Task 1:

Let's read the csv file for both training and test data and print all the features that are present in the data set

```
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

Next, let's print the number of empty values present the training dataset.

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

At this point, let us start to preprocess the dataset. We can do this by filling the empty values in Age and Embarked feature.

Let us start by filling the empty values of the Age feature in the training dataset using KNN algorithm, K = 5. Some of the original and processed Age values are:

```
22.0 22.0
38.0 38.0
26.0 26.0
35.0 35.0
35.0 35.0
nan 29.69911764705882
54.0 54.0
2.0 2.0
27.0 27.0
14.0 14.0
4.0 4.0
58.0 58.0
20.0 20.0
39.0 39.0
14.0 14.0
55.0 55.0
2.0 2.0
nan 29.69911764705882
31.0 31.0
nan 29.69911764705882
35.0 35.0
```

Let us fill the empty values in the Embarked feature using mode. The output looks as follows:

```
In the training dataset, The mode for Embarked freature is:  0    S
dtype: object
The number of empty values in Embarked feature is:  0
After filling the empty values, the number of empty values in the
Embarked feature is:  0
The following are the values in the Embarked feature:  0      S
1      C
2      S
3      S
4      S
      ..
886    S
887    S
888    S
889    C
890    Q
Name: Embarked, Length: 891, dtype: object
```

The next step in preprocessing data is to map the 'Sex' feature to binary values. Female = 1 and Male = 0. Output looks as follows:

```
Sex values:
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    1
889    0
890    0
Name: Sex, Length: 891, dtype: int64
```

I also decided to map the Embarked feature into integer values. This makes it easier for analysis purposes. S = 0, C = 1 and Q = 2. Output looks as follows:

```
Embarked values:
0      0
1      1
2      0
3      0
4      0
```

```
       ..
886    0
887    0
888    0
889    1
890    2
Name: Embarked, Length: 891, dtype: int64
```

In the next step of data preprocessing, I have binned Fare feature. The output looks as follows:

```
Fare Bin values:
0      0
1      3
2      1
3      3
4      1
       ..
886    1
887    2
888    2
889    2
890    0
Name: Fare_bin, Length: 891, dtype: category
Categories (4, object): ['0' < '1' < '2' < '3']
```

Once data preprocessing is done, I move into feature engineering. I start by dropping some of the features that I don't think is critical for my analysis.

I dropped the Cabin feature as it has a lot of null/empty values that I won't be able to fill. I also dropped the Fare feature as I have binned the fare values into different ranges using labels. I also decided to drop the Ticket and Name feature as they don't seem to provide any vital information for our analysis.

Once all these features are dropped, the remaining features are:

```
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Age       891 non-null    float64
 1   Survived  891 non-null    int64
 2   Pclass    891 non-null    int64
 3   Sex       891 non-null    int64
 4   SibSp     891 non-null    int64
 5   Parch     891 non-null    int64   
 6   Embarked  891 non-null    int64   
 7   Fare_bin  891 non-null    category
dtypes: category(1), float64(1), int64(6)
memory usage: 49.9 KB
```

Let’s check for any null values in the feature set:

```
Age         0
Survived    0
Pclass      0
Sex         0
SibSp       0
Parch       0
Embarked    0
Fare_bin    0
dtype: int64
```

Since all the features have non-null values, I can move into feature selection. For this, I used two feature selection methods to compare the results.

First, I used SelectKBest method of feature selection.

In this method of feature selection, chi2 is passed as a score function. SelectKBest will compute the chi2 statistic between each feature of X and y. A small value will mean that the feature is independent of y and a large value will mean that the feature is non-randomly related to y, and thus will likely provide important information. Only the first ‘k’ features of X with the highest score will be retained.

The five features that were selected using SelectKBest feature selection are:

```
  Age Pclass Sex Embarked Fare_bin
0  22      3   0        0        0
1  38      1   1        1        3
2  26      3   1        0        1
3  35      1   1        0        3
4  35      3   0        0        1
```

I expected these features to be more vital in our analysis and this method of feature selection selected exactly the five features I was thinking of.

Next, I used Extra Tree Classifier algorithm to check the best five features I would get using this method.

In this method, at each test node, each tree is provided with a random sample of 'k' features from the feature-set, from which each decision tree must select the best features to split the data based on some mathematical criteria (in this case, Gini Index). Gini index measures the degree or probability of a particular feature being wrongly classified when it is randomly chosen.

Based on this feature selection method, the five features that gained importance are Age, Sex, Pclass, Fare_bin and SibSp.

```
[0.33578989 0.11241822 0.31524843 0.06159658 0.05433983 0.04139183
 0.07921522]
    Age  Pclass  Sex  SibSp  Parch  Embarked Fare_bin
0  22.0       3    0      1      0         0        0
1  38.0       1    1      1      0         1        3
2  26.0       3    1      0      0         0        1
3  35.0       1    1      1      0         0        3
4  35.0       3    0      0      0         0        1
```

Once, the feature selection is completed, Decision tree, Generalized tree and Random Forest tree were constructed for the feature set. For these trees, the accuracy score was measured. The criterion used for these tree generation is criterion = "gini".

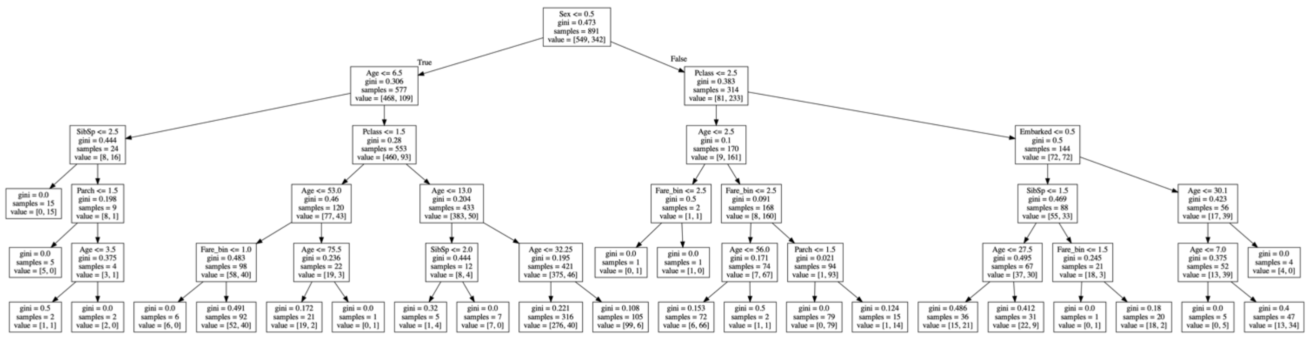
Output were as follows:

```
Decision tree accuracy score is:  0.8208955223880597

Generalized tree accuracy score is:  0.8462401795735129

The accuracy score of the random forest tree is:  0.8134328358208955
```

Once these trees are built, decision tree was graphed using "graphviz". The decision tree looks as follows:


Sex <= 0.5 gini = 0.473 samples = 891 value = [549, 342]
True
False
Age <= 6.5 gini = 0.306 samples = 577 value = [468, 109]
Pclass <= 2.5 gini = 0.383 samples = 314 value = [81, 233]
SibSp <= 2.5 gini = 0.444 samples = 24 value = [8, 16]
Pclass <= 1.5 gini = 0.28 samples = 553 value = [460, 93]
Age <= 2.5 gini = 0.1 samples = 170 value = [9, 161]
Embarked <= 0.5 gini = 0.5 samples = 144 value = [72, 72]
gini = 0.0 samples = 15 value = [0, 15]
Parch <= 1.5 gini = 0.198 samples = 9 value = [8, 1]
Age <= 53.0 gini = 0.46 samples = 120 value = [77, 43]
Age <= 13.0 gini = 0.204 samples = 433 value = [383, 50]
Fare_bin <= 2.5 gini = 0.5 samples = 2 value = [1, 1]
Fare_bin <= 2.5 gini = 0.091 samples = 168 value = [8, 160]
SibSp <= 1.5 gini = 0.469 samples = 88 value = [55, 33]
Age <= 30.1 gini = 0.423 samples = 56 value = [17, 39]
gini = 0.0 samples = 5 value = [5, 0]
Age <= 3.5 gini = 0.375 samples = 4 value = [3, 1]
Fare_bin <= 1.0 gini = 0.483 samples = 98 value = [58, 40]
Age <= 75.5 gini = 0.236 samples = 22 value = [19, 3]
SibSp <= 2.0 gini = 0.444 samples = 12 value = [8, 4]
Age <= 32.25 gini = 0.195 samples = 421 value = [375, 46]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.0 samples = 1 value = [1, 0]
Age <= 56.0 gini = 0.171 samples = 74 value = [7, 67]
Parch <= 1.5 gini = 0.021 samples = 94 value = [1, 93]
Age <= 27.5 gini = 0.495 samples = 67 value = [37, 30]
Fare_bin <= 1.5 gini = 0.245 samples = 21 value = [18, 3]
Age <= 7.0 gini = 0.375 samples = 52 value = [13, 39]
gini = 0.0 samples = 4 value = [4, 0]
gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 2 value = [2, 0]
gini = 0.0 samples = 6 value = [6, 0]
gini = 0.491 samples = 92 value = [52, 40]
gini = 0.172 samples = 21 value = [19, 2]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.32 samples = 5 value = [1, 4]
gini = 0.0 samples = 7 value = [7, 0]
gini = 0.221 samples = 316 value = [276, 40]
gini = 0.108 samples = 105 value = [99, 6]
gini = 0.153 samples = 72 value = [6, 66]
gini = 0.5 samples = 2 value = [1, 1]
gini = 0.0 samples = 79 value = [0, 79]
gini = 0.124 samples = 15 value = [1, 14]
gini = 0.486 samples = 36 value = [15, 21]
gini = 0.412 samples = 31 value = [22, 9]
gini = 0.0 samples = 1 value = [0, 1]
gini = 0.18 samples = 20 value = [18, 2]
gini = 0.0 samples = 5 value = [0, 5]
gini = 0.4 samples = 47 value = [13, 34]

(Full size graph provided with submission)

Once the trees are constructed, I moved into doing five-fold cross validation for both Decision Tree algorithm and Random Tree algorithm. In the process, I calculated mean and standard deviation.

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. It is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data.

Output looks as follows:

```
Five-fold cross validation on Decision Tree algorithm, Mean and
standard deviation is:  0.8025806451612905 0.041271306104801855

Five-fold cross validation on Random Forest algorithm, Mean and standard
deviation is:  0.8089548387096773 0.024225317636063257
```

The mean represents the model skills score and the standard deviation represents the variance of the skill scores.

Based on my analysis, Decision tree model is a better algorithm with higher accuracy score and higher mean value for the model skills score.

The main difference between decision tree and a random forest tree is that a decision tree is built on an entire dataset, using all the features/variables of interest. A random forest randomly selects observations/rows and specific features/variables to build multiple decision trees from and then averages the results. The random forest algorithm combines the output of multiple (randomly created) decision trees to generate the final output.

From my analysis, the accuracy score of the decision tree algorithm (82.08%) is found to be more than that of the random tree algorithm (81.34%). Even though the accuracy score of the decision tree algorithm is not significantly high, every bit of accuracy improvement matters while building a machine learning algorithm. The five-fold cross validation mean for decision tree algorithm and random forest algorithm are very close to each other (0.8025 vs. 0.8089). However, the standard deviation of decision tree algorithm is more than that of random forest algorithm (0.0412 vs. 0.0242).

Task 2:

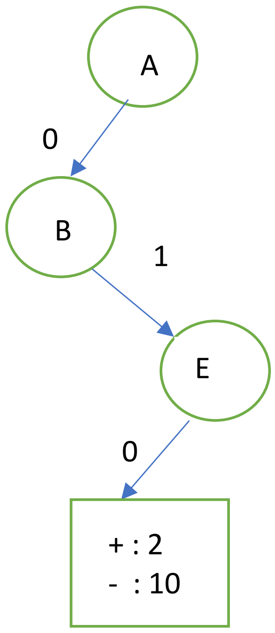
a) To calculate the training error rate, we use the below formula:

Training error rate = (lowest value at each leaf node) / (sum of all leaf node values)

The sum of all the counts at the leaf nodes is (14 + 5 + 6 + 7 + 2 + 10 + 8 + 6 + 5 + 17 + 15 + 5) = 100.
The sum of all the lowest counts at the leaf nodes is (5 + 6 + 2 + 6 + 5 + 5) = 29

Training error rate = (lowest value at each leaf node) / (sum of all leaf node values)
= 29 / 100
= 0.29

b) If the test instance T = {A=0, B=1, C=1, D=1, E=0}
Building a tree based on this:



The class that is assigned by the decision tree is obtained by tracing the values of the test instance T, starting from the root node, A to the leaf node, E.

Starting from root node, A
A = 0, takes to decision node B
B = 1, takes to decision node E
E = 0, takes to the leaf node under it.

Task 3:

Attributes		**Class Label**	
A	**B**	**+**	**-**
T	F	1	0
T	T	1	0
T	T	1	0
T	F	0	1
T	T	1	0
F	F	0	1
F	F	0	1
F	F	0	1
T	T	0	1
T	F	0	1

a) Entropy is the measure of disorder or uncertainty in data and the goal is to reduce it.

Entropy (t) = $-\sum p(j|t) \log [p(j|t)]$

Entropy before split = $-[(4/10)\log(4/10) + (6/10)\log(6/10)]$
$= 0.2922$

b)

	A = T	A = F
+	4	0
-	3	3

Entropy $|_{A=T} = -[(4/7)\log(4/7) + (3/7)\log(3/7)]$
$= 0.2965$

Entropy $|_{A=F} = -[(0/3)\log(0/3) + (3/3)\log(3/3)]$
$= 0$

Weighted Entropy after split = $(7/10) * 0.2965 + (3/10) * 0$
$= 0.2076$

Information Gain $|_A$ = Entropy before split – Weighted Entropy after split
$= 0.2922 - 0.2076$
$= 0.0846$

c)

	B = T	B = F
+	3	1
-	1	5

Entropy $|_{B=T} = -[(3/4)\log(3/4) + (1/4)\log(1/4)]$
$= 0.2442$

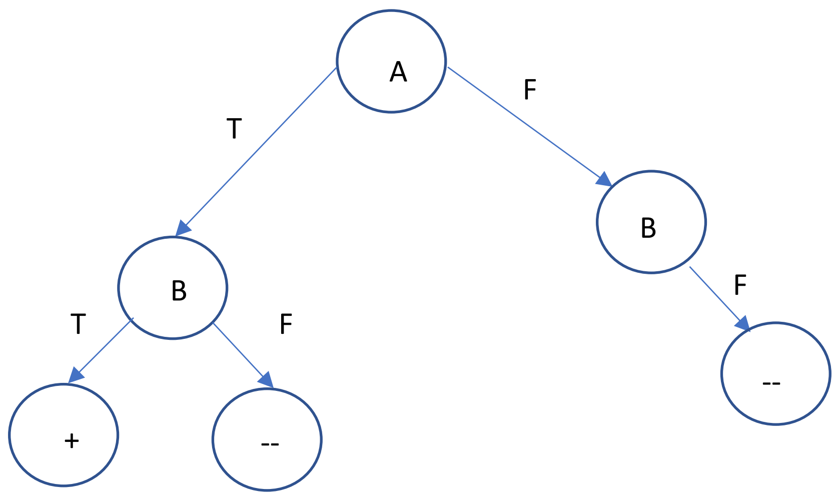
Entropy $|_{B=F} = -[(1/6)\log(1/6) + (5/6)\log(5/6)]$
$= 0.1956$

Weighted Entropy after split = $(4/10) * 0.2442 + (6/10) * 0.1956$
$= 0.2150$

Information Gain $|_A$ = Entropy before split – Weighted Entropy after split
$= 0.2922 - 0.2150$
$= 0.0771$

d) The decision tree would choose attribute A to be the root node as it has the highest information gain.

e)



Task 4:

a) Decision trees are non-linear classifiers like neural networks. It is generally used to classify non-linearly separable data, that is, it is used to do non-linear mapping of X to y.

b) Some of the weaknesses of a decision tree are:
   i) Small change in data can cause a large change in the structure of the decision tree causing instability.
   ii) Calculations for a decision tree might sometimes prove to be far more complex.
   iii) Decision tree can be computationally expensive to train and thus more time is taken, particularly if many values are uncertain.
   iv) They are often relatively inaccurate compared to other algorithms.
   v) Decision trees are inadequate for applying regression and predicting continuous values.

c) Splitting criterion should favor homogeneous or pure nodes. Gini index is more sensitive to changes compared to Misclassification error. Hence, Gini index is better as the splitting criterion for decision trees.