BI Package

IDF | ATOM

dmbi FOR BEGINNERS

sHAKED lUBIN

‏2018

Table of contents

[**BI.Manipulation** 2](#_Toc514066928)

[**Replace\_columns** 2](#_Toc514066929)

[Groups 2](#_Toc514066930)

[Normalization 3](#_Toc514066931)

[WTF 4](#_Toc514066932)

[**Feature\_Selection** 5](#_Toc514066933)

[Select 5](#_Toc514066934)

[**BI.Evaluation** 6](#_Toc514066935)

[**Model\_Evaluation** 6](#_Toc514066936)

[Algorithms 6](#_Toc514066937)

[Precision\_Recall 7](#_Toc514066938)

[Threshold\_Selection 8](#_Toc514066939)

[Algorithm\_Selection 10](#_Toc514066940)

[DM 13](#_Toc514066941)

[STD 14](#_Toc514066942)

[**BI.Analyza** 15](#_Toc514066943)

[**Data\_Analyze** 15](#_Toc514066944)

[Decision\_trees 15](#_Toc514066945)

[Graphs 16](#_Toc514066946)

[Group\_Comparison 17](#_Toc514066947)

# **BI.Manipulation**

## **Replace\_columns**

The ***Replace\_columns*** class holds functions that help us manipulate our data. Sometimes we do not want to use our original data but rather a grouped version of our data, the reason behind this is that if we have a column with grades that go from 1 to 100 it might "confuse" our predictive algorithms instead of helping it. There are a few ways to solve this problem with the package and they will be specified later.

### Groups

This function groups columns by size and target rate proximity. The function gets a list of columns to group, for each column it runs on each unique value by a chronological order and checks the following – If the amount of people that have this value is too small (we can decide what we consider too small but the default is 30 people) or if the target rate for this value is close to the last value/group we checked. If one of those statements is true we combine the current value with the previous value/group we checked, if neither statement is true we consider the value as a new group (for the first value it will always create a new group).

Note that this function can only work on columns that have only integers or floats and for floats it only keeps one number after the dot.

**Parameters**

**dataframe:** *pandas dataframe, default empty dataframe*The main dataframe we work on

**target:** *str, default ''.*The name of our target column

**columns\_for\_groups:** *list, default []*A list of all the columns names we want to group (we group   
each column separately)

**target\_rate\_min\_diff:** *float, default 0.05*The minimum difference in target rate from the previous value/group we except so we will combine the current value with the previous value/group.

**min\_target\_shape:** *int, default 30*The minimum number of people that can be in a group, if the current value has less people than this number it will be combined with the previous value/group.

**replace\_columns:** *bool, default True*If left as true the new grouped column will replace the original column, otherwise it will save the original column as 'column name \_original'

**Returns**

A tuple with the following items:

1. **Updated** *dataframe*
2. **Groups decoding** *dataframe* – ***[****column**name, group**name, group\_temp(the smallest value), group\_final(an int representing the group so it will be an ascending series)****]***

### Normalization

This function normalize our data. In the function we can choose between three different ways of normalization: STD, 0-1 and Special.

**STD** – Standard deviation. This is a mathematical way to combine close values by their deviation from the average value in the column.   
Note that this method usually creates a small amount of groups (between 2 to 4) and we need to decide how the groups are divided. For more information I suggest reading online about standard deviation.

**0-1** – This method of normalizing data changes all our data to a range of 0-1. This is achieved by taking each value and divide it by the maximum value in the column. This is not the best way to normalize data but if we need all our columns to be on the same range it is the easiest way to do it.

**Special** – This is a unique way of normalizing data that works differently from the other two options. In the other normalizing options we manipulate the way the values are given but we try to keep their features as close to what we had before. In this method we give a wanted average and a wanted STD for the column and by a mathematical manipulation we change the data to answer those conditions.

**dataframe:** *pandas dataframe, default empty dataframe*The main dataframe we work on

**col\_for\_method:** *dict, default {}*The dictionary has each method we want to use as a key and its value is a list of all the columns we want to run this method on. For example:   
*{Method1: [col1, col2, col3], Method2: [col8]}*

**replace\_columns:** *bool, default True*If left as true the new grouped column will replace the original column, otherwise it will save the original column as 'column name \_original'

If we want to use the 'STD' method we also need the following parameters:

**target:** *str, default ''*The name of our target column

**key:** *str, default ''*The name of our main key column, usually called 'PN'

**std\_range:** *list, default [float ('-inf'), 0, 1, float ('inf')]*This list determines the STD groups – A group is what goes between two values in the list so in this case: >0, 0-1, >1

If we want to use the 'Special' method we also need the following parameters:

**wanted\_average:** *int/float, default 0*can be any number

**wanted\_std:** *int/float, default 0*can be any number

**Parameters**

**Updated** *dataframe*

**Returns**

### WTF

This function is built to handle multiple column features like the WTF's.  
  
WTF's (The soldier's medical defects) are useful features that can tell us a lot about the soldier, the problem is that the way we extract this information from the BIMS gives us the data in an inconvenient way – we get a column for the first defect, a column for the second defect and so on till the tenth defect (a soldier can only have 10 medical defects). In current format of the data two different soldiers can have the same defect but in different columns, for example: Soldier A has only asthma so it's written in column 1 and the other columns are blank. Soldier B has asthma and also short eyesight, so for him in column 1 he has the eye problem in the asthma is in column 2.

The WTF function first creates a pandas dataframe that have a key (PN) column and a WTF column so a soldier is duplicated for each WTF it has. In addition the function can calculate what are the most influencing WTFs in our data and add Boolean columns to indicate if a soldier has this specific WTF or not.

*\*****Note*** *– This function can be used for every group of columns that have the same prefix.*

**Parameters**

**dataframe:** *pandas dataframe, default empty dataframe*The main dataframe we work on

**key:** *str, default ''*The name of our main key column, usually called 'PN'

**wtf\_prefix:** *str, default 'WTF'*A string that indicates if it is a WTF column. We need to have all our WTF columns in our dataframe have this string in their name.

**add\_boolwtf:** *bool, default False*If True it will add Boolean columns for the most influencing WTFs

**outputpath:** *str, default ''*If left '' than no file will be created, otherwise it will save our key + WTF dataframe.   
\*Needs to be a full path (not only a folder).

If 'add\_boolwtf' == True:

**min\_wtf\_target\_size:** *int/float, default 1.0*The minimum number of people we except for a WTF to be considered as influencing. If *int* it will take it as number of people, if *float* it will take it as a percent from our population. *\*Note that 5% should be 5.0 and not 0.05(!)*

**influencing\_wtfs:** *int, default 3*Number of influencing WTFs we want to add as Boolean columns.

**Returns**

A tuple with the following items:

1. **Updated** *dataframe*
2. **WTF** *dataframe* – ***[****key, WTF****]***

## **Feature\_Selection**

The ***Feature\_Selection*** class holds functions that help us know which features in our dataframe are the most influencing for our prediction. We want to know which features are more important because having a lot of data does not always mean we get better results, sometimes having too many "weak" features harm our predictions and we would want to use only our most influencing features.   
Knowing what are the most important features can also help us know where to invest our time and analyzes.

### Select

This function finds the most influencing features on our target. It has 3 different methods to do it: *SelectKBest*, *ForestClassifier* and *Entropy*. Both SelectKBest and ForestClassifier are built in functions from sklearn, for more information I recommend reading online.  
Entropy is a mathematical formula that calculates the "weight" of each value and sums it up to get the "weight" of the feature, the feature that has less "weight" is considered more influencing.  
Entropy formula for each unique value in the feature column:   
***nc*** = nesher count  
***nnc*** = not nesher count

**Returns**

**Dictionary** - *A dictionary that the keys are the selection method and the value is a descending list of all the features (from the most influencing to the least influencing feature).*

**Parameters**

**train\_df:** *pandas dataframe, default empty dataframe*The dataframe that we want to learn from

**target:** *str, default ''*The name of our target column

**feature\_selection\_methods:** *list, default ['SelectKBest']*A list of the methods we want to use

**relevant\_features:** *list, default []*A list of the features we want to choose from

**outputpath:** *str, default ''*If left '' than no file will be created, otherwise it would save a file with the name of the feature and a column for each method with its rank.   
\*Needs to be a full path (not only a folder).

If SelectKBest is in feature\_selection\_methods:

**amount\_of\_features:** *int, default 12*SelectKBest requires an amount of features to rank, if you have more features than the value of this parameter than the "weakest" features would not get ranked.

# **BI.Evaluation**

## **Model\_Evaluation**

The ***Model\_Evaluation*** class holds functions that let us predict and evaluate our results. Although predicting is not evaluating it is the closest to this class so it is in here.

### Algorithms

This function runs the most used algorithms in our unit. It can run: *LinearDiscriminantAnalysis* (LDA), *GaussianNB* and *RandomForestClassifier*. For RandomForestClassifier we also run it with 'Hyper parameter' method to try enhance its results. Note that the 'Hyper parameter' method takes more time.  
Those three algorithms are built in the sklearn package, for more information I recommend reading online.

**Returns**

**Updated** *dataframe*

**Parameters**

**train\_df:** *pandas dataframe, default empty dataframe*The dataframe that we want to learn from

**pred\_df:** *pandas dataframe, default empty dataframe*The dataframe that we want to predict on

**target:** *str, default ''*The name of our target column

**features:** *list, default []*A list of the features we want to learn from and use for our prediction

**algorithms:** *list, default* *['ld', 'Ga', 'Rf', 'Rf\_Grid']*   
List of the algorithms we want to run.   
\*'Rf\_Grid' is the hyper parameter version of 'Rf'

**outputpath:** *str, default ''*If left '' than no file will be created, otherwise it would save a file with the original dataframe with the addition of our prediction columns  
\*Does **not** need to be a full path (only a folder).

### Precision\_Recall

Each algorithm has a lot of different values for its prediction and it's not only 1/0. When we check the quality of the results we need a way to determine if the results are close to the reality. One way to check is by *precision* and *recall*. Precision is the percentage of people that where actually 1 from the people we predicted that they are 1. Recall is the percentage of people that we said they are 1 and that is true from everyone that in reality is 1. For example:

*Precision* -> 25/ (25+21)   
*Recall* -> 25/ (25+45)

|  |  |  |
| --- | --- | --- |
|  | Real 1 | Real 0 |
| Predicted 1 | 25 | 21 |
| Predicted 0 | 45 | 9 |

We can choose between three methods to calculate our precision/recall: *'PRC'*, *'Quantitative'*, and *'Percentage'*.

*'**Quantitative'* - We take our data and cut it by a certain amount, and then we cut our data by double that amount, and then by triple the amount and so on until we get to the full size of our dataframe. Each time we cut a different size, the code remember it as a new threshold.

*'**Percentage'*- Similar to *'Quantitative'* but instead of an amount we run on a percent until we get to 100.

*'**PRC'* uses *sklearn.metrics.precision\_recall\_curve* and for more information I recommend reading online.

**Returns**

**Dataframe** *-* ***[****Algorithm, Threshold, Precision, Recall****]***

**dataframe:** *pandas dataframe, default empty dataframe.*The main dataframe we work on

**target:** *str, default ''*The name of our target column

**key:** *str, default ''*The name of our main key column, usually called 'PN'

**pred\_column\_list:** *list, default []*List of the names of our prediction columns

**precall\_opt:** *str, default 'PRC'*The precision and recall calculation option, can be: *'PRC'*/*'Quantitative'*/*'Percentage'*

**outputpath:** *str, default ''.*If left '' than no file will be created, otherwise it would save a file with the following columns **[***Algorithm*, *Threshold*, *Precision*, *Recall*, *real\_size***]**  
\*'real\_size' is the amount of people in this threshold  
\*Does **not** need to be a full path (only a folder).

If 'precall\_opt' is 'Quantitative'/'Percentage':

**step:** *int, default 1000*This determine the size of the population in each threshold, it is either an amount or a percent depends on your precall\_opt.  
*\*Note that 5% should be 5.0 and not 0.05(!)*

**cut\_loop:** *bool, default True*If left as True than the code will calculate the precision/recall for each step, otherwise if changed to False it will give us the precision/recall for only the step we gave.

**Parameters**

### Threshold\_Selection

As mentioned in the [*Precision\_Recall*](#_Precision_Recall) chapter the results of our predictions come in a lot of different values and not only 1/0, because of that we need to find the best threshold to decide which prediction values would be considered 1 and which 0.

There are two methods to choose the best threshold for our algorithm: [*MATKA*](#MATKA) and [*AUC*](#AUC).

*MATKA* – A method developed in our unit, in this method after we have a list of all the thresholds we want to choose from, and we know their precision and recall we do the following: We start from the lowest precision and go up, this way we start from the best recall and the worst precision. When we go up in the precision we would probably decrease our recall and we stop doing it when we answer one of the conditions below:

1. The recall is lower than the minimum recall we accept
2. There has not been a big enough increase in the precision for a certain amount of thresholds we checked.

As this method runs on a precision recall result dataframe, you can either run [*Precision\_Recall*](#_Precision_Recall) function separately and give the results as a parameter in this function or you can leave the parameter empty and the *Threshold\_Selection* function would run the Precision\_Recall function with its default values, of course you can also edit those parameters if you would like.

*AUC* – This method uses *sklearn.metrics.roc\_curve*. The way this method works is as follows - it creates a graph where **X** is **'False positive'** and **Y** is **'True positive'**.  
**False positive** - All the people we predicted that are target=1 but are actually target=0 divided by all the people that are really 0.  
**True positive** – The recall: All the people we predicted that are target=1 and are actually 1, divided by all the people that are really 1.   
Each point on the graph is a different threshold, the method searches for the point which is the closest to point (fp=0, tp=1) and that’s the best threshold.  
For more information about this method I recommend reading online.

**threshold\_method:** *str, default 'MATKA'*can be *'*[*MATKA*](#MATKA)*'* or *'*[*AUC*](#AUC)*'*

**dataframe:** *pandas dataframe, default empty dataframe.*The main dataframe we work on

**target:** *str, default ''*The name of our target column

**pred\_column\_list:** *list, default []*List of the names of our prediction columns

**outputpath:** *str, default ''.*If left '' than no file will be created, otherwise it would save a file with the following columns **[***Algorithm*, *Precision*, *Recall*, *Threshold*, *Size***]**  
\*Does **not** need to be a full path (only a folder).

If 'threshold\_method' is 'MATKA':

**key:** *str, default ''*The name of our main key column, usually called 'PN'

**minimum\_recall:** *float, default 0.3*  
a float for the minimum recall for a selected threshold

**change\_in\_pred:** *float, default 0.01*  
a float for the minimum increase in our precision we consider relevant for not stopping our loop.

**max\_attempts:** *int, default 500*  
the amount of different values we run on before we check the 'change\_in\_pred'.

**min\_threshold\_size:** *int, default 30*  
the minimum amount of people we accept for a selected threshold

**precall\_df:** *pandas dataframe, default empty dataframe.*The precision and recall results we run on

If 'precall\_df' is empty you can also change the following parameters for the [**Precision\_Recall**](#_Precision_Recall) function:

**precall\_opt:** *str, default 'PRC'*The precision and recall calculation option, can be: [*'PRC'*](#PRC)/[*'Quantitative'*](#Quantitative)/[*'Percentage'*](#Percentage)

If 'precall\_opt' is 'Quantitative'/'Percentage':

**step:** *int, default 1000*This determine the size of the population in each threshold, it is either an amount or a percent depends on your precall\_opt.  
*\*Note that 5% should be 5.0 and not 0.05(!)*

**cut\_loop:** *bool, default True*If left as True than the code will calculate the precision/recall for each step, otherwise if changed to False it will give us the precision/recall for only the step we gave.

**Parameters**

**Returns**

**Dictionary – *{****Algorithm***:** *{Threshold: float, Precision: float, Recall: float, Size:**int****}}***

### Algorithm\_Selection

After we run all the algorithms and choose for each algorithm its best threshold we end with a lot of prediction columns, because each algorithm work a bit differently the results wouldn't be the same and one algorithm could be the best algorithm in one case but not in another case.

*How do we decide which algorithm gives the best results?* – There are a lot of different ways to evaluate the results and each evaluation method can be the best choice, it depends on the situation. The code will do two things: Firstly it will choose an algorithm by its own general logic, secondly it will return a dataframe with all the different evaluation methods that you want to look at for you to choose by your own logic.

*Evaluation methods:*

***Recall*** *–* How many of the target=1 people **we marked as 1** in our prediction. **Higher** is better.

***Precision*** – How many people **are actually target=1** from the people we marked as 1 in our prediction. **Higher** is better.

***F\_measure*** – This is a method that evaluate how good is the ratio between the precision and recall. **Higher** is better.   
The formula:

***Ratio\_between\_groups*** –   
X – Marked as 1 and actually 1, divide it by all the people marked as 1.  
Y – Marked as 0 and actually 1, divide it by all the people marked as 0.  
The result of is the value of this method. **Higher** is better.

***Average\_difference*** – The prediction average of all the people we marked as 1 divided by the prediction average of all the people we marked as 0. **Higher** is better.

***RMSE*** – Root mean square error. Read more online. **Lower** is better.

***MAE*** – The average absolute difference between the target column and the prediction column. **Lower** is better.

– Read more online. **Higher** is better.

***AUC*** – This method creates a graph were as described *here.* The difference is the in our case we don't search for the best threshold on the graph, instead we calculate the area below the graph. **Higher** is better.

The built in general choosing logic:

We save the algorithm with the highest *AUC* and each algorithm which isn't smaller by more than 0.03. If that leaves us with only one algorithm we choose it, otherwise we do the same with the *Precision* method, and then the *F\_measure* method. If we got to the *F\_measure* method and we still have more than one algorithm left, we choose the algorithm with the highest *Recall*.

**evaluation\_methods:** *list, default ['Recall', 'Precision', 'Ratio\_between\_groups', 'Average\_difference', 'RMSE', 'MAE', 'R^2', 'F\_measure', 'AUC']*we can decide if we want all the evaluation methods or just specific ones.

**dataframe:** *pandas dataframe, default empty dataframe.*The main dataframe we work on

**target:** *str, default ''*The name of our target column

**key:** *str, default ''*The name of our main key column, usually called 'PN'

**pred\_column\_list:** *list, default []*List of the names of our prediction columns

**outputpath:** *str, default ''.*If left '' than no file will be created, otherwise it would save a file with a column for each evaluation method and an *'Algorithm'* column.  
\*Does **not** need to be a full path (only a folder).

**best\_threshold\_precall\_dict:** *dict, default {}*  
We could give it the results of the *Threshold\_Selection* algorithm or if left empty we will run it as well.

If ' best\_threshold\_precall\_dict ' is {}:

**threshold\_method:** *str, default 'MATKA'*can be *'*[*MATKA*](#MATKA)*'* or *'*[*AUC*](#AUC)*'*

If 'threshold\_method' is 'MATKA':

**minimum\_recall:** *float, default 0.3*  
a float for the minimum recall for a selected threshold

**change\_in\_pred:** *float, default 0.01*  
a float for the minimum increase in our precision we consider relevant for not stopping our loop.

**max\_attempts:** *int, default 500*  
the amount of different values we run on before we check the 'change\_in\_pred'.

**min\_threshold\_size:** *int, default 30*  
the minimum amount of people we accept for a selected threshold

**precall\_df:** *pandas dataframe, default empty dataframe.*The precision and recall results we run on

If 'precall\_df' is empty you can also change the following parameters for the [**Precision\_Recall**](#_Precision_Recall) function:

**precall\_opt:** *str, default 'PRC'*The precision and recall calculation option, can be: [*'PRC'*](#PRC)/[*'Quantitative'*](#Quantitative)/[*'Percentage'*](#Percentage)

**Parameters**

If 'precall\_opt' is 'Quantitative'/'Percentage':

**step:** *int, default 1000*This determine the size of the population in each threshold, it is either an amount or a percent depends on your precall\_opt.  
*\*Note that 5% should be 5.0 and not 0.05(!)*

**cut\_loop:** *bool, default True*If left as True than the code will calculate the precision/recall for each step, otherwise if changed to False it will give us the precision/recall for only the step we gave.

A tuple with the following items:

1. **Best algorithm** – *string*
2. **Dictionary – *{****Algorithm***:** *{* *Recall: float, Precision: float, F\_measure: float, Ratio\_between\_groups:**float, Average\_difference:**float, RMSE:**float, MAE:**float, R^2:**float, AUC:**float* ***}}***

**Returns**

### DM

This function runs a full prediction, feature selection and algorithm selection for your database. In general you can run while giving only the essential parameters and it will use all the default values, but if you want to edit the way the function works you can change any of the internal functions parameters as you would like.  
*\*This does not apply to the feature selection, so if you want to choose them differently just do not run it.*

**Main default values:**

* If you don't change the default parameters the code will run the algorithms: LDA, GaussianNB and RandomForestClassifier. For RFC we also run a hyper parameters run.
* Precision recall – *PRC*
* Threshold selection - *MATKA*
* Algorithm selection - Choose by this logic: AUC --> Precision --> F\_measure --> Recall

**Returns**

**Dataframe** *–* It will return our pred\_df with the prediction column called – *'Prediction\_the best algorithm name'*

**target:** *str, default ''*The name of our target column

**key:** *str, default ''*The name of our main key column, usually called 'PN'

**train\_df:** *pandas dataframe, default empty dataframe*The dataframe that we want to learn from

**pred\_df:** *pandas dataframe, default empty dataframe*The dataframe that we want to predict on

**features:** *list, default []*A list of the features we want to choose from

**algorithms:** *list, default* *['ld', 'Ga', 'Rf', 'Rf\_Grid']*   
List of the algorithms we want to run.   
\*'Rf\_Grid' is the hyper parameter version of 'Rf'

**run\_feature\_select:** *bool, default True*Indicate if it runs the feature selection as well or not

**outputpath:** *str, default ''.*If left '' than no files will be created, otherwise it would save all the internal files of the functions used by the DM function.  
\*Does **not** need to be a full path (only a folder).

**Parameters**

### STD

Standard deviation – This is a mathematical way to combine close values by their deviation from the average value in the column. STD teaches us about the connection between the different values in a column, this is why we also use it as a normalizing method for our columns.

This function returns a dataframe which holds for each key, for each feature – in what group he is in and the groups target rate.

**Returns**

**Dataframe** *-* ***[****Key, Feature, std, Group, Target rate****]***

**dataframe:** *pandas dataframe, default empty dataframe.*The main dataframe we work on

**target:** *str, default ''*The name of our target column

**key:** *str, default ''*The name of our main key column, usually called 'PN'

**std\_range:** *list, default [float('-inf'), 0, 1, float('inf')]*list of the group ranges we are trying to create, the code may not succeed in creating some of the groups.

**std\_range\_labels:** *list,**default ['smaller than 0','0-1','bigger than 1']*list of the group labels we are trying to create, the code may not succeed in creating some of the groups.

**feats\_for\_std:** *list, default []*  
list of all the columns we want to run the STD function on

**outputpath:** *str, default ''.*If left '' than no file will be created, otherwise it would save a file with the following columns **[***Key*, *Feature*, *std*, *Group*, *Target rate***]**  
\*Does **not** need to be a full path (only a folder).

**Parameters**

# **BI.Analyza**

## **Data\_Analyze**

The ***Data\_Analyze*** class holds functions that let us analyze our data and show visualized results like graphs.

### Decision\_trees

This function returns a CSV file with instructions on how to cut our data in the way that makes the biggest difference of target between our different groups. This is a very good function to use when you want quick results or a simple analyzing of the important features.  
This function can run on all our data or for each group in a specified column.

1. **Nothing (!)**
2. **Saves a CSV file** *-* ***[****For column name (Optional)****,*** *Pattern1, Pattern1\_size, Pattern1\_nesher, Pattern2, Pattern2\_size, Pattern2\_nesher****]***

**Returns**

**dataframe:** *pandas dataframe, default empty dataframe.*The main dataframe we work on

**target:** *str, default ''*The name of our target column

**features:** *list, default []*The list of all the features we want to see how they influence our target

**for\_column:** *str, default ''*The name of the column by which we want to cut our data. If left as '' then the function would look at all our data as one group.

**outputpath:** *str, default ''.*It would save a file with the following columns ***[****For column name (Optional)****,*** *Pattern1, Pattern1\_size, Pattern1\_nesher, Pattern2, Pattern2\_size, Pattern2\_nesher****]***  
\*Needs to be a full path (not only a folder).

**Parameters**

### Graphs

This function creates graphs were the Y axis is the feature unique values and the X axis is the target percentage in this specific value. We can run this function one time for multiple features (columns).

1. **Nothing (!)**
2. **Saves jpg files**

**Returns**

**dataframe:** *pandas dataframe, default empty dataframe.*The main dataframe we work on

**target:** *str, default ''*The name of our target column

**graph\_features:** *list, default []*The list of all the features we want to create a graph on

**outputpath:** *str, default ''*It will save jpg files with the names of the features.  
\*Does **not** need to be a full path (only a folder).

**Parameters**

### Group\_Comparison

This function helps us determine if a two groups are different or not. There are two ways that we use to decide how similar the groups are:

1. By value – the average of the comparison column
2. By distribution – the way the comparison column's values are distributed in the group

Basically if we choose option 1 we will have different groups and for each group its average comparison column average, we can easily see if the averages are close or not. We can also run a similarity test called *T-test*.   
If we choose option 2 we will get a pivot were we can see for each group how its population distributed between the values of the comparison column. It’s a bit harder to decide if the distribution is similar so we use a special test called *Chi square-test*.

*T-test* – **No info yet**

*Chi square-test* – **No info yet**

**compare\_method:** *str, default 'Value'*Can be either *'Value'* or ' *Distribution'*

**dataframe:** *pandas dataframe, default empty dataframe.*The main dataframe we work on

**target:** *str, default ''*The name of our target column

**compare\_col:** *str, default ''*The name of the column we want to check the similarity between each of his value. The code cuts each value as a different group and compares it to all the other values.

**outputpath:** *str, default ''*It will save CSV file of the basic comparison  
\*Needs to be a full path (not only a folder).

**special\_test\_outputpath:** *str, default ''*It will save CSV file of the basic comparison  
\*Needs to be a full path (not only a folder).

If ' compare\_method ' is 'Distribution':

**distribution\_col:** *str, default ''*The column on which we check the people distribution of each group

**key:** *str, default ''*The name of our main key column, usually called 'PN'. In this case we will check the distribution of the keys in this column

1. **Nothing (!)**
2. **Saves CSV files –** one for the basic comparison and one for the special test

**Returns**

**Parameters**