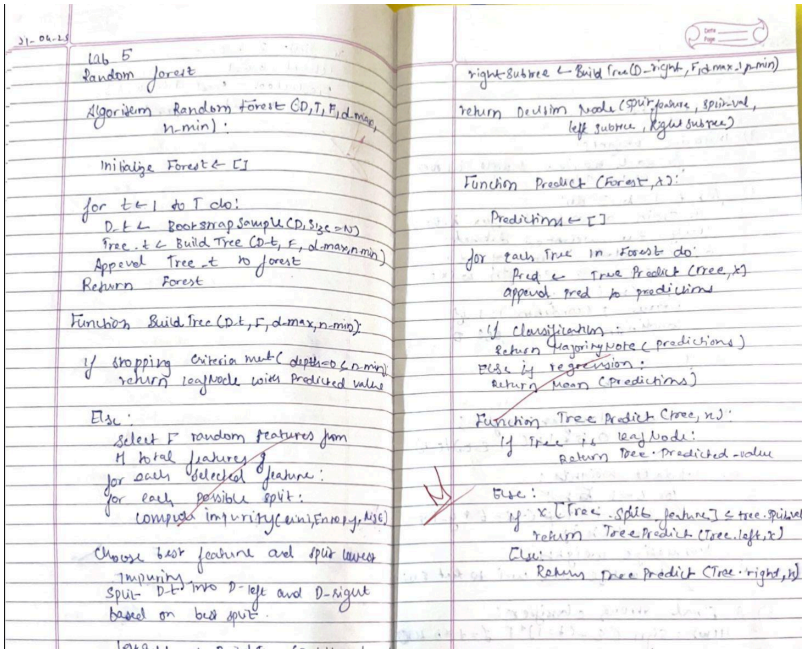
Implement Random Forest ensemble method on a given dataset



Lab 5
Random forest

Algorithm Random Forest (D, T, F, d-max, n-min):

Initialize Forest ← []

for t ← 1 to T do:
D_t ← BootstrapSample(D, size=N)
Tree_t ← Build Tree (D_t, F, d-max, n-min)
Append Tree_t to forest
Return Forest

Function Build Tree (D_t, F, d-max, n-min)

if stopping criteria met (depth=0 / n-min)
return leaf node with predicted value

Else:
select F random features from M total features
for each selected feature:
for each possible split:
compute impurity (gini, Entropy, MSE)

Choose best feature and split where impurity
split D_t into D-left and D-right based on best split

right-subtree ← Build Tree(D_right, F, d-max-1, n-min)

return Decision node (split-feature, split-val, left subtree, right subtree)

Function Predict (Forest, x):

Predictions ← []

for each tree in Forest do:
Pred ← Tree Predict (Tree, x)
append pred to predictions

If classification:
Return majority vote (predictions)
Else if regression:
Return mean (predictions)

Function Tree Predict (Tree, x):
If Tree is leaf Node:
Return Tree.Predicted-value

Else:
if x[Tree.Split_feature] ≤ tree.Split-val
return Tree Predict (Tree.left, x)
Else:
Return Tree Predict (Tree.right, x)

Code:

```
[23] iris['Species'] = iris['Species'].astype("category")
     codes = iris['Species'].cat.codes
```

```
[24] def train_test_split(X, y, random_state=42, test_size=0.2):
         """
         Splits the data into training and testing sets.

         Parameters:
             X (numpy.ndarray): Features array of shape (n_samples, n_features).
             y (numpy.ndarray): Target array of shape (n_samples,).
             random_state (int): Seed for the random number generator. Default is 42.
             test_size (float): Proportion of samples to include in the test set. Default is 0.2.

         Returns:
             Tuple[numpy.ndarray]: A tuple containing X_train, X_test, y_train, y_test.
         """
         # Get number of samples
         n_samples = X.shape[0]

         # Set the seed for the random number generator
         np.random.seed(random_state)

         # Shuffle the indices
         shuffled_indices = np.random.permutation(np.arange(n_samples))

         # Determine the size of the test set
         test_size = int(n_samples * test_size)

         # Split the indices into test and train
         test_indices = shuffled_indices[:test_size]
         train_indices = shuffled_indices[test_size:]

         # Split the features and target arrays into test and train
         X_train, X_test = X[train_indices], X[test_indices]
         y_train, y_test = y[train_indices], y[test_indices]

         return X_train, X_test, y_train, y_test
```

```
[25] X = iris.iloc[:, :-1].values
     y = iris.iloc[:, -1].values.reshape(-1,1)
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=41)
```

```
[26] from sklearn.tree import DecisionTreeClassifier
     m = DecisionTreeClassifier()
```

```
[27] class RandomForest:
         """
         A random forest classifier.

         Parameters
         ----------
         n_trees : int, default=7
             The number of trees in the random forest.
         max_depth : int, default=7
             The maximum depth of each decision tree in the random forest.
         min_samples : int, default=2
             The minimum number of samples required to split an internal node
             of each decision tree in the random forest.

         Attributes
         ----------
         n_trees : int
             The number of trees in the random forest.
         max_depth : int
             The maximum depth of each decision tree in the random forest.
         min_samples : int
             The minimum number of samples required to split an internal node
             of each decision tree in the random forest.
         trees : list of DecisionTreeClassifier
             The decision trees in the random forest.
         """

         def __init__(self, n_trees=7, max_depth=7, min_samples=2):
             """
             Initialize the random forest classifier.
```

Output:

```
[17] from google.colab import drive
     drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[18] import math
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     import plotly.express as px
```

```
[19] iris = pd.read_csv("/content/drive/MyDrive/Iris.csv") #Load Data
     iris.drop('Id',inplace=True,axis=1) #Drop Id column
```

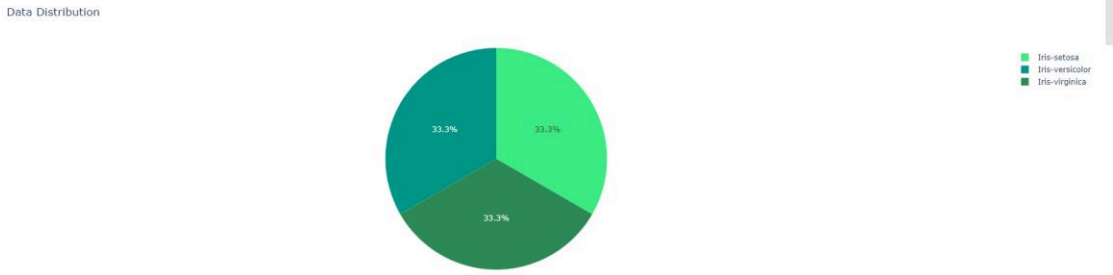
```
[20] iris.head().style.background_gradient(cmap =sns.light_palette("seagreen", as_cmap=True)
     )
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.100000	3.500000	1.400000	0.200000	Iris-setosa
1	4.900000	3.000000	1.400000	0.200000	Iris-setosa
2	4.700000	3.200000	1.300000	0.200000	Iris-setosa
3	4.600000	3.100000	1.500000	0.200000	Iris-setosa
4	5.000000	3.600000	1.400000	0.200000	Iris-setosa

```
[21] X_df = iris.iloc[:,:-1] #Set our training dataframe

     y_df = iris.iloc[:,-1] # Set our training labels dataframe
```

```
[22] fig = px.pie(iris, 'Species',color_discrete_sequence=['#3dec84 ','#009688 ','#2EBB57 '],title='Data Distribution',template='plotly')
     fig.show()
```




```
[23] iris['Species'] = iris['Species'].astype("category")
     codes = iris['Species'].cat.codes
```