Importación de Librerías

In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib as mpl
from scipy import signal
import matplotlib.pyplot as plt #ojo si hay un 'plot'
import matplotlib.ticker as ticker
import matplotlib.pyplot as plt

from datetime import datetime, timedelta, date, time
import math
from calendar import monthrange
import warnings

from pandas_profiling import ProfileReport
from pandas.plotting import autocorrelation_plot
from pandas.plotting import lag_plot
import pandas_datareader as pdr

import statsmodels.formula.api as smf
import statsmodels.stats.api as sms
from statsmodels.compat import lzip
from statsmodels.stats.outliers_influence import reset_ramsey
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller, kpss
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.nonparametric.smoothers_lowess import lowess
from statsmodels.tsa.stattools import grangercausalitytests
import statsmodels.graphics.tsaplots as sgt
from statsmodels.tsa.statespace.sarimax import SARIMAX


from dateutil.parser import parse
from arch import arch_model
import pmdarima as pm

from sklearn.metrics import mean_squared_error
from sklearn.ensemble import BaggingRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import StackingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import ElasticNet
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import MinMaxScaler
from xgboost import XGBRegressor
from scalecast.Forecaster import Forecaster
from scalecast.MVForecaster import MVForecaster
from scalecast.multiseries import export_model_summaries
from scalecast import GridGenerator
from scalecast.Forecaster import Forecaster
from scalecast import GridGenerator
from scalecast.auxmodels import mlp_stack
from tqdm.notebook import tqdm
```

```
from dateutil.relativedelta import relativedelta


sns.set()
warnings.filterwarnings("ignore")
plt.rcParams.update({'figure.figsize': (10, 7), 'figure.dpi': 120})
sns.set(rc={'figure.figsize':(12,8)})
#pd.set_option('max_colwidth',500)
#pd.set_option('display.max_rows',500)
```

Importación de Datos, limpieza y preparación

DF

Agrupación de diferentes archivos en uno solo

In [2]:
```
f2016 = pd.read_excel("2016.xls")
f2017 = pd.read_excel("2017.xls")
f2018 = pd.read_excel("2018.xls")
f2019 = pd.read_excel("2019.xlsx")
f2020 = pd.read_excel("2020.xlsx")
f2021 = pd.read_excel("2021.xlsx")

df = f2016
df = df.append(f2017)
df = df.append(f2018)
df = df.append(f2019)
df = df.append(f2020)
df = df.append(f2021)
df = df.reset_index(drop=True)
```

Eliminación de columnnas sin uso y relleno de espacios en blanco

In [3]:
```
df = df.drop(columns=['Unnamed: 13']).drop(columns=['Unnamed: 14']).drop(columns=['Unna
df = df.fillna(0)
df = df.replace({'NUSD': 0,}, 552)
```

Unificación de criterios de macroruta

In [4]:
```
df['Macroruta'].unique()
```

Out[4]:
```
array(['Orgánico', 'Inservible', 'Inorganico', 'Inorgánico', 'Organico',
       'Material Vegetal', 'Inorgánicos', 'Orgánicos', '  Organico',
       'inservible', 'Material Vegeteal', 'Material vegetal',
       'material vegetal', 'orgánico', 'reciclaje', 'INSERVIBLE',
       'RECICLABLE', 'ORGANICO', 'MATERIAL VEGETAL', 'Reciclable',
       'RECICLAJE', 'RECICLAJE ', 'ORGANICO ', 'INSERVIBLE  ',
       ' ORGANICO', 'INSERVIBLE ', 'MATERIAL VEGERTAL',
       'CONTAMINADO COVID-19', 'ESPECIALES MUEBLES COLCHONES', 'COVID-19',
       'COVID 19'], dtype=object)
```

In [5]:
```
df = df.replace({'Macroruta': ['Orgánico', 'Organico', 'Orgánicos', '  Organico', 'orgá
df = df.replace({'Macroruta': ['INSERVIBLE', 'Inservible', 'Inorganico', 'Inorgánico',
```

```
df = df.replace({'Macroruta': ['Material Vegetal', 'Material Vegeteal', 'Material veget
df = df.replace({'Macroruta': ['reciclaje', 'RECICLABLE', 'Reciclable','RECICLAJE', 'RE
df = df.replace({'Macroruta': ['CONTAMINADO COVID-19', 'COVID 19']}, 'COVID-19')
df['Macroruta'].unique()
```

Out[5]:
```
array(['ORGANICO', 'INORGANICO', 'VEGETAL', 'RECICLAJE', 'COVID-19',
       'ESPECIALES MUEBLES COLCHONES'], dtype=object)
```

Reorganización de columnas

In [6]:
```
df = df[['NUSD', 'NUAP', 'Placa', 'Fecha', 'Hora de entrada del vehículo',
       'Hora de salida del vehículo', 'Macroruta',
       'Toneladas recogidas en suelo urbano asociadas al barrido y limpieza proveniente
       'Toneladas recogidas en suelo no urbano asociadas al barrido y limpieza provenie
       'Toneladas dispuestas en suelo urbano asociadas a la recoleccion y disposicion p
       'Toneladas dispuestas en suelo no urbano asociadas  a la recoleccion y disposici
       'Toneladas dispuestas en suelo urbano asociadas a la limpieza y corte de zonas v
       'Toneladas dispuestas del servicio ordinario provenientes del área de prestación
```

In [7]:
```
df.sample(n=2)
```

Out[7]:

	NUSD	NUAP	Placa	Fecha	Hora de entrada del vehículo	Hora de salida del vehículo	Macroruta	Toneladas recogidas en suelo urbano asociadas al barrido y limpieza provenientes del área de prestación	Toneladas recogidas en suelo no urbano asociadas al barrido y limpieza provenientes del área de prestación	r p
6502	552.0	1044.0	TTT904	2018-08-08 00:00:00	20:41:50	20:49:23	INORGANICO	0.439	0.0	
2458	552.0	1044.0	OAX012	2017-05-18 00:00:00	01:28:27	01:39:47	ORGANICO	0.000	0.0	

Ajuste de dato con error que no permite que la columna sea reconocida como numérica

In [8]:
```
df.dtypes
```

Out[8]:
```
NUSD
float64
NUAP
float64
Placa
object
Fecha
object
Hora de entrada del vehículo
object
Hora de salida del vehículo
```

```
object
Macroruta
object
Toneladas recogidas en suelo urbano asociadas al barrido y limpieza provenientes del áre
a de prestación                    float64
Toneladas recogidas en suelo no urbano asociadas al barrido y limpieza provenientes del
área de prestación                 float64
Toneladas dispuestas en suelo urbano asociadas a la recoleccion y disposicion provenient
es del área de prestación           object
Toneladas dispuestas en suelo no urbano asociadas  a la recoleccion y disposicion proven
ientes del área de prestación      float64
Toneladas dispuestas en suelo urbano asociadas a la limpieza y corte de zonas verdes pro
venientes del área de prestación   float64
Toneladas dispuestas del servicio ordinario provenientes del área de prestación
float64
dtype: object
```

In [9]:
```
df.iloc[9472]
```

Out[9]:
```
NUSD
552.0
NUAP
1044.0
Placa
TTT904
Fecha
2019-06-10 00:00:00
Hora de entrada del vehículo
02:27:00
Hora de salida del vehículo
02:40:00
Macroruta
ORGANICO
Toneladas recogidas en suelo urbano asociadas al barrido y limpieza provenientes del áre
a de prestación                                    0.0
Toneladas recogidas en suelo no urbano asociadas al barrido y limpieza provenientes del
área de prestación                                 0.0
Toneladas dispuestas en suelo urbano asociadas a la recoleccion y disposicion provenient
es del área de prestación                        6,,910
Toneladas dispuestas en suelo no urbano asociadas  a la recoleccion y disposicion proven
ientes del área de prestación                      0.0
Toneladas dispuestas en suelo urbano asociadas a la limpieza y corte de zonas verdes pro
venientes del área de prestación                   0.0
Toneladas dispuestas del servicio ordinario provenientes del área de prestación
0.0
Name: 9472, dtype: object
```

In [10]:
```
df = df.replace({'Toneladas dispuestas en suelo urbano asociadas a la recoleccion y dis
```

Corrección de fechas ingresadas erroneamente

In [11]:
```
df.at[1213, 'Fecha']=df['Fecha'].loc[1212]
df.at[1214, 'Fecha']=df['Fecha'].loc[1212]
df.at[2526, 'Fecha']=df['Fecha'].loc[2527]
df.at[3720, 'Fecha']=df['Fecha'].loc[3719]
df.at[3999, 'Fecha']=df['Fecha'].loc[3998]
df.at[5160, 'Fecha']=df['Fecha'].loc[5159]
df.at[5161, 'Fecha']=df['Fecha'].loc[5159]
```

```
df.at[5162, 'Fecha']=df['Fecha'].loc[5159]
df.at[5678, 'Fecha']=df['Fecha'].loc[5677]
df.at[5941, 'Fecha']=df['Fecha'].loc[5940]
df.at[5942, 'Fecha']=df['Fecha'].loc[5940]
df.at[6585, 'Fecha']=df['Fecha'].loc[6584]
df.at[6586, 'Fecha']=df['Fecha'].loc[6584]
df.at[6587, 'Fecha']=df['Fecha'].loc[6584]
df.at[6588, 'Fecha']=df['Fecha'].loc[6584]
df.at[6589, 'Fecha']=df['Fecha'].loc[6584]
df.at[6590, 'Fecha']=df['Fecha'].loc[6584]
df.at[6591, 'Fecha']=df['Fecha'].loc[6584]
df.at[6592, 'Fecha']=df['Fecha'].loc[6584]
df.at[6593, 'Fecha']=df['Fecha'].loc[6584]
df.at[6594, 'Fecha']=df['Fecha'].loc[6595]
df.at[7636, 'Fecha']=df['Fecha'].loc[7635]
df.at[13208, 'Fecha']=df['Fecha'].loc[13207]
df.at[13209, 'Fecha']=df['Fecha'].loc[13207]
df.at[13210, 'Fecha']=df['Fecha'].loc[13207]
df.at[13211, 'Fecha']=df['Fecha'].loc[13207]
df.at[13212, 'Fecha']=df['Fecha'].loc[13207]
df.at[13213, 'Fecha']=df['Fecha'].loc[13207]
df.at[13214, 'Fecha']=df['Fecha'].loc[13207]
df.at[13215, 'Fecha']=df['Fecha'].loc[13207]
df.at[13216, 'Fecha']=df['Fecha'].loc[13207]
df.at[13217, 'Fecha']=df['Fecha'].loc[13223]
df.at[13218, 'Fecha']=df['Fecha'].loc[13223]
df.at[13219, 'Fecha']=df['Fecha'].loc[13223]
df.at[13220, 'Fecha']=df['Fecha'].loc[13223]
df.at[13221, 'Fecha']=df['Fecha'].loc[13223]
df.at[13222, 'Fecha']=df['Fecha'].loc[13223]
df.at[13998, 'Fecha']=df['Fecha'].loc[13997]
df.at[14139, 'Fecha']=df['Fecha'].loc[14138]
df.at[14289, 'Fecha']=df['Fecha'].loc[14288]
df.at[14589, 'Fecha']=df['Fecha'].loc[14588]
df.at[14590, 'Fecha']=df['Fecha'].loc[14588]
df.at[14772, 'Fecha']=df['Fecha'].loc[14771]
df.at[14813, 'Fecha']=df['Fecha'].loc[14812]
df.at[14819, 'Fecha']=df['Fecha'].loc[14818]
df.at[14900, 'Fecha']=df['Fecha'].loc[14899]
df.at[15558, 'Fecha']=df['Fecha'].loc[15557]
df.at[15662, 'Fecha']=df['Fecha'].loc[15663]
df.at[15898, 'Fecha']=df['Fecha'].loc[15899]
df.at[16786, 'Fecha']=df['Fecha'].loc[16785]
df.at[16787, 'Fecha']=df['Fecha'].loc[16785]
df.at[17502, 'Fecha']=df['Fecha'].loc[17501]
df.at[17503, 'Fecha']=df['Fecha'].loc[17501]

df['Fecha'] = pd.to_datetime(df['Fecha'], infer_datetime_format=True)
df = df.assign(DeltaDias=0)
n, m= df.shape
for x in range(1, n):
    df.at[x, 'DeltaDias'] = df['Fecha'].loc[x]-df['Fecha'].loc[x-1]

for num in range(1, len(df)):
    if abs(df['DeltaDias'].loc[num])>timedelta(days=3) and df['Fecha'].loc[num-1]==df['
        df.at[num, 'Fecha']=df['Fecha'].loc[num-1]
        df.at[num, 'DeltaDias'] = df['Fecha'].loc[num]-df['Fecha'].loc[num-1]
        df.at[num+1, 'DeltaDias'] = df['Fecha'].loc[num+1]-df['Fecha'].loc[num]

    if df['DeltaDias'].loc[num]==timedelta(days=-1) and df['Fecha'].loc[num-1]==df['Fec
```

```
        df.at[num, 'Fecha']=df['Fecha'].loc[num-1]
        df.at[num, 'DeltaDias'] = df['Fecha'].loc[num]-df['Fecha'].loc[num-1]
        df.at[num+1, 'DeltaDias'] = df['Fecha'].loc[num+1]-df['Fecha'].loc[num]

for x in range(1, len(df)):
    df.at[x, 'DeltaDias'] = df['Fecha'].loc[x]-df['Fecha'].loc[x-1]

df['DeltaDias'].unique()
```

Out[11]:
```
array([0, Timedelta('0 days 00:00:00'), Timedelta('1 days 00:00:00'),
       Timedelta('2 days 00:00:00'), Timedelta('3 days 00:00:00')],
      dtype=object)
```

Correción de celdas mal llenadas (Toneladas)

In [12]:
```
df=df.assign(DeltaTotal=0.0)
for num in range(len(df)):
    df.at[num, 'DeltaTotal']=float(df.iloc[num, [7]])+float(df.iloc[num, [8]])+float(df
r=df[df['DeltaTotal']!=0].index
for num in range(len(r)):
    df.at[r[num], 'Toneladas dispuestas del servicio ordinario provenientes del área de
```

Creación de columna 'Hora', 'DiaSemena', 'Semana' y 'Año'

In [13]:
```
df.at[11233, 'Hora de entrada del vehículo']=datetime.strptime('12:00:00',"%H:%M:%S").t
df.at[11705, 'Hora de entrada del vehículo']=datetime.strptime('23:30:00',"%H:%M:%S").t
for num in range(len(df)):
    df.at[num, 'Hora'] = df['Hora de entrada del vehículo'].loc[num].hour
df['DiaSemana'] = df['Fecha'].dt.weekday+1
df['Semana'] = df['Fecha'].dt.week
df['Año'] = df['Fecha'].dt.year

df.at[1183, 'Año']=2016
df.at[1184, 'Año']=2016
df.at[7834, 'Año']=2019
df.at[7835, 'Año']=2019
df.at[7836, 'Año']=2019
df.at[7837, 'Año']=2019
df.at[11634, 'Año']=2020
df.at[11635, 'Año']=2020
df.at[11636, 'Año']=2020
df.at[11637, 'Año']=2020
df.at[11638, 'Año']=2020
df.at[11639, 'Año']=2020
df.at[11640, 'Año']=2020
df.at[11641, 'Año']=2020
df.at[11642, 'Año']=2020
df.at[11643, 'Año']=2020
df.at[11644, 'Año']=2020
df.at[11645, 'Año']=2020
df.at[11646, 'Año']=2020
df.at[11647, 'Año']=2020
df.at[11648, 'Año']=2020
df.at[11649, 'Año']=2020
df.at[11650, 'Año']=2020
df.at[11651, 'Año']=2020
df.at[11652, 'Año']=2020
df.at[11653, 'Año']=2020
```

```
df.at[11654, 'Año']=2020
df.at[11655, 'Año']=2020
df.at[11656, 'Año']=2020
df.at[11657, 'Año']=2020
df.at[15489, 'Año']=2020
df.at[15490, 'Año']=2020
```

Guardado de archivo limpio

In [14]:
```
df.to_csv('CONSOLIDADO.csv')
```

DF2

Pivote de DataFrame para generación de DataFrame Alterno (df2), adición de habitantes y variable de confinamiento COVID

In [15]:
```
df
```

Out[15]:

	NUSD	NUAP	Placa	Fecha	Hora de entrada del vehículo	Hora de salida del vehículo	Macroruta	Toneladas recogidas en suelo urbano asociadas al barrido y limpieza provenientes del área de prestación	Toneladas recogidas en suelo no urbano asociadas al barrido y limpieza provenientes del área de prestación	re pr
0	552.0	1044.0	TTT904	2016-08-01	10:21:27	10:35:28	ORGANICO	0.000	0.0	
1	552.0	1044.0	TTU023	2016-08-01	10:49:21	11:01:11	ORGANICO	0.000	0.0	
2	552.0	1044.0	OAX012	2016-08-01	11:06:49	11:27:41	ORGANICO	0.000	0.0	
3	552.0	1044.0	OAX013	2016-08-01	13:01:25	13:18:29	INORGANICO	0.216	0.0	
4	552.0	1044.0	TTU023	2016-08-01	14:42:30	15:05:37	ORGANICO	0.000	0.0	
...	...	...	...	...	...	...	...	...	...	
17868	552.0	1044.0	WPS527	2021-07-31	01:00:00	01:10:00	INORGANICO	0.000	0.0	
17869	552.0	1044.0	WPS503	2021-07-31	03:10:00	03:20:00	INORGANICO	0.000	0.0	
17870	552.0	1044.0	TTT904	2021-07-31	03:30:00	03:40:00	INORGANICO	0.000	0.0	
17871	552.0	1044.0	TTU023	2021-07-31	03:40:00	03:50:00	INORGANICO	0.000	0.0	

	NUSD	NUAP	Placa	Fecha	Hora de entrada del vehículo	Hora de salida del vehículo	Macroruta	Toneladas recogidas en suelo urbano asociadas al barrido y limpieza provenientes del área de prestación	Toneladas recogidas en suelo no urbano asociadas al barrido y limpieza provenientes del área de prestación	re pr
17872	552.0	1044.0	WPS527	2021-07-31	04:20:00	04:30:00	INORGANICO	0.000	0.0	

17873 rows × 19 columns

In [16]:
```
df2=pd.DataFrame(df.groupby(['Año', 'Semana', 'Macroruta']).sum())
df2=df2.reset_index()
df2=df2.drop('NUSD', axis=1)
df2=df2.drop('NUAP', axis=1)
df2=df2.drop('DeltaTotal', axis=1)
df2=df2.drop('Hora', axis=1)
df2=df2.drop('DiaSemana', axis=1)
R = pd.get_dummies(df2['Macroruta'])
R = R.astype(float)
df2 = df2.merge(R, how='inner', left_index=True, right_index=True)
df2 = df2.drop('Macroruta', axis=1)

for num in range(len(df2)):
    df2.at[num, 'COVID-19']=df2.loc[num, 'Toneladas dispuestas del servicio ordinario p
    df2.at[num, 'ESPECIALES MUEBLES COLCHONES']=df2.loc[num, 'Toneladas dispuestas del
    df2.at[num, 'INORGANICO']=df2.loc[num, 'Toneladas dispuestas del servicio ordinario
    df2.at[num, 'ORGANICO']=df2.loc[num, 'Toneladas dispuestas del servicio ordinario p
    df2.at[num, 'RECICLAJE']=df2.loc[num, 'Toneladas dispuestas del servicio ordinario
    df2.at[num, 'VEGETAL']=df2.loc[num, 'Toneladas dispuestas del servicio ordinario pr

df2 = pd.DataFrame(df2.groupby(['Año', 'Semana'], as_index=False).sum())
df2 = df2.drop('Toneladas recogidas en suelo urbano asociadas al barrido y limpieza pro
df2 = df2.drop('Toneladas recogidas en suelo no urbano asociadas al barrido y limpieza
df2 = df2.drop('Toneladas dispuestas en suelo urbano asociadas a la recoleccion y dispo
df2 = df2.drop('Toneladas dispuestas en suelo no urbano asociadas  a la recoleccion y d
df2 = df2.drop('Toneladas dispuestas en suelo urbano asociadas a la limpieza y corte de
df2 = df2.drop('Toneladas dispuestas del servicio ordinario provenientes del área de pr
df2 = df2.drop('COVID-19', axis=1)
df2 = df2.drop('ESPECIALES MUEBLES COLCHONES', axis=1)

aux = pd.DataFrame(df.groupby(['Año', 'Semana'], as_index=False).min())['Fecha']
aux.loc[203] = aux.loc[203] - timedelta(days=1)
df2 = df2.merge(aux, how='inner', left_index=True, right_index=True)
df2
```

Out[16]:

	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Fecha
0	2016	31	119.565	100.120	0.00	0.000	2016-08-01
1	2016	32	124.960	104.175	0.00	0.000	2016-08-08

	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Fecha
2	2016	33	124.730	104.150	0.00	0.000	2016-08-15
3	2016	34	141.650	105.110	0.00	0.000	2016-08-22
4	2016	35	150.138	90.070	0.00	0.000	2016-08-29
...	...	...	...	...	...	...	...
256	2021	26	248.360	85.890	14.03	1.320	2021-06-28
257	2021	27	245.700	82.160	22.60	2.620	2021-07-05
258	2021	28	225.760	85.820	24.29	3.205	2021-07-12
259	2021	29	248.790	73.920	22.85	2.785	2021-07-19
260	2021	30	217.240	88.090	21.38	1.297	2021-07-26

261 rows × 7 columns

In [17]:

```
hab = pd.read_csv("habitantes.csv",";")
hab['Fecha'] = pd.to_datetime(hab['Fecha'], format='%d/%m/%Y')
hab['Año'] = hab['Fecha'].dt.year
hab['Semana'] = hab['Fecha'].dt.week

hab.at[153, 'Año']=2016
hab.at[1614, 'Año']=2020
hab.at[1615, 'Año']=2020
hab.at[1616, 'Año']=2020
hab.at[1979, 'Año']=2021
hab.at[1980, 'Año']=2021
hab = pd.DataFrame(hab.groupby(['Año', 'Semana'], as_index=False).mean())
hab['Habitantes'] = hab['Habitantes'].astype(int)
df2 = df2.merge(hab['Habitantes'], right_index=True, left_index=True)
df2 = df2.set_index('Fecha')
df2
```

Out[17]:

Fecha	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Habitantes
2016-08-01	2016	31	119.565	100.120	0.00	0.000	60305
2016-08-08	2016	32	124.960	104.175	0.00	0.000	60328
2016-08-15	2016	33	124.730	104.150	0.00	0.000	60351
2016-08-22	2016	34	141.650	105.110	0.00	0.000	60374
2016-08-29	2016	35	150.138	90.070	0.00	0.000	60397
...	...	...	...	...	...	...	...
2021-06-28	2021	26	248.360	85.890	14.03	1.320	66346
2021-07-05	2021	27	245.700	82.160	22.60	2.620	66370
2021-07-12	2021	28	225.760	85.820	24.29	3.205	66395
2021-07-19	2021	29	248.790	73.920	22.85	2.785	66419

Fecha	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Habitantes
2021-07-26	2021	30	217.240	88.090	21.38	1.297	66443

261 rows × 7 columns

In [18]:
```
df2['COVID'] = 0
df2.loc[df2.index >= '2020-03-20', 'COVID'] = 1
df2.loc[df2.index >= '2020-06-01', 'COVID'] = 0
df2
```

Out[18]:

Fecha	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Habitantes	COVID
2016-08-01	2016	31	119.565	100.120	0.00	0.000	60305	0
2016-08-08	2016	32	124.960	104.175	0.00	0.000	60328	0
2016-08-15	2016	33	124.730	104.150	0.00	0.000	60351	0
2016-08-22	2016	34	141.650	105.110	0.00	0.000	60374	0
2016-08-29	2016	35	150.138	90.070	0.00	0.000	60397	0
...	...	...	...	...	...	...	...	...
2021-06-28	2021	26	248.360	85.890	14.03	1.320	66346	0
2021-07-05	2021	27	245.700	82.160	22.60	2.620	66370	0
2021-07-12	2021	28	225.760	85.820	24.29	3.205	66395	0
2021-07-19	2021	29	248.790	73.920	22.85	2.785	66419	0
2021-07-26	2021	30	217.240	88.090	21.38	1.297	66443	0

261 rows × 8 columns

Escalado

In [19]:
```
scaler = MinMaxScaler()
scaled = pd.DataFrame(scaler.fit_transform(df2), index=df2.index)
scaled.columns =['Año', 'Semana', 'INORGANICO', 'ORGANICO', 'RECICLAJE', 'VEGETAL', 'Ha
scaled
```

Out[19]:

Fecha	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Habitantes	COVID
2016-08-01	0.0	0.576923	0.111035	0.488454	0.000000	0.000000	0.000000	0.0
2016-08-08	0.0	0.596154	0.133391	0.515754	0.000000	0.000000	0.003747	0.0
2016-08-15	0.0	0.615385	0.132438	0.515586	0.000000	0.000000	0.007494	0.0
2016-08-22	0.0	0.634615	0.202553	0.522049	0.000000	0.000000	0.011241	0.0

Fecha	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Habitantes	COVID
2016-08-29	0.0	0.653846	0.237726	0.420790	0.000000	0.000000	0.014989	0.0
...	...	...	...	...	...	...	...	...
2021-06-28	1.0	0.480769	0.644746	0.392648	0.185056	0.072848	0.984197	0.0
2021-07-05	1.0	0.500000	0.633723	0.367535	0.298094	0.144592	0.988107	0.0
2021-07-12	1.0	0.519231	0.551094	0.392177	0.320385	0.176876	0.992180	0.0
2021-07-19	1.0	0.538462	0.646527	0.312058	0.301392	0.153698	0.996090	0.0
2021-07-26	1.0	0.557692	0.515788	0.407460	0.282002	0.071578	1.000000	0.0

261 rows × 8 columns

EDA

In [20]:
```
df2.describe()
```

Out[20]:

	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Habitantes	C
count	261.000000	261.000000	261.000000	261.000000	261.000000	261.000000	261.000000	261.00
mean	2018.582375	26.601533	159.428344	103.625690	14.639117	5.192360	63345.049808	0.03
std	1.500614	15.097702	38.650372	23.798345	14.707810	3.613001	1781.846527	0.19
min	2016.000000	1.000000	92.770000	27.570000	0.000000	0.000000	60305.000000	0.00
25%	2017.000000	14.000000	133.640000	88.020000	0.000000	2.550000	61805.000000	0.00
50%	2019.000000	27.000000	145.820000	104.580000	18.766000	4.830000	63351.000000	0.00
75%	2020.000000	40.000000	176.067000	118.080000	27.340000	8.020000	64874.000000	0.00
max	2021.000000	53.000000	334.090000	176.100000	75.815000	18.120000	66443.000000	1.00

In [21]:
```
scaled.describe()
```

Out[21]:

	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Habitantes	COVI
count	261.000000	261.000000	261.000000	261.000000	261.000000	261.000000	261.000000	261.00000
mean	0.516475	0.492337	0.276224	0.512056	0.193090	0.286554	0.495283	0.03831
std	0.300123	0.290340	0.160162	0.160226	0.193996	0.199393	0.290298	0.19232
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.200000	0.250000	0.169360	0.406988	0.000000	0.140728	0.244379	0.00000
50%	0.600000	0.500000	0.219833	0.518481	0.247524	0.266556	0.496253	0.00000

	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Habitantes	COVI
75%	0.800000	0.750000	0.345172	0.609372	0.360615	0.442605	0.744379	0.00000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000

In [22]:
```
prof = ProfileReport(df2)
prof.to_file(output_file='outputSemanal.html')
```

In [23]:
```
prof = ProfileReport(scaled)
prof.to_file(output_file='outputSemanalScaled.html')
```

Regresión Lineal

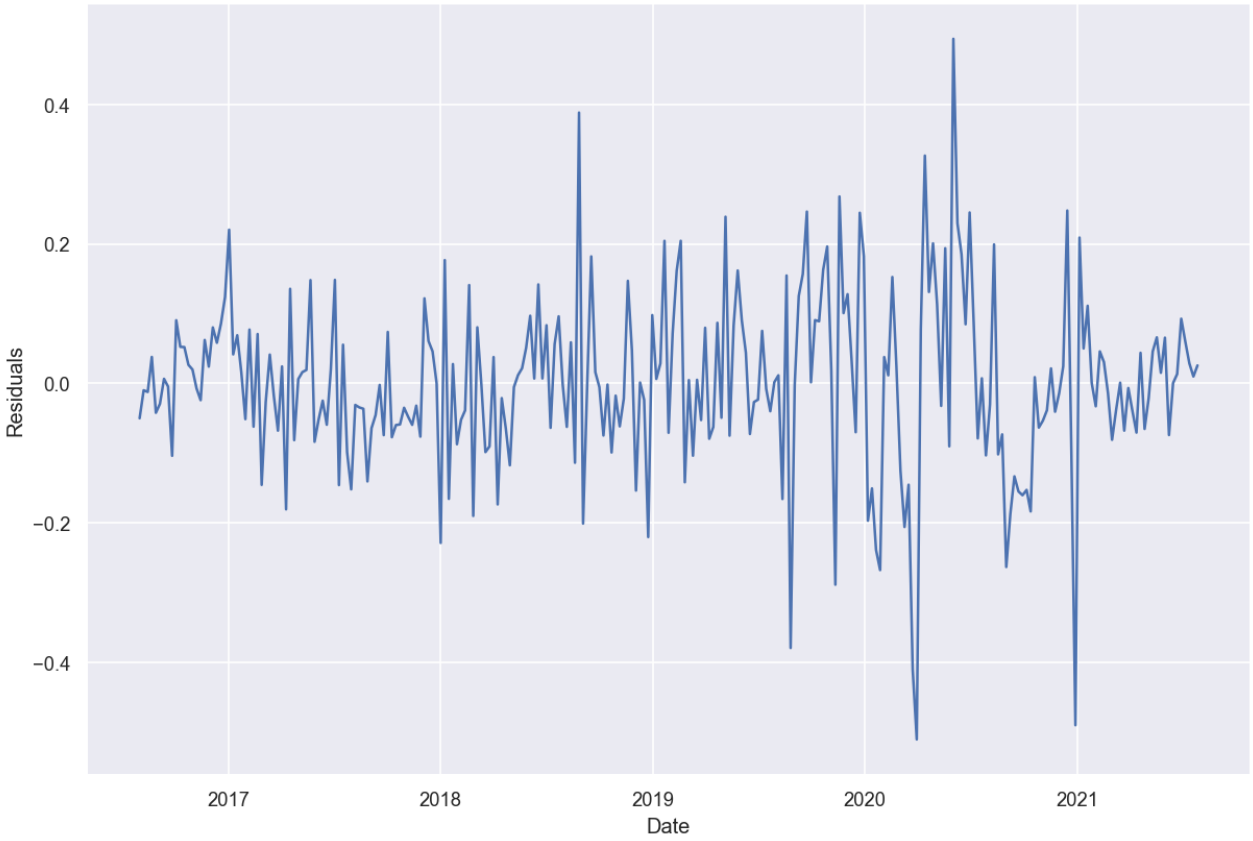
In [24]:
```
formula = 'ORGANICO ~ Habitantes + INORGANICO + RECICLAJE + VEGETAL +  COVID + Semana +
results = smf.ols(formula, scaled).fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:               ORGANICO   R-squared:                       0.373
Model:                            OLS   Adj. R-squared:                  0.356
Method:                 Least Squares   F-statistic:                     21.54
Date:                Sat, 08 Oct 2022   Prob (F-statistic):           1.02e-22
Time:                        13:14:16   Log-Likelihood:                 169.11
No. Observations:                 261   AIC:                            -322.2
Df Residuals:                     253   BIC:                            -293.7
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.3757      0.243      1.548      0.123      -0.102       0.854
Habitantes    -1.6626      2.029     -0.819      0.413      -5.658       2.333
INORGANICO    -0.6467      0.076     -8.523      0.000      -0.796      -0.497
RECICLAJE     -0.0665      0.078     -0.852      0.395      -0.220       0.087
VEGETAL        0.0930      0.042      2.200      0.029       0.010       0.176
COVID          0.0323      0.045      0.721      0.472      -0.056       0.121
Semana         0.4068      0.400      1.017      0.310      -0.381       1.194
Año            1.7875      2.024      0.883      0.378      -2.198       5.773
==============================================================================
Omnibus:                       21.966   Durbin-Watson:                   1.751
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               70.107
Skew:                          -0.227   Prob(JB):                     5.98e-16
Kurtosis:                       5.498   Cond. No.                         519.
==============================================================================
```

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

Testing for heteroscedasticity

In [25]:
```
plt.figure(1)
plt.plot(results.resid)
plt.xlabel('Date')
plt.ylabel('Residuals')
plt.grid(True)
plt.show()
```



Breusch-Pagan Heteroskedasticity test

In [26]:
```
name = ['Lagrange multiplier statistic', 'p-value',
'f-value', 'f p-value']
test = sms.het_breuschpagan(results.resid, results.model.exog)
lzip(name, test)
```

Out[26]:
```
[('Lagrange multiplier statistic', 33.656628620484696),
 ('p-value', 1.996898840790914e-05),
 ('f-value', 5.350702388017734),
 ('f p-value', 9.954508397379168e-06)]
```

Durbin-Watson test

In [27]:
```
residuals = results.resid
sms.durbin_watson(residuals)
```

```
1.751138784485665
```

Out[27]:

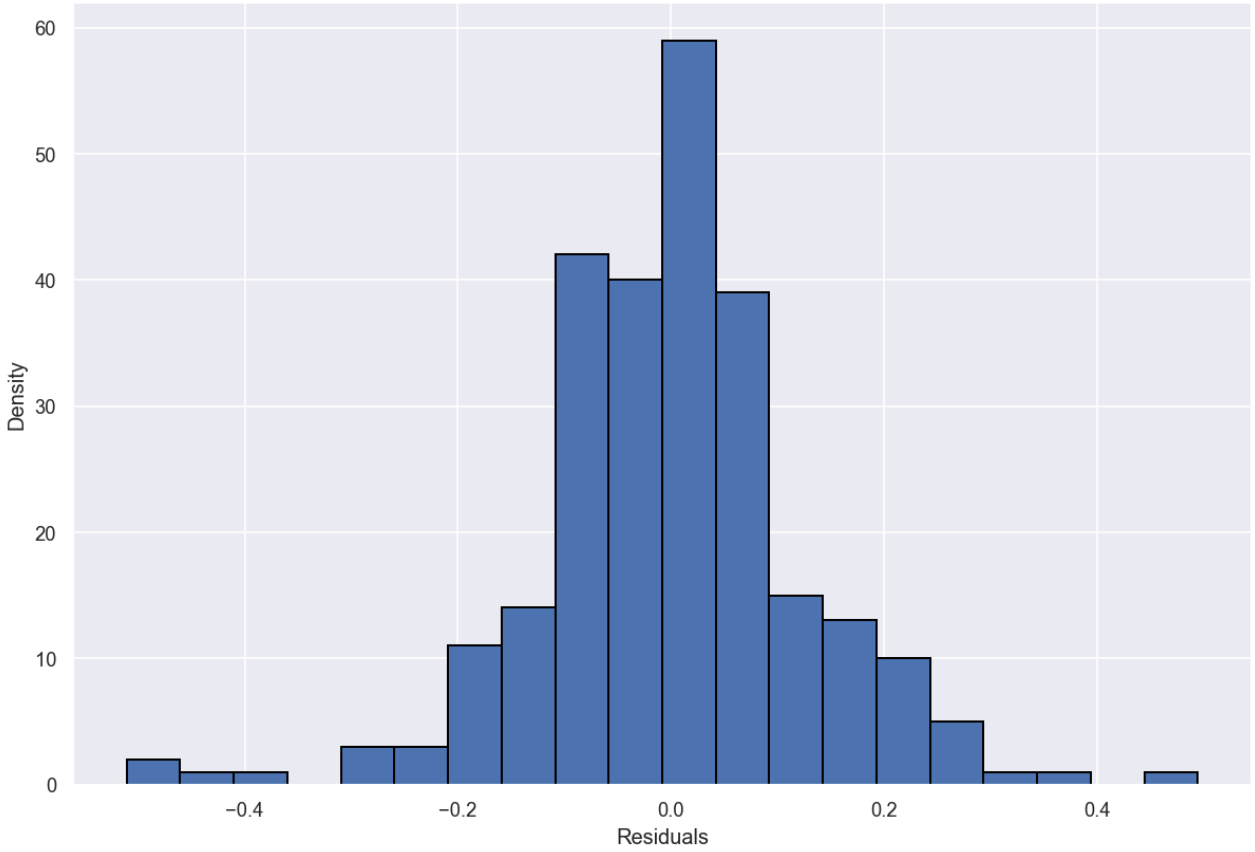
Breusch-Godfrey test

In [28]:
```
name = ['Lagrange multiplier statistic', 'p-value',
'f-value', 'f p-value']
results1 = sms.acorr_breusch_godfrey(results, 10)
lzip(name, results1)
```

Out[28]:
```
[('Lagrange multiplier statistic', 17.15337086342542),
 ('p-value', 0.07103841777392257),
 ('f-value', 1.70938148071663),
 ('f p-value', 0.0791733336563704)]
```

Testing for non-normality

In [29]:
```
plt.figure(1)
plt.hist(residuals,20,edgecolor='black',linewidth=1.2)
plt.xlabel('Residuals')
plt.ylabel('Density')
plt.show()
```



In [30]:
```
name = ['Jarque-Bera', 'Chi^2 two-tail prob.', 'Skew', 'Kurtosis']
test = sms.jarque_bera(residuals)
lzip(name, test)
```

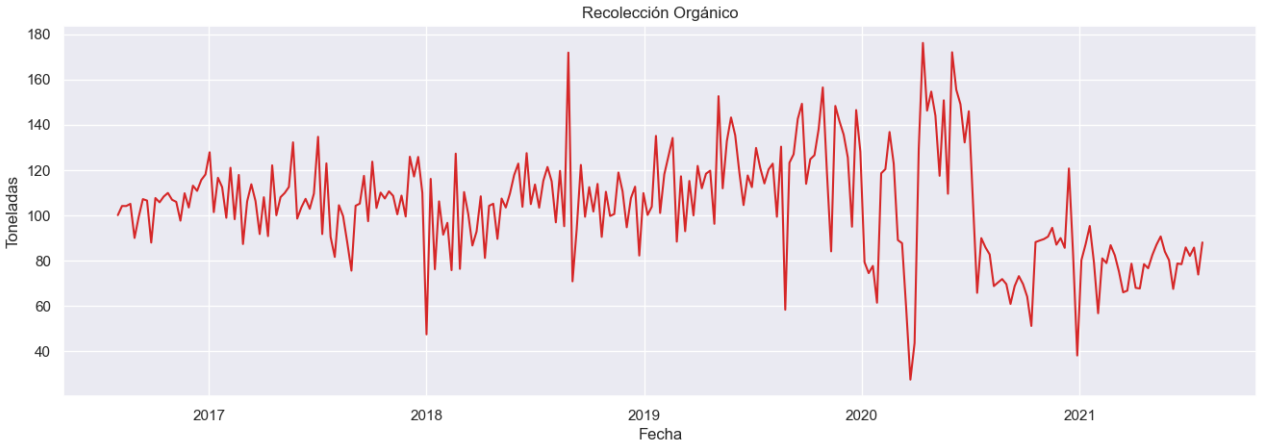
Out[30]:
```
[('Jarque-Bera', 70.10658273312465),
 ('Chi^2 two-tail prob.', 5.97790469637907e-16),
 ('Skew', -0.22692390524576023),
 ('Kurtosis', 5.498120190302976)]
```

In [31]:
```
reset_ramsey(results,degree=4)
```

Out[31]:
```
<class 'statsmodels.stats.contrast.ContrastResults'>
<F test: F=2.64335527672813, p=0.04982749097322095, df_denom=250, df_num=3>
```

Time Series

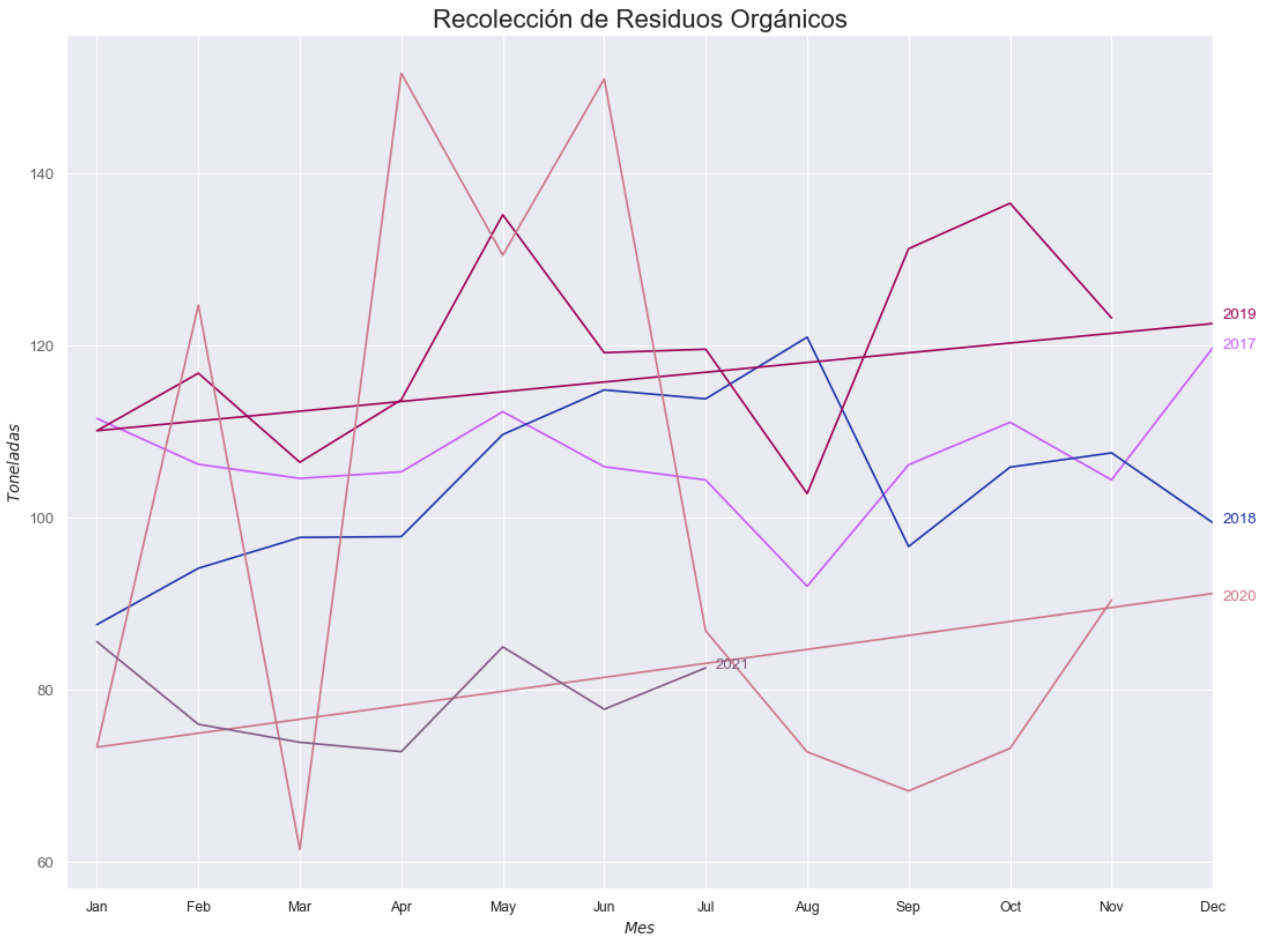
In [32]:
```
def plot_df(df2, x, y, title="Recolección Orgánico", xlabel='Fecha', ylabel='Toneladas'
    plt.figure(figsize=(16,5), dpi=dpi)
    plt.plot(x, y, color='tab:red')
    plt.gca().set(title=title, xlabel=xlabel, ylabel=ylabel)
    plt.show()

plot_df(df2, x=df2.index, y=df2.ORGANICO)
```



Seasonal Plot of a Time Series

In [33]:
```
df2 = df2.assign(Mes=0)

# Prepare data
df2['Mes'] = [d.strftime('%b') for d in df2.index]
years = df2['Año'].unique()

df3 = pd.DataFrame(df2.groupby(['Año', 'Mes'], as_index=False, sort=False).mean())

# Prep Colors
np.random.seed(100)
mycolors = np.random.choice(list(mpl.colors.XKCD_COLORS.keys()), len(years), replace=Fa

# Draw Plot
plt.figure(figsize=(16,12), dpi= 80)
for i, y in enumerate(years):
    if i > 0:
        plt.plot('Mes', 'ORGANICO', data=df3.loc[df3['Año']==y, :], color=mycolors[i],
        plt.text(df3.loc[df3['Año']==y, :].shape[0]-.9, df3.loc[df3['Año']==y, 'ORGANIC

# Decoration
plt.gca().set(xlim=(-0.3, 11), ylabel='$Toneladas$', xlabel='$Mes$')
plt.yticks(fontsize=12, alpha=.7)
```

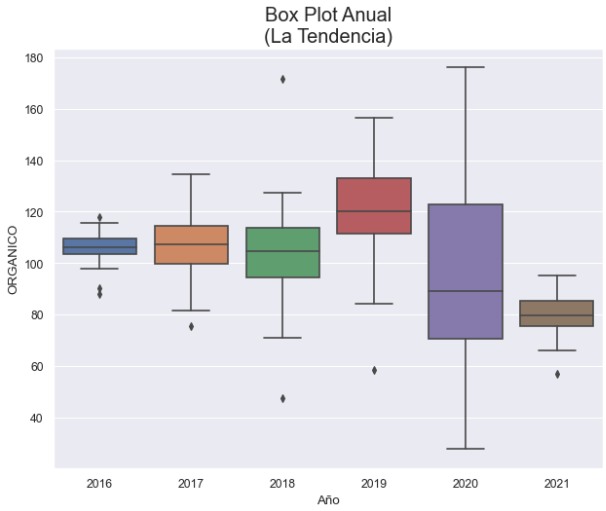
```
plt.title("Recolección de Residuos Orgánicos", fontsize=20)
plt.show()
```

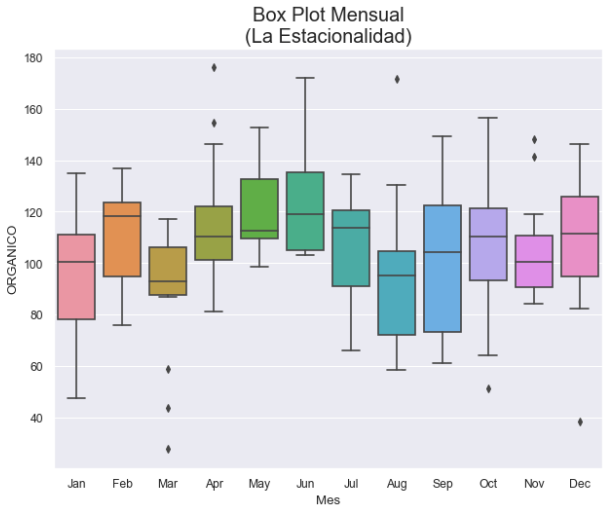


Boxplot of Month-wise (Seasonal) and Year-wise (trend) Distribution

In [34]:

```
# Draw Plot
fig, axes = plt.subplots(1, 2, figsize=(20,7), dpi= 80)
sns.boxplot(x='Año', y='ORGANICO', data=df2, ax=axes[0])
sns.boxplot(x='Mes', y='ORGANICO', data=df2.loc[~df2['Año'].isin([2016, 2021]), :])

# Set Title
axes[0].set_title('Box Plot Anual\n(La Tendencia)', fontsize=18);
axes[1].set_title('Box Plot Mensual\n(La Estacionalidad)', fontsize=18)
plt.show()
```

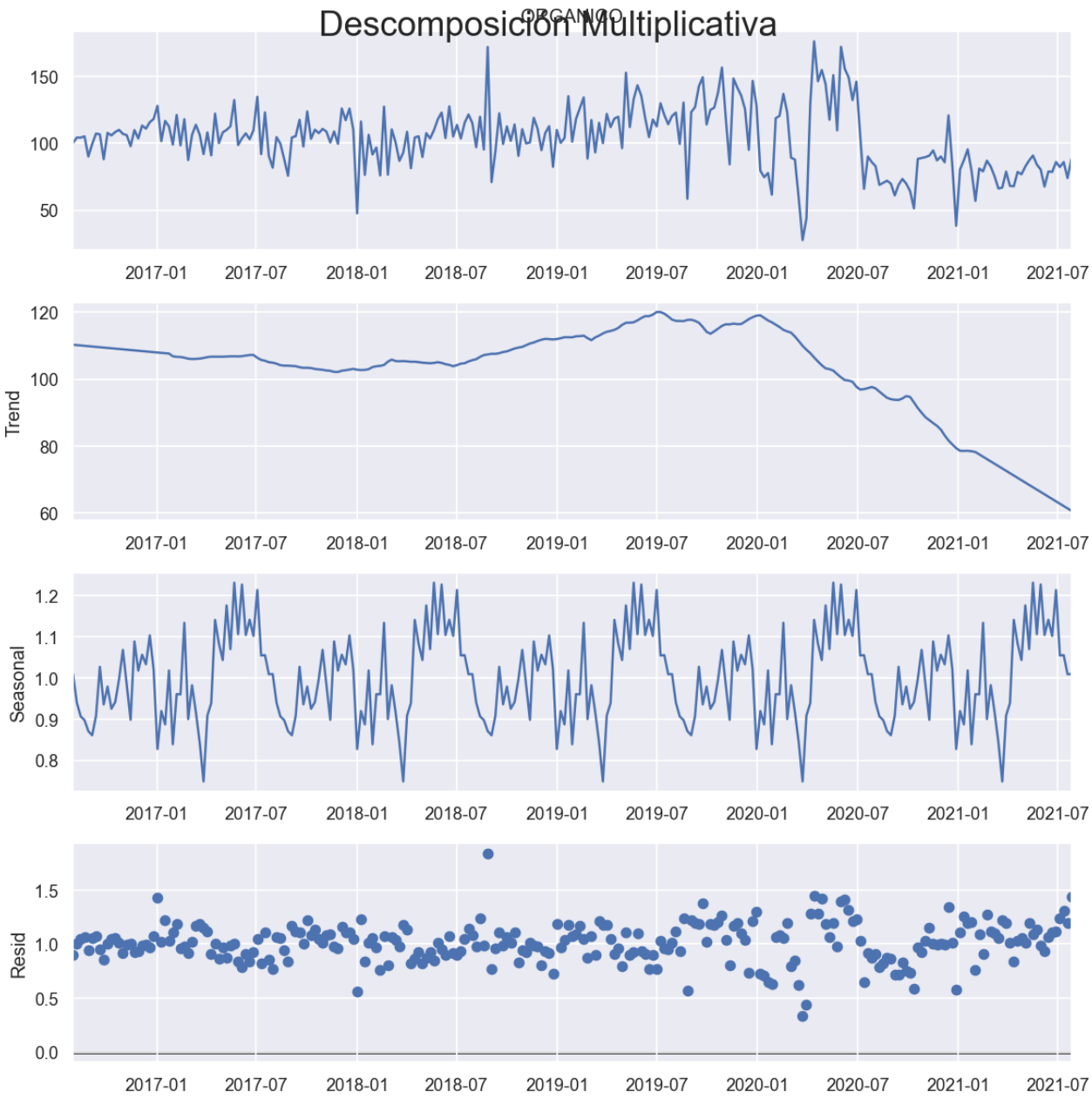


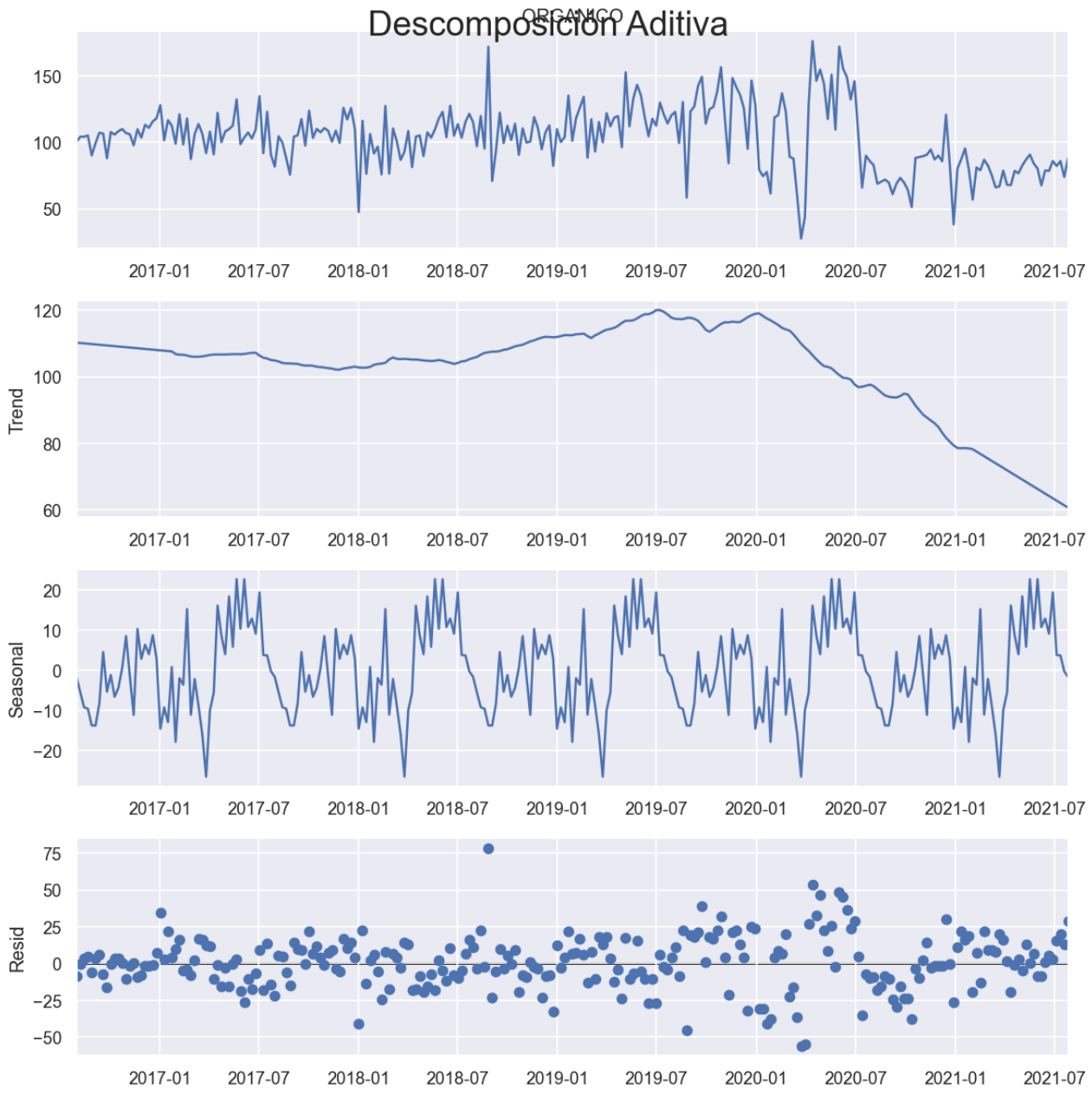


Additive and multiplicative time series

In [35]:

```
# Multiplicative Decomposition
result_mul = seasonal_decompose(df2['ORGANICO'], model='multiplicative', extrapolate_tr

# Additive Decomposition
result_add = seasonal_decompose(df2['ORGANICO'], model='additive', extrapolate_trend='f

# Plot
plt.rcParams.update({'figure.figsize': (10,10)})
result_mul.plot().suptitle('Descomposición Multiplicativa', fontsize=22)
result_add.plot().suptitle('Descomposición Aditiva', fontsize=22)
plt.show()
```


Descomposición Multiplicativa
ORGANICO
Trend
Seasonal
Resid
2017-01
2017-07
2018-01
2018-07
2019-01
2019-07
2020-01
2020-07
2021-01
2021-07



In [36]:

```
# Extract the Components ----
# Actual Values = Product of (Seasonal * Trend * Resid)

df_reconstructed = pd.concat([result_mul.seasonal, result_mul.trend, result_mul.resid,
df_reconstructed.columns = ['seas', 'trend', 'resid', 'actual_values']
df_reconstructed.head()
```

Out[36]:

Fecha	seas	trend	resid	actual_values
2016-08-01	1.009241	110.262666	0.899699	100.120
2016-08-08	0.940121	110.157489	1.005925	104.175
2016-08-15	0.906516	110.052312	1.043962	104.150
2016-08-22	0.897257	109.947134	1.065475	105.110
2016-08-29	0.870562	109.841957	0.941916	90.070

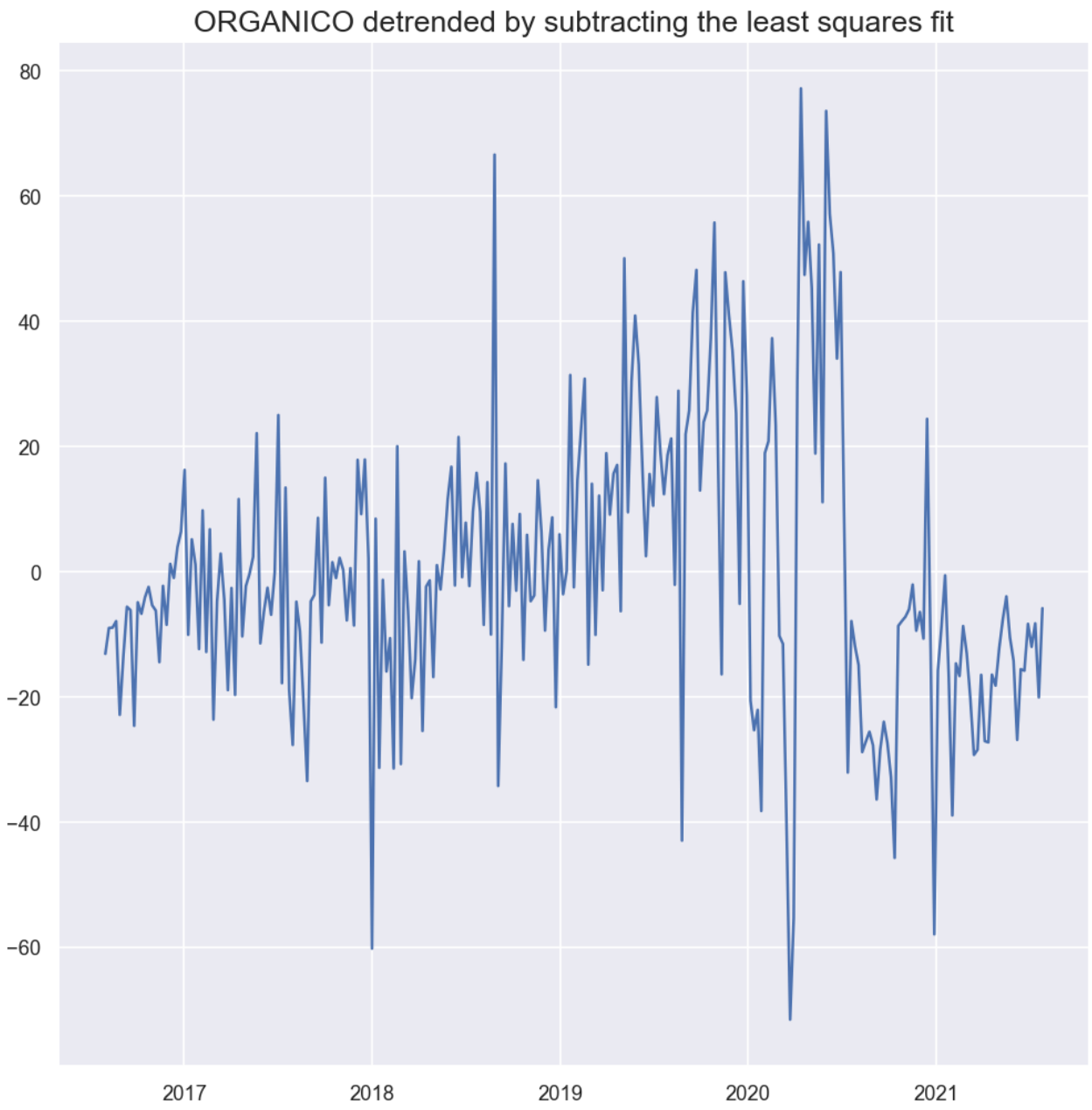
Stationarity

In [37]:
```
# ADF Test
result = adfuller(df2.ORGANICO.values, autolag='AIC')
print(f'ADF Statistic: {result[0]}')
print(f'p-value: {result[1]}')
for key, value in result[4].items():
    print('Critial Values:')
    print(f'   {key}, {value}')

# KPSS Test
result = kpss(df2.ORGANICO.values, regression='c')
print('\nKPSS Statistic: %f' % result[0])
print('p-value: %f' % result[1])
for key, value in result[3].items():
    print('Critial Values:')
```

```
ADF Statistic: -2.7006069703022986
p-value: 0.07393329554974265
Critial Values:
   1%, -3.457215237265747
Critial Values:
   5%, -2.873361841566324
Critial Values:
   10%, -2.5730700760129555

KPSS Statistic: 0.568307
p-value: 0.026282
Critial Values:
Critial Values:
Critial Values:
Critial Values:
```

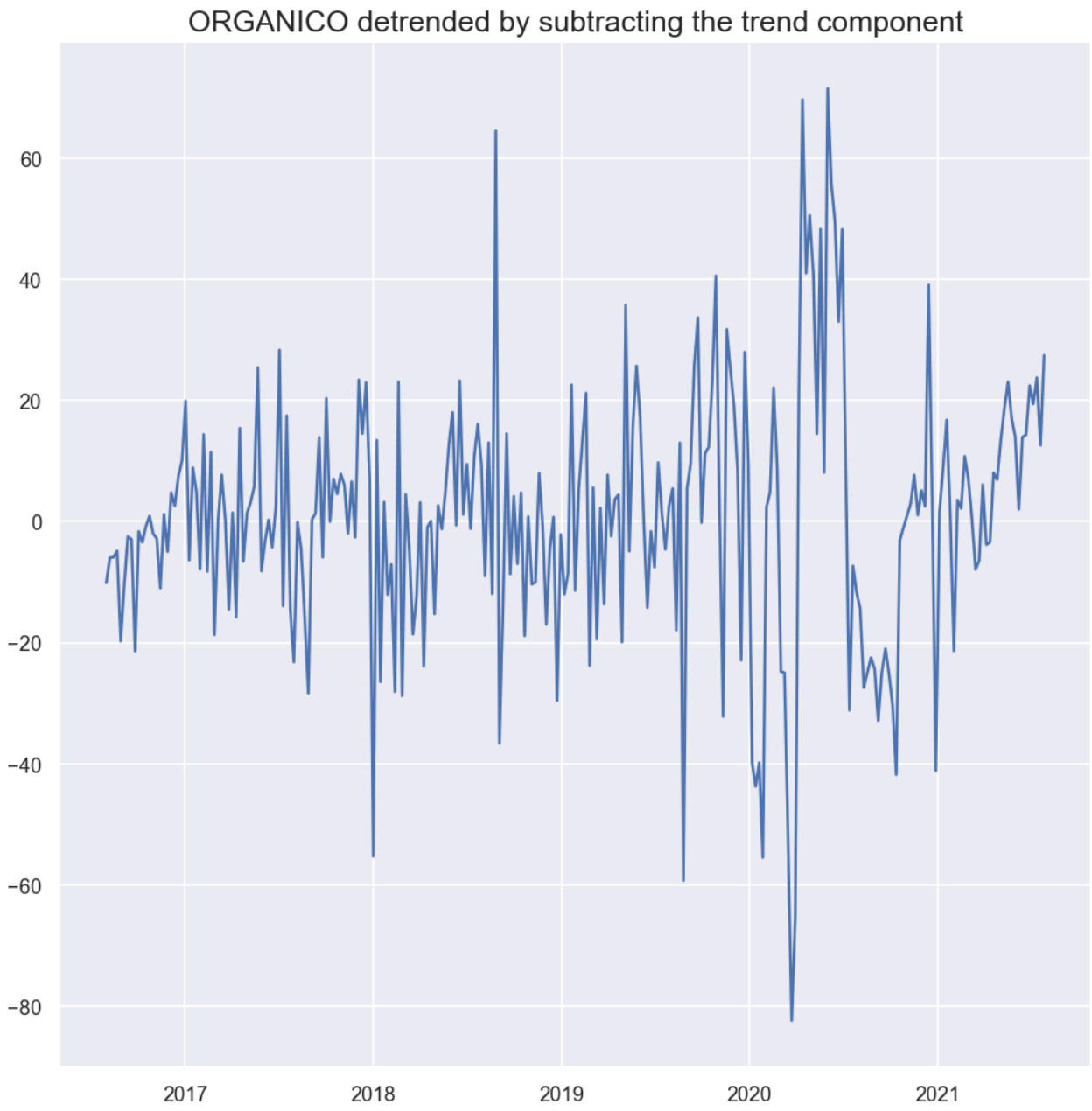
Detrending

In [38]:
```
detrended = signal.detrend(df2.ORGANICO.values)
detrended = pd.DataFrame(detrended, index=df2.index)
plt.plot(detrended)
plt.title('ORGANICO detrended by subtracting the least squares fit', fontsize=16)
plt.show()
```



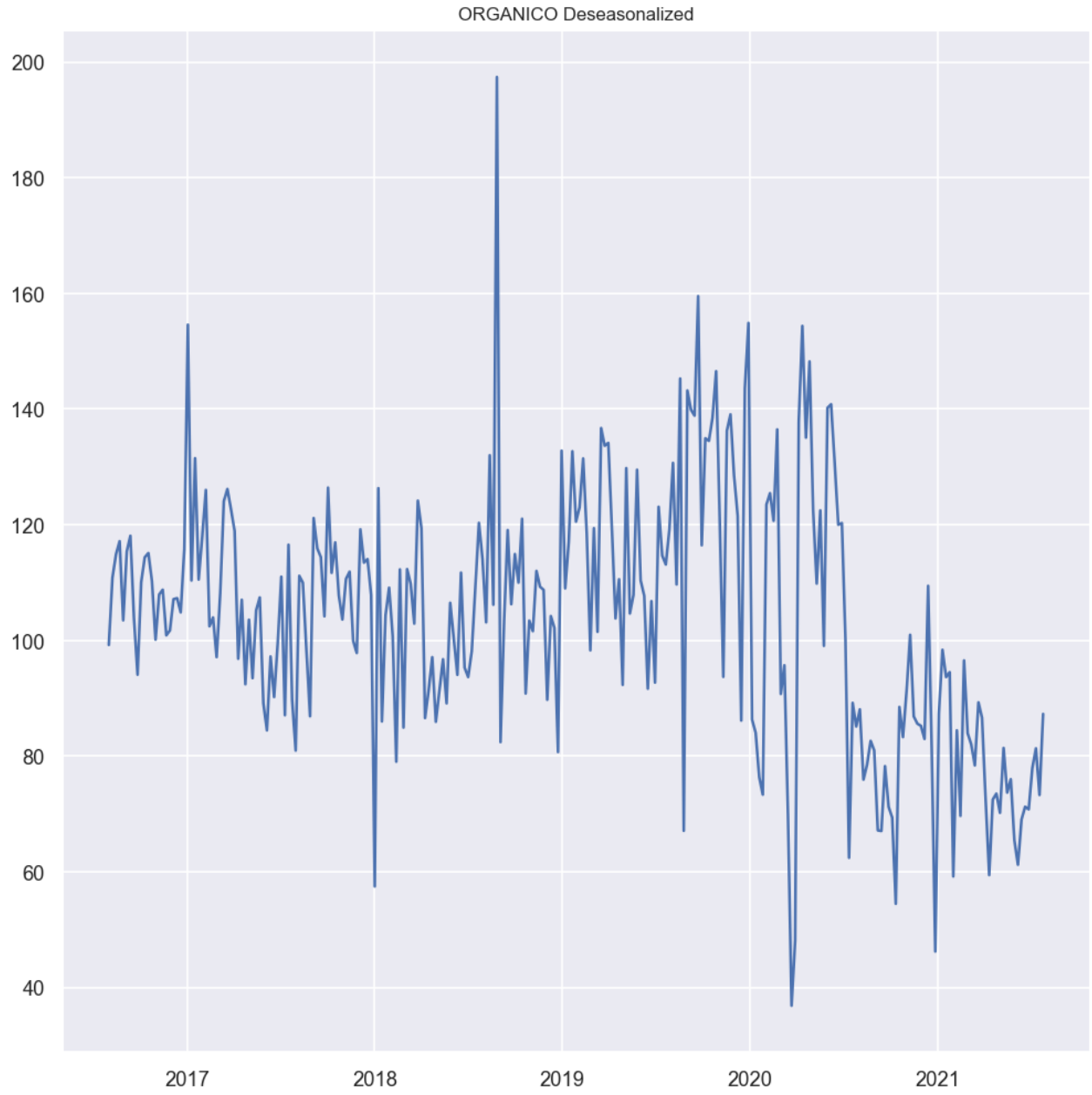
In [39]:

```
# Using statmodels: Subtracting the Trend Component.

result_mul = seasonal_decompose(df2.ORGANICO, model='multiplicative', extrapolate_trend

detrended = df2.ORGANICO.values - result_mul.trend

plt.plot(detrended)
plt.title('ORGANICO detrended by subtracting the trend component', fontsize=16)
plt.show()
```



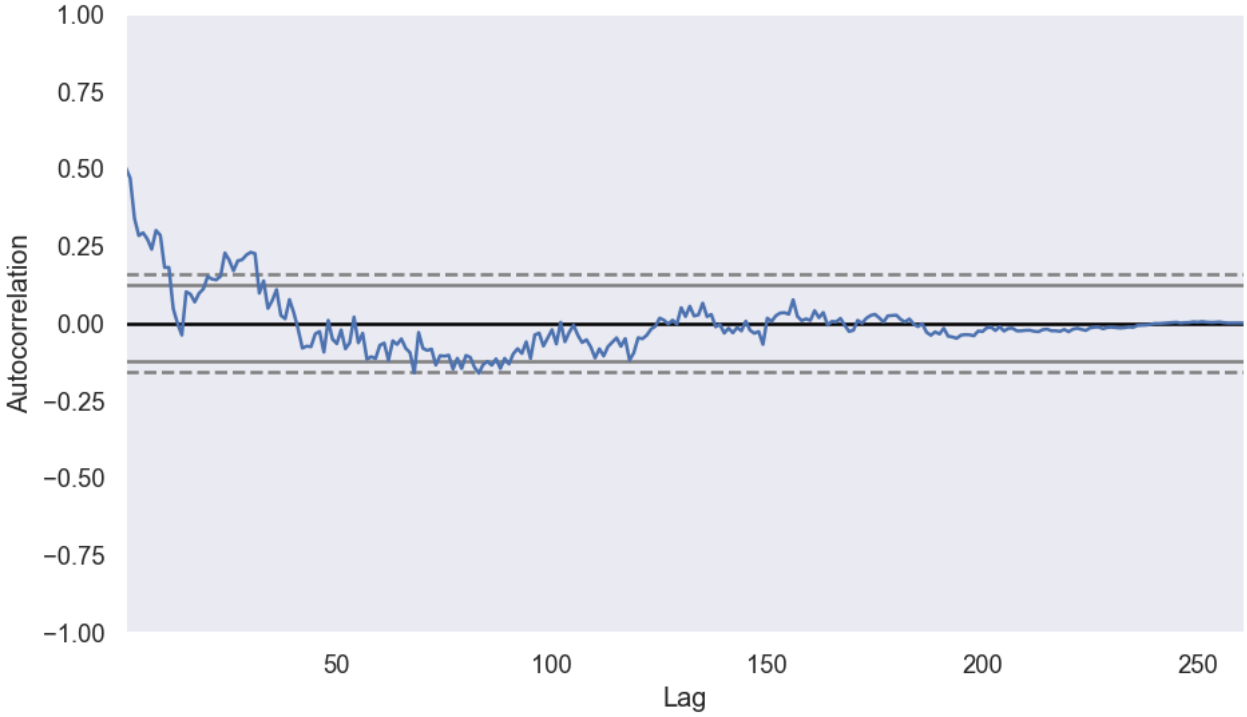
Deseasonalizing

In [40]:
```
# Subtracting the Trend Component.

# Time Series Decomposition
result_mul = seasonal_decompose(df2.ORGANICO, model='multiplicative', extrapolate_trend

# Deseasonalize
deseasonalized = df2.ORGANICO.values / result_mul.seasonal

# Plot
plt.plot(deseasonalized)
plt.title('ORGANICO Deseasonalized', fontsize=10)
plt.show()
```

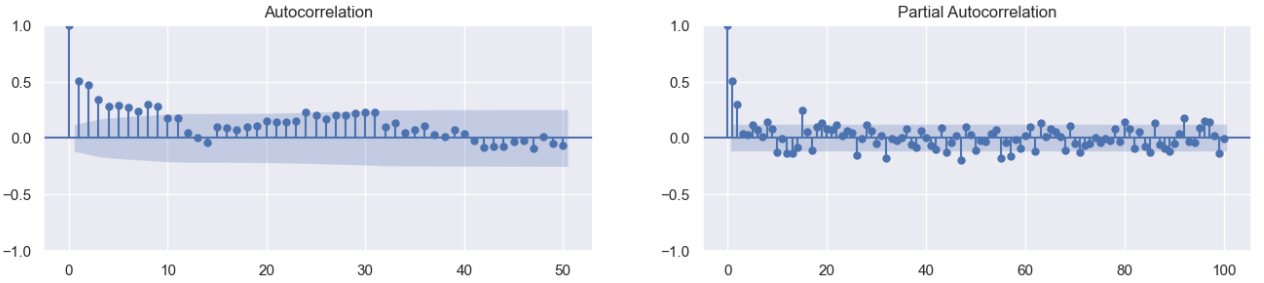


Test for Seasonality

In [41]:
```
# Draw Plot
plt.rcParams.update({'figure.figsize':(9,5), 'figure.dpi':120})
autocorrelation_plot(df2.ORGANICO.tolist())
plt.show()
```



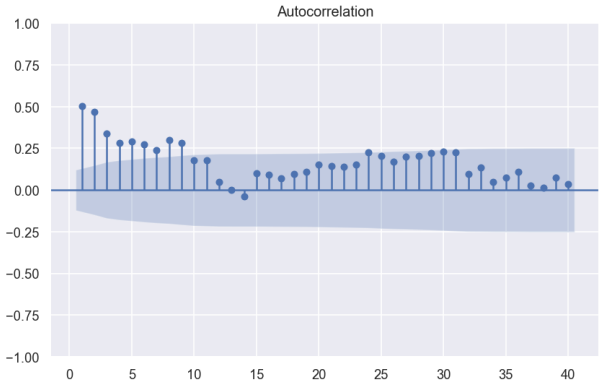
In [42]:
```
# Draw Plot
fig, axes = plt.subplots(1,2,figsize=(16,3), dpi= 100)
plot_acf(df2.ORGANICO.tolist(), lags=50, ax=axes[0])
plot_pacf(df2.ORGANICO.tolist(), lags=100, ax=axes[1])
plt.show()
```

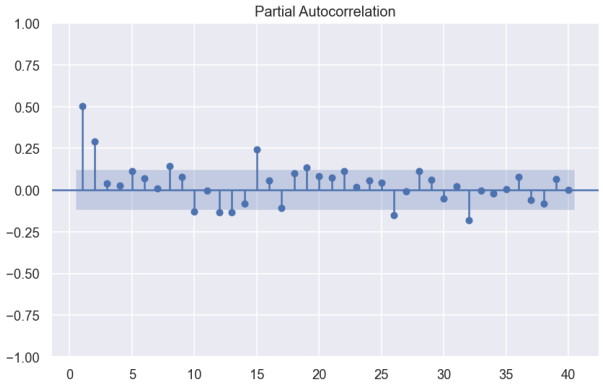


Original Series Stationarity Check

In [43]:
```
def plot_acf_pacf(series):
    plt.rcParams["figure.figsize"] = 18, 5

    fig, axes = plt.subplots(1, 2)

    sgt.plot_acf(series, zero = False, lags = 40, ax = axes[0])
    sgt.plot_pacf(series, zero = False, lags = 40, ax = axes[1])

    plt.show()
```

In [44]:
```
plot_acf_pacf(df2.ORGANICO)
adfuller(df2.ORGANICO)
```



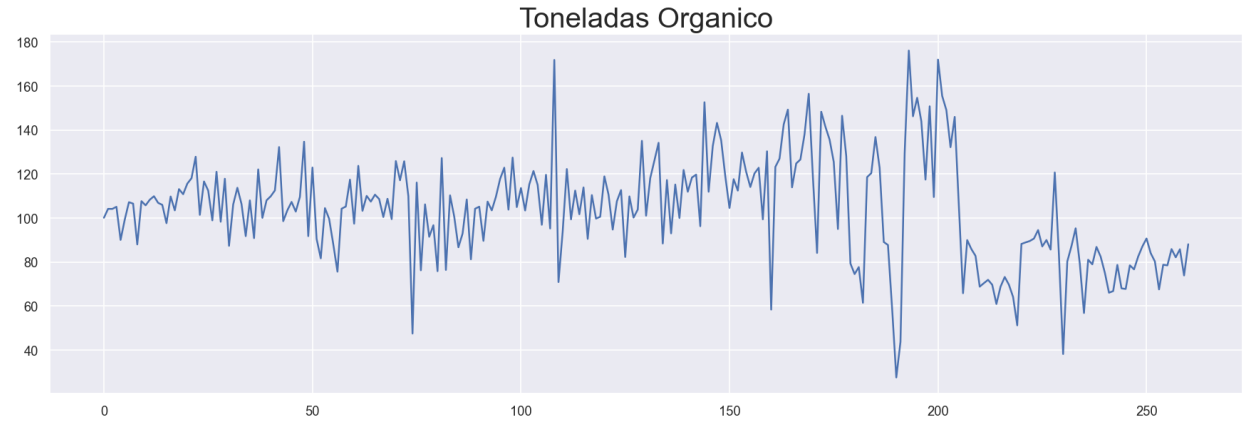


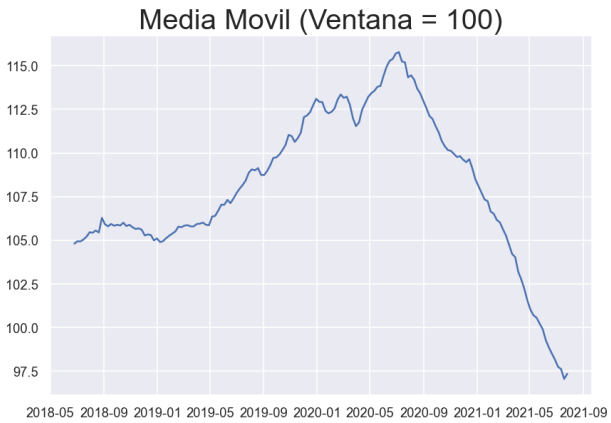
Out[44]:
```
(-2.7006069703022986,
 0.07393329554974265,
 14,
 246,
 {'1%': -3.457215237265747,
  '5%': -2.873361841566324,
  '10%': -2.5730700760129555},
 2153.601157321457)
```

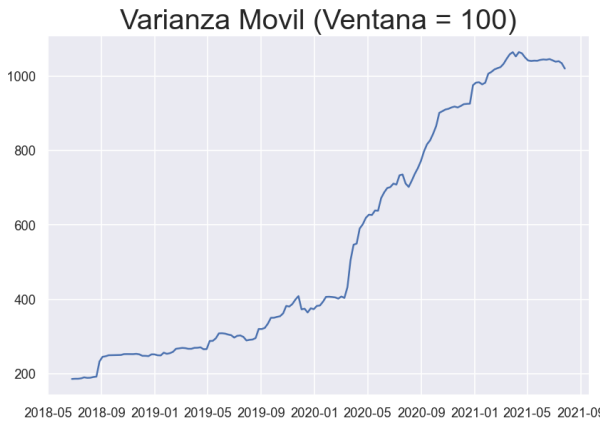
In [45]:
```
adfuller(df2.ORGANICO.tolist())
```

Out[45]:
```
(-2.7006069703022986,
 0.07393329554974265,
 14,
 246,
 {'1%': -3.457215237265747,
  '5%': -2.873361841566324,
  '10%': -2.5730700760129555},
 2153.601157321457)
```

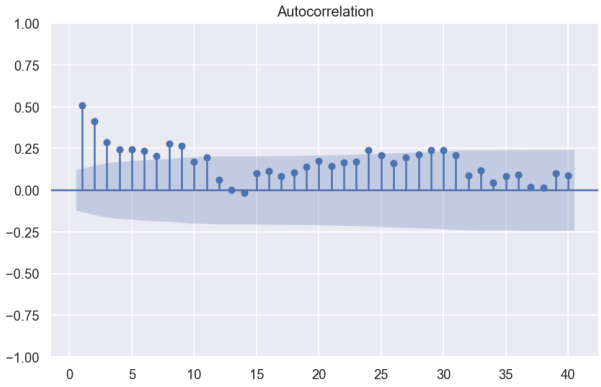
In [46]:
```
rolling_mean = df2.ORGANICO.rolling(100).mean()
rolling_var = df2.ORGANICO.rolling(100).var()

plt.rcParams["figure.figsize"] = 18, 12

plt.subplot(2, 1, 1)
plt.plot(df2.ORGANICO.tolist())
plt.title("Toneladas Organico", size = 24)

plt.subplot(2, 2, 3)
plt.plot(rolling_mean)
plt.title("Media Movil (Ventana = 100)", size = 24)

plt.subplot(2, 2, 4)
plt.plot(rolling_var)
plt.title("Varianza Movil (Ventana = 100)", size = 24)

plt.show()
```

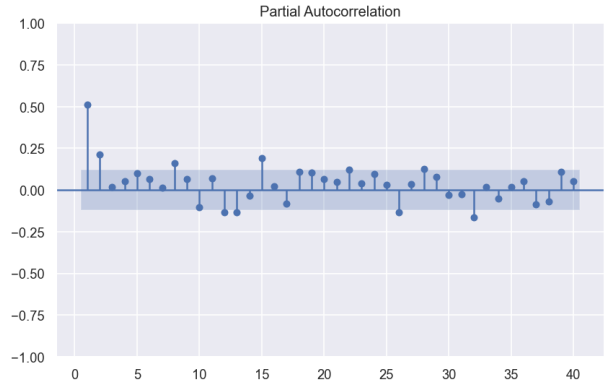






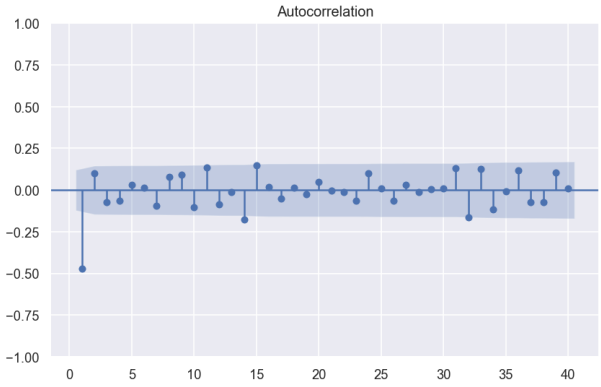
In [47]:
```
plot_acf_pacf(np.log(df2.ORGANICO.values))
adfuller(np.log(df2.ORGANICO.values))
```

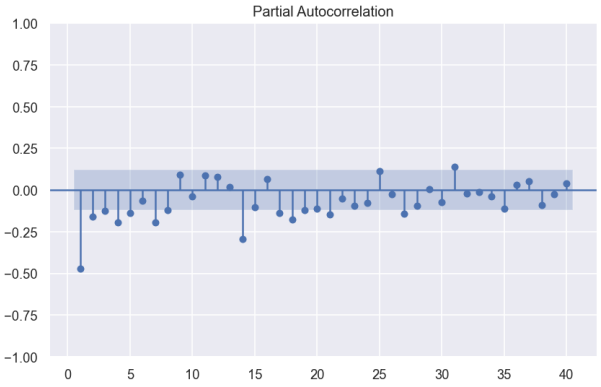




Out[47]:
```
(-2.621562599323426,
 0.0886221021970019,
 14,
 246,
 {'1%': -3.457215237265747,
  '5%': -2.873361841566324,
  '10%': -2.5730700760129555},
 -51.50101448546957)
```

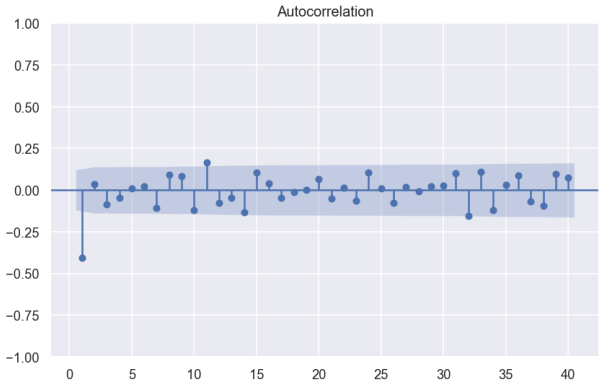
In [48]:
```
plot_acf_pacf(df2.ORGANICO.diff(1).dropna())
adfuller(df2.ORGANICO.diff(1).dropna())
```

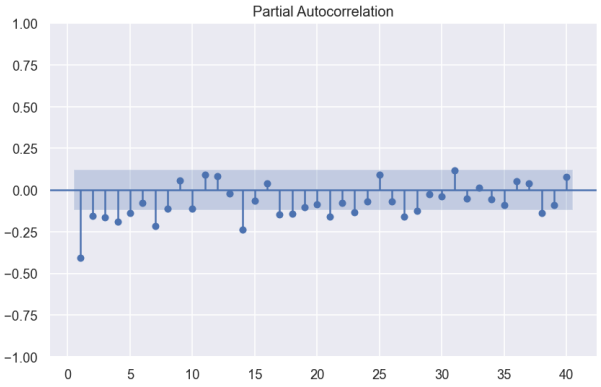




Out[48]:
```
(-5.541307172752409,
 1.6992236027611093e-06,
 16,
 243,
 {'1%': -3.4575505077947746,
  '5%': -2.8735087323013526,
  '10%': -2.573148434859185},
 2150.4184042730767)
```

In [49]:
```
df2["ORGANICO_log_diff"] = np.log(df2["ORGANICO"].values)
df2["ORGANICO_log_diff"] = df2["ORGANICO_log_diff"].diff(1)

plot_acf_pacf(df2["ORGANICO_log_diff"].dropna())
adfuller(df2["ORGANICO_log_diff"].dropna())
```



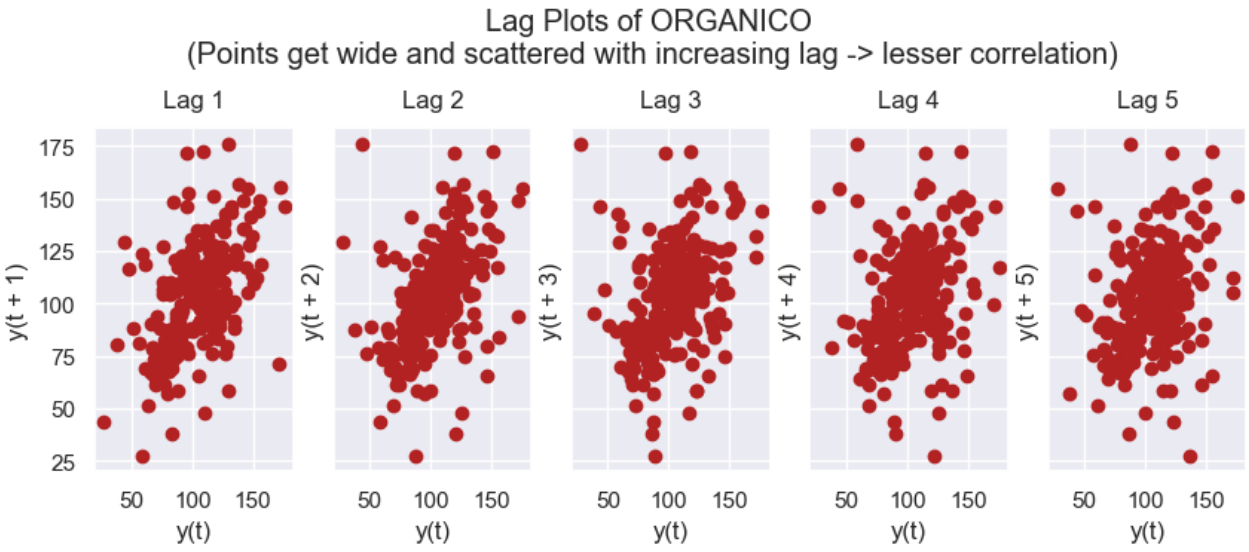


Out[49]:
```
(-6.019555327074041,
 1.5051248804497987e-07,
 13,
 246,
 {'1%': -3.457215237265747,
  '5%': -2.873361841566324,
  '10%': -2.5730700760129555},
 -45.01961251763538)
```

Lag Plots

In [50]:
```
plt.rcParams.update({'ytick.left' : False, 'axes.titlepad':10})

# Plot
fig, axes = plt.subplots(1, 5, figsize=(10,3), sharex=True, sharey=True, dpi=100)
for i, ax in enumerate(axes.flatten()[:5]):
    lag_plot(df2.ORGANICO, lag=i+1, ax=ax, c='firebrick')
    ax.set_title('Lag ' + str(i+1))
```

```
fig.suptitle('Lag Plots of ORGANICO \n(Points get wide and scattered with increasing la

plt.show()
```



Granger Causality test

In [51]:
```
grangercausalitytests(df2[['ORGANICO', 'Habitantes']], maxlag=4)
```

```
Granger Causality
number of lags (no zero) 1
ssr based F test:         F=5.5444  , p=0.0193  , df_denom=257, df_num=1
ssr based chi2 test:   chi2=5.6091  , p=0.0179  , df=1
likelihood ratio test: chi2=5.5494  , p=0.0185  , df=1
parameter F test:         F=5.5444  , p=0.0193  , df_denom=257, df_num=1

Granger Causality
number of lags (no zero) 2
ssr based F test:         F=1.9236  , p=0.1482  , df_denom=254, df_num=2
ssr based chi2 test:   chi2=3.9230  , p=0.1407  , df=2
likelihood ratio test: chi2=3.8935  , p=0.1427  , df=2
parameter F test:         F=1.9236  , p=0.1482  , df_denom=254, df_num=2

Granger Causality
number of lags (no zero) 3
ssr based F test:         F=1.7459  , p=0.1581  , df_denom=251, df_num=3
ssr based chi2 test:   chi2=5.3838  , p=0.1458  , df=3
likelihood ratio test: chi2=5.3284  , p=0.1493  , df=3
parameter F test:         F=1.7459  , p=0.1581  , df_denom=251, df_num=3

Granger Causality
number of lags (no zero) 4
ssr based F test:         F=1.6451  , p=0.1635  , df_denom=248, df_num=4
ssr based chi2 test:   chi2=6.8191  , p=0.1458  , df=4
likelihood ratio test: chi2=6.7302  , p=0.1509  , df=4
parameter F test:         F=1.6451  , p=0.1635  , df_denom=248, df_num=4
```

Out[51]:
```
{1: ({'ssr_ftest': (5.544352637694361, 0.019293105154350253, 257.0, 1),
   'ssr_chi2test': (5.609072707395074, 0.01786771596259989, 1),
   'lrtest': (5.549425776238422, 0.018486750848405554, 1),
   'params_ftest': (5.544352637694651, 0.019293105154346984, 257.0, 1.0)},
```

```
 [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2a9a67ec430>,
  <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2a9a680c520>,
  array([[0., 1., 0.]])]),
2: ({'ssr_ftest': (1.9236103333015884, 0.14820071122796996, 254.0, 2),
  'ssr_chi2test': (3.922953356890641, 0.14065057182793608, 2),
  'lrtest': (3.8935404067624404, 0.14273433077354405, 2),
  'params_ftest': (1.923610333295862, 0.14820071122880363, 254.0, 2.0)},
 [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2a9a680c4c0>,
  <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2a9a67fb9a0>,
  array([[0., 0., 1., 0., 0.],
         [0., 0., 0., 1., 0.]])]),
3: ({'ssr_ftest': (1.7458999459774451, 0.15812416471544063, 251.0, 3),
  'ssr_chi2test': (5.383771148153557, 0.14575804151718558, 3),
  'lrtest': (5.328368100367243, 0.14927218350219876, 3),
  'params_ftest': (1.7458999459680342, 0.1581241647173165, 251.0, 3.0)},
 [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2a9a50f80d0>,
  <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2a9a67fbe50>,
  array([[0., 0., 0., 1., 0., 0., 0.],
         [0., 0., 0., 0., 1., 0., 0.],
         [0., 0., 0., 0., 0., 1., 0.]])]),
4: ({'ssr_ftest': (1.6450772450896198, 0.16349693169423068, 248.0, 4),
  'ssr_chi2test': (6.819110515936005, 0.14576181083762962, 4),
  'lrtest': (6.730212162558928, 0.15085086343656667, 4),
  'params_ftest': (1.6450772450855524, 0.16349693169522408, 248.0, 4.0)},
 [<statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2a9a5f6fc70>,
  <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x2a9a5f6ffa0>,
  array([[0., 0., 0., 0., 1., 0., 0., 0., 0.],
         [0., 0., 0., 0., 0., 1., 0., 0., 0.],
         [0., 0., 0., 0., 0., 0., 1., 0., 0.],
         [0., 0., 0., 0., 0., 0., 0., 1., 0.]])])}
```

MODELADO

ARMA

Non Seasonal Model

In this model, the **ARMA(1, 1)** Model is build using the **spx_log_diff_seas** series.

In [52]:
```
scaled = pd.DataFrame(scaler.fit_transform(df2.drop('Mes', axis=1)), index=df2.index)
scaled.columns =['Año', 'Semana', 'INORGANICO', 'ORGANICO', 'RECICLAJE', 'VEGETAL', 'Ha
scaled
```

Out[52]:

Fecha	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Habitantes	COVID	ORGANICO
2016-08-01	0.0	0.576923	0.111035	0.488454	0.000000	0.000000	0.000000	0.0	
2016-08-08	0.0	0.596154	0.133391	0.515754	0.000000	0.000000	0.003747	0.0	
2016-08-15	0.0	0.615385	0.132438	0.515586	0.000000	0.000000	0.007494	0.0	

Fecha	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Habitantes	COVID	ORGANICO
2016-08-22	0.0	0.634615	0.202553	0.522049	0.000000	0.000000	0.011241	0.0	
2016-08-29	0.0	0.653846	0.237726	0.420790	0.000000	0.000000	0.014989	0.0	
...	...	...	...	...	...	...	...	...	
2021-06-28	1.0	0.480769	0.644746	0.392648	0.185056	0.072848	0.984197	0.0	
2021-07-05	1.0	0.500000	0.633723	0.367535	0.298094	0.144592	0.988107	0.0	
2021-07-12	1.0	0.519231	0.551094	0.392177	0.320385	0.176876	0.992180	0.0	
2021-07-19	1.0	0.538462	0.646527	0.312058	0.301392	0.153698	0.996090	0.0	
2021-07-26	1.0	0.557692	0.515788	0.407460	0.282002	0.071578	1.000000	0.0	

261 rows × 9 columns

In [53]:
```
train_df = df2.loc[:df2.index[int(len(df2)*0.7)]]
test_df = df2.loc[df2.index[int(len(df2)*0.7)+1]:]
```

In [54]:
```
model = pm.auto_arima(scaled['ORGANICO_log_diff'].dropna(),
                      trace = True,
                      suppress_warnings = True,
                      seasonal = True, m=12)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(1,0,1)[12] intercept   : AIC=inf, Time=0.91 sec
 ARIMA(0,0,0)(0,0,0)[12] intercept   : AIC=-332.783, Time=0.08 sec
 ARIMA(1,0,0)(1,0,0)[12] intercept   : AIC=-376.090, Time=0.53 sec
 ARIMA(0,0,1)(0,0,1)[12] intercept   : AIC=-402.192, Time=0.23 sec
 ARIMA(0,0,0)(0,0,0)[12]             : AIC=343.425, Time=0.04 sec
 ARIMA(0,0,1)(0,0,0)[12] intercept   : AIC=-400.799, Time=0.19 sec
 ARIMA(0,0,1)(1,0,1)[12] intercept   : AIC=inf, Time=0.62 sec
 ARIMA(0,0,1)(0,0,2)[12] intercept   : AIC=-403.491, Time=0.64 sec
 ARIMA(0,0,1)(1,0,2)[12] intercept   : AIC=inf, Time=1.28 sec
 ARIMA(0,0,0)(0,0,2)[12] intercept   : AIC=-332.724, Time=0.32 sec
 ARIMA(1,0,1)(0,0,2)[12] intercept   : AIC=-416.776, Time=1.09 sec
 ARIMA(1,0,1)(0,0,1)[12] intercept   : AIC=-414.625, Time=0.73 sec
 ARIMA(1,0,1)(1,0,2)[12] intercept   : AIC=inf, Time=1.26 sec
 ARIMA(1,0,1)(1,0,1)[12] intercept   : AIC=inf, Time=0.72 sec
 ARIMA(1,0,0)(0,0,2)[12] intercept   : AIC=-377.444, Time=0.55 sec
 ARIMA(2,0,1)(0,0,2)[12] intercept   : AIC=-406.422, Time=0.96 sec
 ARIMA(1,0,2)(0,0,2)[12] intercept   : AIC=-399.839, Time=1.03 sec
 ARIMA(0,0,2)(0,0,2)[12] intercept   : AIC=-411.809, Time=0.65 sec
 ARIMA(2,0,0)(0,0,2)[12] intercept   : AIC=-382.495, Time=0.61 sec
 ARIMA(2,0,2)(0,0,2)[12] intercept   : AIC=inf, Time=1.65 sec
```

```
ARIMA(1,0,1)(0,0,2)[12]             : AIC=inf, Time=0.75 sec

Best model:  ARIMA(1,0,1)(0,0,2)[12] intercept
Total fit time: 14.866 seconds
```

In [55]:
```
model.summary()
```

Out[55]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	260
Model:	SARIMAX(1, 0, 1)x(0, 0, [1, 2], 12)	**Log Likelihood**	214.388
Date:	Sat, 08 Oct 2022	**AIC**	-416.776
Time:	13:14:38	**BIC**	-395.412
Sample:	08-08-2016	**HQIC**	-408.187
	- 07-26-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.2996	0.022	13.559	0.000	0.256	0.343
ar.L1	0.3322	0.049	6.726	0.000	0.235	0.429
ma.L1	-0.9346	0.024	-39.530	0.000	-0.981	-0.888
ma.S.L12	-0.1410	0.071	-1.985	0.047	-0.280	-0.002
ma.S.L24	0.1310	0.056	2.349	0.019	0.022	0.240
sigma2	0.0108	0.001	18.754	0.000	0.010	0.012

Ljung-Box (L1) (Q):	0.51	**Jarque-Bera (JB):**	297.52
Prob(Q):	0.47	**Prob(JB):**	0.00
Heteroskedasticity (H):	2.54	**Skew:**	-1.20
Prob(H) (two-sided):	0.00	**Kurtosis:**	7.66

Warnings:

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

In [56]:
```
model = SARIMAX(train_df["ORGANICO_log_diff"].dropna(), order = (1, 0, 1), seasonal_ord
model_fit = model.fit()
model_fit.summary()
```

Out[56]:

SARIMAX Results

Dep. Variable:	ORGANICO_log_diff	**No. Observations:**	182
Model:	SARIMAX(1, 0, 1)x(0, 0, [1, 2], 12)	**Log Likelihood**	64.591
Date:	Sat, 08 Oct 2022	**AIC**	-119.181

Time:	13:14:39	**BIC**	-103.161
Sample:	08-08-2016	**HQIC**	-112.687
	- 01-27-2020		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.2101	0.085	-2.460	0.014	-0.378	-0.043
ma.L1	-0.7402	0.056	-13.234	0.000	-0.850	-0.631
ma.S.L12	-0.1195	0.103	-1.161	0.246	-0.321	0.082
ma.S.L24	0.0985	0.086	1.141	0.254	-0.071	0.268
sigma2	0.0286	0.002	17.640	0.000	0.025	0.032

Ljung-Box (L1) (Q):	0.01	**Jarque-Bera (JB):**	255.23
Prob(Q):	0.93	**Prob(JB):**	0.00
Heteroskedasticity (H):	3.31	**Skew:**	-1.47
Prob(H) (two-sided):	0.00	**Kurtosis:**	8.00

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

Model Predictions

Entrena una vez - Predice una vez

In [57]:

```
# Building a predictions dataframe for this technique
pred1_df = pd.DataFrame(columns = ["ORGANICO", "ORGANICO_1",
                                   "model_preds", "model_preds_lower", "model_preds_upp
                                   "model_preds_exp", "model_preds_lower_exp", "model_p
                                   "ORGANICO_preds", "ORGANICO_preds_lower", "ORGANICO_
                        index = df2.index)

# Storing the original series and one lagged version (y(t) and y(t-1))
pred1_df["ORGANICO"] = df2["ORGANICO"]
pred1_df["ORGANICO_1"] = pred1_df["ORGANICO"].shift(1)

# Predictions on transformed data over the full span of the dataset.
pred1_df["model_preds"] = model_fit.predict(start = train_df.index[1], end = test_df.in

# Getting Confidence Intervals for the transformed predictions on test set
forecast = model_fit.get_forecast(len(test_df.index))
forecast_df = forecast.conf_int(alpha = 0.05) # Confidence level of 95%

for num in test_df.index:
    pred1_df.at[num, "model_preds_lower"] = forecast_df["lower ORGANICO_log_diff"].loc[
    pred1_df.at[num, "model_preds_upper"] = forecast_df["upper ORGANICO_log_diff"].loc[
```

```
# Inverting the log transformation by using exponent
pred1_df["model_preds_exp"] = np.exp(pred1_df["model_preds"].values)
pred1_df["model_preds_lower_exp"] = np.exp(list(pred1_df["model_preds_lower"].values))
pred1_df["model_preds_upper_exp"] = np.exp(list(pred1_df["model_preds_upper"].values))

# Building the In Sample predictions using the formula shown in (*)
for num in train_df.index:
    pred1_df.at[num, "ORGANICO_preds"] = pred1_df.loc[num]["model_preds_exp"] * pred1_d

# The for loop below is used to get out of sample predictions where y(t-1) are derived
pred1_exp_list = list(pred1_df.loc[test_df.index]["model_preds_exp"].values)
y_t_1 = pred1_df.at[train_df.index[-1], "ORGANICO_preds"]
for idx, pred_exp in enumerate(pred1_exp_list):
    pred1_df.at[test_df.index[idx], "ORGANICO_preds"] = pred1_df.at[test_df.index[idx],
    y_t_1 = pred1_df.at[test_df.index[idx], "ORGANICO_preds"]

# Inverting the Transformation on the confidence intervals using the formula in (*)
for num in test_df.index:
    pred1_df.at[num, "ORGANICO_preds_lower"] = pred1_df.loc[num]["model_preds_lower_exp
    pred1_df.at[num, "ORGANICO_preds_upper"] = pred1_df.loc[num]["model_preds_upper_exp
```

In [58]:
```
pred1_df.head()
```

Out[58]:

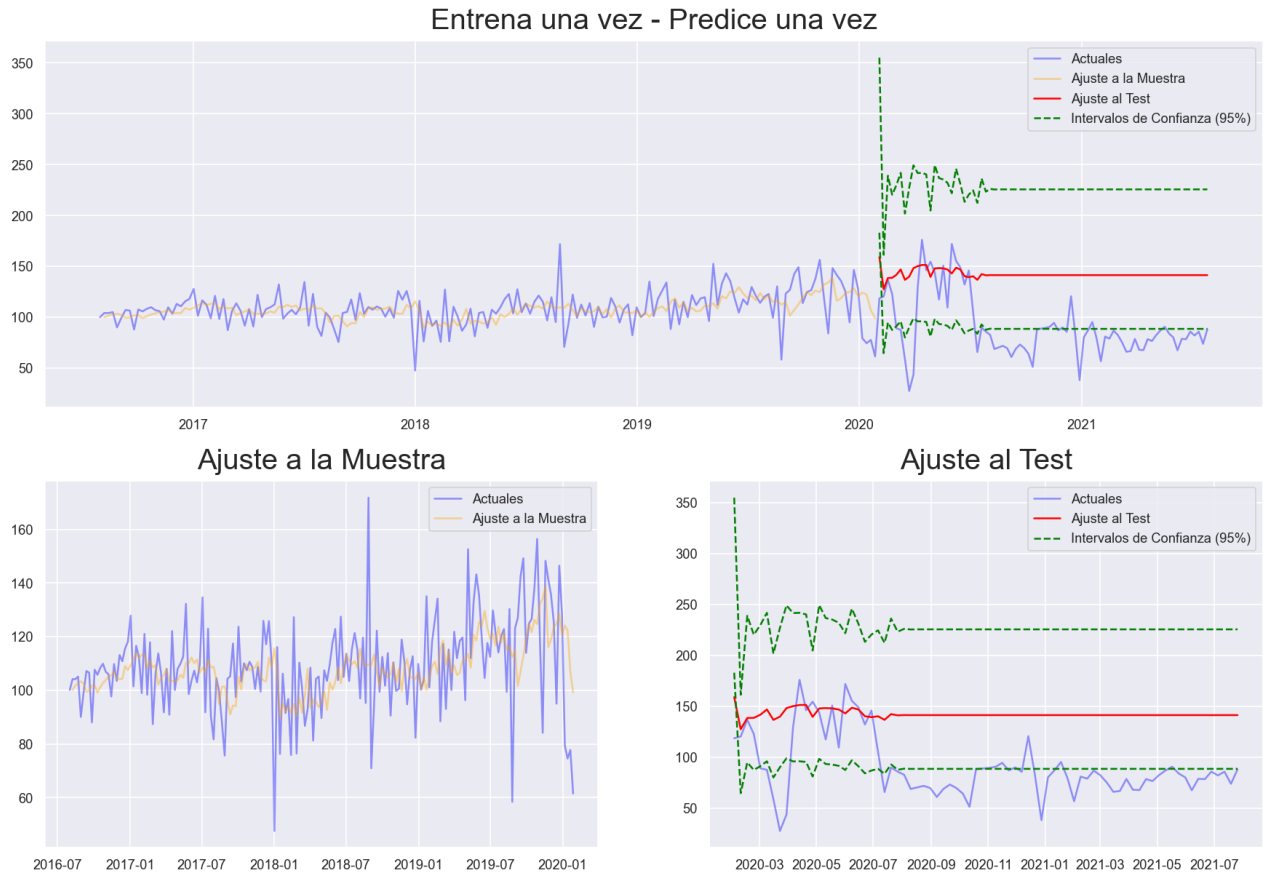
Fecha	ORGANICO	ORGANICO_1	model_preds	model_preds_lower	model_preds_upper	model_preds_ex
2016-08-01	100.120	NaN	NaN	NaN	NaN	Na
2016-08-08	104.175	100.120	0.000000	NaN	NaN	1.00000
2016-08-15	104.150	104.175	-0.023453	NaN	NaN	0.97682
2016-08-22	105.110	104.150	-0.013520	NaN	NaN	0.98657
2016-08-29	90.070	105.110	-0.016991	NaN	NaN	0.98315

Evaluating Model Performance

- **In Sample and Out of Sample Fit**

In [137…
```
plt.rcParams["figure.figsize"] = 18, 12

plt.subplot(2, 1, 1)
plt.plot(pred1_df["ORGANICO"], color = "blue", label = "Actuales", alpha = 0.4)
plt.plot(pred1_df.loc[train_df.index]["ORGANICO_preds"], color = "orange", alpha = 0.4,
plt.plot(pred1_df.loc[test_df.index]["ORGANICO_preds"], color = "red", linestyle = "-",
plt.plot(pred1_df.loc[test_df.index]["ORGANICO_preds_lower"], color = "green", linestyl
plt.plot(pred1_df.loc[test_df.index]["ORGANICO_preds_upper"], color = "green", linestyl
plt.title("Entrena una vez - Predice una vez", size = 24)
```

```
plt.legend()

plt.subplot(2, 2, 3)
plt.plot(pred1_df.loc[train_df.index]["ORGANICO"], color = "blue", label = "Actuales",
plt.plot(pred1_df.loc[train_df.index]["ORGANICO_preds"], color = "orange", alpha = 0.4,
plt.title("Ajuste a la Muestra", size = 24)
plt.legend()

plt.subplot(2, 2, 4)
plt.plot(pred1_df.loc[test_df.index]["ORGANICO"], color = "blue", label = "Actuales", a
plt.plot(pred1_df.loc[test_df.index]["ORGANICO_preds"], color = "red", linestyle = "-",
plt.plot(pred1_df.loc[test_df.index]["ORGANICO_preds_lower"], color = "green", linestyl
plt.plot(pred1_df.loc[test_df.index]["ORGANICO_preds_upper"], color = "green", linestyl
plt.title("Ajuste al Test", size = 24)
plt.legend()

plt.show()
```



- **Root Mean Squared Error**

In [60]:

```
rmse_in = np.sqrt(mean_squared_error(y_true = pred1_df.loc[train_df.index[1]:train_df.i
                                     y_pred = pred1_df.loc[train_df.index[1]:train_df.i

rmse_out = np.sqrt(mean_squared_error(y_true = pred1_df.loc[test_df.index]["ORGANICO"],
                                      y_pred = pred1_df.loc[test_df.index]["ORGANICO_pr

print(f"RMSE for In Sample Fit - {rmse_in}")
print(f"RMSE for Out of Sample Fit - {rmse_out}")
```

```
RMSE for In Sample Fit - 17.177891628667076
```

```
RMSE for Out of Sample Fit - 58.70998532596392
```

Entrena una vez - Predicciones continuas

In [61]:
```
# Building a predictions dataframe for this technique
pred2_df = pd.DataFrame(columns = ["ORGANICO", "ORGANICO_1",
                                   "model_preds", "model_preds_lower", "model_preds_upp
                                   "model_preds_exp", "model_preds_lower_exp", "model_p
                                   "ORGANICO_preds", "ORGANICO_preds_lower", "ORGANICO_
                        index = df2.index)

# Storing the original series and one lagged version (y(t) and y(t-1))
pred2_df["ORGANICO"] = df2["ORGANICO"]
pred2_df["ORGANICO_1"] = pred2_df["ORGANICO"].shift(1)

# Predictions on transformed data over the full span of the dataset.
pred2_df["model_preds"] = model_fit.predict(start = train_df.index[1], end = test_df.in

# Getting Confidence Intervals for the transformed predictions on test set
forecast = model_fit.get_forecast(len(test_df.index))
forecast_df = forecast.conf_int(alpha = 0.05) # Confidence level of 95%

for num in test_df.index:
    pred2_df.at[num, "model_preds_lower"] = forecast_df["lower ORGANICO_log_diff"].loc[
    pred2_df.at[num, "model_preds_upper"] = forecast_df["upper ORGANICO_log_diff"].loc[


# Building the In Sample and Out of Sample predictions using the formula shown in (*)
# Taking Exponent to invert logarithmic effect
pred2_df["model_preds_exp"] = np.exp(pred2_df["model_preds"].values)
pred2_df["model_preds_lower_exp"] = np.exp(list(pred2_df["model_preds_lower"].values))
pred2_df["model_preds_upper_exp"] = np.exp(list(pred2_df["model_preds_upper"].values))

# Multiplying with past lags to get the forecast and the confidence intervals
pred2_df["ORGANICO_preds"] = pred2_df["model_preds_exp"] * pred2_df["ORGANICO_1"]
for num in test_df.index:
    pred2_df.at[num, "ORGANICO_preds_lower"] = pred2_df.loc[num]["model_preds_lower_exp
    pred2_df.at[num, "ORGANICO_preds_upper"] = pred2_df.loc[num]["model_preds_upper_exp
```

In [62]:
```
pred2_df.head()
```

Out[62]:

Fecha	ORGANICO	ORGANICO_1	model_preds	model_preds_lower	model_preds_upper	model_preds_ex
2016-08-01	100.120	NaN	NaN	NaN	NaN	Na
2016-08-08	104.175	100.120	0.000000	NaN	NaN	1.00000
2016-08-15	104.150	104.175	-0.023453	NaN	NaN	0.97682
2016-08-22	105.110	104.150	-0.013520	NaN	NaN	0.98657

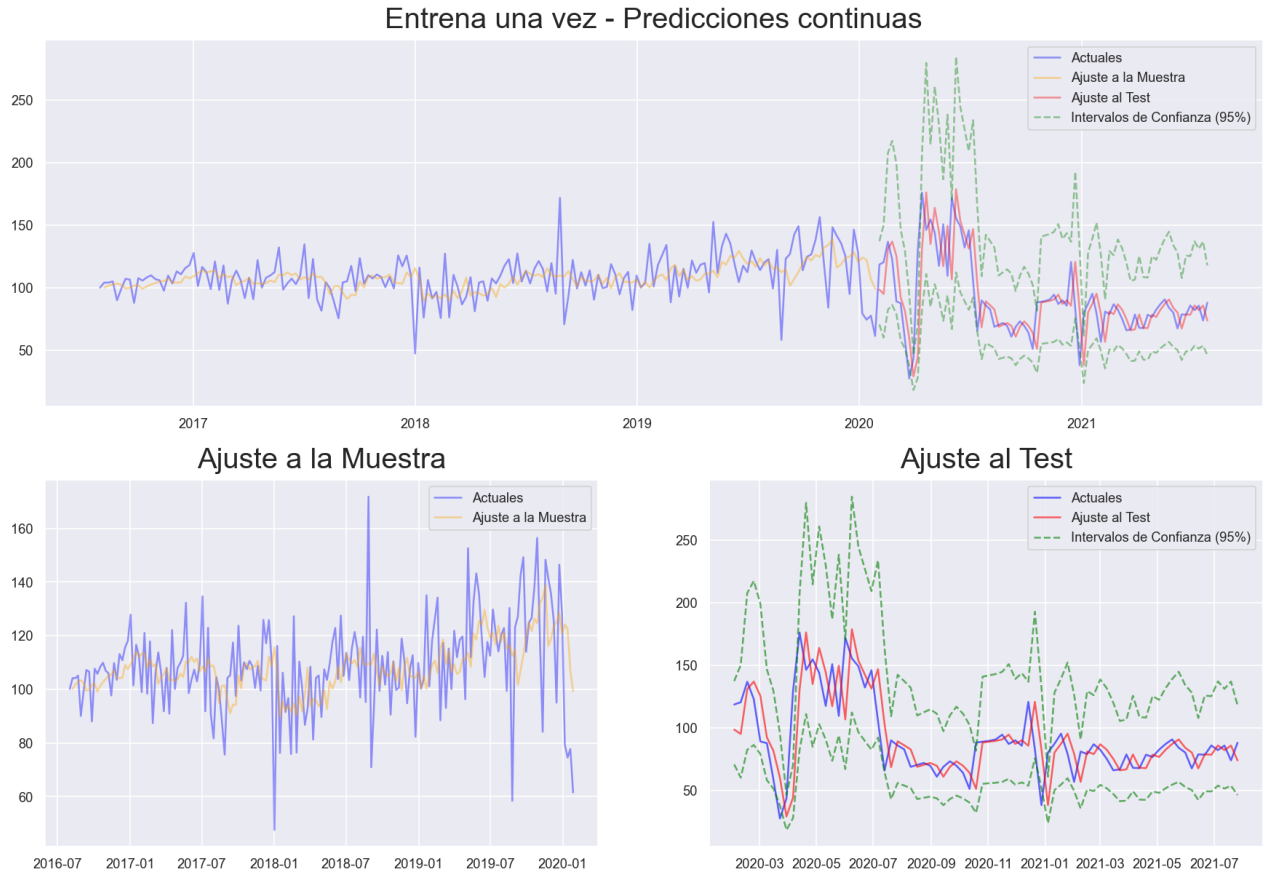
Fecha	ORGANICO	ORGANICO_1	model_preds	model_preds_lower	model_preds_upper	model_preds_ex
2016-08-29	90.070	105.110	-0.016991	NaN	NaN	0.98315

Evaluating Model Performance

- **In Sample and Out of Sample Fit**

In [138…

```
plt.rcParams["figure.figsize"] = 18, 12

plt.subplot(2, 1, 1)
plt.plot(pred2_df["ORGANICO"], color = "blue", label = "Actuales", alpha = 0.4)
plt.plot(pred2_df.loc[train_df.index]["ORGANICO_preds"], color = "orange", alpha = 0.4,
plt.plot(pred2_df.loc[test_df.index]["ORGANICO_preds"], color = "red", linestyle = "-",
plt.plot(pred2_df.loc[test_df.index]["ORGANICO_preds_lower"], color = "green", linestyl
plt.plot(pred2_df.loc[test_df.index]["ORGANICO_preds_upper"], color = "green", linestyl
plt.title("Entrena una vez - Predicciones continuas", size = 24)
plt.legend()

plt.subplot(2, 2, 3)
plt.plot(pred2_df.loc[train_df.index]["ORGANICO"], color = "blue", label = "Actuales",
plt.plot(pred2_df.loc[train_df.index]["ORGANICO_preds"], color = "orange", linestyle =
plt.title("Ajuste a la Muestra", size = 24)
plt.legend()

plt.subplot(2, 2, 4)
plt.plot(pred2_df.loc[test_df.index]["ORGANICO"], color = "blue", label = "Actuales", a
plt.plot(pred2_df.loc[test_df.index]["ORGANICO_preds"], color = "red", linestyle = "-",
plt.plot(pred2_df.loc[test_df.index]["ORGANICO_preds_lower"], color = "green", linestyl
plt.plot(pred2_df.loc[test_df.index]["ORGANICO_preds_upper"], color = "green", linestyl
plt.title("Ajuste al Test", size = 24)
plt.legend()

plt.show()
```



- **Root Mean Squared Error**

In [64]:
```
rmse_in = np.sqrt(mean_squared_error(y_true = pred2_df.loc[train_df.index[1]:train_df.i
                                     y_pred = pred2_df.loc[train_df.index[1]:train_df.i

rmse_out = np.sqrt(mean_squared_error(y_true = pred2_df.loc[test_df.index]["ORGANICO"],
                                      y_pred = pred2_df.loc[test_df.index]["ORGANICO_pr

print(f"RMSE for In Sample Fit - {rmse_in}")
print(f"RMSE for Out of Sample Fit - {rmse_out}")
```

```
RMSE for In Sample Fit - 17.177891628667076
RMSE for Out of Sample Fit - 22.341775739288945
```

Entrenamiento continuo - Predicciones continuas

In [65]:
```
def get_model(rolling_train_df, last_idx, next_i):
    train_series = rolling_train_df.loc[:last_idx]["ORGANICO"]
    train_series = np.log(train_series)
    train_series = train_series.diff(1)
    flag = 0
    while(flag == 0):
        flag = 0
        try:
            rolling_model = SARIMAX(train_series.dropna(), order = (1, 0, 1))
            rolling_model_results = rolling_model.fit()
            flag = 1
        except:
            flag = 0
```

```
    rolling_pred = rolling_model_results.predict(start = test_df.index[next_i], end = t

    rolling_forecast = rolling_model_results.get_forecast(1)
    rolling_forecast_df = rolling_forecast.conf_int(alpha = 0.05) # Confidence level of

    y_t_pred = (np.exp(rolling_pred.values[0]) * rolling_train_df.at[test_df.index[next
    y_t_pred_lower = (np.exp(rolling_forecast_df["lower ORGANICO"].values[0]) * rolling
    y_t_pred_upper = (np.exp(rolling_forecast_df["upper ORGANICO"].values[0]) * rolling

    return rolling_model_results, y_t_pred, y_t_pred_lower, y_t_pred_upper
```

In [66]:

```
# Building a predictions dataframe for this technique
pred3_df = pd.DataFrame(columns = ["ORGANICO", "ORGANICO_1",
                                   "model_preds", "model_preds_lower", "model_preds_upp
                                   "model_preds_exp", "model_preds_lower_exp", "model_p
                                   "ORGANICO_preds", "ORGANICO_preds_lower", "ORGANICO_
                        index = df2.index)

# Storing the original series and one lagged version (y(t) and y(t-1))
pred3_df["ORGANICO"] = df2["ORGANICO"]
pred3_df["ORGANICO_1"] = pred3_df["ORGANICO"].shift(1)

# Predictions on transformed data over the full training dataset and the first lag of t
pred3_df["model_preds"] = model_fit.predict(start = train_df.index[1], end = test_df.in

# Taking Exponent to invert logarithmic effect from predictions
pred3_df["model_preds_exp"] = np.exp(pred3_df["model_preds"].values)

# Building the In Sample Predictions
for num in train_df.index:
    pred3_df.at[num, "ORGANICO_preds"] = pred3_df.loc[num]["model_preds_exp"] * pred3_d

# The prediction for the first testing lag is built here
last_train_idx = train_df.index[-1]
rolling_model, y_t_pred, y_t_pred_lower, y_t_pred_upper = get_model(pred3_df, last_trai
pred3_df.at[test_df.index[0], "ORGANICO_preds"] = y_t_pred
pred3_df.at[test_df.index[0], "ORGANICO_preds_lower"] = y_t_pred_lower
pred3_df.at[test_df.index[0], "ORGANICO_preds_upper"] = y_t_pred_upper

# This loop is used to get new models for each new lag that is added to the training se
for idx in range(len(test_df.index)-1):
    last_train_idx = test_df.index[idx]
    rolling_model, y_t_pred, y_t_pred_lower, y_t_pred_upper = get_model(pred3_df, last_
    pred3_df.at[test_df.index[idx+1], "ORGANICO_preds"] = y_t_pred
    pred3_df.at[test_df.index[idx+1], "ORGANICO_preds_lower"] = y_t_pred_lower
    pred3_df.at[test_df.index[idx+1], "ORGANICO_preds_upper"] = y_t_pred_upper
```

In [67]:

```
for num in train_df.index:
    pred3_df.at[num, "ORGANICO_preds"] = pred3_df.loc[num]["model_preds_exp"] * pred3_d
```

In [68]:

```
pred3_df.head()
```

Out[68]:

	ORGANICO	ORGANICO_1	model_preds	model_preds_lower	model_preds_upper	model_preds_ex

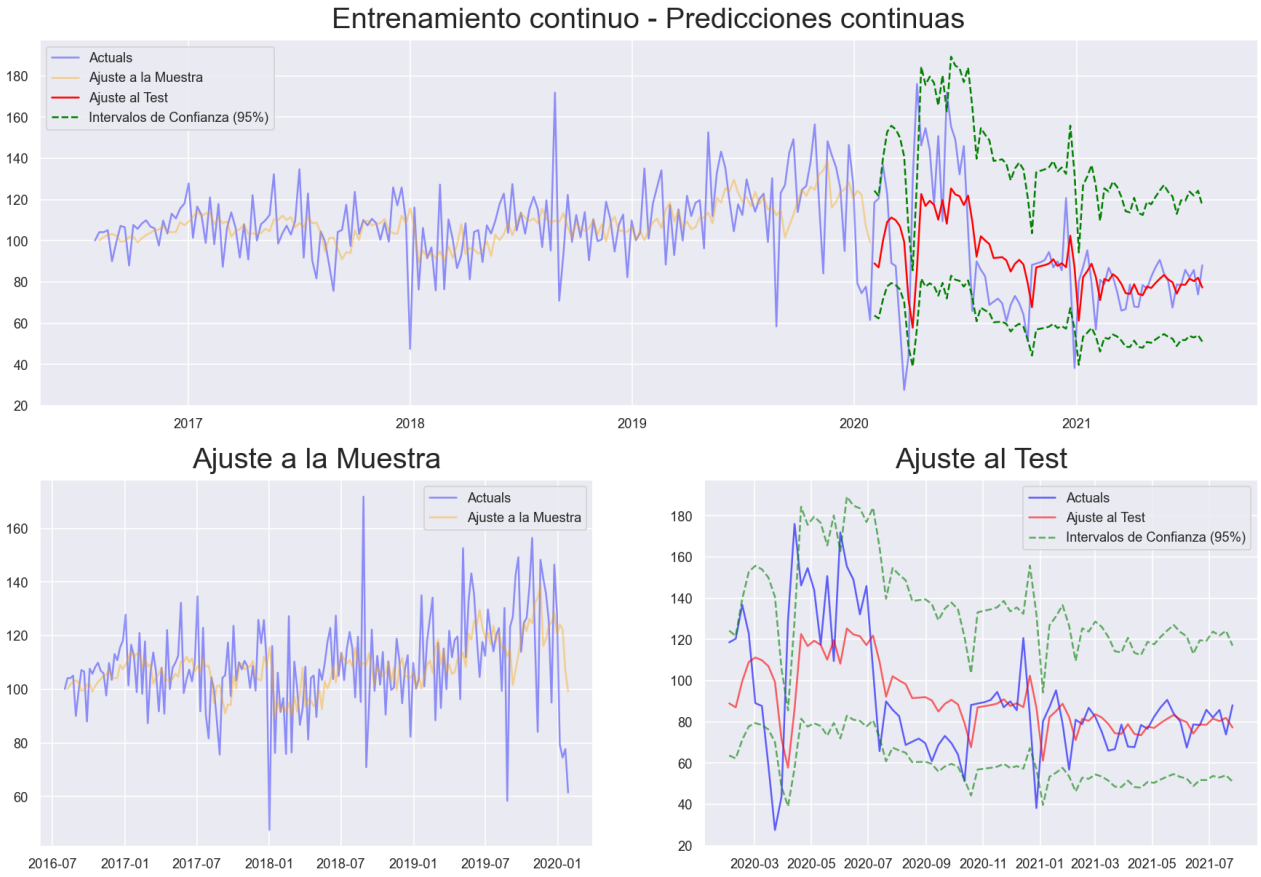
Fecha	ORGANICO	ORGANICO_1	model_preds	model_preds_lower	model_preds_upper	model_preds_ex
Fecha						
2016-08-01	100.120	NaN	NaN	NaN	NaN	Na
2016-08-08	104.175	100.120	0.000000	NaN	NaN	1.00000
2016-08-15	104.150	104.175	-0.023453	NaN	NaN	0.97682
2016-08-22	105.110	104.150	-0.013520	NaN	NaN	0.98657
2016-08-29	90.070	105.110	-0.016991	NaN	NaN	0.98315

Evaluating Model Performance

- **In Sample and Out of Sample Fit**

In [139…

```
plt.rcParams["figure.figsize"] = 18, 12

plt.subplot(2, 1, 1)
plt.plot(pred3_df["ORGANICO"], color = "blue", label = "Actuals", alpha = 0.4)
plt.plot(pred3_df.loc[train_df.index]["ORGANICO_preds"], color = "orange", alpha = 0.4,
plt.plot(pred3_df.loc[test_df.index]["ORGANICO_preds"], color = "red", linestyle = "-",
plt.plot(pred3_df.loc[test_df.index]["ORGANICO_preds_lower"], color = "green", linestyl
plt.plot(pred3_df.loc[test_df.index]["ORGANICO_preds_upper"], color = "green", linestyl
plt.title("Entrenamiento continuo - Predicciones continuas", size = 24)
plt.legend()

plt.subplot(2, 2, 3)
plt.plot(pred3_df.loc[train_df.index]["ORGANICO"], color = "blue", label = "Actuals", a
plt.plot(pred3_df.loc[train_df.index]["ORGANICO_preds"], color = "orange", linestyle =
plt.title("Ajuste a la Muestra", size = 24)
plt.legend()

plt.subplot(2, 2, 4)
plt.plot(pred3_df.loc[test_df.index]["ORGANICO"], color = "blue", label = "Actuals", al
plt.plot(pred3_df.loc[test_df.index]["ORGANICO_preds"], color = "red", linestyle = "-",
plt.plot(pred3_df.loc[test_df.index]["ORGANICO_preds_lower"], color = "green", linestyl
plt.plot(pred3_df.loc[test_df.index]["ORGANICO_preds_upper"], color = "green", linestyl
plt.title("Ajuste al Test", size = 24)
plt.legend()

plt.show()
```



- **Root Mean Squared Error**

In [70]:
```
rmse_in = np.sqrt(mean_squared_error(y_true = pred3_df.loc[train_df.index[1]:train_df.i
                                     y_pred = pred3_df.loc[train_df.index[1]:train_df.i

rmse_out = np.sqrt(mean_squared_error(y_true = pred3_df.loc[test_df.index]["ORGANICO"],
                                      y_pred = pred3_df.loc[test_df.index]["ORGANICO_pr

print(f"RMSE for In Sample Fit - {rmse_in}")
print(f"RMSE for Out of Sample Fit - {rmse_out}")
```

```
RMSE for In Sample Fit - 17.177891628667076
RMSE for Out of Sample Fit - 25.805386517678233
```

SKLEARN

In [71]:
```
df2
```

Out[71]:

Fecha	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Habitantes	COVID	Mes	ORG
2016-08-01	2016	31	119.565	100.120	0.00	0.000	60305	0	Aug	
2016-08-08	2016	32	124.960	104.175	0.00	0.000	60328	0	Aug	

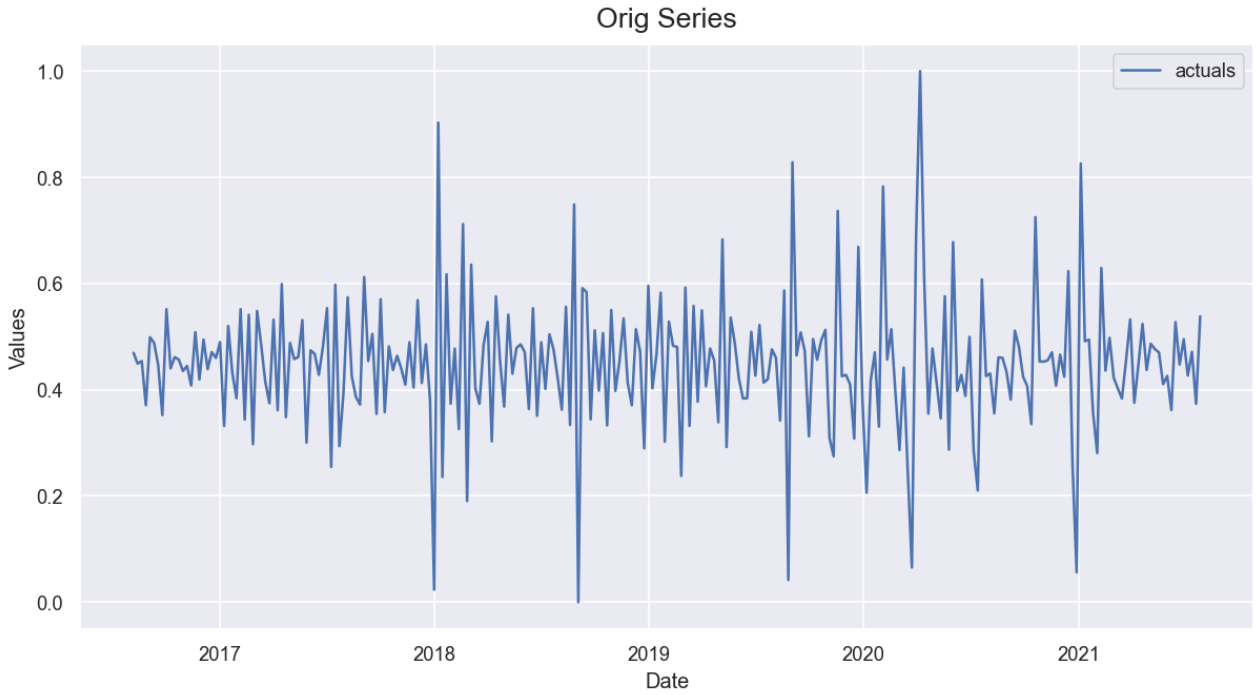
Fecha	Año	Semana	INORGANICO	ORGANICO	RECICLAJE	VEGETAL	Habitantes	COVID	Mes	ORG
2016-08-15	2016	33	124.730	104.150	0.00	0.000	60351	0	Aug	
2016-08-22	2016	34	141.650	105.110	0.00	0.000	60374	0	Aug	
2016-08-29	2016	35	150.138	90.070	0.00	0.000	60397	0	Aug	
...	...	...	...	...	...	...	...	...	...	
2021-06-28	2021	26	248.360	85.890	14.03	1.320	66346	0	Jun	
2021-07-05	2021	27	245.700	82.160	22.60	2.620	66370	0	Jul	
2021-07-12	2021	28	225.760	85.820	24.29	3.205	66395	0	Jul	
2021-07-19	2021	29	248.790	73.920	22.85	2.785	66419	0	Jul	
2021-07-26	2021	30	217.240	88.090	21.38	1.297	66443	0	Jul	

261 rows × 10 columns

In [72]:
```
def plot_test_export_summaries(f):
    """ exports the relevant statisitcal information and displays a plot of the test-se
    """
    f.plot_test_set(models=f.estimator,ci=True)
    plt.title(f'{f.estimator} test-set results',size=16)
    plt.show()
    return f.export('model_summaries',determine_best_by='TestSetRMSE')[
        [
            'ModelNickname',
            'HyperParams',
            'TestSetRMSE',
            'InSampleRMSE'
        ]
    ]
```

In [73]:
```
GridGenerator.get_example_grids()
```

In [74]:
```
f=Forecaster(y=scaled['ORGANICO_log_diff'],current_dates=df2.index)
f.plot()
plt.title('Orig Series',size=16)
plt.show()
```



In [75]:
```
#f.add_sklearn_estimator(Lasso,'lasso')
#f.add_sklearn_estimator(Ridge,'ridge')
f.add_sklearn_estimator(BaggingRegressor,'bagging')
#f.add_sklearn_estimator(StackingRegressor,'stacking')
```

In [76]:
```
fcst_length = 5
f.generate_future_dates(fcst_length)
f.set_test_length(.2)
f.add_ar_terms(7)
f.add_AR_terms((4,7))
f.add_seasonal_regressors('month','quarter','week','dayofyear',raw=False,sincos=True)
f.add_seasonal_regressors('dayofweek','is_leap_year','week',raw=False,dummy=True,drop_f
f.add_seasonal_regressors('year')
f
```

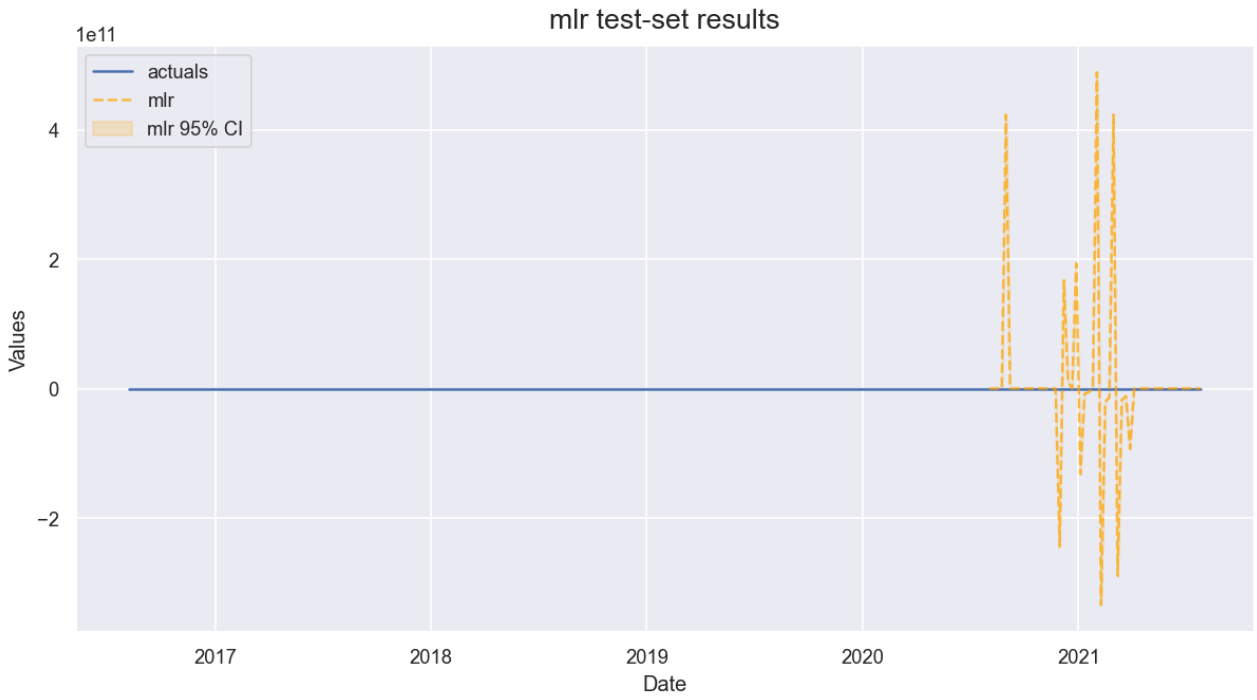
Out[76]:
```
Forecaster(
    DateStartActuals=2016-08-08T00:00:00.000000000
    DateEndActuals=2021-07-26T00:00:00.000000000
    Freq=W-MON
    N_actuals=260
    ForecastLength=5
    Xvars=['AR1', 'AR2', 'AR3', 'AR4', 'AR5', 'AR6', 'AR7', 'AR14', 'AR21', 'AR28', 'mon
thsin', 'monthcos', 'quartersin', 'quartercos', 'weeksin', 'weekcos', 'dayofyearsin', 'd
ayofyearcos', 'is_leap_year_True', 'week_10', 'week_11', 'week_12', 'week_13', 'week_1
4', 'week_15', 'week_16', 'week_17', 'week_18', 'week_19', 'week_2', 'week_20', 'week_2
1', 'week_22', 'week_23', 'week_24', 'week_25', 'week_26', 'week_27', 'week_28', 'week_2
9', 'week_3', 'week_30', 'week_31', 'week_32', 'week_33', 'week_34', 'week_35', 'week_3
6', 'week_37', 'week_38', 'week_39', 'week_4', 'week_40', 'week_41', 'week_42', 'week_4
3', 'week_44', 'week_45', 'week_46', 'week_47', 'week_48', 'week_49', 'week_5', 'week_5
0', 'week_51', 'week_52', 'week_53', 'week_6', 'week_7', 'week_8', 'week_9', 'year']
    Differenced=0
    TestLength=52
    ValidationLength=1
    ValidationMetric=rmse
    ForecastsEvaluated=[]
```

```
    CILevel=0.95
    BootstrapSamples=100
    CurrentEstimator=None
    GridsFile=Grids
)
```

MLR

In [77]:

```
f.set_estimator('mlr')
f.manual_forecast(dynamic_testing=fcst_length)
plot_test_export_summaries(f)
```



Out[77]:

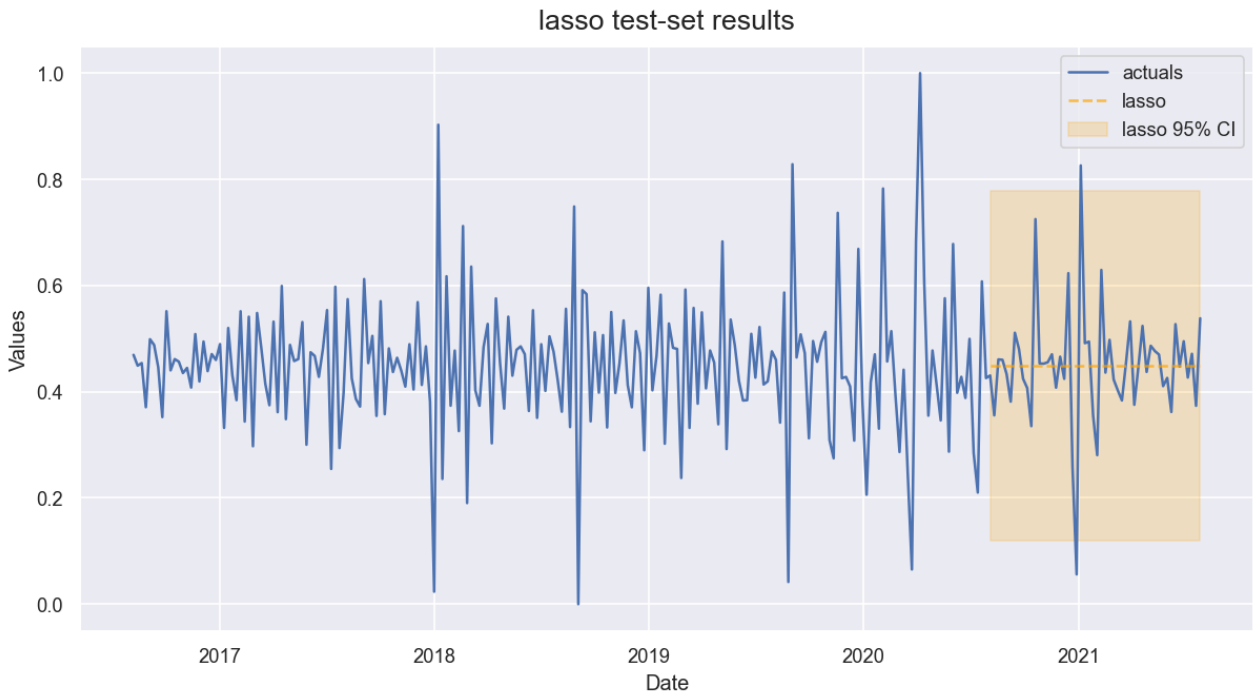
	ModelNickname	HyperParams	TestSetRMSE	InSampleRMSE
0	mlr	{}	1.348555e+11	0.090309

Lasso

- tune alpha with 100 choices using 3-fold cross validation
- use 60-step forecast evaluation in each tuning and testing iteration

In [78]:

```
f.set_estimator('lasso')
lasso_grid = {'alpha':np.linspace(0,2,100)}
f.ingest_grid(lasso_grid)
f.cross_validate(dynamic_tuning=fcst_length,k=3)
f.auto_forecast(dynamic_testing=fcst_length)
plot_test_export_summaries(f)
```



Out[78]:

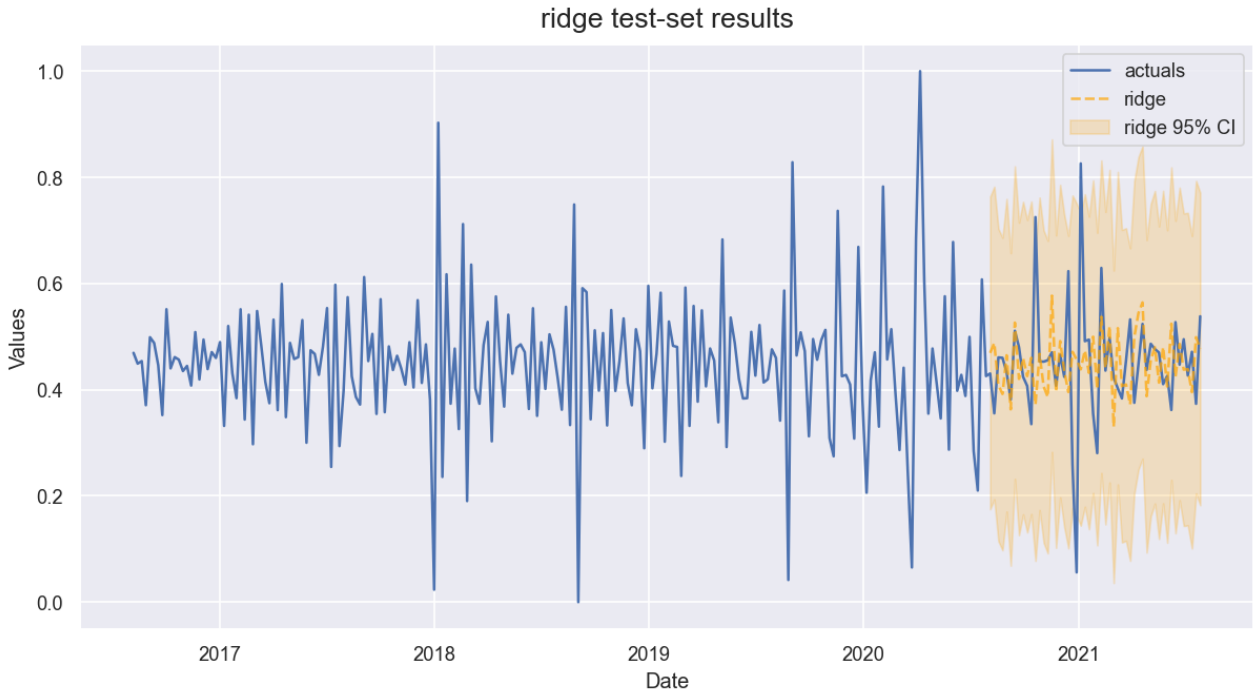
	ModelNickname	HyperParams	TestSetRMSE	InSampleRMSE
0	lasso	{'alpha': 0.020202020202020204}	1.086175e-01	0.132642
1	mlr	{}	1.348555e+11	0.090309

Ridge

- tune alpha with the same grid used for lasso

In [79]:

```
f.set_estimator('ridge')
f.ingest_grid(lasso_grid)
f.cross_validate(dynamic_tuning=fcst_length,k=3)
f.auto_forecast(dynamic_testing=fcst_length)
plot_test_export_summaries(f)
```



Out[79]:

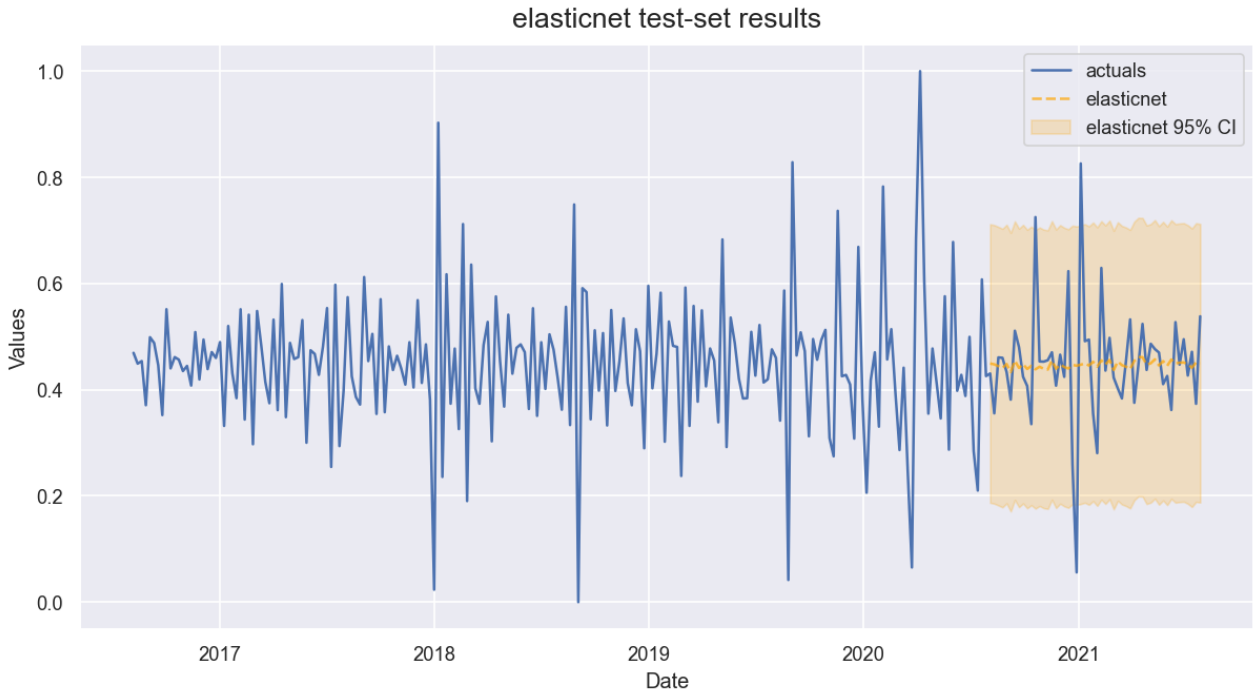
	ModelNickname	HyperParams	TestSetRMSE	InSampleRMSE
0	lasso	{'alpha': 0.020202020202020204}	1.086175e-01	0.132642
1	ridge	{'alpha': 2}	1.243745e-01	0.104643
2	mlr	{}	1.348555e+11	0.090309

Elasticnet

- this model mixes L1 and L2 regularization parameters
- its default grid is pretty good for finding the optimal alpha value

In [80]:

```
f.set_estimator('elasticnet')
f.cross_validate(dynamic_tuning=fcst_length,k=3)
f.auto_forecast(dynamic_testing=fcst_length)
plot_test_export_summaries(f)
```



Out[80]:

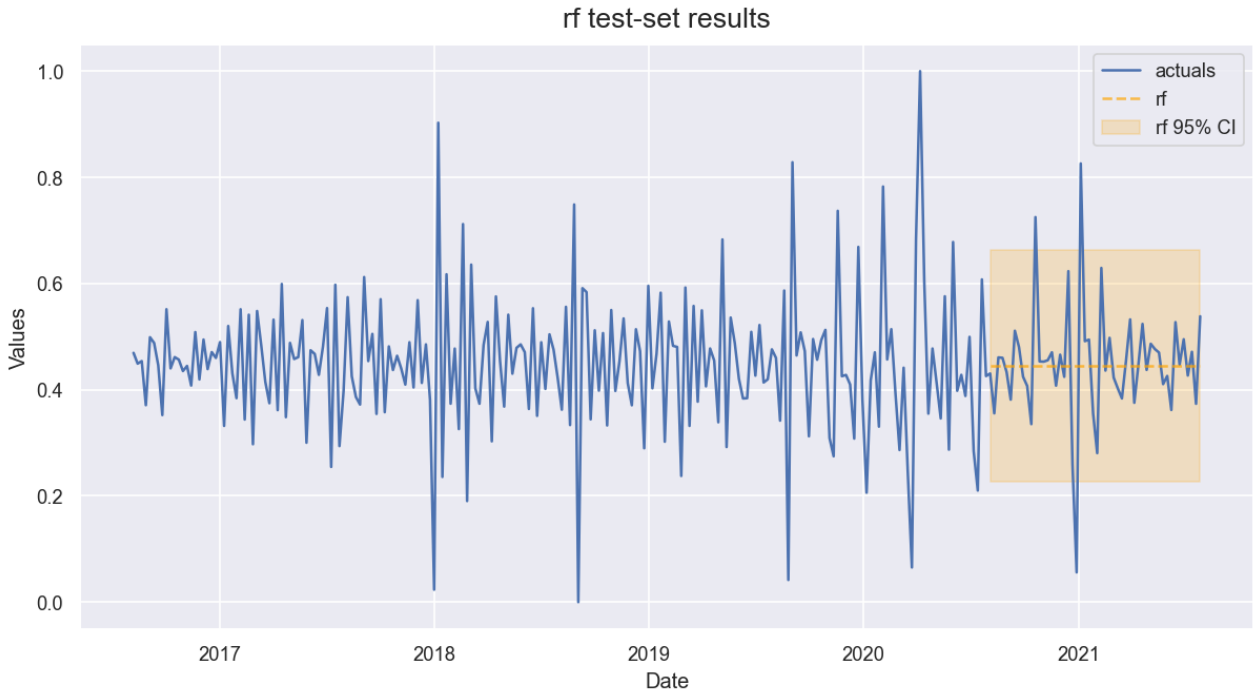
	ModelNickname	HyperParams	TestSetRMSE	InSampleRMSE
0	lasso	{'alpha': 0.020202020202020204}	1.086175e-01	0.132642
1	elasticnet	{'alpha': 0.3, 'l1_ratio': 0}	1.087292e-01	0.129458
2	ridge	{'alpha': 2}	1.243745e-01	0.104643
3	mlr	{}	1.348555e+11	0.090309

RF

create a grid to tune Random Forest with more options than what is available in the default grids

In [81]:

```
f.set_estimator('rf')
rf_grid = {
    'max_depth':[2,3,4,5],
    'n_estimators':[100,200,500],
    'max_features':['auto','sqrt','log2'],
    'max_samples':[.75,.9,1],
}
f.ingest_grid(rf_grid)
f.cross_validate(dynamic_tuning=fcst_length,k=3)
f.auto_forecast(dynamic_testing=fcst_length)
plot_test_export_summaries(f)
```



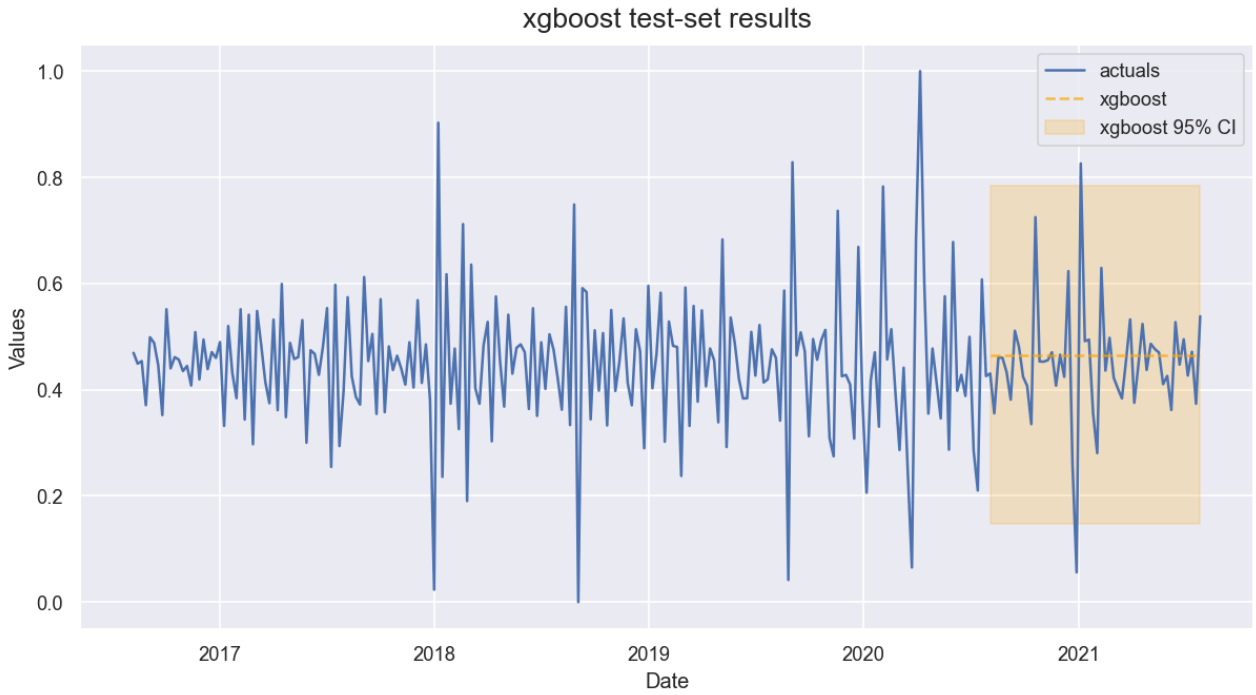
Out[81]:

	ModelNickname	HyperParams	TestSetRMSE	InSampleRMSE
0	lasso	{'alpha': 0.020202020202020204}	1.086175e-01	0.132642
1	rf	{'max_depth': 2, 'n_estimators': 200, 'max_fea...	1.087232e-01	0.132893
2	elasticnet	{'alpha': 0.3, 'l1_ratio': 0}	1.087292e-01	0.129458
3	ridge	{'alpha': 2}	1.243745e-01	0.104643
4	mlr	{}	1.348555e+11	0.090309

XGBoost

In [82]:

```
f.set_estimator('xgboost')
xgboost_grid = {
     'n_estimators':[150,200,250],
     'scale_pos_weight':[5,10],
     'learning_rate':[0.1,0.2],
     'gamma':[0,3,5],
     'subsample':[0.8,0.9]
}
f.ingest_grid(xgboost_grid)
f.cross_validate(dynamic_tuning=fcst_length,k=3)
f.auto_forecast(dynamic_testing=fcst_length)
plot_test_export_summaries(f)
```



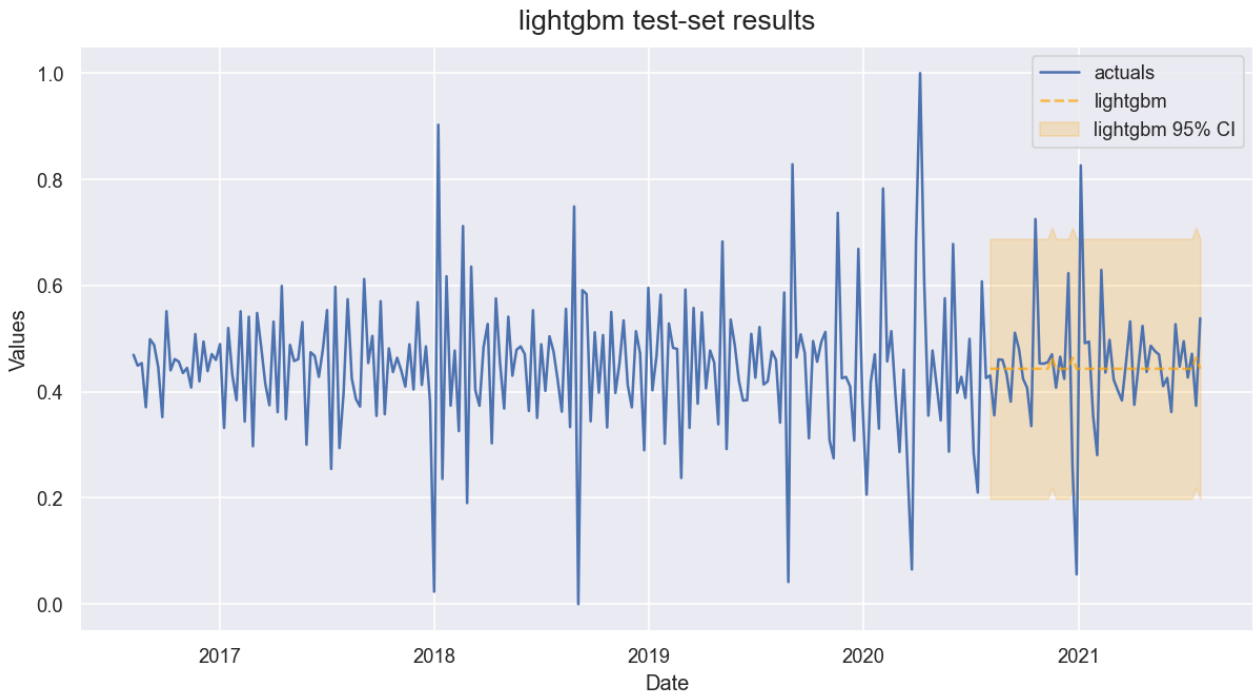
Out[82]:

	ModelNickname	HyperParams	TestSetRMSE	InSampleRMSE
0	lasso	{'alpha': 0.020202020202020204}	1.086175e-01	0.132642
1	rf	{'max_depth': 2, 'n_estimators': 200, 'max_fea...	1.087232e-01	0.132893
2	elasticnet	{'alpha': 0.3, 'l1_ratio': 0}	1.087292e-01	0.129458
3	xgboost	{'n_estimators': 250, 'scale_pos_weight': 5, '...	1.097996e-01	0.132988
4	ridge	{'alpha': 2}	1.243745e-01	0.104643
5	mlr	{}	1.348555e+11	0.090309

LightGBM

In [83]:
```
f.set_estimator('lightgbm')
lightgbm_grid = {
    'n_estimators':[150,200,250],
    'boosting_type':['gbdt','dart','goss'],
    'max_depth':[1,2,3],
    'learning_rate':[0.001,0.01,0.1],
    'reg_alpha':np.linspace(0,1,5),
    'reg_lambda':np.linspace(0,1,5),
    'num_leaves':np.arange(5,50,5),
}
f.ingest_grid(lightgbm_grid)
f.limit_grid_size(100,random_seed=2)
f.cross_validate(dynamic_tuning=fcst_length,k=3)
f.auto_forecast(dynamic_testing=fcst_length)
```

In [84]:
```
plot_test_export_summaries(f)
```



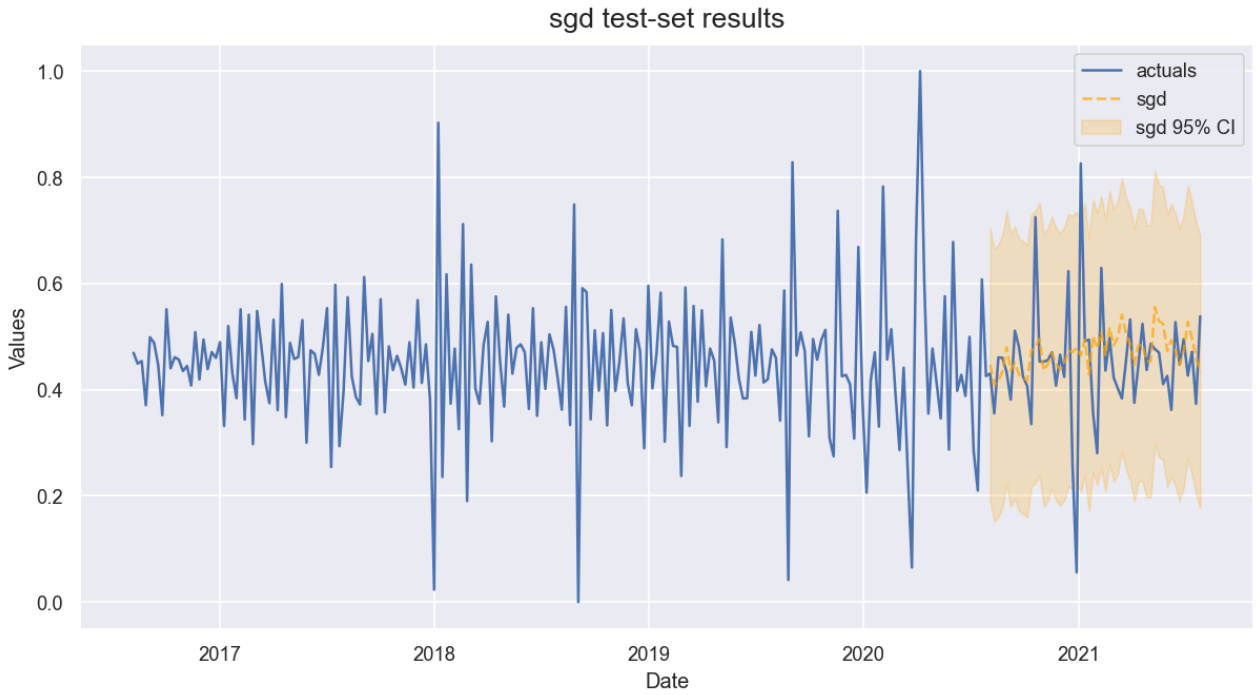
Out[84]:

	ModelNickname	HyperParams	TestSetRMSE	InSampleRMSE
0	lasso	{'alpha': 0.020202020202020204}	1.086175e-01	0.132642
1	rf	{'max_depth': 2, 'n_estimators': 200, 'max_fea...	1.087232e-01	0.132893
2	elasticnet	{'alpha': 0.3, 'l1_ratio': 0}	1.087292e-01	0.129458
3	lightgbm	{'n_estimators': 150, 'boosting_type': 'goss',...	1.097457e-01	0.129366
4	xgboost	{'n_estimators': 250, 'scale_pos_weight': 5, '...	1.097996e-01	0.132988
5	ridge	{'alpha': 2}	1.243745e-01	0.104643
6	mlr	{}	1.348555e+11	0.090309

SGD

In [85]:

```
f.set_estimator('sgd')
f.cross_validate(dynamic_tuning=fcst_length,k=3)
f.auto_forecast(dynamic_testing=fcst_length)
plot_test_export_summaries(f)
```



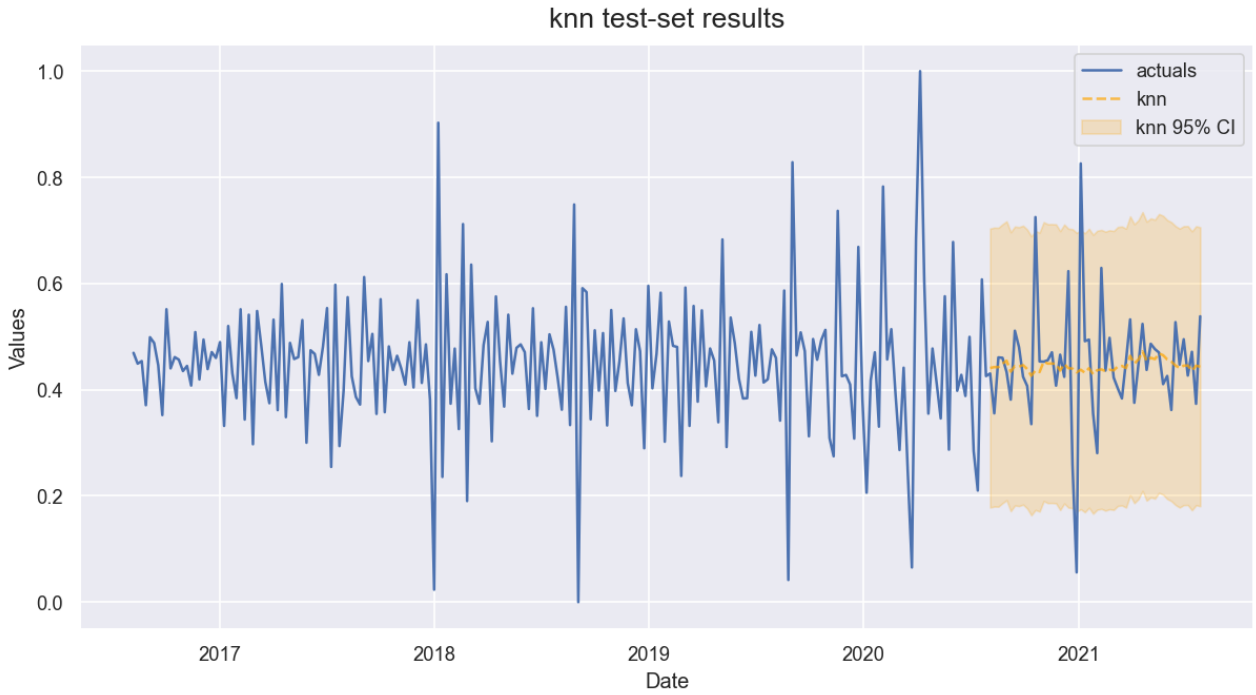
Out[85]:

	ModelNickname	HyperParams	TestSetRMSE	InSampleRMSE
0	lasso	{'alpha': 0.020202020202020204}	1.086175e-01	0.132642
1	rf	{'max_depth': 2, 'n_estimators': 200, 'max_fea...	1.087232e-01	0.132893
2	elasticnet	{'alpha': 0.3, 'l1_ratio': 0}	1.087292e-01	0.129458
3	lightgbm	{'n_estimators': 150, 'boosting_type': 'goss',...	1.097457e-01	0.129366
4	xgboost	{'n_estimators': 250, 'scale_pos_weight': 5, '...	1.097996e-01	0.132988
5	sgd	{'penalty': 'elasticnet', 'l1_ratio': 0.85, 'l...	1.148513e-01	0.128447
6	ridge	{'alpha': 2}	1.243745e-01	0.104643
7	mlr	{}	1.348555e+11	0.090309

KNN

In [86]:

```
f.set_estimator('knn')
f.cross_validate(dynamic_tuning=fcst_length,k=3)
f.auto_forecast(dynamic_testing=fcst_length)
plot_test_export_summaries(f)
```



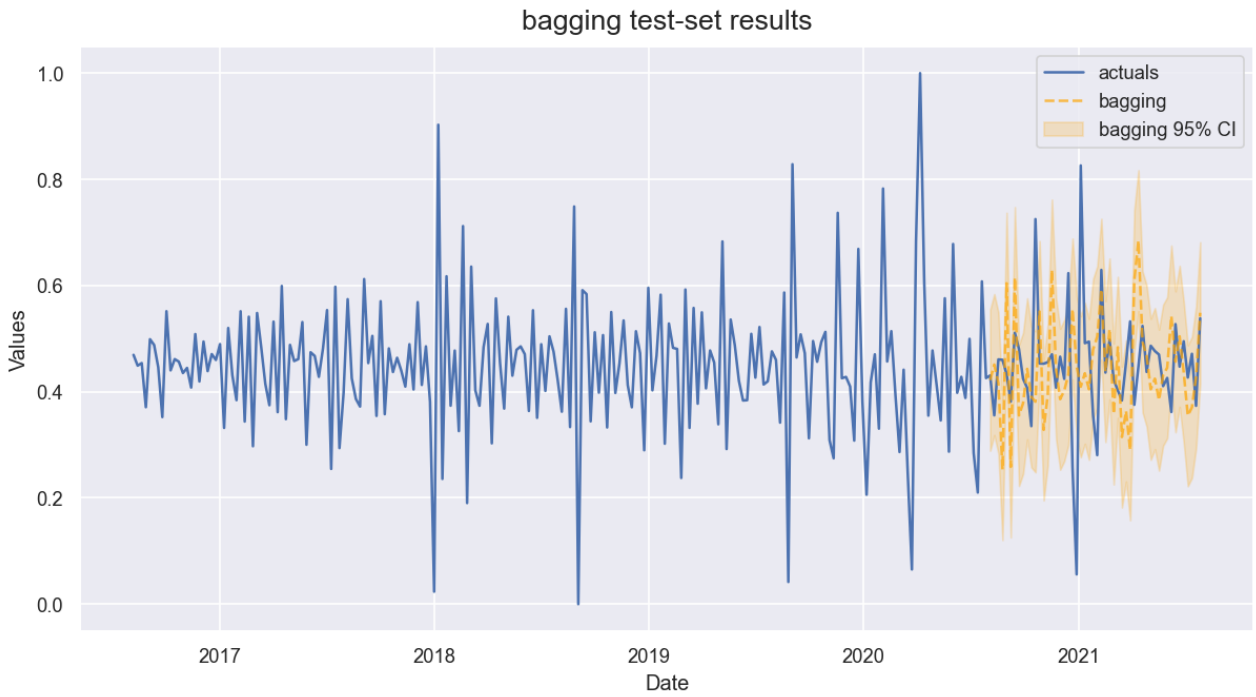
Out[86]:

	ModelNickname	HyperParams	TestSetRMSE	InSampleRMSE
0	knn	{'n_neighbors': 56}	1.080583e-01	0.131835
1	lasso	{'alpha': 0.020202020202020204}	1.086175e-01	0.132642
2	rf	{'max_depth': 2, 'n_estimators': 200, 'max_fea...	1.087232e-01	0.132893
3	elasticnet	{'alpha': 0.3, 'l1_ratio': 0}	1.087292e-01	0.129458
4	lightgbm	{'n_estimators': 150, 'boosting_type': 'goss',...	1.097457e-01	0.129366
5	xgboost	{'n_estimators': 250, 'scale_pos_weight': 5, '...	1.097996e-01	0.132988
6	sgd	{'penalty': 'elasticnet', 'l1_ratio': 0.85, 'l...	1.148513e-01	0.128447
7	ridge	{'alpha': 2}	1.243745e-01	0.104643
8	mlr	{}	1.348555e+11	0.090309

BaggingRegressor

In [87]:

```
f.set_estimator('bagging')
f.manual_forecast(
    base_estimator = MLPRegressor(
        hidden_layer_sizes=(100,100,100)
        ,solver='lbfgs'
    ),
    max_samples = 0.9,
    max_features = 0.5,
    dynamic_testing=fcst_length,
)
plot_test_export_summaries(f)
```



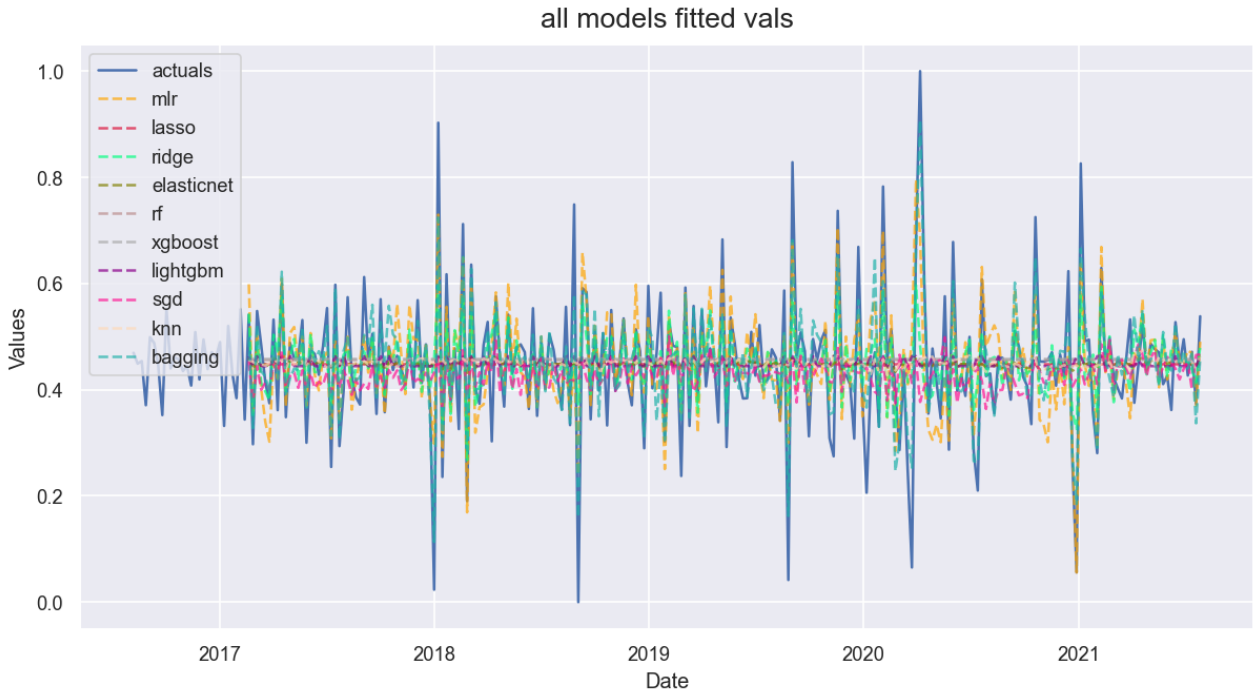
Out[87]:

	ModelNickname	HyperParams	TestSetRMSE	InSampleRMSE
0	knn	{'n_neighbors': 56}	1.080583e-01	0.131835
1	lasso	{'alpha': 0.020202020202020204}	1.086175e-01	0.132642
2	rf	{'max_depth': 2, 'n_estimators': 200, 'max_fea...	1.087232e-01	0.132893
3	elasticnet	{'alpha': 0.3, 'l1_ratio': 0}	1.087292e-01	0.129458
4	lightgbm	{'n_estimators': 150, 'boosting_type': 'goss',...	1.097457e-01	0.129366
5	xgboost	{'n_estimators': 250, 'scale_pos_weight': 5, '...	1.097996e-01	0.132988
6	sgd	{'penalty': 'elasticnet', 'l1_ratio': 0.85, 'l...	1.148513e-01	0.128447
7	ridge	{'alpha': 2}	1.243745e-01	0.104643
8	bagging	{'base_estimator': MLPRegressor(hidden_layer_s...	1.465466e-01	0.063067
9	mlr	{}	1.348555e+11	0.090309

StackingRegressor

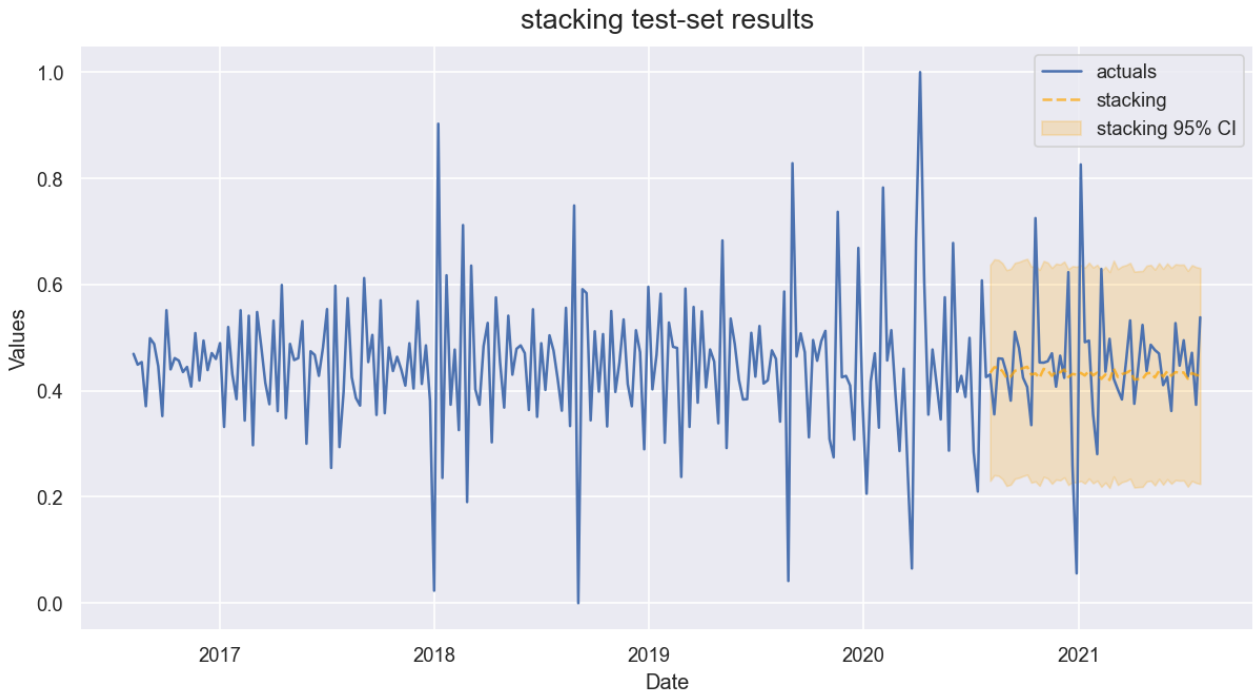
In [88]:

```
f.plot_fitted()
plt.title('all models fitted vals',size=16)
plt.show()
```



In [89]:
```
f.set_estimator('bagging')
results = f.export('model_summaries')
estimators = [
    'knn',
    'xgboost',
    'lightgbm',
    'sgd',
]

mlp_stack(f,estimators,call_me='stacking')
```

In [90]:
```
plot_test_export_summaries(f)
```



Out[90]:

	ModelNickname	HyperParams	TestSetRMSE	InSampleRMSE
0	knn	{'n_neighbors': 56}	1.080583e-01	0.131835
1	lasso	{'alpha': 0.020202020202020204}	1.086175e-01	0.132642
2	rf	{'max_depth': 2, 'n_estimators': 200, 'max_fea...	1.087232e-01	0.132893
3	elasticnet	{'alpha': 0.3, 'l1_ratio': 0}	1.087292e-01	0.129458
4	lightgbm	{'n_estimators': 150, 'boosting_type': 'goss',...	1.097457e-01	0.129366
5	xgboost	{'n_estimators': 250, 'scale_pos_weight': 5, '...	1.097996e-01	0.132988
6	stacking	{'estimators': [('knn', KNeighborsRegressor(n_...	1.104030e-01	0.128179
7	sgd	{'penalty': 'elasticnet', 'l1_ratio': 0.85, 'l...	1.148513e-01	0.128447
8	ridge	{'alpha': 2}	1.243745e-01	0.104643
9	bagging	{'base_estimator': MLPRegressor(hidden_layer_s...	1.465466e-01	0.063067
10	mlr	{}	1.348555e+11	0.090309

Continua?

Multivariate

Scalecast - Univariate

In [91]:
```
forg = Forecaster(y=scaled['ORGANICO_log_diff'],current_dates = scaled.index)
forg
```

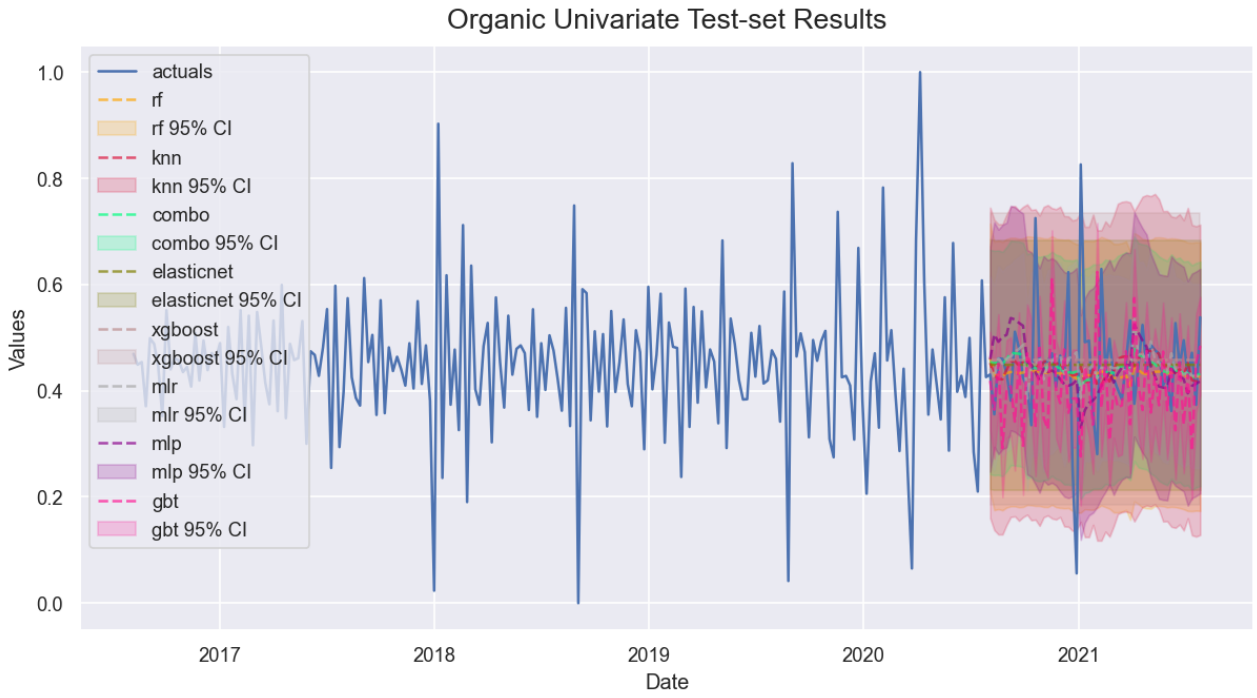
Out[91]:
```
Forecaster(
    DateStartActuals=2016-08-08T00:00:00.000000000
    DateEndActuals=2021-07-26T00:00:00.000000000
    Freq=W-MON
    N_actuals=260
    ForecastLength=0
    Xvars=[]
    Differenced=0
    TestLength=1
    ValidationLength=1
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    BootstrapSamples=100
    CurrentEstimator=None
    GridsFile=Grids
)
```

In [92]:
```
forg.generate_future_dates(52)
forg.set_test_length(.2)
forg.set_validation_length(4)
forg.add_seasonal_regressors('week','month','quarter',raw=False,sincos=True)
forg.add_seasonal_regressors('year')
forg.add_time_trend()
```

```
forg.add_cycle(26)
forg.add_ar_terms(3)
```

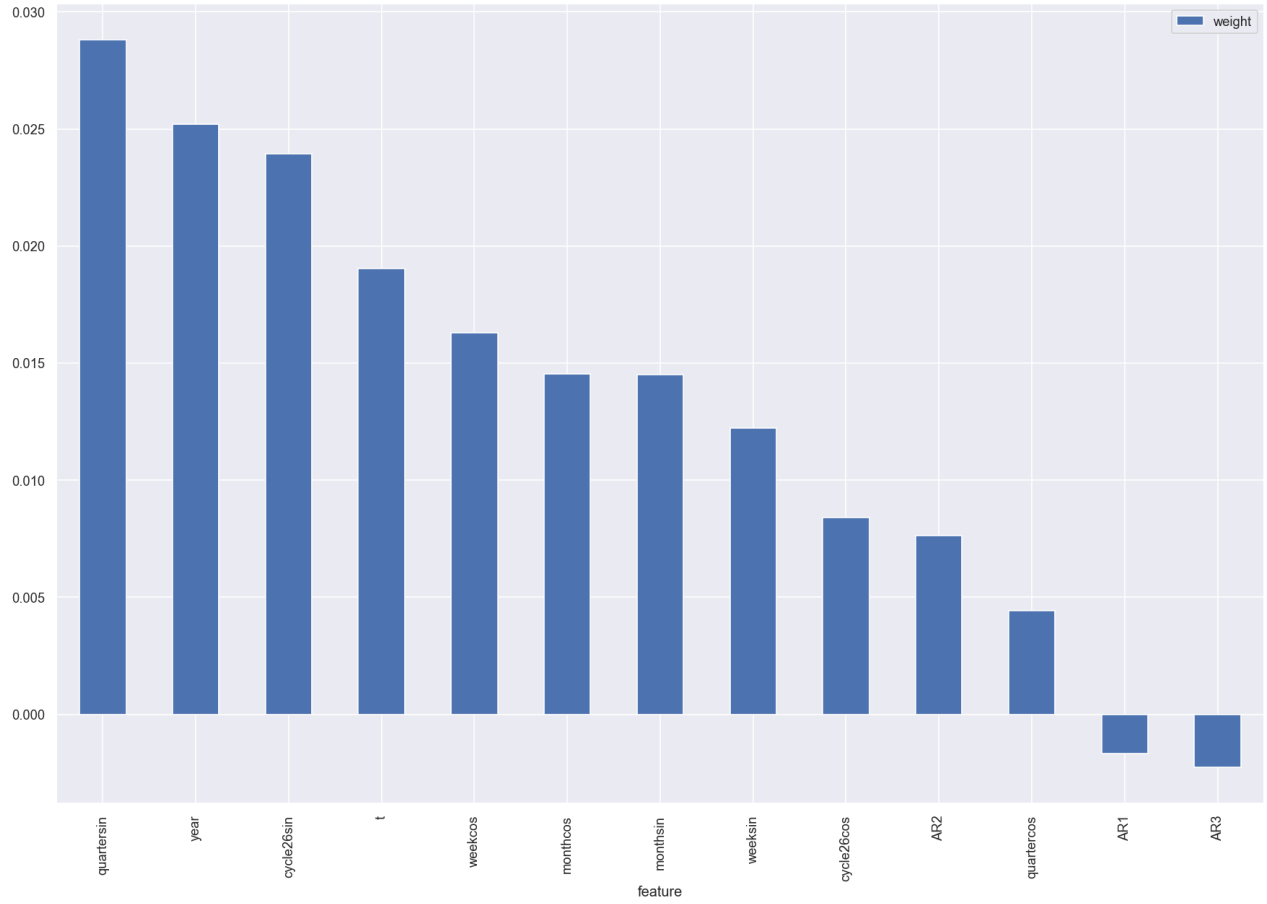
In [93]:
```
models = ('mlr','elasticnet','knn','rf','gbt','xgboost','mlp')
forg.tune_test_forecast(models,feature_importance=True)
forg.set_estimator('combo')
forg.manual_forecast(how='weighted')
```

In [94]:
```
forg.plot_test_set(ci=True,order_by='LevelTestSetMAPE')
plt.title('Organic Univariate Test-set Results',size=16)
plt.show()
```



Feature Importance

In [95]:
```
forg.export_feature_importance('knn').plot.bar(y='weight')
plt.show()
```



Model Summaries

In [96]:
```
pd.set_option('display.float_format',  '{:.4f}'.format)
ms = export_model_summaries({'Organico':forg},determine_best_by='LevelTestSetRMSE')
ms[
    [
        'ModelNickname',
        'Series',
        'Integration',
        'LevelTestSetRMSE',
        'InSampleRMSE',
        'best_model'
    ]
]
```

Out[96]:

	ModelNickname	Series	Integration	LevelTestSetRMSE	InSampleRMSE	best_model
0	elasticnet	Organico	0	0.1086	0.1273	True
1	knn	Organico	0	0.1089	0.1252	False
2	xgboost	Organico	0	0.1091	0.1277	False
3	rf	Organico	0	0.1102	0.1129	False
4	combo	Organico	0	0.1123	0.1183	False
5	mlr	Organico	0	0.1167	0.1112	False
6	mlp	Organico	0	0.1246	0.1272	False

	ModelNickname	Series	Integration	LevelTestSetRMSE	InSampleRMSE	best_model
7	gbt	Organico	0	0.1578	0.0502	False

In [97]:
```
print('-'*100)
for series in ms['Series'].unique():
    print('univariate average test RMSE for {}: {:.2f}'.format(series,ms.loc[ms['Series
    print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
univariate average test RMSE for Organico: 0.12
----------------------------------------------------------------------------------------------------
```

Scalecast - Multivariate

- Habitantes
- Reciclaje
- Organico
- Inorganico

In [98]:
```
scaled.columns
```

Out[98]:
```
Index(['Año', 'Semana', 'INORGANICO', 'ORGANICO', 'RECICLAJE', 'VEGETAL',
       'Habitantes', 'COVID', 'ORGANICO_log_diff'],
      dtype='object')
```

In [99]:
```
finorg = Forecaster(y=scaled['INORGANICO'],current_dates = scaled.index)
frecic = Forecaster(y=scaled['RECICLAJE'],current_dates = scaled.index)
fveg = Forecaster(y=scaled['VEGETAL'],current_dates = scaled.index)
fcovid = Forecaster(y=scaled['COVID'],current_dates = scaled.index)
fhab = Forecaster(y=scaled['Habitantes'],current_dates = scaled.index)
fyear = Forecaster(y=scaled['Año'],current_dates = scaled.index)
fweek = Forecaster(y=scaled['Semana'],current_dates = scaled.index)
```

In [100…
```
mvf = MVForecaster(forg,finorg,frecic,fhab,fveg,fcovid,fyear,fweek,names=['Organico','I
mvf.set_test_length(.2)
mvf.set_validation_length(4)
mvf
```

Out[100…
```
MVForecaster(
    DateStartActuals=2016-08-08T00:00:00.000000000
    DateEndActuals=2021-07-26T00:00:00.000000000
    Freq=W-MON
    N_actuals=260
    N_series=8
    SeriesNames=['Organico', 'Inorganico', 'Reciclaje', 'Habitantes', 'Vegetal', 'COVI
D', 'Año', 'Semana']
    ForecastLength=52
    Xvars=['weeksin', 'weekcos', 'monthsin', 'monthcos', 'quartersin', 'quartercos', 'ye
ar', 't', 'cycle26sin', 'cycle26cos']
    TestLength=52
```

```
    ValidationLength=4
    ValidationMetric=rmse
    ForecastsEvaluated=[]
    CILevel=0.95
    BootstrapSamples=100
    CurrentEstimator=mlr
    OptimizeOn=mean
    GridsFile=MVGrids
)
```

View Series Correlation

In [101…

```
mvf.corr()
```

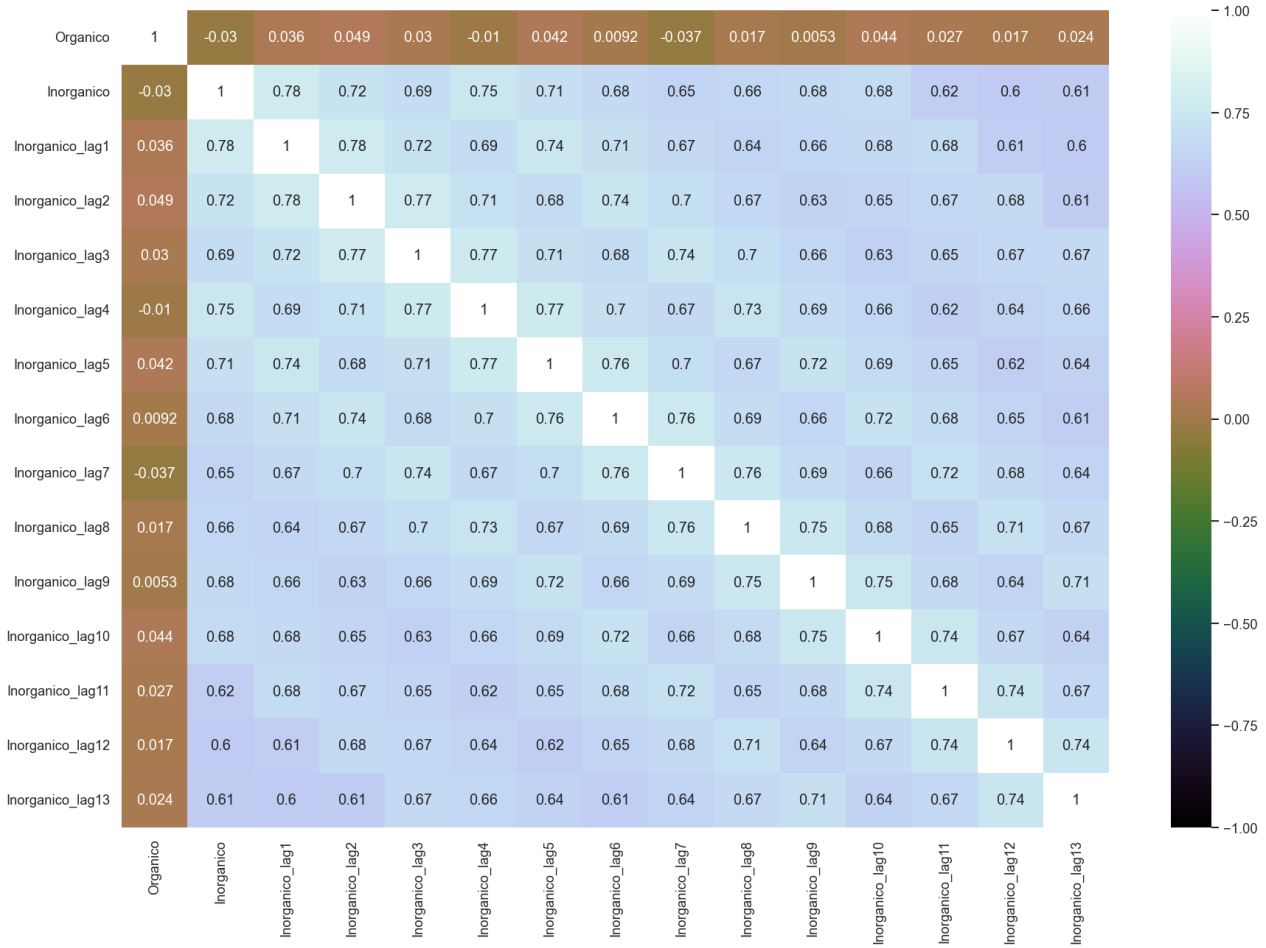
Out[101…

	Organico	Inorganico	Reciclaje	Habitantes	Vegetal	COVID	Año	Semana
Organico	1.0000	-0.0295	0.0032	-0.0040	0.0466	0.0504	0.0022	-0.0306
Inorganico	-0.0295	1.0000	0.3353	0.6095	-0.0641	-0.1398	0.5930	-0.0471
Reciclaje	0.0032	0.3353	1.0000	0.8191	-0.0305	0.1964	0.7954	-0.0209
Habitantes	-0.0040	0.6095	0.8191	1.0000	0.0321	0.1699	0.9814	-0.0842
Vegetal	0.0466	-0.0641	-0.0305	0.0321	1.0000	-0.0553	0.0479	-0.0686
COVID	0.0504	-0.1398	0.1964	0.1699	-0.0553	1.0000	0.1887	-0.1204
Año	0.0022	0.5930	0.7954	0.9814	0.0479	0.1887	1.0000	-0.2734
Semana	-0.0306	-0.0471	-0.0209	-0.0842	-0.0686	-0.1204	-0.2734	1.0000

View Series Correlation with each others' lags

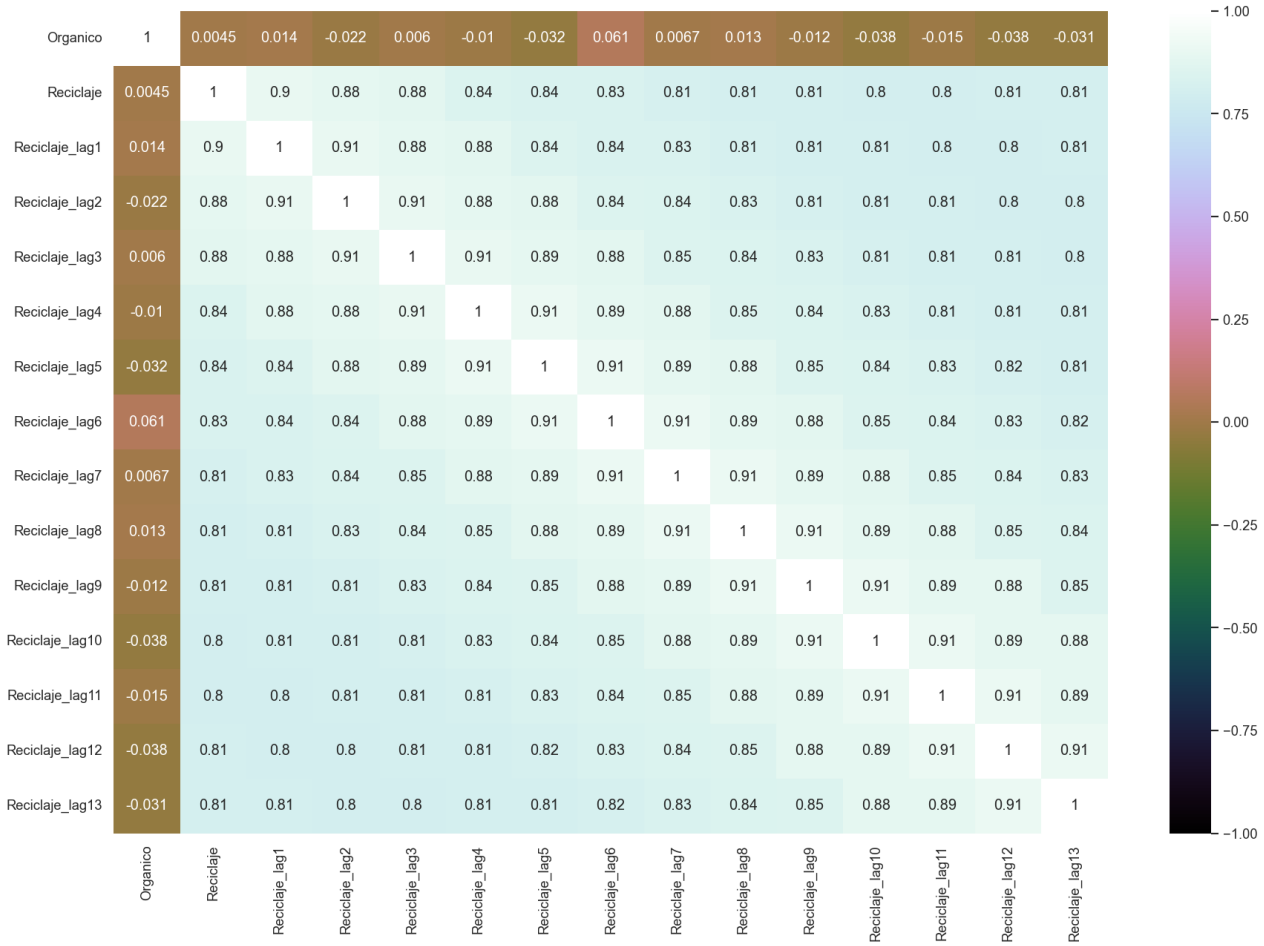
In [102…

```
mvf.corr_lags(
    y='Organico',
    x='Inorganico',
    lags=13,
    disp='heatmap',
    annot=True,
    vmin=-1,
    vmax=1,
    cmap = 'cubehelix',
)
plt.show()
```



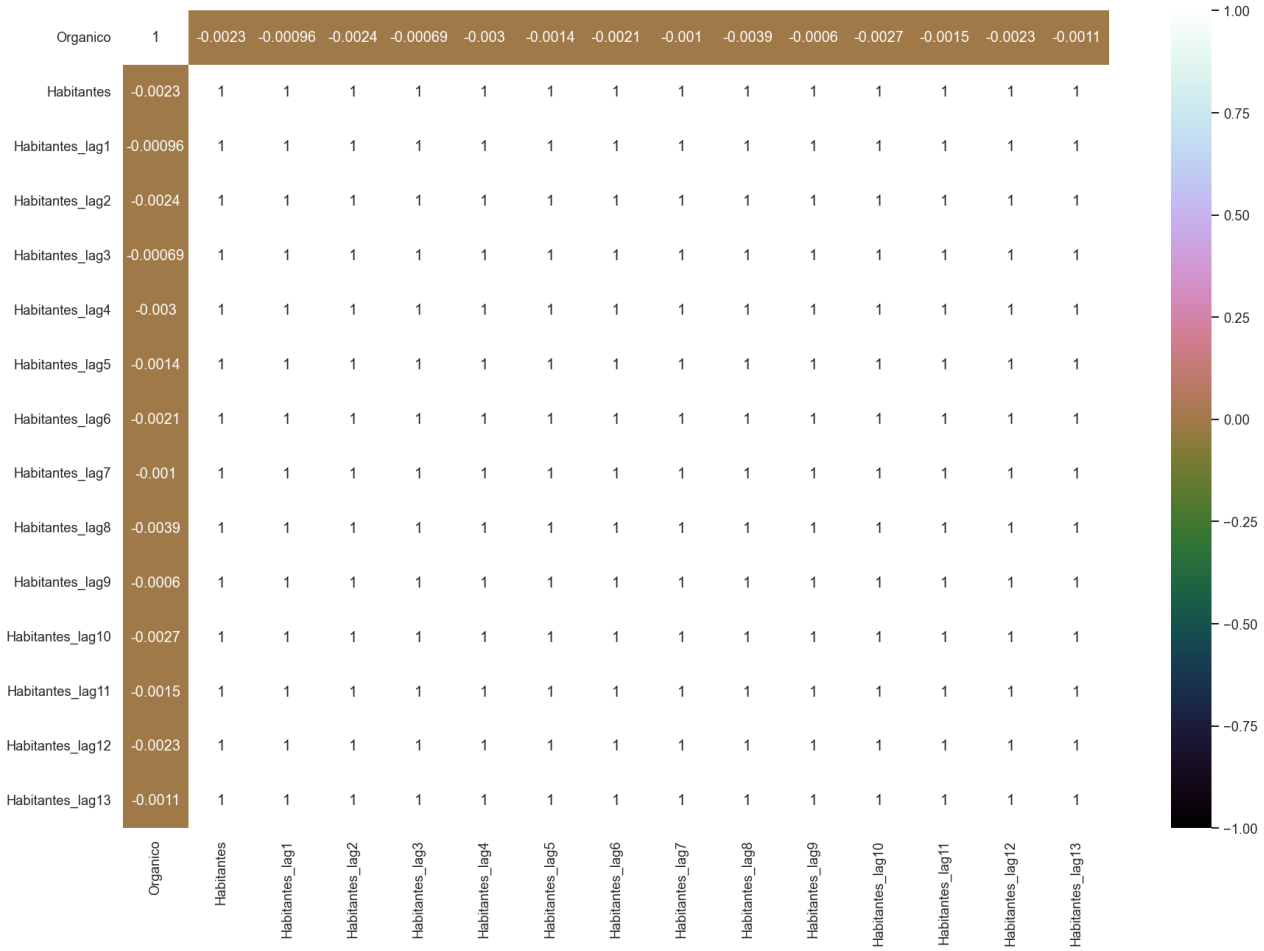
In [103…

```
mvf.corr_lags(
    y='Organico',
    x='Reciclaje',
    lags=13,
    disp='heatmap',
    annot=True,
    vmin=-1,
    vmax=1,
    cmap = 'cubehelix',
)
plt.show()
```



In [104…

```
mvf.corr_lags(
    y='Organico',
    x='Habitantes',
    lags=13,
    disp='heatmap',
    annot=True,
    vmin=-1,
    vmax=1,
    cmap = 'cubehelix',
)
plt.show()
```



Set Optimize On

In [105…

```
mvf.set_optimize_on('Organico')
```

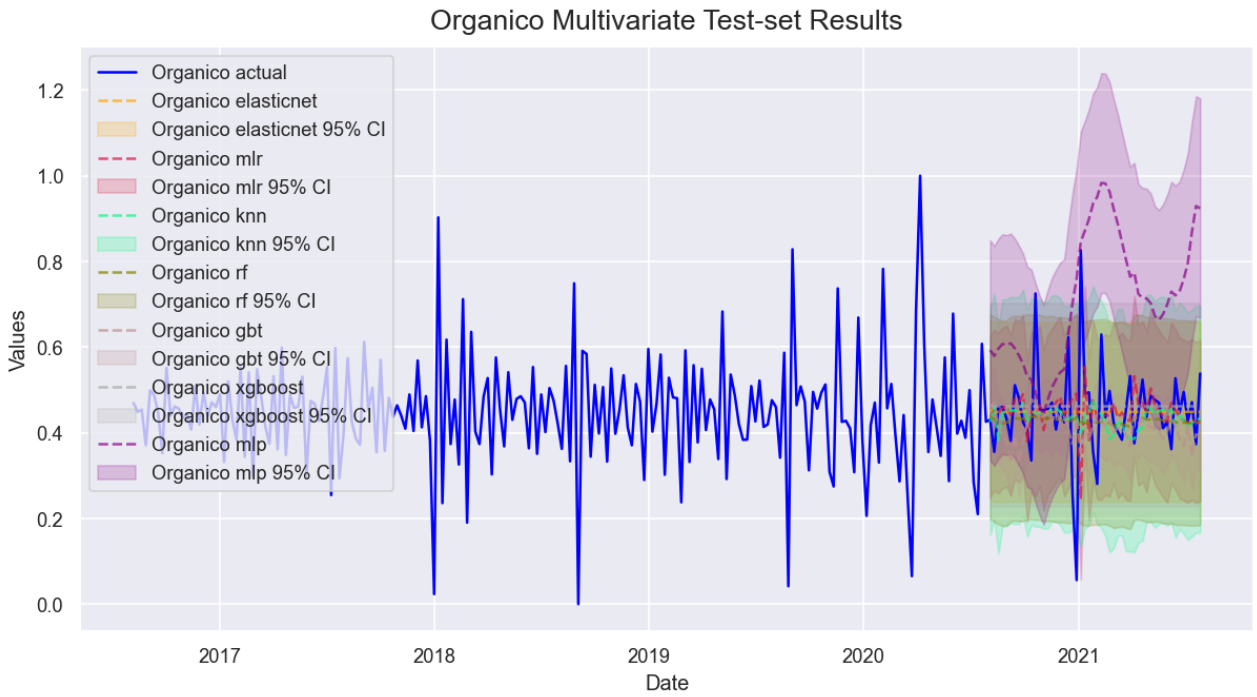
Tune and Test with Selected Models

In [106…

```
mvf.tune_test_forecast(models)
mvf.set_best_model(determine_best_by='LevelTestSetRMSE')
```

In [107…

```
mvf.plot_test_set(series='Organico',put_best_on_top=True,ci=True)
plt.title('Organico Multivariate Test-set Results',size=16)
plt.show()
```



In [108…

```
pd.options.display.max_colwidth = 100
results = mvf.export('model_summaries')
results[
    [
        'ModelNickname',
        'Series',
        'HyperParams',
        'LevelTestSetRMSE',
        'InSampleRMSE',
        'Lags'
    ]
]
```

Out[108…

	ModelNickname	Series	HyperParams	LevelTestSetRMSE	InSampleRMSE	Lags
0	elasticnet	Organico	{'alpha': 0.1, 'l1_ratio': 0.25}	0.1086	0.1194	3
1	mlr	Organico	{}	0.1322	0.1113	1
2	knn	Organico	{'n_neighbors': 11}	0.1168	0.1258	3
3	rf	Organico	{'max_depth': 2, 'n_estimators': 100, 'max_features': 'sqrt', 'max_samples': 0.75}	0.1116	0.1129	1
4	gbt	Organico	{'max_depth': 3, 'max_features': 'sqrt'}	0.1249	0.0419	6
5	xgboost	Organico	{'n_estimators': 200, 'scale_pos_weight': 5, 'learning_rate': 0.2, 'gamma': 3, 'subsample': 0.8}	0.1095	0.1275	1
6	mlp	Organico	{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'solver': 'adam'}	0.3213	0.1389	1

	ModelNickname	Series	HyperParams	LevelTestSetRMSE	InSampleRMSE	Lags
7	elasticnet	Inorganico	{'alpha': 0.1, 'l1_ratio': 0.25}	0.3499	0.0975	3
8	mlr	Inorganico	{}	0.3307	0.0865	1
9	knn	Inorganico	{'n_neighbors': 11}	0.3583	0.0914	3
10	rf	Inorganico	{'max_depth': 2, 'n_estimators': 100, 'max_features': 'sqrt', 'max_samples': 0.75}	0.3095	0.0812	1
11	gbt	Inorganico	{'max_depth': 3, 'max_features': 'sqrt'}	0.2579	0.0324	6
12	xgboost	Inorganico	{'n_estimators': 200, 'scale_pos_weight': 5, 'learning_rate': 0.2, 'gamma': 3, 'subsample': 0.8}	0.3504	0.1223	1
13	mlp	Inorganico	{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'solver': 'adam'}	0.3262	0.1319	1
14	elasticnet	Reciclaje	{'alpha': 0.1, 'l1_ratio': 0.25}	0.1099	0.0780	3
15	mlr	Reciclaje	{}	0.1563	0.0714	1
16	knn	Reciclaje	{'n_neighbors': 11}	0.0746	0.0643	3
17	rf	Reciclaje	{'max_depth': 2, 'n_estimators': 100, 'max_features': 'sqrt', 'max_samples': 0.75}	0.0792	0.0631	1
18	gbt	Reciclaje	{'max_depth': 3, 'max_features': 'sqrt'}	0.0961	0.0148	6
19	xgboost	Reciclaje	{'n_estimators': 200, 'scale_pos_weight': 5, 'learning_rate': 0.2, 'gamma': 3, 'subsample': 0.8}	0.1494	0.1133	1
20	mlp	Reciclaje	{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'solver': 'adam'}	0.0958	0.1065	1
21	elasticnet	Habitantes	{'alpha': 0.1, 'l1_ratio': 0.25}	0.1648	0.0311	3
22	mlr	Habitantes	{}	0.0062	0.0032	1
23	knn	Habitantes	{'n_neighbors': 11}	0.2834	0.0486	3
24	rf	Habitantes	{'max_depth': 2, 'n_estimators': 100, 'max_features': 'sqrt', 'max_samples': 0.75}	0.2780	0.0689	1
25	gbt	Habitantes	{'max_depth': 3, 'max_features': 'sqrt'}	0.1444	0.0023	6

	ModelNickname	Series	HyperParams	LevelTestSetRMSE	InSampleRMSE	Lags
26	xgboost	Habitantes	{'n_estimators': 200, 'scale_pos_weight': 5, 'learning_rate': 0.2, 'gamma': 3, 'subsample': 0.8}	0.3981	0.1253	1
27	mlp	Habitantes	{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'solver': 'adam'}	0.1218	0.0711	1
28	elasticnet	Vegetal	{'alpha': 0.1, 'l1_ratio': 0.25}	0.1688	0.1618	3
29	mlr	Vegetal	{}	0.2025	0.1572	1
30	knn	Vegetal	{'n_neighbors': 11}	0.2073	0.1455	3
31	rf	Vegetal	{'max_depth': 2, 'n_estimators': 100, 'max_features': 'sqrt', 'max_samples': 0.75}	0.1807	0.1577	1
32	gbt	Vegetal	{'max_depth': 3, 'max_features': 'sqrt'}	0.1557	0.0602	6
33	xgboost	Vegetal	{'n_estimators': 200, 'scale_pos_weight': 5, 'learning_rate': 0.2, 'gamma': 3, 'subsample': 0.8}	0.1790	0.1879	1
34	mlp	Vegetal	{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'solver': 'adam'}	0.2955	0.1709	1
35	elasticnet	COVID	{'alpha': 0.1, 'l1_ratio': 0.25}	0.0464	0.0931	3
36	mlr	COVID	{}	0.1741	0.0816	1
37	knn	COVID	{'n_neighbors': 11}	0.0378	0.1053	3
38	rf	COVID	{'max_depth': 2, 'n_estimators': 100, 'max_features': 'sqrt', 'max_samples': 0.75}	0.1517	0.1019	1
39	gbt	COVID	{'max_depth': 3, 'max_features': 'sqrt'}	0.2200	0.0021	6
40	xgboost	COVID	{'n_estimators': 200, 'scale_pos_weight': 5, 'learning_rate': 0.2, 'gamma': 3, 'subsample': 0.8}	0.3136	0.0939	1
41	mlp	COVID	{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'solver': 'adam'}	0.3157	0.0989	1
42	elasticnet	Año	{'alpha': 0.1, 'l1_ratio': 0.25}	0.1388	0.0393	3
43	mlr	Año	{}	0.0110	0.0141	1
44	knn	Año	{'n_neighbors': 11}	0.2790	0.0537	3

	ModelNickname	Series	HyperParams	LevelTestSetRMSE	InSampleRMSE	Lags
45	rf	Año	{'max_depth': 2, 'n_estimators': 100, 'max_features': 'sqrt', 'max_samples': 0.75}	0.2825	0.0808	1
46	gbt	Año	{'max_depth': 3, 'max_features': 'sqrt'}	0.1520	0.0009	6
47	xgboost	Año	{'n_estimators': 200, 'scale_pos_weight': 5, 'learning_rate': 0.2, 'gamma': 3, 'subsample': 0.8}	0.3952	0.1140	1
48	mlp	Año	{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'solver': 'adam'}	0.5606	0.0708	1
49	elasticnet	Semana	{'alpha': 0.1, 'l1_ratio': 0.25}	0.1587	0.1192	3
50	mlr	Semana	{}	0.0525	0.0703	1
51	knn	Semana	{'n_neighbors': 11}	0.1006	0.1187	3
52	rf	Semana	{'max_depth': 2, 'n_estimators': 100, 'max_features': 'sqrt', 'max_samples': 0.75}	0.1037	0.0885	1
53	gbt	Semana	{'max_depth': 3, 'max_features': 'sqrt'}	0.0755	0.0052	6
54	xgboost	Semana	{'n_estimators': 200, 'scale_pos_weight': 5, 'learning_rate': 0.2, 'gamma': 3, 'subsample': 0.8}	0.1654	0.1399	1
55	mlp	Semana	{'activation': 'tanh', 'hidden_layer_sizes': (25,), 'solver': 'adam'}	0.3277	0.1320	1

Import a Foreign Sklearn Estimator for Ensemble Modeling

In [109…

```
estimators = [
    ('mlr',LinearRegression()),
    ('elasticnet',ElasticNet(**results.loc[results['ModelNickname'] == 'elasticnet','Hy
    ('mlp',MLPRegressor(**results.loc[results['ModelNickname'] == 'mlp','HyperParams'].
]

final_estimator = KNeighborsRegressor(**results.loc[results['ModelNickname'] == 'knn','
```

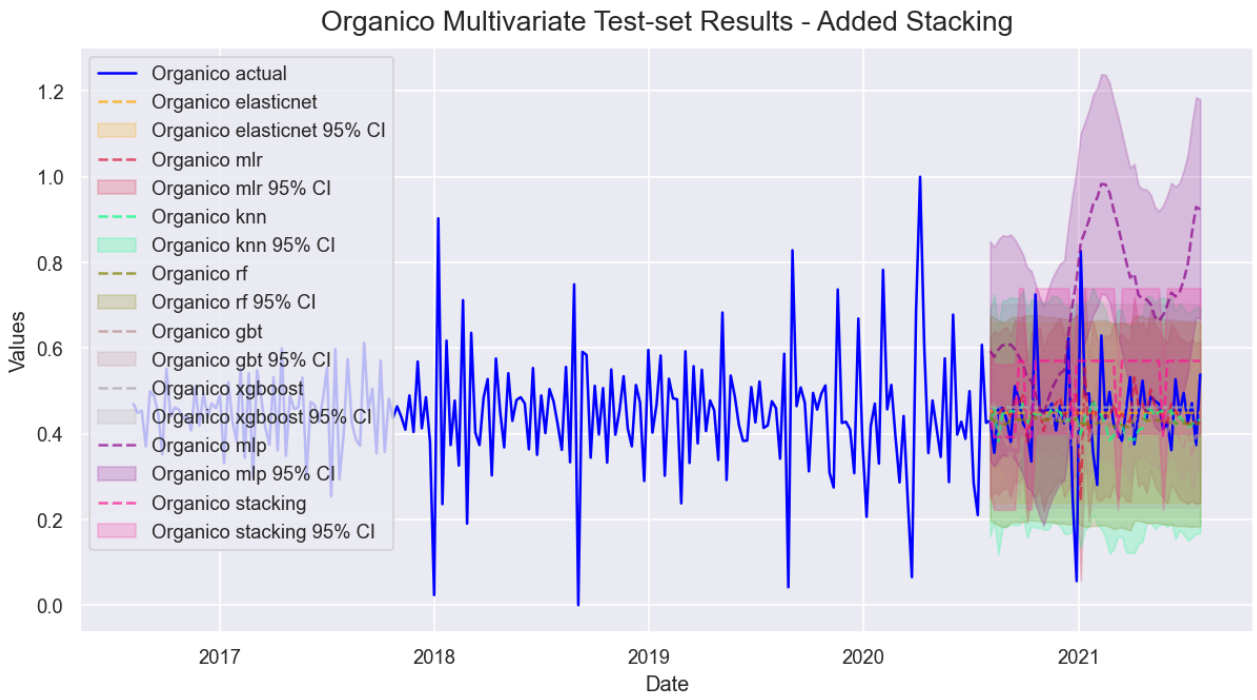
In [110…

```
mvf.add_sklearn_estimator(StackingRegressor,'stacking')
mvf.set_estimator('stacking')
mvf.manual_forecast(estimators=estimators,final_estimator=final_estimator,lags=13)
```

In [111…

```
mvf.plot_test_set(series='Organico',put_best_on_top=True,ci=True)
plt.title('Organico Multivariate Test-set Results - Added Stacking',size=16)
```

```
plt.show()
```



In [125…

```
mvf.set_best_model(determine_best_by='LevelTestSetRMSE')
results2 = mvf.export('model_summaries')
results2[
    [
        'ModelNickname',
        'Series',
        'LevelTestSetRMSE',
        'InSampleRMSE',
        'Lags',
        'best_model'
    ]
].head(8)
```

Out[125…

	ModelNickname	Series	LevelTestSetRMSE	InSampleRMSE	Lags	best_model
0	elasticnet	Organico	0.1086	0.1194	3	True
1	mlr	Organico	0.1322	0.1113	1	False
2	knn	Organico	0.1168	0.1258	3	False
3	rf	Organico	0.1116	0.1129	1	False
4	gbt	Organico	0.1249	0.0419	6	False
5	xgboost	Organico	0.1095	0.1275	1	False
6	mlp	Organico	0.3213	0.1389	1	False
7	stacking	Organico	0.1414	0.1014	13	False

In [113…

```
print('-'*100)
for series in results2['Series'].unique():
```

```
    print('multivariate average test RMSE for {}: {:.2f}'.format(series,results2.loc[re
    print('-'*100)
```

```
----------------------------------------------------------------------------------------
------------
multivariate average test RMSE for Organico: 0.15
----------------------------------------------------------------------------------------
------------
multivariate average test RMSE for Inorganico: 0.33
----------------------------------------------------------------------------------------
------------
multivariate average test RMSE for Reciclaje: 0.13
----------------------------------------------------------------------------------------
------------
multivariate average test RMSE for Habitantes: 0.27
----------------------------------------------------------------------------------------
------------
multivariate average test RMSE for Vegetal: 0.20
----------------------------------------------------------------------------------------
------------
multivariate average test RMSE for COVID: 0.16
----------------------------------------------------------------------------------------
------------
multivariate average test RMSE for Año: 0.33
----------------------------------------------------------------------------------------
------------
multivariate average test RMSE for Semana: 0.19
----------------------------------------------------------------------------------------
------------
```

Plot Final Forecasts

In [128…

```
mvf.plot(series='Organico',models='knn')#,ci=True)
plt.title('Forecast - Orgánico',size=16)
plt.show()
```