Name: Trung Kien Nguyen

Student ID: 104053642

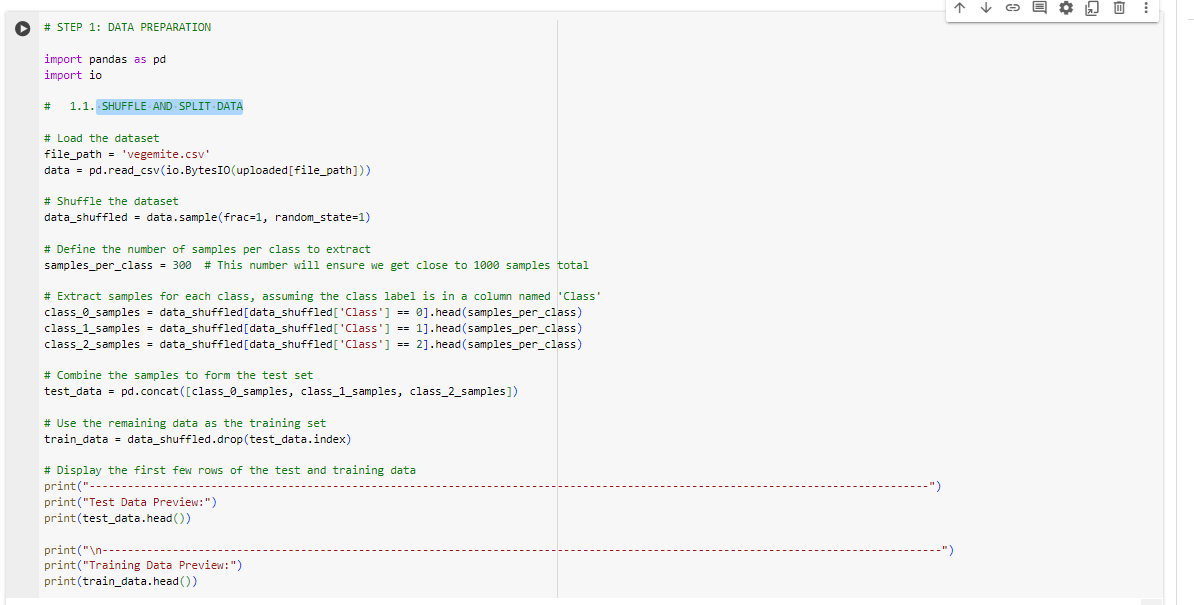
Studio: 1 – 3

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

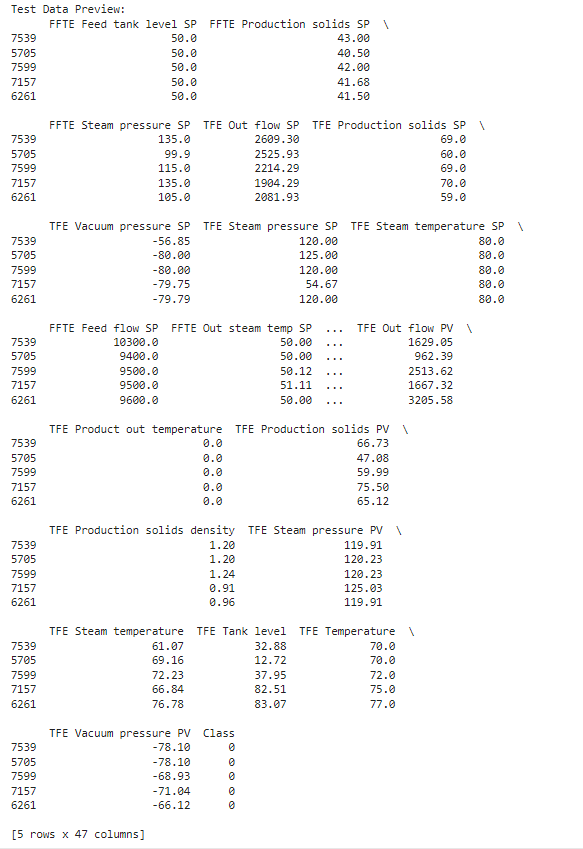
## Shuffle and Split Data



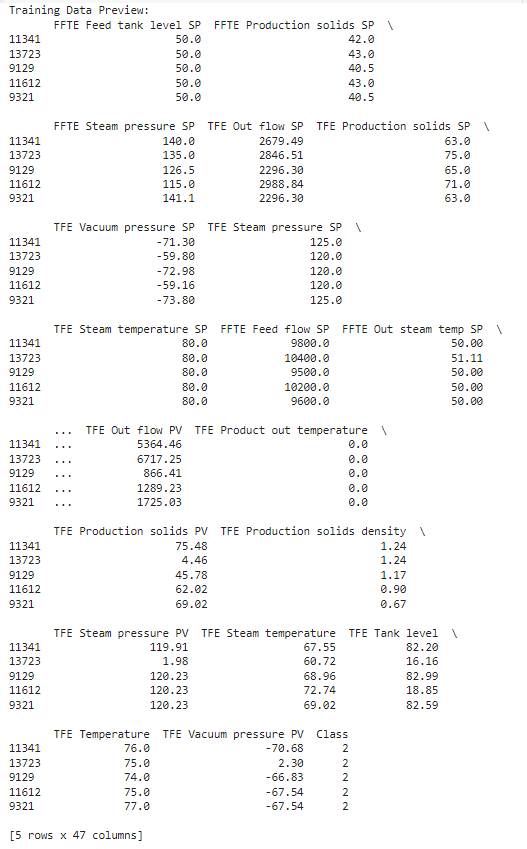
Initially, I have shuffled the whole uploaded dataset (in the given file “vegemite.csv”), using the “sample” method of pandas.DataFrame.

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*



*Train data preview*

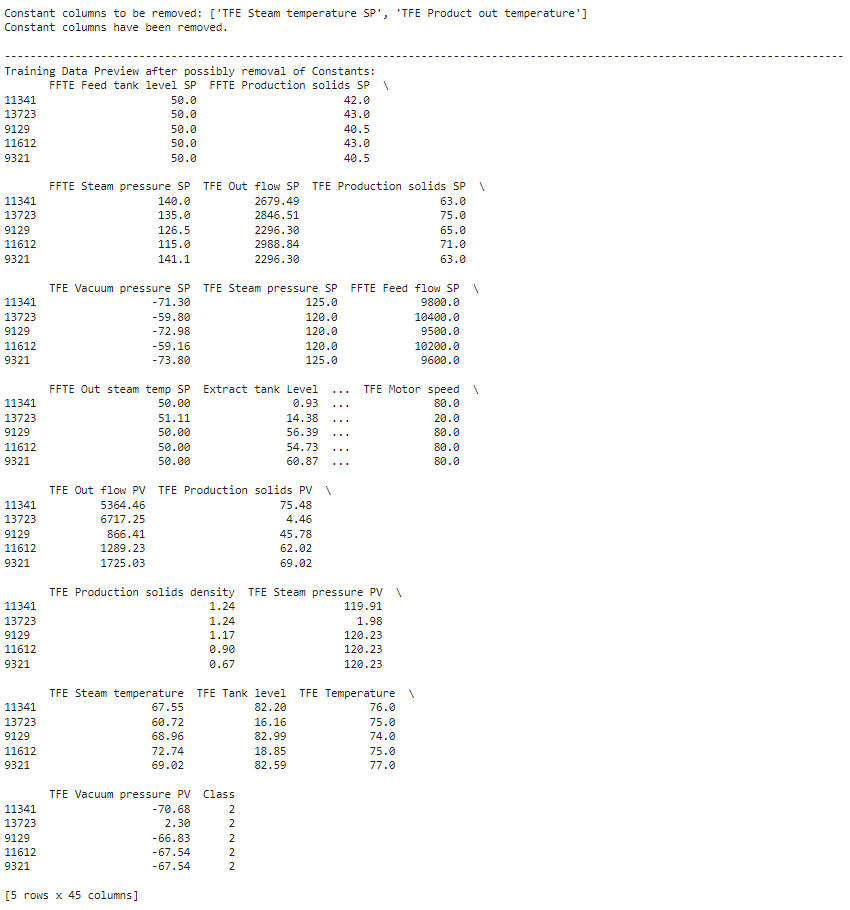
## Remove constant columns



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.

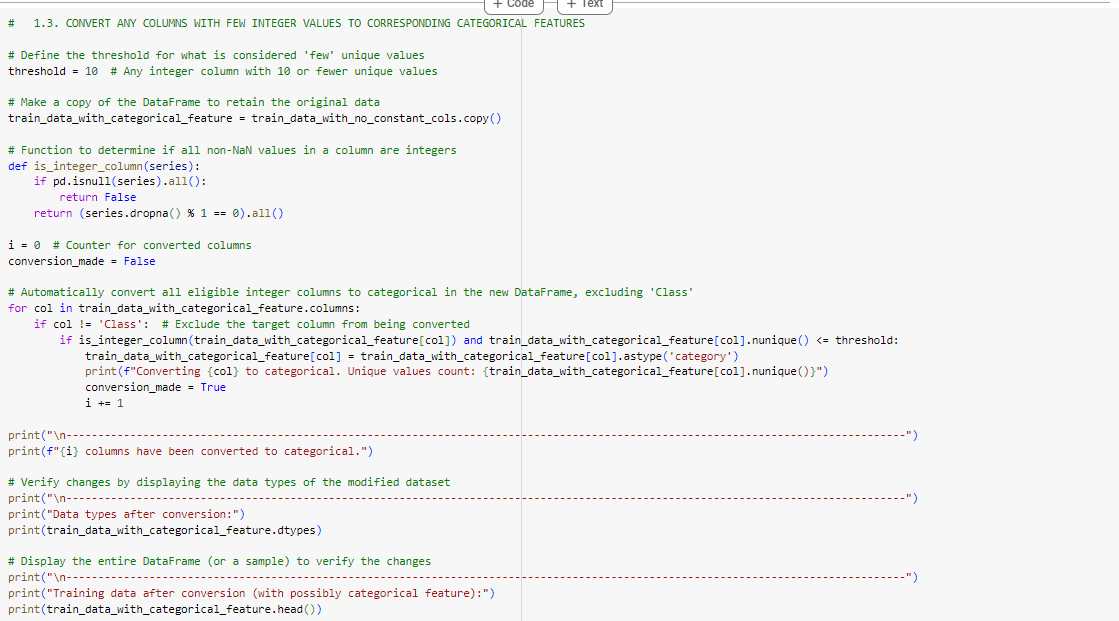
I then removed all constant column by simply using “drop” method of pandas.DataFrame.

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



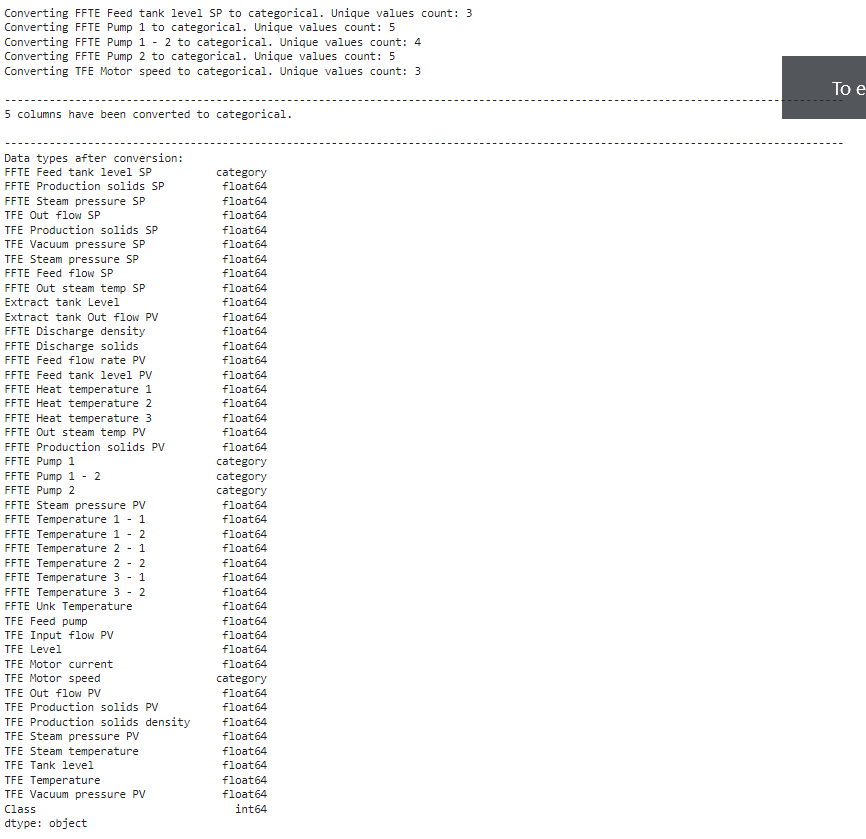
*The two constrant features “TFE Steam temperature SP” and “TFE Product out temperature” have been removed*

## Convert columns with few integer values to categorical features



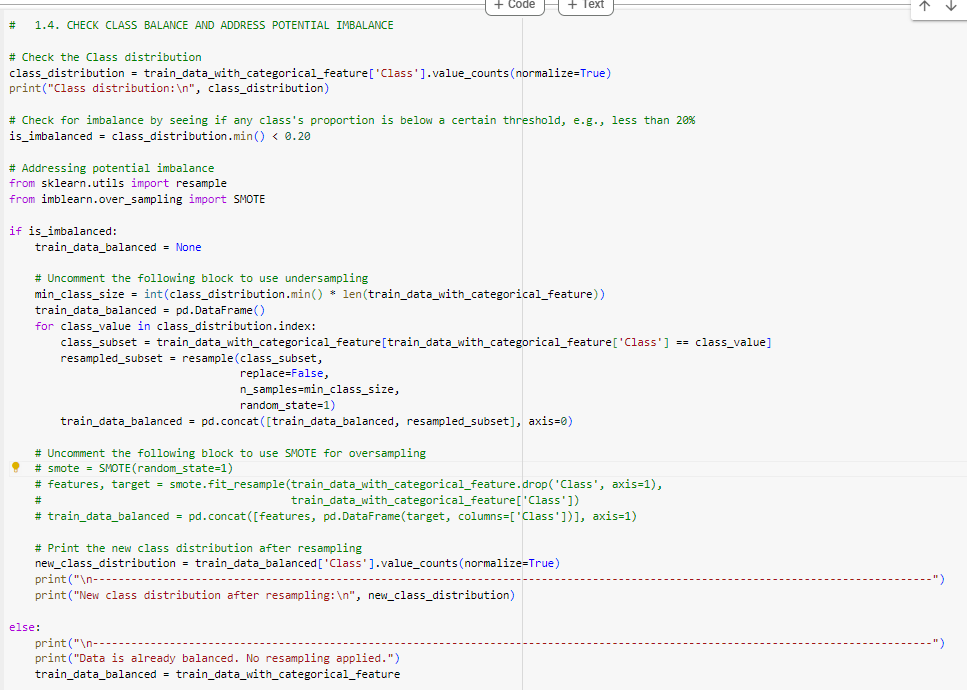
To do this, I have checked in each column, counted the number of unique integers. I determine that if the number of unique integer in a feature column is less than 10 (“threshold”), then it must be converted to categorical features. Also, previously when I uploaded the dataset, the features’ values

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



*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



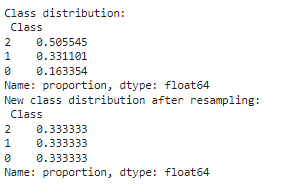
In the implementation for checking class balance, I have counted the distribution of each class ( number of each class’s value’s divide to the number of total samples. Then, if the min in the classes’ distribution is less than a certain value, given that 20%, I stated there is an imbalance in the training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

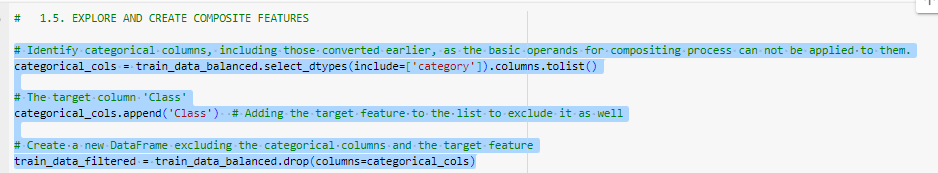
In the train dataset after converting some columns to categorical features, there is a noticable imbalance, which has been resolved:



## 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.

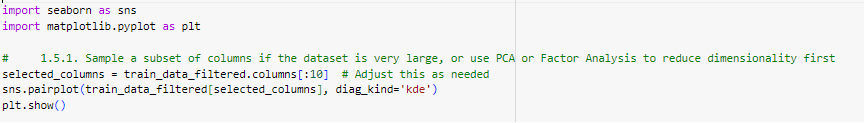
Also note that, the category features have been converted before is not coverred by the above operands, I have to temporarily drop them, and re-add after the composite features creating has done:



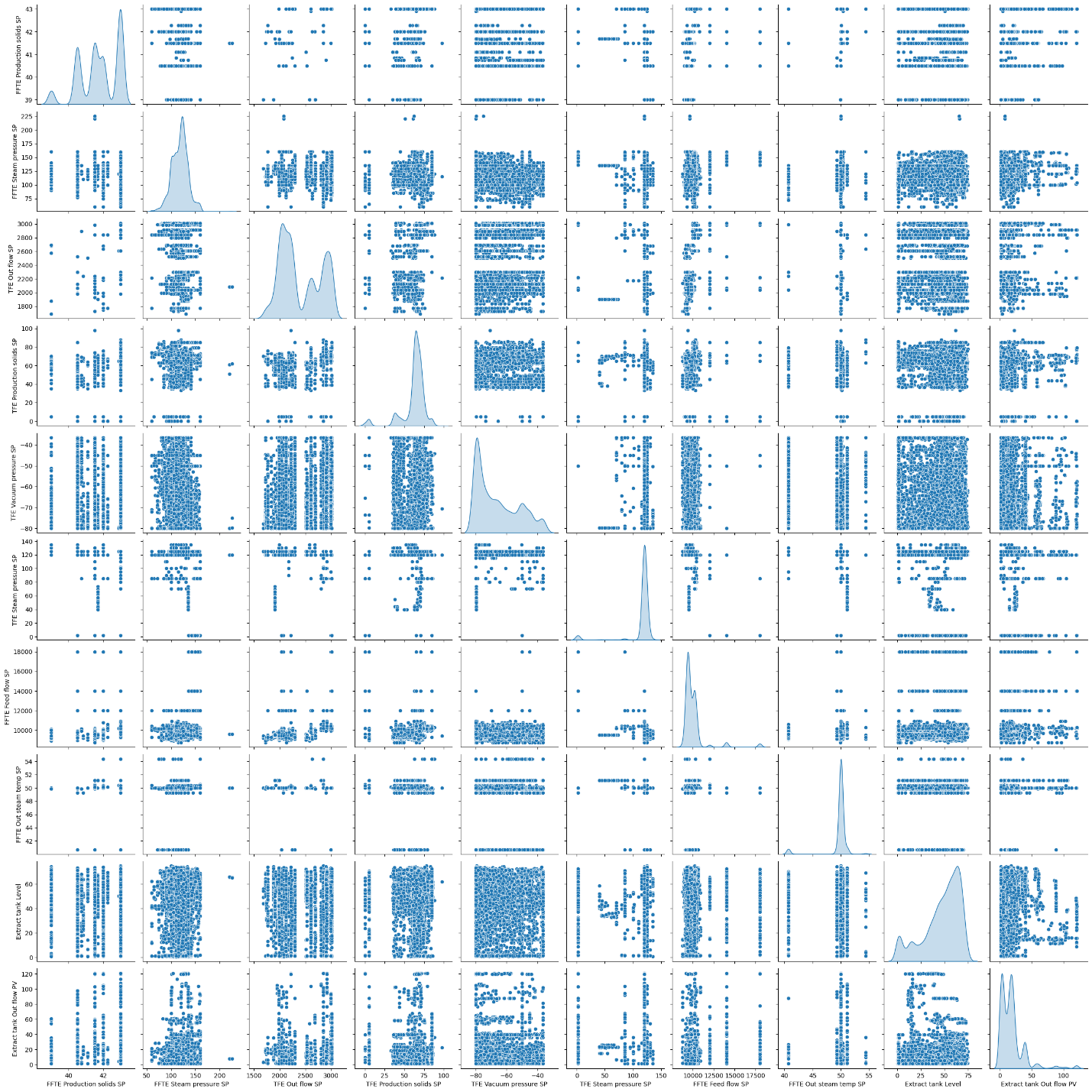
There are two common ways of doing this analysis, using either Pair Plots or Heatmap. I have implemented the code to plot the two types of illustrations.

* 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

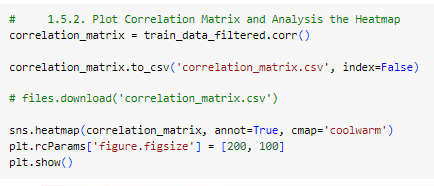
I consider this approach is more complex, so I have only visualized the pair plots, and left the analysis as a potential future extension. Here are the code to visualize with seaborn and matplotib.pyplot



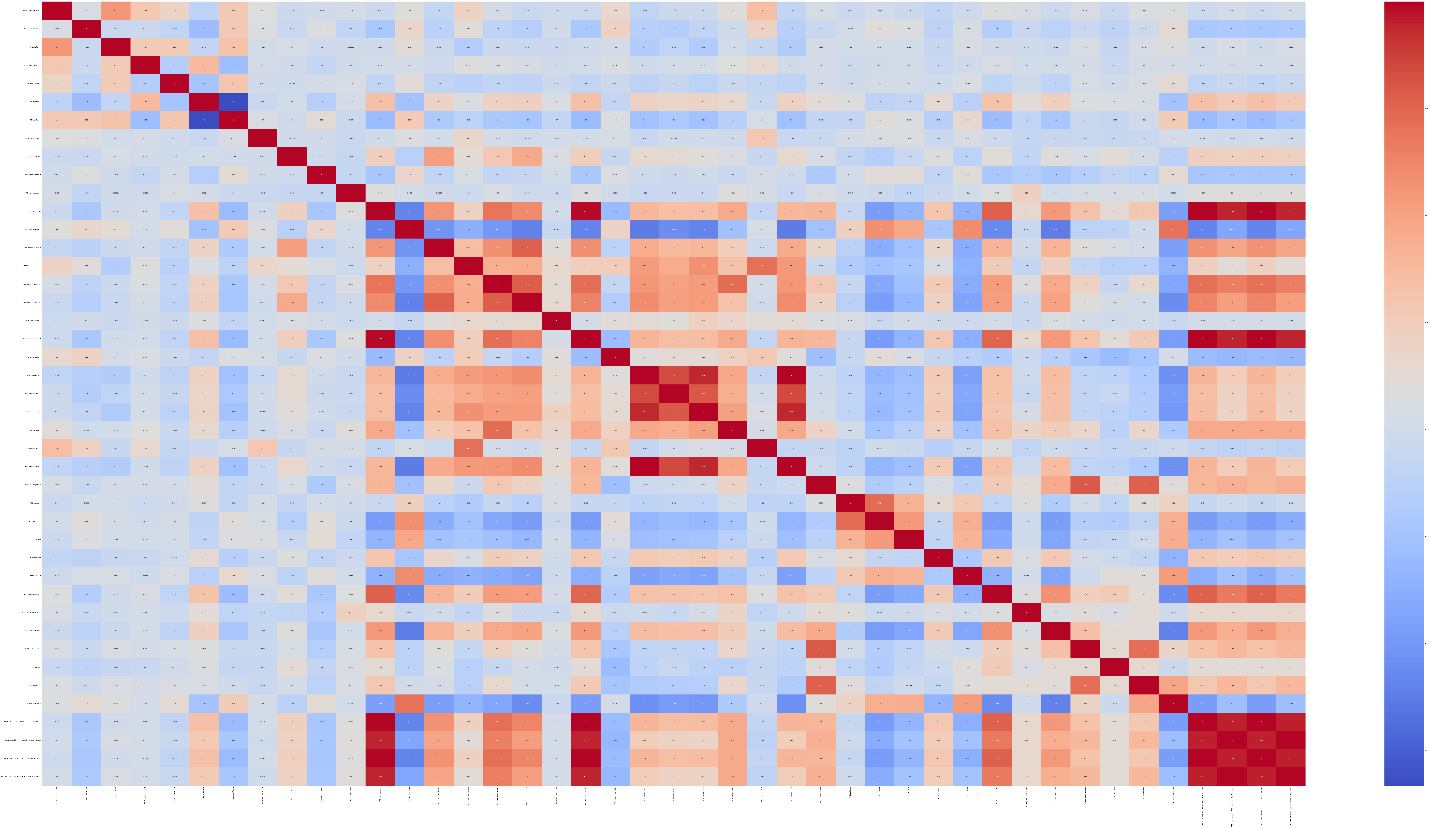
and the output:



* 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:



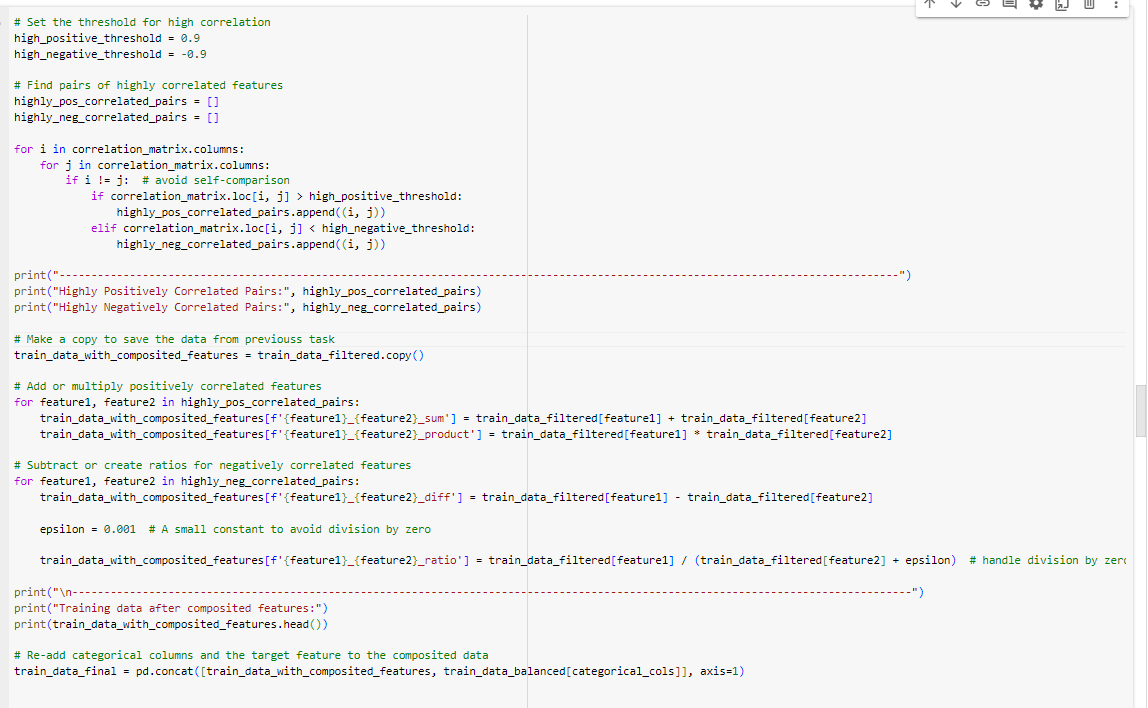
and the output of the Heatmap:



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:



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')]

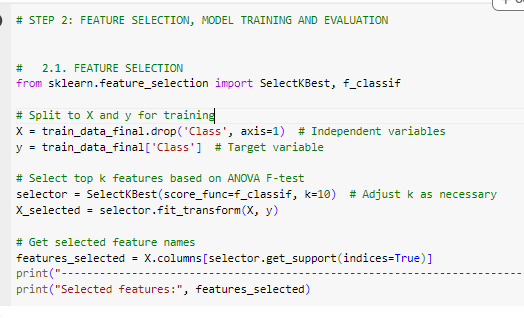
To conclude, there are 35 composite features added to the train dataset. Combine with the existing features in the train dataset, and the categorical features re-added, there are a total of 110 features use for training purposes.

The data is saved in “train\_data\_final”. Here is the link for the CSV file:

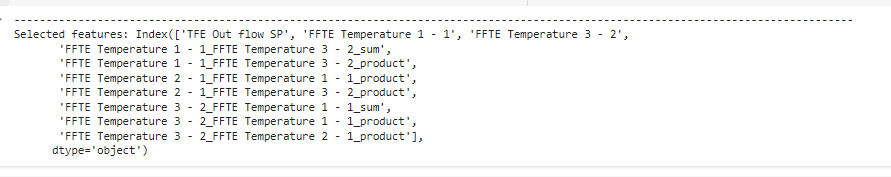
# Step 2: Feature Selection, Model training and Evaluation

## 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:



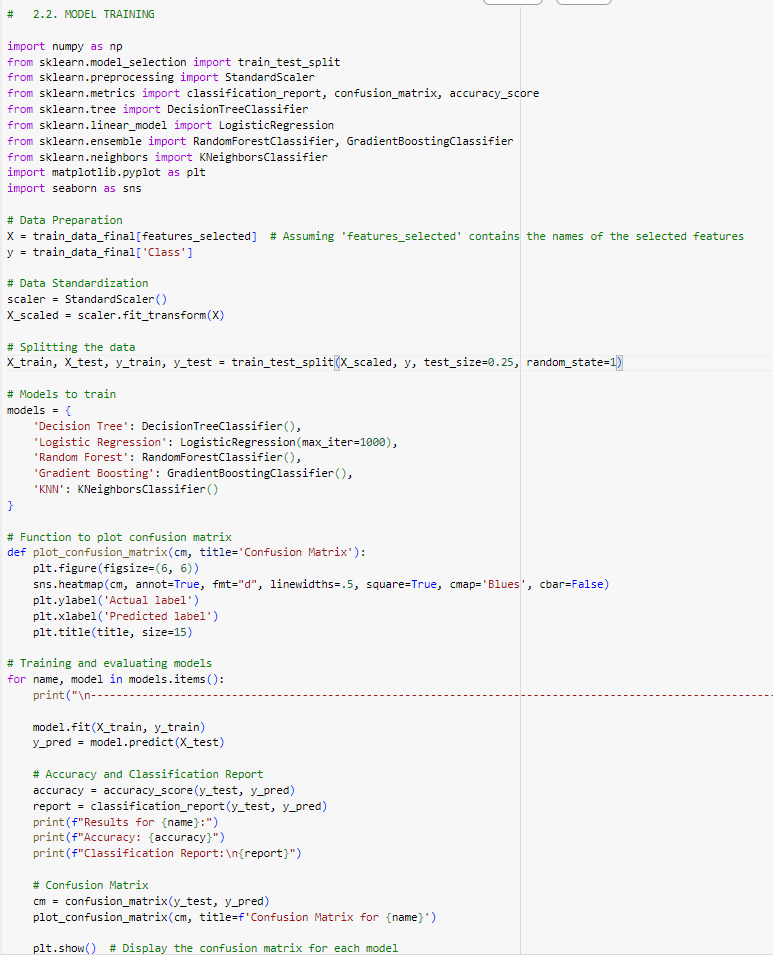
Using this technique, I have chosen these features for the training process:



## Model training

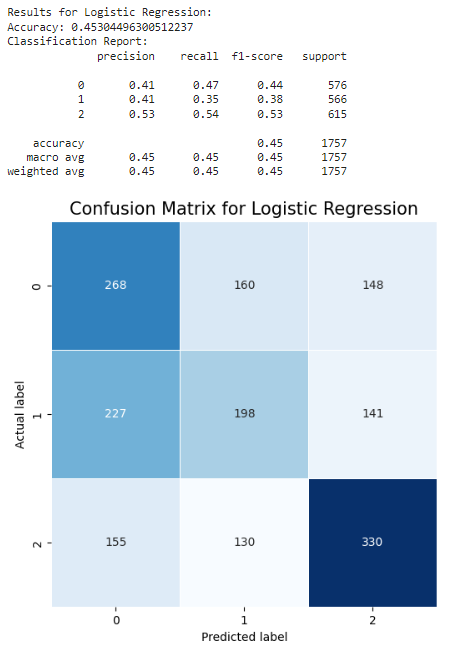
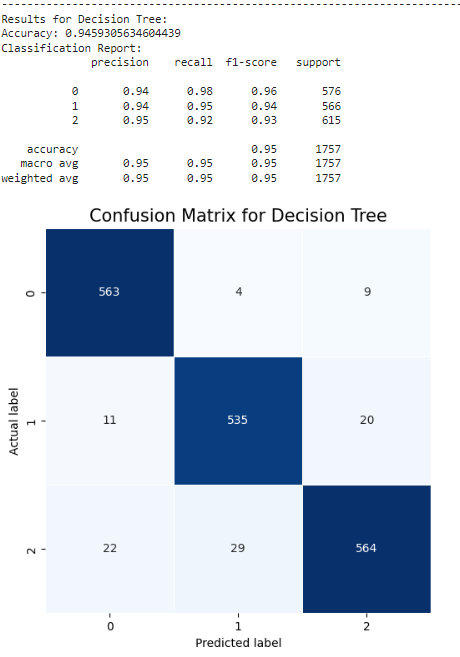
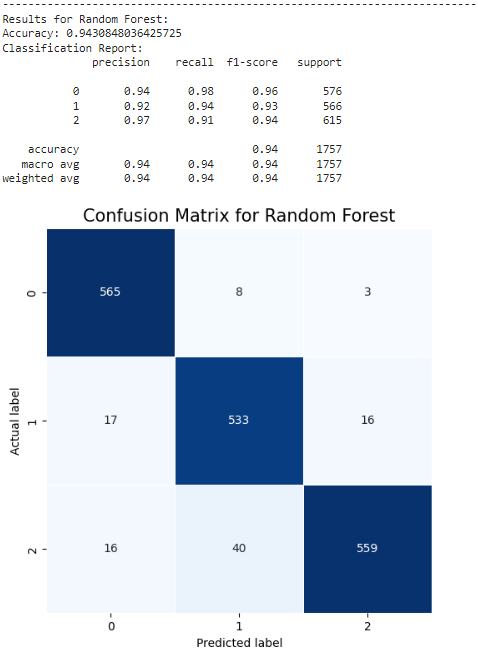
For the 5 models used for training, besides the required Random Forset Classifier, I have chosen: Decision Tree Classifier, Logistic Regression, Gradient Boosting Classifier, K-nearest Neighbors Classifier (KNN). These classes are imported from: sklearn.ensemble.RandomForestClassifier, sklearn.tree.DecisionTreeClassifier,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

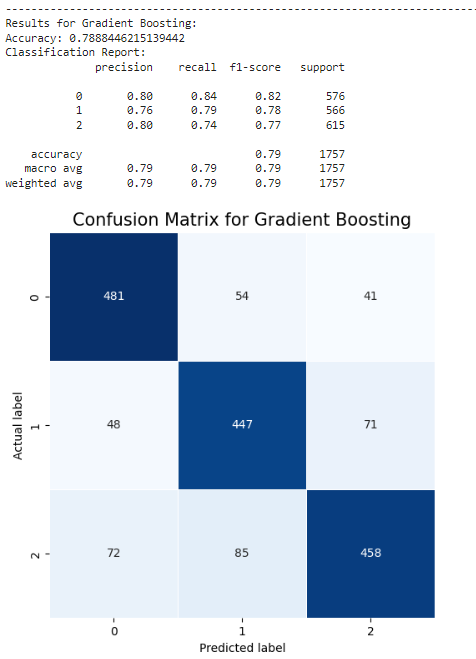
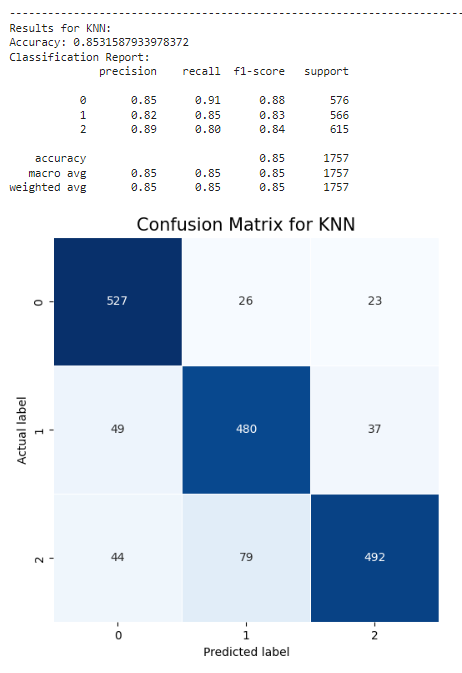
Here is my implementation:



## 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:

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 |

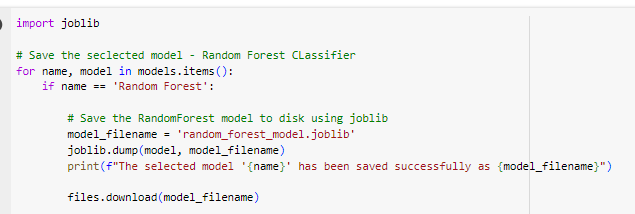
From this table, especially the training accuracy, we can conclude that:

* 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:



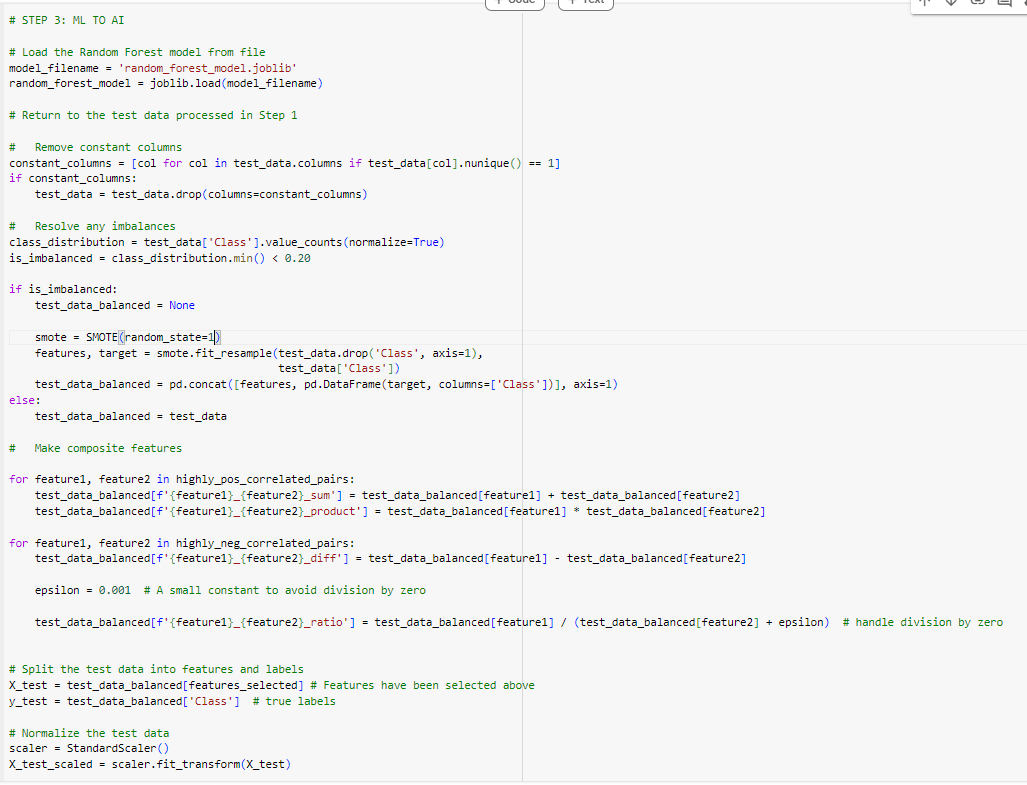
Here is the link to this saved model:

# Step 3: ML to AI

## Preparing

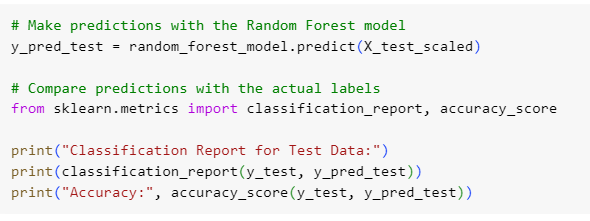
For this task, I loaded the saved Random Forest model. For the test data, it is 900-rows test dataset have been prepared in Step 1. I applied the same techniques for training data:

* 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

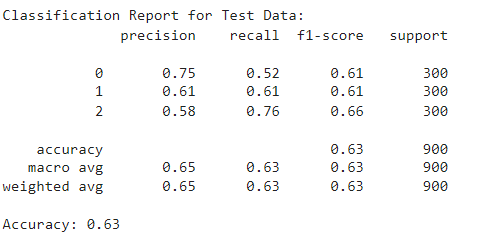


## Make Predictions and Compare

I have used the uploaded Random Forest model, fitting the X\_test\_scaled data:



and, here is the output:



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”