Assignment: Mushrooms Data set

Mohammad Al-Ajlouni C0849924

1. Definition

1.1 Overview

This data set includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525). Each species is identified as definitely edible, poisonous, or of unknown edibility and not recommended. This latter class was combined with the poisonous one. The Guide clearly states that there is no simple rule for determining the edibility of a mushroom; no rule like ``leaflets three, let it be'' for Poisonous Oak and Ivy.

We have 8124 rows and 22 column, where this is the description of each column:

1. cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s
2. cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s
3. cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r, pink=p,purple=u,red=e,white=w,yellow=y
4. bruises?: bruises=t,no=f
5. odor: almond=a,anise=l,creosote=c,fishy=y,foul=f, musty=m,none=n,pungent=p,spicy=s
6. gill-attachment: attached=a,descending=d,free=f,notched=n
7. gill-spacing: close=c,crowded=w,distant=d
8. gill-size: broad=b,narrow=n
9. gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e, white=w,yellow=y
10. stalk-shape: enlarging=e,tapering=t
11. stalk-root: bulbous=b,club=c,cup=u,equal=e, rhizomorphs=z,rooted=r,missing=?

12. stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s
13. stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s
14. stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
15. stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o, pink=p,red=e,white=w,yellow=y
16. veil-type: partial=p,universal=u
17. veil-color: brown=n,orange=o,white=w,yellow=y
18. ring-number: none=n,one=o,two=t
19. ring-type: cobwebby=c,evanescent=e,flaring=f,large=l, none=n,pendant=p,sheathing=s,zone=z
20. spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r, orange=o,purple=u,white=w,yellow=y
21. population: abundant=a,clustered=c,numerous=n, scattered=s,several=v,solitary=y
22. habitat: grasses=g,leaves=l,meadows=m,paths=p, urban=u,waste=w,woods=d

1.2 Problem Statement

We will be comparing two very different machine learning models on the Mushroom Classification Dataset for the task of predicting whether a given mushroom is **poisonous** or **edible**.

The first model will be Logistic Regression without any parameter tuning, and the second model will be K-Means Clustering.

As usual, for the Supervised Learning Algorithm (Logistic Regression), we will simply train the model on 80% of the mushroom data, and then test it's performance on the remaining 20%.

And as for the Unsupervised Learning Algorithm (K-Means Clustering): I've found that if we cluster the data (with it's labels removed) into two different clusters, then one cluster ends up holding most of the **poisonous** mushrooms, while the other cluster ends up holding most of the **edible** mushrooms. So, to build our binary classifier, we will cluster that same 80% of mushroom data

mentioned above into two clusters, and then classify the remaining 20% of mushrooms as **poisonous** or **edible** depending on which cluster they belong to. And at the end, we will compare the performances of the two algorithms on that test data to see who comes out on top.

2. Coding and Analyses

Looking at the data, its categorized by characters in each column as we can see in the below screen shot.

```
orig.head()
```

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a

5 rows × 23 columns

By dividing our data into 'predictors,' X and 'labels,' y:

Now before we encode each of our categorical variables in X & y with numbers so that our learning algorithms can work with them, we will first do a bit of exploration.

```
#The 'class' column contains our labels.
#It tells us whether the mushroom is 'poisonous' or 'edible'.
X = orig.drop(['class'], axis=1)
y = orig['class']
```

Let us look at the values contained within each of X's attributes, so we can get a better picture of the data we're working with:

```
for attr in X.columns:
    print('\n*', attr, '*')
    print(X[attr].value_counts())
```

```
* cap-shape *
x    3656
f    3152
k     828
b     452
s      32
c       4
Name: cap-shape, dtype: int64

* cap-surface *
y    3244
s    2556
f    2320
g       4
Name: cap-surface, dtype: int64

* cap-color *
n    2284
```

```
* veil-type *
p    8124
Name: veil-type, dtype: int64
```

```
* stalk-root *
b    3776
?    2480
e    1120
c     556
r     192
Name: stalk-root, dtype: int64
```

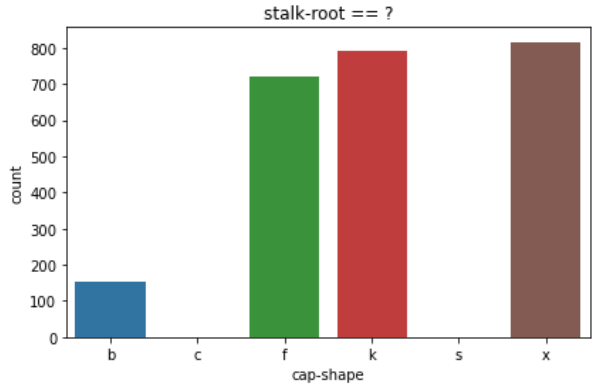
Two things to note here:

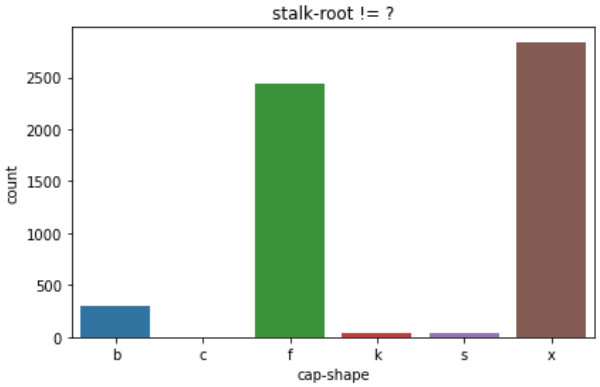
First, the veil-type variable has only one value, '**p**', meaning, every mushroom has the same veil-type. And because, every mushroom has that same veil-type: that column doesn't tell us anything useful - so we can drop that column.

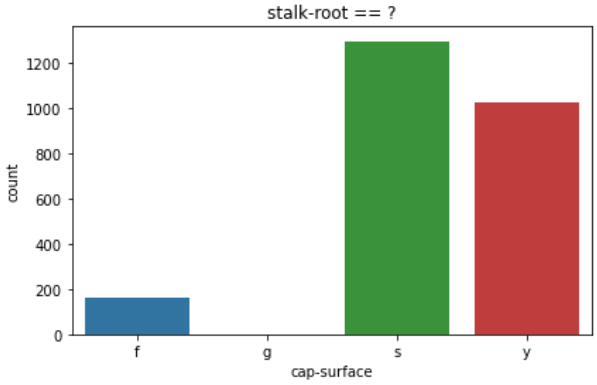
```
X.drop(['veil-type'], axis=1, inplace=True)
```

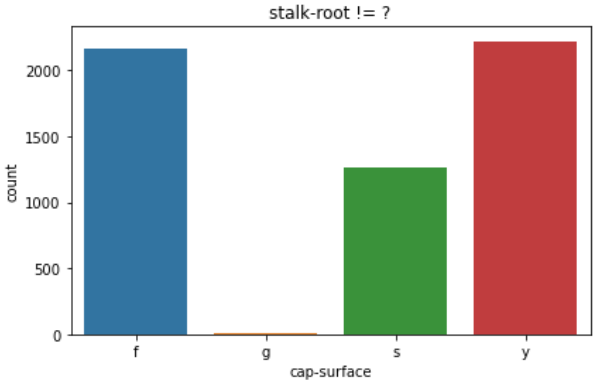
Second, the stalk-root variable has a '**?**' value for its missing values. Rather than impute this missing value, I will divide the dataset into two sections: **(1)** *where* X['stalk-root'] ==? and **(2)** *where* X['stalk-root']! =?. Then, I will analyze the distribution of each variable within those two data sets to determine if they are similar. The fact that the stalk-roots are missing for some of the mushrooms -- *and not missing for the others* --may turn out to be useful/relevant information.

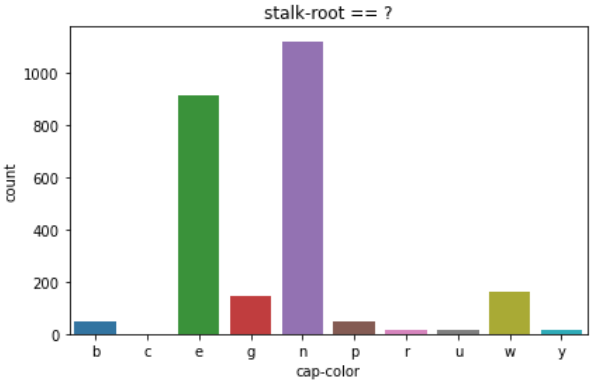
```
for attr in X.columns:
    #Format subplots
    fig, ax = plt.subplots(1,2)
    plt.subplots_adjust(right=2)

    #Construct values to count in each column
    a=set(X[X['stalk-root']=='?'][attr])
    b=set(X[X['stalk-root']!='?'][attr])
    c = a.union(b)
    c = np.sort(np.array(list(c)))

    #Build each subplot
    sns.countplot(x=X[X['stalk-root']=='?'][attr], order=c, ax=ax[0]).set_title('stalk-root == ?')
    sns.countplot(x=X[X['stalk-root']!='?'][attr], order=c, ax=ax[1]).set_title('stalk-root != ?')

    #Plot the plots
    fig.show()
```

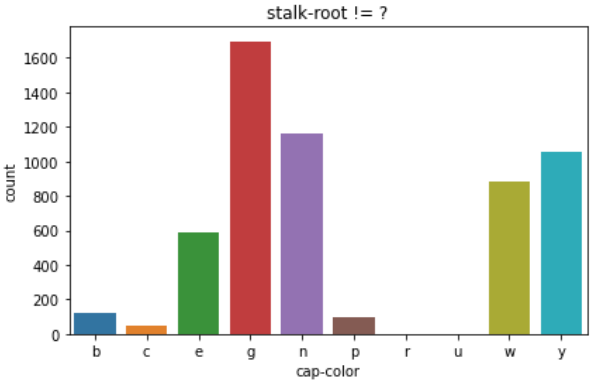


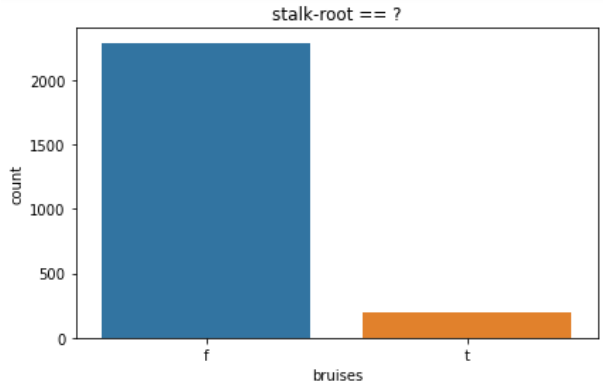


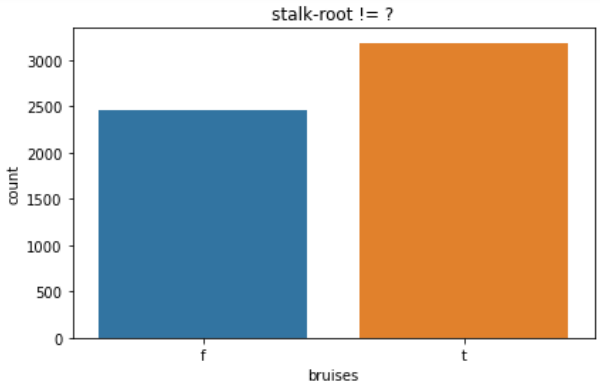


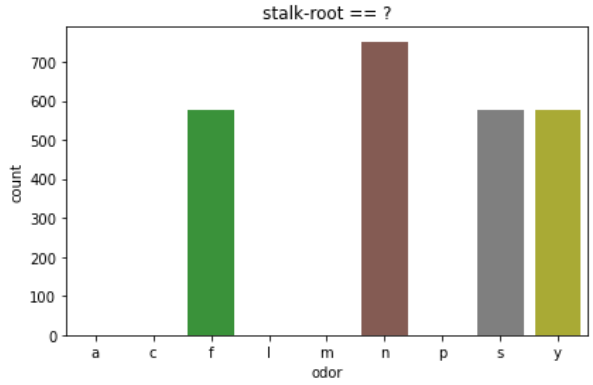


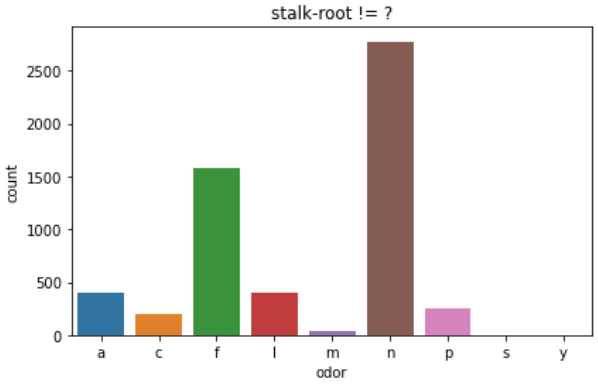


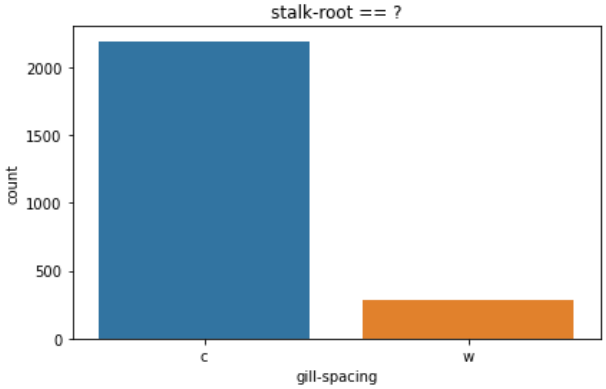


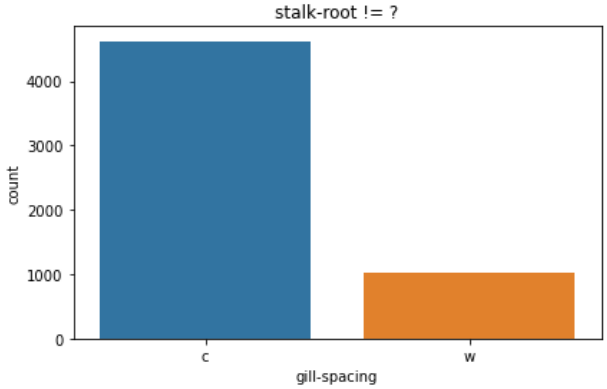

stalk-root == ?
count
bruises
f
t

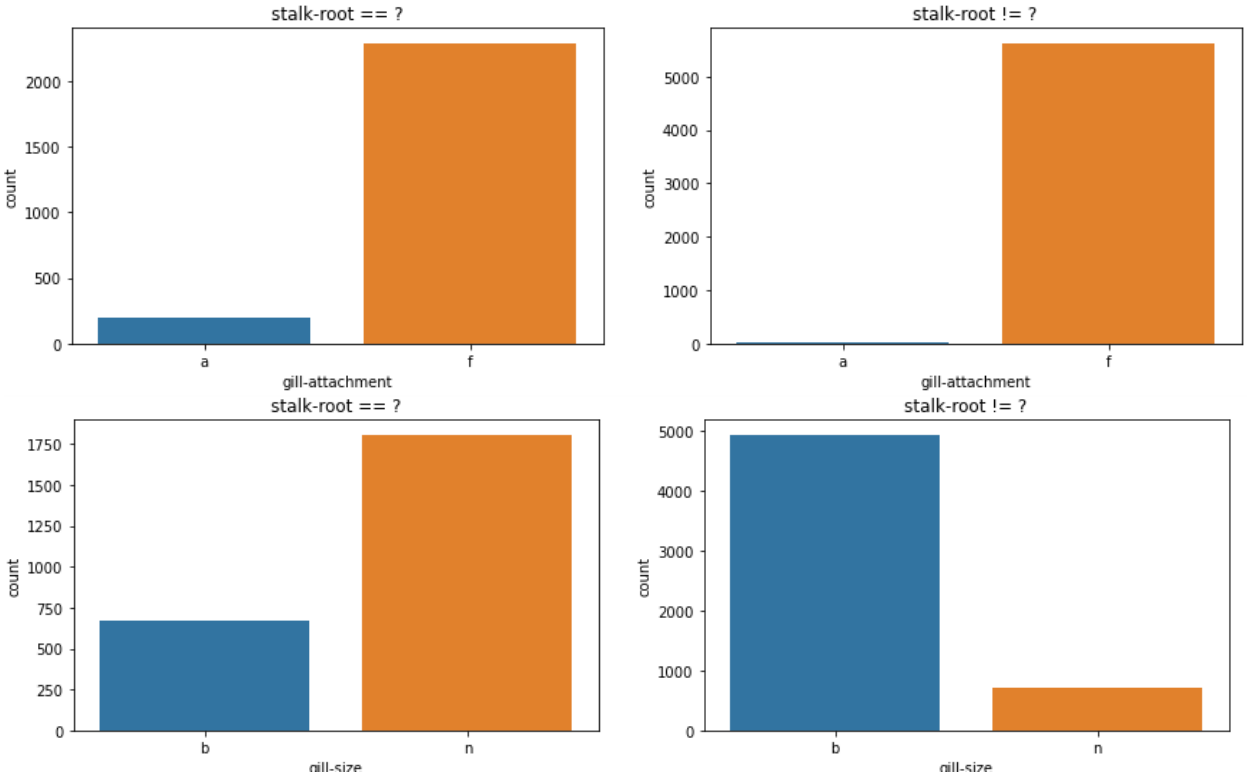

stalk-root != ?
count
bruises
f
t


stalk-root == ?
count
odor
a
c
f
l
m
n
p
s
y


stalk-root != ?
count
odor
a
c
f
l
m
n
p
s
y


stalk-root == ?
count
gill-spacing
c
w


stalk-root != ?
count
gill-spacing
c
w



Since many of the distributions vary greatly, and because of the mushrooms have the value **'?'** for their stalk-root, we will not impute the **'?'** values, rather, we will encode them just as we would the rest of the values in that column.

```
print( (len(X[X['stalk-root']=='?']) / len(X))*100, '%', sep='')
```

```
30.526834071885773%
```

After encoding the variables, the data looks as following:

```
#For columns with only two values
for col in X.columns:
    if len(X[col].value_counts()) == 2:
        le = LabelEncoder()
        X[col] = le.fit_transform(X[col])
```

```
X.head()
```

	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	stalk-shape	...	stalk-surface-above-ring	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-color	ring-number	ring-type	spore-print-color	populati
0	x	s	n	1	p	1	0	1	k	0	...	s	s	w	w	w	o	p	k	
1	x	s	y	1	a	1	0	0	k	0	...	s	s	w	w	w	o	p	n	
2	b	s	w	1	l	1	0	0	n	0	...	s	s	w	w	w	o	p	n	
3	x	y	w	1	p	1	0	1	n	0	...	s	s	w	w	w	o	p	k	
4	x	s	g	0	n	1	1	0	k	1	...	s	s	w	w	w	o	e	n	

5 rows × 21 columns

```
X = pd.get_dummies(X)
```

```
X.head()
```

	bruises	gill-attachment	gill-spacing	gill-size	stalk-shape	cap-shape_b	cap-shape_c	cap-shape_f	cap-shape_k	cap-shape_s	...	population_s	population_v	population_y	habitat_d	habitat_g
0	1	1	0	1	0	0	0	0	0	0	...	1	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	...	0	0	0	0	1
2	1	1	0	0	0	1	0	0	0	0	...	0	0	0	0	0
3	1	1	0	1	0	0	0	0	0	0	...	1	0	0	0	0
4	0	1	1	0	1	0	0	0	0	0	...	0	0	0	0	1

5 rows × 111 columns

3. Cluster analysis

Let’s visualize the result that arises from clustering the mushroom data set with 2 and 3 clusters.

```
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
```

```
KMeans(n_clusters=2)
```

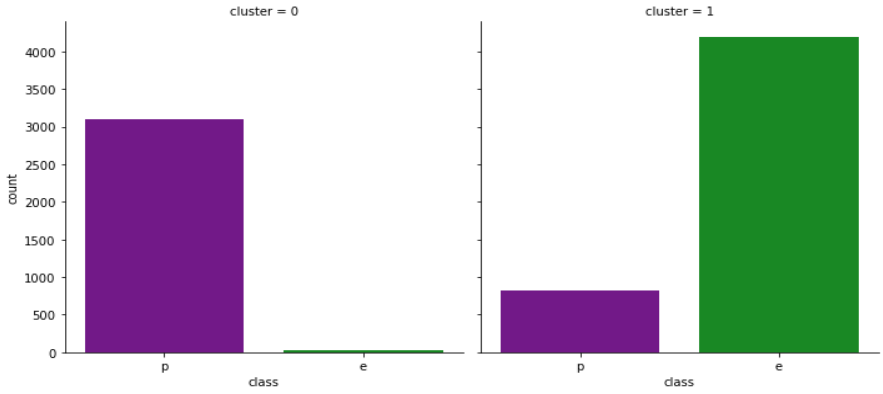
```
clusters = kmeans.predict(X)
```

```
cluster_df = pd.DataFrame()

cluster_df['cluster'] = clusters
cluster_df['class'] = y
```

```
sns.factorplot(col='cluster', y=None, x='class', data=cluster_df, kind='count', order=['p','e'], palette=(["#7d069b","#069b15"]))
```

```
C:\Users\mjeha\anaconda3\lib\site-packages\seaborn\categorical.py:3717: UserWarning: The `factorplot` function has been renamed to `catplot`. The original name will be removed in a future release. Please update your code. Note that the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
```

```
<seaborn.axisgrid.FacetGrid at 0x2417e4bfd30>
```




```
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
```

```
KMeans(n_clusters=3)
```

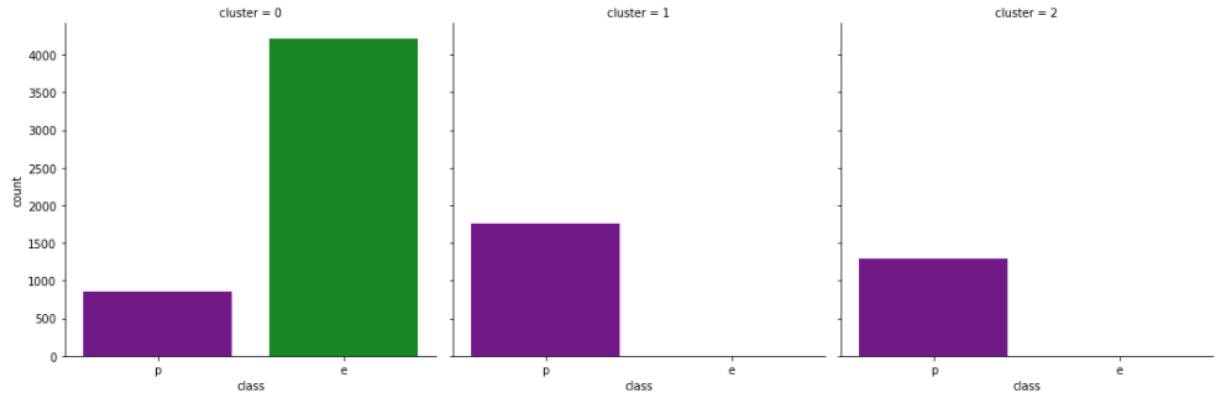
```
clusters = kmeans.predict(X)
```

```
cluster_df = pd.DataFrame()

cluster_df['cluster'] = clusters
cluster_df['class'] = y
```

```
sns.factorplot(col='cluster', y=None, x='class', data=cluster_df, kind='count', order=['p','e'], palette=(["#7d069b","#069b15"]))
```

```
C:\Users\mjeha\anaconda3\lib\site-packages\seaborn\categorical.py:3717: UserWarning: The `factorplot` function has been renamed to `catplot`. The original name will be removed in a future release. Please update your code. Note that the default `kind` in `factorplot` (`'point'`) has changed `'strip'` in `catplot`.
  warnings.warn(msg)
```

```
<seaborn.axisgrid.FacetGrid at 0x2417e59da30>
```



For the 2 clusters, one cluster mostly contains **edible** mushrooms, and the other cluster mostly contains **poisonous** mushrooms.

So, if we were given a mushroom, and we'd like to predict whether it is **edible** or **poisonous**, we could first figure out which cluster it belongs to and then make our prediction based off of the percentage of **poisonous** vs. **edible** mushrooms in that cluster.

4. Classification

encoding our y-labels numerically so that our model can work with it.

Since each mushroom is either **poisonous** or **edible**.

So, when y==1, the mushroom is **poisonous**, and when y==0, the mushroom is **edible.**

```
le = LabelEncoder()
y = le.fit_transform(y)

y
```

```
array([1, 0, 0, ..., 0, 1, 0])
```

Generating and fitting testing and training sets:

```
#Our training set will hold 80% of the data
#and the test set will hold 20% of the data
train_X, test_X, train_y, test_y = train_test_split(X, y, test_size=0.20)
```

```
#K-Means Clustering with two clusters
kmeans = KMeans(n_clusters=2)

#Logistic Regression with no special parameters
logreg = LogisticRegression()
```

```
kmeans.fit(train_X)#Note that kmeans is unlabeled...

logreg.fit(train_X, train_y)#... while logreg IS labeled
```

```
LogisticRegression()
```

Making prediction on the Test data:

```
kmeans_pred = kmeans.predict(test_X)

logreg_pred = logreg.predict(test_X)
```

K-Means clustering does not always give the same results. In order to get around this problem, we will build a second set of predictions from our K-Means model - kmeans_pred_2. This second set of predictions will simply be the bit-wise complement of kmeans_pred, and we will use whichever set of predictions gives us a better score as our final prediction set for the K-Means model.

```
kmeans_pred_2 = []
for x in kmeans_pred:
    if x == 1:
        kmeans_pred_2.append(0)
    elif x == 0:
        kmeans_pred_2.append(1)

kmeans_pred_2 = np.array(kmeans_pred_2)
```

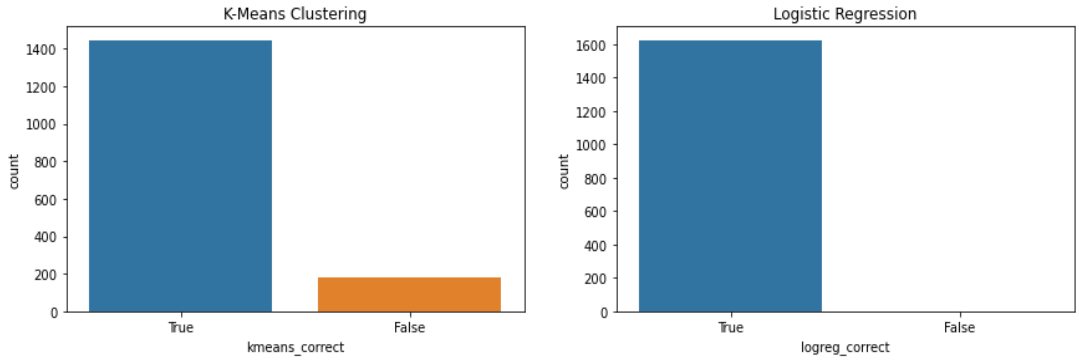
```
if accuracy_score(kmeans_pred, test_y, normalize=False) < accuracy_score(kmeans_pred_2, test_y, normalize=False):
    kmeans_pred = kmeans_pred_2
```

Model evaluation:

```
#This DataFrame will allow us to visualize our results.
result_df = pd.DataFrame()

#The column containing the correct class for each mushroom in the test set, 'test_y'.
result_df['test_y'] = np.array(test_y) #(don't wanna make that mistake again!)

#The predictions made by K-Means on the test set, 'test_X'.
result_df['kmeans_pred'] = kmeans_pred
#The column below will tell us whether each prediction made by our K-Means model was correct.
result_df['kmeans_correct'] = result_df['kmeans_pred'] == result_df['test_y']

#The predictions made by Logistic Regression on the test set, 'test_X'.
result_df['logreg_pred'] = logreg_pred
#The column below will tell us whether each prediction made by our Logistic Regression model was correct.
result_df['logreg_correct'] = result_df['logreg_pred'] == result_df['test_y']
```

```
fig, ax = plt.subplots(1,2)
plt.subplots_adjust(right=2)
sns.countplot(x=result_df['kmeans_correct'], order=[True,False], ax=ax[0]).set_title('K-Means Clustering')
sns.countplot(x=result_df['logreg_correct'], order=[True,False], ax=ax[1]).set_title('Logistic Regression')
fig.show()
```

```
C:\Users\mjeha\AppData\Local\Temp/ipykernel_9584/2202980435.py:5: UserWarning: Matplotlib is currently using module://matplotli
b_inline.backend_inline, which is a non-GUI backend, so cannot show the figure.
  fig.show()
```

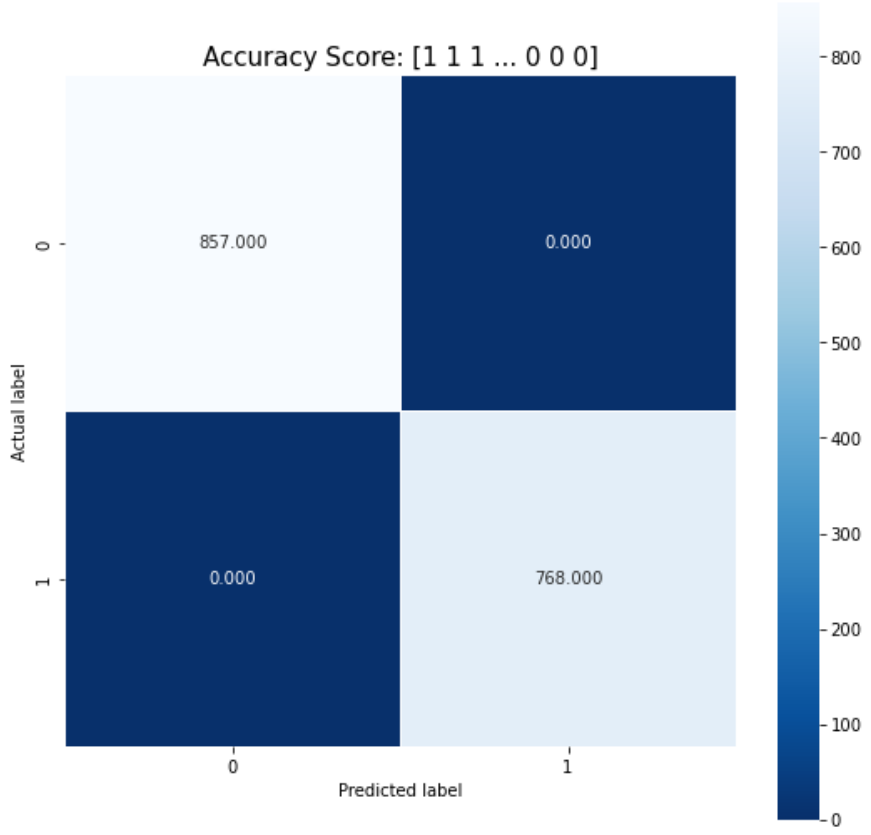


Confusion Matrix:

```
cm = metrics.confusion_matrix(test_y, logreg_pred)
print(cm)
```

```
[[857   0]
 [  0 768]]
```

```
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'Blues_r');
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(logreg_pred)
plt.title(all_sample_title, size = 15);
```



5. Conclusion

from the plots above, I'd say that **Logistic Regression** gave better results, but considering that K-mean is not built for supervised learning it did a great job as well.

6. References

1. Mushroom Data Set, UCI, UCI Machine Learning Repository: Mushroom Data Set
2. Multiple links from google/Kaggle/GitHub to learn about all of that.

Note:

Special thanks to Prof. Kritika Dahiya because of her I could learn about all of that.