Data Analysis with Pandas

Kaggle Dataset

- https://www.kaggle.com/code/prashant111/comprehensive-data-analysis-with-pandas/notebook

1. Introduction

- Pandas is an open source library for Data Analysis in Python
- In this project, I explore pandas and important data analysis tools of pandas
- Pandas offers data structures and operations for manipulating numerical tables and time series

2. Key Features of Pandas

- It provides tools for reading and writing data from different sources such as CSV, Excel, databases such as JSON, SQL
- It provides different data structures like series, data frames for data manipulation and indexing
- It includes subsetting, slicing, filtering, merging, joining, groupby and reshaping operations
- It can deal with missing data by deleting them or filling them zeros
- It integrates with other python libraries such as Scikit-learn, SciPy and Statsmodels

3. Pandas Advantages

- Data Representation
- Data Subsetting and Filtering

In [4]:
```
# Importing Pandas & Numpy

import pandas as pd
import numpy as np
```

In [11]:
```
df = pd.read_csv(r'C:\Users\Windows10 Pro\Downloads\DataScience_AI\2025\May2025\08M
print(df)
```

```
        User_ID Product_ID Gender    Age  Occupation City_Category  \
0       1000001  P00069042      F   0-17          10             A
1       1000001  P00248942      F   0-17          10             A
2       1000001  P00087842      F   0-17          10             A
3       1000001  P00085442      F   0-17          10             A
4       1000002  P00285442      M    55+          16             C
...         ...        ...    ...    ...         ...           ...
550063  1006033  P00372445      M  51-55          13             B
550064  1006035  P00375436      F  26-35           1             C
550065  1006036  P00375436      F  26-35          15             B
550066  1006038  P00375436      F    55+           1             C
550067  1006039  P00371644      F  46-50           0             B

       Stay_In_Current_City_Years  Marital_Status  Product_Category_1  \
0                               2               0                   3
1                               2               0                   1
2                               2               0                  12
3                               2               0                  12
4                              4+               0                   8
...                           ...             ...                 ...
550063                          1               1                  20
550064                          3               0                  20
550065                         4+               1                  20
550066                          2               0                  20
550067                         4+               1                  20

        Product_Category_2  Product_Category_3  Purchase
0                      NaN                 NaN      8370
1                      6.0                14.0     15200
2                      NaN                 NaN      1422
3                     14.0                 NaN      1057
4                      NaN                 NaN      7969
...                    ...                 ...       ...
550063                 NaN                 NaN       368
550064                 NaN                 NaN       371
550065                 NaN                 NaN       137
550066                 NaN                 NaN       365
550067                 NaN                 NaN       490

[550068 rows x 12 columns]
```

In [12]:
```
## Exploratory Data Analysis (EDA)

type(df)
```

Out[12]:
```
pandas.core.frame.DataFrame
```

In [13]:
```
df.shape
```

Out[13]:
```
(550068, 12)
```

In [14]:
```
df.head()
```

Out[14]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Yea
0	1000001	P00069042	F	0-17	10	A	
1	1000001	P00248942	F	0-17	10	A	
2	1000001	P00087842	F	0-17	10	A	
3	1000001	P00085442	F	0-17	10	A	
4	1000002	P00285442	M	55+	16	C	4

In [15]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 550068 entries, 0 to 550067
Data columns (total 12 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   User_ID                     550068 non-null  int64  
 1   Product_ID                  550068 non-null  object 
 2   Gender                      550068 non-null  object 
 3   Age                         550068 non-null  object 
 4   Occupation                  550068 non-null  int64  
 5   City_Category               550068 non-null  object 
 6   Stay_In_Current_City_Years  550068 non-null  object 
 7   Marital_Status              550068 non-null  int64  
 8   Product_Category_1          550068 non-null  int64  
 9   Product_Category_2          376430 non-null  float64
 10  Product_Category_3          166821 non-null  float64
 11  Purchase                    550068 non-null  int64  
dtypes: float64(2), int64(5), object(5)
memory usage: 50.4+ MB
```

In [16]:
```
## Handling Missing Data

df.isnull().sum()
```

Out[16]:
```
User_ID                           0
Product_ID                        0
Gender                            0
Age                               0
Occupation                        0
City_Category                     0
Stay_In_Current_City_Years        0
Marital_Status                    0
Product_Category_1                0
Product_Category_2           173638
Product_Category_3           383247
Purchase                          0
dtype: int64
```

In [20]:
```
df = df.ffill()  # Both pad and ffill mean forward fill, i.e., filling missing valu
                 # The alias ffill() is now the preferred method.
```

In [21]:
```
df.isnull().sum()
```

Out[21]:
```
User_ID                     0
Product_ID                  0
Gender                      0
Age                         0
Occupation                  0
City_Category               0
Stay_In_Current_City_Years  0
Marital_Status              0
Product_Category_1          0
Product_Category_2          1
Product_Category_3          1
Purchase                    0
dtype: int64
```

In [24]:
```
# We can see that the Product_Category_2 and Product_Category_3 have 1 missing valu

df[['Product_Category_2', 'Product_Category_3']].head()
```

Out[24]:

	Product_Category_2	Product_Category_3
0	NaN	NaN
1	6.0	14.0
2	6.0	14.0
3	14.0	14.0
4	14.0	14.0

In [26]:
```
# We can see that the first element of each column are NaN.
# So, in this case pad or fill option does not work. Here, we should use bfill or b


df = df.ffill()
```

In [27]:
```
df.isnull().sum()
```

Out[27]:
```
User_ID                     0
Product_ID                  0
Gender                      0
Age                         0
Occupation                  0
City_Category               0
Stay_In_Current_City_Years  0
Marital_Status              0
Product_Category_1          0
Product_Category_2          0
Product_Category_3          0
Purchase                    0
dtype: int64
```

Assert statement will return nothing if the value being tested is true and will throw an AssertionError if the value is false # assert 1 == 1 (return Nothing if the value is True) # assert 1 == 2 (return Assertion Error if the value is False)

In [29]:
```
#assert that there are no missing values in the dataframe
assert pd.notnull(df).all().all()
```

In [30]:
```
# Indexing and Slicing in Pandas

# make a copy of dataframe
df1 = df.copy()
```

In [31]:
```
# select first row of dataframe

df1.loc[0]
```

Out[31]:
```
User_ID                     1000001
Product_ID                P00069042
Gender                            F
Age                            0-17
Occupation                       10
City_Category                     A
Stay_In_Current_City_Years        2
Marital_Status                    0
Product_Category_1                3
Product_Category_2              6.0
Product_Category_3             14.0
Purchase                       8370
Name: 0, dtype: object
```

In [32]:
```
#select first five rows for a specific column
df1.loc[:,'Purchase'].head()
```

Out[32]:
```
0     8370
1    15200
2     1422
3     1057
4     7969
Name: Purchase, dtype: int64
```

In [33]:
```
df1.loc[:,['Age','Occupation']]
```

Out[33]:

	Age	Occupation
0	0-17	10
1	0-17	10
2	0-17	10
3	0-17	10
4	55+	16
...	...	...
550063	51-55	13
550064	26-35	1
550065	26-35	15
550066	55+	1
550067	46-50	0

550068 rows × 2 columns

In [34]:
```
Select first five rows for multiple columns, say list[]

df1.loc[[0, 1, 2, 3, 4],['Age','Occupation']]
```

Out[34]:

	Age	Occupation
0	0-17	10
1	0-17	10
2	0-17	10
3	0-17	10
4	55+	16

In [36]:
```
df1.head()
```

Out[36]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Yea
0	1000001	P00069042	F	0-17	10	A	
1	1000001	P00248942	F	0-17	10	A	
2	1000001	P00087842	F	0-17	10	A	
3	1000001	P00085442	F	0-17	10	A	
4	1000002	P00285442	M	55+	16	C	4

In [38]:
```
# row selection using .iloc indexer

# select first row of dataframe¶
df1.iloc[0]
```

Out[38]:
```
User_ID                     1000001
Product_ID                P00069042
Gender                            F
Age                            0-17
Occupation                       10
City_Category                     A
Stay_In_Current_City_Years        2
Marital_Status                    0
Product_Category_1                3
Product_Category_2              6.0
Product_Category_3             14.0
Purchase                       8370
Name: 0, dtype: object
```

In [39]:
```
# select last row of dataframe
df1.iloc[-1]
```

Out[39]:
```
User_ID                     1006039
Product_ID                P00371644
Gender                            F
Age                           46-50
Occupation                        0
City_Category                     B
Stay_In_Current_City_Years       4+
Marital_Status                    1
Product_Category_1               20
Product_Category_2              2.0
Product_Category_3             11.0
Purchase                        490
Name: 550067, dtype: object
```

In [40]:
```
#select first row of dataframe
```

```
df1.iloc[0]
```

Out[40]:
```
User_ID                     1000001
Product_ID                P00069042
Gender                            F
Age                            0-17
Occupation                       10
City_Category                     A
Stay_In_Current_City_Years        2
Marital_Status                    0
Product_Category_1                3
Product_Category_2              6.0
Product_Category_3             14.0
Purchase                       8370
Name: 0, dtype: object
```

In [41]:
```
#select last row of dataframe

df1.iloc[-1]
```

Out[41]:
```
User_ID                     1006039
Product_ID                P00371644
Gender                            F
Age                           46-50
Occupation                        0
City_Category                     B
Stay_In_Current_City_Years       4+
Marital_Status                    1
Product_Category_1               20
Product_Category_2              2.0
Product_Category_3             11.0
Purchase                        490
Name: 550067, dtype: object
```

In [42]:
```
# get index of first occurence of maximum Purchase value

df1['Purchase'].idxmax()
```

Out[42]:
```
87440
```

In [44]:
```
# get the row with the maximum Purchase value

df1.loc[df1['Purchase'].idxmax()]
```

Out[44]:
```
User_ID                     1001474
Product_ID                P00052842
Gender                            M
Age                           26-35
Occupation                        4
City_Category                     A
Stay_In_Current_City_Years        2
Marital_Status                    1
Product_Category_1               10
Product_Category_2             15.0
Product_Category_3              8.0
Purchase                      23961
Name: 87440, dtype: object
```

In [45]:
```
# get value at 1st row and Purchase column pair
df1.at[1, 'Purchase']
```

Out[45]: 15200

In [46]:
```
# get value at 1st row and 11th column pair
df1.iat[1, 11]
```

Out[46]: 15200

In [47]:
```
# make a copy of dataframe df

df2 = df.copy()
```

In [48]:
```
df2.head()
```

Out[48]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Yea
0	1000001	P00069042	F	0-17	10	A	
1	1000001	P00248942	F	0-17	10	A	
2	1000001	P00087842	F	0-17	10	A	
3	1000001	P00085442	F	0-17	10	A	
4	1000002	P00285442	M	55+	16	C	4

In [49]:
```
# get the purchase amount with a given user_id and product_id

df2.loc[((df2['User_ID'] == 1000001) & (df2['Product_ID'] == 'P00069042')), 'Purcha
```

Out[49]:
```
0    8370
Name: Purchase, dtype: int64
```

In [51]:
```
# Indexing with isin() method: The isin() method of Series, returns a boolean vecto

values=[1000001,'P00069042','F',0-17,10,'A',2,0,3,6,14,8370]

df2_indexed=df2.isin(values)

df2_indexed.head(10)
```

Out[51]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Yea
0	True	True	True	False	True	True	Fals
1	True	False	True	False	True	True	Fals
2	True	False	True	False	True	True	Fals
3	True	False	True	False	True	True	Fals
4	False	False	False	False	False	False	Fals
5	False	False	False	False	False	True	Fals
6	False	False	False	False	False	False	Fals
7	False	False	False	False	False	False	Fals
8	False	False	False	False	False	False	Fals
9	False	False	False	False	False	True	Fals

In [53]:
```
#  any() and all() methods to quickly select subsets of the data that meet a given

row_mask = df2.isin(values).any(axis=1)


df[row_mask]
```

Out[53]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_Cit
0	1000001	P00069042	F	0-17	10	A	
1	1000001	P00248942	F	0-17	10	A	
2	1000001	P00087842	F	0-17	10	A	
3	1000001	P00085442	F	0-17	10	A	
4	1000002	P00285442	M	55+	16	C	
...	...	...	...	...	...	...	
550063	1006033	P00372445	M	51-55	13	B	
550064	1006035	P00375436	F	26-35	1	C	
550065	1006036	P00375436	F	26-35	15	B	
550066	1006038	P00375436	F	55+	1	C	
550067	1006039	P00371644	F	46-50	0	B	

509212 rows × 12 columns

In [54]:
```
# The where() method and masking: values from a Series with a boolean vector and it
# To guarantee that the output has the same shape as the original data, we can use

df2_where=df2.where(df2 == 0)
(df2_where).head(10)
```

Out[54]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Year
0	NaN	NaN	NaN	NaN	NaN	NaN	Nal
1	NaN	NaN	NaN	NaN	NaN	NaN	Nal
2	NaN	NaN	NaN	NaN	NaN	NaN	Nal
3	NaN	NaN	NaN	NaN	NaN	NaN	Nal
4	NaN	NaN	NaN	NaN	NaN	NaN	Nal
5	NaN	NaN	NaN	NaN	NaN	NaN	Nal
6	NaN	NaN	NaN	NaN	NaN	NaN	Nal
7	NaN	NaN	NaN	NaN	NaN	NaN	Nal
8	NaN	NaN	NaN	NaN	NaN	NaN	Nal
9	NaN	NaN	NaN	NaN	NaN	NaN	Nal

In [55]:

```
# Indexing with query() method: There is a query() method in the DataFrame objects
# This method queries the columns of a DataFrame with a boolean expression

df2.query('(Product_Category_1 > Product_Category_2) & (Product_Category_2 > Produc
```

Out[55]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_Cit
165	1000033	P00111742	M	46-50	3	A	
304	1000053	P00117542	M	26-35	0	B	
351	1000058	P00288642	M	26-35	2	B	
387	1000062	P00087242	F	36-45	3	A	
724	1000137	P00124642	F	46-50	6	C	
...	...	...	...	...	...	...	
545338	1005954	P00327342	M	46-50	11	A	
545339	1005954	P00087842	M	46-50	11	A	
545461	1005972	P00255842	F	26-35	20	B	
545747	1006016	P00058642	M	46-50	1	B	
545896	1006037	P00183142	F	46-50	1	C	

2278 rows × 12 columns

In [56]:
```
# Indexing and reindexing in pandas: Reindexing changes the row labels and column l
# To reindex means to conform the data to match a given set of labels along a parti

# let's create a new dataframe

food = pd.DataFrame({'Place':['Home', 'Home', 'Hotel', 'Hotel'],
                     'Time': ['Lunch', 'Dinner', 'Lunch', 'Dinner'],
                     'Food':['Soup', 'Rice', 'Soup', 'Chapati'],
                     'Price($)':[10, 20, 30, 40]})

food
```

Out[56]:

	Place	Time	Food	Price($)
0	Home	Lunch	Soup	10
1	Home	Dinner	Rice	20
2	Hotel	Lunch	Soup	30
3	Hotel	Dinner	Chapati	40

In [57]:
```
# Set an index: DataFrame has a set_index() method which takes a column name (for a

food_indexed1=food.set_index('Place')

food_indexed1
```

Out[57]:

	Time	Food	Price($)
Place			
Home	Lunch	Soup	10
Home	Dinner	Rice	20
Hotel	Lunch	Soup	30
Hotel	Dinner	Chapati	40

In [58]:
```
food_indexed2=food.set_index(['Place', 'Time'])

food_indexed2
```

Out[58]:

		Food	Price($)
Place	**Time**		
Home	**Lunch**	Soup	10
	Dinner	Rice	20
Hotel	**Lunch**	Soup	30
	Dinner	Chapati	40

In [59]:
```
# Reset the index: There is a function called reset_index() which transfers the ind
#This is the inverse operation of set_index()

food_indexed2.reset_index()
```

Out[59]:

	Place	Time	Food	Price($)
0	Home	Lunch	Soup	10
1	Home	Dinner	Rice	20
2	Hotel	Lunch	Soup	30
3	Hotel	Dinner	Chapati	40

In [60]:
```
# MultiIndex or advanced indexing

sales=pd.DataFrame([['books','online', 200, 50],['books','retail', 250, 75],
                    ['toys','online', 100, 20],['toys','retail', 140, 30],
                    ['watches','online', 500, 100],['watches','retail', 600, 150],
                    ['computers','online', 1000, 200],['computers','retail', 1200,
                    ['laptops','online', 1100, 400],['laptops','retail', 1400, 500]
                    ['smartphones','online', 600, 200],['smartphones','retail', 800
                    columns=['Items', 'Mode', 'Price', 'Profit'])


sales
```

Out[60]:

	Items	Mode	Price	Profit
0	books	online	200	50
1	books	retail	250	75
2	toys	online	100	20
3	toys	retail	140	30
4	watches	online	500	100
5	watches	retail	600	150
6	computers	online	1000	200
7	computers	retail	1200	300
8	laptops	online	1100	400
9	laptops	retail	1400	500
10	smartphones	online	600	200
11	smartphones	retail	800	250

In [61]:
```
sales1=sales.set_index(['Items', 'Mode'])

sales1
```

Out[61]:

Items	Mode	Price	Profit
books	**online**	200	50
	retail	250	75
toys	**online**	100	20
	retail	140	30
watches	**online**	500	100
	retail	600	150
computers	**online**	1000	200
	retail	1200	300
laptops	**online**	1100	400
	retail	1400	500
smartphones	**online**	600	200
	retail	800	250

In [62]:
```
# View index

sales1.index
```

Out[62]:
```
MultiIndex([(      'books', 'online'),
            (      'books', 'retail'),
            (       'toys', 'online'),
            (       'toys', 'retail'),
            (    'watches', 'online'),
            (    'watches', 'retail'),
            (  'computers', 'online'),
            (  'computers', 'retail'),
            (    'laptops', 'online'),
            (    'laptops', 'retail'),
            ('smartphones', 'online'),
            ('smartphones', 'retail')],
           names=['Items', 'Mode'])
```

In [63]:
```
# Swap the column  in multiple index

sales2=sales1.swaplevel('Mode', 'Items')

sales2
```

Out[63]:

Mode	Items	Price	Profit
online	**books**	200	50
retail	**books**	250	75
online	**toys**	100	20
retail	**toys**	140	30
online	**watches**	500	100
retail	**watches**	600	150
online	**computers**	1000	200
retail	**computers**	1200	300
online	**laptops**	1100	400
retail	**laptops**	1400	500
online	**smartphones**	600	200
retail	**smartphones**	800	250

In [64]:
```
# Sorting in pandas
# sort the dataframe df2 by label

df2.sort_index()
```

Out[64]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_Cit
0	1000001	P00069042	F	0-17	10	A	
1	1000001	P00248942	F	0-17	10	A	
2	1000001	P00087842	F	0-17	10	A	
3	1000001	P00085442	F	0-17	10	A	
4	1000002	P00285442	M	55+	16	C	
...	...	...	...	...	...	...	
550063	1006033	P00372445	M	51-55	13	B	
550064	1006035	P00375436	F	26-35	1	C	
550065	1006036	P00375436	F	26-35	15	B	
550066	1006038	P00375436	F	55+	1	C	
550067	1006039	P00371644	F	46-50	0	B	

550068 rows × 12 columns

In [65]:
```
df2.sort_values(by=['Product_Category_1'])
```

Out[65]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_Cit
271814	1005880	P00016042	M	26-35	1	A	
208659	1002109	P00298942	M	26-35	16	B	
436707	1001231	P00334242	M	26-35	12	C	
108508	1004685	P00025442	M	36-45	1	B	
208658	1002109	P00062842	M	26-35	16	B	
...	...	...	...	...	...	...	
547638	1002549	P00375436	M	55+	13	C	
547640	1002553	P00375436	M	26-35	7	C	
547642	1002556	P00371644	M	26-35	4	C	
547644	1002558	P00375436	M	55+	17	C	
550067	1006039	P00371644	F	46-50	0	B	

550068 rows × 12 columns

In [66]:
```
# Categorical data in pandas

df3 = df.copy()

df3.dtypes
```

Out[66]:
```
User_ID                       int64
Product_ID                   object
Gender                       object
Age                          object
Occupation                    int64
City_Category                object
Stay_In_Current_City_Years   object
Marital_Status                int64
Product_Category_1            int64
Product_Category_2          float64
Product_Category_3          float64
Purchase                      int64
dtype: object
```

In [67]:
```
df3['Gender'].describe()
```

Out[67]:
```
count     550068
unique         2
top            M
freq      414259
Name: Gender, dtype: object
```

In [68]:
```
df3['Age'].describe()
```

Out[68]:
```
count     550068
unique         7
top         26-35
freq      219587
Name: Age, dtype: object
```

In [69]:
```
df3['City_Category'].describe()
```

Out[69]:
```
count     550068
unique         3
top            B
freq      231173
Name: City_Category, dtype: object
```

In [70]:
```
df3['Gender'].unique()
```

Out[70]:
```
array(['F', 'M'], dtype=object)
```

In [71]:
```
df3['Age'].unique()
```

Out[71]:
```
array(['0-17', '55+', '26-35', '46-50', '51-55', '36-45', '18-25'],
      dtype=object)
```

In [72]:
```
df3['Gender'].value_counts()
```

Out[72]:
```
Gender
M    414259
F    135809
Name: count, dtype: int64
```

In [73]:
```
df3['City_Category'].value_counts()
```

Out[73]:
```
City_Category
B    231173
C    171175
A    147720
Name: count, dtype: int64
```

In [74]:
```
df3['Gender'].value_counts(ascending=True)
```

Out[74]:
```
Gender
F    135809
M    414259
Name: count, dtype: int64
```

In [75]:
```
df3['City_Category'].value_counts(ascending=True)
```

Out[75]:
```
City_Category
A    147720
C    171175
B    231173
Name: count, dtype: int64
```

In [76]:
```
df4=df.copy()

df4.max(0)
```

Out[76]:
```
User_ID                     1006040
Product_ID                  P0099942
Gender                             M
Age                              55+
Occupation                        20
City_Category                      C
Stay_In_Current_City_Years        4+
Marital_Status                     1
Product_Category_1                20
Product_Category_2              18.0
Product_Category_3              18.0
Purchase                       23961
dtype: object
```

In [77]:
```
df4.describe()
```

Out[77]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_
count	5.500680e+05	550068.000000	550068.000000	550068.000000	550068.00000
mean	1.003029e+06	8.076707	0.409653	5.404270	9.86319
std	1.727592e+03	6.522660	0.491770	3.936211	5.04945
min	1.000001e+06	0.000000	0.000000	1.000000	2.00000
25%	1.001516e+06	2.000000	0.000000	1.000000	5.00000
50%	1.003077e+06	7.000000	0.000000	5.000000	9.00000
75%	1.004478e+06	14.000000	1.000000	8.000000	15.00000
max	1.006040e+06	20.000000	1.000000	20.000000	18.00000

In [83]:
```
# Statistical functions in pandas

df5=df.copy()


# view the covariance
df5 = df5.select_dtypes(include='number')
df5.cov()
```

Out[83]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Produ
User_ID	2.984573e+06	-270.113921	17.367619	26.008008	
Occupation	-2.701139e+02	42.545100	0.077882	-0.195578	
Marital_Status	1.736762e+01	0.077882	0.241838	0.038497	
Product_Category_1	2.600801e+01	-0.195578	0.038497	15.493760	
Product_Category_2	1.442445e+01	-0.032437	0.036533	5.921467	
Product_Category_3	8.800208e+00	0.102383	0.024521	0.800453	
Purchase	4.092159e+04	682.554656	-1.144629	-6795.650007	

In [82]:
```
# Correlation: Correlation shows the linear relationship between any two array of v
# There are multiple methods to compute the correlation.


df5_numeric = df5.select_dtypes(include='number')
df5_numeric.corr()
```

Out[82]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Ca
User_ID	1.000000	-0.023971	0.020443	0.003825	
Occupation	-0.023971	1.000000	0.024280	-0.007618	
Marital_Status	0.020443	0.024280	1.000000	0.019888	
Product_Category_1	0.003825	-0.007618	0.019888	1.000000	
Product_Category_2	0.001654	-0.000985	0.014712	0.297925	
Product_Category_3	0.001238	0.003814	0.012117	0.049417	
Purchase	0.004716	0.020833	-0.000463	-0.343703	

In [84]:
```
# Data Ranking

# view the top 25 rows of ranked dataframe

df5.rank(1).head(25)
```

Out[84]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	Product_
0	7.0	4.0	1.0	2.0	3.0	
1	7.0	4.0	1.0	2.0	3.0	
2	7.0	3.0	1.0	4.0	2.0	
3	7.0	2.0	1.0	3.0	4.5	
4	7.0	5.0	1.0	2.0	3.5	
5	7.0	5.0	1.0	2.0	3.0	
6	7.0	3.0	1.5	1.5	4.0	
7	7.0	3.0	1.5	1.5	4.0	
8	7.0	3.0	1.5	1.5	4.0	
9	7.0	5.0	1.0	2.0	3.0	
10	7.0	5.0	1.0	2.0	3.0	
11	7.0	5.0	1.0	2.0	3.0	
12	7.0	5.0	1.0	2.0	3.0	
13	7.0	5.0	1.5	1.5	3.0	
14	7.0	4.0	1.0	2.0	3.0	
15	7.0	4.0	1.0	2.0	3.0	
16	7.0	5.0	1.0	2.0	3.0	
17	7.0	4.0	1.0	3.0	5.0	
18	7.0	2.0	2.0	2.0	4.0	
19	7.0	4.0	1.5	1.5	3.0	
20	7.0	3.0	1.0	2.0	4.0	
21	7.0	3.0	1.0	2.0	4.0	
22	7.0	3.0	1.0	2.0	4.0	
23	7.0	3.0	1.0	2.0	4.0	
24	7.0	4.0	1.5	1.5	3.0	

In [86]:
```
# Aggregations in pandas

df6=df.copy()
df6['Purchase'].aggregate(np.sum)
```

```
C:\Users\Windows10 Pro\AppData\Local\Temp\ipykernel_7956\1192981986.py:4: FutureWarn
ing: The provided callable <function sum at 0x000002786BD79440> is currently using S
eries.sum. In a future version of pandas, the provided callable will be used directl
y. To keep current behavior pass the string "sum" instead.
  df6['Purchase'].aggregate(np.sum)
```

Out[86]:
```
5095812742
```

In [87]:
```
df6['Purchase'].aggregate([np.sum, np.mean])
```

```
C:\Users\Windows10 Pro\AppData\Local\Temp\ipykernel_7956\2907118823.py:1: FutureWarn
ing: The provided callable <function sum at 0x000002786BD79440> is currently using S
eries.sum. In a future version of pandas, the provided callable will be used directl
y. To keep current behavior pass the string "sum" instead.
  df6['Purchase'].aggregate([np.sum, np.mean])
C:\Users\Windows10 Pro\AppData\Local\Temp\ipykernel_7956\2907118823.py:1: FutureWarn
ing: The provided callable <function mean at 0x000002786BD7A520> is currently using
Series.mean. In a future version of pandas, the provided callable will be used direc
tly. To keep current behavior pass the string "mean" instead.
  df6['Purchase'].aggregate([np.sum, np.mean])
```

Out[87]:
```
sum     5.095813e+09
mean    9.263969e+03
Name: Purchase, dtype: float64
```

In [88]:
```
df6[['Product_Category_1', 'Product_Category_2', 'Product_Category_3']].aggregate(n
```

```
C:\Users\Windows10 Pro\AppData\Local\Temp\ipykernel_7956\3418033068.py:1: FutureWarn
ing: The provided callable <function mean at 0x000002786BD7A520> is currently using
DataFrame.mean. In a future version of pandas, the provided callable will be used di
rectly. To keep current behavior pass the string "mean" instead.
  df6[['Product_Category_1', 'Product_Category_2', 'Product_Category_3']].aggregate
(np.mean)
```

Out[88]:
```
Product_Category_1     5.404270
Product_Category_2     9.863190
Product_Category_3    12.650723
dtype: float64
```

In [89]:
```
df6[['Product_Category_1', 'Product_Category_2', 'Product_Category_3']].aggregate([
```

```
C:\Users\Windows10 Pro\AppData\Local\Temp\ipykernel_7956\3684443074.py:1: FutureWarn
ing: The provided callable <function sum at 0x000002786BD79440> is currently using S
eries.sum. In a future version of pandas, the provided callable will be used directl
y. To keep current behavior pass the string "sum" instead.
  df6[['Product_Category_1', 'Product_Category_2', 'Product_Category_3']].aggregate
([np.sum, np.mean])
C:\Users\Windows10 Pro\AppData\Local\Temp\ipykernel_7956\3684443074.py:1: FutureWarn
ing: The provided callable <function mean at 0x000002786BD7A520> is currently using
Series.mean. In a future version of pandas, the provided callable will be used direc
tly. To keep current behavior pass the string "mean" instead.
  df6[['Product_Category_1', 'Product_Category_2', 'Product_Category_3']].aggregate
([np.sum, np.mean])
```

Out[89]:

	Product_Category_1	Product_Category_2	Product_Category_3
sum	2.972716e+06	5.425425e+06	6.958758e+06
mean	5.404270e+00	9.863190e+00	1.265072e+01

In [90]:
```
df6.aggregate({'Product_Category_1' : np.sum ,'Product_Category_2' : np.mean})
```

```
C:\Users\Windows10 Pro\AppData\Local\Temp\ipykernel_7956\4026433234.py:1: FutureWarn
ing: The provided callable <function sum at 0x000002786BD79440> is currently using S
eries.sum. In a future version of pandas, the provided callable will be used directl
y. To keep current behavior pass the string "sum" instead.
  df6.aggregate({'Product_Category_1' : np.sum ,'Product_Category_2' : np.mean})
C:\Users\Windows10 Pro\AppData\Local\Temp\ipykernel_7956\4026433234.py:1: FutureWarn
ing: The provided callable <function mean at 0x000002786BD7A520> is currently using
Series.mean. In a future version of pandas, the provided callable will be used direc
tly. To keep current behavior pass the string "mean" instead.
  df6.aggregate({'Product_Category_1' : np.sum ,'Product_Category_2' : np.mean})
```

Out[90]:
```
Product_Category_1    2.972716e+06
Product_Category_2    9.863190e+00
dtype: float64
```

In [91]:
```
# Pandas GroupBy operations

df8=df.copy()

df8.groupby('Gender')
```

Out[91]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x00000278016845C0>
```

In [92]:
```
# view groups of Gender column

df8.groupby('Gender').groups
```

Out[92]:
```
{'F': [0, 1, 2, 3, 14, 15, 16, 17, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 65, 66, 70, 71, 72, 73, 74, 75, 76, 77, 78, 7
9, 80, 81, 82, 83, 84, 124, 125, 126, 147, 148, 149, 150, 151, 156, 157, 158, 163,
179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 219, 222, 223, 248, 249, 250, 25
1, 252, 253, 254, 255, 256, 257, 297, 298, 299, 355, 356, 357, 358, 359, 360, 361,
362, 363, 364, 365, 366, 367, 368, 369, 373, ...], 'M': [4, 5, 6, 7, 8, 9, 10, 11,
12, 13, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 50, 51, 52, 53, 54, 55, 56, 5
7, 58, 59, 60, 61, 62, 63, 64, 67, 68, 69, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94,
95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 11
2, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 127, 128, 129, 130, 131,
132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 152, 15
3, ...]}
```

In [ ]:
```
# apply aggregation function sum with groupby

df8.groupby('Gender').sum()
```

In [ ]:
```
# alternative way to apply aggregation function sum

df8.groupby('Gender').agg(np.sum)
```

In [ ]:
```
# attribute access in python pandas

df8_grouped = df8.groupby('Gender')

print(df8_grouped.agg(np.size))
```

In [ ]:
```
df8.groupby('Gender')['Purchase'].agg([np.sum, np.mean])
```

In [ ]:
```
# Transformations: Transformation on a group or a column returns an object that is
# Thus, the transform should return a result that is the same size as that of a gro

df9=df.copy()
score = lambda x: (x - x.mean()) / x.std()*10
print(df9.groupby('Gender')['Purchase'].transform(score).head(5))
```

In [ ]:
```
# Filtration: Filtration filters the data on a defined criteria and returns the sub

df10=df.copy()
df10.groupby('Gender').filter(lambda x: len(x) > 4)
```

In [ ]:
```
# Pandas merging and joining

# let's create two dataframes

batsmen = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Rohit', 'Dhawan', 'Virat', 'Dhoni', 'Kedar'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5']})

bowler = pd.DataFrame(
   {'id':[1,2,3,4,5],
   'Name': ['Kumar', 'Bumrah', 'Shami', 'Kuldeep', 'Chahal'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5']})


print(batsmen)


print(bowler)
```

In [ ]:
```
# merge two dataframes on a key

pd.merge(batsmen, bowler, on='id')
```

In [ ]:
```
# merge two dataframes on multiple keys

pd.merge(batsmen, bowler, on=['id', 'subject_id'])
```

In [ ]:
```
# left join

pd.merge(batsmen, bowler, on='subject_id', how='left')
```

In [ ]:
```
# right join

pd.merge(batsmen, bowler, on='subject_id', how='right')
```

In [ ]:
```
# outer join

pd.merge(batsmen, bowler, on='subject_id', how='outer')
```

In [ ]:
```
# inner join

pd.merge(batsmen, bowler, on='subject_id', how='inner')
```

In [ ]:
```
# Pandas concatenation operation

# let's create two dataframes

batsmen = pd.DataFrame({
   'id':[1,2,3,4,5],
   'Name': ['Rohit', 'Dhawan', 'Virat', 'Dhoni', 'Kedar'],
   'subject_id':['sub1','sub2','sub4','sub6','sub5']})

bowler = pd.DataFrame(
   {'id':[1,2,3,4,5],
   'Name': ['Kumar', 'Bumrah', 'Shami', 'Kuldeep', 'Chahal'],
   'subject_id':['sub2','sub4','sub3','sub6','sub5']})


print(batsmen)
print(bowler)
```

In [ ]:
```
# concatenate the dataframes


team=[batsmen, bowler]

pd.concat(team)
```

In [ ]:
```
# associate keys with the dataframes

pd.concat(team, keys=['x', 'y'])
```

In [ ]:
```
pd.concat(team, keys=['x', 'y'], ignore_index=True)
```

In [ ]:
```
pd.concat(team, axis=1)
```

In [ ]:
```
# Concatenating using append

batsmen.append(bowler)
```

In [ ]:
```
# Reshaping by melt and pivot

df11=df.copy()
```

```
df11.columns
```

In [ ]:
```
df12=(pd.melt(frame=df11, id_vars=['User_ID','Product_ID', 'Gender','Age','Occupati
                            'Marital_Status','Purchase'],
                    value_vars=['Product_Category_1','Product_Category_2','Product_
                    var_name='Product_Category', value_name='Amount'))

df12.head(10)
```

In [ ]:
```
df13=df12[['Product_Category', 'Amount']]

df14=df13.pivot(index=None, columns='Product_Category', values='Amount')

df14.head(25)
```

In [ ]:
```
cols=pd.MultiIndex.from_tuples([('weight', 'kg'), ('weight', 'pounds')])

df15=pd.DataFrame([[75,165], [60, 132]],
                  index=['husband', 'wife'],
                  columns=cols)

df15
```

In [ ]:
```
df16=df15.stack()

df16
```

Options and customization with pandas

- Pandas provide API to customize some aspects of its behavior. In most cases, we would like to adjust the display related options.
- The API is composed of five relevant functions. They are as follows :−
- get_option()
- set_option()
- reset_option()
- describe_option()
- option_context()

In [ ]:
```
# 1. get_option(param): get_option() takes a single parameter and returns the value
# display maximum rows

pd.get_option("display.max_rows")
```

In [ ]:
```
# display maximum columns

pd.get_option("display.max_columns")
```

In [ ]:
```
# 2. set_option(param,value)
set_option() takes two arguments and sets the value to the parameter as shown below

# set maximum rows

pd.set_option("display.max_rows", 80)

pd.get_option("display.max_rows")
```

In [ ]:
```
# set maximum columns

pd.set_option("display.max_columns", 30)

pd.get_option("display.max_columns")
```

In [ ]:
```
# 3. reset_option(param)
reset_option() takes an argument and sets the value back to the default value.

# display maximum rows

pd.reset_option("display.max_rows")

pd.get_option("display.max_rows")
```

In [ ]:
```
# display maximum columns

pd.reset_option("display.max_columns")

pd.get_option("display.max_columns")
```

In [ ]:
```
# 4. describe_option(param)
describe_option() prints the description of the argument.

# description of the display maximum rows parameter

pd.describe_option("display.max_rows")
```

In [ ]:
```
# 5. option_context()
# option_context() context manager is used to set the option in with statement temp
# Option values are restored automatically when you exit with block.

# set the parameter value with option_context

with pd.option_context("display.max_rows",10):
   print(pd.get_option("display.max_rows"))
   print(pd.get_option("display.max_rows"))
```