Bank Details Dataset

Author: Dhaarani Shanmugam
Student ID: 23035833

---

1. Introduction

The procedures used to build a realistic SQLite database with financial data for a working banking system are described in this article. The three main tables in the database are Loan_Info, Account_Info, and Customer_Info. Foreign keys, compound keys, and a variety of data types (nominal, ordinal, interval, and ratio) are used in the design of these tables in order to create associations.

2. Data Generation

- [ ] **Tools used**
    - Python libraries: numpy,pandas,Faker,Random
    - SQLite for database creation and management.
- [ ] **Key Features of Data Generation**

Customer Data:

a. Created with Faker to provide distinct customer IDs, names, and addresses.
b. Birth dates were computed using randomly selected, normally distributed ages.
c. In certain columns, missing values were added to replicate real-world data.

Account Data:

d. Account types are assigned at random using a weighted probability.
e. A normal distribution was used to calculate the account balances.
f. There were three categories for account status: Active, Closed, and Suspended.
g. Postcodes were given their own branch codes.

Loan Data:

h. Age groups determined the loan type, interest rate, and period.
i. Depending on the type of loan, loan amounts fell within particular ranges.

3. Database Schema

Entity Relationship Diagram (ERD):

```
Customer_info(Customer_id PRIMARY KEY)
   |
   |-->Account_info(Account_id PRIMARY KEY,customer_id FOREIGN KEY)
           |
            |-->Loan_info(Loan_id PRIMARY KEY,customer_id FOREIGN KEY)
```

☐ Customer Information:

S.NO	COLUMN NAME	TYPE OF DATA	KEY TYPE	DESCRIPTION
1	**Customer_id**	Nominal	Primary Key	Unique identifier for each customer
2	**First_name**	Nominal	Candidate Key	Customer's first Name
3	**Last_name**	Nominal	Candidate Key	Customer's last name
4	**Age**	Ratio	-	Numeric age of the customer
5	**Dob**	Interval	-	Date of birth of customer
6	**Phone number**	Nominal	Alternate Key	Contact number of customers
7	E-mail	Nominal	Unique Key	E-mail address of customers
8	Address	Nominal	-	Customers address
9	**Postcode**	Nominal	-	Post code of customers

☐ Account Information:

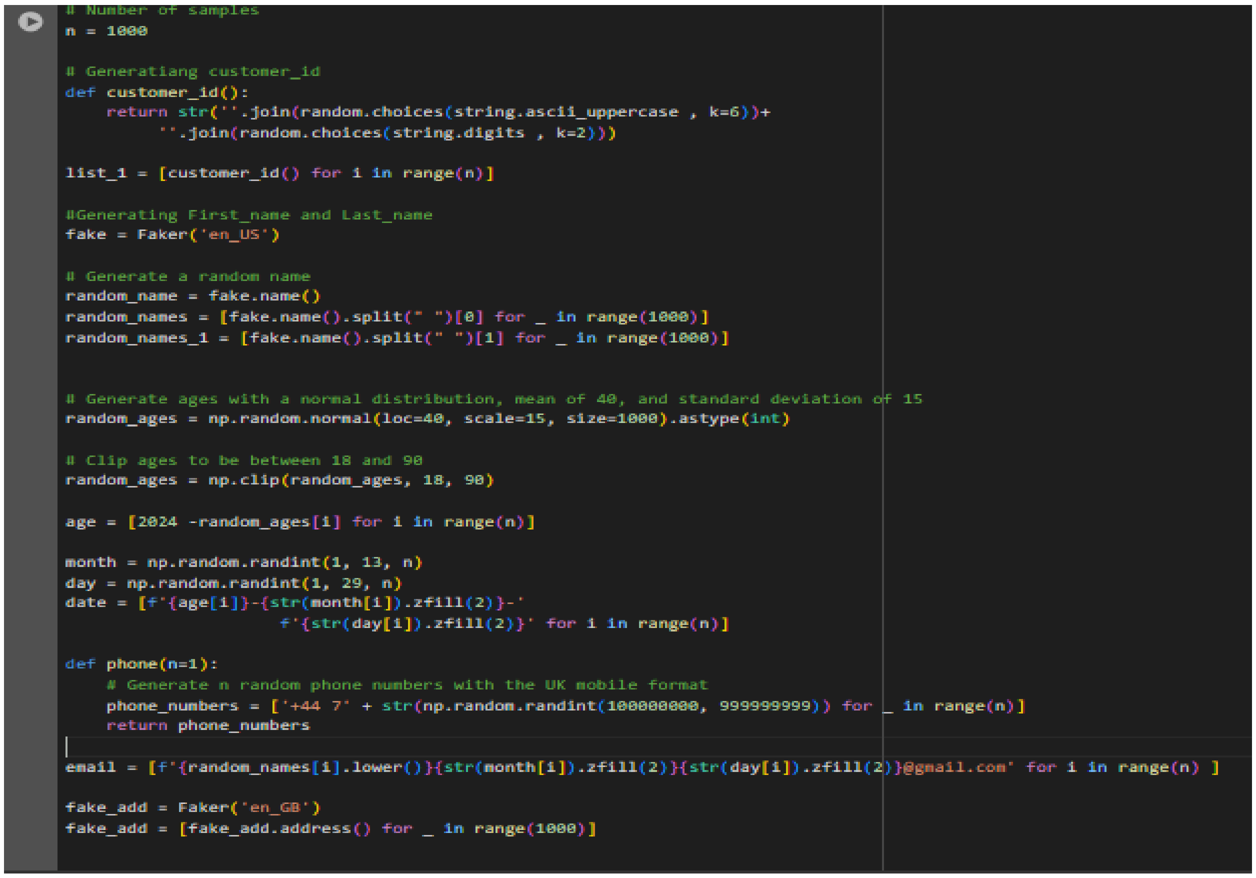
S.NO	COLUMN NAME	TYPE OF DATA	KEY TYPE	DESCRIPTION
1	**Account id**	Nominal	Primary Key	Unique identifier for accounts
2	**Customer Id**	Nominal	Foreign Key	Links account to a customer
3	**Account Type**	Ordinal	-	Categorized as type of Account
4	**Account Balance**	Ratio	-	Numeric value for account balance
5	**Opened Date**	Interval	-	Date the account was opened
6	**Branch code**	Nominal	-	Branch Location code
7	Status	Nominal	-	Current account status

☐ Loan details Information:

S.NO	COLUMN NAME	TYPE OF DATA	KEY TYPE	DESCRIPTION
1	**Loan id**	Nominal	Primary Key	Unique identifier for loan
2	**Customer Id**	Nominal	Foreign Key	Links loan to a customer
3	**Loan Type**	Ordinal	-	Categorized as loan type
4	**Interest Rate**	Ratio	-	Numeric value interest rate
5	**Loan Amount**	Ratio	-	Amount of loan issued
6	**Term in years**	Ratio	-	Loan repayment duration

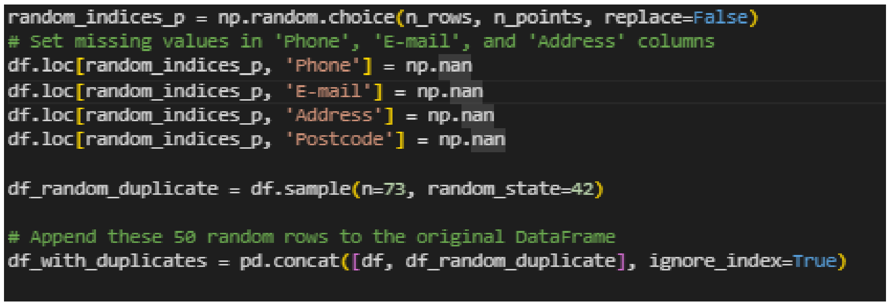
5.Explanation of code

- ☐ <u>**Code for customer table:**</u>

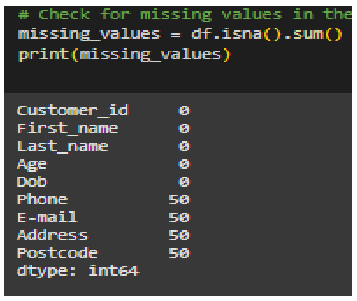


```python
# Number of samples
n = 1000

# Generatiang customer_id
def customer_id():
    return str(''.join(random.choices(string.ascii_uppercase , k=6))+
          ''.join(random.choices(string.digits , k=2)))

list_1 = [customer_id() for i in range(n)]

#Generating First_name and Last_name
fake = Faker('en_US')

# Generate a random name
random_name = fake.name()
random_names = [fake.name().split(" ")[0] for _ in range(1000)]
random_names_1 = [fake.name().split(" ")[1] for _ in range(1000)]


# Generate ages with a normal distribution, mean of 40, and standard deviation of 15
random_ages = np.random.normal(loc=40, scale=15, size=1000).astype(int)

# Clip ages to be between 18 and 90
random_ages = np.clip(random_ages, 18, 90)

age = [2024 -random_ages[i] for i in range(n)]

month = np.random.randint(1, 13, n)
day = np.random.randint(1, 29, n)
date = [f'{age[i]}-{str(month[i]).zfill(2)}-'
                     f'{str(day[i]).zfill(2)}' for i in range(n)]

def phone(n=1):
    # Generate n random phone numbers with the UK mobile format
    phone_numbers = ['+44 7' + str(np.random.randint(100000000, 999999999)) for _ in range(n)]
    return phone_numbers

email = [f'{random_names[i].lower()}{str(month[i]).zfill(2)}{str(day[i]).zfill(2)}@gmail.com' for i in range(n) ]

fake_add = Faker('en_GB')
fake_add = [fake_add.address() for _ in range(1000)]
```

The dataset will consist of 1,000 customers
Customer_id:function creates a unique customer ID by concatenating 6 random uppercase letters and 2 random digits .
First & Last name : Faker Library is used to create first and last names of customers.
Age and D.O.B : Generating random ages of customer between 18 to 90 years using normal distribution , and birth date is calculated from age of customers with random date and month.
Phone number :Phone numbers were generated as random 10-digit numbers with a "+44" country code prefix.
Email : Email IDs were created by combining the first name with padded month and date values.
Address : Faker library was utilized to generate realistic UK addresses.
The following code snippet generates random duplicate and null values, simulating situations where not every individual has a unique phone number, email address, home address, or postcode. It uses a set of 50 random numbers to introduce null values in these fields, reflecting real-world data inconsistencies and 73 duplicate values are introduced into the data set to make it so realistic.

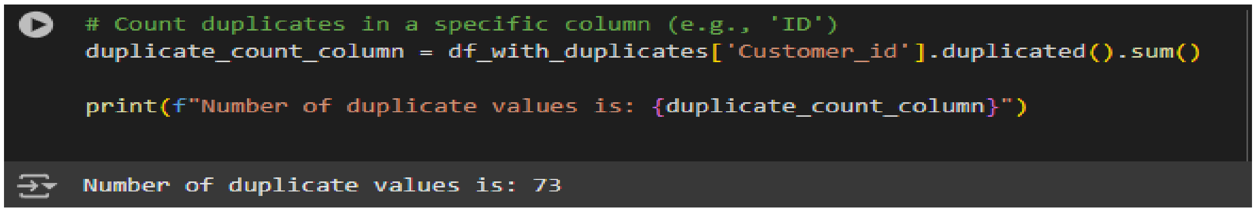


```python
random_indices_p = np.random.choice(n_rows, n_points, replace=False)
# Set missing values in 'Phone', 'E-mail', and 'Address' columns
df.loc[random_indices_p, 'Phone'] = np.nan
df.loc[random_indices_p, 'E-mail'] = np.nan
df.loc[random_indices_p, 'Address'] = np.nan
df.loc[random_indices_p, 'Postcode'] = np.nan

df_random_duplicate = df.sample(n=73, random_state=42)

# Append these 50 random rows to the original DataFrame
df_with_duplicates = pd.concat([df, df_random_duplicate], ignore_index=True)
```



```python
# Check for missing values in the
missing_values = df.isna().sum()
print(missing_values)
```

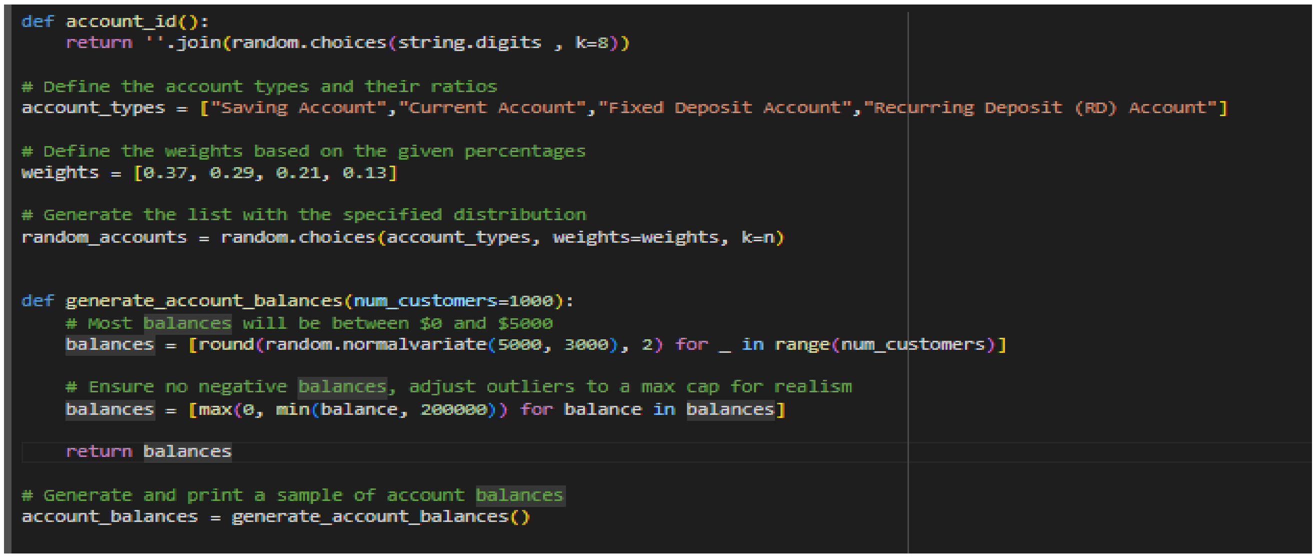
```
Customer_id     0
First_name      0
Last_name       0
Age             0
Dob             0
Phone          50
E-mail         50
Address        50
Postcode       50
dtype: int64
```



```
# Count duplicates in a specific column (e.g., 'ID')
duplicate_count_column = df_with_duplicates['Customer_id'].duplicated().sum()

print(f"Number of duplicate values is: {duplicate_count_column}")
```

```
Number of duplicate values is: 73
```

☐ Code for Account details table:



```
def account_id():
    return ''.join(random.choices(string.digits , k=8))

# Define the account types and their ratios
account_types = ["Saving Account","Current Account","Fixed Deposit Account","Recurring Deposit (RD) Account"]

# Define the weights based on the given percentages
weights = [0.37, 0.29, 0.21, 0.13]

# Generate the list with the specified distribution
random_accounts = random.choices(account_types, weights=weights, k=n)


def generate_account_balances(num_customers=1000):
    # Most balances will be between $0 and $5000
    balances = [round(random.normalvariate(5000, 3000), 2) for _ in range(num_customers)]

    # Ensure no negative balances, adjust outliers to a max cap for realism
    balances = [max(0, min(balance, 200000)) for balance in balances]

    return balances

# Generate and print a sample of account balances
account_balances = generate_account_balances()
```

Account_id: Function creates a unique account ID by generating 8 random digits.
Account_type:Four types of accounts are predefined.**Weights**: Probabilities for assigning each type are

- Saving Account: 37%.
- Current Account: 29%.
- Fixed Deposit Account: 21%.
- Recurring Deposit: 13%.

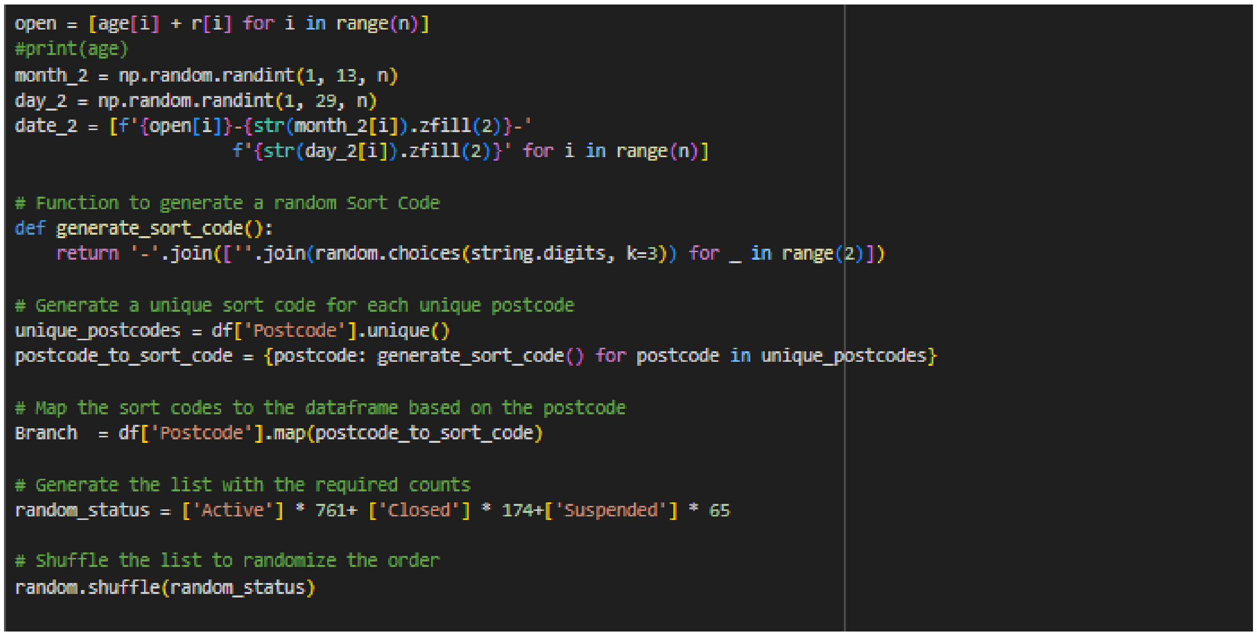
random.choices generates a list of account types for 1,000 accounts based on these probabilities.

Account_balances: Balances follow a normal distribution with a mean of $5,000 and a standard deviation of $3,000.**Adjustments** made here are ,Balances below $0 are set to $0.Balances above $200,000 are capped.**Open_Date:**The account creation date was generated by adding random years between the age of 18 and the customer's current age to their year of birth, along with randomly assigned month and day values.

Branch_code: For generation of sort code each unique postcode is mapped to a unique sort code in XXX-XXX format.
Status:Account status distribution

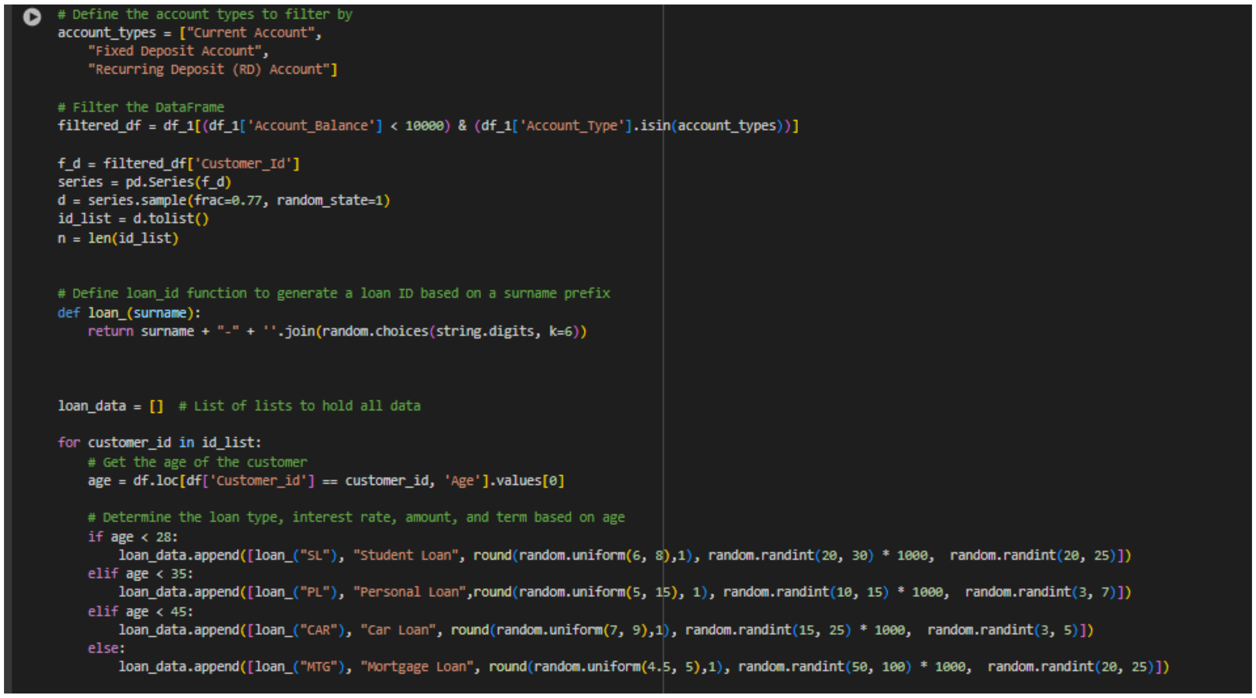
- Active: 76.1%.
- Closed: 17.4%.
- Suspended: 6.5%.

All values are shuffled and randomly inserted into the dataframe all the list is used to create Account details dataset.All lists are then inserted into a dictionary and the dictionary is converted into a data frame using the pd.DataFrame function.



```python
open = [age[i] + r[i] for i in range(n)]
#print(age)
month_2 = np.random.randint(1, 13, n)
day_2 = np.random.randint(1, 29, n)
date_2 = [f'{open[i]}-{str(month_2[i]).zfill(2)}-'
                     f'{str(day_2[i]).zfill(2)}' for i in range(n)]

# Function to generate a random Sort Code
def generate_sort_code():
    return '-'.join([''.join(random.choices(string.digits, k=3)) for _ in range(2)])

# Generate a unique sort code for each unique postcode
unique_postcodes = df['Postcode'].unique()
postcode_to_sort_code = {postcode: generate_sort_code() for postcode in unique_postcodes}

# Map the sort codes to the dataframe based on the postcode
Branch  = df['Postcode'].map(postcode_to_sort_code)

# Generate the list with the required counts
random_status = ['Active'] * 761+ ['Closed'] * 174+['Suspended'] * 65

# Shuffle the list to randomize the order
random.shuffle(random_status)
```

☐ Code for Loan details table



```python
# Define the account types to filter by
account_types = ["Current Account",
    "Fixed Deposit Account",
    "Recurring Deposit (RD) Account"]

# Filter the DataFrame
filtered_df = df_1[(df_1['Account_Balance'] < 10000) & (df_1['Account_Type'].isin(account_types))]

f_d = filtered_df['Customer_Id']
series = pd.Series(f_d)
d = series.sample(frac=0.77, random_state=1)
id_list = d.tolist()
n = len(id_list)


# Define loan_id function to generate a loan ID based on a surname prefix
def loan_(surname):
    return surname + "-" + ''.join(random.choices(string.digits, k=6))



loan_data = []  # List of lists to hold all data

for customer_id in id_list:
    # Get the age of the customer
    age = df.loc[df['Customer_id'] == customer_id, 'Age'].values[0]

    # Determine the loan type, interest rate, amount, and term based on age
    if age < 28:
        loan_data.append([loan_("SL"), "Student Loan", round(random.uniform(6, 8),1), random.randint(20, 30) * 1000,  random.randint(20, 25)])
    elif age < 35:
        loan_data.append([loan_("PL"), "Personal Loan",round(random.uniform(5, 15), 1), random.randint(10, 15) * 1000,  random.randint(3, 7)])
    elif age < 45:
        loan_data.append([loan_("CAR"), "Car Loan", round(random.uniform(7, 9),1), random.randint(15, 25) * 1000,  random.randint(3, 5)])
    else:
        loan_data.append([loan_("MTG"), "Mortgage Loan", round(random.uniform(4.5, 5),1), random.randint(50, 100) * 1000,  random.randint(20, 25)])
```

For loan data set accounts with account type “Current Account”, “FD account”, “RD account” and account balance less than 10,000 are taken into consideration.

For Each Selected Customer (id_list):

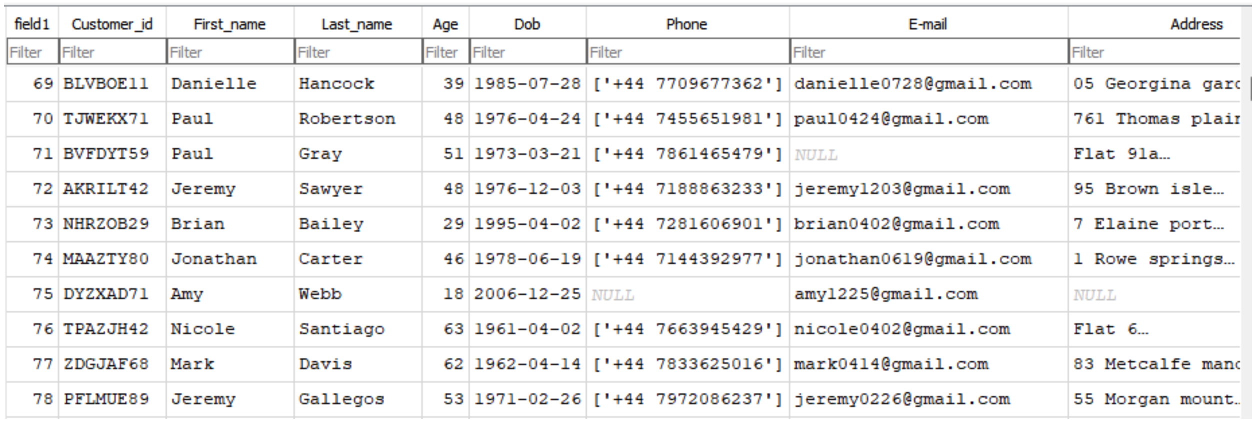
- Retrieves the **age** of the customer from df.
- Determines loan details based on **age group**:
  - **< 28 (Student Loan)**:
    - Interest rate: 6–8%.
    - Loan ID suffix - “SL”

        - Loan amount: £20,000–£30,000.
        - Term: 20–25 years.
    - **28–35 (Personal Loan)**:
        - Interest rate: 5–15%.
        - Loan ID suffix - “PL”
        - Loan amount: £10,000–£15,000.
        - Term: 3–7 years.
    - **35–45 (Car Loan)**:
        - Interest rate: 7–9%.
        - Loan ID suffix - “CAR”
        - Loan amount: £15,000–£25,000.
        - Term: 3–5 years.
    - **45+ (Mortgage Loan)**:
        - Interest rate: 4.5–5%.
        - Loan ID suffix - “MTG”
        - Loan amount: £50,000–£100,000.
        - Term: 20–25 years.

All lists are inserted into a dictionary and a data frame is formed using the pd.DataFrame function .

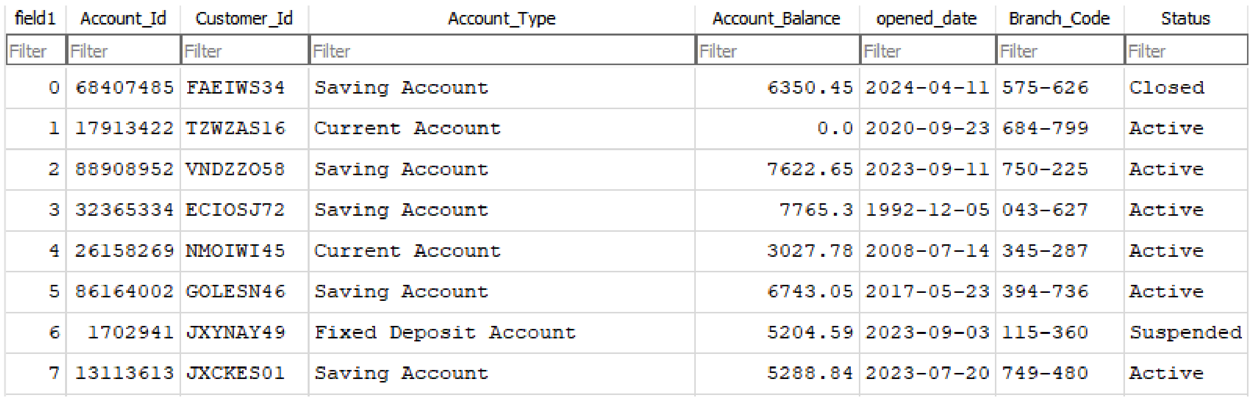
6.Output Tables

Customer details:



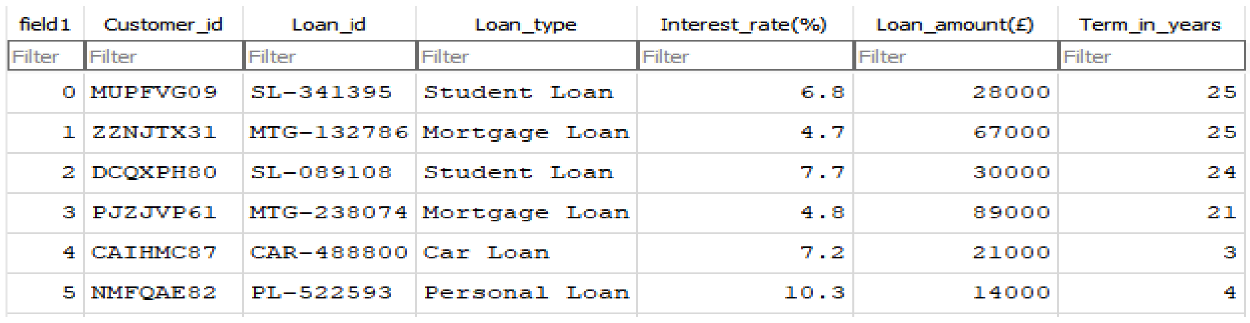
field1	Customer_id	First_name	Last_name	Age	Dob	Phone	E-mail	Address
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
69	BLVBOE11	Danielle	Hancock	39	1985-07-28	['+44 7709677362']	danielle0728@gmail.com	05 Georgina garc
70	TJWEKX71	Paul	Robertson	48	1976-04-24	['+44 7455651981']	paul0424@gmail.com	761 Thomas plair
71	BVFDYT59	Paul	Gray	51	1973-03-21	['+44 7861465479']	NULL	Flat 91a…
72	AKRILT42	Jeremy	Sawyer	48	1976-12-03	['+44 7188863233']	jeremy1203@gmail.com	95 Brown isle…
73	NHRZOB29	Brian	Bailey	29	1995-04-02	['+44 7281606901']	brian0402@gmail.com	7 Elaine port…
74	MAAZTY80	Jonathan	Carter	46	1978-06-19	['+44 7144392977']	jonathan0619@gmail.com	1 Rowe springs…
75	DYZXAD71	Amy	Webb	18	2006-12-25	NULL	amy1225@gmail.com	NULL
76	TPAZJH42	Nicole	Santiago	63	1961-04-02	['+44 7663945429']	nicole0402@gmail.com	Flat 6…
77	ZDGJAF68	Mark	Davis	62	1962-04-14	['+44 7833625016']	mark0414@gmail.com	83 Metcalfe manc
78	PFLMUE89	Jeremy	Gallegos	53	1971-02-26	['+44 7972086237']	jeremy0226@gmail.com	55 Morgan mount…

Account details:



field1	Account_Id	Customer_Id	Account_Type	Account_Balance	opened_date	Branch_Code	Status
Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
0	68407485	FAEIWS34	Saving Account	6350.45	2024-04-11	575-626	Closed
1	17913422	TZWZAS16	Current Account	0.0	2020-09-23	684-799	Active
2	88908952	VNDZZO58	Saving Account	7622.65	2023-09-11	750-225	Active
3	32365334	ECIOSJ72	Saving Account	7765.3	1992-12-05	043-627	Active
4	26158269	NMOIWI45	Current Account	3027.78	2008-07-14	345-287	Active
5	86164002	GOLESN46	Saving Account	6743.05	2017-05-23	394-736	Active
6	1702941	JXYNAY49	Fixed Deposit Account	5204.59	2023-09-03	115-360	Suspended
7	13113613	JXCKES01	Saving Account	5288.84	2023-07-20	749-480	Active

Loan Details:



field1	Customer_id	Loan_id	Loan_type	Interest_rate(%)	Loan_amount(£)	Term_in_years
Filter	Filter	Filter	Filter	Filter	Filter	Filter
0	MUPFVG09	SL-341395	Student Loan	6.8	28000	25
1	ZZNJTX31	MTG-132786	Mortgage Loan	4.7	67000	25
2	DCQXPH80	SL-089108	Student Loan	7.7	30000	24
3	PJZJVP61	MTG-238074	Mortgage Loan	4.8	89000	21
4	CAIHMC87	CAR-488800	Car Loan	7.2	21000	3
5	NMFQAE82	PL-522593	Personal Loan	10.3	14000	4

7. Key Highlights

1. Missing and duplicate values were deliberately introduced in datasets for realism.
2. Weighted probabilities ensured realistic distributions for account types and loan assignments.
3. Relationships between customer, account, and loan data were modeled using foreign keys in the database schema.

8.Importance of Ethical Data Handling

In the financial sector, maintaining ethical standards and prioritizing data privacy are critical responsibilities that must be at the forefront of every operation. As organizations work with realistic datasets, even synthetic ones, they must adhere to strict data management practices that reflect the way actual sensitive customer data should be handled. The importance of protecting sensitive information—such as customer phone numbers, email addresses, and physical addresses—cannot be overstated. These personal details are inherently private, and their exposure can have far-reaching consequences. Mishandling of this data, whether through negligence or unethical practices, can lead to significant breaches of customer trust, damaging the reputation of financial institutions and creating legal and financial liabilities. The consequences of such breaches are not just theoretical—they affect real people, leading to identity theft, financial loss, and emotional distress for individuals whose data is compromised.

In the banking sector, it is essential that access to sensitive customer data is carefully controlled and restricted to those who have a legitimate need for it. Employees in positions of higher authority, such as bank managers, may require access to customer datasets for operational or customer service purposes, but this access should be tightly regulated. Access controls and security protocols should ensure that only authorized individuals with the appropriate responsibilities are able to view or manage this sensitive information. By limiting access to those who truly need it, banks can significantly reduce the risk of data misuse or exposure. Furthermore, instituting comprehensive training programs around data privacy, ethical data use, and legal compliance helps create a culture of responsibility and trust, ensuring that sensitive information is treated with the care and respect it deserves. This not only protects customers but also upholds the integrity of the institution itself, fostering long-term relationships built on trust and transparency.

9.Conclusion

The project successfully demonstrated the process of designing and populating a realistic banking dataset using Python and SQLite. The schema incorporated real-world relationships through foreign keys and reflected financial scenarios such as account types and loan distributions. However, the emphasis on ethics and data privacy ensured that the dataset remained a safe tool for learning and analysis.