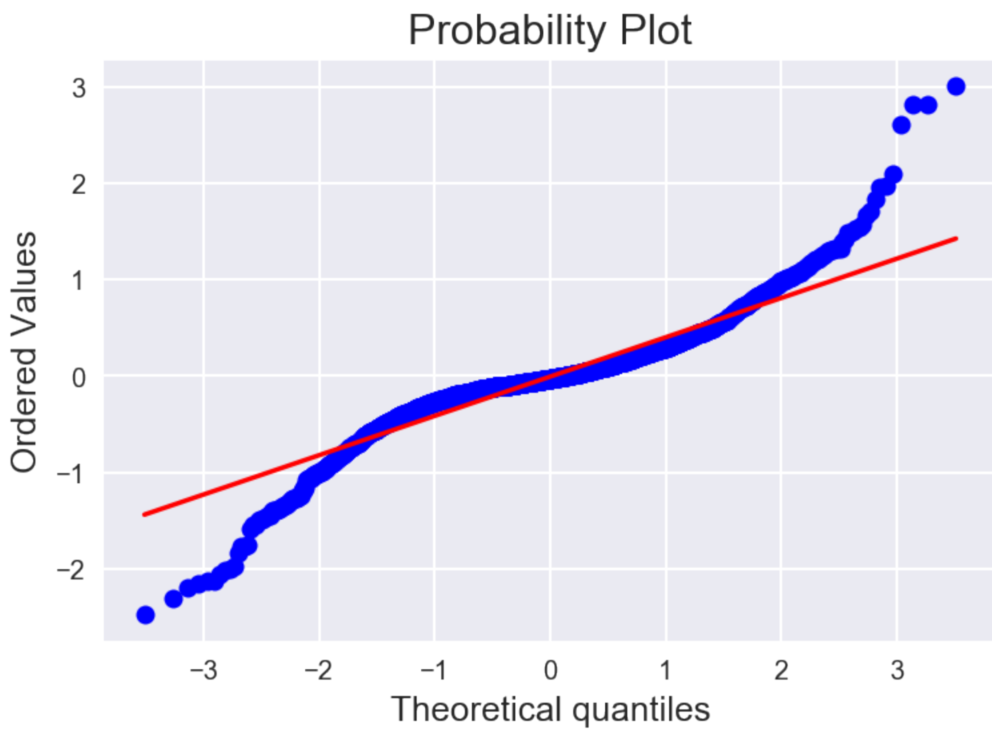
```
In [71]: import graphviz
         xgb.plot_tree(model_xgb_lag1, num_trees=5, rankdir='LR')
         fig = plt.gcf()
         fig.set_size_inches(100, 100)
         fig.savefig('tree.png')
```



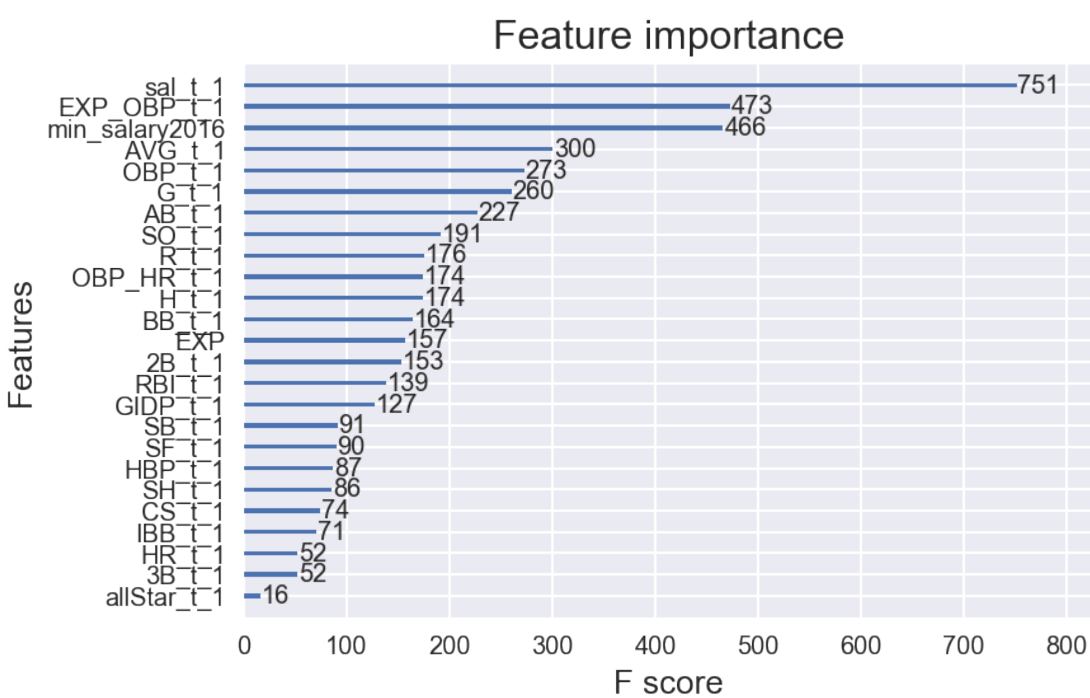
In [72]:
```
residual = y_test_lag1 - y_pred_lag1
plt.scatter(y_pred_lag1, residual)
plt.xlabel('Y-hat')
plt.ylabel('Residuals')
plt.title('Lagged Features XGBoost: Residual Plot')
plt.show()

sns.kdeplot(residual, shade=True, color="b")
plt.title("Lagged Features XGBoost: Probabilty Density of Residuals")
plt.xlabel("Residuals")
plt.show()

import pylab
import scipy.stats as scipystats
scipystats.probplot(residual, dist="norm", plot=pylab)
pylab.show()
```


Lagged Features XGBoost: Residual Plot
Residuals
Y-hat
3
2
1
0
−1
−2
12
13
14
15
16
17


Lagged Features XGBoost: Probabilty Density of Residuals
log_salary2016
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−2
−1
0
1
2
3
Residuals




```
In [73]: xgb.plot_importance(model_xgb_lag1)

Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x1c28dff7b8>
```

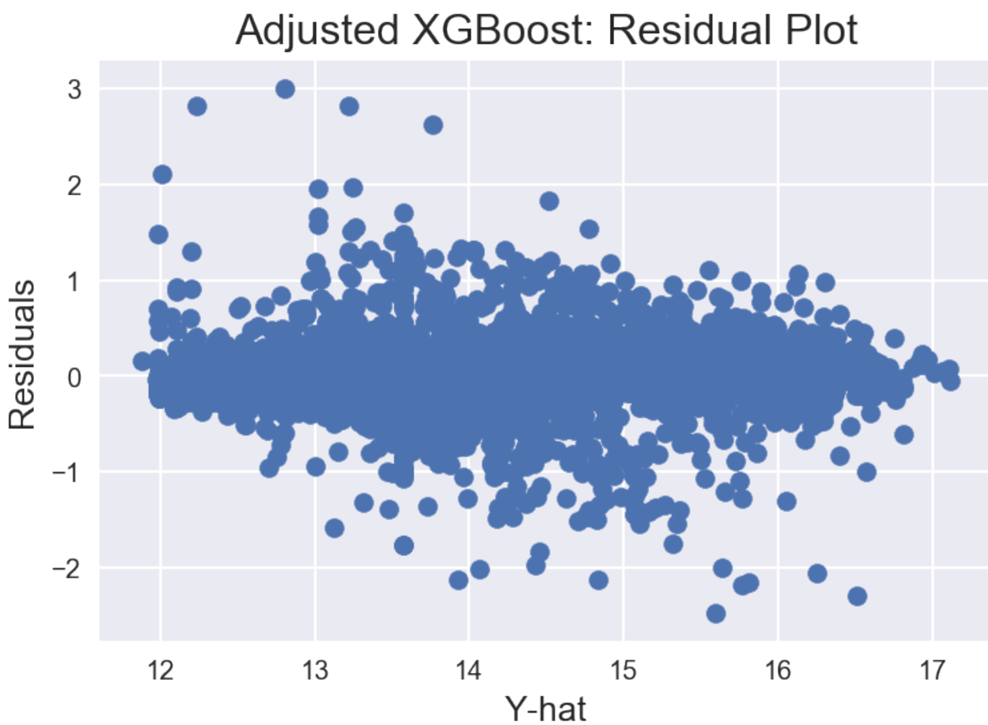


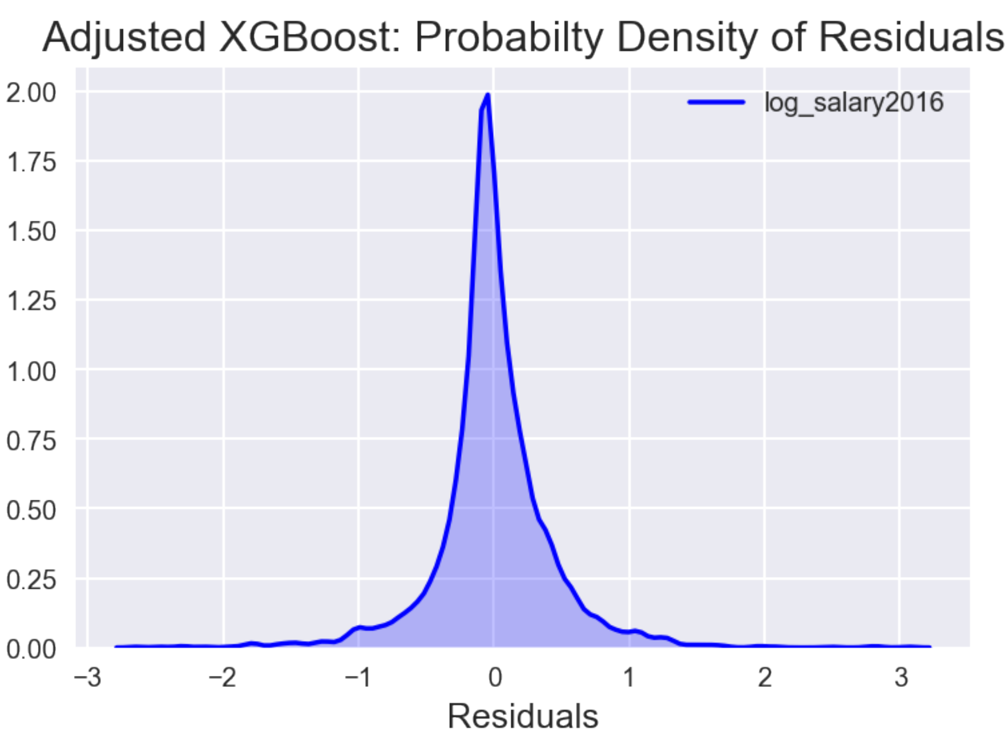
Now let's use the feature importance graph from above to try and increase our model performance. Let's try removing all the features that have a F score of less than 100. Let's also add in two-year lags for the features that have an F score of above 100. Let's see if this makes any difference in our model performance.

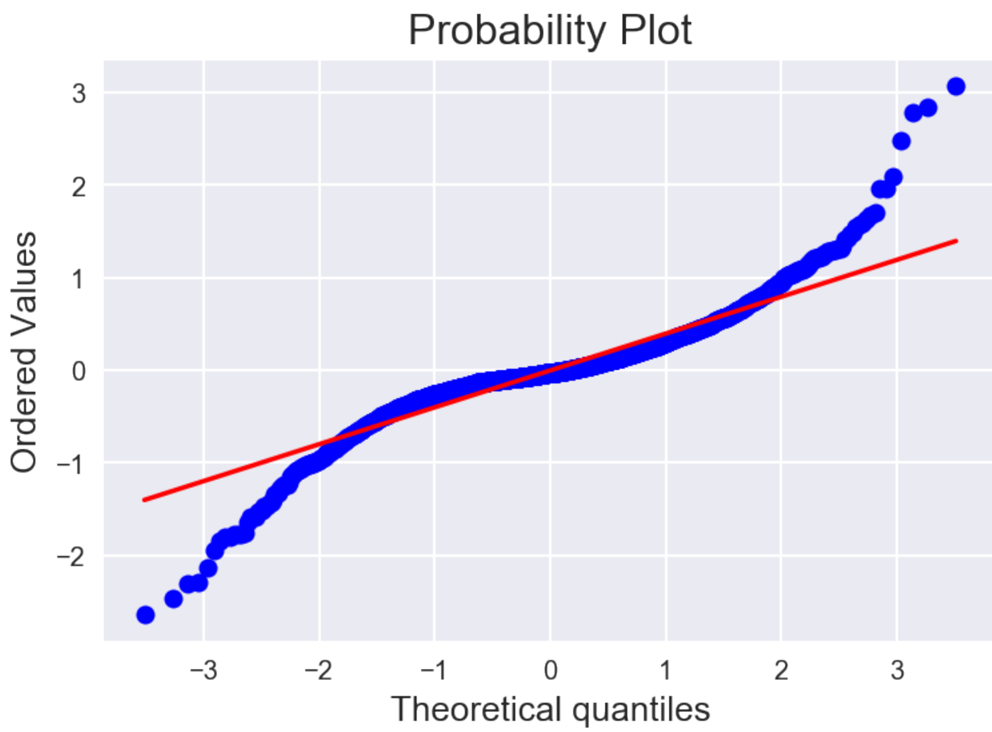
```
In [74]: model_xgb = XGBRegressor(objective='reg:linear',
                                  n_estimators=400,
                                  max_depth=6,
                                  learning_rate = 0.08,
                                  colsample_bytree=1,
                                  subsample = .9,
                                  gamma = 1,
                                  min_child_weight=5,
                                  nthreads=4,
                                  seed=314,
                                  eval_metric="rmse")

         results = cross_val_score(model_xgb, X_train, y_train, cv=kfold)
         print(results)
         model_xgb.fit(X_train, y_train)

         # make predictions for test data
         y_pred = model_xgb.predict(X_test)
         print("R-squared for XGBoost:", model_xgb.score(X_test, y_test))
```

```
[0.88830427 0.90127841 0.88880519 0.88385781 0.87499717]
R-squared for XGBoost: 0.900407025364238
```

```
In [75]: residual_2 = y_test - y_pred
         plt.scatter(y_pred, residual)
         plt.xlabel('Y-hat')
         plt.ylabel('Residuals')
         plt.title("Adjusted XGBoost: Residual Plot")
         plt.show()

         sns.kdeplot(residual_2, shade=True, color ="b")
         plt.title("Adjusted XGBoost: Probabilty Density of Residuals")
         plt.xlabel("Residuals")
         plt.show()

         import pylab
         import scipy.stats as scipystats
         scipystats.probplot(residual_2, dist="norm", plot=pylab)
         pylab.show()
```


Adjusted XGBoost: Residual Plot
Residuals
Y-hat
3
2
1
0
−1
−2
12
13
14
15
16
17


Adjusted XGBoost: Probabilty Density of Residuals
log_salary2016
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−3
−2
−1
0
1
2
3
Residuals




```
In [76]: xgb.plot_importance(model_xgb)

Out[76]: <matplotlib.axes._subplots.AxesSubplot at 0x117130c88>
```