Predicting Wine Quality Using Python: Support Vector Machines, Random Forest and Neural Networks

Applied Machine Learning in sklearn

John Karuitha

2/21/23

Contents

Chapter 1

Background

In this analysis, I use data from the UCI machine learning repository regarding the quality of red wine. The objective is to develop machine learning algorithms that can reasonably predict the wine rating based on a set of variables/features. Specifically, we train 3 machine learning models.

1. Support vector machines (SVM) (Pisner and Schnyer, 2020).
2. Random forest model (Parmar, Katariya and Patel, 2019).
3. Neural network model (Hancock and Khoshgoftaar, 2020).

Read More of my Work

Please visit my rpubs site to see more data projects. Alternatively, copy and paste the link https://www.rpubs.com/Karuitha into your browser.
My data visualizations projects are available in my Tableau Public profile page or copy and paste the link https://public.tableau.com/app/profile/john.karuitha.
My Shiny web apps are available on this site. You can copy-paste this web address instead https://karuitha.shinyapps.io/.

Tools Utilized & Skills Applied

Python, sklearn, matplotlib, pandas, seaborn, numpy, Data Science, Machine Learning

The data that we utilise in this analysis is available on this link. For the purpose of this analysis, I use data for red wine. [1]

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[1] Note: The data is also available for white wine.See https://archive.ics.uci.edu/ml/datasets/wine+quality.

```
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.utils import resample
```

I start by downloading and reading the dataset.

```
# help(pd.read_csv)
wine = pd.read_csv("winequality-red.csv", sep = ";")
wine.head()
```

```
/home/karuitha/anaconda3/envs/py3108/lib/python3.10/site-packages/IPython/core/formatters.py:345: Fut

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `Styler.to_
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0

Next, we look at the number of rows and columns in the dataset.

```
wine.shape
```

```
(1599, 12)
```

```
wine.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
```

Chapter 2

Explore the Data

In this section, we examine the data. First, we list the variables.

The 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

The output variable (based on sensory data):

1. quality (score between 0 and 10)

We then check for missing and duplicate values. We see that there are no missing values.

```
wine.isna().sum()
```

```
/home/karuitha/anaconda3/envs/py3108/lib/python3.10/site-packages/IPython/core/formatters.py:345: Fut

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `Styler.to_
```

	0
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

However, there are 240 duplicated observations that we drop.

```
wine.duplicated().sum()
```

```
240
```

```
wine = wine.drop_duplicates()
```

Next, I do feature engineering by converting the target variable `quality` to a binary variable. As it stands, the wine is in the following categories.

```
wine["quality"].unique()
```

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

Specifically, wines below a rating of () are bad quality while those equal to or above () are good quality wines. Note that this is a personal choice and hence subjective.

Note that after this update, we only have 2 categories for wine quality, 0 and 1.

```
wine["quality"] = [1 if i >= 6.5 else 0 for i in wine["quality"]]

wine.head()
```

```
/home/karuitha/anaconda3/envs/py3108/lib/python3.10/site-packages/IPython/core/formatters.py:345: Fut

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `Styler.to_
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0

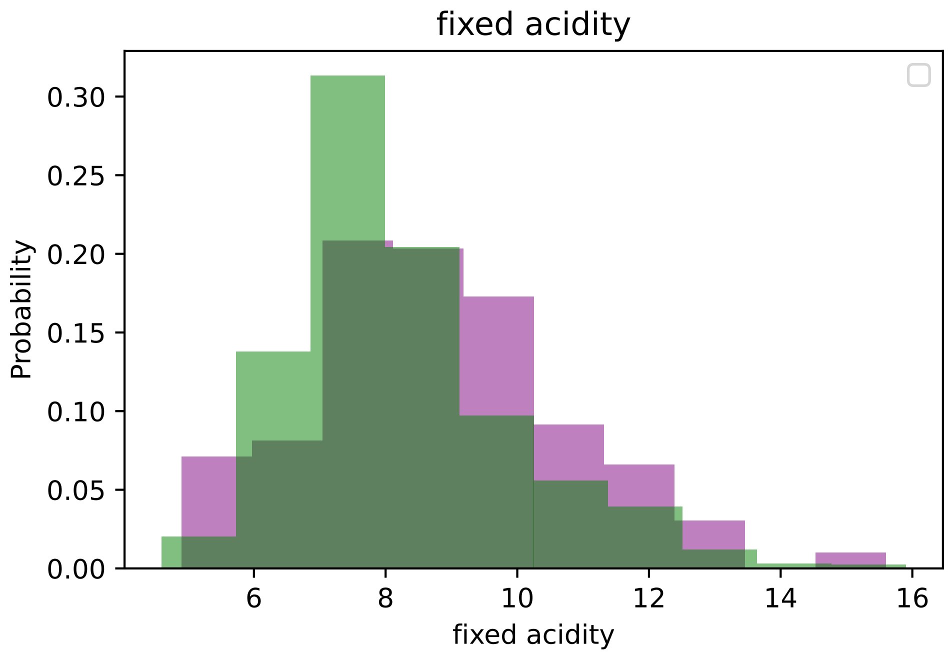
Chapter 3

Data Visualization

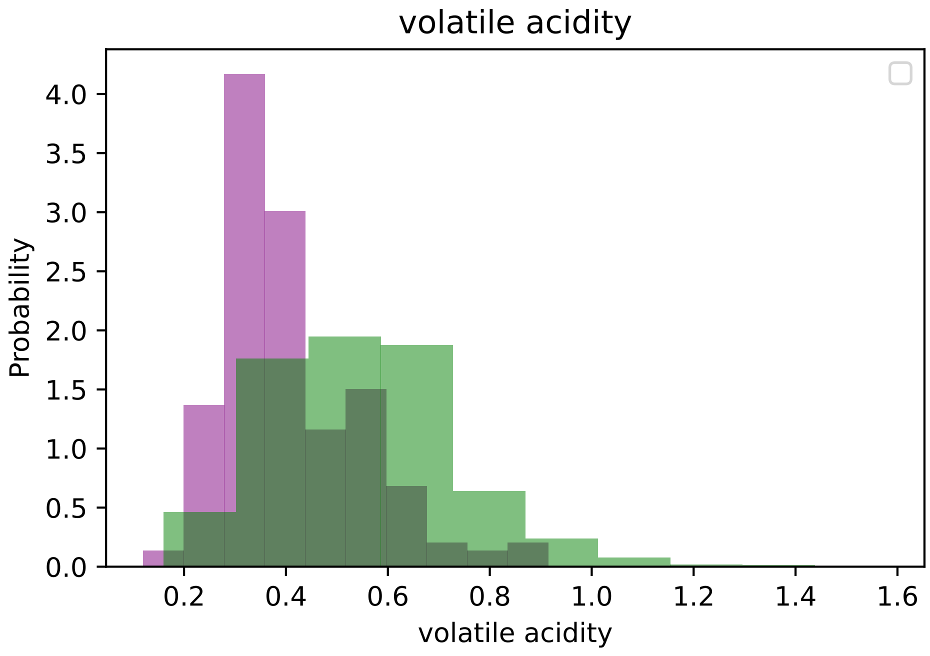
In this section, I visualize the data. To start with, I make a histogram for indepedent variables/features.

```
# plt.subplots(4, 3, figsize=(12, 8))
for variable in wine.columns[:-1]:

    plt.hist(wine[wine["quality"] == 1][variable], color = "purple", alpha = 0.5, density = True)

    plt.hist(wine[wine["quality"] == 0][variable], color = "green", alpha = 0.5, density = True)

    plt.title(variable)
    plt.xlabel(variable)
    plt.ylabel("Probability")
    plt.legend()


    plt.show()
```

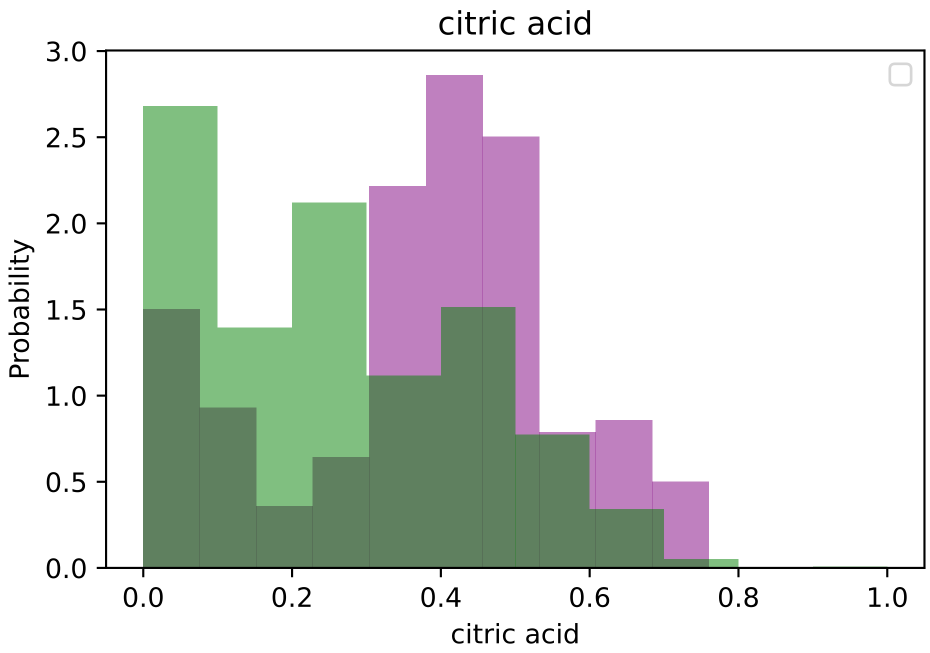
```
No artists with labels found to put in legend.  Note that artists whose label start with an underscor
```



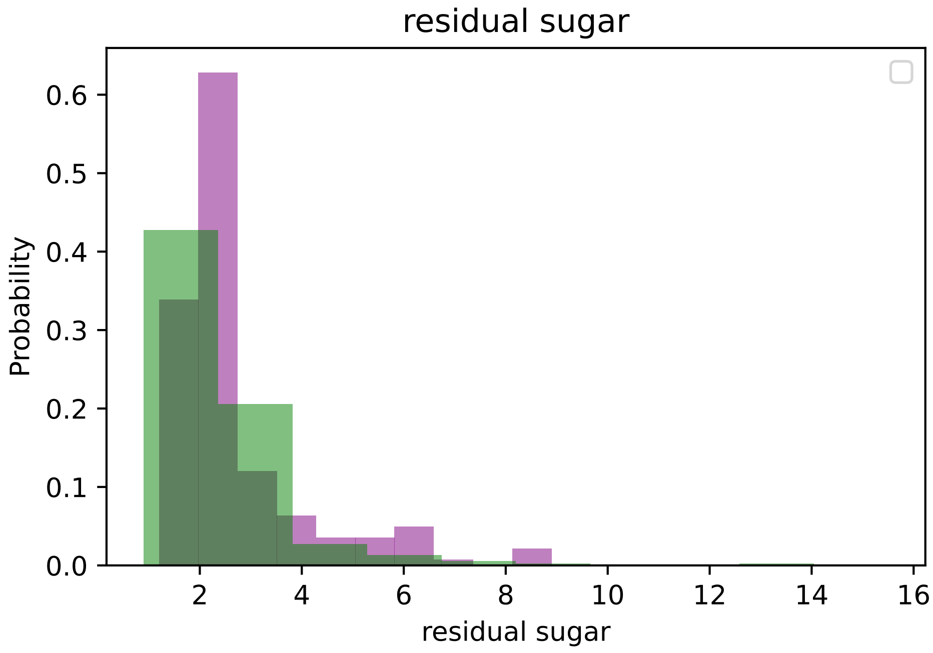
```
No artists with labels found to put in legend.  Note that artists whose label start with an underscor
```



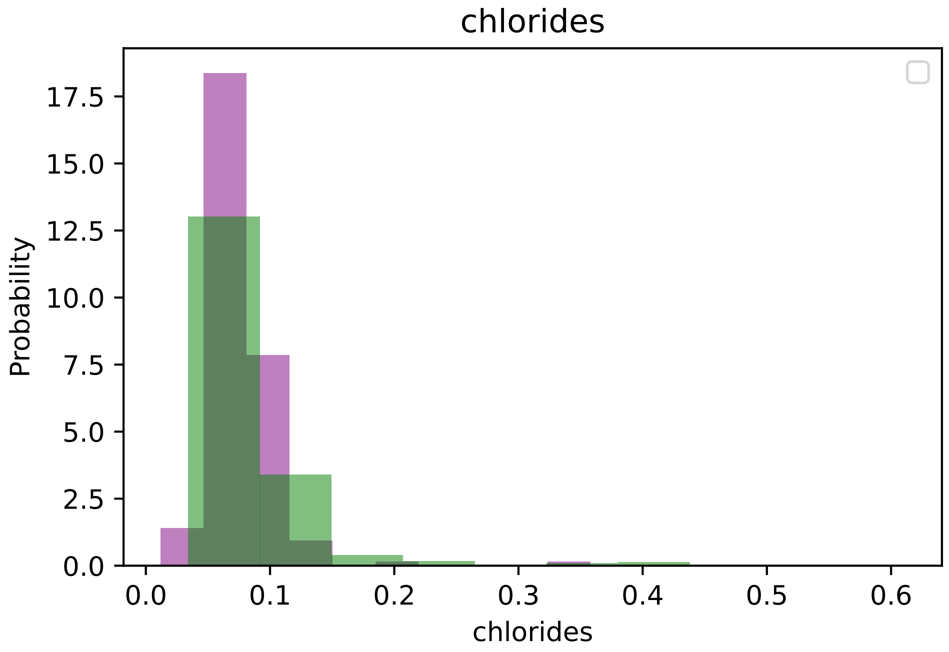
```
No artists with labels found to put in legend.  Note that artists whose label start with an underscor
```



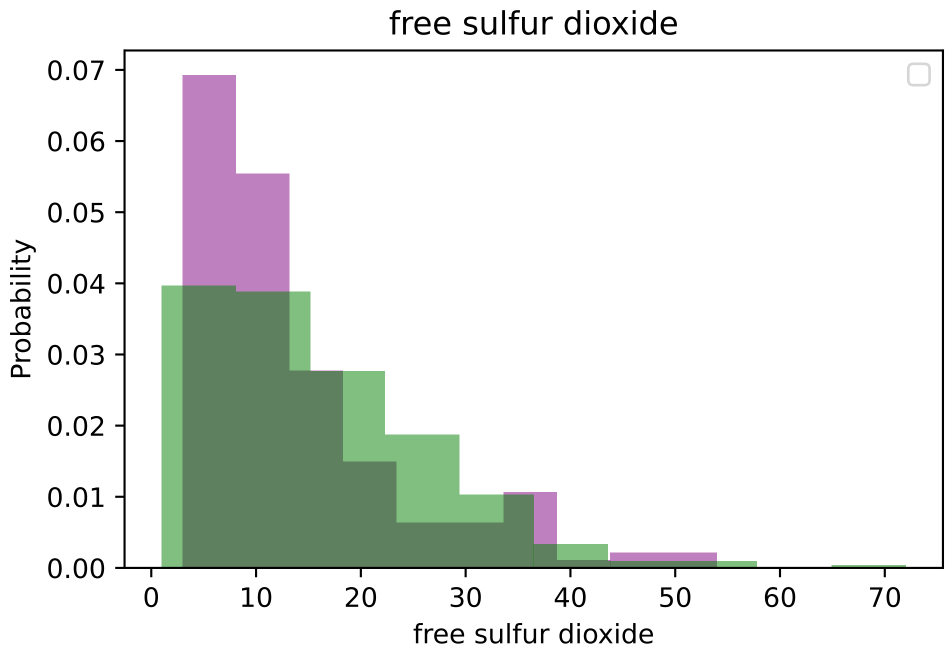
```
No artists with labels found to put in legend.  Note that artists whose label start with an underscor
```



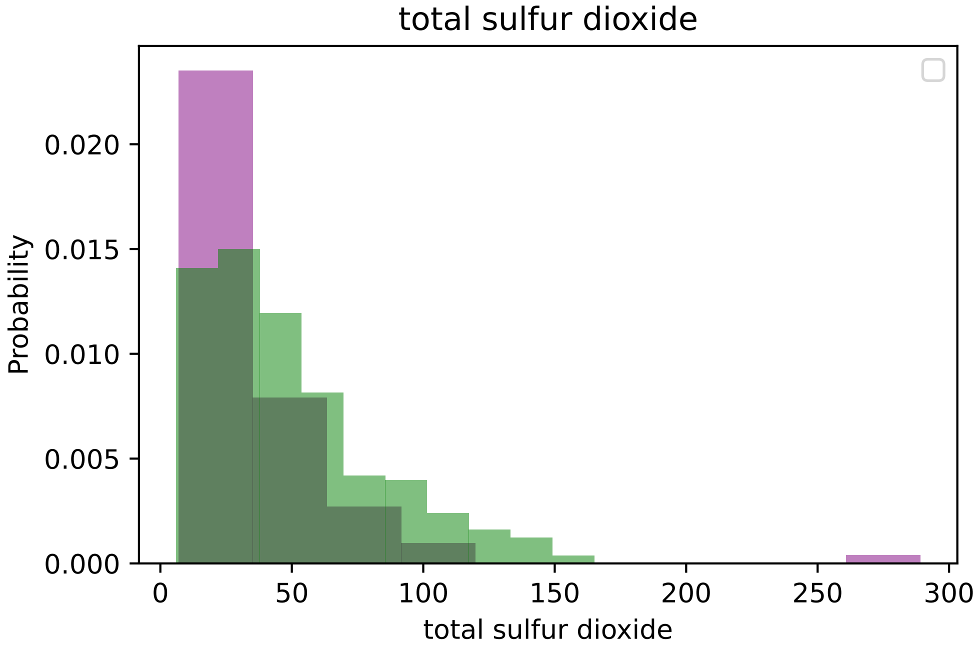
```
No artists with labels found to put in legend.  Note that artists whose label start with an underscor
```



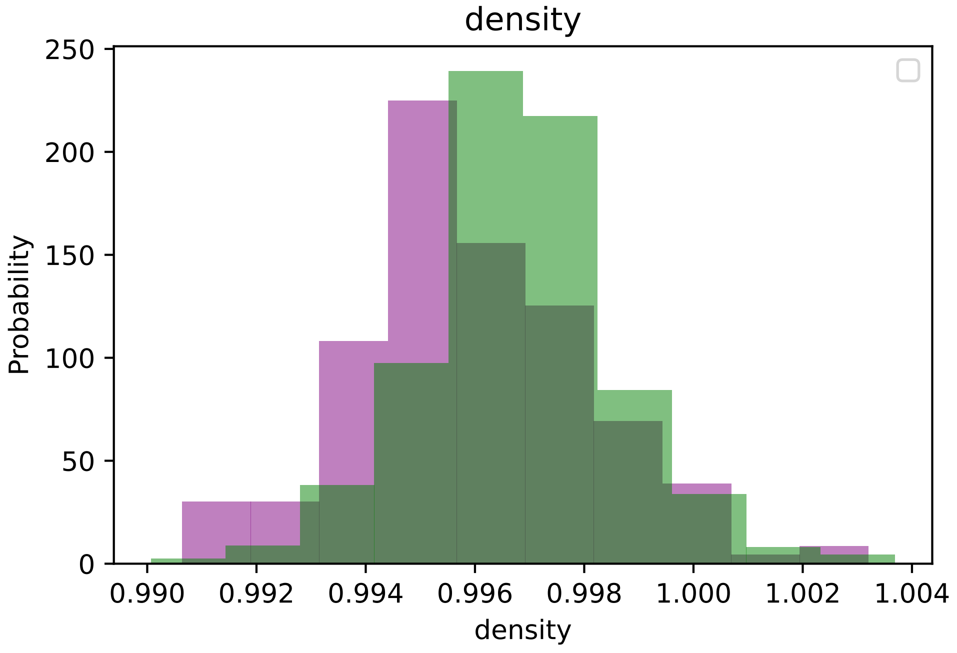
```
No artists with labels found to put in legend.  Note that artists whose label start with an underscor
```



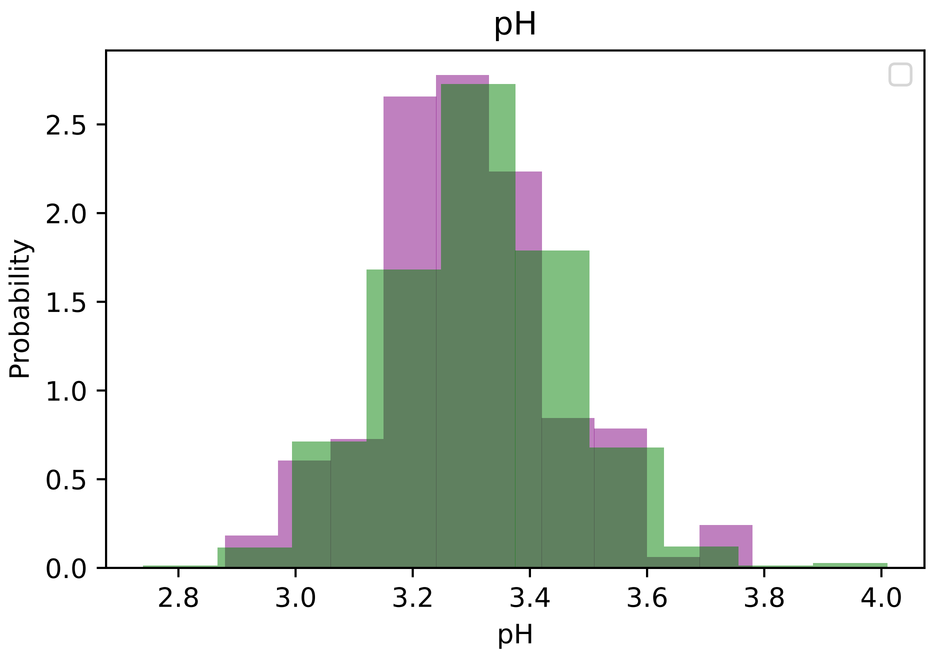
```
No artists with labels found to put in legend.  Note that artists whose label start with an underscor
```



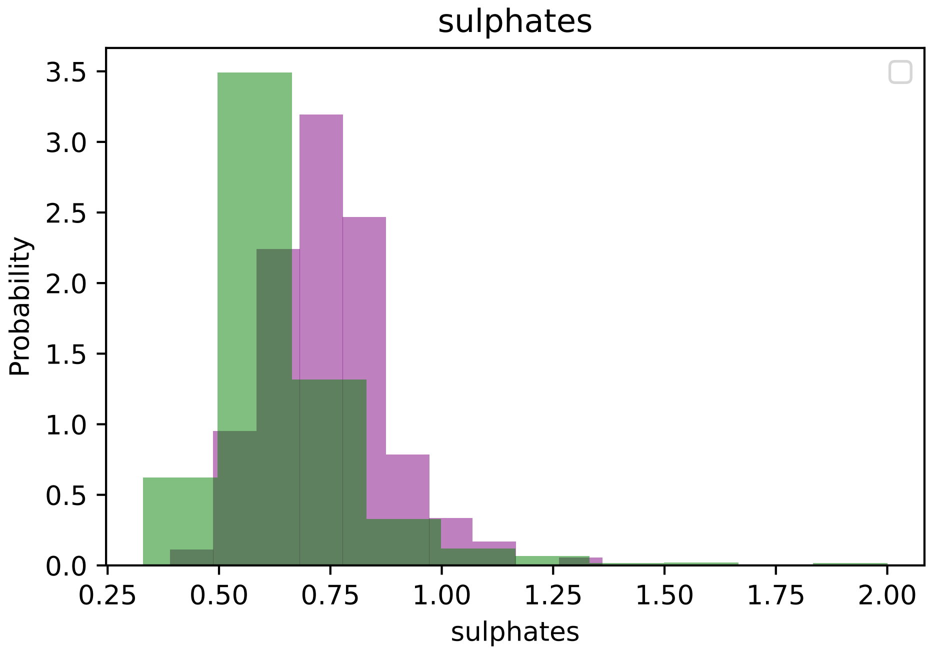
```
No artists with labels found to put in legend.  Note that artists whose label start with an underscor
```



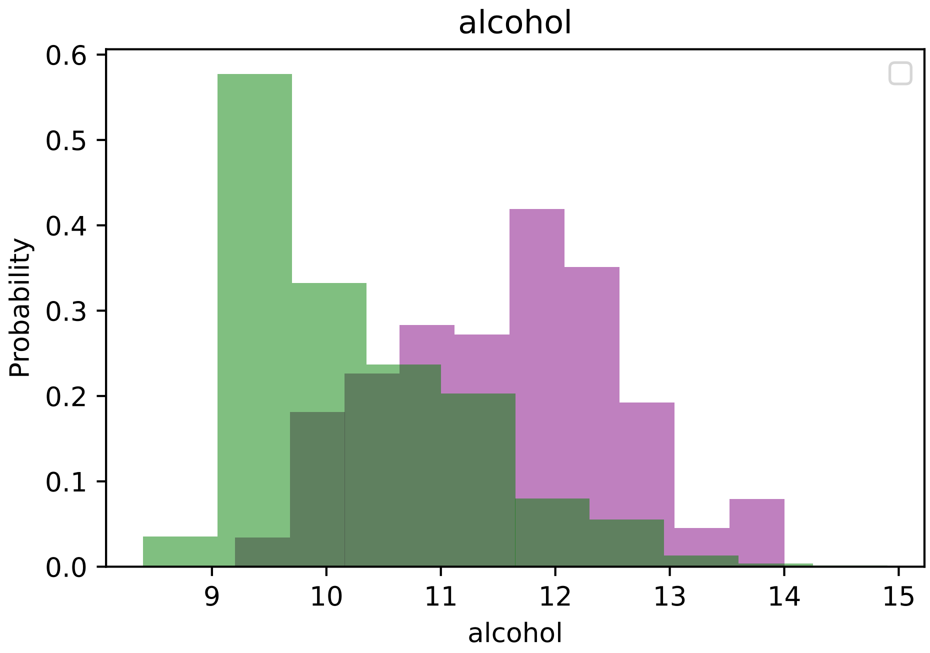
```
No artists with labels found to put in legend.  Note that artists whose label start with an underscor
```



```
No artists with labels found to put in legend.  Note that artists whose label start with an underscor
```



```
No artists with labels found to put in legend.  Note that artists whose label start with an underscor
```



Overall, it appears like Alcohol content is a great disciminator for the quality of wines although the other variables are also useful.

3.1 Evaluating Class Balance/ Imbalance

In this section, we evaluate the class balance in the dataset. As shown below, there is a high degree of the class balance with 217 good wines in the dataset against 1382 bad wines. This level of the class balance could affect the effeciency of the machine learning models. One approach is to upsample the underrepresented class or downsample the overrepresented class. This approach is more cost efficient. The alternative approach is is to get more data for the underrepresented class.

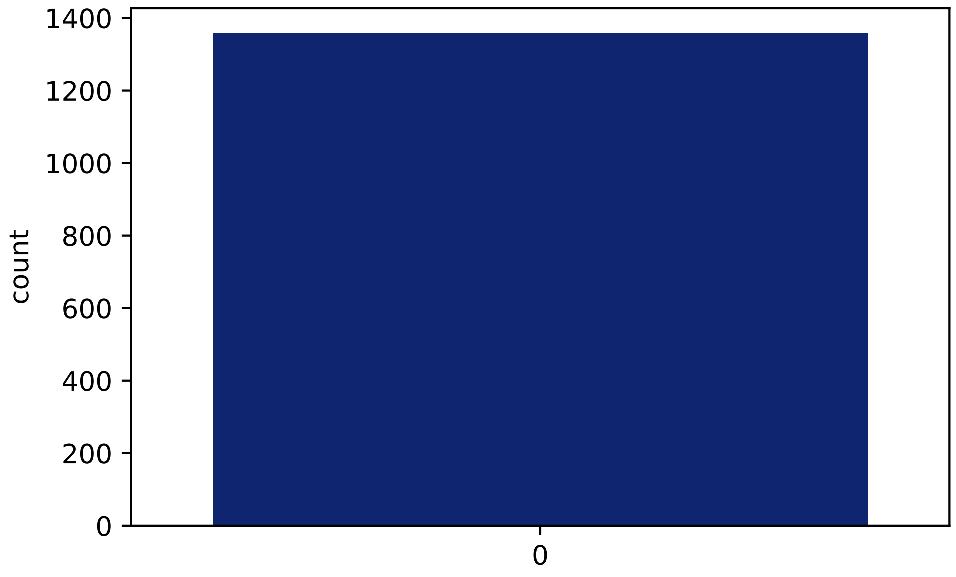
```
wine["quality"].value_counts()
```

```
/home/karuitha/anaconda3/envs/py3108/lib/python3.10/site-packages/IPython/core/formatters.py:345: Fut

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `Styler.to_
```

	quality
0	1175
1	184

```
# help(sns.countplot)
sns.countplot(wine["quality"], palette = "dark")
```

```
<AxesSubplot: ylabel='count'>
```



We shall upsample the training data later.

Chapter 4

Baseline Evaluation Metric

To evaluate whether our models work well, we have to develop the baseline evaluation metric. We pose this question; if a person were to guess that the quality of all wines is bad (remember bad is the dominant class in the data), what would be their accuracy?

```
wine["quality"].value_counts()

1382 / (1382 + 217)
```

```
0.8642901813633521
```

```
[0]*1000
confusion_matrix(wine["quality"], [0] * len(wine))
```

```
array([[1175,    0],
       [ 184,    0]])
```

```
classification_report(wine["quality"], [0] * len(wine))
```

```
/home/karuitha/anaconda3/envs/py3108/lib/python3.10/site-packages/sklearn/metrics/_classification.py:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `

/home/karuitha/anaconda3/envs/py3108/lib/python3.10/site-packages/sklearn/metrics/_classification.py:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `

/home/karuitha/anaconda3/envs/py3108/lib/python3.10/site-packages/sklearn/metrics/_classification.py:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `
```

```
'              precision    recall  f1-score   support\n\n           0       0.86      1.00      0.93
```

In this case, the person would be 86% accurate. Hence, our models have to be more than 86% accurate. For the other scores like precision and recall, we ought to do better than this baseline.

Chapter 5

Training and Testing Sets

In this section we will create a training set and a test set. We set aside 20% of the data for testing and use the remainder for testing.

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

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)

x_train.head()
```

```
/home/karuitha/anaconda3/envs/py3108/lib/python3.10/site-packages/IPython/core/formatters.py:345: Fut

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `Styler.to_
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0
1462	6.8	0.640	0.03	2.3	0.075	14.0	31.0
576	9.9	0.500	0.24	2.3	0.103	6.0	14.0
45	4.6	0.520	0.15	2.1	0.054	8.0	65.0
1352	7.6	0.645	0.03	1.9	0.086	14.0	57.0

Chapter 6

Scaling the Data

The scale of the data could also affect the performance of the machine learning models. For instance, if one variable is in the millions (e.g. 5,233,150) while another is a fraction (e.g. 0.5), the models will likely pick the signal in the larger value more than the fraction. However, the fraction could also contain a valuable signal that is masked due to the scale.

```
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
x_train[:3]
```

```
array([[ 1.65650881, -1.38004607,  1.45916152, -0.44870117, -0.26606052,
         0.1113736 ,  0.37699549,  0.68484681, -0.95193783, -0.44930832,
        -0.57838334],
       [-0.85880973,  0.62805874, -1.24411385, -0.16971466, -0.26606052,
        -0.17258989, -0.46366724, -0.6609637 ,  0.32059051, -0.44930832,
        -0.03276604],
       [ 0.91334651, -0.1528709 , -0.17300474, -0.16971466,  0.30424851,
        -0.92982588, -0.95646954,  0.57929305,  0.19333768, -0.7943441 ,
        -0.39651091]])
```

Now our data is in the same scale.

Chapter 7

Training the Models

In this section we train the three models in the following order:

- Support vector machines model.
- Random forest model.
- Neural network model.

7.1 Support Vector Machines

In this section, we train the support vector machines model.

```
clf = svm.SVC()
clf.fit(x_train, y_train)

clf_predictions = clf.predict(x_test)

confusion_matrix(y_test, clf_predictions)
```

```
array([[234,   4],
       [ 22,  12]])
```

```
classification_report(y_test, clf_predictions)
```

```
'              precision    recall  f1-score   support\n\n           0       0.91      0.98      0.95
```

Again this model does better than the baseline model.

7.2 Random Forest Model

Next, we fit the random forest model with 1000 trees in the forest.

```
rf_model = RandomForestClassifier(n_estimators=1000)

rf_model.fit(x_train, y_train)
```

```
RandomForestClassifier(n_estimators=1000)
```

We examine the metrics of the model, that is how it performs in the testing set. We start by doing the prediction using the model and then evaluate the performance of the model on the testing set. In this case, we have an accuracy of 94%, sensitivity at 98%, and specificity at 96%. This is above the base metrics in section 4.

```
predictions_rfm = rf_model.predict(x_test)

classification_report(y_test, predictions_rfm)
```

```
'              precision    recall  f1-score   support\n\n           0       0.93      0.97      0.95
```

```
confusion_matrix(y_test, predictions_rfm)
```

```
array([[230,   8],
       [ 18,  16]])
```

7.3 Neural Network Model

Neural nets work well with huge amounts of data, more so text, images and other unstructured data. The accuracy of 89% is marginally above the baseline metric of 86% in section 4.

```
mlp = MLPClassifier(hidden_layer_sizes = (11, 11, 11), max_iter = 1000)

mlp.fit(x_train, y_train)

mlp_predictions = mlp.predict(x_test)

confusion_matrix(y_test, mlp_predictions)
```

```
/home/karuitha/anaconda3/envs/py3108/lib/python3.10/site-packages/sklearn/neural_network/_multilayer_

Stochastic Optimizer: Maximum iterations (1000) reached and the optimization hasn't converged yet.
```

```
array([[225,  13],
       [ 18,  16]])
```

```
classification_report(y_test, mlp_predictions)
```

```
'              precision    recall  f1-score   support\n\n           0       0.93      0.95      0.94
```

Chapter 8

Conclusion

In this analysis, we have trained the following classification models to predict the quality of red wine.

- Support vector machines model.
- Random forest model.
- Neural network model.

The baseline accuracy was 86% and the all the models seem to outperform this baseline accuracy. The random forest model does better than the neural network model and the support vector machine model. The models could be improved through hyperparameter tuning and the upsampling or downsampling of the underrepresented class in the dependent or outcome variable.

References

Hancock, J.T. and Khoshgoftaar, T.M. (2020) ‘Survey on categorical data for neural networks’, *Journal of Big Data*, 7(1), pp. 1–41.

Parmar, A., Katariya, R. and Patel, V. (2019) ‘A review on random forest: An ensemble classifier’, in *International conference on intelligent data communication technologies and internet of things (ICICI) 2018*. Springer, pp. 758–763.

Pisner, D.A. and Schnyer, D.M. (2020) ‘Support vector machine’, in *Machine learning*. Elsevier, pp. 101–121.