Multivariate Regression Analysis

We will be performing a multiple regression analysis on South Korea's GDP growth. Our goal is to be able to predict what the GDP growth rate will be in any year, given a few explanatory variables that we will define below.

Assumptions of the Model

Here is a list of the assumptions of the model:

> 1. Regression residuals must be normally distributed.
> 2. A linear relationship is assumed between the dependent variable and the independent variables.
> 3. The residuals are homoscedastic and approximately rectangular-shaped.
> 4. Absence of multicollinearity is expected in the model, meaning that independent variables are not too highly correlated.
> 5. No Autocorrelation of the residuals.

(1) Import Python Libraries

In [3]:

```
import numpy as np
import pandas as pd
import seaborn as sns
from scipy import stats
import matplotlib.pyplot as plt

import statsmodels.api as sm
from statsmodels.stats import diagnostic as diag
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.stats.stattools import durbin_watson

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

import pylab
import math

%matplotlib inline
```

(2) Loading the Data Set

In [6]:

```
path = r"D:\IITG\portfolio_finance\multivariate_regression\korea_data.xlsx"
```

In [7]:

```
econ_df = pd.read_excel('korea_data.xlsx')
econ_df
```

Out[7]:

	Year	GDP growth (annual %)	Gross capital formation (% of GDP)	Population growth (annual %)	Birth rate, crude (per 1,000 people)	Broad money growth (annual %)	Final consumption expenditure (annual % growth)	go con ex (
0	1969	14.541235	29.943577	2.263434	30.663	60.984733	10.693249	
1	1970	9.997407	26.338200	2.184174	31.2	27.422864	10.161539	
2	1971	10.454693	25.558501	1.971324	31.2	20.844481	9.330434	
3	1972	7.150715	21.404761	1.875999	28.4	33.815028	5.788458	
4	1973	14.827554	25.872858	1.768293	28.3	36.415629	8.089952	
5	1974	9.460873	32.533408	1.712705	26.6	24.036652	7.323853	
6	1975	7.863512	28.959267	1.682000	24.8	28.231630	5.714445	
7	1976	13.115159	27.555990	1.596559	22.2	33.484656	7.182714	
8	1977	12.277661	30.630713	1.559039	22.7	39.705763	5.701161	
9	1978	10.774491	34.532492	1.519197	20.3	34.971026	7.924074	
10	1979	8.625632	38.132587	1.516875	23	24.584449	8.252856	
11	1980	-1.701277	34.455668	1.558463	22.6	26.894645	1.604241	
12	1981	7.180511	32.445665	1.560204	22.4	25.024951	5.447428	
13	1982	8.265021	32.208672	1.545469	21.6	27.011414	6.523265	
14	1983	13.242063	32.782495	1.474219	19.3	15.242662	8.165829	
15	1984	10.442911	32.248290	1.234018	16.7	7.705292	7.092262	
16	1985	7.749646	32.565945	0.984566	16.1	15.622989	6.539061	
17	1986	11.224086	32.422167	0.994724	15.4	18.441026	8.448972	
18	1987	12.467266	33.089749	0.985133	15	19.053784	8.209374	
19	1988	11.904719	34.704186	0.979189	15.1	21.498577	9.102648	
20	1989	7.029710	37.096222	0.989093	15.1	19.816453	10.834413	
21	1990	9.811230	39.615998	0.985130	15.2	17.174170	10.066888	
22	1991	10.353951	41.374062	0.989786	16.4	21.887421	8.168925	
23	1992	6.175506	38.485198	1.039161	16.7	14.941525	6.777232	
24	1993	6.846744	37.479541	1.015821	16	16.580735	6.437389	
25	1994	9.206142	38.539129	1.006157	16	18.677391	7.957155	
26	1995	9.570604	39.003294	1.006201	15.7	15.593184	9.279911	
27	1996	7.594509	39.680962	0.952779	15	15.827744	7.545604	
28	1997	5.922185	37.424679	0.937714	14.4	14.143724	3.794729	
29	1998	-5.471219	27.761895	0.721865	13.6	27.026185	-9.288825	
30	1999	11.308621	30.916296	0.710795	13	27.376640	10.399817	

	Year	GDP growth (annual %)	Gross capital formation (% of GDP)	Population growth (annual %)	Birth rate, crude (per 1,000 people)	Broad money growth (annual %)	Final consumption expenditure (annual % growth)	go con ex (
31	2000	8.924426	32.941715	0.836181	13.3	25.425775	7.570168	
32	2001	4.525307	31.559587	0.767242	11.6	85.203081	5.774236	
33	2002	7.432434	30.939581	0.577957	10.2	13.999891	8.284738	
34	2003	2.933218	32.014910	0.518321	10.2	2.980690	0.246284	
35	2004	4.899840	32.117074	0.396331	9.8	6.308320	1.060099	
36	2005	3.923677	32.163065	0.211998	8.9	6.989059	4.441178	
37	2006	5.176154	32.700688	0.525200	9.2	12.512961	5.152136	
38	2007	5.463396	32.579531	0.505234	10	10.819965	5.298003	
39	2008	2.829223	33.018504	0.759317	9.4	11.956204	2.152043	
40	2009	0.707510	28.465658	0.514683	9	9.885949	1.271568	
41	2010	6.496794	32.022875	0.498225	9.4	5.978876	4.258940	
42	2011	3.681689	32.958833	0.768972	9.4	5.475862	2.746709	
43	2012	2.292398	31.001229	0.525714	9.6	4.806465	2.246066	
44	2013	2.896205	29.102217	0.455219	8.6	4.638891	2.191130	
45	2014	3.341448	29.276910	0.628150	8.6	8.144493	2.042800	
46	2015	2.790236	28.918112	0.527288	8.6	8.190748	2.385188	
47	2016	2.929305	29.252387	0.451318	7.9	7.123156	2.981150	
48	2017	3.062768	31.075651	0.429345	..	5.104741	2.802722	

(3) Cleaning the Data Set

In [8]:

```
econ_df = econ_df.replace('..','nan')
econ_df
```

Out[8]:

	Year	GDP growth (annual %)	Gross capital formation (% of GDP)	Population growth (annual %)	Birth rate, crude (per 1,000 people)	Broad money growth (annual %)	Final consumption expenditure (annual % growth)	go con ex (
0	1969	14.541235	29.943577	2.263434	30.663	60.984733	10.693249	
1	1970	9.997407	26.338200	2.184174	31.2	27.422864	10.161539	
2	1971	10.454693	25.558501	1.971324	31.2	20.844481	9.330434	
3	1972	7.150715	21.404761	1.875999	28.4	33.815028	5.788458	
4	1973	14.827554	25.872858	1.768293	28.3	36.415629	8.089952	
5	1974	9.460873	32.533408	1.712705	26.6	24.036652	7.323853	
6	1975	7.863512	28.959267	1.682000	24.8	28.231630	5.714445	
7	1976	13.115159	27.555990	1.596559	22.2	33.484656	7.182714	
8	1977	12.277661	30.630713	1.559039	22.7	39.705763	5.701161	
9	1978	10.774491	34.532492	1.519197	20.3	34.971026	7.924074	
10	1979	8.625632	38.132587	1.516875	23	24.584449	8.252856	
11	1980	-1.701277	34.455668	1.558463	22.6	26.894645	1.604241	
12	1981	7.180511	32.445665	1.560204	22.4	25.024951	5.447428	
13	1982	8.265021	32.208672	1.545469	21.6	27.011414	6.523265	
14	1983	13.242063	32.782495	1.474219	19.3	15.242662	8.165829	
15	1984	10.442911	32.248290	1.234018	16.7	7.705292	7.092262	
16	1985	7.749646	32.565945	0.984566	16.1	15.622989	6.539061	
17	1986	11.224086	32.422167	0.994724	15.4	18.441026	8.448972	
18	1987	12.467266	33.089749	0.985133	15	19.053784	8.209374	
19	1988	11.904719	34.704186	0.979189	15.1	21.498577	9.102648	
20	1989	7.029710	37.096222	0.989093	15.1	19.816453	10.834413	
21	1990	9.811230	39.615998	0.985130	15.2	17.174170	10.066888	
22	1991	10.353951	41.374062	0.989786	16.4	21.887421	8.168925	
23	1992	6.175506	38.485198	1.039161	16.7	14.941525	6.777232	
24	1993	6.846744	37.479541	1.015821	16	16.580735	6.437389	
25	1994	9.206142	38.539129	1.006157	16	18.677391	7.957155	
26	1995	9.570604	39.003294	1.006201	15.7	15.593184	9.279911	
27	1996	7.594509	39.680962	0.952779	15	15.827744	7.545604	
28	1997	5.922185	37.424679	0.937714	14.4	14.143724	3.794729	
29	1998	-5.471219	27.761895	0.721865	13.6	27.026185	-9.288825	
30	1999	11.308621	30.916296	0.710795	13	27.376640	10.399817	

	Year	GDP growth (annual %)	Gross capital formation (% of GDP)	Population growth (annual %)	Birth rate, crude (per 1,000 people)	Broad money growth (annual %)	Final consumption expenditure (annual % growth)	go con ex (
31	2000	8.924426	32.941715	0.836181	13.3	25.425775	7.570168	
32	2001	4.525307	31.559587	0.767242	11.6	85.203081	5.774236	
33	2002	7.432434	30.939581	0.577957	10.2	13.999891	8.284738	
34	2003	2.933218	32.014910	0.518321	10.2	2.980690	0.246284	
35	2004	4.899840	32.117074	0.396331	9.8	6.308320	1.060099	
36	2005	3.923677	32.163065	0.211998	8.9	6.989059	4.441178	
37	2006	5.176154	32.700688	0.525200	9.2	12.512961	5.152136	
38	2007	5.463396	32.579531	0.505234	10	10.819965	5.298003	
39	2008	2.829223	33.018504	0.759317	9.4	11.956204	2.152043	
40	2009	0.707510	28.465658	0.514683	9	9.885949	1.271568	
41	2010	6.496794	32.022875	0.498225	9.4	5.978876	4.258940	
42	2011	3.681689	32.958833	0.768972	9.4	5.475862	2.746709	
43	2012	2.292398	31.001229	0.525714	9.6	4.806465	2.246066	
44	2013	2.896205	29.102217	0.455219	8.6	4.638891	2.191130	
45	2014	3.341448	29.276910	0.628150	8.6	8.144493	2.042800	
46	2015	2.790236	28.918112	0.527288	8.6	8.190748	2.385188	
47	2016	2.929305	29.252387	0.451318	7.9	7.123156	2.981150	
48	2017	3.062768	31.075651	0.429345	nan	5.104741	2.802722	

```
econ_df = econ_df.set_index('Year')
econ_df.head()
```

Out[9]:

	GDP growth (annual %)	Gross capital formation (% of GDP)	Population growth (annual %)	Birth rate, crude (per 1,000 people)	Broad money growth (annual %)	Final consumption expenditure (annual % growth)	Ge govern consum expenc (annu grc
Year							
1969	14.541235	29.943577	2.263434	30.663	60.984733	10.693249	10
1970	9.997407	26.338200	2.184174	31.2	27.422864	10.161539	7
1971	10.454693	25.558501	1.971324	31.2	20.844481	9.330434	8
1972	7.150715	21.404761	1.875999	28.4	33.815028	5.788458	8
1973	14.827554	25.872858	1.768293	28.3	36.415629	8.089952	2

In [10]:

```
econ_df = econ_df.astype(float)
econ_df.dtypes
```

Out[10]:

```
GDP growth (annual %)
float64
Gross capital formation (% of GDP)
float64
Population growth (annual %)
float64
Birth rate, crude (per 1,000 people)
float64
Broad money growth (annual %)
float64
Final consumption expenditure (annual % growth)
float64
General government final consumption expenditure (annual % growth)
float64
Gross capital formation (annual % growth)
float64
Households and NPISHs Final consumption expenditure (annual % growth)
float64
Unemployment, total (% of total labor force) (national estimate)
float64
dtype: object
```

In [11]:

```
econ_df = econ_df.loc['1969':'2016']
econ_df
```

Out[11]:

	GDP growth (annual %)	Gross capital formation (% of GDP)	Population growth (annual %)	Birth rate, crude (per 1,000 people)	Broad money growth (annual %)	Final consumption expenditure (annual % growth)	Ge govern consum expenc (annı grc
Year							
1969	14.541235	29.943577	2.263434	30.663	60.984733	10.693249	10
1970	9.997407	26.338200	2.184174	31.200	27.422864	10.161539	7
1971	10.454693	25.558501	1.971324	31.200	20.844481	9.330434	8
1972	7.150715	21.404761	1.875999	28.400	33.815028	5.788458	8
1973	14.827554	25.872858	1.768293	28.300	36.415629	8.089952	2
1974	9.460873	32.533408	1.712705	26.600	24.036652	7.323853	7
1975	7.863512	28.959267	1.682000	24.800	28.231630	5.714445	6
1976	13.115159	27.555990	1.596559	22.200	33.484656	7.182714	0
1977	12.277661	30.630713	1.559039	22.700	39.705763	5.701161	7
1978	10.774491	34.532492	1.519197	20.300	34.971026	7.924074	5
1979	8.625632	38.132587	1.516875	23.000	24.584449	8.252856	6
1980	-1.701277	34.455668	1.558463	22.600	26.894645	1.604241	11
1981	7.180511	32.445665	1.560204	22.400	25.024951	5.447428	6
1982	8.265021	32.208672	1.545469	21.600	27.011414	6.523265	2
1983	13.242063	32.782495	1.474219	19.300	15.242662	8.165829	3
1984	10.442911	32.248290	1.234018	16.700	7.705292	7.092262	4
1985	7.749646	32.565945	0.984566	16.100	15.622989	6.539061	3
1986	11.224086	32.422167	0.994724	15.400	18.441026	8.448972	5
1987	12.467266	33.089749	0.985133	15.000	19.053784	8.209374	7
1988	11.904719	34.704186	0.979189	15.100	21.498577	9.102648	8
1989	7.029710	37.096222	0.989093	15.100	19.816453	10.834413	11
1990	9.811230	39.615998	0.985130	15.200	17.174170	10.066888	11
1991	10.353951	41.374062	0.989786	16.400	21.887421	8.168925	5
1992	6.175506	38.485198	1.039161	16.700	14.941525	6.777232	6
1993	6.846744	37.479541	1.015821	16.000	16.580735	6.437389	4
1994	9.206142	38.539129	1.006157	16.000	18.677391	7.957155	3
1995	9.570604	39.003294	1.006201	15.700	15.593184	9.279911	3
1996	7.594509	39.680962	0.952779	15.000	15.827744	7.545604	7
1997	5.922185	37.424679	0.937714	14.400	14.143724	3.794729	2
1998	-5.471219	27.761895	0.721865	13.600	27.026185	-9.288825	3

	GDP growth (annual %)	Gross capital formation (% of GDP)	Population growth (annual %)	Birth rate, crude (per 1,000 people)	Broad money growth (annual %)	Final consumption expenditure (annual % growth)	Ge govern consum expenc (annu grc
Year							
1999	11.308621	30.916296	0.710795	13.000	27.376640	10.399817	4
2000	8.924426	32.941715	0.836181	13.300	25.425775	7.570168	0
2001	4.525307	31.559587	0.767242	11.600	85.203081	5.774236	6
2002	7.432434	30.939581	0.577957	10.200	13.999891	8.284738	5
2003	2.933218	32.014910	0.518321	10.200	2.980690	0.246284	3
2004	4.899840	32.117074	0.396331	9.800	6.308320	1.060099	4
2005	3.923677	32.163065	0.211998	8.900	6.989059	4.441178	4
2006	5.176154	32.700688	0.525200	9.200	12.512961	5.152136	7
2007	5.463396	32.579531	0.505234	10.000	10.819965	5.298003	6
2008	2.829223	33.018504	0.759317	9.400	11.956204	2.152043	5
2009	0.707510	28.465658	0.514683	9.000	9.885949	1.271568	5
2010	6.496794	32.022875	0.498225	9.400	5.978876	4.258940	3
2011	3.681689	32.958833	0.768972	9.400	5.475862	2.746709	2
2012	2.292398	31.001229	0.525714	9.600	4.806465	2.246066	3
2013	2.896205	29.102217	0.455219	8.600	4.638891	2.191130	3
2014	3.341448	29.276910	0.628150	8.600	8.144493	2.042800	3
2015	2.790236	28.918812	0.527288	8.600	8.190748	2.385188	2
2016	2.929305	29.252387	0.451318	7.900	7.123156	2.981150	4

```
column_names = {'Unemployment, total (% of total labor force) (national estimate)':'un
                'GDP growth (annual %)':'gdp_growth',
                'Gross capital formation (% of GDP)':'gross_capital_formation',
                'Population growth (annual %)':'pop_growth',
                'Birth rate, crude (per 1,000 people)':'birth_rate',
                'Broad money growth (annual %)':'broad_money_growth',
                'Final consumption expenditure (% of GDP)':'final_consum_gdp',
                'Final consumption expenditure (annual % growth)':'final_consum_growth
                'General government final consumption expenditure (annual % growth)':'
                'Gross capital formation (annual % growth)':'gross_cap_form_growth',
                'Households and NPISHs Final consumption expenditure (annual % growth)
```

In [13]:

```
econ_df = econ_df.rename(columns = column_names)
econ_df.head()
```

Out[13]:

	gdp_growth	gross_capital_formation	pop_growth	birth_rate	broad_money_
Year					
1969	14.541235	29.943577	2.263434	30.663	(
1970	9.997407	26.338200	2.184174	31.200	:
1971	10.454693	25.558501	1.971324	31.200	:
1972	7.150715	21.404761	1.875999	28.400	:
1973	14.827554	25.872858	1.768293	28.300	:

In [14]:

```
econ_df.isnull().any()
```

Out[14]:

```
gdp_growth                 False
gross_capital_formation    False
pop_growth                 False
birth_rate                 False
broad_money_growth         False
final_consum_growth        False
gov_final_consum_growth    False
gross_cap_form_growth      False
hh_consum_growth           False
unemployment               False
dtype: bool
```

(4) Checking for Perfect Multicollinearity

Multicollinearity is where one of the explanatory variables is highly correlated with another explanatory variable. In essence, one of the X variables is almost perfectly correlated with another or multiple X variables.

What is the problem with multicollinearity?

The problem with multicollinearity, from a math perspective, is that the coefficient estimates themselves tend to be unreliable. Additionally, the standard errors of slope coefficients become artificially inflated. **Because the standard error is used to help calculate the p-value, this leads to a higher probability that we will incorrectly conclude that a variable is not statistically significant.**

In [15]:

```
corr = econ_df.corr()
corr
```

Out[15]:

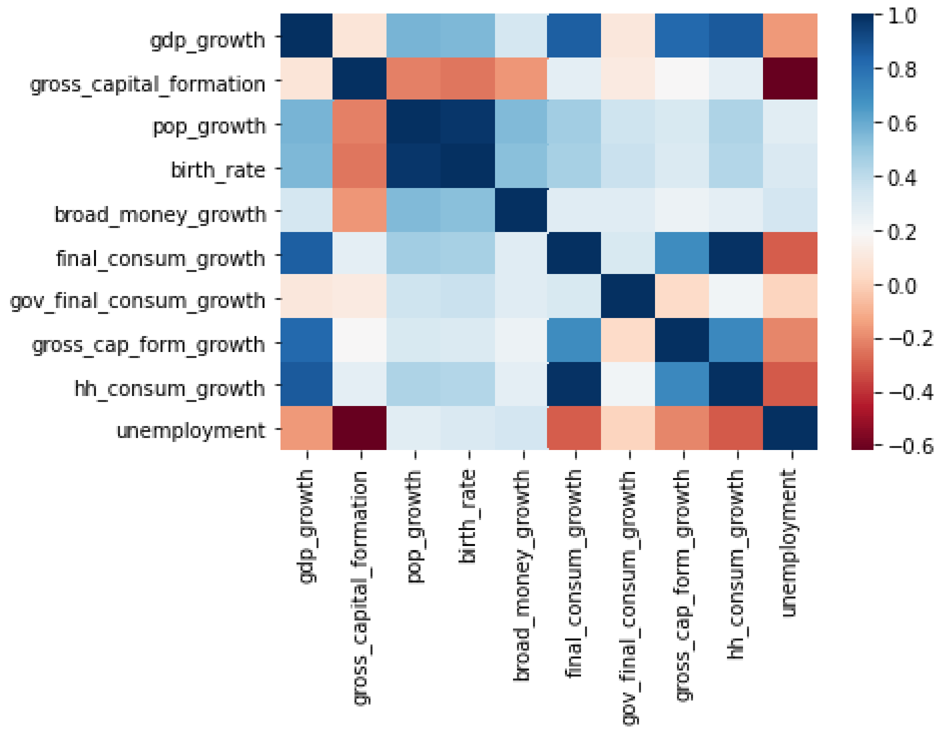
	gdp_growth	gross_capital_formation	pop_growth	birth_rate
gdp_growth	1.000000	0.086712	0.567216	0.5532
gross_capital_formation	0.086712	1.000000	-0.215243	-0.2416
pop_growth	0.567216	-0.215243	1.000000	0.9787
birth_rate	0.553225	-0.241668	0.978754	1.0000
broad_money_growth	0.335249	-0.163803	0.548336	0.5305
final_consum_growth	0.855835	0.266617	0.470449	0.4583
gov_final_consum_growth	0.098183	0.118075	0.357042	0.3705
gross_cap_form_growth	0.825496	0.187885	0.317556	0.3052
hh_consum_growth	0.868848	0.268592	0.442187	0.4282
unemployment	-0.160783	-0.618524	0.279486	0.3137

In [16]:

```
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, cmap='RdBu')
```

Out[16]:

<AxesSubplot:>



Looking at the heatmap along with the correlation matrix we can identify a few highly correlated variables. For example, if you look at the correlation between `birth_rate` and `pop_growth` it ends up at almost .98. This is an extremely high correlation and marks it as a candidate to be removed. Logically it makes sense that these two are highly correlated; if you're having more babies, then the population should be increasing.

However, we should be more systematic in our approach to removing highly correlated variables. One method we can use is the `variance_inflation_factor` which **is a measure of how much a particular variable is contributing to the standard error in the regression model. When significant multicollinearity exists, the variance inflation factor will be huge for the variables in the calculation.**

A general recommendation is that if any of our variables come back with a **value of 5 or higher, then they should be removed from the model.**

In [17]:

```
econ_df_before = econ_df
econ_df_after = econ_df.drop(['gdp_growth','birth_rate', 'final_consum_growth','gross_
```

In [18]:

```
# VIF does expect a constant term in the data, so we need to add one using the add_con
X1 = sm.tools.add_constant(econ_df_before)
X2 = sm.tools.add_constant(econ_df_after)
```

In [19]:

```
series_before = pd.Series([variance_inflation_factor(X1.values, i) for i in range(X1.s
series_after = pd.Series([variance_inflation_factor(X2.values, i) for i in range(X2.sh
```

In [20]:

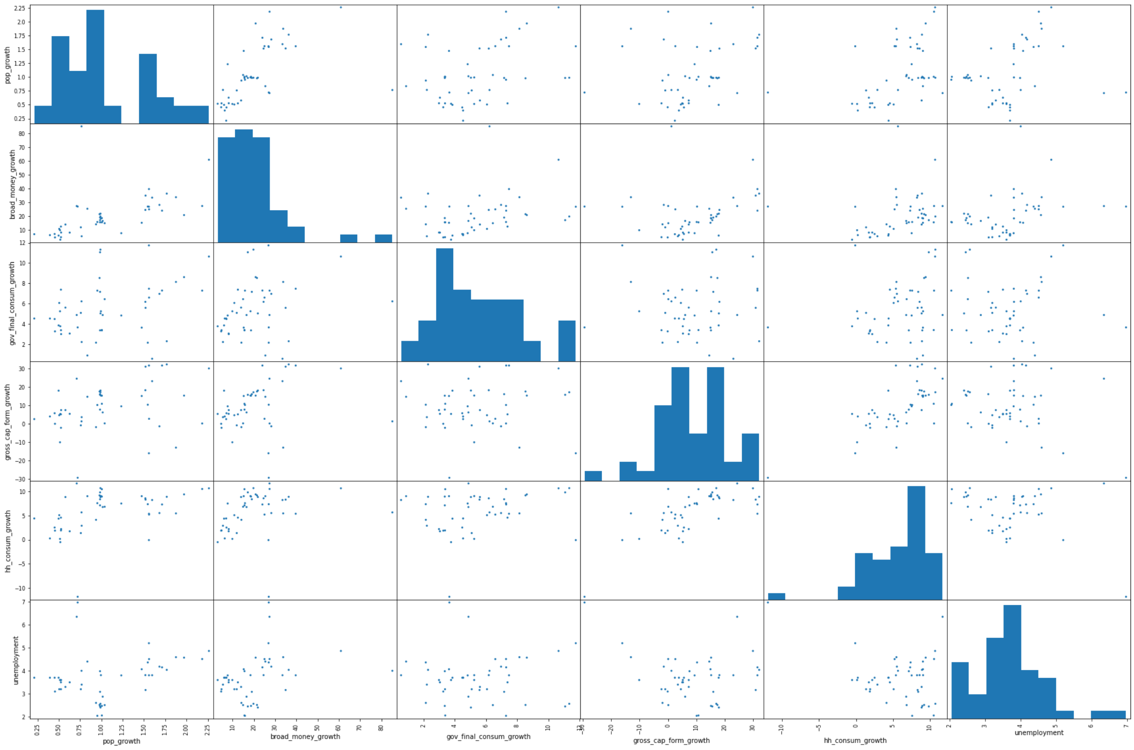
```
print('-'*100)
print('DATA BEFORE')
print('-'*100)
display(series_before)
print('-'*100)

print('DATA AFTER')
print('-'*100)
display(series_after)
```

```
--------------------------------------------------------------------------
--------------------------
DATA BEFORE
--------------------------------------------------------------------------
--------------------------

const                          314.550195
gdp_growth                       9.807879
gross_capital_formation          2.430057
pop_growth                      25.759263
birth_rate                      26.174368
broad_money_growth               1.633079
final_consum_growth           2305.724583
gov_final_consum_growth         32.527332
gross_cap_form_growth            3.796420
hh_consum_growth              2129.093634
unemployment                     2.800008
dtype: float64

--------------------------------------------------------------------------
--------------------------
DATA AFTER
--------------------------------------------------------------------------
--------------------------

const                        27.891150
pop_growth                    1.971299
broad_money_growth            1.604644
gov_final_consum_growth       1.232229
gross_cap_form_growth         2.142992
hh_consum_growth              2.782698
unemployment                  1.588410
dtype: float64
```

Looking at the data above we now get some confirmation about our suspicion. It makes sense to remove either `birth_rate` or `pop_growth` and some of the consumption growth metrics. Once we remove those metrics and recalculate the VIF, we get a passing grade and can move forward.

In [21]:

```
pd.plotting.scatter_matrix(econ_df_after, alpha = 1, figsize = (30, 20))
plt.show()
```



(5) Dealing with Outliers in the Data

In [22]:

```
desc_df = econ_df.describe()
desc_df
```

Out[22]:

	gdp_growth	gross_capital_formation	pop_growth	birth_rate	broad_money_g
count	48.000000	48.000000	48.000000	48.000000	
mean	7.280315	32.433236	1.058072	16.340896	
std	4.209306	4.136932	0.514039	6.814683	
min	-5.471219	21.404761	0.211998	7.900000	
25%	4.374899	29.776910	0.615602	9.950000	
50%	7.513471	32.335229	0.985132	15.150000	
75%	10.376191	34.474874	1.525765	21.750000	
max	14.827554	41.374062	2.263434	31.200000	

In [23]:

```
desc_df.loc['+3_std'] = desc_df.loc['mean'] + (desc_df.loc['std'] * 3)
desc_df.loc['-3_std'] = desc_df.loc['mean'] - (desc_df.loc['std'] * 3)

desc_df
```

Out[23]:

	gdp_growth	gross_capital_formation	pop_growth	birth_rate	broad_money_
count	48.000000	48.000000	48.000000	48.000000	4
mean	7.280315	32.433236	1.058072	16.340896	2
std	4.209306	4.136932	0.514039	6.814683	1
min	-5.471219	21.404761	0.211998	7.900000	
25%	4.374899	29.776910	0.615602	9.950000	1
50%	7.513471	32.335229	0.985132	15.150000	1
75%	10.376191	34.474874	1.525765	21.750000	2
max	14.827554	41.374062	2.263434	31.200000	8
+3_std	19.908232	44.844034	2.600188	36.784945	6
-3_std	-5.347602	20.022439	-0.484044	-4.103153	-2

We have only 50 observations, but 6 (minus the 3 we dropped) exploratory variables. **Generally we should aim for at least 20 instances for each variable.**

Looking at the data frame up above, a few values are standing out, for example, the maximum value in the `broad_money_growth` column is almost four standard deviations above the mean. Such an enormous value would qualify as an outlier.

Should we drop the outliers?

Generally, if we believe the data has been entered in error, we should remove it. However, in this situation, the values that are being identified as outliers are correct values and are not errors. Both of these values were produced during specific moments in time. The one in 1998 was right after the Asian Financial Crisis, and the one in 2001 is right after the DotCom Bubble, so it's entirely conceivable that these values were produced in extreme albeit rare conditions. **For this reason, I will NOT be removing these values from the dataset as they recognize actual values that took place.**

(6) Building the Model

Split the Data

In [24]:

```
econ_df_after = econ_df.drop(['birth_rate', 'final_consum_growth','gross_capital_forma
```

In [25]:

```
X = econ_df_after.drop('gdp_growth', axis = 1)
Y = econ_df_after[['gdp_growth']]
```

In [26]:

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state
```

Create and Fit the Model

In [27]:

```
regression_model = LinearRegression()
regression_model.fit(X_train, y_train)
```

Out[27]:

```
LinearRegression()
```

Explore the Output

In [28]:

```
intercept = regression_model.intercept_[0]
coefficent = regression_model.coef_[0][0]
```

In [29]:

```
print("The intercept for our model is {:.4}".format(intercept))
print('-'*60)

for coef in zip(X.columns, regression_model.coef_[0]):
    print("The coefficient for {} is {:.2}".format(coef[0],coef[1]))
```

```
The intercept for our model is 2.08
------------------------------------------------------------
The coefficient for pop_growth is 2.0
The coefficient for broad_money_growth is -0.0017
The coefficient for gov_final_consum_growth is -0.21
The coefficient for gross_cap_form_growth is 0.14
The coefficient for hh_consum_growth is 0.51
The coefficient for unemployment is 0.027
```

Making multiple predictions at once

In [30]:

```
y_predict = regression_model.predict(X_test)
y_predict[:5]
```

Out[30]:

```
array([[ 7.61317534],
       [ 6.31344066],
       [ 5.06818662],
       [ 4.19869856],
       [11.11885324]])
```

(7) Checking for Heteroscedasticity

Heteroscedasticity merely means the standard errors of a variable, monitored over a specific amount of time, are non-constant.

What is the problem with heteroscedasticity?

1. While heteroscedasticity does not cause bias in the coefficient estimates, **it causes the coefficient estimates to be less precise.** The lower precision increases the likelihood that the coefficient estimates are further from the correct population value.
2. **Heteroscedasticity tends to produce p-values that are smaller than they should be.** This effect occurs because heteroscedasticity increases the variance of the coefficient estimates, but the OLS procedure does not detect this increase. Consequently, OLS calculates the t-values and F-values using an underestimated amount of variance. This problem can lead you to conclude that a model term is statistically significant when it is not significant.

In [31]:

```
X2 = sm.add_constant(X)
model = sm.OLS(Y, X2)
est = model.fit()
```

How to test for heteroscedasticity?

The Breusch-Pagan is a more general test for heteroscedasticity while the White test is a unique case.

- The null hypothesis for both the White's test and the Breusch-Pagan test is that the variances for the errors are equal:
  - **H0 = σ2i = σ2**
- The alternate hypothesis (the one you're testing), is that the variances are not equal:
  - **H1 = σ2i ≠ σ2**

Our goal is to fail to reject the null hypothesis, have a high p-value because that means we have no heteroscedasticity.

In [34]:

```
# Run the White's test
diag.het_white(est.resid, est.model.exog)
```

Out[34]:

```
(27.564940591974057,
 0.43365711028668635,
 0.9991884097265678,
 0.5090811918586821)
```

In [35]:

```
_, pval, __, f_pval = diag.het_white(est.resid, est.model.exog)
print(pval, f_pval)
```

```
0.43365711028668635 0.5090811918586821
```

In [36]:

```
if pval > 0.05:
    print("For the White's Test")
    print("The p-value was {:.4}".format(pval))
    print("We fail to reject the null hypothesis, so there is no heterosecdasticity. \

else:
    print("For the White's Test")
    print("The p-value was {:.4}".format(pval))
    print("We reject the null hypothesis, so there is heterosecdasticity. \n")
```

```
For the White's Test
The p-value was 0.4337
We fail to reject the null hypothesis, so there is no heterosecdasticity.
```

In [37]:

```
# Run the Breusch-Pagan test
diag.het_breuschpagan(est.resid, est.model.exog)
```

Out[37]:

```
(7.816776336764519,
 0.25183646701202067,
 1.3292770821195823,
 0.2662794557854064)
```

In [38]:

```
_, pval, __, f_pval = diag.het_breuschpagan(est.resid, est.model.exog)
print(pval, f_pval)
```

```
0.25183646701202067 0.2662794557854064
```

In [39]:

```
if pval > 0.05:
    print("For the Breusch-Pagan's Test")
    print("The p-value was {:.4}".format(pval))
    print("We fail to reject the null hypothesis, so there is no heterosecdasticity.")

else:
    print("For the Breusch-Pagan's Test")
    print("The p-value was {:.4}".format(pval))
    print("We reject the null hypothesis, so there is heterosecdasticity.")
```

```
For the Breusch-Pagan's Test
The p-value was 0.2518
We fail to reject the null hypothesis, so there is no heterosecdasticity.
```

(8) Checking for Autocorrelation

Autocorrelation is a characteristic of data in which the correlation between the values of the same variables is based on related objects. It violates the assumption of instance independence, which underlies most of conventional models.

When you have a series of numbers, and there is a pattern such that values in the series can be predicted based on preceding values in the series, the set of numbers is said to exhibit autocorrelation. This is also known as serial correlation and serial dependence. It generally exists in those types of data-sets in which the data, instead of being randomly selected, are from the same source.

What is the problem with autocorrelation?

The existence of autocorrelation means that computed standard errors, and consequently p-values, are misleading. Autocorrelation in the residuals of a model is also a sign that the model may be unsound. A workaround is we can compute more robust standard errors.

How to test for autocorrelation?

Use the Ljung-Box test for no autocorrelation of residuals.

- **H0: The data are random.**
- **Ha: The data are not random.**

That means we want to fail to reject the null hypothesis, have a large p-value because then it means we have no autocorrelation. To use the Ljung-Box test, we will call the `acorr_ljungbox` function, pass through the `est.resid` and then define the lags.

The lags can either be calculated by the function itself, or we can calculate them. If the function handles it the max lag will be `min((num_obs // 2 - 2), 40)` , however, there is a rule of thumb that for non-seasonal time series the lag is `min(10, (num_obs // 5))` .

In [40]:

```
# calculate the lag, optional
lag = min(10, (len(X)//5))
print('The number of lags will be {}'.format(lag))
```

```
The number of lags will be 9
```

In [41]:

```
# run the Ljung-Box test for no autocorrelation of residuals
test_results = diag.acorr_ljungbox(est.resid, lags = lag)
test_results
```

```
C:\Users\SHBHAM\anaconda3\lib\site-packages\statsmodels\stats\diagnostic.
py:559: FutureWarning: The value returned will change to a single DataFra
me after 0.12 is released.  Set return_df to True to use to return a Data
Frame now.  Set return_df to False to silence this warning.
  warnings.warn(msg, FutureWarning)
```

Out[41]:

```
(array([1.97846067, 2.05840348, 2.05889136, 2.61809174, 3.60559768,
        4.3098546 , 5.32394408, 7.2885471 , 8.52206017]),
 array([0.15955267, 0.35729206, 0.56027398, 0.62362162, 0.60747302,
        0.63482279, 0.62049549, 0.50584467, 0.48250733]))
```

In [42]:

```
ibvalue, p_val = test_results
```

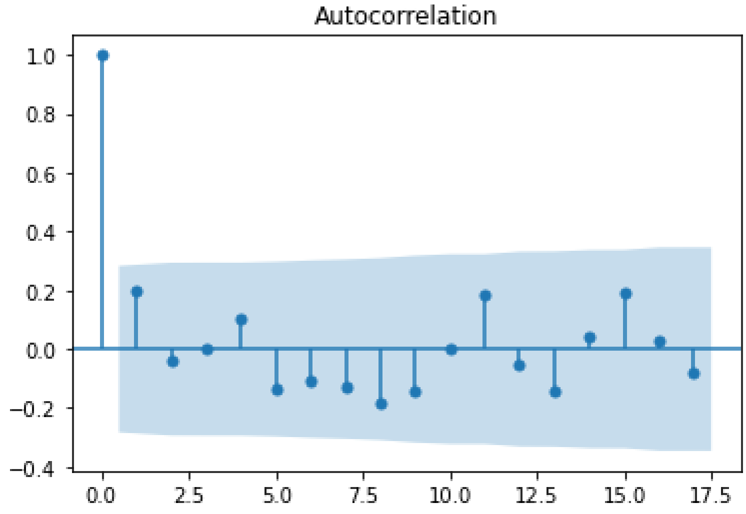
In [43]:

```
if min(p_val) > 0.05:
    print("The lowest p-value found was {:.4}".format(min(p_val)))
    print("We fail to reject the null hypothesis, so there is no autocorrelation.")

else:
    print("The lowest p-value found was {:.4}".format(min(p_val)))
    print("We reject the null hypothesis, so there is autocorrelation.")
```

```
The lowest p-value found was 0.1596
We fail to reject the null hypothesis, so there is no autocorrelation.
```

In [44]:

```
# plot autocorrelation
sm.graphics.tsa.plot_acf(est.resid)
plt.show()
```

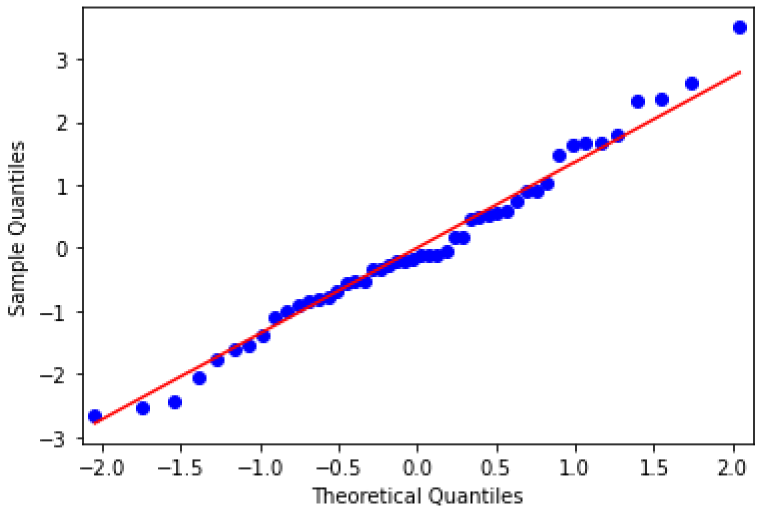


(9) Checking for Normality of Residuals

This will require using a QQ pplot which help us assess if a set of data plausibly came from some theoretical distribution such as a Normal or exponential. Visually what we are looking for is **the data hugs the line tightly**; this would give us confidence in our assumption that the residuals are normally distributed.

In [45]:

```
sm.qqplot(est.resid, line='s')
pylab.show()
```



Checking the Mean of the Residuals

If the mean is very close to zero, then we are good to proceed. It's not uncommon to get a mean that isn't exactly zero; this is because of rounding errors. However, if it's very close to zero, it's ok.

In [46]:

```
mean_residuals = sum(est.resid)/ len(est.resid)
print("The mean of the residuals is {:.4}".format(mean_residuals))
```

```
The mean of the residuals is -1.299e-14
```

(10) Examining Model Fit

- **Mean Absolute Error (MAE):** Is the mean of the absolute value of the errors. This metric gives an idea of magnitude but no idea of direction (too high or too low).
- **Mean Squared Error (MSE):** Is the mean of the squared errors.MSE is more popular than MAE because MSE "punishes" more significant errors.
- **Root Mean Squared Error (RMSE):** Is the square root of the mean of the squared errors. RMSE is even more favored because it allows us to interpret the output in y-units.

In [52]:

```
model_mse = mean_squared_error(y_test, y_predict)
model_mae = mean_absolute_error(y_test, y_predict)
model_rmse =  math.sqrt(model_mse)

print("MSE: {:.3}".format(model_mse))
print("MAE: {:.3}".format(model_mae))
print("RMSE: {:.3}".format(model_rmse))
```

```
MSE: 0.707
MAE: 0.611
RMSE: 0.841
```

R-Squared

The R-Squared metric provides us a way to measure the goodness of fit or how well our data fits the model. The higher the R-Squared metric, the better the data fit our model. However, one limitation is that R-Square increases as the number of feature increases in our model, so it does not pay to select the model with the highest R-Square.

In [49]:

```
model_r2 = r2_score(y_test, y_predict)
print("R2: {:.2}".format(model_r2))
```

```
R2: 0.86
```

Confidence Intervals

By default, confidence intervals are calculated using 95% intervals. We interpret confidence intervals by saying if the population from which this sample was drawn was sampled 100 times, approximately 95 of those confidence intervals would contain the "true" coefficient.

In [32]:

```
est.conf_int()
```

Out[32]:

	0	1
const	-0.323322	4.210608
pop_growth	0.997064	3.366766
broad_money_growth	-0.037652	0.036865
gov_final_consum_growth	-0.372408	-0.005139
gross_cap_form_growth	0.079057	0.179616
hh_consum_growth	0.325648	0.667975
unemployment	-0.570237	0.558631

Hypothesis Testing

With hypothesis testing, we are trying to determine the statistical significance of the coefficient estimates. This test is outlined as the following.

- **Null Hypothesis:** There is no relationship between the exploratory variables and the explanatory variable.
- **Alternative Hypothesis:** There is a relationship between the exploratory variables and the explanatory variable.

In [33]:

```
est.pvalues
```

Out[33]:

```
const                      9.088069e-02
pop_growth                 5.996378e-04
broad_money_growth         9.830934e-01
gov_final_consum_growth    4.419934e-02
gross_cap_form_growth      5.978663e-06
hh_consum_growth           6.801951e-07
unemployment               9.835355e-01
dtype: float64
```

Here it's a little hard to tell, but we have a few insignificant coefficients. The first is the `constant` itself, so technically this should be dropped. However, we will see that once we remove the irrelevant variables that the intercept becomes significant. **If it still wasn't significant, we could have our intercept start at 0 and assume that the cumulative effect of X on Y begins from the origin (0,0).** Along with the constant, we have `unemployment` and `broad_money_growth` both come out as insignificant.

In [53]:

```
print(est.summary())
```

```
                            OLS Regression Results                            
==============================================================================
=====
Dep. Variable:             gdp_growth   R-squared:                       
0.893
Model:                            OLS   Adj. R-squared:                  
0.878
Method:                 Least Squares   F-statistic:                     
57.17
Date:                Mon, 21 Aug 2023   Prob (F-statistic):           2.3
6e-18
Time:                        16:18:39   Log-Likelihood:                -8
2.903
No. Observations:                  48   AIC:                             
179.8
Df Residuals:                      41   BIC:                             
192.9
Df Model:                           6                                         
Covariance Type:            nonrobust                                         
==============================================================================
==================
                             coef    std err          t      P>|t|      
[0.025      0.975]
------------------------------------------------------------------------------
------------------
const                      1.9436      1.123      1.732      0.091      
-0.323       4.211
pop_growth                 2.1819      0.587      3.719      0.001      
0.997       3.367
broad_money_growth        -0.0004      0.018     -0.021      0.983      
-0.038       0.037
gov_final_consum_growth   -0.1888      0.091     -2.076      0.044      
-0.372      -0.005
gross_cap_form_growth      0.1293      0.025      5.195      0.000      
0.079       0.180
hh_consum_growth           0.4968      0.085      5.862      0.000      
0.326       0.668
unemployment              -0.0058      0.279     -0.021      0.984      
-0.570       0.559
==============================================================================
=====
Omnibus:                        0.820   Durbin-Watson:                   
1.589
Prob(Omnibus):                  0.664   Jarque-Bera (JB):                
0.658
Skew:                           0.281   Prob(JB):                        
0.720
Kurtosis:                       2.881   Cond. No.                        
154.
==============================================================================
=====

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

The first thing we notice is that the **p-values** from up above are now easier to read and we can now determine that the coefficients that have a p-value greater than 0.05 can be removed.

The other metric that stands out is our **Adjusted R-Squared value which is .878, lower than our R-Squared value**. This makes sense as we were probably docked for the complexity of our model. However, an R-Squared over .878 is still very strong.

The only additional metrics we will describe here is the t-value which is the coefficient divided by the standard error. **The higher the t-value, the more evidence we have to reject the null hypothesis.** Also the standard error, the standard error is the approximate standard deviation of a statistical sample population.

(11) Removing the Insignificant Variables

In [54]:

```
# define our input variable (X) & output variable
econ_df_after = econ_df.drop(['birth_rate', 'final_consum_growth','gross_capital_forma
                              'unemployment'], axis = 1)

X = econ_df_after.drop('gdp_growth', axis = 1)
Y = econ_df_after[['gdp_growth']]

# Split X and y into X_
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state

# create a Linear Regression model object
regression_model = LinearRegression()

# pass through the X_train & y_train data set
regression_model.fit(X_train, y_train)
```

Out[54]:

```
LinearRegression()
```

In [55]:

```
# define our intput
X2 = sm.add_constant(X)

# create a OLS model
model = sm.OLS(Y, X2)

# fit the data
est = model.fit()

print(est.summary())
```

```
                            OLS Regression Results
==========================================================================
=====
Dep. Variable:             gdp_growth   R-squared:
0.893
Model:                            OLS   Adj. R-squared:
0.883
Method:                 Least Squares   F-statistic:
89.94
Date:                Mon, 21 Aug 2023   Prob (F-statistic):           2.6
1e-20
Time:                        16:52:51   Log-Likelihood:                -8
2.904
No. Observations:                  48   AIC:
175.8
Df Residuals:                      43   BIC:
185.2
Df Model:                           4
Covariance Type:            nonrobust
==========================================================================
==================
                              coef    std err          t      P>|t|
[0.025      0.975]
--------------------------------------------------------------------------
------------------
const                       1.9229      0.573      3.356      0.002
0.767       3.078
pop_growth                  2.1704      0.477      4.546      0.000
1.208       3.133
gov_final_consum_growth    -0.1889      0.087     -2.162      0.036
-0.365      -0.013
gross_cap_form_growth       0.1293      0.024      5.346      0.000
0.081       0.178
hh_consum_growth            0.4976      0.076      6.526      0.000
0.344       0.651
==========================================================================
=====
Omnibus:                        0.831   Durbin-Watson:
1.589
Prob(Omnibus):                  0.660   Jarque-Bera (JB):
0.666
Skew:                           0.282   Prob(JB):
0.717
Kurtosis:                       2.882   Cond. No.
51.9
==========================================================================
=====

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

Looking at the output, we now see that all of the independent variables are significant and even our constant is significant. We could rerun our test for autocorrelation and, but the tests will take us to the same conclusions we found above.

Looking at the coefficents, we would say `pop_growth`, `gross_cap_form_growth`, and `hh_consum_growth` all have a positive effect on GDP growth. Additionally, we would say that

(12) Saving the Model for Future

In [56]:

```
import pickle

# pickle the model
with open('my_mulitlinear_regression.sav','wb') as f:
    pickle.dump(regression_model, f)

# load it back in
with open('my_mulitlinear_regression.sav', 'rb') as pickle_file:
    regression_model_2 = pickle.load(pickle_file)

# make a new prediction
regression_model_2.predict([X_test.loc[2002]])
```

Out[56]:

```
array([[7.6042968]])
```