Personal Finance Transaction Analyzer

A Python-based financial transaction analyzer that provides insights into your spending and income patterns. This interactive command-line tool helps you understand your financial habits through automated analysis of deposits and withdrawals.

Features

- **Transaction Summary**: Calculate total deposits, withdrawals, and current balance
- **Financial Analysis**: Identify largest transactions and compute averages
- **Interactive CLI**: User-friendly command-line interface with menu options
- **Spending Insights**: Track expenses by category (Utilities, Rent, Groceries, etc.)
- **Income Tracking**: Monitor salary, investments, and other income sources
- **Statistical Analysis**: Calculate average deposits and withdrawals
- **Pattern Recognition**: Identify spending trends and outliers

Installation

Prerequisites

- Python 3.7 or higher
- Jupyter Notebook (optional, for running the .ipynb file)

Setup

1. Clone the repository:

bash

```
git clone https://github.com/YOUR-USERNAME/transaction-analyzer.git
cd transaction-analyzer
```

2. No external dependencies required - uses Python standard library only!

Usage

Running the Jupyter Notebook

bash

```
jupyter notebook "Transaction Analyzer Project.ipynb"
```

Running as Python Script

Convert and run as a standalone script:

bash

```
python transaction_analyzer.py
```

Interactive Menu

When you run the program, you'll see an interactive menu:

```
Transaction Analyzer

Choose an option
1. Print summary (type 'print')
2. Analyze transactions (type 'analyze')
3. Stop program (type 'stop')

Enter your option:
```

Menu Options:

- **print**: Display financial summary with total deposits, withdrawals, and balance
- **analyze**: Show detailed analysis including largest transactions and averages
- **stop**: Exit the program

Project Structure

```
transaction-analyzer/
│
├── Transaction Analyzer Project.ipynb    # Main Jupyter Notebook
├── transaction_analyzer.py            # Standalone Python script
├── README.md                      # This file
├── requirements.txt                # Python dependencies
├── sample_data.py                  # Sample transaction data
└── LICENSE                      # License file
```

Functions

1. `print_transactions(transactions)`

Displays all transactions in a formatted list.

Parameters:

- `transactions` (list): List of tuples containing (amount, description)

Example Output:

$749.17 - Investment Return

$-11.54 - Utilities

$-247.58 - Online Shopping

Usage:

python

```
print_transactions(data)
```

---

2. `print_summary(transactions)`

Calculates and displays financial summary statistics.

Parameters:

- `transactions` (list): List of tuples containing (amount, description)

Displays:

- Total deposited amount
- Total withdrawn amount
- Current balance (deposits - withdrawals)

Example Output:

4504.7

-2071.34

$2433.36

Implementation Details:

- Uses list comprehension to filter deposits (amount >= 0)
- Uses list comprehension to filter withdrawals (amount < 0)
- Calculates balance as sum of all transactions

Usage:

python

```
print_summary(data)
```

---

3. `analyze_transactions(transactions)`

Performs detailed statistical analysis on transactions.

Parameters:

- `transactions` (list): List of tuples containing (amount, description)

Analyzes:

- Largest withdrawal transaction
- Largest deposit transaction
- Average deposit amount
- Average withdrawal amount

Example Output:

```
Largest withdrawals: (-881.51, 'Utilities')
Largest deposit: (981.17, 'Investment Return')
Average deposit is: 565.3375
Average_withdrawals is: -295.9057142857142
```

Implementation Details:

- Sorts transactions by amount
- Uses ternary operator to prevent division by zero
- Calculates averages only if transactions exist

Usage:

python

```
analyze_transactions(data)
```

---

4. Interactive Menu Loop

Provides user-friendly interface for accessing all functions.

Features:

- Continuous loop until user chooses to stop
- Case-insensitive input handling
- Input validation with error messages
- Clear menu display

Commands:

- `print` → Calls `print_summary()`
- `analyze` → Calls `analyze_transactions()`
- `stop` → Exits the program

Sample Data

The project includes sample transaction data with various categories:

Transaction Categories:

Category	Type	Example
Investment Return	Income	$749.17, $981.17
Salary	Income	$220.79, $308.49, $870.64, $518.14
Gift	Income	$563.70
Rent	Mixed	$310.60 (income), -$410.65 (expense)
Utilities	Expense	-$11.54, -$881.51
Online Shopping	Expense	-$247.58
Car Maintenance	Expense	-$49.85, -$205.55
Groceries	Expense	-$264.66

Key Programming Concepts

This project demonstrates several important Python concepts:

1. **Tuple Data Structures**: Efficient storage of transaction data
2. **List Comprehensions**: Filtering and transforming data elegantly
3. **Conditional Logic**: Making decisions based on transaction types
4. **User Input Handling**: Interactive CLI with input validation
5. **String Formatting**: Clean, readable output using f-strings
6. **Aggregate Functions**: Using `sum()` and `len()` for calculations
7. **Ternary Operators**: Concise conditional expressions
8. **While Loops**: Creating interactive program flows

Analysis Insights

Financial Health Indicators

Based on the sample data, the analyzer reveals:

Income Sources:

- Salary: 4 transactions
- Investment Returns: 2 transactions
- Gifts: 1 transaction

Expense Categories:

- Utilities: Highest single expense (-$881.51)
- Rent: Major recurring expense
- Car Maintenance: Maintenance costs
- Groceries & Shopping: Daily expenses

Key Metrics:

- Total Income: $4,504.70
- Total Expenses: -$2,071.34
- Net Balance: $2,433.36
- Average Deposit: $565.34
- Average Withdrawal: -$295.91

Contributing

Contributions are welcome! Here's how you can help:

1. Fork the repository
2. Create a feature branch (`git checkout -b feature/AmazingFeature`)
3. Commit your changes (`git commit -m 'Add some AmazingFeature'`)
4. Push to the branch (`git push origin feature/AmazingFeature`)
5. Open a Pull Request

Contribution Ideas:

- Add data visualization features
- Implement CSV import/export
- Create unit tests
- Add date/time support
- Improve output formatting
- Add more statistical analysis

Learning Outcomes

Perfect for learning:

- Python data structures (tuples, lists)
- Functional programming concepts
- List comprehensions and filtering
- User input handling
- Financial data analysis
- CLI application development
- Data aggregation and statistics

License

Educational Use

This project is ideal for:

- **Beginners**: Learning Python basics and data manipulation
- **Students**: Understanding financial calculations and algorithms
- **Self-learners**: Building practical, real-world applications
- **Educators**: Teaching data analysis and user interaction

Contact

For questions, suggestions, or feedback, please open an issue on GitHub.

Track Your Finances, Master Your Money!