Experiment 1

Aim -

Data preparation using NumPy and Pandas.

Todo -

1. Load data in Pandas.
2. Description of the dataset.
3. Drop columns that aren't useful.
4. Drop rows with maximum missing values.
5. Take care of missing data.
6. Create dummy variables.
7. Find out outliers (manually).
8. Standardization and normalization of columns.

Dataset -

The dataset contains data about customers' purchases during the Black Friday sale. This dataset was taken from Kaggle. The dataset has 550k rows and 12 columns. The various columns of the dataset are age, marital status, gender, total purchase amount, and many other features.

Theory -
pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is a fast, powerful, flexible, and easy-to-use open-source data analysis and manipulation tool.

NumPy is a library for the Python programming language, adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. It offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

Results -

1. Load dataset.

In [1]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
```

read_csv():
Read a comma-separated values (CSV) file into DataFrame.

head():

This function returns the first n rows for the object based on position. It is useful for quickly testing if your object has the right type of data in it.

In [2]:
```
# Read the data into datafrome
df = pd.read_csv("train.csv")

# Display the first `n` rows.
df.head()
```

Out[2]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Product_Category_2	Product
0	1000001	P00069042	F	0-17	10	A	2	0	3	NaN	
1	1000001	P00248942	F	0-17	10	A	2	0	1	6.0	
2	1000001	P00087842	F	0-17	10	A	2	0	12	NaN	
3	1000001	P00085442	F	0-17	10	A	2	0	12	14.0	
4	1000002	P00285442	M	55+	16	C	4+	0	8	NaN	

info():

This method prints information about a DataFrame including the index dtype and columns, non-null values and memory usage.

In [3]:
```
# Print a concise summary of a DataFrame.
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
```

2. Describe dataset.

describe():

Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.

```
In [4]: # Generate descriptive statistics.
        df.describe()
```

Out[4]:

	User_ID	Occupation	Marital_Status	Product_Category_1	Product_Category_2	Product_Category_3	Purchase
count	5.500680e+05	550068.000000	550068.000000	550068.000000	376430.000000	166821.000000	550068.000000
mean	1.003029e+06	8.076707	0.409653	5.404270	9.842329	12.668243	9263.968713
std	1.727592e+03	6.522660	0.491770	3.936211	5.086590	4.125338	5023.065394
min	1.000001e+06	0.000000	0.000000	1.000000	2.000000	3.000000	12.000000
25%	1.001516e+06	2.000000	0.000000	1.000000	5.000000	9.000000	5823.000000
50%	1.003077e+06	7.000000	0.000000	5.000000	9.000000	14.000000	8047.000000
75%	1.004478e+06	14.000000	1.000000	8.000000	15.000000	16.000000	12054.000000
max	1.006040e+06	20.000000	1.000000	20.000000	18.000000	18.000000	23961.000000

3. Drop columns that aren't useful.

drop():

Remove rows or columns by specifying label names and corresponding axis, or by specifying direct index or column names. When using a multi-index, labels on different levels can be removed by specifying the level.

```
In [5]: # Drop specified labels from rows or columns.
        df.drop(labels=['Product_Category_2','Product_Category_3'],axis=1,inplace=True)
        df.head()
```

Out[5]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Purchase
0	1000001	P00069042	F	0-17	10	A	2	0	3	8370
1	1000001	P00248942	F	0-17	10	A	2	0	1	15200
2	1000001	P00087842	F	0-17	10	A	2	0	12	1422
3	1000001	P00085442	F	0-17	10	A	2	0	12	1057
4	1000002	P00285442	M	55+	16	C	4+	0	8	7969

isnull():

Detect missing values.

Return a boolean same-sized object indicating if the values are NA. NA values, such as None or numpy. NaN gets mapped to True values. Everything else gets mapped to False values. Characters such as empty strings '' or numpy.inf are not considered NA values (unless you set pandas.options.mode.use_inf_as_na = True).

sum():

Returns a data frame or Series of the same size containing the cumulative sum.

```
In [6]: # Count of the number of missing values in each column.
        df.isnull().sum()

Out[6]: 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
        Purchase                      0
        dtype: int64
```

4. Create dummy variables.

value_counts():

Return a Series containing counts of unique rows in the DataFrame.

```
In [7]: # Printing count of unique values in each columns.
        cols = list(df.columns)
        delete_cols = ["User_ID", "Product_ID", "Product_Category_1", "Purchase"]

        for i in delete_cols:
            cols.remove(i)

        for col in cols:
            print(df[col].value_counts())

M    414259
F    135809
Name: Gender, dtype: int64
26-35    219587
36-45    110013
18-25     99660
46-50     45701
51-55     38501
55+       21504
0-17      15102
Name: Age, dtype: int64
4     72308
0     69638
7     59133
1     47426
17    40043
20    33562
12    31179
14    27309
2     26588
16    25371
6     20355
3     17650
10    12930
5     12177
15    12165
11    11586
19     8461
13     7728
18     6622
```

concat():

Concatenate pandas objects along a particular axis.
Allows optional set logic along the other axes.
Can also add a layer of hierarchical indexing on the concatenation axis, which may be useful if the labels are the same (or overlapping) on the passed axis number.

get_dummies():
Convert categorical variable into dummy/indicator variables.

In [8]:
```python
# Creating dummies.
colm = ['Marital_Status','Gender','City_Category']
dummies = []
for col in colm:
    dummies.append(pd.get_dummies(df[col]))

dummies = pd.concat(dummies,axis=1)
df = pd.concat((df,dummies),axis=1)

df.head()
```

Out[8]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Purchase	0	1	F	M	A	B
0	1000001	P00069042	F	0-17	10	A	2	0	3	8370	1	0	1	0	1	0
1	1000001	P00248942	F	0-17	10	A	2	0	1	15200	1	0	1	0	1	0
2	1000001	P00087842	F	0-17	10	A	2	0	12	1422	1	0	1	0	1	0
3	1000001	P00085442	F	0-17	10	A	2	0	12	1057	1	0	1	0	1	0
4	1000002	P00285442	M	55+	16	C	4+	0	8	7969	1	0	0	1	0	0

In [9]:
```python
# Renaming the columns
col = df.columns
df.columns = [  'User_ID', 'Product_ID', 'Gender', 'Age', 'Occupation', 'City_Category', 'Stay_In_Current_City_Years',
              'Marital_Status', 'Product_Category_1', 'Purchase', 'Married', 'Not_Married', 'Female', 'Male',
              'City_Category_A', 'City_Category_B', 'City_Category_C']

df.head()
```

Out[9]:

	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Stay_In_Current_City_Years	Marital_Status	Product_Category_1	Purchase	Married	Not_Marr
0	1000001	P00069042	F	0-17	10	A	2	0	3	8370	1	
1	1000001	P00248942	F	0-17	10	A	2	0	1	15200	1	
2	1000001	P00087842	F	0-17	10	A	2	0	12	1422	1	
3	1000001	P00085442	F	0-17	10	A	2	0	12	1057	1	
4	1000002	P00285442	M	55+	16	C	4+	0	8	7969	1	

In [10]:
```python
# Dropping redundant columns.
df.drop(['Gender','Marital_Status','City_Category'],axis=1,inplace=True)
df.head()
```

Out[10]:

	User_ID	Product_ID	Age	Occupation	Stay_In_Current_City_Years	Product_Category_1	Purchase	Married	Not_Married	Female	Male	City_Category_A	Ci
0	1000001	P00069042	0-17	10	2	3	8370	1	0	1	0	1	
1	1000001	P00248942	0-17	10	2	1	15200	1	0	1	0	1	
2	1000001	P00087842	0-17	10	2	12	1422	1	0	1	0	1	
3	1000001	P00085442	0-17	10	2	12	1057	1	0	1	0	1	
4	1000002	P00285442	55+	16	4+	8	7969	1	0	0	1	0	

In [11]:
```python
df['Purchase'].describe()
```

Out[11]:
```
count    550068.000000
mean       9263.968713
std        5023.065394
min          12.000000
25%        5823.000000
50%        8047.000000
75%       12054.000000
max       23961.000000
Name: Purchase, dtype: float64
```

5. Find out outliers (manually).

In [12]:
```python
# Calculating the Outliers
interQuartileRange = (df['Purchase'].describe()[6] - df['Purchase'].describe()[4])
upper = df['Purchase'].describe()[6] + (1.5 * interQuartileRange)
lower = df['Purchase'].describe()[4] - (1.5 * interQuartileRange)

print("IQR (Q3-Q1): " + str(interQuartileRange))
print("Lower: " + str(lower))
print("Upper: " + str(upper))

outliers = []
for i in range(df['Purchase'].shape[0]):
    if(df['Purchase'].iloc[i]>upper):
        outliers.append([i,df['Purchase'].iloc[i]])

print("Total number of outliers: " + str(len(outliers)))
```

```
IQR (Q3-Q1): 6231.0
Lower: -3523.5
Upper: 21400.5
Total number of outliers: 2677
```

6. Standardization and normalization of columns.

In [13]:
```python
# Normalization on Purchase column.
# normal_value = (value - min)/(max - min)
df.Purchase = (df.Purchase - df.Purchase.min()) / (df.Purchase.max() - df.Purchase.min())
df.head()
```

Out[13]:

	User_ID	Product_ID	Age	Occupation	Stay_In_Current_City_Years	Product_Category_1	Purchase	Married	Not_Married	Female	Male	City_Category_A	Ci
0	1000001	P00069042	0-17	10	2	3	0.348992	1	0	1	0	1	
1	1000001	P00248942	0-17	10	2	1	0.634181	1	0	1	0	1	
2	1000001	P00087842	0-17	10	2	12	0.058875	1	0	1	0	1	
3	1000001	P00085442	0-17	10	2	12	0.043634	1	0	1	0	1	
4	1000002	P00285442	55+	16	4+	8	0.332248	1	0	0	1	0	

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

Out[14]:

	User_ID	Occupation	Product_Category_1	Purchase	Married	Not_Married	Female	Male	City_Category_A	City_C
count	5.500680e+05	550068.000000	550068.000000	550068.000000	550068.000000	550068.000000	550068.000000	550068.000000	550068.000000	5500
mean	1.003029e+06	8.076707	5.404270	0.386320	0.590347	0.409653	0.246895	0.753105	0.268549	
std	1.727592e+03	6.522660	3.936211	0.209740	0.491770	0.491770	0.431205	0.431205	0.443205	
min	1.000001e+06	0.000000	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.001516e+06	2.000000	1.000000	0.242641	0.000000	0.000000	0.000000	1.000000	0.000000	
50%	1.003077e+06	7.000000	5.000000	0.335505	1.000000	0.000000	0.000000	1.000000	0.000000	
75%	1.004478e+06	14.000000	8.000000	0.502818	1.000000	1.000000	0.000000	1.000000	1.000000	
max	1.006040e+06	20.000000	20.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	

Conclusion-

We have successfully preprocessed the data. We cleaned the dataset by removing missing values and redundant columns. We also used dummy variables and then normalized the values.