Documentation for Paisa Controller Hackathon Project

Github repository

1.Problem Statement: Paisa Controller

In today's fast-paced world, managing personal finances effectively is a challenge. People struggle with tracking expenses, maintaining budgets, and achieving financial goals due to a lack of intuitive tools, real-time insights, and automation. Many existing finance apps are either too complex, lack AI-driven automation, or do not provide seamless cross-platform experiences.

Key Features Include:

- **Expense & Income Tracking:**
  - Fast data entry with auto-fill suggestions
  - Smart categorization
  - Advanced search and filtering options

- **Reports & Financial Insights:**
  - Visual data representations (charts, graphs, trend analysis)
  - Comprehensive monthly and yearly summaries.

- **Security & Customization:**
  - Multi-layer authentication (PIN, biometrics, JWT Token)
  - Multi-currency support, dark mode, and customizable themes
  - Desktop web dashboard for expanded management

The goal is to empower users to achieve financial stability with minimal effort, by harnessing AI to deliver personalized recommendations and automated financial oversight.

2.Workflow

The development process for Paisa Controller is structured into several critical phases to ensure a robust, scalable, and user-friendly solution:

2.1.Planning & Requirements Gathering

- **Team Meeting:**
  Identified pain points and must-have features
  .
- **Market & Competitor Analysis:**
  Evaluated existing finance apps to determine unique selling propositions (USPs) and areas for innovation.

- **Project Specifications:**
  Documented detailed requirements, use cases, and deliverables based on the hackathon brief.

2.2.Design & Prototyping

- **Wireframing & Mockups:**
  Created initial wireframes for the mobile/web interface focusing on simplicity and intuitive navigation.

- **UI/UX Design:**
  Developed high-fidelity designs using tools like Figma/Sketch, ensuring a clean, modern look that promotes ease of use.

- **Prototype Development:**
  Started Building a interactive prototype to gather early problems and iterate on design elements.

2.3.Frontend Development

- **Template Creation:**
  Developed a responsive HTML and EJS templates (or equivalent) for consistent page rendering across devices.

- **Styling & Interactivity:**
  Utilized a CSS3 and JavaScript to implement smooth navigation, real-time data visualization, and interactive UI components.

- **Component Reusability:**
  Then we Developed a reusable UI modules to streamline for future updates.

2.4.Backend Development

- **Server & API Setup:**
  Started building a backend server (e.g., using Node.js and Express.js) to handle business logic, routing, and secure data transactions.

- **Database & Cloud Integration:**
  We Designed a scalable database schema and implemented cloud synchronization with services like Firebase or AWS. Incorporate third-party integrations.

2.4.Testing & Quality Assurance

- **Unit and Integration Testing:**
  Perform thorough testing of individual components and their interactions to ensure functional accuracy.

- **User Acceptance Testing (UAT):**
  Conduct beta tests with a select group of users to gather actionable feedback and identify UX issues.

- **Performance & Security Testing:**
  Validate the application's scalability, responsiveness, and security measures (including authentication and data protection protocols).

2.5.Deployment & Maintenance

- **CI/CD Pipeline:**
  Set up continuous integration and deployment pipelines to automate testing and streamline release cycles.

- **Monitoring & Analytics:**
  Implement monitoring tools to track application performance, user interactions, and potential issues in real time.

- **Iterative Enhancements:**
  Plan for regular updates and feature enhancements based on ongoing user feedback and market trends.

3.Tech Stack Details

3.1.Frontend:

- **HTML5 & CSS3:**
  For structured, semantic markup and responsive design.
- **JavaScript:**
  To create dynamic interactions and enhance user experience.
- **EJS:**
  Depending on the project scope, use a templating engine (like EJS) for efficient UI rendering.

3.2.Backend:

- **Node.js:**
  A scalable JavaScript runtime for server-side development.
- **Express.js:**
  A lightweight web framework for API development and routing.
- **AI/ML Integration:**
  Python-based AI models or JavaScript libraries to power smart categorization and financial insights.

3.3.Database & Cloud Services:

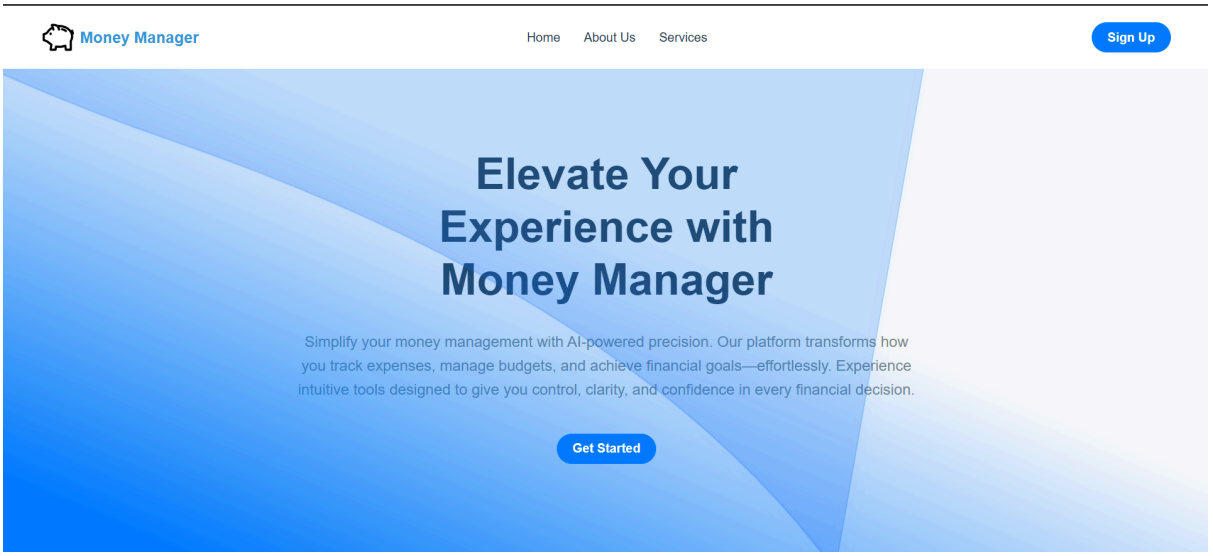
- **Firebase/AWS:**
  For real-time data synchronization, authentication, and cloud storage.
- **SQL/NoSQL Database:**
  Depending on data complexity, use a relational (MySQL/PostgreSQL) or NoSQL (MongoDB) database.

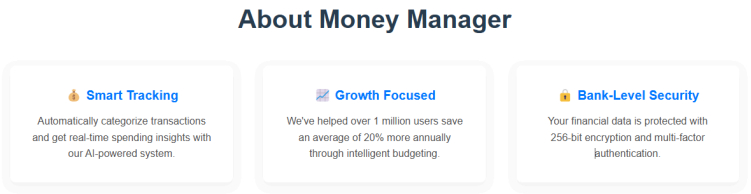
3.4.Additional Tools & Integrations:

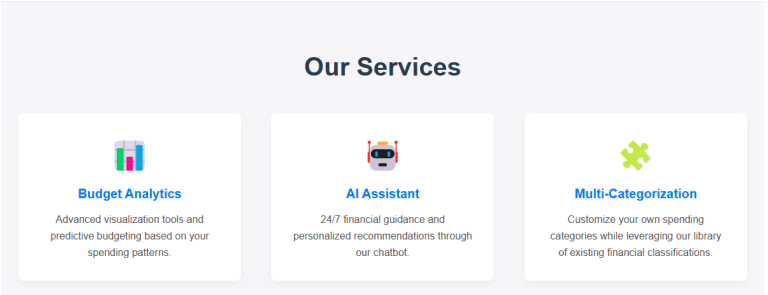
- **Cloudinary:**
  For managing and hosting image assets (e.g., scanned receipts).
- **Version Control (Git):**
  To manage the codebase and facilitate collaborative development.
- **CI/CD Tools (GitHub Actions, Jenkins):**
  For automating tests and deployments.
- **Third-Party Integrations:**
  Integration with services like Google Drive/Dropbox for data backup and export functionality.

4.Results:

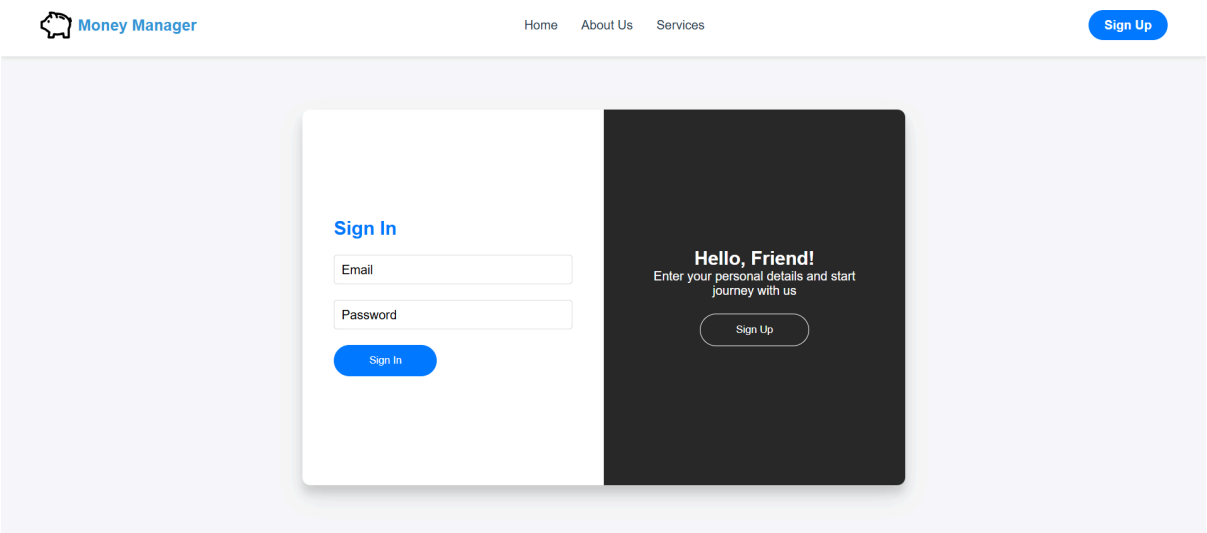
4.1.Landing Page:

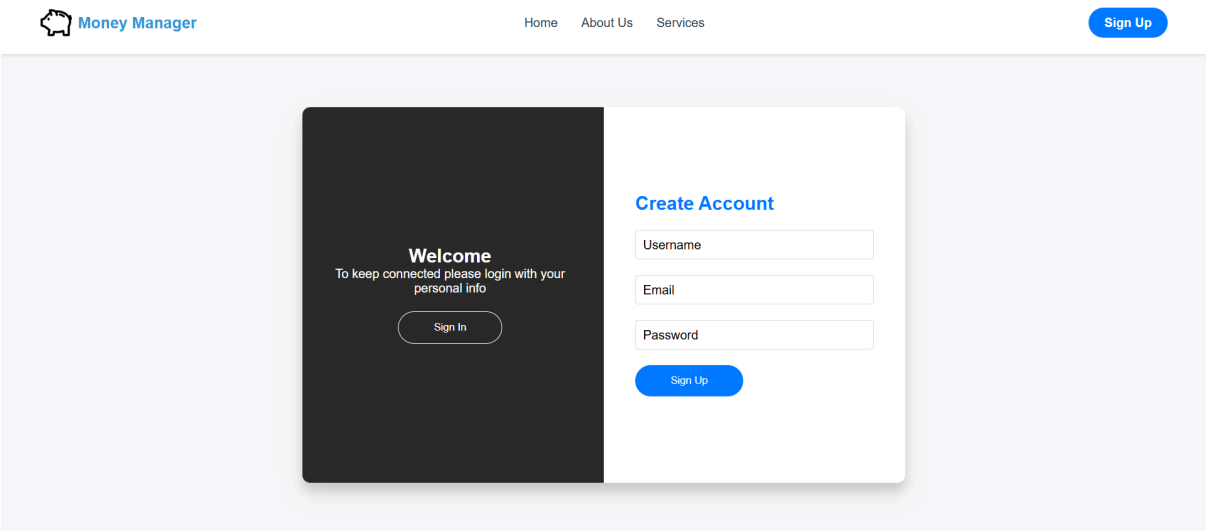




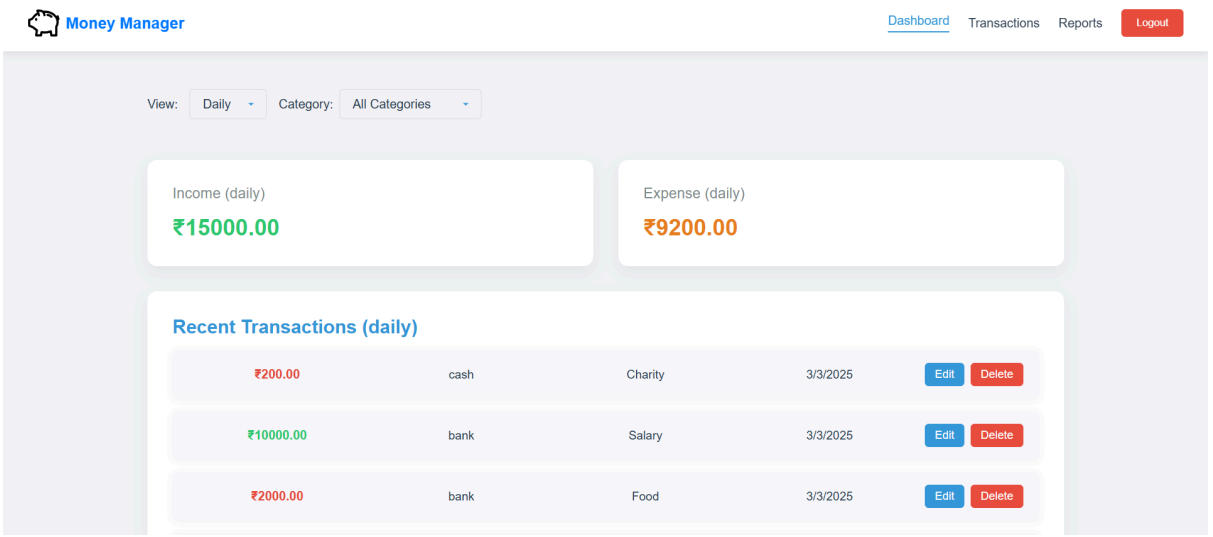


4.2.Authentication Page:

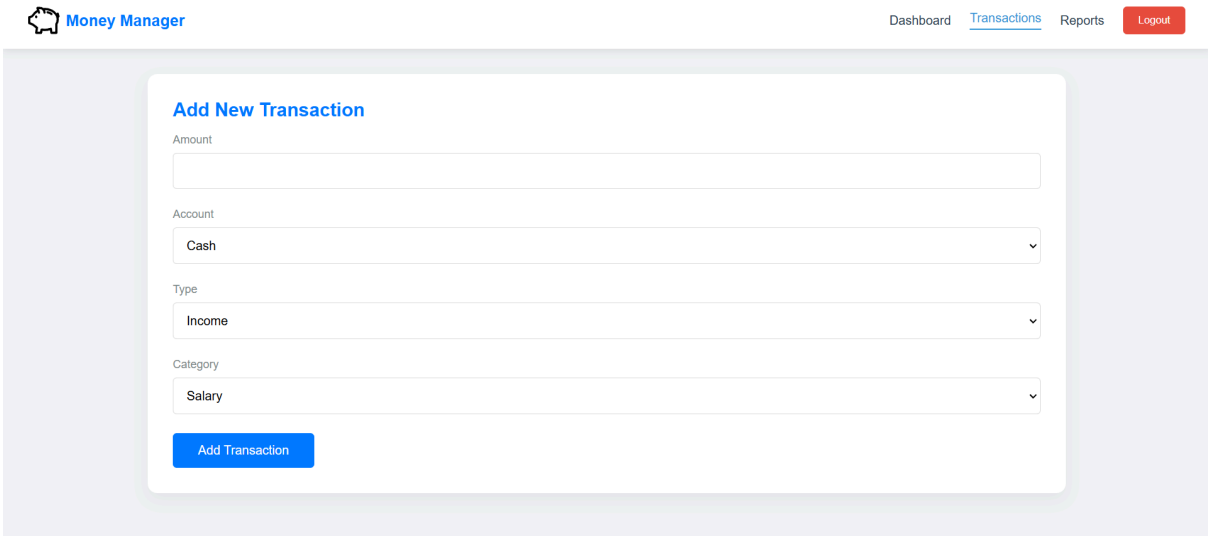




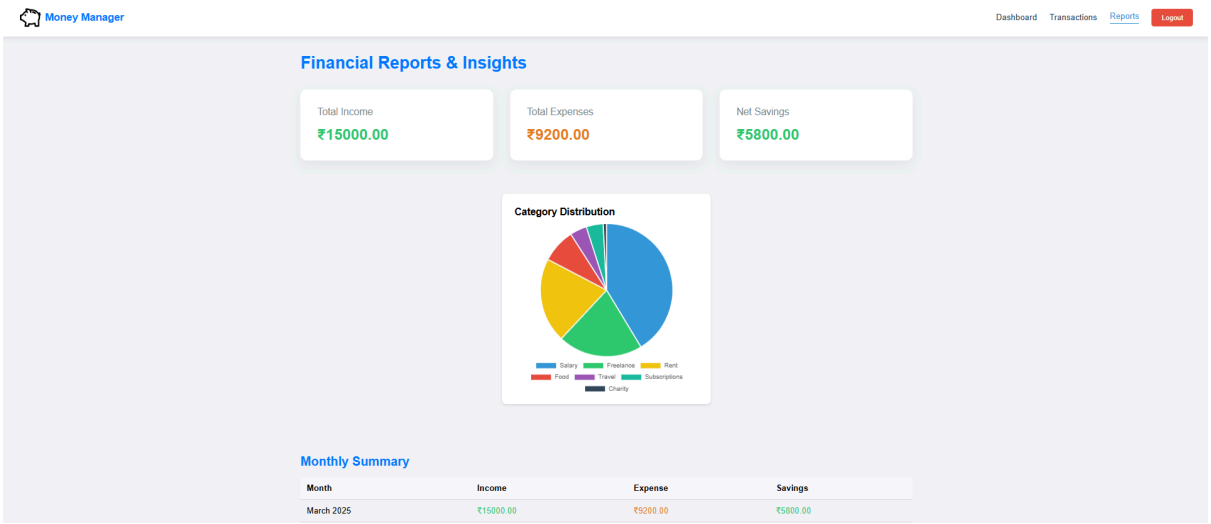
4.3.Dashboard Page:



4.4.Transaction Page



4.5.Report Page



4.6.AI Model Training Code

For AI-powered smart categorization of expenses :

Dataset used : expenses_income_ from kaggle

```
# Step 0: Import Required Libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
import pickle
import kagglehub

# Download the dataset from Kaggle using kagglehub (ensure kagglehub is configured correctly)
path = kagglehub.dataset_download("jg7fujhfydhgc/expenses-2024")
print("Path to dataset files:", path)

# Step 1: Load the Dataset
# Make sure you're using the correct CSV file; here we use 'expenses_income_summary.csv'
df = pd.read_csv('expenses_income_summary.csv')
print("First few rows of the dataset:")
print(df.head())

# Standardize column names to lower-case
df.columns = df.columns.str.lower()
print("Columns in DF:", df.columns)

# Convert 'title' and 'description' to string type, then create a combined text column
df['title'] = df['title'].fillna('').astype(str)
df['description'] = df['description'].fillna('').astype(str)
df['combined_text'] = df['title'] + ' ' + df['description']

# Drop rows missing 'category' (assume 'category' is essential)
df.dropna(subset=['category'], inplace=True)
print("DataFrame shape after dropna on category:", df.shape)

# Step 2: Prepare Input and Output Data
# Use 'combined_text' as the feature and 'category' as the label.
texts = df['combined_text'].values
categories = df['category'].values

# Encode category labels into integers
label_encoder = LabelEncoder()
```

```
# Encode category labels into integers
label_encoder = LabelEncoder()
categories_encoded = label_encoder.fit_transform(categories)
num_classes = len(label_encoder.classes_)
print("Number of categories:", num_classes)

# Step 3: Split the Data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    texts, categories_encoded, test_size=0.2, random_state=42)
print("Training samples:", len(X_train), "Test samples:", len(X_test))

# Step 4: Vectorize the Transaction Descriptions Using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')
X_train_vect = vectorizer.fit_transform(X_train).toarray()
X_test_vect = vectorizer.transform(X_test).toarray()

# Convert labels to one-hot encoded vectors
y_train_cat = to_categorical(y_train, num_classes)
y_test_cat = to_categorical(y_test, num_classes)

# Step 5: Build the Neural Network Model Using Keras
input_dim = X_train_vect.shape[1]
model = Sequential([
    Dense(512, activation='relu', input_shape=(input_dim,)),
    Dense(256, activation='relu'),
    Dense(num_classes, activation='softmax')
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()

# Step 6: Train the Model
history = model.fit(X_train_vect, y_train_cat,
                    epochs=10,              # Increase epochs if needed
                    batch_size=32,
                    validation_data=(X_test_vect, y_test_cat))

# Step 7: Evaluate the Model
loss, accuracy = model.evaluate(X_test_vect, y_test_cat)
print("Test Accuracy:", accuracy)

# Step 8: Save the Trained Model and Preprocessing Tools
model.save('expense_classifier.h5')  # Save the Keras model

with open('vectorizer.pkl', 'wb') as f:
```

For AI-based cash flow analysis with spending recommendations :

Dataset used : Personal_Finance_Dataset From Kaggle

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import StandardScaler
import pickle
import matplotlib.pyplot as plt
import seaborn as sns


# Loaded the dataset with columns: Date, Transaction Description, Category, Amount, Type
data = pd.read_csv('Personal_Finance_Dataset.csv')

# Converted the Date column to datetime and create a YearMonth column for aggregation
data['Date'] = pd.to_datetime(data['Date'])
data['YearMonth'] = data['Date'].dt.to_period('M').astype(str)

print("Dataset Shape:", data.shape)
print("Columns:", data.columns.tolist())
print("\nFirst 5 rows:")
print(data.head())

# --------------------------------
# 2. Aggregated Data by Month
# --------------------------------
# For each month, we calculated:
#   - total_income: Sum of Amount for rows where Type == 'Income'
#   - total_expense: Sum of Amount for rows where Type == 'Expense'
#   - savings: total_income - total_expense
#   - savings_percentage: (savings / total_income) * 100 (if income > 0, else 0)
#   - recommendation: based on savings_percentage thresholds

def aggregate_month(group):
    total_income = group.loc[group['Type'] == 'Income', 'Amount'].sum()
    total_expense = group.loc[group['Type'] == 'Expense', 'Amount'].sum()
    savings = total_income - total_expense
    savings_percentage = (savings / total_income * 100) if total_income > 0 else 0
    # Define recommendation based on savings percentage thresholds
    if savings_percentage < 10:
```

```python
def aggregate_month(group):
    total_income = group.loc[group['Type'] == 'Income', 'Amount'].sum()
    total_expense = group.loc[group['Type'] == 'Expense', 'Amount'].sum()
    savings = total_income - total_expense
    savings_percentage = (savings / total_income * 100) if total_income > 0 else 0
    # Define recommendation based on savings percentage thresholds
    if savings_percentage < 10:
        recommendation = "Increase Savings"
    elif savings_percentage <= 20:
        recommendation = "Maintain Spending"
    else:
        recommendation = "Invest More"
    return pd.Series({
        'total_income': total_income,
        'total_expense': total_expense,
        'savings': savings,
        'savings_percentage': savings_percentage,
        'recommendation': recommendation
    })

# Grouped by YearMonth and aggregate
monthly_data = data.groupby('YearMonth').apply(aggregate_month).reset_index()

print("\nMonthly Aggregated Data:")
print(monthly_data.head())

# Optional: Save monthly data to CSV for inspection
monthly_data.to_csv('monthly_cash_flow.csv', index=False)

# ------------------------------
# 3. Exploratory Data Analysis (Optional)
# ------------------------------
# Plot correlation heatmap for numerical features in the monthly data
numerical_cols = ['total_income', 'total_expense', 'savings_percentage']
plt.figure(figsize=(8, 6))
sns.heatmap(monthly_data[numerical_cols].corr(), annot=True, cmap="coolwarm")
plt.title("Monthly Data Correlation Heatmap")
plt.savefig("monthly_correlation_heatmap.png")
plt.close()
print("Correlation heatmap saved as 'monthly_correlation_heatmap.png'.")

# ------------------------------
```

```
# ------------------------------
# 4. Prepared Data for Modeling
# ------------------------------
# We will use total_income, total_expense, and savings_percentage as features,
# and the generated recommendation as the target.
features = ['total_income', 'total_expense', 'savings_percentage']
target = 'recommendation'

# Dropped any rows with missing values (if any)
monthly_data.dropna(inplace=True)

X = monthly_data[features]
y = monthly_data[target]

# Normalized numerical features using StandardScaler
scaler = StandardScaler()
X[features] = scaler.fit_transform(X[features])

# ------------------------------
# 5. Split, Train, and Evaluate the Model
# ------------------------------
# Split the aggregated data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predicted on the test set and evaluate
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("\nTest Accuracy:", acc)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# ------------------------------
# 6. Saved the Trained Model for Backend Integration
# ------------------------------
# Saved the model along with the scaler and feature list for later use in your backend
model_data = {
    'model': clf,
```

Future Scope

- **Advanced Machine Learning Models:**
    - Explore deep learning models (e.g., LSTM, GRU) for more accurate time-series forecasting and anomaly detection.
    - Integrate reinforcement learning for adaptive spending recommendations based on user behavior over time.
- **Real-Time Data Integration:**
    - Implement real-time data ingestion from bank APIs or financial aggregators to enable live cash flow monitoring.
    - Develop a dashboard for live tracking and visualization of financial data trends.
- **Enhanced User Features:**
    - Add features for automated expense categorization and budget tracking.
    - Incorporate personalized financial advice based on user spending habits and future predictions.
    - Enable user authentication and account management to support multi-user scenarios and data security.
- **Scalability and Performance Improvements:**
    - Consider containerization with Docker and orchestration with Kubernetes for better scalability in production environments.
    - Optimize the API for asynchronous processing to handle high volumes of requests efficiently.
- **Expanded Deployment Options:**
    - Investigate serverless architectures (such as AWS Lambda, Google Cloud Functions) for cost-effective, scalable deployments.
    - Enhance CI/CD pipelines to automate testing, deployment, and monitoring of the application.
- **Integration with Third-Party Services:**
    - Connect with other financial tools and platforms (e.g., budgeting apps, investment platforms) to provide a more comprehensive financial management solution.
    - Add support for multiple currencies and regional financial data for global use.
- **Security and Compliance:**
    - Strengthen security measures and ensure compliance with financial regulations (e.g., PCI-DSS, GDPR) as you handle sensitive user data.
    - Implement advanced monitoring and alerting mechanisms to quickly detect and respond to any security incidents.
- **Comprehensive Documentation and Community Support:**
    - Expand the documentation to include detailed API reference guides, integration examples, and troubleshooting FAQs.
    - Encourage community contributions and feedback to continuously evolve the project based on real-world usage and emerging trends.