Python Project

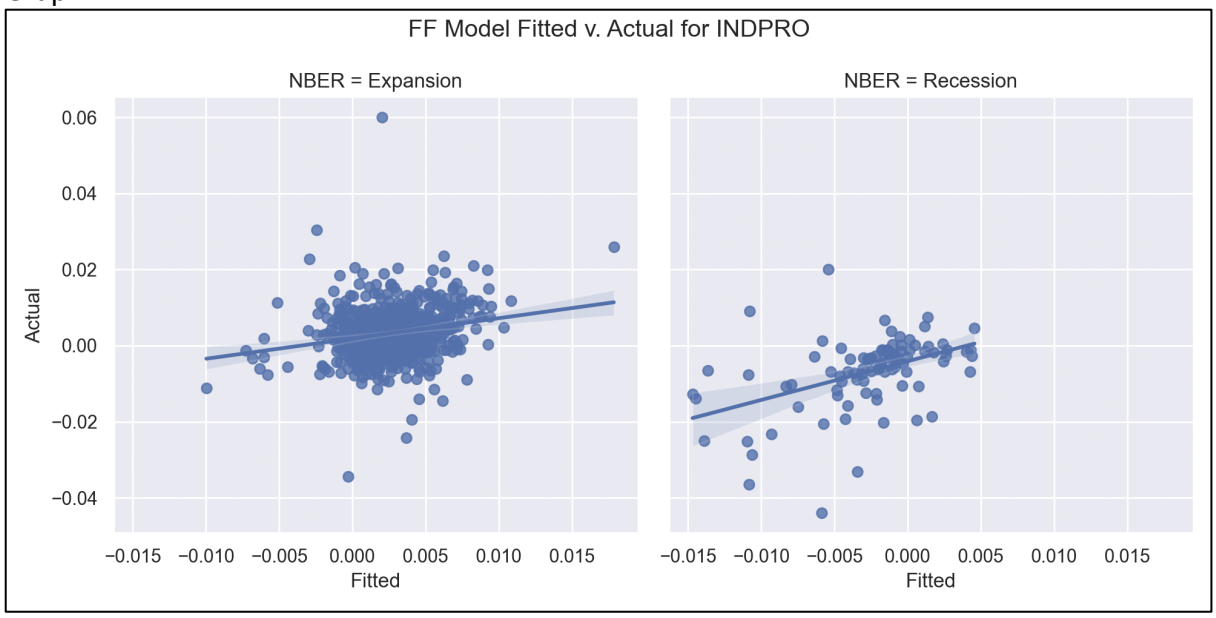
Name : Kota Kobayashi

a. Analytical Results -

First, this section will discuss analytical results. The graph 1-5 showed scatter plots of the transformed variable and its fitted value for five variables, evaluating whether the model gives an appropriate fit for the data and whether it is suitable in recessionary and/ or expansionary periods.

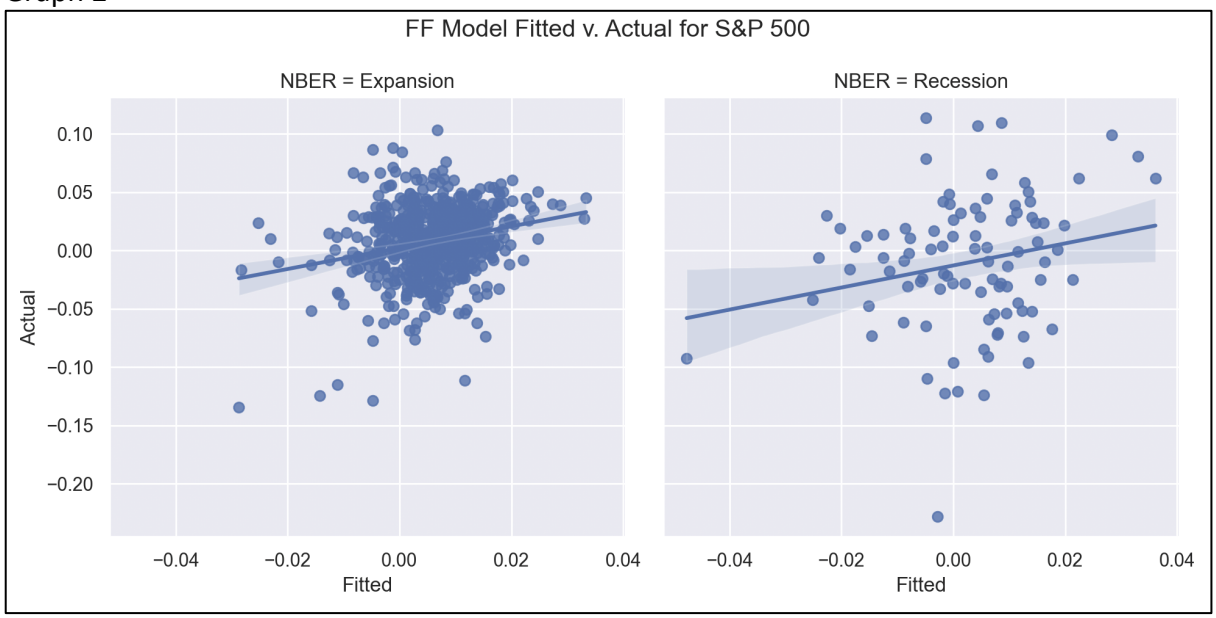
In graph 1, they roughly correlated positively in both expansion and recession data. However, the plots are not perfectly diagonal and it is not close to a perfect correlation coefficient of 1. Therefore, both data showed a roughly suitable fit for data.

Graph 1



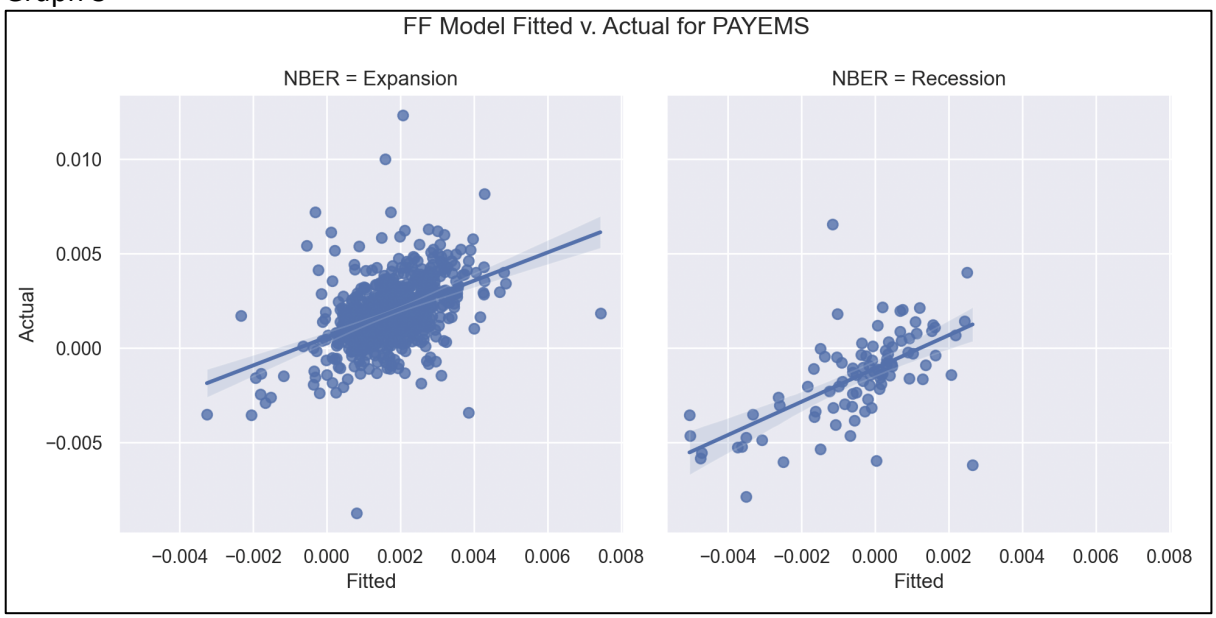
In graph 2, overall, variation is greater than in Graph 1. While showing a positive correlation, the correlation coefficient is not very large. In particular, the graph of recession shows a large discrepancy between actual values and fitted values.

Graph 2



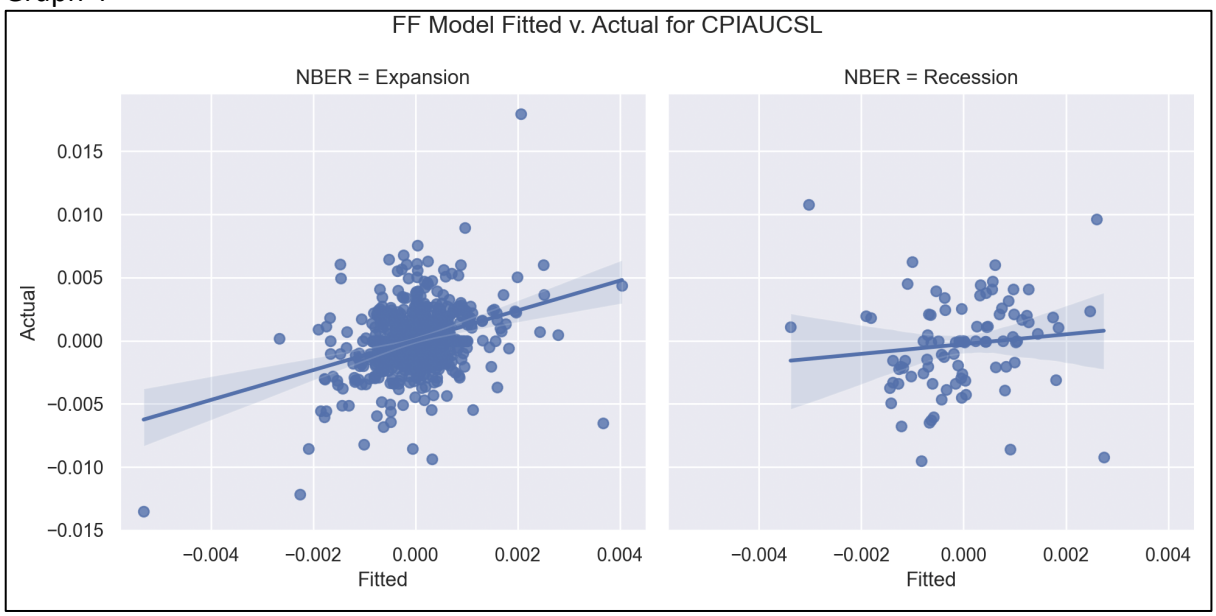
In graph 3, although there are some plots that differ significantly from the Actual values, they are more strongly positively correlated than Graph 2. Expansion and recession both showed the same degree of appropriate fit for data.

Graph 3



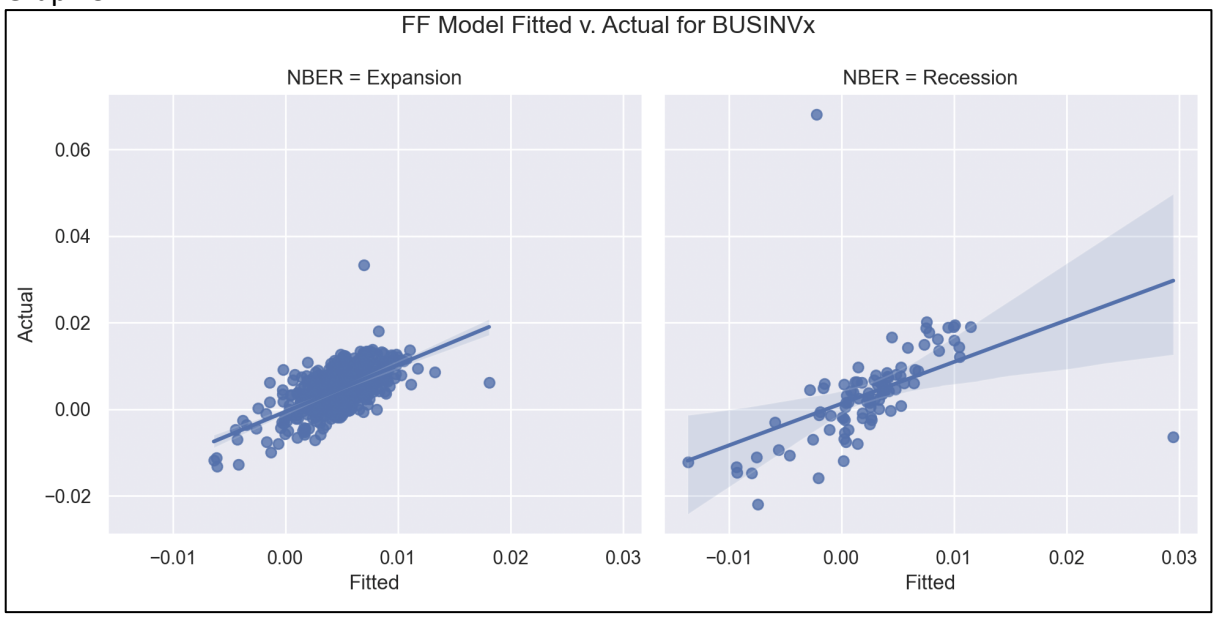
In graph 4, overall, both are not appropriate because there is a lot of variation, and a relatively large number of plots have fitted values that are far from the actual values although they are positively correlated. Especially, in the recession graph, R^2 is much lower than others.

Graph 4



In graph 5, this 'BUSINVx' variable is most strongly positively correlated in five variables. Especially, the model in the expansion provided a more reasonable fit for the data because R^2 and correlation coefficient are higher than others.

Graph 5



b. Development Process –

This section will discuss the development process. First, in Figure 1, libraries which were used in this analysis are imported and Principal Components Analysis (PCA) was defined below.

Figure 1

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import datetime as dt
import statsmodels.api as sma

#the pca function is written for you, call this from your code to calculate the 1st PC
def pca_function(stdata):
    """Returns the sign identified 1st principal component of a data set.
    input: stdata - a n x t pandas data frame
    output: 1st principal component, standardised to s.d = 1 and
    signed to have the same sign as the cross sectional mean of the variables"""
    factor1_us = sma.PCA(stdata, 1).factors
    factor1 = (factor1_us - factor1_us.mean()) / factor1_us.std()
    sgn = np.sign(pd.concat([stdata.mean(1), factor1], axis=1).corr().iloc[1, 0])
    return factor1 * sgn

#produce your analysis for the following five variables
my_srs = ['INDPRO', 'S&P 500', 'PAYEMS', 'CPIAUCSL', 'BUSINVx']

#enter your code below, save this file as 'final_exam.py', Zip and upload as instructed
```

In addition, Figure 2 showed from questions 1 to 3. In question 1, '2021-12.csv' file was loaded, and Nan values were dropped. After unnecessary data was dropped, the index was converted to datetime (See https://pandas.pydata.org/docs/reference/api/pandas.to_datetime.html). In question 2, the time period of the data was narrowed down to December 2019 and a new subset data was created. In question 3, ' fred_md_desc.csv' file was loaded and Nan values were dropped. Moreover, the columns' names were matched to the columns of '2021-12.csv' file.

Figure 2

```
# Question 1
data_original = pd.read_csv('2021-12.csv',index_col=0)
data_original.dropna(how='all', inplace=True)
data_original = data_original.drop('Transform:')
data_original.index = pd.to_datetime(data_original.index)

# Question 2
data = data_original[data_original.index <='12/01/2019']

# Question 3
desc = pd.read_csv('fred_md_desc.csv').dropna(how='all', axis=0)
decs = desc.reset_index(drop = True)
desc = decs.set_index('fred')
desc = desc[desc.index.isin(data.columns) == True]
```

In question 4 and 5, we used the subset created in question 3. The data was mathematically transferred based on 'tfcode'. We used for-loop and if-statement to classify and replace each data in a different way in Figure 3. Figure 4 showed 'tfcode' and how to transfer the subset data. In accordance with this 'tfcode', we used if-statement.

Figure 3

```
# Question 4 & 5
tfdata = data.copy()
for srs in tfdata.columns:

    if srs == 'VIXCLSx':
        continue

    elif desc.loc[srs, 'tcode'] == 2.0:
        tfdata[srs] = tfdata[srs].diff()

    elif desc.loc[srs, 'tcode'] == 3.0:
        tfdata[srs] = tfdata[srs].diff().diff()

    elif desc.loc[srs, 'tcode'] == 4.0:
        tfdata[srs] = np.log(tfdata[srs])

    elif desc.loc[srs, 'tcode'] == 5.0:
        tfdata[srs] = np.log(tfdata[srs]).diff()

    elif desc.loc[srs, 'tcode'] == 6.0:
        tfdata[srs] = np.log(tfdata[srs]).diff().diff()

    elif desc.loc[srs, 'tcode'] == 7.0:
        tfdata[srs] = (tfdata[srs])/(tfdata[srs].shift(1)-1)

    else:
        tfdata[srs] = tfdata[srs]
```

Figure 4

Tfcode	Description	Pandas Code
1	No transformation	`srs`
2	1st differences	`srs.diff()`
3	2nd differences	`srs.diff().diff()`
4	Log	`np.log(srs)`
5	Log 1st differences	`np.log(srs).diff()`
6	Log 2nd differences	`np.log(srs).diff().diff()`
7	Percent Change	`(srs/srs.shift(1) - 1)`

In question 6, we standardize each transformed variable Standardization is a technique for optimization methods, rescaling feature values with a distribution value between 0 and 1. The equation is below:

$$x_{stand} = \frac{x - mean(x)}{standard\ deviation\ (x)}$$

Based on this equation, the data was standardized in the following coding. Furthermore, there are some missing values in the data, and the values are filled with 0. After that, the file was saved as 'transformed_data.csv' by using the command 'DataFrame.to_csv' (see https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html)

In question 7, the data standardized in question 6 was passed to 'pca_function()' to get the principal components in the Principal component analysis (PCA) model.

Figure 5

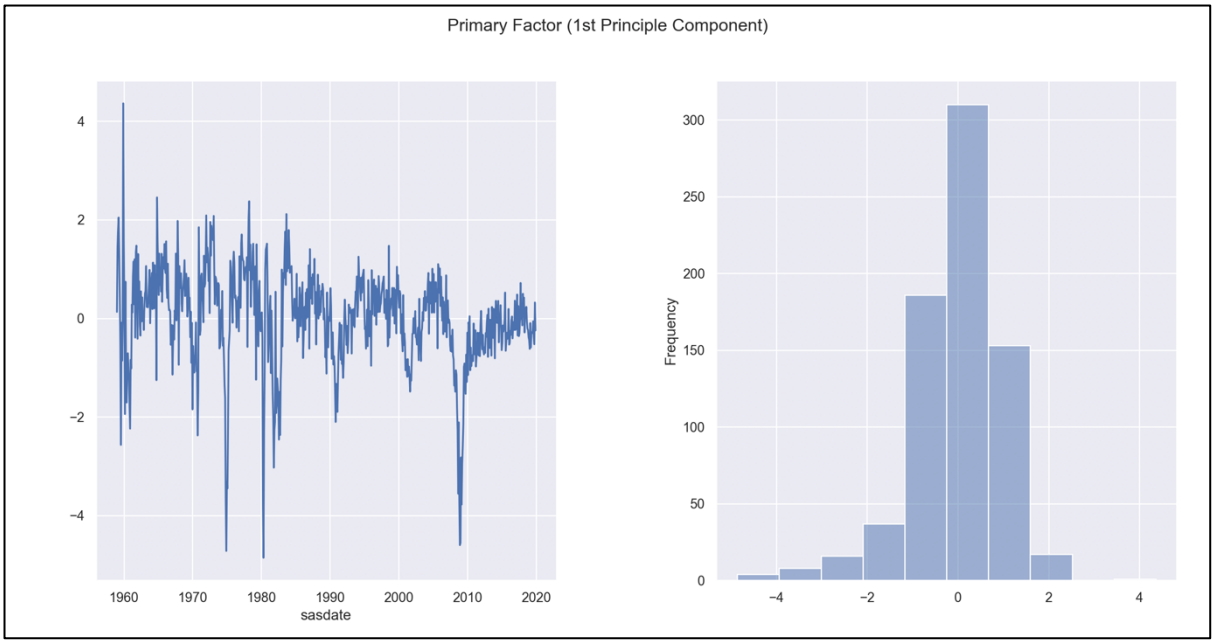
```
# Question 6
stdata = tfdata.copy()
stdata = (stdata - stdata.mean()) / stdata.std()
stdata = stdata.fillna(0)
stdata.to_csv('transformed_data.csv',index=True, header=True)

# Question 7
factor1 = pca_function(stdata)
```

In question 8, Graph 6 showed a plot of the time series of the factor and a histogram of the distribution of the factor. We use the command 'sns.lineplot()' (see https://seaborn.pydata.org/generated/seaborn.lineplot.html)to produce the plot graph and 'sns.histplot()'(see https://seaborn.pydata.org/generated/seaborn.histplot.html) to create the histgraph. These commands are from the seaborn library. The plt.savefig() was used to save the graph as 'factor.pdf'.

Figure 6

```
# Question 8
sns.set_theme()
fig, ax = plt.subplots(nrows=1, ncols=2)
sns.lineplot(data=factor1, ax=ax[0],legend=False)
sns.histplot(data=factor1, ax=ax[1],legend=False, bins=10)
plt.ylabel('Frequency')
plt.suptitle('Primary Factor (1st Principle Component)')
plt.tight_layout()
plt.savefig('factor.pdf')
```

Graph 6



In question 9, we created a new dataframe of 1st lags of the data by shifting transformed data by using 'df.shift()'. In question 10, we also produced a new dataframe by shifting the factor created in question 7. Furthermore, we prepare a new dataframe 'fitted_values', which is empty before it is regressed. To analyze the following five variables ['INDPRO', 'S&P 500','PAYEMS', 'CPIAUCSL', 'BUSINVx'] , the five variables were assigned the columns in the new dataframe. In the for loop, we treated five variables as dependent variables and the new dataframe of 1st lags of the data as an independent variable. Moreover, the factor at time t-1 was treated as an independent variable. By using sma.OLS(), the OLS model was regressed and the results were assigned to the variable 'Summary' by using '.summary()'. From Figure 8 to 12, the summaries were shown. In addition, we were able to capture the fitted value by using '.fittedvalues'. (see https://www.statsmodels.org/dev/generated/statsmodels.regression.linear_model.OLS.html)

Figure 7

```
# Question 9
lag1 = tfdata.shift(1)

# Question 10
factor1_lag1 = factor1.shift(1)
run_dates = lag1[my_srs].dropna().index
fitted_values = pd.DataFrame(index=run_dates,columns=tfdata.columns)
fitted_values = fitted_values[['INDPRO', 'S&P 500', 'PAYEMS', 'CPIAUCSL', 'BUSINVx']]

for srs in my_srs:
    reg_data = pd.concat([tfdata[srs],
                          lag1[srs],
                          factor1_lag1],
                         axis=1).set_axis(['srs','ar1','factor1'],axis=1).dropna()
    reg_model = sma.OLS(reg_data['srs'],sma.add_constant(reg_data[['ar1', 'factor1']])).fit()
    summary = reg_model.summary()
    plt.figure(figsize=(10,5))
    plt.rc(srs)
    plt.text(0.15, 0.03, str(summary), {'fontsize': 10}, fontproperties = 'monospace')
    plt.axis('off')
    plt.tight_layout()
    plt.suptitle('Dep. Variable : ' +srs)
    plt.show( block=False )
    fitted_values[srs]=reg_model.fittedvalues
```

Figure 8

Dep. Variable : INDPRO

OLS Regression Results

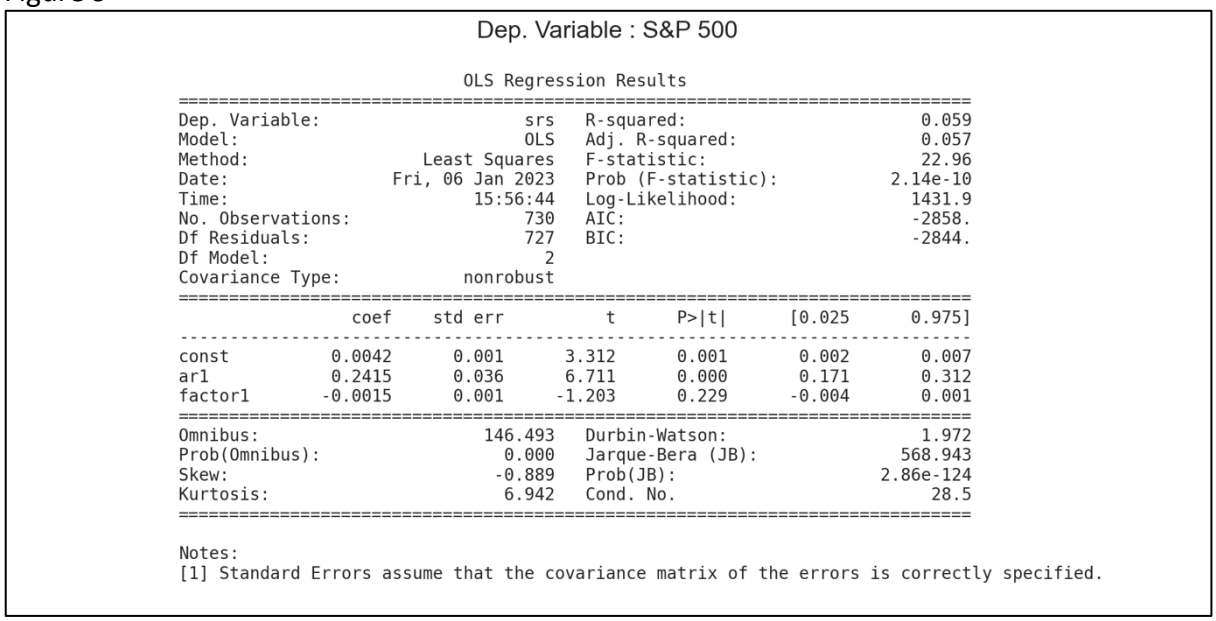
Dep. Variable:	srs	R-squared:	0.190
Model:	OLS	Adj. R-squared:	0.187
Method:	Least Squares	F-statistic:	85.08
Date:	Fri, 06 Jan 2023	Prob (F-statistic):	6.32e-34
Time:	15:56:44	Log-Likelihood:	2568.8
No. Observations:	730	AIC:	-5132.
Df Residuals:	727	BIC:	-5118.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0020	0.000	6.922	0.000	0.001	0.003
ar1	0.0223	0.057	0.392	0.695	-0.089	0.134
factor1	0.0033	0.000	7.303	0.000	0.002	0.004

Omnibus:	146.077	Durbin-Watson:	2.034
Prob(Omnibus):	0.000	Jarque-Bera (JB):	2352.413
Skew:	0.383	Prob(JB):	0.00
Kurtosis:	11.761	Cond. No.	214.

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

Figure 9



Dep. Variable : S&P 500

OLS Regression Results

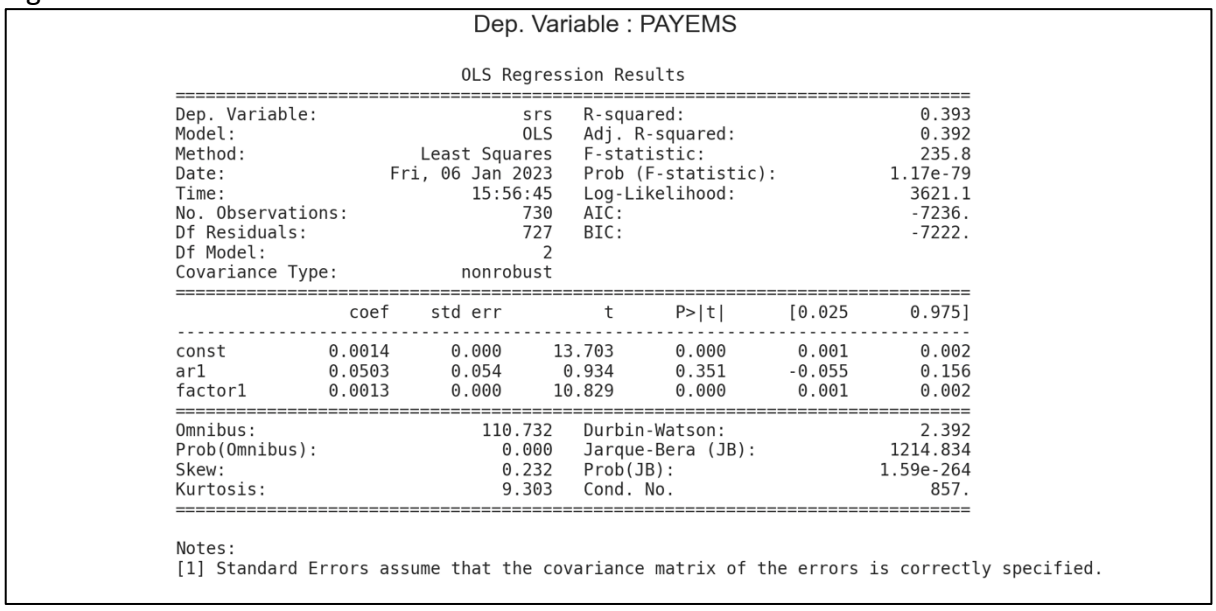
Dep. Variable:	srs	R-squared:	0.059
Model:	OLS	Adj. R-squared:	0.057
Method:	Least Squares	F-statistic:	22.96
Date:	Fri, 06 Jan 2023	Prob (F-statistic):	2.14e-10
Time:	15:56:44	Log-Likelihood:	1431.9
No. Observations:	730	AIC:	-2858.
Df Residuals:	727	BIC:	-2844.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0042	0.001	3.312	0.001	0.002	0.007
ar1	0.2415	0.036	6.711	0.000	0.171	0.312
factor1	-0.0015	0.001	-1.203	0.229	-0.004	0.001

Omnibus:	146.493	Durbin-Watson:	1.972
Prob(Omnibus):	0.000	Jarque-Bera (JB):	568.943
Skew:	-0.889	Prob(JB):	2.86e-124
Kurtosis:	6.942	Cond. No.	28.5

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

Figure 10



Dep. Variable : PAYEMS

OLS Regression Results

Dep. Variable:	srs	R-squared:	0.393
Model:	OLS	Adj. R-squared:	0.392
Method:	Least Squares	F-statistic:	235.8
Date:	Fri, 06 Jan 2023	Prob (F-statistic):	1.17e-79
Time:	15:56:45	Log-Likelihood:	3621.1
No. Observations:	730	AIC:	-7236.
Df Residuals:	727	BIC:	-7222.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0014	0.000	13.703	0.000	0.001	0.002
ar1	0.0503	0.054	0.934	0.351	-0.055	0.156
factor1	0.0013	0.000	10.829	0.000	0.001	0.002

Omnibus:	110.732	Durbin-Watson:	2.392
Prob(Omnibus):	0.000	Jarque-Bera (JB):	1214.834
Skew:	0.232	Prob(JB):	1.59e-264
Kurtosis:	9.303	Cond. No.	857.

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

Figure 11

Dep. Variable : CPIAUCSL

OLS Regression Results

Dep. Variable:	srs	R-squared:	0.088
Model:	OLS	Adj. R-squared:	0.086
Method:	Least Squares	F-statistic:	35.05
Date:	Fri, 06 Jan 2023	Prob (F-statistic):	2.93e-15
Time:	15:56:45	Log-Likelihood:	3314.2
No. Observations:	729	AIC:	-6622.
Df Residuals:	726	BIC:	-6609.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	3.721e-06	9.53e-05	0.039	0.969	-0.000	0.000
ar1	-0.2970	0.035	-8.373	0.000	-0.367	-0.227
factor1	3.921e-05	9.53e-05	0.411	0.681	-0.000	0.000

Omnibus:	88.639	Durbin-Watson:	2.137
Prob(Omnibus):	0.000	Jarque-Bera (JB):	715.764
Skew:	0.156	Prob(JB):	3.75e-156
Kurtosis:	7.844	Cond. No.	373.

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

Figure 12

Dep. Variable : BUSINVx

OLS Regression Results

Dep. Variable:	srs	R-squared:	0.309
Model:	OLS	Adj. R-squared:	0.307
Method:	Least Squares	F-statistic:	162.5
Date:	Fri, 06 Jan 2023	Prob (F-statistic):	4.57e-59
Time:	15:56:45	Log-Likelihood:	2868.0
No. Observations:	730	AIC:	-5730.
Df Residuals:	727	BIC:	-5716.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.0024	0.000	10.840	0.000	0.002	0.003
ar1	0.4401	0.032	13.779	0.000	0.377	0.503
factor1	0.0014	0.000	7.650	0.000	0.001	0.002

Omnibus:	736.661	Durbin-Watson:	2.361
Prob(Omnibus):	0.000	Jarque-Bera (JB):	148192.778
Skew:	4.046	Prob(JB):	0.00
Kurtosis:	72.330	Cond. No.	181.

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

In question 11, we produced a data frame of the fitted values from the five models and save the file as 'fitted_values.csv' by using the command '.to_csv()'. In question 12, seaborn plots were created showing a scatter plot of the transformed variable and its fitted value for each variable. Moreover, we differentiated the plot between periods of time identified by the NBER as recessions and expansions. Graphs 1 through 5 in page 2 -3 show the graphs, and the left one is for expansion and the right one is for recession. We used the fitted values created in question 10, transformed data and 'NBER_DATES.csv'. By using the command 'pd.concat()', we concatenate the three objects. (See https://pandas.pydata.org/docs/reference/api/pandas.concat.html) The x- and y-axes are plotted as 'Actual values' and 'Fitted values', respectively, for expansion and recesstion by

using ‘sns.lmplot’ (See https://seaborn.pydata.org/generated/seaborn.lmplot.html). Finally, we save each to a file ‘srs.pdf’ where ‘srs’ is the key for each series.

Figure 13

```
# Question 11
fitted_values = fitted_values.dropna()
fitted_values.to_csv('fitted_values.csv')

# Question 12
NBER = pd.read_csv('NBER_DATES.csv', index_col=0)
NBER.index = pd.to_datetime(NBER.index)

for s in my_srs:
    srs_data = pd.concat([fitted_values[s],
                          tfdata.loc[fitted_values[s].index,s],
                          NBER.loc[fitted_values[s].index]],axis=1).set_axis(
                              ['Fitted','Actual','NBER'],axis=1)
    sns.lmplot(data=srs_data, x='Fitted', y='Actual', col='NBER')
    plt.suptitle('FF Model Fitted v. Actual for '+s)
    plt.tight_layout()
    plt.savefig( s +'.pdf')
    plt.show()

print('end')
```