In [1]:

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

In [2]:

```
data = pd.read_csv("titanic/train.csv")
```

In [3]:

```
data.shape
```

Out[3]:

```
(891, 12)
```

In [4]:

```
data.head()
```

Out[4]:

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

In [5]:

```
data.isna().sum()*100/891
```

Out[5]:

```
PassengerId     0.000000
Survived        0.000000
Pclass          0.000000
Name            0.000000
Sex             0.000000
Age            19.865320
SibSp           0.000000
Parch           0.000000
Ticket          0.000000
Fare            0.000000
Cabin          77.104377
Embarked        0.224467
dtype: float64
```

In [6]:

```
columns_to_drop = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'Embarked']
```

In [7]:

```
data_clean = data.drop(columns=columns_to_drop)
data_clean.shape
```

Out[7]:

```
(891, 7)
```

In [8]:

```
data_clean.head()
```

Out[8]:

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

In [9]:

```
data_clean['Sex'] = data_clean['Sex'].astype('category').cat.codes
```

In [10]:

```
data_clean.head()
```

Out[10]:

	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 [11]:

```
data_clean['Age'].median()
```

Out[11]:

```
28.0
```

In [12]:

```
data_clean.fillna(value=data_clean['Age'].median(), inplace=True)
```

In [13]:

```
data_clean.isna().sum()
```

Out[13]:

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
dtype: int64
```

In [14]:

```
data_clean.columns
```

Out[14]:

```
Index(['Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare'],
dtype='object')
```

In [15]:

```
input_cols = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']
output_col = ['Survived']
```

In [ ]:

```
```

Decision Trees Implementation

In [16]:

```
def entropy(col):
    unique, counts = np.unique(col, return_counts=True)
    M = col.shape[0]

    ent = 0.0

    prob = counts/M
    log_probab = np.log2(prob)
    ent = np.sum(prob*log_probab)
    return -1*ent
```

In [17]:

```
entropy(np.array([1,0,1,0,0,1]))
```

Out[17]:

```
1.0
```

In [18]:

```
entropy(data_clean['Survived'])
```

Out[18]:

```
0.9607079018756469
```

In [19]:

```
def divide_data(data, fkey, threshold):

    left = pd.DataFrame([], columns=data.columns)
    right = pd.DataFrame([], columns=data.columns)

    for ix in range(data.shape[0]):
        val = data.iloc[ix][fkey]

        if val> threshold:
            # append the entire row to right DF
            right = right.append(data.iloc[ix])
        else:
            # append the entire row to left DF
            left = left.append(data.iloc[ix])

    return left, right
```

In [20]:

```
def information_gain(data, fkey, threshold):
    # call parent's entropy
    p_ent = entropy(data['Survived'])

    # divide the data into left and right nodes
    left, right = divide_data(data, fkey, data[fkey].mean())

    # get entropy for left child
    left_ent = entropy(left['Survived'])

    # get entropy for right child
    right_ent = entropy(right['Survived'])

    # Calculate samples in - left, right, parent
    left_sample = left.shape[0]/data.shape[0]
    right_sample = right.shape[0]/data.shape[0]

    if left_sample==0 or right_sample==0:
        return -1000

    # implement the I.G formula
    IG = p_ent - (left_sample*left_ent + right_sample*right_ent)

    return IG
```

In [ ]:

```

```

In [21]:

```
for col in data_clean[input_cols].columns:
    ig = information_gain(data_clean, col, data_clean[col].mean())
    print(f"For {col}: Information gain: {ig}")
```

```
For Pclass: Information gain: 0.07579362743608165
For Sex: Information gain: 0.2176601066606142
For Age: Information gain: 0.0008836151229467681
For SibSp: Information gain: 0.009584541813400071
For Parch: Information gain: 0.015380754493137694
For Fare: Information gain: 0.042140692838995464
```

Decision Tree Class

In [39]:

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


    def fit(self, data ):
        # train the DT model.

        features = ['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Fare']
        info_gains = []

        for ix in features:
            ig = information_gain(data, ix, data[ix].mean())
            info_gains.append(ig)


        self.fkey =  features[np.argmax(info_gains)]
        self.threshold = data[self.fkey].mean()
        print("\t"*self.depth,f"Making tree with feature {self.fkey}")

        # Split Data
        left, right = divide_data(data, self.fkey, self.threshold)


        # setting the target to all the nodes
        if data['Survived'].mean() >= 0.5:
            self.target = 1
        else:
            self.target = 0



        # Stopping Criteria

        # truly a leaf node
        if left.shape[0] == 0 or right.shape[0] == 0:
            return

        # Early stopping
        if( self.depth >= DecisionTree.max_depth):
            return


        # Recursive calls to create child nodes
        self.left = DecisionTree(depth = self.depth + 1, max_depth=DecisionTree.max_
        self.left.fit(left)

        self.right = DecisionTree(depth = self.depth+1, max_depth=DecisionTree.max_d
        self.right.fit(right)
```

```
    def predict(self, test):
        # predict class label

        if test[self.fkey]>self.threshold:
            #go to right
            if self.right is None:
                return self.target
            return self.right.predict(test)
        else:
            # go to left
            if self.left is None:
                return self.target
            return self.left.predict(test)
```

In [ ]:

```

```

In [40]:

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

In [65]:

```
model = DecisionTree(max_depth=3)
```

In [66]:

```
model.fit(train_data)
```

```
 Making tree with feature Sex
         Making tree with feature Pclass
                 Making tree with feature Age
                         Making tree with feature SibSp
                         Making tree with feature SibSp
                 Making tree with feature Parch
                         Making tree with feature Age
                         Making tree with feature Fare
         Making tree with feature Fare
                 Making tree with feature Parch
                         Making tree with feature Fare
                         Making tree with feature Age
                 Making tree with feature Pclass
                         Making tree with feature Age
                         Making tree with feature SibSp
```

In [67]:

```
model.fkey
```

Out[67]:

```
'Sex'
```

In [68]:

```
model.left.fkey
```

Out[68]:

```
'Pclass'
```

In [69]:

```
model.right.fkey
```

Out[69]:

```
'Fare'
```

In [70]:

```
model.left.left.fkey
```

Out[70]:

```
'Age'
```

In [71]:

```
test_data.iloc[1]
```

Out[71]:

```
Survived     0.0
Pclass       3.0
Sex          1.0
Age         21.0
SibSp        0.0
Parch        0.0
Fare        16.1
Name: 624, dtype: float64
```

In [ ]:

```

```

In [72]:

```
y_pred = []
for ix in range(test_data.shape[0]):
    p = model.predict(test_data.iloc[ix])
    y_pred.append(p)
```

In [73]:

```
y_pred = np.array(y_pred)
```

accuracy

In [74]:

```
y_test = test_data['Survived'].values
```

In [75]:

```
(y_pred == y_test).sum()/y_pred.shape[0]
```

Out[75]:

```
0.8134328358208955
```

In [ ]:

```

```