Recitation_07_Linear_Regression

March 20, 2022

1 Welcome to Recitation 7!

#Go Read `Rec07-Intro.pdf` if you have not already.

This lab will not make sense without it!

1.1 RECITATION ASSIGNMENT

This lab project will guide you through the analysis. You can do the recitation by uploading your notebook and the data to Google colab, or you can do it on your laptop with Anaconda.

When you are finished, print your python notebook as a pdf and upload it to Gradescope.

1.2 Data

To obtain data on how various variable values affect pouring and cooling, a batch of 100 castings is poured with random variations in the mold variables about their baseline values. The data are available in the file `castdata.csv` on github. Each row contains parameter values (the inputs), and the cast batch time. The first line in the file contains the header with the names of the variables. The data start in the second row. The first row of data has the baseline values, that is, the values of the variables used in the current casting approach.

1.3 Variables

The following variables can be varied: `Riser Height`, `Riser Diameter`, `Riser 1 Position`, `Riser 2 Position`, `Gate Diameter`, `Cup Height`, `Sprue Height`, `Sprue Diameter Bottom`, and `Sprue Diameter Top` (see Figure 2). The response variable is "`BatchTime`".

1.4 Importing Data

We first need to import the libraries necessary for this recitation. Next, we have to upload our data into the notebook; Upload the data set `castdata.csv` from the course github. We will call the entire data set `df`.

[1]:
```
#math and arrays
import numpy as np
#dataframes
import pandas as pd
#plotting
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#linear regression and model selection
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
```

[2]:
```
# 'df' is common pandas dataframe nomenclature
df = pd.read_csv('castdata.csv')
df.head()
```

[2]:

	GateDiam	CupHeight	SprueHeight	RiserDiam	SprueDiamBot	SprueDiamTop	\
0	0.500000	5.000000	10.000000	7.000000	0.500000	0.750000	
1	0.318317	5.604768	10.148839	6.745247	0.635632	0.889822	
2	0.353834	5.846289	9.528183	7.345273	0.642489	1.005683	
3	0.596163	4.280124	10.887751	6.952856	0.317180	0.589558	
4	0.484338	5.205026	11.003688	7.254450	0.531424	0.595769	

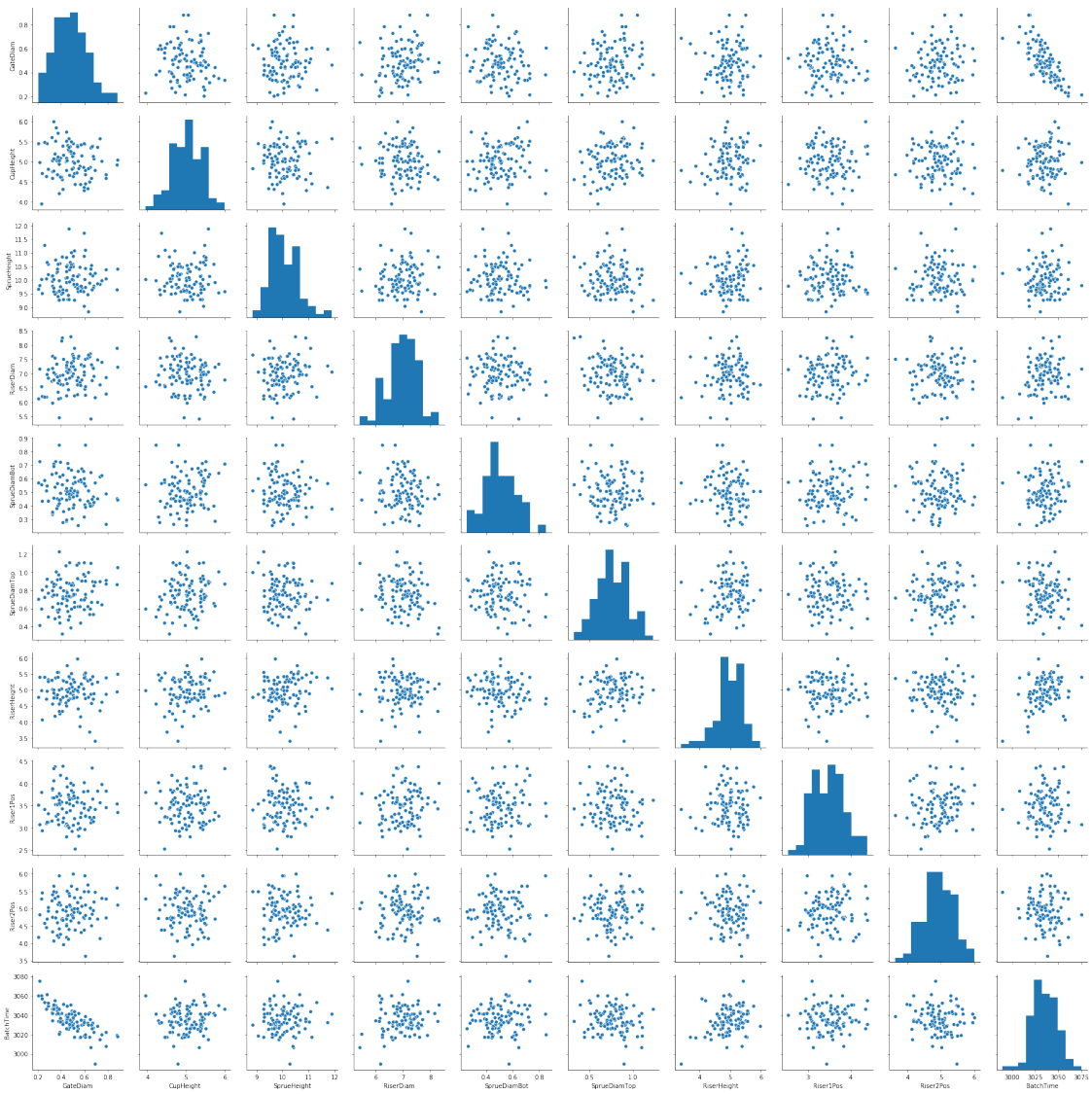
	RiserHeight	Riser1Pos	Riser2Pos	BatchTime	Feasible
0	5.000000	3.500000	5.000000	3031.292318	1
1	5.318354	3.168774	5.233322	3042.417820	0
2	4.836570	3.265961	5.482437	3050.300487	1
3	5.492964	4.029533	4.719791	3034.202547	1
4	4.897376	4.011947	4.822747	3033.046509	1

1.5 Visual Analysis

To examine the relationships between all pairs of variables ("`feasible`" is ignored through the use of `df.loc`, which splices the dataframe in the same way that one would a regular Python list), plot the dataframe with the pairplot command using the seaborn library:

[3]:
```
sns.pairplot(df.loc[:, :'BatchTime'])
```

[3]:
```
<seaborn.axisgrid.PairGrid at 0x7f8819910950>
```



When you pairplot a dataframe you get a scatterplot matrix. This matrix allows you to visualize all pairwise relationships between variables.

1.6 Question 1

Based on the scatterplot matrix, which variable of the first 9 in df most affects "`BatchTime`"?

Hint: Pick a plot with `BatchTime` on the y-axis and one of the other variables on the x-axis. Think about what a LOWESS line, like the one we created in lecture, would look like.

Recall that the height of the LOWESS line at a location on the x-axis is a weighted average of the heights of the points that are nearby on the x-axis, weighting the ones that are closer more heavily. If you'd like, you can actually draw a LOWESS line using python.

Would the height of the LOWESS line change a lot and in a consistent direction over large parts

of the x-axis? If so, then the variable on the x-axis affects `BatchTime` a lot.

1.6.1 Answer:

1.7 Multiple Linear Regression

We will now fit a multiple linear regression model for batch time using the mold variables as predictors. You'll see in the following code that we first import the statsmodels library. We then assign 'batchime' to be our dependent variable and 'X' to be our independent variables by splicing our dataframe as we did before. The line `X = sm.add_constant(X)` adds a column of all ones called `const` to X. This is needed to fit the intercept.

Next, we create a regression model and get the results of the regression using model.fit(). Finally, we summarize these results. Run the code below to view the summary table.

[7]:
```
batchtime = df["BatchTime"]
X = df.loc[:, :'Riser2Pos']
X = sm.add_constant(X)

model = sm.OLS(batchtime, X)
results = model.fit()

# Print out the stats
results.summary()
```

```
/usr/local/anaconda3/lib/python3.7/site-packages/numpy/core/fromnumeric.py:2495:
FutureWarning: Method .ptp is deprecated and will be removed in a future
version. Use numpy.ptp instead.
  return ptp(axis=axis, out=out, **kwargs)
```

[7]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:              BatchTime   R-squared:                       0.768
Model:                            OLS   Adj. R-squared:                  0.745
Method:                 Least Squares   F-statistic:                     33.18
Date:                Sun, 20 Mar 2022   Prob (F-statistic):           6.54e-25
Time:                        15:07:14   Log-Likelihood:                -326.08
No. Observations:                 100   AIC:                             672.2
Df Residuals:                      90   BIC:                             698.2
Df Model:                           9
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       2974.3611     20.133    147.736      0.000    2934.363    3014.359
GateDiam     -73.7652      4.919    -14.996      0.000     -83.538     -63.993
CupHeight      0.0650      1.766      0.037      0.971      -3.444       3.574
```

```
SprueHeight     -0.9823      1.236     -0.794      0.429      -3.439       1.474
RiserDiam        8.1317      1.304      6.238      0.000       5.542      10.722
SprueDiamBot     6.2152      5.638      1.102      0.273      -4.986      17.417
SprueDiamTop     0.5189      4.054      0.128      0.898      -7.536       8.574
RiserHeight      9.1809      1.617      5.679      0.000       5.969      12.392
Riser1Pos       -1.0727      1.816     -0.591      0.556      -4.681       2.536
Riser2Pos        0.8287      1.477      0.561      0.576      -2.105       3.763
==============================================================================
Omnibus:                        0.367   Durbin-Watson:                   2.040
Prob(Omnibus):                  0.832   Jarque-Bera (JB):                0.531
Skew:                           0.035   Prob(JB):                        0.767
Kurtosis:                       2.650   Cond. No.                         469.
==============================================================================

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

This summary output may seem intimidating at first; let's focus on the independent variables (section that vertically lists our variables with "coef", "std err", …). Specifically, we are interested in the column titled $P > |t|$, which gives the p-value or statistical significance for that variable. Here's a way to interpret these values: a p-value of .01 would indicate statistical significance at the 99% level for that variable. In determining the degree of effect a variable has on `BatchTime`, simply observe how large the coefficent is; this means that a small change in that variable has a significant impact on our dependent variable, `BatchTime`.

1.8 Question 2

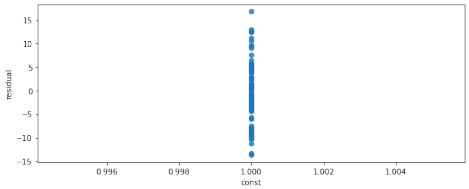
Observe the summary output in the model for `BatchTime`. Which variables appear to be statistically significant at the 95% level, and how do they affect the `BatchTime`? Which predictor has the largest effect on `BatchTime` according to the estimated values of beta-hat?

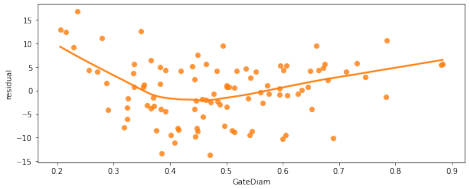
1.8.1 Answer:

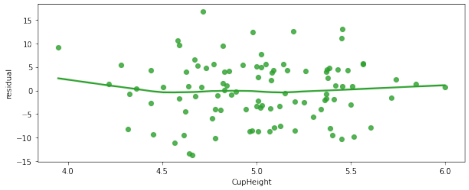
Now, for each of our variables, we would like to plot the residuals from our regression so that we can diagnose any issues; run the code below and observe the graphs.

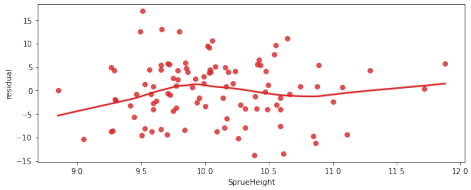
[16]:
```
# This creates 10 "subplots", arranged vertically ---
# subplots are little plots that are arranged together
# as the output of a python notebook cell
fig, ax = plt.subplots(10,figsize =(10,45))
resid = results.resid
for i in range(10): # for each of the nine variables
  # create a residual plot for that variable ('i' indexes the variables)
  # The argument ax = ax[i] puts the plot into the subplot
  sns.regplot(X.iloc[:,i], resid, lowess=True, ax = ax[i])
  ax[i].set_xlabel(X.columns[i])
```

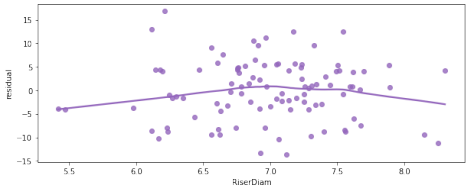
```
  ax[i].set_ylabel("residual")
```

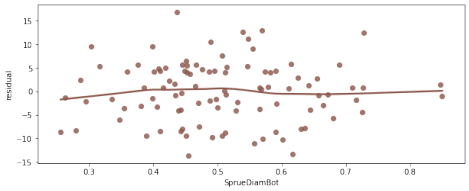

residual
const

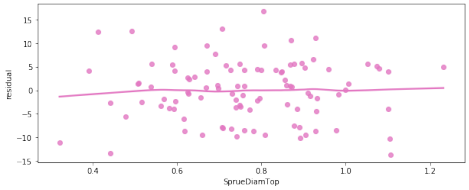

residual
GateDiam

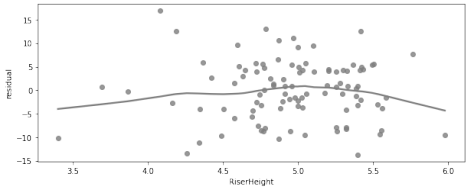

residual
CupHeight

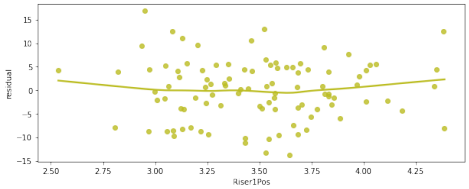

residual
SprueHeight

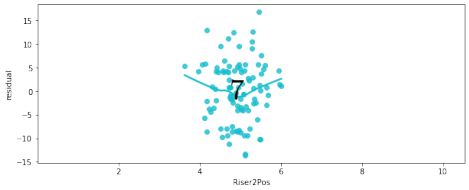

residual
RiserDiam


residual
SprueDiamBot


residual
SprueDiamTop


residual
RiserHeight


residual
Riser1Pos


residual
Riser2Pos

1.9 Question 3

Examine the nine plots, focusing on the covariates that you found to be statistically significant at the 95% level. Among these, does the model seem to be linear in each of these predictors? Or do you see any problems? (This was the 1st thing that we learned checked using Method 1 in Lecture 12.)

To check this, we want the LOWESS line (the solid line that you drew) to be approximately straight. It is normal to see the line bend a modest amount in magnitude in areas where there aren't many points (this is noise), but if the variation is really large and goes through the region where there are many points then this could be a problem.

You may find that you aren't sure whether you see any problems. That is ok. An important part of being a good data scientist is being able to work effectively despite ambiguity. To help you, we expect that there is one covariate (predictor) for which the model is not linear in this predictor. Write down the name of the statistically significant covariate for which you most suspect this to be true.

1.9.1 Answer:

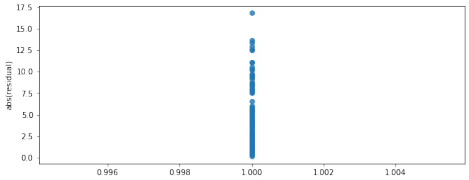
The code below plots the absolute value of the residuals so that we can check whether the variance of the noise depends on the covariates. Take a look at the plots, paying special attention to the covariates that were statistically significant.

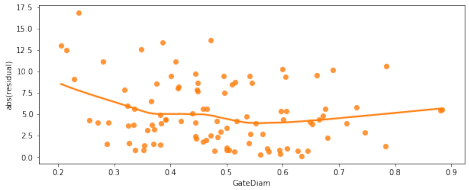
In these plots, the covariate in which the model failed to be linear from Question 3 also has some variation in the variance as we vary this covariate. This is a common phenomenon.

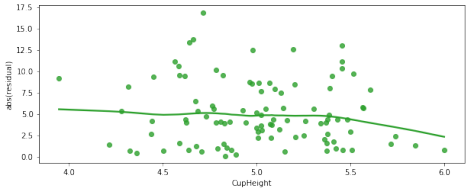
In the other plots for statistically significant covariates, the variance changes a bit, though not excessively.

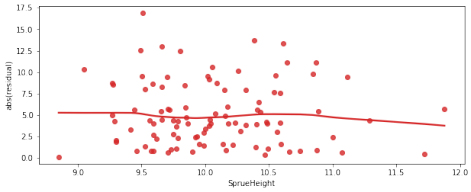
Before we worry about the changes in the variance, we'll fix the problem identified in Question 3 above.

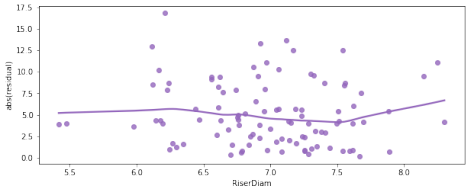
[18]:
```
fig, ax = plt.subplots(10,figsize =(10,45))
resid = results.resid
for i in range(10): # for each of the nine variables
  # create a residual plot for that variable ('i' indexes the variables)
  # The argument ax = ax[i] puts the plot into the subplot
  sns.regplot(X.iloc[:,i], abs(resid), lowess=True, ax = ax[i])
  ax[i].set_xlabel(X.columns[i])
  ax[i].set_ylabel("abs(residual)")
```

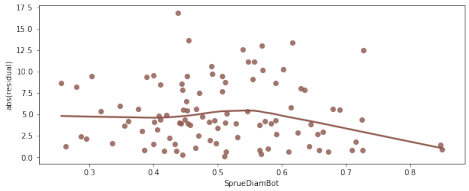

abs(residual)
const

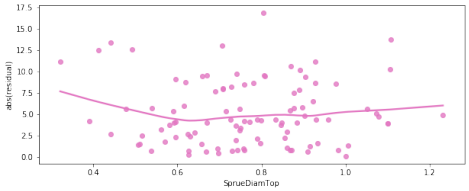

abs(residual)
GateDiam

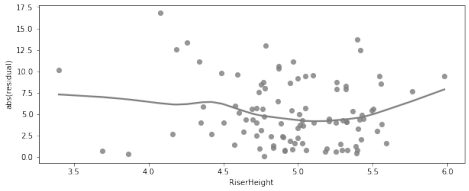

abs(residual)
CupHeight

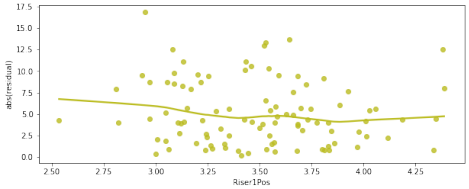

abs(residual)
SprueHeight

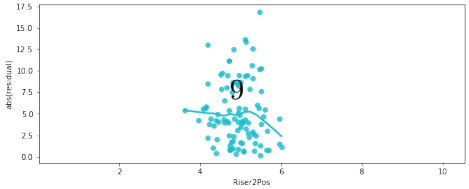

abs(residual)
RiserDiam


abs(residual)
SprueDiamBot


abs(residual)
SprueDiamTop


abs(residual)
RiserHeight


abs(residual)
Riser1Pos


abs(residual)
Riser2Pos

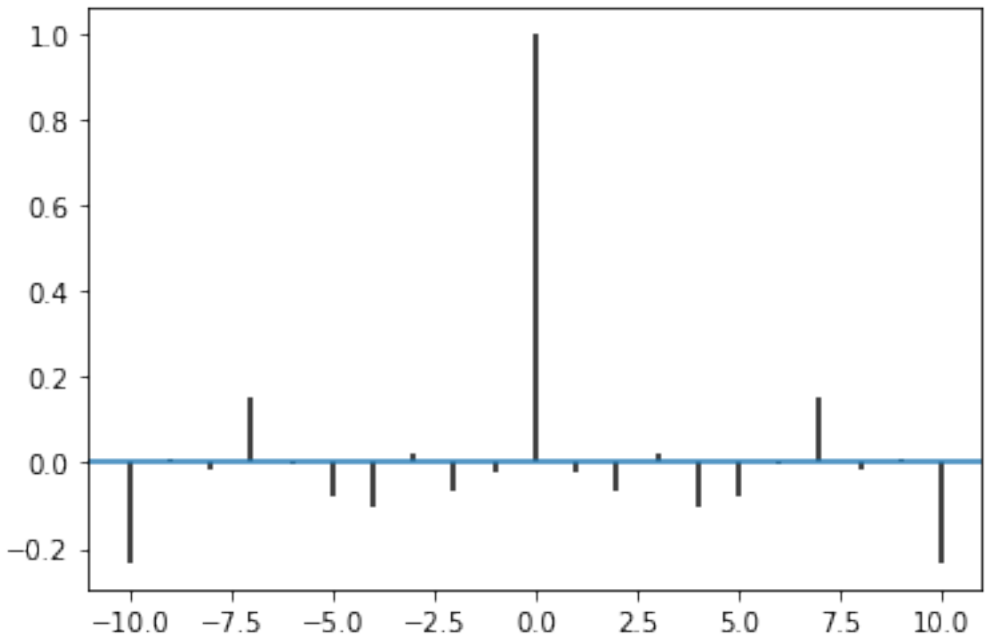
In a perfect world we would also check the other assumptions using Methods 2 and 3 from Lecture 12.

Since we didn't have time to cover these in last Friday's lecture and some people will complete the recitation before we have a chance to cover them in Wednesday's recitation, we just include the code and plots for checking them below, but don't ask any questions about them. The two assumptions that we would ordinarily check are:

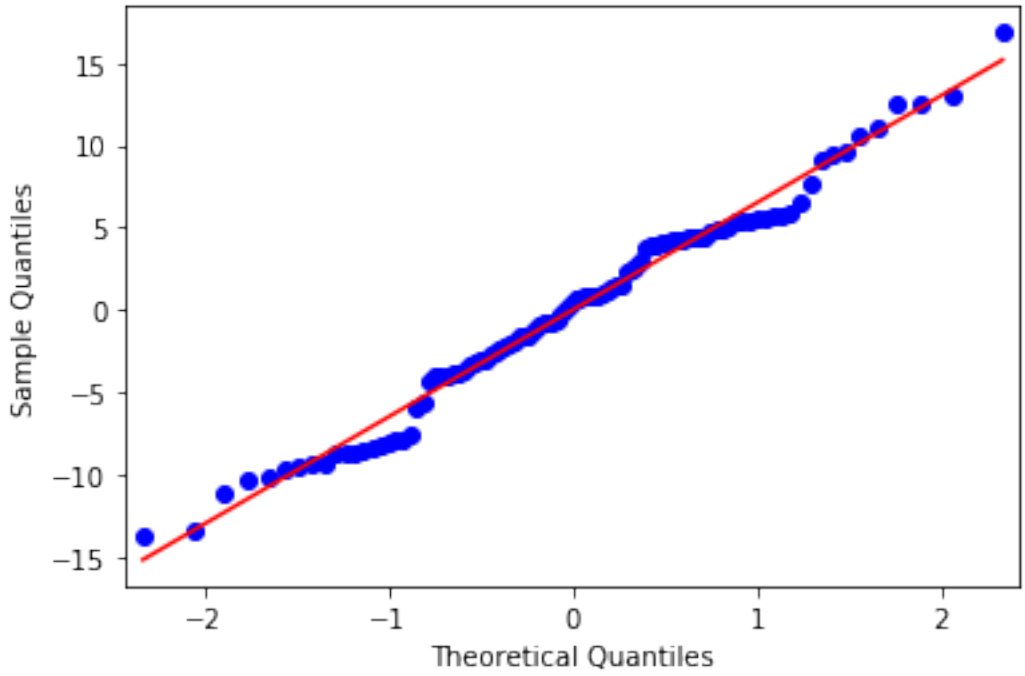
- Whether the residuals appear mutually independent. We would use an autocorrelation plot to check.
- Whether the residuals appear normallly distributed? We would use a qqplot to check.

[23]:
```
plt.acorr(resid) # Autocorrelation plot
```

[23]:
```
(array([-10,  -9,  -8,  -7,  -6,  -5,  -4,  -3,  -2,  -1,   0,   1,   2,
          3,   4,   5,   6,   7,   8,   9,  10]),
 array([-0.23362989,  0.00890035, -0.01343206,  0.15294954, -0.00154269,
        -0.07822909, -0.10422194,  0.02102587, -0.06782409, -0.02168824,
         1.        , -0.02168824, -0.06782409,  0.02102587, -0.10422194,
        -0.07822909, -0.00154269,  0.15294954, -0.01343206,  0.00890035,
        -0.23362989]),
 <matplotlib.collections.LineCollection at 0x7f87d944fa50>,
 <matplotlib.lines.Line2D at 0x7f880945e4d0>)
```



[28]:
```
sm.qqplot(resid, line="r") #qqplot
plt.show() # Statsmodels will produce two plots unless you add this
```



1.10 Adding nonlinear terms

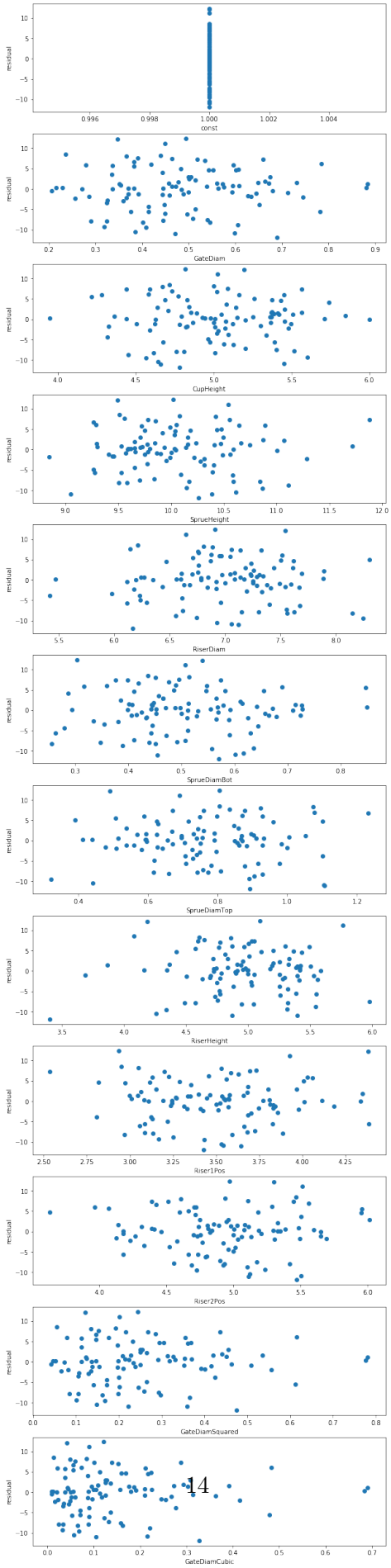
You may have noticed that `BatchTime` is nonlinearly related to `GameDiam`. Use the following commands to repeat the regression analysis with quadratic and cubic terms.

[29]:
```
X['GateDiamSquared'] = X['GateDiam']**2 # add quadratic term for GateDiam
X['GateDiamCubic'] = X['GateDiam']**3  # add cubic term
model2 = sm.OLS(batchtime, X)
results2 = model2.fit()

# Print out the stats
results2.summary()
```

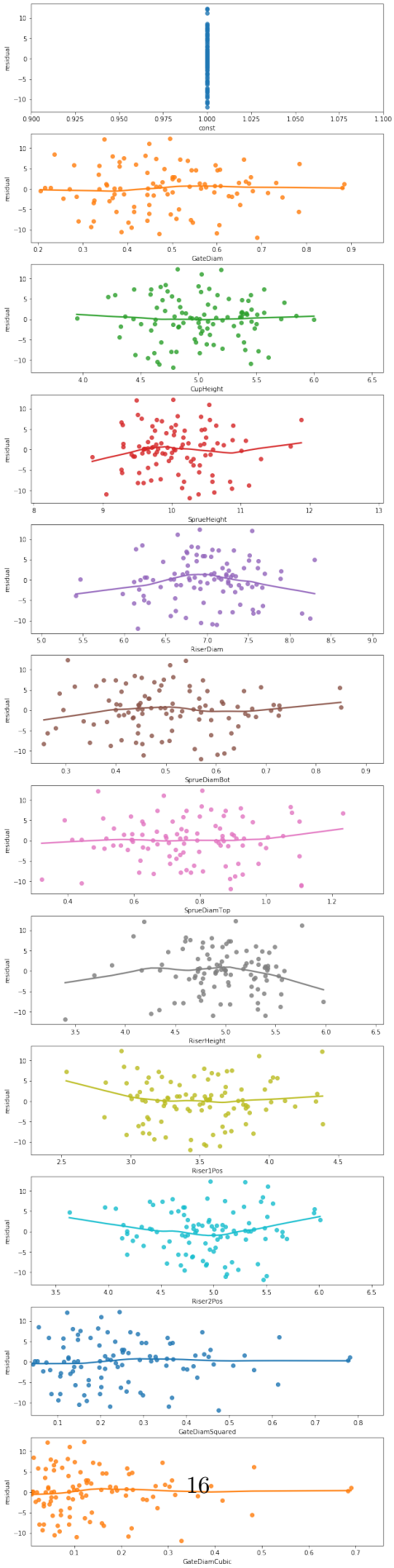
[29]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                            OLS Regression Results
==============================================================================
Dep. Variable:              BatchTime   R-squared:                       0.835
Model:                            OLS   Adj. R-squared:                  0.815
Method:                 Least Squares   F-statistic:                     40.52
Date:                Sun, 20 Mar 2022   Prob (F-statistic):           1.01e-29
Time:                        16:28:05   Log-Likelihood:                -309.10
```

```
No. Observations:                100   AIC:                             642.2
Df Residuals:                     88   BIC:                             673.5
Df Model:                         11
Covariance Type:            nonrobust
==============================================================================
===
                     coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
---
const            3028.3393     20.371    148.662      0.000    2987.857
3068.822
GateDiam         -476.1945     90.302     -5.273      0.000    -655.651
-296.738
CupHeight           1.1435      1.519      0.753      0.454      -1.876
4.162
SprueHeight        -0.8335      1.056     -0.790      0.432      -2.931
1.264
RiserDiam           9.5573      1.141      8.374      0.000       7.289
11.825
SprueDiamBot        4.5050      4.838      0.931      0.354      -5.110
14.120
SprueDiamTop        1.4360      3.480      0.413      0.681      -5.480
8.352
RiserHeight         9.1068      1.381      6.594      0.000       6.362
11.852
Riser1Pos          -0.7958      1.560     -0.510      0.611      -3.895
2.304
Riser2Pos           0.7674      1.263      0.608      0.545      -1.742
3.277
GateDiamSquared   694.4074    176.257      3.940      0.000     344.134
1044.681
GateDiamCubic    -367.9510    108.946     -3.377      0.001    -584.457
-151.445
==============================================================================
Omnibus:                        0.361   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.835   Jarque-Bera (JB):                0.523
Skew:                          -0.102   Prob(JB):                        0.770
Kurtosis:                       2.710   Cond. No.                     6.14e+03
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 6.14e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
"""
```

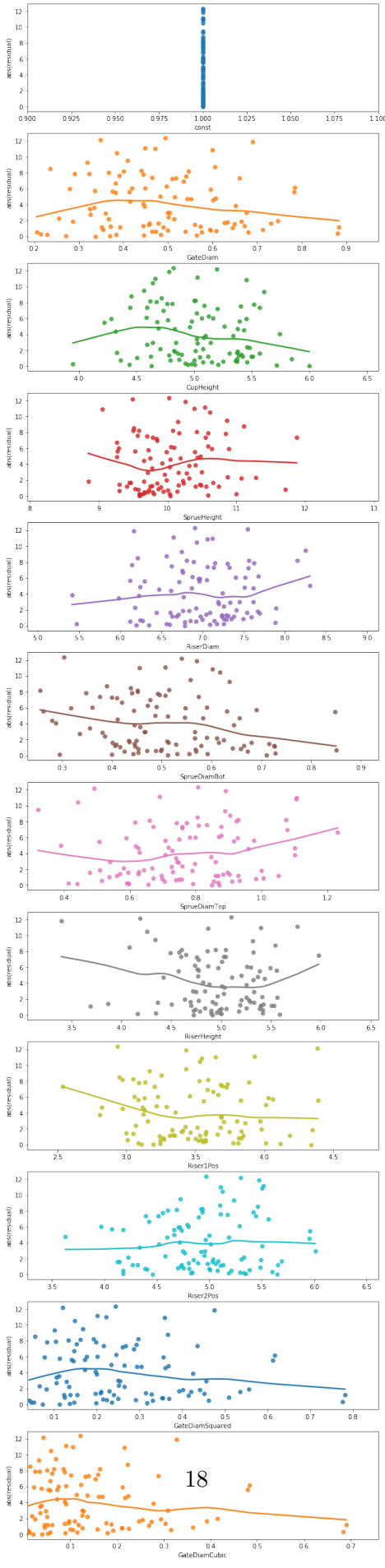
[10]:
```
fig, ax = plt.subplots(12,figsize =(10,45))
resid = results2.resid
for i in range(12): # for each of the nine variables
  # create a fitted plot for that variable ('i' indexes the variables)
  ax[i].scatter(X.iloc[:,i],resid)
  ax[i].set_xlabel(X.columns[i])
  ax[i].set_ylabel("residual")
```


residual
const
GateDiam
CupHeight
SprueHeight
RiserDiam
SprueDiamBot
SprueDiamTop
RiserHeight
Riser1Pos
Riser2Pos
GateDiamSquared
GateDiamCubic

[42]:
```
# Plot the residuals
fig, ax = plt.subplots(12,figsize =(10,45))
resid = results2.resid
for i in range(12): # for each of the nine variables
  sns.regplot(X.iloc[:,i], resid, lowess=True, ax = ax[i])
  ax[i].set_xlabel(X.columns[i])
  xmin = np.min(X.iloc[:,i]) # minimum x value in the data
  xmax = np.max(X.iloc[:,i])
  ax[i].set_xlim(xmin*.9,xmax*1.1) # For some reason the manual method for␣
 ↪setting the x-axis doesn't work well for GateDiamCubic
  ax[i].set_ylabel("residual")
```


residual
const
GateDiam
CupHeight
SprueHeight
RiserDiam
SprueDiamBot
SprueDiamTop
RiserHeight
Riser1Pos
Riser2Pos
GateDiamSquared
GateDiamCubic

[43]:
```
# Plot the absolute value of the residuals
fig, ax = plt.subplots(12,figsize =(10,45))
resid = results2.resid
for i in range(12): # for each of the nine variables
  sns.regplot(X.iloc[:,i], abs(resid), lowess=True, ax = ax[i])
  ax[i].set_xlabel(X.columns[i])
  xmin = np.min(X.iloc[:,i]) # minimum x value in the data
  xmax = np.max(X.iloc[:,i])
  ax[i].set_xlim(xmin*.9,xmax*1.1) # For some reason the manual method for␣
 ↪setting the x-axis doesn't work well for GateDiamCubic
  ax[i].set_ylabel("abs(residual)")
```


abs(residual)
const
GateDiam
CupHeight
SprueHeight
RiserDiam
SprueDiamBot
SprueDiamTop
RiserHeight
Riser1Pos
Riser2Pos
GateDiamSquared
GateDiamCubic

1.10.1 Question 4

Now based on the new model we have just fit, are the assumptions about the errors you checked above satisfied? In particular, check * Is the model linear in the predictors? * Are the variance of the errors constant?

We can skip these two assumptions: * Are there problems with non-constant variance (plot of absolute residuals) * Do the residuals appear normall distributed? (qq plot)

1.10.2 Ans:

[ ]: