BudgetBuddy: Visual Expense Tracker with Smart Spending Recommendations

Group Members: Linlan Cai, Anais Lacreuse, Brunda Sulikunte Gangadhara Reddy

Course Name: CS 661 - Python Programming - 40600

Instructor: Brian Harley

Pace University

Date of Submission: Aug 11, 2024

Summary:

Abstract

The BudgetBuddy project will be focused on making a data visualization dashboard for personal expenses and AI-driven suggestions for expenditures. These insights should be realized by the user with respect to their spending patterns and categorization of expenditure, hence helping them make financial decisions regarding budgeting. The following shall be realized with a Python programming library giving access to the budgeting data set. The dashboard personalizes its interface with the help of some machine learning techniques, offering personalized spending predictions and tangible advice that can assist users in extracting the best from their finances. Regarding this fact, the primary intention of the project lies in developing a rich yet user-friendly tool for visualizing past expenditures and predicting trends in spending for future purposes. This project meets the ever-rising demand for convenient, accessible, and easy-to-use solutions for the effective management of personal finance. In so doing, BudgetBuddy will give users AI-driven recommendations and insights supported with data for better habits and financial wellness.

I. Introduction

Background Information

Effective personal finance management is a crucial aspect of modern life, enabling individuals to achieve financial stability and long-term goals. Despite the importance of tracking expenses and budgeting, many people need help with maintaining a clear understanding of their spending habits. Traditional methods of expense tracking, such as manual entry in spreadsheets, can be time-consuming and prone to errors. In contrast, digital tools and dashboards can provide a more efficient and user-friendly approach to managing personal finances.

Importance of the Project

The SmartSpend project is designed to meet the growing demand for advanced financial management solutions. In a world where personal finance has become increasingly intricate, the need for a tool that offers clear, actionable insights is paramount. SmartSpend addresses this need by providing a platform that not only visualizes spending patterns but also leverages artificial intelligence to offer tailored financial recommendations. With its focus on data visualization and machine learning, SmartSpend aims to make personal finance management more accessible and effective. By transforming raw financial data into actionable advice, it empowers users to make informed decisions and optimize their financial health.

Objectives and Goals

The primary objectives of the SmartSpend project are:

Track and Visualize Expenses: Enable users to upload their expense data and visualize spending patterns through charts and graphs.

Interactive Dashboard: Develop an interactive and user-friendly dashboard that allows users to explore their financial data in detail, with features such as filtering and zooming.

AI-Based Spending Suggestions: Implement a linear regression model to predict future spending and provide personalized recommendations to help users optimize their finances.

Project Specifications

Scope

Included:

- Expense Tracking: The project will include features for users to upload and manage personal expense data.
- Data Visualization: To display spending patterns and trends, various types of visualizations, such as bar charts, pie charts, and line charts, will be created.
- Interactive Dashboard: An interactive dashboard will be developed using Dash by Plotly to enable users to explore their financial data with features like filtering and zooming.
- AI-Based Spending Suggestions: Implement a linear regression model to predict future spending and provide personalized recommendations to help users optimize their finances.

Not Included:

- Detailed Financial Planning: The project will not cover detailed financial planning aspects such as savings plans, investment advice, or tax calculations.
- Real-Time Transaction Syncing: The project will not include real-time syncing with bank accounts or financial institutions.
- Complex AI Models: We will not include advanced machine learning models beyond a basic linear regression for spending predictions.

Objectives

- Track and Visualize Expenses:

  Enable users to upload and manage their expense data.

  Provide visualizations to display monthly spending, spending by category, and overall spending trends.

- Develop an Interactive Dashboard:

  Create a user-friendly interface for users to interact with their financial data.

Implement features like filtering and zooming to enhance user interaction.

- Provide AI-Based Spending Suggestions: Implement a linear regression model to analyze historical spending data.

Provide monthly spending predictions and actionable recommendations to help users manage their finances better.

Deliverables

The BudgetBuddy project will produce the following tangible outcomes:

- Software Application: A fully functional personal finance management dashboard that allows users to track and visualize their expenses, interact with their financial data, and receive AI-driven spending suggestions.

- User Documentation: Comprehensive user guides and documentation to help users understand how to use the dashboard and interpret the data visualizations and recommendations.

- Technical Documentation: Detailed documentation for developers, including the project's architecture, data flow, and implementation details of the AI models and visualizations.

- Data Analysis Report: A report summarizing the key findings from the user data, including common spending patterns and the effectiveness of the AI recommendations.

II. Requirements

Functional Requirements

- Expense Data Management:

Users can upload their personal expense data in various formats (CSV, Excel). The system categorizes expenses automatically based on predefined categories.

- Data Visualization:

Visualizations include bar charts, pie charts, and line charts to display monthly spending, spending by category, and overall spending trends.

- Interactive Dashboard:

The dashboard offers features like filtering by date or category, zooming in on specific data points, and exporting visualizations.

- AI-Based Spending Suggestions:

The system uses a linear regression model to predict future spending and provides personalized recommendations to help users optimize their finances.

- User Authentication: Our website uses firebase authentication for user sign in. The available authentication method is the email/password authentication provided by firebase.
- Data processing: Since our system relies on analysis of data, it is imperative that it process and analyze CSV files to generate budget reports.
- Data Visualization: The system should include integration of modules which facilitate in visualizing data such as expenditure on particular items etc.

Non-Functional Requirements

These define how the system should perform. They describe the quality attributes, performance, and constraints of the system. For example:

- Performance: The system should process CSV files within 2 seconds.
- Scalability: The system should handle up to 10,000 concurrent users.
- Security: The system should encrypt user data and ensure secure communication.
- Usability: The user interface should be intuitive and easy to navigate, with clear instructions and help features.

- Reliability: The system should be robust, with minimal downtime and reliable data processing.

Technical Requirements

- Hardware:

A standard computer with internet access for the users. Server infrastructure to host the application and process the data.

- Software:

Python for backend development. Dash by Plotly is used to create the interactive dashboard.

Machine learning libraries such as scikit-learn for implementing the linear regression model.

Database management system (e.g., PostgreSQL) for storing user data.

- Development Tools:

Integrated Development Environment (IDE) such as PyCharm or VSCode. Version control system (e.g., Git) for code management. The development of BudgetBuddy will follow an iterative and incremental approach, with the following key phases:

- Planning: Define project scope, objectives, and deliverables. Identify technical and functional requirements.

- Design: Create wireframes and mockups of the dashboard. Design the architecture of the system.

- Implementation: Develop the backend and frontend components. Implement data processing, visualization, and AI-based suggestion features.

- Testing: Conduct unit testing, integration testing, and user acceptance testing to ensure the system meets the requirements.

- Deployment: Deploy the application to a production environment. Provide user and technical documentation.

- Maintenance: Monitor the system, address any issues, and implement improvements based on user feedback.

III.Methodology

Approach and Methods Used

- Modular Programming Method: Distinguishing Issues: The project was broken up into separate modules, each in charge of handling a certain function. These comprised web interface, model prediction, and data processing modules.
- Reusability: To increase code efficiency and maintainability, functions and classes were created to be reusable across various application components. For instance, the dcc.Graph components can be used to make a variety of graphs and charts. Usage: A variety of visualizations, including monthly expenses, a budget summary, and spending categories, were displayed using the same dcc.Graph component. This made it possible for the application to have a unified appearance and feel.
- Integration of Flask and Dash: Flask was utilized as the backend to manage data processing and routing, while Dash was utilized to create interactive visualizations. The user experience was flawless because of this connection.
- Processing and Handling of Data: The budget and personal transactions.csv data were cleaned. To address missing values, improper formatting, and inconsistencies, the csv file was cleaned.
- Feature Engineering: To enhance the predictive model's performance, pertinent characteristics were extracted and developed.
- Model Training: To forecast future expenses, a machine learning model was trained on transaction data from the past.
- Designing User Interfaces: The web interface was made with responsive design in mind, guaranteeing a consistent user experience on a variety of devices.
- Card View Layout: For a contemporary appearance, the primary material was encircled by a card view with a border radius and shadow. To improve the visual appeal, a large yellow title was added.
- Interactive Elements: To enable data entry and section navigation, the program was designed with forms and buttons.

Performance Metrics

Performance metrics are used to measure how well a system performs under various conditions. These metrics are often part of non-functional requirements but can be crucial

for ensuring that functional requirements are met efficiently. Here are some common performance metrics:

1. **Response Time**: The time taken for the system to respond to a user action.
2. **Throughput**: The number of transactions the system can handle in a given period.
3. **Scalability**: The system's ability to handle increased load without performance degradation.
4. **Resource Utilization**: The amount of system resources (CPU, memory, etc.) used during operation.
5. **Latency**: The delay between a request and the start of a response.

Usability Metrics

Usability metrics help us quantify how user-friendly and effective our system is, making them essential for ensuring that the system meets user needs and provides a satisfactory experience. This should include:

1. **Effectiveness**: The accuracy and completeness with which users achieve their goals.
2. **Efficiency**: The resources utilized in relation to the accuracy and completeness of goals achieved.
3. **Satisfaction**: Users' comfort and positive attitudes towards using the system.
4. **Error Rate**: The frequency of errors made by users while interacting with the system.
5. **Learnability**: How easy it is for new users to accomplish tasks the first time they encounter the system.

Development Process

1. **Project Setup**: We Initialized our project first by setting up a Flask and Dash applications followed by setting up the necessary directories and files.
2. **Authentication**: Integrating Firebase for user authentication. We first created our project in firebase, got the necessary API keys and other configuration items. We then configured firebase in our project.
3. **Data Handling:** We used Pandas to read and process CSV files containing budget and personal transactions data for later processing.
4. **Visualization**: Utilized Dash to create interactive visualizations for the budget and personal transactions data.

5. **Machine learning models**: We set up our machine learning model(s) for our future expenditure prediction feature.

Key Code Snippets and Explanations

1. Project Setup
Create the project directory followed by installing Flask and Dash:
pip install Flask dash pandas firebase-admin

2. Firebase Authentication
Set up Firebase in our Flask app by first setting up firebase_config file with our API keys etc. We then initialize firebase in our app.py file as shown in the second image:

firebase_config.py > ...

```python
import pyrebase

firebase_config = {
    "apiKey": "AIzaSyBbdk5jF2hKAo0zcPuec4GwQsPL0MBYkIg",
    "authDomain": "budgetbuddy-ac47e.firebaseapp.com",
    "projectId": "budgetbuddy-ac47e",
    "storageBucket": "budgetbuddy-ac47e.appspot.com",
    "messagingSenderId": "23103361616",
    "appId": "1:23103361616:web:5c8baf11580a1138e93755",
    "measurementId": "G-F35EMDZ7TB",
    "databaseURL": ""
}

firebase = pyrebase.initialize_app(firebase_config)
auth = firebase.auth()
```

```python
#Initialize Firebase Authentication
app.secret_key = 'AIzaSyBbdk5jF2hKAo0zcPuec4GwQsPL0MBYkIg'
```

3. Data Handling with Pandas

Read and process CSV files:

```
#Load datasets into pandas dataframe(df)
transactions_df = pd.read_csv(r'C:\Users\kh\Downloads\projectBuddy\personal_transactions.csv')
budget_df = pd.read_csv(r'C:\Users\kh\Downloads\projectBuddy\Budget.csv')

#Create a bar chart for spending by category
spending_by_category = transactions_df.groupby('Category')['Amount'].sum().reset_index()
fig = px.bar(spending_by_category, x='Category', y='Amount', title='Spending by Category')
```

4. Visualization with Dash

We create a Dash app within our Flask app to carry out various data visualisations:

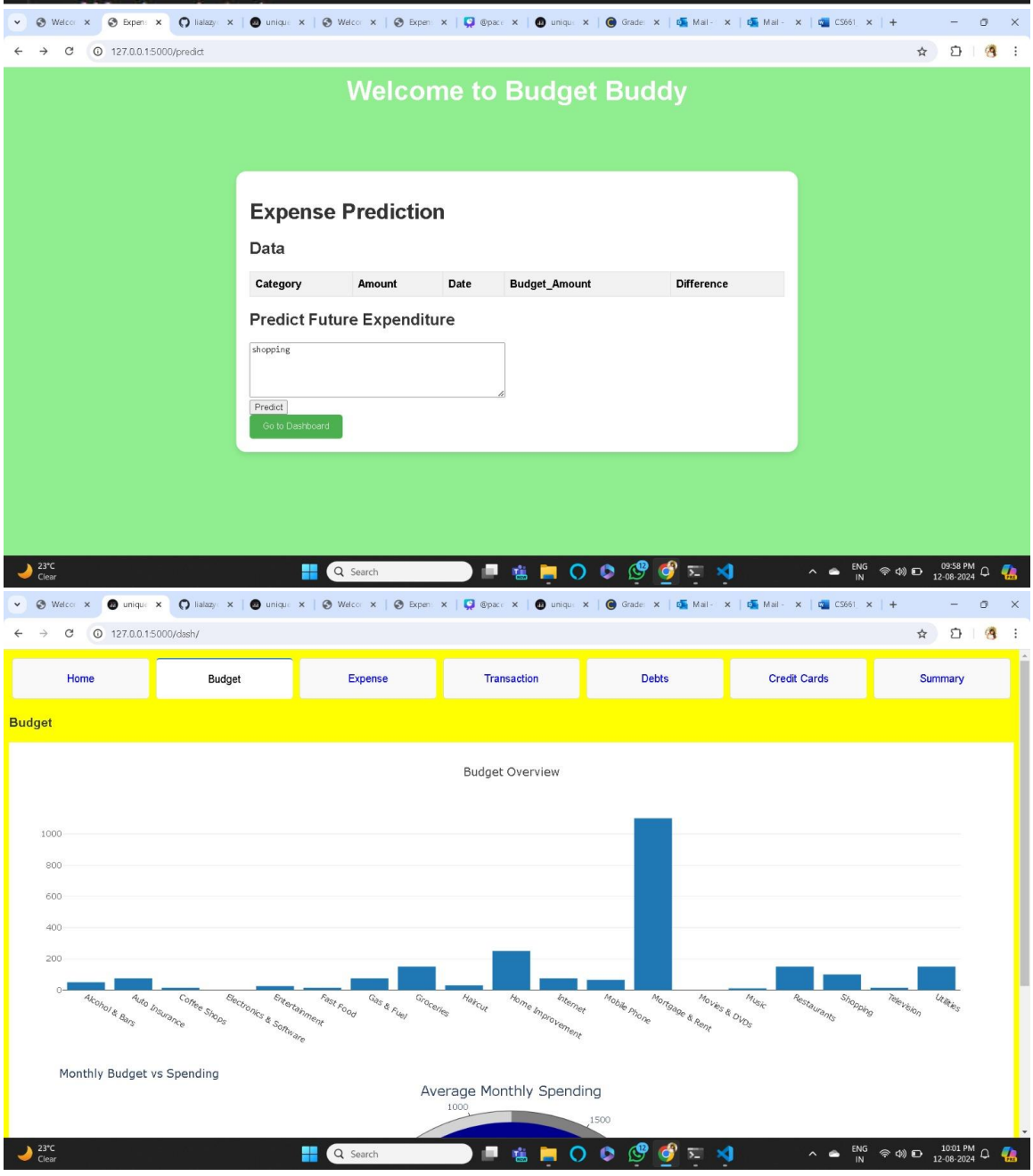
```
# Initialize Dash
dash_app = dash.Dash(__name__, server=app, url_base_pathname='/dash/', external_stylesheets=external_stylesheets)
dash_app.title = 'unique_dash_app'
app.layout = html.Div(id='dash-container')
```

Culculating monthly expenditure summarry for visualisation with Dash

```
#Convert 'Date' column to datetime
transactions_df['Date'] = pd.to_datetime(transactions_df['Date'])

#Extract month and year from our 'Date' column
transactions_df['Month'] = transactions_df['Date'].dt.to_period('M').astype(str)

#we roup spending by month and calculate total amount
monthly_summary = transactions_df.groupby('Month')['Amount'].sum().reset_index()

#Create a bar chart for monthly expenses
fig_monthly_summary = px.bar(monthly_summary, x='Month', y='Amount', title='Monthly Summary of Expenses')
```

Visualizing the summary in a Dash tab

```python
# Callback to update tab content
tabnine: test | explain | document | ask | Codiumate: Options | Test this function
@dash_app.callback(Output('tabs-content', 'children'),
              [Input('tabs', 'value')])
def render_content(tab):
    if tab == 'tab-1':
        return html.Div([
            html.H3('Additional Analytics'),

            dcc.Graph(figure=fig_monthly_summary),
            dcc.Graph(figure=fig_pie),
            dcc.Graph(figure=fig_spending)
        ])
```



Libraries and Tools Used

- **Flask**: A lightweight web application framework.
- **Dash**: A framework for building analytical web applications.
- **Pandas**: A data manipulation and analysis library.
- **Firebase Admin SDK**: For integrating Firebase authentication.
- **Plotly**: For creating interactive graphs and visualizations.
- **Sklearn:** To perform machine learning operations on our prediction data.

User Guide:

To setup and run the project, navigate to the project folder i.e: C:\Users\user1\projects\projectBuddy\ where the app.py file is located. After navigating to the file, click on the project path on your folder, clear the path and type in cmd, command prompt window will show. The type in the command prompt 'cd .' and this will start the project inside your code editor, for example, Visual Studio Code. When the project starts, open a new terminal in your code editor, type in python app.py and press enter. This will run the project in port 5000, a link will be provided to direct you to the web app page.

Output :

IV. Challenges Faced and Solutions

1. Module Integration:

Challenge: Because each framework handles routing and rendering differently, it was difficult to integrate Dash with Flask while preserving a seamless user experience.

Solution: To guarantee a smooth integration, extensive preparation and testing were carried out. Dash was used for the visual aspects, and Flask routes were used for data processing and prediction.

2. Managing JSON Data:

Challenge: When parsing and analysing JSON data from user inputs, mistakes could occasionally occur, particularly when managing NaN values.

Solution: Before being sent to Pandas for additional processing, the JSON data was preprocessed to replace NaN values with null. Debugging statements were included to quickly find and repair problems.

3. Problem

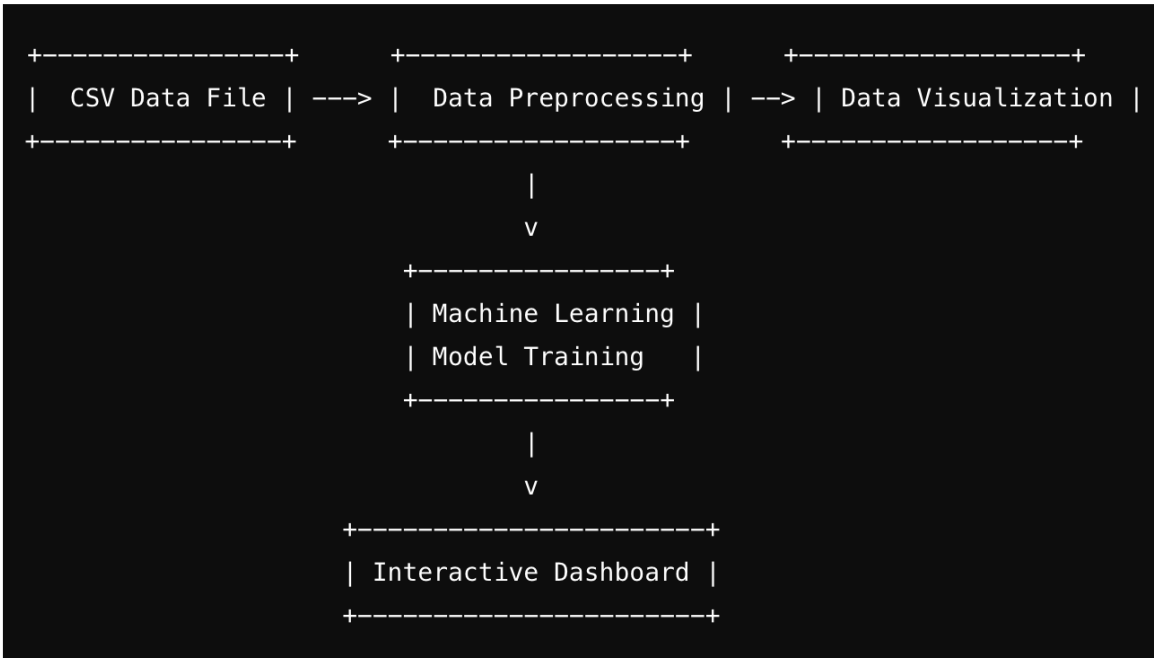
ModuleNotFoundError ModuleNotFoundError was a frequent problem for local modules such as utils.

Solution: Made sure that each module directory has __init__.py files in it and that the project directory structure was configured correctly. The project directory was added to the Python path, enabling correct module imports.

4. Styling and Layout:

Challenge: Designing a visually appealing and user-friendly interface required careful consideration of layout and styling.

Solution: CSS was used extensively to style the card view, tables, and buttons. The layout was tested on different devices to ensure responsiveness and usability.


CSV Data File
Data Preprocessing
Data Visualization
Machine Learning
Model Training
Interactive Dashboard

Code outline

Project Setup

- **Setup and Installation**:

```
# Create the project directory and install necessary libraries
mkdir expense_dashboard
cd expense_dashboard
pip install Flask dash pandas scikit-learn matplotlib
```

- **Directory Structure**:

```
expense_dashboard/
├── [app.py](<http://app.py/>)
├── data_processing.py
├── [model.py](<http://model.py/>)
├── utils/
│   └── data_processing.py
├── templates/
│   └── index.html
└── static/
└── style.css
```

Explanation: This sets up the project structure and installs Flask, Dash, and other essential libraries required for web development, data processing, and machine learning.

2. Data Handling with Pandas

- **Reading and Cleaning CSV Files**:

```
import pandas as pd

# Load data from CSV
def load_data(file_path):
    return pd.read_csv(file_path)

# Clean and preprocess the data
def clean_data(data):
    data.fillna({
        'Description': '',
        'Amount': 0,
```

```
    'Transaction Type': '',
    'Category': '',
    'Account Name': '',
  }, inplace=True)
  data['Amount'] = pd.to_numeric(data['Amount'], errors='coerce')
  data.dropna(subset=['Amount'], inplace=True)
  return data
```

- **Explanation**:
  - **load_data**: This function loads the CSV file containing transaction data.
  - **clean_data**: Cleans the data by handling missing values and converting the Amount column to numeric values. This ensures that the dataset is ready for analysis and visualization.

3. Data Processing and Feature Engineering

- **Feature Engineering**:

```
# Feature engineering to add new columns
def feature_engineering(data, budget_data):
  data['Day_of_Week'] = data['Date'].dt.dayofweek
  data['Is_Weekend'] = data['Day_of_Week'].apply(lambda x: 1 if x >= 5 else 0)
  data['Quarter'] = data['Date'].dt.quarter

  data['Adjusted_Amount'] = data.apply(lambda row: row['Amount'] if
row['Broad_Category'] == 'Income' else -row['Amount'], axis=1)
  data['Cumulative_Spending'] = data.groupby(['Category'])['Adjusted_Amount'].cumsum()

  merged_data = pd.merge(data, budget_data, how='left', on='Category')
  merged_data['Budget'] = merged_data['Budget'].fillna(0)
  merged_data['Difference'] = merged_data['Adjusted_Amount'] - merged_data['Budget']
  merged_data['Balance'] = merged_data['Adjusted_Amount'].cumsum()

  return merged_data
```

- **Explanation**:

- **feature_engineering**: This function adds important features like Day_of_Week, Is_Weekend, Quarter, and Cumulative_Spending to the dataset. It then merges the data with a budget file and calculates the Difference and Balance for each category, setting the stage for machine learning and visualization.

4. Training a Machine Learning Model

- **Random Forest Model**:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

def train_model(data):
    X = data[['Day', 'Month', 'Year', 'Day_of_Week', 'Is_Weekend', 'Quarter',
'Cumulative_Spending']]
    X = pd.get_dummies(X)
    y = data['Adjusted_Amount']

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"Mean Squared Error: {mse:.2f}")
    print(f"R^2 Score: {r2:.2f}")

    return model
```

- **Explanation**:
    - **train_model**: This function trains a Random Forest model using the processed data. It splits the data into training and test sets, fits the model, and evaluates its performance using metrics like Mean Squared Error (MSE) and R^2 Score.

5. Visualization with Dash

- **Integrating Dash for Visualization**:

```
import dash
import dash_core_components as dcc
import dash_html_components as html
import plotly.express as px

app = dash.Dash(__name__)

# Example of visualizing data with a bar chart
def visualize_data(monthly_spending):
   fig = px.bar(monthly_spending, x='Category', y='Spent', color='Category',
            title='Monthly Expenditure by Category')
   return fig

app.layout = html.Div([
   dcc.Graph(figure=visualize_data(monthly_spending))
])
```

- **Explanation**:
    - **visualize_data**: This function uses Plotly Express to create an interactive bar chart that displays monthly expenditure by category. The Dash framework then renders this chart on a web page, allowing users to interact with the data.

1. Data Description and Visualization Before Handling Outliers

Descriptive Statistics Before Handling Outliers:

- count: The total number of valid values in the Amount column is 806.
- mean: The mean (average) value of the data is 273.39, which gives a central tendency of the data.
- std (Standard Deviation): The standard deviation is 667.63, indicating a high level of dispersion in the data.
- min: The minimum value in the dataset is 1.75, representing the smallest transaction amount.
- max: The maximum value is 9200, suggesting that there is an extremely large transaction amount, likely an outlier.

These statistics indicate that the data contains some extreme values (outliers), as shown by the large difference between the mean and the maximum value.

Box Plot Before Handling Outliers:

- The box plot displays the distribution of the Amount data.
- Outliers: In this plot, several points lie outside the "whiskers" of the box plot, which are identified as outliers.
- Box Plot: A box plot is a graphical representation of data that shows the distribution's five-number summary: minimum, first quartile (Q1), median, third quartile (Q3), and maximum.

Key Concepts:

- Outliers: In statistics, these refer to extreme values that are far from the central trend of the data.
- Interquartile Range (IQR): This is a measure used to identify outliers. It is calculated as the difference between the third quartile (Q3) and the first quartile (Q1).

2. Data Description and Visualization After Handling Outliers

Descriptive Statistics After Handling Outliers:

- After handling outliers, the maximum value in the Amount column has been reduced to 270.67, which is close to the upper bound defined by Q3.
- mean: The mean has dropped to 89.56, indicating that the data is now more centered around a realistic value.
- std: The standard deviation has decreased to 100.38, meaning the data is now more concentrated and less dispersed.

These statistics suggest that the outliers have been effectively managed, leading to a more reasonable data distribution.

Box Plot After Handling Outliers:

- The updated box plot shows that all data points are now within the normal range, with no values lying outside the whiskers.
- The use of colors and line styles helps to clearly reflect the central tendency of the data.

Key Concepts:

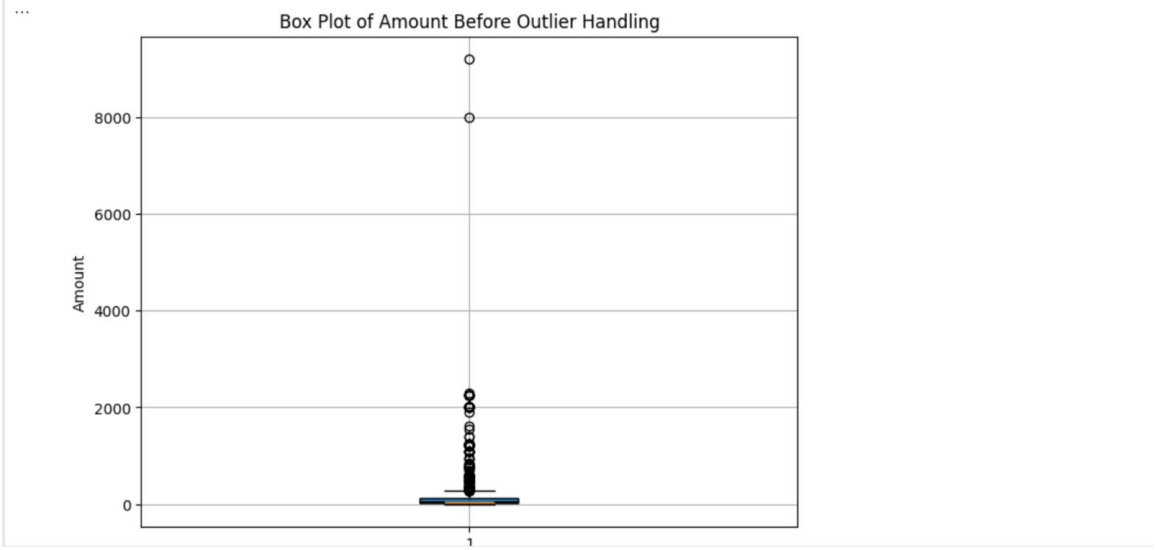
- Clip: In the context of outlier handling, clipping refers to restricting the values within a certain range. This is done using the clip(lower_bound, upper_bound) function in the code, which limits the Amount values to be within the defined lower and upper bounds.

Summary

The key points illustrated by the images and code are:

1. Presence of Outliers: The initial box plot shows that there are many outliers in the data, which significantly affect the mean and standard deviation.
2. Outlier Handling Method: You used the IQR method to detect and handle outliers by clipping values that fall outside the acceptable range.
3. Effectiveness of the Outlier Handling: By comparing the before and after box plots and descriptive statistics, you can confirm that the outlier handling was successful, leading to improved data quality.

```
Summary statistics before handling outliers:
count     806.000000
mean      273.391489
std       667.630374
min         1.750000
25%        15.687500
50%        37.480000
75%       117.680000
max      9200.000000
Name: Amount, dtype: float64
```



Explaination the Output

1. Column Index After Merging:
- The first part of the screenshot shows the column names after merging the transaction data with budget data.
- The columns include:
- Date: The date of the transaction.
- Description: A brief description of the transaction (e.g., Amazon, Netflix).

- Amount: The monetary value of the transaction.
- Transaction Type: Whether the transaction was a debit or credit.
- Category: The category of spending (e.g., Shopping, Mortgage & Rent).
- Account Name: The account used for the transaction (e.g., Platinum Card, Checking).
- Day, Year, Month: The day, year, and month extracted from the date.
- Broad_Category: A broader categorization of the spending category (e.g., Miscellaneous, Housing).
- Day_of_Week: The day of the week on which the transaction occurred.
- Is_Weekend: A binary feature indicating if the transaction occurred on a weekend.
- Quarter: The quarter of the year when the transaction occurred.
- Adjusted_Amount: The transaction amount adjusted based on whether it's income or expense.
- Cumulative_Spending: The cumulative sum of spending in that category up to that date.
- Month_Category_Interaction: An interaction feature combining month and category.
- Budget, Difference, Balance, Budget_budget: Budget-related columns, indicating how the transaction compares with the planned budget.

2. Sample Data Output:

- The second part of the screenshot displays the first few rows of the dataset after processing.
- For example:
- Date: 2018-01-01
- Description: Amazon
- Amount: 11.11
- Transaction Type: Debit
- Category: Shopping
- Broad_Category: Miscellaneous
- Budget: The budget amount for that category, which may be zero if not available.

3. Monthly Summary and Recommendations:

- The system provides recommendations based on spending relative to the budget. For instance:
- Gas & Fuel: "You are within the budget for Gas & Fuel. Keep it up!"
- Groceries: Similar feedback indicating that the user is managing their budget well.

Link to the Data Processing Code

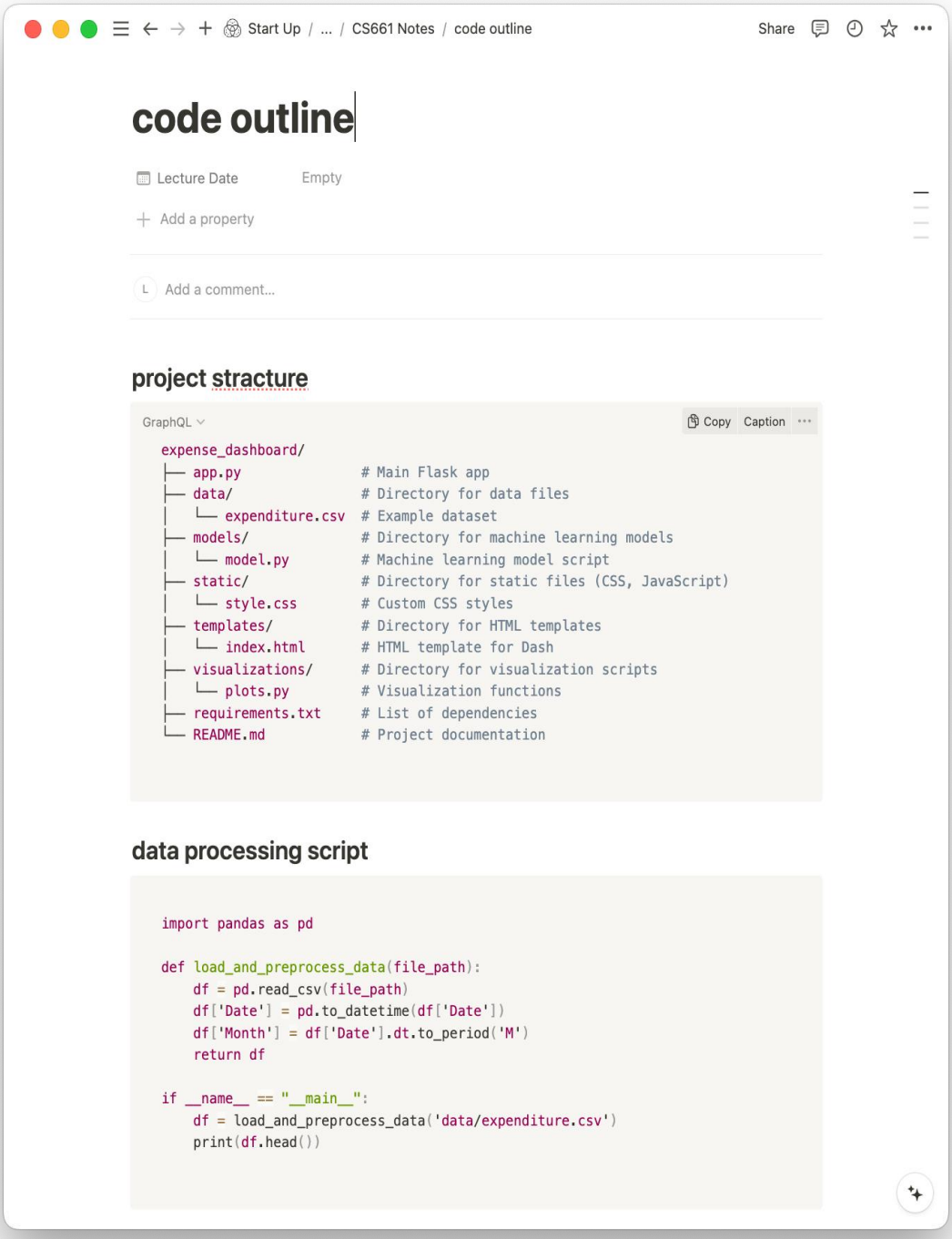
- Data Cleaning (clean_data):
- The code fills missing values and ensures that all transaction amounts are numeric, which is crucial for accurate analysis.
- Feature Engineering (feature_engineering):
- Various features like Day_of_Week, Is_Weekend, and Cumulative_Spending are created to enrich the dataset, enabling more insightful analysis.

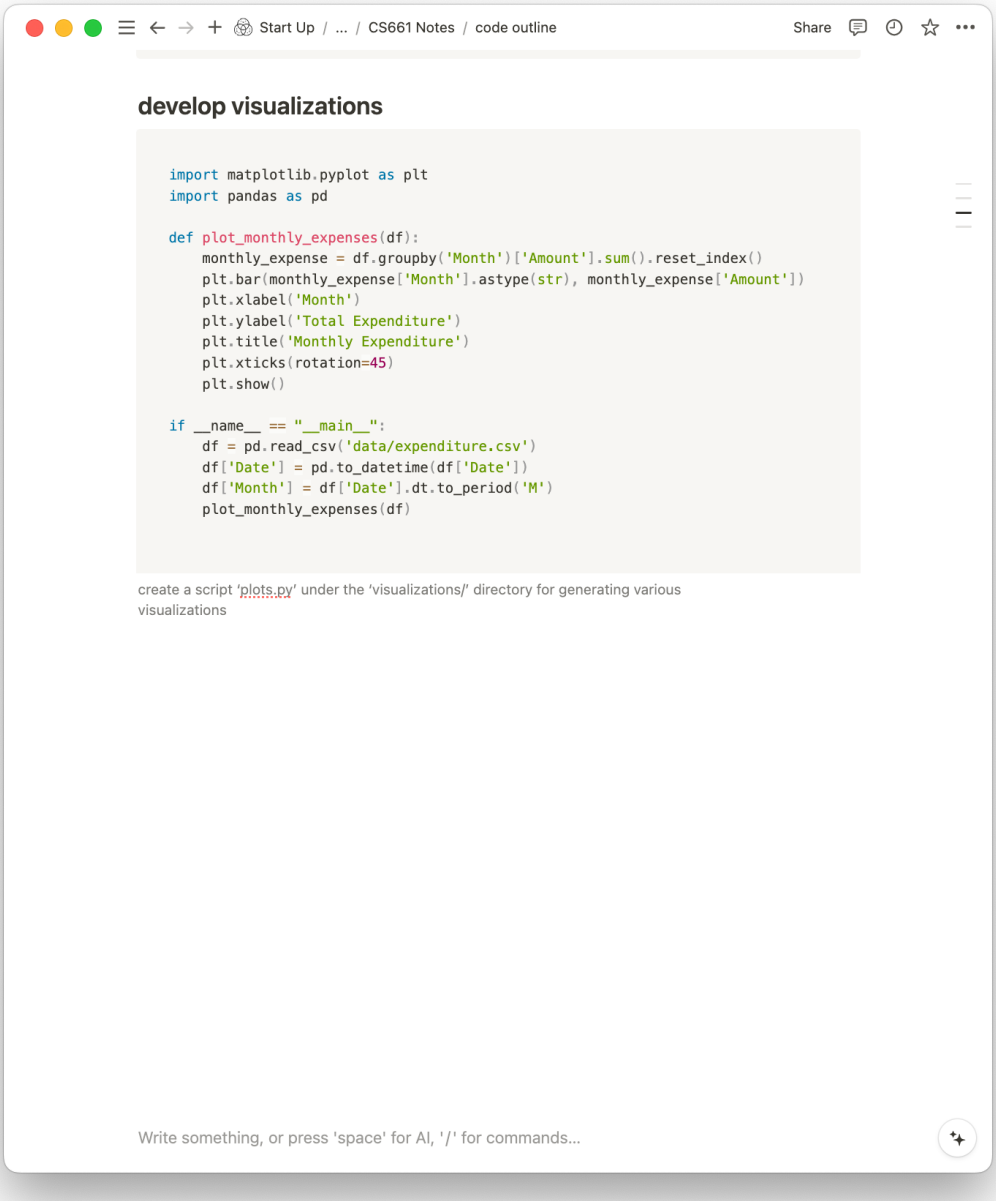
- Interaction features like Month_Category_Interaction help in capturing the combined effect of month and category on spending behavior.
- Outlier Handling (handle_outliers):
- The outliers were capped at the upper bound to prevent them from skewing the analysis, as indicated by the summary statistics and the box plot in the previous explanations.

6. Libraries and Tools Used

- **Explanation**:
    - **Flask**: For building the web application.
    - **Dash**: For creating interactive data visualizations.
    - **Pandas**: For data manipulation and processing.
    - **Scikit-learn**: For machine learning model training and evaluation.
    - **Matplotlib**: For creating plots to visualize data during the development process.

Code Outline :


Start Up / ... / CS661 Notes / code outline
Share
code outline
Lecture Date
Empty
Add a property
Add a comment...
project structure
GraphQL
Copy
Caption
expense_dashboard/
├── app.py # Main Flask app
├── data/ # Directory for data files
│ └── expenditure.csv # Example dataset
├── models/ # Directory for machine learning models
│ └── model.py # Machine learning model script
├── static/ # Directory for static files (CSS, JavaScript)
│ └── style.css # Custom CSS styles
├── templates/ # Directory for HTML templates
│ └── index.html # HTML template for Dash
├── visualizations/ # Directory for visualization scripts
│ └── plots.py # Visualization functions
├── requirements.txt # List of dependencies
└── README.md # Project documentation
data processing script
import pandas as pd
def load_and_preprocess_data(file_path):
df = pd.read_csv(file_path)
df['Date'] = pd.to_datetime(df['Date'])
df['Month'] = df['Date'].dt.to_period('M')
return df
if __name__ == "__main__":
df = load_and_preprocess_data('data/expenditure.csv')
print(df.head())



V. Risks and Challenges

Data Privacy and Security:

- Risk: User data, especially personal financial information, is sensitive and must be protected.

- Mitigation: Implement strong encryption for data storage and transmission. Use secure authentication methods and ensure compliance with data protection regulations (e.g., GDPR, CCPA).

Data Accuracy and Reliability:

- Risk: Inaccurate data input or data processing errors could lead to incorrect financial recommendations.

- Mitigation: Implement data validation techniques to ensure data accuracy. Regularly test and validate the algorithms to ensure they are functioning correctly.

User Adoption and Engagement:

- Risk: Users may find the tool difficult to use or may not trust AI-driven recommendations.

- Mitigation: Conduct user experience (UX) testing and iteratively improve the interface based on feedback. Provide clear explanations of AI recommendations and transparency on how predictions are made.

Technical Challenges:

- Risk: Integrating various technologies (e.g., Dash by Plotly, machine learning models) could present technical difficulties.

-Mitigation: Ensure that the development team has expertise in the required technologies. Break down the project into smaller, manageable tasks and conduct regular code reviews.

Scalability Issues:

- Risk: As the number of users grows, the system might face performance issues.

- Mitigation: Design the system architecture to be scalable. Use cloud services that can handle scaling and conduct performance testing to identify and address bottlenecks.

AI Model Limitations:

- Risk: The linear regression model used for predictions might need to be more accurate for all users' spending patterns.

- Mitigation: Continuously improve the model based on user feedback and new data. Consider incorporating more advanced models, if necessary, while balancing complexity and resource requirements.

Project Scope Creep:

- Risk: The project may expand beyond its initial scope, leading to delays and resource overuse.

- Mitigation: Clearly define the project scope and stick to it. Use project management tools to track progress and manage changes.

Integration with External Systems:

- Risk: Future integration with real-time transaction systems or additional financial planning tools might introduce compatibility issues.

- Mitigation: Design the system with modularity in mind, allowing for easy integration of new features. Use standardized APIs and maintain thorough documentation.

User Data Management:

- Risk: Managing large volumes of user data can be challenging, leading to potential data loss or corruption.

- Mitigation: Implement robust data backup and recovery procedures. Use reliable database management systems and conduct regular audits.

Legal and Regulatory Compliance:

- Risk: Failure to comply with financial regulations and data protection laws can result in legal consequences.

- Mitigation: Stay informed about relevant regulations and ensure the project adheres to them. Consult with legal experts if necessary.

Mitigation Strategies:

- Regular Testing and Quality Assurance: Implement a comprehensive testing strategy that includes unit testing, integration testing, and user acceptance testing.

- User Training and Support: Provide training materials to help users get the most out of the tool. This can include tutorials, FAQs, and customer support channels.

- Agile Development Methodology: Use agile practices for flexibility and iterative improvements. Regularly review and adapt the project plan based on progress and feedback.

-Risk Management Plan: Develop a risk management plan that identifies potential risks, assesses their impact, and outlines mitigation strategies. Review and update this plan regularly throughout the project lifecycle.

Here you can find our link to our GitHub : https://github.com/lialazyoaf/cs661-40600

By proactively identifying these risks and implementing appropriate mitigation strategies, the BudgetBuddy project can improve its chances of success and deliver a valuable tool for personal finance management.

VI. Conclusion

The BudgetBuddy project aims to create a comprehensive, user-friendly tool for managing personal finances through effective expense tracking, data visualization, and AI-driven suggestions. By leveraging modern technologies such as data visualization libraries and machine learning models, BudgetBuddy addresses the growing need for accessible and efficient personal finance management solutions. The project is expected to significantly impact users by providing valuable insights and recommendations, ultimately promoting better financial habits and overall financial wellness.

References

Fatunde, B. (n.d.). *Personal finance* [Data set]. Kaggle. https://www.kaggle.com/datasets/bukolafatunde/personal-finance

Kann. (n.d.). *Feature selection and data visualization* [Notebook]. Kaggle. https://www.kaggle.com/code/kanncaa1/feature-selection-and-data-visualization

Stack Overflow. (n.d.). Running a dash app within a flask app. Retrieved from https://stackoverflow.com/questions/45845872/running-a-dash-app-within-a-flask-app

Plotly. (n.d.). Dash HTML Components. Retrieved from https://dash.plotly.com/dash-html-components

Sentdex. (2022, September 20). Dash Plotly - Flask integration tutorial [Video]. YouTube. https://youtu.be/pGMvvq7R1IM

Academind. (2022, August 15). Build Dashboards with Plotly Dash [Video]. YouTube. https://youtu.be/9MHYHgh4jYc

Charming Data. (2022, July 10). Plotly Dash DataTable with Sorting, Filtering, and More [Video]. YouTube. https://youtu.be/Jl9XzSPXSe4

StatQuest with Josh Starmer. (2022, June 30). Principal Component Analysis (PCA) in Python - Step-by-Step [Video]. YouTube. https://youtu.be/fakRnkw0s9o