🥂 🍷 WINE 🍷 🥂

About Dataset

'Wine'

This data frame contains the following columns:

Input variables (based on physicochemical tests):

- 1 - fixed acidity
- 2 - volatile acidity
- 3 - citric acid
- 4 - residual sugar
- 5 - chlorides
- 6 - free sulfur dioxide
- 7 - total sulfur dioxide
- 8 - density
- 9 - pH
- 10 - sulphates
- 11 - alcohol
- 12 - quality (score between 0 and 10)

🎯 Outline of the Project 🎯

1. 🗂️Importing Libraries: - To perform Data Manipulation,Visualization & Model Building.

2. ⏳ Loading Dataset: - Load the dataset into a suitable data structure using pandas.

3. 📑 Exploration of Dataset: - Generate basic informations about the data.

4. 🧽 Data Cleaning: - garbage cleaning, removing duplicate, and hadling missing values and treating outliers

5. 📊 Exploatory Data Analysis: - To identify trends, patterns, and relationships among the variabels.

6. ⚙️ Data Preprocessing: - To transform data for creating more accurate & robust model.

7. 🧱Model building:- To build predictive models, using various algorithms.

8. 🔎Model evaluation: - To analyze the Model performance using metrics.

9. ⚖️ Model comparision: - overall model performance

🗂️ Importing Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

sns.set()
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

⏳ Loading Dataset

In [2]:

```
dataset = 'winequality-red.csv'
df = pd.read_csv(dataset)
df.head()
```

Out[2]:

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

⏳ Exoloring Dataset

In [3]:

```
df.shape
```

Out[3]:

```
(1599, 12)
```

In [4]:

```
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
```

In [ ]:

```
from summarytools import dfSummary
dfSummary(df)
```

In [5]:

```
```

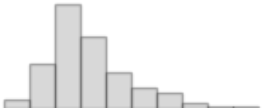
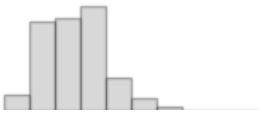
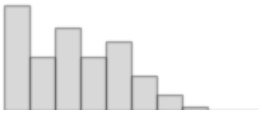
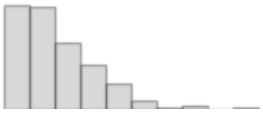
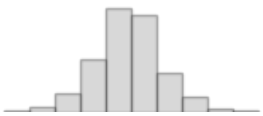
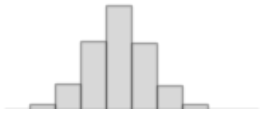
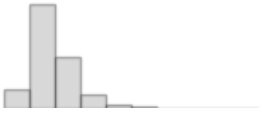
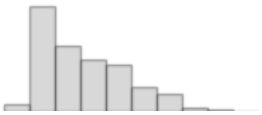
Out[5]:

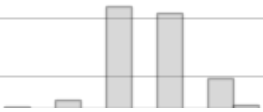
Data Frame Summary

df

Dimensions: 1,599 x 12

Duplicates: 240

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
1	**fixed acidity** [float64]	Mean (sd) : 8.3 (1.7) min < med < max: 4.6 < 7.9 < 15.9 IQR (CV) : 2.1 (4.8)	96 distinct values		0 (0.0%)
2	**volatile acidity** [float64]	Mean (sd) : 0.5 (0.2) min < med < max: 0.1 < 0.5 < 1.6 IQR (CV) : 0.2 (2.9)	143 distinct values		0 (0.0%)
3	**citric acid** [float64]	Mean (sd) : 0.3 (0.2) min < med < max: 0.0 < 0.3 < 1.0 IQR (CV) : 0.3 (1.4)	80 distinct values		0 (0.0%)
4	**residual sugar** [float64]	Mean (sd) : 2.5 (1.4) min < med < max: 0.9 < 2.2 < 15.5 IQR (CV) : 0.7 (1.8)	91 distinct values		0 (0.0%)
5	**chlorides** [float64]	Mean (sd) : 0.1 (0.0) min < med < max: 0.0 < 0.1 < 0.6 IQR (CV) : 0.0 (1.9)	153 distinct values		0 (0.0%)
6	**free sulfur dioxide** [float64]	Mean (sd) : 15.9 (10.5) min < med < max: 1.0 < 14.0 < 72.0 IQR (CV) : 14.0 (1.5)	60 distinct values		0 (0.0%)
7	**total sulfur dioxide** [float64]	Mean (sd) : 46.5 (32.9) min < med < max: 6.0 < 38.0 < 289.0 IQR (CV) : 40.0 (1.4)	144 distinct values		0 (0.0%)
8	**density** [float64]	Mean (sd) : 1.0 (0.0) min < med < max: 1.0 < 1.0 < 1.0 IQR (CV) : 0.0 (528.1)	436 distinct values		0 (0.0%)
9	**pH** [float64]	Mean (sd) : 3.3 (0.2) min < med < max: 2.7 < 3.3 < 4.0 IQR (CV) : 0.2 (21.4)	89 distinct values		0 (0.0%)
10	**sulphates** [float64]	Mean (sd) : 0.7 (0.2) min < med < max: 0.3 < 0.6 < 2.0 IQR (CV) : 0.2 (3.9)	96 distinct values		0 (0.0%)
11	**alcohol** [float64]	Mean (sd) : 10.4 (1.1) min < med < max: 8.4 < 10.2 < 14.9 IQR (CV) : 1.6 (9.8)	65 distinct values		0 (0.0%)

No	Variable	Stats / Values	Freqs / (% of Valid)	Graph	Missing
12	**quality** [int64]	Mean (sd) : 5.6 (0.8) min < med < max: 3.0 < 6.0 < 8.0 IQR (CV) : 1.0 (7.0)	6 distinct values		0 (0.0%)

In [ ]:

In [ ]:

In [ ]:

Cleaning of dataset

Checking and Removing Duplicates

In [6]:

```
# Remove duplicates

def drop_dup(df):
    if df.duplicated().any() == True:
        print('The total duplicate row before removing duplicate:', df.duplicated().sum(
        df.drop_duplicates(inplace=True , keep = 'last')  # Remove duplicates
        df = df.reset_index(drop=True) #Reset the index
        print('The total duplicate row after removing duplicate:', df.duplicated().sum()
    else:
        return 'No duplicate entries'
drop_dup(df)
```

```
The total duplicate row before removing duplicate: 240
The total duplicate row after removing duplicate: 0
shape of dataset after removing duplicate columns : (1359, 12)
```

Checking null values

In [7]:

```
df.isnull().sum()
```

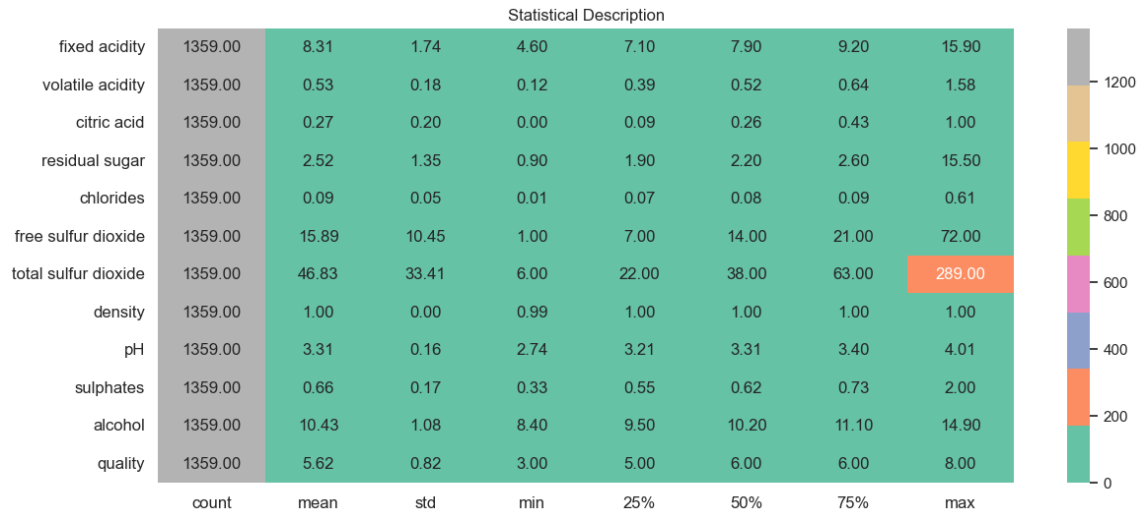
Out[7]:

```
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
```

📊 Descriptive Statistics

In [8]:

```
desc=df.describe().T
def descriptive_stats(df):


    plt.figure(figsize=(14,6))
    sns.heatmap(df, annot=True, cmap='Set2', fmt=".2f")
    plt.xticks(size = 12)
    plt.yticks(size = 12, rotation = 0)
    plt.title('Statistical Description')
    plt.show()


descriptive_stats(desc)
```

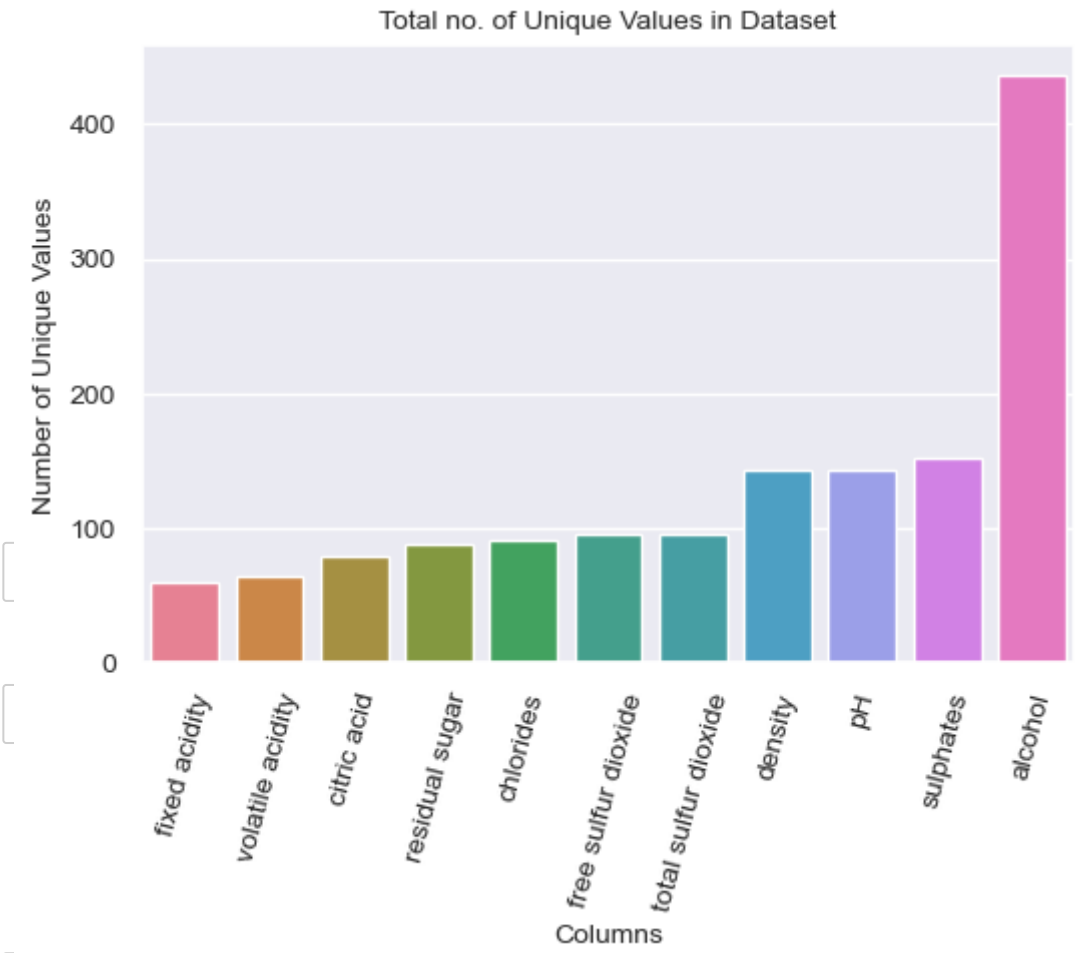


Checking Unique values of different features

In [9]:

```
# Count the number of unique values in each column
def check_unquie_count(df):
    unique_counts = df.nunique()
    print('=='*30)
    print('  '*10, 'Total no. of Unique Values')
    print('=='*30)
    print(unique_counts.sort_values)
    print('=='*30)
# Create a bar plot or count plot of unique values
    #plt.style.use('dark_background')
    plt.figure(figsize=(6, 4))
    sns.barplot(x=unique_counts.index, y=unique_counts.sort_values(),palette='husl' )

    plt.xticks(rotation=75, fontsize= 10)
    plt.yticks( fontsize= 10 )
    plt.xlabel('Columns',fontsize=10)
    plt.ylabel('Number of Unique Values', fontsize=10)
    plt.title('Total no. of Unique Values in Dataset', fontsize=10)

# Display the plot
    plt.show()

check_unquie_count(df.iloc[:,0:-1])
```

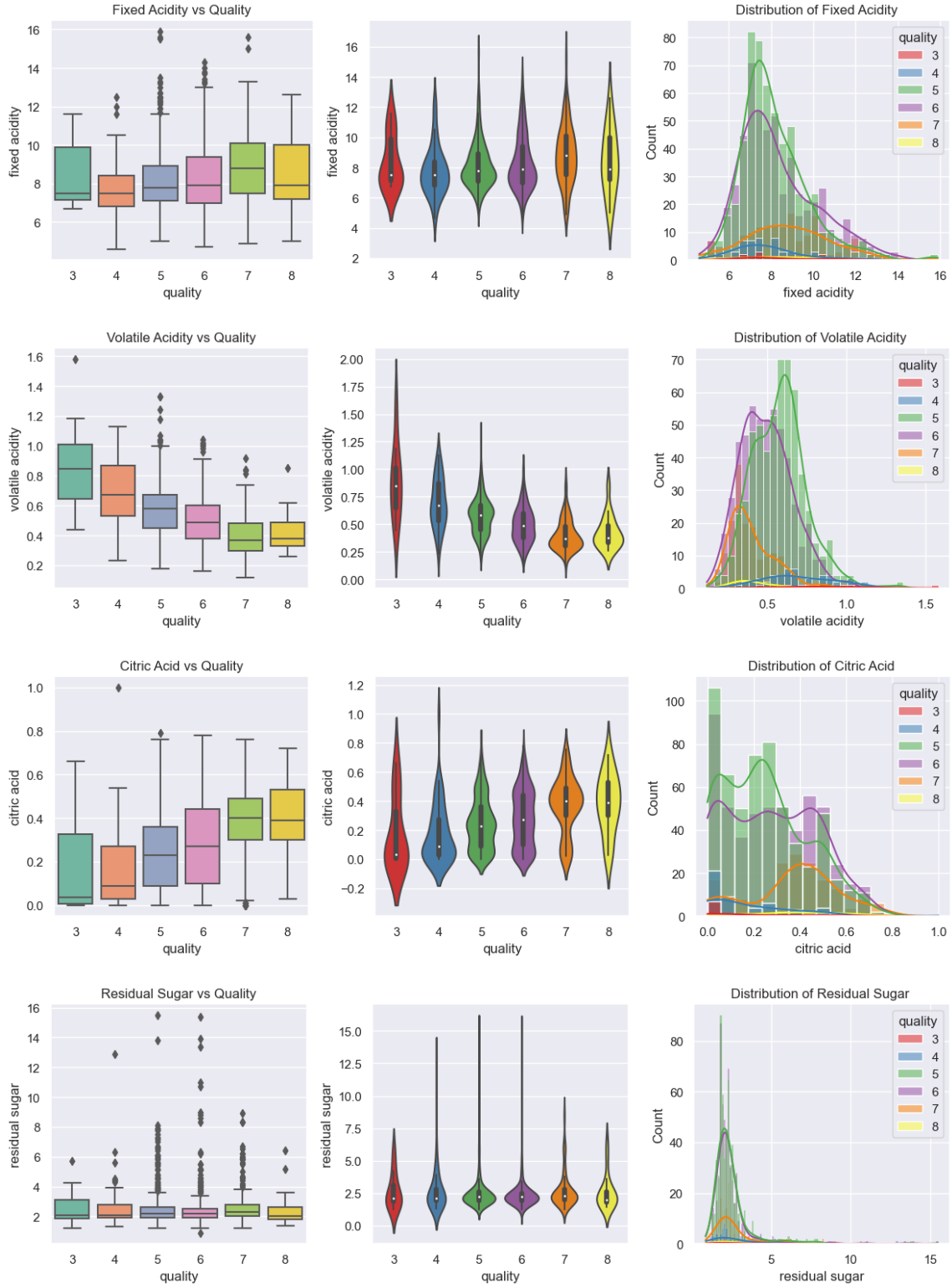
```
============================================================
                    Total no. of Unique Values
============================================================
<bound method Series.sort_values of fixed acidity            96
volatile acidity        143
citric acid              80
residual sugar           91
chlorides               153
free sulfur dioxide      60
total sulfur dioxide    144
density                 436
pH                       89
sulphates                96
alcohol                  65
dtype: int64>
============================================================
```

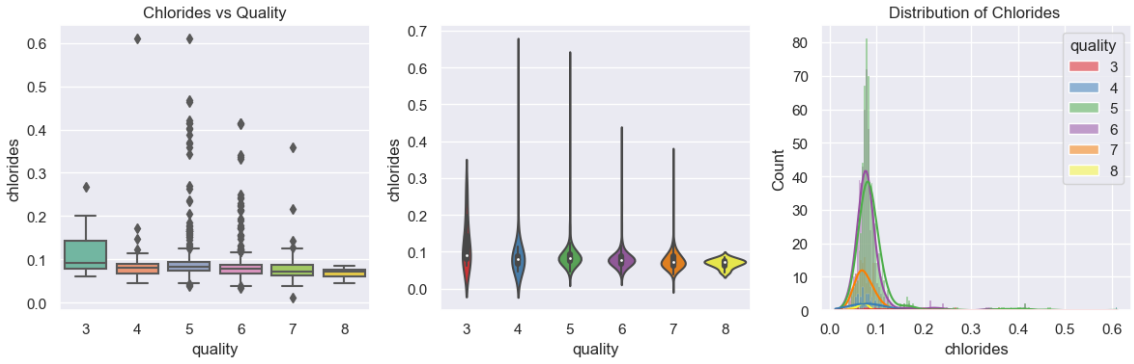


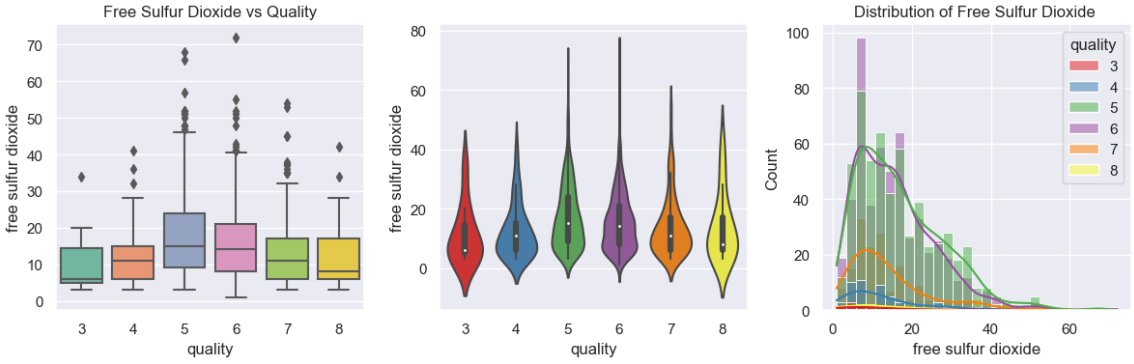

```
def num_dist(column, target= 'quality'):
    plt.figure(figsize=(12,4))
    plt.subplot(1,3,1)
    sns.boxplot(x=target,y=column, data=df, palette="Set2")
    plt.title(f"{column.title()} vs Quality")

    plt.subplot(1,3,2)
    sns.violinplot(x=target, y=column, data=df, palette='Set1',)

    plt.subplot(1,3,3)
    sns.histplot(data=df,x=column,kde=True,hue= target, palette="Set1")
    plt.title(f"Distribution of {column.title()} ")
    plt.tight_layout()
    plt.show()
```

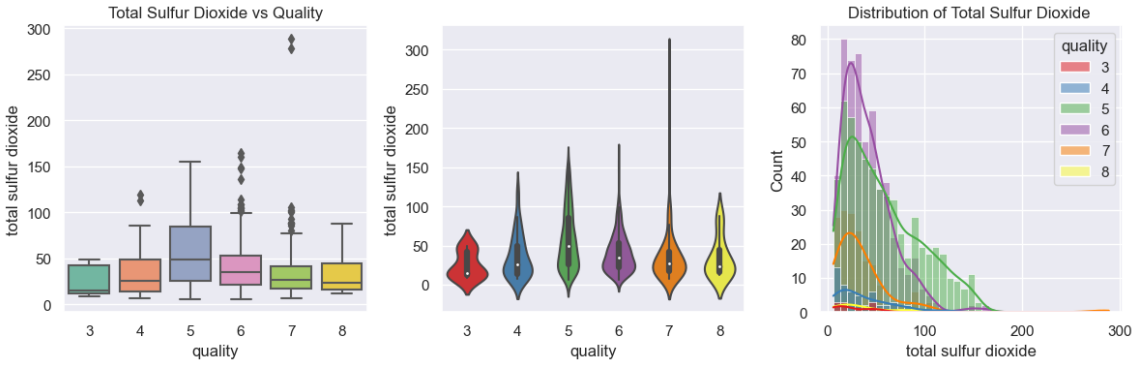
In [11]:

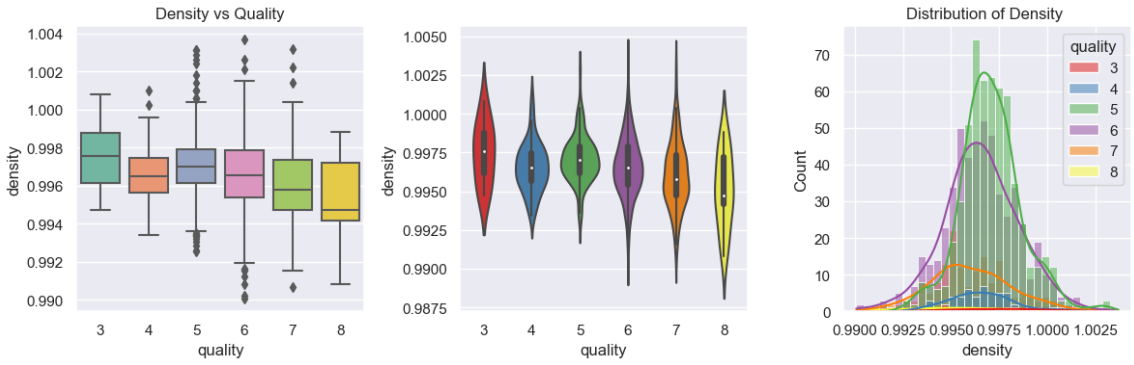
```
for col in df.iloc[:,:-1].columns:
    num_dist(col)
```

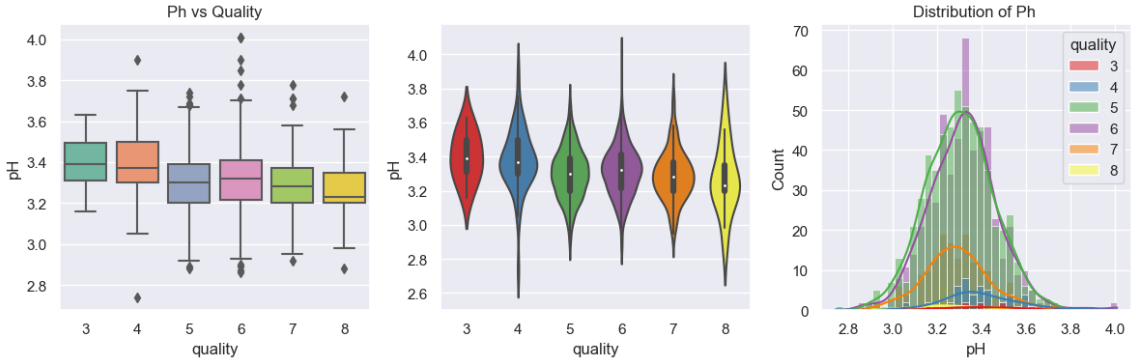


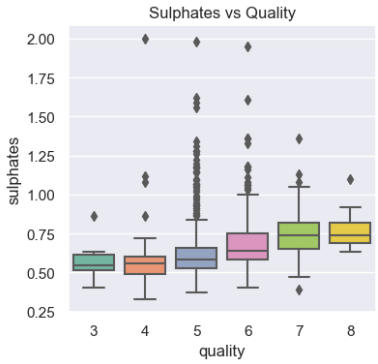

Chlorides vs Quality
chlorides
quality
Distribution of Chlorides
Count
chlorides
quality
3
4
5
6
7
8

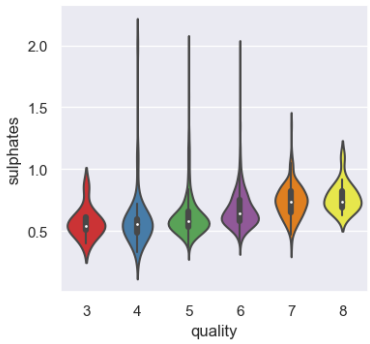

Free Sulfur Dioxide vs Quality
free sulfur dioxide
quality
Distribution of Free Sulfur Dioxide
Count
free sulfur dioxide
quality
3
4
5
6
7
8

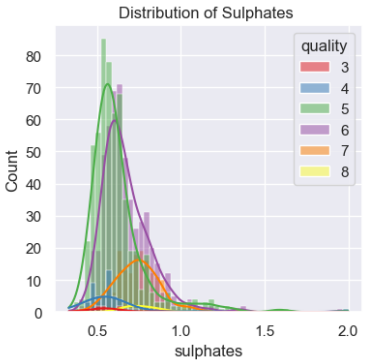

Total Sulfur Dioxide vs Quality
total sulfur dioxide
quality
Distribution of Total Sulfur Dioxide
Count
total sulfur dioxide
quality
3
4
5
6
7
8

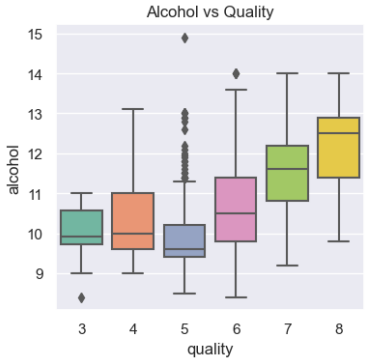

Density vs Quality
density
quality
Distribution of Density
Count
density
quality
3
4
5
6
7
8

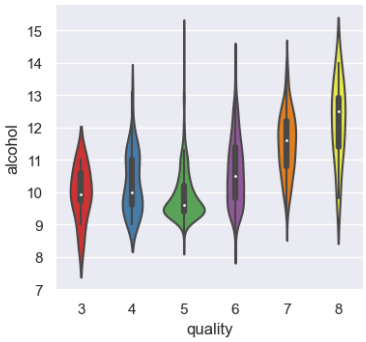

Ph vs Quality
pH
quality
Distribution of Ph
Count
pH
quality
3
4
5
6
7
8

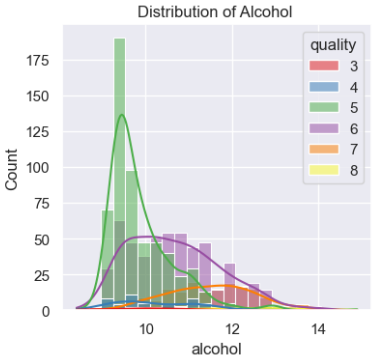

Sulphates vs Quality
sulphates
quality


sulphates
quality


Distribution of Sulphates
quality
3
4
5
6
7
8
Count
sulphates


Alcohol vs Quality
alcohol
quality

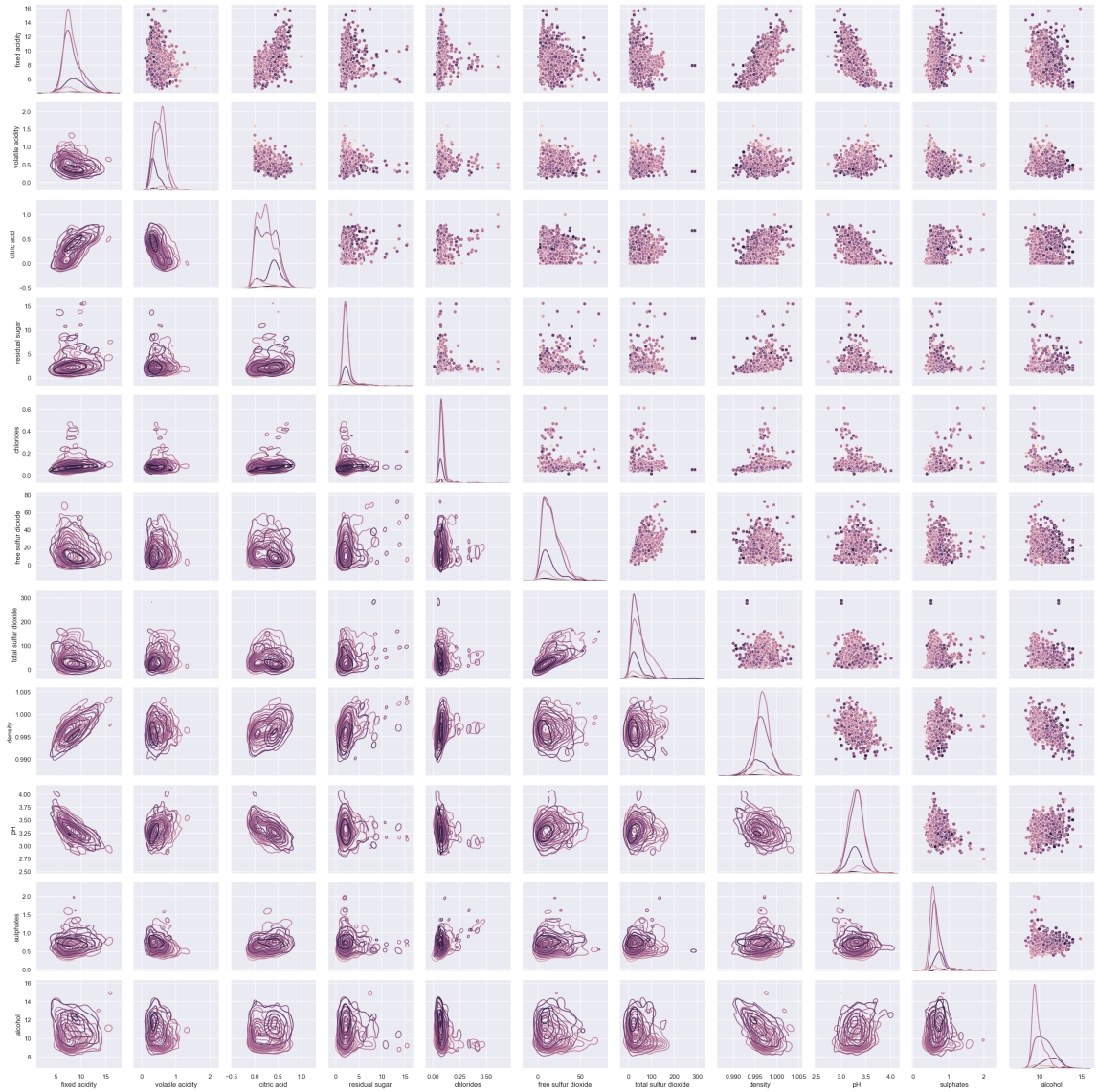

alcohol
quality


Distribution of Alcohol
quality
3
4
5
6
7
8
Count
alcohol

Bivariate Analysis

In [12]:

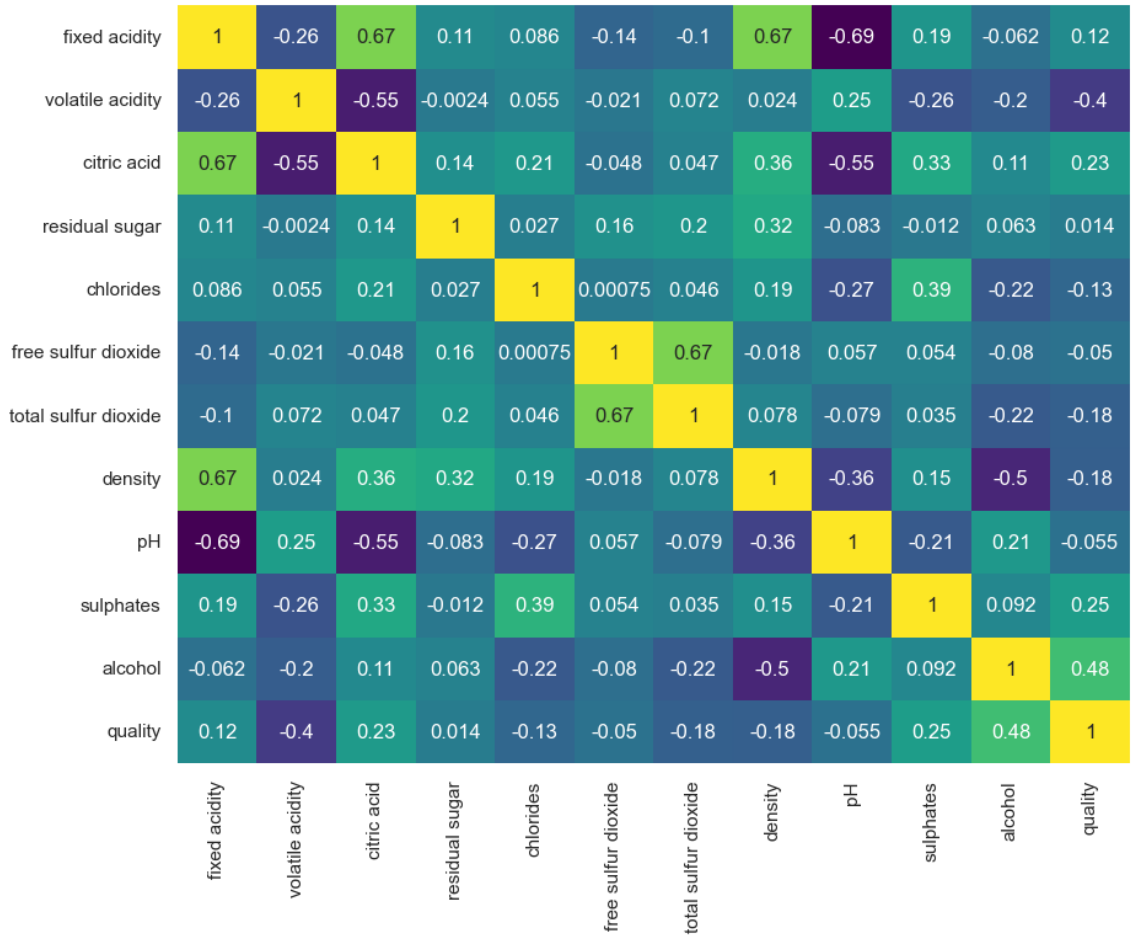
```
g = sns.PairGrid(df, hue='quality' , diag_sharey=False)
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot)
g.map_diag(sns.kdeplot)
plt.show()
```



Finding correlation

In [13]:

```
plt.figure(figsize=(10,8))
corr = df.corr()
sns.heatmap(corr, annot=True, cmap='viridis',cbar=False)
plt.show()
```



In [14]:

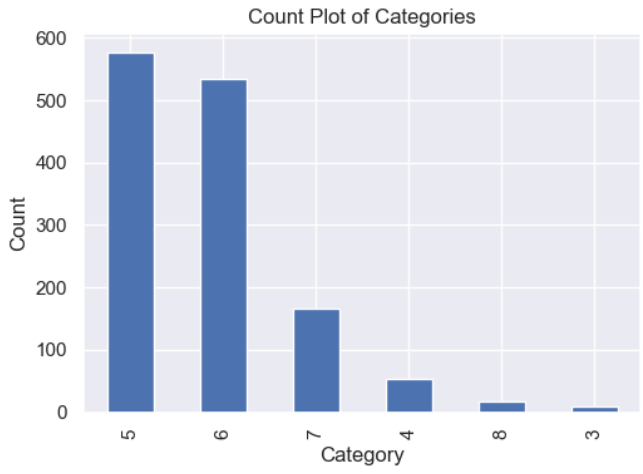
```
df.drop('total sulfur dioxide', axis=1, inplace=True)
```

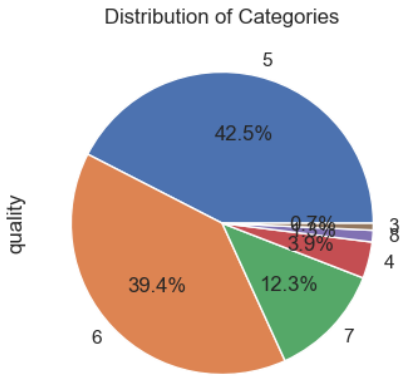
📊 Checking Imbalance between the different classes

In [15]:

```
value_counts =df.quality.value_counts()
print(value_counts)


fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# Plot the bar plot on the first subplot (axes[0])
value_counts.plot(kind='bar', ax=axes[0],)
axes[0].set_xlabel('Category')
axes[0].set_ylabel('Count')
axes[0].set_title('Count Plot of Categories')


value_counts.plot(
    kind='pie',
    ax=axes[1],
    #explode=[0.1, 0.1,0,0, 0, 0],     # Explode the second category to emphasize it
    autopct='%1.1f%%',         # Display percentage with one decimal place
    #shadow=True,               # Add a shadow effect to the chart
    #colors=['cyan', 'gray', 'orange', 'green']  # Custom colors for the pie chart
)
axes[1].set_title('Distribution of Categories')


plt.tight_layout()

plt.show()
```

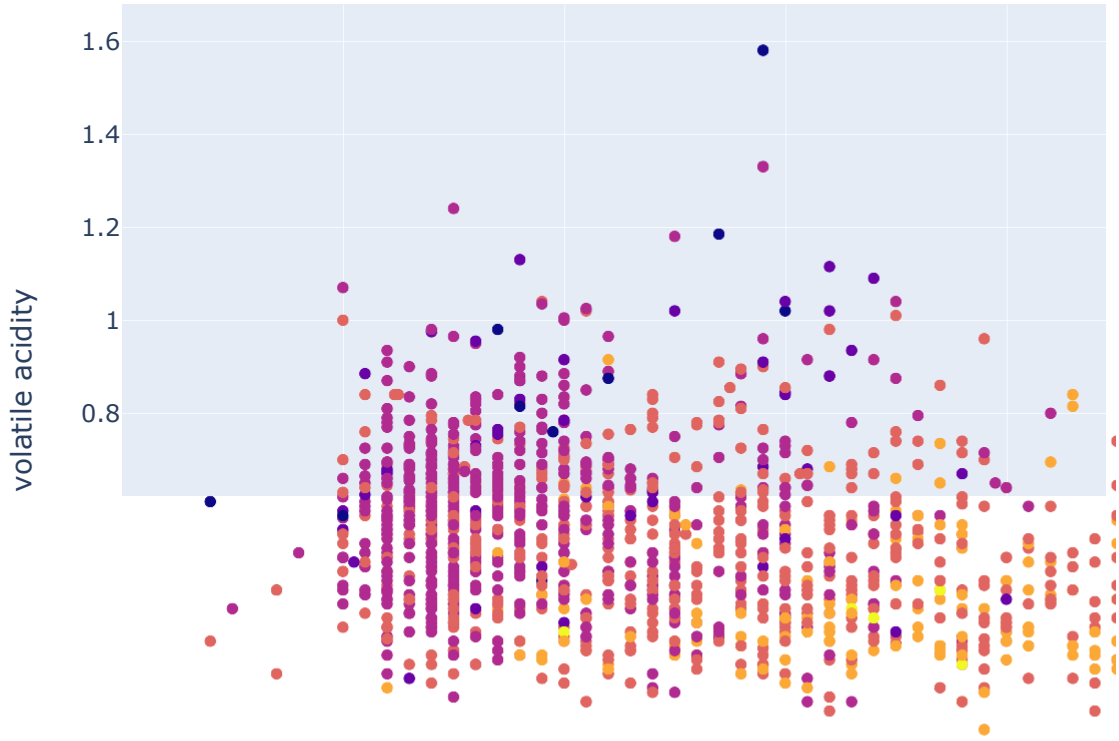
```
5    577
6    535
7    167
4     53
8     17
3     10
Name: quality, dtype: int64
```





In [16]:

```
px.scatter(df, x ='alcohol', y= 'volatile acidity', color ='quality')
```



** Imbalancy in dataset found**

- Most of the wines are having quality of **5 or 6**.
- A **class-imbalance** in the target feature is observed.
- To overcome this imbalancy, oversampling method is used for modeling.

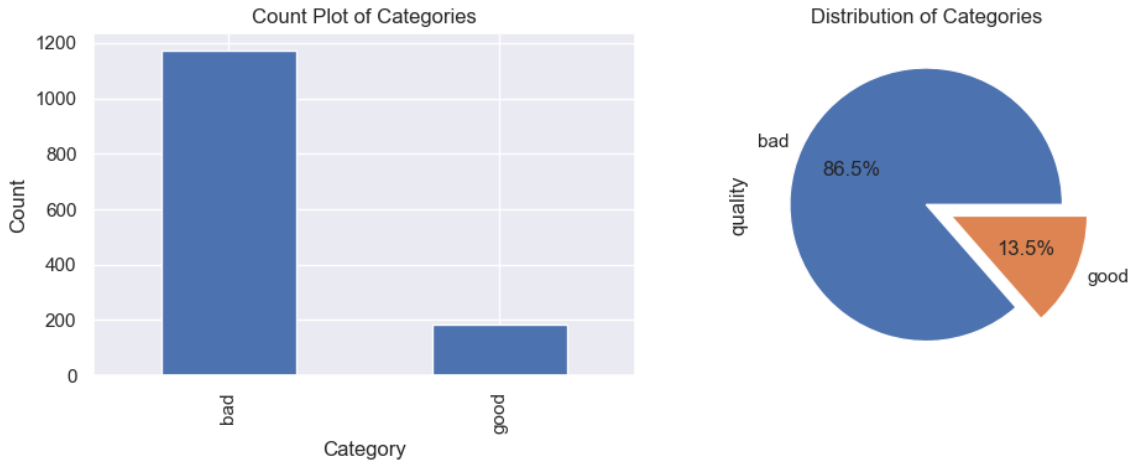
📌 Changing multiclass target feature into binary class

> The wine quality those have quality value 7 and more are considered as good and qualty below 7 is catagorised as bad.

In [17]:

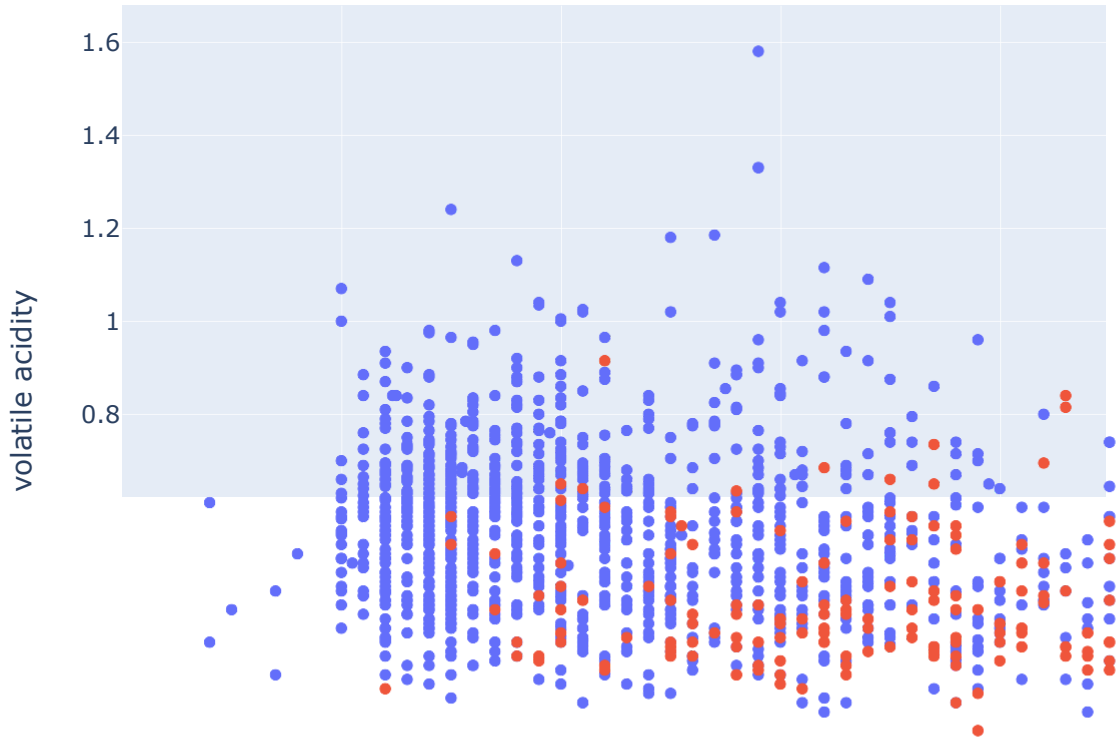
```
quality_th = 7
condition = df['quality'] >= quality_th
values_to_replace = {True: 'good', False: 'bad'}
df['quality'] = np.where(condition, 'good', 'bad')
new_value_counts =df.quality.value_counts()
print( new_value_counts)

fig, axes = plt.subplots(1, 2, figsize=(10, 4))

new_value_counts.plot(kind='bar', ax=axes[0],)
axes[0].set_xlabel('Category')
axes[0].set_ylabel('Count')
axes[0].set_title('Count Plot of Categories')


new_value_counts.plot(
    kind='pie',
    ax=axes[1],
    explode=[0,0.2],     # Explode the second category to emphasize it
    autopct='%1.1f%%',         # Display percentage with one decimal place
    shadow=False               # Add a shadow effect to the chart
    #colors=['cyan', 'gray', 'orange', 'green']  # Custom colors for the pie chart
)
axes[1].set_title('Distribution of Categories')


plt.tight_layout()

plt.show()
```

```
bad     1175
good     184
Name: quality, dtype: int64
```



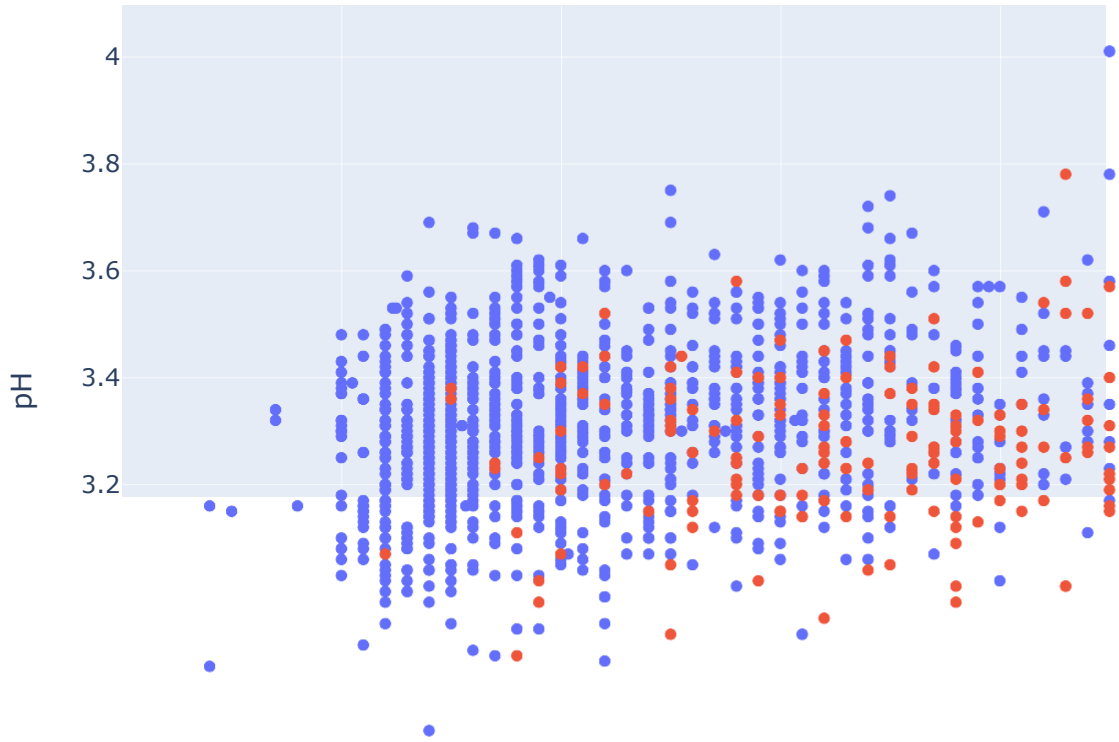
In [18]:

```
px.scatter(df, x ='alcohol', y= 'volatile acidity', color ='quality')
```



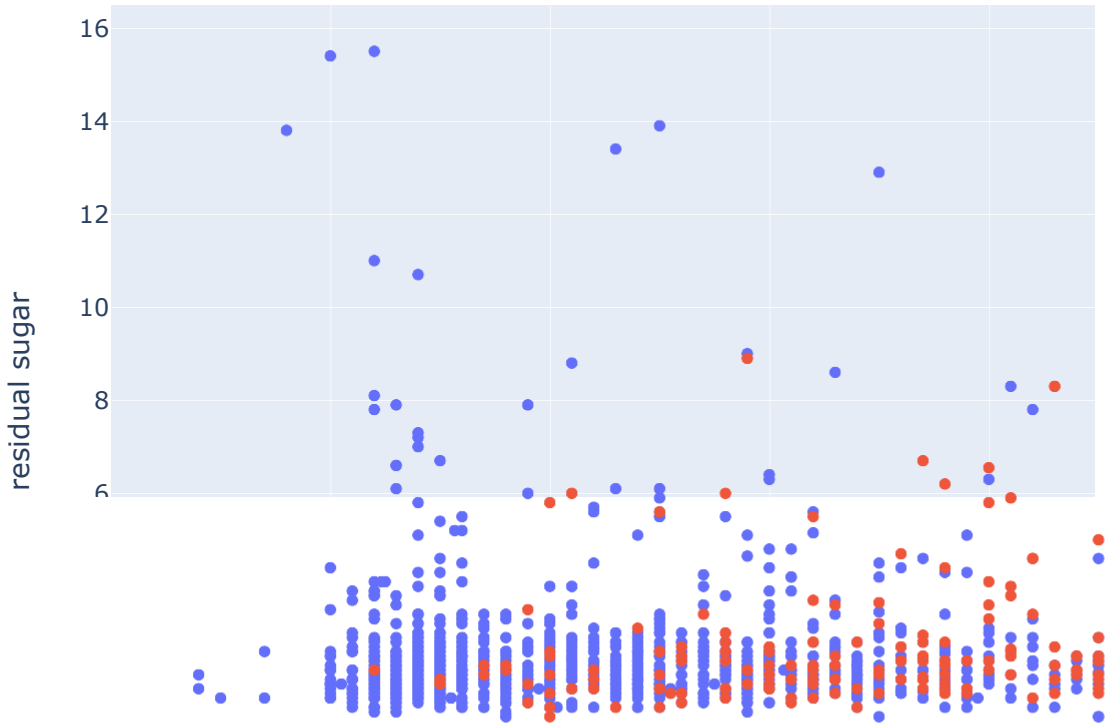
In [19]:

```
px.scatter(df, x ='alcohol', y= 'pH', color ='quality')
```



In [20]:

```
px.scatter(df, x ='alcohol', y= 'residual sugar', color ='quality')
```



🛠️🧱🔨 Model Building Predict Data and Evaluation of Model

Sklearn Libraries

In [21]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.datasets import make_classification

from sklearn.tree import DecisionTreeClassifier, plot_tree, export_text
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, GradientBoosting
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

Encoding

In [22]:

```
df.quality.replace({'good':1, 'bad':0}, inplace= True)
df.quality.unique()
```

Out[22]:

```
array([0, 1], dtype=int64)
```

Splitting into target and independent features

In [23]:

```
target = "quality"
x = df.drop(columns=target)
y = df[target]
```

Feature Scaling

In [24]:

```
#StandardScaler in dataframe
sc = StandardScaler()
sc_x = pd.DataFrame(sc.fit_transform(x) , columns=x.columns)
sc_x.head()
```

Out[24]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	density	pH	sulpl
0	-0.294063	1.915800	-1.393258	0.056665	0.200094	0.872003	0.048737	-0.708395	0.12
1	-0.294063	1.259934	-1.188617	-0.165259	0.078535	-0.085537	0.155790	-0.321247	-0.05
2	1.664067	-1.363534	1.471711	-0.461157	-0.265883	0.105971	0.691057	-0.966495	-0.46
3	-0.524431	0.932000	-1.393258	-0.461157	-0.245623	-0.468554	0.584003	1.291872	-0.57
4	-0.524431	0.713378	-1.393258	-0.535132	-0.265883	-0.277045	0.584003	1.291872	-0.57

Imbalance treatment

In [25]:

```
from imblearn.over_sampling import RandomOverSampler
over = RandomOverSampler()
x_over, y_over = over.fit_resample(sc_x,y)
```

In [26]:

```
x_train,x_test,y_train,y_test = train_test_split(x_over,y_over,test_size=0.25,random_sta
```

Model comparision

In [27]:

```
m1 = LogisticRegression()
m2 = DecisionTreeClassifier()
m3 = BaggingClassifier()
m4 = RandomForestClassifier()
m5 = KNeighborsClassifier()
m6 = SVC()


models = [('LogisticRegression', m1),
          ('DecisionTreeClassifier', m2),
          ('BaggingClassifier', m3),
          ('RandomForestClassifier',m4),
          ('KNeighborsClassifier',m5),
          ('SupportVectorClassifier',m6)
        ]


from tabulate import tabulate
print(tabulate(models, headers=['Model name', 'Sklearn models']),)
```

```
Model name               Sklearn models
-----------------------  ------------------------
LogisticRegression       LogisticRegression()
DecisionTreeClassifier   DecisionTreeClassifier()
BaggingClassifier        BaggingClassifier()
RandomForestClassifier   RandomForestClassifier()
KNeighborsClassifier     KNeighborsClassifier()
SupportVectorClassifier  SVC()
```

In [28]:

```
Train_Accuracy = []
Test_Accuracy=[]
Variance =[]
Train_CV=[]
Test_CV=[]
Variance_CV=[]
Precision_tr =[]
Recall_tr =[]
F1_score_tr =[]
Precision_test =[]
Recall_test =[]
F1_score_test =[]
TP=[]
FP=[]
FN=[]
TN=[]


for model_name, model in models:

    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
    print('==='*23)
    print(f'           Model name: {model_name}')
    print('==='*23)
    print()

#=====================================================
#            Model Accuracy
#-----------------------------------------------------
    train_acc = accuracy_score(y_train, y_pred_train)
    test_acc = accuracy_score(y_test, y_pred_test)
    Train_Accuracy.append(train_acc.round(2)*100)
    Test_Accuracy.append(test_acc.round(2)*100)
    variance = abs(train_acc - test_acc).round(2)*100
    print(f'➥ Training Accuracy, {train_acc.round(2)*100} %' )
    print(f'➥ TestAccuracy, {test_acc.round(2)*100} %' )
    print(f'Variance : {variance}')

    Variance.append(variance)
    if (variance > 9) or  (train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')


    print()

#=====================================================
#            Cross Validation
#-----------------------------------------------------
    CV_train_acc =cross_val_score(model, x_train, y_train, cv =10).mean()
    CV_test_acc =cross_val_score(model, x_test, y_test, cv =10).mean()
    Train_CV.append(CV_train_acc.round(2)*100)
    Test_CV.append(CV_test_acc.round(2)*100)
    print(f'➥ Training Accuracy(CrossValidation), {CV_train_acc.round(2)*100} %' )
    print(f'➥ TestAccuracy(CrossValidation), {CV_test_acc.round(2)*100} %' )
    variance_CV = abs(CV_train_acc - CV_test_acc).round(2)*100
    Variance_CV.append(variance_CV)
    print(f'Variance (CrossValidation) : {variance_CV}')
```

```
    if (variance_CV > 9) or (CV_train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')
    print()
#=========================================================
#           Evalutaion Metrics
#---------------------------------------------------------
#     print( '--'*30)
#     print( 'Classification_report:\n')
#     print( '--'*30)
#     print( 'Train:')
#     print( classification_report(y_train, y_pred_train))

#     print( 'Test:')
#     print( '--'*30)
#     print(classification_report(y_test, y_pred_test))
#     print( '--'*30)

    cm_train= confusion_matrix(y_train, y_pred_train)
    cm_test = confusion_matrix(y_test, y_pred_test)
#     print( '--'*30)
#     print( 'confusion_matrix_Train:\n',cm_train,'\nconfusion_matrix_Test :\n',cm_test )
#     print('--'*30)


#     TP_tr =cm_train[0,0]
#     FP_tr = cm_train[0,1]
#     FN_tr = cm_train[1,0]
#     TN_tr = cm_train[1,1]


#     TP_.append(TP_tr)
#     Train.append(FP_tr)
#     Train.append(FN_tr)
#     Train.append(TN_tr)

    TP_test = cm_test[0,0]
    FP_test = cm_test[0,1]
    FN_test = cm_test[1,0]
    TN_test = cm_test[1,1]


    TP.append(TP_test)
    FP.append(FP_test)
    FN.append(FN_test)
    TN.append(TN_test)


model=[i[0] for i in models]
#print(len(Train_Accuracy), len(Test_Accuracy),len(Variance),len(Train_CV),len(Test_CV),
metrics={"Model":model,"Train accuracy":Train_Accuracy,"Test accuracy":Test_Accuracy,
         "Variance": Variance, "Train accuracy(CV)":Train_CV,"Test accuracy(CV)":Test_CV
          "TP":TP,"TN":TN,"FP":FP,"FN":FN}

metric_df = pd.DataFrame(metrics)
#print(metric_df.shape)
metric_df
#print(TN)
```

```
======================================================================
           Model name: LogisticRegression
======================================================================

- Training Accuracy, 83.0 %
- TestAccuracy, 82.0 %
Variance : 1.0

- Training Accuracy(CrossValidation), 83.0 %
- TestAccuracy(CrossValidation), 81.0 %
Variance (CrossValidation) : 2.0

======================================================================
           Model name: DecisionTreeClassifier
======================================================================

- Training Accuracy, 100.0 %
- TestAccuracy, 94.0 %
Variance : 6.0

- Training Accuracy(CrossValidation), 94.0 %
- TestAccuracy(CrossValidation), 83.0 %
Variance (CrossValidation) : 11.0
ATTENTION : The  DecisionTreeClassifier Model is overfitting

======================================================================
           Model name: BaggingClassifier
======================================================================

- Training Accuracy, 100.0 %
- TestAccuracy, 95.0 %
Variance : 5.0

- Training Accuracy(CrossValidation), 95.0 %
- TestAccuracy(CrossValidation), 87.0 %
Variance (CrossValidation) : 8.0

======================================================================
           Model name: RandomForestClassifier
======================================================================

- Training Accuracy, 100.0 %
- TestAccuracy, 96.0 %
Variance : 4.0

- Training Accuracy(CrossValidation), 95.0 %
- TestAccuracy(CrossValidation), 89.0 %
Variance (CrossValidation) : 6.0

======================================================================
           Model name: KNeighborsClassifier
======================================================================

- Training Accuracy, 91.0 %
- TestAccuracy, 86.0 %
Variance : 5.0

- Training Accuracy(CrossValidation), 85.0 %
- TestAccuracy(CrossValidation), 78.0 %
Variance (CrossValidation) : 7.000000000000001
```

```
========================================================================
           Model name: SupportVectorClassifier
========================================================================

- Training Accuracy, 87.0 %
- TestAccuracy, 82.0 %
Variance : 5.0

- Training Accuracy(CrossValidation), 86.0 %
- TestAccuracy(CrossValidation), 81.0 %
Variance (CrossValidation) : 4.0
```

Out[28]:

	Model	Train accuracy	Test accuracy	Variance	Train accuracy(CV)	Test accuracy(CV)	Varianc
0	LogisticRegression	83.0	82.0	1.0	83.0	81.0	
1	DecisionTreeClassifier	100.0	94.0	6.0	94.0	83.0	
2	BaggingClassifier	100.0	95.0	5.0	95.0	87.0	
3	RandomForestClassifier	100.0	96.0	4.0	95.0	89.0	
4	KNeighborsClassifier	91.0	86.0	5.0	85.0	78.0	
5	SupportVectorClassifier	87.0	82.0	5.0	86.0	81.0	

Enhancing Model Performance through Parameter Tuning

In [29]:

```
m7 = DecisionTreeClassifier(criterion ='gini', max_depth = 10)
m8 = BaggingClassifier(estimator = KNeighborsClassifier())
m9 = RandomForestClassifier(n_estimators = 100, criterion='gini',max_depth =9)
m10 = KNeighborsClassifier(n_neighbors=7)
m11 = SVC(kernel ='rbf', gamma = 'scale', C=30)

models. append(('DecisionTree_tuning',m7))
models. append(('Bagging_tuning',m8))
models. append(('RandomForest_tuning',m9))
models. append(('KNN_tunning',m10))
models. append(('SVC_tuning',m11))
print(tabulate(models, headers=['Model name', 'Sklearn models']),)
```

```
Model name               Sklearn models
-----------------------  ------------------------------------------------
--
LogisticRegression       LogisticRegression()
DecisionTreeClassifier   DecisionTreeClassifier()
BaggingClassifier        BaggingClassifier()
RandomForestClassifier   RandomForestClassifier()
KNeighborsClassifier     KNeighborsClassifier()
SupportVectorClassifier  SVC()
DecisionTree_tuning      DecisionTreeClassifier(max_depth=10)
Bagging_tuning           BaggingClassifier(estimator=KNeighborsClassifier
())
RandomForest_tuning      RandomForestClassifier(max_depth=9)
KNN_tunning              KNeighborsClassifier(n_neighbors=7)
SVC_tuning               SVC(C=30)
```

In [30]:

```
Train_Accuracy = []
Test_Accuracy=[]
Variance =[]
Train_CV=[]
Test_CV=[]
Variance_CV=[]
Precision_tr =[]
Recall_tr =[]
F1_score_tr =[]
Precision_test =[]
Recall_test =[]
F1_score_test =[]
TP=[]
FP=[]
FN=[]
TN=[]


for model_name, model in models:

    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
    print('==='*23)
    print(f'           Model name: {model_name}')
    print('==='*23)
    print()

#=====================================================
#            Model Accuracy
#-----------------------------------------------------
    train_acc = accuracy_score(y_train, y_pred_train)
    test_acc = accuracy_score(y_test, y_pred_test)
    Train_Accuracy.append(train_acc.round(2)*100)
    Test_Accuracy.append(test_acc.round(2)*100)
    variance = abs(train_acc - test_acc).round(2)*100
    print(f'➡ Training Accuracy, {train_acc.round(2)*100} %' )
    print(f'➡ TestAccuracy, {test_acc.round(2)*100} %' )
    print(f'Variance : {variance}')

    Variance.append(variance)
    if (variance > 9) or  (train_acc > 0.99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')


    print()

#=====================================================
#            Cross Validation
#-----------------------------------------------------
    CV_train_acc =cross_val_score(model, x_train, y_train, cv =10).mean()
    CV_test_acc =cross_val_score(model, x_test, y_test, cv =10).mean()
    Train_CV.append(CV_train_acc.round(2)*100)
    Test_CV.append(CV_test_acc.round(2)*100)
    print(f'➡ Training Accuracy(CrossValidation), {CV_train_acc.round(2)*100} %' )
    print(f'➡ TestAccuracy(CrossValidation), {CV_test_acc.round(2)*100} %' )
    variance_CV = abs(CV_train_acc - CV_test_acc).round(2)*100
    Variance_CV.append(variance_CV)
    print(f'Variance (CrossValidation) : {variance_CV}')
```

```
    if (variance_CV > 9) or (CV_train_acc > 0.99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')
    print()
#=========================================================
#           Evalutaion Metrics
#---------------------------------------------------------
    print( '--'*30)
    print( 'Classification_report:\n')
    print( '--'*30)
    print( 'Train:')
    print( classification_report(y_train, y_pred_train))

    print( 'Test:')
    print( '--'*30)
    print(classification_report(y_test, y_pred_test))
    print( '--'*30)

    cm_train= confusion_matrix(y_train, y_pred_train)
    cm_test = confusion_matrix(y_test, y_pred_test)
#     print( '--'*30)
#     print( 'confusion_matrix_Train:\n',cm_train,'\nconfusion_matrix_Test :\n',cm_test )
#     print('--'*30)


#     TP_tr =cm_train[0,0]
#     FP_tr = cm_train[0,1]
#     FN_tr = cm_train[1,0]
#     TN_tr = cm_train[1,1]


#     TP_.append(TP_tr)
#     Train.append(FP_tr)
#     Train.append(FN_tr)
#     Train.append(TN_tr)

    TP_test = cm_test[0,0]
    FP_test = cm_test[0,1]
    FN_test = cm_test[1,0]
    TN_test = cm_test[1,1]


    TP.append(TP_test)
    FP.append(FP_test)
    FN.append(FN_test)
    TN.append(TN_test)

    fpr, tpr, _ = roc_curve(y_test, y_pred_test)
    roc_auc = auc(fpr, tpr)


    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')

# Plot the diagonal line representing a random classifier (AUC = 0.5)

plt.plot([0, 1], [0, 1], 'm--', label='AUC = 0.5')

# Set labels and title

plt.xlabel('False Positive Rate (FPR)')
plt.ylabel('True Positive Rate (TPR) or Recall')
plt.title('Receiver Operating Characteristic (ROC) Curve')
```

```
plt.legend(loc='lower right')

# Show the plot
plt.show()


model=[i[0] for i in models]
#print(len(Train_Accuracy), len(Test_Accuracy),len(Variance),len(Train_CV),len(Test_CV),
metrics={"Model":model,"Train accuracy":Train_Accuracy,"Test accuracy":Test_Accuracy,
         "Variance": Variance, "Train accuracy(CV)":Train_CV,"Test accuracy(CV)":Test_CV
          "TP":TP,"TN":TN,"FP":FP,"FN":FN}

metric_df = pd.DataFrame(metrics)
#print(metric_df.shape)
metric_df
#print(TN)
```

```
======================================================================
           Model name: LogisticRegression
======================================================================

- Training Accuracy, 83.0 %
- TestAccuracy, 82.0 %
Variance : 1.0

- Training Accuracy(CrossValidation), 83.0 %
- TestAccuracy(CrossValidation), 81.0 %
Variance (CrossValidation) : 2.0

----------------------------------------------------------
Classification_report:

----------------------------------------------------------
Train:
              precision    recall  f1-score   support

           0       0.85      0.79      0.82       881
           1       0.81      0.86      0.83       881

    accuracy                           0.83      1762
   macro avg       0.83      0.83      0.83      1762
weighted avg       0.83      0.83      0.83      1762

Test:
----------------------------------------------------------
              precision    recall  f1-score   support

           0       0.84      0.79      0.82       294
           1       0.80      0.85      0.83       294

    accuracy                           0.82       588
   macro avg       0.82      0.82      0.82       588
weighted avg       0.82      0.82      0.82       588

----------------------------------------------------------
======================================================================
           Model name: DecisionTreeClassifier
======================================================================

- Training Accuracy, 100.0 %
- TestAccuracy, 94.0 %
Variance : 6.0
ATTENTION : The  DecisionTreeClassifier Model is overfitting

- Training Accuracy(CrossValidation), 94.0 %
- TestAccuracy(CrossValidation), 84.0 %
Variance (CrossValidation) : 10.0
ATTENTION : The  DecisionTreeClassifier Model is overfitting

----------------------------------------------------------
Classification_report:

----------------------------------------------------------
Train:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       881
           1       1.00      1.00      1.00       881
```

```
    accuracy                           1.00      1762
   macro avg       1.00      1.00      1.00      1762
weighted avg       1.00      1.00      1.00      1762

Test:
--------------------------------------------------------------
              precision    recall  f1-score   support

           0       1.00      0.87      0.93       294
           1       0.89      1.00      0.94       294

    accuracy                           0.94       588
   macro avg       0.94      0.94      0.94       588
weighted avg       0.94      0.94      0.94       588

--------------------------------------------------------------
==============================================================================
           Model name: BaggingClassifier
==============================================================================

- Training Accuracy, 100.0 %
- TestAccuracy, 96.0 %
Variance : 4.0
ATTENTION : The  BaggingClassifier Model is overfitting

- Training Accuracy(CrossValidation), 94.0 %
- TestAccuracy(CrossValidation), 87.0 %
Variance (CrossValidation) : 7.000000000000001

--------------------------------------------------------------
Classification_report:

--------------------------------------------------------------
Train:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       881
           1       1.00      1.00      1.00       881

    accuracy                           1.00      1762
   macro avg       1.00      1.00      1.00      1762
weighted avg       1.00      1.00      1.00      1762

Test:
--------------------------------------------------------------
              precision    recall  f1-score   support

           0       1.00      0.93      0.96       294
           1       0.93      1.00      0.96       294

    accuracy                           0.96       588
   macro avg       0.97      0.96      0.96       588
weighted avg       0.97      0.96      0.96       588

--------------------------------------------------------------
==============================================================================
           Model name: RandomForestClassifier
==============================================================================

- Training Accuracy, 100.0 %
```

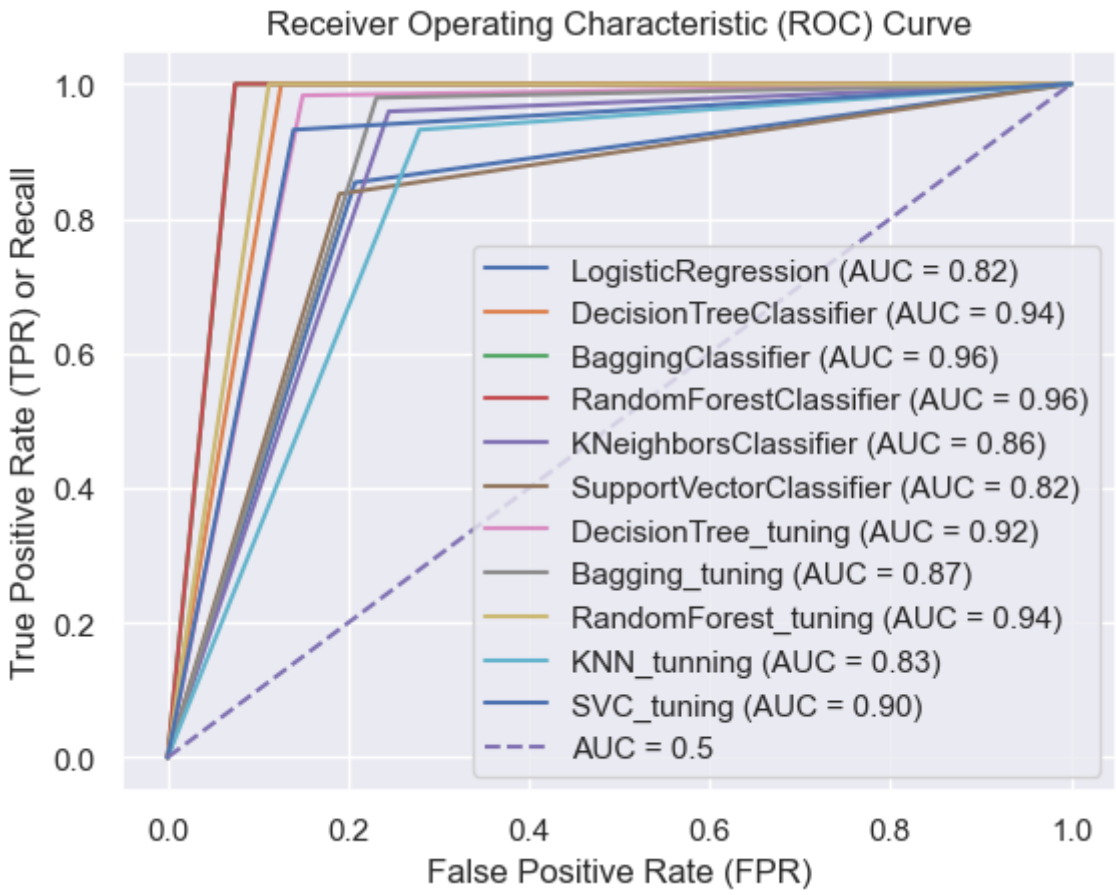
```
➖ TestAccuracy, 96.0 %
Variance : 4.0
ATTENTION : The  RandomForestClassifier Model is overfitting

➖ Training Accuracy(CrossValidation), 95.0 %
➖ TestAccuracy(CrossValidation), 89.0 %
Variance (CrossValidation) : 6.0

------------------------------------------------------------
Classification_report:

------------------------------------------------------------
Train:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       881
           1       1.00      1.00      1.00       881

    accuracy                           1.00      1762
   macro avg       1.00      1.00      1.00      1762
weighted avg       1.00      1.00      1.00      1762

Test:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       1.00      0.93      0.96       294
           1       0.93      1.00      0.96       294

    accuracy                           0.96       588
   macro avg       0.97      0.96      0.96       588
weighted avg       0.97      0.96      0.96       588

------------------------------------------------------------
============================================================================
           Model name: KNeighborsClassifier
============================================================================

➖ Training Accuracy, 91.0 %
➖ TestAccuracy, 86.0 %
Variance : 5.0

➖ Training Accuracy(CrossValidation), 85.0 %
➖ TestAccuracy(CrossValidation), 78.0 %
Variance (CrossValidation) : 7.000000000000001

------------------------------------------------------------
Classification_report:

------------------------------------------------------------
Train:
              precision    recall  f1-score   support

           0       0.98      0.83      0.90       881
           1       0.85      0.98      0.91       881

    accuracy                           0.91      1762
   macro avg       0.92      0.91      0.91      1762
weighted avg       0.92      0.91      0.91      1762

Test:
```

```
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.95      0.76      0.84       294
           1       0.80      0.96      0.87       294

    accuracy                           0.86       588
   macro avg       0.87      0.86      0.86       588
weighted avg       0.87      0.86      0.86       588

------------------------------------------------------------
============================================================================
           Model name: SupportVectorClassifier
============================================================================

- Training Accuracy, 87.0 %
- TestAccuracy, 82.0 %
Variance : 5.0

- Training Accuracy(CrossValidation), 86.0 %
- TestAccuracy(CrossValidation), 81.0 %
Variance (CrossValidation) : 4.0

------------------------------------------------------------
Classification_report:

------------------------------------------------------------
Train:
              precision    recall  f1-score   support

           0       0.89      0.85      0.87       881
           1       0.85      0.89      0.87       881

    accuracy                           0.87      1762
   macro avg       0.87      0.87      0.87      1762
weighted avg       0.87      0.87      0.87      1762

Test:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.83      0.81      0.82       294
           1       0.81      0.84      0.83       294

    accuracy                           0.82       588
   macro avg       0.82      0.82      0.82       588
weighted avg       0.82      0.82      0.82       588

------------------------------------------------------------
============================================================================
           Model name: DecisionTree_tuning
============================================================================

- Training Accuracy, 96.0 %
- TestAccuracy, 92.0 %
Variance : 5.0
```

```
- Training Accuracy(CrossValidation), 91.0 %
- TestAccuracy(CrossValidation), 83.0 %
Variance (CrossValidation) : 8.0

------------------------------------------------------------
Classification_report:

------------------------------------------------------------
Train:
              precision    recall  f1-score   support

           0       0.99      0.94      0.96       881
           1       0.94      0.99      0.96       881

    accuracy                           0.96      1762
   macro avg       0.96      0.96      0.96      1762
weighted avg       0.96      0.96      0.96      1762

Test:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.98      0.85      0.91       294
           1       0.87      0.98      0.92       294

    accuracy                           0.92       588
   macro avg       0.92      0.92      0.92       588
weighted avg       0.92      0.92      0.92       588

------------------------------------------------------------
============================================================================
           Model name: Bagging_tuning
============================================================================

- Training Accuracy, 91.0 %
- TestAccuracy, 87.0 %
Variance : 4.0

- Training Accuracy(CrossValidation), 87.0 %
- TestAccuracy(CrossValidation), 78.0 %
Variance (CrossValidation) : 8.0

------------------------------------------------------------
Classification_report:

------------------------------------------------------------
Train:
              precision    recall  f1-score   support

           0       0.99      0.84      0.91       881
           1       0.86      0.99      0.92       881

    accuracy                           0.91      1762
   macro avg       0.92      0.91      0.91      1762
weighted avg       0.92      0.91      0.91      1762

Test:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.97      0.77      0.86       294
```

```
           1       0.81      0.98      0.89       294

    accuracy                           0.87       588
   macro avg       0.89      0.87      0.87       588
weighted avg       0.89      0.87      0.87       588

------------------------------------------------------------
====================================================================
           Model name: RandomForest_tuning
====================================================================

- Training Accuracy, 98.0 %
- TestAccuracy, 94.0 %
Variance : 3.0

- Training Accuracy(CrossValidation), 94.0 %
- TestAccuracy(CrossValidation), 89.0 %
Variance (CrossValidation) : 5.0

------------------------------------------------------------
Classification_report:

------------------------------------------------------------
Train:
              precision    recall  f1-score   support

           0       1.00      0.96      0.98       881
           1       0.96      1.00      0.98       881

    accuracy                           0.98      1762
   macro avg       0.98      0.98      0.98      1762
weighted avg       0.98      0.98      0.98      1762

Test:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       1.00      0.89      0.94       294
           1       0.90      1.00      0.95       294

    accuracy                           0.94       588
   macro avg       0.95      0.94      0.94       588
weighted avg       0.95      0.94      0.94       588

------------------------------------------------------------
====================================================================
           Model name: KNN_tunning
====================================================================

- Training Accuracy, 88.0 %
- TestAccuracy, 83.0 %
Variance : 5.0

- Training Accuracy(CrossValidation), 83.0 %
- TestAccuracy(CrossValidation), 76.0 %
Variance (CrossValidation) : 7.000000000000001

------------------------------------------------------------
Classification_report:

------------------------------------------------------------
```

```
Train:
              precision    recall  f1-score   support

           0       0.96      0.78      0.86       881
           1       0.82      0.97      0.89       881

    accuracy                           0.88      1762
   macro avg       0.89      0.88      0.88      1762
weighted avg       0.89      0.88      0.88      1762

Test:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.91      0.72      0.81       294
           1       0.77      0.93      0.84       294

    accuracy                           0.83       588
   macro avg       0.84      0.83      0.82       588
weighted avg       0.84      0.83      0.82       588

------------------------------------------------------------
============================================================================
           Model name: SVC_tuning
============================================================================

- Training Accuracy, 96.0 %
- TestAccuracy, 90.0 %
Variance : 6.0

- Training Accuracy(CrossValidation), 90.0 %
- TestAccuracy(CrossValidation), 83.0 %
Variance (CrossValidation) : 6.0

------------------------------------------------------------
Classification_report:

------------------------------------------------------------
Train:
              precision    recall  f1-score   support

           0       0.96      0.95      0.96       881
           1       0.95      0.96      0.96       881

    accuracy                           0.96      1762
   macro avg       0.96      0.96      0.96      1762
weighted avg       0.96      0.96      0.96      1762

Test:
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.93      0.86      0.89       294
           1       0.87      0.93      0.90       294

    accuracy                           0.90       588
   macro avg       0.90      0.90      0.90       588
weighted avg       0.90      0.90      0.90       588

------------------------------------------------------------
```



Out[30]:

	Model	Train accuracy	Test accuracy	Variance	Train accuracy(CV)	Test accuracy(CV)	Varian
0	LogisticRegression	83.0	82.0	1.0	83.0	81.0	
1	DecisionTreeClassifier	100.0	94.0	6.0	94.0	84.0	
2	BaggingClassifier	100.0	96.0	4.0	94.0	87.0	
3	RandomForestClassifier	100.0	96.0	4.0	95.0	89.0	
4	KNeighborsClassifier	91.0	86.0	5.0	85.0	78.0	
5	SupportVectorClassifier	87.0	82.0	5.0	86.0	81.0	
6	DecisionTree_tuning	96.0	92.0	5.0	91.0	83.0	
7	Bagging_tuning	91.0	87.0	4.0	87.0	78.0	
8	RandomForest_tuning	98.0	94.0	3.0	94.0	89.0	
9	KNN_tunning	88.0	83.0	5.0	83.0	76.0	
10	SVC_tuning	96.0	90.0	6.0	90.0	83.0	

⚖️ Comparision of models before and after hyperparameter tuning:

1. Decision Tree Classifier

In [31]:

```
[models[1],models[6]]
```

Out[31]:

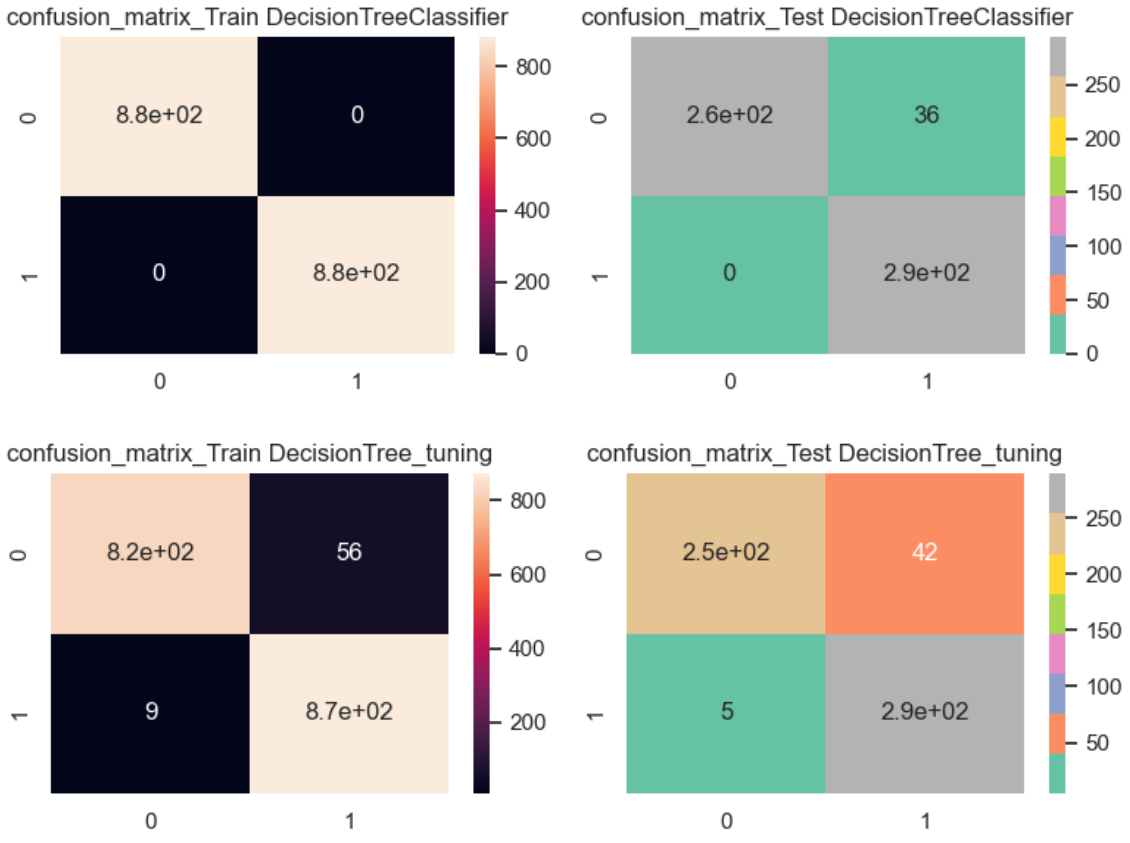
```
[('DecisionTreeClassifier', DecisionTreeClassifier()),
 ('DecisionTree_tuning', DecisionTreeClassifier(max_depth=10))]
```

In [32]:

```
for model_name, model in [models[1],models[6]]:

    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
    print('==='*23)
    print(f'            Model name: {model_name}')
    print('==='*23)
    print()

#=================================================
#            Model Accuracy
#-------------------------------------------------
    train_acc = accuracy_score(y_train, y_pred_train)
    test_acc = accuracy_score(y_test, y_pred_test)
    Train_Accuracy.append(train_acc.round(2)*100)
    Test_Accuracy.append(test_acc.round(2)*100)
    variance = abs(train_acc - test_acc).round(2)*100
    print(f'➡ Training Accuracy, {train_acc.round(2)*100} %' )
    print(f'➡ TestAccuracy, {test_acc.round(2)*100} %' )
    print(f'Variance : {variance}')

    Variance.append(variance)
    if (variance > 9) or  (train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')


    print()

#=========================================================
#            Cross Validation
#---------------------------------------------------------
    CV_train_acc =cross_val_score(model, x_train, y_train, cv =10).mean()
    CV_test_acc =cross_val_score(model, x_test, y_test, cv =10).mean()
    Train_CV.append(CV_train_acc.round(2)*100)
    Test_CV.append(CV_test_acc.round(2)*100)
    print(f'➡ Training Accuracy(CrossValidation), {CV_train_acc.round(2)*100} %' )
    print(f'➡ TestAccuracy(CrossValidation), {CV_test_acc.round(2)*100} %' )
    variance_CV = abs(CV_train_acc - CV_test_acc).round(2)*100
    Variance_CV.append(variance_CV)
    print(f'Variance (CrossValidation) : {variance_CV}')
    if (variance_CV > 9) or (CV_train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')
    print()


    cm_train= confusion_matrix(y_train, y_pred_train)
    cm_test = confusion_matrix(y_test, y_pred_test)

#     print('===='*10)
#     print( 'confusion_matrix_Train :','\n','\n',cm_train)
#     print('===='*10)
#     print( 'confusion_matrix_Test :','\n','\n',cm_test)
#     print('===='*10)
    plt.figure(figsize = (8,3))
    plt.subplot(1,2,1)
    sns.heatmap(cm_train, annot = True, )
    plt.title(f'confusion_matrix_Train {model_name}')
    plt.subplot(1,2,2)
```

```
    sns.heatmap(cm_test, annot = True, cmap= 'Set2')
    plt.title(f'confusion_matrix_Test {model_name}')
    plt.tight_layout()
```

```
======================================================================
           Model name: DecisionTreeClassifier
======================================================================

- Training Accuracy, 100.0 %
- TestAccuracy, 94.0 %
Variance : 6.0

- Training Accuracy(CrossValidation), 94.0 %
- TestAccuracy(CrossValidation), 83.0 %
Variance (CrossValidation) : 10.0
ATTENTION : The  DecisionTreeClassifier Model is overfitting

======================================================================
           Model name: DecisionTree_tuning
======================================================================

- Training Accuracy, 96.0 %
- TestAccuracy, 92.0 %
Variance : 4.0

- Training Accuracy(CrossValidation), 91.0 %
- TestAccuracy(CrossValidation), 84.0 %
Variance (CrossValidation) : 7.000000000000001
```



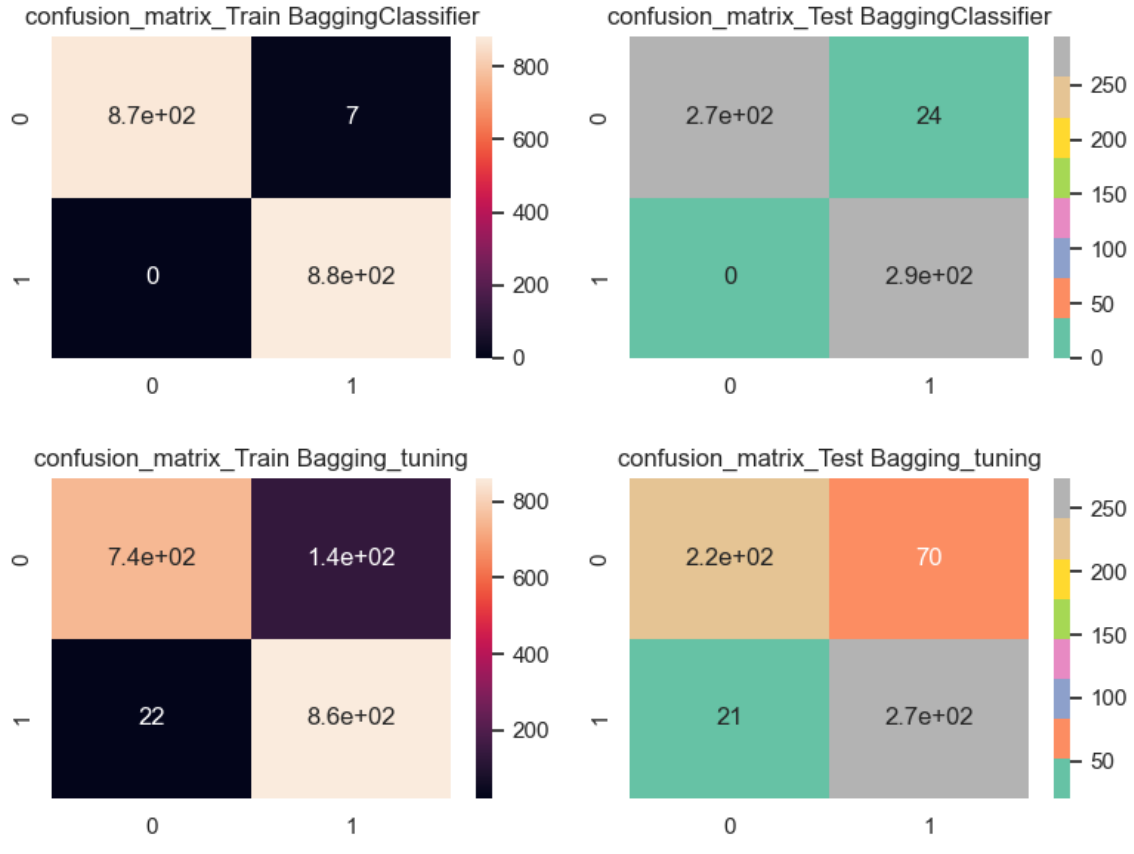
2. BaggingClassifier

In [33]:

```
for model_name, model in [models[2],models[7]]:

    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
    print('==='*23)
    print(f'           Model name: {model_name}')
    print('==='*23)
    print()
#=================================================
#            Model Accuracy
#-------------------------------------------------
    train_acc = accuracy_score(y_train, y_pred_train)
    test_acc = accuracy_score(y_test, y_pred_test)
    Train_Accuracy.append(train_acc.round(2)*100)
    Test_Accuracy.append(test_acc.round(2)*100)
    variance = abs(train_acc - test_acc).round(2)*100
    print(f'➡ Training Accuracy, {train_acc.round(2)*100} %' )
    print(f'➡ TestAccuracy, {test_acc.round(2)*100} %' )
    print(f'Variance : {variance}')

    Variance.append(variance)
    if (variance > 9) or  (train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')


    print()

#=========================================================
#            Cross Validation
#---------------------------------------------------------
    CV_train_acc =cross_val_score(model, x_train, y_train, cv =10).mean()
    CV_test_acc =cross_val_score(model, x_test, y_test, cv =10).mean()
    Train_CV.append(CV_train_acc.round(2)*100)
    Test_CV.append(CV_test_acc.round(2)*100)
    print(f'➡ Training Accuracy(CrossValidation), {CV_train_acc.round(2)*100} %' )
    print(f'➡ TestAccuracy(CrossValidation), {CV_test_acc.round(2)*100} %' )
    variance_CV = abs(CV_train_acc - CV_test_acc).round(2)*100
    Variance_CV.append(variance_CV)
    print(f'Variance (CrossValidation) : {variance_CV}')
    if (variance_CV > 9) or (CV_train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')
    print()

    cm_train= confusion_matrix(y_train, y_pred_train)
    cm_test = confusion_matrix(y_test, y_pred_test)

#     print('===='*10)
#     print( 'confusion_matrix_Train :','\n','\n',cm_train)
#     print('===='*10)
#     print( 'confusion_matrix_Test :','\n','\n',cm_test)
#     print('===='*10)
    plt.figure(figsize = (8,3))
    plt.subplot(1,2,1)
    sns.heatmap(cm_train, annot = True, )
    plt.title(f'confusion_matrix_Train {model_name}')
    plt.subplot(1,2,2)
    sns.heatmap(cm_test, annot = True, cmap= 'Set2')
    plt.title(f'confusion_matrix_Test {model_name}')
```

```
    plt.tight_layout()
```

```
======================================================================
           Model name: BaggingClassifier
======================================================================

- Training Accuracy, 100.0 %
- TestAccuracy, 96.0 %
Variance : 4.0

- Training Accuracy(CrossValidation), 95.0 %
- TestAccuracy(CrossValidation), 88.0 %
Variance (CrossValidation) : 7.000000000000001

======================================================================
           Model name: Bagging_tuning
======================================================================

- Training Accuracy, 91.0 %
- TestAccuracy, 85.0 %
Variance : 7.000000000000001

- Training Accuracy(CrossValidation), 86.0 %
- TestAccuracy(CrossValidation), 78.0 %
Variance (CrossValidation) : 8.0
```



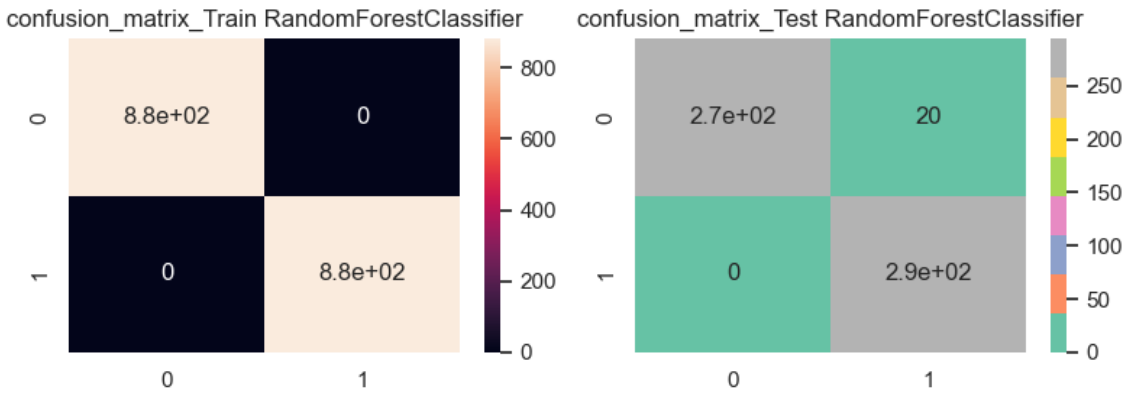
3. RandomForestClassifier

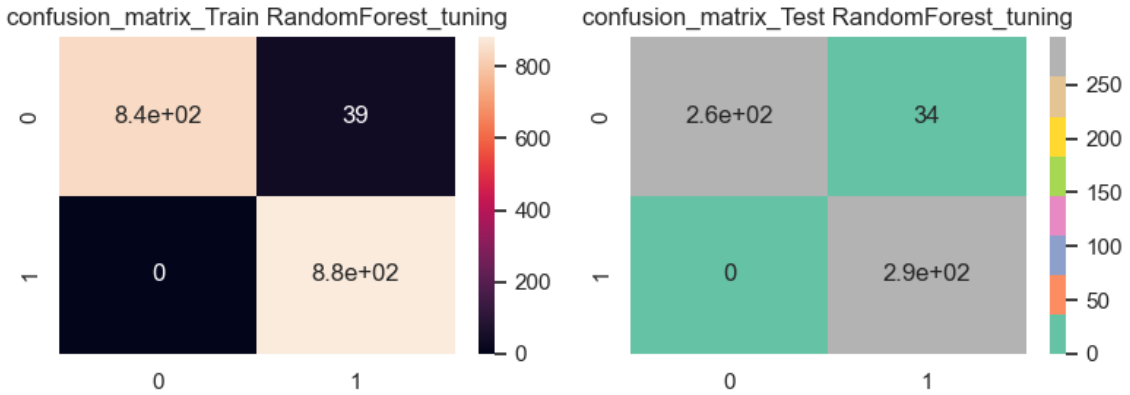
In [34]:

```
for model_name, model in [models[3],models[8]]:

    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
    print('==='*23)
    print(f'           Model name: {model_name}')
    print('==='*23)
    print()
#=================================================
#            Model Accuracy
#-------------------------------------------------
    train_acc = accuracy_score(y_train, y_pred_train)
    test_acc = accuracy_score(y_test, y_pred_test)
    Train_Accuracy.append(train_acc.round(2)*100)
    Test_Accuracy.append(test_acc.round(2)*100)
    variance = abs(train_acc - test_acc).round(2)*100
    print(f'➡ Training Accuracy, {train_acc.round(2)*100} %' )
    print(f'➡ TestAccuracy, {test_acc.round(2)*100} %' )
    print(f'Variance : {variance}')

    Variance.append(variance)
    if (variance > 9) or  (train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')


    print()

#=========================================================
#            Cross Validation
#---------------------------------------------------------
    CV_train_acc =cross_val_score(model, x_train, y_train, cv =10).mean()
    CV_test_acc =cross_val_score(model, x_test, y_test, cv =10).mean()
    Train_CV.append(CV_train_acc.round(2)*100)
    Test_CV.append(CV_test_acc.round(2)*100)
    print(f'➡ Training Accuracy(CrossValidation), {CV_train_acc.round(2)*100} %' )
    print(f'➡ TestAccuracy(CrossValidation), {CV_test_acc.round(2)*100} %' )
    variance_CV = abs(CV_train_acc - CV_test_acc).round(2)*100
    Variance_CV.append(variance_CV)
    print(f'Variance (CrossValidation) : {variance_CV}')
    if (variance_CV > 9) or (CV_train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')
    print()



    cm_train= confusion_matrix(y_train, y_pred_train)
    cm_test = confusion_matrix(y_test, y_pred_test)

#     print('===='*10)
#     print( 'confusion_matrix_Test :','\n','\n',cm_train)
#     print('===='*10)
#     print( 'confusion_matrix_Train :','\n','\n',cm_test)
#     print('===='*10)
    plt.figure(figsize = (8,3))
    plt.subplot(1,2,1)
    sns.heatmap(cm_train, annot = True, )
    plt.title(f'confusion_matrix_Train {model_name}')
```

```
    plt.subplot(1,2,2)
    sns.heatmap(cm_test, annot = True, cmap= 'Set2')
    plt.title(f'confusion_matrix_Test {model_name}')
    plt.tight_layout()
```

```
======================================================================
           Model name: RandomForestClassifier
======================================================================

- Training Accuracy, 100.0 %
- TestAccuracy, 97.0 %
Variance : 3.0

- Training Accuracy(CrossValidation), 95.0 %
- TestAccuracy(CrossValidation), 89.0 %
Variance (CrossValidation) : 6.0

======================================================================
           Model name: RandomForest_tuning
======================================================================

- Training Accuracy, 98.0 %
- TestAccuracy, 94.0 %
Variance : 4.0

- Training Accuracy(CrossValidation), 93.0 %
- TestAccuracy(CrossValidation), 89.0 %
Variance (CrossValidation) : 5.0
```





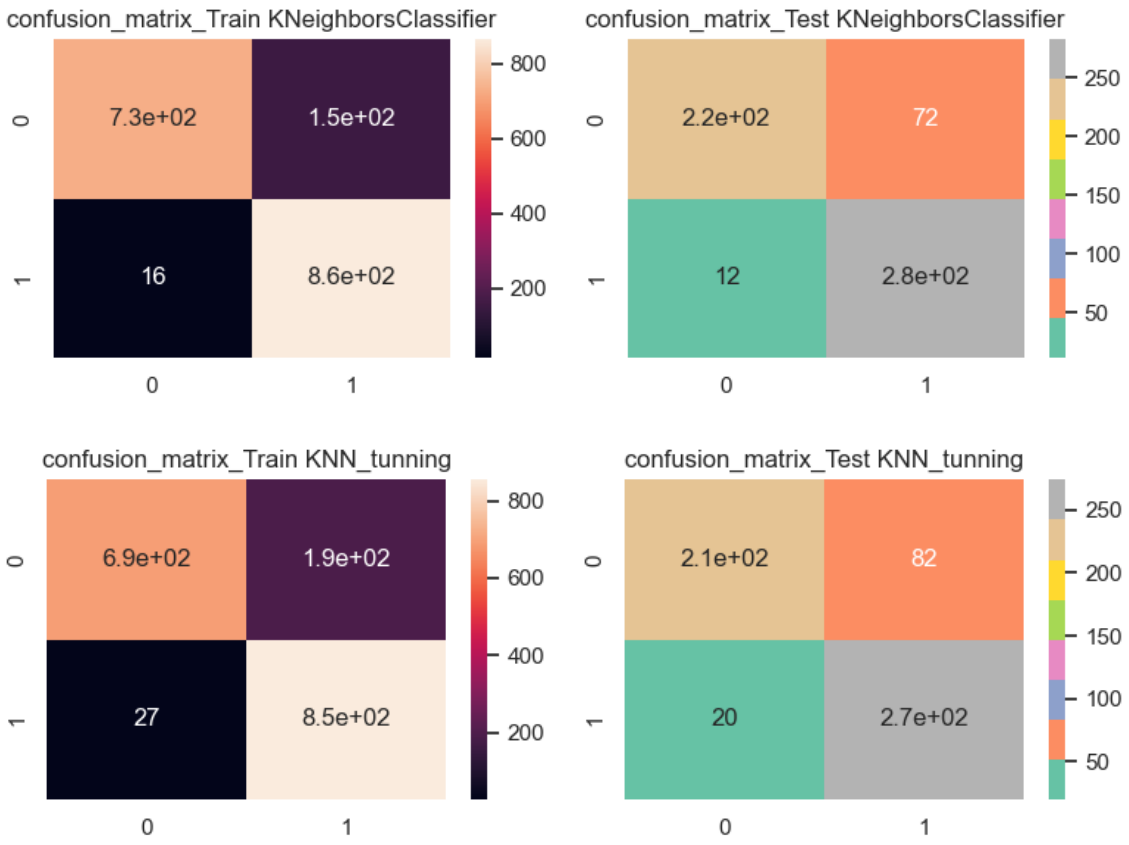
4. KNN

In [35]:

```
for model_name, model in [models[4],models[9]]:

    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
    print('==='*23)
    print(f'           Model name: {model_name}')
    print('==='*23)
    print()

#=================================================
#            Model Accuracy
#-------------------------------------------------
    train_acc = accuracy_score(y_train, y_pred_train)
    test_acc = accuracy_score(y_test, y_pred_test)
    Train_Accuracy.append(train_acc.round(2)*100)
    Test_Accuracy.append(test_acc.round(2)*100)
    variance = abs(train_acc - test_acc).round(2)*100
    print(f'➡ Training Accuracy, {train_acc.round(2)*100} %' )
    print(f'➡ TestAccuracy, {test_acc.round(2)*100} %' )
    print(f'Variance : {variance}')

    Variance.append(variance)
    if (variance > 9) or  (train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')


    print()

#=========================================================
#            Cross Validation
#---------------------------------------------------------
    CV_train_acc =cross_val_score(model, x_train, y_train, cv =10).mean()
    CV_test_acc =cross_val_score(model, x_test, y_test, cv =10).mean()
    Train_CV.append(CV_train_acc.round(2)*100)
    Test_CV.append(CV_test_acc.round(2)*100)
    print(f'➡ Training Accuracy(CrossValidation), {CV_train_acc.round(2)*100} %' )
    print(f'➡ TestAccuracy(CrossValidation), {CV_test_acc.round(2)*100} %' )
    variance_CV = abs(CV_train_acc - CV_test_acc).round(2)*100
    Variance_CV.append(variance_CV)
    print(f'Variance (CrossValidation) : {variance_CV}')
    if (variance_CV > 9) or (CV_train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')
    print()


    cm_train= confusion_matrix(y_train, y_pred_train)
    cm_test = confusion_matrix(y_test, y_pred_test)

#     print('===='*10)
#     print( 'confusion_matrix_Test :','\n','\n',cm_train)
#     print('===='*10)
#     print( 'confusion_matrix_Train :','\n','\n',cm_test)
#     print('===='*10)
    plt.figure(figsize = (8,3))
    plt.subplot(1,2,1)
    sns.heatmap(cm_train, annot = True, )
    plt.title(f'confusion_matrix_Train {model_name}')
    plt.subplot(1,2,2)
```

```
    sns.heatmap(cm_test, annot = True, cmap= 'Set2')
    plt.title(f'confusion_matrix_Test {model_name}')
    plt.tight_layout()
```

```
======================================================================
           Model name: KNeighborsClassifier
======================================================================

- Training Accuracy, 91.0 %
- TestAccuracy, 86.0 %
Variance : 5.0

- Training Accuracy(CrossValidation), 85.0 %
- TestAccuracy(CrossValidation), 78.0 %
Variance (CrossValidation) : 7.000000000000001

======================================================================
           Model name: KNN_tunning
======================================================================

- Training Accuracy, 88.0 %
- TestAccuracy, 83.0 %
Variance : 5.0

- Training Accuracy(CrossValidation), 83.0 %
- TestAccuracy(CrossValidation), 76.0 %
Variance (CrossValidation) : 7.000000000000001
```



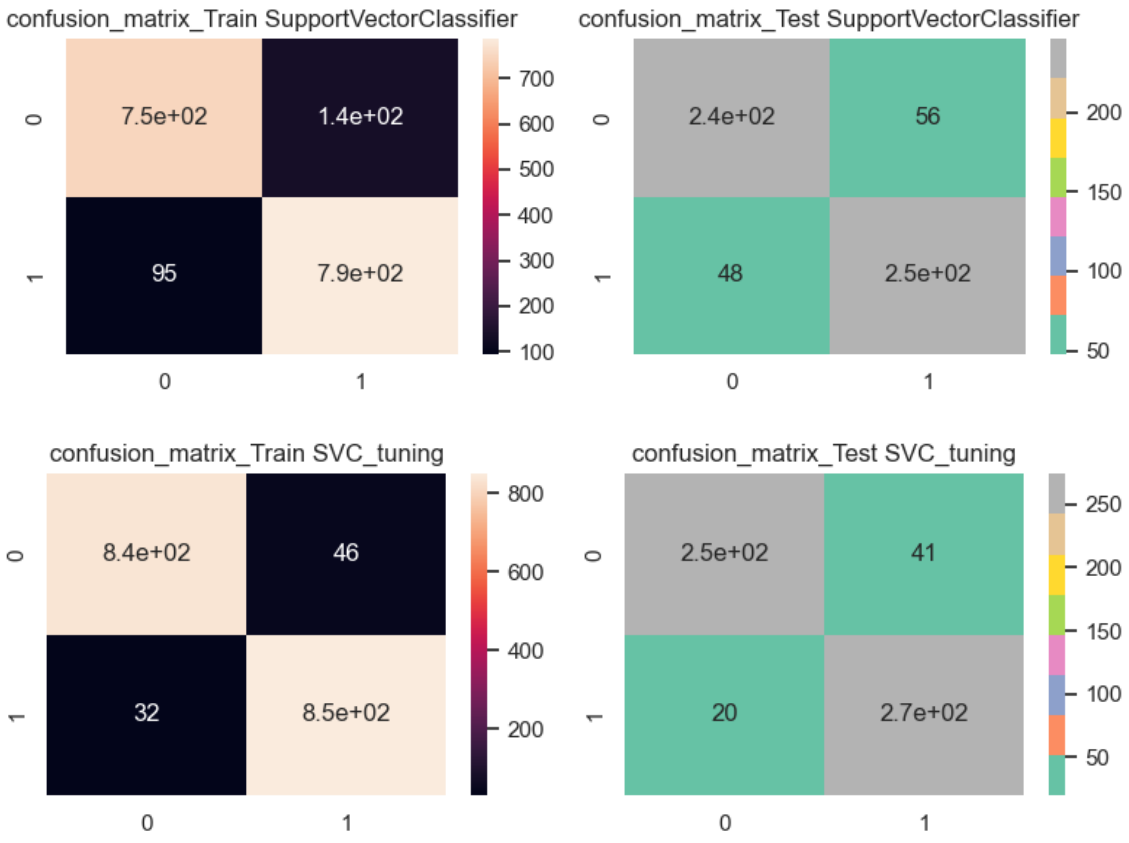
5. SupportVectorClassifier

In [36]:

```
for model_name, model in [models[5],models[10]]:

    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
    print('==='*23)
    print(f'           Model name: {model_name}')
    print('==='*23)
    print()

    #=====================================================
#           Model Accuracy
#-----------------------------------------------------
    train_acc = accuracy_score(y_train, y_pred_train)
    test_acc = accuracy_score(y_test, y_pred_test)
    Train_Accuracy.append(train_acc.round(2)*100)
    Test_Accuracy.append(test_acc.round(2)*100)
    variance = abs(train_acc - test_acc).round(2)*100
    print(f'➡ Training Accuracy, {train_acc.round(2)*100} %' )
    print(f'➡ TestAccuracy, {test_acc.round(2)*100} %' )
    print(f'Variance : {variance}')

    Variance.append(variance)
    if (variance > 9) or  (train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')


    print()

#=========================================================
#           Cross Validation
#---------------------------------------------------------
    CV_train_acc =cross_val_score(model, x_train, y_train, cv =10).mean()
    CV_test_acc =cross_val_score(model, x_test, y_test, cv =10).mean()
    Train_CV.append(CV_train_acc.round(2)*100)
    Test_CV.append(CV_test_acc.round(2)*100)
    print(f'➡ Training Accuracy(CrossValidation), {CV_train_acc.round(2)*100} %' )
    print(f'➡ TestAccuracy(CrossValidation), {CV_test_acc.round(2)*100} %' )
    variance_CV = abs(CV_train_acc - CV_test_acc).round(2)*100
    Variance_CV.append(variance_CV)
    print(f'Variance (CrossValidation) : {variance_CV}')
    if (variance_CV > 9) or (CV_train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')
    print()

    cm_train= confusion_matrix(y_train, y_pred_train)
    cm_test = confusion_matrix(y_test, y_pred_test)

#     print('===='*10)
#     print( 'confusion_matrix_Test :','\n','\n',cm_train)
#     print('===='*10)
#     print( 'confusion_matrix_Train :','\n','\n',cm_test)
#     print('===='*10)
    plt.figure(figsize = (8,3))
    plt.subplot(1,2,1)
    sns.heatmap(cm_train, annot = True, )
    plt.title(f'confusion_matrix_Train {model_name}')
    plt.subplot(1,2,2)
    sns.heatmap(cm_test, annot = True, cmap= 'Set2')
```

```
    plt.title(f'confusion_matrix_Test {model_name}')
    plt.tight_layout()
```

```
======================================================================
           Model name: SupportVectorClassifier
======================================================================

- Training Accuracy, 87.0 %
- TestAccuracy, 82.0 %
Variance : 5.0

- Training Accuracy(CrossValidation), 86.0 %
- TestAccuracy(CrossValidation), 81.0 %
Variance (CrossValidation) : 4.0

======================================================================
           Model name: SVC_tuning
======================================================================

- Training Accuracy, 96.0 %
- TestAccuracy, 90.0 %
Variance : 6.0

- Training Accuracy(CrossValidation), 90.0 %
- TestAccuracy(CrossValidation), 83.0 %
Variance (CrossValidation) : 6.0
```



summary of all models accuracy after cross validation

In [37]:

```
metric_df.set_index('Model', inplace=True)
metric_df.iloc[:,3:6]
```

Out[37]:

Model	Train accuracy(CV)	Test accuracy(CV)	Variance_CV
LogisticRegression	83.0	81.0	2.0
DecisionTreeClassifier	94.0	84.0	10.0
BaggingClassifier	94.0	87.0	7.0
RandomForestClassifier	95.0	89.0	6.0
KNeighborsClassifier	85.0	78.0	7.0
SupportVectorClassifier	86.0	81.0	4.0
DecisionTree_tuning	91.0	83.0	8.0
Bagging_tuning	87.0	78.0	8.0
RandomForest_tuning	94.0	89.0	5.0
KNN_tunning	83.0	76.0	7.0
SVC_tuning	90.0	83.0	6.0

✅ On the basis of accuracy

In [38]:

```
metric_df['Test accuracy(CV)'].sort_values(ascending = False )
```

Out[38]:

```
Model
RandomForestClassifier     89.0
RandomForest_tuning        89.0
BaggingClassifier          87.0
DecisionTreeClassifier     84.0
DecisionTree_tuning        83.0
SVC_tuning                 83.0
LogisticRegression         81.0
SupportVectorClassifier    81.0
KNeighborsClassifier       78.0
Bagging_tuning             78.0
KNN_tunning                76.0
Name: Test accuracy(CV), dtype: float64
```

🥇🏆 RandomForestClassifier is giving a test accuracy of 88% after cross validation 🏆🥇

✅❎ On the basis of misclassification:

In [39]:

```
metric_df[['FP','FN']].sort_values( by=['FP','FN'])
```

Out[39]:

Model	FP	FN
BaggingClassifier	22	0
RandomForestClassifier	22	0
RandomForest_tuning	33	0
DecisionTreeClassifier	37	0
SVC_tuning	41	20
DecisionTree_tuning	44	5
SupportVectorClassifier	56	48
LogisticRegression	61	43
Bagging_tuning	68	6
KNeighborsClassifier	72	12
KNN_tunning	82	20

RandomForest is classifying true cases efficiently having less

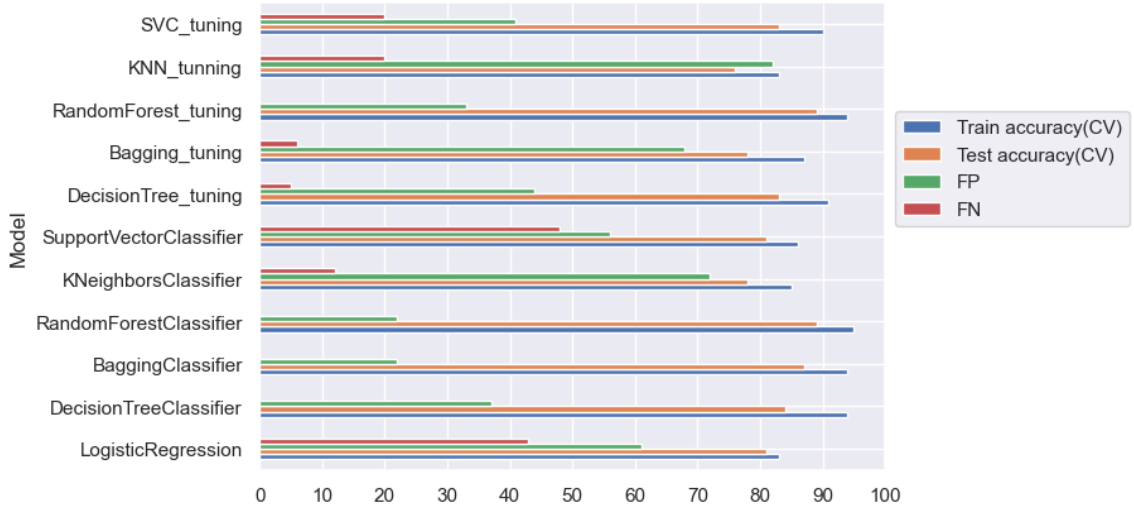
number of misclassification.

> **- FN =1, The number of samples incorrectly classified as "bad" when they are actually "good".**
>
> **- FP = 19 . The number of samples incorrectly classified as "good"**

📈 📊 📉 Visulisation of the performances of differnt models 📈 📊 📉

In [40]:

```
ax = metric_df.loc[:,['Train accuracy(CV)','Test accuracy(CV)','FP','FN']].plot(kind='ba
plt.legend(loc='lower left', bbox_to_anchor=(1.0, 0.5))
ax.set_xticks(range(0, 110, 10))
plt.show()
```



Summary :

- All the models perform nicely with and without hyper-parameter tunning.

- `Logistic Regression` model achieved roughly 80% train and test accuracy after cross validation.

- `DecisionTree` got same accuracy with and without tuning, however the misclassification of classes got incrased after tuning.

After tuning, `Bagging Classifier` 's accuracy decreased and the misclassification of classes got incrased.

- `Random Forest` Classifer performs well for this dataset distinguishing the true calsses efficiently and poor at misclassification and achieved a good accuracy as well.

-Additionally, after cross-validation, the accuracy achieved on the test data is 88% both with and without hyperparameter tuning.

- The Area Under the Curve (AUC) is the highest for the Random Forest Classifier.

- `KNN` model behaves well before tuning. It might have some scope of improvement with further tuning.

- `SupportVectorClassifier` model is performing better with tunning. The accuracy of the model got enhanced and misclassification got reduced.

By - Payal Mohaty, Project completed - 4th August 2023