Assignment_2_SML

Name: Ayush Patel

NUID: 002765119

Collabrations: Abhinav Nippani, Adwait Patil

Problem 1

a)

1. While using lasso regression, one may minimize or regularize these coefficients to prevent overfitting and improve their performance across different datasets.
2. When the dataset exhibits strong multicollinearity or when you wish to automate feature selection and variable selection, this sort of regression is utilized.
3. If there are only a few relevant factors and the rest are close to zero, Lasso usually performs well. One clear benefit of lasso regression over ridge regression is that it generates models that are less complex and easier to understand because they only include a small number of variables.
4. Ridge regression is a technique used to eliminate multicollinearity in data models.
5. Ridge regression is the best technique when there are fewer observations than predictor variables.
6. Even as small changes in the data used to fit the regression are considered, ridge regression model generally display consistency. When the independent variables are substantially multicollinear, however, ordinary least squares estimates may be quite unstable.
7. If there are multiple significant parameters that have roughly the same value, Ridge performs well.
8. Since ridge regression never leads to a coefficient being zero but simply minimizes it, it reduces the complexity of a model without reducing the number of variables.
9. According to me, Ridge regression is better as it uses all the variables instead of dropping the values and minimalizing them to zero.

b)

1. **P-value** is a number describing how likely it is that the data would have occurred under the null hypothesis of a statistical test.

2. The bigger the p-value the more likely the observed sample proportion is resulted by chance and the less we would reject the null hypothesis, but the rejection depends on the significance level we set beforehand. If the p-value is smaller than the significance level we would be confident enough that the observed sample proportion is not led by chance and we would reject the null hypothesis.

3. A **confidence interval** is defined as the range of values that are observed in a sample and for which we expect to find the value that accurately reflects the population.

4. It is an interval statistic used to assess the uncertainty in an estimate. The importance of a confidence interval is its capacity to quantify the uncertainty of an estimate. It provides both a lower and upper bound and a likelihood.

c)

1. The maximum likelihood estimation is the value that maximizes the probability of the observed data. The likelihood function is the probability that the observed values of the dependent variable may be predicted from the observed values of the independent variables. The likelihood varies from 0 to 1.

2. In logistic regression, we want a model that, using the sigmoid curve's best fit, predicts whether the output class will be 0 or 1. To do this, we must determine the best value for the regression coefficients. You may determine how accurate your predictions are in relation to true values using a cost function. Therefore, a cost function that maximizes the likelihood of obtaining output values should exist. We shall use maximum likelihood estimate there.

3. When using maximum likelihood estimation, it is necessary to first assume a probability distribution for the target variable (class label) before defining a likelihood function that determines the likelihood of actually observing the result given the input data and the model. Regression coefficients are selected to optimize the maximum likelihood function.

d)

Explaination:

1. **Variance** describes how the model varies when various parts of the training data set are used. Variance is the range of the model predictions that the machine learning function can make based on the given data.

2. **Covariance matrix** captures how all the variables in a dataset may vary simultaneously and extends the idea of variance to multiple dimensions. A lot may be learned about the variables from the covariance matrix, which is a matrix that summarizes the variances and covariances of a collection of vectors. The variance of each vector is represented by the diagonal of the matrix.

3. A **correlation coefficient** is a statistical measure of how well changes in one variable's value predict changes in another. When two variables are positively correlated, the value either increases or decreases simultaneously. When two variables are negatively correlated, the value of one increases as the value of the other decreases.

4. Correlation coefficients are given values ranging from +1 to -1. A coefficient of 1 denotes a complete positive correlation, meaning that any change in one variable's value will cause the other to change in the same direction. A perfect negative correlation has a coefficient of -1, meaning that any change in one variable's value will cause the other to change in the opposite direction. Non-zero decimals are used to represent correlations with lower degrees. There is no obvious correlation between the variations of the variables, as shown by a coefficient of zero.

Difference:

1. Variance measures the spread between all data points in a dataset.
2. Covariance finds whether two data points are directly or inversely proportional.
3. Correlation finds the direction of the relationship along with the degree of the relationship.
4. The value of covariance lies in the range of -∞ and +∞.
5. Correlation is limited to values between the range -1 and +1.

e)

Here are the **assumptions** needed to perform Linear Discriminant Analysis(LDA).

1. All the predcitions are drawn from a normal or a multivariate Gaussian distribution with some correlation between them.
2. For p > 1, we assume that the observation of the k^{th} class are drawn from a multivariate Gaussian distribution $N(\mu_k, \sum)$ where μ_k is the mean and $\sum$ is the covariance matrix which is same for all the k classes.
3. LDA uses Bayes classification to assign an observation to the class for which the result is largest: $\delta_k(x) = x^T.(\sum)^{-1}.\mu_k - (1/2).\mu_k^T.(\sum)^{-1}.\mu_k + log(\pi_k)$

   Now, we need to estimate the values of the parameters:
   $\mu_1, \mu_2, \cdots, \mu_k; \pi_1, \pi_2, \cdots, \pi_k and \sum$, and substitute these values in the above equation to get the value of $\delta_k(x)$ and classify to he class with the largest value of $\delta_k(x)$. It shows that $\delta_k(x)$ is a linear function of x.

   - π_k is the prior probability that an obervation belongs to the k^{th} class. For instance, we have 3 classes (0,1,2) and the number of observation in class 0 is 40, class 1 is 30 and class 2 is 30. Then, $\pi_0 = 0.4$, $\pi_1 = 0.3$ and $\pi_2 = 0.2$.
   - The estimate for μ_k is the average of all training observation from the k^{th} class.
   - Bayes theorem in simple terms predicts the class of an observation as follows:
     $P(Y = y|X = x) = \frac{\pi_k.f_k(x)}{\sum_l^k.\pi_l.f_l(x)}$

f)

K Nearest Neighbor is an algorithm that sorts incoming information or instances based on similarities between them and all previously stored examples.
A data point is often classified using the classification of its neighbors.
The " k " parameter is dependent on feature similarity. For greater precision, a procedure known as parameter tuning entails selecting the appropriate value of " k ".
As a rule of thumb, we selects odd numbers as k.
KNN may perform better than linear models when there high bias and the size of the dataset is small.
Advantages:

1. Simple to implement
2. Flexible to feature choices
3. Naturally handles multi-class cases
4. Few hyperparameters to tune

Disadvantages:

1. Need to determine the value of parameter K (number of nearest neighbors)
2. Computation cost is quite high because we need to compute the distance of each query instance to all training samples.
3. Large computation cost during runtime if sample size is large.

Liner Regression vs KNN :

1. KNN is a non -parametric model, whereas Linear Regression is a parametric model.
2. KNN is slow in real time as it have to keep track of all training data and find the neighbor nodes, whereas Linear Regression can easily extract output from the tuned θ coefficients.
3. KNN is better than linear regression when the data have high SNR.

Logistic Regression vs KNN :

1. KNN is a non-parametric model, where Logistic Regression is a parametric model.
2. KNN is comparatively slower than Logistic Regression.
3. KNN supports non-linear solutions where Logistic Regression supports only linear solutions.
4. Logistic Regression can derive confidence level, whereas KNN can only output the labels.

Problem 2

a)

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_wine
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.utils import resample
from scipy import stats
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import log_loss
from sklearn.preprocessing import StandardScaler
```

In [2]:
```
wine = load_wine()
```

In [3]:
```
wine_data = pd.DataFrame(data= np.c_[wine['data'], wine['target']],columns=
```

In [4]:
```
wine_data
```

Out[4]:

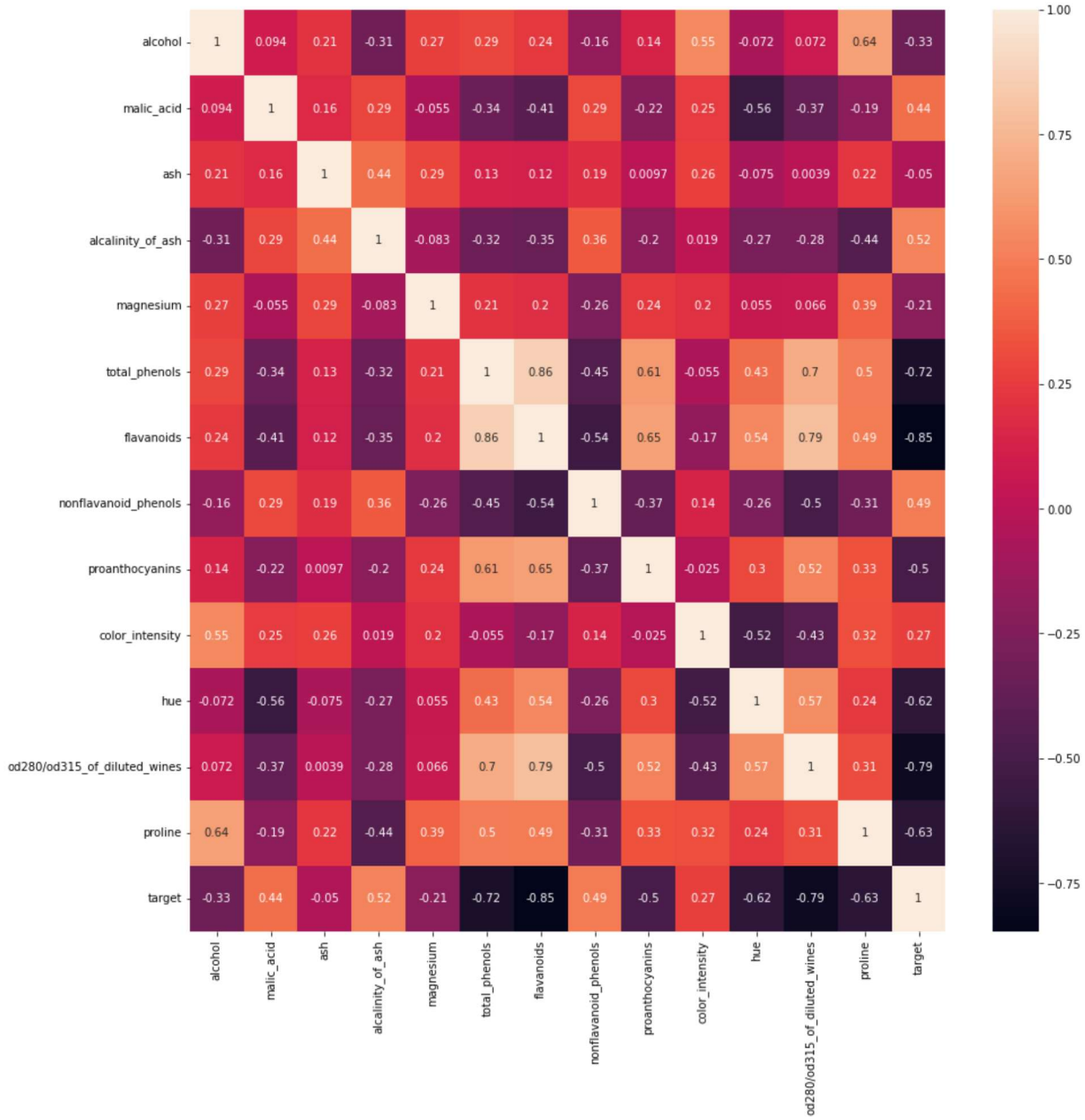
	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavano
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	
...	...	...	...	...	...	...	...	
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	

178 rows × 14 columns

b)

In [5]:
```
plt.figure(figsize=(15,15))
sns.heatmap(wine_data.corr(),annot = True)
```

Out[5]: <AxesSubplot:>



Observation:

The above figure shows the correlation between each feature. The features having value > 0.5 are the most likely to be useful for predciting target.

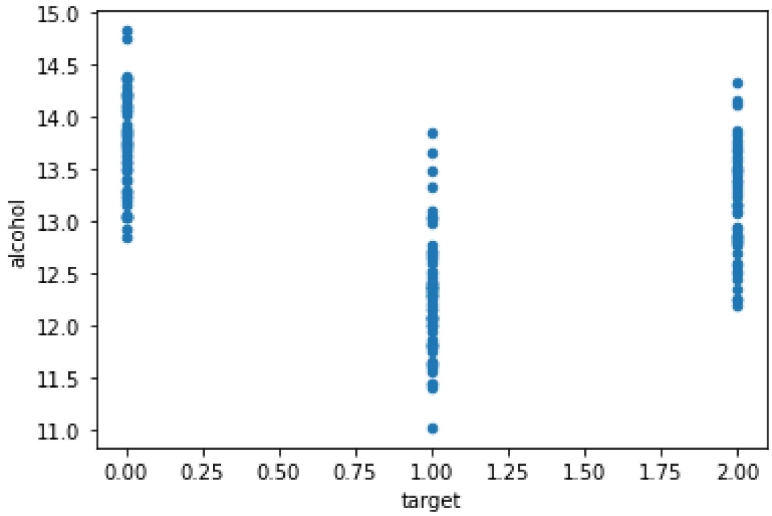
These values are namely:

1. alcalinity_of_ash
2. total_phenols
3. flavanoids

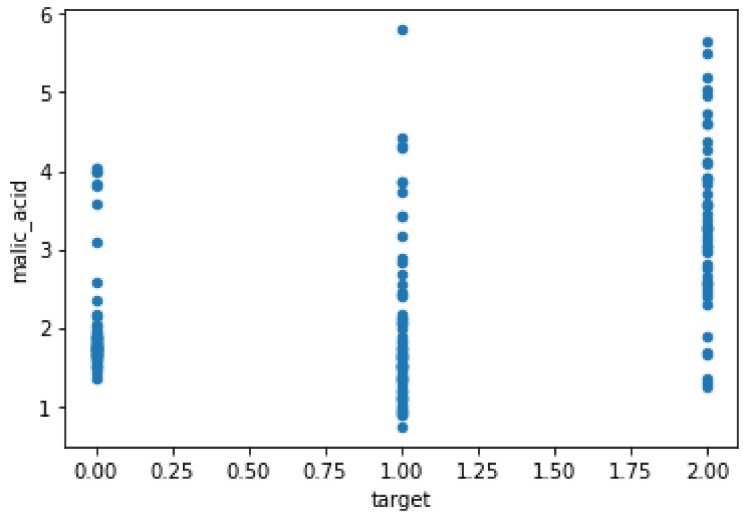
4. proanthocyanins
5. hue
6. od280/od315_of_diluted_wines
7. proline

In [6]:
```
X = wine_data.drop("target", axis=1)
y = wine_data[["target"]]
```

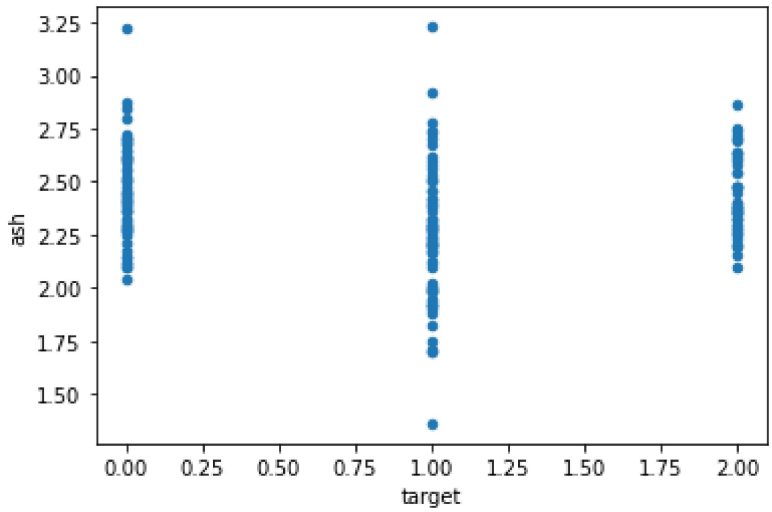
In [7]:
```
ax = wine_data.plot.scatter(x='target', y='alcohol')
```



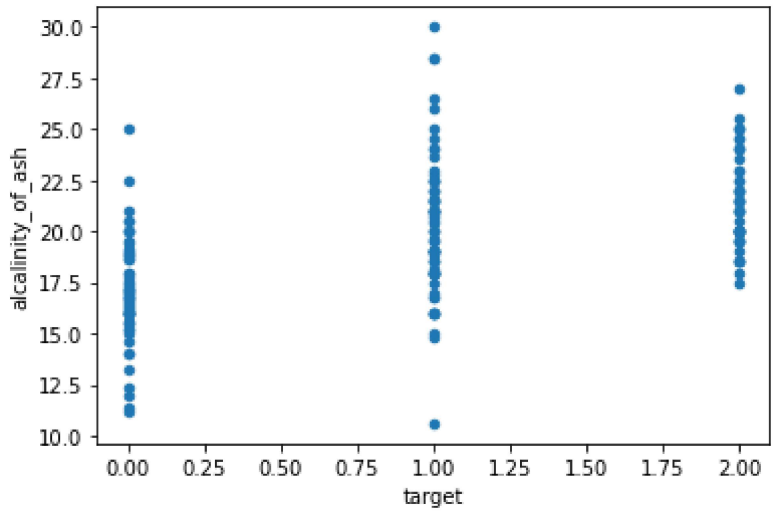
In [8]:
```
ax = wine_data.plot.scatter(x='target', y='malic_acid')
```



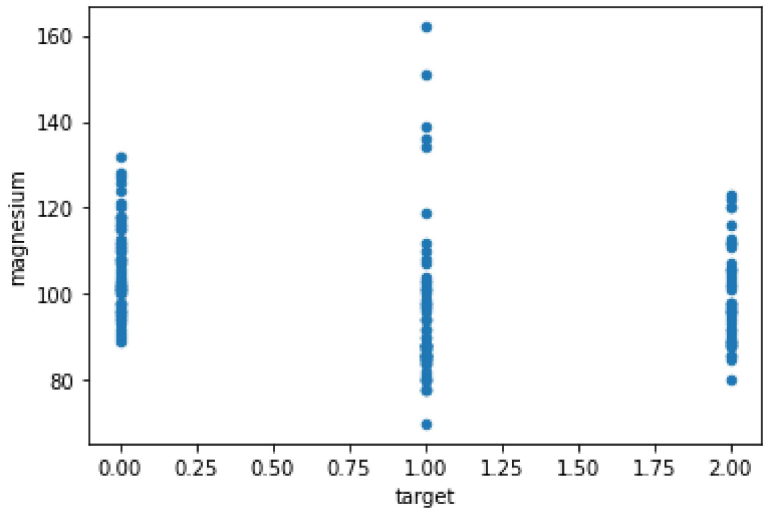
In [9]:
```
1 ax = wine_data.plot.scatter(x='target', y='ash')
```



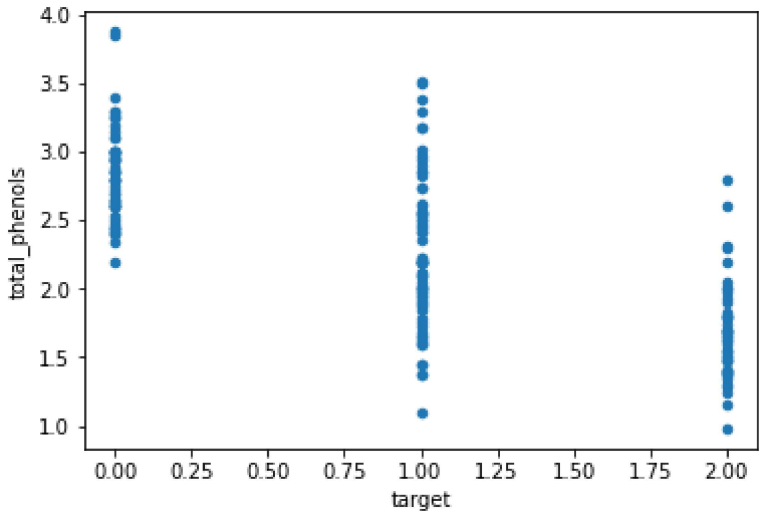
In [10]:
```
1 ax = wine_data.plot.scatter(x='target', y='alcalinity_of_ash')
```



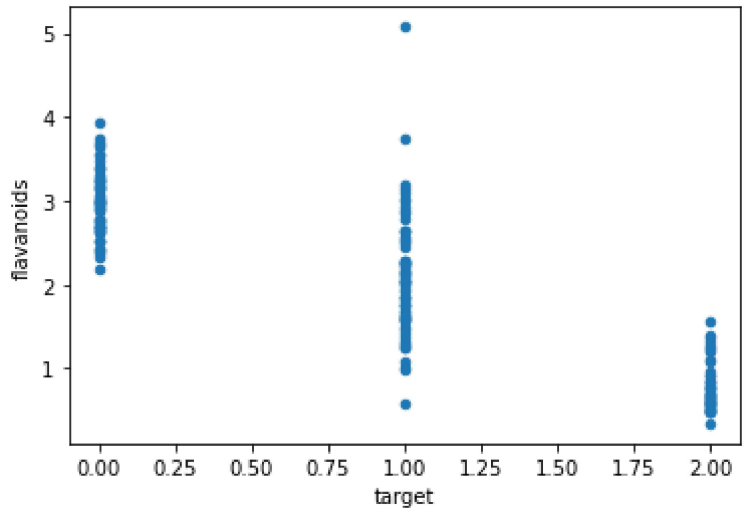
In [11]:
```
1 ax = wine_data.plot.scatter(x='target', y='magnesium')
```



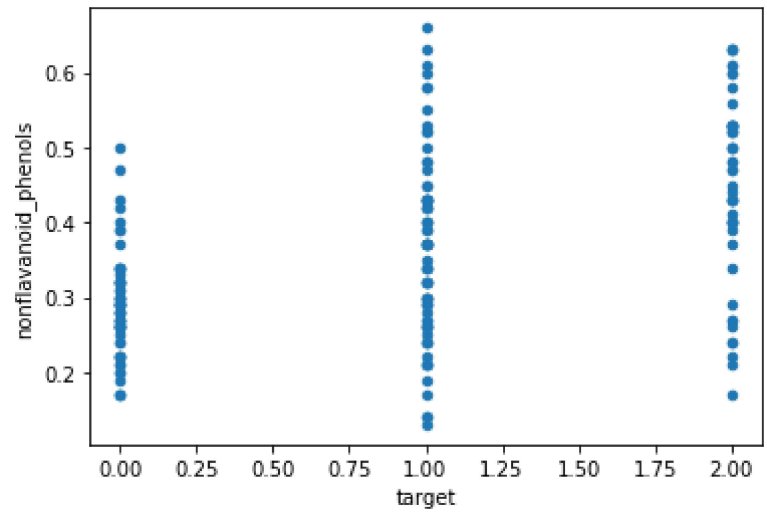
In [12]:
```
1 ax = wine_data.plot.scatter(x='target', y='total_phenols')
```



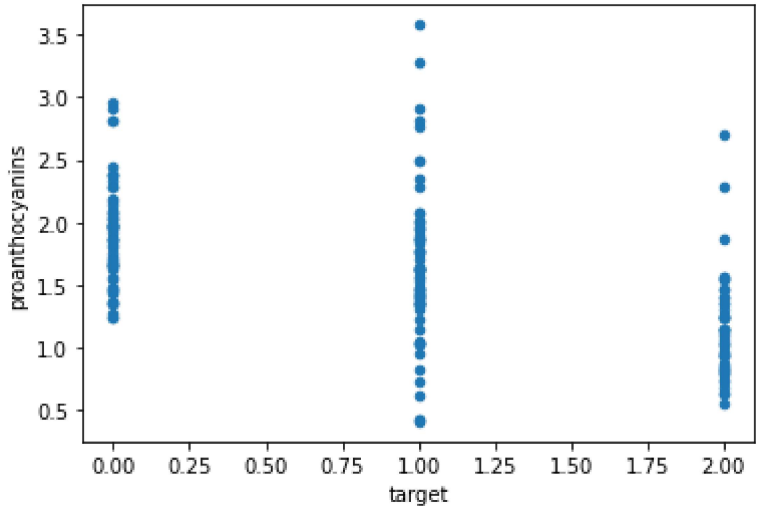
In [13]:
```
1 ax = wine_data.plot.scatter(x='target', y='flavanoids')
```



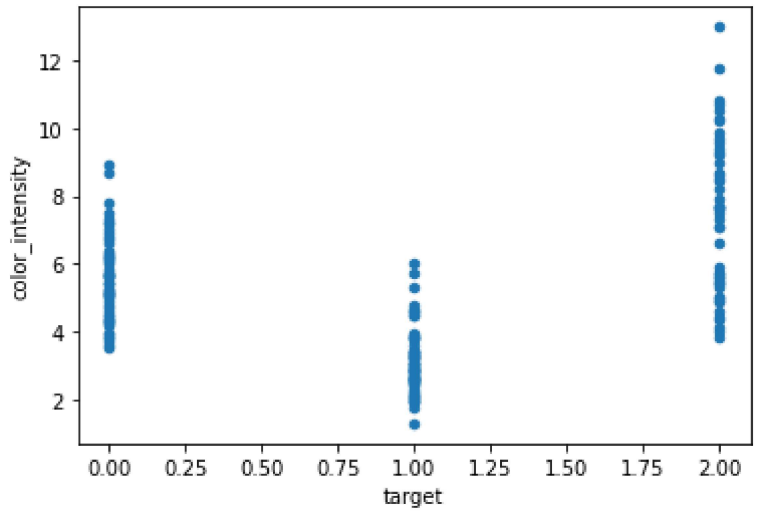
In [14]:
```
1 ax = wine_data.plot.scatter(x='target', y='nonflavanoid_phenols')
```



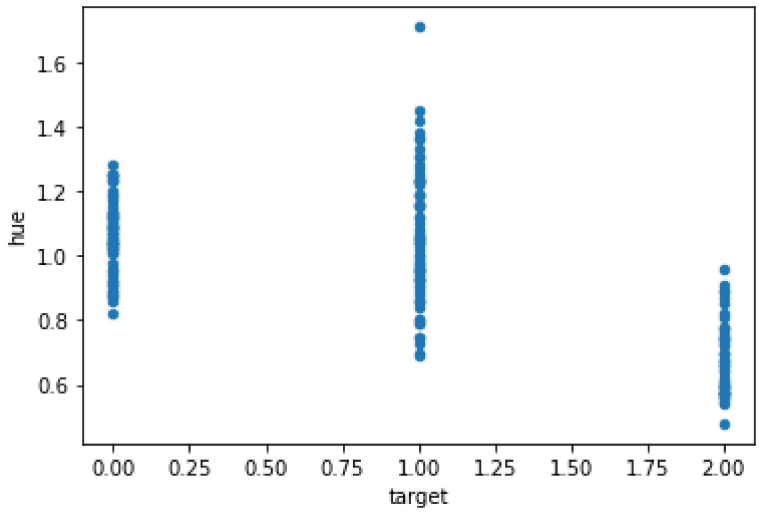
In [15]:
```
1 ax = wine_data.plot.scatter(x='target', y='proanthocyanins')
```



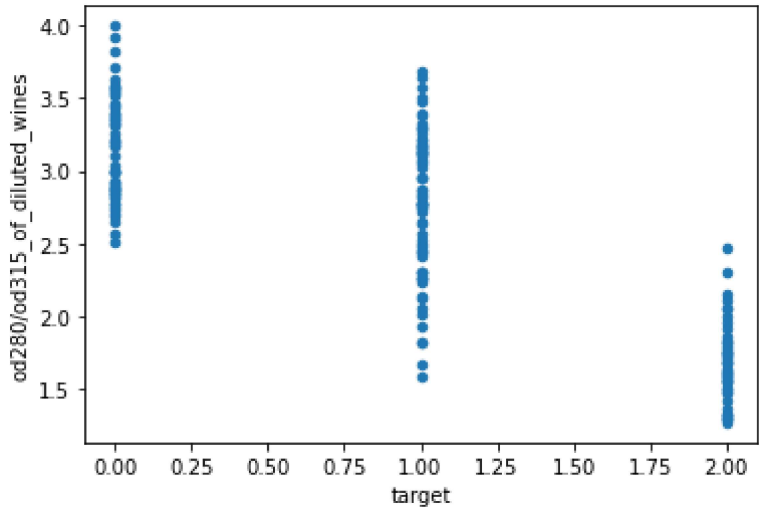
In [16]:
```
1 ax = wine_data.plot.scatter(x='target', y='color_intensity')
```



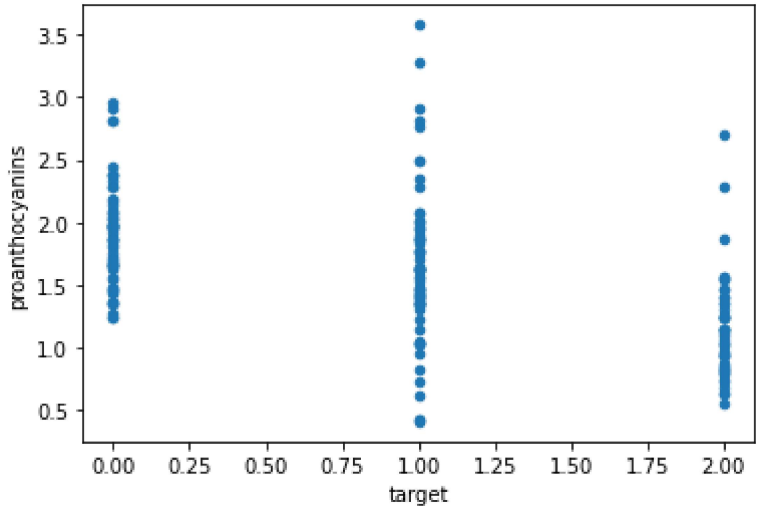
In [17]:
```
1 ax = wine_data.plot.scatter(x='target', y='hue')
```



In [18]:
```
1 ax = wine_data.plot.scatter(x='target', y='od280/od315_of_diluted_wines')
```

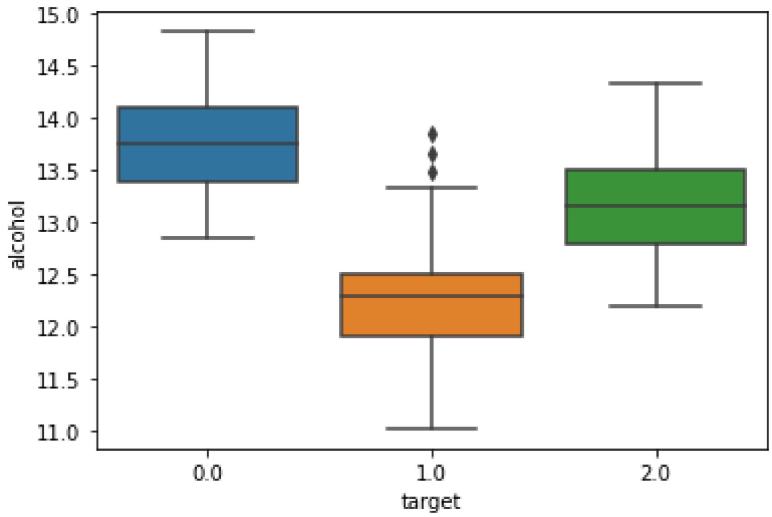


In [19]:
```
1 ax = wine_data.plot.scatter(x='target', y='proanthocyanins')
```



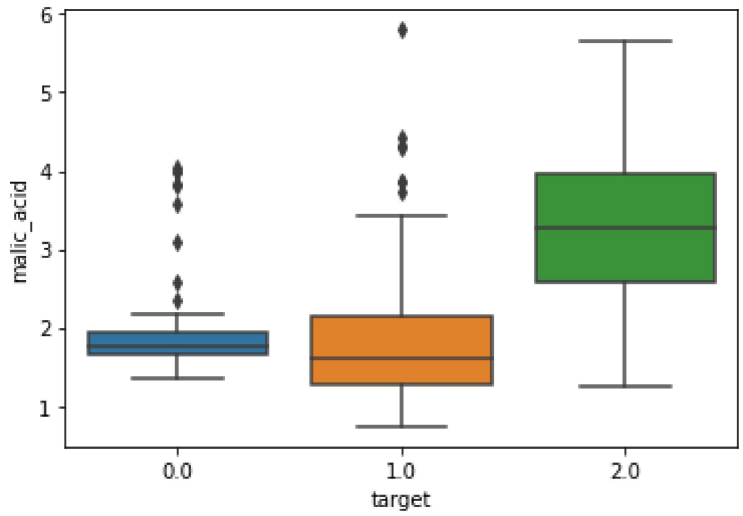
In [20]:
```
1 sns.boxplot(data=wine_data, x="target", y="alcohol")
```

Out[20]: <AxesSubplot:xlabel='target', ylabel='alcohol'>



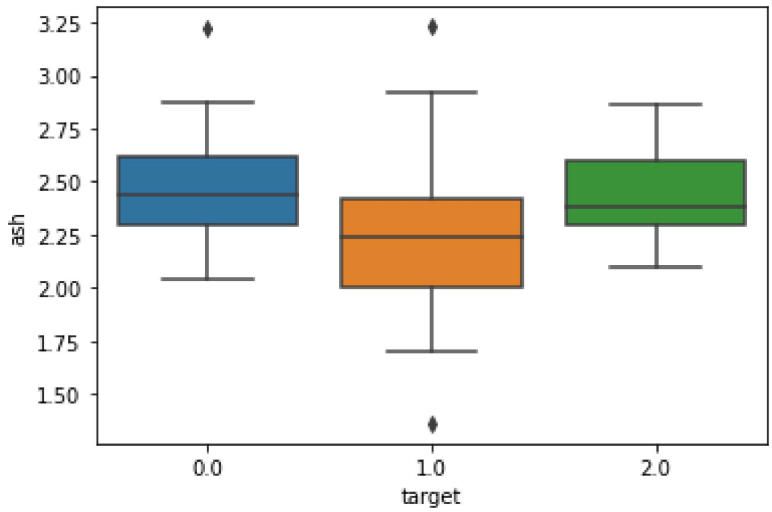
In [21]:
```
1 sns.boxplot(data=wine_data, x="target", y="malic_acid")
```

Out[21]: <AxesSubplot:xlabel='target', ylabel='malic_acid'>



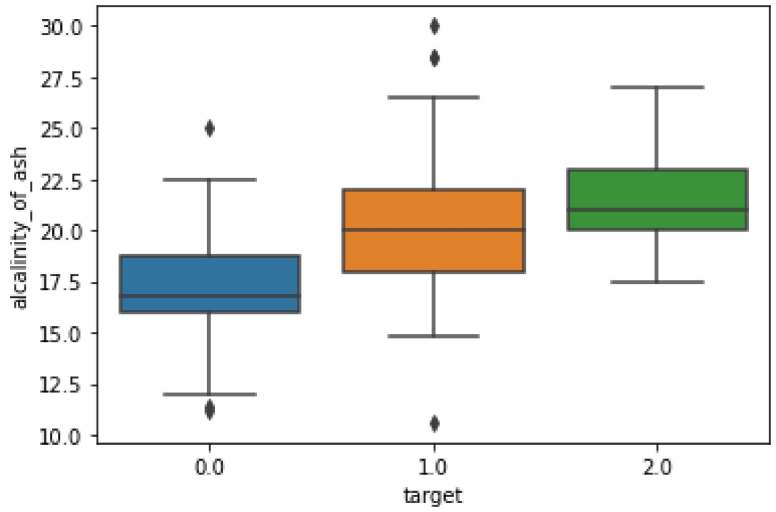
In [22]:
```
1 sns.boxplot(data=wine_data, x="target", y="ash")
```

Out[22]: <AxesSubplot:xlabel='target', ylabel='ash'>



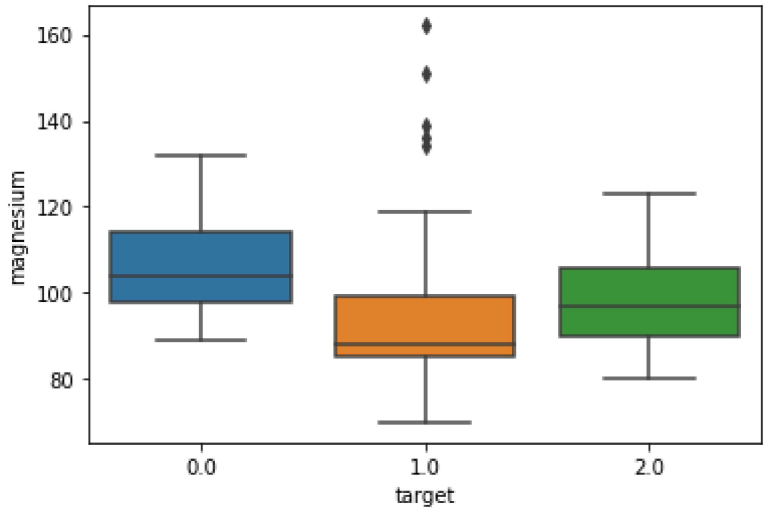
In [23]:
```
1 sns.boxplot(data=wine_data, x="target", y="alcalinity_of_ash")
```

Out[23]: <AxesSubplot:xlabel='target', ylabel='alcalinity_of_ash'>



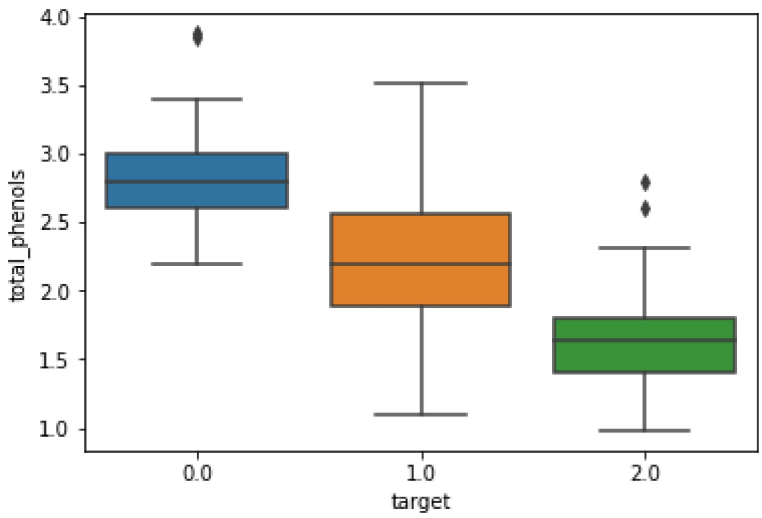
In [24]:
```
1 sns.boxplot(data=wine_data, x="target", y="magnesium")
```

Out[24]: <AxesSubplot:xlabel='target', ylabel='magnesium'>



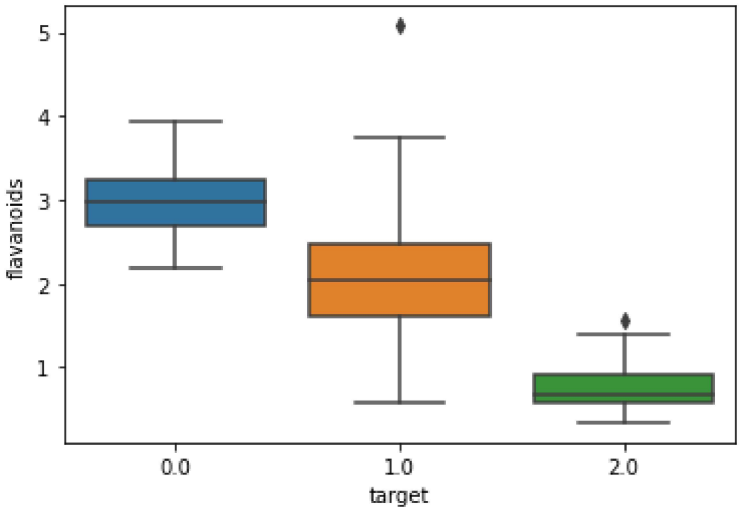
In [25]:
```
1 sns.boxplot(data=wine_data, x="target", y="total_phenols")
```

Out[25]: <AxesSubplot:xlabel='target', ylabel='total_phenols'>



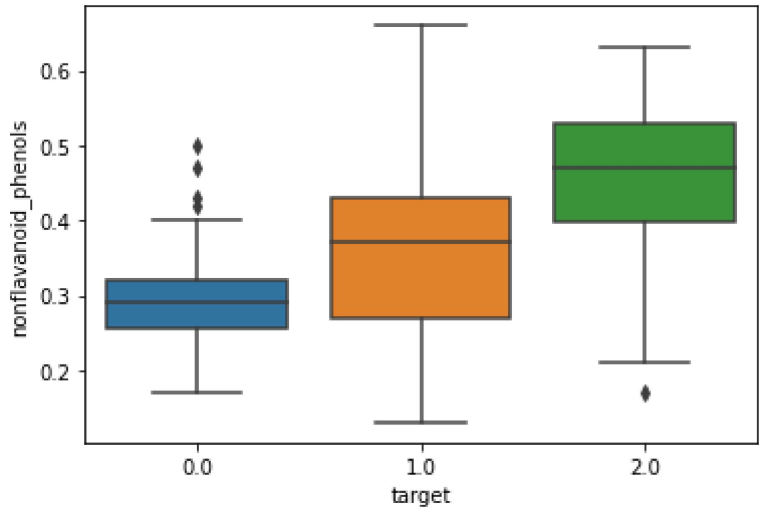
In [26]:
```
1 sns.boxplot(data=wine_data, x="target", y="flavanoids")
```

Out[26]: <AxesSubplot:xlabel='target', ylabel='flavanoids'>



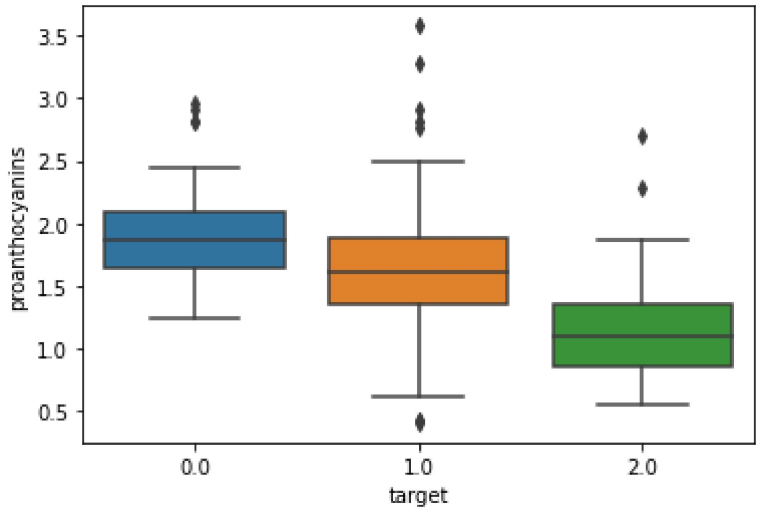
In [27]:
```
1 sns.boxplot(data=wine_data, x="target", y="nonflavanoid_phenols")
```

Out[27]: <AxesSubplot:xlabel='target', ylabel='nonflavanoid_phenols'>



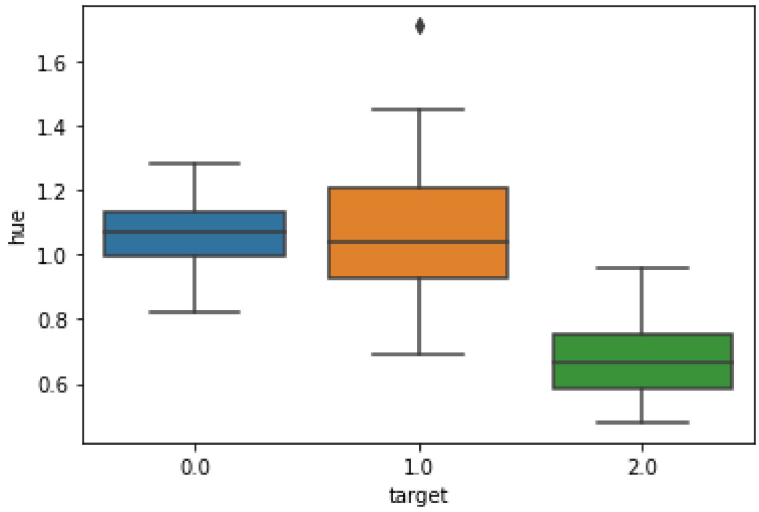
In [28]:
```
1 sns.boxplot(data=wine_data, x="target", y="proanthocyanins")
```

Out[28]: <AxesSubplot:xlabel='target', ylabel='proanthocyanins'>



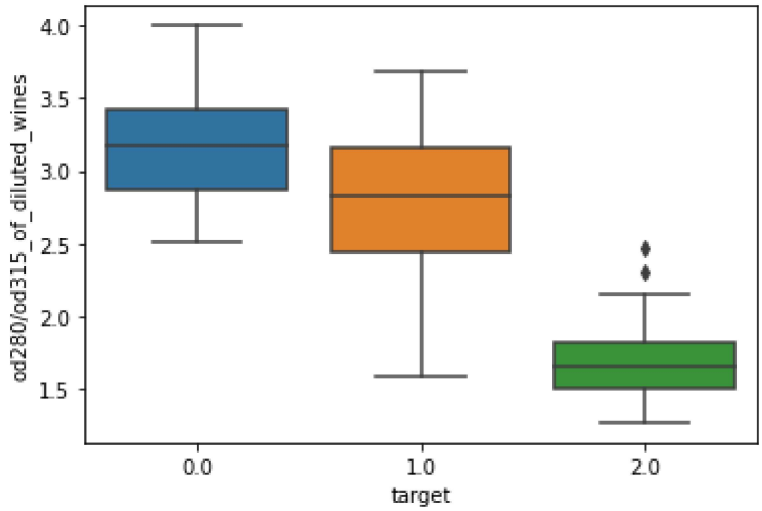
In [29]:
```
1 sns.boxplot(data=wine_data, x="target", y="hue")
```

Out[29]: <AxesSubplot:xlabel='target', ylabel='hue'>



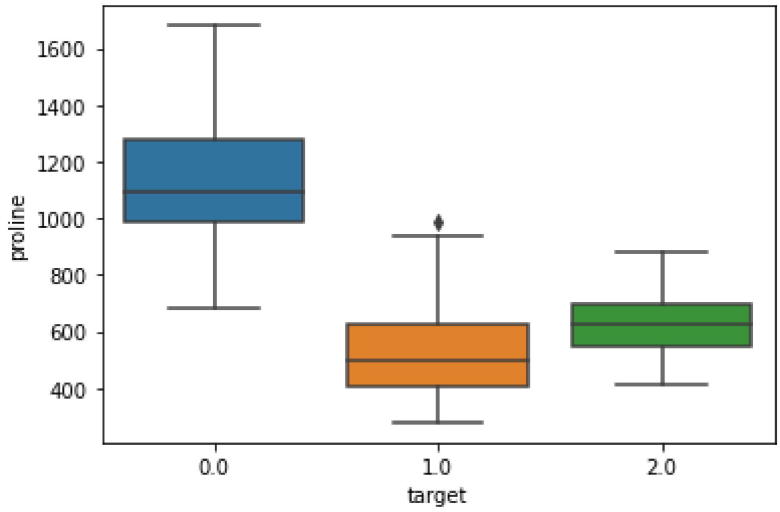
In [30]:
```
1 sns.boxplot(data=wine_data, x="target", y="od280/od315_of_diluted_wines")
```

Out[30]: <AxesSubplot:xlabel='target', ylabel='od280/od315_of_diluted_wines'>



In [31]:
```
1 sns.boxplot(data=wine_data, x="target", y="proline")
```

Out[31]: <AxesSubplot:xlabel='target', ylabel='proline'>



Observation:
From the above scatterplots and boxplots we can get an idea about the dataset, how well it is spread across the classes.
It also enables to identify if the data is skewed towards one or more classes.
We can also deduce that the plots that are more heavily biased towards one or more classes will have the most impact on the classification model.
The following seem most likely to be useful in predicting the target:

1. alcohol
2. total_phenols
3. flavanoids
4. od280/od315_of_diluted_wines
5. hue
6. proanthocyanins

c)

In [32]:
```
1 # Here we are using random_state = 101, to ensure that same results are obta
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, r
```

d)

In [33]:
```
# Scaling the data to get minimal error
scaler = StandardScaler()
scaled = scaler.fit_transform(X_train)
print(scaled)
```

```
[[ 0.25468356 -0.44777631 -0.74860446 ...  0.39030345  1.05926423
   0.98740899]
 [-1.43732049 -0.14038771  1.41273406 ... -0.04120613 -0.50214256
  -0.36900848]
 [-1.10865783 -0.17655107 -2.33358604 ...  1.33962453  0.456616
  -0.09772499]
 ...
 [-0.32960561 -0.42065379 -0.5324706  ...  0.86496399  0.70315391
  -0.08495871]
 [ 1.37457113 -0.71900154 -0.1002029  ...  0.90811495  0.26486429
   1.7214702 ]
 [-0.63392289 -0.68283818 -0.5324706  ...  0.86496399  0.00462982
   0.62676152]]
```

In [34]:
```
scaled1 = scaler.transform(X_test)
print(scaled1)
```

```
[[ 7.20931909e-02 -5.65307244e-01  7.28310198e-01 -4.27046962e-01
  -1.26422733e-01  2.32910361e-01  3.97583835e-01 -5.29922593e-01
  -2.87323159e-01 -3.49881756e-01  6.92360161e-01 -1.59728790e-01
   1.16294536e+00]
 [ 8.38972722e-01 -4.02572103e-01  4.38863337e-02 -6.59498016e-01
   2.86267568e-01  1.85752655e-01  6.56377827e-01  4.85899947e-01
   6.09191805e-01 -5.27973345e-01  1.16702070e+00  3.33347040e-01
   7.95914754e-01]
 [ 4.00755847e-01  8.72186500e-01  1.15930951e-01  5.89926398e-01
  -5.39113033e-01 -5.84489880e-01 -1.26466988e+00  7.20320534e-01
  -6.08715315e-01  1.46144002e+00 -1.76724445e+00 -1.37872181e+00
  -2.89219221e-01]
 [ 1.08242654e+00 -3.48327056e-01  8.72399432e-01 -1.29873841e+00
   7.99224174e-02  1.49044919e+00  1.52234234e+00 -1.46760494e+00
   1.52476635e-01  1.62674526e-01 -3.43262833e-01  1.27840905e+00
   1.13102966e+00]
 [ 2.91201628e-01  9.26431547e-01 -2.44292135e-01 -2.81765054e-01
  -1.26422733e-01 -7.88839940e-01 -1.19499457e+00  1.97056366e+00
   4.40038038e-01  2.37361645e+00 -1.72409349e+00 -1.52938387e+00
```

In [35]:
```
# Logistic Regression on data and predicting the target of test set
logres = LogisticRegression(max_iter = 10000).fit(X_train, y_train.values.ra
y_pred = logres.predict(X_test)
# Here max_iter = 10000 to make sure of convergence.
```

In [36]:
```
# Classification report for Logistic Regression
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        14
         1.0       1.00      0.92      0.96        13
         2.0       0.90      1.00      0.95         9

    accuracy                           0.97        36
   macro avg       0.97      0.97      0.97        36
weighted avg       0.98      0.97      0.97        36
```

In [37]:
```
# Zero-One Test Error:
zero_error = 1 - (np.array(y_test["target"]) == y_pred).sum()/len(y_test)
print("Zero-One Test Error for Logistic Regression is:",round(zero_error,5))
```

```
Zero-One Test Error for Logistic Regression is: 0.02778
```

In [38]:
```
# Log-loss Test Error:
log_error = log_loss(y_test,logres.predict_log_proba(X_test))
print("Log-loss Test Error for Logistic Regression is:",round(log_error,5))
```

```
Log-loss Test Error for Logistic Regression is: 1.09861
```

e)

In [39]:
```
# Linear Discriminant Analysis on data and predicting the target of test set
lda = LinearDiscriminantAnalysis().fit(X_train, y_train.values.ravel())
pred_lda = lda.predict(X_test)
```

In [40]:
```
# Classification report for Linear Discriminant Analysis
print(classification_report(y_test,pred_lda))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        14
         1.0       1.00      1.00      1.00        13
         2.0       1.00      1.00      1.00         9

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```

In [41]:
```
# Linear Discriminant Analysis for Zero-One Test Error
lda_zero_error = 1 - (np.array(y_test["target"]) == pred_lda).sum()/len(y_te
print("Zero-One Test Error for Linear Discriminant Analysis is:",round(lda_z
```

```
Zero-One Test Error for Linear Discriminant Analysis is: 0.0
```

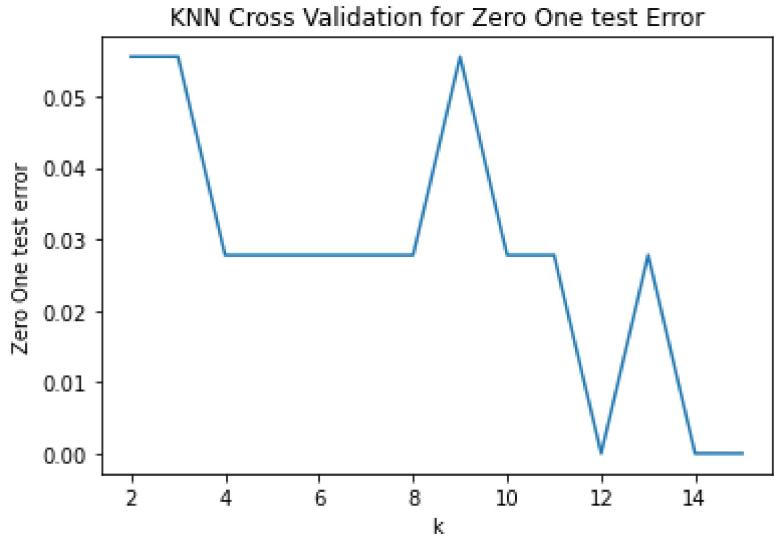
In [42]:
```
# Linear Discriminant Analysis for Log Loss Test Error
lda_log_error = log_loss(y_test,lda.predict_log_proba(X_test))
print("Log-loss Test Error for Linear Discriminant Analysis is:",round(lda_l
```

```
Log-loss Test Error for Linear Discriminant Analysis is: 1.09861
```

f)

In [43]:
```

# Using KNN on the data and predicting the dataset.
# Using Cross Validation by iterating the values of "k" from (2,15) and calc
zero_test_error = []
for k in range(2,16):
    ngh = KNeighborsClassifier(n_neighbors = k).fit(scaled, y_train.values.r

    y_pred = ngh.predict(scaled1)

    test_error1 = 1 - (np.array(y_test["target"]) == y_pred).sum()/len(y_tes

    zero_test_error.append(test_error1)
```

In [44]:
```
# Diagram shows the value of k where error is the lowest
plt.plot(list(range(2,16)), zero_test_error)
plt.ylabel("Zero One test error")
plt.xlabel("k")
plt.title("KNN Cross Validation for Zero One test Error")
plt.show()
```



Observation:

The k value is 2 having zero test error of 0.06
The k value is 4 having zero test error of 0.028
The k value is 6 having zero test error of 0.028
The k value is 8 having zero test error of 0.028
The k value is 10 having zero test error of 0.028

The k value is 12 having zero test error of 0 **(Lowest Error for the K value)**
The k value is 14 having zero test error of 0 **(Lowest Error for the K value)**

Problem 3

a)

Given:
$x_1, x_2, \ldots, x_n$ are the training set observation
$z_1, z_2, \ldots, z_n$ are set of z observation
Solution:
The probability of getting of x_1 is $P(X)$ = 1/n

For $z_1 \neq x_1$ is $P(X')$ = 1 - (The probability of getting x_1)

The probability for $z_1 \neq x_1$ is $P(X')$ = 1 - (1/n)

Therefore, Probability of $z_1 \neq x_1$ is (n-1)/n.

b)

Given:
$x_1, x_2, \ldots, x_n$ are the training set observation
$z_1, z_2, \ldots, z_n$ are set of z observation
Solution:
Similarly as done in the above question.
Since, the sample selection is independent, we will get the same probability as that of Q3. a)
The probability of getting of x_1 is $P(X)$ = 1/n

For $z_2 \neq x_1$ is $P(X')$ = 1 - (The probability of getting x_1)

The probability for $z_2 \neq x_1$ is $P(X')$ = 1 - (1/n)

c)

Solution:

The probability for $z_1 \neq x_1$ is $P(X')$ = 1 - (1/n)

The probability for $z_2 \neq x_1$ is $P(X')$ = 1 - (1/n)

The probability for $z_2 \neq x_1$ and $z_1 \neq x_1$ = [1-(1/n)].[1-(1/n)]

Now, we know that for n terms we get that $[1 - (1/n)]^n$.

Therfore, the probability of getting x_1 in S = 1 - $[1 - (1/n)]^n$ Given that n = 100

The probability of getting x_1 in S = 1 - $((n - 1)/n)^n$

The probability of getting x_1 in S = 1 - $((100 - 1)/100)^{100}$

The probability of getting x_1 in S = 1 - $(99/100)^{100}$

The probability of getting x_1 in S = 1 - 0.366032

The probability of getting x_1 in S = 0.633967.

d)

From 3-c, we know that

The probability of getting x_1 in S = 1 - $[1 - (1/n)]^n$

Here, we have n observations $x_1, x_2, \ldots, x_n$ are the training set observation, therefore for all the n observation the expected number of distinct data point would be:

n * The probability of getting x_1 in S = n.[1 - $[1 - (1/n)]^n$]

Take x = 1/n, we get the probability as $n.(1 - x)^{1/x} \cdots (1)$

Now, $y = \lim_{x \to 0}(1 - x)^{1/x}$

Taking log_e on both sides,

$ln(y) = ln(\lim_{x \to 0}(1 - x)^{1/x})$

Using the exponent properties of log,

$ln(y) = (1/x).ln(\lim_{x \to 0}(1 - x))$

Taking derivation using L' Hopitals Rule,

$ln(y) = \lim_{x \to 0}(1/(1 - x))$

Substituting the value of x = 0,

$ln(y) = 1/(1 - 0)$

$ln(y) = 1$

Taking inverse of the log,

Therefore, $y = e^1$

Now, substituting value of y = e in the above equation (1),

No. of distinct points = n.(1-(1/e))

= n.(1-(1/2.78))

The expected number of distinct data points in the set S = n.(0.632)

Problem 4

a)

In [45]:
```
df = pd.read_csv(r"C:\Users\ayush\Downloads\amazon_bs_20102020.csv")
```

In [46]:
```
df.head()
```

Out[46]:

	Year	Rank	Book_Title	Author	Rating	Num_Customers_Rated	Price
0	2010	1	The Girl Who Kicked the Hornet's Nest (Millenn...	Stieg Larsson	4.7	8475	17.24
1	2010	2	The Girl with the Dragon Tattoo (Millennium Se...	Stieg Larsson	4.4	11516	9.99
2	2010	3	Decision Points	George W. Bush	4.6	2201	17.80
3	2010	4	The Help	Kathryn Stockett	4.8	14772	14.97
4	2010	5	The Girl Who Played with Fire (Millennium Series)	Stieg Larsson	4.7	7949	0.02

In [47]:
```
# Dataframe of Price and Rating
df1 = df[["Price","Rating"]]
```

In [48]:
```
df1.head()
```

Out[48]:

	Price	Rating
0	17.24	4.7
1	9.99	4.4
2	17.80	4.6
3	14.97	4.8
4	0.02	4.7

In [49]:
```
#
print("The population mean of Price is:",np.mean(df["Price"]))
```

```
The population mean of Price is: 10.850950639853718
```

b)

In [50]:
```
print("The standard error is:",np.std(df1['Price'])/np.sqrt(len(df1)))
```

```
The standard error is: 0.35858300263084675
```

Obervation:
The standard error : 0.35858
The standard error is the measure of the difference in mean if we were to conduct the experiment on different values.
The standard error os 0.35858 which means that if we take the same experiment but with different values the new mean will vary between mean $\pm$ 0.35858.
Therefore the new mean will range between (10.5, 11.1)

Finding mean of 1000 resampled values

In [51]:
```
mean_dict = []
for i in range(1000):
    a = resample(df["Price"])
    mean_dict.append(np.mean(a))
```

Finding the mean value of all the mean values

In [52]:
```
mean_sample = sum(mean_dict)/len(mean_dict)
print("The mean of the sample is:",mean_sample)
```

```
The mean of the sample is: 10.841292212065781
```

c)

In [53]:
```
mean_std_error = np.std(np.array(mean_dict))
print("The standard error:",mean_std_error)
```

```
The standard error: 0.36167658001106606
```

Observation:
We conclude that the standard error of original dataset and the sample set is approximately the same.

d)

In [54]:
```
A = mean_sample - 2*mean_std_error
B = mean_sample + 2*mean_std_error
print("The 95% confidence interval of mean price is:", (A, B))
```

```
The 95% confidence interval of mean price is: (10.11793905204365, 11.564645372087913)
```

In [55]:
```
stats.norm.interval(0.95, loc = mean_sample, scale = mean_std_error)
```

Out[55]:
```
(10.132419141192472, 11.55016528293909)
```

Observation:
The confidence interval calculated using the formula and the one calculated using scipy are approximately the same

e)

In [56]:
```
print("The mean of first quartile is:",np.quantile(df["Price"],0.25))
```

```
The mean of first quartile is: 2.75
```

f)

In [57]:
```
first_quart = []
for i in range(1000):
    x = resample(df["Price"])
    first_quart.append(np.quantile(x,0.25))
```

In [58]:
```
print("The standard error of first quartile is:",np.std(first_quart))
```

```
The standard error of first quartile is: 0.38192345567371216
```

Obervation:
Therefore, we come to the conclusion that there is approximately ~18% variation in the quantile

values of the sampled sets.
Hence, we are sampling with replacement there can be redundant values in the sets which results in the variation in the quatile values of the sets.

g)

In [59]:
```
from scipy.stats import linregress
```

In [60]:
```
intercept = []
slope = []

for i in range(1000):
    d = resample(df1)
    M = d['Rating']
    n = d['Price']

    LR = LinearRegression()
    LR.fit(M.values.reshape(-1,1),n.values.reshape(-1,1))
    intercept.append(LR.intercept_)
    slope.append(LR.coef_[0])
```

In [61]:
```
print("Standard error:", np.std(intercept))
```

```
Standard error: 7.180330520092305
```

In [62]:
```
np.std(slope)
```

Out[62]:
```
1.5302083188882474
```

In [63]:
```
M = df1['Rating']
n = df1['Price']

slope_1 = linregress(M,n).stderr
intercept_1 = linregress(M,n).intercept_stderr
```

In [64]:
```
slope_1
```

Out[64]:
```
1.6302666162338253
```

In [65]:
```
intercept_1
```

Out[65]:
```
7.558746805421322
```

Observation:
We can see that the linregress standard error is slightly greater than the bootstrap sampled sets standard error for both slope (β_0) and intercept (β_1).