ASK1

Attestation

I attest that all project team members contribute meaningfully to the project and have complete knowledge of any part of it. Also, our individual contributions in coding, and presentation write-up all come from ourselves, we did not copy or plagiarism from other groups or individuals. - Yachen Wu

In [234…
```
# Identify and describe your dataset
```

- The dataset includes 7 main types of information:

Loan Application Details:

- This part of the dataset provides information about the loans people are applying for. It covers things like the kind of loan, how much money is being borrowed, and the terms for paying it back. It also includes the target variable in this dataset that determine whether the individual has difficulties making payments on their loan.

Risk Assessment Data:

- Includes data points essential for evaluating the likelihood of loan repayment or default, crucial for lending decisions.

Customer Segmentation Information:

- Facilitates the division of applicants into groups based on shared characteristics, aiding in targeted marketing and product customization. Like the detailed information about each applicant, including their income, credit score, gender, and many other factors pertaining to them.

Financial Background:

- Likely includes details about the applicants' financial history, credit scores, or existing debts, crucial for assessing financial health.

Contact and Communication Data:

- It encompasses information about how to contact the applicant, such as phone numbers or email addresses.

Employment History:

- Provides insights into the applicants' career stability and earning potential, important for assessing their ability to repay loans.

- Our dataset is 148.7 MBs and contains 122 columns and 307,511 rows.

In [236…

```
# Identify dataset source
```

We sourced our dataset from The International Institute of Information Technology Bangalore. The IIITB is an industrial research organization relating to information technology. It comes from India credit card applications in September of 2013.

In [ ]:

```
# Why is important and what appeals to you about it
```

- The data can help in creating models to predict which loans might not be paid back, which is important for banks to know. It also helps in understanding different groups of customers better, which can guide banks in offering the right kind of loan products and services to the right people.
- It assists the general public in understanding the variables that impact the default rate, particularly for lenders.
- It is suitable for data analysis and dimensional modeling with a dataset containing over 300k records.
- There are122 variables in a well-structured dataset with a data dictionary to help in the selection and categorization of significant variables and the generation of analysis.
- In summary, this dataset is a key tool for managing risk, understanding customers, and making smart decisions in the financial sector.

In [238…

```
# Acquire data and perform initial exploration to make sure it is suitable for
```

Numerical Data for Fact Tables:

- The dataset includes crucial numerical values like income and credit scores, pivotal for building detailed fact tables in dimensional modeling.

Categorical Attributes for Dimension Tables:

- It features categorical attributes (e.g., loan types, employment status) that are ideal for crafting dimension tables, enhancing data segmentation and analysis.

Star Schema Implementation:

- A star schema is proposed using natural keys as surrogate keys, streamlining the relationship between facts and dimensions for efficient data querying.

Enhanced Data Manipulation:

- The dataset's coded categorical attributes facilitate easier sorting and manipulation, making it highly adaptable for diverse analytical purposes.

In summary, the dataset is well-prepared for dimensional modeling and analysis, with its blend of numerical and categorical data ideally suited for creating a robust and efficient star schema. This structure will enable us to perform in-depth, multifaceted analyses, providing solid insights for decision-making processes.

In [240…
```
# Describe the analytical questions you want to answer with the data.
```

1. In the provided loan data, what is the average loan default rate? What are the loan types along with which income levels have the higher default rate?

2. What are the differences between females and males regarding their average income and default rate? Also, within each gender group, which income level shows the highest frequency of defaults?

3. How can we identify and categorize the risk profiles of our loan applicants based on their income level, age group, and occupation? Specifically, which income group has the highest overall default rate, and within this group, which age bracket is most prone to defaulting? Furthermore, among the high-risk age bracket in the highest defaulting income group, what are the common occupation types, and how do their default rates compare?

In [241…
```
# Describe any concerns with the data and changes you expect to overcome
```

1. **Data Volume**:

- The dataset contains a substantial number of variables, with 122 in the original dataset. This abundance of variables can lead to reduced data accuracy, increased computational time, and challenges in interpretation. To address this, we plan to narrow down the variables to a more manageable 19, focusing on those directly related to loan default exploration.

2. **Categorical Grouping**

- For variables that are currently numerical but represent categories (like the example columns 'days_birth' and 'cnt_children'), consider converting them into categorical variables. This grouping can provide more meaningful insights and facilitate easier analysis.
- For 'days_birth', converting age into categorized age groups (such as young adult, middle-aged, senior) would allow for a clearer understanding of different age demographics. Similarly, for 'cnt_children', categorizing the data into groups (like no children, one child, multiple children) would offer a more structured way to analyze the impact of the number of children on other variables. These changes will not only improve the clarity of the analysis but also make it easier to identify and interpret trends and patterns within the data.

ASK2

Before started

set a new folder for this project

In [2]:
```
# create an final project folder
!mkdir Final_project
```

In [7]:
```
cd /home/ubuntu/notebooks/Final_project
```

```
/home/ubuntu/notebooks/Final_project
```

In [5]:
```
# set working directory
%cd Final_project
```

```
[Errno 2] No such file or directory: 'Final_project'
/home/ubuntu/notebooks/Final_project
```

In [1]:
```
# double-check final directory
!pwd
```

```
/home/ubuntu/notebooks/Final_project
```

Uploaded application_data.csv

In [2]:
```
!csvcut -n application_data.csv
```

```
 1: SK_ID_CURR
 2: TARGET
 3: NAME_CONTRACT_TYPE
 4: CODE_GENDER
 5: FLAG_OWN_CAR
 6: FLAG_OWN_REALTY
 7: CNT_CHILDREN
 8: AMT_INCOME_TOTAL
 9: AMT_CREDIT
10: AMT_ANNUITY
11: AMT_GOODS_PRICE
12: NAME_TYPE_SUITE
13: NAME_INCOME_TYPE
14: NAME_EDUCATION_TYPE
15: NAME_FAMILY_STATUS
16: NAME_HOUSING_TYPE
17: REGION_POPULATION_RELATIVE
18: DAYS_BIRTH
19: DAYS_EMPLOYED
20: DAYS_REGISTRATION
21: DAYS_ID_PUBLISH
22: OWN_CAR_AGE
23: FLAG_MOBIL
24: FLAG_EMP_PHONE
25: FLAG_WORK_PHONE
26: FLAG_CONT_MOBILE
27: FLAG_PHONE
28: FLAG_EMAIL
29: OCCUPATION_TYPE
30: CNT_FAM_MEMBERS
31: REGION_RATING_CLIENT
32: REGION_RATING_CLIENT_W_CITY
33: WEEKDAY_APPR_PROCESS_START
34: HOUR_APPR_PROCESS_START
35: REG_REGION_NOT_LIVE_REGION
36: REG_REGION_NOT_WORK_REGION
37: LIVE_REGION_NOT_WORK_REGION
38: REG_CITY_NOT_LIVE_CITY
39: REG_CITY_NOT_WORK_CITY
40: LIVE_CITY_NOT_WORK_CITY
41: ORGANIZATION_TYPE
42: EXT_SOURCE_1
43: EXT_SOURCE_2
44: EXT_SOURCE_3
45: APARTMENTS_AVG
46: BASEMENTAREA_AVG
47: YEARS_BEGINEXPLUATATION_AVG
48: YEARS_BUILD_AVG
49: COMMONAREA_AVG
50: ELEVATORS_AVG
51: ENTRANCES_AVG
52: FLOORSMAX_AVG
53: FLOORSMIN_AVG
54: LANDAREA_AVG
55: LIVINGAPARTMENTS_AVG
56: LIVINGAREA_AVG
57: NONLIVINGAPARTMENTS_AVG
```

```
 58: NONLIVINGAREA_AVG
 59: APARTMENTS_MODE
 60: BASEMENTAREA_MODE
 61: YEARS_BEGINEXPLUATATION_MODE
 62: YEARS_BUILD_MODE
 63: COMMONAREA_MODE
 64: ELEVATORS_MODE
 65: ENTRANCES_MODE
 66: FLOORSMAX_MODE
 67: FLOORSMIN_MODE
 68: LANDAREA_MODE
 69: LIVINGAPARTMENTS_MODE
 70: LIVINGAREA_MODE
 71: NONLIVINGAPARTMENTS_MODE
 72: NONLIVINGAREA_MODE
 73: APARTMENTS_MEDI
 74: BASEMENTAREA_MEDI
 75: YEARS_BEGINEXPLUATATION_MEDI
 76: YEARS_BUILD_MEDI
 77: COMMONAREA_MEDI
 78: ELEVATORS_MEDI
 79: ENTRANCES_MEDI
 80: FLOORSMAX_MEDI
 81: FLOORSMIN_MEDI
 82: LANDAREA_MEDI
 83: LIVINGAPARTMENTS_MEDI
 84: LIVINGAREA_MEDI
 85: NONLIVINGAPARTMENTS_MEDI
 86: NONLIVINGAREA_MEDI
 87: FONDKAPREMONT_MODE
 88: HOUSETYPE_MODE
 89: TOTALAREA_MODE
 90: WALLSMATERIAL_MODE
 91: EMERGENCYSTATE_MODE
 92: OBS_30_CNT_SOCIAL_CIRCLE
 93: DEF_30_CNT_SOCIAL_CIRCLE
 94: OBS_60_CNT_SOCIAL_CIRCLE
 95: DEF_60_CNT_SOCIAL_CIRCLE
 96: DAYS_LAST_PHONE_CHANGE
 97: FLAG_DOCUMENT_2
 98: FLAG_DOCUMENT_3
 99: FLAG_DOCUMENT_4
100: FLAG_DOCUMENT_5
101: FLAG_DOCUMENT_6
102: FLAG_DOCUMENT_7
103: FLAG_DOCUMENT_8
104: FLAG_DOCUMENT_9
105: FLAG_DOCUMENT_10
106: FLAG_DOCUMENT_11
107: FLAG_DOCUMENT_12
108: FLAG_DOCUMENT_13
109: FLAG_DOCUMENT_14
110: FLAG_DOCUMENT_15
111: FLAG_DOCUMENT_16
112: FLAG_DOCUMENT_17
113: FLAG_DOCUMENT_18
114: FLAG_DOCUMENT_19
```

```
115: FLAG_DOCUMENT_20
116: FLAG_DOCUMENT_21
117: AMT_REQ_CREDIT_BUREAU_HOUR
118: AMT_REQ_CREDIT_BUREAU_DAY
119: AMT_REQ_CREDIT_BUREAU_WEEK
120: AMT_REQ_CREDIT_BUREAU_MON
121: AMT_REQ_CREDIT_BUREAU_QRT
122: AMT_REQ_CREDIT_BUREAU_YEAR
```

There are 122 columns in the original dataset

But not all of them are variables we need.

A lot of columns' descriptions are ambiguous, not clearly defines what they mean. Or the information is normalized, so we cannot get any insights from them, such as the following variables: APARTMENTS_AVG Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor

Eventually, we boiled down to 19 columns that we can use in the dimensional model.In the original dataset of 122 columns, many were not pertinent to our analysis due to ambiguous descriptions or the presence of normalized data that limited insight extraction. An example is 'APARTMENTS_AVG', which offered normalized details on various building aspects. After a thorough review focusing on clarity, relevance, and potential insights, we condensed the dataset to 19 key columns, thereby sharpening our focus and enhancing the effectiveness of our dimensional model analysis. This refined approach ensures we target the most relevant data for our analytical goals.

Removing Unwanted Columns

Keeping only those columns which are part of the dimension model

In [2]:
```
!csvcut -c TARGET,AMT_INCOME_TOTAL,AMT_CREDIT,AMT_REQ_CREDIT_BUREAU_HOUR,AMT_RE
```

Here is the list of the 19 selected columns we use in the project

We store them in a new datatset names application_data2.csv

In [3]:
```
!csvcut -n application_data2.csv
```

```
  1: TARGET
  2: AMT_INCOME_TOTAL
  3: AMT_CREDIT
  4: AMT_REQ_CREDIT_BUREAU_HOUR
  5: AMT_REQ_CREDIT_BUREAU_DAY
  6: FLAG_MOBIL
  7: FLAG_EMAIL
  8: FLAG_CONT_MOBILE
  9: SK_ID_CURR
 10: NAME_CONTRACT_TYPE
 11: AMT_ANNUITY
 12: CODE_GENDER
 13: NAME_INCOME_TYPE
 14: NAME_EDUCATION_TYPE
 15: NAME_FAMILY_STATUS
 16: NAME_HOUSING_TYPE
 17: OCCUPATION_TYPE
 18: CNT_CHILDREN
 19: DAYS_BIRTH
```

In [4]:
```
!csvclean -n application_data2.csv
```

```
No errors.
```

In [5]:
```
!csvcut application_data2.csv | csvstat
```

```
/home/ubuntu/.local/lib/python3.8/site-packages/agate/table/from_csv.py:74: Ru
ntimeWarning: Error sniffing CSV dialect: Could not determine delimiter
  1. "TARGET"

        Type of data:          Boolean
        Contains null values:  False
        Unique values:         2
        Most common values:    False (282686x)
                               True (24825x)

  2. "AMT_INCOME_TOTAL"

        Type of data:          Number
        Contains null values:  False
        Unique values:         2548
        Smallest value:        25650
        Largest value:         117000000
        Sum:                   51907216960.935
        Mean:                  168797.919
        Median:                147150
        StDev:                 237123.146
        Most common values:    135000 (35750x)
                               112500 (31019x)
                               157500 (26556x)
                               180000 (24719x)
                               90000 (22483x)

  3. "AMT_CREDIT"

        Type of data:          Number
        Contains null values:  False
        Unique values:         5603
        Smallest value:        45000
        Largest value:         4050000
        Sum:                   184207084195.5
        Mean:                  599026
        Median:                513531
        StDev:                 402490.777
        Most common values:    450000 (9709x)
                               675000 (8877x)
                               225000 (8162x)
                               180000 (7342x)
                               270000 (7241x)

  4. "AMT_REQ_CREDIT_BUREAU_HOUR"

        Type of data:          Number
        Contains null values:  True (excluded from calculations)
        Unique values:         6
        Smallest value:        0
        Largest value:         4
        Sum:                   1703
        Mean:                  0.006
        Median:                0
        StDev:                 0.084
        Most common values:    0 (264366x)
                               None (41519x)
```

```
                                1 (1560x)
                                2 (56x)
                                3 (9x)

5. "AMT_REQ_CREDIT_BUREAU_DAY"

      Type of data:           Number
      Contains null values:   True (excluded from calculations)
      Unique values:          10
      Smallest value:         0
      Largest value:          9
      Sum:                    1862
      Mean:                   0.007
      Median:                 0
      StDev:                  0.111
      Most common values:     0 (264503x)
                              None (41519x)
                              1 (1292x)
                              2 (106x)
                              3 (45x)

6. "FLAG_MOBIL"

      Type of data:           Boolean
      Contains null values:   False
      Unique values:          2
      Most common values:     True (307510x)
                              False (1x)

7. "FLAG_EMAIL"

      Type of data:           Boolean
      Contains null values:   False
      Unique values:          2
      Most common values:     False (290069x)
                              True (17442x)

8. "FLAG_CONT_MOBILE"

      Type of data:           Boolean
      Contains null values:   False
      Unique values:          2
      Most common values:     True (306937x)
                              False (574x)

9. "SK_ID_CURR"

      Type of data:           Number
      Contains null values:   False
      Unique values:          307511
      Smallest value:         100002
      Largest value:          456255
      Sum:                    85543569448
      Mean:                   278180.519
      Median:                 278202
      StDev:                  102790.175
      Most common values:     100002 (1x)
```

```
                        100003 (1x)
                        100004 (1x)
                        100006 (1x)
                        100007 (1x)

10. "NAME_CONTRACT_TYPE"

        Type of data:          Text
        Contains null values:  False
        Unique values:         2
        Longest value:         15 characters
        Most common values:    Cash loans (278232x)
                               Revolving loans (29279x)

11. "AMT_ANNUITY"

        Type of data:          Number
        Contains null values:  True (excluded from calculations)
        Unique values:         13673
        Smallest value:        1615.5
        Largest value:         258025.5
        Sum:                   8335859368.5
        Mean:                  27108.574
        Median:                24903
        StDev:                 14493.737
        Most common values:    9000 (6385x)
                               13500 (5514x)
                               6750 (2279x)
                               10125 (2035x)
                               37800 (1602x)

12. "CODE_GENDER"

        Type of data:          Text
        Contains null values:  False
        Unique values:         3
        Longest value:         3 characters
        Most common values:    F (202448x)
                               M (105059x)
                               XNA (4x)

13. "NAME_INCOME_TYPE"

        Type of data:          Text
        Contains null values:  False
        Unique values:         8
        Longest value:         20 characters
        Most common values:    Working (158774x)
                               Commercial associate (71617x)
                               Pensioner (55362x)
                               State servant (21703x)
                               Unemployed (22x)

14. "NAME_EDUCATION_TYPE"

        Type of data:          Text
        Contains null values:  False
```

```
        Unique values:         5
        Longest value:         29 characters
        Most common values:    Secondary / secondary special (218391x)
                               Higher education (74863x)
                               Incomplete higher (10277x)
                               Lower secondary (3816x)
                               Academic degree (164x)

15. "NAME_FAMILY_STATUS"

        Type of data:          Text
        Contains null values:  False
        Unique values:         6
        Longest value:         20 characters
        Most common values:    Married (196432x)
                               Single / not married (45444x)
                               Civil marriage (29775x)
                               Separated (19770x)
                               Widow (16088x)

16. "NAME_HOUSING_TYPE"

        Type of data:          Text
        Contains null values:  False
        Unique values:         6
        Longest value:         19 characters
        Most common values:    House / apartment (272868x)
                               With parents (14840x)
                               Municipal apartment (11183x)
                               Rented apartment (4881x)
                               Office apartment (2617x)

17. "OCCUPATION_TYPE"

        Type of data:          Text
        Contains null values:  True (excluded from calculations)
        Unique values:         19
        Longest value:         21 characters
        Most common values:    None (96391x)
                               Laborers (55186x)
                               Sales staff (32102x)
                               Core staff (27570x)
                               Managers (21371x)

18. "CNT_CHILDREN"

        Type of data:          Number
        Contains null values:  False
        Unique values:         15
        Smallest value:        0
        Largest value:         19
        Sum:                   128248
        Mean:                  0.417
        Median:                0
        StDev:                 0.722
        Most common values:    0 (215371x)
                               1 (61119x)
```

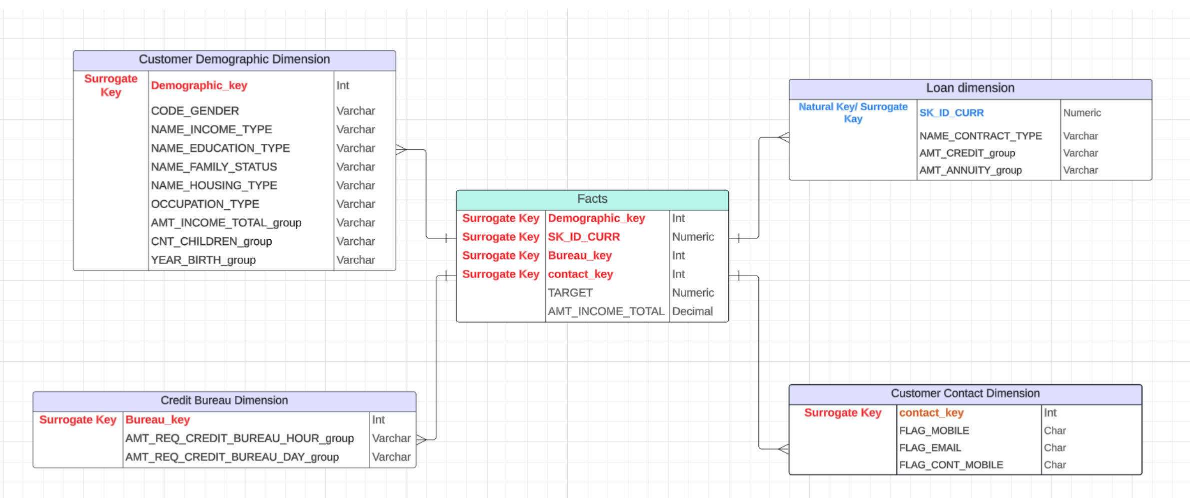
```
                                 2 (26749x)
                                 3 (3717x)
                                 4 (429x)

 19. "DAYS_BIRTH"

        Type of data:           Number
        Contains null values:   False
        Unique values:          17460
        Smallest value:         -25229
        Largest value:          -7489
        Sum:                    -4931552390
        Mean:                   -16036.995
        Median:                 -15750
        StDev:                  4363.989
        Most common values:     -13749 (43x)
                                -13481 (42x)
                                -18248 (41x)
                                -10020 (41x)
                                -10292 (40x)

Row count: 307511
```

Using csvstat to know the detail of each column in csv file

Schema

In [6]:
```python
from IPython.display import Image
Image(url="https://github.com/zilingli1/Data-Wrangling-Project-/blob/main/Dimer
```

Out[6]:



Column Description

- `Taget` : The 'Target' column in our dataset is crucial for understanding clients' loan repayment behavior. This column does not have any missing values, ensuring the completeness and reliability of our analysis. It stores values of either 1 or 0, where 1 indicates clients who are experiencing payment difficulties, and 0 represents clients who are managing their loan repayments without any issues. For analytical purposes, we use a derived equation based on the 'Target' variable to calculate the default rate. This approach allows us to quantitatively assess the risk of loan defaults. In our analyses, therefore, the 'Target' variable is treated as a numeric measure that helps in evaluating the financial stability and reliability of clients.

- `AMT_INCOME_TOTAL_GROUP` : There are no missing values in AMT_INCOME_TOTAL_Group. This column describes the income of client. Since the income has a range from 25,650 to 117,000,000 with either whole numbers or one decimal, we assume that they are monthly income provided by the client for the loan. We decided to group the income into 4 income level groups and name it as amt_income_total_group to store it in the client demographic dimension table.

- `AMT_CREDIT_GROUP` : There are no missing values in AMT_CREDIT. AMT_CREDIT stores the credit amount to the loan. AMT_CREDIT ranging from 45,000 to 4,050,00 with either whole numbers or one decimal number. We decided to group them into 5 credit level groups and name it as amt_credit_group to store in the loan dimension table.

- `AMT_CREDIT_BUREAU_HOUR_GROUP` : Number of enquiries to Credit Bureau about the client one hour before application

- `AMT_CREDIT_BUREAU_DAY_GROUP` : Number of enquiries to Credit Bureau about the client one day before application (excluding one hour before application)

- `FLAG_MOBIL` : There are no missing values in FLAG_MOBLE. FLAG_MOBIL records whether client provided the mobile number when applying for the loan. 0 and 1 are the only values in this column with 1 represents mobile number provided and 0 represents mobile did not provide.

- `FLAG_EMAIL` : There are no missing values in FLAG_EMAIL. FLAG_EMAIL records whether client provided the email when applying for the loan. 0 and 1 are the only values in this column with 1 represents a provided email and 0 represents email address did not provide.

- `FLAG_CONT_MOBILE` : There are no missing values in this column. Was mobile phone that custumer provided reachable or not? (1=YES, 0=NO)

- `SK_ID_CURR` : There are no missing values in this column. SK_ID_CURR is the UNIQUE loanID for each loan entry.

- `NAME_CONTRACT_TYPE` : There are no missing values in this column. Cash loans and revolving loans are the two inputs in this column.

- `AMT_ANNUITY` : There are missing values in this column and we replace them as NULL. AMT_ANNUITY is the annuity on the loan that needs to be paid by the client. AMT_ANNUITY ranging from 1615.5 to 258025.5 with either whole numbers or one decimal number. We grouped this column into 6 annuity level groups and name it as amt_annuity_group to store in the loan dimension table.

- `CODE_GENDER` : There are no missing values in this column. 'M', 'F', and 'XNA' are the 3 inputs included in this column.

- `NAME_INCOME_TYPE` : There are no missing values in this column. 'working', 'state servant', 'commerical associate', 'pensioner', 'unemployed', 'student', 'businessman', and 'maternity leave' are the 8 inputs in this column.

- `NAME_EDUCATION_TYPE` : There are no missing values in this column. 'Secondary / secondary special', 'Higher education', 'Incomplete higher', 'Lower secondary', and 'Academic degree' are the 5 inputs in this column.

- `NAME_FAMILY_STATUS` : There are no missing values in this column. 'Single / not married', 'Married', 'Civil marriage', 'Widow', 'Separated', and 'Unknown' are the 6 inputs in this column.

- `NAME_HOUSING_TYPE` : There are no missing values in this column. 'House / apartment', 'Rented apartment', 'With parents', 'Municipal apartment', 'Office apartment', and 'Co-op apartment' are the 6 housing types.

- `OCCUPATION_TYPE` : There are missing values in this column and we replace them as NULL. 18 occupations are included in this column.

- `CNT_CHILDREN` : There are no missing values in this column. CNT_CHILDREN records number of children the client has, and it ranges from 0 to 19. We grouped CNT_CHILDREN into 3 children level groups and name it as cnt_children_group to store it in the client dimension table.

- `DAYS_BIRTH` : There are no missing values in this column. DAYS_BIRTH records the client's age in days at the time of the application. We divided the values by 365 to get the expected client's age and grouped the whole age number into 5 age groups and store it in the client dimension table.

Modified columns in data wrangling:

- amt_income_total_group
- amt_credit_group
- amt_annuity_group
- cnt_children_group
- days_birth_group
- amt_req_credit_bureau_hour_group
- amt_req_credit_bureau_day_group

Set up the database and create the table

Dropping Final_Project Database if it exist

In [7]:
```
!dropdb -U student Final_Project
```

Creating database Final_Project

In [8]:
```
!createdb -U student Final_Project
```

The ipython-sql library is loaded using the %load_ext iPython extension syntax and is pointed to the connection object

In [9]:
```
%load_ext sql
```

In [10]:
```
%sql postgresql://student@/Final_Project
```

In [11]:
```
!psql --version
```

```
psql (PostgreSQL) 12.17 (Ubuntu 12.17-0ubuntu0.20.04.1)
```

Creating Application_Data Table

In [12]:
```
%%sql
DROP TABLE IF EXISTS APPLICATION_DATA Cascade;

CREATE TABLE APPLICATION_DATA (
    TARGET NUMERIC(1) NOT NULL,
    AMT_INCOME_TOTAL DECIMAL(10,1) NOT NULL,
    AMT_CREDIT DECIMAL(10,1) NOT NULL,
    AMT_REQ_CREDIT_BUREAU_HOUR NUMERIC(5),
    AMT_REQ_CREDIT_BUREAU_DAY NUMERIC(5),
    FLAG_MOBIL CHAR(1) NOT NULL,
    FLAG_EMAIL CHAR(1) NOT NULL,
    FLAG_CONT_MOBILE CHAR(1) NOT NULL,
    SK_ID_CURR NUMERIC(6) NOT NULL,
    NAME_CONTRACT_TYPE VARCHAR(15) NOT NULL,
    AMT_ANNUITY DECIMAL(8,1),
    CODE_GENDER VARCHAR(3) NOT NULL,
    NAME_INCOME_TYPE VARCHAR(20) NOT NULL,
    NAME_EDUCATION_TYPE VARCHAR(29) NOT NULL,
    NAME_FAMILY_STATUS VARCHAR(20) NOT NULL,
    NAME_HOUSING_TYPE VARCHAR(19) NOT NULL,
    OCCUPATION_TYPE VARCHAR(21),
    CNT_CHILDREN NUMERIC(2) NOT NULL,
    DAYS_BIRTH NUMERIC(5) NOT NULL
);
```

```
 * postgresql://student@/Final_Project
Done.
Done.
```

Out[12]: []

comment:

We decide to treat the columns `FLAG_MOBIL`, `FLAG_EMAIL`, and `FLAG_CONT_MOBILE` as `CHAR(1)` instead of boolean (`BOOLEAN`) in the dimension table and analytical questions can offer certain benefits:

1. **Clarity in Dimension Table**:
   - Using `CHAR(1)` provides more explicit information in the dimension table. The values '0' and '1' in `CHAR(1)` indicate the absence and presence of a feature, respectively. This clarity can help users easily understand the meaning of these columns without prior knowledge of boolean conventions.
2. **Flexible for Future Needs**:
   - Treating these columns as `CHAR(1)` offers flexibility. If there is a need to expand the encoding beyond binary values ('0' and '1') to represent additional states or categories in the future, using character data allows for such expansion without major schema changes.
3. **Ease of Reporting and Visualization**:
   - Character values can be more informative when generating reports or visualizations. They can be easily labeled in a user-friendly way, making it simpler for stakeholders to understand the data and results.

Loading contents of application_data.csv into application_data sql table

In [13]:
```
%%sql
COPY APPLICATION_DATA FROM '/home/ubuntu/notebooks/Final_project/application_da
CSV
HEADER;
```

```
 * postgresql://student@/Final_Project
307511 rows affected.
```

Out[13]: []

Checking the Total Number of Rows are same in csv file and sql table

Becasue of headers the count of csv will be greater by 1 in sql table

In [14]:
```
%%sql
SELECT COUNT(*) FROM APPLICATION_DATA;
```

```
 * postgresql://student@/Final_Project
1 rows affected.
```

Out[14]:

count
307511

In [15]:
```
!wc -l application_data2.csv
```

```
307512 application_data2.csv
```

In [16]:
```
%%sql
SELECT * FROM APPLICATION_DATA
LIMIT 10;
```

```
 * postgresql://student@/Final_Project
10 rows affected.
```

Out[16]:

target	amt_income_total	amt_credit	amt_req_credit_bureau_hour	amt_req_credit_bureau_day
1	202500.0	406597.5	0	0
0	270000.0	1293502.5	0	0
0	67500.0	135000.0	0	0
0	135000.0	312682.5	None	None
0	121500.0	513000.0	0	0
0	99000.0	490495.5	0	0
0	171000.0	1560726.0	0	0
0	360000.0	1530000.0	0	0
0	112500.0	1019610.0	0	0
0	135000.0	405000.0	None	None

Data Wrangling

Creating amt_income_total_group from amt_income_total

In [17]:
```
%%sql
ALTER TABLE APPLICATION_DATA
ADD COLUMN amt_income_total_group VARCHAR(20);
```

```
 * postgresql://student@/Final_Project
Done.
```

Out[17]: []

In [18]:
```
%%sql
UPDATE APPLICATION_DATA
SET amt_income_total_group=CASE
    when amt_income_total<=100000 then '<=100K'
    when amt_income_total between 100001 and 150000 then '100K-150K'
    when amt_income_total between 150001 and 200000 then '150K-200K'
    else '>200K'
END;
```

```
 * postgresql://student@/Final_Project
307511 rows affected.
```

Out[18]: []

Creating amt_credit_group from amt_credit

In [19]:
```
%%sql
ALTER TABLE APPLICATION_DATA
ADD COLUMN amt_credit_group VARCHAR(20);
```

```
 * postgresql://student@/Final_Project
Done.
```

Out[19]: []

In [20]:
```
%%sql
UPDATE APPLICATION_DATA
SET amt_credit_group=CASE
    when amt_credit<=250000 then '<=250K'
    when amt_credit between 250001 and 500000 then '250K-500K'
    when amt_credit between 500001 and 750000 then  '500K-750K'
    when amt_credit between 750001 and 1000000 then '750K-1000K'
    else '>1000K'
END;
```

```
 * postgresql://student@/Final_Project
307511 rows affected.
```

Out[20]: []

Creating amt_annuity_group from amt_annuity

In [21]:
```
%%sql
ALTER TABLE APPLICATION_DATA
ADD COLUMN amt_annuity_group VARCHAR(20);
```

```
 * postgresql://student@/Final_Project
Done.
```

Out[21]: []

In [22]:
```
%%sql
UPDATE APPLICATION_DATA
SET amt_annuity_group=CASE
    when AMT_ANNUITY<=15000 then '<=15K'
    when AMT_ANNUITY between 15001 and 25000 then '15K-25K'
    when AMT_ANNUITY between 25001 and 35000 then '25K-35K'
    when AMT_ANNUITY between 35001 and 45000 then '35K-45K'
    when AMT_ANNUITY between 45001 and 55000 then '45K-55K'
    else '>55K'
END;
```

```
 * postgresql://student@/Final_Project
307511 rows affected.
```

Out[22]: []

In [23]:
```
%%sql
ALTER TABLE APPLICATION_DATA
```

```
ADD COLUMN Year_Birth NUMERIC(3),
ADD COLUMN Year_Birth_GROUP VARCHAR(20);
```

```
 * postgresql://student@/Final_Project
Done.
```

Out[23]: []

In [24]:
```
%%sql
UPDATE APPLICATION_DATA
SET Year_Birth=round((DAYS_BIRTH*-1)/365)
```

```
 * postgresql://student@/Final_Project
307511 rows affected.
```

Out[24]: []

Cleaning the `DAYS_BIRTH` column and creating `Year_Birth` and `Year_Birth_group` columns has several advantages:

1. **Data Clarity**: Converting 'DAYS_BIRTH' to 'Year_Birth' makes the age representation more intuitive and easier to understand. Instead of negative values, we have positive whole numbers representing years.

2. **Consistency**: The transformation ensures consistency in representing ages across the dataset, making it easier to work with and interpret.

3. **Age Grouping**: The 'Year_Birth' column can be further grouped into age brackets or 'Year_Birth_group,' which is valuable for segmenting and analyzing the data by age ranges.

4. **Improved Readability**: The transformed column enhances the readability of the dataset, making it more accessible for analysis and reporting.

By making these changes, we enhance the overall quality and usability of the data for our analytical questions.

- check if it is successfully transformed

In [25]:
```
%%sql
SELECT Year_Birth
from APPLICATION_DATA
LIMIT 5;
```

```
 * postgresql://student@/Final_Project
5 rows affected.
```

Out[25]:

year_birth
54
57
55
55
65

- now group the age into four levels

In [26]:
```
%%sql
UPDATE APPLICATION_DATA
SET Year_Birth_GROUP=case
    when Year_Birth between 21 and 30 then '21-30 Years'
    when Year_Birth between 31 and 40 then '31-40 Years'
    when Year_Birth between 41 and 50 then '41-50 Years'
    when Year_Birth between 51 and 60 then '51-60 Years'
    else '>=61 Years'
END;
```

```
 * postgresql://student@/Final_Project
307511 rows affected.
```

Out[26]: []

In [27]:
```
%%sql
select * from application_data
limit 10;
```

```
 * postgresql://student@/Final_Project
10 rows affected.
```

Out[27]:

target	amt_income_total	amt_credit	amt_req_credit_bureau_hour	amt_req_credit_bureau_day
0	94500.0	1546020.0	None	None
0	81000.0	1125000.0	None	None
0	225000.0	1800000.0	None	None
0	112500.0	157500.0	None	None
0	67500.0	807984.0	None	None
0	360000.0	1264428.0	None	None
0	90000.0	544491.0	None	None
0	337500.0	225000.0	None	None
0	225000.0	518562.0	0	0
0	90000.0	202500.0	None	None

Check whether the min and max for each age group is imputed right

In [28]:
```
%%sql
select Year_Birth_GROUP,max(Year_Birth) as max1,
```

```
min(Year_Birth) as min1
from APPLICATION_DATA
group by 1;
```

```
 * postgresql://student@/Final_Project
5 rows affected.
```

Out[28]:

year_birth_group	max1	min1
21-30 Years	30	21
31-40 Years	40	31
41-50 Years	50	41
51-60 Years	60	51
>=61 Years	69	61

In [29]:

```
%%sql
select * from application_data
limit 10;
```

```
 * postgresql://student@/Final_Project
10 rows affected.
```

Out[29]:

target	amt_income_total	amt_credit	amt_req_credit_bureau_hour	amt_req_credit_bureau_day
0	90000.0	202500.0	None	None
0	202500.0	1575000.0	None	None
0	54000.0	113211.0	None	None
0	90000.0	50940.0	0	0
0	247500.0	274779.0	None	None
0	90000.0	1507869.0	0	0
0	65250.0	450000.0	None	None
0	261000.0	450000.0	None	None
0	94500.0	518562.0	None	None
0	157500.0	722394.0	0	0

Creating `CNT_CHILDREN_Group` from `CNT_CHILDREN`

In [30]:

```
%%sql
ALTER TABLE APPLICATION_DATA
ADD COLUMN CNT_CHILDREN_Group VARCHAR(20);
```

```
 * postgresql://student@/Final_Project
Done.
```

Out[30]: []

In [31]:

```
%%sql
UPDATE APPLICATION_DATA
SET CNT_CHILDREN_Group=CASE
    WHEN CNT_CHILDREN=0 THEN 'No children'
    WHEN CNT_CHILDREN=1 THEN '1 child'
```

```
    WHEN CNT_CHILDREN>=2 THEN '>=2 children'
END;
```

```
 * postgresql://student@/Final_Project
307511 rows affected.
```

Out[31]: []

Check whether the grouping makes sense, for example, children group >=2 children has minimum of 2 children and maximum of 19 children

In [32]:

```
%%sql
select CNT_CHILDREN_Group,
max(CNT_CHILDREN) as max1,
min(CNT_CHILDREN) as min1
from APPLICATION_DATA
group by 1;
```

```
 * postgresql://student@/Final_Project
3 rows affected.
```

Out[32]:

cnt_children_group	max1	min1
1 child	1	1
>=2 children	19	2
No children	0	0

Creating AMT_REQ_CREDIT_BUREAU_HOUR_group from AMT_REQ_CREDIT_BUREAU_HOUR

In [33]:

```
%%sql
ALTER TABLE APPLICATION_DATA
ADD COLUMN AMT_REQ_CREDIT_BUREAU_HOUR_group VARCHAR(50);
```

```
 * postgresql://student@/Final_Project
Done.
```

Out[33]: []

In [34]:

```
%%sql
UPDATE APPLICATION_DATA
SET AMT_REQ_CREDIT_BUREAU_HOUR_group=CASE
    WHEN AMT_REQ_CREDIT_BUREAU_HOUR is Null or AMT_REQ_CREDIT_BUREAU_HOUR=0 THE
    WHEN AMT_REQ_CREDIT_BUREAU_HOUR>=1 THEN '1 or more Enquiries'
END;
```

```
 * postgresql://student@/Final_Project
307511 rows affected.
```

Out[34]: []

In [35]:

```
%%sql
select AMT_REQ_CREDIT_BUREAU_HOUR_group,
max(AMT_REQ_CREDIT_BUREAU_HOUR) as max1,
min(AMT_REQ_CREDIT_BUREAU_HOUR) as min1
from APPLICATION_DATA
group by 1;
```

```
 * postgresql://student@/Final_Project
2 rows affected.
```

Out[35]:

amt_req_credit_bureau_hour_group	max1	min1
1 or more Enquiries	4	1
Zero Enquiry	0	0

Creating AMT_REQ_CREDIT_BUREAU_DAY_group from AMT_REQ_CREDIT_BUREAU_DAY

In [36]:

```
%%sql
ALTER TABLE APPLICATION_DATA
ADD COLUMN AMT_REQ_CREDIT_BUREAU_DAY_group VARCHAR(50);
```

```
 * postgresql://student@/Final_Project
Done.
```

Out[36]: []

In [37]:

```
%%sql
UPDATE APPLICATION_DATA
SET AMT_REQ_CREDIT_BUREAU_DAY_group=CASE
    WHEN AMT_REQ_CREDIT_BUREAU_DAY is Null or AMT_REQ_CREDIT_BUREAU_DAY=0 THEN
    WHEN AMT_REQ_CREDIT_BUREAU_DAY>=1 THEN '1 or more Enquiries'
END;
```

```
 * postgresql://student@/Final_Project
307511 rows affected.
```

Out[37]: []

In [38]:

```
%%sql
select AMT_REQ_CREDIT_BUREAU_DAY_group,
max(AMT_REQ_CREDIT_BUREAU_DAY) as max1,
min(AMT_REQ_CREDIT_BUREAU_Day) as min1
from APPLICATION_DATA
group by 1;
```

```
 * postgresql://student@/Final_Project
2 rows affected.
```

Out[38]:

amt_req_credit_bureau_day_group	max1	min1
1 or more Enquiries	9	1
Zero Enquiry	0	0

Drop Columns which are not required since we have categorical version of it, so the original column we do not need to use

In [39]:

```
%%sql
ALTER TABLE APPLICATION_DATA
DROP COLUMN AMT_REQ_CREDIT_BUREAU_HOUR,
DROP COLUMN AMT_REQ_CREDIT_BUREAU_DAY,
DROP COLUMN AMT_CREDIT,
DROP COLUMN AMT_ANNUITY,
DROP COLUMN CNT_CHILDREN,
```

```
DROP COLUMN DAYS_BIRTH,
DROP COLUMN YEAR_BIRTH;
```

```
 * postgresql://student@/Final_Project
Done.
```

Out[39]: []

In [40]:
```
%%sql
select * from application_data
limit 10;
```

```
 * postgresql://student@/Final_Project
10 rows affected.
```

Out[40]:

target	amt_income_total	flag_mobil	flag_email	flag_cont_mobile	sk_id_curr	name_contract_
0	157500.0	1	0	1	315919	Cash l
0	225000.0	1	1	1	316082	Cash l
0	94500.0	1	0	1	316755	Cash l
0	81000.0	1	0	1	317502	Cash l
0	414000.0	1	1	1	324023	Cash l
0	225000.0	1	0	1	324367	Cash l
0	81000.0	1	0	1	326417	Cash l
0	67500.0	1	0	1	327181	Cash l
0	360000.0	1	0	1	333274	Cash l
0	90000.0	1	0	1	333796	Cash l

In [41]:
```
%%sql
UPDATE APPLICATION_DATA
SET OCCUPATION_TYPE='DATA NOT AVAILABLE'
WHERE OCCUPATION_TYPE IS NULL;
```

```
 * postgresql://student@/Final_Project
96391 rows affected.
```

Out[41]: []

Creating Dimension and Fact Table

Credit Bureau Dimension

Creating credit_bureau_dimension table

In [42]:
```
%%sql
DROP TABLE IF EXISTS CREDIT_BUREAU_DIMENSION CASCADE;
CREATE TABLE CREDIT_BUREAU_DIMENSION (
    Bureau_KEY SERIAL PRIMARY KEY,
    amt_req_credit_bureau_hour_group VARCHAR(50),
    amt_req_credit_bureau_day_group VARCHAR(50)
);
```

```
* postgresql://student@/Final_Project
Done.
Done.
```

Out[42]: []

Populate credit bureau dimension table with data from application_data table

In [43]:
```
%%sql
INSERT INTO CREDIT_BUREAU_DIMENSION (amt_req_credit_bureau_hour_group,amt_req_c
SELECT DISTINCT amt_req_credit_bureau_hour_group,amt_req_credit_bureau_day_grou
FROM APPLICATION_DATA;
```

```
* postgresql://student@/Final_Project
4 rows affected.
```

Out[43]: []

- Verify that wrangling steps have succeeded

In [44]:
```
%%sql
SELECT * FROM CREDIT_BUREAU_DIMENSION
```

```
* postgresql://student@/Final_Project
4 rows affected.
```

Out[44]:

bureau_key	amt_req_credit_bureau_hour_group	amt_req_credit_bureau_day_group
1	1 or more Enquiries	1 or more Enquiries
2	1 or more Enquiries	Zero Enquiry
3	Zero Enquiry	1 or more Enquiries
4	Zero Enquiry	Zero Enquiry

Mapping the Key in the application_data (fact table)

In [45]:
```
%%sql
ALTER TABLE APPLICATION_DATA
ADD COLUMN Bureau_KEY INTEGER,
ADD FOREIGN KEY(Bureau_KEY) REFERENCES CREDIT_BUREAU_DIMENSION;
```

```
* postgresql://student@/Final_Project
Done.
```

Out[45]: []

Populate the Bureau_KEY

In [46]:
```
%%sql
UPDATE APPLICATION_DATA
SET Bureau_KEY=CREDIT_BUREAU_DIMENSION.Bureau_KEY
FROM CREDIT_BUREAU_DIMENSION
WHERE APPLICATION_DATA.amt_req_credit_bureau_hour_group=CREDIT_BUREAU_DIMENSION
and APPLICATION_DATA.amt_req_credit_bureau_day_group=CREDIT_BUREAU_DIMENSION.ar
```

```
* postgresql://student@/Final_Project
307511 rows affected.
```

Out[46]: []

- Verify that your wrangling steps have succeeded

In [47]:
```
%%sql
select * from APPLICATION_DATA
limit 10;
```

```
 * postgresql://student@/Final_Project
10 rows affected.
```

Out[47]:

target	amt_income_total	flag_mobil	flag_email	flag_cont_mobile	sk_id_curr	name_contract_
0	360000.0	1	0	1	333274	Cash l
0	90000.0	1	0	1	333796	Cash l
0	225000.0	1	0	1	348486	Revolving l
0	225000.0	1	0	1	350300	Cash l
0	360000.0	1	0	1	350485	Cash l
0	67500.0	1	0	1	354448	Cash l
0	90000.0	1	0	1	363790	Cash l
0	90000.0	1	0	1	371805	Cash l
0	81000.0	1	0	1	382276	Cash l
0	211500.0	1	0	1	390020	Cash l

Customer Contact Dimension

Creating customer_contact_dimension table

In [48]:
```
%%sql
DROP TABLE IF EXISTS CUSTOMER_CONTACT_DIMENSION CASCADE;
CREATE TABLE CUSTOMER_CONTACT_DIMENSION (
    CONTACT_KEY SERIAL PRIMARY KEY,
    FLAG_MOBIL CHAR(1) NOT NULL,
    FLAG_EMAIL CHAR(1) NOT NULL,
    FLAG_CONT_MOBILE CHAR(1) NOT NULL
);
```

```
 * postgresql://student@/Final_Project
Done.
Done.
```

Out[48]: []

Populate customer contact dimension table with data from application data table

In [49]:
```
%%sql
INSERT INTO CUSTOMER_CONTACT_DIMENSION (FLAG_MOBIL,FLAG_EMAIL,FLAG_CONT_MOBILE)
SELECT DISTINCT FLAG_MOBIL,FLAG_EMAIL,FLAG_CONT_MOBILE
FROM APPLICATION_DATA;
```

```
 * postgresql://student@/Final_Project
5 rows affected.
```

Out[49]: []

- Verify that your wrangling steps have succeeded

In [50]:
```
%%sql
SELECT * FROM CUSTOMER_CONTACT_DIMENSION
```

```
 * postgresql://student@/Final_Project
5 rows affected.
```

Out[50]:

contact_key	flag_mobil	flag_email	flag_cont_mobile
1	1	0	0
2	1	1	1
3	1	1	0
4	1	0	1
5	0	0	1

Mapping the Key in the application_data

In [51]:
```
%%sql
ALTER TABLE APPLICATION_DATA
ADD COLUMN CONTACT_KEY INTEGER,
ADD FOREIGN KEY(CONTACT_KEY) REFERENCES CUSTOMER_CONTACT_DIMENSION;
```

```
 * postgresql://student@/Final_Project
Done.
```

Out[51]: []

Populate CONTACT_KEY

In [52]:
```
%%sql
UPDATE APPLICATION_DATA
SET CONTACT_KEY=CUSTOMER_CONTACT_DIMENSION.CONTACT_KEY
FROM CUSTOMER_CONTACT_DIMENSION
WHERE APPLICATION_DATA.FLAG_MOBIL=CUSTOMER_CONTACT_DIMENSION.FLAG_MOBIL
AND APPLICATION_DATA.FLAG_EMAIL=CUSTOMER_CONTACT_DIMENSION.FLAG_EMAIL
AND APPLICATION_DATA.FLAG_CONT_MOBILE=CUSTOMER_CONTACT_DIMENSION.FLAG_CONT_MOBI
```

```
 * postgresql://student@/Final_Project
307511 rows affected.
```

Out[52]: []

- Verify that your wrangling steps have succeeded

In [53]:
```
%%sql
SELECT * FROM APPLICATION_DATA
LIMIT 10;
```

```
 * postgresql://student@/Final_Project
10 rows affected.
```

Out[53]:

target	amt_income_total	flag_mobil	flag_email	flag_cont_mobile	sk_id_curr	name_contract_
0	211500.0	1	0	1	390020	Cash l
0	270000.0	1	0	1	404231	Cash l
0	360000.0	1	0	1	409190	Cash l
0	225000.0	1	0	1	434449	Cash l
0	90000.0	1	0	1	437145	Cash l
0	67500.0	1	0	1	442509	Cash l
0	202500.0	1	0	1	443021	Cash l
0	225000.0	1	0	1	450519	Cash l
0	180000.0	1	0	1	455613	Cash l
0	112500.0	1	0	1	100011	Cash l

LOAN DIMENSION

Since SK_ID_CURR is already in the original dataset, we don't need to populate another forign key to the application data. We use SK_ID_CURR as the surrogate key

Creating loan_dimension table

In [54]:
```
%%sql
DROP TABLE IF EXISTS LOAN_DIMENSION Cascade;
CREATE TABLE LOAN_DIMENSION (
    SK_ID_CURR NUMERIC(6) NOT NULL,
    NAME_CONTRACT_TYPE VARCHAR(15) NOT NULL,
    amt_credit_group VARCHAR(20),
    amt_annuity_group VARCHAR(20),
    PRIMARY KEY (SK_ID_CURR)
);
```

```
 * postgresql://student@/Final_Project
Done.
Done.
```

Out[54]: []

Populate loan dimension table with data from application_data table

In [55]:
```
%%sql
INSERT INTO LOAN_DIMENSION (SK_ID_CURR,NAME_CONTRACT_TYPE,amt_credit_group,amt_
SELECT SK_ID_CURR,NAME_CONTRACT_TYPE,amt_credit_group,amt_annuity_group
FROM APPLICATION_DATA
```

```
 * postgresql://student@/Final_Project
307511 rows affected.
```

Out[55]: []

- Verify that your wrangling steps have succeeded

In [56]:
```
%%sql
SELECT * FROM LOAN_DIMENSION
LIMIT 10;
```

```
 * postgresql://student@/Final_Project
10 rows affected.
```

Out[56]:

sk_id_curr	name_contract_type	amt_credit_group	amt_annuity_group
100011	Cash loans	>1000K	25K-35K
100002	Cash loans	250K-500K	15K-25K
100003	Cash loans	>1000K	35K-45K
100004	Revolving loans	<=250K	<=15K
100006	Cash loans	250K-500K	25K-35K
100007	Cash loans	500K-750K	15K-25K
100008	Cash loans	250K-500K	25K-35K
100009	Cash loans	>1000K	35K-45K
100010	Cash loans	>1000K	35K-45K
100012	Revolving loans	250K-500K	15K-25K

CUSTOMER DEMOGRAPHIC DIMENSION

Creating customer_demographic table

In [57]:
```
%%sql
DROP TABLE IF EXISTS CUSTOMER_DEMOGRAPHIC_DIMENSION Cascade;
CREATE TABLE CUSTOMER_DEMOGRAPHIC_DIMENSION (
    DEMOGRAPHIC_KEY SERIAL PRIMARY KEY,
    CODE_GENDER VARCHAR(3) NOT NULL,
    NAME_INCOME_TYPE VARCHAR(20) NOT NULL,
    NAME_EDUCATION_TYPE VARCHAR(29) NOT NULL,
    NAME_FAMILY_STATUS VARCHAR(20) NOT NULL,
    NAME_HOUSING_TYPE VARCHAR(19) NOT NULL,
    OCCUPATION_TYPE VARCHAR(21),
    AMT_INCOME_TOTAL_GROUP VARCHAR(20),
    CNT_CHILDREN_Group VARCHAR(20),
```

```
    Year_Birth_GROUP VARCHAR(20)
);
```

```
 * postgresql://student@/Final_Project
Done.
Done.
```

Out[57]: []

Populate customer demographic dimension with data from application data

In [58]:
```
%%sql
INSERT INTO CUSTOMER_DEMOGRAPHIC_DIMENSION (CODE_GENDER,NAME_INCOME_TYPE,NAME_E
SELECT DISTINCT CODE_GENDER,
    NAME_INCOME_TYPE,
    NAME_EDUCATION_TYPE,
    NAME_FAMILY_STATUS,
    NAME_HOUSING_TYPE,
    OCCUPATION_TYPE,
    AMT_INCOME_TOTAL_GROUP,
    CNT_CHILDREN_Group,
    Year_Birth_GROUP
FROM APPLICATION_DATA;
```

```
 * postgresql://student@/Final_Project
38826 rows affected.
```

Out[58]: []

- Verify that your wrangling steps have succeeded

In [59]:
```
%%sql
SELECT * FROM CUSTOMER_DEMOGRAPHIC_DIMENSION
LIMIT 10;
```

```
 * postgresql://student@/Final_Project
10 rows affected.
```

Out[59]:

demographic_key	code_gender	name_income_type	name_education_type	name_family_status
1	F	Businessman	Higher education	Civil marriage
2	F	Businessman	Higher education	Married
3	F	Businessman	Higher education	Single / not married
4	F	Commercial associate	Academic degree	Civil marriage
5	F	Commercial associate	Academic degree	Civil marriage
6	F	Commercial associate	Academic degree	Married
7	F	Commercial associate	Academic degree	Married
8	F	Commercial associate	Academic degree	Married
9	F	Commercial associate	Academic degree	Married
10	F	Commercial associate	Academic degree	Married

Mapping the Key in the application_data

In [60]:
```
%%sql
ALTER TABLE APPLICATION_DATA
ADD COLUMN DEMOGRAPHIC_KEY INTEGER,
ADD FOREIGN KEY(DEMOGRAPHIC_KEY)
REFERENCES CUSTOMER_DEMOGRAPHIC_DIMENSION
```

```
 * postgresql://student@/Final_Project
Done.
```

Out[60]: []

In [61]:
```
%%sql
UPDATE APPLICATION_DATA
SET DEMOGRAPHIC_KEY=CUSTOMER_DEMOGRAPHIC_DIMENSION.DEMOGRAPHIC_KEY
FROM CUSTOMER_DEMOGRAPHIC_DIMENSION
WHERE APPLICATION_DATA.CODE_GENDER=CUSTOMER_DEMOGRAPHIC_DIMENSION.CODE_GENDER
AND APPLICATION_DATA.NAME_INCOME_TYPE=CUSTOMER_DEMOGRAPHIC_DIMENSION.NAME_INCON
AND APPLICATION_DATA.NAME_EDUCATION_TYPE=CUSTOMER_DEMOGRAPHIC_DIMENSION.NAME_EI
AND APPLICATION_DATA.NAME_FAMILY_STATUS=CUSTOMER_DEMOGRAPHIC_DIMENSION.NAME_FAN
AND APPLICATION_DATA.NAME_HOUSING_TYPE=CUSTOMER_DEMOGRAPHIC_DIMENSION.NAME_HOUS
AND APPLICATION_DATA.OCCUPATION_TYPE=CUSTOMER_DEMOGRAPHIC_DIMENSION.OCCUPATION_
AND APPLICATION_DATA.AMT_INCOME_TOTAL_GROUP=CUSTOMER_DEMOGRAPHIC_DIMENSION.AMT_
AND APPLICATION_DATA.CNT_CHILDREN_Group=CUSTOMER_DEMOGRAPHIC_DIMENSION.CNT_CHII
AND APPLICATION_DATA.Year_Birth_GROUP=CUSTOMER_DEMOGRAPHIC_DIMENSION.Year_Birth
```

```
 * postgresql://student@/Final_Project
307511 rows affected.
```

Out[61]: []

- Verify that your wrangling steps have succeeded

In [62]:
```
%%sql
SELECT *
FROM APPLICATION_DATA
LIMIT 10
;
```

```
 * postgresql://student@/Final_Project
10 rows affected.
```

Out[62]:

target	amt_income_total	flag_mobil	flag_email	flag_cont_mobile	sk_id_curr	name_contract_
0	171000.0	1	0	1	100009	Cash l
0	112500.0	1	0	1	100011	Cash l
0	270000.0	1	0	1	100003	Cash l
0	135000.0	1	0	1	100006	Cash l
0	360000.0	1	0	1	100010	Cash l
0	99000.0	1	0	1	100008	Cash l
0	121500.0	1	0	1	100007	Cash l
0	135000.0	1	0	1	100012	Revolving l
0	67500.0	1	0	1	100004	Revolving l
1	202500.0	1	0	1	100002	Cash l

Fact Table

Creating Fact table and loading contents into it

In [63]:
```
%%sql
DROP TABLE IF EXISTS FACT Cascade;
CREATE TABLE FACT (
    SK_ID_CURR NUMERIC(6),
    BUREAU_KEY INTEGER,
    CONTACT_KEY INTEGER,
    DEMOGRAPHIC_KEY INTEGER,
    TARGET NUMERIC(1),
    AMT_INCOME_TOTAL NUMERIC(10),
    FOREIGN KEY (SK_ID_CURR) REFERENCES LOAN_DIMENSION,
    FOREIGN KEY (BUREAU_KEY) REFERENCES CREDIT_BUREAU_DIMENSION,
    FOREIGN KEY (CONTACT_KEY) REFERENCES CUSTOMER_CONTACT_DIMENSION,
    FOREIGN KEY (DEMOGRAPHIC_KEY) REFERENCES CUSTOMER_DEMOGRAPHIC_DIMENSION
);
```

```
 * postgresql://student@/Final_Project
Done.
Done.
```

Out[63]: []

In [64]:
```
%%sql
INSERT INTO FACT (SK_ID_CURR,BUREAU_KEY,CONTACT_KEY,DEMOGRAPHIC_KEY,TARGET,AMT_
SELECT SK_ID_CURR,BUREAU_KEY,CONTACT_KEY,DEMOGRAPHIC_KEY,TARGET,AMT_INCOME_TOT/
FROM APPLICATION_DATA;
```

```
 * postgresql://student@/Final_Project
307511 rows affected.
```

Out[64]: []

Now, check the final fact table

- Verify that wrangling steps have succeeded

In [65]:
```
%%sql
select * from FACT
LIMIT 5
;
```

```
 * postgresql://student@/Final_Project
5 rows affected.
```

Out[65]:

sk_id_curr	bureau_key	contact_key	demographic_key	target	amt_income_total
100009	4	4	529	0	171000
100011	4	4	8357	0	112500
100003	4	4	9112	0	270000
100006	4	4	18559	0	135000
100010	4	4	30039	0	360000

In [66]:
```
%%sql
select count(*)
from FACT
;
```

```
 * postgresql://student@/Final_Project
1 rows affected.
```

Out[66]:

count
307511

ASK3: Data analysis and visualization

Q1: In the provided loan data, what is the average loan default rate? What is the average credit limit on the loan that the consumer is granted? What are the loan types

along with which income levels have the higher default rate?

In [67]:
```
# Average default rate
```

In [68]:
```
%%sql
SELECT AVG(CAST(target AS DECIMAL)) AS average_default_rate
FROM FACT;
```

```
 * postgresql://student@/Final_Project
1 rows affected.
```

Out[68]:

average_default_rate
0.08072881945686495768

Average Loan Default Rate: The average loan default rate in the provided loan data is approximately 8.07%. This means that, on average, about 8.07% of the borrowers in the dataset experienced payment difficulties.

In [69]:
```
# Loan Type with highest Default, we can see that cash loan is the highest.
```

In [70]:
```
%%sql
SELECT LD.name_contract_type, AVG(CAST(F.target AS DECIMAL)) AS average_default
FROM FACT F
JOIN LOAN_DIMENSION LD ON F.sk_id_curr = LD.sk_id_curr
GROUP BY LD.name_contract_type
ORDER BY average_default_rate DESC
```

```
 * postgresql://student@/Final_Project
2 rows affected.
```

Out[70]:

name_contract_type	average_default_rate
Cash loans	0.08345912763449207855
Revolving loans	0.05478329177909081594

Loan Types and Default Rates:

For "Cash loans," the default rate is approximately 8.35%. This suggests that a relatively higher percentage of borrowers with cash loans had payment difficulties. For "Revolving loans," the default rate is approximately 5.48%. Borrowers with revolving loans had a lower default rate compared to cash loans.

In [71]:
```
# typical income group and loan contract type with default_rate
```

In [72]:
```
%%sql
select NAME_CONTRACT_TYPE, amt_income_total_group,
round(sum(TARGET)/count(*),3) as default_rate
from FACT a
left join Loan_Dimension b
on a.SK_ID_CURR=b.SK_ID_CURR
left join Customer_Demographic_Dimension c
on a.demographic_key=c.demographic_key
```

```
group by NAME_CONTRACT_TYPE, AMT_INCOME_TOTAL_GROUP
order by default_rate desc
;
```

```
 * postgresql://student@/Final_Project
8 rows affected.
```

Out[72]:

name_contract_type	amt_income_total_group	default_rate
Cash loans	100K-150K	0.089
Cash loans	150K-200K	0.088
Cash loans	<=100K	0.084
Cash loans	>200K	0.075
Revolving loans	<=100K	0.068
Revolving loans	100K-150K	0.064
Revolving loans	150K-200K	0.047
Revolving loans	>200K	0.035

Expectations vs. Results:

- The average loan default rate of approximately 8.07% is within the expected range for consumer loans, where single-digit default rates are typical.
- The finding that "Cash loans" have a higher default rate compared to "Revolving loans" aligns with the general understanding that installment loans (like cash loans) tend to have higher default rates than revolving credit (like credit cards).
- The variation in default rates across different income groups is also expected, as borrowers with higher incomes may have better financial stability and a lower likelihood of defaulting.
- Overall, the results align with common trends in the lending industry, where loan types and income levels are known factors influencing default rates.

In [73]:

```
q1=%sql select NAME_CONTRACT_TYPE, amt_income_total_group,round(sum(TARGET)/cou
```

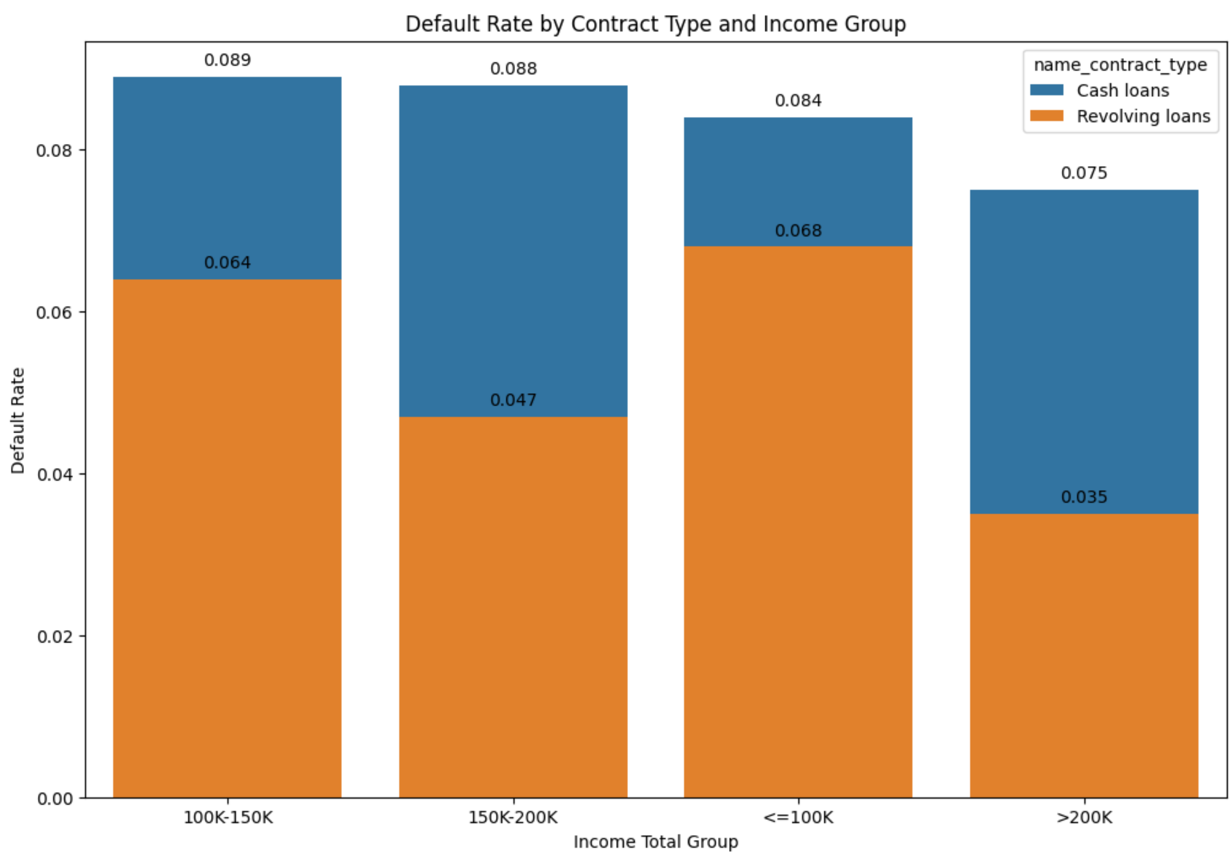
```
 * postgresql://student@/Final_Project
8 rows affected.
```

visualization

In [74]:

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Data for plotting
data = {
    'name_contract_type': ['Cash loans', 'Cash loans', 'Cash loans', 'Cash loar
                           'Revolving loans', 'Revolving loans', 'Revolving loa
    'amt_income_total_group': ['100K-150K', '150K-200K', '<=100K', '>200K',
```

```
                                  '<=100K', '100K-150K', '150K-200K', '>200K'],
    'default_rate': [0.089, 0.088, 0.084, 0.075, 0.068, 0.064, 0.047, 0.035]
}

# Convert data to DataFrame
df = pd.DataFrame(data)

# Create the plot with overlapping bars
plt.figure(figsize=(12, 8))
plot = sns.barplot(x='amt_income_total_group', y='default_rate', hue='name_cont

# Add the text (default rate percentage) on each bar
for p in plot.patches:
    height = p.get_height()
    plt.text(p.get_x() + p.get_width() / 2.,
             height + 0.001,
             f'{height:.3f}',
             ha='center',
             va='bottom')

# Setting the labels and title
plt.xlabel('Income Total Group')
plt.ylabel('Default Rate')
plt.title('Default Rate by Contract Type and Income Group')
plt.show()
```



Question 2: what are the differences between females and males regarding their average income and default

rate? Also, within each gender group, which income level shows the highest frequency of defaults?

In [75]:
```
import pandas as plt
import matplotlib.pyplot as plt
%matplotlib inline
```

In [76]:
```
# the differences between females and males regarding their average income and
```

In [77]:
```
%%sql
select CODE_GENDER,
count(*) as Number_0f_loan,
round(avg(AMT_INCOME_TOTAL),2) as Avg_INCOME,
round(sum(Target)/count(*),3) as default_rate
from FACT a
left join Customer_Demographic_Dimension b
on a.demographic_key=b.demographic_key
group by CODE_GENDER
limit 2;
```

```
 * postgresql://student@/Final_Project
2 rows affected.
```

Out[77]:

code_gender	number_0f_loan	avg_income	default_rate
F	202448	156032.31	0.070
M	105059	193396.48	0.101

Average Income:

- Females (F) have an average income of approximately 156,032.31 rupee.
- Males (M) have a higher average income of approximately 193,396.48 rupee.

In [78]:
```
# within each gender group shows the highest frequency of defaults?
```

In [79]:
```
q2 = %sql select CODE_GENDER, count(*) as Total_Credit_Cards, round(avg(AMT_IN(
```

```
 * postgresql://student@/Final_Project
2 rows affected.
```

In [80]:
```
df=q2.DataFrame()
```

In [81]:
```
df
```

Out[81]:

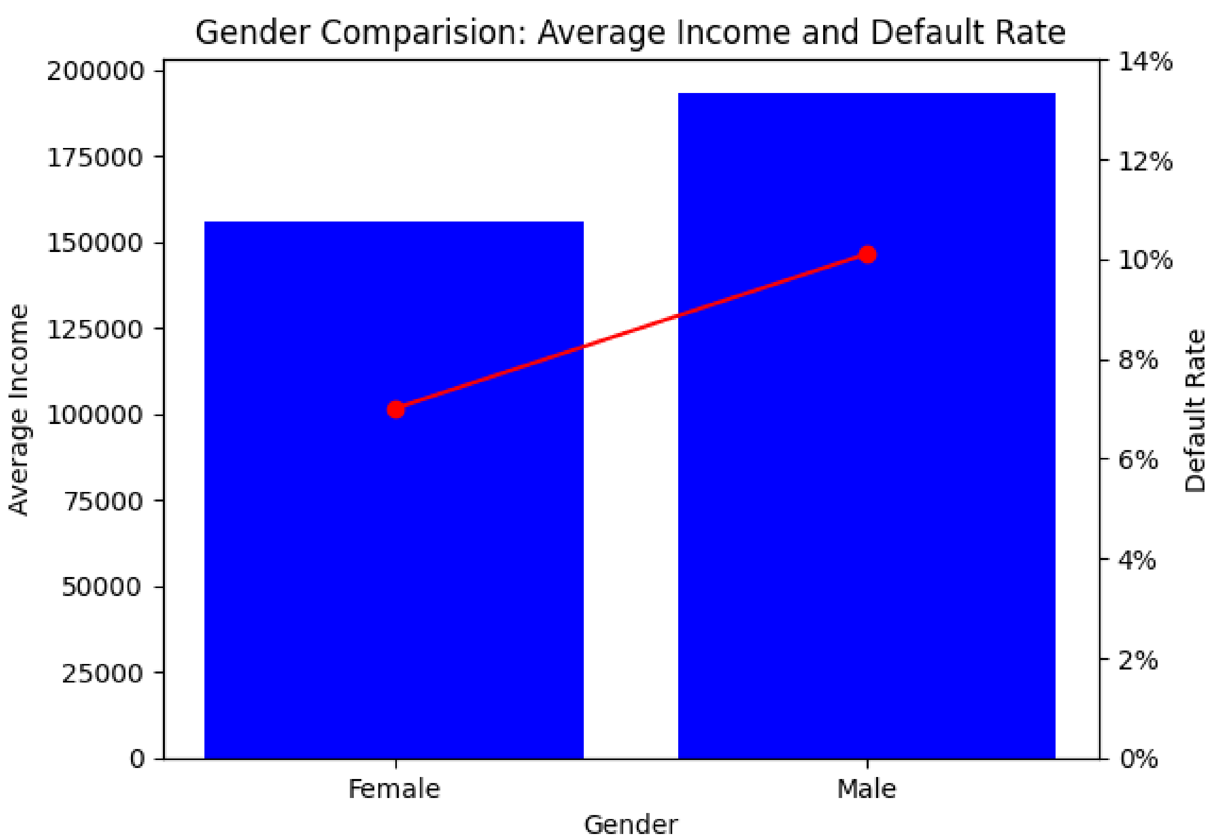
	code_gender	total_credit_cards	avg_income	default_rate
0	F	202448	156032.31	0.070
1	M	105059	193396.48	0.101

Default Rate:

- Females have a default rate of approximately 7.0% with lower income.
- Males have a higher default rate of approximately 10.1% with higher income.

visualization

In [82]:
```
fig, ax = plt.subplots()
ax2=ax.twinx()

ax.bar(df['code_gender'],df['avg_income'],color='b', label='Bar')
ax2.plot(df['code_gender'],df['default_rate'],color='r',marker='o')
plt.yticks([0.00,0.02,0.04,0.06,0.08,0.10,0.12,0.14],['0%','2%','4%','6%','8%',

plt.xticks(['F','M'],['Female','Male'])
ax.set_xlabel('Gender')
ax.set_ylabel('Average Income')
ax2.set_ylabel('Default Rate')
plt.title('Gender Comparision: Average Income and Default Rate')

plt.show()
```



In [83]:
```
# within each gender group, which income level shows the highest frequency of c
```

In [105…
```
q2_2 = %sql select AMT_INCOME_TOTAL_group,CODE_GENDER, count(*) as Total_Credit
```

```
 * postgresql://student@/Final_Project
8 rows affected.
```

```
In [106… df_2=q2_2.DataFrame()
```

```
In [107… df_2
```

Out[107]:

	amt_income_total_group	code_gender	total_credit_cards	avg_income	default_rate
0	100K-150K	F	64026	123797.13	0.0736
1	100K-150K	M	27564	125789.15	0.1154
2	150K-200K	F	39700	168905.03	0.0695
3	150K-200K	M	24606	170252.55	0.1087
4	<=100K	F	51485	77571.94	0.0743
5	<=100K	M	12213	80190.99	0.1147
6	>200K	F	47237	274422.02	0.0609
7	>200K	M	40676	287200.86	0.0836

Income Levels and Default Rates Within Gender Groups:

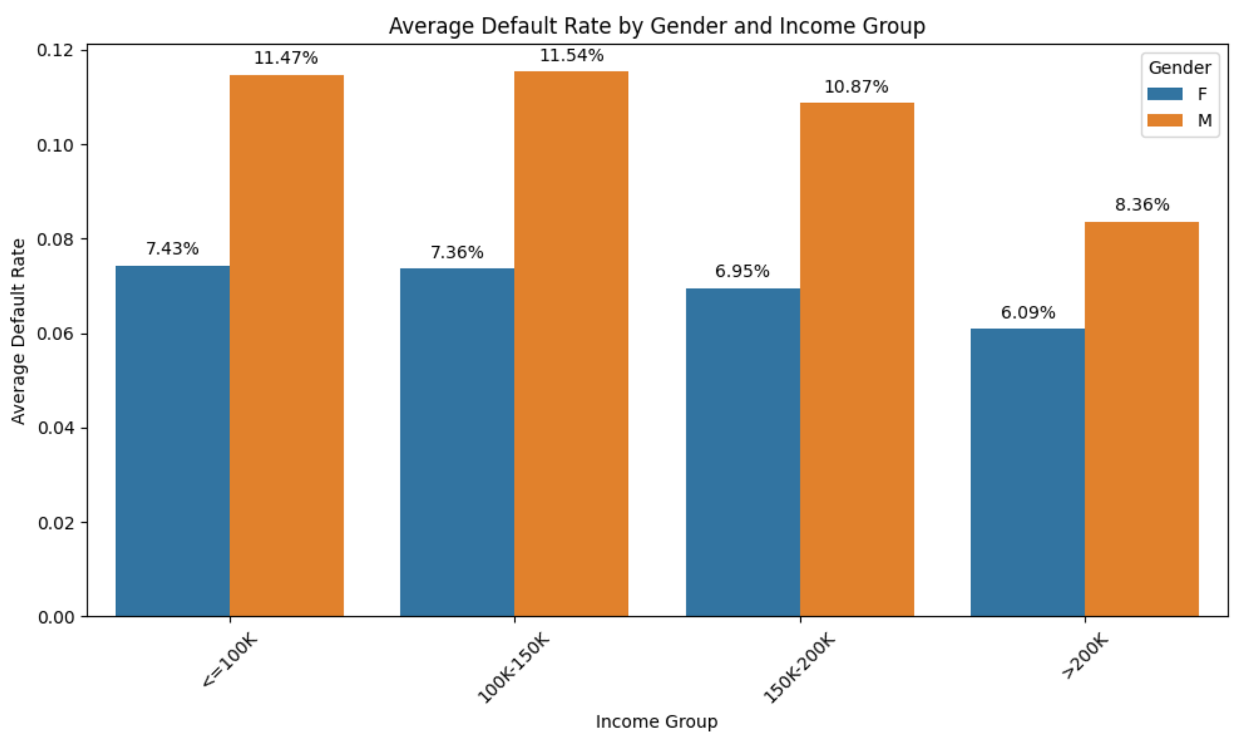
- Among Females (F):
  - The income group "100K-150K" has a default rate of approximately 7.4%.
  - The income group "150K-200K" has a default rate of approximately 6.9%.
  - The income group "<=100K" has a default rate of approximately 7.4%.
  - The income group ">200K" has the lowest default rate at approximately 6.1%.
- Among Males (M):
  - The income group "100K-150K" has the highest default rate at approximately 11.5%.
  - The income group "150K-200K" has a default rate of approximately 10.9%.
  - The income group "<=100K" has a default rate of approximately 11.5%.
  - The income group ">200K" has a lower default rate than the previous income groups at approximately 8.4%.

Expectations vs. Results:

- It is expected that males have a higher average income compared to females, as this trend is commonly observed in income disparities.
- The finding that males have a higher default rate aligns with statistical data showing that, on average, males tend to default on loans at a higher rate than females.
- Within each gender group, it is expected that higher income levels are associated with lower default rates. However, there is a notable exception among males in the "100K-150K" income group, where the default rate is higher. This could be due to various factors affecting loan repayment behavior.
- Overall, the results reflect known trends in income and default rates between genders and income levels.

In [109…

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Define the desired order of income groups
income_group_order = ['<=100K', '100K-150K', '150K-200K', '>200K']

# Convert the 'amt_income_total_group' column to a categorical type with the sp
df_2['amt_income_total_group'] = pd.Categorical(df_2['amt_income_total_group'],

# Grouping by gender and income group and calculating mean default rate
grouped_data = df_2.groupby(['code_gender', 'amt_income_total_group']).agg({'de

# Creating a bar plot
plt.figure(figsize=(10, 6))
bar_plot = sns.barplot(x='amt_income_total_group', y='default_rate', hue='code_

plt.title('Average Default Rate by Gender and Income Group')
plt.xlabel('Income Group')
plt.ylabel('Average Default Rate')
plt.xticks(rotation=45)
plt.legend(title='Gender')

# Adding annotations
for p in bar_plot.patches:
    bar_plot.annotate(format(p.get_height() * 100, '.2f') + '%',
                      (p.get_x() + p.get_width() / 2., p.get_height()),
                      ha = 'center', va = 'center',
                      xytext = (0, 9),
                      textcoords = 'offset points')

plt.tight_layout()
plt.show()
```



Q3: How can we identify and categorize the risk profiles of our loan applicants based on their income level, age group, and occupation? Specifically, which income group has the highest overall default rate, and within this group, which age bracket is most prone to defaulting? Furthermore, among the high-risk age bracket in the highest defaulting income group, what are the common occupation types, and how do their default rates compare?

In [88]:
```
# average default rate across four different income groups
```

In [89]:
```
%%sql
SELECT CDD.AMT_INCOME_TOTAL_GROUP, ROUND(AVG(CAST(F.target AS DECIMAL)),4) AS a
FROM FACT F
JOIN CUSTOMER_DEMOGRAPHIC_DIMENSION CDD ON F.demographic_key = CDD.demographic_
GROUP BY CDD.AMT_INCOME_TOTAL_GROUP
ORDER BY average_default_rate DESC;
```

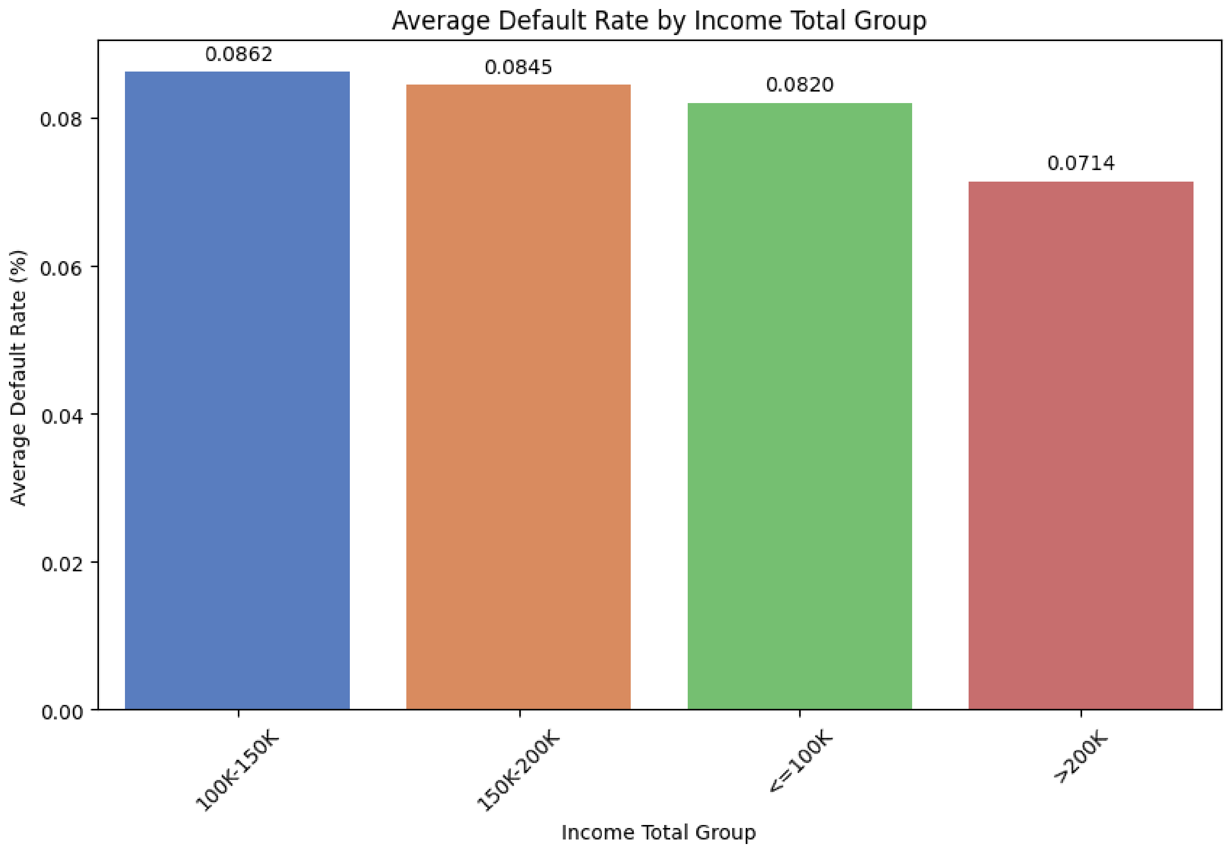
```
 * postgresql://student@/Final_Project
4 rows affected.
```

Out[89]:

amt_income_total_group	average_default_rate
100K-150K	0.0862
150K-200K	0.0845
<=100K	0.0820
>200K	0.0714

In [90]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Sample data
data = {
    'amt_income_total_group': ['100K-150K', '150K-200K', '<=100K', '>200K'],
    'average_default_rate': [0.0862, 0.0845, 0.0820, 0.0714]
}

df = pd.DataFrame(data)

# Creating the plot with percentage numbers on top of each bar
plt.figure(figsize=(10, 6))
ax = sns.barplot(x='amt_income_total_group', y='average_default_rate', data=df,

# Adding the text on top of each bar
for p in ax.patches:
    ax.annotate(f'{p.get_height():.4f}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha = 'center',
                va = 'center',
```

```
                xytext = (0, 9),
                textcoords = 'offset points')

plt.xlabel('Income Total Group')
plt.ylabel('Average Default Rate (%)')
plt.title('Average Default Rate by Income Total Group')
plt.xticks(rotation=45)
plt.show()
```



- Income Level and Default Risk:

Clients with higher incomes (>200K) appear to have a lower default rate (0.0713) compared to other groups. This suggests that higher income might be associated with a lower likelihood of payment difficulties, possibly due to better financial stability.

- Vulnerability of Middle-Income Groups:

The middle-income groups, particularly those earning between 100K-150K and 150K-200K, show higher default rates (0.0862 and 0.0845 respectively). This could indicate that these groups are more vulnerable to financial challenges leading to payment difficulties, possibly due to lifestyle, debt burdens, or other financial commitments.

- Lowest Income Group's Resilience:

Interestingly, the group with the lowest income (<=100K) does not have the highest default rate; in fact, its default rate is lower (0.0820) than the middle-income groups. This might reflect more cautious borrowing behavior, less access to high credit amounts, or effective financial management strategies within this group.

In [91]:
```
# break down the income group "100K-150K" by age group
```

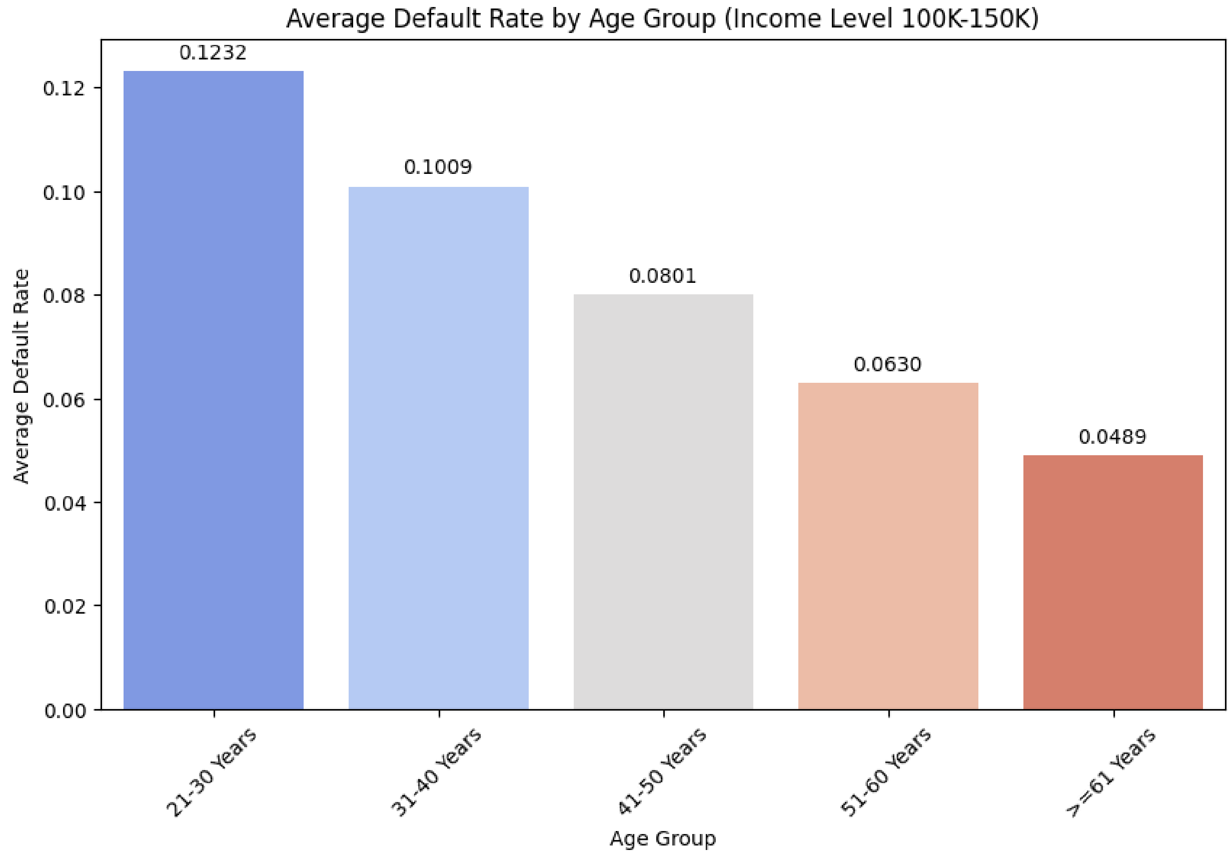
In [92]:
```
%%sql
SELECT CDD.Year_Birth_GROUP, ROUND(AVG(CAST(F.target AS DECIMAL)),4) AS average
FROM FACT F
JOIN CUSTOMER_DEMOGRAPHIC_DIMENSION CDD ON F.demographic_key = CDD.demographic_
WHERE CDD.AMT_INCOME_TOTAL_GROUP = '100K-150K'
GROUP BY CDD.Year_Birth_GROUP
ORDER BY average_default_rate DESC;
```

```
 * postgresql://student@/Final_Project
5 rows affected.
```

Out[92]:

year_birth_group	average_default_rate
21-30 Years	0.1232
31-40 Years	0.1009
41-50 Years	0.0801
51-60 Years	0.0630
>=61 Years	0.0489

In [93]:
```
# New data for age groups and their average default rates
age_data = {
    'year_birth_group': ['21-30 Years', '31-40 Years', '41-50 Years', '51-60 Ye
    'average_default_rate': [0.1232, 0.1009, 0.0801, 0.0630, 0.0489]
}

# Creating a DataFrame
age_df = pd.DataFrame(age_data)

# Creating the plot with percentage numbers on top of each bar
plt.figure(figsize=(10, 6))
ax = sns.barplot(x='year_birth_group', y='average_default_rate', data=age_df, p

# Adding the text on top of each bar
for p in ax.patches:
    ax.annotate(f'{p.get_height():.4f}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha = 'center',
                va = 'center',
                xytext = (0, 9),
                textcoords = 'offset points')

plt.xlabel('Age Group')
plt.ylabel('Average Default Rate')
plt.title('Average Default Rate by Age Group (Income Level 100K-150K)')
plt.xticks(rotation=45)
plt.show()
```



- Higher Default Rates in Younger Age Groups:

The highest default rate is observed in the youngest age group, "21-30 Years" (0.1232), which suggests that younger clients in this income bracket may have a higher likelihood of payment difficulties. This could be due to factors like less established credit history, higher likelihood of unstable employment, or other financial commitments such as student loans.

- Decreasing Default Rate with Age:

There is a clear trend of decreasing default rates with increasing age. The "31-40 Years" group has a default rate of 0.1009, which drops further for "41-50 Years" (0.0801), "51-60 Years" (0.0630), and is lowest for the ">=61 Years" group (0.0489). This trend could indicate increased financial stability, better credit management, or more conservative borrowing behavior in older age groups.

- Financial Planning and Risk Management:

Financial institutions might use this information for risk assessment, tailoring loan products, or financial advice to younger clients, particularly those between 21 and 40 years of age. This data could also guide targeted interventions or educational programs focusing on credit management and financial planning for younger customers.

In [94]:
```
%%sql
SELECT CDD.OCCUPATION_TYPE, ROUND(AVG(CAST(F.TARGET AS DECIMAL)),3) AS average_
FROM FACT F
```

```
JOIN CUSTOMER_DEMOGRAPHIC_DIMENSION CDD ON F.demographic_key = CDD.demographic_
WHERE CDD.Year_Birth_GROUP = '21-30 Years'and AMT_INCOME_TOTAL_GROUP = '100K-15
GROUP BY CDD.OCCUPATION_TYPE
ORDER BY average_default_rate DESC;
```

```
 * postgresql://student@/Final_Project
19 rows affected.
```

Out[94]:

occupation_type	average_default_rate
Cleaning staff	0.235
Low-skill Laborers	0.221
Security staff	0.156
Cooking staff	0.151
Laborers	0.150
Waiters/barmen staff	0.144
HR staff	0.143
Drivers	0.137
Sales staff	0.134
DATA NOT AVAILABLE	0.116
Secretaries	0.111
Realty agents	0.104
Managers	0.098
Private service staff	0.095
Medicine staff	0.090
High skill tech staff	0.090
Core staff	0.088
IT staff	0.075
Accountants	0.066

- Higher Risk Occupations:

'Cleaning staff' and 'Low-skill Laborers' have the highest default rates at 0.235 and 0.221, respectively. This suggests that occupations typically associated with lower income and potentially less job security are at a higher risk of loan defaults.

- Moderate Risk Occupations:

Occupations such as 'Security staff', 'Cooking staff', 'Laborers', and 'Waiters/barmen staff' have moderate default rates, ranging from 0.144 to 0.156. These jobs might offer more stability than the highest risk occupations but still show a significant level of risk.

- Lower Risk Occupations:

'Managers', 'Private service staff', 'Medicine staff', 'High skill tech staff', 'Core staff', 'IT staff', and 'Accountants' exhibit lower default rates, with 'Accountants' showing the lowest at 0.066. This could indicate that these occupations are generally more stable, come with higher income, and the individuals in these roles have a better track record of managing their finances effectively. Data Not Available:

The category 'DATA NOT AVAILABLE' has a relatively low default rate of 0.116, suggesting that even without specific occupational data, this group as a whole represents a lower credit risk than several specific occupations.

- Professional Occupations:

Professional and technical occupations, such as 'IT staff' and 'Accountants', tend to have lower default rates, which aligns with the possibility that higher education and specialized skills could lead to better financial stability.

- Implications for Lending:

Lenders might use this information to adjust their credit risk models, possibly assigning different risk weights to borrowers based on occupation. The bank could also consider tailoring financial products, advice, or support services to higher-risk occupations to help mitigate the risk of default.

Expectations vs. Results:

- It is expected that lower-income groups have higher default rates, as they may have more financial constraints.
- The findings align with expectations, as the "100K-150K" income group has the highest default rate.

Younger age groups are typically associated with higher risk, which is confirmed by the high default rate in the "21-30 Years" age group.

- Occupations that require lower skills and may have lower income levels tend to have higher default rates, consistent with common industry knowledge.
- Within the high-risk "21-30 Years" age group, occupations that often have less stable employment, such as "Cleaning staff" and "Low-skill Laborers," show the highest default rates.

In [95]:
```
import seaborn as sns

# Data for plotting
occupation_types = [
    "Cleaning staff", "Low-skill Laborers", "Security staff", "Cooking staff",
    "Laborers", "Waiters/barmen staff", "HR staff", "Drivers",
    "Sales staff", "DATA NOT AVAILABLE", "Secretaries", "Realty agents",
    "Managers", "Private service staff", "Medicine staff", "High skill tech sta
    "Core staff", "IT staff", "Accountants"
]
```

```
average_default_rates = [
    0.235, 0.221, 0.156, 0.151, 0.150, 0.144, 0.143, 0.137,
    0.134, 0.116, 0.111, 0.104, 0.098, 0.095, 0.090, 0.090,
    0.088, 0.075, 0.066
]

# Convert data to DataFrame
df = pd.DataFrame({
    'Occupation Type': occupation_types,
    'Average Default Rate': average_default_rates
})

# Sort the DataFrame by default rate for better visualization
df = df.sort_values('Average Default Rate', ascending=False)


# Create the bar chart using seaborn with percentages
plt.figure(figsize=(12, 10))
plot = sns.barplot(x='Average Default Rate', y='Occupation Type', data=df, pale

# Add the text (percentage) on each bar
for p in plot.patches:
    width = p.get_width()
    plt.text(width + 0.005,  # position of text
             p.get_y() + p.get_height() / 2,  # y-position
             f'{width:.4f}',  # text to display
             ha = 'left',  # horizontal alignment
             va = 'center')  # vertical alignment

plt.show()
```