In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

In [ ]:

```
data = pd.read_csv("titanic.csv")
data.head()
```

Out[ ]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Em
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [ ]:

```
drop_cols = ["Embarked","Cabin","Name","Ticket","PassengerId"]
data = data.drop(drop_cols,axis=1)
```

In [ ]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  891 non-null    int64  
 1   Pclass    891 non-null    int64  
 2   Sex       891 non-null    object 
 3   Age       714 non-null    float64
 4   SibSp     891 non-null    int64  
 5   Parch     891 non-null    int64  
 6   Fare      891 non-null    float64
```

```
dtypes: float64(2), int64(4), object(1)
memory usage: 48.9+ KB
```

In [ ]:

```
data = data.fillna(data["Age"].mean())
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  891 non-null    int64  
 1   Pclass    891 non-null    int64  
 2   Sex       891 non-null    object 
 3   Age       891 non-null    float64
 4   SibSp     891 non-null    int64  
 5   Parch     891 non-null    int64  
 6   Fare      891 non-null    float64
dtypes: float64(2), int64(4), object(1)
memory usage: 48.9+ KB
```

In [ ]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data["Sex"] = le.fit_transform(data["Sex"])
```

In [ ]:

```
data.info()
data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Survived  891 non-null    int64  
 1   Pclass    891 non-null    int64  
 2   Sex       891 non-null    int32  
 3   Age       891 non-null    float64
 4   SibSp     891 non-null    int64  
 5   Parch     891 non-null    int64  
 6   Fare      891 non-null    float64
dtypes: float64(2), int32(1), int64(4)
memory usage: 45.4 KB
```

Out[ ]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare
0	0	3	1	22.0	1	0	7.2500
1	1	1	0	38.0	1	0	71.2833
2	1	3	0	26.0	0	0	7.9250
3	1	1	0	35.0	1	0	53.1000
4	0	3	1	35.0	0	0	8.0500

In [ ]:

```
input_cols = ["Pclass","Sex","Age","SibSp","Parch","Fare"]
output_cols = ["Survived"]
```

```
X = data[input_cols]
Y = data[output_cols]
print(X.shape,Y.shape)
```

```
(891, 6) (891, 1)
```

In [ ]:

```
def entropy(cols):
    counts = np.unique(cols,return_counts=True)
    N = float(cols.shape[0])

    ent = 0.0
    for ix in counts[1]:
        p = ix/N
        ent += (-1.0*p*np.log2(p))

    return ent
```

In [ ]:

```
def divided_cols(x_data,fkey,fval):
    x_left = pd.DataFrame([],columns=x_data.columns)
    x_right = pd.DataFrame([],columns=x_data.columns)

    for ix in range(x_data.shape[0]):
        val = x_data[fkey].loc[ix]

        if val>fval:
            x_right = x_right.append(x_data.loc[ix])
        else:
            x_left = x_left.append(x_data.loc[ix])

    return x_left,x_right
```

In [ ]:

```
left,right = divided_cols(data[:10],"Survived",0.5)
print(left)
print(right)
```

```
   Survived  Pclass  Sex        Age  SibSp  Parch     Fare
0       0.0     3.0  1.0  22.000000    1.0    0.0   7.2500
4       0.0     3.0  1.0  35.000000    0.0    0.0   8.0500
5       0.0     3.0  1.0  29.699118    0.0    0.0   8.4583
6       0.0     1.0  1.0  54.000000    0.0    0.0  51.8625
7       0.0     3.0  1.0   2.000000    3.0    1.0  21.0750
   Survived  Pclass  Sex   Age  SibSp  Parch     Fare
1       1.0     1.0  0.0  38.0    1.0    0.0  71.2833
2       1.0     3.0  0.0  26.0    0.0    0.0   7.9250
3       1.0     1.0  0.0  35.0    1.0    0.0  53.1000
8       1.0     3.0  0.0  27.0    0.0    2.0  11.1333
9       1.0     2.0  0.0  14.0    1.0    0.0  30.0708
```

In [ ]:

```
def information_gain(x_data,fkey,fval):
    left,right = divided_cols(x_data,fkey,fval)
    l = float(left.shape[0])/x_data.shape[0]
    r = float(right.shape[0])/x_data.shape[0]

    if l==0 or r==0:
        return -100000
    i_gain = entropy(x_data.Survived) - (l*entropy(left.Survived)+r*entropy(right.Survi
    return i_gain
```

In [ ]:

```
for ix in X.columns:
    print(ix)
    print(information_gain(data,ix,data[ix].mean()))
```

```
Pclass
0.07579362743608165
Sex
0.2176601066606142
Age
0.0008836151229467681
SibSp
0.009584541813400071
Parch
0.015380754493137694
Fare
0.042140692838995464
```

In [ ]:

```
class DecisionTree:

    def __init__(self,depth=0,max_depth=5):
        self.left = None
        self.right = None
        self.fkey = None
        self.fval = None
        self.depth = depth
        self.max_depth = max_depth
        self.target = None

    def train(self,X_train):
        features = ["Pclass","Sex","Age","SibSp","Parch","Fare"]
        info_gain = []

        for ix in features:
            i_gain = information_gain(X_train,ix,X_train[ix].mean())
            info_gain.append(i_gain)

        self.fkey = features[np.argmax(info_gain)]
        self.fval = X_train[self.fkey].mean()
        print("Making Tree Features is",self.fkey)

        data_left,data_right = divided_cols(X_train,self.fkey,self.fval)
        data_left = data_left.reset_index(drop=True)
        data_right = data_right.reset_index(drop=True)

        if data_left.shape[0]==0 or data_right.shape[0]==0:
            if X_train.Survived.mean() >= 0.5:
                self.target = "Survive"
            else:
                self.target = "Dead"
            return

        if(self.depth>=self.max_depth):
            if X_train.Survived.mean() >= 0.5:
                self.target = "Survive"
            else:
                self.target = "Dead"
            return

        self.left = DecisionTree(depth=self.depth+1,max_depth=self.max_depth)
        self.left.train(data_left)

        self.right = DecisionTree(depth=self.depth+1,max_depth=self.max_depth)
        self.right.train(data_right)

        if X_train.Survived.mean() >= 0.5:
            self.target = "Survive"
        else:
            self.target = "Dead"
        return

    def predict(self,test):

        if test[self.fkey] > self.fval:
            if self.right==None:
                return self.target
            return self.right.predict(test)
        else:
            if self.left==None:
                return self.target
            return self.left.predict(test)
```

In [ ]:

```
split = int(0.7*data.shape[0])
train_data = data[:split]
test_data = data[split:]
test_data = test_data.reset_index(drop=True)

print(train_data.shape,test_data.shape)
```

```
(623, 7) (268, 7)
```

In [ ]:

```
d = DecisionTree()
d.train(train_data)
```

```
Making Tree Features is Sex
Making Tree Features is Pclass
Making Tree Features is Age
Making Tree Features is SibSp
Making Tree Features is Pclass
Making Tree Features is Age
Making Tree Features is Age
Making Tree Features is SibSp
Making Tree Features is Parch
Making Tree Features is Pclass
Making Tree Features is SibSp
Making Tree Features is Fare
Making Tree Features is Parch
Making Tree Features is Age
Making Tree Features is Pclass
Making Tree Features is Age
Making Tree Features is Age
Making Tree Features is Parch
Making Tree Features is SibSp
Making Tree Features is Fare
Making Tree Features is Age
Making Tree Features is Age
Making Tree Features is Fare
Making Tree Features is Age
Making Tree Features is Age
Making Tree Features is Fare
Making Tree Features is Age
Making Tree Features is Parch
Making Tree Features is Fare
Making Tree Features is Fare
Making Tree Features is Fare
Making Tree Features is Age
Making Tree Features is Fare
Making Tree Features is Parch
Making Tree Features is Fare
Making Tree Features is Age
Making Tree Features is Age
Making Tree Features is Fare
Making Tree Features is Fare
Making Tree Features is SibSp
Making Tree Features is Fare
Making Tree Features is Age
Making Tree Features is Fare
Making Tree Features is Pclass
Making Tree Features is SibSp
Making Tree Features is Age
Making Tree Features is Age
Making Tree Features is Age
Making Tree Features is Pclass
Making Tree Features is Age
Making Tree Features is SibSp
Making Tree Features is Fare
Making Tree Features is SibSp
Making Tree Features is Age
Making Tree Features is Parch
Making Tree Features is SibSp
Making Tree Features is SibSp
Making Tree Features is Age
Making Tree Features is Age
Making Tree Features is Age
Making Tree Features is Parch
Making Tree Features is Age
Making Tree Features is Age
```

In [ ]:

```
pred = []

for ix in range(test_data.shape[0]):
    pred.append(d.predict(test_data.loc[ix]))
```

In [ ]:

```
pred
```

Out[ ]:

```
['Dead',
 'Dead',
 'Dead',
 'Dead',
 'Survive',
 'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Survive',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Survive',
       'Dead',
       'Dead',
       'Survive',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Survive',
       'Dead',
       'Survive',
       'Dead',
       'Survive',
       'Survive',
       'Dead',
       'Dead',
       'Survive',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Survive',
       'Survive',
       'Survive',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Survive',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
       'Dead',
                'Survive',
                'Survive',
                'Survive',
                'Dead',
                'Dead',
                'Dead',
                'Dead',
                'Dead',
                'Survive',
                'Dead',
                'Dead',
                'Survive',
                'Dead',
                'Survive',
                'Dead',
                'Dead',
                'Dead',
                'Survive',
                'Dead',
                'Survive',
                'Dead',
                'Survive',
                'Dead',
                'Dead',
                'Dead',
                'Dead',
                'Dead',
                'Survive',
                'Survive',
                'Dead',
                'Dead',
                'Survive',
                'Dead',
                'Dead',
                'Dead',
                'Survive',
                'Dead',
                'Survive',
                'Survive',
                'Dead',
                'Dead',
                'Survive',
                'Dead',
                'Dead',
                'Dead',
                'Dead',
                'Dead',
                'Dead',
                'Dead',
                'Dead',
                'Dead',
                'Dead',
                'Survive',
                'Survive',
                'Dead',
                'Dead',
                'Dead',
                'Survive',
                'Survive',
                'Survive',
            'Dead',
            'Survive',
            'Dead',
            'Dead',
            'Dead',
            'Survive',
            'Dead',
            'Dead',
            'Dead',
            'Dead',
            'Survive',
            'Dead',
            'Dead',
            'Dead',
            'Survive',
            'Dead',
            'Survive',
            'Dead',
            'Survive',
            'Dead',
            'Dead',
            'Dead',
            'Dead',
            'Survive',
            'Dead',
            'Survive',
            'Dead',
            'Dead',
            'Dead',
            'Dead',
            'Survive',
            'Survive',
            'Survive',
            'Dead',
            'Dead',
            'Dead',
            'Dead',
            'Survive',
            'Dead',
            'Survive',
            'Dead',
            'Dead',
            'Dead',
            'Dead',
            'Dead',
            'Dead',
            'Dead',
            'Survive',
            'Dead',
            'Dead',
            'Dead',
            'Dead',
            'Survive',
            'Survive',
            'Dead',
            'Dead',
            'Dead',
            'Dead',
            'Survive',
            'Dead',
 'Survive',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Survive',
 'Dead',
 'Dead',
 'Dead',
 'Survive',
 'Dead',
 'Dead',
 'Survive',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Survive',
 'Dead',
 'Survive',
 'Dead',
 'Dead',
 'Dead',
 'Survive',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Survive',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Survive',
 'Dead',
 'Dead',
 'Survive',
 'Survive',
 'Survive',
 'Survive',
 'Survive',
 'Dead',
 'Survive',
 'Dead',
 'Dead',
 'Dead',
 'Survive',
 'Dead',
 'Dead',
 'Survive',
 'Survive',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Survive',
 'Dead',
 'Dead',
 'Survive',
 'Survive',
 'Dead',
 'Dead',
 'Dead',
 'Survive',
 'Survive',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Dead',
 'Survive',
 'Dead',
 'Dead',
 'Dead']
```

In [ ]:

```
le = LabelEncoder()
y_pred = le.fit_transform(pred)
```

In [ ]:

```
y_pred = np.array(y_pred).reshape((-1,1))
y_actual = test_data[output_cols]
print(y_pred.shape,y_actual.shape)
```

```
(268, 1) (268, 1)
```

In [ ]:

```
y_pred = np.array(y_pred).reshape((-1,1))
acc = np.sum(np.array(y_pred)==np.array(y_actual))/y_actual.shape[0]
```

In [ ]:

```
print(acc)
```

```
0.8171641791044776
```

Ensemble Method

In [ ]:

```
X_train = train_data[input_cols]
Y_train = np.array(train_data[output_cols]).reshape((-1,))
X_test = test_data[input_cols]
Y_test = np.array(test_data[output_cols]).reshape((-1,))
```

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:

```
rf = RandomForestClassifier(n_estimators = 10,criterion='entropy',max_depth=5)
```

In [ ]:

```
rf.fit(X_train,Y_train)
```

```
Out[ ]: RandomForestClassifier(criterion='entropy', max_depth=5, n_estimators=10)
```

```
In [ ]: rf.score(X_train,Y_train)
```

```
Out[ ]: 0.8426966292134831
```

```
In [ ]: rf.score(X_test,Y_test)
```

```
Out[ ]: 0.8171641791044776
```

```
In [ ]: from sklearn.model_selection import cross_val_score
        acc = cross_val_score(RandomForestClassifier(n_estimators=10,max_depth=5,criterion='ent
        print(acc)
```

```
0.8170064516129031
```

```
In [ ]:
```