Data Analysis Initial Assessment Report

Mojdeh Hosseini — mojdeh.h.hosseini@gmail.com

January 24, 2025

1 Introduction

During the "Test Data Analyst" exercise, several tables were provided. These tables contained users, and transactions, each with slightly different columns. In order to consolidate all information, these tables were **merged on the `Account ID` field** and combined into a single Excel file named `Joined_database.xlsx`. The newly created, comprehensive dataset allowed for a unified view of user transactions spanning multiple accounts. You can see the results and the code on **Visualization jupyter notebook**.

With this merged data, the following objectives were pursued:

- Determine whether the user has a mortgage, a credit card, or ongoing projects.
- Identify if the user receives a salary (and approximate its amount).
- Investigate any mobile phone providers.
- Categorize user spending to visualize monthly expenditures and daily inflows/outflows.

2 Data and Methodology

The dataset contains columns such as `Account ID`, `Date`, `Amount In`, `Amount Out`, `Balance`, transaction descriptions, and other financial indicators. Analysis steps included:

2.1 Data Cleaning

- **Renamed columns** for consistency (removing spaces, standardizing casing).
- **Parsed** date fields as `datetime` and converted monetary values to floats.
- **Sorted** rows by date and removed invalid entries.

2.2 Categorization

A Python function scanned each transaction's description for keywords (e.g., "mortgage," "deposit," "credit card," etc.) to label it as:

- `Mortgage Payment`
- `Credit Card`
- `Salary Deposit`
- `Loan Payment`
- `Purchase`
- `Other`

Another step classified **stores/merchants** into top-level categories (`Groceries`, `Online`, `Hardware`, etc.) based on keywords (e.g., `Costco`, `Amazon`, `Ace Hardware`).

3 Visualizations and Screenshots

Figures 1 and 4 illustrate examples of plots and a Jupyter Notebook screenshot.

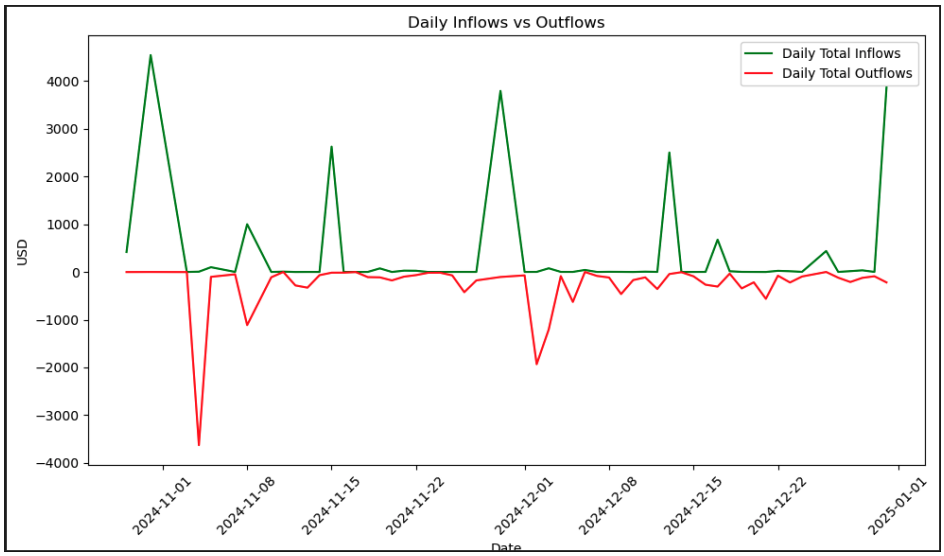

Figure 1: Sample Daily Inflows vs. Outflows Plot.

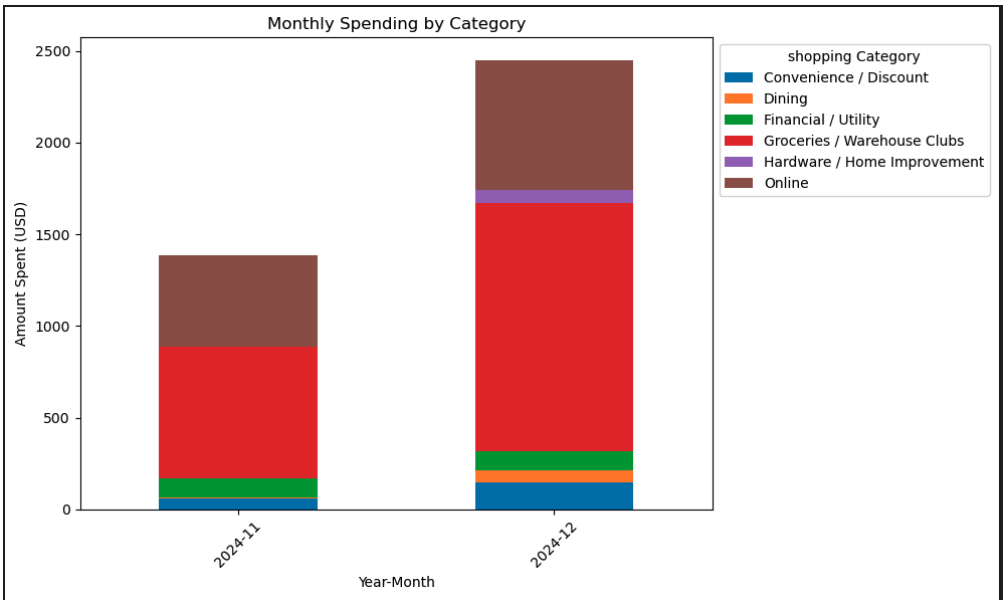

Figure 2: Spending Habit on Different Category.

4 Results and Answers to Key Questions

Based on the categorized data and summarized transactions, the following conclusions emerged:

1. **Mortgage?** The dataset contains references to `Mortgage Payment` in the description. Hence, the user does appear to have a mortgage.

Figure 3: A Jupyter Notebook Screenshot Demonstrating the Analysis.

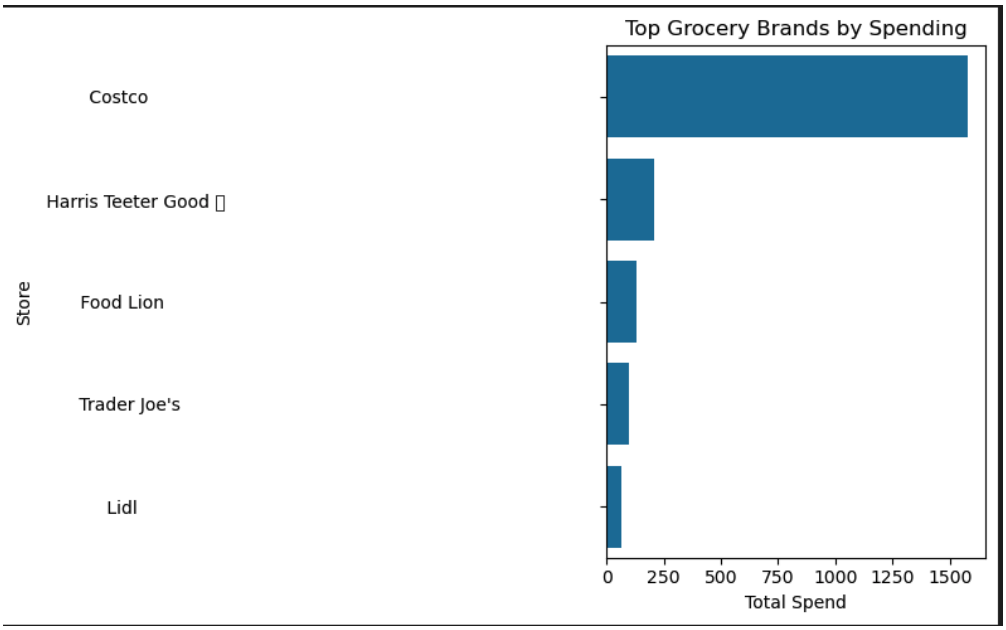

Figure 4: Most Popular Grocery Brands.

2. **Salary / Employment Status?** Regular large deposits were identified, suggesting the user is employed and receiving salary-type inflows.

3. **Approximate Salary Amount?** By calculating the mean of these recurring large deposits, the user's average net salary deposit is approximately `$2,174` per pay period (or whichever figure was computed).

4. **Ongoing Project?** Text searches for "project," "remodel," "renovation," etc. yielded no consistent pattern of repeated large outflows. Thus, no clear ongoing project is detected.

5. **Credit Cards?** Keywords like "credit card," or recognized card references (e.g., "CITI CARD ONLINE"), indicate the user has at least one active credit card.

6. **Phone Providers?** A search for common U.S. carriers (e.g., Verizon, T-Mobile, AT&T) identified two negative outflow entries referencing `VERIZON` in the ledger:

   - `#434027101367 | VERIZON BILL PAYMENT`

- `#435521103285 | VERIZON*RECURRING PAY`

Both transactions were outflows of $89.99 and $119.99, respectively. Therefore, the user's **average mobile bill** is approximately $104.99. Hence:

- **Phone Provider Found:** Verizon.
- **Average Monthly Bill:** $104.99.

4.1 Monthly Spending and Top Stores

Stacked bar charts show monthly spending by category (Groceries, Online, Hardware, etc.), and further breakdown identified "Costco" and "Harris Teeter Good" as major grocery/warehouse expenditures. The user heavily prefers Costco for grocery or bulk items, with other stores filling more specialized or smaller purchases. Stacked Bar Chart showed that the user spent more in December than in November (roughly a $900 increase). Groceries and online purchases consistently dominate the budget. Minor categories (dining, hardware, convenience stores) represent a smaller share but still contribute to overall expenses.

5 Conclusion

The analysis reveals consistent salary deposits, mortgage payments, active credit card usage, and a specific phone provider, but no evidence of an ongoing project. Additional insights include monthly category spending, top vendors, and daily inflow/outflow trends. Future improvements could explore the user's apparent one-dollar-per-day saving habit, examine recurring bills and debit payments in more depth, or adopt a more robust method to confirm whether the user's pay intervals are biweekly or monthly.

6 Final Code

Below is the complete Python script used. This code performs data loading, cleaning, categorization, searches for mortgage/credit card/phone providers, identifies salary patterns, and generates plots.

```
# %%
from IPython.display import display
import pandas as pd
import numpy as np
import re


#============================================================
# 1. LOAD & MERGE DATA
#============================================================

# Define file path
file_path = "Joined_database.xlsx"


# Read the Excel
df = pd.read_excel(file_path,engine="openpyxl", header=0)

# Inspect first few rows
print("=== HEAD OF DATAFRAME ===")
print(df.columns)
print(df.head(), "\n")

# %%
#============================================================
# 2. BASIC CLEANING & DATA PREP
#============================================================

# 2.1 Standardize column names
df.rename(columns={
```

```
    'Account ID':       'Account_ID',
    'Date':             'Date',
    'Amount In':        'AmountIn',
    'Amount Out':       'AmountOut',
    'Balance':          'Balance',
    'Type':             'Type',
    'Desc1':            'Description1',
    'Desc2':            'Description2',
    'Exif':             'Exit',
    'FI ID':            'FI_ID',
    'Number':           'Number',
    'Credit Limit':     'CreditLimit',
    'Interest Rate':    'InterestRate',
    'Joint':            'JointAccount',
    'Status':           'Status',
    'Open':             'OpenDate',
    'Closed':           'ClosedDate',
    'Known since':      'KnownSince',
    'Total Trxs':       'TotalTransactions',
    '2024-12':          'Balance_2024_12',
    '2024-11':          'Balance_2024_11',
    '2024-10':          'Balance_2024_10',
    '2024-09':          'Balance_2024_09',
    '2024-08':          'Balance_2024_08',
    '2024-07':          'Balance_2024_07'
}, inplace=True)



df.columns = df.columns.str.strip()



# print(df.head(), "\n")
# 2.2 Convert date column to datetime
if 'Date' in df.columns:
    df['Date'] = pd.to_datetime(df['Date'], errors='coerce')

# 2.3 Convert numeric columns from strings (if needed)
for col in ['AmountIn', 'AmountOut', 'Balance']:
    if col in df.columns:
            # Remove non-numeric characters and convert to float
            df[col] = pd.to_numeric(df[col].replace('[\$,]', '', regex=True), errors='coerce')
            # Replace NaN with 0
            df[col].fillna(0, inplace=True)


# # 2.4 Sort by date
df.sort_values(by='Date', inplace=True)

# Drop any rows that have invalid or missing date
df = df.dropna(subset=['Date'])

df['Date'] = pd.to_datetime(df['Date'], errors='coerce')
df.dropna(subset=['Date'], inplace=True)  # remove rows where date couldn't be parsed




# Inspect first few rows
print("=== HEAD OF Cleaned DATAFRAME ===")
# print(df.head(2), "\n")
print(df.columns)


# %%
```

```
#============================================================
# 3. HELPER FUNCTIONS & CATEGORIZATION
#============================================================
def categorize_transaction(row):
    """
    Categorize each transaction based on description or other fields.
    """
    desc = str(row.get('Desc1|Desc2', '')).lower()
    type_money = str(row.get('Type', '')).lower()
    exit_info = str(row.get('ExIf', '')).strip().lower()

    if 'purchase' in desc or 'purchase' in type_money:
        return 'Purchase'
    elif 'mortgage' in desc or 'mortgage' in exit_info:
        return 'Mortgage Payment'
    elif 'credit card' in desc or 'credit card' in exit_info:
        return 'Credit Card'
    elif 'loan' in desc or 'externalloan' in type_money:
        return 'Loan Payment'
    elif any(keyword in desc for keyword in ['deposit', 'payroll', 'salary','pay']):
        return 'Salary Deposit'
    else:
        return 'Other'


df['Category'] = df.apply(categorize_transaction, axis=1)
display(df)



# %%


# %%
#============================================================
# 4. CHECK FOR MORTGAGE
#============================================================
# Strategy:
#   - If there are explicit "Mortgage" strings in the description

# Identify rows that are labeled as 'Mortgage Payment' in Category
mortgage_payments = df[df['Category'] == 'Mortgage Payment']
display(mortgage_payments)

# %%
#============================================================
# 5. CHECK IF USER HAS A SALARY (BIWEEKLY DETECTION)
#============================================================




# 5.1 Parse the inflows data, focusing on deposits > 0
inflows = df[df['AmountIn'] > 0].copy()
# display(inflows)


# create a "Payor" column by extracting everything before the '|'.
def extract_payor(desc):
    if not isinstance(desc, str):
        return None
    parts = desc.split('|', maxsplit=1)
    return parts[0].strip()

inflows['Payor'] = inflows['Desc1|Desc2'].apply(extract_payor)

# 5.2 We'll define a threshold to consider "large deposit" as potential salary
MIN_SALARY_AMOUNT = 500  # e.g. £5,000 min average deposit
```

```
BIWEEKLY_TOLERANCE = 3    # +/- 3 days from 14

# We'll store all potential salaries in a list
potential_salaries = []

# Group by Payor, sort by Date, then check for average deposit
for payor, group in inflows.groupby('Payor'):
    group_sorted = group.sort_values(by='Date')
    # print(group_sorted)
    # Check the average deposit for that payor
    avg_deposit = group_sorted['AmountIn'].mean()
    if avg_deposit < MIN_SALARY_AMOUNT:
        continue  # skip if too small to be considered a salary


    total_received = group_sorted['AmountIn'].sum()
    num_payments = len(group_sorted)

    potential_salaries.append({
        'Payor': payor,
        'AvgDeposit': round(avg_deposit, 2),
        'NumPayments': num_payments,
        'TotalReceived': round(total_received, 2)
    })

# Convert potential_salaries to a DataFrame for display
df_salaries = pd.DataFrame(potential_salaries)

# 5.4 Determine if user has a salary
if not df_salaries.empty:
    has_salary = True
    # For demonstration, let's assume the "main salary" is the largest total received
    top_salary_source = df_salaries.loc[df_salaries['TotalReceived'].idxmax()]
    avg_salary = top_salary_source['AvgDeposit']

    print("=== POTENTIAL BIWEEKLY SALARY SOURCES DETECTED ===")
    display(df_salaries)
    print("\nLikely Salary Payor:", top_salary_source['Payor'])
    print("Average Salary Deposit: $", avg_salary)
else:
    has_salary = False
    avg_salary = 0
    print("No recurring biweekly high deposits found. User may not have a salary or the pattern is
    ↪  different.")



# %%

#============================================================
# 6. ONGOING PROJECT?
#============================================================
# We might guess a "project" if we see repeated outflows to the same vendor
# with a pattern like "Project," "Remodel," "Construction," "Kickstarter," etc.
# This is very data-dependent. We'll do a naive text search:
project_keywords = ['project', 'remodel', 'renovation', 'kickstarter', 'crowdfunding']
project_regex = '|'.join(project_keywords)

df['IsProject'] = df['Desc1|Desc2'].str.contains(project_regex, case=False, na=False)
has_project = df['IsProject'].any()
print(df['IsProject'] )

# %%
df_credit = df[df['Category']=='Credit Card']
display(df_credit)
```

```
def extract_name_before_card(text):
    """
    Returns the word/phrase immediately before "CARD".
    For example,
      "CITI CARD ONLINE|TYPE: PAYMENT..." -> "CITI"
    If nothing is found, returns None.
    """
    if not isinstance(text, str):
        return None

    # Regex explanation:
    # (.*?)  -> capture any characters (non-greedy)
    # \s+CARD -> that are followed by whitespace and then "CARD"
    # re.IGNORECASE makes "CARD" case-insensitive
    pattern = re.compile(r'(.*?)\s+CARD', re.IGNORECASE)
    match = pattern.search(text)

    if match:
        # group(1) is everything before the word "CARD"
        return match.group(1).strip()
    else:
        return None

print(df_credit['Desc1|Desc2'])
df_credit['ExtractedName'] = df_credit['Desc1|Desc2'].apply(extract_name_before_card)
display(df_credit[['Desc1|Desc2', 'ExtractedName']].head())

card_usage = df_credit.groupby('ExtractedName')['AmountOut'].mean()
print("Average Payment by Card:")
print(card_usage)



# %%
#============================================================
# 8. WHO IS THE MOBILE PHONE PROVIDER?
#============================================================
phone_providers = [
    'Verizon',
    'T-Mobile',
    'AT&T',
    'Sprint',
    'Boost',
    'Cricket',
    'U.S. Cellular',
    'Metro by T-Mobile',
    'Google Fi',
    'TracFone',
    'Straight Talk',
    'Xfinity Mobile',
    'Spectrum Mobile'
]

provider_regex = '|'.join(phone_providers)

# Mark rows that reference a known provider in Desc1|Desc2
df['IsPhoneBill'] = df['Desc1|Desc2'].str.contains(provider_regex, case=False, na=False)
possible_bills = df[df['IsPhoneBill'] == True]

# display(possible_bills)

if len(possible_bills) > 0:
    # Identify which providers appear, collecting them into a list
    user_providers = []
    for desc in possible_bills['Desc1|Desc2']:
        for p in phone_providers:
            if p.lower() in desc.lower():
                user_providers.append(p)
```

```
    user_providers = list(set(user_providers))  # get unique providers

    print("Providers found in the ledger: ", user_providers)

    # Now show a DataFrame of the phone-bill payments (assuming AmountOut is the payment)
    phone_payments = possible_bills[possible_bills['AmountOut'] < 0].copy()
    phone_payments['PaymentAmount'] = phone_payments['AmountOut'].abs()

    print("\n=== PHONE BILL PAYMENTS (Negative Outflows) ===")
    display(phone_payments[['Date', 'Desc1|Desc2', 'AmountOut', 'PaymentAmount']])
    mobile_avg_bill = phone_payments['PaymentAmount'].mean()
    print("Average Mobile Bill: ",mobile_avg_bill)
else:
    user_providers = []
    print("No phone providers found in the data.")

# %%
#================================================================
# 9. VISUALIZE
#================================================================
# plot total inflows and outflows over time

import matplotlib.pyplot as plt



plt.figure(figsize=(10,6))
df_daily = df.groupby('Date').agg({'AmountIn':'sum','AmountOut':'sum'}).reset_index()
plt.plot(df_daily['Date'], df_daily['AmountIn'], label='Daily Total Inflows', color='green')
plt.plot(df_daily['Date'], df_daily['AmountOut'], label='Daily Total Outflows', color='red')
plt.title('Daily Inflows vs Outflows')
plt.xlabel('Date')
plt.ylabel('USD')
plt.legend()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()



# %%
category_keywords = {
    'Groceries / Warehouse Clubs': [
        'harris teeter',
        'trader joe',
        'food lion',
        'lidl',
        'costco',
        'sam\'s club',
        'walmart'
    ],
    'Online': [
        'amazon',
        'apple',
        'best buy'
    ],
    'Hardware / Home Improvement': [
        'ace hardware',
        'lowe\'s'
    ],
    'Convenience / Discount': [
        '7-eleven',
        'family dollar',
        'dollar tree',
        'ollie\'s bargain'
    ],
    'Dining': [
```

```
        'blaze pizza',
        'first watch',
        'virginia abc'
    ],
    'Financial / Utility ': [
        'citi bank',        # credit card references
        'wells fargo',      # mortgage references
        'dominion energy',  # utility
        'gofundme'          # crowdfunding/donation
    ]
}

def assign_category(desc):
    if not isinstance(desc, str):
        return 'Other'
    desc_lower = desc.lower()
    for cat, keywords in category_keywords.items():
        for kw in keywords:
            if kw in desc_lower:
                return cat
    return 'Other'



# 2) APPLY TO DATAFRAME
# Assume df is your merged DataFrame
df['shopping_cat'] = df['Desc1|Desc2'].apply(assign_category)
df_spending = df[df['shopping_cat'] != 'Other']
df_spending['PaymentAmount'] = df_spending['AmountOut'].abs()

display(df)

# 3) MONTHLY SPENDING
df_spending['YearMonth'] = df_spending['Date'].dt.to_period('M')
monthly_spend = df_spending.groupby(['YearMonth','shopping_cat'], as_index=False)['PaymentAmount'].sum()
monthly_pivot = monthly_spend.pivot(index='YearMonth', columns='shopping_cat',
↪  values='PaymentAmount').fillna(0)

print("=== MONTHLY SPENDING BY CATEGORY ===")
print(monthly_pivot)

# 4) PLOT
monthly_pivot.plot(kind='bar', stacked=True, figsize=(10,6))
plt.title('Monthly Spending by Category')
plt.xlabel('Year-Month')
plt.ylabel('Amount Spent (USD)')
plt.xticks(rotation=45)
plt.legend(title='shopping Category', bbox_to_anchor=(1.0, 1.0))
plt.tight_layout()
plt.show()

# %%
import seaborn as sns



# 1) Filter rows whose shopping_cat is in grocery_stores
df_groceries = df[df['shopping_cat']=='Groceries / Warehouse Clubs']


# 2) Sum total spending for each grocery store
grocery_spend = (
    df_groceries
    .groupby('ExIf')['AmountOut']
    .sum()
    .abs()
    .sort_values(ascending=False)
)
```

```
print("=== Grocery Spend by Store ===")
print(grocery_spend)



# Plot top grocery brands
if not grocery_spend.empty:
    top_grocery = grocery_spend.head(5)  # top 5 stores if you like
    df_grocery_plot = top_grocery.reset_index()
    df_grocery_plot.columns = ['Store', 'Spending']
    plt.figure(figsize=(8, 5))
    ax = sns.barplot(
        y="Store",
        x="Spending",
        data=df_grocery_plot,
        orient='h'
    )
    plt.title("Top Grocery Brands by Spending")
    plt.xlabel("Total Spend")
    plt.ylabel("Store")
    plt.tight_layout()
    plt.show()
else:
    print("No grocery stores found in data.")
```