Analyzing the Impacts of the Status of Automobiles on Repair Records — A Python Programming Exercise for Ordinal Response Models

April 8, 2024

Code contributor: Ziyue Wang (Third Year Undergraduate Student of IESR of Jinan University)

1 Introduction

1.1 Main Focus and Dataset

This exercise aims to examine how different characteristics of automobiles impact their repair records. The "`fullauto`" dataset provided in the supplemental material contains repair records of automobiles along with various characteristics of these automobiles, and will be used for this cross-sectional analysis. "`fullauto`" contains the following main variables:

- The dependent variable `rep77` measures the repair record of the automobile in 1977. The variable take 5 possible ordinal values {1, 2, 3, 4, 5} which respectively represent the repair record of the car is: poor, fair, average, good, and excellent.
- There are 11 key independent variables and I choose to focus on: `foreign` (domestic or foreign car), `length` (measured in inches), and `mpg` (miles per gallon, a measure of fuel efficiency).

The key independent variables that I am focusing on may have potential implications in three areas:

1. **Consumer Decision-Making**: Consumers might consider a car's origin, size, and fuel efficiency when making a purchase decision. However, from consumers aspects, they may tend to overlook the repair record of a car, which is a critical indicator of its reliability and potential future costs.

2. **Policy Implications**: For example, if larger or less fuel-efficient cars have worse repair records, policies could be implemented to promote the production and purchase of smaller, more fuel-efficient cars.

3. **Manufacturer Strategies**: For instance, manufacturers could focus on developing more fuel-efficient technologies if fuel efficiency is found to be a significant factor for better repair records.

1.2 Econometric Methods

Regarding the econometric method, I employ the ordered logit (a.k.a. ordinal logistic) regression. Using the ordinal response model, one can estimate the probability of the repair record being in a certain category or lower, based on the explanatory variables. The model can be formulated as

$$\Pr(rep77_i = j|\mathbf{x}_i, \beta, \alpha) = \Lambda(\alpha_j + \beta'\mathbf{x}_i) - \Lambda(\alpha_{j-1} + \beta'\mathbf{x}_i)$$

for any $i \in \{1, \dots, n\}$ and $j \in \{0, 1, 2, 3, 4\}$, where

- $\beta'\mathbf{x}_i = \beta_1 foreign_i + \beta_2 length_i + \beta_3 mpg_i$,
- $\Lambda(.)$ is the CDF function of the standard logistic distribution (a.k.a. the sigmoid function in the machine learning context),
- and $\alpha = (\alpha_{-1}, \dots, \alpha_4)'$ refers to the cutoff points.

Please kindly note that, for the purpose of Python programming convinience, I have recoded the numerals of the dependent variable from {1, 2, 3, 4, 5} to {0, 1, 2, 3, 4}. Additionally, for the econometric identification, the smallest and the largest cutoff points are anchored at $-\infty$ and ∞. These will be reflected when I manually codifying the MLE estimation in Python.

Here, I chose the logit link (instead of the probit one) for my analysis as it was more technically challenging to implement in Python programming. When a variable has a large support, using it in the exponential function of the logistic CDF can lead to *numerical issues*. The exponential function grows exponentially, and with somehow large values, it can exceed the capacity of Python installed on a conventional personal computer. As a result, the estimation can fail in Python – numerically.

Apart from solely estimating the model parameters and conducting regular inferences, I will also calculate the marginal effects by calling Stata from Python to obtain meaningful economic interpretations.

1.3 Challenges in Python Programming

While programming in Python, I encountered three main challenges:

1. The repair records variable is documented as an ordinal variable. To analyze it in Python, one needs to utilize Python's `statsmodels` package. However, the `statsmodels` package (until the current version) does not contain a robust routine for estimating the cutoff points (in which the reparameterization is done). Therefore, manual coding based on econometric formulae is possibly required.

2. Based on the above mentioned reason, it is very helpful to use the *Python-Stata* integration to call Stata and verify my estimates. Even though I have already computed the marginal effects, the standard errors for these marginal effects require delta methods to be approximated, which seems technically difficult. Therefore, calling Stata in Python to fulfill this task becomes a natural solution.

3. The manual programming of maximum likelihood estimation for the ordinal response is challenging as the log-likelihood surface may not be alway concave (especially when applying the real data). In real analysis, one needs to seriously account for the numerical issue.

In what follows,

- I start with importing data and then clean it up to prepare it for Python analysis.
- In the descriptive analysis, I generate summary statistics and graphs to get a general sense of the data.
- When it comes to the econometric analysis in Python, I use a combination of methods including

– the existing `statsmodels` routines to compute model parameter estimates,
– manually coding up the negative log-likelihood and conducting `scipy`'s optimization routine to perform maximum likelihood estimation (MLE),
– checking the global concavity of the log-likelihood curve,
– applying the bootstrap method to compute standard errors – in a different way,
– generating various plots to visualize estimation results,
– and calling Stata from Python to implement the task.

2 Data Cleaning

Please note that the raw data file is saved in Stata's `.dta` format. I import the data and check for any possible missing values. It is important to identify and address missing values at the outset, as they can cause manual Python programming to fail. This is why I prioritize checking for missing values at the beginning of this exercise.

[1]:
```
import pandas as pd
# Read Stata format datafile in Python
data=pd.read_stata("fullauto.dta")
# Drop the missing values
data = data.dropna(subset=['rep77'])
category_rep = {'Poor': 1, 'Fair': 2, 'Average': 3, 'Good': 4, 'Excellent': 5}
category_origin = {'Foreign':1, 'Domestic':0}
data['rep77'] = data['rep77'].replace(category_rep)
data['rep78'] = data['rep78'].replace(category_rep)
data['foreign'] = data['foreign'].replace(category_origin)
data.head()
```

[1]:
```
   make    model  price  mpg rep78 rep77  hdroom  rseat  trunk  weight  \
0   AMC  Concord   4099   22     3     2     2.5   27.5     11    2930
1   AMC    Pacer   4749   17     3     1     3.0   25.5     11    3350
3  Audi      Fox   6295   23     3     3     2.5   28.0     11    2070
4  Audi     5000   9690   17     5     2     3.0   27.0     15    2830
5   BMW      320   9735   25     4     4     2.5   26.0     12    2650

   length  turn  displ  gratio  order foreign    wgtd    wgtf
0     186    40    121    3.58      1       0  2930.0     NaN
1     173    40    258    2.53      2       0  3350.0     NaN
3     174    36     97    3.70      5       1     NaN  2070.0
4     189    37    131    3.20      4       1     NaN  2830.0
5     177    34    121    3.64      6       1     NaN  2650.0
```

3 Descriptive Analysis

As the data is imported in this running Python instance as a Pandas' DataFrame object, to have a general understanding of the data structure, I apply the next Python Pandas function.

[2]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 66 entries, 0 to 73
Data columns (total 18 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   make     66 non-null     category
 1   model    66 non-null     category
 2   price    66 non-null     int16
 3   mpg      66 non-null     int16
 4   rep78    66 non-null     category
 5   rep77    66 non-null     category
 6   hdroom   66 non-null     float32
 7   rseat    66 non-null     float32
 8   trunk    66 non-null     int16
 9   weight   66 non-null     int16
 10  length   66 non-null     int16
 11  turn     66 non-null     int16
 12  displ    66 non-null     int16
 13  gratio   66 non-null     float32
 14  order    66 non-null     int16
 15  foreign  66 non-null     category
 16  wgtd     45 non-null     float32
 17  wgtf     21 non-null     float32
dtypes: category(5), float32(5), int16(8)
memory usage: 7.0 KB
```

It is important to check the maximum and minimum values of each variable used in the analysis. This is because if the variable has a *somehow* large range, using it in the exponential function of the logistic CDF can cause **numerical issue**. This is because the exponential function increases in a geometric order, and with very large values, it can easily exceed the limits of Python. This can cause the estimation to fail *numerically*.

The table of summary statistics is produced by the next command.

[3]:
```
data.describe()
```

[3]:

	price	mpg	hdroom	rseat	trunk	weight
count	66.000000	66.000000	66.000000	66.000000	66.000000	66.000000
mean	6222.575758	21.333333	3.007576	27.083334	13.939394	3058.181818
std	2955.821115	6.207522	0.843493	3.026274	4.381355	788.144675
min	3291.000000	12.000000	1.500000	21.000000	5.000000	1760.000000
25%	4189.000000	17.250000	2.500000	25.125000	11.000000	2302.500000
50%	5138.000000	20.000000	3.000000	27.000000	15.000000	3205.000000
75%	6332.250000	24.000000	3.500000	29.000000	17.000000	3685.000000
max	15906.000000	41.000000	5.000000	37.500000	23.000000	4840.000000

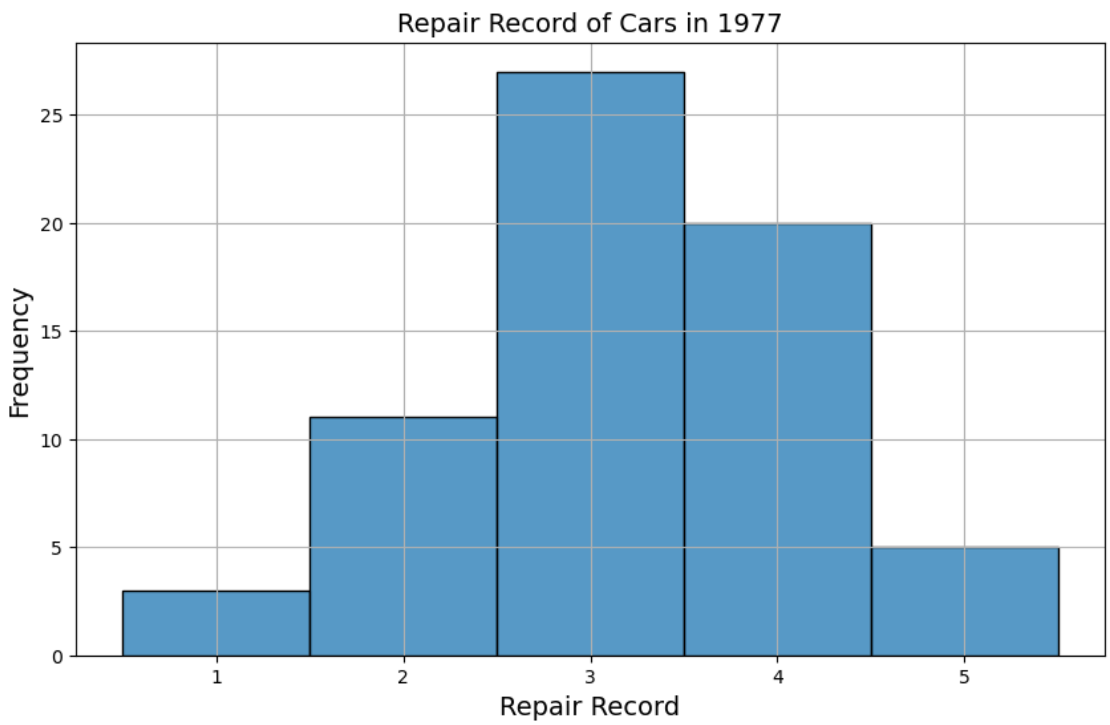
	length	turn	displ	gratio	order	wgtd
count	66.000000	66.000000	66.000000	66.000000	66.000000	45.000000

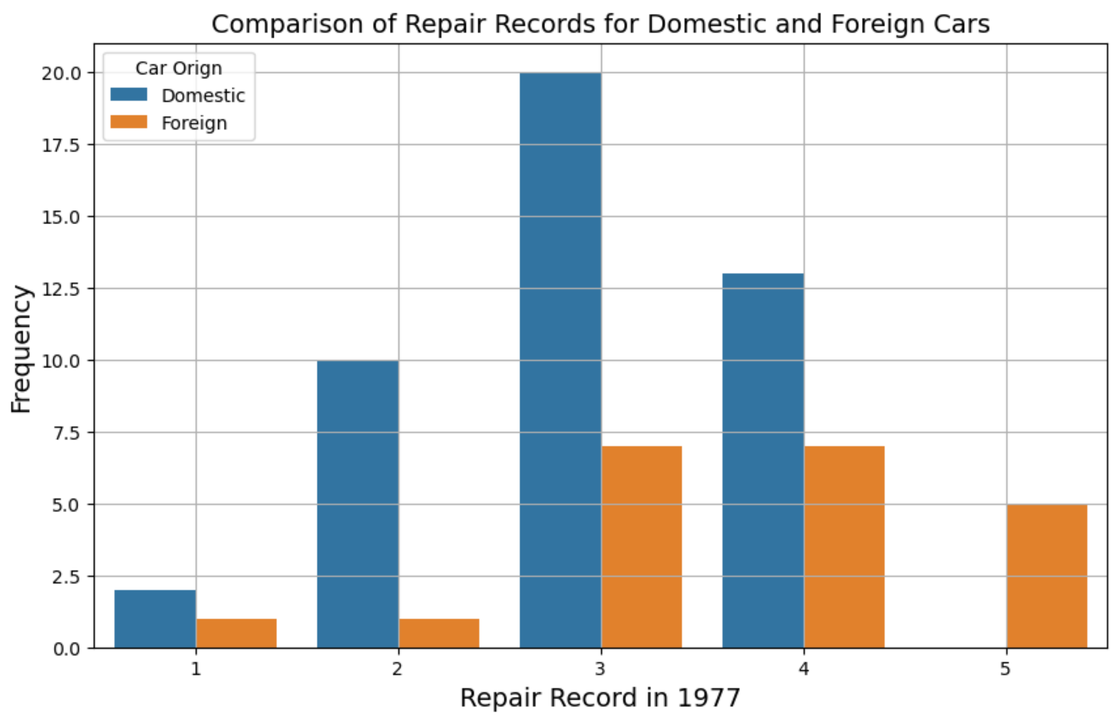
```
mean    189.121212  39.939394  200.136364   3.000000  37.227273  3429.111084
std      22.463314   4.433713   93.508516   0.465301  21.528368   644.151428
min     142.000000  31.000000   79.000000   2.190000   1.000000  1800.000000
25%     170.500000  36.000000  119.500000   2.730000  19.250000  3200.000000
50%     194.000000  40.500000  198.000000   2.930000  36.500000  3400.000000
75%     205.500000  43.000000  245.250000   3.282500  54.750000  3830.000000
max     233.000000  51.000000  425.000000   3.890000  74.000000  4840.000000

              wgtf
count    21.000000
mean   2263.333252
std     364.709930
min    1760.000000
25%    2020.000000
50%    2160.000000
75%    2410.000000
max    3170.000000
```

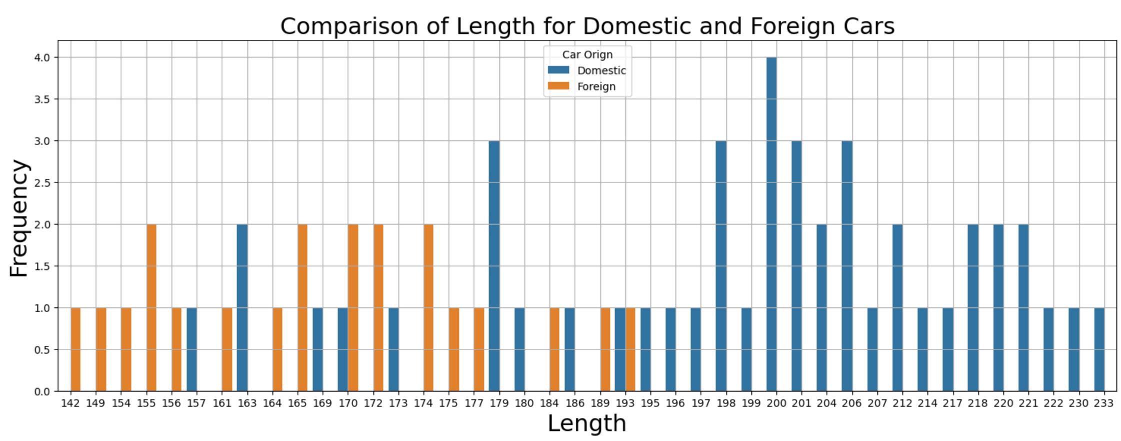
Next scripts generates figures for getting a sense of the distribution of the data.

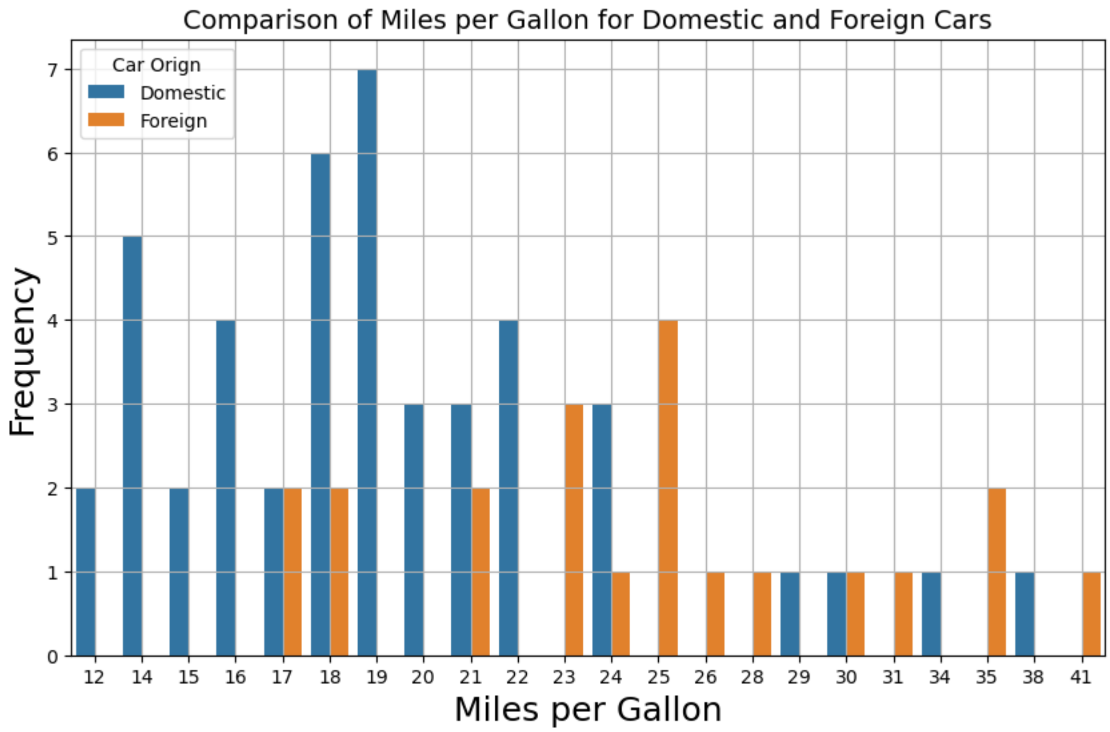
[4]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
# Distribution of Repair Record
plt.figure(figsize=(10, 6))
sns.histplot(data=data, x='rep77')
plt.xlabel('Repair Record', fontsize = 14)
plt.ylabel('Frequency', fontsize = 14)
plt.title('Repair Record of Cars in 1977', fontsize = 14)
plt.grid()
plt.show()

# Comparison between Domestic and Foreign Cars
plt.figure(figsize=(10, 6))
sns.countplot(x='rep77', hue='foreign', data=data)
plt.title('Comparison of Repair Records for Domestic and Foreign Cars',␣
 ↪fontsize = 14)
plt.xlabel('Repair Record in 1977', fontsize = 14)
plt.ylabel('Frequency', fontsize = 14)
plt.legend(title='Car Orign', labels=['Domestic', 'Foreign'])
plt.grid()
plt.show()

# Comparison between Car Size
plt.figure(figsize=(18, 6))
sns.countplot(x='length', hue='foreign', data=data)
plt.title('Comparison of Length for Domestic and Foreign Cars', fontsize = 22)
plt.xlabel('Length', fontsize = 22)
```

```
plt.ylabel('Frequency', fontsize = 22)
plt.legend(title='Car Orign', labels=['Domestic', 'Foreign'])
plt.grid()
plt.show()

# Comparison betweem Fuel Efficiency
plt.figure(figsize=(10, 6))
sns.countplot(x='mpg', hue='foreign', data=data)
plt.title('Comparison of Miles per Gallon for Domestic and Foreign Cars',␣
 ↪fontsize = 14)
plt.xlabel('Miles per Gallon', fontsize = 18)
plt.ylabel('Frequency', fontsize = 18)
plt.legend(title='Car Orign', labels=['Domestic', 'Foreign'])
plt.grid()
plt.show()
```




Comparison of Repair Records for Domestic and Foreign Cars
Car Orign
Domestic
Foreign
Frequency
Repair Record in 1977


Comparison of Length for Domestic and Foreign Cars
Car Orign
Domestic
Foreign
Frequency
Length



Above four plots give basic information about the reliability, size, and fuel efficiency of domestic and foreign cars in 1977. The first plot shows the distribution of the repair records for cars in that year, and the second plot compares the repair records of domestic and foreign cars, helping us understand if there are any significant differences in their reliability unconditionally.

The third plot compares the lengths of domestic and foreign cars. Moreover, the last plot compares the miles per gallon (mpg) of domestic and foreign cars, helping us understand if there is a significant difference in their fuel efficiency.

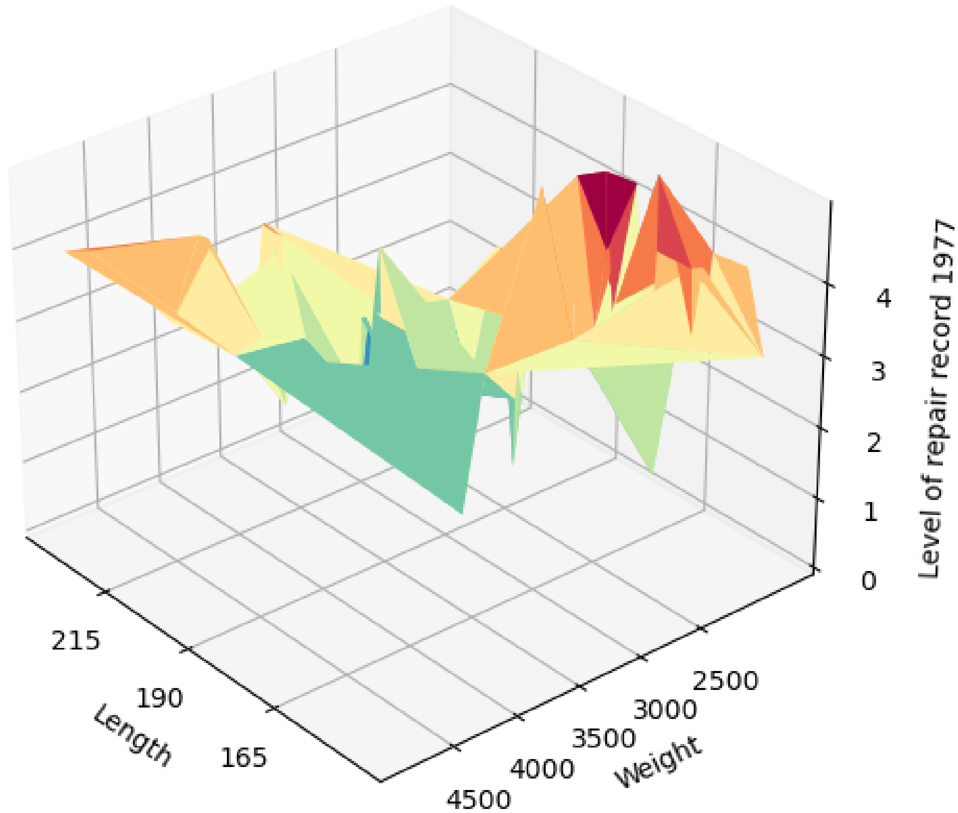
From the descriptive analysis, it is concluded that foreign cars tend to have better repair records than domestic cars. This observation may be attributed, in part, to their smaller size, as measured by car length, and superior fuel efficiency, as measured by miles per gallon.

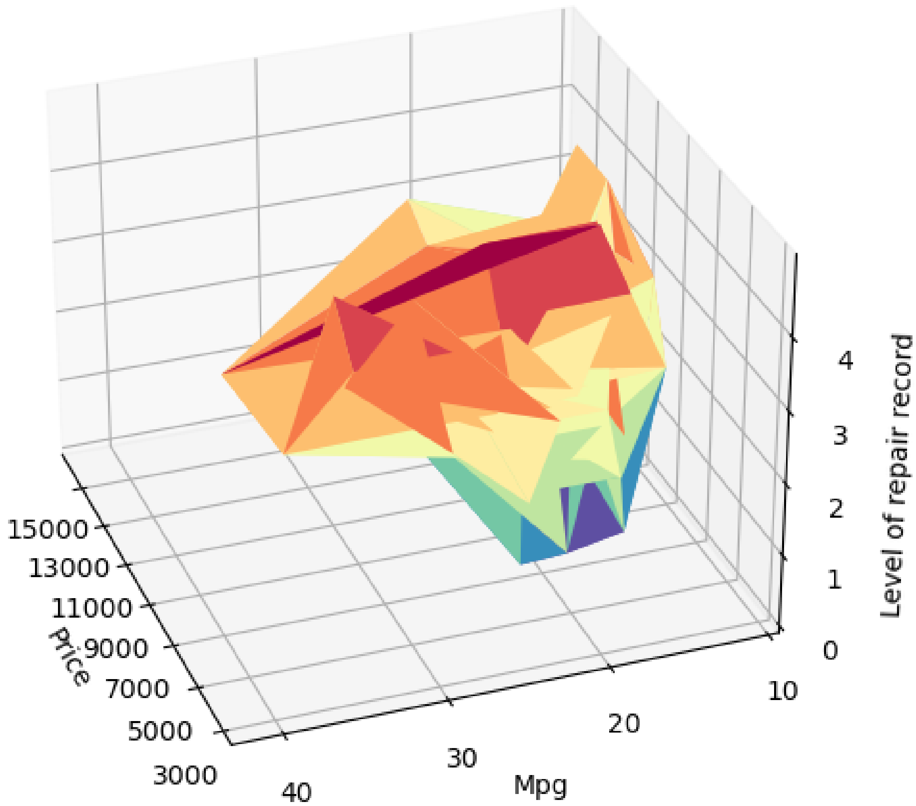
Even though, to gain a more comprehensive understanding of these findings, we need to look deeper into the matter and justify the assumptions by the next econometic modelling.

Next code is contributed by Yuming Zhang (3rd yr UG student of IESR). The code draws 3D plots of variables as stylzed examples.

[5]:
```
import numpy as np
from mpl_toolkits import mplot3d

fig = plt.figure(1, figsize=(10, 6))
ax = plt.axes(projection='3d')

# Draw a 3D plan of the length and weight distribution
```

```
ax.plot_trisurf(data['length'], data['weight'], data['rep77'],cmap=plt.cm.
 ↪Spectral_r)

# Set scale and axis labels
ax.set_xticks(np.arange(165, 240, step=25))
ax.set_yticks(np.arange(2500, 5000, step=500))
ax.set_zticks(np.arange( 0, 5, step=1))
ax.set_xlabel("Length")
ax.set_ylabel("Weight")
ax.set_zlabel("Level of repair record 1977")

# Adjust the angle of the 3D image
ax.view_init(elev=30, azim=140)

# Save the 3D image - with DPI of 600
plt.savefig("3Dplot.png", dpi = 600)
plt.show()

fig = plt.figure(1, figsize=(10, 6))
ax = plt.axes(projection='3d')

# Draw a 3D image of the price and mpg distribution
ax.plot_trisurf(data['price'], data['mpg'], data['rep77'],cmap=plt.cm.
 ↪Spectral_r)

#  Set scale and axis labels
ax.set_xticks(np.arange(3000, 16000, step=2000))
ax.set_yticks(np.arange(10, 50, step=10))
ax.set_zticks(np.arange( 0, 5, step=1))
ax.set_xlabel("Price")
ax.set_ylabel("Mpg")
ax.set_zlabel("Level of repair record")

#  Adjust the angle of the 3D image
ax.view_init(elev=30, azim=160)

#  Save the 3D image and display it with a clarity DPI of 600
plt.savefig("3Dplot.png", dpi = 600)
plt.show()
```


215
190
165
Length
2500
3000
3500
4000
4500
Weight
0
1
2
3
4
Level of repair record 1977



3.1 Python-Stata Integration

Python-Stata integration can be activated on my laptop by executing:

```
# Essential packages
pip install pystata
pip install stata_setup
```

```
(installation processes are omited)
```

```
# Configurations
import os os.chdir('E:/Stata17/utilities')
import sys sys.path.append('E:/Stata17/utilities')
from pystata import config config.init('mp')
```

Then the next code uses Stata to produce some data cleaning operations as well as the generating the table of summary statistics and return results in Python.

[6]:
```
from pystata import stata
# Run a selection of Stata code
```

```
stata.run(
    '''clear all
    sysuse fullauto
    misstable summarize
    drop if rep77==.
    drop if rep78==.
    sum price rep77 rep78 foreign length weight''')
```

```
  ___  ____  ____  ____  ____ ©
 /__    /   ____/   /   ____/      17.0
___/   /   /___/   /   /___/       MP-Parallel Edition

 Statistics and Data Science       Copyright 1985-2021 StataCorp LLC
                                   StataCorp
                                   4905 Lakeway Drive
                                   College Station, Texas 77845 USA
                                   800-STATA-PC        https://www.stata.com
                                   979-696-4600        stata@stata.com

Stata license: Single-user 8-core


Notes:
      1. Unicode is supported; see help unicode_advice.
      2. More than 2 billion observations are allowed; see help obs_advice.
      3. Maximum number of variables is set to 5,000; see help set_maxvar.

. clear all

.     sysuse fullauto
(Automobile Models)

.     misstable summarize
                                                               Obs<.
                                                +------------------------------
               |                                | Unique
      Variable |     Obs=.     Obs>.     Obs<.  | values        Min         Max
  -------------+--------------------------------+------------------------------
         rep78 |         5                  69  |      5          1           5
         rep77 |         8                  66  |      5          1           5
          wgtd |        22                  52  |     47       1800        4840
          wgtf |        52                  22  |     22       1760        3420
  -----------------------------------------------------------------------------

.     drop if rep77==.
(8 observations deleted)
```

```
.     drop if rep78==.
(0 observations deleted)

.     sum price rep77 rep78 foreign length weight

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
       price |         66    6222.576    2955.821       3291      15906
       rep77 |         66     3.19697    .9642805          1          5
       rep78 |         66    3.409091    1.007316          1          5
     foreign |         66    .3181818    .4693397          0          1
      length |         66    189.1212    22.46331        142        233
-------------+---------------------------------------------------------
      weight |         66    3058.182    788.1447       1760       4840

.
```

4 Econometric Analysis

4.1 Using statsmodels Commands

Based on the methodology stated in Section 1, first, I estimate the model parameters using statsmodels to yield preliminary findings.

[16]:

```python
import statsmodels.api as sm
from statsmodels.miscmodels.ordinal_model import OrderedModel
import locale
locale.setlocale(locale.LC_ALL, 'C')

X = data[['foreign', 'length', 'mpg']]
y = data['rep77']
model = OrderedModel(y, X, distr='logit')
result = model.fit(method='bfgs')
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 1.185617
         Iterations: 32
         Function evaluations: 37
         Gradient evaluations: 37
                             OrderedModel Results
==============================================================================
Dep. Variable:                  rep77   Log-Likelihood:                -78.251
Model:                   OrderedModel   AIC:                             170.5
Method:            Maximum Likelihood   BIC:                             185.8
Date:                Mon, 08 Apr 2024
Time:                        23:03:33
```

```
No. Observations:                  66
Df Residuals:                      59
Df Model:                           7
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
foreign        2.8968      0.791      3.664      0.000       1.347       4.446
length         0.0828      0.023      3.646      0.000       0.038       0.127
mpg            0.2308      0.070      3.275      0.001       0.093       0.369
1/2           17.9275      5.551      3.229      0.001       7.047      28.808
2/3            0.6614      0.300      2.203      0.028       0.073       1.250
3/4            0.8057      0.171      4.706      0.000       0.470       1.141
4/5            0.9512      0.218      4.360      0.000       0.524       1.379
==============================================================================
```

For the cutoff points estimates,

[17]:
```
cut_off_points = model.transform_threshold_params(result.params)
print(cut_off_points)
```

```
[       -inf 17.92746199 19.86504069 22.10328917 24.6921089          inf]
```

Based on the estimation conducted above, I would find the following conclusions (note that this is not the effects in the marginal sense):

- All three explanatory variables, "foreign," "length," and "mpg," are highly statistically significant at conventional confidence levels. This indicates that these variables have a significant relationship with the repair records of cars.
- On average, foreign cars have repair records that are 2.8 levels better than non-foreign cars. This result suggests that being a foreign car is associated with a higher likelihood of better durability.
- Additionally, the length of the car and its fuel efficiency (mpg) also significantly affect the repair records. For every unit increase in car length, there is an expected increase in the repair record level by 0.083, holding other variables constant. Similarly, for every unit increase in mpg, the repair record level is expected to increase by 0.231, all else being equal.

4.2 Bootstrapped Standard Errors

Here, I attempt to compute bootstrapped standard errors using the following scripts. The bootstrap algorithm follows the lecture materials delivered in this course, so formulae details are omitted.

Please notice that the bootstrap algorithm makes use of the *sample with replacement*. I find it is easier to utilize Python's machine learning module (`sklearn.utils`) to perform such `resample`.

[19]:
```
import numpy as np
from sklearn.utils import resample

bootstrap_iterations = 1000
bootstrap_estimates = np.zeros((bootstrap_iterations, len(X.columns)))
```

```
for i in range(bootstrap_iterations):
    # Resample the data
    X_sample, y_sample = resample(X, y)

    # Fit the model and get the parameter estimates
    model = OrderedModel(y_sample, X_sample, distr='logit')
    result = model.fit(method='bfgs', disp=0)

    # Store the parameter estimates
    bootstrap_estimates[i, :] = (result.params)[:3].values

# Compute the standard errors
bootstrap_standard_errors = bootstrap_estimates.std(axis=0)

# Print the results
for i, column in enumerate(X.columns):
    print(f'Bootstrap standard error for {column}:␣
 ↪{bootstrap_standard_errors[i]}')
```

```
Bootstrap standard error for foreign: 0.9738380107392711
Bootstrap standard error for length: 0.029093106408630783
Bootstrap standard error for mpg: 0.11230855685709225
```

4.3 Codifying the MLE Manually in Python

Let α and β be vector parameters as defined in Section 1. Before continuing with the programming for optimizations, I first write out the coding friendly representation of the objective function. Define the log-likelihood function ℓ as

$$\ell(\beta, \alpha) := \sum_{i=1}^{n} \sum_{j=1}^{5} \ln\left(\Pr(rep77_i = j | \mathbf{x}_i, \beta, \alpha)\right).$$

Mathematically equivalently, we have

$$\ell(\beta, \alpha) := \sum_{i=1}^{n} \sum_{j=1}^{5} \left[\mathbb{I}(\text{rep77}_i \leq j) \log(\pi_{ij}) + \mathbb{I}(\text{rep77}_i > j) \log(1 - \pi_{ij})\right],$$

where $\mathbb{I}(.)$ is an indicator function, and in specific,

$$\pi_{ij} := \Pr(rep77_i \leq j | \mathbf{x}_i, \beta, \alpha) = \frac{\exp(\alpha_j - \beta_1 foreign_i - \beta_2 length_i - \beta_3 mpg_i)}{1 + \exp(\alpha_j - \beta_1 foreign_i - \beta_2 length_i - \beta_3 mpg_i)}.$$

The model parameters are estimated according to the the maximization of this log-likelihood function.

[9]:
```
from scipy.optimize import minimize
from scipy.special import expit
```

```python
# Extracting data
data[['foreign', 'rep77']] = data[['foreign', 'rep77']].apply(pd.to_numeric)
foreign=data['foreign'].values
rep77=data['rep77'].values
length=data['length'].values
mpg=data['mpg'].values

# Objective function --- the negative of the log-likelihood
def ologit(theta):
    # Sorting parameters of betas
    beta1=theta[0]
    beta2=theta[1]
    beta3=theta[2]

    # Linear combination
    BX=foreign*beta1+length*beta2+mpg*beta3

    # Sorting parameters of alphas
    alpha1=theta[3]
    alpha2=theta[4]
    alpha3=theta[5]
    alpha4=theta[6]

    # Cloning the dep. var.
    optput=np.copy(rep77)

    # Computing the different parts of the log-likelihood
    part0=(np.log(expit(alpha1-BX[rep77==1])-expit(-np.inf-BX[rep77==1])))
    part1=(np.log(expit(alpha2-BX[rep77==2])-expit(alpha1-BX[rep77==2])))
    part2=(np.log(expit(alpha3-BX[rep77==3])-expit(alpha2-BX[rep77==3])))
    part3=(np.log(expit(alpha4-BX[rep77==4])-expit(alpha3-BX[rep77==4])))
    part4=(np.log(expit(np.inf-BX[rep77==5])-expit(alpha4-BX[rep77==5])))
    return -(part0.sum() + part1.sum() + part2.sum() + part3.sum() + part4.
 ↪sum())

# Perform optimization using minimize function
result = minimize(ologit, x0=np.array([2.8, 0.00, 0.29, 17, 19, 22, 24]))

print(result)
```

```
     fun: 78.25071924594866
hess_inv: array([[6.72752859e-01, 1.15050056e-02, 1.18546668e-02, 2.51364525e+00,
        2.54919414e+00, 2.65530318e+00, 2.80674775e+00],
       [1.15050056e-02, 5.35599014e-04, 1.25469933e-03, 1.28353712e-01,
        1.29989307e-01, 1.32596418e-01, 1.36131797e-01],
```

```
        [1.18546668e-02, 1.25469933e-03, 5.21865331e-03, 3.44645321e-01,
         3.48005498e-01, 3.54228063e-01, 3.65211314e-01],
        [2.51364525e+00, 1.28353712e-01, 3.44645321e-01, 3.19641347e+01,
         3.20713307e+01, 3.26532105e+01, 3.35505081e+01],
        [2.54919414e+00, 1.29989307e-01, 3.48005498e-01, 3.20713307e+01,
         3.24833865e+01, 3.30535280e+01, 3.39573863e+01],
        [2.65530318e+00, 1.32596418e-01, 3.54228063e-01, 3.26532105e+01,
         3.30535280e+01, 3.37738022e+01, 3.46812431e+01],
        [2.80674775e+00, 1.36131797e-01, 3.65211314e-01, 3.35505081e+01,
         3.39573863e+01, 3.46812431e+01, 3.59402038e+01]])
       jac: array([-9.53674316e-07,  5.88607788e-03,  5.79833984e-04, -1.04904175e-05,
        -2.09808350e-05,  8.58306885e-06, -9.53674316e-07])
   message: 'Desired error not necessarily achieved due to precision loss.'
      nfev: 473
       nit: 22
      njev: 57
    status: 2
   success: False
         x: array([ 2.89676628,  0.08282575,  0.23076365, 17.92704828, 19.86462704,
        22.10287117, 24.69168257])
```

Note that I chose the initial values close to the MLE. As described earlier, this model may encounter numerical computing issues. Therefore, I will check the concavity of the objective function.

4.4 Python Chat Box for Plotting Likelihood Surfaces

Next script designs a Python chat box for users to interactively select the horizontal axis among all βs in the figure:

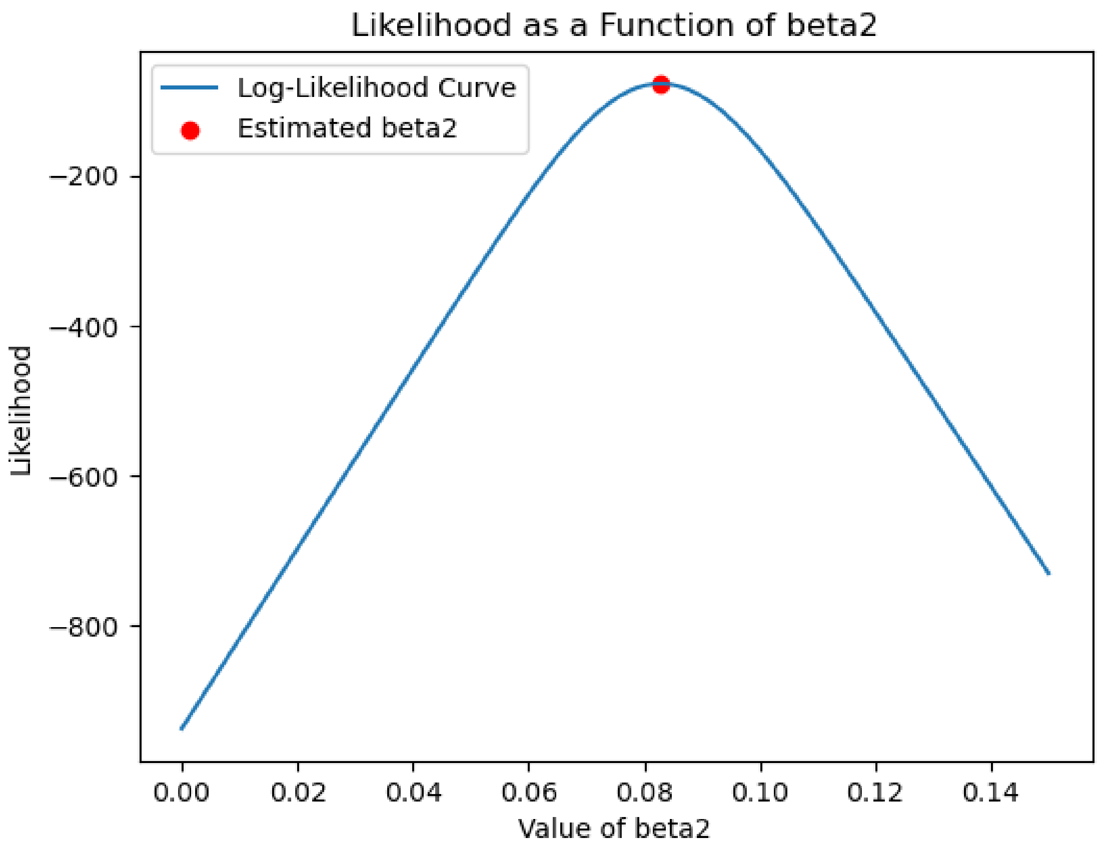
```python
# Here the user is allowed to choose which coefficient to plot
estimated_parameter = input("Enter the coefficient you want to plot (beta1, beta2 or beta3): ")

# Initialize the beta values to the optimal values
beta_values = np.copy(result.x)

# Set the range of values for the selected beta
if estimated_parameter == 'beta1':
    beta_range = np.linspace(-5, 5, 100)
    idx = 0
elif estimated_parameter == 'beta2':
    beta_range = np.linspace(0, 0.15, 100)
    idx = 1
elif estimated_parameter == 'beta3':
    beta_range = np.linspace(-0.5, 0.5, 100)
    idx = 2

# Store the likelihood for each value
likelihood_values = []
```

```
for b in beta_range:
    beta_values[idx] = b
    likelihood_values.append(-ologit(beta_values))

plt.plot(beta_range, likelihood_values, label='Log-Likelihood Curve')

# Calculate the likelihood at the optimal beta
optimal_likelihood = -ologit(result.x)

# Add the optimal point to the plot
plt.scatter(result.x[idx], optimal_likelihood, color='red', label='Estimated ' + estimated_para

plt.xlabel('Value of ' + estimated_parameter)
plt.ylabel('Likelihood')
plt.title('Likelihood as a Function of ' + estimated_parameter)
plt.legend()
plt.show()
```

```
Enter the coefficient you want to plot (beta1, beta2 or beta3): [_______]
```

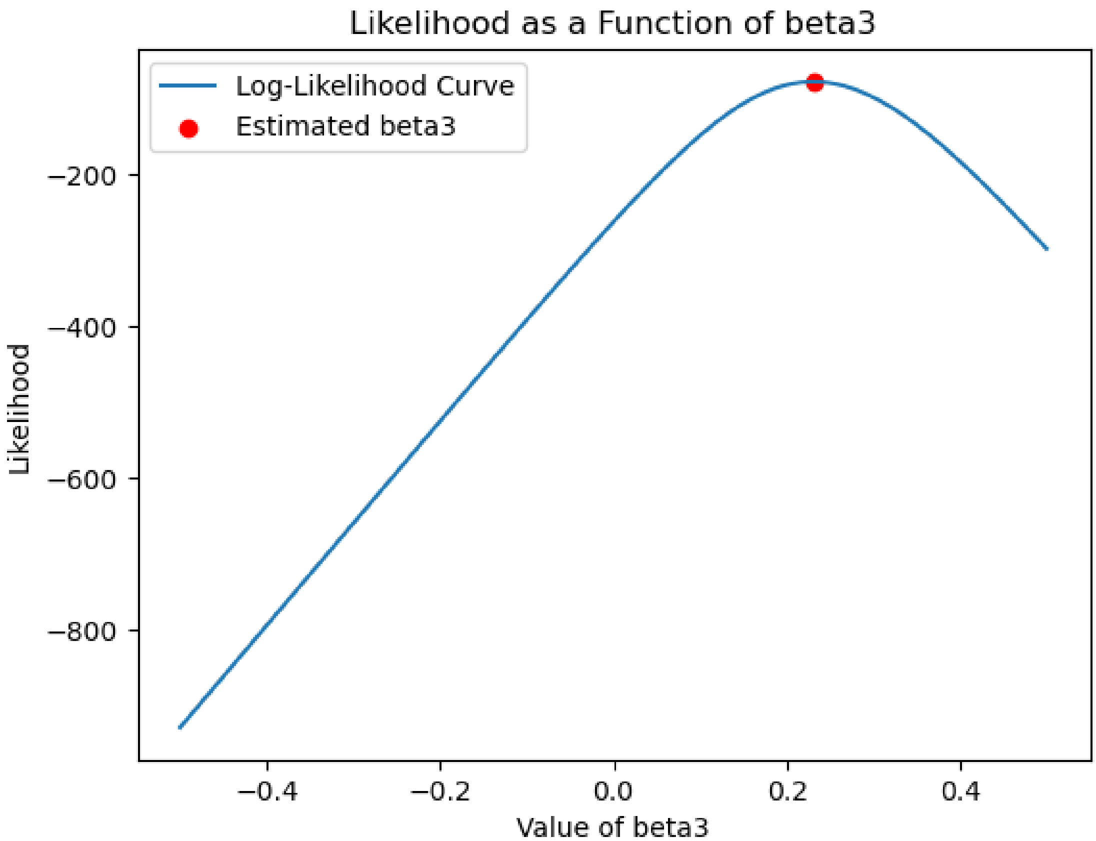
[22]:
```
# (repeated code omitted)
```

```
Enter the coefficient you want to plot (beta1, beta2 or beta3): beta2
```



[21]:
```
# (repeated code omitted)
```

```
Enter the coefficient you want to plot (beta1, beta2 or beta3): beta3
```



4.5 Python-Stata Integrations

We can also use Stata to generate the same estimation.

[12]:
```
stata.run('''
          ologit rep77 foreign length mpg
          ''')
```

```
. use "fullauto.dta"
(Automobile Models)

.            ologit rep77 foreign length mpg
```

```
Iteration 0:   log likelihood = -89.895098
Iteration 1:   log likelihood = -78.775147
Iteration 2:   log likelihood = -78.254294
Iteration 3:   log likelihood = -78.250719
Iteration 4:   log likelihood = -78.250719

Ordered logistic regression                             Number of obs =     66
                                                        LR chi2(3)    =  23.29
                                                        Prob > chi2   = 0.0000
Log likelihood = -78.250719                             Pseudo R2     = 0.1295

------------------------------------------------------------------------------
       rep77 | Coefficient  Std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
     foreign |   2.896807   .7906411     3.66   0.000     1.347179    4.446435
      length |   .0828275     .02272     3.65   0.000     .0382972    .1273579
         mpg |   .2307677   .0704548     3.28   0.001     .0926788    .3688566
-------------+----------------------------------------------------------------
       /cut1 |   17.92748   5.551191                      7.047344    28.80761
       /cut2 |   19.86506    5.59648                      8.896161    30.83396
       /cut3 |   22.10331   5.708936                        10.914    33.29262
       /cut4 |   24.69213   5.890754                      13.14647     36.2378
------------------------------------------------------------------------------

.
```

Also, I compute the marginal effects. This task is easier to do in Stata, for instance,

[13]:
```
stata.run('''mfx''')
```

```
Marginal effects after ologit
      y  = Pr(rep77==1) (predict)
         =  .02707434
------------------------------------------------------------------------------
variable |      dy/dx    Std. err.     z    P>|z|  [    95% C.I.   ]      X
---------+--------------------------------------------------------------------
 foreign*|  -.0615285      .03648   -1.69   0.092  -.133028  .009971   .318182
  length |  -.0021818      .00131   -1.66   0.096  -.004754   .00039   189.121
     mpg |  -.0060787      .00378   -1.61   0.108   -.01349  .001333   21.3333
------------------------------------------------------------------------------
(*) dy/dx is for discrete change of dummy variable from 0 to 1
```