Multivariable forecasting problem about Price Index Agricultural in Ireland

Table of contents

Following the Cross Industry Standard Process **CRISP-DM**, the phases and plan of the project are available here: https://github.com/users/sba22223nestorpereira/projects/1

In [1]:

```
import pandas as pd
import numpy as np
from numpy import array
from numpy import hstack

import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import datetime as dt
%matplotlib inline

sns.set_style('darkgrid')
```

```python
import warnings
warnings.filterwarnings('ignore') # We can suppress the warnings
```

In [2]:

```python
# importing necessary libraries
import pandas as pd
import numpy as np

# Seaborn
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import date, datetime, timedelta
from scipy import stats

sns.set_style('darkgrid')

import markdown

%matplotlib inline
import warnings
warnings.filterwarnings('ignore') # We can suppress the warnings


# from Scipy statistics distribution
from scipy.stats import poisson
from scipy.stats import norm
import statistics

from numpy import exp
from scipy.stats import boxcox
```

In [3]:

```python
# train test
from sklearn.model_selection import train_test_split


# stratified k-fold cross validation evaluation regression models
from numpy import loadtxt
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from keras import optimizers
from keras import losses
from keras import metrics


#Feature Scaling
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder

#import libraries for regression

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import ElasticNet
from sklearn.linear_model import ElasticNetCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error

import sys

import tensorflow.keras
import pandas as pd
import sklearn as sk
```

```
import tensorflow as tf
from numpy.random import seed
#from tensorflow import set_random_seed
from tensorflow.keras.utils import set_random_seed

print(f"Tensor Flow Version: {tf.__version__}")
print(f"Keras Version: {tensorflow.keras.__version__}")
print()
print(f"Python {sys.version}")
print(f"Pandas {pd.__version__}")
print(f"Scikit-Learn {sk.__version__}")

# Load libraries NN

from keras import models
from keras import layers
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasRegressor


# statsmodels is a Python modules statistical models

import statsmodels
import statsmodels.api as sm
import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
```

```
2023-01-03 08:47:29.421779: I tensorflow/core/platform/cpu_feature_guard.cc:193] This Te
nsorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler fla
gs.
Tensor Flow Version: 2.10.0
Keras Version: 2.10.0

Python 3.8.15 (default, Nov 24 2022, 09:04:07)
[Clang 14.0.6 ]
Pandas 1.5.2
Scikit-Learn 1.1.3
```

Define the project and research

EU Agricultural Price Indices (API)

How to see Ireland with respect to its EU community partners?

(CRISP-DM Phase: Business/ Research Understanding Phase)

Based on the data provided by the European Union Eurostat it would be comparing the performance of agriculture in Ireland with their neighbours and partners of the European Union based on two indicators: The index of price and the Index of expenditure to produce the products. Also, it will be considered the value of gross domestic product GDP which is one of the principal factors in the index of price.

In this project, it will be introduced one **sentimental feature** which indicates the opinion of the expert about the GDP, whether is positive or negative the economy of the countries. For example, according to the experts if the GDP is higher consequently inflation is increasing, therefore, the index of expenditure (cost to produce the products) and the index of price increase increases, which means that a very higher GDP it is not desirable for the economy.

Following the Cross Industry Standard Process **CRISP-DM**, https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining, the phases and plan of the project are available here:

https://github.com/users/sba22223nestorpereira/projects/1

Justification, Please see:

https://www.investopedia.com/articles/06/gdpinflation.asp

https://www.imf.org/en/Publications/fandd/issues/Series/Back-to-Basics/gross-domestic-product-GDP

Process of acquiring data (research)

(CRISP-DM Phase: Data Understanding Phase)

EU Agricultural Price Index (API)

How to see Ireland with respect to its EU community partners?

Function to read file excel downloaded from

https://ec.europa.eu/eurostat/web/agriculture/data/database

https://ec.europa.eu/eurostat/cache/metadata/en/apri_pi_esms.htm

An Agricultural Price Index shows how agricultural revenue (**output**) and expenditure (**input**) are influenced by their price component and is therefore connected with Economic Accounts for Agriculture (EAA).

The agricultural price indices may serve various purposes of economic analysis.

The EU Agricultural Price Indices (API) comprise:

1- the index of purchase prices of the means of agricultural production (**input**)

Index of variation of the **expenditure** incurred by farmers in purchasing the means of production (goods and services as well as investment goods), including crop products from other agricultural units for intermediate consumption, over a given period.

2- the index of producer prices of agricultural products (**output**)

Index of variation of prices reflecting **revenue** received by the producer for goods and services actually sold to customers over a period.

Index of expenditure (input) by period (ina)

Price indices of the means of agricultural production, input (2015 = 100) - annual data

Price indices of the means of agricultural production, input (2010 = 100) - annual data

Price indices of the means of agricultural production, input (2005 = 100) - annual data

Price indices of the means of agricultural production, input (2000 = 100) - annual data

Index of prices (output) by period (outa)

Price indices of agricultural products, output (2015 = 100) - annual data

Price indices of agricultural products, output (2010 = 100) - annual data

Price indices of agricultural products, output (2005 = 100) - annual data

Price indices of agricultural products, output (2000 = 100) - annual data

The **input price** indices cover agricultural inputs including intermediate consumption of goods and services (fertilisers, pesticides, feed, seed, energy and lubricants, maintenance and repairs, etc.) and gross fixed capital formation related to investments goods (machinery and equipment, farms, buildings, etc.)

The **output price** indices cover agricultural goods and services. They include crops, livestock and livestock products. The producer prices index of agricultural products (output) represents the measure of transaction prices reflecting **revenue** received by the producer for goods and services actually sold to customers over a period.

Observation about the base price by year

(2015 = 100) indicate the base price of an index is 100 by 2015 (2015-2021)

(2010 = 100) indicate the base price of an index is 100 by 2010 (2010-2017)

(2005 = 100) indicate the base price of an index is 100 by 2005 (2005-2012)

(2000 = 100) indicate the base price of an index is 100 by 2000 (2000-2008)

Important: website https://ec.europa.eu/eurostat/web/main/home does not allow reading directly from the website because it's a **web application** in which needs to **choose an option** before downloading the excel.

Data: https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/tree/data

GDP - Gross domestic product on output, expenditure and income

The four components of gross domestic product are personal consumption, business investment, government spending, and net exports.

All those indexes are impacted by other economical factors but in particular by the GDP - Gross domestic product on output, expenditure and income.

Eurostat publishes annual and quarterly national accounts use and input-output tables, which are each presented with associated metadata with the index of prices: this is a index of **GDP and main components (output, expenditure and income)**.

Data are available from 2010 in Eurostat.

In order to maintain the consistency and coherence of the data in this project, its development a **second part of the analysis from 2010 to 2021**.

https://ec.europa.eu/eurostat/cache/metadata/en/namq_10_esms.htm

https://www.thebalancemoney.com/components-of-gdp-explanation-formula-and-chart-3306015

Sentimental Categorical features

Finally, it will be added to the data, characteristics (Sentimental Categorical features) based on the opinion of the expert in GDP related when the GDP is negative or positive.

Most economists today agree that a small amount of inflation about 1% to 2% is beneficial, and is essential that the GDP of the countries needs to grow. However, if GDP growth is higher than 2.5% to 3.5% could be dangerous, because causes inflation or even worse hyperinflation.

This economic parameter is essential in the index of producer prices of agricultural products (output) and the index of purchase prices of agricultural production (input) for Ireland and all the countries of the EU.

Therefore, **GDP between 0% to 3.5%** could be considered **"positive"**, in another way, out of this range, could be considered **"negative"**.

This **rule will be applied** to this project.

Justification, Please see:

https://www.imf.org/en/Publications/fandd/issues/Series/Back-to-Basics/gross-domestic-product-GDP

https://www.investopedia.com/articles/06/gdpinflation.asp

https://www.investopedia.com/terms/f/farmprices.asp

https://www.kaggle.com/code/kirolosatef/stock-prediction-using-twitter-sentiment-analysis#Load-the-dataset

.

Function to read file excel downloaded from index of prices (Input and Output) and fixed column names

1- read excel from https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/tree/data

2- delete row unnecessaries (bottom of the original excel that does NOT contain relevant data)

3- fixing columns name (years)

4- convert to numerical all values of price indices

.

In [4]:
```
# function to read file excel downloaded from index of prices input and output

# https://ec.europa.eu/eurostat/web/agriculture/data/database


def readexcel(df, column_fix, readexcel_name):

    # link to GitHub
    link = readexcel_name
    print(link)
    # to read just one sheet to dataframe:
    df = pd.read_excel(link,'Sheet 1')

    # Cleaning and fixing columns
    # delete row innecesaries (headers of the original excel that do not contain relevan

    df.drop(df.index[0:8], inplace=True)
    #df.drop(df.index[-8:], inplace=True)
    column = df.iloc[0].values.tolist()
    df.columns = column
    df = df[df.columns.dropna()]
    df.iloc[0:2]
    df.drop(df.index[0:2], inplace=True)

    # Fixing the columns names
    #column = ['Geo', '2015', '2016', '2017', '2018', '2019', '2020', '2021']
    df.columns = column_fix

    # Fixing the value of standard columns
    df['Geo'].iloc[0] = 'European Union: 27 countries'
    df['Geo'] = df['Geo'].replace('Germany (until 1990 former territory of the FRG)', 'G

    # convert to numerical, objects values

    df.loc[:, df.columns != 'Geo'] = df.loc[:, df.columns != 'Geo'].apply(pd.to_numeric,
    # use this option to convert "special" characters to NaN
    # invalid parsing will be set as NaN
    df = df.apply(pd.to_numeric, errors='ignore')
    # Convert all columns that can be converted into float
    # Error were raised because their type was Object

    return df

#df = df[df.columns.drop(list(df.filter(regex='Unnamed:')))]
```

Index of prices expenditure (input) by period

(CRISP-DM Phase: Data Understanding Phase)

Read data of Index of prices (input) by period

Period 2015: df_ina_2015

Read data Index of prices or expenditure (input) by period 2015

In [5]:

```python
# read data Index of prices (input) by period 2015

readexcel_name = "https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw


df_ina_2015 = pd.DataFrame()

# columns specific for df_ina_2015

# Fixing the columns names
column_fix = ['Geo', '2015', '2016', '2017', '2018', '2019', '2020', '2021']



df_ina_2015 = readexcel(df_ina_2015, column_fix, readexcel_name)


# Cleaning and fixing columns 2015

# this is specific for each excel
df_ina_2015.drop(df_ina_2015.index[-6:], inplace=True)


df_ina_2015
```

```
https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw/data/apri_pi15_in
a_2015.xlsx
```

Out[5]:

	Geo	2015	2016	2017	2018	2019	2020	2021
10	European Union: 27 countries	100.0	97.84	97.95	99.60	99.68	98.11	105.03
11	Belgium	100.0	97.09	98.35	99.59	100.08	98.22	107.70
12	Bulgaria	100.0	97.99	98.35	99.63	98.96	94.95	102.20
13	Czechia	100.0	96.23	95.05	94.96	95.18	92.46	94.35
14	Denmark	100.0	100.22	99.86	101.33	101.72	100.83	105.18
15	Germany	100.0	97.91	97.94	99.81	100.09	99.72	104.21
16	Estonia	100.0	97.06	94.44	94.17	93.78	93.11	95.73
17	Ireland	100.0	98.53	98.50	101.71	103.00	101.40	107.08
18	Greece	100.0	98.10	99.43	100.84	100.36	98.76	105.40
19	Spain	100.0	97.06	95.54	97.25	97.57	95.59	104.76
20	France	100.0	97.20	97.26	98.69	98.95	96.95	103.55
21	Croatia	100.0	95.49	93.89	94.85	94.57	91.93	105.52
22	Italy	100.0	100.00	99.70	101.76	102.62	103.20	108.57
23	Cyprus	100.0	95.24	95.97	93.16	95.40	94.51	103.91
24	Latvia	100.0	98.18	96.35	97.85	96.19	95.29	98.28
25	Lithuania	100.0	98.64	98.10	95.17	85.99	84.02	95.05
26	Luxembourg	100.0	98.42	98.18	98.72	98.96	98.77	104.53
27	Hungary	100.0	97.36	95.20	97.41	98.03	95.89	106.69
28	Malta	100.0	99.10	97.50	96.96	97.32	97.37	103.93
29	Netherlands	100.0	96.79	98.52	99.59	97.66	95.18	106.93

	Geo	2010	2011	2012	2013	2014	2015	2016
30	Austria	100.0	98.25	96.98	98.09	97.78	96.43	100.39
31	Poland	100.0	98.24	98.66	100.99	102.18	97.68	104.59
32	Portugal	100.0	98.88	97.04	97.50	97.77	97.73	109.22
33	Romania	100.0	96.19	101.66	103.23	101.77	100.08	107.23
34	Slovenia	100.0	98.50	97.80	99.55	100.49	99.36	106.88
35	Slovakia	100.0	95.80	94.14	96.70	96.12	90.81	95.70
36	Finland	100.0	97.04	97.73	100.21	101.10	97.13	105.16
37	Sweden	100.0	97.88	98.52	103.22	104.55	101.18	107.61

In [ ]:

Period 2010: df_ina_2010

Cleaning and fixing columns: data Index of prices or expenditure (input) by period 2010

In [6]:

```
# read data Index of prices (input) by period 2010

readexcel_name = "https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw


df_ina_2010 = pd.DataFrame()

# columns specific for df_ina_2010

# Fixing the columns names
column_fix = ['Geo', '2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017']


df_ina_2010 = readexcel(df_ina_2010, column_fix, readexcel_name)


# this is specific for each excel
df_ina_2010.drop(df_ina_2010.index[-5:], inplace=True)


df_ina_2010
```

```
https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw/data/apri_pi10_in
a_2010.xlsx
```

Out[6]:

	Geo	2010	2011	2012	2013	2014	2015	2016	2017
10	European Union: 27 countries	100.0	106.7	108.5	108.4	104.8	102.7	100.2	100.3
11	Belgium	100.0	107.6	111.1	110.2	101.7	99.3	96.8	97.8
12	Bulgaria	100.0	106.9	110.5	109.0	106.0	103.4	100.4	100.6
13	Czechia	100.0	105.9	106.4	108.1	106.0	103.0	98.8	97.8
14	Denmark	100.0	106.3	108.9	112.4	111.6	109.5	109.2	110.1
15	Germany	100.0	108.1	110.8	111.1	106.5	104.9	102.3	102.1
16	Ireland	100.0	108.2	111.1	113.4	108.8	106.6	104.0	103.2
17	Greece	100.0	105.9	107.0	107.3	106.0	104.8	103.1	104.6
18	Spain	100.0	107.3	110.3	108.7	105.3	105.0	102.3	100.8
19	France	100.0	106.2	107.3	107.6	104.6	102.3	99.7	99.5

20	Croatia	100.0	109.8	111.7	108.5	99.2	96.1	92.0	90.4
21	Italy	100.0	103.9	105.5	106.3	104.2	101.0	100.6	100.6
22	Cyprus	100.0	95.4	96.4	106.7	107.3	110.3	105.9	107.1
23	Latvia	100.0	107.1	108.9	109.3	106.4	104.4	102.5	100.4
24	Lithuania	100.0	114.4	120.3	114.8	109.1	112.3	100.3	95.3
25	Luxembourg	100.0	104.5	106.0	104.5	102.6	100.7	98.9	98.5
26	Hungary	100.0	108.1	108.9	109.3	106.2	104.9	102.8	100.7
27	Malta	100.0	107.2	108.8	108.8	104.8	102.3	100.7	99.9
28	Netherlands	100.0	107.2	107.3	107.1	101.9	100.1	96.1	98.0
29	Austria	100.0	103.3	105.1	104.9	102.9	101.8	100.6	100.3
30	Poland	100.0	106.3	109.8	109.4	107.0	104.4	102.8	103.4
31	Portugal	100.0	106.2	108.6	111.0	107.7	105.2	103.7	102.1
32	Romania	100.0	106.2	109.7	109.4	104.9	101.1	98.4	100.3
33	Slovenia	100.0	108.3	109.8	110.2	105.8	103.6	101.8	101.2
34	Slovakia	100.0	109.4	109.4	107.9	101.1	95.6	91.7	90.9
35	Finland	100.0	108.0	108.5	108.2	105.3	103.7	100.5	101.3
36	Sweden	100.0	105.2	106.2	106.1	105.8	105.6	103.1	103.7

In [7]:

```
df_ina_2010.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27 entries, 10 to 36
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Geo     27 non-null     object 
 1   2010    27 non-null     float64
 2   2011    27 non-null     float64
 3   2012    27 non-null     float64
 4   2013    27 non-null     float64
 5   2014    27 non-null     float64
 6   2015    27 non-null     float64
 7   2016    27 non-null     float64
 8   2017    27 non-null     float64
dtypes: float64(8), object(1)
memory usage: 2.0+ KB
```

Period 2005: df_ina_2005

Cleaning and fixing columns: data Index of prices or expenditure (input) by period 2005

In [8]:

```
# read data Index of prices (input) by period 2005

readexcel_name = "https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw

df_ina_2005 = pd.DataFrame()


# columns specific for df_ina_2005

# Fixing the columns names
column_fix = ['Geo', '2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012']
```

```python
df_ina_2005 = readexcel(df_ina_2005, column_fix, readexcel_name)


# Cleaning and fixing columns 2015

# this is specific for each excel
df_ina_2005.drop(df_ina_2005.index[-5:], inplace=True)


df_ina_2005
```

https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw/data/apri_pi05_ina_2005.xlsx

Out[8]:

	Geo	2005	2006	2007	2008	2009	2010	2011	2012
10	European Union: 27 countries	100.0	101.2	106.1	116.8	108.7	107.7	114.9	116.4
11	Belgium	100.0	103.1	111.7	122.3	108.8	104.8	113.6	116.4
12	Bulgaria	100.0	95.9	98.7	100.6	98.9	97.9	104.6	108.4
13	Czechia	100.0	98.6	101.1	104.1	96.6	93.8	98.5	98.8
14	Denmark	100.0	100.6	107.0	121.1	111.0	108.7	117.9	119.9
15	Germany	100.0	102.2	107.6	118.3	112.3	111.8	120.2	123.0
16	Estonia	100.0	101.5	103.6	103.4	95.6	94.8	100.8	100.6
17	Ireland	100.0	100.5	103.8	115.4	109.1	109.3	117.8	120.1
18	Greece	100.0	100.5	103.9	110.6	102.7	102.1	107.8	109.6
19	Spain	100.0	99.7	104.4	115.3	104.8	104.7	112.2	115.0
20	France	100.0	100.9	105.1	119.4	109.2	106.7	113.4	114.9
21	Croatia	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
22	Italy	100.0	101.2	105.8	114.4	110.2	111.0	114.7	115.8
23	Cyprus	100.0	104.9	110.7	115.4	101.7	97.2	94.0	92.9
24	Latvia	100.0	102.8	106.2	109.3	98.3	97.1	102.7	103.8
25	Lithuania	100.0	111.3	113.4	131.6	94.5	93.3	109.3	115.1
26	Luxembourg	100.0	99.6	103.1	108.5	103.1	102.8	106.9	108.2
27	Hungary	100.0	101.9	105.6	115.3	104.4	104.1	113.1	114.3
28	Malta	100.0	100.7	105.3	119.4	110.3	110.6	120.2	122.4
29	Netherlands	100.0	104.4	110.5	116.5	105.9	107.8	115.8	116.4
30	Austria	100.0	100.7	104.2	110.1	106.9	106.9	110.3	112.7
31	Poland	100.0	99.0	102.7	109.4	107.1	105.8	111.8	115.4
32	Portugal	100.0	99.7	104.9	116.5	114.8	113.8	117.6	119.0
33	Romania	100.0	NaN	NaN	112.2	101.8	104.2	115.2	118.8
34	Slovenia	100.0	100.8	105.5	117.9	110.6	110.0	119.7	121.4
35	Slovakia	100.0	99.2	101.9	107.2	93.1	93.3	99.5	98.5
36	Finland	100.0	102.3	105.4	117.3	106.0	106.9	116.1	116.8
37	Sweden	100.0	101.6	106.4	117.4	111.8	108.6	114.2	115.9

In [ ]:

Period 2000: df_ina_2000

Cleaning and fixing columns: data Index of prices or expenditure (input) by period 2000 (means based from 2000-2008)

In [9]:

```
# read data Index of prices (input) by period 2000

readexcel_name = "https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw

df_ina_2000 = pd.DataFrame()

# columns specific for df_ina_2000

# Fixing the columns names
column_fix = ['Geo', '2000', '2001', '2002', '2003', '2004', '2005', '2006', '2007', '20


df_ina_2000 = readexcel(df_ina_2000, column_fix, readexcel_name)


# Cleaning and fixing columns df_ina_2000

# this is specific for each excel
df_ina_2000.drop(df_ina_2000.index[-5:], inplace=True)


df_ina_2000
```

```
https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw/data/apri_pi00_in
a_2000.xlsx
```

Out[9]:

	Geo	2000	2001	2002	2003	2004	2005	2006	2007	2008
10	European Union: 27 countries	100.0	101.4	99.7	99.4	101.5	100.7	101.5	106.4	115.0
11	Belgium	100.0	100.2	99.1	97.7	96.2	97.6	97.9	106.1	116.5
12	Bulgaria	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
13	Czechia	100.0	100.1	97.4	96.0	99.8	97.7	96.4	99.7	104.2
14	Denmark	100.0	103.5	102.2	99.0	101.2	101.0	100.9	107.8	124.1
15	Germany	100.0	102.1	100.3	99.5	101.3	99.8	100.9	105.7	99.8
16	Estonia	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
17	Ireland	100.0	100.4	97.5	96.0	97.1	98.9	100.1	103.2	116.1
18	Greece	100.0	98.4	97.4	97.9	102.2	103.8	104.5	107.8	116.0
19	Spain	100.0	100.0	97.4	95.7	96.5	95.1	94.9	97.6	110.3
20	France	100.0	101.3	99.9	99.0	100.3	100.3	101.2	105.2	115.0
21	Italy	100.0	102.1	100.7	100.8	103.4	99.1	100.0	104.8	112.4
22	Cyprus	100.0	NaN	NaN	NaN	128.6	136.0	135.8	145.7	161.9
23	Latvia	100.0	99.2	97.9	99.1	100.9	111.3	113.5	115.3	120.5
24	Lithuania	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	120.4
25	Luxembourg	100.0	101.1	100.3	99.1	100.2	97.7	94.9	98.4	105.4
26	Hungary	100.0	102.5	98.3	99.7	100.8	97.4	99.0	104.6	112.7
27	Malta	100.0	98.3	96.7	90.7	93.2	93.8	93.8	99.0	113.2

28	Netherlands	100.0	100.9	98.2	97.7	97.9	97.9	101.9	108.1	114.8
29	Austria	100.0	99.6	97.6	98.2	99.6	98.8	99.9	104.0	109.5
30	Poland	100.0	101.2	101.6	103.9	107.8	108.0	107.0	112.6	120.5
31	Portugal	100.0	100.1	96.4	94.5	96.1	97.3	95.8	100.1	107.2
32	Romania	100.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
33	Slovenia	100.0	103.1	98.9	98.1	103.0	101.9	102.9	108.3	121.8
34	Slovakia	100.0	NaN	NaN	NaN	89.2	87.5	88.4	91.2	97.5
35	Finland	100.0	99.6	98.2	98.2	100.8	103.5	107.1	115.5	121.9
36	Sweden	100.0	102.3	102.3	102.1	104.8	106.0	107.5	113.3	125.8

In [ ]:

Join the data of Index of prices or expenditure (input) and create NEW DF by period 2000 to 2021

Groups and join the data frames: df_ina_2015, df_ina_2010, df_ina_2005, and df_ina_2000 in order to create a DF from all periods from 2000 to 2021.

Join: it is a inner join in which the "priority" is the newest DF because has the most recent calculation of the index of prices, means from df_ina_2015.

https://pandas.pydata.org/docs/user_guide/merging.html

Important: Joins will be set up in several steps in order to bring clarity and a better understanding of the code.

In [10]:

```
# Step 1: Join df_ina_2015 with df_ina_2010
df_ina = pd.merge(df_ina_2010[['Geo', '2011', '2012', '2013', '2014', '2015']],
         df_ina_2015[['Geo', '2016', '2017', '2018', '2019', '2020', '2021']],
         how='right',
         on = 'Geo')
df_ina
```

Out[10]:

	Geo	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
0	European Union: 27 countries	106.7	108.5	108.4	104.8	102.7	97.84	97.95	99.60	99.68	98.11	105.03
1	Belgium	107.6	111.1	110.2	101.7	99.3	97.09	98.35	99.59	100.08	98.22	107.70
2	Bulgaria	106.9	110.5	109.0	106.0	103.4	97.99	98.35	99.63	98.96	94.95	102.20
3	Czechia	105.9	106.4	108.1	106.0	103.0	96.23	95.05	94.96	95.18	92.46	94.35
4	Denmark	106.3	108.9	112.4	111.6	109.5	100.22	99.86	101.33	101.72	100.83	105.18
5	Germany	108.1	110.8	111.1	106.5	104.9	97.91	97.94	99.81	100.09	99.72	104.21
6	Estonia	NaN	NaN	NaN	NaN	NaN	97.06	94.44	94.17	93.78	93.11	95.73
7	Ireland	108.2	111.1	113.4	108.8	106.6	98.53	98.50	101.71	103.00	101.40	107.08
8	Greece	105.9	107.0	107.3	106.0	104.8	98.10	99.43	100.84	100.36	98.76	105.40
9	Spain	107.3	110.3	108.7	105.3	105.0	97.06	95.54	97.25	97.57	95.59	104.76

10	France	106.2	107.3	107.6	104.6	102.3	97.20	97.26	98.69	98.95	96.95	103.55
11	Croatia	109.8	111.7	108.5	99.2	96.1	95.49	93.89	94.85	94.57	91.93	105.52
12	Italy	103.9	105.5	106.3	104.2	101.0	100.00	99.70	101.76	102.62	103.20	108.57
13	Cyprus	95.4	96.4	106.7	107.3	110.3	95.24	95.97	93.16	95.40	94.51	103.91
14	Latvia	107.1	108.9	109.3	106.4	104.4	98.18	96.35	97.85	96.19	95.29	98.28
15	Lithuania	114.4	120.3	114.8	109.1	112.3	98.64	98.10	95.17	85.99	84.02	95.05
16	Luxembourg	104.5	106.0	104.5	102.6	100.7	98.42	98.18	98.72	98.96	98.77	104.53
17	Hungary	108.1	108.9	109.3	106.2	104.9	97.36	95.20	97.41	98.03	95.89	106.69
18	Malta	107.2	108.8	108.8	104.8	102.3	99.10	97.50	96.96	97.32	97.37	103.93
19	Netherlands	107.2	107.3	107.1	101.9	100.1	96.79	98.52	99.59	97.66	95.18	106.93
20	Austria	103.3	105.1	104.9	102.9	101.8	98.25	96.98	98.09	97.78	96.43	100.39
21	Poland	106.3	109.8	109.4	107.0	104.4	98.24	98.66	100.99	102.18	97.68	104.59
22	Portugal	106.2	108.6	111.0	107.7	105.2	98.88	97.04	97.50	97.77	97.73	109.22
23	Romania	106.2	109.7	109.4	104.9	101.1	96.19	101.66	103.23	101.77	100.08	107.23
24	Slovenia	108.3	109.8	110.2	105.8	103.6	98.50	97.80	99.55	100.49	99.36	106.88
25	Slovakia	109.4	109.4	107.9	101.1	95.6	95.80	94.14	96.70	96.12	90.81	95.70
26	Finland	108.0	108.5	108.2	105.3	103.7	97.04	97.73	100.21	101.10	97.13	105.16
27	Sweden	105.2	106.2	106.1	105.8	105.6	97.88	98.52	103.22	104.55	101.18	107.61

In [11]:

```
# Step 2: Join previous result with df_ina_2005
df_ina = pd.merge(df_ina_2005[['Geo', '2006', '2007', '2008', '2009', '2010']],
         df_ina,
         how='right',
         on = 'Geo')
df_ina
```

Out[11]:

	Geo	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018
0	European Union: 27 countries	101.2	106.1	116.8	108.7	107.7	106.7	108.5	108.4	104.8	102.7	97.84	97.95	99.6
1	Belgium	103.1	111.7	122.3	108.8	104.8	107.6	111.1	110.2	101.7	99.3	97.09	98.35	99.5
2	Bulgaria	95.9	98.7	100.6	98.9	97.9	106.9	110.5	109.0	106.0	103.4	97.99	98.35	99.6
3	Czechia	98.6	101.1	104.1	96.6	93.8	105.9	106.4	108.1	106.0	103.0	96.23	95.05	94.9
4	Denmark	100.6	107.0	121.1	111.0	108.7	106.3	108.9	112.4	111.6	109.5	100.22	99.86	101.3
5	Germany	102.2	107.6	118.3	112.3	111.8	108.1	110.8	111.1	106.5	104.9	97.91	97.94	99.8
6	Estonia	101.5	103.6	103.4	95.6	94.8	NaN	NaN	NaN	NaN	NaN	97.06	94.44	94.1
7	Ireland	100.5	103.8	115.4	109.1	109.3	108.2	111.1	113.4	108.8	106.6	98.53	98.50	101.7
8	Greece	100.5	103.9	110.6	102.7	102.1	105.9	107.0	107.3	106.0	104.8	98.10	99.43	100.8
9	Spain	99.7	104.4	115.3	104.8	104.7	107.3	110.3	108.7	105.3	105.0	97.06	95.54	97.2
10	France	100.9	105.1	119.4	109.2	106.7	106.2	107.3	107.6	104.6	102.3	97.20	97.26	98.6
11	Croatia	NaN	NaN	NaN	NaN	NaN	109.8	111.7	108.5	99.2	96.1	95.49	93.89	94.8
12	Italy	101.2	105.8	114.4	110.2	111.0	103.9	105.5	106.3	104.2	101.0	100.00	99.70	101.7
13	Cyprus	104.9	110.7	115.4	101.7	97.2	95.4	96.4	106.7	107.3	110.3	95.24	95.97	93.1

14	Latvia	102.8	106.2	109.3	98.3	97.1	107.1	108.9	109.3	106.4	104.4	98.18	96.35	97.8
15	Lithuania	111.3	113.4	131.6	94.5	93.3	114.4	120.3	114.8	109.1	112.3	98.64	98.10	95.1
16	Luxembourg	99.6	103.1	108.5	103.1	102.8	104.5	106.0	104.5	102.6	100.7	98.42	98.18	98.7
17	Hungary	101.9	105.6	115.3	104.4	104.1	108.1	108.9	109.3	106.2	104.9	97.36	95.20	97.4
18	Malta	100.7	105.3	119.4	110.3	110.6	107.2	108.8	108.8	104.8	102.3	99.10	97.50	96.9
19	Netherlands	104.4	110.5	116.5	105.9	107.8	107.2	107.3	107.1	101.9	100.1	96.79	98.52	99.5
20	Austria	100.7	104.2	110.1	106.9	106.9	103.3	105.1	104.9	102.9	101.8	98.25	96.98	98.0
21	Poland	99.0	102.7	109.4	107.1	105.8	106.3	109.8	109.4	107.0	104.4	98.24	98.66	100.9
22	Portugal	99.7	104.9	116.5	114.8	113.8	106.2	108.6	111.0	107.7	105.2	98.88	97.04	97.5
23	Romania	NaN	NaN	112.2	101.8	104.2	106.2	109.7	109.4	104.9	101.1	96.19	101.66	103.2
24	Slovenia	100.8	105.5	117.9	110.6	110.0	108.3	109.8	110.2	105.8	103.6	98.50	97.80	99.5
25	Slovakia	99.2	101.9	107.2	93.1	93.3	109.4	109.4	107.9	101.1	95.6	95.80	94.14	96.7
26	Finland	102.3	105.4	117.3	106.0	106.9	108.0	108.5	108.2	105.3	103.7	97.04	97.73	100.2
27	Sweden	101.6	106.4	117.4	111.8	108.6	105.2	106.2	106.1	105.8	105.6	97.88	98.52	103.2

In [12]:
```python
# Step 3: Join previous result with df_ina_2000

# Using right join because df_ina_2000 does not has data from Croatia

df_ina = pd.merge(df_ina_2000[['Geo', '2000', '2001', '2002', '2003', '2004', '2005']],
         df_ina,
         how='right', # using right join
         on = 'Geo')
df_ina
```

Out[12]:

	Geo	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	2012	2013	2014	20
0	European Union: 27 countries	100.0	101.4	99.7	99.4	101.5	100.7	101.2	106.1	116.8	...	108.5	108.4	104.8	10
1	Belgium	100.0	100.2	99.1	97.7	96.2	97.6	103.1	111.7	122.3	...	111.1	110.2	101.7	9
2	Bulgaria	NaN	NaN	NaN	NaN	NaN	NaN	95.9	98.7	100.6	...	110.5	109.0	106.0	10
3	Czechia	100.0	100.1	97.4	96.0	99.8	97.7	98.6	101.1	104.1	...	106.4	108.1	106.0	10
4	Denmark	100.0	103.5	102.2	99.0	101.2	101.0	100.6	107.0	121.1	...	108.9	112.4	111.6	10
5	Germany	100.0	102.1	100.3	99.5	101.3	99.8	102.2	107.6	118.3	...	110.8	111.1	106.5	10
6	Estonia	NaN	NaN	NaN	NaN	NaN	NaN	101.5	103.6	103.4	...	NaN	NaN	NaN	N
7	Ireland	100.0	100.4	97.5	96.0	97.1	98.9	100.5	103.8	115.4	...	111.1	113.4	108.8	10
8	Greece	100.0	98.4	97.4	97.9	102.2	103.8	100.5	103.9	110.6	...	107.0	107.3	106.0	10
9	Spain	100.0	100.0	97.4	95.7	96.5	95.1	99.7	104.4	115.3	...	110.3	108.7	105.3	10
10	France	100.0	101.3	99.9	99.0	100.3	100.3	100.9	105.1	119.4	...	107.3	107.6	104.6	10
11	Croatia	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	111.7	108.5	99.2	9
12	Italy	100.0	102.1	100.7	100.8	103.4	99.1	101.2	105.8	114.4	...	105.5	106.3	104.2	10
13	Cyprus	100.0	NaN	NaN	NaN	128.6	136.0	104.9	110.7	115.4	...	96.4	106.7	107.3	11
14	Latvia	100.0	99.2	97.9	99.1	100.9	111.3	102.8	106.2	109.3	...	108.9	109.3	106.4	10
15	Lithuania	NaN	NaN	NaN	NaN	NaN	NaN	111.3	113.4	131.6	...	120.3	114.8	109.1	11

16	Luxembourg	100.0	101.1	100.3	99.1	100.2	97.7	99.6	103.1	108.5	...	106.0	104.5	102.6	100
17	Hungary	100.0	102.5	98.3	99.7	100.8	97.4	101.9	105.6	115.3	...	108.9	109.3	106.2	104
18	Malta	100.0	98.3	96.7	90.7	93.2	93.8	100.7	105.3	119.4	...	108.8	108.8	104.8	102
19	Netherlands	100.0	100.9	98.2	97.7	97.9	97.9	104.4	110.5	116.5	...	107.3	107.1	101.9	100
20	Austria	100.0	99.6	97.6	98.2	99.6	98.8	100.7	104.2	110.1	...	105.1	104.9	102.9	101
21	Poland	100.0	101.2	101.6	103.9	107.8	108.0	99.0	102.7	109.4	...	109.8	109.4	107.0	104
22	Portugal	100.0	100.1	96.4	94.5	96.1	97.3	99.7	104.9	116.5	...	108.6	111.0	107.7	105
23	Romania	100.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	112.2	...	109.7	109.4	104.9	10
24	Slovenia	100.0	103.1	98.9	98.1	103.0	101.9	100.8	105.5	117.9	...	109.8	110.2	105.8	103
25	Slovakia	100.0	NaN	NaN	NaN	89.2	87.5	99.2	101.9	107.2	...	109.4	107.9	101.1	95
26	Finland	100.0	99.6	98.2	98.2	100.8	103.5	102.3	105.4	117.3	...	108.5	108.2	105.3	103
27	Sweden	100.0	102.3	102.3	102.1	104.8	106.0	101.6	106.4	117.4	...	106.2	106.1	105.8	105

28 rows × 23 columns

Data Wrangling

Data wrangling: cleaning, missing values and outliers

Missing values

Basically, the price index illustrates how the expenditure to produce the product or a basket of products has changed since the base period.

The **base price of an index is 100** by agreement (according to Eurostat), meaning that, for instance, an index equal to 110 reflects an increase in the absolute price of 10% and an index equal to 95 a decrease of 5%.

Please see: https://ec.europa.eu/eurostat/cache/metadata/en/apri_pi_esms.htm

This value: 100, will be considered in order to fix the missing values, meaning that any **missing value will be substituted by the base price** instead of the mean or median as conventionally used.

In [13]:
```
# Using Pandas

base_price=100.0  # base price of an index is 100 by agreement

df_ina = df_ina.fillna(base_price)

df_ina
```

Out[13]:

	Geo	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	2012	2013	2014	20
0	European Union: 27 countries	100.0	101.4	99.7	99.4	101.5	100.7	101.2	106.1	116.8	...	108.5	108.4	104.8	102

1	Belgium	100.0	100.2	99.1	97.7	96.2	97.6	103.1	111.7	122.3	...	111.1	110.2	101.7	99
2	Bulgaria	100.0	100.0	100.0	100.0	100.0	100.0	95.9	98.7	100.6	...	110.5	109.0	106.0	103
3	Czechia	100.0	100.1	97.4	96.0	99.8	97.7	98.6	101.1	104.1	...	106.4	108.1	106.0	103
4	Denmark	100.0	103.5	102.2	99.0	101.2	101.0	100.6	107.0	121.1	...	108.9	112.4	111.6	109
5	Germany	100.0	102.1	100.3	99.5	101.3	99.8	102.2	107.6	118.3	...	110.8	111.1	106.5	104
6	Estonia	100.0	100.0	100.0	100.0	100.0	100.0	101.5	103.6	103.4	...	100.0	100.0	100.0	100
7	Ireland	100.0	100.4	97.5	96.0	97.1	98.9	100.5	103.8	115.4	...	111.1	113.4	108.8	106
8	Greece	100.0	98.4	97.4	97.9	102.2	103.8	100.5	103.9	110.6	...	107.0	107.3	106.0	104
9	Spain	100.0	100.0	97.4	95.7	96.5	95.1	99.7	104.4	115.3	...	110.3	108.7	105.3	105
10	France	100.0	101.3	99.9	99.0	100.3	100.3	100.9	105.1	119.4	...	107.3	107.6	104.6	102
11	Croatia	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	...	111.7	108.5	99.2	9
12	Italy	100.0	102.1	100.7	100.8	103.4	99.1	101.2	105.8	114.4	...	105.5	106.3	104.2	10
13	Cyprus	100.0	100.0	100.0	100.0	128.6	136.0	104.9	110.7	115.4	...	96.4	106.7	107.3	110
14	Latvia	100.0	99.2	97.9	99.1	100.9	111.3	102.8	106.2	109.3	...	108.9	109.3	106.4	104
15	Lithuania	100.0	100.0	100.0	100.0	100.0	100.0	111.3	113.4	131.6	...	120.3	114.8	109.1	112
16	Luxembourg	100.0	101.1	100.3	99.1	100.2	97.7	99.6	103.1	108.5	...	106.0	104.5	102.6	100
17	Hungary	100.0	102.5	98.3	99.7	100.8	97.4	101.9	105.6	115.3	...	108.9	109.3	106.2	104
18	Malta	100.0	98.3	96.7	90.7	93.2	93.8	100.7	105.3	119.4	...	108.8	108.8	104.8	102
19	Netherlands	100.0	100.9	98.2	97.7	97.9	97.9	104.4	110.5	116.5	...	107.3	107.1	101.9	10
20	Austria	100.0	99.6	97.6	98.2	99.6	98.8	100.7	104.2	110.1	...	105.1	104.9	102.9	10
21	Poland	100.0	101.2	101.6	103.9	107.8	108.0	99.0	102.7	109.4	...	109.8	109.4	107.0	104
22	Portugal	100.0	100.1	96.4	94.5	96.1	97.3	99.7	104.9	116.5	...	108.6	111.0	107.7	105
23	Romania	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	112.2	...	109.7	109.4	104.9	10
24	Slovenia	100.0	103.1	98.9	98.1	103.0	101.9	100.8	105.5	117.9	...	109.8	110.2	105.8	103
25	Slovakia	100.0	100.0	100.0	100.0	89.2	87.5	99.2	101.9	107.2	...	109.4	107.9	101.1	9
26	Finland	100.0	99.6	98.2	98.2	100.8	103.5	102.3	105.4	117.3	...	108.5	108.2	105.3	103
27	Sweden	100.0	102.3	102.3	102.1	104.8	106.0	101.6	106.4	117.4	...	106.2	106.1	105.8	105

28 rows × 23 columns

Analysis Outliers Index of prices or expenditure (input): df_ina

In [14]:

```
# Analysis outliers

# DF will be melted in order to analyse the principal features of Index prices.
```

Tukey fence method

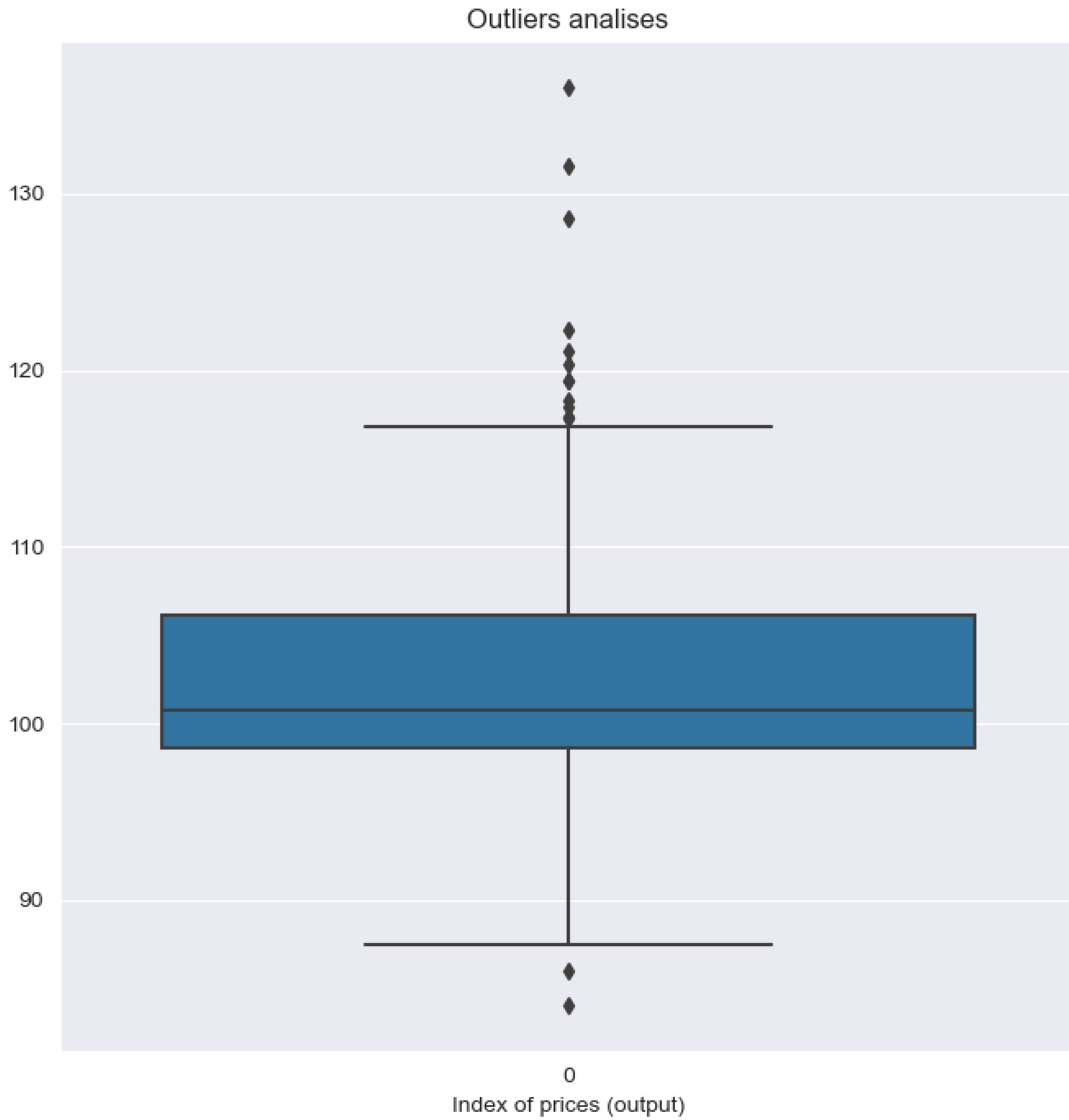
Tukey distinguishes between the inner and the outer fence.

A possible outlier is located between the inner and the outer fence, the strategy will be change outliers outside of the inner/outer fence.

The great advantage of Tukey's box plot method is that the statistics (e.g. IQR, inner and outer fence) are robust to outliers, meaning to find one outlier is independent of all other outliers. Furthermore, this method does not require a normal distribution of the data.

In [15]:
```
# melt_pivot

df_ina_index = df_ina.melt(id_vars=["Geo"],
                 var_name="Year",
                 value_name="Price_Index")
```

In [16]:
```
# visualize univariable outliers

fig = plt.figure(figsize=(8, 8))
ax = sns.boxplot(data=df_ina_index['Price_Index'])
ax.set_xlabel('Index of prices (output)')
plt.title('Outliers analises')
plt.show()
```



In [17]:
```
def tukeys_method(df, feature):
    # calculate Q1 and Q3
    q1 = df[feature].quantile(0.25)
    q3 = df[feature].quantile(0.75)
    print('Q1: ',q1)
    print('Q3: ',q3)
    iqr = q3-q1
    inner_fence = 1.5*iqr
    outer_fence = 3*iqr

    #inner fence lower and upper end corresponding with 1.5 IQR point
    inner_fence_le = q1-inner_fence
    inner_fence_ue = q3+inner_fence
    #print(inner_fence_ue)
    #outer fence lower and upper end corresponding with 3.0 IQR point
    outer_fence_le = q1-outer_fence
    outer_fence_ue = q3+outer_fence

    outliers_outer = []
    outliers_inner = []
    # outer fence
```

```
    for index, x in enumerate(df[feature]):
        if x <= outer_fence_le or x >= outer_fence_ue:
            outliers_outer.append(index)
    # inner fence
    for index, x in enumerate(df[feature]):
        if x <= inner_fence_le or x >= inner_fence_ue:
            outliers_inner.append(index)

    return outliers_outer, outliers_inner # return the index of the outliers in inner fe
```

In [18]:

```
# Search for tukey fence on the target feature df_index['Price_Index']
outliers_outer_indexes, outliers_inner_indexes = tukeys_method(df_ina_index, 'Price_Inde

print('\nOuter index:  ', outliers_outer_indexes)
print('\nInner index:  ', outliers_inner_indexes)
```

```
Q1:  98.655
Q3:  106.1

Outer index:   [125, 153, 239]

Inner index:   [125, 153, 225, 228, 229, 234, 239, 242, 248, 250, 251, 351, 547, 575]
```

It is just tree observations out of the Outer Fence according to the Tukey Methods.

For this project it will consider this has a **low impact on the samples, and in the model**, therefore, it will **not change** values to any observations.

In [19]:

```
# It is just two observations is out of the Outer Fence according to the Tukey Methodhs

df_ina_index.iloc[[125, 153, 239]]
```

Out[19]:

	Geo	Year	Price_Index
125	Cyprus	2004	128.6
153	Cyprus	2005	136.0
239	Lithuania	2008	131.6

End Index of prices (input)

Index of price (output) by period

(CRISP-DM Phase: Data Understanding Phase)

Read data of Index of price (output) by period

Period 2015: df_outa_2015

In [20]:

```
# read data Index of price (output)  by period 2015

readexcel_name = "https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw
```

```python
df_outa_2015 = pd.DataFrame()


# columns specific for df_outa_2015

# Fixing the columns names
column_fix = ['Geo', '2015', '2016', '2017', '2018', '2019', '2020', '2021']

df_outa_2015 = readexcel(df_outa_2015, column_fix, readexcel_name)


# Cleaning and fixing columns df_outa_2015

# this is specific for each excel
df_outa_2015.drop(df_outa_2015.index[-6:], inplace=True)


df_outa_2015
```

```
https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw/data/apri_pi15_ou
ta_2015.xlsx
```

Out[20]:

	Geo	2015	2016	2017	2018	2019	2020	2021
10	European Union: 27 countries	100.0	98.46	103.97	102.97	103.43	102.46	108.96
11	Belgium	100.0	101.66	104.84	110.28	108.05	103.92	111.13
12	Bulgaria	100.0	96.07	94.94	93.53	92.68	99.24	115.99
13	Czechia	100.0	93.55	98.16	96.29	98.61	92.28	95.48
14	Denmark	100.0	94.66	102.87	96.52	102.94	103.62	104.30
15	Germany	100.0	98.31	106.37	104.81	105.78	101.89	107.60
16	Estonia	100.0	96.60	106.56	105.24	104.47	101.62	113.17
17	Ireland	100.0	95.31	106.32	103.49	101.45	102.71	111.90
18	Greece	100.0	98.04	98.58	97.77	97.72	97.83	107.58
19	Spain	100.0	96.65	101.75	99.44	94.57	95.08	100.60
20	France	100.0	99.83	102.22	102.16	102.92	102.91	109.99
21	Croatia	100.0	98.56	101.70	99.52	99.47	99.92	109.58
22	Italy	100.0	97.10	103.65	103.41	103.88	104.95	111.81
23	Cyprus	100.0	99.66	101.49	99.80	105.05	102.20	96.66
24	Latvia	100.0	98.45	107.30	110.86	109.96	107.64	121.45
25	Lithuania	100.0	92.64	100.31	100.35	102.28	99.78	110.28
26	Luxembourg	100.0	98.00	107.11	102.02	102.24	101.02	102.37
27	Hungary	100.0	96.01	98.55	98.30	100.24	104.29	118.97
28	Malta	100.0	100.69	97.45	96.40	103.91	102.22	104.14
29	Netherlands	100.0	100.19	106.02	102.21	102.76	96.39	102.56
30	Austria	100.0	98.05	103.47	100.09	100.58	99.94	105.78
31	Poland	100.0	100.52	116.34	114.23	120.21	115.40	119.00
32	Portugal	100.0	101.77	102.75	104.01	104.42	104.93	109.83
33	Romania	100.0	101.80	102.83	104.74	110.86	116.55	123.11
34	Slovenia	100.0	98.34	105.43	102.84	104.91	103.08	109.37
35	Slovakia	100.0	102.32	105.90	97.72	96.81	95.29	104.06

36	Finland	100.0	96.63	98.65	100.84	99.55	96.91	101.74
37	Sweden	100.0	100.61	106.21	115.61	112.18	110.18	117.58

In [21]:
```
df_outa_2015
```

Out[21]:

	Geo	2015	2016	2017	2018	2019	2020	2021
10	European Union: 27 countries	100.0	98.46	103.97	102.97	103.43	102.46	108.96
11	Belgium	100.0	101.66	104.84	110.28	108.05	103.92	111.13
12	Bulgaria	100.0	96.07	94.94	93.53	92.68	99.24	115.99
13	Czechia	100.0	93.55	98.16	96.29	98.61	92.28	95.48
14	Denmark	100.0	94.66	102.87	96.52	102.94	103.62	104.30
15	Germany	100.0	98.31	106.37	104.81	105.78	101.89	107.60
16	Estonia	100.0	96.60	106.56	105.24	104.47	101.62	113.17
17	Ireland	100.0	95.31	106.32	103.49	101.45	102.71	111.90
18	Greece	100.0	98.04	98.58	97.77	97.72	97.83	107.58
19	Spain	100.0	96.65	101.75	99.44	94.57	95.08	100.60
20	France	100.0	99.83	102.22	102.16	102.92	102.91	109.99
21	Croatia	100.0	98.56	101.70	99.52	99.47	99.92	109.58
22	Italy	100.0	97.10	103.65	103.41	103.88	104.95	111.81
23	Cyprus	100.0	99.66	101.49	99.80	105.05	102.20	96.66
24	Latvia	100.0	98.45	107.30	110.86	109.96	107.64	121.45
25	Lithuania	100.0	92.64	100.31	100.35	102.28	99.78	110.28
26	Luxembourg	100.0	98.00	107.11	102.02	102.24	101.02	102.37
27	Hungary	100.0	96.01	98.55	98.30	100.24	104.29	118.97
28	Malta	100.0	100.69	97.45	96.40	103.91	102.22	104.14
29	Netherlands	100.0	100.19	106.02	102.21	102.76	96.39	102.56
30	Austria	100.0	98.05	103.47	100.09	100.58	99.94	105.78
31	Poland	100.0	100.52	116.34	114.23	120.21	115.40	119.00
32	Portugal	100.0	101.77	102.75	104.01	104.42	104.93	109.83
33	Romania	100.0	101.80	102.83	104.74	110.86	116.55	123.11
34	Slovenia	100.0	98.34	105.43	102.84	104.91	103.08	109.37
35	Slovakia	100.0	102.32	105.90	97.72	96.81	95.29	104.06
36	Finland	100.0	96.63	98.65	100.84	99.55	96.91	101.74
37	Sweden	100.0	100.61	106.21	115.61	112.18	110.18	117.58

Period 2010: df_outa_2010

In [22]:
```
# read data Index of price (output)  by period 2010

readexcel_name = "https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw
```

```python
df_outa_2010 = pd.DataFrame()


# columns specific for df_outa_2010

# Fixing the columns names
column_fix = ['Geo', '2010', '2011', '2012', '2013', '2014', '2015', '2016', '2017']

df_outa_2010 = readexcel(df_outa_2010, column_fix, readexcel_name)


# Cleaning and fixing columns df_outa_2010

# this is specific for each excel
df_outa_2010.drop(df_outa_2010.index[-6:], inplace=True)


df_outa_2010
```

```
https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw/data/apri_pi10_ou
ta_2010.xlsx
```

Out[22]:

	Geo	2010	2011	2012	2013	2014	2015	2016	2017
10	European Union: 27 countries	100.0	107.4	111.0	112.3	104.4	102.1	100.9	105.8
11	Belgium	100.0	98.7	108.3	111.7	92.6	88.0	89.4	92.3
12	Bulgaria	100.0	119.6	135.1	110.8	107.9	113.5	109.3	107.4
13	Czechia	100.0	118.8	119.1	122.9	117.7	110.3	104.2	109.9
14	Denmark	100.0	111.5	117.2	121.4	107.4	100.9	96.8	103.7
15	Germany	100.0	110.6	114.0	113.5	103.6	99.6	99.0	105.4
16	Ireland	100.0	113.9	117.5	126.9	116.2	111.6	106.4	116.5
17	Greece	100.0	99.9	97.5	100.0	99.6	105.5	104.0	104.0
18	Spain	100.0	97.5	105.9	107.5	99.7	106.7	103.6	109.6
19	France	100.0	110.0	113.6	115.1	109.4	105.2	106.2	107.8
20	Croatia	100.0	105.3	110.1	100.9	95.7	96.3	95.3	98.3
21	Italy	100.0	106.3	109.2	112.0	107.0	106.3	102.7	108.5
22	Cyprus	100.0	113.3	112.5	112.8	110.5	112.2	117.9	129.1
23	Latvia	100.0	113.4	114.7	109.0	99.4	91.9	90.9	103.2
24	Lithuania	100.0	118.9	114.6	116.1	101.8	93.7	89.3	98.2
25	Luxembourg	100.0	104.8	107.7	109.4	105.9	96.1	95.1	102.1
26	Hungary	100.0	116.2	126.6	115.1	108.2	108.6	103.8	106.8
27	Malta	100.0	100.7	107.0	107.2	97.4	105.3	107.7	101.8
28	Netherlands	100.0	102.1	103.0	107.4	100.6	97.5	97.5	104.5
29	Austria	100.0	104.2	106.3	105.6	99.6	96.1	94.1	99.6
30	Poland	100.0	114.8	115.7	112.3	104.7	101.2	101.9	109.1
31	Portugal	100.0	96.5	97.2	101.8	96.6	94.1	96.5	98.1
32	Romania	100.0	109.2	116.8	118.4	104.0	101.5	102.1	103.9
33	Slovenia	100.0	107.6	108.7	114.7	108.5	104.8	102.5	109.1
34	Slovakia	100.0	113.2	116.4	109.2	100.9	99.0	94.3	97.4
35	Finland	100.0	110.6	113.1	119.1	103.0	99.6	96.5	98.2

36	Sweden	100.0	104.6	104.0	106.0	102.2	100.2	100.7	105.8

In [23]:
```
df_outa_2010
```

Out[23]:

	Geo	2010	2011	2012	2013	2014	2015	2016	2017
10	European Union: 27 countries	100.0	107.4	111.0	112.3	104.4	102.1	100.9	105.8
11	Belgium	100.0	98.7	108.3	111.7	92.6	88.0	89.4	92.3
12	Bulgaria	100.0	119.6	135.1	110.8	107.9	113.5	109.3	107.4
13	Czechia	100.0	118.8	119.1	122.9	117.7	110.3	104.2	109.9
14	Denmark	100.0	111.5	117.2	121.4	107.4	100.9	96.8	103.7
15	Germany	100.0	110.6	114.0	113.5	103.6	99.6	99.0	105.4
16	Ireland	100.0	113.9	117.5	126.9	116.2	111.6	106.4	116.5
17	Greece	100.0	99.9	97.5	100.0	99.6	105.5	104.0	104.0
18	Spain	100.0	97.5	105.9	107.5	99.7	106.7	103.6	109.6
19	France	100.0	110.0	113.6	115.1	109.4	105.2	106.2	107.8
20	Croatia	100.0	105.3	110.1	100.9	95.7	96.3	95.3	98.3
21	Italy	100.0	106.3	109.2	112.0	107.0	106.3	102.7	108.5
22	Cyprus	100.0	113.3	112.5	112.8	110.5	112.2	117.9	129.1
23	Latvia	100.0	113.4	114.7	109.0	99.4	91.9	90.9	103.2
24	Lithuania	100.0	118.9	114.6	116.1	101.8	93.7	89.3	98.2
25	Luxembourg	100.0	104.8	107.7	109.4	105.9	96.1	95.1	102.1
26	Hungary	100.0	116.2	126.6	115.1	108.2	108.6	103.8	106.8
27	Malta	100.0	100.7	107.0	107.2	97.4	105.3	107.7	101.8
28	Netherlands	100.0	102.1	103.0	107.4	100.6	97.5	97.5	104.5
29	Austria	100.0	104.2	106.3	105.6	99.6	96.1	94.1	99.6
30	Poland	100.0	114.8	115.7	112.3	104.7	101.2	101.9	109.1
31	Portugal	100.0	96.5	97.2	101.8	96.6	94.1	96.5	98.1
32	Romania	100.0	109.2	116.8	118.4	104.0	101.5	102.1	103.9
33	Slovenia	100.0	107.6	108.7	114.7	108.5	104.8	102.5	109.1
34	Slovakia	100.0	113.2	116.4	109.2	100.9	99.0	94.3	97.4
35	Finland	100.0	110.6	113.1	119.1	103.0	99.6	96.5	98.2
36	Sweden	100.0	104.6	104.0	106.0	102.2	100.2	100.7	105.8

Period 2005: df_outa_2005

In [24]:
```
# read data Index of price (output)  by period 2005

readexcel_name = "https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw

df_outa_2005 = pd.DataFrame()
```

```python
# columns specific for df_outa_2005

# Fixing the columns names
column_fix = ['Geo', '2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012']

df_outa_2005 = readexcel(df_outa_2005, column_fix, readexcel_name)


# Cleaning and fixing columns df_outa_2005

# this is specific for each excel
df_outa_2005.drop(df_outa_2005.index[-5:], inplace=True)


df_outa_2005
```

```
https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw/data/apri_pi05_ou
ta_2005.xlsx
```

Out[24]:

	Geo	2005	2006	2007	2008	2009	2010	2011	2012
10	European Union: 27 countries	100.0	102.7	110.4	112.0	98.2	104.1	110.7	114.1
11	Belgium	100.0	111.2	109.2	103.6	92.0	97.5	96.6	105.0
12	Bulgaria	100.0	100.7	117.5	117.3	90.9	98.1	108.5	116.5
13	Czechia	100.0	98.6	108.8	110.7	84.0	88.7	101.2	102.7
14	Denmark	100.0	102.1	104.2	112.0	93.7	100.5	111.3	117.2
15	Germany	100.0	105.4	115.2	116.3	93.9	106.0	116.7	119.7
16	Estonia	100.0	98.1	107.2	101.3	78.5	92.4	104.0	101.2
17	Ireland	100.0	101.1	106.5	111.4	94.5	105.8	119.9	122.1
18	Greece	100.0	104.1	111.7	103.6	100.0	108.4	104.1	98.0
19	Spain	100.0	94.9	97.5	96.6	85.9	89.5	87.4	94.0
20	France	100.0	103.5	113.3	116.4	101.4	106.9	115.8	120.6
21	Croatia	100.0	97.2	105.3	99.6	90.2	92.1	96.7	102.8
22	Italy	100.0	102.8	108.7	111.9	101.3	101.2	106.4	109.4
23	Cyprus	100.0	103.4	110.3	119.7	104.2	103.7	100.1	101.0
24	Latvia	100.0	105.9	117.3	104.0	79.4	95.6	108.2	106.4
25	Lithuania	100.0	102.4	113.4	112.3	83.9	96.6	109.8	104.9
26	Luxembourg	100.0	100.0	108.0	107.0	88.5	94.2	99.4	102.1
27	Hungary	100.0	108.8	134.0	113.6	98.7	110.7	126.0	139.2
28	Malta	100.0	97.7	104.2	103.4	106.9	101.6	101.1	107.2
29	Netherlands	100.0	107.6	110.5	107.7	95.2	104.6	107.1	107.5
30	Austria	100.0	105.1	111.3	111.5	97.9	108.1	111.5	113.7
31	Poland	100.0	104.6	117.6	111.3	100.4	107.5	121.8	122.5
32	Portugal	100.0	101.7	103.1	103.1	97.8	101.7	97.2	97.6
33	Romania	100.0	99.5	115.0	122.4	108.0	112.7	121.9	127.2
34	Slovenia	100.0	103.1	107.8	116.9	99.0	99.0	105.6	107.0
35	Slovakia	100.0	95.9	103.3	104.8	77.5	88.0	98.6	101.6
36	Finland	100.0	103.7	107.6	113.1	99.9	104.3	114.6	117.3

37	Sweden	100.0	103.7	116.0	121.7	106.3	116.3	119.2	117.4

In [25]:
```
df_outa_2005
```

Out[25]:

	Geo	2005	2006	2007	2008	2009	2010	2011	2012
10	European Union: 27 countries	100.0	102.7	110.4	112.0	98.2	104.1	110.7	114.1
11	Belgium	100.0	111.2	109.2	103.6	92.0	97.5	96.6	105.0
12	Bulgaria	100.0	100.7	117.5	117.3	90.9	98.1	108.5	116.5
13	Czechia	100.0	98.6	108.8	110.7	84.0	88.7	101.2	102.7
14	Denmark	100.0	102.1	104.2	112.0	93.7	100.5	111.3	117.2
15	Germany	100.0	105.4	115.2	116.3	93.9	106.0	116.7	119.7
16	Estonia	100.0	98.1	107.2	101.3	78.5	92.4	104.0	101.2
17	Ireland	100.0	101.1	106.5	111.4	94.5	105.8	119.9	122.1
18	Greece	100.0	104.1	111.7	103.6	100.0	108.4	104.1	98.0
19	Spain	100.0	94.9	97.5	96.6	85.9	89.5	87.4	94.0
20	France	100.0	103.5	113.3	116.4	101.4	106.9	115.8	120.6
21	Croatia	100.0	97.2	105.3	99.6	90.2	92.1	96.7	102.8
22	Italy	100.0	102.8	108.7	111.9	101.3	101.2	106.4	109.4
23	Cyprus	100.0	103.4	110.3	119.7	104.2	103.7	100.1	101.0
24	Latvia	100.0	105.9	117.3	104.0	79.4	95.6	108.2	106.4
25	Lithuania	100.0	102.4	113.4	112.3	83.9	96.6	109.8	104.9
26	Luxembourg	100.0	100.0	108.0	107.0	88.5	94.2	99.4	102.1
27	Hungary	100.0	108.8	134.0	113.6	98.7	110.7	126.0	139.2
28	Malta	100.0	97.7	104.2	103.4	106.9	101.6	101.1	107.2
29	Netherlands	100.0	107.6	110.5	107.7	95.2	104.6	107.1	107.5
30	Austria	100.0	105.1	111.3	111.5	97.9	108.1	111.5	113.7
31	Poland	100.0	104.6	117.6	111.3	100.4	107.5	121.8	122.5
32	Portugal	100.0	101.7	103.1	103.1	97.8	101.7	97.2	97.6
33	Romania	100.0	99.5	115.0	122.4	108.0	112.7	121.9	127.2
34	Slovenia	100.0	103.1	107.8	116.9	99.0	99.0	105.6	107.0
35	Slovakia	100.0	95.9	103.3	104.8	77.5	88.0	98.6	101.6
36	Finland	100.0	103.7	107.6	113.1	99.9	104.3	114.6	117.3
37	Sweden	100.0	103.7	116.0	121.7	106.3	116.3	119.2	117.4

Period 2000: df_outa_2000

In [26]:
```
# read data Index of price (output)  by period 2000

readexcel_name = "https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw


df_outa_2000 = pd.DataFrame()
```

```
# columns specific for df_outa_2000

# Fixing the columns names
column_fix = ['Geo', '2000', '2001', '2002', '2003', '2004', '2005', '2006', '2007', '20

df_outa_2000 = readexcel(df_outa_2000, column_fix, readexcel_name)


# Cleaning and fixing columns df_outa_2000

# this is specific for each excel
df_outa_2000.drop(df_outa_2000.index[-5:], inplace=True)


df_outa_2000
```

https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw/data/apri_pi00_outa_2000.xlsx

Out[26]:

	Geo	2000	2001	2002	2003	2004	2005	2006	2007	2008
10	European Union: 27 countries	100.0	102.7	97.1	98.2	96.1	92.0	94.6	100.8	102.4
11	Belgium	100.0	102.0	91.0	92.5	92.7	91.4	97.7	98.0	90.8
12	Bulgaria	100.0	108.0	90.2	90.1	94.3	76.6	74.0	82.9	80.7
13	Czechia	100.0	106.1	94.4	91.0	94.7	87.2	86.3	97.9	100.3
14	Denmark	100.0	105.0	92.4	86.5	88.0	85.2	88.5	89.0	96.6
15	Germany	100.0	105.2	96.9	97.1	93.9	91.3	97.6	104.9	105.0
16	Estonia	100.0	NaN	NaN	NaN	117.4	118.0	117.9	125.4	111.4
17	Ireland	100.0	100.4	91.8	87.9	87.9	86.4	88.4	94.4	94.9
18	Greece	100.0	102.4	105.5	110.9	105.6	105.2	108.5	112.8	111.7
19	Spain	100.0	100.2	94.2	96.3	94.4	94.2	90.1	92.9	91.9
20	France	100.0	101.7	96.3	97.7	93.8	88.1	90.8	99.5	102.6
21	Italy	100.0	103.4	102.0	105.4	101.1	93.7	94.5	96.2	98.8
22	Cyprus	100.0	NaN	NaN	NaN	109.0	107.6	111.1	113.8	127.1
23	Latvia	100.0	105.9	102.1	97.3	108.6	114.9	113.0	122.3	112.7
24	Lithuania	100.0	113.0	112.1	101.1	100.9	111.2	126.4	143.2	129.7
25	Luxembourg	100.0	99.4	95.2	93.8	93.8	89.8	88.7	93.9	96.2
26	Hungary	100.0	97.2	90.8	92.0	81.4	79.2	84.2	95.3	87.2
27	Malta	100.0	106.7	104.9	96.9	89.7	86.6	83.9	88.7	88.4
28	Netherlands	100.0	100.9	95.0	94.0	88.0	87.9	96.3	99.5	95.9
29	Austria	100.0	104.3	97.8	96.9	94.5	93.5	96.6	101.9	102.8
30	Poland	100.0	96.5	88.3	89.3	94.4	90.6	96.2	107.4	100.2
31	Portugal	100.0	106.3	96.5	100.4	97.2	93.1	94.0	95.8	95.9
32	Romania	100.0	104.3	106.6	100.6	108.6	93.1	95.6	108.8	112.0
33	Slovenia	100.0	100.4	94.2	92.0	87.8	86.8	88.5	92.3	98.7
34	Slovakia	100.0	100.8	96.3	84.4	80.3	76.2	73.0	75.5	75.7
35	Finland	100.0	102.5	99.0	93.3	96.0	92.4	95.2	99.2	105.6

36	Sweden	100.0	102.4	97.7	94.0	91.6	89.5	91.7	102.2	98.6

Join the data of Index of prices (output) and create NEW DF by period 2000 to 2021

Groups and join the data frames: df_outa_2015, df_outa_2010, df_outa_2005, and df_outa_2000 in order to create a DF from all periods from 2000 to 2021.

Join: it is a inner join in which the "priority" is the newest DF because has the most recent calculation of the index of prices, means from df_outa_2015.

https://pandas.pydata.org/docs/user_guide/merging.html

Important: Joins will be set up in several steps in order to bring clarity and a better understanding of the code.

In [27]:

```
# Step 1: Join df_outa_2015 with df_outa_2010
df_outa = pd.merge(df_outa_2010[['Geo', '2011', '2012', '2013', '2014', '2015']],
         df_outa_2015[['Geo', '2016', '2017', '2018', '2019', '2020', '2021']],
         how='right',
         on = 'Geo')
df_outa
```

Out[27]:

	Geo	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
0	European Union: 27 countries	107.4	111.0	112.3	104.4	102.1	98.46	103.97	102.97	103.43	102.46	108.96
1	Belgium	98.7	108.3	111.7	92.6	88.0	101.66	104.84	110.28	108.05	103.92	111.13
2	Bulgaria	119.6	135.1	110.8	107.9	113.5	96.07	94.94	93.53	92.68	99.24	115.99
3	Czechia	118.8	119.1	122.9	117.7	110.3	93.55	98.16	96.29	98.61	92.28	95.48
4	Denmark	111.5	117.2	121.4	107.4	100.9	94.66	102.87	96.52	102.94	103.62	104.30
5	Germany	110.6	114.0	113.5	103.6	99.6	98.31	106.37	104.81	105.78	101.89	107.60
6	Estonia	NaN	NaN	NaN	NaN	NaN	96.60	106.56	105.24	104.47	101.62	113.17
7	Ireland	113.9	117.5	126.9	116.2	111.6	95.31	106.32	103.49	101.45	102.71	111.90
8	Greece	99.9	97.5	100.0	99.6	105.5	98.04	98.58	97.77	97.72	97.83	107.58
9	Spain	97.5	105.9	107.5	99.7	106.7	96.65	101.75	99.44	94.57	95.08	100.60
10	France	110.0	113.6	115.1	109.4	105.2	99.83	102.22	102.16	102.92	102.91	109.99
11	Croatia	105.3	110.1	100.9	95.7	96.3	98.56	101.70	99.52	99.47	99.92	109.58
12	Italy	106.3	109.2	112.0	107.0	106.3	97.10	103.65	103.41	103.88	104.95	111.81
13	Cyprus	113.3	112.5	112.8	110.5	112.2	99.66	101.49	99.80	105.05	102.20	96.66
14	Latvia	113.4	114.7	109.0	99.4	91.9	98.45	107.30	110.86	109.96	107.64	121.45
15	Lithuania	118.9	114.6	116.1	101.8	93.7	92.64	100.31	100.35	102.28	99.78	110.28
16	Luxembourg	104.8	107.7	109.4	105.9	96.1	98.00	107.11	102.02	102.24	101.02	102.37
17	Hungary	116.2	126.6	115.1	108.2	108.6	96.01	98.55	98.30	100.24	104.29	118.97
18	Malta	100.7	107.0	107.2	97.4	105.3	100.69	97.45	96.40	103.91	102.22	104.14
19	Netherlands	102.1	103.0	107.4	100.6	97.5	100.19	106.02	102.21	102.76	96.39	102.56

20	Austria	104.2	106.3	105.6	99.6	96.1	98.05	103.47	100.09	100.58	99.94	105.78
21	Poland	114.8	115.7	112.3	104.7	101.2	100.52	116.34	114.23	120.21	115.40	119.00
22	Portugal	96.5	97.2	101.8	96.6	94.1	101.77	102.75	104.01	104.42	104.93	109.83
23	Romania	109.2	116.8	118.4	104.0	101.5	101.80	102.83	104.74	110.86	116.55	123.11
24	Slovenia	107.6	108.7	114.7	108.5	104.8	98.34	105.43	102.84	104.91	103.08	109.37
25	Slovakia	113.2	116.4	109.2	100.9	99.0	102.32	105.90	97.72	96.81	95.29	104.06
26	Finland	110.6	113.1	119.1	103.0	99.6	96.63	98.65	100.84	99.55	96.91	101.74
27	Sweden	104.6	104.0	106.0	102.2	100.2	100.61	106.21	115.61	112.18	110.18	117.58

In [28]:

```python
# Step 2: Join previous result with df_outa_2005
df_outa = pd.merge(df_outa_2005[['Geo', '2006', '2007', '2008', '2009', '2010']],
         df_outa,
         how='right',
         on = 'Geo')
df_outa
```

Out[28]:

	Geo	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	201
0	European Union: 27 countries	102.7	110.4	112.0	98.2	104.1	107.4	111.0	112.3	104.4	102.1	98.46	103.97	102.9
1	Belgium	111.2	109.2	103.6	92.0	97.5	98.7	108.3	111.7	92.6	88.0	101.66	104.84	110.2
2	Bulgaria	100.7	117.5	117.3	90.9	98.1	119.6	135.1	110.8	107.9	113.5	96.07	94.94	93.5
3	Czechia	98.6	108.8	110.7	84.0	88.7	118.8	119.1	122.9	117.7	110.3	93.55	98.16	96.2
4	Denmark	102.1	104.2	112.0	93.7	100.5	111.5	117.2	121.4	107.4	100.9	94.66	102.87	96.5
5	Germany	105.4	115.2	116.3	93.9	106.0	110.6	114.0	113.5	103.6	99.6	98.31	106.37	104.8
6	Estonia	98.1	107.2	101.3	78.5	92.4	NaN	NaN	NaN	NaN	NaN	96.60	106.56	105.2
7	Ireland	101.1	106.5	111.4	94.5	105.8	113.9	117.5	126.9	116.2	111.6	95.31	106.32	103.4
8	Greece	104.1	111.7	103.6	100.0	108.4	99.9	97.5	100.0	99.6	105.5	98.04	98.58	97.7
9	Spain	94.9	97.5	96.6	85.9	89.5	97.5	105.9	107.5	99.7	106.7	96.65	101.75	99.4
10	France	103.5	113.3	116.4	101.4	106.9	110.0	113.6	115.1	109.4	105.2	99.83	102.22	102.1
11	Croatia	97.2	105.3	99.6	90.2	92.1	105.3	110.1	100.9	95.7	96.3	98.56	101.70	99.5
12	Italy	102.8	108.7	111.9	101.3	101.2	106.3	109.2	112.0	107.0	106.3	97.10	103.65	103.4
13	Cyprus	103.4	110.3	119.7	104.2	103.7	113.3	112.5	112.8	110.5	112.2	99.66	101.49	99.8
14	Latvia	105.9	117.3	104.0	79.4	95.6	113.4	114.7	109.0	99.4	91.9	98.45	107.30	110.8
15	Lithuania	102.4	113.4	112.3	83.9	96.6	118.9	114.6	116.1	101.8	93.7	92.64	100.31	100.3
16	Luxembourg	100.0	108.0	107.0	88.5	94.2	104.8	107.7	109.4	105.9	96.1	98.00	107.11	102.0
17	Hungary	108.8	134.0	113.6	98.7	110.7	116.2	126.6	115.1	108.2	108.6	96.01	98.55	98.3
18	Malta	97.7	104.2	103.4	106.9	101.6	100.7	107.0	107.2	97.4	105.3	100.69	97.45	96.4
19	Netherlands	107.6	110.5	107.7	95.2	104.6	102.1	103.0	107.4	100.6	97.5	100.19	106.02	102.2
20	Austria	105.1	111.3	111.5	97.9	108.1	104.2	106.3	105.6	99.6	96.1	98.05	103.47	100.0
21	Poland	104.6	117.6	111.3	100.4	107.5	114.8	115.7	112.3	104.7	101.2	100.52	116.34	114.2
22	Portugal	101.7	103.1	103.1	97.8	101.7	96.5	97.2	101.8	96.6	94.1	101.77	102.75	104.0
23	Romania	99.5	115.0	122.4	108.0	112.7	109.2	116.8	118.4	104.0	101.5	101.80	102.83	104.7

24	Slovenia	103.1	107.8	116.9	99.0	99.0	107.6	108.7	114.7	108.5	104.8	98.34	105.43	102.8
25	Slovakia	95.9	103.3	104.8	77.5	88.0	113.2	116.4	109.2	100.9	99.0	102.32	105.90	97.7
26	Finland	103.7	107.6	113.1	99.9	104.3	110.6	113.1	119.1	103.0	99.6	96.63	98.65	100.8
27	Sweden	103.7	116.0	121.7	106.3	116.3	104.6	104.0	106.0	102.2	100.2	100.61	106.21	115.6

In [29]:

```
# Step 3: Join previous result with df_outa_2000

# Using right join because df_outa_2000 does not has data from Croatia

df_outa = pd.merge(df_outa_2000[['Geo', '2000', '2001', '2002', '2003', '2004', '2005']]
         df_outa,
         how='right', # using right join
         on = 'Geo')
df_outa
```

Out[29]:

	Geo	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	2012	2013	2014	20
0	European Union: 27 countries	100.0	102.7	97.1	98.2	96.1	92.0	102.7	110.4	112.0	...	111.0	112.3	104.4	10
1	Belgium	100.0	102.0	91.0	92.5	92.7	91.4	111.2	109.2	103.6	...	108.3	111.7	92.6	88
2	Bulgaria	100.0	108.0	90.2	90.1	94.3	76.6	100.7	117.5	117.3	...	135.1	110.8	107.9	113
3	Czechia	100.0	106.1	94.4	91.0	94.7	87.2	98.6	108.8	110.7	...	119.1	122.9	117.7	110
4	Denmark	100.0	105.0	92.4	86.5	88.0	85.2	102.1	104.2	112.0	...	117.2	121.4	107.4	100
5	Germany	100.0	105.2	96.9	97.1	93.9	91.3	105.4	115.2	116.3	...	114.0	113.5	103.6	99
6	Estonia	100.0	NaN	NaN	NaN	117.4	118.0	98.1	107.2	101.3	...	NaN	NaN	NaN	Na
7	Ireland	100.0	100.4	91.8	87.9	87.9	86.4	101.1	106.5	111.4	...	117.5	126.9	116.2	11
8	Greece	100.0	102.4	105.5	110.9	105.6	105.2	104.1	111.7	103.6	...	97.5	100.0	99.6	105
9	Spain	100.0	100.2	94.2	96.3	94.4	94.2	94.9	97.5	96.6	...	105.9	107.5	99.7	106
10	France	100.0	101.7	96.3	97.7	93.8	88.1	103.5	113.3	116.4	...	113.6	115.1	109.4	105
11	Croatia	NaN	NaN	NaN	NaN	NaN	NaN	97.2	105.3	99.6	...	110.1	100.9	95.7	96
12	Italy	100.0	103.4	102.0	105.4	101.1	93.7	102.8	108.7	111.9	...	109.2	112.0	107.0	106
13	Cyprus	100.0	NaN	NaN	NaN	109.0	107.6	103.4	110.3	119.7	...	112.5	112.8	110.5	112
14	Latvia	100.0	105.9	102.1	97.3	108.6	114.9	105.9	117.3	104.0	...	114.7	109.0	99.4	9
15	Lithuania	100.0	113.0	112.1	101.1	100.9	111.2	102.4	113.4	112.3	...	114.6	116.1	101.8	93
16	Luxembourg	100.0	99.4	95.2	93.8	93.8	89.8	100.0	108.0	107.0	...	107.7	109.4	105.9	9
17	Hungary	100.0	97.2	90.8	92.0	81.4	79.2	108.8	134.0	113.6	...	126.6	115.1	108.2	108
18	Malta	100.0	106.7	104.9	96.9	89.7	86.6	97.7	104.2	103.4	...	107.0	107.2	97.4	105
19	Netherlands	100.0	100.9	95.0	94.0	88.0	87.9	107.6	110.5	107.7	...	103.0	107.4	100.6	9
20	Austria	100.0	104.3	97.8	96.9	94.5	93.5	105.1	111.3	111.5	...	106.3	105.6	99.6	9
21	Poland	100.0	96.5	88.3	89.3	94.4	90.6	104.6	117.6	111.3	...	115.7	112.3	104.7	10
22	Portugal	100.0	106.3	96.5	100.4	97.2	93.1	101.7	103.1	103.1	...	97.2	101.8	96.6	9
23	Romania	100.0	104.3	106.6	100.6	108.6	93.1	99.5	115.0	122.4	...	116.8	118.4	104.0	10
24	Slovenia	100.0	100.4	94.2	92.0	87.8	86.8	103.1	107.8	116.9	...	108.7	114.7	108.5	104
25	Slovakia	100.0	100.8	96.3	84.4	80.3	76.2	95.9	103.3	104.8	...	116.4	109.2	100.9	9

26	Finland	100.0	102.5	99.0	93.3	96.0	92.4	103.7	107.6	113.1	...	113.1	119.1	103.0	99
27	Sweden	100.0	102.4	97.7	94.0	91.6	89.5	103.7	116.0	121.7	...	104.0	106.0	102.2	10

28 rows × 23 columns

Data wrangling: cleaning, missing values and outliers

Missing values

Basically, the price index illustrates how the price of a product or a basket of products has changed since the base period.

The **base price of an index is 100** by agreement (according to Eurostat), meaning that, for instance, an index equal to 110 reflects an increase in the absolute price of 10% and an index equal to 95 a decrease of 5%.

Please see: https://ec.europa.eu/eurostat/cache/metadata/en/apri_pi_esms.htm

This value: 100, will be considered in order to fix the missing values, meaning that any **missing value will be substituted by the base price** instead of the mean or median as conventionally used.

In [30]:

```
# Using Pandas

base_price=100.0  # base price of an index is 100 by agreement

df_outa = df_outa.fillna(base_price)

df_outa
```

Out[30]:

	Geo	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	2012	2013	2014	20
0	European Union: 27 countries	100.0	102.7	97.1	98.2	96.1	92.0	102.7	110.4	112.0	...	111.0	112.3	104.4	10
1	Belgium	100.0	102.0	91.0	92.5	92.7	91.4	111.2	109.2	103.6	...	108.3	111.7	92.6	88
2	Bulgaria	100.0	108.0	90.2	90.1	94.3	76.6	100.7	117.5	117.3	...	135.1	110.8	107.9	11
3	Czechia	100.0	106.1	94.4	91.0	94.7	87.2	98.6	108.8	110.7	...	119.1	122.9	117.7	11
4	Denmark	100.0	105.0	92.4	86.5	88.0	85.2	102.1	104.2	112.0	...	117.2	121.4	107.4	10
5	Germany	100.0	105.2	96.9	97.1	93.9	91.3	105.4	115.2	116.3	...	114.0	113.5	103.6	9
6	Estonia	100.0	100.0	100.0	100.0	117.4	118.0	98.1	107.2	101.3	...	100.0	100.0	100.0	10
7	Ireland	100.0	100.4	91.8	87.9	87.9	86.4	101.1	106.5	111.4	...	117.5	126.9	116.2	11
8	Greece	100.0	102.4	105.5	110.9	105.6	105.2	104.1	111.7	103.6	...	97.5	100.0	99.6	10
9	Spain	100.0	100.2	94.2	96.3	94.4	94.2	94.9	97.5	96.6	...	105.9	107.5	99.7	10
10	France	100.0	101.7	96.3	97.7	93.8	88.1	103.5	113.3	116.4	...	113.6	115.1	109.4	10
11	Croatia	100.0	100.0	100.0	100.0	100.0	100.0	97.2	105.3	99.6	...	110.1	100.9	95.7	9
12	Italy	100.0	103.4	102.0	105.4	101.1	93.7	102.8	108.7	111.9	...	109.2	112.0	107.0	10
13	Cyprus	100.0	100.0	100.0	100.0	109.0	107.6	103.4	110.3	119.7	...	112.5	112.8	110.5	11

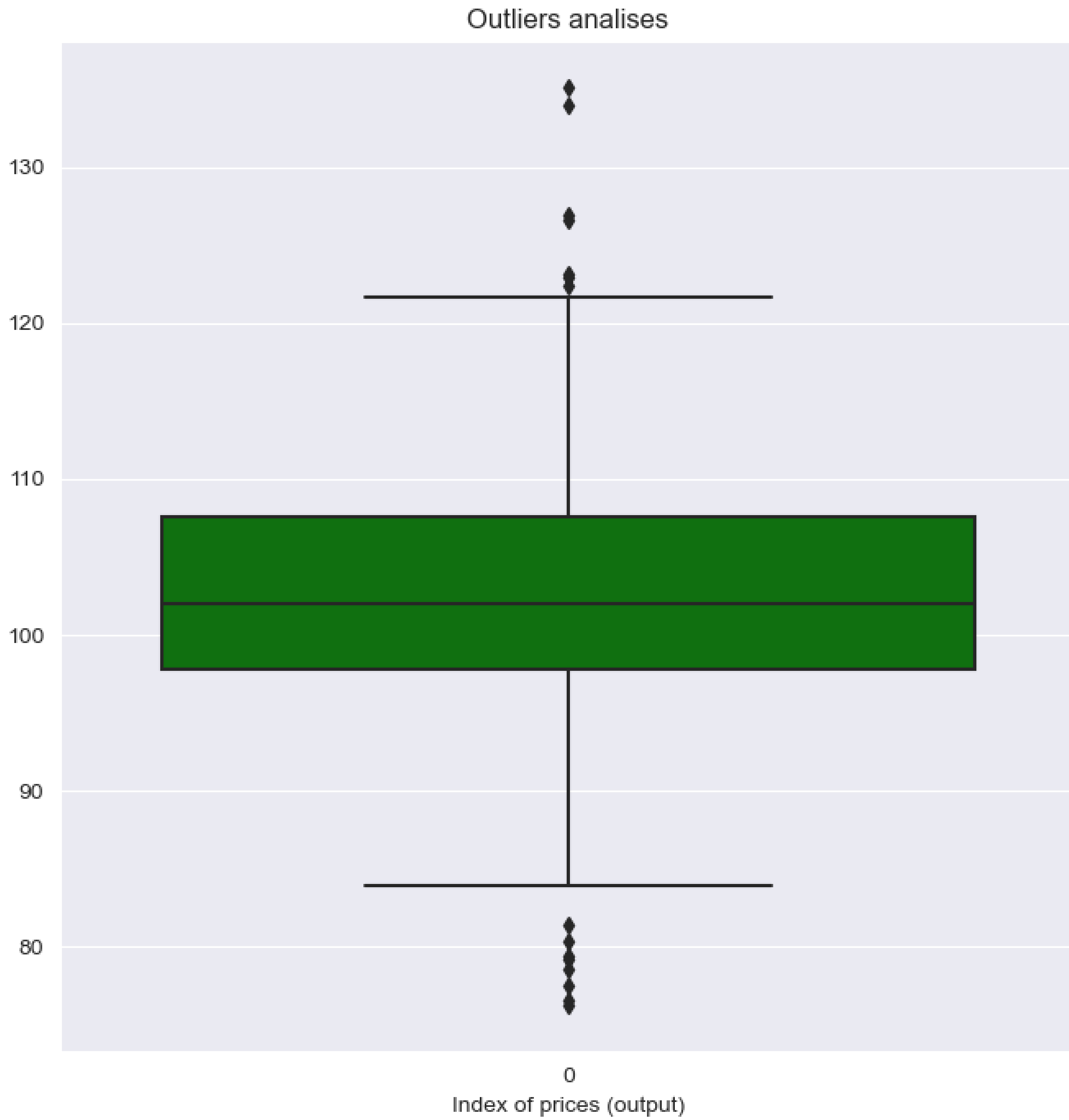
14	Latvia	100.0	105.9	102.1	97.3	108.6	114.9	105.9	117.3	104.0	...	114.7	109.0	99.4	91
15	Lithuania	100.0	113.0	112.1	101.1	100.9	111.2	102.4	113.4	112.3	...	114.6	116.1	101.8	93
16	Luxembourg	100.0	99.4	95.2	93.8	93.8	89.8	100.0	108.0	107.0	...	107.7	109.4	105.9	96
17	Hungary	100.0	97.2	90.8	92.0	81.4	79.2	108.8	134.0	113.6	...	126.6	115.1	108.2	108
18	Malta	100.0	106.7	104.9	96.9	89.7	86.6	97.7	104.2	103.4	...	107.0	107.2	97.4	105
19	Netherlands	100.0	100.9	95.0	94.0	88.0	87.9	107.6	110.5	107.7	...	103.0	107.4	100.6	97
20	Austria	100.0	104.3	97.8	96.9	94.5	93.5	105.1	111.3	111.5	...	106.3	105.6	99.6	96
21	Poland	100.0	96.5	88.3	89.3	94.4	90.6	104.6	117.6	111.3	...	115.7	112.3	104.7	101
22	Portugal	100.0	106.3	96.5	100.4	97.2	93.1	101.7	103.1	103.1	...	97.2	101.8	96.6	94
23	Romania	100.0	104.3	106.6	100.6	108.6	93.1	99.5	115.0	122.4	...	116.8	118.4	104.0	101
24	Slovenia	100.0	100.4	94.2	92.0	87.8	86.8	103.1	107.8	116.9	...	108.7	114.7	108.5	104
25	Slovakia	100.0	100.8	96.3	84.4	80.3	76.2	95.9	103.3	104.8	...	116.4	109.2	100.9	99
26	Finland	100.0	102.5	99.0	93.3	96.0	92.4	103.7	107.6	113.1	...	113.1	119.1	103.0	99
27	Sweden	100.0	102.4	97.7	94.0	91.6	89.5	103.7	116.0	121.7	...	104.0	106.0	102.2	100

28 rows × 23 columns

Analysis Outliers Index of prices (input): df_outa

In [31]:
```
# melt_pivot

df_outa_index = df_outa.melt(id_vars=["Geo"],
                  var_name="Year",
                  value_name="Price_Index")
```

In [32]:
```
# visualize univariable outliers

fig = plt.figure(figsize=(8, 8))
ax = sns.boxplot(data=df_outa_index['Price_Index'], color='green')
# used the colour green to be different to the analysis of input data: df_ina

ax.set_xlabel('Index of prices (output)')
plt.title('Outliers analises')
plt.show()
```



In [33]:
```
# Search for tukey fence on the target feature df_outa_index['Price_Index']
outliers_outer_indexes, outliers_inner_indexes = tukeys_method(df_outa_index, 'Price_Ind

print('\nOuter index:  ', outliers_outer_indexes)
print('\nInner index:  ', outliers_inner_indexes)
```

```
Q1:  97.79249999999999
Q3:  107.6

Outer index:   []

Inner index:   [129, 137, 142, 157, 165, 213, 247, 258, 266, 277, 338, 353, 367, 371, 61
1]
```

It is no observations out of the Outer Fence according to the Tukey Methods.

For this project it will consider this has a **low impact on the samples, and in the model**, therefore, it will **not change** values to any observations.

Annual evolution of the index of producer prices of agricultural products (output) by period 2000-2021: Irland vs EU

In [34]:

```
df_outa
```

Out[34]:

	Geo	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	2012	2013	2014	20
0	European Union: 27 countries	100.0	102.7	97.1	98.2	96.1	92.0	102.7	110.4	112.0	...	111.0	112.3	104.4	10
1	Belgium	100.0	102.0	91.0	92.5	92.7	91.4	111.2	109.2	103.6	...	108.3	111.7	92.6	88
2	Bulgaria	100.0	108.0	90.2	90.1	94.3	76.6	100.7	117.5	117.3	...	135.1	110.8	107.9	113
3	Czechia	100.0	106.1	94.4	91.0	94.7	87.2	98.6	108.8	110.7	...	119.1	122.9	117.7	110
4	Denmark	100.0	105.0	92.4	86.5	88.0	85.2	102.1	104.2	112.0	...	117.2	121.4	107.4	100
5	Germany	100.0	105.2	96.9	97.1	93.9	91.3	105.4	115.2	116.3	...	114.0	113.5	103.6	99
6	Estonia	100.0	100.0	100.0	100.0	117.4	118.0	98.1	107.2	101.3	...	100.0	100.0	100.0	100
7	Ireland	100.0	100.4	91.8	87.9	87.9	86.4	101.1	106.5	111.4	...	117.5	126.9	116.2	11
8	Greece	100.0	102.4	105.5	110.9	105.6	105.2	104.1	111.7	103.6	...	97.5	100.0	99.6	105
9	Spain	100.0	100.2	94.2	96.3	94.4	94.2	94.9	97.5	96.6	...	105.9	107.5	99.7	106
10	France	100.0	101.7	96.3	97.7	93.8	88.1	103.5	113.3	116.4	...	113.6	115.1	109.4	105
11	Croatia	100.0	100.0	100.0	100.0	100.0	100.0	97.2	105.3	99.6	...	110.1	100.9	95.7	96
12	Italy	100.0	103.4	102.0	105.4	101.1	93.7	102.8	108.7	111.9	...	109.2	112.0	107.0	106
13	Cyprus	100.0	100.0	100.0	100.0	109.0	107.6	103.4	110.3	119.7	...	112.5	112.8	110.5	112
14	Latvia	100.0	105.9	102.1	97.3	108.6	114.9	105.9	117.3	104.0	...	114.7	109.0	99.4	9
15	Lithuania	100.0	113.0	112.1	101.1	100.9	111.2	102.4	113.4	112.3	...	114.6	116.1	101.8	93
16	Luxembourg	100.0	99.4	95.2	93.8	93.8	89.8	100.0	108.0	107.0	...	107.7	109.4	105.9	9
17	Hungary	100.0	97.2	90.8	92.0	81.4	79.2	108.8	134.0	113.6	...	126.6	115.1	108.2	108
18	Malta	100.0	106.7	104.9	96.9	89.7	86.6	97.7	104.2	103.4	...	107.0	107.2	97.4	105
19	Netherlands	100.0	100.9	95.0	94.0	88.0	87.9	107.6	110.5	107.7	...	103.0	107.4	100.6	9
20	Austria	100.0	104.3	97.8	96.9	94.5	93.5	105.1	111.3	111.5	...	106.3	105.6	99.6	9
21	Poland	100.0	96.5	88.3	89.3	94.4	90.6	104.6	117.6	111.3	...	115.7	112.3	104.7	10
22	Portugal	100.0	106.3	96.5	100.4	97.2	93.1	101.7	103.1	103.1	...	97.2	101.8	96.6	9
23	Romania	100.0	104.3	106.6	100.6	108.6	93.1	99.5	115.0	122.4	...	116.8	118.4	104.0	10
24	Slovenia	100.0	100.4	94.2	92.0	87.8	86.8	103.1	107.8	116.9	...	108.7	114.7	108.5	10
25	Slovakia	100.0	100.8	96.3	84.4	80.3	76.2	95.9	103.3	104.8	...	116.4	109.2	100.9	99
26	Finland	100.0	102.5	99.0	93.3	96.0	92.4	103.7	107.6	113.1	...	113.1	119.1	103.0	99
27	Sweden	100.0	102.4	97.7	94.0	91.6	89.5	103.7	116.0	121.7	...	104.0	106.0	102.2	100

28 rows × 23 columns

Function for melt DF's

In [35]:

```
# function for melt
```

```python
# melt_pivot all df's

def melt_pivot(df, v):
    df = df.melt(id_vars=['Geo'],
                 var_name='Year',
                 value_name=v)
    df[v] = df[v].astype(float)
    return df
```

In [36]:
```python
# melt_pivot


df_outa_t = melt_pivot(df_outa, 'Price_Index')

#df_outa_t = df_outa.melt(id_vars=['Geo'],
#                    var_name='Year',
#                    value_name="Price_Index")
#df_outa_T['Year'] = df_outa_T['Year'].astype(int)
#df_outa_T['Price_Index'] = df_outa_T['Price_Index'].astype(float)
df_outa_t
```

Out[36]:

	Geo	Year	Price_Index
0	European Union: 27 countries	2000	100.00
1	Belgium	2000	100.00
2	Bulgaria	2000	100.00
3	Czechia	2000	100.00
4	Denmark	2000	100.00
...	...	...	...
611	Romania	2021	123.11
612	Slovenia	2021	109.37
613	Slovakia	2021	104.06
614	Finland	2021	101.74
615	Sweden	2021	117.58

616 rows × 3 columns

In [37]:
```python
df_tmp = df_outa_t[(df_outa_t['Geo']=='Ireland') | (df_outa_t['Geo']=='European Union: 2

plt.figure(figsize=(25,7))
sns.lineplot(x=df_tmp["Year"],y=df_tmp['Price_Index'],hue= df_tmp['Geo'])
sns.scatterplot(x=df_tmp["Year"],y=df_tmp['Price_Index'],hue=df_tmp['Geo'])
plt.xticks(rotation=45);
plt.title("Index of producer prices of agricultural products (output) from 2000 to 2021:
plt.show()
```

Annual evolution of the Index of variation of the expenditure incurred by farmers(input) by period 2000-2021: Irland vs EU

In [38]:

```
df_ina
```

Out[38]:

	Geo	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	2012	2013	2014	20
0	European Union: 27 countries	100.0	101.4	99.7	99.4	101.5	100.7	101.2	106.1	116.8	...	108.5	108.4	104.8	102
1	Belgium	100.0	100.2	99.1	97.7	96.2	97.6	103.1	111.7	122.3	...	111.1	110.2	101.7	99
2	Bulgaria	100.0	100.0	100.0	100.0	100.0	100.0	95.9	98.7	100.6	...	110.5	109.0	106.0	103
3	Czechia	100.0	100.1	97.4	96.0	99.8	97.7	98.6	101.1	104.1	...	106.4	108.1	106.0	103
4	Denmark	100.0	103.5	102.2	99.0	101.2	101.0	100.6	107.0	121.1	...	108.9	112.4	111.6	109
5	Germany	100.0	102.1	100.3	99.5	101.3	99.8	102.2	107.6	118.3	...	110.8	111.1	106.5	104
6	Estonia	100.0	100.0	100.0	100.0	100.0	100.0	101.5	103.6	103.4	...	100.0	100.0	100.0	100
7	Ireland	100.0	100.4	97.5	96.0	97.1	98.9	100.5	103.8	115.4	...	111.1	113.4	108.8	106
8	Greece	100.0	98.4	97.4	97.9	102.2	103.8	100.5	103.9	110.6	...	107.0	107.3	106.0	104
9	Spain	100.0	100.0	97.4	95.7	96.5	95.1	99.7	104.4	115.3	...	110.3	108.7	105.3	105
10	France	100.0	101.3	99.9	99.0	100.3	100.3	100.9	105.1	119.4	...	107.3	107.6	104.6	102
11	Croatia	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	...	111.7	108.5	99.2	96
12	Italy	100.0	102.1	100.7	100.8	103.4	99.1	101.2	105.8	114.4	...	105.5	106.3	104.2	101
13	Cyprus	100.0	100.0	100.0	100.0	128.6	136.0	104.9	110.7	115.4	...	96.4	106.7	107.3	110
14	Latvia	100.0	99.2	97.9	99.1	100.9	111.3	102.8	106.2	109.3	...	108.9	109.3	106.4	104
15	Lithuania	100.0	100.0	100.0	100.0	100.0	100.0	111.3	113.4	131.6	...	120.3	114.8	109.1	112
16	Luxembourg	100.0	101.1	100.3	99.1	100.2	97.7	99.6	103.1	108.5	...	106.0	104.5	102.6	100
17	Hungary	100.0	102.5	98.3	99.7	100.8	97.4	101.9	105.6	115.3	...	108.9	109.3	106.2	104
18	Malta	100.0	98.3	96.7	90.7	93.2	93.8	100.7	105.3	119.4	...	108.8	108.8	104.8	102
19	Netherlands	100.0	100.9	98.2	97.7	97.9	97.9	104.4	110.5	116.5	...	107.3	107.1	101.9	100
20	Austria	100.0	99.6	97.6	98.2	99.6	98.8	100.7	104.2	110.1	...	105.1	104.9	102.9	101
21	Poland	100.0	101.2	101.6	103.9	107.8	108.0	99.0	102.7	109.4	...	109.8	109.4	107.0	104
22	Portugal	100.0	100.1	96.4	94.5	96.1	97.3	99.7	104.9	116.5	...	108.6	111.0	107.7	105
23	Romania	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	112.2	...	109.7	109.4	104.9	10

24	Slovenia	100.0	103.1	98.9	98.1	103.0	101.9	100.8	105.5	117.9	...	109.8	110.2	105.8	103
25	Slovakia	100.0	100.0	100.0	100.0	89.2	87.5	99.2	101.9	107.2	...	109.4	107.9	101.1	95
26	Finland	100.0	99.6	98.2	98.2	100.8	103.5	102.3	105.4	117.3	...	108.5	108.2	105.3	103
27	Sweden	100.0	102.3	102.3	102.1	104.8	106.0	101.6	106.4	117.4	...	106.2	106.1	105.8	105

28 rows × 23 columns

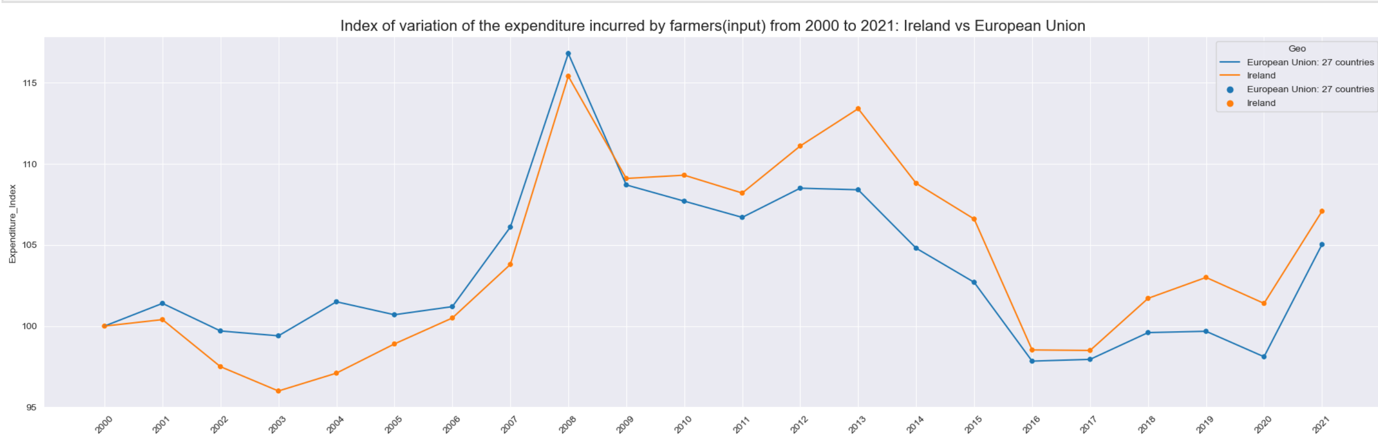
In [39]:
```
# melt_pivot

# calling function melt_pivot

df_ina_t = melt_pivot(df_ina, 'Expenditure_Index')

df_ina_t
```

Out[39]:

	Geo	Year	Expenditure_Index
0	European Union: 27 countries	2000	100.00
1	Belgium	2000	100.00
2	Bulgaria	2000	100.00
3	Czechia	2000	100.00
4	Denmark	2000	100.00
...	...	...	...
611	Romania	2021	107.23
612	Slovenia	2021	106.88
613	Slovakia	2021	95.70
614	Finland	2021	105.16
615	Sweden	2021	107.61

616 rows × 3 columns

In [40]:
```
df_tmp = df_ina_t[(df_ina_t['Geo']=='Ireland') | (df_ina_t['Geo']=='European Union: 27 c

plt.figure(figsize=(25,7))
sns.lineplot(x=df_tmp["Year"],y=df_tmp['Expenditure_Index'],hue= df_tmp['Geo'])
sns.scatterplot(x=df_tmp["Year"],y=df_tmp['Expenditure_Index'],hue=df_tmp['Geo'])
plt.xticks(rotation=45);
plt.title("Index of variation of the expenditure incurred by farmers(input) from 2000 to
plt.show()
```



End data of Index of prices (output)

GDP - Gross domestic product on output, expenditure and income

(CRISP-DM Phase: Data Understanding Phase)

GDP - Gross domestic product on output, expenditure and income

All those indexes are impacted by other economical factors but in particular by the GDP - Gross domestic product on output, expenditure and income.

Eurostat publishes annual and quarterly national accounts use and input-output tables, which are each presented with associated metadata with the index of prices. Even though consistency checks are a major aspect of data validation, temporary (usually limited) inconsistencies between datasets may occur, mainly due to vintage effects.

Data are available from 2010 in Eurostat.

In order to maintain the consistency and coherence of the data in this project, its development a second part of the analysis from 2010 to 2021.

https://ec.europa.eu/eurostat/cache/metadata/en/namq_10_esms.htm

In [41]:

```
# function to read file excel downloaded from index of prices input and output

# https://ec.europa.eu/eurostat/web/agriculture/data/database


def readexcelGDP(df, readexcel_name):

    # link to GitHub
    link = readexcel_name
    print(link)
    # to read just one sheet to dataframe:
    df = pd.read_excel(link,'Sheet 1')

    # Cleaning and fixing columns

    # delete row innecesaries (headers of the original excel that do not contain relevan

    df.drop(df.index[0:8], inplace=True)
    #df.drop(df.index[-8:], inplace=True)
    column = df.iloc[0].values.tolist()
    df.columns = column
    df = df[df.columns.dropna()]
    df.iloc[0:2]
    df.drop(df.index[0:2], inplace=True)

    # Fixing the columns names

    df.rename(columns={'TIME':'Geo'}, inplace=True)
```

```python
    # Fixing the value of standard columns
    df['Geo'].iloc[0] = 'European Union: 27 countries'
    df['Geo'] = df['Geo'].replace('Germany (until 1990 former territory of the FRG)', 'G

    # convert to numerical, objects values

    df.loc[:, df.columns != 'Geo'] = df.loc[:, df.columns != 'Geo'].apply(pd.to_numeric,
    # use this option to convert "special" characters to NaN
    # invalid parsing will be set as NaN
    df = df.apply(pd.to_numeric, errors='ignore')
    # Convert all columns that can be converted into float
    # Error were raised because their type was Object

    return df
```

In [42]:

```python
# function to read file excel downloaded from Gross domestic product on output, expendit

# https://ec.europa.eu/eurostat/cache/metadata/en/namq_10_esms.htm

# Read data GDP by period 2010-2021

# Percentage change on previous period

readexcel_name = "https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw

df_gdp_2010= pd.DataFrame()

df_gdp_2010 = readexcelGDP(df_gdp_2010, readexcel_name)

# delete row unnecessaries (bottom of the original excel that does NOT contain relevant
# this is specific for each excel
df_gdp_2010.drop(df_gdp_2010.index[-6:], inplace=True)

df_gdp_2010
```

```
https://github.com/sba22223nestorpereira/CCT_sba22223nestorpereira/raw/data/namq_10_gdp_
2010.xlsx
```

Out[42]:

	Geo	2010-Q1	2010-Q2	2010-Q3	2010-Q4	2011-Q1	2011-Q2	2011-Q3	2011-Q4	2012-Q1	...	2020-Q2	2020-Q3	20:
10	European Union: 27 countries	0.1	0.2	-0.1	-0.2	-0.2	0.1	0.2	-0.4	-0.2	...	-11.1	11.5	-
11	Belgium	0.2	0.3	0.2	0.2	0.0	0.2	0.3	0.3	0.2	...	-11.4	11.7	-
12	Bulgaria	1.0	0.1	-0.2	0.7	-0.2	1.0	0.1	-0.1	0.7	...	-4.8	3.6	
13	Czechia	0.3	-0.2	-0.4	-0.2	-0.5	0.3	-0.2	0.1	-0.2	...	-8.8	7.0	
14	Denmark	1.0	-1.2	0.1	-0.1	0.1	1.0	-1.2	0.8	-0.1	...	-6.3	5.9	
15	Germany	0.1	0.9	0.3	0.2	0.2	0.1	0.9	-0.3	0.2	...	-9.5	9.0	
16	Estonia	1.9	1.7	0.5	0.6	1.2	1.9	1.7	0.0	0.6	...	-6.8	4.8	
17	Ireland	1.2	-1.4	-1.0	0.0	1.3	1.2	-1.4	-0.2	0.0	...	-5.6	12.9	-
18	Greece	-1.8	-2.0	-1.5	-0.3	-1.7	-1.8	-2.0	-4.5	-0.3	...	-13.4	5.3	
19	Spain	-0.3	-0.6	-0.5	-0.9	-1.0	-0.3	-0.6	-0.7	-0.9	...	-17.8	16.6	-
20	France	-0.1	0.5	0.2	0.0	-0.2	-0.1	0.5	0.2	0.0	...	-13.5	18.3	-
21	Croatia	1.5	-0.4	0.3	-1.2	-0.9	1.5	-0.4	-0.7	-1.2	...	-14.7	5.8	

22	Italy	0.0	-0.5	-0.5	-1.1	-0.7	0.0	-0.5	-1.0	-1.1	...	-12.1	14.5	-
23	Cyprus	0.4	-2.1	-1.2	-0.5	-1.8	0.4	-2.1	0.5	-0.5	...	-11.5	6.1	
24	Latvia	3.4	1.5	1.3	5.1	-0.6	3.4	1.5	0.2	5.1	...	-7.1	6.0	
25	Lithuania	1.4	0.9	2.1	0.5	0.6	1.4	0.9	1.1	0.5	...	-5.3	3.7	
26	Luxembourg	-0.9	1.2	1.1	0.5	0.2	-0.9	1.2	-0.5	0.5	...	-5.6	8.3	-
27	Hungary	0.0	0.2	0.3	-1.8	-0.6	0.0	0.2	1.2	-1.8	...	-14.4	11.9	
28	Malta	-1.2	1.3	1.6	1.1	0.9	-1.2	1.3	1.3	1.1	...	-13.5	6.4	
29	Netherlands	-0.1	0.0	-0.4	-0.2	0.1	-0.1	0.0	-0.6	-0.2	...	-7.9	6.3	
30	Austria	0.4	0.5	-0.2	0.9	-0.4	0.4	0.5	0.0	0.9	...	-11.3	11.3	-
31	Poland	1.2	1.1	0.0	0.3	-0.2	1.2	1.1	0.8	0.3	...	-9.2	6.8	
32	Portugal	-0.4	-0.8	-1.1	-0.5	-1.3	-0.4	-0.8	-1.5	-0.5	...	-15.1	14.6	
33	Romania	0.3	1.8	-1.4	1.0	1.5	0.3	1.8	-0.7	1.0	...	-9.6	3.7	
34	Slovenia	-0.5	-0.2	0.2	-0.6	-1.8	-0.5	-0.2	0.0	-0.6	...	-9.9	12.6	-
35	Slovakia	0.7	0.5	0.1	0.2	0.2	0.7	0.5	0.8	0.2	...	-7.4	9.2	
36	Finland	-0.2	0.1	-0.4	-0.4	-1.1	-0.2	0.1	0.0	-0.4	...	-6.2	5.0	
37	Sweden	0.3	1.3	-0.1	0.2	0.2	0.3	1.3	-1.4	0.2	...	-8.1	7.5	-

28 rows × 52 columns

GDP by period 2010-2021: percentage change on previous period

Data of GDP: percentage change on previous period by period 2010 (means based from 2010-2021)

The second part of the analysis includes data from the GDP, available from 2010. Therefore, it will be chosen the quarters from 2010 to 2021, and generate a **sentimental feature** which indicates the general opinion of the experts about the change in the GDP and how this impact the index of prices: positive or negative.

Generate a sentimental feature

(CRISP-DM Phase: Data Preparation Phase)

Generate a sentimental feature

It will be added a categorical feature based on the general opinion of the experts in GDP related when the GDP is negative or positive.

Most economists today agree that a small amount of inflation about 1% to 2% is beneficial, and is essential that the GDP of the countries needs to grow. However, if GDP growth is higher than 2.5% to 3.5% could be dangerous, because causes inflation or even worse hyperinflation.

This economic parameter is essential in the index of producer prices of agricultural products (output) and the index of purchase prices of agricultural production (input) for Ireland and all the countries of the EU.

Therefore, **GDP between 0% to 3.5%** could be considered **"positive"**, in another way, out of this range, could be considered **"negative"**.

This **rule will be applied** to this project.

Justification, Please see:

https://www.imf.org/en/Publications/fandd/issues/Series/Back-to-Basics/gross-domestic-product-GDP

https://www.investopedia.com/articles/06/gdpinflation.asp

https://www.investopedia.com/terms/f/farmprices.asp

https://www.kaggle.com/code/kirolosatef/stock-prediction-using-twitter-sentiment-analysis#Load-the-dataset

.

Preparing the data for annual analysis of GDP by period 2010-2021

All data available, for all countries, is by year, therefore, it is necessary to regroup all data about GDP by year instead of a quarter. In order to do that, it will substitute the values of the four(4) quarters by the mean of the GDP per year for each country. That means creating a new feature equal to the mean of the 4 quarters, for example, the GDP for 2010-Q1, 2010-Q2, 2010-Q3, and 2010-Q4, will be substituted by **only one feature per year**, 2010, per country.

It is just for data from **2010 to 2021**.

Rename the columns

In [43]:

```
df_gdp_2010.columns
df_gdp_2010.rename(columns = lambda x: x.replace('-Q1', ''),inplace=True)
df_gdp_2010.rename(columns = lambda x: x.replace('-Q2', ''),inplace=True)
df_gdp_2010.rename(columns = lambda x: x.replace('-Q3', ''),inplace=True)
df_gdp_2010.rename(columns = lambda x: x.replace('-Q4', ''),inplace=True)

df_gdp_2010
```

Out[43]:

	Geo	2010	2010	2010	2010	2011	2011	2011	2011	2012	...	2020	2020	2020	2021
10	European Union: 27 countries	0.1	0.2	-0.1	-0.2	-0.2	0.1	0.2	-0.4	-0.2	...	-11.1	11.5	-0.1	0.2
11	Belgium	0.2	0.3	0.2	0.2	0.0	0.2	0.3	0.3	0.2	...	-11.4	11.7	-0.5	1.4
12	Bulgaria	1.0	0.1	-0.2	0.7	-0.2	1.0	0.1	-0.1	0.7	...	-4.8	3.6	1.8	2.7
13	Czechia	0.3	-0.2	-0.4	-0.2	-0.5	0.3	-0.2	0.1	-0.2	...	-8.8	7.0	1.1	-0.5
14	Denmark	1.0	-1.2	0.1	-0.1	0.1	1.0	-1.2	0.8	-0.1	...	-6.3	5.9	0.0	1.0
15	Germany	0.1	0.9	0.3	0.2	0.2	0.1	0.9	-0.3	0.2	...	-9.5	9.0	0.6	-1.5
16	Estonia	1.9	1.7	0.5	0.6	1.2	1.9	1.7	0.0	0.6	...	-6.8	4.8	2.7	2.6
17	Ireland	1.2	-1.4	-1.0	0.0	1.3	1.2	-1.4	-0.2	0.0	...	-5.6	12.9	-4.6	8.9
18	Greece	-1.8	-2.0	-1.5	-0.3	-1.7	-1.8	-2.0	-4.5	-0.3	...	-13.4	5.3	4.1	3.1

19	Spain	-0.3	-0.6	-0.5	-0.9	-1.0	-0.3	-0.6	-0.7	-0.9	...	-17.8	16.6	-0.1	-0.2
20	France	-0.1	0.5	0.2	0.0	-0.2	-0.1	0.5	0.2	0.0	...	-13.5	18.3	-0.9	0.1
21	Croatia	1.5	-0.4	0.3	-1.2	-0.9	1.5	-0.4	-0.7	-1.2	...	-14.7	5.8	5.6	7.3
22	Italy	0.0	-0.5	-0.5	-1.1	-0.7	0.0	-0.5	-1.0	-1.1	...	-12.1	14.5	-0.8	0.3
23	Cyprus	0.4	-2.1	-1.2	-0.5	-1.8	0.4	-2.1	0.5	-0.5	...	-11.5	6.1	3.9	1.8
24	Latvia	3.4	1.5	1.3	5.1	-0.6	3.4	1.5	0.2	5.1	...	-7.1	6.0	1.6	-0.7
25	Lithuania	1.4	0.9	2.1	0.5	0.6	1.4	0.9	1.1	0.5	...	-5.3	3.7	1.8	2.2
26	Luxembourg	-0.9	1.2	1.1	0.5	0.2	-0.9	1.2	-0.5	0.5	...	-5.6	8.3	-0.4	2.0
27	Hungary	0.0	0.2	0.3	-1.8	-0.6	0.0	0.2	1.2	-1.8	...	-14.4	11.9	1.4	1.1
28	Malta	-1.2	1.3	1.6	1.1	0.9	-1.2	1.3	1.3	1.1	...	-13.5	6.4	4.0	5.9
29	Netherlands	-0.1	0.0	-0.4	-0.2	0.1	-0.1	0.0	-0.6	-0.2	...	-7.9	6.3	0.0	0.0
30	Austria	0.4	0.5	-0.2	0.9	-0.4	0.4	0.5	0.0	0.9	...	-11.3	11.3	-1.9	-1.0
31	Poland	1.2	1.1	0.0	0.3	-0.2	1.2	1.1	0.8	0.3	...	-9.2	6.8	0.1	2.6
32	Portugal	-0.4	-0.8	-1.1	-0.5	-1.3	-0.4	-0.8	-1.5	-0.5	...	-15.1	14.6	0.4	-2.6
33	Romania	0.3	1.8	-1.4	1.0	1.5	0.3	1.8	-0.7	1.0	...	-9.6	3.7	3.4	1.7
34	Slovenia	-0.5	-0.2	0.2	-0.6	-1.8	-0.5	-0.2	0.0	-0.6	...	-9.9	12.6	-0.2	1.2
35	Slovakia	0.7	0.5	0.1	0.2	0.2	0.7	0.5	0.8	0.2	...	-7.4	9.2	0.4	-1.4
36	Finland	-0.2	0.1	-0.4	-0.4	-1.1	-0.2	0.1	0.0	-0.4	...	-6.2	5.0	0.7	-0.2
37	Sweden	0.3	1.3	-0.1	0.2	0.2	0.3	1.3	-1.4	0.2	...	-8.1	7.5	-0.4	1.6

28 rows × 52 columns

In [44]:
```
# Period under analysis: from 2010 to 2021

years = ('2010', '2011', '2012',  '2013', '2014', '2015', '2016', '2017', '2018',  '2019
```

In [45]:
```
# Create a new DF for the means of the GDP by countries

df_gdp = df_gdp_2010.copy()
```

Create a new DF for the means of the GDP by countries

In [46]:
```
# Transform data in order to create the DF with means of GDP by countries

for y in years:
    i = 'y'+y
    df_gdp[i] = df_gdp[y].mean(axis=1) # create columns with means
    df_gdp.drop([y],axis=1, inplace=True)
    df_gdp.rename(columns = lambda x: x.replace('y', ''),inplace=True)
    df_gdp[y] = df_gdp[y].map('{:,.2f}'.format) # format values
    df_gdp[y] = df_gdp[y].astype(float)
```

In [47]:
```
# In order to maintain coherence between the years: period from 2010 to 2021,
# it will be deleted the columns about the year 2022.

df_gdp.drop(['2022'],axis=1, inplace=True) # year 2022
```

In [48]:
```
df_gdp
```

Out[48]:

	Geo	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
10	European Union: 27 countries	0.00	-0.08	-0.23	0.20	0.45	0.55	0.53	0.80	0.38	0.38	-0.65	1.25
11	Belgium	0.22	0.20	0.10	0.20	0.47	0.45	0.33	0.40	0.57	0.53	-0.78	1.50
12	Bulgaria	0.40	0.20	-0.00	0.03	0.47	0.85	0.75	0.65	0.82	0.95	-0.80	1.85
13	Czechia	-0.12	-0.08	-0.30	0.40	0.68	1.27	0.55	1.45	0.62	0.70	-0.97	0.85
14	Denmark	-0.05	0.18	-0.03	0.37	0.57	0.42	1.02	0.55	0.53	0.25	-0.17	1.60
15	Germany	0.38	0.23	0.07	0.38	0.57	0.28	0.48	0.90	0.02	0.22	-0.33	0.30
16	Estonia	1.17	1.20	0.80	0.00	1.20	0.28	0.93	1.45	0.90	0.80	0.35	1.75
17	Ireland	-0.30	0.23	0.08	0.70	2.10	5.78	2.30	2.00	0.82	1.57	1.33	3.38
18	Greece	-1.40	-2.50	-1.10	-0.03	-0.05	0.05	-0.02	0.10	0.52	0.27	-1.48	2.15
19	Spain	-0.57	-0.65	-0.77	-0.05	0.62	1.02	0.62	0.72	0.55	0.35	-1.70	1.65
20	France	0.15	0.10	-0.03	0.35	0.17	0.25	0.30	0.72	0.38	0.27	-0.40	1.27
21	Croatia	0.05	-0.12	-0.65	-0.03	0.07	0.55	1.17	0.72	0.75	0.57	-1.10	2.98
22	Italy	-0.53	-0.55	-0.75	-0.25	-0.02	0.38	0.35	0.45	0.07	-0.08	-1.07	1.60
23	Cyprus	-0.85	-0.75	-1.38	-1.28	-0.20	1.43	1.68	1.20	1.40	1.20	-0.53	1.60
24	Latvia	2.83	1.12	1.57	0.62	0.38	0.88	0.57	0.88	1.20	0.30	0.03	0.63
25	Lithuania	1.23	1.00	0.90	0.93	0.62	0.55	0.82	1.00	1.05	1.05	0.15	1.40
26	Luxembourg	0.47	0.00	0.88	0.30	1.35	0.00	1.43	0.30	0.20	0.65	0.28	1.00
27	Hungary	-0.33	0.20	-0.55	0.90	0.90	0.90	0.55	1.25	1.27	1.05	-0.40	1.83
28	Malta	0.70	0.57	1.00	1.38	2.22	2.27	0.75	2.42	1.65	1.75	-1.82	3.23
29	Netherlands	-0.17	-0.15	-0.30	0.32	0.40	0.30	0.75	0.72	0.40	0.47	-0.78	1.52
30	Austria	0.40	0.12	0.05	0.18	0.10	0.30	0.60	0.55	0.68	0.00	-1.10	1.50
31	Poland	0.65	0.73	-0.00	0.47	1.03	1.07	0.95	1.27	1.35	0.93	-0.40	2.12
32	Portugal	-0.70	-1.00	-1.12	0.55	0.15	0.38	0.72	0.80	0.65	0.70	-1.12	1.62
33	Romania	0.43	0.73	0.40	0.45	0.98	0.82	0.75	2.05	1.38	0.72	-0.45	1.12
34	Slovenia	-0.27	-0.62	-1.05	0.60	0.43	0.47	1.00	1.55	1.00	0.82	-0.60	2.60
35	Slovakia	0.38	0.55	0.10	0.35	0.88	1.22	0.30	0.93	0.90	0.45	-0.15	0.28
36	Finland	-0.23	-0.30	-0.50	-0.08	-0.10	0.32	0.70	0.82	0.02	0.38	-0.15	0.75
37	Sweden	0.42	0.10	-0.10	0.57	0.77	1.12	0.28	0.72	0.48	0.42	-0.25	1.43

Annual evolution of GDP by period 2010-2021: Irland vs EU

In [49]:

```
# melt_pivot

# calling function melt_pivot

df_gdp_t = melt_pivot(df_gdp, 'GDP means')

df_gdp_t
```

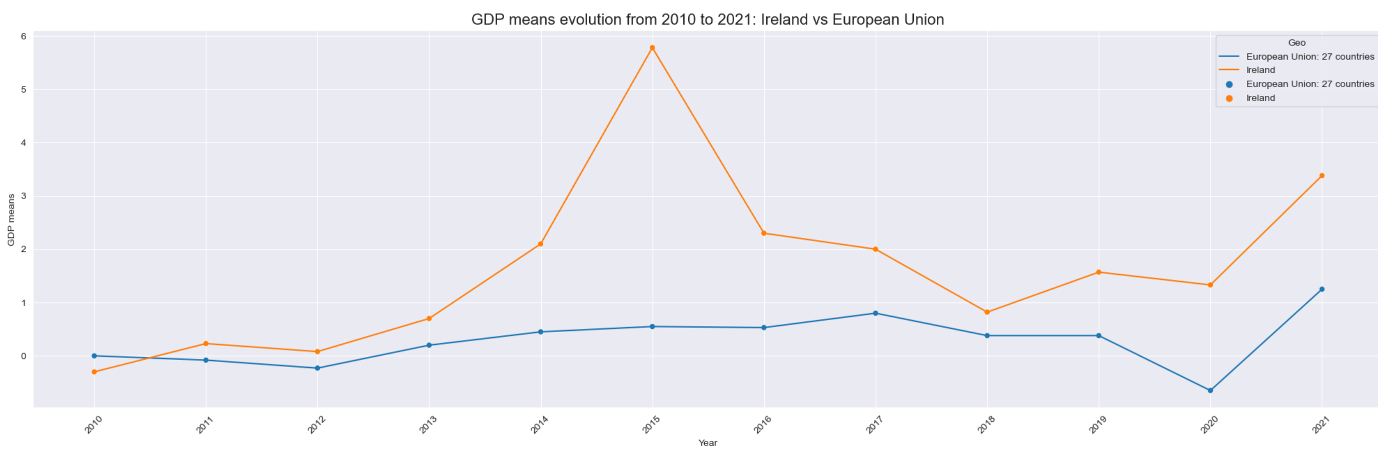
Out[49]:

	Geo	Year	GDP means
0	European Union: 27 countries	2010	0.00

1	Belgium	2010	0.22
2	Bulgaria	2010	0.40
3	Czechia	2010	-0.12
4	Denmark	2010	-0.05
...	...	...	...
331	Romania	2021	1.12
332	Slovenia	2021	2.60
333	Slovakia	2021	0.28
334	Finland	2021	0.75
335	Sweden	2021	1.43

336 rows × 3 columns

In [50]:
```
df_tmp = df_gdp_t[(df_gdp_t['Geo']=='Ireland') | (df_gdp_t['Geo']=='European Union: 27 c

plt.figure(figsize=(25,7))
sns.lineplot(x=df_tmp["Year"],y=df_tmp['GDP means'],hue= df_tmp['Geo'])
sns.scatterplot(x=df_tmp["Year"],y=df_tmp['GDP means'],hue=df_tmp['Geo'])
plt.xticks(rotation=45);
plt.title("GDP means evolution from 2010 to 2021: Ireland vs European Union",fontsize=16
plt.show()
```



Generate a sentimental feature

GDP is an economic parameter essential in the index of producer prices of agricultural products (output) and the index of purchase prices of agricultural production (input) for Ireland and all the countries of the EU.

A Sentimental feature will be created according this creteria:

1- GDP between 0% to 3.5%could be considered **"positive"**

2- GDP out of this range, could be considered **"negative"**

In [51]:
```
# create a new DF

df_gdp_emo = df_gdp.copy()
df_gdp_emo.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 28 entries, 10 to 37
Data columns (total 13 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Geo     28 non-null     object 
 1   2010    28 non-null     float64
 2   2011    28 non-null     float64
 3   2012    28 non-null     float64
 4   2013    28 non-null     float64
 5   2014    28 non-null     float64
 6   2015    28 non-null     float64
 7   2016    28 non-null     float64
 8   2017    28 non-null     float64
 9   2018    28 non-null     float64
 10  2019    28 non-null     float64
 11  2020    28 non-null     float64
 12  2021    28 non-null     float64
dtypes: float64(12), object(1)
memory usage: 3.0+ KB
```

In [52]:

```python
# Period under analysis: from 2010 to 2021
years = ('2010', '2011', '2012',  '2013', '2014', '2015', '2016', '2017', '2018',  '2019

for y in years:
#    i='emo'+y
     df_gdp_emo[y] = df_gdp[y].apply(lambda x: 'pos' if 0.0 > x <= 3.5 else 'neg')

df_gdp_emo
```

Out[52]:

	Geo	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
10	European Union: 27 countries	neg	pos	pos	neg	neg	neg	neg	neg	neg	neg	pos	neg
11	Belgium	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	pos	neg
12	Bulgaria	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	pos	neg
13	Czechia	pos	pos	pos	neg	neg	neg	neg	neg	neg	neg	pos	neg
14	Denmark	pos	neg	pos	neg	neg	neg	neg	neg	neg	neg	pos	neg
15	Germany	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	pos	neg
16	Estonia	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg
17	Ireland	pos	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg
18	Greece	pos	pos	pos	pos	pos	neg	pos	neg	neg	neg	pos	neg
19	Spain	pos	pos	pos	pos	neg	neg	neg	neg	neg	neg	pos	neg
20	France	neg	neg	pos	neg	neg	neg	neg	neg	neg	neg	pos	neg
21	Croatia	neg	pos	pos	pos	neg	neg	neg	neg	neg	neg	pos	neg
22	Italy	pos	pos	pos	pos	pos	neg	neg	neg	neg	pos	pos	neg
23	Cyprus	pos	pos	pos	pos	pos	neg	neg	neg	neg	neg	pos	neg
24	Latvia	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg
25	Lithuania	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg
26	Luxembourg	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg
27	Hungary	pos	neg	pos	neg	neg	neg	neg	neg	neg	neg	pos	neg
28	Malta	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	pos	neg
29	Netherlands	pos	pos	pos	neg	neg	neg	neg	neg	neg	neg	pos	neg

30	Austria	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	pos	neg
31	Poland	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	pos	neg
32	Portugal	pos	pos	pos	neg	neg	neg	neg	neg	neg	neg	pos	neg
33	Romania	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	pos	neg
34	Slovenia	pos	pos	pos	neg	neg	neg	neg	neg	neg	neg	pos	neg
35	Slovakia	neg	neg	neg	neg	neg	neg	neg	neg	neg	neg	pos	neg
36	Finland	pos	pos	pos	pos	pos	neg	neg	neg	neg	neg	pos	neg
37	Sweden	neg	neg	pos	neg	neg	neg	neg	neg	neg	neg	pos	neg

The index of purchase prices of the means of agricultural production (input)

In [53]:
```
df_ina
```

Out[53]:

	Geo	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	2012	2013	2014	20
0	European Union: 27 countries	100.0	101.4	99.7	99.4	101.5	100.7	101.2	106.1	116.8	...	108.5	108.4	104.8	102
1	Belgium	100.0	100.2	99.1	97.7	96.2	97.6	103.1	111.7	122.3	...	111.1	110.2	101.7	99
2	Bulgaria	100.0	100.0	100.0	100.0	100.0	100.0	95.9	98.7	100.6	...	110.5	109.0	106.0	103
3	Czechia	100.0	100.1	97.4	96.0	99.8	97.7	98.6	101.1	104.1	...	106.4	108.1	106.0	103
4	Denmark	100.0	103.5	102.2	99.0	101.2	101.0	100.6	107.0	121.1	...	108.9	112.4	111.6	109
5	Germany	100.0	102.1	100.3	99.5	101.3	99.8	102.2	107.6	118.3	...	110.8	111.1	106.5	104
6	Estonia	100.0	100.0	100.0	100.0	100.0	100.0	101.5	103.6	103.4	...	100.0	100.0	100.0	100
7	Ireland	100.0	100.4	97.5	96.0	97.1	98.9	100.5	103.8	115.4	...	111.1	113.4	108.8	106
8	Greece	100.0	98.4	97.4	97.9	102.2	103.8	100.5	103.9	110.6	...	107.0	107.3	106.0	104
9	Spain	100.0	100.0	97.4	95.7	96.5	95.1	99.7	104.4	115.3	...	110.3	108.7	105.3	105
10	France	100.0	101.3	99.9	99.0	100.3	100.3	100.9	105.1	119.4	...	107.3	107.6	104.6	102
11	Croatia	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	...	111.7	108.5	99.2	96
12	Italy	100.0	102.1	100.7	100.8	103.4	99.1	101.2	105.8	114.4	...	105.5	106.3	104.2	101
13	Cyprus	100.0	100.0	100.0	100.0	128.6	136.0	104.9	110.7	115.4	...	96.4	106.7	107.3	110
14	Latvia	100.0	99.2	97.9	99.1	100.9	111.3	102.8	106.2	109.3	...	108.9	109.3	106.4	104
15	Lithuania	100.0	100.0	100.0	100.0	100.0	100.0	111.3	113.4	131.6	...	120.3	114.8	109.1	112
16	Luxembourg	100.0	101.1	100.3	99.1	100.2	97.7	99.6	103.1	108.5	...	106.0	104.5	102.6	100
17	Hungary	100.0	102.5	98.3	99.7	100.8	97.4	101.9	105.6	115.3	...	108.9	109.3	106.2	104
18	Malta	100.0	98.3	96.7	90.7	93.2	93.8	100.7	105.3	119.4	...	108.8	108.8	104.8	102
19	Netherlands	100.0	100.9	98.2	97.7	97.9	97.9	104.4	110.5	116.5	...	107.3	107.1	101.9	100
20	Austria	100.0	99.6	97.6	98.2	99.6	98.8	100.7	104.2	110.1	...	105.1	104.9	102.9	101
21	Poland	100.0	101.2	101.6	103.9	107.8	108.0	99.0	102.7	109.4	...	109.8	109.4	107.0	104
22	Portugal	100.0	100.1	96.4	94.5	96.1	97.3	99.7	104.9	116.5	...	108.6	111.0	107.7	105
23	Romania	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	112.2	...	109.7	109.4	104.9	10
24	Slovenia	100.0	103.1	98.9	98.1	103.0	101.9	100.8	105.5	117.9	...	109.8	110.2	105.8	103

25	Slovakia	100.0	100.0	100.0	100.0	89.2	87.5	99.2	101.9	107.2	...	109.4	107.9	101.1	9
26	Finland	100.0	99.6	98.2	98.2	100.8	103.5	102.3	105.4	117.3	...	108.5	108.2	105.3	10
27	Sweden	100.0	102.3	102.3	102.1	104.8	106.0	101.6	106.4	117.4	...	106.2	106.1	105.8	10

28 rows × 23 columns

The index of producer prices of agricultural products (output)

In [54]:

```
df_outa
```

Out[54]:

	Geo	2000	2001	2002	2003	2004	2005	2006	2007	2008	...	2012	2013	2014	20
0	European Union: 27 countries	100.0	102.7	97.1	98.2	96.1	92.0	102.7	110.4	112.0	...	111.0	112.3	104.4	10
1	Belgium	100.0	102.0	91.0	92.5	92.7	91.4	111.2	109.2	103.6	...	108.3	111.7	92.6	8
2	Bulgaria	100.0	108.0	90.2	90.1	94.3	76.6	100.7	117.5	117.3	...	135.1	110.8	107.9	11
3	Czechia	100.0	106.1	94.4	91.0	94.7	87.2	98.6	108.8	110.7	...	119.1	122.9	117.7	11
4	Denmark	100.0	105.0	92.4	86.5	88.0	85.2	102.1	104.2	112.0	...	117.2	121.4	107.4	10
5	Germany	100.0	105.2	96.9	97.1	93.9	91.3	105.4	115.2	116.3	...	114.0	113.5	103.6	9
6	Estonia	100.0	100.0	100.0	100.0	117.4	118.0	98.1	107.2	101.3	...	100.0	100.0	100.0	10
7	Ireland	100.0	100.4	91.8	87.9	87.9	86.4	101.1	106.5	111.4	...	117.5	126.9	116.2	11
8	Greece	100.0	102.4	105.5	110.9	105.6	105.2	104.1	111.7	103.6	...	97.5	100.0	99.6	10
9	Spain	100.0	100.2	94.2	96.3	94.4	94.2	94.9	97.5	96.6	...	105.9	107.5	99.7	10
10	France	100.0	101.7	96.3	97.7	93.8	88.1	103.5	113.3	116.4	...	113.6	115.1	109.4	10
11	Croatia	100.0	100.0	100.0	100.0	100.0	100.0	97.2	105.3	99.6	...	110.1	100.9	95.7	9
12	Italy	100.0	103.4	102.0	105.4	101.1	93.7	102.8	108.7	111.9	...	109.2	112.0	107.0	10
13	Cyprus	100.0	100.0	100.0	100.0	109.0	107.6	103.4	110.3	119.7	...	112.5	112.8	110.5	11
14	Latvia	100.0	105.9	102.1	97.3	108.6	114.9	105.9	117.3	104.0	...	114.7	109.0	99.4	9
15	Lithuania	100.0	113.0	112.1	101.1	100.9	111.2	102.4	113.4	112.3	...	114.6	116.1	101.8	9
16	Luxembourg	100.0	99.4	95.2	93.8	93.8	89.8	100.0	108.0	107.0	...	107.7	109.4	105.9	9
17	Hungary	100.0	97.2	90.8	92.0	81.4	79.2	108.8	134.0	113.6	...	126.6	115.1	108.2	10
18	Malta	100.0	106.7	104.9	96.9	89.7	86.6	97.7	104.2	103.4	...	107.0	107.2	97.4	10
19	Netherlands	100.0	100.9	95.0	94.0	88.0	87.9	107.6	110.5	107.7	...	103.0	107.4	100.6	9
20	Austria	100.0	104.3	97.8	96.9	94.5	93.5	105.1	111.3	111.5	...	106.3	105.6	99.6	9
21	Poland	100.0	96.5	88.3	89.3	94.4	90.6	104.6	117.6	111.3	...	115.7	112.3	104.7	10
22	Portugal	100.0	106.3	96.5	100.4	97.2	93.1	101.7	103.1	103.1	...	97.2	101.8	96.6	9
23	Romania	100.0	104.3	106.6	100.6	108.6	93.1	99.5	115.0	122.4	...	116.8	118.4	104.0	10
24	Slovenia	100.0	100.4	94.2	92.0	87.8	86.8	103.1	107.8	116.9	...	108.7	114.7	108.5	10
25	Slovakia	100.0	100.8	96.3	84.4	80.3	76.2	95.9	103.3	104.8	...	116.4	109.2	100.9	9
26	Finland	100.0	102.5	99.0	93.3	96.0	92.4	103.7	107.6	113.1	...	113.1	119.1	103.0	9
27	Sweden	100.0	102.4	97.7	94.0	91.6	89.5	103.7	116.0	121.7	...	104.0	106.0	102.2	10

28 rows × 23 columns

End GDP

Data preparation: period under study: 2010 to 2021

Organize the data by years-countries: period 2010 to 2021

(CRISP-DM Phase: Data Preparation Phase)

Melt all df's: df_gdp, df_gdp_emo, df_ina, df_outa

Period under study: 2010 to 2021

In [55]:
```
df_gdp_t
```

Out[55]:

	Geo	Year	GDP means
0	European Union: 27 countries	2010	0.00
1	Belgium	2010	0.22
2	Bulgaria	2010	0.40
3	Czechia	2010	-0.12
4	Denmark	2010	-0.05
...	...	...	...
331	Romania	2021	1.12
332	Slovenia	2021	2.60
333	Slovakia	2021	0.28
334	Finland	2021	0.75
335	Sweden	2021	1.43

336 rows × 3 columns

In [ ]:

In [56]:
```
# Dataframe with data of Expenditure index

# melt df_ina for the period under study

# calling function melt_pivot

df_ina[['Geo', '2010', '2011', '2012',  '2013', '2014', '2015', '2016', '2017', '2018',

df_ina_t = melt_pivot(df_ina[['Geo', '2010', '2011', '2012',  '2013', '2014', '2015', '2
        'Expenditure_Index')
df_ina_t
```

Out[56]:

	Geo	Year	Expenditure_Index
0	European Union: 27 countries	2010	107.70

1	Belgium	2010	104.80
2	Bulgaria	2010	97.90
3	Czechia	2010	93.80
4	Denmark	2010	108.70
...	...	...	...
331	Romania	2021	107.23
332	Slovenia	2021	106.88
333	Slovakia	2021	95.70
334	Finland	2021	105.16
335	Sweden	2021	107.61

336 rows × 3 columns

In [57]:

```
# Dataframe with data of Price index

# melt df_outa for the period under study

# calling function melt_pivot

df_outa[['Geo', '2010', '2011', '2012',  '2013', '2014', '2015', '2016', '2017', '2018',

df_outa_t = melt_pivot(df_outa[['Geo', '2010', '2011', '2012',  '2013', '2014', '2015',
        'Price_Index')
df_outa_t
```

Out[57]:

	Geo	Year	Price_Index
0	European Union: 27 countries	2010	104.10
1	Belgium	2010	97.50
2	Bulgaria	2010	98.10
3	Czechia	2010	88.70
4	Denmark	2010	100.50
...	...	...	...
331	Romania	2021	123.11
332	Slovenia	2021	109.37
333	Slovakia	2021	104.06
334	Finland	2021	101.74
335	Sweden	2021	117.58

336 rows × 3 columns

In [ ]:

In [58]:

```
# Dataframe with data of emotional feature about GDP

# it needs to convert the values 'pos' and 'neg' into an integer in order to prepare the

# Period under analysis: from 2010 to 2021
years = ('2010', '2011', '2012',  '2013', '2014', '2015', '2016', '2017', '2018',  '2019
```

```python
for y in years:
#     i='emo'+y
    df_gdp_emo[y] = df_gdp_emo[y].apply(lambda x: 1 if x=='pos' else -1)

df_gdp_emo

# lambda x: 1 if x>0 else 0 if x ==0 else -1
```

Out[58]:

	Geo	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021
10	European Union: 27 countries	-1	1	1	-1	-1	-1	-1	-1	-1	-1	1	-1
11	Belgium	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1
12	Bulgaria	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1
13	Czechia	1	1	1	-1	-1	-1	-1	-1	-1	-1	1	-1
14	Denmark	1	-1	1	-1	-1	-1	-1	-1	-1	-1	1	-1
15	Germany	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1
16	Estonia	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
17	Ireland	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
18	Greece	1	1	1	1	1	-1	1	-1	-1	-1	1	-1
19	Spain	1	1	1	1	-1	-1	-1	-1	-1	-1	1	-1
20	France	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	1	-1
21	Croatia	-1	1	1	1	-1	-1	-1	-1	-1	-1	1	-1
22	Italy	1	1	1	1	1	-1	-1	-1	-1	1	1	-1
23	Cyprus	1	1	1	1	1	-1	-1	-1	-1	-1	1	-1
24	Latvia	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
25	Lithuania	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
26	Luxembourg	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
27	Hungary	1	-1	1	-1	-1	-1	-1	-1	-1	-1	1	-1
28	Malta	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1
29	Netherlands	1	1	1	-1	-1	-1	-1	-1	-1	-1	1	-1
30	Austria	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1
31	Poland	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1
32	Portugal	1	1	1	-1	-1	-1	-1	-1	-1	-1	1	-1
33	Romania	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1
34	Slovenia	1	1	1	-1	-1	-1	-1	-1	-1	-1	1	-1
35	Slovakia	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1
36	Finland	1	1	1	1	1	-1	-1	-1	-1	-1	1	-1
37	Sweden	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	1	-1

In [59]:

```python
# melt df with emotional feature

# calling function melt_pivot

df_gdp_emo_t = melt_pivot(df_gdp_emo, 'emo')
```

```
df_gdp_emo_t.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 336 entries, 0 to 335
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Geo     336 non-null    object 
 1   Year    336 non-null    object 
 2   emo     336 non-null    float64
dtypes: float64(1), object(2)
memory usage: 8.0+ KB
```

Join data: df_gdp, df_gdp_emo, df_ina, df_outa just in one df and then, melt in order to show like time series format

In [60]:

```
from functools import reduce

# Merge the DF's Using Inner Join

df_final = reduce(lambda left, right:      # Merge three DF
                     pd.merge(left , right,
                              on = ['Geo','Year'],
                              how = 'outer'),
                     [df_gdp_t,
                      df_gdp_emo_t,
                      df_ina_t,
                      df_outa_t])
df_final
```

Out[60]:

	Geo	Year	GDP means	emo	Expenditure_Index	Price_Index
0	European Union: 27 countries	2010	0.00	-1.0	107.70	104.10
1	Belgium	2010	0.22	-1.0	104.80	97.50
2	Bulgaria	2010	0.40	-1.0	97.90	98.10
3	Czechia	2010	-0.12	1.0	93.80	88.70
4	Denmark	2010	-0.05	1.0	108.70	100.50
...	...	...	...	...	...	...
331	Romania	2021	1.12	-1.0	107.23	123.11
332	Slovenia	2021	2.60	-1.0	106.88	109.37
333	Slovakia	2021	0.28	-1.0	95.70	104.06
334	Finland	2021	0.75	-1.0	105.16	101.74
335	Sweden	2021	1.43	-1.0	107.61	117.58

336 rows × 6 columns

In [61]:

```
df_final['Year'] = df_final['Year'].astype(int)
df_final['emo'] = df_final['emo'].astype(int)
df_final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 336 entries, 0 to 335
Data columns (total 6 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Geo                336 non-null    object 
```

```
 1   Year                336 non-null    int64
 2   GDP means           336 non-null    float64
 3   emo                 336 non-null    int64
 4   Expenditure_Index   336 non-null    float64
 5   Price_Index         336 non-null    float64
dtypes: float64(3), int64(2), object(1)
memory usage: 18.4+ KB
```

Data for visualization

In [62]:
```
df_vs = df_final.copy()
```

EDA and statistical analysis

(CRISP-DM Phase: Data Understanding Phase)

General visualization of distribution and relation using scatter plot and histogram

In [63]:
```
# Select only data from Ireland to compare with the EU in general
df_final.loc[df_final['Geo'].isin(['European Union: 27 countries','Ireland'])].head()
```

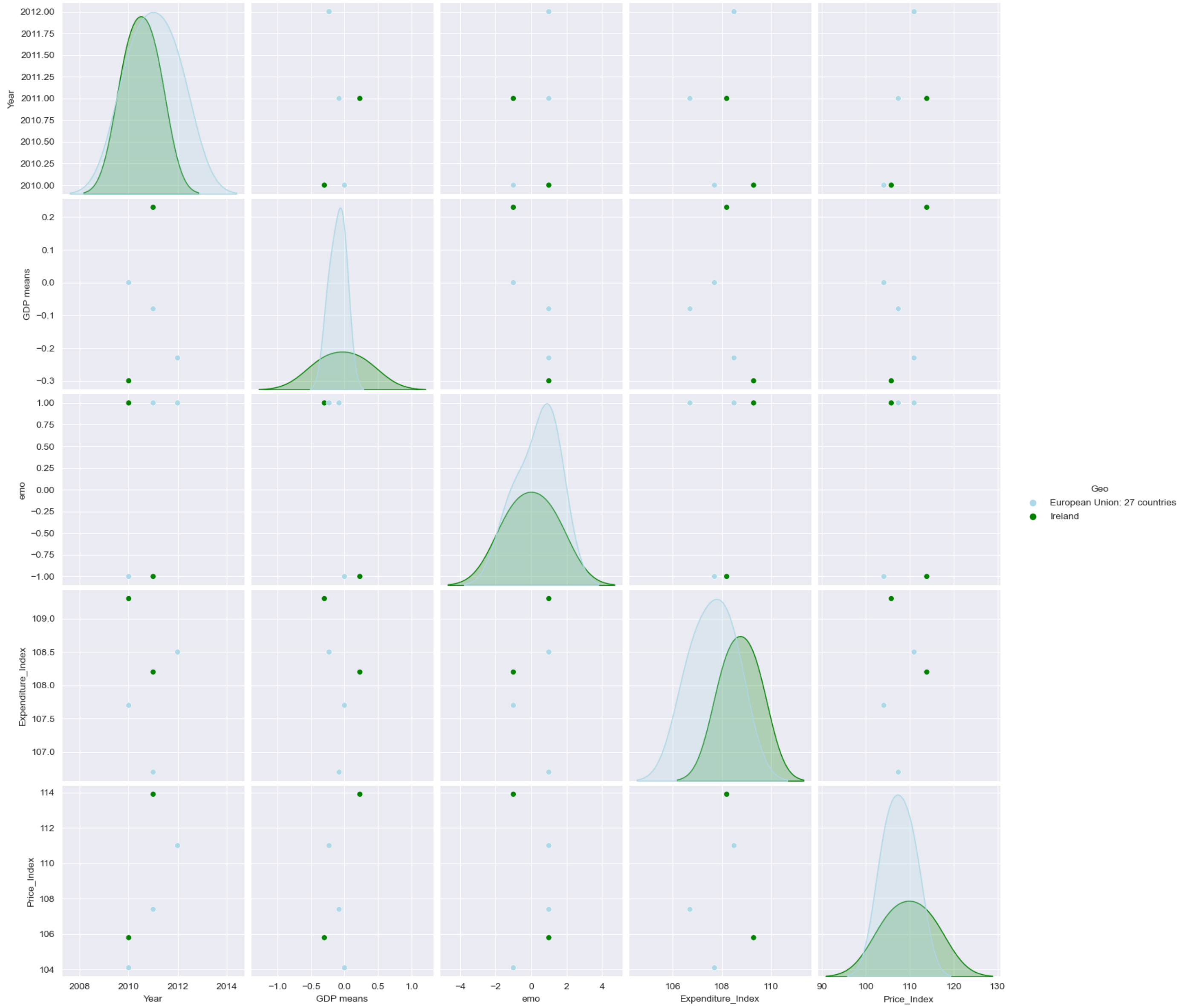
Out[63]:

	Geo	Year	GDP means	emo	Expenditure_Index	Price_Index
0	European Union: 27 countries	2010	0.00	-1	107.7	104.1
7	Ireland	2010	-0.30	1	109.3	105.8
28	European Union: 27 countries	2011	-0.08	1	106.7	107.4
35	Ireland	2011	0.23	-1	108.2	113.9
56	European Union: 27 countries	2012	-0.23	1	108.5	111.0

In [ ]:
```

```

In [64]:
```
# Visualize the data using scatter plot and histogram

fig = plt.figure(figsize=(30, 15))

fig.suptitle('Full visual comparative Price_Index between Irelnad and EU in general', fo

ax = sns.set_palette(["lightblue", "green"])
ax = sns.pairplot(data=df_final.loc[df_final['Geo'].isin(['Ireland', 'European Union: 27
             height=3, hue='Geo'
            )

plt.show()
```

```
<Figure size 3000x1500 with 0 Axes>
```



Descriptive statistics of indexes by countries, specifically from Ireland and the EU in general.

In [65]:
```
# Statistics Ireland

df_final[['Geo', 'GDP means','Expenditure_Index','Price_Index']][df_final['Geo'].isin(['
```

Out[65]:

	GDP means	Expenditure_Index	Price_Index
count	12.000000	12.000000	12.000000
mean	1.665833	105.635000	109.423333
std	1.672730	4.907163	8.598597
min	-0.300000	98.500000	95.310000
25%	0.582500	101.632500	103.295000
50%	1.450000	106.840000	108.960000
75%	2.150000	108.925000	114.475000
max	5.780000	113.400000	126.900000

In [66]:
```
df_final
```

Out[66]:

	Geo	Year	GDP means	emo	Expenditure_Index	Price_Index
0	European Union: 27 countries	2010	0.00	-1	107.70	104.10
1	Belgium	2010	0.22	-1	104.80	97.50
2	Bulgaria	2010	0.40	-1	97.90	98.10
3	Czechia	2010	-0.12	1	93.80	88.70
4	Denmark	2010	-0.05	1	108.70	100.50
...	...	...	...	...	...	...
331	Romania	2021	1.12	-1	107.23	123.11
332	Slovenia	2021	2.60	-1	106.88	109.37
333	Slovakia	2021	0.28	-1	95.70	104.06
334	Finland	2021	0.75	-1	105.16	101.74
335	Sweden	2021	1.43	-1	107.61	117.58

336 rows × 6 columns

In [67]:

```
# Statistics EU (in General)
df_final[['Geo', 'GDP means','Expenditure_Index','Price_Index']][df_final['Geo'].isin(['
```

Out[67]:

	GDP means	Expenditure_Index	Price_Index
count	12.000000	12.000000	12.000000
mean	0.298333	103.084167	105.129167
std	0.498267	4.276559	4.011136
min	-0.650000	97.840000	98.460000
25%	-0.020000	99.227500	102.842500
50%	0.380000	103.750000	104.035000
75%	0.535000	106.950000	107.790000
max	1.250000	108.500000	112.300000

Maximum and minimum values of the Price Index and expenditure Index for the EU countries

In [68]:

```
# Maximum and minimum values of the Price Index and expenditure Index for the EU countri

df_final.groupby('Geo').aggregate({'Price_Index': ['min', 'max', 'mean'],
                                   'Expenditure_Index': ['min', 'max', 'mean']}
                                 ).sort_values(by=('Price_Index', 'mean'), ascending=Fal


# The Price index and also the Expenditure index of Ireland, by mean, are the ones of
# the highest values in the EU countries.

# However, Ireland has a Price Index, by means, below Poland and Romania
# despite the fact that the expenditure index is higher than both countries.
```

Out[68]:

	Price_Index			Expenditure_Index		
	min	max	mean	min	max	mean
Geo						

Poland	100.52	120.21	111.825000	97.68	109.8	103.753333
Romania	101.50	123.11	110.207500	96.19	109.7	103.805000
Ireland	95.31	126.90	109.423333	98.50	113.4	105.635000
Hungary	96.01	126.60	108.480000	95.20	109.3	102.673333
Sweden	100.20	117.58	107.972500	97.88	108.6	104.205000
France	99.83	115.10	106.685833	96.95	107.6	102.275000
Latvia	91.90	121.45	106.638333	95.29	109.3	101.278333
Bulgaria	92.68	135.10	106.454167	94.95	110.5	102.148333
Germany	98.31	114.00	106.005000	97.91	111.8	104.406667
Cyprus	96.66	113.30	105.821667	93.16	110.3	99.290833
Slovenia	98.34	114.70	105.605833	97.80	110.2	104.190000
Italy	97.10	112.00	105.566667	99.70	111.0	103.979167
Denmark	94.66	121.40	105.317500	99.86	112.4	105.545000
European Union: 27 countries	98.46	112.30	105.129167	97.84	108.5	103.084167
Czechia	88.70	122.90	104.322500	92.46	108.1	99.285833
Lithuania	92.64	118.90	103.945000	84.02	120.3	101.764167
Finland	96.63	119.10	103.668333	97.04	108.5	103.247500
Belgium	88.00	111.70	103.056667	97.09	111.1	102.977500
Luxembourg	94.20	109.40	102.571667	98.18	106.0	101.556667
Slovakia	88.00	116.40	102.400000	90.81	109.4	98.830833
Austria	96.10	108.10	102.317500	96.43	106.9	101.068333
Netherlands	96.39	107.40	102.110833	95.18	107.8	102.172500
Malta	96.40	107.20	102.000833	96.96	110.6	102.890000
Estonia	92.40	113.17	101.671667	93.11	100.0	96.924167
Portugal	94.10	109.83	101.300833	97.04	113.8	104.220000
Croatia	92.10	110.10	100.762500	91.93	111.7	100.129167
Greece	97.50	108.40	100.701667	98.10	107.3	102.999167
Spain	89.50	107.50	99.574167	95.54	110.3	102.422500

In general opinion, Netherlands and Belgium have a similar economy to Ireland so for the purpose of comparison, it would be to obtain statistics values of those countries.

In [69]:

```
# Price Index comparison

C = ['European Union: 27 countries','Ireland','Belgium','Netherlands']
print('Price Index comparison ')
for c in C:
    print('\n',c,'has mean : %.2f' % df_final[df_final['Geo'].isin([c])].Price_Index.mea
    print('', c, 'has std : %.2f' % df_final[df_final['Geo'].isin([c])].Price_Index.std(
```

```
Price Index comparison

 European Union: 27 countries has mean : 105.13
 European Union: 27 countries has std : 4.01
```

```
 Ireland has mean : 109.42
 Ireland has std : 8.60

 Belgium has mean : 103.06
 Belgium has std : 7.61

 Netherlands has mean : 102.11
 Netherlands has std : 3.17
```

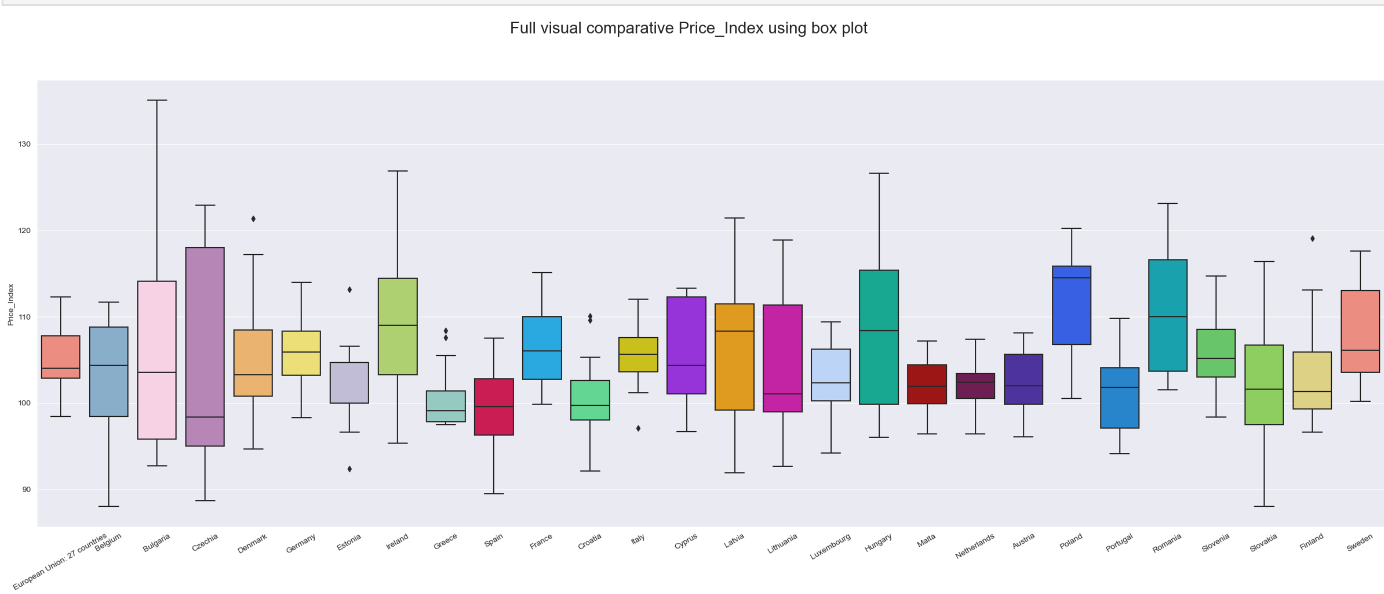
In [70]:

```python
# Expenditure Index comparison

C = ['European Union: 27 countries','Ireland','Belgium','Netherlands']
print('Expenditure Index comparison ')
for c in C:
    print('\n',c,'has mean : %.2f' % df_final[df_final['Geo'].isin([c])].Price_Index.mea
    print('', c, 'has std : %.2f' % df_final[df_final['Geo'].isin([c])].Price_Index.std(
```

```
Expenditure Index comparison

 European Union: 27 countries has mean : 105.13
 European Union: 27 countries has std : 4.01

 Ireland has mean : 109.42
 Ireland has std : 8.60

 Belgium has mean : 103.06
 Belgium has std : 7.61

 Netherlands has mean : 102.11
 Netherlands has std : 3.17
```

In [71]:

```python
fig = plt.figure(figsize=(30, 10))

fig.suptitle('Full visual comparative Price_Index using box plot', fontsize=20)

ax= sns.boxplot(  y=df_final['Price_Index'], x=df_final['Geo'],
                palette=["#fd7f6f", "#7eb0d5", "#fdcce5", "#bd7ebe", "#ffb55a", "#ffee65
                        "#e60049", "#0bb4ff", "#50e991", "#e6d800", "#9b19f5", "#ffa300"
                        "#b30000", "#7c1158", "#4421af", "#1a53ff", "#0d88e6", "#00b7c7"
               )  # In order to has no equal colors for the countries

ax.tick_params(axis="x", rotation=30)

plt.show()
```



Comparison between the Index of prices (output) and the index of expenditure (input) incurred by farmers between Ireland and the EU (in general) and also, specifically with Belgium and Netherlands.

The EU Agricultural Price Indices (API) comprise:

1- the index of purchase prices of the means of agricultural production (**input**)

Index of variation of the **expenditure** incurred by farmers in purchasing the means of production (goods and services as well as investment goods), including crop products from other agricultural units for intermediate consumption, over a given period.

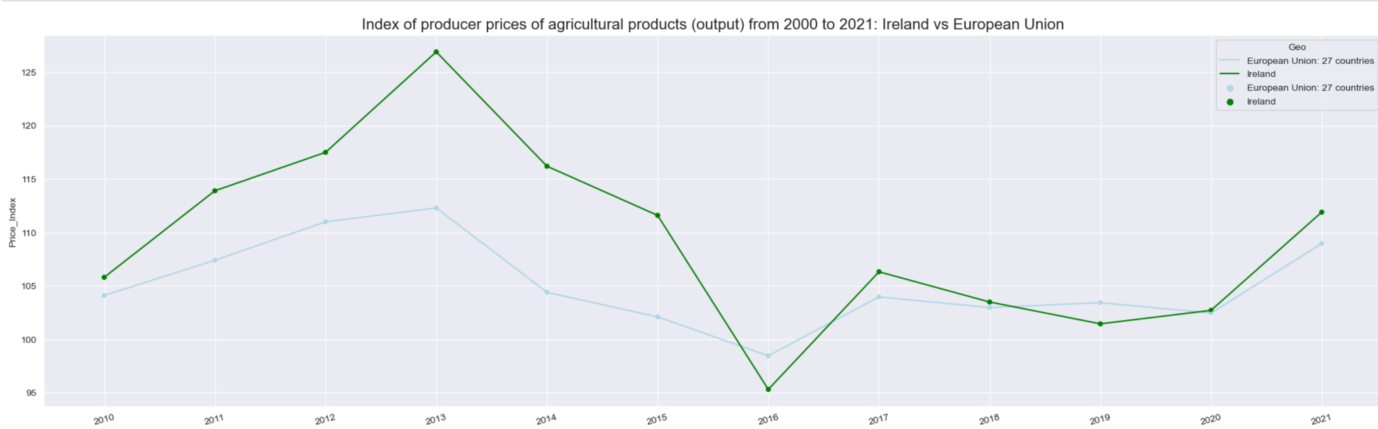
2- the index of producer prices of agricultural products (**output**)

Index of variation of prices reflecting **revenue** received by the producer for goods and services actually sold to customers over a period.

According to this comparison, Ireland shows more expensive, input and output, in comparison with the EU. Looking at the comparison of the two partners: Belgium and Netherlands shows evidence that is also more expensive because the indexes of prices are higher. However, no the higher between the members of the EU.

Index of variation of prices reflecting revenue received by the producer for goods and services actually sold to customers over a period.

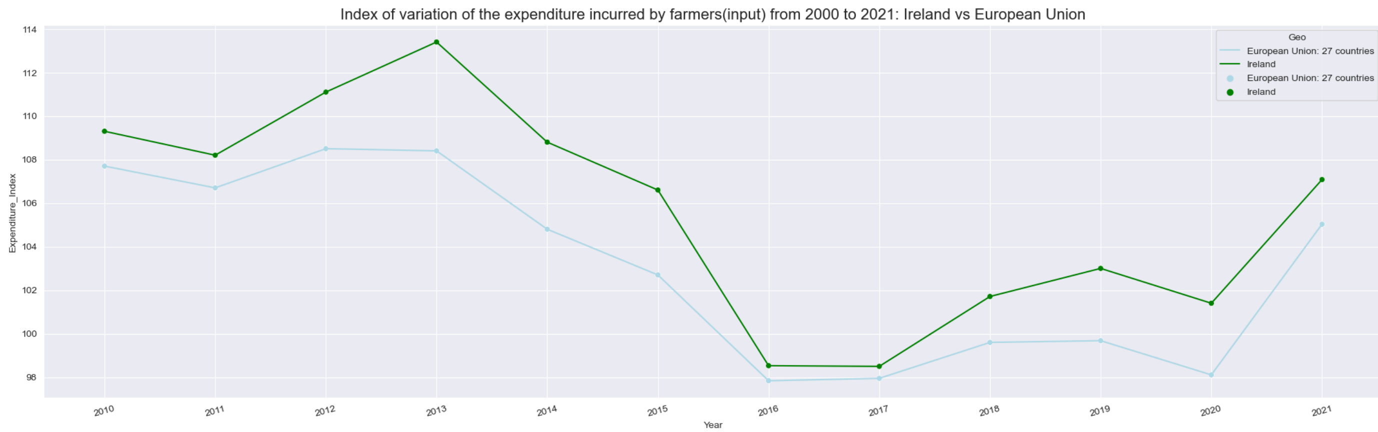
In [72]:
```
df_tmp = df_outa_t[(df_outa_t['Geo']=='Ireland') | (df_outa_t['Geo']=='European Union: 2

plt.figure(figsize=(25,7))
sns.lineplot(x=df_tmp["Year"],y=df_tmp['Price_Index'],hue= df_tmp['Geo'])
sns.scatterplot(x=df_tmp["Year"],y=df_tmp['Price_Index'],hue=df_tmp['Geo'])
plt.xticks(rotation=15);
plt.title("Index of producer prices of agricultural products (output) from 2000 to 2021:
plt.show()
```



Index of variation of the expenditure incurred by farmers in purchasing the means of production (goods and services as well as investment goods), including crop products from other agricultural units for intermediate consumption, over a given period.

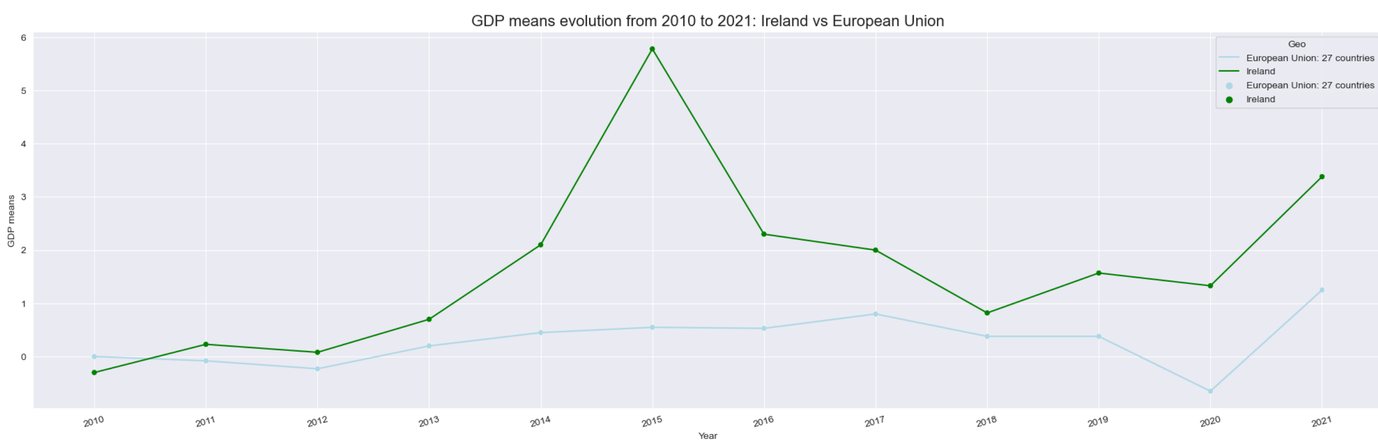
In [73]:
```
df_tmp = df_ina_t[(df_ina_t['Geo']=='Ireland') | (df_ina_t['Geo']=='European Union: 27 c

plt.figure(figsize=(25,7))
sns.lineplot(x=df_tmp["Year"],y=df_tmp['Expenditure_Index'],hue= df_tmp['Geo'])
```

```
sns.scatterplot(x=df_tmp["Year"],y=df_tmp['Expenditure_Index'],hue=df_tmp['Geo'])
plt.xticks(rotation=15);
plt.title("Index of variation of the expenditure incurred by farmers(input) from 2000 to
plt.show()
```



Comparison the index of prices: this is a index of GDP and main components (output, expenditure and income).

In [74]:
```
df_tmp = df_gdp_t[(df_gdp_t['Geo']=='Ireland') | (df_gdp_t['Geo']=='European Union: 27 c

plt.figure(figsize=(25,7))
sns.lineplot(x=df_tmp["Year"],y=df_tmp['GDP means'],hue= df_tmp['Geo'])
sns.scatterplot(x=df_tmp["Year"],y=df_tmp['GDP means'],hue=df_tmp['Geo'])
plt.xticks(rotation=15);
plt.title("GDP means evolution from 2010 to 2021: Ireland vs European Union",fontsize=16
plt.show()
```



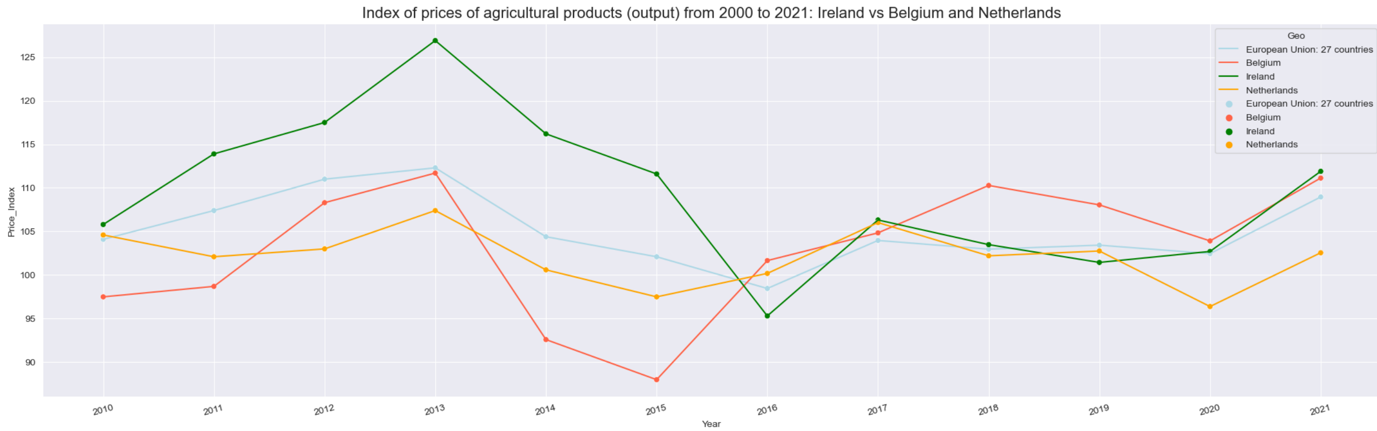
Index of variation of prices reflects revenue received by the producer for goods and services actually sold to customers over a period.

Comparison between Ireland and their neighbour Belgium and Netherlands, and also with EU (in general).

Note: Choose the **colour** associated with the country

In [75]:
```
df_tmp = df_outa_t[(df_outa_t['Geo']=='Ireland') | (df_outa_t['Geo']=='European Union: 2
                   (df_outa_t['Geo']=='Belgium') | (df_outa_t['Geo']=='Netherlands')]


plt.figure(figsize=(25,7))

PALETTE=["lightblue","tomato",'Green', "orange"] # Choose the colour associated with the
```

```
sns.lineplot(x=df_tmp["Year"],y=df_tmp['Price_Index'],hue= df_tmp['Geo'], palette=PALETT
sns.scatterplot(x=df_tmp["Year"],y=df_tmp['Price_Index'],hue=df_tmp['Geo'], palette=PALE
plt.xticks(rotation=15);
plt.title("Index of prices of agricultural products (output) from 2000 to 2021: Ireland
plt.show()
```

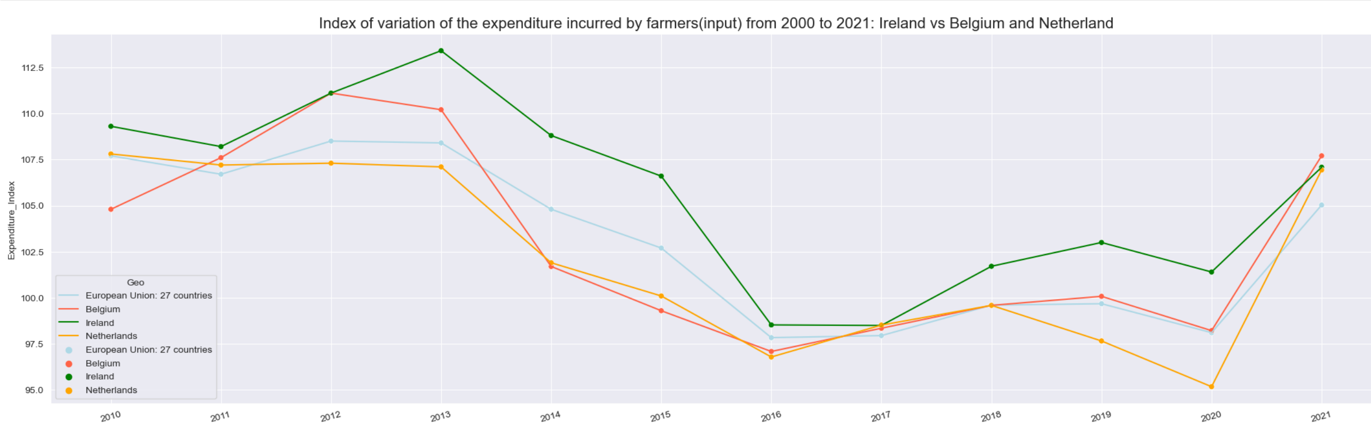


Index of variation of the expenditure incurred by farmers in purchasing the means of production (goods and services as well as investment goods), including crop products from other agricultural units for intermediate consumption, over a given period.

Comparison between Ireland and their neighbour Belgium and Netherlands, and also with EU (in general).

Note: Choose the **colour** associated with the country

In [76]:
```
df_tmp = df_ina_t[(df_ina_t['Geo']=='Ireland') | (df_ina_t['Geo']=='European Union: 27 c
                  (df_ina_t['Geo']=='Belgium') | (df_ina_t['Geo']=='Netherlands')]

PALETTE=["lightblue","tomato",'Green', "orange"] # Choose the colour associated with the

plt.figure(figsize=(25,7))
sns.lineplot(x=df_tmp["Year"],y=df_tmp['Expenditure_Index'],hue= df_tmp['Geo'], palette=
sns.scatterplot(x=df_tmp["Year"],y=df_tmp['Expenditure_Index'],hue=df_tmp['Geo'], palett
plt.xticks(rotation=15);
plt.title("Index of variation of the expenditure incurred by farmers(input) from 2000 to
plt.show()
```



In [77]:
```
## Analysis correlation
# The most common method for calculating correlation is Pearson's Correlation
# Coefficient.
# that assumes a normal distribution of the attributes involved.
# A correlation of -1 or 1 shows a full negative or positive correlation respectively.
```

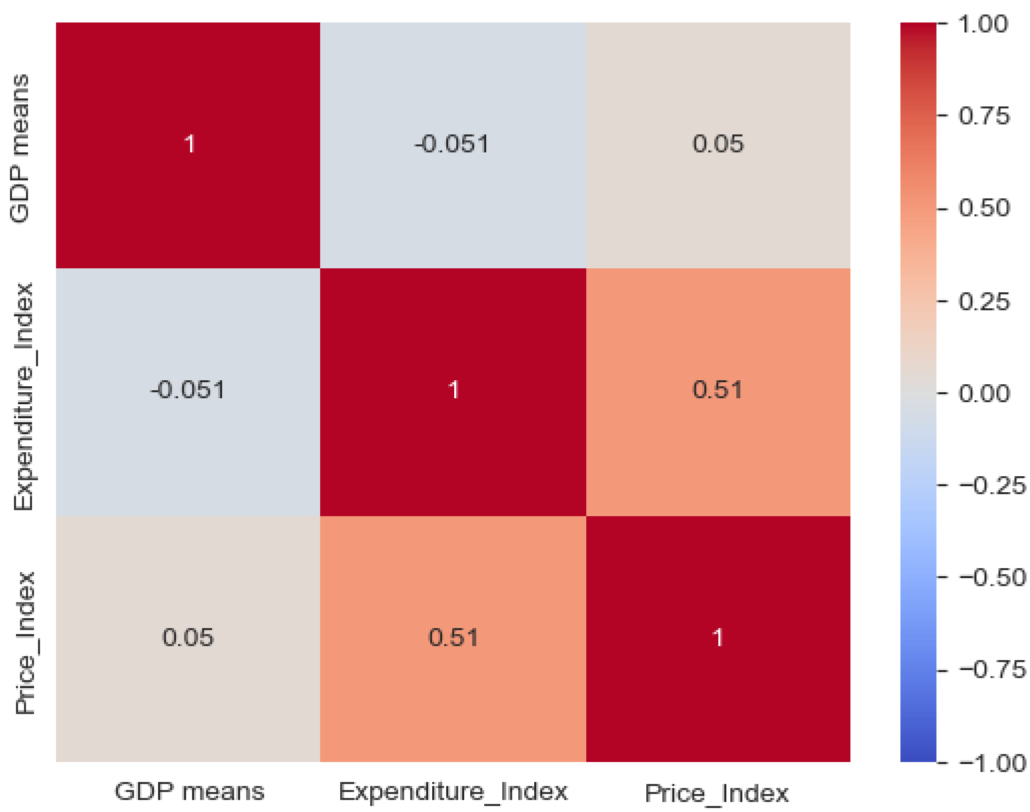
```
# Whereas a value of 0 shows no correlation at all.

df_final[['Geo', 'GDP means','Expenditure_Index','Price_Index']].corr(method='pearson')
```

Out[77]:

	GDP means	Expenditure_Index	Price_Index
GDP means	1.000000	-0.051372	0.050492
Expenditure_Index	-0.051372	1.000000	0.507172
Price_Index	0.050492	0.507172	1.000000

In [78]:

```
import seaborn as sns

sns.heatmap(df_final[['Geo', 'GDP means','Expenditure_Index','Price_Index']].corr(method
annot=True,cmap='coolwarm')

plt.show()
```



In [79]:

```
## Inference statistics Test Normality: Shapiro-Wilk test
import math
import numpy as np
from scipy.stats import shapiro


# Perform Shapiro-Wilk test for normality for data from Ireland

print('\n\n Test for normality for data from Ireland: \n',
     shapiro(df_final[['Price_Index']][df_final['Geo'].isin(['Ireland'])])
     )

# Perform Shapiro-Wilk test for normality for data from European Union: 27 countries
print('\n\n Test for normality for data from EU: \n',
      shapiro(df_final[['Price_Index']][df_final['Geo'].isin(['European Union: 27 countr
     )


# Our null hypothesis Ho is that the distribution is Normal.
```

```
# In both cases, since the p-value of the test is greater than α = .05, the test statist

# We fail to reject the null hypothesis of the Shapiro-Wilk test.

# Therefore, the data is assumed to be normally distributed.
```

```
Test for normality for data from Ireland:
ShapiroResult(statistic=0.975931942462921l, pvalue=0.9620502591133118)


Test for normality for data from EU:
ShapiroResult(statistic=0.9341965913772583, pvalue=0.4267212450504303)
```

Inferences statistics for two population means: t-student's test

All data coming from countries into the EU therefore, we assume that all those countries follow common rules about the production and sell agricultural products. Therefore, it is a reasonable belief that there are relations (dependencies) between the countries in the EU related to those indexes of the Agricultural Price Index (API) under the evidence that exist Agricultural common policies.

It is can be considered that all these features are correlated and have dependencies between the countries so it can be used **"paired dependence test"**.

The variances in the populations are **unknown**.

Ref:

N. Weiss. Introductory Statistics. Pearosn 2017. Inferences for two Populations Means. P. 460-520

In [80]:
```
import scipy.stats as stats

# Ho: Price index mean is equal in Ireland and EU
# H1: Price index mean is non equal in Ireland and EU

# Perform the paired samples t-test

ttest = stats.ttest_rel(df_final[['Price_Index']][df_final['Geo'].isin(['Ireland'])],  #
                 df_final[['Price_Index']][df_final['Geo'].isin(['European Union: 27 coun

# https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_rel.html

print('\n\n Score paired samples t-test Ireland vs EU: \n', ttest)

# Since the p-value (0.021) is less than 0.05, and the test statistic is 2.687, we rejec

# We have sufficient evidence to say that the price index are different between Ireland
```

```
Score paired samples t-test Ireland vs EU:
Ttest_relResult(statistic=array([2.68710252]), pvalue=array([0.02113828]))
```

Inferences statistics to compare the price index means between Ireland and their neighbours Belgium and Netherlands.

Analisis of variance (ANOVA)

According to mentioned before, all countries from the EU follow common rules about the production and sell agricultural products. Therefore, it is a reasonable belief that there are relations (dependencies) between the countries in the EU related to those indexes of the Agricultural Price Index (API) under the evidence that exist Agricultural common policies. means that the populations are not independent.

Hypothesis null, Ho: Price index means are equal for Ireland vs Belgium and Netherlands (significant difference between the means).

An alternative hypothesis, H1: Price index means are non-equal for Ireland vs Belgium and Netherlands.

ANOVA conditions:

- the distribution of the population are Normal
- the variances of the population are equal
- the populations independent

Analisis of variance: test assumption of Normality based on the Shapiro-Wilk test

In [81]:

```
## Analisis of variance: test assumption of Normality based on the Shapiro-Wilk test

## Inference statistics Test Normality: Shapiro-Wilk test
import math
import numpy as np
from scipy.stats import shapiro


# Perform Shapiro-Wilk test for normality for data from Ireland

print('\n\n Test for normality for data from Ireland: \n',
     shapiro(df_final[['Price_Index']][df_final['Geo'].isin(['Ireland'])])
     )

# Perform Shapiro-Wilk test for normality for data from Belgium
print('\n\n Test for normality for data from Belgium: \n',
      shapiro(df_final[['Price_Index']][df_final['Geo'].isin(['Belgium'])])
     )

# Perform Shapiro-Wilk test for normality for data from Netherlands
print('\n\n Test for normality for data from Netherlands: \n',
      shapiro(df_final[['Price_Index']][df_final['Geo'].isin(['Netherlands'])])
     )



# Our null hypothesis Ho is that the distribution is Normal.

# In all cases, since the p-value of the test is greater than α = .05, the test statitic

# Wwe fail to reject the null hypothesis of the Shapiro-Wilk test.

# Therefore, the data is assumed to be normally distributed.
```

```
Test for normality for data from Ireland:
ShapiroResult(statistic=0.9759319424629211, pvalue=0.9620502591133118)
```

```
Test for normality for data from Belgium:
ShapiroResult(statistic=0.9263584613800049, pvalue=0.3431062698364258)


Test for normality for data from Netherlands:
ShapiroResult(statistic=0.9661213159561157, pvalue=0.8662456274032593)
```

Analisis of variance: test assumption of the variances of the populations that the samples come from are equal Levene's Test.

In [82]:
```python
## Analisis of variance: test assumption of the variances of the populations that the sa

## Inference statistics Levene's Test for test the variances are equal.

## Using the 'mean' which is recommended for symmetric or moderate-tailed distributions.

# https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.levene.html


import scipy.stats as stats


# Perform Levene's test centered at the mean for test variance for data from Ireland, Be
print('\n\n Levene s test centered at the mean: \n',
      stats.levene(
                  df_final[['Price_Index']][df_final['Geo'].isin(['Ireland'])].values.re
                  df_final[['Price_Index']][df_final['Geo'].isin(['Belgium'])].values.re
                  df_final[['Price_Index']][df_final['Geo'].isin(['Netherlands'])].value
                  center='mean')
     )


# Note: use values.reshape(-1) to convert from 2D to 1D numpry array rrquerired from lev

# the hypothesis null, Ho: the variances are equal
# An alternative hypothesis, H1: the variances are different.


# In the test, the p-value (0.011) is less than .05. The test statistic is 5.122.

# This means that we can reject the null hypothesis.

# This means we have sufficient evidence to say that the variances in price index betwee
```

```
Levene s test centered at the mean:
LeveneResult(statistic=5.122188959499782, pvalue=0.011551562070365987)
```

Analisis of variance: assumption the populations are independent

Unfortunatly, There is no formal test to verify that the observations in each group are independent and than was mencionaed before, all countries into the EU follow similar rules. Therefore, we can considered that the samples ar not cimpletly indepedent.

According to the previous results of the tests applied, the best way to continue the analysis is using a Non-parametric test.

In this case, it has been chosen to use the **Kruskal-Wallis test**, which is the non-parametric version of the one-way ANOVA.

Analisis using a non-parametric test: Kruskal-Wallis

A Kruskal-Wallis test is used to determine whether there is a statistically significant difference between the medians of three or more independent groups.

Like the ANOVA test, a Kruskal-Wallis test has some assumptions:

- the variable understudied is ordinal or continuous
- the distributions are similar
- the observations in each need to be independent

In this case, the variable price index for each country is not completely independent for the reasons explained before. However, in order to continue the study, it could be right to **consider** that the samples from the three countries are almost independent.

The null hypothesis Ho: The median of the price index across the three countries are equal.

The alternative hypothesis H1: At least one of the median of the price index is different from the others countries.

In [83]:

```
from scipy import stats

# Perform the Kruskal-Wallis Test


print('\n\n Non-parametric Kruskal-Wallis Test to determine difference between the media
stats.kruskal(
            df_final[['Price_Index']][df_final['Geo'].isin(['Ireland'])].values.reshape(
            df_final[['Price_Index']][df_final['Geo'].isin(['Belgium'])].values.reshape(
            df_final[['Price_Index']][df_final['Geo'].isin(['Netherlands'])].values.resh
             )

# In this case, the test statistic is 6.14 and the corresponding p-value is 0.0461.

# Since this p-value is less than 0.05, we can reject the null hypothesis

# We have sufficient evidence to conclude that the median of the price index for the thr
# statistically significant differences.
```

```
 Non-parametric Kruskal-Wallis Test to determine difference between the medians :
```

Out[83]:

```
KruskalResult(statistic=6.149815506071129, pvalue=0.04619388943121816)
```

Statistical analysis: conclusion

According to the results, the values of the features from the countries in the EU has some dependency that was expected because of the common rules in the EU.

Ireland has a price index means and expenditure index mean higher than the EU in general and also higher than their neighbours Belgium and Netherlands. However, it is not higher, than Poland or Romania despite the fact that the Ireland expenditure index mean is higher.

The data did not show a strong influence that the GDP variation on the expenditure index or price index that was expected.

.

Implement interactive, dynamic and dashboard

(CRISP-DM Phase: Data Preparation Phase)

Organize the data by years-countries: analysis of geodata

In [84]:
```
link = 'https://github.com/lukes/ISO-3166-Countries-with-Regional-Codes/raw/master/all/a
print(link)
    # to read just one sheet to dataframe:
code = pd.read_csv(link)
code.rename(columns={'name':'Geo'}, inplace=True)
code = code[['Geo','alpha-3','region','iso_3166-2']]
code.head()
```

```
https://github.com/lukes/ISO-3166-Countries-with-Regional-Codes/raw/master/all/all.csv
```

Out[84]:

	Geo	alpha-3	region	iso_3166-2
0	Afghanistan	AFG	Asia	ISO 3166-2:AF
1	Åland Islands	ALA	Europe	ISO 3166-2:AX
2	Albania	ALB	Europe	ISO 3166-2:AL
3	Algeria	DZA	Africa	ISO 3166-2:DZ
4	American Samoa	ASM	Oceania	ISO 3166-2:AS

In [85]:
```
df_vs_back = df_vs.copy()
```

In [86]:
```
df_vs = pd.merge(df_vs,code,on='Geo',how='left')
```

In [87]:
```
df_vs = df_vs.fillna('EU')
df_vs.head()
```

Out[87]:

	Geo	Year	GDP means	emo	Expenditure_Index	Price_Index	alpha-3	region	iso_3166-2
0	European Union: 27 countries	2010	0.00	-1	107.7	104.1	EU	EU	EU
1	Belgium	2010	0.22	-1	104.8	97.5	BEL	Europe	ISO 3166-2:BE
2	Bulgaria	2010	0.40	-1	97.9	98.1	BGR	Europe	ISO 3166-2:BG
3	Czechia	2010	-0.12	1	93.8	88.7	CZE	Europe	ISO 3166-2:CZ
4	Denmark	2010	-0.05	1	108.7	100.5	DNK	Europe	ISO 3166-2:DK

In [88]:
```
# Prepare the data
```

```
df_vs['emo'] = df_vs['emo'].replace([-1, 1], ['neg', 'pos'])
df_vs.head()
```

Out[88]:

	Geo	Year	GDP means	emo	Expenditure_Index	Price_Index	alpha-3	region	iso_3166-2
0	European Union: 27 countries	2010	0.00	neg	107.7	104.1	EU	EU	EU
1	Belgium	2010	0.22	neg	104.8	97.5	BEL	Europe	ISO 3166-2:BE
2	Bulgaria	2010	0.40	neg	97.9	98.1	BGR	Europe	ISO 3166-2:BG
3	Czechia	2010	-0.12	pos	93.8	88.7	CZE	Europe	ISO 3166-2:CZ
4	Denmark	2010	-0.05	pos	108.7	100.5	DNK	Europe	ISO 3166-2:DK

Verify Tufts principles

Tufte's 6 principles:

(CRISP-DM Phase:Data Understanding Phase)

1. Comparisons: Show data by comparisons (bar charts and the like) to depict differences between an index of price in Ireland vs EU in general, and also EU Countries.
2. Causality: Show how the GDP and the index of expenditure impact the index of price.
3. Multivariate: simple graphics for easy interpretation from the general audience and the farmer.
4. Integration: Incorporate maps with numerical data to show the difference between Ireland and the EU countries.
5. Documentation: include attribution, detailed titles, and measurements.
6. Context: Show the trend by years from the period 2010 to 2021.

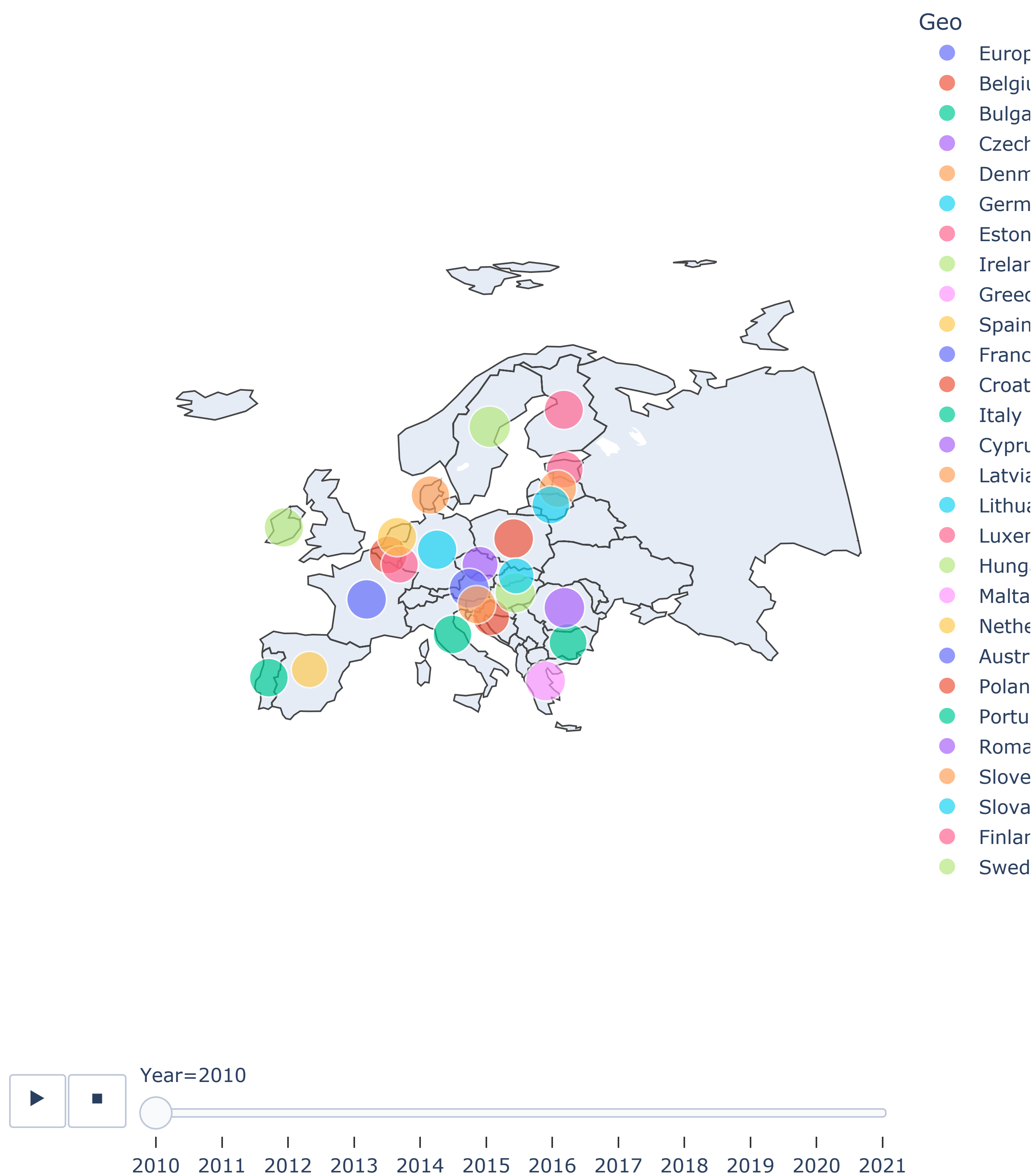
The colours chosen in the graphics follow the default values of the tools because in general, all countries in the analysis are under the same policy related to the agriculture production of the EU, the colour just represents that it is a different country (do not have another connection).

In [89]:

```
import plotly.express as px


fig = px.scatter_geo(df_vs,
                     locations="alpha-3", color="Geo", hover_name="Geo", size="Price_Ind
                     animation_frame="Year", projection="natural earth", scope='europe',
                     title='Comparison Index of price between EU countries respected Ire
                     width=860,
                     height=860)


fig.show()
```

Comparison Index of price between EU countries respected Ireland. Source:

Comparison Index of price between EU countries respected Ireland. Source:



In [90]:
```
import plotly.express as px

fig = px.scatter_3d(df_vs, x=df_vs['Price_Index'].values,
                    y=df_vs['Expenditure_Index'].values,
                    z=df_vs['GDP means'].values,
                    title='3D graph: Index of Price adn Expenditure vs variation of GDP
              color=df_vs['Geo'].values)
fig.show()
```

3D graph: Index of Price adn Expenditure vs variation of GDP between E

In [91]:

```
#Select Rows Based on List of Column Values
countries=["European Union: 27 countries","Ireland"]

df_vs[df_vs["Geo"].isin(countries)].head()
```

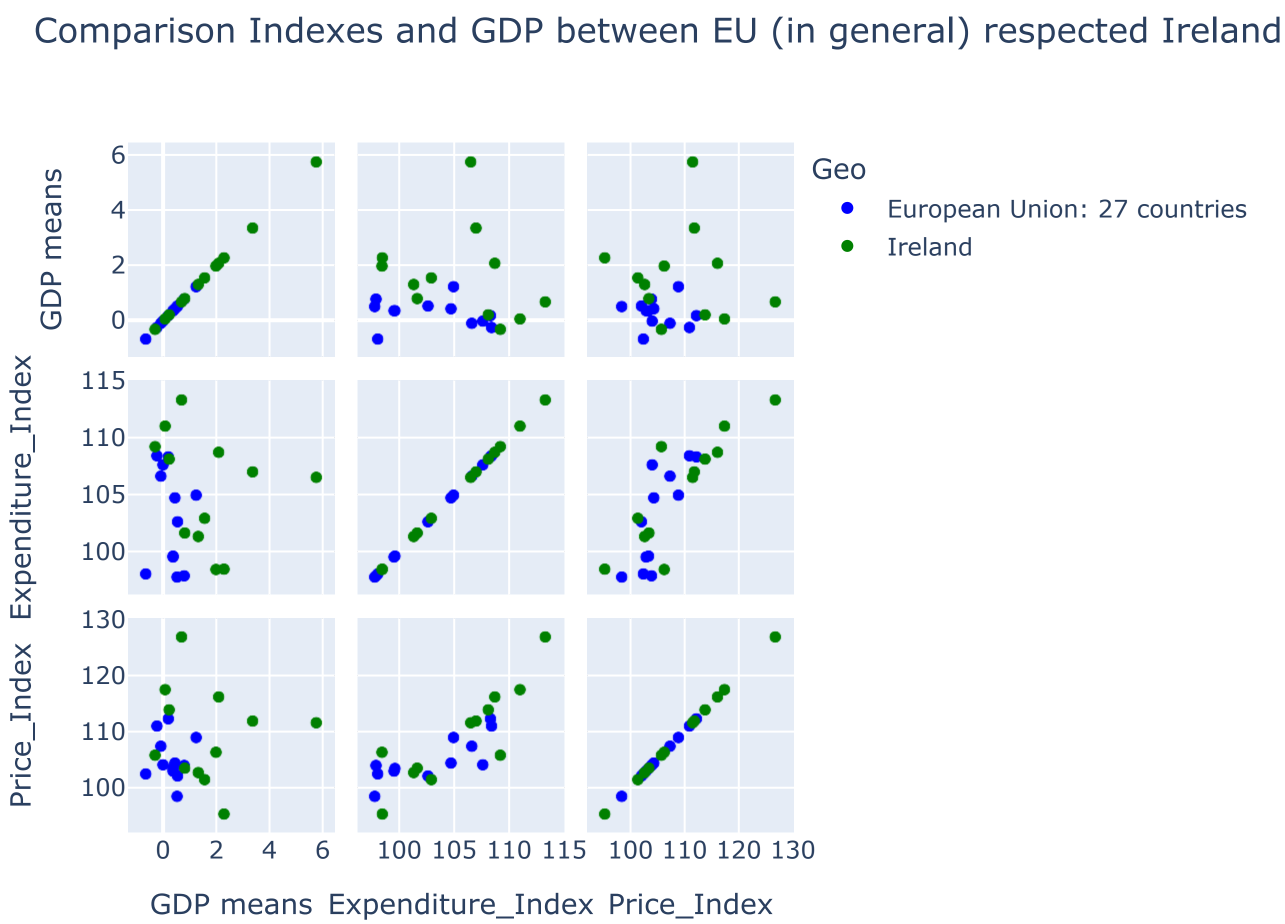
Out[91]:

	Geo	Year	GDP means	emo	Expenditure_Index	Price_Index	alpha-3	region	iso_3166-2
0	European Union: 27 countries	2010	0.00	neg	107.7	104.1	EU	EU	EU
7	Ireland	2010	-0.30	pos	109.3	105.8	IRL	Europe	ISO 3166-2:IE
28	European Union: 27 countries	2011	-0.08	pos	106.7	107.4	EU	EU	EU
35	Ireland	2011	0.23	neg	108.2	113.9	IRL	Europe	ISO 3166-2:IE
56	European Union: 27 countries	2012	-0.23	pos	108.5	111.0	EU	EU	EU

In [92]:

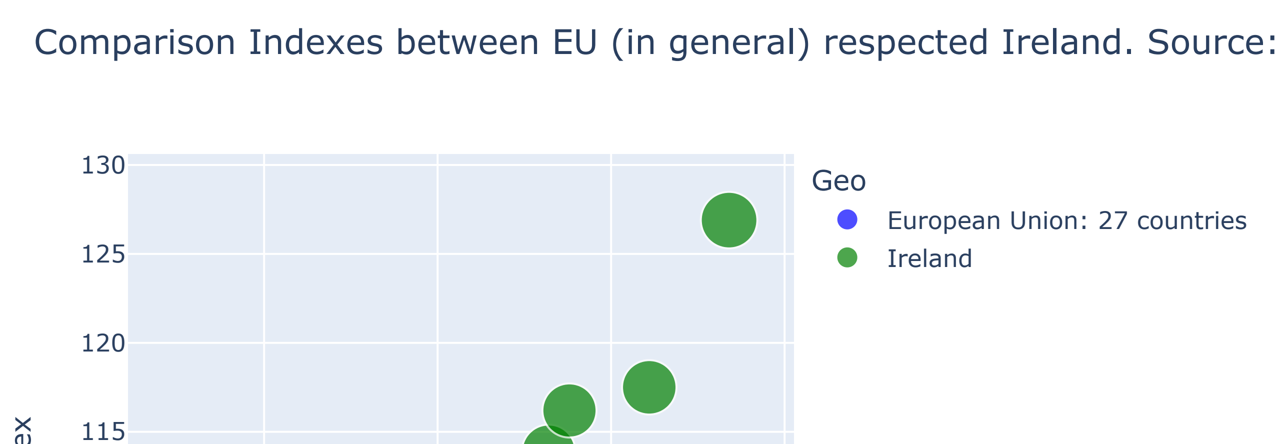
```
import plotly.express as px


from itertools import cycle
palette = cycle(px.colors.qualitative.Pastel1)
palette = cycle(px.colors.qualitative.Bold)
#palette = cycle(['black', 'grey', 'red', 'blue'])
palette = cycle(px.colors.sequential.PuBu)

# plotly setup

fig = px.scatter_matrix(df_vs[df_vs["Geo"].isin(countries)], dimensions=["GDP means", "E
                        color='Geo',
                        title='Comparison Indexes and GDP between EU (in general) respec
```

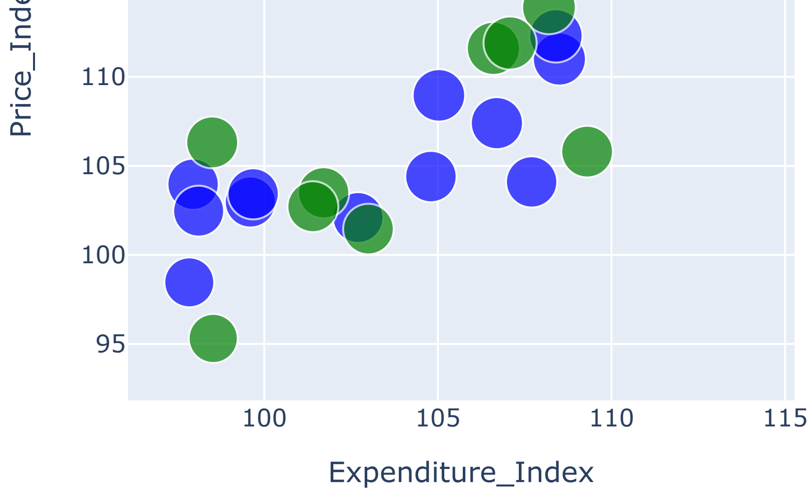
```
                         color_discrete_map={'Ireland': 'green','European Union: 27 count


fig.show()
```



In [93]:

```
# Matplotlib Scatter Plot

fig = px.scatter(df_vs[df_vs["Geo"].isin(countries)],
                 x="Expenditure_Index", y="Price_Index",
                 color="Geo",
                 size='Price_Index',
                  title='Comparison Indexes between EU (in general) respected Ireland. S
                 color_discrete_map={'Ireland': 'green','European Union: 27 countries':

fig.show()
```





Design and define daskboard using hvplot and Panel

hvPlot: A familiar and high-level API for data exploration and visualization

https://hvplot.holoviz.org/

In [94]:
```
df_vs   # data for visualization and daskboard
```

Out[94]:

	Geo	Year	GDP means	emo	Expenditure_Index	Price_Index	alpha-3	region	iso_3166-2
0	European Union: 27 countries	2010	0.00	neg	107.70	104.10	EU	EU	EU
1	Belgium	2010	0.22	neg	104.80	97.50	BEL	Europe	ISO 3166-2:BE
2	Bulgaria	2010	0.40	neg	97.90	98.10	BGR	Europe	ISO 3166-2:BG
3	Czechia	2010	-0.12	pos	93.80	88.70	CZE	Europe	ISO 3166-2:CZ
4	Denmark	2010	-0.05	pos	108.70	100.50	DNK	Europe	ISO 3166-2:DK
...	...	...	...	...	...	...	...	...	...
331	Romania	2021	1.12	neg	107.23	123.11	ROU	Europe	ISO 3166-2:RO
332	Slovenia	2021	2.60	neg	106.88	109.37	SVN	Europe	ISO 3166-2:SI
333	Slovakia	2021	0.28	neg	95.70	104.06	SVK	Europe	ISO 3166-2:SK
334	Finland	2021	0.75	neg	105.16	101.74	FIN	Europe	ISO 3166-2:FI
335	Sweden	2021	1.43	neg	107.61	117.58	SWE	Europe	ISO 3166-2:SE

336 rows × 9 columns

In [95]:
```
import panel as pn
```

```python
pn.extension('tabulator', sizing_mode="stretch_width")
```

In [96]:
```python
import hvplot.pandas
import holoviews as hv
import numpy as np
import hvplot.pandas
#import hvplot.dask

hvplot.extension('plotly')
#hv.extension('bokeh')
```

In [97]:
```python
# define color palette

PALETTE = ["#fd7f6f", "#7eb0d5", "#b2e061", "#bd7ebe", "#ffb55a", "#ffee65", "#beb9db",
#PALETTE = ["#ff6f69", "#ffcc5c", "#88d8b0", ]
pn.Row(
    pn.layout.HSpacer(height=50, background=PALETTE[2]),
    pn.layout.HSpacer(height=50, background=PALETTE[1]),
    pn.layout.HSpacer(height=50, background=PALETTE[4]),
)
```

Out[97]:

In [98]:
```python
df_daskboard = df_vs[['Geo','Year','GDP means','emo','Expenditure_Index','Price_Index']]
```

In [99]:
```python
 #define dataframe pipeline
(
    df_daskboard
    .reset_index()
    .sort_values(by='Year')
)
```

Out[99]:

	index	Geo	Year	GDP means	emo	Expenditure_Index	Price_Index
0	0	European Union: 27 countries	2010	0.00	neg	107.70	104.10
27	27	Sweden	2010	0.42	neg	108.60	116.30
26	26	Finland	2010	-0.23	pos	106.90	104.30
25	25	Slovakia	2010	0.38	neg	93.30	88.00
24	24	Slovenia	2010	-0.27	pos	110.00	99.00
...	...	...	...	...	...	...	...
309	309	Belgium	2021	1.50	neg	107.70	111.13
308	308	European Union: 27 countries	2021	1.25	neg	105.03	108.96
334	334	Finland	2021	0.75	neg	105.16	101.74
320	320	Italy	2021	1.60	neg	108.57	111.81
335	335	Sweden	2021	1.43	neg	107.61	117.58

336 rows × 7 columns

In [100…

```
# make dataframe pipeline interactive
idf = df_daskboard.interactive()
```

In [101…

```
# make dataframe pipeline interactive
idf_irl = df_daskboard.loc[df_daskboard['Geo']=='Ireland'].interactive()
```

In [102…

```
# define widgets
# https://panel.holoviz.org/user_guide/Widgets.html


year_slider = pn.widgets.IntSlider(name='Year', start=2015, end=2021, step=1, value=2015


year_slider
```

Out[102]:

Year: **2015**

In [103…

```
# define widgets
# https://panel.holoviz.org/user_guide/Widgets.html

Geo = pn.widgets.MultiSelect(
    name='Geo',
    options=df_vs['Geo'].unique().tolist(),
    value=df_vs['Geo'].unique().tolist())
    #button_type='success')
Geo
```

Out[103]:

Geo

European Union: 27 countries
Belgium
Bulgaria
Czechia

In [104…

```
#xaxis = pn.widgets.RadioButtonGroup(
#    name='x axis',
#    options=['Year'],
#    button_type='success')
#xaxis
```

In [105…

```
yaxis = pn.widgets.RadioButtonGroup(
    name='Choose the values on Y axis',
    options=['Price_Index', 'Expenditure_Index','GDP means'],
    button_type='success')
yaxis
```

Out[105]:

Price_Index	Expenditure_Index	GDP means

In [106…

```
# combine pipeline and widgets for all countires
ipipeline = (
    idf[
        (idf.Year <= year_slider) &
        (idf.Geo.isin(Geo))
        ]
    .reset_index()
    .sort_values(by='Year')
    .reset_index(drop=True)
```

```
    )
ipipeline.head()
```

Out[106]:

Year: **2015**

Geo

European Union: 27 countries
Belgium
Bulgaria
Czechia

	index	Geo	Year	GDP means	emo	Expenditure_Index	Price_Index
0	0	European Union: 27 countries	2010	0.00	neg	107.7	104.1
1	27	Sweden	2010	0.42	neg	108.6	116.3
2	26	Finland	2010	-0.23	pos	106.9	104.3
3	25	Slovakia	2010	0.38	neg	93.3	88.0
4	24	Slovenia	2010	-0.27	pos	110.0	99.0

In [107…

```
# combine pipeline and widgets only Irland
ipipeline2 = (
    idf_irl[
            (idf.Year <= year_slider) &
            (idf.Geo == 'Ireland')
            ]
    .reset_index()
    .sort_values(by='Year')
    .reset_index(drop=True)
    )
ipipeline2.head()
```

Out[107]:

Year: **2015**

	index	Geo	Year	GDP means	emo	Expenditure_Index	Price_Index
0	7	Ireland	2010	-0.30	pos	109.3	105.8
1	35	Ireland	2011	0.23	neg	108.2	113.9
2	63	Ireland	2012	0.08	neg	111.1	117.5
3	91	Ireland	2013	0.70	neg	113.4	126.9
4	119	Ireland	2014	2.10	neg	108.8	116.2

In [108…

```
# create interactive table for all countries
# pipe to table
# https://panel.holoviz.org/reference/widgets/Tabulator.html

itable = ipipeline.pipe(pn.widgets.Tabulator, pagination='remote',
                        page_size=10, sizing_mode='stretch_width')
itable
```

Out[108]: Year: **2015**

Geo

European Union: 27 countries
Belgium
Bulgaria
Czechia

	index	Geo	Year	GDP means	emo	Expenditure_I
0	0	European Union: 27 countries	2,010	0.0	neg	
1	27	Sweden	2,010	0.42	neg	
2	26	Finland	2,010	-0.23	pos	
3	25	Slovakia	2,010	0.38	neg	
4	24	Slovenia	2,010	-0.27	pos	
5	23	Romania	2,010	0.43	neg	
6	22	Portugal	2,010	-0.7	pos	
7	21	Poland	2,010	0.65	neg	
8	19	Netherlands	2,010	-0.17	pos	
9	18	Malta	2,010	0.7	neg	

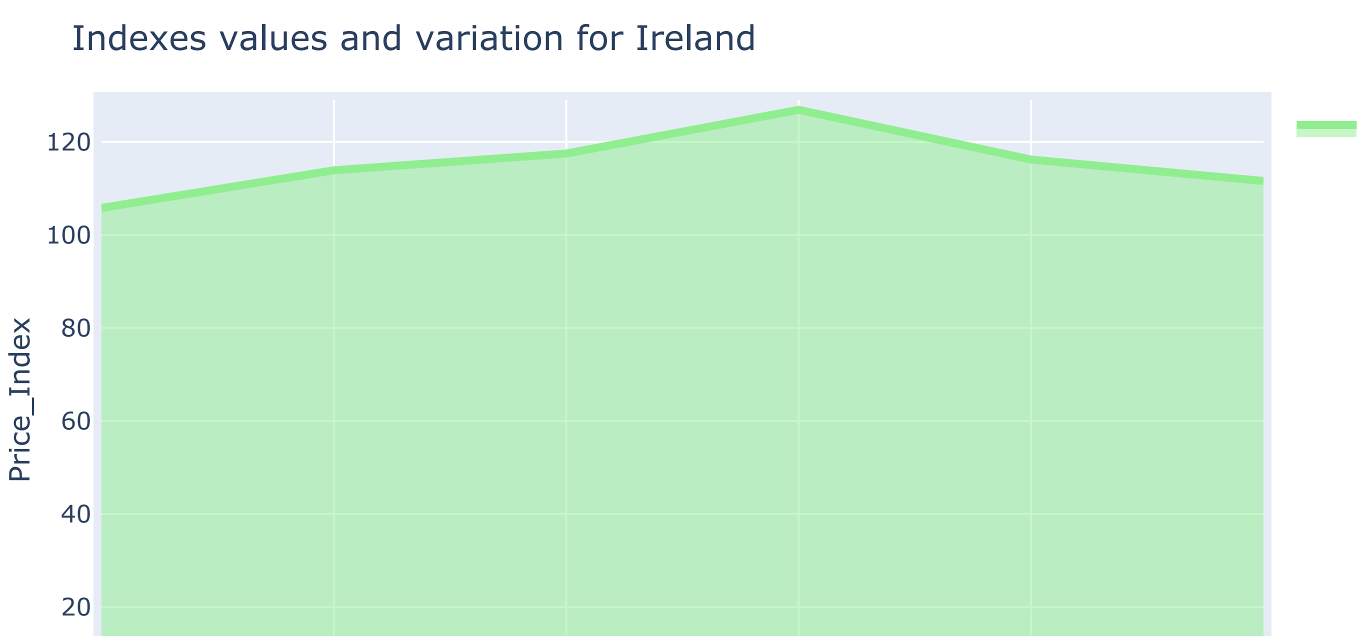
First Prev 1 2 3 4 5 Next Last

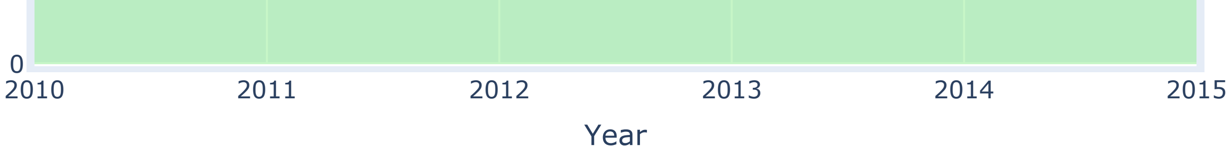
In [109…

```
# IRELAND
# create interactive plot
# pipe to hvplot
import hvplot.pandas

ihvplot_irl = ipipeline2.hvplot(x='Year', y=yaxis, kind='area', color='lightgreen', line
                                height=400, title='Indexes values and variation for Ireland
ihvplot_irl
```

Out[109]: Year: **2015**

Price_Index Expenditure_Index GDP means





In [110…

```
# All Countries
# create interactive plot
# pipe to hvplot

ihvplot = ipipeline.hvplot(x='Year', y=yaxis, kind='area',
                           color='lightblue', line_width=4,
                           title='Indexes values and variation for each Country in the E
                           height=400)
ihvplot
```

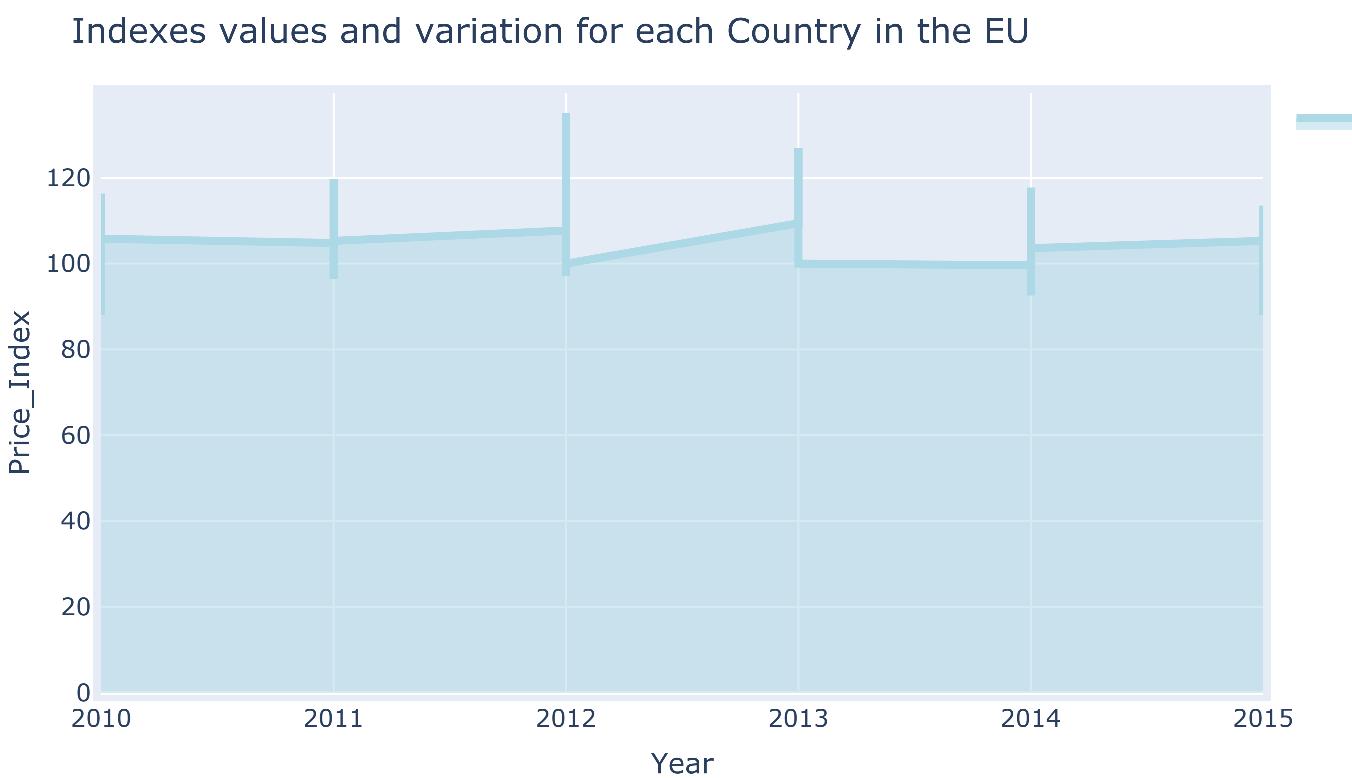
Out[110]:

Year: **2015**

Geo

European Union: 27 countries
Belgium
Bulgaria
Czechia

Price_Index | Expenditure_Index | GDP means



In [111…

```
# create interactive plot
# pipe to hvplot
import hvplot.pandas


ihvplot_index = ipipeline.hvplot(x='Year', y=['Price_Index', 'Expenditure_Index'],
                                 kind='area',
                                 title='Comparative Price Index vs Expenditure Index',
                                 color=PALETTE)
```

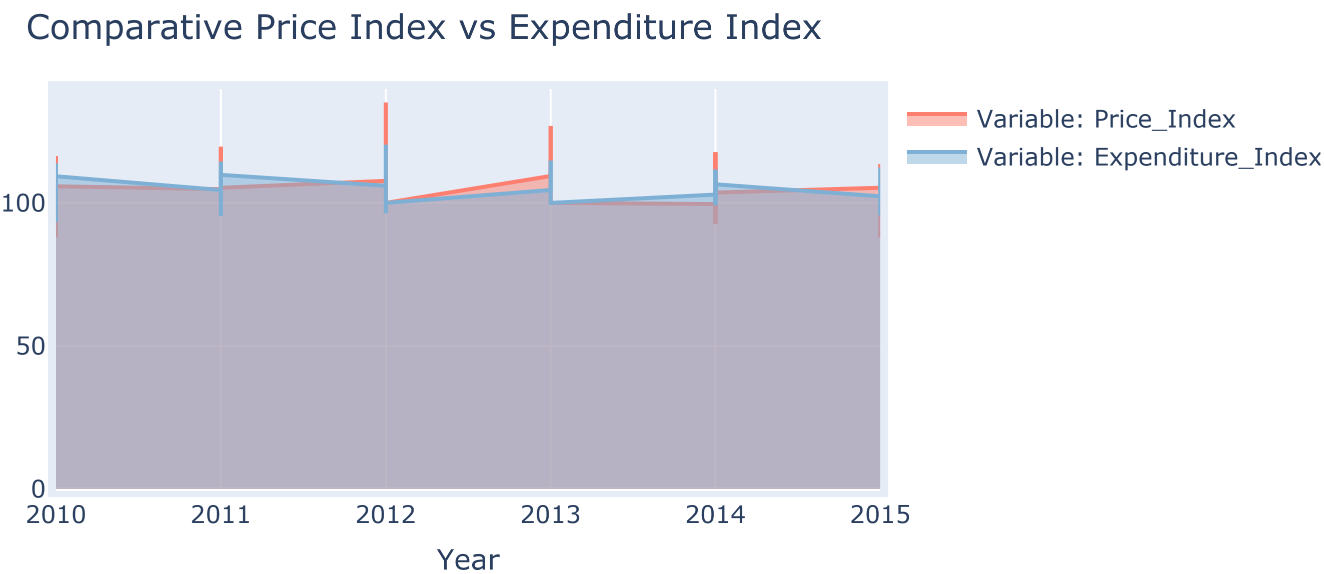
```python
ihvplot_index

#ihvplot = ipipeline.hvplot(x='GDP means', y=yaxis, by='Geo', color=PALETTE, line_width=
#ihvplot
```

Out[111]:

Year: **2015**

Geo

European Union: 27 countries
Belgium
Bulgaria
Czechia



In [112…

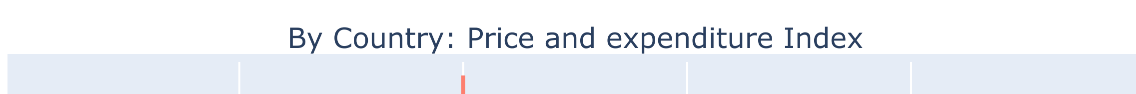
```python
# create interactive plot
# pipe to hvplot


plot4 = ipipeline.hvplot(x='Year', y=['Price_Index', 'Expenditure_Index'], kind='area',
                          title='By Country: Price and expenditure Index',
                          color=PALETTE)
plot5 = ipipeline2.hvplot(x='Year', y=['Price_Index', 'Expenditure_Index'], kind='line',
                           color=PALETTE,
                           title='Ireland: Price and expenditure Index')
ihvplot_irl_vs = plot4 + plot5

ihvplot_irl_vs
#ihvplot = ipipeline.hvplot(x='GDP means', y=yaxis, by='Geo', color=PALETTE, line_width=
#ihvplot
```

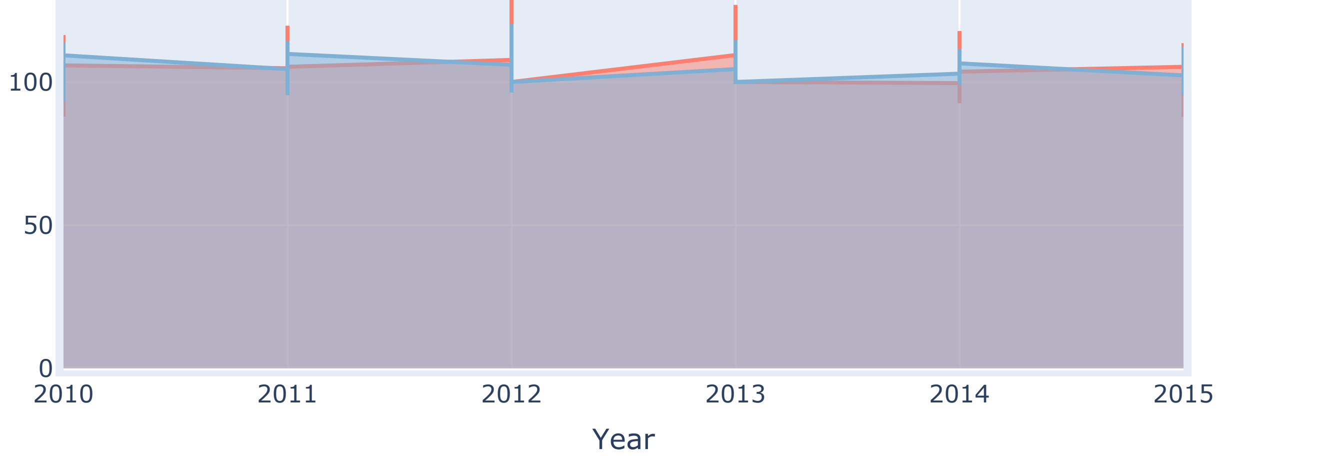
Out[112]:

Year: **2015**

Geo

European Union: 27 countries
Belgium
Bulgaria
Czechia





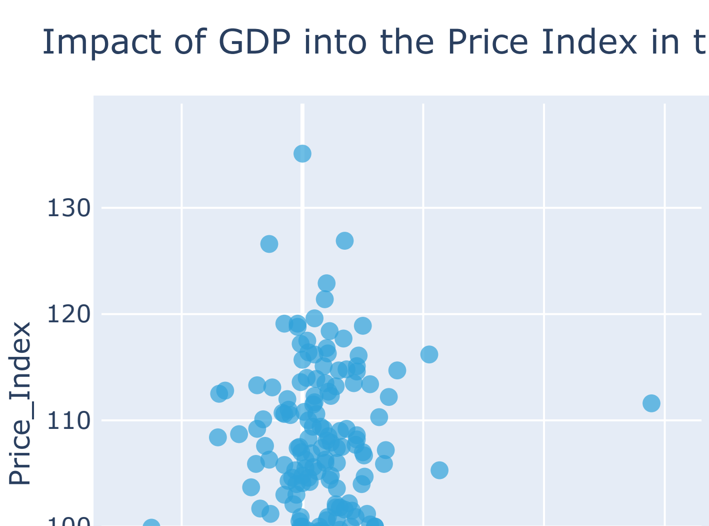
In [113…
```
# Scatterplot Price Index vs Variation on the GDP


ipipeline3 = (
    idf[
        (idf.Year <= year_slider) &
        (idf.Geo.isin(Geo))
        ]
    .reset_index()
    .sort_values(by='Year')
    .reset_index(drop=True)
    )
```

In [114…
```
index_vs_gdp_scatterplot = ipipeline3.hvplot(x='GDP means',
                                            y='Price_Index',
                                            size=80, kind="scatter",
                                            alpha=0.7,
                                            legend=False,
                                            height=400,
                                            width=400,
                                            title='Impact of GDP into the Price Index in
index_vs_gdp_scatterplot
```

Out[114]: Year: **2015**

Geo

European Union: 27 countries
Belgium
Bulgaria
Czechia



Using layout from Panel to create a daskboard

In [115…

```
# Layout using template
# https://panel.holoviz.org/reference/templates/FastListTemplate.html#templates-gallery-

template = pn.template.FastListTemplate(
    title='Interactive Dashboards for the Index of the agriculture products in Ireland a
    sidebar=[year_slider,
             'Country', Geo,
             'Choose the values on Y axis' , yaxis,
              pn.pane.PNG('Ireland.png', sizing_mode='scale_both'),
              pn.pane.Markdown("#### The Price Index and Expenditure Index on Agricultur
             ],

    main=[
          pn.Row(pn.Column(ihvplot_irl.panel(width=700), margin=(0,25)),
                 pn.Column(ihvplot.panel(width=700), margin=(0,25))
                 ),
          pn.Row(
                 pn.Column(ihvplot_index.panel(width=700), margin=(0,50)),
                 pn.Column(index_vs_gdp_scatterplot.panel(width=300), margin=(0,50)),
                 ),
          pn.Row(ihvplot_irl_vs.panel(width=600), margin=(0,25)
                ),
          pn.Row(itable.panel())
          ],



 #main=[pn.Row(pn.Column(yaxis_co2,
 #                        co2_plot.panel(width=700), margin=(0,25)),
 #               co2_table.panel(width=500)
 #             ),
 #      pn.Row(pn.Column(co2_vs_gdp_scatterplot.panel(width=600), margin=(0,25)),
 #              pn.Column(yaxis_co2_source, co2_source_bar_plot.panel(width=600))
 #             )],

    accent_base_color="#88d8b0",
    header_background="#88d8b0",
    )

template.show()
```

```
Launching server at http://localhost:50469
```

Out[115]:

```
<panel.io.server.Server at 0x7f8aac6c4b20>
```

Strategic to approach the problem and modelling the data

(CRISP-DM Phase: Modelling Phase)

In this project, the approach to tackle the problem to estimate the Index of price for agriculture products in Ireland, and consequently all EU countries based on the data by Eurostat, would be

1- Applying a Neural Network model for regression over data in 3D included a **categorical sentimental feature** as input.

2- Applying regression models based on the Random Forest model and two types of techniques of gradient boosting framework: XGBoost and Light GBM (light gradient-boosting machine). Also included a **categorical sentimental feature** as input.

Artificial Neural Network

(CRISP-DM Phase: Modelling Phase)

ANN Neural Networks

In this part, it implemented a simple ANN model for regression applying **3 Dimensional data** for:

- Country
- Year
- Features available:
    - GDP means (means of GDP)
    - feature "emo" (including emotional features from the opinion of experts)
    - Expenditure_Index (Index of variation of the expenditure incurred by farmers(input))
    - Price_Index (feature target: index of producer prices of agricultural products (output))

Specifically, this is a problem for **multivariable (features)** time series forecasting that can be approached using ANN models for

- input shape is 4 features, 28 countries (27 + EU global)
- activation function: rectified linear unit ReLU
- fully connected by using 32 nodes hidden layers

For regression: One unit with no activation function.

Loss function for regression: Mean square error.

Ref:

A. Géron, “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow” (p. 289-311) 2019. ,O'Reilly

J. Brownlee, 2020, Deep Learning for Time Series Forecasting, Edition: v1.7, Pag.123-149.

https://books.google.ie/books?hl=en&lr=&id=o5qnDwAAQBAJ&oi=fnd&pg=PP1&dq=J.+Brownlee,+2020,+Deep+Learning+for+Time+Serie

https://en.wikipedia.org/wiki/Long_short-term_memory

Artificial Neural Network

Neural network

Data preparation 3D

(CRISP-DM Phase: Data Preparation Phase)

Prepare the data array 3D for Neural Network algorithm

In [116…
```
df_final
```

Out[116]:

	Geo	Year	GDP means	emo	Expenditure_Index	Price_Index
0	European Union: 27 countries	2010	0.00	-1	107.70	104.10
1	Belgium	2010	0.22	-1	104.80	97.50
2	Bulgaria	2010	0.40	-1	97.90	98.10
3	Czechia	2010	-0.12	1	93.80	88.70
4	Denmark	2010	-0.05	1	108.70	100.50
...	...	...	...	...	...	...
331	Romania	2021	1.12	-1	107.23	123.11
332	Slovenia	2021	2.60	-1	106.88	109.37
333	Slovakia	2021	0.28	-1	95.70	104.06
334	Finland	2021	0.75	-1	105.16	101.74
335	Sweden	2021	1.43	-1	107.61	117.58

336 rows × 6 columns

Transform categorical data into integer

In [117…
```
# LabelEncoder can be used to transform categorical data into integers:

from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
df_final['Geo'] = label_encoder.fit_transform(df_final['Geo'])
```

Select features and target for all models below

In [118…
```
features = df_final[['Geo','Year','GDP means','emo','Expenditure_Index']]
target = df_final['Price_Index']
```

Standardizer the data of features and reshape df in format array 3D for ANN model

In [119…
```
#standardizing the training dataset before training.
from sklearn.preprocessing import StandardScaler
```

```
# define standard scaler
scaler = StandardScaler()

# transform data
features_scaled = scaler.fit_transform(features)

# creating back a Dataframe object
features_scaled = pd.DataFrame(features_scaled)
```

reshape DF in format array 3 D

In [120…
```
features_scaled3D = features_scaled.to_numpy().reshape(12, 28, 5)
```

In [121…
```
target3D = target.to_numpy().reshape(12, 28, 1)
```

In [122…
```
print(features_scaled3D.shape)
print(target3D.shape)
```

```
(12, 28, 5)
(12, 28, 1)
```

In [123…
```
# Divide the data into training and test sets
x_train, x_test, y_train, y_test = train_test_split(
features_scaled3D, target3D, test_size=0.20, random_state=61)

print('features and target: ', features_scaled3D.shape, target3D.shape)
print('data for train: ', x_train.shape, y_train.shape)
print('data for test: ', x_test.shape, y_test.shape)
```

```
features and target:  (12, 28, 5) (12, 28, 1)
data for train:  (9, 28, 5) (9, 28, 1)
data for test:  (3, 28, 5) (3, 28, 1)
```

In [124…
```
#input_shape=train_dataset.shape[1:])
x_train.shape[1:]
```

Out[124]:
```
(28, 5)
```

In [125…
```
# Fix the randomness of an ANN using seed
from numpy.random import seed
from tensorflow.keras.utils import set_random_seed
# setting seed to fix randomness
seed(0)
set_random_seed(0)
```

In [126…
```
# Define neural network model
network = models.Sequential()

#  https://keras.io/api/layers/core_layers/
# -------------- ReLU ------
# Regression:
# One unit with no activation function.
# --------------------------

# ------- loss function -------
# Regression: Mean square error.
# -----------------------------


# Add fully connected layer with a ReLU activation function
network.add(layers.Dense(units=32,  # 32
                         activation="relu",
                         input_shape=(28,5)))  # flatten features
```

```
# Add fully connected layer with a ReLU activation function
network.add(layers.Dense(units=32, activation="relu"))

# Add  leyer of dropout rate of 50% each, to reduce overfitting.
#network.add(layers.Dropout(0.5))

# Add fully connected layer with no activation function
network.add(layers.Dense(units=1))  # Regression: One unit with no activation function.


# Compile neural network
network.compile(loss="mse", # Mean squared error for regression
                optimizer="Adam", # Adam is an update to the RMSProp optimizer
                #optimizer="RMSprop", # Optimization algorithm
                metrics=["mse"]) # Mean squared error

print(network.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 28, 32)            192

 dense_1 (Dense)             (None, 28, 32)            1056

 dense_2 (Dense)             (None, 28, 1)             33

=================================================================
Total params: 1,281
Trainable params: 1,281
Non-trainable params: 0

_________________________________________________________________
None
```

```
2023-01-03 08:47:54.388714: I tensorflow/core/platform/cpu_feature_guard.cc:193] This Te
nsorFlow binary is optimized with oneAPI Deep Neural Network Library (oneDNN) to use the
following CPU instructions in performance-critical operations:  SSE4.1 SSE4.2
To enable them in other operations, rebuild TensorFlow with the appropriate compiler fla
gs.
```

In [127…

```
# Training neural network
history = network.fit(x_train, # Features
                      y_train, # Target vector
                      epochs=10, # Number of epochs 10
                      verbose=0, # No output
                      batch_size=10, # Number of observations per batch (100)
                      validation_data=(x_test, y_test)) # Test data

# Usually Batch Size <= Size of Training Set
```

In [128…

```
print(y_train.shape)
print(y_test.shape)
```

```
(9, 28, 1)
(3, 28, 1)
```

Making prediction

In [129…

```
# Predict classes of test set
predicted_target = network.predict(x_test)

#print('\n Data for target test:', y_test)
```

```
#print('\n Data predicted for the feature test:', predicted_target)

print('\n Number of values for target test: ', y_test.shape[1])
print('\n Dimension for target test: ',  y_test.shape)
print('\n Dimension of values for predicted values: ',  predicted_target.shape)

pd.DataFrame(tf.keras.metrics.mean_squared_error(y_test, predicted_target)).describe()
```

```
1/1 [==============================] - 0s 91ms/step

 Number of values for target test:  28

 Dimension for target test:  (3, 28, 1)

 Dimension of values for predicted values:  (3, 28, 1)
```

Out[129]:

	0	1	2	3	4	5	
count	3.000000	3.000000	3.000000	3.000000	3.000000	3.000000	3.000
mean	10953.424805	10106.360352	11514.340820	12344.872070	11039.469727	11245.825195	10277.349
std	506.302521	1781.150513	2812.611816	2769.123535	1632.112305	813.842346	564.427
min	10494.124023	8552.689453	8650.048828	9150.628906	9227.520508	10697.707031	9938.546
25%	10681.975586	9134.402344	10135.383789	11483.760254	10362.083008	10778.260742	9951.563
50%	10869.827148	9716.115234	11620.718750	13816.891602	11496.645508	10858.814453	9964.581
75%	11183.075195	10883.195801	12946.486328	13941.994629	11945.444336	11519.884766	10446.750
max	11496.323242	12050.276367	14272.253906	14067.097656	12394.243164	12180.955078	10928.920

8 rows × 28 columns

Evaluate model using MSE

Accuracy using mean squared error (MSE)

MSE, measures the average of the squares of the errors—

that is, the average squared difference between the estimated values and the actual value.

In [130…

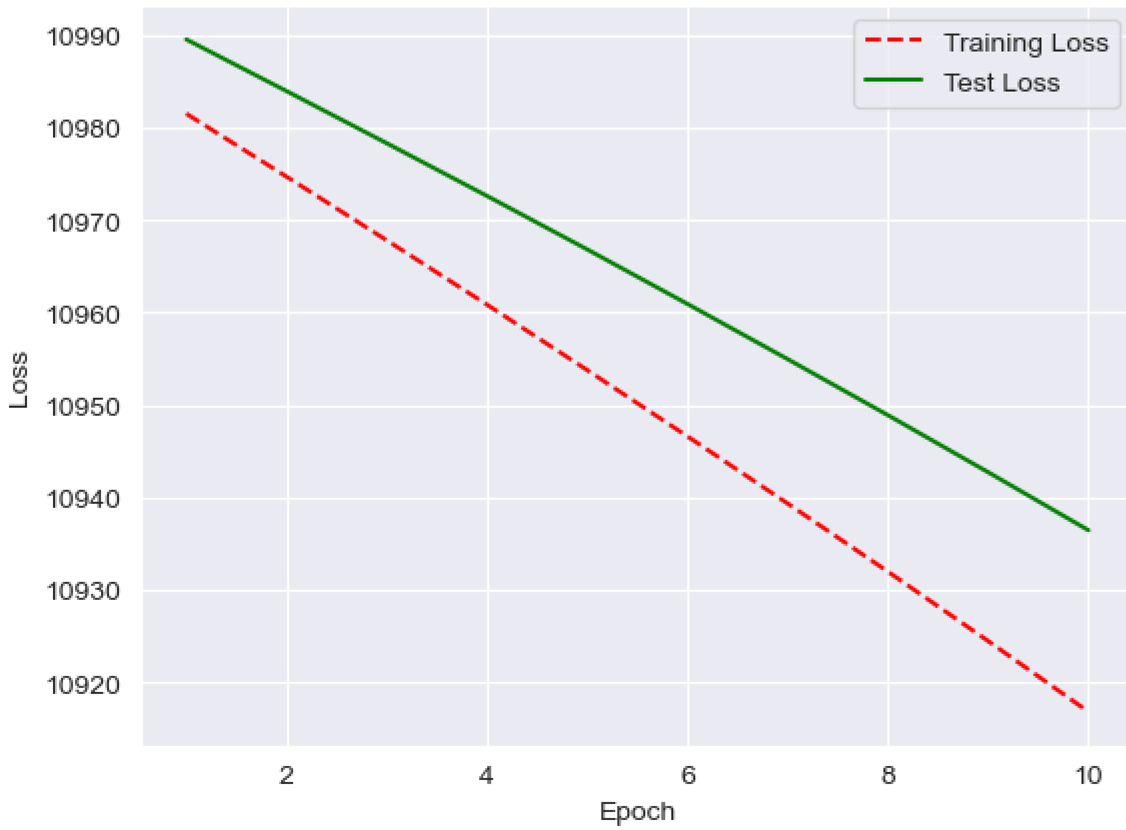
```
scores = network.evaluate(x_train, y_train)
print("Score MSE for data training: %.2f%%\n" % (scores[1]))

scores = network.evaluate(x_test, y_test)
print("Score MSE for data test: %.2f%%\n" % (scores[1]))
```

```
1/1 [==============================] - 0s 23ms/step - loss: 10909.0596 - mse: 10909.0596
Score MSE for data training: 10909.06%

1/1 [==============================] - 0s 18ms/step - loss: 10936.4561 - mse: 10936.4551
Score MSE for data test: 10936.46%
```

In [131…

```
# Get training and test loss histories
training_loss = history.history["loss"]
test_loss = history.history["val_loss"]

# Visualize loss history
# Create count of the number of epochs
```

```python
epoch_count = range(1, len(training_loss) + 1)
plt.plot(epoch_count, training_loss, "r--")
plt.plot(epoch_count, test_loss, "g-")
plt.legend(["Training Loss", "Test Loss"])
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.show();
```



k-Fold Cross-Validating Neural Networks

(CRISP-DM Phase: Modelling Phase)

k-Fold Cross-Validating Neural Networks

Ref:

A. Géron, “Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow” (p.322) 2019. ,O'Reilly

In [132…
```python
# k-Fold Cross-Validating Neural Networks
# Load libraries

from keras.wrappers.scikit_learn import KerasRegressor  # using for k-folk CV
from sklearn.model_selection import KFold               # using for k-folk CV

# Create function for Neural Network Model
def network_kfold():

    # Define neural network
    network = models.Sequential()

    # Add fully connected layer with a ReLU activation function
```

```
    network.add(layers.Dense(units=64, activation="relu",
                             input_shape=(28,5)))  # 5 features, 28 countries

    # Add fully connected layer with a ReLU activation function
    network.add(layers.Dense(units=32, activation="relu"))

    # Add fully connected layer with a sigmoid activation function
    network.add(layers.Dense(units=1, activation="sigmoid"))

    # Compile neural network
    #rmsprop = optimizers.RMSprop(lr=0.001)
    network.compile(loss="binary_crossentropy", # Cross-entropy
                    #optimizer="rmsprop", # Root Mean Square Propagation
                    optimizer="Adam", # Adam is an update to the RMSProp optimizer
                    metrics=["accuracy"]) # Accuracy performance metric

    # Return compiled network
    return network
```

Define a dictionary for accumulating the results from k-fold over all models in order to compare them

scores_models

In [133…

```
scores_models = {}
```

In [134…

```
# Wrap Keras model so it can be used by scikit-learn
kfold = KFold(n_splits = 10)

neural_network = KerasRegressor(build_fn=network_kfold,
                                 epochs=10,
                                 batch_size=10,   #
                                 verbose=0)

# Evaluate neural network using 10 ten-fold cross-validation
results= cross_val_score(neural_network, features_scaled3D, target3D, cv=kfold)

scores_models['KerasRegressor'] = results
```

```
WARNING:tensorflow:5 out of the last 16 calls to <function Model.make_test_function.<loc
als>.test_function at 0x7f8a89f04e50> triggered tf.function retracing. Tracing is expens
ive and the excessive number of tracings could be due to (1) creating @tf.function repea
tedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects i
nstead of tensors. For (1), please define your @tf.function outside of the loop. For
(2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing.
For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing
and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
WARNING:tensorflow:6 out of the last 17 calls to <function Model.make_test_function.<loc
als>.test_function at 0x7f8a9b0ad8b0> triggered tf.function retracing. Tracing is expens
ive and the excessive number of tracings could be due to (1) creating @tf.function repea
tedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects i
nstead of tensors. For (1), please define your @tf.function outside of the loop. For
(2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing.
For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing
and https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```

In [135…

```
print("\n Results MSE: %.2f (%.2f) " % (results.mean(), results.std()))
```

```
 Results MSE: 78.69 (35.75)
```

Use ensemble method to improve performance and

accuracy

1- Random Forest for regression

2- XGBoost or eXtreme Gradient Boosting for regression

3- Light GBM or light gradient-boosting machine for regression

(CRISP-DM Phase: Modelling Phase)

All these models below come from the same concept of decision trees with differences in order to obtain performance and avoid overfitting.

- Random Forests (RF is used extensively in the industry because provides good results for many problems) https://scikit-learn.org/stable/modules/ensemble.html?highlight=random+forest#forests-of-randomized-trees

The faster development of algorithms based on the technique of **gradient boosting** framework has two principal options very popular in the Kaggle competition:

- XGBoost or eXtreme Gradient Boosting from the Distributed (Deep) Machine Learning Community (DMLC) group. https://xgboost.readthedocs.io/en/stable/
- Light GBM or light gradient-boosting machine development by Microsoft. https://lightgbm.readthedocs.io/en/v3.3.2/

This project will be implemented both algorithms applied for a regression problem.

In theory, Light GBM would be better from the point of view of faster training speed and higher efficiency.

https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMRegressor.html

https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/

In [136…

```
#importing standard libraries
import numpy as np
import pandas as pd
from numpy import mean
from numpy import std
from pandas import Series, DataFrame

#import lightgbm and xgboost
from lightgbm import LGBMRegressor
import lightgbm as lgb
import xgboost as xgb
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score, mean_squared_error


# plot tree

import graphviz
```

In [137…
```
print(xgb.__version__)
print(lgb.__version__)
```

```
1.5.0
3.2.1
```

Prepare the data: features and target for all ensemble models

(CRISP-DM Phase: Data Preparation Phase)

In [138…
```
print('target: ', target.shape)
print('features: ')
features
```

```
target:  (336,)
features:
```

Out[138]:

	Geo	Year	GDP means	emo	Expenditure_Index
0	8	2010	0.00	-1	107.70
1	1	2010	0.22	-1	104.80
2	2	2010	0.40	-1	97.90
3	5	2010	-0.12	1	93.80
4	6	2010	-0.05	1	108.70
...	...	...	...	...	...
331	23	2021	1.12	-1	107.23
332	25	2021	2.60	-1	106.88
333	24	2021	0.28	-1	95.70
334	9	2021	0.75	-1	105.16
335	27	2021	1.43	-1	107.61

336 rows × 5 columns

In [139…
```
#standardizing the training dataset before training.
from sklearn.preprocessing import StandardScaler

# define standard scaler
scaler = StandardScaler()

# transform data
features_scaled = scaler.fit_transform(features)
```

In [140…
```
# Divide the data into training and test sets
x_train, x_test, y_train, y_test = train_test_split(
features_scaled, target, test_size=0.20, random_state=61)

print('features and target: ', features_scaled.shape, target.shape)
print('data for train: ', x_train.shape, y_train.shape)
print('data for test: ', x_test.shape, y_test.shape)
```

```
features and target:  (336, 5) (336,)
data for train:  (268, 5) (268,)
data for test:  (68, 5) (68,)
```

Use ensemble method to improve performance and accuracy

1- Random Forest for regression

2- XGBoost or eXtreme Gradient Boosting for regression

3- Light GBM or light gradient-boosting machine for regression

(CRISP-DM Phase: Modelling Phase)

In [141…
```
models = []
```

- Random Forests (RF is used extensively in the industry because provides good results for many problems)
- XGBoost (XGBoost is used extensively in Kaggle competitions)
- Lightgbm or light gradient-boosting (also popular in Kaggle competition and in theory, faster)

In [142…
```
models.append(('randomforest', RandomForestRegressor(n_estimators = 300, min_samples_spl
                                   min_samples_leaf= 1, max_features = 'sqrt', random_st
                                   max_depth= 10, bootstrap=True)))

models.append(('XGBoost', XGBRegressor(n_estimators=100, max_depth=4,reg_alpha=0.9)
              ))

models.append(('lgmBoost', LGBMRegressor(n_estimators=100, max_depth=4, num_leaves=10)
              ))
```

In [143…
```
results = []
name_model = []
for name, model in models:
    # Conduct k-fold cross-validation
    print ("\n Model:  ", name)
    kfold = KFold(n_splits=10, shuffle=True, random_state=61)
    kf_results = cross_val_score(model,
                             x_train,y_train,
                             cv=kfold,
                             scoring='neg_mean_squared_error', # MSE for regression
                             #scoring='r2', #
                             n_jobs=-1) # use all cpu available

    model.fit( x_train, y_train)
    print(' Score train data:  ', model.score(x_train, y_train))
    print(' Score test data:  ', model.score(x_test, y_test))

    scores_models[name] = kf_results

    results.append(kf_results)
    name_model.append(name)

    # end
```

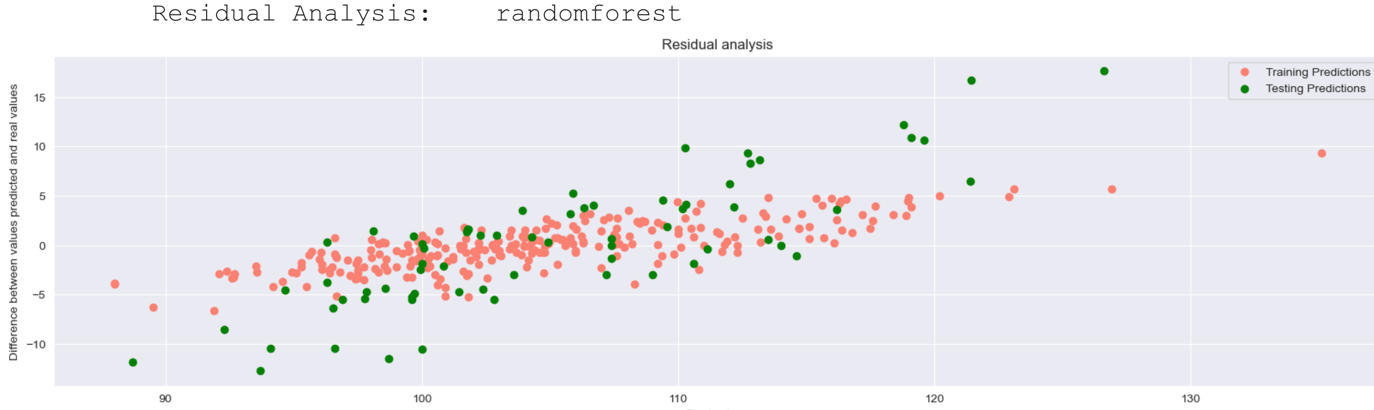
```
 Model:   randomforest
 Score train data:   0.8933731987884356
```

```
Score test data:   0.3414473709887955

Model:   XGBoost
Score train data:   0.9877436193544469
Score test data:   0.41745129345636967

Model:   lgmBoost
Score train data:   0.7002110123634081
Score test data:   0.29173875178425446
```

In [144…

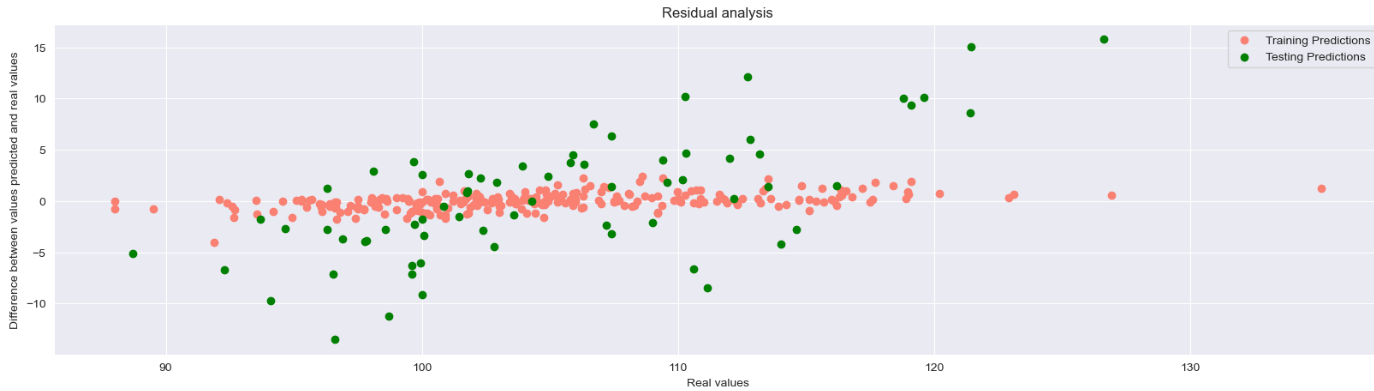
```python
for name, model in models:
    # Predicting the Test set results
    y_pred = model.predict(x_test)
    ytrain_pred = model.predict(x_train)
    print('\n\t\t Model:    ', name)
    print(' \n\t Score train data:  ', model.score(x_train, y_train))
    print('\t Score test data:  ', model.score(x_test, y_test))

    print("\n ")
    print("\t Residual Analysis:   ", name)
    plt.figure(figsize = (20,5))
    plt.scatter(y_train,(y_train-ytrain_pred),color = "salmon",label = 'Training Predict
    plt.scatter(y_test,(y_test-y_pred),color = "green",label = 'Testing Predictions')
    plt.legend()
    plt.xlabel('Real values')
    plt.ylabel('Difference between values predicted and real values')
    plt.title('Residual analysis')
    #plt.set_xlabel('')
    #plt.set_ylabel('')
    #plt.set_title('')

    plt.show()

    print("\n\t  For Test Data:  \n ")
    print("\t MAE: ",mean_absolute_error(y_test, y_pred))
    print("\t MSE: ",mean_squared_error(y_test, y_pred))
    print("\t RMSE: ",np.sqrt(mean_squared_error(y_test, y_pred)))
    print('\n\n')
    # end
```

```
                Model:     randomforest

        Score train data:   0.8933731987884356
        Score test data:   0.3414473709887955


        Residual Analysis:    randomforest
```



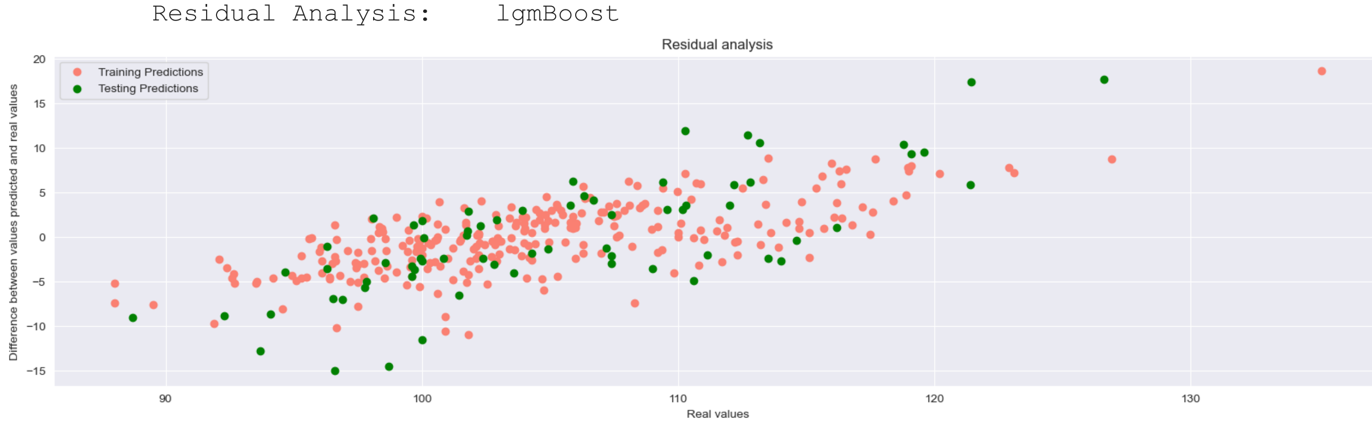

```
         For Test Data:

        MAE:   4.873042075083828
        MSE:   40.68433793341727
        RMSE:   6.378427543949783
```

```
        Model:     XGBoost

Score train data:   0.9877436193544469
Score test data:   0.41745129345636967


Residual Analysis:    XGBoost
```




```
 For Test Data:

MAE:  4.781071014404297
MSE:  35.98893603276913
RMSE:  5.99907793187996
```

```
        Model:     lgmBoost

Score train data:   0.7002110123634081
Score test data:   0.29173875178425446


Residual Analysis:    lgmBoost
```




```
 For Test Data:

MAE:  5.142487199971555
MSE:  43.75525766379267
RMSE:  6.6147757077464595
```

Final comparison between the models based on MSE

Comparative models based on the scoring metrics

(CRISP-DM Phase: Evaluation Phase)

In [145…

```
df_score_models = abs(pd.DataFrame(scores_models))
df_score_models
```

Out[145]:

	KerasRegressor	randomforest	XGBoost	lgmBoost
0	150.446655	48.033553	41.355402	44.683154
1	69.641411	23.683737	35.233136	23.976182
2	87.556435	48.779773	59.491680	50.073500
3	85.074448	18.088374	20.303456	20.242686
4	58.528622	29.666283	24.063735	29.290550
5	24.155834	40.863778	44.377609	37.937648
6	54.060452	16.703616	23.807834	17.896157
7	44.779335	23.920414	58.626959	32.274694
8	86.731270	30.253799	26.530033	32.726593
9	125.902725	36.150512	51.000052	37.795836

In [146…

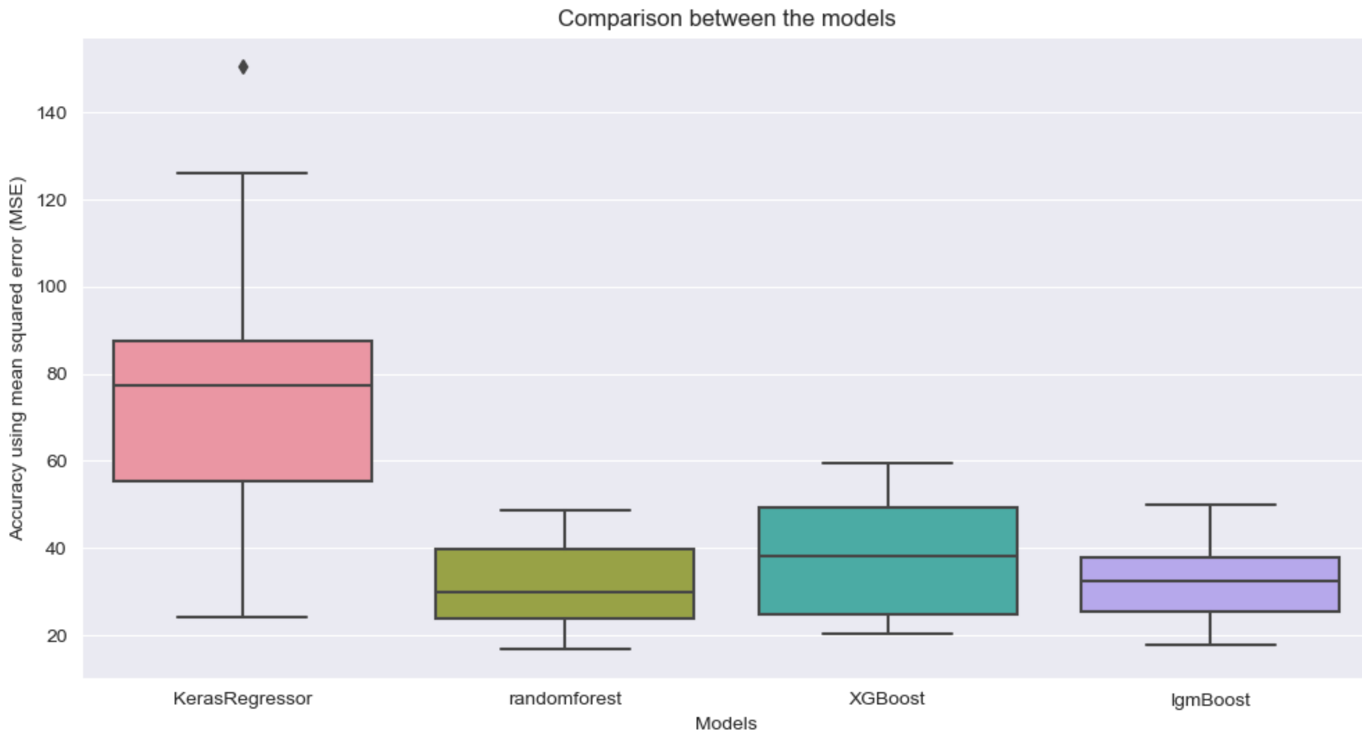
```
df_score_models.describe()
```

Out[146]:

	KerasRegressor	randomforest	XGBoost	lgmBoost
count	10.000000	10.000000	10.000000	10.000000
mean	78.687719	31.614384	38.478990	32.689700
std	37.684328	11.563304	14.734781	10.333781
min	24.155834	16.703616	20.303456	17.896157
25%	55.177494	23.742906	24.680310	25.304774
50%	77.357929	29.960041	38.294269	32.500643
75%	87.350143	39.685461	49.344442	37.902195
max	150.446655	48.779773	59.491680	50.073500

In [147…

```
plt.figure(figsize=(12, 6))
sns.boxplot(data=df_score_models)
#plt.legend()

plt.xlabel('Models')

plt.ylabel('Accuracy using mean squared error (MSE)')
plt.title('Comparison between the models')
plt.show()
```



Statistical analysis: conclusion

Statistical analysis:

Despite the fact that the mse mean in the Random Forest model is less than the mean on the Light gradient-boosting, **31.44 vs 32.68**, the standard deviation is less in the Light gradient-boosting mean that the values of mse are most stable. Less dispersion, therefore, will be chosen the **Light gradient-boosting** as the best model for this problem.

Tune ensemble method: GridSearchCV (Light GBM or Light gradient-boosting)

(CRISP-DM Phase: Evaluation Phase)

GridSearchCV

Based on the Gridsearchcv technique from Sci kit-Learn package it is possible to tunning the Hyper parameter fro the model selected.

This facility allows us to find the best hyper parameter combination to obtain the best results.

In [148…

```
#LGBMRegressor(n_estimators=100, max_depth=4, num_leaves=31

model = LGBMRegressor()

kfold = KFold(n_splits=10, shuffle=True, random_state=61)


n_estimators = [50, 100, 150]
max_depth = [4, 6, 8]
num_leaves = [10, 20, 30]

param_grid = dict(max_depth=max_depth, n_estimators=n_estimators, num_leaves=num_leaves)
```

In [149…

```
#Model Selection

grid_search = GridSearchCV(model,
                           param_grid,
                           scoring="neg_mean_squared_error",
                           #scoring="r2",
                           n_jobs=-1,
                           cv=kfold,
                           verbose=1)


grid_result = grid_search.fit(x_train, y_train)


# summarize results
print("Best: %f using %s" % (-grid_result.best_score_, grid_result.best_params_))

grid_result


# ML Python_Data_Science_Handbook.pdf
# Pag 79
# pag 366 365

# https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.h
# https://scikit-learn.org/stable/modules/model_evaluation.html#mean-squared-error

# It's simple: minimizing MSE is equivalent to maximizing negative-MSE.

# An objective function that the scorer can maximize is just by "convention"
# as the Sklearn documentation suggests
```

```
Fitting 10 folds for each of 27 candidates, totalling 270 fits
Best: 31.983541 using {'max_depth': 6, 'n_estimators': 50, 'num_leaves': 10}
```

Out[149]:

▸ **GridSearchCV**

▸ **estimator: LGBMRegressor**

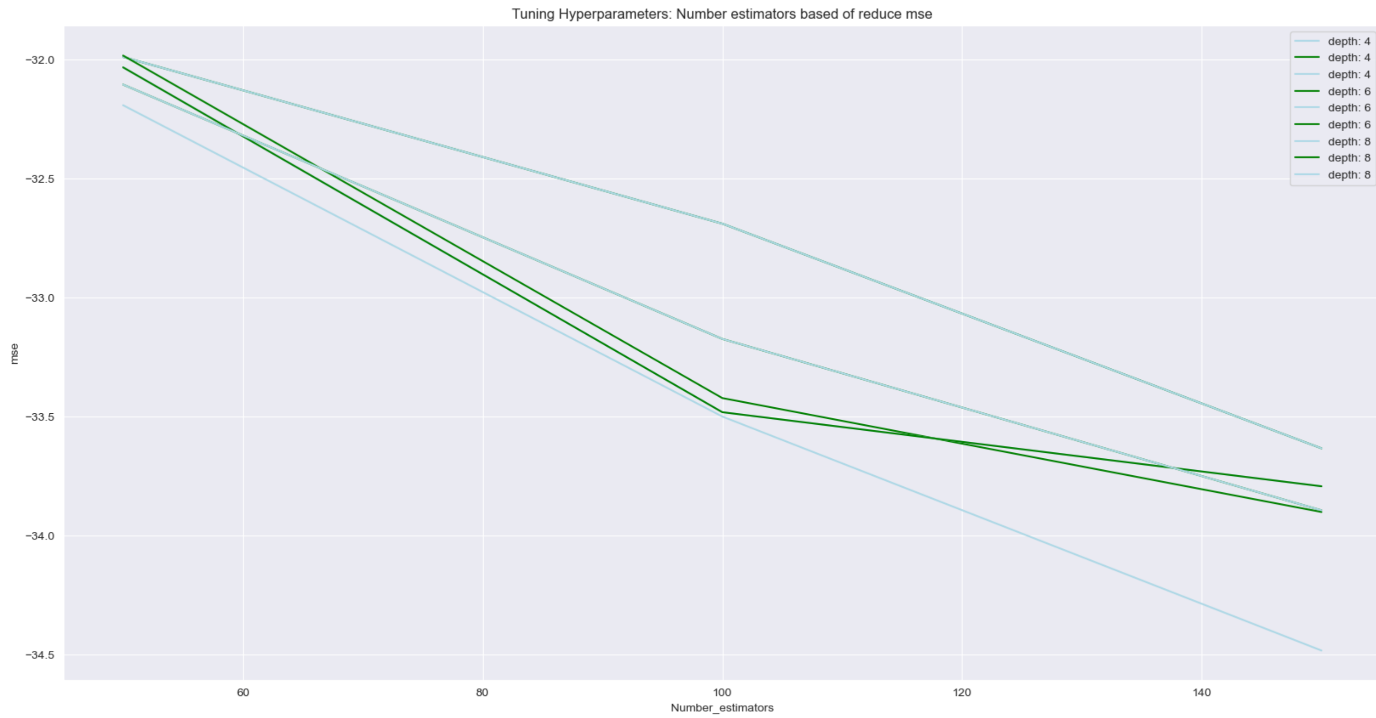
▸ LGBMRegressor

In [150…

```
grid_result.cv_results_
means = grid_result.cv_results_[ 'mean_test_score' ]
stds = grid_result.cv_results_[ 'std_test_score' ]
params = grid_result.cv_results_[ 'params' ]
#for mean, stdev, param in zip(means, stds, params):
#   print("%f (%f) using these parameters: %r" % ((-mean), stdev, param))

# chose the small value
```

In [151…

```
# plot results
plt.figure(figsize = (20,10))
scores = np.array(means).reshape(len(max_depth), len(n_estimators), len(num_leaves))
for i, value in enumerate(max_depth):
    plt.plot(n_estimators, scores[i], label= 'depth: ' + str(value))
plt.legend()
plt.xlabel( 'Number_estimators' )
plt.ylabel( 'mse' )
plt.title('Tuning Hyperparameters: Number estimators based of reduce mse')

plt.show()
```



Final model: Light GBM or Light gradient-boosting

(CRISP-DM Phase: Deployment Phase)

Best hyperparameters to use: {'max_depth': 6, 'n_estimators': 50, 'num_leaves': 10}

In [152…

```
print(x_train.shape, x_test.shape)
print(y_train.shape, y_test.shape)
```

```
(268, 5) (68, 5)
(268,) (68,)
```

In [153…

```
# apply XGBoost regressor with optimous hyper parameters
lgbm_model = LGBMRegressor(n_estimators=50,
                           max_depth=6,
                           num_leaves=10).fit(x_train, y_train)
```

In [154…

```
y_pred=lgbm_model.predict(x_test)         # prediction for test data
y_train_pred=lgbm_model.predict(x_train) # prediction for train data
```

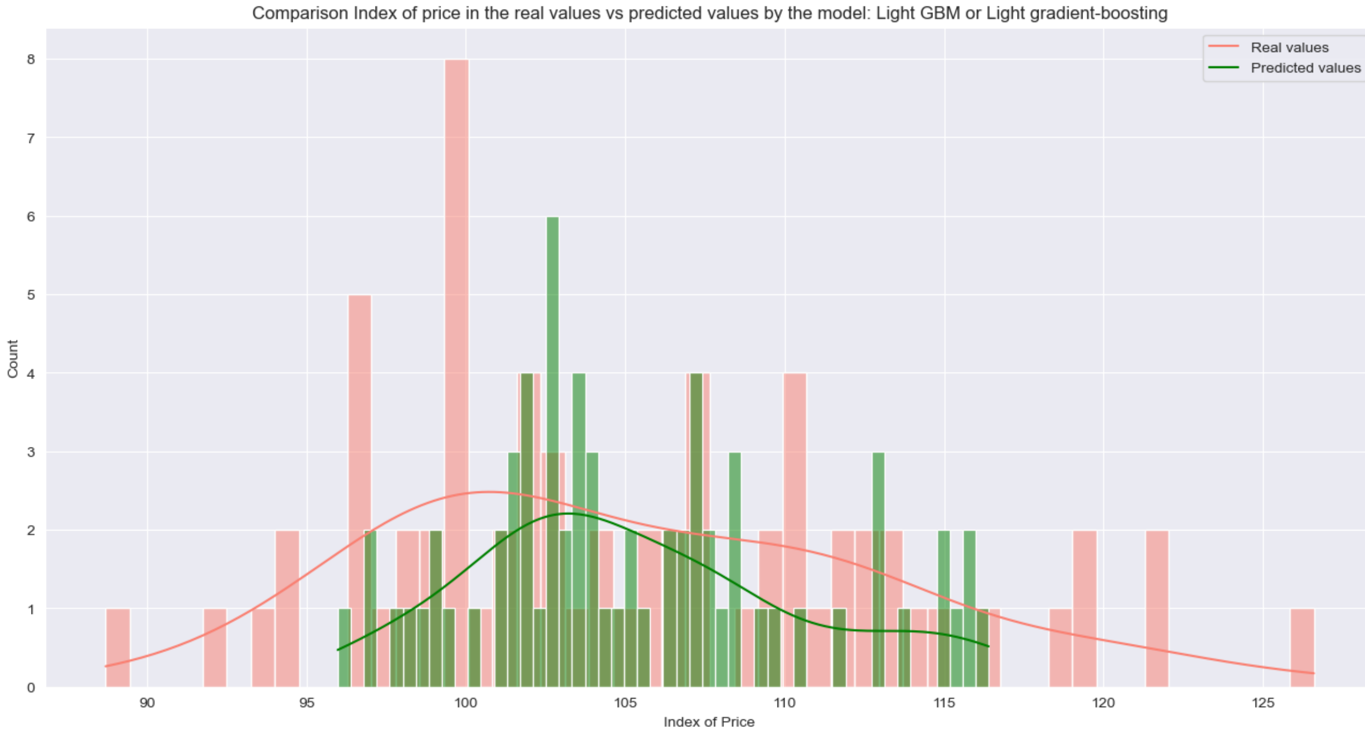
In [155…

```
# metrics

print("R2_Score: ", r2_score(y_test, y_pred))
```

```
R2_Score:  0.2507756195506433
```

In [156…

```
fig = plt.figure(figsize=(16, 8))
sns.set_palette("Paired")
sns.histplot(y_test, kde=True, color='salmon', bins=50)
sns.histplot((y_pred), kde=True, color='green', bins=50)
plt.legend(['Real values', 'Predicted values'])
plt.xlabel( 'Index of Price' )

plt.title('Comparison Index of price in the real values vs predicted values by the model

plt.show()
```



End

Making prediction

In [157…

```
input= np.array([[ 1,  2023,  0.32, -1,  105.90]])
input
```

Out[157]:

```
array([[ 1.000e+00,  2.023e+03,  3.200e-01, -1.000e+00,  1.059e+02]])
```

In [158…

```
#standardizing the training dataset before training.
from sklearn.preprocessing import StandardScaler

# define standard scaler
scaler = StandardScaler()

# transform data
input_scaled = scaler.fit_transform(input)
```

In [159…

```
# Predict Index of price
predicted = lgbm_model.predict(input_scaled)

print('\n Index price prediction: ', predicted)

# [[ 1,  2010,  0.22, -1,  104.80]]
# 1 2010 0.22 -1 104.80
```

```
 Index price prediction:  [103.99619172]
```

In [ ]: