Hybrid Prophet-XGBoost Forecasting Model Documentation

Overview

This Python script implements a hybrid forecasting model that combines the Prophet time series model with an XGBoost regressor to forecast quarterly financial metrics, specifically net profit, using historical data that includes total revenue and total expenditure. The code is structured as a modular pipeline with two primary functions: `load_data` for data preprocessing and `run_hybrid_model` for modeling, evaluation, and forecasting. It is designed to handle quarterly financial data from an Excel file, perform robust preprocessing, and return structured outputs for further analysis.

Purpose

- **Objective**: Forecast net profit for the next 8 quarters using a hybrid model where Prophet captures trend and seasonality, and XGBoost models residuals using additional features.
- **Use Case**: Financial analysis for companies, such as predicting future profitability based on historical quarterly results with enhanced accuracy through residual correction.
- **Key Features**:
  - Loads and validates Excel data with date, net profit, revenue, and expenditure columns.
  - Preprocesses data by parsing dates, handling missing values, capping outliers, and engineering features.
  - Trains a Prophet model with revenue and expenditure as regressors.
  - Uses XGBoost to predict residuals, improving final predictions.
  - Evaluates model performance using MAE, RMSE, R^2, and MAPE.
  - Forecasts future values with confidence intervals from Prophet.

- Returns structured results (training data, test data, forecasts, and metrics).

Code Structure

The code is organized into two main functions, with dependencies on standard Python libraries for data manipulation, statistical modeling, machine learning, and forecasting.

Dependencies

- **pandas**: For data manipulation and DataFrame operations.
- **numpy**: For numerical computations (e.g., outlier capping, feature engineering).
- **xgboost**: For training the XGBoost regressor to model residuals.
- **sklearn.metrics**: For calculating mean absolute error (MAE), mean squared error (MSE), and R^2 score.
- **sklearn.preprocessing**: For feature scaling using `StandardScaler`.
- **statsmodels.tsa.holtwinters**: For Exponential Smoothing to forecast exogenous variables.
- **prophet**: For time series forecasting with trend, seasonality, and regressors.
- **matplotlib.pyplot**: Imported but not used in the core functions (likely for optional plotting).
- **os**: For file system operations (not used in the provided code but imported).

Functions

1. `load_data(df)`:
   - Purpose: Loads and preprocesses the input Excel data.
   - Input: A pandas DataFrame (`df`) from an Excel file.
   - Output: A tuple containing the processed DataFrame and an error message (if any).
2. `run_hybrid_model(df)`:
   - Purpose: Trains the hybrid Prophet-XGBoost model, evaluates performance, and forecasts future values.

- Input: A preprocessed DataFrame from `load_data`.
- Output: A dictionary with training data, test data, forecasts, and evaluation metrics.

Detailed Functionality

1. `load_data(df)`

This function validates and preprocesses the input data to ensure it meets the requirements for the hybrid model.

Input

- **df**: A pandas DataFrame loaded from an Excel file, expected to have:
  - A date column (first column, e.g., "Mar 05 Q4").
  - Columns named "Net profit/(loss) for the period", "Total Revenue", and "Total Expenditure".

Processing Steps

1. **Validation**:
   - Checks if the DataFrame is non-empty and has at least 4 columns.
   - Verifies the presence of required columns: "Net profit/(loss) for the period", "Total Revenue", "Total Expenditure".
   - Returns `None` and an error message if validation fails.
2. **Column Renaming**:
   - Renames the first column to `ds` (date).
   - Renames required columns to `y` (net profit), `revenue`, and `total_expenditure`.
3. **Date Parsing**:
   - Defines a nested function `parse_quarter` to convert date strings (e.g., "Mar 05") to datetime objects.
   - Supports quarters: "mar" (Q1), "jun" (Q2), "sep" (Q3), "dec" (Q4).
   - Handles year suffixes (e.g., "05" → 2005 if ≤ 50, else 1905).
   - Assigns end-of-quarter dates (e.g., "Mar 05" → 2005-03-31).

- Drops rows with invalid dates and sorts by date.

4. **Debugging**:
   - Prints sample values from the date column for debugging.
5. **Output**:
   - Returns a tuple: `(processed_df, error_message)`.
     - `processed_df`: DataFrame with columns `ds`, `y`, `revenue`, `total_expenditure` (or `None` if errors occur).
     - `error_message`: `None` if successful, else a descriptive error string.

Error Handling

- Empty DataFrame or insufficient columns: Returns error message.
- Missing required columns: Lists missing columns in the error message.
- Invalid dates: Returns sample invalid values for debugging.

2. `run_hybrid_model(df)`

This function performs feature engineering, trains the hybrid Prophet-XGBoost model, evaluates performance, and forecasts future values.

Input

- **df**: A preprocessed DataFrame from `load_data` with columns `ds`, `y`, `revenue`, `total_expenditure`.

Processing Steps

1. **Feature Engineering**:
   - Creates features for XGBoost:
     - `trend`: Linear sequence (`np.arange(len(df))`) to capture long-term trends.
     - `lag_1`, `lag_2`, `lag_3`: Lagged values of `y` (net profit) for 1, 2, and 3 quarters.
     - `rolling_mean`: 4-quarter rolling mean of `y` to capture smooth trends.
     - `rolling_std`: 4-quarter rolling standard deviation of `y` to capture volatility.
     - `revenue_expenditure`: Interaction term (`revenue * total_expenditure`).

2. **Missing Value Handling**:
    - Applies forward fill (`ffill`) followed by backward fill (`bfill`) to impute missing values in all columns.
3. **Outlier Capping**:
    - Computes the interquartile range (IQR) for `y` (net profit).
    - Caps `y` values outside `[Q1 - 1.5*IQR, Q3 + 1.5*IQR]` to mitigate the impact of extreme values.
4. **Train-Test Split**:
    - Splits the data, reserving the last 10 quarters for testing (`test_df`) and the rest for training (`train_df`).
    - Stores training and test dates (`dates_train`, `dates_test`) for potential use (though not used in this version).
5. **Prophet Model**:
    - Prepares a DataFrame for Prophet with `ds`, `y`, `revenue`, and `total_expenditure`.
    - Configures a Prophet model with:
        - `yearly_seasonality=True`, `weekly_seasonality=False`, `daily_seasonality=False`.
        - `seasonality_mode="additive"`.
        - `revenue` and `total_expenditure` as regressors.
    - Fits the model on `prophet_df`.
    - Generates predictions for training (`prophet_train_pred`) and test sets (`prophet_test_pred`).
    - Stores predictions in `train_df["prophet_pred"]` and `test_df["prophet_pred"]`.
6. **Residual Computation**:
    - Computes residuals as the difference between actual and Prophet-predicted values:
        - `train_df["residual"] = train_df["y"] - train_df["prophet_pred"]`.
        - `test_df["residual"] = test_df["y"] - test_df["prophet_pred"]`.
7. **XGBoost Model**:

- Defines features for XGBoost: `["trend", "lag_1", "lag_2", "lag_3", "rolling_mean", "revenue", "total_expenditure", "revenue_expenditure"]`.
- Extracts feature matrices (`X_train`, `X_test`) and residual targets (`y_train_residual`, `y_test_residual`).
- Handles infinite values in residuals by replacing with `NaN` and filling with 0.
- Scales features using `StandardScaler` to normalize `X_train` and `X_test`.
- Configures an XGBoost regressor with:
  - Objective: `reg:squarederror`.
  - Parameters: `max_depth=3`, `learning_rate=0.2`, `subsample=0.8`, `colsample_bytree=0.8`, `eval_metric="mae"`, `random_state=42`.
- Fits the model on scaled training data (`X_train_scaled`, `y_train_residual`).
- Predicts residuals for training (`xgb_train_residual_pred`) and test sets (`xgb_test_residual_pred`).

8. **Final Predictions**:
   - Combines Prophet predictions with XGBoost residual predictions:
     - `train_df["final_pred"] = train_df["prophet_pred"] + xgb_train_residual_pred`.
     - `test_df["final_pred"] = test_df["prophet_pred"] + xgb_test_residual_pred`.
   - Fills any `NaN` values in `final_pred` with `prophet_pred` to ensure robustness.
9. **Evaluation**:
   - Computes metrics on the test set for `final_pred`:
     - **MAE**: Mean Absolute Error.
     - **RMSE**: Root Mean Squared Error.
     - **R^2**: Coefficient of determination.
     - **MAPE**: Mean Absolute Percentage Error (as a percentage).
   - Stores metrics in a dictionary: `{"mae": mae, "rmse": rmse, "r2": r2, "mape": mape}`.
10. **Forecasting**:
    - Creates a `future` DataFrame with 8 quarterly dates starting after the last historical date (`df["ds"].iloc[-1]`).

- Forecasts exogenous variables using Exponential Smoothing:
  - `revenue`: Fits a model with additive trend and seasonal components (period=4).
  - `total_expenditure`: Same as above.
- Generates Prophet forecasts for the future (`prophet_future_pred`), including `yhat`, `yhat_lower`, and `yhat_upper`.
- Prepares future features for XGBoost:
  - `trend`: Continues the linear sequence from historical data.
  - `lag_1`, `lag_2`, `lag_3`: Initializes with the last known `y` values, updated iteratively.
  - `rolling_mean`: Uses the last historical rolling mean.
  - `revenue_expenditure`: Computed as `revenue * total_expenditure`.
- Scales future features using the same `StandardScaler`.
- Iteratively predicts residuals for each future period using XGBoost, updating lagged features dynamically.
- Combines Prophet forecasts with XGBoost residual predictions to get final `yhat` values.
- Stores forecasts in `future[["ds", "yhat", "yhat_lower", "yhat_upper"]]`.

11. **Output**:
    - Returns a dictionary:
      - `train_df`: Training DataFrame with `ds`, `y`, `revenue`, `total_expenditure`, `trend`, `lag_1`, `lag_2`, `lag_3`, `rolling_mean`, `rolling_std`, `revenue_expenditure`, `prophet_pred`, `residual`, `final_pred`.
      - `test_df`: Test DataFrame with the same columns plus `prophet_pred`, `residual`, `final_pred`.
      - `future_df`: Forecast DataFrame with `ds`, `yhat` (forecasted values), `yhat_lower`, `yhat_upper` (confidence intervals from Prophet).
      - `metrics`: Dictionary with `mae`, `rmse`, `r2`, `mape`.

Error Handling

- Handles missing values and outliers to prevent model failures.

- Replaces infinite residual values with `NaN` and fills with 0.
- Fills `NaN` predictions with Prophet predictions.
- Uses forward/backward filling for future feature imputation.

Usage

Prerequisites

- **Python Version**: 3.6 or higher.
- **Libraries**: Install required packages:

```
pip install pandas numpy xgboost scikit-learn statsmodels prophet mat
plotlib
```

- **Input Data**: An Excel file with:
  - First column: Quarterly dates (e.g., "Mar 05 Q4").
  - Columns: "Net profit/(loss) for the period", "Total Revenue", "Total Expenditure".
  - Example format:

```
Quarterly Results | Net profit/(loss) for the period | Total Revenue |
Total Expenditure
Mar 05 Q4       | 1000                    | 5000        | 4000
Jun 05 Q2       | 1100                    | 5200        | 4100
...
```

Example Usage

```
# Load Excel file
df = pd.read_excel("path/to/Infosys_Sorted_Quarterly_Data.xlsx", sheet_na
me="Sheet1")

# Preprocess data
processed_df, error = load_data(df)
if error:
    print(error)
```

```
    exit()

# Run hybrid model
results = run_hybrid_model(processed_df)

# Access results
print("Evaluation Metrics:")
for metric, value in results["metrics"].items():
    print(f"{metric.upper()}: {value:.2f}")

print("\nForecast for Next 8 Quarters:")
print(results["future_df"][["ds", "yhat"]].round(2))

# Optional: Save results to CSV
results["train_df"].to_csv("train_predictions.csv", index=False)
results["test_df"].to_csv("test_predictions.csv", index=False)
results["future_df"].to_csv("forecasts.csv", index=False)
```

Example Output

```
Sample values in first column ('Quarterly Results'): ['Mar 05 Q4', 'Jun 05 Q
2', 'Sep 05 Q3', 'Dec 05 Q4', 'Mar 06 Q1']
Evaluation Metrics:
MAE: 320.50
RMSE: 400.25
R2: 0.85
MAPE: 4.95

Forecast for Next 8 Quarters:
          ds    yhat
2025-03-31  6250.10
2025-06-30  5900.45
2025-09-30  6650.30
2025-12-31  6900.75
2026-03-31  6850.20
2026-06-30  6550.60
```

```
2026-09-30  7300.15
2026-12-31  7550.90
```