https://www.linkedin.com/in/erielmatthew03/

WINE CLASSIFICATION

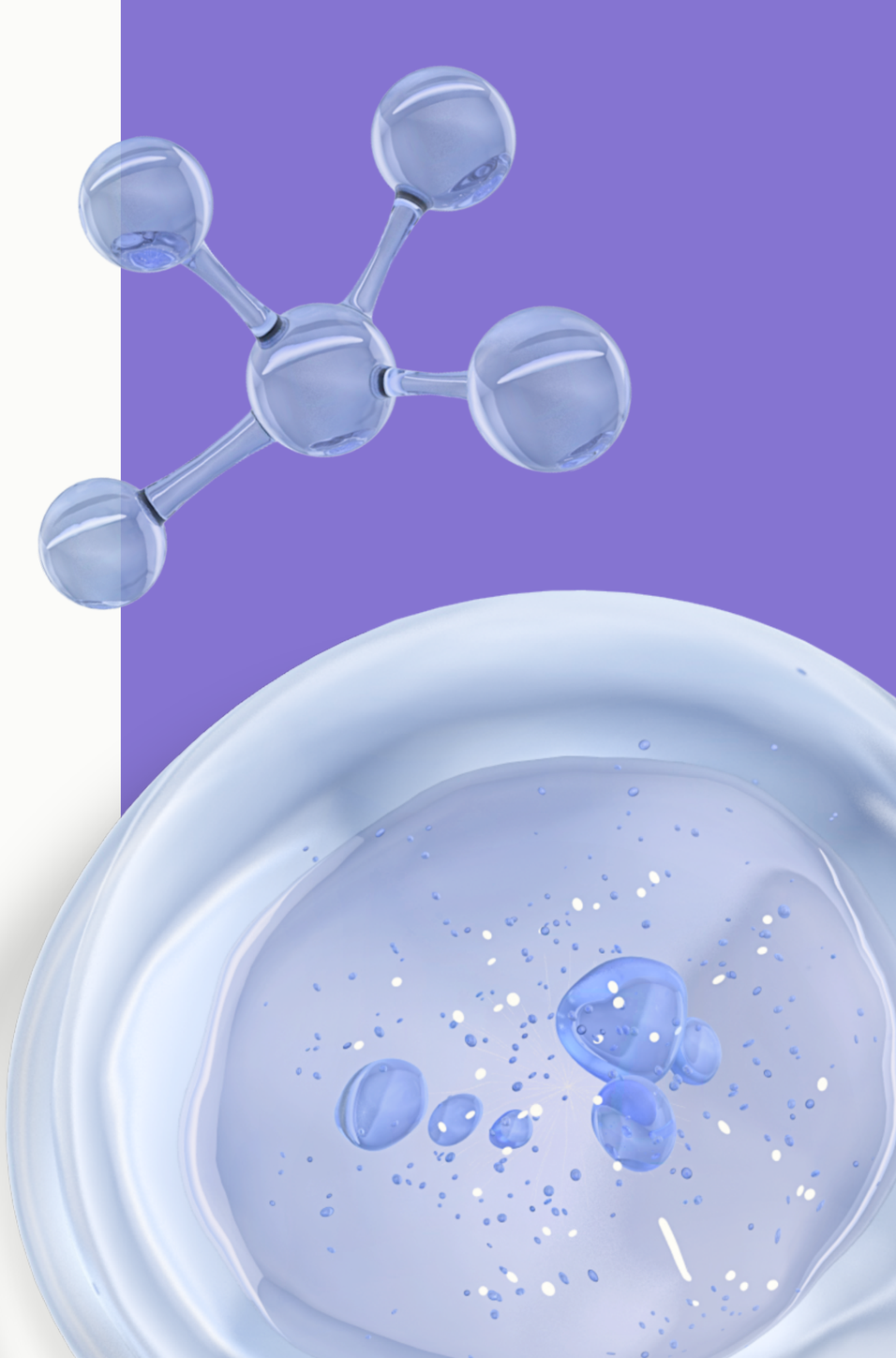
Eriel Matthew Gerald Morado

TABLE
of contents

Introduction

This dataset was downloaded from the UC Irvine Machine Learning Repository. These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

URL: https://archive.ics.uci.edu/dataset/109/wine



Introduction

The attributes are:

1. Alcohol
2. Malic Acid
3. Ash
4. Alkalinity of Ash
5. Magnesium
6. Total Phenols
7. Flavonoids
8. Nonflavonoid Phenols
9. Proanthocyanins
10. Color Intensity
11. Hue
12. OD280/OD315 of diluted wines
13. Proline

URL: https://archive.ics.uci.edu/dataset/109/wine

Setup

This step involves importing the necessary libraries for analysis.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

In this problem I use pandas for dataframe manipulation, matplotlib and seaborn for data visualization, and scikit-learn for data splitting, data preprocessing, modelling, and model evaluation.

Data Reading

In this step, we read the data that we just downloaded from the UCI Machine Learning Repository.

```
col_names = ['Wine Type', 'Alcohol', 'Malic Acid', 'Ash', 'Alkalinity of Ash', 'Magnesium', 'Total Phenols', 'Flavonoids',
             'Nonflavonoid Phenols', 'Proanthocyanins', 'Color Intensity', 'Hue', 'OD280/OD315', 'Proline']
df = pd.read_csv('wine.data', names=col_names)
df.head(10)
```

Because the original dataset doesn't contain the features' name, we have to assign the names manually. First, we make a list that contains all the features' name, then assign it into the dataframe when we use the read_csv function to read the dataset. We can assign it by utilizing the argument "names".

Data Reading

	Wine Type	Alcohol	Malic Acid	Ash	Alkalinity of Ash	Magnesium	Total Phenols	Flavonoids	Nonflavonoid Phenols	Proanthocyanins	Color Intensity	Hue	OD280/OD315	Proline
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735
5	1	14.20	1.76	2.45	15.2	112	3.27	3.39	0.34	1.97	6.75	1.05	2.85	1450
6	1	14.39	1.87	2.45	14.6	96	2.50	2.52	0.30	1.98	5.25	1.02	3.58	1290
7	1	14.06	2.15	2.61	17.6	121	2.60	2.51	0.31	1.25	5.05	1.06	3.58	1295
8	1	14.83	1.64	2.17	14.0	97	2.80	2.98	0.29	1.98	5.20	1.08	2.85	1045
9	1	13.86	1.35	2.27	16.0	98	2.98	3.15	0.22	1.85	7.22	1.01	3.55	1045

These are the first 10 rows of the dataset. The “Wine Type” column is the target variable, while the rest are the features.

Dataset Splitting

This process splits the dataset into training data and testing data. I split the data in the beginning of the analysis to ensure the testing data is unknown and there is no data leakage when we preprocess the data later.

```
x = df.drop('Wine Type', axis=1)
y = df['Wine Type']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=0)
```

Here I used the train_test_split function from sklearn.preprocessing to split the dataset. I used 75% of the data for training and the remaining 25% for testing.

Dataset Splitting

`x_train`

	Alcohol	Malic Acid	Ash	Alkalinity of Ash	Magnesium	Total Phenols	Flavonoids	Nonflavonoid Phenols	Proanthocyanins	Color Intensity	Hue	OD280/OD315	Proline
19	13.64	3.10	2.56	15.2	116	2.70	3.03	0.17	1.66	5.10	0.96	3.36	845
135	12.60	2.46	2.20	18.5	94	1.62	0.66	0.63	0.94	7.10	0.73	1.58	695
74	11.96	1.09	2.30	21.0	101	3.38	2.14	0.13	1.65	3.21	0.99	3.13	886
144	12.25	3.88	2.20	18.5	112	1.38	0.78	0.29	1.14	8.21	0.65	2.00	855
16	14.30	1.92	2.72	20.0	120	2.80	3.14	0.33	1.97	6.20	1.07	2.65	1280
...	...	...	...	...	...	...	...	...	...	...	...	...	...
103	11.82	1.72	1.88	19.5	86	2.50	1.64	0.37	1.42	2.06	0.94	2.44	415
67	12.37	1.17	1.92	19.6	78	2.11	2.00	0.27	1.04	4.68	1.12	3.48	510
117	12.42	1.61	2.19	22.5	108	2.00	2.09	0.34	1.61	2.06	1.06	2.96	345
47	13.90	1.68	2.12	16.0	101	3.10	3.39	0.21	2.14	6.10	0.91	3.33	985
172	14.16	2.51	2.48	20.0	91	1.68	0.70	0.44	1.24	9.70	0.62	1.71	660

We can see by the index that the splitting was done randomly to prevent bias.

Dataset Splitting

```
y_train
```

19	1
135	3
74	2
144	3
16	1
..	..
103	2
67	2
117	2
47	1
172	3

We can see by the index that the splitting was done randomly to prevent bias.

Exploratory Data Analysis

In this step we explore the data to uncover patterns and meaningful informations.

```
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 133 entries, 19 to 172
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Alcohol               133 non-null    float64
 1   Malic Acid            133 non-null    float64
 2   Ash                   133 non-null    float64
 3   Alkalinity of Ash     133 non-null    float64
 4   Magnesium             133 non-null    int64  
 5   Total Phenols         133 non-null    float64
 6   Flavonoids            133 non-null    float64
 7   Nonflavonoid Phenols  133 non-null    float64
 8   Proanthocyanins       133 non-null    float64
 9   Color Intensity       133 non-null    float64
 10  Hue                   133 non-null    float64
 11  OD280/OD315           133 non-null    float64
 12  Proline               133 non-null    int64  
dtypes: float64(11), int64(2)
memory usage: 14.5 KB
```

Here we check the datatype and the number of data points for each feature. There are 133 non-null data points for each feature, and all of them are already a numerical data

```
x_train.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Alcohol	133.0	12.999398	0.803811	11.03	12.37	13.05	13.64	14.75
Malic Acid	133.0	2.390977	1.122554	0.89	1.61	1.90	3.24	5.65
Ash	133.0	2.365489	0.271995	1.36	2.21	2.36	2.56	3.22
Alkalinity of Ash	133.0	19.512030	3.525320	10.60	17.10	19.50	21.50	30.00
Magnesium	133.0	100.458647	14.484846	70.00	89.00	98.00	107.00	162.00
Total Phenols	133.0	2.261579	0.613286	1.10	1.70	2.20	2.74	3.88
Flavonoids	133.0	1.956917	0.984769	0.47	1.02	2.04	2.79	3.74
Nonflavonoid Phenols	133.0	0.363985	0.127873	0.13	0.27	0.34	0.45	0.66
Proanthocyanins	133.0	1.609474	0.601056	0.42	1.15	1.56	1.96	3.58
Color Intensity	133.0	5.113083	2.405212	1.28	3.21	4.60	6.60	13.00
Hue	133.0	0.947338	0.233392	0.54	0.77	0.95	1.11	1.71
OD280/OD315	133.0	2.586015	0.725185	1.27	1.86	2.77	3.17	4.00
Proline	133.0	749.812030	303.322165	312.00	515.00	675.00	985.00	1547.00

Here we check the desriptive statistic of the dataset for each feature. We can see that Proline and Magnesium have higher means than other features.

```
y_train.value_counts()
```

```
2    50
1    43
3    40
```

Checking the count of each class in the target variable

```
x_train.duplicated().sum()
```

```
0
```

There is no duplicate value

```
x_train.isna().sum()
```

```
Alcohol                0
Malic Acid             0
Ash                    0
Alkalinity of Ash      0
Magnesium              0
Total Phenols          0
Flavonoids             0
Nonflavonoid Phenols   0
Proanthocyanins        0
Color Intensity        0
Hue                    0
OD280/OD315            0
Proline                0
```

There is no null value

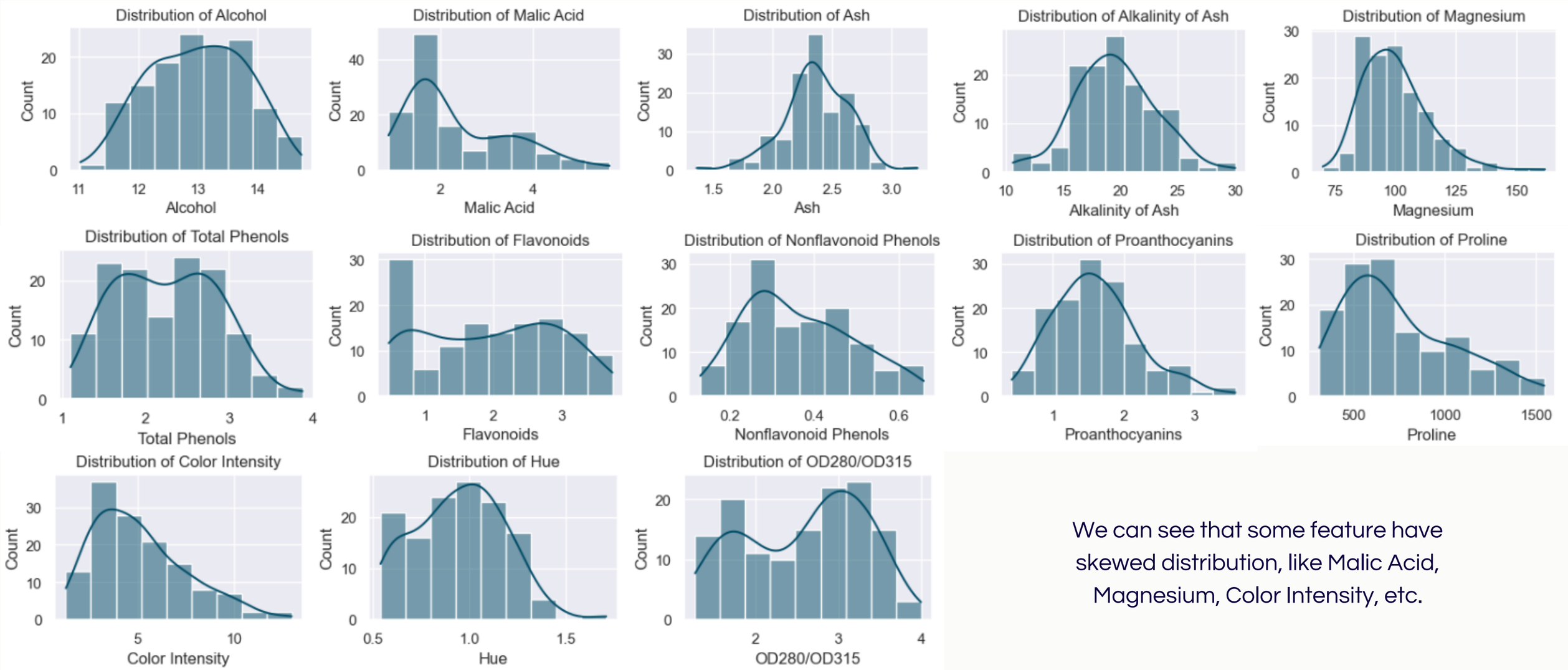
Exploratory Data Analysis

We can see how the data in each feature is distributed, by using the following code.

```
x_cols = x_train.columns
plt.figure(figsize=(10,12))
for idx in range(0, len(x_cols)):
    plt.subplot(5,3,idx+1)
    plt.title(f'Distribution of {x_cols[idx]}')
    sns.histplot(x_train, x=x_train[x_cols[idx]], kde=True, color='#004b66')
    plt.tight_layout()
    idx = idx + 1
```

Here, we are using a function from matplotlib called “subplot” that allows us to make multiple plots in one figure. In this subplot, we are using seaborn’s histplot (histogram) to visualize the distribution of each feature, which will be shown in the following slide.

Exploratory Data Analysis



We can see that some feature have skewed distribution, like Malic Acid, Magnesium, Color Intensity, etc.

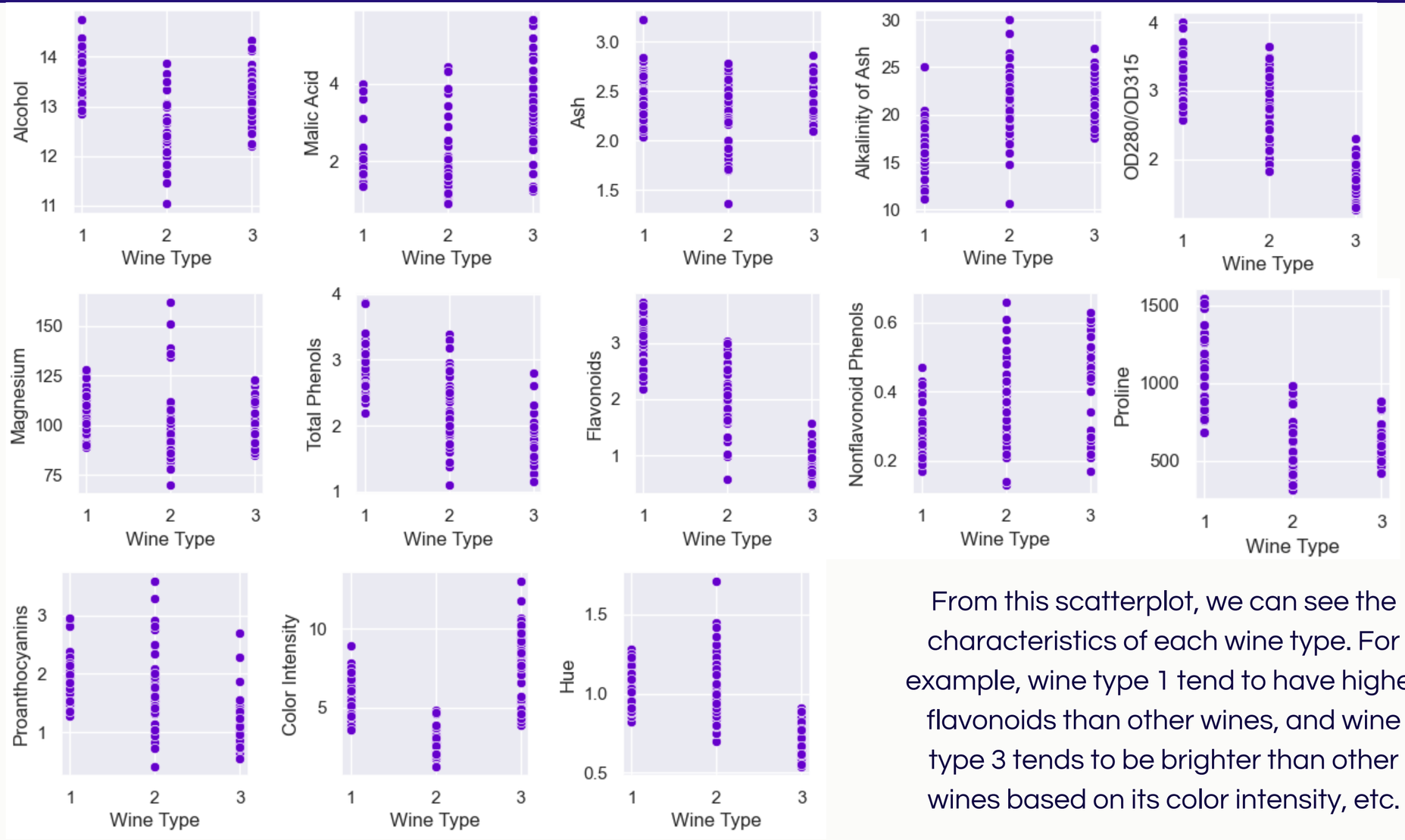
Exploratory Data Analysis

Now, we are going to see the relationship of each feature to the target variable by using scatterplot.

```
train_data = pd.concat([x_train, y_train], axis=1, ignore_index=True)
train_data.columns = ['Alcohol', 'Malic Acid', 'Ash', 'Alkalinity of Ash', 'Magnesium', 'Total Phenols', 'Flavonoids',
                      'Nonflavonoid Phenols', 'Proanthocyanins', 'Color Intensity', 'Hue', 'OD280/OD315', 'Proline', 'Wine Type']
train_data
plt.figure(figsize=(10,10))
for i in range(0, len(x_cols)):
    plt.subplot(4,4,i+1)
    sns.scatterplot(data=train_data, x=train_data['Wine Type'], y=train_data.columns[i], color='#6600cc')
    plt.tight_layout()
    i = i + 1
```

First, I concatenated the “x_train” and “y_train” data into a new dataframe “train_data” in order to be able to plot it. Again, I used subplot to make multiple plots in one figure, but I am using scatterplot this time instead of histogram. I used seaborn’s built-in scatterplot function to plot the relationship. Each feature’s relationship to the target variable will be displayed in the next slide.

Exploratory Data Analysis



From this scatterplot, we can see the characteristics of each wine type. For example, wine type 1 tend to have higher flavonoids than other wines, and wine type 3 tends to be brighter than other wines based on its color intensity, etc.

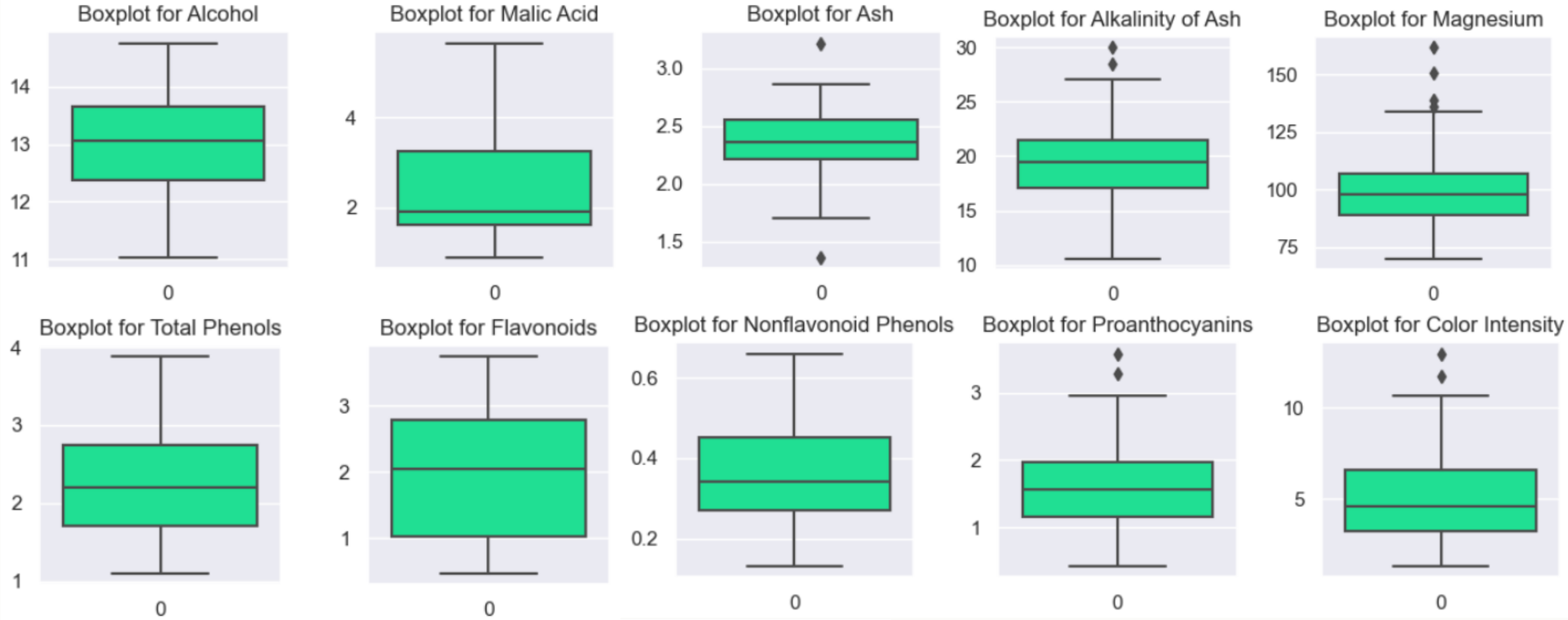
Exploratory Data Analysis

After seeing the relationship of each feature to the target variable, we are going to identify outliers in the dataset.

```
plt.figure(figsize=(10,10))
for i in range(0, len(col_names)):
    plt.subplot(4,4,i+1)
    plt.title(f'Boxplot for {col_names[i]}')
    sns.boxplot(train_data[col_names[i]], color='#00ff99')
    plt.tight_layout()
    i = i + 1
```

To identify outliers, I used boxplot visualization from seaborn's built-in function. Again, I am using subplot to make it easier to draw multiple plots.

Exploratory Data Analysis



The datapoints that fell outside of the whiskers are considered outliers, and will be removed in the next step.

Data Preprocessing

In data preprocessing, we process the data further to make it suitable for modelling. In this case, there are 2 steps involved, specifically:

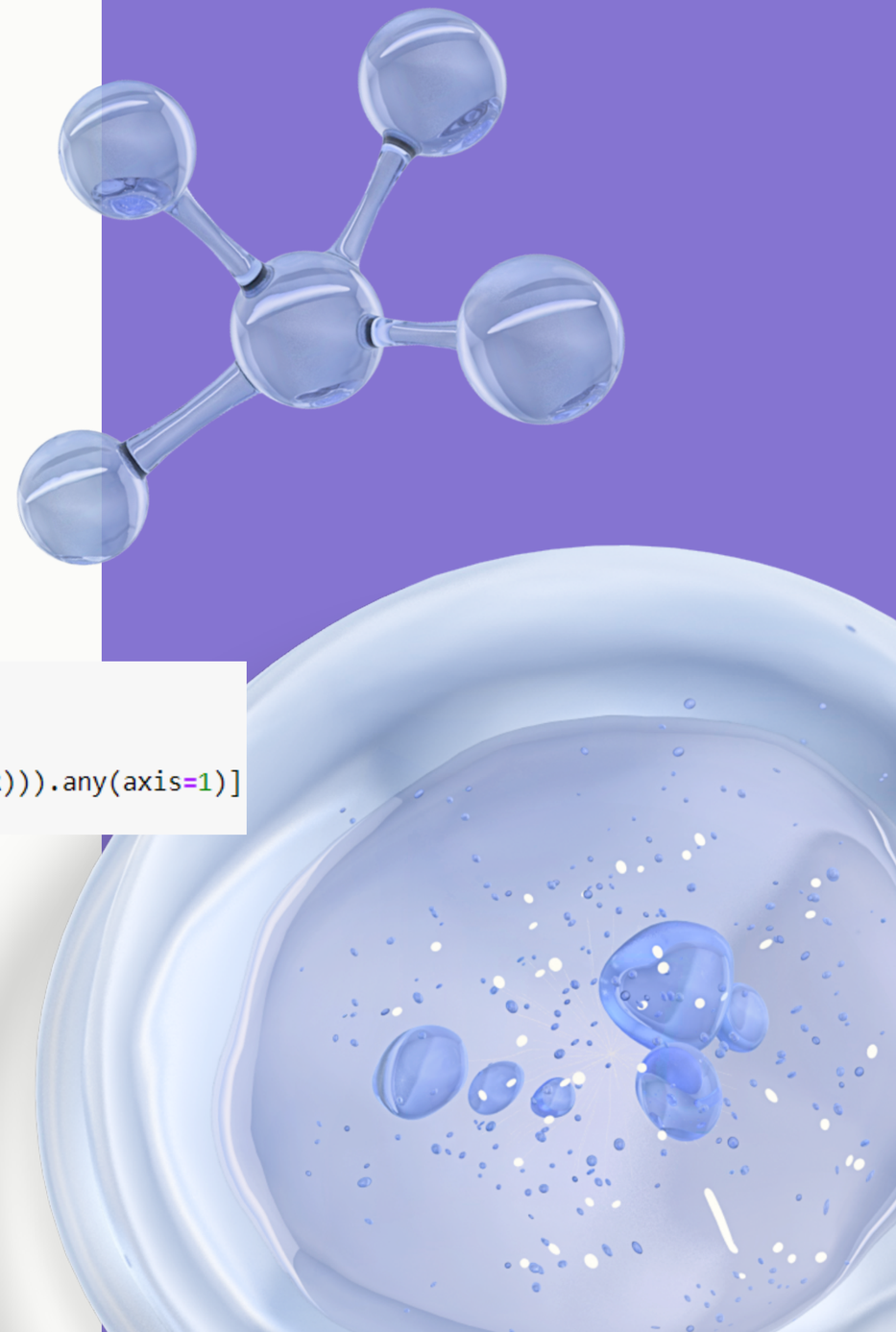
1. Outlier Removal
2. Data Scaling

Outlier Removal

We identified the outliers before using boxplot earlier. Now, we are going to remove it using the IQR (Interquartile Range) method, as follows.

```
Q1 = train_data.quantile(0.25)
Q3 = train_data.quantile(0.75)
IQR = Q3 - Q1
train_data_1 = train_data[~((train_data < (Q1 - 1.5 * IQR)) | (train_data > (Q3 + 1.5 * IQR))).any(axis=1)]
train_data_1
```

The IQR method works by defining the lower bound (Q1 - 1.5 * IQR) and the upper bound (Q3 + 1.5 * IQR). Anything that falls below the lower bound or over the upper bound will be considered an outlier and will be excluded immediately.



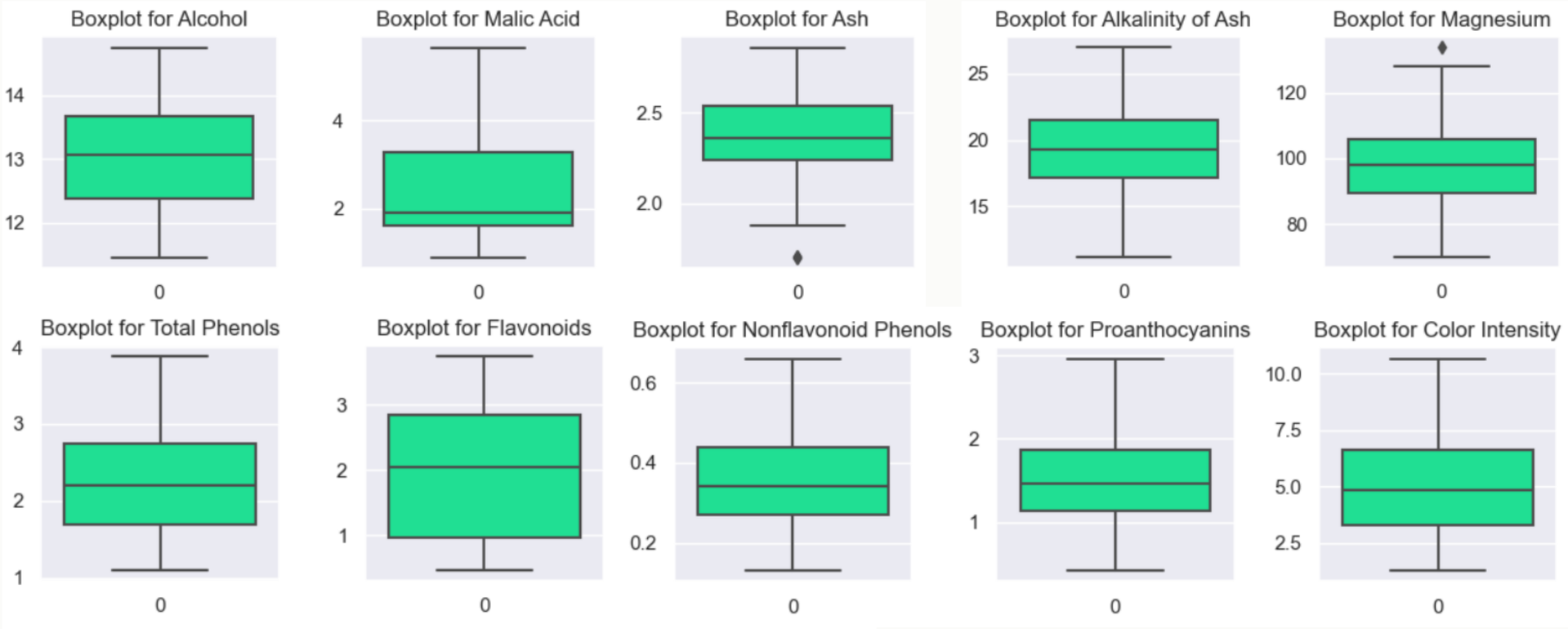
Outlier Removal

	Alcohol	Malic Acid	Ash	Alkalinity of Ash	Magnesium	Total Phenols	Flavonoids	Nonflavonoid Phenols	Proanthocyanins	Color Intensity	Hue	OD280/OD315	Proline	Wine Type
19	13.64	3.10	2.56	15.2	116	2.70	3.03	0.17	1.66	5.10	0.96	3.36	845	1
135	12.60	2.46	2.20	18.5	94	1.62	0.66	0.63	0.94	7.10	0.73	1.58	695	3
74	11.96	1.09	2.30	21.0	101	3.38	2.14	0.13	1.65	3.21	0.99	3.13	886	2
144	12.25	3.88	2.20	18.5	112	1.38	0.78	0.29	1.14	8.21	0.65	2.00	855	3
16	14.30	1.92	2.72	20.0	120	2.80	3.14	0.33	1.97	6.20	1.07	2.65	1280	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
103	11.82	1.72	1.88	19.5	86	2.50	1.64	0.37	1.42	2.06	0.94	2.44	415	2
67	12.37	1.17	1.92	19.6	78	2.11	2.00	0.27	1.04	4.68	1.12	3.48	510	2
117	12.42	1.61	2.19	22.5	108	2.00	2.09	0.34	1.61	2.06	1.06	2.96	345	2
47	13.90	1.68	2.12	16.0	101	3.10	3.39	0.21	2.14	6.10	0.91	3.33	985	1
172	14.16	2.51	2.48	20.0	91	1.68	0.70	0.44	1.24	9.70	0.62	1.71	660	3

122 rows × 14 columns

After removing the outliers, we can see that only 122 data points were left.

Outlier Removal



We can see that there are still some outliers, like in Ash and Magnesium, so we are going to re-iterate the outlier removal process one more time.

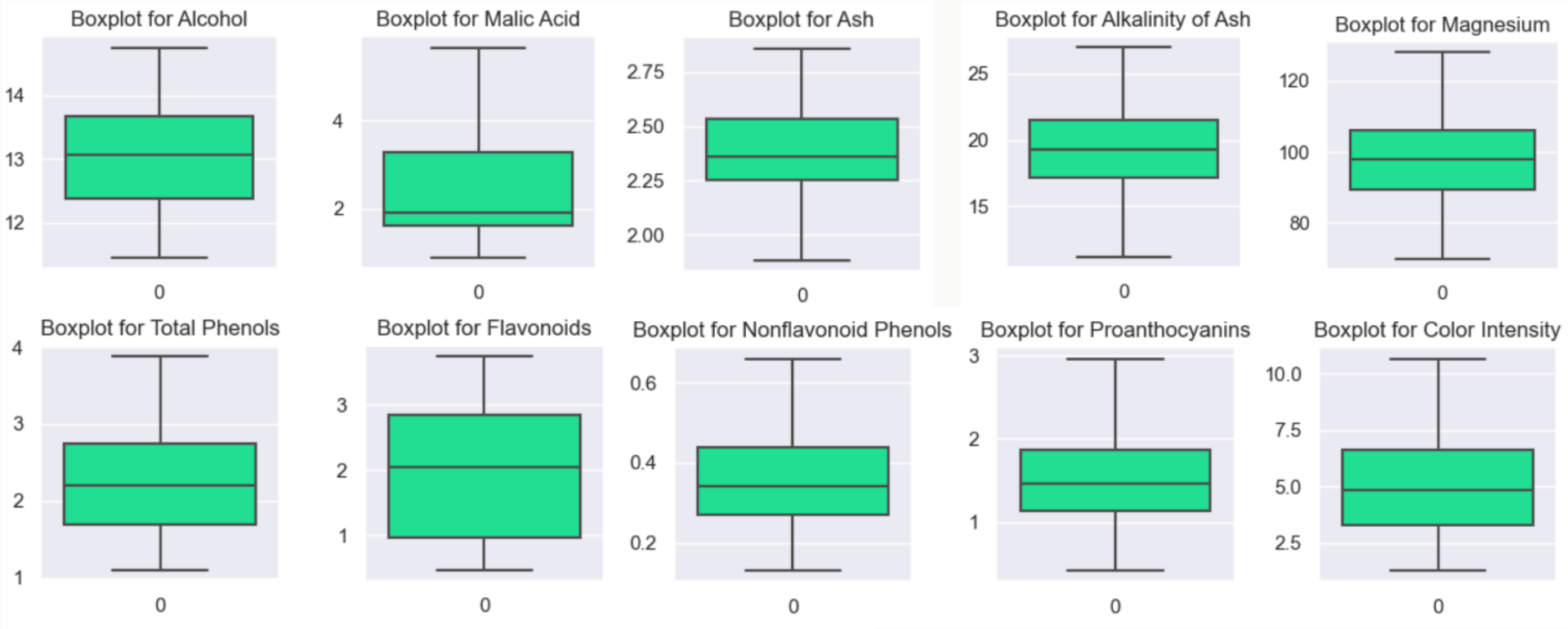
Outlier Removal

	Alcohol	Malic Acid	Ash	Alkalinity of Ash	Magnesium	Total Phenols	Flavonoids	Nonflavonoid Phenols	Proanthocyanins	Color Intensity	Hue	OD280/OD315	Proline	Wine Type
19	13.64	3.10	2.56	15.2	116	2.70	3.03	0.17	1.66	5.10	0.96	3.36	845	1
135	12.60	2.46	2.20	18.5	94	1.62	0.66	0.63	0.94	7.10	0.73	1.58	695	3
74	11.96	1.09	2.30	21.0	101	3.38	2.14	0.13	1.65	3.21	0.99	3.13	886	2
144	12.25	3.88	2.20	18.5	112	1.38	0.78	0.29	1.14	8.21	0.65	2.00	855	3
16	14.30	1.92	2.72	20.0	120	2.80	3.14	0.33	1.97	6.20	1.07	2.65	1280	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
103	11.82	1.72	1.88	19.5	86	2.50	1.64	0.37	1.42	2.06	0.94	2.44	415	2
67	12.37	1.17	1.92	19.6	78	2.11	2.00	0.27	1.04	4.68	1.12	3.48	510	2
117	12.42	1.61	2.19	22.5	108	2.00	2.09	0.34	1.61	2.06	1.06	2.96	345	2
47	13.90	1.68	2.12	16.0	101	3.10	3.39	0.21	2.14	6.10	0.91	3.33	985	1
172	14.16	2.51	2.48	20.0	91	1.68	0.70	0.44	1.24	9.70	0.62	1.71	660	3

119 rows × 14 columns

This process was done using the same exact method like the previous one, and 2 data points were removed, leaving 119 data points in the training dataset.

Outlier Removal



After repeating the process one more time, we can see that there are no more outliers in the dataset.

Data Scaling

Data scaling is necessary to ensure all of the values are on the same scale so that modelling can be done optimally. In this case, we are using the MinMax scaling method from the scikit-learn library.

```
scaler = MinMaxScaler()
scaler.fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_train_scaled = pd.DataFrame(x_train_scaled, columns=x_cols)
x_train_scaled
```

Here we assign the MinMaxScaler to the variable “scaler” and fit it into the training data. (fitting the scaler should only be done on the training data, not on the testing data, to prevent data leakage.) After fitting, the training data will be transformed.

Data Scaling

	Alcohol	Malic Acid	Ash	Alkalinity of Ash	Magnesium	Total Phenols	Flavonoids	Nonflavonoid Phenols	Proanthocyanins	Color Intensity	Hue	OD280/OD315	Proline
0	0.663636	0.464286	0.693878	0.253165	0.793103	0.575540	0.782875	0.075472	0.488189	0.406383	0.461538	0.765568	0.431579
1	0.348485	0.329832	0.326531	0.462025	0.413793	0.187050	0.058104	0.943396	0.204724	0.619149	0.208791	0.113553	0.310121
2	0.154545	0.042017	0.428571	0.620253	0.534483	0.820144	0.510703	0.000000	0.484252	0.205319	0.494505	0.681319	0.464777
3	0.242424	0.628151	0.326531	0.462025	0.724138	0.100719	0.094801	0.301887	0.283465	0.737234	0.120879	0.267399	0.439676
4	0.863636	0.216387	0.857143	0.556962	0.862069	0.611511	0.816514	0.377358	0.610236	0.523404	0.582418	0.505495	0.783806
...	...	...	...	...	...	...	...	...	...	...	...	...	...
114	0.112121	0.174370	0.000000	0.525316	0.275862	0.503597	0.357798	0.452830	0.393701	0.082979	0.439560	0.428571	0.083401
115	0.278788	0.058824	0.040816	0.531646	0.137931	0.363309	0.467890	0.264151	0.244094	0.361702	0.637363	0.809524	0.160324
116	0.293939	0.151261	0.316327	0.715190	0.655172	0.323741	0.495413	0.396226	0.468504	0.082979	0.571429	0.619048	0.026721
117	0.742424	0.165966	0.244898	0.303797	0.534483	0.719424	0.892966	0.150943	0.677165	0.512766	0.406593	0.754579	0.544939
118	0.821212	0.340336	0.612245	0.556962	0.362069	0.208633	0.070336	0.584906	0.322835	0.895745	0.087912	0.161172	0.281781

Now the training data are already on the same scale, ranging from 0 to 1. This data is ready for model training.

Model Training

Model training means fitting the model(s) to the training data to make predictions based on the data in the futue. In this step, I did hyperparameter tuning and cross validation simultaneously using GridSearchCV from scikit-learn. For this classification problem, I am using the K-Nearest Neighbors algorithm because it is simple and the dataset is small.

```
knn = KNeighborsClassifier()

# We store the hyperparameters we want to tune in the dictionary 'param_grid'
param_grid = {
    'n_neighbors' : range(1,10),
    'weights' : ['uniform', 'distance'],
    'p' : [1, 2]
}
grid_search = GridSearchCV(knn, param_grid, cv=3)
grid_search.fit(x_train_scaled, y_train)
```

In this step, we assign the classifier to the variable “knn”, and we define the hyperparameters of the KNN classifier that we want to tune and store it in the dictionary “param_grid”. Then, we use GridSearchCV to search for the best hyperparameter for the model by fitting it to the scaled training data, “x_train_scaled” and “y_train”.

Model Training

To see the best hyperparameter for the model, we use the function “best_params_” from the GridSearchCV, as follows.

```
print(f'Best Hyperparameters: {grid_search.best_params_}')
```

```
Best Hyperparameters: {'n_neighbors': 2, 'p': 1, 'weights': 'uniform'}
```

To use the best model, we can use the function “best_estimator_” from the GridSearchCV, and assign it into a variable.

```
best_knn = grid_search.best_estimator_
```

In this step, the model training is done succesfully.

Model Testing

Before we predict the data using the model, we have to scale the testing data to ensure the model's performance is consistent. The previous scaler will be used to transform, NOT fit, the testing data.

```
x_test_scaled = pd.DataFrame(scaler.transform(x_test), columns=x_cols)
x_test_scaled.head(10)
```

	Alcohol	Malic Acid	Ash	Alkalinity of Ash	Magnesium	Total Phenols	Flavonoids	Nonflavonoid Phenols	Proanthocyanins	Color Intensity	Hue	OD280/OD315	Proline
0	0.693939	0.163866	0.377551	0.329114	0.827586	0.539568	0.743119	0.150943	0.472441	0.486170	0.417582	0.706960	0.605668
1	0.406061	0.373950	0.612245	0.683544	0.724138	0.136691	0.272171	0.207547	0.330709	1.012766	-0.065934	0.073260	0.136032
2	0.278788	0.050420	0.285714	0.493671	0.293103	0.863309	0.804281	0.113208	0.570866	0.337234	0.747253	0.586081	0.087449
3	0.639394	0.176471	0.591837	0.588608	0.793103	0.669065	0.706422	0.132075	0.799213	0.528723	0.483516	0.644689	0.654251
4	0.484848	1.031513	0.255102	0.651899	0.275862	0.546763	0.666667	0.320755	0.625984	0.140426	0.208791	0.670330	0.055061
5	0.033333	0.243697	1.377551	1.094937	0.844828	0.748201	1.409786	0.641509	0.570866	0.502128	0.428571	0.886447	0.123887
6	0.790909	0.264706	0.744898	0.405063	0.879310	0.539568	0.623853	0.339623	0.326772	0.401064	0.571429	0.846154	0.795951
7	0.275758	0.617647	0.510204	0.620253	0.310345	0.431655	0.137615	0.698113	0.244094	0.677660	0.021978	0.113553	0.168421
8	0.242424	0.176471	0.244898	0.493671	0.172414	0.197842	0.477064	0.452830	0.476378	0.225532	0.505495	0.695971	0.160324
9	0.190909	0.197479	0.448980	0.462025	0.189655	0.179856	0.314985	0.735849	0.480315	0.119149	0.593407	0.366300	0.136032

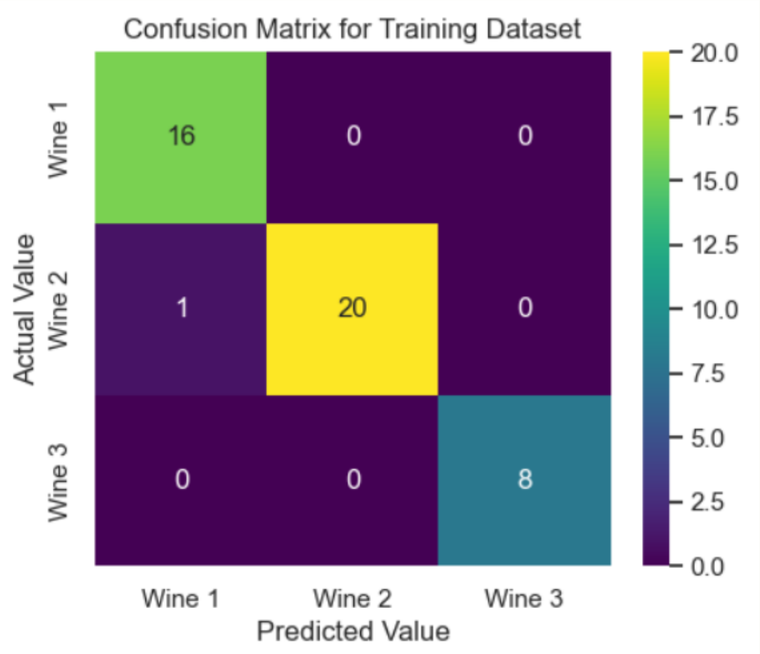
After scaling, we will use the model to predict the testing data, as follows.

```
y_test_pred = best_knn.predict(x_test_scaled)
```

Model Evaluation

- **Confusion Matrix**

```
test_matrix = confusion_matrix(y_test, y_test_pred)
plt.figure(figsize=(5,4))
plt.title('Confusion Matrix for Training Dataset')
sns.heatmap(test_matrix, annot=True, cmap='viridis',
            xticklabels=['Wine 1', 'Wine 2', 'Wine 3'], yticklabels=['Wine 1', 'Wine 2', 'Wine 3'])
plt.xlabel('Predicted Value')
plt.ylabel('Actual Value')
plt.show()
```



From the confusion matrix, we can see that the model only classified 1 data point incorrecly out of 45 data points.

Model Evaluation

- **Accuracy**

```
test_accu = accuracy_score(y_test, y_test_pred)
print(f'Accuracy Score for Testing Dataset: {test_accu:.2%}')
```

```
Accuracy Score for Testing Dataset: 97.78%
```

The accuracy score on the testing data is 97.78%, which is considered very high.

- **Classification Report**

```
test_report = classification_report(y_test, y_test_pred)
print(test_report)
```

	precision	recall	f1-score	support
1	0.94	1.00	0.97	16
2	1.00	0.95	0.98	21
3	1.00	1.00	1.00	8
accuracy			0.98	45
macro avg	0.98	0.98	0.98	45
weighted avg	0.98	0.98	0.98	45

We can see the overall report of the classification we just did that contains other metrics, like precision, recall, and f1-score.

Conclusions

From my analysis, each wine type has certain characteristics regarding its constituents. For example, wine type 1 has higher alcohol level than other wine, or wine type 2 has lower color intensity which might make the color kind of muted.

We can also see that the model that I developed is considered very accurate in classifying which type a wine belongs to based on its constituents, obtaining and accuracy score of 97.78%

THANK YOU!

https://www.linkedin.com/in/erielmatthew03/

erielmatthew03@gmail.com