Wine Analyzer

Group 4
Dinnara Hitt, Jung_Hsuan Lin (Coco), Jocelyn DeLeon, Fannie Chang, Caroline Ambriz
https://josiedeleon.github.io/Final-Project/
https://github.com/josiedeleon/Final-Project

Data Source

UCI Archive

https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/

Two datasets related to red and white **Vinho Verde** wine samples, from the north of Portugal.

- ❏ winequality-red.csv
- ❏ winequality-white.csv

P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.
Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

Data Cleaning

```
df = df.drop_duplicates(keep = 'first', inplace = False)
df = df.dropna(axis='columns', how='all')
df = df.reset_index(drop=True)
df
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	6.2	0.32	0.16	7.0	0.045	30.0	136.0	0.99490	3.18	0.47	9.6	6
...	...	...	...	...	...	...	...	...	...	...	...	...
3952	6.2	0.21	0.29	1.6	0.039	24.0	92.0	0.99114	3.27	0.50	11.2	6
3953	6.6	0.32	0.36	8.0	0.047	57.0	168.0	0.99490	3.15	0.46	9.6	5
3954	6.5	0.24	0.19	1.2	0.041	30.0	111.0	0.99254	2.99	0.46	9.4	6
3955	5.5	0.29	0.30	1.1	0.022	20.0	110.0	0.98869	3.34	0.38	12.8	7
3956	6.0	0.21	0.38	0.8	0.020	22.0	98.0	0.98941	3.26	0.32	11.8	6

```
#import CSV file
df = pd.read_csv('Resources/Data/winequality-white.csv', delimiter=';')
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

```
#import CSV file
df = pd.read_csv('Resources/Data/winequality-red.csv', delimiter=';')
df.head()
```

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

```
# categorize wine quality
bins = (2,6,9)
group_names = ['Fair','Very Good']
#group_names = [0,1]
categories = pd.cut(df['quality'], bins, labels = group_names)
#categories
```

```
df['quality'] =categories
#df
```

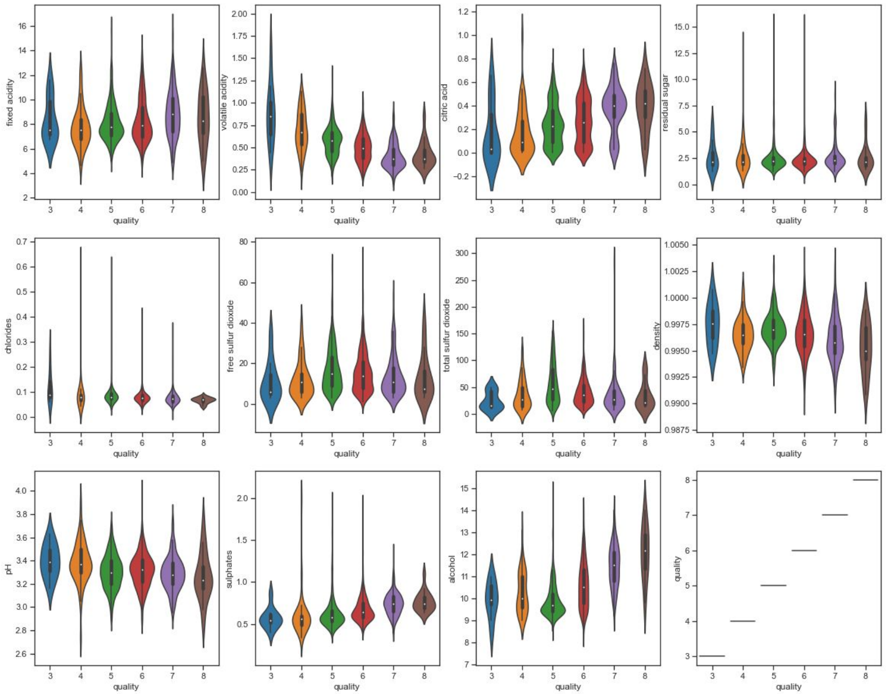
```
df.quality.value_counts()
```

```
Fair         3134
Very Good     823
Name: quality, dtype: int64
```

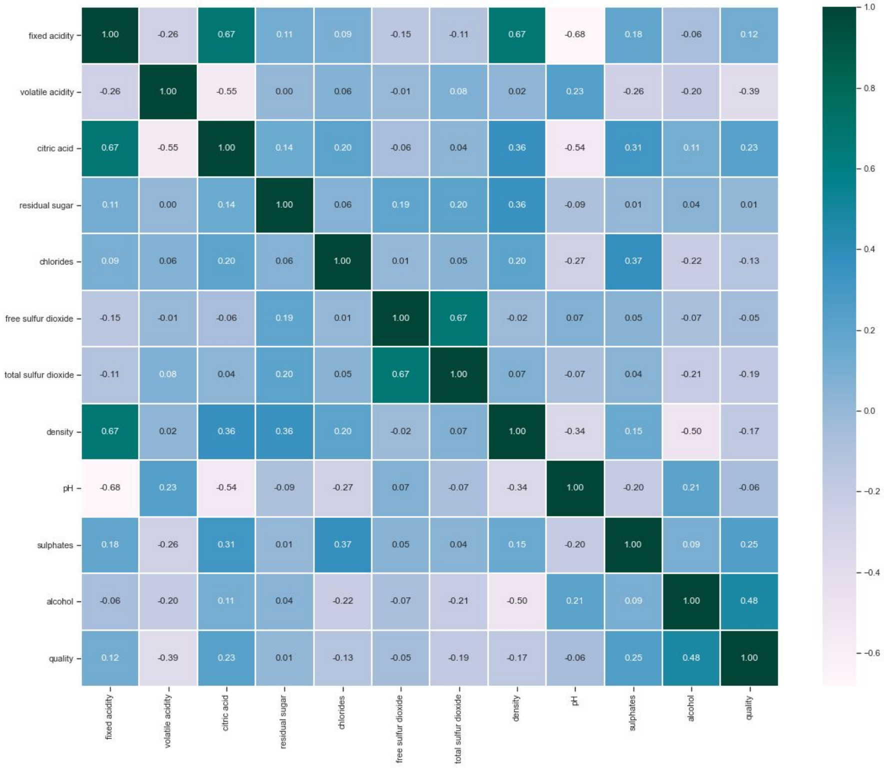
```
df.to_csv("./Resources/Data/winequality-white-clean.csv",index=False)
```

Red Wine Features

Red Wine

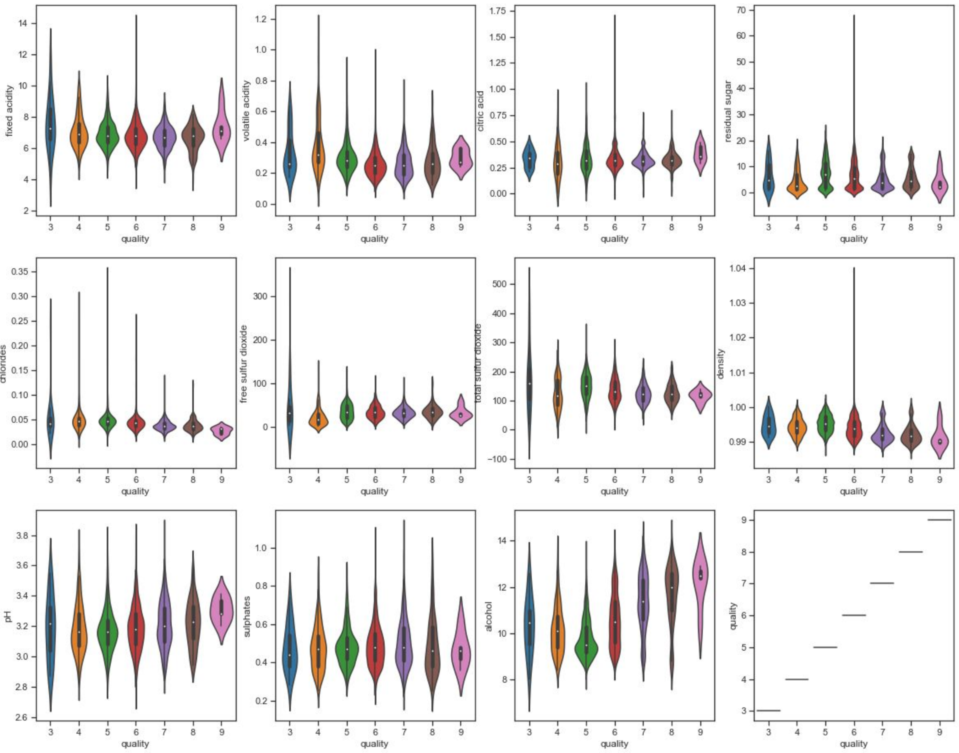


Correlation Heat Map

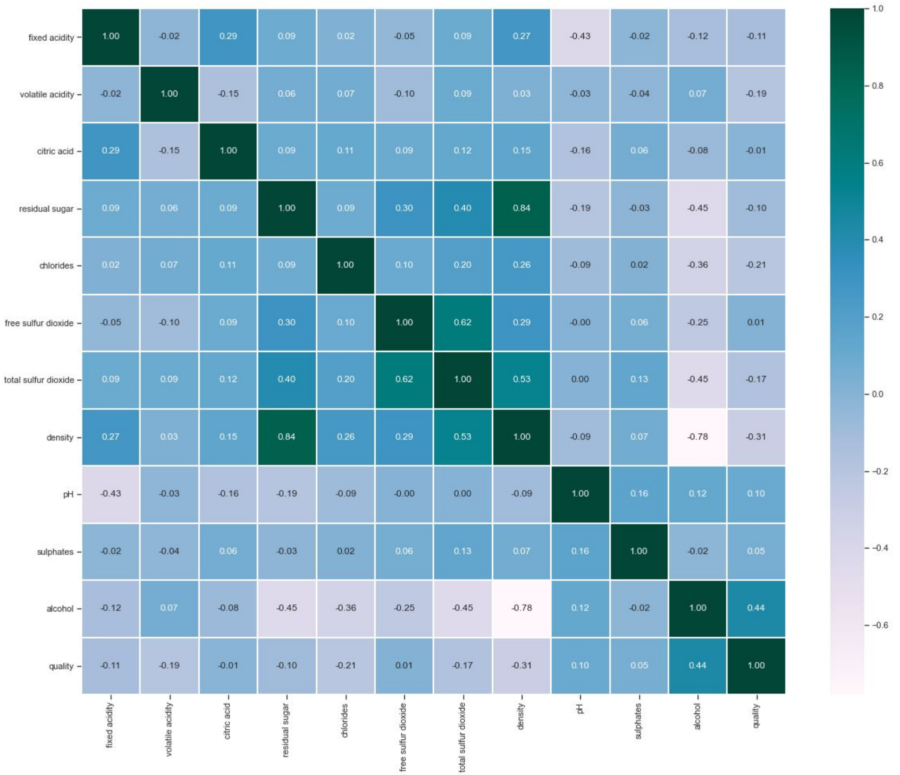


White Wine Features

White Wine

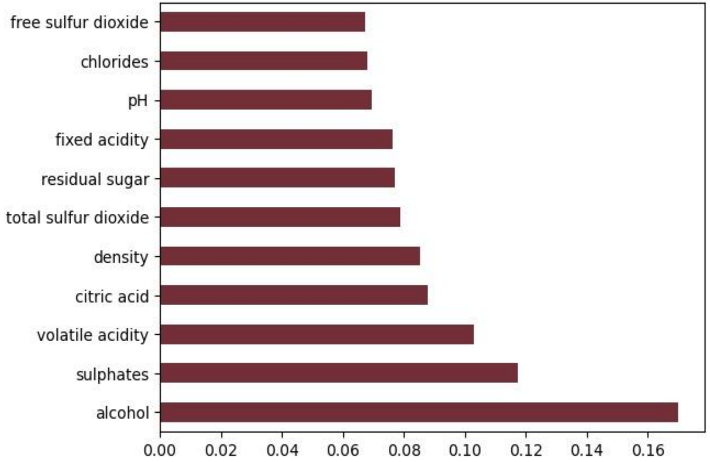


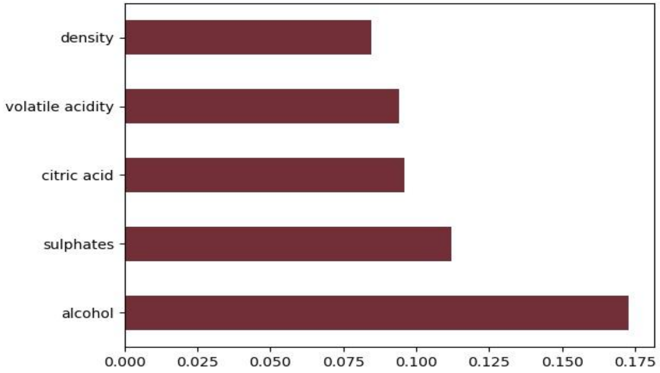
Correlation Heat Map



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1.00	-0.02	0.29	0.09	0.02	-0.05	0.09	0.27	-0.43	-0.02	-0.12	-0.11
volatile acidity	-0.02	1.00	-0.15	0.06	0.07	-0.10	0.09	0.03	-0.03	-0.04	0.07	-0.19
citric acid	0.29	-0.15	1.00	0.09	0.11	0.09	0.12	0.15	-0.16	0.06	-0.08	-0.01
residual sugar	0.09	0.06	0.09	1.00	0.09	0.30	0.40	0.84	-0.19	-0.03	-0.45	-0.10
chlorides	0.02	0.07	0.11	0.09	1.00	0.10	0.20	0.26	-0.09	0.02	-0.36	-0.21
free sulfur dioxide	-0.05	-0.10	0.09	0.30	0.10	1.00	0.62	0.29	-0.00	0.06	-0.25	0.01
total sulfur dioxide	0.09	0.09	0.12	0.40	0.20	0.62	1.00	0.53	0.00	0.13	-0.45	-0.17
density	0.27	0.03	0.15	0.84	0.26	0.29	0.53	1.00	-0.09	0.07	-0.78	-0.31
pH	-0.43	-0.03	-0.16	-0.19	-0.09	-0.00	0.00	-0.09	1.00	0.16	0.12	0.10
sulphates	-0.02	-0.04	0.06	-0.03	0.02	0.06	0.13	0.07	0.16	1.00	-0.02	0.05
alcohol	-0.12	0.07	-0.08	-0.45	-0.36	-0.25	-0.45	-0.78	0.12	-0.02	1.00	0.44
quality	-0.11	-0.19	-0.01	-0.10	-0.21	0.01	-0.17	-0.31	0.10	0.05	0.44	1.00

Machine Learning





Target Value:

Quality

(Fair and Very Good)

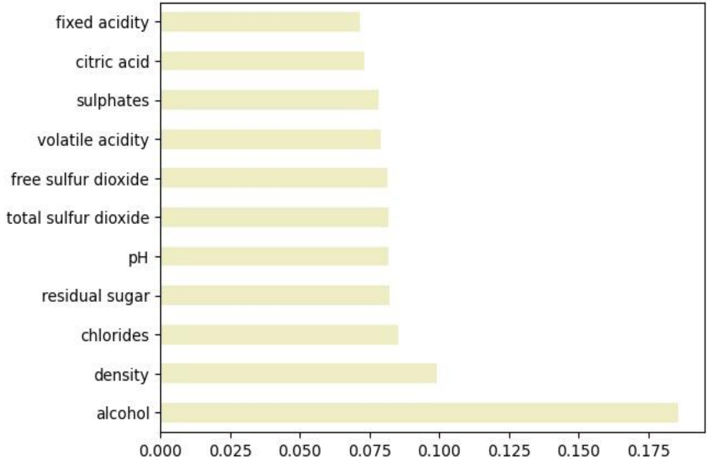
Methods:

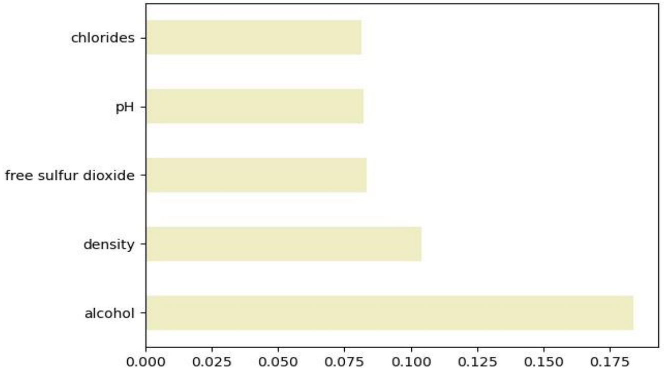
- Support Vector Machines
- Deep Learning
- K Nearest Neighbor
- Logistic Regression
- Random Forests
- XG Boost

Features :

All Features vs Top 5 Features

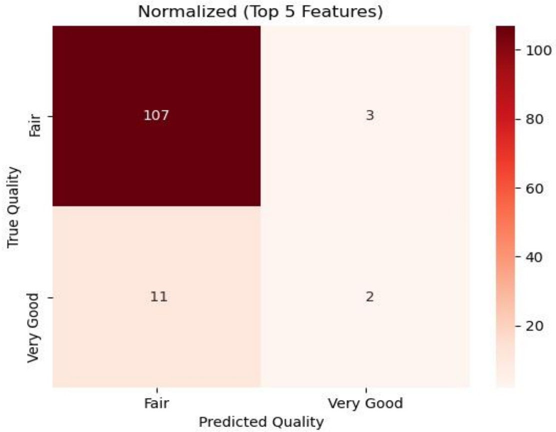
(Red Wine vs White Wine)





Support Vector Machines (SVM)

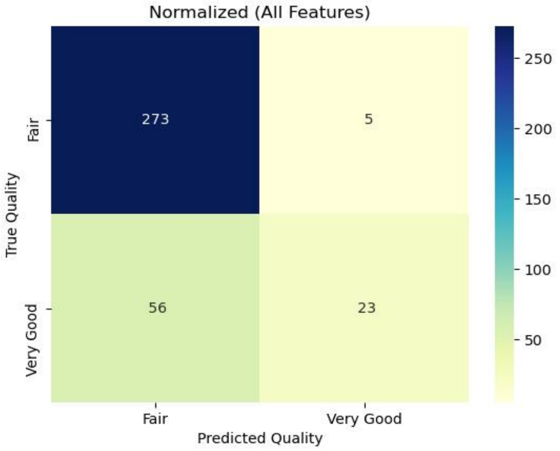
Red Wine



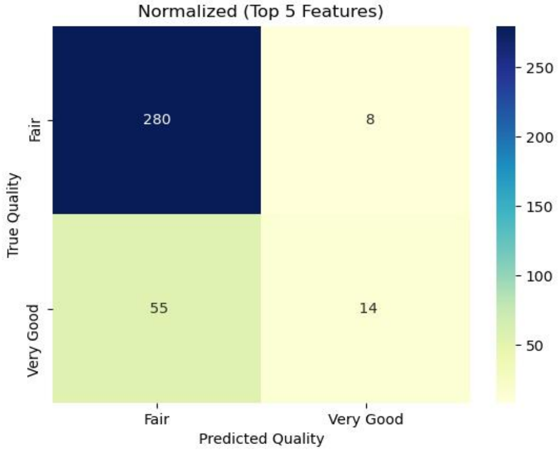
	precision	recall	f1-score	support
Fair	0.86	0.98	0.92	103
Very Good	0.67	0.20	0.31	20
accuracy			0.85	123
macro avg	0.76	0.59	0.61	123
weighted avg	0.83	0.85	0.82	123

	precision	recall	f1-score	support
Fair	0.91	0.97	0.94	110
Very Good	0.40	0.15	0.22	13
accuracy			0.89	123
macro avg	0.65	0.56	0.58	123
weighted avg	0.85	0.89	0.86	123

Support Vector Machines (SVM)



White Wine



	precision	recall	f1-score	support
Fair	0.83	0.97	0.89	278
Very Good	0.74	0.29	0.42	79
accuracy			0.82	357
macro avg	0.79	0.63	0.66	357
weighted avg	0.81	0.82	0.79	357

	precision	recall	f1-score	support
Fair	0.83	0.96	0.89	282
Very Good	0.66	0.25	0.37	75
accuracy			0.82	357
macro avg	0.74	0.61	0.63	357
weighted avg	0.79	0.82	0.78	357

Support Vector Machines (SVM) - Jupyter Notebook

Create a Train Test Split

All Features

```
# Split the data using train_test_split
# create the train and validation datasets

from sklearn.model_selection import train_test_split

X_train, X_left, y_train, y_left = train_test_split(X, y, train_size=.7)
X_val, X_test, y_val, y_test = train_test_split(X_left, y_left, train_size=.7)
```

```
X_train.shape, X_val.shape, X_test.shape
```

```
((2772, 11), (832, 11), (357, 11))
```

Pre-pocessing

```
# Scale your data
X_scale = StandardScaler().fit(X_train)
```

```
X_train_scaled = X_scale.transform(X_train)
X_test_scaled = X_scale.transform(X_test)
```

Train the Model

```
from sklearn.svm import SVC

model_svm = SVC()
model_svm.fit(X_train_scaled, y_train)
```

```
SVC()
```

```
#And score the model using the unseen testing data
model_svm.score(X_train, y_train), model_svm.score(X_val, y_val)
```

```
(0.7911255411255411, 0.7956730769230769)
```

```
# Overall Score for the model
model_svm.score(X_val, y_val)
```

```
0.7956730769230769
```

```
print(f"Training Data Score: {model_svm.score(X_train_scaled, y_train)}")
print(f"Testing Data Score: {model_svm.score(X_test_scaled, y_test)}")
```

```
Training Data Score: 0.8376623376623377
Testing Data Score: 0.834733893557423
```

```
predictions = model_svm.predict(X_test_scaled)
```

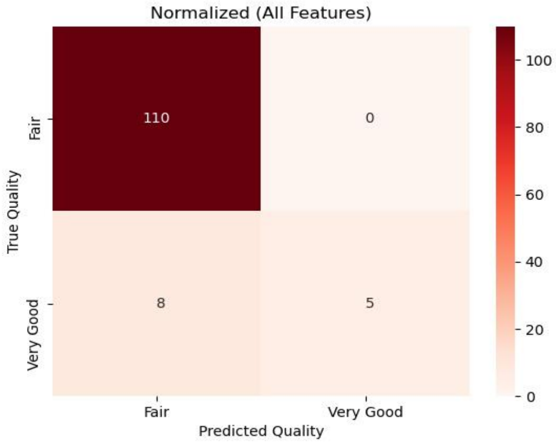
```
# Create the GridSearchCV model
from sklearn.model_selection import GridSearchCV
param_grid = {'C': [ 0.001,0.01,1,10],
              'gamma': [0.001,0.01,1,10],
             'kernel': ['linear', 'rbf', 'sigmoid']}

grid = GridSearchCV(model_svm, param_grid, verbose=3, return_train_score=True, scoring='accuracy' , cv=10)
```

```
# Train the model with GridSearch
grid.fit(X_train_scaled, y_train)
```

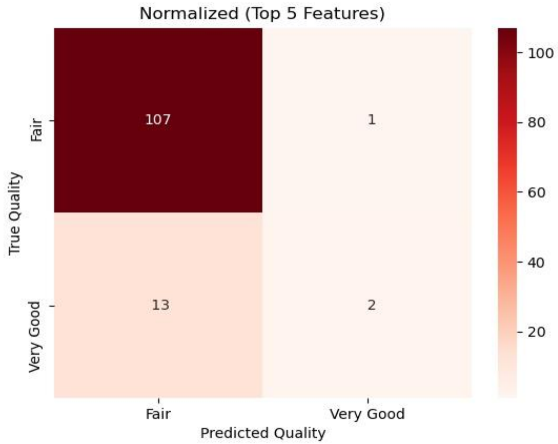
```
print(grid.best_params_)
print(grid.best_score_)
```

```
{'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
0.8188985273874764
```

Deep Learning



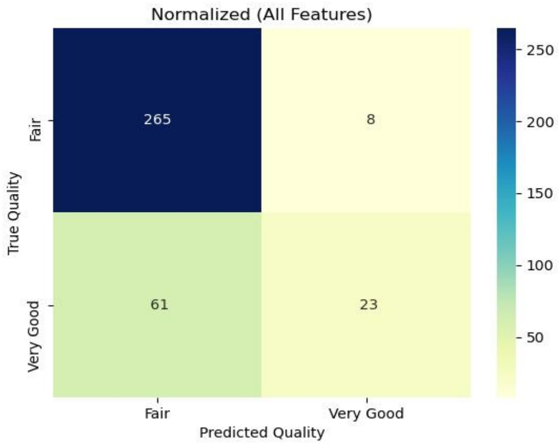
Red Wine



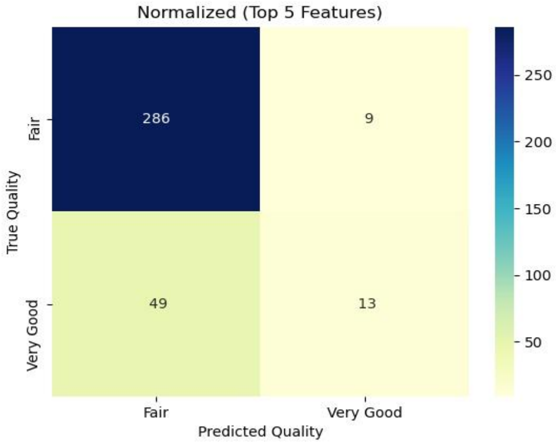
	precision	recall	f1-score	support
Fair	0.93	1.00	0.96	110
Very Good	1.00	0.38	0.56	13
accuracy			0.93	123
macro avg	0.97	0.69	0.76	123
weighted avg	0.94	0.93	0.92	123

	precision	recall	f1-score	support
Fair	0.89	0.99	0.94	108
Very Good	0.67	0.13	0.22	15
accuracy			0.89	123
macro avg	0.78	0.56	0.58	123
weighted avg	0.86	0.89	0.85	123

Deep Learning



White Wine



	precision	recall	f1-score	support
Fair	0.81	0.97	0.88	273
Very Good	0.74	0.27	0.40	84
accuracy			0.81	357
macro avg	0.78	0.62	0.64	357
weighted avg	0.80	0.81	0.77	357

	precision	recall	f1-score	support
Fair	0.85	0.97	0.91	295
Very Good	0.59	0.21	0.31	62
accuracy			0.84	357
macro avg	0.72	0.59	0.61	357
weighted avg	0.81	0.84	0.80	357

Deep Learning- Jupyter Notebook

```
# Selected important features - top 5
selected_features = X[['alcohol', 'density','free sulfur dioxide','residual sugar','pH']]
```

```
from sklearn.model_selection import train_test_split
X_train, X_left, y_train, y_left = train_test_split(selected_features,y, train_size=.7)
X_val, X_test, y_val, y_test = train_test_split(X_left, y_left, train_size=.7)
```

```
X_train.shape, X_val.shape, X_test.shape, y_train.shape, y_val.shape, y_test.shape
```

```
((2772, 5), (832, 5), (357, 5), (2772,), (832,), (357,))
```

Pre-pocessing ¶

```
# Scale the data using the StandardScaler and perform some feature selection
X_scale = StandardScaler().fit(X_train)
```

```
X_train_scaled = X_scale.transform(X_train)
X_test_scaled = X_scale.transform(X_test)
```

```
print(X_train_scaled.shape, X_test_scaled.shape, y_train.shape)
```

```
(2772, 5) (357, 5) (2772,)
```

```
# Step 1: Label-encode data set
label_encoder = LabelEncoder()
label_encoder.fit(y_train)
encoded_y_train = label_encoder.transform(y_train)
encoded_y_test = label_encoder.transform(y_test)
```

```
# Step 2: Convert encoded labels to one-hot-encoding
y_train_categorical = to_categorical(encoded_y_train)
y_test_categorical = to_categorical(encoded_y_test)
```

Quantify our Trained Model

```
#evaluate the model
deep_model_loss, deep_model_accuracy = deep_model.evaluate(
    X_test_scaled, y_test_categorical, verbose=2)
print(
    f"Normal Neural Network - Loss: {deep_model_loss}, Accuracy: {deep_model_accuracy}")
```

```
12/12 - 0s - loss: 0.4010 - accuracy: 0.8263
Normal Neural Network - Loss: 0.40100419521331787, Accuracy: 0.8263305425643921
```

Train the Model

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
# Create model and add layers
deep_model = Sequential()
deep_model.add(Dense(units=20, activation='relu', input_dim=5))
deep_model.add(Dense(units=20, activation='relu'))
deep_model.add(Dense(units=2, activation='softmax'))
```

```
# Compile and fit the model
deep_model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
deep_model.summary()
```

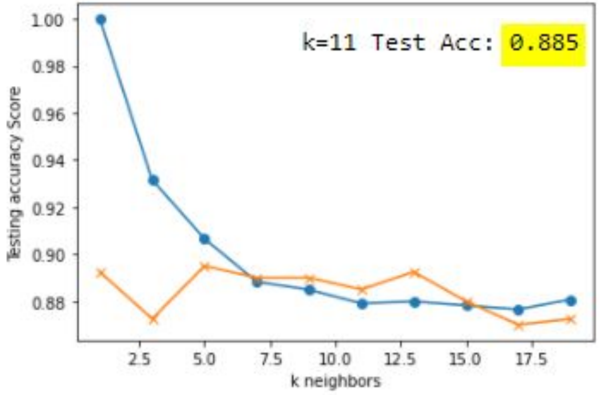
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 20)                120
_________________________________________________________________
dense_1 (Dense)              (None, 20)                420
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 42
=================================================================
Total params: 582
Trainable params: 582
Non-trainable params: 0
_________________________________________________________________
```

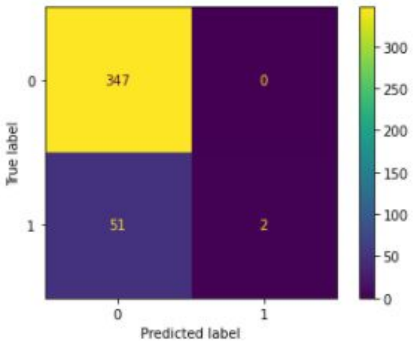
```
# fit model

history = deep_model.fit(
    X_train_scaled,
    y_train_categorical,
    validation_split=.2 ,
    epochs=30,
    shuffle=True,
    verbose=2,
    validation_data=(X_test_scaled, y_test_categorical)
)
```

```
Epoch 1/30
70/70 - 0s - loss: 0.6170 - accuracy: 0.6640 - val_loss: 0.4828 - val_accuracy: 0.8000
Epoch 2/30
70/70 - 0s - loss: 0.4633 - accuracy: 0.7988 - val_loss: 0.4359 - val_accuracy: 0.8018
Epoch 3/30
70/70 - 0s - loss: 0.4338 - accuracy: 0.8074 - val_loss: 0.4259 - val_accuracy: 0.8018
Epoch 4/30
70/70 - 0s - loss: 0.4245 - accuracy: 0.8065 - val_loss: 0.4239 - val_accuracy: 0.7982
Epoch 5/30
70/70 - 0s - loss: 0.4198 - accuracy: 0.8051 - val_loss: 0.4209 - val_accuracy: 0.7982
Epoch 6/30
70/70 - 0s - loss: 0.4156 - accuracy: 0.8088 - val_loss: 0.4215 - val_accuracy: 0.7982
```

KNN- K Nearest Neighbor

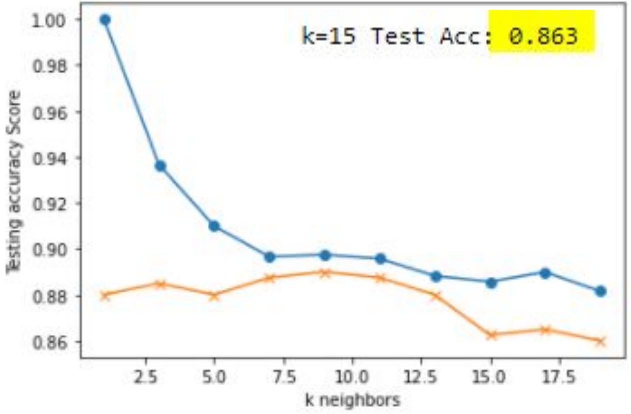
All Features

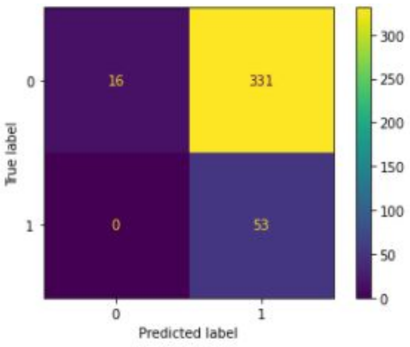




Red Wine

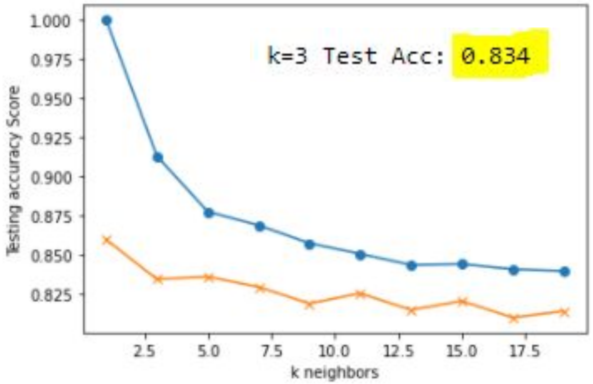
Top 5 Features

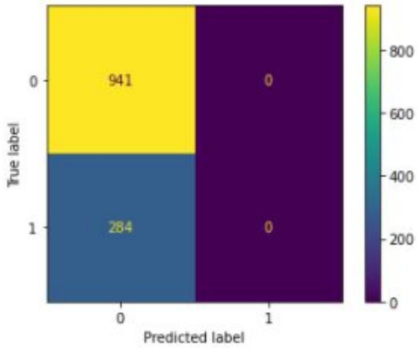




KNN- K Nearest Neighbor

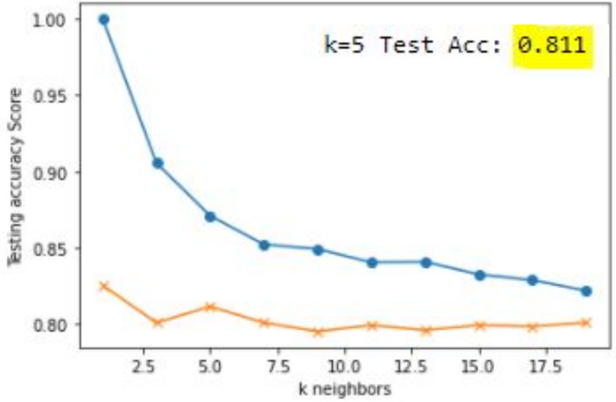
All Features

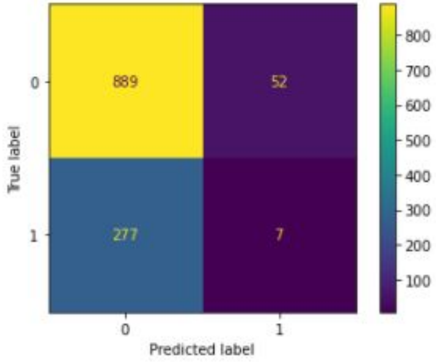




White Wine

Top 5 Features





KNN- K Nearest Neighbor

```
#All features
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
X_value, X_test, y_value, y_test= train_test_split(X, y, random_state=42)
```

```
X_train.shape, X_value.shape, X_test.shape
```

```
((3673, 11), (3673, 11), (1225, 11))
```

```
# Create a StandardScater model and fit it to the training data

X_scaler = StandardScaler().fit(X_train)
```

```
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

```
print(X_train_scaled.shape, X_test_scaled.shape, y_train.shape)
```

```
(3673, 11) (1225, 11) (3673,)
```

```
# Loop through different k values to see which has the highest accuracy
# Note: We only use odd numbers because we don't want any ties
train_scores = []
test_scores = []
for k in range(1, 20, 2):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train_scaled, y_train)
    train_score = knn.score(X_train_scaled, y_train)
    test_score = knn.score(X_test_scaled, y_test)
    train_scores.append(train_score)
    test_scores.append(test_score)
    print(f"k: {k}, Train/Test Score: {train_score:.3f}/{test_score:.3f}")
```

```
k: 1, Train/Test Score: 1.000/0.860
k: 3, Train/Test Score: 0.913/0.834
k: 5, Train/Test Score: 0.877/0.836
k: 7, Train/Test Score: 0.869/0.829
k: 9, Train/Test Score: 0.857/0.819
k: 11, Train/Test Score: 0.851/0.825
k: 13, Train/Test Score: 0.843/0.815
k: 15, Train/Test Score: 0.844/0.820
k: 17, Train/Test Score: 0.841/0.810
k: 19, Train/Test Score: 0.840/0.814
```

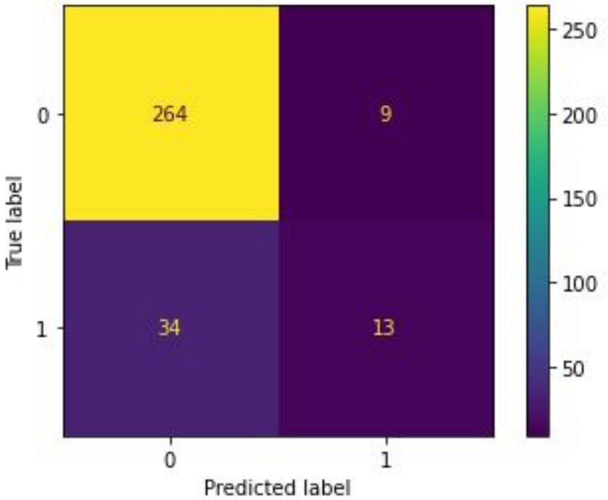
```
#plt.plot(range(1, 20, 2), train_scores, marker='o')
#plt.plot(range(1, 20, 2), test_scores, marker="x")
#plt.xlabel("k neighbors")
#plt.ylabel("Testing accuracy Score")
#plt.savefig('Resources/images/white_all_features_KNN.jpg')
#plt.show()
```

```
 # Note that k: 15 seems to be the best choice for this dataset
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train_scaled, y_train)
print('k=3 Test Acc: %.3f' % knn.score(X_test_scaled, y_test))
```

Logistic Regression

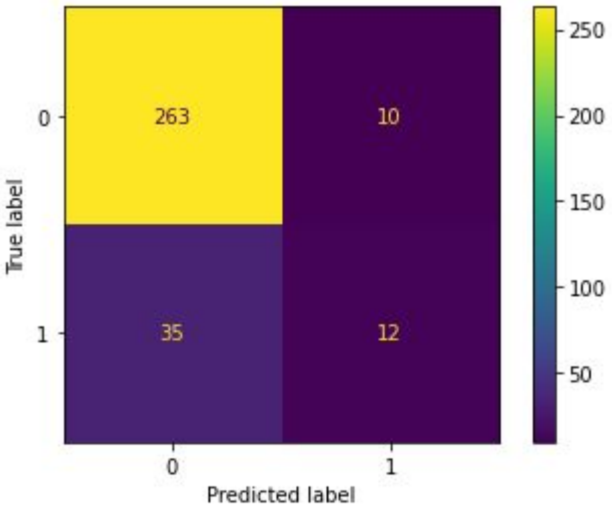
Red Wine

All Features



	precision	recall	f1-score	support
Fair	0.97	0.89	0.92	298
Very Good	0.28	0.59	0.38	22
accuracy			0.87	320
macro avg	0.62	0.74	0.65	320
weighted avg	0.92	0.87	0.89	320

Top 5 Features

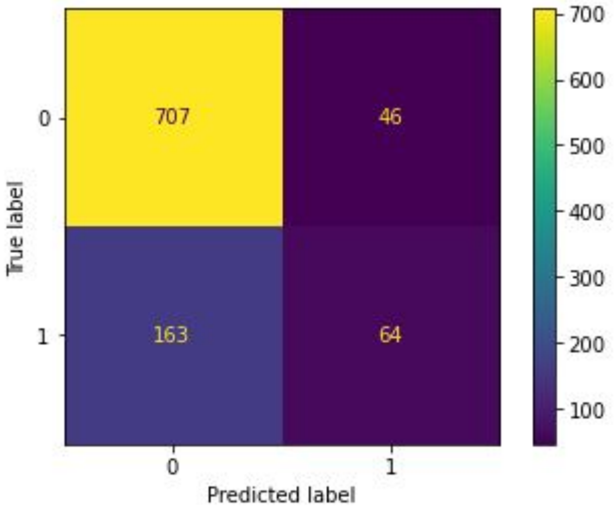


	precision	recall	f1-score	support
Fair	0.96	0.88	0.92	298
Very Good	0.26	0.55	0.35	22
accuracy			0.86	320
macro avg	0.61	0.71	0.63	320
weighted avg	0.91	0.86	0.88	320

Logistic Regression

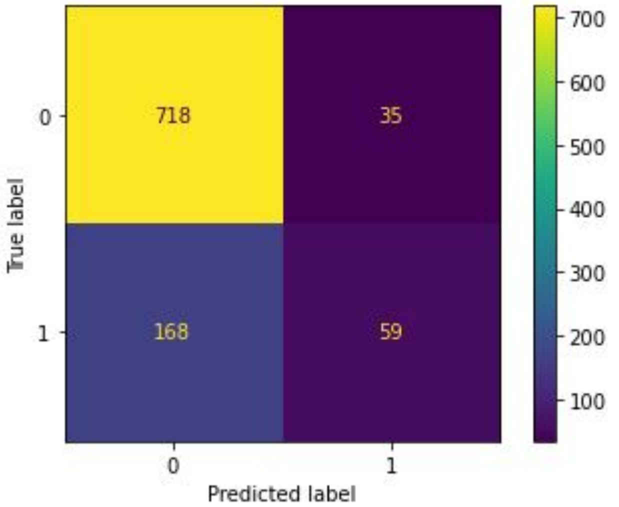
White Wine

All Features



	precision	recall	f1-score	support
Fair	0.94	0.81	0.87	870
Very Good	0.28	0.58	0.38	110
accuracy			0.79	980
macro avg	0.61	0.70	0.63	980
weighted avg	0.87	0.79	0.82	980

Top 5 Features



	precision	recall	f1-score	support
Fair	0.95	0.81	0.88	886
Very Good	0.26	0.63	0.37	94
accuracy			0.79	980
macro avg	0.61	0.72	0.62	980
weighted avg	0.89	0.79	0.83	980

Logistic Regression

```
x = df[df.columns[:-1]]
y = df['quality']
sc = StandardScaler()
x = sc.fit_transform(x)

# Split our data into training and testing
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=.2, random_state=42)
```

```
for data in [y_train, y_test]:
    print(data.describe())
```

```
count      1279
unique        2
top        Fair
freq       1109
Name: quality, dtype: object
count       320
unique        2
top        Fair
freq        273
Name: quality, dtype: object
```

```
#Create a Logistic Regression Model
classifier = LogisticRegression()
classifier
```

```
LogisticRegression()
```

```
 classifier.fit(x_train, y_train)
```

```
LogisticRegression()
```

```
print(f"Training Data Score: {classifier.score(x_train, y_train)}")
print(f"Testing Data Score: {classifier.score(x_test, y_test)}")
```

```
Training Data Score: 0.8858483189992181
Testing Data Score: 0.865625
```

```
predictions = classifier.predict(x_test)
print(f"First 10 Predictions:   {predictions[:10]}")
print(f"First 10 Actual labels: {y_test[:10].tolist()}")
```

```
First 10 Predictions:   ['Fair' 'Fair' 'Fair' 'Fair' 'Fair' 'Fair' 'Fair' 'Fair' 'Fair' 'Fair']
First 10 Actual labels: ['Fair', 'Fair', 'Fair', 'Fair', 'Fair', 'Fair', 'Fair', 'Fair', 'Fair', 'Fair']
```

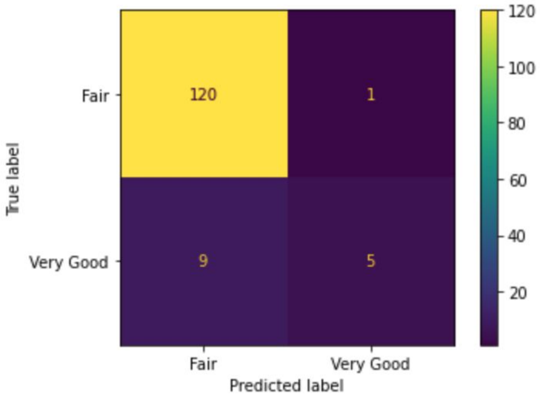
```
pd.DataFrame({"Prediction": predictions, "Actual": y_test}).reset_index(drop=True)
```

	Prediction	Actual
0	Fair	Fair
1	Fair	Fair
2	Fair	Fair
3	Fair	Fair
4	Fair	Fair
...	...	...
315	Fair	Fair
316	Fair	Fair
317	Fair	Fair
318	Fair	Fair
319	Fair	Fair

320 rows × 2 columns

Random Forests - Red Wine

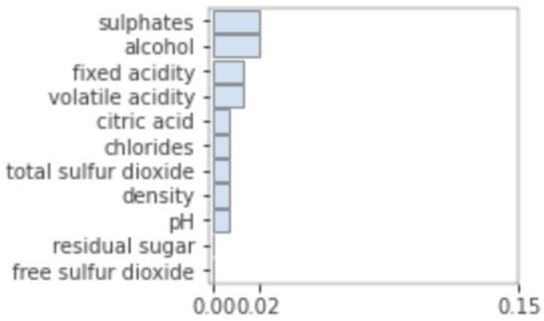
All Features

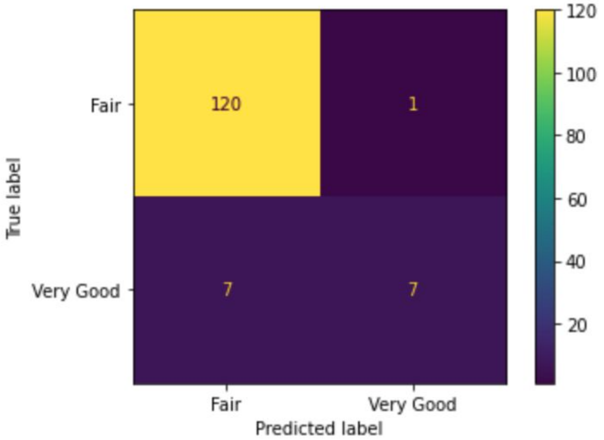


	precision	recall	f1-score	support
Fair	0.93	0.99	0.96	121
Very Good	0.83	0.36	0.50	14
accuracy			0.93	135
macro avg	0.88	0.67	0.73	135
weighted avg	0.92	0.93	0.91	135

TOP Features

Minus 'free sulfur dioxide','residual sugar' and 'pH'

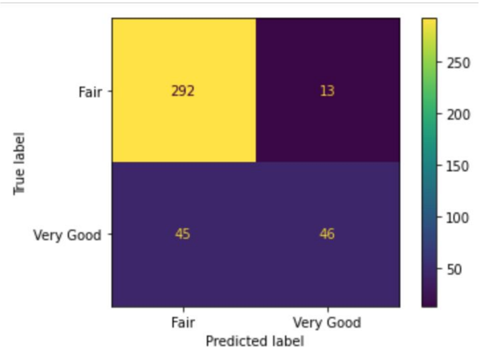




	precision	recall	f1-score	support
Fair	0.94	0.99	0.97	121
Very Good	0.88	0.50	0.64	14
accuracy			0.94	135
macro avg	0.91	0.75	0.80	135
weighted avg	0.94	0.94	0.93	135

Random Forests - White Wine

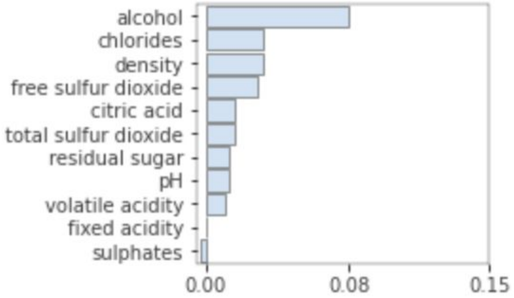
All Features

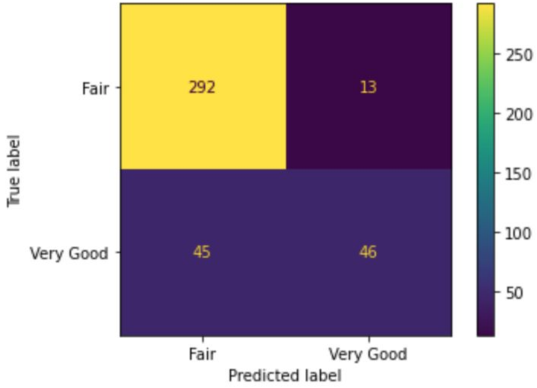


	precision	recall	f1-score	support
Fair	0.87	0.96	0.91	305
Very Good	0.78	0.51	0.61	91
accuracy			0.85	396
macro avg	0.82	0.73	0.76	396
weighted avg	0.85	0.85	0.84	396

TOP Features

Minus 'sulphates','fixed acidity' and 'volatile acidity'.





	precision	recall	f1-score	support
Fair	0.87	0.96	0.91	305
Very Good	0.78	0.51	0.61	91
accuracy			0.85	396
macro avg	0.82	0.73	0.76	396
weighted avg	0.85	0.85	0.84	396

Random Forests Classifier Jupyter Notebook

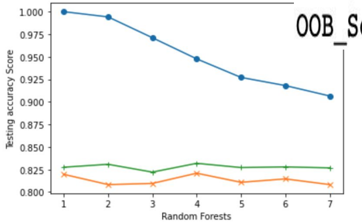
Create a Train Test Split

```
X_train, X_, y_train, y_ = train_test_split(X,y,train_size=.7,random_state=3)
X_val,X_test, y_val, y_test = train_test_split(X_,y_,test_size=.333,random_state=3)

print(X_train.shape, X_val.shape, X_test.shape, y_train.shape, y_val.shape, y_test.shape)
```

```
(2772, 11) (793, 11) (396, 11) (2772,) (793,) (396,)
```

```
#Train your first model with choosen parameters

train_scores = []
val_scores = []
oob_scores = []
for k in range(1, 8, 1):
    rf = RandomForestClassifier(n_estimators=100,
                                min_samples_leaf=k,
                                n_jobs=-1,
                                oob_score=True,
                                criterion='gini')
```

```
k: 4, Train/Test Score: 0.948/0.821
OOB_Score : 0.8318903318903319
```




```
# Create the GridSearch estimator along with a parameter object containing the values to adjust
# Check it out how GridSearchCV can perform

param_grid = {'criterion':['gini','entropy'],
              'n_estimators':[100, 200],
              'min_samples_leaf':[1,2,3,4,5,6]}


grid = GridSearchCV(rf, param_grid, verbose=3, return_train_score=True)
```

```
print(grid.best_params_)
print(grid.best_score_)
```

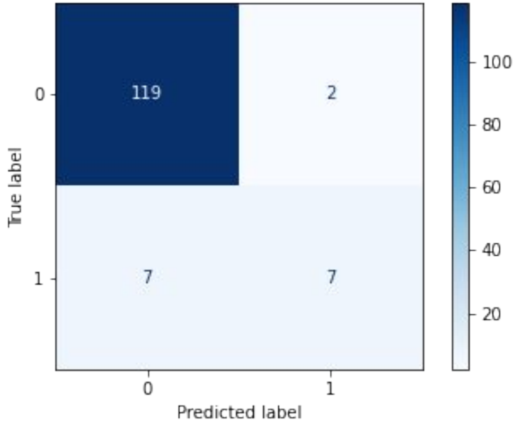
```
{'criterion': 'entropy', 'min_samples_leaf': 1, 'n_estimators': 200}
0.8326132630825771
```

```
#Choose the best parameter and run a final test from one of those:
#1 - Previus tuning or
#2 - GridSearch
best_rf = grid.best_estimator_ #Chose this because GridSearch got a 0.883 score
#best_rf = rf
print(best_rf)
#Run one more time on the train data
best_rf.fit(X_train, y_train)
print(f'Train Data: {best_rf.score(X_train, y_train), best_rf.score(X_val, y_val)}')
#Concatenate the train and test
X_train_c = pd.concat([X_train, X_val], ignore_index=True)
y_train_c = pd.concat([y_train, y_val], ignore_index=True)
best_rf.fit(X_train_c, y_train_c)
print(f'Train and Validation Data Concat: {best_rf.score(X_train_c,y_train_c)}')
print(f'Validation Data : {best_rf.score(X_val,y_val)}')
print(f'Final result Test Data: {best_rf.score(X_test,y_test)}')
```

```
RandomForestClassifier(criterion='entropy', n_estimators=200, n_jobs=-1,
                       oob_score=True)
Train Data: (1.0, 0.8095838587641866)
Train and Validation Data Concat: 1.0
Validation Data : 1.0
Final result Test Data: 0.8535353535353535
```

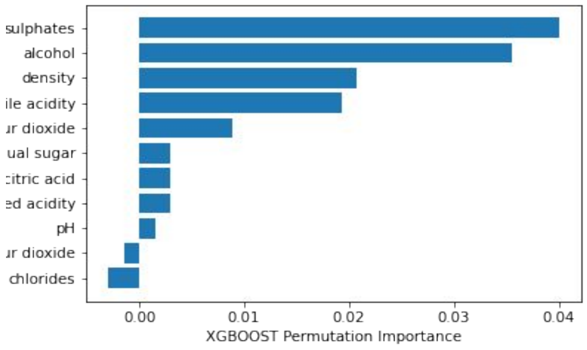
XG Boost- Red Wine

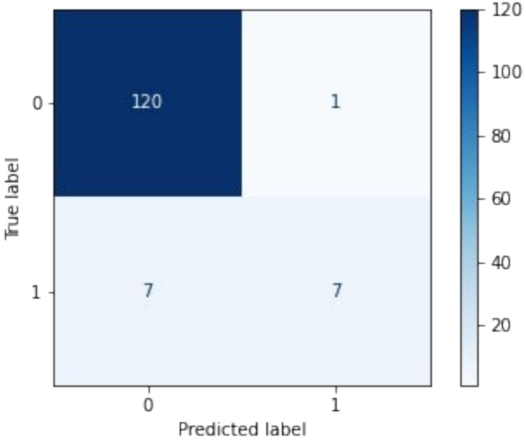
All Features



	precision	recall	f1-score	support
Fair	0.94	0.98	0.96	121
Very Good	0.78	0.50	0.61	14
accuracy			0.93	135
macro avg	0.86	0.74	0.79	135
weighted avg	0.93	0.93	0.93	135

TOP Features
Minus 'chlorides', 'free sulfur dioxide' and 'pH'

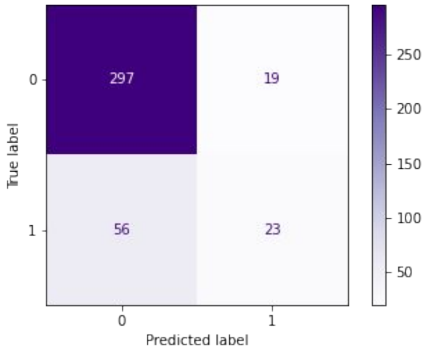




	precision	recall	f1-score	support
Fair	0.94	0.99	0.97	121
Very Good	0.88	0.50	0.64	14
accuracy			0.94	135
macro avg	0.91	0.75	0.80	135
weighted avg	0.94	0.94	0.93	135

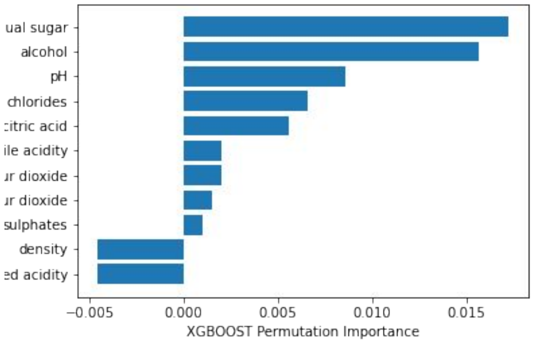
XG Boost- White Wine

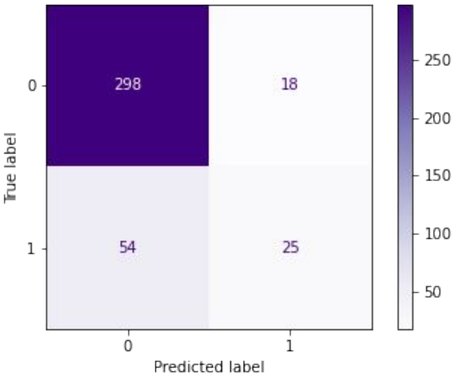
All Features



	precision	recall	f1-score	support
Fair	0.84	0.94	0.89	316
Very Good	0.55	0.29	0.38	79
accuracy			0.81	395
macro avg	0.69	0.62	0.63	395
weighted avg	0.78	0.81	0.79	395

TOP Features
Minus 'fixed acidity','density','sulphates'





	precision	recall	f1-score	support
Fair	0.85	0.94	0.89	316
Very Good	0.58	0.32	0.41	79
accuracy			0.82	395
macro avg	0.71	0.63	0.65	395
weighted avg	0.79	0.82	0.80	395

XGBoost Classifier Jupyter Notebook

Create a Train Test Split ¶

```
# Split the data using train_test_split
X_train, X_, y_train, y_ = train_test_split(X,y,train_size=.7,random_state=3)
X_val,X_test, y_val, y_test = train_test_split(X_,y_,test_size=.332,random_state=3)
X_train.shape, X_val.shape, X_test.shape, y_train.shape, y_val.shape, y_test.shape
```

```
((2769, 11), (793, 11), (395, 11), (2769,), (793,), (395,))
```

```
# Create a StandardScater model and fit it to the training data
X_scaler = StandardScaler().fit(X_train)
```

```
# Transform the training and testing data using the X_scaler and y_scaler models
X_train = X_scaler.transform(X_train)
X_val = X_scaler.transform(X_val)
X_test = X_scaler.transform(X_test)
```

```
# Init classifier
xgb_cl = xgb.XGBClassifier(use_label_encoder=False, objective="binary:logistic")
print(xgb_cl)
xgb_cl.fit(X_train, y_train)
print(f'Train Data Score: {xgb_cl.score(X_train,y_train)}')
print(f'Validation Data Score: {xgb_cl.score(X_val,y_val)}')
```

```
Train Data Score: 0.9992777175875768
Validation Data Score: 0.8272383354350568
```

```
# Create the GridSearch estimator along with a parameter object containing the values to adjust
from sklearn.model_selection import GridSearchCV
param_grid = {
    "max_depth": [3, 4, 5],
    "learning_rate": [0.01, 0.1, 0.2],
    "subsample": [0.5,0.7,0.9],
    "colsample_bytree": [0.5, 0.7, 0.9],
}

# Init classifier
xgb_cl = xgb.XGBClassifier(objective="binary:logistic",use_label_encoder=False)

# Init Grid Search
grid = GridSearchCV(xgb_cl, param_grid, n_jobs=-1, verbose=3, cv=3, scoring="roc_auc",return_train_score=True)
```

```
print(grid.best_params_)
print(grid.best_score_)
```

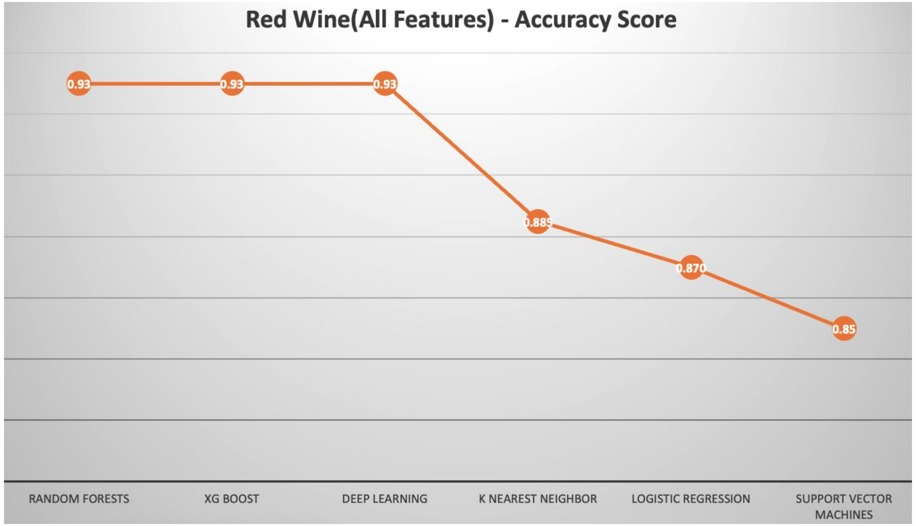
```
{'colsample_bytree': 0.9, 'learning_rate': 0.1, 'max_depth': 3, 'subsample': 0.5}
0.8344659962039832
```

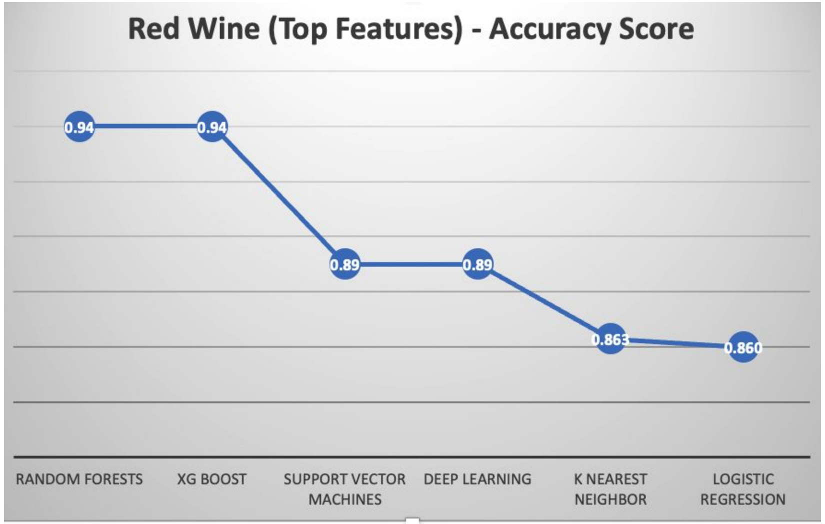
```
#Choose the best parameter and run a final test from one of those:
#1 - Previous tuning or
#2 - GridSearch
#best_xgb_cl = xgb_cl
#or
best_xgb_cl = grid.best_estimator_ #Chose this because GridSearch got a 0.834 score
print(best_xgb_cl)
best_xgb_cl.fit(X_train, y_train)
X_train_c = np.concatenate([X_train, X_val])
y_train_c = np.concatenate([y_train, y_val])
best_xgb_cl.fit(X_train_c, y_train_c)
print(f'Train and Validation Data Concat: {best_xgb_cl.score(X_train_c,y_train_c)}')
print(f'Validation Data: {best_xgb_cl.score(X_val,y_val)}')
print(f'Final result Test Data: {best_xgb_cl.score(X_test,y_test)}')
```

```
Train and Validation Data Concat: 0.8573834924199888
Validation Data: 0.8663303909205549
Final result Test Data: 0.810126582278481
```

Comparisons - Red Wine

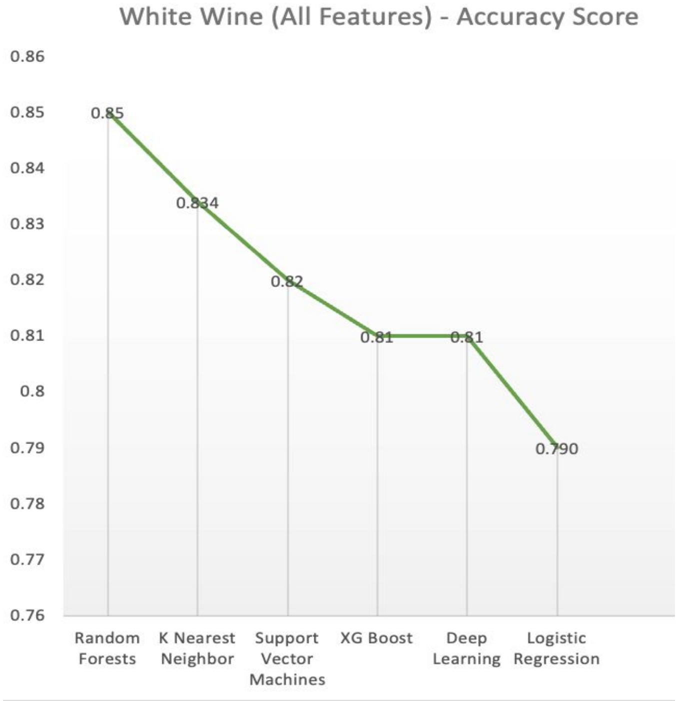
Models Vs Accuracy

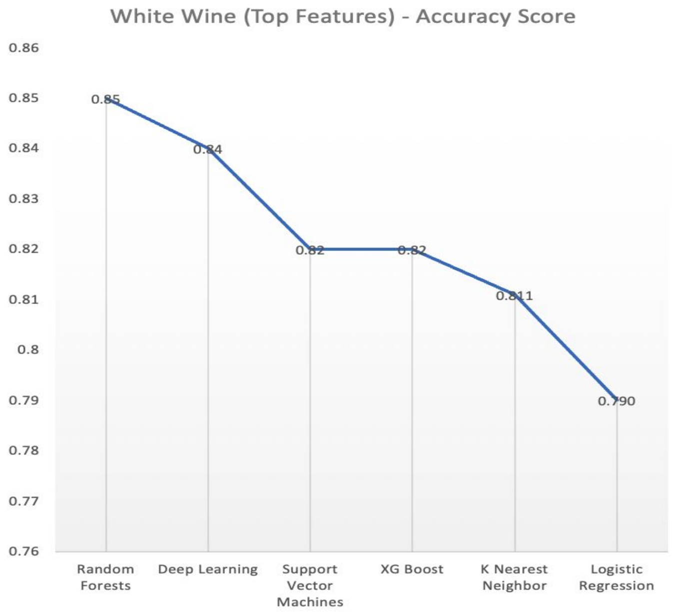




Comparisons - White Wine

Model Vs Accuracy





https://josiedeleon.github.io/Final-Project/

Home

Wine Analyzer

With the use of Machine Learning, can you tell if a wine is good without having to taste it? The purpose of this project is to be able to analyze a variety of different data points to better predict the probability of a wine being good.

Data Source

Two datasets related to red and white Vinho Verde wine samples, from the north of Portugal.