🧬 Project Context: Biosensor Pocket Explorer

This project is a Streamlit-based interactive tool for exploring fpocket-derived protein pockets and matching them with specific analytes. The tool integrates protein structure viewing, ligand property interpretation, ligand–pocket compatibility scoring, docking preparation, and downloadable docking packages.

🔧 Project Structure
Root directory contains:
- `scripts/pocket_gui.py` → main Streamlit app (core UI)
- `scripts/prepare_docking_setup.py` → generates docking boxes, ligand SDF, docking_targets.csv
- `structures/` → AlphaFold protein structures (`AF-<uniprot>.pdb`)
- `results/pockets_fpocket.csv` → fpocket-derived pocket table
- `docking/` → auto-generated analyte folders containing:
  - `<analyte>/docking_targets.csv`
  - `<analyte>/*_box.txt`
  - `ligands/<analyte>.sdf`
  - `docked/<prot>_pocket<pocket>_pose.pdb`

🧠 Key Concepts
- Users choose an analyte (glucose, dopamine, cortisol…)
- App fetches physicochemical properties from PubChem
- If no PubChem data exists, the system uses heuristic analyte profiles
- Pockets are filtered using volume, druggability, hydrophobicity, etc.
- A compatibility score matches ligand properties to pocket properties
- A best pocket is selected and displayed with a py3Dmol viewer
- The UI allows:
  - Automatic docking setup
  - Auto-packaging docking files into a ZIP (one-button workflow)
  - Displaying ligand 3D structure (local SDF or PubChem fallback)
  - Downloading docking packages

🎯 Coding Style Requirements
- Use Streamlit best practices (cache_data for heavy operations)
- Avoid PyArrow conversion errors:
  - Always convert object columns to strings before st.dataframe()
  - Use helper function `to_streamlit_df()`
- UI must stay responsive, clean, and simple for non-expert users
- Whenever a button performs a workflow, do it in one click
  (e.g., docking setup + ZIP generation + download)
- 3D visualization uses py3Dmol (no plotly/molstar)
- Maintain soft pastel color-coded pills for ligand compatibility

📦 Docking Workflow Notes
- `prepare_docking_setup.py` receives:

  --analyte <name>
  --top_n <int>
- It creates a folder: `docking/<analyte>/`
- It outputs:
  - ligand.sdf (downloaded or generated)
  - N pocket box files named `<protein>_pocket<pocket>_box.txt`
  - docking_targets.csv with pocket metadata
- UI button must:
  - Run prepare_docking_setup.py
  - Build a ZIP archive containing all output artefacts
  - Return a download_button for that ZIP
  - Show errors from stderr/stdout if script fails

🧪 Testing Requirements
- Use “dummy docked pose” button to simulate a ligand pose
- Viewer should overlay:
  - protein (cartoon)
  - pocket atoms (stick + sphere)
  - ligand pose if present (stick colored by element)

🧹 Common Bugs to Avoid
- PyArrow ArrowTypeError when st.dataframe() receives non-string object columns
- Missing analyte due to undefined `interpreted_name` or `raw_analyte`
- Path confusion between local ligand and PubChem-fetched ligand
- Duplicate dataframe rendering
- Docking folder not created before writing files

🆘 What Codex Should Do
Codex should assist with:
- Simplifying and cleaning Streamlit components
- Debugging dataframe conversion issues
- Improving the docking packaging logic
- Adding progress indicators and UX improvements
- Generating missing scripts or viewers
- Refactoring large functions into manageable helpers
- Optimizing performance with caching
- Ensuring paths, file names, analyte identifiers stay consistent

The assistant should NEVER break:
- the docking folder naming format (`my_analyte` → underscores)
- SDF fallback logic (local → PubChem)
- Dataframe_column cleanup before rendering

✔️ End of Context