‘ We start by importing essential Python libraries for data handling and manipulation.

- `pandas` for structured data operations.
- `numpy` for numerical operations.
- `os` for interacting with the operating system and directory structures.

```
import os
import pandas as pd
import numpy as np
```

Define and Create Directory Paths

To ensure reproducibility and organized storage, we programmatically create directories for:

- **raw data**
- **processed data**
- **results**
- **documentation**

These directories will store intermediate and final outputs for reproducibility.

```
#Get working directory
current_dir = os.getcwd()
#go one directory up to root directory
project_root_dir = os.path.dirname(current_dir)
#Define path to data files
data_dir = os.path.join(project_root_dir, 'data')
raw_dir = os.path.join(data_dir, 'raw')
processed_dir = os.path.join(data_dir, 'processed')
#Define path to results folder
results_dir = os.path.join(project_root_dir, 'results')
```

```
#Define path to results folder
docs_dir = os.path.join(project_root_dir, 'docs')

#Create directories if they do not exist
os.makedirs(raw_dir, exist_ok=True)
os.makedirs(processed_dir, exist_ok=True)
os.makedirs(results_dir, exist_ok=True)
os.makedirs(docs_dir, exist_ok=True)
```

Read in the data

We load the **Credit Card dataset** and **Customer dataset**as a CSV file.

Key considerations here are:

- We treat `?` as missing values (`na_values = '?'`).
- We use `skipinitialspace = True` to remove extra spaces after delimiters, which is common in text-based datasets.

After loading, we inspect tcolumns for each dataset

```
credit_card = os.path.join(raw_dir, "credit_card.csv")
credit_card_df = pd.read_csv(credit_card, na_values="?", skipinitialspace=True)

customer = os.path.join(raw_dir, "customer.csv")
customer_df = pd.read_csv(customer, na_values="?", skipinitialspace=True)
```

```
print(credit_card_df.columns)
print(customer_df.columns)
```

```
Index(['Client_Num', 'Card_Category', 'Annual_Fees', 'Activation_30_Days',
       'Customer_Acq_Cost', 'Week_Start_Date', 'Week_Num', 'Qtr',
       'current_year', 'Credit_Limit', 'Total_Revolving_Bal',
       'Total_Trans_Amt', 'Total_Trans_Vol', 'Avg_Utilization_Ratio',
       'Use Chip', 'Exp Type', 'Interest_Earned', 'Delinquent_Acc'],
      dtype='object')
Index(['Client_Num', 'Customer_Age', 'Gender', 'Dependent_Count',
       'Education_Level', 'Marital_Status', 'state_cd', 'Zipcode', 'Car_Owner',
       'House_Owner', 'Personal_loan', 'contact', 'Customer_Job', 'Income',
       'Cust_Satisfaction_Score'],
      dtype='object')
```

Merge the datasets

- Purpose: Combines both datasets using `Client_Num` as the key.
- inner join ensures only matching records from both datasets are kept.

```python
merged_df = pd.merge(credit_card_df, customer_df, on="Client_Num", how="inner")
```

```python
merged_df
```

	Client_Num	Card_Category	Annual_Fees	Activation_30_Days	Customer_Acq_Cost	Week_
0	708082083	Blue	200	0	87	01-01-
1	708083283	Blue	445	1	108	01-01-
2	708084558	Blue	140	0	106	01-01-
3	708085458	Blue	250	1	150	01-01-
4	708086958	Blue	320	1	106	01-01-
...	...	...	...	...	...	...
10103	827695683	Blue	340	1	106	24-12-
10104	827703258	Blue	395	1	104	24-12-
10105	827712108	Blue	125	1	107	24-12-
10106	827888433	Blue	410	0	96	24-12-
10107	827890758	Blue	100	0	43	24-12-

We also inspect the dataset's shape. We see that the data has *32,561* rows and *15* columns

```python
merged_df.shape
```

```
(10108, 32)
```

In addition, we check the data types using .info.

```python
merged_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10108 entries, 0 to 10107
Data columns (total 32 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Client_Num                10108 non-null  int64
```

```
 1   Card_Category           10108 non-null  object
 2   Annual_Fees             10108 non-null  int64
 3   Activation_30_Days      10108 non-null  int64
 4   Customer_Acq_Cost       10108 non-null  int64
 5   Week_Start_Date         10108 non-null  object
 6   Week_Num                10108 non-null  object
 7   Qtr                     10108 non-null  object
 8   current_year            10108 non-null  int64
 9   Credit_Limit            10108 non-null  float64
 10  Total_Revolving_Bal     10108 non-null  int64
 11  Total_Trans_Amt         10108 non-null  int64
 12  Total_Trans_Vol         10108 non-null  int64
 13  Avg_Utilization_Ratio   10108 non-null  float64
 14  Use Chip                10108 non-null  object
 15  Exp Type                10108 non-null  object
 16  Interest_Earned         10108 non-null  float64
 17  Delinquent_Acc          10108 non-null  int64
 18  Customer_Age            10108 non-null  int64
 19  Gender                  10108 non-null  object
 20  Dependent_Count         10108 non-null  int64
 21  Education_Level         10108 non-null  object
 22  Marital_Status          10108 non-null  object
 23  state_cd                10108 non-null  object
 24  Zipcode                 10108 non-null  int64
 25  Car_Owner               10108 non-null  object
 26  House_Owner             10108 non-null  object
 27  Personal_loan           10108 non-null  object
 28  contact                 10108 non-null  object
 29  Customer_Job            10108 non-null  object
 30  Income                  10108 non-null  int64
 31  Cust_Satisfaction_Score  10108 non-null  int64
dtypes: float64(3), int64(14), object(15)
memory usage: 2.5+ MB
```

Data Cleaning

1. Understanding Datasets

Table 1: summary of dataset columns

Column Name	*Description*	*Type*	*Values / Range*
Client_Num	Unique identifier for each customer	Categorical	Unique alphanumeric
Card_Category	Type of credit card held	Categorical	Blue, Silver, Gold, Platinum
Annual_Fees	Annual fee charged for the credit card	Numeric	0 – several hundred (currency units)
Activation_30_Days	Card activated within 30 days (1 = Yes, 0 = No)	Numeric	0, 1
Customer_Acq_Cost	Cost to acquire the customer	Numeric	Positive numeric
Week_Start_Date	Start date of the observation week	Categorical	YYYY-MM-DD format
Week_Num	Week number label	Categorical	Week-1 to Week-52
Qtr	Financial quarter	Categorical	Q1, Q2, Q3, Q4
current_year	Year of data collection	Numeric	2020, 2021, etc.
Credit_Limit	Maximum credit limit	Numeric	Positive numeric
Total_Revolving_Bal	Revolving balance from past months	Numeric	Positive numeric
Total_Trans_Amt	Total transaction amount	Numeric	Positive numeric
Total_Trans_Vol	Total number of transactions	Numeric	Positive integer
Avg_Utilization_Ratio	Credit utilization ratio	Numeric	0.0 – 1.0
Use Chip	Whether card has a chip	Categorical	Yes, No
Exp Type	Type of expense	Categorical	Travel, Shopping, Food, etc.
Interest_Earned	Interest earned by customer	Numeric	Positive numeric
Delinquent_Acc	Number of delinquent accounts	Numeric	0 and above
Customer_Age	Age of the customer	Numeric	18 – 90+
Gender	Gender of the customer	Categorical	Male, Female
Dependent_Count	Number of dependents	Numeric	0 – 10+
Education_Level	Highest education level	Categorical	High School, Graduate, PhD, etc.
Marital_Status	Marital status	Categorical	Single, Married, Divorced, etc.
state_cd	Customer state code	Categorical	Two-letter state codes
Car_Owner	Car ownership	Categorical	Yes, No
House_Owner	House ownership	Categorical	Yes, No
Personal_loan	Has a personal loan	Categorical	Yes, No
Customer_Job	Occupation or job type	Categorical	Businessman, Selfemployed, etc.
Income	Annual income	Numeric	In currency units (e.g., 10,000 – 100,000+)
Cust_Satisfaction_Score	Satisfaction score	Numeric	1 – 5
Month	Month of the activity	Categorical	Jan, Feb, ..., Dec

2. Deal with missing values and Duplicated Values

Duplicates can distort statistical summaries and model performance. Using `.duplicated().sum()`, we count duplicate records.

```
merged_df.duplicated().sum()
```

```
0
```

We inspect to ensure we don't have any duplicated values. using `isna().sum()`

```
merged_df.isna().sum()
```

```
Client_Num                0
Card_Category             0
Annual_Fees               0
Activation_30_Days        0
Customer_Acq_Cost         0
Week_Start_Date           0
Week_Num                  0
Qtr                       0
current_year              0
Credit_Limit              0
Total_Revolving_Bal       0
Total_Trans_Amt           0
Total_Trans_Vol           0
Avg_Utilization_Ratio     0
Use Chip                  0
Exp Type                  0
Interest_Earned           0
Delinquent_Acc            0
Customer_Age              0
Gender                    0
Dependent_Count           0
Education_Level           0
Marital_Status            0
state_cd                  0
Zipcode                   0
Car_Owner                 0
House_Owner               0
Personal_loan             0
```

```
contact                    0
Customer_Job               0
Income                     0
Cust_Satisfaction_Score    0
dtype: int64
```

3. Standardized Categorical Variables

Remove any leading or trailing spaces and convert the strings to lowercase

To prepare categorical variables for consistent processing, we first of all remove extra spaces and convert them to lowercase. This step ensures categorical variables are clean and consistently organized.

```
merged_df.dtypes == object
```

```
Client_Num                 False
Card_Category               True
Annual_Fees                False
Activation_30_Days         False
Customer_Acq_Cost          False
Week_Start_Date             True
Week_Num                    True
Qtr                         True
current_year               False
Credit_Limit               False
Total_Revolving_Bal        False
Total_Trans_Amt            False
Total_Trans_Vol            False
Avg_Utilization_Ratio      False
Use Chip                    True
Exp Type                    True
Interest_Earned            False
Delinquent_Acc             False
Customer_Age               False
Gender                      True
Dependent_Count            False
Education_Level             True
Marital_Status              True
state_cd                    True
Zipcode                    False
Car_Owner                   True
```

```
House_Owner                  True
Personal_loan                True
contact                      True
Customer_Job                 True
Income                      False
Cust_Satisfaction_Score     False
dtype: bool
```

```
merged_df.columns[(merged_df.dtypes == object)]
```

```
Index(['Card_Category', 'Week_Start_Date', 'Week_Num', 'Qtr', 'Use Chip',
       'Exp Type', 'Gender', 'Education_Level', 'Marital_Status', 'state_cd',
       'Car_Owner', 'House_Owner', 'Personal_loan', 'contact', 'Customer_Job'],
      dtype='object')
```

```
categorical_cols = merged_df.columns[merged_df.dtypes == object]
for col in categorical_cols:
    merged_df.loc[:,col] = merged_df[col].str.strip().str.lower()
```

```
merged_df.head(5)
```

	Client_Num	Card_Category	Annual_Fees	Activation_30_Days	Customer_Acq_Cost	Week_Sta
0	708082083	blue	200	0	87	01-01-2023
1	708083283	blue	445	1	108	01-01-2023
2	708084558	blue	140	0	106	01-01-2023
3	708085458	blue	250	1	150	01-01-2023
4	708086958	blue	320	1	106	01-01-2023

Re-code `Gender` column

We re-coded the Gender column to replace shorthand values with more descriptive labels, improving clarity and consistency. Table 2 shows the updated encoding:

Table 2: Re-encoding of the `Gender` column | Old Gender | New Gender | | :———- | :———- | | `f` | `Female` | | `m` | `Male` |

```
merged_df['Gender'].unique()
```

```
array(['f', 'm'], dtype=object)
```

```python
merged_df.loc[:, "Gender"] = merged_df["Gender"].replace({
    "f": "Female",
    "m": "Male",
})
```

```python
merged_df['Gender'].unique()
```

```
array(['Female', 'Male'], dtype=object)
```

create `month` column

To analyze trends over time more effectively, we extracted the month name from the `Week_Start_Date` column. First, we converted the `Week_Start_Date` to a proper date-time format using `pd.to_datetime()`. Then, we derived the full month name using `.dt.strftime("%B")`. This transformation helps in grouping and visualizing data by calendar months.

```python
merged_df["Week_Start_Date"] = pd.to_datetime(merged_df["Week_Start_Date"], format="%d-%m-%Y"
```

```python
merged_df["Month"] = merged_df["Week_Start_Date"].dt.strftime("%B")
```

```python
merged_df['Month'].unique()
```

```
array(['January', 'February', 'March', 'April', 'May', 'June', 'July',
       'August', 'September', 'October', 'November', 'December'],
      dtype=object)
```

Re-code the state_cd column

We re-coded the `state_cd` column to replace two-letter state abbreviations with full U.S. state names. This improves readability and helps non-technical stakeholders understand the data more easily. Table 3 shows a sample of the new encoding:

Table 3: Re-encoding of the `state_cd` column

Old State	New State
CA	California
TX	Texas

Old State	New State
NY	New York
FL	Florida
IL	Illinois
…	…

```
merged_df["state_cd"] = merged_df["state_cd"].str.upper()
merged_df['state_cd'].unique()
```

```
array(['FL', 'NJ', 'NY', 'TX', 'CA', 'MO', 'MA', 'IA', 'AK', 'MI', 'GA',
       'CT', 'IL', 'VA', 'UT', 'HI', 'AZ', 'WA', 'NV', 'CO', 'MN', 'AR',
       'PA', 'OR', 'OH', 'NM', 'SC', 'NE'], dtype=object)
```

```
merged_df["state_cd"] = merged_df["state_cd"].replace({
    "AL": "Alabama", "AK": "Alaska", "AZ": "Arizona", "AR": "Arkansas", "CA": "California",
    "CO": "Colorado", "CT": "Connecticut", "DE": "Delaware", "FL": "Florida", "GA": "Georgia'
    "HI": "Hawaii", "ID": "Idaho", "IL": "Illinois", "IN": "Indiana", "IA": "Iowa",
    "KS": "Kansas", "KY": "Kentucky", "LA": "Louisiana", "ME": "Maine", "MD": "Maryland",
    "MA": "Massachusetts", "MI": "Michigan", "MN": "Minnesota", "MS": "Mississippi", "MO": "I
    "MT": "Montana", "NE": "Nebraska", "NV": "Nevada", "NH": "New Hampshire", "NJ": "New Jers
    "NM": "New Mexico", "NY": "New York", "NC": "North Carolina", "ND": "North Dakota", "OH"
    "OK": "Oklahoma", "OR": "Oregon", "PA": "Pennsylvania", "RI": "Rhode Island", "SC": "Sou
    "SD": "South Dakota", "TN": "Tennessee", "TX": "Texas", "UT": "Utah", "VT": "Vermont",
    "VA": "Virginia", "WA": "Washington", "WV": "West Virginia", "WI": "Wisconsin", "WY": "Wy
})
```

```
merged_df['state_cd'].unique()
```

```
array(['Florida', 'New Jersey', 'New York', 'Texas', 'California',
       'Missouri', 'Massachusetts', 'Iowa', 'Alaska', 'Michigan',
       'Georgia', 'Connecticut', 'Illinois', 'Virginia', 'Utah', 'Hawaii',
       'Arizona', 'Washington', 'Nevada', 'Colorado', 'Minnesota',
       'Arkansas', 'Pennsylvania', 'Oregon', 'Ohio', 'New Mexico',
       'South Carolina', 'Nebraska'], dtype=object)
```

Displayed Merged Dataset

```
merged_df
```

	Client_Num	Card_Category	Annual_Fees	Activation_30_Days	Customer_Acq_Cost	Week_
0	708082083	blue	200	0	87	2023-0
1	708083283	blue	445	1	108	2023-0
2	708084558	blue	140	0	106	2023-0
3	708085458	blue	250	1	150	2023-0
4	708086958	blue	320	1	106	2023-0
...	...	...	...	...	...	...
10103	827695683	blue	340	1	106	2023-1
10104	827703258	blue	395	1	104	2023-1
10105	827712108	blue	125	1	107	2023-1
10106	827888433	blue	410	0	96	2023-1
10107	827890758	blue	100	0	43	2023-1

Create age group based on the `Customer_Age` column

Age is binned into groups such as `<21-25`, `26-35`, ···, `75` to facilitate easier demographic analysis.

```
merged_df['Customer_Age'].unique()
```

```
array([24, 62, 32, 38, 48, 33, 34, 53, 31, 51, 36, 49, 47, 43, 37, 63, 55,
       40, 54, 46, 44, 60, 52, 42, 58, 57, 39, 50, 26, 41, 61, 45, 56, 29,
       59, 27, 35, 65, 28, 64, 30, 23, 22, 25, 73, 21, 68, 67, 70, 66],
      dtype=int64)
```

```
bins = [20, 25, 35, 45, 55, 65, 75]
labels = ['21-25', '26-35', '36-45', '46-55', '56-65', '66-75']

merged_df['Customer_Age'] = pd.cut(merged_df['Customer_Age'], bins=bins, labels=labels, righ
```

```
merged_df['Customer_Age'].unique()
```

```
['21-25', '56-65', '26-35', '36-45', '46-55', '66-75']
Categories (6, object): ['21-25' < '26-35' < '36-45' < '46-55' < '56-65' < '66-75']
```

Drop unnecessary columns

After recoding, some columns such as `Zipcode`,and `contact` become redundant. We drop them to avoid multicollinearity and simplify our dataset.

```
merged_df.drop(columns=["Zipcode", "contact"], inplace=True)
```

```
merged_df
```

	Client_Num	Card_Category	Annual_Fees	Activation_30_Days	Customer_Acq_Cost	Week_
0	708082083	blue	200	0	87	2023-0
1	708083283	blue	445	1	108	2023-0
2	708084558	blue	140	0	106	2023-0
3	708085458	blue	250	1	150	2023-0
4	708086958	blue	320	1	106	2023-0
...	...	...	...	...	...	...
10103	827695683	blue	340	1	106	2023-1
10104	827703258	blue	395	1	104	2023-1
10105	827712108	blue	125	1	107	2023-1
10106	827888433	blue	410	0	96	2023-1
10107	827890758	blue	100	0	43	2023-1

Save the Clean Dataset

Before saving the clean dataset, we re-inspect it to ensure no new issues have risen up due to re-encoding. We first of all inspect the shape of the dataset. We see that we have *10108* rows and *31* columns. This means that there is a new column, `Month`, added to the original dataset.

```
merged_df.shape
```

```
(10108, 31)
```

```
merged_df.columns
```

```
Index(['Client_Num', 'Card_Category', 'Annual_Fees', 'Activation_30_Days',
       'Customer_Acq_Cost', 'Week_Start_Date', 'Week_Num', 'Qtr',
       'current_year', 'Credit_Limit', 'Total_Revolving_Bal',
```

```
       'Total_Trans_Amt', 'Total_Trans_Vol', 'Avg_Utilization_Ratio',
       'Use Chip', 'Exp Type', 'Interest_Earned', 'Delinquent_Acc',
       'Customer_Age', 'Gender', 'Dependent_Count', 'Education_Level',
       'Marital_Status', 'state_cd', 'Car_Owner', 'House_Owner',
       'Personal_loan', 'Customer_Job', 'Income', 'Cust_Satisfaction_Score',
       'Month'],
      dtype='object')
```

Save Merged dataset

Finally, we save the clean, processed dataset as a CSV file in our processed directory for future modelling and analysis.

```
merged_df.to_csv("Credit_Card_Financial.csv", index=False)
clean_filename = os.path.join(processed_dir, "Credit_Card_Financial.csv")
merged_df.to_csv(clean_filename, index=False)
print(f"\nCleaned data saved to: {clean_filename}")
```

```

Cleaned data saved to: C:\Users\user\Documents\tekHer\LookerStudio\Credit-Card-Financial\data
```