BASEBALL | Econ 430 Project #2

Group members: Leonardo Livio Fancello, Hayden Johnson, Cade Foster, Quentin Bidwell

INTRODUCTION

All four of us are big sports fans. Leonardo follows fútbol; Quentin enjoys basketball; Hayden loves baseball; and, Cade took a basketball analytics course in college. We wanted to find a dataset where we could help predict an outcome in one of our favorite sports. Sports analytics is a growing industry, and this project is a good way to break into the industry. The sport with the most amount of data is baseball, so naturally we turned to this sport for our analysis. Our dataset can be found on Baseball Savant. The problem we are addressing is how can we increase a player's expected batting average based on other statistical variables that are tracked in baseball. The movie Moneyball used linear regressions to calculate how many wins they needed in a season to make it to the playoffs, and how to score more runs to win games. They wanted players who "got on base." We want to increase our odds of winning but by increasing a player's batting average. If all 9 players in a lineup are batting 0.300, and every player is guaranteed at least 3 plate appearances a game, then every player in the lineup will get a hit at least once a game on average. This conclusion is a compelling argument in our eyes to win games for a baseball team.

Explanation of Variables

xba = Expected Batting Average: A Statcast metric that measures the likelihood that a batted ball will become a hit
xslg = Expected Slugging Percentage: Formulated using exit velocity, launch angle and, on certain types of batted balls, Sprint Speed
xwoba = Expected Weighted On-base Average: Formulated using exit velocity, launch angle and, on certain types of batted balls, Sprint Speed
xobp = Expected On-base Percentage: Measures the likelihood a batter will reach base per plate appearance
xiso = Expected Isolated Power: Measures the the raw power of a hitter by taking only extra-base hits -- and the type of extra-base hit -- into account
exit_velocity_avg = Average speed with which the ball leaves the bat
barrel_batted_rate = To be Barreled, a batted ball requires an exit velocity of at least 98 mph. At that speed, balls struck with a launch angle between 26-30 degrees always garner Barreled classification. For every mph over 98, the range of launch angles expands
solidcontact_percent = Represent the percentage of a hitter's batted balls that have been hit with a certain amount of authority. The percentages will sum to 100%, totaling all of a player's batted balls hit
avg_best_speed = An average of 50% of his hardest hit balls
avg_hyper_speed = Takes the average of a player's batted ball velocities subtracted from 88. Anything below 88 mph is considered 0
z_swing_miss_percent = An average of how many times a player swings and misses at a ball in the strike-zone

oz_swing_miss_percent = The percent of pitches outside of the zone that a hitter swings at
hp_to_1b = The amount of time in seconds that it takes for player to reach 1st base
sprint_speed = A measurement of a player's top running speed, expressed in "feet per second in a player's fastest one-second window"
AverageSpeed = Indicator variable which will be 1 if the player's sprint speed is in the top 50% of players, and 0 if the player's sprint speed is in the bottom 50% of players

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
import statsmodels.formula.api as smf
import statsmodels.api as sm
import statsmodels.stats.api as sms
import statsmodels.stats.outliers_influence as smo
import patsy as pt
import statsmodels.regression.linear_model as rg
import statsmodels.stats.diagnostic as dg
from RegscorePy import mallow
from BorutaShap import BorutaShap
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn import metrics
from simple_colors import *
from matplotlib.ticker import ScalarFormatter
```

VARIABLE SELECTION

In [2]:
```
ba = pd.read_csv("stats.csv")
ba.loc[ba['sprint_speed'] < ba.sprint_speed.mean(), 'AverageSpeed'] = 0
ba.loc[ba['sprint_speed'] > ba.sprint_speed.mean(), 'AverageSpeed'] = 1
ba.head()
```

Out[2]:

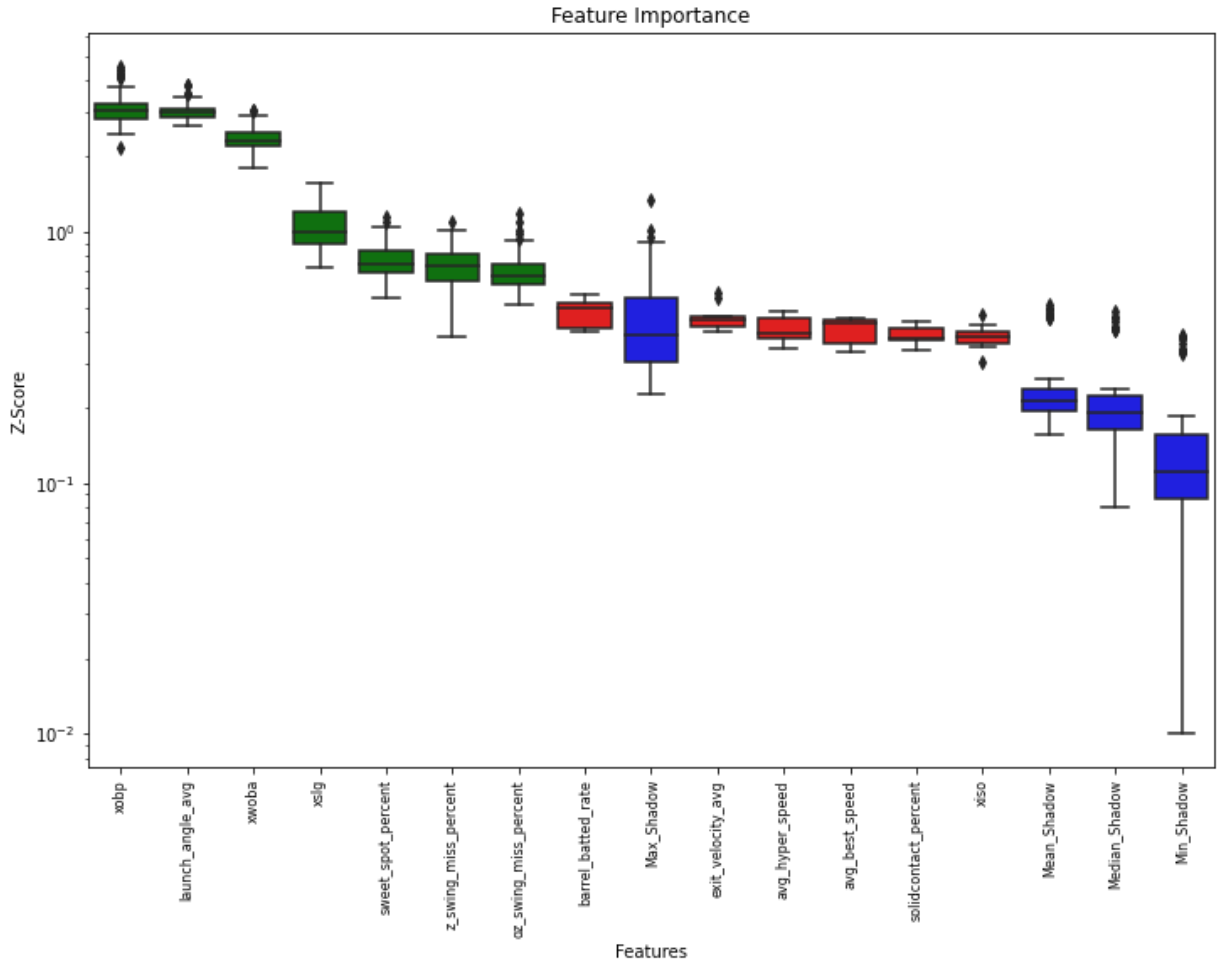
	last_name	first_name	player_id	year	xba	xslg	xwoba	xobp	xiso	exit_velocity_avg	...
0	Cruz Jr.	Nelson	443558	2022	0.241	0.399	0.320	0.323	0.157	90.9	...
1	Blackmon	Charlie	453568	2022	0.256	0.376	0.301	0.308	0.120	86.2	...
2	McCutchen	Andrew	457705	2022	0.252	0.406	0.325	0.329	0.155	89.1	...
3	Turner	Justin	457759	2022	0.267	0.427	0.339	0.343	0.160	89.5	...
4	Andrus	Elvis	462101	2022	0.245	0.360	0.293	0.300	0.114	87.9	...

5 rows × 23 columns

The best 10 predictors

In [4]:
```
x = ba.iloc[:,5:18]
y = ba['xba']

Feature_Selector = BorutaShap(importance_measure='shap', classification=False)
Feature_Selector.fit(X=x, y=y, n_trials=50, random_state=0)
Feature_Selector.plot(which_features='all')
```

```
  0%|          | 0/50 [00:00<?, ?it/s]

7 attributes confirmed important: ['sweet_spot_percent', 'xwoba', 'xobp', 'xsl
g', 'z_swing_miss_percent', 'launch_angle_avg', 'oz_swing_miss_percent']
6 attributes confirmed unimportant: ['barrel_batted_rate', 'xiso', 'solidcontac
t_percent', 'avg_hyper_speed', 'exit_velocity_avg', 'avg_best_speed']
0 tentative attributes remains: []
```



Predictors selection

In [3]:
```
mr_mod = smf.ols(formula = 'xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_
mr_fit = mr_mod.fit()
mr_fit.summary()
```

Out[3]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.815
Model:	OLS	**Adj. R-squared:**	0.804
Method:	Least Squares	**F-statistic:**	76.53
Date:	Mon, 14 Nov 2022	**Prob (F-statistic):**	1.32e-41
Time:	20:36:45	**Log-Likelihood:**	424.69
No. Observations:	130	**AIC:**	-833.4
Df Residuals:	122	**BIC:**	-810.4
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.1351	0.016	8.614	0.000	0.104	0.166
xobp	0.5632	0.218	2.585	0.011	0.132	0.994
launch_angle_avg	-0.0021	0.000	-9.916	0.000	-0.003	-0.002
xwoba	-0.7886	0.406	-1.944	0.054	-1.592	0.014
xslg	0.4398	0.132	3.340	0.001	0.179	0.700
sweet_spot_percent	0.0017	0.000	5.976	0.000	0.001	0.002
z_swing_miss_percent	-0.0014	0.000	-6.270	0.000	-0.002	-0.001
AverageSpeed	0.0018	0.002	1.021	0.309	-0.002	0.005

Omnibus:	4.346	**Durbin-Watson:**	2.215
Prob(Omnibus):	0.114	**Jarque-Bera (JB):**	2.860
Skew:	-0.183	**Prob(JB):**	0.239
Kurtosis:	2.372	**Cond. No.**	2.30e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.3e+04. This might indicate that there are strong multicollinearity or other numerical problems.

DESCRIPTIVE ANALYSIS

Basic analysis

In [26]:
```
ba.isnull().sum()
```

Out[26]:
```
last_name                 0
 first_name               0
player_id                 0
year                      0
xba                       0
xslg                      0
xwoba                     0
xobp                      0
xiso                      0
exit_velocity_avg         0
launch_angle_avg          0
sweet_spot_percent        0
barrel_batted_rate        0
solidcontact_percent      0
avg_best_speed            0
avg_hyper_speed           0
z_swing_miss_percent      0
oz_swing_miss_percent     0
oz_contact_percent        0
hp_to_1b                  0
sprint_speed              0
Unnamed: 21             130
AverageSpeed              0
dtype: int64
```

In [4]:
```
ba = ba.drop(['Unnamed: 21'], axis=1)
```

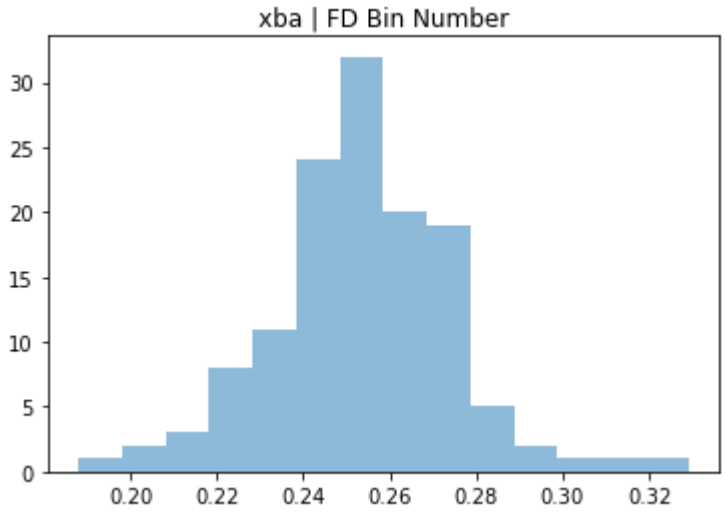
We are dropping the column *Unnamed: 21* as it is a column made of all NULL values.

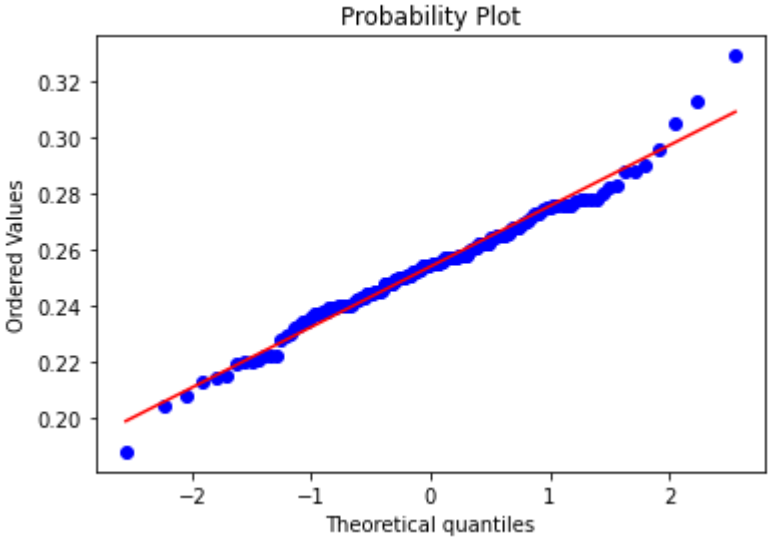
In [4]:
```
ba.describe()
```

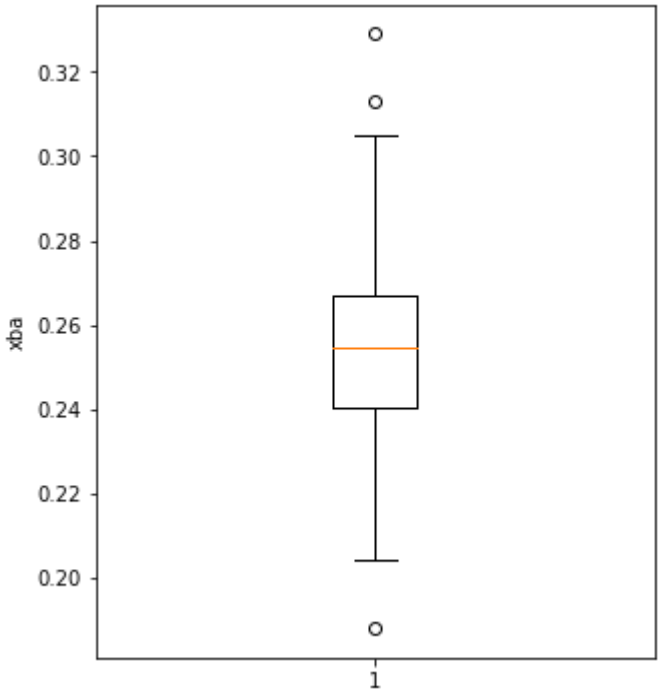
Out[4]:

	player_id	year	xba	xslg	xwoba	xobp	xiso	exit_vel
count	130.000000	130.0	130.000000	130.000000	130.000000	130.000000	130.000000	1
mean	607829.930769	2022.0	0.253931	0.420654	0.328285	0.327485	0.166792	
std	61551.798138	0.0	0.021504	0.061844	0.032098	0.029719	0.052620	
min	443558.000000	2022.0	0.188000	0.291000	0.267000	0.267000	0.065000	
25%	571987.750000	2022.0	0.240250	0.379250	0.306500	0.306000	0.135000	
50%	622163.500000	2022.0	0.254500	0.416000	0.326000	0.327000	0.158000	
75%	663574.000000	2022.0	0.266750	0.454000	0.346500	0.342000	0.205500	
max	683734.000000	2022.0	0.329000	0.706000	0.463000	0.430000	0.401000	

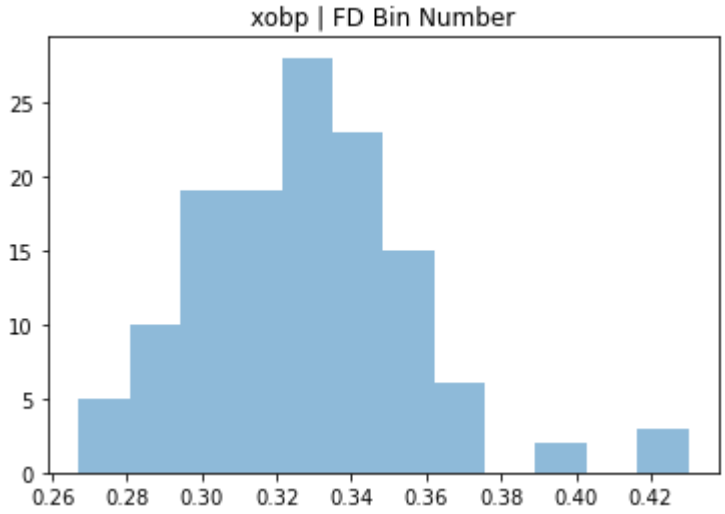
In [14]:
```
plt.hist(ba.xba, alpha = .5, bins = "fd")
plt.title("xba | FD Bin Number")
plt.show()
stats.probplot(ba.xba, dist = "norm", plot = plt)
plt.show()
plt.figure(figsize=(5,6))
plt.boxplot(ba.xba)
plt.ylabel("xba")
plt.show()
```

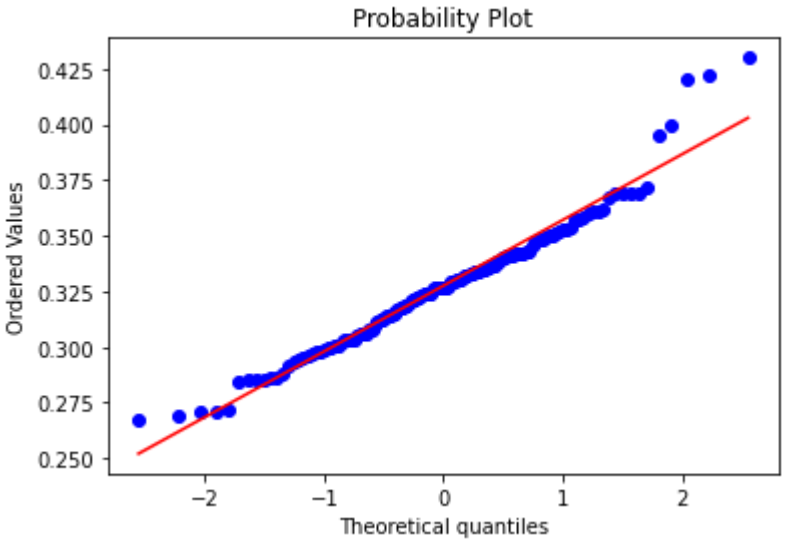


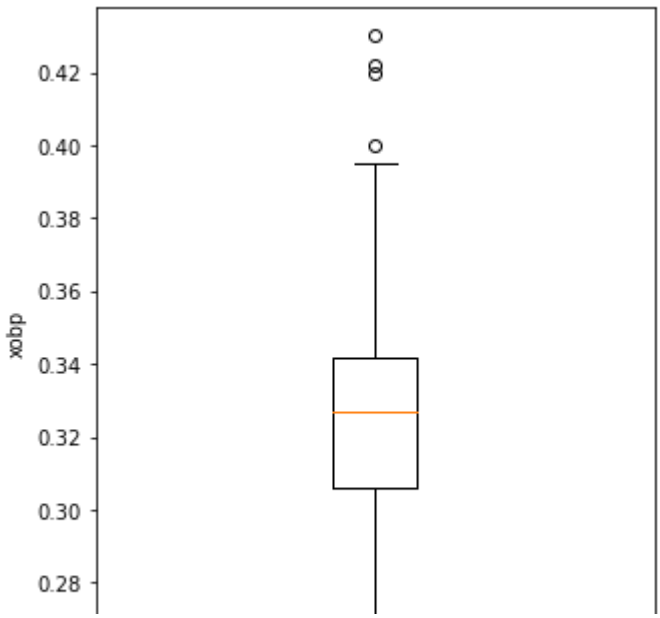



0.32
0.30
0.28
0.26
0.24
0.22
0.20
xba
1

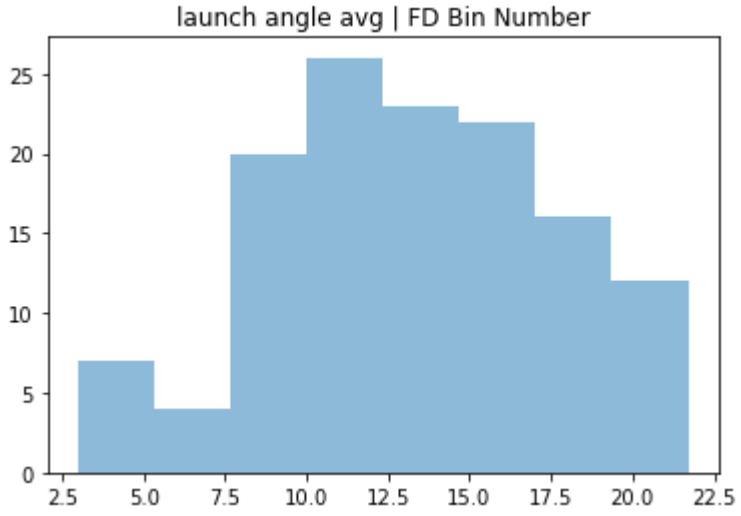
In [15]:
```
plt.hist(ba.xobp, alpha = .5, bins = "fd")
plt.title("xobp | FD Bin Number")
plt.show()
stats.probplot(ba.xobp, dist = "norm", plot = plt)
plt.show()
plt.figure(figsize=(5,6))
plt.boxplot(ba.xobp)
plt.ylabel("xobp")
plt.show()
```

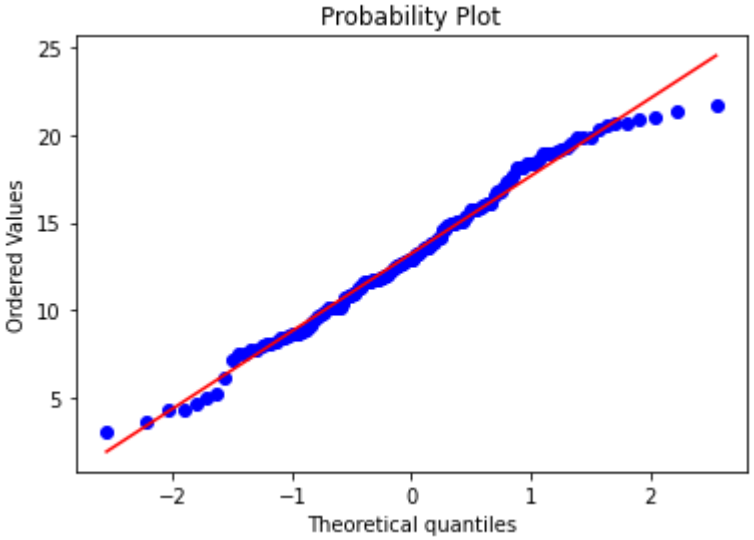


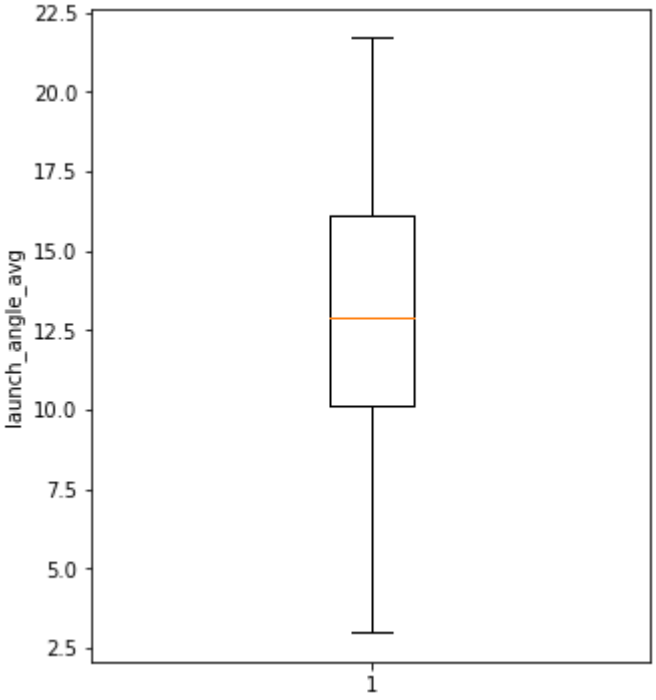



0.42
0.40
0.38
0.36
xobp
0.34
0.32
0.30
0.28

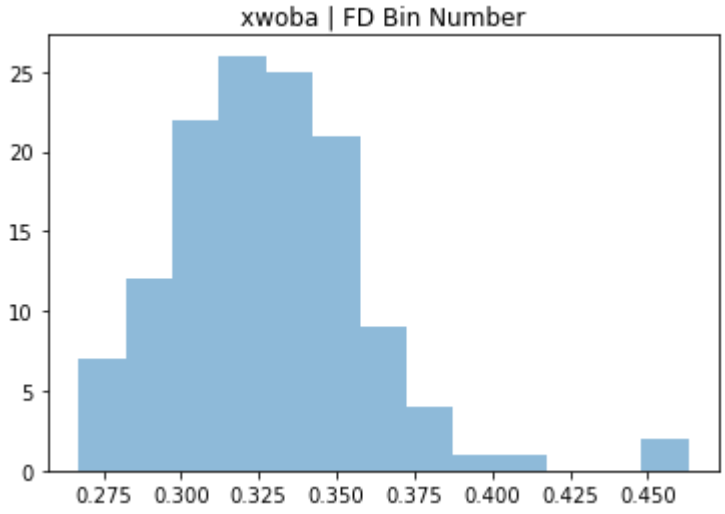
In [16]:
```
plt.hist(ba.launch_angle_avg, alpha = .5, bins = "fd")
plt.title("launch angle avg | FD Bin Number")
plt.show()
stats.probplot(ba.launch_angle_avg, dist = "norm", plot = plt)
plt.show()
plt.figure(figsize=(5,6))
plt.boxplot(ba.launch_angle_avg)
plt.ylabel("launch_angle_avg")
plt.show()
```

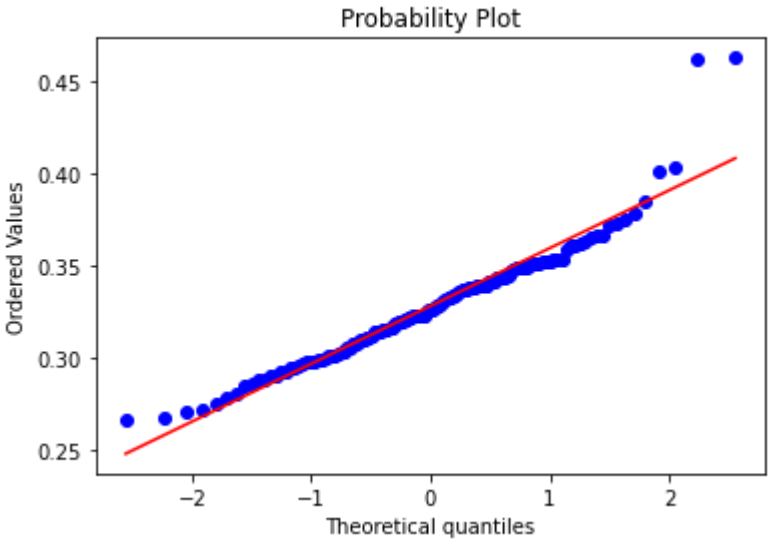


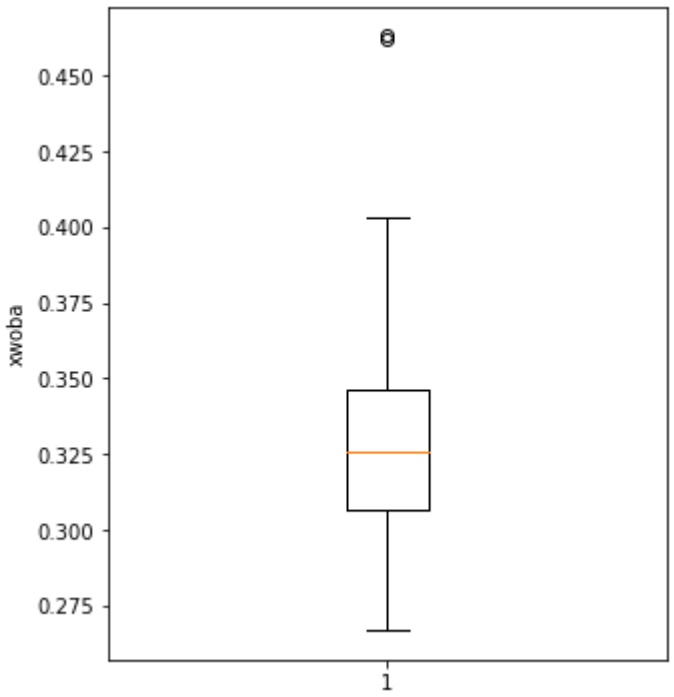



22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
launch_angle_avg
1

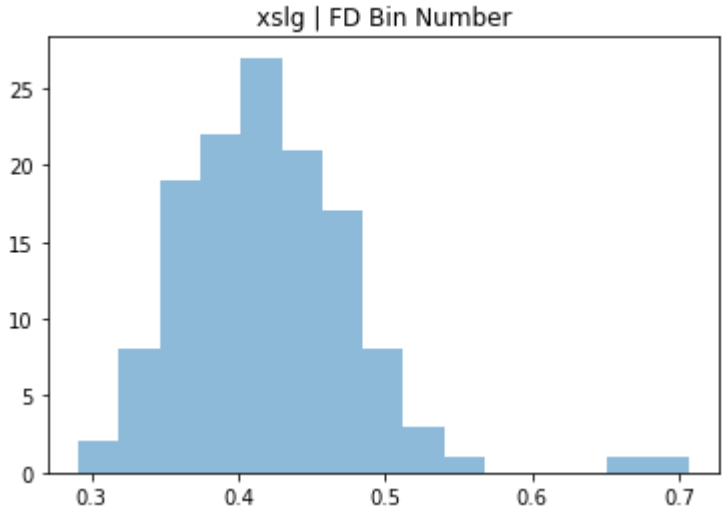
In [17]:
```
plt.hist(ba.xwoba, alpha = .5, bins = "fd")
plt.title("xwoba | FD Bin Number")
plt.show()
stats.probplot(ba.xwoba, dist = "norm", plot = plt)
plt.show()
plt.figure(figsize=(5,6))
plt.boxplot(ba.xwoba)
plt.ylabel("xwoba")
plt.show()
```

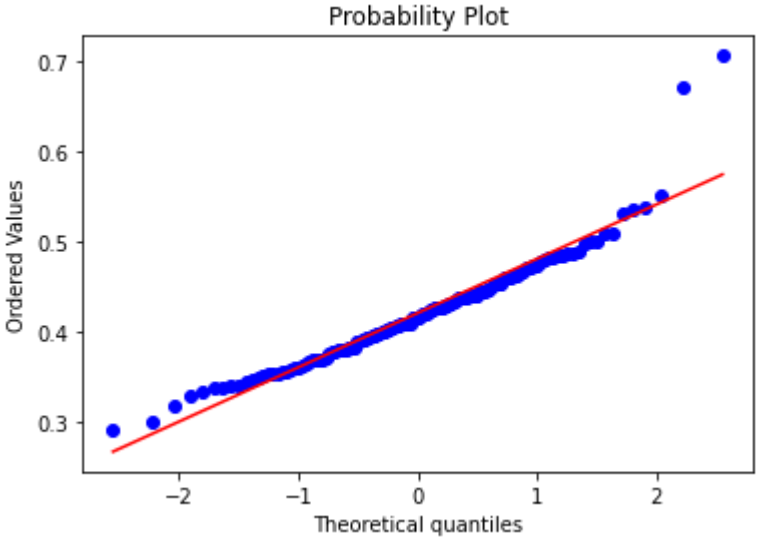


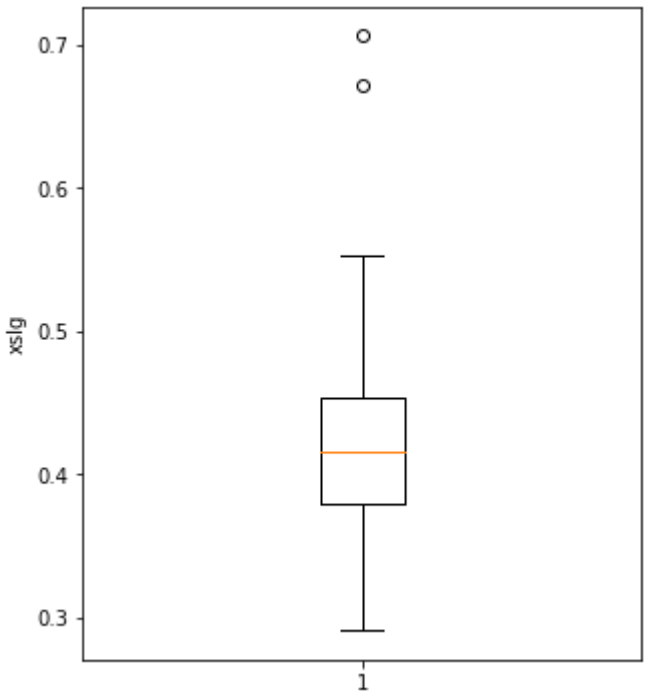



0.450
0.425
0.400
0.375
0.350
0.325
0.300
0.275
xwoba
1

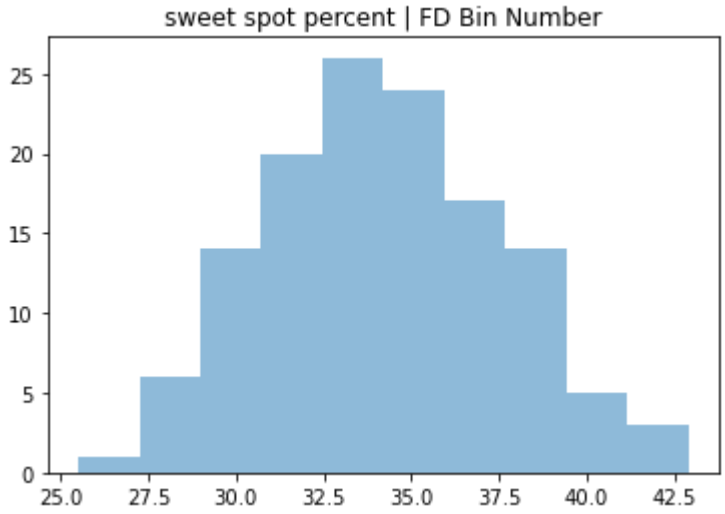
In [18]:
```
plt.hist(ba.xslg, alpha = .5, bins = "fd")
plt.title("xslg | FD Bin Number")
plt.show()
stats.probplot(ba.xslg, dist = "norm", plot = plt)
plt.show()
plt.figure(figsize=(5,6))
plt.boxplot(ba.xslg)
plt.ylabel("xslg")
plt.show()
```

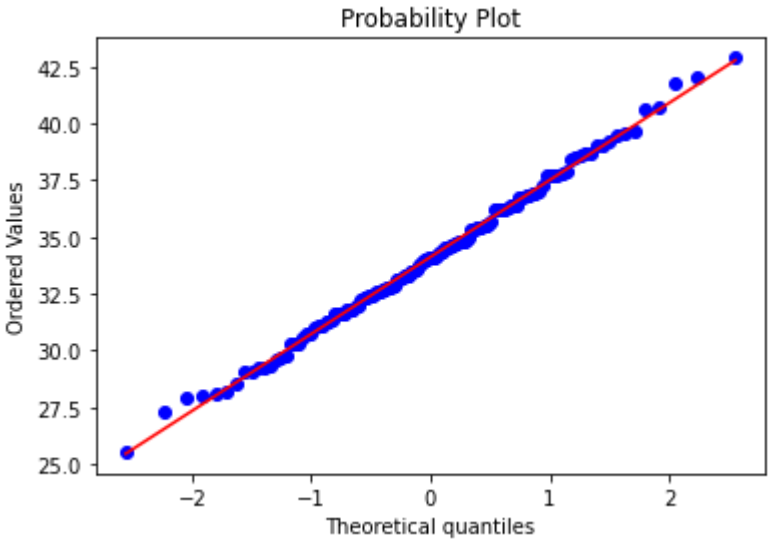


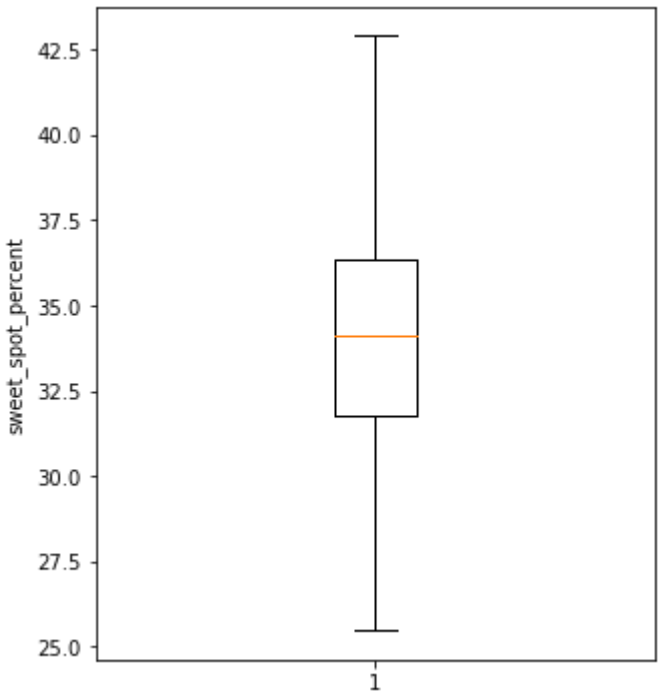



0.7
0.6
0.5
0.4
0.3
xslg
1

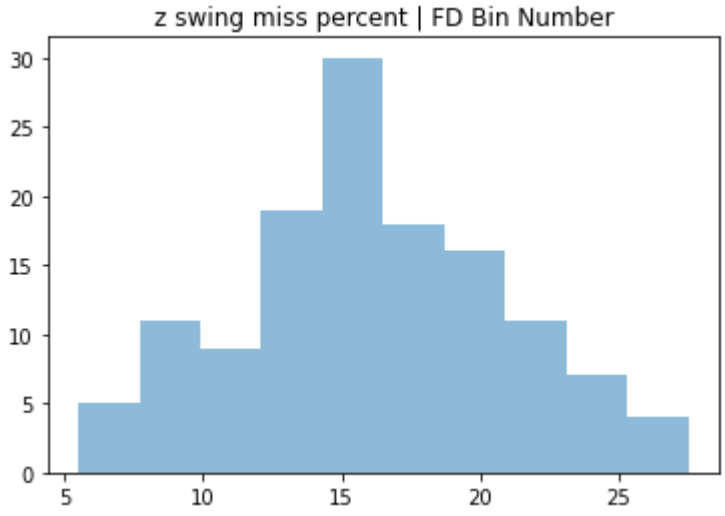
In [19]:
```
plt.hist(ba.sweet_spot_percent, alpha = .5, bins = "fd")
plt.title("sweet spot percent | FD Bin Number")
plt.show()
stats.probplot(ba.sweet_spot_percent, dist = "norm", plot = plt)
plt.show()
plt.figure(figsize=(5,6))
plt.boxplot(ba.sweet_spot_percent)
plt.ylabel("sweet_spot_percent")
plt.show()
```

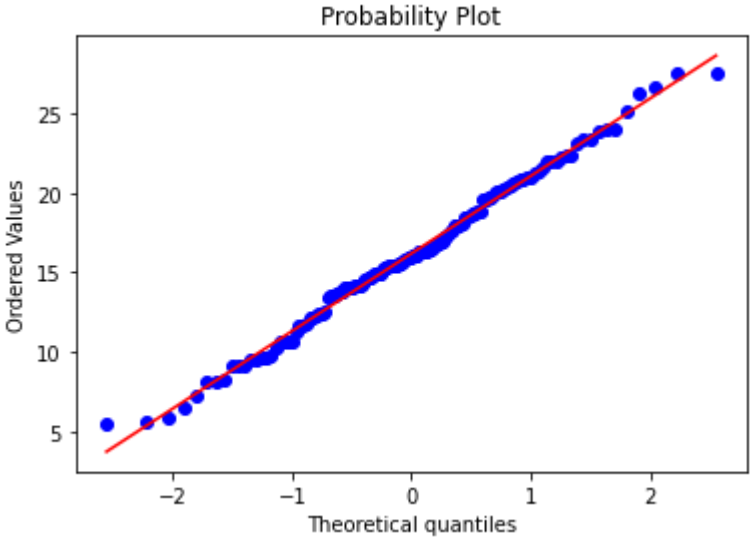


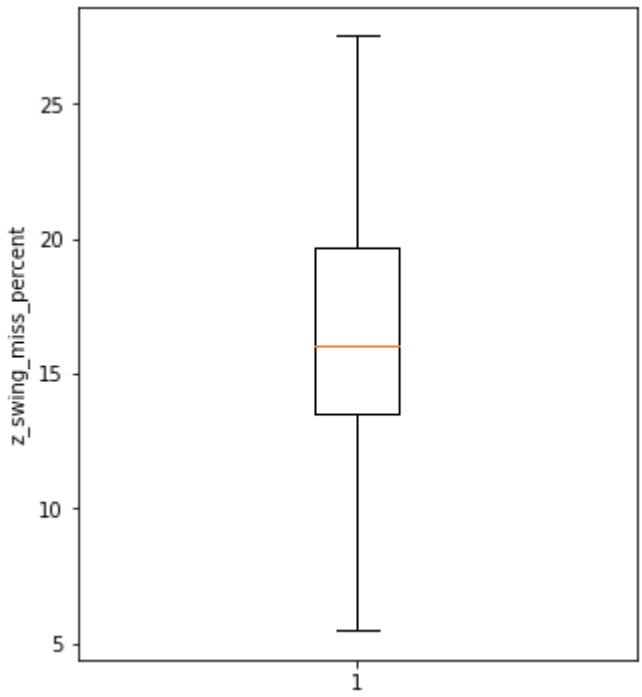



sweet_spot_percent
42.5
40.0
37.5
35.0
32.5
30.0
27.5
25.0
1

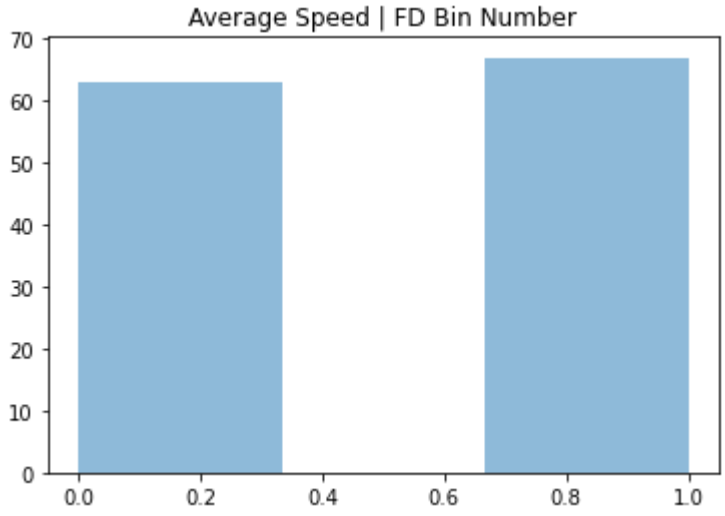
In [20]:
```
plt.hist(ba.z_swing_miss_percent, alpha = .5, bins = "fd")
plt.title("z swing miss percent | FD Bin Number")
plt.show()
stats.probplot(ba.z_swing_miss_percent, dist = "norm", plot = plt)
plt.show()
plt.figure(figsize=(5,6))
plt.boxplot(ba.z_swing_miss_percent)
plt.ylabel("z_swing_miss_percent")
plt.show()
```

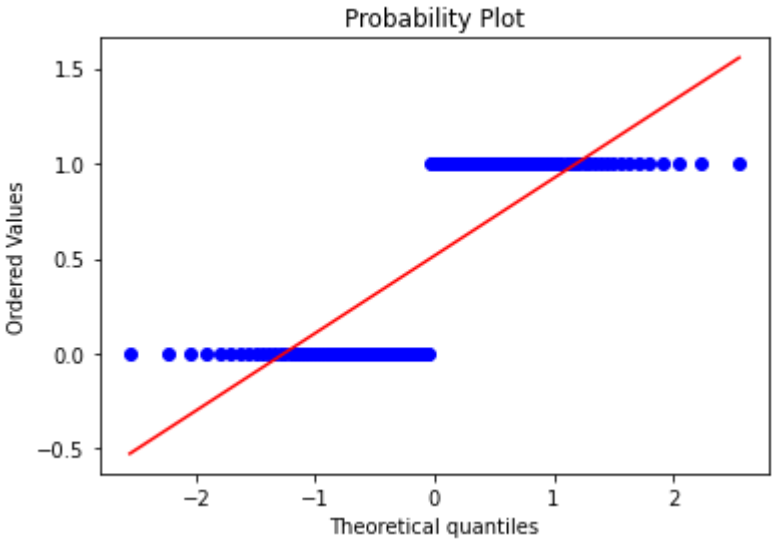


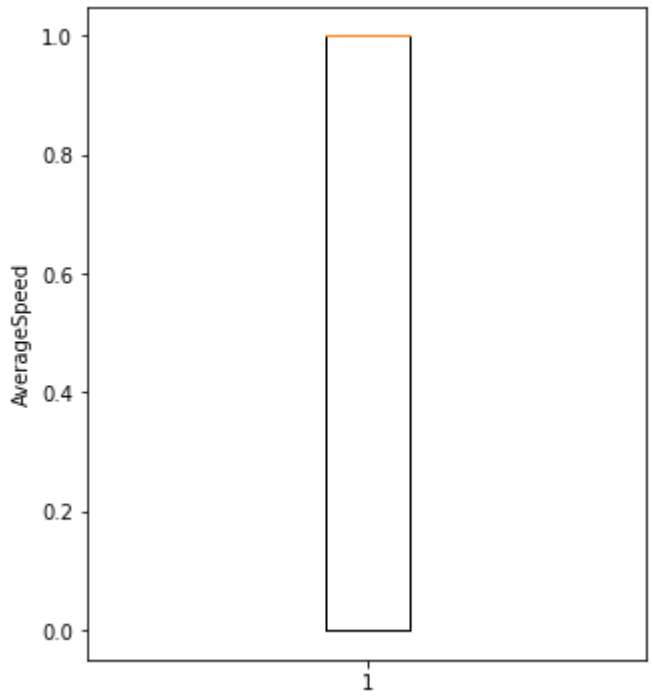



z_swing_miss_percent
25
20
15
10
5
1

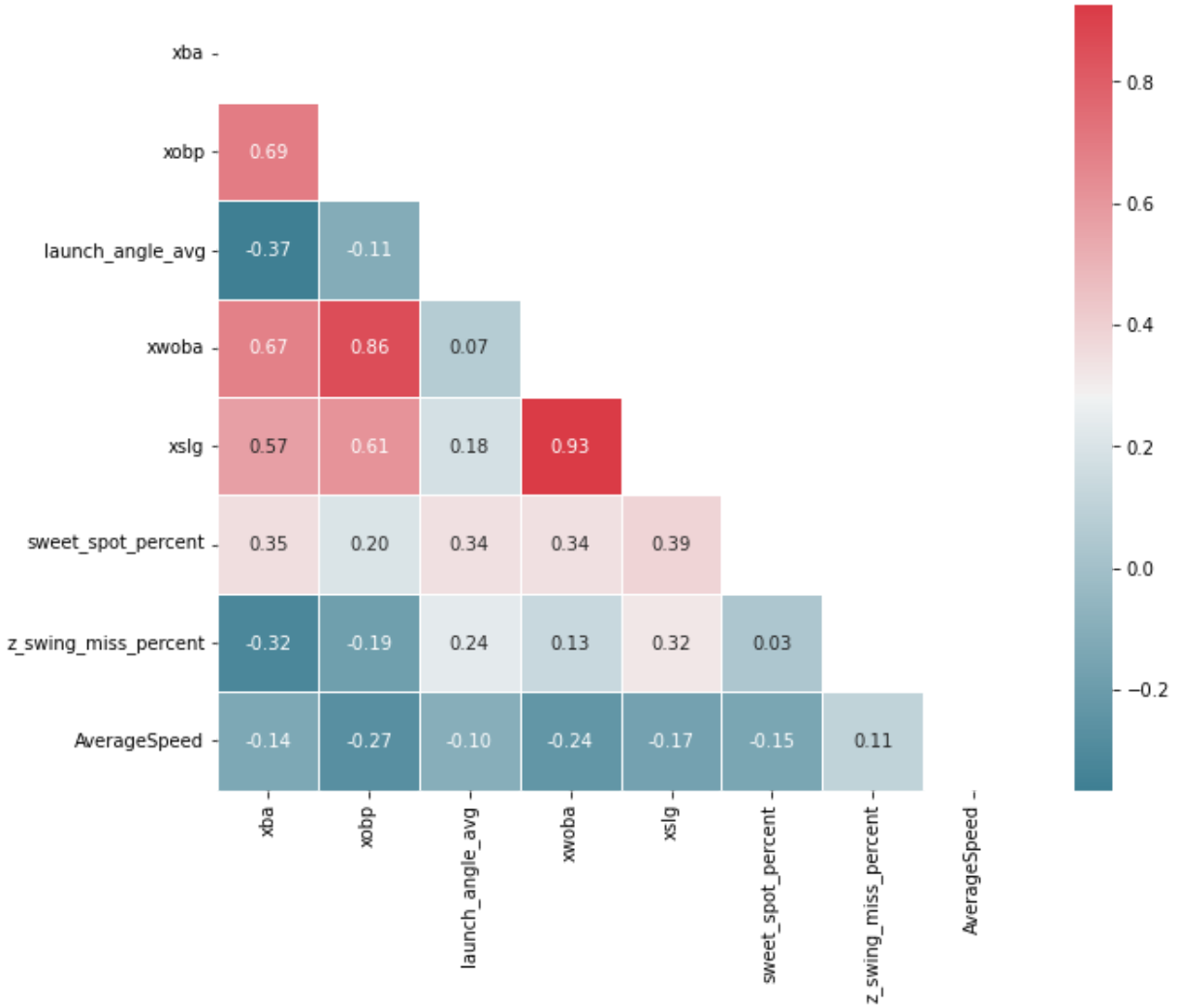
In [21]:
```
plt.hist(ba.AverageSpeed, alpha = .5, bins = "fd")
plt.title("Average Speed | FD Bin Number")
plt.show()
stats.probplot(ba.AverageSpeed, dist = "norm", plot = plt)
plt.show()
plt.figure(figsize=(5,6))
plt.boxplot(ba.AverageSpeed)
plt.ylabel("AverageSpeed")
plt.show()
```






1.0
0.8
0.6
0.4
0.2
0.0
AverageSpeed
1

In [22]:
```
#correlation plot of all dependant variables
sub_df =  ba[["xba", "xobp", "launch_angle_avg", "xwoba", "xslg", "sweet_spot_per
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", ma
plt.show()
```

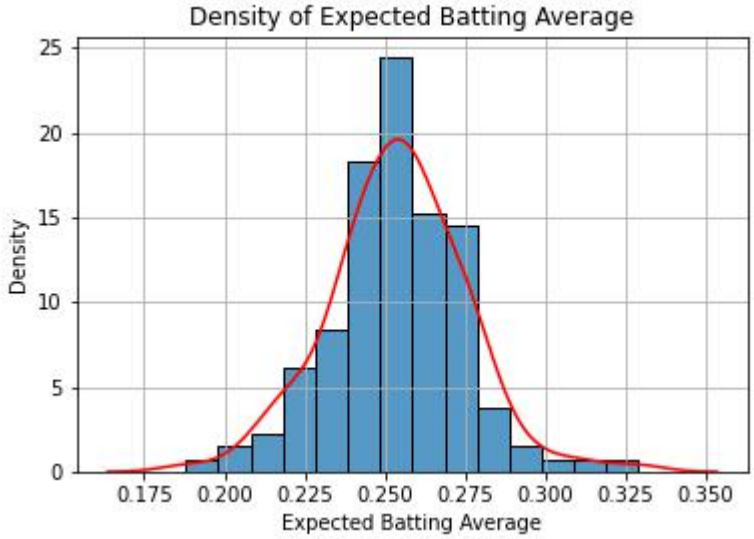


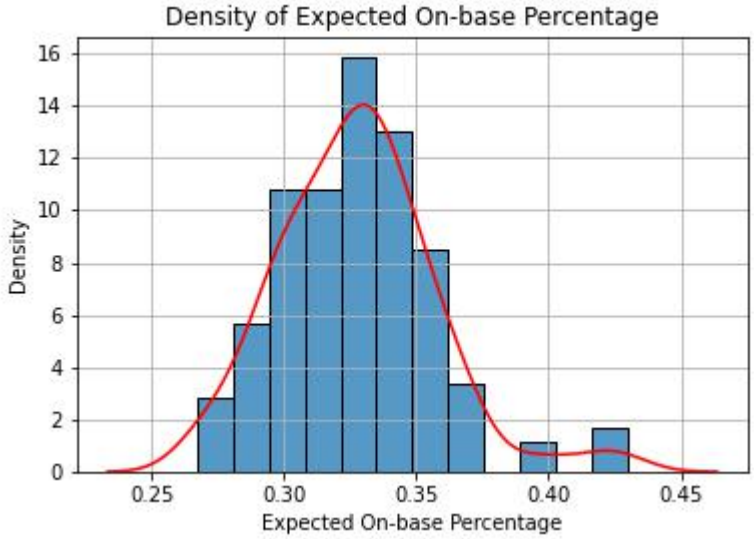
Density plots

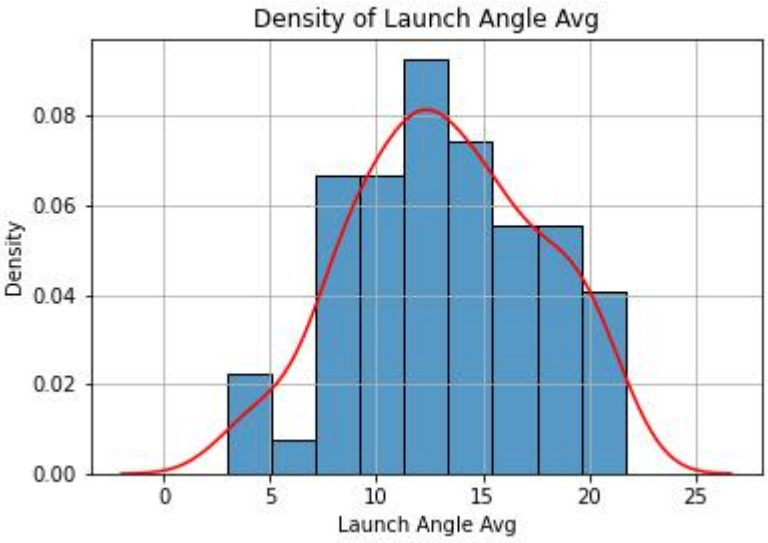
In [12]:

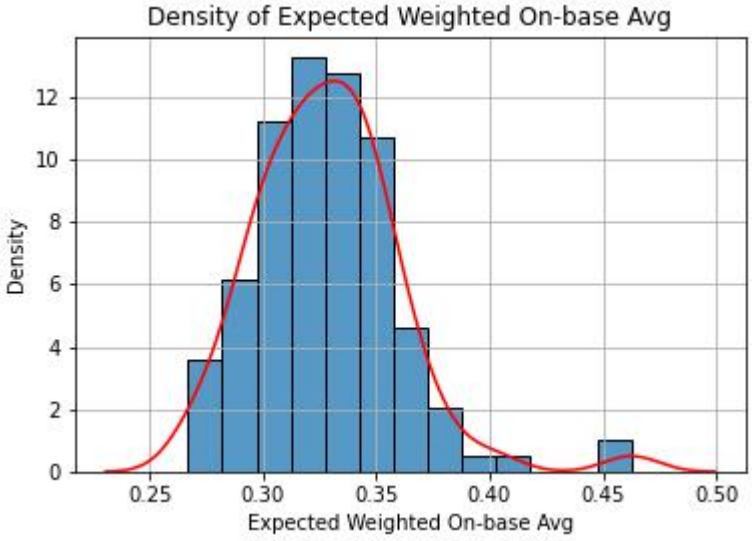
```
sns.histplot(ba.xba, stat="density")
sns.kdeplot(ba.xba, color="red")
plt.title("Density of Expected Batting Average")
plt.xlabel("Expected Batting Average")
plt.ylabel("Density")
plt.grid()
plt.savefig("xba.jpg")
plt.close()

sns.histplot(ba.xobp, stat="density")
sns.kdeplot(ba.xobp, color="red")
plt.title("Density of Expected On-base Percentage")
plt.xlabel("Expected On-base Percentage")
plt.ylabel("Density")
plt.grid()
plt.savefig("xobp.jpg")
plt.close()

sns.histplot(ba.launch_angle_avg, stat="density")
sns.kdeplot(ba.launch_angle_avg, color="red")
plt.title("Density of Launch Angle Avg")
plt.xlabel("Launch Angle Avg")
plt.ylabel("Density")
plt.grid()
plt.savefig("launch_angle_avg.jpg")
plt.close()

sns.histplot(ba.xwoba, stat="density")
sns.kdeplot(ba.xwoba, color="red")
plt.title("Density of Expected Weighted On-base Avg")
plt.xlabel("Expected Weighted On-base Avg")
plt.ylabel("Density")
plt.grid()
plt.savefig("xwoba.jpg")
plt.close()

sns.histplot(ba.xslg, stat="density")
sns.kdeplot(ba.xslg, color="red")
plt.title("Density of Expected Slugging Percentage")
plt.xlabel("Expected Slugging Percentage")
plt.ylabel("Density")
plt.grid()
plt.savefig("xlsg.jpg")
plt.close()

sns.histplot(ba.sweet_spot_percent, stat="density")
sns.kdeplot(ba.sweet_spot_percent, color="red")
plt.title("Density of Sweet Spot Percentage")
plt.xlabel("Sweet Spot Percentage")
plt.ylabel("Density")
plt.grid()
plt.savefig("sweet_spot_percent.jpg")
plt.close()

sns.histplot(ba.z_swing_miss_percent, stat="density")
sns.kdeplot(ba.z_swing_miss_percent, color="red")
```

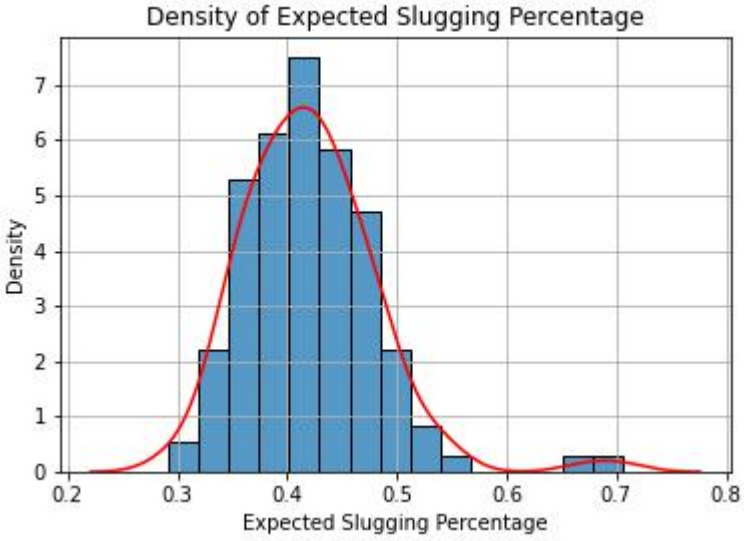
```
plt.title("Density of Swing Miss Percentage")
plt.xlabel("Swing Miss Percentage")
plt.ylabel("Density")
plt.grid()
plt.savefig("z_swing_miss_percent.jpg")
plt.close()

sns.histplot(ba.AverageSpeed, stat="density")
sns.kdeplot(ba.AverageSpeed, color="red")
plt.title("Density of Average Speed")
plt.xlabel("Average Speed")
plt.ylabel("Density")
plt.grid()
plt.savefig("AverageSpeed.jpg")
plt.close()
```

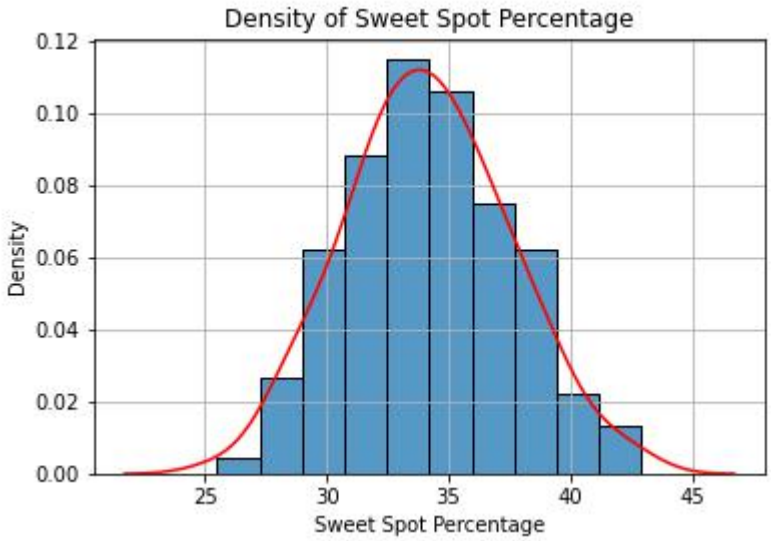


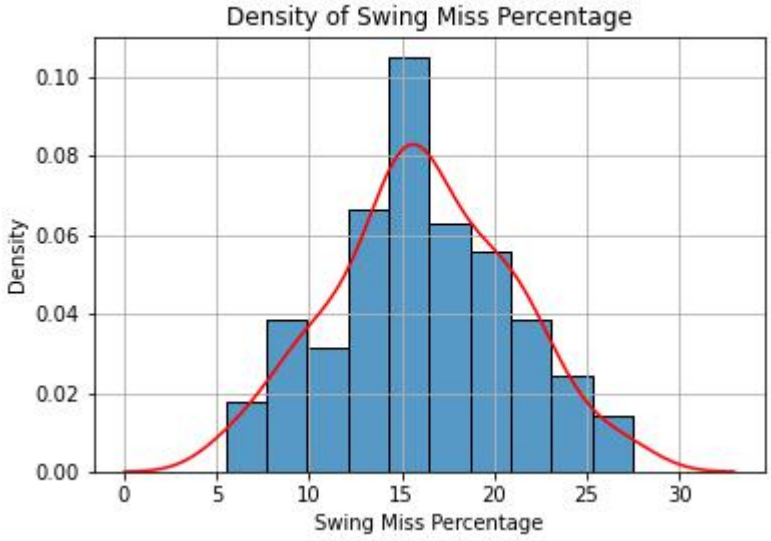


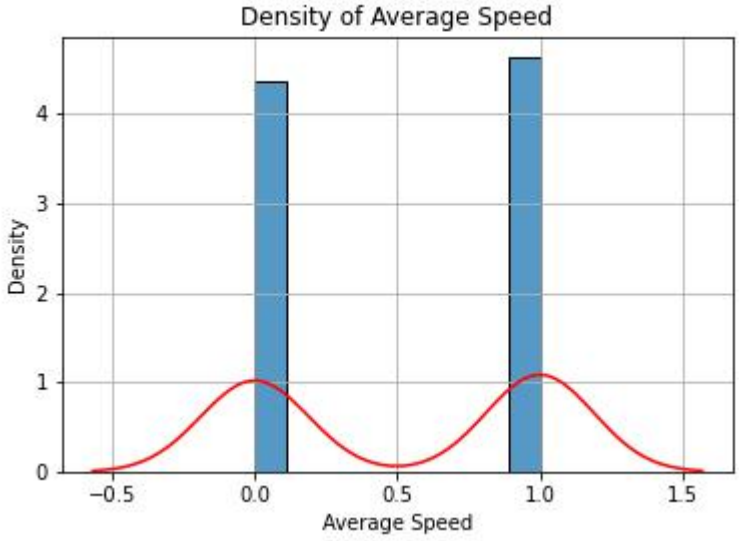

Density of Launch Angle Avg
Density
0.00
0.02
0.04
0.06
0.08
0
5
10
15
20
25
Launch Angle Avg


Density of Expected Weighted On-base Avg
Density
0
2
4
6
8
10
12
0.25
0.30
0.35
0.40
0.45
0.50
Expected Weighted On-base Avg


Density of Expected Slugging Percentage
Density
0
1
2
3
4
5
6
7
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Expected Slugging Percentage



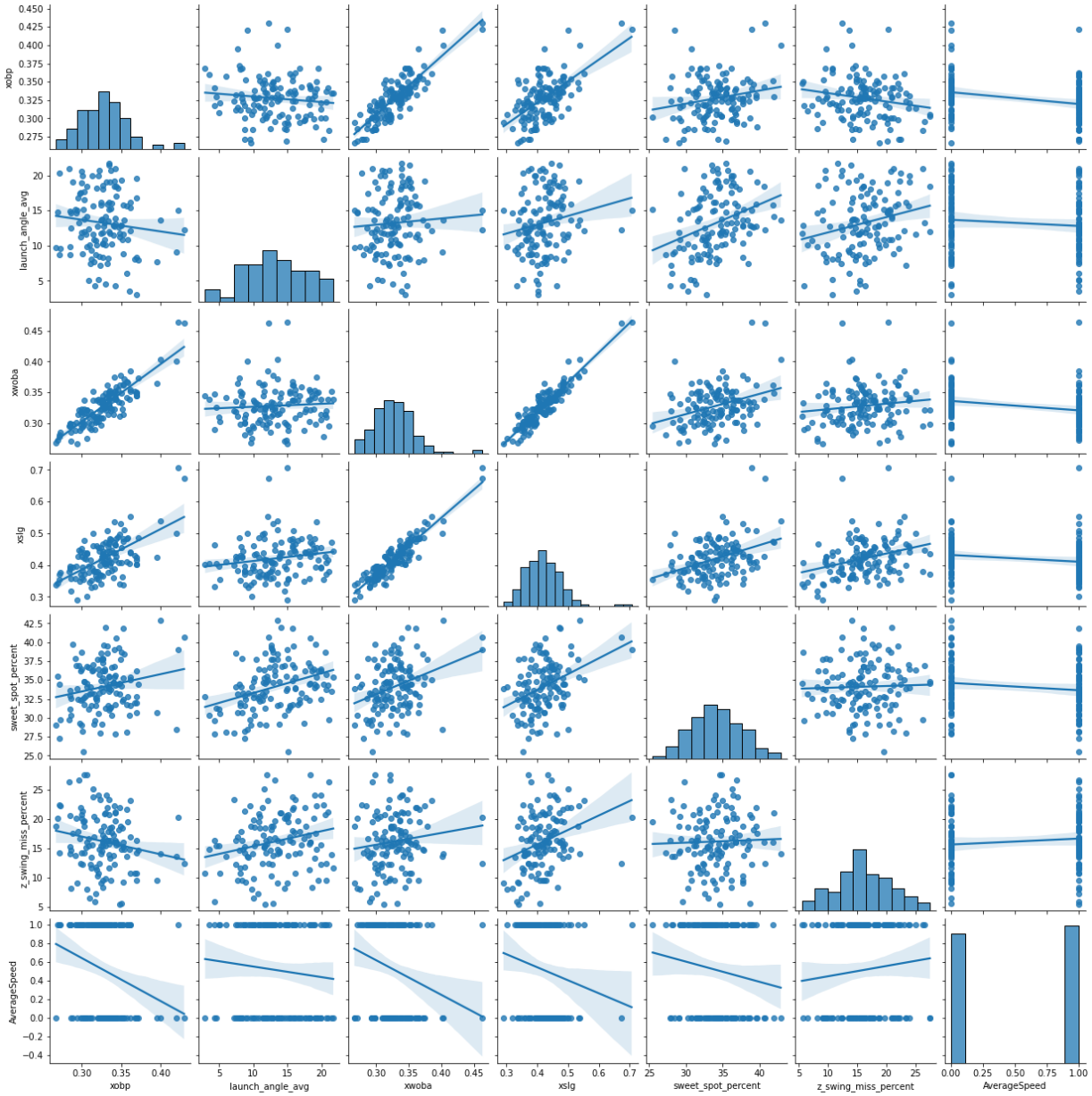




Linearities and transformation

In [27]:
```
sns.pairplot(ba, vars=['xobp', 'launch_angle_avg', 'xwoba', 'xslg', 'sweet_spot_p
```

Out[27]: <seaborn.axisgrid.PairGrid at 0x2dc991b8670>



In [5]:

```
xobp = smf.ols(formula='xba ~ xobp', data=ba)
xobp_fit = xobp.fit()
name = ["t-stat", "p-value"]
test = sms.linear_harvey_collier(xobp_fit)
print("Linearity Test Results:",['xobp'])
print(list(zip(name, test)))
print("\n")

LAA = smf.ols(formula='xba ~ launch_angle_avg', data=ba)
LAA_fit = LAA.fit()
name = ["t-stat", "p-value"]
test = sms.linear_harvey_collier(LAA_fit)
print("Linearity Test Results:",['launch_angle_avg'])
print(list(zip(name, test)))
print("\n")

xwoba = smf.ols(formula='xba ~ xwoba', data=ba)
xwoba_fit = xwoba.fit()
name = ["t-stat", "p-value"]
test = sms.linear_harvey_collier(xwoba_fit)
print("Linearity Test Results:",['xwoba'])
print(list(zip(name, test)))
print("\n")

xslg = smf.ols(formula='xba ~ xslg', data=ba)
xslg_fit = xslg.fit()
name = ["t-stat", "p-value"]
test = sms.linear_harvey_collier(xslg_fit)
print("Linearity Test Results:",['xslg'])
print(list(zip(name, test)))
print("\n")

AS = smf.ols(formula='xba ~ AverageSpeed', data=ba).fit()
name = ["t-stat", "p-value"]
test = sms.linear_harvey_collier(AS)
print("Linearity Test Results:",['AverageSpeed'])
print(list(zip(name, test)))
print("\n")

SSP = smf.ols(formula='xba ~ sweet_spot_percent', data=ba).fit()
name = ["t-stat", "p-value"]
test = sms.linear_harvey_collier(SSP)
print("Linearity Test Results:",['sweet_spot_percent'])
print(list(zip(name, test)))
print("\n")

zsmp = smf.ols(formula='xba ~ z_swing_miss_percent', data=ba).fit()
name = ["t-stat", "p-value"]
test = sms.linear_harvey_collier(zsmp)
print("Linearity Test Results:",['z_swing_miss_percent'])
print(list(zip(name, test)))
print("\n")
```

```
Linearity Test Results: ['xobp']
[('t-stat', 0.543447871655758), ('p-value', 0.5877816675040985)]
```

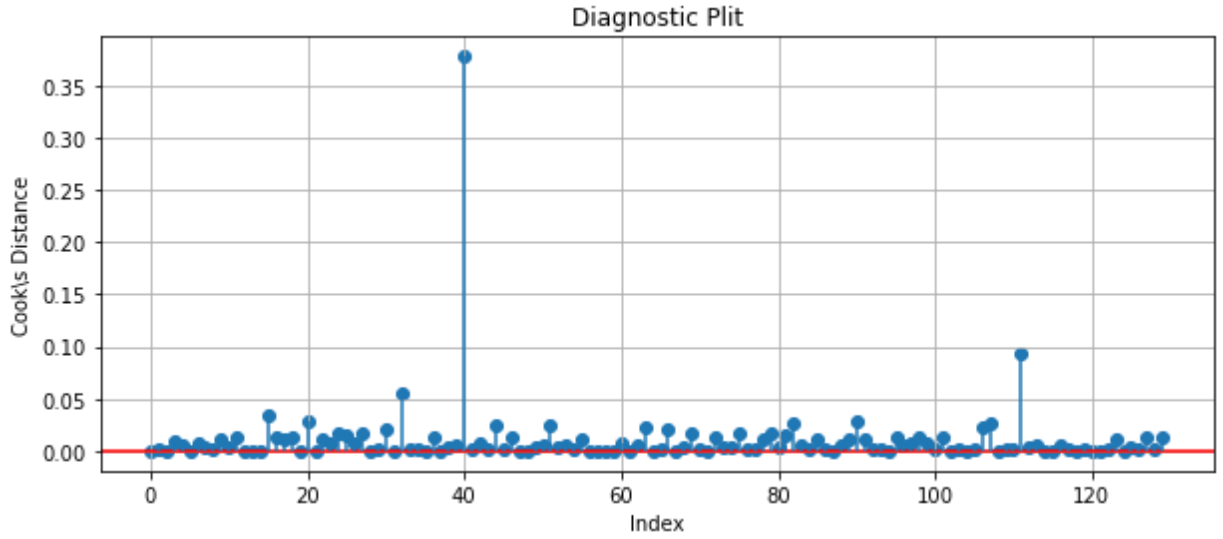
```
Linearity Test Results: ['launch_angle_avg']
[('t-stat', 0.16873426292706833), ('p-value', 0.8662762365630223)]


Linearity Test Results: ['xwoba']
[('t-stat', 0.36100351698937394), ('p-value', 0.7187013815466339)]


Linearity Test Results: ['xslg']
[('t-stat', 0.29674058670392006), ('p-value', 0.7671531656464156)]


Linearity Test Results: ['AverageSpeed']
[('t-stat', 0.7464675672965022), ('p-value', 0.45677555194591446)]


Linearity Test Results: ['sweet_spot_percent']
[('t-stat', 1.1675456782496039), ('p-value', 0.24519474038930197)]


Linearity Test Results: ['z_swing_miss_percent']
[('t-stat', 0.5686250825245491), ('p-value', 0.5706226934008758)]
```

As all predictors inside the regression fail to reject the null of the Harvey-Collier test, we can assume all present predictors exhibit linearity. If there were predictors that rejected the null of the Harvey-Collier test, we'd use the component+residual/CERES plots to visualize the nonlinear variables and the Box-Tidwell method to linearize each variable.

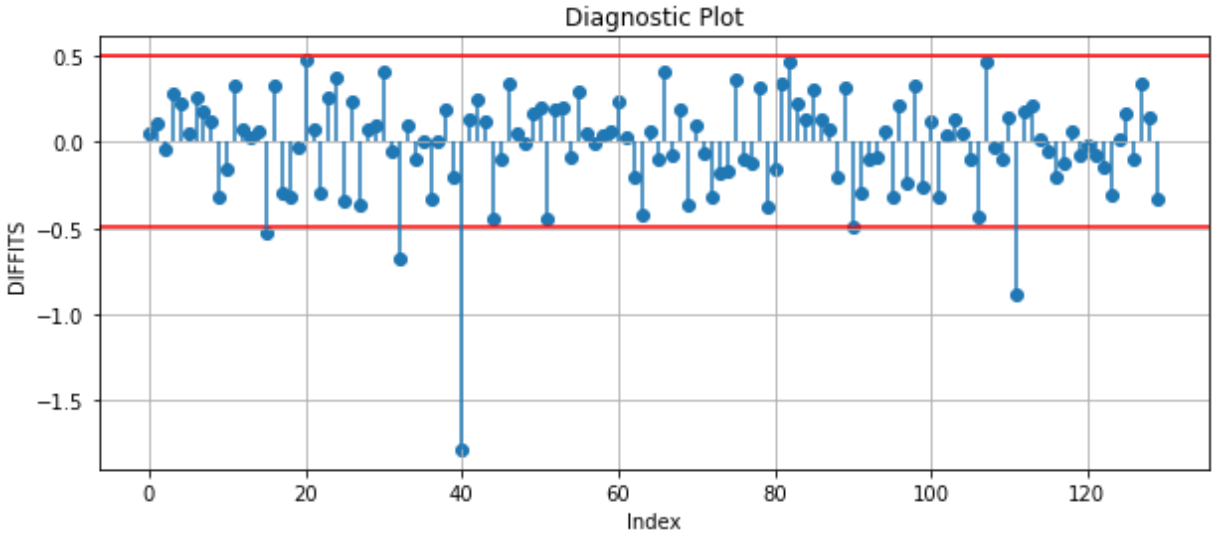
The regression results would not be interpretable if the variables are non-linear. The Regression expects a linear relationship between the predictors and the dependent variable and including non-linear variables that haven't been transformed will skew and distort the results of your regression.

Outliers

In [114]:
```
cooks_distance = mr_fit.get_influence().cooks_distance

plt.figure(figsize = (10,4))
plt.scatter(ba.index, cooks_distance[0])
plt.axhline(0, color = 'red')
plt.vlines(x = ba.index, ymin = 0, ymax = cooks_distance[0])
plt.xlabel('Index')
plt.ylabel('Cook\s Distance')
plt.title('Diagnostic Plit')
plt.grid()
```



In [5]:
```
dffits, threshold=mr_fit.get_influence().dffits

plt.figure(figsize=(10,4))
plt.scatter(ba.index,dffits)
plt.axhline(threshold,color='red')
plt.axhline(-threshold,color='red')
plt.vlines(x=ba.index,ymin=0,ymax=dffits)
plt.xlabel('Index')
plt.ylabel('DIFFITS')
plt.title('Diagnostic Plot')
plt.grid()
```



In [6]:
```
bav = ba[np.abs(dffits)<threshold]
```

In [9]:
```
bav.head()
```

Out[9]:

	last_name	first_name	player_id	year	xba	xslg	xwoba	xobp	xiso	exit_velocity_avg	...
0	Cruz Jr.	Nelson	443558	2022	0.241	0.399	0.320	0.323	0.157	90.9	...
1	Blackmon	Charlie	453568	2022	0.256	0.376	0.301	0.308	0.120	86.2	...
2	McCutchen	Andrew	457705	2022	0.252	0.406	0.325	0.329	0.155	89.1	...
3	Turner	Justin	457759	2022	0.267	0.427	0.339	0.343	0.160	89.5	...
4	Andrus	Elvis	462101	2022	0.245	0.360	0.293	0.300	0.114	87.9	...

5 rows × 22 columns

In [7]:
```
print("The wage dataset with outliers counted", ba.shape[0], "observations. The r
print("Therefore, the suspected observations were", ba.shape[0]-bav.shape[0])
```

```
The wage dataset with outliers counted 130 observations. The new database count
s 126 observations.
Therefore, the significant observations were 4
```

In [7]:
```
regr = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spot
result = regr.fit()
result.summary()
```

Out[7]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.825
Model:	OLS	**Adj. R-squared:**	0.815
Method:	Least Squares	**F-statistic:**	79.55
Date:	Mon, 14 Nov 2022	**Prob (F-statistic):**	1.12e-41
Time:	20:37:14	**Log-Likelihood:**	420.50
No. Observations:	126	**AIC:**	-825.0
Df Residuals:	118	**BIC:**	-802.3
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.1262	0.015	8.517	0.000	0.097	0.156
xobp	0.8189	0.221	3.708	0.000	0.382	1.256
launch_angle_avg	-0.0021	0.000	-10.083	0.000	-0.002	-0.002
xwoba	-1.1826	0.411	-2.879	0.005	-1.996	-0.369
xslg	0.5714	0.136	4.211	0.000	0.303	0.840
sweet_spot_percent	0.0016	0.000	5.753	0.000	0.001	0.002
z_swing_miss_percent	-0.0012	0.000	-5.460	0.000	-0.002	-0.001
AverageSpeed	0.0019	0.002	1.145	0.255	-0.001	0.005

Omnibus:	3.662	**Durbin-Watson:**	2.170
Prob(Omnibus):	0.160	**Jarque-Bera (JB):**	2.225
Skew:	-0.072	**Prob(JB):**	0.329
Kurtosis:	2.365	**Cond. No.**	2.46e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.46e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In [10]:
```
print("R^2 before removing the observations was:", mr_fit.rsquared)
print("R^2 after removing the observations is:", result.rsquared)
if result.rsquared > mr_fit.rsquared:
    print("The observations were influential")
else:
    print("The observations were not influential")
```

```
R^2 before removing the observations was: 0.8145110252354499
R^2 after removing the observations is: 0.8251419135997036
The observations were influential
```

MODEL BUIDLING

Models selection

In [45]:
```
subdat = bav[['xba', 'xobp', 'launch_angle_avg', 'xwoba', 'xslg', 'sweet_spot_per
```

In [46]:
```
import itertools

regr = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spot
result = regr.fit()
y = bav['xba']
y_pred=result.fittedvalues


storage_cp = pd.DataFrame(columns = ["Variables", "CP"])
k = 8

for L in range(1, len(subdat.columns[1:]) + 1):
    for subset in itertools.combinations(subdat.columns[1:], L):

        formula1 = 'xba~'+'+'.join(subset)

        result = smf.ols(formula=formula1, data=bav).fit()
        y_sub = result.fittedvalues
        p = len(subset)+1

        cp = mallow.mallow(y, y_pred,y_sub, k, p)

        storage_cp = storage_cp.append({'Variables': subset, 'CP': cp}, ignore_ir
```

In [47]:
```
storage_cp.sort_values(by = "CP")
```

Out[47]:

	Variables	CP
119	(xobp, launch_angle_avg, xwoba, xslg, sweet_sp...	7.310144
126	(xobp, launch_angle_avg, xwoba, xslg, sweet_sp...	8.0
104	(xobp, launch_angle_avg, xslg, sweet_spot_perc...	13.313511
123	(xobp, launch_angle_avg, xslg, sweet_spot_perc...	14.289172
113	(launch_angle_avg, xwoba, xslg, sweet_spot_per...	18.51479
...	...	...
1	(launch_angle_avg,)	460.554843
4	(sweet_spot_percent,)	466.895103
27	(z_swing_miss_percent, AverageSpeed)	474.997107
5	(z_swing_miss_percent,)	490.058095
6	(AverageSpeed,)	526.661796

127 rows × 2 columns

In [48]:
```
model = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spo
results = model.fit()
y = bav.xba
y_pred=results.fittedvalues

mr_sub = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_sp
mr_sub_fit = mr_sub.fit()
y_sub=mr_sub_fit.fittedvalues

k = 8
p = 8

mallow.mallow(y, y_pred,y_sub, k, p)
```

Out[48]: 8.0

In [49]:
```
model = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spo
results = model.fit()
y = bav.xba
y_pred=results.fittedvalues

mr_sub = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_sp
mr_sub_fit = mr_sub.fit()
y_sub=mr_sub_fit.fittedvalues

k = 8
p = 7

mallow.mallow(y, y_pred,y_sub, k, p)
```

Out[49]: 7.310143591915278

In [50]:
```
model = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spo
results = model.fit()
y = bav.xba
y_pred=results.fittedvalues

mr_sub = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot_perce
mr_sub_fit = mr_sub.fit()
y_sub=mr_sub_fit.fittedvalues

k = 8
p = 6

mallow.mallow(y, y_pred,y_sub, k, p)
```

Out[50]: 13.313510634845656

In [51]:
```
model = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spo
results = model.fit()
y = bav.xba
y_pred=results.fittedvalues

mr_sub = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot_perce
mr_sub_fit = mr_sub.fit()
y_sub=mr_sub_fit.fittedvalues

k = 8
p = 7

mallow.mallow(y, y_pred,y_sub, k, p)
```

Out[51]: 14.289172303442626

Regressions

In [8]:
```
# MODEL 1 BENCHMARK: include all the variables
result_bm = result
result_bm.summary()
```

Out[8]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.825
Model:	OLS	**Adj. R-squared:**	0.815
Method:	Least Squares	**F-statistic:**	79.55
Date:	Mon, 14 Nov 2022	**Prob (F-statistic):**	1.12e-41
Time:	20:37:23	**Log-Likelihood:**	420.50
No. Observations:	126	**AIC:**	-825.0
Df Residuals:	118	**BIC:**	-802.3
Df Model:	7		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	0.1262	0.015	8.517	0.000	0.097	0.156

In [9]:
```
# MODEL 2: Exclude AverageSpeed
result2 = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_s
result2.summary()
```

Out[9]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.823
Model:	OLS	**Adj. R-squared:**	0.814
Method:	Least Squares	**F-statistic:**	92.35
Date:	Mon, 14 Nov 2022	**Prob (F-statistic):**	2.13e-42
Time:	20:37:23	**Log-Likelihood:**	419.80
No. Observations:	126	**AIC:**	-825.6
Df Residuals:	119	**BIC:**	-805.8
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.1304	0.014	9.063	0.000	0.102	0.159
xobp	0.8003	0.221	3.629	0.000	0.364	1.237
launch_angle_avg	-0.0021	0.000	-10.288	0.000	-0.002	-0.002
xwoba	-1.1607	0.411	-2.825	0.006	-1.974	-0.347
xslg	0.5619	0.136	4.143	0.000	0.293	0.830
sweet_spot_percent	0.0016	0.000	5.713	0.000	0.001	0.002
z_swing_miss_percent	-0.0012	0.000	-5.358	0.000	-0.002	-0.001

Omnibus:	3.469	**Durbin-Watson:**	2.171
Prob(Omnibus):	0.176	**Jarque-Bera (JB):**	2.138
Skew:	-0.062	**Prob(JB):**	0.343
Kurtosis:	2.374	**Cond. No.**	2.45e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.45e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In [10]:
```
# MODEL 3: exclude xwoba and AverageSpeed
result3 = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot_perc
result3.summary()
```

Out[10]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.811
Model:	OLS	**Adj. R-squared:**	0.803
Method:	Least Squares	**F-statistic:**	103.2
Date:	Mon, 14 Nov 2022	**Prob (F-statistic):**	9.20e-42
Time:	20:37:23	**Log-Likelihood:**	415.71
No. Observations:	126	**AIC:**	-819.4
Df Residuals:	120	**BIC:**	-802.4
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.1132	0.013	8.441	0.000	0.087	0.140
xobp	0.1884	0.043	4.402	0.000	0.104	0.273
launch_angle_avg	-0.0021	0.000	-10.025	0.000	-0.003	-0.002
xslg	0.1840	0.023	8.020	0.000	0.139	0.229
sweet_spot_percent	0.0016	0.000	5.458	0.000	0.001	0.002
z_swing_miss_percent	-0.0014	0.000	-6.534	0.000	-0.002	-0.001

Omnibus:	4.469	**Durbin-Watson:**	2.133
Prob(Omnibus):	0.107	**Jarque-Bera (JB):**	2.705
Skew:	-0.143	**Prob(JB):**	0.259
Kurtosis:	2.341	**Cond. No.**	2.31e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.31e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In [11]:
```
# MODEL 4: exclude xwoba
result4 = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot_perc
result4.summary()
```

Out[11]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.813
Model:	OLS	**Adj. R-squared:**	0.803
Method:	Least Squares	**F-statistic:**	86.15
Date:	Mon, 14 Nov 2022	**Prob (F-statistic):**	6.12e-41
Time:	20:37:23	**Log-Likelihood:**	416.22
No. Observations:	126	**AIC:**	-818.4
Df Residuals:	119	**BIC:**	-798.6
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.1092	0.014	7.800	0.000	0.081	0.137
xobp	0.1947	0.043	4.498	0.000	0.109	0.280
launch_angle_avg	-0.0021	0.000	-9.829	0.000	-0.002	-0.002
xslg	0.1861	0.023	8.075	0.000	0.140	0.232
sweet_spot_percent	0.0016	0.000	5.484	0.000	0.001	0.002
z_swing_miss_percent	-0.0015	0.000	-6.606	0.000	-0.002	-0.001
AverageSpeed	0.0017	0.002	0.982	0.328	-0.002	0.005

Omnibus:	4.631	**Durbin-Watson:**	2.127
Prob(Omnibus):	0.099	**Jarque-Bera (JB):**	2.764
Skew:	-0.144	**Prob(JB):**	0.251
Kurtosis:	2.334	**Cond. No.**	2.33e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.33e+03. This might indicate that there are strong multicollinearity or other numerical problems.

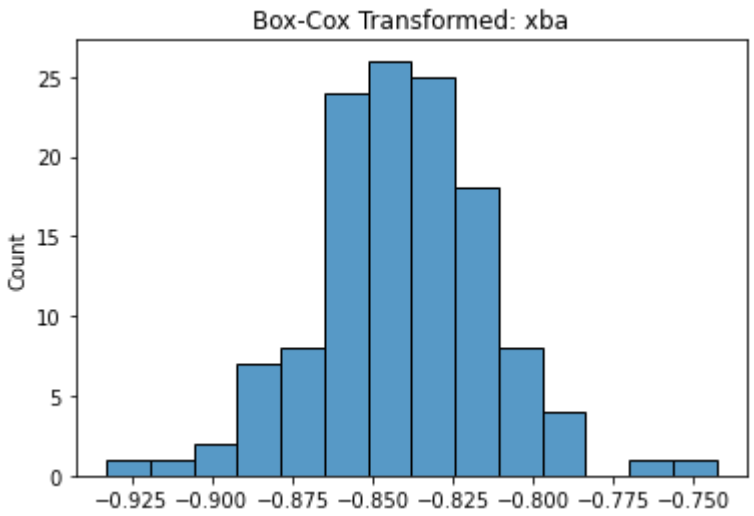
Evaluation of transformation

In [9]:
```
bacopy = bav.copy()
bacopy = bacopy[["xba", "xobp", "launch_angle_avg", "xwoba", "xslg", "sweet_spot_

for i in range(len(bacopy.columns)):
    bc_den, lambda_den = stats.boxcox(abs(bacopy.iloc[:,i]))
    print("lambda for", bacopy.columns[i], ":")
    print(lambda_den)
```

```
lambda for xba :
0.7815521917143773
lambda for xobp :
-0.24469628777823632
lambda for launch_angle_avg :
0.9924139319984278
lambda for xwoba :
-0.8021701209556577
lambda for xslg :
-0.378390915770604
lambda for sweet_spot_percent :
0.4976788691298119
lambda for z_swing_miss_percent :
0.9499318331134093
```

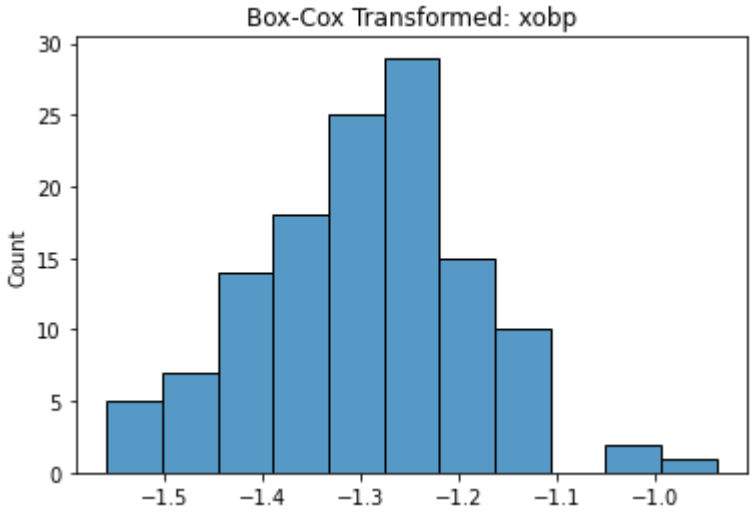
In [10]:
```
bc_xba, lambda_xba = stats.boxcox(bav["xba"])
print(lambda_xba)
sns.histplot(bc_xba)
plt.title("Box-Cox Transformed: xba")
plt.show()
```

```
0.7815521917143773
```



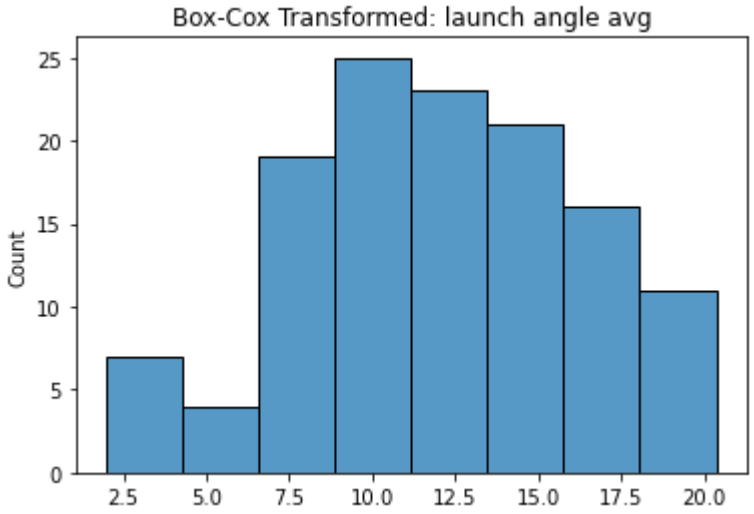
In [11]:
```
bc_xobp, lambda_xobp = stats.boxcox(bav["xobp"])
print(lambda_xobp)
sns.histplot(bc_xobp)
plt.title("Box-Cox Transformed: xobp")
plt.show()
```

```
-0.24469628777823632
```



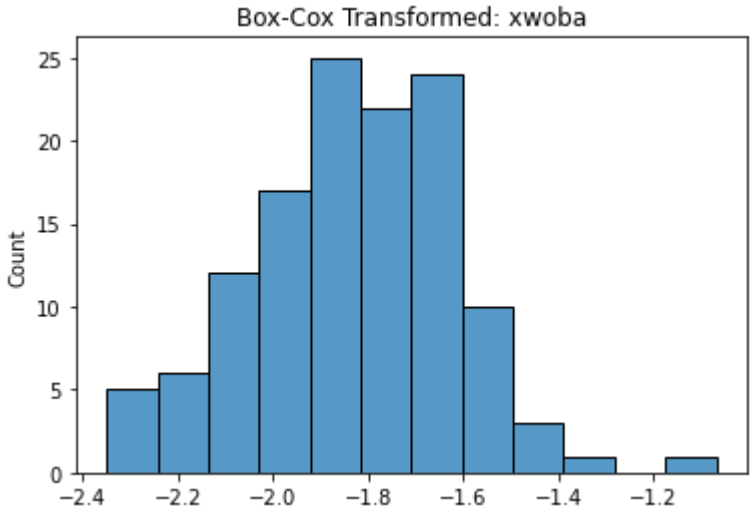
In [12]:
```
bc_launch_angle_avg, lambda_launch_angle_avg = stats.boxcox(bav["launch_angle_avg
print(lambda_launch_angle_avg)
sns.histplot(bc_launch_angle_avg)
plt.title("Box-Cox Transformed: launch angle avg")
plt.show()
```

```
0.9924139319984278
```



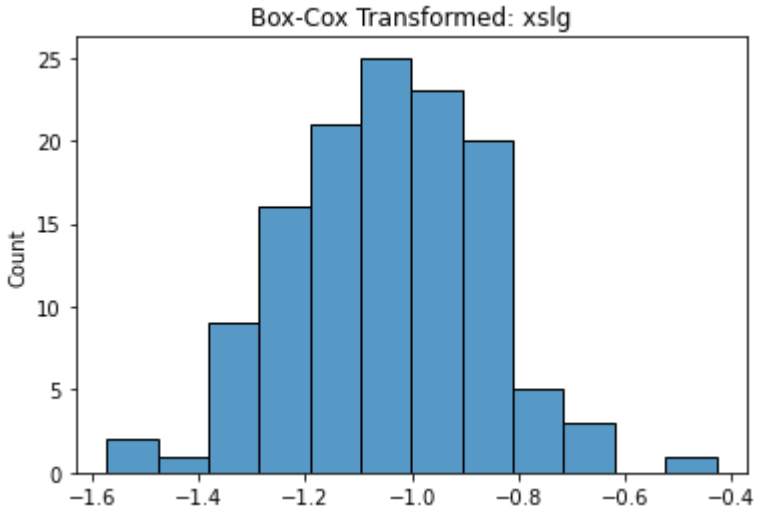
In [13]:
```
bc_xwoba, lambda_xwoba= stats.boxcox(bav["xwoba"])
print(lambda_xwoba)
sns.histplot(bc_xwoba)
plt.title("Box-Cox Transformed: xwoba")
plt.show()
```

```
-0.8021701209556577
```



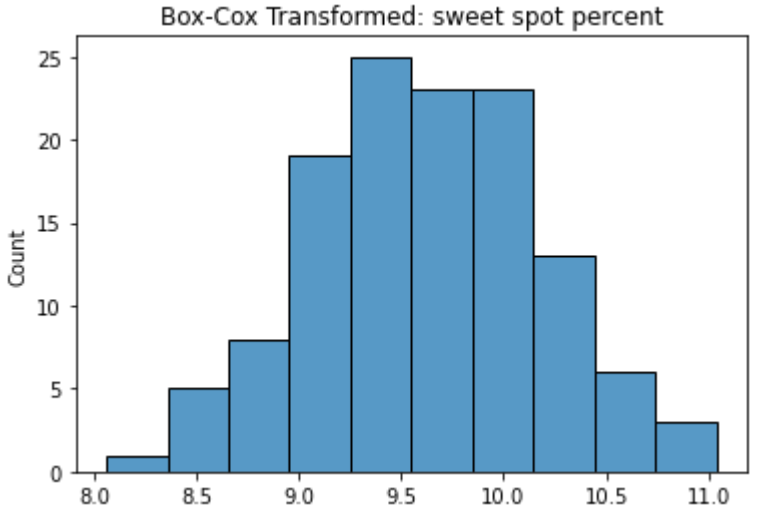
In [14]:
```
bc_xslg, lambda_xslg= stats.boxcox(bav["xslg"])
print(lambda_xslg)
sns.histplot(bc_xslg)
plt.title("Box-Cox Transformed: xslg")
plt.show()
```

```
-0.378390915770604
```



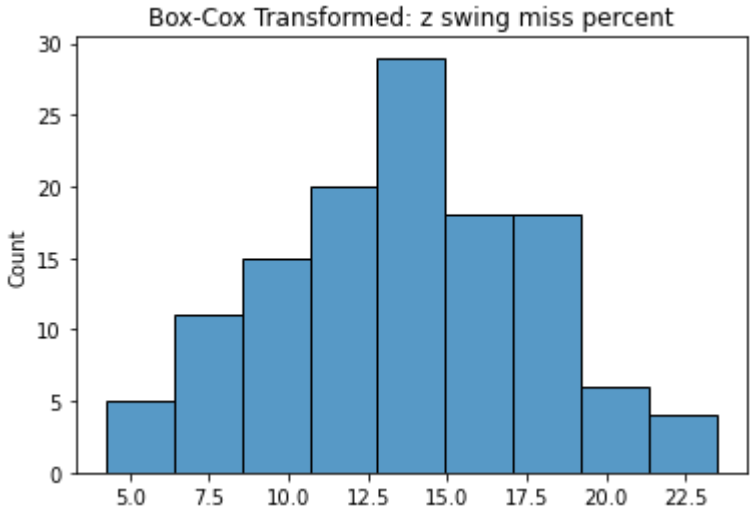
In [15]:
```
bc_sweet_spot_percent, lambda_sweet_spot_percent = stats.boxcox(bav["sweet_spot_p
print(lambda_sweet_spot_percent)
sns.histplot(bc_sweet_spot_percent)
plt.title("Box-Cox Transformed: sweet spot percent")
plt.show()
```

```
0.4976788691298119
```



In [16]:
```
bc_z_swing_miss_percent, lambda_z_swing_miss_percent = stats.boxcox(bav["z_swing_
print(lambda_z_swing_miss_percent)
sns.histplot(bc_z_swing_miss_percent)
plt.title("Box-Cox Transformed: z swing miss percent")
plt.show()
```

```
0.9499318331134093
```



In [17]:
```
ba_T = pd.DataFrame(bc_xba, columns = ["xba"])
ba_T["xobp"] = bc_xobp
ba_T["launch_angle_avg"] = bc_launch_angle_avg
ba_T["xwoba"] = bc_xwoba
ba_T["xslg"] = bc_xslg
ba_T["sweet_spot_percent"] = bc_sweet_spot_percent
ba_T["z_swing_miss_percent"] = bc_z_swing_miss_percent
ba_T["AverageSpeed"] = ba.AverageSpeed
ba_T.head()
```

Out[17]:

	xba	xobp	launch_angle_avg	xwoba	xslg	sweet_spot_percent	z_swing_miss
0	-0.858724	-1.301814	6.139555	-1.862859	-1.098746	8.541117	1
1	-0.838391	-1.364881	11.250764	-2.019349	-1.183754	10.061916	1
2	-0.843787	-1.277600	12.035441	-1.824426	-1.074204	9.979787	1
3	-0.823646	-1.223178	17.127874	-1.722266	-1.003947	10.684138	1
4	-0.853276	-1.400101	10.760147	-2.090690	-1.247238	8.745400	1

In [18]:
```
ba_T = pd.DataFrame(ba_T, columns = ['xba','xobp','launch_angle_avg', 'xwoba', 'x
ba_T.head()
```

Out[18]:

	xba	xobp	launch_angle_avg	xwoba	xslg	sweet_spot_percent	z_swing_miss
0	-0.858724	-1.301814	6.139555	-1.862859	-1.098746	8.541117	1
1	-0.838391	-1.364881	11.250764	-2.019349	-1.183754	10.061916	1
2	-0.843787	-1.277600	12.035441	-1.824426	-1.074204	9.979787	1
3	-0.823646	-1.223178	17.127874	-1.722266	-1.003947	10.684138	1
4	-0.853276	-1.400101	10.760147	-2.090690	-1.247238	8.745400	1

Mallow CP with transformed variables

We are going to evaluate whether the best 4 models we picked are going to match with the transformed-Mallow CP.

In [61]:
```
subdat = ba_T[['xba', 'xobp', 'launch_angle_avg', 'xwoba', 'xslg', 'sweet_spot_pe
```

In [62]:
```
regr = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spot
result = regr.fit()
y = ba_T['xba']
y_pred=result.fittedvalues


storage_cp = pd.DataFrame(columns = ["Variables", "CP"])
k = 8

for L in range(1, len(subdat.columns[1:]) + 1):
    for subset in itertools.combinations(subdat.columns[1:], L):

        formula1 = 'xba~'+'+'.join(subset)

        result = smf.ols(formula=formula1, data=ba_T).fit()
        y_sub = result.fittedvalues
        p = len(subset)+1

        cp = mallow.mallow(y, y_pred,y_sub, k, p)

        storage_cp = storage_cp.append({'Variables': subset, 'CP': cp}, ignore_ir
```

In [63]:
```
storage_cp.sort_values(by = "CP").head()
```

Out[63]:

	Variables	CP
119	(xobp, launch_angle_avg, xwoba, xslg, sweet_sp...	6.00594
126	(xobp, launch_angle_avg, xwoba, xslg, sweet_sp...	8.0
104	(xobp, launch_angle_avg, xslg, sweet_spot_perc...	14.394311
123	(xobp, launch_angle_avg, xslg, sweet_spot_perc...	16.179035
113	(launch_angle_avg, xwoba, xslg, sweet_spot_per...	20.802362

In [64]:
```
model = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spo
results = model.fit()
y = ba_T.xba
y_pred=results.fittedvalues

mr_sub = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_sp
mr_sub_fit = mr_sub.fit()
y_sub=mr_sub_fit.fittedvalues

k = 8
p = 8

mallow.mallow(y, y_pred,y_sub, k, p)
```

Out[64]: 8.0

In [65]:
```
model = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spo
results = model.fit()
y = ba_T.xba
y_pred=results.fittedvalues

mr_sub = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_sp
mr_sub_fit = mr_sub.fit()
y_sub=mr_sub_fit.fittedvalues

k = 8
p = 7

mallow.mallow(y, y_pred,y_sub, k, p)
```

Out[65]: 6.005940233404857

In [66]:
```
model = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spo
results = model.fit()
y = ba_T.xba
y_pred=results.fittedvalues

mr_sub = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot_perce
mr_sub_fit = mr_sub.fit()
y_sub=mr_sub_fit.fittedvalues

k = 8
p = 6

mallow.mallow(y, y_pred,y_sub, k, p)
```

Out[66]: 14.394311319733845

In [67]:
```
model = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spo
results = model.fit()
y = ba_T.xba
y_pred=results.fittedvalues

mr_sub = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot_perce
mr_sub_fit = mr_sub.fit()
y_sub=mr_sub_fit.fittedvalues

k = 8
p = 7

mallow.mallow(y, y_pred,y_sub, k, p)
```

Out[67]: 16.17903477925239

We confirm the choice of our models.

In [69]:
```
result_bm_T = smf.ols(formula='xba ~ xobp + launch_angle_avg + + xwoba + xslg + s
result_bm_T.summary()
```

Out[69]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.828
Model:	OLS	**Adj. R-squared:**	0.817
Method:	Least Squares	**F-statistic:**	80.92
Date:	Sun, 13 Nov 2022	**Prob (F-statistic):**	4.90e-42
Time:	16:15:26	**Log-Likelihood:**	383.64
No. Observations:	126	**AIC:**	-751.3
Df Residuals:	118	**BIC:**	-728.6
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.6925	0.030	-23.454	0.000	-0.751	-0.634
xobp	0.2649	0.065	4.054	0.000	0.135	0.394
launch_angle_avg	-0.0029	0.000	-10.411	0.000	-0.003	-0.002
xwoba	-0.2019	0.063	-3.190	0.002	-0.327	-0.077
xslg	0.2238	0.046	4.823	0.000	0.132	0.316
sweet_spot_percent	0.0127	0.002	5.831	0.000	0.008	0.017
z_swing_miss_percent	-0.0020	0.000	-5.653	0.000	-0.003	-0.001
AverageSpeed	-0.0002	0.002	-0.077	0.939	-0.004	0.004

Omnibus:	3.373	**Durbin-Watson:**	2.126
Prob(Omnibus):	0.185	**Jarque-Bera (JB):**	2.143
Skew:	-0.085	**Prob(JB):**	0.342
Kurtosis:	2.384	**Cond. No.**	2.03e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.03e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In [71]:
```
result2_T = smf.ols(formula='xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet
result2_T.summary()
```

Out[71]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.828
Model:	OLS	**Adj. R-squared:**	0.819
Method:	Least Squares	**F-statistic:**	95.21
Date:	Sun, 13 Nov 2022	**Prob (F-statistic):**	4.82e-43
Time:	16:15:59	**Log-Likelihood:**	383.63
No. Observations:	126	**AIC:**	-753.3
Df Residuals:	119	**BIC:**	-733.4
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.6924	0.029	-23.590	0.000	-0.751	-0.634
xobp	0.2655	0.065	4.116	0.000	0.138	0.393
launch_angle_avg	-0.0029	0.000	-10.499	0.000	-0.003	-0.002
xwoba	-0.2025	0.063	-3.237	0.002	-0.326	-0.079
xslg	0.2242	0.046	4.885	0.000	0.133	0.315
sweet_spot_percent	0.0127	0.002	5.858	0.000	0.008	0.017
z_swing_miss_percent	-0.0020	0.000	-5.696	0.000	-0.003	-0.001

Omnibus:	3.416	**Durbin-Watson:**	2.124
Prob(Omnibus):	0.181	**Jarque-Bera (JB):**	2.155
Skew:	-0.082	**Prob(JB):**	0.341
Kurtosis:	2.381	**Cond. No.**	2.02e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.02e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In [72]:
```
result3_T = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot_pe
result3_T.summary()
```

Out[72]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.812
Model:	OLS	**Adj. R-squared:**	0.805
Method:	Least Squares	**F-statistic:**	103.9
Date:	Sun, 13 Nov 2022	**Prob (F-statistic):**	6.54e-42
Time:	16:16:29	**Log-Likelihood:**	378.32
No. Observations:	126	**AIC:**	-744.6
Df Residuals:	120	**BIC:**	-727.6
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.7315	0.028	-26.337	0.000	-0.787	-0.677
xobp	0.0614	0.014	4.342	0.000	0.033	0.089
launch_angle_avg	-0.0029	0.000	-10.186	0.000	-0.003	-0.002
xslg	0.0786	0.009	8.337	0.000	0.060	0.097
sweet_spot_percent	0.0125	0.002	5.525	0.000	0.008	0.017
z_swing_miss_percent	-0.0024	0.000	-6.887	0.000	-0.003	-0.002

Omnibus:	5.629	**Durbin-Watson:**	2.151
Prob(Omnibus):	0.060	**Jarque-Bera (JB):**	3.406
Skew:	-0.210	**Prob(JB):**	0.182
Kurtosis:	2.312	**Cond. No.**	545.

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

In [73]:
```
result4_T = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot_pe
result4_T.summary()
```

Out[73]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.813
Model:	OLS	**Adj. R-squared:**	0.803
Method:	Least Squares	**F-statistic:**	86.08
Date:	Sun, 13 Nov 2022	**Prob (F-statistic):**	6.37e-41
Time:	16:16:57	**Log-Likelihood:**	378.42
No. Observations:	126	**AIC:**	-742.8
Df Residuals:	119	**BIC:**	-723.0
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	-0.7317	0.028	-26.252	0.000	-0.787	-0.676
xobp	0.0610	0.014	4.295	0.000	0.033	0.089
launch_angle_avg	-0.0029	0.000	-10.148	0.000	-0.003	-0.002
xslg	0.0786	0.009	8.314	0.000	0.060	0.097
sweet_spot_percent	0.0125	0.002	5.522	0.000	0.008	0.017
z_swing_miss_percent	-0.0024	0.000	-6.878	0.000	-0.003	-0.002
AverageSpeed	-0.0010	0.002	-0.447	0.656	-0.005	0.003

Omnibus:	5.172	**Durbin-Watson:**	2.160
Prob(Omnibus):	0.075	**Jarque-Bera (JB):**	3.300
Skew:	-0.217	**Prob(JB):**	0.192
Kurtosis:	2.337	**Cond. No.**	545.

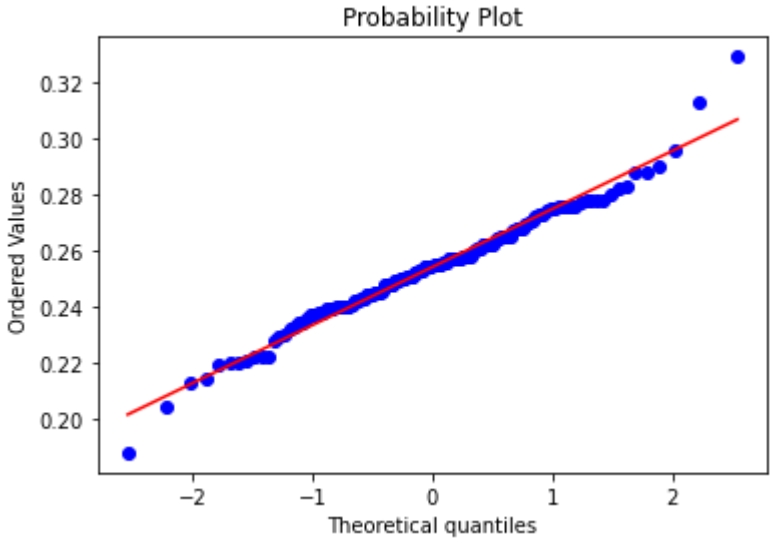
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

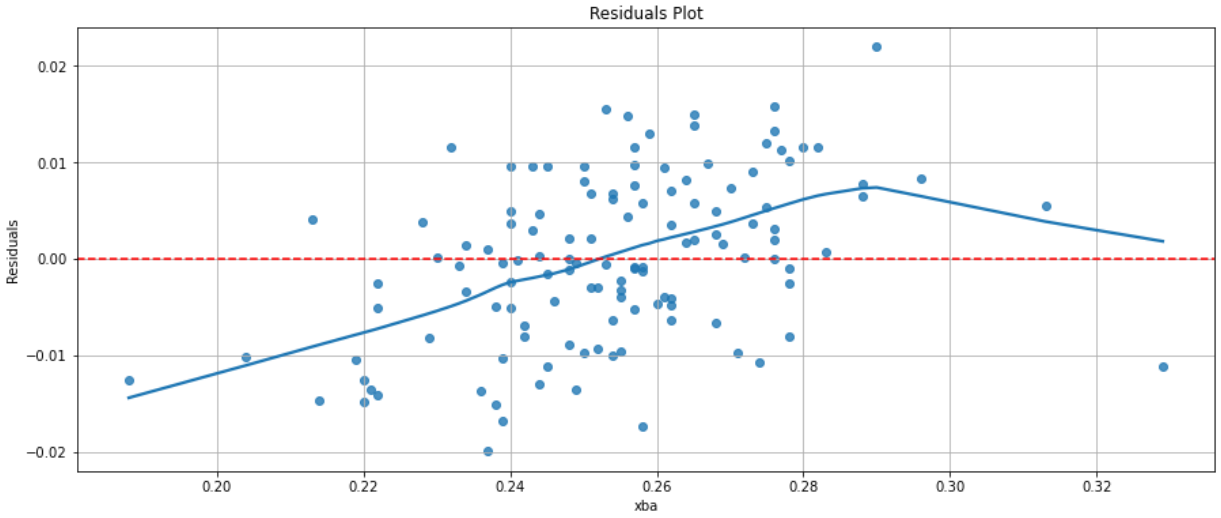
Transformations, in this, case would not improve our models accuracy and interpretability. Moreover, we would like to preserve the interpretation of our variables. Therefore, we will not used the transformed models.

Influential observations plot

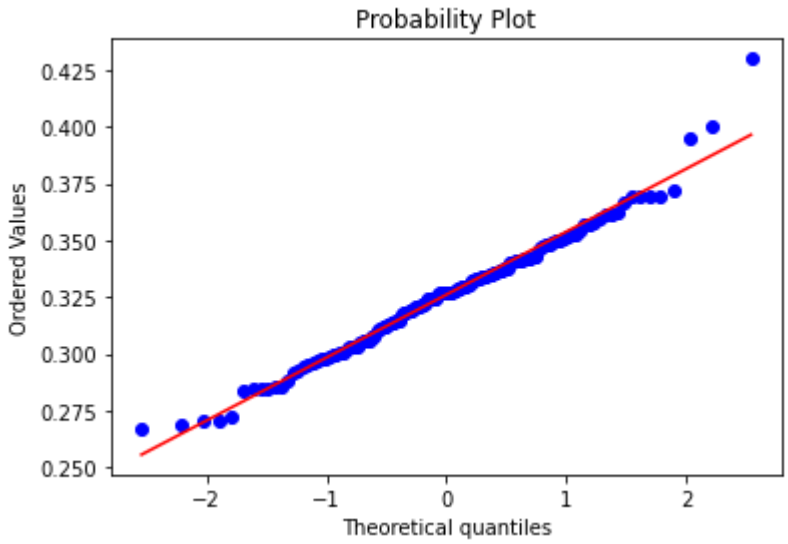
Model 1

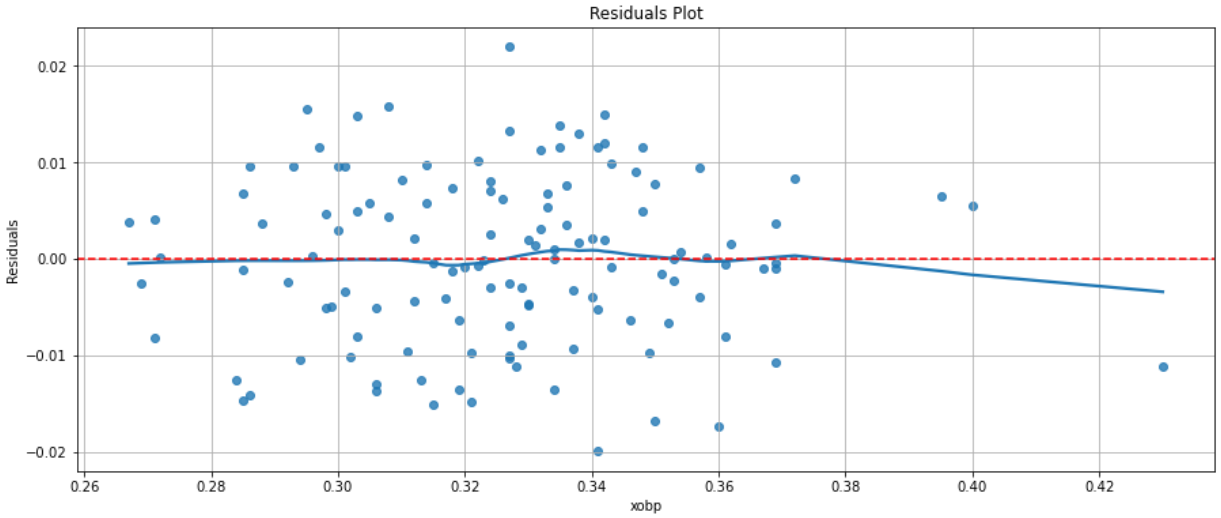
In [119]:
```
#xba
stats.probplot(bav.xba, dist = "norm", plot = plt)
plt.show()

plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xba, y = result_bm.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xba")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



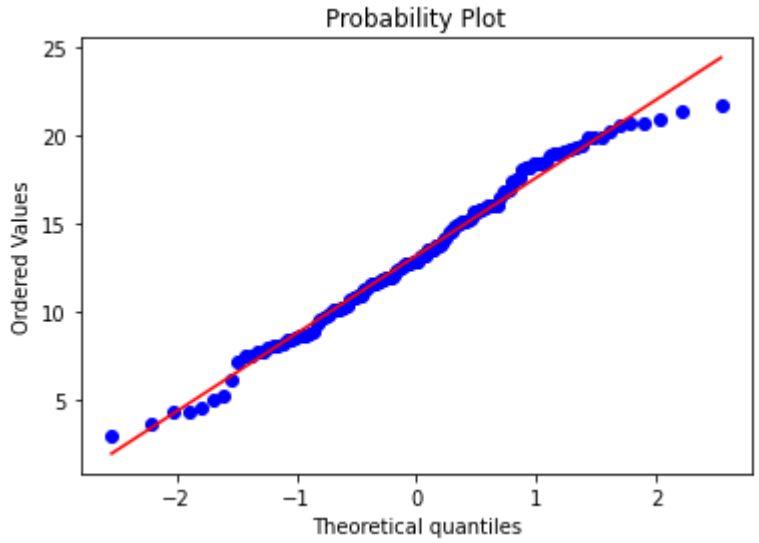


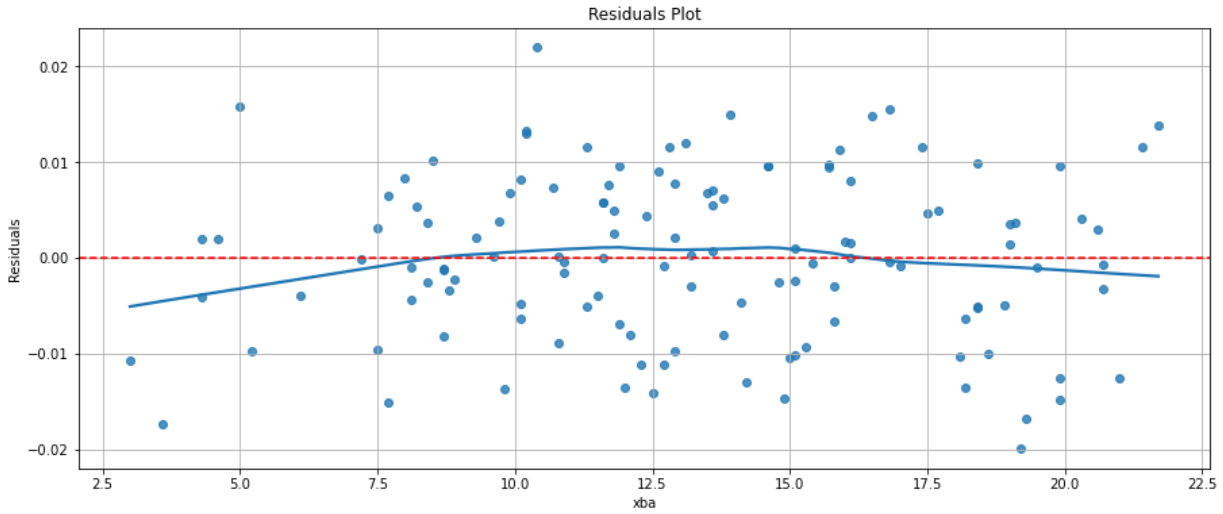
In [118]:
```
#xobp
stats.probplot(bav.xobp, dist = "norm", plot = plt)
plt.show()

plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xobp, y = result_bm.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xobp")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



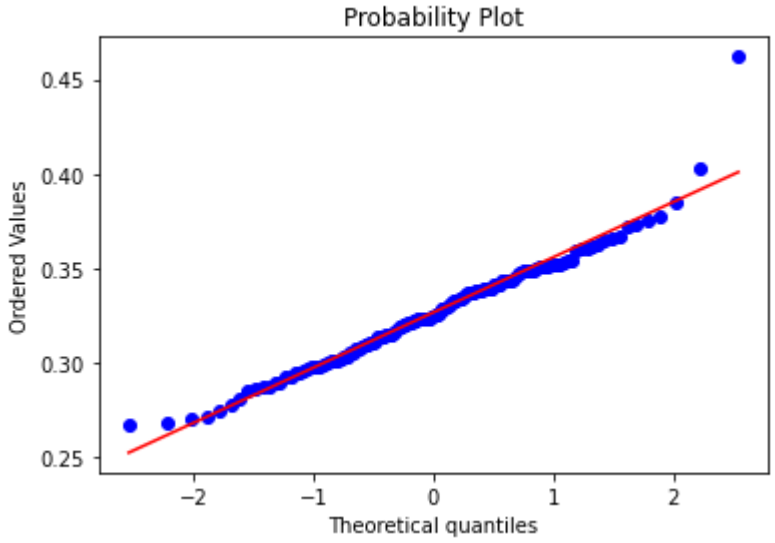


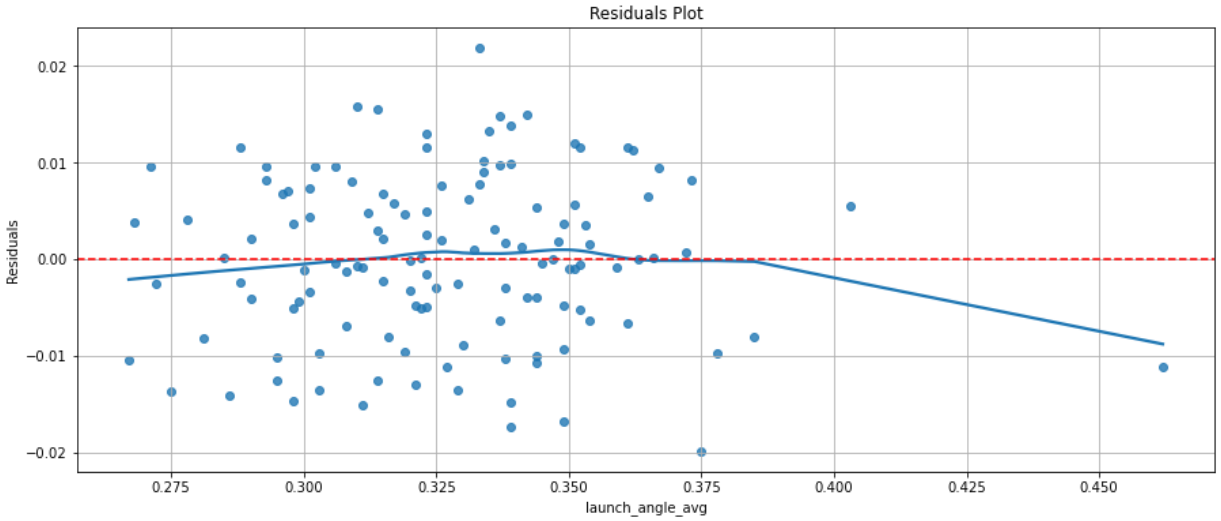
In [120]:
```
#launch_angle_avg
stats.probplot(bav.launch_angle_avg, dist = "norm", plot = plt)
plt.show()

plt.figure(figsize = (15, 6))
sns.regplot(x = bav.launch_angle_avg, y = result_bm.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("launch_angle_avg")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



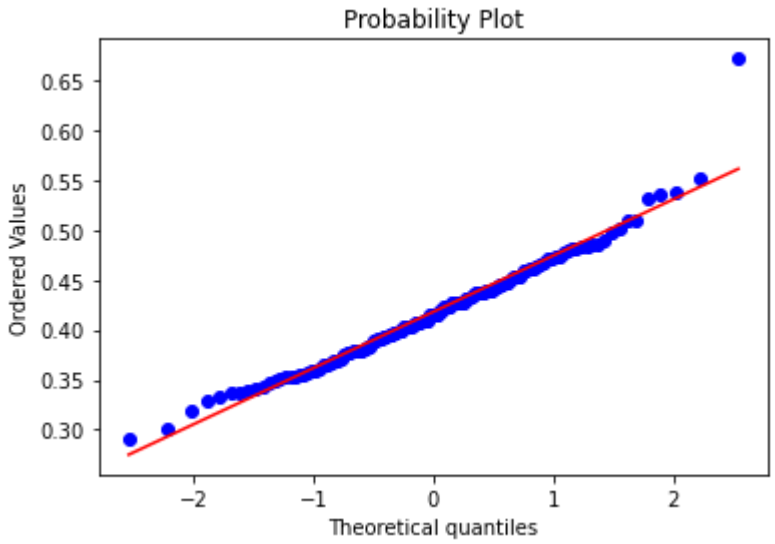


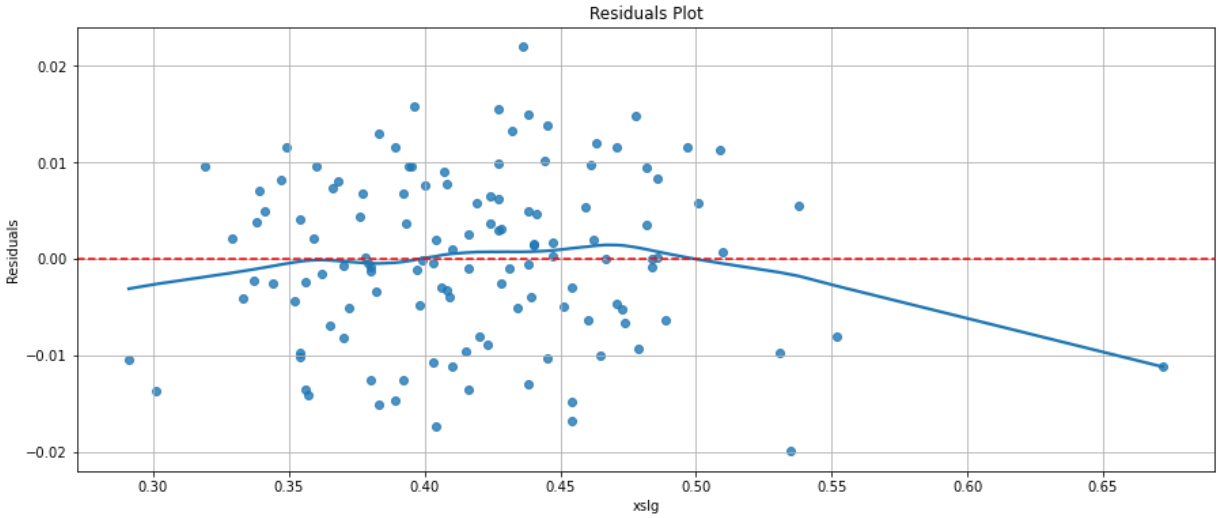
In [17]:
```
#xwoba
stats.probplot(bav.xwoba, dist = "norm", plot = plt)
plt.show()

plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xwoba, y = result_bm.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("launch_angle_avg")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



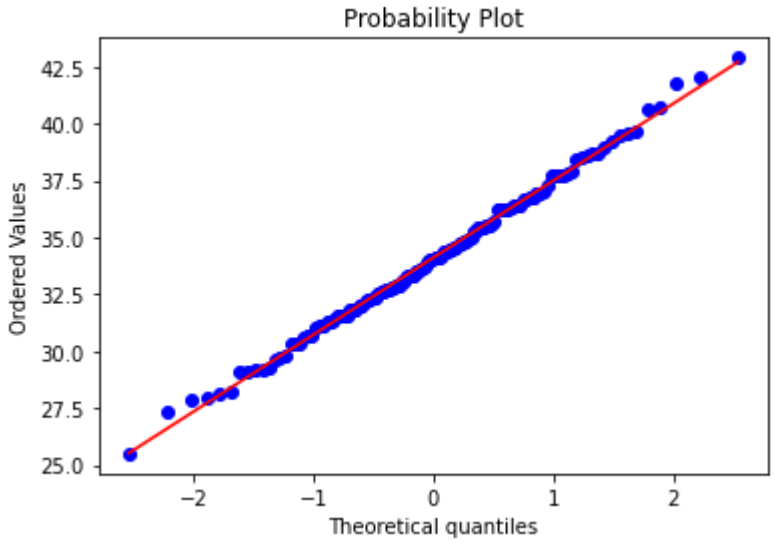


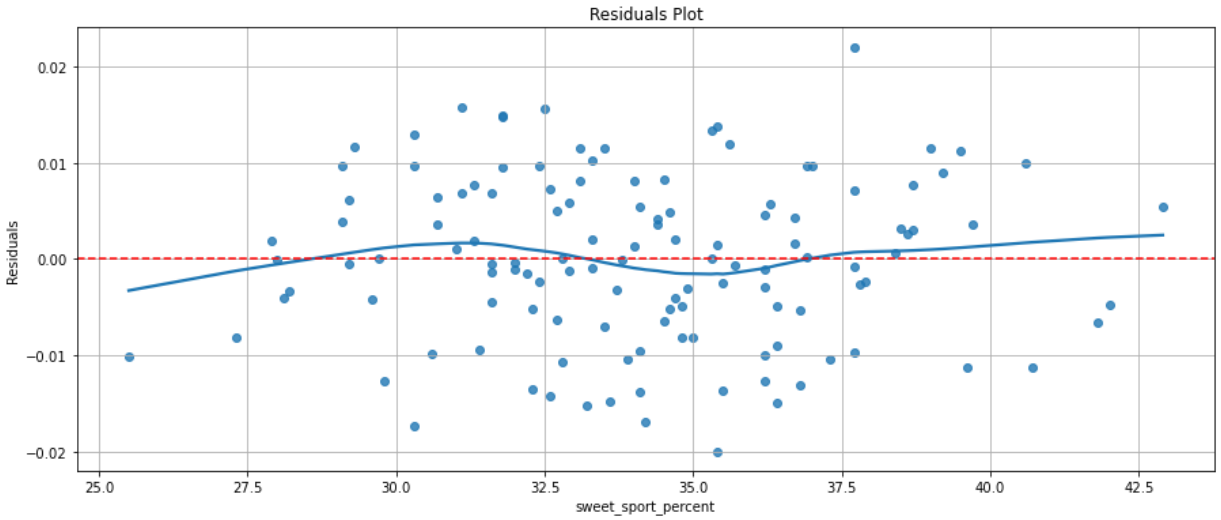
In [122]:
```
#xslg
stats.probplot(bav.xslg, dist = "norm", plot = plt)
plt.show()

plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xslg, y = result_bm.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xslg")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



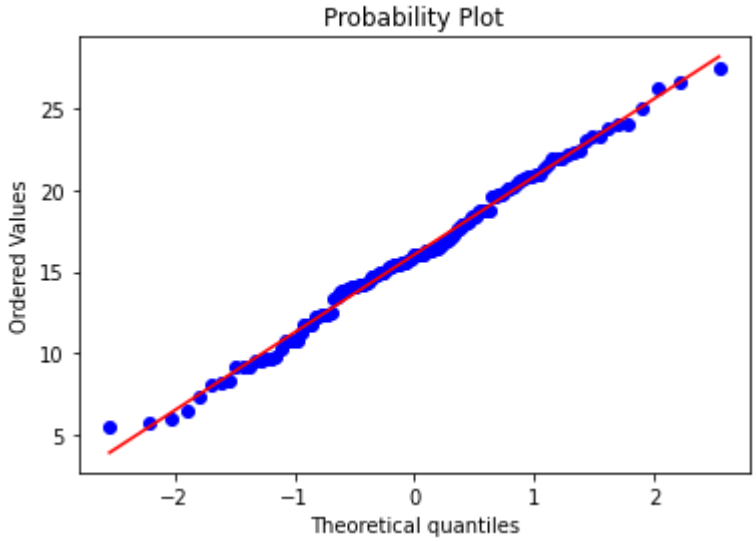


In [123]:
```
#sweet_spot_percent
stats.probplot(bav.sweet_spot_percent, dist = "norm", plot = plt)
plt.show()

plt.figure(figsize = (15, 6))
sns.regplot(x = bav.sweet_spot_percent, y = result_bm.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("sweet_sport_percent")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```

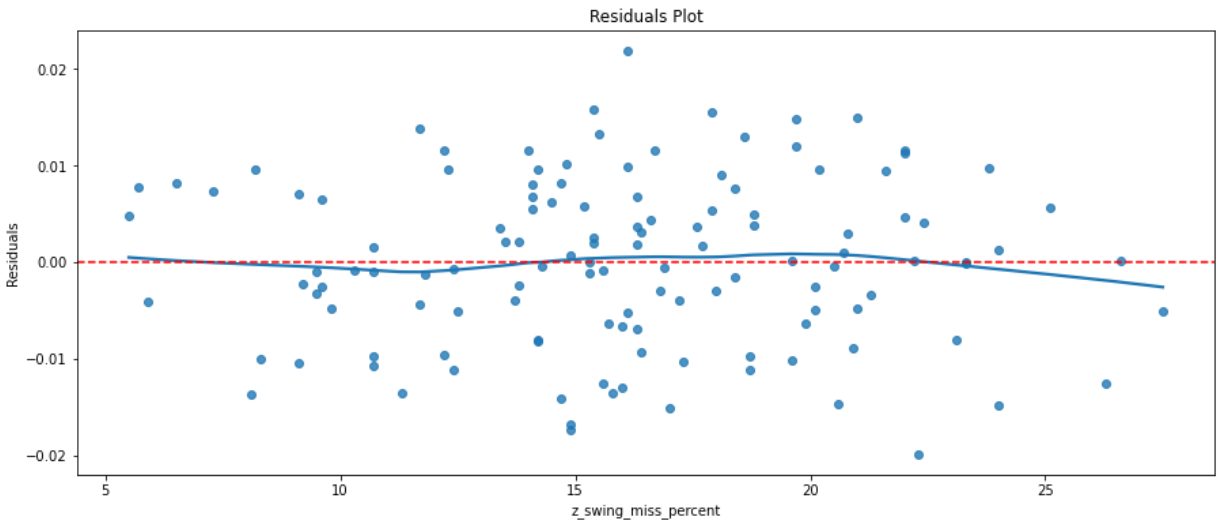




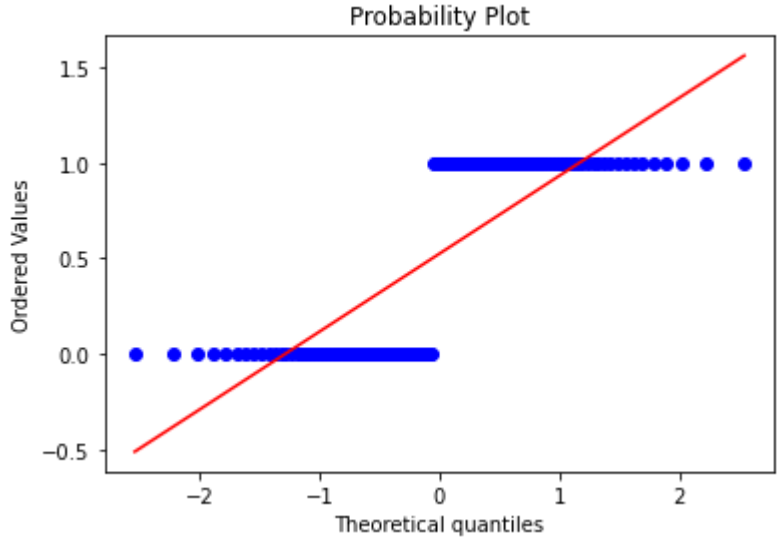
In [124]:
```
#z_swing_miss_percent
stats.probplot(bav.z_swing_miss_percent, dist = "norm", plot = plt)
plt.show()

plt.figure(figsize = (15, 6))
sns.regplot(x = bav.z_swing_miss_percent, y = result_bm.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("z_swing_miss_percent")
plt.title("Residuals Plot")
```

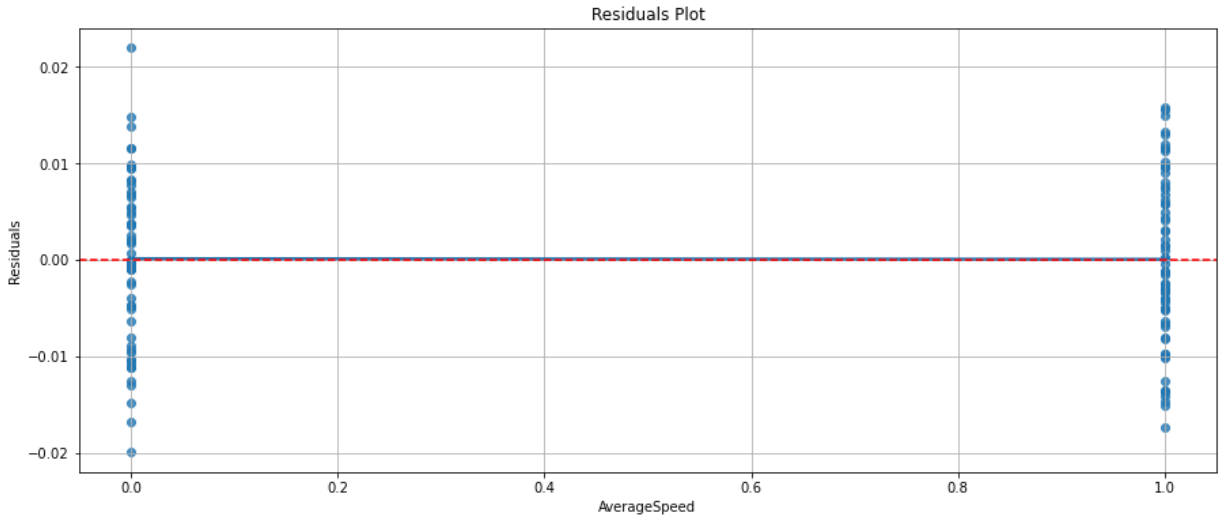


Out[124]: Text(0.5, 1.0, 'Residuals Plot')



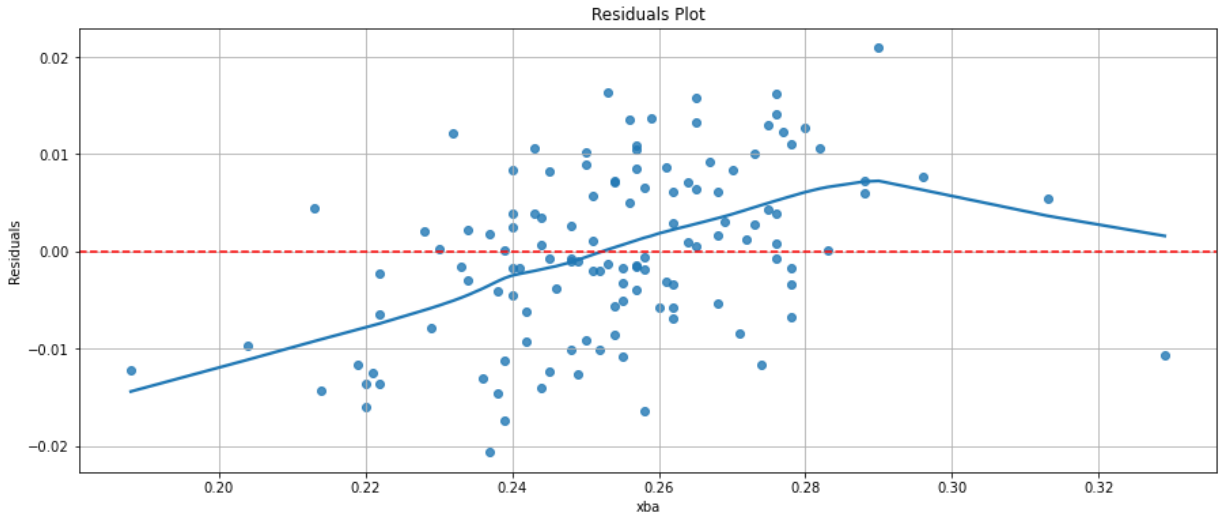
In [125]:
```
#AverageSpeed
stats.probplot(bav.AverageSpeed, dist = "norm", plot = plt)
plt.show()

plt.figure(figsize = (15, 6))
sns.regplot(x = bav.AverageSpeed, y = result_bm.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("AverageSpeed")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



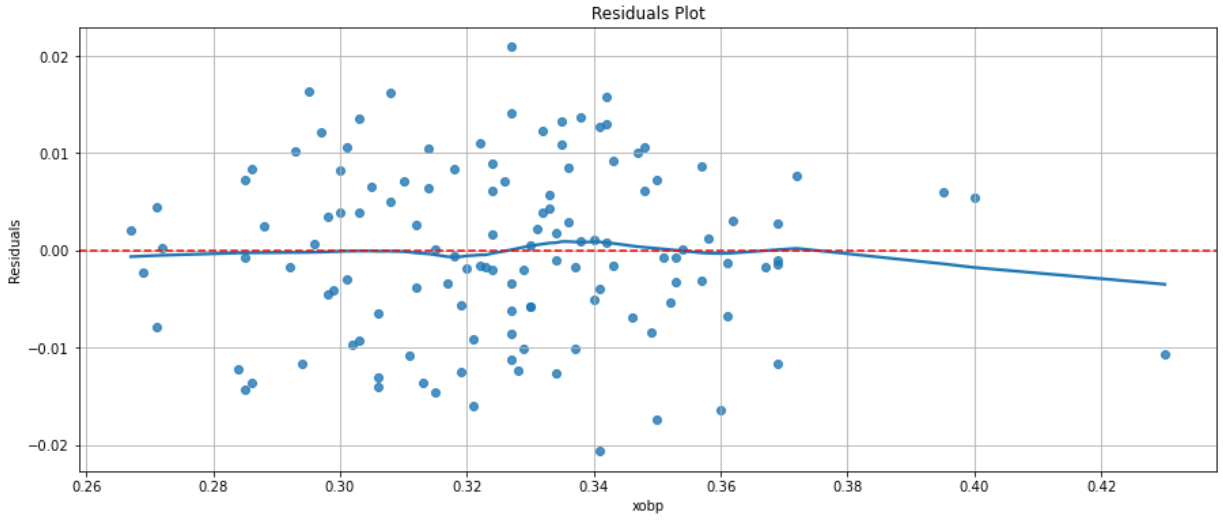


Model 2

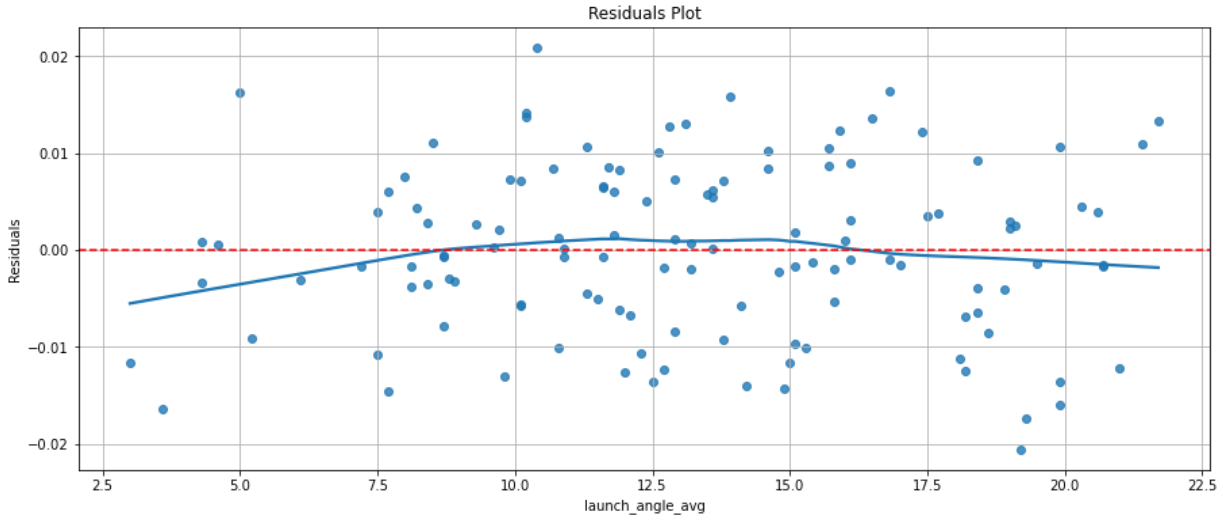
In [18]:
```
#xba
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xba, y = result2.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xba")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



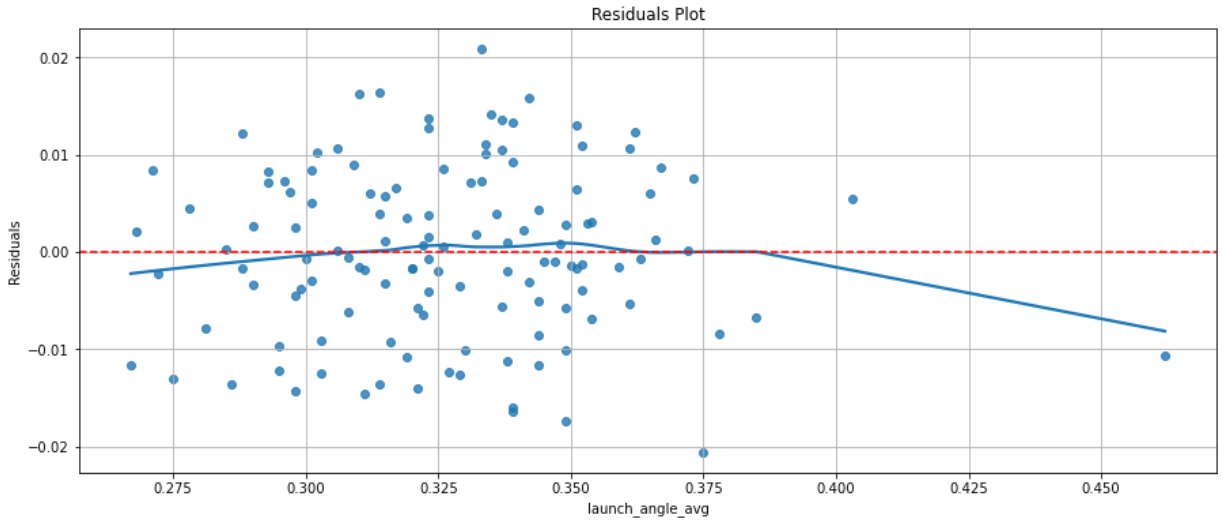
In [19]:
```
#xobp
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xobp, y = result2.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xobp")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



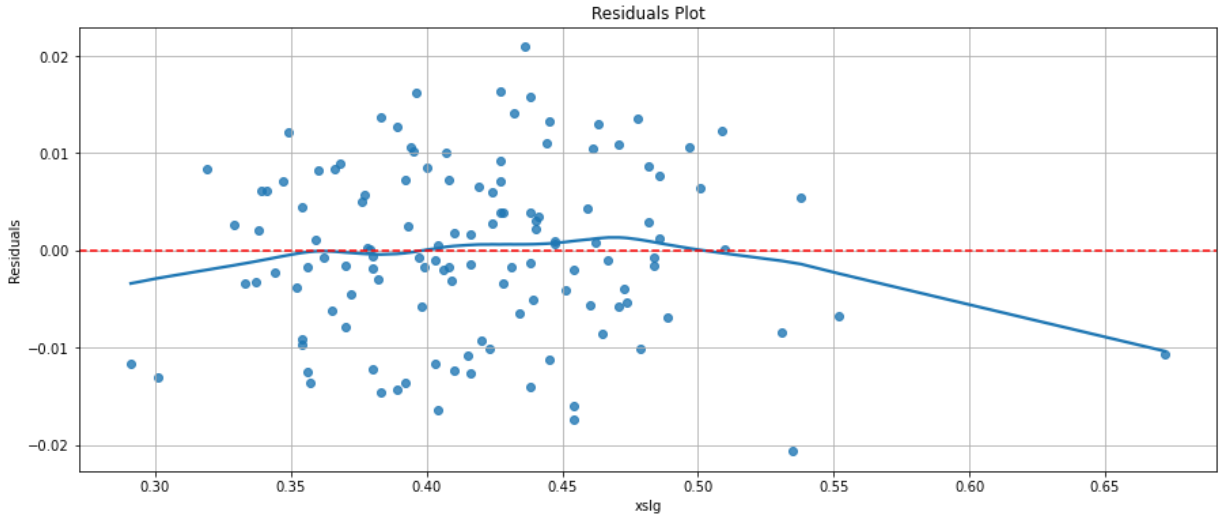
In [20]:
```
#launch_angle_avg
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.launch_angle_avg, y = result2.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("launch_angle_avg")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



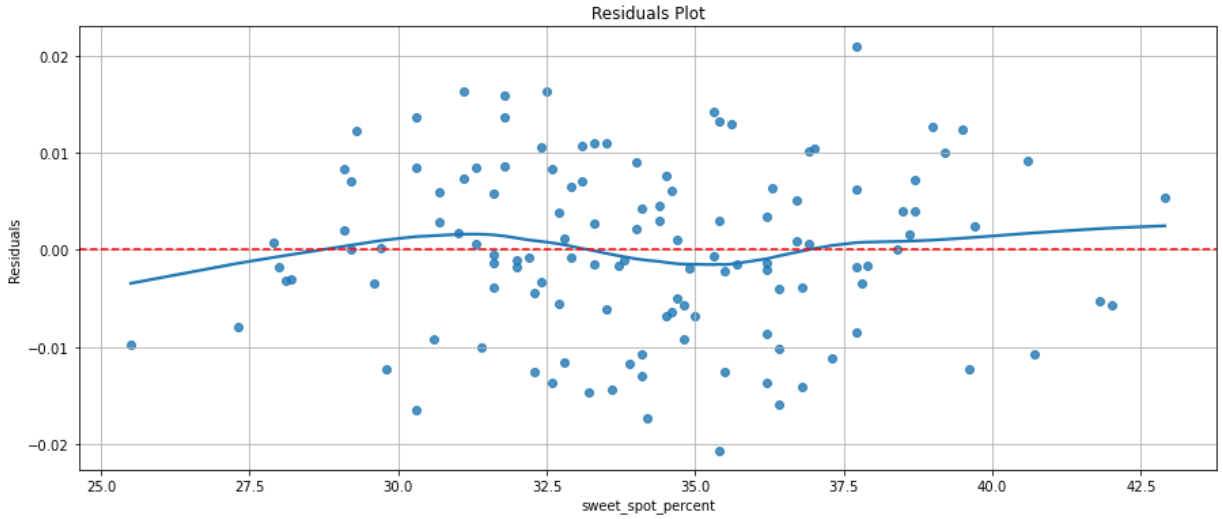
In [22]:
```
#launch_angle_avg
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xwoba, y = result2.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("launch_angle_avg")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



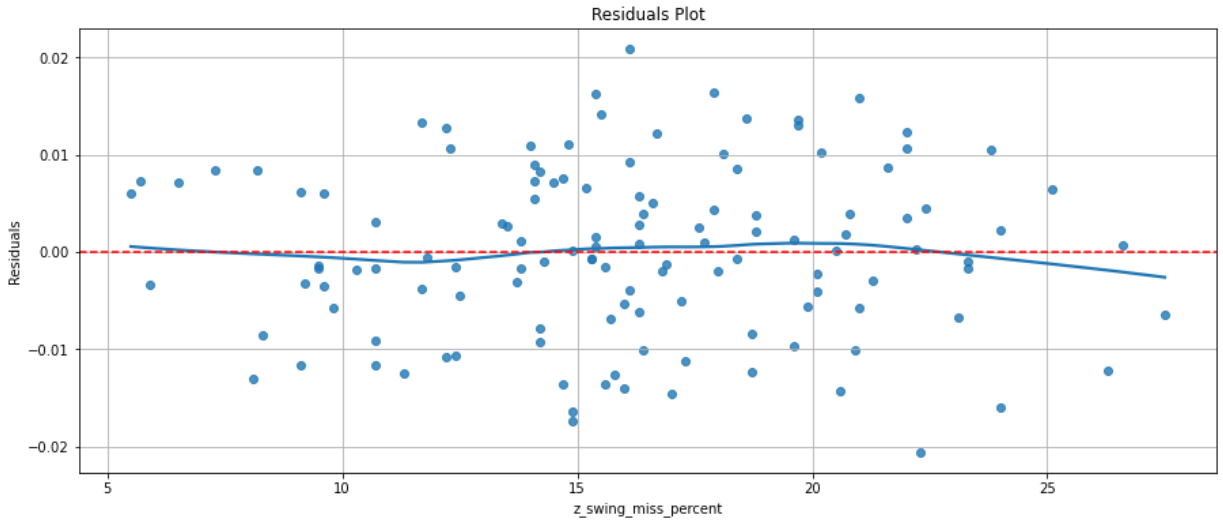
In [21]:
```
#xslg
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xslg, y = result2.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xslg")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



In [23]:
```
#sweet_spot_percent
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.sweet_spot_percent, y = result2.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("sweet_spot_percent")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```

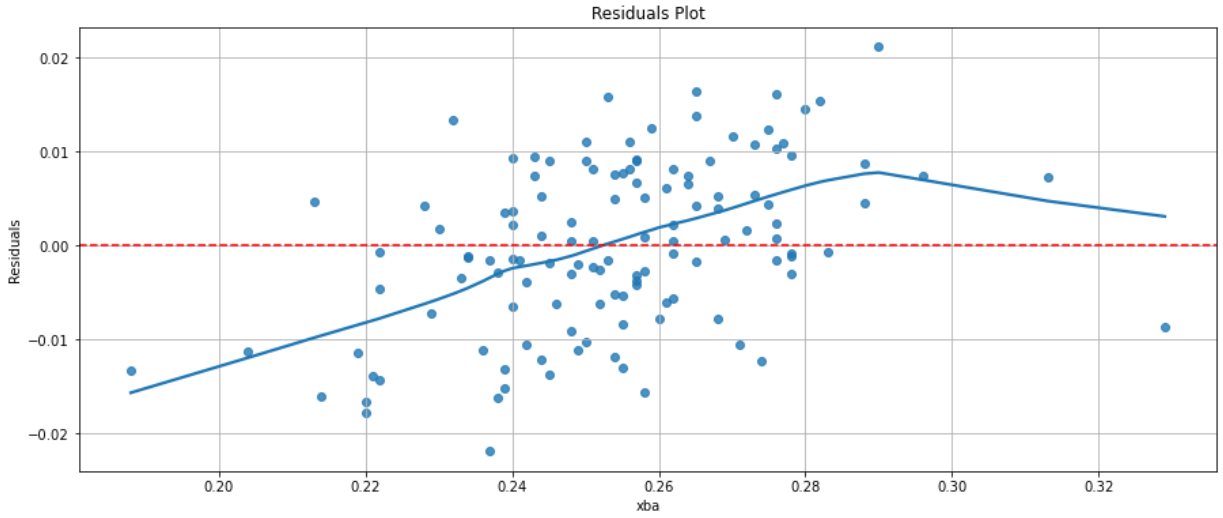


In [24]:
```
#z_swing_miss_percent
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.z_swing_miss_percent, y = result2.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("z_swing_miss_percent")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```

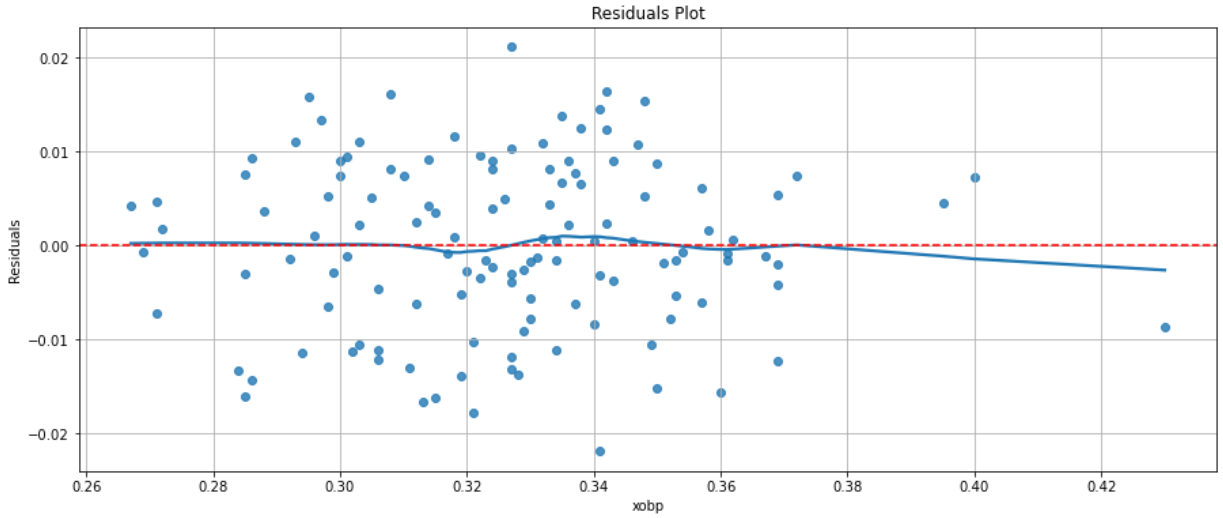


Model 3

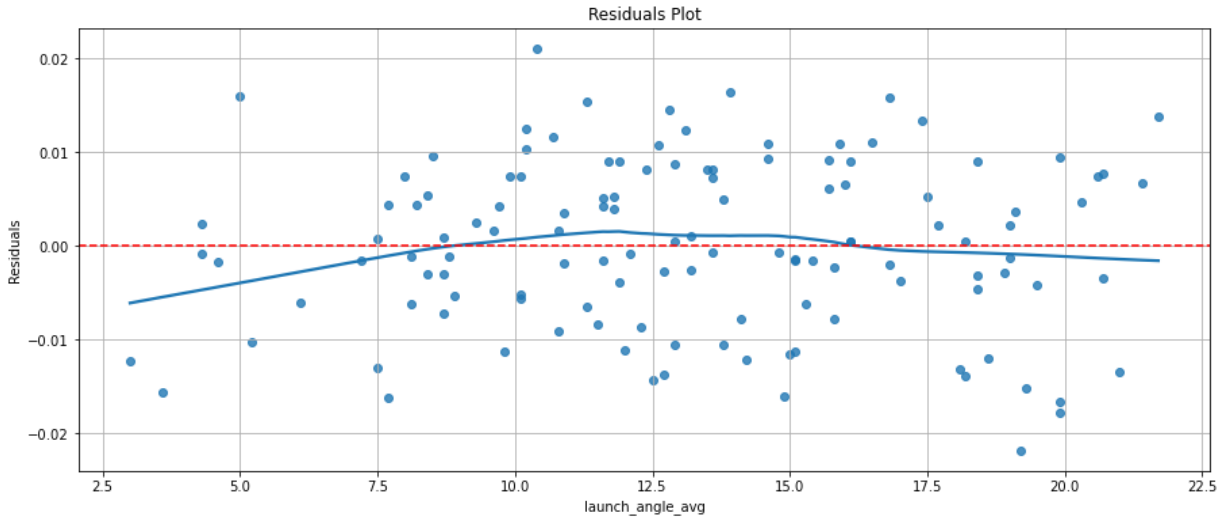
In [25]:
```
#xba
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xba, y = result3.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xba")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



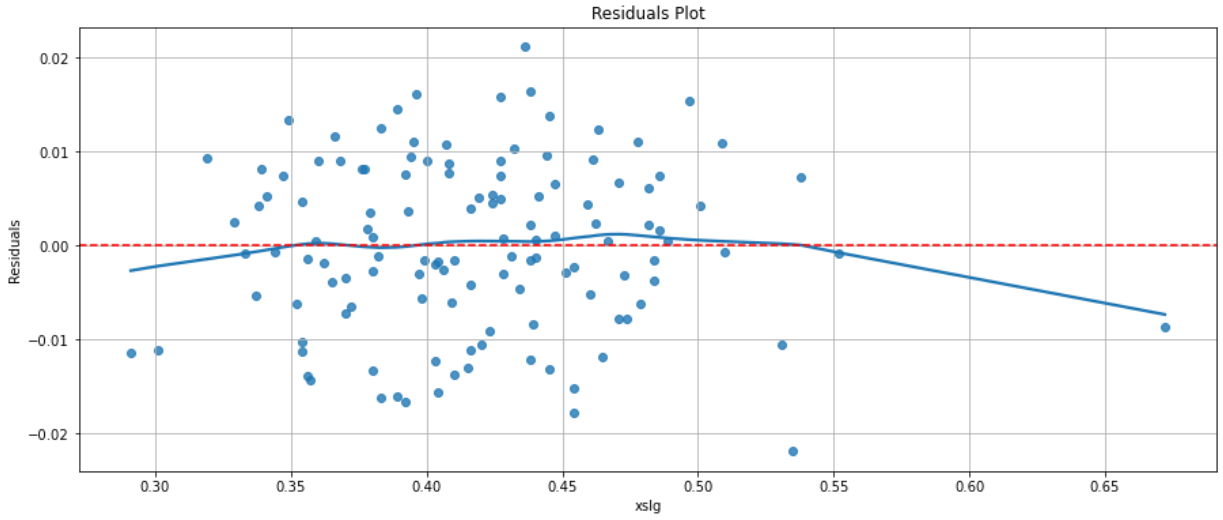
In [26]:
```
#xobp
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xobp, y = result3.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xobp")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



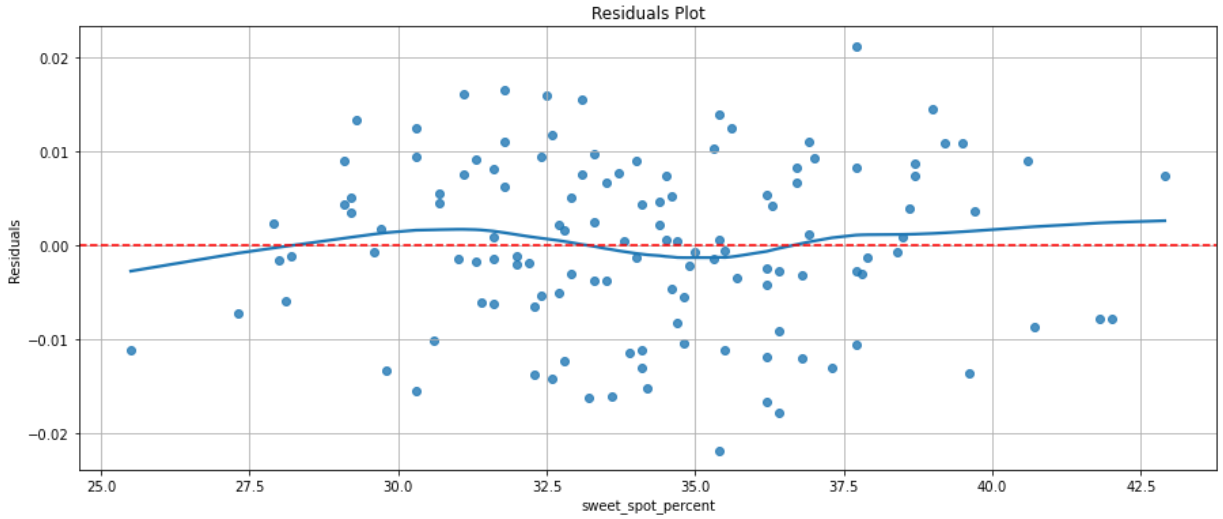
In [27]:
```
#launch_angle_avg
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.launch_angle_avg, y = result3.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("launch_angle_avg")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



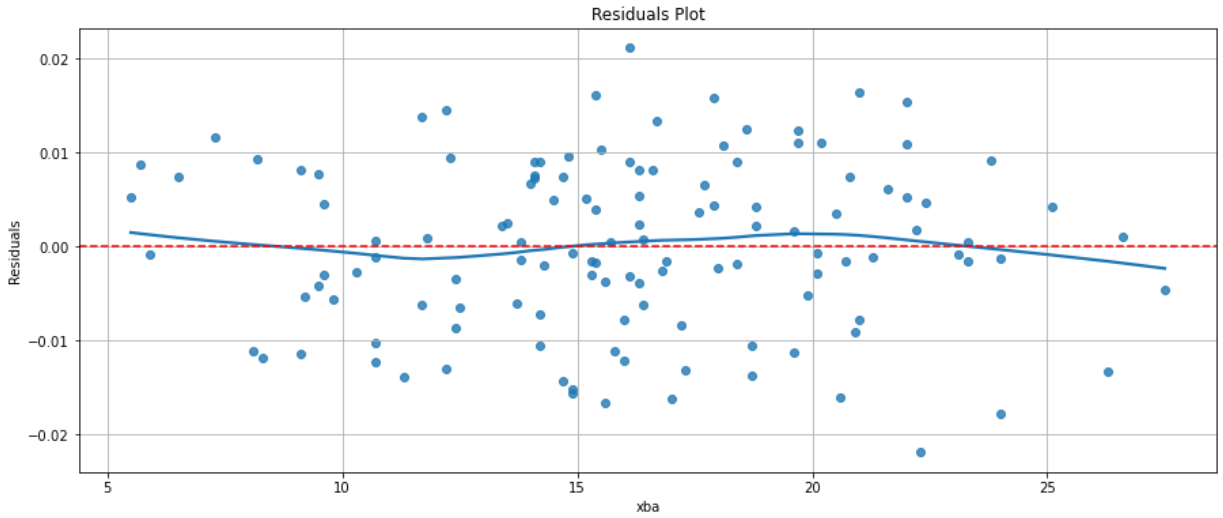
In [28]:
```
#xslg
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xslg, y = result3.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xslg")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



In [29]:
```
#sweet_spot_percent
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.sweet_spot_percent, y = result3.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("sweet_spot_percent")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```

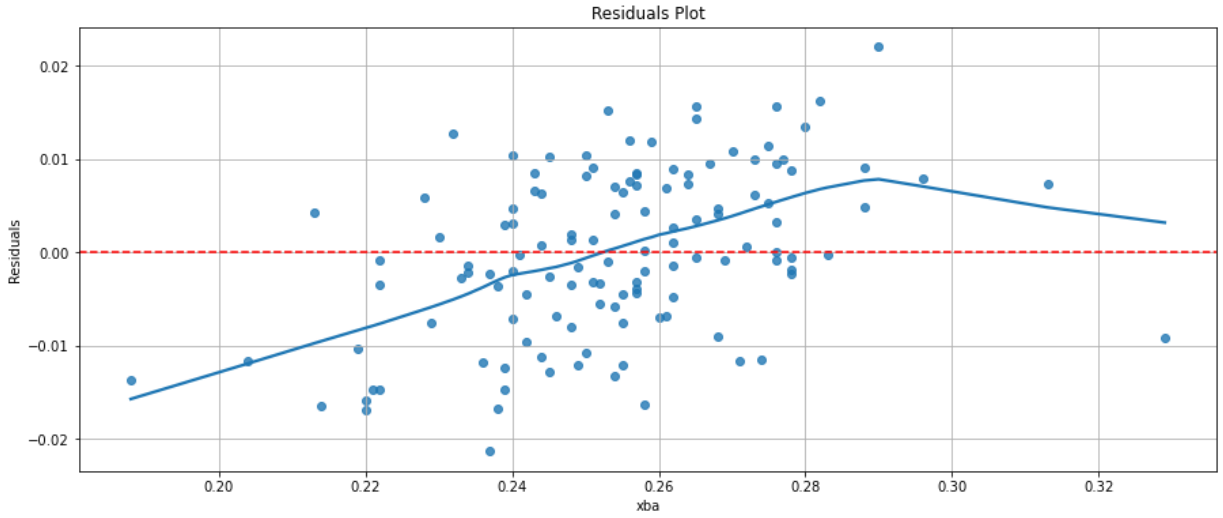


In [30]:
```
#z_swing_miss_percent
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.z_swing_miss_percent, y = result3.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xba")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```

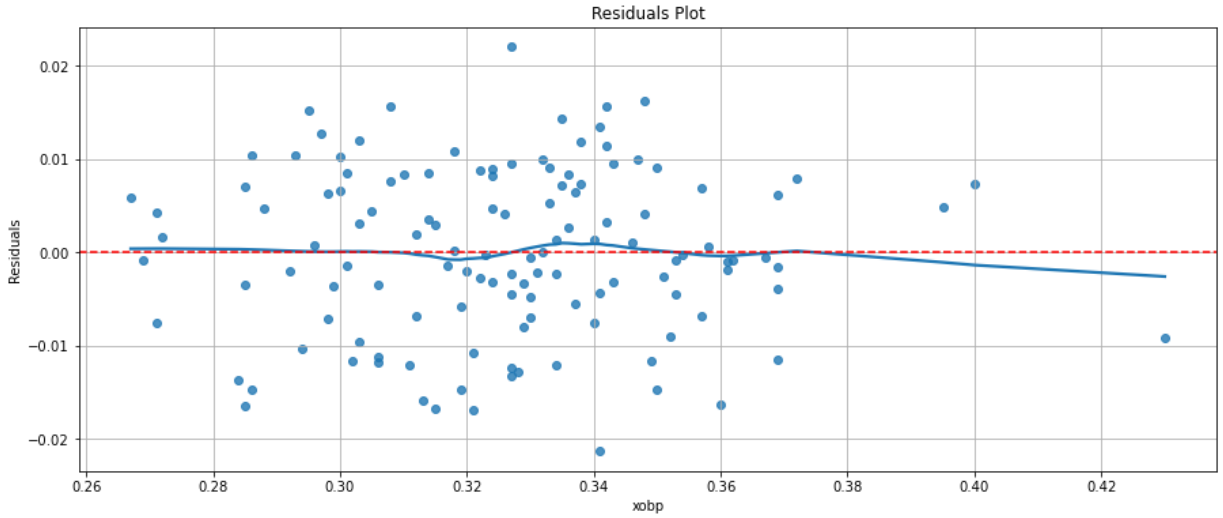


Model 4

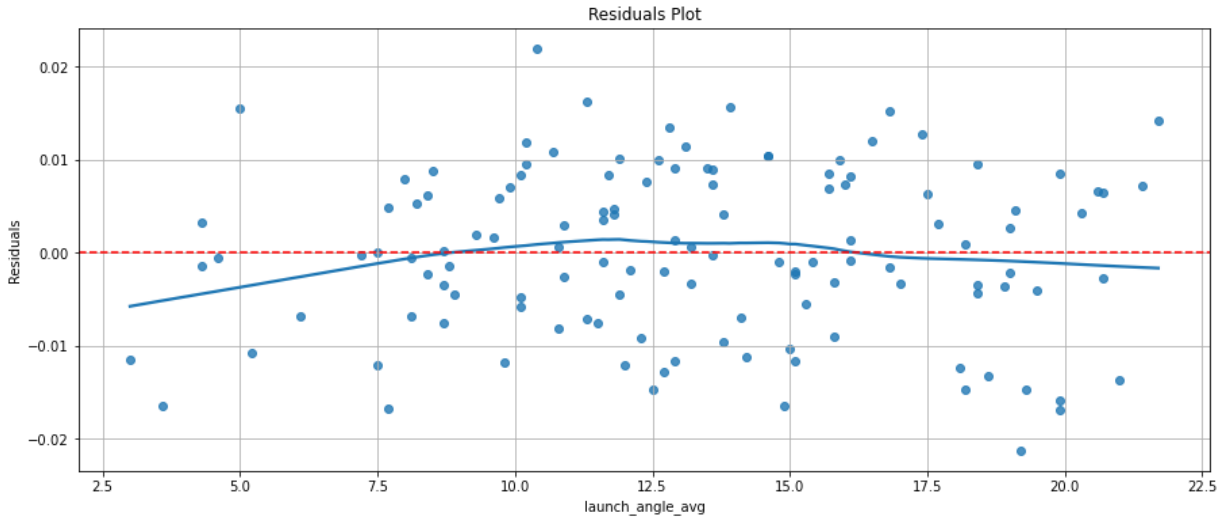
In [31]:
```
#xba
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xba, y = result4.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xba")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



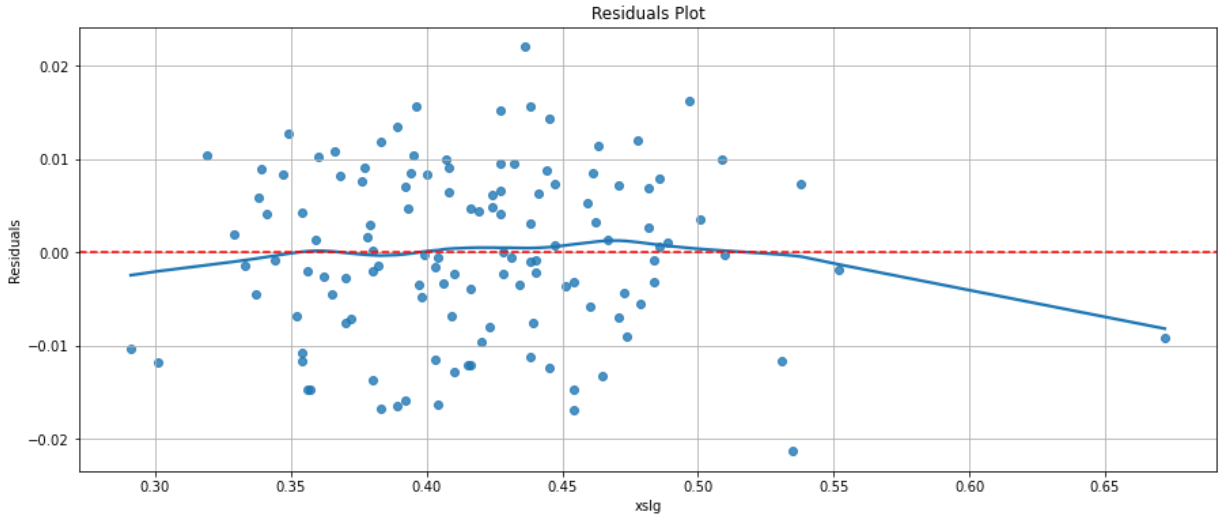
In [32]:
```
#xobp
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xobp, y = result4.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xobp")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



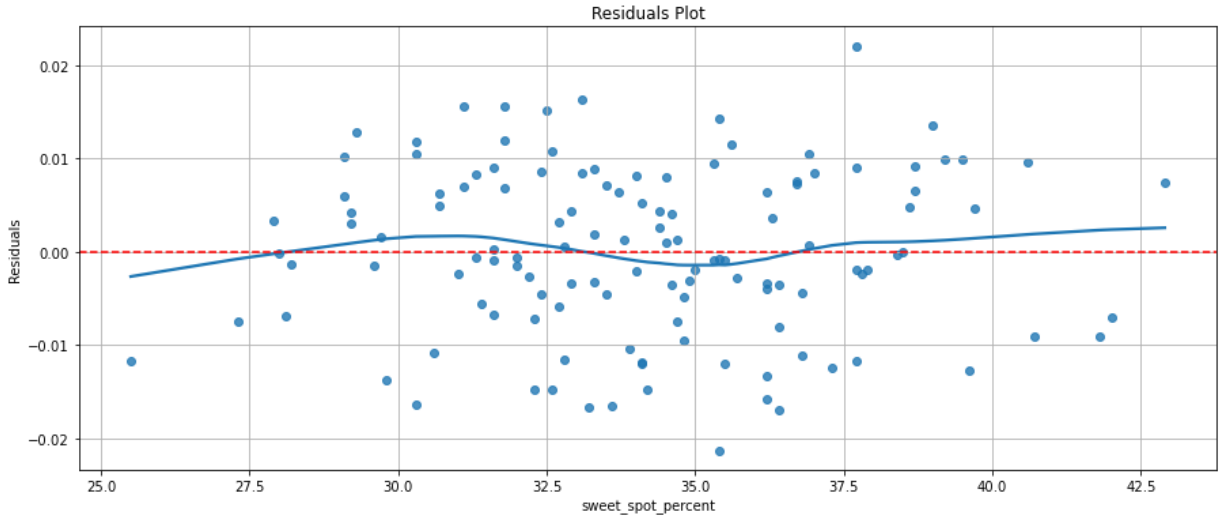
In [33]:
```
#launch_angle_avg
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.launch_angle_avg, y = result4.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("launch_angle_avg")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



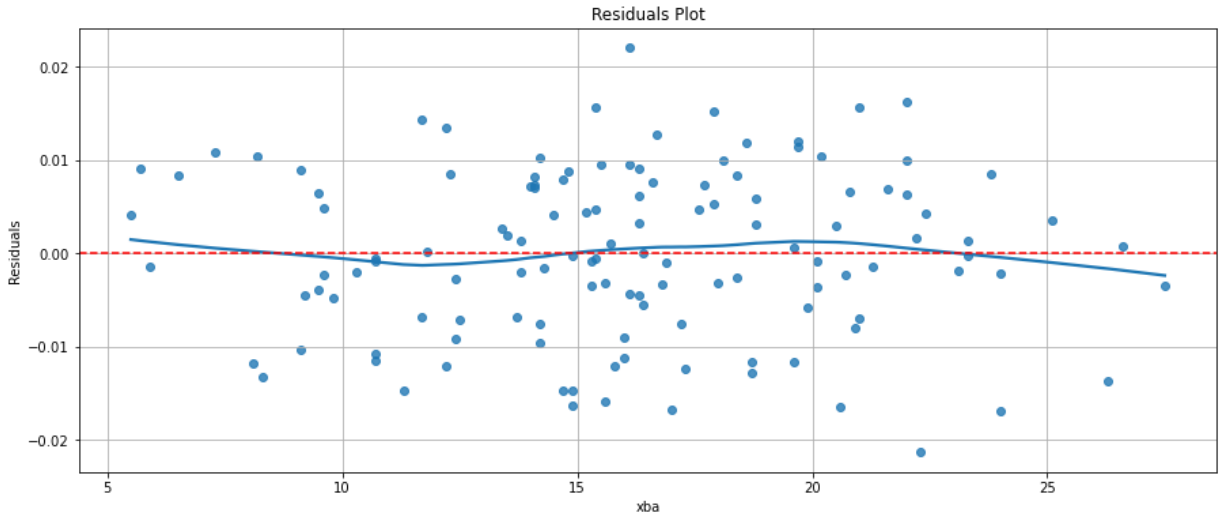
In [34]:
```
#xslg
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.xslg, y = result4.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xslg")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



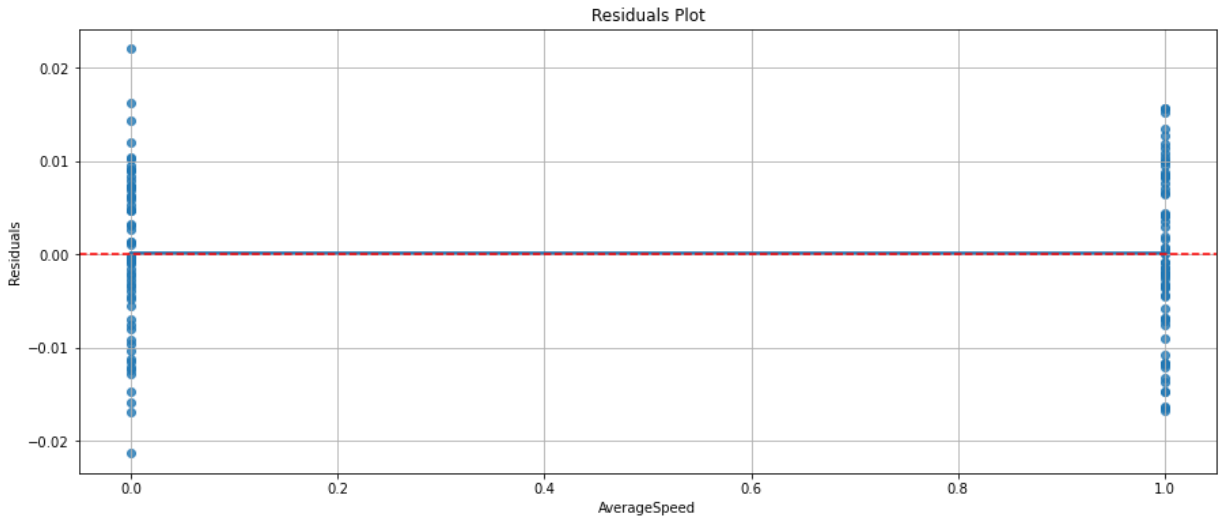
In [35]:
```
#sweet_spot_percent
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.sweet_spot_percent, y = result4.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("sweet_spot_percent")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



In [36]:
```
#z_swing_miss_percent
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.z_swing_miss_percent, y = result4.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("xba")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



In [37]:
```
#AverageSpeed
plt.figure(figsize = (15, 6))
sns.regplot(x = bav.AverageSpeed, y = result4.resid, lowess = True)
plt.axhline(0, linestyle = '--', color = "red")
plt.ylabel("Residuals")
plt.xlabel("AverageSpeed")
plt.title("Residuals Plot")
plt.grid()
plt.show()
```



Testing multicollinearity

Model 1

In [75]:
```
y, X1 = pt.dmatrices('xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spot_p
                     data=bav, return_type='dataframe')

K = X1.shape[1]
VIF = np.empty(K)
for i in range(K):
    VIF[i] = smo.variance_inflation_factor(X1.values, i)
print(f'VIF: \n{VIF}\n')
```

```
VIF:
[350.59993073  58.93040708   1.27300303 231.27470609  93.79190065
   1.36081954   1.85165228   1.15542874]
```

Model 2

In [76]:
```
y, X2 = pt.dmatrices('xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spot_p
                     data=bav, return_type='dataframe')

K = X2.shape[1]
VIF = np.empty(K)
for i in range(K):
    VIF[i] = smo.variance_inflation_factor(X2.values, i)
print(f'VIF: \n{VIF}\n')
```

```
VIF:
[329.53271866  58.60986253   1.25393559 230.7766351   93.44328737
   1.35955043   1.82739989]
```

Model 3

In [77]:
```
y, X3 = pt.dmatrices('xba ~ xobp + launch_angle_avg + xslg + sweet_spot_percent +
                     data=bav, return_type='dataframe')

K = X3.shape[1]
VIF = np.empty(K)
for i in range(K):
    VIF[i] = smo.variance_inflation_factor(X3.values, i)
print(f'VIF: \n{VIF}\n')
```

```
VIF:
[270.33941023   2.08748062   1.25383895   2.52672659   1.35777406
   1.61557459]
```

Model 4

In [78]:
```
y, X4 = pt.dmatrices('xba ~ xobp + launch_angle_avg + xslg + sweet_spot_percent +
                     data=ba_T, return_type='dataframe')

K = X4.shape[1]
VIF = np.empty(K)
for i in range(K):
    VIF[i] = smo.variance_inflation_factor(X4.values, i)
print(f'VIF: \n{VIF}\n')
```

```
VIF:
[641.25280374   2.03367371   1.26822134   2.51011995   1.35597123
   1.69117524   1.01781104]
```

AIC and BIC

In [85]:
```
print("AIC for model 1 is: ", result_bm.aic)
print("AIC for model 2 is: ", result2.aic)
print("AIC for model 3 is: ", result3.aic)
print("AIC for model 4 is: ", result4.aic)
```

```
AIC for model 1 is:  -824.9962955155124
AIC for model 2 is:  -825.6050379209653
AIC for model 3 is:  -819.4243265438613
AIC for model 4 is:  -818.4421969398902
```

In [86]:
```
print("BIC for model 1 is: ", result_bm.bic)
print("BIC for model 2 is: ", result2.bic)
print("BIC for model 3 is: ", result3.bic)
print("BIC for model 4 is: ", result4.bic)
```

```
BIC for model 1 is:  -802.3060402599006
BIC for model 2 is:  -805.7510645723049
BIC for model 3 is:  -802.4066351021524
BIC for model 4 is:  -798.5882235912298
```

According to AIC and BIC our best model is Model 2. Model 1 is the second best model. This is different from what the VIF metric was suggesting.
We still do not know the impact of multicollinearity in our AIC and BIC scores.

F-test

Model 1

In [13]:
```
hypotheses = ['xobp = 0', "launch_angle_avg = 0", "xwoba = 0", "xslg = 0", "sweet
result_bm.f_test(hypotheses)
```

Out[13]:
```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=79.5475656934631, p=1.1241959385664292e-41, df_denom=118, df_num=7>
```

Model 2

In [14]:
```
hypotheses = ['xobp = 0', "launch_angle_avg = 0", "xwoba = 0", "xslg = 0", "sweet
result2.f_test(hypotheses)
```

Out[14]:
```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=92.34645820970707, p=2.13215785359752e-42, df_denom=119, df_num=6>
```

Model 3

In [85]:
```
hypotheses = ['xobp = 0', "launch_angle_avg = 0", "xslg = 0", "sweet_spot_percent
result3.f_test(hypotheses)
```

Out[85]:
```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=103.21345091928012, p=9.204327507617126e-42, df_denom=120, df_num=5>
```

Model 4

In [86]:
```
hypotheses = ['xobp = 0', "launch_angle_avg = 0", "xslg = 0", "sweet_spot_percent
result4.f_test(hypotheses)
```

Out[86]:
```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=86.14714617249983, p=6.124075673129004e-41, df_denom=119, df_num=6>
```

Testing heteroskedasticity

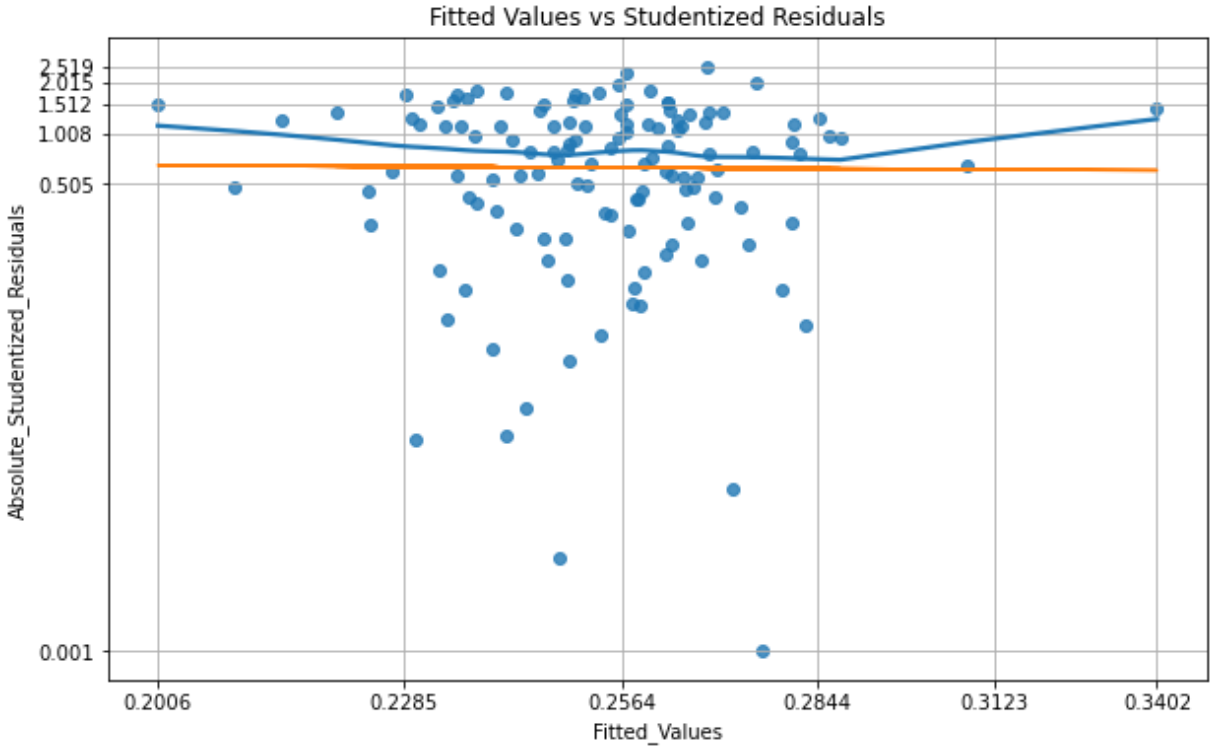
In [18]:
```
def spread_level(model, data):
    df_copy = data.copy()

    df_copy["Absolute_Studentized_Residuals"] = (np.abs(model.get_influence().res
    df_copy["Fitted_Values"] = (model.fittedvalues)

    slreg = smf.rlm("np.log(Absolute_Studentized_Residuals) ~ np.log(Fitted_Value
    slope = slreg.params[1]

    fig, ax = plt.subplots(figsize = (10, 6))
    ax.set_title("Fitted Values vs Studentized Residuals")
    sns.regplot(x = "Fitted_Values", y = "Absolute_Studentized_Residuals", data =
    ax.plot(df_copy.Fitted_Values.values, np.exp(slreg.fittedvalues).values)

    ax.set_yscale('log')
    ax.set_xscale('log')

    ax.yaxis.set_major_formatter(ScalarFormatter())
    ax.xaxis.set_major_formatter(ScalarFormatter())

    ax.minorticks_off()

    ax.set_xticks(np.linspace(df_copy["Fitted_Values"].min(),df_copy["Fitted_Valu
    ax.set_yticks(np.linspace(df_copy["Absolute_Studentized_Residuals"].min(),
                              df_copy["Absolute_Studentized_Residuals"].max(), 6)

    ax.grid()
```

Model 1

In [15]:
```
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
test = sms.het_breuschpagan(result_bm.resid, result_bm.model.exog)
print("BP Results:",['Model 1'])
print(list(zip(name, test)))
```

```
BP Results: ['Model 1']
[('Lagrange multiplier statistic', 3.7226306910109495), ('p-value', 0.811113026
5611785), ('f-value', 0.5132014020041799), ('f p-value', 0.8231014888700311)]
```

In [19]:
```
spread_level(result_bm, bav)
```

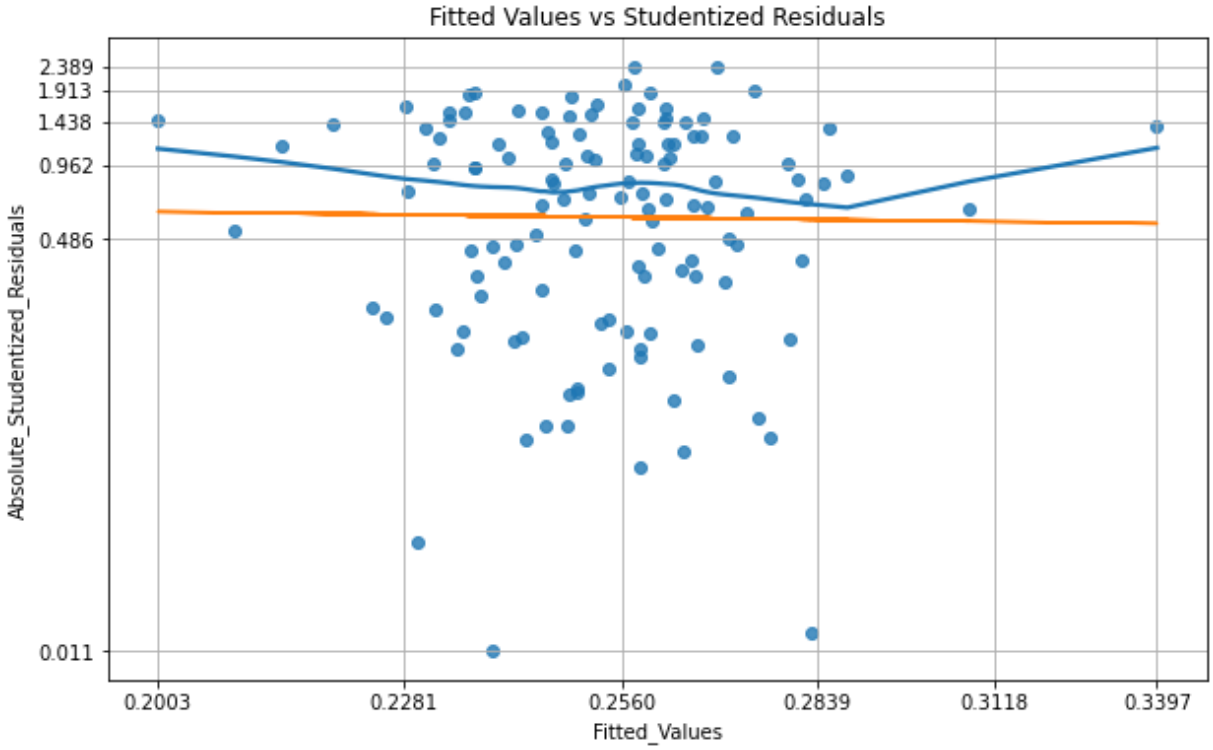


Model 2

In [16]:
```
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
test = sms.het_breuschpagan(result2.resid, result2.model.exog)
print("BP Results:",['Model 2'])
print(list(zip(name, test)))
```

```
BP Results: ['Model 2']
[('Lagrange multiplier statistic', 4.017001092591057), ('p-value', 0.6743755823
272788), ('f-value', 0.6531280783660387), ('f p-value', 0.687515954594228)]
```

In [20]:
```
spread_level(result2, bav)
```

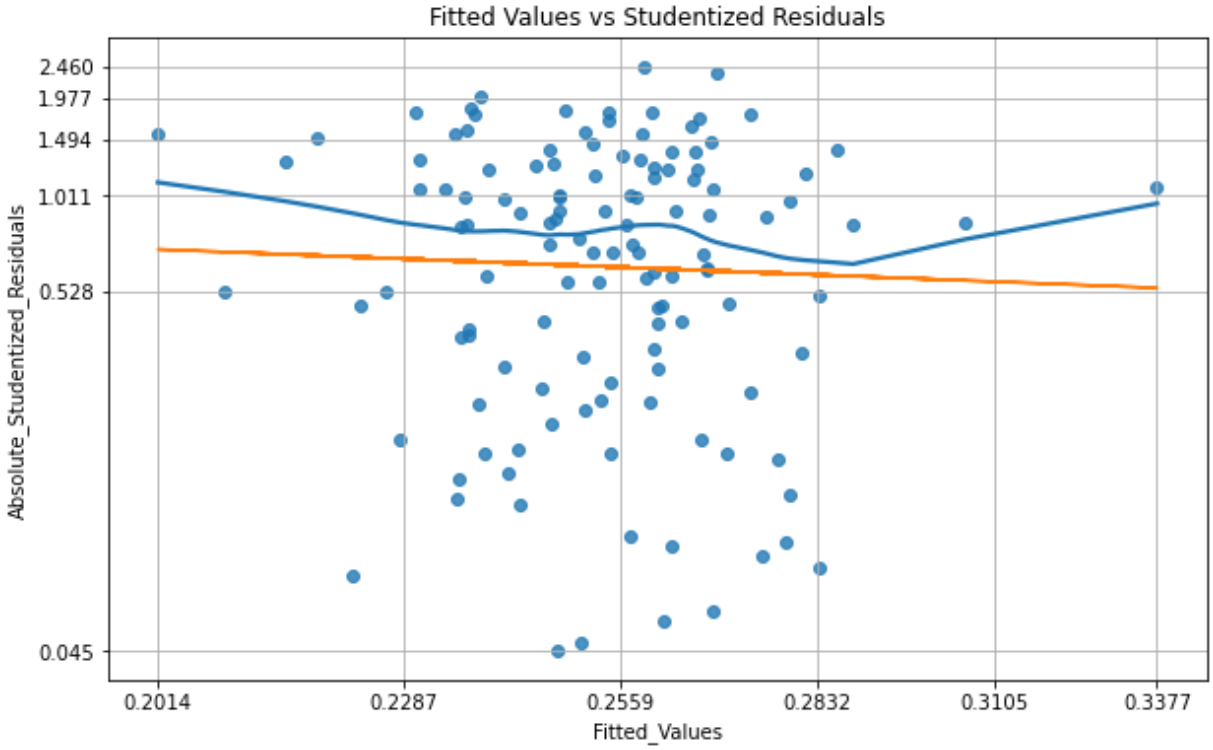


Model 3

In [89]:
```
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
test = sms.het_breuschpagan(result3.resid, result3.model.exog)
print("BP Results:",['Model 3'])
print(list(zip(name, test)))
```

```
BP Results: ['Model 3']
[('Lagrange multiplier statistic', 2.218549021320912), ('p-value', 0.8181522025
390319), ('f-value', 0.4301547290867777), ('f p-value', 0.8268494543574764)]
```

In [94]:
```
spread_level(result3, bav)
```

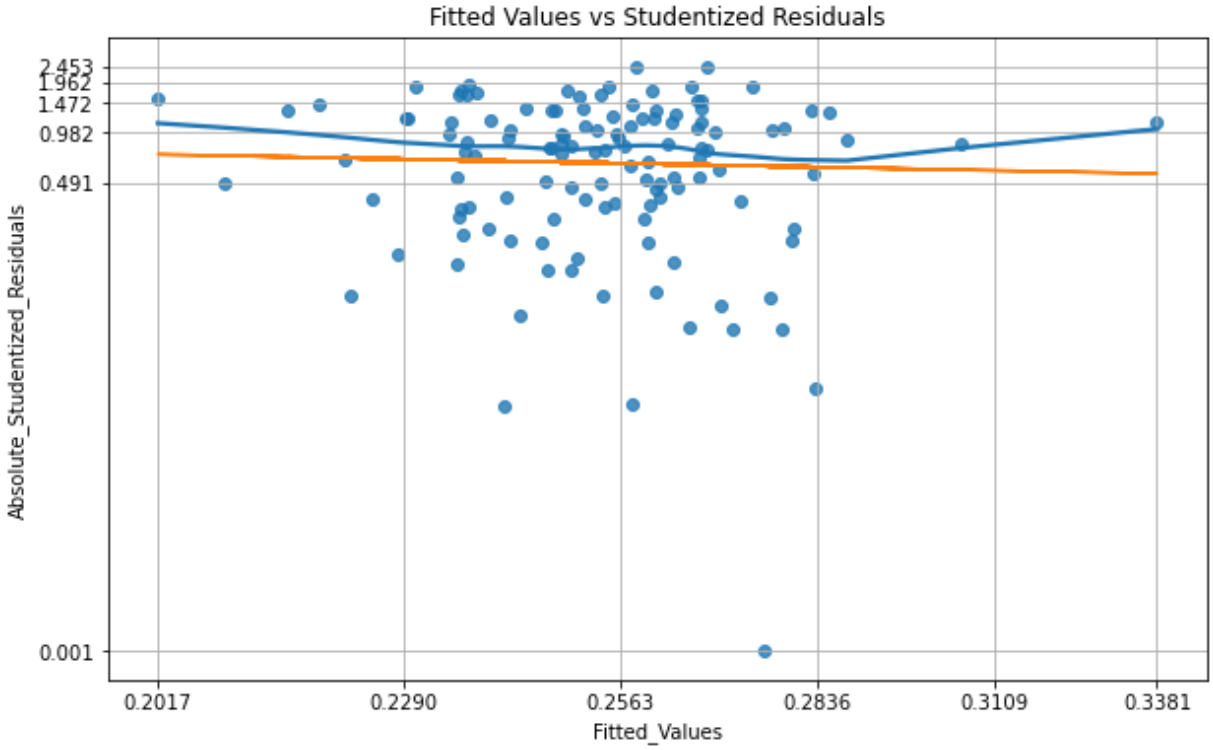


Model 4

In [90]:
```
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
test = sms.het_breuschpagan(result4.resid, result4.model.exog)
print("BP Results:",['Model 4'])
print(list(zip(name, test)))
```

```
BP Results: ['Model 4']
[('Lagrange multiplier statistic', 1.7991280630387265), ('p-value', 0.937214833
4545469), ('f-value', 0.287298358112271), ('f p-value', 0.9419920230929079)]
```

In [95]:
```
spread_level(result4, bav)
```



Results

We fail to reject H_0 for each model, therefore, we assume that there is no heteroskedasticity.

Testing model misspecification

Model 1

In [76]:
```
test =  dg.linear_reset(result_bm, power=2,  test_type='fitted', use_f = True)

print("Ramsey-RESET: Model 1")
print(test)
```

```
Ramsey-RESET: Model 1
<F test: F=2.524748358168967, p=0.11477135676141413, df_denom=117, df_num=1>
```

Model 2

In [77]:
```
test =  dg.linear_reset(result2, power=2,  test_type='fitted', use_f = True)

print("Ramsey-RESET: Model 2")
print(test)
```

```
Ramsey-RESET: Model 2
<F test: F=2.314851709699946, p=0.13081877588109372, df_denom=118, df_num=1>
```

Model 3

In [78]:
```
test =  dg.linear_reset(result3, power=2,  test_type='fitted', use_f = True)

print("Ramsey-RESET: Model 3")
print(test)
```

```
Ramsey-RESET: Model 3
<F test: F=1.3669038521132728, p=0.24468114275171823, df_denom=119, df_num=1>
```

Model 4

In [80]:
```
test =  dg.linear_reset(result4, power=2,  test_type='fitted', use_f = True)

print("Ramsey-RESET: Model 4")
print(test)
```

```
Ramsey-RESET: Model 4
<F test: F=1.468353467712061, p=0.2280265752518757, df_denom=118, df_num=1>
```

Results

We fail to reject H_0 for all the models, therefore, they are correctly specified as linear.

White test

Model 1

In [14]:
```
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
test = sm.stats.diagnostic.het_white(result_bm.resid, result_bm.model.exog)
print("White SE Results:",['Model 1'])
print(list(zip(name, test)))
```

```
White SE Results: ['Model 1']
[('Lagrange multiplier statistic', 36.09217311360245), ('p-value', 0.3709869020
766881), ('f-value', 1.0744297037243646), ('f p-value', 0.3835730795430564)]
```

Model 2

In [15]:
```
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
test = sm.stats.diagnostic.het_white(result2.resid, result2.model.exog)
print("White SE Results:",['Model 2'])
print(list(zip(name, test)))
```

```
White SE Results: ['Model 2']
[('Lagrange multiplier statistic', 31.117016579034843), ('p-value', 0.266361153
13745506), ('f-value', 1.1903424754241607), ('f p-value', 0.2637957384808572)]
```

Model 3

In [16]:
```
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
test = sm.stats.diagnostic.het_white(result3.resid, result3.model.exog)
print("White SE Results:",['Model 3'])
print(list(zip(name, test)))
```

```
White SE Results: ['Model 3']
[('Lagrange multiplier statistic', 17.943744037652102), ('p-value', 0.591114210
1104854), ('f-value', 0.871811218690559), ('f p-value', 0.6220254548026218)]
```

Model 4

In [17]:
```
name = ["Lagrange multiplier statistic", "p-value", "f-value", "f p-value"]
test = sm.stats.diagnostic.het_white(result4.resid, result4.model.exog)
print("White SE Results:",['Model 4'])
print(list(zip(name, test)))
```

```
White SE Results: ['Model 4']
[('Lagrange multiplier statistic', 21.427755213245526), ('p-value', 0.719561246
1248521), ('f-value', 0.7802290068743299), ('f p-value', 0.7618584270804245)]
```

Results

We fail to reject H_0 for all the models, therefore, we assume that there is no heteroskedasticity.

Cross-validation

Model 1

In [61]:
```
kf = KFold(n_splits=5)

mse1 = []
for train_index, test_index in kf.split(bav):
    results = smf.ols('xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spot_
    s = ((bav.iloc[test_index]['xba'] - results.predict(bav.iloc[test_index])**2)
    mse1.append(s)

mse1
```

Out[61]:
```
[0.19106043457575603,
 0.18768914385924787,
 0.18723060368412262,
 0.18946473958271423,
 0.19014069233827263]
```

In [62]:
```
(result_bm.resid**2).mean()
```

Out[62]:
```
7.392563959327307e-05
```

In [63]:
```
x = bav[['xba']]
y = bav[['xobp','launch_angle_avg','xslg','xwoba','sweet_spot_percent','z_swing_m
regr = LinearRegression()
score1 = cross_val_score(regr,x,y, cv=5,scoring='neg_mean_squared_error')
print('5-Fold CV MSE Scores:', score1)
```

```
5-Fold CV MSE Scores: [-7.52114418 -6.0402728  -6.84550113 -7.94557186 -5.79687
028]
```

Model 2

In [64]:
```
kf = KFold(n_splits=5)

mse2 = []
for train_index, test_index in kf.split(bav):
    results = smf.ols('xba ~ xobp + launch_angle_avg + xwoba + xslg + sweet_spot_
    s = ((bav.iloc[test_index]['xba'] - results.predict(bav.iloc[test_index])**2)
    mse2.append(s)

mse2
```

Out[64]:
```
[0.1908712477338212,
 0.1875732390886567,
 0.18725356532483395,
 0.18952171911205898,
 0.19038722342863848]
```

In [65]:
```
(result2.resid**2).mean()
```

Out[65]:
```
7.474642944913205e-05
```

In [66]:
```
x = bav[['xba']]
y = bav[['xobp','launch_angle_avg','xslg','xwoba','sweet_spot_percent','z_swing_m
regr = LinearRegression()
score2 = cross_val_score(regr,x,y, cv=5,scoring='neg_mean_squared_error')
print('5-Fold CV MSE Scores:', score2)
```

```
5-Fold CV MSE Scores: [-8.73148429 -6.99745637 -7.94768646 -9.22991679 -6.72355
94 ]
```

Model 3

In [67]:
```
mse3 = []
for train_index, test_index in kf.split(bav):
    results = smf.ols('xba ~ xobp + launch_angle_avg + xslg + sweet_spot_percent
    s = ((bav.iloc[test_index]['xba'] - results.predict(bav.iloc[test_index])**2)
    mse3.append(s)

mse3
```

Out[67]:
```
[0.1907944236651124,
 0.18751860554903158,
 0.18727188308808504,
 0.18962665927225913,
 0.190467638653148]
```

In [68]:
```
(result3.resid**2).mean()
```

Out[68]:
```
7.976044663174517e-05
```

In [69]:
```
x = bav[['xba']]
y = bav[['xobp','launch_angle_avg','xslg','sweet_spot_percent','z_swing_miss_perc
regr = LinearRegression()
score3 = cross_val_score(regr,x,y, cv=5,scoring='neg_mean_squared_error')
print('5-Fold CV MSE Scores:', score3)
```

```
5-Fold CV MSE Scores: [-10.47769337  -8.39687982  -9.53713997 -11.07575234  -8.
06818123]
```

Model 4

In [70]:
```
ex, test_index in kf.split(bav):
 smf.ols('xba ~ xobp + launch_angle_avg + xslg + sweet_spot_percent + z_swing_miss
.iloc[test_index]['xba'] - results.predict(bav.iloc[test_index])**2)).mean()
nd(s)
```

Out[70]:
```
[0.19095969798813764,
 0.1876437338912218,
 0.18725409210594865,
 0.18958179244289325,
 0.19027306329443547]
```

In [71]:
```
(result4.resid**2).mean()
```

Out[71]:
```
7.911871047658528e-05
```

In [72]:
```
x = bav[['xba']]
y = bav[['xobp','launch_angle_avg','xslg','sweet_spot_percent','z_swing_miss_perc
regr = LinearRegression()
score4 = cross_val_score(regr,x,y, cv=5,scoring='neg_mean_squared_error')
print('5-Fold CV MSE Scores:', score4)
```

```
5-Fold CV MSE Scores: [-8.77459505 -7.04692841 -7.98634817 -9.26971066 -6.76294
029]
```

Comparison

In [73]:
```
score_tot = score1, score2, score3, score4
count = 1
for score in score_tot:
    mean = abs(score.mean())
    print("Model " + str(count) + ": " + str(mean))
    count += 1
```

```
Model 1: 6.829872051718472
Model 2: 7.926020663423893
Model 3: 9.511129344724765
Model 4: 7.968104517518294
```

Out of sample performance

Model 1

In [13]:
```
x = bav[['xba']]
y = bav[['xobp','launch_angle_avg','xwoba','xslg','sweet_spot_percent','z_swing_m
regr = LinearRegression()
model = regr.fit(x,y)
regr.coef_
regr.intercept_
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3, random_st
regr = LinearRegression()
regr.fit(x_train,y_train)
y_pred=regr.predict(x_test)
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)))
regr = linear_model.LinearRegression()
scores1 = cross_val_score(regr,x,y,cv=5,scoring='neg_root_mean_squared_error')
print('5-Fold CV MSE Scores:', scores1)
```

```
MAE: 1.465311225846298
MSE: 6.678574848138799
RMSE: 2.5842938780523395
5-Fold CV MSE Scores: [-1.86996997 -1.6764068  -1.7728827  -1.87774835 -1.62654
767]
```

Model 2

In [14]:
```
x = bav[['xba']]
y = bav[['xobp','launch_angle_avg','xwoba','xslg','sweet_spot_percent','z_swing_m
regr = LinearRegression()
model = regr.fit(x,y)
regr.coef_
regr.intercept_
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3, random_st
regr = LinearRegression()
regr.fit(x_train,y_train)
y_pred=regr.predict(x_test)
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)))
regr = linear_model.LinearRegression()
scores2 = cross_val_score(regr,x,y,cv=5,scoring='neg_root_mean_squared_error')
print('5-Fold CV MSE Scores:', scores2)
```

```
MAE: 1.630321849258289
MSE: 7.753448825110508
RMSE: 2.784501539793165
5-Fold CV MSE Scores: [-2.09679461 -1.86495221 -1.98801855 -2.10914146 -1.81654
649]
```

Model 3

In [15]:
```
x = bav[['xba']]
y = bav[['xobp','launch_angle_avg','xslg','sweet_spot_percent','z_swing_miss_perc
regr = LinearRegression()
model = regr.fit(x,y)
regr.coef_
regr.intercept_
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3, random_st
regr = LinearRegression()
regr.fit(x_train,y_train)
y_pred=regr.predict(x_test)
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)))
regr = linear_model.LinearRegression()
scores3 = cross_val_score(regr,x,y,cv=5,scoring='neg_root_mean_squared_error')
print('5-Fold CV MSE Scores:', scores3)
```

```
MAE: 1.9526451667917637
MSE: 9.30403726985567
RMSE: 3.050252001041171
5-Fold CV MSE Scores: [-2.5119634  -2.23425946 -2.38152879 -2.52553262 -2.17561
205]
```

Model 4

In [16]:
```
x = bav[['xba']]
y = bav[['xobp','launch_angle_avg','xslg','sweet_spot_percent','z_swing_miss_perc
regr = LinearRegression()
model = regr.fit(x,y)
regr.coef_
regr.intercept_
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.3, random_st
regr = LinearRegression()
regr.fit(x_train,y_train)
y_pred=regr.predict(x_test)
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)))
regr = linear_model.LinearRegression()
scores4 = cross_val_score(regr,x,y,cv=5,scoring='neg_root_mean_squared_error')
print('5-Fold CV MSE Scores:', scores4)
```

```
MAE: 1.7064122198888618
MSE: 7.791586222597815
RMSE: 2.7913412945388485
5-Fold CV MSE Scores: [-2.17813985 -1.9527386  -2.06495192 -2.18617547 -1.89410
251]
```

Comparison

In [29]:
```
score_tot = scores1, scores2, scores3, scores4
count = 1
for score in score_tot:
    mean = abs(score.mean())
    print("Model " + str(count) + ": " + str(mean))
    count += 1
```

```
Model 1: 1.7647110976368272
Model 2: 1.9750906645598143
Model 3: 2.3657792630980734
Model 4: 2.055221668598212
```

Model pre-selection

xobp and *xslg* have high multicollinearity with *xwoba* as can be seen with the VIF. When we remove *xwoba*, the VIF is below the treshold of 4 for all the variables, which helps to mitigate multicollinearity. Therefore, just for the purpose of the rest of our analysis, **we decided to consider only model 3 and 4**.

Testing interaction

Interaction: xslg - sweet spot percent

In [54]:
```
result3_int = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot_
result3_int.summary()
```

Out[54]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.814
Model:	OLS	**Adj. R-squared:**	0.805
Method:	Least Squares	**F-statistic:**	86.73
Date:	Sun, 13 Nov 2022	**Prob (F-statistic):**	4.42e-41
Time:	22:22:07	**Log-Likelihood:**	416.57
No. Observations:	126	**AIC:**	-819.1
Df Residuals:	119	**BIC:**	-799.3
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.0397	0.059	0.673	0.502	-0.077	0.157
xobp	0.1867	0.043	4.370	0.000	0.102	0.271
launch_angle_avg	-0.0021	0.000	-10.130	0.000	-0.003	-0.002
xslg	0.3639	0.143	2.548	0.012	0.081	0.647
sweet_spot_percent	0.0037	0.002	2.188	0.031	0.000	0.007
xslg:sweet_spot_percent	-0.0050	0.004	-1.276	0.204	-0.013	0.003
z_swing_miss_percent	-0.0015	0.000	-6.650	0.000	-0.002	-0.001

Omnibus:	3.690	**Durbin-Watson:**	2.167
Prob(Omnibus):	0.158	**Jarque-Bera (JB):**	2.402
Skew:	-0.136	**Prob(JB):**	0.301
Kurtosis:	2.381	**Cond. No.**	8.07e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.07e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In [49]:
```
# MODEL 3 WITH INTERACTION TERM
y, X5 = pt.dmatrices('xba ~ xobp + launch_angle_avg + xslg + sweet_spot_percent +
                     data=bav, return_type='dataframe')

K = X5.shape[1]
VIF = np.empty(K)
for i in range(K):
    VIF[i] = smo.variance_inflation_factor(X5.values, i)
print(f'VIF: \n{VIF}\n')
```

```
VIF:
[5.27404722e+03 2.08965423e+00 1.26749131e+00 9.84473516e+01
 4.69482231e+01 1.92615798e+02 1.63481329e+00]
```

In [47]:
```
result4_int = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot_
result4_int.summary()
```

Out[47]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.816
Model:	OLS	**Adj. R-squared:**	0.805
Method:	Least Squares	**F-statistic:**	74.76
Date:	Sun, 13 Nov 2022	**Prob (F-statistic):**	2.22e-40
Time:	22:19:10	**Log-Likelihood:**	417.29
No. Observations:	126	**AIC:**	-818.6
Df Residuals:	118	**BIC:**	-795.9
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.0263	0.060	0.438	0.662	-0.093	0.145
xobp	0.1940	0.043	4.499	0.000	0.109	0.279
launch_angle_avg	-0.0021	0.000	-9.958	0.000	-0.003	-0.002
xslg	0.3878	0.144	2.691	0.008	0.102	0.673
sweet_spot_percent	0.0039	0.002	2.326	0.022	0.001	0.007
xslg:sweet_spot_percent	-0.0056	0.004	-1.418	0.159	-0.014	0.002
z_swing_miss_percent	-0.0015	0.000	-6.760	0.000	-0.002	-0.001
AverageSpeed	0.0020	0.002	1.162	0.247	-0.001	0.006

Omnibus:	3.727	**Durbin-Watson:**	2.160
Prob(Omnibus):	0.155	**Jarque-Bera (JB):**	2.385
Skew:	-0.126	**Prob(JB):**	0.304
Kurtosis:	2.375	**Cond. No.**	8.16e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 8.16e+03. This might indicate that there are strong multicollinearity or other numerical problems.

In [98]:
```
# MODEL 4 WITH INTERACTION TERM
y, X6 = pt.dmatrices('xba ~ xobp + launch_angle_avg + xslg + sweet_spot_percent +
                     data=bav, return_type='dataframe')

K = X6.shape[1]
VIF = np.empty(K)
for i in range(K):
    VIF[i] = smo.variance_inflation_factor(X6.values, i)
print(f'VIF: \n{VIF}\n')
```

```
VIF:
[5.47670188e+03 2.13477290e+00 1.28271121e+00 1.00490814e+02
 4.77891341e+01 1.95910629e+02 1.67345312e+00 1.17266229e+00]
```

Interaction: launch angle average - sweet spot percent

In [100]:
```
result3_int2 = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot
result3_int2.summary()
```

Out[100]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.812
Model:	OLS	**Adj. R-squared:**	0.803
Method:	Least Squares	**F-statistic:**	85.79
Date:	Sun, 13 Nov 2022	**Prob (F-statistic):**	7.48e-41
Time:	18:25:32	**Log-Likelihood:**	416.01
No. Observations:	126	**AIC:**	-818.0
Df Residuals:	119	**BIC:**	-798.2
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.0924	0.031	2.995	0.003	0.031	0.153
xobp	0.1881	0.043	4.386	0.000	0.103	0.273
launch_angle_avg	-0.0005	0.002	-0.213	0.832	-0.005	0.004
xslg	0.1841	0.023	8.012	0.000	0.139	0.230
sweet_spot_percent	0.0022	0.001	2.452	0.016	0.000	0.004
sweet_spot_percent:launch_angle_avg	-4.889e-05	6.54e-05	-0.747	0.456	-0.000	8.06e-05
z_swing_miss_percent	-0.0014	0.000	-6.519	0.000	-0.002	-0.001

Omnibus:	5.625	**Durbin-Watson:**	2.108
Prob(Omnibus):	0.060	**Jarque-Bera (JB):**	3.141
Skew:	-0.159	**Prob(JB):**	0.208
Kurtosis:	2.294	**Cond. No.**	2.82e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.82e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In [104]:
```
# MODEL 3 WITH INTERACTION TERM 2
y, X7 = pt.dmatrices('xba ~ xobp + launch_angle_avg + xslg + sweet_spot_percent +
                     data=bav, return_type='dataframe')

K = X7.shape[1]
VIF = np.empty(K)
for i in range(K):
    VIF[i] = smo.variance_inflation_factor(X7.values, i)
print(f'VIF: \n{VIF}\n')
```

```
VIF:
[1426.30493831    2.08769414  137.10159501    2.52699498   13.26123586
  176.07050956    1.61559127]
```

In [105]:
```
result4_int2 = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot
result4_int2.summary()
```

Out[105]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.814
Model:	OLS	**Adj. R-squared:**	0.803
Method:	Least Squares	**F-statistic:**	73.60
Date:	Sun, 13 Nov 2022	**Prob (F-statistic):**	4.66e-40
Time:	18:30:25	**Log-Likelihood:**	416.48
No. Observations:	126	**AIC:**	-817.0
Df Residuals:	118	**BIC:**	-794.3
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.0898	0.031	2.899	0.004	0.028	0.151
xobp	0.1942	0.043	4.476	0.000	0.108	0.280
launch_angle_avg	-0.0005	0.002	-0.246	0.806	-0.005	0.004
xslg	0.1861	0.023	8.062	0.000	0.140	0.232
sweet_spot_percent	0.0022	0.001	2.416	0.017	0.000	0.004
sweet_spot_percent:launch_angle_avg	-4.599e-05	6.55e-05	-0.702	0.484	-0.000	8.37e-05
z_swing_miss_percent	-0.0015	0.000	-6.585	0.000	-0.002	-0.001
AverageSpeed	0.0017	0.002	0.946	0.346	-0.002	0.005

Omnibus:	5.750	**Durbin-Watson:**	2.102
Prob(Omnibus):	0.056	**Jarque-Bera (JB):**	3.176
Skew:	-0.158	**Prob(JB):**	0.204
Kurtosis:	2.289	**Cond. No.**	2.84e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 2.84e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In [106]:
```
# MODEL 4 WITH INTERACTION TERM 2
y, X8 = pt.dmatrices('xba ~ xobp + launch_angle_avg + xslg + sweet_spot_percent +
                     data=bav, return_type='dataframe')

K = X8.shape[1]
VIF = np.empty(K)
for i in range(K):
    VIF[i] = smo.variance_inflation_factor(X8.values, i)
print(f'VIF: \n{VIF}\n')
```

```
VIF:
[1.43757268e+03 2.13503718e+00 1.37267785e+02 2.54851698e+00
 1.32775551e+01 1.76455896e+02 1.64702906e+00 1.15546400e+00]
```

Interaction: xslg - xobp

In [107]:
```
result3_int3 = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot
result3_int3.summary()
```

Out[107]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.813
Model:	OLS	**Adj. R-squared:**	0.803
Method:	Least Squares	**F-statistic:**	86.19
Date:	Sun, 13 Nov 2022	**Prob (F-statistic):**	5.99e-41
Time:	18:38:53	**Log-Likelihood:**	416.25
No. Observations:	126	**AIC:**	-818.5
Df Residuals:	119	**BIC:**	-798.6
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.0662	0.049	1.365	0.175	-0.030	0.162
xobp	0.3254	0.143	2.279	0.024	0.043	0.608
launch_angle_avg	-0.0021	0.000	-10.065	0.000	-0.003	-0.002
xslg	0.3020	0.120	2.525	0.013	0.065	0.539
sweet_spot_percent	0.0016	0.000	5.451	0.000	0.001	0.002
xobp:xslg	-0.3362	0.334	-1.006	0.317	-0.998	0.326
z_swing_miss_percent	-0.0015	0.000	-6.600	0.000	-0.002	-0.001

Omnibus:	4.376	**Durbin-Watson:**	2.158
Prob(Omnibus):	0.112	**Jarque-Bera (JB):**	2.741
Skew:	-0.160	**Prob(JB):**	0.254
Kurtosis:	2.352	**Cond. No.**	1.89e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.89e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In [108]:
```
# MODEL 3 WITH INTERACTION TERM 3
y, X9 = pt.dmatrices('xba ~ xobp + launch_angle_avg + xslg + xobp*xslg + sweet_sp
                     data=bav, return_type='dataframe')

K = X9.shape[1]
VIF = np.empty(K)
for i in range(K):
    VIF[i] = smo.variance_inflation_factor(X9.values, i)
print(f'VIF: \n{VIF}\n')
```

```
VIF:
[3.54489827e+03 2.32176553e+01 1.25754289e+00 6.87155456e+01
 1.28930252e+02 1.35785345e+00 1.68803332e+00]
```

In [110]:
```
result4_int2 = smf.ols(formula='xba ~ xobp + launch_angle_avg + xslg + sweet_spot
result4_int2.summary()
```

Out[110]:

OLS Regression Results

Dep. Variable:	xba	**R-squared:**	0.815
Model:	OLS	**Adj. R-squared:**	0.804
Method:	Least Squares	**F-statistic:**	74.04
Date:	Sun, 13 Nov 2022	**Prob (F-statistic):**	3.51e-40
Time:	18:41:14	**Log-Likelihood:**	416.79
No. Observations:	126	**AIC:**	-817.6
Df Residuals:	118	**BIC:**	-794.9
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	0.0608	0.049	1.246	0.215	-0.036	0.158
xobp	0.3358	0.143	2.346	0.021	0.052	0.619
launch_angle_avg	-0.0021	0.000	-9.872	0.000	-0.002	-0.002
xslg	0.3076	0.120	2.569	0.011	0.070	0.545
sweet_spot_percent	0.0016	0.000	5.478	0.000	0.001	0.002
xobp:xslg	-0.3459	0.334	-1.034	0.303	-1.008	0.316
z_swing_miss_percent	-0.0015	0.000	-6.678	0.000	-0.002	-0.001
AverageSpeed	0.0018	0.002	1.012	0.314	-0.002	0.005

Omnibus:	4.529	**Durbin-Watson:**	2.151
Prob(Omnibus):	0.104	**Jarque-Bera (JB):**	2.778
Skew:	-0.156	**Prob(JB):**	0.249
Kurtosis:	2.343	**Cond. No.**	1.89e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.89e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In [109]:
```
# MODEL 4 WITH INTERACTION TERM 3
y, X10 = pt.dmatrices('xba ~ xobp + launch_angle_avg + xslg + xobp*xslg + sweet_s
                      data=bav, return_type='dataframe')

K = X10.shape[1]
VIF = np.empty(K)
for i in range(K):
    VIF[i] = smo.variance_inflation_factor(X10.values, i)
print(f'VIF: \n{VIF}\n')
```

```
VIF:
[3.58816406e+03 2.33387478e+01 1.27598410e+00 6.88596653e+01
 1.29035333e+02 1.35896832e+00 1.72213979e+00 1.15388010e+00]
```

Robustness

Model 3

In [136]:
```
coefs = pd.DataFrame(columns = ["B0","B1", "B2", "B3", "B4", "B5"])
boot_adjR2 = []
n_boots = 1000
plt.figure()
for _ in range(n_boots):
    sample_bav = bav.sample(bav.shape[0] ,replace = True)
    ols_model_temp = smf.ols(formula = 'xba ~ xobp + launch_angle_avg + xslg + sw
    results_temp = ols_model_temp.fit()
    b0,b1,b2,b3,b4,b5 = results_temp.params
    coefs = coefs.append({"B0":b0, "B1":b1, "B2":b2, "B3":b3, "B4":b4, "B5":b5},
    boot_adjR2.append(results_temp.rsquared_adj)
```

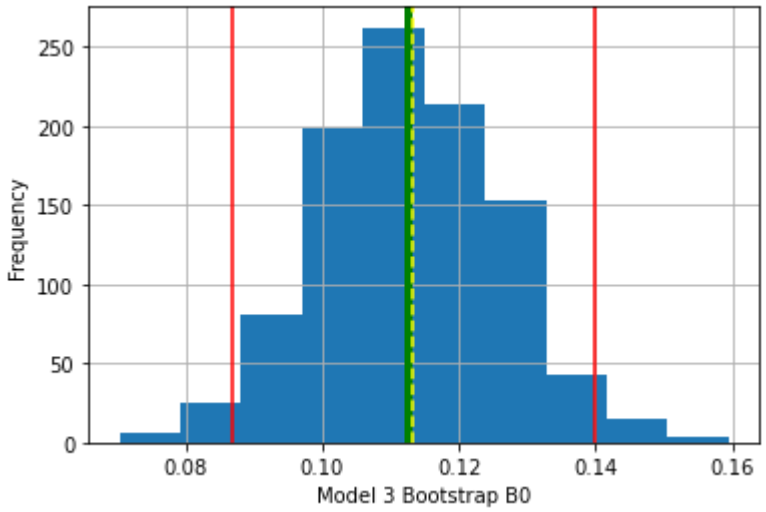
```
<Figure size 432x288 with 0 Axes>
```

In [137]:
```
b0_u = coefs["B0"].quantile(.975)
b0_l = coefs["B0"].quantile(.025)
b1_u = coefs["B1"].quantile(.975)
b1_l = coefs["B1"].quantile(.025)
b2_u = coefs["B2"].quantile(.975)
b2_l = coefs["B2"].quantile(.025)
b3_u = coefs["B3"].quantile(.975)
b3_l = coefs["B3"].quantile(.025)
b4_u = coefs["B4"].quantile(.975)
b4_l = coefs["B4"].quantile(.025)
b5_u = coefs["B5"].quantile(.975)
b5_l = coefs["B5"].quantile(.025)
```

In [138]:
```
a0,a1,a2,a3,a4,a5 = result3.params
```

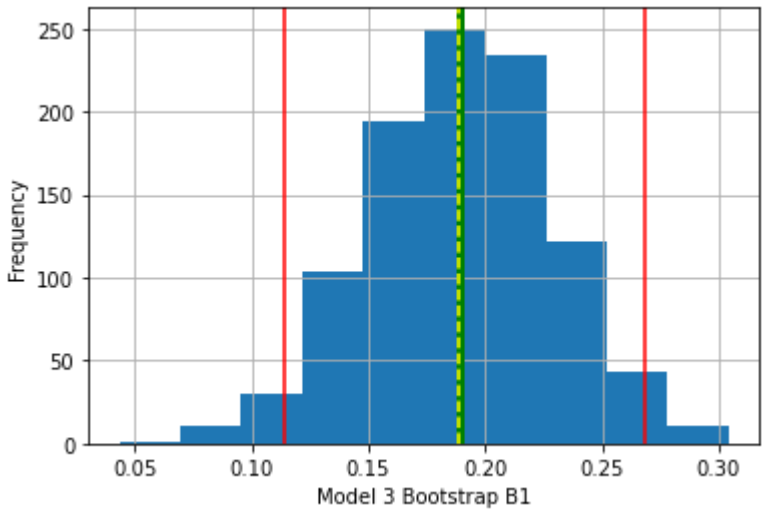
In [139]:
```
coefs.B0.hist()
b0_mean = coefs["B0"].mean()
plt.xlabel("Model 3 Bootstrap B0")
plt.ylabel("Frequency")
plt.axvline(b0_u, color = "red")
plt.axvline(b0_l, color = "red")
plt.axvline(b0_mean, color = "green", linewidth=4)
plt.axvline(a0, color = "yellow", linestyle="--")
```

Out[139]: <matplotlib.lines.Line2D at 0x2399e5c0d30>



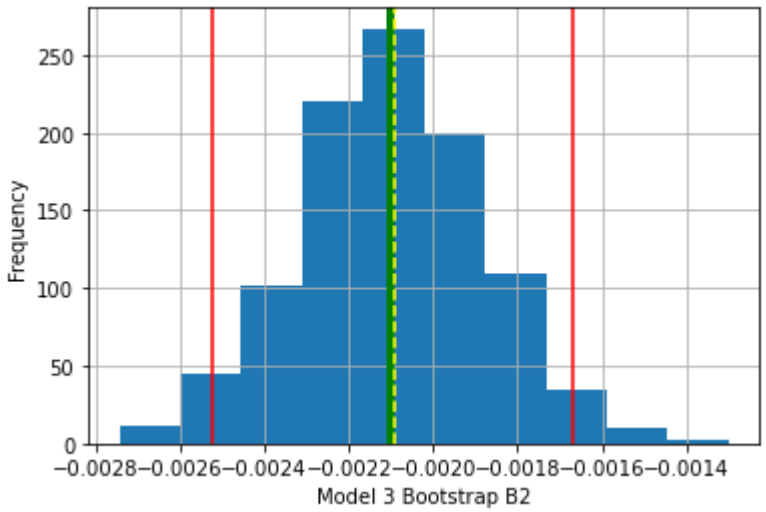
In [140]:
```
coefs.B1.hist()
b1_mean = coefs["B1"].mean()
plt.xlabel("Model 3 Bootstrap B1")
plt.ylabel("Frequency")
plt.axvline(b1_u, color = "red")
plt.axvline(b1_l, color = "red")
plt.axvline(b1_mean, color = "green", linewidth=4)
plt.axvline(a1, color = "yellow", linestyle="--")
```

Out[140]: <matplotlib.lines.Line2D at 0x2399bc7f490>



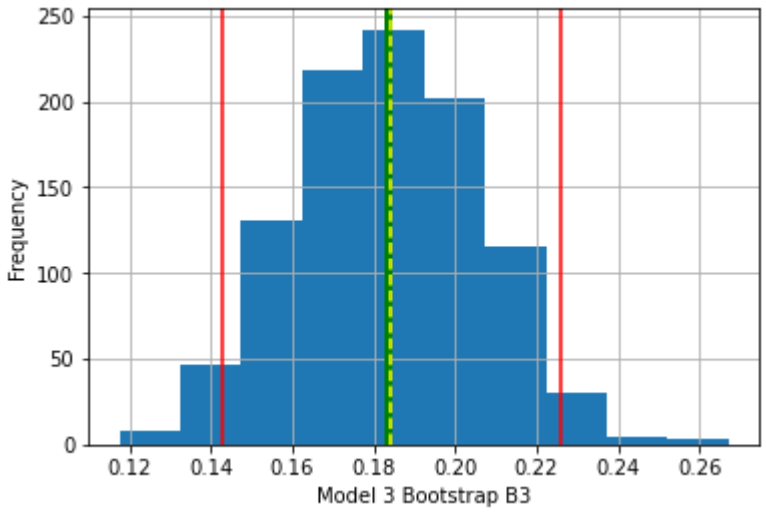
In [141]:
```
coefs.B2.hist()
b2_mean = coefs["B2"].mean()
plt.xlabel("Model 3 Bootstrap B2")
plt.ylabel("Frequency")
plt.axvline(b2_u, color = "red")
plt.axvline(b2_l, color = "red")
plt.axvline(b2_mean, color = "green", linewidth=4)
plt.axvline(a2, color = "yellow", linestyle="--")
```

Out[141]: <matplotlib.lines.Line2D at 0x2399bc5cb80>



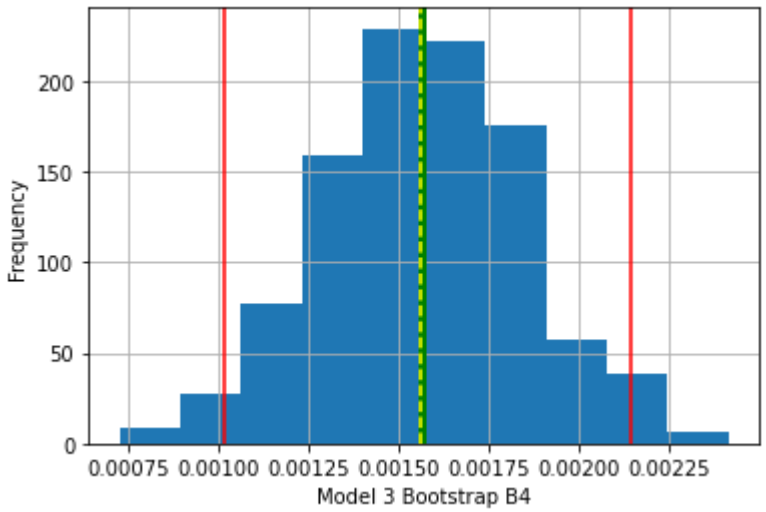
In [142]:
```
coefs.B3.hist()
b3_mean = coefs["B3"].mean()
plt.xlabel("Model 3 Bootstrap B3")
plt.ylabel("Frequency")
plt.axvline(b3_u, color = "red")
plt.axvline(b3_l, color = "red")
plt.axvline(b3_mean, color = "green", linewidth=4)
plt.axvline(a3, color = "yellow", linestyle="--")
```

Out[142]: <matplotlib.lines.Line2D at 0x2399bac1c40>



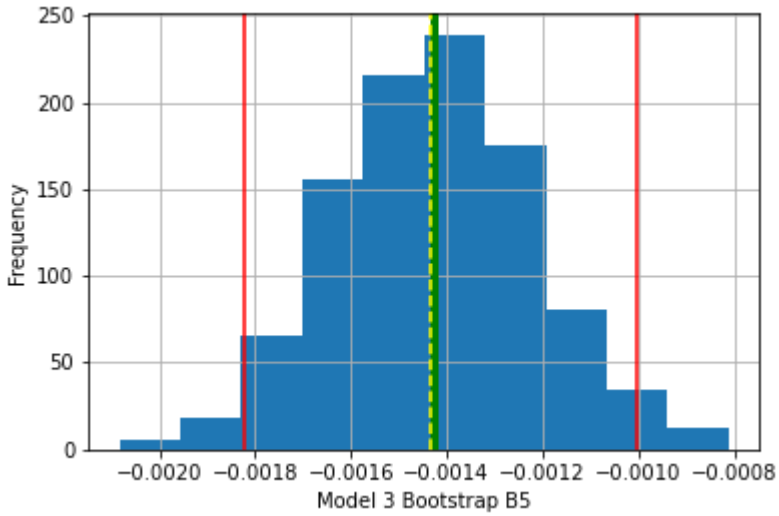
In [143]:
```
coefs.B4.hist()
b4_mean = coefs["B4"].mean()
plt.xlabel("Model 3 Bootstrap B4")
plt.ylabel("Frequency")
plt.axvline(b4_u, color = "red")
plt.axvline(b4_l, color = "red")
plt.axvline(b4_mean, color = "green", linewidth=4)
plt.axvline(a4, color = "yellow", linestyle="--")
```

Out[143]: <matplotlib.lines.Line2D at 0x2399b9b1550>



In [144]:
```
coefs.B5.hist()
b5_mean = coefs["B5"].mean()
plt.xlabel("Model 3 Bootstrap B5")
plt.ylabel("Frequency")
plt.axvline(b5_u, color = "red")
plt.axvline(b5_l, color = "red")
plt.axvline(b5_mean, color = "green", linewidth=4)
plt.axvline(a5, color = "yellow", linestyle="--")
```

Out[144]: <matplotlib.lines.Line2D at 0x2399b862df0>



Model 4

In [145]:
```
coefs4 = pd.DataFrame(columns = ["B0","B1", "B2", "B3", "B4", "B5", "B6"])
boot_adjR2 = []
n_boots = 1000
plt.figure()
for _ in range(n_boots):
    sample_bav = bav.sample(bav.shape[0] ,replace = True)
    ols_model_temp = smf.ols(formula = 'xba ~ xobp + launch_angle_avg + xslg + sw
    results_temp = ols_model_temp.fit()
    b0,b1,b2,b3,b4,b5, b6 = results_temp.params
    coefs4 = coefs4.append({"B0":b0, "B1":b1, "B2":b2, "B3":b3, "B4":b4, "B5":b5,
    boot_adjR2.append(results_temp.rsquared_adj)
```

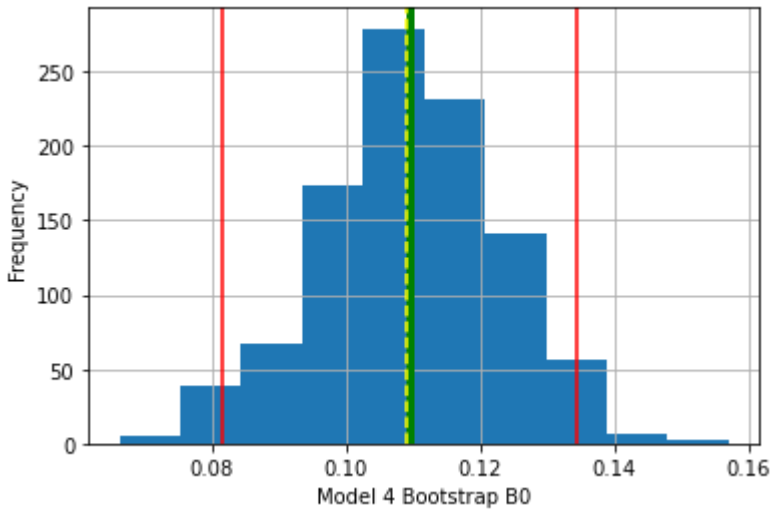
```
<Figure size 432x288 with 0 Axes>
```

In [146]:
```
b0_u = coefs4["B0"].quantile(.975)
b0_l = coefs4["B0"].quantile(.025)
b1_u = coefs4["B1"].quantile(.975)
b1_l = coefs4["B1"].quantile(.025)
b2_u = coefs4["B2"].quantile(.975)
b2_l = coefs4["B2"].quantile(.025)
b3_u = coefs4["B3"].quantile(.975)
b3_l = coefs4["B3"].quantile(.025)
b4_u = coefs4["B4"].quantile(.975)
b4_l = coefs4["B4"].quantile(.025)
b5_u = coefs4["B5"].quantile(.975)
b5_l = coefs4["B5"].quantile(.025)
b6_u = coefs4["B6"].quantile(.975)
b6_l = coefs4["B6"].quantile(.025)
```

In [147]:
```
a0,a1,a2,a3,a4,a5,a6 = result4.params
```

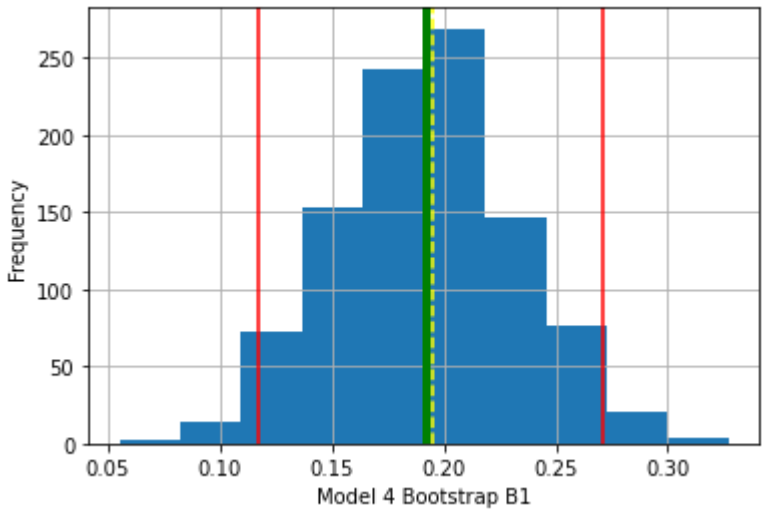
In [148]:
```
coefs4.B0.hist()
b0_mean = coefs4["B0"].mean()
plt.xlabel("Model 4 Bootstrap B0")
plt.ylabel("Frequency")
plt.axvline(b0_u, color = "red")
plt.axvline(b0_l, color = "red")
plt.axvline(b0_mean, color = "green", linewidth=4)
plt.axvline(a0, color = "yellow", linestyle="--")
```

Out[148]: <matplotlib.lines.Line2D at 0x2399bf38220>



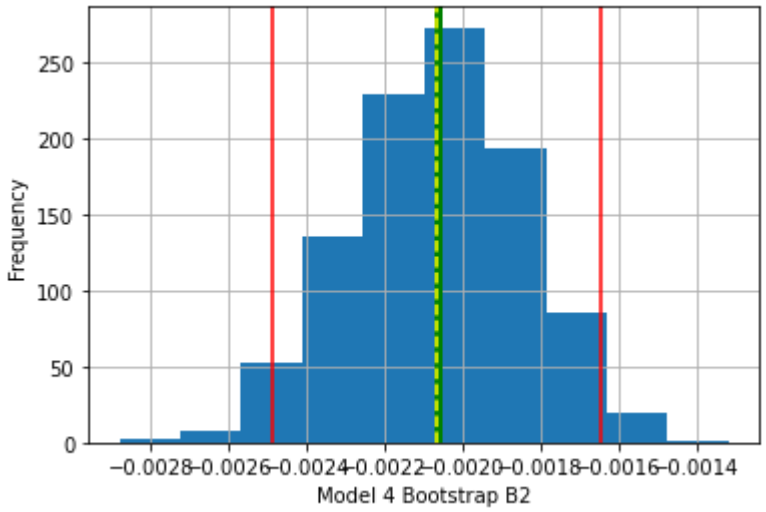
In [149]:
```
coefs4.B1.hist()
b1_mean = coefs4["B1"].mean()
plt.xlabel("Model 4 Bootstrap B1")
plt.ylabel("Frequency")
plt.axvline(b1_u, color = "red")
plt.axvline(b1_l, color = "red")
plt.axvline(b1_mean, color = "green", linewidth=4)
plt.axvline(a1, color = "yellow", linestyle="--")
```

Out[149]: <matplotlib.lines.Line2D at 0x2399b6f60d0>



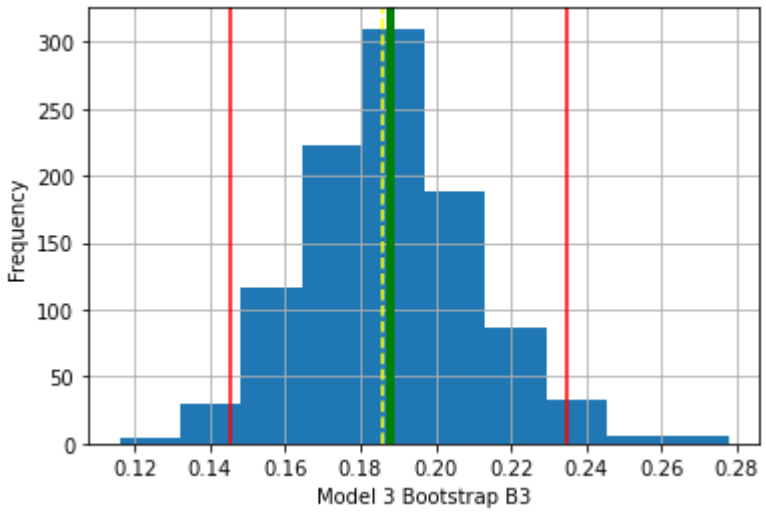
In [151]:
```
coefs4.B2.hist()
b2_mean = coefs4["B2"].mean()
plt.xlabel("Model 4 Bootstrap B2")
plt.ylabel("Frequency")
plt.axvline(b2_u, color = "red")
plt.axvline(b2_l, color = "red")
plt.axvline(b2_mean, color = "green", linewidth=4)
plt.axvline(a2, color = "yellow", linestyle="--")
```

Out[151]: <matplotlib.lines.Line2D at 0x2399bb67eb0>



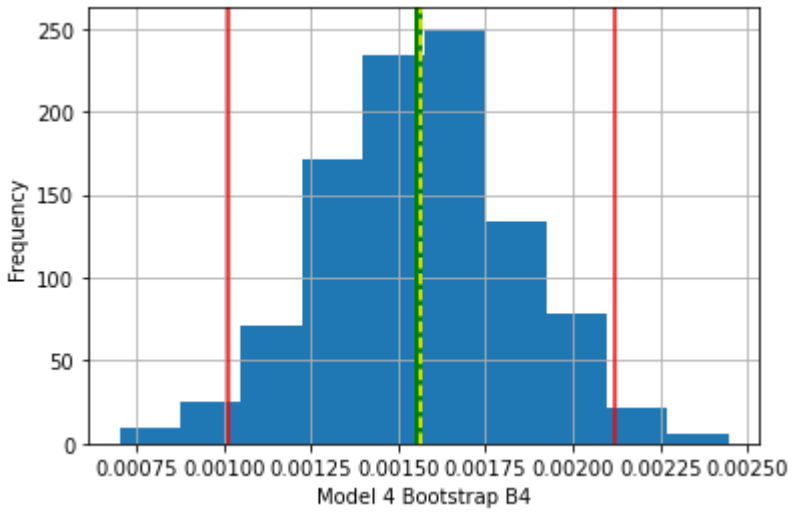
In [152]:
```
coefs4.B3.hist()
b3_mean = coefs4["B3"].mean()
plt.xlabel("Model 3 Bootstrap B3")
plt.ylabel("Frequency")
plt.axvline(b3_u, color = "red")
plt.axvline(b3_l, color = "red")
plt.axvline(b3_mean, color = "green", linewidth=4)
plt.axvline(a3, color = "yellow", linestyle="--")
```

Out[152]: <matplotlib.lines.Line2D at 0x2399bc1f3a0>



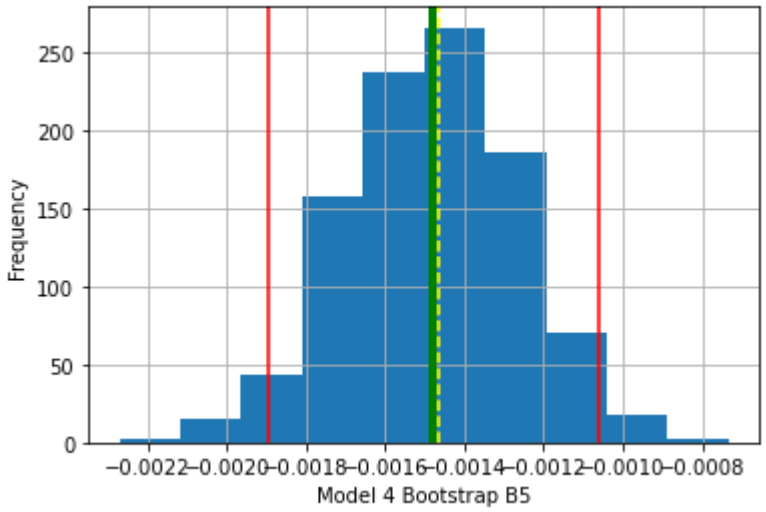
In [153]:
```
coefs4.B4.hist()
b4_mean = coefs4["B4"].mean()
plt.xlabel("Model 4 Bootstrap B4")
plt.ylabel("Frequency")
plt.axvline(b4_u, color = "red")
plt.axvline(b4_l, color = "red")
plt.axvline(b4_mean, color = "green", linewidth=4)
plt.axvline(a4, color = "yellow", linestyle="--")
```

Out[153]: <matplotlib.lines.Line2D at 0x2399b97c3a0>



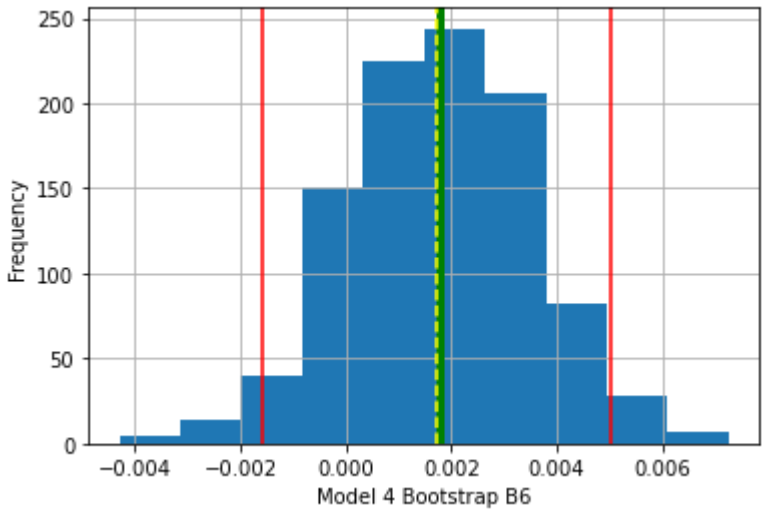
In [154]:
```
coefs4.B5.hist()
b5_mean = coefs4["B5"].mean()
plt.xlabel("Model 4 Bootstrap B5")
plt.ylabel("Frequency")
plt.axvline(b5_u, color = "red")
plt.axvline(b5_l, color = "red")
plt.axvline(b5_mean, color = "green", linewidth=4)
plt.axvline(a5, color = "yellow", linestyle="--")
```

Out[154]: <matplotlib.lines.Line2D at 0x2399ba2c8e0>



In [155]:
```
coefs4.B6.hist()
b6_mean = coefs4["B6"].mean()
plt.xlabel("Model 4 Bootstrap B6")
plt.ylabel("Frequency")
plt.axvline(b6_u, color = "red")
plt.axvline(b6_l, color = "red")
plt.axvline(b6_mean, color = "green", linewidth=4)
plt.axvline(a6, color = "yellow", linestyle="--")
```

Out[155]: <matplotlib.lines.Line2D at 0x2399b87df40>



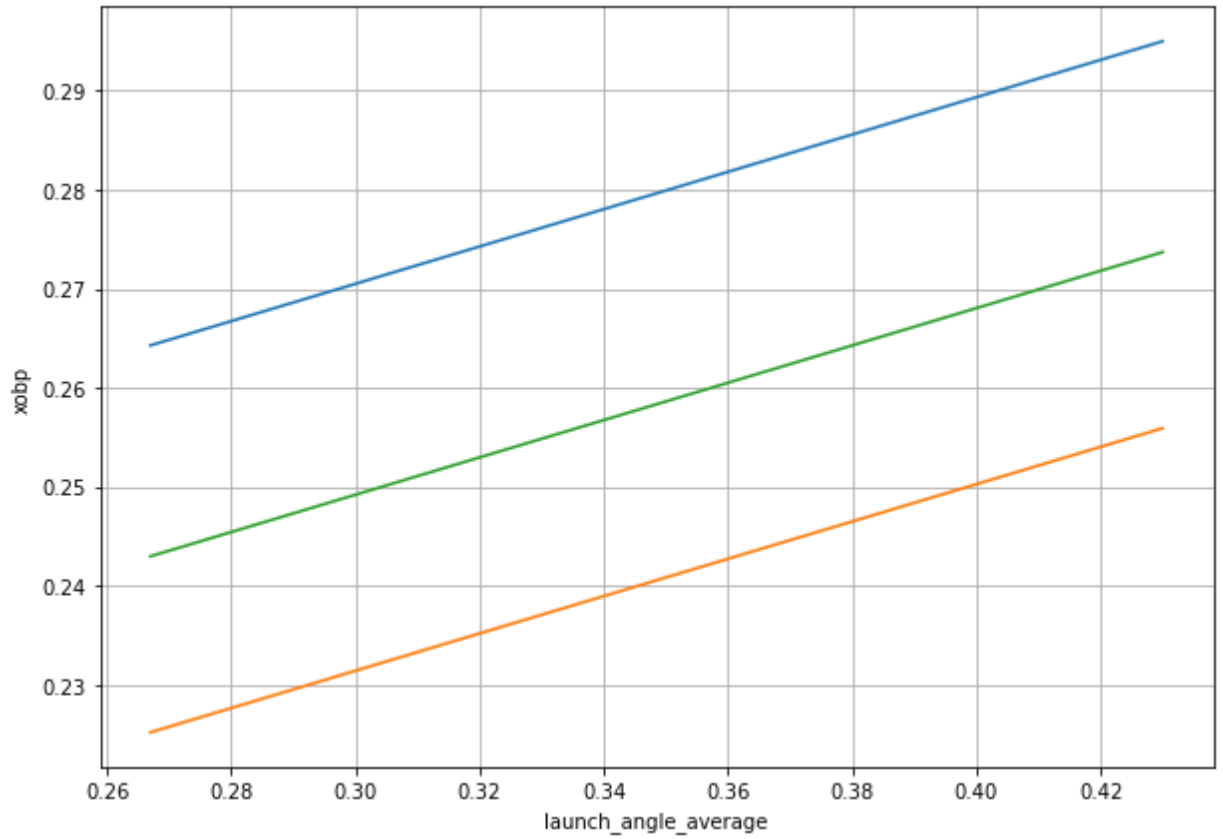
Marginal effects

Model 3

In [69]:
```
xobpME = np.linspace(bav.xobp.min(),bav.xobp.max(), 50)
minlaunch_angle_avg = bav.launch_angle_avg.min()
maxlaunch_angle_avg = bav.launch_angle_avg.max()
meanlaunch_angle_avg = bav.launch_angle_avg.mean()
meanxslg = np.mean(bav.xslg)
meansweet_spot_percent = np.mean(bav.sweet_spot_percent)
meanz_swing_miss_percent = np.mean(bav.z_swing_miss_percent)
```

In [70]:
```
y1 = result3.params[0] +result3.params[1]*xobpME + result3.params[2]*minlaunch_ar
y2 = result3.params[0] +result3.params[1]*xobpME + result3.params[2]*maxlaunch_ar
y3 =  result3.params[0] + result3.params[1]*xobpME+ result3.params[2]*meanlaunch_
```

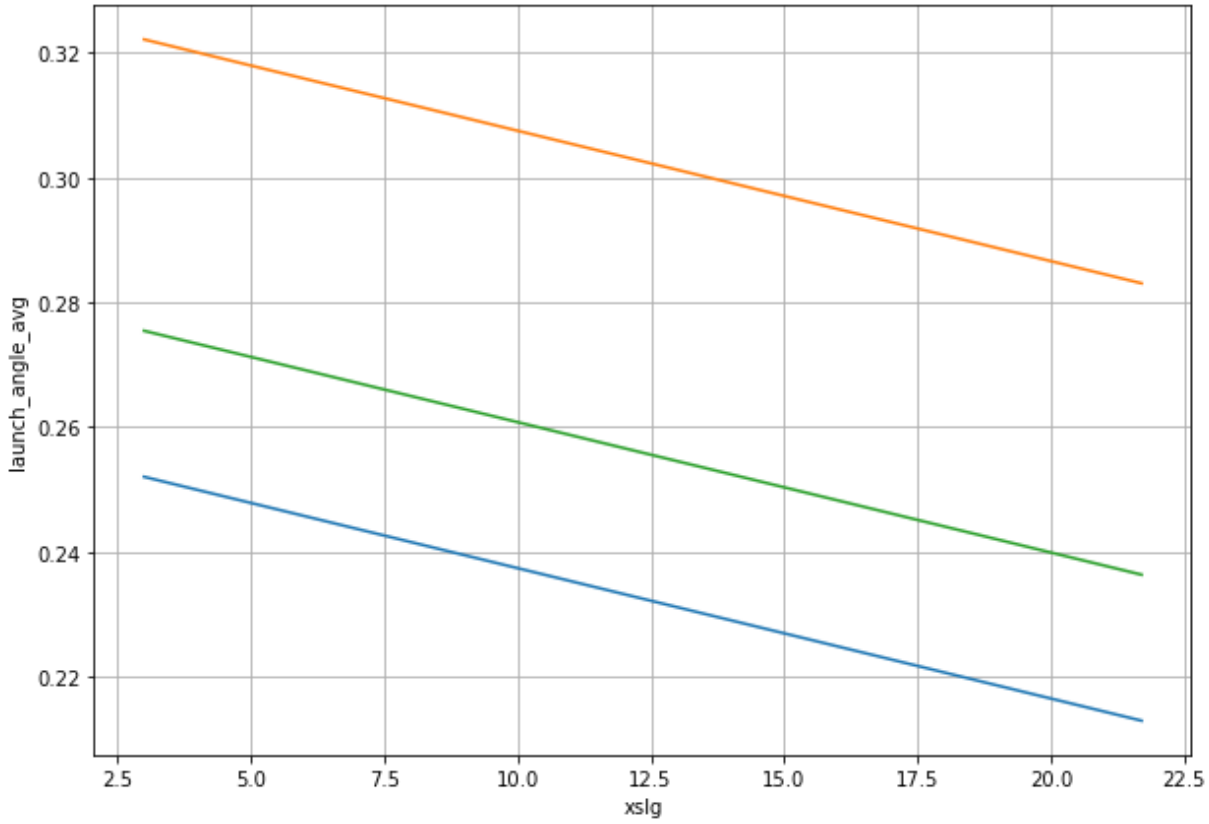
In [71]:
```
plt.figure(figsize = (10, 7))
plt.plot(xobpME, y1)
plt.plot(xobpME, y2)
plt.plot(xobpME, y3)
plt.xlabel("launch_angle_average")
plt.ylabel("xobp")
plt.grid()
```



In [72]:
```
launch_angle_avg_ME = np.linspace(bav.launch_angle_avg.min(),bav.launch_angle_avg
minxslg = bav.xslg.min()
maxxslg = bav.xslg.max()
meanxslg = bav.xslg.mean()
meanxobp = np.mean(bav.xobp)
meansweet_spot_percent = np.mean(bav.sweet_spot_percent)
meanz_swing_miss_percent = np.mean(bav.z_swing_miss_percent)
```

In [73]:
```
y1 = result3.params[0] +result3.params[1]*meanxobp + result3.params[2]*launch_ang
y2 = result3.params[0] +result3.params[1]*meanxobp + result3.params[2]*launch_ang
y3 =  result3.params[0] + result3.params[1]*meanxobp+ result3.params[2]*launch_ar
```

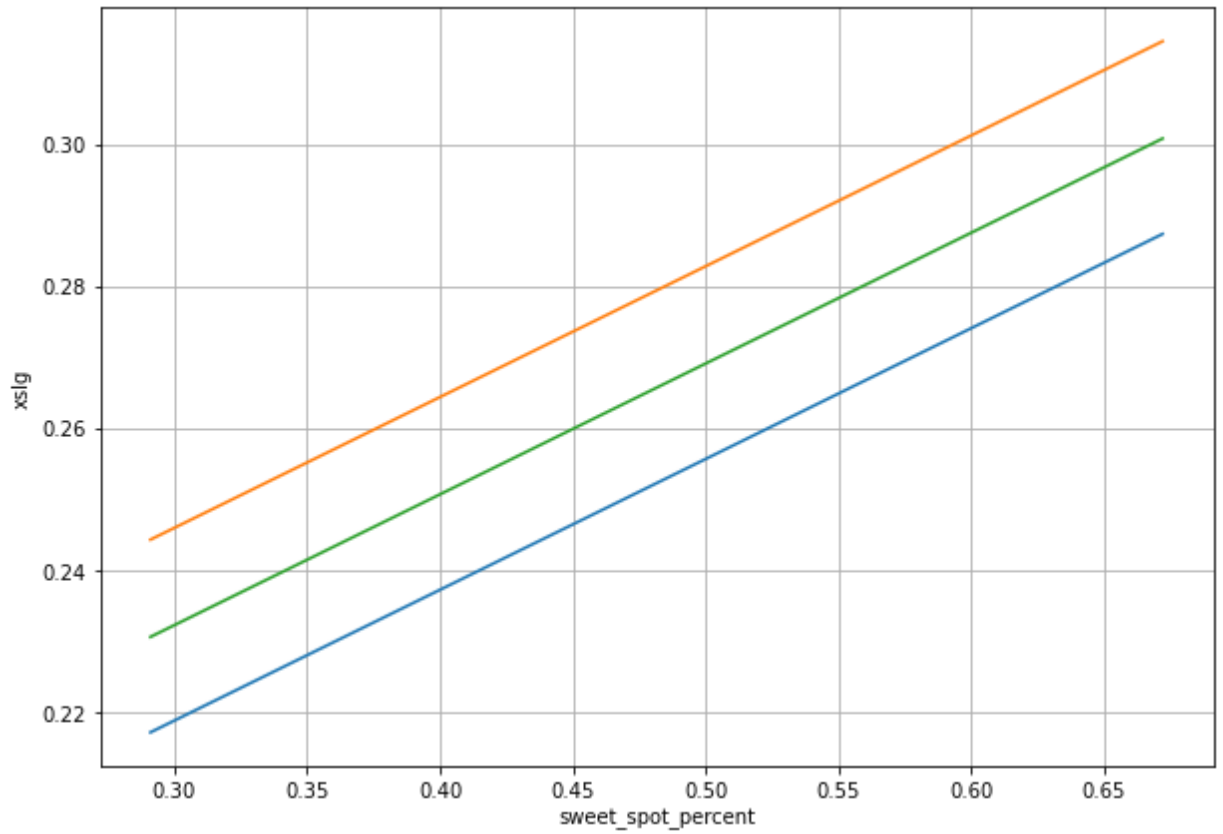
In [74]:
```
plt.figure(figsize = (10, 7))
plt.plot(launch_angle_avg_ME, y1)
plt.plot(launch_angle_avg_ME, y2)
plt.plot(launch_angle_avg_ME, y3)
plt.xlabel("xslg")
plt.ylabel("launch_angle_avg")
plt.grid()
```



In [75]:
```
xslgME = np.linspace(bav.xslg.min(),bav.xslg.max(), 50)
minsweet_spot_percent = bav.sweet_spot_percent.min()
maxsweet_spot_percent = bav.sweet_spot_percent.max()
meansweet_spot_percent = bav.sweet_spot_percent.mean()
meanxobp = np.mean(bav.xobp)
meanlaunch_angle_avg = np.mean(bav.launch_angle_avg)
meanz_swing_miss_percent = np.mean(bav.z_swing_miss_percent)
```

In [76]:
```
y1 = result3.params[0] +result3.params[1]*meanxobp + result3.params[2]*meanlaunch
y2 = result3.params[0] +result3.params[1]*meanxobp + result3.params[2]*meanlaunch
y3 =  result3.params[0] + result3.params[1]*meanxobp+ result3.params[2]*meanlaunc
```

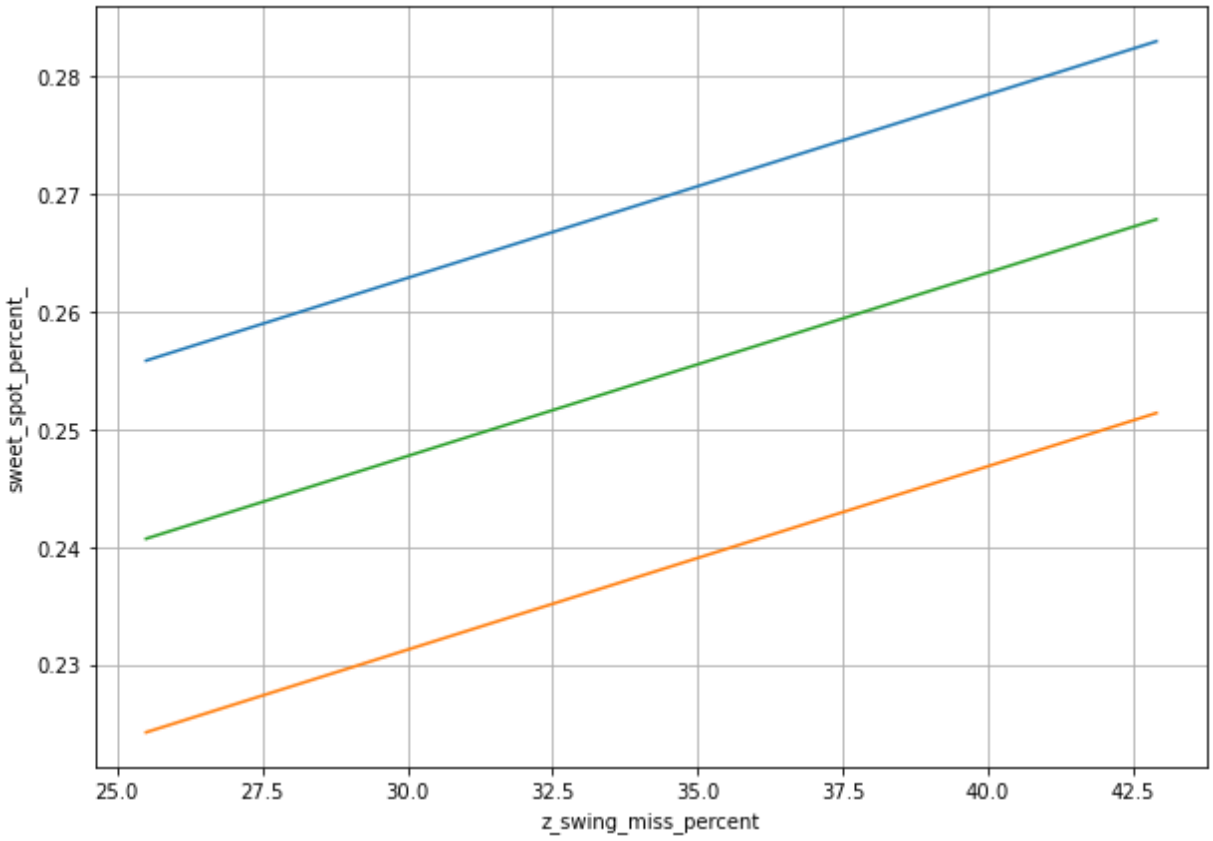
In [77]:
```
plt.figure(figsize = (10, 7))
plt.plot(xslgME, y1)
plt.plot(xslgME, y2)
plt.plot(xslgME, y3)
plt.xlabel("sweet_spot_percent")
plt.ylabel("xslg")
plt.grid()
```



In [78]:
```
sweet_spot_percent_ME = np.linspace(bav.sweet_spot_percent.min(),bav.sweet_spot_p
minz_swing_miss_percent = bav.z_swing_miss_percent.min()
maxz_swing_miss_percent = bav.z_swing_miss_percent.max()
meanz_swing_miss_percent = bav.z_swing_miss_percent.mean()
meanxobp = np.mean(bav.xobp)
meanlaunch_angle_avg = np.mean(bav.launch_angle_avg)
meanxslg = np.mean(bav.xslg)
```

In [79]:
```
y1 = result3.params[0] +result3.params[1]*meanxobp + result3.params[2]*meanlaunch
y2 = result3.params[0] +result3.params[1]*meanxobp + result3.params[2]*meanlaunch
y3 =  result3.params[0] + result3.params[1]*meanxobp+ result3.params[2]*meanlaunc
```

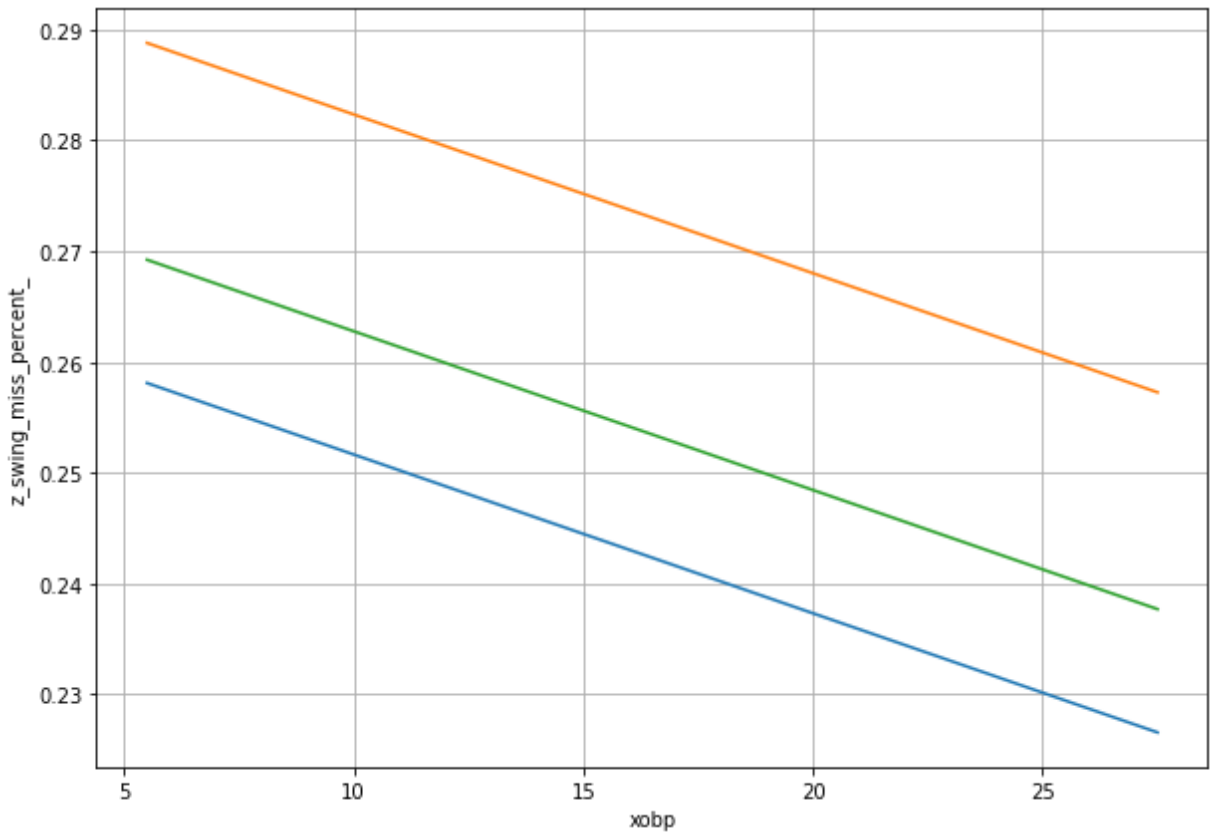
In [80]:
```
plt.figure(figsize = (10, 7))
plt.plot(sweet_spot_percent_ME, y1)
plt.plot(sweet_spot_percent_ME, y2)
plt.plot(sweet_spot_percent_ME, y3)
plt.xlabel("z_swing_miss_percent")
plt.ylabel("sweet_spot_percent_")
plt.grid()
```



In [81]:
```
z_swing_miss_percent_ME = np.linspace(bav.z_swing_miss_percent.min(),bav.z_swing_
minxobp = bav.xobp.min()
maxxobp = bav.xobp.max()
meanxobp = bav.xobp.mean()
meanlaunch_angle_avg = np.mean(bav.launch_angle_avg)
meanxslg = np.mean(bav.xslg)
meansweet_spot_percent = np.mean(bav.sweet_spot_percent)
```

In [82]:
```
y1 = result3.params[0] +result3.params[1]*minxobp + result3.params[2]*meanlaunch_
y2 = result3.params[0] +result3.params[1]*maxxobp + result3.params[2]*meanlaunch_
y3 =  result3.params[0] + result3.params[1]*meanxobp+ result3.params[2]*meanlaunc
```

In [83]:
```
plt.figure(figsize = (10, 7))
plt.plot(z_swing_miss_percent_ME, y1)
plt.plot(z_swing_miss_percent_ME, y2)
plt.plot(z_swing_miss_percent_ME, y3)
plt.xlabel("xobp")
plt.ylabel("z_swing_miss_percent_")
plt.grid()
```

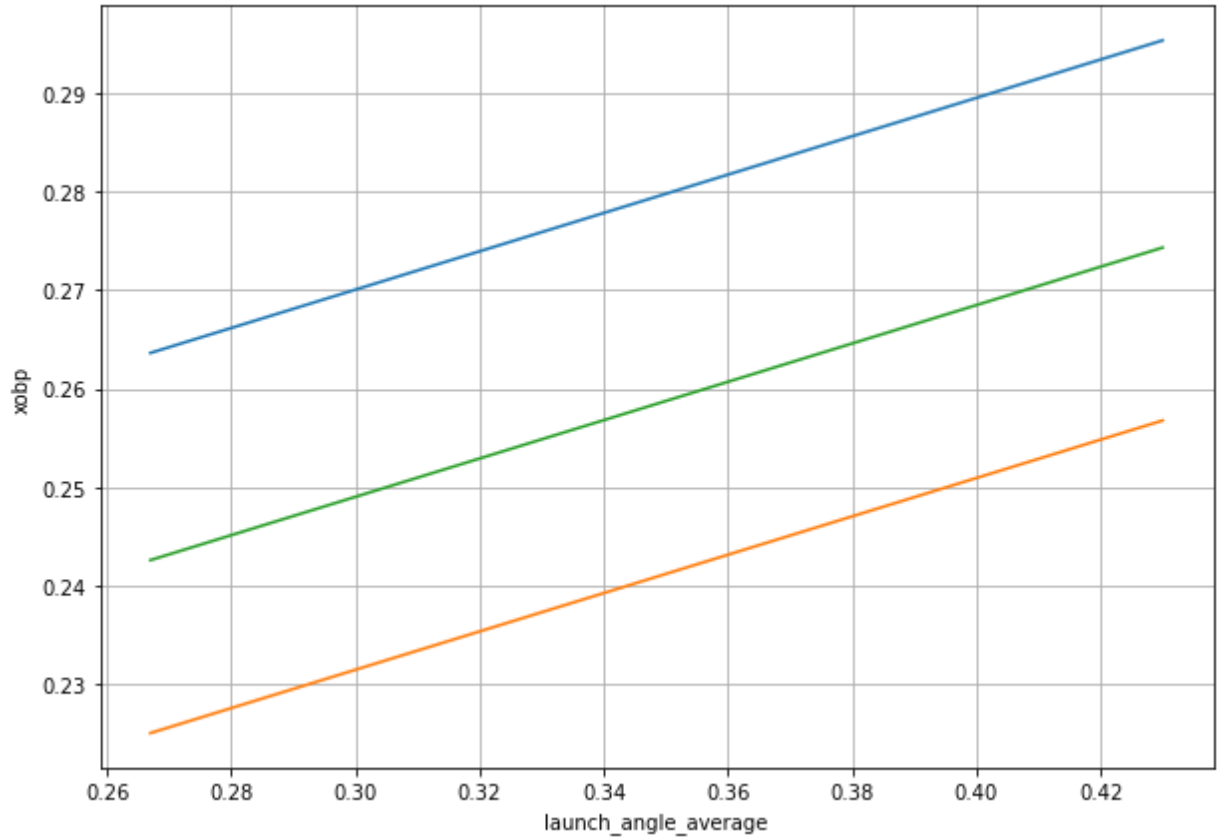


Model 4

In [38]:
```
xobpME = np.linspace(bav.xobp.min(),bav.xobp.max(), 50)
minlaunch_angle_avg = bav.launch_angle_avg.min()
maxlaunch_angle_avg = bav.launch_angle_avg.max()
meanlaunch_angle_avg = bav.launch_angle_avg.mean()
meanxslg = np.mean(bav.xslg)
meansweet_spot_percent = np.mean(bav.sweet_spot_percent)
meanz_swing_miss_percent = np.mean(bav.z_swing_miss_percent)
meanAverageSpeed = np.mean(bav.AverageSpeed)
```

In [39]:
```
y1 = result4.params[0] +result4.params[1]*xobpME + result4.params[2]*minlaunch_ar
y2 = result4.params[0] +result4.params[1]*xobpME + result4.params[2]*maxlaunch_ar
y3 =  result4.params[0] + result4.params[1]*xobpME+ result4.params[2]*meanlaunch_
```

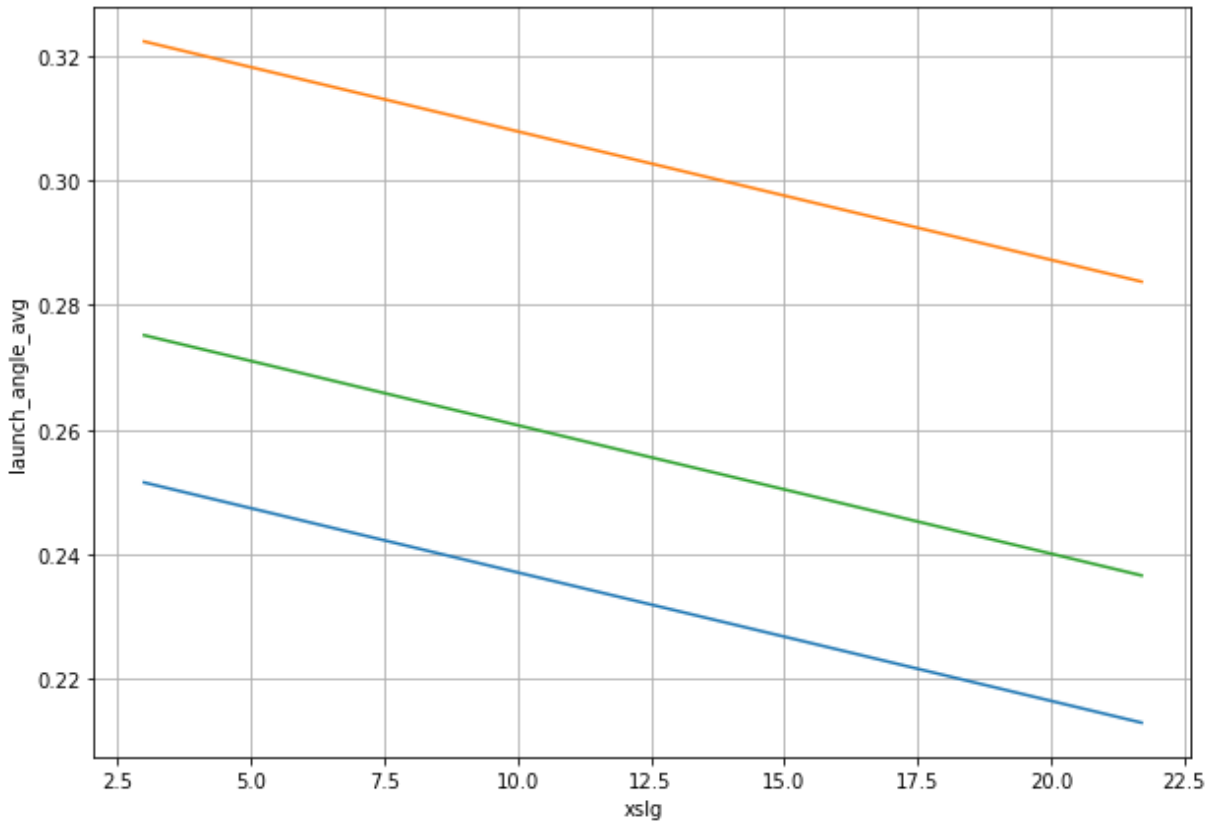
In [40]:
```
plt.figure(figsize = (10, 7))
plt.plot(xobpME, y1)
plt.plot(xobpME, y2)
plt.plot(xobpME, y3)
plt.xlabel("launch_angle_average")
plt.ylabel("xobp")
plt.grid()
```



In [41]:
```
launch_angle_avg_ME = np.linspace(bav.launch_angle_avg.min(),bav.launch_angle_avg
minxslg = bav.xslg.min()
maxxslg = bav.xslg.max()
meanxslg = bav.xslg.mean()
meanxobp = np.mean(bav.xobp)
meansweet_spot_percent = np.mean(bav.sweet_spot_percent)
meanz_swing_miss_percent = np.mean(bav.z_swing_miss_percent)
meanAverageSpeed = np.mean(bav.AverageSpeed)
```

In [42]:
```
y1 = result4.params[0] +result4.params[1]*meanxobp + result4.params[2]*launch_ang
y2 = result4.params[0] +result4.params[1]*meanxobp + result4.params[2]*launch_ang
y3 =  result4.params[0] + result4.params[1]*meanxobp+ result4.params[2]*launch_ar
```

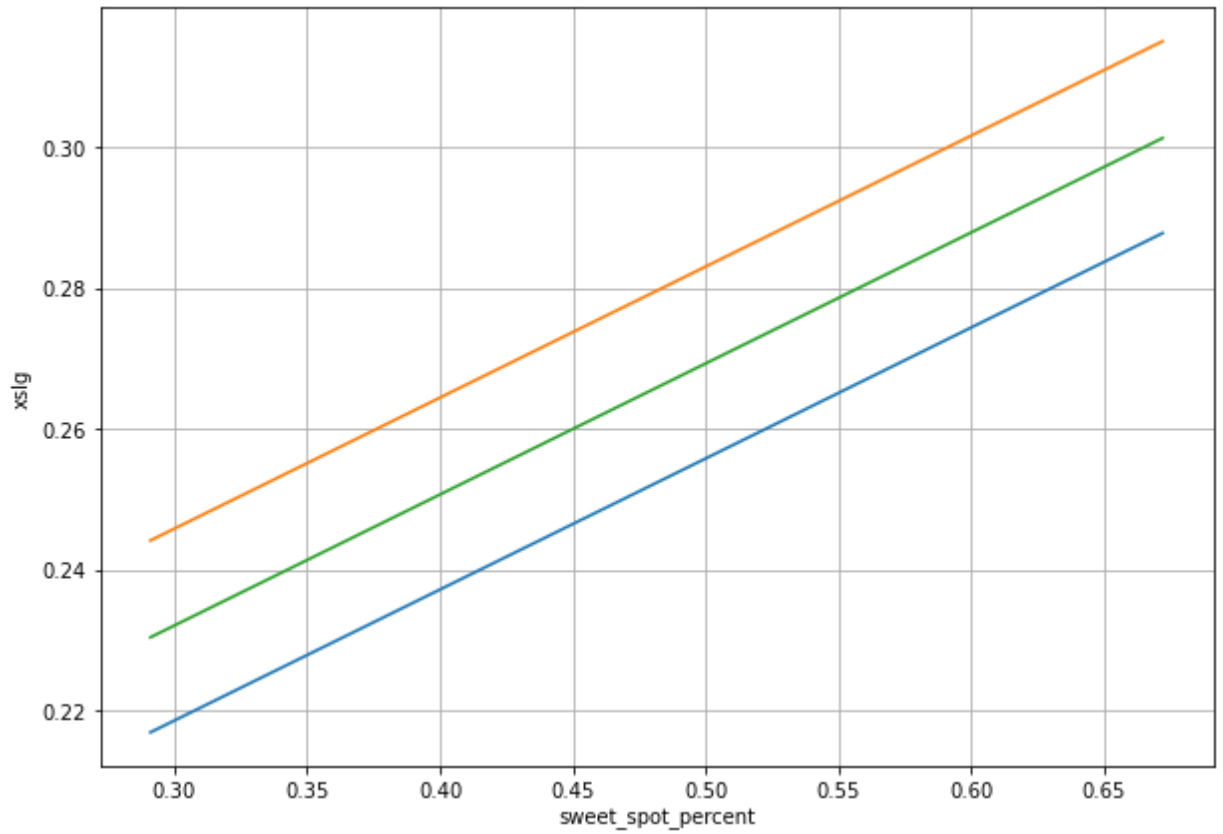
In [43]:
```
plt.figure(figsize = (10, 7))
plt.plot(launch_angle_avg_ME, y1)
plt.plot(launch_angle_avg_ME, y2)
plt.plot(launch_angle_avg_ME, y3)
plt.xlabel("xslg")
plt.ylabel("launch_angle_avg")
plt.grid()
```



In [55]:
```
xslgME = np.linspace(bav.xslg.min(),bav.xslg.max(), 50)
minsweet_spot_percent = bav.sweet_spot_percent.min()
maxsweet_spot_percent = bav.sweet_spot_percent.max()
meansweet_spot_percent = bav.sweet_spot_percent.mean()
meanxobp = np.mean(bav.xobp)
meanlaunch_angle_avg = np.mean(bav.launch_angle_avg)
meanz_swing_miss_percent = np.mean(bav.z_swing_miss_percent)
meanAverageSpeed = np.mean(bav.AverageSpeed)
```

In [56]:
```
y1 = result4.params[0] +result4.params[1]*meanxobp + result4.params[2]*meanlaunch
y2 = result4.params[0] +result4.params[1]*meanxobp + result4.params[2]*meanlaunch
y3 =  result4.params[0] + result4.params[1]*meanxobp+ result4.params[2]*meanlaunc
```

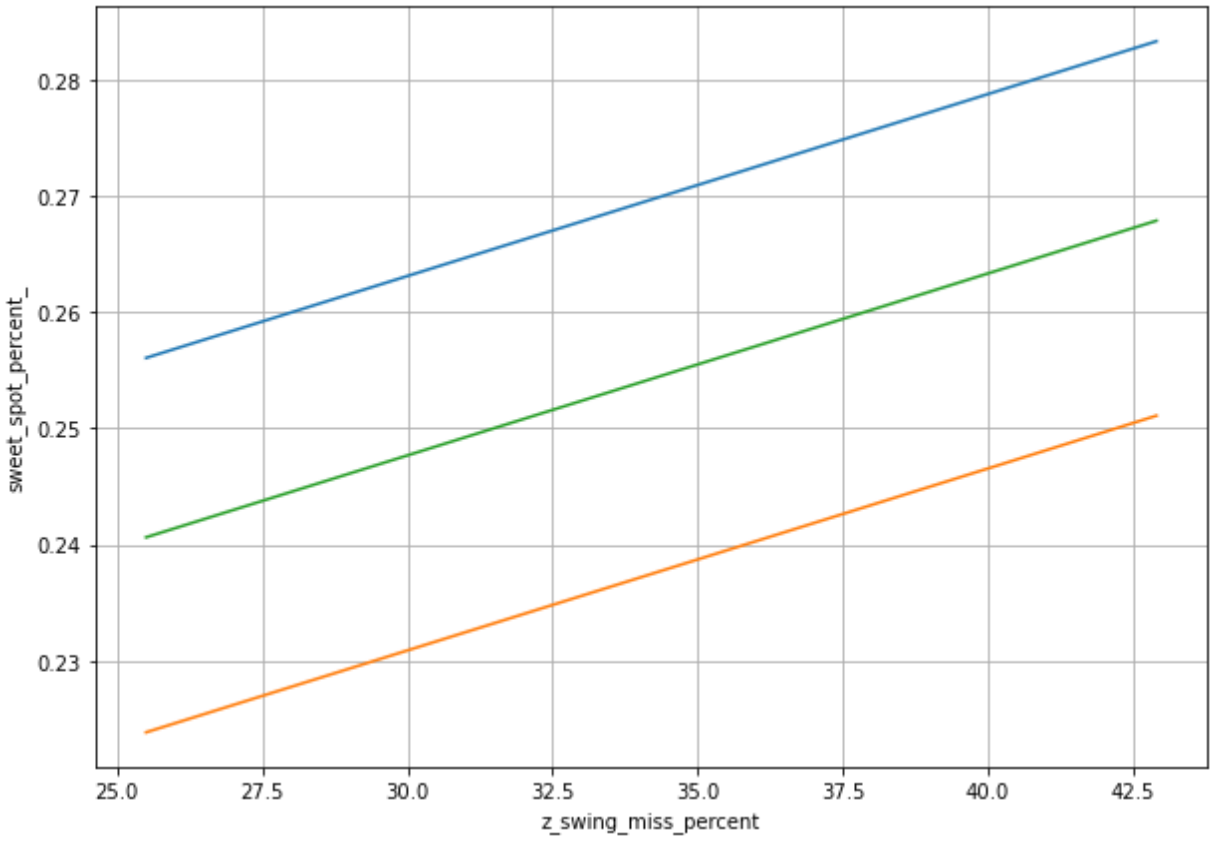
In [57]:
```
plt.figure(figsize = (10, 7))
plt.plot(xslgME, y1)
plt.plot(xslgME, y2)
plt.plot(xslgME, y3)
plt.xlabel("sweet_spot_percent")
plt.ylabel("xslg")
plt.grid()
```



In [60]:
```
sweet_spot_percent_ME = np.linspace(bav.sweet_spot_percent.min(),bav.sweet_spot_p
minz_swing_miss_percent = bav.z_swing_miss_percent.min()
maxz_swing_miss_percent = bav.z_swing_miss_percent.max()
meanz_swing_miss_percent = bav.z_swing_miss_percent.mean()
meanxobp = np.mean(bav.xobp)
meanlaunch_angle_avg = np.mean(bav.launch_angle_avg)
meanxslg = np.mean(bav.xslg)
meanAverageSpeed = np.mean(bav.AverageSpeed)
```

In [61]:
```
y1 = result4.params[0] +result4.params[1]*meanxobp + result4.params[2]*meanlaunch
y2 = result4.params[0] +result4.params[1]*meanxobp + result4.params[2]*meanlaunch
y3 =  result4.params[0] + result4.params[1]*meanxobp+ result4.params[2]*meanlaunc
```

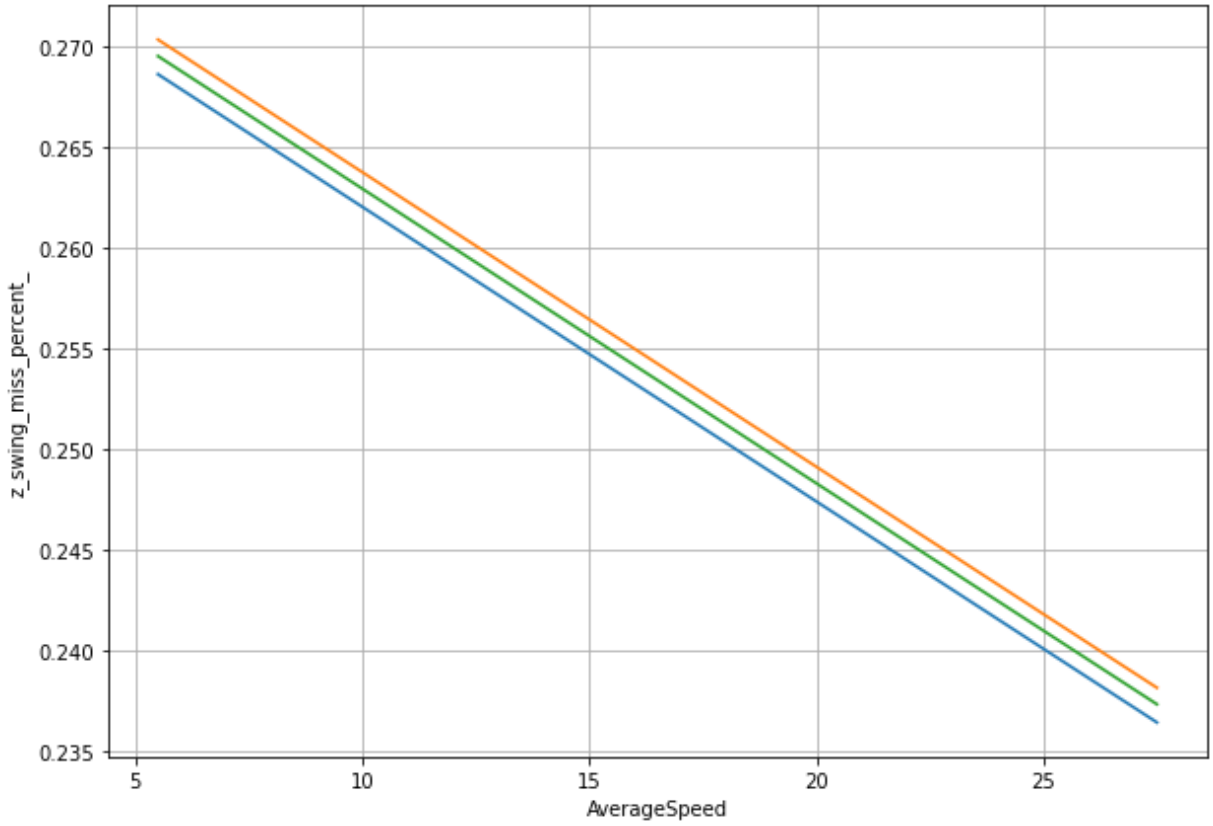
In [62]:
```
plt.figure(figsize = (10, 7))
plt.plot(sweet_spot_percent_ME, y1)
plt.plot(sweet_spot_percent_ME, y2)
plt.plot(sweet_spot_percent_ME, y3)
plt.xlabel("z_swing_miss_percent")
plt.ylabel("sweet_spot_percent_")
plt.grid()
```



In [63]:
```
z_swing_miss_percent_ME = np.linspace(bav.z_swing_miss_percent.min(),bav.z_swing_
minAverageSpeed = bav.AverageSpeed.min()
maxAverageSpeed = bav.AverageSpeed.max()
meanAverageSpeed = bav.AverageSpeed.mean()
meanxobp = np.mean(bav.xobp)
meanlaunch_angle_avg = np.mean(bav.launch_angle_avg)
meanxslg = np.mean(bav.xslg)
meansweet_spot_percent = np.mean(bav.sweet_spot_percent)
```

In [64]:
```
y1 = result4.params[0] +result4.params[1]*meanxobp + result4.params[2]*meanlaunch
y2 = result4.params[0] +result4.params[1]*meanxobp + result4.params[2]*meanlaunch
y3 =  result4.params[0] + result4.params[1]*meanxobp+ result4.params[2]*meanlaunc
```

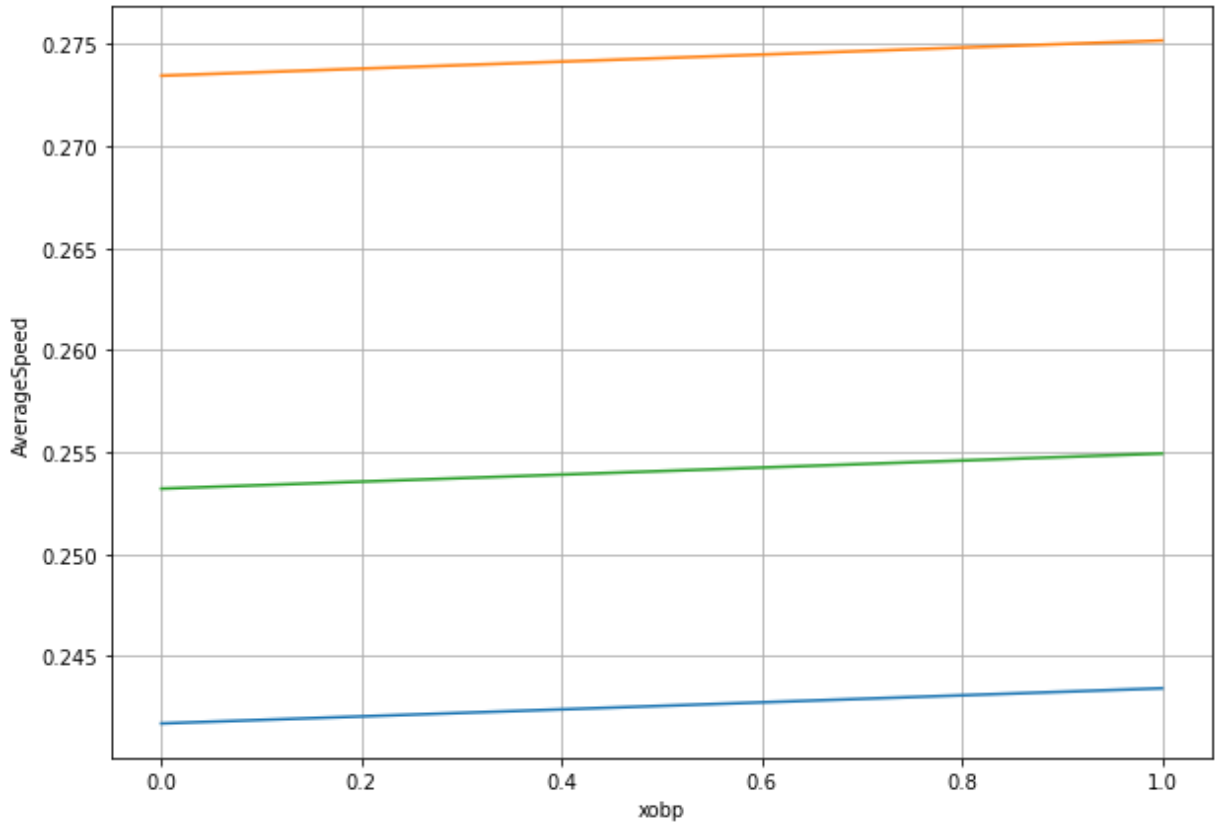
In [65]:
```
plt.figure(figsize = (10, 7))
plt.plot(z_swing_miss_percent_ME, y1)
plt.plot(z_swing_miss_percent_ME, y2)
plt.plot(z_swing_miss_percent_ME, y3)
plt.xlabel("AverageSpeed")
plt.ylabel("z_swing_miss_percent_")
plt.grid()
```



In [66]:
```
AverageSpeedME = np.linspace(bav.AverageSpeed.min(),bav.AverageSpeed.max(), 50)
minxobp = bav.xobp.min()
maxAverageSpeed = bav.xobp.max()
meanxobp = bav.xobp.mean()
meanz_swing_miss_percent = np.mean(bav.z_swing_miss_percent)
meanlaunch_angle_avg = np.mean(bav.launch_angle_avg)
meanxslg = np.mean(bav.xslg)
meansweet_spot_percent = np.mean(bav.sweet_spot_percent)
```

In [67]:
```
y1 = result4.params[0] +result4.params[1]*minxobp + result4.params[2]*meanlaunch_
y2 = result4.params[0] +result4.params[1]*maxAverageSpeed + result4.params[2]*mea
y3 =  result4.params[0] + result4.params[1]*meanxobp+ result4.params[2]*meanlaunc
```

In [68]:
```
plt.figure(figsize = (10, 7))
plt.plot(AverageSpeedME, y1)
plt.plot(AverageSpeedME, y2)
plt.plot(AverageSpeedME, y3)
plt.xlabel("xobp")
plt.ylabel("AverageSpeed")
plt.grid()
```



CONCLUSION

Selection of the model

According to 5-Fold Cross Validation, the model with the lowest RMSE is model 1, followed by model 2, 4 and 3. Out of sample performance concludes the same results. Looking at the F-Test, all models have very low p-values. This means that we can reject the null hypothesis and conclude that there's a significant relationship between the predictor variables and the response variable. Model 3 had the highest F-Test followed by model 4, 2 and 1. The adjusted R-squared for model 1 is .815; model 2 is .814; model 3 is .803; model 4 is .803. Model 2 had the lowest AIC and BIC followed by model 1, 3 and 4. After using the BP test and the White test for all the models, we

failed to reject the null values. This means our models follow a fairly constant variance pointing to homoskedasticity. The VIF for models 1 and 2 show very high multicollinearity. After removing *xwoba* in 3 and 4, the suspect in driving multicollinearity in models, the VIFs are under the threshold of 4 for models 3 and 4. While multicollinearity does not affect AIC and BIC significantly, we want to balance our model accordingly. Model 1 and 2 give a better fit with lower AIC and BIC, but we are also running hypothesis tests. The hypothesis tests are negatively influenced by multicollinearity and become biased. In order to balance our model, we will pick between Model 3 and 4 because of the high multicollinearity in Model 1 and 2. Model 3 has a lower RMSE than 4, a higher F-test than 4 and a lower AIC and BIC than model 4. Therefore, we conclude that model 3 is our best model.

Interpretation of the coefficients

If *xobp* increases by 1 unit, *xba* increases by 0.1884 on average. This is an incredibly large increase in *xba* that many hitters would love to know.
If *launch_angle_avg* increases by 1 unit, on average *xba* will decrease by -0.0021. Increasing the launch angle may help hit home runs, but increasing it too much leads to fly balls. Fly balls are outs and do not increase *xba*.
If *xslg* increases by 1 unit, on average *xba* increases by 0.1840. This too is a humongous increase in *xba*. If a hitter can hit for more than just a single, their batting average will go up, especially home runs which are guaranteed hits.
If *sweet_spot_percent* increases by 1 unit, *xba* increases by 0.0016 on average. This makes sense. Hitting the ball in the best part of the bat will increase the likelihood of a hit since the ball will be hit at a higher velocity, making it harder to catch.
If *z_swing_miss_percent* increases by 1, on average *xba* decreases by -0.0014. If a player is not even making contact with the ball, how can they get a hit? So, *xba* will decrease if a player continues to miss the ball.

Economic interpretation

Overall, the economic interpretation of our model and coefficient is useful for a variety of purposes within baseball athletics- our model could be used to predict the success of certain players, either by speculative fans or recruiters. On the other side, it could even be used by players and trainers to hone in on certain qualities of their play in order to improve performance. Our model is very likely a very rudimentary version of tools used by real MLB analyst teams in order to build holistic predictions of a player’s expected aptitude and potential.