*High Level Design*

**Components**

**Data Loader:** Reads user & peer data from CSV files.

**Preprocessor:** Cleans data, categorizes transactions (rule-based or simple ML).

**Peer Comparator:** Uses K-Means to find similar users & compares spending.

**Market Trend Fetcher:** Gets inflation/trend data from free APIs (e.g., FRED, World Bank).

Analysis Engine Combines peer comparison + market trends to generate insights.

**Data Flow**

1. **Input** → User uploads CSV (transactions.csv).
2. **Preprocessing** → Categorize transactions (e.g., "Food", "Transport").
3. **Peer Comparison** → Cluster similar users & compute average spending.
4. **Market Data** → Fetch inflation rate via API.
5. **Adjust & Analyze** → Adjust peer benchmarks for inflation.
6. **Output** → Returns JSON with spending differences & suggestions.

**Tools & Libraries**

Data Loading: pandas (read CSV)

Categorization: Rule-based (keyword matching) or scikit-learn (NLP)

Peer Clustering: scikit-learn (K-Means)

Market Data API: requests (fetch JSON from FRED/World Bank)

Trend Analysis: statsmodels (time-series) or manual math

## *Low Level Design* **1. Class Diagram**

classDiagram

class Transaction {

+date: str

+amount: float

+description: str

+category: str

}

class UserProfile {

+age: int

+income: float

+transactions: List[Transaction]

+get\_spending\_by\_category() dict

}

class PeerAnalyzer {

+peer\_data: List[UserProfile]

+kmeans\_model: KMeans

+find\_similar\_users(user: UserProfile) List[UserProfile]

+calculate\_peer\_benchmark(users: List[UserProfile]) dict

}

class MarketAnalyzer {

+fetch\_inflation\_rate() float

+adjust\_for\_inflation(spending: dict, inflation: float) dict

}

class SpendingAnalyzer {

+user: UserProfile

+peer\_analyzer: PeerAnalyzer

+market\_analyzer: MarketAnalyzer

+analyze() dict

}

UserProfile "1" \*-- "0..\*" Transaction

SpendingAnalyzer "1" --> "1" UserProfile

SpendingAnalyzer "1" --> "1" PeerAnalyzer

SpendingAnalyzer "1" --> "1" MarketAnalyzer

PeerAnalyzer "1" --> "0..\*" UserProfile

**2. Detailed Component Specifications**

**A. Transaction Class**

class Transaction:

def \_\_init\_\_(self, date: str, amount: float, description: str):

self.date = date

self.amount = amount

self.description = description

self.category = self.\_categorize()

def \_categorize(self) -> str:

"""Rule-based categorization"""

categories = {

"Food": ["mcdonalds", "starbucks", "groceries"],

"Transport": ["uber", "gas", "metro"]

}

for category, keywords in categories.items():

if any(kw in self.description.lower() for kw in keywords):

return category

return "Other"

**B. UserProfile Class**

class UserProfile:

def \_\_init\_\_(self, age: int, income: float, transactions: List[Transaction]):

self.age = age

self.income = income

self.transactions = transactions

def get\_spending\_by\_category(self) -> dict:

spending = {}

for txn in self.transactions:

spending[txn.category] = spending.get(txn.category, 0) + txn.amount

return spending

**C. PeerAnalyzer Class**

from sklearn.cluster import KMeans

import numpy as np

class PeerAnalyzer:

def \_\_init\_\_(self, peer\_data: List[UserProfile]):

self.peer\_data = peer\_data

self.kmeans\_model = None

def \_prepare\_features(self):

"""Convert user profiles to feature matrix"""

features = []

for user in self.peer\_data:

spending = user.get\_spending\_by\_category()

features.append([

user.age,

user.income,

spending.get("Food", 0),

spending.get("Transport", 0)

])

return np.array(features)

def find\_similar\_users(self, target\_user: UserProfile) -> List[UserProfile]:

features = self.\_prepare\_features()

self.kmeans\_model = KMeans(n\_clusters=3).fit(features)

# Get target user's features

target\_features = np.array([

target\_user.age,

target\_user.income,

target\_user.get\_spending\_by\_category().get("Food", 0),

target\_user.get\_spending\_by\_category().get("Transport", 0)

]).reshape(1, -1)

cluster = self.kmeans\_model.predict(target\_features)[0]

return [user for i, user in enumerate(self.peer\_data)

if self.kmeans\_model.labels\_[i] == cluster]

**D. MarketAnalyzer Class**

import requests

class MarketAnalyzer:

@staticmethod

def fetch\_inflation\_rate() -> float:

"""Example: Fetch from World Bank API"""

response = requests.get("https://api.worldbank.org/v2/country/US/indicator/FP.CPI.TOTL.ZG")

return float(response.json()[1][0]['value'])

@staticmethod

def adjust\_for\_inflation(spending: dict, inflation\_rate: float) -> dict:

return {k: v \* (1 + inflation\_rate/100) for k, v in spending.items()}

**E. SpendingAnalyzer (Main Controller)**

class SpendingAnalyzer:

def \_\_init\_\_(self, user: UserProfile, peer\_data: List[UserProfile]):

self.user = user

self.peer\_analyzer = PeerAnalyzer(peer\_data)

self.market\_analyzer = MarketAnalyzer()

def analyze(self) -> dict:

# Step 1: Get user's spending

user\_spending = self.user.get\_spending\_by\_category()

# Step 2: Find similar peers

similar\_users = self.peer\_analyzer.find\_similar\_users(self.user)

peer\_benchmark = self.\_calculate\_peer\_benchmark(similar\_users)

# Step 3: Adjust for market trends

inflation = self.market\_analyzer.fetch\_inflation\_rate()

adjusted\_benchmark = self.market\_analyzer.adjust\_for\_inflation(peer\_benchmark, inflation)

return {

"user\_spending": user\_spending,

"peer\_benchmark": peer\_benchmark,

"inflation\_rate": inflation,

"adjusted\_benchmark": adjusted\_benchmark

}

def \_calculate\_peer\_benchmark(self, users: List[UserProfile]) -> dict:

"""Calculate average spending for peer group"""

totals = {}

counts = {}

for user in users:

spending = user.get\_spending\_by\_category()

for cat, amt in spending.items():

totals[cat] = totals.get(cat, 0) + amt

counts[cat] = counts.get(cat, 0) + 1

return {k: v/counts[k] for k, v in totals.items()}

**3. Data Structures**

1. **Transaction Data**

{

"date": "2023-01-01",

"amount": 12.50,

"description": "Starbucks coffee",

"category": "Food" # Auto-generated

}

1. **Analysis Output**

{

"user\_spending": {"Food": 400, "Transport": 150},

"peer\_benchmark": {"Food": 350, "Transport": 120},

"inflation\_rate": 6.2,

"adjusted\_benchmark": {"Food": 371.7, "Transport": 127.44}

}

**4. Key Algorithms**

1. **K-Means Clustering**
   * Groups users by age/income/spending patterns
   * Uses Euclidean distance metric
2. **Inflation Adjustment**
   * Simple percentage-based scaling
   * new\_value = old\_value × (1 + inflation\_rate/100)
3. **Spending Categorization**
   * Rule-based keyword matching
   * (Optional upgrade: NLP with TF-IDF)