Bank Transaction Dashboard and Analytics

EDULYT INDIA

NANDIPATI KUSUMA

Intern Id : EI-3431

June 30, 2025

1 Project Overview :

This project has been implemented using **Python**, leveraging libraries such as **Pandas** for data manipulation ,analysis tasks and **Matplotlib** for data Visualizationand,**ipyWidgets** for creating drop-down and multiselects. The analysis is based on 8-sheets containing Transaction information,they are -

1. New_Account
2. New_Card
3. District
4. Loan
5. New_Client
6. New_Disposition
7. New_Transaction
8. Order

✓ The goal of this project is to clean, enrich, and analyze bank transaction data to uncover meaningful financial insights.

✓ By resolving missing values and standardizing fields like **k_symbol** and **operation**, the data becomes significantly more interpretable.

✓ It enables accurate tracking of transaction purposes—such as loan payments, pensions, or interest—helping analysts identify customer behavior.

✓ By creating a **Visual Dashboard**,it helps banks or financial analysts monitor trends, detect anomalies, and improve service targeting.

✓ The analysis was performed using the pandas library for efficient data manipulation and cleaning, while matplotlib was employed to visualize transaction trends and account behavior.

NOTE : The dataset primarily consists of financial transaction records, and some fields (such as k_symbol and operation) include entries in Czech, reflecting the origin of the data from Czech banking systems.

2 Sanity Checks-Data Cleaning :

To perform data cleaning –
1. Dropping out the Duplicate rows:

```
df_New_Transaction.drop_duplicates();
```

2. Treating missing values :
i. Finding out in which sheets ,missing values are there

```
def inspect_df(df, name):
  print(f"\n{name} ------------------")
  print("Missing values:\n", df.isnull().sum())
  print("\nData types:\n", df.dtypes)

# Inspect each DataFrame
inspect_df(df_New_Account, "New_Account")
inspect_df(df_New_Transaction, "New_Transaction")
inspect_df(df_New_Client, "New_Client")
inspect_df(df_New_Disposition, "New_Disposition")
inspect_df(df_New_Card, "New_Card")
inspect_df(df_District, "District")
inspect_df(df_Loan, "Loan")
inspect_df(df_Order, "Order")
```

output : Except New_Transaction and Orders ,remaining have zero(0) null values.

```
🔍 New_Transaction ---------------
Missing values:
 Unnamed: 0         0
account_id         0
date               0
type               0
operation     178663
amount             0
balance            0
k_symbol      478646
dtype: int64

Data types:
 Unnamed: 0      int64
account_id      int64
date            int64
type           object
operation      object
amount        float64
balance       float64
k_symbol       object
dtype: object
```

1

```
🔍 Order ------------------
Missing values:
 order_id         0
account_id       0
bank_to          0
account_to       0
amount           0
k_symbol      1379
dtype: int64

Data types:
 order_id        int64
account_id      int64
bank_to        object
account_to      int64
amount        float64
k_symbol       object
dtype: object
```

2

3.Filling 'operation' column with the value 'collection from another bank' ,when the operation is null and corresponding k_symbol is either **'OLD AGE PENSION'** or **'INTEREST CREDIT'**.

CODE :

```
df_New_Transaction.loc[
(df_New_Transaction['k_symbol'].isin(['OLD AGE PENSION', '
    INTEREST CREDITED'])) &
(df_New_Transaction['operation'].isna()),
'operation'] = 'COLLECTION FROM ANOTHER BANK'
```

OUTPUT :

```
🔍 New_Transaction ----------
Missing values:
 Unnamed: 0         0
account_id         0
date               0
type               0
operation          0
amount             0
balance            0
k_symbol      478646
dtype: int64

Data types:
 Unnamed: 0      int64
account_id      int64
date            int64
type           object
operation      object
amount        float64
balance       float64
k_symbol       object
dtype: object
```

4. filling out the null value for the k_symbol column ,with the help of k_symbol values in Order sheet ,by matching account-id and amount ,if there is no match found,treating it as "UNKNOWN".

CODE :

```
order_missing_k = df_Order[df_Order['k_symbol'].isna()].copy()

merged_order = order_missing_k.merge(
    df_New_Transaction[['account_id', 'amount', 'k_symbol']],
    on=['account_id', 'amount'],
    how='left',
    suffixes=('', '_txn')
)
for idx, row in merged_order.iterrows():
    if pd.notna(row['k_symbol_txn']):
        df_Order.loc[
            (df_Order['account_id'] == row['account_id']) &
            (df_Order['amount'] == row['amount']) &
            (df_Order['k_symbol'].isna()),
            'k_symbol'
        ] = row['k_symbol_txn']

df_Order['k_symbol'] = df_Order['k_symbol'].fillna('UNKNOWN')
```

5.Filling out the **k_symbol** column in **Order** ,with the help of values in k_symbol in Transactions,by matching account_id and amount. Result: NULL values count reduced from 1379 to 32 and ,those remaining 32 will be filled up with "UNKNOWN"

OUTPUT :

```
🔍 New_Transaction ----
Missing values:
 Unnamed: 0    0
account_id    0
date          0
type          0
operation     0
amount        0
balance       0
k_symbol      0
dtype: int64

Data types:
 Unnamed: 0      int64
account_id      int64
date            int64
type           object
operation      object
amount        float64
balance       float64
k_symbol       object
dtype: object
```

1

```
🔍 Order -----------------
Missing values:
 order_id      0
account_id    0
bank_to       0
account_to    0
amount        0
k_symbol      0
dtype: int64

Data types:
 order_id        int64
account_id      int64
bank_to        object
account_to      int64
amount        float64
k_symbol       object
dtype: object
```

2

3 TASKS

1. Creating dashboard for the accounts contained in the districts prague and morovia.
1.1(Account-Wise)

Explanation :

1. **Merging Data:** It merges account and district data to link each account with its district name (merged_df).

2. **get_credit_data():** Filters transactions of type "CREDIT" for accounts in the selected district and counts how many such transactions each account made.

3. **dashboard():** Takes a district name, gets credit data, and sets up an account selector to allow users to choose accounts interactively.

4. **plot_selected_accounts():** Plots a bar chart of credit transaction counts for selected accounts.

5. **User Interface:** Uses ipywidgets for dropdown and multi-select interaction, enabling dynamic visual exploration in Jupyter.

CODE :

```
    import pandas as pd
import matplotlib.pyplot as plt
from ipywidgets import interact, widgets, Layout
from IPython.display import display

# Rename district columns for consistency
df_District.rename(columns={"A1": "district_id", "A3": "
   district_name"}, inplace=True)

# Merge account with district to get district-wise account data
merged_df = pd.merge(df_New_Account, df_District, on="district_id")

# Normalize transaction types and district names
df_New_Transaction["type"] = df_New_Transaction["type"].str.strip().
   str.upper()
merged_df["district_name"] = merged_df["district_name"].str.strip()

# Function to get credit transaction data for a given district
def get_credit_data(district):
    district_accounts = merged_df[merged_df["district_name"].str.
       lower() == district.lower()]
    account_ids = district_accounts["account_id"]
    credit_data = df_New_Transaction[
        (df_New_Transaction["account_id"].isin(account_ids)) &
        (df_New_Transaction["type"] == "CREDIT")
    ]
    grouped = credit_data.groupby("account_id").size().reset_index(
       name="credit_transaction_count")
    grouped = grouped.sort_values(by="credit_transaction_count",
       ascending=False)
    return grouped, credit_data

# Dashboard function for interactive plotting
def dashboard(district_name):
    acc_df, full_info = get_credit_data(district_name)

    account_selector = widgets.SelectMultiple(
        options=acc_df["account_id"].astype(str),
        value=tuple(acc_df["account_id"].astype(str).tolist()[:10]),
        description="Accounts",
        layout=Layout(width="50%")
    )

    def plot_selected_accounts(selected_ids):
        filtered = acc_df[acc_df["account_id"].astype(str).isin(
           selected_ids)]
        if filtered.empty:
            print("No data for selected accounts.")
            return
        plt.figure(figsize=(10, 5))
        plt.bar(filtered["account_id"].astype(str), filtered["
```

```
            credit_transaction_count"], color="royalblue")
        plt.xlabel("Account ID")
        plt.ylabel("Credit Transaction Count")
        plt.title(f"Credit Transactions     {district_name}")
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.show()

    interact(plot_selected_accounts, selected_ids=account_selector)

# Dropdown for district selection
district_dropdown = widgets.Dropdown(
    options=['Prague', 'south Moravia', 'north Moravia'],
    description="District",
    layout=Layout(width="50%")
)

# Launch the dashboard
interact(dashboard, district_name=district_dropdown)
```

OUTPUT :

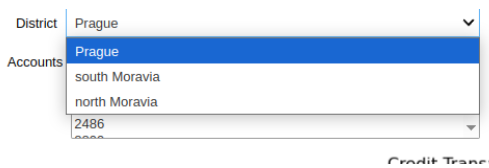


Figure 1: Drop-Down to select District

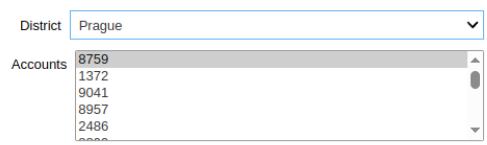


Figure 2: Drop-Down to select Account(s)

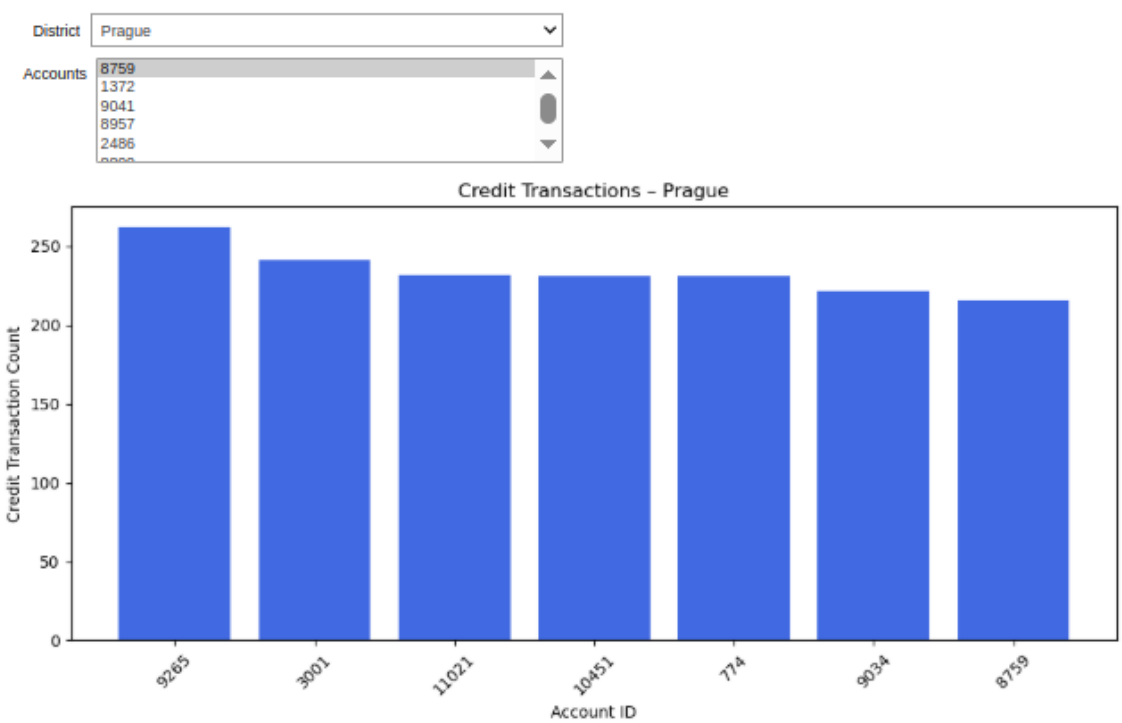


Figure 3: Bar-Chart showing credit-Transaction count by account-wise

1.2. (Month-Wise) **Explanation :**

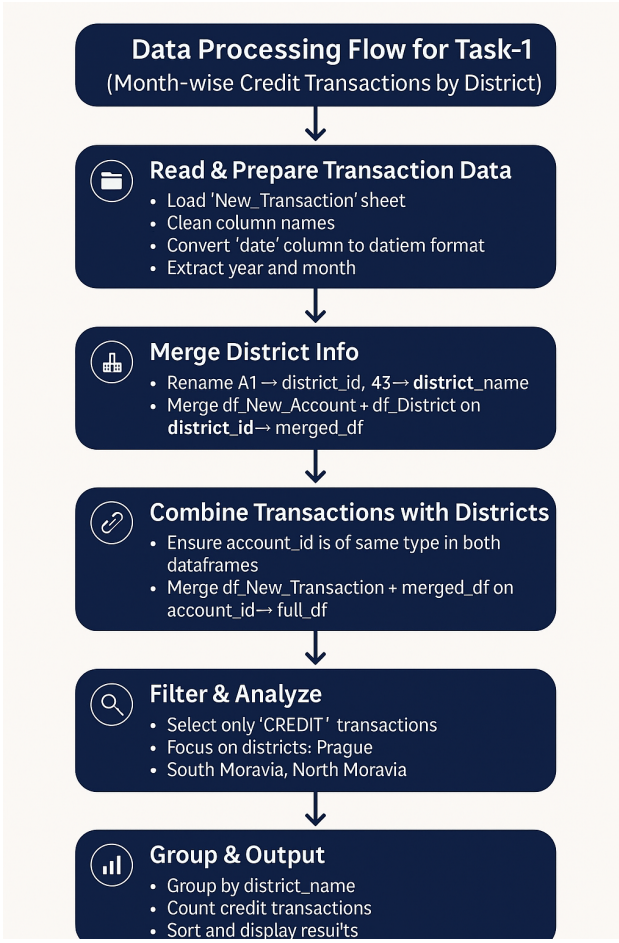


Figure 4: Flow-chart

Code :

```
import pandas as pd
import ipywidgets as widgets
from IPython.display import display, clear_output

# Load and prepare data
df_New_Transaction = pd.read_excel("data.xlsx", sheet_name="
    New_Transaction", dtype=str)
df_New_Account = pd.read_excel("data.xlsx", sheet_name="New_Account"
    )
df_District = pd.read_excel("data.xlsx", sheet_name="District")

df_New_Transaction.columns = df_New_Transaction.columns.str.strip()
df_New_Transaction["date"] = pd.to_datetime(df_New_Transaction["date
    "], format="%y%m%d", errors="coerce")
df_New_Transaction["year"] = df_New_Transaction["date"].dt.year
df_New_Transaction["month"] = df_New_Transaction["date"].dt.month

df_District.rename(columns={"A1": "district_id", "A3": "
    district_name"}, inplace=True)
merged_df = pd.merge(df_New_Account, df_District, on="district_id")
```

```python
df_New_Transaction["account_id"] = df_New_Transaction["account_id"].
    astype(int)
merged_df["account_id"] = merged_df["account_id"].astype(int)

full_df = pd.merge(df_New_Transaction, merged_df, on="account_id")

selected_districts = ["Prague", "south Moravia", "north Moravia"]

# Dropdowns for filtering
year_options = sorted(full_df["year"].dropna().unique())
month_options = sorted(full_df["month"].dropna().unique())

year_dropdown = widgets.Dropdown(options=year_options, description="
    Year:")
month_dropdown = widgets.Dropdown(options=month_options, description
    ="Month:")

output = widgets.Output()

def update_dashboard(change):
    with output:
        clear_output()
        year = year_dropdown.value
        month = month_dropdown.value

        filtered_df = full_df[
            (full_df["type"] == "CREDIT") &
            (full_df["district_name"].isin(selected_districts)) &
            (full_df["year"] == year) &
            (full_df["month"] == month)
        ]

        credit_by_district = (
            filtered_df.groupby("district_name")
            .size()
            .reset_index(name="credit_transaction_count")
            .sort_values(by="credit_transaction_count", ascending=
                False)
        )

        display(f"Credit transactions for {month}/{year}")
        display(credit_by_district)

year_dropdown.observe(update_dashboard, names="value")
month_dropdown.observe(update_dashboard, names="value")

display(widgets.HBox([year_dropdown, month_dropdown]))
display(output)

# Trigger display once at start
update_dashboard(None)
```

OUTPUT :

Figure 5: Drop-Down for selecting Year

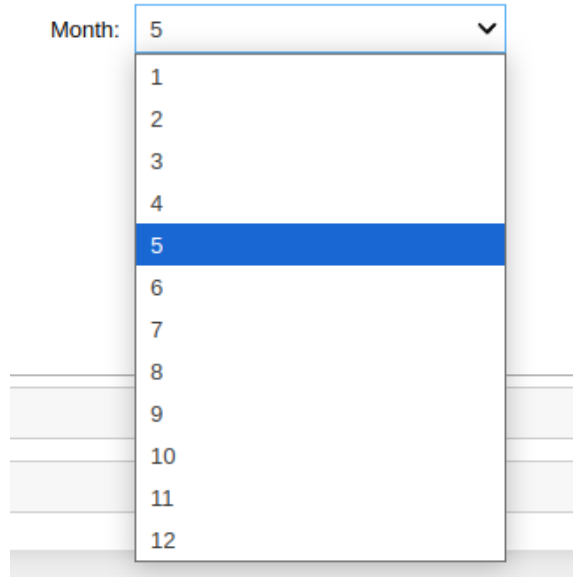
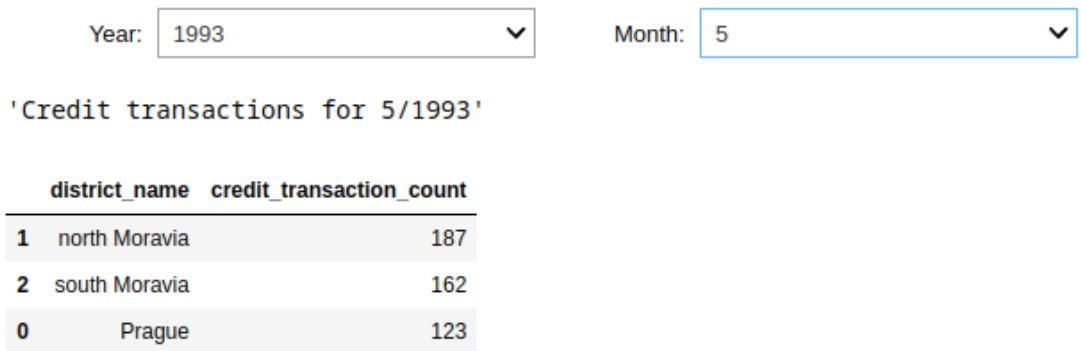


Figure 6: Drop-Down for selecting Month



	district_name	credit_transaction_count
1	north Moravia	187
2	south Moravia	162
0	Prague	123

Figure 7: Credit-Transaction-Count for District-wise, sorted by month-wise

1.3 (Account+Month Wise)

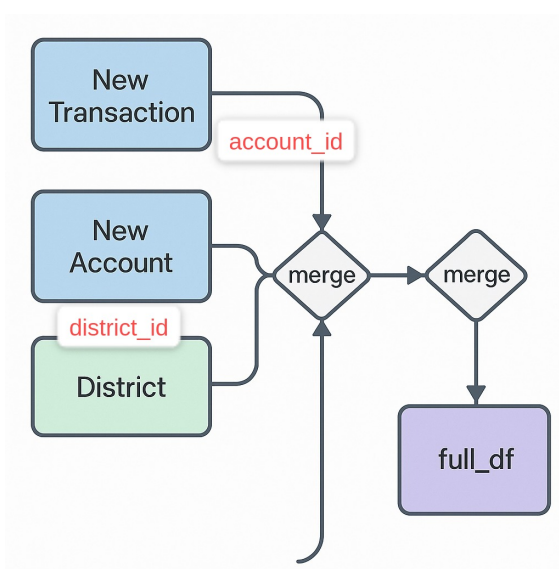


Figure 8: Flow-Chart

CODE :

```
    #task-1(acc+month)
import pandas as pd
import ipywidgets as widgets
from IPython.display import display, clear_output
import matplotlib.pyplot as plt

# Load and clean data
df_New_Transaction = pd.read_excel("data.xlsx", sheet_name="
    New_Transaction", dtype=str)
df_New_Account = pd.read_excel("data.xlsx", sheet_name="New_Account"
    )
df_District = pd.read_excel("data.xlsx", sheet_name="District")

df_New_Transaction.columns = df_New_Transaction.columns.str.strip()
df_New_Account.columns = df_New_Account.columns.str.strip()
df_District.columns = df_District.columns.str.strip()

df_New_Transaction["date"] = pd.to_datetime(df_New_Transaction["date
    "], format="%y%m%d", errors="coerce")
df_New_Transaction["year"] = df_New_Transaction["date"].dt.year
df_New_Transaction["month"] = df_New_Transaction["date"].dt.month

df_New_Transaction["account_id"] = df_New_Transaction["account_id"].
    astype(int)
df_New_Account["account_id"] = df_New_Account["account_id"].astype(
    int)
df_New_Account["district_id"] = df_New_Account["district_id"].astype
    (int)
df_District["A1"] = df_District["A1"].astype(int)
df_District.rename(columns={"A1": "district_id", "A3": "
    district_name"}, inplace=True)

merged_accounts = pd.merge(df_New_Account, df_District, on="
    district_id")
full_df = pd.merge(df_New_Transaction, merged_accounts, on="
    account_id")
full_df["district_name"] = full_df["district_name"].str.strip()
full_df["type"] = full_df["type"].str.strip()

# Widgets
years = sorted(full_df["year"].dropna().unique())
months = sorted(full_df["month"].dropna().unique())
districts = sorted(full_df["district_name"].dropna().unique())
selected_districts=['Prague','south Moravia','north Moravia']
year_dropdown = widgets.Dropdown(options=years, description="Year:")
month_dropdown = widgets.Dropdown(options=months, description="Month
    :")
district_dropdown = widgets.Dropdown(options=selected_districts,
    description="District:")
account_multiselect = widgets.SelectMultiple(options=[], description
    ="Accounts:", rows=10)

output = widgets.Output()

# Dynamic account filtering
```

```python
def update_account_options(*args):
    y = year_dropdown.value
    m = month_dropdown.value
    d = district_dropdown.value

    filtered_accounts = full_df[
        (full_df["year"] == y) &
        (full_df["month"] == m) &
        (full_df["district_name"] == d)
    ]["account_id"].unique()

    account_multiselect.options = sorted(filtered_accounts.tolist())

def update_plot(change=None):
    with output:
        clear_output()

        y = year_dropdown.value
        m = month_dropdown.value
        d = district_dropdown.value
        accs = list(account_multiselect.value)

        data = full_df[
            (full_df["year"] == y) &
            (full_df["month"] == m) &
            (full_df["district_name"] == d) &
            (full_df["account_id"].isin(accs)) &
            (full_df["type"] == "CREDIT")
        ]

        result = (
            data.groupby("account_id")
            .size()
            .reset_index(name="credit_transaction_count")
            .sort_values(by="credit_transaction_count", ascending=
               False)
        )

        if result.empty:
            print("No data for selected filters.")
        else:
            plt.figure(figsize=(10, 5))
            plt.bar(result["account_id"].astype(str), result["
               credit_transaction_count"], color="skyblue")
            plt.xlabel("Account ID")
            plt.ylabel("Credit Transaction Count")
            plt.title(f"CREDIT Transactions - {d}, {m}/{y}")
            plt.xticks(rotation=45)
            plt.grid(axis="y")
            plt.tight_layout()
            plt.show()

# Link filters
year_dropdown.observe(update_account_options, names="value")
month_dropdown.observe(update_account_options, names="value")
```

```
district_dropdown.observe(update_account_options, names="value")
account_multiselect.observe(update_plot, names="value")

# Display dashboard
display(widgets.HBox([year_dropdown, month_dropdown,
    district_dropdown]))
display(account_multiselect)
display(output)

# Initialize options and plot
update_account_options()
update_plot()
```

OUTPUT :

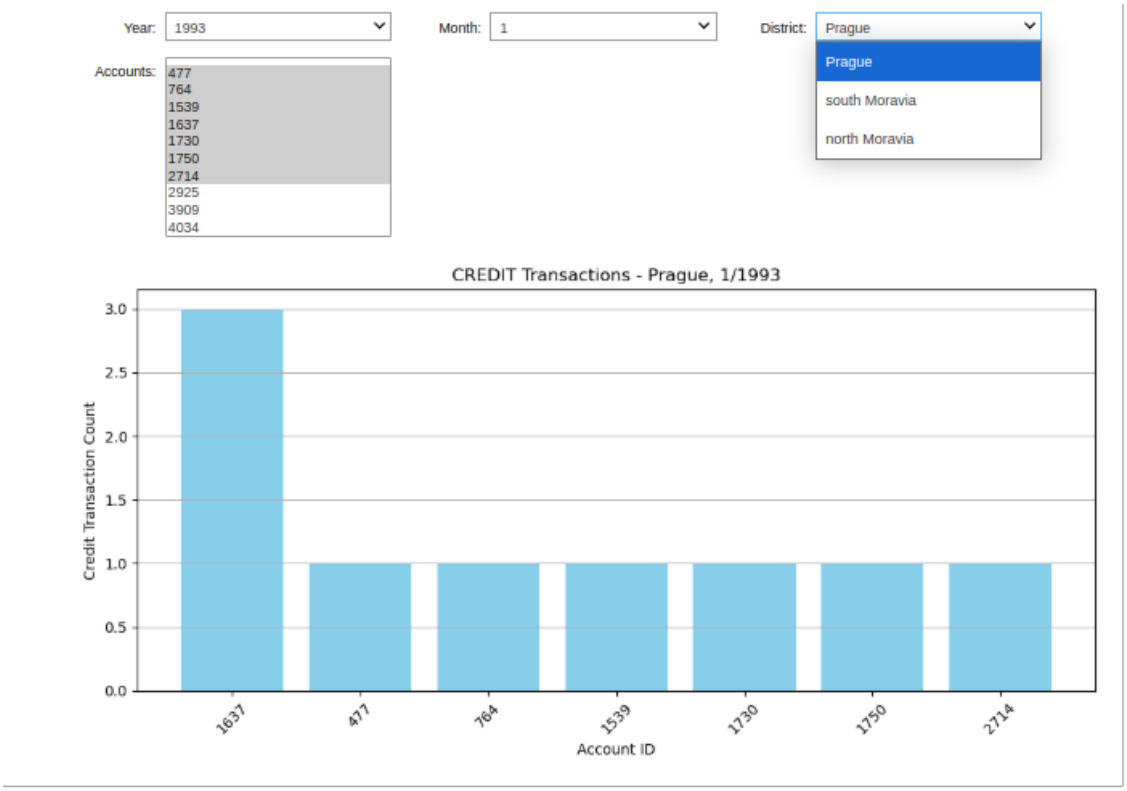


Figure 9: Credit-Transaction-Count by account and month wise.

2. Analysis on Highly populated versus Low populated districts. Find out the amount of Credit and Debit transaction from 5 highly populated and 5 lowest populated areas respectively, above analysis should be from last 3 months.

Explanation : This code merges account, district, and transaction data to prepare for visualizing credit transactions. Users can interactively filter by year, month, district, and account. Based on filters, it generates a bar chart showing the count of credit transactions per account. **CODE :**

```
import pandas as pd
import ipywidgets as widgets
from IPython.display import display, clear_output
import matplotlib.pyplot as plt

df_New_Transaction = pd.read_excel("data.xlsx", sheet_name="
    New_Transaction", dtype=str)
```

```
df_New_Account = pd.read_excel("data.xlsx", sheet_name="New_Account"
    )
df_District = pd.read_excel("data.xlsx", sheet_name="District")

df_New_Transaction.columns = df_New_Transaction.columns.str.strip()
df_New_Account.columns = df_New_Account.columns.str.strip()
df_District.columns = df_District.columns.str.strip()

df_New_Transaction["date"] = pd.to_datetime(df_New_Transaction["date
    "], format="%y%m%d", errors="coerce")
df_New_Transaction["year"] = df_New_Transaction["date"].dt.year
df_New_Transaction["month"] = df_New_Transaction["date"].dt.month

df_New_Transaction["account_id"] = df_New_Transaction["account_id"].
    astype(int)
df_New_Account["account_id"] = df_New_Account["account_id"].astype(
    int)
df_New_Account["district_id"] = df_New_Account["district_id"].astype
    (int)
df_District["A1"] = df_District["A1"].astype(int)
df_District.rename(columns={"A1": "district_id", "A3": "
    district_name"}, inplace=True)
merged_accounts = pd.merge(df_New_Account, df_District, on="
    district_id")
full_df = pd.merge(df_New_Transaction, merged_accounts, on="
    account_id")
full_df["district_name"] = full_df["district_name"].str.strip()
full_df["type"] = full_df["type"].str.strip()
years = sorted(full_df["year"].dropna().unique())
months = sorted(full_df["month"].dropna().unique())
districts = sorted(full_df["district_name"].dropna().unique())
selected_districts = ['Prague', 'south Moravia', 'north Moravia']
year_dropdown = widgets.Dropdown(options=years, description="Year:")
month_dropdown = widgets.Dropdown(options=months, description="Month
    :")
district_dropdown = widgets.Dropdown(options=selected_districts,
    description="District:")
account_multiselect = widgets.SelectMultiple(options=[], description
    ="Accounts:", rows=10)
output = widgets.Output()
def update_account_options(*args):
    y = year_dropdown.value
    m = month_dropdown.value
    d = district_dropdown.value
    filtered_accounts = full_df[
        (full_df["year"] == y) &
        (full_df["month"] == m) &
        (full_df["district_name"] == d)
    ]["account_id"].unique()
    account_multiselect.options = sorted(filtered_accounts.tolist())
def update_plot(change=None):
    with output:
        clear_output()
        y = year_dropdown.value
        m = month_dropdown.value
```

```
        d = district_dropdown.value
        accs = list(account_multiselect.value)
        data = full_df[
            (full_df["year"] == y) &
            (full_df["month"] == m) &
            (full_df["district_name"] == d) &
            (full_df["account_id"].isin(accs)) &
            (full_df["type"] == "CREDIT")]
        result = data.groupby("account_id").size().reset_index(name=
           "credit_transaction_count").sort_values(by="
           credit_transaction_count", ascending=False)
        if result.empty:
            print("No data for selected filters.")
        else:
            plt.figure(figsize=(10, 5))
            plt.bar(result["account_id"].astype(str), result["
               credit_transaction_count"], color="skyblue")
            plt.xlabel("Account ID")
            plt.ylabel("Credit Transaction Count")
            plt.title(f"CREDIT Transactions - {d}, {m}/{y}")
            plt.xticks(rotation=45)
            plt.grid(axis="y")
            plt.tight_layout()
            plt.show()
year_dropdown.observe(update_account_options, names="value")
month_dropdown.observe(update_account_options, names="value")
district_dropdown.observe(update_account_options, names="value")

account_multiselect.observe(update_plot, names="value")
display(widgets.HBox([year_dropdown, month_dropdown,
   district_dropdown]))

display(account_multiselect)
display(output)
update_account_options()
update_plot()
```

OUTPUT :

```
📊 Top 5 Most Populated Districts (Oct-Dec 1998):
type
CREDIT    16607836.2
DEBIT     14536180.2
Name: amount, dtype: float64

📉 Bottom 5 Least Populated Districts (Oct-Dec 1998):
type
CREDIT    47612835.5
DEBIT     42486323.5
Name: amount, dtype: float64
```

3. no.of cards issued to mid-age females

Explanation :

1. checking for duplicate card IDs in the card dataset to ensure uniqueness.
2. Merging df_New_Card, df_New_Disposition, and df_New_Client,to get all valid client details together.
3. Finally, filter this merged data to focus on female, middle-aged clients who hold valid (non-zero) cards and count how many such individuals exist, grouped by age level.

OUTPUT:

	age_levels	count
0	MIDDLE AGED	227

4. Number of cards issued in district, where average salary is more than 9000.

Explanation :

First, districts with an average salary above 9000 are selected from the df_District dataset. Then, the code merges card data with account and disposition info to connect each card to its district. Finally, it filters this combined data to include only high-salary districts and counts the number of cardholders per district, sorted in descending order.

CODE :

```
    #task-4
df_District.rename(columns={"A11":"avg_salary"},inplace=True)
district=df_District[(df_District["avg_salary"])>9000]
#print(district[["district_name","avg_salary","district_id"]])

merge_one=pd.merge(df_District,df_New_Account,on="district_id")
merge_two=pd.merge(merge_one,df_New_Disposition,on="account_id")
merge_final=pd.merge(merge_two,df_New_Card,on="disp_id")
#print(merge_final["district_name"])

data=merge_final[(merge_final["district_name"].isin(district["
   district_name"]))].groupby("district_name").size().reset_index(
   name="count").sort_values(by="count",ascending=False)
print(data)
```

OUTPUT :

```
          district_name  count
6           Hl.m. Praha    132
8               Karvina     24
16      Ostrava - mesto     22
26                 Zlin     17
17            Pardubice     17
11              Liberec     17
12           Litomerice     16
0           Brno - mesto     16
10                Kolin     15
5       Frydek - Mistek     15
25       Usti nad Labem     12
14       Mlada Boleslav     10
3         Cesky Krumlov     10
9                Kladno     10
18        Plzen - mesto     10
2      Ceske Budejovice      9
23                Tabor      9
4              Chomutov      9
24             Teplice      9
21  Rychnov nad Kneznou      9
15                 Most      8
20        Praha - zapad      8
22              Sokolov      8
13               Melnik      7
19       Praha - vychod      7
1            Ceska Lipa      6
7        Hradec Kralove      5
```

5. Are we providing loans to members belonging to district where committed crimes are more than 6000 for code 95, if yes then provide the number of loans per district?

ANS : Yes

```
df_District.rename(columns={"A15":"code95"},inplace=True)
df_District["code95"]=df_District["code95"].replace("?",0).astype(
    int)
district_names=df_District[(df_District["code95"]>6000)]
merged_o=pd.merge(df_District,df_New_Account,on="district_id")
merge_full=pd.merge(merged_o,df_Loan,on="account_id")
data=merge_full[(merge_full["district_name"].isin(district_names["
    district_name"]))].groupby("district_name").size().reset_index(
    name="count")
print(data.head(10))
```

OUTPUT :

```
      district_name  count
0      Brno - mesto     24
1  Ceske Budejovice      8
2       Hl.m. Praha     84
3           Karvina     24
4           Olomouc     14
5   Ostrava - mesto     20
6         Pardubice     10
7     Plzen - mesto      6
8           Teplice      6
9    Usti nad Labem      3
```

6. How much money was collected from other banks for customer belongs to districts where unemployment rate for any year is greater than 2%.

CODE & OUTPUT :

```
In [128]: df_District.rename(columns={"A12":"urcode95","A13":"urcode96"},inplace=True)
          unemployment_cols = ["urcode95","urcode96"]
          df_District[unemployment_cols] = df_District[unemployment_cols].replace("?", 0).astype(float)
          high_unemp_districts = df_District[df_District[unemployment_cols].gt(2).any(axis=1)]
          high_unemp_accounts=pd.merge(high_unemp_districts,df_New_Account,on="district_id")
          merged_col=pd.merge(high_unemp_accounts,df_New_Transaction,on="account_id")
          data=merged_col[(merged_col["operation"])=="COLLECTION FROM ANOTHER BANK"]
          data["amount"].sum()

Out[128]: 571773129.0
```

7. Create profile of customers in accordance of districts where max money is being paid to – a. Insurance. b. Household c. Leasing d. Loan **CODE :**

```
#task-7
import pandas as pd

# Step 1: Clean and prepare data
df_District.rename(columns={"A2": "district_name"}, inplace=True)

# Step 2: Merge transaction     account     district
txn_merge1 = pd.merge(df_New_Transaction, df_New_Account, on='
    account_id')
txn_merge2 = pd.merge(txn_merge1, df_District[['district_id', '
    district_name']], on='district_id')

# Step 3: Merge with disposition     client
txn_disp = pd.merge(txn_merge2, df_New_Disposition[['account_id', '
    client_id']], on='account_id')
full_txn = pd.merge(txn_disp, df_New_Client, on='client_id')

# Step 4: Analyze top-paying districts and custome profiles for each
     operation
operations = ['OLD AGE PENSION', 'HOUSEHOLD', 'INSURANCE PAYMENT', '
    LOAN PAYMENT']
results = {}

for op in operations:
    op_data = full_txn[full_txn['k_symbol'] == op]
    if op_data.empty:
        continue

    # Find district with max total amount
    district_sum = op_data.groupby('district_name')['amount'].sum().
        reset_index()
    max_row = district_sum.sort_values(by='amount', ascending=False)
        .iloc[0]
    max_district = max_row['district_name']
    max_amount = max_row['amount']

    # Filter customer profile from top district
    customer_data = op_data[op_data['district_name'] == max_district
        ]
    customer_profile = customer_data[['client_id', 'birth_number', '
        district_name', 'amount']]

    results[op] = {
        'district': max_district,
```

```
        'total_amount': max_amount,
        'customers': customer_profile
    }

# Step 5: Display result
for op, info in results.items():
    print(f"\n Operation: {op}")
    print(f"  Max Paying District: {info['district']}")
    print(f" Total Amount Paid: {info['total_amount']}")
    print(" Customers Profile:")
    print(info['customers'].tail(5))
```

OUTPUT :

```
📌 Operation: OLD AGE PENSION
🏙️  Max Paying District: Hl.m. Praha
💰 Total Amount Paid: 27306272.0
👥 Customers Profile:
        client_id  birth_number district_name  amount
254618        767        230616   Hl.m. Praha  4373.0
254622        767        230616   Hl.m. Praha  4373.0
254627        767        230616   Hl.m. Praha  4373.0
254632        767        230616   Hl.m. Praha  4373.0
254637        767        230616   Hl.m. Praha  4373.0

📌 Operation: HOUSEHOLD
🏙️  Max Paying District: Hl.m. Praha
💰 Total Amount Paid: 77802766.0
👥 Customers Profile:
        client_id  birth_number district_name  amount
255351       3966        691007   Hl.m. Praha  1722.0
255357       3966        691007   Hl.m. Praha  1722.0
255364       3966        691007   Hl.m. Praha  1722.0
255371       3966        691007   Hl.m. Praha  1722.0
255378       3966        691007   Hl.m. Praha  1722.0
```

Figure 10: 1

```
📌 Operation: INSURANCE PAYMENT
🏙️  Max Paying District: Hl.m. Praha
💰 Total Amount Paid: 6048829.0
👥 Customers Profile:
        client_id  birth_number district_name  amount
255353       3966        691007   Hl.m. Praha  1017.0
255360       3966        691007   Hl.m. Praha  1017.0
255367       3966        691007   Hl.m. Praha  1017.0
255374       3966        691007   Hl.m. Praha  1017.0
255382       3966        691007   Hl.m. Praha  1017.0

📌 Operation: LOAN PAYMENT
🏙️  Max Paying District: Hl.m. Praha
💰 Total Amount Paid: 9355987.6
👥 Customers Profile:
        client_id  birth_number district_name  amount
253808       5563        760101   Hl.m. Praha  2778.7
253817       5563        760101   Hl.m. Praha  2778.7
253825       5563        760101   Hl.m. Praha  2778.7
253836       5563        760101   Hl.m. Praha  2778.7
253852       5563        760101   Hl.m. Praha  2778.7
```

Figure 11: 2

8. Create profile of customers in accordance of districts for the status of loan payment, there will be 4 categories.(A,B,C,D).

Explanation :

- A' stands for contract finished, no problems.
- 'B' stands for contract finished, loan not payed.
- 'C' stands for running contract, OK so far.
- 'D' stands for running contract, client in debt.

CODE :

```
#task-8
import pandas as pd
import ipywidgets as widgets
from IPython.display import display

# Step 1: Rename for clarity (if not already done)
df_District.rename(columns={"A2": "district_name"}, inplace=True)

# Step 2: Merge data to get client details along with loan and
    district
merge1 = pd.merge(df_New_Account, df_District[['district_id', '
    district_name']], on='district_id')
merge2 = pd.merge(merge1, df_Loan, on='account_id')
```

```
merge3 = pd.merge(merge2, df_New_Disposition[['account_id', '
    client_id']], on='account_id')
full_data = pd.merge(merge3, df_New_Client[['client_id', '
    birth_number']], on='client_id')

# Step 3: Dropdown widgets
district_options = sorted(full_data['district_name'].unique())
status_options = sorted(full_data['status'].unique())

district_dropdown = widgets.Dropdown(
    options=district_options,
    description='District:',
    style={'description_width': 'initial'})
status_dropdown = widgets.Dropdown(
    options=status_options,
    description='Loan Status:',
    style={'description_width': 'initial'})
# Step 4: Filtering function
def filter_data(district, status):
    filtered = full_data[
        (full_data['district_name'] == district) &
        (full_data['status'] == status)]
    display(filtered[['client_id', 'birth_number', 'status', '
        district_name']])
# Step 5: Interactive output
ui = widgets.VBox([district_dropdown, status_dropdown])
out = widgets.interactive_output(
    filter_data,
    {'district': district_dropdown, 'status': status_dropdown})

display(ui, out)
```

OUTPUT :

District: Frydek - Mistek

Loan Status: B

	client_id	birth_number	status	district_name
332	2422	760609	B	Frydek - Mistek

9. Relate the output of above with district conditions like Crime, Unemployment Rate and Average Salary.

Logic :

```
district_info = df_District[df_District['district_name'] == district
    ][['district_name', 'code95', 'urcode95', 'urcode96', 'avg_salary
    ']]
    print("------ District Socio-economic Profile:")
    display(district_info)

    print("\n----------Client Loan Information:")
    display(filtered[['client_id', 'birth_number', 'status', 'amount
        ', 'payments', 'date']].head(10))
```

OUTPUT :

District: Benesov

Loan Status: A

```
------ District Socio-economic Profile:
```

	district_name	code95	urcode95	urcode96	avg_salary
1	Benesov	2159	1.67	1.85	8507.0

```
----------Client Loan Information:
```

	client_id	birth_number	status	amount	payments	date
703	5701	591122	A	107640	4485	950811

10. Owners from which district are issuing permanent orders and asking for a loan.

CODE :

```
owners=df_New_Disposition[df_New_Disposition["type"]=="OWNER"]
owners_with_orders=pd.merge(owners,df_Order,on="account_id")
owners_with_loans=pd.merge(owners_with_orders,df_Loan,on="account_id
    ")
account_merge=pd.merge(owners_with_loans,df_New_Account,on="
    account_id")
district_merge=pd.merge(account_merge,df_District,on="district_id")
data=district_merge.groupby("district_name").size().reset_index(name
    ="count")
data
```

OUTPUT :

Out[174]:

	district_name	count
0	Benesov	13
1	Beroun	9
2	Blansko	16
3	Breclav	21
4	Brno - mesto	47
...	...	...
72	Vsetin	13
73	Vyskov	18
74	Zdar nad Sazavou	17
75	Zlin	31
76	Znojmo	10

77 rows × 2 columns

11. Can we say customers from Bohemia are the ones having more male customers possessing Gold cards in comparison of Moravia?

Explanation : This task identifies male customers holding Gold cards from Bohemia and Moravia regions. After merging card, client, and district data, it filters for the required demographics and counts them per region. A final comparison prints which region has more male Gold card holders.

Result: Bohemia have more male customers possessing Gold cards in comparison of Moravia.

CODE:

```
#task11
merge=pd.merge(df_New_Card,df_New_Disposition,on="disp_id")
merge_two=pd.merge(acc_gold_cards,df_New_Client,on="client_id")
merge_full=pd.merge(merge_two,df_District[["district_id","A3"]],on="
    district_id")
districts_Bohemia=['central Bohemia', 'south Bohemia', 'west Bohemia
    ',
        'north Bohemia', 'east Bohemia']
districts_Moravia=['south Moravia','north Moravia"]
data1=merge_full[(merge_full["A3"].isin(districts_Bohemia)) &(
    merge_full["type_x"]=="GOLD")&(merge_full["gender"]=="MALE")].
    groupby("A3").size().reset_index(name="count")
data2=merge_full[(merge_full["A3"].isin(districts_Moravia)) &(
    merge_full["type_x"]=="GOLD")&(merge_full["gender"]=="MALE")].
    groupby("A3").size().reset_index(name="count")
Bohemia_count=data1["count"].sum()
Moravia_count=data2["count"].sum()
print("count of Bohemia male customers who are having gold card--->"
    ,Bohemia_count)
print("count of Moravia male customers who are having gold card--->"
    ,Moravia_count)
if Bohemia_count >Moravia_count:
    print("Bohemia have more male customers possessing Gold cards in
        comparison of Moravia.")
else:
    print("Moravia have more male customers possessing Gold cards in
        comparison of Bohemia.")
```

OUTPUT:

```
count of Bohemia male customers who are having gold card---> 28
count of Moravia male customers who are having gold card---> 18
Bohemia have more male customers possessing Gold cards in comparison of Moravia.
```

12. How many customers having credit card are also availing the loan facilities.

CODE :

```
card_disp = pd.merge(df_New_Card, df_New_Disposition, on="disp_id")
card_loan = pd.merge(card_disp, df_Loan, on="account_id")
customers_with_card_and_loan = card_loan["client_id"].nunique()
print("Number of customers having credit card and availing loan
    facilities:", customers_with_card_and_loan)
```

OUTPUT :

```
Number of customers having credit card and availing loan facilities: 170
```

13. Can we say that customers having Classic and Junior card are the ones who are more in debt.

ANS :FALSE

CODE & OUTPUT :

```
merge1=pd.merge(df_New_Card,df_New_Disposition[["disp_id","account_id"]],on="disp_id")
merge2=pd.merge(merge1,df_Loan[["account_id","status"]],on="account_id")
cards=["CLASSIC","JUNIOR"]
filter_data=merge2[merge2["type"].str.upper().isin([c.upper() for c in cards])]
all_defaulted=filter_data["status"].eq("D").all()
print("YES" if all_defaulted else "FALSE")
```

```
FALSE
```

14. How will you analyze the performance of Mid age vs adults in terms of loan repayments.

CODE & OUTPUT :

```
In [254]: owners = df_New_Disposition[df_New_Disposition["type"] == "OWNER"]
owners_clients = pd.merge(owners, df_New_Client[["client_id", "age_levels"]], on="client_id")
owners_loans = pd.merge(owners_clients, df_Loan[["account_id", "status"]], on="account_id")
loan_analysis = owners_loans.groupby(["age_levels", "status"]).size().reset_index(name="count")
loan_analysis
```

Out[254]:

	age_levels	status	count
0	ADULT	A	51
1	ADULT	B	4
2	ADULT	C	109
3	ADULT	D	13
4	MIDDLE AGED	A	128
5	MIDDLE AGED	B	22
6	MIDDLE AGED	C	246
7	MIDDLE AGED	D	25

📌 Conclusion :

This project analyzed banking data to uncover regional and demographic patterns. Key findings include higher Gold card ownership among Bohemian males and a positive link between high district salaries and card usage. The insights support targeted banking strategies across Czech regions.