# CFD Command Line Interface Guide

*Please read this guide in its entirety before using this tool for the first time.*

## What is CFD?

Combination Frequency Differencing (CFD) is a technique used to identify patterns that distinguish one group of data from another by analyzing how combinations of variable values (ex, color + shape + size) appear across the groups. For the purposes of this document, variables will be referred to as features.

Rather than looking at features individually, CFD examines all t-way interactions (ex, 2-way, 3-way, etc.) to find which combinations of feature values occur distinguishingly more frequently in one file than in the other, making them Distinguishing Value Combinations.

Distinguishing Value Combinations (DVCs) which can be represented with fewer values **will** be included in higher t-way outputs by default. For example, if (Small, Circle) was a 2-way DVC, then (Small, Blue, Circle) would **also** be a 3-way DVC. This behavior can be disabled, see the optional `--filter` flag.

## Notes:

- The file, `CFD_CLI.py`, is a standalone file requiring no other downloads to run.
- You will need to install the following libraries if not already present
    - pandas —  Dataframes for reading CSV data
    - numpy — Mathematical operations
    - matplotlib.pyplot — Graph creation
    - matplotlib.axes — Type enforcing the axes object
    - matplotlib_venn — Venn diagram graphs
    - termcolor — Colored terminal output
    - tqdm — Progress bars in terminal when logging output and an operation takes more than 3 seconds
    - kmeans1d — Binning operations

# Running the program:

To run the file, execute the following command in your command line/terminal:
`python .\CFD_CLI.py .\class.csv .\nonclass.csv #`
where `.\class.csv` and `.\nonclass.csv` are replaced by the names of the files you wish to compare, or paths to them if they're not where the CLI is, and '`#`' replaced by your cutoff for distinguishability, which can be any real number above 1.

- Ex: `python .\CFD_CLI.py .\reptiles.csv .\notreptiles.csv 5`

If running in VSCode or similar, make sure you are where your python is installed before executing the command (ex, your Powershell terminal)

# Required Parameters:

## Class and Nonclass Files:

- The Class file contains the group of interest
    - Ex, positive cases, successes, target outcomes, or "selected" items.
- The Nonclass file contains a comparison group
    - Ex, general data, failures, negatives, or "non-selected" items.

Each file should have the same columns (features), but different sets of rows. Additionally, each feature should be either categorical (discrete) or binned.

## Distinguishability Cutoff:

The distinguishability cutoff is a threshold that determines how strong the difference must be between the class and nonclass frequencies for a combination to be considered "distinguishable."

A higher cutoff means only the most distinct or extreme differences are reported. A lower cutoff includes more subtle differences. This helps filter out noise to focus on the most meaningful patterns. T-way value combinations unique to one file will always be considered DVCs.

## Optional Flags:

There are several optional flags you may also use:

- "`-v`" or "`--verbose`" will enable verbose outputs should either of your two data files contain a header row with the feature names. If false, features will be named Var_1, Var_2, …, Var_n. *Default: False*
- "`-l`" (lowercase L) or "`--logs`" will log mid-program execution statements to the console and a dedicated logs file. *Default: False*
- "`-o`" or "`--overwrite`" will disable the manual Y/N check for overwriting existing output files with new ones when run on the same data files with the same distinguishability cutoff. Note: any files the user has added to the output folder's subfolder will be lost if an overwrite occurs. Additionally, the program only checks what the class, nonclass, and distinguishability cutoff were when determining whether an output folder for this data already exists, not any optional flags like max t-ways or graphs. *Default: False*
- "`-g`" or "`--graphs`" will additionally produce graphical outputs in a dedicated Graphs subfolder. More detail about these is further in this guide. *Default: False*
- "`-m`" or "`--mvcs`" will additionally log all missing value combinations up to the specified max t-way. More detail about these is further in this guide. *Default: False*
- "`-r`" or "`--remove_duplicates`" will disable the manual Y/N check for removing duplicate rows from both the class and nonclass files, always removing these rows before continuing instead. This will not affect the original data files themselves. *Default: False*
- "`-c`" or "`--complete`" will disable the manual Y/N check for removing rows which contain missing data, always removing these rows before continuing instead. This will not affect the original data files themselves. *Default: False*
- "`-t #`" or "`--tway #`" will set the max number of t-way interactions you would like to calculate, indicated by #, which can be any number between 1 and 6 inclusive. For this guide, $t_{max}$ represents this value. *Default: 3*

- "`-f`" or "`--filter`" will filter out higher t-way DVCs which contain lower t-level DVCs within them. Ex, if (Small, Circle) and/or (Blue, Circle) and/or (Small, Blue) was a 2-way DVC, then (Small, Blue, Circle) would not be a 3-way DVC, (Small, Blue, Smooth, Circle) would not be a 4-way DVC, and so on upwards. *Default: False*

- "`--silent`" will silence all terminal print statements except for required user interaction prompts and error messages. Logs can still be enabled for generating a logs file output. *Default: False*
    - If you wish to just silence the timestamp statement and/or the info statements in the terminal, you can specify "`--silent ts`" and/or "`--silent info`" respectively instead. *Defaults: False*

- "`--drop`" will automatically drop all columns which the CFD tool believes will not assist in its calculations — namely all columns which contain only one value in both files. If the value is the same between files then it cannot contribute to distinguishing them. If it is a different value, but still only one within each file respectively, then it may be the column the data was split into class/nonclass files upon. Regardless, it will also be dropped. This will not affect the data files themselves. If the flag is not enabled, the user will be prompted whether they'd like to remove each qualifying column. *Default: Check*
    - If you wish to automatically never drop these columns, use "`--drop none`"

- "`--out '[name]'`" will specify an output folder name for all results. It will be created if it does not exist. *Default: "output"*

- "`--delim '[char]'`" will specify a delimiting character for your class and nonclass CSV files. It must be the same for both. *Default: ','* (a comma)

    - For tabs or whitespace delimiters, use "`--delim '\s+'`"

- "`--help`" can be used to get general information about the CFD tool.

# Output:

## Terminal Messages:

If "`--logs`" were enabled, then as various parts of the program finish executing, timestamp messages will be printed to the console. Should a particular portion of the program take more than 3 seconds to complete, a progress bar will appear and update as it progresses. These are printed in green.

Some informational print statements will print as the program executes, such as whether any rows with duplicate values were detected and/or removed, whether any rows with missing data were detected and/or removed, if header row(s) were detected in the input files but verbose mode was not enabled, and the program's total runtime. These are printed in cyan.

Some non-fatal warnings may be printed as well, such as if verbose mode was enabled but no header row was detected in either input file, if columns which contain unhelpful data are detected, if the class and nonclass file names are the same, whether any feature values are distinguished solely by quotes or whitespace, and, if the "`--overwrite`" flag was not enabled, before any folder overwrite. These warnings are printed in yellow.

Fatal errors include whether the input files do not refer to the same features, if $t_{max}$ is invalid, if the cutoff for distinguishability is invalid, or if $t_{max}$ exceeds the number of features present in the input files, among others. These will stop the program execution, and are printed in red.

## Outputs Folder:

The program's primary output will be saved to an outputs folder, and will be a folder named `class-nonclass-#`, where `class`, `nonclass`, and `#` are the input parameters. Within this folder will be all the outputs for a given runtime. If this folder already exists, and the overwrite parameter is not enabled, the program will prompt the user whether they wish to overwrite the existing folder before proceeding. Below are all the outputs which this folder will or can contain:

## Info:

An info file named Info.txt will contain basic information about when the program was run, the files it was generated with, how many rows each had, the number of features, their names (if verbose was enabled), whether any columns were dropped, whether any rows were present in both files, and a series of other summary statistics.

## Distinguishing Value Combinations (DVCs):

The DVCs folder will contain a series of CSV files separated by t-level, each containing a header row. The internal data will be stored as:
`[U/D];[1/0];(feature, value)`[1]`;Freq;Freq_diff`[2]

- `U/D` refers to whether the value combination is Unique or Distinguishing but not unique.
- `1/0` refers to whether this combination is present uniquely or distinguishably more often in the class file (`1`) or the nonclass file (`0`).
- A series of `(feature, value)` pairings ([1] t of them, for a given t-way interaction), separated by semicolons, will then indicate the features and values comprising this DVC. If verbose was enabled the feature name will be used instead of its index.
    - Ex, for t=3: `(Var 2, 1);(Var 5, 0);(Var 7, 1);` can be produced.
- `Freq` indicates how often this DVC appeared in the file indicated by `1/0`.
- `Freq_diff` indicates how much more frequently this DVC appeared in one file than the other. [2]For unique DVCs, this column is left empty as those t-way value combinations only appear in one file.

An example of a full row is: `D;1;(Var 9, 1);(Var 12, 0);0.4000;7.2000;`
Here, when Var 9 had a value of 1 and Var 12 had a value of 0, that 2-way value combo occurred in 40% of class rows, and 7.2 times more often in the class file. Entries are sorted first by unique then distinguishing, and sub-sorted by frequency appearance in the majority class. This ordering can be changed by uncommenting a line in `CFD_CLI.py`. See the `#NOTE`s in the `generate_output_statements` method.

## [filename]_duplicates.csv

This CSV file, which can be either [class_filename]_duplicates and/or [nonclass_filename]_duplicates, will contain a CSV-formatted output of all duplicate rows in your class and/or nonclass folder respectively, if present.

## Logs.txt

The Logs file will contain timestamped program execution statements, as well as all Info and Warning statements. It will only appear if the optional "`--logs`" parameter is enabled.

## Missing Value Combinations (MVCs):

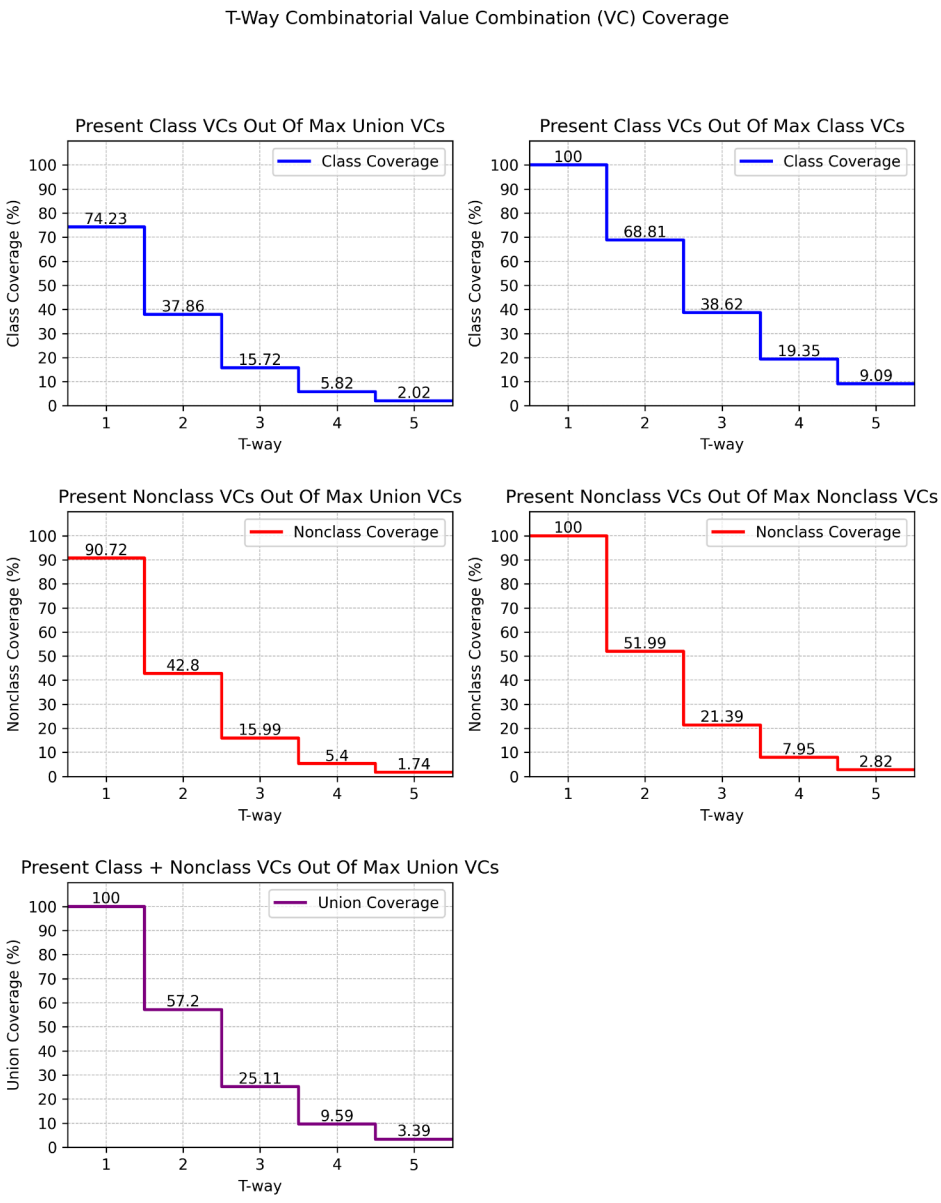
The MVCs folder contains a series of text files explaining every missing value combination (MVC) at a given t-level, up to $t_{max}$. If verbose mode is enabled, feature names will be used instead of indexes. To briefly explain what a MVC is, if a file had just a blue square and an orange circle, then (blue, circle) and (orange, square) would be two 2-way MVCs. 3-way MVCs which contain a 2-way MVC are excluded from the list of 3-way MVCs. Ex, if we had a small blue square and a large orange circle, then (large, blue, circle) would not be listed as a 3-way MVC as it is already covered by (blue, circle) and by (large, blue). This applies further upward as well, any t-way MVC which contains a $p<t$ p-way MVC will not be listed as a t-way MVC. As such, one can infer that if (blue, circle) is absent, so are all 3-way size groupings which include it. This folder will only be generated if the "`--mvcs`" flag is enabled.

# T-way Coverage Graphs:

T-way Combinatorial Value Combination Coverage Graphs will always be produced regardless of whether other graphical output is requested. These indicate the combinatorial coverage of the class and nonclass files compared to the maximum number of t-way value combinations possible, up to $t_{max}$.

Some values may only be possible / present within the class or nonclass file, but not the other. To account for this, the rightmost graphs only consider the feature values present within that class when considering the maximum number of possible t-way feature value interactions within it.

The leftmost three graphs assume that all features can take on any value present for that feature between the two files in any combination. Then, they see how many of those combinations are present within the class file, the nonclass file, and the union between the class and nonclass files.

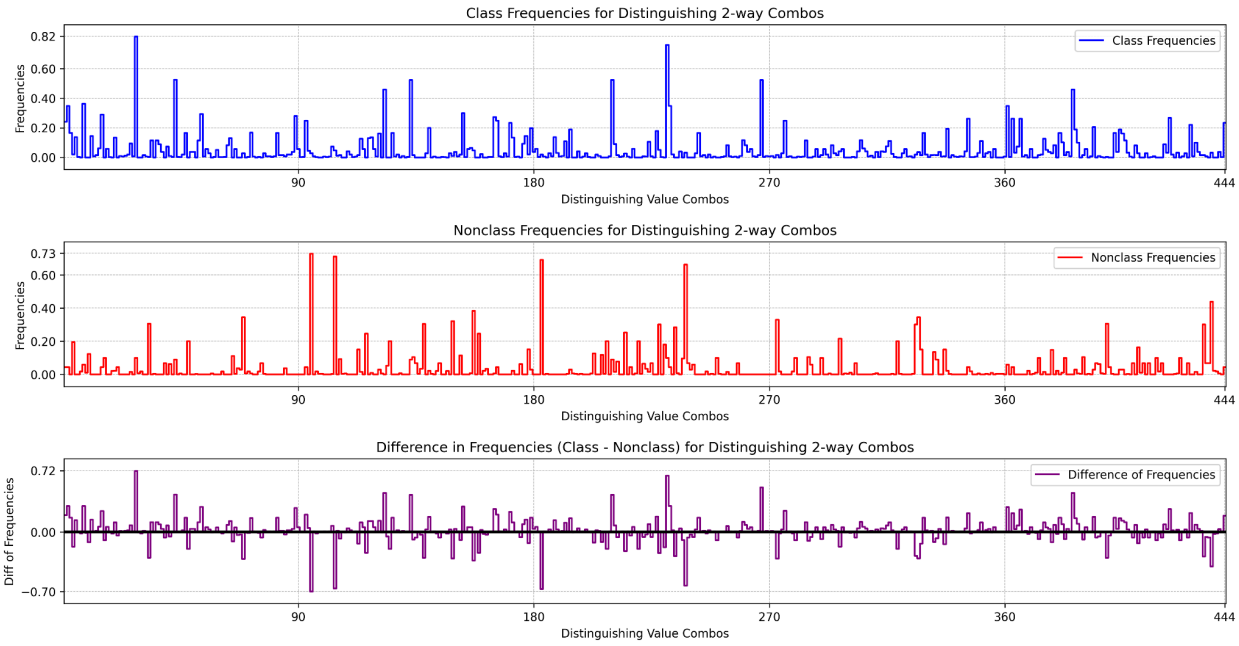


# Other Graphs:

If "`--graphs`" are enabled, then several other graphical outputs will be saved to a Graphs subfolder.

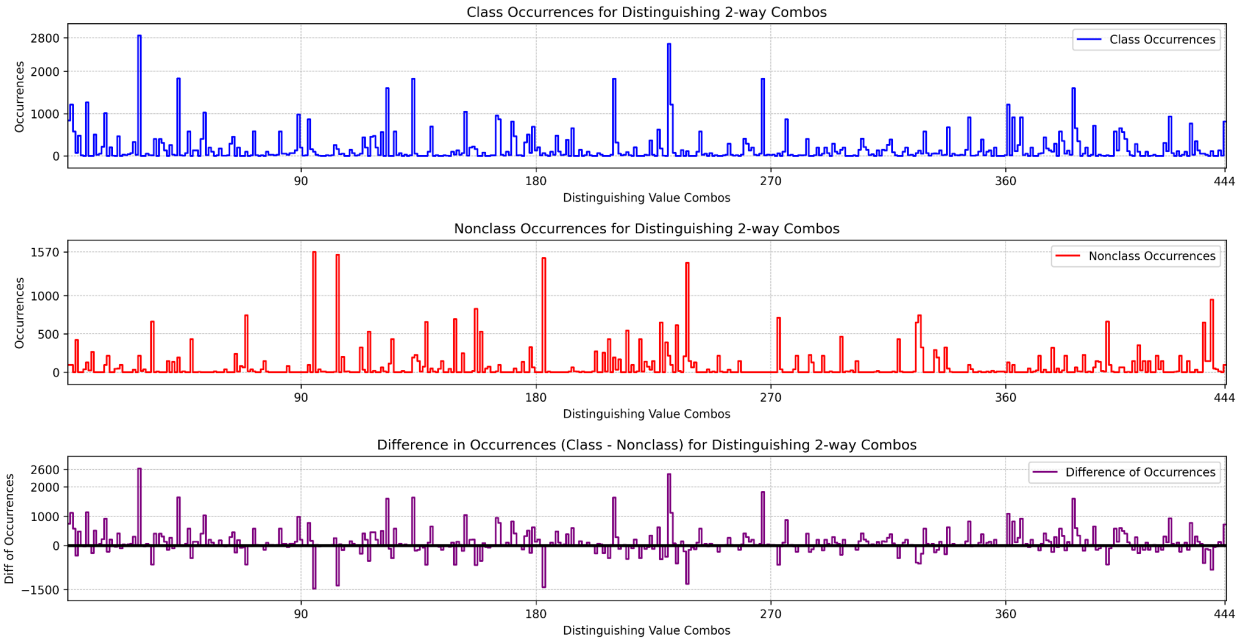
## Distinguishing Frequencies

Within the Graphs subfolder will be a subfolder named Distinguishing Frequencies. In this folder, there will be several step graphs named `Distinguishing_Frequencies_[t]_way.png` indicating the spread of DVC coverage across the class and nonclass files for that t-way value interaction respectively, from t=1 to $t_{max}$. Each step represents a t-way DVC. The height of the step for class represents how often it occurred as a percent of the number of entries in its respective file (class/nonclass). The third graph shows the frequency difference for each DVC between the class and nonclass files, with negative values indicating a greater frequency in the nonclass file than in the class file. The approximate highest and lowest frequencies will always be marked on the y-axis.



## Distinguishing Occurrences

Within the Graphs subfolder will also be a subfolder named Distinguishing Occurrences. In this folder, there will be several step graphs named `Distinguishing_Occurrences_[t]_way.png` indicating the spread of DVC coverage across the class and nonclass files for that t-way value interaction respectively, from t=1 to $t_{max}$. Similarly to the distinguishing frequencies graphs, each step represents a t-way DVC. In contrast, for these graphs the height of the step for class represents the numerical total of how often it occurred in its respective file (class/nonclass). The third graph shows the occurrence difference for each DVC between the class and nonclass files, with negative values indicating a greater number of occurrences in the nonclass file than in the class file. The approximate highest and lowest occurrences will always be marked on the y-axis.




Between the Distinguishing Frequencies and Distinguishing Occurrences graphs, the class and nonclass graphs will have the same overall shape. The difference graphs, however, have the possibility of variation, especially if the datasets are unbalanced (leading to, for example, more occurrences of a DVC in the class but its frequency being lower due to far more rows in the class file balancing it out.

## Value Combo Graphs

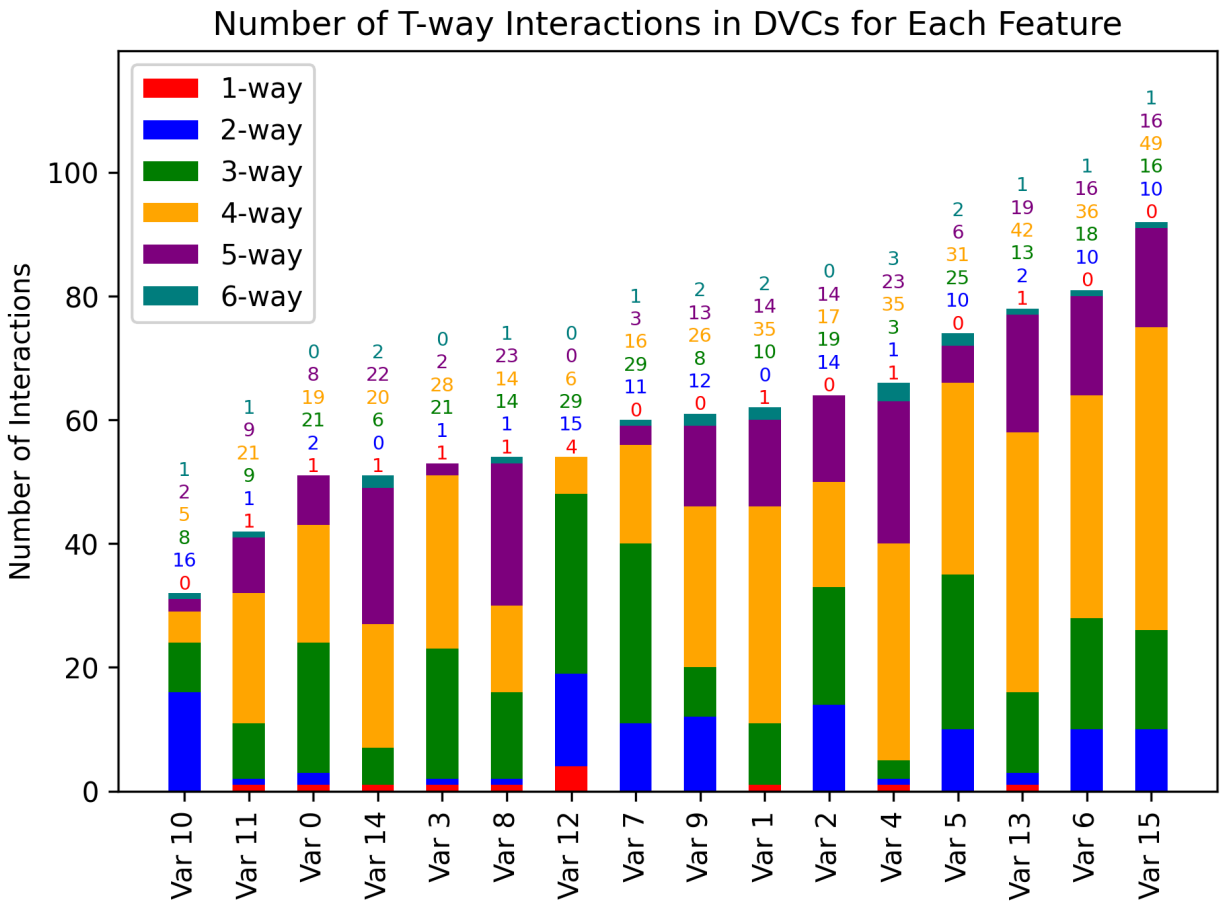
There are two value combo graphs, named `Value_Combo_Frequencies` and `Value_Combo_Occurrences`. These graphs include ALL t-way value combinations (VCs) from t=1 to $t_{max}$, regardless of whether they are distinguishing or not.

`Value_Combo_Frequencies` charts how often each VC appears relative to the size of the class file, to the size of the nonclass file, and the difference between those values (with negative values representing VCs which had higher nonclass frequency. The `Value_Combo_Occurrences` graph shows the same information, but substitutes the VC's frequency with the number of times that VC occurred in the class and nonclass file respectively. Positive values represent VCs which occurred more often (or potentially uniquely) in the class file, while negative values represent the same but for the nonclass file. The top and bottommost y axis tick values represent the approximate extremes of the values within the graphs.

For both graphs, the x axis of VCs is also separated into portions representing what t-way is responsible for which portion of the graph, as well as the total number of VCs which have been covered to that point (cumulatively)

## Feature Bar Graph

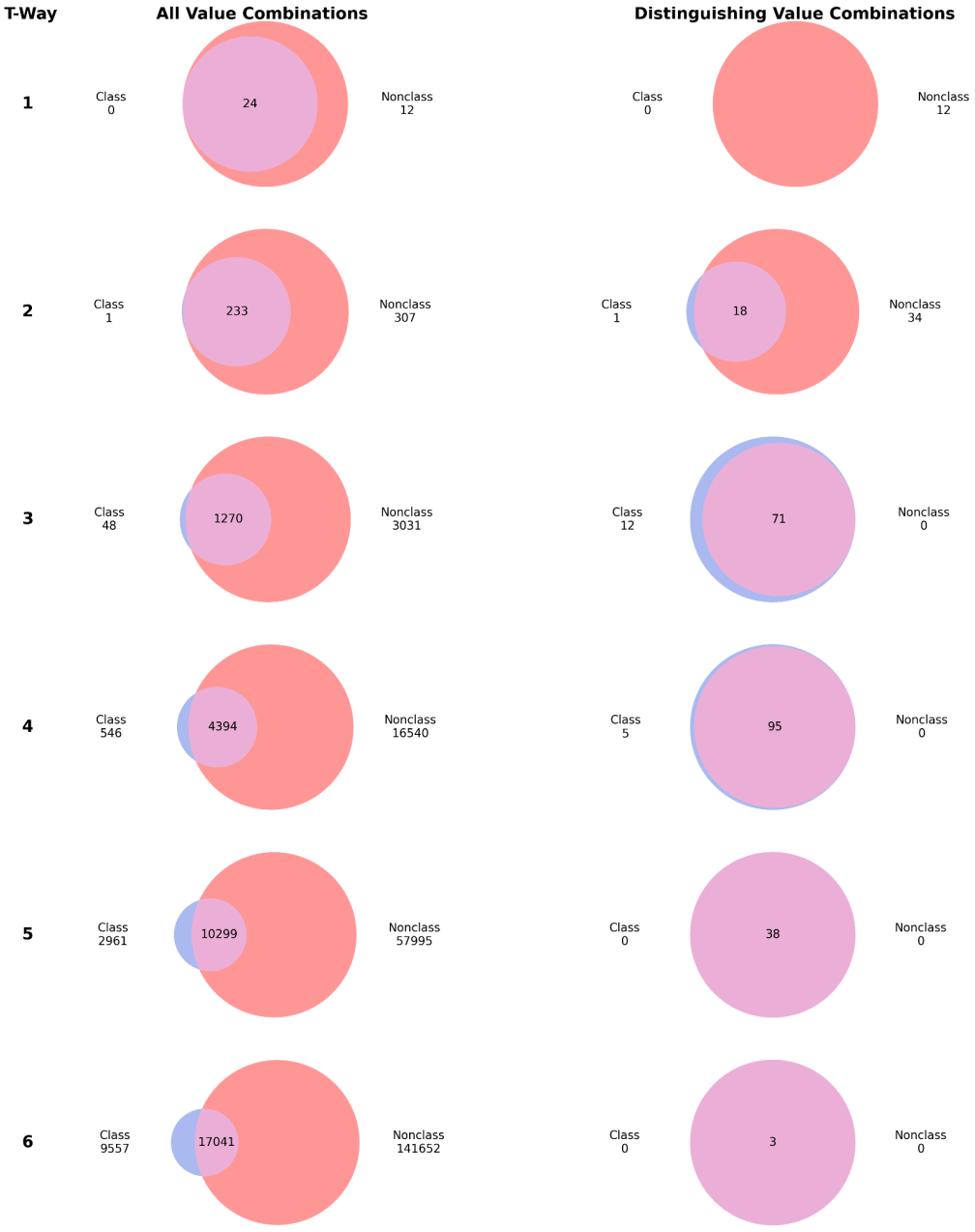
Additionally, there is a `feature_bar_graph.png` which shows how often each feature is part of a t-way DVC, up to $t_{max}$. Feature names are used if present. Each color represents a different t-way, up to $t_{max}$, and are stacked on top of each other cumulatively in the order of lowest t-values first. All bars are then sorted by total height, to easily see which feature(s) were part of the most or fewest DVCs. The numbers above the bars show the values for each t-way, color coded the same as that t-way's bar portion, and in the same order.




As this can become illegibly crowded when there are too many features, all information presented in the bar graph is also present in CSV format in a file named `feature_DVC_bar_graph_[class_name]_[nonclass_name]_[cutoff].csv`. The CSV is formatted `Name,1-way,2-way,3-way,4-way,5-way;6-way` up to $t_{max}$.

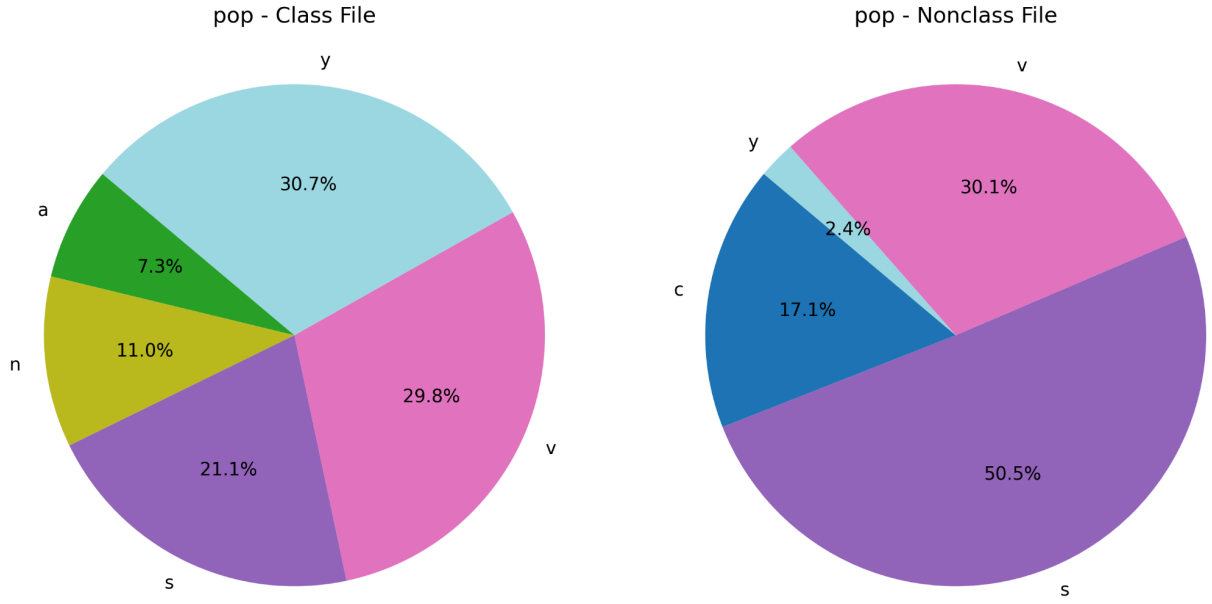
# Venn Diagrams

This program also provides a set of venn diagrams, showing for each t-level how many total value combinations and distinguishing value combinations were present in the class, nonclass, or in both files respectively, up to $t_{max}$.



## Pie Charts

Lastly, there will be an inner subfolder labeled Pie Graphs, which contains PNGs for each feature present. Each will contain two venn diagrams, the left for class and the right for nonclass, which contain the breakdown of values present for that feature in that file, as well as how often each appeared as a percent. Feature names are indicated around the outside of the circle.



# Disclaimer (README)

This tool is provided as a research prototype and is intended for experimental and informational purposes only. Users are advised that the tool may produce inaccurate or incomplete information and may contain unknown bugs or errors. The results generated by the tool should be treated with caution and should not be relied upon as the sole basis for decision-making in critical settings.

Users are strongly encouraged to validate the tool's output using other reliable and established methods if able before making any critical decisions. The developers and providers of this tool shall not be held responsible for any consequences arising from the use of its results without proper validation or for any actions taken based on the tool's output.

By using this tool, you acknowledge and accept these inherent limitations and uncertainties, and agree to exercise due diligence in verifying its results prior to implementation in critical or important contexts.

# Known Bugs / Issues / Quirks:

As bugs / issues / quirks become known, they will be listed here as well as how to avoid them and what versions they affect. Bugs are known problems, issues are unintended behavior from certain functions in specific circumstances, and quirks are behaviors which may be unexpected but are inherent to the tool's design.

## CSV Header Detection:

- QUIRK:
    - Header detection in the class / nonclass CSV files works by checking whether any of the values in the first row of that file appear elsewhere in their column. If any do, it is not considered a header row. Otherwise, it is considered a header row (and removed as such). This allows for CSVs with and without headers to be accepted by the CFD tool without further user specification, including within the same runtime instance.
- AVOIDANCE:
    - To avoid an intended header row being misinterpreted as data, ensure that none of the feature names are also a value that feature can take on.
    - To avoid a data row being misinterpreted as a header, ensure that at least one of the values in the first row of your data appears elsewhere in its column. Since your data should be categorical, this will typically be the case.
- VERSIONS AFFECTED:
    - Full Release 1.0.0+
    - Dev Release 1.0.1+

## Accidental Column Removal:

- QUIRK:
    - When the "`--drop`" flag is enabled, some columns may be dropped unintentionally. Specifically, if the class file's column A contains just one value, ex 'yes', and the nonclass file's column A contains just one of a different value, ex 'no', then the CFD tool would assume this was

the column used to split the class and nonclass files and drop it. However, another column may have been used and not this one.

- AVOIDANCE:
    - When using "`--drop`", ensure that the above is not true. Specifically, make sure that the feature column used to split the data is not present in the class and nonclass files, and that no column in both contains just a single value, shared across both files. Once done, there is no need to enable "`--drop`". By default, the user will be prompted whether to drop these columns.
- VERSIONS AFFECTED:
    - Full Release 1.0.0+
    - Dev Release 1.0.1+

## Non-ASCII Characters:

- BUG:
    - Non-ASCII characters (ex, letters with accents, Hanzi/Kanji characters, emojis) can break the tool.
- AVOIDANCE:
    - Replace these characters in your datasets before using the tool.
    - Additionally, avoid using ASCII control characters (ex, '&' or '\').
- VERSIONS AFFECTED:
    - Full Release 1.0.0+
    - Dev Release 1.0.1+

## Missing Values in Data:

- BUG:
    - CSV files with missing and/or NaN values can cause the tool to fail under some unknown circumstances.
- AVOIDANCE:
    - Use files with no missing data if able. Otherwise, consider dropping incomplete rows / columns if the tool breaks.
- VERSIONS AFFECTED:
    - Full Release 1.0.0+
    - Dev Release 1.0.1+

## Semicolons in Values

- QUIRK:
    - Feature values containing a semicolon — ; — will break the tool as it uses these when generating DVC files. This was chosen so that the more common case of feature values containing a comma would *not* break the tool.
- AVOIDANCE:
    - Replace semicolons in feature values before processing.
- VERSIONS AFFECTED:
    - Full Release 1.0.0+
    - Dev Release 1.0.1+

## Quotes Around Feature Values

- ISSUE:
    - Feature values which contain quotes around them in the dataset, ex “True” or “7”, while acceptable on their own, will be treated as equivalent to those values without quotes and/or with single quotes in an unpredictable way if both are present within a given feature.
- AVOIDANCE:
    - Do not use the presence or absence of quotes to differentiate between values within a given feature.
- VERSIONS AFFECTED:
    - Full Release 1.0.0+
    - Dev Release 1.0.1+