Supervised Learning:

Predicting Wine Quality

Classification Problem

Nariman Pashayev

Contents

- Main Objectives
- Data Description
- Data Exploration And Analysis
- Machine Learning Analysis and Modelling
- Key Findings and Recommendations

DATA DESCRIPTION AND ANALYSIS SECTION

Main Objectives

- Two datasets are included, related to red and white vinho verde wine samples, from the north of Portugal.
- The goal is to model (predict) wine quality based on physicochemical tests by using ML classification techniques
- The dataset was downloaded from the UCI Machine Learning Repository, alsa available in Kaggle.
- https://www.kaggle.com/code/narimanpashayev/predicting-wine-quality-with-different-models/data

Data Description

- The Wine data set contains 13 attributes
- Data types are 11 floats, 1 integer, and 1 object type
- There are a total of 6497 rows in the dataset
- The goal is to predict wine quality based on its attributes
- The quality range is 3-9.
- So I decided [2-6] as a bad quality wine and (6-10] good quality wine

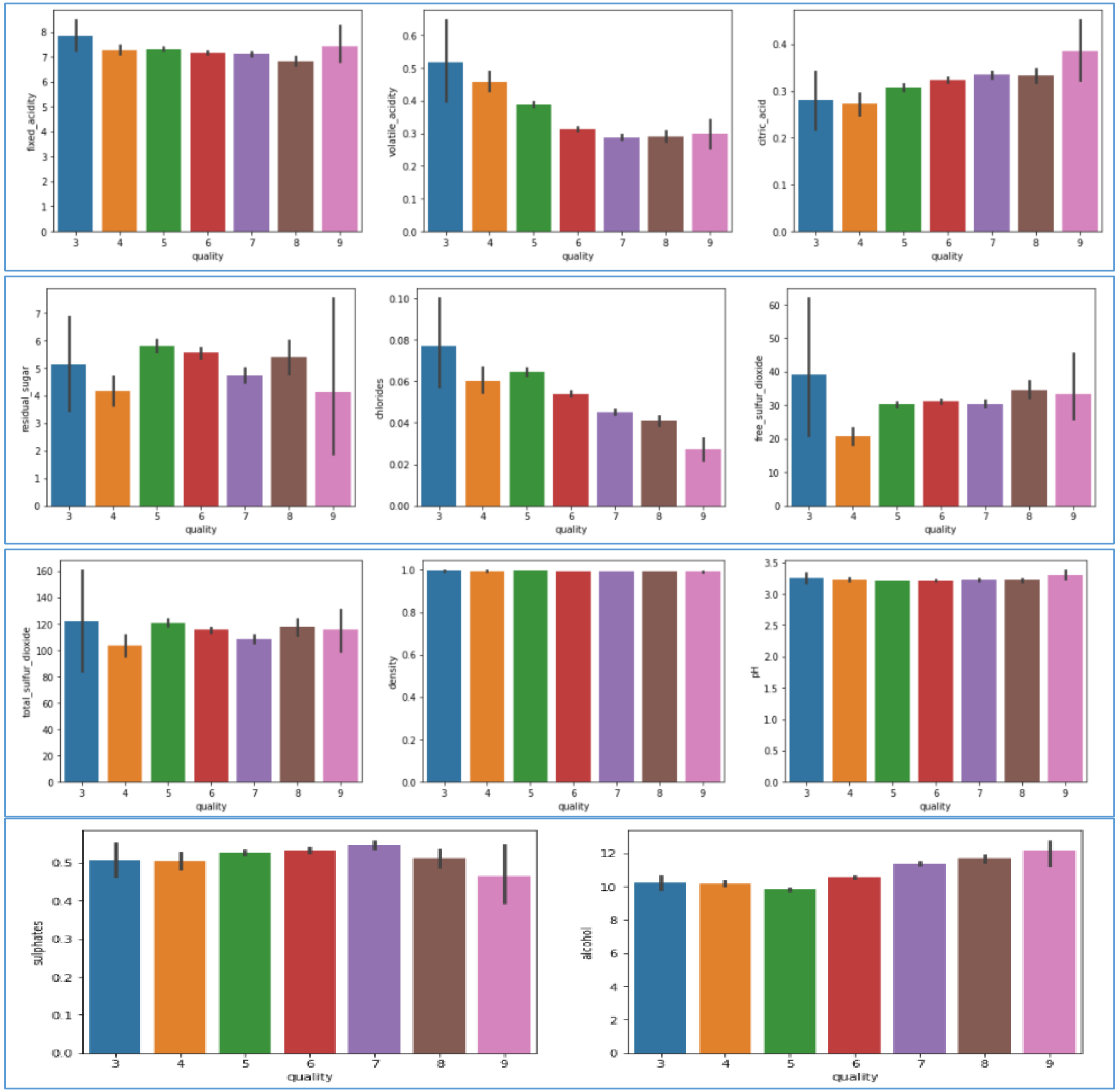
```
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed_acidity         6497 non-null   float64
 1   volatile_acidity      6497 non-null   float64
 2   citric_acid           6497 non-null   float64
 3   residual_sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free_sulfur_dioxide   6497 non-null   float64
 6   total_sulfur_dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64
 12  color                 6497 non-null   object
```

```
df.quality.value_counts()

6    2836
5    2138
7    1079
4     216
8     193
3      30
9       5
Name: quality, dtype: int64
```

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_sulfur_dioxide	density	pH	sulphates	alcohol	quality	color
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	red
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5	red
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5	red
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6	red
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5	red
...	...	...	...	...	...	...	...	...	...	...	...	...	...
492	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6	white
493	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5	white
494	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6	white
495	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7	white
496	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6	white

Data Exploration and Analysis-1

- Here we see that fixed acidity, and residual sugar, does not give any specification to classify the quality.
- Wine quality increases as the volatile acidity and chlorides decreases
- Higher quality wines contains higher citric acid and alcohol
- Other contents does not tell much how they change quality of wine
- As mentioned before, wine quality converted into bad and good quality wine from integer values and then label encoded into 0 (bad quality) and **1** (good quality)
- Also color column was removed from the dataset



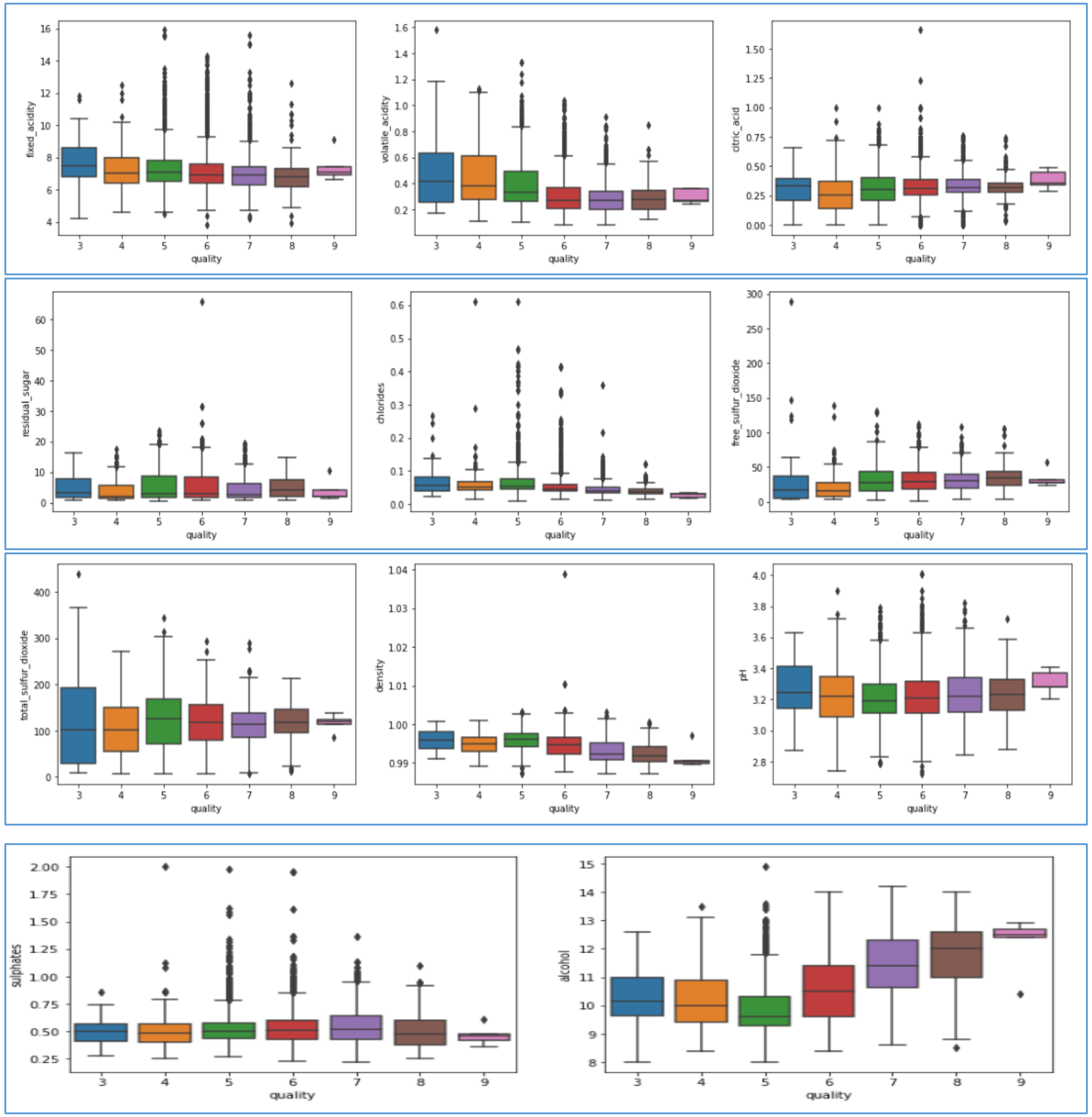
- [2-6] - bad quality wine
- (6-10] - good quality wine

```
0       bad
1       bad
2       bad
3       bad
4       bad
        ...
6492    bad
6493    bad
6494    bad
6495    good
6496    bad
```

```
0       0
1       0
2       0
3       0
4       0
        ..
6492    0
6493    0
6494    0
6495    1
6496    0
```

Data Exploration and Analysis-2

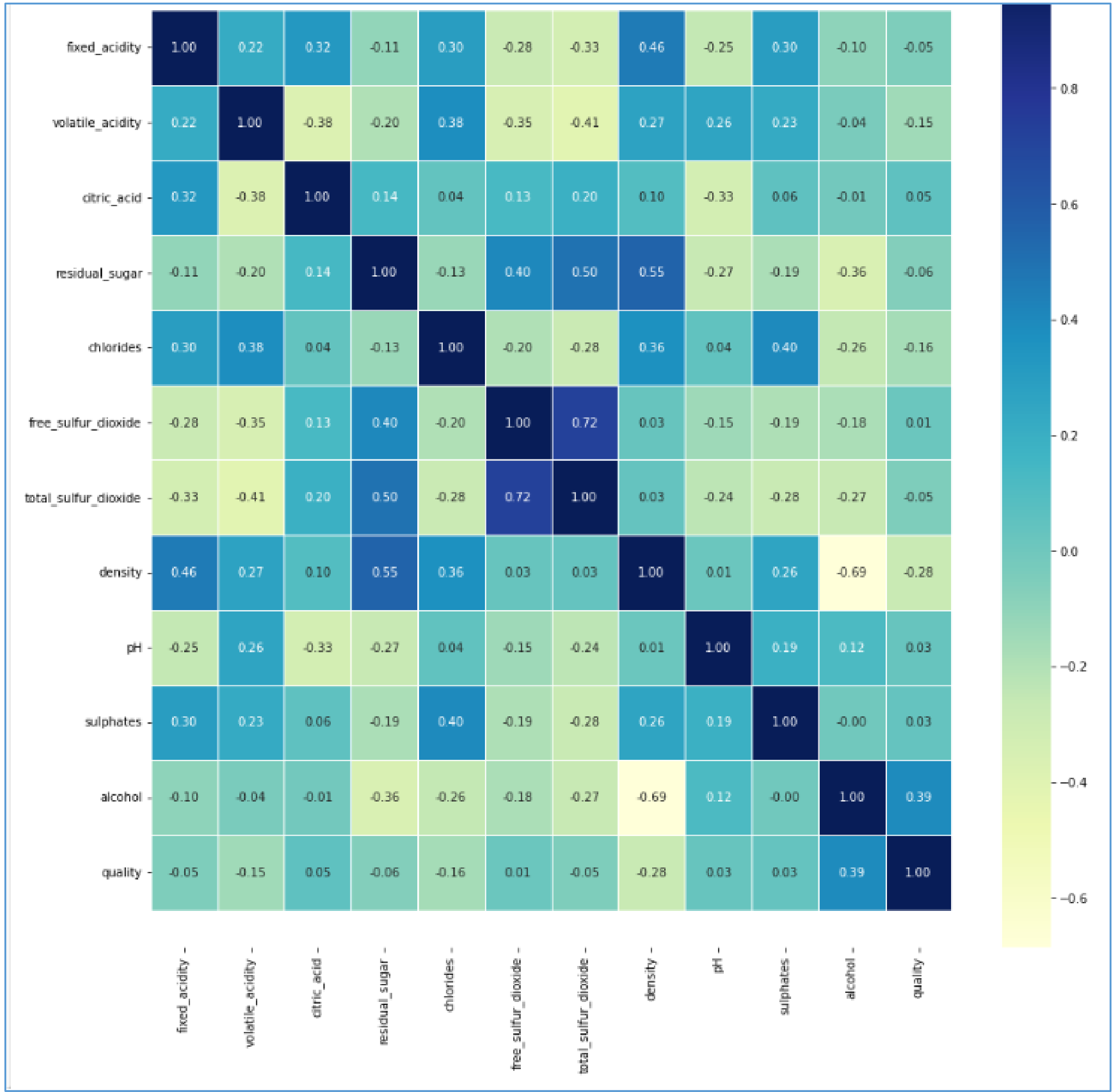
- Boxplot shows there are quite outliers for every column especially fixed_acidity, volatile-acidity, citric_acid, chlorides, sulphates



Data Exploration and Analysis-3

- Studying the correlations between features using Heat Map
- Both from correlation heatmap and Variance Inflation Factor, it is seen that there are multicollinear features.
- Density, pH, alcohol and fixed_acidity are the most multicollinear features

	variables	VIF
0	fixed_acidity	58.897405
1	volatile_acidity	8.943681
2	citric_acid	9.340251
3	residual_sugar	3.576148
4	chlorides	5.575434
5	free_sulfur_dioxide	8.452180
6	total_sulfur_dioxide	14.732237
7	density	936.984064
8	pH	589.005172
9	sulphates	18.491253
10	alcohol	107.135452



Data Exploration and Analysis-4

- There are no NULL values in the data set, which is a good thing
- Target values are imbalanced, 80% (5220) are bad quality, while 20% (1277) are good quality wine
- So, StratifiedShuffleSplit (with test_size=0.2) method was used to keep that balance while dividing the data into train and test splits
- Then, all the features are scaled with StandardScaler()

```python
data.isna().sum()
```

```
fixed_acidity           0
volatile_acidity        0
citric_acid             0
residual_sugar          0
chlorides               0
free_sulfur_dioxide     0
total_sulfur_dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

```python
# Create the data sets with scaling
scaler=StandardScaler()

# Create the data sets

X_train_s = scaler.fit_transform(data.loc[train_idx, feature_cols])
y_train = data.loc[train_idx, 'quality']

X_test_s = scaler.transform(data.loc[test_idx, feature_cols])
y_test = data.loc[test_idx, 'quality']
```

```python
data['quality'].value_counts()
```

```
0    5220
1    1277
Name: quality, dtype: int64
```

```python
data['quality'].value_counts(normalize=True)
```

```
0    0.803448
1    0.196552
Name: quality, dtype: float64
```

MACHINE LEARNING ANALYSIS AND KEY FINDINGS SECTION

Machine Learning-1

- In the Machine Learning section, I used below ML classification techniques for predicting the wine quality

1.Logistic Regression	2. KNN Algorithm	3.SVM Algorithm	4.Decision Tree Algorithm	5.Bagging Techniques	6.Boosting Algorithms
				• Bagging with SVM • Bagging with Decision Tree • Random Forest	• Gradient Boosting • AdaBoost Classifier • XGBoost Classifier

- While building these models, I have used the GridSearchCV technique for hyperparameter tuning
- As a metric to compare the results, I used ‘F1’ score in the GridSearchCV
- Also for each GridSearchCV , I divided data into 4 folds using cross-validation
- Results are given as in the form of Classification_Report and Confusion Matrix

Machine Learning-Logistic Regression

- I tried all the 3 methods here, Vanilla, Ridge and Lasso Regularizations
- All 3 methods did w
- As we can see, Vanilla, Lasso and Ridge regressions showed approximately the same results on both class 0 and class 1.
- Despite the accuracy is around 82% for 3 of them, but recall and F1-score is much lower on class 1.
- All 3 models did well on predicting class 0, while did not good enough for Class 1
- This is due to the imbalance between those classes (80% of target value are class 1-bad quality and 20% are class 2-good quality).

```
clf_report_lr_0
```

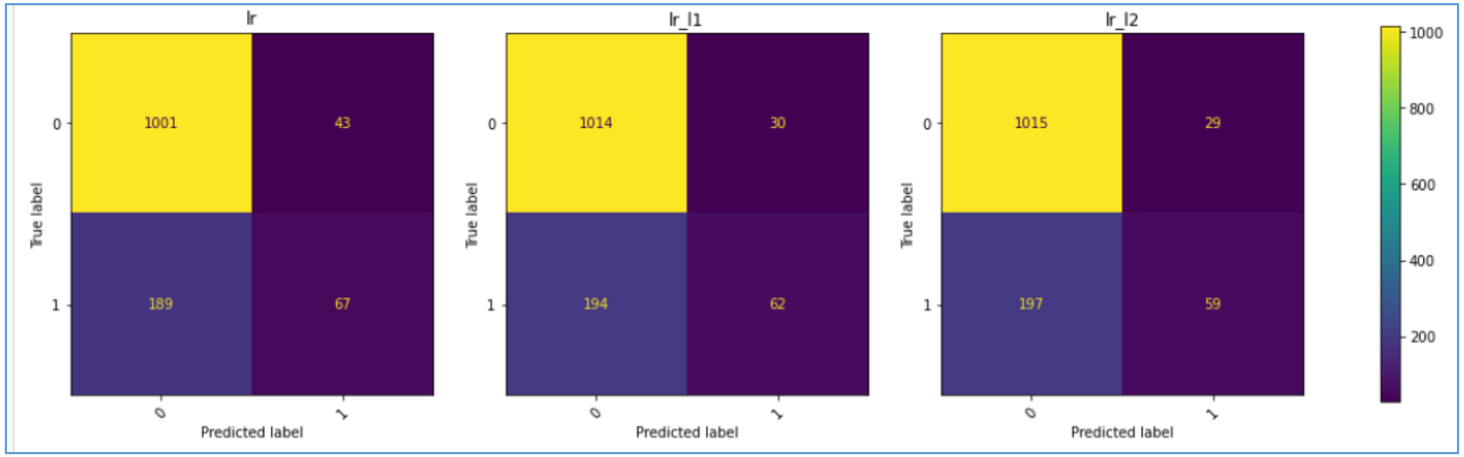
	0	1	accuracy	macro avg	weighted avg
precision	0.841176	0.609091	0.821538	0.725134	0.795473
recall	0.958812	0.261719	0.821538	0.610266	0.821538
f1-score	0.896150	0.366120	0.821538	0.631135	0.791775
support	1044.000000	256.000000	0.821538	1300.000000	1300.000000

```
clf_report_lr_1
```

	0	1	accuracy	macro avg	weighted avg
precision	0.839404	0.673913	0.827692	0.756659	0.806815
recall	0.971264	0.242188	0.827692	0.606726	0.827692
f1-score	0.900533	0.356322	0.827692	0.628427	0.793365
support	1044.000000	256.000000	0.827692	1300.000000	1300.000000

```
clf_report_lr_2
```

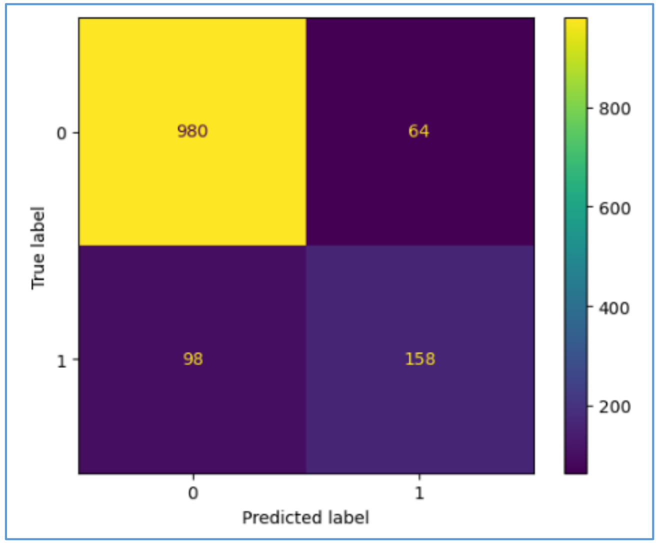
	0	1	accuracy	macro avg	weighted avg
precision	0.837459	0.670455	0.826154	0.753957	0.804572
recall	0.972222	0.230469	0.826154	0.601345	0.826154
f1-score	0.899823	0.343023	0.826154	0.621423	0.790176
support	1044.000000	256.000000	0.826154	1300.000000	1300.000000



Machine Learning-KNN Algorithm

- KNN algorithm gave the best results with
  - n_neighbors=8, metric=Euclidean and weights=distance
- Accuracy increased to 87.5%
- F1 score=66.11% for Class 1 and 92.36% for Class 0
- KNN obviously did better than Logistic Regression

	0	1	accuracy	macro avg	weighted avg
precision	0.909091	0.711712	0.875385	0.810401	0.870222
recall	0.938697	0.617188	0.875385	0.777942	0.875385
f1-score	0.923657	0.661088	0.875385	0.792372	0.871951
support	1044.000000	256.000000	0.875385	1300.000000	1300.000000




```
grid_params={

     'n_neighbors':list(range(1,100)),
     'weights':['uniform','distance'],
     'metric':['euclidean','manhattan']
}

gs=GridSearchCV(
    KNeighborsClassifier(),
    grid_params,
    verbose=2,
    cv=4,
    scoring='f1',
    n_jobs=-1
    )

gs_results=gs.fit(X_train_s,y_train)
```

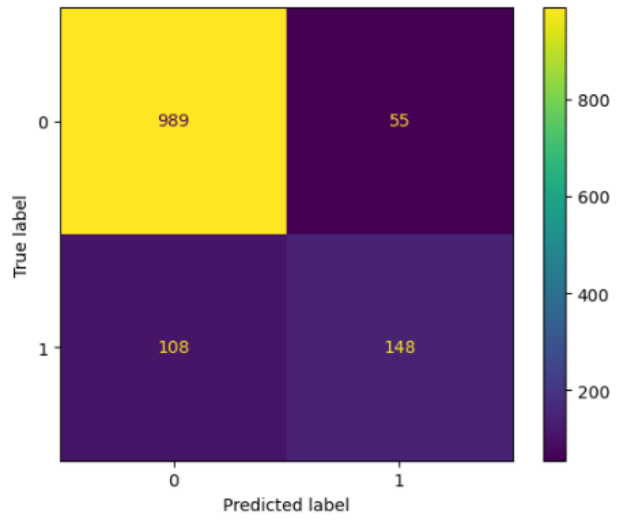
```
gs_results.best_params_
```

```
{'metric': 'euclidean', 'n_neighbors': 8, 'weights': 'distance'}
```

Machine Learning-SVM Algorithm

- KNN algorithm gave the best results with
  - C=10, gamma=1 and kernel=rbf
- Accuracy =87.46%
- F1 score=64.48% for Class 1 and 92.38% for Class 0

	0	1	accuracy	macro avg	weighted avg
precision	0.901550	0.729064	0.874615	0.815307	0.867583
recall	0.947318	0.578125	0.874615	0.762722	0.874615
f1-score	0.923867	0.644880	0.874615	0.784374	0.868928
support	1044.000000	256.000000	0.874615	1300.000000	1300.000000




```python
from sklearn.svm import SVC

params_grid = {
    'C': [0.001,0.1,1,10,100,1000],
    'gamma':[0.001,0.01,0.1,1,10],
    #'gamma':[0.001,0.01,0.1,1,10,100],
    #'kernel': ['poly', 'rbf', 'sigmoid'],
    'kernel': ['rbf', 'sigmoid']
}

# Define a GridSearchCV to search the best parameters
SVM_GS = GridSearchCV(estimator =SVC(),
                      param_grid = params_grid,
                      scoring='f1',
                      cv = 4, verbose = 2,n_jobs=-1)
```

```python
SVM_GS_results.best_params_
```

```
{'C': 10, 'gamma': 1, 'kernel': 'rbf'}
```

Machine Learning-DT Algorithm

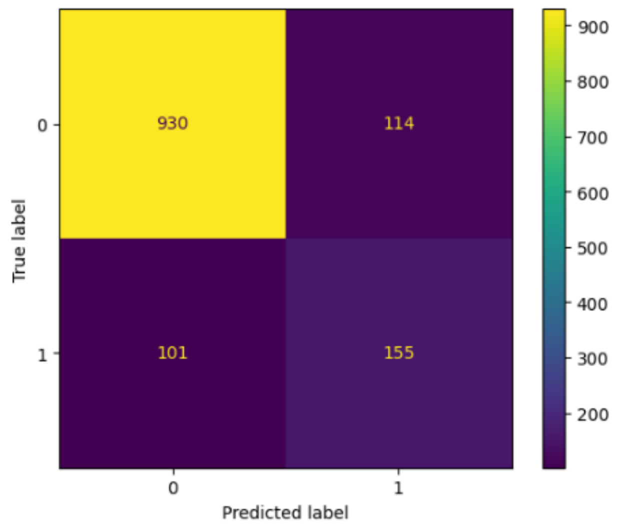
- DT algorithm gave the best results with
  - Ccriterion=entropy, max_depth=25 and min_samples_leaf=1
- Accuracy =83.46%
- F1 score=59.05% for Class 1 and 89.63% for Class 0

```
from sklearn.tree import DecisionTreeClassifier

DT = DecisionTreeClassifier(random_state=42)
params_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [5, 10, 15, 20,25,30,35,40,45,50],
    'min_samples_leaf': [1, 2, 5,10]
}

DT_GS = GridSearchCV(DT,
                     param_grid=params_grid,
                     scoring='f1',
                     cv=4,
                     verbose=2,
                     n_jobs=-1)
```

	0	1	accuracy	macro avg	weighted avg
precision	0.902037	0.576208	0.834615	0.739123	0.837874
recall	0.890805	0.605469	0.834615	0.748137	0.834615
f1-score	0.896386	0.590476	0.834615	0.743431	0.836145
support	1044.000000	256.000000	0.834615	1300.000000	1300.000000

```
DT_GS_results.best_params_
```

```
{'criterion': 'entropy', 'max_depth': 25, 'min_samples_leaf': 1}
```



Machine Learning-Bagging

- Bagging algorithm both with SVM and DT Classifiers gave the same results
- But Bagging with DT classifier predicted the Class 1 a little bit more accurately than Bagging with SVM
- F1 score=65.49%
- Accuracy=87.92%

Bagging with SVM

	0	1	accuracy	macro avg	weighted avg
precision	0.895255	0.759563	0.876154	0.827409	0.868534
recall	0.957854	0.542969	0.876154	0.750412	0.876154
f1-score	0.925497	0.633257	0.876154	0.779377	0.867949
support	1044.000000	256.000000	0.876154	1300.000000	1300.000000

Bagging with DT Classifier

	0	1	accuracy	macro avg	weighted avg
precision	0.902816	0.748744	0.879231	0.825780	0.872475
recall	0.952107	0.582031	0.879231	0.767069	0.879231
f1-score	0.926807	0.654945	0.879231	0.790876	0.873271
support	1044.000000	256.000000	0.879231	1300.000000	1300.000000

Machine Learning-Random Forest

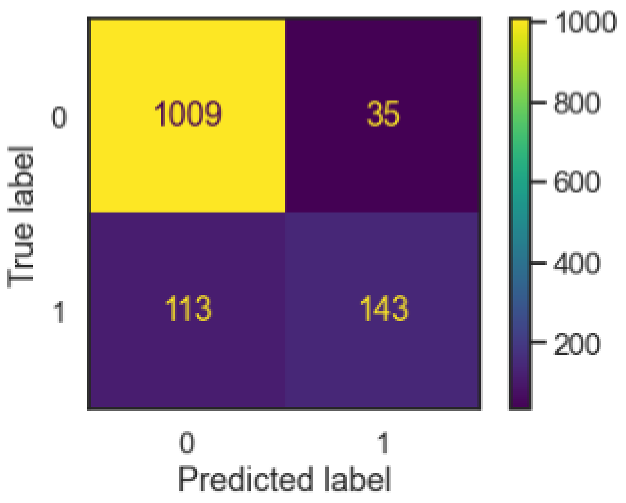
- Random Forest algorithm gave the best results with
  - max_depth=100 and n_estimators=300
- Accuracy =88.61%
- F1 score=65.89% for Class 1 and 93.16% for Class 0

```
from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(oob_score=True,warm_start=True)


param_grid = {'n_estimators': [15,20,50,100,200,300,400],
              #'n_estimators': [5,10,15,20,25,30,40,50,60,80,100,120,140,150,170,180,200],
              'max_depth' : [10,20,40,60,80,100],
              #'max_depth' : [20,25,30,35,40,45,50 ]
             #'max_features':["auto", "sqrt", "log2"]
             }

RF_GS=GridSearchCV(estimator=RF, param_grid=param_grid,scoring='f1', cv=4, verbose=2, n_jobs=-1)
RF_GS_results=RF_GS.fit(X_train_s,y_train)
```

```
RF_GS_results.best_params_
```

```
{'max_depth': 100, 'n_estimators': 300}
```

	0	1	accuracy	macro avg	weighted avg
precision	0.899287	0.803371	0.886154	0.851329	0.880399
recall	0.966475	0.558594	0.886154	0.762534	0.886154
f1-score	0.931671	0.658986	0.886154	0.795329	0.877973
support	1044.000000	256.000000	0.886154	1300.000000	1300.000000



Machine Learning-Gradient Boosting

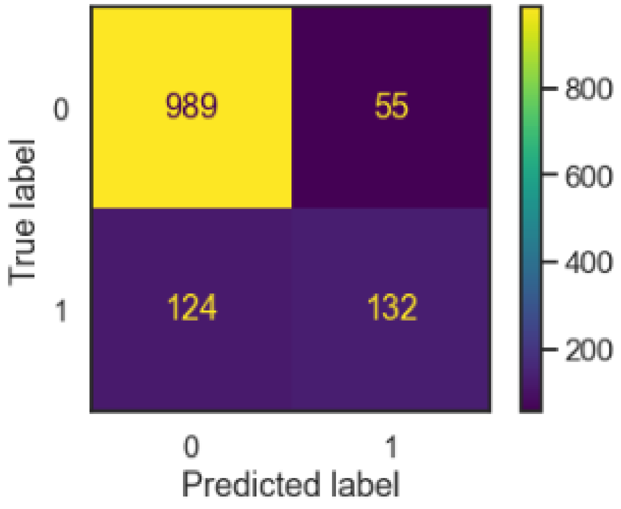
- Gradient Boosting algorithm gave the best results with

```
GBC_GS_results.best_params_
```

```
{'learning_rate': 0.1,
 'max_features': 6,
 'n_estimators': 400,
 'subsample': 1.0}
```

- Accuracy =86.23%
- F1 score=59.59% for Class 1 and 91.70 % for Class 0

```python
from sklearn.ensemble import GradientBoostingClassifier

param_grid = {'n_estimators': [15,20,50,100,200,300,400],
              'learning_rate': [0.1, 0.01, 0.001, 0.0001],
              'subsample': [1.0, 0.5],
              'max_features': [1, 2, 3, 4,6,8,10,11]}
GBC=GradientBoostingClassifier(random_state=42)
GBC_GS = GridSearchCV(estimator=GBC,
                      param_grid=param_grid,
                      scoring='f1',
                      cv=4, verbose=2, n_jobs=-1)
GBC_GS_results=GBC_GS.fit(X_train_s,y_train)
```



	0	1	accuracy	macro avg	weighted avg
precision	0.888589	0.705882	0.862308	0.797236	0.852610
recall	0.947318	0.515625	0.862308	0.731472	0.862308
f1-score	0.917014	0.595937	0.862308	0.756476	0.853787
support	1044.000000	256.000000	0.862308	1300.000000	1300.000000

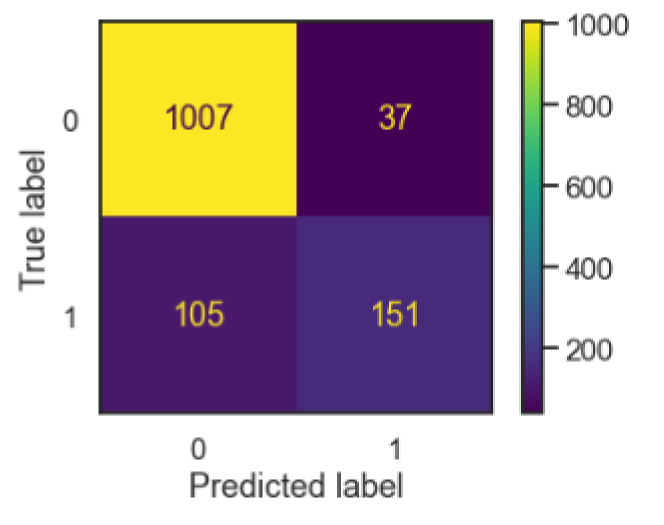
Machine Learning-AdaBoost

- AdaBoosting algorithm gave the best results with

```
ABC_GS_results.best_params_

{'base_estimator__max_depth': 10, 'learning_rate': 0.1, 'n_estimators': 400}
```

```
from sklearn.ensemble import AdaBoostClassifier

ABC = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1))

param_grid = {'base_estimator__max_depth':[5, 10, 15, 20,25,30,35,40,45,50],
              'n_estimators': [15,20,50,100,200,300,400],
              'learning_rate': [0.1, 0.01, 0.001, 0.00011]}

ABC_GS = GridSearchCV(estimator=ABC,
                      param_grid=param_grid,
                      scoring='f1',
                      cv=4, verbose=2, n_jobs=-1)
ABC_GS_results=ABC_GS.fit(X_train_s,y_train)
```

- Accuracy =89.07%
- F1 score=68.01% for Class 1 and 93.41 % for Class 0



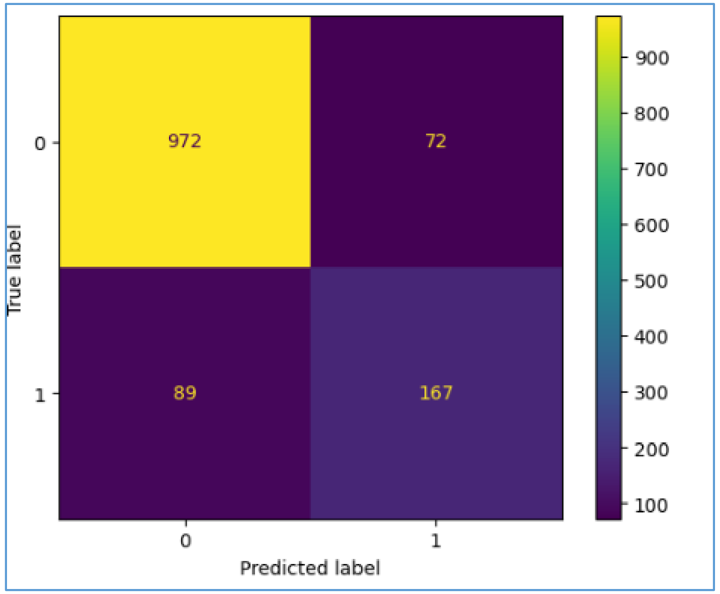
	0	1	accuracy	macro avg	weighted avg
precision	0.905576	0.803191	0.890769	0.854384	0.885414
recall	0.964559	0.589844	0.890769	0.777202	0.890769
f1-score	0.934137	0.680180	0.890769	0.807159	0.884127
support	1044.000000	256.000000	0.890769	1300.000000	1300.000000

Machine Learning-XGBoost

- XGBoost algorithm gave the best results with

```
XGBOOST_GS_results.best_params_

{'colsample_bytree': 0.9,
 'learning_rate': 0.01,
 'max_depth': 20,
 'n_estimators': 300,
 'scale_pos_weight': 4,
 'subsample': 0.8}
```

- Accuracy =87.61%
- F1 score=67.47% for Class 1 and 92.35 % for Class 0




```
import xgboost as xgb
params = { 'max_depth': [20,60,100],
           'learning_rate': [0.001, 0.01, 0.1],
           'n_estimators': [20,100,300,400],
           'scale_pos_weight':[3,4,6],
           #'gamma': [0, 0.5, 1, 10],
           'subsample': [0.1, 0.4, 0.8],
           'colsample_bytree': [0.3, 0.5,0.9]}

XGB=xgb.XGBClassifier(booster='gbtree',objective="binary:logistic", random_state=42)

XGBOOST_GS = GridSearchCV(estimator=XGB,
                          param_grid=params,
                          scoring='f1',
                          cv=4, verbose=2, n_jobs=-1)

XGBOOST_GS_results=XGBOOST_GS.fit(X_train_s,y_train)
```

	0	1	accuracy	macro avg	weighted avg
precision	0.916117	0.698745	0.876154	0.807431	0.873311
recall	0.931034	0.652344	0.876154	0.791689	0.876154
f1-score	0.923515	0.674747	0.876154	0.799131	0.874527
support	1044.000000	256.000000	0.876154	1300.000000	1300.000000

Findings,Results and Recommendations

Key Findings and Results

- Wine quality data set is imbalanced 80% Class 0 and 20% Class 1
- There are quite outliers for every column especially fixed_acidity, volatile-acidity, citric_acid, chlorides, sulphates
- Density, pH, alcohol, and fixed_acidity are the most multicollinear features
- All the models did well on predicting class 0 , while did not good enough for Class 1
- This is due to the imbalance between those classes
- Among those models, AdaBoost and XGBoost did fairly well. They both predicted the Class 0 with Accuracy=87-89%, F1=92-93%, Precision=90-91% and Recall=93-96%
- But Class 1 prediction of these 2 is not that high

  Accuracy=87-89%, F1=67-68%, Precision=70-80% and Recall=59-65%

Recommendations:

- In order to increase the predictability of Class 1, the data can be balanced before modelling
- Different metrics like AUC score can be used
- Different weight parameters in XGBoost can be checked
- Oversampling or Undersampling methods can be used

	Accuracy	Precision(Class 1)	Recall (Class 1)	F1 score (Class 1)
Logistic Regression	82.15	60.9	26.17	36.6
KNN	87.5	71.17	61.71	66.11
SVM	87.46	72.9	57.81	64.48
DT	83.46	57.62	60.54	59.04
Random Forest	88.61	80.33	55.85	65.89
Gradient Boosting	86.23	70.58	51.56	59.59
AdaBoost	89.07	80.31	58.98	68.01
XGBoost	87.61	69.87	65.23	67.47

	Accuracy	Precision(Class 0)	Recall (Class 0)	F1 score (Class 0)
Logistic Regression	82.15	83.74	97.22	89.98
KNN	87.5	90.90	93.86	92.36
SVM	87.46	90.15	94.73	92.83
DT	83.46	90.20	89.08	89.63
Random Forest	88.61	89.92	96.64	93.16
Gradient Boosting	86.23	88.58	94.73	91.7
AdaBoost	89.07	90.55	96.45	93.41
XGBoost	87.61	91.61	93.10	92.35