21st Century Real Wage Growth in the UK

Table of contents

Introduction

Importing the Data

```python
# Importing the relevant modules and tools necessary

import numpy as np
import pandas as pd
import seaborn as sns
import statsmodels.api as sm

from pathlib import Path
import matplotlib.pyplot as plt
from functools import reduce
```

```python
from statsmodels.regression.linear_model import OLS
from statsmodels.tools import add_constant

from statsmodels.stats.diagnostic import het_breuschpagan
from statsmodels.stats.outliers_influence import variance_inflation_factor

from scipy import stats
from statsmodels.tsa.stattools import grangercausalitytests

# Since the wage and interest rate data was monthly, have to adjust it

# Involves resampling this data into quarterly format for analysis later on

# For wages, took the mean of the period
# For interest rates, took the rate at the end of the quarter

wages_path = Path('Data') / 'wages.csv'
wages_df = pd.read_csv(wages_path)
wages_df['Date'] = pd.to_datetime(wages_df['Date'], format = 'mixed')

wages_df.set_index('Date', inplace = True)
quarterly_wages_df = wages_df.resample('QE-MAR').mean()
quarterly_wages_df.index = quarterly_wages_df.index.to_period('Q')

# Repeating for bank rate

rates_path = Path('Data') / 'interest_rates.csv'
rates_df = pd.read_csv(rates_path)
rates_df['Date'] = pd.to_datetime(rates_df['Date'], format = 'mixed')

rates_df.set_index('Date', inplace = True)
quarterly_rates_df = rates_df.resample('QE-MAR').last()
quarterly_rates_df.index = quarterly_rates_df.index.to_period('Q')

# Creating the first dataframe, joining on the 'Date'column

first_df = pd.merge(quarterly_wages_df, quarterly_rates_df, on = ['Date'])


# Since the rest of the data was quarterly and in a similar format

# Wrote a function that could sort and create the dataframesy
# Takes file path, reads in data, much of the data was 2000 Q1

# However this isn't recognised and requires 2000-Q1 format instead
# If data was 2000-Q1, nothing is changed

def create_df(dataset, column = 'Date', folder = 'Data'):

    dataset_path = Path(folder) / dataset

    try:
        new_df = pd.read_csv(dataset_path)

    except FileNotFoundError:
        print(f'File {dataset} not found in folder {folder}')

    if column in new_df.columns:
```

```
        new_df[column] = pd.PeriodIndex((new_df[column].str.replace(' ', '-')), freq = 'Q')

    return new_df


# Importing the rest of the data into dataframes

inflation_df = create_df('inflation.csv')
unemployment_df = create_df('unemployment.csv')
OECD_growth_df = create_df('OECD_growth.csv')
gvt_spending_df = create_df('gvt_spending.csv')
```

Creating DataFrames

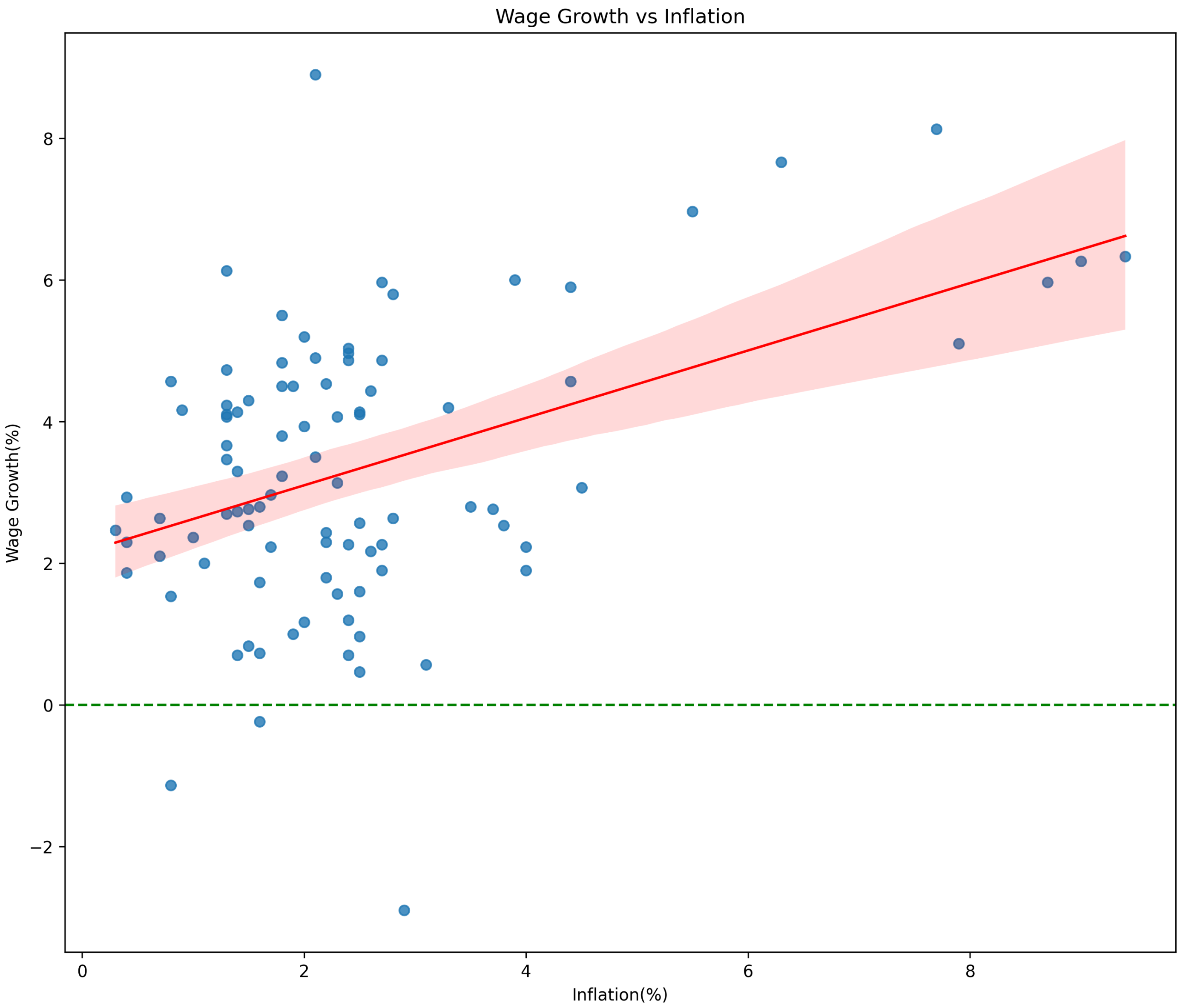
```
# Creating a dataframe with real (inflation adjusted) values in for future use

real_variables_df = pd.merge(first_df, inflation_df, on = ['Date'])

real_variables_df['Real Wage Growth(%)'] = (real_variables_df['Wage Growth(%)'] - real_variables_df['Inflatio

real_variables_df['Real Interest Rate(%)'] = (real_variables_df['Bank Rate(%)'] - real_variables_df['Inflatio

# First going to gather all dataframes

all_dfs = [first_df, inflation_df, unemployment_df, gvt_spending_df, OECD_growth_df]

# Then perform a merge on the 'Date' column

analysis_df = reduce(lambda left, right: pd.merge(left, right, on = ['Date'], how ='inner'), all_dfs)

analysis_df = analysis_df.dropna()

analysis_df
```

	Date	Wage Growth(%)	Bank Rate(%)	Inflation(%)	Unemployment Rate(%)	Gvt Expenditure Growth(%)	OEC
4	2001Q1	6.133333	5.7500	1.3	5.1	9.48	2.64
5	2001Q2	5.500000	5.2500	1.8	5.0	6.04	1.51
6	2001Q3	4.833333	4.8875	1.8	5.1	6.79	0.90
7	2001Q4	4.133333	4.0000	1.4	5.2	9.28	0.46
8	2002Q1	2.966667	4.0000	1.7	5.2	9.01	0.77
...	...	...	...	...	...	...	...
92	2023Q1	6.266667	4.0761	9.0	4.0	0.71	1.68
93	2023Q2	8.133333	4.6591	7.7	4.2	9.17	1.66
94	2023Q3	7.666667	5.2500	6.3	4.1	7.88	1.65
95	2023Q4	5.900000	5.2500	4.4	3.9	8.13	1.72
96	2024Q1	6.000000	5.2500	3.9	4.3	7.58	1.70

How Wage Growth Moves With Key Variables

Inflation

```python
# Setting the size of the plot

plt.figure(figsize = (12, 10))

# Using seaborn to plot a scatterplot with a line of best fit

sns.regplot(
    x = analysis_df['Inflation(%)'],
    y = analysis_df['Wage Growth(%)'],
    line_kws = {'color': 'red', 'linewidth': 1.5}
)

# Giving it a title

plt.title('Wage Growth vs Inflation')
plt.axhline(0, color = 'green', linestyle = '--')
plt.show()

# Calculating the pearsons correlation coefficient

inflation_corr = analysis_df['Wage Growth(%)'].corr(analysis_df['Inflation(%)'])

# Printing the coefficient

print(f'The pearsons correlation coefficient is {inflation_corr}')
```

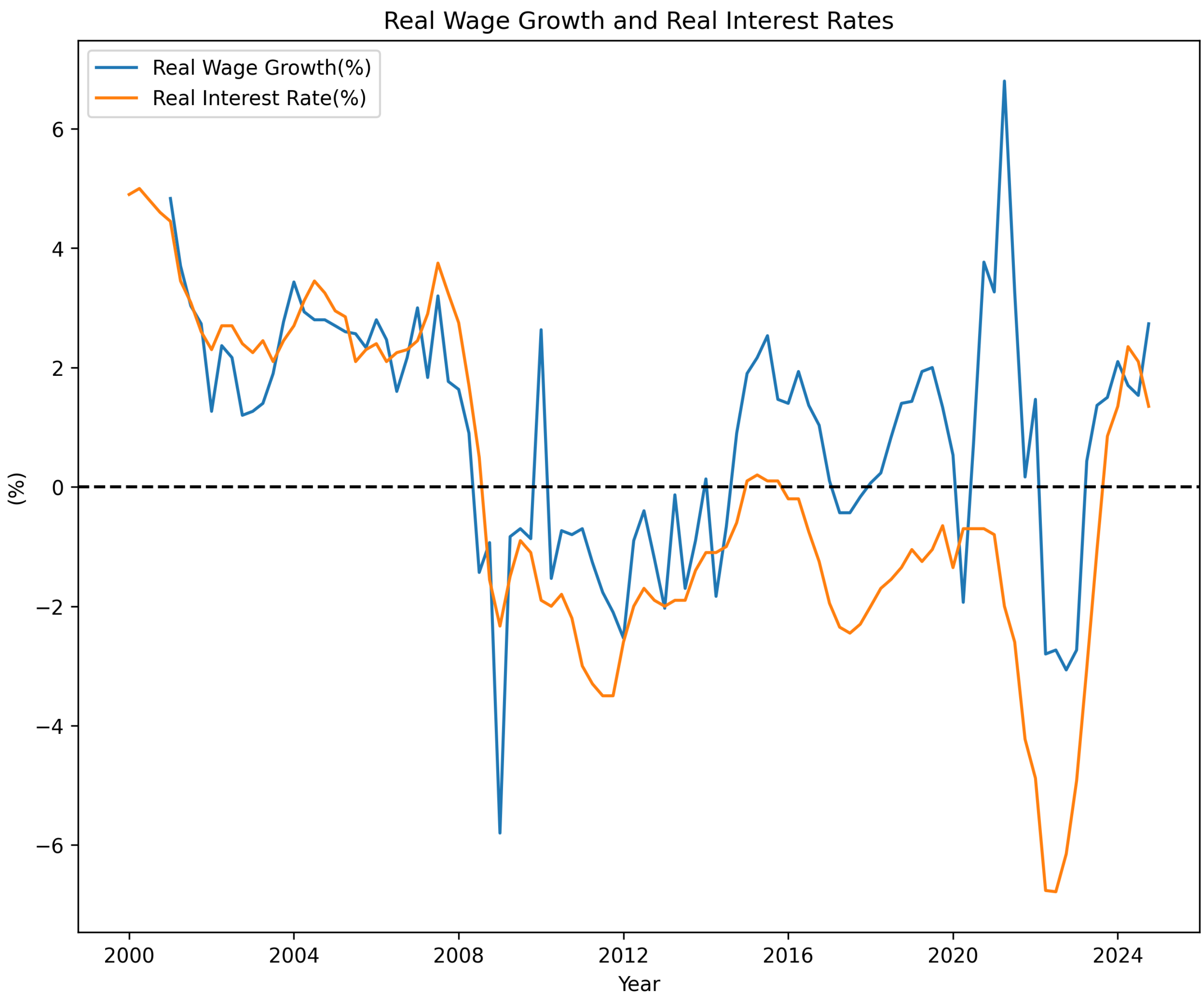



```
The pearsons correlation coefficient is 0.43312549815507967
```

Real Interest Rates

```python
plt.figure(figsize = (10, 8))

# Converting 'Date' column to datetime so it can plotted

real_variables_df['Date'] = pd.PeriodIndex(real_variables_df['Date'], freq = 'Q').to_timestamp()

# Looping through the columns in the dataframe to plot both time series together

for col in ['Real Wage Growth(%)', 'Real Interest Rate(%)']:
    plt.plot(real_variables_df['Date'], real_variables_df[col]\
        , label = col)


# Customising the graph
```

```python
plt.title('Real Wage Growth and Real Interest Rates')
plt.xlabel('Year')
plt.ylabel('(%)')
plt.legend()
plt.axhline(0, color = 'black', linestyle = '--')
plt.show()
```



The OLS Regression

```python
# Dropping the 4 NaN values at the start of the wage column
# This is because the data starts at 2000 so there's obviously no data for
# the first year's quarters

ols_regression_df = analysis_df.copy()
ols_regression_df.dropna()

ols_regression_df = analysis_df.drop('Date', axis = 1)
```

```python
# Catergorising my variables and ensuring the columns exist

dep_var = 'Wage Growth(%)'
indep_var = ['Bank Rate(%)', 'Inflation(%)', 'Unemployment Rate(%)', 'OECD Economic Growth(%)', 'Gvt Expendit

assert dep_var in ols_regression_df.columns, "'Wage Growth(%)' is not recognsied"
for column in indep_var:
    assert column in ols_regression_df.columns, f"'{column}' is not recognised"


# Creating my Y and X values for the regression

Y = ols_regression_df[dep_var]
X = ols_regression_df[indep_var]


# Runinng the OLS regression and printing the results

ols_regression = OLS(Y, add_constant(X)).fit(cov_type = 'HC1')
print(ols_regression.summary())
```

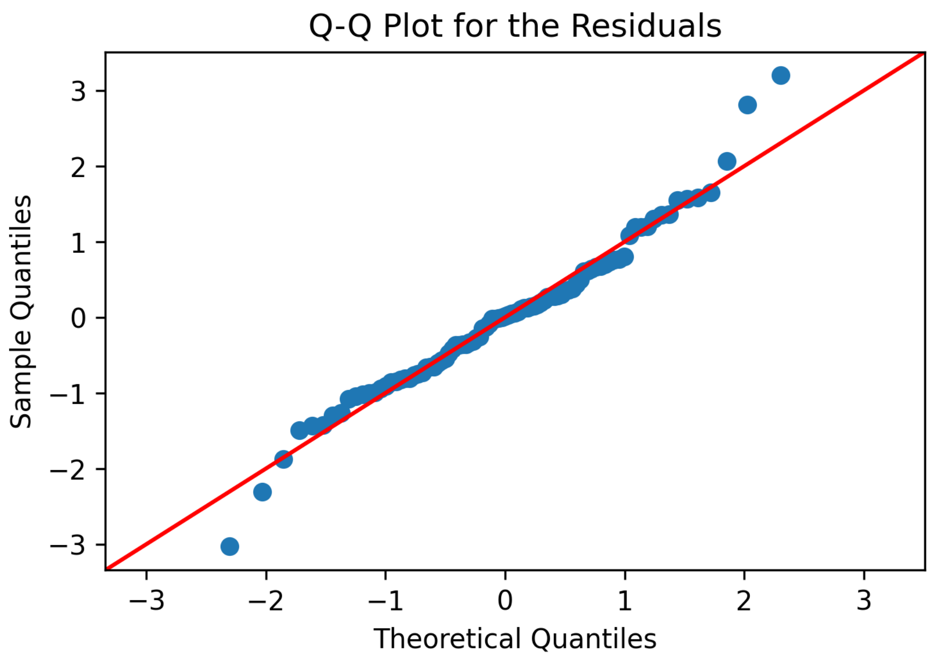
```
                            OLS Regression Results
==============================================================================
Dep. Variable:         Wage Growth(%)   R-squared:                       0.776
Model:                            OLS   Adj. R-squared:                  0.763
Method:                 Least Squares   F-statistic:                     57.20
Date:                Sun, 27 Apr 2025   Prob (F-statistic):           4.82e-26
Time:                        15:30:57   Log-Likelihood:                -125.77
No. Observations:                  93   AIC:                             263.5
Df Residuals:                      87   BIC:                             278.7
Df Model:                           5
Covariance Type:                  HC1
==============================================================================================
                                coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                         1.8140      0.577      3.145      0.002       0.684       2.944
Bank Rate(%)                  0.2367      0.061      3.862      0.000       0.117       0.357
Inflation(%)                  0.3987      0.051      7.881      0.000       0.300       0.498
Unemployment Rate(%)         -0.2998      0.084     -3.570      0.000      -0.464      -0.135
OECD Economic Growth(%)       0.4203      0.061      6.869      0.000       0.300       0.540
Gvt Expenditure Growth(%)     0.1726      0.039      4.400      0.000       0.096       0.250
==============================================================================
Omnibus:                        5.379   Durbin-Watson:                   1.561
Prob(Omnibus):                  0.068   Jarque-Bera (JB):                6.237
Skew:                           0.267   Prob(JB):                       0.0442
Kurtosis:                       4.151   Cond. No.                         59.2
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
```

Validation Tests

Q-Q plot for normality checks

```
# Plotting a Q-Q plot to test the normality of my residuals

sm.qqplot(ols_regression.resid, line='45', fit=True)
plt.title('Q-Q Plot for the Residuals')
```

```
Text(0.5, 1.0, 'Q-Q Plot for the Residuals')
```



Bootstrapping Confidence Intervals

```
# Setup for bootstrapping, including how many times to repeat the process

B = 1000
n = len(ols_regression_df)

# Creating an empty list to append to later

results_boot_list = []

# Writing a function that repeats the OLS as many times as specified
# Then, stores the coefficients

for i in range (B):
    boot_df = ols_regression_df.sample(n, replace = True)

    indep_boot = boot_df[indep_var]

    dep_boot = boot_df[dep_var]
```

```
    ols_boot = OLS(dep_boot, add_constant(indep_boot)).fit()

    results_boot_list.append(ols_boot.params.values)


# The dataframe with the initial bootstrapping results

results_boot_df = pd.DataFrame(results_boot_list, columns = ['const'] + indep_var)

# Getting my confidence intervals from the coefficients

CI_lower = results_boot_df.quantile(0.025)
CI_upper = results_boot_df.quantile(0.975)

# Table for the data so that the confidence intervals are columns

CI_boot = pd.concat([CI_lower, CI_upper], axis = 1)
CI_boot.columns = [' Bootstrap 0.025', ' Bootstrap 0.975']

# Getting the confidence intervals from my OLS for comparison

CI_ols = ols_regression.conf_int(alpha = 0.05)
CI_ols.columns = ['OLS 0.025', 'OLS 0.975']

# Joining the two datasets into one table

CI_merged = pd.concat([CI_boot, CI_ols], axis = 1)
CI_merged
```

	Bootstrap 0.025	Bootstrap 0.975	OLS 0.025	OLS 0.975
const	0.719963	3.111832	0.683629	2.944456
Bank Rate(%)	0.120171	0.375976	0.116579	0.356845
Inflation(%)	0.308183	0.505842	0.299581	0.497918
Unemployment Rate(%)	-0.467718	-0.139388	-0.464374	-0.135190
OECD Economic Growth(%)	0.221371	0.532913	0.300377	0.540222
Gvt Expenditure Growth(%)	0.078707	0.255429	0.095736	0.249543

Heteroskedasticity test

```
# Running a heteroskedasticity test

bp_test = het_breuschpagan(ols_regression.resid, ols_regression.model.exog)
bp_test
```

```
(np.float64(8.598077317110123),
 np.float64(0.12620996250069955),
 np.float64(1.7725490197635583),
 np.float64(0.1268895556659283))
```

Multicollinearity test

```python
# Running a test for multicollnearity

X_with_constant = add_constant(X)
vif_test = pd.DataFrame()
vif_test['Variable'] = X_with_constant.columns
vif_test['VIF'] = [variance_inflation_factor(X_with_constant, i)
                   for i in range(X_with_constant.shape[1])]


vif_test
```

	Variable	VIF
0	const	45.152924
1	Bank Rate(%)	1.453048
2	Inflation(%)	1.158060
3	Unemployment Rate(%)	1.393704
4	OECD Economic Growth(%)	1.390184
5	Gvt Expenditure Growth(%)	1.996178

Discussion of the OLS Results

Setup For ARIMAX

First Differencing and ADF Tests

```python
from statsmodels.tsa.stattools import adfuller

# Creating a new dataframe to use for first-differencing and VECM tests

copy_df = ols_regression_df.copy()
stationary_df = copy_df.diff().dropna()

# Runs the ADF(Augmented Dickey-Fuller) test on every column

# Returns the results into a previously empty list
# Purpose is to make sure data is stationary after first-differencing

def adf_test(df):
    adf_results = {}

    for column in df.columns:
        result = adfuller(df[column].dropna(), maxlag = 5)

        adf_results[column] = {
            'ADF Statistic': result[0],
            'p-value': result[1],
            'Lags Used': result[2],
            'Number of Observations Used': result[3],
            'Critical Values': result[4]
        }

    return adf_results

# Running it on the stationary dataframe
```

```python
adf_results = adf_test(stationary_df)

# Printing out the results from the ADF test
# Can inspect lags used and the relevant t-stats and p-values

for column, result in adf_results.items():
    print(f"\nResults for {column}:")

    print(f"ADF Statistic: {result['ADF Statistic']}")

    print(f"p-value: {result['p-value']}")

    print(f"Lags Used: {result['Lags Used']}")
```

```
Results for Wage Growth(%):
ADF Statistic: -7.419205473710482
p-value: 6.806704603396944e-11
Lags Used: 3

Results for Bank Rate(%):
ADF Statistic: -5.233964606420418
p-value: 7.491639441559678e-06
Lags Used: 0

Results for Inflation(%):
ADF Statistic: -5.441922357424594
p-value: 2.764021647259073e-06
Lags Used: 3

Results for Unemployment Rate(%):
ADF Statistic: -4.153122515613913
p-value: 0.0007902001474869033
Lags Used: 1

Results for Gvt Expenditure Growth(%):
ADF Statistic: -5.793944360466175
p-value: 4.800369834604658e-07
Lags Used: 3

Results for OECD Economic Growth(%):
ADF Statistic: -7.6491531405294255
p-value: 1.8087007502155e-11
Lags Used: 3
```

```python
# For forecasting purposes

stationary_df['Date'] = analysis_df['Date']
stationary_df = stationary_df.set_index('Date')

stationary_df = stationary_df.asfreq(stationary_df.index.freq)
stationary_df.index = stationary_df.index.to_timestamp()
```

ARIMAX - Testing Predictive Power of the Model

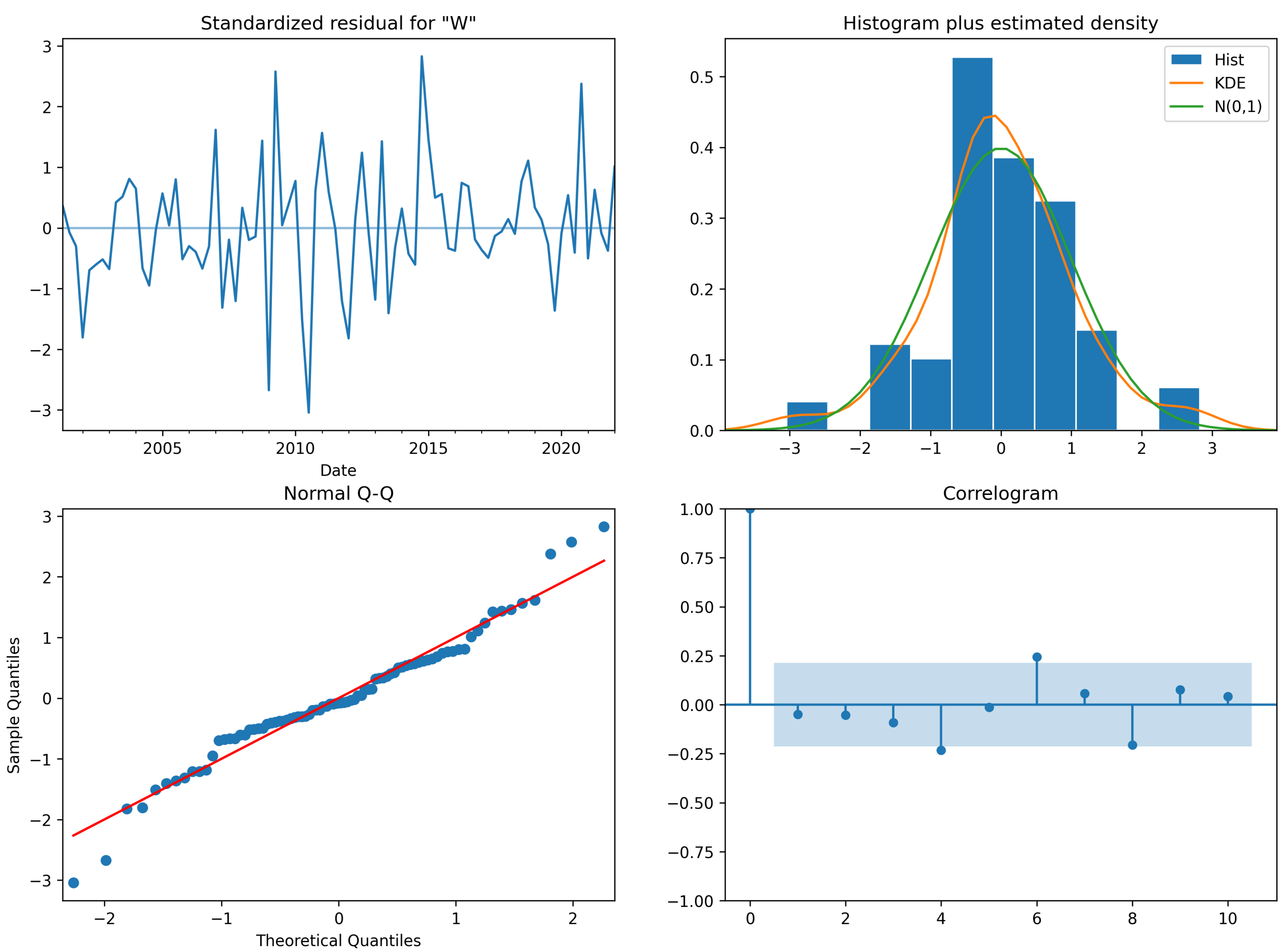
```
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error

# Set the period I'm going to be forecasting

testing_timeframe = 8

# Create the test and train datasets

arimax_train = stationary_df[:-testing_timeframe]
arimax_test  = stationary_df[-testing_timeframe:]


# Define the exogenous variables
exog_arimax_train = arimax_train[['Bank Rate(%)',\
    'Inflation(%)','Unemployment Rate(%)',\
    'Gvt Expenditure Growth(%)', 'OECD Economic Growth(%)']]

exog_arimax_test  = arimax_test[['Bank Rate(%)',\
    'Inflation(%)','Unemployment Rate(%)',\
    'Gvt Expenditure Growth(%)', 'OECD Economic Growth(%)']]

# Creates and fits the model with the appropriate parameters
arimax_model = ARIMA(arimax_train['Wage Growth(%)'],
                    order = (2,0,1),
                    exog = exog_arimax_train
                   ).fit()

print(arimax_model.summary())

# Creates forecasts on training set

forecasts_on_train = arimax_model.predict()

# Creates forecasts on test set

forecasts_on_test  = arimax_model.forecast(len(arimax_test), exog = exog_arimax_test)
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:         Wage Growth(%)   No. Observations:                   84
Model:                 ARIMA(2, 0, 1)   Log Likelihood                -101.128
Date:                Sun, 27 Apr 2025   AIC                            222.257
Time:                        15:30:59   BIC                            246.565
Sample:                    04-01-2001   HQIC                           232.028
                         - 01-01-2022
Covariance Type:                  opg
===============================================================================================
                                coef    std err          z      P>|z|      [0.025      0.975]
-----------------------------------------------------------------------------------------------
const                         0.0430      0.065      0.663      0.507      -0.084       0.170
Bank Rate(%)                  0.7434      0.203      3.668      0.000       0.346       1.141
Inflation(%)                  0.0510      0.168      0.303      0.762      -0.279       0.381
Unemployment Rate(%)          0.5157      0.330      1.565      0.118      -0.130       1.162
Gvt Expenditure Growth(%)     0.0782      0.046      1.691      0.091      -0.012       0.169
```

```
OECD Economic Growth(%)        0.3805      0.059      6.463      0.000       0.265       0.496
ar.L1                         -1.4818      0.070    -21.083      0.000      -1.620      -1.344
ar.L2                         -0.7198      0.053    -13.515      0.000      -0.824      -0.615
ma.L1                          0.9531      0.053     18.083      0.000       0.850       1.056
sigma2                         0.6324      0.086      7.378      0.000       0.464       0.800
===================================================================================
Ljung-Box (L1) (Q):                   0.20   Jarque-Bera (JB):                 5.40
Prob(Q):                              0.65   Prob(JB):                         0.07
Heteroskedasticity (H):               0.89   Skew:                            -0.00
Prob(H) (two-sided):                  0.75   Kurtosis:                         4.24
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

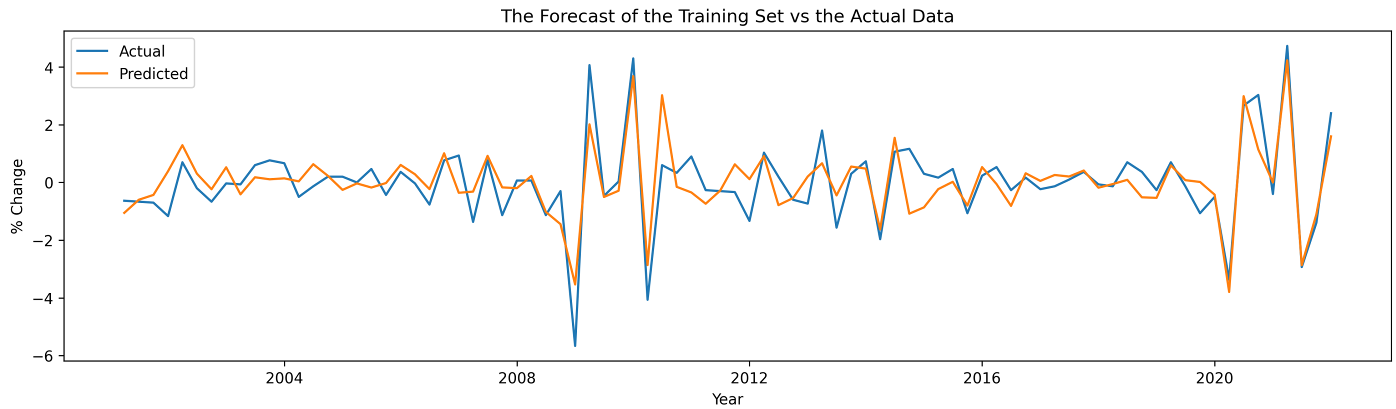
Diagnostic Check

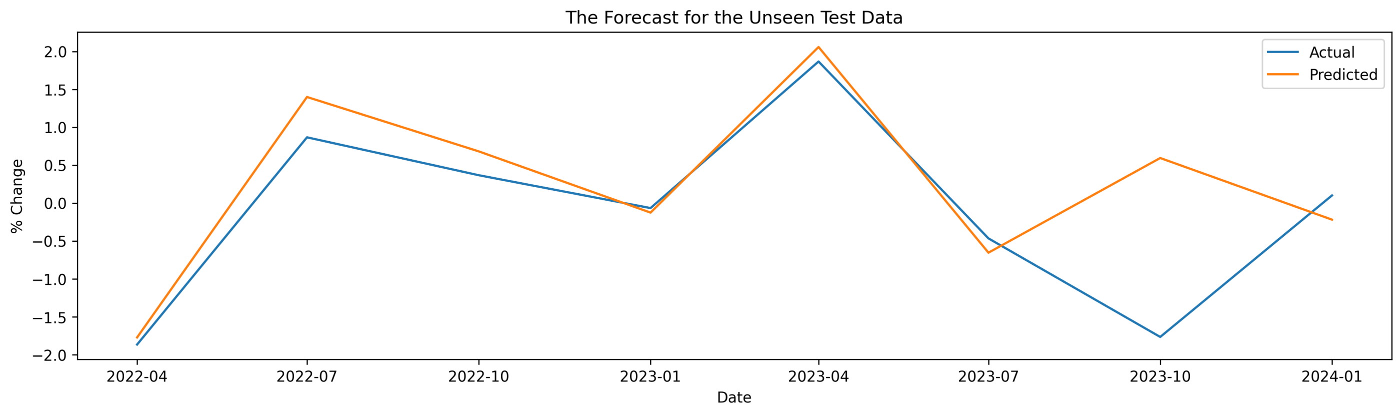
```python
arimax_model.plot_diagnostics(figsize = (14,10))
plt.show()
```



Forecasting

```
# Plotting the train and test data against their corresponding forecasts

plt.figure(figsize=(16,4))
plt.plot(arimax_train['Wage Growth(%)'], label="Actual")
plt.plot(forecasts_on_train, label="Predicted")

plt.title('The Forecast of the Training Set vs the Actual Data')
plt.xlabel('Year')
plt.ylabel('% Change')
plt.legend()

# Repeating for test data, where the model is really tested
plt.figure(figsize=(16,4))
plt.plot(arimax_test['Wage Growth(%)'], label="Actual")
plt.plot(forecasts_on_test, label="Predicted")

plt.title('The Forecast for the Unseen Test Data')
plt.xlabel('Date')
plt.ylabel('% Change')
plt.legend()
plt.show()
```





Discussing the ARIMAX Results, and Comparing to the OLS

Final Words