Wine Classification
A Comprehensive Data Science Journey

Department of Computer Science and Engineering, The LNM Institute of Information Technology, Jaipur.

15th May 2023

Project Objective

The goal of this project is to apply appropriate ML Classification algorithms to the data set and draw conclusions from it.

Following pre-processing, we would perform a preliminary analysis of the dataset and classify our data into appropriate categories with proper validation using various machine learning algorithms.

Source of Dataset

The dataset was sourced from the UCI Machine Learning repository and can be found here https://archive.ics.uci.edu/dataset/109/wine .

Introduction to dataset

Three different cultivators grew wines in a particular region of Italy, and their chemical composition analyses are included in our dataset. Based on the amounts of the different (13), constituents in the wine, the differentiation is carried out.
As part of this project, the wine data will be categorized into one of three groups, each of which corresponds to a different cultivator.

Data Description

Following are the details of our dataset.

- Multivariate Dataset
- Number of Attributes: 13
    - Alcohol
    - Malic acid
    - Ash
    - Alkalinity of ash
    - Magnesium

- Total phenols
- Flavonoids
- Non-Flavonoid phenols
- Proanthocyanins
- Color intensity
- Hue
- OD280/OD315 of diluted wines
- Proline

- Attribute Characteristics: Integer, Float
- Number of Instances: 178

Implementation

We begin by bringing in the necessary libraries:

1. **Numpy**: Short for 'numerical python', Numpy is a python library used for various math and science tasks. It includes multi-dimensional arrays and matrices, as well as the math functions that work on them.

2. **Pandas**: Pandas provides objects like data frames and series for displaying and processing data. The above two libraries are key for working with the dataset, which is essentially a 178x13 matrix.

3. **Matplotlib and Seaborn**: These libraries are used for data visualization. They offer a variety of graphs and plots. Seaborn actually uses matplotlib to create its plots.

4. **SciKit Learn**: This is the main library used in the project for applying classification models and implementing preprocessing and cross-validation steps. We first use the model selection method to split the dataset into training and testing parts. We also use the sklearn.metrics module, which implements metrics like loss, accuracy, score, etc. These metrics help evaluate the accuracy of our model. Specifically, we use the confusion matrix, accuracy. The confusion matrix shows the number of false positives, true negatives, true positives, and false negatives.

As the name suggests, accuracy tells us how accurate our model is. In multi-label classification, the predicted label set should exactly match the respective input label set.

Importing Dependencies

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import warnings
        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import KFold
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.naive_bayes import GaussianNB
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import classification_report
        from sklearn.exceptions import DataConversionWarning
        from sklearn.ensemble import RandomForestClassifier
```

Importing the Dataset

The dataset, stored as *wines.csv (*downloaded from the official source as cited above), was read using the ***read_csv*** commandof pandas and further stored as a DataFrame. CSV is a delimited text file that uses commas to separate values. We have stored our data in the df variable here.

Importing Dataset

```
In [2]: df = pd.read_csv('wines.csv')
```

Understanding the Dataset

To get a better grasp of the dataset we were dealing with, we used the ***df.info()*** command to find out the type and count of the attributes. This command provides a brief summary of the dataset and doesn't return anything. It displays information about the data frame, including the number of instances for each attribute and their data types.

Information About Dataset

```
In [3]: # Concise Summary
        df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Class                         178 non-null    int64
 1   Alcohol                       178 non-null    float64
 2   Malic_acid                    178 non-null    float64
 3   Ash                           178 non-null    float64
 4   Alcalinity_of_ash             178 non-null    float64
 5   Magnesium                     178 non-null    int64
 6   Total_phenols                 178 non-null    float64
 7   Flavanoids                    178 non-null    float64
 8   Nonflavanoid_phenols          178 non-null    float64
 9   Proanthocyanins               178 non-null    float64
 10  Color_intensity               178 non-null    float64
 11  Hue                           178 non-null    float64
 12  OD280-OD315_of_diluted_wines  178 non-null    float64
 13  Proline                       178 non-null    int64
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

We also obtained a rough overview of the dataset using the ***df.head()*** command. This command returns the top n instances of the dataset it is called on. The default value of n is 5 . Therefore we can see our command has returned the first 5 instances of the database.
We found it useful to get a quick, preliminary overview of our data.

Further, we used ***df.describe()*** to understand the ranges and value distribution of our attributes. As all our attributes were numerical, this helped us get a fair idea of the dataset.

The ***describe()*** command gives a summary of basic statistical details for the attributes in the DataFrame. This includes count, mean, standard deviation, minimum, maximum, and all quartiles for each column.

These values are very important for deciding the pre-processing steps. Based on these, we can determine if our data needs handling for outliers, normalization, or any other type of processing.

```
In [4]: # Overview of Dataset
        df.head()
```

Out[4]:

	Class	Alcohol	Malic_acid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocyanins	Color_intensity	Hue	OD
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	

```
In [5]: # Preliminary Statistical Summary
        df.describe()
```

Out[5]:

	Class	Alcohol	Malic_acid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocyanins	Color
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	1
mean	1.938202	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	
std	0.775035	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.572359	
min	1.000000	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	
25%	1.000000	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.250000	
50%	2.000000	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.555000	
75%	3.000000	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	
max	3.000000	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	

Visualizing the Dataset

Using the EDA techniques, we maximized our insights from the dataset. This helped us understand outliers and anomalies present and deal with them.

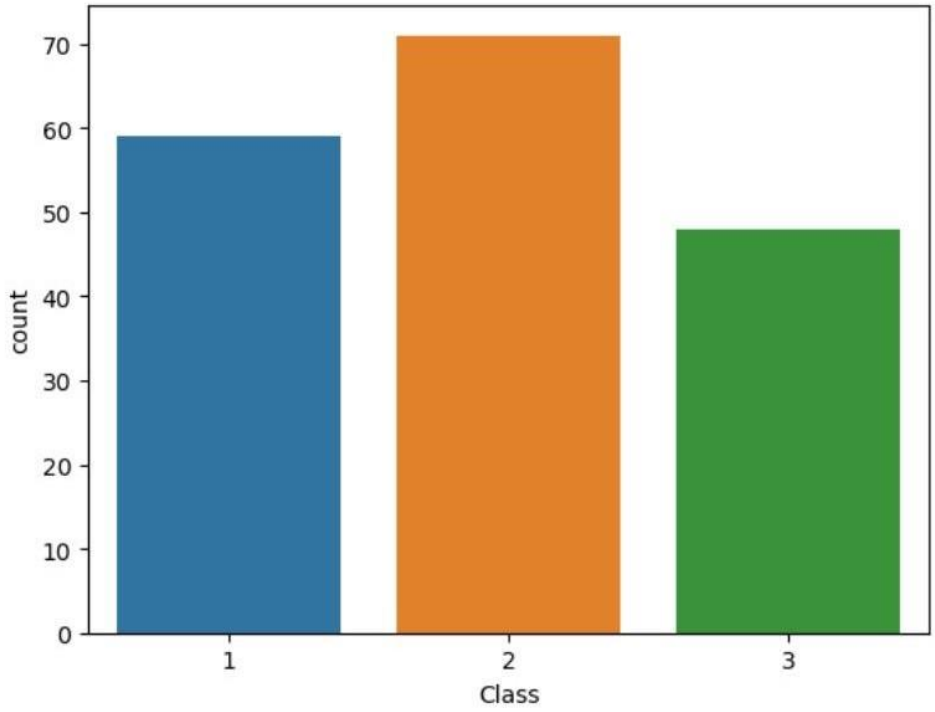
- Numeric Distribution of Wine Categories

  Here, we plotted the different categories of output, i.e. the varieties of wine, through a countplot. Similar to histograms, countplot is instrumental in comparing categorical data in numerical terms.

Exploratory Data Analysis (EDA)

```
In [6]: # Count of Different Classes
        sns.countplot(x = df['Class'])
```

```
Out[6]: <AxesSubplot:xlabel='Class', ylabel='count'>
```

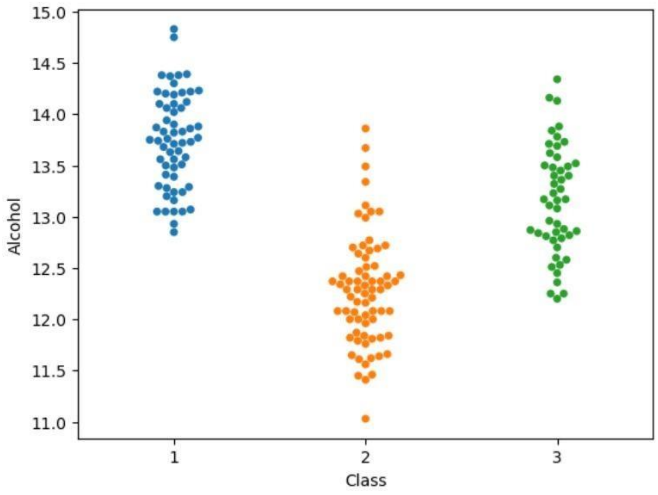


- Categories of wine against Alcohol content

Here, we plotted the different categories of output, i.e. the varieties of wine, against the Alcohol content in them through a swarmplot. Heavier the concentration of plot points, the more the number of instances falling on that value. This helps us identify the range of alcohol content and the most frequent alcohol content in each wine.

```
In [7]: # Swarm Plot
        sns.swarmplot(x = df['Class'], y = df['Alcohol'])
```

```
Out[7]: <AxesSubplot:xlabel='Class', ylabel='Alcohol'>
```

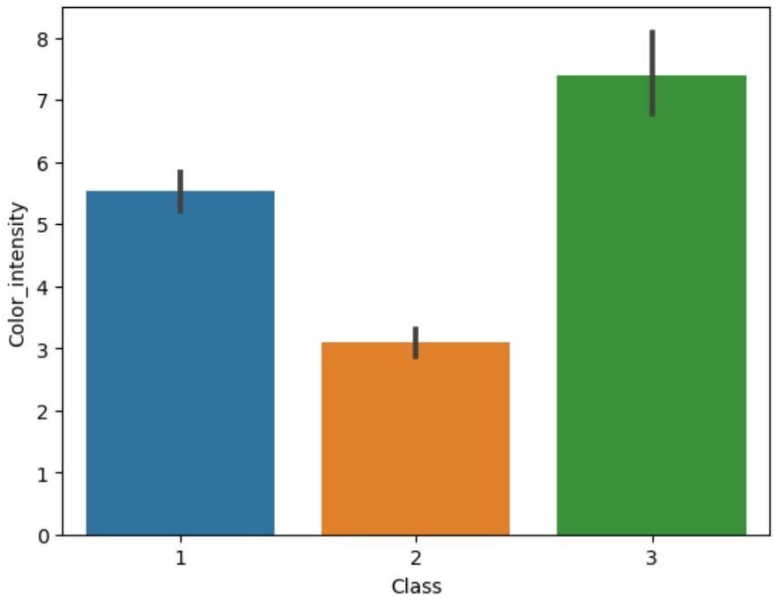


- Categories of wine against Color Intensity

Here, we plotted the different categories of output, i.e., the varieties of wine against the color intensity in them through a barplot.

```
In [9]: # Bar Plot
        sns.barplot(x = df['Class'], y = df['Color_intensity'])
```

```
Out[9]: <AxesSubplot:xlabel='Class', ylabel='Color_intensity'>
```

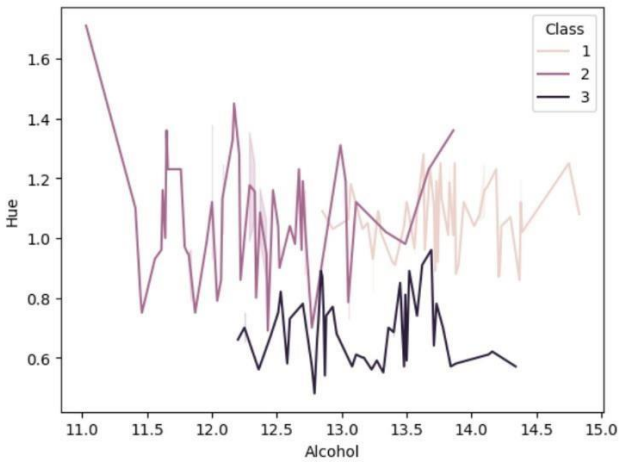


- Alcohol content against Hue w.r.t. Categories of wine

Here, we plotted the alcohol content against hue for each category of wine through a line plot. Each line shows the relation between the two variables for each wine variety.

```
In [11]: # Line Plot
         sns.lineplot(x = df['Alcohol'], y = df['Hue'], hue = df['Class'])
```

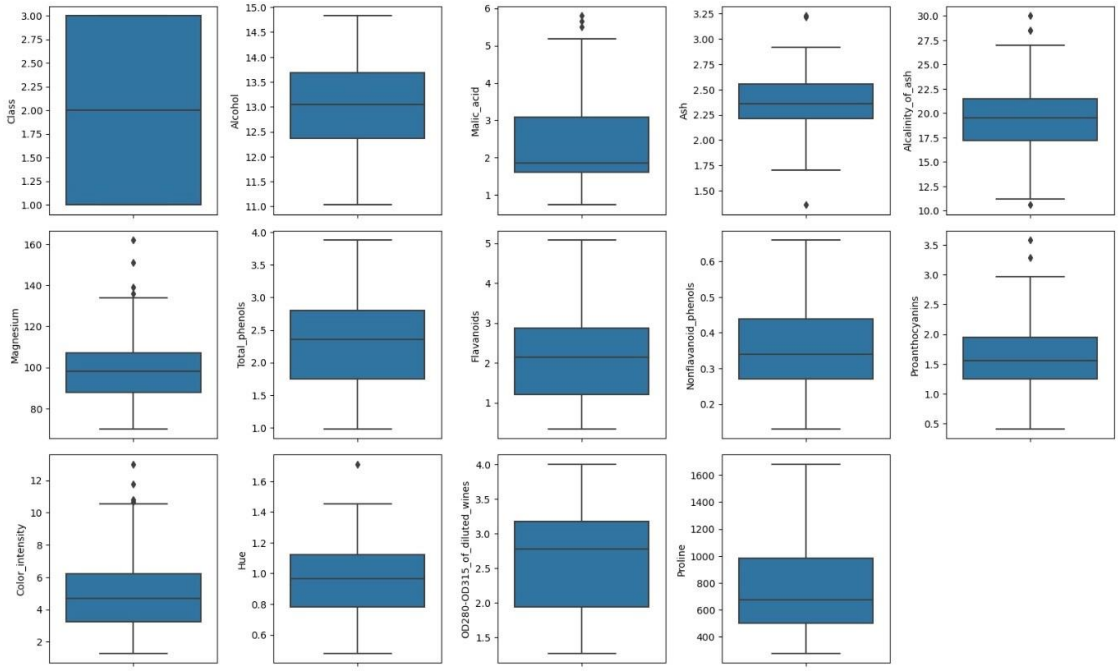
```
Out[11]: <AxesSubplot:xlabel='Alcohol', ylabel='Hue'>
```



- Boxplot

Another essential tool, boxplot helps us visualize the 5 statistical measures of an attribute (Minimum, Maximum, Median and the two quartiles).

```
In [12]: # Boxplot for Every Column
         plt.figure(figsize = (16, 16))
         for j, attr in enumerate(list(df.columns.values)):
           plt.subplot(5, 5, j+1)
           sns.boxplot(y = df[attr])
           plt.tight_layout()
```



Studying the Correlation

Next is an essential part of preprocessing we find out how well are attributes related (the degree of relationship between our variables). We can use this correlation to further predict how a change in the value of one attribute affects the other.

- Correlation Matrix
  Next we found the numeric correlation between the attributes using the ***.corr***() command.

```
In [13]: correlationmatrix = df[df.columns].corr()
         correlationmatrix
```

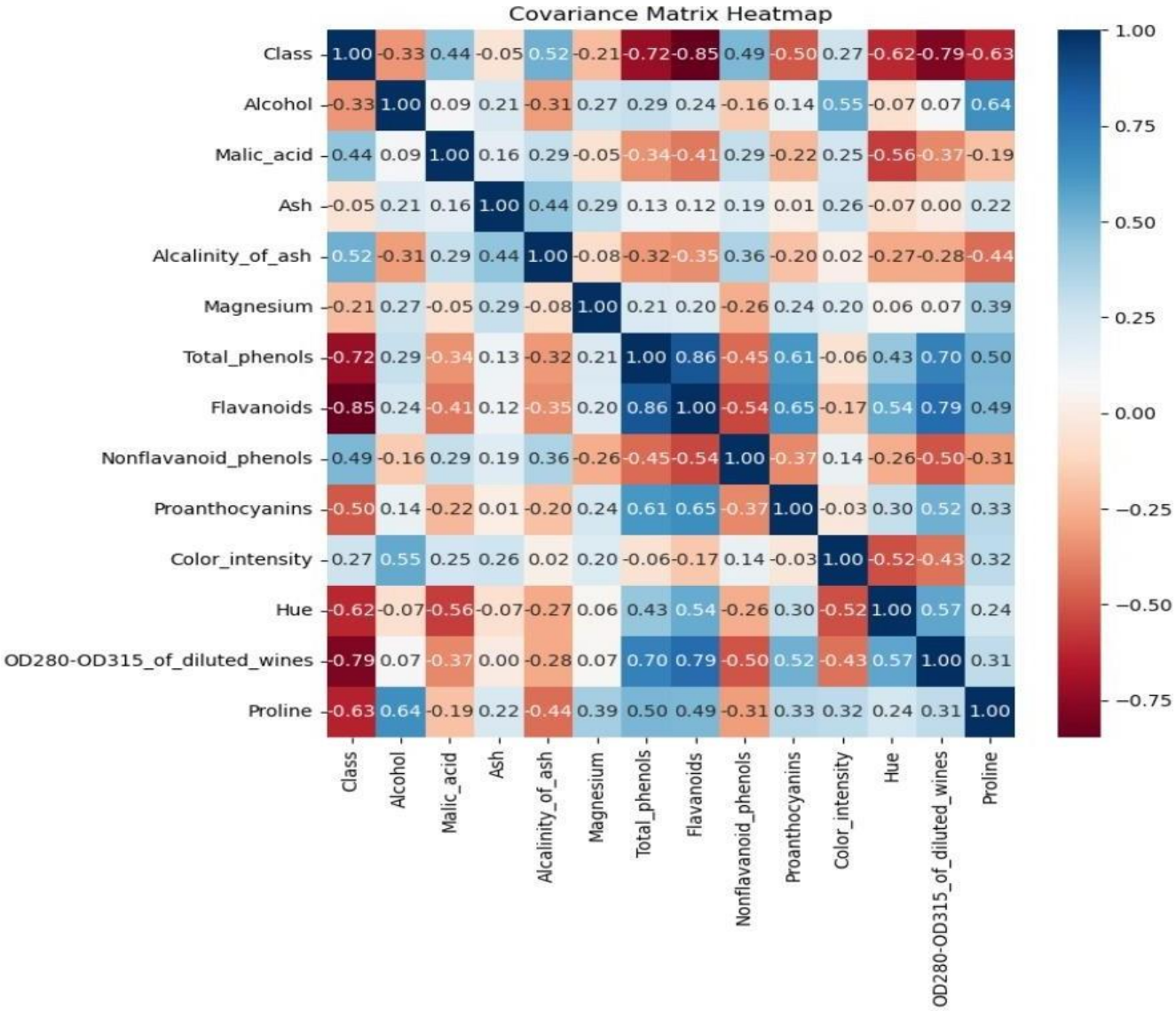
Out[13]:

	Class	Alcohol	Malic_acid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocy
Class	1.000000	-0.328222	0.437776	-0.049643	0.517859	-0.209179	-0.719163	-0.847498	0.489109	-0.4
Alcohol	-0.328222	1.000000	0.094397	0.211545	-0.310235	0.270798	0.289101	0.236815	-0.155929	0.1
Malic_acid	0.437776	0.094397	1.000000	0.164045	0.288500	-0.054575	-0.335167	-0.411007	0.292977	-0.2
Ash	-0.049643	0.211545	0.164045	1.000000	0.443367	0.286587	0.128980	0.115077	0.186230	0.0
Alcalinity_of_ash	0.517859	-0.310235	0.288500	0.443367	1.000000	-0.083333	-0.321113	-0.351370	0.361922	-0.1
Magnesium	-0.209179	0.270798	-0.054575	0.286587	-0.083333	1.000000	0.214401	0.195784	-0.256294	0.2
Total_phenols	-0.719163	0.289101	-0.335167	0.128980	-0.321113	0.214401	1.000000	0.864564	-0.449935	0.6
Flavanoids	-0.847498	0.236815	-0.411007	0.115077	-0.351370	0.195784	0.864564	1.000000	-0.537900	0.6
Nonflavanoid_phenols	0.489109	-0.155929	0.292977	0.186230	0.361922	-0.256294	-0.449935	-0.537900	1.000000	-0.3
Proanthocyanins	-0.499130	0.136698	-0.220746	0.009652	-0.197327	0.236441	0.612413	0.652692	-0.365845	1.0
Color_intensity	0.265668	0.546364	0.248985	0.258887	0.018732	0.199950	-0.055136	-0.172379	0.139057	-0.0
Hue	-0.617369	-0.071747	-0.561296	-0.074667	-0.273955	0.055398	0.433681	0.543479	-0.262640	0.2
OD280-OD315_of_diluted_wines	-0.788230	0.072343	-0.368710	0.003911	-0.276769	0.066004	0.699949	0.787194	-0.503270	0.5
Proline	-0.633717	0.643720	-0.192011	0.223626	-0.440597	0.393351	0.498115	0.494193	-0.311385	0.3

- Heatmap

  To simplify the analysis, we plotted this correlation in the form of a heatmap. The underlying concept of heatmaps is in the normalized correlation between the attributes. More the correlation, lighter the colour of their box (shown in the index below)

```
In [14]: plt.figure(figsize=(8, 8))
         sns.heatmap(correlationmatrix, annot = True, cmap = 'RdBu', fmt = '.2f')
         plt.title('Covariance Matrix Heatmap')
         plt.show()
```



Looking at the heatmap, we observed that the correlation of ‘Ash’ and ‘Magnesium’ with ‘Class’ was extremely low (i.e., -0.05 & -0.12 respectively) so we decided to drop it from our dataset.

In [15]:
```
# Very Low Correlation
df = df.drop('Ash', axis = 1)
df = df.drop('Magnesium', axis = 1)
```

Splitting the Dataset

We divide our data into two parts: one for training and one for testing. This helps us see how well different methods work on the data. We use the training part to teach our model using a certain method. After that, we test its accuracy by using the testing data. This gives us a good understanding of how well the method works.

Splitting the Dataset for ML Models

```
In [16]: # Feature Vector (Independent Variable) Containing All Columns Except Class Label
         X = df
         X = X.drop(['Class'],axis=1)
```

```
In [17]: # Response Variable
         Y = df['Class']
```

```
In [18]: # Overview of X
         X.head(10)
```

Out[18]:

	Alcohol	Malic_acid	Alcalinity_of_ash	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocyanins	Color_intensity	Hue	OD280-OD315_of_diluted_wines	Pro
0	14.23	1.71	15.6	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1
1	13.20	1.78	11.2	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1
2	13.16	2.36	18.6	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1
3	14.37	1.95	16.8	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1
4	13.24	2.59	21.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	
5	14.20	1.76	15.2	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1
6	14.39	1.87	14.6	2.50	2.52	0.30	1.98	5.25	1.02	3.58	1
7	14.06	2.15	17.6	2.60	2.51	0.31	1.25	5.05	1.06	3.58	1
8	14.83	1.64	14.0	2.80	2.98	0.29	1.98	5.20	1.08	2.85	1
9	13.86	1.35	16.0	2.98	3.15	0.22	1.85	7.22	1.01	3.55	1

```
In [19]: # Overview of Y
         Y.head()
```

```
Out[19]: 0    1
         1    1
         2    1
         3    1
         4    1
         Name: Class, dtype: int64
```

We've divided the data into a 6:4 ratio for training and testing. The 'random state' parameter controls how the data is split into training and testing parts. It's important to set this parameter because it makes sure that the same split is made every time we run the code.

```
In [20]: X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size = 0.60, random_state = 42)
```

Training the Dataset

We used our training data to train three models i.e. K Nearest Neighbours, Naive Bayes and Random Forest.

Since labels are available in this dataset, we go with three approaches of supervised learning and pick the model with the most accuracy.

❖ KNN (K-Nearest Neighbor Algorithm)

This is a type of learning where we already know the outcome. Also known as Supervised Learning. It's used to sort things into groups. When we have a new point to label, it looks at the "k" closest neighbours and sees what group they belong to. Then it decides what group the new point should be in based on that.

We test the KNN model for many K values. Finally we pick, K=3.

Model - 1]

Applying KNN [K- nearest neighbor algorithm] to Train Our Dataset

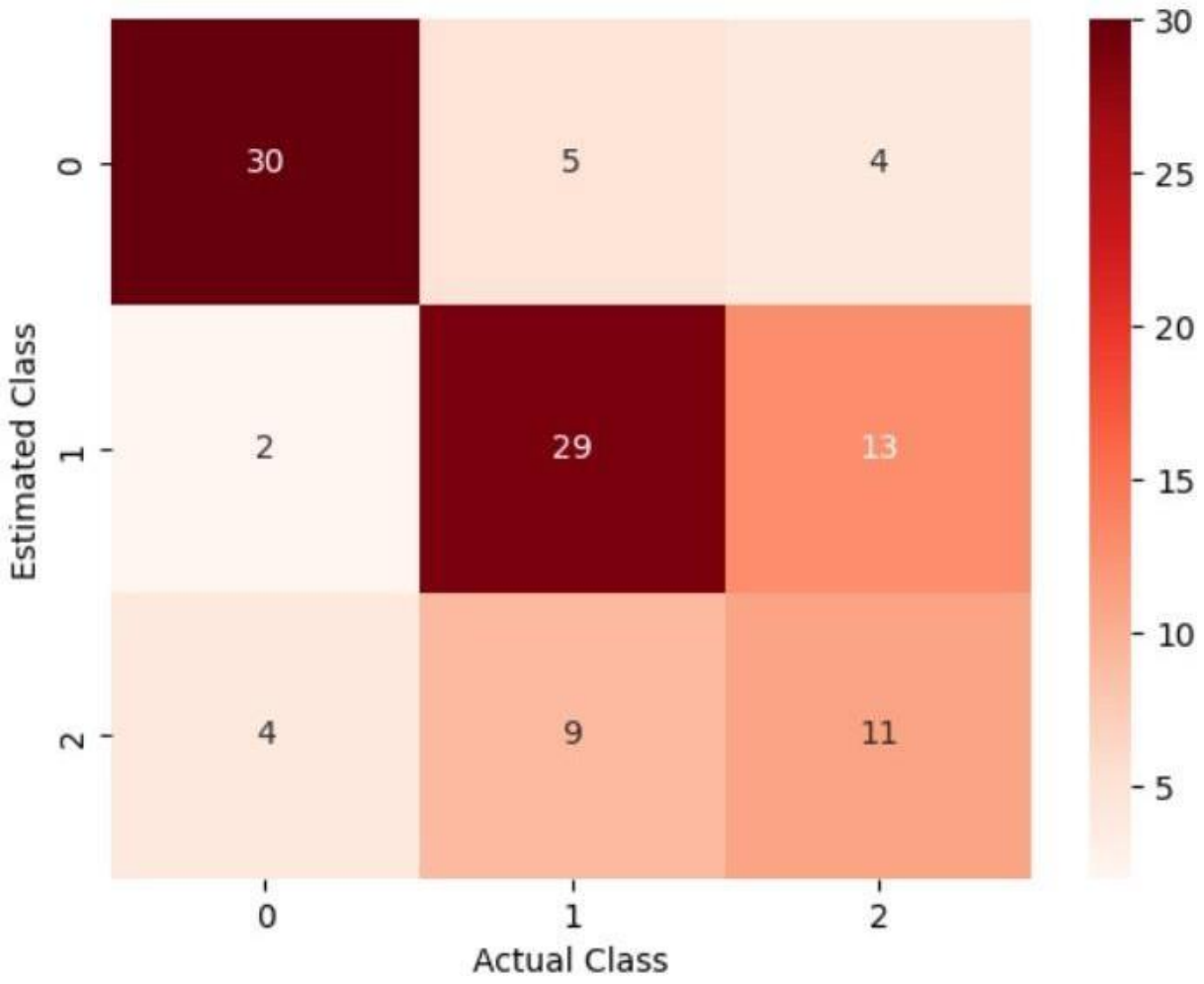
```
In [21]: model1 = KNeighborsClassifier(n_neighbors = 3)

In [22]: model1.fit(X_train, Y_train)
Out[22]: KNeighborsClassifier(n_neighbors=3)

In [23]: # Ignore Warnings
         warnings.simplefilter(action = 'ignore', category = FutureWarning)

In [24]: Y_pred = model1.predict(X_test)

In [25]: print("Accuracy is", accuracy_score(Y_test, Y_pred)*100)
         Accuracy is 65.42056074766354

In [26]: matrix = confusion_matrix(Y_test, Y_pred)
         sns.heatmap(matrix.T, annot = True, cmap = "Reds")
         plt.xlabel('Actual Class')
         plt.ylabel('Estimated Class')
         plt.show()
```

We use our training data to train our model, so it can learn from these examples.

Next we used the ***.predict()*** function on our test function and store the result in Y_pred for future references.

After getting our predicted results, the natural next step was to compare these with the actual results (Y_test) to see how well the model learned. The predictions showed that the model was 65.42% accurate.

The confusion matrix further revealed that the model correctly classified 70 (30+29+11) wines, but mislabeled 37 instances.



❖ Naive Bayes Algorithm:

This algorithm belongs to the group of "probability-based classifiers". It's based on the Bayes theorem. The classifier assumes that all attributes in our dataset are mutually independent, meaning that the presence or absence of one attribute doesn't affect the presence or absence of any other attribute. However, these assumptions can sometimes be wrong. Next, we repeated the same steps as we did for the previous model. We first used our training data to teach our model.

Model - 2]

Naive Bayes to Train Our Dataset

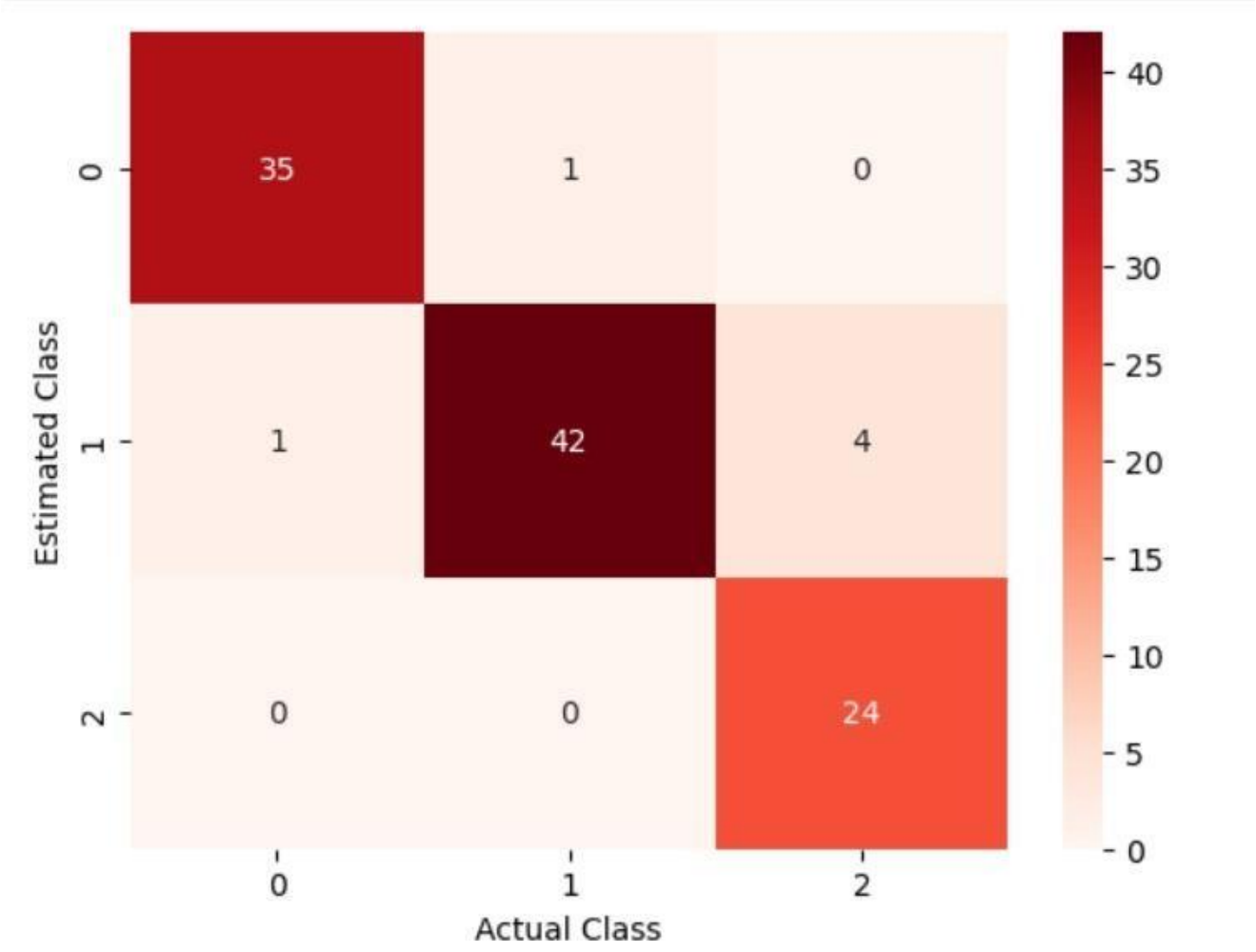
```
In [27]: model2 = GaussianNB()

In [28]: model2.fit(X_train, Y_train)
Out[28]: GaussianNB()

In [29]: Y_pred = model2.predict(X_test)

In [30]: print("Accuracy is", accuracy_score(Y_test, Y_pred)*100)
         Accuracy is 94.39252336448598

In [31]: matrix = confusion_matrix(Y_test, Y_pred)
         sns.heatmap(matrix.T, annot = True, cmap = "Reds")
         plt.xlabel('Actual Class')
         plt.ylabel('Estimated Class')
         plt.show()
```

Next, we used the ***.predict()*** function to predict the results using the model based on Naive Bayes algorithm.

Below is the Y_test that is the desired result.

Next, we examine the accuracy and the classification reports. The predictions showed an impressive accuracy of 94.39 %!!!

The confusion matrix further revealed that the model had accurately classified 101 (35+42+24) wines and had only misclassified six instances.



❖ Random Forest Classifier

Random Forest is an ensemble learning algorithm that builds multiple decision trees during training and outputs the mode of the classes for classification tasks.

It's well-suited for diverse datasets with a mix of features, handling both numerical and categorical data effectively.

The algorithm's strength lies in mitigating overfitting, providing robust predictions by aggregating the outputs of individual trees, making it versatile and suitable for datasets with complex relationships and varied characteristics.

Model - 3]

Random Forest to Train Our Dataset

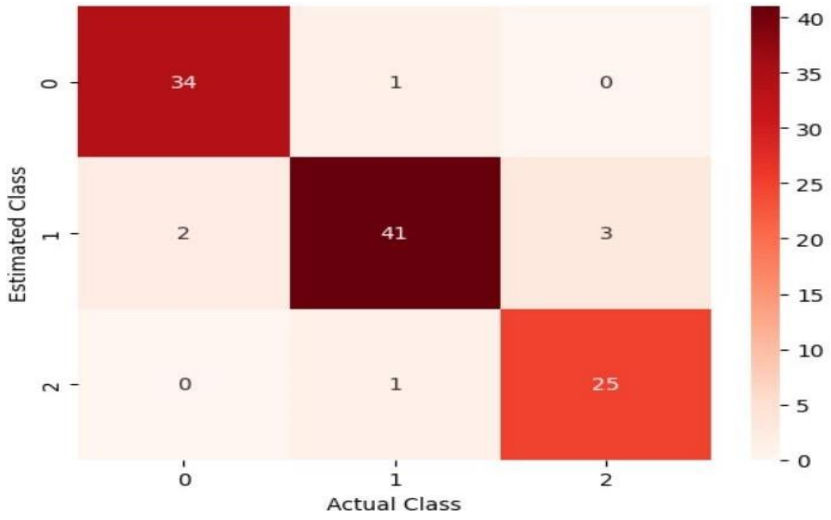
```
In [32]: random_forest = RandomForestClassifier(n_estimators = 100, random_state = 42)

In [33]: random_forest.fit(X_train, Y_train)

Out[33]: RandomForestClassifier(random_state=42)

In [34]: Y_pred = random_forest.predict(X_test)

In [35]: print("Accuracy is", accuracy_score(Y_test, Y_pred)*100)

         Accuracy is 93.45794392523365

In [36]: matrix = confusion_matrix(Y_test, Y_pred)
         sns.heatmap(matrix.T, annot = True, cmap = "Reds")
         plt.xlabel('Actual Class')
         plt.ylabel('Estimated Class')
         plt.show()
```

Next, we used the ***.predict()*** function to predict the results using the model based on Random Forest algorithm.

Below is the Y_test that is the desired result.

Next, we examine the accuracy. The predictions showed an impressive accuracy of 93.46 %!!!

The confusion matrix further revealed that the model had accurately classified 100 (34+41+25) wines and had only misclassified seven instances.



❖ Feedforward Neural Network

The neural network is a feedforward model designed for multi class classification with 11 input features.

It comprises an input layer, a hidden layer with 64 neurons using the Rectified Linear Unit (ReLU) activation function, a dropout layer with a 50% dropout rate for regularization, a second hidden layer with 32 neurons and ReLU activation, and an output layer with 3 nodes utilizing the softmax activation function.

The ReLU activation introduces non-linearity, while dropout helps prevent overfitting by randomly disabling neurons during training.

The softmax activation in the output layer converts the model's raw output into class probabilities, making it suitable for predicting one of three classes based on input data.

Model - 4]

Feedforward Aritificial Neural Network

In [38]:
```
Y = Y - 1
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.60, random_state = 42)

# Define the model
model = keras.Sequential([
    layers.Dense(64, activation='relu', input_shape=(11,)),
    layers.Dropout(0.5),
    layers.Dense(32, activation='relu'),
    layers.Dense(3, activation='softmax')
])

# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, Y_train, epochs=50, batch_size=32, validation_data=(X_test, Y_test))

# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, Y_test)
print(f"Test Accuracy: {test_accuracy}")
```

Next, we examine the accuracy. The neural network showed a low accuracy of about 35%!!!

The possible explanations for a low accuracy for a neural network could be the insufficient data or imbalanced class distribution which may affect the model's ability to generalize effectively.

Conclusions

In this study, we discovered that:

- The data we collected is related to classification. We determined this using various visualization techniques.
- We also saw the correlation between various attributes in the dataset and the contribution of each one of them in determining the response variable.
- We applied several classification methods and compared them. After evaluating the results, we concluded that the Naive Bayes algorithm is the most suitable for this dataset.

Github Repository Link

https://github.com/SwastikDalal