1. **utils.py の整備:**
   * 診療科マッピング (create\_dept\_mapping\_table, get\_display\_name\_for\_dept, create\_dept\_display\_options) と病棟マッピング (create\_ward\_name\_mapping, get\_ward\_display\_name, create\_ward\_display\_options) の関数群を完成させ、セッションステートへの保存・読み込みロジックを確立します。
   * initialize\_all\_mappings 関数を実装し、データロード後にこれらのマッピングを初期化するフローを確立します。
2. **data\_processing\_tab.py での初期化呼び出し:**
   * データ処理完了後、st.session\_state.df と st.session\_state.target\_data を引数として initialize\_all\_mappings を呼び出します。
3. **各タブでのUI修正:**
   * 診療科/病棟選択の st.selectbox や st.multiselect で、options 引数には create\_dept\_display\_options や create\_ward\_display\_options から得られる表示オプションリストを使用します。
   * ユーザーの選択結果（表示オプション）を、内部処理用のコードに変換するロジックを組み込みます（option\_to\_code 辞書を利用）。
4. **各タブでの表示修正:**
   * グラフのタイトル、凡例、テーブルのヘッダーやデータセル内で診療科/病棟コードを表示している箇所を特定し、get\_display\_name\_for\_dept や get\_ward\_display\_name を使って表示名に置き換えます。

**注意点:**

* **マッピングのタイミング:** マッピング辞書は、元となるデータ（実績データや目標値ファイル）がロードまたは更新された直後に作成・更新される必要があります。
* **存在しないコードの扱い:** get\_display\_name\_for\_dept や get\_ward\_display\_name は、マッピングに存在しないコードが渡された場合に、デフォルト名（通常はコード自体）を返すようにします。
* **セッションステートのクリア:** データがリセットされたり、新しい目標値ファイルがアップロードされたりした場合には、関連するマッピングのセッションステート (dept\_mapping\_initialized, ward\_mapping\_initialized など）も適切にリセットする必要があります。

**コードレビュー結果と修正提案**

以下に、コード全体を通して見られた改善点や修正が必要と思われる箇所を、重要度が高い順にリストアップします。

**【重要度：高】修正・検討を強く推奨する項目**

1. **preprocess.py と integrated\_preprocessing.py の役割重複・整理**
   * **問題点:** preprocess.py と integrated\_preprocessing.py には、データ前処理に関する類似の関数（例: add\_weekday\_flag）や重複する可能性のあるロジックが含まれています。preprocess\_data 関数も両方に存在し、内容が異なります。
   * **影響:** コードの混乱、保守性の低下、意図しない前処理の実行。
   * **提案:**
     + どちらのファイル/関数を正式な前処理モジュールとするか決定してください。integrated\_preprocessing.py の方が高機能でエラー処理も詳細に見えるため、こちらを主とし、preprocess.py は廃止または必要な関数のみを integrated\_preprocessing.py や utils.py に移植することを検討してください。
     + app.py でインポートされている integrated\_preprocess\_data が主に使われていると推測されますが、他の箇所で preprocess.py が使われていないか確認が必要です。
2. **グローバルキャッシュ (global\_chart\_cache in chart.py) の使用**
   * **問題点:** chart.py で定義されている global\_chart\_cache はグローバル変数であり、Streamlitのセッション管理や再実行の挙動と相性が悪く、意図しないキャッシュの共有やメモリリーク、マルチプロセス/スレッド環境での問題を引き起こす可能性があります。
   * **影響:** アプリケーションの不安定化、予期せぬグラフの表示、メモリ効率の低下。
   * **提案:**
     + Streamlitのセッション状態 (st.session\_state) を使用したキャッシュ管理に移行することを強く推奨します。get\_chart\_cache 関数が st.session\_state.chart\_cache を返しているので、global\_chart\_cache を完全に廃止し、こちらに統一してください。
     + pdf\_generator.py でも同様に st.session\_state.pdf\_chart\_cache が使われているため、キャッシュ戦略を一元化できます。
3. **マルチプロセス時のキャッシュとフォント管理 (batch\_processor.py, pdf\_generator.py)**
   * **問題点:** batch\_processor.py のワーカープロセス (process\_pdf\_in\_worker\_revised) 内では、メインプロセスの st.session\_state に直接アクセスできません。しかし、pdf\_generator.py 内のグラフ生成関数 (create\_alos\_chart\_for\_pdf など) が st.session\_state.pdf\_chart\_cache を参照しようとしています。また、フォント登録もワーカープロセスごとに適切に行われるか注意が必要です。
   * **影響:** キャッシュが機能しない、フォントが見つからずエラーまたはデフォルトフォントでの描画、パフォーマンス低下。
   * **提案:**
     + **キャッシュ:** ワーカープロセス用のキャッシュ戦略を再考する必要があります。
       - 案1: キャッシュを行わない (最もシンプルだがパフォーマンスは犠牲になる)。
       - 案2: ワーカープロセスごとに独立したインメモリキャッシュを持つ (限定的効果)。
       - 案3: joblib.Memory のようなディスクベースのキャッシュをマルチプロセス対応モードで使用する。
       - 現状 pdf\_generator.py の create\_alos\_chart\_for\_pdf などは st.cache\_data を使用していないため、メインプロセスから渡されたバッファを使うか、内部で都度生成する形になっています。alos\_chart\_buffers を渡すアプローチは良いですが、他のグラフについても同様の仕組みが必要か検討してください。
     + **フォント:** pdf\_generator.py の register\_fonts() はモジュールインポート時に実行されるため、各ワーカープロセスで呼び出されるはずです。これは現状で問題ない可能性が高いですが、ログでフォント登録の成功/失敗を確認すると良いでしょう。
4. **冗長な関数の集約 (safe\_date\_filterなど)**
   * **問題点:** safe\_date\_filter 関数が analysis\_tabs.py, alos\_analysis\_tab.py, dow\_analysis\_tab.py, revenue\_dashboard\_tab.py, utils.py の複数ファイルに定義されています。
   * **影響:** コードの重複、修正時の手間増加、潜在的な不整合。
   * **提案:** utils.py に定義されているものを正とし、他のファイルからは from utils import safe\_date\_filter のようにインポートして使用するように統一してください。他の共通関数についても同様の確認を推奨します。

**【重要度：中】修正を推奨する項目**

1. **エラーハンドリングと例外処理の具体性**
   * **問題点:** 一部の except Exception as e: ブロックでは、具体的なエラー型を捕捉していません。また、エラー発生時にユーザーに提示される情報が限定的な場合があります（例: loader.py の read\_excel\_cached で単に print(f"Excel読込エラー: {str(e)}")）。
   * **影響:** デバッグの困難化、ユーザーが問題解決しにくい。
   * **提案:**
     + 可能な範囲で具体的な例外型（例: FileNotFoundError, ValueError, pd.errors.EmptyDataError など）を捕捉してください。
     + ユーザー向けのエラーメッセージは st.error() を使い、問題解決に繋がる情報（例: 「ファイル形式を確認してください」「必要な列名が見つかりません」など）を含めるようにしてください。
     + 開発者向けのより詳細なエラー情報は traceback.format\_exc() を使ってログに出力するか、デバッグモードでのみ表示するようにしてください。
2. **Streamlitウィジェットのキーの重複回避**
   * **問題点:** 複数のタブやセクションで類似のUI要素を使用する場合、キーが重複する可能性があります。例えば、各分析タブ内の日付選択やラジオボタンなど。
   * **影響:** Streamlitのセッションステートが意図通りに動作せず、UIの選択が他のタブに影響するなどのバグ。
   * **提案:** department\_tables\_tab.py のように、キーに department\_type などのプレフィックス/サフィックスを追加して、UI要素のキーが一意になるようにしてください。 他のタブ（analysis\_tabs.py から呼び出される各タブ、individual\_analysis\_tab.py など）も同様の観点で確認してください。
3. **設定値の参照方法の一貫性 (config.py)**
   * **問題点:** config.py で定義された設定値（例: DEFAULT\_TOTAL\_BEDS）は多くのモジュールで利用されていますが、これらの設定値がUI（例: app.py のサイドバー）で変更された場合、その変更がリアルタイムに全モジュールに伝播するわけではありません。多くは st.session\_state を介して共有されていますが、初期値として config.py の値が使われている箇所と、常に st.session\_state を参照する箇所が混在している可能性があります。
   * **影響:** 設定変更が一部機能に反映されない、古い設定値で処理が実行される可能性。
   * **提案:**
     + アプリケーション起動時（app.py の最初）に config.py の値を st.session\_state のデフォルト値として設定し、以降は常に st.session\_state から設定値を取得するように統一してください。
     + app.py の create\_sidebar 関数では、セッションステートとデフォルト値の扱いが比較的適切に行われていますが、他のモジュールでも同様の設計思想を徹底してください。
4. **日付処理の型の一貫性 (datetime.date vs pd.Timestamp)**
   * **問題点:** st.date\_input は datetime.date オブジェクトを返しますが、Pandasの操作では pd.Timestamp が使われることが多いです。これらが混在すると、比較や算術演算で意図しない結果を招くことがあります。例えば dow\_analysis\_tab.py の期間比較部分では、datetime.date と pd.Timestamp の間で変換や比較が頻繁に行われています。
   * **影響:** 期間比較の不具合、予期せぬエラー。
   * **提案:**
     + 関数やモジュール間で日付データを渡す際は、型を統一する（例: 常に pd.Timestamp に変換してから処理を開始する）ことを検討してください。
     + dow\_analysis\_tab.py の期間比較部分では、日付の型変換が注意深く行われていますが、コードが複雑になっています。入力時点で型を統一することで、よりシンプルにできる可能性があります。
5. **メモリ管理 (memory\_manager.py) の実効性と呼び出し箇所**
   * **問題点:** MemoryManager クラスは定義されていますが、実際にアプリケーション全体でどのようにインスタンス化され、check\_memory が定期的に呼び出されているかが不明です。また、グローバルキャッシュのクリアロジック (\_clear\_unused\_caches, \_force\_cleanup) は global\_chart\_cache のようなグローバル変数を前提としていますが、推奨される st.session\_state ベースのキャッシュとは連携していません。
   * **影響:** メモリ管理機能が意図通りに動作しない可能性。
   * **提案:**
     + MemoryManager を使用する場合、app.py のメイン処理ループの適切な箇所で定期的に check\_memory を呼び出すようにしてください。
     + キャッシュクリアのロジックを st.session\_state に対応させてください。
     + Streamlitの組み込みキャッシュ機能 (st.cache\_data, st.cache\_resource) を活用している場合、これらのキャッシュを直接クリアするAPIは限定的なため、MemoryManager での解放は主に st.session\_state 内の大きなオブジェクトや、独自に管理しているキャッシュオブジェクトが対象になります。
6. **forecast.py 内の generate\_filtered\_summaries での st.error 等のUI要素使用**
   * **問題点:** forecast.py の generate\_filtered\_summaries 関数内で st.error, st.warning, st.info が使用されています。この関数はバックグラウンド処理（例: batch\_processor.py のワーカープロセス）からも呼び出される可能性があり、その場合UI要素の呼び出しはエラーになります。
   * **影響:** バックグラウンド処理の失敗。
   * **提案:**
     + generate\_filtered\_summaries のようなコアな処理関数は、UI要素の呼び出しを避け、処理結果やエラー情報を戻り値として返すように修正してください。UIへの表示は、この関数を呼び出す側のフロントエンドに近いモジュール（例: analysis\_tabs.py や individual\_analysis\_tab.py）で行うようにしてください。
     + これは integrated\_preprocessing.py の validation\_results のようなアプローチに近いです。

**【重要度：低】改善の余地がある項目**

1. **列名のハードコーディングと柔軟性**
   * **問題点:** 多くの箇所で '入院患者数（在院）', '総入院患者数' のような具体的な列名がハードコーディングされています。analysis\_tabs.py の calculate\_ward\_summary などでは、ある程度の柔軟性を持たせようとしていますが、全体的な一貫性が求められます。
   * **影響:** 入力データの列名が少しでも異なると処理が失敗する。
   * **提案:**
     + config.py に主要な列名のマッピング（標準名: [想定される別名のリスト]）を定義し、データ読み込み後、早い段階で列名を標準化する処理（例: integrated\_preprocessing.py 内）を入れることを検討してください。
     + loader.py の read\_excel\_cached に列名マッピングのロジックの一部が見られますが、これをより汎用的にし、アプリケーション全体で標準化された列名を使用するようにしてください。
2. **utils.py の create\_dept\_mapping\_table と get\_display\_name\_for\_dept**
   * **問題点:** 診療科マッピングのロジックで、special\_mappings のようなハードコーディングされた対応が含まれています。これは特定のデータセットに依存する可能性があります。
   * **影響:** 異なる病院やデータソースに対応しにくい。
   * **提案:**
     + 可能であれば、このような特殊マッピングも外部ファイル（例: 目標値ファイルや別途用意するマッピングファイル）から読み込めるようにすると、より柔軟性が増します。
     + 現状のままでも問題ない場合は、コメントでこの特殊対応の意図を明確に記述しておくと良いでしょう。
3. **data\_processing\_tab.py の目標値抽出ロジックの複雑性**
   * **問題点:** extract\_targets\_from\_file 関数が、複数の列名パターンやキーワードで目標値行を検索しており、複雑になっています。
   * **影響:** 維持管理の困難性、新しい目標値ファイル形式への対応の難しさ。
   * **提案:**
     + 目標値ファイルのフォーマットをある程度標準化し、ユーザーにそのフォーマットに従うようガイドすることで、抽出ロジックを簡素化できないか検討してください。
     + 現状のままであれば、各検索パターンと優先順位についてコメントを充実させると良いでしょう。
4. **pdf\_generator.py のグラフ生成とキャッシュ**
   * **問題点:** PDF生成時にMatplotlibでグラフを都度生成しており、キャッシュ機構も独自に実装されています。create\_patient\_chart\_with\_target\_wrapper は chart.py の関数をラップしつつ、目標値描画のために再実装に近い形になっています。
   * **影響:** 処理速度の低下、キャッシュ管理の複雑化。
   * **提案:**
     + chart.py のMatplotlib用グラフ関数 (create\_patient\_chart, create\_dual\_axis\_chart) が、目標値描画やPDF出力に必要なカスタマイズ（フォントサイズ、解像度など）を直接サポートできるように拡張し、pdf\_generator.py からはこれらの関数を直接呼び出すようにすることを検討してください。これにより、グラフ生成ロジックが一元化され、キャッシュも chart.py の @st.cache\_data に任せることができます。
     + alos\_chart\_buffer のように、生成済みのグラフ（BytesIOオブジェクト）を引数で渡す方式は、グラフ生成ロジックとPDF構築ロジックを分離する良いアプローチです。これを他のグラフにも適用できないか検討してください。
5. **未使用の可能性のあるインポートや変数**
   * **問題点:** いくつかのファイルで、import gc や import time されているものの、直接使用されていない箇所があるかもしれません（例: department\_tables\_tab.py の time）。また、定義されているが使われていないローカル変数など。
   * **影響:** コードの軽微なノイズ。
   * **提案:** コード全体を見直し、不要なインポートや変数を削除してください。
6. **定数と設定値の管理**
   * **問題点:** data\_processing\_tab.py 内に EXCEL\_USE\_COLUMNS や EXCEL\_DTYPES のような定数が定義されています。
   * **影響:** 設定値が分散し、管理しにくい。
   * **提案:** このような設定値も config.py に集約することを検討してください。

**総括**

全体として非常によく構造化され、多機能なアプリケーションコードであると感じました。特に、各機能がモジュール化されている点、エラーハンドリングやキャッシュによるパフォーマンス向上が試みられている点は素晴らしいです。

上記の提案は、主にコードの堅牢性、保守性、パフォーマンスをさらに向上させるためのものです。特に【重要度：高】の項目から対応していただくことで、アプリケーションの品質が大きく向上すると期待できます。