Geotechnical Data Analysis Application
Entry Points & Session Management

main_app.py
main()

auth.py
check_password()

initialize_session_state()

Session State Structure:
• data_loaded: bool - Controls UI visibility
• lab_data: pd.DataFrame - Raw uploaded data (cached)
• bh_data: pd.DataFrame - Optional BH interpretation data
• filtered_data: pd.DataFrame - Globally filtered data
• test_availability: dict - Cached test counts
• plot_display_settings: dict - Plot size controls

render_file_upload()
load_and_validate_data()

render_simple_filters()
apply_global_filters()

Tab Rendering (13 tabs)
Each tab: render_[analysis]_tab()

Key Files & Responsibilities:

main_app.py: Application controller, session state, tab orchestration
auth.py: Password authentication with session persistence
utils/data_processing.py: Core data operations (load_and_validate_data, apply_global_filters)
utils/comprehensive_analysis.py: CBR/WPI analysis (render_cbr_wpi_analysis_tab)
utils/atterberg_analysis.py: Plasticity analysis (render_atterberg_analysis_tab)
utils/psd_analysis.py: Particle size distribution (render_psd_analysis_tab)
utils/spt_analysis.py: SPT analysis (render_spt_analysis_tab)
utils/spatial_analysis.py: Spatial analysis (render_property_depth_tab, render_property_chainage_tab)
Functions/: 21 plotting functions (original Jupyter notebook logic)
- plot_CBR_swell_WPI_histogram.py: Core CBR/WPI plotting (500+ lines)
- plot_histogram.py: General histogram plotting
- plot_atterberg_chart.py: Plasticity charts

Data Flow and Processing Pipeline

Stage 1: File Upload

File Upload →
load_and_validate_data()
@st.cache_data

Stage 2: Session Storage

st.session_state.lab_data
st.session_state.bh_data
(Persistent Cache)

Stage 3: Global Filtering

apply_global_filters()
Depth/Chainage/Geology
Filters

Stage 4: Test Analysis

get_test_availability() @st.cache_data
extract_test_columns()
get_id_columns_from_data()

Stage 5: Tab-Specific Processing

Each tab processes filtered_data
Tab-specific data preparation
Parameter collection

CBR/WPI Specific Data Processing Pipeline:

filtered_data → prepare_cbr_wpi_data() → plot_CBR_swell_WPI_histogram() → Streamlit Display

Key Steps:
1. Extract CBR Data: Find CBR column, filter non-null values
2. Extract WPI Data: Find WPI column, filter non-null values
3. Add Categories: Apply thresholds (Low/Moderate/High/Very high/Extreme)
4. Add Cut Categories: Above Cut/Below Cut based on depth_cut parameter
5. Add Map Symbol: Include map_symbol column for stacking
6. Select Columns: Keep only ['Name', 'Geology_Orgin', 'category', map_symbol, 'Cut_Category']
7. Concatenate: Combine CBR and WPI datasets

Performance Characteristics:

• File Upload: ~1-2 seconds (cached after first load)
• Global Filtering: ~200-500ms (depends on data size)
• prepare_cbr_wpi_data(): ~500ms-1s per execution (NOT cached - runs on every parameter change)
• plot_CBR_swell_WPI_histogram(): ~1-2 seconds (complex matplotlib generation)
• Total CBR/WPI tab render time: ~2-4 seconds per parameter change

⚠ BOTTLENECK: Complete reprocessing on any parameter change

Data Persistence Strategy:

Session State Management:
• Raw data: Cached in session_state.lab_data (persistent across interactions)
• Filtered data: Stored in session_state.filtered_data (updated on global filter changes)
• Test availability: Cached to avoid recalculation
• Plot settings: Stored in session_state.plot_display_settings

Caching Strategy:
• @st.cache_data on load_and_validate_data() - avoids re-reading files
• @st.cache_data on get_test_availability() - avoids recounting tests
• No caching on tab-specific processing (opportunity for optimization)

Tab Architecture and Rendering Patterns

13 Analysis Tabs (All rendered simultaneously - Performance Issue):
Data | PSD | Atterberg | SPT | Emerson | UCS vs Depth | UCS vs Is50 | Property vs Depth | Property vs Chainage | Thickness Analysis | Histograms | CBR Swell/WPI | Export

Data
render_data_overview()

PSD
render_psd_analysis_tab()

Atterberg
render_atterberg_analysis_tab()

SPT
render_spt_analysis_tab()

Emerson
render_emerson_analysis_tab()

UCS Depth
render_ucs_depth_tab()

UCS Is50
render_ucs_is50_tab()

Prop Depth
render_property_depth_tab()

Prop Chainage
render_property_chainage_tab()

Thickness
render_thickness_analysis_tab()

Histograms
render_comprehensive_histograms_tab()

CBR/WPI □
render_cbr_wpi_analysis_tab()

Export
render_batch_export_tab()

Common Tab Rendering Pattern:
def render_[analysis]_tab(filtered_data: pd.DataFrame):
1. Parameter Collection → st.expander with form controls (UI state stored in widget keys)
2. Data Processing → Specific to analysis type (e.g., prepare_cbr_wpi_data())
3. Plotting → Calls Functions/ folder functions (e.g., plot_CBR_swell_WPI_histogram())
4. Download Button → Create matplotlib figure download
5. Optional → Statistics/preview display

⚠ Current Performance Issues:

1. All Tabs Rendered Simultaneously: Every parameter change triggers main() rerun → ALL 13 tabs re-render
2. No Tab Isolation: Changing CBR/WPI parameters affects entire application state
3. No Lazy Loading: Inactive tabs still process data and create UI elements
4. Heavy Computation Blocking: Long-running plotting operations block UI
5. No Caching Between Tabs: Each tab recalculates shared data (test_availability, id_columns)

□ Optimization Opportunities:
• Implement lazy tab loading (only render active tab)
• Add tab-specific state management
• Cache shared computations
• Add progressive loading with spinners

CBR/WPI Analysis Tab - Detailed Workflow

render_cbr_wpi_analysis_tab()

Parameter Box Structure (5 rows × 5 columns = 25+ parameters):
Row 1: Analysis Type | Depth Cut | Stack By | Category Order | Facet Order
Row 2: Filter 1 By | Filter 1 Value | Filter 2 By | Filter 2 Value | [Empty]
Row 3: Figure Size | X-Axis Limits | Y-Axis Limits | Custom Title | Custom Y-Label
Row 4: Colormap | Alpha | Show Grid | Show Legend | [Empty]
Advanced: 24 additional parameters in collapsed expander

Data Processing Pipeline:

1. prepare_cbr_wpi_data(filtered_data, depth_cut):
• Extract CBR data → Apply thresholds → Create categories
• Extract WPI data → Apply thresholds → Create categories
• Add Cut_Category (Above Cut/Below Cut based on depth_cut)
• Add map_symbol column for stacking
• Select columns: ['Name', 'Geology_Orgin', 'category', map_symbol, 'Cut_Category']
• Concatenate CBR and WPI datasets

2. Filter by analysis_type (CBR only/WPI only/Combined)

3. Apply additional filters (Filter 1 & 2)

Plotting Stage:

plot_CBR_swell_WPI_histogram(processed_data, **all_parameters):
• Complex matplotlib figure generation (500+ lines of code)
• Stacked bar chart creation with categorical data
• Advanced styling: colors, fonts, grids, legends, layout
• Multiple subplot handling for different analysis types
• Export-ready high-resolution figure generation

Display and Interaction:

• Streamlit plot display with download button
• Test distribution charts (CBR and WPI vs chainage)
• Two-column layout: Data Preview (left) | Statistics (right)
• Optional statistics table and data preview table
• Download controls for plot export

⚠ Performance Analysis:

Execution Times (per parameter change):
• prepare_cbr_wpi_data(): 500ms - 1s (complex data processing)
• plot_CBR_swell_WPI_histogram(): 1-2s (matplotlib generation)
• Test distribution rendering: 300-500ms
• Statistics calculation: 100-200ms
• Total: 2-4 seconds per parameter change

Critical Issues:
• No caching of processed data
• Complete reprocessing on any parameter change
• Heavy matplotlib operations on every update
• No distinction between light vs heavy parameter changes

Parameter Dependencies and Impact Analysis

 LIGHT PARAMETERS
(UI-only changes)

• stack_by
• analysis_type
• plot styling (colors, fonts)
• show_grid, show_legend
• alpha, colormap
• axis labels, titles

Impact: Keep processed data, only re-plot (500ms)

 MEDIUM PARAMETERS
(data filtering)

• Filter 1/2 (by/value)
• facet_order
• category_order
• xlim, ylim
• figure_size

Impact: Keep base processed data, apply filters, re-plot (800ms - 1.2s)

 HEAVY PARAMETERS
(complete reprocessing)

• depth_cut
(triggers Cut_Category recalculation)

Impact: Complete data reprocessing from scratch (2-4 seconds)

CURRENT BEHAVIOR: All parameters trigger complete workflow

ANY parameter change → prepare_cbr_wpi_data() → plot_CBR_swell_WPI_histogram() → full re-render
Result: Changing 'alpha' takes same time as changing 'depth_cut' (2-4 seconds)

OPTIMIZED BEHAVIOR: Intelligent parameter change detection

Light params → cached_data → re-plot only (500ms)
Medium params → cached_base_data → filter → re-plot (800ms)
Heavy params → full reprocessing (2s)

 IMPLEMENTATION STRATEGY:

1. Parameter Change Detection:
• Track previous parameter state in session_state
• Compare current vs previous to determine change type
• Route to appropriate processing pathway

2. Intelligent Caching:
@st.cache_data
def prepare_cbr_wpi_data_cached(data_hash, depth_cut):
# Cache by depth_cut value

@st.cache_data
def generate_plot_cached(processed_data_hash, plot_params_hash):
# Cache plots by parameter combinations

3. Progressive Enhancement:
• Show lightweight preview immediately
• Load full plot with loading spinner
• Allow cancellation of expensive operations

 EXPECTED BENEFITS:

Performance Improvements:
• Light parameter changes: 70% faster (2-4s → 500ms)
• Medium parameter changes: 40% faster (2-4s → 800ms-1.2s)
• Heavy parameter changes: Same speed but isolated impact
• Overall user experience: 3-5x more responsive

User Experience:
• Immediate feedback for styling changes
• Predictable response times
• Better understanding of parameter impact
• Reduced frustration with interface responsiveness

Performance Bottlenecks and Optimization Roadmap

🚨 CRITICAL PERFORMANCE BOTTLENECKS:

1. Complete App Rerun (2-3s): Any parameter change triggers full main() → ALL tabs re-render
2. Heavy Data Processing (500ms-1s): prepare_cbr_wpi_data() runs on every change
3. Complex Plotting (1-2s): plot_CBR_swell_WPI_histogram() 500+ lines, complete regeneration
4. No Parameter Isolation: Light changes (colors) = Heavy changes (depth_cut) impact
5. Redundant Session State Operations (100-200ms): Unnecessary state updates
6. No Tab Caching: Inactive tabs still consume resources

🔥 PHASE 1: Critical Fixes
(Week 1 - High Impact)

🎯 Add @st.cache_data to
prepare_cbr_wpi_data()
🎯 Parameter change detection
🎯 Loading indicators
🎯 Fix depth_cut variable error

Expected: 70% improvement

⚡ PHASE 2: Smart Optimization
(Week 2 - Medium Impact)

• Plot-level caching
• Tab state isolation
• Progressive enhancement
• Lazy tab loading

Expected: 50% additional
improvement

🚀 PHASE 3: Advanced Features
(Week 3 - Long-term)

• Async processing
• Pre-computation
• Incremental updates
• Background caching

Expected: Additional
performance polish

📋 DETAILED IMPLEMENTATION PLAN:

Immediate Optimizations:
1. Smart Caching:
@st.cache_data(hash_funcs={pd.DataFrame: lambda df: df.shape})
def prepare_cbr_wpi_data_cached(data_hash, depth_cut, map_symbol_col):
return prepare_cbr_wpi_data(filtered_data, depth_cut)

2. Parameter Change Detection:
if 'cbr_wpi_previous_params' not in st.session_state:
st.session_state.cbr_wpi_previous_params = {}

current_params = {'depth_cut': depth_cut, 'analysis_type': analysis_type, ...}
changed_params = {k: v for k, v in current_params.items()
if k not in st.session_state.cbr_wpi_previous_params
or st.session_state.cbr_wpi_previous_params[k] != v}

3. Conditional Processing:
if heavy_params_changed(['depth_cut']):
data = prepare_cbr_wpi_data_cached(...) # Full reprocessing
elif medium_params_changed(['filter1', 'filter2']):
data = apply_filters_only(cached_base_data, ...) # Filter only
else:
data = cached_processed_data # Use existing data

📊 EXPECTED PERFORMANCE RESULTS:

Current State:
• Any parameter change: 2-4 seconds
• User frustration with slow response
• Poor development experience

After Phase 1 (Week 1):
• Light parameter changes: 500ms (70% improvement)
• Heavy parameter changes: 2s (still need full processing)
• Much better user experience

After Phase 2 (Week 2):
• Light parameter changes: 200ms (90% improvement)
• Medium parameter changes: 800ms (60% improvement)
• Heavy parameter changes: 1.5s (25% improvement)
• Excellent responsiveness

After Phase 3 (Week 3):
• Near-instant response for cached scenarios
• Background processing for complex operations
• Professional-grade application performance

🎯 **IMPLEMENTATION PRIORITY ORDER:**
1. Fix depth_cut error (DONE ✅)
2. Add caching to prepare_cbr_wpi_data()
3. Implement parameter change detection
4. Add loading indicators for operations >500ms
5. Plot-level caching by parameter hash
6. Tab state isolation
7. Lazy tab loading

GOAL: 3-5x faster response times with better user experience