Micro-module 3: Basic Statistical Analysis

In this session, we're going to explore the fundamentals of statistical analysis. Our journey will take us through the essentials of understanding and interpreting data. More than just numbers, we'll uncover how these statistics reveal the intricate connections between various elements in our everyday lives.

Our module covers two key methods: Pearson correlation analysis, which helps us understand the strength and direction of relationships, and Linear regression analysis using the Ordinary Least Squares approach, a powerful tool for predicting and explaining these relationships. In the tutorial, we'll demonstrate how to use these statistical models to conduct meaningful analysis in research.

1. Content

- **Structure and design of the module**

In this module, we're going to break down our module into several parts. We'll start by laying the groundwork with some fundamental concepts. This includes understanding different types of data and variables, giving us a solid base to build on. Next, we'll delve into basic descriptive analysis, where we'll learn how to summarize and describe our data effectively. Moving forward, we'll explore the fascinating aspect of discovering relationships between variables. We'll do this through two key methods: Pearson correlation analysis, which helps us understand the strength and direction of relationships, and Linear regression analysis using the Ordinary Least Squares approach, a powerful tool for predicting and explaining these relationships.

CONTENTS

The theoretical and conceptual part of these statistical models can refer to our videos and this manual is mainly intended to provide guidance for the tutorial parts.

2. Pearson Correlation Analysis

- **Introduction**

  Correlation analysis is a statistical method used to investigate the strength and direction of the linear relationships between two or more random variables. It specifically focuses on studying the linear associations among random variables. In other words, it helps us understand if and how changes in one variable correspond to changes in another.

- **Pearson Correlation**

  Applicable to Interval or Ratio Variables, Pearson correlation coefficient, denoted as r, measures the strength and direction of a linear relationship between two continuous variables. It ranges from -1 (perfect negative correlation) to 1 (perfect positive correlation), with 0 indicating no linear correlation. Interpretation: A positive r suggests a positive correlation, meaning as one variable increases, the other tends to increase. A negative r indicates a negative correlation, where one variable tends to decrease as the other increases.

- **Understanding the data**

  We use a dataset includes built environment features and people's sentiment level by tweets density and tweets sentiment score in Hong Kong.

- **Check the data frame columns, and drop irrelevant columns**

```
[32] 1  xy_grid_300 = pd.read_csv('xy_grid_300.csv')
```

```
[34] 1  xy_grid_300.head()
```

	Unnamed: 0	GID300	water	sea	lake	sky	ceiling	wall	railing	fence	...	bus_count	bert_score
0	0	6020.0	2.696940e-03	0.009256	0.0	0.171062	0.000000	0.054596	0.028240	0.025273	...	3.0	1.00000
1	1	6196.0	0.000000e+00	0.000256	0.0	0.186852	0.000000	0.024781	0.003231	0.007602	...	0.0	1.00000
2	2	7900.0	8.958333e-07	0.000000	0.0	0.155079	0.000000	0.025409	0.012748	0.033251	...	0.0	1.00000
3	3	8916.0	0.000000e+00	0.000000	0.0	0.110615	0.000235	0.020692	0.013527	0.058213	...	3.0	1.52381
4	4	9085.0	2.285160e-05	0.000000	0.0	0.083890	0.000076	0.019702	0.009002	0.117858	...	2.0	1.00000

5 rows × 67 columns

```
[35] 1  xy_grid_300.columns
```

```
Index(['Unnamed: 0', 'GID300', 'water', 'sea', 'lake', 'sky', 'ceiling',
       'wall', 'railing', 'fence', 'skyscraper', 'road', 'sidewalk', 'floor',
       'building', 'signboard', 'bannister', 'poster', 'person', 'car',
       'windowpane', 'pole', 'light', 'green', 'blue', 'enclo', 'imgage',
       'complex', 'walk', 'vader_mean', 'pos_c', 'neg_c', 'senti_c',
       'overallsco', 'score', 'score_shop', 'score_majo', 'score_bus',
       'score_rest', 'score_ente', 'score_park', 'score_spor', 'score_scho',
       'score_dome', 'score_heal', 'entroscr', 'street_den', 'inter_den',
       'LLen_agg', 'LConn_agg', 'NQPDA800_agg', 'NQPDA2000_agg',
       'NQPDA5000_agg', 'BtA800_agg', 'BtA2000_agg', 'TPBtA800_agg',
       'metro_count', 'bus_count', 'bert_score', 'count_station',
       'count_Airbnb', 'count_hotel', 'count_POI', 'distance_cityCenter(m)',
       'distance_Metro(m)', 'area_park(m)', 'attraction_count'],
      dtype='object')
```

- **Log transformation for dependent variables**

  It's not a compulsory step, but log transformation would be very useful when dealing with real world data. This process is useful for compressing the y-axis when plotting histograms. For example, if we have a very large range of data, then smaller values can get overwhelmed by the larger values. Taking the log of each variable enables the visualization to be clearer.

```python
xy_grid_300 = xy_grid_300.drop(columns = ['Unnamed: 0', 'GID300', 'attraction_count', 'neg_c', 'pos_c'])
```

```python
xy_grid_300['senti_log'] = np.log(xy_grid_300['senti_c'])
```

```python
[18] xy_grid_300 = xy_grid_300.drop(columns = ['senti_c'])
```

- **Create correlation matrix**

  By this codes you can generate the r value for each pair of variables in our dataset.

```python
[39] qua_senti_0 = xy_grid_300
```

```python
[40] correlation_matrix = qua_senti_0.corr(method='pearson')
```

```python
correlation_matrix.head()
```

	water	sea	lake	sky	ceiling	wall	railing	fence	skyscraper	road	...	bus_count	bert_score	count_station	count_Airbnb	count_hotel	count_POI	distance_cityCenter(m)	distance_Metro(m)	area_park(m)	senti_log
water	1.000000	0.250978	-0.000255	0.116816	-0.037957	0.077908	-0.027573	-0.023716	-0.045288	-0.138615	...	-0.084718	-0.000521	-0.112997	-0.041086	-0.051515	-0.067898	0.023008	0.069689	-0.094080	-0.052412
sea	0.250978	1.000000	-0.003337	0.197392	-0.035154	-0.008195	0.034762	-0.052345	-0.028382	-0.049947	...	-0.055167	0.091321	-0.112754	-0.035762	-0.044507	-0.065543	-0.010390	0.048430	-0.111637	0.000370
lake	-0.000255	-0.003337	1.000000	0.009301	-0.009669	0.056968	-0.019898	-0.025317	-0.014438	-0.031397	...	-0.030048	-0.022182	-0.030049	-0.011391	-0.012572	-0.021248	-0.006645	-0.006789	-0.037696	-0.030004
sky	0.116816	0.197392	0.009301	1.000000	-0.105339	-0.080428	0.035563	0.041679	0.043325	-0.149303	...	-0.194801	0.063816	-0.272395	-0.100937	-0.149171	-0.197071	0.152849	0.294763	-0.133745	-0.155893
ceiling	-0.037957	-0.035154	-0.009669	-0.105339	1.000000	0.348712	-0.046490	-0.166383	-0.045208	0.037562	...	0.081862	-0.006287	0.078564	0.100405	0.114802	0.102378	-0.006553	-0.159640	0.074336	0.139887

5 rows × 62 columns

- **Calculate the p-value and selected statistic correlated variables;**

```python
from scipy.stats import pearsonr

# Get column names
cols = qua_senti_0.columns

# Create an empty DataFrame to store the results
significant_correlations = pd.DataFrame(columns=['index1', 'index2', 'correlation', 'p-value'])

# Loop over all pairs of columns
for i in range(len(cols)):
    for j in range(i+1, len(cols)):
        # Convert columns to numeric type
        col_i = pd.to_numeric(qua_senti_0[cols[i]], errors='coerce')
        col_j = pd.to_numeric(qua_senti_0[cols[j]], errors='coerce')

        # Remove any NaN values
        mask = ~np.isnan(col_i) & ~np.isnan(col_j)
        if np.sum(mask) >= 2:
            corr, p_value = pearsonr(col_i[mask], col_j[mask])

            # If p-value is less than 0.05, the correlation is considered significant
            if p_value < 0.05:
                new_row = pd.DataFrame({'index1': [cols[i]], 'index2': [cols[j]], 'correlation': [corr], 'p-value': [p_value]})
                significant_correlations = pd.concat([significant_correlations, new_row], ignore_index=True)

# Print the significant correlations
print(significant_correlations)
```

```
                   index1        index2  correlation       p-value
0                   water           sea     0.250978  2.376436e-18
1                   water           sky     0.116816  5.933601e-05
2                   water          wall     0.077908  7.519862e-03
3                   water          road    -0.138615  1.828537e-06
4                   water      sidewalk    -0.127616  1.135388e-05
...                   ...           ...          ...           ...
1347  distance_cityCenter(m)  area_park(m)    -0.327262  9.335493e-31
1348  distance_cityCenter(m)     senti_log    -0.173771  1.991883e-09
1349       distance_Metro(m)  area_park(m)    -0.380442  8.457694e-42
1350       distance_Metro(m)     senti_log    -0.220990  1.784313e-14
1351            area_park(m)     senti_log     0.090164  1.968231e-03

[1352 rows x 4 columns]
```

- **Filter out specific correlated variables**

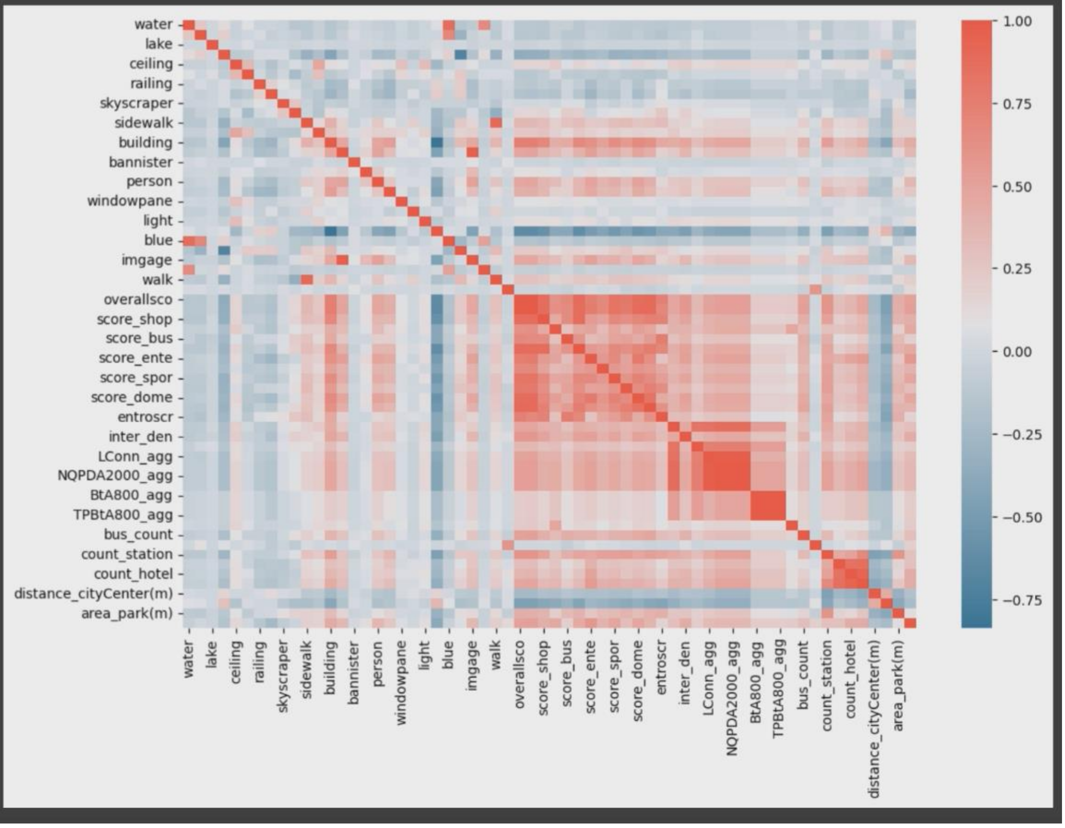
```python
#为correration matrix只保留小数点后三位
correlation_matrix = correlation_matrix.round(3)
significant_correlations = significant_correlations.round(3)
```

```python
significant_correlations[(significant_correlations['index1'] == 'senti_log') | (significant_correlations['index2'] == 'senti_log')]
```

```python
significant_correlations.to_csv('corr.csv', index=False)
```

- **Visualize the correlation matrix**

```
# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)
fig, ax = plt.subplots(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=False, cmap=cmap)
fig.set_facecolor('white')
plt.savefig('correlation.png')
plt.show()
```



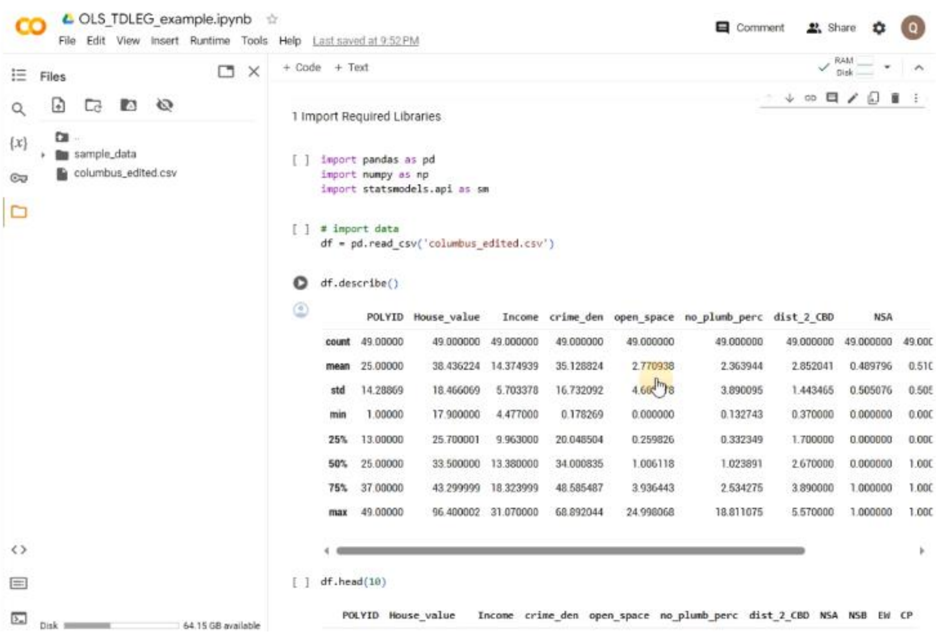
3. OLS Regression Analysis

- Understanding the data

we will use a dataset related to crime in 1980 in the city of Columbus, which is the capital city of Ohio State in the USA.

- Open the notebook and csv file in the Google Colab

open Google Colab Notebook website, upload the notebook to open it, then drag the csv file to the left panel to upload the data for the code.



- Descriptive Analysis

Importing the necessary libraries such as pandas, numpy and statsmodels.

Calling the 'describe' function to obtain the count, mean, standard deviation, min, max, Q1, Q2 and Q3. This function allows us to examine the data conveniently as the first step.

1 Import Required Libraries

```
[1] import pandas as pd
    import numpy as np
    import statsmodels.api as sm
```

```
[2] # import data
    df = pd.read_csv('columbus_edited.csv')
```

```
df.describe()
```

	POLYID	House_value	Income	crime_den	open_space	no_plumb_perc	dist_2_CBD	NSA	NSB	EW	CP
count	49.00000	49.000000	49.000000	49.000000	49.000000	49.000000	49.000000	49.000000	49.000000	49.000000	49.000000
mean	25.00000	38.436224	14.374939	35.128824	2.770938	2.363944	2.852041	0.489796	0.510204	0.591837	0.489796
std	14.28869	18.466069	5.703378	16.732092	4.668078	3.890095	1.443465	0.505076	0.505076	0.496587	0.505076
min	1.00000	17.900000	4.477000	0.178269	0.000000	0.132743	0.370000	0.000000	0.000000	0.000000	0.000000
25%	13.00000	25.700001	9.963000	20.048504	0.259826	0.332349	1.700000	0.000000	0.000000	0.000000	0.000000
50%	25.00000	33.500000	13.380000	34.000835	1.006118	1.023891	2.670000	0.000000	1.000000	1.000000	0.000000
75%	37.00000	43.299999	18.323999	48.585487	3.936443	2.534275	3.890000	1.000000	1.000000	1.000000	1.000000
max	49.00000	96.400002	31.070000	68.892044	24.998068	18.811075	5.570000	1.000000	1.000000	1.000000	1.000000

- **Printing the first 10 rows**
  Printing the first 10 rows of data, you could compare it with the csv directly opened through excel, just to make sure everything is correct.

```
df.head(10)
```

	POLYID	House_value	Income	crime_den	open_space	no_plumb_perc	dist_2_CBD	NSA	NSB	EW	CP
0	1	80.467003	19.531000	15.725980	2.850747	0.217155	5.03	1	1	1	0
1	2	44.567001	21.232000	18.801754	5.296720	0.320581	4.27	1	1	0	0
2	3	26.350000	15.956000	30.626781	4.534649	0.374404	3.89	1	1	1	0
3	4	33.200001	4.477000	32.387760	0.394427	1.186944	3.70	1	1	0	0
4	5	23.225000	11.252000	50.731510	0.405664	0.624596	2.83	1	1	1	0
5	6	28.750000	16.028999	26.066658	0.563075	0.254130	3.78	1	1	1	0
6	7	75.000000	8.438000	0.178269	0.000000	2.402402	2.74	1	1	0	0
7	8	37.125000	11.337000	38.425858	3.483478	2.739726	2.89	1	1	0	0
8	9	52.599998	17.586000	30.515917	0.527488	0.890736	3.17	1	1	1	0
9	10	96.400002	13.598000	34.000835	1.548348	0.557724	4.33	1	1	1	0

- **Running a Simple OLS regression**
  Our research wants to understand whether the housing price is going to influence the crime rate in the neighbourhood and by how much. In our analysis, our dependent variable is the crime density here represented as Y, and X is the independent variable, which we will choose the Income as the variable. After writing the code to add a constant and intercept term beta 0, we are going to fit the OLS model, we name it as model 1. Then we could also get the model result by printing its summary.

Simple OLS Linear Regression Model

```
#Fit a Simple OLS Regression model
# study the relationship between average household Income and Crime density.

x = df['Income']

y = df['crime_den']

x = sm.add_constant(x)  # Add a constant term to the X matrix

model1 = sm.OLS(y, x)  # Specify the dependent variable (y) and independent variable (x) for the model
result1 = model1.fit()  # Fit the model
```

```
[ ] print(result1.summary())  # Print the model summary
```

- **Interpreting Simple OLS regression result**
  Several important things we need to check and evaluate this model. First, the overall model seems to show a decent fit. The R2 of OLS model is 0.484, which means it could explain 48.4% of the variance in the dependent variable.
  Next, we want to see what the impact of Income is. The p-value shows it could reject

the null hypothesis, meaning it has significant impact. And then the coefficient is -2.04, it is negatively correlated with the crime. And also means that every 1 unit increase in household income leads to the drop in crime density by 2 cases in every 1000 people in the neighborhood.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              crime_den   R-squared:                       0.484
Model:                            OLS   Adj. R-squared:                  0.473
Method:                 Least Squares   F-statistic:                     44.06
Date:                Sun, 23 Jul 2023   Prob (F-statistic):           2.90e-08
Time:                        06:11:41   Log-Likelihood:                -190.87
No. Observations:                  49   AIC:                             385.7
Df Residuals:                      47   BIC:                             389.5
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         64.4632      4.748     13.577      0.000      54.912      74.015
Income        -2.0407      0.307     -6.638      0.000      -2.659      -1.422
==============================================================================
Omnibus:                       16.605   Durbin-Watson:                   1.819
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               36.549
Skew:                          -0.835   Prob(JB):                     1.16e-08
Kurtosis:                       6.888   Cond. No.                         42.4
==============================================================================

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

Impact: every 1 unit increase in independent variable (i.e., income) lead to the drop in crime density by 2 cases.

-

- **Conducting Multiple Linear Regression (MLR)**

  Although we have explored the simple linear regression model, but it could only estimate the effect of a single variable each time. What if we want to know how do all the 9 variables jointly affect the crime density? In this case, we will add more variables into the model, and it becomes the multiple linear regression.

- **Checking the VIF**

  To do the MLR, we need to check the multicollinearity issue among our independent variables. So we could import the VIF module from the library, then we include all the independent variables into it, then running these line of code returns the value of VIF for each variable.

  We now detect the issue, for instance, there are many variables will VIF higher than 10.

```
[9] # import the modules of VIF
    from statsmodels.stats.outliers_influence import variance_inflation_factor

# Select the columns for which you want to calculate the VIF
cols = ['House_value','Income', 'open_space',
            'no_plumb_perc','dist_2_CBD','NSB','NSA','EW','CP']


# Calculate VIF for each column
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(df[cols].values, i) for i in range(len(cols))]
vif["features"] = cols

# Print the results
print(vif)

   VIF Factor       features
0   10.543929    House_value
1   17.372066         Income
2    1.583814     open_space
3    2.948480  no_plumb_perc
4   17.236252     dist_2_CBD
5   37.368598            NSB
6   38.572134            NSA
7    3.072969             EW
8    2.831274             CP
```

- **Deleting some variables and re-examining the VIF**
- We need to delete them in order to not inflate the OLS and it clearly violates its assumptions.

  In our final model, we will only include 5 variables, which are income, open space, plumbing percentage, EW and CP. And we recalculate the VIF of the modified list of variables. they are below 4, which means the correlation coefficient falls under 0.75. This meets the requirements to do MLR.

```
# Select the columns for which you want to calculate the VIF (revised)
cols2 = ['Income', 'open_space',
            'no_plumb_perc','EW','CP']


# Calculate VIF for each column
vif = pd.DataFrame()
vif["VIF Factor"] = [variance_inflation_factor(df[cols2].values, i) for i in range(len(cols2))]
vif["features"] = cols2

# Print the results
print(vif)

   VIF Factor       features
0    3.156801         Income
1    1.498505     open_space
2    2.119176  no_plumb_perc
3    3.014115             EW
4    2.449208             CP
```

- **Running MLR**

  And now we could add these 5 variables into the OLS and run the model 3 separately and let's report the result.

```
#Fit a revised Multiple OLS model

X = df[['Income', 'open_space', 'no_plumb_perc','EW','CP']]

y = df['crime_den']

X = sm.add_constant(X)  # Add a constant term to the X matrix

model3 = sm.OLS(y, X)  # Specify the dependent variable (y) and independent variables (X) for the model
result3 = model3.fit()  # Fit the model

[ ] print(result3.summary())  # Print the model summary
```

-
-

- **Interpreting MLR result**

  The overall performance of the model is good, the 5 variables jointly could explain roughly 70% of the variance in crime density.

  And after solving the multicollinearity issue in the model, we could see the p value of variables show that the Income and CP variable both are statistically significant now because they are lower than 0.05.

  We can see that the coefficient of Income, now shows that every 1 unit increase in the income could lead to the drop in crime density by 1.14 cases in the neighbourhood.

  And on average, a neighbourhood in the core urban area is more likely to have crimes.

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              crime_den   R-squared:                       0.684
Model:                            OLS   Adj. R-squared:                  0.647
Method:                 Least Squares   F-statistic:                     18.58
Date:                Sun, 23 Jul 2023   Prob (F-statistic):           8.54e-10
Time:                        07:11:10   Log-Likelihood:                -178.88
No. Observations:                  49   AIC:                             369.8
Df Residuals:                      43   BIC:                             381.1
Df Model:                           5
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const           41.6321      5.927      7.024      0.000      29.679      53.586
Income          -1.1438      0.327     -3.494      0.001      -1.804      -0.484
open_space      -0.2117      0.324     -0.653      0.517      -0.865       0.442
no_plumb_perc    0.7136      0.460      1.550      0.128      -0.215       1.642
EW               2.5232      3.244      0.778      0.441      -4.020       9.066
CP              14.9966      3.980      3.768      0.000       6.970      23.024
==============================================================================
Omnibus:                       12.332   Durbin-Watson:                   2.159
Prob(Omnibus):                  0.002   Jarque-Bera (JB):               14.857
Skew:                          -0.889   Prob(JB):                     0.000594
```