NAME: RADHIKA GUPTA

REG. NO: 22MCA1119

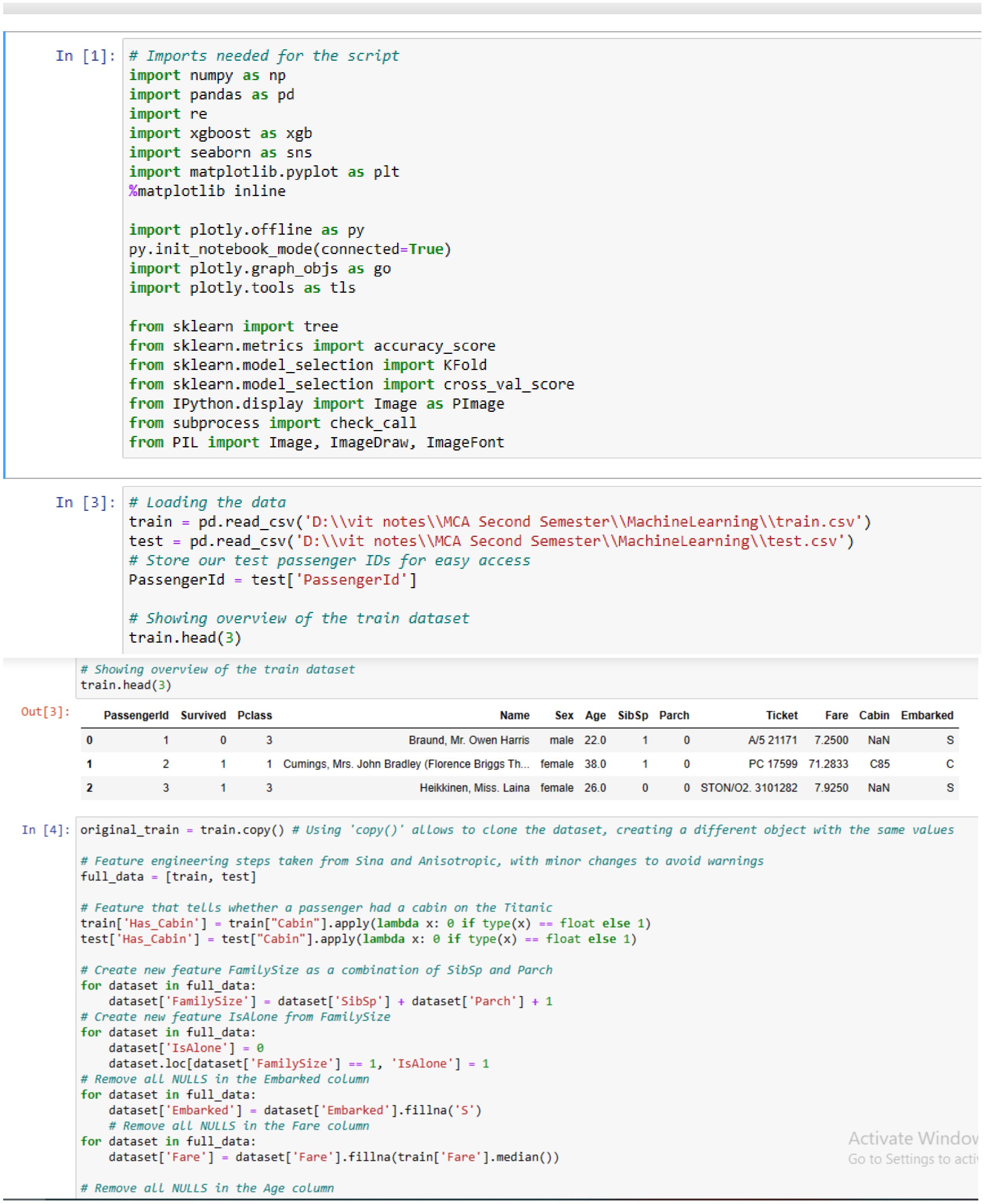
ITA-6016 Machine Learning

Digital Assignment –Lab -II

SUBMITTED TO: Dr Dominic Savio M

DECISION TREE ALGORITHM (ID3):

CODE OF THE PROGRAM AND OUTPUT:



```
In [1]: # Imports needed for the script
        import numpy as np
        import pandas as pd
        import re
        import xgboost as xgb
        import seaborn as sns
        import matplotlib.pyplot as plt
        %matplotlib inline

        import plotly.offline as py
        py.init_notebook_mode(connected=True)
        import plotly.graph_objs as go
        import plotly.tools as tls

        from sklearn import tree
        from sklearn.metrics import accuracy_score
        from sklearn.model_selection import KFold
        from sklearn.model_selection import cross_val_score
        from IPython.display import Image as PImage
        from subprocess import check_call
        from PIL import Image, ImageDraw, ImageFont
```

```
In [3]: # Loading the data
        train = pd.read_csv('D:\\vit notes\\MCA Second Semester\\MachineLearning\\train.csv')
        test = pd.read_csv('D:\\vit notes\\MCA Second Semester\\MachineLearning\\test.csv')
        # Store our test passenger IDs for easy access
        PassengerId = test['PassengerId']

        # Showing overview of the train dataset
        train.head(3)
```

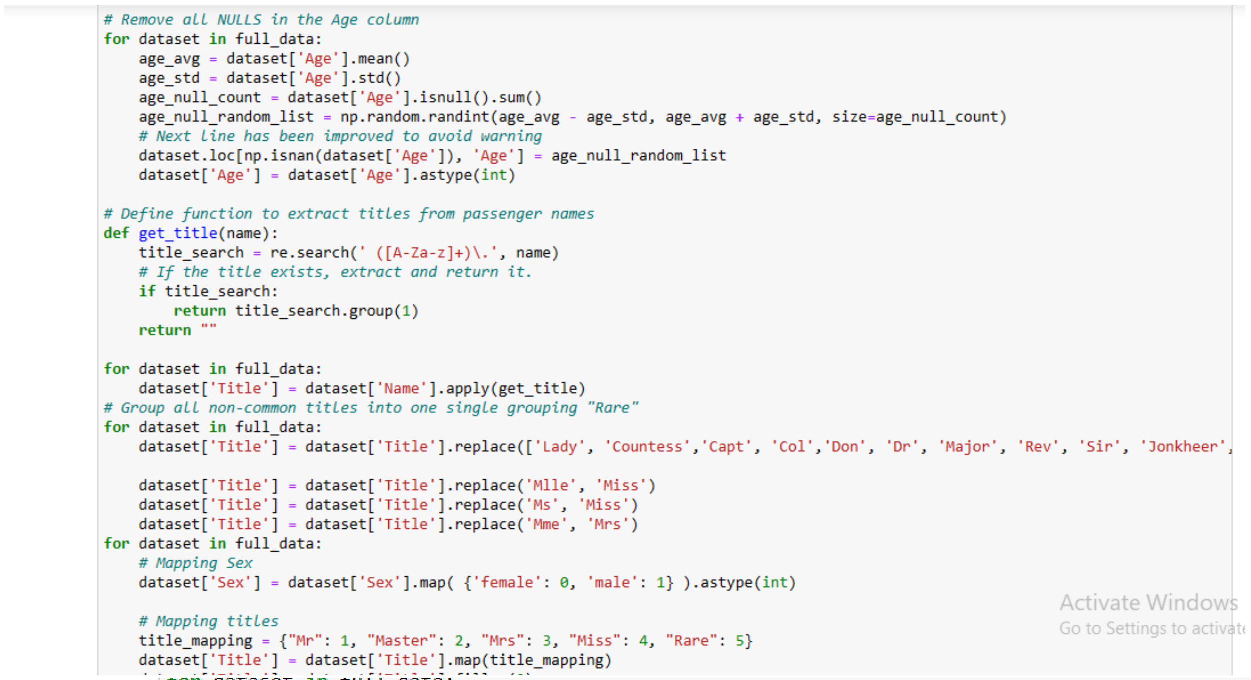
```
# Showing overview of the train dataset
train.head(3)
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

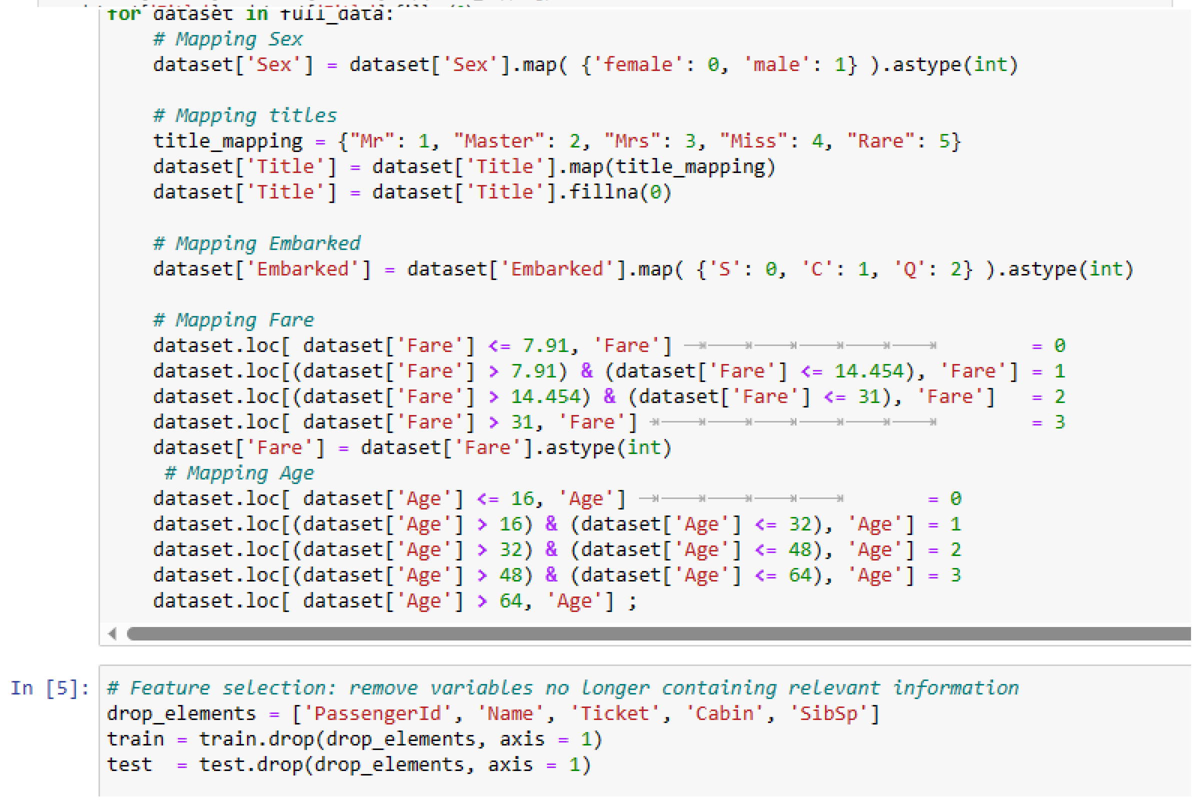
```
In [4]: original_train = train.copy() # Using 'copy()' allows to clone the dataset, creating a different object with the same values

        # Feature engineering steps taken from Sina and Anisotropic, with minor changes to avoid warnings
        full_data = [train, test]

        # Feature that tells whether a passenger had a cabin on the Titanic
        train['Has_Cabin'] = train["Cabin"].apply(lambda x: 0 if type(x) == float else 1)
        test['Has_Cabin'] = test["Cabin"].apply(lambda x: 0 if type(x) == float else 1)

        # 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['Fare'].median())

        # Remove all NULLS in the Age column
```

Activate Window
Go to Settings to acti



```python
# 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_avg + age_std, size=age_null_count)
    # Next line has been improved to avoid warning
    dataset.loc[np.isnan(dataset['Age']), 'Age'] = age_null_random_list
    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','Capt', 'Col','Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer',

    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} ).astype(int)

    # Mapping titles
    title_mapping = {"Mr": 1, "Master": 2, "Mrs": 3, "Miss": 4, "Rare": 5}
    dataset['Title'] = dataset['Title'].map(title_mapping)
```

Activate Windows
Go to Settings to activate

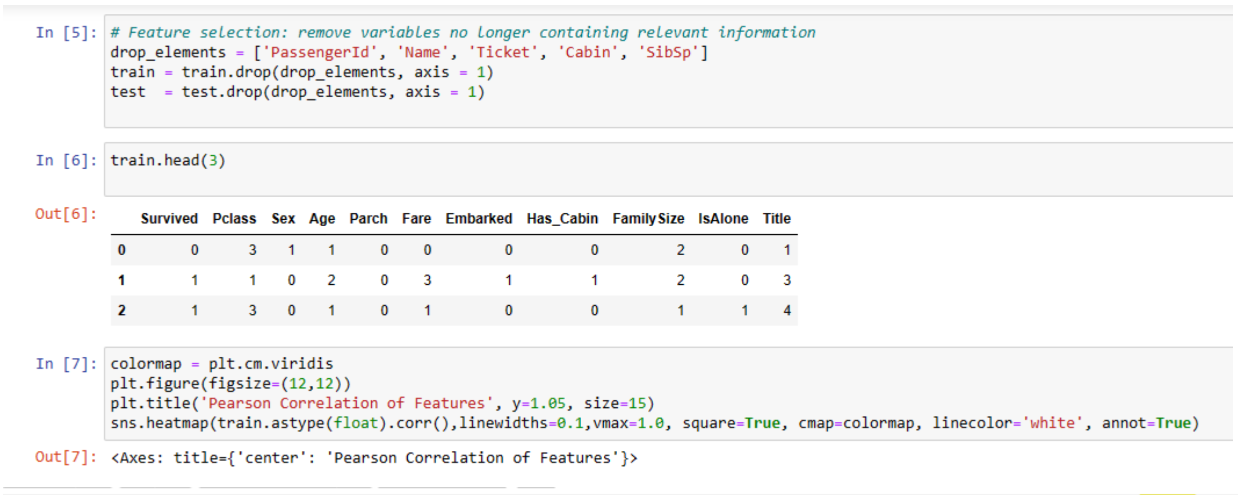


```python
for dataset in full_data:
    # Mapping Sex
    dataset['Sex'] = dataset['Sex'].map( {'female': 0, 'male': 1} ).astype(int)

    # Mapping titles
    title_mapping = {"Mr": 1, "Master": 2, "Mrs": 3, "Miss": 4, "Rare": 5}
    dataset['Title'] = dataset['Title'].map(title_mapping)
```

```python
    dataset['Title'] = dataset['Title'].fillna(0)

    # Mapping Embarked
    dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)

    # Mapping Fare
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare']                                = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare']   = 2
    dataset.loc[ dataset['Fare'] > 31, 'Fare']                                   = 3
    dataset['Fare'] = dataset['Fare'].astype(int)
     # Mapping Age
    dataset.loc[ dataset['Age'] <= 16, 'Age']                          = 0
    dataset.loc[(dataset['Age'] > 16) & (dataset['Age'] <= 32), 'Age'] = 1
    dataset.loc[(dataset['Age'] > 32) & (dataset['Age'] <= 48), 'Age'] = 2
    dataset.loc[(dataset['Age'] > 48) & (dataset['Age'] <= 64), 'Age'] = 3
    dataset.loc[ dataset['Age'] > 64, 'Age'] ;
```

In [5]:

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



```
In [5]: # Feature selection: remove variables no longer containing relevant information
        drop_elements = ['PassengerId', 'Name', 'Ticket', 'Cabin', 'SibSp']
        train = train.drop(drop_elements, axis = 1)
        test  = test.drop(drop_elements, axis = 1)
```

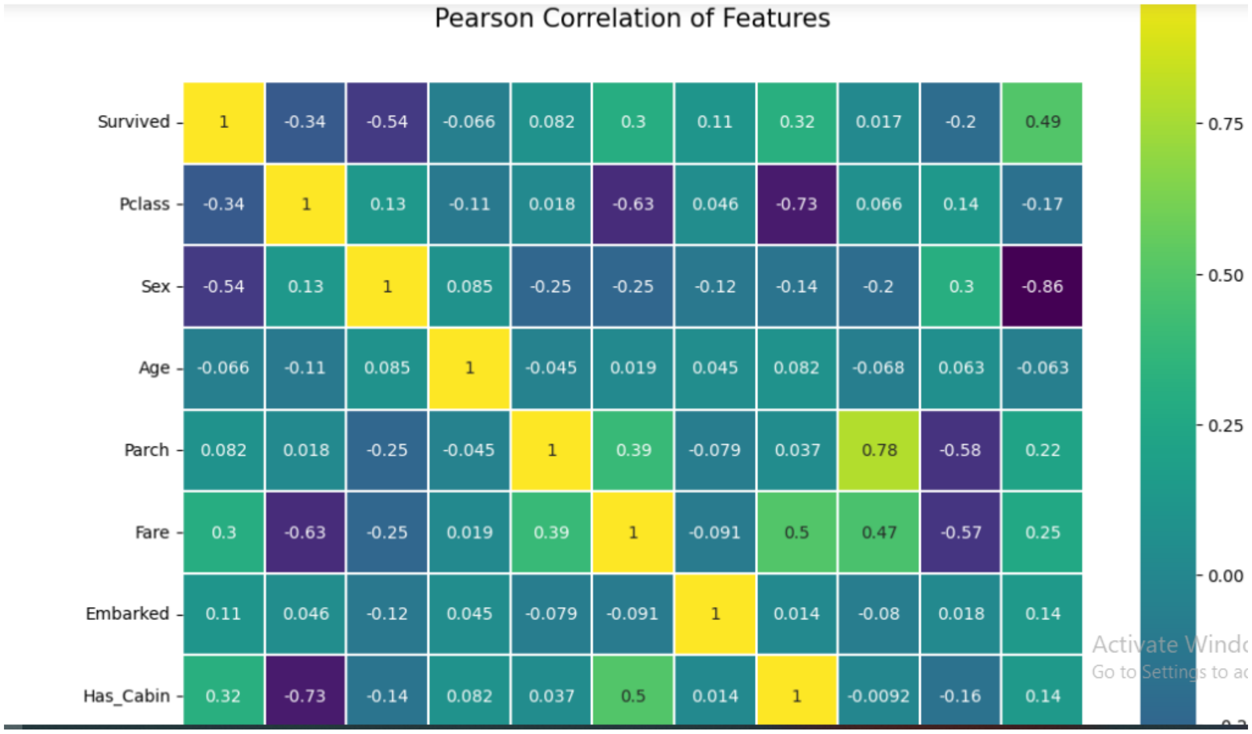
```
In [6]: train.head(3)
```

Out[6]:

	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

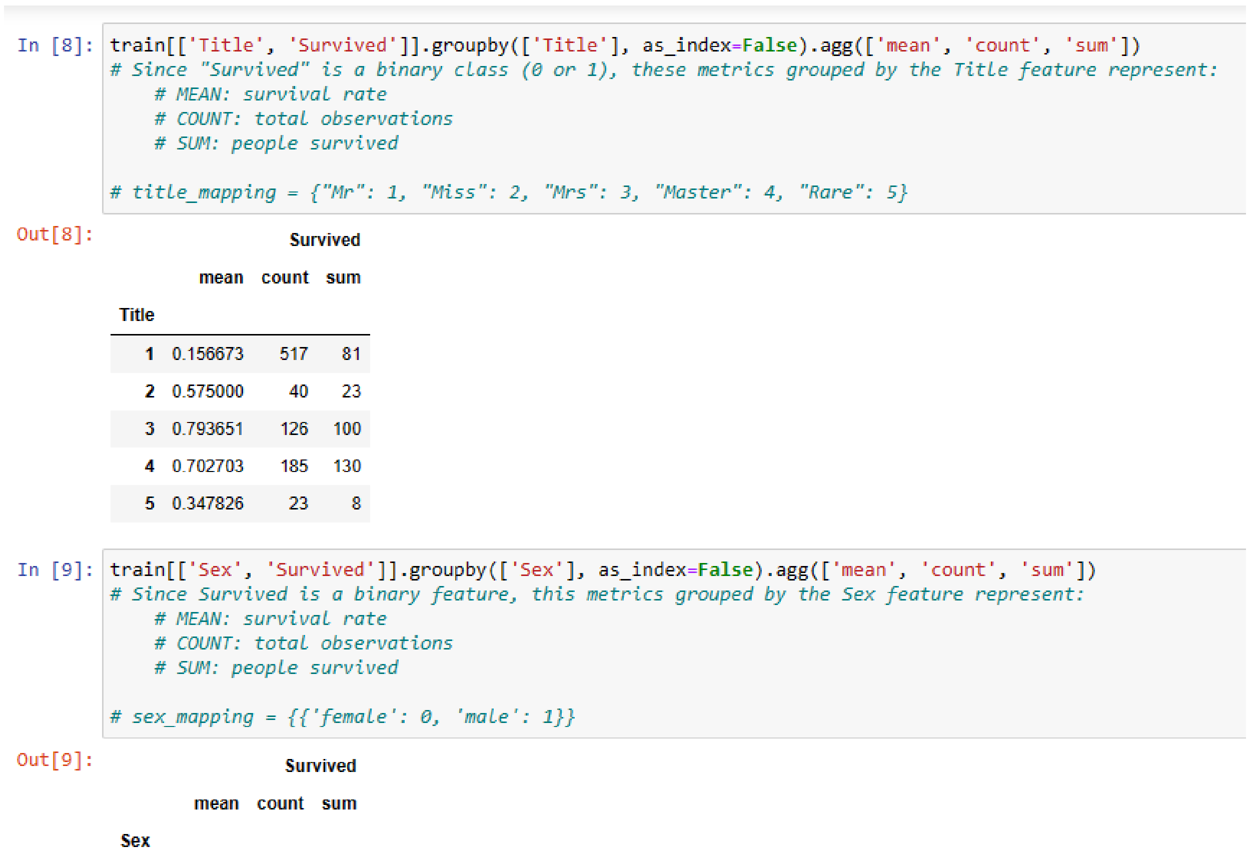
```
In [7]: colormap = plt.cm.viridis
        plt.figure(figsize=(12,12))
        plt.title('Pearson Correlation of Features', y=1.05, size=15)
        sns.heatmap(train.astype(float).corr(),linewidths=0.1,vmax=1.0, square=True, cmap=colormap, linecolor='white', annot=True)
```

```
Out[7]: <Axes: title={'center': 'Pearson Correlation of Features'}>
```



Survived	1	-0.34	-0.54	-0.066	0.082	0.3	0.11	0.32	0.017	-0.2	0.49
Pclass	-0.34	1	0.13	-0.11	0.018	-0.63	0.046	-0.73	0.066	0.14	-0.17
Sex	-0.54	0.13	1	0.085	-0.25	-0.25	-0.12	-0.14	-0.2	0.3	-0.86
Age	-0.066	-0.11	0.085	1	-0.045	0.019	0.045	0.082	-0.068	0.063	-0.063
Parch	0.082	0.018	-0.25	-0.045	1	0.39	-0.079	0.037	0.78	-0.58	0.22
Fare	0.3	-0.63	-0.25	0.019	0.39	1	-0.091	0.5	0.47	-0.57	0.25
Embarked	0.11	0.046	-0.12	0.045	-0.079	-0.091	1	0.014	-0.08	0.018	0.14
Has_Cabin	0.32	-0.73	-0.14	0.082	0.037	0.5	0.014	1	-0.0092	-0.16	0.14

Activate Windo
Go to Settings to ac



In [8]:
```python
train[['Title', 'Survived']].groupby(['Title'], as_index=False).agg(['mean', 'count', 'sum'])
# Since "Survived" is a binary class (0 or 1), these metrics grouped by the Title feature represent:
    # MEAN: survival rate
    # COUNT: total observations
    # SUM: people survived

# title_mapping = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}
```

Out[8]:
```
                    Survived
           mean     count  sum
Title
    1  0.156673      517    81
    2  0.575000       40    23
    3  0.793651      126   100
    4  0.702703      185   130
    5  0.347826       23     8
```

In [9]:
```python
train[['Sex', 'Survived']].groupby(['Sex'], as_index=False).agg(['mean', 'count', 'sum'])
# Since Survived is a binary feature, this metrics grouped by the Sex feature represent:
    # MEAN: survival rate
    # COUNT: total observations
    # SUM: people survived

# sex_mapping = {{'female': 0, 'male': 1}}
```

Out[9]:
```
                    Survived
           mean     count  sum
Sex
```

Out[9]:

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

In [10]:
```
# Let's use our 'original_train' dataframe to check the sex distribution for each title.
# We use copy() again to prevent modifications in out original_train dataset
title_and_sex = original_train.copy()[['Name', 'Sex']]

# Create 'Title' feature
title_and_sex['Title'] = title_and_sex['Name'].apply(get_title)

# Map 'Sex' as binary feature
title_and_sex['Sex'] = title_and_sex['Sex'].map( {'female': 0, 'male': 1} ).astype(int)

# Table with 'Sex' distribution grouped by 'Title'
title_and_sex[['Title', 'Sex']].groupby(['Title'], as_index=False).agg(['mean', 'count', 'sum'])
```

Out[10]:

	Sex		
	mean	count	sum
Title			
Capt	1.000000	1	1
Col	1.000000	2	2
Countess	0.000000	1	0
Don	1.000000	1	1
Dr	0.857143	7	6

In [11]:
```
# Define function to calculate Gini Impurity
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_survived_prob)
    mislabelling_survided_prob = not_survival_prob * random_observation_survived_prob
    mislabelling_not_survided_prob = survival_prob * random_observation_not_survived_prob
    gini_impurity = mislabelling_survided_prob + mislabelling_not_survided_prob
    return gini_impurity
```

In [12]:
```
# Gini Impurity of starting node
gini_impurity_starting_node = get_gini_impurity(342, 891)
gini_impurity_starting_node
```

Out[12]:
```
0.47301295786144265
```

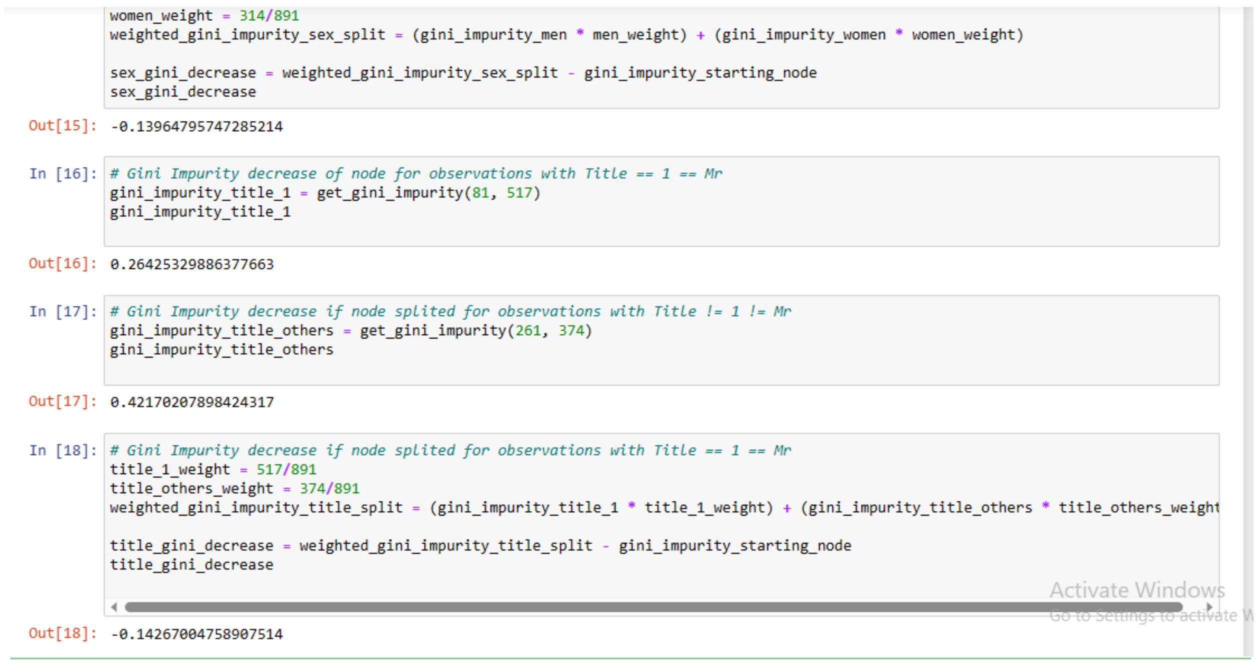
In [13]:
```
# Gini Impurity decrease of node for 'male' observations
gini_impurity_men = get_gini_impurity(109, 577)
gini_impurity_men
```

Out[13]:
```
0.3064437162277843
```

In [14]:
```
# Gini Impurity decrease if node splited for 'female' observations
gini_impurity_women = get_gini_impurity(233, 314)
gini_impurity_women
```

Out[14]:
```
0.3828350034484158
```

In [15]:
```
# Gini Impurity decrease if node splited by Sex
men_weight = 577/891
women_weight = 314/891
weighted_gini_impurity_sex_split = (gini_impurity_men * men_weight) + (gini_impurity_women * women_weight)
```



```
women_weight = 314/891
weighted_gini_impurity_sex_split = (gini_impurity_men * men_weight) + (gini_impurity_women * women_weight)

sex_gini_decrease = weighted_gini_impurity_sex_split - gini_impurity_starting_node
sex_gini_decrease
```

```
Out[15]: -0.13964795747285214
```

```
In [16]: # Gini Impurity decrease of node for observations with Title == 1 == Mr
gini_impurity_title_1 = get_gini_impurity(81, 517)
gini_impurity_title_1
```

```
Out[16]: 0.26425329886377663
```

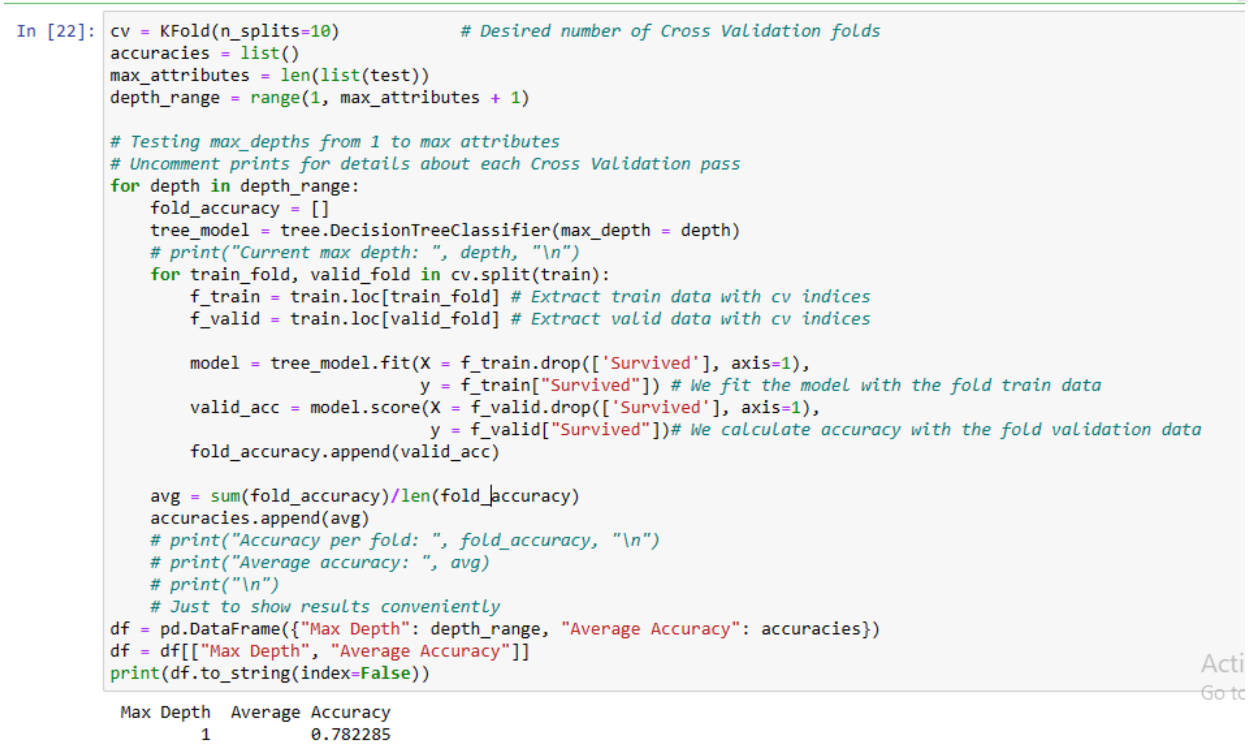
```
In [17]: # Gini Impurity decrease if node splited for observations with Title != 1 != Mr
gini_impurity_title_others = get_gini_impurity(261, 374)
gini_impurity_title_others
```

```
Out[17]: 0.42170207898424317
```

```
In [18]: # Gini Impurity decrease if node splited for observations with Title == 1 == Mr
title_1_weight = 517/891
title_others_weight = 374/891
weighted_gini_impurity_title_split = (gini_impurity_title_1 * title_1_weight) + (gini_impurity_title_others * title_others_weight

title_gini_decrease = weighted_gini_impurity_title_split - gini_impurity_starting_node
title_gini_decrease
```

Activate Windows
Go to Settings to activate W

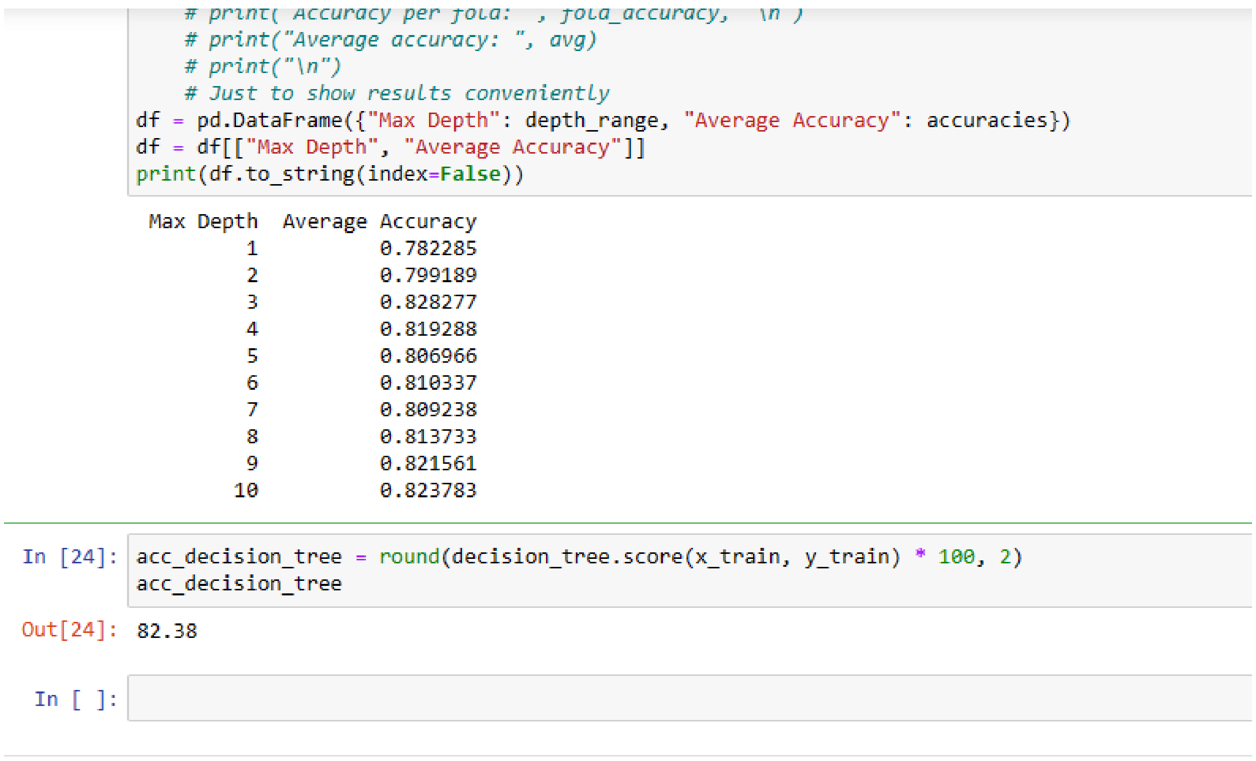
```
Out[18]: -0.14267004758907514
```



```
In [22]: cv = KFold(n_splits=10)            # Desired number of Cross Validation folds
accuracies = list()
max_attributes = len(list(test))
depth_range = range(1, max_attributes + 1)

# Testing max_depths from 1 to max attributes
# Uncomment prints for details about each Cross Validation pass
for depth in depth_range:
    fold_accuracy = []
    tree_model = tree.DecisionTreeClassifier(max_depth = depth)
    # print("Current max depth: ", depth, "\n")
    for train_fold, valid_fold in cv.split(train):
        f_train = train.loc[train_fold] # Extract train data with cv indices
        f_valid = train.loc[valid_fold] # Extract valid data with cv indices

        model = tree_model.fit(X = f_train.drop(['Survived'], axis=1),
                               y = f_train["Survived"]) # We fit the model with the fold train data
        valid_acc = model.score(X = f_valid.drop(['Survived'], axis=1),
                                y = f_valid["Survived"])# We calculate accuracy with the fold validation data
        fold_accuracy.append(valid_acc)

    avg = sum(fold_accuracy)/len(fold_accuracy)
    accuracies.append(avg)
    # print("Accuracy per fold: ", fold_accuracy, "\n")
    # print("Average accuracy: ", avg)
    # print("\n")
    # Just to show results conveniently
df = pd.DataFrame({"Max Depth": depth_range, "Average Accuracy": accuracies})
df = df[["Max Depth", "Average Accuracy"]]
print(df.to_string(index=False))
```

Acti
Go to

```
 Max Depth  Average Accuracy
         1          0.782285
```



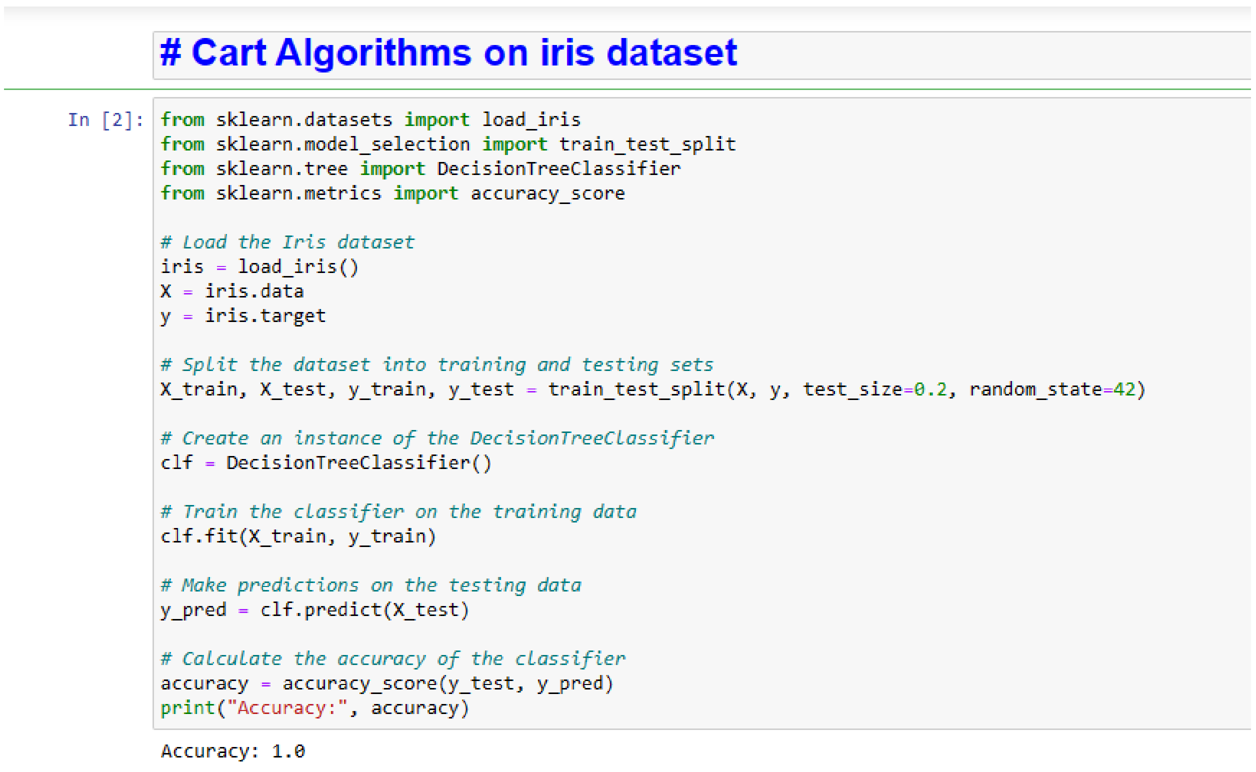
```
    # print( Accuracy per fold:  , fold_accuracy,  \n )
    # print("Average accuracy: ", avg)
    # print("\n")
    # Just to show results conveniently
df = pd.DataFrame({"Max Depth": depth_range, "Average Accuracy": accuracies})
df = df[["Max Depth", "Average Accuracy"]]
print(df.to_string(index=False))
```

```
 Max Depth  Average Accuracy
         1          0.782285
         2          0.799189
         3          0.828277
         4          0.819288
         5          0.806966
         6          0.810337
         7          0.809238
         8          0.813733
         9          0.821561
        10          0.823783
```

```
In [24]: acc_decision_tree = round(decision_tree.score(x_train, y_train) * 100, 2)
         acc_decision_tree

Out[24]: 82.38

In [ ]:
```

<u>CART ALGORITHM (Iris dataset):</u>

CODE OF THE PROGRAM AND OUTPUT:

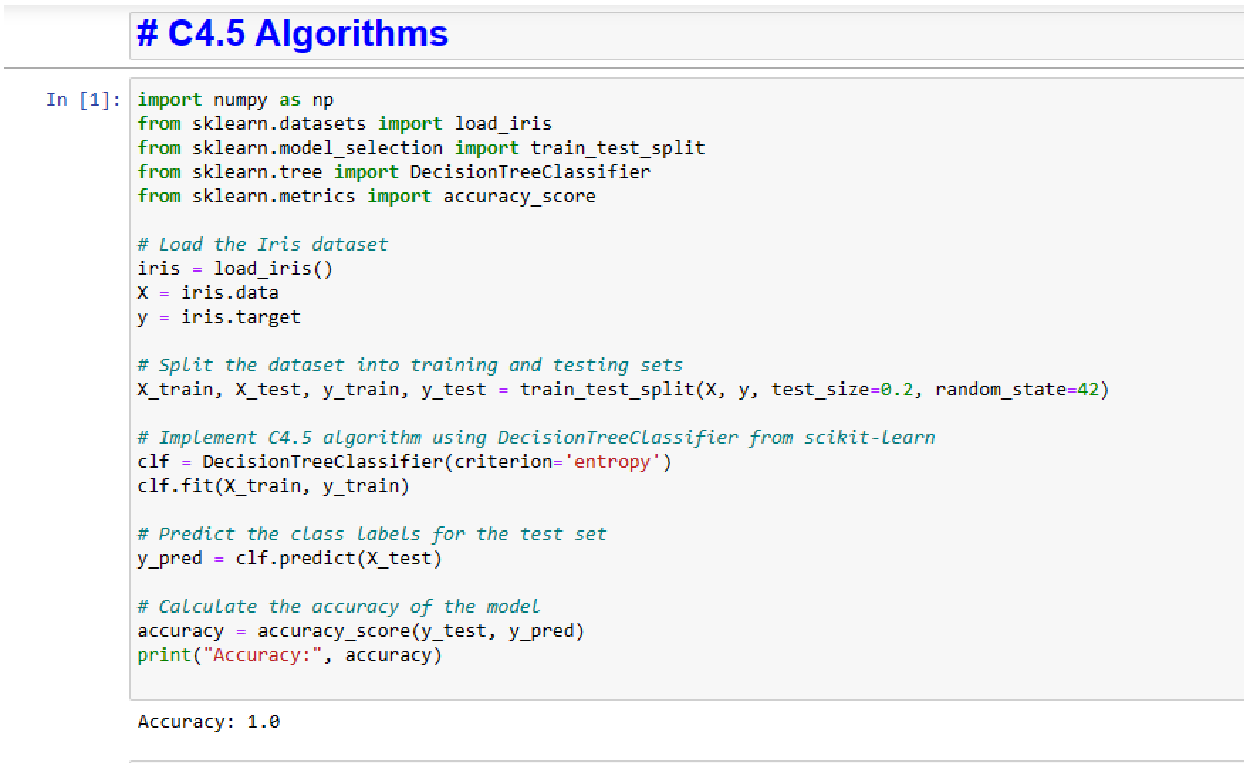

Cart Algorithms on iris dataset

In [2]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create an instance of the DecisionTreeClassifier
clf = DecisionTreeClassifier()

# Train the classifier on the training data
clf.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = clf.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```

C4.5 (Iris dataset):

CODE OF THE PROGRAM AND OUTPUT:



# C4.5 Algorithms

In [1]:
```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Implement C4.5 algorithm using DecisionTreeClassifier from scikit-learn
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)

# Predict the class labels for the test set
y_pred = clf.predict(X_test)

# Calculate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 1.0
```