**READ\_ME**

**AIM** : To classify high and low-risk cancer patients using machine learning and deep learning models

**DATA PROVIDED** : Train, Test and Sample data

**Usage**

* Environment Setup

Ensure you have Python and the required libraries installed. You can use Python package managers like pip to install missing dependencies.

* Data Preparation

Place your training data in a file named "kaggle\_train.csv" and your test data in a file named "kaggle\_test.csv."

* Run the Code

Execute the Python script by running the code. Make sure the script's requirements, such as input data files, are correctly configured.

* Review Results

After running the code, you will find multiple result files (e.g., "result1.csv," "result2.csv") with the final predictions.

**EXTRA TREES CLASSIFIER**

The ExtraTreesClassifier, short for Extremely Randomized Trees Classifier, is a popular machine learning algorithm that falls under the ensemble learning category. It's closely related to Random Forests but with some key differences. Here's a more detailed explanation of the ExtraTreesClassifier:

* Ensemble Learning:

ExtraTreesClassifier is an ensemble learning method, which means it combines the predictions from multiple individual models (in this case, decision trees) to make more accurate predictions.

* Decision Trees:

Like Random Forests, ExtraTreesClassifier uses decision trees as the base models. Decision trees are binary trees that make decisions based on a set of rules learned from the training data.

* Randomization:

The "extra" in ExtraTrees comes from the additional randomization it introduces during the training process. While a Random Forest selects the best split among a random subset of features, ExtraTrees goes a step further and selects the best split among all features at each node. This increased randomization can reduce overfitting.

* Bootstrapping:

ExtraTreesClassifier uses bootstrapping, which means it creates multiple bootstrap samples (subsets) from the training data. Each tree is trained on a different bootstrap sample.

Aggregation of Predictions:

The predictions made by individual decision trees are aggregated to produce a final prediction. For classification tasks, this can be done by taking a majority vote, and for regression tasks, it's often the mean of the predictions.

In summary, the ExtraTreesClassifier is a powerful ensemble learning method that builds upon the concept of decision trees while introducing additional randomization to improve generalization and reduce overfitting. It's a valuable tool in the machine learning toolkit and is often used in conjunction with other ensemble methods to create highly accurate predictive models.

**DESCRIPTION OF THE CODE**

* **IMPORTING LIBRARIES**: The code begins by importing various libraries and modules. These libraries are crucial for different aspects of the data analysis and machine learning pipeline. Some of the important libraries include scikit-learn (for machine learning), NumPy and Pandas (for data manipulation), and XGBoost. These libraries provide functions and tools for building, training, and evaluating machine learning models.Here's a bit more detail on the libraries used:

sklearn (scikit-learn): This library provides tools for data preprocessing, modeling, and evaluation, including support vector machines (SVM), logistic regression, random forests, and more.

numpy: NumPy is a library for numerical and matrix operations.

pandas: Pandas is used for data manipulation and handling DataFrames.

csv: The CSV module is used for handling CSV files.

sys: The sys module is used to access command-line arguments.



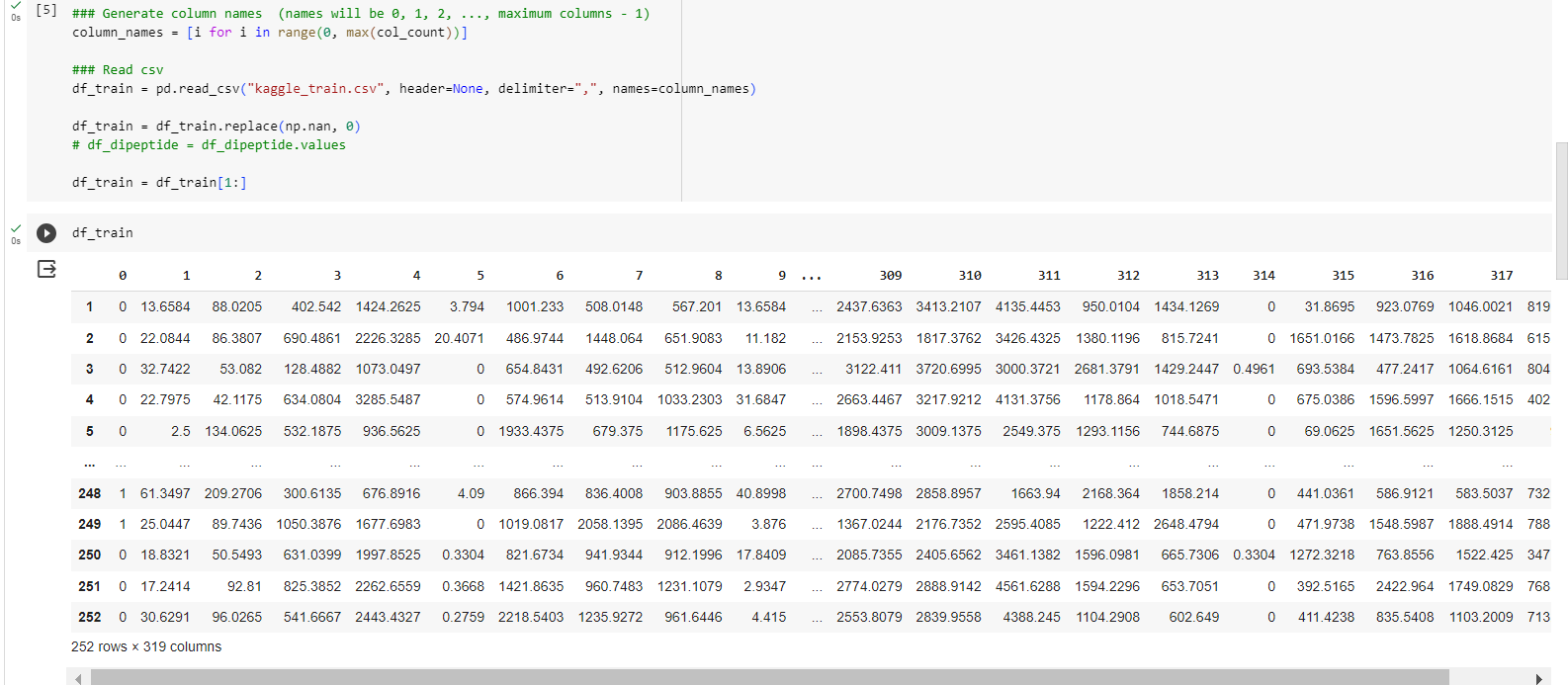
* **COMMAND-LINE ARGUMENTS: THE** script is designed to accept command-line arguments, which are read and stored in the arr list. In the provided code, these command-line arguments are not used, so they don't affect the script's behavior.
* **DATA PREPROCESSING - READING AND PREPARING DATA:**

The script reads data from a CSV file named "kaggle\_train.csv." It counts the number of columns in each line of the file to generate column names.

The data is then read into a Pandas DataFrame (df\_train), where missing values (NaN) are replaced with 0.

The first row of the DataFrame is skipped, assuming it contains column headers.

This section also performs similar data preprocessing for the test dataset. It reads the "kaggle\_test.csv" file, determines the number of columns in each line, generates column names, and stores the data in a NumPy array. Missing values are also replaced with 0. And then prints the training data

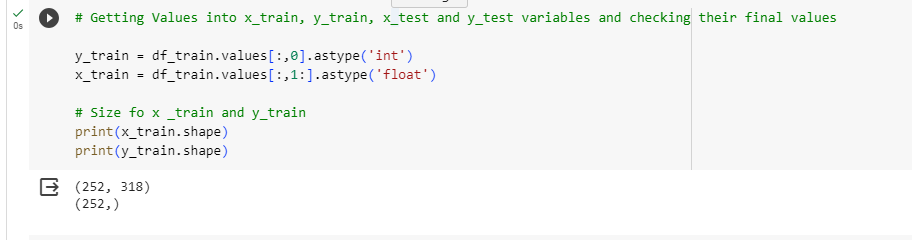


* **SEPARATING FEATURES AND LABELS:**

The target labels (y\_train) are extracted from the first column of the training data.

The remaining columns (features) are stored in the x\_train variable as a NumPy array. The shapes of x\_train and y\_train are printed to provide information about the size of the dataset.

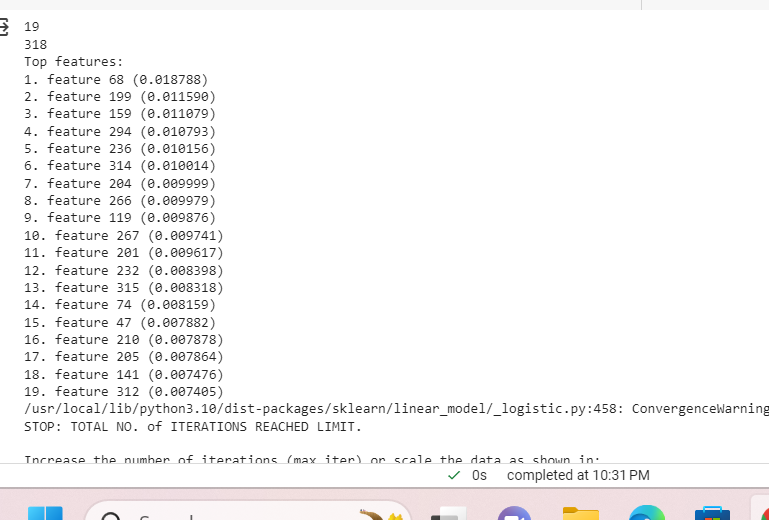
Here, the code separates the training data into features (x\_train) and labels (y\_train). The first column contains the target labels, and the rest of the columns are considered features. The labels are converted to integers, and the features are converted to floating-point numbers.



* **FEATURE SELECTION USING EXTRATREESCLASSIFIER:**

Feature selection is performed using the ExtraTreesClassifier, a tree-based ensemble model. It's executed multiple times with different random states (1, 2, 3, ..., 9, 1) to evaluate feature importance variability.

For each run, the script identifies the top 19 important features and stores them in the x\_train\_2 variable.

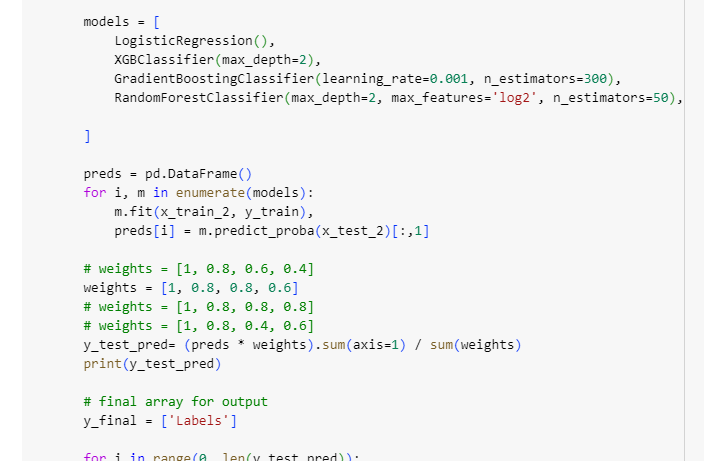


* **MODEL TRAINING AND PREDICTION:**

The script defines a list of machine learning models, including Logistic Regression, XGBoost, GradientBoostingClassifier, and RandomForestClassifier.

For each model, it fits the model to the training data (using x\_train\_2) and makes predictions on the test data (x\_test\_2).

Predictions are stored in a DataFrame called preds, where each model's predictions are in a separate column.



* **COMBINING MODEL PREDICTIONS:**

The script defines a list of weights that correspond to the different models. These weights are used to combine the model predictions.

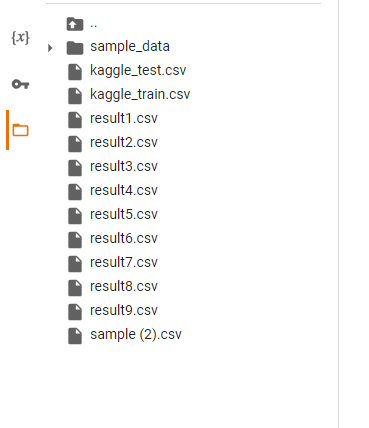
The weighted predictions are averaged to produce a final prediction for each data point. This ensemble approach aims to improve prediction accuracy by giving different models different levels of importance.

* **SAVING OUTPUT:**

The code constructs a final array called y\_final, which includes the labels and the predicted values.

It creates a DataFrame (df2) to store the final output and removes the second column from the original test data (the features).

The final output is saved as a CSV file with a name based on the random state used during feature selection (e.g., "result1.csv", "result2.csv"). Multiple output files are generated, one for each random state.



**SUMMARY** :The provided Python code offers a comprehensive solution for data preprocessing, feature selection, model training, and prediction generation. It imports a range of essential libraries, including scikit-learn, Pandas, and XGBoost, to facilitate these tasks. While it accepts command-line arguments, they aren't utilized within the script. The code reads and preprocesses training and test data from CSV files, distinguishing features from labels. Notably, it employs ExtraTreesClassifier with varying random states for feature selection. The code also includes the training of multiple machine learning models, such as Logistic Regression and XGBoost, and combines their predictions using predefined weights to yield a final prediction. Results are saved in separate CSV files, each corresponding to a different random state. This code is designed for educational purposes and can be adapted to specific use cases, offering a versatile platform for machine learning experiments.