In [45]:

```
# importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import re

from sklearn import tree

from IPython.display import Image as PImage
from subprocess import check_call
from PIL import Image, ImageDraw, ImageFont

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
```

In [46]:

```
train_df = pd.read_csv('train.csv')
test_df = pd.read_csv('test.csv')

# Store our test passenger IDs for easy access
PassengerId = test_df['PassengerId']
```

In [47]:

```
full_data = [train_df, test_df]

# Feature that tells whether a passenger had a cabin on the Titanic
train_df['Has_Cabin'] = train_df["Cabin"].apply(lambda x: 0 if type(x)
test_df['Has_Cabin'] = test_df["Cabin"].apply(lambda x: 0 if type(x) =

# Create new feature FamilySize as a combination of SibSp and Parch
for dataset in full_data:
    dataset['FamilySize'] = dataset['SibSp'] + dataset['Parch'] + 1
# Create new feature IsAlone from FamilySize
for dataset in full_data:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1
# Remove all NULLS in the Embarked column
for dataset in full_data:
    dataset['Embarked'] = dataset['Embarked'].fillna('S')
# Remove all NULLS in the Fare column
for dataset in full_data:
    dataset['Fare'] = dataset['Fare'].fillna(train_df['Fare'].median()

# Remove all NULLS in the Age column
# Remove all NULLS in the Age column
for dataset in full_data:
    age_avg = dataset['Age'].mean()
    age_std = dataset['Age'].std()
    age_null_count = dataset['Age'].isnull().sum()
    age_null_random_list = np.random.randint(age_avg - age_std, age_av
    # Next line has been improved to avoid warning
    dataset.loc[np.isnan(dataset['Age']), 'Age'] = age_null_random_lis
    dataset['Age'] = dataset['Age'].astype(int)

# Define function to extract titles from passenger names
def get_title(name):
    title_search = re.search(' ([A-Za-z]+)\.', name)
    # If the title exists, extract and return it.
    if title_search:
        return title_search.group(1)
    return ""

for dataset in full_data:
    dataset['Title'] = dataset['Name'].apply(get_title)
# Group all non-common titles into one single grouping "Rare"
for dataset in full_data:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess','C

    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')

for dataset in full_data:
    # Mapping Sex
    dataset['Sex'] = dataset['Sex'].map( {'female': 0, 'male': 1} ).as

    # Mapping titles
    title_mapping = {"Mr": 1, "Master": 2, "Mrs": 3, "Miss": 4, "Rare"
    dataset['Title'] = dataset['Title'].map(title_mapping)
    dataset['Title'] = dataset['Title'].fillna(0)

    # Mapping Embarked
    dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q

    # Mapping Fare
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare']
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454)
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31),
    dataset.loc[ dataset['Fare'] > 31, 'Fare']
    dataset['Fare'] = dataset['Fare'].astype(int)

    # Mapping Age
    dataset.loc[ dataset['Age'] <= 16, 'Age']
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age']
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age']
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age']
    dataset.loc[ dataset['Age'] > 64, 'Age'] ;
```

In [48]:

```
# Feature selection: remove variables no longer containing relevant in
drop_elements = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'SibSp']
train_df = train_df.drop(drop_elements, axis = 1)
test_df  = test_df.drop(drop_elements, axis = 1)
```

In [49]:

```
train_df.head()
```

Out[49]:

	Survived	Pclass	Sex	Age	Parch	Fare	Embarked	Has_Cabin	FamilySize	IsAlone	Title
0	0	3	1	1	0	0	0	0	2	0	1
1	1	1	0	2	0	3	1	1	2	0	3
2	1	3	0	1	0	1	0	0	1	1	4
3	1	1	0	2	0	3	0	1	2	0	3
4	0	3	1	2	0	1	0	0	1	1	1

In [50]:

```
def get_gini_impurity(survived_count, total_count):
    survival_prob = survived_count/total_count
    not_survival_prob = (1 - survival_prob)
    random_observation_survived_prob = survival_prob
    random_observation_not_survived_prob = (1 - random_observation_sur
    mislabelling_survided_prob = not_survival_prob * random_observatio
    mislabelling_not_survided_prob = survival_prob * random_observatio
    gini_impurity = mislabelling_survided_prob + mislabelling_not_surv
    return gini_impurity
```

In [51]:

```
# Gini Impurity of starting node
gini_impurity_starting_node = get_gini_impurity(342, 891)
gini_impurity_starting_node
```

Out[51]:

```
0.47301295786144265
```

In [52]:

```
f = train_df.drop("Survived", axis=1)
t = train_df["Survived"]

X_train, X_test, y_train, y_test = train_test_split(f,t,test_size=0.3,
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

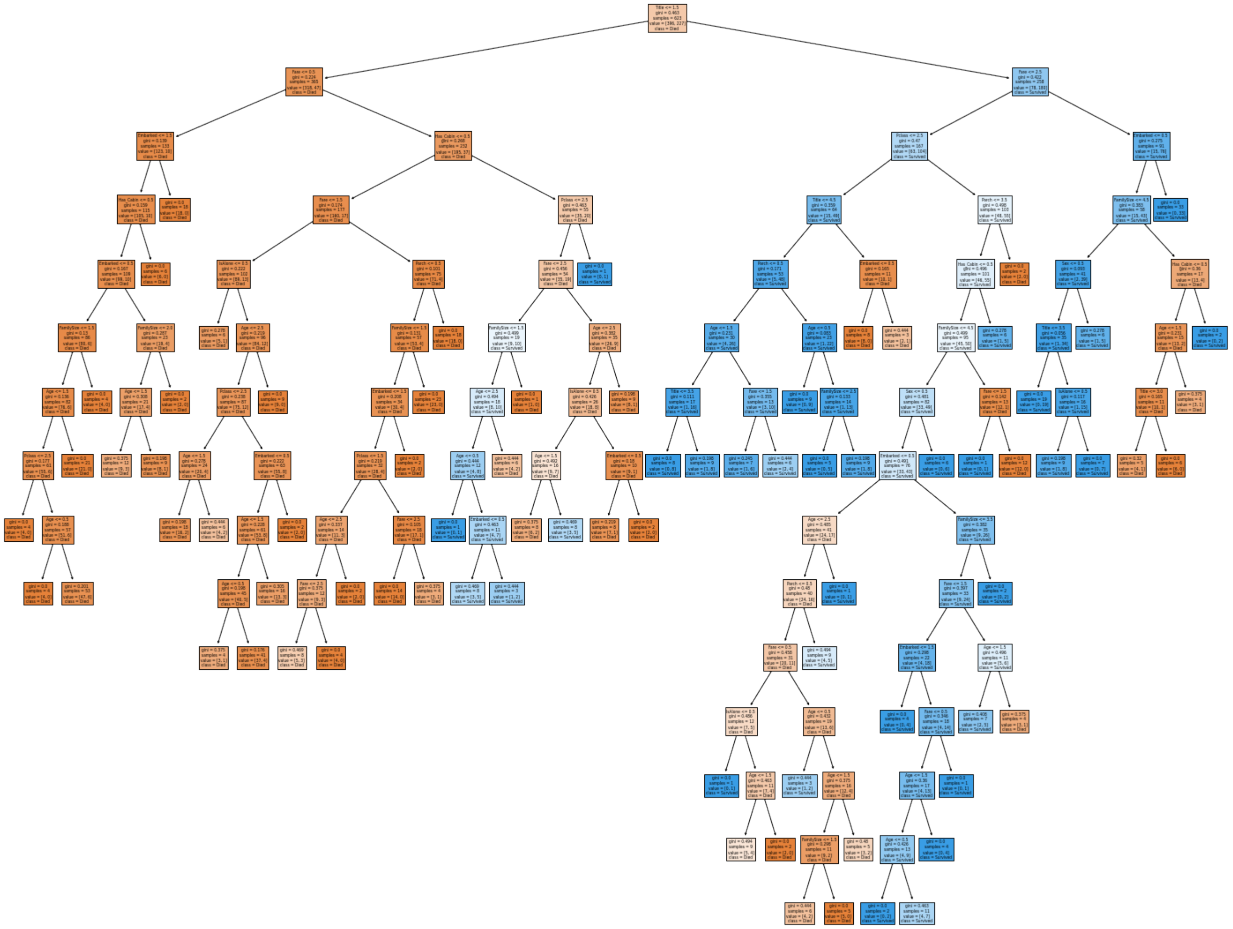
Out[52]:

```
((623, 10), (268, 10), (623,), (268,))
```

In [53]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

model = DecisionTreeClassifier(criterion='gini', min_samples_split=10,
t1 = model.fit(X_train,y_train)

prediction_tree = model.predict(X_test)
print('The accuracy of the DecisionTree Classifier is',round(accuracy_

kfold = KFold(n_splits=5)
result_tree = cross_val_score(model,f,t,cv=5,scoring='accuracy')

print('The cross validated score for Decision Tree classifier is:',rou

fig = plt.figure(figsize=(25,20))
_ = tree.plot_tree(
    model,
    feature_names = list(train_df.drop(['Survived'], axis=1)),
    class_names = ['Died', 'Survived'],
    filled=True)
```

```
The accuracy of the DecisionTree Classifier is 77.99
The cross validated score for Decision Tree classifier is: 79.46
```



In [54]:

```
f = train_df.drop("Survived",axis=1)
t = train_df["Survived"]
X_train,X_test,y_train,y_test = train_test_split(f,t,test_size=0.3,rar
X_train.shape,X_test.shape,y_train.shape,y_test.shape
```

Out[54]:

```
((623, 10), (268, 10), (623,), (268,))
```

In [55]:

```
model = RandomForestClassifier(
    criterion='gini',
    n_estimators=1000,
    min_samples_split=10,
    min_samples_leaf=1,
    max_features='auto',
    oob_score=True,
    n_jobs=-1)

model.fit(X_train,y_train)
prediction_rm=model.predict(X_test)
print('The accuracy of the Random Forest Classifier is', round(accurac
kfold = KFold(n_splits=5)
result_rm=cross_val_score(model,f,t,cv=5,scoring='accuracy')
print('The cross validated score for Random Forest Classifier is:',rou
y_pred = cross_val_predict(model,f,t,cv=5)
```

```
The accuracy of the Random Forest Classifier is 78.73
The cross validated score for Random Forest Classifier is: 82.04
```

In [ ]: