**Personal Finance Dashboard**

Douglas Colina  
June 2025  
Streamlit, Python, Plotly, yfinance, pandas

**1. Introduction**

Personal Finance Dashboard, A Complete Web Application for Managing, Analyzing and Visualizing Financial Data This is a Streamlit application that users can use to upload transaction data, log spending patterns and investments as well as explore detailed financial reports in one place.

**Purpose and Scope**

This is a web application whose main role is to democratize financial analysis, and deliver you with pro-level finance insights without needing special skills or having to pay for expensive software. Dashboards enable raw transaction data to be translated into meaningful interaction visualizations, analysis reporting and real-time market integration.

**Target Audience**

• Solos who want financial management from personal perspective

• SMBS Tracking Expenses • Credit Card Compatibility

• Somebody who is ready to get real about their spending and financial health

**2. Usability Goals**

We had five usability goals that we wanted to make sure the fulfillment of these addressed specific user needs in how they interact with personal finances:

**2.1 Accessibility**

**Goal:** Develop a user-friendly UI that is easily accessible to people of all tech-level.

**Implementation:**

* **Clear Navigation Structure:** The sidebar contains all major controls and settings
* **Visual Hierarchy:** Custom CSS styling creates distinct sections with gradient backgrounds and consistent spacing
* **Help Text Integration:** Each input field includes contextual help text explaining its purpose
* **Tutorial System:** A collapsible welcome tutorial guides new users through the application's key features

**Code Example:**

    # Tutorial

    if st.session\_state.show\_tutorial:

        st.info("""

        👋 \*\*Welcome to your Finance Dashboard!\*\*

        1. Upload your transaction data (CSV/Excel)

        2. Set your currency and budget

        3. Explore your financial insights

        4. Set financial goals and track progress

                Have a wonderful day :-)

        """)

**2.2 Versatile and Customized**

**Goal:** allow for different user requirements and data types while enforcing consistency.

**Implementation:**

* Diverse Data Format: Validates CSV & Excel files with auto-scan format.
* Currency Flexibility- 7 major currencies with real-time conversion rates.
* Theme Customization (dark/light mode toggle with extensive styling changes [still goes wrong sometimes])
* Advanced Options Include : • Flexible Filtering: By date range, categories etc.

**Technical Achievement:**

def apply\_custom\_styling(dark\_mode=False):

    """styling with dark/light mode"""

    if dark\_mode:

        # Dark mode

    else:

        # Light mode

**2.3 Data Visualization and Comprehension**

**Goal:** Transform complex financial data into easily understandable visual representations.

**Implementation:**

* **Multi-perspective Analysis:** Four distinct tabs (Spending Analysis, Income vs Expenses, Monthly Trends, Category Breakdown) provide different viewpoints
* **Interactive Charts:** Hover details, zooming, and dynamic filtering
* **Progressive Disclosure:** Information is organized in expandable sections

**2.4 Real-time Data Integration**

**Goal:** Provide current market data and exchange rates for comprehensive financial analysis.

**Implementation:**

* **Stock Market Integration:** yfinance API provides real-time stock prices and percentage changes
* **Currency Exchange Rates:** Live exchange rate API ensures accurate multi-currency support
* **Error Handling:** Graceful degradation when APIs are unavailable, with fallback to cached data

**2.5 Performance and Responsiveness**

**Goal:** Ensure smooth user experience regardless of dataset size or complexity.

**Implementation:**

* **Efficient Data Processing:** Pandas operations are optimized for large datasets
* **Progressive Loading:** Sample data generation allows immediate user interaction
* **Responsive Design:** It works and adjust to the web (I need to finish the responsive implementation)

**3. Design Process**

**3.1 Conceptual Design Phase**

The design process began with identifying the core user journey: data input → analysis → insights → action. This linear flow informed the application's structure and navigation hierarchy.

**Initial Sketching Concepts:**

* Sidebar for controls and settings (we keep this)
* Main area for visualizations (the official dashboard)
* Tabbed interface for different analysis types (organized complexity – we changed this for a more user friendly view)
* Metric cards for key performance indicators (important)

**3.2 Prototyping and Iteration**

**First Iteration (I have time):**

* Basic data upload and display functionality
* Simple metric calculations (income, expenses, balance)
* Minimal styling (it looked awful)

**Second Iteration (Lets make it look better):**

* Advanced visualizations with Plotly integration
* Custom CSS styling system with theme support
* API integrations for real-time data (more details later but it took too much time)
* Comprehensive filtering and export capabilities

**Third Iteration (I dont have more time):**

* Enhanced error handling and user feedback (It was confusing)
* More optimizations (it took too much to load)
* budget tracking, investment monitoring
* More Styles

**3.3 Technical Architecture Decisions**

**Framework Selection:** Streamlit was chosen for its rapid development capabilities and built-in reactive programming model, allowing focus on user experience rather than backend infrastructure (And also because the teacher said so).

**Data Processing Strategy:** Pandas provides robust data manipulation capabilities while maintaining performance with large datasets.(For the visual part to the user)

**Visualization Philosophy:** Plotly was selected over static charts to enable interactive exploration, supporting the goal of user-driven analysis.(I found it in the internet)

**4. API Integration**

The application leverages multiple APIs to provide comprehensive, real-time financial data:

**4.1 Exchange Rate API Integration**

**Purpose:** Enable multi-currency support with current conversion rates.

**Implementation:**

def fetch\_exchange\_rates(base\_currency="USD"):

try:

url = f"https://api.exchangerate-api.com/v4/latest/{base\_currency}"

response = requests.get(url, timeout=3)

if response.status\_code == 200:

return response.json().get("rates", {})

except:

# Fallback to cached rates for offline functionality

return {"USD": 1.0, "EUR": 0.85, "GBP": 0.73, ...}

**User Experience Benefits:**

* Automatic currency conversion for international users
* Real-time accuracy for multi-currency portfolios

(Programmer nightmare, I am not sure if it works completely well, and the documentation is awfull I don’t know why)

**4.2 Stock Market Data Integration (yfinance)**

**Purpose:** Provide real-time investment tracking capabilities.

**Implementation:**

def fetch\_stock\_data(tickers):

stock\_data = {}

for ticker in tickers:

try:

stock = yf.Ticker(ticker)

hist = stock.history(period="2d")

# Calculate price changes and percentages

except Exception as e:

st.warning(f"Could not fetch data for {ticker}: {str(e)}")

**Advanced Features:**

* Percentage change calculations with color-coded indicators
* Error handling for invalid ticker symbols

(This one was easy to implement and I liked the design that come with it)

**4.3 API Error Handling Strategy**

**Robust Fallback System:**

* Timeout protection prevents UI freezing
* User-friendly error messages explain issues
* Graceful degradation maintains core functionality

(In some point it was necessary only to make this work it’s a project with too many insight for only one page of code but at the same time I didn’t want to start making more and more functions and class to unify it)

**5. Interactive Widgets**

The application incorporates various interactive widgets, each serving specific usability purposes:

**5.1 Data Input Widgets**

**File Uploader Widget:**

uploaded\_file = st.file\_uploader(

"Upload Transaction Data",

type=["csv", "xlsx"],

help="Upload a CSV or Excel file with columns: Date, Description, Category, Amount"

)

* **Purpose:** Enable flexible data import from multiple formats

**Sample Data Button:**

* **Implementation:** Generates realistic financial data with proper categories and trends

**5.2 Configuration Widgets**

**Currency Selector:**

* **Purpose:** Localization and multi-currency support

**Budget Input:**

* **Purpose:** Personal financial goal setting and tracking

**Dark Mode Toggle:**

st.session\_state.dark\_mode = st.toggle(

"🌙 Dark Mode",

value=st.session\_state.dark\_mode,

help="Switch between light and dark themes"

)

* **Purpose:** Accessibility and user preference accommodation (Sometimes the dark or light mode works strange in the computer)

**5.3 Analysis and Filtering Widgets**

**Date Range Picker:**

* **Purpose:** Temporal analysis and data focusing

**Multi-select Filters:**

* **Purpose:** Category and type-based data segmentation

**Tabbed Interface:**

* **Purpose:** Organized presentation of different analytical perspectives

**5.4 Export and Action Widgets**

**Download Buttons:**

* **Purpose:** Data portability and external analysis

**Report Generation:**

* **Purpose:** Automated financial health assessment

**6. HCI Design Principles**

The application adheres to established Human-Computer Interaction design principles:

**6.1 Consistency Principle**

**Implementation:**

* **Visual Consistency:** Uniform color schemes, typography, and spacing throughout the application
* **Interaction Consistency:** Similar widgets behave identically across different sections
* **Terminology Consistency:** Financial terms and labels are used consistently

**Code Example:**

def create\_custom\_metric(label, value, delta=None, help\_text=None):

"""Standardized metric card creation ensures visual consistency"""

metric\_html = f"""

<div class="metric-container">

<div class="metric-value">{value}</div>

<div class="metric-label">{label}</div>

</div>

"""

**6.2 Feedback Principle**

**Immediate Feedback:**

* Success messages after data upload
* Error messages with specific guidance
* Visual indicators for budget status

**6.3 Affordance and Signifiers**

**Visual Affordances:**

* Buttons with hover effects suggest interactivity
* Gradient backgrounds indicate interactive areas
* Icons provide intuitive function recognition
* Color coding communicates data meaning (green=positive, red=negative)

**6.4 Error Prevention and Recovery**

**Prevention Strategies:**

* Input validation prevents invalid data entry
* File format checking before processing
* Required field indicators
* Type checking for numeric inputs

**6.5 Recognition vs. Recall**

**Supporting Recognition:**

* Visual icons for each section and function
* Consistent layout patterns
* Contextual help text

**Minimizing Recall Requirements:**

* Persistent sidebar navigation
* Recent transaction display
* Automatic data format detection
* Saved user preferences

**6.6 Flexibility and Efficiency**

**Novice User Support:**

* Tutorial system for onboarding
* Sample data for immediate exploration
* Help text for all inputs
* Progressive disclosure of advanced features

**7. Conclusion**

Consequently, the development of this Financial Dashboard provided unique insights in three discipline areas: financial analysis, user experience design and web application deliverables. The project is one of those great cases that prove us how complex financial data can be easy to understand (hasrd to implement) by using carefully crafted interface design and choosing the right technology.