Lesson 6 - Solution Code

In [63]:
```
%matplotlib inline
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
import sklearn.linear_model

# read in the mammal dataset
wd = '../../assets/dataset/msleep/'
mammals = pd.read_csv(wd+'msleep.csv')
mammals = mammals[mammals.brainwt.notnull()].copy()
```

Explore our mammals dataset

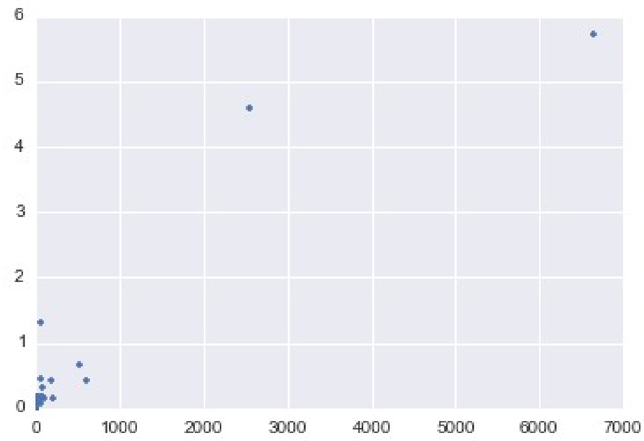
In [64]:
```
mammals.head()
```

Out[64]:

	name	genus	vore	order	conservation	sleep_total	sleep_rem	sleep_cycle	awake	l
1	Owl monkey	Aotus	omni	Primates	NaN	17.0	1.8	NaN	7.0	(
3	Greater short-tailed shrew	Blarina	omni	Soricomorpha	lc	14.9	2.3	0.133333	9.1	(
4	Cow	Bos	herbi	Artiodactyla	domesticated	4.0	0.7	0.666667	20.0	(
8	Dog	Canis	carni	Carnivora	domesticated	10.1	2.9	0.333333	13.9	(
9	Roe deer	Capreolus	herbi	Artiodactyla	lc	3.0	NaN	NaN	21.0	(

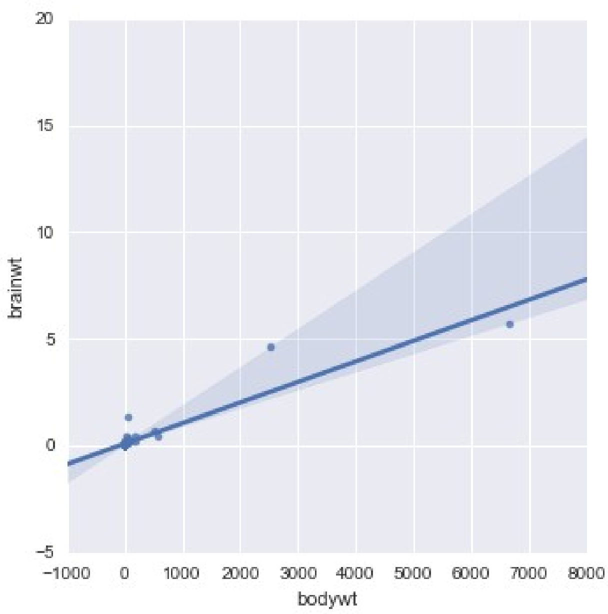
Lets check out a scatter plot of body wieght and brain weight

In [66]:
```
# create a matplotlib figure
plt.figure()
# generate a scatterplot inside the figure
plt.plot(mammals.bodywt, mammals.brainwt, '.')
# show the plot
plt.show()
```



In [67]:
```
sns.lmplot('bodywt', 'brainwt', mammals)
```

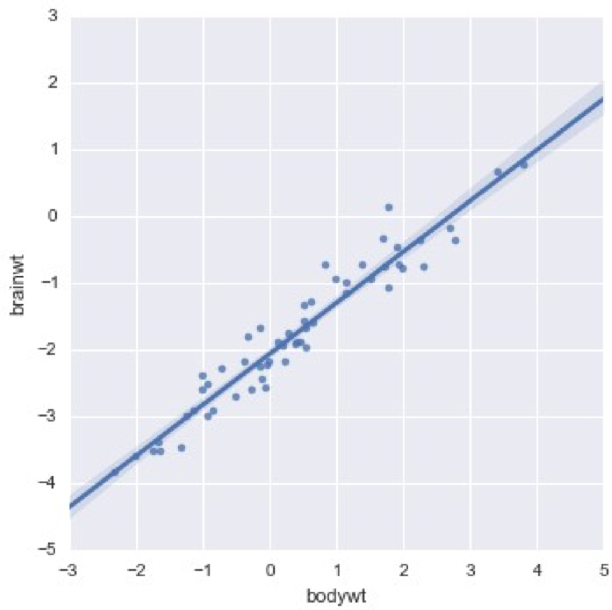
Out[67]: <seaborn.axisgrid.FacetGrid at 0x10ca7fb90>



In [39]:
```
log_columns = ['bodywt', 'brainwt',]
log_mammals = mammals.copy()
log_mammals[log_columns] = log_mammals[log_columns].apply(np.log10)
```

In [40]:
```
sns.lmplot('bodywt', 'brainwt', log_mammals)
```

Out[40]: <seaborn.axisgrid.FacetGrid at 0x10b64ea90>



Guided Practice: Using Seaborn to generate single variable linear model plots (15 mins)

Update and complete the code below to use lmplot and display correlations between body weight and two dependent variables: sleep_rem and awake.

In [ ]:
```
log_columns = ['bodywt', 'brainwt',]  # any others?
log_mammals = mammals.copy()
log_mammals[log_columns] = log_mammals[log_columns].apply(np.log10)
```

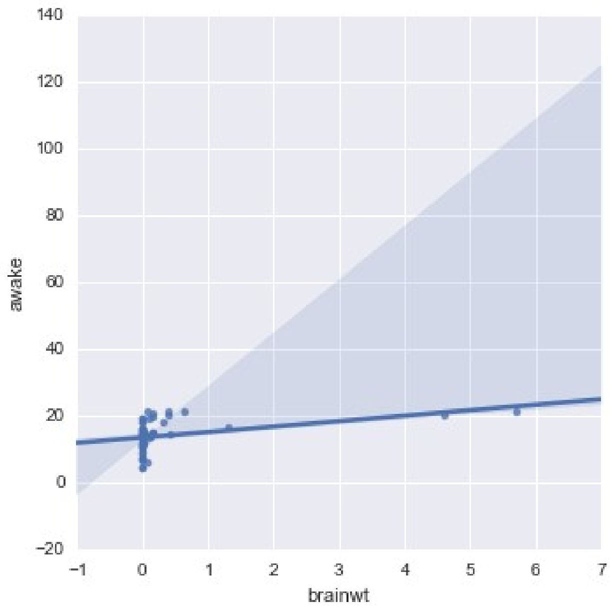
Complete below for sleep_rem and awake as a y, with variables you've already used as x.

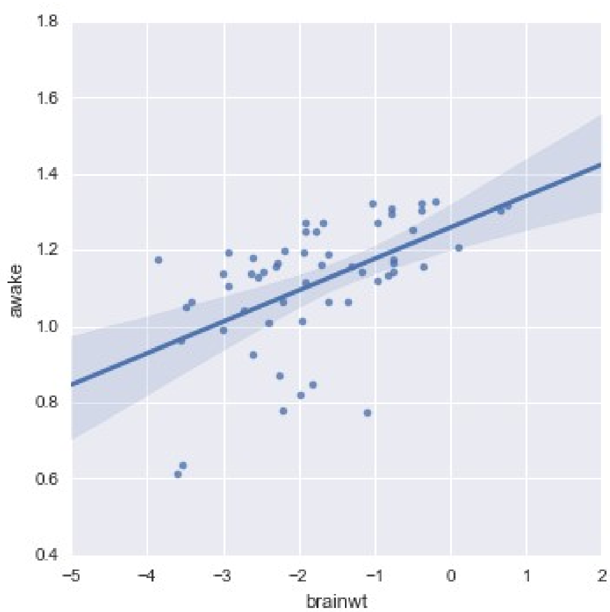
In [ ]:
```
sns.lmplot(x, y, mammals)
sns.lmplot(x, y, log_mammals)
```

Solution:

In [41]:
```
log_columns = ['bodywt', 'brainwt', 'awake', 'sleep_rem']  # any others?
log_mammals = mammals.copy()
log_mammals[log_columns] = log_mammals[log_columns].apply(np.log10)

# one other example, using brainwt and awake.
x = 'brainwt'
y = 'awake'
sns.lmplot(x, y, mammals)
sns.lmplot(x, y, log_mammals)
```

Out[41]: <seaborn.axisgrid.FacetGrid at 0x10b68ae10>





Introduction: Single Regression Analysis in statsmodels & scikit (10 mins)

In [42]:
```
# this is the standard import if you're using "formula notation" (similar to R)
import statsmodels.formula.api as smf

X = mammals[['bodywt']]
y = mammals['brainwt']

# create a fitted model in one line
#formula notiation is the equivalent to writting out our models such that 'outcome
= predictor'
#with the follwing syntax formula = 'outcome ~ predictor1 + predictor2 ... predicto
rN'
lm = smf.ols(formula='y ~ X', data=mammals).fit()
#print the full summary
lm.summary()
```

Out[42]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.872
Model:	OLS	**Adj. R-squared:**	0.870
Method:	Least Squares	**F-statistic:**	367.7
Date:	Thu, 04 Feb 2016	**Prob (F-statistic):**	9.16e-26
Time:	10:28:54	**Log-Likelihood:**	-20.070
No. Observations:	56	**AIC:**	44.14
Df Residuals:	54	**BIC:**	48.19
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
Intercept	0.0859	0.048	1.782	0.080	-0.011 0.183
X	0.0010	5.03e-05	19.176	0.000	0.001 0.001

Omnibus:	85.068	**Durbin-Watson:**	2.376
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1330.630
Skew:	4.258	**Prob(JB):**	1.14e-289
Kurtosis:	25.311	**Cond. No.**	981.

use Statsmodels to make the prediction

In [43]:
```
# you have to create a DataFrame since the Statsmodels formula interface expects it
X_new = pd.DataFrame({'X': [50]})
X_new.head()
```

Out[43]:

	X
0	50

In [44]:
```
lm.predict(X_new)
```

Out[44]:
```
array([ 0.13411477])
```

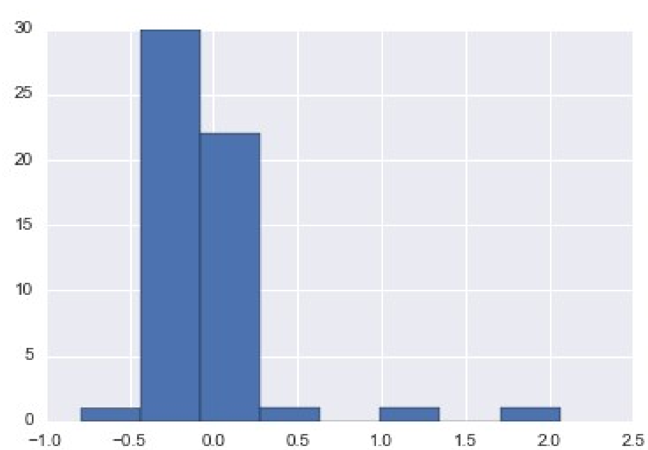
Repeat in Scikit with handy plotting

When modeling with sklearn, you'll use the following base principals.

- All sklearn estimators (modeling classes) are based on this base estimator. This allows you to easily rotate through estimators without changing much code.
- All estimators take a matrix, X, either sparse or dense.
- Many estimators also take a vector, y, when working on a supervised machine learning problem. Regressions are supervised learning because we already have examples of y given X.
- All estimators have parameters that can be set. This allows for customization and higher level of detail to the learning process. The parameters are appropriate to each estimator algorithm.

In [45]:
```
from sklearn import feature_selection, linear_model

def get_linear_model_metrics(X, y, algo):
    # get the pvalue of X given y. Ignore f-stat for now.
    pvals = feature_selection.f_regression(X, y)[1]
    # start with an empty linear regression object
    # .fit() runs the linear regression function on X and y
    algo.fit(X,y)
    residuals = (y-algo.predict(X)).values

    # print the necessary values
    print 'P Values:', pvals
    print 'Coefficients:', algo.coef_
    print 'y-intercept:', algo.intercept_
    print 'R-Squared:', algo.score(X,y)
    plt.figure()
    plt.hist(residuals, bins=np.ceil(np.sqrt(len(y))))
    # keep the model
    return algo

X = mammals[['bodywt']]
y = mammals['brainwt']
lm = linear_model.LinearRegression()
lm = get_linear_model_metrics(X, y, lm)
```

```
P Values: [  9.15540205e-26]
Coefficients: [ 0.00096395]
y-intercept: 0.0859173102936
R-Squared: 0.871949198087
```



Demo: Significance is Key (20 mins)

What does our output tell us?

Our output tells us that:

- The relationship between bodywt and brainwt isn't random (p value approaching 0)
- The model explains, roughly, 87% of the variance of the dataset (the largest errors being in the large brain and body sizes)
- With this current model, brainwt is roughly bodywt * 0.00096395
- The residuals, or error in the prediction, is not normal, with outliers on the right. A better with will have similar to normally distributed error.

Evaluating Fit, Evaluating Sense

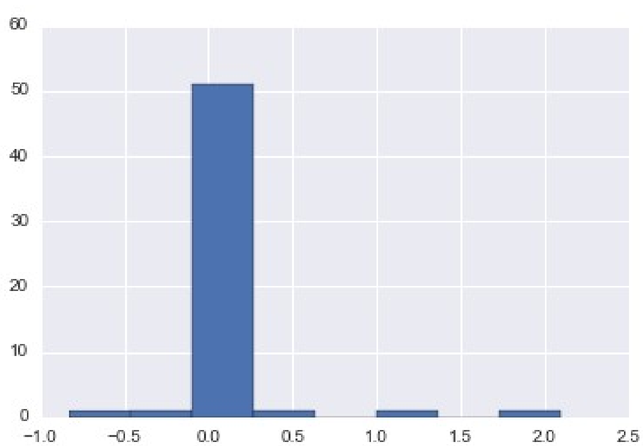
Although we know there is a better solution to the model, we should evaluate some other sense things first. For example, given this model, what is an animal's brainwt if their bodywt is 0?

In [46]:
```
# prediction at 0?
print lm.predict([[0]])
```

```
[ 0.08591731]
```

In [47]:
```
lm = linear_model.LinearRegression(fit_intercept=False)
lm = get_linear_model_metrics(X, y, lm)
# prediction at 0?
print lm.predict([[0]])
```

```
P Values: [  9.15540205e-26]
Coefficients: [ 0.00098291]
y-intercept: 0.0
R-Squared: 0.864418807451
[ 0.]
```



Intrepretation

With linear modeling we call this part of the linear assumption. Consider it a test to the model. If an animal's body weights nothing, we expect their brain to be nonexistent. That given, we can improve the model by telling sklearn's LinearRegression object we do not want to fit a y intercept.

Now, the model fits where brainwt = 0, bodywt = 0. Because we start at 0, the large outliers have a greater effect, so the coefficient has increased. Fitting the this linear assumption also explains slightly less of the variance.

Guided Practice: Using the LinearRegression object (15 mins)

We learned earlier that the the data in its current state does not allow for the best linear regression fit.

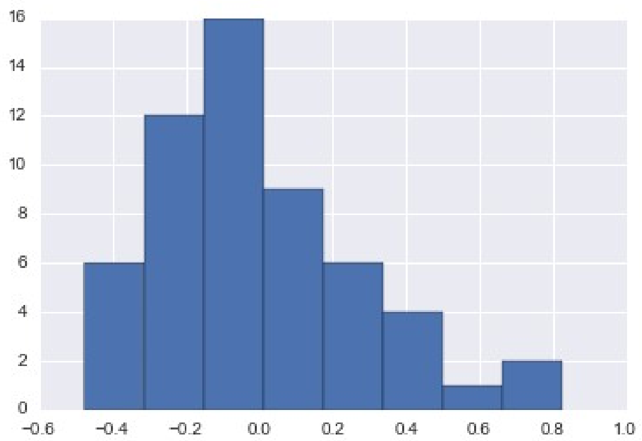
With a partner, generate two more models using the log-transformed data to see how this transform changes the model's performance.

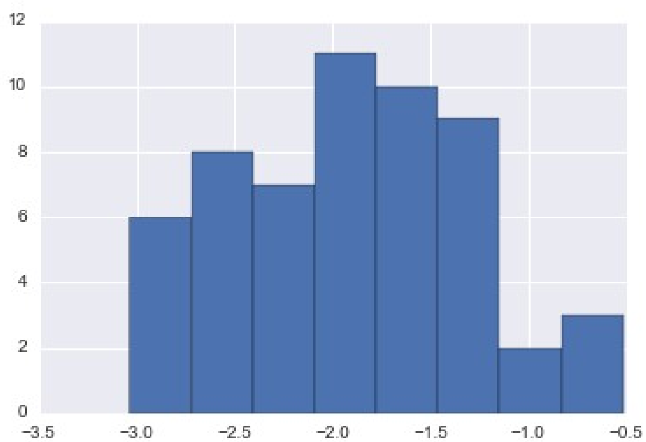
Complete the following code to update X and y to match the log-transformed data. Complete the loop by setting the list to be one True and one False.

In [ ]:
```
#starter
X =
y =
loop = []
for boolean in loop:
    print 'y-intercept:', boolean
    lm = linear_model.LinearRegression(fit_intercept=boolean)
    get_linear_model_metrics(X, y, lm)
    print
```

In [73]:
```
#solution
X = log_mammals[['bodywt']]
y = log_mammals['brainwt']
loop = [True, False]
for boolean in loop:
    print 'y-intercept:', boolean
    lm = linear_model.LinearRegression(fit_intercept=boolean)
    get_linear_model_metrics(X, y, lm)
    print
```

```
y-intercept: True
P Values: [  3.56282243e-33]
Coefficients: [ 0.76516177]
y-intercept: -2.07393164084
R-Squared: 0.931851615367

y-intercept: False
P Values: [  3.56282243e-33]
Coefficients: [ 0.35561441]
y-intercept: 0.0
R-Squared: -2.41053211437
```





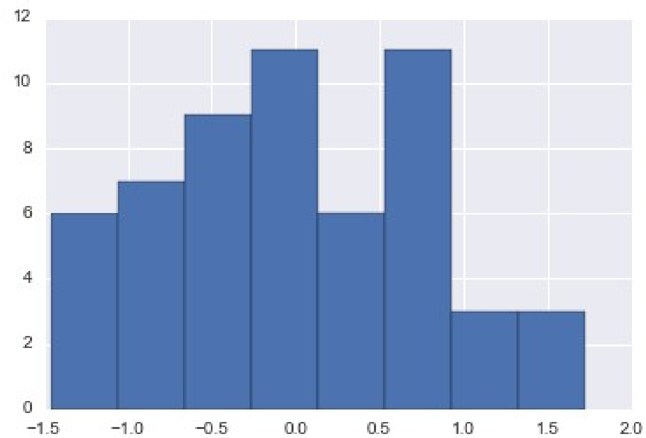
Check: Which model performed the best? The worst? Why?

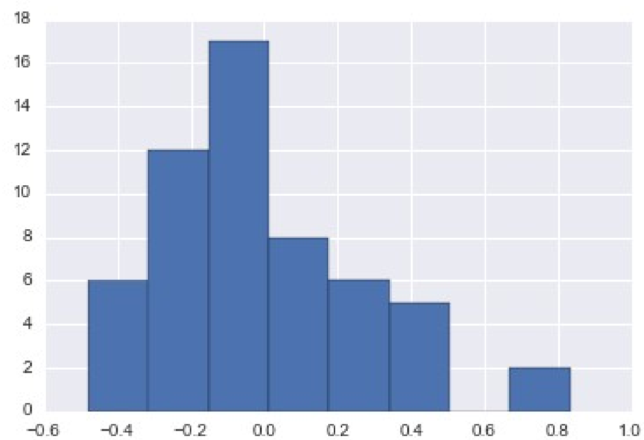
Advanced Methods!

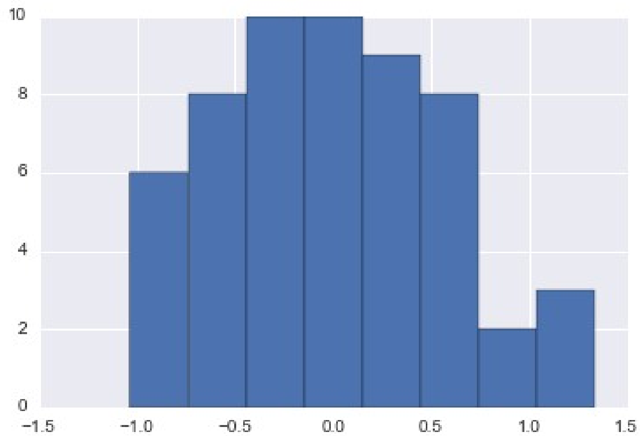
We will go over different estimators in detail in the future but check it out in the docs if you're curious...

In [49]:
```
# loading other sklearn regression estimators
X = log_mammals[['bodywt']]
y = log_mammals['brainwt']

estimators = [
    linear_model.Lasso(),
    linear_model.Ridge(),
    linear_model.ElasticNet(),
]

for est in estimators:
    print est
    get_linear_model_metrics(X, y, est)
    print
```

```
Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
P Values: [  3.56282243e-33]
Coefficients: [ 0.23454772]
y-intercept: -1.85931606304
R-Squared: 0.483728109403

Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, solver='auto', tol=0.001)
P Values: [  3.56282243e-33]
Coefficients: [ 0.75797972]
y-intercept: -2.07102674342
R-Squared: 0.931769516561

ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
      max_iter=1000, normalize=False, positive=False, precompute=False,
      random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
P Values: [  3.56282243e-33]
Coefficients: [ 0.39504621]
y-intercept: -1.9242323166
R-Squared: 0.71382228495
```







Introduction: Multiple Regression Analysis using citi bike data (10 minutes)

In the previous example, one variable explained the variance of another; however, more often than not, we will need multiple variables.

For example, a house's price may be best measured by square feet, but a lot of other variables play a vital role: bedrooms, bathrooms, location, appliances, etc.

For a linear regression, we want these variables to be largely independent of each other, but all of them should help explain the y variable.

We'll work with bikeshare data to showcase what this means and to explain a concept called multicollinearity.

In [76]:
```
wd = '../../assets/dataset/bikeshare/'
bike_data = pd.read_csv(wd+'bikeshare.csv')
bike_data.head()
```

Out[76]:

	instant	dteday	season	yr	mnth	hr	holiday	weekday	workingday	weathersit	temp	atemp
0	1	2011-01-01	1	0	1	0	0	6	0	1	0.24	0.2879
1	2	2011-01-01	1	0	1	1	0	6	0	1	0.22	0.2727
2	3	2011-01-01	1	0	1	2	0	6	0	1	0.22	0.2727
3	4	2011-01-01	1	0	1	3	0	6	0	1	0.24	0.2879
4	5	2011-01-01	1	0	1	4	0	6	0	1	0.24	0.2879

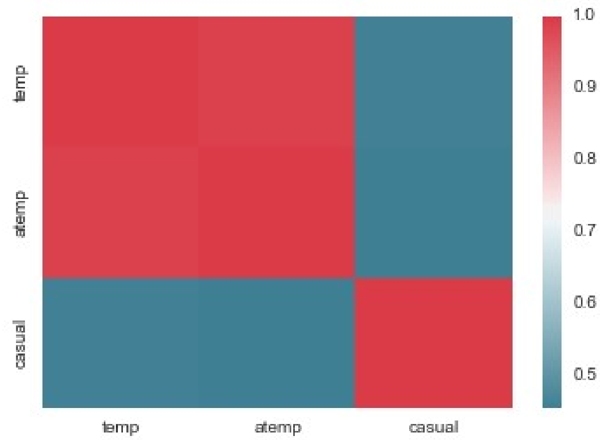
What is Multicollinearity?

With the bike share data, let's compare three data points: actual temperature, "feel" temperature, and guest ridership.

Our data is already normalized between 0 and 1, so we'll start off with the correlations and modeling.

In [79]:
```
cmap = sns.diverging_palette(220, 10, as_cmap=True)

correlations = bike_data[['temp', 'atemp', 'casual']].corr()
print correlations
print sns.heatmap(correlations, cmap=cmap)
```

```
            temp     atemp    casual
temp    1.000000  0.987672  0.459616
atemp   0.987672  1.000000  0.454080
casual  0.459616  0.454080  1.000000
Axes(0.125,0.125;0.62x0.775)
```



The correlation matrix explains that:

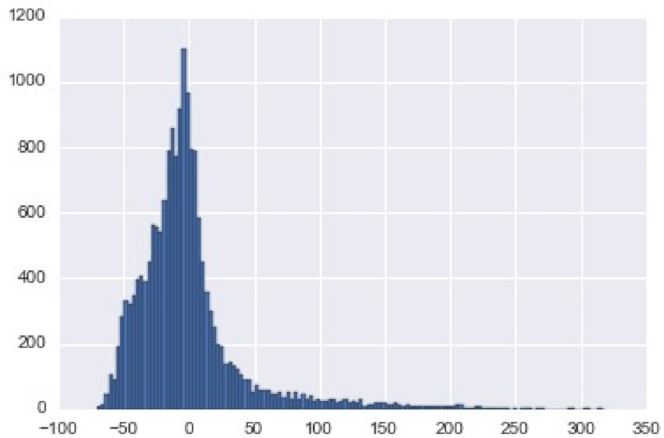
- both temperature fields are moderately correlated to guest ridership;
- the two temperature fields are highly correlated to each other.

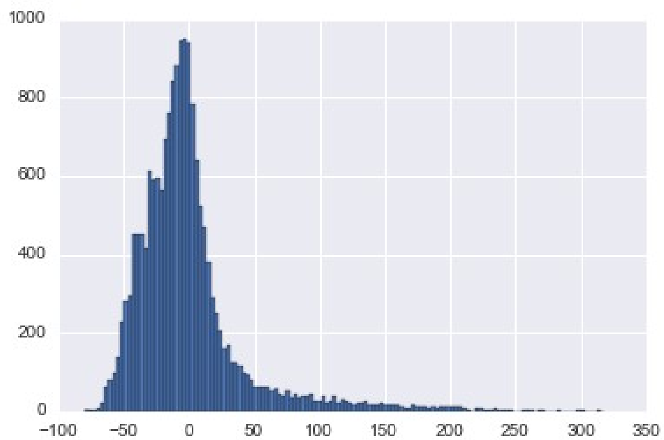
Including both of these fields in a model could introduce a pain point of multicollinearity, where it's more difficult for a model to determine which feature is effecting the predicted value.

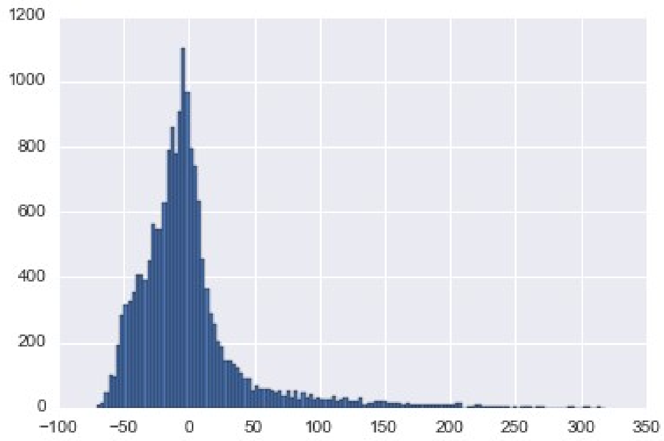
We can measure this effect in the coefficients:

In [80]:
```
y = bike_data['casual']
x_sets = (
    ['temp'],
    ['atemp'],
    ['temp', 'atemp'],
)

for x in x_sets:
    print ', '.join(x)
    get_linear_model_metrics(bike_data[x], y, linear_model.LinearRegression())
    print
```

```
temp
P Values: [ 0.]
Coefficients: [ 117.68705779]
y-intercept: -22.812739188
R-Squared: 0.21124654163

atemp
P Values: [ 0.]
Coefficients: [ 130.27875081]
y-intercept: -26.3071675481
R-Squared: 0.206188705733

temp, atemp
P Values: [ 0.  0.]
Coefficients: [ 116.34021588    1.52795677]
y-intercept: -22.8703398286
R-Squared: 0.21124723661
```







Intrepretation:

Even though the 2-variable model temp + atemp has a higher explanation of variance than two variables on their own, and both variables are considered significant (p values approaching 0), we can see that together, their coefficients are wildly different.

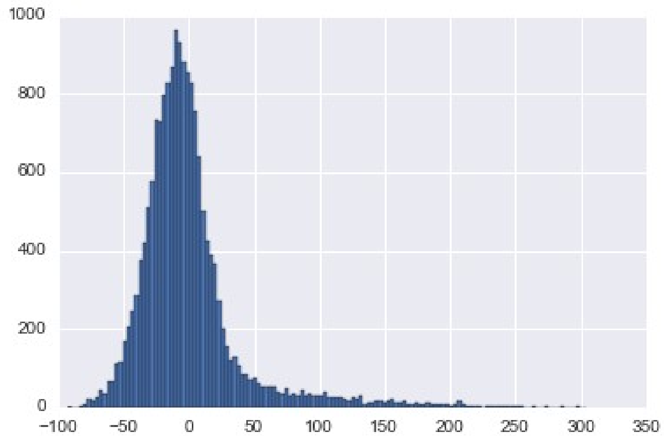
This can introduce error in how we explain models.

What happens if we use a second variable that isn't highly correlated with temperature, like humidity?

In [99]:
```
y = bike_data['casual']
x = bike_data[['temp', 'hum']]
get_linear_model_metrics(x, y, linear_model.LinearRegression())
```

```
P Values: [ 0.  0.]
Coefficients: [ 112.02457031  -80.87301833]
y-intercept: 30.7273338581
R-Squared: 0.310901196913
```

Out[99]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```



Guided Practice: Multicollinearity with dummy variables (15 mins)

There can be a similar effect from a feature set that is a singular matrix, which is when there is a clear relationship in the matrix (for example, the sum of all rows = 1).

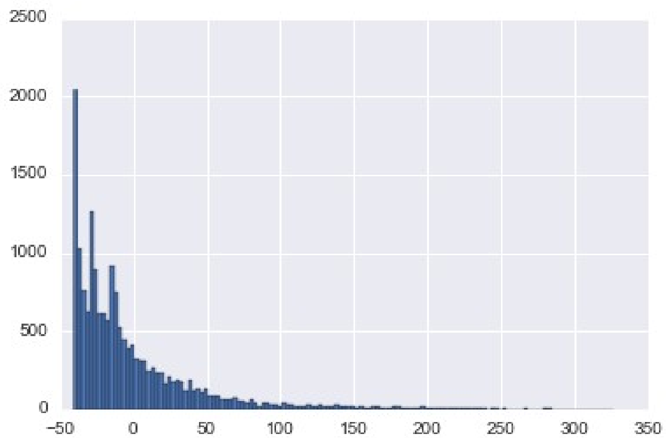
Run through the following code on your own.

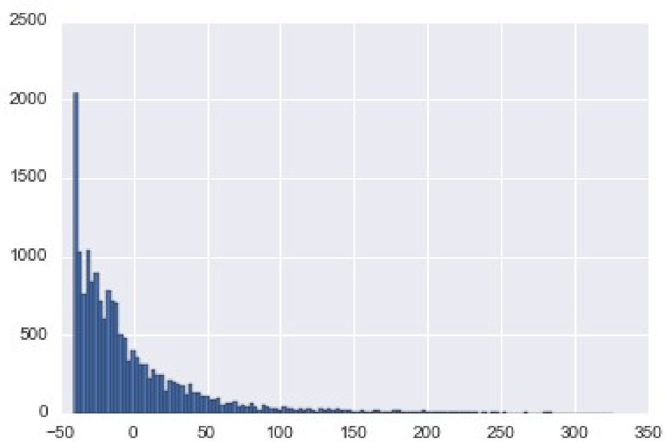
What happens to the coefficients when you include all weather situations instead of just including all except one?

In [100]:
```
lm = linear_model.LinearRegression()
weather = pd.get_dummies(bike_data.weathersit)

get_linear_model_metrics(weather[[1, 2, 3, 4]], y, lm)
print
# drop the least significant, weather situation  = 4
get_linear_model_metrics(weather[[1, 2, 3]], y, lm)
```

```
P Values: [  3.75616929e-73   3.43170021e-22   1.57718666e-55   2.46181288e-01]
Coefficients: [  4.05930101e+12   4.05930101e+12   4.05930101e+12   4.05930101e+
12]
y-intercept: -4.05930100616e+12
R-Squared: 0.0233497737473

P Values: [  3.75616929e-73   3.43170021e-22   1.57718666e-55]
Coefficients: [ 37.87876398  26.92862383  13.38900634]
y-intercept: 2.66666666652
R-Squared: 0.0233906873841
```

Out[100]: `LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)`





Similar in Statsmodels

In [101]:
```
# all dummies in the model
lm_stats = smf.ols(formula='y ~ weather[[1, 2, 3, 4]]', data=bike_data).fit()
lm_stats.summary()
```

Out[101]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.023
Model:	OLS	**Adj. R-squared:**	0.023
Method:	Least Squares	**F-statistic:**	104.0
Date:	Thu, 04 Feb 2016	**Prob (F-statistic):**	1.04e-87
Time:	11:34:52	**Log-Likelihood:**	-92197.
No. Observations:	17379	**AIC:**	1.844e+05
Df Residuals:	17374	**BIC:**	1.844e+05
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[95.0% Conf. Int.]**
Intercept	6.782e+11	1.08e+13	0.063	0.950	-2.06e+13 2.19e+13
weather[[1, 2, 3, 4]][0]	-6.782e+11	1.08e+13	-0.063	0.950	-2.19e+13 2.06e+13
weather[[1, 2, 3, 4]][1]	-6.782e+11	1.08e+13	-0.063	0.950	-2.19e+13 2.06e+13
weather[[1, 2, 3, 4]][2]	-6.782e+11	1.08e+13	-0.063	0.950	-2.19e+13 2.06e+13
weather[[1, 2, 3, 4]][3]	-6.782e+11	1.08e+13	-0.063	0.950	-2.19e+13 2.06e+13

Omnibus:	9002.161	**Durbin-Watson:**	0.136
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	58970.408
Skew:	2.469	**Prob(JB):**	0.00
Kurtosis:	10.554	**Cond. No.**	8.15e+13

In [102]:
```
#droping one
lm_stats = smf.ols(formula='y ~ weather[[1, 2, 3]]', data=bike_data).fit()
lm_stats.summary()
```

Out[102]:

OLS Regression Results

Dep. Variable:	y	**R-squared:**	0.023
Model:	OLS	**Adj. R-squared:**	0.023
Method:	Least Squares	**F-statistic:**	138.7
Date:	Thu, 04 Feb 2016	**Prob (F-statistic):**	8.08e-89
Time:	11:34:53	**Log-Likelihood:**	-92197.
No. Observations:	17379	**AIC:**	1.844e+05
Df Residuals:	17375	**BIC:**	1.844e+05
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
Intercept	2.6667	28.134	0.095	0.924	-52.478 57.812
weather[[1, 2, 3]][0]	37.8788	28.138	1.346	0.178	-17.274 93.031
weather[[1, 2, 3]][1]	26.9286	28.143	0.957	0.339	-28.235 82.092
weather[[1, 2, 3]][2]	13.3890	28.164	0.475	0.635	-41.814 68.592

Omnibus:	9001.632	**Durbin-Watson:**	0.136
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	58962.554
Skew:	2.468	**Prob(JB):**	0.00
Kurtosis:	10.553	**Cond. No.**	189.

Interpretation:

This model makes more sense, because we can more easily explain the variables compared to the one we left out.

For example, this suggests that a clear day (weathersit:1) on average brings in about 38 more riders hourly than a day with heavy snow.

In fact, since the weather situations "degrade" in quality (1 is the nicest day, 4 is the worst), the coefficients now reflect that well.

However at this point, there is still a lot of work to do, because weather on its own fails to explain ridership well.

Guided Practice: Combining non-correlated features into a better model (15 mins)

In [103]:
```
bike_data.dtypes
```

Out[103]:
```
instant          int64
dteday          object
season           int64
yr               int64
mnth             int64
hr               int64
holiday          int64
weekday          int64
workingday       int64
weathersit       int64
temp           float64
atemp          float64
hum            float64
windspeed      float64
casual           int64
registered       int64
cnt              int64
dtype: object
```

With a partner, complete this code together and visualize the correlations of all the numerical features built into the data set.

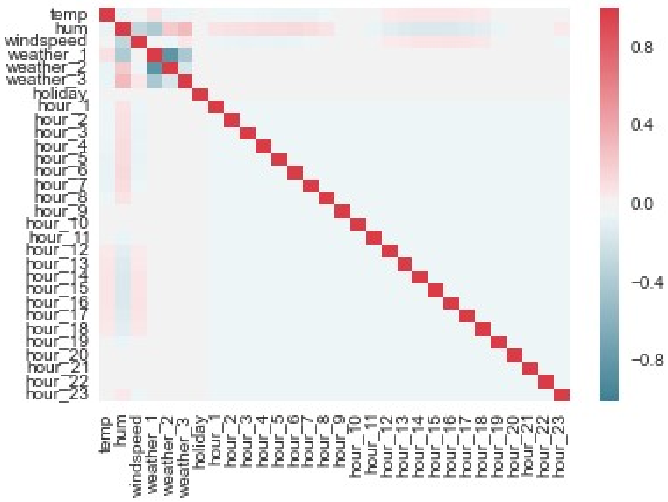
We want to:

- Add the three significant weather situations into our current model
- Find two more features that are not correlated with current features, but could be strong indicators for predicting guest riders.

In [ ]:
```
#starter
lm = linear_model.LinearRegression()
bikemodel_data = bike_data.join() # add in the three weather situations

cmap = sns.diverging_palette(220, 10, as_cmap=True)
correlations = # what are we getting the correlations of?
print correlations
print sns.heatmap(correlations, cmap=cmap)

columns_to_keep = [] #[which_variables?]
final_feature_set = bikemodel_data[columns_to_keep]

get_linear_model_metrics(final_feature_set, y, lm)
```

In [123]:
```
#solution
lm = linear_model.LinearRegression()
weather = pd.get_dummies(bike_data.weathersit)
weather.columns = ['weather_' + str(i) for i in weather.columns]

hours = pd.get_dummies(bike_data.hr)
hours.columns = ['hour_' + str(i) for i in hours.columns]

season = pd.get_dummies(bike_data.season)
season.columns = ['season_' + str(i) for i in season.columns]


bikemodel_data = bike_data.join(weather) # add in the three weather situations
bikemodel_data = bikemodel_data.join(hours)
bikemodel_data = bikemodel_data.join(season)

cmap = sns.diverging_palette(220, 10, as_cmap=True)

columns_to_keep = ['temp', 'hum', 'windspeed', 'weather_1', 'weather_2', 'weather_
3', 'holiday',]
columns_to_keep.extend(['hour_' + str(i) for i in range(1, 24)])

correlations = bikemodel_data[columns_to_keep].corr()
print correlations
print sns.heatmap(correlations, cmap=cmap)
```

```
                temp       hum  windspeed  weather_1  weather_2  weather_3  \
temp        1.000000 -0.069881  -0.023125   0.101044  -0.069657  -0.062406
hum        -0.069881  1.000000  -0.290105  -0.383425   0.220758   0.309737
windspeed  -0.023125 -0.290105   1.000000   0.005150  -0.049241   0.070018
weather_1   0.101044 -0.383425   0.005150   1.000000  -0.822961  -0.412414
weather_2  -0.069657  0.220758  -0.049241  -0.822961   1.000000  -0.177417
weather_3  -0.062406  0.309737   0.070018  -0.412414  -0.177417   1.000000
holiday    -0.027340 -0.010588   0.003988   0.009167   0.004910  -0.023664
hour_1     -0.040738  0.083197  -0.053580   0.008819  -0.006750  -0.005379
hour_2     -0.045627  0.096198  -0.060241   0.005156  -0.003921  -0.002518
hour_3     -0.046575  0.108659  -0.065444  -0.001685   0.003843  -0.003117
hour_4     -0.053459  0.121990  -0.057285  -0.000450   0.000506   0.000096
hour_5     -0.065571  0.124406  -0.067411  -0.004791   0.011541  -0.010083
hour_6     -0.069911  0.126481  -0.055217  -0.014011   0.017969  -0.004410
hour_7     -0.062825  0.112289  -0.044717  -0.020841   0.015641   0.011168
hour_8     -0.045570  0.081720  -0.023117  -0.022657   0.025452  -0.001427
hour_9     -0.021986  0.037325   0.001989  -0.029315   0.035263  -0.005625
hour_10     0.003896 -0.012090   0.020399  -0.020236   0.026106  -0.006675
hour_11     0.027808 -0.060432   0.029448  -0.018420   0.028068  -0.012973
hour_12     0.047007 -0.098114   0.044294  -0.021224   0.025918  -0.004659
hour_13     0.062752 -0.125421   0.053938  -0.009517   0.011360  -0.001596
hour_14     0.073992 -0.141266   0.072461  -0.002867   0.002216   0.001548
hour_15     0.077838 -0.146532   0.077046   0.003782  -0.008235   0.006789
hour_16     0.073918 -0.142656   0.080822   0.018486  -0.026678   0.009842
hour_17     0.062626 -0.123506   0.074068   0.016674  -0.028636   0.017174
hour_18     0.047992 -0.098888   0.059114   0.013256  -0.021142   0.010026
hour_19     0.029525 -0.059376   0.034269   0.018700  -0.019835  -0.000463
hour_20     0.012609 -0.027918   0.008759   0.025354  -0.032907   0.008977
hour_21    -0.001830  0.004671  -0.015770   0.021120  -0.021142  -0.002561
hour_22    -0.013554  0.028089  -0.026419   0.018700  -0.017220  -0.004659
hour_23    -0.023847  0.049900  -0.043234   0.008417  -0.013952   0.007928

             holiday    hour_1    hour_2    hour_3    ...      hour_14  \
temp       -0.027340 -0.040738 -0.045627 -0.046575    ...     0.073992
hum        -0.010588  0.083197  0.096198  0.108659    ...    -0.141266
windspeed   0.003988 -0.053580 -0.060241 -0.065444    ...     0.072461
weather_1   0.009167  0.008819  0.005156 -0.001685    ...    -0.002867
weather_2   0.004910 -0.006750 -0.003921  0.003843    ...     0.002216
weather_3  -0.023664 -0.005379 -0.002518 -0.003117    ...     0.001548
holiday     1.000000  0.000293  0.000744 -0.003602    ...     0.000045
hour_1      0.000293  1.000000 -0.043188 -0.042618    ...    -0.043627
hour_2      0.000744 -0.043188  1.000000 -0.042340    ...    -0.043343
hour_3     -0.003602 -0.042618 -0.042340  1.000000    ...    -0.042771
hour_4     -0.000093 -0.042618 -0.042340 -0.041782    ...    -0.042771
hour_5      0.000643 -0.043251 -0.042969 -0.042402    ...    -0.043406
hour_6      0.000244 -0.043502 -0.043219 -0.042648    ...    -0.043658
hour_7      0.000144 -0.043564 -0.043281 -0.042710    ...    -0.043721
hour_8      0.000144 -0.043564 -0.043281 -0.042710    ...    -0.043721
hour_9      0.000144 -0.043564 -0.043281 -0.042710    ...    -0.043721
hour_10     0.000144 -0.043564 -0.043281 -0.042710    ...    -0.043721
hour_11     0.000144 -0.043564 -0.043281 -0.042710    ...    -0.043721
hour_12     0.000095 -0.043596 -0.043312 -0.042740    ...    -0.043752
hour_13     0.000045 -0.043627 -0.043343 -0.042771    ...    -0.043784
hour_14     0.000045 -0.043627 -0.043343 -0.042771    ...     1.000000
hour_15     0.000045 -0.043627 -0.043343 -0.042771    ...    -0.043784
hour_16    -0.000004 -0.043658 -0.043374 -0.042802    ...    -0.043815
hour_17    -0.000004 -0.043658 -0.043374 -0.042802    ...    -0.043815
hour_18     0.000095 -0.043596 -0.043312 -0.042740    ...    -0.043752
hour_19     0.000095 -0.043596 -0.043312 -0.042740    ...    -0.043752
hour_20     0.000095 -0.043596 -0.043312 -0.042740    ...    -0.043752
hour_21     0.000095 -0.043596 -0.043312 -0.042740    ...    -0.043752
hour_22     0.000095 -0.043596 -0.043312 -0.042740    ...    -0.043752
hour_23     0.000095 -0.043596 -0.043312 -0.042740    ...    -0.043752
```



Independent Practice: Building models for other y variables (25 minutes)

We've completely a model together that explains casual guest riders. Now it's your turn to build another model, using a different y variable: registered riders.

Pay attention to:

- the distribution of riders (should we rescale the data?)
- checking correlations with variables and registered riders
- having a feature space (our matrix) with low multicollinearity
- model complexity vs explanation of variance: at what point do features in a model stop improving r-squared?
- the linear assumption -- given all feature values being 0, should we have no ridership? negative ridership? positive ridership?

Bonus

- Which variables would make sense to dummy (because they are categorical, not continuous)?
- What features might explain ridership but aren't included in the data set?
- Is there a way to build these using pandas and the features available?
- Outcomes: If your model at least improves upon the original model and the explanatory effects (coefficients) make sense, consider this a complete task.

If your model has an r-squared above .4, this a relatively effective model for the data available. Kudos!

In [111]:
```
bikemodel_data.columns
```

Out[111]:
```
Index([u'instant', u'dteday', u'season', u'yr', u'mnth', u'hr', u'holiday',
       u'weekday', u'workingday', u'weathersit', u'temp', u'atemp', u'hum',
       u'windspeed', u'casual', u'registered', u'cnt', u'weather_1',
       u'weather_2', u'weather_3', u'weather_4', u'hour_0', u'hour_1',
       u'hour_2', u'hour_3', u'hour_4', u'hour_5', u'hour_6', u'hour_7',
       u'hour_8', u'hour_9', u'hour_10', u'hour_11', u'hour_12', u'hour_13',
       u'hour_14', u'hour_15', u'hour_16', u'hour_17', u'hour_18', u'hour_19',
       u'hour_20', u'hour_21', u'hour_22', u'hour_23', u'season_1',
       u'season_2', u'season_3', u'season_4'],
      dtype='object')
```

In [124]:
```
y = bike_data['registered']
log_y = np.log10(y+1)
lm = smf.ols(formula=' log_y ~ temp + hum + windspeed + weather_1 + weather_2 + we
ather_3 + holiday + hour_1 + hour_2 + hour_3 + hour_4 + hour_5 + hour_6 + hour_7 +
hour_8 + hour_9 + hour_10 + hour_11 + hour_12 + hour_13 + hour_14 + hour_15 + hour
_16 + hour_18 + hour_19 + hour_20 + hour_21 + hour_22 + hour_23', data=bikemodel_d
ata).fit()
#print the full summary
lm.summary()
```

Out[124]:

OLS Regression Results

Dep. Variable:	log_y	**R-squared:**	0.722
Model:	OLS	**Adj. R-squared:**	0.721
Method:	Least Squares	**F-statistic:**	1553.
Date:	Thu, 04 Feb 2016	**Prob (F-statistic):**	0.00
Time:	11:44:24	**Log-Likelihood:**	-4868.9
No. Observations:	17379	**AIC:**	9798.
Df Residuals:	17349	**BIC:**	1.003e+04
Df Model:	29		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[95.0% Conf. Int.]
Intercept	1.8137	0.186	9.747	0.000	1.449 2.178
temp	0.7374	0.013	56.233	0.000	0.712 0.763
hum	-0.2402	0.016	-14.574	0.000	-0.273 -0.208
windspeed	-0.0988	0.021	-4.644	0.000	-0.140 -0.057
weather_1	0.0102	0.185	0.055	0.956	-0.353 0.373
weather_2	0.0196	0.185	0.106	0.916	-0.344 0.383
weather_3	-0.1737	0.185	-0.937	0.349	-0.537 0.190
holiday	-0.1262	0.015	-8.672	0.000	-0.155 -0.098
hour_1	-0.7016	0.015	-47.740	0.000	-0.730 -0.673
hour_2	-0.9087	0.015	-61.469	0.000	-0.938 -0.880
hour_3	-1.1141	0.015	-74.600	0.000	-1.143 -1.085
hour_4	-1.2190	0.015	-81.464	0.000	-1.248 -1.190
hour_5	-0.7704	0.015	-51.897	0.000	-0.799 -0.741
hour_6	-0.2697	0.015	-18.240	0.000	-0.299 -0.241
hour_7	0.1413	0.015	9.600	0.000	0.112 0.170
hour_8	0.4064	0.015	27.720	0.000	0.378 0.435
hour_9	0.2346	0.015	16.069	0.000	0.206 0.263
hour_10	0.0358	0.015	2.461	0.014	0.007 0.064
hour_11	0.0696	0.015	4.771	0.000	0.041 0.098
hour_12	0.1510	0.015	10.334	0.000	0.122 0.180
hour_13	0.1287	0.015	8.789	0.000	0.100 0.157
hour_14	0.0767	0.015	5.226	0.000	0.048 0.105
hour_15	0.1050	0.015	7.149	0.000	0.076 0.134
hour_16	0.2418	0.015	16.491	0.000	0.213 0.271
hour_18	0.4222	0.015	28.917	0.000	0.394 0.451
hour_19	0.3061	0.015	21.029	0.000	0.278 0.335

In [ ]:

```

```