Experiment No.3
Perform data pre-processing
Date of Performance:
Date of Submission:

Aim: To implement data preprocessing Algorithm

Objective:-Develop a program to implement data preprocessing algorithm

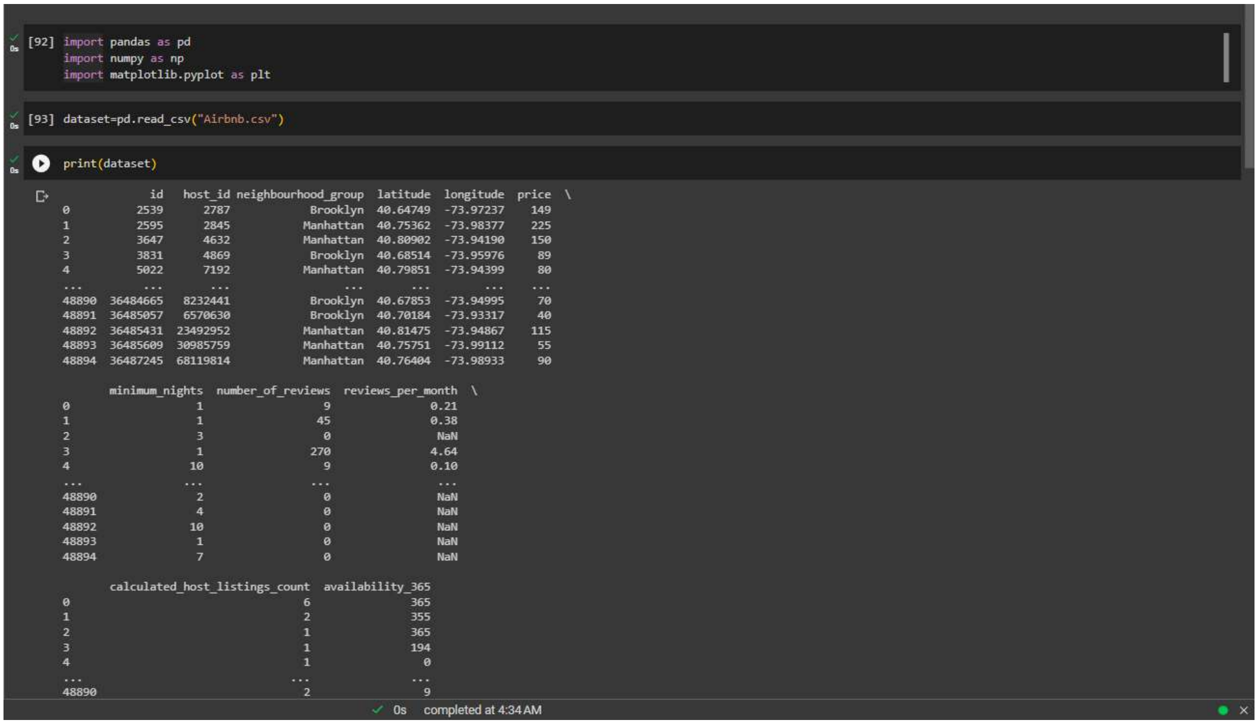
Theory: Why preprocess the data? Because data in the real world is dirty, incomplete and noisy. Incomplete in lacking attributes values and lacking attributes of interest or containing only aggregate value noisy in terms of containing errors or outliers and inconsistent containing discrepancies in names or codes. Now the question arises why is the data dirty? Because incomplete data may come from —not applicable‖ data value when data has to be collected and the major issue is a different consideration between the times when the data was analyzed and human hardware and software issues are common. Noisy data may come from the when a human enters the wrong value at the time of data entry as Nobody is perfect. Errors in transmission of data and instruments that collect the faulty data. Inconsistent data may come from the different data sources. Duplicates records also need data cleaning.

Why data preprocessing is important? Data is not clean, Duplicity of data and the no quality data and the most important is no quality result so data preprocessing is important. Quality decisions must be based on the quality data. Data warehouse needs consistent integration of quality data. By the processing of data, data quality can be measures in term of accuracy, completeness, consistency, timeliness, believability, interpretability. There are three methods to handle the noisy data.

The different pre-processing steps that can be applied are:

1) Filling up the missing values
2) Removing duplicate data
3) Handling noisy data
4) Handling outliers
5) Scaling of data
6) Encoding of text or categorical values

Code and output:



```
[92] import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
```

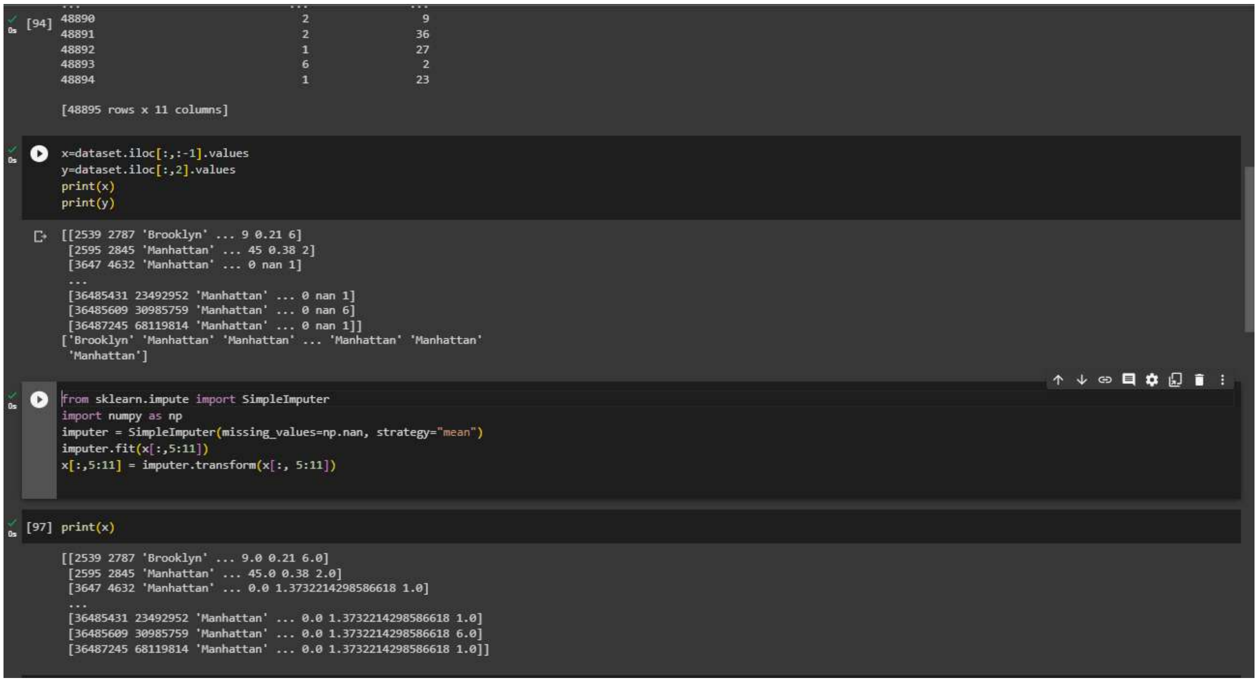
```
[93] dataset=pd.read_csv("Airbnb.csv")
```

```
print(dataset)
```

```
             id   host_id neighbourhood_group  latitude  longitude  price  \
0          2539      2787            Brooklyn  40.64749  -73.97237    149
1          2595      2845           Manhattan  40.75362  -73.98377    225
2          3647      4632           Manhattan  40.80902  -73.94190    150
3          3831      4869            Brooklyn  40.68514  -73.95976     89
4          5022      7192           Manhattan  40.79851  -73.94399     80
...         ...       ...                 ...       ...        ...    ...
48890  36484665   8232441            Brooklyn  40.67853  -73.94995     70
48891  36485057   6570630            Brooklyn  40.70184  -73.93317     40
48892  36485431  23492952           Manhattan  40.81475  -73.94867    115
48893  36485609  30985759           Manhattan  40.75751  -73.99112     55
48894  36487245  68119814           Manhattan  40.76404  -73.98933     90

       minimum_nights  number_of_reviews  reviews_per_month  \
0                   1                  9               0.21
1                   1                 45               0.38
2                   3                  0                NaN
3                   1                270               4.64
4                  10                  9               0.10
...               ...                ...                ...
48890               2                  0                NaN
48891               4                  0                NaN
48892              10                  0                NaN
48893               1                  0                NaN
48894               7                  0                NaN

       calculated_host_listings_count  availability_365
0                                   6               365
1                                   2               355
2                                   1               365
3                                   1               194
4                                   1                 0
...                               ...               ...
48890                               2                 9
```



```
[94] ...                               ...               ...
     48890                               2                 9
     48891                               2                36
     48892                               1                27
     48893                               6                 2
     48894                               1                23

     [48895 rows x 11 columns]
```

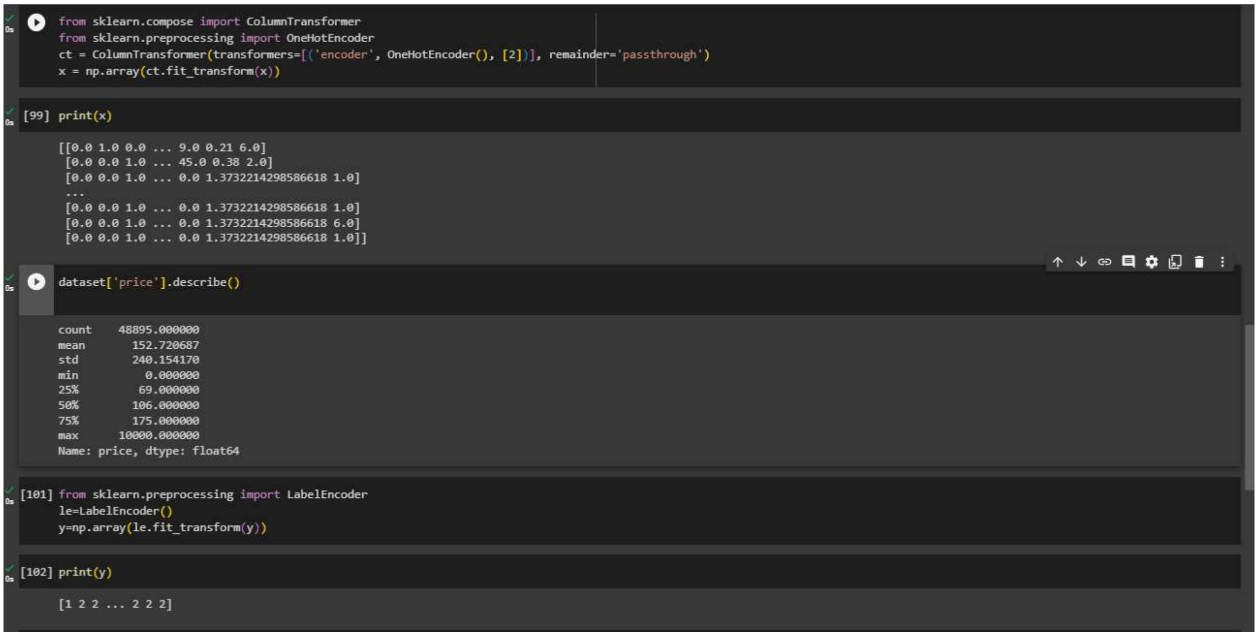
```
x=dataset.iloc[:,:-1].values
y=dataset.iloc[:,2].values
print(x)
print(y)
```

```
[[2539 2787 'Brooklyn' ... 9 0.21 6]
 [2595 2845 'Manhattan' ... 45 0.38 2]
 [3647 4632 'Manhattan' ... 0 nan 1]
 ...
 [36485431 23492952 'Manhattan' ... 0 nan 1]
 [36485609 30985759 'Manhattan' ... 0 nan 6]
 [36487245 68119814 'Manhattan' ... 0 nan 1]]
['Brooklyn' 'Manhattan' 'Manhattan' ... 'Manhattan' 'Manhattan'
 'Manhattan']
```

```
from sklearn.impute import SimpleImputer
import numpy as np
imputer = SimpleImputer(missing_values=np.nan, strategy="mean")
imputer.fit(x[:,5:11])
x[:,5:11] = imputer.transform(x[:, 5:11])
```

```
[97] print(x)
```

```
[[2539 2787 'Brooklyn' ... 9.0 0.21 6.0]
 [2595 2845 'Manhattan' ... 45.0 0.38 2.0]
 [3647 4632 'Manhattan' ... 0.0 1.3732214298586618 1.0]
 ...
 [36485431 23492952 'Manhattan' ... 0.0 1.3732214298586618 1.0]
 [36485609 30985759 'Manhattan' ... 0.0 1.3732214298586618 6.0]
 [36487245 68119814 'Manhattan' ... 0.0 1.3732214298586618 1.0]]
```



```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [2])], remainder='passthrough')
x = np.array(ct.fit_transform(x))
```

```
[99] print(x)

[[0.0 1.0 0.0 ... 9.0 0.21 6.0]
 [0.0 0.0 1.0 ... 45.0 0.38 2.0]
 [0.0 0.0 1.0 ... 0.0 1.3732214298586618 1.0]
 ...
 [0.0 0.0 1.0 ... 0.0 1.3732214298586618 1.0]
 [0.0 0.0 1.0 ... 0.0 1.3732214298586618 6.0]
 [0.0 0.0 1.0 ... 0.0 1.3732214298586618 1.0]]
```

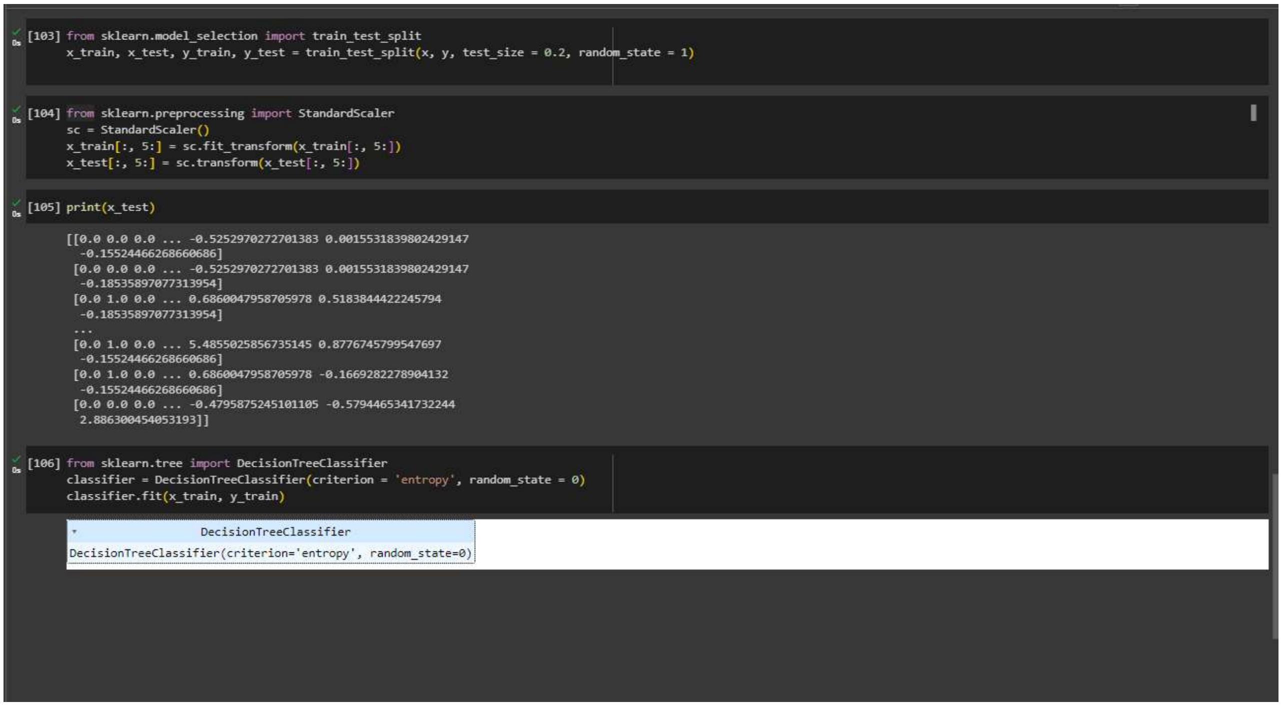
```
dataset['price'].describe()

count    48895.000000
mean       152.720687
std        240.154170
min          0.000000
25%         69.000000
50%        106.000000
75%        175.000000
max      10000.000000
Name: price, dtype: float64
```

```
[101] from sklearn.preprocessing import LabelEncoder
      le=LabelEncoder()
      y=np.array(le.fit_transform(y))
```

```
[102] print(y)

      [1 2 2 ... 2 2 2]
```



```
[103] from sklearn.model_selection import train_test_split
      x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 1)
```

```
[104] from sklearn.preprocessing import StandardScaler
      sc = StandardScaler()
      x_train[:, 5:] = sc.fit_transform(x_train[:, 5:])
      x_test[:, 5:] = sc.transform(x_test[:, 5:])
```

```
[105] print(x_test)

[[0.0 0.0 0.0 ... -0.5252970272701383 0.0015531839802429147
  -0.15524466268660686]
 [0.0 0.0 0.0 ... -0.5252970272701383 0.0015531839802429147
  -0.18535897077313954]
 [0.0 1.0 0.0 ... 0.6860047958705978 0.5183844422245794
  -0.18535897077313954]
 ...
 [0.0 1.0 0.0 ... 5.4855025856735145 0.8776745799547697
  -0.15524466268660686]
 [0.0 1.0 0.0 ... 0.6860047958705978 -0.1669282278904132
  -0.15524466268660686]
 [0.0 0.0 0.0 ... -0.4795875245101105 -0.5794465341732244
  2.886300454053193]]
```

```
[106] from sklearn.tree import DecisionTreeClassifier
      classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
      classifier.fit(x_train, y_train)

DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

Conclusion: Pre-processing is crucial in data analysis and machine learning. It involves cleaning, transforming, and organizing data to make it suitable for analysis. Without proper pre-processing, data can be noisy, inconsistent, or irrelevant, leading to inaccurate results. Pre-processing addresses issues like missing values, outliers, and scaling, enhancing the quality of the dataset. Skipping pre-processing can result in biased models, reduced accuracy, and unreliable insights.