Análise do Projeto: Processador de Cenários de Previsão de Séries Temporais

1. Visão Geral do Projeto

O projeto "Processador de Cenários de Previsão de Séries Temporais" é uma aplicação desktop desenvolvida em Python, utilizando a biblioteca CustomTkinter para a interface gráfica. O objetivo é permitir ao usuário gerenciar, executar e visualizar cenários de previsão para séries temporais, com suporte aos modelos ARIMA, Prophet e RandomForest. Os resultados são persistidos em bancos SQLite e podem ser exportados para CSV/Excel.

2. Estrutura do Projeto

A estrutura do repositório está organizada em módulos funcionais e utilitários:

- `main.py`: Ponto de entrada da aplicação, inicializa a interface gráfica.
- `app_gui.py`: Implementa a janela principal, navegação entre telas e integração dos frames de funcionalidades.
- `config_manager_gui.py`: Gerencia a configuração dos cenários, incluindo adição, edição, remoção e importação de dados históricos via CSV/Excel.
- `scenarios_config.yaml`: Arquivo YAML que armazena as configurações dos cenários de previsão.
- `requirements.txt`: Lista de dependências Python.
- `create_dummy_db.py`: Script para gerar o banco de dados de exemplo `dados_bcb.db`.
- `methods/_run_forecasting.py`: Orquestra a execução de previsões para cenários individuais.
- `modules/`: Lógica de negócio principal:
  - `chart_generator.py`: Geração de gráficos interativos.
  - `data_exporter.py`: Exportação de resultados para CSV/Excel.
  - `data_loader.py`: Carregamento e consolidação de dados históricos.
  - `forecasting_model.py`: Implementação dos modelos ARIMA, Prophet, RandomForest.
  - `model_evaluator.py`: Avaliação de desempenho dos modelos.
  - `results_processor.py`: Processamento dos resultados das previsões.
  - `scenario_loader.py`: Carregamento das configurações dos cenários.
- `persistence/`: Persistência de dados:
  - `base_adapter.py`: Interface base para adaptadores de banco.
  - `sqlite_adapter.py`: Implementação para SQLite, manipula `previsoes.db`.
- `utils/`: Utilitários:
  - `get_base_path.py`: Gerenciamento de caminhos de arquivos.
  - `logger_config.py`: Configuração e integração de logging com a GUI.
- `tests/`: Testes automatizados para todos os módulos principais.

3. Funcionalidades Principais

A aplicação oferece as seguintes funcionalidades:

3.1. Gerenciamento de Cenários

- **Criação e Edição:** Permite adicionar novos cenários de previsão e editar os existentes através de uma interface gráfica.
- **Remoção:** Possibilita a exclusão de cenários configurados.
- **Seleção de Séries:** Permite associar cenários a diferentes séries temporais disponíveis no banco de dados.
- **Configuração de Modelos:** Suporte para diferentes modelos de previsão (ARIMA, Prophet, RandomForest) com configuração de parâmetros.
- **Importação de Dados:** Capacidade de importar dados históricos de séries temporais a partir de arquivos CSV e Excel, servindo como base para futuras integrações de dados.

Permite ao usuário definir e armazenar diferentes cenários de previsão. Cada cenário inclui:

- Nome do cenário
- ID da série temporal a ser prevista
- Modelo de previsão (ARIMA, Prophet, RandomForest)
- Parâmetros específicos do modelo (em formato JSON)
- Horizonte de previsão
- Intervalo de confiança

Os cenários são salvos no arquivo `scenarios_config.yaml`.

3.2. Execução de Previsões

O usuário pode iniciar a execução de todos os cenários configurados. A execução ocorre em uma thread separada para não bloquear a GUI, e os logs de progresso são exibidos em tempo real na interface. Após a execução, os resultados são salvos em um banco de dados SQLite (`previsoes.db`).

- **Execução em Lote:** Permite executar previsões para todos os cenários configurados de uma só vez.
- **Feedback de Progresso:** Exibe uma barra de progresso e mensagens de status durante a execução.
- **Registro de Logs:** Detalhes da execução são registrados em tempo real em uma área de logs na interface.

3.3. Visualização de Resultados

- **Tabela de Resultados:** Exibe os resultados de todas as previsões realizadas em uma tabela organizada.
- **Gráficos Interativos:** Permite visualizar graficamente os dados históricos e as previsões para cenários selecionados.
- **Seleção Múltipla:** Suporte para selecionar múltiplos cenários na tabela e visualizá-los simultaneamente no gráfico para comparação.
- **Comparação de Cenários:** Facilita a análise comparativa entre diferentes modelos ou configurações de cenários através da sobreposição de gráficos.
- **Métricas de Avaliação:** Exibe métricas de desempenho do modelo (RMSE, MAE, MAPE) para cada previsão, permitindo uma análise quantitativa.
- **Frequência da Série:** A frequência inferida da série temporal é armazenada e exibida, auxiliando na compreensão dos dados.

3.4. Exportação de Resultados

- Exportação dos resultados das previsões para arquivos CSV e Excel diretamente pela interface.

4. Arquitetura e Fluxo de Dados

O fluxo de dados segue etapas bem definidas:

1. **Configuração:** Cenários definidos via GUI e salvos em `scenarios_config.yaml`.
2. **Carregamento de Dados:** `data_loader.py` carrega dados históricos de `dados_bcb.db`.
3. **Modelagem e Previsão:** `forecasting_model.py` instancia e executa o modelo apropriado.
4. **Processamento de Resultados:** `results_processor.py` formata os resultados, adicionando metadados e data de execução.
5. **Persistência:** `sqlite_adapter.py` salva os resultados em `previsoes.db`.
6. **Visualização e Exportação:** Resultados exibidos na GUI, com opção de exportação via `data_exporter.py`.

5. Pontos Fortes e Boas Práticas

- **Modularidade:** O código é bem dividido em módulos com responsabilidades claras (GUI, lógica de negócio, persistência, utilitários).
- **Separação de Preocupações:** A lógica da interface (CustomTkinter) está separada da lógica de negócio e persistência.
- **Extensibilidade de Modelos:** A arquitetura com `BaseModel` e `forecast_factory` facilita a adição de novos modelos de previsão no futuro.
- **Persistência de Dados:** Utilização de SQLite para armazenar dados históricos e resultados de previsão, garantindo que os dados sejam mantidos entre as sessões.
- **Configuração Flexível:** O uso de um arquivo YAML para cenários permite fácil configuração e modificação.
- **GUI Responsiva:** A execução de previsões em uma thread separada evita que a interface congele, proporcionando uma melhor experiência ao usuário.
- **Logging:** A presença de um sistema de logging (`logger_config.py`) ajuda no rastreamento de eventos e depuração.
- **Testes Automatizados:** A inclusão de testes unitários para os principais módulos assegura a qualidade e a robustez do código.