Machine Learning Project
Glass Identification

M121 : Machine and Deep Learning

Anouar MOUHOUB
Oussama DERAOUI

Professor :
Pr. IDRI Ali

Supervised by :
Fatima Zahra NEQACH
Hafsa OUIFAK

March 13, 2023

.

Acknowledgments :

We would like to extend our most heartfelt and sincere gratitude to dear Professor Ali IDRI for his support and guidance throughout this entire semester .
It has been an honour to be under your wing for the past month dear professor and we hope this humble effort lives up to your expectations and meets your standards of excellence.
Additionally, we would like to express our appreciation to our supervisors, Hafsa OUIFAK and Fatim Zahra NAKKACH, for their valuable feedback and guidance throughout this project. Their insights and expertise have been instrumental in shaping our work and helping us reach our goals. Thank you for your dedication and support.

Contents

1 Introduction

The Glass Identification dataset contains data collected by Dr. Vina Spiehler from the American Board of Forensic Toxicologists in a comparison test of three classification algorithms used to identify types of glass. The attributes of the dataset include the refractive index and the weight percentages of several chemical elements present in the glass. The class attribute identifies the type of glass, with seven different classes: building windows that were float processed, building windows that were not float processed, vehicle windows that were float processed, vehicle windows that were not float processed, containers, tableware, and headlamps. The dataset has 214 instances with no missing attribute values, and all the attributes are continuously valued.

N.B: Float processing is a method used to produce flat glass by floating the molten glass on top of a bed of molten tin.

1.1 Objective

The objective of using machine learning on this dataset is to develop a model that can accurately classify different types of glass based on their attributes (chemical components and refractive index). This model, if correctly identified, can be used to help detect unknown types of glass found at a crime scene, that's why it interests criminologists. It can also be used to improve quality control processes in glass manufacturing. Machine learning algorithms can be trained on this dataset to learn the patterns and relationships among the different attributes of glass samples and use this knowledge to predict the class label of new instances. Furthermore, feature selection techniques can be used to identify the most important attributes that contribute to accurate classification, which can help design future experiments or develop more efficient glass manufacturing processes.

1.2 Problematic

How can we accurately classify the type of glass sample and distinguish between float-processed and non-float-processed samples based on their chemical composition? And which chemical attributes are the most important in determining the type of glass sample? Can we identify any patterns or relationships between the chemical attributes and the type of glass sample? Finally, which machine learning algorithm(s) perform(s) the best in classifying the glass samples?

1.3 Real life scenario

A burglary has taken place at a jewelry store. The thief has stolen several pieces of valuable objects, including unique handmade vases and sculptures. The police have collected several pieces of broken glass from the store, including pieces of glass that came from a hand watch and suitcase which they believe may be related to the burglary.

The model can analyze the fragments of glass collected from the crime scene and determine if they match any known types of glass, providing crucial evidence to help the police identify the thief and recover the stolen items.

The model's accuracy can make all the difference in the investigation and the case outcome.

2 Materials and Methods

2.1 Dataset Description

Number of Instances: 214

Number of Attributes: 10 (including an Id) plus the class attribute, all attributes are continuously valued.

Columns description

1. **Id number:** 1 to 214
2. **RI:** refractive index
3. **Na:** Sodium
4. **Mg:** Magnesium
5. **Al:** Aluminum
6. **Si:** Silicon
7. **K:** Potassium
8. **Ca:** Calcium
9. **Ba:** Barium
10. **Fe:** Iron
11. **Types of glass**
    (a) building windows float processed
    (b) building windows nonfloat processed
    (c) vehicle windows float processed
    (d) vehicle windows non float processed (none in this database)
    (e) containers
    (f) tableware
    (g) headlamps

(unit measurement: weight percent in corresponding oxide, as are attributes 4-10)
Summary Statistics

Attribute:	Min	Max	Mean	SD	Correlation with class
RI	1.5112	1.5339	1.5184	0.0030	-0.1642
Na	10.73	17.38	13.4079	0.8166	0.5030
Mg	0	4.49	2.6845	1.4424	-0.7447
Al	0.29	3.5	1.4449	0.4993	-0.7447
Si	69.81	75.41	72.6509	0.7745	0.1515
K	0	6.21	0.4971	0.6522	-0.0100
Ca	5.43	16.19	8.9570	1.4232	0.0007
Ba	0	3.15	0.1750	0.4972	0.5751
Fe	0	0.51	0.0570	0.0974	-0.1879

2.2 Performance Criteria

2.2.1 Accuracy score

The accuracy score is commonly denoted as Accuracy and it is calculated as follows:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

where the "Number of Correct Predictions" refers to the number of instances in the dataset that were correctly classified by the model, and the "Total Number of Predictions" refers to the total number of instances in the dataset.

The resulting value is a percentage that ranges from 0 to 100, where a higher value indicates better performance of the classification model.

2.2.2 F1 score

The F1 score is commonly denoted as F1 Score, and it is calculated as follows:

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

where Precision and Recall are calculated as:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

where True Positive, False Positive, and False Negative refer to the number of instances that were correctly predicted as positive, incorrectly predicted as positive, and incorrectly predicted as negative, respectively.

The F1 score is a metric that combines the precision and recall scores into a single value that ranges from 0 to 1, where a higher value indicates better performance of the

classification model. It is a harmonic mean of precision and recall and is a good metric to use when you want to balance the importance of both metrics.

2.2.3 Recall score

The recall score is commonly denoted as Recall or Sensitivity, and it is calculated as follows:

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

where True Positive refers to the number of instances in the dataset that were correctly classified as positive by the model, and False Negative refers to the number of instances in the dataset that were incorrectly classified as negative by the model.

The resulting value of Recall ranges from 0 to 1, where a higher value indicates better performance of the model in correctly identifying positive instances.

2.2.4 Precision score

The precision score is commonly denoted as Precision, and it is calculated as follows:

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

where True Positive refers to the number of instances in the dataset that were correctly classified as positive by the model, and False Positive refers to the number of instances in the dataset that were incorrectly classified as positive by the model.

The resulting value of Precision ranges from 0 to 1, where a higher value indicates better performance of the model in correctly identifying positive instances. The precision score is a metric that measures the proportion of true positive predictions among the total positive predictions made by the model. It is useful in cases where the cost of false positives is high, such as in medical diagnosis or credit risk assessment, where incorrectly labeling something as positive can have serious consequences.

2.3 Pre-processing Techniques

2.3.1 Data Cleaning –Missing data

Data cleaning from missing data involves identifying the missing values in a dataset and taking appropriate actions to either impute or remove them. This process is important to ensure the accuracy and completeness of the data, which can affect the quality and validity of subsequent analyses or modeling. Various techniques such as mean imputation, regression imputation, and multiple imputation can be used to handle missing

data depending on the nature and extent of missingness in the dataset.

For this part we check for any missing values using the *isna*() fucntion

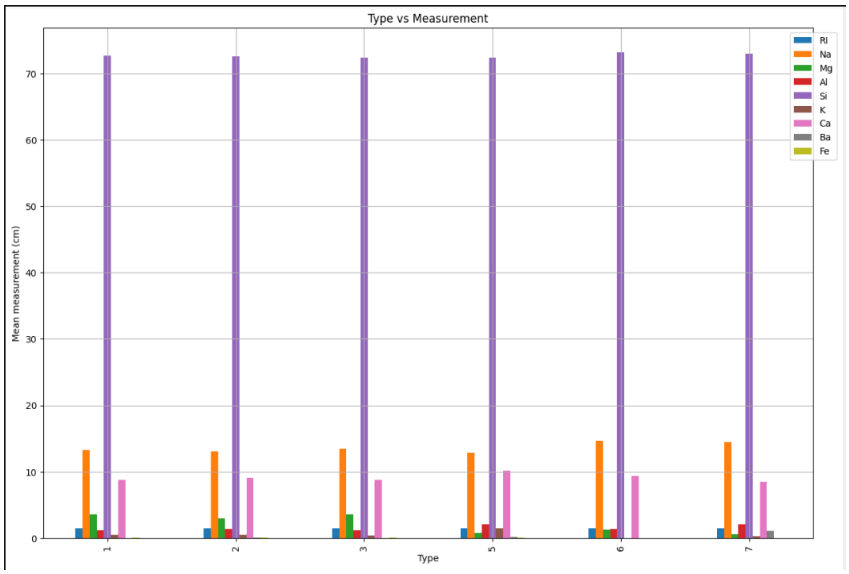
```
data.isna().any()
```

```
RI      False
Na      False
Mg      False
Al      False
Si      False
K       False
Ca      False
Ba      False
Fe      False
Type    False
dtype: bool
```

All columns returned a false value meaning no missing values exists.

2.3.2 Data Transformation -Normalization

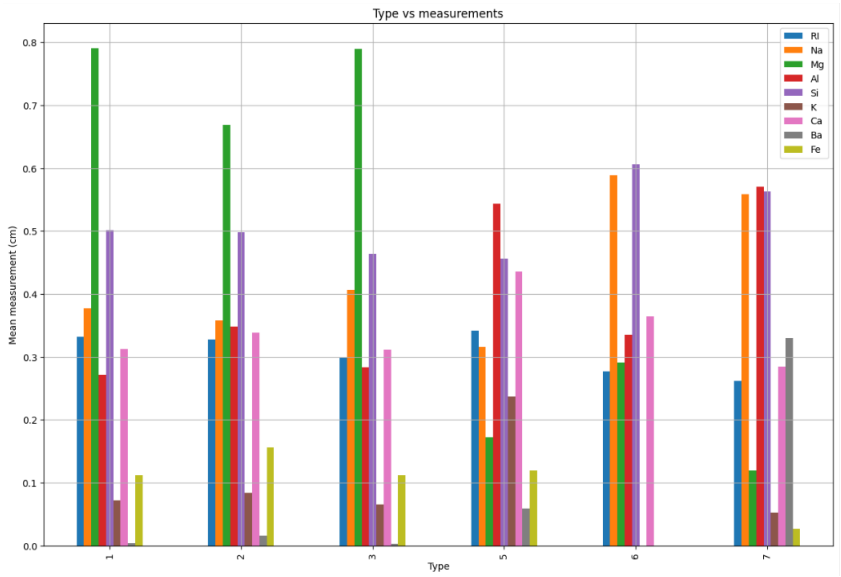
To further investigate our data we can visualize it. The best way is to plot all mean value of attributes for each class :



First, the majority of the attributes are not well represented while the mean value of other attributes is high, this indicates that we have a scale problem. Our solution to this latter is to normalize our data using a scaler. We usually use a scaler to transform numerical data into a specific range or distribution to improve the performance of machine learning algorithms that are sensitive to the scale or distribution of the input features. This can also help with data visualization and interpretation. In our case we will use a MinMaxScaler, It works by subtracting the minimum value of the data and

dividing by the range (max value - min value).

Here is our data after the normalization



Now we can proceed to work on our dataset

2.3.3 Data Balancing

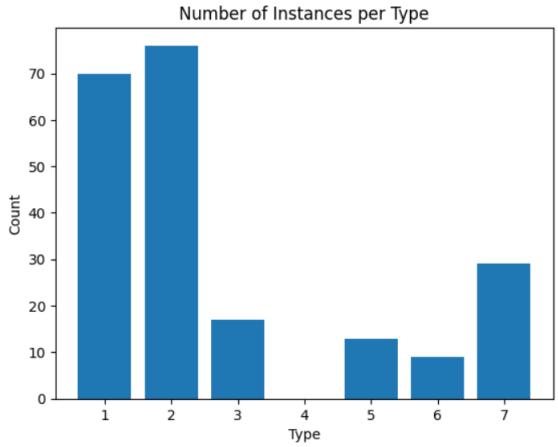
Data balancing is the process of adjusting the class distribution in a dataset to ensure that each class has roughly equal representation. Imbalanced datasets are common in many real-world applications, where one or more classes may be significantly underrepresented compared to others, which can lead to biased or inaccurate predictions by classification models. Data balancing can be achieved through various techniques such as oversampling, undersampling, or a combination of both.

1. **Oversampling:** it is a technique that involves increasing the number of minority class instances in a dataset. This can be achieved by either duplicating existing observations.

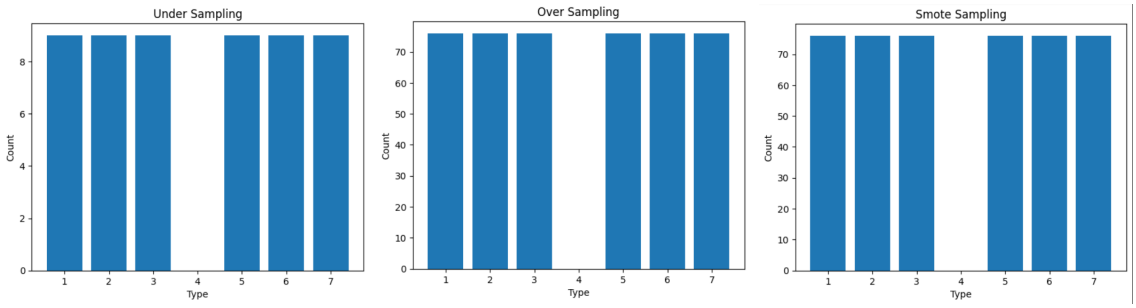
2. **Undersampling:** it is a technique that involves decreasing the number of majority class instances in a dataset. This can be done randomly to reduce class imbalance and create a more balanced dataset.

3. **SMOTE:** this technique generates synthetic minority class instances based on interpolation between neighboring instances. This approach helps to address the overfitting problem that can occur with traditional oversampling methods by creating new synthetic minority class instances

Now we visualize our target class instances to check if our data is balanced or not



it is apparent that the class instances are indeed not balanced. We can apply now the 3 techniques cited earlier to solve this problem(undersampling,oversampling and SMOTE)



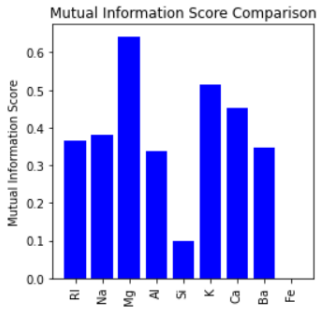
After using these 3 techniques all class instances are balanced.

2.3.4 Data Reduction

Data reduction is the process of reducing the amount of data by selecting, transforming, or summarizing it in some way that it is more manageable and easier to analyze.There are various techniques used for data reduction, but in this paper, we will focus only on the feature selection techniques.
Feature selection is an important step in machine learning that involves selecting a subset of the most relevant features (or variables) from a larger set of features to improve the accuracy and efficiency of the learning algorithm. There are several classes of feature selection methods, including:

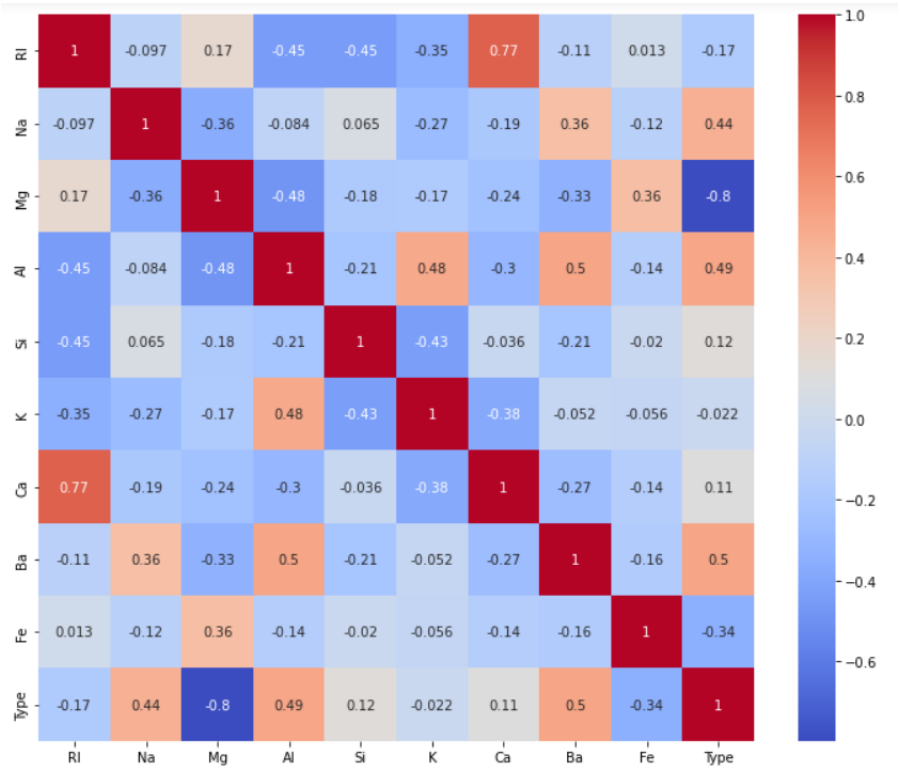
1. **Filter methods:** These methods use statistical measures to evaluate the relevance of each feature independently of the learning algorithm. In this paper we used 3 filter methods:

(a) **Mutual information gain:** Mutual information gain measures the dependence between two variables. In feature selection, it identifies the most relevant features for predicting a target variable by measuring how much knowing one variable reduces uncertainty about the other. Highly relevant features are selected for inclusion in a predictive model.



In our paper, it is remarkable that Mg is the most important followed by K and Ca

(b) **Correlation Matrix:** A correlation matrix shows pairwise correlations between variables in a dataset. In feature selection, it can identify highly correlated features, which can be removed to improve model performance. Highly correlated features with the target may be selected for inclusion in a predictive model.



In our paper, we can see high correlations with Ba, Al and Na

(c) **Select K-Best:** K-Best is a feature selection method that selects K top-scoring features based on statistical tests such as ANOVA or chi-squared. It ranks features by their scores and selects the K best ones for inclusion in a predictive model.

2. **Wrapper methods:** These methods use the learning algorithm itself to evaluate the relevance of each feature by repeatedly training and testing the algorithm on different subsets of features. Examples include recursive feature elimination and forward selection.

(a) **Forward feature selection:** Forward feature selection with a random forest classifier is a method that iteratively adds the most important features to a predictive model based on their importance scores from a random forest classifier. It starts with an empty feature set, adds features in each iteration, and selects the best performing model.

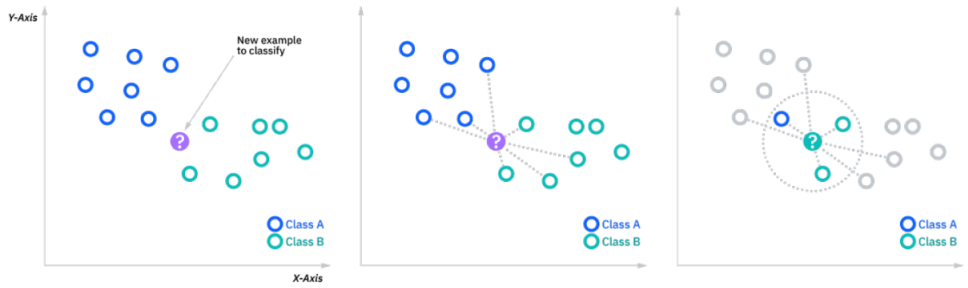
(b) **Exhaustive feature selection:** Exhaustive feature selection is a feature selection technique that evaluates all possible combinations of features to find the subset that optimizes a performance metric. It can be computationally expensive, but it guarantees finding the best performing subset of features for a given model.

There are other methods such as embedded and hybrid but they won't be our focus in this paper.

2.4 Classification Techniques

2.4.1 K-Nearest Neighbour (KNN)

KNN (k-nearest neighbors) is a classification algorithm that works by identifying the k nearest neighbors to a new instance in a training dataset. The distance between the new instance and each data point in the dataset is calculated using a given distance metric. The k nearest neighbors are then selected based on their proximity to the new instance. The new instance is then classified based on the majority class among those k nearest neighbors.



This simple yet effective algorithm is often used in applications such as image recognition, recommender systems, and anomaly detection. The choice of k is important in KNN, as a larger k value will result in a smoother decision boundary, while a smaller k value will result in a more complex decision boundary.

KNN can be used for both binary and multi-class classification problems and is relatively simple to implement. However, its performance may be sensitive to the choice of K and the distance metric used, and it can be computationally expensive for large datasets or high-dimensional feature spaces.

Here's an example of how some of distance metrics commonly used in K-Nearest Neighbors (KNN) :
Euclidean distance:

$$d_E(A, B) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

Manhattan distance:

$$d_M(A, B) = \sum_{i=1}^{n} |a_i - b_i|$$

where A and B are two n-dimensional points, and a_i and b_i are the i^{th} components of A and B, respectively.

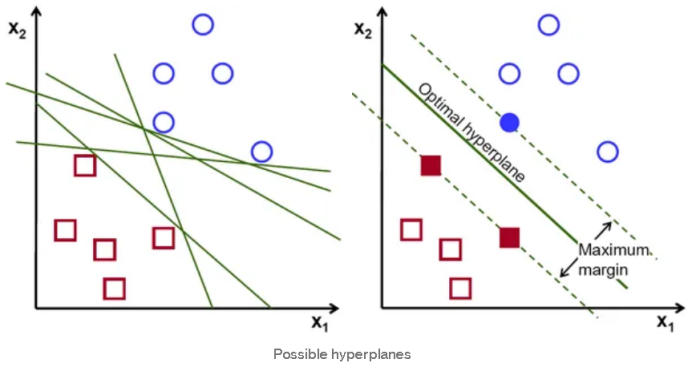
2.4.2 Support Vector Machines (SVM)

Support Vector Machines (SVMs) are a type of supervised machine learning algorithm that can be used for classification or regression tasks.

The main idea behind SVMs is to find a hyperplane in a high-dimensional space that can best separate the data into different classes. For example, in a binary classification problem, the hyperplane is used to separate data points into two classes.

To explain SVMs in a simple way, let's consider a 2D space where we have two classes of data points that need to be separated. A linear SVM aims to find a line that can separate the two classes with maximum margin, where the margin is the distance between the line and the closest data points from each class. The data points that

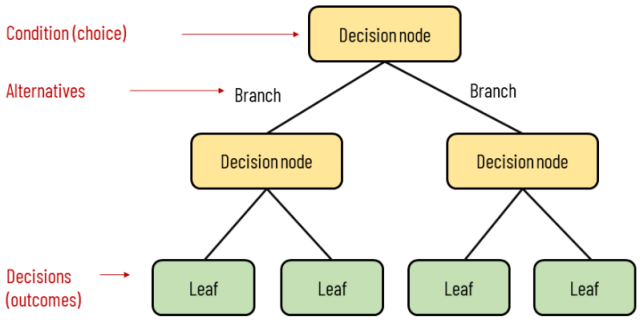
are closest to the line are called support vectors, and they define the location and orientation of the line.



2.4.3 Decision tree

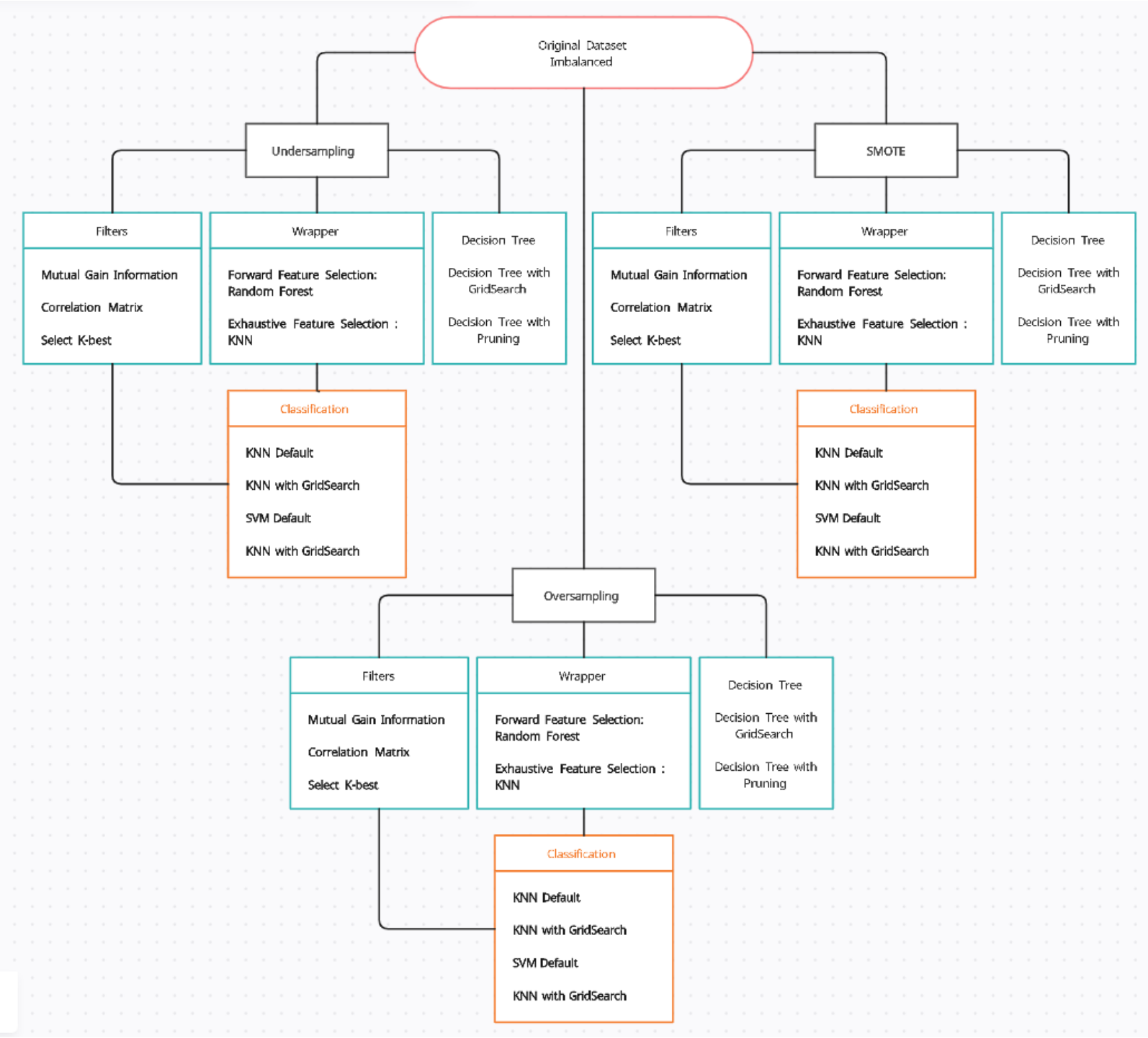
A decision tree is a supervised machine learning algorithm used for classification and regression analysis. It partitions data into subsets based on input features, recursively applying this process until a stopping criterion is met. This creates a tree structure where internal nodes represent feature tests and leaf nodes represent output. The algorithm selects the most informative feature to split the data into subsets, using statistical measures such as information gain, Gini index, or entropy. Stopping criteria include maximum depth, minimum samples required to split an internal node, or form a leaf node. Overfitting and underfitting can occur if stopped too late or too early.

Once the decision tree is built, it can predict new data points by traversing from root to leaf node based on feature tests at each internal node. The output corresponds to the leaf node.



Decision trees have several advantages such as interpretability, fast training, and the ability to handle both categorical and numerical data. They are also robust to noisy data and outliers. However, decision trees can suffer from overfitting, especially when the tree is too complex or when there are many irrelevant or redundant features in the data.

3 Experimental Design



4 Results and Discussion

.

In this chapter, we present the results of our analysis on the impact of different data balancing techniques, namely undersampling, oversampling, and SMOTE, in combination with various filter and wrapper feature selection methods, on the performance of several machine learning models. We compared the performance of multiple models, including K-nearest neighbors (KNN) with and without grid search, support vector machine (SVM) with and without grid search, and decision trees, across each combination of data balancing technique, filter, wrapper, and feature selection method.

For each data balancing technique, we used three different filter methods: mutual information gain, correlation matrix, and selecting k-best features. We also used two wrapper feature selection methods: random forest and k-nearest neighbors (KNN). We evaluated the performance of each combination using several metrics, including accuracy, precision, recall, and F1-score.

We implemented a function that takes a dataset as a parameter and generates all metrics, we used it for eveqry combination of the techniques mentioned earlier, here is the general form of the resulting data frame

		Accuracy	F1_score	Precision	Recall
us_ig	Knn	0.363636	0.363636	0.363636	0.363636
	Knn_gs	0.554545	0.508889	0.554545	0.566667
	Svm	0.363636	0.333333	0.363636	0.555556
	Svm_gs	0.627778	0.588889	0.627778	0.633333
us_corr	Knn	0.352941	0.352941	0.352941	0.352941
...	...	...	...	...	...
smt_knn	Svm	0.766423	0.771290	0.766423	0.795794
	Svm_gs	0.893304	0.884711	0.893304	0.885168
us	Decision Tree	0.352941	0.352941	0.352941	0.352941
os	Decision Tree	0.897810	0.897810	0.897810	0.897810
smt	Decision Tree	0.839416	0.839416	0.839416	0.839416

63 rows × 4 columns

4.1 Undersampling

		Accuracy	F1_score	Precision	Recall
us_ig	**Knn**	0.363636	0.363636	0.363636	0.363636
	Knn_gs	0.554545	0.508889	0.554545	0.566667
	Svm	0.363636	0.333333	0.363636	0.555556
	Svm_gs	0.627778	0.588889	0.627778	0.633333
us_corr	**Knn**	0.352941	0.352941	0.352941	0.352941
	Knn_gs	0.609091	0.615556	0.609091	0.650000
	Svm	0.470588	0.460606	0.470588	0.611111
	Svm_gs	0.732143	0.648667	0.732143	0.713333
us_skb	**Knn**	0.352941	0.352941	0.352941	0.352941
	Knn_gs	0.609091	0.615556	0.609091	0.650000
	Svm	0.470588	0.460606	0.470588	0.611111
	Svm_gs	0.732143	0.648667	0.732143	0.713333
us_rf	**Knn**	0.470588	0.470588	0.470588	0.470588
	Knn_gs	0.741818	0.701111	0.741818	0.733333
	Svm	0.411765	0.388384	0.411765	0.555556
	Svm_gs	0.703571	0.597556	0.703571	0.656667
us_knn	**Knn**	0.352941	0.352941	0.352941	0.352941
	Knn_gs	0.627273	0.576984	0.627273	0.633333
	Svm	0.411765	0.388384	0.411765	0.555556
	Svm_gs	0.764286	0.739778	0.764286	0.763333
us	**Decision Tree**	0.352941	0.352941	0.352941	0.352941

. For this undersampling dataset it is apparent that the majority of combinations are underperforming since all the metrics show results that vary between 0.3 and 0.6 and with the best combination being (us_knn, Svm_gs),$(0.76, 0.73, 0.76, 0.76)$

4.2 Oversampling

os_ig	**Knn**	0.795620	0.795620	0.795620	0.795620
	Knn_gs	0.894744	0.888779	0.894744	0.894861
	Svm	0.795620	0.793734	0.795620	0.816905
	Svm_gs	0.924851	0.917435	0.924851	0.918199
os_corr	**Knn**	0.708029	0.708029	0.708029	0.708029
	Knn_gs	0.855184	0.841844	0.855184	0.855556
	Svm	0.598540	0.572822	0.598540	0.657619
	Svm_gs	0.883978	0.873386	0.883978	0.873620
os_skb	**Knn**	0.708029	0.708029	0.708029	0.708029
	Knn_gs	0.855184	0.841844	0.855184	0.855556
	Svm	0.598540	0.572822	0.598540	0.657619
	Svm_gs	0.883978	0.873386	0.883978	0.873620
os_rf	**Knn**	0.861314	0.861314	0.861314	0.861314
	Knn_gs	0.899068	0.896103	0.899068	0.899722
	Svm	0.664234	0.656434	0.664234	0.718364
	Svm_gs	0.918452	0.910295	0.918452	0.911347
os_knn	**Knn**	0.854015	0.854015	0.854015	0.854015
	Knn_gs	0.899068	0.895168	0.899068	0.899444
	Svm	0.700730	0.697166	0.700730	0.755293
	Svm_gs	0.908978	0.899693	0.908978	0.900303
os	**Decision Tree**	0.897810	0.897810	0.897810	0.897810

. Compared with the undersampling dataset, we can clearly see that the metrics results has increased overall with the best model being SVM with GridSearch and with the best combination being (os_ig, Svm_gs),its results are $(0.924, 0.917, 0.924, 0.918)$. It is also important to note that default SVM is underperforming in this dataset.

4.3 SMOTE

smt_ig	**Knn**	0.832117	0.832117	0.832117	0.832117
	Knn_gs	0.833493	0.823879	0.833493	0.834306
	Svm	0.686131	0.667335	0.686131	0.730556
	Svm_gs	0.868254	0.859756	0.868254	0.859973
smt_corr	**Knn**	0.773723	0.773723	0.773723	0.773723
	Knn_gs	0.813545	0.803531	0.813545	0.814306
	Svm	0.627737	0.616994	0.627737	0.671984
	Svm_gs	0.852381	0.843882	0.852381	0.842138
smt_skb	**Knn**	0.773723	0.773723	0.773723	0.773723
	Knn_gs	0.813545	0.803531	0.813545	0.814306
	Svm	0.627737	0.616994	0.627737	0.671984
	Svm_gs	0.852381	0.843882	0.852381	0.842138
smt_rf	**Knn**	0.824818	0.824818	0.824818	0.824818
	Knn_gs	0.844290	0.836954	0.844290	0.845278
	Svm	0.744526	0.740931	0.744526	0.774800
	Svm_gs	0.862004	0.853129	0.862004	0.854934
smt_knn	**Knn**	0.824818	0.824818	0.824818	0.824818
	Knn_gs	0.875012	0.864789	0.875012	0.875139
	Svm	0.766423	0.771290	0.766423	0.795794
	Svm_gs	0.893304	0.884711	0.893304	0.885168
smt	**Decision Tree**	0.839416	0.839416	0.839416	0.839416

. While oversampling may achieve higher classification accuracy than SMOTE and undersampling methods in some cases, SMOTE has been shown to be a more effective technique for improving the classification performance without overfitting, making it a popular choice for imbalanced datasets

5 Conclusion and Perspectives

. The use of data balancing techniques such as undersampling, oversampling, and SMOTE is critical when working with imbalanced datasets. These techniques help to address the issue of class imbalance and improve the performance of machine learning models. In this project, the oversampling technique performed the best, which is not surprising since oversampling tends to generate more data for the minority class, thereby improving the model's ability to learn from it.

Feature selection is another crucial step in building machine learning models. The use of filters such as mutual information gain, correlation matrix, and k-best, as well as wrappers such as random forest and KNN, helps to select the most relevant features for the model. The mutual information gain filter seemed to be the most effective in this project, as it was used in the best-performing combination.

In terms of the models used, both SVM and KNN are popular choices for classification tasks. The use of grid search is also a common practice when working with these models, as it helps to find the optimal hyperparameters for the model. The fact that the best-performing combination used SVM with grid search is a testament to the effectiveness of this approach.

Overall, this project provides valuable insights into the use of data balancing techniques and feature selection filters in building machine learning models. The combination of oversampled mutual information gain and SVM with grid search seems to be a promising approach that can be further explored in future work.

5.1 Real life scenario outcome

. In conclusion, the machine learning model developed to analyze glass fragments collected from the crime scene played a crucial role in identifying the thief and recovering the stolen items in the burglary scenario. The police were able to collect several pieces of broken glass from the store, including pieces of glass that came from a hand watch and suitcase which they believed may be related to the burglary. The model accurately identified the types of glass fragments found at the crime scene, including the one from the thief's handwatch, which provided valuable evidence that helped the police narrow down their search and ultimately catch the thief. The model's accuracy in matching glass fragments to known types of glass was key to its successful deployment in this investigation. Overall, this project demonstrates the potential of machine learning in forensic investigations, and highlights the importance of accurate and reliable models in solving complex crimes.

6 References

https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

https://www.ibm.com/topics/knn

https://scikit-learn.org/stable/modules/classes.html

https://scikit-learn.org/stable/modules/generated/
sklearn.feature_selection.mutual_info_classif.htmlr50b872b699c4-2

https://scikit-learn.org/stable/modules/generated/
sklearn.svm.SVC.htmlsklearn.svm.SVC.score

https://scikit-learn.org/stable/modules/generated/
sklearn.metrics.f1$_score.htmlsklearn.metrics.f1_score$

7 Appendix

You can find the Python code for our Glass Identification project in the following GitHub link to our notebook.
Click here to access the GitHub repository

7.1 The function that generates all metrics

```
def metric_func(X,Y,X_train,Y_train,X_test,Y_test):

    ######## knn with no grid search ########

    clf =  KNeighborsClassifier(n_neighbors=6)
    clf.fit(X_train, Y_train)
    y_pred= clf.predict(X_test)
    list_knn=[metrics.accuracy_score(Y_test, y_pred),
              metrics.f1_score(Y_test, y_pred,average='micro'),
              metrics.precision_score(Y_test, y_pred,average='micro'),
              metrics.recall_score(Y_test, y_pred,average='micro')
             ]

    ############ knn with gridsearch #########

    Knn2 = KNeighborsClassifier()
    grid_param={'n_neighbors': range(1,31),
     'weights' : ['uniform', 'distance'],
     'metric' : ['euclidean', 'manhattan', 'minkowski']}
    grid = GridSearchCV(Knn2, grid_param, cv = 5, scoring = ['accuracy
    ', 'f1_macro','recall_macro','precision_micro'], refit='accuracy')
    grid.fit(X,Y)

    list_knn_gs=[grid.cv_results_['mean_test_accuracy'][grid.
    best_index_],
                 grid.cv_results_['mean_test_f1_macro'][grid.
    best_index_],
                 grid.cv_results_['mean_test_precision_micro'][grid.
    best_index_],
                 grid.cv_results_['mean_test_recall_macro'][grid.
    best_index_]]

   ############ svm with no gridsearch ########

    svm = SVC()
    svm.fit(X_train, Y_train)
    Y_pred= svm.predict(X_test)
```

```
 list_svm=[svm.score(X_test, Y_test),
           metrics.f1_score(Y_test, Y_pred, average='macro'),
           precision_score(Y_test, Y_pred, average='micro'),
           recall_score(Y_test,  Y_pred, average='macro')]

############ svm with gridsearch #########

 parametres = {"kernel":['linear','poly','rbf'],"C"
:[0.1,0.5,1.0,2.0]}
 #classifieur    utiliser
 svmc = SVC()
 #instanciation de la recherche
 grille = GridSearchCV(Knn2, grid_param, cv = 5, scoring = ['
accuracy', 'f1_macro','recall_macro','precision_micro'], refit='
accuracy')
 #lancer l'exploration
 resultats = grille.fit(X_train,Y_train)
 list_svm_gs=[resultats.cv_results_['mean_test_accuracy'][grille.
best_index_],
              resultats.cv_results_['mean_test_f1_macro'][grille.
best_index_],
              resultats.cv_results_['mean_test_precision_micro'][
resultats.best_index_],
              resultats.cv_results_['mean_test_recall_macro'][
resultats.best_index_]]

 ############ extracting the method name from the variable name
#########
 import inspect

 var_name = None
 calling_frame = inspect.currentframe().f_back
 for name, value in calling_frame.f_locals.items():
     if value is X:
         var_name = name
         break
 method_name= var_name[2:]
 ############ creating DataFrame out of outputs #########

 list_metrics=["Accuracy","F1_score","Precision","Recall"]
 list_methods=["Knn","Knn_gs","Svm","Svm_gs"]
 df = pd.DataFrame([list_knn,list_knn_gs,list_svm,list_svm_gs],
index=list_methods , columns=list_metrics)
 new_index = pd.Index([method_name,method_name,method_name,
method_name])
```

```
    new_multiindex = pd.MultiIndex.from_arrays([new_index,df.index]) # to add another index on the existing index
    df.index = new_multiindex
    return df
```

7.2 Function that generates all metrics for Decision Trees

```
def metric_func_dt(X_train,Y_train,X_test,Y_test):

       ############ Decision Tree metrics ########

    clf = tree.DecisionTreeClassifier()
    clf.fit(X_train, Y_train)
    y_pred= clf.predict(X_test)
    list_metric = [metrics.accuracy_score(Y_test, y_pred),
            metrics.f1_score(Y_test, y_pred,average='micro'),
            metrics.precision_score(Y_test, y_pred,average='micro'),
            metrics.recall_score(Y_test, y_pred,average='micro')]
    ############ extracting the method name from the variable name #########

    import inspect

    var_name = None
    calling_frame = inspect.currentframe().f_back
    for name, value in calling_frame.f_locals.items():
        if value is X_train:
            var_name = name
            break
    if len(var_name)==11:
        method_name= var_name[2:5]
    else:
        method_name= var_name[2:4]

    ############ creating DataFrame out of outputs #########

    list_metrics=["Accuracy","F1_score","Precision","Recall"]
    list_methods=["Decision Tree"]
    df = pd.DataFrame([list_metric], index=list_methods , columns= list_metrics)
    new_index = pd.Index([method_name])
    new_multiindex = pd.MultiIndex.from_arrays([new_index,df.index]) # to add another index on the existing index
    df.index = new_multiindex
    return df
```

7.3 Dataframe with all the metrics

```
#generating all the 15 dataframes from different method with aggregating data

df1= metric_func(X_us_ig, y_us_ig, X_us_ig_train, y_us_ig_train, X_us_ig_test, y_us_ig_test)

df2= metric_func(X_us_corr, y_us_corr, X_us_corr_train, y_us_corr_train, X_us_corr_test, y_us_corr_test)

df3= metric_func(X_us_skb, y_us_skb, X_us_skb_train, y_us_skb_train, X_us_skb_test, y_us_skb_test)

df4= metric_func(X_us_rf, y_us_rf, X_us_rf_train, y_us_rf_train, X_us_rf_test, y_us_rf_test)

df5= metric_func(X_us_knn, y_us_knn, X_us_knn_train, y_us_knn_train, X_us_knn_test, y_us_knn_test)


df6= metric_func(X_os_ig, y_os_ig, X_os_ig_train, y_os_ig_train, X_os_ig_test, y_os_ig_test)

df7= metric_func(X_os_corr, y_os_corr, X_os_corr_train, y_os_corr_train, X_os_corr_test, y_os_corr_test)

df8= metric_func(X_os_skb, y_os_skb, X_os_skb_train, y_os_skb_train, X_os_skb_test, y_os_skb_test)

df9= metric_func(X_os_rf, y_os_rf, X_os_rf_train, y_os_rf_train, X_os_rf_test, y_os_rf_test)

df10= metric_func(X_os_knn, y_os_knn, X_os_knn_train, y_os_knn_train, X_os_knn_test, y_os_knn_test)


df11= metric_func(X_smt_ig, y_smt_ig, X_smt_ig_train, y_smt_ig_train, X_smt_ig_test, y_smt_ig_test)

df12= metric_func(X_smt_corr, y_smt_corr, X_smt_corr_train, y_smt_corr_train, X_smt_corr_test, y_smt_corr_test)

df13= metric_func(X_smt_skb, y_smt_skb, X_smt_skb_train, y_smt_skb_train, X_smt_skb_test, y_smt_skb_test)

```

```
df14= metric_func(X_smt_rf, y_smt_rf, X_smt_rf_train, y_smt_rf_train,
    X_smt_rf_test, y_smt_rf_test)

df15= metric_func(X_smt_knn, y_smt_knn, X_smt_knn_train,
    y_smt_knn_train, X_smt_knn_test, y_smt_knn_test)


df16= metric_func_dt(X_us_train,y_us_train,X_us_test,y_us_test)

df17= metric_func_dt(X_os_train,y_os_train,X_os_test,y_os_test)

df18= metric_func_dt(X_smt_train,y_smt_train,X_smt_test,y_smt_test)



    #concaatenating all the dfs to create final df of all the project

final_df = pd.concat([df1, df2 , df3 ,
                      df4, df5 , df6 ,
                      df7, df8 , df9 ,
                      df10,df11,df12 ,
                      df13,df14,df15 ,
                      df16,df17,df18])
```