Load and Organize Data

First let's import the usual data science modules

KNN Wine Classification

```
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
%matplotlib inline
```

Now from the skearn datasets let's load the 'wine' dataset and inspect it

```
wine=datasets.load_wine()
wine
```

```
{'data': array([[1.423e+01, 1.710e+00, 2.430e+00, ..., 1.040e+00,
3.920e+00,
         1.065e+03],
        [1.320e+01, 1.780e+00, 2.140e+00, ..., 1.050e+00, 3.400e+00,
         1.050e+03],
        [1.316e+01, 2.360e+00, 2.670e+00, ..., 1.030e+00, 3.170e+00,
         1.185e+03],
        ...,
        [1.327e+01, 4.280e+00, 2.260e+00, ..., 5.900e-01, 1.560e+00,
         8.350e+02],
        [1.317e+01, 2.590e+00, 2.370e+00, ..., 6.000e-01, 1.620e+00,
         8.400e+02],
        [1.413e+01, 4.100e+00, 2.740e+00, ..., 6.100e-01, 1.600e+00,
         5.600e+02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2,
2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
        2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
2,
```

```
       2, 2]),
 'frame': None,
 'target_names': array(['class_0', 'class_1', 'class_2'],
dtype='<U7'),
 'DESCR': '.. _wine_dataset:\n\nWine recognition dataset\
n----------------------\n\n**Data Set Characteristics:**\n\
n    :Number of Instances: 178\n    :Number of Attributes: 13 numeric,
predictive attributes and the class\n    :Attribute Information:\n \t\
t- Alcohol\n \t\t- Malic acid\n \t\t- Ash\n\t\t- Alcalinity of ash  \n
\t\t- Magnesium\n\t\t- Total phenols\n \t\t- Flavanoids\n \t\t-
Nonflavanoid phenols\n \t\t- Proanthocyanins\n\t\t- Color intensity\
n \t\t- Hue\n \t\t- OD280/OD315 of diluted wines\n \t\t- Proline\n\n
- class:\n            - class_0\n            - class_1\n            -
class_2\n\t\t\n    :Summary Statistics:\n    \n
============================== ==== ===== ======= =====\n
Min   Max   Mean     SD\n    ============================== ==== =====
======= =====\n    Alcohol:                      11.0  14.8    13.0
0.8\n    Malic Acid:                  0.74  5.80    2.34  1.12\n
Ash:                           1.36  3.23    2.36  0.27\n    Alcalinity
of Ash:            10.6  30.0    19.5   3.3\n    Magnesium:
70.0 162.0    99.7  14.3\n    Total Phenols:                0.98  3.88
2.29  0.63\n    Flavanoids:                   0.34  5.08    2.03
1.00\n    Nonflavanoid Phenols:         0.13  0.66    0.36  0.12\n
Proanthocyanins:              0.41  3.58    1.59  0.57\n    Colour
Intensity:              1.3  13.0     5.1   2.3\n    Hue:
0.48  1.71    0.96  0.23\n    OD280/OD315 of diluted wines: 1.27  4.00
2.61  0.71\n    Proline:                       278  1680     746
315\n    ============================== ==== ===== ======= =====\n\n
:Missing Attribute Values: None\n    :Class Distribution: class_0
(59), class_1 (71), class_2 (48)\n    :Creator: R.A. Fisher\
n    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)\
n    :Date: July, 1988\n\nThis is a copy of UCI ML Wine recognition
datasets.\nhttps://archive.ics.uci.edu/ml/machine-learning-databases/
wine/wine.data\n\nThe data is the results of a chemical analysis of
wines grown in the same\nregion in Italy by three different
cultivators. There are thirteen different\nmeasurements taken for
different constituents found in the three types of\nwine.\n\nOriginal
Owners: \n\nForina, M. et al, PARVUS - \nAn Extendible Package for
Data Exploration, Classification and Correlation. \nInstitute of
Pharmaceutical and Food Analysis and Technologies,\nVia Brigata
Salerno, 16147 Genoa, Italy.\n\nCitation:\n\nLichman, M. (2013). UCI
Machine Learning Repository\n[https://archive.ics.uci.edu/ml]. Irvine,
CA: University of California,\nSchool of Information and Computer
Science. \n\n.. topic:: References\n\n  (1) S. Aeberhard, D. Coomans
and O. de Vel, \n  Comparison of Classifiers in High Dimensional
Settings, \n  Tech. Rep. no. 92-02, (1992), Dept. of Computer Science
and Dept. of  \n  Mathematics and Statistics, James Cook University of
North Queensland. \n  (Also submitted to Technometrics). \n\n  The
data was used with many others for comparing various \n  classifiers.
The classes are separable, though only RDA \n  has achieved 100%
```

```
correct classification. \n  (RDA : 100%, QDA 99.4%, LDA 98.9%, 1NN
96.1% (z-transformed data)) \n  (All results using the leave-one-out
technique) \n\n  (2) S. Aeberhard, D. Coomans and O. de Vel, \n  "THE
CLASSIFICATION PERFORMANCE OF RDA" \n  Tech. Rep. no. 92-01, (1992),
Dept. of Computer Science and Dept. of \n  Mathematics and Statistics,
James Cook University of North Queensland. \n  (Also submitted to
Journal of Chemometrics).\n',
 'feature_names': ['alcohol',
  'malic_acid',
  'ash',
  'alcalinity_of_ash',
  'magnesium',
  'total_phenols',
  'flavanoids',
  'nonflavanoid_phenols',
  'proanthocyanins',
  'color_intensity',
  'hue',
  'od280/od315_of_diluted_wines',
  'proline']}
```

The data is the results of a chemical analysis of wines grown in the same region in Italy by three different cultivators. There are thirteen different measurements taken for different constituents found in the three types of wine.

```
df=pd.DataFrame(wine["data"],columns=wine["feature_names"])
df["target"]=wine["target"]
df['class']=df['target'].map(lambda ind: wine['target_names'][ind])
df.head()
```

```
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium
total_phenols  \
0    14.23        1.71  2.43               15.6      127.0
2.80
1    13.20        1.78  2.14               11.2      100.0
2.65
2    13.16        2.36  2.67               18.6      101.0
2.80
3    14.37        1.95  2.50               16.8      113.0
3.85
4    13.24        2.59  2.87               21.0      118.0
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
```

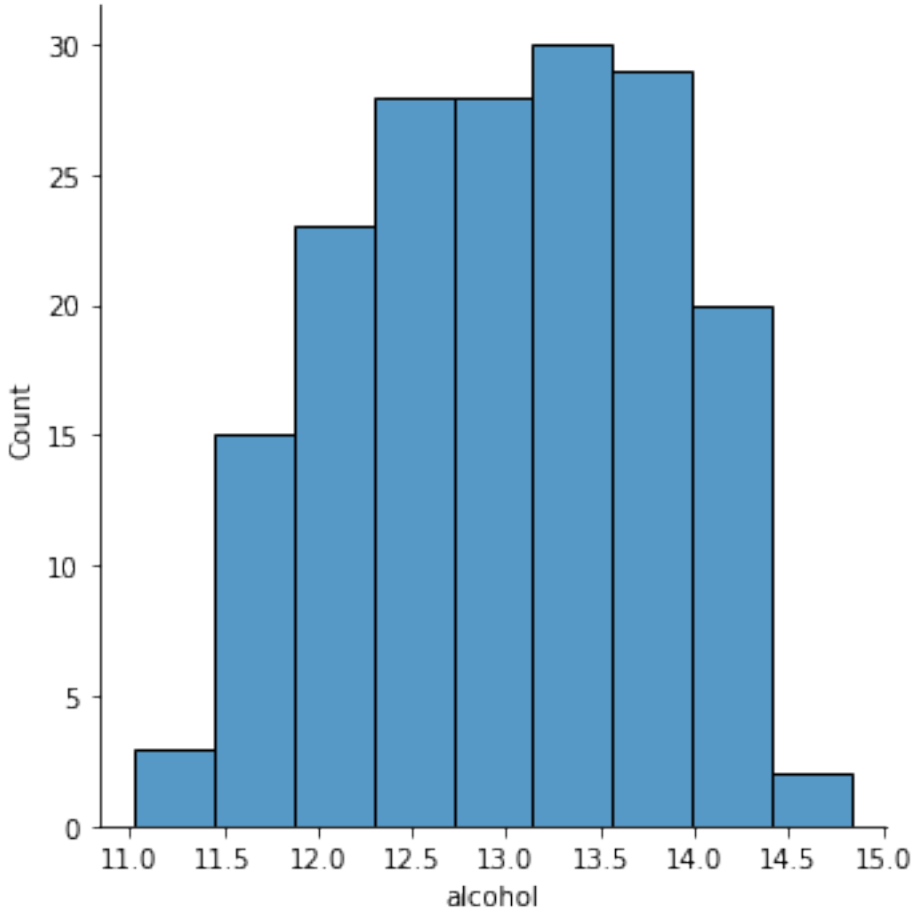
```
1.03
3        3.49                  0.24             2.18              7.80
0.86
4        2.69                  0.39             1.82              4.32
1.04
```

```
   od280/od315_of_diluted_wines  proline  target    class
0                          3.92   1065.0       0  class_0
1                          3.40   1050.0       0  class_0
2                          3.17   1185.0       0  class_0
3                          3.45   1480.0       0  class_0
4                          2.93    735.0       0  class_0
```

What is distribution of alcohol content among all of wines?

```
sns.displot(df['alcohol'],kde=0)
```

```
<seaborn.axisgrid.FacetGrid at 0x2a1d6efe4a0>
```

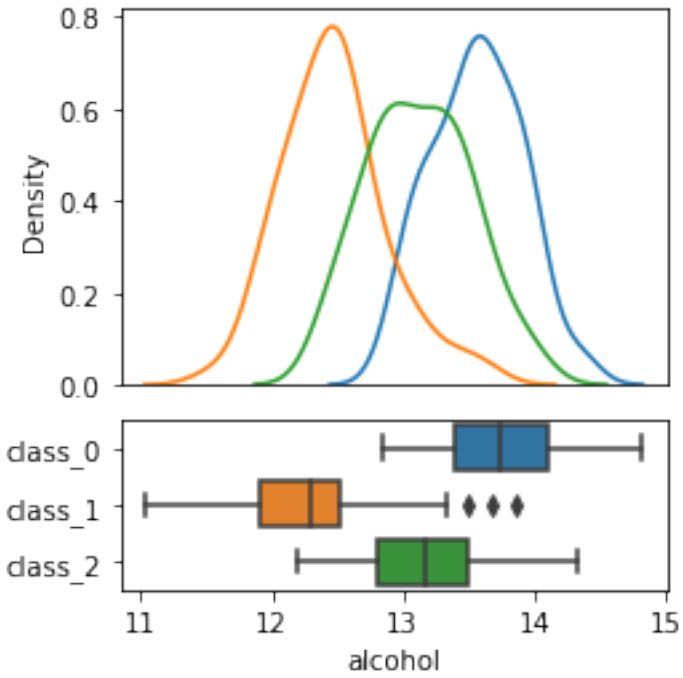


Distribution of classes

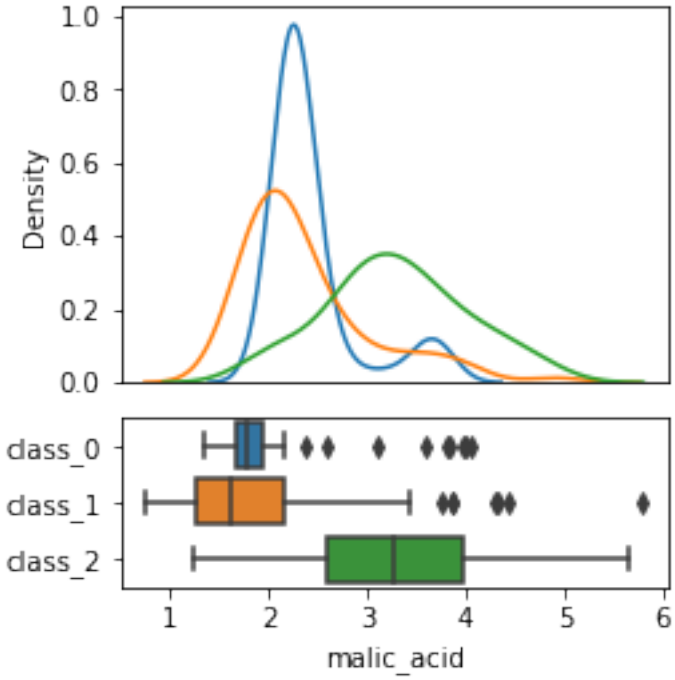
```
import matplotlib.gridspec as gridspec
for feature in wine["feature_names"]:
    print(feature)
```

```
    gs1 = gridspec.GridSpec(3,1)
    ax1 = plt.subplot(gs1[:-1])
    ax2 = plt.subplot(gs1[-1])
    gs1.update(right=0.60)
    sns.boxplot(x=feature,y='class',data=df,ax=ax2)
    sns.kdeplot(df[feature][df.target==0],ax=ax1,label='0')
    sns.kdeplot(df[feature][df.target==1],ax=ax1,label='1')
    sns.kdeplot(df[feature][df.target==2],ax=ax1,label='2')
    ax2.yaxis.label.set_visible(False)
    ax1.xaxis.set_visible(False)
    plt.show()
```

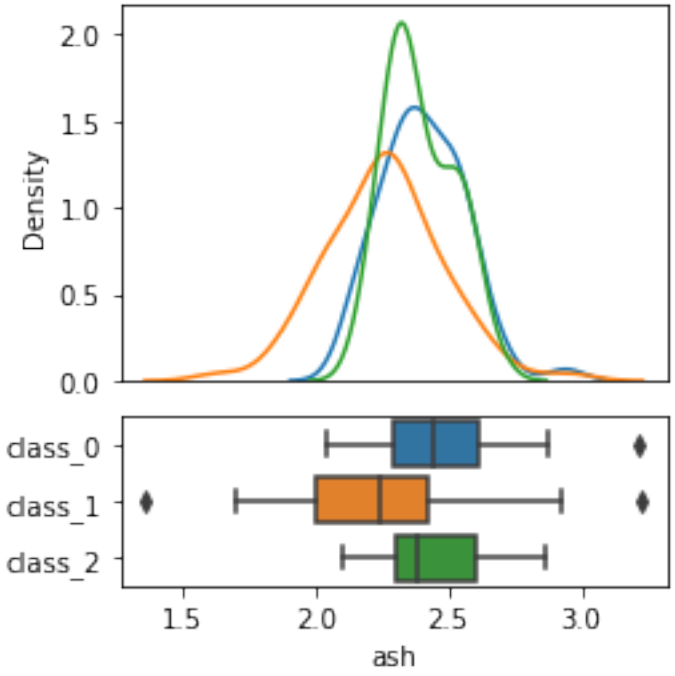
alcohol



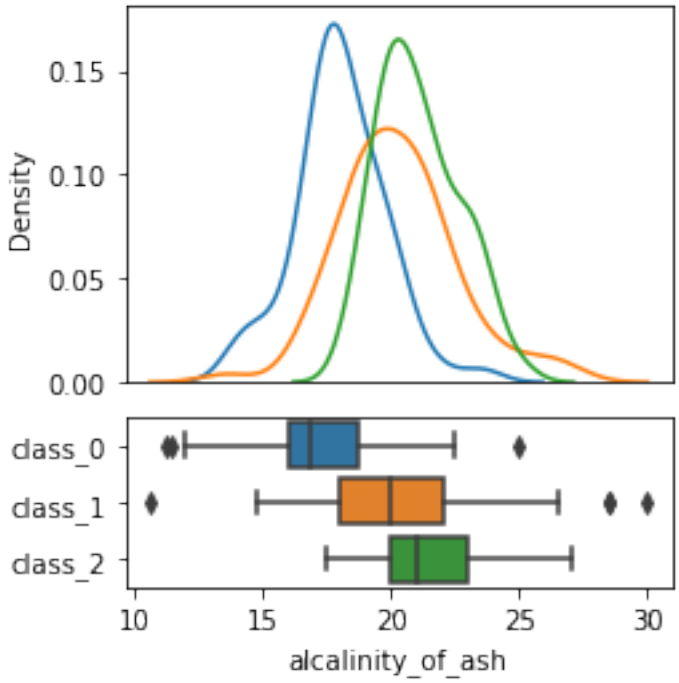
malic_acid



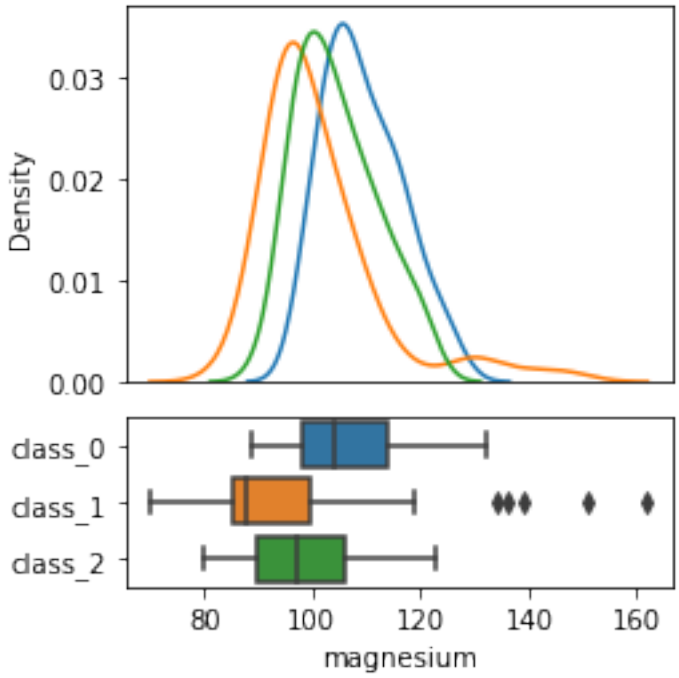
ash



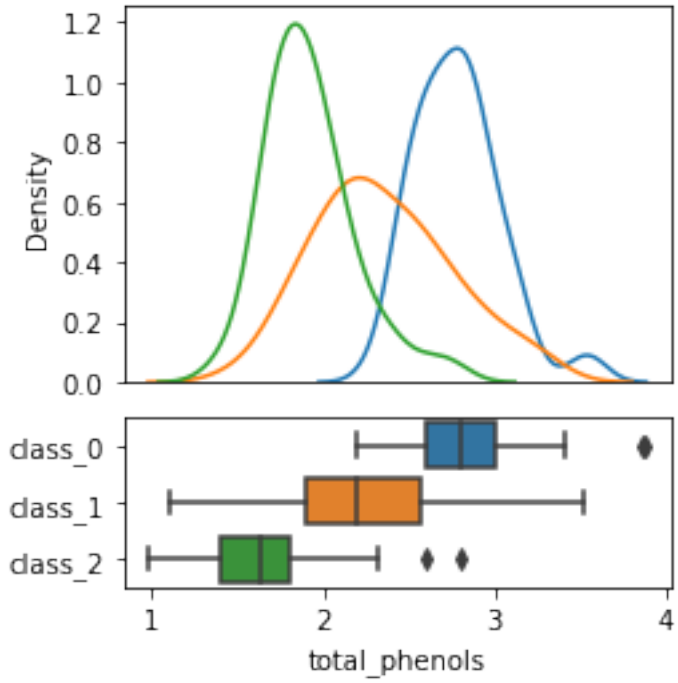
alcalinity_of_ash



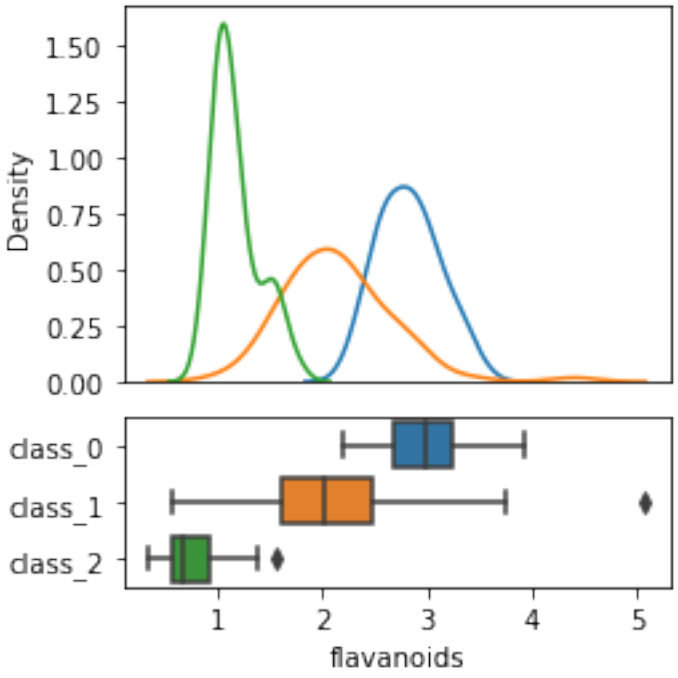
magnesium



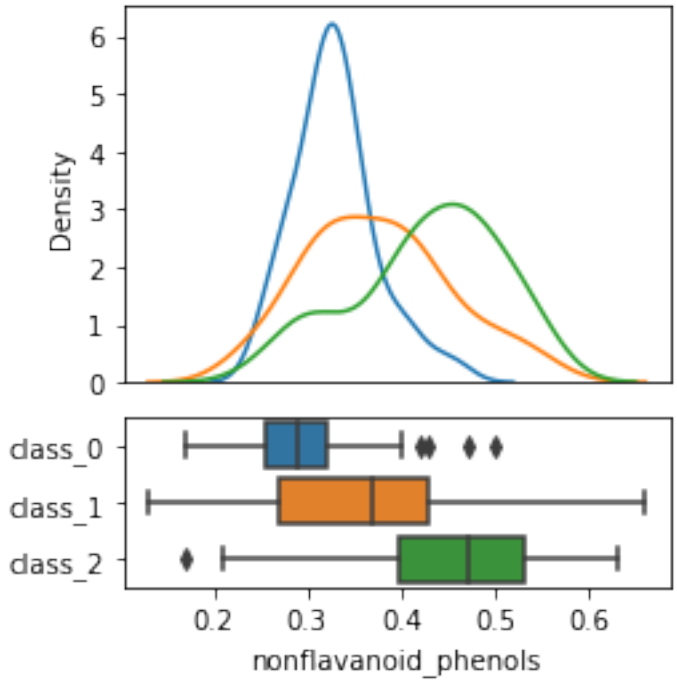
total_phenols



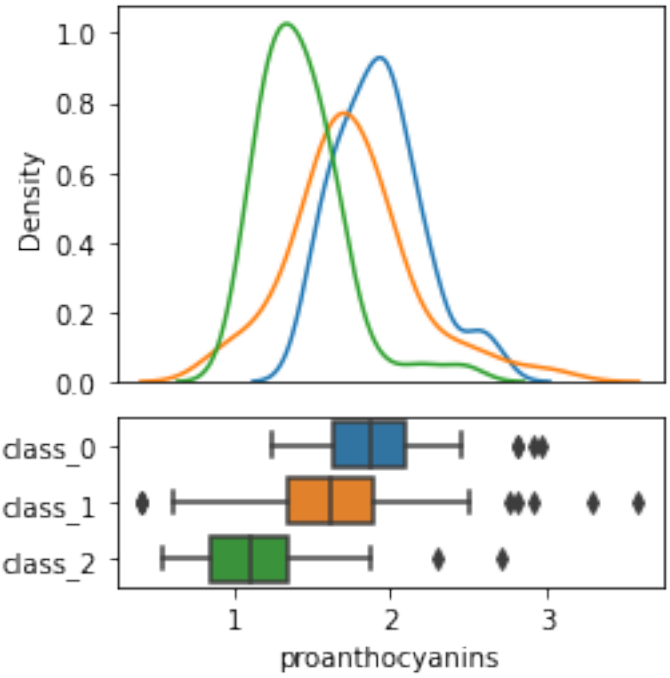
flavanoids



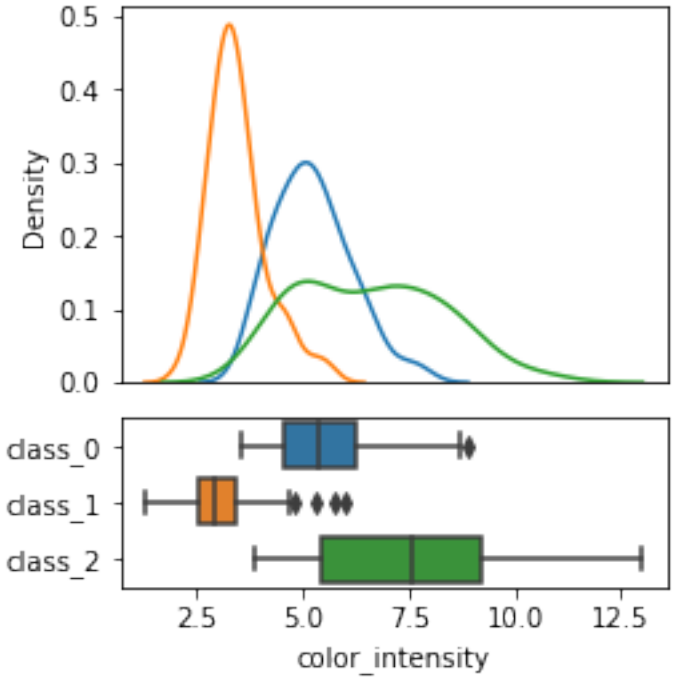
nonflavanoid_phenols



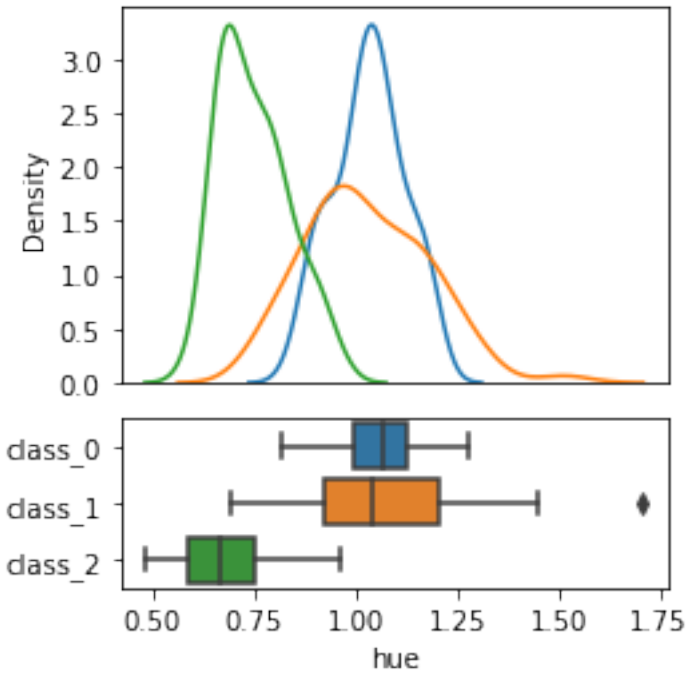
proanthocyanins



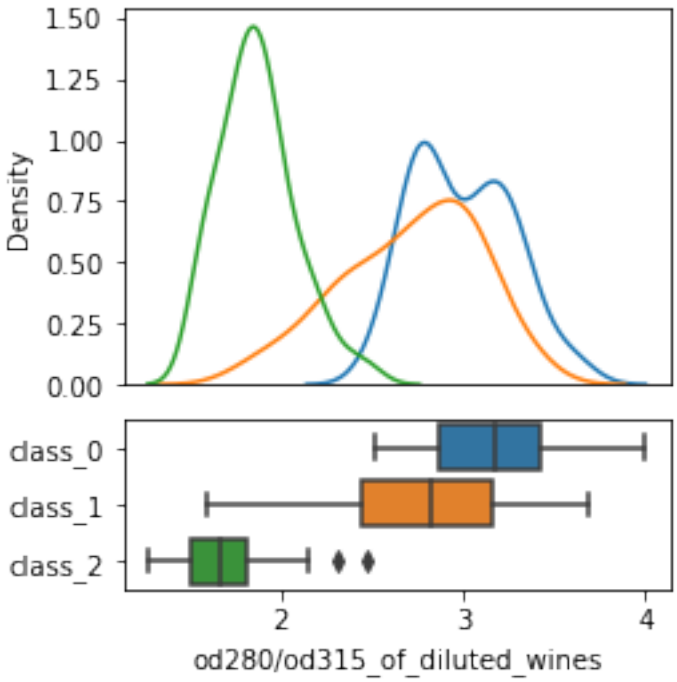
color_intensity



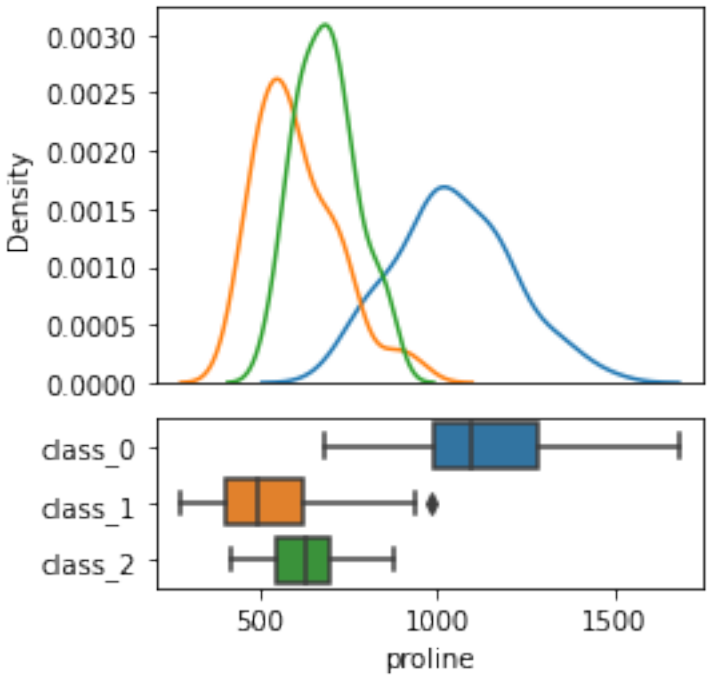
hue



od280/od315_of_diluted_wines




```
proline
```



Test Train Split

```
x=df
y=x.pop("class")

x.head()
```

```
   alcohol  malic_acid   ash  alcalinity_of_ash  magnesium
total_phenols  \
0    14.23        1.71  2.43               15.6      127.0
2.80
1    13.20        1.78  2.14               11.2      100.0
2.65
2    13.16        2.36  2.67               18.6      101.0
2.80
3    14.37        1.95  2.50               16.8      113.0
3.85
4    13.24        2.59  2.87               21.0      118.0
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/od315_of_diluted_wines  proline
0                          3.92   1065.0
1                          3.40   1050.0
2                          3.17   1185.0
3                          3.45   1480.0
4                          2.93    735.0

y.head()

0    class_0
1    class_0
2    class_0
3    class_0
4    class_0
Name: class, dtype: object

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.7,rando
m_state=55)
```

Split data

In order to effectively train and test our model, we need to separate the data into a training set which we will feed to our model along the the training labels. Then after we have

trained the model, we will test it on the 'test' data, so that we can gauge the real-world applicability of the model.

Scikit-learn has a useful functionality here with the `train_test_split()` method. `test_size` governs the proportion of data that is reserved for testing. We want to train on enough data that our model can make good predictions but we also need enough test data to determine if we've overfit the model. We'll use 30% of the data for testing.

```
x_train.shape
```

```
(53, 13)
```

```
x_test.shape
```

```
(125, 13)
```

Training

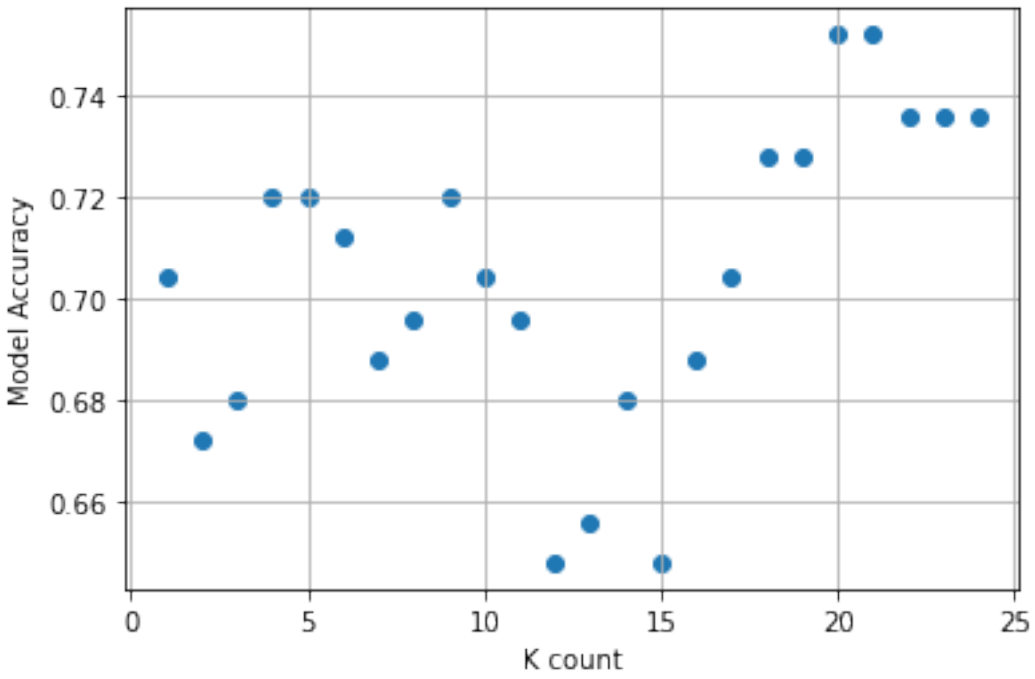
Number of Neigbhors K value = 20

```
knn = KNeighborsClassifier(n_neighbors=20)
knn.fit(x_train,y_train)
knn.score(x_test,y_test)
```

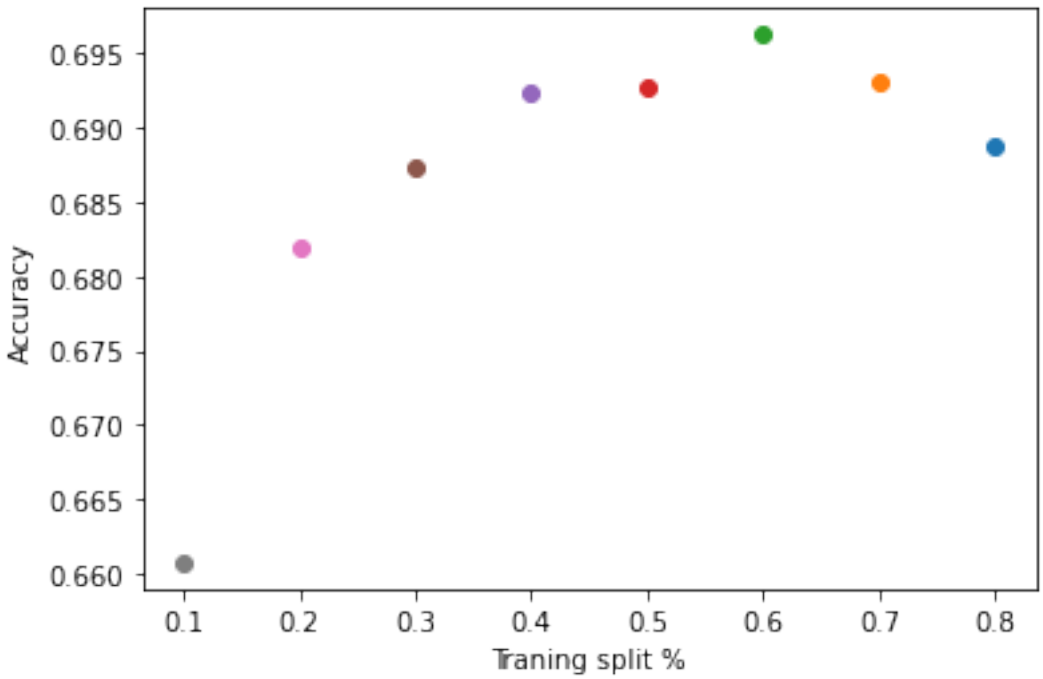
```
0.752
```

Tunning Sensitivity of model to n_neighbors

```
k_range =range(1,25)
scores =[]

for k in k_range:
    knn= KNeighborsClassifier(n_neighbors=k)
    knn.fit(x_train,y_train)
    scores.append(knn.score(x_test,y_test))


plt.figure()
plt.xlabel("K count")
plt.ylabel("Model Accuracy")
plt.scatter(k_range, scores)
plt.grid()
plt.xticks([0, 5, 10, 15, 20, 25])
plt.show()
```




```
test_sizes= [0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1]

knn= KNeighborsClassifier(n_neighbors= 5)

plt.figure()

for test_size in test_sizes:
    scores = []
    for i in range(1, 1000):
        x_train, x_test, y_train, y_test =train_test_split(x, y,
test_size = 1 - test_size)
        knn.fit(x_train, y_train)
        scores.append(knn.score(x_test, y_test))
    plt.scatter(test_size, np.mean(scores))

plt.xlabel("Traning split %")
plt.ylabel("Accuracy")

Text(0, 0.5, 'Accuracy')
```



Predictions

```
prediction=knn.predict(x_test)
prediction
```

```
array(['class_1', 'class_1', 'class_0', 'class_1', 'class_0',
'class_0',
       'class_0', 'class_2', 'class_1', 'class_0', 'class_1',
'class_0',
       'class_2', 'class_0', 'class_2', 'class_1', 'class_0',
'class_0',
       'class_1', 'class_2', 'class_0', 'class_0', 'class_1',
'class_1',
       'class_0', 'class_1', 'class_2', 'class_2', 'class_1',
'class_1',
       'class_2', 'class_0', 'class_1', 'class_1', 'class_2',
'class_1',
       'class_0', 'class_0', 'class_0', 'class_0', 'class_1',
'class_1',
       'class_1', 'class_0', 'class_0', 'class_2', 'class_0',
'class_1',
       'class_2', 'class_1', 'class_1', 'class_1', 'class_2',
'class_1',
       'class_1', 'class_2', 'class_2', 'class_0', 'class_0',
'class_1',
       'class_2', 'class_1', 'class_2', 'class_1', 'class_1',
'class_0',
       'class_1', 'class_1', 'class_0', 'class_1', 'class_0',
'class_0',
       'class_2', 'class_1', 'class_1', 'class_1', 'class_2',
```

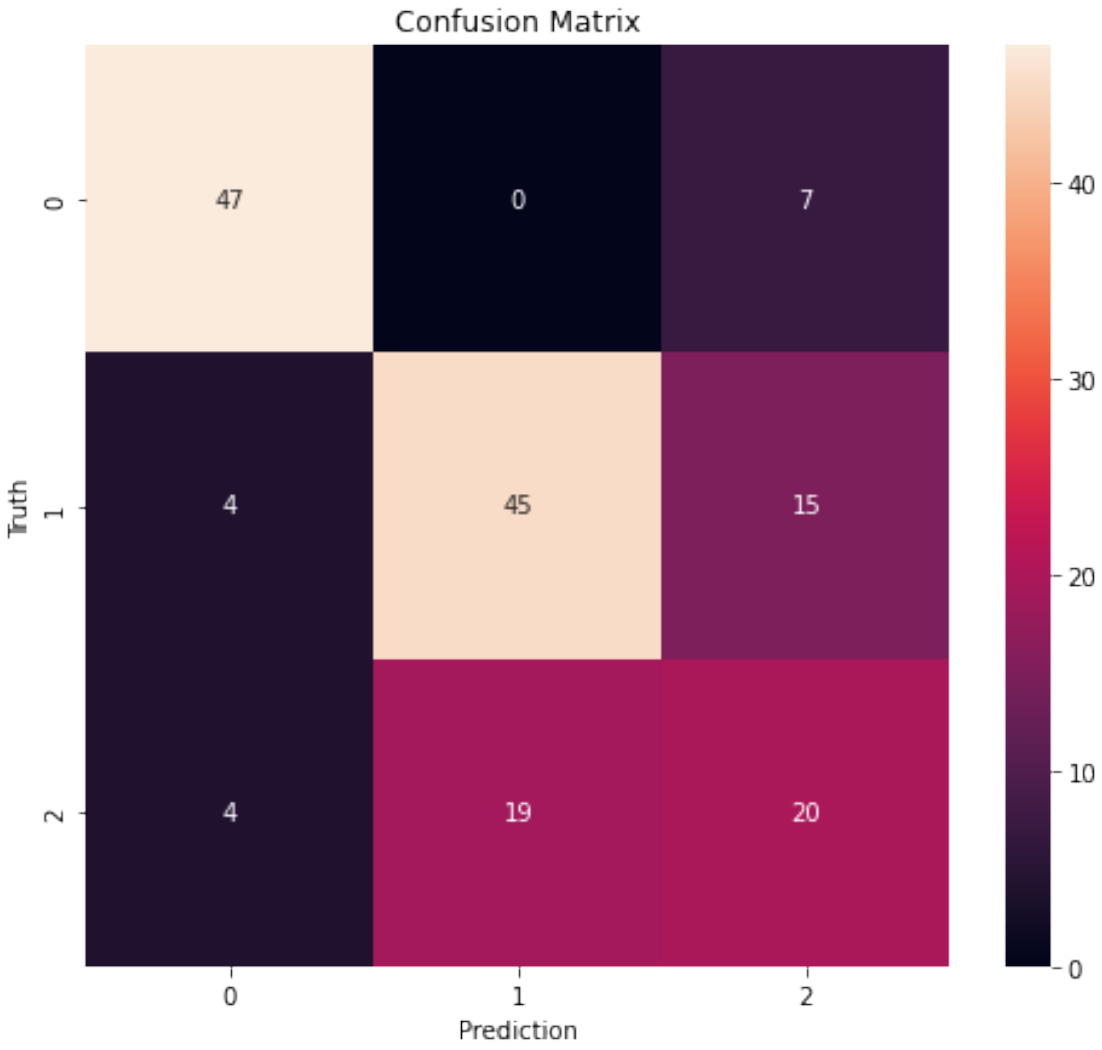
```
'class_0',
       'class_0', 'class_1', 'class_1', 'class_2', 'class_0',
'class_2',
       'class_0', 'class_0', 'class_2', 'class_0', 'class_0',
'class_0',
       'class_2', 'class_2', 'class_2', 'class_0', 'class_2',
'class_2',
       'class_1', 'class_0', 'class_0', 'class_1', 'class_0',
'class_1',
       'class_1', 'class_2', 'class_1', 'class_0', 'class_1',
'class_2',
       'class_1', 'class_1', 'class_1', 'class_1', 'class_2',
'class_1',
       'class_1', 'class_1', 'class_2', 'class_1', 'class_0',
'class_0',
       'class_0', 'class_1', 'class_1', 'class_2', 'class_1',
'class_0',
       'class_0', 'class_0', 'class_2', 'class_1', 'class_2',
'class_2',
       'class_1', 'class_0', 'class_1', 'class_1', 'class_2',
'class_2',
       'class_0', 'class_0', 'class_1', 'class_2', 'class_0',
'class_0',
       'class_2', 'class_1', 'class_0', 'class_2', 'class_0',
'class_1',
       'class_2', 'class_2', 'class_1', 'class_0', 'class_1',
'class_0',
       'class_0', 'class_2', 'class_2', 'class_1', 'class_1'],
      dtype=object)
```

```
cm = confusion_matrix(y_test, prediction)
```

```
cm
```

```
array([[47,  0,  7],
       [ 4, 45, 15],
       [ 4, 19, 20]], dtype=int64)
```

```
plt.figure(figsize=(8,7))
sns.heatmap(cm,annot=True)
plt.title("Confusion Matrix")
plt.ylabel("Truth")
plt.xlabel("Prediction")
plt.show()
```


Confusion Matrix
47
0
7
4
45
15
4
19
20
Truth
0
1
2
Prediction
0
1
2
40
30
20
10
0