Importing Libraries

In [ ]:
```
import pandas as pd
import numpy as np
```

Reading Dataset

In [ ]:
```
df = pd.read_csv("Train.csv")
```

In [ ]:
```
df.tail()
```

Out[ ]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked
1004	1.0	1.0	Blank, Mr. Henry	male	40.0	0.0	0.0	112277	31.0000	A31	C
1005	3.0	0.0	Laitinen, Miss. Kristina Sofia	female	37.0	0.0	0.0	4135	9.5875	NaN	S
1006	1.0	1.0	Newell, Miss. Marjorie	female	23.0	1.0	0.0	35273	113.2750	D36	C
1007	3.0	1.0	Nicola-Yarred, Master. Elias	male	12.0	1.0	0.0	2651	11.2417	NaN	C
1008	3.0	0.0	Corn, Mr. Harry	male	30.0	0.0	0.0	SOTON/OQ 392090	8.0500	NaN	S

In [ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   pclass     1009 non-null   float64
 1   survived   1009 non-null   float64
 2   name       1009 non-null   object 
 3   sex        1009 non-null   object 
 4   age        812 non-null    float64
 5   sibsp      1009 non-null   float64
 6   parch      1009 non-null   float64
 7   ticket     1009 non-null   object 
 8   fare       1008 non-null   float64
 9   cabin      229 non-null    object 
 10  embarked   1008 non-null   object 
 11  boat       374 non-null    object 
 12  body       98 non-null     float64
```

```
 13  home.dest  582 non-null    object
dtypes: float64(7), object(7)
memory usage: 110.5+ KB
```

Dropping Redundant Columns

In [ ]:
```
columns_to_drop = ["cabin","embarked","home.dest","name","body","boat", "ticket"]
```

In [ ]:
```
data_clean = df.drop(columns_to_drop,axis=1)
```

In [ ]:
```
data_clean.head()
```

Out[ ]:

	pclass	survived	sex	age	sibsp	parch	fare
0	3.0	0.0	female	NaN	0.0	0.0	7.750
1	2.0	0.0	male	39.0	0.0	0.0	26.000
2	2.0	1.0	female	40.0	0.0	0.0	13.000
3	3.0	1.0	female	31.0	1.0	1.0	20.525
4	3.0	1.0	female	NaN	2.0	0.0	23.250

Encoding Class Labels to Numeric Labels

In [ ]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
data_clean['sex'] = le.fit_transform(data_clean['sex'])
```

In [ ]:
```
data_clean.head()
```

Out[ ]:

	pclass	survived	sex	age	sibsp	parch	fare
0	3.0	0.0	0	NaN	0.0	0.0	7.750
1	2.0	0.0	1	39.0	0.0	0.0	26.000
2	2.0	1.0	0	40.0	0.0	0.0	13.000
3	3.0	1.0	0	31.0	1.0	1.0	20.525
4	3.0	1.0	0	NaN	2.0	0.0	23.250

In [ ]:
```
data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   pclass    1009 non-null   float64
 1   survived  1009 non-null   float64
 2   sex       1009 non-null   int32
 3   age       812 non-null    float64
```

```
 4   sibsp     1009 non-null   float64
 5   parch     1009 non-null   float64
 6   fare      1008 non-null   float64
dtypes: float64(6), int32(1)
memory usage: 51.4 KB
```

In [ ]:
```
data_clean = data_clean.fillna(data_clean['age'].mean())
data_clean = data_clean.fillna(data_clean['fare'].mode())
```

In [ ]:
```
data_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1009 entries, 0 to 1008
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   pclass    1009 non-null   float64
 1   survived  1009 non-null   float64
 2   sex       1009 non-null   int32
 3   age       1009 non-null   float64
 4   sibsp     1009 non-null   float64
 5   parch     1009 non-null   float64
 6   fare      1009 non-null   float64
dtypes: float64(6), int32(1)
memory usage: 51.4 KB
```

Dividing Data into X and Y

In [ ]:
```
input_cols = ['pclass', 'sex', 'age', 'sibsp', 'parch', 'fare']
output_cols = ['survived']
```

In [ ]:
```
X = data_clean[input_cols]
Y = data_clean[output_cols]
```

In [ ]:
```
X.shape,Y.shape
```

Out[ ]:
```
((1009, 6), (1009, 1))
```

Defining Entropy and Information Gain

In [ ]:
```
def entropy(col):
    count = np.unique(col,return_counts=True)
    N = float(col.shape[0])
    ent = 0.0
    for ix in count[1]:
        p = ix/N
        ent+=(-1.0 * p*np.log2(p))
    return ent

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

def info_gain(x_data,fkey,fval):
    left,right = divide_data(x_data,fkey,fval)

    l = float(left.shape[0])/x_data.shape[0]
    r = float(right.shape[0])/x_data.shape[0]

    if(left.shape[0] == 0 or right.shape[0] == 0):
        return -100000
    i_gain = entropy(x_data.survived) - (l*entropy(left.survived) + r*entropy(right.
    return i_gain
```

In [ ]:

```
for fx in X.columns:
    print(fx)
    print(info_gain(data_clean,fx,data_clean[fx].mean()))
```

```
pclass
0.055456910002982474
sex
0.19274737190850932
age
0.001955929827451075
sibsp
0.006492394392888956
parch
0.01975608012294816
fare
0.04242793401428169
```

Implementing Decision Tree Class

In [ ]:

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

    def train(self,X_train):
        features = ['pclass', 'sex', 'age', 'sibsp', 'parch', 'fare']
        info_gains = []
        for ix in features:
            i_gain = info_gain(X_train,ix,X_train[ix].mean())
            info_gains.append(i_gain)
        self.fkey = features[np.argmax(info_gains)]
        self.fval = X_train[self.fkey].mean()

        print("Making Decision Tree, Current Node is: ",self.fkey)

        data_left,data_right = divide_data(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 = "Survived"
            else:
                self.target = "Dead"
            return
        if self.depth >= self.max_depth:
            if(X_train.survived.mean()>0.5):
                self.target = "Survived"
            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 = "Survived"
        else:
            self.target = "Dead"
        return

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

Creating test and train split

In [ ]:

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

In [ ]:

```
print(train_data.shape)
```

```
(706, 7)
```

Training Decsion Tree

In [ ]:

```
dt = DecisionTree(max_depth=5)
dt.train(train_data)
```

```
Making Decision Tree, Current Node is:  sex
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
```

```
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  sibsp
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  parch
Making Decision Tree, Current Node is:  fare
Making Decision Tree, Current Node is:  pclass
Making Decision Tree, Current Node is:  age
Making Decision Tree, Current Node is:  age
```

Predicting test data

In [ ]:

```
y_pred = []
for ix in range(test_data.shape[0]):
    y_pred.append(dt.predict(test_data.loc[ix]))
```

In [ ]:
```
y_actual = test_data[output_cols]
```

In [ ]:
```
le = LabelEncoder()
y_pred = le.fit_transform(y_pred)
```

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

```
(303, 1)
```

In [ ]:
```
print(acc)
```

```
survived    0.752475
dtype: float64
```

Creating Random forest Classifier Object using Sk-Learn

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

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 [ ]:
```
rf = RandomForestClassifier(n_estimators=12,criterion='entropy',max_depth=5)
rf.fit(X_train,Y_train)
```

Out[ ]:
```
RandomForestClassifier(criterion='entropy', max_depth=5, n_estimators=12)
```

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

Out[ ]:
```
0.8569405099150141
```