Notebook

April 18, 2021

[1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import sklearn
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import sklearn.metrics
from sklearn.linear_model import LinearRegression
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
```

[2]:

```
breast = pd.read_csv('/Users/jeandre/DataScience874/PostBlockAssignment2/
↪breast_cancer_dataset.csv')
```

[3]:

```
breast.head()
```

[3]:

```
         id diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0    842302         M        17.99         10.38          122.80     1001.0
1    842517         M        20.57         17.77          132.90     1326.0
2  84300903         M        19.69         21.25          130.00     1203.0
3  84348301         M        11.42         20.38           77.58      386.1
4  84358402         M        20.29         14.34          135.10     1297.0

   smoothness_mean  compactness_mean  concavity_mean  concave points_mean  \
0          0.11840           0.27760          0.3001              0.14710
1          0.08474           0.07864          0.0869              0.07017
2          0.10960           0.15990          0.1974              0.12790
3          0.14250           0.28390          0.2414              0.10520
4          0.10030           0.13280          0.1980              0.10430

   …  texture_worst  perimeter_worst  area_worst  smoothness_worst  \
0  …          17.33           184.60      2019.0            0.1622
1  …          23.41           158.80      1956.0            0.1238
2  …          25.53           152.50      1709.0            0.1444
3  …          26.50            98.87       567.7            0.2098
4  …          16.67           152.20      1575.0            0.1374
```

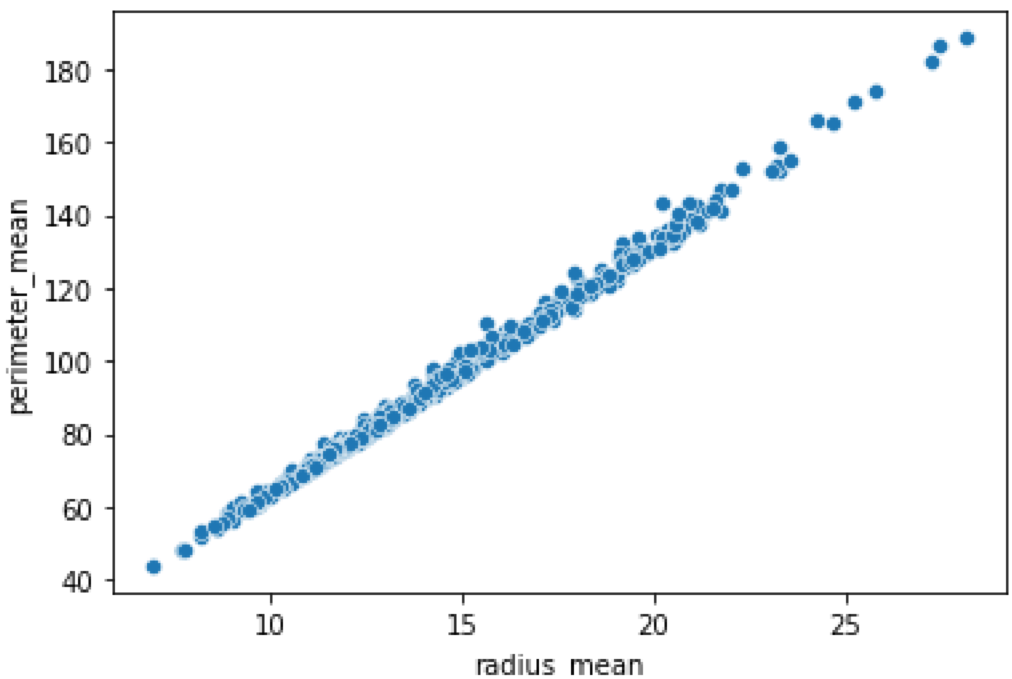
```
   compactness_worst  concavity_worst  concave points_worst  symmetry_worst  \
0             0.6656           0.7119                0.2654          0.4601
1             0.1866           0.2416                0.1860          0.2750
2             0.4245           0.4504                0.2430          0.3613
3             0.8663           0.6869                0.2575          0.6638
4             0.2050           0.4000                0.1625          0.2364

   fractal_dimension_worst  Unnamed: 32
0                  0.11890          NaN
1                  0.08902          NaN
2                  0.08758          NaN
3                  0.17300          NaN
4                  0.07678          NaN

[5 rows x 33 columns]
```

1 Question 1

1.1 Scatter plot showing radius_mean and perimeter_mean

[4]:
```
sns.scatterplot(data=breast, x="radius_mean", y="perimeter_mean")
```

[4]:
```
<AxesSubplot:xlabel='radius_mean', ylabel='perimeter_mean'>
```



1.2 There is a strong positive correleation between radius_mean and perimeter_mean. There is almost no deviation to a straight line relationship between the two.

1.3 the correlation value for the radius and mean below:

[5]:
```
breast['radius_mean'].corr(breast['perimeter_mean'])
```

[5]:
```
0.9978552814938109
```

A value of 0.997 is very close to 1 and therefore confirms the previous observation made from the graph that the two features have a strong postive correlation.

1.4 Training KNN using 70% training set and values of k = [3, 7, 15, 31, 61]

[6]:
```
X = breast[list(breast.columns[2:32])]
Y = breast["diagnosis"]

values = {'B':1, 'M':0}
Y = Y.map(values)
```

[7]:
```
X.head()
```

[7]:
```
   radius_mean  texture_mean  perimeter_mean  area_mean  smoothness_mean  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   compactness_mean  concavity_mean  concave points_mean  symmetry_mean  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   fractal_dimension_mean  …  radius_worst  texture_worst  perimeter_worst  \
0                 0.07871  …         25.38          17.33           184.60
1                 0.05667  …         24.99          23.41           158.80
2                 0.05999  …         23.57          25.53           152.50
3                 0.09744  …         14.91          26.50            98.87
4                 0.05883  …         22.54          16.67           152.20

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      2019.0            0.1622             0.6656           0.7119
1      1956.0            0.1238             0.1866           0.2416
2      1709.0            0.1444             0.4245           0.4504
```

```
3      567.7            0.2098            0.8663           0.6869
4     1575.0            0.1374            0.2050           0.4000

   concave points_worst  symmetry_worst  fractal_dimension_worst
0                0.2654          0.4601                  0.11890
1                0.1860          0.2750                  0.08902
2                0.2430          0.3613                  0.08758
3                0.2575          0.6638                  0.17300
4                0.1625          0.2364                  0.07678

[5 rows x 30 columns]
```

[8]:
```
k_list = [3,7,15,31,61]
```

[9]:
```
X_train, X_test, Y_train, Y_test = sklearn.model_selection.
 ↪train_test_split(X,Y, test_size = 0.3, random_state = 0)
```

[10]:
```
for k in k_list:
    neigh = KNeighborsClassifier(n_neighbors=k)
    neigh.fit(X_train,Y_train)
    print("Accuracy for k =", k)
    print(neigh.score(X_test,Y_test))
```

```
Accuracy for k = 3
0.9181286549707602
Accuracy for k = 7
0.9532163742690059
Accuracy for k = 15
0.9649122807017544
Accuracy for k = 31
0.9532163742690059
Accuracy for k = 61
0.9298245614035088
```

2 Question 2

[11]:
```
wine = pd.read_csv('/Users/jeandre/DataScience874/PostBlockAssignment2/
 ↪wine-composition-dataset.csv')
```

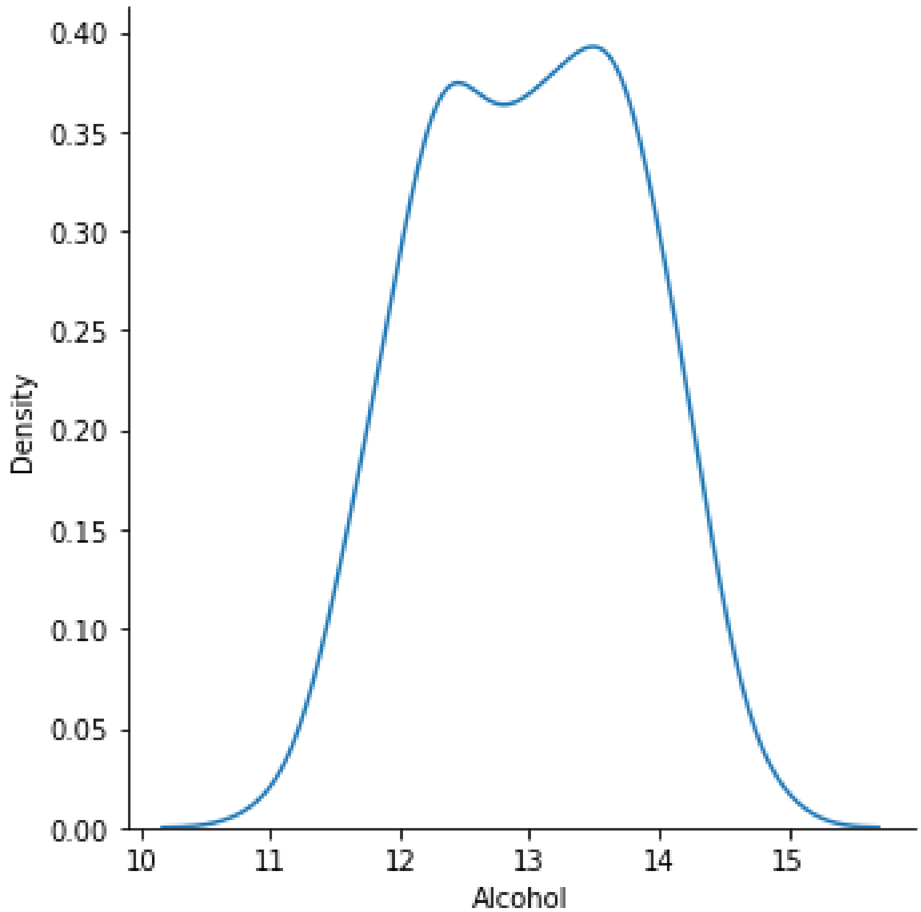
[12]:
```
wine.head()
```

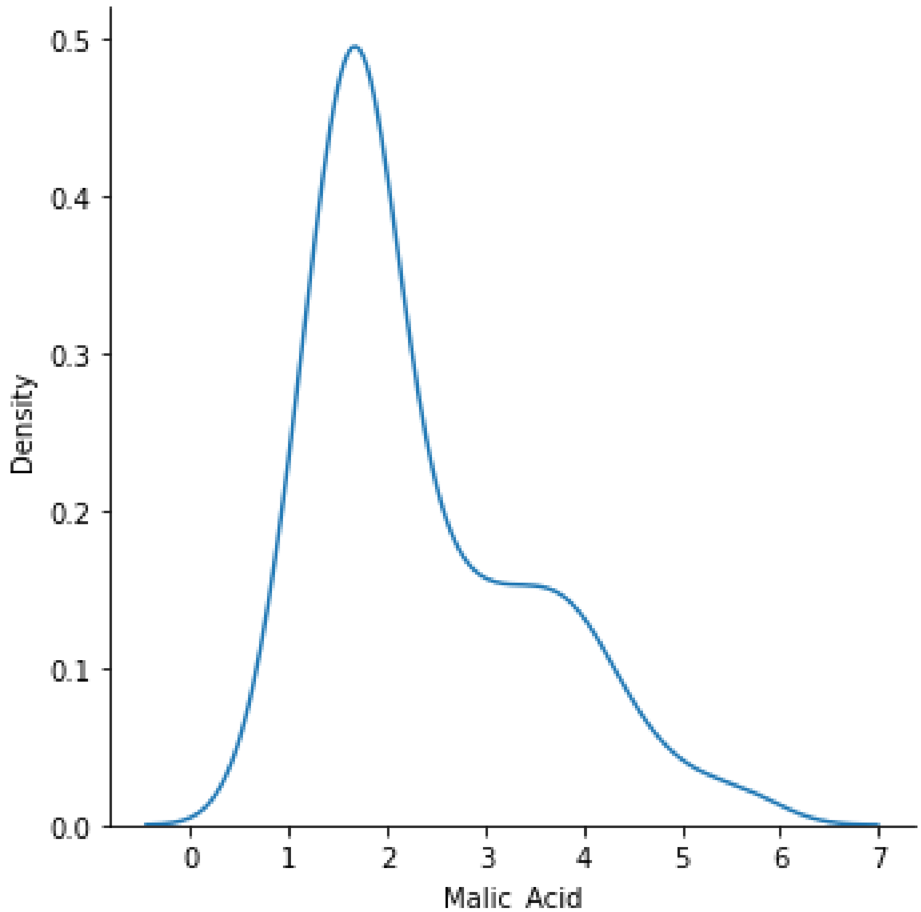
[12]:
```
   Alcohol  Malic_Acid   Ash  Ash_Alcanity  Magnesium  Total_Phenols  \
0    14.23        1.71  2.43          15.6        127           2.80
1    13.20        1.78  2.14          11.2        100           2.65
2    13.16        2.36  2.67          18.6        101           2.80
```

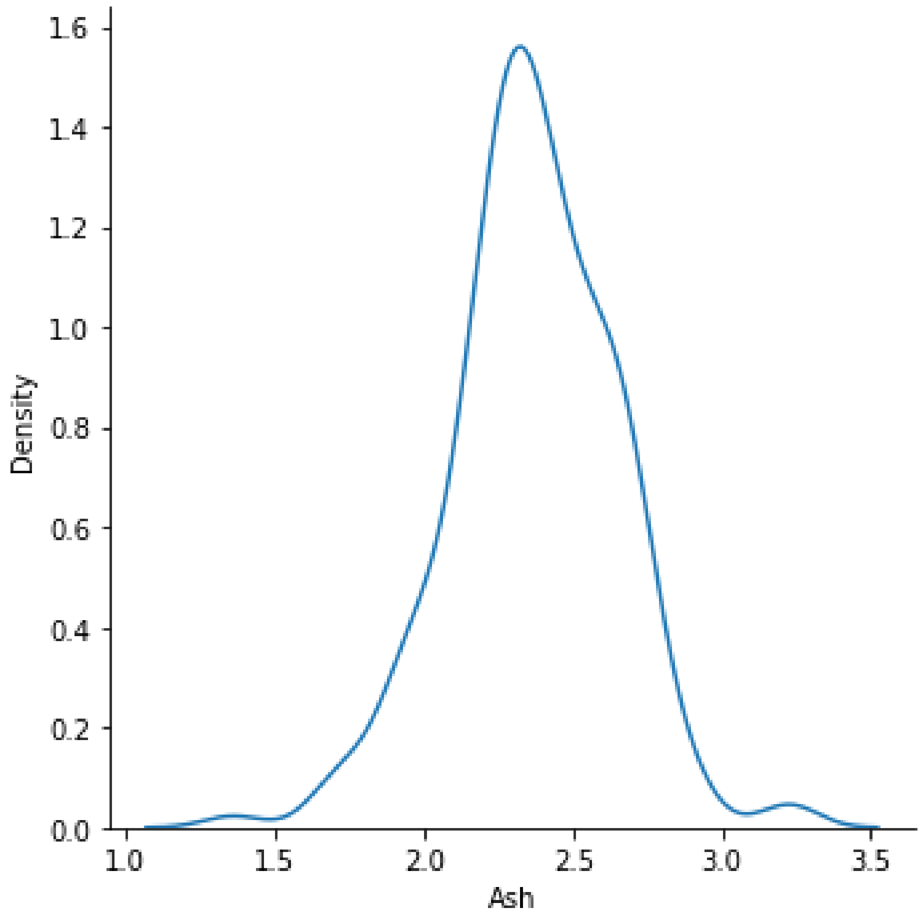
```
3    14.37         1.95  2.50           16.8        113             3.85
4    13.24         2.59  2.87           21.0        118             2.80

   Flavanoids  Nonflavanoid_Phenols  Proanthocyanins  Color_Intensity   Hue  \
0        3.06                  0.28             2.29             5.64  1.04
1        2.76                  0.26             1.28             4.38  1.05
2        3.24                  0.30             2.81             5.68  1.03
3        3.49                  0.24             2.18             7.80  0.86
4        2.69                  0.39             1.82             4.32  1.04

   OD280  Proline
0   3.92     1065
1   3.40     1050
2   3.17     1185
3   3.45     1480
4   2.93      735
```

2.1

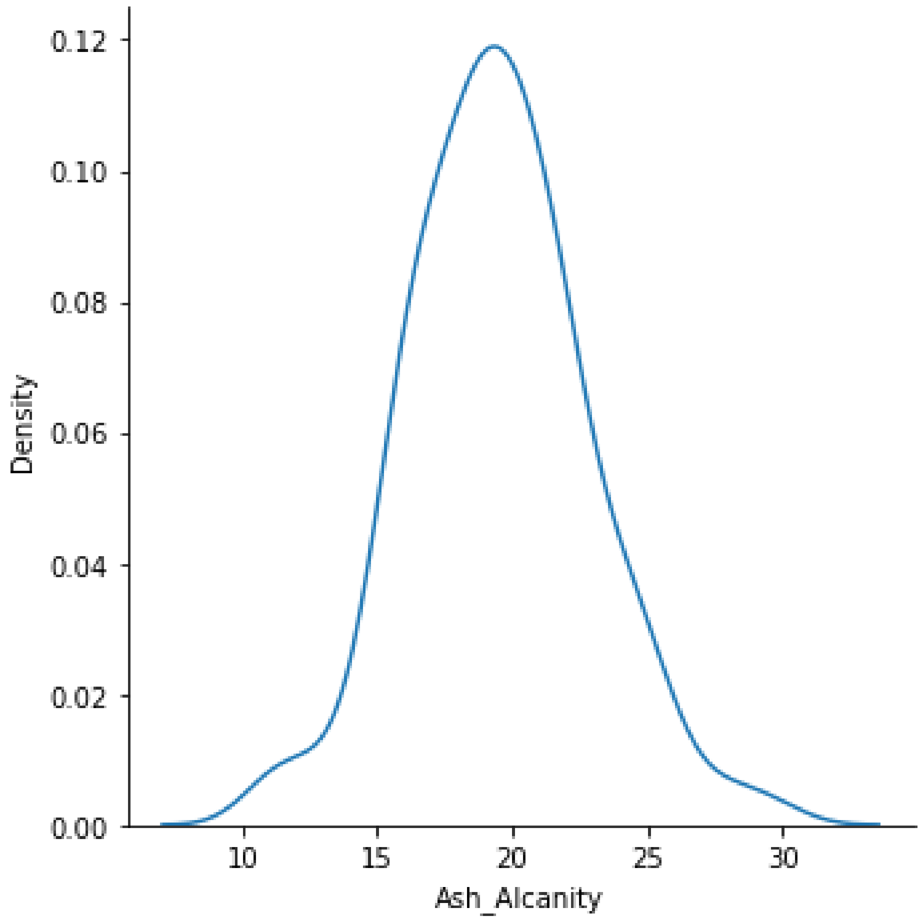
[13]:
```
columns =␣
↪["Alcohol","Malic_Acid","Ash","Ash_Alcanity","Magnesium","Total_Phenols","Flavanoids","Nonf
           "Proanthocyanins","Color_Intensity","Hue","OD280","Proline"]
for column in columns:
    sns.displot(data = wine, x = column, kind = "kde" )
```

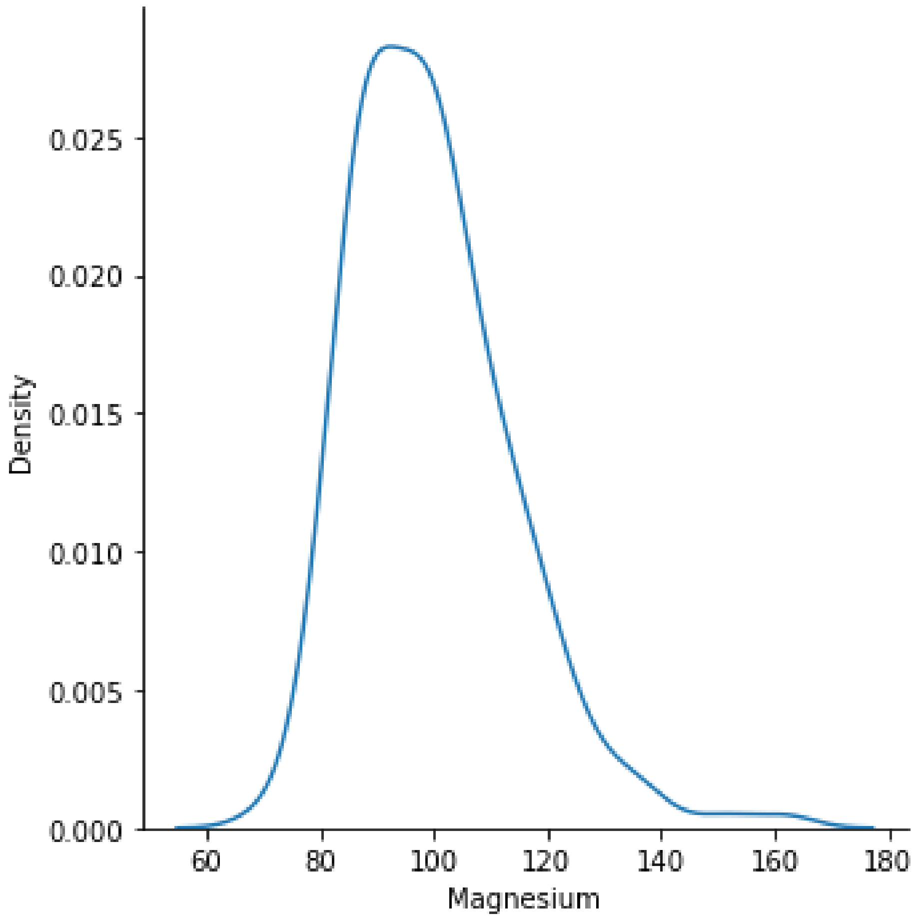

0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
10
11
12
13
14
15
Alcohol

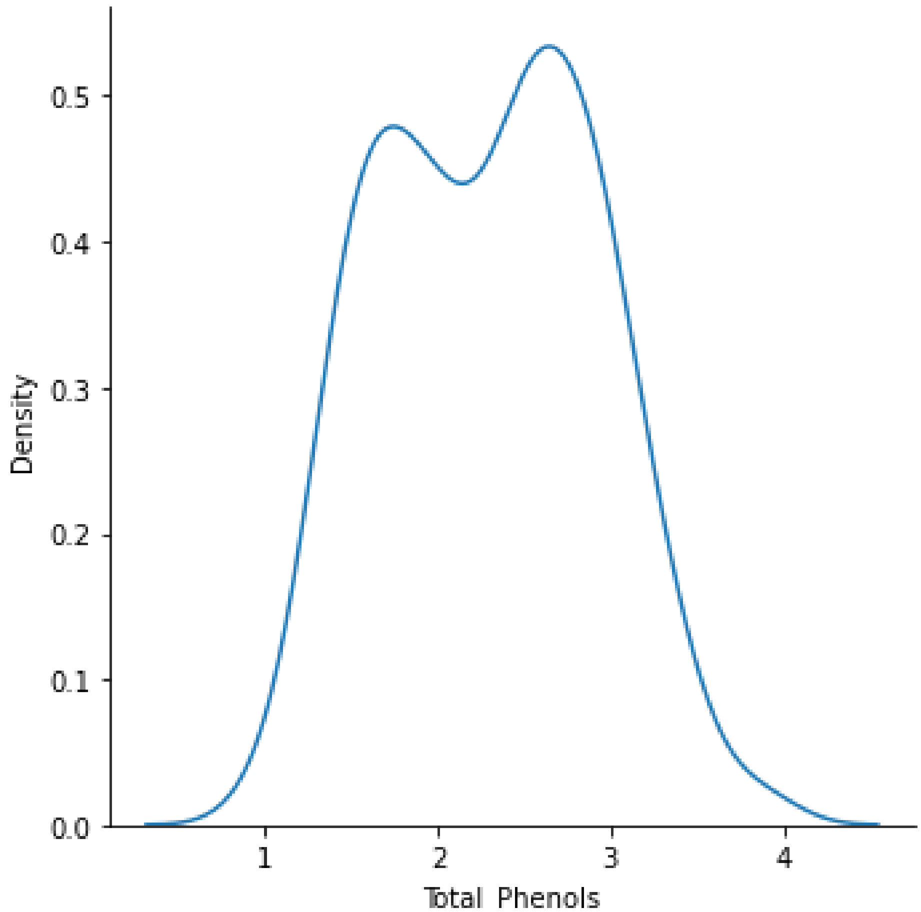

0.5
0.4
0.3
0.2
0.1
0.0
Density
0
1
2
3
4
5
6
7
Malic_Acid

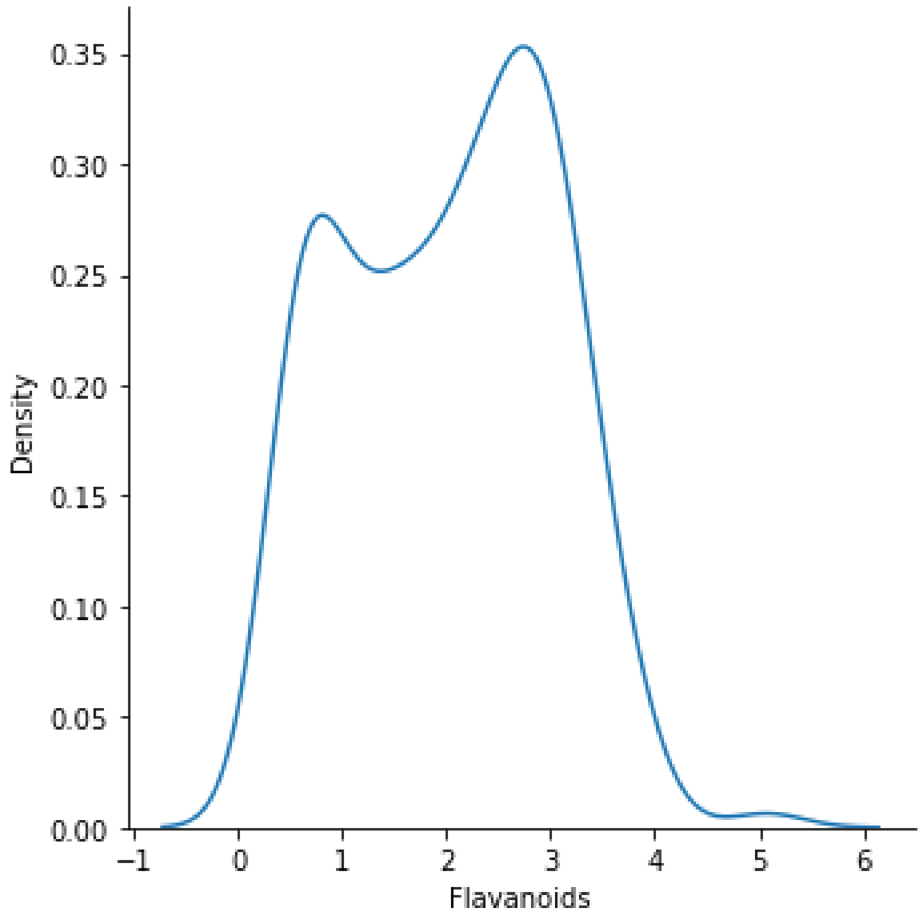


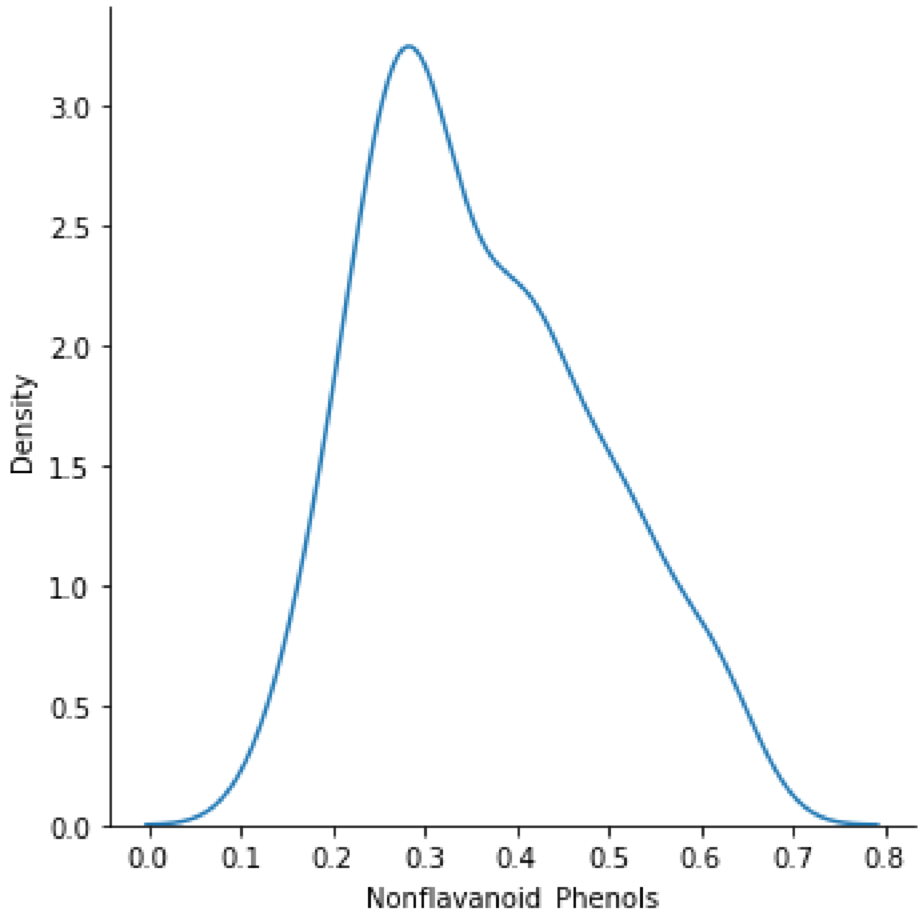
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Density
1.0
1.5
2.0
2.5
3.0
3.5
Ash

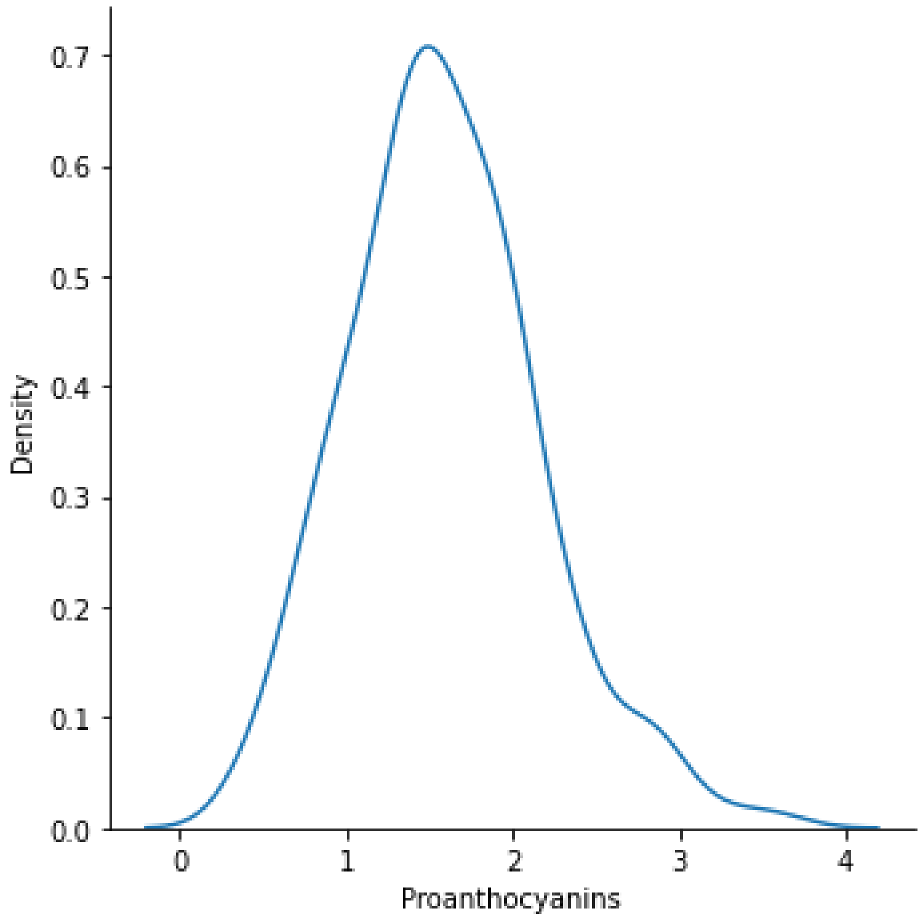

0.12
0.10
0.08
0.06
0.04
0.02
0.00
Density
10
15
20
25
30
Ash_Alcanity

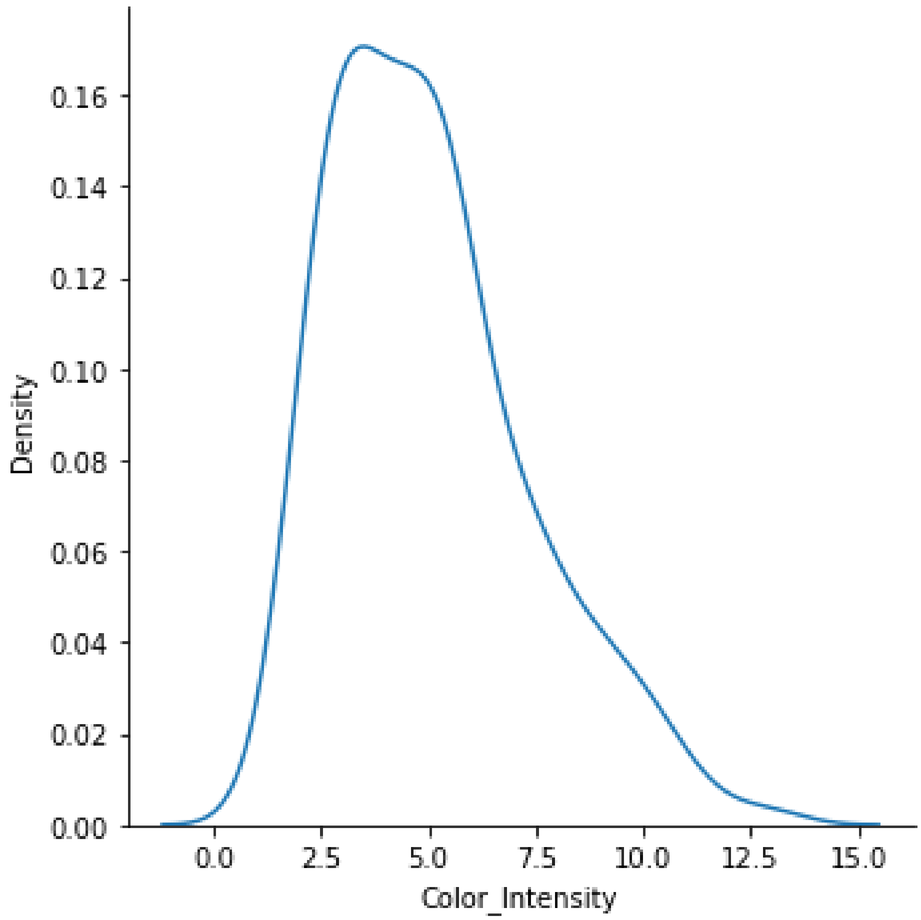

0.025
0.020
0.015
0.010
0.005
0.000
Density
60
80
100
120
140
160
180
Magnesium


Density
0.0
0.1
0.2
0.3
0.4
0.5
1
2
3
4
Total_Phenols

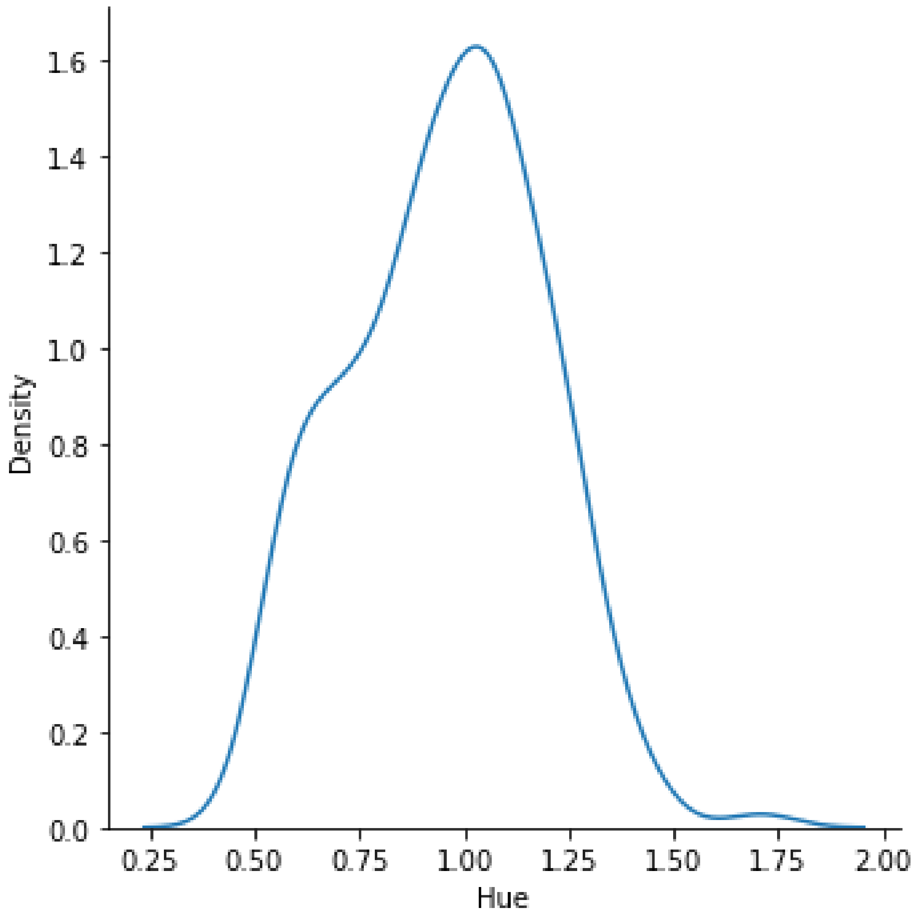

0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Density
−1
0
1
2
3
4
5
6
Flavanoids


3.0
2.5
2.0
1.5
1.0
0.5
0.0
Density
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Nonflavanoid_Phenols

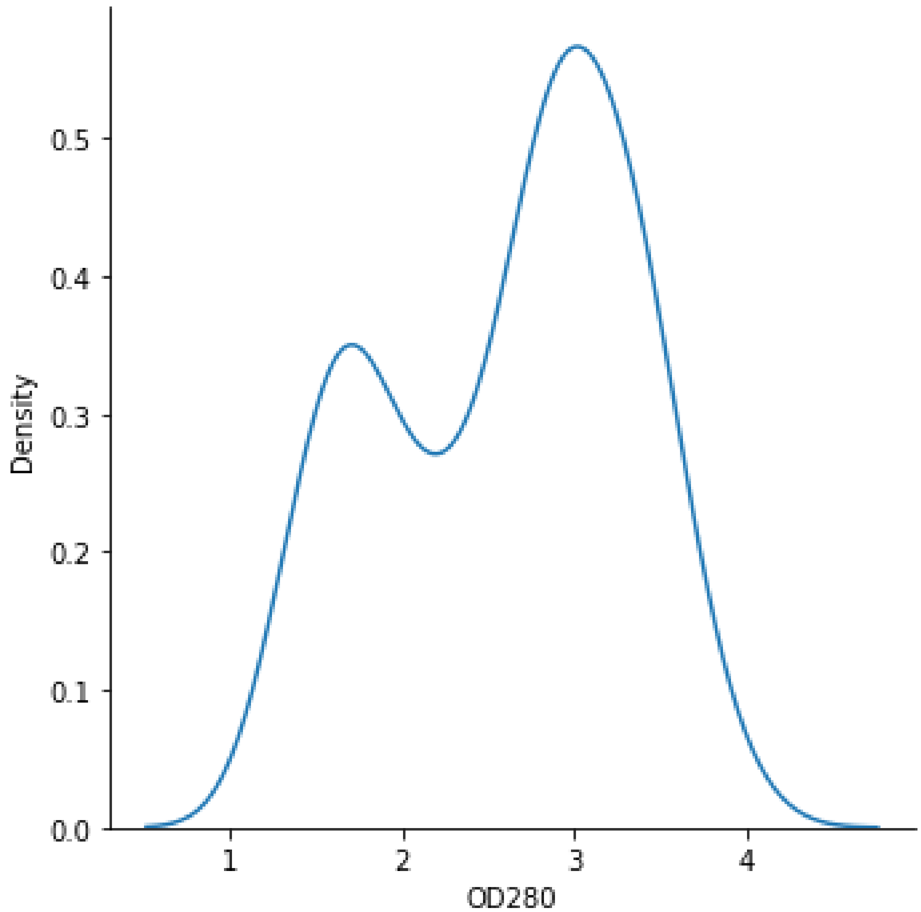

0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Density
0
1
2
3
4
Proanthocyanins



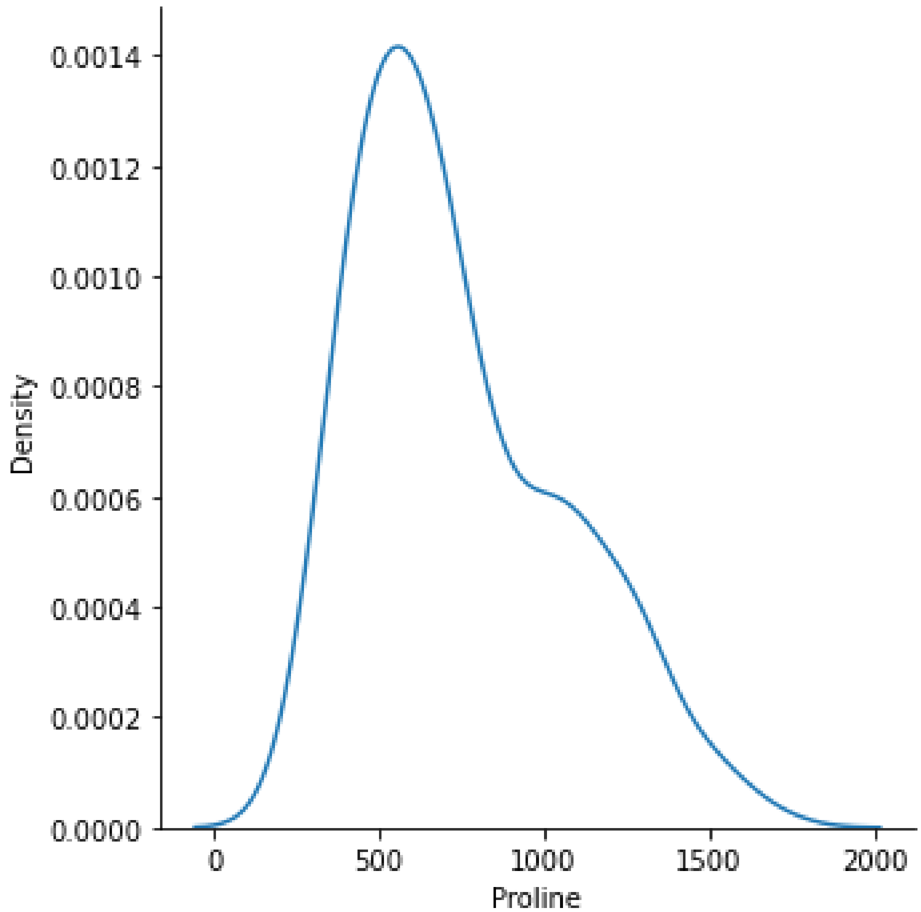
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
Density
0.0
2.5
5.0
7.5
10.0
12.5
15.0
Color_Intensity



1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Density
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Hue



Density
0.0
0.1
0.2
0.3
0.4
0.5
1
2
3
4
OD280



Plotting the distribution of the features in your datset is important because it helps to identify key distributions such as normal or multimodal distributions that at a glance can show two distinct cases present within one feature. Looking at the distribution can also help quickly identify key trends in the data or any outliers, this can later help to confirm any standard deviation values calculated later on the dataset.

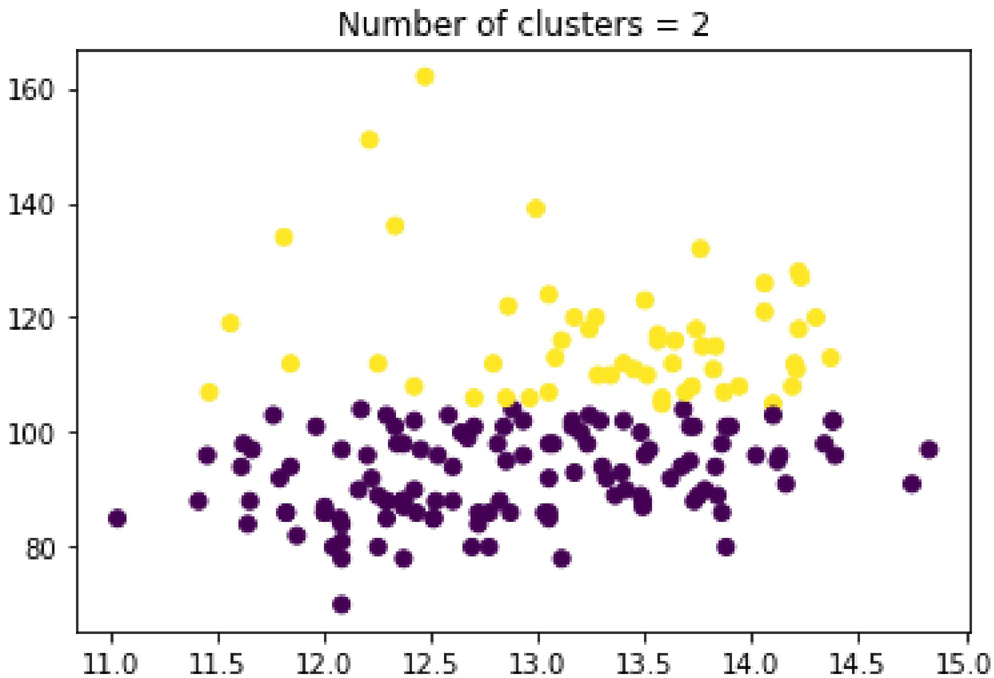
2.2

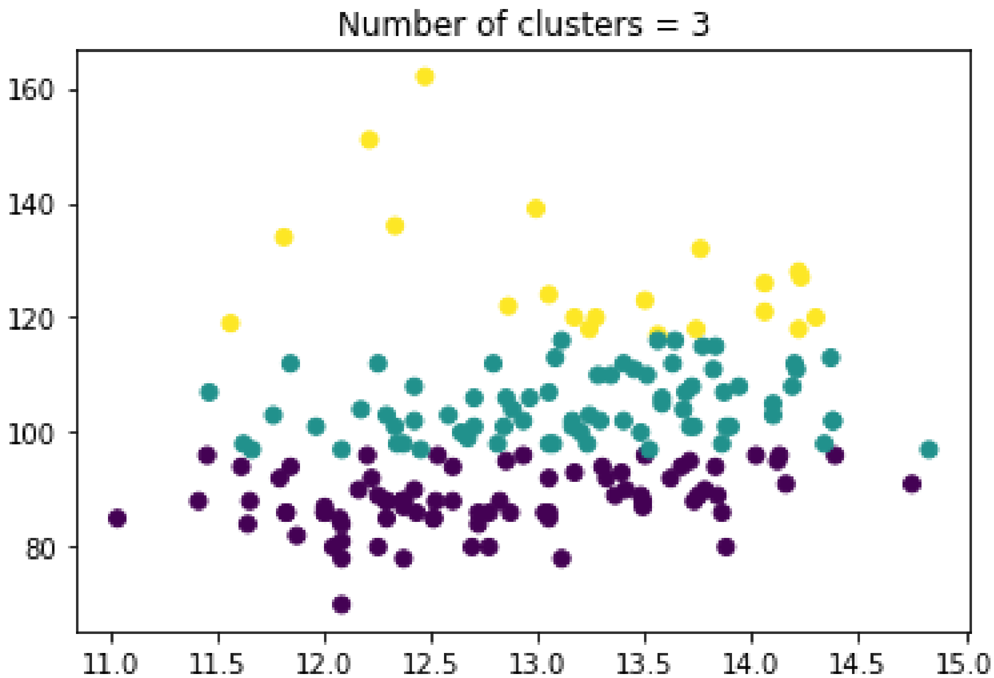
[14]:
```
k_values = [2,3,4,5,6,7]
X = wine[['Alcohol','Magnesium']]
X.head()
```

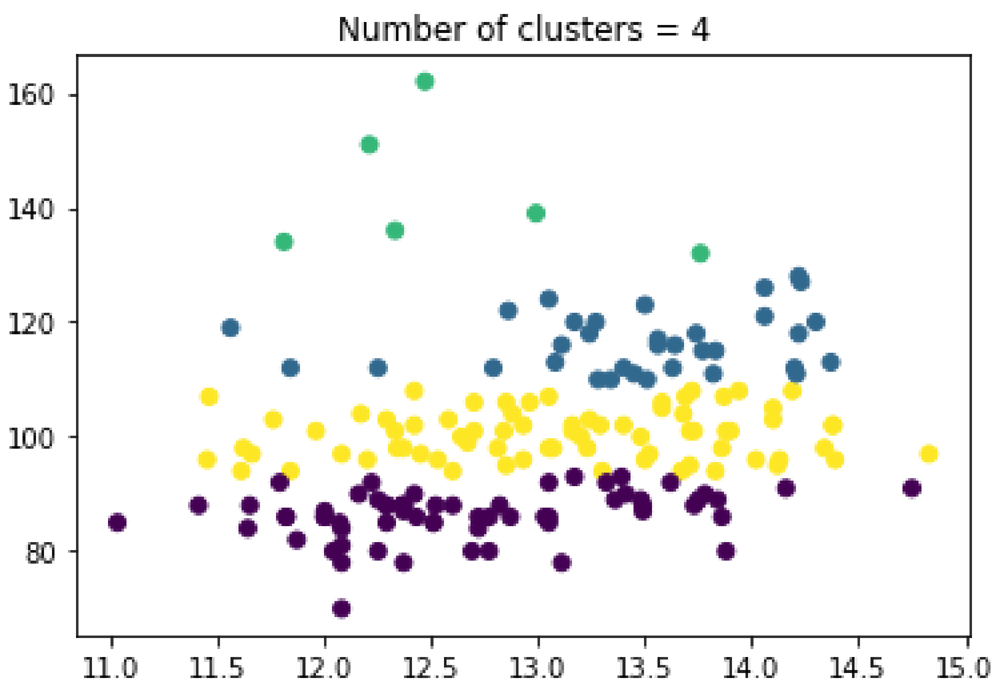
[14]:

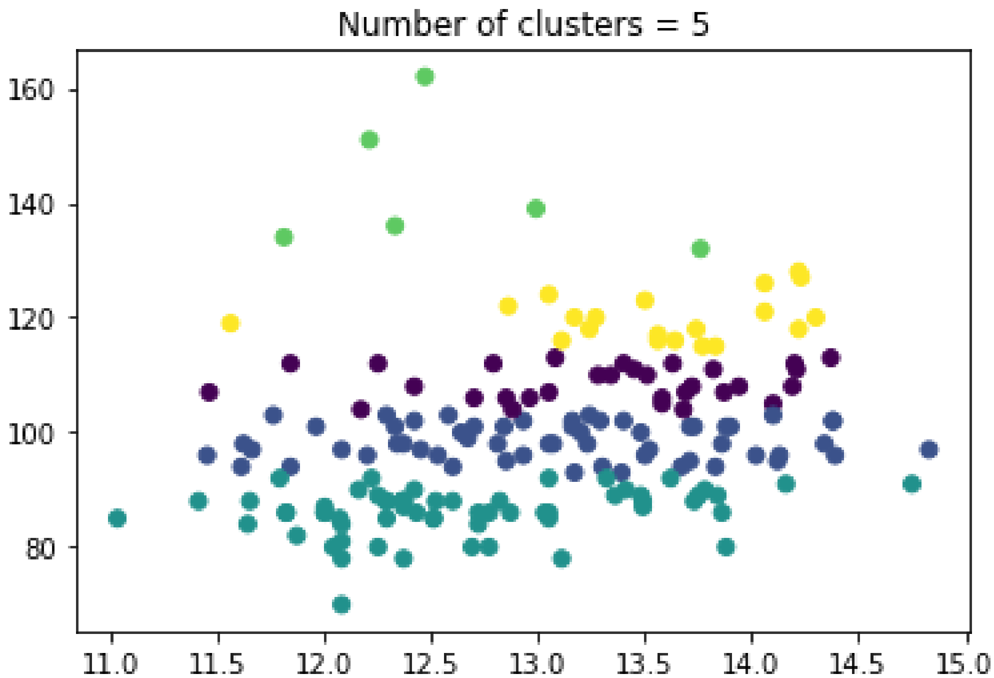
	Alcohol	Magnesium
0	14.23	127
1	13.20	100
2	13.16	101
3	14.37	113
4	13.24	118

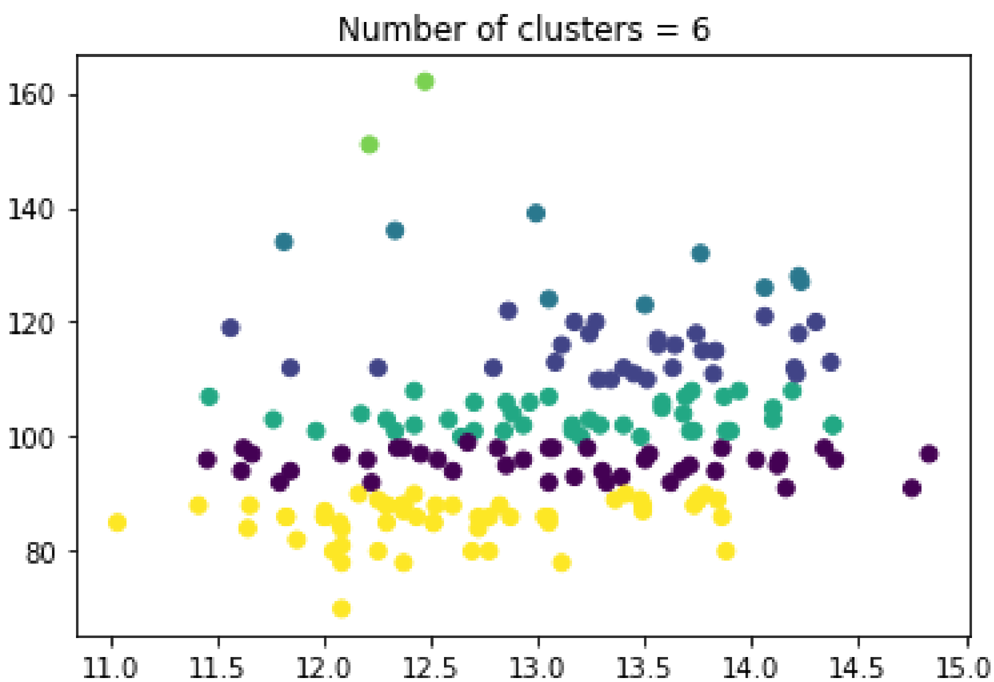
[15]:
```
score = []
for k in k_values:
    pred = KMeans(n_clusters=k, random_state=40)
    cluster = pred.fit_predict(X)
    score.append(pred.fit(X).inertia_)
    plt.scatter(X['Alcohol'], X['Magnesium'], c=cluster)
    plt.title("Number of clusters = %i"%k )
    plt.show()
```

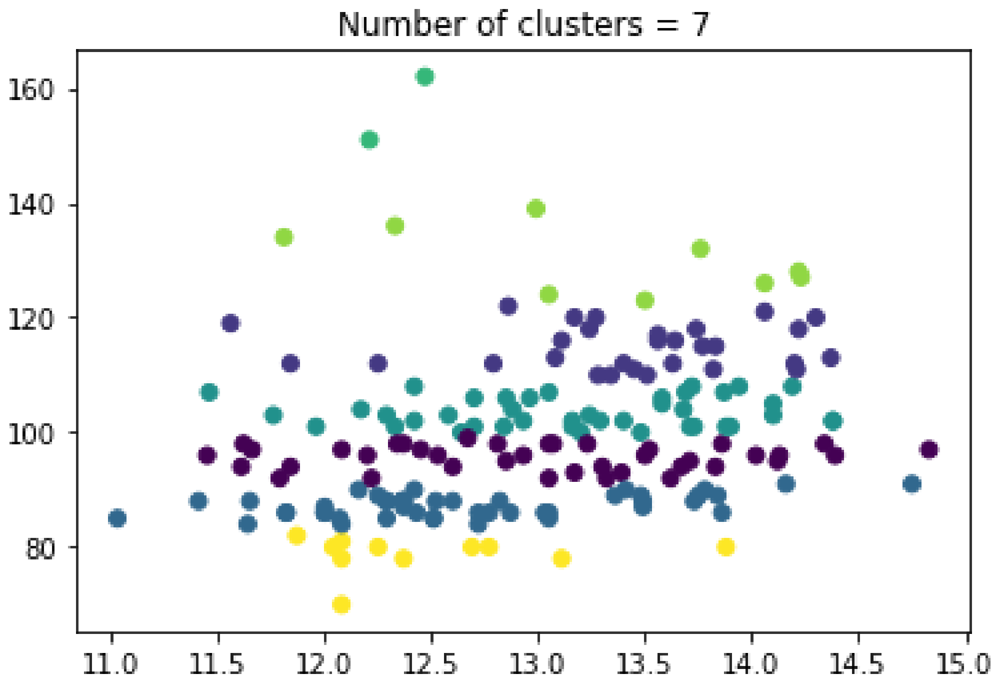



Number of clusters = 3
160
140
120
100
80
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0


Number of clusters = 4
160
140
120
100
80
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0


Number of clusters = 5
160
140
120
100
80
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0

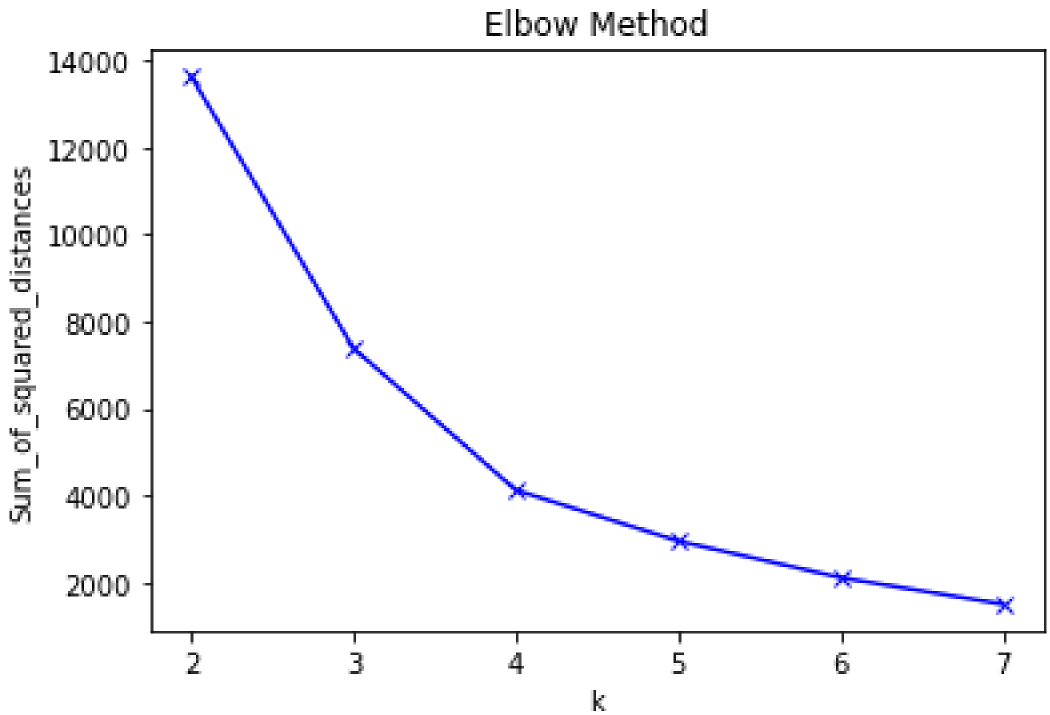

Number of clusters = 6
160
140
120
100
80
11.0
11.5
12.0
12.5
13.0
13.5
14.0
14.5
15.0



Judging the above plots the clusters become very entangled and without meaning towards the higher k values but there is good separation and distinction at K values of 3 and 4. K = 4 looks slightly better than 3 and will be the value of choice.

2.3

[16]:
```
plt.plot(k_values, score, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method')
plt.show()
```



3 Question 3

[17]:
```
admission = pd.read_csv('/Users/jeandre/DataScience874/PostBlockAssignment2/
↪admission_predict.csv')
```

[18]:
```
admission.head(2)
```

[18]:
```
   Serial No.  GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  \
0           1        337          118                  4  4.5   4.5  9.65
1           2        324          107                  4  4.0   4.5  8.87

   Research  Chance of Admit
0         1             0.92
1         1             0.76
```

[19]:
```
X = admission[list(admission.columns[1:8])]
X.head(2)
```

[19]:
```
   GRE Score  TOEFL Score  University Rating  SOP  LOR   CGPA  Research
0        337          118                  4  4.5   4.5  9.65         1
1        324          107                  4  4.0   4.5  8.87         1
```

[20]:
```
Y = admission['Chance of Admit']
Y.head(2)
```

[20]:
```
0    0.92
1    0.76
Name: Chance of Admit, dtype: float64
```

[21]:
```
X_train, X_test, Y_train, Y_test = sklearn.model_selection.
 ↪train_test_split(X,Y, test_size = 0.2, random_state = 0)
```

[22]:
```
reg = LinearRegression().fit(X_train, Y_train)
score = reg.score(X_test,Y_test)
print(score)
```

```
0.7355078738145215
```

[23]:
```
record1 = np.array([[322, 109, 5, 4.5, 3.5, 8.80, 0]])
record2 = np.array([[307, 52, 5, 4.4, 3.5, 8.20, 2]])
```

[24]:
```
print(reg.predict(record1))
print(reg.predict(record2))
```

```
[0.75784394]
[0.58930783]
```

Based on the input data the linear regression model derives a line of best fit that best matches the X and Y variables given. The algortihm is then given an X to predict and uses the line of best fit y=mx+b to predict what the value of Y should be from input X.

4 Question 4

[25]:
```
iris = pd.read_csv('/Users/jeandre/DataScience874/PostBlockAssignment2/
 ↪iris_dataset.csv')
```

[26]:
```
iris.head()
```

[26]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[27]:
```
X = iris[list(iris.columns[0:-1])]
Y = iris['species']
labels = iris['species'].unique()
```

```
X.head(2)
```

[27]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2

[28]:
```
X_train, X_test, Y_train, Y_test = sklearn.model_selection.
 ↪train_test_split(X,Y, test_size = 0.2, random_state = 0)
```

4.1 KNN

[29]:
```
k = 3
neigh = KNeighborsClassifier(n_neighbors=k)
neigh.fit(X_train,Y_train)
K_pred = neigh.predict(X_test)
print("Accuracy score =",sklearn.metrics.accuracy_score(Y_test,K_pred))
print("Precision score =",sklearn.metrics.
 ↪precision_score(Y_test,K_pred,average='weighted'))
print("Recall = ",sklearn.metrics.
 ↪recall_score(Y_test,K_pred,average='weighted'))
```

```
Accuracy score = 0.9666666666666667
Precision score = 0.9714285714285714
Recall =  0.9666666666666667
```

4.2 Decision tree

[30]:
```
tree = tree.DecisionTreeClassifier()
tree = tree.fit(X_train, Y_train)
D_pred = tree.predict(X_test)
print("Accuracy score =",sklearn.metrics.accuracy_score(Y_test,D_pred))
print("Precision score =",sklearn.metrics.
 ↪precision_score(Y_test,D_pred,average='weighted'))
print("Recall = ",sklearn.metrics.
 ↪recall_score(Y_test,D_pred,average='weighted'))
```

```
Accuracy score = 1.0
Precision score = 1.0
Recall =  1.0
```

4.3 Gausian Naive Bayes

[31]:
```
nb = GaussianNB()
nb.fit(X_train,Y_train)
N_pred = nb.predict(X_test)
print("Accuracy score =",sklearn.metrics.accuracy_score(Y_test,N_pred))
print("Precision score =",sklearn.metrics.
 ↪precision_score(Y_test,N_pred,average='weighted'))
print("Recall = ",sklearn.metrics.
 ↪recall_score(Y_test,N_pred,average='weighted'))
sklearn.metrics.confusion_matrix(Y_test,N_pred)
```

```
Accuracy score = 0.9666666666666667
Precision score = 0.9690476190476189
Recall =  0.9666666666666667
```

[31]:
```
array([[11,  0,  0],
       [ 0, 13,  0],
       [ 0,  1,  5]])
```

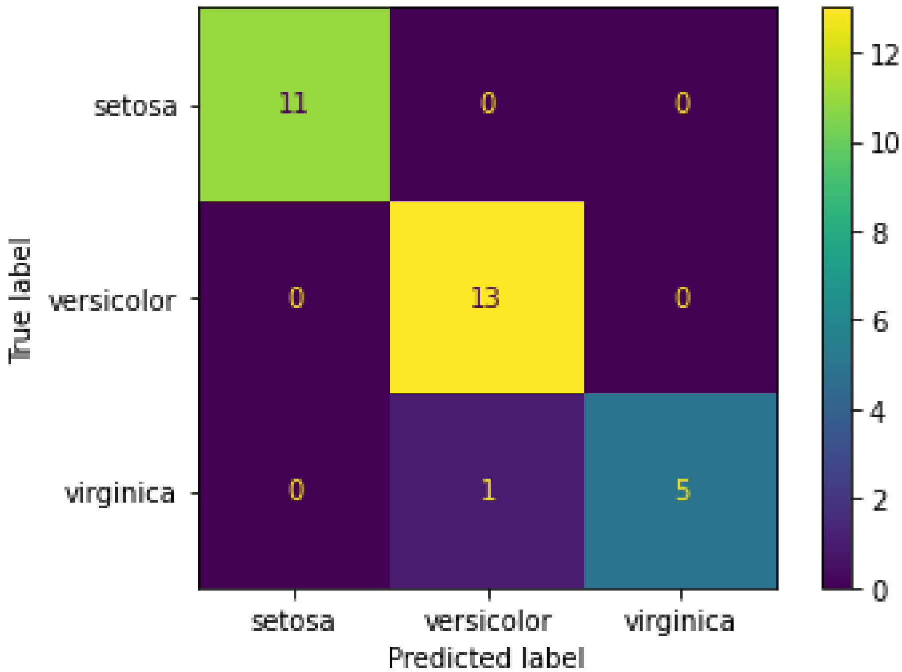
plotting the raw array above to validate the output of the plot below:

Naive bayes has a slightly lower performance than the other two models here because decision trees have a tendency to overfit the data and do much better with a large amount of data than with a small dataset. Here we have a relatively small dataset and makes it easy for the decision tree to overfit. Naive Bayes is a more robust method when it comes to lower amounts of data and wont overfit as easily.

Naive bayes operates based on probabilities rather than a distance metric such as KNN, using a larger dataset will be a better way to judge the performance of the three models.

[32]:
```
sklearn.metrics.plot_confusion_matrix(nb,X_test,Y_test,display_labels=labels)
```

[32]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f80f470e370>
```



A confusion matrix is always important to analyse after a classification process as it is not just a single valyue evaluating the performance of the model. Instead it offers a more detailed view at the classification process. It helps to see if the model is confusing one class with another or if it has poor performance over the whole class set or just has issues with one particular class. This makes the confusion matrix invaluable when it comes to diagnosing model performance.

[ ]:

```

```