```
"""
Pilot Trip Scheduling Analysis - Streamlit App
Analyzes trip scheduling data for airline pilot bids
"""

import streamlit as st
import pandas as pd
import plotly.express as px
from datetime import datetime
import analysis_engine
import hashlib

# Page config
st.set_page_config(
    page_title="Pilot Trip Analysis",
    page_icon="✈️",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Password Protection
def check_password():
    """Returns `True` if the user had the correct password."""

    def password_entered():
        """Checks whether a password entered by the user is correct."""
        if "APP_PASSWORD" in st.secrets:
            correct_password = st.secrets["APP_PASSWORD"]
        else:
            # Fallback default password if no password set in secrets
            correct_password = "pilot2026"

        if st.session_state["password"] == correct_password:
            st.session_state["password_correct"] = True
            del st.session_state["password"]  # Don't store password
        else:
            st.session_state["password_correct"] = False

    # First run, show input for password
    if "password_correct" not in st.session_state:
        st.markdown("# 🔒 Pilot Trip Scheduling Analysis")
        st.markdown("### Please enter the password to access the application")
        st.text_input(
            "Password",
            type="password",
            on_change=password_entered,
            key="password"
        )
        st.info("💡 Contact your administrator for the access password")
        return False

    # Password incorrect, show input + error
    elif not st.session_state["password_correct"]:
        st.markdown("# 🔒 Pilot Trip Scheduling Analysis")
        st.markdown("### Please enter the password to access the application")
        st.text_input(
            "Password",
            type="password",
            on_change=password_entered,
            key="password"
        )
        st.error("😕 Password incorrect. Please try again.")
        return False

    # Password correct
    else:
        return True

if not check_password():
    st.stop()  # Don't continue if check_password is not True

# Main application starts here (password verified)
# Initialize session state
if 'uploaded_files' not in st.session_state:
    st.session_state.uploaded_files = {}
if 'analysis_results' not in st.session_state:
    st.session_state.analysis_results = {}
if 'file_counter' not in st.session_state:
    st.session_state.file_counter = 0

def get_file_hash(content):
    """Generate unique hash for file content"""
    return hashlib.md5(content.encode()).hexdigest()[:8]

# Sidebar
st.sidebar.title("✈️ Trip Analysis Settings")

# Time selectors
st.sidebar.subheader("Commutability Settings")
time_options = [f"{h:02d}:{m:02d}" for h in range(24) for m in [0, 30]]
time_to_minutes = {t: int(t[:2])*60 + int(t[3:]) for t in time_options}
```

```
front_end_time = st.sidebar.selectbox(
    "Front End Commutable (Report ≥)",
    time_options,
    index=time_options.index("10:30"),
    key='sidebar_front_time'
)

back_end_time = st.sidebar.selectbox(
    "Back End Commutable (Release ≤)",
    time_options,
    index=time_options.index("18:00"),
    key='sidebar_back_time'
)

# Checkbox for including short trips in commutability
include_short_commute = st.sidebar.checkbox(
    "Include 1-2 Day Trips in Commutability",
    value=False,
    key='include_short_commute',
    help="By default, commutability analysis only includes trips 3+ days. Check t
)

# Base filter
st.sidebar.subheader("Base Filter")
base_options = ["All Bases", "ATL", "BOS", "NYC", "DTW", "SLC", "MSP", "SEA", "LA
selected_base = st.sidebar.selectbox("Select Base", base_options, key='sidebar_ba

# Update Analysis button
st.sidebar.markdown("---")
if st.session_state.uploaded_files and st.sidebar.button("🔄 Update Analysis", ty
    with st.spinner("Updating analysis with new settings..."):
        st.session_state.analysis_results = {}
        st.session_state.detailed_trips = {}  # Clear detailed trips cache too

        front_minutes = time_to_minutes[front_end_time]
        back_minutes = time_to_minutes[back_end_time]

        for fname, fdata in st.session_state.uploaded_files.items():
            result = analysis_engine.analyze_file(
                fdata['content'],
                selected_base,
                front_minutes,
                back_minutes,
                include_short_commute,
                fdata['year']  # Pass the bid year from uploaded file
            )
            st.session_state.analysis_results[fname] = result

        st.success("✅ Analysis updated!")
        st.rerun()

# Clear button
if st.sidebar.button("🗑️ Clear All Data", type="primary", key='sidebar_clear'):
    st.session_state.uploaded_files = {}
    st.session_state.analysis_results = {}
    st.session_state.file_counter = 0
    st.rerun()

# Main title
st.title("✈️ Pilot Trip Scheduling Analysis")
st.markdown("Upload trip schedule files to analyze metrics including trip length,

# File upload section
st.header("📁 Upload Schedule Files")

# Show current file count
col1, col2 = st.columns([3, 1])
with col2:
    st.metric("Files Loaded", len(st.session_state.uploaded_files))
    if len(st.session_state.uploaded_files) >= 12:
        st.warning("⚠️ Maximum 12 files")

# File uploader with form to avoid key conflicts
with col1:
    if len(st.session_state.uploaded_files) < 12:
        uploaded_files = st.file_uploader(
            "Upload trip schedule files (.txt)",
            type=['txt'],
            accept_multiple_files=True,
            key=f'file_uploader_{st.session_state.file_counter}'
        )

        if uploaded_files:
            for uploaded_file in uploaded_files:
                # Read content immediately
                content = uploaded_file.read().decode('utf-8')
                file_hash = get_file_hash(content)

                # Check if this exact file content is already uploaded
                already_exists = False
                for existing_name, existing_data in st.session_state.uploaded_fil
                    if get_file_hash(existing_data['content']) == file_hash:
                        already_exists = True
                        st.warning(f"⚠️ File '{uploaded_file.name}' (same content
                        break

                # Also check if there's a pending form for this exact content
                if not already_exists:
                    # Show date selection form
                    with st.form(key=f'date_form_{file_hash}'):
                        st.subheader(f"📅 Set Date for: {uploaded_file.name}")

                        col_m, col_y = st.columns(2)
                        with col_m:
                            month = st.selectbox(
                                "Month",
                                ["January", "February", "March", "April", "May",
                                 "July", "August", "September", "October", "Novem
                                index=0
                            )
                        with col_y:
                            year = st.number_input(
                                "Year",
                                min_value=2020,
                                max_value=2030,
                                value=2026
                            )

                        # Fleet field (optional)
                        fleet = st.text_input(
                            "Fleet (optional)",
                            placeholder="e.g., 320, 737, A220",
                            help="Add a fleet identifier to differentiate files f
                        )

                        submitted = st.form_submit_button("✅ Add File")

                        if submitted:
                            # Create unique filename based on date and optional f
                            # Extract base name without extension
                            base_name = uploaded_file.name.rsplit('.', 1)[0] if '
                            extension = uploaded_file.name.rsplit('.', 1)[1] if '

                            # Create new filename: basename_MMYYYY_fleet.ext or b
                            month_num = {
                                'January': '01', 'February': '02', 'March': '03',
                                'May': '05', 'June': '06', 'July': '07', 'August'
                                'September': '09', 'October': '10', 'November': '
                            }[month]
```

```
                            if fleet:
                                new_filename = f"{base_name}_{month_num}{year}_{f
                                display_name = f"{month} {year} ({fleet})"
                            else:
                                new_filename = f"{base_name}_{month_num}{year}.{e
                                display_name = f"{month} {year}"

                            # Add to uploaded files with new filename
                            st.session_state.uploaded_files[new_filename] = {
                                'content': content,
                                'month': month,
                                'year': year,
                                'fleet': fleet if fleet else None,
                                'display_name': display_name,
                                'original_name': uploaded_file.name
                            }
                            st.session_state.file_counter += 1
                            st.success(f"✅ Added '{uploaded_file.name}' as '{new
                            st.rerun()
    else:
        st.info("Maximum of 12 files reached. Remove files to add more.")

# Display loaded files
if st.session_state.uploaded_files:
    st.subheader("📋 Loaded Files")

    for fname, fdata in st.session_state.uploaded_files.items():
        col1, col2, col3 = st.columns([3, 1, 1])
        with col1:
            st.text(fdata['display_name'])
        with col2:
            st.text(f"Base: {selected_base}")
        with col3:
            if st.button("❌ Remove", key=f"remove_{fname}_{get_file_hash(fdata['
                del st.session_state.uploaded_files[fname]
                if fname in st.session_state.analysis_results:
                    del st.session_state.analysis_results[fname]
                st.rerun()

# Run analysis button
if st.session_state.uploaded_files:
    if st.button("🔍 Analyze Data", type="primary", key='analyze_button'):
        with st.spinner("Analyzing trip data..."):
            st.session_state.analysis_results = {}

            front_minutes = time_to_minutes[front_end_time]
            back_minutes = time_to_minutes[back_end_time]
            for fname, fdata in st.session_state.uploaded_files.items():
                result = analysis_engine.analyze_file(
                    fdata['content'],
                    selected_base,
                    front_minutes,
                    back_minutes,
                    include_short_commute,
                    fdata['year']  # Pass the bid year from uploaded file
                )
                st.session_state.analysis_results[fname] = result

            st.success("✅ Analysis complete!")
            st.rerun()

# Display results
if st.session_state.analysis_results:
    st.header("📊 Analysis Results")

    if len(st.session_state.analysis_results) == 1:
        # Single file - show summary or detailed view based on toggle
        fname = list(st.session_state.analysis_results.keys())[0]
        result = st.session_state.analysis_results[fname]
        fdata = st.session_state.uploaded_files[fname]

        st.subheader(f"Analysis: {fdata['display_name']}")

        # View toggle
        view_mode = st.radio(
            "View Mode",
            ["Summary", "Detailed Trip Table"],
            horizontal=True,
            key='view_mode_toggle'
        )

        if view_mode == "Summary":
            # AI Chat Section for Summary View
            with st.expander("💬 Ask AI About This Analysis", expanded=False):
                st.markdown("Ask questions about the summary statistics and metri
                st.markdown("*Examples: 'What does this data tell me?', 'Are 3-da

                # Try to get API key from secrets, then from session state, then
                api_key_summary = None
                if "ANTHROPIC_API_KEY" in st.secrets:
                    api_key_summary = st.secrets["ANTHROPIC_API_KEY"]
                    st.success("✅ Using API key from Streamlit secrets")
                else:
```

```
    # API Key input
    api_key_summary = st.text_input(
        "Anthropic API Key",
        type="password",
        help="Get your API key at https://console.anthropic.com,
        key="anthropic_api_key_summary",
        value=st.session_state.get('saved_api_key', '')
    )

    if api_key_summary and api_key_summary != st.session_state.ge
        st.session_state.saved_api_key = api_key_summary
        st.info("💡 API key saved for this session. To save perma

# Question input
user_question_summary = st.text_area(
    "Your Question",
    placeholder="e.g., What's the best trip length for maximizing
    height=80,
    key="ai_question_summary"
)

if st.button("Ask AI", type="primary", key="ask_ai_summary"):
    if not api_key_summary:
        st.error("Please enter your Anthropic API key first")
    elif not user_question_summary:
        st.error("Please enter a question")
    else:
        with st.spinner("Analyzing..."):
            try:
                import anthropic

                # Prepare summary data for AI
                summary_data = {
                    'file': fdata['display_name'],
                    'total_trips': result['total_trips'],
                    'avg_trip_length': result['avg_trip_length'],
                    'avg_credit_per_trip': result['avg_credit_per
                    'avg_credit_per_day': result['avg_credit_per_
                    'trip_counts_by_length': result['trip_counts'
                    'avg_credit_by_length': result['avg_credit_by
                    'avg_credit_per_day_by_length': result['avg_c
                    'single_leg_last_day_pct': result['single_leg
                    'redeye_pct': result['redeye_pct'],
                    'front_end_commutable_pct': result['front_com
                    'back_end_commutable_pct': result['back_commu
                }
```

```
                                # Call Claude API
                                client = anthropic.Anthropic(api_key=api_key_summ
                                message = client.messages.create(
                                    model="claude-sonnet-4-20250514",
                                    max_tokens=2000,
                                    messages=[{
                                        "role": "user",
                                        "content": f"""You are analyzing pilot tr

{summary_data}

The user's question is: {user_question_summary}

Please provide a helpful, concise answer based on this data. Explain patterns and
                                    }]
                                )

                                # Display response
                                st.success("✨ AI Analysis:")
                                st.markdown(message.content[0].text)

                            except ImportError:
                                st.error("❌ Anthropic library not installed. Run
                            except Exception as e:
                                st.error(f"❌ Error: {str(e)}")

            # SUMMARY VIEW (existing code)
            # Summary metrics
            col1, col2, col3, col4 = st.columns(4)
            with col1:
                st.metric("Total Trips", result['total_trips'])
            with col2:
                st.metric("Avg Trip Length", f"{result['avg_trip_length']:.2f} da
            with col3:
                st.metric("Avg Credit/Trip", f"{result['avg_credit_per_trip']:.2f
            with col4:
                st.metric("Avg Credit/Day", f"{result['avg_credit_per_day']:.2f}

            col1, col2, col3 = st.columns(3)
            with col1:
                st.metric("Front-End Commute", f"{result['front_commute_rate']:.1
            with col2:
                st.metric("Back-End Commute", f"{result['back_commute_rate']:.1f}
            with col3:
                st.metric("Both Ends Commute", f"{result['both_commute_rate']:.1f

            # Charts in tabs
```

```
tab1, tab2, tab3, tab4, tab5, tab6, tab7 = st.tabs([
    "Trip Length", "Single Leg Last Day", "Credit/Trip",
    "Credit/Day", "Commutability", "Red-Eye Trips", "Staffing Heat Ma
])

with tab1:
    # Calculate percentages for each trip length
    total_trips = result['total_trips']
    trip_counts = [result['trip_counts'][i] for i in range(1, 6)]
    trip_percentages = [(count / total_trips * 100) if total_trips >

    # Create bar chart with count and percentage
    data = pd.DataFrame({
        'Length': [f"{i}-day" for i in range(1, 6)],
        'Count': trip_counts,
        'Percentage': trip_percentages
    })

    fig = px.bar(
        data,
        x='Length',
        y='Count',
        labels={'Length': 'Trip Length', 'Count': 'Number of Trips'},
        title='Trip Length Distribution',
        text=[f"{count}<br>({pct:.1f}%)" for count, pct in zip(trip_c
    )
    fig.update_traces(textposition='outside')
    st.plotly_chart(fig, use_container_width=True)

with tab2:
    data = pd.DataFrame({
        'Length': [f"{i}-day" for i in range(1, 6)],
        'Percentage': [result['single_leg_pct'][i] for i in range(1,
    })
    fig = px.bar(data, x='Length', y='Percentage',
                title='Trips with Single Leg on Last Day (%)')
    st.plotly_chart(fig, use_container_width=True)

with tab3:
    data = pd.DataFrame({
        'Length': [f"{i}-day" for i in range(1, 6)],
        'Hours': [result['avg_credit_by_length'][i] for i in range(1,
    })
    fig = px.bar(data, x='Length', y='Hours',
                title='Average Credit Hours per Trip')
    st.plotly_chart(fig, use_container_width=True)
```

```
with tab4:
    data = pd.DataFrame({
        'Length': [f"{i}-day" for i in range(1, 6)],
        'Hours/Day': [result['avg_credit_per_day_by_length'][i] for i
    })
    fig = px.bar(data, x='Length', y='Hours/Day',
                title='Average Credit Hours per Day')
    st.plotly_chart(fig, use_container_width=True)

with tab5:
    data = pd.DataFrame({
        'Length': [f"{i}-day" for i in range(1, 6)] * 3,
        'Percentage': [result['front_commute_pct'][i] for i in range(
                     [result['back_commute_pct'][i] for i in range(1,
                     [result['both_commute_pct'][i] for i in range(1,
        'Type': ['Front End']*5 + ['Back End']*5 + ['Both Ends']*5
    })
    fig = px.bar(data, x='Length', y='Percentage', color='Type',
                barmode='group', title='Commutability by Trip Length
    st.plotly_chart(fig, use_container_width=True)

with tab6:
    # Calculate red-eye counts and percentages by trip length
    trip_counts = [result['trip_counts'][i] for i in range(1, 6)]
    redeye_pcts = [result['redeye_pct'][i] for i in range(1, 6)]
    redeye_counts = [int(trip_counts[i-1] * redeye_pcts[i-1] / 100) f

    # Create bar chart with count and percentage
    data = pd.DataFrame({
        'Length': [f"{i}-day" for i in range(1, 6)],
        'Count': redeye_counts,
        'Percentage': redeye_pcts
    })

    fig = px.bar(
        data,
        x='Length',
        y='Count',
        labels={'Length': 'Trip Length', 'Count': 'Number of Trips wi
        title='Trips Containing Red-Eye Flight by Trip Length',
        text=[f"{count}<br>({pct:.1f}%)" for count, pct in zip(redeye
    )
    fig.update_traces(textposition='outside')
    st.plotly_chart(fig, use_container_width=True)

with tab7:
    st.markdown("### 📅 Daily Staffing Heat Map")
    st.caption("Shows the number of pilots working each day of the mo

    # Generate heat map data
    with st.spinner("Generating staffing heat map..."):
        heatmap_data = analysis_engine.generate_staffing_heatmap(
            fdata['content'],
            fdata['month'],
            fdata['year'],
            selected_base
        )

    # Create calendar-style heat map
    dates = heatmap_data['dates']
    pilot_counts = heatmap_data['pilot_counts']
    trip_details = heatmap_data['trip_details']

    # Format dates for display
    day_names = [d.strftime('%a') for d in dates]  # Mon, Tue, Wed, e
    day_numbers = [d.day for d in dates]

    # Create x-axis labels with day number and day of week
    x_labels = [f"{day}<br>{dow}" for day, dow in zip(day_numbers, da

    # Create text to display in each cell (the count)
    cell_text = [[str(count) if count > 0 else "" for count in pilot_

    # Create heat map using plotly
    import plotly.graph_objects as go

    # Determine color scale max (use 95th percentile to avoid outlier
    import numpy as np
    non_zero_counts = [c for c in pilot_counts if c > 0]
    if non_zero_counts:
        color_max = int(np.percentile(non_zero_counts, 95))
    else:
        color_max = 1

    fig = go.Figure(data=go.Heatmap(
        z=[pilot_counts],
        x=x_labels,
        y=['Pilots Working'],
        text=cell_text,
        texttemplate='<b>%{text}</b>',
        textfont={"size": 14, "color": "white"},
        hovertemplate='<b>Day %{customdata[0]}</b><br>Pilots: %{z}<br
        colorscale='Blues',
        zmin=0,
        zmax=color_max,
        colorbar=dict(title="Pilot<br>Count"),
        customdata=[[f"{day_numbers[i]} ({day_names[i]})", trip_detai
    ))

    fig.update_layout(
        title=f"Daily Pilot Operations - {heatmap_data['month']} {hea
        xaxis_title="",
        yaxis_title="",
        height=300,
        xaxis=dict(
            tickmode='array',
            tickvals=list(range(len(x_labels))),
            ticktext=x_labels,
            tickangle=0,
            side='bottom'
        ),
        yaxis=dict(
            showticklabels=False
        )
    )

    st.plotly_chart(fig, use_container_width=True)

    # Summary statistics
    col1, col2, col3, col4 = st.columns(4)
    with col1:
        st.metric("Peak Day", f"{max(pilot_counts)} pilots" if pilot_
    with col2:
        avg_pilots = sum(pilot_counts) / len(pilot_counts) if pilot_c
        st.metric("Avg Daily", f"{avg_pilots:.1f} pilots")
    with col3:
        days_with_ops = sum(1 for c in pilot_counts if c > 0)
        st.metric("Days with Ops", f"{days_with_ops}/{len(pilot_count
    with col4:
        total_pilot_days = sum(pilot_counts)
        st.metric("Total Pilot-Days", total_pilot_days)

    # Show day with peak operations
    if pilot_counts:
        max_idx = pilot_counts.index(max(pilot_counts))
        peak_date = dates[max_idx]
        st.info(f"📊 **Peak Operations:** {peak_date.strftime('%A, %E

    # Reserve Correlation Analysis Section
    st.markdown("---")
    st.markdown("### 📊 Reserve vs Operations Correlation Analysis")
    st.caption("Compare required reserve levels with daily pilot oper

    with st.expander("📥 Enter Reserve Data for Correlation Analysis"
        st.markdown("""
        **Instructions:** Enter the required reserve count for each d

        You can find this data in your reserve requirements document/
        """)

        # Create input fields for reserve data
        st.markdown("**Enter Reserve Requirements:**")

        # Option to bulk paste data
        bulk_input = st.text_area(
            "Paste reserve data (format: date,required or one per lin
            placeholder="Example:\n03FEB,39\n04FEB,40\n05FEB,42\n..."
            height=150,
            key='reserve_bulk_input'
        )

        if st.button("📊 Analyze Reserve Correlation", type="primary"
            if bulk_input.strip():
                try:
                    # Parse the bulk input
                    reserve_data = {}
                    month_abbr_map = {
                        'JAN': 1, 'FEB': 2, 'MAR': 3, 'APR': 4, 'MAY'
                        'JUL': 7, 'AUG': 8, 'SEP': 9, 'OCT': 10, 'NOV
                    }

                    lines = bulk_input.strip().split('\n')
                    for line in lines:
                        line = line.strip()
                        if not line or ',' not in line:
                            continue

                        parts = line.split(',')
                        if len(parts) >= 2:
                            date_str = parts[0].strip().upper()
                            required = int(parts[1].strip())

                            # Parse date (e.g., "03FEB" or "3FEB")
                            import re
                            match = re.match(r'(\d{1,2})([A-Z]{3})',
                            if match:
                                day = int(match.group(1))
                                month_abbr = match.group(2)
                month_num = month_abbr_map.get(month_

                if month_num == month_map.get(fdata['
                    date_key = datetime(fdata['year']
                    reserve_data[date_key] = required

    if reserve_data:
        # Match reserve data with pilot operations
        matched_dates = []
        reserve_required = []
        pilots_on_duty = []

        for date in dates:
            if date in reserve_data:
                matched_dates.append(date)
                reserve_required.append(reserve_data[
                pilots_on_duty.append(pilot_counts[da

        if len(matched_dates) >= 3:
            # Calculate correlation
            import numpy as np
            correlation = np.corrcoef(reserve_require

            # Display results
            st.success(f"✅ Analysis complete! Found

            # Correlation coefficient
            col1, col2, col3 = st.columns(3)
            with col1:
                st.metric("Correlation Coefficient",
            with col2:
                strength = "Strong" if abs(correlatio
                st.metric("Relationship Strength", st
            with col3:
                direction = "Positive" if correlation
                st.metric("Direction", direction)

            # Interpretation
            if correlation > 0.7:
                interpretation = "🟢 **Strong Positiv
            elif correlation > 0.4:
                interpretation = "🟡 **Moderate Posit
            elif correlation > 0:
                interpretation = "🟠 **Weak Positive
            else:
                interpretation = "🔴 **Negative/No Cc
```

```
                                st.info(interpretation)

                                # Scatter plot
                                import plotly.express as px
                                scatter_df = pd.DataFrame({
                                    'Pilots on Duty': pilots_on_duty,
                                    'Reserves Required': reserve_required
                                    'Date': [d.strftime('%b %d') for d in
                                    'Day of Week': [d.strftime('%a') for
                                })

                                fig = px.scatter(
                                    scatter_df,
                                    x='Pilots on Duty',
                                    y='Reserves Required',
                                    hover_data=['Date', 'Day of Week'],
                                    title='Reserve Requirements vs Pilot
                                    trendline='ols'
                                )
                                fig.update_traces(marker=dict(size=10))
                                st.plotly_chart(fig, use_container_width=

                                # Day-by-day comparison table
                                st.markdown("**Day-by-Day Comparison:**")
                                comparison_df = pd.DataFrame({
                                    'Date': [d.strftime('%b %d (%a)') for
                                    'Pilots on Duty': pilots_on_duty,
                                    'Reserves Required': reserve_required
                                    'Ratio (%)': [f"{(res/pilots*100):.1f
                                                 for res, pilots in zip(r
                                })
                                st.dataframe(comparison_df, use_container

                            else:
                                st.warning(f"⚠️ Only found {len(matched_c
                        else:
                            st.error("❌ No valid reserve data found. Che

                    except Exception as e:
                        st.error(f"❌ Error parsing reserve data: {str(e)
                        st.info("Expected format: 03FEB,39 (one per line)
                else:
                    st.warning("⚠️ Please enter reserve data to analyze."

else:
    # DETAILED TRIP TABLE VIEW
    # Get detailed trip data
```

```
if 'detailed_trips' not in st.session_state:
    st.session_state.detailed_trips = {}

if fname not in st.session_state.detailed_trips:
    with st.spinner("Loading detailed trip data..."):
        # Get bid month from uploaded file data
        bid_month = fdata['month']

        detailed_trips = analysis_engine.get_detailed_trips(
            fdata['content'],
            selected_base,
            bid_month,
            fdata['year']  # Pass the bid year
        )
        st.session_state.detailed_trips[fname] = detailed_trips

trips = st.session_state.detailed_trips[fname]

# Initialize filter state
if 'trip_filters' not in st.session_state:
    st.session_state.trip_filters = {
        'trip_length': 'All',
        'report_start': '00:00',
        'report_end': '23:59',
        'release_start': '00:00',
        'release_end': '23:59',
        'search_term': '',
        'sort_column': None,
        'sort_ascending': True
    }

# Filters section
st.markdown("### Filters")

# Create filter columns
filter_col1, filter_col2, filter_col3, filter_col4, filter_col5, filt

with filter_col1:
    trip_length_filter = st.selectbox(
        "Trip Length",
        ['All', '1-day', '2-day', '3-day', '4-day', '5-day'],
        key='filter_trip_length'
    )

with filter_col2:
    # Time options: 00:00 to 23:59 in 30-min increments, plus 23:59
    time_options = [f"{h:02d}:{m:02d}" for h in range(24) for m in [0
    time_options.append("23:59")

    col_a, col_b = st.columns(2)
    with col_a:
        report_start = st.selectbox("Report Start", time_options, ind
    with col_b:
        report_end = st.selectbox("Report End", time_options, index=l

with filter_col3:
    col_a, col_b = st.columns(2)
    with col_a:
        release_start = st.selectbox("Release Start", time_options, i
    with col_b:
        release_end = st.selectbox("Release End", time_options, index

with filter_col4:
    search_term = st.text_input("Search Trip #", key='filter_search',

with filter_col5:
    num_legs_filter = st.selectbox(
        "Number of Legs",
        ['All', '2', '3', '4', '5', '6', '7', '8', '9', '10+'],
        key='filter_num_legs'
    )

with filter_col6:
    credit_filter = st.selectbox(
        "Credit",
        ['All', 'Hard Block', '<15 minutes', '15-30 minutes', '30-60
        key='filter_credit'
    )

with filter_col7:
    st.write("")  # Spacer
    st.write("")  # Spacer
    if st.button("🔄 Clear", key='clear_filters'):
        # Clear detailed trips cache to force reload with unchecked b
        if fname in st.session_state.detailed_trips:
            del st.session_state.detailed_trips[fname]
        # Delete all filter widget keys - they'll reset to defaults o
        keys_to_delete = ['filter_trip_length', 'filter_report_start'
                         'filter_release_start', 'filter_release_end'
                         'filter_one_leg_home', 'filter_has_sit', 'fi
                         'filter_has_hol', 'filter_has_carve', 'filte
        for key in keys_to_delete:
            if key in st.session_state:
                del st.session_state[key]
        st.rerun()

# Checkbox filters row
st.markdown("#### Additional Filters")
checkbox_col1, checkbox_col2, checkbox_col3, checkbox_col4, checkbox_

with checkbox_col1:
    one_leg_home = st.checkbox("One Leg Home Last Day", key='filter_o

with checkbox_col2:
    has_sit = st.checkbox("Has SIT Pay", key='filter_has_sit')

with checkbox_col3:
    has_edp = st.checkbox("Has EDP", key='filter_has_edp')

with checkbox_col4:
    has_hol = st.checkbox("Has Holiday Pay", key='filter_has_hol')

with checkbox_col5:
    has_carve = st.checkbox("Has CARVE Pay", key='filter_has_carve')

with checkbox_col6:
    has_redeye = st.checkbox("Has Red-Eye", key='filter_has_redeye')

# Second checkbox row for additional filters
dh_col1, dh_col2, dh_col3, dh_col4, dh_col5, dh_col6 = st.columns(6)
with dh_col1:
    last_leg_dh_filter = st.checkbox(
        "Last Leg DH",
        key='filter_last_leg_dh',
        help="Show only trips where the final flight leg is a Deadhea
    )

# Apply filters
filtered_trips = trips.copy()

# Trip length filter
if trip_length_filter != 'All':
    length = int(trip_length_filter.split('-')[0])
    filtered_trips = [t for t in filtered_trips if t['length'] == len

# Report time filter
def time_to_minutes(time_str):
    h, m = map(int, time_str.split(':'))
    return h * 60 + m

report_start_min = time_to_minutes(report_start)
```

```
report_end_min = time_to_minutes(report_end)
filtered_trips = [t for t in filtered_trips
                if t['report_time_minutes'] is not None
                and report_start_min <= t['report_time_minutes'] <= r

# Release time filter
release_start_min = time_to_minutes(release_start)
release_end_min = time_to_minutes(release_end)
filtered_trips = [t for t in filtered_trips
                if t['release_time_minutes'] is not None
                and release_start_min <= t['release_time_minutes'] <=

# Search filter - match partial trip number (including suffix like -1
if search_term:
    filtered_trips = [t for t in filtered_trips
                    if t['trip_number'] and search_term in str(t['tri

# Number of legs filter
if num_legs_filter != 'All':
    if num_legs_filter == '10+':
        filtered_trips = [t for t in filtered_trips if t['total_legs'
    else:
        num_legs = int(num_legs_filter)
        filtered_trips = [t for t in filtered_trips if t['total_legs'

# Credit filter (CR time in minutes)
if credit_filter != 'All':
    if credit_filter == 'Hard Block':
        # CR = 0 minutes (hard block, no credit beyond block time)
        filtered_trips = [t for t in filtered_trips if t.get('credit_
    elif credit_filter == '<15 minutes':
        filtered_trips = [t for t in filtered_trips if t.get('credit_
    elif credit_filter == '15-30 minutes':
        filtered_trips = [t for t in filtered_trips if t.get('credit_
    elif credit_filter == '30-60 minutes':
        filtered_trips = [t for t in filtered_trips if t.get('credit_
    elif credit_filter == '>60 minutes':
        filtered_trips = [t for t in filtered_trips if t.get('credit_

# Checkbox filters
if one_leg_home:
    filtered_trips = [t for t in filtered_trips if t.get('last_day_le

if has_sit:
    filtered_trips = [t for t in filtered_trips if t.get('sit') is no

if has_edp:
    filtered_trips = [t for t in filtered_trips if t.get('edp') is no

if has_hol:
    filtered_trips = [t for t in filtered_trips if t.get('hol') is no

if has_carve:
    filtered_trips = [t for t in filtered_trips if t.get('carve') is

if has_redeye:
    filtered_trips = [t for t in filtered_trips if t.get('has_redeye'

if last_leg_dh_filter:
    filtered_trips = [t for t in filtered_trips if t.get('last_leg_dh

# Display trip count (sum of all occurrences)
total_occurrences = sum(trip.get('occurrences', 1) for trip in filter
st.markdown(f"**Showing {total_occurrences} trips** *({len(filtered_t

# AI Chat Section (above the table)
with st.expander("💬 Ask AI About Your Trips", expanded=False):
    st.markdown("Ask questions about the filtered trips in natural la
    st.markdown("*Examples: 'Which trips have the best credit ratio?'

    # Try to get API key from secrets, then from session state, then
    api_key = None
    if "ANTHROPIC_API_KEY" in st.secrets:
        api_key = st.secrets["ANTHROPIC_API_KEY"]
        st.success("✅ Using API key from Streamlit secrets")
    else:
        # API Key input
        api_key = st.text_input(
            "Anthropic API Key",
            type="password",
            help="Get your API key at https://console.anthropic.com,
            key="anthropic_api_key",
            value=st.session_state.get('saved_api_key', '')
        )

        if api_key and api_key != st.session_state.get('saved_api_key
            st.session_state.saved_api_key = api_key
            st.info("💡 API key saved for this session. To save perma

    # Question input
    user_question = st.text_area(
        "Your Question",
        placeholder="e.g., What are the top 5 trips by credit per day
        height=80,
    key="ai_question"
)

col1, col2 = st.columns(2)
with col1:
    ask_ai_btn = st.button("Ask AI", type="primary", use_containe
with col2:
    quick_calc_btn = st.button("⚡ Quick Calculate", use_containe

if quick_calc_btn:
    # Quick Python-only calculation for common queries
    question_lower = user_question.lower()
    asking_commutable = any(word in question_lower for word in ['
    asking_weekday_only = any(phrase in question_lower for phrase

    if not (asking_commutable and asking_weekday_only):
        st.warning("Quick Calculate works best for queries like:
    else:
        with st.spinner("Calculating..."):
            # Calculate directly in Python (fast!)
            def parse_time_to_minutes(time_str):
                if not time_str or time_str == 'N/A':
                    return None
                try:
                    parts = time_str.split(':')
                    return int(parts[0]) * 60 + int(parts[1])
                except:
                    return None

            front_threshold = parse_time_to_minutes(front_end_tim
            back_threshold = parse_time_to_minutes(back_end_time)

            matching_trips = []
            for trip in filtered_trips:
                report_min = parse_time_to_minutes(trip.get('repo
                release_min = parse_time_to_minutes(trip.get('rel

                front_ok = report_min is not None and front_thres
                back_ok = release_min is not None and back_thresh
                both_ok = front_ok and back_ok

                days = trip.get('days_of_week', [])
                weekday_only = 'SA' not in days and 'SU' not in d

                if both_ok and weekday_only:
                    matching_trips.append(trip)
```

```
                            # Calculate totals
                            total_occurrences = sum(t.get('occurrences', 1) for t
                            by_length = {}
                            for trip in matching_trips:
                                length = trip['length']
                                by_length[length] = by_length.get(length, 0) + tr

                            # Display results
                            st.success("✨ Quick Answer:")
                            st.markdown(f"""
**Found {len(matching_trips)} unique trip patterns ({total_occurrences} total occ
- ✅ Both-ends commutable (report ≥ {front_end_time}, release ≤ {back_end_time})
- ✅ Operate Monday-Friday only (no weekends)

**Breakdown by trip length:**
""")
                            for length in sorted(by_length.keys()):
                                count = by_length[length]
                                st.markdown(f"- **{length}-day trips:** {count} o

                            if len(matching_trips) > 0:
                                st.markdown(f"\n💡 *Analyzed all {len(filtered_tr

                if ask_ai_btn:
                    if not api_key:
                        st.error("Please enter your Anthropic API key first")
                    elif not user_question:
                        st.error("Please enter a question")
                    elif len(filtered_trips) == 0:
                        st.warning("No trips to analyze. Adjust your filters.")
                    elif len(filtered_trips) > 500:
                        # Check if user has asked the same question before
                        question_key = f"ai_question_{user_question.strip().lower
                        warned_before = st.session_state.get(question_key, False)

                        if not warned_before:
                            # First time asking with >500 trips - show warning
                            st.warning(f"""
⚠️ **Large Dataset Warning**

You have **{len(filtered_trips)} trips** to analyze. The AI can only process the

**Recommended options:**
1. **Filter your data** using the sidebar filters (Base, Length, Credit, etc.) to
2. **Use ⚡ Quick Calculate** for counting queries (instant and analyzes ALL trip
3. **Click "Ask AI" again** to proceed anyway (AI will provide statistical summar
""")
```

```
        # Mark that we've warned the user for this question
        st.session_state[question_key] = True
        st.stop()
    else:
        # User asked again - proceed with statistical summary
        st.info(f"📊 Analyzing {len(filtered_trips)} trips -

# If we get here, proceed with AI analysis
if not (len(filtered_trips) > 500 and not st.session_state.ge
    with st.spinner("Analyzing trips..."):
        try:
            # Prepare trip data for AI
            import anthropic

            # Determine mode: detailed or statistical
            statistical_mode = len(filtered_trips) > 500

            if not statistical_mode:
                # Standard mode: send trip details
                trip_summary = []

                # Get current commutability thresholds from s
                def parse_time_to_minutes(time_str):
                    """Convert HH:MM to minutes, returns None
                    if not time_str or time_str == 'N/A':
                        return None
                    try:
                        parts = time_str.split(':')
                        return int(parts[0]) * 60 + int(parts
                    except:
                        return None

            front_threshold = parse_time_to_minutes(front_end
            back_threshold = parse_time_to_minutes(back_end_t

            for trip in filtered_trips[:500]:  # Increased to
                # Calculate commutability flags by parsing ti
                report_minutes = parse_time_to_minutes(trip.g
                release_minutes = parse_time_to_minutes(trip.

                front_commutable = report_minutes is not None
                back_commutable = release_minutes is not None
                both_ends_commutable = front_commutable and b

                trip_summary.append({
                    'trip_number': trip.get('trip_number', 'N
                    'base': trip['base'],
        'length': f"{trip['length']}-day",
        'days_of_week': trip.get('days_of_week',
        'occurrences': trip.get('occurrences', 1)
        'report': trip.get('report_time'),
        'release': trip.get('release_time'),
        'front_end_commutable': front_commutable,
        'back_end_commutable': back_commutable,
        'both_ends_commutable': both_ends_commuta
        'legs': trip['total_legs'],
        'longest_leg': trip.get('longest_leg'),
        'shortest_leg': trip.get('shortest_leg'),
        'credit': trip.get('total_credit'),
        'pay': trip.get('total_pay'),
        'sit': trip.get('sit'),
        'edp': trip.get('edp'),
        'hol': trip.get('hol'),
        'carve': trip.get('carve'),
        'credit_per_day': trip.get('total_credit'
        'last_day_legs': trip.get('last_day_legs'
    })

else:
    # Statistical mode: Calculate stats in Python
    def parse_time_to_minutes(time_str):
        if not time_str or time_str == 'N/A':
            return None
        try:
            parts = time_str.split(':')
            return int(parts[0]) * 60 + int(parts
        except:
            return None

    front_threshold = parse_time_to_minutes(front
    back_threshold = parse_time_to_minutes(back_e

    # Calculate comprehensive statistics
    stats = {
        'total_trips': len(filtered_trips),
        'total_occurrences': sum(t.get('occurrenc
        'by_length': {},
        'by_base': {},
        'commutability': {
            'front': 0,
            'back': 0,
            'both': 0
        },
        'weekday_only': 0,
        'weekend_included': 0,
        'avg_credit': sum(t.get('total_credit', 0
        'avg_length': sum(t['length'] * t.get('oc
    }

    for trip in filtered_trips:
        length = trip['length']
        base = trip['base']
        occurrences = trip.get('occurrences', 1)

        # By length
        if length not in stats['by_length']:
            stats['by_length'][length] = 0
        stats['by_length'][length] += occurrences

        # By base
        if base not in stats['by_base']:
            stats['by_base'][base] = 0
        stats['by_base'][base] += occurrences

        # Commutability
        report_min = parse_time_to_minutes(trip.g
        release_min = parse_time_to_minutes(trip.

        front_ok = report_min is not None and fro
        back_ok = release_min is not None and bac

        if front_ok:
            stats['commutability']['front'] += oc
        if back_ok:
            stats['commutability']['back'] += occ
        if front_ok and back_ok:
            stats['commutability']['both'] += occ

        # Weekday analysis
        days = trip.get('days_of_week', [])
        if 'SA' not in days and 'SU' not in days
            stats['weekday_only'] += occurrences
        elif 'SA' in days or 'SU' in days:
            stats['weekend_included'] += occurren

    trip_summary = stats

# Call Claude API
client = anthropic.Anthropic(api_key=api_key)

if statistical_mode:
```

```
                                    # Statistical mode prompt
                                    prompt_content = f"""You are analyzing pilot

{trip_summary}

Statistics explanation:
- total_trips: Total unique trip patterns
- total_occurrences: Total trips when counting each occurrence
- by_length: Breakdown by trip length (1-5 days)
- by_base: Breakdown by airline base
- commutability: Counts of trips that are front/back/both ends commutable (thresh
- weekday_only: Trips operating Monday-Friday only (no SA or SU)
- weekend_included: Trips that include Saturday or Sunday
- avg_credit: Average credit hours per trip
- avg_length: Average trip length in days

The user's question is: {user_question}

Please analyze these statistics and provide a clear, concise answer. Focus on the
                                else:
                                    # Detailed mode prompt
                                    prompt_content = f"""You are analyzing pilot

{trip_summary}

Each trip includes:
- days_of_week: Which days of the week this trip operates (e.g., ['MO', 'TU', 'WE
- occurrences: How many times this trip pattern operates during the bid period
- report/release: Time strings (HH:MM format)
- front_end_commutable: True if report_minutes >= front_threshold (630 = 10:30)
- back_end_commutable: True if release_minutes <= back_threshold (1080 = 18:00)
- both_ends_commutable: True if BOTH front and back are commutable

CRITICAL FILTERING RULES:
- "Monday-Friday only" means days_of_week must contain ONLY weekday codes (MO, TU
- If days_of_week contains 'SA' or 'SU', the trip is NOT Monday-Friday only
- Example: ['SA'] = Saturday trip = NOT Monday-Friday
- Example: ['MO'] = Monday trip = Monday-Friday only ✓
- Example: ['MO', 'SA'] = Monday and Saturday = NOT Monday-Friday only

Common day patterns:
- Monday-Friday only: days_of_week contains only ['MO', 'TU', 'WE', 'TH', 'FR'] o
- Weekends only: days_of_week contains only ['SA', 'SU']
- Every day: days_of_week is empty or contains all 7 days

The user's question is: {user_question}
```

```
When counting trips, remember to sum the 'occurrences' field to get total trips.

                                message = client.messages.create(
                                    model="claude-sonnet-4-20250514",
                                    max_tokens=4000,
                                    messages=[{
                                        "role": "user",
                                        "content": prompt_content
                                    }]
                                )

                                # Display response
                                st.success("✨ AI Analysis:")
                                st.markdown(message.content[0].text)

                            except ImportError:
                                st.error("❌ Anthropic library not installed. Run
                            except Exception as e:
                                st.error(f"❌ Error: {str(e)}")

            # Create dataframe for display
            if filtered_trips:
                # Initialize selected trip in session state
                if 'selected_trip_index' not in st.session_state:
                    st.session_state.selected_trip_index = None

                df_data = []
                for i, trip in enumerate(filtered_trips):
                    df_data.append({
                        'Select': False,
                        'Trip #': trip['trip_number'] or 'N/A',
                        'Base': trip['base'],
                        'Length': f"{trip['length']}-day",
                        'Report': trip['report_time'] or 'N/A',
                        'Release': trip['release_time'] or 'N/A',
                        'Legs': trip['total_legs'],
                        'Longest': trip['longest_leg'],
                        'Shortest': trip['shortest_leg'],
                        'Credit': trip['total_credit'] if trip['total_credit'] is
                        'Pay': trip['total_pay'] if trip['total_pay'] is not None
                        'SIT': trip.get('sit'),
                        'EDP': trip.get('edp'),
                        'HOL': trip.get('hol'),
                        'CARVE': trip.get('carve'),
                        'Occurs': trip['occurrences']
                    })
```

```
df = pd.DataFrame(df_data)

# Use data_editor for selection capability with column sorting en
edited_df = st.data_editor(
    df,
    column_config={
        'Select': st.column_config.CheckboxColumn(
            'Select',
            help='Click to view trip details',
            default=False,
            width='small'
        ),
        'Trip #': st.column_config.TextColumn('Trip #', width='me
        'Base': st.column_config.TextColumn('Base', width='small'
        'Length': st.column_config.TextColumn('Length', width='sm
        'Report': st.column_config.TextColumn('Report', width='sm
        'Release': st.column_config.TextColumn('Release', width='
        'Legs': st.column_config.NumberColumn('Legs', width='smal
        'Longest': st.column_config.TextColumn('Longest', width='
        'Shortest': st.column_config.TextColumn('Shortest', width
        'Credit': st.column_config.NumberColumn('Credit', width='
        'Pay': st.column_config.NumberColumn('Pay', width='small'
        'SIT': st.column_config.NumberColumn('SIT', width='small'
        'EDP': st.column_config.NumberColumn('EDP', width='small'
        'HOL': st.column_config.NumberColumn('HOL', width='small'
        'CARVE': st.column_config.NumberColumn('CARVE', width='sm
        'Occurs': st.column_config.NumberColumn('Occurs', width='
    },
    disabled=['Trip #', 'Base', 'Length', 'Report', 'Release', 'L
    hide_index=True,
    use_container_width=True,
    height=600,
    key='trip_table'
)

# Check which trips are selected
selected_indices = edited_df[edited_df['Select']].index.tolist()

# Display details for selected trips
if selected_indices:
    st.markdown("---")

    selected_trip_objects = [filtered_trips[idx] for idx in selec

    # Build settings string for PDF header
    pdf_settings = (
        f"Base: {selected_base}  |  "
    f"Front-End: ≥{front_end_time}  |  "
    f"Back-End: ≤{back_end_time}  |  "
    f"{fdata.get('display_name', '')}"
)

# Header row with export buttons
hdr_col1, hdr_col2, hdr_col3 = st.columns([3, 1, 1])
with hdr_col1:
    total_sel_occ = sum(t.get('occurrences', 1) for t in sele
    st.markdown(
        f"### ✈️ Selected Trip Details &nbsp;"
        f"<span style='font-size:0.85rem;color:grey'>"
        f"({len(selected_indices)} pattern{'s' if len(selecte
        f"{total_sel_occ} occurrence{'s' if total_sel_occ !=
        unsafe_allow_html=True
    )
with hdr_col2:
    try:
        pdf_bytes = analysis_engine.generate_selected_trips_p
            selected_trip_objects,
            display_name=fdata.get('display_name', ''),
            settings_text=pdf_settings
        )
        st.download_button(
            label="📄 Print / Save PDF",
            data=pdf_bytes,
            file_name=f"selected_trips_{datetime.now().strfti
            mime="application/pdf",
            use_container_width=True,
            key='export_selected_pdf'
        )
    except Exception as e:
        st.error(f"PDF error: {e}")
with hdr_col3:
    txt_content = analysis_engine.generate_selected_trips_txt
    st.download_button(
        label="📋 Export Raw TXT",
        data=txt_content.encode('utf-8'),
        file_name=f"selected_trips_{datetime.now().strftime('
        mime="text/plain",
        use_container_width=True,
        key='export_selected_txt'
    )

# Trip raw-text expanders below the export buttons
for idx in selected_indices:
    trip = filtered_trips[idx]
```

```
                    trip_num = trip['trip_number']
                    with st.expander(f"Trip #{trip_num} - Full Details", expa
                        st.code(trip['raw_text'], language=None)
        else:
            st.info("No trips match the current filters.")

else:
    # Multiple files - show detailed comparison tables
    st.subheader("📈 Detailed Comparison Analysis")

    # Add grouping mode selector
    grouping_mode = st.radio(
        "Display Mode",
        ["Sequential (by date)", "Year-over-Year (by month)"],
        horizontal=True,
        help="Sequential shows all files in chronological order. Year-over-Ye
    )

    # AI Chat Section for Comparison View
    with st.expander("💬 Ask AI About This Comparison", expanded=False):
        st.markdown("Ask questions comparing the different bid packs!")
        st.markdown("*Examples: 'Which month has better 3-day trips?', 'How d

        # Try to get API key from secrets, then from session state, then from
        api_key_comparison = None
        if "ANTHROPIC_API_KEY" in st.secrets:
            api_key_comparison = st.secrets["ANTHROPIC_API_KEY"]
            st.success("✅ Using API key from Streamlit secrets")
        else:
            # API Key input
            api_key_comparison = st.text_input(
                "Anthropic API Key",
                type="password",
                help="Get your API key at https://console.anthropic.com, or s
                key="anthropic_api_key_comparison",
                value=st.session_state.get('saved_api_key', '')
            )

            if api_key_comparison and api_key_comparison != st.session_state.
                st.session_state.saved_api_key = api_key_comparison
                st.info("💡 API key saved for this session. To save permanent

        # Question input
        user_question_comparison = st.text_area(
            "Your Question",
            placeholder="e.g., Which month should I bid for to maximize credi
            height=80,
                key="ai_question_comparison"
            )

            if st.button("Ask AI", type="primary", key="ask_ai_comparison"):
                if not api_key_comparison:
                    st.error("Please enter your Anthropic API key first")
                elif not user_question_comparison:
                    st.error("Please enter a question")
                else:
                    with st.spinner("Analyzing comparison data..."):
                        try:
                            import anthropic

                            # Prepare comparison data for AI
                            comparison_data = {}
                            for fname in st.session_state.analysis_results.keys()
                                fdata = st.session_state.uploaded_files[fname]
                                result = st.session_state.analysis_results[fname]

                                comparison_data[fdata['display_name']] = {
                                    'total_trips': result['total_trips'],
                                    'avg_trip_length': result['avg_trip_length'],
                                    'avg_credit_per_trip': result['avg_credit_per
                                    'avg_credit_per_day': result['avg_credit_per_
                                    'trip_counts_by_length': result['trip_counts'
                                    'avg_credit_by_length': result['avg_credit_by
                                    'avg_credit_per_day_by_length': result['avg_c
                                    'single_leg_last_day_pct': result['single_leg
                                    'redeye_pct': result['redeye_pct'],
                                    'front_end_commutable_pct': result['front_com
                                    'back_end_commutable_pct': result['back_commu
                                }

                            # Call Claude API
                            client = anthropic.Anthropic(api_key=api_key_comparis
                            message = client.messages.create(
                                model="claude-sonnet-4-20250514",
                                max_tokens=2000,
                                messages=[{
                                    "role": "user",
                                    "content": f"""You are analyzing pilot trip s

{comparison_data}

The user's question is: {user_question_comparison}

Please provide a helpful, detailed comparison highlighting key differences and pr
```

```
                        }]
                    )

                    # Display response
                    st.success("✨ AI Comparison Analysis:")
                    st.markdown(message.content[0].text)

                except ImportError:
                    st.error("❌ Anthropic library not installed. Run: `p
                except Exception as e:
                    st.error(f"❌ Error: {str(e)}")

# Sort files by date (month/year)
month_order = {
    'January': 1, 'February': 2, 'March': 3, 'April': 4,
    'May': 5, 'June': 6, 'July': 7, 'August': 8,
    'September': 9, 'October': 10, 'November': 11, 'December': 12
}

sorted_files = sorted(
    st.session_state.analysis_results.keys(),
    key=lambda f: (
        st.session_state.uploaded_files[f]['year'],
        month_order[st.session_state.uploaded_files[f]['month']]
    )
)

num_files = len(sorted_files)
show_differences = (num_files == 2)

# YEAR-OVER-YEAR MODE
if grouping_mode == "Year-over-Year (by month)":
    # Group files by month
    files_by_month = {}
    for fname in sorted_files:
        month = st.session_state.uploaded_files[fname]['month']
        if month not in files_by_month:
            files_by_month[month] = []
        files_by_month[month].append(fname)

    # Only show months with 2+ years
    multi_year_months = {m: fs for m, fs in files_by_month.items() if len

    if not multi_year_months:
        st.warning("⚠️ No months with multiple years found.\n\nTo use Yea

    # Display each month
```

```
for month in ['January', 'February', 'March', 'April', 'May', 'June',
              'July', 'August', 'September', 'October', 'November', '
    if month not in multi_year_months:
        continue

    st.markdown(f"## 📅 {month}")
    month_files = multi_year_months[month]

    # 1. Trip Length
    st.markdown("### 1️⃣ Trip Length Distribution")
    data = []
    for fname in month_files:
        r = st.session_state.analysis_results[fname]
        row = {'Year': st.session_state.uploaded_files[fname]['displa
        for l in range(1, 6):
            cnt = r['trip_counts'][l]
            pct = (cnt / r['total_trips'] * 100) if r['total_trips']
            row[f'{l}-day'] = f"{cnt} ({pct:.2f}%)"
        row['Total'] = f"{r['total_trips']} (100%)"
        data.append(row)
    st.dataframe(pd.DataFrame(data), use_container_width=True, hide_i

    # 2. Single Leg
    st.markdown("### 2️⃣ Trips with Single Leg on Last Day")
    data = []
    for fname in month_files:
        r = st.session_state.analysis_results[fname]
        row = {'Year': st.session_state.uploaded_files[fname]['displa
        for l in range(1, 6):
            row[f'{l}-day'] = f"{r['single_leg_pct'][l]:.2f}%"
        data.append(row)
    st.dataframe(pd.DataFrame(data), use_container_width=True, hide_i

    # 3. Credit/Trip
    st.markdown("### 3️⃣ Average Credit per Trip")
    data = []
    for fname in month_files:
        r = st.session_state.analysis_results[fname]
        row = {'Year': st.session_state.uploaded_files[fname]['displa
        for l in range(1, 6):
            row[f'{l}-day'] = f"{r['avg_credit_by_length'][l]:.2f}"
        row['Overall'] = f"{r['avg_credit_per_trip']:.2f}"
        data.append(row)
    st.dataframe(pd.DataFrame(data), use_container_width=True, hide_i

    # 4. Credit/Day
    st.markdown("### 4️⃣ Average Credit per Day")
```

```
        data = []
        for fname in month_files:
            r = st.session_state.analysis_results[fname]
            row = {'Year': st.session_state.uploaded_files[fname]['displa
            for l in range(1, 6):
                row[f'{l}-day'] = f"{r['avg_credit_per_day_by_length'][l]
            row['Overall'] = f"{r['avg_credit_per_day']:.2f}"
            data.append(row)
        st.dataframe(pd.DataFrame(data), use_container_width=True, hide_i

        # 5. Commutability
        st.markdown("### 5️⃣ Commutability (Both Ends)")
        data = []
        for fname in month_files:
            r = st.session_state.analysis_results[fname]
            row = {'Year': st.session_state.uploaded_files[fname]['displa
            for l in range(1, 6):
                row[f'{l}-day'] = f"{r['both_commute_pct'][l]:.2f}%"
            row['Overall'] = f"{r['both_commute_rate']:.2f}%"
            data.append(row)
        st.dataframe(pd.DataFrame(data), use_container_width=True, hide_i

        # 6. Red-Eye
        st.markdown("### 6️⃣ Trips Containing Red-Eye Flight")
        data = []
        for fname in month_files:
            r = st.session_state.analysis_results[fname]
            row = {'Year': st.session_state.uploaded_files[fname]['displa
            for l in range(1, 6):
                row[f'{l}-day'] = f"{r['redeye_pct'][l]:.2f}%"
            row['Overall'] = f"{r['redeye_rate']:.2f}%"
            # Calculate total red-eyes
            total_redeye = sum(r['trip_counts'][i] * r['redeye_pct'][i] /
            row['Total Red-Eyes'] = f"{int(total_redeye)}"
            data.append(row)
        st.dataframe(pd.DataFrame(data), use_container_width=True, hide_i

        st.markdown("---")

# SEQUENTIAL MODE (original logic)
else:

    # 1. TRIP LENGTH DISTRIBUTION
    st.markdown("### 1️⃣ Trip Length Distribution")
    trip_dist_data = []
    for fname in sorted_files:
        result = st.session_state.analysis_results[fname]
    display_name = st.session_state.uploaded_files[fname]['display_na
    row = {'File': display_name}
    for length in range(1, 6):
        count = result['trip_counts'][length]
        pct = (count / result['total_trips'] * 100) if result['total_
        row[f'{length}-day'] = f"{count} ({pct:.2f}%)"
    row['Total'] = f"{result['total_trips']} (100.00%)"
    trip_dist_data.append(row)
st.dataframe(pd.DataFrame(trip_dist_data), use_container_width=True,

# Show differences only if exactly 2 files
if show_differences:
    st.markdown("**Change (Newer - Older):**")
    diff_data = []
    r1 = st.session_state.analysis_results[sorted_files[0]]
    r2 = st.session_state.analysis_results[sorted_files[1]]
    row = {'Metric': 'Percentage Point Difference'}
    for length in range(1, 6):
        pct1 = (r1['trip_counts'][length] / r1['total_trips'] * 100)
        pct2 = (r2['trip_counts'][length] / r2['total_trips'] * 100)
        diff = pct2 - pct1
        row[f'{length}-day'] = f"{diff:+.2f} points"
    diff_data.append(row)
    st.dataframe(pd.DataFrame(diff_data), use_container_width=True, h

st.markdown("---")

# 2. SINGLE LEG ON LAST DAY
st.markdown("### 2️⃣ Trips with Single Leg on Last Day")
single_leg_data = []
for fname in sorted_files:
    result = st.session_state.analysis_results[fname]
    display_name = st.session_state.uploaded_files[fname]['display_na
    row = {'File': display_name}
    for length in range(1, 6):
        total = result['trip_counts'][length]
        single_count = int(total * result['single_leg_pct'][length] /
        pct = result['single_leg_pct'][length]
        row[f'{length}-day'] = f"{single_count} ({pct:.2f}%)"
    # Overall
    total_single = sum(result['trip_counts'][i] * result['single_leg_
    overall_pct = (total_single / result['total_trips'] * 100) if res
    overall_count = int(total_single)
    row['Overall'] = f"{overall_count} ({overall_pct:.2f}%)"
    single_leg_data.append(row)
st.dataframe(pd.DataFrame(single_leg_data), use_container_width=True,
if show_differences:
    st.markdown("**Change (Newer - Older):**")
    diff_data = []
    r1 = st.session_state.analysis_results[sorted_files[0]]
    r2 = st.session_state.analysis_results[sorted_files[1]]
    row = {'Metric': 'Percentage Point Difference'}
    for length in range(1, 6):
        diff = r2['single_leg_pct'][length] - r1['single_leg_pct'][le
        row[f'{length}-day'] = f"{diff:+.2f} points"
    # Overall difference
    total_single1 = sum(r1['trip_counts'][i] * r1['single_leg_pct'][i
    overall_pct1 = (total_single1 / r1['total_trips'] * 100) if r1['t
    total_single2 = sum(r2['trip_counts'][i] * r2['single_leg_pct'][i
    overall_pct2 = (total_single2 / r2['total_trips'] * 100) if r2['t
    row['Overall'] = f"{overall_pct2 - overall_pct1:+.2f} points"
    diff_data.append(row)
    st.dataframe(pd.DataFrame(diff_data), use_container_width=True, h

st.markdown("---")

# 3. AVERAGE CREDIT PER TRIP
st.markdown("### 3️⃣ Average Credit per Trip (hours)")
credit_trip_data = []
for fname in sorted_files:
    result = st.session_state.analysis_results[fname]
    display_name = st.session_state.uploaded_files[fname]['display_na
    row = {'File': display_name}
    for length in range(1, 6):
        row[f'{length}-day'] = f"{result['avg_credit_by_length'][leng
    row['Overall'] = f"{result['avg_credit_per_trip']:.2f} hrs"
    credit_trip_data.append(row)
st.dataframe(pd.DataFrame(credit_trip_data), use_container_width=True

if show_differences:
    st.markdown("**Change (Newer - Older):**")
    diff_data = []
    r1 = st.session_state.analysis_results[sorted_files[0]]
    r2 = st.session_state.analysis_results[sorted_files[1]]
    row = {'Metric': 'Hour Difference'}
    for length in range(1, 6):
        diff = r2['avg_credit_by_length'][length] - r1['avg_credit_by
        row[f'{length}-day'] = f"{diff:+.2f} hrs"
    overall_diff = r2['avg_credit_per_trip'] - r1['avg_credit_per_tri
    row['Overall'] = f"{overall_diff:+.2f} hrs"
    diff_data.append(row)
    st.dataframe(pd.DataFrame(diff_data), use_container_width=True, h
st.markdown("---")

# 4. AVERAGE CREDIT PER DAY
st.markdown("### 4️⃣ Average Credit per Day (hours/day)")
credit_day_data = []
for fname in sorted_files:
    result = st.session_state.analysis_results[fname]
    display_name = st.session_state.uploaded_files[fname]['display_na
    row = {'File': display_name}
    for length in range(1, 6):
        row[f'{length}-day'] = f"{result['avg_credit_per_day_by_lengt
    row['Overall'] = f"{result['avg_credit_per_day']:.2f} hrs/day"
    credit_day_data.append(row)
st.dataframe(pd.DataFrame(credit_day_data), use_container_width=True,

if show_differences:
    st.markdown("**Change (Newer - Older):**")
    diff_data = []
    r1 = st.session_state.analysis_results[sorted_files[0]]
    r2 = st.session_state.analysis_results[sorted_files[1]]
    row = {'Metric': 'Hours/Day Difference'}
    for length in range(1, 6):
        diff = r2['avg_credit_per_day_by_length'][length] - r1['avg_c
        row[f'{length}-day'] = f"{diff:+.2f} hrs/day"
    overall_diff = r2['avg_credit_per_day'] - r1['avg_credit_per_day'
    row['Overall'] = f"{overall_diff:+.2f} hrs/day"
    diff_data.append(row)
    st.dataframe(pd.DataFrame(diff_data), use_container_width=True, h

st.markdown("---")

# 5. COMMUTABILITY
st.markdown("### 5️⃣ Commutability")

st.markdown("**Front-End Commutable (Report ≥ threshold):**")
front_data = []
for fname in sorted_files:
    result = st.session_state.analysis_results[fname]
    display_name = st.session_state.uploaded_files[fname]['display_na
    row = {'File': display_name}
    for length in range(1, 6):
        total = result['trip_counts'][length]
        commute_count = int(total * result['front_commute_pct'][lengt
        pct = result['front_commute_pct'][length]
        row[f'{length}-day'] = f"{commute_count} ({pct:.2f}%)"
    # Overall
    total_commute = sum(result['trip_counts'][i] * result['front_comm
    overall_count = int(total_commute)
    row['Overall'] = f"{overall_count} ({result['front_commute_rate']
    front_data.append(row)
st.dataframe(pd.DataFrame(front_data), use_container_width=True, hide

if show_differences:
    diff_data = []
    r1 = st.session_state.analysis_results[sorted_files[0]]
    r2 = st.session_state.analysis_results[sorted_files[1]]
    row = {'Metric': 'Difference'}
    for length in range(1, 6):
        diff = r2['front_commute_pct'][length] - r1['front_commute_pc
        row[f'{length}-day'] = f"{diff:+.2f} points"
    overall_diff = r2['front_commute_rate'] - r1['front_commute_rate'
    row['Overall'] = f"{overall_diff:+.2f} points"
    diff_data.append(row)
    st.dataframe(pd.DataFrame(diff_data), use_container_width=True, h

st.markdown("**Back-End Commutable (Release ≤ threshold):**")
back_data = []
for fname in sorted_files:
    result = st.session_state.analysis_results[fname]
    display_name = st.session_state.uploaded_files[fname]['display_na
    row = {'File': display_name}
    for length in range(1, 6):
        total = result['trip_counts'][length]
        commute_count = int(total * result['back_commute_pct'][length
        pct = result['back_commute_pct'][length]
        row[f'{length}-day'] = f"{commute_count} ({pct:.2f}%)"
    # Overall
    total_commute = sum(result['trip_counts'][i] * result['back_commu
    overall_count = int(total_commute)
    row['Overall'] = f"{overall_count} ({result['back_commute_rate']:
    back_data.append(row)
st.dataframe(pd.DataFrame(back_data), use_container_width=True, hide_

if show_differences:
    diff_data = []
    r1 = st.session_state.analysis_results[sorted_files[0]]
    r2 = st.session_state.analysis_results[sorted_files[1]]
    row = {'Metric': 'Difference'}
    for length in range(1, 6):
        diff = r2['back_commute_pct'][length] - r1['back_commute_pct'
        row[f'{length}-day'] = f"{diff:+.2f} points"
    overall_diff = r2['back_commute_rate'] - r1['back_commute_rate']
    row['Overall'] = f"{overall_diff:+.2f} points"
    diff_data.append(row)
    st.dataframe(pd.DataFrame(diff_data), use_container_width=True, h

st.markdown("**Both Ends Commutable:**")
both_data = []
for fname in sorted_files:
    result = st.session_state.analysis_results[fname]
    display_name = st.session_state.uploaded_files[fname]['display_na
    row = {'File': display_name}
    for length in range(1, 6):
        total = result['trip_counts'][length]
        commute_count = int(total * result['both_commute_pct'][length
        pct = result['both_commute_pct'][length]
        row[f'{length}-day'] = f"{commute_count} ({pct:.2f}%)"
    # Overall
    total_commute = sum(result['trip_counts'][i] * result['both_commu
    overall_count = int(total_commute)
    row['Overall'] = f"{overall_count} ({result['both_commute_rate']:
    both_data.append(row)
st.dataframe(pd.DataFrame(both_data), use_container_width=True, hide_

if show_differences:
    diff_data = []
    r1 = st.session_state.analysis_results[sorted_files[0]]
    r2 = st.session_state.analysis_results[sorted_files[1]]
    row = {'Metric': 'Difference'}
    for length in range(1, 6):
        diff = r2['both_commute_pct'][length] - r1['both_commute_pct'
        row[f'{length}-day'] = f"{diff:+.2f} points"
    overall_diff = r2['both_commute_rate'] - r1['both_commute_rate']
    row['Overall'] = f"{overall_diff:+.2f} points"
    diff_data.append(row)
    st.dataframe(pd.DataFrame(diff_data), use_container_width=True, h

st.markdown("---")

# 6. RED-EYE TRIPS
st.markdown("### 6️⃣ Trips Containing Red-Eye Flight")
redeye_data = []
for fname in sorted_files:
    result = st.session_state.analysis_results[fname]
    display_name = st.session_state.uploaded_files[fname]['display_na
    row = {'File': display_name}
    for length in range(1, 6):
        total = result['trip_counts'][length]
        redeye_count = int(total * result['redeye_pct'][length] / 100
        pct = result['redeye_pct'][length]
        row[f'{length}-day'] = f"{redeye_count} ({pct:.2f}%)"
```

```
            # Overall
            total_redeye = sum(result['trip_counts'][i] * result['redeye_pct'
            overall_count = int(total_redeye)
            row['Overall'] = f"{overall_count} ({result['redeye_rate']:.2f}%)
            row['Total Red-Eyes'] = f"{overall_count}"
            redeye_data.append(row)
        st.dataframe(pd.DataFrame(redeye_data), use_container_width=True, hid

        if show_differences:
            st.markdown("**Change (Newer - Older):**")
            diff_data = []
            r1 = st.session_state.analysis_results[sorted_files[0]]
            r2 = st.session_state.analysis_results[sorted_files[1]]
            row = {'Metric': 'Percentage Point Difference'}
            for length in range(1, 6):
                diff = r2['redeye_pct'][length] - r1['redeye_pct'][length]
                row[f'{length}-day'] = f"{diff:+.2f} points"
            # Overall difference
            overall_diff = r2['redeye_rate'] - r1['redeye_rate']
            row['Overall'] = f"{overall_diff:+.2f} points"
            # Absolute count difference
            total_redeye_1 = sum(r1['trip_counts'][i] * r1['redeye_pct'][i] /
            total_redeye_2 = sum(r2['trip_counts'][i] * r2['redeye_pct'][i] /
            count_diff = int(total_redeye_2) - int(total_redeye_1)
            row['Total Red-Eyes'] = f"{count_diff:+d}"
            diff_data.append(row)
            st.dataframe(pd.DataFrame(diff_data), use_container_width=True, h

    # Export buttons row
btn_col1, btn_col2 = st.columns(2)

with btn_col1:
    if st.button("📊 Export Summary/Comparison PDF Report", key='pdf_export')
        with st.spinner("Generating PDF..."):
            pdf_bytes = analysis_engine.generate_pdf_report(
                st.session_state.analysis_results,
                st.session_state.uploaded_files,
                selected_base,
                front_end_time,
                back_end_time
            )
            st.download_button(
                label="⬇️ Download Summary/Comparison PDF",
                data=pdf_bytes,
                file_name=f"trip_analysis_{datetime.now().strftime('%Y%m%d_%H
                mime="application/pdf",
                key='pdf_download'
                )

    with btn_col2:
        # Only available for single-file analysis
        single_file_available = len(st.session_state.uploaded_files) == 1
        if not single_file_available:
            st.info("📋 Comprehensive Base Report available for single-file analy
        else:
            if st.button("📋 Export Comprehensive Base Report", key='comprehensiv
                with st.spinner("Generating comprehensive report (this may take a
                    fname = list(st.session_state.uploaded_files.keys())[0]
                    fdata = st.session_state.uploaded_files[fname]
                    pdf_bytes = analysis_engine.generate_comprehensive_base_repor
                        fdata['content'],
                        fdata,
                        selected_base,
                        front_end_time,
                        back_end_time
                    )
                    st.download_button(
                        label="⬇️ Download Comprehensive Base Report",
                        data=pdf_bytes,
                        file_name=f"base_report_{selected_base.replace(' ', '_')}
                        mime="application/pdf",
                        key='comprehensive_download'
                    )

# Footer
st.markdown("---")
st.markdown("✈️ Pilot Trip Scheduling Analysis Tool | Upload up to 12 files for c
st.caption("Version: 66.3 - Comprehensive Base Report + Top-20 Longest Legs | 202
```