In [1]:
```
import pandas as pd
import numpy as np
import statsmodels.api as sm
from scipy import *
from pandas import Series,DataFrame
from patsy import dmatrices
plt.figure(figsize=(12,8))
results={}
```

In [2]:
```
# Sets our path
#cd "C:\\Users\\Andrew\\Documents\\Kaggle\\Blue Book For Bulldozers/Dups/BlueBook git/"
```

In [3]:
```
import sys
sys.path.append('C:\Users\Andrew\Documents\Kaggle\Kaggel Aux')
import kaggelaux as agc
```

In [4]:
```
df = pd.read_csv("Train.csv") # Import Data
df = df.ix[:,'SalesID':'ProductGroupDesc'] # Remove extraneous Features
```

Data Quality? What are we up against?

Like most real world data, there were many inaccruaceys and occurances of malformed data in the dataset provided by FastIron.

Some of the Highlights inlcude:

- Many Bulldozers were reported as being manufactured in 1000 A.D.
- Many Bulldozers had multiple manufacturing dates.
- Machine ID, that was suppsoed to track an induvidual machine over its lifetime, was assigned to many machines.
- Auctioneer ID was later revealed as an aggregate of of binned auction houses created by Fastlorn, instead of indivudial auction houses

To compensate for these errors FastIron released an appendix to correct the data, though many belived the appendix had many of the same problems. These issues are compensated for in below.

Auxilary Data:

In Addition to the information provided by Fastlorn, I included serveral features to account for seasonality, inflation, and the changing demand of heavy machinary.

Those Features are:

- GDP -- FRED
- US Inflation -- FRED
- US Heavy Idustrial Production Index - FRED

- Catapiler's adjusted closing price -- Yahoo Finance

All added features were lagged 90 days so that they would be availiable at the auction time

Below shows process of lagging and mergeing the new data as well as the new Machine Appendix to fix data quality issues.

In [5]:
```
# Read in additional Data
machine_appendix = pd.read_csv('Machine_Appendix.csv')
GDP = pd.read_csv('GDP.csv', names=['DATE','GDP'], header=0)
inflation = pd.read_csv('CPIAUCSL.csv',names=['DATE','Infation_Index'], header=0)
Industrial_production = pd.read_csv('INDPRO_vb.csv',names=['DATE','Industrial_production_index'], header=0)
cat = pd.read_csv('cat_data.csv',names=['Date','cat_open','cat_high','cat_low','cat_close','cat_vol','cat_adj_close'], header=0)
```

In [6]:
```
# Prepare data for merge and comparison
# New data
GDP.DATE = GDP.DATE.apply(lambda x: datetime.datetime.strptime(x, "%Y-%m-%d")) # Coverts string dates into datetime objects
inflation.DATE = inflation.DATE.apply(lambda x: datetime.datetime.strptime(x, "%Y-%m-%d"))
Industrial_production.DATE = Industrial_production.DATE.apply(lambda x: datetime.datetime.strptime(x, "%Y-%m-%d"))
cat.Date = cat.Date.apply(lambda x: datetime.datetime.strptime(x, "%m/%d/%Y"))
# Our original data
df.saledate = df.saledate.apply(lambda x: datetime.datetime.strptime(str(x), '%m/%d/%Y %H:%M'))
```

In [7]:
```
# Merge data on the machine appendix to correct for data quality .
df = pd.merge(df, machine_appendix, left_on='MachineID', right_on='MachineID', how='right', suffixes=('_orginal','_machine_appendix'))
```

In [8]:
```
# Fill data forward to make a complete calender; ie. not just trading/business days as auctions happend 24/7.

# Prepare data for forward filling
GDP = GDP.set_index('DATE')
inflation = inflation.set_index('DATE')
Industrial_production = Industrial_production.set_index('DATE')
cat = cat.set_index('Date')

# Fill data forward
GDP = GDP.resample('D', fill_method='ffill')
inflation = inflation.resample('D', fill_method='ffill')
Industrial_production = Industrial_production.resample('D', fill_method='ffill')
cat = cat.resample('D', fill_method='ffill')
```

In [9]:
```
# Lag our new variales and then merge them to our dataset
df = pd.merge(GDP.shift(90, freq='D'), df, left_index=True, right_on='saledate', how='right')
df = pd.merge(inflation.shift(90, freq='D'), df, left_index=True, right_on='saledate', how='right')
df = pd.merge(Industrial_production.shift(90, freq='D'), df, left_index=True, right_on='saledate', how='right')
df = pd.merge(cat.shift(90, freq='D'), df, left_index=True, right_on='saledate', how='right')
```

In [10]:
```python
# Below are several helpful functions that we will use to clean the dozer manufacturing date data and prepare it for our analysis.

def dozeryearclean(s):
    if s == 1000:
        s = np.nan
    return s

def MfgClean(s):
    if isinstance(s, str):
        s = np.nan
    return s
```

In [11]:
```python
#Lets take a look at our new DataFrame
df
```

Out[11]:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 418691 entries, 0 to 418690
Data columns (total 44 columns):
cat_open                              401125  non-null values
cat_high                              401125  non-null values
cat_low                               401125  non-null values
cat_close                             401125  non-null values
cat_vol                               401125  non-null values
cat_adj_close                         401125  non-null values
Industrial_production_index           401125  non-null values
Infation_Index                        401125  non-null values
GDP                                   401125  non-null values
SalesID                               401125  non-null values
SalePrice                             401125  non-null values
MachineID                             418691  non-null values
ModelID_orginal                       401125  non-null values
datasource                            401125  non-null values
auctioneerID                          380989  non-null values
YearMade                              401125  non-null values
MachineHoursCurrentMeter              142765  non-null values
UsageBand                             69639  non-null values
saledate                              401125  non-null values
fiModelDesc_orginal                   401125  non-null values
fiBaseModel_orginal                   401125  non-null values
fiSecondaryDesc_orginal               263934  non-null values
fiModelSeries_orginal                 56908  non-null values
fiModelDescriptor_orginal             71919  non-null values
ProductSize                           190350  non-null values
fiProductClassDesc_orginal            401125  non-null values
state                                 401125  non-null values
ProductGroup_orginal                  401125  non-null values
ProductGroupDesc_orginal              401125  non-null values
ModelID_machine_appendix              418691  non-null values
fiModelDesc_machine_appendix          418691  non-null values
fiBaseModel_machine_appendix          418691  non-null values
```

```
fiSecondaryDesc_machine_appendix       277506  non-null values
fiModelSeries_machine_appendix         60164  non-null values
fiModelDescriptor_machine_appendix     78268  non-null values
fiProductClassDesc_machine_appendix    418691  non-null values
ProductGroup_machine_appendix          418691  non-null values
ProductGroupDesc_machine_appendix      418691  non-null values
MfgYear                                418432  non-null values
fiManufacturerID                       418691  non-null values
fiManufacturerDesc                     418691  non-null values
PrimarySizeBasis                       413365  non-null values
PrimaryLower                           413365  non-null values
PrimaryUpper                           413365  non-null values
dtypes: datetime64[ns](1), float64(20), int64(2), object(21)
```

In [12]:

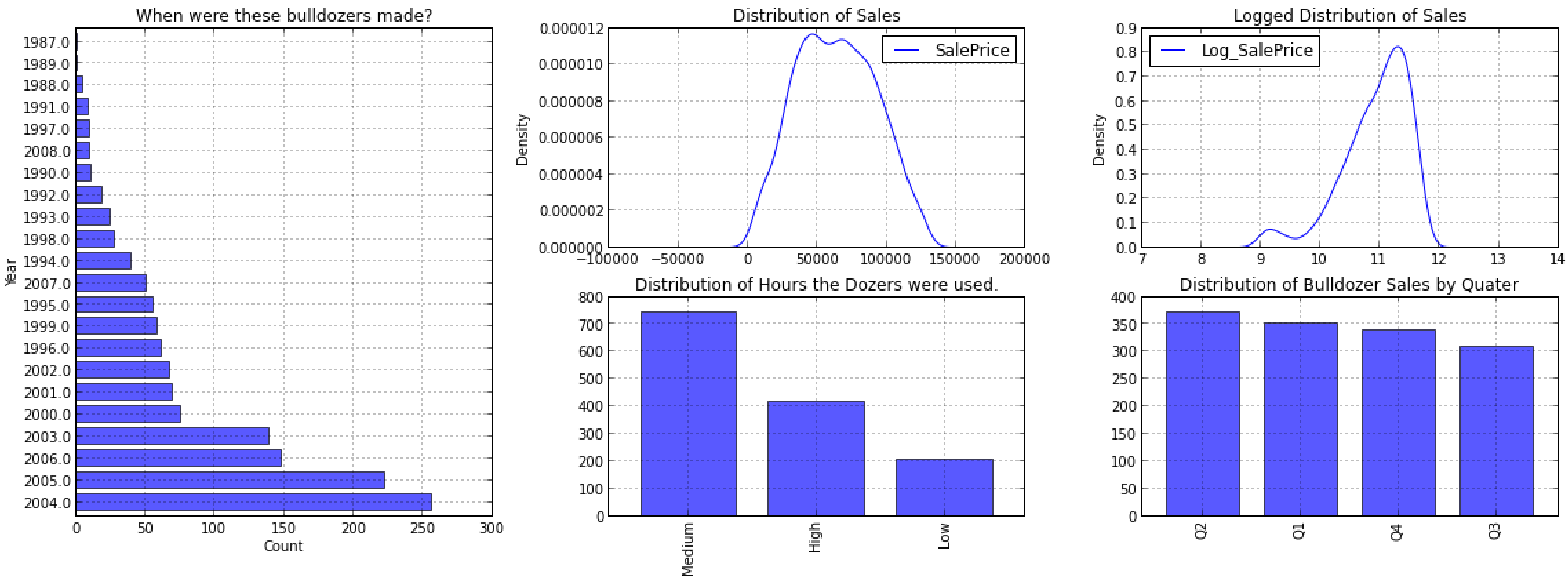
```
# Clean Data to prepare for analysis
df.YearMade = df.YearMade.apply(dozeryearclean)
df.MfgYear = df.MfgYear.apply(MfgClean)
df['Quater'] = df.saledate.apply(agc.quater_maker)# adds a column for that specifices the quater each machine was sold in.

df = df.dropna() # Drops Null Values from the dataframe.
```

Visual Summary of the Data:

In [13]:

```
fig = plt.figure(figsize=(16,6))
df['Log_SalePrice'] = df.SalePrice.apply(lambda x: log(x)) # transforms sales price
a=.65 # sets the alpha level

ax1 = plt.subplot2grid((2, 3), (0, 1))
df.SalePrice.plot(kind='kde')
title("Distribution of Sales"); legend(loc='best')

ax2 = plt.subplot2grid((2, 3), (0, 2), colspan=2)
df.Log_SalePrice.plot(kind='kde')
title("Logged Distribution of Sales"); legend(loc='best')

ax3 = plt.subplot2grid((2, 3), (0, 0), colspan=1, rowspan=2)
df.YearMade.value_counts().plot(kind='barh', alpha=a)
title("When were these bulldozers made?")
plt.xlabel('Count')
plt.ylabel('Year')
# you could show a distrubution of how old they are

ax4 = plt.subplot2grid((2, 3), (1, 2), rowspan=2)
df.Quater.value_counts().plot(kind='bar',alpha= a)
title("Distribution of Bulldozer Sales by Quater")

#add one for prices over time. to show inlation # show frequency of sales per year
#ax4 = plt.subplot2grid((2, 3), (1, 2))
```

```
#dfs=df.sort('saledate')
#dfs.set_index('saledate')
#plot_date(dfs.SalePrice,df.saledate)
#title("Sale Price Over Time:")
#plt.xlabel('SalMachineID')
#plt.ylabel('Price ($)')
#plt.xticks(np.arange(0,5))
#plt.yticks(np.arange(0,5))

ax5 = plt.subplot2grid((2, 3), (1, 1))
df.UsageBand.value_counts().plot(kind='bar', alpha=a)
title("Distribution of Hours the Dozers were used. ")


plt.tight_layout()
```



My Regressions:

Regression 1:

show a typicall competitor reg with GDP without my adds. show reg with my adds. show how you zeroed in to things that made sense to explain the dozers and how r2 went up. expalin the seasonalitly/ inflation in the dataset show how you adj. then show output to kaggel. Explain why everything is lagged 90 days

In [14]:
```
df.set_index("SalesID")
```

```
# So our regression Data frames objs keep thier salesid
```

Out[14]:

```
<class 'pandas.core.frame.DataFrame'>
Index: 1368 entries, 4359500.0 to 6304507.0
Data columns (total 45 columns):
cat_open                                1368  non-null values
cat_high                                1368  non-null values
cat_low                                 1368  non-null values
cat_close                               1368  non-null values
cat_vol                                 1368  non-null values
cat_adj_close                           1368  non-null values
Industrial_production_index             1368  non-null values
Infation_Index                          1368  non-null values
GDP                                     1368  non-null values
SalePrice                               1368  non-null values
MachineID                               1368  non-null values
ModelID_orginal                         1368  non-null values
datasource                              1368  non-null values
auctioneerID                            1368  non-null values
YearMade                                1368  non-null values
MachineHoursCurrentMeter                1368  non-null values
UsageBand                               1368  non-null values
saledate                                1368  non-null values
fiModelDesc_orginal                     1368  non-null values
fiBaseModel_orginal                     1368  non-null values
fiSecondaryDesc_orginal                 1368  non-null values
fiModelSeries_orginal                   1368  non-null values
fiModelDescriptor_orginal               1368  non-null values
ProductSize                             1368  non-null values
fiProductClassDesc_orginal              1368  non-null values
state                                   1368  non-null values
ProductGroup_orginal                    1368  non-null values
ProductGroupDesc_orginal                1368  non-null values
ModelID_machine_appendix                1368  non-null values
fiModelDesc_machine_appendix            1368  non-null values
fiBaseModel_machine_appendix            1368  non-null values
fiSecondaryDesc_machine_appendix        1368  non-null values
fiModelSeries_machine_appendix          1368  non-null values
fiModelDescriptor_machine_appendix      1368  non-null values
fiProductClassDesc_machine_appendix     1368  non-null values
ProductGroup_machine_appendix           1368  non-null values
ProductGroupDesc_machine_appendix       1368  non-null values
MfgYear                                 1368  non-null values
fiManufacturerID                        1368  non-null values
fiManufacturerDesc                      1368  non-null values
PrimarySizeBasis                        1368  non-null values
PrimaryLower                            1368  non-null values
PrimaryUpper                            1368  non-null values
Quater                                  1368  non-null values
Log_SalePrice                           1368  non-null values
dtypes: datetime64[ns](1), float64(20), int64(2), object(22)
```

In [15]:

```
df.columns
```

Out[15]:
```
Index([cat_open, cat_high, cat_low, cat_close, cat_vol, cat_adj_close, Industrial_production_index, Infation_Index, GDP, SalesID, SalePrice,
MachineID, ModelID_orginal, datasource, auctioneerID, YearMade, MachineHoursCurrentMeter, UsageBand, saledate, fiModelDesc_orginal,
fiBaseModel_orginal, fiSecondaryDesc_orginal, fiModelSeries_orginal, fiModelDescriptor_orginal, ProductSize, fiProductClassDesc_orginal,
state, ProductGroup_orginal, ProductGroupDesc_orginal, ModelID_machine_appendix, fiModelDesc_machine_appendix, fiBaseModel_machine_appendix,
fiSecondaryDesc_machine_appendix, fiModelSeries_machine_appendix, fiModelDescriptor_machine_appendix, fiProductClassDesc_machine_appendix,
ProductGroup_machine_appendix, ProductGroupDesc_machine_appendix, MfgYear, fiManufacturerID, fiManufacturerDesc, PrimarySizeBasis,
PrimaryLower, PrimaryUpper, Quater, Log_SalePrice], dtype=object)
```

In [16]:
```
formula = 'np.log(SalePrice) ~ C(YearMade) + C(ModelID_machine_appendix) + C(Quater) + C(auctioneerID) + C(UsageBand) + GDP'
```

In [17]:
```
y, x = dmatrices(formula, data=df, return_type='dataframe')# Prepare data frame for regression
mod = sm.OLS(y, x)    # Describe model, so it can be fitted mod = sm.GLM(y, x, family=sm.families.Gamma())
res1 = mod.fit()       # Fit model
results['Typical_competitor_Regression']=[res1, formula] # save results for later
res1.summary()    # Summarize model -- you dont have to print
```

Out[17]:

OLS Regression Results

Dep. Variable:	np.log(SalePrice)	**R-squared:**	0.848
Model:	OLS	**Adj. R-squared:**	0.832
Method:	Least Squares	**F-statistic:**	53.02
Date:	Mon, 13 May 2013	**Prob (F-statistic):**	0.00
Time:	15:01:14	**Log-Likelihood:**	110.83
No. Observations:	1368	**AIC:**	40.34
Df Residuals:	1237	**BIC:**	724.3
Df Model:	130		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
Intercept	11.4038	0.353	32.279	0.000	10.711 12.097
C(YearMade)[T.1988.0]	0.0430	0.260	0.165	0.869	-0.467 0.553

Regression 2:

In [18]:
```
formula = 'np.log(SalePrice) ~ Infation_Index + Industrial_production_index + np.log(cat_adj_close) + C(YearMade) + C(ModelID_machine_appendix
```

In [19]:
```
y, x = dmatrices(formula, data=df, return_type='dataframe')
mod = sm.OLS(y, x)    # Describe model, so it can be fitted C(UsageBand)
res2 = mod.fit()      # Fit model
results['My_final_regression']=[res2, formula]
res2.summary()        # Summarize model
```

Out[19]:

OLS Regression Results

Dep. Variable:	np.log(SalePrice)	**R-squared:**	0.889
Model:	OLS	**Adj. R-squared:**	0.877
Method:	Least Squares	**F-statistic:**	74.84
Date:	Mon, 13 May 2013	**Prob (F-statistic):**	0.00
Time:	15:01:26	**Log-Likelihood:**	325.90
No. Observations:	1368	**AIC:**	-385.8
Df Residuals:	1235	**BIC:**	308.6
Df Model:	132		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
Intercept	12.1411	0.421	28.823	0.000	11.315 12.968
C(YearMade)[T.1988.0]	0.0131	0.222	0.059	0.953	-0.423 0.449

Regresion 3:

In [20]:
```
df.columns
```

Out[20]:
```
Index([cat_open, cat_high, cat_low, cat_close, cat_vol, cat_adj_close, Industrial_production_index, Infation_Index, GDP, SalesID, SalePrice,
MachineID, ModelID_orginal, datasource, auctioneerID, YearMade, MachineHoursCurrentMeter, UsageBand, saledate, fiModelDesc_orginal,
fiBaseModel_orginal, fiSecondaryDesc_orginal, fiModelSeries_orginal, fiModelDescriptor_orginal, ProductSize, fiProductClassDesc_orginal,
state, ProductGroup_orginal, ProductGroupDesc_orginal, ModelID_machine_appendix, fiModelDesc_machine_appendix, fiBaseModel_machine_appendix,
fiSecondaryDesc_machine_appendix, fiModelSeries_machine_appendix, fiModelDescriptor_machine_appendix, fiProductClassDesc_machine_appendix,
ProductGroup_machine_appendix, ProductGroupDesc_machine_appendix, MfgYear, fiManufacturerID, fiManufacturerDesc, PrimarySizeBasis,
PrimaryLower, PrimaryUpper, Quater, Log_SalePrice], dtype=object)
```

In [21]:
```
formula = 'np.log(SalePrice) ~ fiBaseModel_machine_appendix + C(fiProductClassDesc_machine_appendix) + Infation_Index + Industrial_production_
```

In [22]:
```
y, x = dmatrices(formula, data= df, return_type='dataframe')
mod = sm.OLS(y, x)     # Describe model, so it can be fitted  fredb
res3 = mod.fit()       # Fit model
results['My_regression_corrected_for_overfitting']=[res3, formula]
res3.summary()         # Summarize model
```

Out[22]:

OLS Regression Results

Dep. Variable:	np.log(SalePrice)	**R-squared:**	0.877
Model:	OLS	**Adj. R-squared:**	0.869
Method:	Least Squares	**F-statistic:**	109.4
Date:	Mon, 13 May 2013	**Prob (F-statistic):**	0.00
Time:	15:01:38	**Log-Likelihood:**	259.04
No. Observations:	1368	**AIC:**	-348.1

Df Residuals:	1283	BIC:	95.71
Df Model:	84		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
Intercept	10.4202	0.358	29.081	0.000	9.717 11.123
fiBaseModel_machine_appendix[T.1850]	0.2852	0.114	2.499	0.013	0.061 0.509

Use Our Models to Make Predictions:

In [23]:
```
test_data = pd.read_csv("Test.csv") # Load in the dataset
test_data = test_data.ix[:,'SalesID':'ProductGroupDesc'] # Remove extraneous Features
```

In [24]:
```
# Clean data in same way as our training data
test_data.saledate = test_data.saledate.apply(lambda x: datetime.datetime.strptime(str(x), '%m/%d/%Y %H:%M'))
```

In [25]:
```
# Merge data on the machine appendix to correct for data quality .
test_data = pd.merge(test_data, machine_appendix, left_on='MachineID', right_on='MachineID', how='left', suffixes=('_orginal','_machine_append
```

In [26]:
```
# Lag our new variales and then merge them to our dataset
test_data = pd.merge(GDP.shift(90, freq='D'), test_data, left_index=True, right_on='saledate', how='right')
test_data = pd.merge(inflation.shift(90, freq='D'), test_data, left_index=True, right_on='saledate', how='right')
test_data = pd.merge(Industrial_production.shift(90, freq='D'), test_data, left_index=True, right_on='saledate', how='right')
test_data = pd.merge(cat.shift(90, freq='D'), test_data, left_index=True, right_on='saledate', how='right')
```

In [27]:
```
test_data['Quater'] = test_data.saledate.apply(agc.quater_maker)
test_data['SalePrice'] = 1.23
test_data = test_data.set_index("SalesID")
```

Out Put results and score based on Cross Validation Data and test data

In [*]:
```
# Load in solved Cross Validation dataset for scoring
valid_solution = pd.read_csv('ValidSolution.csv')
valid = pd.read_csv('Valid.csv')
```

In [*]:
```
# Set Variables for manipulation
prog = 0 # variable to keep track of the progress of the output.
check_points = 17 # total number of progress points per iteration.
```

```python
for i in results:

    model_params = Series(results[i][0].params)
    model_params = model_params.to_dict()
    formula = results[i][1]
    print formula
    prog += 1
    agc.progress(prog, check_points) # a simple textual progress bar

    # Create reg friendly test dataframe
    yt, xt = dmatrices(formula, data=test_data, return_type='dataframe')
    prog += 1
    agc.progress(prog, check_points)

    # Use our models to create predictions
    yholder = 0

    for _ in xt.index:
        for w in xt.columns:
            if w in model_params:
                yholder += xt.ix[_,w] * model_params[w]
        yt.ix[_] = yholder
        yholder = 0
    prog += 1
    agc.progress(prog, check_points)

    # Output Results so our performance on the test set can be scored by Kaggle
    yt['SalePrice'] = yt['np.log(SalePrice)'].apply(lambda x: exp(x))
    yt.to_csv(i + ".csv", na_rep=0, float_format='%.3f')
    prog += 1
    agc.progress(prog, check_points)

    # Score the results based on our training set
    yt, xt = dmatrices(formula, data=valid, return_type='dataframe')
    prog += 1
    agc.progress(prog, check_points)

    #rmsle = agc.score_rmsle('SalePrice', yt, valid_solution)
    prog += 1
    agc.progress(prog, check_points)

    #rmse = agc.score_rmse('SalePrice', yt, valid_solution)
    prog += 1
    agc.progress(prog, check_points)

    print "Model %s scored an RMSE of : %.4f and an RMSLE of : %.4f\n" %(i, rmse, rmsle)
```

```
np.log(SalePrice) ~ fiBaseModel_machine_appendix + C(fiProductClassDesc_machine_appendix) + Infation_Index + Industrial_production_index +
np.log(cat_adj_close) + C(YearMade) + C(Quater) + C(auctioneerID) + C(UsageBand)
[####             ]
```

In [*]:
```python
show my two versions perfomance, how one was compensated for over fitting
```

```
show winners final place
```

Post Mortem

The my two final models scord on the validation set and on the test set. I selected my final model for submission and finished in 108th place. The winning model conducted by team Leustagos & Titericz scored an RMSLE of 0.22773 using an ensemble of GLM models.

What could I have done to improve my own model?

If I had used my second model, which corrected for what I belived the overfitting effect of the ModelID varaable, I would have scored and palced . But what would I have done to get in the top models? Its as simple as the random forest shown below. It would have had an RMSLE of 0.26704, only 0.039 greater than the winning model, and I would have palced 60th if submitted.

In [*]:
```
from sklearn.ensemble import RandomForestRegressor

formula = 'SalePrice~'
for i in test_data.columns:
    formula += i

yt, xt = dmatrices(formula, data=test_data, return_type='dataframe')

rf = RandomForestRegressor(n_estimators=50, n_jobs=1, compute_importances = True)
rf.fit(xt, yt)
predictions = rf.predict(test_fea)
```

Type *Markdown* and LaTeX: α^2