```python
#!/usr/bin/env python
# coding: utf-8

# ## a) Using an appropriate method identify the top 4 most influential features in classifying this dataset.

# ### Data set loading:

# In[90]:


import pandas as pd
df = pd.read_csv('Mortgage.csv')

## Rows that have the target value
in_progress = df[df.outcome==" "]
## Rows that have no values for target column
done = df[df.outcome!=" "]


# ### Data Processing:

# In[91]:


## Seperating the independent and dependent variables
X=done.loc[:, done.columns != 'outcome'] # It will contains all columns except our target column
Y=done.loc[:, done.columns == 'outcome'] #It will contain target column
Y = pd.to_numeric(Y['outcome'])


# In[92]:


def get_percentage_missing(series):
    """ Calculates percentage of NaN values in DataFrame
    :param series: Pandas DataFrame object
    :return: float
    """
    num = series.isnull().sum()
    den = len(series)
```

```
    return round(num/den, 2)


# In[93]:


get_percentage_missing(X)
## Since there are no missing values , so we can move forward with this
data set.


# ### Data Splitting:

# In[94]:


from sklearn.model_selection import train_test_split
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,
random_state=101)
X_train.shape, y_train.shape, X_test.shape, y_test.shape


# ### Features Selection:

# In[95]:


from sklearn.feature_selection import mutual_info_classif
import numpy as np

# Calculate Mutual Information between each feature and the target
mutual_info = mutual_info_classif(X_train.values, np.ravel(y_train.values))


# In[96]:


# Create Feature Target Mutual Information Series
mi_series = pd.Series(mutual_info)
mi_series.index = X_train.columns
mi_series.sort_values(ascending=False)
```

```
# In[97]:


from sklearn.feature_selection import SelectKBest, SelectPercentile
import matplotlib.pyplot as plt
k_best_features = SelectKBest(mutual_info_classif, k=4).fit(X_train,
np.ravel(y_train))
print('Selected top 4 features:
{}'.format(X_train.columns[k_best_features.get_support()]))


# In[109]:


## Updating our training features
X = X[(X.columns[k_best_features.get_support()])]


# ## b) Now build a model using the Decision Tree Classifier. By adjusting
two suitable parameters (one at a time) reduce the size of the tree to not
more than 10 to 15 nodes in order to improve the interpretability of the
model generated. Which of the two parameters yielded better accuracy while
producing smaller trees?

# ### Adjusting parameters:

# #### Adjusting Max depth:

# In[100]:


from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_curve,auc
max_depths = np.linspace(1, 10, 10)
train_results = []
max_depths
test_results = []
for max_depth in max_depths:
    dt = DecisionTreeClassifier(max_depth=max_depth)
    dt.fit(X_train, y_train)
    train_pred = dt.predict(X_train)
    false_positive_rate, true_positive_rate, thresholds =
```

```
roc_curve(y_train, train_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    # Add auc score to previous train results
    train_results.append(roc_auc)
    y_pred = dt.predict(X_test)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,
y_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    # Add auc score to previous test results
    test_results.append(roc_auc)


from matplotlib.legend_handler import HandlerLine2D
line1, = plt.plot(max_depths, train_results, 'b', label="Train AUC")
line2, = plt.plot(max_depths, test_results, 'r', label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('AUC score')
plt.xlabel('Tree depth')
plt.show()


# #### Adjusting minimum samples leaf:

# In[101]:


min_samples_leafs = np.linspace(0.1, 0.5, 5, endpoint=True)
train_results = []
test_results = []
for min_samples_leaf in min_samples_leafs:
    dt = DecisionTreeClassifier(min_samples_leaf=min_samples_leaf)
    dt.fit(X_train, y_train)
    train_pred = dt.predict(X_train)
    false_positive_rate, true_positive_rate, thresholds =
roc_curve(y_train, train_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    train_results.append(roc_auc)
    y_pred = dt.predict(X_test)
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,
y_pred)
    roc_auc = auc(false_positive_rate, true_positive_rate)
    test_results.append(roc_auc)
```

```
line1, = plt.plot(min_samples_leafs, train_results, 'b', label="Train AUC")
line2, = plt.plot(min_samples_leafs, test_results, 'r', label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel("AUC score")
plt.xlabel("min samples leaf")
plt.show()


# ### Testing Decision Tree Classifier:

# In[103]:


from sklearn.model_selection import KFold


def Cross_validation(data, targets, clf_cv, model_name): #### Performs
cross-validation

    kf = KFold(n_splits=10, shuffle=True, random_state=1) # 10-fold
cross-validation

    scores=[]
    data_train_list = []
    targets_train_list = []
    data_test_list = []
    targets_test_list = []
    iteration = 0
    print("Performing cross-validation for {}...".format(model_name))
    for train_index, test_index in kf.split(data):
        iteration += 1
        print("Iteration ", iteration)
        data_train_cv, targets_train_cv = data[train_index],
targets[train_index]
        data_test_cv, targets_test_cv = data[test_index],
targets[test_index]
        data_train_list.append(data_train_cv) # appending training data for
each iteration
        data_test_list.append(data_test_cv) # appending test data for each
iteration
        targets_train_list.append(targets_train_cv) # appending training
targets for each iteration
        targets_test_list.append(targets_test_cv) # appending test targets
```

```
for each iteration
        print("Shape of training data: ", data_train_cv.shape)
        print("Shape of test data: ", data_test_cv.shape)

        clf_cv.fit(data_train_cv, targets_train_cv) # Fitting SVC

        score = clf_cv.score(data_test_cv, targets_test_cv) # Calculating
accuracy
        scores.append(score) # appending cross-validation accuracy for each
iteration

    print("List of cross-validation accuracies for {}:
".format(model_name), scores)
    mean_accuracy = np.mean(scores)
    print("Mean cross-validation accuracy for {}: ".format(model_name),
mean_accuracy)
    print("Best cross-validation accuracy for {}: ".format(model_name),
max(scores))
    max_acc_index = scores.index(max(scores)) # best cross-validation
accuracy
    max_acc_data_train = data_train_list[max_acc_index] # training data
corresponding to best cross-validation accuracy
    max_acc_data_test = data_test_list[max_acc_index] # test data
corresponding to best cross-validation accuracy
    max_acc_targets_train = targets_train_list[max_acc_index] # training
targets corresponding to best cross-validation accuracy
    max_acc_targets_test = targets_test_list[max_acc_index] # test targets
corresponding to best cross-validation accuracy



    return mean_accuracy, max_acc_data_train, max_acc_data_test,
max_acc_targets_train, max_acc_targets_test,scores


# In[113]:


## To visualize the performance of each classifier, we will be noting its
accuracy and its classification report
from sklearn.metrics import confusion_matrix
import seaborn as sns
def visualize_results(max_acc_data_train, max_acc_data_test,
```

```
max_acc_targets_train, max_acc_targets_test, targets, clf):
    clf.fit(max_acc_data_train, max_acc_targets_train) #
    targets_pred = clf.predict(max_acc_data_test) # Prediction on test data
    rep = confusion_matrix(max_acc_targets_test, targets_pred)
    sns.heatmap(rep, annot=True)
    plt.show()
    return


# In[115]:


from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(max_depth=6,min_samples_leaf=0.2) #
DecisionTreeClassifier
mean_accuracy, max_acc_data_train, max_acc_data_test,
max_acc_targets_train, max_acc_targets_test,scores =
Cross_validation(X.values,Y.values, clf, "D tree") # DecisionTreeClassifier
cross-validation
visualize_results(max_acc_data_train, max_acc_data_test,
max_acc_targets_train, max_acc_targets_test, Y.values, clf)

## Comparision through graphs


## argument has accuracy values
plt.plot(scores,color='b',label="D tree")
plt.legend(loc="best")

plt.title("Accuracy accross each fold") ## Setting subtitle of fig 1
plt.xlabel("Folds") ## Setting x-label of fig 1
plt.ylabel("Accuracy"); ## Setting y-label of fig 1


# In[ ]:
```