#Team members

#20201381733 ياسمين مجدي علي البرعي

#20201381068 جيهان انور محمد

#20201378053 هبة محمد صالح

#20201378044 ماهيتاب محمد متولي عبداللطيف

#20201290250 ندى سعد حسن ابوبكر

#20201447071 هبه الله محمد احمد محمود

#20201323258 ميان اسلام محمد احمد

Part 1 project data mining:-

We begin our project by importing the libraries that we are going to need in the project.

Then we load the data using (read_csv) function from pandas library as shown in fig no.1.

```
In [522]: #import needed libraries and take an object from them
          import numpy as np
          import pandas as pd
          import matplotlib.pyplot as plt
          from sklearn.neighbors import KNeighborsClassifier
          from sklearn.metrics import accuracy_score
          from sklearn.model_selection import train_test_split
          from sklearn.tree import DecisionTreeClassifier
          from sklearn.preprocessing import LabelEncoder
          from sklearn.metrics import confusion_matrix,accuracy_score
          from sklearn.preprocessing import StandardScaler
          from sklearn.naive_bayes import GaussianNB
```

Fig no.1

Then we get information about the data using (info()) function as shown in fig no.2.

```
In [50]: #print all information about the Data
         Data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17446 entries, 0 to 17445
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   SIZE            17446 non-null  int64
 1   FUEL            17445 non-null  object
 2   DISTANCE        17433 non-null  float64
 3   DESIBEL         17413 non-null  float64
 4   AIRFLOW         17382 non-null  float64
 5   FREQUENCY       17402 non-null  float64
 6   BOS             17426 non-null  float64
 7   Operation_Code  17444 non-null  object
 8   STATUS          17446 non-null  int64
dtypes: float64(5), int64(2), object(2)
memory usage: 1.2+ MB
```

Fig no.2

As we know we should always handle the data before using it so, we will handle some of the problems (e.g. missing values, noise, duplication,……).

<u>1st : the missing values:</u>

Firstly, we get to know the number of the data missed, so we use (isnull(). sum()) function to count the number of null object in the data so we can deal with as shown in fig. no.3.

```
In [51]: #getting the number of the missed values in the data using the following function
         Data.isnull().sum()

Out[51]: SIZE               0
         FUEL               1
         DISTANCE          13
         DESIBEL           33
         AIRFLOW           64
         FREQUENCY         44
         BOS               20
         Operation_Code     2
         STATUS             0
         dtype: int64
```

Fig. no.3

Missing data problem can be solved by many ways in our project we choose the way of removing the rows that contain missing values.

Using a function (dropna()) we drop all the rows containing a null value.

Then we use the function (isnull().sum()) to make sure that we get rid of all unwanted values in our data as shown in fig. no.4 .

```
In [52]: #we handle the problem of missing data by removind the row which contain the missing values
         #using dropna() function we drop the missing valus's rows
         Data=Data.dropna()
```

```
In [53]: #then checking whether there are missing values or not again to make sure that we handle the problem of missing values
         Data.isnull().sum()

Out[53]: SIZE              0
         FUEL              0
         DISTANCE          0
         DESIBEL           0
         AIRFLOW           0
         FREQUENCY         0
         BOS               0
         Operation_Code    0
         STATUS            0
         dtype: int64
```

Fig. no.4

Then we check on the data after cleaning the missing values as shown fig. no.5.

```
In [54]: #print information about the Data after remove missing values
         Data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 17273 entries, 2 to 17445
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   SIZE            17273 non-null  int64
 1   FUEL            17273 non-null  object
 2   DISTANCE        17273 non-null  float64
 3   DESIBEL         17273 non-null  float64
 4   AIRFLOW         17273 non-null  float64
 5   FREQUENCY       17273 non-null  float64
 6   BOS             17273 non-null  float64
 7   Operation_Code  17273 non-null  object
 8   STATUS          17273 non-null  int64
dtypes: float64(5), int64(2), object(2)
memory usage: 1.3+ MB
```

Fig. no.5

<u>2nd : noise data:-</u>

After we remove missing values from our data, we will remove noise from data.

Here we see our boxplot to show the outliers as shown in fig. no.6

```
In [55]: #boxplot when data is noise
         plt.figure(figsize = (10, 6))
         Data.boxplot()

Out[55]: <AxesSubplot:>
```

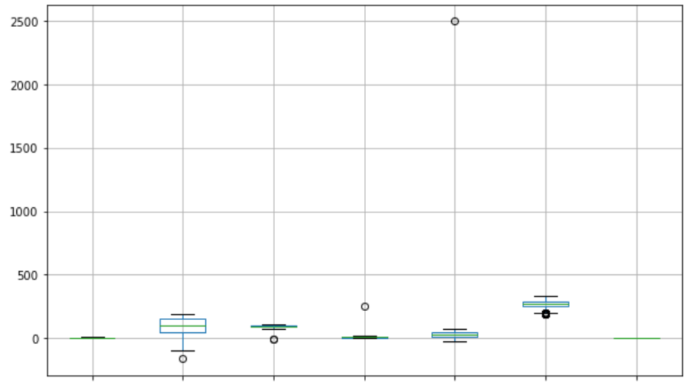


Fig. no.6

we have outliers in 5 columns (DISTANCE, DESIBEL, AIRFLOW, FREQUENCY, BOS), so we want to remove all outliers from this columns

we use this method to remove them , this method calculate Q75, Q25, intr_qr to know the max and min value of each column and turn all values that bigger than max and smaller than min to null value

we use Data.isnull().sum() to sum all null values in each column.

we use Data.dropna(axis = 0) to drop all null values

after do this method in each column that we use data.info() to print information about the Data after remove null values that we found in each column and after each process we check on the data as shown from fig. no.7 to fig. no.8.

```
In [58]: #drop all null values that we found in column 'DISTANCE'
         Data = Data.dropna(axis = 0)
```

```
In [59]: #print information about the Data after remove null values that we found in 'DISTANCE' column
         Data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 17272 entries, 2 to 17445
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   SIZE            17272 non-null  int64  
 1   FUEL            17272 non-null  object 
 2   DISTANCE        17272 non-null  float64
 3   DESIBEL         17272 non-null  float64
 4   AIRFLOW         17272 non-null  float64
 5   FREQUENCY       17272 non-null  float64
 6   BOS             17272 non-null  float64
 7   Operation_Code  17272 non-null  object 
 8   STATUS          17272 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 1.3+ MB
```

Fig. no.7

In [60]:
```
#remove outliers from column 'AIRFLOW'
#remove all data that bigger than max and smaller than min after we calculate max, min values
#by change all outlier values to null vales to drop them
for y in ['AIRFLOW']:
    Q75,Q25 = np.percentile(Data.loc[:,y],[75,25])
    intr_qr = Q75-Q25

    max = Q75+(1.5*intr_qr)
    min = Q25-(1.5*intr_qr)

    Data.loc[Data[y] < min,y] = np.nan
    Data.loc[Data[y] > max,y] = np.nan
```

In [61]:
```
#sum all null vales in 'AIRFLOW' column
Data.isnull().sum()
```

Out[61]:
```
SIZE              0
FUEL              0
DISTANCE          0
DESIBEL           0
AIRFLOW           1
FREQUENCY         0
BOS               0
Operation_Code    0
STATUS            0
dtype: int64
```

In [62]:
```
#drop all null values that we found in 'AIRFLOW' column
Data = Data.dropna(axis = 0)
```

Fig. no.8

In [62]:
```
#drop all null values that we found in 'AIRFLOW' column
Data = Data.dropna(axis = 0)
```

In [63]:
```
#print information about the Data after remove null values that we found in 'AIRFLOW' column
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17271 entries, 2 to 17445
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   SIZE            17271 non-null  int64  
 1   FUEL            17271 non-null  object 
 2   DISTANCE        17271 non-null  float64
 3   DESIBEL         17271 non-null  float64
 4   AIRFLOW         17271 non-null  float64
 5   FREQUENCY       17271 non-null  float64
 6   BOS             17271 non-null  float64
 7   Operation_Code  17271 non-null  object 
 8   STATUS          17271 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 1.3+ MB
```

Fig. no.9

In [64]:
```python
#remove outliers from column 'BOS'
#remove all data that bigger than max and smaller than min after we calculate max, min values
#by change all outlier values to null vales to drop them
for z in ['BOS']:
    Q75,Q25 = np.percentile(Data.loc[:,z],[75,25])
    intr_qr = Q75-Q25

    max = Q75+(1.5*intr_qr)
    min = Q25-(1.5*intr_qr)

    Data.loc[Data[z] < min,z] = np.nan
    Data.loc[Data[z] > max,z] = np.nan
```

In [65]:
```python
#sum all null vales in 'BOS' column
Data.isnull().sum()
```

Out[65]:
```
SIZE               0
FUEL               0
DISTANCE           0
DESIBEL            0
AIRFLOW            0
FREQUENCY          0
BOS               59
Operation_Code     0
STATUS             0
dtype: int64
```

Fig. no.10

In [66]:
```python
#drop all null values that we found in 'BOS' column
Data = Data.dropna(axis = 0)
```

In [67]:
```python
#print information about the Data after remove null values that we found in 'BOS' column
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17212 entries, 2 to 17445
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   SIZE            17212 non-null  int64  
 1   FUEL            17212 non-null  object 
 2   DISTANCE        17212 non-null  float64
 3   DESIBEL         17212 non-null  float64
 4   AIRFLOW         17212 non-null  float64
 5   FREQUENCY       17212 non-null  float64
 6   BOS             17212 non-null  float64
 7   Operation_Code  17212 non-null  object 
 8   STATUS          17212 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 1.3+ MB
```

Fig. no.11

In [68]:
```python
#remove outliers from column 'FREQUENCY'
#remove all data that bigger than max and smaller than min after we calculate max, min values
#by change all outlier values to null vales to drop them
for j in ['FREQUENCY']:
    Q75,Q25 = np.percentile(Data.loc[:,j],[75,25])
    intr_qr = Q75-Q25

    max = Q75+(1.5*intr_qr)
    min = Q25-(1.5*intr_qr)

    Data.loc[Data[j] < min,j] = np.nan
    Data.loc[Data[j] > max,j] = np.nan
```

In [69]:
```python
#sum all null vales in 'FREQUENCY' column
Data.isnull().sum()
```

Out[69]:
```
SIZE              0
FUEL              0
DISTANCE          0
DESIBEL           0
AIRFLOW           0
FREQUENCY         1
BOS               0
Operation_Code    0
STATUS            0
dtype: int64
```

Fig. no.12

In [62]:
```python
#drop all null values that we found in 'AIRFLOW' column
Data = Data.dropna(axis = 0)
```

In [63]:
```python
#print information about the Data after remove null values that we found in 'AIRFLOW' column
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17271 entries, 2 to 17445
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   SIZE            17271 non-null  int64  
 1   FUEL            17271 non-null  object 
 2   DISTANCE        17271 non-null  float64
 3   DESIBEL         17271 non-null  float64
 4   AIRFLOW         17271 non-null  float64
 5   FREQUENCY       17271 non-null  float64
 6   BOS             17271 non-null  float64
 7   Operation_Code  17271 non-null  object 
 8   STATUS          17271 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 1.3+ MB
```

Fig. no.13

```
In [64]: #remove outliers from column 'BOS'
         #remove all data that bigger than max and smaller than min after we calculate max, min values
         #by change all outlier values to null vales to drop them
         for z in ['BOS']:
             Q75,Q25 = np.percentile(Data.loc[:,z],[75,25])
             intr_qr = Q75-Q25

             max = Q75+(1.5*intr_qr)
             min = Q25-(1.5*intr_qr)

             Data.loc[Data[z] < min,z] = np.nan
             Data.loc[Data[z] > max,z] = np.nan
```

```
In [65]: #sum all null vales in 'BOS' column
         Data.isnull().sum()

Out[65]: SIZE              0
         FUEL              0
         DISTANCE          0
         DESIBEL           0
         AIRFLOW           0
         FREQUENCY         0
         BOS              59
         Operation_Code    0
         STATUS            0
         dtype: int64
```

```
In [66]: #drop all null values that we found in 'BOS' column
         Data = Data.dropna(axis = 0)
```

Fig. no.14

```
In [66]: #drop all null values that we found in 'BOS' column
         Data = Data.dropna(axis = 0)
```

```
In [67]: #print information about the Data after remove null values that we found in 'BOS' column
         Data.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 17212 entries, 2 to 17445
         Data columns (total 9 columns):
          #   Column          Non-Null Count  Dtype
         ---  ------          --------------  -----
          0   SIZE            17212 non-null  int64
          1   FUEL            17212 non-null  object
          2   DISTANCE        17212 non-null  float64
          3   DESIBEL         17212 non-null  float64
          4   AIRFLOW         17212 non-null  float64
          5   FREQUENCY       17212 non-null  float64
          6   BOS             17212 non-null  float64
          7   Operation_Code  17212 non-null  object
          8   STATUS          17212 non-null  int64
         dtypes: float64(5), int64(2), object(2)
```

Fig. no. 15

```
In [71]: #print information about the Data after remove null values that we found in 'FREQUENCY' column
         Data.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 17211 entries, 2 to 17445
         Data columns (total 9 columns):
          #   Column          Non-Null Count  Dtype
         ---  ------          --------------  -----
          0   SIZE            17211 non-null  int64
          1   FUEL            17211 non-null  object
          2   DISTANCE        17211 non-null  float64
          3   DESIBEL         17211 non-null  float64
          4   AIRFLOW         17211 non-null  float64
          5   FREQUENCY       17211 non-null  float64
          6   BOS             17211 non-null  float64
          7   Operation_Code  17211 non-null  object
          8   STATUS          17211 non-null  int64
         dtypes: float64(5), int64(2), object(2)
         memory usage: 1.3+ MB
```

Fig. no. 16

In [72]:
```
#remove outliers from column 'DESIBEL'
#remove all data that bigger than max and smaller than min after we calculate max, min values
#by change all outlier values to null vales to drop them
for i in ['DESIBEL']:
    Q75,Q25 = np.percentile(Data.loc[:,i],[75,25])
    intr_qr = Q75-Q25

    max = Q75+(1.5*intr_qr)
    min = Q25-(1.5*intr_qr)

    Data.loc[Data[i] < min,i] = np.nan
    Data.loc[Data[i] > max,i] = np.nan
```

In [73]:
```
#sum all null vales in 'DESIBEL' column
Data.isnull().sum()
```

Out[73]:
```
SIZE              0
FUEL              0
DISTANCE          0
DESIBEL           2
AIRFLOW           0
FREQUENCY         0
BOS               0
Operation_Code    0
STATUS            0
dtype: int64
```

In [74]:
```
#drop all null values that we found in column 'DESIBEL'
Data = Data.dropna(axis = 0)
```

Fig. no.17

In [75]:
```
#print information about the Data after remove null values that we found in 'DESIBEL' column
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17209 entries, 2 to 17445
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   SIZE            17209 non-null  int64  
 1   FUEL            17209 non-null  object 
 2   DISTANCE        17209 non-null  float64
 3   DESIBEL         17209 non-null  float64
 4   AIRFLOW         17209 non-null  float64
 5   FREQUENCY       17209 non-null  float64
 6   BOS             17209 non-null  float64
 7   Operation_Code  17209 non-null  object 
 8   STATUS          17209 non-null  int64  
dtypes: float64(5), int64(2), object(2)
memory usage: 1.3+ MB
```

Fig. no.18

We do this in each column have outliers until remove all outliers.

Aftwe remove outliers this is our boxplot after cleaning the data from noise ,null and remove outliers as shown in fig. no.19.

```
In [76]: #boxplot after cleaning the data from noise,null and remove outliers
         plt.figure(figsize = (10, 6))
         Data.boxplot()

Out[76]: <AxesSubplot:>
```

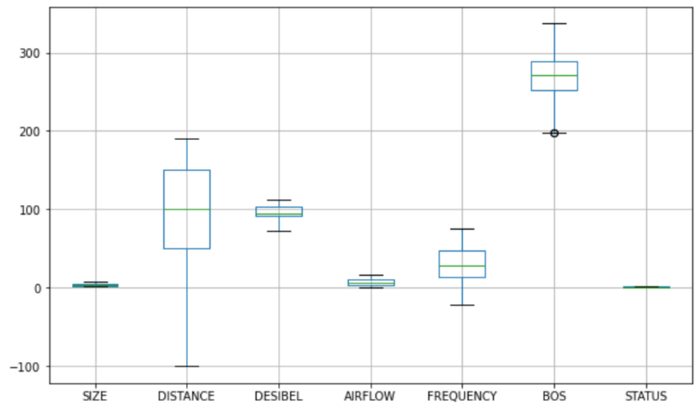


Fig. no.19

3rd : duplication:-

we want to remove duplicate from our data so first we used Data.duplicated() to know duplicated rows

second we used duplicated().sum() to know the sum of all duplicated data

third we want to drop them so we used Data.drop_duplicates(inplace=True) to drop all duplicated data

finally we want to make sure that all duplicated data are removed so we used Data.duplicated().sum() again as shown in fig. no.20.

```
In [77]: #show the duplicate
         Data[Data.duplicated()].head()
```

Out[77]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	BOS	Operation_Code	STATUS
6936	4	thinner	160.0	105.0	4.4	36.0	304.75	35970	0
6937	6	lpg	90.0	102.0	0.0	68.0	272.10	47559b	0
6938	1	gasoline	180.0	87.0	2.8	5.0	225.85	47580	0
6939	2	thinner	170.0	76.0	0.0	1.0	225.80	47591d	0
7885	3	kerosene	60.0	105.0	12.5	50.0	310.75	22749	1

```
In [78]: #sum all duplicated data
         Data.duplicated().sum()
```

Out[78]: 80

```
In [79]: #drop all duplicated data
         Data.drop_duplicates(inplace=True)
```

```
In [80]: ##sum all duplicate data to make sure that there is no duplication
         Data.duplicated().sum()
```

Out[80]: 0

fig. no.20

4rth: irrelevant features:-

many of features will make our model has lack of accuracy so we need remove irrelevant attributes which are columns we do not need ,in this case we need remove "Operation_Code" which is the column we do not need in our inputs to the model because it is do not mean anything and do not affect in our output("status") as shown in fig. no.21.

```
In [81]: #this column don't related to our output(status)so we remove it
         Data=Data.drop(columns="Operation_Code")
         Data
```

Out[81]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	BOS	STATUS
2	4	thinner	160.0	105.0	4.4	36.0	304.75	0
3	6	lpg	90.0	102.0	0.0	68.0	272.10	0
4	1	gasoline	180.0	87.0	2.8	5.0	225.85	0
5	2	thinner	170.0	76.0	0.0	1.0	225.80	0
6	2	kerosene	80.0	75.0	0.0	1.0	217.25	0
...	...	...	...	...	...	...	...	...
17440	1	thinner	80.0	106.0	11.2	46.0	275.30	1
17441	2	thinner	50.0	82.0	3.2	2.0	250.10	0
17442	5	kerosene	160.0	94.0	1.9	55.0	251.70	0
17444	1	kerosene	20.0	105.0	13.5	30.0	307.75	1
17445	1	gasoline	20.0	91.0	11.9	12.0	244.05	1

17129 rows × 8 columns

Fig. no.21

5th: correlation:-

Correlation is to find how strong the data is related to each other and as it increases the relation between them increases the need to delete the reason for the strong relationship. So, the function (.corr()) is a built in function in python to calculate that correlation. So, as shown in fig. no.22 and fig. no23 we use it to show the matrix which hold the values of correlation.

In [82]:
```
#create a matrix filled with correlation values for each two columns
Data.corr()
```

Out[82]:

	SIZE	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	BOS	STATUS
SIZE	1.000000	-0.001222	0.000578	0.000070	-0.000310	0.001389	-0.097070
DISTANCE	-0.001222	1.000000	-0.238583	-0.708477	0.002117	-0.197739	-0.644349
DESIBEL	0.000578	-0.238583	1.000000	0.369381	0.557475	0.812272	0.197569
AIRFLOW	0.000070	-0.708477	0.369381	1.000000	-0.221464	0.303507	0.760283
FREQUENCY	-0.000310	0.002117	0.557475	-0.221464	1.000000	0.452123	-0.250557
BOS	0.001389	-0.197739	0.812272	0.303507	0.452123	1.000000	0.159832
STATUS	-0.097070	-0.644349	0.197569	0.760283	-0.250557	0.159832	1.000000

Fig. no.22

In [83]:
```
#create a matrix filled with correlation values for each two columns with absolute value
CorrelationMatrix = Data.corr().abs()
CorrelationMatrix
```

Out[83]:

	SIZE	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	BOS	STATUS
SIZE	1.000000	0.001222	0.000578	0.000070	0.000310	0.001389	0.097070
DISTANCE	0.001222	1.000000	0.238583	0.708477	0.002117	0.197739	0.644349
DESIBEL	0.000578	0.238583	1.000000	0.369381	0.557475	0.812272	0.197569
AIRFLOW	0.000070	0.708477	0.369381	1.000000	0.221464	0.303507	0.760283
FREQUENCY	0.000310	0.002117	0.557475	0.221464	1.000000	0.452123	0.250557
BOS	0.001389	0.197739	0.812272	0.303507	0.452123	1.000000	0.159832
STATUS	0.097070	0.644349	0.197569	0.760283	0.250557	0.159832	1.000000

Fig. no.23

the Correlation matrix will be mirror image about the diagonal and all the diagonal elements will be 1. So, It does not matter that we select the upper triangular or lower triangular part of the correlation matrix but we should not include the diagonal elements. So we are selecting the upper traingular as shown in fig. no.24.

```
In [84]: #Selecting the Upper triangular matrix
up_tri = CorrelationMatrix.where(np.triu(np.ones(CorrelationMatrix.shape),k=1).astype(np.bool))
up_tri
```

```
C:\Users\MCC\AppData\Local\Temp/ipykernel_2900/2541172571.py:2: DeprecationWarning: `np.bool` is a deprecated alias for the bui
ltin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifi
cally wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  up_tri = CorrelationMatrix.where(np.triu(np.ones(CorrelationMatrix.shape),k=1).astype(np.bool))
```

Out[84]:

	SIZE	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	BOS	STATUS
SIZE	NaN	0.001222	0.000578	0.000070	0.000310	0.001389	0.097070
DISTANCE	NaN	NaN	0.238583	0.708477	0.002117	0.197739	0.644349
DESIBEL	NaN	NaN	NaN	0.369381	0.557475	0.812272	0.197569
AIRFLOW	NaN	NaN	NaN	NaN	0.221464	0.303507	0.760283
FREQUENCY	NaN	NaN	NaN	NaN	NaN	0.452123	0.250557
BOS	NaN	NaN	NaN	NaN	NaN	NaN	0.159832
STATUS	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
In [85]: #this method help us to know which columns with high correlation that we will drop
C_drop = [column for column in up_tri.columns if any(up_tri[column] >= 0.8)]
print(C_drop)

['BOS']
```

Fig. no.24

Then we determine which features are make the correlation more than 0.8 and delete this features as shown in fig. no.25.

```
In [85]: #this method help us to know which columns with high correlation that we will drop
C_drop = [column for column in up_tri.columns if any(up_tri[column] >= 0.8)]
print(C_drop)

['BOS']
```

```
In [86]: #Drop columns  with high correlation 'BOS'
Data=Data.drop(Data[C_drop],axis=1)
Data.head()
```

Out[86]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	STATUS
2	4	thinner	160.0	105.0	4.4	36.0	0
3	6	lpg	90.0	102.0	0.0	68.0	0
4	1	gasoline	180.0	87.0	2.8	5.0	0
5	2	thinner	170.0	76.0	0.0	1.0	0
6	2	kerosene	80.0	75.0	0.0	1.0	0

fig. no.25

6th : discretization:-

We use discretion to change discrete data into continues to compress the size of the data by divide the data into intervals as shown in the fig. no.26

```
In [87]: #by using cut () function we do the discretization to each numerical attribute
         #T_SIZE is the types of the size which divided into 3 parts
         Data['T_SIZE'] = pd.cut(Data['SIZE'],3,labels=['small','medium','larg'])

In [88]: #T_DISTANCE is the types of distance which divided into 5 part
         Data['T_DISTANCE'] = pd.cut(Data['DISTANCE'],5,labels=['short','Below_average','Average','Above_Average','long'])

In [89]: #T_DESTBEL is how strong the sound
         Data['T_DESIBEL'] = pd.cut(Data['DESIBEL'],3,labels=['low','normal','high'])

In [90]: #T_AIRFLOW is how much the airflow strong
         Data['T_AIRFLOW'] = pd.cut(Data['AIRFLOW'],3,labels=['low','normal','high'])

In [91]: #T_FREQUENCY is the how strong the frequency  of each device in the system
         Data['T_FREQUENCY'] = pd.cut(Data['FREQUENCY'],3,labels=['low','normal','high'])
```

Fig. no.26

In the end:-

we show the finale view of the data after done cleaning it like shown in the fig. No.27.

```
In [92]: #final shape for the dataset after the preprocessing on it
         Data
```

Out[92]:

	SIZE	FUEL	DISTANCE	DESIBEL	AIRFLOW	FREQUENCY	STATUS	T_SIZE	T_DISTANCE	T_DESIBEL	T_AIRFLOW	T_FREQUENCY
2	4	thinner	160.0	105.0	4.4	36.0	0	medium	long	high	low	normal
3	6	lpg	90.0	102.0	0.0	68.0	0	larg	Above_Average	high	low	high
4	1	gasoline	180.0	87.0	2.8	5.0	0	small	long	normal	low	low
5	2	thinner	170.0	76.0	0.0	1.0	0	small	long	low	low	low
6	2	kerosene	80.0	75.0	0.0	1.0	0	small	Above_Average	low	low	low
...	...	...	...	...	...	...	...	...	...	...	...	...
17440	1	thinner	80.0	106.0	11.2	46.0	1	small	Above_Average	high	normal	high
17441	2	thinner	50.0	82.0	3.2	2.0	0	small	Average	low	low	low
17442	5	kerosene	160.0	94.0	1.9	55.0	0	medium	long	normal	low	high
17444	1	kerosene	20.0	105.0	13.5	30.0	1	small	Average	high	high	normal
17445	1	gasoline	20.0	91.0	11.9	12.0	1	small	Average	normal	high	normal

17129 rows × 12 columns

fig. no.27

Part 2 : Modeling:-

To use the data for the used models we encode the categorical feature (“ FUEL ”) as shown in Fig. NO.28into numerical one and split the data into training set and test set.

We used function called “train_test_split ()” which here parameters are the chosen features , the label feature (“ STATUS ”) , how much the test set size and the random state.

```
In [567]: #encoding the fuel column from categorical data to numeric for the model to understand and puting it in the datafarme
          le = LabelEncoder()
          le.fit(Data["FUEL"])
          Data["FUEL"] = le.transform(Data["FUEL"])
          # dividing data into train and test
          X_train, X_test, y_train, y_test = train_test_split(Data[["SIZE","FUEL", "DISTANCE", "AIRFLOW", "FREQUENCY"]]
                                                              , Data["STATUS"], test_size=0.2, random_state=100)
```

Fig. NO.28

1st : KNN:-

KNN is a supervised model which use with label data. In this model it depend on k which parameter to help in the decision. So, first we choose k and make odd to facilitate the decision making with default mathematical calculation which is Euclidean . Then we start to fit our model by giving it the input and output of the training set. Secondly, we predict the output of the test set then finally we calculate the accuracy by compare the predicted results with real output of the test set as shown in Fig. NO.29.

```
In [568]: #fit training data with n_neighbors = 1001
          KNN = KNeighborsClassifier(n_neighbors = 1001)
          KNN.fit(X_train, y_train)

Out[568]: KNeighborsClassifier(n_neighbors=1001)

In [569]: #predict training data with n_neighbors = 1001
          p_KNN = KNN.predict(X_test)
          p_KNN

Out[569]: array([0, 1, 1, ..., 0, 0, 1], dtype=int64)

In [570]: #calculate Accuracy after training
          Accuracy_KNN = accuracy_score(y_test, p_KNN)
          Accuracy_KNN

Out[570]: 0.8534734384121424
```

Fig. NO.29

Then , we calculate the confusion matrix which consist of (TP,TN,FP,FN) as shown in Fig. NO.30.

```
In [571]: #Get the confusion matrix of KNN
          CM_KNN = confusion_matrix(y_test, p_KNN)
          print(CM_KNN)

          [[1381  346]
           [ 156 1543]]
```

Fig. NO.30

Then here we plot the confusion matrix with predicted values as x-axis and actual values as y-axis as shown in Fig.NO.31.

```
In [571]: #Get the confusion matrix of KNN
          CM_KNN = confusion_matrix(y_test, p_KNN)
          print(CM_KNN)

          [[1381  346]
           [ 156 1543]]

In [572]: #we use heatmap() method to return the matplotlib axes that can be stored in a variable.
          axis = sns.heatmap(CM_KNN, annot=True, cmap='Purples')

          #set the title of x-axis, y-axis
          axis.set_xlabel('Predicted Values')
          axis.set_ylabel('Actual Values ');

          #Tick labels that used to denote the datapoints on the axes and it must be in ascending order.
          axis.xaxis.set_ticklabels(['False','True'])
          axis.yaxis.set_ticklabels(['False','True'])

          #ploting the confusion matrix of KNN
          plt.show()
```

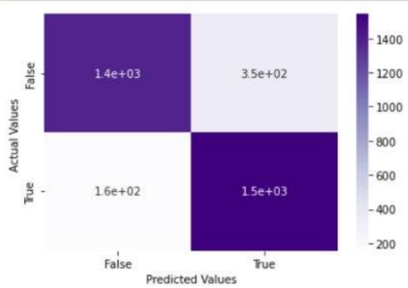


Fig. NO31.

In shown fig. NO.32 we use the KNN model but we change the way of calculation from Euclidean to Manhattan which increase bit the accuracy and we use same steps as before.

```
In [573]: #fitting test data using manhattan with n_neighbors = 1001
          knn_manhattan = KNeighborsClassifier(n_neighbors=1001, metric='manhattan')
          knn_manhattan.fit(X_train, y_train)
          p_manhattan= knn_manhattan.predict(X_test)
          p_manhattan

Out[573]: array([0, 1, 1, ..., 0, 0, 1], dtype=int64)

In [574]: #calculate Accuracy after test
          Accuracy_manhattan = accuracy_score(y_test, p_manhattan)
          Accuracy_manhattan

Out[574]: 0.8590192644483362
```

Fig. NO.32

2nd : decision tree:-

we used a decision tree to classify the dataset which we give it feature to get nodes and leaves. We only use numerical feature in this model because it became easier to the model that it can understand the features.

Firstly, we pass x_train (features) and y-train(output) by using DecisionTreeClassifier() and start to fit the model on the training set of data as shown Fig. NO.33

```
In [290]: #making the decision tree model and fitting it to the data
          clf = DecisionTreeClassifier(random_state=0)
          clf.fit(X_train,y_train)

Out[290]: DecisionTreeClassifier(random_state=0)
```

Fig. NO.33

To predict the output (status)if 1 refers to the fire happens else 0 refers to no fire happens we predict that to x-test subset of data as shown in the below fig.

```
In [575]: #making the decision tree model and fitting it to the data
          #predict responce for my test data
          clf = DecisionTreeClassifier(criterion = "gini", random_state = 100, max_depth=3, min_samples_leaf=5)
          clf.fit(X_train,y_train)
          #to predict the output of x)test subset
          P_DT = clf.predict(X_test)
          P_DT

Out[575]: array([0, 1, 0, ..., 0, 1, 1], dtype=int64)
```

calculate accuracy by score(). we pass for it actual output and predicted output As shown in Fig. NO.34

```
In [576]: #calculate Accuracy after test
          Accuracy_DecisionTree = accuracy_score(y_test, P_DT)
          Accuracy_DecisionTree

Out[576]: 0.8523058960887332
```

In Fig.NO.35 we plot the confusion matrix .

```
In [578]: #we use heatmap() method to return the matplotlib axes that can be stored in a variable.
axis = sns.heatmap(CM_DT, annot=True, cmap='ocean')

#set the title of x-axis, y-axis
axis.set_xlabel('Predicted Values')
axis.set_ylabel('Actual Values ');

#Tick labels that used to denote the datapoints on the axes and it must be in ascending order.
axis.xaxis.set_ticklabels(['False','True'])
axis.yaxis.set_ticklabels(['False','True'])

#ploting the confusion matrix of decision tree
plt.show()
```

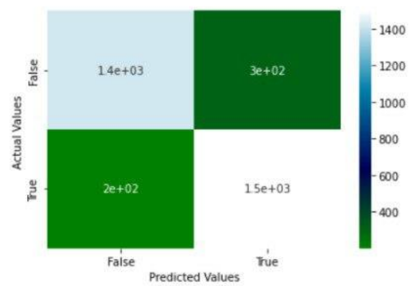


Fig. NO.35

3rd : Naïve bayes:-

Feature scaling is a method used to normalize the range of independent variables or features of data. so we used standardization as (StandardScaler).

the function (fit_transform()) is used for performs fit and transform on the input data at a single time and converts the data points.

If we use fit and transform separate when we need both then it will decrease the efficiency of the model so we use fit_transform() which will do both the work as shown in Fig. NO.36.

```
In [293]: #Feature Scaling
Sd_S = StandardScaler()
X_train = Sd_S.fit_transform(X_train)
X_test = Sd_S.transform(X_test)
```

Fig. NO.36

Naïve Bayes algorithm is a supervised learning algorithm,

used for solving classification problems.

It is mainly used in text classification that includes a high-dimensional training dataset as our used data.

GaussianNB implements the Gaussian Naive Bayes algorithm for classification and the data is entered as the parameter as shown in Fig. NO.37.

```
In [294]: #Training the Naive Bayes model on the Training set
          class_model = GaussianNB()
          class_model.fit(X_train, y_train)

Out[294]: GaussianNB()
```

Fig. NO.37

we used the function (.predict()) on the test set to predict the results(output). the function(accuracy_score()) is a function used to calculate the accuracy of confusion matrix as shown in Fig. NO.38 and its used to define the performance of a classification algorithm and its result was about 87% and its nearly a good accuracy but we should seek for better accuracy.

```
In [295]: #Predicting the Test set results
          P_naive = class_model.predict(X_test)

In [296]: #Calculate accuracy in Naive BayesMaking the Confusion Matrix
          accuracy_naive = accuracy_score(y_test,P_naive)
          accuracy_naive

Out[296]: 0.8718622300058377
```

Fig. NO.38

As shown in Fig. NO.39 we calculate and print the confusion matrix to show the performance of classification as it was illustrated before.

```
In [297]: #Making the Confusion Matrix
          CM_naive = confusion_matrix(y_test, P_naive)
          print(CM_naive)

          [[1558  163]
           [ 276 1429]]
```

Fig. NO.39

For fig. NO.40 we plot the confusion matrix .

```
In [298]: #we use heatmap() method to return the matplotlib axes that can be stored in a variable.
axis = sns.heatmap(CM_naive, annot=True, cmap='ocean')

#set the title of x-axis, y-axis
axis.set_xlabel('Predicted Values')
axis.set_ylabel('Actual Values ');

#Tick labels that used to denote the datapoints on the axes and it must be in ascending order.
axis.xaxis.set_ticklabels(['False','True'])
axis.yaxis.set_ticklabels(['False','True'])

#ploting the confusion matrix
plt.show()
```

Fig. NO.40

4th : the K-Means:

As we used k-means which unsupervised model we change the used data to unlabeled data and make the same steps to clean data with additional steps like what we did in solving the noise in the data.

the libraries:

```
In [1]: #import needed libraries and take an object from them
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
```

loading the data:

In [43]:
```
#load the data
Data=pd.read_csv("Ecommerce_data.csv")
Data.head()
```

Out[43]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	536365	71053	WHITE METAL LANTERN	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	12/1/2010 8:26	3.39	17850.0	United Kingdom

information about data:

removing the missing values:

In [44]:
```
#print all information about the Data
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 541909 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype  
---  ------       --------------   -----  
 0   InvoiceNo    541909 non-null  object 
 1   StockCode    541909 non-null  object 
 2   Description  540455 non-null  object 
 3   Quantity     541909 non-null  int64  
 4   InvoiceDate  541909 non-null  object 
 5   UnitPrice    541909 non-null  float64
 6   CustomerID   406829 non-null  float64
 7   Country      541909 non-null  object 
dtypes: float64(2), int64(1), object(5)
memory usage: 33.1+ MB
```

In [45]:
```
#getting the number of the missed values in the data using the following function
Data.isnull().sum().sort_values(ascending=False)
```

Out[45]:
```
CustomerID     135080
Description      1454
InvoiceNo           0
StockCode           0
Quantity            0
InvoiceDate         0
UnitPrice           0
Country             0
dtype: int64
```

check on the data:

```
In [46]: #we handle the problem of missing data by removind the row which contain the missing values
         #using dropna() function we drop the missing valus's rows
         #then checking whether there are missing values or not again to make sure that we handle the problem of missing values
         Data=Data.dropna()
         Data.isnull().sum().sort_values(ascending=False)

Out[46]: InvoiceNo      0
         StockCode      0
         Description    0
         Quantity       0
         InvoiceDate    0
         UnitPrice      0
         CustomerID     0
         Country        0
         dtype: int64
```

```
In [47]: #print information about the Data after remove missing values
         Data.info()

         <class 'pandas.core.frame.DataFrame'>
         Int64Index: 406829 entries, 0 to 541908
         Data columns (total 8 columns):
          #   Column       Non-Null Count   Dtype
         ---  ------       --------------   -----
          0   InvoiceNo    406829 non-null  object
          1   StockCode    406829 non-null  object
          2   Description  406829 non-null  object
          3   Quantity     406829 non-null  int64
          4   InvoiceDate  406829 non-null  object
          5   UnitPrice    406829 non-null  float64
          6   CustomerID   406829 non-null  float64
          7   Country      406829 non-null  object
         dtypes: float64(2), int64(1), object(5)
         memory usage: 27.9+ MB
```

additional process in deal with the noise was return each feature to it's true type :

```
In [48]: #change type of column 'InvoiceDate' from object to datetime
         #make column 'Description' his string to lower case
         #change type of column 'CustomerID' from flout64 to object
         Data['InvoiceDate']=pd.to_datetime(Data.InvoiceDate, format='%m/%d/%Y %H:%M')
         Data['Description']=Data.Description.str.lower()
         Data['CustomerID']=Data.CustomerID.astype('object')
         Data.head()
```

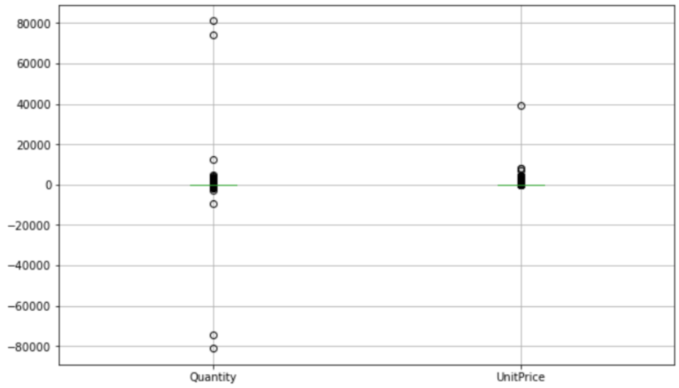
Out[48]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
0	536365	85123A	white hanging heart t-light holder	6	2010-12-01 08:26:00	2.55	17850.0	United Kingdom
1	536365	71053	white metal lantern	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
2	536365	84406B	cream cupid hearts coat hanger	8	2010-12-01 08:26:00	2.75	17850.0	United Kingdom
3	536365	84029G	knitted union flag hot water bottle	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom
4	536365	84029E	red woolly hottie white heart.	6	2010-12-01 08:26:00	3.39	17850.0	United Kingdom

use the boxplot to see the noise data:

In [49]:
```
#boxplot when data is noise
plt.figure(figsize = (10, 6))
Data.boxplot()
```

Out[49]: <AxesSubplot:>



---

the process to delete the outliers and checking on data each time after deletion:

In [50]:
```
#remove outliers from column 'Quantity'
#remove all data that bigger than max and smaller than min after we calculate max, min values
#by change all outlier values to null vales to drop them
for x in ['Quantity']:
    Q75,Q25 = np.percentile(Data.loc[:,x],[75,25])
    intr_qr = Q75-Q25

    max = Q75+(1.5*intr_qr)
    min = Q25-(1.5*intr_qr)

    Data.loc[Data[x] < min,x] = np.nan
    Data.loc[Data[x] > max,x] = np.nan
```

In [51]:
```
#sum all null vales in 'Quantity' column
Data.isnull().sum()
```

Out[51]:
```
InvoiceNo          0
StockCode          0
Description        0
Quantity       26682
InvoiceDate        0
UnitPrice          0
CustomerID         0
Country            0
dtype: int64
```

In [52]:
```
#drop all null values that we found in column 'Quantity'
Data = Data.dropna(axis = 0)
```

In [53]:
```python
#print information about the Data after remove null values that we found in 'Quantity' column
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 380147 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    380147 non-null  object
 1   StockCode    380147 non-null  object
 2   Description  380147 non-null  object
 3   Quantity     380147 non-null  float64
 4   InvoiceDate  380147 non-null  datetime64[ns]
 5   UnitPrice    380147 non-null  float64
 6   CustomerID   380147 non-null  object
 7   Country      380147 non-null  object
dtypes: datetime64[ns](1), float64(2), object(5)
memory usage: 26.1+ MB
```

In [54]:
```python
#remove outliers from column 'UnitPrice'
#remove all data that bigger than max and smaller than min after we calculate max, min values
#by change all outlier values to null vales to drop them
for y in ['UnitPrice']:
    Q75,Q25 = np.percentile(Data.loc[:,y],[75,25])
    intr_qr = Q75-Q25

    max = Q75+(1.5*intr_qr)
    min = Q25-(1.5*intr_qr)

    Data.loc[Data[y] < min,y] = np.nan
    Data.loc[Data[y] > max,y] = np.nan
```

In [55]:
```python
#sum all null vales in 'UnitPrice' column
Data.isnull().sum()
```

Out[55]:
```
InvoiceNo          0
StockCode          0
Description        0
Quantity           0
InvoiceDate        0
UnitPrice      35754
CustomerID         0
Country            0
dtype: int64
```

In [56]:
```python
#drop all null values that we found in column 'UnitPrice'
Data = Data.dropna(axis = 0)
```

In [57]:
```python
#sum all null vales in 'UnitPrice' column to make sure that all null values are dropped
Data.isnull().sum()
```

In [56]:
```python
#drop all null values that we found in column 'UnitPrice'
Data = Data.dropna(axis = 0)
```

In [57]:
```python
#sum all null vales in 'UnitPrice' column to make sure that all null values are dropped
Data.isnull().sum()
```


Out[57]:
```
InvoiceNo      0
StockCode      0
Description    0
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     0
Country        0
dtype: int64
```

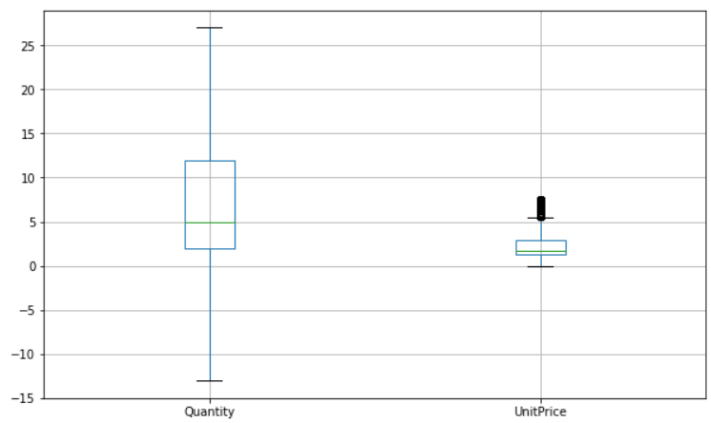
In [58]:
```python
#print information about the Data after remove null values that we found in 'UnitPrice' column
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 344393 entries, 0 to 541908
Data columns (total 8 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   InvoiceNo    344393 non-null  object
 1   StockCode    344393 non-null  object
 2   Description  344393 non-null  object
 3   Quantity     344393 non-null  float64
 4   InvoiceDate  344393 non-null  datetime64[ns]
 5   UnitPrice    344393 non-null  float64
 6   CustomerID   344393 non-null  object
 7   Country      344393 non-null  object
dtypes: datetime64[ns](1), float64(2), object(5)
```

boxplot after delete most of the outliers but this data need more dealing with noise with advanced ways that 's why there still noise in it:

In [59]:
```
#boxplot after cleaning the data from noise,null and remove outliers
plt.figure(figsize = (10, 6))
Data.boxplot()
```

Out[59]: <AxesSubplot:>



other needed libraries:

In [42]:
```
#import needed libraries and take an object from them
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
```

dealing with the duplication:

In [60]:
```
#show the duplicate
Data[Data.duplicated()].head()
```

Out[60]:

	InvoiceNo	StockCode	Description	Quantity	InvoiceDate	UnitPrice	CustomerID	Country
517	536409	21866	union jack flag luggage tag	1.0	2010-12-01 11:45:00	1.25	17908.0	United Kingdom
527	536409	22866	hand warmer scotty dog design	1.0	2010-12-01 11:45:00	2.10	17908.0	United Kingdom
537	536409	22900	set 2 tea towels i love london	1.0	2010-12-01 11:45:00	2.95	17908.0	United Kingdom
539	536409	22111	scottie dog hot water bottle	1.0	2010-12-01 11:45:00	4.95	17908.0	United Kingdom
555	536412	22327	round snack boxes set of 4 skulls	1.0	2010-12-01 11:49:00	2.95	17920.0	United Kingdom

In [61]:
```
#sum all duplicated data
Data.duplicated().sum()
```

Out[61]: 4940

In [62]:
```
#drop all duplicated data
Data.drop_duplicates(inplace=True)
```

In [63]:
```
#sum all duplicate data to make sure that there is no duplication
Data.duplicated().sum()
```

Out[63]: 0

dealing with irrelevant feature by deleting it and showing the data:

In [65]:
```
#this column don't related to our output(status)so we remove it
Data=Data.drop(columns="StockCode")
Data=Data.drop(columns="InvoiceNo")
Data=Data.drop(columns="InvoiceDate")
Data=Data.drop(columns="CustomerID")
Data
```

Out[65]:

	Description	Quantity	UnitPrice	Country
0	white hanging heart t-light holder	6.0	2.55	United Kingdom
1	white metal lantern	6.0	3.39	United Kingdom
2	cream cupid hearts coat hanger	8.0	2.75	United Kingdom
3	knitted union flag hot water bottle	6.0	3.39	United Kingdom
4	red woolly hottie white heart.	6.0	3.39	United Kingdom
...	...	...	...	...
541904	pack of 20 spaceboy napkins	12.0	0.85	France
541905	children's apron dolly girl	6.0	2.10	France
541906	childrens cutlery dolly girl	4.0	4.15	France
541907	childrens cutlery circus parade	4.0	4.15	France
541908	baking set 9 piece retrospot	3.0	4.95	France

339453 rows × 4 columns

---

calculate the correlation :

In [66]:
```
#create a matrix filled with correlation values for each two columns
Data.corr()
```

Out[66]:

	Quantity	UnitPrice
Quantity	1.000000	-0.344021
UnitPrice	-0.344021	1.000000

In [67]:
```
#create a matrix filled with correlation values for each two columns with absolute value
CorrelationMatrix = Data.corr().abs()
CorrelationMatrix
```

Out[67]:

	Quantity	UnitPrice
Quantity	1.000000	0.344021
UnitPrice	0.344021	1.000000

We find nothing to delete because no correlation equal or bigger than 0.8

In [68]:
```
#Selecting the Upper triangular matrix
up_tri = CorrelationMatrix.where(np.triu(np.ones(CorrelationMatrix.shape),k=1).astype(np.bool))
up_tri
```

```
C:\Users\MCC\AppData\Local\Temp/ipykernel_3248/2541172571.py:2: DeprecationWarning: `np.bool` is a deprecated alias for the bui
ltin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specifi
cally wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  up_tri = CorrelationMatrix.where(np.triu(np.ones(CorrelationMatrix.shape),k=1).astype(np.bool))
```

Out[68]:

	Quantity	UnitPrice
Quantity	NaN	0.344021
UnitPrice	NaN	NaN

In [69]:
```
#this method help us to know which columns with high correlation that we will drop
C_drop = [column for column in up_tri.columns if any(up_tri[column] >= 0.8)]
print(C_drop)
```

```
[]
```

In [70]:
```
#their is no corr >= 0.8 between columns so we will not drop columns
Data.head()
```

Out[70]:

	Description	Quantity	UnitPrice	Country
0	white hanging heart t-light holder	6.0	2.55	United Kingdom
1	white metal lantern	6.0	3.39	United Kingdom
2	cream cupid hearts coat hanger	8.0	2.75	United Kingdom
3	knitted union flag hot water bottle	6.0	3.39	United Kingdom
4	red woolly hottie white heart.	6.0	3.39	United Kingdom

Apply discretization :

In [71]:
```
#by using cut () function we do the discretization to each numerical attribute
#T_Quantity is the types of the Quantity which divided into 3 parts
Data['T_Quantity'] = pd.cut(Data['Quantity'],3,labels=['small','medium','larg'])
```

In [72]:
```
#T_UnitPrice is the types of the Quantity which divided into 3 parts
Data['T_UnitPrice'] = pd.cut(Data['UnitPrice'],3,labels=['low','normal','great'])
```

In [73]:
```
#final shape for the dataset after the preprocessing on it
Data
```

Out[73]:

	Description	Quantity	UnitPrice	Country	T_Quantity	T_UnitPrice
0	white hanging heart t-light holder	6.0	2.55	United Kingdom	medium	normal
1	white metal lantern	6.0	3.39	United Kingdom	medium	normal
2	cream cupid hearts coat hanger	8.0	2.75	United Kingdom	medium	normal
3	knitted union flag hot water bottle	6.0	3.39	United Kingdom	medium	normal
4	red woolly hottie white heart.	6.0	3.39	United Kingdom	medium	normal
...	...	...	...	...	...	...
541904	pack of 20 spaceboy napkins	12.0	0.85	France	medium	low
541905	children's apron dolly girl	6.0	2.10	France	medium	low
541906	childrens cutlery dolly girl	4.0	4.15	France	medium	normal
541907	childrens cutlery circus parade	4.0	4.15	France	medium	normal
541908	baking set 9 piece retrospot	3.0	4.95	France	medium	normal

339453 rows × 6 columns

Now we can apply the K-Means algorithm:

we incode the categorical features into numarical so we can use it.

```
In [74]: #encoding the Description,Country column from categorical data to numeric for the model to understand and put it in the datafarme
         le = LabelEncoder()
         le.fit(Data["Description"])
         Data["Description"] = le.transform(Data["Description"])
         le.fit(Data["Country"])
         Data["Country"] = le.transform(Data["Country"])
```

First : choose the features we want from the data , then creating K-Means classifier ,setting the number of cluster we want (centroids) we choose 3 , and initializing this centroid randomly.

```
In [75]: #select feature i want from Data
         #Create KMeans Classifier, set number of clusters or centroids, random number generation for centroid initialization.
         #Use an int in (random_state) to make the randomness deterministic.
         #we use Kmeans.fit(Data) to train our model in our data set that need to be Clustered.
         Data = Data[['Description', 'Quantity', 'UnitPrice', 'Country']]
         kmeans = KMeans(n_clusters=3, random_state=0).fit(Data)
         kmeans

Out[75]: KMeans(n_clusters=3, random_state=0)
```

Secondary : Using function called “kmeans.predict()” the data is clustered into three cluster after 300 iteration (by default) which are (0 , 1 or 2).

Finally , we print the final centroid we have reach

```
In [76]: #Predict the closest cluster each sample in X belongs to
         P_KM = kmeans.predict(Data)
         P_KM

Out[76]: array([0, 0, 1, ..., 1, 1, 1])
```

```
In [77]: #Coordinates of cluster centers
         Centroid = kmeans.cluster_centers_
         print(Centroid)

         [[2.92780934e+03 7.49903988e+00 2.17490696e+00 3.27464738e+01]
          [5.53764356e+02 7.12732066e+00 2.33352365e+00 3.29737583e+01]
          [1.73061742e+03 7.35655950e+00 2.13618892e+00 3.33643332e+01]]
```

Here we print the name of each cluster.

```
In [78]: #Getting labels of each point using Kmeans.labels_ method
         #Getting unique labels using np.unique(Labels)
         Labels = kmeans.labels_
         unique_Labels = np.unique(Labels)
         unique_Labels

Out[78]: array([0, 1, 2])
```