Machine Learning: Assignment No. 1

Question No. 2 (c): Ordinal Regression & Linear Regression on Wine Dataset

---

Student Name: Mayuresh Rajesh Dindorkar

Roll No. : CS23MTECH14007

Student Name: Sanyam Kaul

Roll No. : CS23MTECH14011

Index:

In [8]:
```
# Installing required packages
# 'mord' Package is imported for ordinal regression
!pip install mord
!pip install numpy
!pip install pandas
!pip install matplotlib
```

In [1]:
```
# Importing required packages
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import seaborn as sns
# LogisticIT - Immediate threshold, LogisticAT - All threshold
from mord import LogisticIT
%matplotlib inline
```

Step 1: Reading the dataset from csv file

In [2]:
```
red_wine_dataset = pd.read_csv('Wine_Dataset/winequality-red.csv',sep=';')
print('Red wine dataset dimensions (rows,columns):',red_wine_dataset.shape)
white_wine_dataset = pd.read_csv('Wine_Dataset/winequality-white.csv',sep=';')
print('White wine dataset dimensions (rows,columns):',white_wine_dataset.shape)
```

```
Red wine dataset dimensions (rows,columns): (1599, 12)
White wine dataset dimensions (rows,columns): (4898, 12)
```

In [3]:
```
print('Printing Red wine samples :-')
red_wine_dataset.head()
```

```
Printing Red wine samples :-
```

Out[3]:

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

In [4]:
```
print('Printing White wine samples :-')
white_wine_dataset.head()
```

```
Printing White wine samples :-
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	qua
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	

Step 2: Data Preprocessing

Combining both 'Red wine' and 'White wine' datasets

In [5]:
```
red_white_dataset = pd.concat([white_wine_dataset])
print('Red-White wine combined dataset dimensions (rows,columns):',red_white_dataset.s
```

```
Red-White wine combined dataset dimensions (rows,columns): (4898, 12)
```

In [6]:
```
# Analysing the data: All features have real value
red_white_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         4898 non-null   float64
 1   volatile acidity      4898 non-null   float64
 2   citric acid           4898 non-null   float64
 3   residual sugar        4898 non-null   float64
 4   chlorides             4898 non-null   float64
 5   free sulfur dioxide   4898 non-null   float64
 6   total sulfur dioxide  4898 non-null   float64
 7   density               4898 non-null   float64
 8   pH                    4898 non-null   float64
 9   sulphates             4898 non-null   float64
 10  alcohol               4898 non-null   float64
 11  quality               4898 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
```

In [7]:
```
red_white_dataset.describe().T
```

Out[7]:

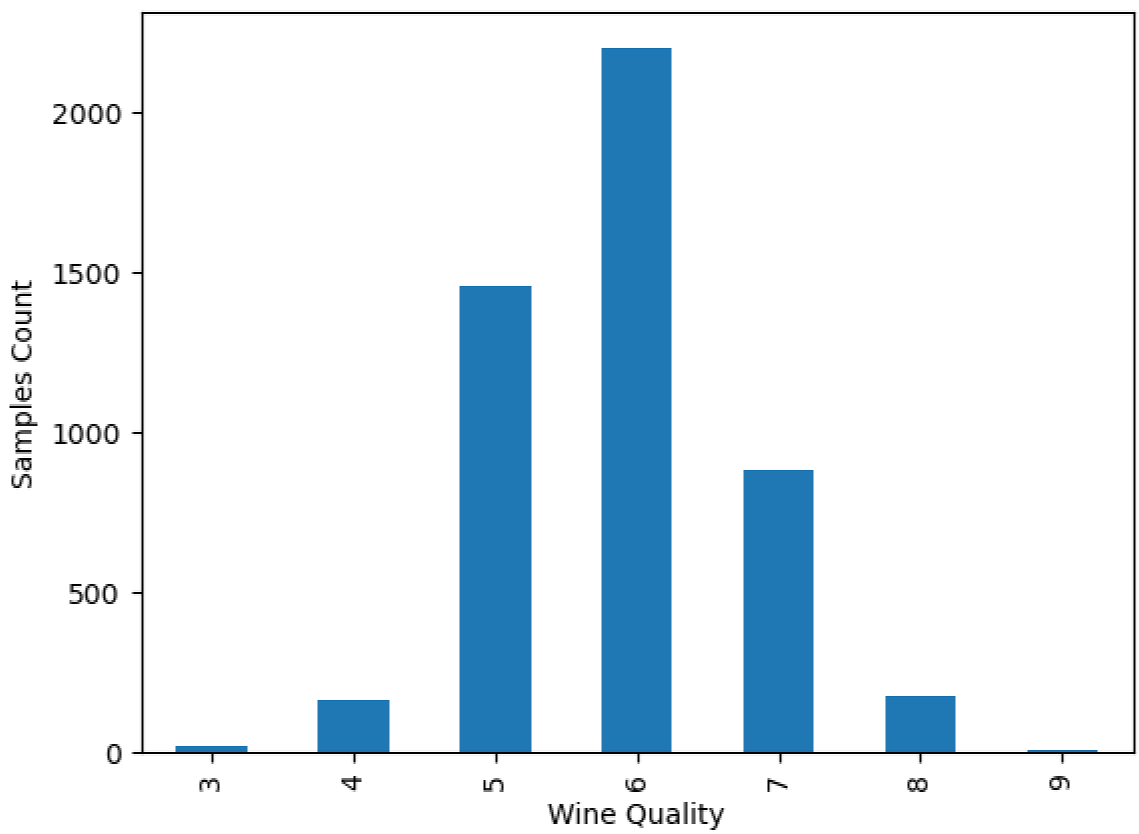
	count	mean	std	min	25%	50%	75%	max
fixed acidity	4898.0	6.854788	0.843868	3.80000	6.300000	6.80000	7.3000	14.20000
volatile acidity	4898.0	0.278241	0.100795	0.08000	0.210000	0.26000	0.3200	1.10000
citric acid	4898.0	0.334192	0.121020	0.00000	0.270000	0.32000	0.3900	1.66000
residual sugar	4898.0	6.391415	5.072058	0.60000	1.700000	5.20000	9.9000	65.80000
chlorides	4898.0	0.045772	0.021848	0.00900	0.036000	0.04300	0.0500	0.34600
free sulfur dioxide	4898.0	35.308085	17.007137	2.00000	23.000000	34.00000	46.0000	289.00000
total sulfur dioxide	4898.0	138.360657	42.498065	9.00000	108.000000	134.00000	167.0000	440.00000
density	4898.0	0.994027	0.002991	0.98711	0.991723	0.99374	0.9961	1.03898
pH	4898.0	3.188267	0.151001	2.72000	3.090000	3.18000	3.2800	3.82000
sulphates	4898.0	0.489847	0.114126	0.22000	0.410000	0.47000	0.5500	1.08000
alcohol	4898.0	10.514267	1.230621	8.00000	9.500000	10.40000	11.4000	14.20000
quality	4898.0	5.877909	0.885639	3.00000	5.000000	6.00000	6.0000	9.00000

In [8]:
```
red_white_dataset.isnull().sum()
```

Out[8]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

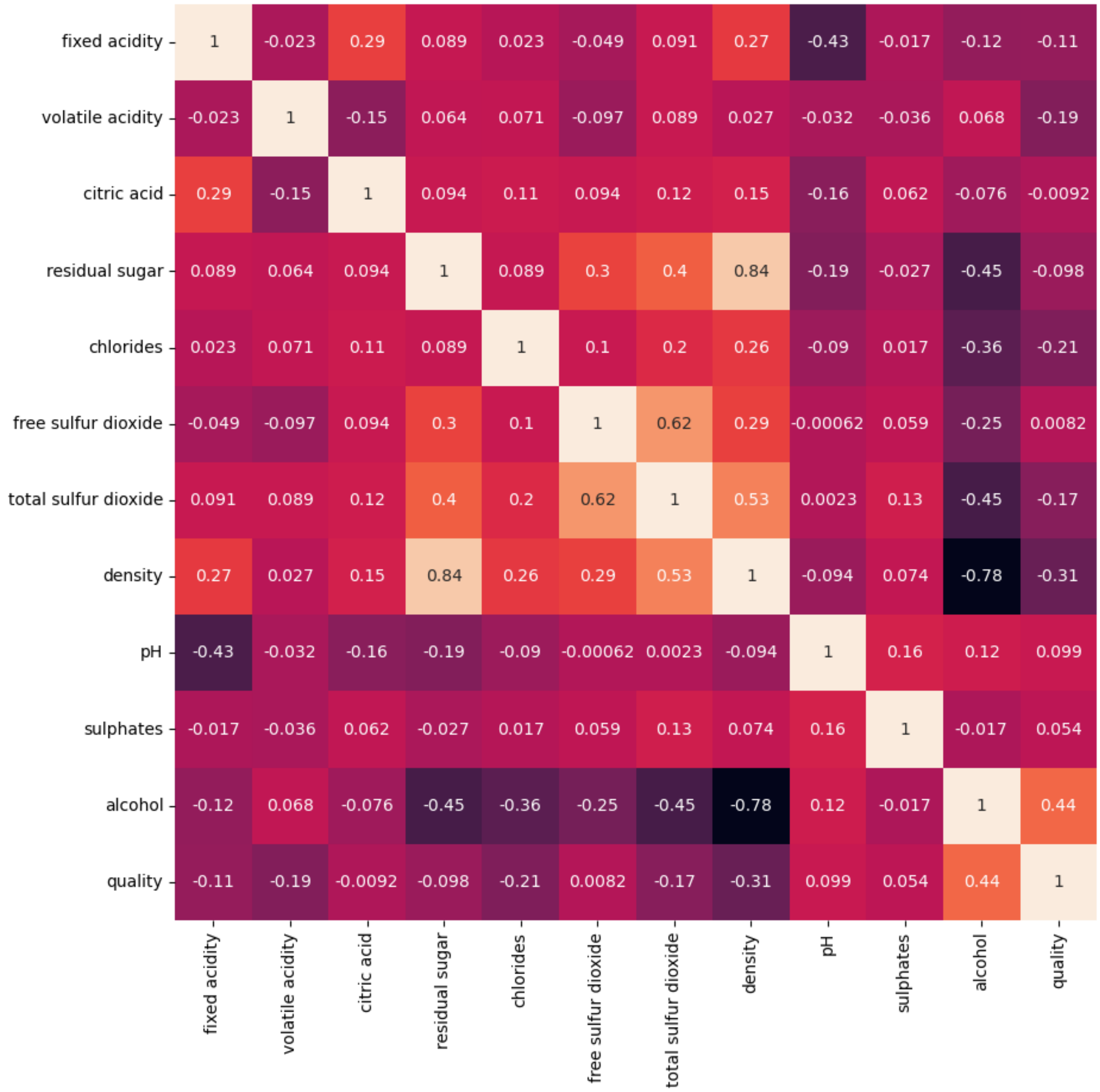
In [9]:
```
# Printing the number of records per quality class (quality score is between 0 and 10)
red_white_dataset['quality'].value_counts().sort_index(ascending=True) \
            .plot(kind='bar', xlabel='Wine Quality', ylabel='Samples Count')
```

Out[9]:
```
<Axes: xlabel='Wine Quality', ylabel='Samples Count'>
```



In [10]:
```
plt.figure(figsize = (10,10))
sns.heatmap(red_white_dataset.corr(), annot=True, cbar=False)
```

Out[10]:
```
<Axes: >
```



In [11]:
```
# Segregating the dependant(Y) and Independant(X) varibles
X = red_white_dataset.drop(columns=['quality'])
print('Dimensions of X (rows,columns):',X.shape)
Y = red_white_dataset['quality']
print('Dimensions of Y (rows,columns):',Y.shape)
```

```
Dimensions of X (rows,columns): (4898, 11)
Dimensions of Y (rows,columns): (4898,)
```

In [12]:
```
X.head()
```

Out[12]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9

In [13]:

```
Y.head()
```

Out[13]:

```
0    6
1    6
2    6
3    6
4    6
Name: quality, dtype: int64
```

Step 3: Splitting the dataset into 80% Train set & 20% Test set

In [14]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=
print('------------------ Train Set -------------------------')
print('Dimensions of X_train (rows,columns):', X_train.shape)
print('Dimensions of Y_train (rows,columns):', Y_train.shape)
print('------------------ Test Set --------------------------')
print('Dimensions of X_test (rows,columns):', X_test.shape)
print('Dimensions of Y_test (rows,columns):', Y_test.shape)
```

```
------------------ Train Set -------------------------
Dimensions of X_train (rows,columns): (3918, 11)
Dimensions of Y_train (rows,columns): (3918,)
------------------ Test Set --------------------------
Dimensions of X_test (rows,columns): (980, 11)
Dimensions of Y_test (rows,columns): (980,)
```

Step 4: Fitting the Ordinal Regression model on data

In [15]:

```
ordinal_model = LogisticIT()
ordinal_model.fit(X_train, Y_train)
```

Out[15]:

```
▾ LogisticIT
LogisticIT()
```

In [16]:

```
ordinal_predictions = ordinal_model.predict(X_test)
```

In [17]:

```
print('Ordinal regression Test MAE:', np.mean(np.abs(Y_test - ordinal_predictions)))
```

```
Ordinal regression Test MAE: 0.5418367346938775
```

Step 5: Fitting the Linear Regression model on data

In [18]:
```
linear_model = LinearRegression()
linear_model.fit(X_train, Y_train)
```

Out[18]:
```
▾ LinearRegression
LinearRegression()
```

In [19]:
```
# Getting the predictions on test data
linear_reg_pred = linear_model.predict(X_test)

# Rounding off the linear regression prediction to nearest integer value
roundedoff_linear_reg_pred = np.rint(linear_reg_pred)
```

In [20]:
```
print('Linear regression Test MAE:', np.mean(np.abs(Y_test - linear_reg_pred)))
```

```
Linear regression Test MAE: 0.5862665383250459
```

Step 6: Comparing Linear regression and Ordinal regression performance

From MAE of ordinal & linear regression models, we can observe that,
Ordinal regression model (MAE = 0.5418) performed better than Linear regression model (MAE = 0.5862)