

## 4.1

Since we don't know the actual relationship between X and y (i.e whether it is linear dependence or quadratic or anything else), I applied a simple transformation of assigning 0 if  $X[i, j]$  if greater than the mean of  $X[i, :]$  and 1 otherwise. I also experimented with median but got better results for mean.

## 4.2

```
def entropy(x):
    n = len(x)
    zero_count = np.count_nonzero(x)
    one_count = n - zero_count
    p_x_0 = zero_count/n
    p_x_1 = one_count/n
    ans = 0
    ans += p_x_0*math.log(1/p_x_0, 2) if p_x_0>0 else 0
    ans += p_x_1*math.log(1/p_x_1, 2) if p_x_1>0 else 0
    return ans
```

```
def conditional_entropy(x, y):

    x_y_given_1_arr = x[np.where(y==1)]
    x_1_y_given_1 = np.count_nonzero(x_y_given_1_arr)
    x_0_y_given_1 = len(x_y_given_1_arr) - x_1_y_given_1

    p_y_1 = len(x_y_given_1_arr)/len(y)
    p_x_0_given_y_1 = x_0_y_given_1/len(x_y_given_1_arr)
    p_x_0_and_y_1 = p_x_0_given_y_1*p_y_1

    p_x_1_given_y_1 = x_1_y_given_1/len(x_y_given_1_arr)
    p_x_1_and_y_1 = p_x_1_given_y_1*p_y_1

    x_y_given_0_arr = x[np.where(y==0)]
    x_1_y_given_0 = np.count_nonzero(x_y_given_0_arr)
    x_0_y_given_0 = len(x_y_given_0_arr) - x_1_y_given_0

    p_y_0 = len(x_y_given_0_arr)/len(y)
    p_x_0_given_y_0 = x_0_y_given_0/len(x_y_given_0_arr)
    p_x_0_and_y_0 = p_x_0_given_y_0*p_y_0

    p_x_1_given_y_0 = x_1_y_given_0/len(x_y_given_0_arr)
    p_x_1_and_y_0 = p_x_1_given_y_0*p_y_0

    ans = 0
    ans += p_x_0_and_y_0 * math.log(1/p_x_0_given_y_0 ,2) if p_x_0_given_y_0 >0 else 0
    ans += p_x_0_and_y_1 * math.log(1/p_x_0_given_y_1 ,2) if p_x_0_given_y_1 >0 else 0
    ans += p_x_1_and_y_0 * math.log(1/p_x_1_given_y_0 ,2) if p_x_1_given_y_0 >0 else 0
    ans += p_x_1_and_y_1 * math.log(1/p_x_1_given_y_1 ,2) if p_x_1_given_y_1 >0 else 0
    return ans
```

```

def mutual_information(x, y):
    return entropy(x) - conditional_entropy(x,y)

```

4.3 :

Stopping criteria:

1. All of the tuples in the partition belong to the same class ( pure split). The entropy of the response y in the current subset of data is zero
2. There are no remaining attributes on which the tuples may be further partitioned.
3. There are no tuples for a given branch.
4. The current subset of data contains too few samples(<= 5% of the training data)

```

def split_data(X, y, split_col):
    first_split_X = []
    first_split_y = []
    second_split_X = []
    second_split_y = []
    for i, row in enumerate(X):
        if row[split_col] == 0:
            first_split_X.append(row)
            first_split_y.append(y[i])
        else:
            second_split_X.append(row)
            second_split_y.append(y[i])
    if len(first_split_X):
        first_split_X = np.delete(first_split_X, split_col, axis=1)
    # print(second_split_X)
    if len(second_split_X):
        second_split_X = np.delete(second_split_X, split_col, axis=1)
    return first_split_X, np.array(first_split_y), second_split_X, np.array(second_split_y)

```

```

def best_split(X, y):
    m = len(X[0])

    mutual_information_arr = []
    for i in range(m):
        mutual_information_arr.append(mutual_information(X[:,i], y))
    return mutual_information_arr.index(max(mutual_information_arr))

```

```
def build_decision_tree(X,y, number_of_samples, orig_columns):

    if len(y) and entropy(y) == 0:
        return "y="+str(y[0])

    if len(X) <= (0.05*number_of_samples):
        y_1 = np.count_nonzero(y)
        return "y=1" if y_1 > (len(y) - y_1) else "y=0"

    if len(X[0]) == 0:
        y_1 = np.count_nonzero(y)
        return "y=1" if y_1 > (len(y) - y_1) else "y=0"

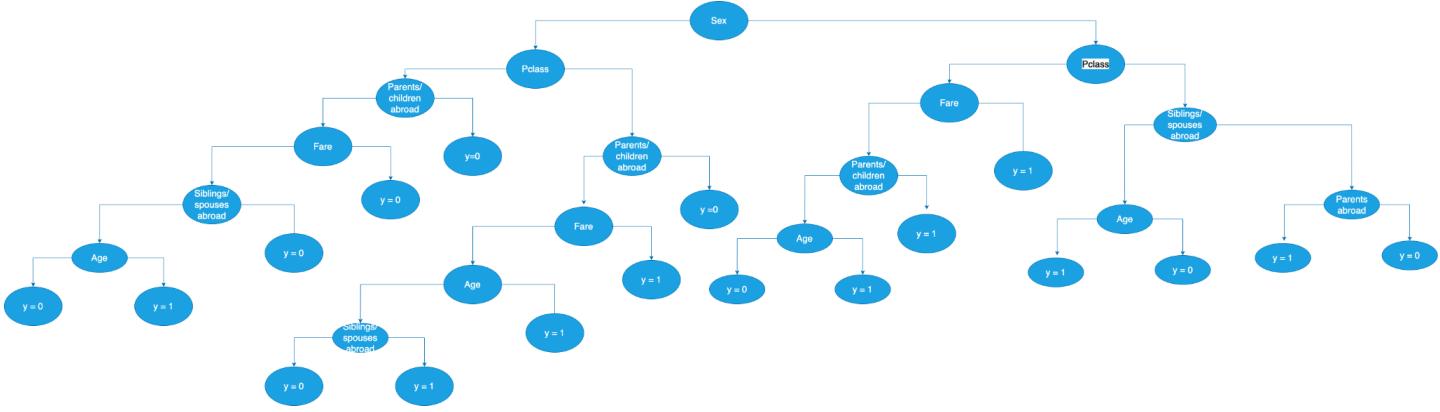
    split_col = best_split(X, y)
    root = {"Feature considered":orig_columns[split_col]}
    orig_columns.pop(split_col)
    first_split_X, first_split_y, second_split_X, second_split_y = split_data(X, y, split_col)

    if len(first_split_X):
        root["0"] = build_decision_tree(first_split_X, first_split_y, number_of_samples, orig_columns[:])
    else:
        y_1 = np.count_nonzero(second_split_y)
        root["0"] = "y=1" if y_1 > (len(second_split_y) - y_1) else "y=0"

    if len(second_split_X):
        root["1"] = build_decision_tree(second_split_X, second_split_y, number_of_samples, orig_columns[:])
    else:
        y_1 = np.count_nonzero(first_split_y)
        root["1"] = "y=1" if y_1 > (len(first_split_y) - y_1) else "y=0"

    return root
```

#### 4.4



#### 4.5

```
0s  i = 0
    number_of_splits = 10
    l = int(len(X)/number_of_splits)
    acc = 0

    while i+l < len(X):
        X_1, y_1 = X[:i], y[:i]
        X_test, y_test = X[i:i+l], y[i:i+l]
        X_2, y_2 = X[i+l:], y[i+l:]
        X_train = np.concatenate((X_1, X_2))
        y_train = np.concatenate((y_1, y_2))

        root = build_decision_tree(X_train, y_train, len(X_train), [i for i in range(len(X_train[0]))])
        acc += (accuracy(X_test, y_test, root))
        i += l

    print("Accuracy = ", acc/number_of_splits)

    ↵ Accuracy =  0.8079545454545455
```

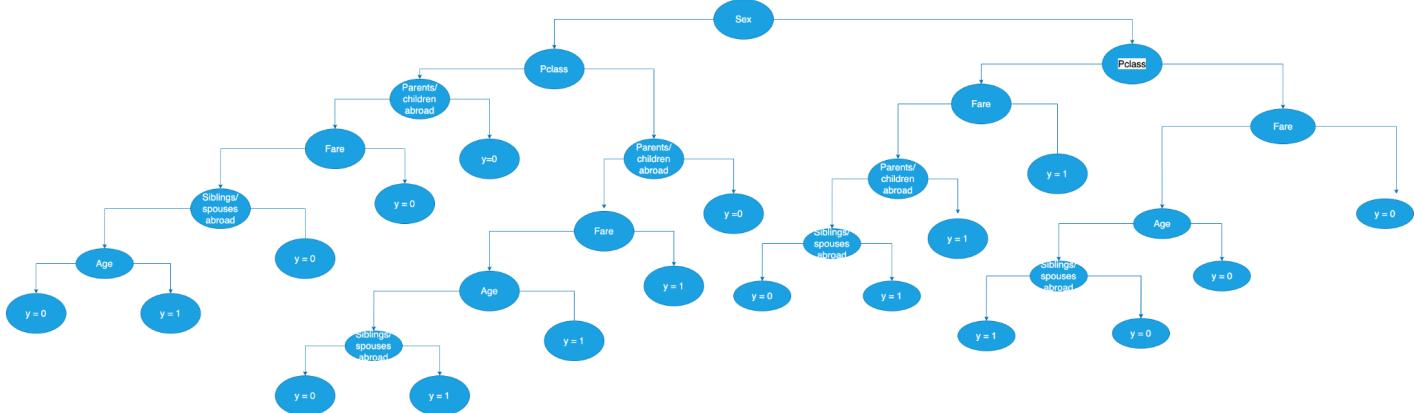
## 4.6

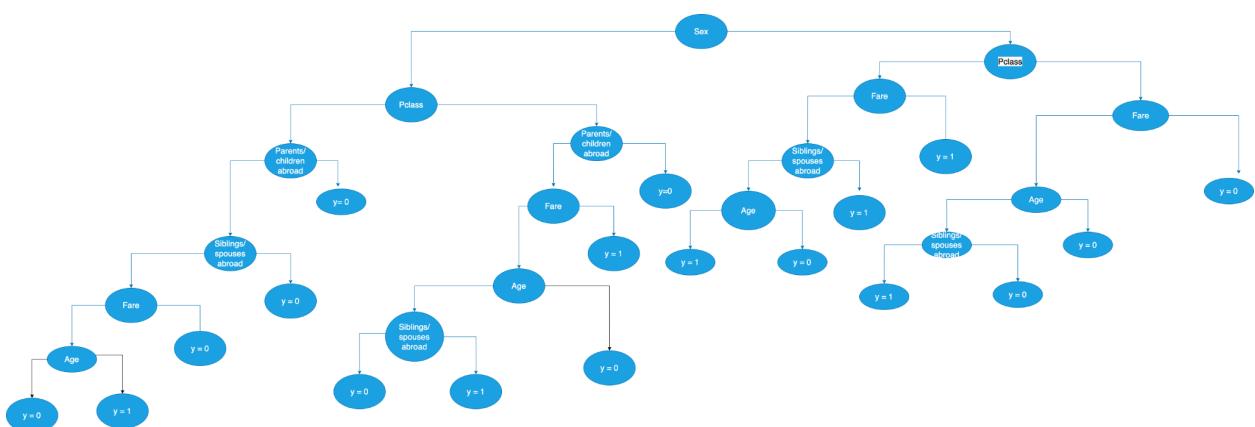
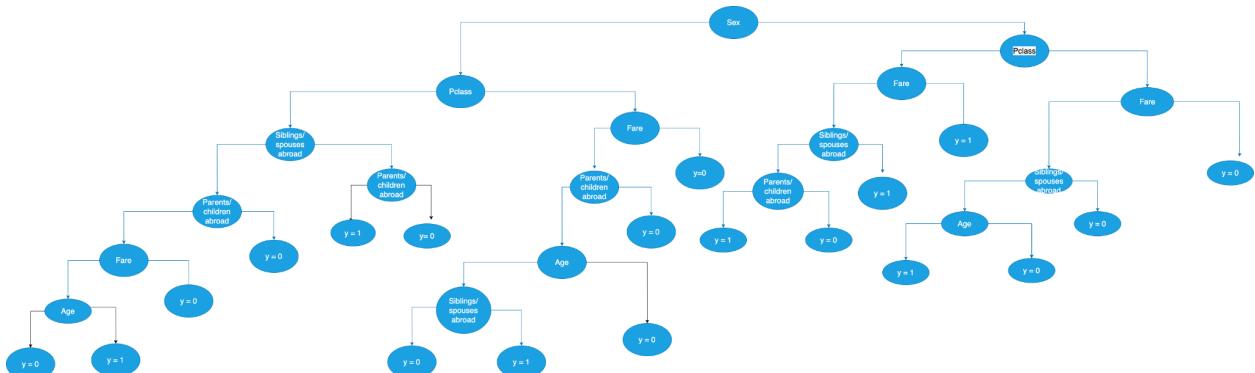
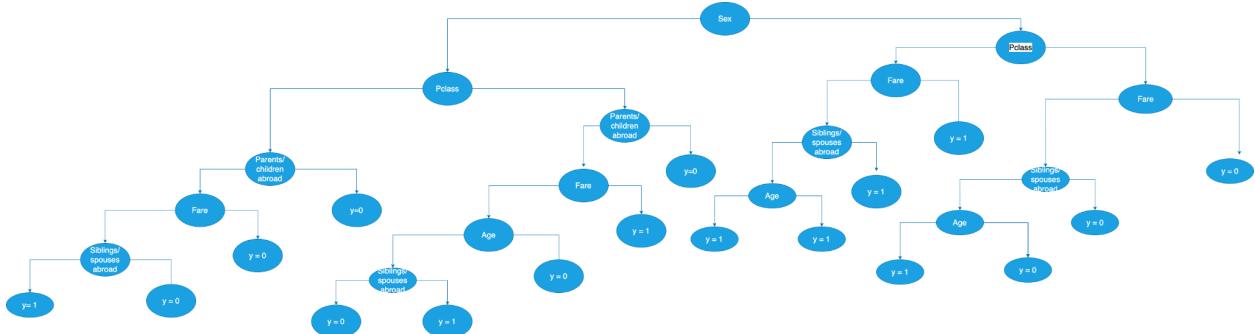
```
x = [1, 0, 45, 2, 2, 20] -> [0, 0, 1, 1, 1, 0] after binary transformation
```

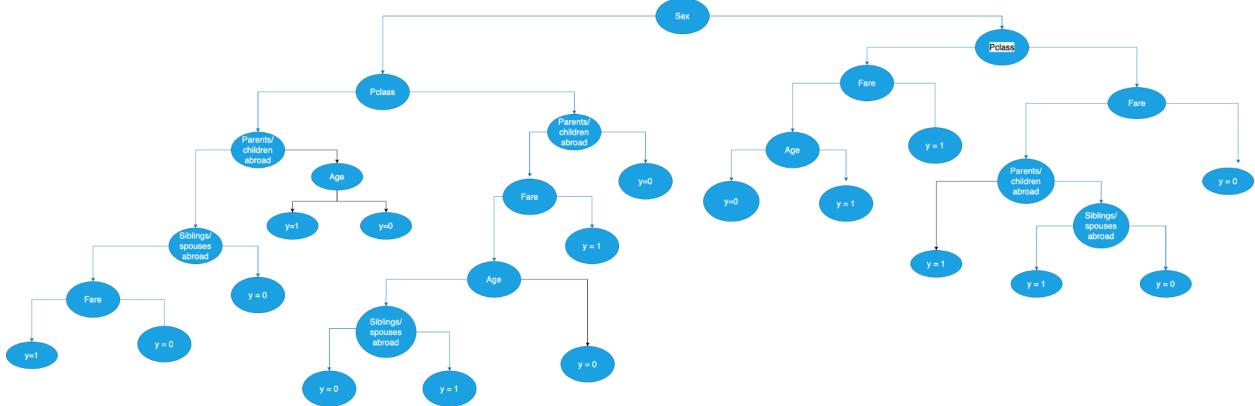
y = 0 i.e x would not have survived the Titanic sinking

## 4.7 (a)

```
[55] # 4. 7 a
forest = []
number_of_trees = 5
for _ in range(number_of_trees):
    X_train_subset, _, y_train_subset, _ = train_test_split(X, y, test_size=0.2, shuffle=True)
    root = build_decision_tree(X_train_subset, y_train_subset, len(X_train_subset), [i for i in range(len(X_train_subset[0]))])
    forest.append(root)
```







4.7 (b)

```

i = 0
number_of_splits = 10
l = int(len(X)/number_of_splits)
forests = []
number_of_trees = 5
accs = []

while i+1 < len(X):
    X_1, y_1 = X[:i], y[:i]
    X_test, y_test = X[i:i+l], y[i:i+l]
    X_2, y_2 = X[i+l:], y[i+l:]
    X_train = np.concatenate((X_1, X_2))
    y_train = np.concatenate((y_1, y_2))

    acc = 0
    forest = []
    for _ in range(number_of_trees):
        X_train_subset, _, y_train_subset, _ = train_test_split(X_train, y_train, test_size=0.2, shuffle=True)
        root = build_decision_tree(X_train_subset, y_train_subset, len(X_train_subset), [i for i in range(len(X_train_subset[0]))])
        forest.append(root)
    for root in forest:
        acc += (accuracy(X_test, y_test, root))

    accs.append(acc/(number_of_trees))
    print("Accuracy for "+str(i/l)+"th fold : "+ str(acc/(number_of_trees)))

    forests.append(forest)
    i += 1

print("10-cross fold avergae accuracy : "+ str(sum(accs)/(number_of_splits)))

```

```
Accuracy for 0th fold : 0.7818181818181819
Accuracy for 1th fold : 0.8204545454545455
Accuracy for 2th fold : 0.7636363636363636
Accuracy for 3th fold : 0.8045454545454545
Accuracy for 4th fold : 0.85
Accuracy for 5th fold : 0.7772727272727273
Accuracy for 6th fold : 0.8113636363636363
Accuracy for 7th fold : 0.7863636363636364
Accuracy for 8th fold : 0.8522727272727272
Accuracy for 9th fold : 0.7886363636363637
10-cross fold average accuracy : 0.8036363636363637
```

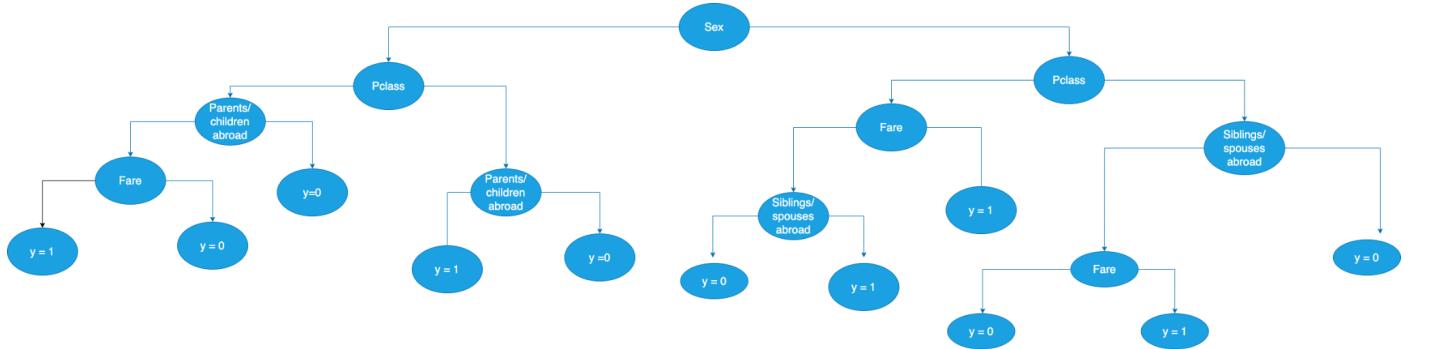
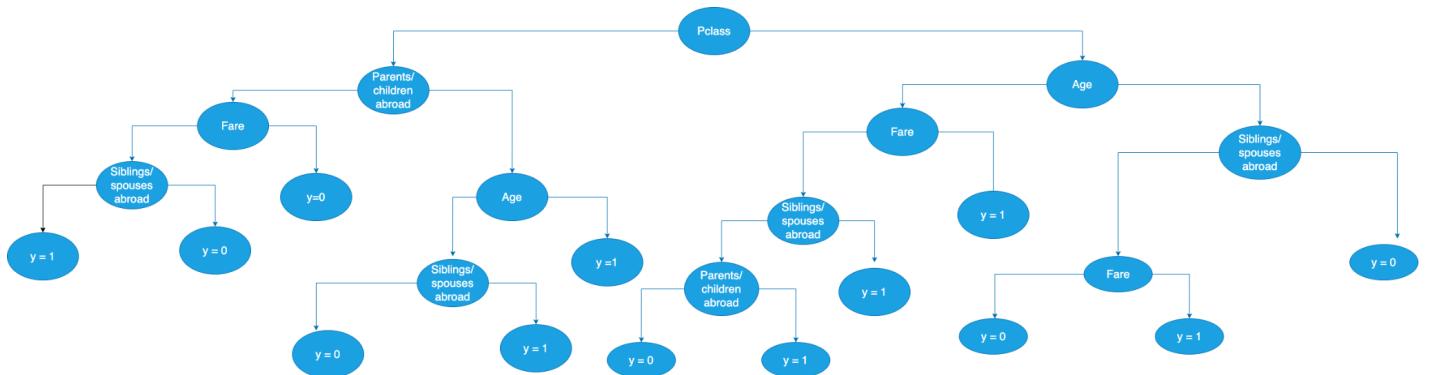
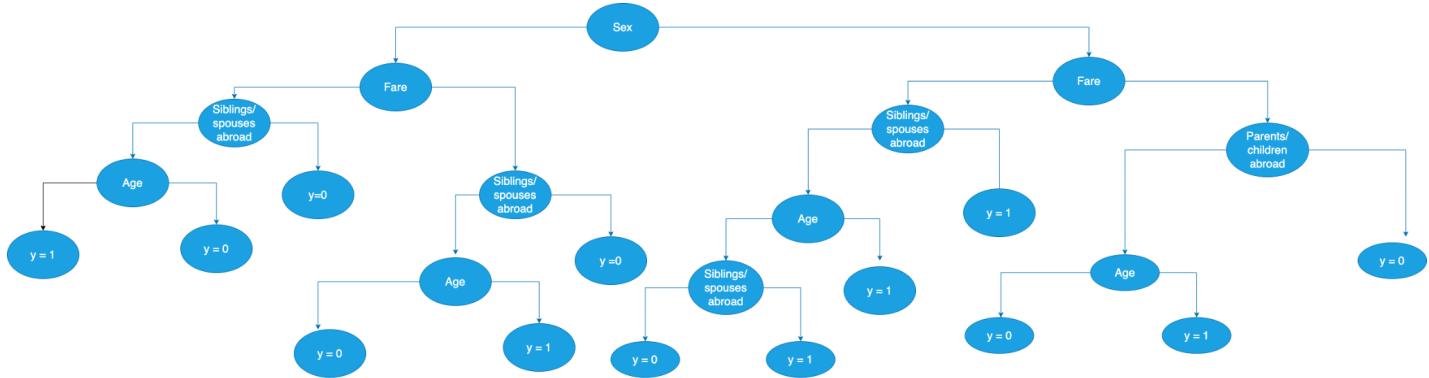
```
def predict(x, root):
    node = root
    while isinstance(node, dict):
        col = node['Feature considered']
        if x[col] == 0:
            node = node['0']
        else:
            node = node['1']
    return float(node.split("=")[-1])
```

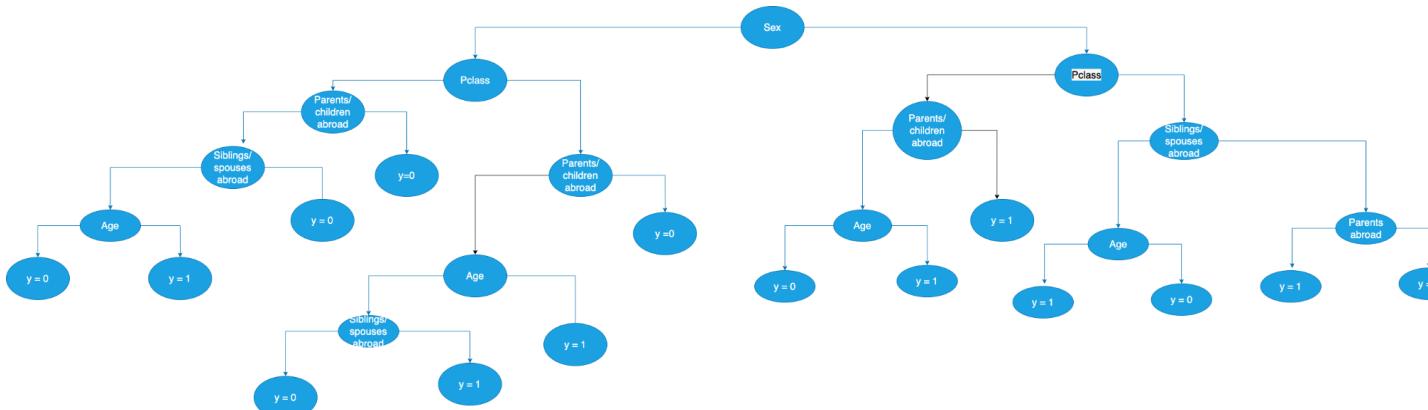
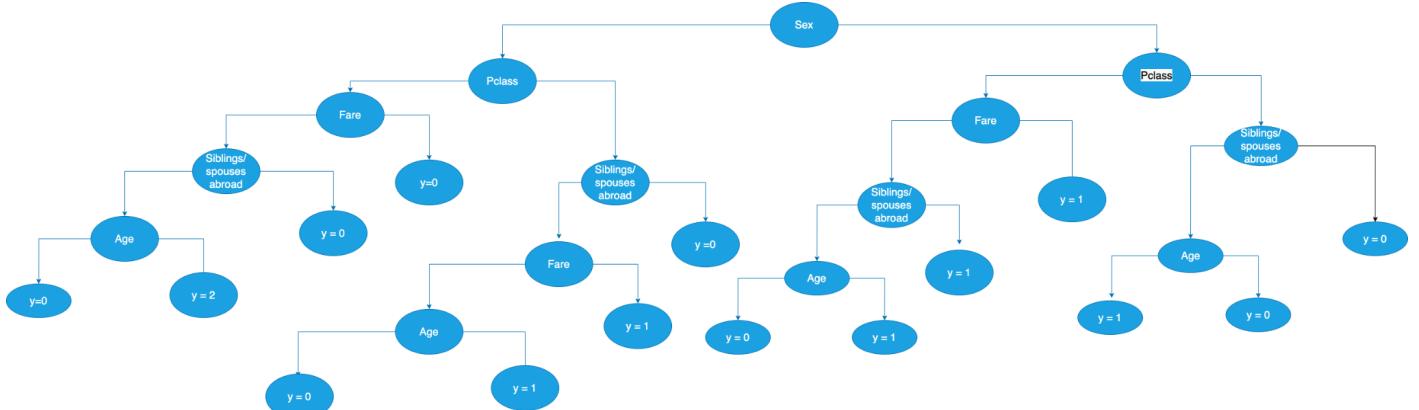
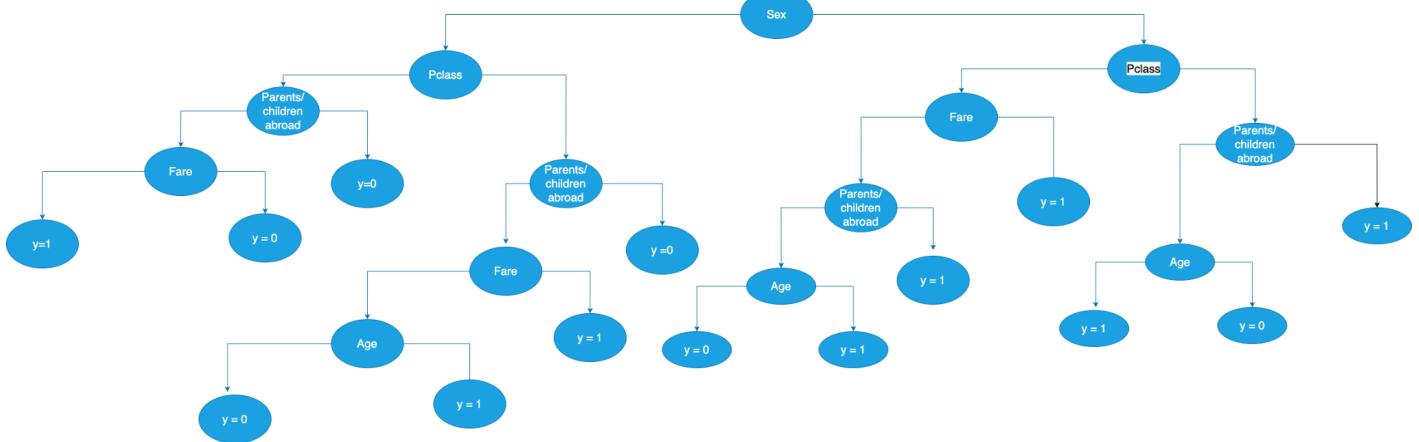
```
def accuracy(X, y, root):
    predictions = []
    for x in X:
        predictions.append(predict(x, root))
    predictions = np.array(predictions)
    return (y == predictions).sum() / y.shape[0]
```

4.7 (c)

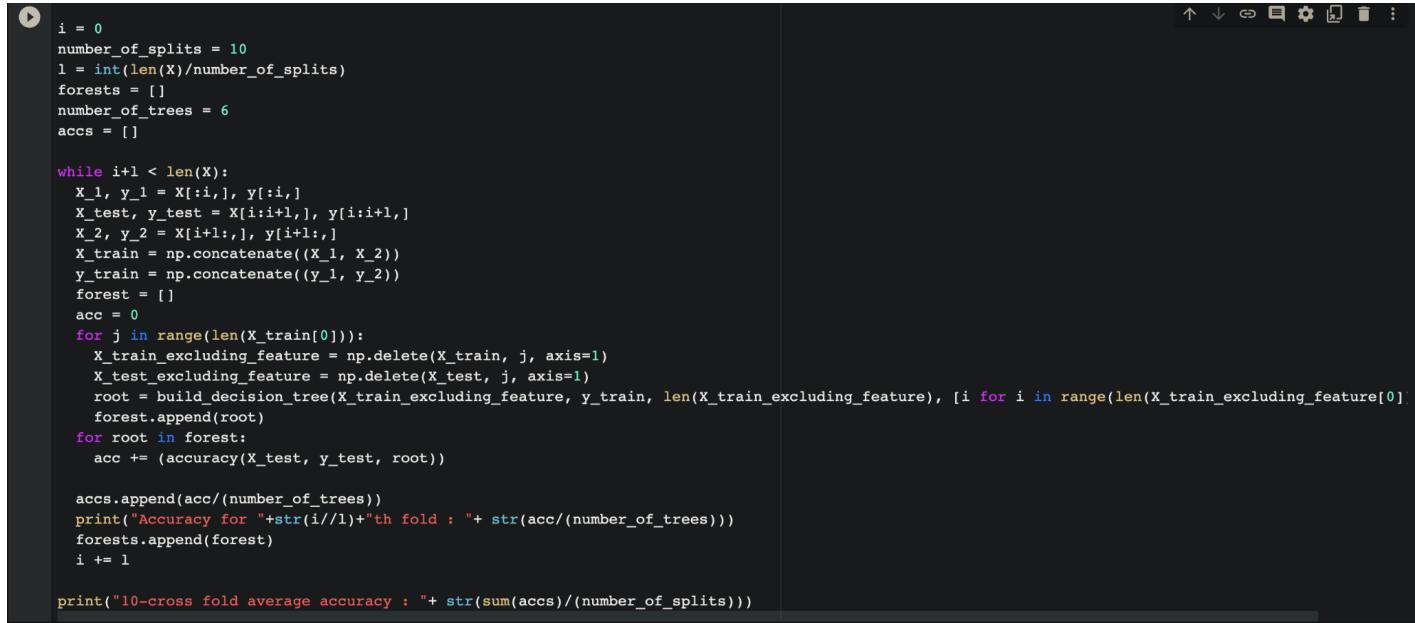
No, x would not have survived the sinking. All five decision trees predicted 0

4.8 (a)





#### 4.8 (b)



```
i = 0
number_of_splits = 10
l = int(len(X)/number_of_splits)
forests = []
number_of_trees = 6
accs = []

while i+1 < len(X):
    X_1, y_1 = X[:i,:], y[:i,:]
    X_test, y_test = X[i:i+l,:], y[i:i+l,:]
    X_2, y_2 = X[i+l:,:], y[i+l:,:]
    X_train = np.concatenate((X_1, X_2))
    y_train = np.concatenate((y_1, y_2))
    forest = []
    acc = 0
    for j in range(len(X_train[0])):
        X_train_excluding_feature = np.delete(X_train, j, axis=1)
        X_test_excluding_feature = np.delete(X_test, j, axis=1)
        root = build_decision_tree(X_train_excluding_feature, y_train, len(X_train_excluding_feature), [i for i in range(len(X_train_excluding_feature[0]))])
        forest.append(root)
    for root in forest:
        acc += (accuracy(X_test, y_test, root))

    accs.append(acc/(number_of_trees))
    print("Accuracy for "+str(i//l)+"th fold : "+ str(acc/(number_of_trees)))
    forests.append(forest)
    i += 1

print("10-cross fold average accuracy : "+ str(sum(accs)/(number_of_splits)))
```

```
Accuracy for 0th fold : 0.6723484848484849
Accuracy for 1th fold : 0.7348484848484849
Accuracy for 2th fold : 0.6723484848484849
Accuracy for 3th fold : 0.6742424242424242
Accuracy for 4th fold : 0.7007575757575758
Accuracy for 5th fold : 0.6742424242424242
Accuracy for 6th fold : 0.6837121212121212
Accuracy for 7th fold : 0.6704545454545454
Accuracy for 8th fold : 0.7253787878787878
Accuracy for 9th fold : 0.6780303030303031
10-cross fold average accuracy : 0.6886363636363636
```

#### 4.8 (c)

No, x would not have survived the sinking. Five of the six decision trees predicted 0.

4.9 -> Yes, both logistic regression model and the decision tree/random forests predicted 'x' would not survive the sinking. I would prefer to use both Random Forests and logistic regression models even though Random Forests gave a higher cross-validation accuracy.. While comparing ML models, we have to compare not just the accuracy but other metrics too such as F1 score, confusion matrix, AUC curves etc. Ideally we should have computed these for both the models, but since we haven't, it's better to rely on both model predictions. Final output can be a weighted average of RF and logistic regression with slightly higher weight for RF since it has higher accuracy.

4.10

4.10

$$H(x|y) = \sum_x \sum_y p(x,y) \log\left(\frac{1}{p(x|y)}\right)$$

$$p(x|y) = \frac{p(y|x) \cdot p(x)}{p(y)}$$

$$H(x|y) = - \sum_x \sum_y p(x,y) \log\left(\frac{1}{p(y)}\right) + \sum_x \sum_y p(x,y) \cdot \log\left(\frac{1}{p(x)}\right) \\ + \sum_x \sum_y p(x,y) \log\left(\frac{1}{p(y|x)}\right)$$

Applying marginal probability

$$= - \sum_y p(y) \log\left(\frac{1}{p(y)}\right) + \sum_x p(x) \log\left(\frac{1}{p(x)}\right) \\ + H(y|x)$$

$$H(x|y) = - H(y) + H(x) + H(y|x)$$

$$H(y) - H(y|x) = H(x) - H(x|y)$$

$$I(y|x) = I(x|y)$$