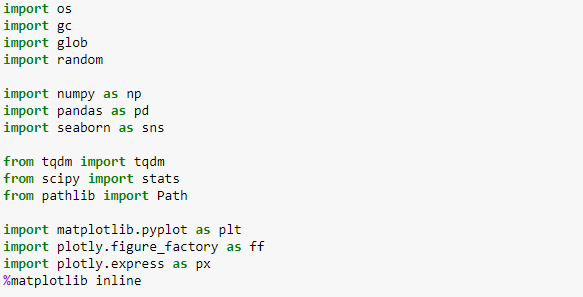
**TPS22Nov - Golden results & KNN & LGBM 1**

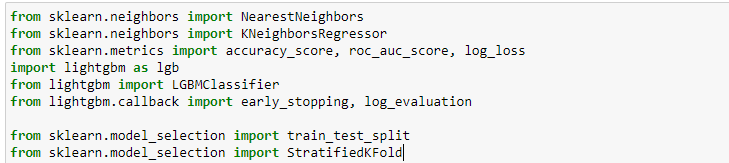
# Created by Kaushik Kar

# Employment id- 2216027

# Importing Libraries:

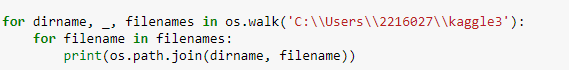
First, importing the important external Python packages using the pip package manager.





1. NumPy is used for mathematical operations like addition, subtraction, multiplication, division, etc. on arrays and matrices.
2. Pandas provides data structures for efficiently storing and manipulating large datasets, and tools for reading and writing data to and from various file formats, including CSV, Excel, and SQL databases
3. Seaborn is a data visualization library based on Matplotlib which is a plotting library used for creating static, interactive, and animated visualizations in Python.
4. OS This library provides a way to interact with the operating system, allowing you to perform operations such as reading and writing files, creating directories, and running system commands.
5. GC This library provides tools for managing the memory used by Python programs, allowing you to control when and how memory is allocated and deallocated.
6. Glob This library provides a way to find all the pathnames matching a specified pattern according to the rules used by the Unix shell, although results are returned in arbitrary order.
7. Random This library provides tools for generating random numbers and sequences, as well as shuffling and sampling items from a sequence.
8. tqdm library, which provides a progress bar that can be used to track the progress of a loop or iterator.
9. stats module from the scipy library, which provides a collection of statistical functions and tools.
10. Path class from the pathlib library, which provides a way to work with file paths and directories.
11. KNeighborsRegressor class from the sklearn.neighbors module, which is a class for supervised regression using the k-nearest neighbors algorithm.
12. Import the accuracy\_score, roc\_auc\_score, and log\_loss functions from the sklearn.metrics module, which are used for evaluating classification models.
13. lightgbm is a gradient boosting framework that uses tree-based learning algorithms.
14. LGBMClassifier class from the lightgbm library, which is a class for classification using the LightGBM algorithm.
15. early\_stopping and log\_evaluation callbacks from the lightgbm.callback module. These are used to enable early stopping and log evaluation metrics during model training.
16. train\_test\_split function from the sklearn.model\_selection module, which is used for splitting data into training and testing sets.
17. StratifiedKFold class from the sklearn.model\_selection module, which is a cross-validation technique that preserves the percentage of samples for each class.

This below code is using the OS module to walk through a directory tree and print out the path of each file in the tree such as data.csv and sample\_submission.csv



**2.Data Import**

With the help of Pandas library, we can read and upload the data in csv form.



Import the train labels csv and assign the dataframe to labels and make the index column as id and extract the label column from labels and assign it to the y variable then count the unique values from the y variable and plot in the above.

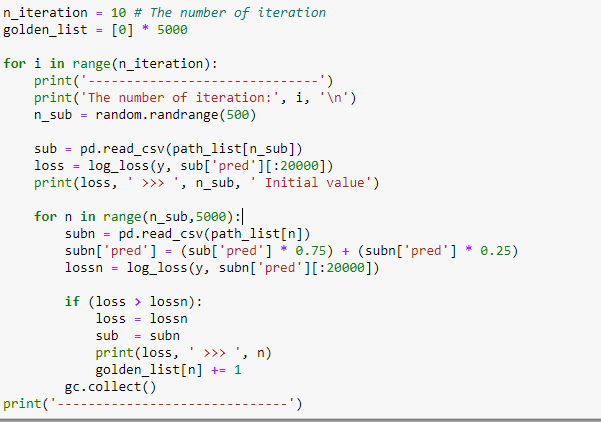
Use the glob function from the glob module to find all files matching the path files pattern, sort the resulting list of file paths in reverse order, and assign it to a variable path list.

Read in the first CSV file in the path list variable and assign it to a variable sub0.

Compute the logarithmic loss between the true labels in y and the predicted probabilities in the pred column of sub0, using only the first 20,000 examples. Assign the resulting loss value to a variable loss0. And the values is “0.7575039918285069”

1. **Sharing the Coefficient 25%,15%,5%:**

* **For 25%:**



* First, set the number of iteration and make a list of 5000 zeros so that it can keep track of how many times each model has been selected for the final ensemble.
* For each iteration print the specified iteration number and also a separator line to make the output more readable.
* Applying the logloss method between the true labels in y and the predicted probabilities in the pred column of sub0, using only the first 20,000 examples. Print the initial loss value and the index of the initial submission file.
* Then start a nested loop and upload the dataframe for the current path and Combine the predictions of the current submission file with the predictions of the initial submission file, using a weight of 0.75 for the initial submission and 0.25 for the current submission. Compute the log loss between the true labels in y and the combined predictions in the pred column of subn, using only the first 20,000 examples.
* Check if the combined predictions from the current submission file result in a lower log loss than the previous best.
* loss = lossn: If the current submission file results in a lower log loss, update the best loss value.
* sub = subn: If the current submission file results in a lower log loss, update the best submission file.
* Print the new best loss value and the index of the current submission file.
* golden\_list[n] += 1: Increment the count of how many times the current submission file has been selected for the final ensemble. And then make the dataframe as golden\_df of golden\_list. returns the proportion of occurrences of each unique value in the DataFrame, sorted in descending order.



That is, more than 0.9770 results (4885 cases) were never good in "Ensembling". But on the other hand, 0.0014 results (7 cases) have been effective in every ten "Ensembling".



creates a numpy array g\_list from the list golden\_list, excluding any elements that are equal to 0. Then, a Pandas DataFrame g\_df is created from this numpy array. The DataFrame contains the frequency count of how many times each index value in the filtered golden\_list was updated during the iterative process.

The above code creates an empty list golden. Then, a for loop iterates through each index k in the range of the length of golden\_list. If the value at index k in golden\_list is not equal to 0, a new list [k, golden\_list[k]] is appended to golden.

Finally, save the 'golden25.npy' file in the directory.

* **For 15%:**

Similarly for the 15% iterate the same things and run those code and we can find That is, more than 0.97 results (4853 cases) were never good in "Ensembling". But on the other hand, 0.003 results (15 cases) have been effective in every ten "Ensembling".

Then again create a numpy array list and excluding the elements that are equal to 0 and the create a pandas dataframe and also create a goldenlist append the nonzero values and make a npy file named as ‘golden15.npy’.

* **For 5%:**

Similarly for the 5% iterate the same things and run those code and we can That is, more than 0.941 results (4708 cases) were never good in "Ensembling". But on the other hand, 0.0108 results (54 cases) have been effective in every ten "Ensembling".

Then again create a numpy array list and excluding the elements that are equal to 0 and the create a pandas dataframe and also create a goldenlist append the nonzero values and make a npy file named as ‘golden5.npy’

**TPS22Nov - Golden results & KNN & LGBM 2**

We have already imported the important library and import the data as we have discussed about it above.



Now load the golden215.npy file and set the variable name as g15 and then creates a new NumPy array g15\_10 by filtering the elements of g15 using a list comprehension. The condition in the list comprehension is that the second element of each element in g15 (presumably each element is a two-element array or tuple) is equal to 10.

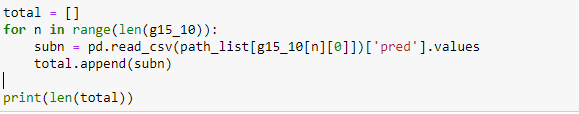


Create a function named hist\_data and then run a for loop. Within the loop, a file is loaded using Pandas read\_csv function, based on the file path specified by path\_list[data[i][0]]. The [:20000] slice takes only the first 20,000 rows of the loaded file. The function then prints out the log loss of the loaded file with respect to some target values y and some predicted values sub['pred'], as well as summary statistics of the second column of the loaded file using the describe() method of a Pandas DataFrame.

Next, a list is created containing the second column of the loaded file, and a group label is created for this list. Then, a distribution plot is created using Plotly's create\_distplot function, with the list of data to plot, the group label, a bin size of 0.2, and with the histogram and rug plot turned off. Finally, the plot is displayed using the show() method of the created figure object.

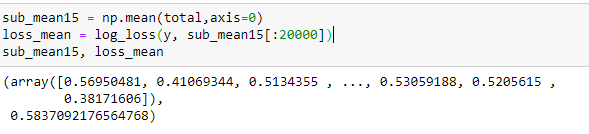
Finally, the hist\_data function is called with the NumPy array g15\_10 as its argument. The expected behavior of the function is to print out some information about each of the files specified in g15\_10, and to create and display a distribution plot of the second column of each file.

**Data Total:**



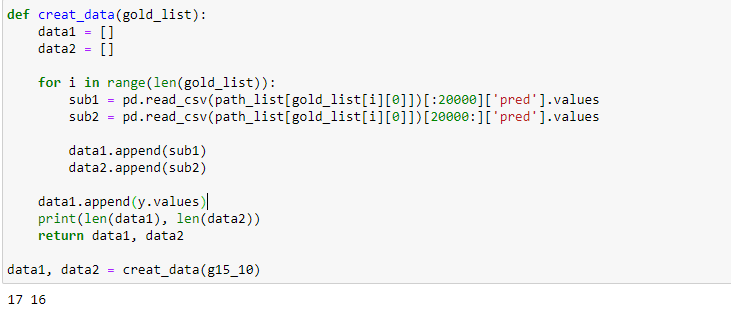
this code appears to be creating a list total that contains the "pred" column from each of the files specified in g15\_10. The expected behaviour is to have a list of NumPy arrays, where each array corresponds to the "pred" column of a file. The length of total would be equal to the number of files specified in g15\_10.

**Mean Submission:**



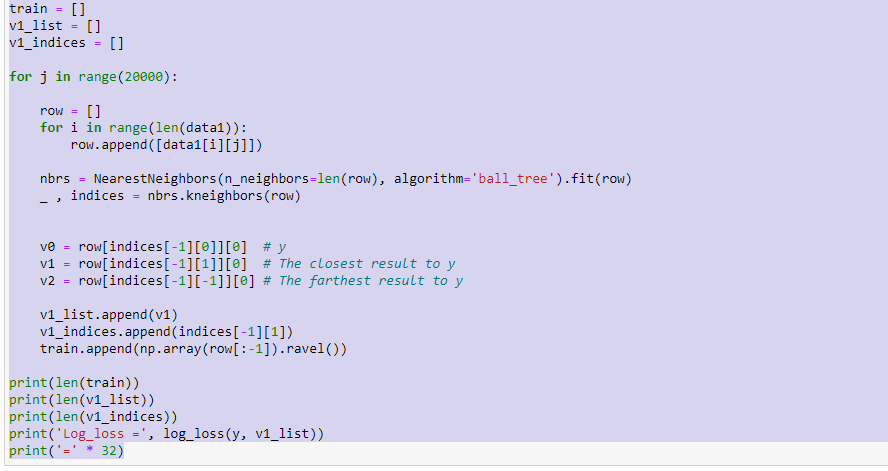
this code appears to be calculating the mean of the "pred" columns from each file specified in g15\_10 and storing it in sub\_mean15. It also calculates the log loss of sub\_mean15 with respect to some target values y and stores it in loss\_mean. The expected behavior is to return a tuple with the mean of the "pred" columns and its log loss.

**NearestNeighbors (Closest to the target):**



this code appears to be creating two lists data1 and data2 that contain the "pred" columns from each of the files specified in g15\_10. The first 20,000 elements of each "pred" column are stored in data1, while the rest of the elements are stored in data2. The expected behavior is to return a tuple with two lists, one containing the first 20,000 elements of the "pred" columns and another containing the remaining elements of the "pred" columns, as well as the target values y.

**Data Train:**



The code appears to be generating a training set by finding the closest neighbors to each y target value from the "pred" columns of the files in g15\_10. The training set is stored in the list train, while the closest neighbor's "pred" values are stored in v1\_list. The expected behavior is to return a list train containing the concatenated "pred" columns of each file in g15\_10, with the corresponding closest neighbor's "pred" values stored in v1\_list.

**Data Test:**

test set in a similar manner as the training set. It loops over 20,000 indices, and within each iteration, a new empty list row is created. Then, a second loop is executed over the length of data2, which should be equal to the number of files in g15\_10.

After the loop completes, the length of test is printed. The expected behavior is to return a test set containing the concatenated "pred" columns of each file in g15\_10, except for the first 20,000 values of each file, which were used to generate the training set.

**KNeighborsRegressor:**

KNeighborsRegressor model with 299 neighbors to the train dataset generated earlier, using the target values y as labels. Then, it predicts the labels of the test dataset using the predict method of the trained KNeighborsRegressor model and assigns the result to sub1.This predicts the "pred" values for the remaining (i.e. last 299) rows of each file in g15\_10, based on the nearest neighbors found in the training set. The resulting sub1 array should have length 299\*len(g15\_10), since there are 299 predictions for each file in g15\_10.

**LGBM:**

In the first approach, you're using KNN to find the nearest neighbor for each test instance based on the training data, and you use the predicted values from the nearest neighbor as the final prediction. This method is called K-Nearest Neighbor Regression.

In the second approach, you're using LightGBM classifier to train on the training data and then predict on the test data. The predicted probabilities are then used as the final prediction.