The selection of quality strawberries.

```
#This data analyst for the selection of quality strawberries
#there are 9 point parameter data
# fixed acidity,volatile acidity,     citric acid,     residual sugar,
# chlorides,     free sulfur dioxide,  total sulfur dioxide, density,
      pH,
# sulphates,     alcohol, and     quality

#import library python
#numpy for make a array, matplotlib for visualization data ex: table
or plot
#seaborn for complicated matplotlib, and pandas for make table data
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

from google.colab import drive

drive.mount('/content/drive')
df = pd.read_csv('/content/drive/My Drive/Colab
Notebooks/progresproject/posting/quality_strawberries.csv')
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
#print df value for the next information
print(df)
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
0               7.4             0.700         0.00             1.9
0.076
1               7.8             0.880         0.00             2.6
0.098
2               7.8             0.760         0.04             2.3
0.092
3              11.2             0.280         0.56             1.9
0.075
4               7.4             0.700         0.00             1.9
0.076
...             ...               ...          ...             ...
...
1594            6.2             0.600         0.08             2.0
0.090
1595            5.9             0.550         0.10             2.2
0.062
1596            6.3             0.510         0.13             2.3
0.076
```

```
1597            5.9             0.645        0.12             2.0
0.075
1598            6.0             0.310        0.47             3.6
0.067

      free sulfur dioxide  total sulfur dioxide  density    pH
sulphates  \
0                    11.0                  34.0  0.99780  3.51
0.56
1                    25.0                  67.0  0.99680  3.20
0.68
2                    15.0                  54.0  0.99700  3.26
0.65
3                    17.0                  60.0  0.99800  3.16
0.58
4                    11.0                  34.0  0.99780  3.51
0.56
...                   ...                   ...      ...   ...
...
1594                 32.0                  44.0  0.99490  3.45
0.58
1595                 39.0                  51.0  0.99512  3.52
0.76
1596                 29.0                  40.0  0.99574  3.42
0.75
1597                 32.0                  44.0  0.99547  3.57
0.71
1598                 18.0                  42.0  0.99549  3.39
0.66

      alcohol  quality
0         9.4        5
1         9.8        5
2         9.8        5
3         9.8        6
4         9.4        5
...       ...      ...
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
1598     11.0        6

[1599 rows x 12 columns]
```

```
#To display the first five rows of a DataFrame
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

#To display the last five rows of a DataFrame
df.tail()

      fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
1594            6.2             0.600         0.08             2.0
0.090
1595            5.9             0.550         0.10             2.2
0.062
1596            6.3             0.510         0.13             2.3
0.076
1597            5.9             0.645         0.12             2.0
0.075
1598            6.0             0.310         0.47             3.6
0.067

      free sulfur dioxide  total sulfur dioxide  density    pH
sulphates  \
1594                 32.0                  44.0  0.99490  3.45
```

```
0.58
1595                 39.0                  51.0  0.99512  3.52
0.76
1596                 29.0                  40.0  0.99574  3.42
0.75
1597                 32.0                  44.0  0.99547  3.57
0.71
1598                 18.0                  42.0  0.99549  3.39
0.66

      alcohol  quality
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
1598     11.0        6
```

```
#to display a raw and column
df.shape
```

```
(1599, 12)
```

```
#provides detailed information about the DataFrame
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

```
#for calculate the number of missing values
df.isna().sum()
```

```
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
```

```
#identify rows that have duplicates
duplicates=df.duplicated()
df[duplicates]
```

```
      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]
```

```
print(duplicates)
```

```
0       False
1       False
2       False
3       False
4        True
        ...
1594    False
1595    False
1596     True
1597    False
1598    False
Length: 1599, dtype: bool
```

```
df=df.drop_duplicates()
```

```
#used to access the list of column names
df.columns
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual
sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide',
'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

```
#generate summary statistics from numeric columns
df.describe()
```

```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1359.000000       1359.000000  1359.000000     1359.000000
mean        8.310596          0.529478     0.272333        2.523400
std         1.736990          0.183031     0.195537        1.352314
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.390000     0.090000        1.900000
50%         7.900000          0.520000     0.260000        2.200000
75%         9.200000          0.640000     0.430000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide
density  \
count  1359.000000          1359.000000           1359.000000
1359.000000
mean      0.088124            15.893304             46.825975
0.996709
std       0.049377            10.447270             33.408946
0.001869
min       0.012000             1.000000              6.000000
0.990070
25%       0.070000             7.000000             22.000000
0.995600
50%       0.079000            14.000000             38.000000
0.996700
75%       0.091000            21.000000             63.000000
0.997820
max       0.611000            72.000000            289.000000
1.003690

                pH    sulphates      alcohol      quality
count  1359.000000  1359.000000  1359.000000  1359.000000
mean      3.309787     0.658705    10.432315     5.623252
std       0.155036     0.170667     1.082065     0.823578
min       2.740000     0.330000     8.400000     3.000000
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
max       4.010000     2.000000    14.900000     8.000000
```

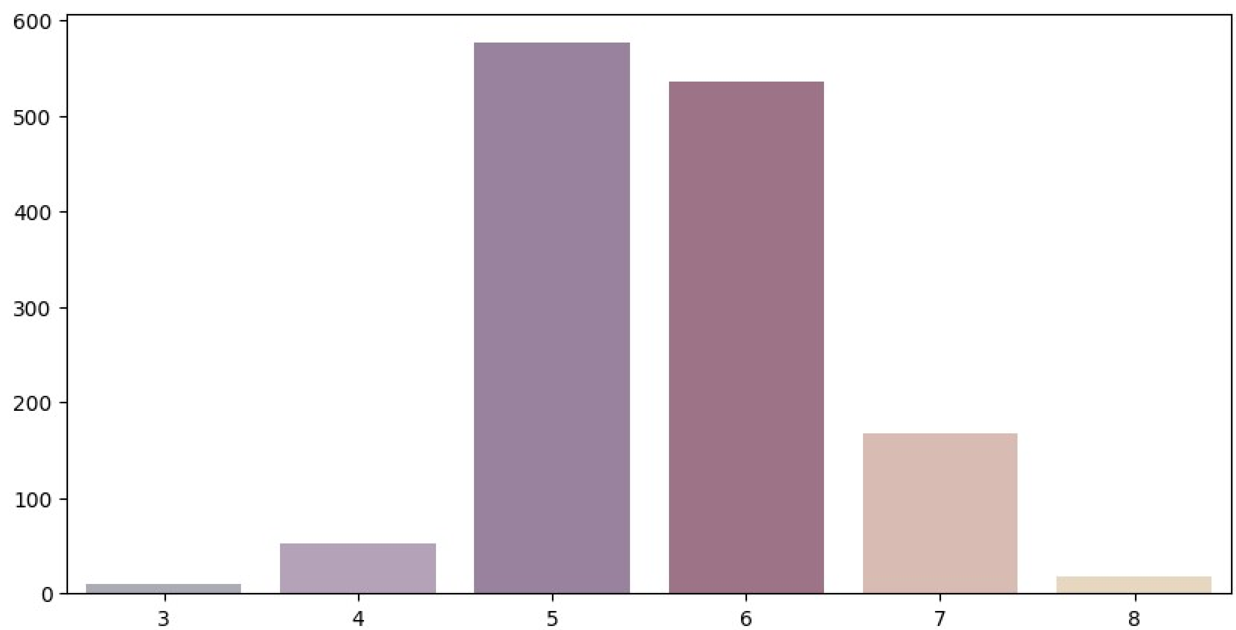
```
#gets a list of unique values contained in the "quality" column of the DataFrame
df['quality'].unique()
```

```
array([5, 6, 7, 4, 8, 3])
```

Data Visualization

```
#get color for bar plot visualization data
grey_palette = 
sns.color_palette(['#AAAAB6','#B79EBC','#9C7DA1','#A46D87','#DEB8AD','#EDD8BB'])
sns.set_palette(grey_palette)

plt.figure(figsize=(10,5))
sns.barplot(x=df['quality'].value_counts().index,y=df['quality'].value_counts().values)
```

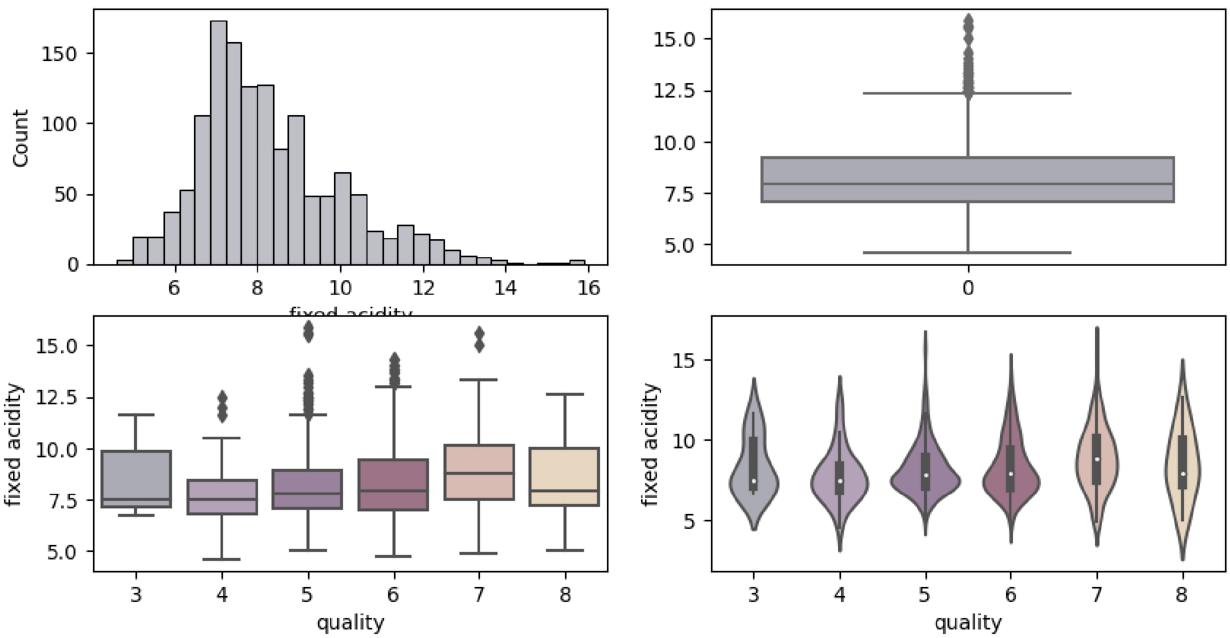
```
<Axes: >
```




```
#parameter fixed acidity
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.histplot(x=df['fixed acidity'])
plt.subplot(2,2,2)
sns.boxplot(df['fixed acidity'])
plt.subplot(2,2,3)
sns.boxplot(x=df['quality'],y=df['fixed acidity'])
plt.subplot(2,2,4)
```

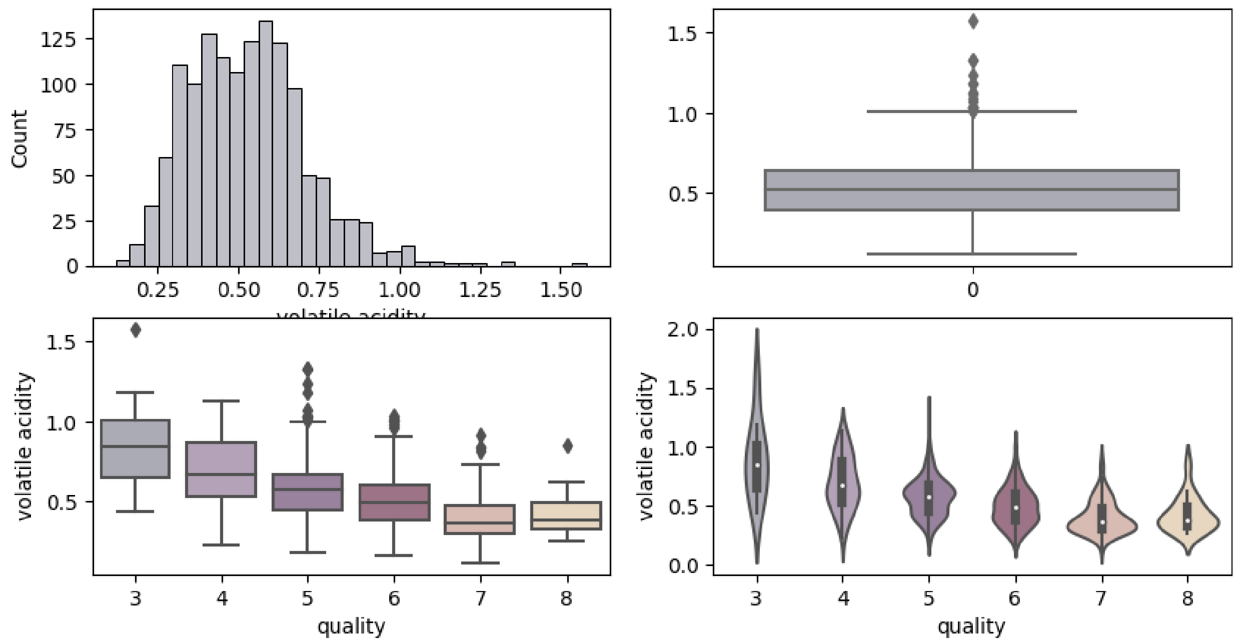
```
sns.violinplot(x=df['quality'],y=df['fixed acidity'])
print('Correlation between fixed acidity and quality is',df['quality'].corr(df['fixed acidity']))
print('Skewness of the fixed column is',df['fixed acidity'].skew())
```

```
Correlation between fixed acidity and quality is 0.1190236656134977
Skewness of the fixed column is 0.9410413664561449
```



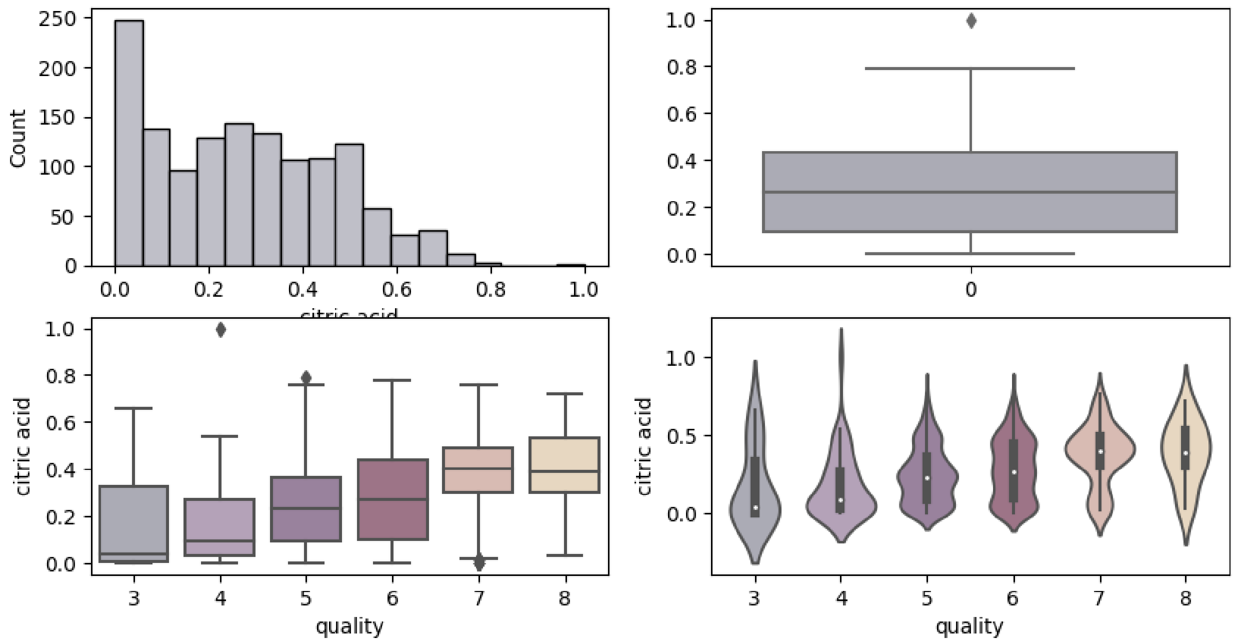

```
#conclution fixed acidity parameter
#1. There is no correlation or relationship between "quality" and "fixed acidity"
#2. All available grades of strawberry have an acidity level that remains within a fixed range of 7-9

#parameters volatile acidity
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.histplot(x=df['volatile acidity'])
plt.subplot(2,2,2)
sns.boxplot(df['volatile acidity'])
plt.subplot(2,2,3)
sns.boxplot(x=df['quality'],y=df['volatile acidity'])
plt.subplot(2,2,4)
sns.violinplot(x=df['quality'],y=df['volatile acidity'])
print('Correlation between volatile acidity and quality is',df['quality'].corr(df['volatile acidity']))
print('Skewness of the volatile acidity column is',df['volatile acidity'].skew())
```

```
Correlation between volatile acidity and quality is -0.39521368900984055
Skewness of the volatile acidity column is 0.7292789463991854
```



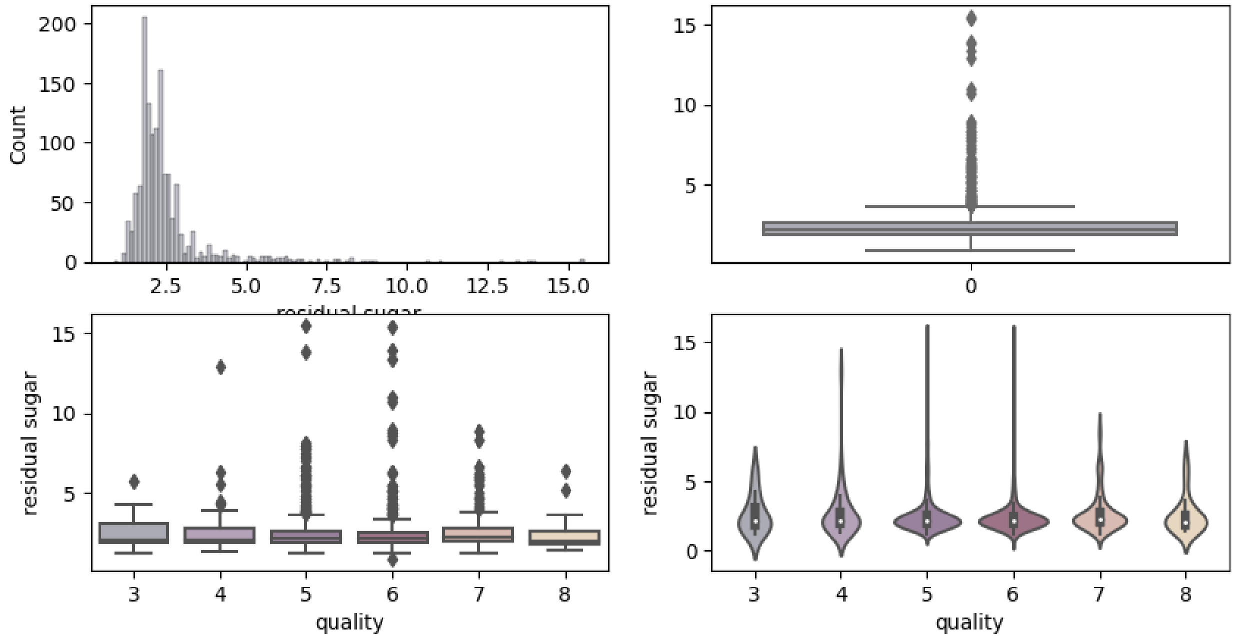

```
#1. there are corelation "volatile acility" dan "quality", pada quality 3 range total is 0.65 until 1
#quality 4 is 0.50 until 0.85, quality 5 is 0.45 until 0.65, quality 6 in range 0.45 until 0.60
#quality 7 in range 0.25 until 0,50, and quality 8 in range 0.30 until 0.50
#2. there is negative corelation between "volatile acility" dan "quality"
#3. The lower the volatile acidity value, the better the quality of the strawberries selected

#parameters citric acid
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.histplot(x=df['citric acid'])
plt.subplot(2,2,2)
sns.boxplot(df['citric acid'])
plt.subplot(2,2,3)
sns.boxplot(x=df['quality'],y=df['citric acid'])
plt.subplot(2,2,4)
sns.violinplot(x=df['quality'],y=df['citric acid'])
print('Correlation between citric acid and quality is',df['quality'].corr(df['citric acid']))
print('Skewness of the citric acid column is',df['citric acid'].skew())
```

```
Correlation between citric acid and quality is 0.22805745919929968
Skewness of the citric acid column is 0.31272554238899036
```



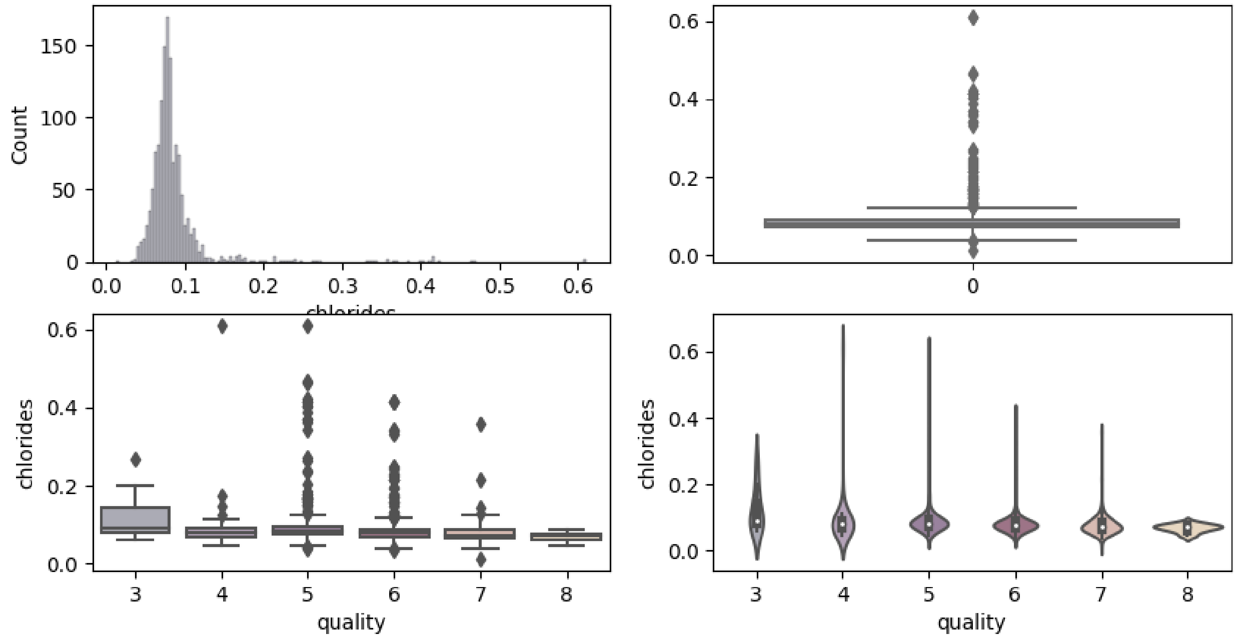

```
#1. there is positive corelation between "citric acid" dan "quality"
#2. The higher the citric acid value, the higher the quality of the
strawberries selected

#parameter residual sugar
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.histplot(x=df['residual sugar'])
plt.subplot(2,2,2)
sns.boxplot(df['residual sugar'])
plt.subplot(2,2,3)
sns.boxplot(x=df['quality'],y=df['residual sugar'])
plt.subplot(2,2,4)
sns.violinplot(x=df['quality'],y=df['residual sugar'])
print('Correlation between residual sugar and quality
is',df['quality'].corr(df['residual sugar']))
print('Skewness of the residual sugar column is',df['residual
sugar'].skew())
```

```
Correlation between residual sugar and quality is 0.013640470048445878
Skewness of the residual sugar column is 4.548153403940447
```



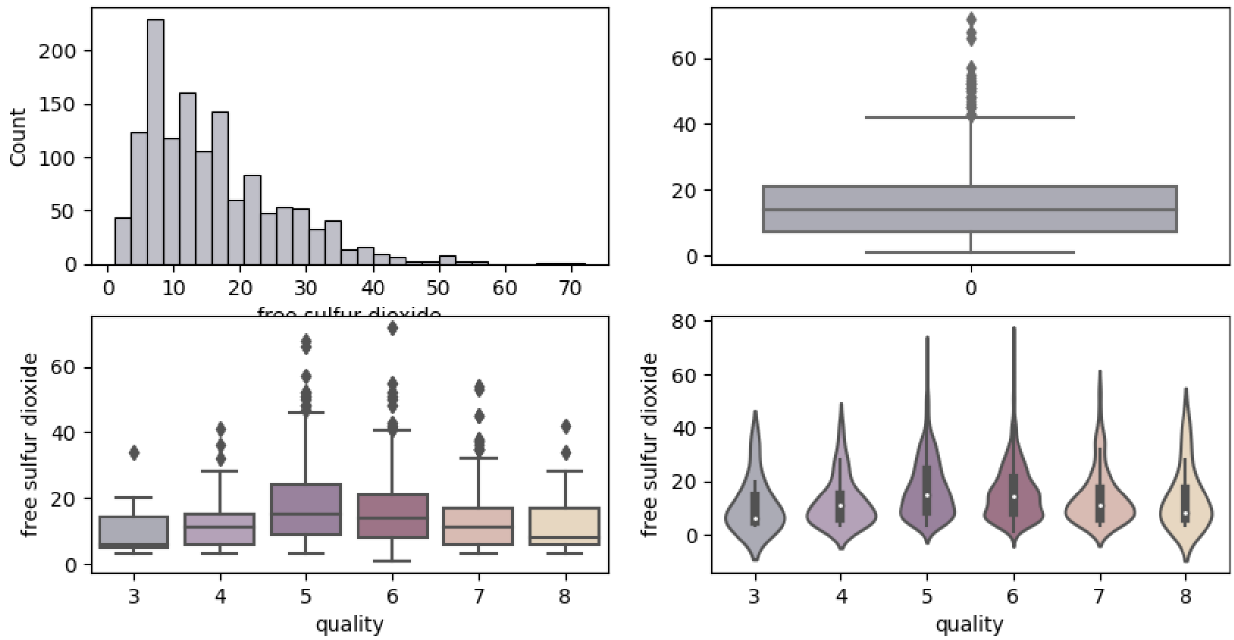

```
#parameter chlorides
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.histplot(x=df['chlorides'])
plt.subplot(2,2,2)
sns.boxplot(df['chlorides'])
plt.subplot(2,2,3)
sns.boxplot(x=df['quality'],y=df['chlorides'])
plt.subplot(2,2,4)
sns.violinplot(x=df['quality'],y=df['chlorides'])
print('Correlation between chlorides and quality
is',df['quality'].corr(df['chlorides']))
print('Skewness of the chlorides column is',df['chlorides'].skew())
```

```
Correlation between chlorides and quality is -0.13098841286642665
Skewness of the chlorides column is 5.502487294623722
```



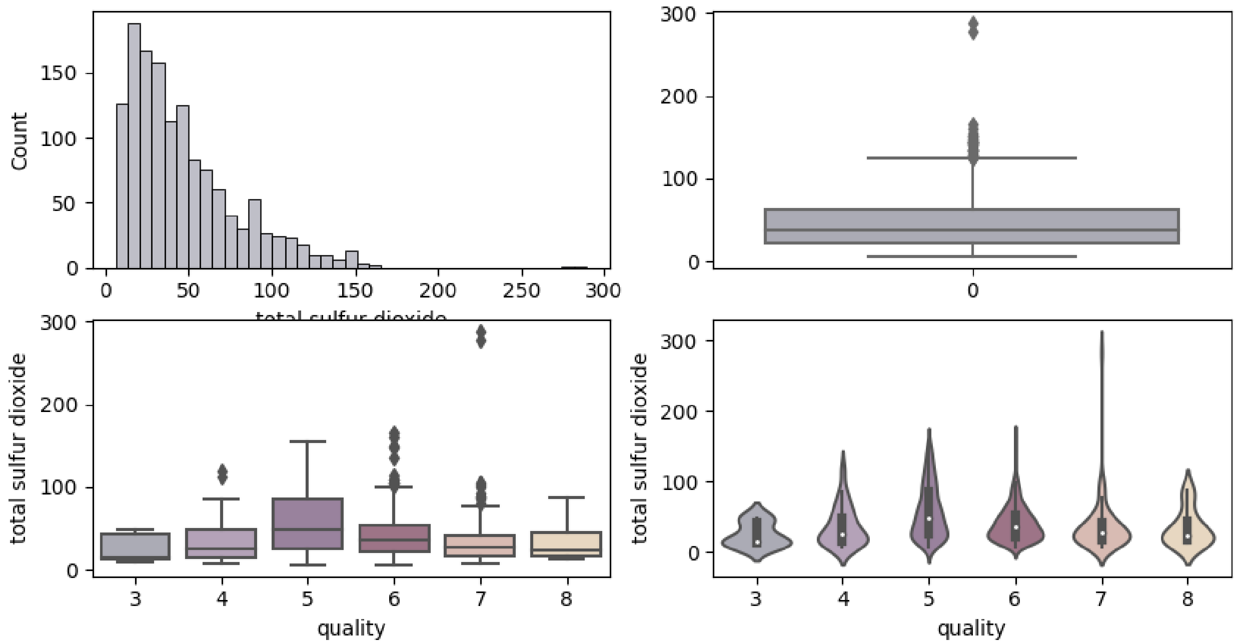

```
#parameter free sulfur dioxide
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.histplot(x=df['free sulfur dioxide'])
plt.subplot(2,2,2)
sns.boxplot(df['free sulfur dioxide'])
plt.subplot(2,2,3)
sns.boxplot(x=df['quality'],y=df['free sulfur dioxide'])
plt.subplot(2,2,4)
sns.violinplot(x=df['quality'],y=df['free sulfur dioxide'])
print('Correlation between free sulfur dioxide and quality
is',df['quality'].corr(df['free sulfur dioxide']))
print('Skewness of the free sulfur dioxide column is',df['free sulfur
dioxide'].skew())
```

```
Correlation between free sulfur dioxide and quality is -
0.05046276680502577
Skewness of the free sulfur dioxide column is 1.2265794991760643
```



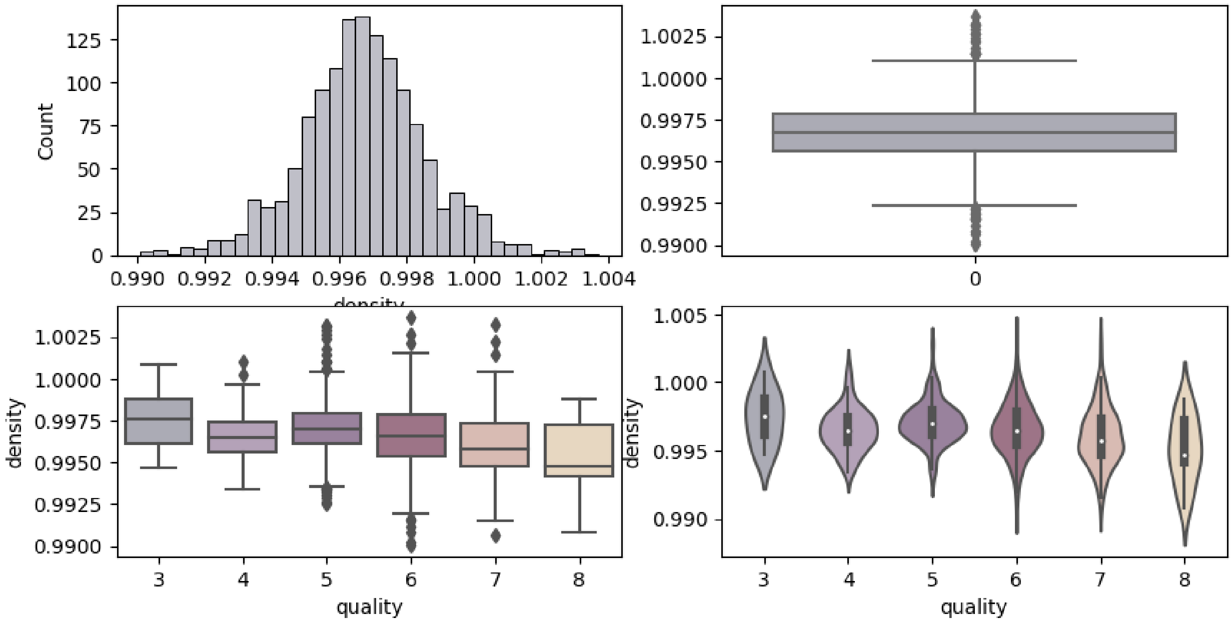

```
#parameter total sulfur dioxide
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.histplot(x=df['total sulfur dioxide'])
plt.subplot(2,2,2)
sns.boxplot(df['total sulfur dioxide'])
plt.subplot(2,2,3)
sns.boxplot(x=df['quality'],y=df['total sulfur dioxide'])
plt.subplot(2,2,4)
sns.violinplot(x=df['quality'],y=df['total sulfur dioxide'])
print('Correlation between total sulfur dioxide and quality
is',df['quality'].corr(df['total sulfur dioxide']))
print('Skewness of the total sulfur dioxide column is',df['total
sulfur dioxide'].skew())
```

```
Correlation between total sulfur dioxide and quality is -
0.177855365680296
Skewness of the total sulfur dioxide column is 1.5403680777213933
```



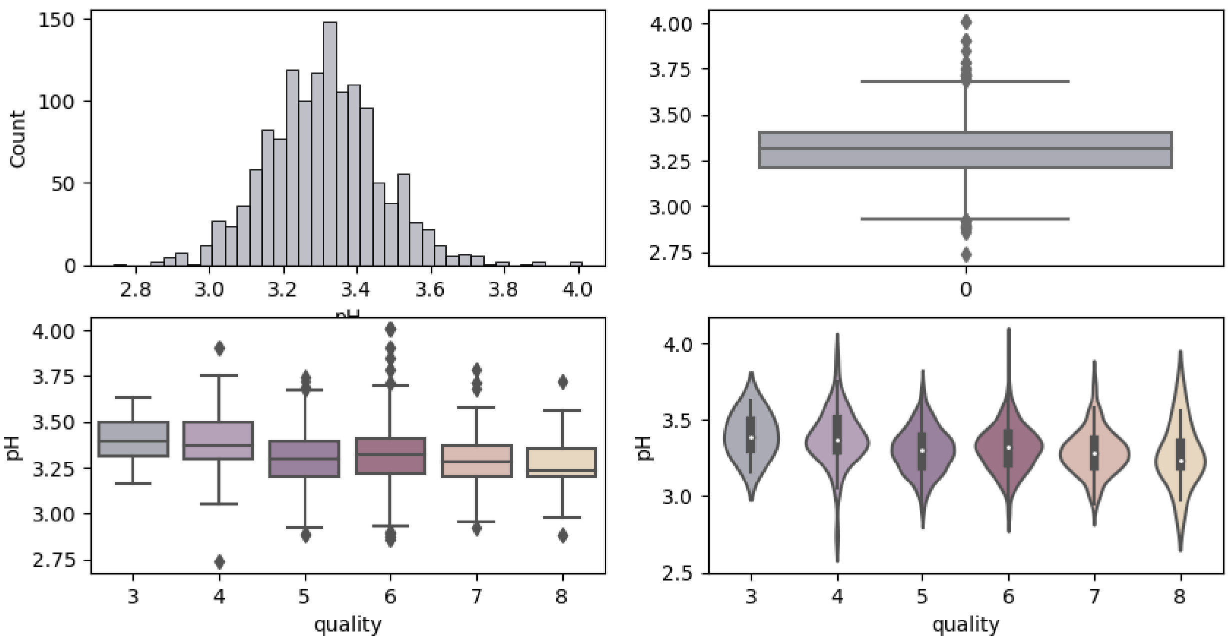

```python
#parameter density
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.histplot(x=df['density'])
plt.subplot(2,2,2)
sns.boxplot(df['density'])
plt.subplot(2,2,3)
sns.boxplot(x=df['quality'],y=df['density'])
plt.subplot(2,2,4)
sns.violinplot(x=df['quality'],y=df['density'])
print('Correlation between density and quality
is',df['quality'].corr(df['density']))
print('Skewness of the density column is',df['density'].skew())
```

```
Correlation between density and quality is -0.18425165011902406
Skewness of the density column is 0.04477785573116107
```



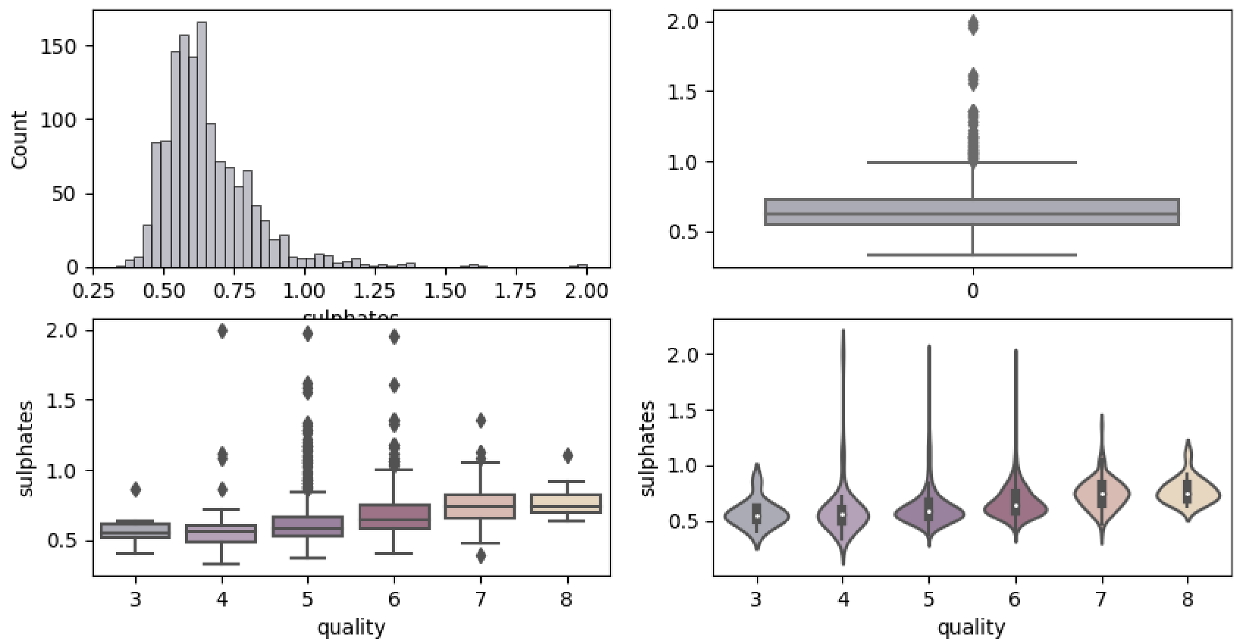

```
#parameter PH
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.histplot(x=df['pH'])
plt.subplot(2,2,2)
sns.boxplot(df['pH'])
plt.subplot(2,2,3)
sns.boxplot(x=df['quality'],y=df['pH'])
plt.subplot(2,2,4)
sns.violinplot(x=df['quality'],y=df['pH'])
print('Correlation between pH and quality
is',df['quality'].corr(df['pH']))
print('Skewness of the pH column is',df['pH'].skew())
```

```
Correlation between pH and quality is -0.05524511495867183
Skewness of the pH column is 0.2320322752014824
```



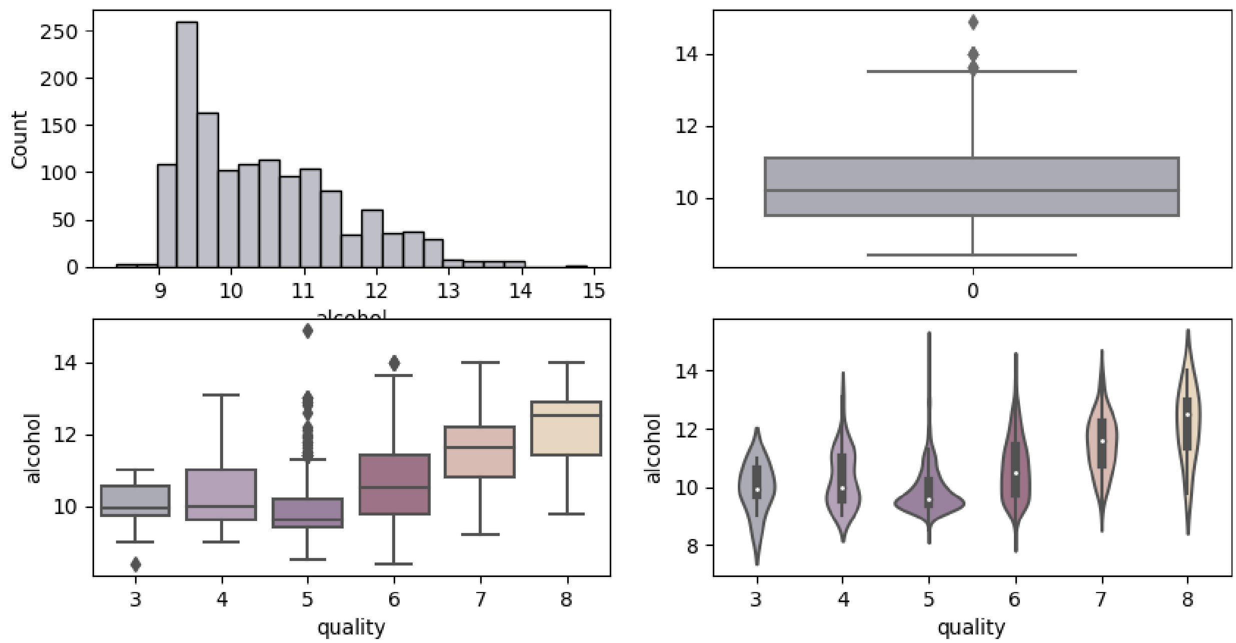

```
#parameter sulphates
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.histplot(x=df['sulphates'])
plt.subplot(2,2,2)
sns.boxplot(df['sulphates'])
plt.subplot(2,2,3)
sns.boxplot(x=df['quality'],y=df['sulphates'])
plt.subplot(2,2,4)
sns.violinplot(x=df['quality'],y=df['sulphates'])
print('Correlation between sulphates and quality
is',df['quality'].corr(df['sulphates']))
print('Skewness of the sulphates column is',df['sulphates'].skew())
```

```
Correlation between sulphates and quality is 0.2488351355778882
Skewness of the sulphates column is 2.4065046145674196
```



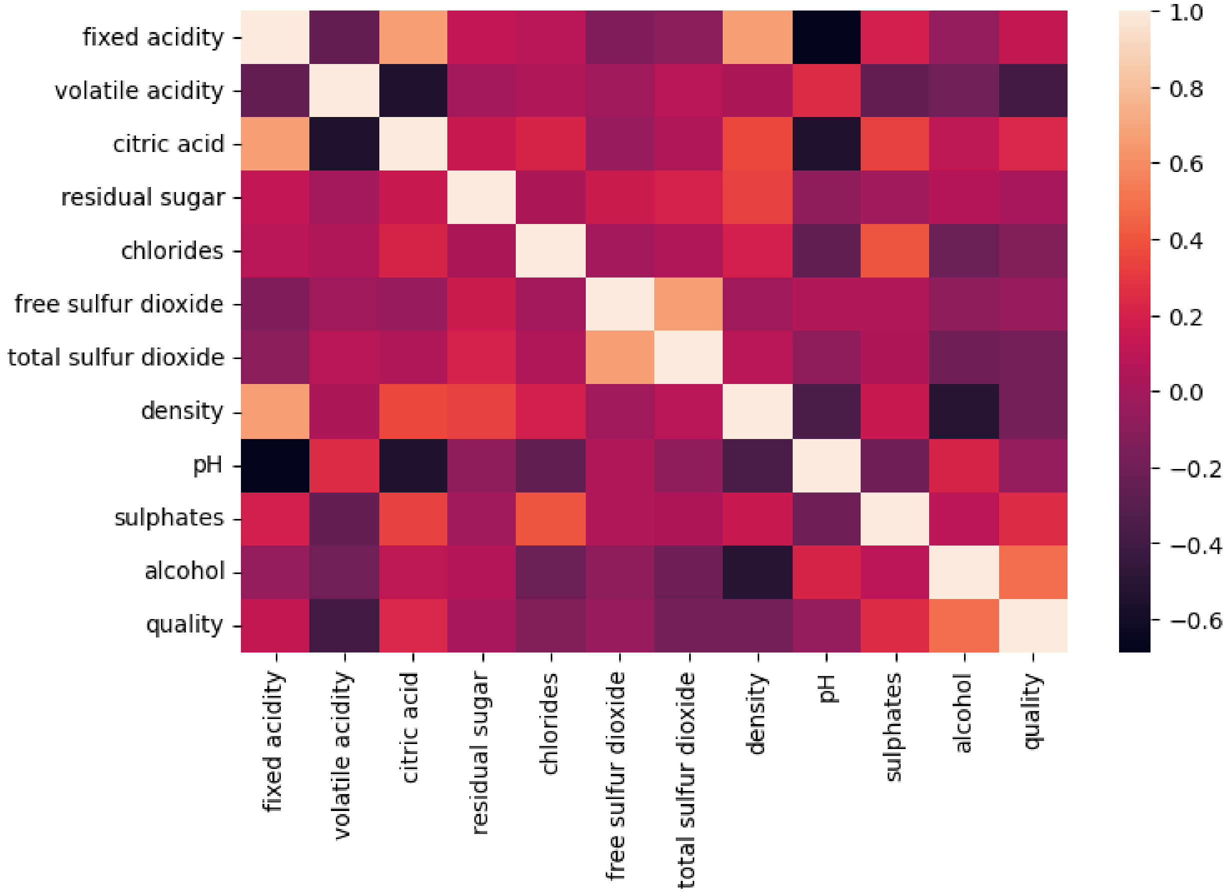

```
#parameter alcohol
plt.figure(figsize=(10,5))
plt.subplot(2,2,1)
sns.histplot(x=df['alcohol'])
plt.subplot(2,2,2)
sns.boxplot(df['alcohol'])
plt.subplot(2,2,3)
sns.boxplot(x=df['quality'],y=df['alcohol'])
plt.subplot(2,2,4)
sns.violinplot(x=df['quality'],y=df['alcohol'])
print('Correlation between alcohol and quality
is',df['quality'].corr(df['alcohol']))
print('Skewness of the alcohol column is',df['alcohol'].skew())
```

```
Correlation between alcohol and quality is 0.48034289800199176
Skewness of the alcohol column is 0.8598411692032926
```

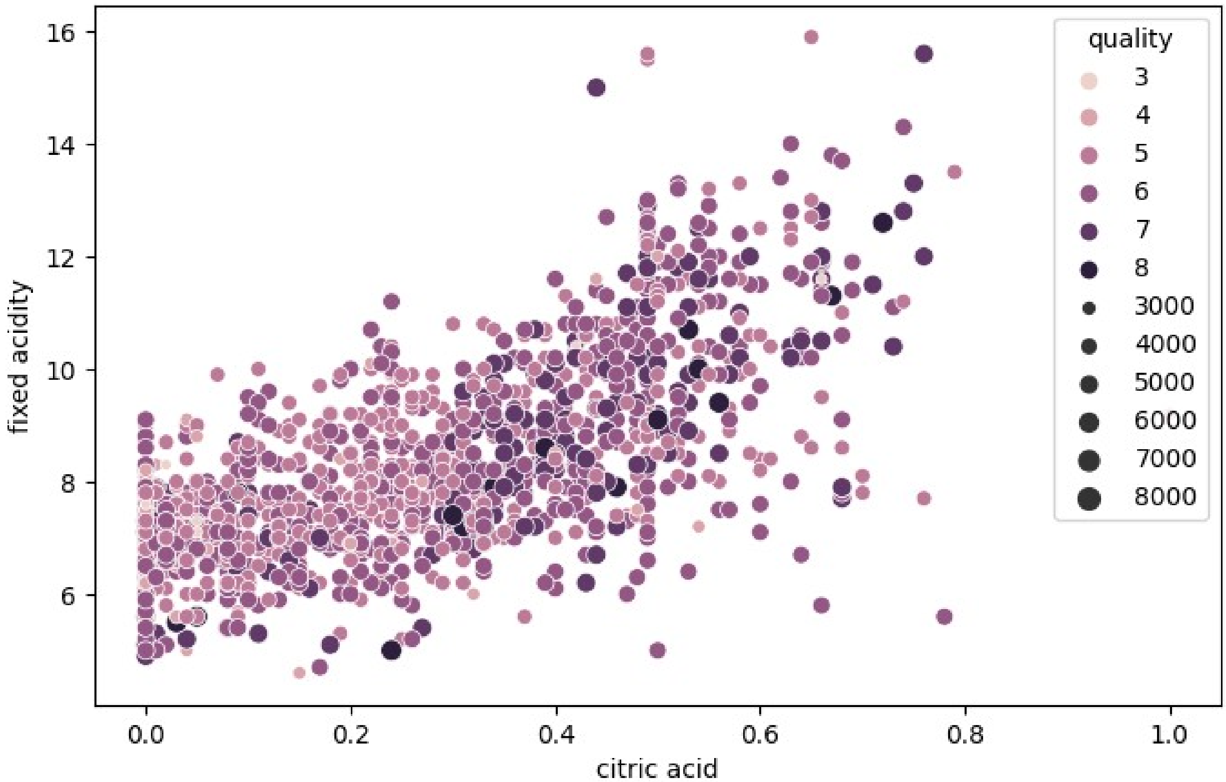



```
#ANALYST CONFUSION MATRIKS
plt.figure(figsize=(8,5))
sns.heatmap(df.corr())
```

```
<Axes: >
```

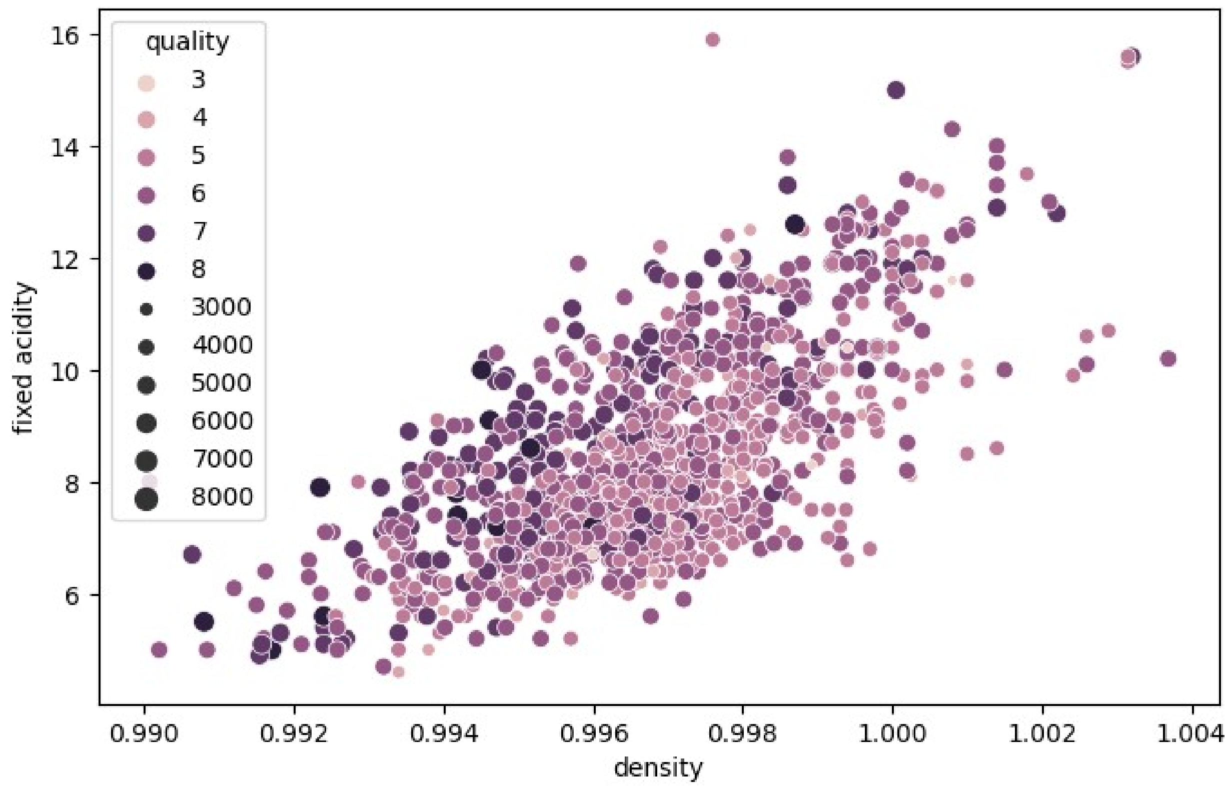



```
# from seaborn library for describe to connection x,y and know
relationship between these variables
plt.figure(figsize=(8,5))
sns.scatterplot(x=df['citric acid'],y=df['fixed
acidity'],hue=df['quality'],size=(df['quality']*1000))

<Axes: xlabel='citric acid', ylabel='fixed acidity'>
```




```
# from seaborn library for describe to connection x,y and know
relationship between these variables
plt.figure(figsize=(8,5))
sns.scatterplot(x=df['density'],y=df['fixed
acidity'],hue=df['quality'],size=df['quality']*1000)
```

```
<Axes: xlabel='density', ylabel='fixed acidity'>
```



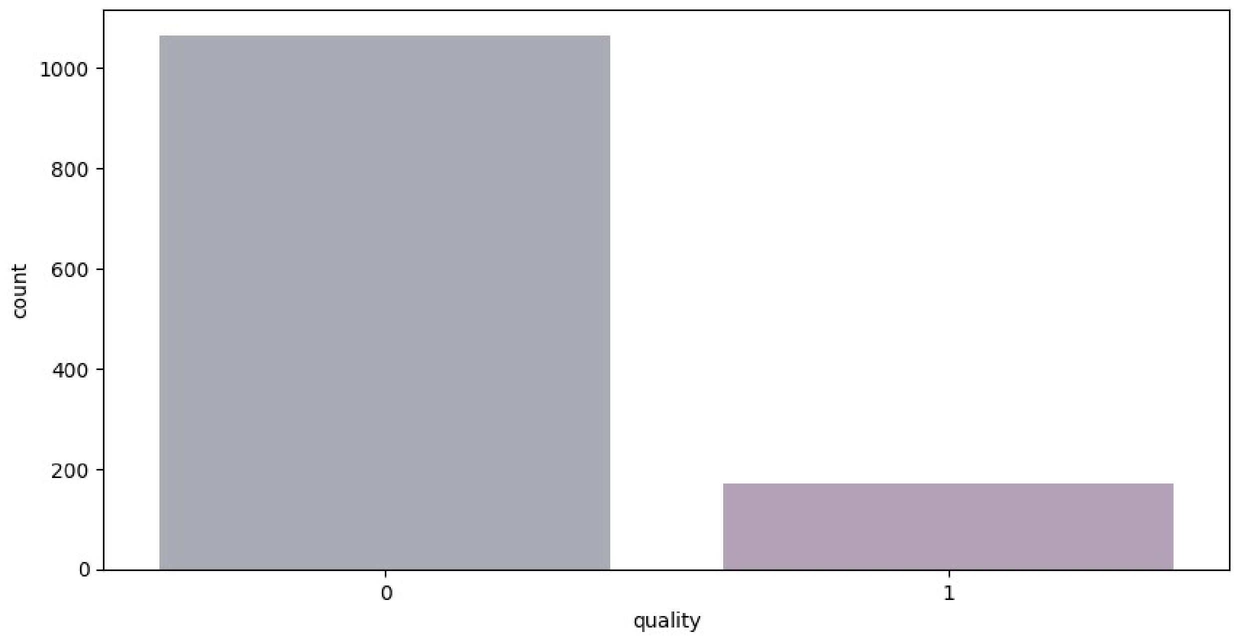

```
#take a dimensional from dataframe
df.shape
```

```
(1359, 12)
```

REMOVING OUTLIER

```
# address outliers that could impact statistical analysis.
from scipy import stats
z = np.abs(stats.zscore(df[df.dtypes[df.dtypes != 'object'].index]))
df = df[(z < 3).all(axis=1)]

# Values greater than or equal to 7 are considered good quality
# and are converted to 1, while values less than 7 are converted to 0.
X=df.drop('quality',axis=1)
y=df['quality']
y=df['quality'].apply(lambda y_value:1 if y_value>=7 else 0)

plt.figure(figsize=(10,5))
sns.countplot(x=y)
```

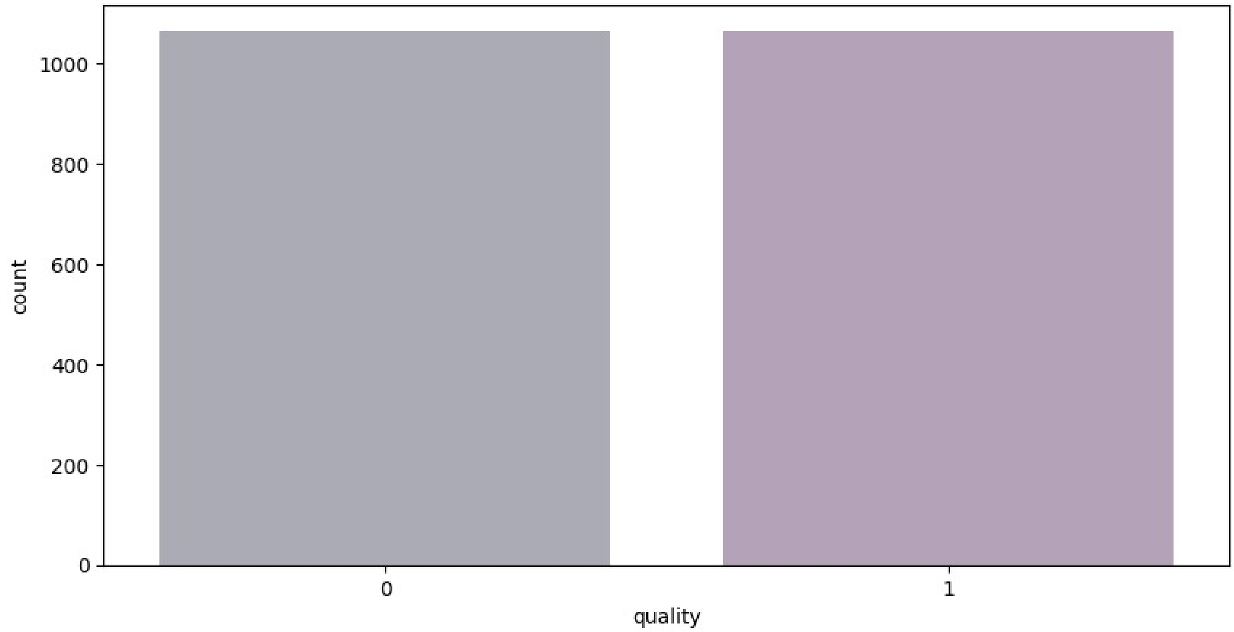
```
<Axes: xlabel='quality', ylabel='count'>
```



HANDLING IMBALANCE CLASESS

```
#addressing class imbalance in classification problems
from imblearn.over_sampling import SMOTE
smote=SMOTE()
x_smote,y_smote=smote.fit_resample(X,y)

plt.figure(figsize=(10,5))
sns.countplot(x=y_smote)
```

```
<Axes: xlabel='quality', ylabel='count'>
```



TRAIN TEST SPLIT DATA

```
#for split data -> with testing is 30% for dataset
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(x_smote,y_smote,test_si
ze=0.3,random_state=42)
```

LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
lr_model=LogisticRegression(solver='newton-cg')

lr_model.fit(X_train,y_train)

LogisticRegression(solver='newton-cg')

lr_model.score(X_test,y_test)

0.8479623824451411

y_lr_pred=lr_model.predict(X_test)

from sklearn.metrics import classification_report
print(classification_report(y_test,y_lr_pred))

              precision    recall  f1-score   support

           0       0.87      0.82      0.84       321
           1       0.83      0.87      0.85       317
```

```
    accuracy                           0.85       638
   macro avg       0.85      0.85      0.85       638
weighted avg       0.85      0.85      0.85       638
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,y_lr_pred))
```

```
[[264  57]
 [ 40 277]]
```

DECISION TREE

```
from sklearn.tree import DecisionTreeClassifier
dtr=DecisionTreeClassifier()

parameters = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    'splitter': ['best', 'random'],
    'max_depth': [2, 4, 6, 8],
    'min_samples_split': [2, 4, 6, 8],
    'min_samples_leaf': [1, 2, 3, 4],
    'max_features': ['sqrt'],  # Mengatur max_features menjadi 'sqrt'
}

from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(dtr, parameters, cv=5, scoring='f1_macro')
grid_search.fit(X_train, y_train)

print('The Parameters are:', grid_search.best_params_)
```

```
The Parameters are: {'criterion': 'log_loss', 'max_depth': 8,
'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 4,
'splitter': 'best'}
```

```
dtr=DecisionTreeClassifier(criterion =
grid_search.best_params_.get('criterion'),
                                  splitter =
grid_search.best_params_.get('splitter'),
                                  max_depth =
grid_search.best_params_.get('max_depth'),
                                  max_features =
grid_search.best_params_.get('max_features'),
                                  min_samples_leaf =
grid_search.best_params_.get('min_samples_leaf'),
                                  min_samples_split =
grid_search.best_params_.get('min_samples_split'),
                                  random_state = 42)
```

```
dtr.fit(X_train,y_train)

DecisionTreeClassifier(criterion='log_loss', max_depth=8,
max_features='sqrt',
                       min_samples_leaf=4, min_samples_split=4,
                       random_state=42)

y_pred=dtr.predict(X_test)

from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))

              precision    recall  f1-score   support

           0       0.84      0.83      0.84       321
           1       0.83      0.84      0.84       317

    accuracy                           0.84       638
   macro avg       0.84      0.84      0.84       638
weighted avg       0.84      0.84      0.84       638


from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,y_pred))

[[266  55]
 [ 50 267]]
```

RANDOM FOREST

```
from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()

rfc.fit(X_train,y_train)

RandomForestClassifier()

y_rfc_pred=rfc.predict(X_test)

print(classification_report(y_test,y_rfc_pred))

              precision    recall  f1-score   support

           0       0.92      0.88      0.90       321
           1       0.88      0.92      0.90       317

    accuracy                           0.90       638
   macro avg       0.90      0.90      0.90       638
weighted avg       0.90      0.90      0.90       638
```

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,y_rfc_pred))

[[282  39]
 [ 24 293]]
```

ADABOOST CLASSIFIER

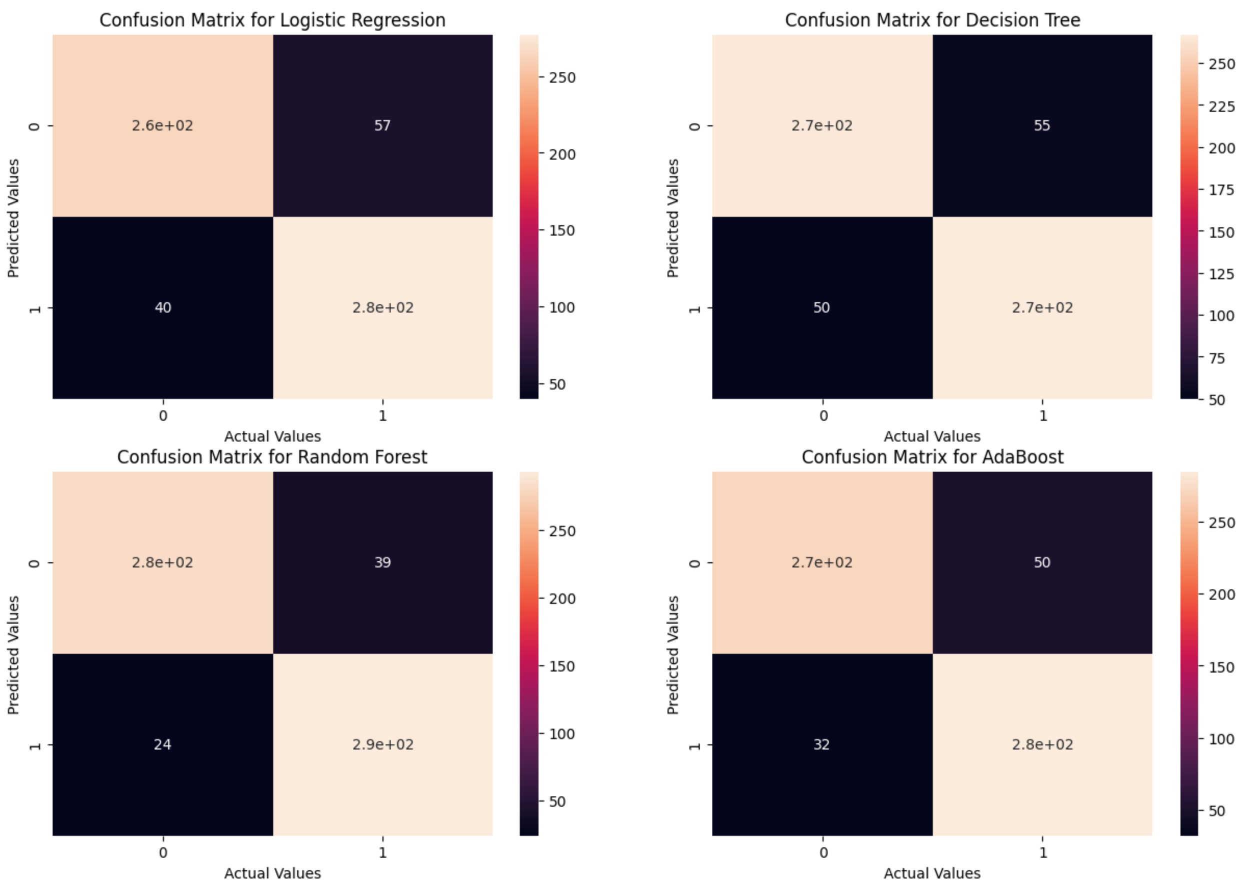
```
from sklearn.ensemble import AdaBoostClassifier
abc=AdaBoostClassifier()

abc.fit(X_train,y_train)

AdaBoostClassifier()

y_abc_pred=abc.predict(X_test)

print(classification_report(y_test,y_abc_pred))

              precision    recall  f1-score   support

           0       0.89      0.84      0.87       321
           1       0.85      0.90      0.87       317

    accuracy                           0.87       638
   macro avg       0.87      0.87      0.87       638
weighted avg       0.87      0.87      0.87       638


from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test,y_abc_pred))

[[271  50]
 [ 32 285]]
```

MODEL ANALYST AND EVALUATION

```
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.heatmap(confusion_matrix(y_test,y_lr_pred),annot=True)
plt.ylabel('Predicted Values')
plt.xlabel('Actual Values')
plt.title('Confusion Matrix for Logistic Regression')
plt.subplot(2,2,2)
sns.heatmap(confusion_matrix(y_test,y_pred),annot=True)
plt.ylabel('Predicted Values')
plt.xlabel('Actual Values')
plt.title('Confusion Matrix for Decision Tree')
```

```
plt.subplot(2,2,3)
sns.heatmap(confusion_matrix(y_test,y_rfc_pred),annot=True)
plt.ylabel('Predicted Values')
plt.xlabel('Actual Values')
plt.title('Confusion Matrix for Random Forest')
plt.subplot(2,2,4)
sns.heatmap(confusion_matrix(y_test,y_abc_pred),annot=True)
plt.ylabel('Predicted Values')
plt.xlabel('Actual Values')
plt.title('Confusion Matrix for AdaBoost')
```

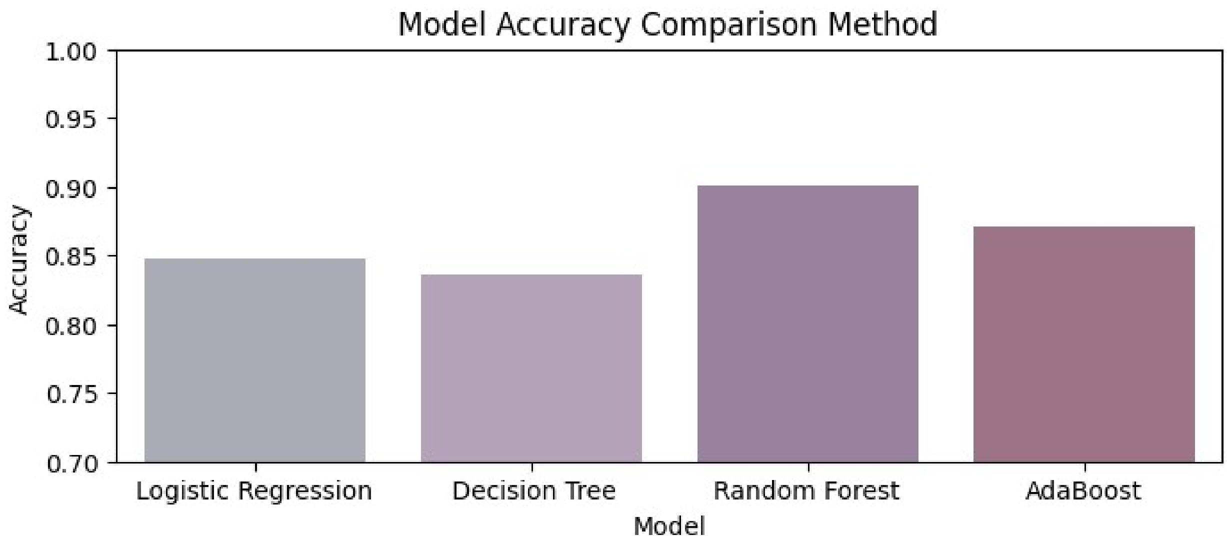
```
Text(0.5, 1.0, 'Confusion Matrix for AdaBoost')
```



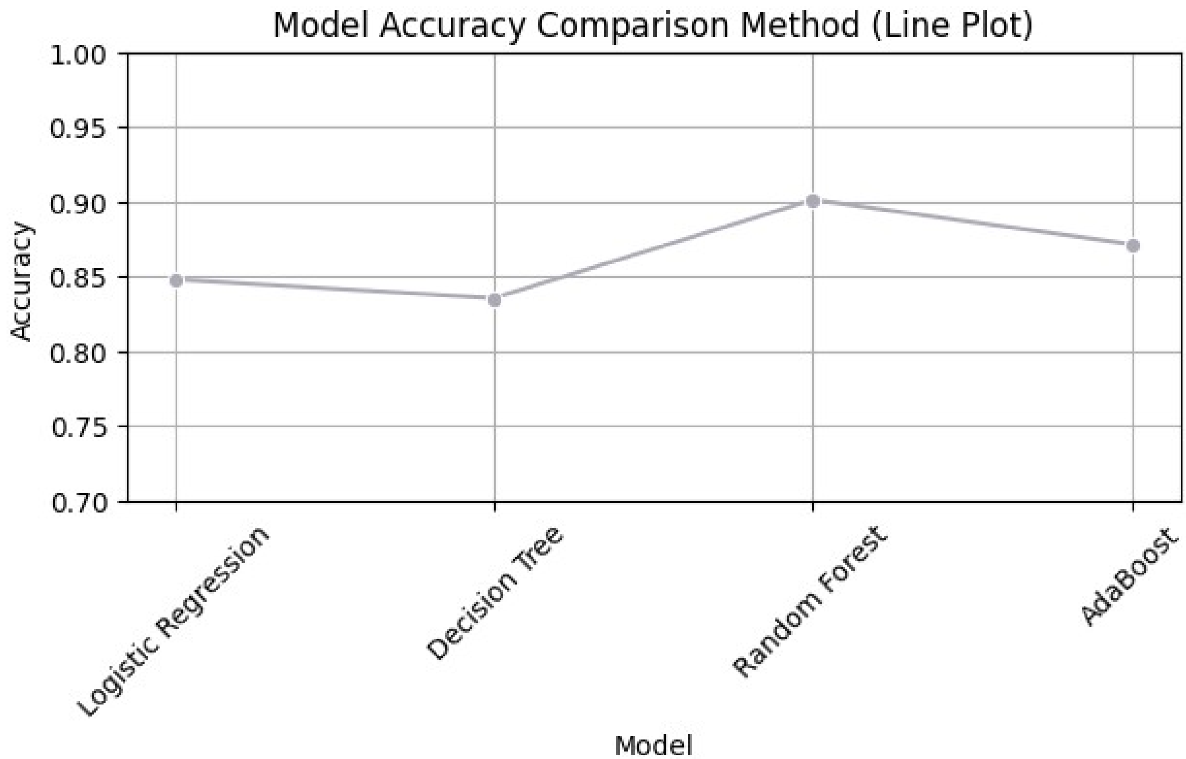

```
from sklearn.metrics import accuracy_score
models = ['Logistic Regression', ' Decision Tree', 'Random Forest', 
'AdaBoost']
accuracy = [accuracy_score(y_test, y_lr_pred), accuracy_score(y_test, 
y_pred), accuracy_score(y_test, y_rfc_pred), accuracy_score(y_test, 
y_abc_pred)]
plt.figure(figsize=(8,3))
sns.barplot(x=models, y=accuracy)
plt.title('Model Accuracy Comparison Method')
```

```
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0.7, 1.0)
```

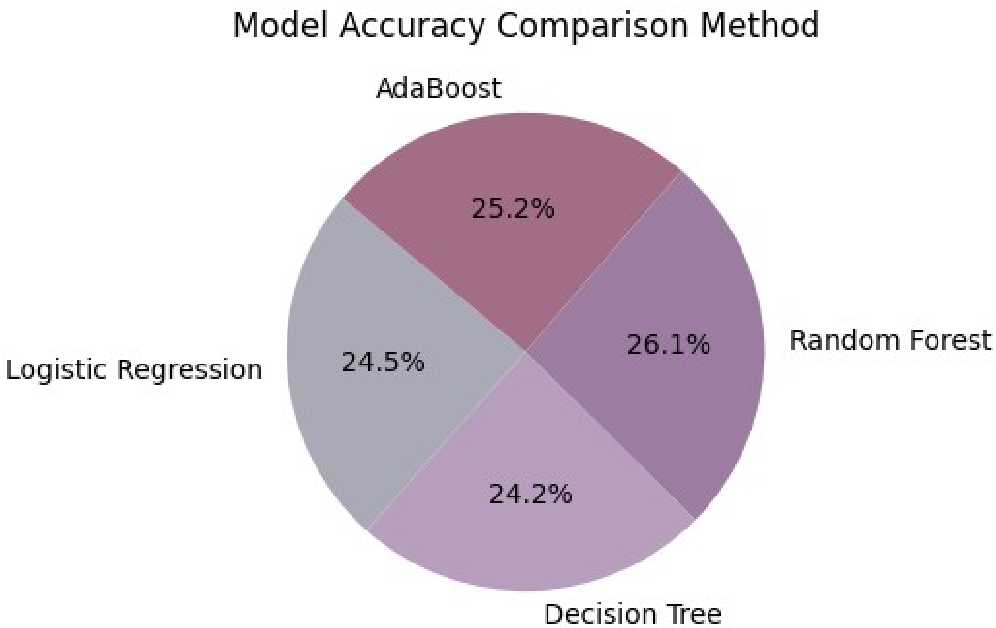
```
(0.7, 1.0)
```




```
#Random Forest is getting us the maximum accuracy i.e. 90 %.

plt.figure(figsize=(7, 3))
sns.lineplot(x=models, y=accuracy, marker="o", linestyle="-")
plt.title('Model Accuracy Comparison Method (Line Plot)')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0.7, 1.0)
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```




```
#Random Forest is getting us the maximum accuracy i.e. 90 %.

plt.figure(figsize=(4,4))
plt.pie(accuracy, labels=models, autopct='%1.1f%%', startangle=140)
plt.title('Model Accuracy Comparison Method')
plt.show()
```




```
#Random Forest provides the most optimal classification percentage
level, namely 26.1%.
```