Приложение 1

Курсовая работа БВТ2003 Глазков Даниил

In [1]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import ListedColormap
import numpy as np
import pandas as pd
import sklearn
from itertools import cycle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_a
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import RocCurveDisplay
from sklearn.preprocessing import label_binarize
from scipy import interp
from sklearn.exceptions import NotFittedError
from sklearn.decomposition import PCA
import time as time
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier as KNN
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import preprocessing
plt.rc("font", size=14)
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)
palette = sns.color_palette("Spectral")
import warnings
warnings.simplefilter(action='ignore')
```

In [2]:
```
train_df = pd.read_csv("./winequality-red.csv")
train_df.head(10)
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10

In [3]:
```
train_df.shape[0]
```

Out[3]:
```
1599
```

In [4]:
```
print('train_df objects: \n\n',train_df.count())
```

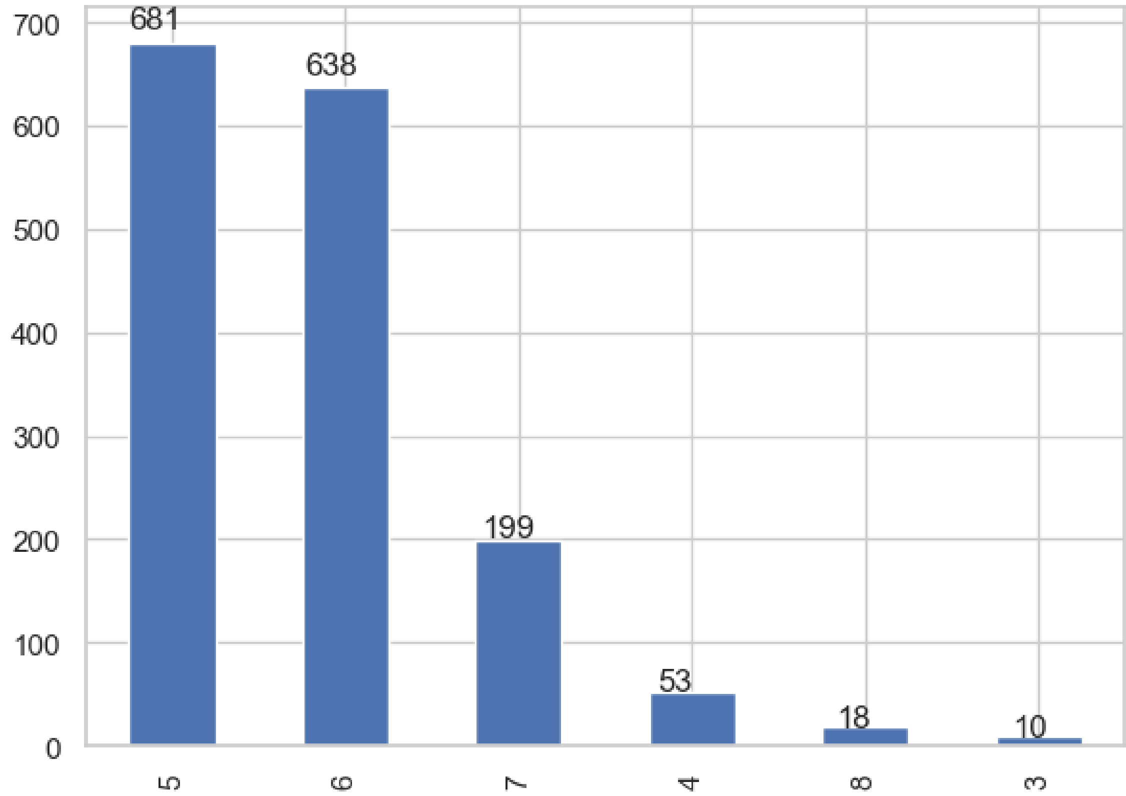
```
train_df objects: 

 fixed acidity           1599
volatile acidity        1599
citric acid             1599
residual sugar          1599
chlorides               1599
free sulfur dioxide     1599
total sulfur dioxide    1599
density                 1599
pH                      1599
sulphates               1599
alcohol                 1599
quality                 1599
dtype: int64
```

In [5]:
```
train_df.isna().sum()
```

Out[5]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

In [6]:
```
ax = train_df["quality"].value_counts().plot.bar(figsize=(7,5))
for p in ax.patches:
    ax.annotate(str(p.get_height()), (p.get_x() * 1.02, p.get_height() * 1.02))

print(train_df["quality"].value_counts(normalize=True)*100)
```

```
5    42.589118
6    39.899937
7    12.445278
4     3.314572
8     1.125704
3     0.625391
Name: quality, dtype: float64
```



In [7]:
```
train_df["is good"] = 0
train_df.loc[train_df["quality"]>=7,"is good"] = 1
train_df
```

Out[7]:

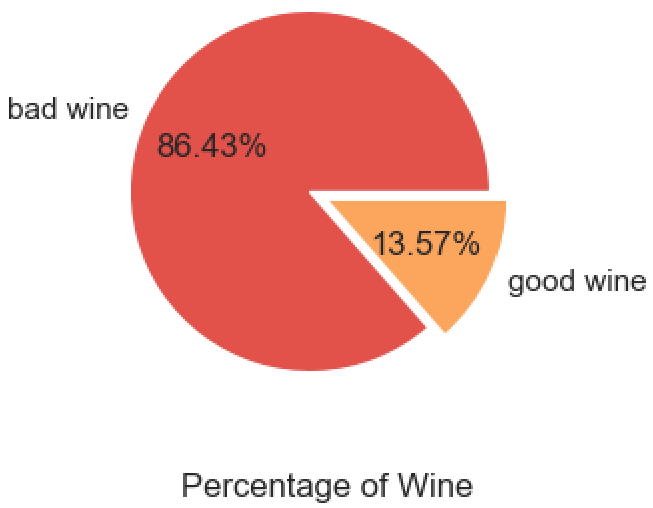
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	a
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	
3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

1599 rows × 13 columns

In [8]:

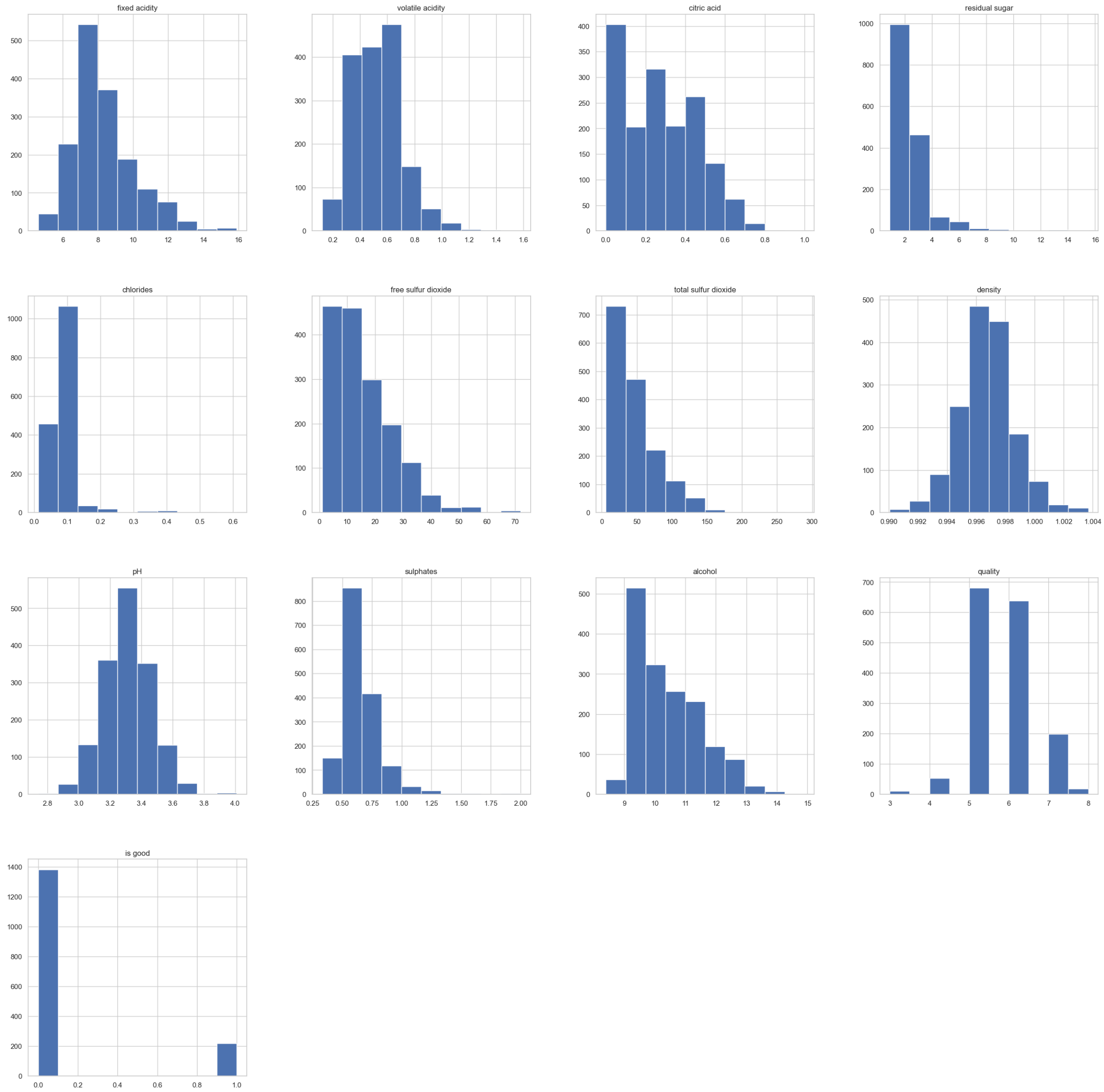
```
wineScale_insight = train_df['is good']

plt.subplot(1, 2, 2)
plt.pie(wineScale_insight.value_counts().values, labels=['bad wine', 'good wine'
plt.xlabel("\nPercentage of Wine")

print(train_df["is good"].value_counts(normalize=True)*100)
```

```
0    86.429018
1    13.570982
Name: is good, dtype: float64
```



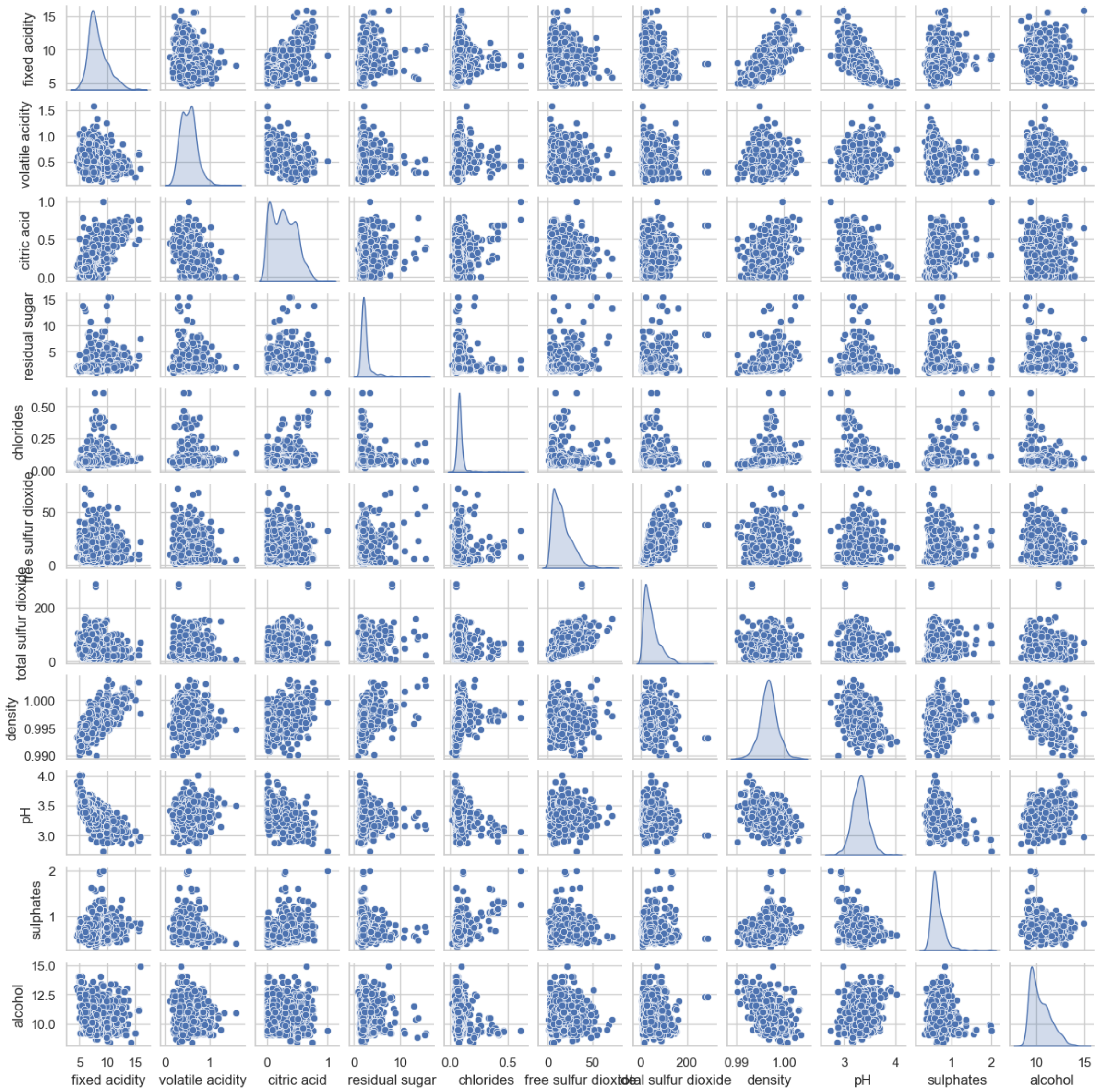
In [10]:

```
train_df.hist(figsize=(30,30));
```

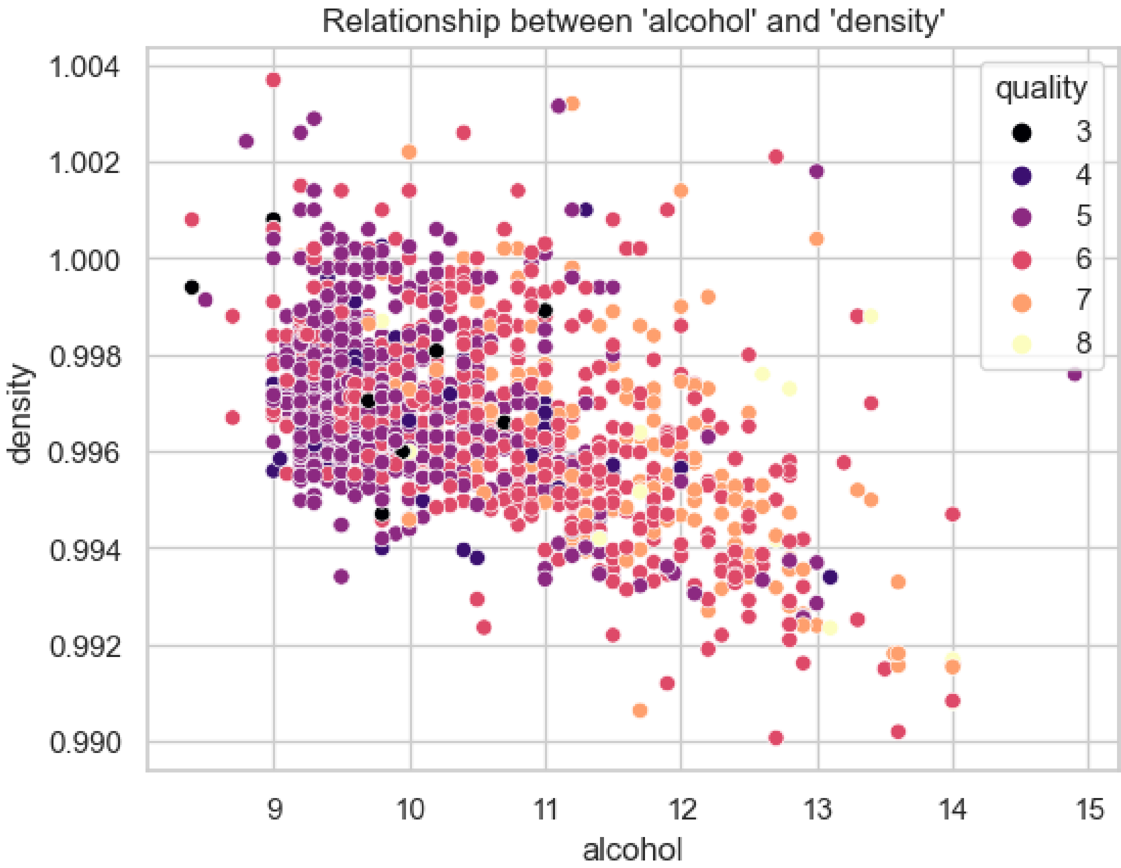


In [11]:
```
features = train_df.columns[:-2]
output = train_df.columns[-1]
```

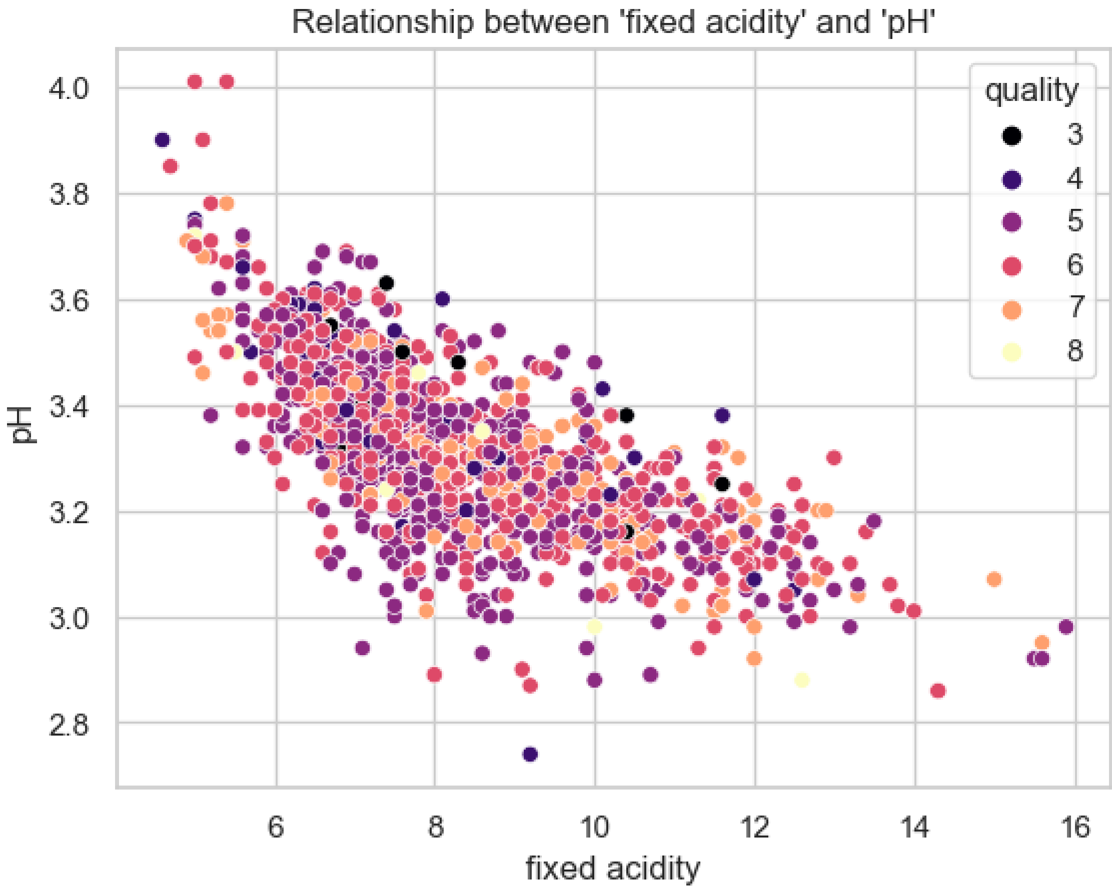
In [12]:
```
sns.pairplot(train_df[features],palette='coolwarm', size=1.2, diag_kind='kde')
plt.show()
```



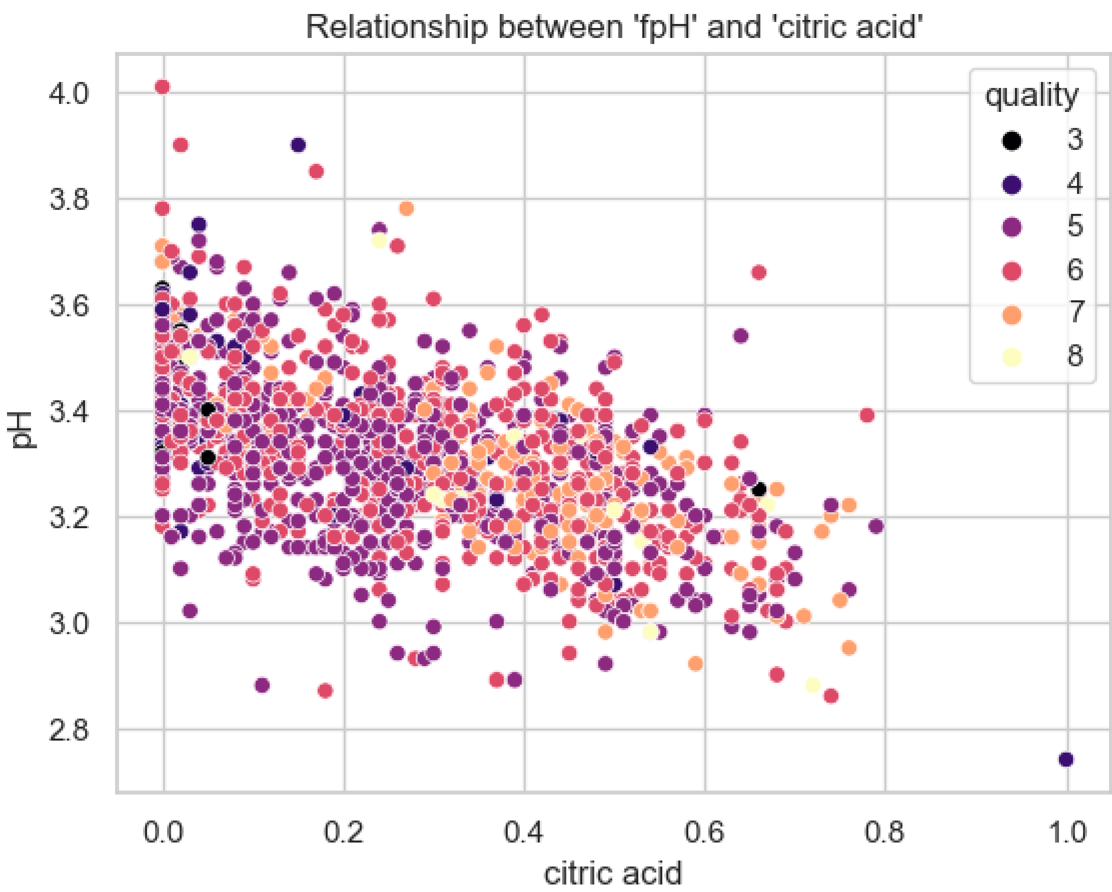
In [13]:
```
sns.scatterplot(x = "alcohol",
                y = "density",
                hue = "quality",
                palette = "magma",
                data = train_df).set(title = "Relationship between 'alcohol' anc
```



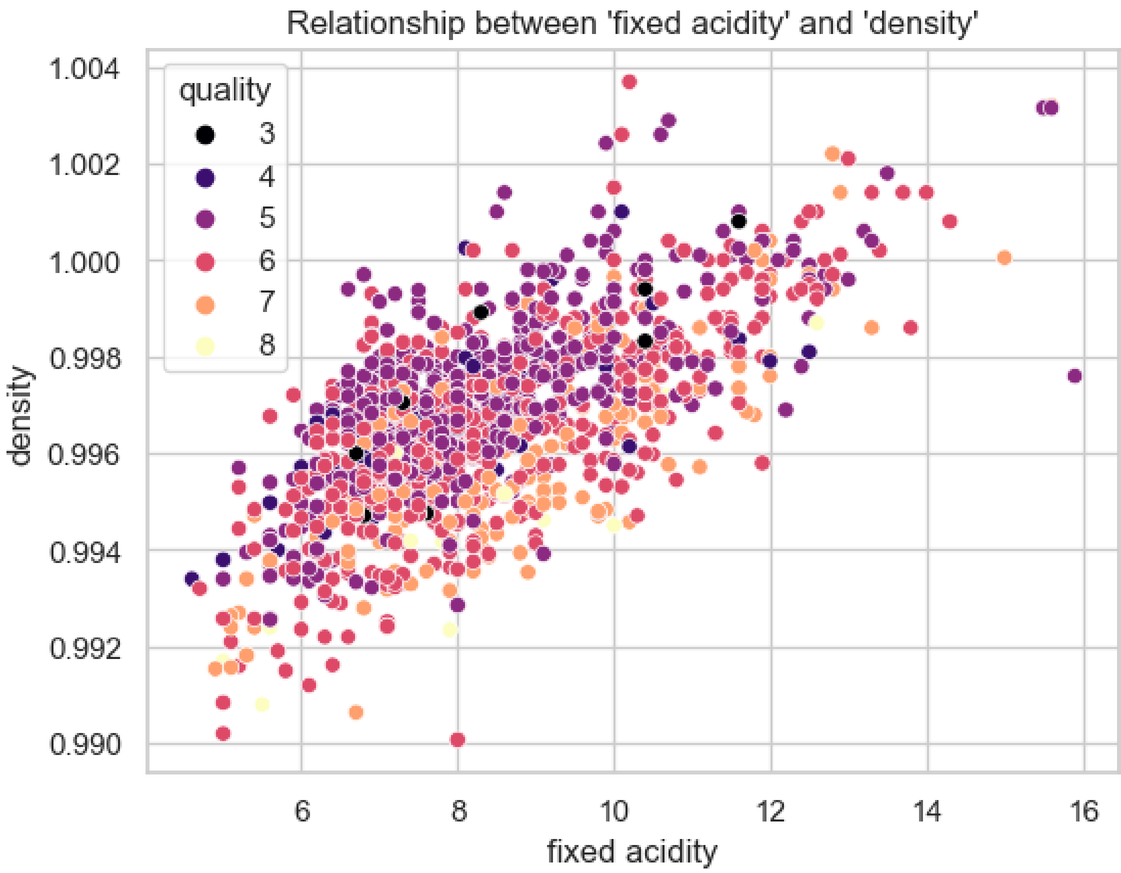
In [14]:
```
sns.scatterplot(x = "fixed acidity",
                y = "pH",
                hue = "quality",
                palette = "magma",
                data = train_df).set(title = "Relationship between 'fixed acidit
```



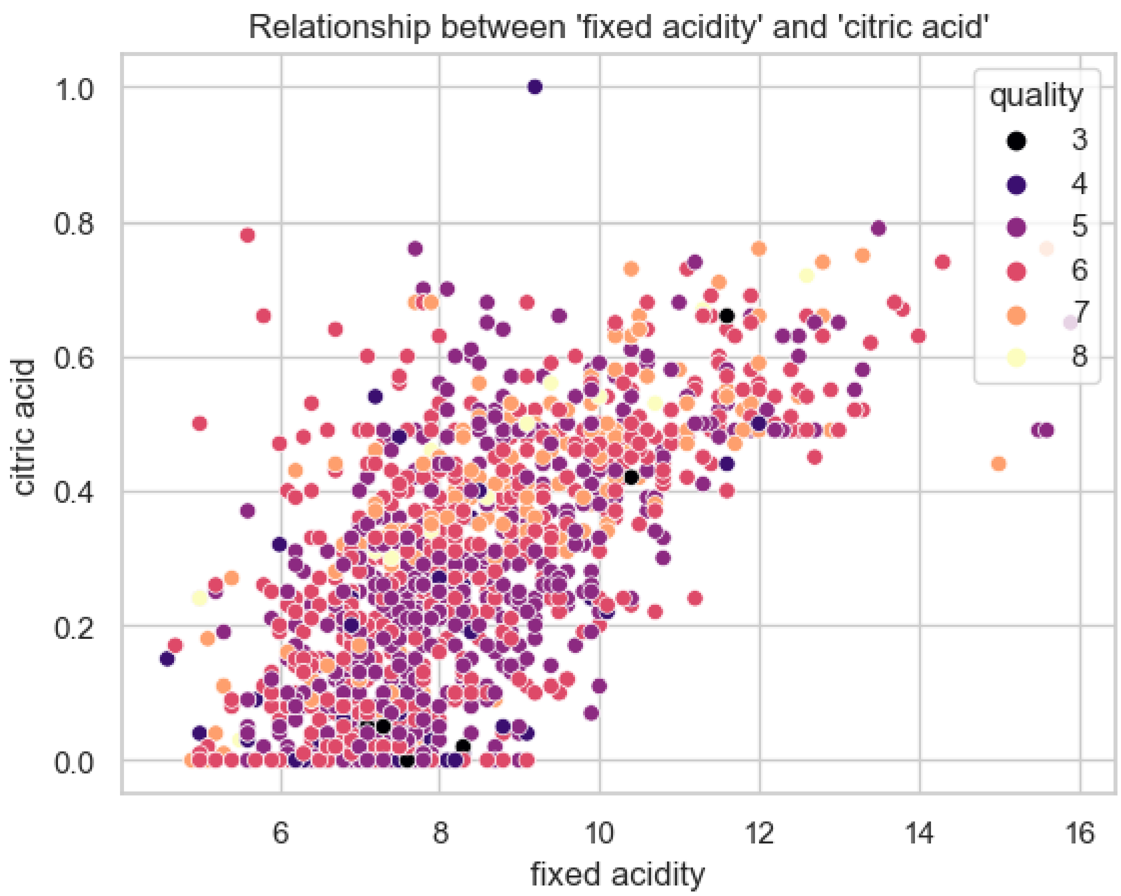
In [15]:
```
sns.scatterplot(x = "citric acid",
                y = "pH",
                hue = "quality",
                palette = "magma",
                data = train_df).set(title = "Relationship between 'fpH' and 'ci
```



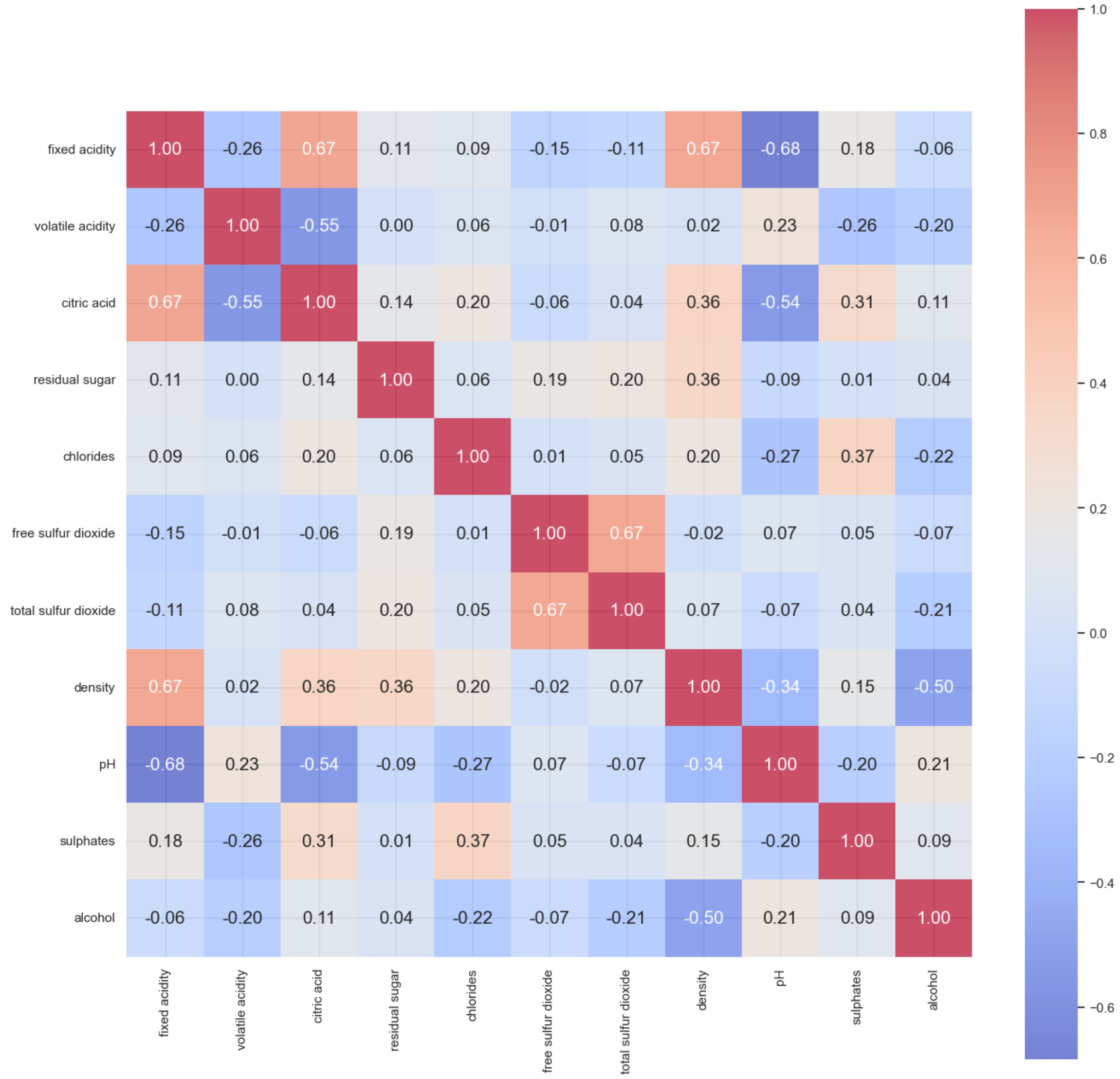
In [16]:
```
sns.scatterplot(x = "fixed acidity",
                y = "density",
                hue = "quality",
                palette = "magma",
                data = train_df).set(title = "Relationship between 'fixed acidit
```



In [17]:
```
sns.scatterplot(x = "fixed acidity",
                y = "citric acid",
                hue = "quality",
                palette = "magma",
                data = train_df).set(title = "Relationship between 'fixed acidit
```



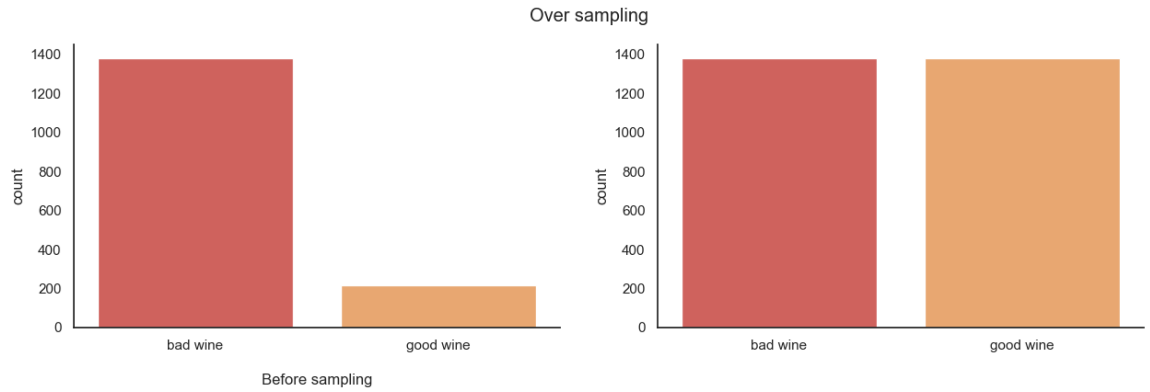
In [18]:
```
corr = train_df[features].corr()
plt.figure(figsize=(16,16))
sns.heatmap(corr, cbar = True,  square = True, annot=True, fmt= '.2f',annot_kws=
            xticklabels= features, yticklabels= features, alpha = 0.7,   cmap= 'c
plt.show()
```



In [19]:
```
X = train_df[features].values
y = train_df[output].values
```

In [20]:
```
from imblearn.over_sampling import SMOTE

smote = SMOTE()
X_smote, y_smote = smote.fit_resample(X, y)
```

In [21]:
```
sns.set_style("white")
plt.figure(figsize=(15, 4))

plt.subplot(1, 2, 1)
sns.countplot(data=train_df, x='is good', palette=palette)
plt.xticks([0,1], ['bad wine', 'good wine'])
plt.xlabel("\nBefore sampling")

plt.subplot(1, 2, 2)
sns.countplot(x=y_smote, palette=palette)
plt.xticks([0,1], ['bad wine', 'good wine'])


plt.suptitle('Over sampling \n\n\n')

sns.despine()
```



Разбиение датасета

In [24]:
```
import numpy as np
from sklearn.model_selection import KFold,StratifiedKFold
sfolder = StratifiedKFold(n_splits=2, shuffle=True, random_state=42)
cnt = 1
for train, test in sfolder.split(X_smote,y_smote):
    print(f'Fold:{cnt}, Train set: {len(train)}, Test set:{len(test)}')
    cnt += 1
    X2_train, X2_test = X_smote[train], X_smote[test]
    y2_train, y2_test = y_smote[train], y_smote[test]
```

```
Fold:1, Train set: 1382, Test set:1382
Fold:2, Train set: 1382, Test set:1382
```

In [25]:
```
sc_X2 = StandardScaler()
X2_train = sc_X2.fit_transform(X2_train)
X2_test = sc_X2.transform(X2_test)
times_array2 = {}
```

In [27]:
```
def get_classification_report(y_test,predictions,average="macro"):

    acc = accuracy_score(y_test, predictions)
    pre = precision_score(y_test, predictions, average=average)
    rec = recall_score(y_test, predictions, average=average)
    # Prediction Report
    print(classification_report(y_test, predictions, digits=3))
    print("Overall Accuracy:", acc)
    print("Overall Precision:", pre)
    print("Overall Recall:", rec)

    return acc,pre,rec
```

In [28]:
```
def get_classification_ROC(X,y,model,test_size,model_fitted=False,random_state=0

    def check_fitted(clf):
        return hasattr(clf, "classes_")

    if(len(np.unique(y)) == 2):
        #Binary Classifier
        if not check_fitted(model):
            model = model.fit(X,y)

        RocCurveDisplay.from_estimator(model, X, y)
        y_score = model.predict_proba(X)[:, 1]
        fpr, tpr, threshold = roc_curve(y, y_score)
```

```
        auc = roc_auc_score(y, y_score)
        return auc
#         print("False Positive Rate: {} \nTrue Positive Rate: {} \nThreshold:{}

    else:
        #Multiclass Classifier
        y_bin = label_binarize(y, classes=np.unique(y))
        n_classes = y_bin.shape[1]

        # shuffle and split training and test sets
        X_train, X_test, y_train, y_test = train_test_split(X, y_bin, test_size=

        # Learn to predict each class against the other
        classifier = OneVsRestClassifier(model)
        model_fitted = classifier.fit(X_train, y_train)
        try:
            y_score = model_fitted.decision_function(X_test)
        except:
            y_score = model_fitted.predict_proba(X_test)



        # Compute ROC curve and ROC area for each class
        fpr = dict()
        tpr = dict()
        roc_auc = dict()
        for i in range(n_classes):
            fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
            roc_auc[i] = auc(fpr[i], tpr[i])


        # Compute micro-average ROC curve and ROC area
        fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel(
        roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])


        plt.figure()
        lw = 2
        plt.plot(fpr[2], tpr[2], color='darkorange',
                 lw=lw, label='ROC curve (area = %0.2f)' % roc_auc[2])
        plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver operating characteristic averaged')
        plt.legend(loc="lower right")
        plt.show()



        # First aggregate all false positive rates
        all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))

        # Then interpolate all ROC curves at this points
        mean_tpr = np.zeros_like(all_fpr)
        for i in range(n_classes):
            mean_tpr += interp(all_fpr, fpr[i], tpr[i])

        # Finally average it and compute AUC
```

```
        mean_tpr /= n_classes

        fpr["macro"] = all_fpr
        tpr["macro"] = mean_tpr
        roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

        # Plot all ROC curves
        plt.figure(figsize=(10,10))
        plt.plot(fpr["micro"], tpr["micro"],
                 label='micro-average ROC curve (area = {0:0.2f})'
                       ''.format(roc_auc["micro"]),
                 color='deeppink', linestyle=':', linewidth=4)

        plt.plot(fpr["macro"], tpr["macro"],
                 label='macro-average ROC curve (area = {0:0.2f})'
                       ''.format(roc_auc["macro"]),
                 color='navy', linestyle=':', linewidth=4)

        colors = cycle(['aqua', 'darkorange', 'cornflowerblue', 'red', 'blue', '
        for i, color in zip(range(n_classes), colors):
            plt.plot(fpr[i], tpr[i], color=color, lw=lw,
                     label='ROC curve of class {0} (area = {1:0.2f})'
                     ''.format(i, roc_auc[i]))

        plt.plot([0, 1], [0, 1], 'k--', lw=lw)
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title('multi-class ROC (One vs All)')
        plt.legend(loc="lower right")
        plt.show()
```

SVC

In [32]:
```
t11 = time.time_ns()
model31 = SVC(probability=True)
model31.fit(X2_train, y2_train)
times_array2["SVC"] = time.time_ns()-t11
```

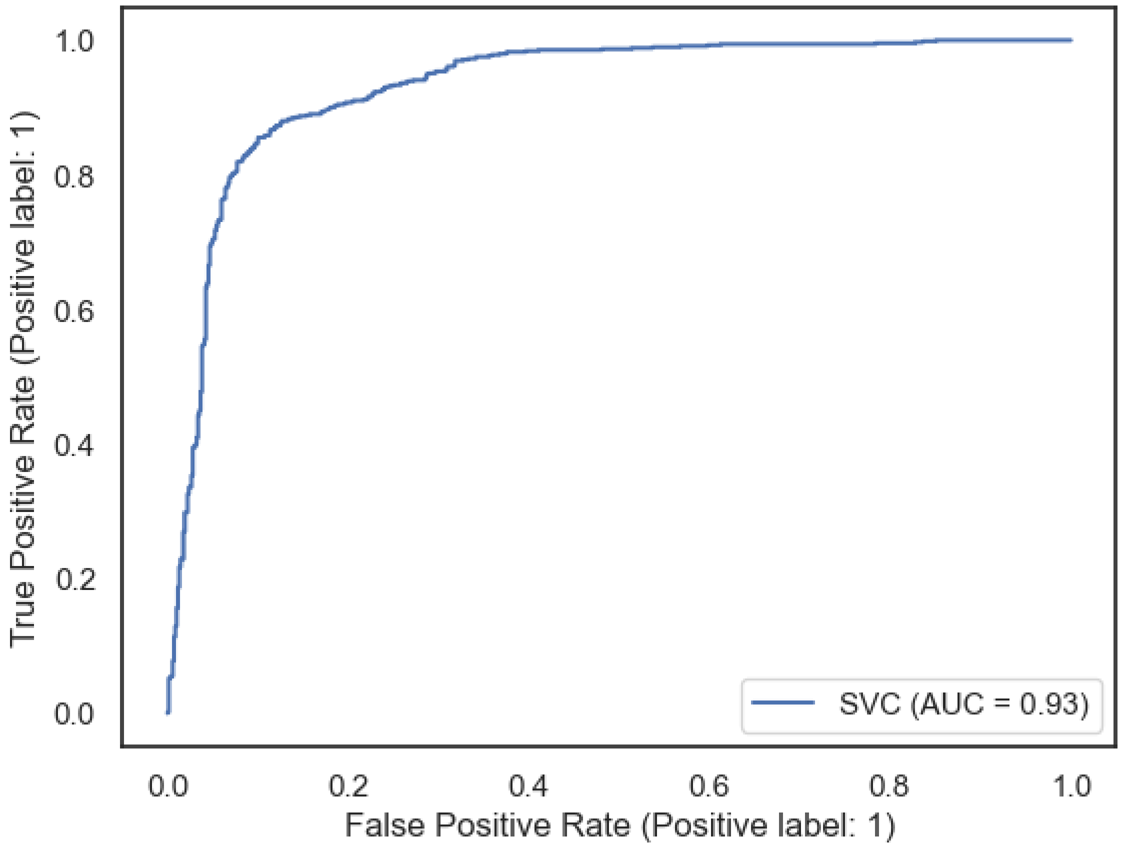
In [33]:
```
predictions_SVC2 = model31.predict(X2_test)
print("Predictions:",predictions_SVC2[:10])
print("Actual:",y2_test[:10])
```

```
Predictions: [0 0 0 0 0 0 0 0 0 0]
Actual: [0 0 0 1 0 0 0 0 1 0]
```

In [34]:
```
acc_SVC,pre_SVC,rec_SVC = get_classification_report(y2_test,predictions_SVC2)
```

```
              precision    recall  f1-score   support

           0      0.894     0.805     0.847       691
           1      0.822     0.904     0.861       691

    accuracy                          0.855      1382
   macro avg      0.858     0.855     0.854      1382
weighted avg      0.858     0.855     0.854      1382

Overall Accuracy: 0.8545586107091172
Overall Precision: 0.8581295481468945
Overall Recall: 0.8545586107091172
```

In [35]:
```
auc_SVC = get_classification_ROC(X2_test,y2_test,model31,test_size=0.3,random_st
```



KNN

In [36]:
```
from sklearn.neighbors import KNeighborsClassifier

t12 = time.time_ns()
model41=KNeighborsClassifier(n_neighbors=7)
model41.fit(X2_train,y2_train)
times_array2["KNN"] = time.time_ns()-t12
```

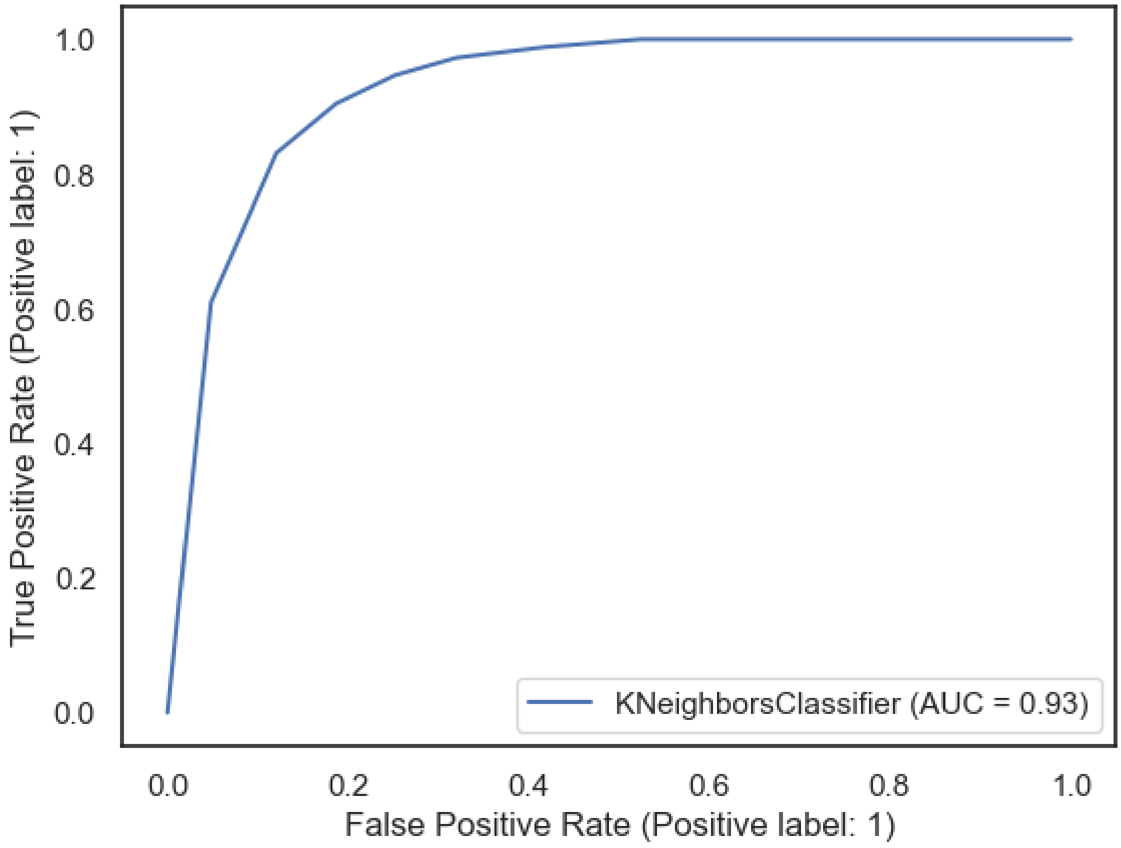
In [37]:
```
predictions_KNN2 = model41.predict(X2_test)
print("Predictions:",predictions_KNN2[:10])
print("Actual:",y2_test[:10])
```

```
Predictions: [0 0 0 0 0 0 1 0 0 0]
Actual: [0 0 0 1 0 0 0 0 1 0]
```

In [38]:
```
acc_KNN,pre_KNN,rec_KNN = get_classification_report(y2_test,predictions_KNN2)
```

```
              precision    recall  f1-score   support

           0      0.933     0.748     0.831       691
           1      0.790     0.946     0.861       691

    accuracy                          0.847      1382
   macro avg      0.862     0.847     0.846      1382
weighted avg      0.862     0.847     0.846      1382

Overall Accuracy: 0.8473227206946454
Overall Precision: 0.8615340344268299
Overall Recall: 0.8473227206946454
```

In [39]:
```
auc_SVC = get_classification_ROC(X2_test,y2_test,model41,test_size=0.3,random_st
```



GBC

In [40]:
```
from sklearn.ensemble import GradientBoostingClassifier

t14 = time.time_ns()
model71=GradientBoostingClassifier()
model71.fit(X2_train,y2_train)
times_array2["GradientBoostingClassifier"] = time.time_ns()-t14
```

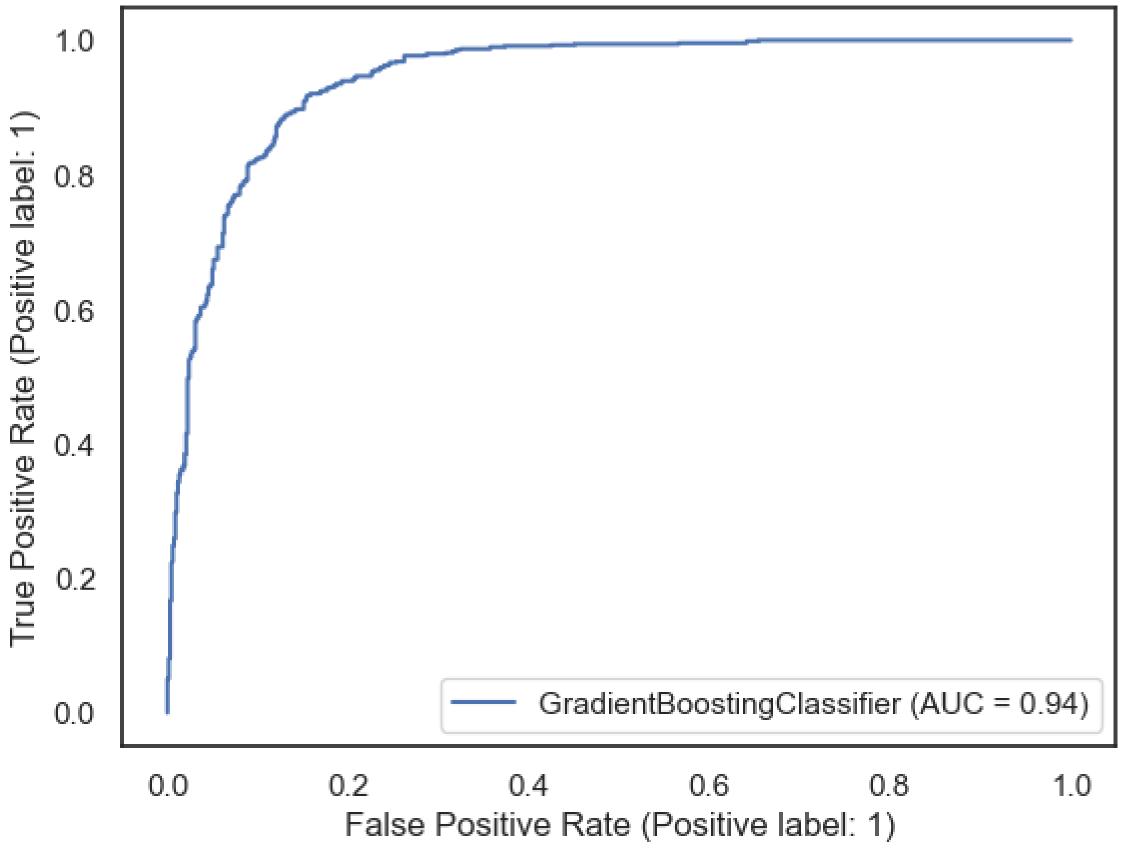
In [41]:
```
predictions_GBC2 = model71.predict(X2_test)
print("Predictions:",predictions_GBC2[:10])
print("Actual:",y2_test[:10])
```

```
Predictions: [0 0 0 0 0 0 0 0 0 0]
Actual: [0 0 0 1 0 0 0 0 1 0]
```

In [42]:
```
acc_GBC,pre_GBC,rec_GBC = get_classification_report(y2_test,predictions_GBC2)
```

```
              precision    recall  f1-score   support

           0      0.910     0.847     0.877       691
           1      0.857     0.916     0.885       691

    accuracy                          0.881      1382
   macro avg      0.883     0.881     0.881      1382
weighted avg      0.883     0.881     0.881      1382

Overall Accuracy: 0.8813314037626628
Overall Precision: 0.8831803727874035
Overall Recall: 0.8813314037626627
```

In [43]:
```
auc_GBC = get_classification_ROC(X2_test,y2_test,model71,test_size=0.3,random_st
```



NB

In [44]:
```
t51 = time.time_ns()
model_NB1 = GaussianNB()
model_NB1.fit(X2_train, y2_train)
times_array2["NB"] = time.time_ns()-t51
```

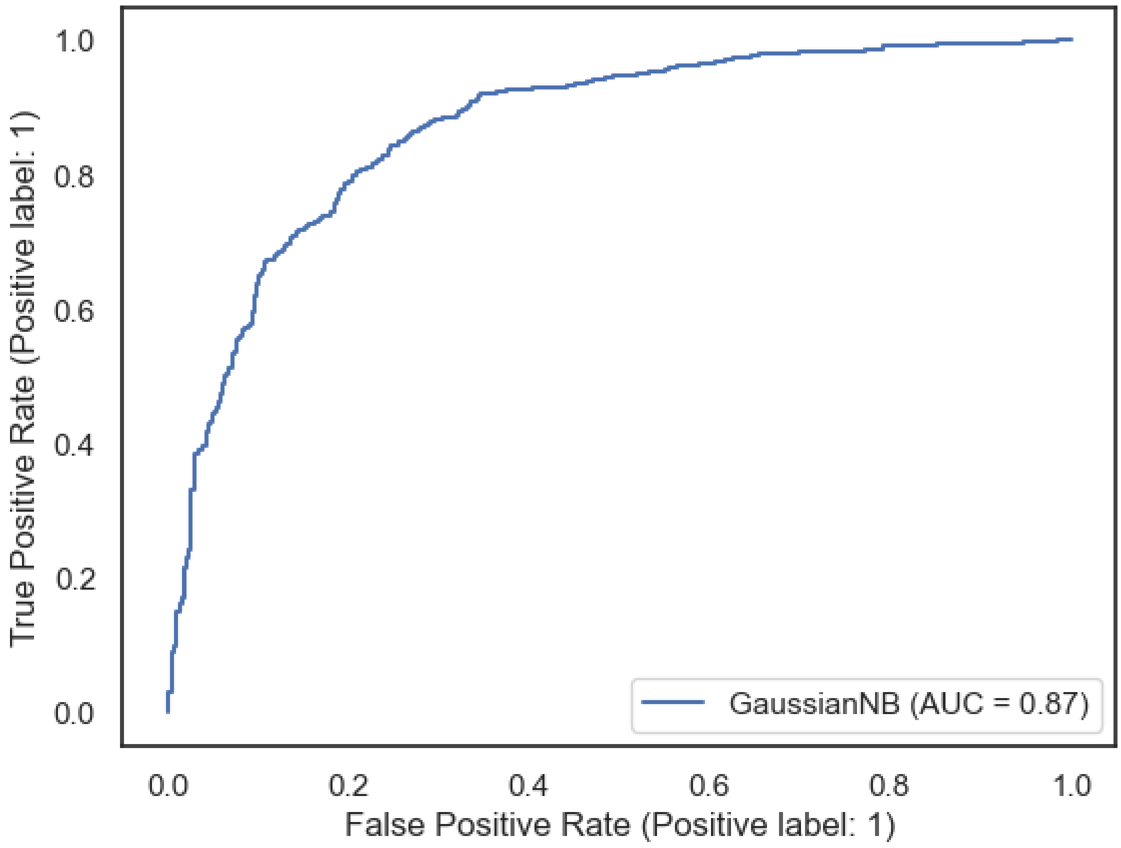
In [45]:
```
predictions_NB1 = model_NB1.predict(X2_test)
print("Predictions:",predictions_NB1[:10])
print("Actual:",y2_test[:10])
```

```
Predictions: [0 1 0 0 0 0 0 0 1 0]
Actual: [0 0 0 1 0 0 0 0 1 0]
```

In [46]:
```
acc_NB1,pre_NB1,rec_NB1 = get_classification_report(y2_test,predictions_NB1)
```

```
              precision    recall  f1-score   support

           0      0.844     0.726     0.781       691
           1      0.760     0.865     0.809       691

    accuracy                          0.796      1382
   macro avg      0.802     0.796     0.795      1382
weighted avg      0.802     0.796     0.795      1382

Overall Accuracy: 0.7959479015918958
Overall Precision: 0.8017725006139687
Overall Recall: 0.7959479015918958
```

In [47]:
```
auc_NB1 = get_classification_ROC(X2_test,y2_test,model_NB1,test_size=0.3,random_
```



LR

In [48]:
```
t61= time.time_ns()
model_LR1 = LogisticRegression()
model_LR1.fit(X2_train, y2_train)
times_array2["LR"]= time.time_ns()-t61
```

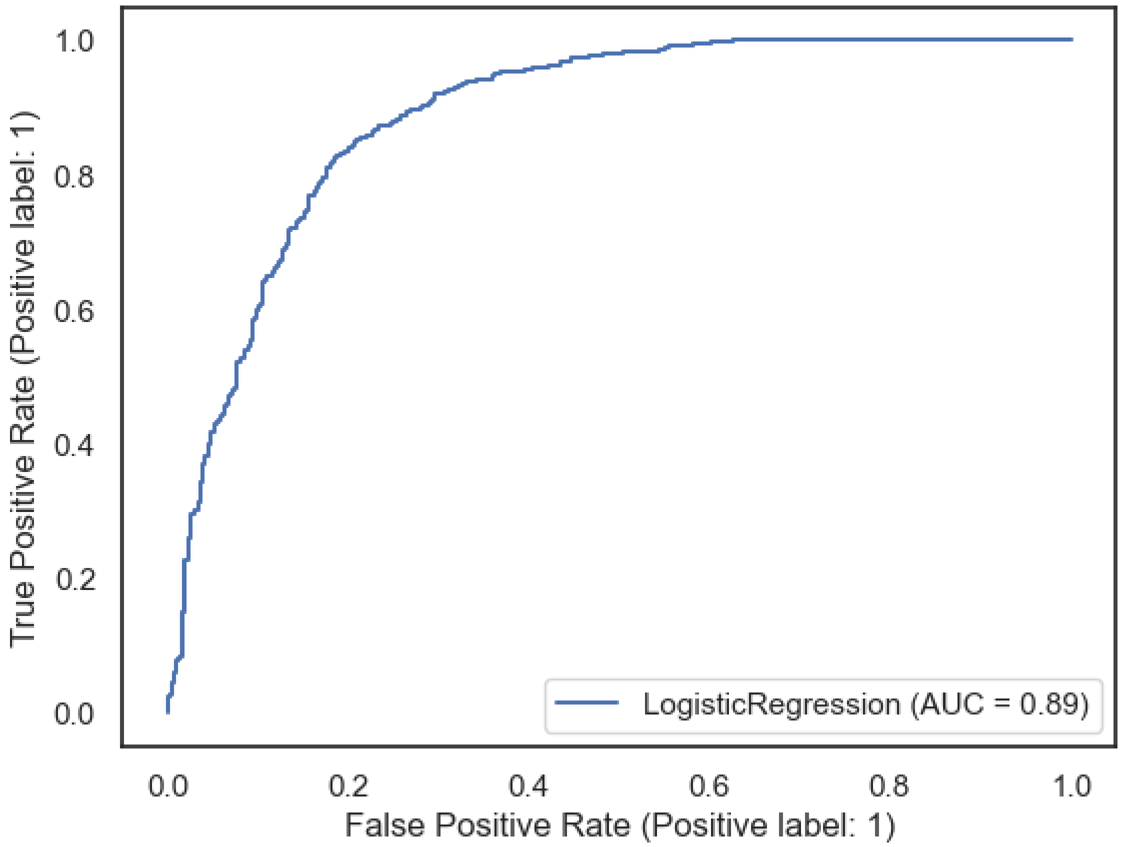
In [49]:
```
predictions_LR1 = model_LR1.predict(X2_test)
print("Predictions:",predictions_LR1[:10])
print("Actual:",y2_test[:10])
```

```
Predictions: [0 0 0 0 0 0 1 0 0 0]
Actual: [0 0 0 1 0 0 0 0 1 0]
```

In [50]:
```
acc_LR1,pre_LR1,rec_LR1 = get_classification_report(y2_test,predictions_LR1)
```

```
              precision    recall  f1-score   support

           0      0.842     0.787     0.814       691
           1      0.800     0.852     0.826       691

    accuracy                          0.820      1382
   macro avg      0.821     0.820     0.820      1382
weighted avg      0.821     0.820     0.820      1382

Overall Accuracy: 0.8198263386396527
Overall Precision: 0.8211885011441648
Overall Recall: 0.8198263386396527
```

In [51]:
```
auc_LR1 = get_classification_ROC(X2_test,y2_test,model_LR1,test_size=0.3,random_
```



In [52]:
```
print(times_array2)
```

```
{'SVC': 109581200, 'KNN': 1753400, 'GradientBoostingClassifier': 467796500, 'N
B': 1500200, 'LR': 10759600}
```

In [53]:
```
print("SVC2:",model31.score(X2_train,y2_train)*100)
print("KNN2:",model41.score(X2_train,y2_train)*100)
print("GBC2:",model71.score(X2_train,y2_train)*100)
print("NB2:",model_NB1.score(X2_train,y2_train)*100)
print("LR2:",model_LR1.score(X2_train,y2_train)*100)
```

```
SVC2: 89.36324167872648
KNN2: 88.5672937771346
GBC2: 95.58610709117221
NB2: 80.8972503617945
LR2: 83.21273516642546
```