Warzone Analysis

Release v2.2.0

Peter Rigali

Aug 31, 2021

CONTENTS:

CHAPTER

ONE

INTRO

1.1 Usage

Calling the CallofDuty Class:

```
cod = CallofDuty(hacker_data=False, squad_data=True, streamer_mode=True)
```

This is the core class which holds all objects the user may need.

1.2 Write Ups and Examples

Home Page

CHAPTER

TWO

CLASSES

This chapter documents the classes used.

2.1 CallofDuty

CallofDuty(hacker_data, squad_data, streamer_mode):

Calculate stats for all maps/modes for each squad member.

Parameters

- **hacker_data** (*bool*) – This Requires a seperate csv with hacker data saved. This data can be collected by finding hackers after the fact and scraping there data from CodTracker, this can then be used to find hackers in other games. Default is False. *Optional*
- **squad_data** (*bool*) – If True, will build the Squad class. default is True. *Optional*
- **streamer_mode** (*bool*) – If True, will hide User inputted Gamertag's and Uno's. default is False. *Optional*

Example

```
>>> from Classes.call_of_duty import CallofDuty
>>> cod = CallofDuty(hacker_data=False, squad_data=True, streamer_
↪mode=False)
```

Note This will calculate and build the CallofDuty class.

call_of_duty.CallofDuty.whole	The unedited player matches DataFrame
call_of_duty.CallofDuty.gun_dictionary	Returns a dict of gun names
call_of_duty.CallofDuty.last_match_date_time	Returns a Timestamp of the latest game in the players data.
call_of_duty.CallofDuty.name_uno_dict	Returns a dict of gamertags and respective unos
call_of_duty.CallofDuty.my_uno	Returns the user uno value
call_of_duty.CallofDuty.our_df	Returns a DataFrame of all data related to player and there teammates
call_of_duty.CallofDuty.other_df	Returns a DataFrame of all data related to other teams in a lobby
call_of_duty.CallofDuty.hacker_df	If a hacker DataFrame is provided, will return just the hacker DataFrame
call_of_duty.CallofDuty.name_uno_dict_hacker	If a hacker DataFrame is provided, will return the gamertags: unos for the hacker DataFrame

continues on next page

Table 1 – continued from previous page

call_of_duty.CallofDuty.user	Returns a User class object of related info to the user
call_of_duty.CallofDuty.squad	Returns a Squad class object of stats related to the user squad mates

2.2 DocumentFilter

DocumentFilter class objects.

DocumentFilter(hacker_data, squad_data, streamer_mode):

Get a selection from a DataFrame. Uses a set of filters to return a desired set of data to be used in later analysis.

Parameters

- **original_df** (*pd.DataFrame*) – Input DataFrame to be filtered.
- **map_choice** (*str*) – Map filter. Either 'mp_e' for Rebirth and 'mp_d' for Verdansk. *Optional*
- **mode_choice** (*str*) – Mode filter. Either 'solo', 'duo', 'trio', or 'quad'. *Optional*
- **username** (*str*) – Filter by a players username. Can cause errors if same username as another player. *Optional*
- **uno** (*str*) – Filter by a players uno. *Optional*
- **username_dic** (*dict*) – Required if 'username' or 'username_lst' is used. {username1: uno1, username2: uno2, etc}. *Optional*
- **username_lst** (*List[str]*) – Filter using a list of usernames. *Optional*

Example

```
>>> from document_filter import DocumentFilter
>>> doc = DocumentFilter(original_df=cod.our_df, map_choice='mp_e',␣
↪mode_choice='quad')
```

Note All inputs, except original_df, are *Optional* amd defaults are set to None. This will return any data with map = rebirth and mode = Quads. By specifiying 'cod.our_df', this will only return data related to the user.

document_filter.DocumentFilter.df	Returns the filtered DataFrame
document_filter.DocumentFilter.map_choice	Returns the map used to filter
document_filter.DocumentFilter.mode_choice	Returns the mode used to filter
document_filter.DocumentFilter.uno	Returns the uno used to filter
document_filter.DocumentFilter.username	Returns the username used to filter
document_filter.DocumentFilter.username_lst	Returns the username list used to filter
document_filter.DocumentFilter.unique_ids	Returns unique match ids from the filtered DataFrame
document_filter.DocumentFilter.ids	Returns match ids from the filtered DataFrame
document_filter.DocumentFilter.username_dic	Returns username: uno dict

2.3 Plot

Plot class objects.

font size = ['xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large']

Legend location = ['best', 'upper right', 'upper left', 'lower left', 'lower right', 'right', 'center left', 'center right', 'lower center', 'upper center', 'center']

Line:

Class for plotting line plots.

Parameters

- **data** (*pd.DataFrame,*) – Input data.
- **limit** (*int*) – Limit the length of data. *Optional*
- **label_lst** (*List[str]*) – List of labels to include, if None will include all columns. *Optional*
- **color_lst** (*List[str]*) – List of colors to graph. *Optional*
- **normalize_x** (*List[str]*) – List of columns to normalize. *Optional*
- **running_mean_x** (*List[str]*) – List of columns to calculate running mean. *Optional*
- **running_mean_value** (*int*) – Value used when calculating running mean, default = 50. *Optional*
- **cumulative_mean_x** (*List[str]*) – List of columns to calculate cumulative mean. *Optional*
- **fig_size** (*tuple*) – Figure size, default = (10, 7). *Optional*
- **ylabel** (*str*) – Y axis label. *Optional*
- **ylabel_color** (*str*) – Y axis label color, default = 'black'. *Optional*
- **ylabel_size** (*str*) – Y label size, default = 'medium'. *Optional*
- **xlabel** (*str*) – X axis label. *Optional*
- **xlabel_color** (*str*) – X axis label color, default = 'black'. *Optional*
- **xlabel_size** (*str*) – X label size, default = 'medium'. *Optional*
- **title** (*str*) – Graph title, default = 'Line Plot'. *Optional*
- **title_size** (*str*) – Title size, default = 'xx-large'. *Optional*
- **grid** (*bool*) – If True will show grid, default = true. *Optional*
- **grid_alpha** (*float*) – Grid alpha, default = 0.75. *Optional*
- **grid_dash_sequence** (*tuple*) – Grid dash sequence, default = (3, 3). *Optional*
- **grid_lineweight** (*float*) – Grid lineweight, default = 0.5. *Optional*
- **legend_fontsize** (*str*) – Legend fontsize, default = 'medium'. *Optional*
- **legend_transparency** (*float*) – Legend transparency, default = 0.75. *Optional*
- **legend_location** (*str*) – legend location, default = 'lower right'. *Optional*

Example *None*

Note *None*

plot.Line.ax	Returns a plot

Scatter:

Class for plotting scatter plots.

Parameters

- **data** (*pd.DataFrame,*) – Input data.
- **limit** (*int*) – Limit the length of data. *Optional*
- **label_lst** (*List[str]*) – List of labels to include, if None will include all columns. *Optional*
- **color_lst** (*List[str]*) – List of colors to graph. *Optional*
- **normalize_x** (*List[str]*) – List of columns to normalize. *Optional*
- **regression_line** (*List[str]*) – If included, requires a column str or List[str], default = None. *Optional*
- **regression_line_color** (*str*) – Color of regression line, default = 'red'. *Optional*
- **regression_line_lineweight** (*float*) – Regression lineweight, default = 2.0. *Optional*
- **running_mean_x** (*List[str]*) – List of columns to calculate running mean. *Optional*
- **running_mean_value** (*Optional[int] = 50,*) – List of columns to calculate running mean. *Optional*
- **cumulative_mean_x** (*List[str]*) – List of columns to calculate cumulative mean. *Optional*
- **fig_size** (*tuple*) – default = (10, 7), *Optional*
- **ylabel** (*str*) – Y axis label. *Optional*
- **ylabel_color** (*str*) – Y axis label color, default = 'black'. *Optional*
- **ylabel_size** (*str*) – Y label size, default = 'medium'. *Optional*
- **xlabel** (*str*) – X axis label. *Optional*
- **xlabel_color** (*str*) – X axis label color, default = 'black'. *Optional*
- **xlabel_size** (*str*) – X label size, default = 'medium'. *Optional*
- **title** (*str*) – Graph title, default = 'Scatter Plot'. *Optional*
- **title_size** (*str*) – Title size, default = 'xx-large'. *Optional*
- **grid** (*bool*) – If True will show grid, default = true. *Optional*
- **grid_alpha** (*float*) – Grid alpha, default = 0.75. *Optional*
- **grid_dash_sequence** (*tuple*) – Grid dash sequence, default = (3, 3). *Optional*
- **grid_lineweight** (*float*) – Grid lineweight, default = 0.5. *Optional*
- **legend_fontsize** (*str*) – Legend fontsize, default = 'medium'. *Optional*
- **legend_transparency** (*float*) – Legend transparency, default = 0.75. *Optional*
- **legend_location** (*str*) – legend location, default = 'lower right'. *Optional*

Example *None*

Note *None*

plot.Scatter.ax	Returns a plot

Histogram:

Class for plotting histograms.

Parameters

- **data** (*pd.DataFrame,*) – Input data.
- **limit** (*int*) – Limit the length of data. *Optional*
- **label_lst** (*List[str]*) – List of labels to include, if None will include all columns. *Optional*
- **color_lst** (*List[str]*) – List of colors to graph. *Optional*
- **include_norm** (*str*) – Include norm. If included, requires a column str, default = None. *Optional*
- **norm_color** (*str*) – Norm color, default = 'red'. *Optional*
- **norm_lineweight** (*float*) – Norm lineweight, default = 1.0. *Optional*
- **norm_ylabel** (*str*) – Norm Y axis label. *Optional*
- **norm_legend_location** (*str*) – Location of norm legend, default = 'upper right'. *Optional*
- **fig_size** (*tuple*) – default = (10, 7), *Optional*
- **bins** (*str*) – Way of calculating bins, default = 'sturges'. *Optional*
- **hist_type** (*str*) – Type of histogram, default = 'bar'. *Optional*
- **stacked** (*bool*) – If True, will stack histograms, default = False. *Optional*
- **ylabel** (*str*) – Y axis label. *Optional*
- **ylabel_color** (*str*) – Y axis label color, default = 'black'. *Optional*
- **ylabel_size** (*str*) – Y label size, default = 'medium'. *Optional*
- **ytick_rotation** (*Optional[int] = 0,*) –
- **xlabel** (*str*) – X axis label. *Optional*
- **xlabel_color** (*str*) – X axis label color, default = 'black'. *Optional*
- **xlabel_size** (*str*) – X label size, default = 'medium'. *Optional*
- **xtick_rotation** (*Optional[int] = 0,*) –
- **title** (*str*) – Graph title, default = 'Histogram'. *Optional*
- **title_size** (*str*) – Title size, default = 'xx-large'. *Optional*
- **grid** (*bool*) – If True will show grid, default = true. *Optional*
- **grid_alpha** (*float*) – Grid alpha, default = 0.75. *Optional*
- **grid_dash_sequence** (*tuple*) – Grid dash sequence, default = (3, 3). *Optional*
- **grid_lineweight** (*float*) – Grid lineweight, default = 0.5. *Optional*
- **legend_fontsize** (*str*) – Legend fontsize, default = 'medium'. *Optional*
- **legend_transparency** (*float*) – Legend transparency, default = 0.75. *Optional*
- **legend_location** (*str*) – legend location, default = 'lower right'. *Optional*

Example *None*

Note *None*

`plot.Histogram.ax`	Returns a plot

`Table:`
Class for plotting tables.

Parameters

- **`data`** (*`pd.DataFrame`*) – Input data.
- **`label_lst`** (*`List[str]`*) – List of labels to include, if None will include all columns. *Optional*
- **`fig_size`** (*`tuple`*) – default = (10, 10), *Optional*
- **`font_size`** (*`str`*) – Font size inside cells, default = 'medium'. *Optional*
- **`col_widths`** (*`float`*) – Width of columns, default = 0.30. *Optional*
- **`row_colors`** (*`str`*) – Color of rows. *Optional*
- **`header_colors`** (*`str`*) – Header of table color. *Optional*
- **`edge_color`** (*`str`*) – Color of cell edges, default = 'w'. *Optional*
- **`sequential_cells`** (*`bool`*) – If True will color ever other row. *Optional*
- **`color_map`** (*`str`*) – Color map used in cells, default = 'Greens'. *Optional*

Example *None*

Note *None*

`plot.Table.ax`	Returns a plot

2.4 Regression

Regression class object.

`Regression:`
Calculate a linear regression.

Parameters

- **`doc_filter`** (*`DocumentFilter`*) – Input DocumentFilter.
- **`x_column`** (*`str, or List[str]`*) – Name of column or columns to be used in regression analysis.
- **`y_column`** (*`str`*) – Name of column to be used as y variable in regression.

Example

```
>>> from document_filter import DocumentFilter
>>> from regression import Regression
>>> doc = DocumentFilter(original_df=cod.our_df, map_choice='mp_e',␣
↪mode_choice='quad')
>>> model = Regression(doc_filter=doc, x_column='kills', y_column=
↪'placementPercent')
```

(continues on next page)

(continued from previous page)

Note This will return a Regression object with regression result information.

regression.Regression.r2	Returns R Squared
regression.Regression.constant_coefficient	Returns Constant Coefficient, if only one x_column is provided
regression.Regression.x_coefficient	Returns X Coefficient, if only one x_column is provided
regression.Regression.lower_confidence	Returns Lower Confidence Value, if only one x_column is provided
regression.Regression.upper_confidence	Returns Upper Confidence Value, if only one x_column is provided
regression.Regression.pvalue	Returns P Value or Values
regression.Regression.residuals	Returns residuals
regression.Regression.mse	Returns Mean Squared Error
regression.Regression.ssr	Returns Sum of Squared Residuals
regression.Regression.ess	Returns Sum of Squared Error
regression.Regression.confidence	Returns Confidence Values, if more than one x_column is provided
regression.Regression.coefficients	Returns Coefficient Values, if more than one x_column is provided

2.5 Squad

Squad class objects.

Performance:

The Performance class is used to evaluate a players performance on a given map and mode

Parameters

- **original_df** (*pd.DataFrame*) – Input data.
- **nap_choice** – Map filter. Either 'mp_e' for Rebirth and 'mp_d' for Verdansk.
- **mode_choice** (*str*) – Mode filter. Either 'solo', 'duo', 'trio', or 'quad'.
- **uno** (*str*) – Input person uno Id.

Example *None*

Note *None*

squad.Performance.map	Returns the map selected
squad.Performance.mode	Returns the mode selected
squad.Performance.stats	Returns a dict of stats

Person:

The Person class is used to gather all map/mode stats for a given player

Parameters

- **original_df** (*pd.DataFrame*) – Input data.
- **uno** (*str*) – Input person uno Id.

- **gamertag** (*str*) – Input person's gamertag.

Example *None*

Note *None*

squad.Person.gamertag	Returns player gamertag
squad.Person.uno	Returns player uno
squad.Person.rebirth	Returns a dict of all mode stats for Rebirth
squad.Person.verdansk	Returns a dict of all mode stats for Verdansk

Squad:

Calculate stats for all maps/modes for each squad memeber.

Parameters

- **squad_lst** (*List[str]*) – List of gamertags. Include your gamertag in the list.
- **original_df** (*pd.DataFrame*) – Original DataFrame for stats to be calculated from.
- **uno_name_dic** (*dict*) – A dict of all gamertags and respective unos.

Example

```
>>> from credentials import user_inputs
>>> from user import User
>>> from squad import Squad
>>> _User = User(info=user_inputs)
>>> _Squad = Squad(squad_lst=_User.squad_lst, original_df=cod.our_df,␣
↪uno_name_dic=cod.name_uno_dict)
```

Note This will calculate and return the stats for all squad members.

squad.Squad.squad_dic	Returns the dict of results
squad.Squad.squad_df	Returns the dict of results in DataFrame format

2.6 User

User class objects.

User:

Organizes the Users input data.

Parameters **info** (*dict*) – User input dict.

Example

```
>>> from user import User
>>> from credentials import user_inputs
>>> user = User(info=user_input)
```

Note *None*

user.User.file_name	Returns the file name of the users data
user.User.repo	Returns the directory location of the users data

continues on next page

Table 11 – continued from previous page

user.User.gamertag	Returns the users gamertag
user.User.squad_lst	Returns the users squad gamertags as a list

CHAPTER

THREE

UTILS

This chapter documents the Utils. Functions and plots to aid in exploratory analysis

3.1 Analysis

One off functions for various analysis.

first_top5_bottom_stats(doc_filter, col_lst):

Calculate mu, std, var, max, min, skew, kurt for all matches depending on teamPlacement. The intent is for a map_choice and mode_choice to be fed into the DocumentFilter. Does calculations for all matches, regardless of matchID.

Parameters

- **doc_filter** (*DocumentFilter*) – Input DocumentFilter.
- **col_lst** (*List[str] or str*) – Input List of Columns to analyze.

Returns Stats, related to the items in col_lst, for winners, top 5 or 10, and bottom.

Return type pd.DataFrame

Example *None*

Note If Rebirth is selected in the DocumentFilter, will return top 5. If Verdansk, top 10 is returned.

bucket_stats(doc_filter, placement, col_lst):

Calculate mu, std, var, max, min, skew, kurt for all matches depending on teamPlacement. The intent is for a map_choice and mode_choice to be fed into the DocumentFilter. Does calculations for all matches, considering of matchID.

Parameters

- **doc_filter** (*DocumentFilter*) – Input DocumentFilter.
- **placement** (*List[int] or int*) – Target placement.
- **col_lst** (*List[str] or str*) – Input List of Columns to analyze.

Returns Stats, related to the items in col_lst, for placement value.

Return type pd.DataFrame

Example *None*

Note teamPlacement value used to filter data. If two int's are provided, will filter within that range. First value should be the lower value. Example [0,6] will return top 5 placements.

previous_next_placement(doc_filter):
Calculate mu teamPlacement before and after a teamPlacement. The intent is for a map_choice and mode_choice to be fed into the DocumentFilter.

Parameters doc_filter (*DocumentFilter*) – Input DocumentFilter.

Returns Previous and next expected placement based on current placement.

Return type pd.DataFrame

Example *None*

Note *None*

match_difficulty(our_doc_filter, other_doc_filter, mu_lst, sum_lst, test):
Calculate the relative match difficulty based on player and player squad stats.

Parameters

- **our_doc_filter** (*DocumentFilter*) – A DocumentFilter with squad and player data only.
- **other_doc_filter** (*DocumentFilter*) – A DocumentFilter with all other players data.
- **mu_lst** (*List[str]*) – A list of columns to consider the mu. *Optional*
- **sum_lst** (*List[str]*) – A list of columns to consider the sum. *Optional*
- **test** (*bool*) – If True, will use all columns for the analysis. *Optional*

Returns Match difficulty.

Return type pd.DataFrame

Example *None*

Note The intent is for a map_choice and mode_choice to be fed into both DocumentFilter's.

get_daily_hourly_weekday_stats(doc_filter):
Calculate kills, deaths, wins, top 5s or 10s, match count, and averagePlacement for every day, week, hour.

Parameters doc_filter (*DocumentFilter*) – Input DocumentFilter.

Returns 3 pd.DataFrames and a dict

Return type *None*

Example *None*

Note The intent is for a map_choice and mode_choice to be fed into the DocumentFilter.

get_weapons(doc_filter):
Calculate the Kills, deaths, assists, headshots, averagePlacement and count for each weapon.

Parameters doc_filter (*DocumentFilter*) – Input DocumentFilter.

Returns A DataFrame with a players gun stats.

Return type pd.DataFrame

Example *None*

Note The intent is for a username to be fed into the DocumentFilter and this will return the information for that specific player.

find_hackers(doc_filter, y_column, col_lst, std):
Calculate hackers based on various Outlier detection methods.

Parameters

- **doc_filter** (*DocumentFilter*) – A DocumentFilter.
- **y_column** (*str*) – A column to consider for Outlier analysis.
- **col_lst** (*List[str]*) – A list of columns used for Outlier analysis.
- **std** (*int*) – The std to be considered for as a threshold, default is 3. *Optional*

Returns Returns an index of suspected hackers.

Return type List[int]

Example *None*

Note The intent is for a map_choice and mode_choice to be fed into the DocumentFilter.

meta_weapons(doc_filter, top_5_or_10, top_1):

Calculate the most popular weapons. Map_choice is required in DocumentFilter if top_5_or_10 or top_1 is True. If Neither top_5_or_10 or top_1 are True, it will calculate based on all team placements. This will only include loadouts where all attachment slots are filled. This calculates based on a daily interval.

Parameters

- **doc_filter** (*DocumentFilter*) – A DocumentFilter.
- **top_5_or_10** (*bool*) – If True, will calculate using only the top 5 or 10 place teams, default is False. *Optional*
- **top_1** (*bool*) – If True, will calculate using only the 1st place or winning team, default is False. *Optional*

Returns The First DataFrame is filled with dict's {kills: 0, deaths: 0, count: 0}. The Second is the percent of the lobby using.

Return type List[pd.DataFrame]

Example *None*

Note *None*

3.2 Base

General transformations.

normalize(arr, multi):

Normalize an Array.

Parameters

- **arr** (*np.ndarray*) – Input array.
- **multi** (*bool*) – If array has multiple columns, default is None. *Optional*

Returns Normalized array.

Return type np.ndarray

Example *None*

Note Set *multi* to True, if multiple columns.

running_mean(arr, num):

Calculate the running mean on *num* interval

Parameters

- **arr** (*np.ndarray*) – Input array.
- **num** (*int*) – Input int, default is 50. *Optional*

Returns Running mean for a given array.

Return type np.ndarray

Example *None*

Note *None*

cumulative_mean(arr):
Calculate the cumulative mean.

Parameters **arr** (*np.ndarray*) – Input array.

Returns Cumulative mean for a given array.

Return type np.ndarray

Example *None*

Note *None*

3.3 Build

These functions are used when building the CallofDuty class.

CallofDuty

3.4 Outlier

Various outlier detection functions.

stack(x_arr, y_arr, multi):
Stacks x_arr and y_arr.

Parameters

- **x_arr** (*np.ndarray*) – An array to stack.
- **y_arr** (*np.ndarray*) – An array to stack.
- **mutli** – If True, will stack based on multiple x_arr columns, default is False. *Optional*

Returns Array with a x column and a y column

Return type np.ndarray

Example *None*

Note *None*

_cent(x_lst, y_lst):
Calculate Centroid from x and y value(s).

Parameters

- **x_lst** (*List[float]*) – A list of values.
- **y_lst** (*List[float]*) – A list of values.

Returns A list of x and y values representing the centriod of two lists.

Return type List[float]

Example *None*

Note *None*

`_dis(cent1, cent2):`
Calculate Distance between two centroids.

Parameters

- **cent1** (*List[float]*) – An x, y coordinate representing a centroid.
- **cent2** – An x, y coordinate representing a centroid.

Returns A distance measurement.

Return type float

Example *None*

Note *None*

`outlier_std(arr, data, y_column, _std, plus):`
Calculate Outliers using a simple std value.

Parameters

- **arr** (*np.ndarray*) – An Array to get data from. *Optional*
- **data** (*pd.DataFrame*) – A DataFrame to get data from. *Optional*
- **y_column** (*str*) – A target column. *Optional*
- **_std** (*int*) – A std threshold, default is 3. *Optional*
- **plus** (*bool*) – If True, will grab all values above the threshold, default is True. *Optional*

Returns An array of indexes.

Return type np.ndarray

Example *None*

Note If **arr** not passed, data and respective column names are required.

`outlier_var(arr, data, y_column, per, plus):`
Calculate Outliers using a simple var value.

Parameters

- **arr** (*np.ndarray*) – An Array to get data from. *Optional*
- **data** (*pd.DataFrame*) – A DataFrame to get data from. *Optional*
- **y_column** (*str*) – A target column. *Optional*
- **per** (*float*) – A percent threshold, default is 0.95. *Optional*
- **plus** (*bool, default is True*) – If True, will grab all values above the threshold. *Optional*

Returns An array of indexes.

Return type np.ndarray

Example *None*

Note If **arr** not passed, data and respective column names are required.

`outlier_regression(arr, data, x_column, y_column, _std, plus):`
Calculate Outliers using regression.

Parameters

- **arr** (*np.ndarray*) – An Array to get data from. *Optional*
- **data** (*pd.DataFrame*) – A DataFrame to get data from. *Optional*
- **x_column** (*str*) – A column for x variables. *Optional*
- **y_column** (*str*) – A column for y variables. *Optional*
- **_std** (*int*) – A std threshold, default is 3. *Optional*
- **plus** (*bool*) – If True, will grab all values above the threshold, default is True. *Optional*

Returns An array of indexes.

Return type np.ndarray

Example *None*

Note If **arr** not passed, data and respective column names are required.

`outlier_distance(arr, data, x_column, y_column, _std, plus):`
Calculate Outliers using distance measurements.

Parameters

- **arr** (*np.ndarray*) – An Array to get data from. *Optional*
- **x_column** (*str*) – A column for x variables. *Optional*
- **y_column** (*str*) – A column for y variables. *Optional*
- **_std** (*int*) – A std threshold, default is 3. *Optional*
- **plus** (*bool*) – If True, will grab all values above the threshold, default is True. *Optional*

Param data: A DataFrame to get data from. *Optional*

Returns An array of indexes.

Return type np.ndarray

Example *None*

Note If **arr** not passed, data and respective column names are required.

`outlier_hist(arr, data, x_column, per, plus):`
Calculate Outliers using Histogram.

Parameters

- **arr** (*np.ndarray*) – An Array to get data from. *Optional*
- **x_column** (*str*) – A column for x variables. *Optional*
- **per** (*float*) – A std threshold, default is 3. *Optional*
- **plus** (*bool*) – If True, will grab all values above the threshold, default is 0.75. *Optional*

Param data: A DataFrame to get data from. *Optional*

Returns An array of indexes.

Return type np.ndarray

Example *None*

Note If **arr** not passed, data and respective column names are required.

outlier_knn(arr, data, x_column, y_column, _std, plus):
Calculate Outliers using KNN.

Parameters

- **arr** (*np.ndarray*) – An Array to get data from. *Optional*
- **x_column** (*str*) – A column for x variables. *Optional*
- **y_column** (*str*) – A column for y variables. *Optional*
- **_std** (*int*) – A std threshold, default is 3. *Optional*
- **plus** (*bool*) – If True, will grab all values above the threshold, default is True. *Optional*

Param data: A DataFrame to get data from. *Optional*

Returns An array of indexes.

Return type np.ndarray

Example *None*

Note If **arr** not passed, data and respective column names are required.

outlier_cooks_distance(arr, data, x_column, y_column, plus, return_df):
Calculate Outliers using Cooks Distance.

Parameters

- **arr** (*np.ndarray*) – An Array to get data from. *Optional*
- **data** (*pd.DataFrame*) – A DataFrame to get data from. *Optional*
- **x_column** (*str*) – A column for x variables. *Optional*
- **y_column** (*str*) – A column for y variables. *Optional*
- **_std** (*int*) – A std threshold, default is 3. *Optional*
- **plus** (*bool*) – If True, will grab all values above the threshold, default is True. *Optional*
- **return_df** (*bool*) – If True, will return a DataFrame, default is False. *Optional*

Returns An array of indexes.

Return type np.ndarray or pd.DataFrame

Example *None*

Note If **arr** not passed, data and respective column names are required.

3.5 Plots

Various one off plots.

personal_plot(doc_filter):
Returns a series of plots.

Parameters **doc_filter** (*DocumentFilter*) – A DocumentFilter.

Returns *None*

Example *None*

Note This is intended to be used with map_choice, mode_choice and a Gamertag inputted into the DocumentFilter.

`lobby_plot(doc_filter):`
Returns a series of plots.

Parameters `doc_filter` (*DocumentFilter*) – A DocumentFilter.

Returns *None*

Example *None*

Note This is intended to be used with map_choice and mode_choice inputted into the DocumentFilter.

`squad_plot(doc_filter, col_lst):`
Build a Polar plot for visualizing squad stats.

Parameters

- `doc_filter` (*DocumentFilter*) – A DocumentFilter.
- `col_lst` (*List[str] or str*) – Input List of Columns to analyze.

Returns *None*

Example *None*

Note This is intended to be used with map_choice and mode_choice inputted into the DocumentFilter.

3.6 Scrape

Functions for getting and dealing with new data.

`connect_to_api(_id: str):`
Connect to Call of Duty API.

Parameters `_id` (*str*) – A matchID str.

Returns A Json of lobby data related to specified matchID.

Return type Json

Example *None*

Note Connect to Cod API to receive lobby information.

`clean_api_data(json_object):`
Cleans the JSON output from *connect_to_api*

Parameters `json_object` (*Json*) – Json object.

Returns Match information in a table.

Return type pd.DataFrame

Example *None*

Note Takes a Json object related to a matchID and constructs a pd.DataFrame with all relevant information. This will need to be saved(or concatenated to an existing csv) and loaded through the _evaulate_df() to work properly in this model.

CHAPTER

FOUR

GLOSSARY

Terms used in this documentation.

col_lst What the Norwegien Blue does when it misses its homeland, for example, pining for the fjords.

DocumentFilter Theoretically, the sound a parrot makes when four-thousand volts of electricity pass through it.

map_choice What the Norwegien Blue does when it misses its homeland, for example, pining for the fjords.

matchId What the Norwegien Blue does when it misses its homeland, for example, pining for the fjords.

mode_choice What the Norwegien Blue does when it misses its homeland, for example, pining for the fjords.

other_df What the Norwegien Blue does when it misses its homeland, for example, pining for the fjords.

our_df What the Norwegien Blue does when it misses its homeland, for example, pining for the fjords.

teamPlacement What the Norwegien Blue does when it misses its homeland, for example, pining for the fjords.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- search