CS156 (Introduction to AI), Spring 2022

Final term project

Roster Name: Andy Luong

Student ID: 014222676

Email address: andy.v.luong@sjsu.edu

Project description/introduction text (the background information)

Add the description of this project. Describe the background of the problem your project is trying to solve. Describe the ML problem you are solving below.

Machine learning algorithm selected for this project

Decision Tree & Random Forest

Dataset source

https://www.kaggle.com/competitions/titanic/overview

References and sources

- IF/ELSE All Values = https://stackoverflow.com/questions/44991438/lambda-including-if-elif-else
- Map Values = https://pandas.pydata.org/docs/reference/api/pandas.Series.map.html
- PLT & SNS = https://machinelearningknowledge.ai/seaborn-heatmap-using-sns-heatmap-with-examples-for-beginners/
- Fill In Null Values = https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html
- Decision Tree = https://scikit-learn.org/stable/modules/tree.html#:~:text=Decision%20Trees%20(DTs)%20are%20a,as%20a%20piecewise%20constant%20approximation.
- Graphviz = https://towardsdatascience.com/visualizing-decision-trees-with-python-scikit-learn-graphviz-matplotlib-1c50b4aa68dc
- Check call & PImage = https://stackoverflow.com/questions/5316206/converting-dot-to-png-in-python
- Google Colab Output File = https://cyublog.com/articles/python-en/colab-pandas-three-ways-to-save-dataframe-data/
- Extra = https://towardsdatascience.com/visualizing-decision-trees-in-jupyter-notebook-with-python-and-graphviz-78703230a7b1

Solution

Load libraries and set random number generator seed

```
import numpy as np
import pandas as pd
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from subprocess import check_call
from IPython.display import Image as PImage
from google.colab import files
import graphviz
import io
import seaborn as sns
import matplotlib.pyplot as plt

np.random.seed(42)
```

Code the solution

###Making Titanic Dataset

```
# Loading the data
uploaded = files.upload()
train = pd.read_csv(io.BytesIO(uploaded['train.csv']))
uploaded2 = files.upload()
test = pd.read_csv(io.BytesIO(uploaded2['test.csv']))
```

```
<IPython.core.display.HTML object>

Saving train.csv to train (2).csv

<IPython.core.display.HTML object>

Saving test.csv to test (2).csv
```

```
train.head()
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age
SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0
1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                             Heikkinen, Miss. Laina  female  26.0
```

```
0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                            Allen, Mr. William Henry    male  35.0
0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

```
test.head() #test does not know who survives
```

```
   PassengerId  Pclass                                          Name
Sex  \
0          892       3                              Kelly, Mr. James
male
1          893       3              Wilkes, Mrs. James (Ellen Needs)
female
2          894       2                     Myles, Mr. Thomas Francis
male
3          895       3                              Wirz, Mr. Albert
male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)
female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked
0  34.5      0      0   330911   7.8292   NaN        Q
1  47.0      1      0   363272   7.0000   NaN        S
2  62.0      0      0   240276   9.6875   NaN        Q
3  27.0      0      0   315154   8.6625   NaN        S
4  22.0      1      1  3101298  12.2875   NaN        S
```

```
#Testing Use
train2 = train.copy()
test2 = test.copy()
```

###Add Columns, Fix Nulls, Remove Unnecessary Variables, & Plot Seaborne

```
#Testing Use
train = train2.copy()
test = test2.copy()

#Adds a column where...
#value is 1 if there is a cabin, 0 otherwise
train['Has_Cabin'] = train["Cabin"].apply(lambda x: 0 if type(x) ==
float else 1)
#it combines SibSp and Parch to find family size
```

```
train['Family_Size'] = train['SibSp'] + train['Parch'] + 1
train
```

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name     Sex   Age
SibSp  \
0                              Braund, Mr. Owen Harris    male  22.0
1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                               Heikkinen, Miss. Laina  female  26.0
0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                             Allen, Mr. William Henry    male  35.0
0
..                                                 ...     ...   ...
...
886                              Montvila, Rev. Juozas    male  27.0
0
887                       Graham, Miss. Margaret Edith  female  19.0
0
888           Johnston, Miss. Catherine Helen "Carrie"  female   NaN
1
889                              Behr, Mr. Karl Howell    male  26.0
0
890                                Dooley, Mr. Patrick    male  32.0
0

     Parch            Ticket     Fare Cabin Embarked  Has_Cabin
Family_Size
0        0         A/5 21171   7.2500   NaN        S          0
2
1        0          PC 17599  71.2833   C85        C          1
2
2        0  STON/O2. 3101282   7.9250   NaN        S          0
1
3        0            113803  53.1000  C123        S          1
```

```
2
4        0            373450   8.0500   NaN        S          0
1
..     ...               ...      ...   ...      ...        ...
...
886      0            211536  13.0000   NaN        S          0
1
887      0            112053  30.0000   B42        S          1
1
888      2        W./C. 6607  23.4500   NaN        S          0
4
889      0            111369  30.0000  C148        C          1
1
890      0            370376   7.7500   NaN        Q          0
1

[891 rows x 14 columns]
```

```
#Replaces Null Values in Embarked, Fare, & Age With the Average Value
train['Embarked'] = train['Embarked'].fillna('S')
train['Fare'] = train['Fare'].fillna(train['Fare'].median())
train['Age'] = train['Age'].fillna(train['Age'].median())
train
```

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name     Sex   Age
SibSp  \
0                              Braund, Mr. Owen Harris    male  22.0
1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
1
2                               Heikkinen, Miss. Laina  female  26.0
0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
1
4                             Allen, Mr. William Henry    male  35.0
0
..                                                 ...     ...   ...
```

```
...
886                              Montvila, Rev. Juozas    male  27.0
0
887                       Graham, Miss. Margaret Edith  female  19.0
0
888           Johnston, Miss. Catherine Helen "Carrie"  female  28.0
1
889                              Behr, Mr. Karl Howell    male  26.0
0
890                                Dooley, Mr. Patrick    male  32.0
0

     Parch            Ticket     Fare Cabin Embarked  Has_Cabin
Family_Size
0        0         A/5 21171   7.2500   NaN        S          0
2
1        0          PC 17599  71.2833   C85        C          1
2
2        0  STON/O2. 3101282   7.9250   NaN        S          0
1
3        0            113803  53.1000  C123        S          1
2
4        0            373450   8.0500   NaN        S          0
1
..     ...               ...      ...   ...      ...        ...
...
886      0            211536  13.0000   NaN        S          0
1
887      0            112053  30.0000   B42        S          1
1
888      2        W./C. 6607  23.4500   NaN        S          0
4
889      0            111369  30.0000  C148        C          1
1
890      0            370376   7.7500   NaN        Q          0
1

[891 rows x 14 columns]
```

```
# Map Sex
if (train['Sex'][0] != 0) and (train['Sex'][0] != 1):
  train['Sex'] = train['Sex'].map( {'female': 0, 'male':
1} ).astype(int)

# Map Embarked
if (train['Embarked'][0] != 0) and (train['Embarked'][0] != 1) and
(train['Embarked'][0] != 2):
  train['Embarked'] = train['Embarked'].map( {'S': 0, 'C': 1, 'Q':
2} ).astype(int)
```

```
# Map Fare
fare_min = train['Fare'].min()
fare_q1 = train['Fare'].quantile(.25)
fare_median = train['Fare'].median()
fare_q3 = train['Fare'].quantile(.75)
fare_max = train['Fare'].max()
fare_min, fare_q1, fare_median, fare_q3, fare_max

train.loc[(train['Fare'] >= fare_min) & (train['Fare'] <= fare_q1),
'Fare'] = 0
train.loc[(train['Fare'] > fare_q1) & (train['Fare'] < fare_median),
'Fare'] = 1
train.loc[(train['Fare'] > fare_median) & (train['Fare'] < fare_q3),
'Fare'] = 2
train.loc[(train['Fare'] >= fare_q3) & (train['Fare'] <= fare_max),
'Fare'] = 3

# Map Age
age_min = train['Age'].min()
age_q1 = train['Age'].quantile(.25)
age_median = train['Age'].median()
age_q3 = train['Age'].quantile(.75)
age_max = train['Age'].max()
age_min, age_q1, age_median, age_q3, age_max

train.loc[(train['Age'] >= age_min) & (train['Age'] <= age_q1), 'Age']
= 0
train.loc[(train['Age'] > age_q1) & (train['Age'] < age_median),
'Age'] = 1
train.loc[(train['Age'] > age_median) & (train['Age'] < age_q3),
'Age'] = 2
train.loc[(train['Age'] >= age_q3) & (train['Age'] <= age_max), 'Age']
= 3
train
```

```
     PassengerId  Survived  Pclass  \
0              1         0       3
1              2         1       1
2              3         1       3
3              4         1       1
4              5         0       3
..           ...       ...     ...
886          887         0       2
887          888         1       1
888          889         0       3
889          890         1       1
890          891         0       3

                                                  Name  Sex   Age
SibSp  \
```

```
0                               Braund, Mr. Owen Harris    1   0.0
1
1    Cumings, Mrs. John Bradley (Florence Briggs Th...    0   3.0
1
2                                Heikkinen, Miss. Laina    0   1.0
0
3         Futrelle, Mrs. Jacques Heath (Lily May Peel)    0   3.0
1
4                              Allen, Mr. William Henry    1   3.0
0
..                                                  ...  ...   ...    .
..
886                              Montvila, Rev. Juozas    1   1.0
0
887                        Graham, Miss. Margaret Edith    0   0.0
0
888           Johnston, Miss. Catherine Helen "Carrie"    0  28.0
1
889                              Behr, Mr. Karl Howell    1   1.0
0
890                                Dooley, Mr. Patrick    1   2.0
0

     Parch            Ticket  Fare Cabin  Embarked  Has_Cabin
Family_Size
0        0         A/5 21171   0.0   NaN         0          0
2
1        0          PC 17599   3.0   C85         1          1
2
2        0  STON/O2. 3101282   1.0   NaN         0          0
1
3        0            113803   3.0  C123         0          1
2
4        0            373450   1.0   NaN         0          0
1
..     ...               ...   ...   ...       ...        ...
...
886      0            211536   1.0   NaN         0          0
1
887      0            112053   2.0   B42         0          1
1
888      2        W./C. 6607   2.0   NaN         0          0
4
889      0            111369   2.0  C148         1          1
1
890      0            370376   0.0   NaN         2          0
1

[891 rows x 14 columns]
```

```
#Drop columns that are not needed in calculation
try:
  PassengerId = test['PassengerId'] #Save ID for future use
  dropColumns = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'SibSp']
  train = train.drop(dropColumns, axis = 1)
  train
except:
  print("Already Dropped")

#Do the same for test
#Adds a column where...
#value is 1 if there is a cabin, 0 otherwise
test['Has_Cabin'] = test["Cabin"].apply(lambda x: 0 if type(x) ==
float else 1)
#it combines SibSp and Parch to find family size
test['Family_Size'] = test['SibSp'] + test['Parch'] + 1
#Replaces Null Values in Embarked, Fare, & Age With the Average Value
test['Embarked'] = test['Embarked'].fillna('S')
test['Fare'] = test['Fare'].fillna(test['Fare'].median())
test['Age'] = test['Age'].fillna(test['Age'].median())
test
#Replaces Null Values in Embarked, Fare, & Age With the Average Value
test['Embarked'] = test['Embarked'].fillna('S')
test['Fare'] = test['Fare'].fillna(test['Fare'].median())
test['Age'] = test['Age'].fillna(test['Age'].median())
test
# Map Sex
if (test['Sex'][0] != 0) and (test['Sex'][0] != 1):
 test['Sex'] = test['Sex'].map( {'female': 0, 'male': 1} ).astype(int)

# Map Embarked
if (test['Embarked'][0] != 0) and (test['Embarked'][0] != 1) and
(test['Embarked'][0] != 2):
 test['Embarked'] = test['Embarked'].map( {'S': 0, 'C': 1, 'Q':
2} ).astype(int)

# Map Fare
fare_min = test['Fare'].min()
fare_q1 = test['Fare'].quantile(.25)
fare_median = test['Fare'].median()
fare_q3 = test['Fare'].quantile(.75)
fare_max = test['Fare'].max()
fare_min, fare_q1, fare_median, fare_q3, fare_max

test.loc[(test['Fare'] >= fare_min) & (test['Fare'] <= fare_q1),
'Fare'] = 0
test.loc[(test['Fare'] > fare_q1) & (test['Fare'] < fare_median),
'Fare'] = 1
test.loc[(test['Fare'] > fare_median) & (test['Fare'] < fare_q3),
'Fare'] = 2
```

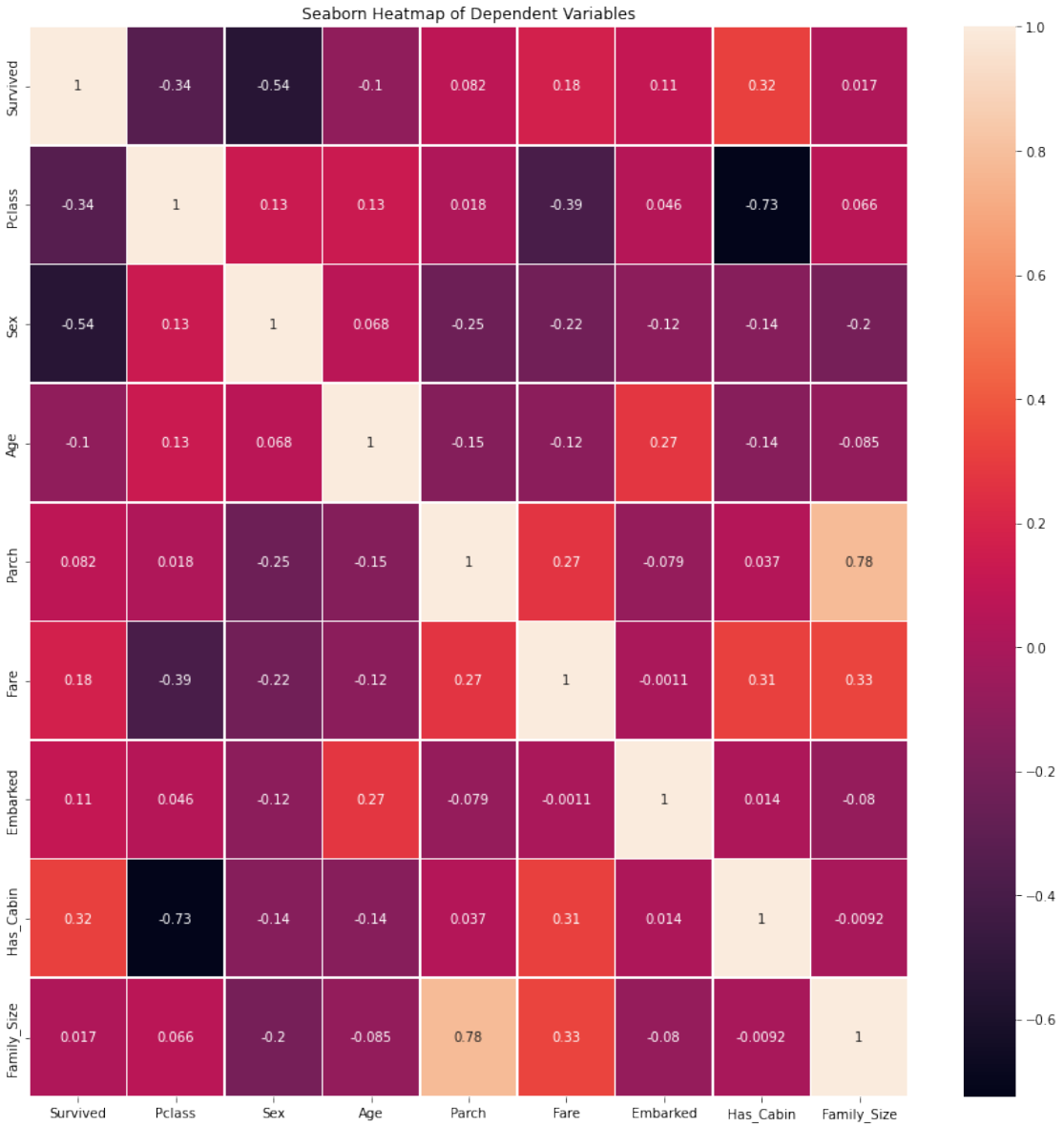
```
test.loc[(test['Fare'] >= fare_q3) & (test['Fare'] <= fare_max),
'Fare'] = 3

# Map Age
age_min = test['Age'].min()
age_q1 = test['Age'].quantile(.25)
age_median = test['Age'].median()
age_q3 = test['Age'].quantile(.75)
age_max = test['Age'].max()
age_min, age_q1, age_median, age_q3, age_max

test.loc[(test['Age'] >= age_min) & (test['Age'] <= age_q1), 'Age'] =
0
test.loc[(test['Age'] > age_q1) & (test['Age'] < age_median), 'Age'] =
1
test.loc[(test['Age'] > age_median) & (test['Age'] < age_q3), 'Age'] =
2
test.loc[(test['Age'] >= age_q3) & (test['Age'] <= age_max), 'Age'] =
3
test
#Drop columns that are not needed in calculation
try:
 dropColumns = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'SibSp']
 test = test.drop(dropColumns, axis = 1)
 test
except:
 print("Already Dropped")

plt.figure(figsize=(15,15))
plt.title('Seaborn Heatmap of Dependent Variables')
sns.heatmap(train.astype(float).corr(),linewidths=0.6, annot=True)

<matplotlib.axes._subplots.AxesSubplot at 0x7f6a01e6ad50>
```



###Survival Rate For Each Variable

```
#mean = survival rate, count = total people, sum = survivors
train[['Pclass', 'Survived']].groupby(['Pclass'],
as_index=False).agg(['mean', 'count', 'sum'])
```

```
        Survived
            mean count  sum
Pclass
1       0.629630   216  136
2       0.472826   184   87
3       0.242363   491  119
```

```
#mean = survival rate, count = total people, sum = survivors
train[['Sex', 'Survived']].groupby(['Sex'],
as_index=False).agg(['mean', 'count', 'sum'])
```

```
     Survived
         mean count  sum
Sex
0    0.742038   314  233
1    0.188908   577  109
```

```
train[['Age', 'Survived']].groupby(['Age'],
as_index=False).agg(['mean', 'count', 'sum'])
```

```
      Survived
          mean count sum
Age
0.0   0.424242   231  98
1.0   0.405660   106  43
2.0   0.410256   117  48
3.0   0.400000   235  94
28.0  0.292079   202  59
```

```
train[['Parch', 'Survived']].groupby(['Parch'],
as_index=False).agg(['mean', 'count', 'sum'])
```

```
       Survived
           mean count  sum
Parch
0      0.343658   678  233
1      0.550847   118   65
2      0.500000    80   40
3      0.600000     5    3
4      0.000000     4    0
5      0.200000     5    1
6      0.000000     1    0
```

```
train[['Embarked', 'Survived']].groupby(['Embarked'],
as_index=False).agg(['mean', 'count', 'sum'])
```

```
          Survived
              mean count  sum
Embarked
0         0.339009   646  219
1         0.553571   168   93
2         0.389610    77   30
```

```
train[['Has_Cabin', 'Survived']].groupby(['Has_Cabin'],
as_index=False).agg(['mean', 'count', 'sum'])
```

```
           Survived
               mean count  sum
Has_Cabin
```

```
0          0.299854   687  206
1          0.666667   204  136
```

```
train[['Family_Size', 'Survived']].groupby(['Family_Size'],
as_index=False).agg(['mean', 'count', 'sum'])
```

```
             Survived
                 mean count  sum
Family_Size
1            0.303538   537  163
2            0.552795   161   89
3            0.578431   102   59
4            0.724138    29   21
5            0.200000    15    3
6            0.136364    22    3
7            0.333333    12    4
8            0.000000     6    0
11           0.000000     7    0
```

###Decision Tree

```
#Use Decision Tree
accuracyList = list()
for x in range(1, 12):
    accuracyFold = []
    kf = KFold(n_splits=10)
    clf = tree.DecisionTreeClassifier(max_depth = x)
    for trainFold, testFold in kf.split(train):
        trainData = train.loc[trainFold]
        testData = train.loc[testFold]
        model = clf.fit(X = trainData.drop(['Survived'], axis=1), y =
trainData["Survived"])
        testAccount = model.score(X = testData.drop(['Survived'],
axis=1), y = testData["Survived"])
        accuracyFold.append(testAccount)
    accuracyList.append(sum(accuracyFold)/len(accuracyFold))

df = pd.DataFrame({"Max Depth":range(1,12), "Average Accuracy":
accuracyList})
df = df[["Max Depth", "Average Accuracy"]]
df
```

```
    Max Depth  Average Accuracy
0           1          0.786729
1           2          0.771099
2           3          0.797990
3           4          0.805855
4           5          0.805843
5           6          0.806979
6           7          0.809201
7           8          0.811473
```

```
8           9          0.800237
9          10          0.786804
10         11          0.793496
```

```
y_train = train['Survived']
x_train = train.drop(['Survived'], axis=1).values
x_test = test.values

#Checks with max_depth should be used
Max_depth = 0;
for row in df.index:
  if df['Average Accuracy'][row] == max(accuracyList):
    Max_depth = row

# Decision with max_depth
decision_tree = tree.DecisionTreeClassifier(max_depth = Max_depth)
decision_tree.fit(x_train, y_train)

# Predicting results for test dataset
prediction = decision_tree.predict(x_test)
submission = pd.DataFrame({
        "PassengerId": PassengerId,
        "Survived": prediction
    })

survivors = 0
for x in submission['Survived']:
  if (x == 1):
    survivors = survivors + 1
print(str(survivors) + " will survive out of " +
str(submission.index.size))
```

```
168 will survive out of 418
```

```
#Save to file
submission.to_csv('submission.csv', encoding = 'utf-8-sig')
files.download('submission.csv')
```

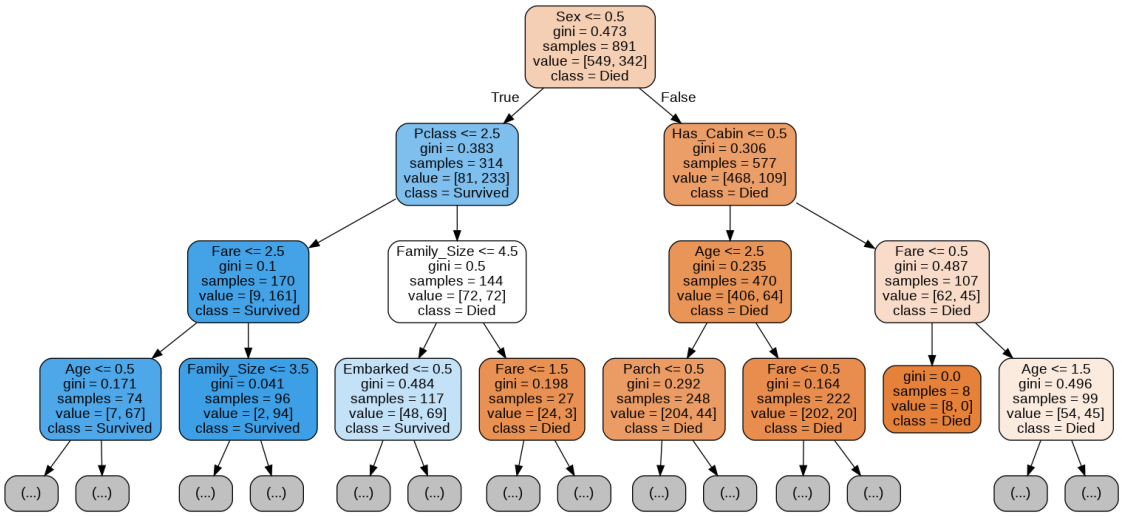
```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
dot_data = tree.export_graphviz(decision_tree, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("Titanic")
dot_data = tree.export_graphviz(decision_tree, out_file=None,
                      feature_names = list(train.drop(['Survived'],
axis=1)),
                      class_names = ['Died', 'Survived'],
                     filled=True, rounded=True,
                     special_characters=True)
```

```
graph = graphviz.Source(dot_data)

#Display Graph as Image
img = graph.save('image.png')
check_call(['dot','-Tpng','tree1.dot','-o','image.png'])
PImage(img)
```




```
totalAccuracyScore = decision_tree.score(x_train, y_train)
totalAccuracyScore
```

```
0.8653198653198653
```

###Random Forest Classifier

```
#Use Random Forest Tree
accuracyList = list()
for x in range(1, 12):
    accuracyFold = []
    kf = KFold(n_splits=10)
    clf = RandomForestClassifier(max_depth = x, random_state = 0)
    for trainFold, testFold in kf.split(train):
        trainData = train.loc[trainFold]
        testData = train.loc[testFold]
        model = clf.fit(X = trainData.drop(['Survived'], axis=1), y =
trainData["Survived"])
        testAccount = model.score(X = testData.drop(['Survived'],
axis=1), y = testData["Survived"])
        accuracyFold.append(testAccount)
    accuracyList.append(sum(accuracyFold)/len(accuracyFold))

df = pd.DataFrame({"Max Depth":range(1,12), "Average Accuracy":
accuracyList})
df = df[["Max Depth", "Average Accuracy"]]
df
```

```
    Max Depth  Average Accuracy
0           1          0.741935
1           2          0.777890
2           3          0.778989
3           4          0.800300
4           5          0.805868
5           6          0.806979
6           7          0.811461
7           8          0.808065
8           9          0.803620
9          10          0.809226
10         11          0.814844
```

```
y_train = train['Survived']
x_train = train.drop(['Survived'], axis=1).values
x_test = test.values

#Checks with max_depth should be used
Max_depth = 0;
for row in df.index:
  if df['Average Accuracy'][row] == max(accuracyList):
    Max_depth = row

# Decision with max_depth
randomForest = RandomForestClassifier(max_depth = Max_depth)
randomForest.fit(x_train, y_train)

# Predicting results for test dataset
prediction = decision_tree.predict(x_test)
submission = pd.DataFrame({
        "PassengerId": PassengerId,
        "Survived": prediction
    })

survivors = 0
for x in submission['Survived']:
  if (x == 1):
    survivors = survivors + 1
print(str(survivors) + " will survive out of " +
str(submission.index.size))
```

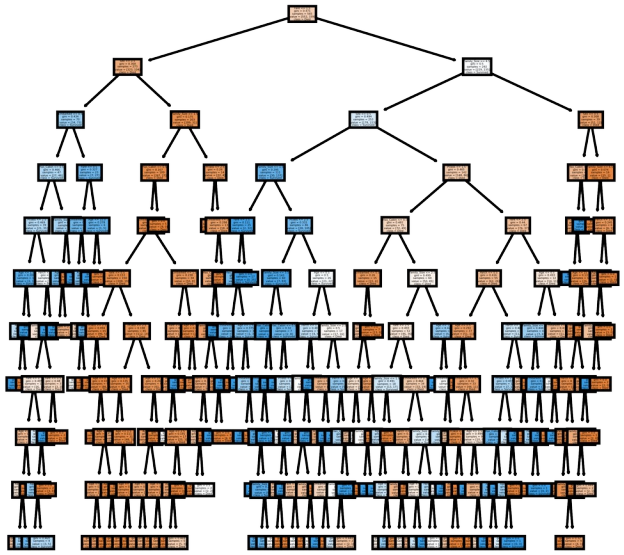
```
168 will survive out of 418
```

```
#Save to file
submission.to_csv('submission2.csv', encoding = 'utf-8-sig')
files.download('submission2.csv')
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=800)
tree.plot_tree(randomForest.estimators_[0],
                   feature_names = list(train.drop(['Survived'],
axis=1)),
                   class_names = ['Died', 'Survived'],
                filled = True);
fig.savefig('rf_individualtree.png')
```



```
totalAccuracyScore = randomForest.score(x_train, y_train)
totalAccuracyScore
```

```
0.8945005611672279
```