Name:- Aniket kumar

S.No:- 30

Roll no:- PF45

PRN no:- 1032171203

AML LAB ASSIGNMENT -01

Aim :- To study Data Collection, Data Preparation, Data Handling and perform Exploratory Data Analysis.

Theory :-

Data Collection :-

collection of data is the most fundamental step in any data science process. There are many different ways of collecting data.

1. Building dataset from scratch
2. Using government websites.
3. Accessing private datasets.

Various types of data :-

1. Numerical
2. Discrete Data
3. continuous Data
4. categorical Data
5. Ordinal Data

Operations to be performed on dataset :-

Steps in Preprocessing of Data.

1) Importing Python Modules / Libraries.

2) Importing data

3) Displaying data

4) Creating the Independent and Dependent variables.

5) Replacing missing value with meaningful value

6) Encoding categorical data.

7) Spliting the data into training and test set

8) Doing feature scaling on data.

9) Use any 3-4 graphs / plots.

FAQs:-

1] List two common libraries for data manipulation. Give an example for each library.

Ans- Pandas:- It is a perfect tool for data wrangling or munging. It is designed for quick and easy data manipulation, reading, aggregation and visualizati-on.

data = pd.read_csv("filename.csv")

This statement reads a csv file into a pandas dataframe.

Tensor Flow:- It is an AI library that helps developers to create large-scale neural networks with many layers using data flow graphs also facilitating the building of Deep Learning models. Tensorflow usually works on top of pandas.

dataset = tf.data.Dataset.from_tensor_slices
([8,3,0,8,2,1]).

This line creates a 1D tensor.

2] Give an example on how ordinal data is handled in a Machine Learning algorithm.

Ans:- Ordinal data is categorical data which is ordered. To use that feature in a Machine Learning pipeline, it is usually encoded in some form to maintain coherence in the dataset. Some of the basic encoders are ordinal encoding, binary encoding, or one Hot encoding. If we use ordinal encoding each category value is assigned to an integer value and the original order is maintained.

For eg:- while grading 'A' is 1, 'B' is 2, 'C' is 3 and so on. The integer values have a natural ordered relationship between each other and machine learning algorithm may be able to understand and harness this relationship.

3] Can one hot encoding be used for continuous data. If yes, give an example.

Ans:- No, one hot encoding can only be used on a finite set of values, hence it is usually used with categorical data. In the case of continuous data there are an infinite number of values and for each value one will need a binary variable. This will led to an encoding using an infinite number of binary variable which isn't feasible. Hence One hot encoding cannot be used with continous data.

4]. Why is it necessary to encode strings?

Ans:- Machine tends to only understand numbers, for example, machine won't be able to understand red, blue, green colour if we provide them as strings. Machine convert the string provided into a numeric representa-tion that will help machine to perform analysis. So, in order to reduce the complexity of data it is very important that the input string data is converted using a encoding technique that will in turn help to reduce the complexity and time require-d to train a particular machine learning model.

In [18]:
```python
#Name:- Aniket Kumar
#PRN:- 1032171203
#Roll No:- PF45
#S.No:-30
#Lab Assignment 01
```

In [1]:
```python
# Importing Python Modules
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```python
# Importing data and display
data=pd.read_csv(r"C:\Users\Aniket\Desktop\Aml\1\archive\winequality-red.csv")
data
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

In [3]:
```python
# Displaying Data
data.head(5)
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

In [4]:
```python
data.tail(5)
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

In [5]:
```python
# displaying data information
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
fixed acidity           1599 non-null float64
volatile acidity        1599 non-null float64
citric acid             1599 non-null float64
residual sugar          1599 non-null float64
chlorides               1599 non-null float64
free sulfur dioxide     1599 non-null float64
total sulfur dioxide    1599 non-null float64
density                 1599 non-null float64
pH                      1599 non-null float64
sulphates               1599 non-null float64
alcohol                 1599 non-null float64
quality                 1599 non-null int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [6]:
```python
data.shape
```

Out[6]:
```
(1599, 12)
```

In [7]:
```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
fixed acidity           1599 non-null float64
volatile acidity        1599 non-null float64
citric acid             1599 non-null float64
residual sugar          1599 non-null float64
chlorides               1599 non-null float64
free sulfur dioxide     1599 non-null float64
total sulfur dioxide    1599 non-null float64
density                 1599 non-null float64
pH                      1599 non-null float64
sulphates               1599 non-null float64
alcohol                 1599 non-null float64
quality                 1599 non-null int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [8]:
```python
# creating array of independent variable
X=data.iloc[:,:-1].values
X
```

Out[8]:
```
array([[ 7.4  ,  0.7  ,  0.   , ...,  3.51 ,  0.56 ,  9.4  ],
       [ 7.8  ,  0.88 ,  0.   , ...,  3.2  ,  0.68 ,  9.8  ],
       [ 7.8  ,  0.76 ,  0.04 , ...,  3.26 ,  0.65 ,  9.8  ],
       ...,
       [ 6.3  ,  0.51 ,  0.13 , ...,  3.42 ,  0.75 , 11.   ],
       [ 5.9  ,  0.645,  0.12 , ...,  3.57 ,  0.71 , 10.2  ],
       [ 6.   ,  0.31 ,  0.47 , ...,  3.39 ,  0.66 , 11.   ]])
```

In [9]:
```python
# creating array of dependent variable
y=data.iloc[:,-1].values # or we can use y=data.iloc[:,3].values
y
```

Out[9]:
```
array([5, 5, 5, ..., 6, 5, 6], dtype=int64)
```

In [10]:
```python
# replacing missing values with mean values of the columns
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
imputer.fit(X)
X=imputer.transform(X)
X
```

Out[10]:
```
array([[ 7.4  ,  0.7  ,  0.   , ...,  3.51 ,  0.56 ,  9.4  ],
       [ 7.8  ,  0.88 ,  0.   , ...,  3.2  ,  0.68 ,  9.8  ],
       [ 7.8  ,  0.76 ,  0.04 , ...,  3.26 ,  0.65 ,  9.8  ],
       ...,
       [ 6.3  ,  0.51 ,  0.13 , ...,  3.42 ,  0.75 , 11.   ],
       [ 5.9  ,  0.645,  0.12 , ...,  3.57 ,  0.71 , 10.2  ],
       [ 6.   ,  0.31 ,  0.47 , ...,  3.39 ,  0.66 , 11.   ]])
```

In [11]:
```python
# Splitting data into training and test data
#from sklearn.cross_validation import train_test_split

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=0)
```

In [12]:
```python
# Doing Feature scaling on data
from sklearn.preprocessing import StandardScaler
sc_X=StandardScaler()
X_train=sc_X.fit_transform(X_train)
X_test=sc_X.fit_transform(X_test)
```

In [13]:
```python
# Checking train and test
X_train
```

Out[13]:
```
array([[ 0.90103398,  0.05480282,  0.9094138 , ...,  0.52091013,
        -0.22358408, -0.95579434],
       [ 1.41998736, -1.47967601,  0.9094138 , ..., -1.16841553,
        -0.68130963, -0.76727388],
       [ 0.90103398, -0.98645067,  1.4208416 , ..., -0.3237527 ,
         0.74908272,  0.17532846],
       ...,
       [-0.25219574,  0.21921126,  0.19341488, ..., -0.12883051,
         0.17692578, -0.86153411],
       [ 2.68854005, -0.32881689,  1.11398492, ..., -0.06385645,
         0.11971008,  2.15479335],
       [ 0.84337249,  2.46612668,  0.24455766, ..., -0.38872677,
        -1.0246038 , -0.95579434]])
```

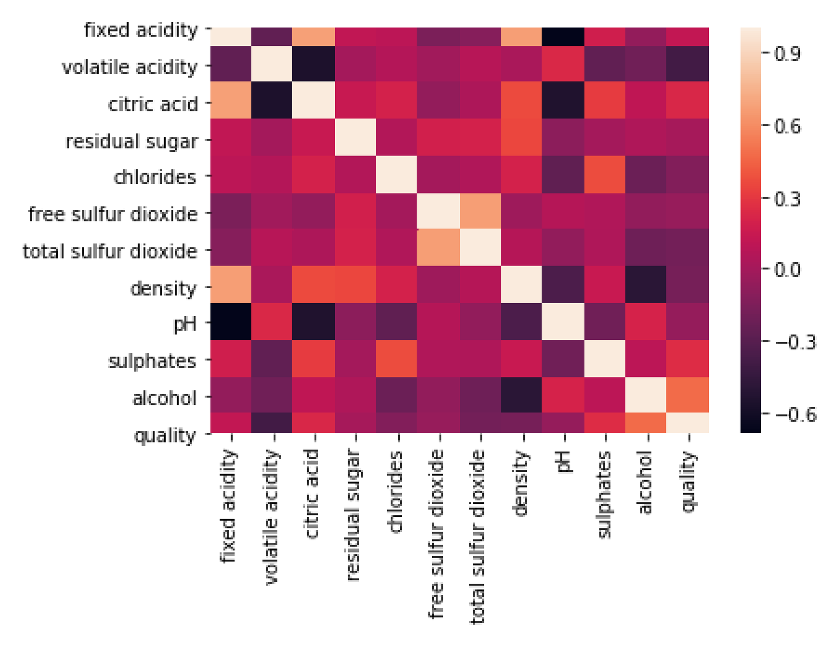
In [14]:
```python
X_test
```

Out[14]:
```
array([[ 1.44653617, -0.29922497,  0.8557103 , ..., -0.93785052,
         0.72205872,  0.31519731],
       [-0.08433993,  1.83333604, -1.39006001, ...,  0.28055357,
        -0.85116573, -0.79369656],
       [ 0.48265122, -1.39597062,  0.33343814, ..., -0.36071174,
         1.26926723,  1.14686771],
       ...,
       [ 0.48265122, -1.09131905,  0.80348308, ..., -0.87372399,
        -0.7143636 ,  0.86964424],
       [ 0.48265122,  1.49821931, -1.18115114, ..., -0.16833214,
        -0.78276467, -0.70128874],
       [-0.02764082, -1.21317968,  0.80348308, ..., -0.29658521,
        -0.7143636 ,  1.70131464]])
```

In [15]:
```python
import seaborn as sns
```

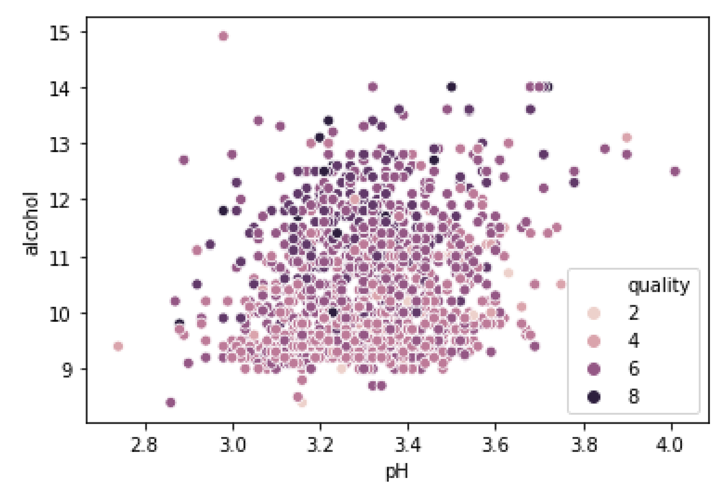
In [16]:
```python
sns.heatmap(data.corr())
```

Out[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a826270e48>
```



In [17]:
```python
sns.scatterplot(data.pH,data.alcohol,hue=data.quality)
```

Out[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a826417308>
```



In [ ]: