Prediction Intervals

Prediction intervals provide a measure of the uncertainty in the forecasted values. In time series forecasting, a prediction interval gives an estimated range within which a future observation will fall, based on the level of confidence or uncertainty you set. This level of uncertainty is crucial for making informed decisions, risk assessments, and planning.

For instance, a 95% prediction interval means that 95 out of 100 times, the actual future value will fall within the estimated range. Therefore, a wider interval indicates greater uncertainty about the forecast, while a narrower interval suggests higher confidence.

When using TimeGPT for time series forecasting, you have the option to set the level of prediction intervals according to your requirements. TimeGPT uses conformal prediction to calibrate the intervals.

In [ ]:
```
# | hide
from itertools import product

from fastcore.test import test_eq, test_fail, test_warns
from dotenv import load_dotenv
```

In [ ]:
```
# | hide
load_dotenv()
```

Out[ ]:
```
True
```

In [ ]:
```
import pandas as pd
from nixtlats import TimeGPT
import os
```

In [ ]:
```
timegpt = TimeGPT(token=os.getenv("TIMEGPT_TOKEN"))
```

In [ ]:
```
# | hide
timegpt = TimeGPT()
```

You can test the validate of your token calling the `validate_token` method:

In [ ]:
```
timegpt.validate_token()
```

```
INFO:nixtlats.timegpt:Happy Forecasting! :), If you have questions or need support,
please email ops@nixtla.io
```

Out[ ]:
```
True
```

When using TimeGPT for time series forecasting, you can set the level (or levels) of prediction intervals according to your requirements. Here's how you could do it:

In [ ]:
```
df = pd.read_csv(
    "https://raw.githubusercontent.com/Nixtla/transfer-learning-time-series/main/da
)
df.head()
```

Out[ ]:

	timestamp	value
0	1949-01-01	112
1	1949-02-01	118
2	1949-03-01	132
3	1949-04-01	129
4	1949-05-01	121

In [ ]:
```
timegpt_fcst_pred_int_df = timegpt.forecast(
    df=df,
    h=12,
    level=[80, 90, 99.7],
    time_col="timestamp",
    target_col="value",
)
timegpt_fcst_pred_int_df.head()
```

```
INFO:nixtlats.timegpt:Validating inputs...
INFO:nixtlats.timegpt:Preprocessing dataframes...
INFO:nixtlats.timegpt:Inferred freq: MS
INFO:nixtlats.timegpt:Restricting input...
INFO:nixtlats.timegpt:Calling Forecast Endpoint...
```

Out[ ]:

	timestamp	TimeGPT	TimeGPT-lo-99.7	TimeGPT-lo-90	TimeGPT-lo-80	TimeGPT-hi-80	TimeGPT-hi-90	Ti
0	1961-01-01	437.837921	415.826453	423.783707	431.987061	443.688782	451.892136	459
1	1961-02-01	426.062714	402.833523	407.694061	412.704926	439.420502	444.431366	449
2	1961-03-01	463.116547	423.434062	430.316862	437.412534	488.820560	495.916231	502
3	1961-04-01	478.244507	444.885193	446.776764	448.726837	507.762177	509.712250	511
4	1961-05-01	505.646484	465.736694	471.976787	478.409872	532.883096	539.316182	545

> BEFORE: *9 API Calls | 391158 Tokens | 611.07 Spent*
> AFTER: *10 API Calls | 391326 Tokens | 611.44 Spent*
> **USAGE: 1 API Call | 168 Tokens | 0.37 Spent**

In [ ]:
```python
# | hide
# test shorter horizon
level_short_horizon_df = timegpt.forecast(
    df=df,
    h=6,
    level=[80, 90, 99.7],
    time_col="timestamp",
    target_col="value",
)
test_eq(level_short_horizon_df.shape, (6, 8))
```

```
INFO:nixtlats.timegpt:Validating inputs...
INFO:nixtlats.timegpt:Preprocessing dataframes...
INFO:nixtlats.timegpt:Inferred freq: MS
INFO:nixtlats.timegpt:Restricting input...
INFO:nixtlats.timegpt:Calling Forecast Endpoint...
```

> BEFORE: *10 API Calls | 391326 Tokens | 611.44 Spent*
>
> AFTER: *11 API Calls | 391452 Tokens | 611.68 Spent*
>
> **USAGE: 1 API Call | 126 Tokens | 0.24 Spent**

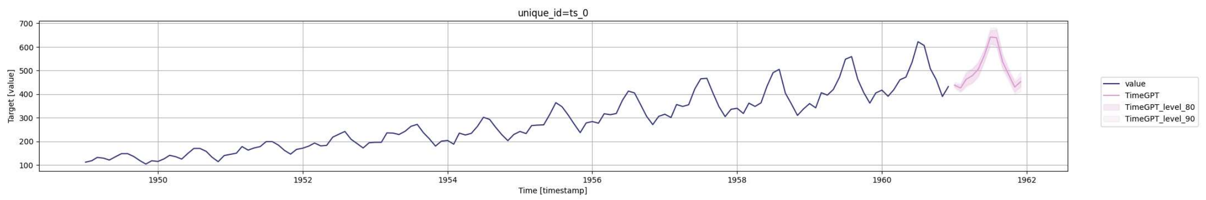
In [ ]:
```python
# | hide
test_level = [80, 90.5]
cols_fcst_df = timegpt.forecast(
    df=df,
    h=12,
    level=[80, 90.5],
    time_col="timestamp",
    target_col="value",
).columns
assert all(f"TimeGPT-{pos}-{lv}" for pos, lv in product(test_level, ["lo", "hi"]))
```

```
INFO:nixtlats.timegpt:Validating inputs...
INFO:nixtlats.timegpt:Preprocessing dataframes...
INFO:nixtlats.timegpt:Inferred freq: MS
INFO:nixtlats.timegpt:Restricting input...
INFO:nixtlats.timegpt:Calling Forecast Endpoint...
```

> BEFORE: *11 API Calls | 391452 Tokens | 611.68 Spent*
>
> AFTER: *12 API Calls | 391596 Tokens | 611.97 Spent*
>
> **USAGE: 1 API Call | 144 Tokens | 0.29 Spent**

In [ ]:
```python
timegpt.plot(
    df,
    timegpt_fcst_pred_int_df,
    time_col="timestamp",
    target_col="value",
    level=[80, 90],
)
```

Out[ ]:



It's essential to note that the choice of prediction interval level depends on your specific use case. For high-stakes predictions, you might want a wider interval to account for more uncertainty. For less critical forecasts, a narrower interval might be acceptable.

Historical Forecast

You can also compute prediction intervals for historical forecasts adding the `add_history=True` parameter as follows:

In [ ]:

```python
timegpt_fcst_pred_int_historical_df = timegpt.forecast(
    df=df,
    h=12,
    level=[80, 90],
    time_col="timestamp",
    target_col="value",
    add_history=True,
)
timegpt_fcst_pred_int_historical_df.head()
```

```
INFO:nixtlats.timegpt:Validating inputs...
INFO:nixtlats.timegpt:Preprocessing dataframes...
INFO:nixtlats.timegpt:Inferred freq: MS
INFO:nixtlats.timegpt:Calling Forecast Endpoint...
INFO:nixtlats.timegpt:Calling Historical Forecast Endpoint...
```

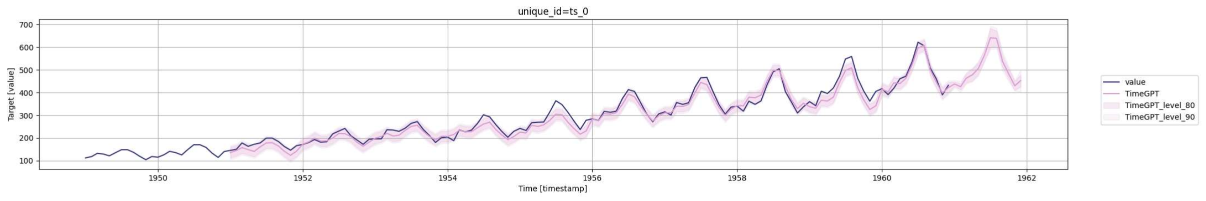
Out[ ]:

	timestamp	TimeGPT	TimeGPT-lo-80	TimeGPT-lo-90	TimeGPT-hi-80	TimeGPT-hi-90
0	1951-01-01	135.483673	111.937768	105.262831	159.029579	165.704516
1	1951-02-01	144.442398	120.896493	114.221556	167.988304	174.663241
2	1951-03-01	157.191910	133.646004	126.971067	180.737815	187.412752
3	1951-04-01	148.769363	125.223458	118.548521	172.315269	178.990206
4	1951-05-01	140.472946	116.927041	110.252104	164.018852	170.693789

> BEFORE: *12 API Calls | 391596 Tokens | 611.97 Spent*
>
> AFTER: *14 API Calls | 392664 Tokens | 614.81 Spent*
>
> **USAGE: 2 API Calls | 1068 Tokens | 2.84 Spent**

```
In [ ]: timegpt.plot(
            df,
            timegpt_fcst_pred_int_historical_df,
            time_col="timestamp",
            target_col="value",
            level=[80, 90],
        )
```

Out[ ]:


unique_id=ts_0
Target [value]
Time [timestamp]
value
TimeGPT
TimeGPT_level_80
TimeGPT_level_90