Part1: Determining Splits

see photo inserted below

1. What is the entropy of the target variable, “poisonous”?

> The entropy is 0.295. This is because the split of samples is 7/12 for not poisonous and 5/12 for poisonous. Then plug those ratios into the formula e = -((p0 * log(p0)) + (p1 * log(p1)))

2. What is the first attribute a decision tree trained using the entropy and the information gain method

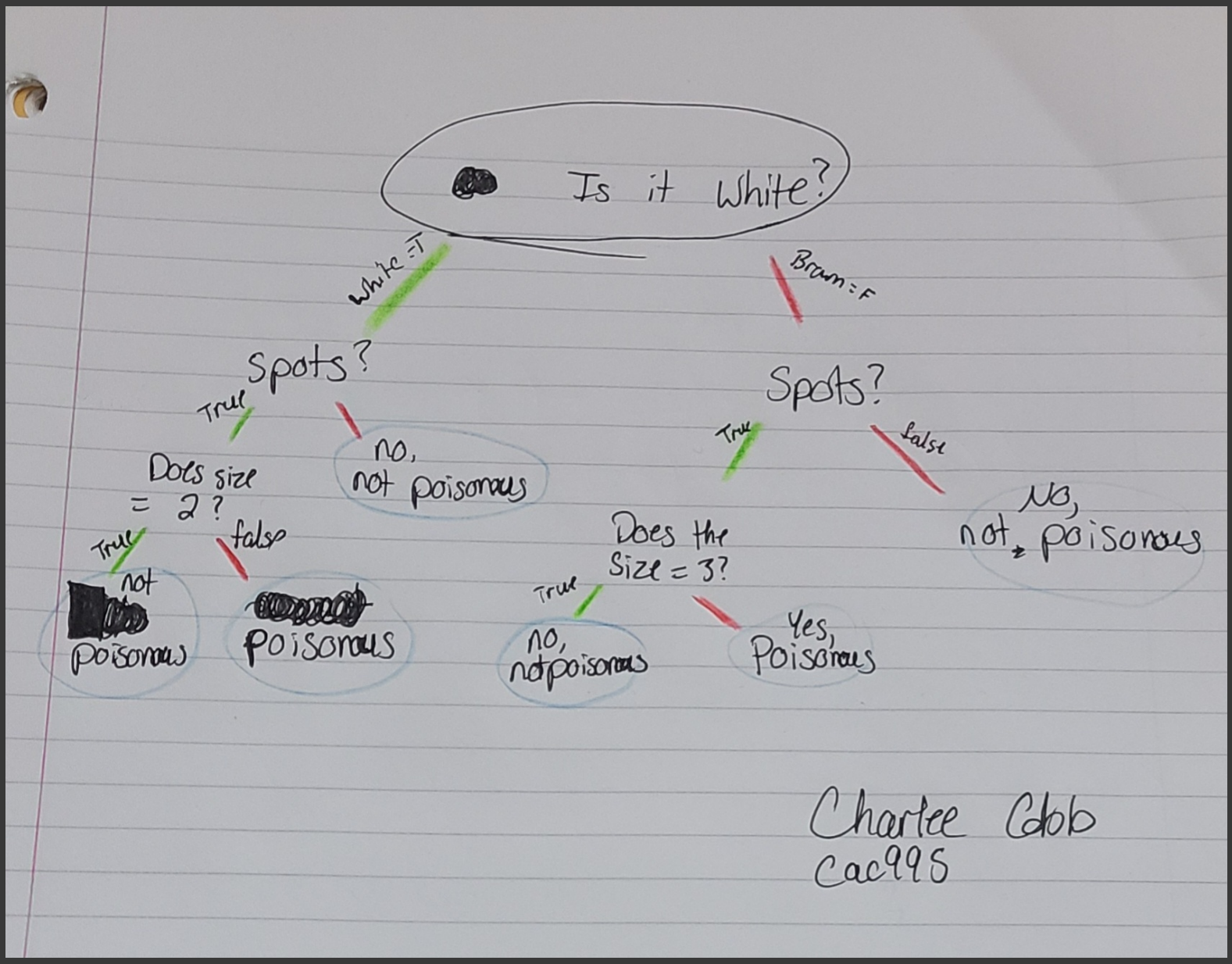
> The first attribute is the one with the highest information gain from the data set. In this case, the first attribute is asking if the mushroom has spots after determining color of the mushroom.

3. What is the information gain of this attribute?

> The Entropy of the parent node(is it white?) is 1 because there are 6/12 options for brown and 6/12 options for white. The entropy of the child is 0, because the split creates 6 white mushrooms and 6 brown mushrooms. Because the child is the same class of color, it is a pure dataset, resulting in an entropy of 0. Therefore, the information gain of this attribute is 1 (parent entropy - child entropy)

4. Draw the full decision tree learned from this data set.

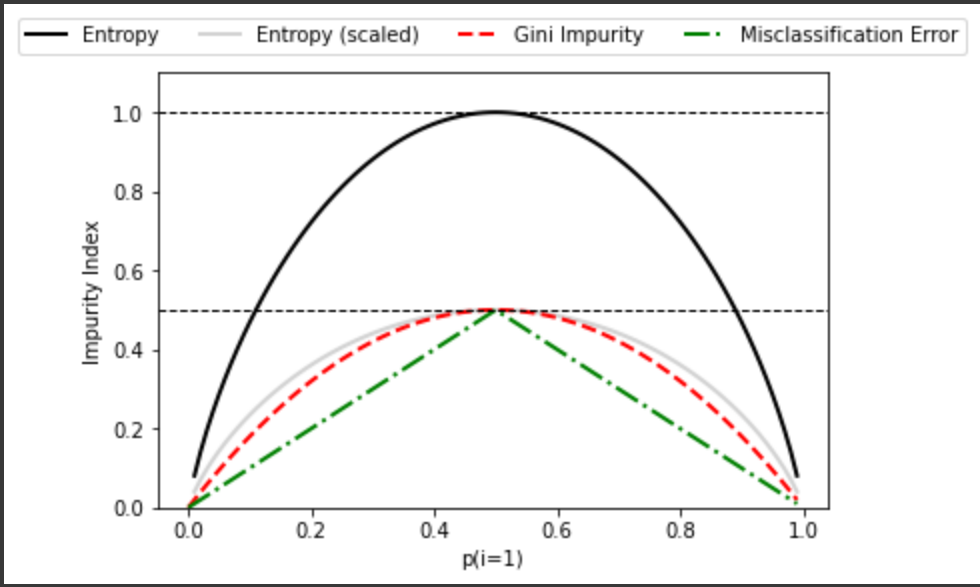
> see photo below



Part 2: Decision Tree using Python

```
# function definitions
#didn't end up using these, But I thought it was still helpful to keep
import matplotlib.pyplot as plt
import numpy as np
def gini(p):
    return (p)*(1 - (p)) + (1 - p)*(1 - (1-p))

def entropy(p):
    return - p*np.log2(p) - (1 - p)*np.log2((1 - p))

def error(p):
    return 1 - np.max([p, 1 - p])

x = np.arange(0.0, 1.0, 0.01)
ent = [entropy(p) if p != 0 else None for p in x]
sc_ent = [e*0.5 if e else None for e in ent]
```

```
err = [error(i) for i in x]
fig = plt.figure()
ax = plt.subplot(111)
for i, lab, ls, c, in zip([ent, sc_ent, gini(x), err],
                          ['Entropy', 'Entropy (scaled)',
                           'Gini Impurity',
                           'Misclassification Error'],
                          ['-', '-', '--', '-.'],
                          ['black', 'lightgray',
                           'red', 'green', 'cyan']):
    line = ax.plot(x, i, label=lab,
                   linestyle=ls, lw=2, color=c)

ax.legend(loc='upper center', bbox_to_anchor=(0.5, 1.15),
          ncol=5, fancybox=True, shadow=False)

ax.axhline(y=0.5, linewidth=1, color='k', linestyle='--')
ax.axhline(y=1.0, linewidth=1, color='k', linestyle='--')
plt.ylim([0, 1.1])
plt.xlabel('p(i=1)')
plt.ylabel('Impurity Index')
plt.show()


from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
    np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=colors[idx],
                    marker=markers[idx], label=cl,
                    edgecolor='black')
    # highlight test samples
    if test_idx:
    # plot all samples
        X_test, y_test = X[test_idx, :], y[test_idx]
        plt.scatter(X_test[:, 0], X_test[:, 1],
                    c='', edgecolor='black', alpha=1.0,
                    linewidth=1, marker='o',
                    s=100, label='test set')
```




```
#1) Divide the data into train (80%) and test (20%)
#loading data:
from sklearn import datasets
from sklearn.model_selection import train_test_split
import numpy as np

#importing through sklearn dataset. sklearn documentation verified they are the same
breast_cancer = datasets.load_breast_cancer()


#dividing data, using all features in dataset:
X = breast_cancer.data[:, 1:31]
y = breast_cancer.target
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, #20% test
                                                    random_state=1,
                                                    stratify=y)
```

```
#2) Using the Scikit-Learn Library train the Decision Tree Classifier using all
# the features of the data and test your model on the test data
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier(criterion='gini',
                              max_depth=6,
                              random_state=1)
#showing accuracy of the Decision Tree Classifier
tree.fit(X_train, y_train)
sc = tree.score(X_test, y_test)
print(sc)
X_combined = np.vstack((X_train, X_test))
y_combined = np.hstack((y_train, y_test))


#comparing accuracy to KNeighborsClassifier
from sklearn.neighbors import KNeighborsClassifier
logreg = KNeighborsClassifier(n_neighbors=6)
logreg.fit(X_train, y_train)
logsc = logreg.score(X_test, y_test)
print(logsc)
```

```
0.956140350877193
0.9473684210526315
```

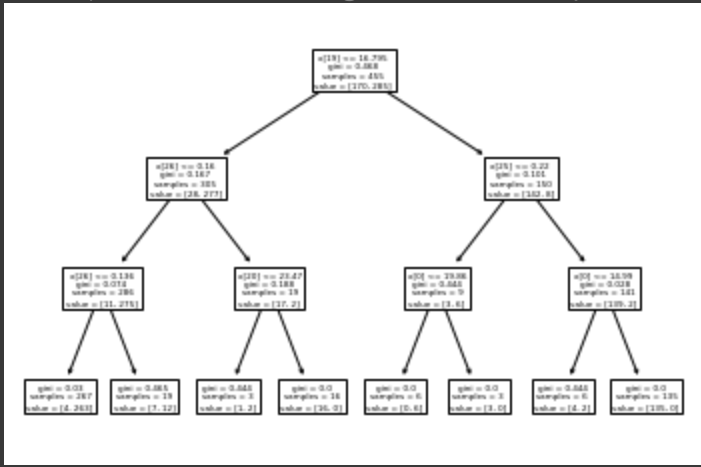
```
#3) Use the Grid Search method to run the model for trees of depth 1, 2, 3, 4, 5,
# and 6 and for the Gini Impurity and Entropy impurity measures
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler


sc = StandardScaler()
tree = DecisionTreeClassifier()

#create pipeline
pipe = Pipeline([("sc", sc), ("tree", tree)])

#make parameters to put in a parameters dictionary
criterion = ["gini", "entropy"]
max_depth = [1, 2, 3, 4, 5, 6]
parameters = dict(tree__criterion =criterion,
                  tree__max_depth = max_depth)

#run gridsearch
gs = GridSearchCV(estimator=pipe,
                  param_grid=parameters,
                  scoring='accuracy',
                  cv=10,
                  n_jobs=-1)
gs.fit(X_train, y_train)
print(gs.best_estimator_.get_params()["tree"])
CV_Result = cross_val_score(gs, X, y, cv=3, n_jobs=-1, scoring="accuracy")
print(CV_Result)
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
[0.88947368 0.94736842 0.9047619 ]
0.9138680033416876
0.024496814922389337
```

```
#4) Determine the best model use the plot.tree() method to visualize it.
#the best model has the depth of 3
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
dctree = DecisionTreeClassifier(criterion='gini',
```

```
                                max_depth=3,
                                random_state=1)
#showing accuracy
plt_dctree = dctree.fit(X_train, y_train)

tree.plot_tree(plt_dctree)
```

```
[Text(0.5, 0.875, 'x[19] <= 16.795\ngini = 0.468\nsamples = 455\nvalue = [170, 285]'),
 Text(0.25, 0.625, 'x[26] <= 0.16\ngini = 0.167\nsamples = 305\nvalue = [28, 277]'),
 Text(0.125, 0.375, 'x[26] <= 0.136\ngini = 0.074\nsamples = 286\nvalue = [11, 275]'),
 Text(0.0625, 0.125, 'gini = 0.03\nsamples = 267\nvalue = [4, 263]'),
 Text(0.1875, 0.125, 'gini = 0.465\nsamples = 19\nvalue = [7, 12]'),
 Text(0.375, 0.375, 'x[20] <= 23.47\ngini = 0.188\nsamples = 19\nvalue = [17, 2]'),
 Text(0.3125, 0.125, 'gini = 0.444\nsamples = 3\nvalue = [1, 2]'),
 Text(0.4375, 0.125, 'gini = 0.0\nsamples = 16\nvalue = [16, 0]'),
 Text(0.75, 0.625, 'x[25] <= 0.22\ngini = 0.101\nsamples = 150\nvalue = [142, 8]'),
 Text(0.625, 0.375, 'x[0] <= 19.86\ngini = 0.444\nsamples = 9\nvalue = [3, 6]'),
 Text(0.5625, 0.125, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]'),
 Text(0.6875, 0.125, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.875, 0.375, 'x[0] <= 14.99\ngini = 0.028\nsamples = 141\nvalue = [139, 2]'),
 Text(0.8125, 0.125, 'gini = 0.444\nsamples = 6\nvalue = [4, 2]'),
 Text(0.9375, 0.125, 'gini = 0.0\nsamples = 135\nvalue = [135, 0]')]
```



```
#5) Use Adaboost to improve the model and evaluate the performance using the test set.
#6) What is the accuracy?
from sklearn.ensemble import AdaBoostClassifier
from sklearn import metrics

ada_test = AdaBoostClassifier()
model = ada_test.fit(X_train, y_train)
y_prediction = model.predict(X_test)
ada_test.score(X_train, y_train)

#finding accuracy
print("Accuracy:",metrics.accuracy_score(y_test, y_prediction))
```

```
Accuracy: 0.9649122807017544
```

With adaboostclassifier, the accuracy has improved to 0.96

Part 3: Random Forest using Python

```
#2) Use the RandomForest classifier (https://scikit-learn.org/stable/modules/generated/
#sklearn.ensemble.RandomForestClassifier.html) to create a model.
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics

forest = RandomForestClassifier(criterion='gini',
                                n_estimators=25,
                                random_state=1,
                                n_jobs=2)

model = forest.fit(X_train, y_train)

y_prediction = model.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_prediction))
```

```
Accuracy: 0.9649122807017544
```

3) Compare the parameters that are provided for the Random Forest classifier and Decision Tree classifier. How many are the same and how many are different?

class sklearn.ensemble.RandomForestClassifier(n_estimators=100, , *criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None, min_impurity_decrease=0.0, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)*
**italicized text*

class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)*
**italicized text*

Random forest classifier contains the n_estimators, bootstrap, oob_score, n_jobs, verbose, warm_start parameters that the Decision Tree Classifier does not. In order, these parameters set up the number of trees in the 'forest', determines if bootstraps samples are used to build trees, and if true can use out-of-bag samples to get the generalization score(oob_score), set the number of jobs that are running parallel to each other, control verbosity, and finally improve fitting by reusing the solution of the previous call to fit or fit a new forest. These parameters are needed because we are considering a large number of decision tree classifiers at once through sub-sampling. Otherwise, the methods in Random forest classifier are very similar to Decision Tree Classifier.

```
#4) Use the Grid Search method to run the model for trees of depth 1, 2, 3, 4,
# 5, and 6 and for the Gini Impurity and Entropy impurity measures. Also set the
# parameter so it will use the "outof-bag" samples for calculating accuracy.
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn import tree
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler


sc = StandardScaler()
tree = RandomForestClassifier(oob_score = True)


pipe = Pipeline([("sc", sc), ("tree", tree)])

#adding bootstrap=T and oob_sore=T so parameter so it will use the "outof-bag"
#samples for calculating accuracy
criterion = ["gini", "entropy"]
max_depth = [1, 2, 3, 4, 5, 6]


parameters = dict(tree__criterion =criterion,
                  tree__max_depth = max_depth)


gs = GridSearchCV(estimator=pipe,
                  param_grid=parameters,
                  scoring='accuracy',
                  cv=10,
                  n_jobs=-1)
gs.fit(X_train, y_train)
print(gs.best_estimator_.get_params()["tree"])
CV_Result = cross_val_score(gs, X, y, cv=3, n_jobs=-1, scoring="accuracy")
```

```
RandomForestClassifier(criterion='entropy', max_depth=4, oob_score=True)
```

```
#5) Test the accuracy of RandomForest using the Test set
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
y_prediction = model.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_prediction))
print(CV_Result)
```

```
Accuracy: 0.9649122807017544
[0.93157895 0.97894737 0.96825397]
```

6) Compare the performance of Decision Tree with Boost and Random Forest.

The decision tree is not as accurate, though it performed better at a shorter depth when using grid search. Using a longer depth of 6 made the decision tree more accurate. However with ada booster, we recieved the same accuracy as we did with the decsion tree at it's longest depth. But ultimately, the Random Forest is the