Appendix: GoveaR_US vs EU (HICP)

Import Packages

```
# DF Manipulation
import numpy as np
import pandas as pd
# Graphs
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
# Directories
import glob
import os
# Stationarity Testing
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import acf, pacf
# Modeling
import statsmodels.api as sm
from statsmodels.tsa.holtwinters import ExponentialSmoothing

```

+ Code + Text

Data Import and Cleansing

US: Harmonized Index of Consumer Prices

Source: https://fred.stlouisfed.org/series/CP0000USM086NEST

1. Wanting to use the seasonally adjusted index (sai)
2. converting the date to datetime format
3. limiting columns to the title of the commonity and cpi while indexing the date
4. importing all columns and data for exploratory analysis

```
hicp = pd.read_excel('/Users/reygovea/Documents/Fall 2022/Stat Consulting /Final Proj./Data/US XLSX/CP0000USM086NEST.xls', sk

```

```
US_hicp = hicp.rename(columns = {'observation_date': 'DATE', 'CP0000USM086NEST': 'HICP'})

US = US_hicp.set_index('DATE')
US.head()
```

	HICP
DATE	
2001-12-01	74.09
2002-01-01	74.22
2002-02-01	74.39
2002-03-01	74.83
2002-04-01	75.34

```
US.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 245 entries, 2001-12-01 to 2022-04-01
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   HICP    245 non-null    float64
dtypes: float64(1)
memory usage: 3.8 KB
```

```
def train_test_split(df = US, percent = 0.75, train = 'US_train', test = 'US_test'):
    dataset_len = len(df)
    split_index = round(dataset_len*percent)
    train_set_end_date = df.index[split_index]
    train = df.loc[df.index <= train_set_end_date].copy()
    test = df.loc[df.index > train_set_end_date].copy()

    return train, test

US_train = train_test_split()[0]
US_test = train_test_split()[1]
```

```
# US_train.head()
US_test.head()
```

EU: Harmonized Index of Consumer Prices

Source: https://fred.stlouisfed.org/series/CP0000EZ19M086NEST

```
# starting at index row 10 to skip text in excel file
hicp = pd.read_excel('/Users/reygovea/Documents/Fall 2022/Stat Consulting /Final Proj./Data/Euro XLSX/CP0000EZ19M086NEST.xls'
# hicp.info()
```

```
EU_hicp = hicp.rename(columns = {'observation_date': 'DATE', 'CP0000EZ19M086NEST': 'HICP'})
EU = EU_hicp.set_index('DATE')
EU.head()
```

```
EU_hicp.info()
```

```
EU_train = train_test_split(df = EU, percent = 0.75, train = 'EU_train', test = 'EU_test')[0]
EU_test = train_test_split(df = EU, percent = 0.75, train = 'EU_train', test = 'EU_test')[1]
```

```
# EU_train.head()
# EU_test.head()
```

EDA

```
# No null datatype or year classifications
EU['HICP'].isnull().any()
```

US: HICP All Items

General Graphs

```
# Scatter plot of the CPI for all items
def scatter(df = US_hicp, y_lab = 'HICP', title = 'US'):
    scatter =px.scatter(df,
               x = 'DATE',
               y = y_lab,
               title = '{} HICP Scatter Plot (All items)'.format(title))
    return scatter

scatter()
```

Stationarity - All Items|Seasonally Adj. Index

Dickey Fuller Test for Stationarity

H0: $\alpha = 1$

In the Dickey fuller Test, the test statistics (0.463599) is greater than the critical value @ 5% of -2.873559 such that we fail to reject the null hypothesis that the time series is not stationary.

source: https://builtin.com/data-science/time-series-python

```
def test_stationarity(df = US_train, y_lab = 'HICP'):
#     Filtering to desired title and then dropping
    t = df.copy()

    t['TYPE'] = 'ui'
#     Calculating rolling mean/std for 6 mo. pd.
    rolmean = t.rolling(6).mean()
    rolmean['TYPE'] = 'rolmean'

    rolstd = t.rolling(6).std()
    rolstd['TYPE'] = 'rolstd'

#     Concatonating the means and std s.t. can distinguish in the color scheme of the plot
    s = pd.concat([t, rolmean, rolstd])
#     Plotting calculations
    fig = px.line( s,
            x = s.index,
            y = y_lab,
            #Variable to label sai, mean, or std
            color = 'TYPE',
            title = '{} Unadjusted Index & Rolling Mean/Std (All items)'.format(y_lab)
    )

#     Dickey-Fuller Stationarity Test
    print('Dickey-Fuller Test:')
    t = t.drop(columns = ['TYPE'])

    test = adfuller(t, autolag = 'BIC')
    output = pd.Series(test[0:4], index = ['Test Statistic',
                                           'p-value',
                                           '#Lags Used',
                                           '# of Obs. Used'])
    for key,value in test[4].items():
        output['Critical Value (%s)'%key] = value
    print(output)

    return(fig)


test_stationarity()
```

```
# Using Differencing of 1 to detrend the data
US_stationary = US_train.diff().dropna()
US_stationary.head()
```

https://otexts.com/fpp2/stationarity.html

https://stats.stackexchange.com/questions/394796/should-my-time-series-be-stationary-to-use-arima-model

```
# After First Order Differencing, we reject the null hypothesis that the data is not stationary
test_stationarity(df = US_stationary, y_lab = 'HICP')
```

```
# US_stationary.head()
```

```
# taking 12 month seasonal difference of first order differencing
US_seasonal = US_stationary - US_stationary.shift(12)
US_seasonal = US_seasonal.dropna(inplace = False)
# US_seasonal.head()
test_stationarity(df = US_seasonal, y_lab = 'HICP')
```

Decomposition

Note: edit to set axis as date

Note: consider why there are two means and how to eliminate them.

from statsmodels.tsa.seasonal import seasonal_decompose def decompose(df = US_stationary, title = 'All items', y_lab = 'HICP'):

Filtering to desired title and then dropping

```
    titles = [title]
    t = df[df['TITLE'].isin(titles)]
    t = t.drop(columns = ['TITLE'])
    avg_t = t.groupby(['DATE'], as_index = False)[[y_lab]].mean()


    decompose = seasonal_decompose(avg_t[y_lab], model = 'additive', period = 6)

    plt.rcParams["figure.figsize"] = (16,10)
    decompose.plot()
    plt.show()

    return decompose
```

decompose()

ACF and PACF

source: https://towardsdatascience.com/time-series-from-scratch-autocorrelation-and-partial-autocorrelation-explained-1dd641e3076f

source: https://builtin.com/data-science/time-series-python

Calculating ARIMA(p,d,q): https://analyticsindiamag.com/quick-way-to-find-p-d-and-q-values-for-arima/

```
# Stationary ACF
def ACF(df = US_train, y_lab = 'HICP', title = 'US'):
    t = df.copy()

   #Generating the ACF for Lags(i)
   # Lag is the range of 1-25 but Lags is a field of key lag values
    Lag = list(range(1,26))
    autocorr = []
    Lags =[1,3,9,12,15,20,25]
    print('{} Autocorrelation:'.format(title))
    for i in Lag:
        autocorr_lag = t[y_lab].autocorr(lag=i)
        if i in Lags:
            print(i,'Month Lag:', autocorr_lag)

        autocorr.append(autocorr_lag)

    #changing the index s.t. it is = Lag number
    autocorr = pd.DataFrame(autocorr)
    autocorr.index += 1

    #Plotting ACF
    Lag_Autocorr = plot_acf(autocorr)
#     Lag_Autocorr = px.bar(autocorr,
#                         labels ={
#                             'value': 'ACF',
#                             'index': 'Lag'},
#                         title = 'Autocorrelation ({})'.format(title))

#     Lag_Autocorr.update_layout(showlegend = False)

    return Lag_Autocorr

ACF()
```

```
# Stationary PACF
def PACF(df = US_train, y_lab = 'HICP'):
    #Filtering to desired title and then dropping
    t = df.copy()

#     Generating PACF values
    p_autocorr = pacf(t[y_lab])

    #Plotting PACF
    Lag_pacf = plot_pacf(p_autocorr)

```

```
    return Lag_pacf

PACF()
```

US Seasonal ARIMA

Source: https://www.youtube.com/watch?v=5Q5p6eVM7zM

Source: https://www.youtube.com/watch?v=l7jpmJLDmxQ

Source: https://medium.com/@ooemma83/how-to-interpret-acf-and-pacf-plots-for-identifying-ar-ma-arma-or-arima-models-498717e815b6#:~:text=The%20basic%20guideline%20for%20interpreting,q%20for%20MA(q).

ARIMA(p,d,q)X(P,D,Q)S

p = Non-Seasonal AR order(PACF)

d = Non-Seasonal Differencing

q = Non-Seasonal MA Order (ACF)

P = Seasonal AR order

D = Seasonal Differencing

Q = Seasonal MA Order

S = Time Span of Seasonal Pattern

US_ARIMA(1,1,0)X(0,1,0)_12

Note: lag=1 is always 1

p=1 bc there is 1 initial lags that are significant in US PACF for AR(1)

d=1 bc the basic scatter plot is positive linear and non-seasonal 1st order differencing makes stationary

q=2 bc there are several significant lags (2) after lag = 1 in the ACF which determines the MA(2) order

P=7 bc there are 7 significant spikes as the lags continue in the PACF graph after initial significance

D=1 bc not all seasonal ups/dows are equal, this helps make the seasonality less prevelant (subtracts previous cycle from current.

Q=0 bc there are no repeated goups of spikes as the lags coninue in the ACF graph

S=12 bc this is the month differencing of each apparent cycle.

```
def SARIMA(train = US_train, test = US_test, pdq = [3,1,2], PDQS=[6,1,0,12]):
#     Creating copies of df
    t = train.copy()
    tx = test.copy()
#     Modeling the seasonal ARIMA
    mod = sm.tsa.statespace.SARIMAX(t, order = (pdq[0],pdq[1],pdq[2]),
                                    seasonal_order = (PDQS[0],PDQS[1],PDQS[2],PDQS[3]))
    results = mod.fit()
#     printing the summary of the fitted model
    print(results.summary())

#     Forecasting the testing data
    y_pred = results.get_forecast(len(tx.index))
    y_pred_df = y_pred.conf_int(alpha = 0.05)
    y_pred_df["Predictions"] = results.predict(start = y_pred_df.index[0], end = y_pred_df.index[-1])
    y_pred_df.index = tx.index #setting the dated index
    y_pred_out = y_pred_df["Predictions"] #creating exclusive prediction df

#     Summary of predictions and their errors
    pred_summary = y_pred.summary_frame()

#     Defining CI bounds
    upper_95 = pred_summary['mean_ci_upper']
    upper_95.index = tx.index

    lower_95 = pred_summary['mean_ci_lower']
    lower_95.index = tx.index

#     plotting figures
    upper = plt.plot(upper_95, color = 'blue', label = 'Upper_95%_CI')
```

```
    train = plt.plot(t, color='black', label = 'train')
    test = plt.plot(tx, color='red', label = 'test')
    predicted = plt.plot(y_pred_out, color='green', label = 'Predictions')
    lower = plt.plot(lower_95, color = 'blue', label = 'Lower_95%_CI')
    plt.fill_between(tx.index, lower_95, upper_95, color = 'b', alpha = 0.2)
    plt.xlabel('DATE')
    plt.ylabel('HICP')
    plt.title('SARIMA')
    plt.legend()
    plt.show()

    #Returning the prediction summary dataframe for plotting the errors
    return pred_summary

US_SARIMA_MSE = SARIMA()
```

US ETS Model

https://www.statsmodels.org/dev/examples/notebooks/generated/ets.html

https://www.statsmodels.org/dev/generated/statsmodels.tsa.exponential_smoothing.ets.ETSModel.html

https://www.statsmodels.org/dev/examples/nothttps://towardsdatascience.com/time-series-in-python-exponential-smoothing-and-arima-processes-2c67f2a52788ebooks/generated/ets.html

https://medium.com/analytics-vidhya/python-code-on-holt-winters-forecasting-3843808a9873

```
from sklearn.metrics import mean_squared_error

def ETS(train = US_train, test = US_test, period = 12):
#     Creating copies of df
    t = train.copy()
    tx = test.copy()
#     Modeling the seasonal Exponential Smoothing
    mod = ExponentialSmoothing(t,
                               trend = 'add',
                               seasonal = 'mul',
                               seasonal_periods = period)

    results = mod.fit()
    y_pred = results.forecast(len(tx))

#     calculating the MSE for i in the range of test set

    MSE = []
    for i in range(1,len(tx)):
        mean_square = mean_squared_error(y_pred.iloc[:i],tx.iloc[:i])
        MSE.append(mean_square)

    MSE = pd.DataFrame(MSE, columns = ['ETS_MSE'])
    MSE.index = tx.index[1:] #Adjusting the index to exclude tx first row for non zero MSE

#     plotting figures
    train = plt.plot(t, color='black', label = 'train')
    test = plt.plot(tx, color='red', label = 'test')
    predicted = plt.plot(y_pred, color='green', label = 'Predictions')
    plt.xlabel('DATE')
    plt.ylabel('HICP')
    plt.title('Exponential Smoothing')
    plt.legend()
    plt.show()


    return MSE

US_ETS_MSE = ETS()
```

US ARIMA vs. ETS MSE

```
US_SARIMA_MSE = pd.DataFrame(US_SARIMA_MSE['mean_se'])
US_SARIMA_MSE = US_SARIMA_MSE.rename(columns = {'mean_se':'SARIMA_MSE'})

```

```

mse = pd.merge(US_SARIMA_MSE, US_ETS_MSE, left_index = True, right_index = True)
# mse.head()
```

```
plt.plot(mse['SARIMA_MSE'], color = 'purple', label = 'SARIMA')
plt.plot(mse['ETS_MSE'], color='orange', label = 'ETS')

plt.xlabel('DATE')
plt.ylabel('MSE')
plt.title('US SARIMA vs. ETS MSE')
plt.legend()
plt.show()
```

US GARCH

Source: https://github.com/ritvikmath/Time-Series-Analysis/blob/master/GARCH%20Model.ipynb Source: https://www.youtube.com/watch?v=96nSIMS9_Y0 ARCH and Garch Models: https://online.stat.psu.edu/stat510/lesson/11/11.1 P Q determination Video: https://www.google.com/search?q=pq+garch+python&rlz=1C5CHFA_enUS760US760&sxsrf=ALiCzsYT2CAEH3tRmvVvzx9enROjq8rZtQ%3A1668124233786&ei=SY5tY5_DL_mFwbkP1-2fgA4&ved=0ahUKEwjfi5WD56T7AhX5QjABHdf2B-AQ4dUDCBA&uact=5&oq=pq+garch+python&gs_lcp=Cgxnd3Mtd2l6LXNlcnAQAzIFCCEQoAEyBQghEKABMgUIIRCgATIFCCEQqwI6BAgAEEdKBAhNGAFKBAhBGABKBAhGGABQK1jYCGCpCmgAcAJ4AYABjwOIAccHkgEHMC4yLjEuMZgBAKABAcgBCMABAQ&sclient=gws-wiz-serp#kpvalbx=_TY5tY9OvEcGOwbkP9r2M-Ac_30

GARCH(1,1) model is the best because the p-value on the volitility model is completely correlated for beta with p,q greater than 1,1

```
# applying seasonal differencing to test data for predictive GARCH modeling
US_test_stationary =  US_test.diff().dropna()
US_test_seasonal = US_test_stationary - US_test_stationary.shift(12)
US_test_seasonal = US_test_seasonal.dropna(inplace = False)
# US_seasonal.head()
test_stationarity(df = US_test_seasonal, y_lab = 'HICP')
```

```
US_seasonal.mean()
```

https://www.w3resource.com/python-exercises/pandas/plotting/pandas-plotting-exercise-18.php

https://goldinlocks.github.io/ARCH_GARCH-Volatility-Forecasting/#:~:text=GARCH(1%2C1)%20parameter%20dynamics&text=Intuitively%2C%20GARCH%20variance%20forecast%20can,the%20previous%20forecast%20was%20made.

```
from arch import arch_model

def GARCH(train = US_seasonal, test = US_test_seasonal, vol = 'GARCH', pq = [1,1]):
    t = train
    tx = test

#     fitting train data
    model = arch_model(train, mean = 'Zero', vol = vol, p=pq[0], q=pq[1])
    model_fit = model.fit()
    print(model_fit.summary())


#     predicting test data
    pred = model_fit.forecast(horizon=len(tx), reindex = False)

    pred_vol = np.sqrt(pred.residual_variance.values[-1, :])
    pred_vol = pd.DataFrame(pred_vol, columns = ['Vol'])
    pred_vol.index = tx.index

#     Defining Data Volitility through variance
    t_rolstd = t.rolling(6).var()
    tx_rolstd = tx.rolling(6).var()

    #     plotting
#     train = plt.plot(t, color='black', label = 'train', linestyle = '--')
#     test = plt.plot(tx, color='black', label = 'test', linestyle = '--')
    preds = plt.plot(pred_vol, color='red', label = 'Predicted Volitility')
```

```
    train_vol = plt.plot(t_rolstd, color='green', label = 'Train Rolling Variance')
    test_vol = plt.plot(tx_rolstd, color='green', label = 'Test Rolling Variance')
    plt.xlabel('DATE')
    plt.ylabel('Volatility')
    plt.title('GARCH')
    plt.legend()
    plt.show()


garch_pred = GARCH()
```

https://builtin.com/data-science/time-series-forecasting-python

EU: HCIP All Items

General Graphs

```
scatter =px.line(EU_hicp,
               x = 'DATE',
               y = 'HICP',
               title = 'HICP Scatter Plot (All Items)')
scatter
```

Stationarity

```
test_stationarity(df = EU_train, y_lab = 'HICP')
```

```
# Using Differencing of 1 to detrend the data
EU_stationary = EU_train.diff().dropna()
# EU_stationary
```

```
test_stationarity(df = EU_stationary, y_lab = 'HICP')
```

```
# taking 6 month seasonal difference of first order differencing
EU_seasonal = EU_stationary - US_stationary.shift(6)
EU_seasonal = EU_seasonal.dropna(inplace = False)
# US_seasonal.head()
test_stationarity(df = EU_seasonal, y_lab = 'HICP')
```

Seasonal Decomposition

```
# decompose(df=EU_hicp, title = 'All items', y_lab = 'HICP')
```

ACF and PACF

```
ACF(df = EU_train, y_lab = 'HICP')
```

```
PACF(df=EU_train, y_lab = 'HICP')
```

EU: SARIMA

EU_SARIMA(2,1,3)X(

p = 1 -> initial lags have 1 significant values in PACF (AR(1) model)

d = 1 -> linear trend s.t. first order differencing.

q = 2 -> ACF has 3 significant initial lags (MA(2) model)

P = 9 -> PACF has 3 significant lags after 1st initial grouping

D = 1 -> 1 order of seasonal differencing

Q = 0 -> No significant lags after initial lags

S = 6 -> seasonal cycles of graph seem to be every 6 months

```
EU_SARIMA_MSE = SARIMA(train=EU_train, test = EU_test, pdq = [2,1,2], PDQS = [9,1,0,6])
```

EU ETS Model

https://otexts.com/fpp2/holt.html

```
EU_ETS_MSE = ETS(train = EU_train, test = EU_test, period = 6)
```

EU SARIMA vs. ETS

```
EU_SARIMA_MSE = pd.DataFrame(EU_SARIMA_MSE['mean_se'])
EU_MSE = EU_SARIMA_MSE.rename(columns = {'mean_se':'SARIMA_MSE'})


mse = pd.merge(EU_SARIMA_MSE, EU_ETS_MSE, left_index = True, right_index = True)
# mse.head()
```

```
plt.plot(EU_SARIMA_MSE, color = 'purple', label = 'SARIMA')
plt.plot(EU_ETS_MSE, color='orange', label = 'ETS')

plt.xlabel('DATE')
plt.ylabel('MSE')
plt.title('EU SARIMA vs. ETS MSE')
plt.legend()
plt.show()
```

US & EU MSE Comparison

```
# US_SARIMA_MSE = pd.DataFrame(US_SARIMA_MSE['mean_se'])
# US_SARIMA_MSE = US_SARIMA_MSE.rename(columns = {'mean_se':'US'})

EU_SARIMA_MSE = pd.DataFrame(EU_SARIMA_MSE['mean_se'])
EU_SARIMA_MSE = EU_SARIMA_MSE.rename(columns = {'mean_se':'EU'})


mse = pd.merge(US_SARIMA_MSE, EU_SARIMA_MSE, left_index = True, right_index = True)
mse.head()
```

```
plt.plot(US_SARIMA_MSE, color = 'blue', label = 'US')
plt.plot(EU_SARIMA_MSE, color='green', label = 'EU')

plt.xlabel('DATE')
plt.ylabel('MSE')
plt.title('US vs. EU ARIMA MSE')
plt.legend()
plt.show()
```

```
plt.plot(US_ETS_MSE, color = 'blue', label = 'US')
plt.plot(EU_ETS_MSE, color='green', label = 'EU')

plt.xlabel('DATE')
plt.ylabel('MSE')
plt.title('US vs. EU ETS MSE')
plt.legend()
plt.show()
```

EU GARCH

```
# applying seasonal differencing to test data for predictive GARCH modeling
EU_test_stationary =  EU_test.diff().dropna()
EU_test_seasonal = EU_test_stationary - EU_test_stationary.shift(6)
EU_test_seasonal = EU_test_seasonal.dropna(inplace = False)
# US_seasonal.head()
test_stationarity(df = EU_test_seasonal, y_lab = 'HICP')
```

```
# GARCH of seasonally differenced data
GARCH(train = EU_seasonal, test = EU_test_seasonal, pq = [1,1])

```