

1. Regression discontinuity: banking recovery

After a debt has been legally declared "uncollectible" by a bank, the account is considered to be "charged-off." But that doesn't mean the bank simply **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 (for a fixed time period such as one year). 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 contact the customer and 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](https://assets.datacamp.com/production/project_504/img/Regression%20Discontinuity%20graph.png))

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

```
In [56]: # Import modules
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
df.head()
```

```
Out[56]:
```

	id	expected_recovery_amount	actual_recovery_amount	recovery_strategy	age	sex
0	2030	194	263.50	Level 0 Recovery	19	Male
1	1150	486	416.00	Level 0 Recovery	26	Female
2	140	527	429.350	Level 0 Recovery	27	Male
3	1338	9.00	296.900	Level 0 Recovery	25	Male
4	1095	541	345.35	Level 0 Recovery	34	Male

```
In [57]: #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_2630_was_selected():
    try:
        assert '2630' in first_value.to_string()
    except AttributeError:
        assert False, "It seems you have not displayed the first entries of df."
```

```
Out[57]:
```

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).

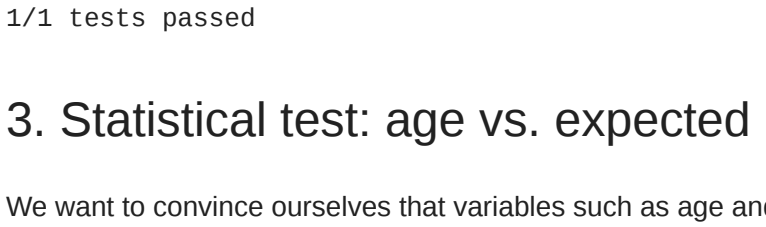
Here's a quick summary of the Levels and thresholds again:

- Level 0: Expected recovery amounts >40 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 [58]: # Scatter plot of Age vs. Expected Recovery Amount
import matplotlib.pyplot as plt

plt.plot(df[['expected_recovery_amount', 'age']], c='g', s=2)
plt.xlabel('Expected Recovery Amount')
plt.ylabel('Age')
plt.show()
```



```
In [59]: #nose

# no tests for plots

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

def test_matplotlib_loaded_2():
    assert 'plt' in globals(), \
        "You must import the pyplot module from matplotlib under the alias plt."
```

```
Out[59]:
```

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 expected 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 be more confident in our conclusions if we 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 which is a statistical test that makes no distributional assumptions.

```
In [60]: # Import statistical module
from scipy import stats

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

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

```
Out[60]:
```

	recovery_strategy	count	Level 0 Recovery	Level 1 Recovery
mean	Level 0 Recovery	89.000000	84.000000	27.224719
std	Level 1 Recovery	28.755319	18.000000	5.858907
min	Level 0 Recovery	18.000000	18.000000	23.000000
25%	Level 0 Recovery	23.000000	24.000000	26.000000
50%	Level 0 Recovery	26.000000	29.000000	31.000000
75%	Level 0 Recovery	31.000000	33.000000	56.000000
max	Level 0 Recovery	56.000000	56.000000	43.000000
dtype:	float64			

```
Out[60]:
```

KruskalResult(statistic=3.4572342749517513, pvalue=0.06297556896097407)

We see that the mean ages are fairly close to each other, and, statistically speaking, there isn't much difference between the two.

```
In [61]: #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'].mean()
    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 correctly 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'].mean()
    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 correctly assign Level_1_age?"
```

```
Out[61]:
```

3/3 tests passed

4. Statistical test: sex vs. expected recovery amount

We were able to convince ourselves 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 as well 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 [62]: # Number of customers in each category
crosstab = pd.crosstab(df[(df['expected_recovery_amount']<=1100) & (df['expected_recovery_amount']>=900)]['recovery_strategy'],
                      df['sex'])
print(crosstab)

# Chi-square test
chi2_stat, p_val, dof, ex = stats.chi2_contingency(crosstab)
print('p value:', p_val)
```

```
Out[62]:
```

	sex	Female	Male
recovery_strategy	Level 0 Recovery	32	57
	Level 1 Recovery	39	95

p value: 0.5377947810444592

Because the p value is very large, we can say that the differences between male and female across this boundary are similar

```
In [63]: #nose

def test_crosstab():
    correct_crosstab = pd.crosstab(df.loc[(df['expected_recovery_amount']<=1100) & (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[63]:
```

2/2 tests passed

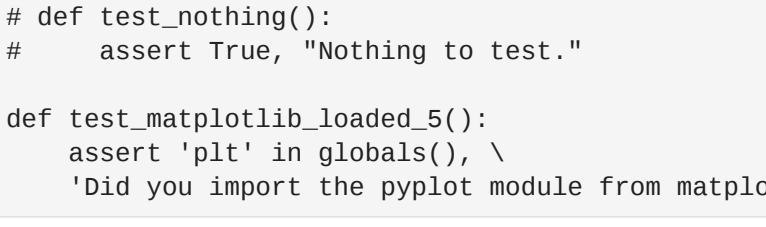
5. Exploratory graphical analysis: recovery amount

We are now reasonably confident that customers just above and just below the \$1000 threshold are, on average, similar in terms of 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 [64]: # 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)
plt.show()
```



```
In [65]: #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[65]:
```

1/1 tests passed

6. Statistical analysis: recovery amount

Just 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 narrower range of \$950 to \$1050 to see if our results are consistent.

Again, the statistical test we will use is the Kruskal-Wallis test, a test that makes no assumptions about the distribution of the actual recovery amount.

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 [66]: # Compute average actual recovery amount just below and above the threshold
print(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']
print(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']>=950) &
                     (df['expected_recovery_amount']<=1050)]
Level_0_actual = era_950_1050[era_950_1050['recovery_strategy']=='Level 0 Recovery']['actual_recovery_amount']
Level_1_actual = era_950_1050[era_950_1050['recovery_strategy']=='Level 1 Recovery']['actual_recovery_amount']
stats.kruskal(Level_0_actual, Level_1_actual)
```

```
Out[66]:
```

	recovery_strategy	count	Level 0 Recovery	Level 1 Recovery
mean	Level 0 Recovery	89.000000	84.000000	623.017022
std	Level 1 Recovery	955.825581	211.620854	293.732434
min	Level 0 Recovery	282.850800	433.399166	491.425900
25%	Level 0 Recovery	491.425900	777.765114	575.435600
50%	Level 0 Recovery	575.435600	987.271525	762.950800
75%	Level 0 Recovery	762.950800	1060.34387	1225.630080
max	Level 1 Recovery	2953.290125		
dtype:	float64			

```
Out[66]:
```

KruskalResult(statistic=65.37966302508078, pvalue=6.177390752003169e-16)

```
Out[67]:
```

KruskalResult(statistic=36.246609000000038, pvalue=8.80575314300276e-08)

From the above data, we see that the means are very different above and below the threshold, and the very small p-values indicate a change between each population on either side of the threshold.

```
In [67]: #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_recovery_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 incorrect. 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_recovery_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 incorrect. Did you correctly assign Level_1_actual?"
```

```
Out[67]:
```

2/2 tests passed

7. Regression modeling: no threshold

We now want to take a regression-based approach to estimate the impact of the program at the \$1000 threshold using the data that is just above and just below the threshold. In order to do that, we will build two models. The first model does not have a threshold while the second model will include a threshold.

The first model predicts the actual recovery amount (outcome or dependent variable) as a function of the expected recovery amount (input or 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 that is explained by the model. In this model, we are not trying to represent the threshold but simply trying to see how the variable used for assigning the customers (expected recovery amount) relates to the outcome variable (actual recovery amount).

```
In [68]: # 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
print(model.summary())
```

```
Out[68]:
```

OLS Regression Results

	Dep. Variable:	actual_recovery_amount	R-squared:	0.261
Model:				0.266
Method:		Least Squares	F-statistic:	63.78
Date:		Sat, 28 Apr 2019	Prob (F-statistic):	1.56e-13
Time:		06:48:48	Log-Likelihood:	-1278.9
No. Observations:		183	AIC:	2568
Of Residuals:		181	BIC:	2568
DF Model:				2
Covariance Type:		nonrobust		

coef std err t P>|t| [0.025 0.075]

	coef	std err	t	P> t	[0.025	0.075]
const	-1078.7597	347.741	-5.690	0.000	-2604.907	-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 [69]: #nose

def test_x():
    correct_x = df.loc[(df['expected_recovery_amount']<=1100) & (df['expected_recovery_amount']>=900)]['expected_recovery_amount']
    correct_y = sm.add_constant(correct_x)
    assert correct_x.equals(correct_y).mean() == X['expected_recovery_amount'].mean(), \
        "The mean expected_recovery_amount for X appears Incorrect. Check your assignment of X. It should include expected_recovery_amount and indicator_1000 when the expected_recovery_amount is <=1100."

def test_y():
    correct_y = df.loc[(df['expected_recovery_amount']<=1100) & (df['expected_recovery_amount']>=900)]['actual_recovery_amount']
    assert correct_y.mean() == y.mean(), \
        "The mean actual_recovery_amount for y appears incorrect. Check your assignment of y. It should include the actual_recovery_amount when the expected_recovery_amount is <=1100."

# 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[69]:
```

2/2 tests passed

8. Regression modeling: adding true threshold

From the first model, we see that the regression coefficient is statistically significant for the expected recovery amount and the adjusted R-squared value was about 0.26. As we saw from the graph, on average the actual recovery amount increases as the expected recovery amount increases. We could add polynomial terms of expected recovery amount (such as the squared value of expected recovery amount) to the model but, for the purposes of this practice, let's stick with using just the linear model.

The second model adds an indicator of the true threshold to the model. If there was no impact of the higher recovery strategy on the actual recovery amount, then we would expect that the relationship between the expected recovery amount and the actual recovery amount would be continuous.

In this case, we know the true threshold is 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 did help recover more money, then the regression coefficient of the true threshold will be greater than zero. If the higher recovery strategy did not help recover more money than the regression coefficient will not be statistically significant.

```
In [70]: # Create indicator (0 or 1) for expected recovery amount >= $1000
df['indicator_1000'] = np.where(df['expected_recovery_amount']>=1000, 1, 0)
era_900_1100 = df[(df['expected_recovery_amount']<=1100) &
                 (df['expected_recovery_amount']>=900)]

# Define X and y
X = era_900_1100[['expected_recovery_amount', 'indicator_1000']]
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
print(model.summary())
```

```
Out[70]:
```

OLS Regression Results

	Dep. Variable:	actual_recovery_amount	R-squared:	0.314
Model:				0.307
Method:		Least Squares	F-statistic:	41.22
Date:		Sat, 28 Apr 2019	Prob (F-statistic):	1.83e-15
Time:		06:48:48	Log-Likelihood:	-1272.0
No. Observations:		183	AIC:	2568
Of Residuals:		181	BIC:	2568
DF Model:				2
Covariance Type:		nonrobust		

coef std err t P>|t| [0.025 0.075]

	coef	std err	t	P> t	[0.025	0.075]
const	-3.448	626.274	-0.005	0.999	-1322.440	1239.128
expected_recovery_amount	0.9359	0.048	19.991	0.000	0.839	0.997
indicator_1000	277.6344	74.843	3.750	0.000	131.530	423.739

Omnibus: 65.977 Durbin-Watson: 1.995

Prob(Omnibus): 0.000 Jarque-Bera (JB): 186.537

Skew: 1.510 Prob(JB): 3.12e-41

Kurtosis: 6.917 Cond. No. 1.37e+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 [71]: #nose

def test_x():
    correct_x = df.loc[(df['expected_recovery_amount']<=1050) & (df['expected_recovery_amount']>=950), ['expected_recovery_amount', 'indicator_1000']]
    correct_x = sm.add_constant(correct_x)
    assert correct_x.equals(correct_y).mean() == X['expected_recovery_amount'].mean(), \
        "The mean expected_recovery_amount for X appears Incorrect. Check your assignment of X. It should include expected_recovery_amount and indicator_1000 when the expected_recovery_amount is <=1100."

def test_y():
    correct_y = df.loc[(df['expected_recovery_amount']<=1050) & (df['expected_recovery_amount']>=950)]['actual_recovery_amount']
    assert correct_y.mean() == y.mean(), \
        "The mean actual_recovery_amount for y appears incorrect. Check your assignment of y. It should include the actual_recovery_amount when the expected_recovery_amount is <=1100."

# 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[71]:
```

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 and a 95 percent confidence interval of \$132 to \$424. This is much larger than the incremental cost of running the higher recovery strategy which was \$50 per customer. At this point, we are feeling reasonably confident that the higher recovery strategy is worth the additional costs of the program for customers just above and just below the threshold.

Before showing this to our managers, we want to convince ourselves that this result wasn't due just to us choosing a window of \$900 to \$1100 for the expected recovery amount. If the higher recovery strategy really had an impact of an extra few hundred dollars, then we should see a similar regression coefficient if we choose a slightly bigger or a slightly smaller window for the expected recovery amount. Let's repeat this analysis for the window of expected recovery amount from \$950 to \$1050 to see if we get similar results.

The answer? Whether we use a wide window (\$900 to \$1100) or a 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 can say that the higher recovery strategy is worth the extra \$50 per customer that the bank is spending.

```
In [72]: # 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
print(model.summary())
```

```
Out[72]:
```

OLS Regression Results

	Dep. Variable:	actual_recovery_amount	R-squared:	0.566
Model:				0.566
Method:		Least Squares	F-statistic:	595.2
Date:		Sat, 28 Apr 2019	Prob (F-statistic):	3.18e-166
Time:		06:48:59	Log-Likelihood:	-6709.0
No. Observations:		917	AIC:	1.342e+04
Of Residuals:		915	BIC:	1.344e+04
DF Model:				2
Covariance Type:		nonrobust		

coef std err t P>|t| [0.025 0.075]

	coef	std err	t	P> t	[0.025	0.075]
const	-229.4596	45.604	-4.966	0.000	-315.969	-136.970
expected_recovery_amount	0.9035	0.048	18.991	0.000	0.814	0.997
indicator_1000	286.5337	111.352	2.573	0.012	65.502	507.566

Omnibus: 381.132 Durbin-Watson: 1.976

Prob(Omnibus): 0.000 Jarque-Bera (JB): 3202.819

Skew: 1.675 Prob(JB): 8.2e-58

Kurtosis: 11.523 Cond. No. 0.00

Warnings:

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

[2] The condition number is large, 5.81e+03. This might indicate that there are strong multicollinearity or other numerical problems.

```
In [73]: #let's do it for the entire level ranges
era_0_2000 = df.loc[(df['expected_recovery_amount']<=2000)]

# Define X and y
X = era_0_2000[['expected_recovery_amount', 'indicator_1000']]
y = era_0_2000['actual_recovery_amount']
X = sm.add_constant(X)

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

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

```
Out[73]:
```

OLS Regression Results

	Dep. Variable:	actual_recovery_amount	R-squared:	0.666
Model:				0.666
Method:		Least Squares	F-statistic:	950.2
Date:		Sat, 28 Apr 2019	Prob (F-statistic):	3.18e-166
Time:		06:48:59	Log-Likelihood:	-6709.0
No. Observations:		917	AIC:	1.342e+04
Of Residuals:		915	BIC:	1.344e+04
DF Model:				2
Covariance Type:		nonrobust		

coef std err t P>|t| [0.025 0.075]

	coef	std err	t	P> t	[0.025	0.075]
const	-226					