**EX.NO.:**

**DATE:**

**Wine Quality Data Set  
AIM:**

To perform EDA on Wine Quality Data Set.

**DATASET:**

This is a subset of wine quality dataset which contains only red wine samples.

**PROGRAM:**

import numpy as np  
import pandas as pd  
import seaborn as sns  
import matplotlib.pyplot as plt  
from scipy import stats  
from sklearn.model\_selection import train\_test\_split,cross\_val\_score,GridSearchCV  
from sklearn.metrics import accuracy\_score,precision\_score  
from sklearn.preprocessing import MinMaxScaler  
from warnings import filterwarnings  
filterwarnings(action='ignore')

df = pd.read\_csv('/content/winequality-red.csv')  
df.head()

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

duplicate = df.duplicated()  
print(duplicate.sum())  
df[duplicate]

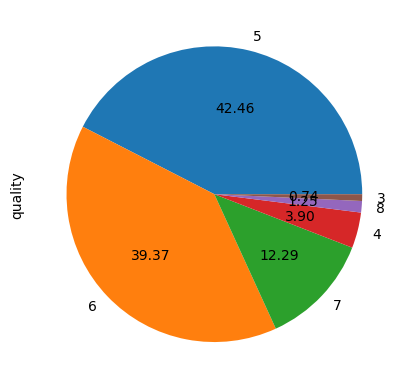
240

fixed acidity volatile acidity citric acid residual sugar chlorides \  
4 7.4 0.700 0.00 1.90 0.076   
11 7.5 0.500 0.36 6.10 0.071   
27 7.9 0.430 0.21 1.60 0.106   
40 7.3 0.450 0.36 5.90 0.074   
65 7.2 0.725 0.05 4.65 0.086   
... ... ... ... ... ...   
1563 7.2 0.695 0.13 2.00 0.076   
1564 7.2 0.695 0.13 2.00 0.076   
1567 7.2 0.695 0.13 2.00 0.076   
1581 6.2 0.560 0.09 1.70 0.053   
1596 6.3 0.510 0.13 2.30 0.076   
  
 free sulfur dioxide total sulfur dioxide density pH sulphates \  
4 11.0 34.0 0.99780 3.51 0.56   
11 17.0 102.0 0.99780 3.35 0.80   
27 10.0 37.0 0.99660 3.17 0.91   
40 12.0 87.0 0.99780 3.33 0.83   
65 4.0 11.0 0.99620 3.41 0.39   
... ... ... ... ... ...   
1563 12.0 20.0 0.99546 3.29 0.54   
1564 12.0 20.0 0.99546 3.29 0.54   
1567 12.0 20.0 0.99546 3.29 0.54   
1581 24.0 32.0 0.99402 3.54 0.60   
1596 29.0 40.0 0.99574 3.42 0.75   
  
 alcohol quality   
4 9.4 5   
11 10.5 5   
27 9.5 5   
40 10.5 5   
65 10.9 5   
... ... ...   
1563 10.1 5   
1564 10.1 5   
1567 10.1 5   
1581 11.3 5   
1596 11.0 6   
  
[240 rows x 12 columns]

df.drop\_duplicates(inplace = True)

df['quality'].value\_counts().plot(kind = 'pie' , autopct='%0.2f')

<Axes: ylabel='quality'>

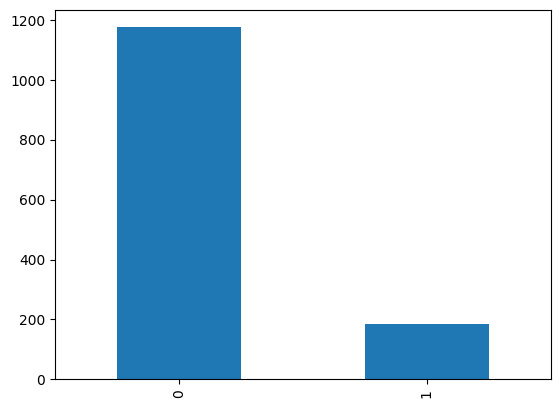


**OBSERVATION:**

Each section is color-coded and the numerical values adjacent to each colored segment indicate the respective percentages they occupy in the pie chart. The chart seems to be a representation of some quality-related data. However, without additional context, it’s hard to infer what exactly these numbers and percentages represent. If you could provide more details or context, I might be able to give a more precise interpretation.

df['quality'] = [1 if val >=7 else 0 for val in df['quality']]

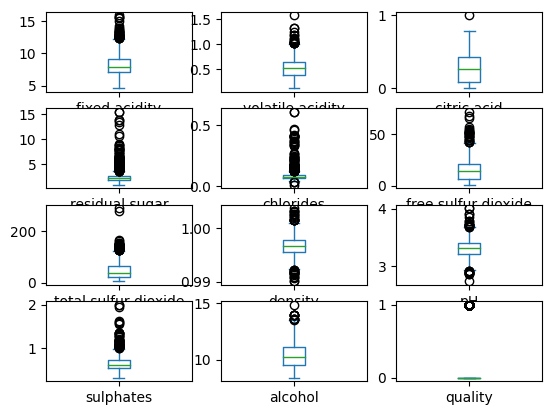
df['quality'].value\_counts().plot(kind = 'bar')  
plt.show()



**OBSERVATION:**

The graph seems to be comparing two categories or groups labeled as “0” and “1”. However, without additional context, it’s hard to infer what exactly these numbers represent. If you could provide more details or context, I might be able to give a more precise interpretation.

df.plot(kind = 'box' , subplots = True , layout = (4,3) , sharex = False)  
plt.show()



**OBSERVATION:**

**Each box plot shows the median, quartiles, and potential outliers (marked as black dots) for each property. However, without additional context, it’s hard to infer more specific observations. If you could provide more details or context, I might be able to give a more precise interpretation.**

def cap\_data(df):  
 df\_copy = df.copy()  
 for col in df\_copy.columns:  
 print("capping the", col)  
 if df\_copy[col].dtype in [float, int]: # No need for separate checks  
 percentiles = df\_copy[col].quantile([0.25, 0.75]).values  
 df\_copy[col][df\_copy[col] <= percentiles[0]] = percentiles[0]  
 df\_copy[col][df\_copy[col] >= percentiles[1]] = percentiles[1]  
 return df\_copy

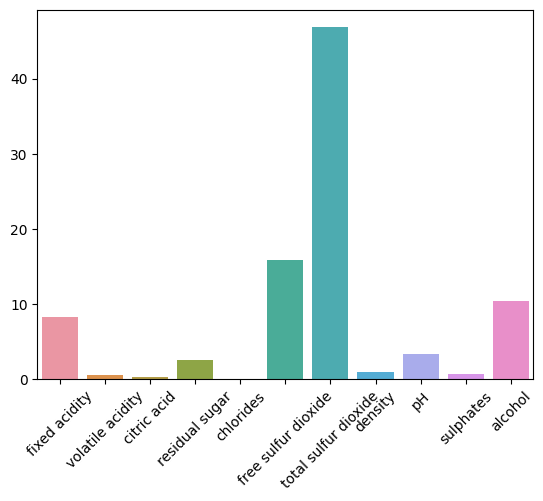
final\_df = cap\_data(df[['fixed acidity','volatile acidity','citric acid','residual sugar','chlorides','free sulfur dioxide','total sulfur dioxide','density','pH','sulphates','alcohol']])  
final\_df['quality'] = df['quality'].copy()

capping the fixed acidity  
capping the volatile acidity  
capping the citric acid  
capping the residual sugar  
capping the chlorides  
capping the free sulfur dioxide  
capping the total sulfur dioxide  
capping the density  
capping the pH  
capping the sulphates  
capping the alcohol

items = []  
  
for col in final\_df.drop('quality', axis=1).columns:  
 items.append(df[col].mean())  
  
items\_array = np.array(items).reshape(1, -1)  
  
ingredient\_dataframe = pd.DataFrame(items\_array, columns=final\_df.drop('quality', axis=1).columns)  
ingredient\_dataframe

fixed acidity volatile acidity citric acid residual sugar chlorides \  
0 8.310596 0.529478 0.272333 2.5234 0.088124   
  
 free sulfur dioxide total sulfur dioxide density pH sulphates \  
0 15.893304 46.825975 0.996709 3.309787 0.658705   
  
 alcohol   
0 10.432315

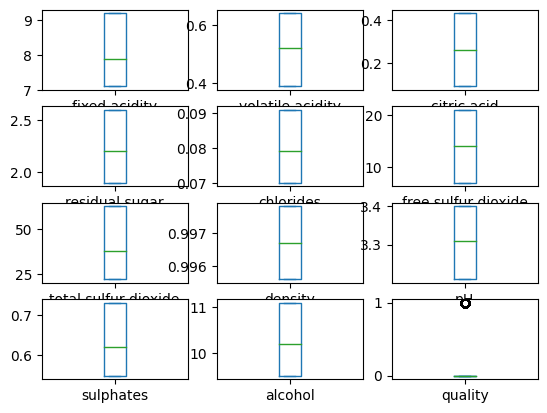
ax = sns.barplot(data=ingredient\_dataframe)  
ax.set\_xticklabels(ax.get\_xticklabels(), rotation=45)  
plt.show()



**OBSERVATION:**

**Each bar’s height indicates the quantity or concentration of each component. However, without additional context, it’s hard to infer more specific observations. If you could provide more details or context, I might be able to give a more precise interpretation.**

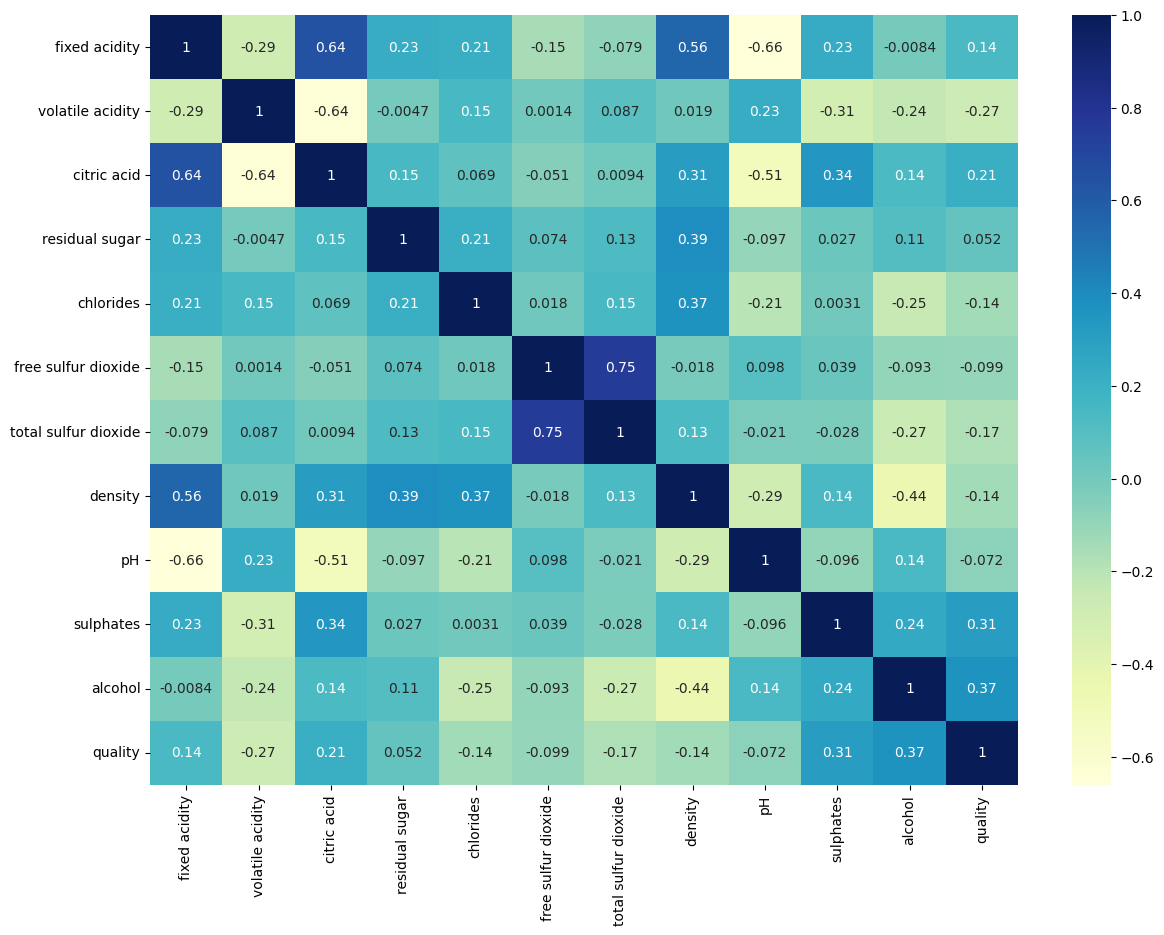
final\_df.plot(kind = 'box' , subplots = True , layout = (4,3) , sharex = False)  
plt.show()



**OBSERVATION:**

The image you provided is a set of box plots representing the distribution of various chemical properties in wines, such as fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, sulphates, alcohol, and quality. Each plot shows the median, upper, and lower quartiles of the data. The ‘quality’ box plot at the bottom right corner is marked with an outlier depicted by a small circle.

plt.figure(figsize=(14,10))  
sns.heatmap(final\_df.corr() , annot=True , cmap='YlGnBu')  
plt.show()

  
**OBSERVATION:**

This heatmap can be useful in understanding the relationships between different wine characteristics and how they might impact the overall quality of the wine. However, without specific numerical values or a clearer view of the heatmap, it’s difficult to provide a more detailed conclusion. Please provide more context or a clearer image for a more comprehensive analysis.

X\_train , X\_test , y\_train , y\_test = train\_test\_split(final\_df.drop(['quality'] , axis = 1) ,  
 final\_df['quality'] , random\_state=45 ,  
 test\_size=0.1)

# It will remove the first feature that is correlated with anything other feature  
  
def correlation(dataset , threshold):  
  
 # Input validation  
 if not isinstance(dataset, pd.DataFrame):  
 raise ValueError("Input 'dataset' must be a pandas DataFrame.")  
 if not (-1 <= threshold <= 1):  
 raise ValueError("Threshold must be between -1 and 1.")  
  
 col\_corr = set() # set of all the names of correlated column  
 corr\_matrix = dataset.corr()  
 for i in range(len(corr\_matrix.columns)):  
 for j in range(i):  
 if abs(corr\_matrix.iloc[i,j]) > threshold:  
 colname = corr\_matrix.columns[i] # Getting the name of the colummn  
 col\_corr.add(colname)  
 return col\_corr

corr\_features = correlation(X\_train , 0.8)  
len(set(corr\_features))

0

scaler = MinMaxScaler()  
X\_train\_trf = scaler.fit\_transform(X\_train)  
X\_test\_trf = scaler.transform(X\_test)

from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import SVC  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.ensemble import AdaBoostClassifier,RandomForestClassifier,IsolationForest,BaggingClassifier,ExtraTreesClassifier,GradientBoostingClassifier  
from xgboost import XGBClassifier  
from catboost import CatBoostClassifier  
from lightgbm import LGBMClassifier  
ModuleNotFoundError Traceback (most recent call last)  
<ipython-input-20-2ef68fa579ff> in <cell line: 8>()  
 6 from sklearn.ensemble import AdaBoostClassifier,RandomForestClassifier,IsolationForest,BaggingClassifier,ExtraTreesClassifier,GradientBoostingClassifier  
 7 from xgboost import XGBClassifier  
----> 8 from catboost import CatBoostClassifier  
 9 from lightgbm import LGBMClassifier  
  
ModuleNotFoundError: No module named 'catboost'  
NOTE: If your import is failing due to a missing package, you can  
manually install dependencies using either !pip or !apt.  
  
To view examples of installing some common dependencies, click the  
"Open Examples" button below.  
svc = SVC(kernel='sigmoid', gamma=1.0)  
knc = KNeighborsClassifier()  
mnb = MultinomialNB()  
dtc = DecisionTreeClassifier(max\_depth=5)  
lrc = LogisticRegression(solver='liblinear', penalty='l1')  
rfc = RandomForestClassifier(n\_estimators=50, random\_state=2)  
ifc = IsolationForest(n\_estimators=50, random\_state=2)  
abc = AdaBoostClassifier(n\_estimators=50, random\_state=2)  
bc = BaggingClassifier(n\_estimators=50, random\_state=2)  
etc = ExtraTreesClassifier(n\_estimators=50, random\_state=2)  
gbdt = GradientBoostingClassifier(n\_estimators=50,random\_state=2)  
xgb = XGBClassifier(n\_estimators=50,random\_state=2)  
cat = CatBoostClassifier(n\_estimators=50,random\_state=2)  
lgb = LGBMClassifier(n\_estimators=50,random\_state=2)

---------------------------------------------------------------------------  
NameError Traceback (most recent call last)  
<ipython-input-21-91d21be4adb9> in <cell line: 13>()  
 11 gbdt = GradientBoostingClassifier(n\_estimators=50,random\_state=2)  
 12 xgb = XGBClassifier(n\_estimators=50,random\_state=2)  
---> 13 cat = CatBoostClassifier(n\_estimators=50,random\_state=2)  
 14 lgb = LGBMClassifier(n\_estimators=50,random\_state=2)  
  
NameError: name 'CatBoostClassifier' is not defined

classifiers = {  
 'SVC' : svc,  
 'KN' : knc,  
 'NB': mnb,  
 'DT': dtc,  
 'LR': lrc,  
 'RF': rfc,  
 'ifc': ifc,  
 'AdaBoost': abc,  
 'BgC': bc,  
 'ETC': etc,  
 'GBDT':gbdt,  
 'xgb':xgb,  
 'cat':cat,  
 'lgb':lgb  
}

---------------------------------------------------------------------------  
NameError Traceback (most recent call last)  
<ipython-input-22-a023ebd524ac> in <cell line: 2>()  
 12 'GBDT':gbdt,  
 13 'xgb':xgb,  
---> 14 'cat':cat,  
 15 'lgb':lgb  
 16 }  
  
NameError: name 'cat' is not defined

def train\_classifier(clf, X\_train, y\_train, X\_test, y\_test):  
 clf.fit(X\_train, y\_train)  
  
 y\_pred = clf.predict(X\_test)  
  
 accuracy = accuracy\_score(y\_test, y\_pred)  
 precision = precision\_score(y\_test, y\_pred, average='weighted')  
  
 return accuracy, precision

accuracy\_scores = []  
precision\_scores = []  
  
for name,clf in classifiers.items():  
  
 current\_accuracy,current\_precision = train\_classifier(clf, X\_train\_trf,y\_train,X\_test\_trf,y\_test)  
  
 print("For ",name)  
 print("Accuracy - ",current\_accuracy)  
 print("Precision - ",current\_precision)  
  
 accuracy\_scores.append(current\_accuracy)  
 precision\_scores.append(current\_precision)

For SVC  
Accuracy - 0.7720588235294118  
Precision - 0.77805239742956  
For KN  
Accuracy - 0.9191176470588235  
Precision - 0.9170312753200455  
For NB  
Accuracy - 0.8823529411764706  
Precision - 0.7785467128027681  
For DT  
Accuracy - 0.8897058823529411  
Precision - 0.8927335640138407  
For LR  
Accuracy - 0.9117647058823529  
Precision - 0.9073563851770218  
For RF  
Accuracy - 0.8823529411764706  
Precision - 0.8700189753320683  
For ifc  
Accuracy - 0.0  
Precision - 0.0  
For AdaBoost  
Accuracy - 0.9117647058823529  
Precision - 0.9117647058823529  
For BgC  
Accuracy - 0.8897058823529411  
Precision - 0.8867606546750931  
For ETC  
Accuracy - 0.875  
Precision - 0.8649155722326455  
For GBDT  
Accuracy - 0.8970588235294118  
Precision - 0.8869386464263124  
For xgb  
Accuracy - 0.8897058823529411  
Precision - 0.8811389471360777  
Learning rate set to 0.175109  
0: learn: 0.6095490 total: 148ms remaining: 7.23s  
1: learn: 0.5463367 total: 149ms remaining: 3.58s  
2: learn: 0.4987047 total: 151ms remaining: 2.36s  
3: learn: 0.4601700 total: 152ms remaining: 1.75s  
4: learn: 0.4304117 total: 154ms remaining: 1.38s  
5: learn: 0.4019515 total: 155ms remaining: 1.13s  
6: learn: 0.3821320 total: 156ms remaining: 958ms  
7: learn: 0.3642696 total: 158ms remaining: 828ms  
8: learn: 0.3499192 total: 159ms remaining: 726ms  
9: learn: 0.3360945 total: 160ms remaining: 641ms  
10: learn: 0.3274360 total: 162ms remaining: 574ms  
11: learn: 0.3176992 total: 163ms remaining: 517ms  
12: learn: 0.3098309 total: 165ms remaining: 469ms  
13: learn: 0.3015097 total: 166ms remaining: 427ms  
14: learn: 0.2945209 total: 167ms remaining: 391ms  
15: learn: 0.2883442 total: 169ms remaining: 359ms  
16: learn: 0.2838811 total: 171ms remaining: 331ms  
17: learn: 0.2785106 total: 172ms remaining: 307ms  
18: learn: 0.2741958 total: 174ms remaining: 284ms  
19: learn: 0.2706270 total: 175ms remaining: 263ms  
20: learn: 0.2661222 total: 177ms remaining: 244ms  
21: learn: 0.2629114 total: 178ms remaining: 227ms  
22: learn: 0.2600341 total: 180ms remaining: 211ms  
23: learn: 0.2569646 total: 181ms remaining: 196ms  
24: learn: 0.2530899 total: 183ms remaining: 183ms  
25: learn: 0.2499421 total: 184ms remaining: 170ms  
26: learn: 0.2481356 total: 186ms remaining: 158ms  
27: learn: 0.2454760 total: 187ms remaining: 147ms  
28: learn: 0.2429308 total: 189ms remaining: 137ms  
29: learn: 0.2408115 total: 190ms remaining: 127ms  
30: learn: 0.2387768 total: 191ms remaining: 117ms  
31: learn: 0.2354433 total: 193ms remaining: 108ms  
32: learn: 0.2331784 total: 194ms remaining: 100ms  
33: learn: 0.2312679 total: 196ms remaining: 92.1ms  
34: learn: 0.2293053 total: 197ms remaining: 84.6ms  
35: learn: 0.2275462 total: 199ms remaining: 77.3ms  
36: learn: 0.2253462 total: 200ms remaining: 70.4ms  
37: learn: 0.2240164 total: 202ms remaining: 63.8ms  
38: learn: 0.2223988 total: 203ms remaining: 57.4ms  
39: learn: 0.2207730 total: 205ms remaining: 51.2ms  
40: learn: 0.2189043 total: 206ms remaining: 45.3ms  
41: learn: 0.2162479 total: 208ms remaining: 39.6ms  
42: learn: 0.2146772 total: 209ms remaining: 34.1ms  
43: learn: 0.2131053 total: 211ms remaining: 28.8ms  
44: learn: 0.2108250 total: 212ms remaining: 23.6ms  
45: learn: 0.2091346 total: 214ms remaining: 18.6ms  
46: learn: 0.2075613 total: 216ms remaining: 13.8ms  
47: learn: 0.2059893 total: 217ms remaining: 9.05ms  
48: learn: 0.2041406 total: 219ms remaining: 4.46ms  
49: learn: 0.2022993 total: 220ms remaining: 0us  
For cat  
Accuracy - 0.9117647058823529  
Precision - 0.9038583175205566  
[LightGBM] [Info] Number of positive: 168, number of negative: 1055  
[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000659 seconds.  
You can set `force\_col\_wise=true` to remove the overhead.  
[LightGBM] [Info] Total Bins 323  
[LightGBM] [Info] Number of data points in the train set: 1223, number of used features: 11  
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.137367 -> initscore=-1.837332  
[LightGBM] [Info] Start training from score -1.837332  
For lgb  
Accuracy - 0.8823529411764706  
Precision - 0.8700189753320683

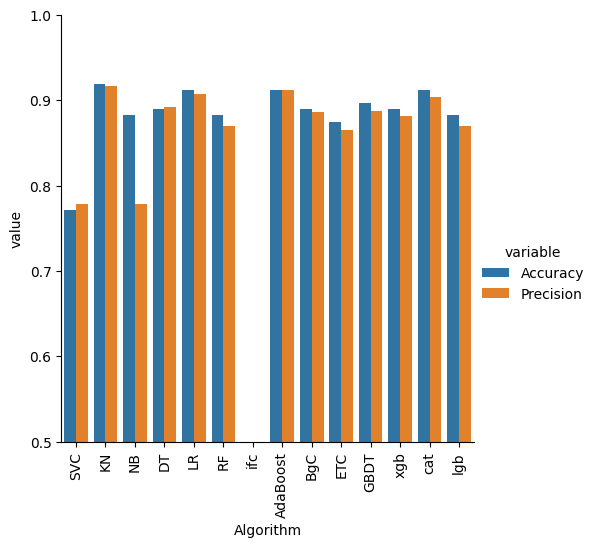
performance\_df = pd.DataFrame({'Algorithm':classifiers.keys(),'Accuracy':accuracy\_scores , 'Precision':precision\_scores})  
performance\_df

Algorithm Accuracy Precision  
0 SVC 0.772059 0.778052  
1 KN 0.919118 0.917031  
2 NB 0.882353 0.778547  
3 DT 0.889706 0.892734  
4 LR 0.911765 0.907356  
5 RF 0.882353 0.870019  
6 ifc 0.000000 0.000000  
7 AdaBoost 0.911765 0.911765  
8 BgC 0.889706 0.886761  
9 ETC 0.875000 0.864916  
10 GBDT 0.897059 0.886939  
11 xgb 0.889706 0.881139  
12 cat 0.911765 0.903858  
13 lgb 0.882353 0.870019

performance\_df1 = pd.melt(performance\_df, id\_vars = "Algorithm")  
performance\_df1

---------------------------------------------------------------------------  
NameError Traceback (most recent call last)  
<ipython-input-24-597a15d946cc> in <cell line: 1>()  
----> 1 performance\_df1 = pd.melt(performance\_df, id\_vars = "Algorithm")  
 2 performance\_df1  
  
NameError: name 'performance\_df' is not defined

# Categorical Plot  
sns.catplot(x='Algorithm', y='value', hue='variable', data=performance\_df1, kind='bar', height=5)  
plt.ylim(0.5, 1.0)  
plt.xticks(rotation='vertical')  
plt.show()



**OBSERVATION:**

**The image you provided is a bar graph comparing the accuracy and precision of various machine learning algorithms. Here are some observations:**

**The graph includes a variety of machine learning algorithms such as SVC, KNC, NB, DT, LR, RF, AdaBoost, BGC, ETC, GBDT, xgb, cat, and lgb.**

X = final\_df.iloc[: , :11]  
y = final\_df.iloc[: , -1]

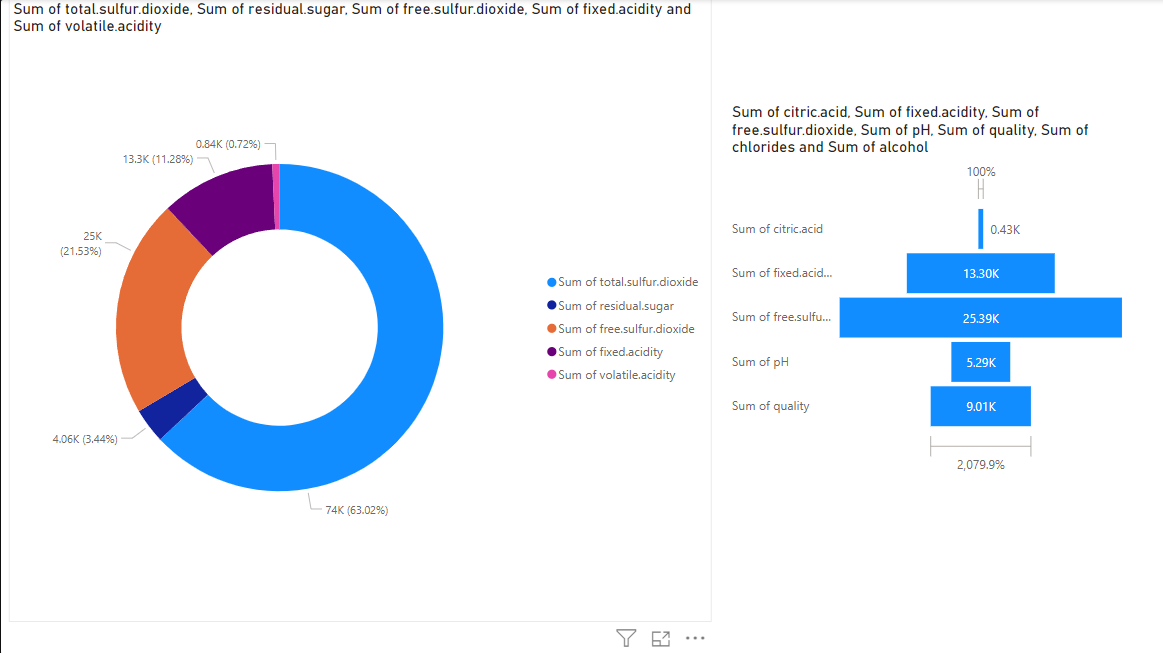
from sklearn.ensemble import VotingClassifier

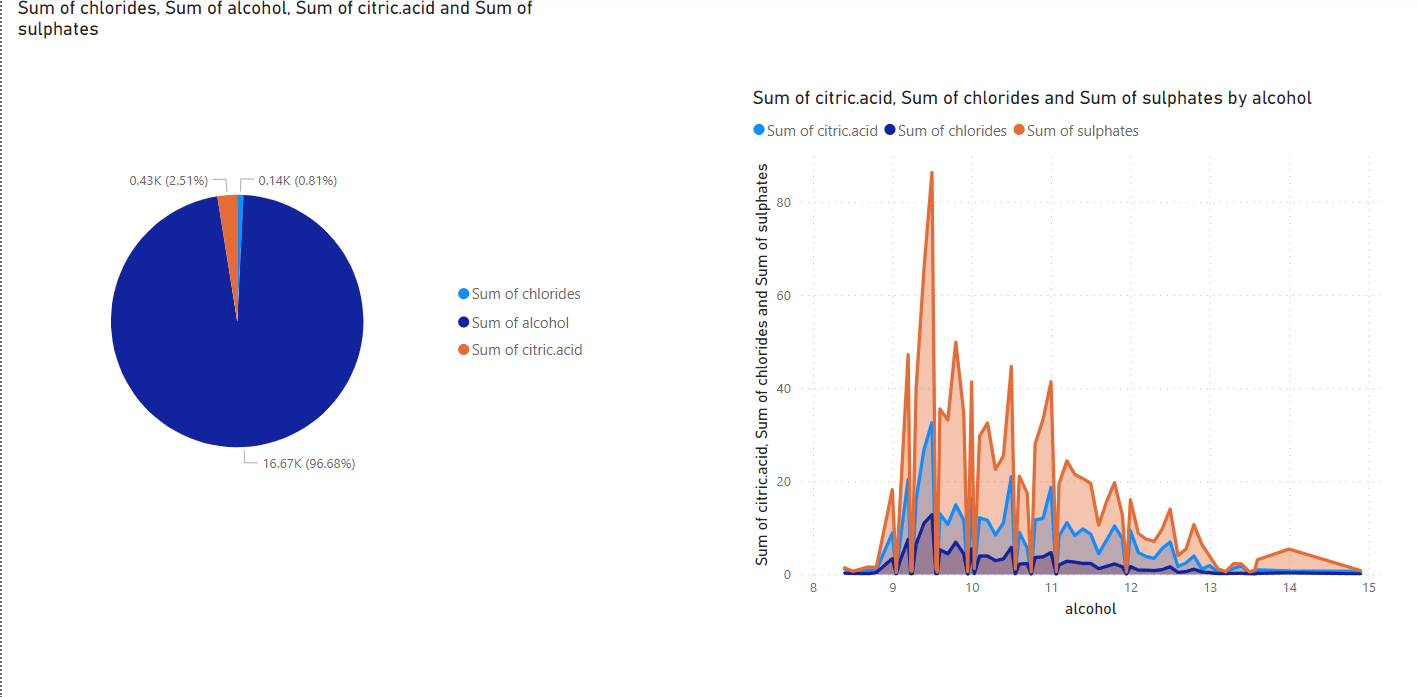
voting = VotingClassifier(estimators=[  
 ('AdaBoost', abc),  
 ('KN', knc),  
 ('cat',cat)  
],voting='soft')

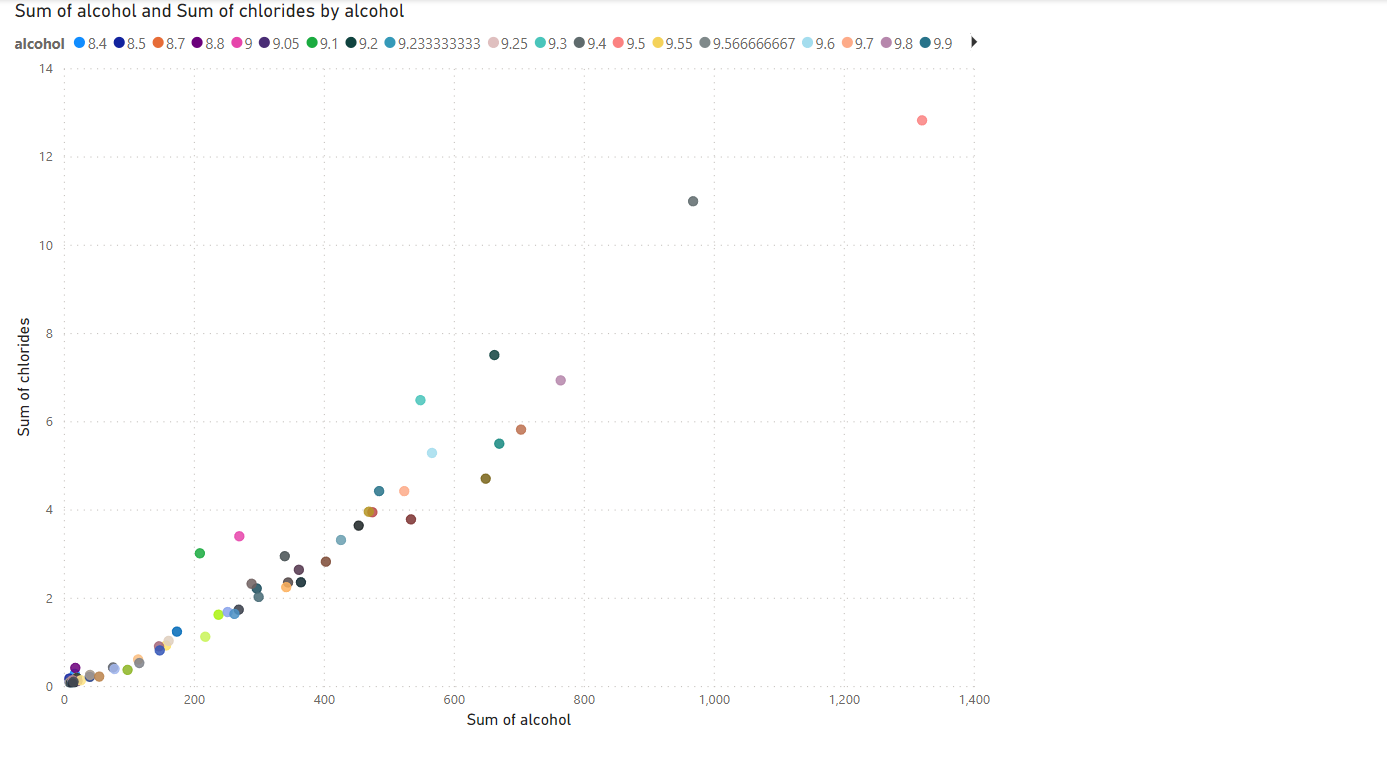
voting.fit(X\_train\_trf , y\_train)  
y\_pred = voting.predict(X\_test\_trf)  
print('Accuracy Score is ::',accuracy\_score(y\_test , y\_pred))  
print('Cross Val Score is ::',np.mean(cross\_val\_score(voting , X , y , scoring='accuracy' , cv = 5))) #85

Learning rate set to 0.175109  
0: learn: 0.6095490 total: 2.79ms remaining: 137ms  
1: learn: 0.5463367 total: 4.66ms remaining: 112ms  
2: learn: 0.4987047 total: 6.25ms remaining: 98ms  
3: learn: 0.4601700 total: 7.87ms remaining: 90.5ms  
4: learn: 0.4304117 total: 9.41ms remaining: 84.7ms  
5: learn: 0.4019515 total: 10.3ms remaining: 75.8ms  
6: learn: 0.3821320 total: 11.7ms remaining: 72.1ms  
7: learn: 0.3642696 total: 13.1ms remaining: 68.9ms  
8: learn: 0.3499192 total: 14.6ms remaining: 66.7ms  
9: learn: 0.3360945 total: 15.8ms remaining: 63.1ms  
10: learn: 0.3274360 total: 17.1ms remaining: 60.6ms  
11: learn: 0.3176992 total: 18.6ms remaining: 58.9ms  
12: learn: 0.3098309 total: 20.1ms remaining: 57.2ms  
13: learn: 0.3015097 total: 21.5ms remaining: 55.4ms  
14: learn: 0.2945209 total: 22.9ms remaining: 53.5ms  
15: learn: 0.2883442 total: 24.3ms remaining: 51.6ms  
16: learn: 0.2838811 total: 25.7ms remaining: 49.9ms  
17: learn: 0.2785106 total: 27.1ms remaining: 48.1ms  
18: learn: 0.2741958 total: 28.5ms remaining: 46.4ms  
19: learn: 0.2706270 total: 30.2ms remaining: 45.3ms  
20: learn: 0.2661222 total: 31.6ms remaining: 43.7ms  
21: learn: 0.2629114 total: 33.4ms remaining: 42.5ms  
22: learn: 0.2600341 total: 34.9ms remaining: 41ms  
  
Accuracy Score is :: 0.9117647058823529  
Learning rate set to 0.166513  
0: learn: 0.6200855 total: 2.53ms remaining: 124ms  
1: learn: 0.5608856 total: 4.26ms remaining: 102ms  
2: learn: 0.5160913 total: 5.7ms remaining: 89.3ms  
3: learn: 0.4756587 total: 7.02ms remaining: 80.7ms  
4: learn: 0.4452519 total: 8.37ms remaining: 75.3ms  
5: learn: 0.4147791 total: 9.05ms remaining: 66.3ms  
6: learn: 0.3934474 total: 10.4ms remaining: 63.8ms  
7: learn: 0.3758283 total: 11.9ms remaining: 62.5ms  
8: learn: 0.3606133 total: 13.4ms remaining: 61ms  
9: learn: 0.3450759 total: 14.8ms remaining: 59.2ms  
10: learn: 0.3320763 total: 16.2ms remaining: 57.4ms  
11: learn: 0.3232229 total: 18.3ms remaining: 58ms  
12: learn: 0.3142848 total: 20ms remaining: 56.9ms  
13: learn: 0.3055437 total: 21.4ms remaining: 54.9ms  
14: learn: 0.2988255 total: 22.8ms remaining: 53.3ms  
15: learn: 0.2925646 total: 24.3ms remaining: 51.6ms  
16: learn: 0.2850733 total: 25.7ms remaining: 50ms  
17: learn: 0.2807400 total: 27.2ms remaining: 48.3ms  
18: learn: 0.2768777 total: 28.6ms remaining: 46.6ms  
19: learn: 0.2716131 total: 30.2ms remaining: 45.3ms  
20: learn: 0.2676031 total: 32ms remainin  
Learning rate set to 0.166513  
Cross Val Score is :: 0.8594557195571955

**POWERBI:**

****

****

****

**RESULT:**

Thus the performation of EDA on Wine Quality Data Set has been done successfully.