ACTIVITY #7

CLASSIFICATION MODEL

KNN

Decision Tree

Random forest

AGENDA

7.1	Data Preparation
7.2	Model Training and Testing
7.3	Hyperparameter tuning

7.1

DATA PREPARATION

Data Exploration

Transform

Feature Selection

Train-Test-Split

LIBRARIES

1. import numpy as np
2. import pandas as pd
3. import matplotlib.pyplot as plt
4. import plotly.express as px
5. from sklearn import preprocessing
6. from sklearn.model_selection import train_test_split, GridSearchCV
7. from sklearn.neighbors import KNeighborsClassifier
8. from sklearn.tree import DecisionTreeClassifier
9. from sklearn.ensemble import RandomForestClassifier
10. From sklearn.metrics import accuracy_score, classification_report, confusion_matrix

7.1 (a) Get Data

1

- # Load data from csv file
  - read_csv('Coffee-modified.csv')
  - Select only
    - ['Total.Cup.Points', 'Species','Country.of.Origin','Processing.Method', 'Aroma', 'Flavor', 'Aftertaste', 'Acidity','Body', 'Balance', 'Uniformity', 'Moisture', 'altitude_mean_meters']

2

- # Data Preparation
  - # Drop NA
    - Dropna()
  - # View Statistics
    - Describe()

3

- # Assign X, Y (drop datetime index)
  - Y = 1st column (['Total.Cup.Points'])
  - X = 2nd :last column

7.1 (b) Data Preparation (prepare Y)

1

- # Process Y from values to Coffee Bean Grade
  - # define Bean_Grade = [1,2,3] using
    - rating_pctile = np.percentile(Y, [75, 90])

      Bean_grade = 1 if Y < rating_pctile [0] #75 percentile

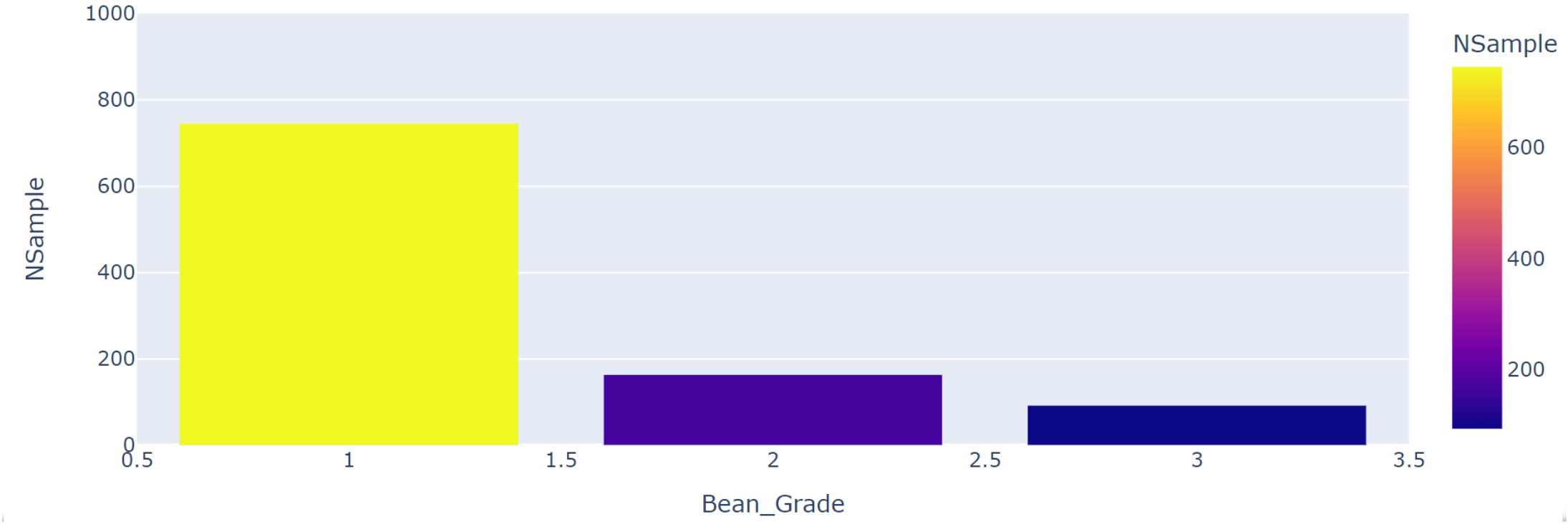
      Bean_grade = 2 if rating_pctile [0] <= Y < rating_pctile [1] # 90 percentile

      Bean_grade = 3 if Y >= rating_pctile[1]

2

- # Visualize Bar Graph of Number of Samples for each Bean Grade
- # ตัวอย่างการลองใช้ plotly express library

```
fig = px.bar( df_Y, x = 'Bean grade', y = 'NSamples',  color='NSamples', range_y=[0.0,1000])
fig.show()
```



Visualize Bar Graph of Number of Samples for each Bean Grade

7.1 (c) Data preparation (Prepare X)

1
- # Standardized data (X [numerical feature columns])
  - standard_scaler.fit_transform(X[numerical_columns])

2
- # feature selection (correlation)
  - Calculate correlation between variables for only continuous data columns
    - corr()
    - Reduce Corr() to Lower Matrix
    - Drop columns if |correlation value| > 0.8

3
- # One hot encoding for Categorical feature columns
  - pd.get_dummies(X, columns = categorical_features, drop_first=True)

4
- # Prepare X train, Xtest , Y train, Ytest
  - train_test_split()

7.2

MODEL PREPARATION

KNN
DECISION TREE
RANDOM FOREST

7.2 (a) KNN Model Training and Testing

1

- **# KNN parameter**
  - k = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 25, 35] # try at least 3 values

2

- # Model Training
  - modelKNN = KNeighborsClassifier(n_neighbors=k, p=2)
  - modelKNN.fit(x_train,y_train)

3

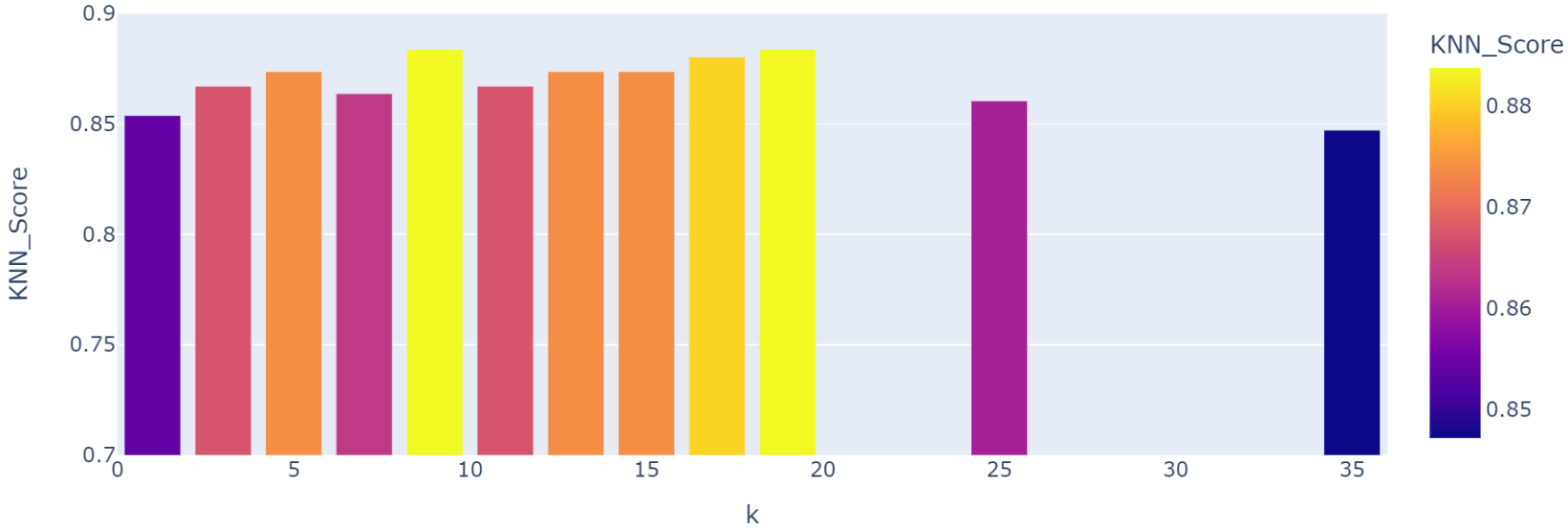
- # Model Testing
  - y_pred= modelKNN.predict(x_test)
  - KNNScore = accuracy_score(y_test, y_pred)

4

- # Visualize compare accuracy of selected k values (at least 3 values of k)
  - fig = px.bar(KNNScore, x = 'k', y = 'KNN_Score', color='KNN_Score', range_y=[0.7,1.0])
  - fig.show()

5

- # Print Confusion Matrix and Classification Report for best k
  - print('Confusion Matrix: ')
  - print(confusion_matrix(y_test, y_pred))
  - print('Classification Report: ')
  - print(classification_report(y_test, y_pred))




```
Confusion Matrix:
[[213   5   2]
 [ 16  33   2]
 [  0  10  20]]
---------------------------
Classification Report:
              precision    recall  f1-score   support

           1       0.93      0.97      0.95       220
           2       0.69      0.65      0.67        51
           3       0.83      0.67      0.74        30

    accuracy                           0.88       301
   macro avg       0.82      0.76      0.79       301
weighted avg       0.88      0.88      0.88       301
```

Visualize compare accuracy of selected k values, Confusion Matrix, and Classification Report

7.2 (b) Decision Tree Model Training and Testing

1

- **# Decision Tree parameter**
  - ASM_function = [‘entropy’, ‘gini’] / maxD = [4, 5, 6, None] # try at least 2 values

2

- # Model Training
  - ModelDT = DecisionTreeClassifier(criterion=ASM_function, splitter='best',max_depth =maxD)
  - ModelDT.fit(x_train,y_train)

3

- # Model Testing
  - y_pred= modelDT.predict(x_test)
  - DTScore = accuracy_score(y_test, y_pred)
  - Print(DTScrore)

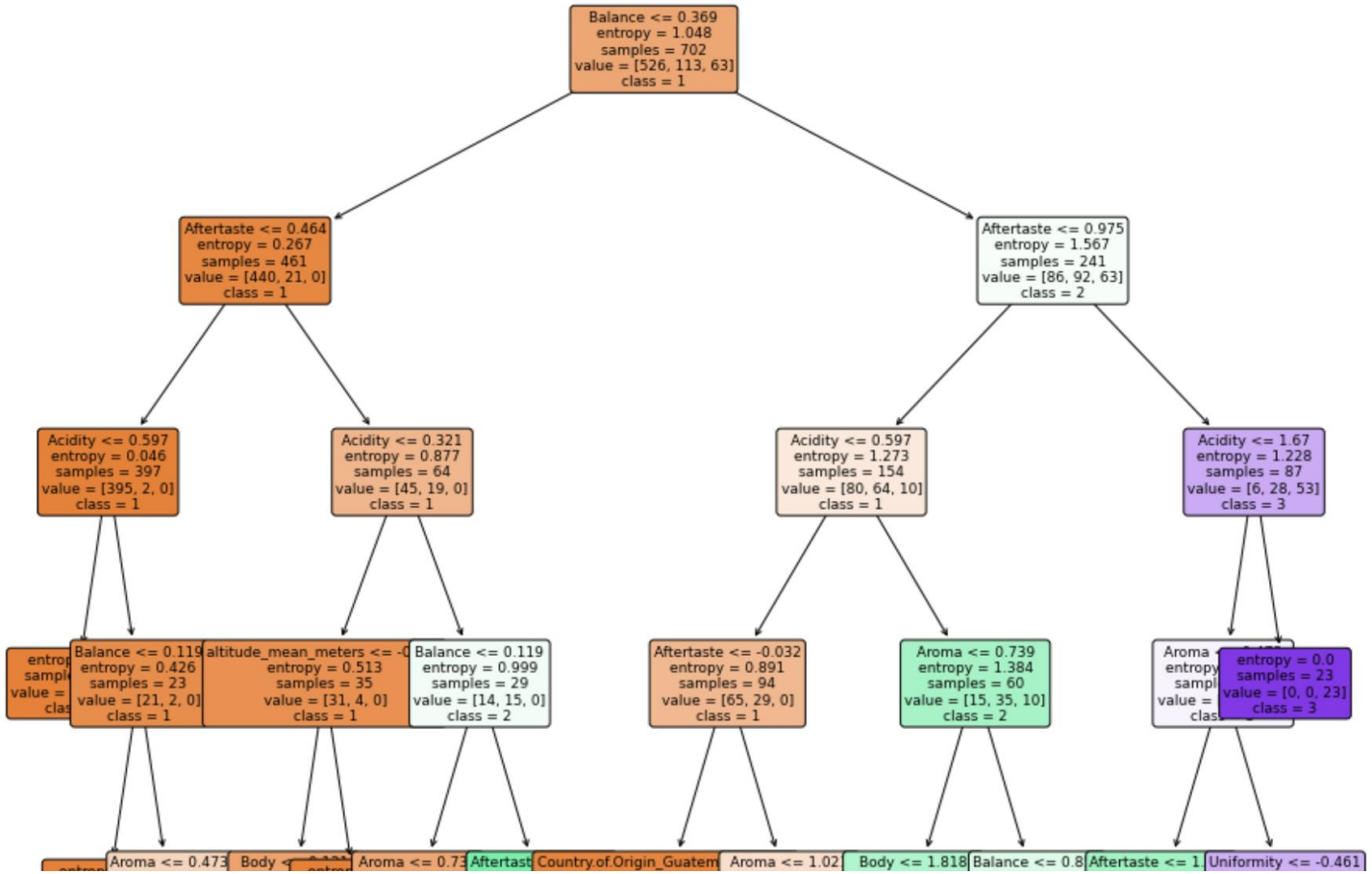
4

- # Print Confusion Matrix and Classification Report for best k
  - print('Confusion Matrix: ')
  - print(confusion_matrix(y_test, y_pred))
  - print('Classification Report: ')
  - print(classification_report(y_test, y_pred))

7.2 (b) Decision Tree Model Training and Testing

5

```
# Visualize Decision Tree
feature_names = x_train.columns
Labels = np.unique(y_train)

tree.plot_tree( modelDT,
                feature_names = feature_names,
                class_names = labels,
                rounded = True,
                filled = True, fontsize=9)
plt.show()
```




```
Max Depth:  5
----------------------------
Confusion Matrix:
[[206  12   2]
 [ 12  28  11]
 [  1   5  24]]
----------------------------
Classification Report:
              precision    recall  f1-score   support

           1       0.94      0.94      0.94       220
           2       0.62      0.55      0.58        51
           3       0.65      0.80      0.72        30

    accuracy                           0.86       301
   macro avg       0.74      0.76      0.75       301
weighted avg       0.86      0.86      0.86       301
```

Visualize Decision Tree Structure, Confusion Matrix, and Classification Report

7.2 (c) Random Forest Model Training and Testing

1

- **# Random Forest parameter**
  - ASM_function = ['entropy', 'gini'] / nEstimator = 100 / nJob = 2, rState = 10

2

- # Model Training
  - RandomF = RandomForestClassifier(criterion=ASM_function,n_estimators=nEstimator, n_jobs=nJob, random_state=rState)
  - RandomF.fit(x_train,y_train)

3

- # Model Testing
  - y_pred= RandF.predict(x_test)
  - RFScore = accuracy_score(y_test, y_pred)
  - Print(RFScore)

4

- # Print Confusion Matrix and Classification Report for best k
  - print('Confusion Matrix: ')
  - print(confusion_matrix(y_test, y_pred))
  - print('Classification Report: ')
  - print(classification_report(y_test, y_pred))

7.2 (b) Random Forest Model Training and Testing

5

- # Visualize Feature Important Score

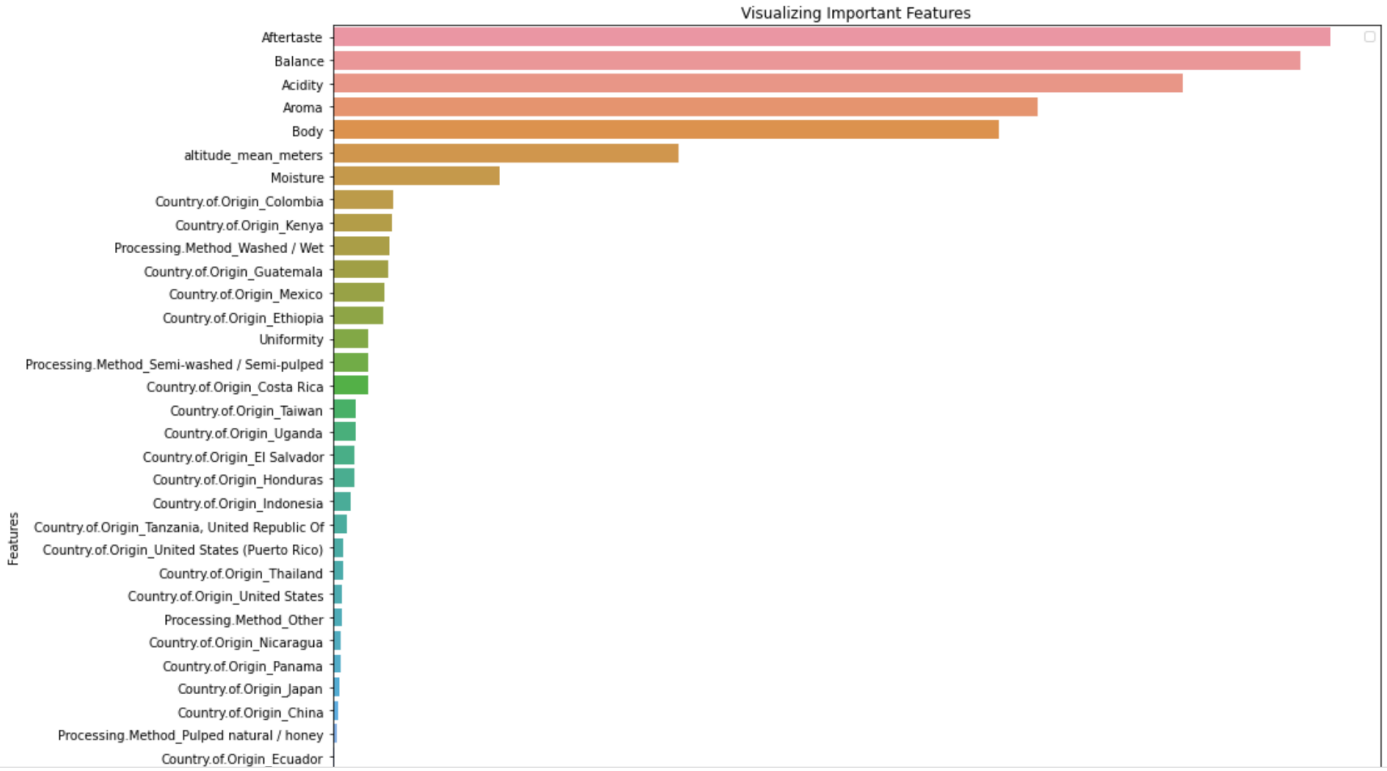
```
feature_imp = pd.Series(RandomF.feature_importances_, index = feature_names).sort_values(ascending=False)

# Creating a bar plot
plt.figure(figsize=(15,15))
sns.barplot(x=feature_imp, y=feature_imp.index)
```

6

- # Visualize selected estimator [0-5] tree structure of Random forest

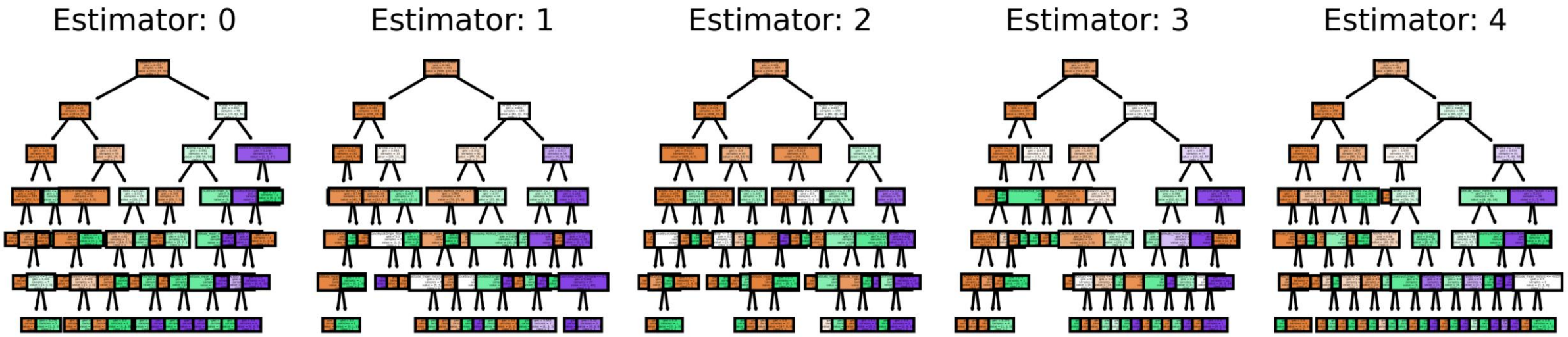
```
fig, axes = plt.subplots(nrows = 1,ncols = 5,figsize = (10,2), dpi=500)
for index in range(0, 5):
   tree.plot_tree( RandomF.estimators_[index],
                   feature_names = feature_names,
                   class_names= labels,
                   filled = True,
                   ax = axes[index]);
```

- axes[index].set_title('Estimator: ' + str(index), fontsize = 11)




```
criterion=entropy,n_estimators=50
---------------------------
Confusion Matrix:
[[213   5   2]
 [ 12  37   2]
 [  0  10  20]]
---------------------------
Classification Report:
              precision    recall  f1-score   support

           1       0.95      0.97      0.96       220
           2       0.71      0.73      0.72        51
           3       0.83      0.67      0.74        30

    accuracy                           0.90       301
   macro avg       0.83      0.79      0.81       301
weighted avg       0.90      0.90      0.90       301
```

Visualize Random Forest Feature Important, Confusion Matrix, and Classification Report



Visualize Random Forest Feature Important, Confusion Matrix, and Classification Report

7.3

HYPERPARAMETER TUNING

(GRIDSEARCHCV())

KNN
DECISION TREE
RANDOM FOREST

7.3 Hyperparameter Tuning (GridsearchCV)

1

- **# Create Model List**
  - **classification = { 'KNN': KNeighborsClassifier(), 'DT': DecisionTreeClassifier(), 'RF': RandomForestClassifier() }**

2

- # Create Parameter Dictionary for KNN
  - K_list = [1, 3, 5, 7, 9 , 11, 13, 15, 17, 19, 21, 23, 25, 35, 45]
  - KNN_param = dict(n_neighbors=K_list)

3

- # Create Parameter Dictionary for Decision Tree
  - ASM_function = ['entropy', 'gini']
  - maxD = [4, 5, 6, None]
  - *maxF = ["auto", "log2", None]*
  - minSample = [1,2, 4]
  - DT_param= dict(criterion=ASM_function, max_depth = maxD, min_samples_leaf = minSample, max_features = maxF)

4

- # Create Parameter Dictionary for Random Forest (including same parameters as Decision Tree)
  - nEst = [10, 30, 50, 100]
  - RF_param = dict(n_estimators = nEst, criterion=ASM_function, max_depth = maxD, min_samples_leaf = minSample, max_features = maxF)

7.3 Hyperparameter Tuning (GridsearchCV)

5

- # Perform GridsearchCV() for each classification model

```
grid = GridSearchCV(  model,
                      n_jobs,
                      verbose,
                      scoring = 'accuracy'
                      cv = 2
                      param_grid     )
```

- grid_result = grid.fit(x_train, y_train)

6

- # Show best search results
  - Print()
- # Show and Display Mean, std, params
  - Print()
  - Bar()