1. Regression discontinuity: banking recovery

After a debt has been legally declared "uncollectable" by a bank, the account is considered "charged-off." But that doesn't mean the bank ***walks away*** from the debt. They still want to collect some of the money they are owed. The bank will score the account to assess the expected recovery amount, that is, the expected amount that the bank may be able to receive from the customer in the future. This amount is a function of the probability of the customer paying, the total debt, and other factors that impact the ability and willingness to pay.

The bank has implemented different recovery strategies at different thresholds ($1000, $2000, etc.) where the greater the expected recovery amount, the more effort the bank puts into contacting the customer. For low recovery amounts (Level 0), the bank just adds the customer's contact information to their automatic dialer and emailing system. For higher recovery strategies, the bank incurs more costs as they leverage human resources in more efforts to obtain payments. Each additional level of recovery strategy requires an additional $50 per customer so that customers in the Recovery Strategy Level 1 cost the company $50 more than those in Level 0. Customers in Level 2 cost $50 more than those in Level 1, etc.

The big question: does the extra amount that is recovered at the higher strategy level exceed the extra $50 in costs? In other words, was there a jump (also called a "discontinuity") of more than $50 in the amount recovered at the higher strategy level? We'll find out in this notebook.

![Regression discontinuity graph](https://assets.datacamp.com/production/project_504/img/Regression Discontinuity graph.png)

First, we'll load the banking dataset and look at the first few rows of data. This lets us understand the dataset itself and begin thinking about how to analyze the data.

In [87]:
```python
# Import modules
# ... YOUR CODE FOR TASK 1 ...
# ... YOUR CODE FOR TASK 1 ...
import pandas as pd
import numpy as np


# Read in dataset
df = pd.read_csv("datasets/bank_data.csv")

# Print the first few rows of the DataFrame
# ... YOUR CODE FOR TASK 1 ...
df.head()
```

Out[87]:

	id	expected_recovery_amount	actual_recovery_amount	recovery_strategy	age	sex
0	2030	194	263.540	Level 0 Recovery	19	Male
1	1150	486	416.090	Level 0 Recovery	25	Female
2	380	527	429.350	Level 0 Recovery	27	Male
3	1838	536	296.990	Level 0 Recovery	25	Male
4	1995	541	346.385	Level 0 Recovery	34	Male

In [88]:
```
%%nose

first_value = _

def test_pandas_loaded():
    assert pd.__name__ == 'pandas', \
        "pandas should be imported as pd."

def test_numpy_loaded():
    assert np.__name__ == 'numpy', \
        "numpy should be imported as np."

import pandas as pd

def test_df_correctly_loaded():
    correct_df = pd.read_csv('datasets/bank_data.csv')

    assert correct_df.equals(df), \
        "The variable df should contain the data in 'datasets/bank_data.csv'."

def test_head_output():
    try:
        assert "2030" in first_value.to_string()
    except AttributeError:
        assert False, \
            "Please use df.head() as the last line of code in the cell to insp
ect the data, not the display() or print() functions."
    except AssertionError:
        assert False, \
            "Hmm, the output of the cell is not what we expected. You should s
ee 2030 in the first five rows of the df DataFrame."
```

Out[88]:
```
4/4 tests passed
```

2. Graphical exploratory data analysis

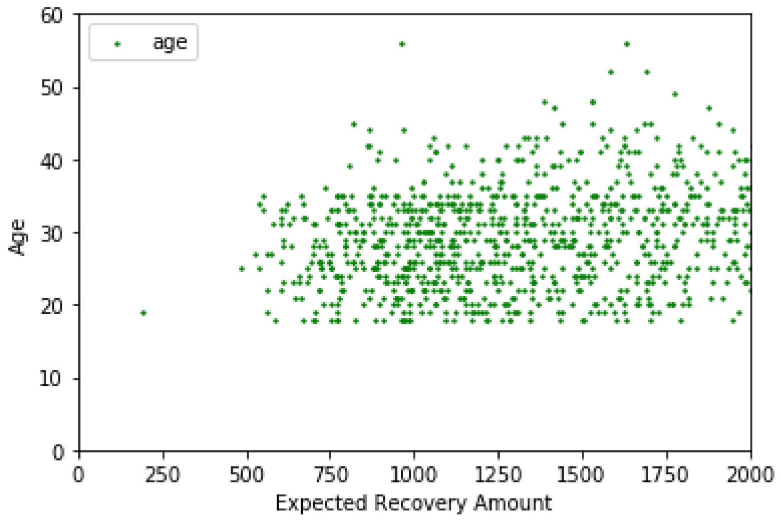
The bank has implemented different recovery strategies at different thresholds ($1000, $2000, $3000 and $5000) where the greater the Expected Recovery Amount, the more effort the bank puts into contacting the customer. Zeroing in on the first transition (between Level 0 and Level 1) means we are focused on the population with Expected Recovery Amounts between $0 and $2000 where the transition between Levels occurred at $1000. We know that the customers in Level 1 (expected recovery amounts between $1001 and $2000) received more attention from the bank and, by definition, they had higher Expected Recovery Amounts than the customers in Level 0 (between $1 and $1000).

Here's a quick summary of the Levels and thresholds again:

- Level 0: Expected recovery amounts >$0 and <=$1000
- Level 1: Expected recovery amounts >$1000 and <=$2000
- The threshold of $1000 separates Level 0 from Level 1

A key question is whether there are other factors besides Expected Recovery Amount that also varied systematically across the $1000 threshold. For example, does the customer age show a jump (discontinuity) at the $1000 threshold or does that age vary smoothly? We can examine this by first making a scatter plot of the age as a function of Expected Recovery Amount for a small window of Expected Recovery Amount, $0 to $2000. This range covers Levels 0 and 1.

In [89]:
```
# Scatter plot of Age vs. Expected Recovery Amount
from matplotlib import pyplot as plt
%matplotlib inline
plt.scatter(x=df['expected_recovery_amount'], y=df['age'], c="g", s=2)
plt.xlim(0, 2000)
plt.ylim(0, 60)
plt.xlabel("Expected Recovery Amount")
plt.ylabel("Age")
plt.legend(loc=2)
# ... YOUR CODE FOR TASK 2 ...
plt.show()
```



In [90]:
```
%%nose

# no tests for plots

# def test_nothing():
#     assert True, "Nothing to test."

def test_matplotlib_loaded_2():
    assert 'plt' in globals(), \
    'Did you import the pyplot module from matplotlib under the alias plt?'
```

Out[90]:
```
1/1 tests passed
```

3. Statistical test: age vs. expected recovery amount

We want to convince ourselves that variables such as age and sex are similar above and below the $1000 Expected Recovery Amount threshold. This is important because we want to be able to conclude that differences in the actual recovery amount are due to the higher Recovery Strategy and not due to some other difference like age or sex.

The scatter plot of age versus Expected Recovery Amount did not show an obvious jump around $1000. We will now do statistical analysis examining the average age of the customers just above and just below the threshold. We can start by exploring the range from $900 to $1100.

For determining if there is a difference in the ages just above and just below the threshold, we will use the Kruskal-Wallis test, a statistical test that makes no distributional assumptions.

In [91]:
```
# Import stats module
from scipy import stats

# Compute average age just below and above the threshold
era_900_1100 = df.loc[(df['expected_recovery_amount']<1100) &
                      (df['expected_recovery_amount']>=900)]
by_recovery_strategy = era_900_1100.groupby(['recovery_strategy'])
by_recovery_strategy['age'].describe().unstack()

# Perform Kruskal-Wallis test
Level_0_age = era_900_1100.loc[df['recovery_strategy']=="Level 0 Recovery"]['a
ge']
Level_1_age = era_900_1100.loc[df['recovery_strategy']=="Level 1 Recovery"]['a
ge']
stats.kruskal(Level_0_age,Level_1_age)
```

Out[91]:
```
KruskalResult(statistic=3.4572342749517513, pvalue=0.06297556896097407)
```

In [92]:
```
%%nose

def test_stats_loaded():
    assert 'stats' in globals(), \
    'Did you import the stats module from scipy?'

def test_level_0():
    correct_Level_0_age_mean= df.loc[(df['expected_recovery_amount']<1100) &
 (df['expected_recovery_amount']>=900) &
                     (df['recovery_strategy']=="Level 0 Recovery")]['age'].mea
n()
    Level_0_age_mean= Level_0_age.mean()
    assert correct_Level_0_age_mean == Level_0_age_mean, \
        "The mean age for Level_0_age appears to be incorrect. Did you correct
ly assign Level_0_age?"

def test_level_1():
    correct_Level_1_age_mean= df.loc[(df['expected_recovery_amount']<1100) &
 (df['expected_recovery_amount']>=900) &
                     (df['recovery_strategy']=="Level 1 Recovery")]['age'].mea
n()
    Level_1_age_mean= Level_1_age.mean()
    assert correct_Level_1_age_mean == Level_1_age_mean, \
        "The mean age for Level_1_age appears to be incorrect. Did you correct
ly assign Level_1_age?"
```

Out[92]:
```
3/3 tests passed
```

4. Statistical test: sex vs. expected recovery amount

We have seen that there is no major jump in the average customer age just above and just below the $1000 threshold by doing a statistical test as well as exploring it graphically with a scatter plot.

We want to also test that the percentage of customers that are male does not jump across the $1000 threshold. We can start by exploring the range of $900 to $1100 and later adjust this range.

We can examine this question statistically by developing cross-tabs as well as doing chi-square tests of the percentage of customers that are male vs. female.

In [93]:
```
# Number of customers in each category
crosstab = pd.crosstab(df.loc[(df['expected_recovery_amount']<1100) &
                              (df['expected_recovery_amount']>=900)]['recovery
_strategy'],
                       df['sex'])
# ... YOUR CODE FOR TASK 4 ...
crosstab
# Chi-square test
chi2_stat, p_val, dof, ex = stats.chi2_contingency(crosstab)
# ... YOUR CODE FOR TASK 4 ...
p_val
```

Out[93]:
```
0.5377947810444592
```

In [94]:
```
%%nose

def test_crosstab():
    correct_crosstab = pd.crosstab(df.loc[(df['expected_recovery_amount']<110
0) & (df['expected_recovery_amount']>=900)]['recovery_strategy'], df['sex'])
    assert correct_crosstab.equals(crosstab), \
    "The crosstab should select the expected_recovery_amount <1100 and >=900."

def test_pval():
    chi2_stat, correct_p_val, dof, ex = stats.chi2_contingency(crosstab)
    assert correct_p_val==p_val, \
    "The chi-square test function should use crosstab as the input variable."
```

Out[94]:
```
2/2 tests passed
```

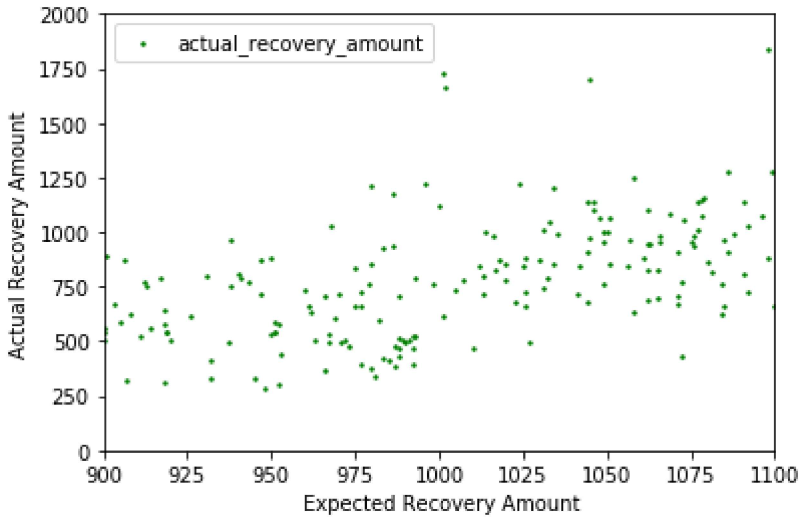
5. Exploratory graphical analysis: recovery amount

We are now reasonably confident that customers just above and just below the $1000 threshold are, on average, similar in their average age and the percentage that are male.

It is now time to focus on the key outcome of interest, the actual recovery amount.

A first step in examining the relationship between the actual recovery amount and the expected recovery amount is to develop a scatter plot where we want to focus our attention at the range just below and just above the threshold. Specifically, we will develop a scatter plot of Expected Recovery Amount (Y) vs. Actual Recovery Amount (X) for Expected Recovery Amounts between $900 to $1100. This range covers Levels 0 and 1. A key question is whether or not we see a discontinuity (jump) around the $1000 threshold.

In [95]:
```
# Scatter plot of Actual Recovery Amount vs. Expected Recovery Amount
plt.scatter(x=df['expected_recovery_amount'], y=df['actual_recovery_amount'],
c="g", s=2)
plt.xlim(900, 1100)
plt.ylim(0, 2000)
plt.xlabel(" Expected Recovery Amount")
plt.ylabel("Actual Recovery Amount")
plt.legend(loc=2)
# ... YOUR CODE FOR TASK 5 ...
plt.show()
```



In [96]:
```
%%nose

# no tests for plots

# def test_nothing():
#     assert True, "Nothing to test."

def test_matplotlib_loaded_5():
    assert 'plt' in globals(), \
    'Did you import the pyplot module from matplotlib under the alias plt?'
```

Out[96]:
```
1/1 tests passed
```

6. Statistical analysis: recovery amount

As we did with age, we can perform statistical tests to see if the actual recovery amount has a discontinuity above the $1000 threshold. We are going to do this for two different windows of the expected recovery amount $900 to $1100 and for a narrow range of $950 to $1050 to see if our results are consistent.

Again, we will use the Kruskal-Wallis test.

We will first compute the average actual recovery amount for those customers just below and just above the threshold using a range from $900 to $1100. Then we will perform a Kruskal-Wallis test to see if the actual recovery amounts are different just above and just below the threshold. Once we do that, we will repeat these steps for a smaller window of $950 to $1050.

In [97]:
```
# Compute average actual recovery amount just below and above the threshold
by_recovery_strategy['actual_recovery_amount'].describe().unstack()

# Perform Kruskal-Wallis test
Level_0_actual= era_900_1100.loc[df['recovery_strategy']=='Level 0 Recovery'][
'actual_recovery_amount']
Level_1_actual= era_900_1100.loc[df['recovery_strategy']=='Level 1 Recovery'][
'actual_recovery_amount']
stats.kruskal(Level_0_actual,Level_1_actual)

# Repeat for a smaller range of $950 to $1050
era_950_1050 = df.loc[(df['expected_recovery_amount']<1050) &
                      (df['expected_recovery_amount']>=950)]
Level_0_actual= era_950_1050.loc[df['recovery_strategy']=='Level 0 Recovery'][
'actual_recovery_amount']

Level_1_actual= era_950_1050.loc[df['recovery_strategy']=='Level 1 Recovery'][
'actual_recovery_amount']
# ... YOUR CODE FOR TASK 6 ...
stats.kruskal(Level_0_actual,Level_1_actual)
```

Out[97]:
```
KruskalResult(statistic=30.246000000000038, pvalue=3.80575314300276e-08)
```

In [98]:
```
%%nose

def test_level_0():
    correct_Level_0_actual_mean= df.loc[(df['expected_recovery_amount']<1050)
 & (df['expected_recovery_amount']>=950) &
                     (df['recovery_strategy']=="Level 0 Recovery")]['actual_re
covery_amount'].mean()
    Level_0_actual_mean= Level_0_actual.mean()
    assert correct_Level_0_actual_mean == Level_0_actual_mean, \
        "The mean actual_recovery_amount for Level_0_actual appears to be inco
rrect. Did you correctly assign Level_0_actual?"

def test_level_1():
    correct_Level_1_actual_mean= df.loc[(df['expected_recovery_amount']<1050)
 & (df['expected_recovery_amount']>=950) &
                     (df['recovery_strategy']=="Level 1 Recovery")]['actual_re
covery_amount'].mean()
    Level_1_actual_mean= Level_1_actual.mean()
    assert correct_Level_1_actual_mean == Level_1_actual_mean, \
        "The mean actual_recovery_amount for Level_1_actual appears to be inco
rrect. Did you correctly assign Level_1_actual?"
```

Out[98]:
```
2/2 tests passed
```

7. Regression modeling: no threshold

We now want to take a regression-based approach to estimate the program impact at the $1000 threshold using data that is just above and below the threshold.

We will build two models. The first model does not have a threshold while the second will include a threshold.

The first model predicts the actual recovery amount (dependent variable) as a function of the expected recovery amount (independent variable). We expect that there will be a strong positive relationship between these two variables.

We will examine the adjusted R-squared to see the percent of variance explained by the model. In this model, we are not representing the threshold but simply seeing how the variable used for assigning the customers (expected recovery amount) relates to the outcome variable (actual recovery amount).

In [99]:
```
# Import statsmodels
import statsmodels.api as sm

# Define X and y
X = era_900_1100['expected_recovery_amount']
y = era_900_1100['actual_recovery_amount']

X = sm.add_constant(X)

# Build linear regression model
model = sm.OLS(y, X).fit()
predictions = model.predict(X)

# Print out the model summary statistics
# ... YOUR CODE FOR TASK 7 ...
model.summary()
```

Out[99]:

OLS Regression Results

Dep. Variable:	actual_recovery_amount	**R-squared:**	0.261
Model:	OLS	**Adj. R-squared:**	0.256
Method:	Least Squares	**F-statistic:**	63.78
Date:	Mon, 07 Oct 2019	**Prob (F-statistic):**	1.56e-13
Time:	18:21:39	**Log-Likelihood:**	-1278.9
No. Observations:	183	**AIC:**	2562.
Df Residuals:	181	**BIC:**	2568.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-1978.7597	347.741	-5.690	0.000	-2664.907	-1292.612
expected_recovery_amount	2.7577	0.345	7.986	0.000	2.076	3.439

Omnibus:	64.493	**Durbin-Watson:**	1.777
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	185.818
Skew:	1.463	**Prob(JB):**	4.47e-41
Kurtosis:	6.977	**Cond. No.**	1.80e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.8e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [100]:
```
%%nose

def test_x():
    correct_x= df.loc[(df['expected_recovery_amount']<1100) & (df['expected_re
covery_amount']>=900)]['expected_recovery_amount']
    correct_x = sm.add_constant(correct_x)
    assert correct_x['expected_recovery_amount'].mean() == X['expected_recover
y_amount'].mean(), \
        "The mean expected_recovery_amount for X appears incorrect. Check your
assignment of X.  It should include expected_recovery_amount and indicator_100
0 when the expected_recovery_amount is <1100 and >=900."
def test_y():
    correct_y= df.loc[(df['expected_recovery_amount']<1100) & (df['expected_re
covery_amount']>=900)]['actual_recovery_amount']
    assert correct_y.mean() == y.mean(), \
        "The mean actual_recovery_amount for y appears incorrect. Check your a
ssignment of y. It should include the actual_recovery_amount when the expected
_recovery_amount is <1100 and >=900."

# def test_df_correctly_loaded():
#     correct_model = sm.OLS(y,x).fit()
#     assert correct_model.params[1] == model.params[1], \
#         "Check your assignment of model. It should be equal to sm.OLS(y,X).f
it()."
```

Out[100]:
```
2/2 tests passed
```

8. Regression modeling: adding true threshold

From the first model, we see that the expected recovery amount's regression coefficient is statistically significant.

The second model adds an indicator of the true threshold to the model (in this case at $1000).

We will create an indicator variable (either a 0 or a 1) that represents whether or not the expected recovery amount was greater than $1000. When we add the true threshold to the model, the regression coefficient for the true threshold represents the additional amount recovered due to the higher recovery strategy. That is to say, the regression coefficient for the true threshold measures the size of the discontinuity for customers just above and just below the threshold.

If the higher recovery strategy helped recovery more money, then the regression coefficient of the true threshold will be greater than zero. If the higher recovery strategy did not help recovery more money, then the regression coefficient will not be statistically significant.

In [101]:
```
# Create indicator (0 or 1) for expected recovery amount >= $1000
df['indicator_1000'] = np.where(df['expected_recovery_amount']<1000, 0, 1)
era_900_1100 = df.loc[(df['expected_recovery_amount']<1100) &
                      (df['expected_recovery_amount']>=900)]

# Define X and y
X = era_900_1100['expected_recovery_amount']
y = era_900_1100['actual_recovery_amount']
X = sm.add_constant(X)

# Build linear regression model
model = sm.OLS(y,X).fit()

# Print the model summary
model.summary()
```

Out[101]:

OLS Regression Results

Dep. Variable:	actual_recovery_amount	**R-squared:**	0.261
Model:	OLS	**Adj. R-squared:**	0.256
Method:	Least Squares	**F-statistic:**	63.78
Date:	Mon, 07 Oct 2019	**Prob (F-statistic):**	1.56e-13
Time:	18:21:39	**Log-Likelihood:**	-1278.9
No. Observations:	183	**AIC:**	2562.
Df Residuals:	181	**BIC:**	2568.
Df Model:	1		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-1978.7597	347.741	-5.690	0.000	-2664.907	-1292.612
expected_recovery_amount	2.7577	0.345	7.986	0.000	2.076	3.439

Omnibus:	64.493	**Durbin-Watson:**	1.777
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	185.818
Skew:	1.463	**Prob(JB):**	4.47e-41
Kurtosis:	6.977	**Cond. No.**	1.80e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.8e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [102]:
```
%%nose

def test_x():
    correct_x= df.loc[(df['expected_recovery_amount']<1100) & (df['expected_re
covery_amount']>=900),
           ['expected_recovery_amount','indicator_1000']]
    correct_x = sm.add_constant(correct_x)
    assert correct_x['expected_recovery_amount'].mean() == X['expected_recover
y_amount'].mean(), \
        "The mean expected_recovery_amount for X appears incorrect. Check your
assignment of X.  It should include expected_recovery_amount and indicator_100
0 when the expected_recovery_amount is <1100 and >=900."
def test_y():
    correct_y= df.loc[(df['expected_recovery_amount']<1100) & (df['expected_re
covery_amount']>=900)]['actual_recovery_amount']
    assert correct_y.mean() == y.mean(), \
        "The mean actual_recovery_amount for y appears incorrect. Check your a
ssignment of y. It should include the actual_recovery_amount when the expected
_recovery_amount is <1100 and >=900."

# def test_df_correctly_loaded():
#     correct_model = sm.OLS(y,X).fit()
#     assert correct_model.params[1] == model.params[1], \
#         "Check your assignment of model. It should be equal to sm.OLS(y,X).f
it()."
```

Out[102]:
```
2/2 tests passed
```

9. Regression modeling: adjusting the window

The regression coefficient for the true threshold was statistically significant with an estimated impact of around $278. This is much larger than the $50 per customer needed to run this higher recovery strategy.

Before showing this to our manager, we want to convince ourselves that this result wasn't due to choosing an expected recovery amount window of $900 to $1100. Let's repeat this analysis for the window from $950 to $1050 to see if we get similar results.

The answer? Whether we use a wide ($900 to $1100) or narrower window ($950 to $1050), the incremental recovery amount at the higher recovery strategy is much greater than the $50 per customer it costs for the higher recovery strategy. So we conclude that the higher recovery strategy is worth the extra cost of $50 per customer.

In [103]:
```
# Redefine era_950_1050 so the indicator variable is included
era_950_1050 = df.loc[(df['expected_recovery_amount']<1050) &
                      (df['expected_recovery_amount']>=950)]

# Define X and y
X = era_950_1050[['expected_recovery_amount','indicator_1000']]
y = era_950_1050['actual_recovery_amount']
X = sm.add_constant(X)

# Build linear regression model
model = sm.OLS(y,X).fit()

# Print the model summary
model.summary()
```

Out[103]:

OLS Regression Results

Dep. Variable:	actual_recovery_amount	**R-squared:**	0.283
Model:	OLS	**Adj. R-squared:**	0.269
Method:	Least Squares	**F-statistic:**	18.99
Date:	Mon, 07 Oct 2019	**Prob (F-statistic):**	1.12e-07
Time:	18:21:39	**Log-Likelihood:**	-692.92
No. Observations:	99	**AIC:**	1392.
Df Residuals:	96	**BIC:**	1400.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-279.5243	1840.707	-0.152	0.880	-3933.298	3374.250
expected_recovery_amount	0.9189	1.886	0.487	0.627	-2.825	4.663
indicator_1000	286.5337	111.352	2.573	0.012	65.502	507.566

Omnibus:	39.302	**Durbin-Watson:**	1.955
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	82.258
Skew:	1.564	**Prob(JB):**	1.37e-18
Kurtosis:	6.186	**Cond. No.**	6.81e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.81e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [104]:
```
%%nose

def test_x():
    correct_x= df.loc[(df['expected_recovery_amount']<1050) & (df['expected_re
covery_amount']>=950),['expected_recovery_amount','indicator_1000']]
    correct_x = sm.add_constant(correct_x)
    assert correct_x['expected_recovery_amount'].mean() == X['expected_recover
y_amount'].mean(), \
        "The mean expected_recovery_amount for X appears incorrect. Check your
assignment of X.  It should include expected_recovery_amount and indicator_100
0 when the expected_recovery_amount is <1050 and >=950."
def test_y():
    correct_y= df.loc[(df['expected_recovery_amount']<1050) & (df['expected_re
covery_amount']>=950)]['actual_recovery_amount']
    assert correct_y.mean() == y.mean(), \
        "The mean actual_recovery_amount for y appears incorrect. Check your a
ssignment of y. It should include the actual_recovery_amount when the expected
_recovery_amount is <1050 and >=950."

# def test_df_correctly_loaded():
#     correct_model = sm.OLS(y,X).fit()
#     assert correct_model.params[1] == model.params[1], \
#         "Check your assignment of model.  It should be equal to sm.OLS(y,X).
fit()."
```

Out[104]:
```
2/2 tests passed
```