Name: Trung Kien Nguyen

Student ID: 104053642

Studio: 1 – 3

PORTFOLIO – WEEK 4

The implementation file for this pportfolio is a Juppiter Notebook, which is retrieved from https://colab.research.google.com/drive/1kf2qg8c82fJPgKYez7QwCBgAyFY6of92#scrollTo=xuUkQtZLgaMN

Step 1: Data Preparation

Shuffle and Split Data

```python
# STEP 1: DATA PREPARATION

import pandas as pd
import io

#   1.1. SHUFFLE AND SPLIT DATA

# Load the dataset
file_path = 'vegemite.csv'
data = pd.read_csv(io.BytesIO(uploaded[file_path]))

# Shuffle the dataset
data_shuffled = data.sample(frac=1, random_state=1)

# Define the number of samples per class to extract
samples_per_class = 300  # This number will ensure we get close to 1000 samples total

# Extract samples for each class, assuming the class label is in a column named 'Class'
class_0_samples = data_shuffled[data_shuffled['Class'] == 0].head(samples_per_class)
class_1_samples = data_shuffled[data_shuffled['Class'] == 1].head(samples_per_class)
class_2_samples = data_shuffled[data_shuffled['Class'] == 2].head(samples_per_class)

# Combine the samples to form the test set
test_data = pd.concat([class_0_samples, class_1_samples, class_2_samples])

# Use the remaining data as the training set
train_data = data_shuffled.drop(test_data.index)

# Display the first few rows of the test and training data
print("------------------------------------------------------------------------------------------------------------------------------------")
print("Test Data Preview:")
print(test_data.head())

print("\n------------------------------------------------------------------------------------------------------------------------------------")
print("Training Data Preview:")
print(train_data.head())
```

Initially, I have shuffled the whole uploaded dataset (in the given file “vegemite.csv”), using the “sample” method of pandas.DataFrame.

I have then made sure that each class (0, 1, and 2) contributes 300 samples, which meet the minimum requirements, leading to a total of 900 samples for the test set and the rest 14338 samples is for train set.

Here is the preview of splitted train and test datasets (5 rows each):

```
Test Data Preview:
      FFTE Feed tank level SP  FFTE Production solids SP  \
7539                     50.0                      43.00
5705                     50.0                      40.50
7599                     50.0                      42.00
7157                     50.0                      41.68
6261                     50.0                      41.50

      FFTE Steam pressure SP  TFE Out flow SP  TFE Production solids SP  \
7539                   135.0          2609.30                      69.0
5705                    99.9          2525.93                      60.0
7599                   115.0          2214.29                      69.0
7157                   135.0          1904.29                      70.0
6261                   105.0          2081.93                      59.0

      TFE Vacuum pressure SP  TFE Steam pressure SP  TFE Steam temperature SP  \
7539                  -56.85                 120.00                      80.0
5705                  -80.00                 125.00                      80.0
7599                  -80.00                 120.00                      80.0
7157                  -79.75                  54.67                      80.0
6261                  -79.79                 120.00                      80.0

      FFTE Feed flow SP  FFTE Out steam temp SP  ...  TFE Out flow PV  \
7539            10300.0                   50.00  ...          1629.05
5705             9400.0                   50.00  ...           962.39
7599             9500.0                   50.12  ...          2513.62
7157             9500.0                   51.11  ...          1667.32
6261             9600.0                   50.00  ...          3205.58

      TFE Product out temperature  TFE Production solids PV  \
7539                          0.0                     66.73
5705                          0.0                     47.08
7599                          0.0                     59.99
7157                          0.0                     75.50
6261                          0.0                     65.12

      TFE Production solids density  TFE Steam pressure PV  \
7539                           1.20                 119.91
5705                           1.20                 120.23
7599                           1.24                 120.23
7157                           0.91                 125.03
6261                           0.96                 119.91

      TFE Steam temperature  TFE Tank level  TFE Temperature  \
7539                  61.07           32.88             70.0
5705                  69.16           12.72             70.0
7599                  72.23           37.95             72.0
7157                  66.84           82.51             75.0
6261                  76.78           83.07             77.0

      TFE Vacuum pressure PV  Class
7539                  -78.10      0
5705                  -78.10      0
7599                  -68.93      0
7157                  -71.04      0
6261                  -66.12      0

[5 rows x 47 columns]
```

Test data preview

```
Training Data Preview:
       FFTE Feed tank level SP  FFTE Production solids SP  \
11341                     50.0                       42.0
13723                     50.0                       43.0
9129                      50.0                       40.5
11612                     50.0                       43.0
9321                      50.0                       40.5

       FFTE Steam pressure SP  TFE Out flow SP  TFE Production solids SP  \
11341                   140.0          2679.49                      63.0
13723                   135.0          2846.51                      75.0
9129                    126.5          2296.30                      65.0
11612                   115.0          2988.84                      71.0
9321                    141.1          2296.30                      63.0

       TFE Vacuum pressure SP  TFE Steam pressure SP  \
11341                  -71.30                  125.0
13723                  -59.80                  120.0
9129                   -72.98                  120.0
11612                  -59.16                  120.0
9321                   -73.80                  125.0

       TFE Steam temperature SP  FFTE Feed flow SP  FFTE Out steam temp SP  \
11341                      80.0             9800.0                   50.00
13723                      80.0            10400.0                   51.11
9129                       80.0             9500.0                   50.00
11612                      80.0            10200.0                   50.00
9321                       80.0             9600.0                   50.00

       ...  TFE Out flow PV  TFE Product out temperature  \
11341  ...          5364.46                          0.0
13723  ...          6717.25                          0.0
9129   ...           866.41                          0.0
11612  ...          1289.23                          0.0
9321   ...          1725.03                          0.0

       TFE Production solids PV  TFE Production solids density  \
11341                     75.48                           1.24
13723                      4.46                           1.24
9129                      45.78                           1.17
11612                     62.02                           0.90
9321                      69.02                           0.67

       TFE Steam pressure PV  TFE Steam temperature  TFE Tank level  \
11341                 119.91                  67.55           82.20
13723                   1.98                  60.72           16.16
9129                  120.23                  68.96           82.99
11612                 120.23                  72.74           18.85
9321                  120.23                  69.02           82.59

       TFE Temperature  TFE Vacuum pressure PV  Class
11341             76.0                  -70.68      2
13723             75.0                    2.30      2
9129              74.0                  -66.83      2
11612             75.0                  -67.54      2
9321              77.0                  -67.54      2

[5 rows x 47 columns]
```

Train data preview

Remove constant columns

```
[ ] #   1.2. REMOVE ANY CONSTANT COLUMNS

    # Check for constant value columns
    constant_columns = [col for col in train_data.columns if train_data[col].nunique() == 1]

    # Print constant columns and remove if any
    if constant_columns:
        print("Constant columns to be removed:", constant_columns)

        # Remove constant value columns from the training dataset
        train_data_with_no_constant_cols = train_data.drop(columns=constant_columns)

        # Confirm removal
        print("Constant columns have been removed.")
    else:
        print("No constant columns found in the training dataset.")

    print("\n-----------------------------------------------------------------------------------------------------------------------")
    print("Training Data Preview after possibly removal of Constants:")
    print(train_data_with_no_constant_cols.head())
```

For this task, firstly I checked if there is any constants columns in the splitted train dataset. The method "nunique", which returns the number of unique values in each column, is used. If that value is equal to 1, then the checking column is the constant one.

I then removed all constant column by simply using "drop" method of pandas.DataFrame.

With the given dataset that have been shuffled and splitted to train dataset, here is the output:

```
Constant columns to be removed: ['TFE Steam temperature SP', 'TFE Product out temperature']
Constant columns have been removed.

-----------------------------------------------------------------------------------------------------------------------
Training Data Preview after possibly removal of Constants:
       FFTE Feed tank level SP  FFTE Production solids SP  \
11341                     50.0                       42.0
13723                     50.0                       43.0
9129                      50.0                       40.5
11612                     50.0                       43.0
9321                      50.0                       40.5

       FFTE Steam pressure SP  TFE Out flow SP  TFE Production solids SP  \
11341                   140.0          2679.49                      63.0
13723                   135.0          2846.51                      75.0
9129                    126.5          2296.30                      65.0
11612                   115.0          2988.84                      71.0
9321                    141.1          2296.30                      63.0

       TFE Vacuum pressure SP  TFE Steam pressure SP  FFTE Feed flow SP  \
11341                  -71.30                  125.0             9800.0
13723                  -59.80                  120.0            10400.0
9129                   -72.98                  120.0             9500.0
11612                  -59.16                  120.0            10200.0
9321                   -73.80                  125.0             9600.0

       FFTE Out steam temp SP  Extract tank Level  ...  TFE Motor speed  \
11341                   50.00                0.93  ...             80.0
13723                   51.11               14.38  ...             20.0
9129                    50.00               56.39  ...             80.0
11612                   50.00               54.73  ...             80.0
9321                    50.00               60.87  ...             80.0

       TFE Out flow PV  TFE Production solids PV  \
11341          5364.46                     75.48
13723          6717.25                      4.46
9129            866.41                     45.78
11612          1289.23                     62.02
9321           1725.03                     69.02

       TFE Production solids density  TFE Steam pressure PV  \
11341                           1.24                 119.91
13723                           1.24                   1.98
9129                            1.17                 120.23
11612                           0.90                 120.23
9321                            0.67                 120.23

       TFE Steam temperature  TFE Tank level  TFE Temperature  \
11341                  67.55           82.20             76.0
13723                  60.72           16.16             75.0
9129                   68.96           82.99             74.0
11612                  72.74           18.85             75.0
9321                   69.02           82.59             77.0

       TFE Vacuum pressure PV  Class
11341                  -70.68      2
13723                    2.30      2
9129                   -66.83      2
11612                  -67.54      2
9321                   -67.54      2

[5 rows x 45 columns]
```

The two constrant features "TFE Steam temperature SP" and "TFE Product out temperature" have been removed

Convert columns with few integer values to categorical features

```python
#   1.3. CONVERT ANY COLUMNS WITH FEW INTEGER VALUES TO CORRESPONDING CATEGORICAL FEATURES

# Define the threshold for what is considered 'few' unique values
threshold = 10  # Any integer column with 10 or fewer unique values

# Make a copy of the DataFrame to retain the original data
train_data_with_categorical_feature = train_data_with_no_constant_cols.copy()

# Function to determine if all non-NaN values in a column are integers
def is_integer_column(series):
    if pd.isnull(series).all():
        return False
    return (series.dropna() % 1 == 0).all()

i = 0  # Counter for converted columns
conversion_made = False

# Automatically convert all eligible integer columns to categorical in the new DataFrame, excluding 'Class'
for col in train_data_with_categorical_feature.columns:
    if col != 'Class':  # Exclude the target column from being converted
        if is_integer_column(train_data_with_categorical_feature[col]) and train_data_with_categorical_feature[col].nunique() <= threshold:
            train_data_with_categorical_feature[col] = train_data_with_categorical_feature[col].astype('category')
            print(f"Converting {col} to categorical. Unique values count: {train_data_with_categorical_feature[col].nunique()}")
            conversion_made = True
            i += 1

print("\n---------------------------------------------------------------------------------------------------------------------")
print(f"{i} columns have been converted to categorical.")

# Verify changes by displaying the data types of the modified dataset
print("\n---------------------------------------------------------------------------------------------------------------------")
print("Data types after conversion:")
print(train_data_with_categorical_feature.dtypes)

# Display the entire DataFrame (or a sample) to verify the changes
print("\n---------------------------------------------------------------------------------------------------------------------")
print("Training data after conversion (with possibly categorical feature):")
print(train_data_with_categorical_feature.head())
```

To do this, I have checked in each column, counted the number of unique integers. I determine that if the number of unique integer in a feature column is less than 10 ("threshold"), then it must be converted to categorical features. Also, previously when I uploaded the dataset, the features' values

To convert, I simply use the built-in method "astype", with parameter "category". Here is the outpout after the checking and converting process:

```
Converting FFTE Feed tank level SP to categorical. Unique values count: 3
Converting FFTE Pump 1 to categorical. Unique values count: 5
Converting FFTE Pump 1 - 2 to categorical. Unique values count: 4
Converting FFTE Pump 2 to categorical. Unique values count: 5
Converting TFE Motor speed to categorical. Unique values count: 3

---------------------------------------------------------------------------------------------------------------------
5 columns have been converted to categorical.

---------------------------------------------------------------------------------------------------------------------
Data types after conversion:
FFTE Feed tank level SP          category
FFTE Production solids SP         float64
FFTE Steam pressure SP            float64
TFE Out flow SP                   float64
TFE Production solids SP          float64
TFE Vacuum pressure SP            float64
TFE Steam pressure SP             float64
FFTE Feed flow SP                 float64
FFTE Out steam temp SP            float64
Extract tank Level                float64
Extract tank Out flow PV          float64
FFTE Discharge density            float64
FFTE Discharge solids             float64
FFTE Feed flow rate PV            float64
FFTE Feed tank level PV           float64
FFTE Heat temperature 1           float64
FFTE Heat temperature 2           float64
FFTE Heat temperature 3           float64
FFTE Out steam temp PV            float64
FFTE Production solids PV         float64
FFTE Pump 1                      category
FFTE Pump 1 - 2                  category
FFTE Pump 2                      category
FFTE Steam pressure PV            float64
FFTE Temperature 1 - 1            float64
FFTE Temperature 1 - 2            float64
FFTE Temperature 2 - 1            float64
FFTE Temperature 2 - 2            float64
FFTE Temperature 3 - 1            float64
FFTE Temperature 3 - 2            float64
FFTE Unk Temperature              float64
TFE Feed pump                     float64
TFE Input flow PV                 float64
TFE Level                         float64
TFE Motor current                 float64
TFE Motor speed                  category
TFE Out flow PV                   float64
TFE Production solids PV          float64
TFE Production solids density     float64
TFE Steam pressure PV             float64
TFE Steam temperature             float64
TFE Tank level                    float64
TFE Temperature                   float64
TFE Vacuum pressure PV            float64
Class                               int64
dtype: object
```

The five converted features include "FFTE Feed tank level SP", "FFTE Pump 1", "FFTE Pump 1 – 2", FFTE Pump 2", and "TFE Motor speed"

Check class balance

```python
#   1.4. CHECK CLASS BALANCE AND ADDRESS POTENTIAL IMBALANCE

# Check the Class distribution
class_distribution = train_data_with_categorical_feature['Class'].value_counts(normalize=True)
print("Class distribution:\n", class_distribution)

# Check for imbalance by seeing if any class's proportion is below a certain threshold, e.g., less than 20%
is_imbalanced = class_distribution.min() < 0.20

# Addressing potential imbalance
from sklearn.utils import resample
from imblearn.over_sampling import SMOTE

if is_imbalanced:
    train_data_balanced = None

    # Uncomment the following block to use undersampling
    min_class_size = int(class_distribution.min() * len(train_data_with_categorical_feature))
    train_data_balanced = pd.DataFrame()
    for class_value in class_distribution.index:
        class_subset = train_data_with_categorical_feature[train_data_with_categorical_feature['Class'] == class_value]
        resampled_subset = resample(class_subset,
                                    replace=False,
                                    n_samples=min_class_size,
                                    random_state=1)
        train_data_balanced = pd.concat([train_data_balanced, resampled_subset], axis=0)

    # Uncomment the following block to use SMOTE for oversampling
    # smote = SMOTE(random_state=1)
    # features, target = smote.fit_resample(train_data_with_categorical_feature.drop('Class', axis=1),
    #                                       train_data_with_categorical_feature['Class'])
    # train_data_balanced = pd.concat([features, pd.DataFrame(target, columns=['Class'])], axis=1)

    # Print the new class distribution after resampling
    new_class_distribution = train_data_balanced['Class'].value_counts(normalize=True)
    print("\n-------------------------------------------------------------------------------------------------------------------")
    print("New class distribution after resampling:\n", new_class_distribution)

else:
    print("\n-------------------------------------------------------------------------------------------------------------------")
    print("Data is already balanced. No resampling applied.")
    train_data_balanced = train_data_with_categorical_feature
```

In the implementation for checking class balance, I have counted the distribution of each class (number of each class's value's divide to the number of total samples. Then, if the min in the classes' distribution is less than a certain value, given that 20%, I stated there is an imbalance in the training dataset

For example, given the 10-element class array: [1, 1, 1, 1, **2, 2, 2**, 3, 3, 3, 3], the min distribution will be 3/10 = 30%, corresponding to the class value of 2. As 30% is an acceptable value, which is greater than 20%, there is no imbalance.

Next, I have suggested two techniques to solve the possible imblance, if found ("is_imbalanced == True"):

- Undersampling: This reduces the size of majority classes to match the one with min distribution. Specially, some random samples of those majority classes are discarded directly from the dataset, using sklearn.utils.resample method
- SMOTE (Synthetic Minority Over-sampling Technique): Undersampling directly discard samples of majority classes, which might result in information loss. SMOTE provide a more effiecient approach, generating synthetic examples instead of removing existing ones. I have used the the available method "fit_resample" of class imblearn.over_sampling.SMOTE to generate new features and targets, then use pandas.concat method to add them to the train dataset

In the train dataset after converting some columns to categorical features, there is a noticable imbalance, which has been resolved:

```
Class distribution:
 Class
2    0.505545
1    0.331101
0    0.163354
Name: proportion, dtype: float64
New class distribution after resampling:
 Class
2    0.333333
1    0.333333
0    0.333333
Name: proportion, dtype: float64
```

Explore the Dataset and Create Composite Features

Analysing the relationships between features to find interactions that might improve model performance when stated as a single feature is necessary before deciding which pairs of features could benefit from being combined into composite features. Depending on how closely related and meaningful a pair of characteristics is to the goal variable, composite features are frequently produced by multiplying, adding, dividing, or removing those pairs.

Also note that, the category features have been converted before is not coverred by the above operands, I have to temporarily drop them, and re-add after the composite features creating has done:

```
#   1.5. EXPLORE AND CREATE COMPOSITE FEATURES

# Identify categorical columns, including those converted earlier, as the basic operands for compositing process can not be applied to them.
categorical_cols = train_data_balanced.select_dtypes(include=['category']).columns.tolist()

# The target column 'Class'
categorical_cols.append('Class')  # Adding the target feature to the list to exclude it as well

# Create a new DataFrame excluding the categorical columns and the target feature
train_data_filtered = train_data_balanced.drop(columns=categorical_cols)
```

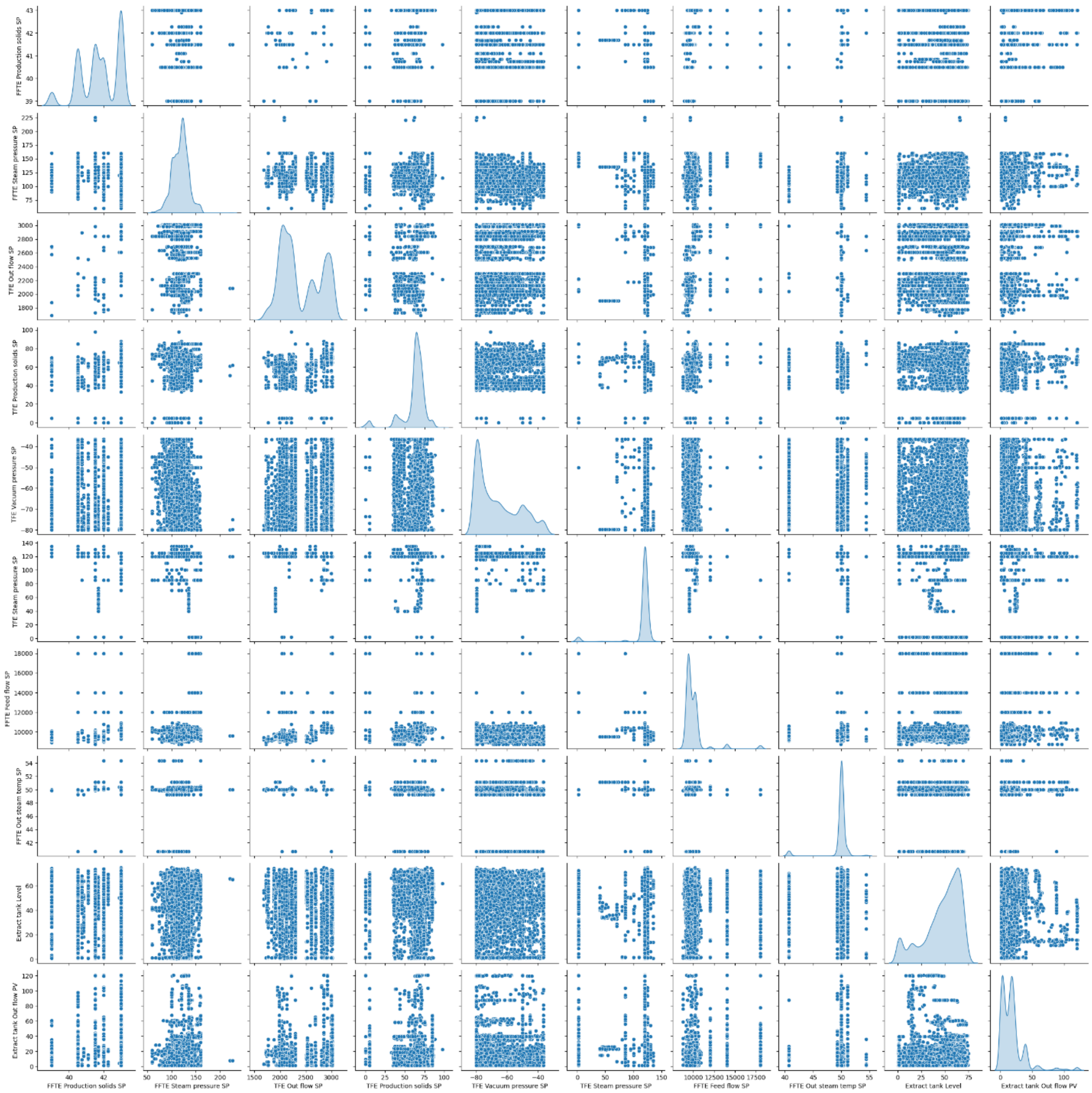
There are two common ways of doing this analysis, using either Pair Plots or Heatmap. I have implemented the code to plot the two types of illustrations.

- Pair plots: The theory of this analysis include
    - If we see any linear trends between two features, it is recommend to think about combining two features using basic operations like addition or subtraction
    - If we see non-linear relationships (curvilinear patterns, exponential growth, logarithmic decay), it can be more challenging but also more rewarding if accurately mode
        - Polynomial Features: For quadratic or higher-order relationships, consider generating polynomial features
        - Use logarithmic or exponential technique to linearize these non-linear reletionships

I consider this approach is more complex, so I have only visualized the pair plots, and left the analysis as a potential future extension. Here are the code to visualize with seaborn and matplotib.pyplot

```
import seaborn as sns
import matplotlib.pyplot as plt

#    1.5.1. Sample a subset of columns if the dataset is very large, or use PCA or Factor Analysis to reduce dimensionality first
selected_columns = train_data_filtered.columns[:10]  # Adjust this as needed
sns.pairplot(train_data_filtered[selected_columns], diag_kind='kde')
plt.show()
```

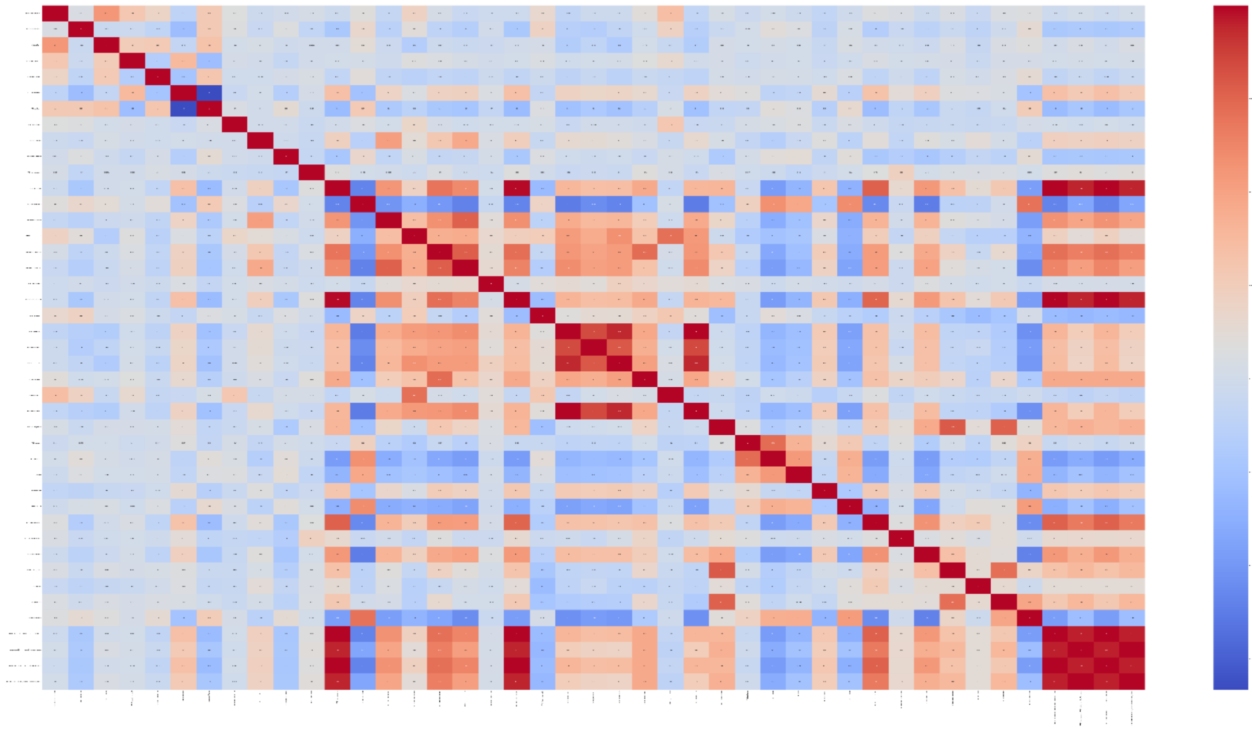
and the output:



- Heatmap (Correlation matrix): I chose this one for my analysis. I have used two techniques to create composite features where needed. First, let see the implementation of visualization of heatmap of correlation matrix:

```
#     1.5.2. Plot Correlation Matrix and Analysis the Heatmap
correlation_matrix = train_data_filtered.corr()

correlation_matrix.to_csv('correlation_matrix.csv', index=False)

# files.download('correlation_matrix.csv')

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.rcParams['figure.figsize'] = [200, 100]
plt.show()
```

and the output of the Heatmap:



The two techniques involves in the two cases of correlation value:

- Pairs of features with High positive correlation (close to 1): given these features are feature_A and feature_B, I created two new features:
  - feature_A_plus_B_sum = feature_A + feature_B.
  - feature_A_plus_B_product = featureA * feature_B
- Pairs of features with High negative correlation (close to -1): given these are feature_A and feature_B, I also created two new:
  - feature_A_plus_B_diff: = feature_A – feature_B, to emphasize their inverse relationship.
  - feature_A_plus_B_ratio = feature_A / feature_B, to emphasize their proportional inverse effects more distinctly.
- Also, I also implement the division so that it avoid division by zero by adding a very small value (“epsilon”)
- The value determining whether high positive/negative correlation or not, is >0.9 or <-0.9 respectively

Here is the full implementation:

```
# Set the threshold for high correlation
high_positive_threshold = 0.9
high_negative_threshold = -0.9

# Find pairs of highly correlated features
highly_pos_correlated_pairs = []
highly_neg_correlated_pairs = []

for i in correlation_matrix.columns:
    for j in correlation_matrix.columns:
        if i != j:  # avoid self-comparison
            if correlation_matrix.loc[i, j] > high_positive_threshold:
                highly_pos_correlated_pairs.append((i, j))
            elif correlation_matrix.loc[i, j] < high_negative_threshold:
                highly_neg_correlated_pairs.append((i, j))

print("-------------------------------------------------------------------------------------------------------------------------------")
print("Highly Positively Correlated Pairs:", highly_pos_correlated_pairs)
print("Highly Negatively Correlated Pairs:", highly_neg_correlated_pairs)

# Make a copy to save the data from previous task
train_data_with_composited_features = train_data_filtered.copy()

# Add or multiply positively correlated features
for feature1, feature2 in highly_pos_correlated_pairs:
    train_data_with_composited_features[f'{feature1}_{feature2}_sum'] = train_data_filtered[feature1] + train_data_filtered[feature2]
    train_data_with_composited_features[f'{feature1}_{feature2}_product'] = train_data_filtered[feature1] * train_data_filtered[feature2]

# Subtract or create ratios for negatively correlated features
for feature1, feature2 in highly_neg_correlated_pairs:
    train_data_with_composited_features[f'{feature1}_{feature2}_diff'] = train_data_filtered[feature1] - train_data_filtered[feature2]

    epsilon = 0.001  # A small constant to avoid division by zero

    train_data_with_composited_features[f'{feature1}_{feature2}_ratio'] = train_data_filtered[feature1] / (train_data_filtered[feature2] + epsilon)  # handle division by zer

print("\n-------------------------------------------------------------------------------------------------------------------------------")
print("Training data after composited features:")
print(train_data_with_composited_features.head())

# Re-add categorical columns and the target feature to the composited data
train_data_final = pd.concat([train_data_with_composited_features, train_data_balanced[categorical_cols]], axis=1)
```

With the above implementation, here are the pair of features that is used to composition:

[('FFTE Discharge solids', 'FFTE Production solids PV'),

('FFTE Discharge solids', 'FFTE Discharge solids_FFTE Production solids PV_sum'),

('FFTE Discharge solids', 'FFTE Discharge solids_FFTE Production solids PV_product'),

('FFTE Discharge solids', 'FFTE Production solids PV_FFTE Discharge solids_sum'),

('FFTE Discharge solids', 'FFTE Production solids PV_FFTE Discharge solids_product'),

('FFTE Production solids PV', 'FFTE Discharge solids'),

('FFTE Production solids PV', 'FFTE Discharge solids_FFTE Production solids PV_sum'),

('FFTE Production solids PV', 'FFTE Discharge solids_FFTE Production solids PV_product'),

('FFTE Production solids PV', 'FFTE Production solids PV_FFTE Discharge solids_sum'),

('FFTE Production solids PV', 'FFTE Production solids PV_FFTE Discharge solids_product'),

('FFTE Temperature 1 - 1', 'FFTE Temperature 2 - 1'),

('FFTE Temperature 1 - 1', 'FFTE Temperature 3 - 2'),

('FFTE Temperature 2 - 1', 'FFTE Temperature 1 - 1'),

('FFTE Temperature 2 - 1', 'FFTE Temperature 3 - 2'),

('FFTE Temperature 3 - 2', 'FFTE Temperature 1 - 1'),

('FFTE Temperature 3 - 2', 'FFTE Temperature 2 - 1'), ('FFTE Discharge solids_FFTE Production solids PV_sum', 'FFTE Discharge solids'),

('FFTE Discharge solids_FFTE Production solids PV_sum', 'FFTE Production solids PV'),

('FFTE Discharge solids_FFTE Production solids PV_sum', 'FFTE Discharge solids_FFTE Production solids PV_product'),

('FFTE Discharge solids_FFTE Production solids PV_sum', 'FFTE Production solids PV_FFTE Discharge solids_sum'),

('FFTE Discharge solids_FFTE Production solids PV_sum', 'FFTE Production solids PV_FFTE Discharge solids_product'),

('FFTE Discharge solids_FFTE Production solids PV_product', 'FFTE Discharge solids'),

('FFTE Discharge solids_FFTE Production solids PV_product', 'FFTE Production solids PV'),

('FFTE Discharge solids_FFTE Production solids PV_product', 'FFTE Discharge solids_FFTE Production solids PV_sum'),

('FFTE Discharge solids_FFTE Production solids PV_product', 'FFTE Production solids PV_FFTE Discharge solids_sum'),

('FFTE Discharge solids_FFTE Production solids PV_product', 'FFTE Production solids PV_FFTE Discharge solids_product'),

('FFTE Production solids PV_FFTE Discharge solids_sum', 'FFTE Discharge solids'),

('FFTE Production solids PV_FFTE Discharge solids_sum', 'FFTE Production solids PV'),

('FFTE Production solids PV_FFTE Discharge solids_sum', 'FFTE Discharge solids_FFTE Production solids PV_sum'),

('FFTE Production solids PV_FFTE Discharge solids_sum', 'FFTE Discharge solids_FFTE Production solids PV_product'),

('FFTE Production solids PV_FFTE Discharge solids_sum', 'FFTE Production solids PV_FFTE Discharge solids_product'),

('FFTE Production solids PV_FFTE Discharge solids_product', 'FFTE Discharge solids'), ('FFTE Production solids PV_FFTE Discharge solids_product', 'FFTE Production solids PV'),

('FFTE Production solids PV_FFTE Discharge solids_product', 'FFTE Discharge solids_FFTE Production solids PV_sum'),

('FFTE Production solids PV_FFTE Discharge solids_product', 'FFTE Discharge solids_FFTE Production solids PV_product'),

('FFTE Production solids PV_FFTE Discharge solids_product', 'FFTE Production solids PV_FFTE Discharge solids_sum')]

To conclude, there are 35 composite features added to the train dataset. Combine with the existing features in the train dataset, and the categorical features re-added, there are a total of 110 features use for training purposes.

The data is saved in "train_data_final". Here is the link for the CSV file: https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%204/train_data_final.csv

Step 2: Feature Selection, Model training and Evaluation

Feature Selection

For this task, I simply use the sklearn.feature_selection.SelectKBest class, which have been introduce in the last portfolio. Here is the implementation:

```
# STEP 2: FEATURE SELECTION, MODEL TRAINING AND EVALUATION


#   2.1. FEATURE SELECTION
from sklearn.feature_selection import SelectKBest, f_classif

# Split to X and y for training
X = train_data_final.drop('Class', axis=1)  # Independent variables
y = train_data_final['Class']  # Target variable

# Select top k features based on ANOVA F-test
selector = SelectKBest(score_func=f_classif, k=10)  # Adjust k as necessary
X_selected = selector.fit_transform(X, y)

# Get selected feature names
features_selected = X.columns[selector.get_support(indices=True)]
print("------------------------------------------------------------------------
print("Selected features:", features_selected)
```

Using this technique, I have chosen these features for the training process:

```
------------------------------------------------------------------------------------------------------------------------
Selected features: Index(['TFE Out flow SP', 'FFTE Temperature 1 - 1', 'FFTE Temperature 3 - 2',
       'FFTE Temperature 1 - 1_FFTE Temperature 3 - 2_sum',
       'FFTE Temperature 1 - 1_FFTE Temperature 3 - 2_product',
       'FFTE Temperature 2 - 1_FFTE Temperature 1 - 1_product',
       'FFTE Temperature 2 - 1_FFTE Temperature 3 - 2_product',
       'FFTE Temperature 3 - 2_FFTE Temperature 1 - 1_sum',
       'FFTE Temperature 3 - 2_FFTE Temperature 1 - 1_product',
       'FFTE Temperature 3 - 2_FFTE Temperature 2 - 1_product'],
      dtype='object')
```

Model training

For the 5 models used for training, besides the required Random Forset Classifier, I have chosen: Decision Tree Classifier, Logistic Regression, Gradient Boosting Classifier, K-nearest Neighbors Classifier (KNN). These classes are imported from: sklearn.ensemble.RandomForestClassifier, sklearn.tree.DecisionTreeClassifier, sklearn.linear_model.LogisticRegression, sklearn.ensemble.GradientBoostingClassifier, sklearn.neighbors.KneighborsClassifier

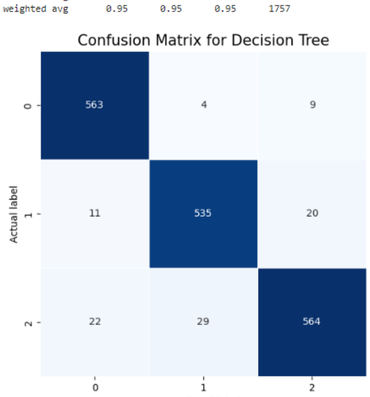
Here is my implementation:

```python
#   2.2. MODEL TRAINING

import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import seaborn as sns

# Data Preparation
X = train_data_final[features_selected]  # Assuming 'features_selected' contains the names of the selected features
y = train_data_final['Class']

# Data Standardization
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.25, random_state=1)

# Models to train
models = {
    'Decision Tree': DecisionTreeClassifier(),
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'Random Forest': RandomForestClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'KNN': KNeighborsClassifier()
}

# Function to plot confusion matrix
def plot_confusion_matrix(cm, title='Confusion Matrix'):
    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt="d", linewidths=.5, square=True, cmap='Blues', cbar=False)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.title(title, size=15)

# Training and evaluating models
for name, model in models.items():
    print("\n-----------------------------------------------------------------------------------------

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Accuracy and Classification Report
    accuracy = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    print(f"Results for {name}:")
    print(f"Accuracy: {accuracy}")
    print(f"Classification Report:\n{report}")

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    plot_confusion_matrix(cm, title=f'Confusion Matrix for {name}')

    plt.show()  # Display the confusion matrix for each model
```

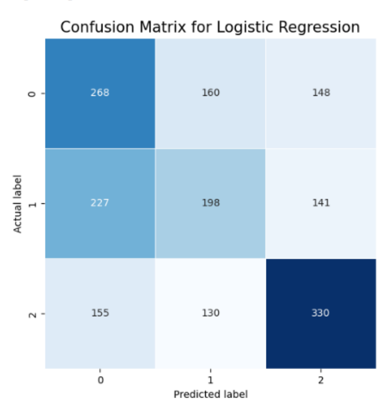
Model evaluation

In the above implementation, after training with each model, I have plotted the output of accuracy rate (with sklearn.metrics.accuracy_score), and classification report (with sklearn.metrics.classification_report). Here is the output for each case:

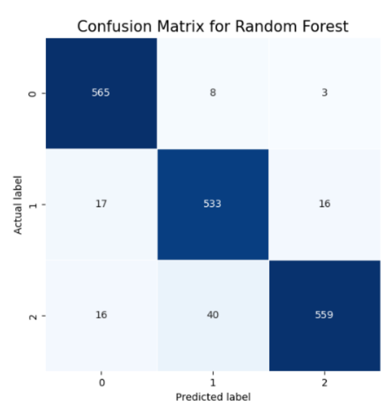
```
--------------------------------------------------------------------------------
Results for Decision Tree:
Accuracy: 0.9459305634604439
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       576
           1       0.94      0.95      0.94       566
           2       0.95      0.92      0.93       615

    accuracy                           0.95      1757
   macro avg       0.95      0.95      0.95      1757
weighted avg       0.95      0.95      0.95      1757
```

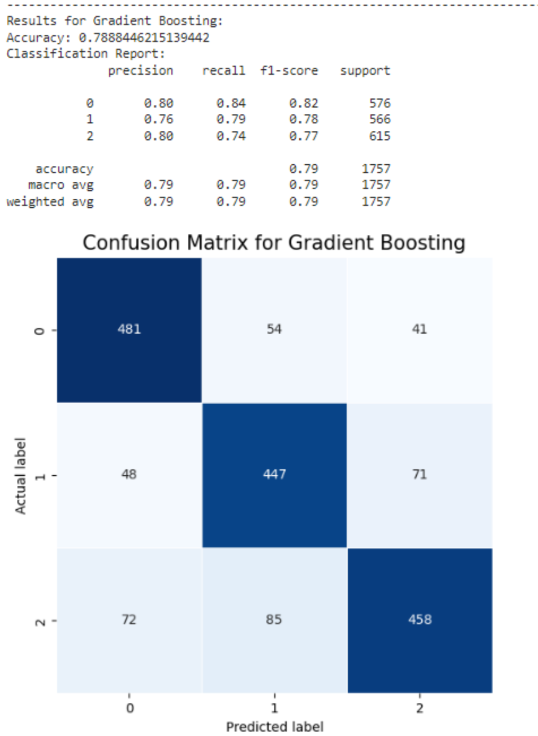


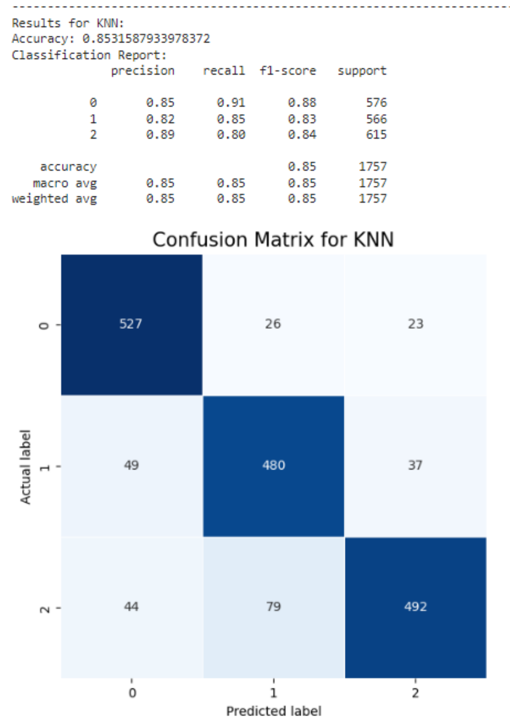

```
Results for Logistic Regression:
Accuracy: 0.45304496300512237
Classification Report:
              precision    recall  f1-score   support

           0       0.41      0.47      0.44       576
           1       0.41      0.35      0.38       566
           2       0.53      0.54      0.53       615

    accuracy                           0.45      1757
   macro avg       0.45      0.45      0.45      1757
weighted avg       0.45      0.45      0.45      1757
```




```
--------------------------------------------------------------------------------
Results for Random Forest:
Accuracy: 0.9430848036425725
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.98      0.96       576
           1       0.92      0.94      0.93       566
           2       0.97      0.91      0.94       615

    accuracy                           0.94      1757
   macro avg       0.94      0.94      0.94      1757
weighted avg       0.94      0.94      0.94      1757
```







Here is the summary table of training:

Model	**Class**	**Precision**	**Recall**	**F1-Score**	**Overall Accuracy**
Decision Tree	0	0.93	0.98	0.96	0.94
	1	0.95	0.91	0.93	
	2	0.95	0.93	0.94	
Logistic Regression	0	0.41	0.46	0.43	0.46
	1	0.44	0.36	0.39	
	2	0.52	0.56	0.54	
Gradient Boosting	0	0.83	0.85	0.84	0.80
	1	0.76	0.8	0.77	
	2	0.81	0.75	0.78	
KNN	0	0.86	0.91	0.89	0.83
	1	0.80	0.80	0.80	
	2	0.84	0.79	0.83	
Random Forest	0	0.96	0.97	0.96	0.94
	1	0.92	0.91	0.91	
	2	0.93	0.93	0.93	

From this table, especially the training accuracy, we can conclude that:

- Both Random Forest and Decision Tree exhibits extremely high F1-scores and overall accuracy, making them formidable competitors, particularly for classes 0 and 1. This suggest that the **Random Forest appears to be the most robust model**, followed closely by the classifier DecisionTree.
- Because logistic regression may be not able to capture non-linear correlations, it struggles severely across all metrics, suggesting that it may not be appropriate for this particular dataset.

- All classes saw balanced performance from gradient boosting, albeit marginally worse than the top models.
- When compared to Random Forest, KNN performs well, particularly when it comes to recall for class 0, although it exhibits certain shortcomings in terms of precision and recall for class 2.

Here is my assumption of why Random Forrest is the most efficient model in the train process:

- It is more suitable for capturing complexity of the nonlinear relationships between features
- It averages multiple decision trees, and reduce the change of overfitting

Finally, I have saved and downloaded the model of RandomForest:

```
import joblib

# Save the seclected model - Random Forest CLassifier
for name, model in models.items():
    if name == 'Random Forest':

        # Save the RandomForest model to disk using joblib
        model_filename = 'random_forest_model.joblib'
        joblib.dump(model, model_filename)
        print(f"The selected model '{name}' has been saved successfully as {model_filename}")

        files.download(model_filename)
```

Here is the link to this saved model:
https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%204/random_forest_model.joblib

Step 3: ML to AI

Preparing

For this task, I loaded the saved Random Forest model. For the test data, it is 900-rows test dataset have been prepared in Step 1. I applied the same techniques for training data:

- Remove constant columns
- Resolve any imbalances
- Make composite features, the pairs of original features used have been determined before (saved in “highly_pos_correlated_pairs” and “highly_neg_correlated_pairs”)
- Split the test data into input features (X_test) and labels (y_test). The features used for X_test had also previously determined with SelectKBest (saved in “features_selected”)
- Normalized with StandardScaler

```python
# STEP 3: ML TO AI

# Load the Random Forest model from file
model_filename = 'random_forest_model.joblib'
random_forest_model = joblib.load(model_filename)

# Return to the test data processed in Step 1

#   Remove constant columns
constant_columns = [col for col in test_data.columns if test_data[col].nunique() == 1]
if constant_columns:
    test_data = test_data.drop(columns=constant_columns)

#   Resolve any imbalances
class_distribution = test_data['Class'].value_counts(normalize=True)
is_imbalanced = class_distribution.min() < 0.20

if is_imbalanced:
    test_data_balanced = None

    smote = SMOTE(random_state=1)
    features, target = smote.fit_resample(test_data.drop('Class', axis=1),
                                          test_data['Class'])
    test_data_balanced = pd.concat([features, pd.DataFrame(target, columns=['Class'])], axis=1)
else:
    test_data_balanced = test_data

#   Make composite features

for feature1, feature2 in highly_pos_correlated_pairs:
    test_data_balanced[f'{feature1}_{feature2}_sum'] = test_data_balanced[feature1] + test_data_balanced[feature2]
    test_data_balanced[f'{feature1}_{feature2}_product'] = test_data_balanced[feature1] * test_data_balanced[feature2]

for feature1, feature2 in highly_neg_correlated_pairs:
    test_data_balanced[f'{feature1}_{feature2}_diff'] = test_data_balanced[feature1] - test_data_balanced[feature2]

    epsilon = 0.001  # A small constant to avoid division by zero

    test_data_balanced[f'{feature1}_{feature2}_ratio'] = test_data_balanced[feature1] / (test_data_balanced[feature2] + epsilon)  # handle division by zero


# Split the test data into features and labels
X_test = test_data_balanced[features_selected] # Features have been selected above
y_test = test_data_balanced['Class']  # true labels

# Normalize the test data
scaler = StandardScaler()
X_test_scaled = scaler.fit_transform(X_test)
```

Make Predictions and Compare

I have used the uploaded Random Forest model, fitting the X_test_scaled data:

```python
# Make predictions with the Random Forest model
y_pred_test = random_forest_model.predict(X_test_scaled)

# Compare predictions with the actual labels
from sklearn.metrics import classification_report, accuracy_score

print("Classification Report for Test Data:")
print(classification_report(y_test, y_pred_test))
print("Accuracy:", accuracy_score(y_test, y_pred_test))
```

and, here is the output:

```
Classification Report for Test Data:
              precision    recall  f1-score   support

           0       0.75      0.52      0.61       300
           1       0.61      0.61      0.61       300
           2       0.58      0.76      0.66       300

    accuracy                           0.63       900
   macro avg       0.65      0.63      0.63       900
weighted avg       0.65      0.63      0.63       900

Accuracy: 0.63
```

As you can see, the model cannot reach the efficient result as in the training process. The overall accuracy for the testing is only about 63%, along with the reduction of other indices in the classification report. I think the major reason is that the parameters used for testing dataset ("highly_pos_correlated_pairs" and "highly_neg_correlated_pairs" for making new composited features, "features_selected" for selecting best input features) have been pre-determined, and specified for the training dataset only.

Step 4: Develop rules from ML model

For this task, I reused that train data after step 1 – Data Preparation (before choosing best features), which is saved in "train_data_final". Now, I have forwarded the data into the Decision Tree Classifier, and output the rules (using sklearn.tree.export_text):

```python
# STEP 4: DEVELOP RULES FROM ML MODEL

from sklearn.tree import export_text

# Filter columns that end with 'SP'
sp_features = [col for col in train_data_final.columns if col.endswith('SP')]
X = train_data_final[sp_features]
y = train_data_final['Class']  # Ensure 'Class' is the target variable

# Initialize and train the Decision Tree using the full dataset
dt_classifier = DecisionTreeClassifier(max_depth=5, random_state=1)
dt_classifier.fit(X, y)

# Make predictions on the training set
y_pred_train = dt_classifier.predict(X)

# Print the output
print("\n-----------------------------------------------------------------------------------
print("Classification Report for Training Data:")
print(classification_report(y, y_pred_train))

print("\n-----------------------------------------------------------------------------------
print("Accuracy:", accuracy_score(y, y_pred_train))

# Print the decision tree rules
tree_rules = export_text(dt_classifier, feature_names=sp_features)
print("\n-----------------------------------------------------------------------------------
print(f"Tree rules: \n{tree_rules}")
```

And, this is the output:

```
------------------------------------------------------------------------------------------------------------------------------------

Classification Report for Training Data:

              precision    recall  f1-score   support


           0       0.60      0.76      0.67      2342

           1       0.78      0.39      0.52      2342

           2       0.58      0.71      0.64      2342


    accuracy                           0.62      7026

   macro avg       0.65      0.62      0.61      7026

weighted avg       0.65      0.62      0.61      7026



------------------------------------------------------------------------------------------------------------------------------------

Accuracy: 0.6216908625106746


------------------------------------------------------------------------------------------------------------------------------------

Tree rules:

|--- FFTE Feed flow SP <= 10165.00

|   |--- FFTE Steam pressure SP <= 122.18

|   |   |--- FFTE Feed flow SP <= 9225.00

|   |   |   |--- FFTE Production solids SP <= 41.31

|   |   |   |   |--- FFTE Steam pressure SP <= 102.00

|   |   |   |   |   |--- class: 1

|   |   |   |   |--- FFTE Steam pressure SP >  102.00

|   |   |   |   |   |--- class: 2

|   |   |   |--- FFTE Production solids SP >  41.31

|   |   |   |   |--- TFE Out flow SP <= 2178.90

|   |   |   |   |   |--- class: 1

|   |   |   |   |--- TFE Out flow SP >  2178.90

|   |   |   |   |   |--- class: 0

|   |   |--- FFTE Feed flow SP >  9225.00

|   |   |   |--- FFTE Out steam temp SP <= 49.58

|   |   |   |   |--- FFTE Feed flow SP <= 9750.00

|   |   |   |   |   |--- class: 2

|   |   |   |   |--- FFTE Feed flow SP >  9750.00

|   |   |   |   |   |--- class: 1

|   |   |   |--- FFTE Out steam temp SP >  49.58

|   |   |   |   |--- FFTE Steam pressure SP <= 93.20

|   |   |   |   |   |--- class: 1

|   |   |   |   |--- FFTE Steam pressure SP >  93.20

|   |   |   |   |   |--- class: 0

|   |--- FFTE Steam pressure SP >  122.18

|   |   |--- FFTE Feed flow SP <= 9450.00

|   |   |   |--- TFE Out flow SP <= 2014.46

|   |   |   |   |--- FFTE Production solids SP <= 39.75

|   |   |   |   |   |--- class: 1

|   |   |   |   |--- FFTE Production solids SP >  39.75

|   |   |   |   |   |--- class: 0

|   |   |   |--- TFE Out flow SP >  2014.46
```

```
|   |   |   |   |--- TFE Out flow SP <= 2269.35
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- TFE Out flow SP >  2269.35
|   |   |   |   |   |--- class: 0
|   |   |--- FFTE Feed flow SP >  9450.00
|   |   |   |--- TFE Steam pressure SP <= 79.40
|   |   |   |   |--- class: 0
|   |   |   |--- TFE Steam pressure SP >  79.40
|   |   |   |   |--- FFTE Out steam temp SP <= 49.94
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- FFTE Out steam temp SP >  49.94
|   |   |   |   |   |--- class: 2
|--- FFTE Feed flow SP >  10165.00
|   |--- TFE Production solids SP <= 84.00
|   |   |--- TFE Production solids SP <= 76.50
|   |   |   |--- FFTE Out steam temp SP <= 50.23
|   |   |   |   |--- TFE Production solids SP <= 66.50
|   |   |   |   |   |--- class: 2
|   |   |   |   |--- TFE Production solids SP >  66.50
|   |   |   |   |   |--- class: 2
|   |   |   |--- FFTE Out steam temp SP >  50.23
|   |   |   |   |--- FFTE Out steam temp SP <= 50.37
|   |   |   |   |   |--- class: 0
|   |   |   |   |--- FFTE Out steam temp SP >  50.37
|   |   |   |   |   |--- class: 2
|   |   |--- TFE Production solids SP >  76.50
|   |   |   |--- TFE Production solids SP <= 78.50
|   |   |   |   |--- TFE Production solids SP <= 77.50
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- TFE Production solids SP >  77.50
|   |   |   |   |   |--- class: 2
|   |   |   |--- TFE Production solids SP >  78.50
|   |   |   |   |--- TFE Production solids SP <= 82.00
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- TFE Production solids SP >  82.00
|   |   |   |   |   |--- class: 1
|   |--- TFE Production solids SP >  84.00
|   |   |--- FFTE Feed tank level SP <= 37.50
|   |   |   |--- class: 0
|   |   |--- FFTE Feed tank level SP >  37.50
|   |   |   |--- FFTE Steam pressure SP <= 120.05
|   |   |   |   |--- class: 1
|   |   |   |--- FFTE Steam pressure SP >  120.05
|   |   |   |   |--- class: 2
```

With the above output, the following table is my extraction to define specific rules for each class (among the values of 1, 2, or 3), storing the class and corresponding condition(s) so that the class can be predicted:

Prediction: Class 0	**Prediction: Class 1**	**Prediction: Class 2**
FFTE Feed flow SP <= 10165.00 AND FFTE Steam pressure SP > 122.18 AND FFTE Feed flow SP > 9450.00 AND TFE Steam pressure SP <= 79.40	FFTE Feed flow SP <= 10165.00 AND FFTE Steam pressure SP <= 122.18 AND FFTE Feed flow SP <= 9225.00 AND FFTE Production solids SP <= 41.31 AND FFTE Steam pressure SP <= 102.00	FFTE Feed flow SP <= 10165.00 AND FFTE Steam pressure SP <= 122.18 AND FFTE Feed flow SP <= 9225.00 AND FFTE Production solids SP <= 41.31 AND FFTE Steam pressure SP > 102.00
FFTE Feed flow SP <= 10165.00 AND FFTE Steam pressure SP > 122.18 AND FFTE Feed flow SP > 9450.00 AND TFE Steam pressure SP > 79.40 AND FFTE Out steam temp SP <= 49.94	FFTE Feed flow SP <= 10165.00 AND FFTE Steam pressure SP <= 122.18 AND FFTE Feed flow SP <= 9225.00 AND FFTE Production solids SP > 41.31 AND TFE Out flow SP <= 2178.90	FFTE Feed flow SP <= 10165.00 AND FFTE Steam pressure SP <= 122.18 AND FFTE Feed flow SP > 9225.00 AND FFTE Out steam temp SP <= 49.58 AND FFTE Feed flow SP <= 9750.00
FFTE Feed flow SP > 10165.00 AND TFE Production solids SP <= 84.00 AND FFTE Out steam temp SP > 50.23 AND FFTE Out steam temp SP <= 50.37	FFTE Feed flow SP <= 10165.00 AND FFTE Steam pressure SP > 122.18 AND FFTE Feed flow SP <= 9450.00 AND TFE Out flow SP <= 2014.46 AND FFTE Production solids SP <= 39.75	FFTE Feed flow SP <= 10165.00 AND FFTE Steam pressure SP <= 122.18 AND FFTE Feed flow SP > 9225.00 AND FFTE Out steam temp SP > 49.58 AND FFTE Steam pressure SP > 93.20
FFTE Feed flow SP > 10165.00 AND TFE Production solids SP > 84.00 AND FFTE Feed tank level SP <= 37.50	FFTE Feed flow SP > 10165.00 AND TFE Production solids SP <= 84.00 AND TFE Production solids SP > 76.50 AND TFE Production solids SP <= 78.50 AND TFE Production solids SP <= 77.50	FFTE Feed flow SP <= 10165.00 AND FFTE Steam pressure SP > 122.18 AND FFTE Feed flow SP > 9450.00 AND TFE Steam pressure SP > 79.40 AND FFTE Out steam temp SP > 49.94
	FFTE Feed flow SP > 10165.00 AND TFE Production solids SP > 78.50 AND TFE Production solids SP <= 82.00	FFTE Feed flow SP > 10165.00 AND TFE Production solids SP <= 84.00 AND TFE Production solids SP <= 76.50 AND FFTE Out steam temp SP <= 50.23 AND TFE Production solids SP <= 66.50
	FFTE Feed flow SP > 10165.00 AND TFE Production solids SP > 84.00 AND FFTE Feed tank level SP > 37.50 AND FFTE Steam pressure SP <= 120.05	FFTE Feed flow SP > 10165.00 AND TFE Production solids SP <= 84.00 AND TFE Production solids SP > 76.50 AND TFE Production solids SP > 78.50 AND TFE Production solids SP > 77.50
		FFTE Feed flow SP > 10165.00 AND TFE Production solids SP > 84.00 AND FFTE Feed tank level SP > 37.50 AND FFTE Steam pressure SP > 120.05