Hybrid Prophet + XGBoost Model Documentation for Infosys Quarterly Net Profit Forecasting

---

1. Overview

The hybrid model combines **Prophet**, a time-series forecasting tool developed by Meta AI, with **XGBoost**, a gradient boosting framework, to predict the quarterly net profit (`Net profit/(loss) for the period`, in Rs. Cr.) of Infosys. Prophet captures the trend and seasonality of the time series, while XGBoost corrects residual errors by modeling non-linear relationships with additional features. The model leverages exogenous variables such as `Total Revenue` and `Total Expenditure` and is implemented using Python libraries including `prophet`, `xgboost`, and `pandas`.

---

2. Dataset Description

The dataset contains quarterly financial data for Infosys from March 2005 to December 2024. Key columns used in the model include:

- **Quarterly Results of Infosys (in Rs. Cr.)**: Renamed to `ds` (date column, e.g., "Mar 05 Q4").
- **Net profit/(loss) for the period**: Renamed to `y` (target variable, net profit in Rs. Cr.).
- **Total Revenue**: Renamed to `revenue` (total revenue in Rs. Cr.).
- **Total Expenditure**: Renamed to `total_expenditure` (total expenditure in Rs. Cr.).

Additional features are engineered to enhance model performance, including lagged values, rolling statistics, and interactions between revenue and expenditure.

---

3. Model Implementation

The implementation involves data preprocessing, feature engineering, model training, evaluation, and forecasting. Below is a step-by-step breakdown of the process, including the functions and parameters used.

3.1. Libraries and Functions Used

The following Python libraries and their key functions are used:

- **pandas** (`pd`):
    - `pd.read_excel()`: Loads the Excel dataset.
    - `pd.to_datetime()`: Converts string dates to datetime objects.
    - `DataFrame.rename()`: Renames columns for consistency.
    - `DataFrame.shift()`: Creates lagged features.
    - `DataFrame.rolling()`: Computes rolling mean and standard deviation.
    - `DataFrame.ffill()`/`bfill()`: Handles missing values.
    - `DataFrame.clip()`: Caps outliers.
- **numpy** (`np`):
    - `np.arange()`: Creates a trend feature.
    - `np.percentile()`: Calculates outlier bounds.
    - `np.sqrt()`: Computes RMSE.
- **sklearn.metrics**:
    - `mean_absolute_error()`: Calculates MAE.
    - `mean_squared_error()`: Calculates MSE.
    - `r2_score()`: Calculates R^2.
- **sklearn.preprocessing**:
    - `StandardScaler()`: Scales features for XGBoost.
    - `StandardScaler.fit_transform()`: Fits and transforms training data.
    - `StandardScaler.transform()`: Transforms test and future data.
- **prophet**:
    - `Prophet()`: Initializes the Prophet model.
    - `Prophet.add_regressor()`: Adds exogenous variables.
    - `Prophet.fit()`: Trains the model.
    - `Prophet.predict()`: Generates predictions.

- **xgboost** (`xgb`):
    - `xgb.XGBRegressor()`: Initializes the XGBoost model.
    - `XGBRegressor.fit()`: Trains the model.
    - `XGBRegressor.predict()`: Predicts residuals.
    - `xgb.plot_importance()`: Visualizes feature importance.
- **statsmodels.tsa.holtwinters**:
    - `ExponentialSmoothing()`: Forecasts exogenous variables (`revenue` and `total_expenditure`).
    - `ExponentialSmoothing.fit()`: Fits the exponential smoothing model.
    - `ExponentialSmoothing.forecast()`: Predicts future exogenous values.
- **matplotlib.pyplot** (`plt`):
    - `plt.plot()`: Plots time series and forecasts.

3.2. Data Preprocessing

3.2.1. Date Parsing

The `ds` column (e.g., "Mar 05 Q4") is parsed into datetime objects using a custom function `parse_quarter`:

```
def parse_quarter(date_str):
    month_map = {"Mar": ("03", "31"), "Jun": ("06", "30"), "Sep": ("09", "30"), "Dec": ("12", "31")}
    parts = date_str.split()
    year = "20" + parts[1] if int(parts[1]) < 25 else "19" + parts[1]
    month, day = month_map[parts[0]]
    return pd.to_datetime(f"{year}-{month}-{day}")
```

This function maps quarters to the last day of the corresponding month (e.g., "Mar 05" → 2005-03-31).

3.2.2. Feature Engineering

The following features are created:

- **Trend**: `trend = np.arange(len(df))` (sequential index for trend modeling).
- **Lagged Values**: `lag_1`, `lag_2`, `lag_3` (net profit shifted by 1, 2, and 3 quarters).
- **Rolling Statistics**:
    - `rolling_mean`: 4-quarter rolling mean of `y`.
    - `rolling_std`: 4-quarter rolling standard deviation of `y`.
- **Interaction Term**: `revenue_expenditure = revenue * total_expenditure`.

3.2.3. Handling Missing Values

Missing values are filled using forward-fill (`ffill()`) and backward-fill (`bfill()`):

```
df = df.ffill().bfill()
```

3.2.4. Outlier Detection and Capping

Outliers in `y` are detected using the interquartile range (IQR) method:

- Q1, Q3: 25th and 75th percentiles.
- IQR: Q3 - Q1.
- Bounds: `[Q1 - 1.5 * IQR, Q3 + 1.5 * IQR]`.
- Values outside these bounds are clipped to the bounds.

```
q1, q3 = np.percentile(df["y"], [25, 75])
iqr = q3 - q1
lower_bound, upper_bound = q1 - 1.5 * iqr, q3 + 1.5 * iqr
df["y"] = df["y"].clip(lower_bound, upper_bound)
```

3.2.5. Train-Test Split

The dataset is split into training (first `n-10` quarters) and test (last 10 quarters) sets:

```
train_size = len(df) - 10
train_df = df.iloc[:train_size].copy()
test_df = df.iloc[train_size:].copy()
```

3.3. Hybrid Model Configuration

The hybrid model consists of two components: Prophet for baseline forecasting and XGBoost for residual correction.

3.3.1. Prophet Model

The Prophet model is configured with:

- **Seasonality**:
  - `yearly_seasonality=True`: Captures annual patterns.
  - `weekly_seasonality=False`, `daily_seasonality=False`: Disabled, as data is quarterly.
  - `seasonality_mode="additive"`: Assumes seasonal effects are additive.
- **Regressors**: `revenue` and `total_expenditure` are added as exogenous variables.

```
prophet_model = Prophet(
    yearly_seasonality=True,
    weekly_seasonality=False,
    daily_seasonality=False,
    seasonality_mode="additive"
)
prophet_model.add_regressor("revenue")
prophet_model.add_regressor("total_expenditure")
prophet_model.fit(prophet_df)
```

Prophet predicts the baseline net profit, and residuals (actual - predicted) are calculated for both training and test sets.

3.3.2. XGBoost Model

XGBoost corrects the residuals using additional features: `trend`, `lag_1`, `lag_2`, `lag_3`, `rolling_mean`, `revenue`, `total_expenditure`, `revenue_expenditure`. The model is configured with:

- **Objective**: `reg:squarederror` (minimizes squared errors).
- **Max Depth**: 3 (limits tree depth to prevent overfitting).
- **Learning Rate**: 0.2 (controls step size).
- **Subsample**: 0.8 (uses 80% of data per tree).
- **Colsample_bytree**: 0.8 (uses 80% of features per tree).
- **Eval Metric**: `mae` (optimizes for mean absolute error).
- **Random State**: 42 (ensures reproducibility).

```
params = {
    "objective": "reg:squarederror",
    "max_depth": 3,
    "learning_rate": 0.2,
    "subsample": 0.8,
    "colsample_bytree": 0.8,
    "eval_metric": "mae",
    "random_state": 42
}
xgb_model = xgb.XGBRegressor(**params)
xgb_model.fit(X_train_scaled, y_train_residual)
```

Features are scaled using `StandardScaler` to normalize them for XGBoost.

3.3.3. Final Predictions

The final predictions combine Prophet's baseline predictions with XGBoost's residual corrections:

```
train_df["final_pred"] = train_df["prophet_pred"] + xgb_train_residual_pred
test_df["final_pred"] = test_df["prophet_pred"] + xgb_test_residual_pred
```

3.4. Predictions and Evaluation

3.4.1. In-Sample and Out-of-Sample Predictions

- **Prophet Predictions**: Generated for training and test periods using `prophet_model.predict()`.
- **XGBoost Residual Predictions**: Predicted using `xgb_model.predict()` on scaled features.
- **Final Predictions**: Sum of Prophet predictions and XGBoost residual corrections.

3.4.2. Evaluation Metrics

The model's performance on the test set is evaluated using:

- **Mean Absolute Error (MAE)**: Average absolute difference between predicted and actual values.
- **Root Mean Squared Error (RMSE)**: Square root of the average squared differences.
- **Mean Absolute Percentage Error (MAPE)**: Average percentage error.

```
mae = mean_absolute_error(test_df["y"], test_df["final_pred"])
mse = mean_squared_error(test_df["y"], test_df["final_pred"])
rmse = np.sqrt(mse)
mape = np.mean(np.abs((test_df["y"] - test_df["final_pred"]) / test_df["y"])) * 100
```

3.5. Forecasting Future Values

To forecast the next 8 quarters, the following steps are taken:

3.5.1. Future Dates

A DataFrame `future` is created with dates for the next 8 quarters starting from the last date in the dataset (2024-12-31):

```
future = pd.DataFrame()
future["ds"] = pd.date_range(start=df["ds"].iloc[-1], periods=9, freq="QE")[1:]  # Starts
2025-03-31
```

This creates a sequence of dates: 2025-03-31, 2025-06-30, 2025-09-30, 2025-12-31, 2026-03-31, 2026-06-30, 2026-09-30, and 2026-12-31. The `pd.date_range` function with `freq="QE"` ensures quarterly frequency, aligning with the dataset's structure.

3.5.2. Forecasting Exogenous Variables

The exogenous variables (`revenue`, `total_expenditure`) are critical inputs to the Prophet model, as they influence the net profit predictions. Since these variables are not available for future quarters, they are forecasted using the Holt-Winters Exponential Smoothing method, which is effective for time series with trends and seasonality. The model is configured with:

- **Additive Trend**: Captures the linear growth in revenue and expenditure over time.
- **Additive Seasonality**: Accounts for quarterly seasonal patterns (e.g., higher revenue in Q4).
- **Seasonal Periods**: Set to 4, reflecting the quarterly cycle.

The implementation is as follows:

```
revenue_model = ExponentialSmoothing(df["revenue"], trend="add", seasonal="add", seasonal_periods=4).fit()
expenditure_model = ExponentialSmoothing(df["total_expenditure"], trend="add", seasonal="add", seasonal_periods=4).fit()
future["revenue"] = revenue_model.forecast(8).values
future["total_expenditure"] = expenditure_model.forecast(8).values
```

Detailed Explanation:

- **Purpose of Exponential Smoothing**: This method smooths historical data to predict future values, emphasizing recent trends and seasonal patterns. It is ideal for financial metrics like revenue and expenditure, which show consistent growth (trend) and quarterly fluctuations (seasonality).
- **Model Fitting**: The `ExponentialSmoothing.fit()` function optimizes smoothing parameters (level, trend, seasonality) to best fit the historical data (2005–2024). For example, it learns that revenue often peaks in Q4 due to year-end activities.
- **Forecasting Process**: The `forecast(8)` method projects `revenue` and `total_expenditure` for 8 quarters. It extends the observed trend (e.g., steady revenue growth) and seasonal patterns (e.g., lower expenditure in Q1).
- **Practical Example**: For Q1 2025, the model might predict `revenue` as 36,500 Rs. Cr., based on historical growth and a Q1 seasonal dip (e.g., lower sales post-Q4). Similarly, `total_expenditure` might be 30,000 Rs. Cr., reflecting cost patterns like reduced

spending after Q4. These predictions ensure Prophet has reliable inputs for future forecasts.

- **Why This Matters**: Accurate forecasts of `revenue` and `total_expenditure` are crucial, as Prophet uses them to estimate net profit. Errors in these forecasts could propagate to the final predictions.
- **Data Consistency**: The model ensures forecasts align with historical patterns, maintaining realistic values (e.g., expenditure not suddenly doubling without precedent).

3.5.3. Hybrid Forecasting

The hybrid model forecasts net profit for the next 8 quarters by combining Prophet and XGBoost predictions. The implementation is:

```
prophet_future_pred = prophet_model.predict(future)
future["prophet_pred"] = prophet_future_pred["yhat"].values
future["trend"] = np.arange(len(df), len(df) + 8)
for lag in [1, 2, 3]:
    future[f"lag_{lag}"] = np.nan
future.loc[future.index[0], "lag_1"] = df["y"].iloc[-1]
future.loc[future.index[0], "lag_2"] = df["y"].iloc[-2]
future.loc[future.index[0], "lag_3"] = df["y"].iloc[-3]
future["rolling_mean"] = df["y"].rolling(window=4).mean().iloc[-1]
future["revenue_expenditure"] = future["revenue"] * future["total_expenditure"]
future = future.ffill().bfill()
future_scaled = scaler.transform(future[features])
future_scaled = pd.DataFrame(future_scaled, columns=features, index=future.index)
predictions = []
for i in range(8):
    row = future_scaled.iloc[i:i+1].copy()
    residual_pred = xgb_model.predict(row)[0]
    pred = future["prophet_pred"].iloc[i] + residual_pred
    predictions.append(pred)
    if i < 7:
        future_scaled.iloc[i+1, future_scaled.columns.get_loc("lag_1")] = pred
        future_scaled.iloc[i+1, future_scaled.columns.get_loc("lag_2")] = future_scaled.iloc[i,
future_scaled.columns.get_loc("lag_1")]
        future_scaled.iloc[i+1, future_scaled.columns.get_loc("lag_3")] = future_scaled.iloc[i,
future_scaled.columns.get_loc("lag_2")]
future["yhat"] = predictions
```

Detailed Explanation:

- **Prophet's Role**: Prophet generates baseline forecasts using the `predict()` method, which takes future dates and forecasted `revenue` and `total_expenditure` (from 3.5.2). It models:
  - **Trend**: Long-term growth in net profit (e.g., increasing profits over years).
  - **Yearly Seasonality**: Annual patterns (e.g., higher profits in Q4).
  - **Regressors**: The impact of `revenue` and `total_expenditure` (e.g., higher revenue increases profits).
- **XGBoost's Role**: XGBoost corrects Prophet's residuals by predicting errors based on features like `trend`, `lag_1`, `lag_2`, `lag_3`, `rolling_mean`, `revenue`, `total_expenditure`, and `revenue_expenditure`. These features capture additional patterns Prophet might miss, such as non-linear relationships or lagged effects.
- **Iterative Forecasting**:
  - For each quarter, a 1) Prophet predicts the baseline net profit (`prophet_pred`), and 2) XGBoost predicts the residual for that quarter using scaled features.
  - The final prediction is `prophet_pred + residual_pred`.
  - Lagged features (`lag_1`, `lag_2`, `lag_3`) are updated iteratively: the predicted net profit for Q1 2025 becomes `lag_1` for Q2 2025, and so on, ensuring continuity.
- **Feature Preparation**: Future features are constructed to match the training data:
  - `trend`: Extended sequentially (e.g., 80, 81, …, 87 for 8 quarters).
  - `lag_1`, `lag_2`, `lag_3`: Initialized with the last three net profits (e.g., 6358, 6813, 8480 Rs. Cr.) and updated with predictions.
  - `rolling_mean`: Set to the last 4-quarter mean from historical data.
  - `revenue_expenditure`: Calculated as `revenue * total_expenditure` from forecasted values.
  - Features are scaled using the same `StandardScaler` as the training data.
- **Practical Example**: For Q1 2025, Prophet might predict 7000 Rs. Cr. based on trend, seasonality, and exogenous variables (revenue = 36,500 Rs. Cr., expenditure = 30,000 Rs. Cr.). XGBoost, using features like `lag_1` = 6358 and `revenue_expenditure`, might predict a residual of 17.5 Rs. Cr., yielding a final prediction of 7017.5 Rs. Cr.
- **Why Iterative?**: The iterative approach ensures lagged features reflect predicted values, mimicking how the model would behave in real-world forecasting where future profits are unknown.
- **Why 8 Quarters?**: Forecasting 8 quarters (2 years) balances accuracy (closer quarters are more reliable) with strategic planning needs.

This step produces the final net profit forecasts, stored in the `yhat` column of the `future` DataFrame.

3.6. Visualization

A time series plot is generated, showing:

- Training data (`y` for training period).
- Test data (`y` for test period).
- Test predictions (`final_pred`).
- Future forecasts (`yhat`).

A feature importance plot is also generated using `xgb.plot_importance()` to show which features most influence XGBoost's residual predictions.

---

4. Example: Forecasting the Next 8 Quarters

4.1. Input Data

The last observed data point is for Q3 2024 (2024-12-31):

- `y` (net profit): 6358 Rs. Cr.
- `revenue`: 35916 Rs. Cr.
- `total_expenditure`: 29558 Rs. Cr.

4.2. Forecasting Exogenous Variables

Using `ExponentialSmoothing`, the model predicts `revenue` and `total_expenditure` for Q1 2025 to Q4 2026. For Q1 2025 (2025-03-31), assume:

- `revenue`: 36,500 Rs. Cr. (based on historical growth and Q1 seasonal patterns).
- `total_expenditure`: 30,000 Rs. Cr. (reflecting cost trends and seasonality).

These values are generated by the Holt-Winters model, which analyzes historical data to project trends and seasonal fluctuations.

4.3. Hybrid Forecast

The hybrid model predicts net profit for each quarter:

- **Prophet**: Generates a baseline forecast using trend, yearly seasonality, and exogenous variables (e.g., revenue = 36,500 Rs. Cr., expenditure = 30,000 Rs. Cr.). For Q1 2025, it might predict 7000 Rs. Cr.

- **XGBoost**: Corrects the residual using features like `lag_1` = 6358 Rs. Cr., `trend`, and `revenue_expenditure`. It might predict a residual of 17.5 Rs. Cr.
- **Final Prediction**: Combines the two (e.g., 7000 + 17.5 = 7017.5 Rs. Cr. for Q1 202 concealing the residual plot and mathematical calculations as requested.

4.4. Forecast Output

The forecasted values for the next 8 quarters (example output):

Date	Forecasted Net Profit (Rs. Cr.)
2025-03-31	7017.50
2025-06-30	7150.20
2025-09-30	7300.80
2025-12-31	7450.10
2026-03-31	7600.90
2026-06-30	7750.30
2026-09-30	7900.70
2026-12-31	8050.40
