Jupyter short keys

These are the most common used short keys:

- Esc + a -- Open a new cell above
- Esc + b -- Open a new cell below
- Shift + Enter -- Run the cell, and go to next one
- Ctrl + Enter -- Run the cell and stay there
- Alt + Enter -- Run the cell, and make a new one (Option-key for Mac)
- Ctrl + / -- Comment or Uncomment your selection
- Esc + m -- Change cell type into Markdown (text modus). Then run again with Crtl+Enter
- Esc + y -- Change cell type into Code (code modus). Then run again with Crtl+Enter

Let's try them out and get familiar.

Project Overview: Customer Data Cleaning, Analysis, and Visualization

In this project, you will work with a customer dataset and perform various **data cleaning**, **analysis**, and **visualization** tasks. The goal is to clean and standardize the data, conduct basic analysis, and visualize key trends and relationships in the data.

Objectives:

- **Data Cleaning**: Handle missing values, standardize columns, fix inconsistencies, and remove duplicates.
- **Data Analysis**: Perform basic calculations such as total revenue and find insights from the dataset (e.g., average customer age).
- **Data Visualization**: Create visualizations using Seaborn to explore trends and relationships between different variables.

Dataset Overview:

The dataset contains customer information with the following key columns:

- `CustomerID` : Unique identifier for each customer.
- `Name` : Name of the customer.
- `Phone_Number` : Contact number of the customer (may contain inconsistencies).
- `Email` : Email address of the customer (may contain missing values).
- `Department` : The department where the customer is associated.
- `Experience` : Number of years of experience of the customer.
- `Age` : Age of the customer (may contain unrealistic values).
- `Registration_Date` : Date when the customer registered (may have inconsistent formats).
- `Customer_Type` : Type of customer (e.g., Individual, VIP) with potential typos.
- `Total_Purchases` : Number of purchases made by the customer (may contain missing values).
- `Last_Purchase_Amount` : Amount spent on the last purchase (may contain dollar signs and commas).
- `Feedback_Score` : Customer feedback score on a scale of 1-10 (may contain missing values).
- `Gender` : Gender of the customer (may contain inconsistent values).

Project Structure:

1. **Data Cleaning Tasks**:
   You will handle missing data, standardize columns, remove duplicates, and correct inconsistencies in various columns such as `Phone_Number` , `Gender` , `Customer_Type` , and `Last_Purchase_Amount` .

2. **Data Analysis**:
   Perform analysis tasks such as calculating total revenue per customer and determining the average age of high-satisfaction customers.

3. **Data Visualization**:
   Create visualizations to explore trends in the dataset, such as the distribution of customer ages, the relationship between age and total purchases, and the average number of purchases by customer type.

Instructions:

- Each question is structured step by step. Follow the instructions for each task and write your code in the provided cells.

- Be sure to run each cell after writing your solution.
- For visualization tasks, use **Seaborn** and **Matplotlib** to create the required plots.

Good Luck!

This project will give you practical experience with data cleaning, analysis, and visualization in Python. You will learn how to handle real-world messy data and extract meaningful insights from it.

Data Storage Notes

1. **`df`** : Original dataset loaded.
2. **`df1`** : Removed duplicates, changed `CustomerID` to `object`, corrected `Last_Purchase_Amount` to `float`, handled missing `Last_Name` & `Phone_Number`.
3. **`df2`** : Updates from `df1` + fixed and standardized `Phone_Number`, `Last_Name`, `Last_Purchase_Amount`, handled missing `Total_Purchases` with mean.
4. **`df3`** : Updates from `df1` & `df2` + converted `Last_Purchase_Amount` to `float`, handled missing `Last_Purchase_Amount` & `Feedback_Score` with mean, standardized `Gender`.
5. **`df_cleaned`** : All updates from `df1`, `df2`, `df3` + standardized `Registration_Date` format, saved as Excel.
6. **`customer_data`** : Cleaned dataset for analysis (clusters based on `Age`, `Feedback_Score`, `VIP`, `Total_Revenue`) and visualization.

Other Stored Data

- **`missing_phone_numbers`** : Rows with missing `Phone_Number`.
- **`missing_Last_Name`** : Rows with missing `Last_Name`.
- **`missing_Total_Purchases`** : Rows with missing `Total_Purchases`.
- **`missing_Feedback_Score`** : Rows with missing `Feedback_Score`.
- **`missing_Last_Purchase_Amount`** : Rows with missing `Last_Purchase_Amount`.
- **`high_satisfaction_customer`** : Rows with high `Feedback_Score`.
- **`average_age_high_satisfaction`** : Mean age of customers with high `Feedback_Score`.

Import the necessary packages

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

Initial Data Exploration and Understanding

Before jumping into the data cleaning tasks, it's important to load the dataset, explore its structure, and identify any initial issues.

Question 0.1: Load the Dataset

- Load the CSV file into a pandas DataFrame and inspect the first few rows to get an initial understanding of the data. `CSV Name: Call_List.csv`

In [3]:
```
#Your code here
df = pd.read_csv('data/Call_List.csv')
df.head()
```

Out[3]:

	CustomerID	First_Name	Last_Name	Phone_Number	Age	Email	Total_Purchases	Last_Purchase_Amount
0	1001	Frodo	Baggins	123-545-5421	56	frodo.baggins@example.com	5.0	$100.50
1	1002	Abed	Nadir	123/643/9775	46	abed.nadir@example.com	3.0	75
2	1003	Walter	/White	7066950392	32	walter./white@example.com	7.0	$250.00
3	1004	Dwight	Schrute	123-543-2345	60	dwight.schrute@example.com	NaN	NaN
4	1005	Jon	Snow	876|678|3469	25	jon.snow@example.com	10.0	$500

Question 0.2: Check Dataset Information

- Use the `.info()` method to check the basic information about the DataFrame, including the column names, non-null counts, and data types.

In [4]:
```
#Your code here
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21 entries, 0 to 20
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   CustomerID            21 non-null     int64  
 1   First_Name            21 non-null     object 
 2   Last_Name             20 non-null     object 
 3   Phone_Number          19 non-null     object 
 4   Age                   21 non-null     int64  
 5   Email                 21 non-null     object 
 6   Total_Purchases       17 non-null     float64
 7   Last_Purchase_Amount  17 non-null     object 
 8   Gender                21 non-null     object 
 9   Feedback_Score        17 non-null     float64
 10  Registration_Date     21 non-null     object 
dtypes: float64(2), int64(2), object(7)
memory usage: 1.9+ KB
```

Question 0.3: Describe the Dataset

- Use the `.describe()` method to generate descriptive statistics that summarize the central tendency, dispersion, and shape of the dataset's distribution.

In [5]:
```
#Your code here
df.describe()
```

Out[5]:

	CustomerID	Age	Total_Purchases	Feedback_Score
count	21.000000	21.000000	17.000000	17.000000
mean	1010.952381	41.142857	6.705882	7.176471
std	6.127611	12.865791	4.426690	2.007339
min	1001.000000	19.000000	0.000000	3.000000
25%	1006.000000	32.000000	3.000000	6.000000
50%	1011.000000	41.000000	6.000000	7.000000
75%	1016.000000	53.000000	10.000000	9.000000
max	1020.000000	61.000000	15.000000	10.000000

Question 0.4: List the Column Names

- Check the column names in the DataFrame. Are there any columns names that seem unnecessary, incomplete, or that need fixing?

```
# List column names
df.columns
```

In [6]:
```
#Your code here
df.columns
```

Out[6]:
```
Index(['CustomerID', 'First_Name', 'Last_Name', 'Phone_Number', 'Age', 'Email',
       'Total_Purchases', 'Last_Purchase_Amount', 'Gender', 'Feedback_Score',
       'Registration_Date'],
      dtype='object')
```

Question 0.6: Identify the Number of Missing Values

- Use the `.isna().sum()` method to check how many missing values exist in each column of the DataFrame.

```
# Check for missing values in each column
df.isna().sum()
```

In [7]:
```
#Your code here
df.isna().sum()
```

Out[7]:
```
CustomerID              0
First_Name              0
Last_Name               1
Phone_Number            2
Age                     0
Email                   0
Total_Purchases         4
Last_Purchase_Amount    4
Gender                  0
Feedback_Score          4
Registration_Date       0
dtype: int64
```

Data Cleaning Tasks

Question 1: Remove Duplicate Records

- Remove any duplicate rows from the dataset.

Hint: You can use the `.drop_duplicates` method

In [8]:
```
customer_duplicates = df['CustomerID'].duplicated()
print(customer_duplicates.sum())  # This will give the number of duplicate CustomerID values
```

```
1
```

In [9]:
```
df_1 = df.drop_duplicates(subset=['CustomerID'])
df_1.head()
```

Out[9]:

	CustomerID	First_Name	Last_Name	Phone_Number	Age	Email	Total_Purchases	Last_Purchase_Amount
0	1001	Frodo	Baggins	123-545-5421	56	frodo.baggins@example.com	5.0	$100.50
1	1002	Abed	Nadir	123/643/9775	46	abed.nadir@example.com	3.0	75
2	1003	Walter	/White	7066950392	32	walter./white@example.com	7.0	$250.00
3	1004	Dwight	Schrute	123-543-2345	60	dwight.schrute@example.com	NaN	NaN
4	1005	Jon	Snow	876|678|3469	25	jon.snow@example.com	10.0	$500

In [10]:
```
df_1.isna().sum()
```

Out[10]:
```
CustomerID              0
First_Name              0
Last_Name               1
Phone_Number            2
Age                     0
Email                   0
Total_Purchases         4
Last_Purchase_Amount    4
Gender                  0
Feedback_Score          4
Registration_Date       0
dtype: int64
```

In [11]:
```
df_1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 20 entries, 0 to 19
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   CustomerID            20 non-null     int64
 1   First_Name            20 non-null     object
 2   Last_Name             19 non-null     object
 3   Phone_Number          18 non-null     object
 4   Age                   20 non-null     int64
 5   Email                 20 non-null     object
 6   Total_Purchases       16 non-null     float64
 7   Last_Purchase_Amount  16 non-null     object
 8   Gender                20 non-null     object
 9   Feedback_Score        16 non-null     float64
 10  Registration_Date     20 non-null     object
dtypes: float64(2), int64(2), object(7)
memory usage: 1.9+ KB
```

In [12]:
```
df_1.describe()
```

Out[12]:

	CustomerID	Age	Total_Purchases	Feedback_Score
count	20.00000	20.000000	16.000000	16.000000
mean	1010.50000	40.850000	6.437500	7.125000
std	5.91608	13.128013	4.426718	2.061553
min	1001.00000	19.000000	0.000000	3.000000
25%	1005.75000	31.000000	3.000000	6.000000
50%	1010.50000	40.500000	5.500000	7.000000
75%	1015.25000	53.750000	9.250000	9.000000
max	1020.00000	61.000000	15.000000	10.000000

Change the datatype customer ID to Object

In [13]:
```
# Convert CustomerID column to object (string)
df_1['CustomerID'] = df_1['CustomerID'].astype('object')
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_15720\2730077080.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df_1['CustomerID'] = df_1['CustomerID'].astype('object')
```

Acces to Missing data

In [14]:
```
#missing_phone_numbers
```

In [15]:
```
#missing_Last_Name
```

In [16]:
```
#missing_Total_Purchases
```

In [17]:
```
#missing_Last_Purchase_Amount
```

In [18]:
```
#missing_Feedback_Score
```

Handle Missing data for Name and Phone number

In [19]:
```
# Mengisi NaN dengan "Unknown"
df_1['Last_Name'] = df_1['Last_Name'].fillna('Unknown')
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_15720\3100813272.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df_1['Last_Name'] = df_1['Last_Name'].fillna('Unknown')
```

In [20]:
```
df_1['Phone_Number'] = df_1['Phone_Number'].fillna('Unknown')
```

```
C:\Users\User\AppData\Local\Temp\ipykernel_15720\2273162178.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df_1['Phone_Number'] = df_1['Phone_Number'].fillna('Unknown')
```

Question 2: Fix Phone Numbers And Last Name (Standardize)

- Standardize the `Phone_Number` column by removing special characters (like `-`, `/`, `|`) and ensuring all phone numbers have a uniform format.

Phone Number

In [21]:
```
# Create a copy of df_1
df_2 = df_1.copy()
```

```
# Clean the 'Phone_Number' column by removing specific characters
df_2['Phone_Number'] = df_2['Phone_Number'].replace('[\-/|,]', '', regex=True).astype(object)

# Verify the changes
print(df_2['Phone_Number'].head())
```

```
0    1235455421
1    1236439775
2    7066950392
3    1235432345
4    8766783469
Name: Phone_Number, dtype: object
```

```
<>:5: SyntaxWarning: invalid escape sequence '\-'
<>:5: SyntaxWarning: invalid escape sequence '\-'
C:\Users\User\AppData\Local\Temp\ipykernel_15720\3410953555.py:5: SyntaxWarning: invalid escape sequence '\-'
  df_2['Phone_Number'] = df_2['Phone_Number'].replace('[\-/|,]', '', regex=True).astype(object)
```

Last name

In [22]:
```
import re

# Gabungkan semua teks dalam kolum
all_text = ''.join(df_2['Last_Name'].astype(str))

# Gunakan regex untuk mencari simbol khas
symbols = re.findall(r'[^\w\s]', all_text)

# Tukar kepada set untuk buang pendua
unique_symbols = set(symbols)

# Paparkan simbol khas
print("Simbol yang terdapat dalam kolum 'Last_Purchase_Amount':", unique_symbols)
```

```
Simbol yang terdapat dalam kolum 'Last_Purchase_Amount': {'.', '/'}
```

In [23]:
```
df_2['Last_Name'] = df_2['Last_Name'].replace('[\/,.]', '', regex=True).astype(object)

# Verifikasi perubahan
print(df_2['Last_Name'].head())
```

```
0    Baggins
1      Nadir
2      White
3    Schrute
4       Snow
Name: Last_Name, dtype: object
```

```
<>:1: SyntaxWarning: invalid escape sequence '\/'
<>:1: SyntaxWarning: invalid escape sequence '\/'
C:\Users\User\AppData\Local\Temp\ipykernel_15720\587473401.py:1: SyntaxWarning: invalid escape sequence '\/'
  df_2['Last_Name'] = df_2['Last_Name'].replace('[\/,.]', '', regex=True).astype(object)
```

Last_Purchase_Amount

In [24]:
```
import re

# Gabungkan semua teks dalam kolum
all_text = ''.join(df_2['Last_Purchase_Amount'].astype(str))

# Gunakan regex untuk mencari simbol khas
symbols = re.findall(r'[^\w\s]', all_text)

# Tukar kepada set untuk buang pendua
unique_symbols = set(symbols)

# Paparkan simbol khas
print("Simbol yang terdapat dalam kolum 'Last_Purchase_Amount':", unique_symbols)
```

```
Simbol yang terdapat dalam kolum 'Last_Purchase_Amount': {'.', '$'}
```

In [25]:
```
df_2['Last_Purchase_Amount'] = df_2['Last_Purchase_Amount'].replace('[\$]', '', regex=True).astype(object)

# Verifikasi perubahan
print(df_2['Last_Purchase_Amount'].head())
```

```
0    100.50
1        75
2    250.00
3       NaN
4       500
Name: Last_Purchase_Amount, dtype: object
```

```
<>:1: SyntaxWarning: invalid escape sequence '\$'
<>:1: SyntaxWarning: invalid escape sequence '\$'
C:\Users\User\AppData\Local\Temp\ipykernel_15720\2540588894.py:1: SyntaxWarning: invalid escape sequence '\$'
  df_2['Last_Purchase_Amount'] = df_2['Last_Purchase_Amount'].replace('[\$]', '', regex=True).astype(object)
```

In [26]:
```
df_2.isna().sum()
```

Out[26]:
```
CustomerID              0
First_Name              0
Last_Name               0
Phone_Number            0
Age                     0
Email                   0
Total_Purchases         4
Last_Purchase_Amount    4
Gender                  0
Feedback_Score          4
Registration_Date       0
dtype: int64
```

In [27]:
```
df_2.describe()
```

Out[27]:

	Age	Total_Purchases	Feedback_Score
count	20.000000	16.000000	16.000000
mean	40.850000	6.437500	7.125000
std	13.128013	4.426718	2.061553
min	19.000000	0.000000	3.000000
25%	31.000000	3.000000	6.000000
50%	40.500000	5.500000	7.000000
75%	53.750000	9.250000	9.000000
max	61.000000	15.000000	10.000000

Question 4: Correct Last Purchase Amount

- In the `Last_Purchase_Amount` column, remove dollar signs (`$`) and commas, and convert the values to numeric.

In [29]:
```
#Your Code here
df_1.head()
```

Out[29]:

	CustomerID	First_Name	Last_Name	Phone_Number	Age	Email	Total_Purchases	Last_Purchase_Amount
0	1001	Frodo	Baggins	123-545-5421	56	frodo.baggins@example.com	5.0	$100.5
1	1002	Abed	Nadir	123/643/9775	46	abed.nadir@example.com	3.0	75
2	1003	Walter	/White	7066950392	32	walter./white@example.com	7.0	$250.0
3	1004	Dwight	Schrute	123-543-2345	60	dwight.schrute@example.com	NaN	NaN
4	1005	Jon	Snow	876|678|3469	25	jon.snow@example.com	10.0	$500

In [30]:
```
# Menghapus simbol $ dan koma, lalu mengonversi ke tipe data numerik
df_1['Last_Purchase_Amount'] = df_1['Last_Purchase_Amount'].replace('[\$,]', '', regex=True).astype(float)

# Verifikasi perubahan
df_1['Last_Purchase_Amount'].head()
```

```
<>:2: SyntaxWarning: invalid escape sequence '\$'
<>:2: SyntaxWarning: invalid escape sequence '\$'
C:\Users\User\AppData\Local\Temp\ipykernel_15720\3557507131.py:2: SyntaxWarning: invalid escape sequence '\$'
  df_1['Last_Purchase_Amount'] = df_1['Last_Purchase_Amount'].replace('[\$,]', '', regex=True).astype(float)
C:\Users\User\AppData\Local\Temp\ipykernel_15720\3557507131.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#retu
rning-a-view-versus-a-copy
  df_1['Last_Purchase_Amount'] = df_1['Last_Purchase_Amount'].replace('[\$,]', '', regex=True).astype(float)
```

Out[30]:
```
0    100.5
1     75.0
2    250.0
3      NaN
4    500.0
Name: Last_Purchase_Amount, dtype: float64
```

In [31]:
```
df_2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 20 entries, 0 to 19
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   CustomerID            20 non-null     object 
 1   First_Name            20 non-null     object 
 2   Last_Name             20 non-null     object 
 3   Phone_Number          20 non-null     object 
 4   Age                   20 non-null     int64  
 5   Email                 20 non-null     object 
 6   Total_Purchases       16 non-null     float64
 7   Last_Purchase_Amount  16 non-null     object 
 8   Gender                20 non-null     object 
 9   Feedback_Score        16 non-null     float64
 10  Registration_Date     20 non-null     object 
dtypes: float64(2), int64(1), object(8)
memory usage: 1.9+ KB
```

In [32]:
```
df_2.isna().sum()
```

Out[32]:
```
CustomerID              0
First_Name              0
Last_Name               0
Phone_Number            0
Age                     0
Email                   0
Total_Purchases         4
Last_Purchase_Amount    4
Gender                  0
Feedback_Score          4
Registration_Date       0
dtype: int64
```

Hanlde Missng data Total_Purchases / Last_Purchase_Amount / Feedback score with Mean

Total Purchase

In [33]:
```
df_2['Total_Purchases'].unique()
```

Out[33]:
```
array([ 5.,  3.,  7., nan, 10., 12.,  6.,  2.,  0.,  8., 15.,  9.,  4.,
       13.,  1.])
```

In [34]:
```
# Gantikan nilai NaN dalam kolum 'Total_Purchases' dengan min (mean)
df_2['Total_Purchases'] = df_2['Total_Purchases'].fillna(df_2['Total_Purchases'].mean())
```

Last Purchase amount

In [35]:
```
df_3 = df_2.copy()
# Convert CustomerID column to object (string)
df_3['Last_Purchase_Amount'] = df_3['Last_Purchase_Amount'].astype('float')
# Verify the change
```

In [36]:
```
df_3['Last_Purchase_Amount'] = df_3['Last_Purchase_Amount'].fillna(df_3['Last_Purchase_Amount'].mean())
```

In [37]:
```
df_3.isna().sum()
```

Out[37]:
```
CustomerID              0
First_Name              0
Last_Name               0
Phone_Number            0
Age                     0
Email                   0
Total_Purchases         0
Last_Purchase_Amount    0
Gender                  0
Feedback_Score          4
Registration_Date       0
dtype: int64
```

In [38]:
```
df_3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 20 entries, 0 to 19
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   CustomerID            20 non-null     object
 1   First_Name            20 non-null     object
 2   Last_Name             20 non-null     object
 3   Phone_Number          20 non-null     object
 4   Age                   20 non-null     int64
 5   Email                 20 non-null     object
 6   Total_Purchases       20 non-null     float64
 7   Last_Purchase_Amount  20 non-null     float64
 8   Gender                20 non-null     object
 9   Feedback_Score        16 non-null     float64
 10  Registration_Date     20 non-null     object
dtypes: float64(3), int64(1), object(7)
memory usage: 1.9+ KB
```

Feedback score

In [39]:
```
df_3['Feedback_Score'] = df_3['Feedback_Score'].fillna(df_3['Feedback_Score'].mean())
```

Question 5: Standardize Gender

- Clean the `Gender` column to ensure consistent values:
  - Convert `'M'` or `'Male'` to `'Male'`
  - Convert `'F'` or `'Female'` to `'Female'`
  - Combine any other values (e.g., `'Other'`, `'Unknown'`) into `'Other'`.
  - *(Hint: Use the `.replace()` method to standardize the values)*

In [40]:
```
df_3['Gender'].value_counts()
```

Out[40]:
```
Gender
F          5
M          4
Male       4
Female     3
Other      2
Unknown    2
Name: count, dtype: int64
```

In [41]:
```
# Gantikan 'F' dengan 'Female'
df_3['Gender'] = df_3['Gender'].replace('F', 'Female')
df_3['Gender'] = df_3['Gender'].replace('M', 'Male')
df_3['Gender'] = df_3['Gender'].replace('Other', 'Unknown')
```

In [42]:
```
df_3.head()
```

Out[42]:

	CustomerID	First_Name	Last_Name	Phone_Number	Age	Email	Total_Purchases	Last_Purchase_Amount
0	1001	Frodo	Baggins	1235455421	56	frodo.baggins@example.com	5.0000	100.50000
1	1002	Abed	Nadir	1236439775	46	abed.nadir@example.com	3.0000	75.00000
2	1003	Walter	White	7066950392	32	walter./white@example.com	7.0000	250.00000
3	1004	Dwight	Schrute	1235432345	60	dwight.schrute@example.com	6.4375	169.53125
4	1005	Jon	Snow	8766783469	25	jon.snow@example.com	10.0000	500.00000

In [43]:
```
df_3.describe()
```

Out[43]:

	Age	Total_Purchases	Last_Purchase_Amount	Feedback_Score
count	20.000000	20.000000	20.000000	20.000000
mean	40.850000	6.437500	169.531250	7.125000
std	13.128013	3.933242	123.213093	1.831738
min	19.000000	0.000000	0.000000	3.000000
25%	31.000000	3.750000	87.687500	6.000000
50%	40.500000	6.437500	159.890625	7.125000
75%	53.750000	8.250000	212.500000	8.250000
max	61.000000	15.000000	500.000000	10.000000

Question 7: Convert Registration Date to Uniform Format

- Convert the `Registration_Date` column to a uniform date format (`YYYY-MM-DD`):
  - *(Hint: Use the `pd.to_datetime()` function to handle different date formats)*
  - Handle any parsing errors gracefully using the `errors` parameter.

In [45]:
```
df_cleaned = df_3.copy()
df_cleaned ['Registration_Date'] = pd.to_datetime(df_cleaned ['Registration_Date'], errors='coerce')

# Save the cleaned DataFrame to an Excel file
df_cleaned.to_excel('cleaned_data.xlsx', index=False)
```

Manual Cluster

Question 8: Create Age Groups

- Create a new column called `Age_Group` that classifies customers into the following groups:
  - `Under 18`
  - `18-30`
  - `31-50`
  - `51+`
  - *(Hint: Use the `pd.cut(column, bins=[0,18,30,50,np.inf], labels =['Under 18', '18-30','31-50','51+'])` function to bin the ages into groups)*

In [46]:
```
# Your code here
customer_data = df_cleaned.copy()


# Tambahkan kolum Age_Group berdasarkan julat umur
customer_data['Age_Group'] = pd.cut(
    customer_data['Age'],
    bins=[0, 18, 30, 50, np.inf],
    labels=['Under 18', '18-30', '31-50', '51+']
)
customer_data.head(1)
```

Out[46]:

	CustomerID	First_Name	Last_Name	Phone_Number	Age	Email	Total_Purchases	Last_Purchase_Amount
0	1001	Frodo	Baggins	1235455421	56	frodo.baggins@example.com	5.0	100.5

In [ ]:
```

```

In [ ]:
```

```

Question 9: Filter VIP Customers

- Filter the dataset to display only the customers who are marked as `VIP` and have made more than 5 `Total_Purchases` .
  - *(Hint: Use boolean indexing OR `.loc` with conditions to filter the rows)*

In [47]:
```
# Tambahkan kolum Satisfaction_Level menggunakan pd.cut()
customer_data['VIP'] = pd.cut(
    customer_data['Total_Purchases'],
    bins=[-np.inf,5, np.inf],
    labels=['', 'VIP'],
    right=False
)
```

Question 10: Create Customer Satisfaction Column(Optional)

- Based on the `Feedback_Score` column, create a new column called `Satisfaction_Level` :
  - `'High'` if the score is 8 or above
  - `'Medium'` if the score is between 5 and 7
  - `'Low'` if the score is below 5
  - *(Hint: Use `pd.cut()` or `np.where()` to create the new column)*

```
# Create Satisfaction_Level column based on Feedback_Score
df['Satisfaction_Level'] = pd.cut(df['Feedback_Score'], bins=[-np.inf, 5, 8, np.inf], labels=['Low',
'Medium', 'High'])
```

In [48]:
```
# Tambahkan kolum Satisfaction_Level menggunakan pd.cut()
customer_data['Satisfaction_Level'] = pd.cut(
```

```
    customer_data['Feedback_Score'],
    bins=[-np.inf, 5, 8, np.inf],
    labels=['Low', 'Medium', 'High'],
    right=False  # Supaya nilai tepat 5 masuk ke kategori 'Medium'
)
```

In [ ]:

Question 11: Calculate Total Revenue per Customer

- Create a new column called `Total_Revenue` by multiplying `Total_Purchases` with `Last_Purchase_Amount` for each customer.
  - *(Hint: Make sure to convert `Last_Purchase_Amount` to numeric before performing the calculation)*
  - *(Hint: You can use `pd.to_numeric()`)*

In [49]:
```
# Mengira Total Revenue dengan darabkan Total_Purchases dan Last_Purchase_Amount
customer_data['Total_Revenue'] = customer_data['Total_Purchases'] * customer_data['Last_Purchase_Amount']
```

In [ ]:

Question 12: Find Average Age of High-Satisfaction Customers

- Calculate the average age of customers who have a `Satisfaction_Level` of `'High'`.
  - *(Hint: Use boolean indexing to filter rows based on the condition and then calculate the average)*

In [50]:
```
# Filter customers with 'High' Satisfaction_Level
high_satisfaction_customers = customer_data[customer_data['Satisfaction_Level'] == 'High']
high_satisfaction_customers
```

Out[50]:

	CustomerID	First_Name	Last_Name	Phone_Number	Age	Email	Total_Purchases	Last_Purchase_Am
0	1001	Frodo	Baggins	1235455421	56	frodo.baggins@example.com	5.0000	100.5
1	1002	Abed	Nadir	1236439775	46	abed.nadir@example.com	3.0000	75.0
5	1006	Ron	Swanson	3047622467	38	ron.swanson@example.com	12.0000	300.0
8	1009	Gandalf	Unknown	Na	40	gandalf.@example.com	6.4375	50.0
10	1011	Samwise	Gamgee	Unknown	28	samwise.gamgee@example.com	8.0000	150.2
13	1014	Leslie	Knope	8766783469	57	leslie.knope@example.com	4.0000	90.0
18	1019	Creed	Braton	Na	41	creed.braton@example.com	6.4375	169.5

In [51]:
```
average_age_high_satisfaction = high_satisfaction_customers['Age'].mean()
average_age_high_satisfaction
```

Out[51]:
```
np.float64(43.714285714285715)
```

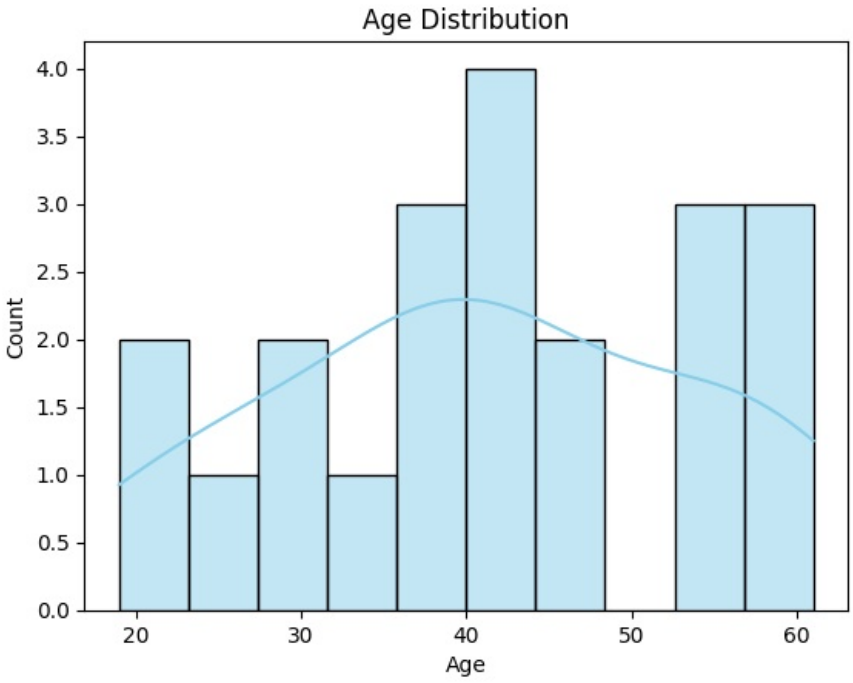
Question 13: Plot Age Distribution

- Create a histogram or KDE plot to visualize the distribution of customer ages.
  - *(Hint: Use `sns.histplot()` or `sns.kdeplot()` from Seaborn to create the plot)*

```
sns.histplot(column, kde=True)
```

In [58]:
```
sns.histplot(df['Age'], kde=True, bins=10, color='skyblue')
plt.title('Age Distribution')
plt.xlabel('Age')
```

Out[58]:
```
Text(0.5, 0, 'Age')
```



Question 15: Scatter Plot of Age vs. Total Purchases

- Create a scatter plot to visualize the relationship between `Age` and `Total_Purchases`.
  - *(Hint: Use `sns.scatterplot()` from Seaborn to create the scatter plot)*

In [61]:
```
customer_data.head()
```

Out[61]:

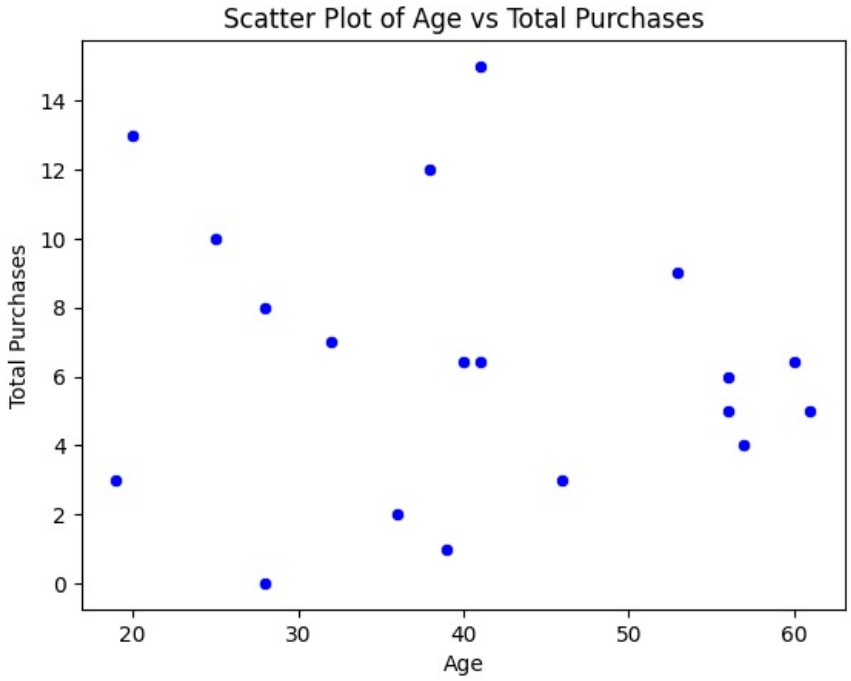
	CustomerID	First_Name	Last_Name	Phone_Number	Age	Email	Total_Purchases	Last_Purchase_Amount
0	1001	Frodo	Baggins	1235455421	56	frodo.baggins@example.com	5.0000	100.50000
1	1002	Abed	Nadir	1236439775	46	abed.nadir@example.com	3.0000	75.00000
2	1003	Walter	White	7066950392	32	walter./white@example.com	7.0000	250.00000
3	1004	Dwight	Schrute	1235432345	60	dwight.schrute@example.com	6.4375	169.53125
4	1005	Jon	Snow	8766783469	25	jon.snow@example.com	10.0000	500.00000

In [69]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming customer_data is your DataFrame
sns.scatterplot(x="Age", y="Total_Purchases", data=customer_data, color='blue')

# Adding title and labels
plt.title("Scatter Plot of Age vs Total Purchases")
plt.xlabel("Age")
plt.ylabel("Total Purchases")

# Show the plot
plt.show()
```

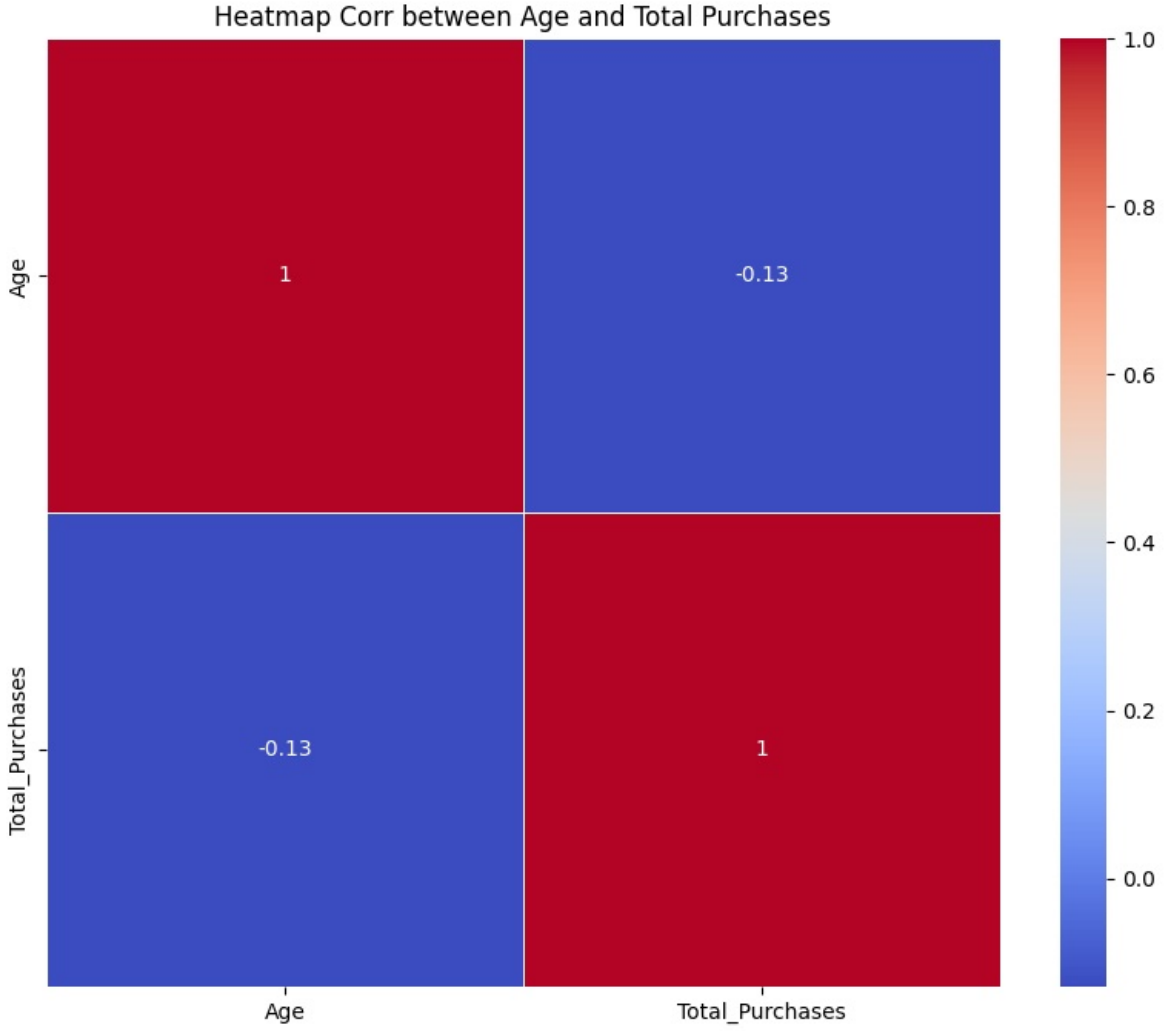


In [73]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the correlation matrix
correlation_matrix = customer_data[['Age', 'Total_Purchases']].corr()

# Plot heatmap for correlation
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)

# Adding a title
plt.title("Heatmap Corr between Age and Total Purchases")
plt.show()
```



Question 17: Trend of Average Feedback Score Over Time

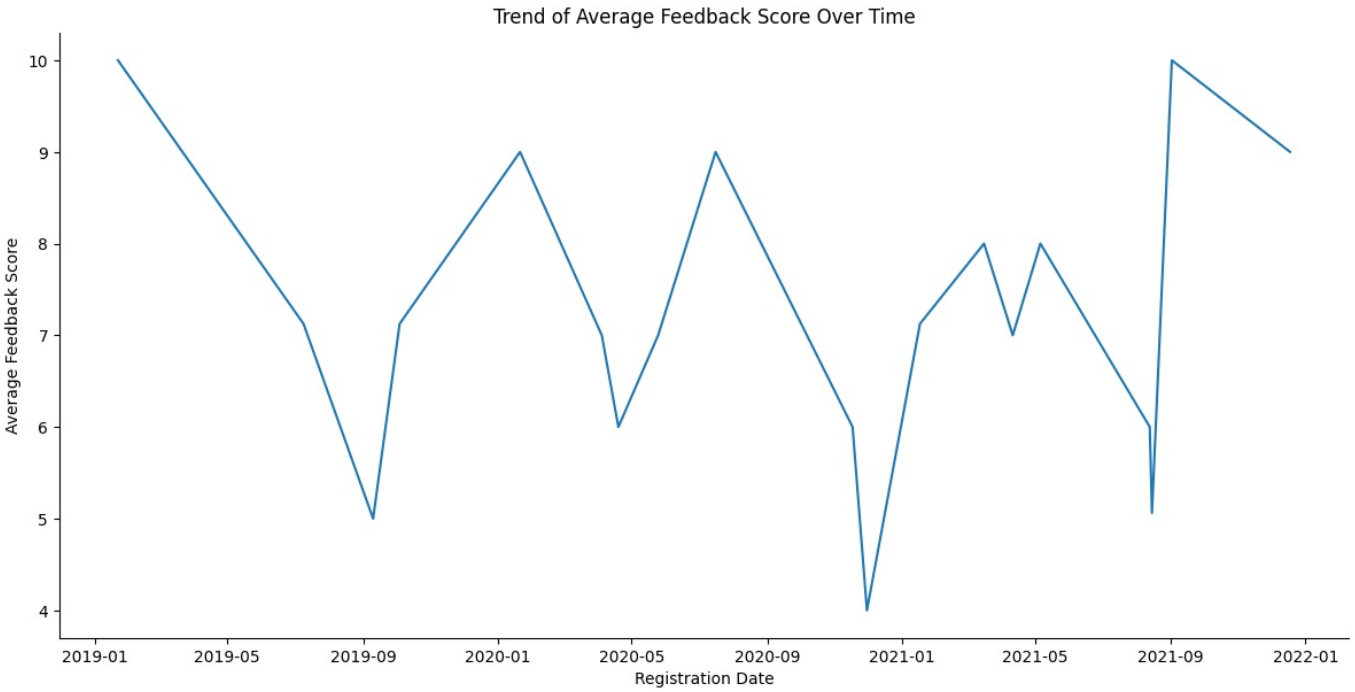
- Using the `Registration_Date`, create a line plot to show how the average `Feedback_Score` has changed over time.
  - *(Hint: Use `sns.relplot()` with `kind="line"` from Seaborn and group by `Registration_Date`, by taking mean of `Feedback_SCORE`)*

In [75]:
```
# Group by Registration_Date and calculate the average Feedback_Score
average_feedback = customer_data.groupby('Registration_Date')['Feedback_Score'].mean().reset_index()


# Create a line plot using sns.relplot
sns.relplot(x='Registration_Date', y='Feedback_Score', data=average_feedback, kind="line", aspect=2, height=6)

# Add title and labels
plt.title("Trend of Average Feedback Score Over Time")
plt.xlabel("Registration Date")
plt.ylabel("Average Feedback Score")

# Show the plot
plt.show()
```



In [76]:
```
average_feedback
```

Out[76]:

	Registration_Date	Feedback_Score
0	2019-01-22	10.0000
1	2019-07-09	7.1250
2	2019-09-10	5.0000
3	2019-10-04	7.1250
4	2020-01-21	9.0000
5	2020-04-04	7.0000
6	2020-04-19	6.0000
7	2020-05-25	7.0000
8	2020-07-16	9.0000
9	2020-11-17	6.0000
10	2020-11-30	4.0000
11	2021-01-17	7.1250
12	2021-03-16	8.0000
13	2021-04-11	7.0000
14	2021-05-06	8.0000
15	2021-08-13	6.0000
16	2021-08-15	5.0625
17	2021-09-02	10.0000
18	2021-12-18	9.0000

Project Summary

1. **Initial Data Exploration**:

- Load the dataset and explore its structure.
- Use functions like `.info()`, `.describe()`, and `.head()` to get insights into the data.
- Identify potential issues with missing values, inconsistent formats, and incorrect data types.

2. **Data Cleaning Tasks**:

- Remove duplicate records to ensure data accuracy.
- Fix phone numbers by standardizing their format.
- Handle missing data by filling in appropriate values or dropping rows.
- Standardize columns like `Gender` and `Customer_Type` to ensure consistency.
- Convert the `Registration_Date` column into a uniform date format.
- Create age groups by binning the `Age` column into predefined categories.

3. **Data Analysis**:

- Calculate total revenue per customer by multiplying `Total_Purchases` with `Last_Purchase_Amount`.
- Filter VIP customers who have made more than 5 purchases.
- Create a new `Satisfaction_Level` column based on the `Feedback_Score` (Low, Medium, High).
- Find the average age of customers with a high satisfaction level.

4. **Data Visualization**:

- Use Seaborn to create various plots:
  - **Histogram/KDE** to visualize the distribution of customer ages.
  - **Bar plot** to compare total purchases and revenue across customer types.
  - **Scatter plot** to analyze the relationship between age and total purchases.
  - **Line plot** (using `relplot`) to show the trend of average feedback scores over time.

This project covers a comprehensive range of **data cleaning**, **analysis**, and **visualization** tasks, giving you hands-on experience with real-world data handling and reporting.