# \*\*Fuzzy Time Series Forecasting for NIFTY 50 - Detailed Documentation\*\*

## \*\*1. Introduction\*\*

This code implements a \*\*Fuzzy Time Series (FTS) forecasting model\*\* to predict the closing prices of the \*\*NIFTY 50 stock index\*\*. The model uses historical data to:

1. \*\*Fuzzify\*\* numerical stock prices into linguistic fuzzy sets.

2. \*\*Learn patterns\*\* (fuzzy logical relationships) from past transitions.

3. \*\*Forecast future values\*\* based on these relationships.

4. \*\*Evaluate accuracy\*\* using MAE, MSE, and RMSE.

The key improvement over the previous version is the use of \*\*triangular membership functions\*\* instead of crisp intervals, allowing smoother transitions between fuzzy sets.

---

**## \*\*2. Code Walkthrough & Explanation\*\***

**### \*\*2.1 Data Loading & Preprocessing\*\***

```python

import pandas as pd

import matplotlib.pyplot as plt

**# 1. Load and clean data**

df = pd.read\_csv('/content/NIFTY 50\_daily\_data.csv')

# 2. Convert dates (ONCE is enough)

df['date'] = pd.to\_datetime(df['date'], format='%d-%m-%Y %H:%M')

end\_date = df['date'].max()

start\_date = end\_date - pd.DateOffset(years=10)

df = df[(df['date'] >= start\_date) & (df['date'] <= end\_date)]

```

**\*\*Explanation:\*\***

- Loads the dataset containing NIFTY 50 daily closing prices.

- Converts the date column to `datetime` format for time-series operations.

- Filters data for the \*\*last 10 years\*\* to ensure relevance.

**\*\*Why?\*\***

- Ensures consistency in date handling.

- Limits data to a meaningful time window (older data may not reflect current trends).

---

**### \*\*2.2 Data Visualization\*\***

```python

plt.figure(figsize=(12, 6))

plt.plot(df['date'], df['close'], label='NIFTY 50 Close Price')

plt.title('NIFTY 50 Closing Prices - Last 10 Years')

plt.xlabel('date')

plt.ylabel('Closing Price')

plt.legend()

plt.grid(True)

plt.show()

```

**\*\*Explanation:\*\***

- Plots the \*\*historical closing prices\*\* to visualize trends, seasonality, and volatility.

**\*\*Why?\*\***

- Helps in understanding the data before applying forecasting models.

---

**### \*\*2.3 Defining the Universe of Discourse\*\***

```python

data\_min = df['close'].min()

data\_max = df['close'].max()

margin = 0.1 \* (data\_max - data\_min)

u\_max = data\_max + margin

u\_min = data\_min - margin

u = [u\_min, u\_max]

```

\*\***Explanation**:\*\*

- The \*\*universe of discourse (U)\*\* defines the range of possible stock prices.

- `data\_min` and `data\_max` are the observed min/max values.

- A \*\*10% margin\*\* is added to accommodate future fluctuations.

\*\***Why**?\*\*

- Ensures the model can handle values slightly outside historical extremes.

---

**### \*\*2.4 Partitioning into Intervals\*\***

```python

num\_intervals = 30

interval\_width = (u\_max - u\_min) / num\_intervals

intervals = []

for i in range(num\_intervals):

lower = u\_min + i \* interval\_width

upper = lower + interval\_width

intervals.append((lower, upper))

```

**\*\*Explanation:\*\***

- Divides the universe into \*\*30 equal-width intervals\*\*.

- Each interval represents a range of stock prices (e.g., `[8000, 8100]`).

\*\***Why**?\*\*

- More intervals → \*\*Higher granularity\*\* but slower computation.

- Fewer intervals → \*\*Less precise forecasts\*\*.

---

**### \*\*2.5 Triangular Membership Functions\*\***

```python

def triangular\_membership(x, a, b, c):

if x <= a or x >= c:

return 0

elif a < x < b:

return (x - a) / (b - a)

elif b <= x < c:

return (c - x) / (c - b)

else:

return 0

fuzzy\_sets = {}

for i in range(num\_intervals):

if i == 0:

a = intervals[i][0]

b = intervals[i][0]

c = intervals[i][1]

elif i == num\_intervals - 1:

a = intervals[i][0]

b = intervals[i][1]

c = intervals[i][1]

else:

a = intervals[i - 1][0] + interval\_width

b = intervals[i][0] + interval\_width / 2

c = intervals[i][1]

fuzzy\_sets[f'F{i+1}'] = (a, b, c)

```

\*\***Explanation**:\*\*

- Defines \*\*triangular membership functions\*\* (instead of crisp intervals).

- Each fuzzy set `F1, F2, ..., F30` has three parameters:

- `a` = Left base

- `b` = Peak

- `c` = Right base

- Ensures \*\*smooth transitions\*\* between sets (better for forecasting).

\*\***Why**?\*\*

- Crisp intervals can lead to abrupt changes in forecasts.

- Triangular functions allow \*\*overlapping regions\*\*, improving model flexibility.

---

**### \*\*2.6 Fuzzification of Historical Data\*\***

```python

def fuzzify(value):

membership = {}

for label, (a, b, c) in fuzzy\_sets.items():

membership[label] = triangular\_membership(value, a, b, c)

return max(membership, key=membership.get)

df['fuzzy'] = df['close'].apply(fuzzify)

```

\*\***Explanation**:\*\*

- Converts numerical stock prices into \*\*linguistic fuzzy labels\*\* (`F1, F2, ...`).

- For each price, computes membership in all fuzzy sets and picks the \*\*most relevant one\*\*.

\*\***Why**?\*\*

- Allows the model to work with \*\*patterns\*\* instead of exact numbers.

---

**### \*\*2.7 Establishing Fuzzy Logical Relationships (FLRs)\*\***

```python

flrs = []

for i in range(1, len(df)):

prev = df.iloc[i-1]['fuzzy']

curr = df.iloc[i]['fuzzy']

if prev and curr:

flrs.append((prev, curr))

```

\*\***Explanation**:\*\*

- Records transitions between fuzzy sets (e.g., `F5 → F6` means "if today is `F5`, tomorrow is `F6`").

\*\***Why**?\*\*

- Captures \*\*market behavior patterns\*\* (e.g., "After a sharp rise, prices often stabilize").

---

**### \*\*2.8 Grouping FLRs into FLRGs\*\***

```python

flrgs = {}

for prev, curr in flrs:

if prev in flrgs:

if curr not in flrgs[prev]:

flrgs[prev].append(curr)

else:

flrgs[prev] = [curr]

```

\*\***Explanation**:\*\*

- Groups similar transitions (e.g., `F5 → [F4, F5, F6]` means `F5` can lead to multiple outcomes).

\*\***Why**?\*\*

- Helps in \*\*probabilistic forecasting\*\* (not just deterministic).

---

**### \*\*2.9 Defuzzification (Centroid Method)\*\***

```python

def get\_centroid(label):

a, b, c = fuzzy\_sets[label]

return (a + b + c) / 3

```

\*\***Explanation**:\*\*

- Converts fuzzy sets back to numerical values using the \*\*centroid\*\* (average of `a, b, c`).

\*\***Why**?\*\*

- Needed to generate \*\*actual numerical forecasts\*\*.

---

### \*\*2.10 **Forecasting**\*\*

```python

forecasts = []

for i in range(1, len(df)):

prev\_fuzzy = df.iloc[i-1]['fuzzy']

if prev\_fuzzy in flrgs:

curr = flrgs[prev\_fuzzy]

centroid = [get\_centroid(label) for label in curr]

forecast = np.mean(centroid)

else:

forecast = get\_centroid(prev\_fuzzy)

forecasts.append(forecast)

```

**\*\*Explanation:\*\***

- For each day, checks the \*\*previous day’s fuzzy set\*\*.

- If historical transitions exist:

- Takes the \*\*mean of possible next states\*\*.

- If no history:

- Uses the \*\*centroid of the current fuzzy set\*\*.

\*\***Why**?\*\*

- Averages possible outcomes for \*\*smoother predictions\*\*.

---

**### \*\*2.11 Evaluation & Visualization\*\***

```python

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error

df\_eval = df\_forecast.dropna(subset=['forecast'])

y\_true = df\_eval['close']

y\_pred = df\_eval['forecast']

mae = mean\_absolute\_error(y\_true, y\_pred)

mse = mean\_squared\_error(y\_true, y\_pred)

rmse = np.sqrt(mse)

print(f"MAE: {mae:.2f}")

print(f"MSE: {mse:.2f}")

print(f"RMSE: {rmse:.2f}")

plt.figure(figsize=(14, 6))

plt.plot(df\_eval['date'], df\_eval['close'], label='Actual Close')

plt.plot(df\_eval['date'], df\_eval['forecast'], label='Fuzzy Forecast')

plt.title('Fuzzy Time Series Forecast vs Actual - NIFTY 50')

plt.xlabel('Date')

plt.ylabel('Price')

plt.legend()

plt.grid(True)

plt.tight\_layout()

plt.show()

```

**\*\*Explanation:\*\***

- \*\*MAE, MSE, RMSE\*\* quantify forecast accuracy.

- \*\*Plot\*\* compares actual vs predicted values visually.

**\*\*Why?\*\***

- Helps assess model performance and identify areas for improvement.

---

**## \*\*3. Conclusion\*\***

This model:

✅ Uses \*\*triangular fuzzy sets\*\* for smoother transitions.

✅ Captures \*\*market patterns\*\* via fuzzy relationships.

✅ Provides \*\*numerical forecasts\*\* via defuzzification.

✅ Evaluates performance using \*\*MAE, MSE, RMSE\*\*.

**\*\*Possible Improvements:\*\***

- Use \*\*trapezoidal/Gaussian membership\*\* for better accuracy.

- Incorporate \*\*volume or external factors\*\* (hybrid models).

- Optimize the \*\*number of intervals\*\* (hyperparameter tuning).

This documentation provides a \*\*detailed, step-by-step breakdown\*\* of the fuzzy time series forecasting process. 🚀