Universität St.Gallen

School of Management, Economics, Law, Social Sciences, and International Affairs

Python Group Project: Expense Categorization and Report

Authors

Davet Luca (20-612-123)

Prontera Giuliano (19-615-467)

Zimmer Nicolas (19-609-460)

Lecturer

Silic Mario

December 23, 2022

Assignment Overview

This assignment focuses on processing, cleansing, and visualizing expenses extracted from a list of transactions (possibly) provided in the form of a CSV file by your bank. The goal is to insert transactions within categories name that you create based on your needs and display them in a useful way.

Assignment Deliverable

The deliverable for this assignment is the following file:

Code - Expense Report Program.ipynb – the Jupyter Notebook for your program

Assignment Background

There were times when people wrote their expenses down on a piece of paper, and used this to track their spending habits. Nowadays, most people spend money using their credit/debit cards and via bank transfers. At the same time, some banks have made it easier for their customers to track their expenses. One way to gain a better understanding is by categorizing expenses into different baskets – like groceries, transportation, personal care, utilities, etc.

Oftentimes, however, these tracking tools are limited in terms of how customers can customize their spending categories. Luckily, with the help of Python, it is possible to solve this customization problem.

The goal of our project is to enable the customization of spending reports and make it easier for people to understand their finances, tailored to their specific spending habits. Our program allows users to choose their expense categories, and then use this to view a breakdown of their expenses. To visualize the expenses, we use two graphs that display the data in both relative and absolute terms.

Steps to Do Before Running the Code

1. Download your personal history expenses report as csv file (or download the one we provide as example)
2. Access your Phyton Console
3. Download Plotly library on your machine - if your Using Anaconda, go on *Environments* and download the plotly library
4. Open the Code - Expense Report Program.ipynb file
5. If you open the file with Google Colab, pls make sure to uncomment the first to line of code to import the csv file

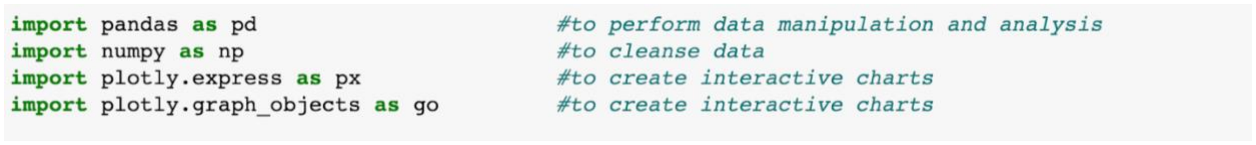
Assignment Specifications

Provide the following functions – you get to use your own expense report as csv file and choose the categories name that best suits you. Also, you will likely want more functions and visual representation that you personalize in line with your own needs.

We provide an example to show how a possible application of the program we created looks like. We created a csv file containing 60 transactions (5 per month) that you can use to run the program and get a first overview.

STEP 1: IMPORT PHYTON LIBRARIES

First thing we need to do is import the python libraries that we are going to use to perform the needed actions.

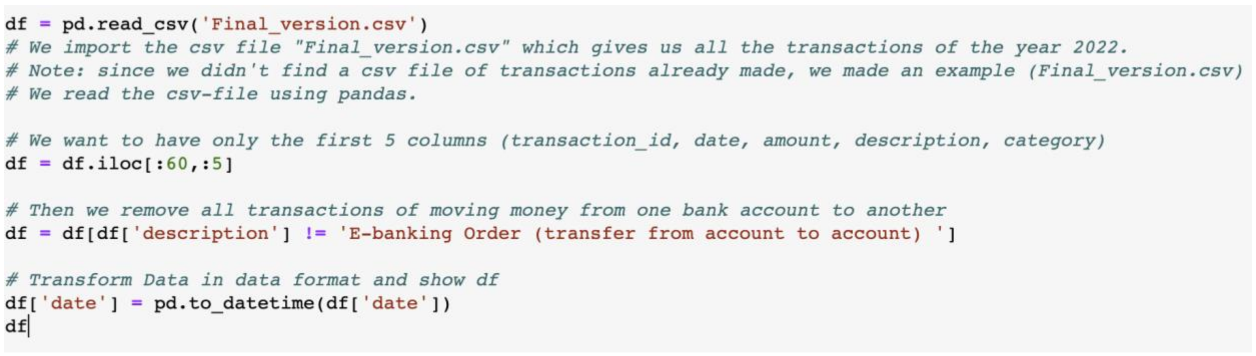


```
import pandas as pd                         #to perform data manipulation and analysis
import numpy as np                          #to cleanse data
import plotly.express as px                 #to create interactive charts
import plotly.graph_objects as go           #to create interactive charts
```

STEP 2: GET THE DATA

The second step requires getting all the transaction data. This can be easily done by downloading your transactions history report as a csv file from your online banking portal.
For the purpose of this exercise, we extracted from our csv file the 5 columns of most interest (transactions_id, date, amount, description, category) (**IMPORTANT: when you download your csv file columns might have different names, that must be changed within the program in order to run properly).**

After importing the file, we translate our transaction data into a Pandas DataFrame. For simplicity, we remove all transactions related to intra account transfers, and we only focus on expenses and income

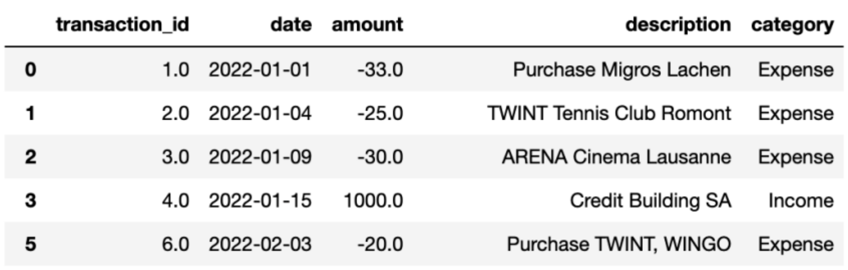


```
df = pd.read_csv('Final_version.csv')
# We import the csv file "Final_version.csv" which gives us all the transactions of the year 2022.
# Note: since we didn't find a csv file of transactions already made, we made an example (Final_version.csv)
# We read the csv-file using pandas.

# We want to have only the first 5 columns (transaction_id, date, amount, description, category)
df = df.iloc[:60,:5]

# Then we remove all transactions of moving money from one bank account to another
df = df[df['description'] != 'E-banking Order (transfer from account to account) ']

# Transform Data in data format and show df
df['date'] = pd.to_datetime(df['date'])
df
```



	transaction_id	date	amount	description	category
0	1.0	2022-01-01	-33.0	Purchase Migros Lachen	Expense
1	2.0	2022-01-04	-25.0	TWINT Tennis Club Romont	Expense
2	3.0	2022-01-09	-30.0	ARENA Cinema Lausanne	Expense
3	4.0	2022-01-15	1000.0	Credit Building SA	Income
5	6.0	2022-02-03	-20.0	Purchase TWINT, WINGO	Expense

In this subset of the df we can see that transactions are categorized in the column named "*category*" as "*Expense*" or "*Income*". Here is where we're manipulating our df in the next steps.

STEP 3: MANIPULATING THE DATA AND CUSTOMIZE YOUR TRANSACTIONS CATEGORY

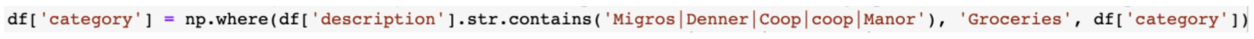
STEP 3.1. CREATE YOUR OWN CATEGORIES

Now it's time to define your own categories where transactions should fall into. In our case we choose 13 categories:

- Groceries, Sport, Rent&Bills, Transport, Materialistic Desires, Entertainment, Income, Resturants&Bars, Travel, Cash Withdrawal, Gifts, Personal Care, Clothes

STEP 3.2. ASSIGN TRANSACTIONS TO THE CORRECT CATEGORY

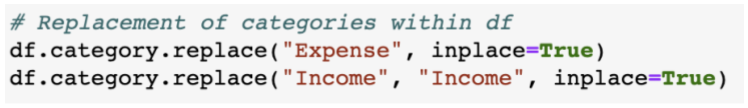
After defining the categories you want to use to manage your transactions, we can start manipulating the df moving the transactions into the category we want to fit them into. The way of doing this step is to locate the rows **containing** a specific string and assign those to a certain category (E.g. all rows under *category* containing in their *description* the string *Migros, Denner, Coop, coop* **(Pay attention in differentiating capital and lower letters)**, or *Manor* should be put under the category *Groceries*)



```
df['category'] = np.where(df['description'].str.contains('Migros|Denner|Coop|coop|Manor'), 'Groceries', df['category'])
```

STEP 3.4. CHANGE THE CATEGORIES' NAMES

Now we have to change the name of the existing categories with the new ones we created.



```
# Replacement of categories within df
df.category.replace("Expense", inplace=True)
df.category.replace("Income", "Income", inplace=True)
```

The first line of code replaces the value "*Expense*" with None (an empty value), placing all previously named transactions in their respective new categories. The second line of code replaces the value "Income" with the string "Income". As with the first line, the inplace parameter specifies that the replacement should be done in-place, so the original DataFrame is modified.
It's worth noting that the second line of code is essentially a no-op, since it's replacing the value "Income" with the same value. This line of code could be omitted without affecting the output. We inserted this line just to let you understand easier the process, and in case you want to name your Income category in a different way.

STEP 4: DESIGN YOUR EXPENSE REPORT

After completion of step 3 we should have a DataFrame containing transactions that falls into categories that best suits our needs. We are now ready to display visually the result.

4.1. CREATE A TABLE THAT GROUPS TRANSACTIONS BY CATEGORY

This section refers to the part of code commented as *#CREATE A TABLE WITH EACH EXPENSE CATEGORY PER MONTH.*

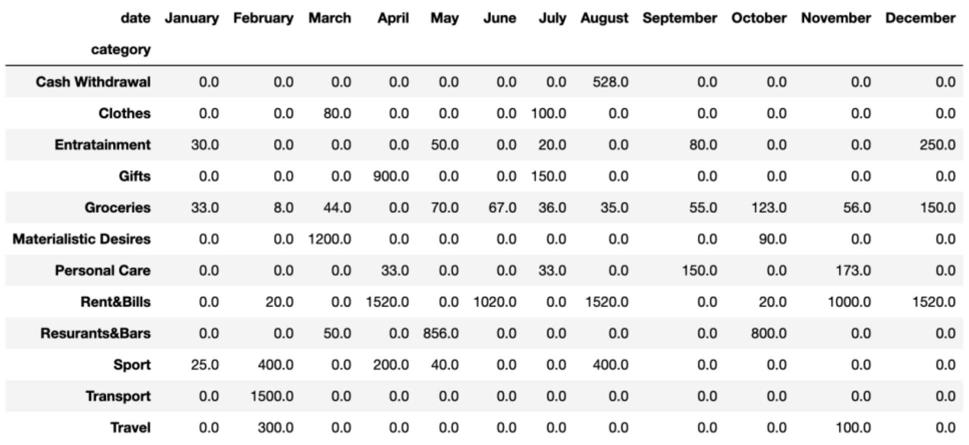
This code creates a table that summarizes the expenses for each category by month. The data is first sorted by date, and then grouped by month and transaction category. The resulting data is then pivoted to create a table with months as columns and categories as rows. Finally, the table is displayed.

Here is a more detailed explanation of what each line of code does:

- df.sort_values(by='date', inplace=True): This line sorts the dataframe df by the 'date' column. The inplace parameter is set to True, which means that the original dataframe will be modified and sorted in place, rather than creating a new sorted dataframe.
- monthly_expenses = df[df['category'] != 'Income'].groupby([df['date'].dt.strftime('%B'), 'category'])['amount'].sum().reset_index(): This line creates a new dataframe called monthly_expenses that groups the data in df by month and transaction category, and sums the 'amount' column for each group. The resulting data is then reset to a new index. The data is filtered to exclude rows with a 'category' of 'Income'.
- months = ['January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December']: This line creates a list of the months in the desired order.

- monthly_expenses['date'] = pd.Categorical(monthly_expenses['date'], categories=months, ordered=True): This line converts the 'date' column in the monthly_expenses dataframe to a categorical variable with the desired order of the months.
- table = pd.pivot_table(monthly_expenses, index='category', columns='date', values='amount', aggfunc=np.sum): This line creates a pivot table from the monthly_expenses dataframe, with 'category' as the rows, 'date' as the columns, and the sum of the 'amount' column as the values. The resulting pivot table is stored in a new dataframe called table.
- table: This line displays the contents of the table dataframe.

The final output looks like this:



date	January	February	March	April	May	June	July	August	September	October	November	December
category												
Cash Withdrawal	0.0	0.0	0.0	0.0	0.0	0.0	0.0	528.0	0.0	0.0	0.0	0.0
Clothes	0.0	0.0	80.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0	0.0	0.0
Entratainment	30.0	0.0	0.0	0.0	50.0	0.0	20.0	0.0	80.0	0.0	0.0	250.0
Gifts	0.0	0.0	0.0	900.0	0.0	0.0	150.0	0.0	0.0	0.0	0.0	0.0
Groceries	33.0	8.0	44.0	0.0	70.0	67.0	36.0	35.0	55.0	123.0	56.0	150.0
Materialistic Desires	0.0	0.0	1200.0	0.0	0.0	0.0	0.0	0.0	0.0	90.0	0.0	0.0
Personal Care	0.0	0.0	0.0	33.0	0.0	0.0	33.0	0.0	150.0	0.0	173.0	0.0
Rent&Bills	0.0	20.0	0.0	1520.0	0.0	1020.0	0.0	1520.0	0.0	20.0	1000.0	1520.0
Resurants&Bars	0.0	0.0	50.0	0.0	856.0	0.0	0.0	0.0	0.0	800.0	0.0	0.0
Sport	25.0	400.0	0.0	200.0	40.0	0.0	0.0	400.0	0.0	0.0	0.0	0.0
Transport	0.0	1500.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Travel	0.0	300.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	100.0	0.0

4.2. CREATE A BAR CHART THAT DISPLAYS CATEGORIES

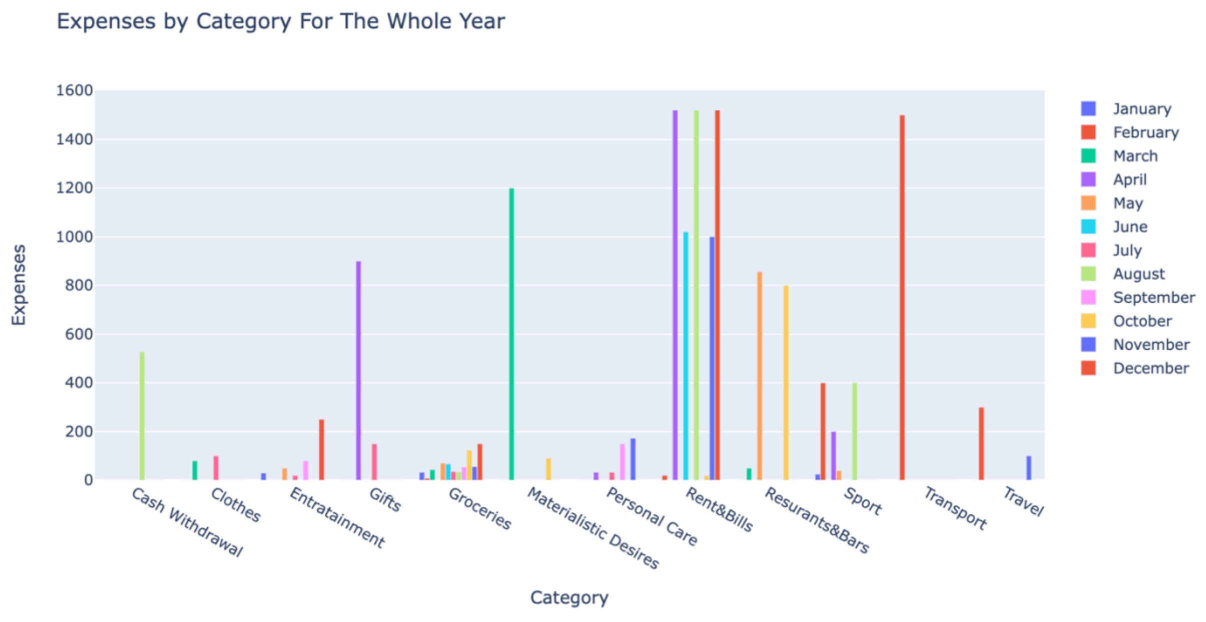
This section refers to the part of code commented as *#CREATE A BAR CHART WITH EXPENSES PER MONTH PER EACH CATEGORY.*

This code creates a bar chart that shows expenses per month for each category. The chart is created using the plotly library, specifically the graph_objects module.

Here is a more detailed explanation of what each line of code does:

- categories = list(table.index): This line creates a list of categories from the index of the table data frame.
- january_values = list(table['January']), february_values = list(table['February']), etc.: These lines create a list of values for each month from the corresponding column in the table dataframe.
- fig = go.Figure(data=[...])): This line creates a new figure object using plotly.graph_objects and specifies the data for the chart as a list of bar objects. Each bar object represents a month and has a name, a list of categories for the x-axis, and a list of values for the y-axis.
- fig.update_layout(title='Expenses by Category For The Whole Year'): This line sets the title of the chart to 'Expenses by Category For The Whole Year'.
- fig.update_layout(xaxis_title='Category'): This line sets the title of the x-axis to 'Category'.
- fig.update_layout(yaxis_title='Expenses'): This line sets the title of the y-axis to 'Expenses'.
- fig.show(): This line displays the chart.

The final output is an interactive bar chart that looks like this:



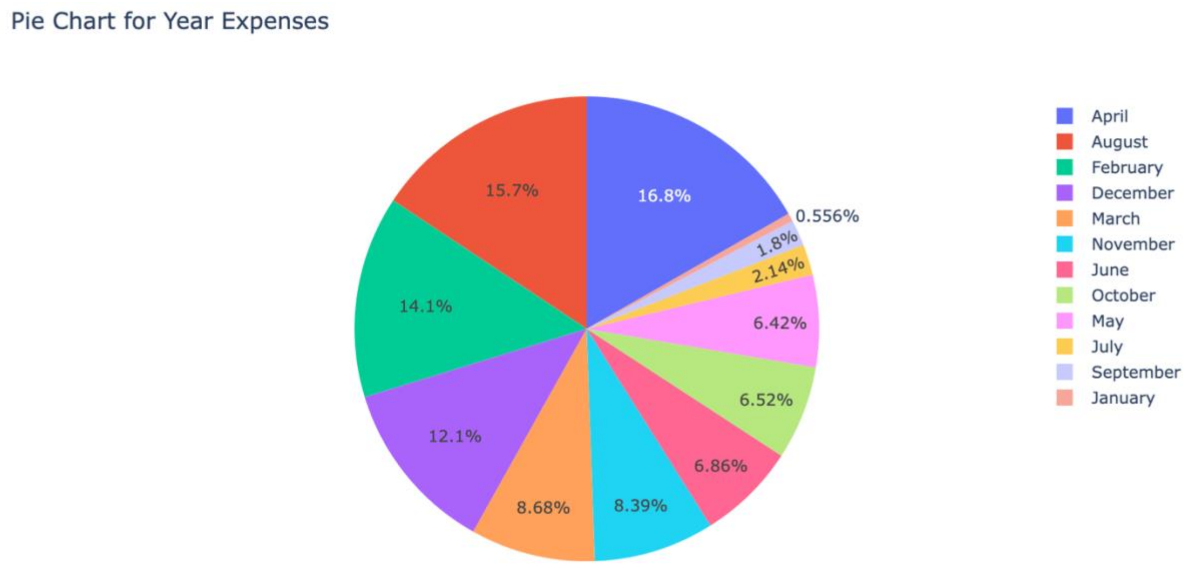
4.3. CREATE A PIE CHART

This section refers to the part of code commented as *#CREATE A PIE CHART DISPLAYING YEARLY SHARE OF EXPENSES FOR EACH MONTH*.
This code creates a pie chart that shows the yearly share of expenses for each month. The chart is created using the plotly library, specifically the graph_objects module.

Here is a more detailed explanation of what each line of code does:

- fig_2 = go.Figure(data=[go.Pie(labels=months, values=[sum(january_values), sum(february_values), sum(march_values), sum(april_values), sum(may_values), sum(june_values), sum(july_values), sum(august_values), sum(september_values), sum(october_values), sum(november_values), sum(december_values)])]): This line creates a new figure object using plotly.graph_objects and specifies the data for the chart as a pie object. The pie object has a list of labels (the months) and a list of values (the sums of the expenses for each month).
- fig_2.update_layout(title='Pie Chart for Year Expenses'): This line sets the title of the chart to 'Pie Chart for Year Expenses'.
- fig_2.show(): This line displays the chart.

The final output is an interactive pie chart that looks like this:


Pie Chart for Year Expenses
April
August
February
December
March
November
June
October
May
July
September
January