**** YOUTUBE VIDEO ATTACHED IN COMMENT SECTION**

Project 2 Report - FitCast: Outfit Weather Assistant
CEN 3721

a. Introduction
FitCast is a simple weather based outfit assistant built with Streamlit. The goal of the app is to help users quickly figure out what to wear based on forecasted temperature, rain probability, their personal comfort level, and the style of outfit they want. Instead of checking multiple apps or guessing what the weather "feels like," FitCast pulls real weather data from the Open-Meteo API and translates it to an easy to understand outfit suggestion.
The web app was designed to feel friendly, lightweight, and clear for everyday users like students or young professionals who just want a fast answer of what they should wear today. It is also the application I used in my Project 1 usability testing workflow.

b. Usability Goals
Before I started coding, I defined a few usability goals that guided my layout and widget choices:

1. <u>Effectiveness</u>
   The app should help users actually achieve the main task: get a reasonable outfit suggestion. I achieved this by:
   - Only asking for the minimum choices (city, date, style, comfort level).
   - Auto-jumping to the "Outfit Planner" when the user clicks *Get My Outfit.*
   - Showing informative messages if the data isn't loaded yet.
2. <u>Efficiency</u>
   Users shouldn't click through so many unnecessary steps. To address this:
   - All controls are in the sidebar.
   - One button triggers the entire forecast retrieval.
   - The suggested outfit appears immediately after data loads.
3. <u>Learnability</u>
   New users shouldn't need instructions. I supported this with:
   - Familiar widgets (selectbox, radio, checkbox, date input).
   - A simple, consistent page structure.
   - A "How to use it" section on the Home page.
4. <u>Memorability</u>
   Even if the users come back, the flow of the site will be the same.
   - A consistent light-blue theme.
   - Clear page names: Home, Outfit Planner, Forecast Details, About HCI
5. <u>Error Prevention</u>
   The app prevents confusion by …
   - Only showing outfit content when forecast data has been loaded.
   - Displaying warnings/info boxes to guide users.
6. <u>User Satisfaction</u>
   The comfort level phrases ("I am a furnace", "I run cold", etc) make the experience more fun and relatable.

c. Design Process

1. Starting Point: Program 1 (Usability Testing Tool)
   Before creating FitCast, I built Program 1, a full usability-testing platform that included:
   - A consent form
   - A demographics form
   - Tasks with timers
   - Data collection
   - Exit surveys
   - A built-in report and charts

   Because I already created a working interface in Program 1, I wanted Program 2 to match the same structure, style, and HCI patterns. This influenced my design of FitCast heavily, sidebar navigation, metrics, charts, success/error boxes, and colour theme all carried over.
2. Picking Components
   Since Program 1 already used a lot of UI components, I reused the best performing ones:
   - Radio buttons → navigation + comfort levels
   - Selectboxes → city + style
   - Date input → selecting forecast date
   - Checkbox → show/hide shoes
   - Button → fetch API + auto-navigate
   - Chart → temperature line chart
   - Map → city location
   - Dataframe → forecast table

   Every required widget in the assignment is included.
3. Implementation in Streamlit
   When I built Program 2 (FitCast), I took a good amount of structural inspiration from Program 1, even though the tools serve different purposes. In Program 1, I learned how helpful it was to have a clear sidebar, simple navigation, and success/error feedback, so I brought those patterns into Program 2.
   I started by setting up the main pages (Home, Outfit Planner, Forecast Details, and HCI Choices) using the same kind of radio navigation I used in Program 1. Then I added the controls in the side bar (selectboxes, radio buttons, date input, and a main action button), similar to how Program 1 handled form inputs.
   Unlike Program 1, which keeps everything mostly white and default, I added a light-blue theme to Program 2, to make it feel more like a small product app rather than a form-heavy testing tool. After the basic structure was set, I connected the app to the Open-Meteo API and converted the JSON data into a pandas DataFrame. From there, I built the outfit-logic function, the line chart, interactive table, map visualization, and the automatic redirect to the Outfit Planner page when the user clicks "Get my outfit."
   Just like in Program 1, I had to be careful with *st.session_state* because Streamlit reruns on every interaction. Managing that correctly helped keep the navigation smooth and prevented inputs from resetting.

4. Testing and Iteration
   I tested this app almost the same way I tested Program 1, by clicking through every page myself, trying to break things, and watching how the UI behaved when Streamlit reran the script.
   During testing, I found several issues:
   - The navigation radio button sometimes required multiple clicks (Streamlit reran issue).
   - The map kept defaulting to Miami even when selecting other cities.
   - My comfort-level radio button didn't match the old select-slider mapping, which caused errors.

- The “Get my outfit” button wasn’t redirecting reliably to the Outfit Planner.

I fixed each of these by adjusting session state, adding clearer conditions, separating sidebar inputs from forecast outputs, and making sure the app only displayed data after the forecast was loaded. This testing method was very similar to how I debugged my Program 1 tool.

5. Final Improvements

At the end of development, I polished the app by:

- Adding friendly, conversational messages rather than technical ones.
- Making the outfit recommendations fun and easy to understand.
- Ensuring the chart, map dataframe, and widgets all met the assignment requirements.

These improvements helped the app feel more complete, more fun to use, and more consistent with what I learned from developing Program 1.

d. API Integration

For this project, I used the Open-Meteo API, which is a free and public API that doesn’t require an API key. I chose it because it’s simple, well documented, and worked well for a beginner app that just needs basic weather data.

The API returns weather forecasts in JSON format, including hourly temperature and precipitation probabilities, which is exactly what I needed. Since Program 1 already taught me how to structure data (using CSV files and pandas), handling JSON in Program 2 felt more familiar.

I made a GET request inside a helper function called *fetch_weather().* After receiving the JSON, I converted it into a pandas DataFrame, cleaned it up, added extra columns like Fahrenheit, “feels like” categories, and a boolean for a “rainy hour.” The API itself was easy to use, but I did run into some small issues:

- Some days returned an empty forecast, so I added error handling to avoid crashes.
- I had to convert the API’s ISO timestamp strings into real datetime objects before charting them.
- Since the API gives hourly data, I chose the middle hour as a “representative” temperature for outfit recommendations.

Overall, the Open-Meteo API worked well and made the app feel way more dynamic than using static data.

e. Interactive Widgets

Streamlit makes adding widgets pretty easy, and because I already used them heavily in Program 1, a lot of that knowledge carried over. My app uses way more than the assignment’s minimum requirements. Here are the ones I included and why:

Required Widget

- Button - “Get my outfit,” which trigger the forecast loading and page redirect
- Checkbox - “Show shoe suggestions,” so users can choose a simpler or detailed output
- Radio buttons - Use for both page navigation and the comfort-level selector
- Selectbox - City selection + style choice
- Date input - chose the day to get an outfit for

Additional Widgets

- Slider / select-slider - Not needed here because radio buttons worked better for comfort
- Metric widgets - Shows min temp, max temp, and rainy hours
- Map - Displays the selected city
- Line Chart - Plots the temperature throughout the day
- Dataframe - Interactive hourly-forecast table

These widgets all contribute to a more hands-on, interactive feel, which was the whole goal of the assignment.

f. HCI Design Principles

One of the biggest lessons from Program 1 was that every screen show has a purpose and users shouldn't have to guess what to do next. I applied that same thinking in Program 2.
Here's how the core HCI principles show up in the final app:

(1) Visibility
The sidebar makes the main input options always available. Users don't have to hunt for them.

(2) Feedback
Every big action returns a clear message. Success messages, info messages, and warnings for missing data. Ex. "Use the sidebar and click Get my outfit first"

(3) Consistency
Program 1 and Program 2 use a similar structure:
- Sidebar navigation
- Mulit-page layout
- Repeated use of metrics, headers, and info boxes

(4) Learnability
All actions follow the same pattern: select → click → get a result.
There are no hidden steps, and the comfort-level labels use human language instead of confusing numbers.

(5) Error Prevention
If the forecast is missing, the app never crashes. It simply explains what to do next. If the user clicks ForeCast Details without loading data, it redirects them to the sidebar instructions.

(6) User Satisfaction
Instead of giving an awkward output like "wear a jacket," the app gives friendly conversational suggestions. The "shoe suggestions" checkbox adds a bit of personality and fun.

h. Conclusion

Building Program 2 felt a lot smoother because of everything I learned from Program 1. I already knew how Streamlit rerun works, how to manage user inputs, and how to structure my page in a way that feels easy to navigate. This lets me focus on more creativity. Such as, the outfit logic, theme styling, and fun comfort options.
If I had more time, I would've tried to add:

- A weekly outfit planner
- A packing list for trips

But overall, FitCast meets all the assignment requirements, uses a real public API, includes interactive visuals, and applies everything we learned about HCI.