Homework 5

Question 1

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pycaret.time_series import *
import warnings
warnings.filterwarnings('ignore')
```

1. Read the csv file from the URL and set the first column in the data as the index column.

In [2]:
```
df = pd.read_csv('https://raw.githubusercontent.com/PJalgotrader/Deep_forecasting-U
df.set_index(df.columns[0], inplace=True)
df.index.name = 'date'
df.head()
```

Out[2]:

date	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbilrate	unemp	po
1959-03-31	2710.349	1707.4	286.898	470.045	1886.9	28.98	139.7	2.82	5.8	177.14
1959-06-30	2778.801	1733.7	310.859	481.301	1919.7	29.15	141.7	3.08	5.1	177.83
1959-09-30	2775.488	1751.8	289.226	491.260	1916.4	29.35	140.5	3.82	5.3	178.65
1959-12-31	2785.204	1753.7	299.356	484.052	1931.3	29.37	140.0	4.33	5.6	179.38
1960-03-31	2847.699	1770.5	331.722	462.199	1955.5	29.54	139.6	3.50	5.2	180.00

2. Before moving forward, we first need to change the data frame index type into "datetime"?

In [3]:
```
df.index = pd.to_datetime(df.index).to_period('Q')
df.head()
```

Out[3]:

date	realgdp	realcons	realinv	realgovt	realdpi	cpi	m1	tbilrate	unemp	
1959Q1	2710.349	1707.4	286.898	470.045	1886.9	28.98	139.7	2.82	5.8	177.
1959Q2	2778.801	1733.7	310.859	481.301	1919.7	29.15	141.7	3.08	5.1	177.
1959Q3	2775.488	1751.8	289.226	491.260	1916.4	29.35	140.5	3.82	5.3	178.
1959Q4	2785.204	1753.7	299.356	484.052	1931.3	29.37	140.0	4.33	5.6	179.
1960Q1	2847.699	1770.5	331.722	462.199	1955.5	29.54	139.6	3.50	5.2	180.

3. Our variable of interest is "cpi" which stands for consumer price index. Keep this variable in the data and drop the rest.

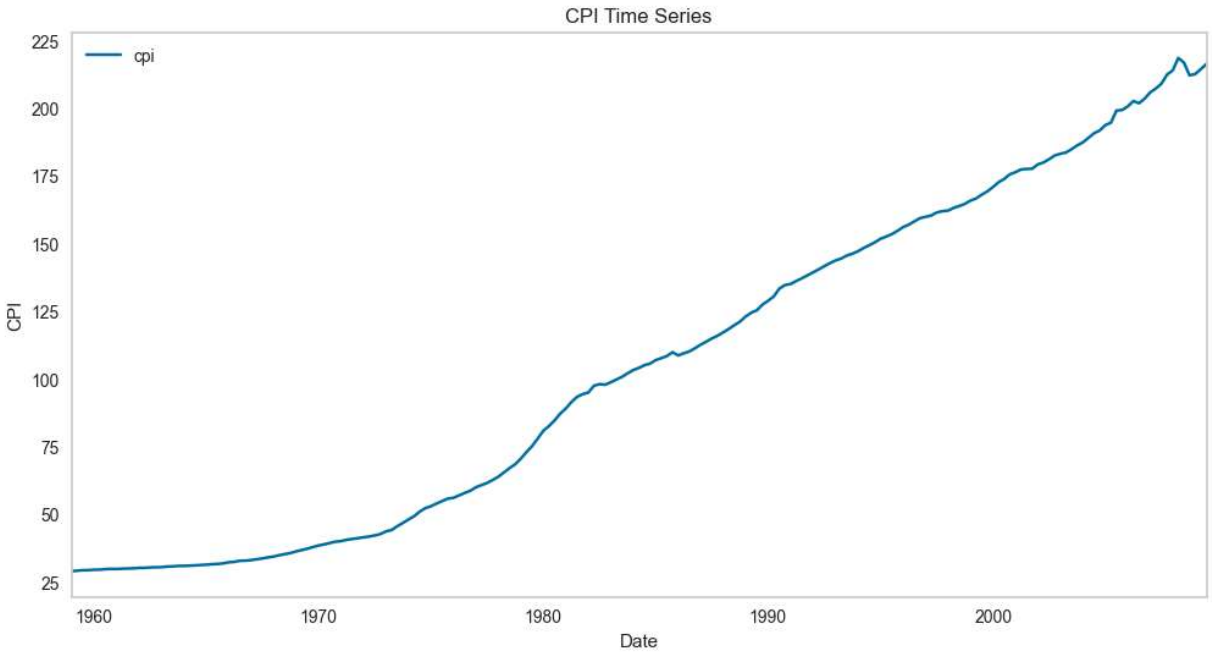
In [4]:
```
df = df[['cpi']]
df.head()
```

Out[4]:

date	cpi
1959Q1	28.98
1959Q2	29.15
1959Q3	29.35
1959Q4	29.37
1960Q1	29.54

4. Using Matplotlib, plot the time series for cpi. As you can see, the cpi data is NOT stationary. what are the implications if we apply the decision tree or random forest models from sklearn to this data without preprocessing it? preprocessing simply means, transforming, detrending, deseasonalizing the data if needed.

In [5]:
```
df.plot(figsize=(12,6))
plt.title('CPI Time Series')
plt.grid(visible=False)
plt.xlabel('Date')
plt.ylabel('CPI')
plt.show()
```



4. (Answer) - If we feed the data to a random forest model unprocessed, then because of the clear trend, it will probably perform quite poorly on the test set. If the tree is deep enough, it can perform "perfectly" on the train set, but It will most likely fail in the test set.

5. Prepare your dataset for a supervised machine learning task by adding 3 lags to the data set.

In [6]:
```
series = df['cpi'].dropna().to_numpy()

lag1 = np.roll(series, 1)
lag1[0] = np.nan

lag2 = np.roll(series, 2)
lag2[:2] = np.nan

lag3 = np.roll(series, 3)
lag3[:3] = np.nan

df['lag-1'] = lag1
df['lag-2'] = lag2
df['lag-3'] = lag3

df.head()
```

Out[6]:

date	cpi	lag-1	lag-2	lag-3
1959Q1	28.98	NaN	NaN	NaN
1959Q2	29.15	28.98	NaN	NaN
1959Q3	29.35	29.15	28.98	NaN
1959Q4	29.37	29.35	29.15	28.98
1960Q1	29.54	29.37	29.35	29.15

6. Prepare the train and test set:

- **Split the data into train and test. Keep the last 24 observations in the test set and the rest in train set. Specify your features X and target variable y. What is the shape of your df_train and df_test sets?**
- **Why can't you use train_test_split() function from sklearn here? what's wrong with that approach?**

In [7]:
```
test_size = 24
train_size = len(df) - test_size

train = df.iloc[:train_size]
test = df.iloc[train_size:]

train_indicator = (df.index <= train.index[-1])
test_indicator =  (df.index > train.index[-1])

train_indicator[:3] = False

X = np.array(df[['lag-1', 'lag-2', 'lag-3']])
Y = np.array(df[['cpi']])

Xtrain, Ytrain = X[3:-test_size], Y[3:-test_size]
Xtest, Ytest = X[-test_size:], Y[-test_size:]

df.loc[train_indicator, 'train_set'] = df.loc[train_indicator, 'cpi']
df.loc[test_indicator, 'test_set'] = df.loc[test_indicator, 'cpi']
```

In [8]:
```
print(train.shape, test.shape)
```

```
(179, 4) (24, 4)
```

6. (Answer) - The train set contains 178 observations of 3 features and one target; and the test set contains 24 observations of 3 features and one target. We can't use `train_test_split()` because it shuffles the data, which defeats the time-series purpose.

7. Train a simple decision tree model with max depth = 10 and report the MAPE in the train set? Does this mean that your model is complex enough? can the model overfit the data?

In [9]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_percentage_error

dt_model = DecisionTreeRegressor(max_depth=10)
dt_model.fit(Xtrain, Ytrain)
```

Out[9]:
```
DecisionTreeRegressor
DecisionTreeRegressor(max_depth=10)
```

In [10]:
```
preds = dt_model.predict(Xtrain)
df.loc[train_indicator, '1step_train_forecast'] = preds

mape_train = mean_absolute_percentage_error(Ytrain, preds)
print("Train Set MAPE:",mape_train)
```

```
Train Set MAPE: 0.0
```

7. (Answer) - The MAPE is 0. This means that our model has almost assuredly overfit the data, which is a good thing, as it means our model is complex enough.

8. Now we need to make some forecasts in the test set (the last 24 observations, this is what PyCaret calls hold out set).

- Why we cannot use something like the following for multi-step forecasts: y_test_pred = dt_model.predict(X_test). What's wrong with this approach?
- The correct way is to create a forecast_future function (as we did in class) and make predictions into future using the past predictions. Go ahead and create your forecast_future function and report the MAPE in the test set now.

In [11]:
```
test_preds = []

input_X = Xtest[0]

while len(test_preds) < test_size:
  prediction = dt_model.predict(input_X.reshape(1, -1))[0]
  test_preds.append(prediction)

  input_X = np.roll(input_X, -1)
  input_X[-1] = prediction

mape_test = mean_absolute_percentage_error(Ytest, test_preds)
print("Test Set MAPE:",mape_test)
```

```
df.loc[test_indicator, '1step_test_forecast'] = test_preds
df.tail()
```

```
Test Set MAPE: 0.08872469814856854
```

Out[11]:

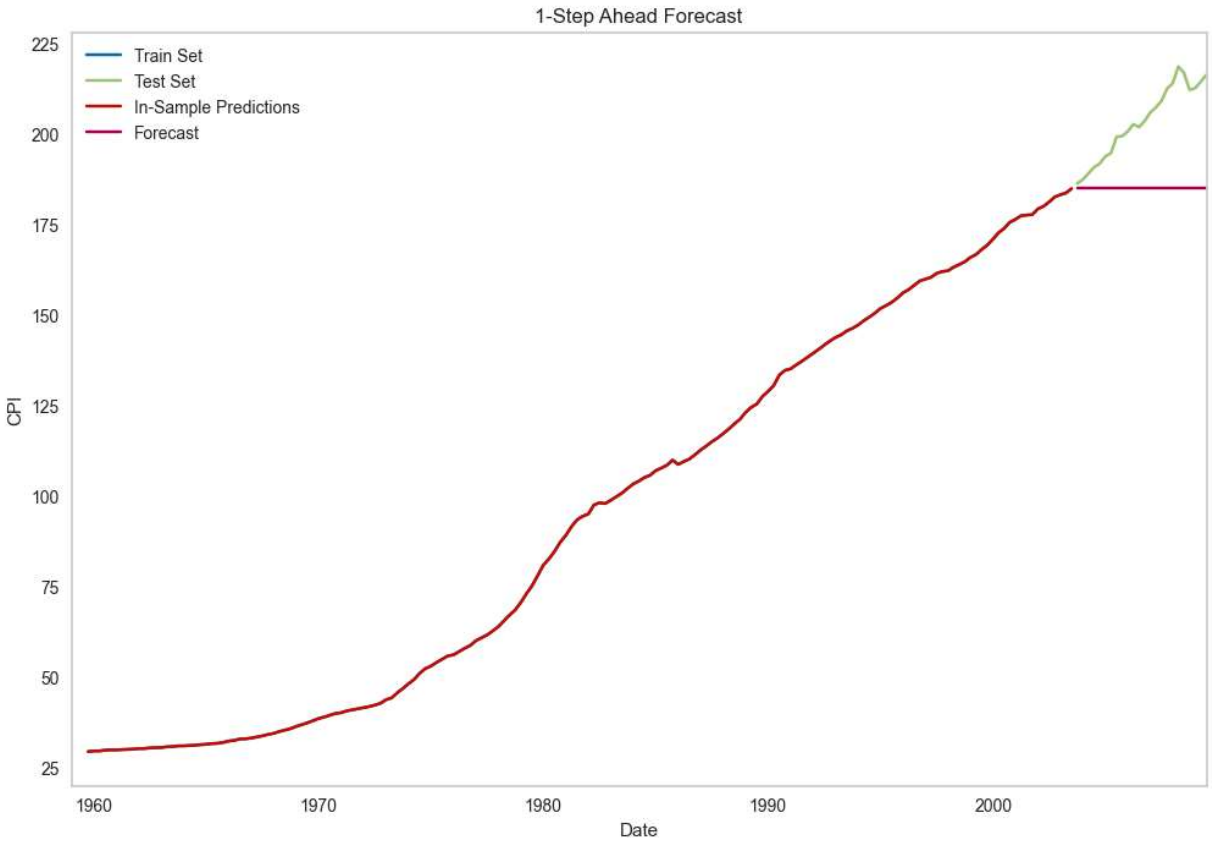
date	cpi	lag-1	lag-2	lag-3	train_set	test_set	1step_train_forecast	1step_t
2008Q3	216.889	218.610	213.997	212.495	NaN	216.889	NaN	
2008Q4	212.174	216.889	218.610	213.997	NaN	212.174	NaN	
2009Q1	212.671	212.174	216.889	218.610	NaN	212.671	NaN	
2009Q2	214.469	212.671	212.174	216.889	NaN	214.469	NaN	
2009Q3	216.385	214.469	212.671	212.174	NaN	216.385	NaN	

8. (Answer) - Similar to the reason we can't use `train_test_split()`, we can't use `model.predict()` because it doesn't account for the sequential dependencies of the time-series data.

9. Plot the actuals vs predictions for train and test set in one figure! why is your decision tree performing poorly in the test set?

In [12]:

```
df[['train_set', 'test_set', '1step_train_forecast', '1step_test_forecast']].plot(f
plt.title('1-Step Ahead Forecast')
plt.legend(['Train Set', 'Test Set', 'In-Sample Predictions', 'Forecast'])
plt.grid(visible=False)
plt.xlabel("Date")
plt.ylabel("CPI")
plt.show()
```



9. (Answer) - the models performance is abysmal because we did not preprocess it, and decision trees aren't great at handling non-stationary time-series.

P.S. Sorry I don't know why the colors are atrocious for this graph...

10. As you can see in the figure above, the performance of our machine learning model in the test set is very bad! one solution is to make the data stationary and rerun the model. How about differencing the cpi data once and apply the same decision tree model to that? what is your MAPE in the test set now? any progress?

NOTE: Follow these steps

- Go back to original dataset with cpi series. You need to import the csv file again! don't forget to change the index to datetime.
- create a variable named diff_cpi which is simply the first difference of cpi series.
- create 3 lag variables for diff_cpi. Make sure you drop na values.
- split the data into train and test. To be consistent with previous part, keep the last 24 observations for test set and the rest for the train set. Create your features X and target y.
- Train the model on the new X and y

- Make predictions for train set! Be careful! you need to undifference the data in the train set to get back to the actual levels of cpi.
- Make forecasts for the test set! this is where you need to create your forecast_future function again. Important: use the last know cpi from the train set and add the forecasted diff_cpi cumulatively to that value in order to get the level forecasts of cpi.
- Report the MAPE in the test set.

In [13]:
```
df2 = pd.read_csv('https://raw.githubusercontent.com/PJalgotrader/Deep_forecasting-
df2.set_index(df2.columns[0], inplace=True)
df2.index.name = 'date'
df2.index = pd.to_datetime(df2.index).to_period('Q')
df2 = df2[['cpi']]
```

In [14]:
```
diff_cpi = np.array(df2['cpi']) - np.roll(np.array(df2['cpi']), 1)
diff_cpi[0] = np.nan
df2['diff_cpi'] = diff_cpi
df2 = df2[1:]
df2.head()
```

Out[14]:

date	cpi	diff_cpi
1959Q2	29.15	0.17
1959Q3	29.35	0.20
1959Q4	29.37	0.02
1960Q1	29.54	0.17
1960Q2	29.55	0.01

In [15]:
```
series = df2['diff_cpi'].dropna().to_numpy()

lag1 = np.roll(series, 1)
lag1[0] = np.nan

lag2 = np.roll(series, 2)
lag2[:2] = np.nan

lag3 = np.roll(series, 3)
lag3[:3] = np.nan


df2['lag-1'] = lag1
df2['lag-2'] = lag2
df2['lag-3'] = lag3

initial_cpi = df2['cpi'][2]
df2.head()
```

Out[15]:

date	cpi	diff_cpi	lag-1	lag-2	lag-3
1959Q2	29.15	0.17	NaN	NaN	NaN
1959Q3	29.35	0.20	0.17	NaN	NaN
1959Q4	29.37	0.02	0.20	0.17	NaN
1960Q1	29.54	0.17	0.02	0.20	0.17
1960Q2	29.55	0.01	0.17	0.02	0.20

In [16]:
```
test_size = 24
train_size = len(df2) - test_size

train = df2.iloc[:train_size]
test = df2.iloc[train_size:]

train_indicator = (df2.index <= train.index[-1])
test_indicator =  (df2.index > train.index[-1])

train_indicator[:3] = False

X = np.array(df2[['lag-1', 'lag-2', 'lag-3']])
Y = np.array(df2[['diff_cpi']])

Xtrain, Ytrain = X[3:-test_size], Y[3:-test_size]
Xtest, Ytest = X[-test_size:], Y[-test_size:]

df2.loc[train_indicator, 'train_set'] = df2.loc[train_indicator, 'cpi']
df2.loc[test_indicator, 'test_set'] = df2.loc[test_indicator, 'cpi']
```

In [17]:
```
dt_model = DecisionTreeRegressor(max_depth=10)
dt_model.fit(Xtrain, Ytrain)
```

Out[17]:

▾ DecisionTreeRegressor

```
DecisionTreeRegressor(max_depth=10)
```

In [18]:
```
preds = dt_model.predict(Xtrain)

pred_cpi = []
for i in range(len(preds)):
    prediction = df2['cpi'][i+2] + preds[i]
    pred_cpi.append(prediction)

df2.loc[train_indicator, '1step_train_forecast'] = pred_cpi

mape_train = mean_absolute_percentage_error(df2.loc[train_indicator,'cpi'], df2.loc
print("Train Set MAPE:",mape_train)

df2.head()
```

```
Train Set MAPE: 0.0004368547212117046
```

Out[18]:

date	cpi	diff_cpi	lag-1	lag-2	lag-3	train_set	test_set	1step_train_forecast
1959Q2	29.15	0.17	NaN	NaN	NaN	NaN	NaN	NaN
1959Q3	29.35	0.20	0.17	NaN	NaN	NaN	NaN	NaN
1959Q4	29.37	0.02	0.20	0.17	NaN	NaN	NaN	NaN
1960Q1	29.54	0.17	0.02	0.20	0.17	29.54	NaN	29.49
1960Q2	29.55	0.01	0.17	0.02	0.20	29.55	NaN	29.55

In [19]:

```
test_preds = []

input_X = Xtest[0]

while len(test_preds) < test_size:
  prediction = dt_model.predict(input_X.reshape(1, -1))[0]
  test_preds.append(prediction)

  input_X = np.roll(input_X, -1)
  input_X[-1] = prediction

initial_prediction = test_preds[0] + df2['cpi'][train_size]
test_pred_cpi = [initial_prediction]
for i in range(1,len(test_preds)):
    prediction = test_pred_cpi[i-1] + test_preds[i]
    test_pred_cpi.append(prediction)

df2.loc[test_indicator, '1step_test_forecast'] = test_pred_cpi
df2.tail()
```

Out[19]:

date	cpi	diff_cpi	lag-1	lag-2	lag-3	train_set	test_set	1step_train_forecast	1s
2008Q3	216.889	-1.721	4.613	1.502	3.362	NaN	216.889	NaN	
2008Q4	212.174	-4.715	-1.721	4.613	1.502	NaN	212.174	NaN	
2009Q1	212.671	0.497	-4.715	-1.721	4.613	NaN	212.671	NaN	
2009Q2	214.469	1.798	0.497	-4.715	-1.721	NaN	214.469	NaN	
2009Q3	216.385	1.916	1.798	0.497	-4.715	NaN	216.385	NaN	

In [20]:

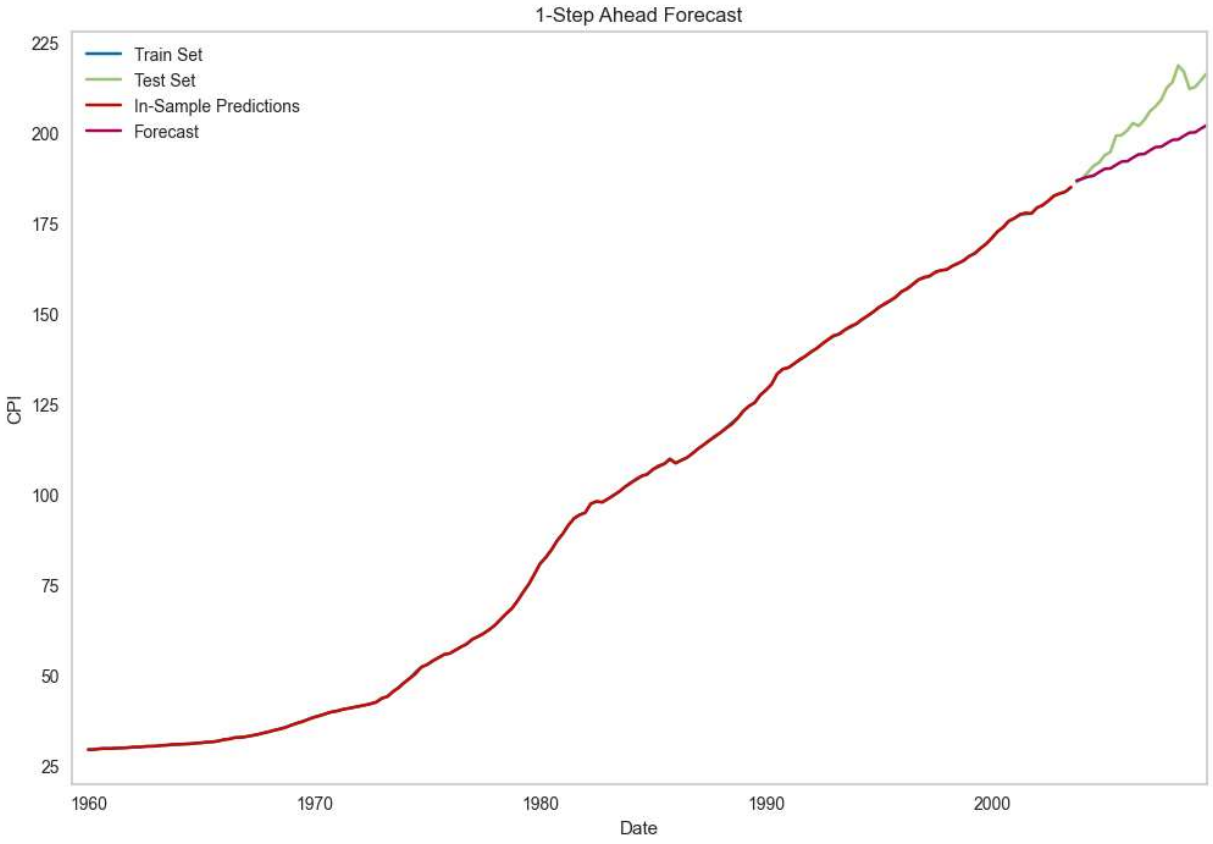
```
mape_test = mean_absolute_percentage_error(df2.loc[test_indicator,'cpi'], df2.loc[t
print("Test Set MAPE:",mape_test)
```

```
Test Set MAPE: 0.04448819604338389
```

10. (Answer) - The test-set MAPE is now 0.0445 compared to the 0.0889 previously. A 50% decrease in MAPE is not negligible, but there's still a ways to go.

11. Great job, now visualize the actuals vs predictions both in the train and test set!

In [21]:
```
df2[['train_set', 'test_set', '1step_train_forecast', '1step_test_forecast']].plot(
plt.title('1-Step Ahead Forecast')
plt.legend(['Train Set', 'Test Set', 'In-Sample Predictions', 'Forecast'])
plt.grid(visible=False)
plt.xlabel("Date")
plt.ylabel("CPI")
plt.show()
```



12. You tried so hard to get here, now you deserve to sit back and enjoy doing all the econometrics and ML models in PyCaret. Pycaret is taking care of everything (I mean everything you did above) in couple of lines :)

- set up your pycaret experiment: Hint: exp = setup(data = df, target = 'cpi', fh = 24, session_id=1000)
- run a horse race between all the timeseries models in pycaret. Make sure to set the cross validation = False, to get a fair comparison in the test set only. Hint:

exp.compare_models(cross_validation=False)

- Which family of models are winning this competition? Econometrics models or ML models? :)

In [22]:
```
from pycaret.time_series import *
exp = TSForecastingExperiment()

df3 = pd.read_csv('https://raw.githubusercontent.com/PJalgotrader/Deep_forecasting-
df3.set_index(df3.columns[0], inplace=True)
df3.index.name = 'date'
df3.index = pd.to_datetime(df3.index).to_period('Q')
df3 = df3[['cpi']]

exp.setup(data=df3, target='cpi', fh=24, session_id=1000)
```

	Description	Value
0	session_id	1000
1	Target	cpi
2	Approach	Univariate
3	Exogenous Variables	Not Present
4	Original data shape	(203, 1)
5	Transformed data shape	(203, 1)
6	Transformed train set shape	(179, 1)
7	Transformed test set shape	(24, 1)
8	Rows with missing values	0.0%
9	Fold Generator	ExpandingWindowSplitter
10	Fold Number	3
11	Enforce Prediction Interval	False
12	Splits used for hyperparameters	all
13	User Defined Seasonal Period(s)	None
14	Ignore Seasonality Test	False
15	Seasonality Detection Algo	auto
16	Max Period to Consider	60
17	Seasonal Period(s) Tested	[3, 11]
18	Significant Seasonal Period(s)	[3, 11]
19	Significant Seasonal Period(s) without Harmonics	[3, 11]
20	Remove Harmonics	False
21	Harmonics Order Method	harmonic_max
22	Num Seasonalities to Use	1
23	All Seasonalities to Use	[3]
24	Primary Seasonality	3
25	Seasonality Present	True
26	Seasonality Type	mul
27	Target Strictly Positive	True
28	Target White Noise	No
29	Recommended d	2

	Description	Value
30	Recommended Seasonal D	0
31	Preprocess	False
32	CPU Jobs	-1
33	Use GPU	False
34	Log Experiment	False
35	Experiment Name	ts-default-name
36	USI	c506

Out[22]: `<pycaret.time_series.forecasting.oop.TSForecastingExperiment at 0x1e24c3ab2d0>`

In [23]:
```
exp.compare_models()
```

	Model	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
exp_smooth	Exponential Smoothing	0.7522	0.7403	1.7975	2.2702	0.0125	0.0127	0.9076
ets	ETS	0.7522	0.7403	1.7975	2.2702	0.0125	0.0127	0.9076
auto_arima	Auto ARIMA	0.9251	0.9041	2.2362	2.7871	0.0152	0.0154	0.8711
rf_cds_dt	Random Forest w/ Cond. Deseasonalize & Detrending	0.9502	0.9228	2.2502	2.8269	0.0161	0.0164	0.8528
et_cds_dt	Extra Trees w/ Cond. Deseasonalize & Detrending	0.9620	0.9382	2.2793	2.8741	0.0163	0.0166	0.8502
lightgbm_cds_dt	Light Gradient Boosting w/ Cond. Deseasonalize & Detrending	0.9815	0.9732	2.3141	2.9745	0.0166	0.0169	0.8342
catboost_cds_dt	CatBoost Regressor w/ Cond. Deseasonalize & Detrending	1.0439	0.9959	2.4515	3.0415	0.0179	0.0183	0.8191
gbr_cds_dt	Gradient Boosting w/ Cond. Deseasonalize & Detrending	1.0713	1.0197	2.5238	3.1180	0.0183	0.0187	0.8134
knn_cds_dt	K Neighbors w/ Cond. Deseasonalize & Detrending	1.1903	1.1182	2.8530	3.4466	0.0201	0.0205	0.7807
stlf	STLF	1.1905	1.1485	2.8587	3.5379	0.0198	0.0202	0.7952
ada_cds_dt	AdaBoost w/ Cond. Deseasonalize & Detrending	1.1906	1.1035	2.8189	3.3814	0.0203	0.0207	0.7859
arima	ARIMA	1.2909	1.2499	3.1621	3.8817	0.0210	0.0213	0.7542
omp_cds_dt	Orthogonal Matching Pursuit w/ Cond.	1.8327	1.7378	4.4530	5.3855	0.0302	0.0309	0.5198

	Model	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
	Deseasonalize & Detrending							
dt_cds_dt	Decision Tree w/ Cond. Deseasonalize & Detrending	1.8358	1.7992	4.5511	5.5955	0.0289	0.0297	0.3167
xgboost_cds_dt	Extreme Gradient Boosting w/ Cond. Deseasonalize & Detrending	1.8466	1.6702	4.2797	5.0730	0.0323	0.0339	0.3975
huber_cds_dt	Huber w/ Cond. Deseasonalize & Detrending	2.0607	2.0241	4.9098	6.2196	0.0344	0.0358	0.3470
en_cds_dt	Elastic Net w/ Cond. Deseasonalize & Detrending	2.0774	1.9632	5.0190	6.0701	0.0344	0.0354	0.3993
lr_cds_dt	Linear w/ Cond. Deseasonalize & Detrending	2.1099	2.0541	5.0624	6.3322	0.0351	0.0363	0.3431
br_cds_dt	Bayesian Ridge w/ Cond. Deseasonalize & Detrending	2.1125	2.0558	5.0690	6.3376	0.0351	0.0363	0.3423
ridge_cds_dt	Ridge w/ Cond. Deseasonalize & Detrending	2.1152	2.0558	5.0763	6.3381	0.0351	0.0364	0.3430
llar_cds_dt	Lasso Least Angular Regressor w/ Cond. Deseasonalize & Detrending	2.1397	2.0189	5.1706	6.2430	0.0354	0.0365	0.3642
lasso_cds_dt	Lasso w/ Cond. Deseasonalize & Detrending	2.1397	2.0189	5.1706	6.2430	0.0354	0.0365	0.3642
theta	Theta Forecaster	2.8126	2.6918	6.8287	8.3383	0.0461	0.0479	-0.1126

	Model	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
naive	Naive Forecaster	5.1972	4.8216	12.6986	14.9762	0.0849	0.0903	-2.6703
polytrend	Polynomial Trend Forecaster	5.3190	4.2051	12.8238	13.0095	0.0914	0.0967	-1.8639
snaive	Seasonal Naive Forecaster	5.5629	5.0617	13.5986	15.7255	0.0911	0.0971	-3.0611
croston	Croston	9.3512	7.7892	22.8656	24.2165	0.1553	0.1703	-8.7837
grand_means	Grand Means Forecaster	33.2275	26.4112	81.9687	82.3747	0.5516	0.7625	-119.2732

Out[23]:
```
ExponentialSmoothing(seasonal='mul', sp=3, trend='add')
```

12. (Answer) - The Econometrics models are winning this race here, Which is surprising to me, since The cpi doesn't seem to exhibit a well-behaved tendency.

Question 2

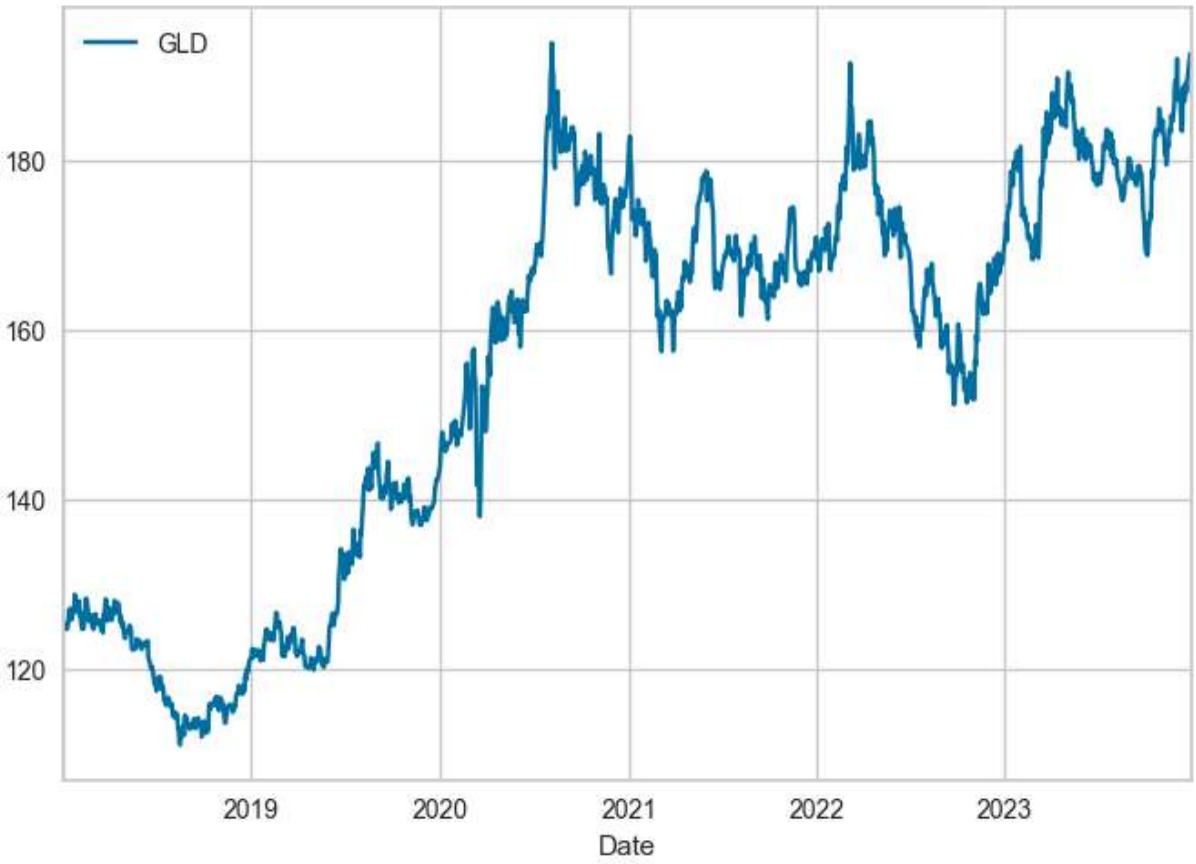
In [24]:
```
stocks = pd.read_csv('https://raw.githubusercontent.com/PJalgotrader/Deep_forecasti
stocks.index = pd.to_datetime(stocks.index).to_period('B')
stocks.head()
```

Out[24]:

	Adj Close							
	AAPL	GLD	MSFT	QQQ	SPY	TSLA	USO	
Date								
2018-01-02	40.722874	125.150002	80.229012	152.072800	243.072266	21.368668	96.559998	43.(
2018-01-03	40.715786	124.820000	80.602394	153.550400	244.609711	21.150000	98.720001	43.(
2018-01-04	40.904907	125.459999	81.311806	153.819046	245.640732	20.974667	98.959999	43.2
2018-01-05	41.370617	125.330002	82.319908	155.363861	247.277679	21.105333	98.480003	43.1
2018-01-08	41.216949	125.309998	82.403923	155.968399	247.729935	22.427334	99.040001	43.5

5 rows × 42 columns

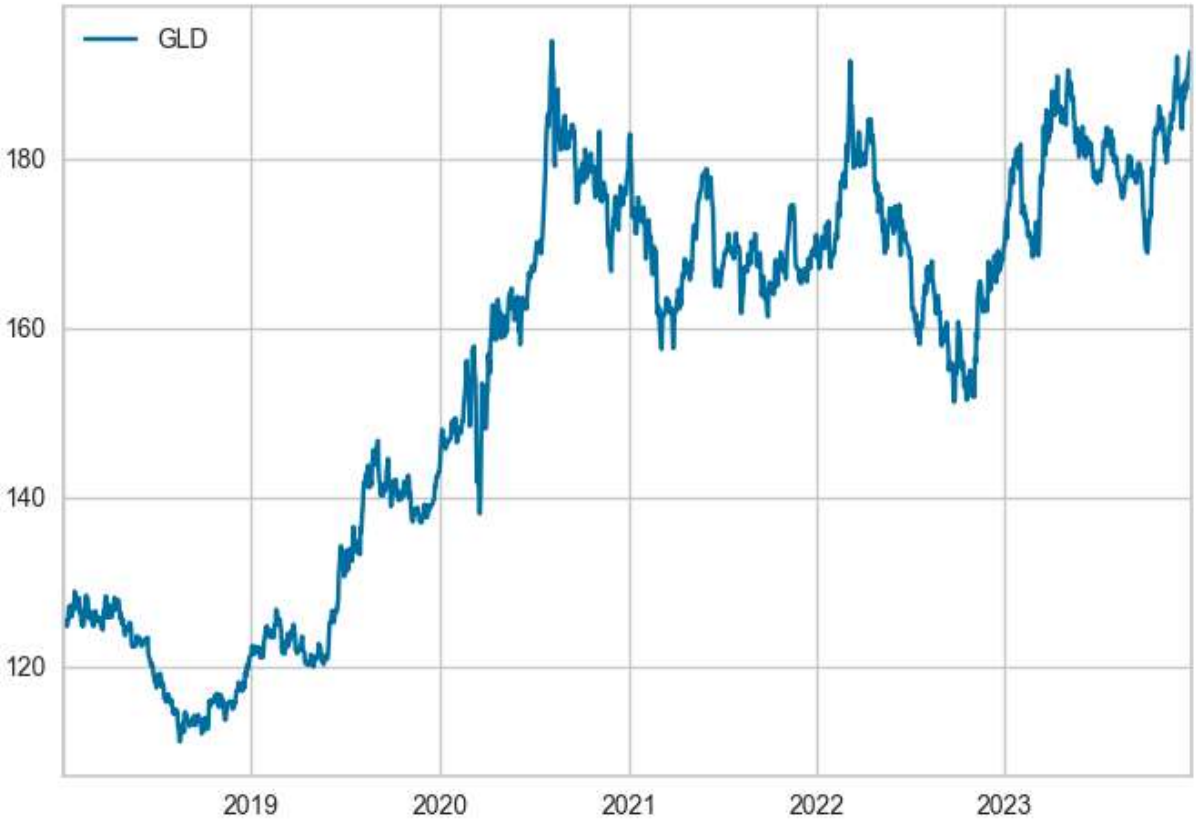
In [25]:
```
df = stocks['Close'][['GLD']]
df.plot()
plt.show()
```



In [26]:
```
idx = pd.period_range(min(df.index), max(df.index))
df = df.reindex(idx, fill_value=np.nan)
```

```
df = df.fillna(method = 'ffill')

df.plot()
plt.show()
```



In [27]:
```
from sktime.forecasting.model_selection import SlidingWindowSplitter

exp = TSForecastingExperiment()
exp.setup(data=df, target='GLD', coverage=0.9, fold_strategy=SlidingWindowSplitter(
```

	Description	Value
0	session_id	1000
1	Target	GLD
2	Approach	Univariate
3	Exogenous Variables	Not Present
4	Original data shape	(1564, 1)
5	Transformed data shape	(1564, 1)
6	Transformed train set shape	(1542, 1)
7	Transformed test set shape	(22, 1)
8	Rows with missing values	0.0%
9	Fold Generator	SlidingWindowSplitter
10	Fold Number	11
11	Enforce Prediction Interval	False
12	Splits used for hyperparameters	all
13	User Defined Seasonal Period(s)	None
14	Ignore Seasonality Test	False
15	Seasonality Detection Algo	auto
16	Max Period to Consider	60
17	Seasonal Period(s) Tested	[]
18	Significant Seasonal Period(s)	[1]
19	Significant Seasonal Period(s) without Harmonics	[1]
20	Remove Harmonics	False
21	Harmonics Order Method	harmonic_max
22	Num Seasonalities to Use	1
23	All Seasonalities to Use	[1]
24	Primary Seasonality	1
25	Seasonality Present	False
26	Seasonality Type	None
27	Target Strictly Positive	True
28	Target White Noise	No
29	Recommended d	1

	Description	Value
30	Recommended Seasonal D	0
31	Preprocess	False
32	CPU Jobs	-1
33	Use GPU	False
34	Log Experiment	False
35	Experiment Name	ts-default-name
36	USI	7566

Out[27]: <pycaret.time_series.forecasting.oop.TSForecastingExperiment at 0x1e25293d150>

In [28]:
```
exp.plot_model(plot='train_test_split')
```

In [29]:
```
exp.plot_model(plot='cv')
```

In [30]:
```
exp.compare_models(sort='mape')
```

	Model	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2	TT (Se
exp_smooth	Exponential Smoothing	3.0319	2.6232	3.1693	3.7344	0.0194	0.0195	-1.6915	0.0
ets	ETS	3.0318	2.6231	3.1692	3.7343	0.0194	0.0195	-1.6914	0.0
ada_cds_dt	AdaBoost w/ Cond. Deseasonalize & Detrending	3.0666	2.6587	3.2347	3.7977	0.0199	0.0200	-1.6574	0.0
knn_cds_dt	K Neighbors w/ Cond. Deseasonalize & Detrending	3.0652	2.6556	3.2483	3.8020	0.0199	0.0200	-1.6534	0.0
theta	Theta Forecaster	3.2032	2.7730	3.2713	3.8717	0.0202	0.0204	-1.9947	0.0
catboost_cds_dt	CatBoost Regressor w/ Cond. Deseasonalize & Detrending	3.2395	2.7754	3.3807	3.9255	0.0209	0.0209	-1.9513	0.1
naive	Naive Forecaster	3.3667	2.9112	3.4404	4.0639	0.0213	0.0214	-2.2461	0.0
gbr_cds_dt	Gradient Boosting w/ Cond. Deseasonalize & Detrending	3.3137	2.8262	3.4507	3.9862	0.0213	0.0213	-2.0551	0.0
rf_cds_dt	Random Forest w/ Cond. Deseasonalize & Detrending	3.3526	2.8553	3.5440	4.0938	0.0218	0.0219	-2.0685	0.0
lightgbm_cds_dt	Light Gradient Boosting w/ Cond. Deseasonalize & Detrending	3.6655	3.0773	3.5129	4.0626	0.0222	0.0223	-2.7846	0.0
auto_arima	Auto ARIMA	3.6746	3.1494	3.7138	4.3593	0.0229	0.0232	-2.8283	0.0
et_cds_dt	Extra Trees w/ Cond. Deseasonalize & Detrending	3.6528	3.2189	3.9002	4.6370	0.0239	0.0238	-3.0192	0.0
dt_cds_dt	Decision Tree w/ Cond.	3.9846	3.5037	3.9812	4.7737	0.0247	0.0247	-4.7867	0.0

	Model	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2	TT (Se
	Deseasonalize & Detrending								
huber_cds_dt	Huber w/ Cond. Deseasonalize & Detrending	4.1262	3.5090	4.0200	4.6627	0.0253	0.0254	-3.7001	0.0
xgboost_cds_dt	Extreme Gradient Boosting w/ Cond. Deseasonalize & Detrending	4.4955	3.6915	4.2303	4.8104	0.0269	0.0271	-6.9093	0.0
lr_cds_dt	Linear w/ Cond. Deseasonalize & Detrending	4.4622	3.7682	4.2640	4.9127	0.0270	0.0272	-4.5780	0.0
omp_cds_dt	Orthogonal Matching Pursuit w/ Cond. Deseasonalize & Detrending	4.4622	3.7682	4.2640	4.9127	0.0270	0.0272	-4.5780	0.0
ridge_cds_dt	Ridge w/ Cond. Deseasonalize & Detrending	4.4859	3.7866	4.2787	4.9282	0.0271	0.0273	-4.6521	0.0
br_cds_dt	Bayesian Ridge w/ Cond. Deseasonalize & Detrending	4.4826	3.7840	4.2801	4.9292	0.0271	0.0273	-4.6328	0.0
arima	ARIMA	4.6307	3.9315	4.4937	5.2120	0.0283	0.0286	-5.1174	0.0
croston	Croston	4.9803	4.0295	4.6511	5.1805	0.0298	0.0301	-5.8446	0.0
en_cds_dt	Elastic Net w/ Cond. Deseasonalize & Detrending	5.8013	4.7022	5.1757	5.7867	0.0335	0.0338	-9.3254	0.0
lasso_cds_dt	Lasso w/ Cond. Deseasonalize & Detrending	5.8605	4.7370	5.2152	5.8196	0.0338	0.0341	-9.5368	0.0
llar_cds_dt	Lasso Least Angular Regressor w/ Cond.	5.8605	4.7370	5.2152	5.8196	0.0338	0.0341	-9.5368	0.0

	Model	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2	TT (Se
	Deseasonalize & Detrending								
polytrend	Polynomial Trend Forecaster	6.8180	5.3384	6.1283	6.5938	0.0396	0.0398	-12.6470	0.0
grand_means	Grand Means Forecaster	8.6614	6.6135	7.6728	8.0364	0.0503	0.0511	-20.4265	0.0

Out[30]:

```
ExponentialSmoothing(sp=1, trend='add')
```

In [31]:

```
ets = exp.create_model('ets')
tuned_ets = exp.tune_model(ets)
```

	cutoff	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
0	2018-07-02	1.5816	1.3510	0.8669	1.0065	0.0074	0.0074	0.4040
1	2018-12-31	1.6805	1.6289	0.8379	1.1218	0.0069	0.0068	-0.4363
2	2019-07-01	3.7149	2.9675	2.1653	2.3836	0.0161	0.0163	-3.5415
3	2019-12-30	3.6263	2.9973	3.0203	3.2648	0.0205	0.0208	-3.5504
4	2020-06-29	2.5582	2.9022	3.6059	5.7131	0.0202	0.0207	-0.0705
5	2020-12-28	2.4066	1.8167	3.3908	3.7243	0.0193	0.0193	-0.2497
6	2021-06-28	2.6589	2.0969	3.0306	3.3309	0.0179	0.0181	-4.0326
7	2021-12-27	1.1775	1.0484	1.0728	1.3192	0.0063	0.0063	0.1628
8	2022-06-27	5.8564	5.0079	7.4181	8.2954	0.0461	0.0448	-4.1345
9	2022-12-26	6.6821	5.7871	7.7305	8.8817	0.0436	0.0449	-3.3471
10	2023-06-26	1.4070	1.2503	1.7226	2.0364	0.0095	0.0095	0.1907
Mean	NaT	3.0318	2.6231	3.1692	3.7343	0.0194	0.0195	-1.6914
SD	NaT	1.7290	1.4750	2.2837	2.6355	0.0131	0.0131	1.8762

	cutoff	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
0	2018-07-02	1.5853	1.3531	0.8690	1.0080	0.0074	0.0074	0.4022
1	2018-12-31	1.5545	1.5045	0.7751	1.0362	0.0064	0.0063	-0.2253
2	2019-07-01	3.6759	2.9416	2.1426	2.3627	0.0160	0.0161	-3.4625
3	2019-12-30	3.5069	2.9028	2.9208	3.1619	0.0199	0.0201	-3.2680
4	2020-06-29	2.3797	2.7536	3.3543	5.4207	0.0187	0.0192	0.0363
5	2020-12-28	2.5615	1.9151	3.6091	3.9261	0.0206	0.0205	-0.3888
6	2021-06-28	2.4975	1.9678	2.8466	3.1259	0.0168	0.0170	-3.4321
7	2021-12-27	1.1665	1.0221	1.0626	1.2861	0.0063	0.0063	0.2042
8	2022-06-27	5.9792	5.1115	7.5736	8.4669	0.0471	0.0458	-4.3490
9	2022-12-26	6.5175	5.6463	7.5401	8.6656	0.0426	0.0438	-3.1381
10	2023-06-26	1.3146	1.1417	1.6095	1.8596	0.0089	0.0089	0.3251
Mean	NaT	2.9763	2.5691	3.1185	3.6654	0.0191	0.0192	-1.5723
SD	NaT	1.7341	1.4786	2.2926	2.6344	0.0132	0.0132	1.8222

```
Fitting 11 folds for each of 6 candidates, totalling 66 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done  66 out of  66 | elapsed:    0.2s finished
```

ETS once again is the winner. Interestingly enough, KNN untuned was only 0.005 off on MAPE. but ETS is definitely the winner, barely edging out the random walk. Sheesh. Forecasting really is hard

In [32]:
```
arima = exp.create_model('arima')
tuned_arima = exp.tune_model(arima)
```

	cutoff	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
0	2018-07-02	4.5214	3.9015	2.4784	2.9065	0.0213	0.0210	-3.9705
1	2018-12-31	4.1656	3.5951	2.0770	2.4759	0.0169	0.0172	-5.9958
2	2019-07-01	7.6863	5.9231	4.4801	4.7576	0.0334	0.0341	-17.0936
3	2019-12-30	6.2860	5.1649	5.2354	5.6259	0.0356	0.0364	-12.5117
4	2020-06-29	6.1459	5.6126	8.6629	11.0488	0.0491	0.0511	-3.0038
5	2020-12-28	2.1975	1.6986	3.0962	3.4823	0.0176	0.0176	-0.0925
6	2021-06-28	1.0295	0.9080	1.1734	1.4424	0.0069	0.0070	0.0564
7	2021-12-27	2.1859	1.9413	1.9914	2.4427	0.0117	0.0118	-1.8706
8	2022-06-27	6.9066	5.8758	8.7484	9.7330	0.0544	0.0526	-6.0683
9	2022-12-26	7.8128	6.7172	9.0387	10.3092	0.0511	0.0528	-4.8567
10	2023-06-26	2.0001	1.9078	2.4488	3.1074	0.0135	0.0136	-0.8844
Mean	NaT	4.6307	3.9315	4.4937	5.2120	0.0283	0.0286	-5.1174
SD	NaT	2.3682	1.9576	2.8606	3.3435	0.0164	0.0166	5.1000

	cutoff	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
0	2018-07-02	2.2803	1.8398	1.2499	1.3706	0.0107	0.0107	-0.1053
1	2018-12-31	1.6066	1.7107	0.8011	1.1781	0.0065	0.0066	-0.5840
2	2019-07-01	6.7626	5.1719	3.9417	4.1542	0.0294	0.0299	-12.7950
3	2019-12-30	4.5429	3.7341	3.7837	4.0674	0.0257	0.0261	-6.0626
4	2020-06-29	3.8479	3.9233	5.4237	7.7233	0.0305	0.0315	-0.9563
5	2020-12-28	2.0932	1.6423	2.9494	3.3669	0.0168	0.0168	-0.0214
6	2021-06-28	1.7920	1.4241	2.0425	2.2622	0.0121	0.0122	-1.3213
7	2021-12-27	1.2929	1.1419	1.1779	1.4369	0.0069	0.0069	0.0067
8	2022-06-27	5.7965	4.9444	7.3422	8.1902	0.0457	0.0444	-4.0050
9	2022-12-26	6.7679	5.8287	7.8297	8.9456	0.0442	0.0455	-3.4098
10	2023-06-26	1.9561	1.8744	2.3950	3.0529	0.0132	0.0133	-0.8189
Mean	NaT	3.5217	3.0214	3.5397	4.1589	0.0220	0.0222	-2.7339
SD	NaT	2.0231	1.6498	2.3131	2.7202	0.0135	0.0135	3.6878

```
Fitting 11 folds for each of 10 candidates, totalling 110 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 110 out of 110 | elapsed:    0.5s finished
```

Arima can't keep up with ETS, and is actually the worst of the four models considered.

In [33]:
```
xgboost = exp.create_model('xgboost_cds_dt', window_length=7)
tuned_xgboost = exp.tune_model(xgboost)
```

	cutoff	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
0	2018-07-02	2.2318	1.8539	1.2233	1.3811	0.0105	0.0104	-0.1223
1	2018-12-31	1.8546	1.8336	0.9247	1.2628	0.0075	0.0076	-0.8198
2	2019-07-01	13.5976	10.0204	7.9256	8.0487	0.0593	0.0611	-50.7839
3	2019-12-30	2.2938	2.1022	1.9104	2.2898	0.0130	0.0131	-1.2384
4	2020-06-29	2.8702	3.0720	4.0456	6.0475	0.0227	0.0233	-0.1995
5	2020-12-28	2.4259	2.0137	3.4181	4.1284	0.0194	0.0194	-0.5356
6	2021-06-28	2.1022	1.6822	2.3960	2.6721	0.0142	0.0143	-2.2387
7	2021-12-27	1.5780	1.5066	1.4376	1.8957	0.0084	0.0085	-0.7290
8	2022-06-27	7.4038	6.2956	9.3782	10.4284	0.0583	0.0563	-7.1145
9	2022-12-26	6.2295	5.4484	7.2070	8.3619	0.0407	0.0418	-2.8531
10	2023-06-26	1.6945	1.4980	2.0747	2.4398	0.0115	0.0114	-0.1617
Mean	NaT	4.0256	3.3933	3.8128	4.4506	0.0241	0.0243	-6.0724
SD	NaT	3.5399	2.6133	2.8437	3.0840	0.0186	0.0187	14.2710

	cutoff	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
0	2018-07-02	0.9464	0.8427	0.5188	0.6278	0.0044	0.0044	0.7681
1	2018-12-31	2.1919	2.0501	1.0929	1.4119	0.0090	0.0089	-1.2751
2	2019-07-01	2.7140	2.4028	1.5819	1.9300	0.0118	0.0119	-1.9774
3	2019-12-30	10.7844	8.6607	8.9820	9.4337	0.0611	0.0633	-36.9918
4	2020-06-29	2.4262	2.7950	3.4198	5.5021	0.0191	0.0196	0.0071
5	2020-12-28	3.6011	2.7790	5.0739	5.6972	0.0287	0.0293	-1.9244
6	2021-06-28	7.8016	5.8591	8.8920	9.3072	0.0527	0.0513	-38.2917
7	2021-12-27	1.1372	0.9837	1.0360	1.2378	0.0061	0.0061	0.2629
8	2022-06-27	1.3991	1.3062	1.7722	2.1637	0.0109	0.0109	0.6507
9	2022-12-26	2.7846	2.3838	3.2216	3.6585	0.0182	0.0184	0.2624
10	2023-06-26	2.8938	2.7425	3.5430	4.4668	0.0195	0.0198	-2.8939
Mean	NaT	3.5164	2.9823	3.5577	4.1306	0.0220	0.0222	-7.4003
SD	NaT	2.8991	2.2074	2.8456	2.9593	0.0179	0.0181	14.3051

```
Fitting 11 folds for each of 10 candidates, totalling 110 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 110 out of 110 | elapsed:    2.9s finished
```

xgboost just can't quite hang with the best of the econometrics models, shame... It does however outperform the arima models, so that's a plus.

I played around with the window_length and found that a value of 7 produced the lowest MAPE, but it still doesn't compete with ETS

In [34]:
```
knn = exp.create_model('knn_cds_dt')
tuned_knn = exp.tune_model(knn)
```

	cutoff	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
0	2018-07-02	2.1126	1.8025	1.1580	1.3428	0.0099	0.0099	-0.0609
1	2018-12-31	1.8561	1.8082	0.9255	1.2453	0.0075	0.0076	-0.7698
2	2019-07-01	1.8907	1.6605	1.1021	1.3337	0.0082	0.0083	-0.4220
3	2019-12-30	4.4149	3.6178	3.6770	3.9406	0.0250	0.0254	-5.6293
4	2020-06-29	2.8548	3.0919	4.0239	6.0866	0.0226	0.0232	-0.2150
5	2020-12-28	2.2106	1.6934	3.1148	3.4717	0.0178	0.0177	-0.0859
6	2021-06-28	2.1211	1.6775	2.4176	2.6648	0.0143	0.0144	-2.2210
7	2021-12-27	1.2968	1.1752	1.1814	1.4788	0.0070	0.0070	-0.0521
8	2022-06-27	6.7049	5.6492	8.4928	9.3577	0.0528	0.0511	-5.5337
9	2022-12-26	6.9580	5.9338	8.0497	9.1069	0.0455	0.0468	-3.5703
10	2023-06-26	1.2970	1.1009	1.5880	1.7932	0.0088	0.0088	0.3725
Mean	NaT	3.0652	2.6556	3.2483	3.8020	0.0199	0.0200	-1.6534
SD	NaT	1.9515	1.6446	2.5841	2.9195	0.0150	0.0149	2.1509

	cutoff	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
0	2018-07-02	3.9733	3.3234	2.1779	2.4759	0.0187	0.0185	-2.6066
1	2018-12-31	2.4714	2.2028	1.2322	1.5171	0.0100	0.0101	-1.6265
2	2019-07-01	8.9429	6.6369	5.2126	5.3309	0.0389	0.0398	-21.7172
3	2019-12-30	4.1006	3.4002	3.4153	3.7037	0.0232	0.0235	-4.8560
4	2020-06-29	2.5374	2.9067	3.5766	5.7221	0.0200	0.0205	-0.0739
5	2020-12-28	2.0341	1.5984	2.8661	3.2770	0.0163	0.0163	0.0325
6	2021-06-28	1.7915	1.4226	2.0418	2.2598	0.0121	0.0122	-1.3163
7	2021-12-27	1.2632	1.1450	1.1508	1.4407	0.0068	0.0068	0.0015
8	2022-06-27	6.4321	5.4120	8.1473	8.9647	0.0506	0.0491	-4.9964
9	2022-12-26	7.0621	6.0027	8.1702	9.2126	0.0462	0.0476	-3.6770
10	2023-06-26	1.2891	1.0715	1.5782	1.7452	0.0087	0.0087	0.4056
Mean	NaT	3.8089	3.1929	3.5972	4.1500	0.0229	0.0230	-3.6755
SD	NaT	2.4752	1.9111	2.4286	2.6980	0.0147	0.0147	6.0034

```
Fitting 11 folds for each of 10 candidates, totalling 110 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 32 concurrent workers.
[Parallel(n_jobs=-1)]: Done 110 out of 110 | elapsed:    4.4s finished
```

Interestingly, KNN seems to do worse after it is tuned...

In [35]:
```
exp.plot_model(estimator=tuned_ets, plot="diagnostics")
```

In [36]:
```
exp.plot_model(estimator=tuned_ets, plot="insample")
```

In [37]:
```
exp.plot_model(tuned_ets, plot='forecast', data_kwargs={'fh':36})
```

In [38]:
```
holdout_predictions = exp.predict_model(tuned_ets)
```

	Model	MASE	RMSSE	MAE	RMSE	MAPE	SMAPE	R2
0	ETS	2.1456	1.8932	2.1236	2.6952	0.0114	0.0113	-0.2642

In [39]:
```
finalized_ets = exp.finalize_model(tuned_ets)
exp.plot_model(finalized_ets, plot='forecast', data_kwargs={'fh':24})
```

In [40]:
```
unseen_predictions = exp.predict_model(finalized_ets, fh=10)
unseen_predictions
```

Out[40]:

	y_pred
2024-01-01	191.2193
2024-01-02	191.2686
2024-01-03	191.3179
2024-01-04	191.3672
2024-01-05	191.4165
2024-01-08	191.4658
2024-01-09	191.5151
2024-01-10	191.5644
2024-01-11	191.6137
2024-01-12	191.6630

As shown above, the ETS model of the econometrics family outperformed all the other models, barely beating the random walk.