

AI Financial Advice Chatbot - The FinSavvys Team

Group 9: Sasi Vasa, Sudeepa Bal, Mitali Yadav, Rucha Tatwar and Sahit Ceeka

Deployment & Installation

The **AI Financial Chatbot** system combines a

-**Python based backend** that is responsible for **cleaning of data, extraction of transaction and agentic flow reasoning** with a

-**frontend** based on **React** that proved helpful for us to create an user friendly and interactive interface.

In total for this entire project, a total of **3 programming languages** were leveraged i.e.,

-**Python** for the **agentic logic and computation** at the backend side,

-**JavaScript/JSX** used for the application's **frontend development** and

-**CSS/TailwindCSS** for the **styling purpose** of the interface.

When all integrated together, **these components support**

-**PDF-to-transaction conversion,**

-**preprocessing techniques of exploratory data, and**

-**some intelligent decision-making from the conversations front.**

To install the **backend component**, the user should have the **Python 3.10 or later on a modern OS such as Windows, MacOS or Linux**. The backend depends more heavily on the cleaning data and processing techniques that are well described in our Python code. These sections do **require libraries such as the**

-**Pandas,**

-**NumPy,**

-**pdfplumber** and

-**Groq**, that is supposed to be installed within the isolated Python setup. **Creation of a virtual environment actually ensures consistency** and also avoids conflicts that can arise due to version. Once the setup is prepared, the user can then proceed to install all the **dependencies** with the help of **pip command**. Since the AI system depends heavily on the Agentic flow and reasoning of our Python code, the **backend code also integrates with the Groq API for intelligently classifying and the decision-making abilities**, The user is expected to

-**create a free Groq account, generate the API key and**

-**store it within an .env file in the backend directory,**

-**with the help of the format GROQ_API_KEY=your_key_here.**

This environment variable is then automatically loaded by the backend system when taking care of queries. After the configurational settings, **the backend can be launched through a very standard Python execution command.**

-**An updated and modern web browser like the Google Chrome or the Microsoft Edge is mandatory to run the application smoothly**, since the application leverages **JavaScript and Tailwind utility classes**.

-From the hardware front, **the application does not need any specialized GPU acceleration** since all the reasoning of the advanced models is carried out by Groq API.

-A standard **CPU-based system with at least 8 GB of RAM should be sufficient** enough to support data processing, PDF parsing and executing both the backend and frontend.

The front installation starts with **installing Node.js version 18 or higher**, which does provide **npm** for taking care of **JavaScript dependencies**. The user then pays attention to the frontend project part, and installs all the packages required for running the **React interface**. Since the interface leverages TailwindCSS for the purpose of layout and styling, **the Tailwind toolchain must be installed explicitly** with the help of the command **npm install -D tailwindcss postcss autoprefixer**.

-Tailwind is then initialized with the help of **npx tailwindcss init -p**, and its directives should be within the project's main stylesheet.

-Once the installation is all complete, the user may start the frontend with the command **npm run dev**, that is responsible for the launch of a development server accessible through the browser.

To connect the 2 components, the frontend's API configuration file points to the backend server URL. When **both the systems are up and running parallelly**, the pipeline becomes fully operational, the **user then proceeds to upload a PDF**, for which the backend converts into some organized transactions for the purpose of analysis, while the agentic flow orchestrates **-classification,**

-summarization and

-some financial recommendations.

The Groq API processes natural-language queries and this way the system returns enriched replies to the React interface.

Regardless of the hosting platform, **the Groq API key must be stored securely with the help of the platform's secrets or environment variable management**. With the help of the above stated installation and deployment techniques, it makes sure that any user can configure the whole system without any more supporting guidance while preserving the consistency across the environments.

Main Features

1. Easy File Upload

Users can upload their bank statements in PDF, Excel or CSV format. The system reads these files automatically, removing the need to manually type or scan through long transaction lists. This makes getting started quick and effortless.

2. Automatic Transaction Grouping

As soon as the statements are uploaded, the chat bot files and buckets every transaction into simple descriptions like expenses, wants, and savings. Hence, users can instantly understand their money distribution without spending time to self categorize.

3. Clear Spending Breakdown

The system calculates users spending and categorizes the amount spent in each bucket. It also shows the totals of needs and wants while listing out savings as well, allowing users to easily understand where their money was spent each month.

4. Personalized Budget Setup

Users are able to enter their income for each month and the chatbot determines optimal figures for needs, wants, and savings. This is then aligned with the users spending goals. Doing this creates a spending plan tailored to their financial behavior.

5. Budget Analysis and Insights

By comparing the money spent and money projected in the budget, the chatbot informs the user whether they are on pace or over budget. It indicates potential areas of concern and the remaining cash in every section of the budget.

6. Add New Transactions Anytime

As soon as users input a new auto spending item, the system modifies their total instantaneously. It calculates the new budgeted intervals, determines if the user is still spending within their budget and provides minor tips to keep them on course.

7. Ask Questions in Natural Language

Users can ask the chatbot simple questions like “What did I spend the most on?” or “Can I afford to go out this weekend?” The system responds with clear answers based on the user’s real financial data, making the experience feel like talking to a personal assistant.

8. Simple Chat-Based Interface

Users can upload documents, send and receive financial data, and analyze their reports in a dedicated chat. This keeps all information in one easily accessible and user friendly chat, simplifying the user experience.

Primary Walkthrough

1. Begin with the backend service and creation of the Groq API Key: Creation and activation of a Python virtual environment with the installation of all the Python requirements (pandas, pdfplumber, scikit-learn and etc), next comes the Groq API key storage in an .env file as GROQ_API_KEY=your_key_here. The backend’s FinancialChatbot class reads that key and leverages it to call Groq for classification and chat replies (call_groq_api)

The screenshot shows a Google Colab notebook titled "Copy of 6150 Financial Chatbot.ipynb". The notebook displays a table of transaction history and some Python code for initializing a financial chatbot.

Your Transaction History:

	amount	description	category	category_type	date
0	1200	Monthly Rent	Housing	expenses	2024-11-01
1	45	Grocery Shopping	Food	expenses	2024-11-02
2	150	Dinner with Friends	Dining	wants	2024-11-03
3	80	Gas Station	Transportation	expenses	2024-11-04
4	200	New Clothes	Shopping	expenses	2024-11-05
5	35	Coffee Shop	Dining	wants	2024-11-06
6	500	401k Contribution	Retirement	savings	2024-11-07
7	60	Internet Bill	Utilities	expenses	2024-11-08
8	25	Movie Tickets	Entertainment	wants	2024-11-09
9	100	Gym Membership	Health	expenses	2024-11-10
10	500	Emergency Fund	Savings	savings	2024-11-11
11	75	Lunch Out	Dining	wants	2024-11-12

Total Transactions: 12
Total Amount: \$2,770.00

Step 2: Initialize the Financial Chatbot

```
[1]: from google.colab import userdataset
# Set your Grok API key
# Get your API key from: https://console.x.ai/
GROQ_API_KEY = userdataset.get('GROQ_API_KEY')# Replace with your actual API key

# Initialize the chatbot with your transaction data
chatbot = FinancialChatbot(GROQ_API_KEY, transactions_df)
```

The screenshot shows a Google Colab notebook titled "Copy of 6150 Financial Chatbot.ipynb". The notebook interface includes a toolbar with File, Edit, View, Insert, Runtime, Tools, Help, and a search bar for Commands, Code, Text, and Run all.

The code cell contains Python code for setting environment variables and downloading a dataset:

```
from google.colab import userdata
import pandas as pd
import os
from financial_chatbot import FinancialChatbot

os.environ["KAGGLE_KEY"] = userdata.get('KAGGLE_KEY')
os.environ["KAGGLE_USERNAME"] = userdata.get('KAGGLE_USERNAME')
```

The output of the cell shows the execution of a command to download a dataset from Kaggle:

```
| kaggle datasets download -d pradeepkumar2424/usa-banking-transactions-dataset-2023-2024
|   ! unzip "usa-banking-transactions-dataset-2023-2024.zip"
```

The output continues with details about the dataset download:

```
... Dataset URL: https://www.kaggle.com/datasets/pradeepkumar2424/usa-banking-transactions-dataset-2023-2024
License(s): CC0-1.0
Downloading usa-banking-transactions-dataset-2023-2024.zip to /content
  0% 0.00/580k [00:00<?, ?B/s]
100% 580k/580k [00:00<00:00, 1.01GB/s]
Archive: usa-banking-transactions-dataset-2023-2024.zip
inflating: Banking_Transactions_USA_2023_2024.csv
```

2. Run the backend: The backend script serves as the core processing system of this financial chatbot. It loads the cleaned transactional data, carries on with the task of category classification using the Groq API, and then executes the agentic flow reasoning that calculates the spending summaries, budget adherence and some financial recommendations. The main functionalities of this system includes, initializing the chatbot, setting some monthly income values, defining budget goals across different expenditure categories, inclusion of transactions and generating initial financial analysis. On running the backend, this script carries out the tasks of transaction categorization, conversational processing and summarizing the user's financial data.

```
374 # Example usage function for notebook
375 def example_usage():
376     """
377     Example of how to use the FinancialChatbot in a notebook.
378     """
379     # Sample transaction data
380     sample_data = {
381         'amount': [1200, 45, 150, 80, 200, 35, 500],
382         'description': ['Rent', 'Groceries', 'Dinner out', 'Gas', 'Shopping', 'Coffee', 'Investment'],
383         'category': ['Housing', 'Food', 'Dining', 'Transportation', 'Shopping', 'Dining', 'Savings'],
384         'category_type': ['expenses', 'expenses', 'wants', 'expenses', 'wants', 'wants', 'savings']
385     }
386
387     df = pd.DataFrame(sample_data)
388
389     # Initialize chatbot (replace with your actual API key)
390     API_KEY = "gsk_Evd6t3cxyjhgrsPyHzGgzb3FVN5wtIq07yc1LzQPgBjRs3GcZ"
391     chatbot = FinancialChatbot(API_KEY, df)
392
393     # Step 1: Set income
394     chatbot.set_income(5000)
395
396     # Step 2: Set budget goals (optional - uses defaults if not called)
397     chatbot.set_budget_goals(expenses=0.70, wants=0.20, savings=0.10)
398
399     # Step 3: Get initial analysis
400     print("\n" + "="*60)
401     print("INITIAL ANALYSIS")
402     print("="*60)
403     #analysis = chatbot.get_initial_analysis()
404     #print(analysis)
405
406     # Step 4: Add new transactions interactively
407     print("\n" + "="*60)
408     print("ADDING NEW TRANSACTION")
409     print("="*60)
410     response = chatbot.add_transaction(75, "Bought new shoes", "Shopping", "wants")
411     print(response)
412
413     # Step 5: Add another transaction (auto-classify)
414     print("\n" + "="*60)
415     print("ADDING ANOTHER TRANSACTION (AUTO-CLASSIFIED)")
416     print("="*60)
417     response = chatbot.add_transaction(120, "Dinner at Italian restaurant")
```

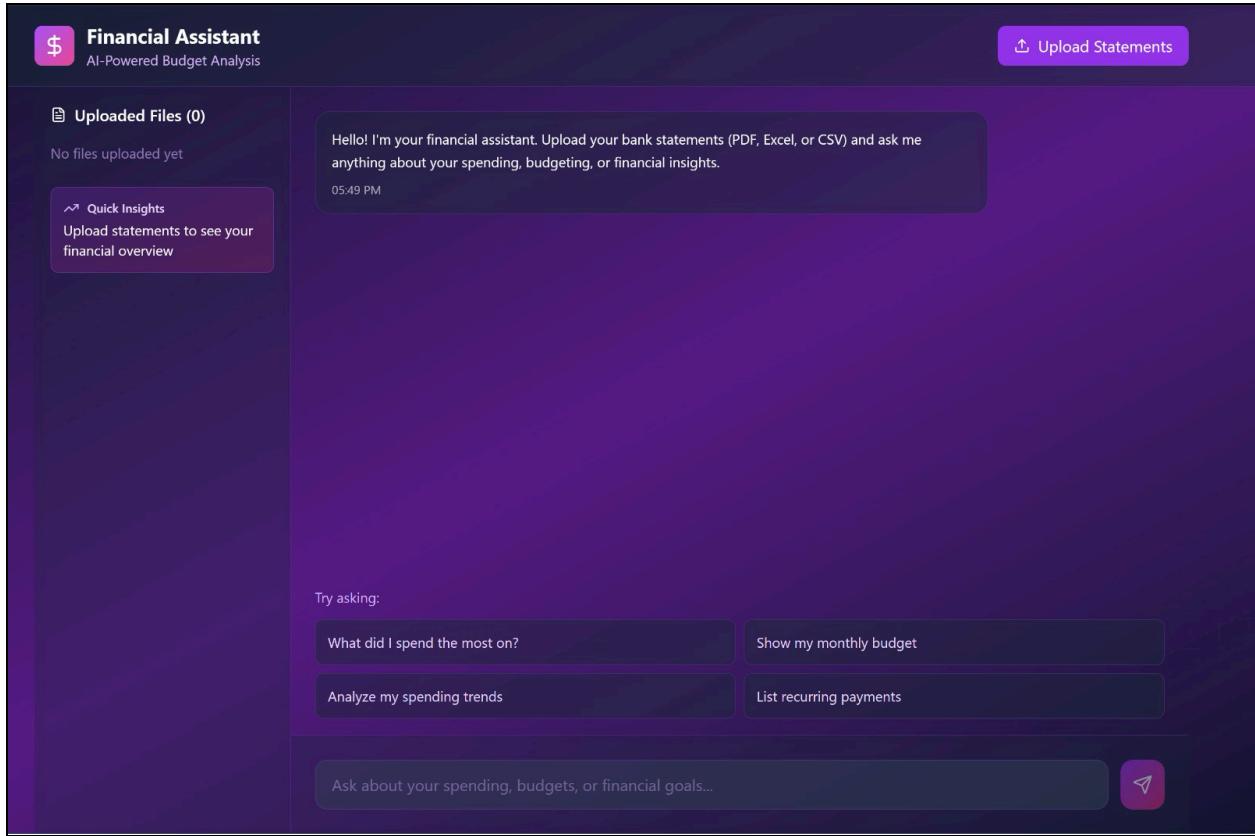
3. Initializing the frontend: The frontend environment is initialized by installing all the required Javascript dependencies leveraging npm install. TailwindCSS and its associated components are added basically with the help of npm install -D tailwindcss postcss autoprefixer, keeping the interface styling system as ready. The frontend development server is primarily launched with the help of npm run dev.

The screenshot shows a code editor with two files open: `financial_chatbot.py` and `FinancialChatbot.jsx`. The `FinancialChatbot.jsx` file contains the following code:

```
financial-chatbot > src > components > FinancialChatbot.jsx > FinancialChatbot
1 import React, { useState, useRef, useEffect } from 'react';
2 import { Upload, Send, DollarSign, FileText, X, TrendingUp, PieChart, Calendar } from 'lucide-react';
3
4 export default function FinancialChatbot() {
5   const [messages, setMessages] = useState([
6     {
7       role: 'assistant',
8       content: 'Hello! I\'m your financial assistant. Upload your bank statements (PDF, Excel, or CSV) and ask me anything about your finances!',
9       timestamp: new Date()
10    }
11  ]);
12  const [input, setInput] = useState('');
13  const [uploadedFiles, setUploadedFiles] = useState([]);
14  const [isProcessing, setIsProcessing] = useState(false);
15  const messagesEndRef = useRef(null);
16  const fileInputRef = useRef(null);
17
18  const scrollToBottom = () => {
19    messagesEndRef.current.scrollIntoView({ behavior: 'smooth' });
20  };
21
22  useEffect(() => {
23    scrollToBottom();
24  }, [messages]);
25
26  const handleFileUpload = async (e) => {
27    const files = Array.from(e.target.files);
28    const validTypes = [
29      'application/pdf',
30      'application/vnd.ms-excel',
31      'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet',
32      'text/csv'
33    ];
34
35    const validFiles = files.filter(file =>
36      validTypes.includes(file.type) || file.name.endsWith('.csv')
37    );
38
39    if (validFiles.length === 0) {
40      alert('Please upload PDF, Excel, or CSV files only.');
41      return;
42    }
43
44    setIsProcessing(true);
45  }
46}
```

The right side of the interface features a sidebar titled "Build with agent mode" which includes a "Generate Agent Instructions to onboard AI onto your codebase." button. A tooltip provides instructions: "AI responses may be inaccurate. Add context (#), extensions (@), commands, and variables (\$).". Below the sidebar, there's a "Agent" dropdown set to "Auto" and a "Send" button.

4. Dashboard display: Immediately after opening the frontend URL, it actually displays the React-based interface, which shows the dashboard that contains the file-upload section, the list of already uploaded files, the chat area and message input section.



5. User interaction: Users start interacting with the system by uploading their financial statement files in the form of PDF, CSV or XLSX formats. The selected files immediately appear on the left-hand side list of files. The upload interface actually performs all the file handling operations that are defined in the frontend component.

The screenshot shows a Microsoft Excel spreadsheet titled "bank_statement_ex". The ribbon menu is visible at the top, with "Home" selected. The table below contains transactional data:

	A	B	C	D	E	F	G	H	I	J	K
1	Date	Description	Ref.	Withdrawals	Deposits	Balance					
2	2003-10-08	Previous balance				\$0.55					
3	2003-10-14	Payroll Deposit - HOTEL			\$ 694.81	\$695.36					
4	2003-10-14	Textbook Payment - MASTERCARD	9685	\$ 200.00		\$495.36					
5	2003-10-16	ATM Withdrawal - INTERAC	3990	\$ 21.25		\$474.11					
6	2003-10-16	Fees - Interac		\$ 1.50		\$472.61					
7	2003-10-20	Interac Purchase - ELECTRONICS	1975	\$ 2.99		\$469.62					
8	2003-10-21	UNC Charlotte Tuition Payment	3314	\$ 300.00		\$169.62					
9	2003-10-22	ATM Withdrawal - FIRST BANK	0064	\$ 100.00		\$69.62					
10	2003-10-23	Interac Purchase - SUPERMARKET	1559	\$ 29.08		\$40.54					
11	2003-10-24	Interac Refund - ELECTRONICS	1975		\$ 2.99	\$43.53					
12	2003-10-27	Telephone Bill Payment - VISA	2475	\$ 6.77		\$36.76					
13	2003-10-28	Payroll Deposit - HOTEL			\$ 694.81	\$731.57					
14	2003-11-03	Web Funds Transfer - From SAVINGS	2620		\$ 50.00	\$781.57					
15	2003-11-03	Pre-Auth. Payment - INSURANCE		\$ 33.55		\$748.02					
16	2003-11-03	Cheque No. - 409		\$ 100.00		\$648.02					
17	2003-11-06	Rent Payment		\$ 710.49		-\$62.47					

6. Extraction of transactional data: Once the files are uploaded, the backend system automatically extracts transactional data leveraging the routines stated in the backend files. The system identifies the financial fields, standardizes column names and produces an organized transaction table with dates, descriptions and amounts.

The screenshot shows a Google Colab notebook titled "Copy of 6150 Financial Chatbot.ipynb". The code in the cell is for loading transaction files (CSV, XLSX, PDF) into a pandas DataFrame. It handles file uploads, path extraction, and data parsing using pd.read_csv, pd.read_excel, and pdfplumber. The code includes error handling for unsupported file types and normalization of headers.

```
def load_transactions_interactive(keep_extra=False) -> pd.DataFrame:
    print("Please upload your transaction file (.csv, .xlsx, or .pdf)")
    uploaded = files.upload()
    path = list(uploaded.keys())[0]
    print(f"Uploaded: {path}")

    p = path.lower()
    # --- Load file ---
    if p.endswith(".csv"):
        t = pd.read_csv(path, dtype=str) # read as str, parse later
    elif p.endswith((".xls", ".xlsx")):
        # read as str to preserve formatting; many bank sheets have blank/extraneous columns
        t = pd.read_excel(path, dtype=str)
    elif p.endswith(".pdf"):
        rows = []
        with pdfplumber.open(path) as pdf:
            for page in pdf.pages:
                table = page.extract_table()
                if not table:
                    continue
                headers = [norm(h) for h in table[0]]
                for r in table[1:]:
                    rows.append(dict(zip(headers, r)))
        t = pd.DataFrame(rows)
    else:
        raise ValueError("Unsupported file type")

    # Drop fully-empty columns
    t = t.dropna(axis=1, how="all")

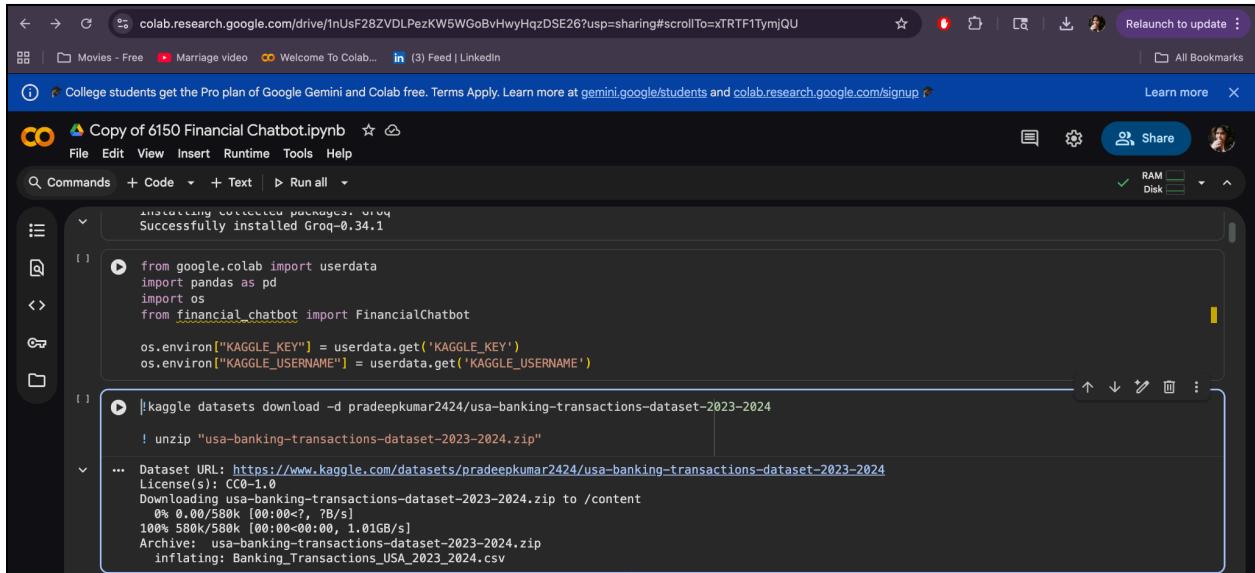
    # Normalize headers
    return t
```

The screenshot shows the execution result of the previous code cell. The output displays the first few rows of the DataFrame 'transactions' created by the function. The table includes columns for transaction date, description, amount, reference, and running balance. A note indicates that the 'transaction_date' column was loaded and standardized.

	transaction_date	transaction_description	transaction_amount	reference	running_balance
0	2003-10-08	Previous balance	0.00	nan	0.55
1	2003-10-14	Payroll Deposit - HOTEL	694.81	nan	695.36
2	2003-10-14	Textbook Payment - MASTERCARD	-200.00	9685	495.36
3	2003-10-16	ATM Withdrawal - INTERAC	-21.25	3990	474.11
4	2003-10-16	Fees - Interac	-1.50	nan	472.61

7. Data cleaning workflow: Immediately after the extraction phase, the data cleaning workflow gets leveraged. The cleaning process overall normalizes headers, converts dates and also handles

any missing entries from the uploaded files. It is also responsible for preparing numerical and categorical attributes that in turn generates a very clean Dataframe that is ready for the analysis phase.



Copy of 6150 Financial Chatbot.ipynb

```
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

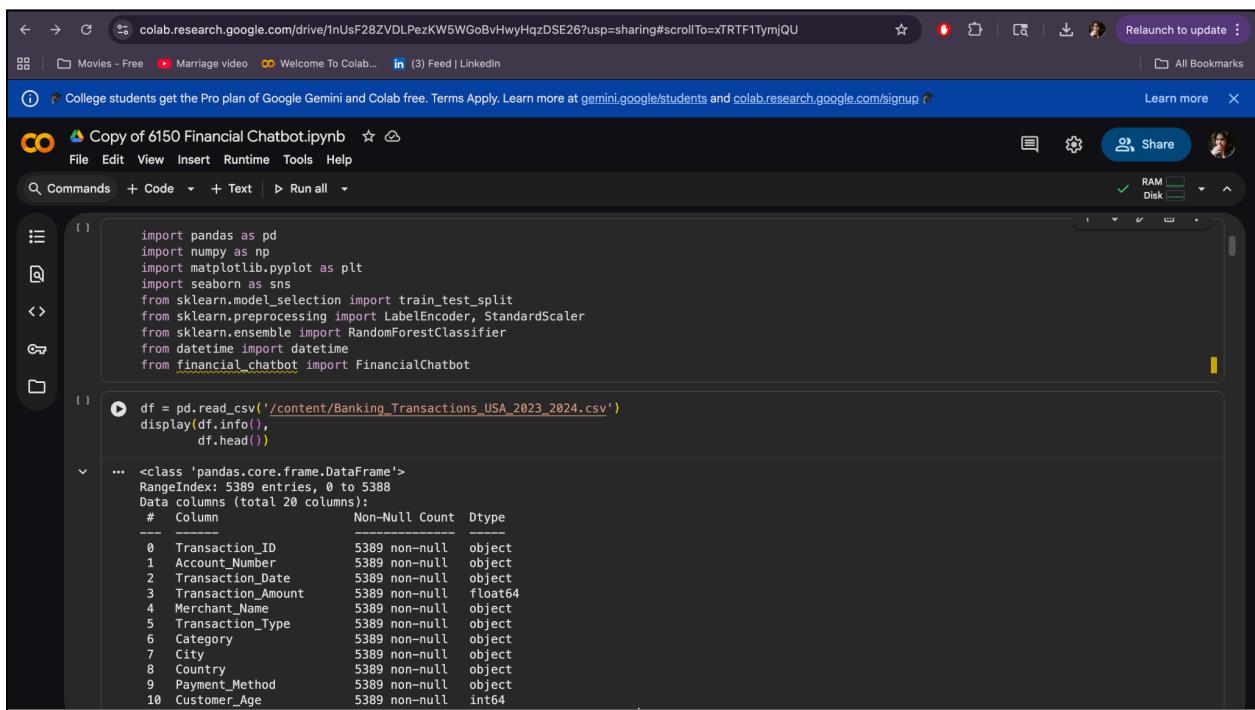
Installing collected packages: Grod
Successfully installed Grod-0.34.1

from google.colab import userdata
import pandas as pd
import os
from financial_chatbot import FinancialChatbot

os.environ["KAGGLE_KEY"] = userdata.get("KAGGLE_KEY")
os.environ["KAGGLE_USERNAME"] = userdata.get("KAGGLE_USERNAME")

!kaggle datasets download -d pradeepkumar2424/usa-banking-transactions-dataset-2023-2024
!unzip "usa-banking-transactions-dataset-2023-2024.zip"

Dataset URL: https://www.kaggle.com/datasets/pradeepkumar2424/usa-banking-transactions-dataset-2023-2024
License(s): CC0-1.0
Downloading usa-banking-transactions-dataset-2023-2024.zip to /content
  0% 0.00/580k [0:00<-, 7B/s]
100% 580k/580k [0:00<0:00, 1.01GB/s]
Archive: usa-banking-transactions-dataset-2023-2024.zip
  inflating: Banking_Transactions_USA_2023_2024.csv
```



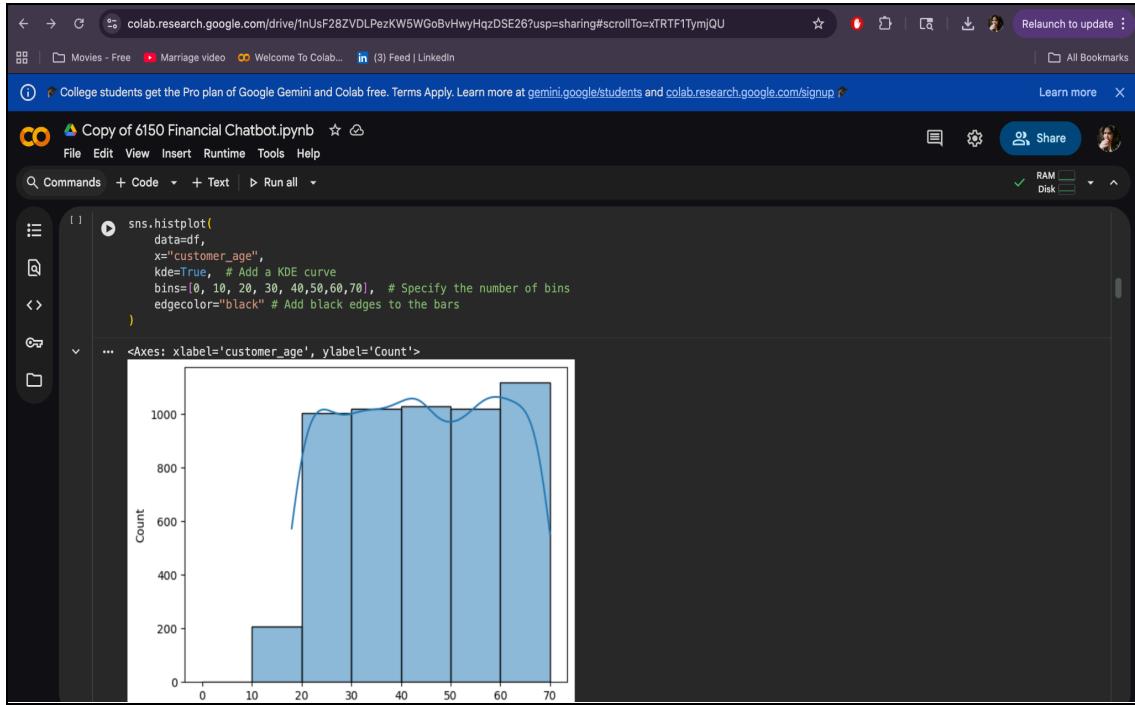
Copy of 6150 Financial Chatbot.ipynb

```
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all

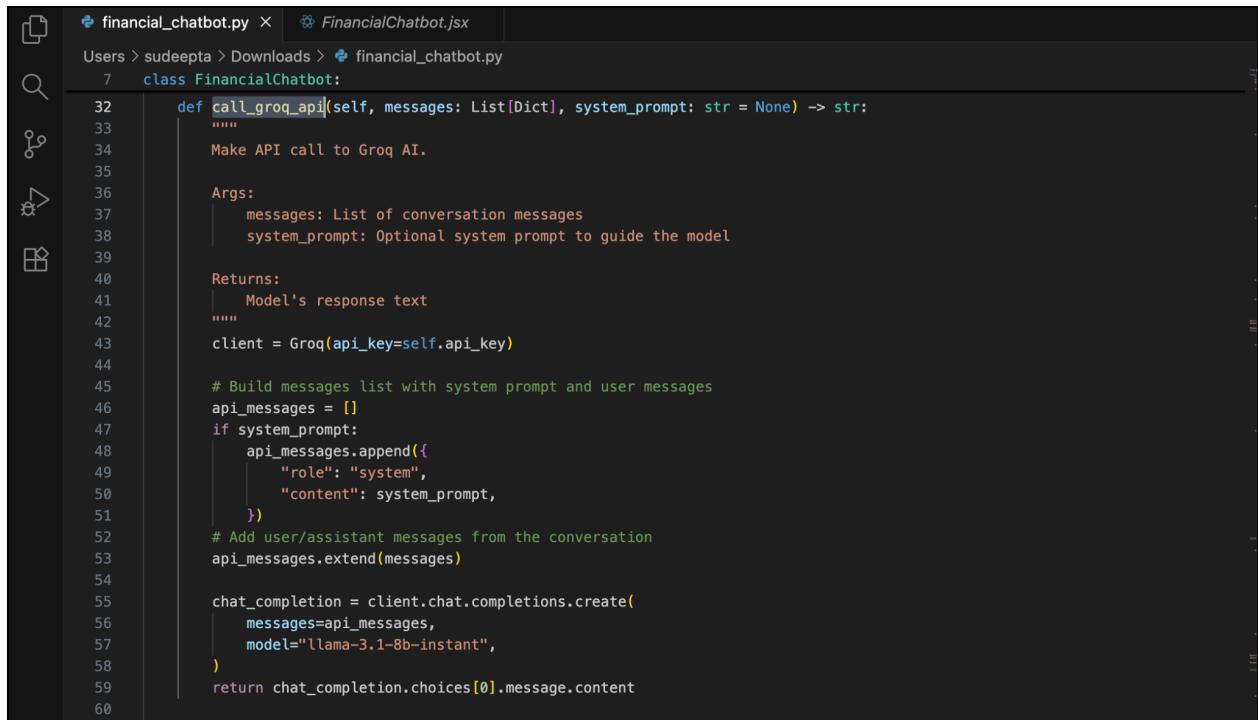
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from datetime import datetime
from financial_chatbot import FinancialChatbot

df = pd.read_csv('/content/Banking_Transactions_USA_2023_2024.csv')
display(df.info())
df.head()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5389 entries, 0 to 5388
Data columns (total 20 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   Transaction_ID  5389 non-null    object  
 1   Account_Number  5389 non-null    object  
 2   Transaction_Date 5389 non-null    object  
 3   Transaction_Amount 5389 non-null    float64 
 4   Merchant_Name   5389 non-null    object  
 5   Transaction_Type 5389 non-null    object  
 6   Category        5389 non-null    object  
 7   City            5389 non-null    object  
 8   Country          5389 non-null    object  
 9   Payment_Method   5389 non-null    object  
 10  Customer_Age    5389 non-null    int64
```

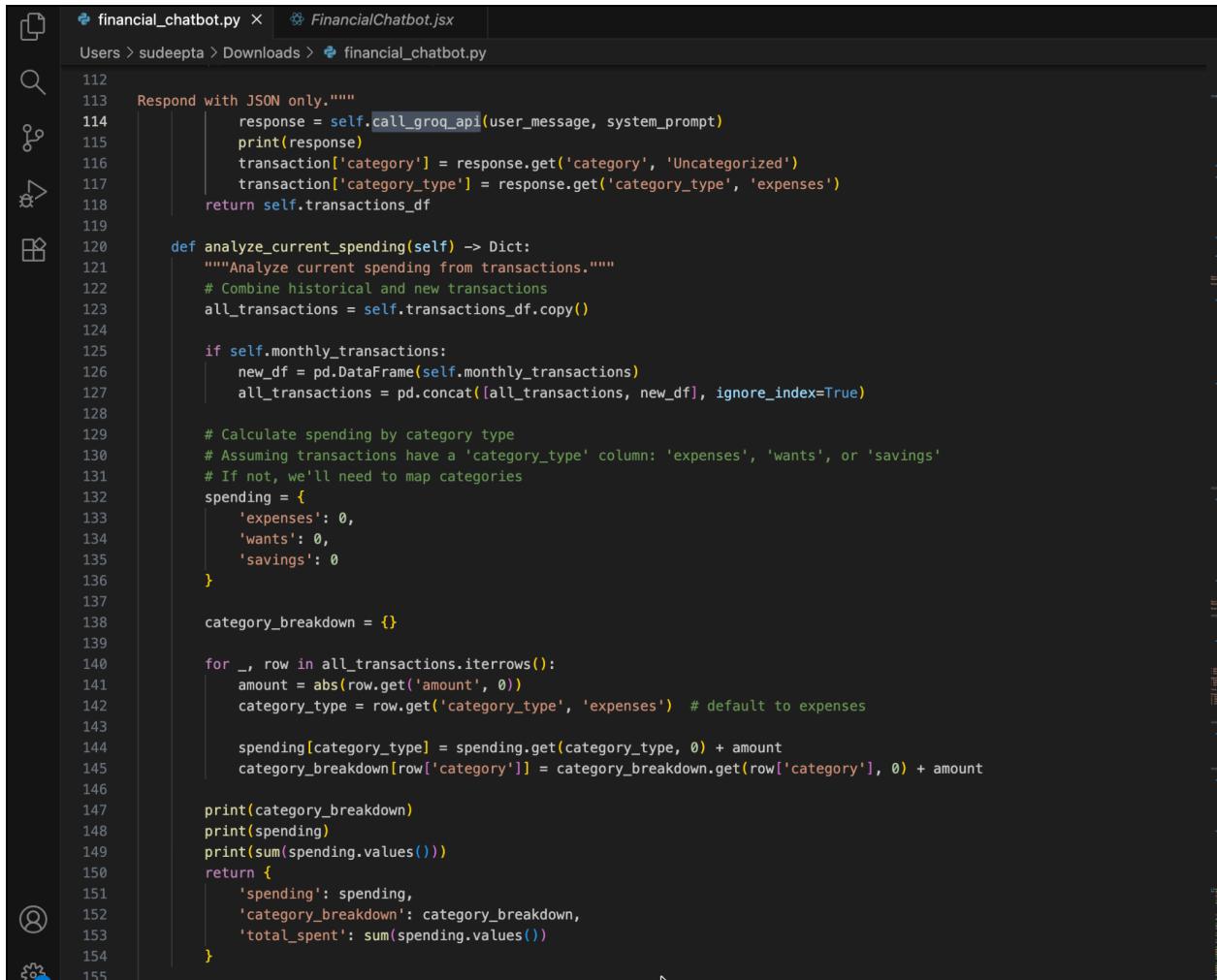


8. Agentic Reasoning pipeline: Now as the data cleaning process is done, the agentic reasoning pipeline takes over. At this stage, the backend logic applies the process of the Large Language Model (LLM) integrated through the Groq API - implemented in the call_groq_api method. The LLM classifies each transaction, interprets user queries, generates the financial analysis and also shapes the intelligent aspect of the chatbot. The classification actually determines categories such as groceries, rent, entertainment and so on and supports downstream insights including summaries on the expenditure and recommendations on the budget. **Earlier, we tried using several traditional ML models like RF, XGboost but the accuracy prediction did not come to a sophisticated percentage due to which we had to switch to LLMs for classification purposes.**



The screenshot shows a code editor interface with two tabs open: `financial_chatbot.py` and `FinancialChatbot.jsx`. The `financial_chatbot.py` tab is active, displaying Python code for a `FinancialChatbot` class. The code includes docstrings for the `call_groq_api` method, which describes making an API call to Groq AI. It shows how to build a messages list with system prompt and user messages, and then use the `Groq` client to create a chat completion.

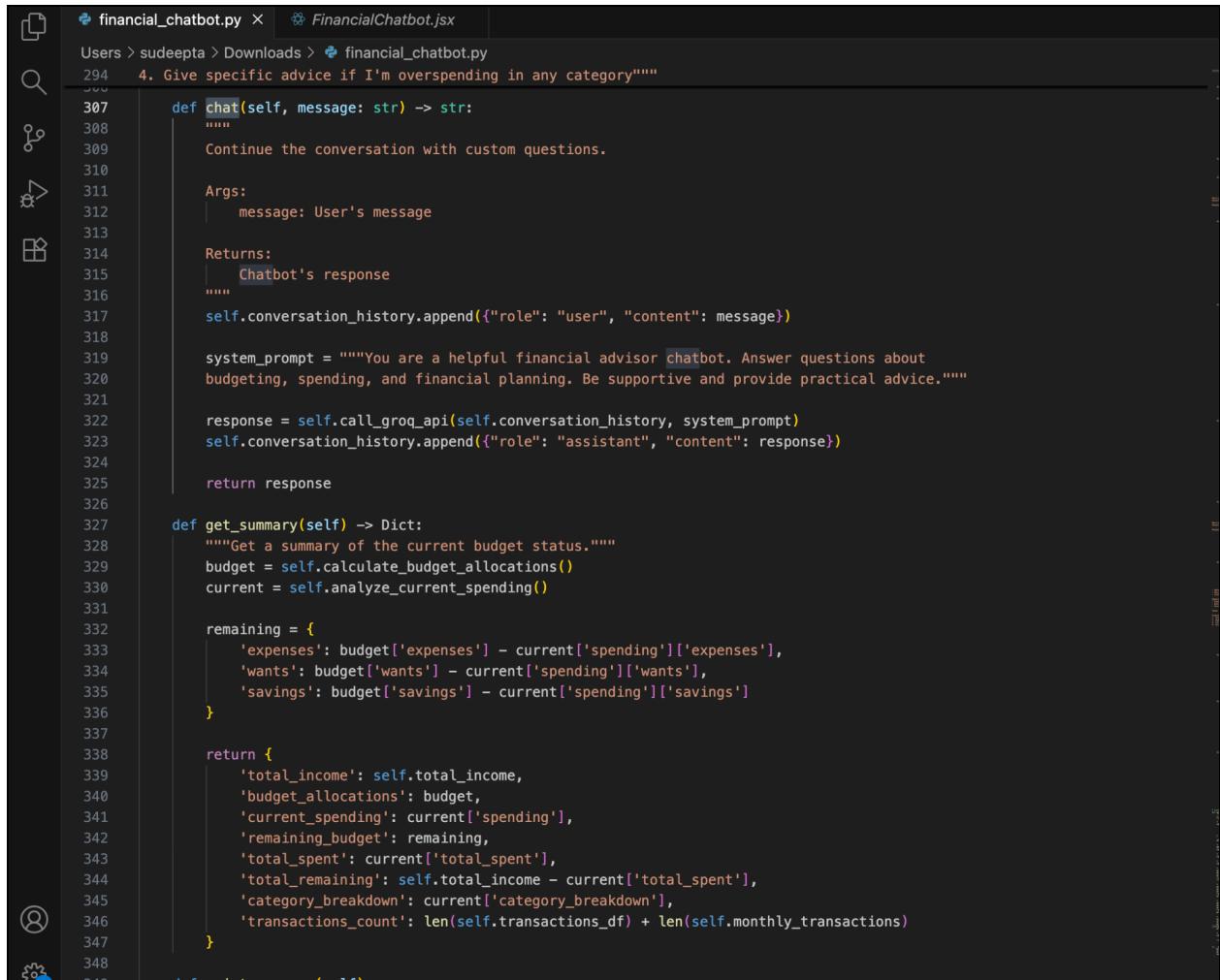
```
7  class FinancialChatbot:
32      def call_groq_api(self, messages: List[Dict], system_prompt: str = None) -> str:
33          """
34              Make API call to Groq AI.
35
36          Args:
37              messages: List of conversation messages
38              system_prompt: Optional system prompt to guide the model
39
40          Returns:
41              Model's response text
42          """
43
44          client = Groq(api_key=self.api_key)
45
46          # Build messages list with system prompt and user messages
47          api_messages = []
48          if system_prompt:
49              api_messages.append({
50                  "role": "system",
51                  "content": system_prompt,
52              })
53          # Add user/assistant messages from the conversation
54          api_messages.extend(messages)
55
56          chat_completion = client.chat.completions.create(
57              messages=api_messages,
58              model="llama-3.1-8b-instant",
59          )
60
61          return chat_completion.choices[0].message.content
```



The screenshot shows a code editor interface with two tabs: 'financial_chatbot.py' and 'FinancialChatbot.jsx'. The 'financial_chatbot.py' tab is active, displaying Python code. The code defines a class with methods for responding with JSON, analyzing current spending, and calculating spending by category type. It uses pandas DataFrames to manage transaction data and Groq API calls to get category information.

```
112     Respond with JSON only."
113     response = self.call_groq_api(user_message, system_prompt)
114     print(response)
115     transaction['category'] = response.get('category', 'Uncategorized')
116     transaction['category_type'] = response.get('category_type', 'expenses')
117     return self.transactions_df
118
119
120     def analyze_current_spending(self) -> Dict:
121         """Analyze current spending from transactions."""
122         # Combine historical and new transactions
123         all_transactions = self.transactions_df.copy()
124
125         if self.monthly_transactions:
126             new_df = pd.DataFrame(self.monthly_transactions)
127             all_transactions = pd.concat([all_transactions, new_df], ignore_index=True)
128
129         # Calculate spending by category type
130         # Assuming transactions have a 'category_type' column: 'expenses', 'wants', or 'savings'
131         # If not, we'll need to map categories
132         spending = {
133             'expenses': 0,
134             'wants': 0,
135             'savings': 0
136         }
137
138         category_breakdown = {}
139
140         for _, row in all_transactions.iterrows():
141             amount = abs(row.get('amount', 0))
142             category_type = row.get('category_type', 'expenses') # default to expenses
143
144             spending[category_type] = spending.get(category_type, 0) + amount
145             category_breakdown[row['category']] = category_breakdown.get(row['category'], 0) + amount
146
147             print(category_breakdown)
148             print(spending)
149             print(sum(spending.values()))
150             return {
151                 'spending': spending,
152                 'category_breakdown': category_breakdown,
153                 'total_spent': sum(spending.values())
154             }
155
```

9. User interaction with the analysis: User interacts with the analysis section with the help of the chat interface. Messages that are entered in the input field are processed through the backend's chat method, which actually generates AI replies leveraging Groq. The assistant replies to questions like spending breakdown, financial summaries month-wise, while also supporting the inclusion of new transactions through natural language.



The screenshot shows a code editor interface with two tabs: 'financial_chatbot.py' and 'FinancialChatbot.jsx'. The 'financial_chatbot.py' tab is active, displaying Python code for a financial chatbot. The code includes functions for handling user messages and generating summaries based on budget data. The 'FinancialChatbot.jsx' tab is visible in the background.

```
Users > sudeepa > Downloads > financial_chatbot.py
294 4. Give specific advice if I'm overspending in any category"""
295
307     def chat(self, message: str) -> str:
308         """
309             Continue the conversation with custom questions.
310
311         Args:
312             |   message: User's message
313
314         Returns:
315             |   Chatbot's response
316         """
317         self.conversation_history.append({"role": "user", "content": message})
318
319         system_prompt = """You are a helpful financial advisor chatbot. Answer questions about
320             budgeting, spending, and financial planning. Be supportive and provide practical advice."""
321
322         response = self.call_groq_api(self.conversation_history, system_prompt)
323         self.conversation_history.append({"role": "assistant", "content": response})
324
325
326         return response
327
328     def get_summary(self) -> Dict:
329         """
330             Get a summary of the current budget status.
331         """
332         budget = self.calculate_budget_allocations()
333         current = self.analyze_current_spending()
334
335         remaining = {
336             'expenses': budget['expenses'] - current['spending']['expenses'],
337             'wants': budget['wants'] - current['spending']['wants'],
338             'savings': budget['savings'] - current['spending']['savings']
339         }
340
341         return {
342             'total_income': self.total_income,
343             'budget_allocations': budget,
344             'current_spending': current['spending'],
345             'remaining_budget': remaining,
346             'total_spent': current['total_spent'],
347             'total_remaining': self.total_income - current['total_spent'],
348             'category_breakdown': current['category_breakdown'],
349             'transactions_count': len(self.transactions_df) + len(self.monthly_transactions)
350         }
```

10. Inclusion of comprehensive summaries: Throughout the interaction phase, users can also ask for the comprehensive summaries. The backend does provide total spending, distributions ordered category-wise, remaining budget amounts and financial adjustments recommended by the system.

← → ⌂ colab.research.google.com/drive/1nUsF28ZVDLPezKW5WGoBvHqzDSE26?usp=sharing#scrollTo=xTRTF1TymjQU

Movies - Free Marriage video Welcome To Colab... (3) Feed | LinkedIn .api(self.conversation_history)| 0/0

College students get the Pro plan of Google Gemini and Colab free. Terms Apply. Learn more at [gemini.google/students](#) and [colab.research.google.com/sic](#)

Copy of 6150 Financial Chatbot.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

```
[ ] ,  
    'category_type': [  
        'expenses', 'expenses', 'wants', 'expenses', 'wants', 'wants', 'savings',  
        'expenses', 'wants', 'expenses', 'savings', 'wants'  
    ],  
    'date': pd.date_range(start='2024-11-01', periods=12, freq='D')  
}  
  
transactions_df = pd.DataFrame(sample_transactions)  
  
# Display the transactions  
print("Your Transaction History:")  
print(transactions_df)  
print(f"\nTotal Transactions: {len(transactions_df)}")  
print(f"Total Amount: ${transactions_df['amount'].sum():,.2f}")
```

Your Transaction History:

	amount	description	category	category_type	date
0	1200	Monthly Rent	Housing	expenses	2024-11-01
1	45	Grocery Shopping	Food	expenses	2024-11-02
2	150	Dinner with Friends	Dining	wants	2024-11-03
3	80	Gas Station	Transportation	expenses	2024-11-04
4	200	New Clothes	Shopping	wants	2024-11-05
5	35	Coffee Shop	Dining	wants	2024-11-06
6	500	401k Contribution	Retirement	savings	2024-11-07
7	60	Internet Bill	Utilities	expenses	2024-11-08
8	25	Movie Tickets	Entertainment	wants	2024-11-09
9	100	Gym Membership	Health	expenses	2024-11-10
10	300	Emergency Fund	Savings	savings	2024-11-11
11	75	Lunch Out	Dining	wants	2024-11-12

Total Transactions: 12
Total Amount: \$2,770.00

```

print("\n" + "="*80)
print("INITIAL BUDGET ANALYSIS")
print("="*80 + "\n")

analysis = chatbot.get_initial_analysis()
print(analysis)

=====
INITIAL BUDGET ANALYSIS
=====

{'Housing': 1200, 'Food': 45, 'Dining': 260, 'Transportation': 80, 'Shopping': 200, 'Retirement': 500, 'Utilities': 60, 'Entertainment': 25, 'Health': 2770
Let's dive into your budget analysis.

**Overview of Spending vs Budget:**

You're currently overspending in the Expenses category by a significant margin ($1,485.00). However, you're underutilizing the Wants and Savings categories.

You've allocated 70% of your income for Expenses, 20% for Wants, and 10% for Savings. To achieve these goals, you need to make adjustments in your spending.

**Overspending Categories:**

Based on your Category Breakdown, the top three areas where you're overspending are:

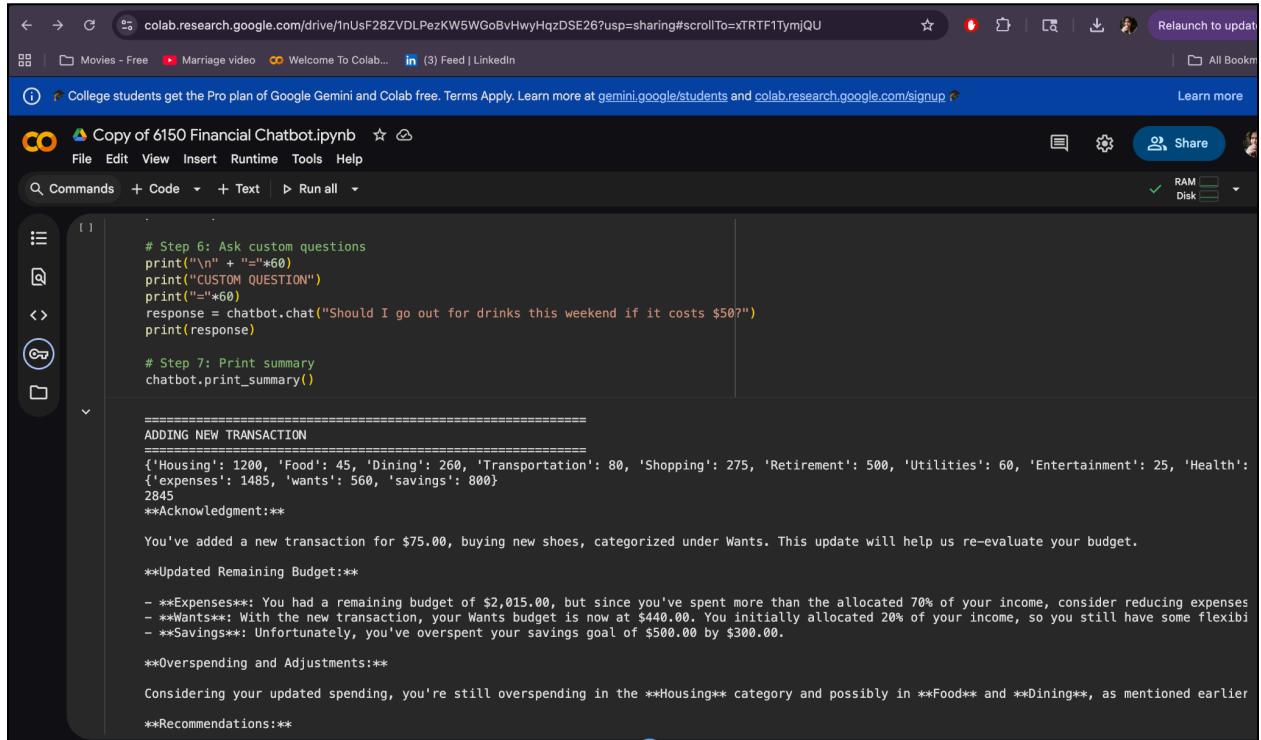
1. **Housing**: Budget: 1200, Actual: $1,485.00 (an overspending of $285.00)
2. **Food**: Budget: 45, Actual unknown, but a possible underestimation. However this could be low assuming you are dining out more than expected which will be covered later in this response,
   Actual: Not provided
3. **Dining**: Budget: 260, Actual: unknown, but may be higher than budgeted, especially considering the 'Dining' + 'Food' budget might be higher. Actual to higher 'Expenses'.

```

11. Financial statements security: The system handles and processes the uploaded financial statements in a secured way. Vital information is sanitized during the extraction phase and emphasis is laid only upon the transactional data. The Groq API key is stored securely in environment variables to make sure of the credentials getting responsibly handled.

Notebook access	Name	Value	Actions
<input checked="" type="checkbox"/>	GROQ_API_KEY	
<input type="checkbox"/>	HF_TOKEN	
<input checked="" type="checkbox"/>	KAGGLE_KEY	
<input checked="" type="checkbox"/>	KAGGLE_USERNAME	
<input type="checkbox"/>	OPENAI_API_KEY	

12. Recalculation in real-time: When the user continues including transactions or asking queries, the agentic workflow updates the internal state and recomputes all the analysis results in almost real time, helping in maintaining a continuous and dynamic management of the financial experience.



```

# Step 6: Ask custom questions
print("\n" + "="*60)
print("CUSTOM QUESTION")
print("="*60)
response = chatbot.chat("Should I go out for drinks this weekend if it costs $50?")
print(response)

# Step 7: Print summary
chatbot.print_summary()

=====
ADDING NEW TRANSACTION
=====
{'Housing': 1200, 'Food': 45, 'Dining': 260, 'Transportation': 80, 'Shopping': 275, 'Retirement': 500, 'Utilities': 60, 'Entertainment': 25, 'Health': 2845
expenses: 1485, 'wants': 560, 'savings': 800}
**Acknowledgment:**

You've added a new transaction for $75.00, buying new shoes, categorized under Wants. This update will help us re-evaluate your budget.

**Updated Remaining Budget:**

- **Expenses:** You had a remaining budget of $2,015.00, but since you've spent more than the allocated 70% of your income, consider reducing expenses
- **Wants:** With the new transaction, your Wants budget is now at $440.00. You initially allocated 20% of your income, so you still have some flexibility
- **Savings:** Unfortunately, you've overspent your savings goal of $500.00 by $300.00.

**Overspending and Adjustments:**

Considering your updated spending, you're still overspending in the **Housing** category and possibly in **Food** and **Dining**, as mentioned earlier

**Recommendations:**

```

Additional Walkthroughs:

Scenario A:

The user inputs: "Which categories should I address first, and where am I currently overspending?"

Chatbot's response: "Looking at your current budget, you're overspending in a few key areas,"

- Total Costs (Needs): Although you intended to keep needs at 70% of income, spending on things like housing, food, and dining is exceeding your ideal level.
- Housing: A significant portion of your monthly budget and a primary contributor to your overall spending is your \$1,200 rent.
- Food & Dining: Although you have set aside \$45 for food and \$260 for dining, your actual spending is probably higher and accounts for a large portion of your \$1,485 in expenses.

The screenshot shows a Google Colab notebook titled "Copy of 6150 Financial Chatbot.ipynb". The code in the cell prints an initial budget analysis and then calls a function to get detailed analysis. The output shows a breakdown of expenses like Housing, Food, Dining, Transportation, Shopping, Retirement, Utilities, Entertainment, and Health. It highlights overspending in Expenses and underutilization in Wants and Savings. It also lists categories where spending is higher than budgeted.

```

print("\n" + "*"*80)
print("INITIAL BUDGET ANALYSIS")
print("*"*80 + "\n")

analysis = chatbot.get_initial_analysis()
print(analysis)

...
=====
INITIAL BUDGET ANALYSIS
=====

{'Housing': 1200, 'Food': 45, 'Dining': 260, 'Transportation': 80, 'Shopping': 200, 'Retirement': 500, 'Utilities': 60, 'Entertainment': 25, 'Health': 2770
Let's dive into your budget analysis.

**Overview of Spending vs Budget:**

You're currently overspending in the Expenses category by a significant margin ($1,485.00). However, you're underutilizing the Wants and Savings categories.

You've allocated 70% of your income for Expenses, 20% for Wants, and 10% for Savings. To achieve these goals, you need to make adjustments in your spending.

**Overspending Categories:**

Based on your Category Breakdown, the top three areas where you're overspending are:

1. **Housing**: Budget: 1200, Actual: $1,485.00 (an overspending of $285.00)
2. **Food**: Budget: 45, Actual unknown, but a possible underestimation. However this could be low assuming you are dining out more than expected which will be covered later in this response,
   Actual: Not provided
3. **Dining**: Budget: 260, Actual: unknown, but may be higher than budgeted, especially considering the 'Dining' + 'Food' budget might be higher. Actual to higher 'Expenses'.

Also, the actual cost of food could contribute more to overall 'Expenses' of $1450.

```

Scenario B:

User enters:

"Is it okay if I keep spending on things like shopping and entertainment this month?"

Chatbot's response:

"Right now, your Wants budget looks like this:

- Wants Budget: \$1,000
- Wants Spent so far: \$680
- Wants Remaining: \$320

You've made recent purchases in Shopping (shoes, \$75) and Entertainment (Italian restaurant, \$120). With only \$320 left:

- You still can spend on Wants, but you should be very selective.
- One or two more medium-sized purchases could use up your entire Wants budget.

These changes can help bring your spending closer to your budget goals.”

```
Copy of 6150 Financial Chatbot.ipynb
=====
... ADDING NEW TRANSACTION
=====
{'Housing': 1200, 'Food': 45, 'Dining': 260, 'Transportation': 80, 'Shopping': 275, 'Retirement': 500, 'Utilities': 60, 'Entertainment': 25, 'Health': 2845
**Acknowledgment:**

You've added a new transaction for $75.00, buying new shoes, categorized under Wants. This update will help us re-evaluate your budget.

**Updated Remaining Budget:**

- **Expenses:** You had a remaining budget of $2,015.00, but since you've spent more than the allocated 70% of your income, consider reducing expenses.
- **Wants:** With the new transaction, your Wants budget is now at $440.00. You initially allocated 20% of your income, so you still have some flexibility.
- **Savings:** Unfortunately, you've overspent your savings goal of $500.00 by $300.00.

**Overspending and Adjustments:**

Considering your updated spending, you're still overspending in the **Housing** category and possibly in **Food** and **Dining**, as mentioned earlier.

**Recommendations:** 

1. **Re-evaluate Housing Costs:** Negotiate a rent reduction or explore more cost-effective living arrangements.
2. **Optimize Dining and Food Expenses:** Review your meal planning and dining habits. Consider cooking at home more often, using coupons, and taking advantage of grocery store deals.
3. **Review Overall Expenses:** Identify areas to reduce expenses, such as subscription services, transportation costs, or utilities.
4. **Prioritize Savings:** Since you've overspent your savings goal, try to allocate any extra funds towards saving before making non-essential purchases.

**Action Items:** 

- Review and adjust your spending in the **Housing** category.
- Optimize your dining and food expenses by reducing unnecessary costs.
- Prioritize saving by allocating any spare funds towards retirement or emergency funds.
- Consider increasing income by taking on a side job, selling unwanted items, or asking for a raise at work.

===== ADDING ANOTHER TRANSACTION (AUTO-CLASSIFIED)
=====
```

Scenario C:

User enters:

"Given my current budget, what 3–5 concrete steps should I take next to get on track?"

Chatbot's response:

Based on your present budget, the following are specific next steps:

The largest fixed cost is housing:

- Look into ways to cut utilities or rent over the long term (roommate, smaller space, negotiating lease).
- Your budget would have a lot more room if you could cut this amount by even \$50 to \$100 a month.

Dining & Food:

- Plan to cook at home for the remainder of the month and limit additional dining out to a small, set amount (for instance, \$40–\$60).
- To prevent impulsive food purchases, use grocery lists and meal planning.

Desires (Amusement & Shopping):

- You've already spent \$680; don't spend more than \$100 to \$150 on Wants.
- Put off buying unnecessary items and choose free or inexpensive activities instead.

In this scenario, going out for drinks would account for about 15-20% (\$50) of your remaining Wants budget. While it might be tempting to spend some more, consider the long-term impact on your financial health.

... **Alternative Options:**

1. **Wait and See:** You could consider waiting until the next month to indulge in a night out, making sure you have a larger buffer for future discrete expenses.
2. **Plan a More Budget-Friendly Night:** If you still want to spend time with friends, consider planning a free or low-cost activity, like going for a walk or a movie.
3. **Reassess Priorities:** Take a closer look at your expenses and priorities. Ask yourself if there are any other areas where you could adjust spending without sacrificing essential needs.

Final Decision:

If you feel that spending \$50 on drinks is essential for your mental well-being or an exceptional experience, go for it! However, if you're on a tight budget, consider prioritizing other expenses like housing, food, and transportation.

Total Income: \$5,000.00
Total Spent: \$2,965.00
Remaining: \$2,035.00

Category	Budget	Spent	Remaining
Expenses	\$ 3,500.00	\$ 1,485.00	\$ 2,015.00 ✓
Wants	\$ 1,000.00	\$ 680.00	\$ 320.00 ✓
Savings	\$ 500.00	\$ 800.00	\$ -300.00 △

Total Transactions: 14

Start coding or generate with AI.

Variables Terminal System Settings Python 3 6:23PM