2.4.3+The+Extraordinary+Power+of+Explanatory+Power

August 28, 2018

In [1]:
```
import math
import warnings

from IPython.display import display
from matplotlib import pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn import linear_model
import statsmodels.formula.api as smf

# Display preferences.
%matplotlib inline
pd.options.display.float_format = '{:.3f}'.format

# Suppress annoying harmless error.
warnings.filterwarnings(
    action="ignore",
    module="scipy",
    message="^internal gelsd"
)
```

0.1 The Extraordinary Power of Explanatory Power

The strength of multiple linear regression lies in its ability to provide straightforward and interpretable solutions that not only predict future outcomes, but also provide insight into the underlying processes that create these outcomes. For example, after fitting the following model:

$$HourlyWidgetProduction = \alpha + \beta_1 WorkerAgeFrom18 + \beta_2 WorkerYearsinJob + \beta_3 IsRoundWidget$$

we get these parameters:

$$\alpha = 2$$

$$\beta_1 = .1$$

$$\beta_2 = .2$$

$$\beta_3 = 4$$

Using those parameters, we learn that round widgets are twice as fast to produce as non-round widgets. We can tell because α represents the intercept, the hourly rate of production for widgets that are not round (2 an hour) and β_3 represents the difference between the intercept and the hourly rate of production for round widgets (also 2 an hour, for a total of 4 round widgets an hour).

We also learn that for every year a worker ages after the age of 18, their hourly production-rate goes up by .1 (β_1). In addition, for every year a worker has been in that job, their hourly production-rate goes up by .2 (β_2).

Furthermore, using this model, we can predict that a 20-year-old worker who has been in the job for a year and is making only round widgets will make $2 + .1 * 2 + .2 * 1 + 4 = 6.3$ round widgets an hour.

Finally, and probably of greatest interest, we get an **R-Squared** value. This is a proportion (between 0 and 1) that expresses how much variance in the outcome variable our model was able to explain. Higher R^2 values are better to a point– a low R^2 indicates that our model isn't explaining much information about the outcome, which means it will not give very good predictions. However, a very high R^2 is a warning sign for overfitting. No dataset is a perfect representation of reality, so a model that perfectly fits our data (R^2 of 1 or close to 1) is likely to be biased by quirks in the data, and will perform less well on the test-set.

Here's an example using a toy advertising dataset:

```
In [61]: # Acquire, load, and preview the data.
         data = pd.read_csv('https://tf-curricula-prod.s3.amazonaws.com/data-science/Advertising
         display(data.head())
         data.to_csv('sales.csv')
         # Instantiate and fit our model.
         regr = linear_model.LinearRegression()
         Y = data['Sales'].values.reshape(-1, 1)
         X = data[['TV','Radio','Newspaper']]
         regr.fit(X, Y)

         # Inspect the results.
         print('\nCoefficients: \n', regr.coef_)
         print('\nIntercept: \n', regr.intercept_)
         print('\nR-squared:')
         print(regr.score(X, Y))
```

	Unnamed: 0	TV	Radio	Newspaper	Sales
0	1	230.100	37.800	69.200	22.100
1	2	44.500	39.300	45.100	10.400
2	3	17.200	45.900	69.300	9.300
3	4	151.500	41.300	58.500	18.500
4	5	180.800	10.800	58.400	12.900

```
Coefficients:
 [[ 0.04576465  0.18853002 -0.00103749]]
```

```
Intercept:
 [2.93888937]

R-squared:
0.8972106381789521
```

The model where the outcome Sales is predicted by the features TV, Radio, and Newspaper explains 89.7% of the variance in Sales. Note that we don't know from these results how much of that variance is explained by each of the three features. Looking at the coefficients, there appears to be a base rate of Sales that happen even with no ads in any medium (intercept: 2.939) and sales have the highest per-unit increase when ads are on the radio (0.189).

0.2 Assumptions of Multivariable Linear Regression

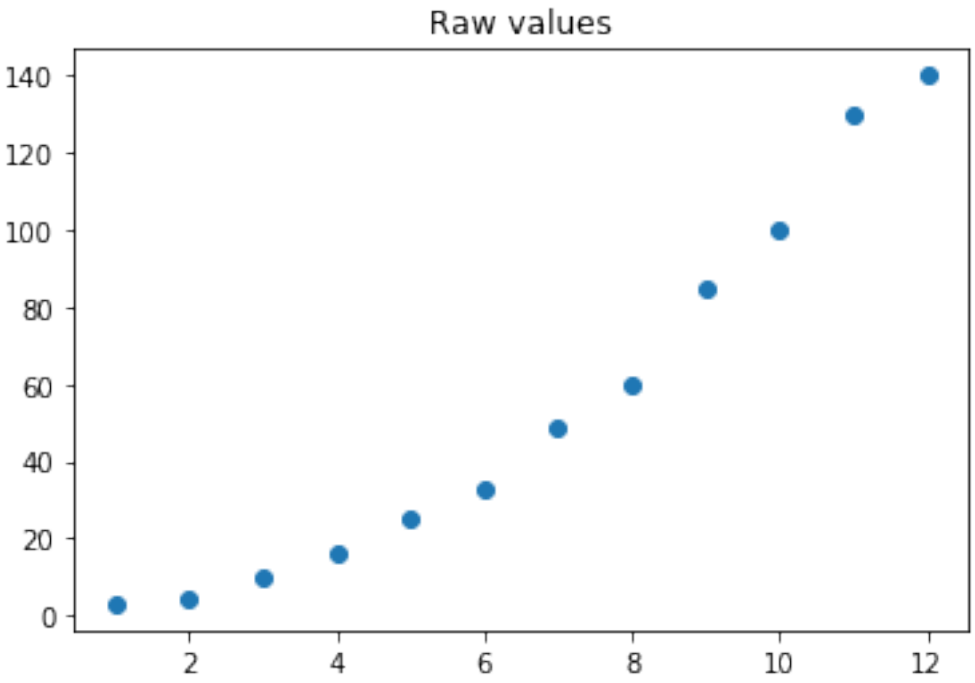
For regression to work its magic, inputs to the model need to be consistent with four assumptions:

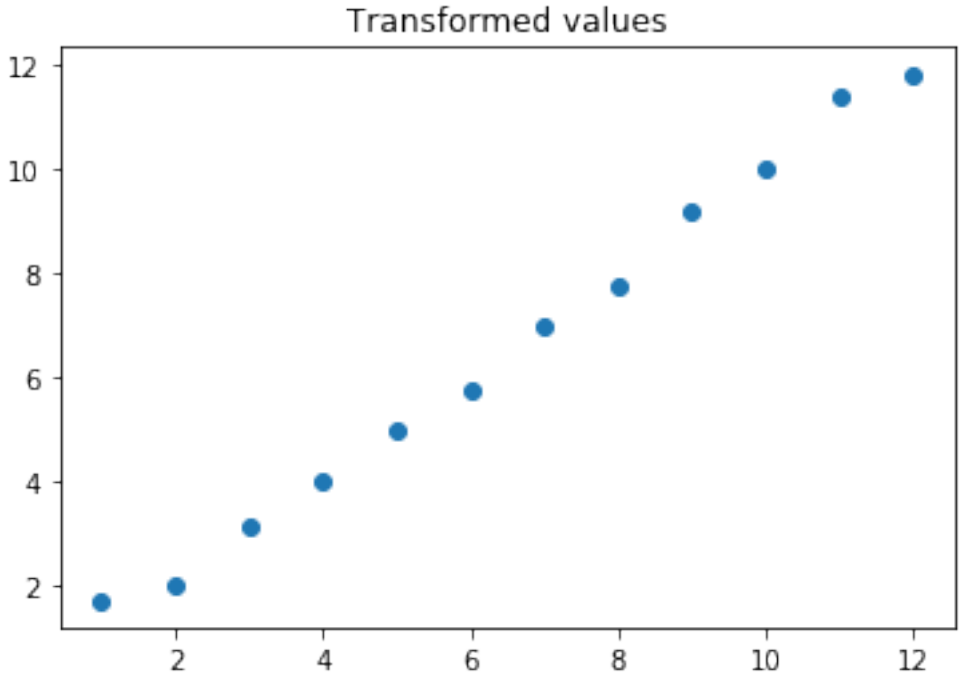
0.2.1 Assumption one: linear relationship

As mentioned earlier, features in a regression need to have a linear relationship with the outcome. If the relationship is non-linear, the regression model will try to find any hint of a linear relationship, and only explain that – with predictable consequences for the validity of the model.

Sometimes this can be fixed by applying a non-linear transformation function to a feature. For example, if the relationship between feature and outcome is quadratic and all feature scores are > 0, we can take the square root of the features, resulting in a linear relationship between the outcome and sqrt(feature).

```
In [3]: # Sample data.
        outcome = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
        feature = [3, 4, 10, 16, 25, 33, 49, 60, 85, 100, 130, 140]

        # Plot the data as-is. Looks a mite quadratic.
        plt.scatter(outcome, feature)
        plt.title('Raw values')
        plt.show()

        # Create a feature using a non-linear transformation.
        sqrt_feature = [math.sqrt(x) for x in  feature]


        # Well now isn't that nice.
        plt.scatter(outcome, sqrt_feature)
        plt.title('Transformed values')
        plt.show()
```





When interpreting features with non-linear transformations, it is important to keep the transformation in mind. For example, in the equation $y = 2log(x)$, y increases by one unit for every

two-unit increase in $log(x)$. The relationship between y and x, however, is non-linear, and the amount of change in y varies based on the absolute value of x:

x	log(x)	y
1	0	0
10	1	2
100	2	4
1000	3	6

So a one-unit change in x from 1 to 2 will result in a much greater change in y than a one-unit change in x from 100 to 101.

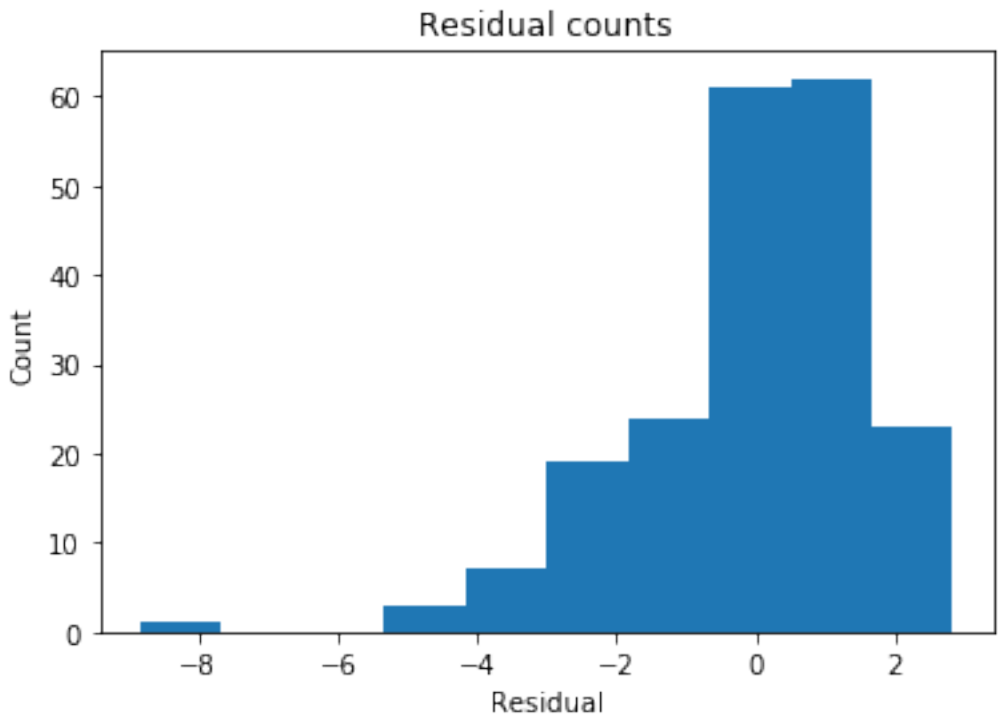
There are many variable transformations. For a deep dive, check out the Variable Linearization section of Fifty Ways to Fix Your Data.

0.2.2 Assumption two: multivariate normality

The error from the model (calculated by subtracting the model-predicted values from the real outcome values) should be normally distributed. Since ordinary least squares regression models are fitted by choosing the parameters that best minimize error, skewness or outliers in the error can result in serious miss-estimations.

Outliers or skewness in error can often be traced back to outliers or skewness in data.

```
In [4]: # Extract predicted values.
        predicted = regr.predict(X).ravel()
        actual = data['Sales']

        # Calculate the error, also called the residual.
        residual = actual - predicted

        # This looks a bit concerning.
        plt.hist(residual)
        plt.title('Residual counts')
        plt.xlabel('Residual')
        plt.ylabel('Count')
        plt.show()
```



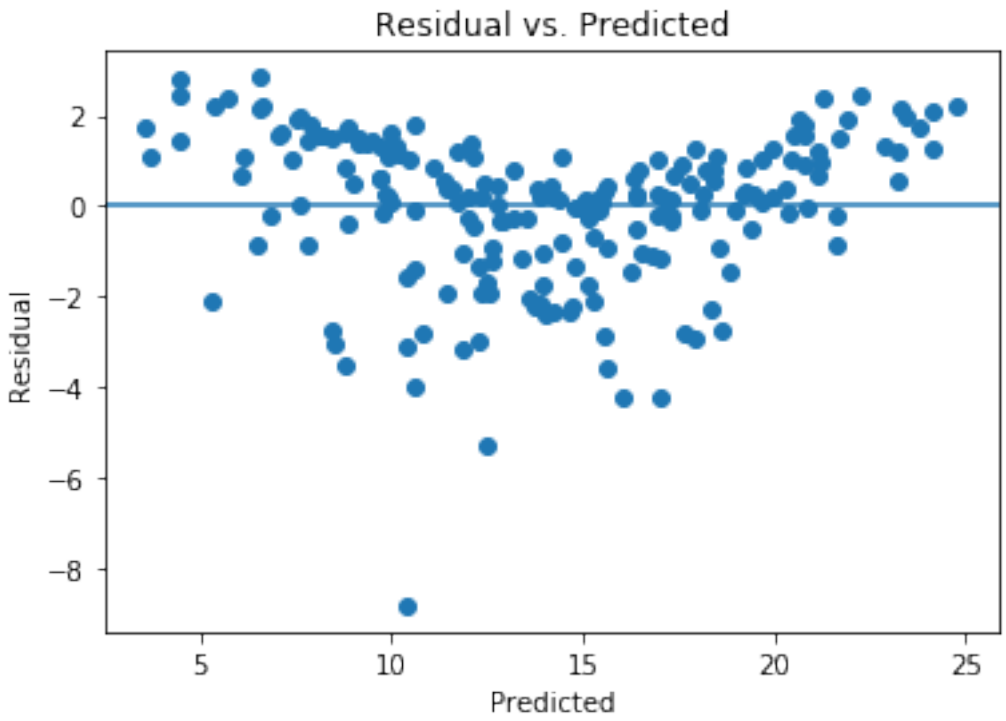
0.2.3 Assumption three: homoscedasticity

The distribution of your error terms (its "scedasticity"), should be consistent for all predicted values, or **homoscedastic**.

For example, if your error terms aren't consistently distributed and you have more variance in the error for large outcome values than for small ones, then the confidence interval for large predicted values will be too small because it will be based on the average error variance. This leads to overconfidence in the accuracy of your model's predictions.

Some fixes to heteroscedasticity include transforming the dependent variable and adding features that target the poorly-estimated areas. For example, if a model tracks data over time and model error variance jumps in the September to November period, a binary feature indicating season may be enough to resolve the problem.

```
In [5]: plt.scatter(predicted, residual)
        plt.xlabel('Predicted')
        plt.ylabel('Residual')
        plt.axhline(y=0)
        plt.title('Residual vs. Predicted')
        plt.show()

        # Hm... looks a bit concerning.
```



0.2.4 Assumption four: low multicollinearity

Correlations among features should be low or nonexistent. When features are correlated, they may both explain the same pattern of variance in the outcome. The model will attempt to find a solution, potentially by attributing half the explanatory power to one feature and half to the other. This isn't a problem if our only goal is prediction, because then all that matters is that the variance gets explained. However, if we want to know which features matter most when predicting an outcome, multicollinearity can cause us to underestimate the relationship between features and outcomes.

Multicollinearity can be fixed by PCA or by discarding some of the correlated features.

```
In [6]: correlation_matrix = X.corr()
        display(correlation_matrix)

        X_all = data[['TV','Radio','Newspaper', 'Sales']]
        correlation_matrix2 = X_all.corr()
        display(correlation_matrix2)
```

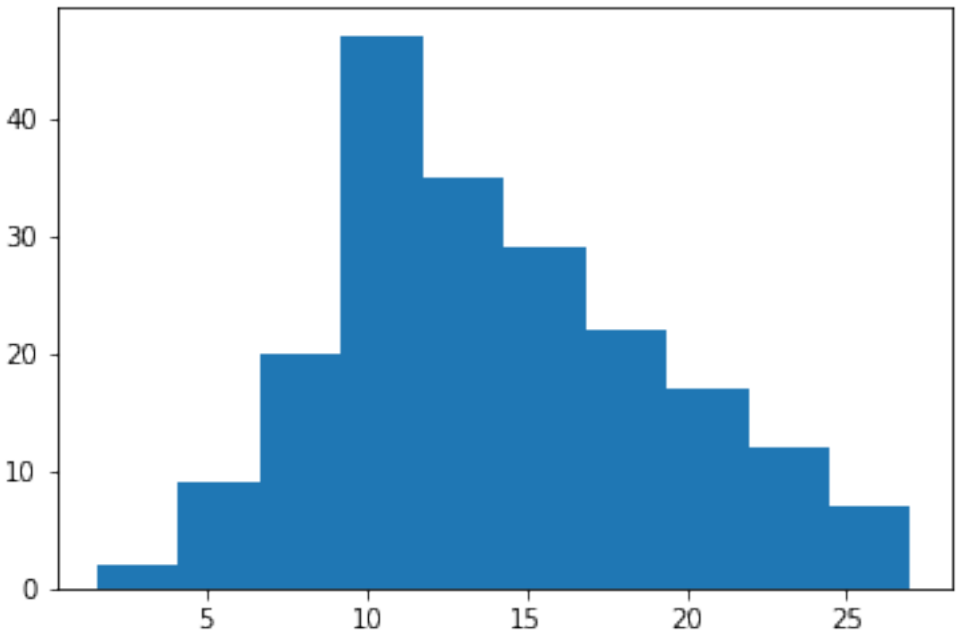
```
                TV  Radio  Newspaper
TV           1.000  0.055      0.057
Radio        0.055  1.000      0.354
Newspaper    0.057  0.354      1.000
```

```
              TV  Radio  Newspaper  Sales
TV         1.000  0.055      0.057  0.782
Radio      0.055  1.000      0.354  0.576
Newspaper  0.057  0.354      1.000  0.228
Sales      0.782  0.576      0.228  1.000
```

0.3 Drill: fixing assumptions

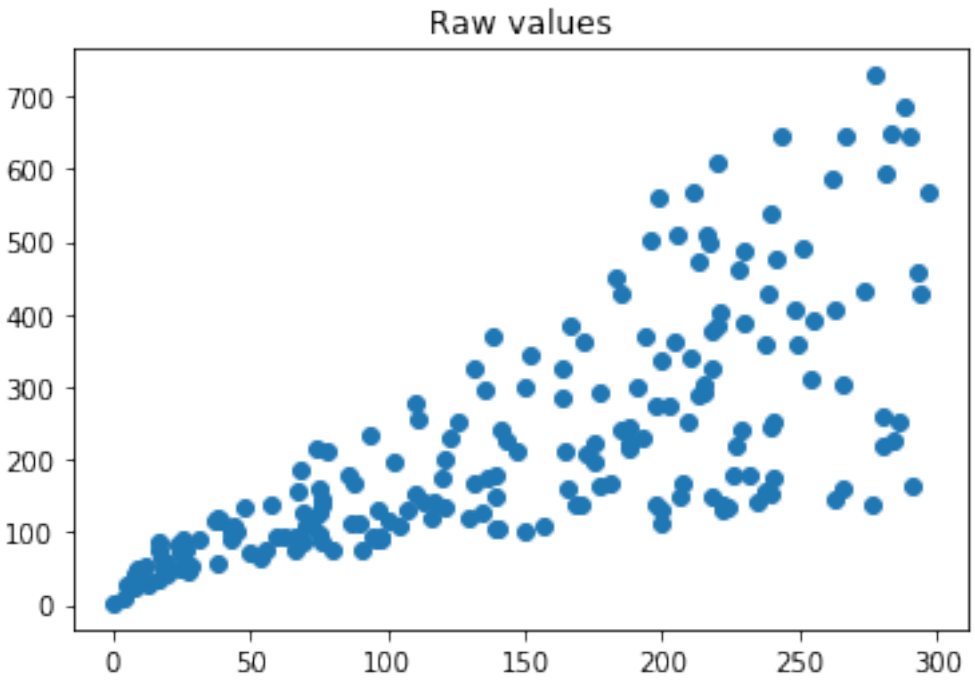
Judging from the diagnostic plots, your data has a problem with both heteroscedasticity and multivariate non-normality. Use the cell(s) below to see what you can do to fix it.

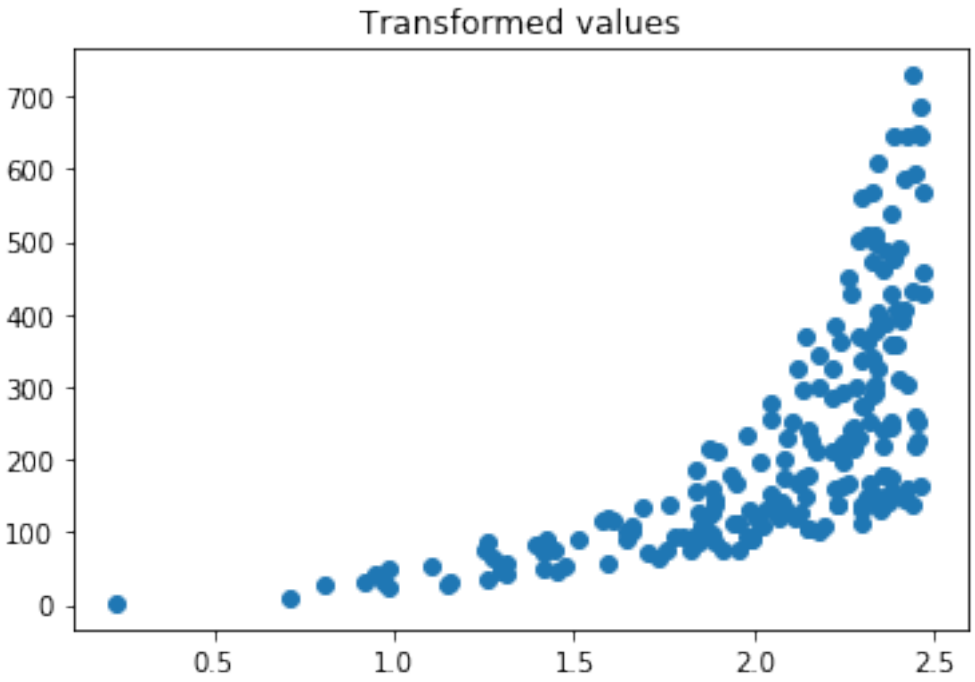
In [7]:
```
# Your code here.

plt.hist(data['Sales'])
plt.show()

#Newspaper is a problem.
```



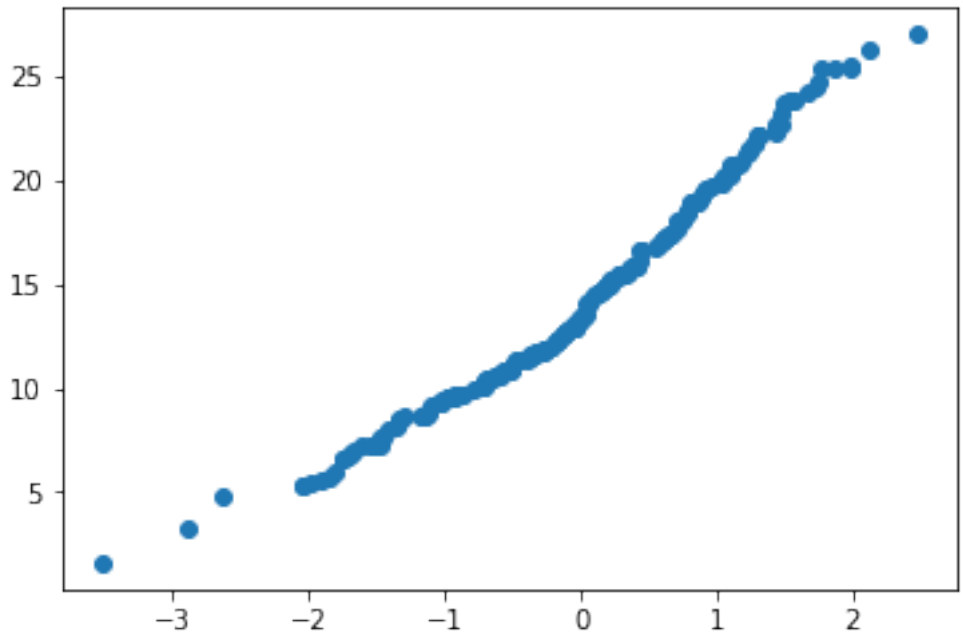
In [16]:
```
import math
outcome = data['Sales'] ** 2
feature = data['TV']

# Plot the data as-is. Looks a mite quadratic.
plt.scatter(feature, outcome)
plt.title('Raw values')
```

```
plt.show()
# plt.hist(feature)
plt.show()
# Create a feature using a non-linear transformation.
log_feature = [math.log10(1+x) for x in  feature]
# log_feature = [x ** .5 for x in  feature]
# log_outcome = [x ** 2 for x in  outcome]

# Well now isn't that nice.
plt.scatter(log_feature, outcome)
plt.title('Transformed values')
plt.show()
# plt.hist(log_feature)
# plt.show()


# plt.hist(feature)

# # Create a feature using a non-linear transformation.
# log_feature2 = [math.log10(1+x) for x in  feature]
# #log_feature = [x ** .5 for x in  feature]
# # log_outcome = [x ** 2 for x in  outcome]

# plt.scatter(log_feature2, outcome)
# plt.title('Transformed 2 values')
# plt.show()
```






```
In [31]: norm = np.random.normal(0, 1, 200)
         norm.sort()
         data_arr = np.array(data['Sales'])
         data_arr.sort()
         plt.plot(norm, data_arr, "o")
         plt.show()
```



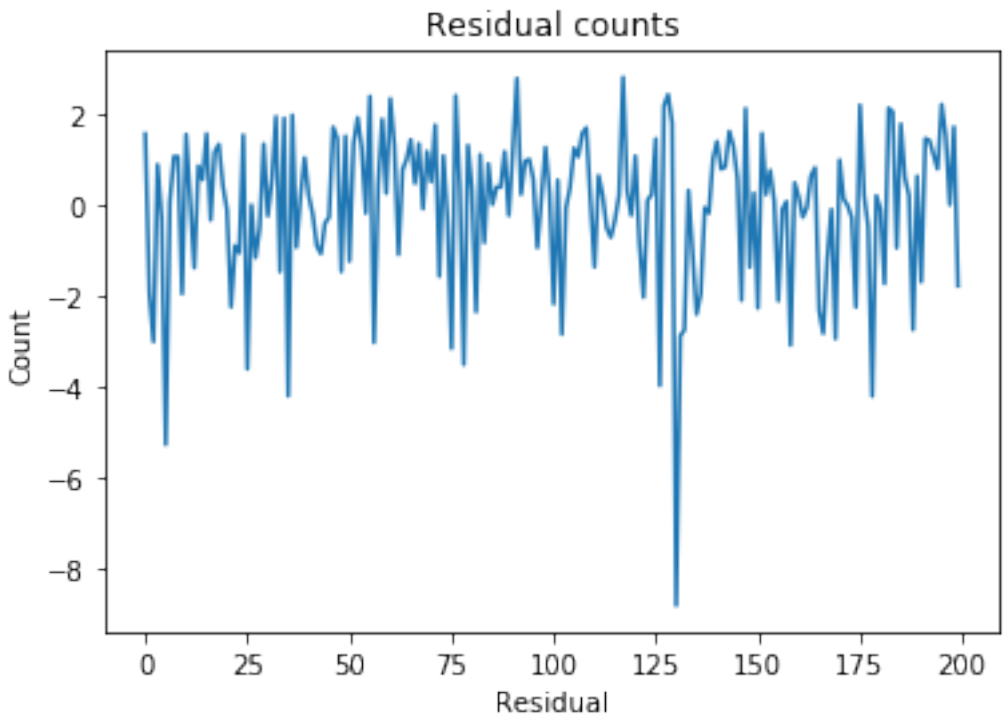
In [32]:
```
# data2 = pd.read_csv('https://tf-curricula-prod.s3.amazonaws.com/data-science/Advertis
data2 = data
display(data2.head())

# Instantiate and fit our model.
regr = linear_model.LinearRegression()
Y2 = data2['Sales'].values.reshape(-1, 1)
X2 = data2[['TV', 'Radio', 'Newspaper']]
regr.fit(X2, Y2)

# Inspect the results.
print('\nCoefficients: \n', regr.coef_)
print('\nIntercept: \n', regr.intercept_)
print('\nR-squared:')
print(regr.score(X2, Y2))
```

	Unnamed: 0	TV	Radio	Newspaper	Sales	LogTV	LogRadio	LTV_sq	\
0	1	230.100	37.800	69.200	22.100	2.364	1.589	5.588	
1	2	44.500	39.300	45.100	10.400	1.658	1.605	2.749	
2	3	17.200	45.900	69.300	9.300	1.260	1.671	1.588	
3	4	151.500	41.300	58.500	18.500	2.183	1.626	4.767	
4	5	180.800	10.800	58.400	12.900	2.260	1.072	5.106	

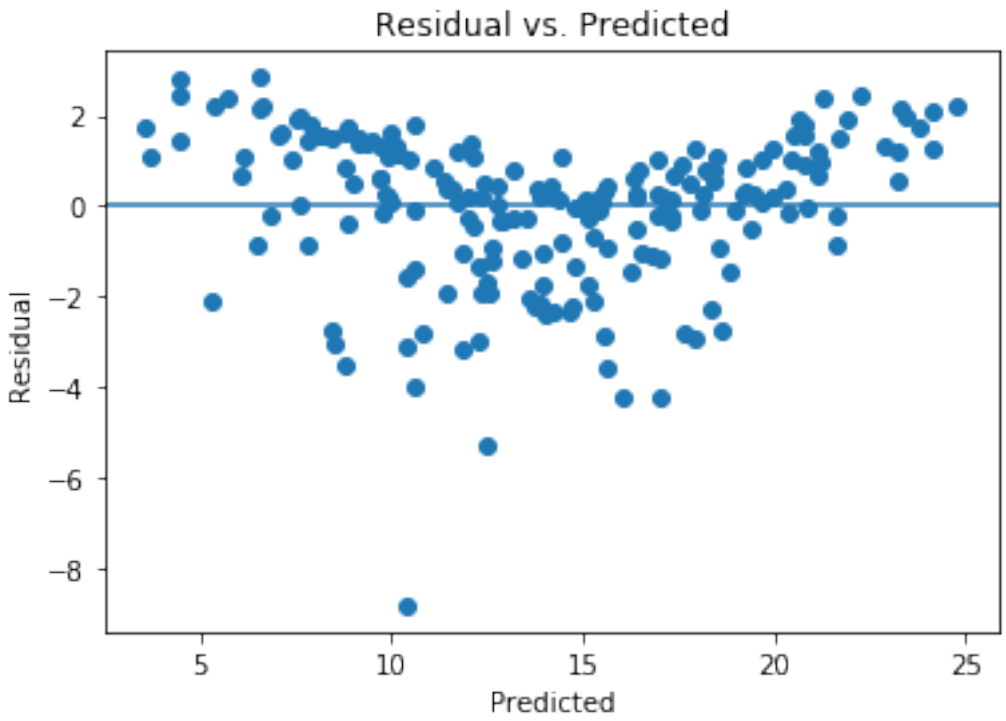
	LR_sq	TR_sum	TV_Radio	S2
0	2.524	267.900	8697.780	488.410
1	2.577	83.800	1748.850	108.160

```
2  2.793  63.100   789.480  86.490
3  2.645 192.800  6256.950 342.250
4  1.149 191.600  1952.640 166.410



Coefficients:
 [[ 0.04576465  0.18853002 -0.00103749]]

Intercept:
 [2.93888937]

R-squared:
0.8972106381789521
```

```
In [33]: # Extract predicted values.
         predicted2 = regr.predict(X2).ravel()
         actual2 = data2['Sales']

         # Calculate the error, also called the residual.
         residual2 = actual2 - predicted2

         # This looks a bit concerning.
         plt.plot(residual2)
         # plt.hist(residual2)
         plt.title('Residual counts')
         plt.xlabel('Residual')
         plt.ylabel('Count')
         plt.show()
         # with pd.option_context('display.max_rows', None, 'display.max_columns', 3):
         #     print(residual2)
```




```
In [34]: plt.scatter(predicted2, residual2)
         plt.xlabel('Predicted')
         plt.ylabel('Residual')
         plt.axhline(y=0)
         plt.title('Residual vs. Predicted')
         plt.show()

         # Hm... looks a bit concerning.
```



In [91]:
```
# data2 = pd.read_csv('https://tf-curricula-prod.s3.amazonaws.com/data-science/Advertis
# data3 = data.drop(130)
data3 = data
display(data3.head())
# data3['N2'] = data3['Newspaper'] ** 0.5
data3['LogTV'] = np.log10(data3['TV'] + 1)
data3['LogRadio'] = np.log10(data3['Radio'] + 1)
data3['LTV_sq'] = data3['LogTV'] ** 2
data3['LR_sq'] = data3['LogRadio'] ** 2
data3['TR_sum'] = data3['TV'] + data3['Radio']
data3['TV_Radio'] = data3['TV'] * data3['Radio']
data3['TV_Radio_sq'] = data3['TV_Radio'] ** 2
data3['LTV_LR'] = data3['LogTV'] * data3['LogRadio']
# data3['HighSpend'] = 0
# for row in data3:
#     if data3['TR_sum']>100:
#         data3['HighSpend'] = 1
# df['b']=df['a'].where(df['a']>0,other=0)
# data3['Inv'] = 1 / data3['TV']
data3['S2'] = data3['Sales'] ** 2
# data3['S2'] = np.log10(data3['Sales'])
display(data3.head())
# Instantiate and fit our model.
regr = linear_model.LinearRegression()
```

```
        Y3 = data3['Sales'].values.reshape(-1, 1)
        # X3 = data3[['LogTV', 'LogRadio', 'LTV_sq', 'LR_sq']]
        # X3 = data3[['LTV_sq', 'LR_sq']]
        #X3 = data3[['LogTV', 'LogRadio', 'TV_Radio']] # R_sq > 0.99, gorgeous residuals
        X3 = data3[['TV', 'Radio', 'TV_Radio']] # R_sq > 0.96, outliers in residuals, clear tre
        # X3 = data3[['LogTV', 'LogRadio', 'LTV_LR']]
        regr.fit(X3, Y3)

        # Inspect the results.
        print('\nCoefficients: \n', regr.coef_)
        print('\nIntercept: \n', regr.intercept_)
        print('\nR-squared:')
        print(regr.score(X3, Y3))
```

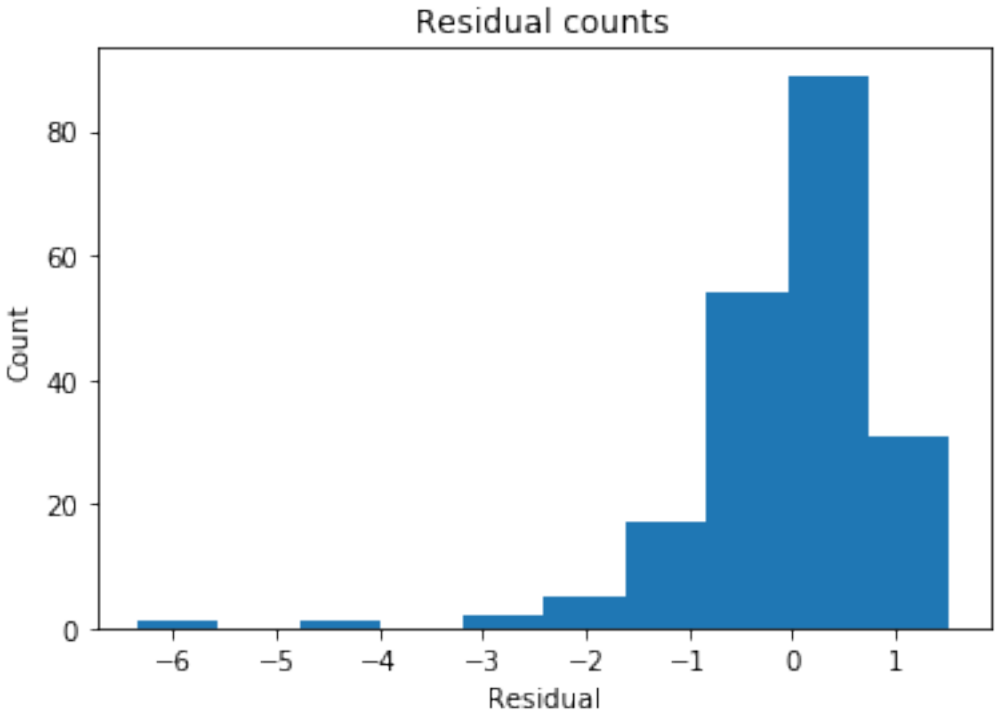
```
   Unnamed: 0      TV  Radio  Newspaper  Sales  LogTV  LogRadio  LTV_sq  \
0           1 230.100 37.800     69.200 22.100  2.364     1.589   5.588
1           2  44.500 39.300     45.100 10.400  1.658     1.605   2.749
2           3  17.200 45.900     69.300  9.300  1.260     1.671   1.588
3           4 151.500 41.300     58.500 18.500  2.183     1.626   4.767
4           5 180.800 10.800     58.400 12.900  2.260     1.072   5.106

   LR_sq  TR_sum  TV_Radio  LTV_LR      S2  TV_Radio_sq
0  2.524 267.900  8697.780   3.756 488.410 75651376.928
1  2.577  83.800  1748.850   2.662 108.160  3058476.322
2  2.793  63.100   789.480   2.106  86.490   623278.670
3  2.645 192.800  6256.950   3.551 342.250 39149423.302
4  1.149 191.600  1952.640   2.422 166.410  3812802.970
```


```
   Unnamed: 0      TV  Radio  Newspaper  Sales  LogTV  LogRadio  LTV_sq  \
0           1 230.100 37.800     69.200 22.100  2.364     1.589   5.588
1           2  44.500 39.300     45.100 10.400  1.658     1.605   2.749
2           3  17.200 45.900     69.300  9.300  1.260     1.671   1.588
3           4 151.500 41.300     58.500 18.500  2.183     1.626   4.767
4           5 180.800 10.800     58.400 12.900  2.260     1.072   5.106

   LR_sq  TR_sum  TV_Radio  LTV_LR      S2  TV_Radio_sq
0  2.524 267.900  8697.780   3.756 488.410 75651376.928
1  2.577  83.800  1748.850   2.662 108.160  3058476.322
2  2.793  63.100   789.480   2.106  86.490   623278.670
3  2.645 192.800  6256.950   3.551 342.250 39149423.302
4  1.149 191.600  1952.640   2.422 166.410  3812802.970
```


```
Coefficients:
 [[0.01910107 0.02886034 0.00108649]]
```

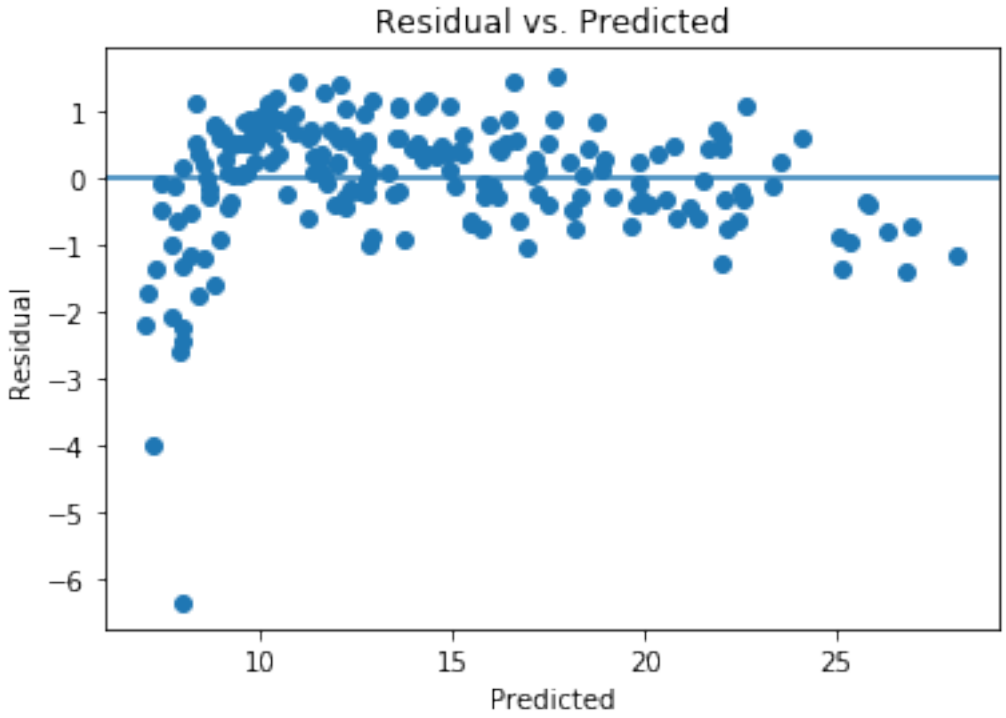
```
Intercept:
 [6.7502202]

R-squared:
0.9677905498482523
```

```
In [92]: # Extract predicted values.
         predicted3 = regr.predict(X3).ravel()
         actual3 = data3['Sales']

         # Calculate the error, also called the residual.
         residual3 = actual3 - predicted3

         # This looks a bit concerning.
         plt.hist(residual3)
         plt.title('Residual counts')
         plt.xlabel('Residual')
         plt.ylabel('Count')
         plt.show()
         # with pd.option_context('display.max_rows', None, 'display.max_columns', 3):
         #     print(residual2)
```

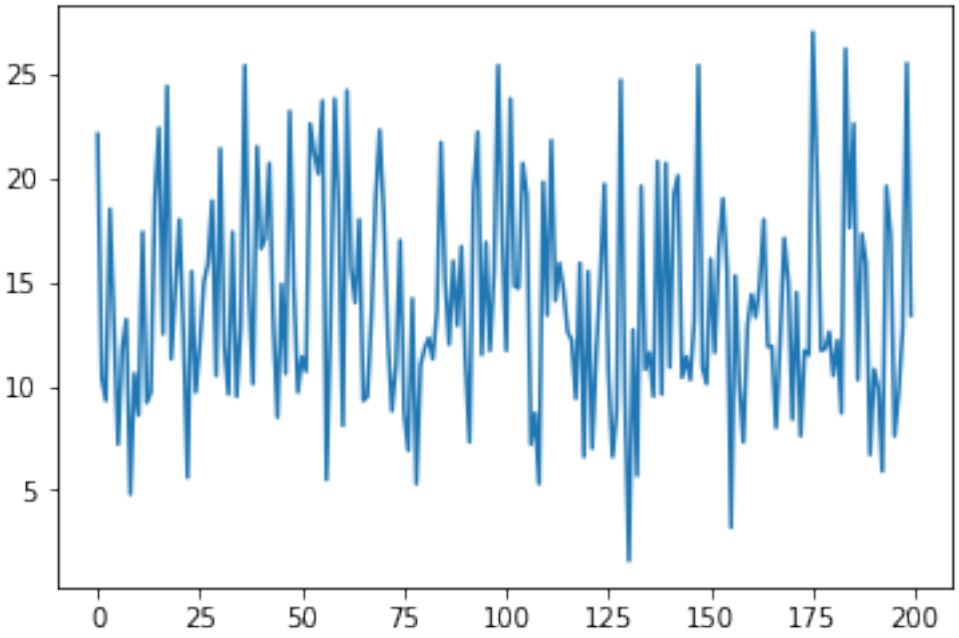



```
In [90]: plt.scatter(predicted3, residual3)
         plt.xlabel('Predicted')
```

```
plt.ylabel('Residual')
plt.axhline(y=0)
plt.title('Residual vs. Predicted')
plt.show()

# Hm... looks a bit concerning.
```




```
In [65]: plt.plot(data3['Sales'])

Out[65]: [<matplotlib.lines.Line2D at 0x118631898>]
```


25
20
15
10
5
0
25
50
75
100
125
150
175
200