Warzone Analysis

Release v2.2.0

Peter Rigali

Aug 30, 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

class call_of_duty.**CallofDuty**(*hacker_data: bool = False*, *squad_data: bool = False*, *streamer_mode: bool = True*)

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

hacker_data [bool, default is False.] 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.

squad_data [bool, default is False.] If True, will build the Squad class.

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

This will calculate and build the CallofDuty class.

property gun_dictionary: dict
Returns a dict of gun names

property hacker_df: pandas.core.frame.DataFrame
If a hacker DataFrame is provided, will return just the hacker DataFrame

property last_match_date_time
Returns a Timestamp of the latest game in the players data. Useful when scraping from Cod Tracker

property my_uno: str
Returns the user uno value

property name_uno_dict: dict
Returns a dict of gamertags and respective unos

property name_uno_dict_hacker: dict
If a hacker DataFrame is provided, will return the gamertags: unos for the hacker DataFrame

property other_df: pandas.core.frame.DataFrame
Returns a DataFrame of all data related to other teams in a lobby

property our_df: pandas.core.frame.DataFrame
Returns a DataFrame of all data related to player and there teammates

property squad: squad.Squad
Returns a Squad class object of stats related to the user squad mates

property user: user.User
Returns a User class object of related info to the user

property whole: pandas.core.frame.DataFrame
Returns the unedited player matches DataFrame

2.2 DocumentFilter

class document_filter.**DocumentFilter**(*original_df: pandas.core.frame.DataFrame*, *map_choice: Optional[str] = None*, *mode_choice: Optional[str] = None*, *username: Optional[str] = None*, *uno: Optional[int] = None*, *username_dic: Optional[dict] = None*, *username_lst: Optional[List[str]] = None*)

Get a selection from a DataFrame.

Uses a set of filters to return a desired set of data to be used in later analysis.

original_df [pd.DataFrame] Input DataFrame to be filtered.

map_choice [str, default is None] Map filter. Either 'mp_e' for Rebirth and 'mp_d' for Verdansk.

mode_choice [str, default is None] Mode filter. Either 'solo', 'duo', 'trio', or 'quad'.

username [str, default is None] Filter by a players username. Can cause errors if same username as another player.

uno [str, default is None] Filter by a players uno.

username_dic [dict, default is None] Required if 'username' or 'username_lst' is used. {username1: uno1, username2: uno2, etc}.

username_lst [List[str], default is None] Filter using a list of usernames.

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

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.

property df: pandas.core.frame.DataFrame
Returns the filtered DataFrame

property ids: Optional[List[str]]
Returns match ids from the filtered DataFrame

property map_choice: Optional[str]
Returns the map used to filter

property mode_choice: Optional[str]
Returns the mode used to filter

property unique_ids: Optional[List[str]]
Returns unique match ids from the filtered DataFrame

property uno: Optional[str]
Returns the uno used to filter

property username: Optional[str]
Returns the username used to filter

property username_dic: Optional[dict]
Returns username: uno dict

property username_lst: Optional[List[str]]
Returns the username list used to filter

2.3 Plot

class plot.**Line**(*data: pandas.core.frame.DataFrame*, *limit: Optional[int] = None*, *label_lst: Optional[List[str]] = None*, *color_lst: Optional[List[str]] = None*, *normalize_x: Optional[List[str]] = None*, *running_mean_x: Optional[List[str]] = None*, *running_mean_value: Optional[int] = 50*, *cumulative_mean_x: Optional[List[str]] = None*, *fig_size: Optional[tuple] = (10, 7)*, *ylabel: Optional[str] = None*, *ylabel_color: Optional[str] = 'black'*, *ylabel_size: Optional[str] = 'medium'*, *xlabel: Optional[str] = None*, *xlabel_color: Optional[str] = 'black'*, *xlabel_size: Optional[str] = 'medium'*, *title: Optional[str] = 'Line Plot'*, *title_size: Optional[str] = 'xx-large'*, *grid: Optional[bool] = True*, *grid_alpha: Optional[float] = 0.75*, *grid_dash_sequence: Optional[tuple] = (3, 3)*, *grid_lineweight: Optional[float] = 0.5*, *legend_fontsize: Optional[str] = 'medium'*, *legend_transparency: Optional[float] = 0.75*, *legend_location: Optional[str] = 'lower right'*)
Creates a Line Plot

class plot.**Scatter**(*data: pandas.core.frame.DataFrame*, *limit: Optional[int] = None*, *label_lst: Optional[List[str]] = None*, *color_lst: Optional[List[str]] = None*, *normalize_x: Optional[List[str]] = None*, *regression_line: Optional[List[str]] = None*, *regression_line_color: Optional[str] = 'r'*, *regression_line_lineweight: Optional[float] = 2.0*, *running_mean_x: Optional[List[str]] = None*, *running_mean_value: Optional[int] = 50*, *cumulative_mean_x: Optional[List[str]] = None*, *fig_size: Optional[tuple] = (10, 7)*, *ylabel: Optional[str] = None*, *ylabel_color: Optional[str] = 'black'*, *ylabel_size: Optional[str] = 'medium'*, *xlabel: Optional[str] = None*, *xlabel_color: Optional[str] = 'black'*, *xlabel_size: Optional[str] = 'medium'*, *title: Optional[str] = 'Scatter Plot'*, *title_size: Optional[str] = 'xx-large'*, *grid: Optional[bool] = True*, *grid_alpha: Optional[float] = 0.75*, *grid_dash_sequence: Optional[tuple] = (3, 3)*, *grid_lineweight: Optional[float] = 0.5*, *legend_fontsize: Optional[str] = 'medium'*, *legend_transparency: Optional[float] = 0.75*, *legend_location: Optional[str] = 'lower right'*)
Creates a Scatter Plot

class plot.**Histogram**(*data: pandas.core.frame.DataFrame*, *color_lst: Optional[List[str]] = None*, *label_lst: Optional[List[str]] = None*, *limit: Optional[int] = None*, *include_norm: Optional[str] = None*, *norm_color: Optional[str] = 'r'*, *norm_lineweight: Optional[float] = 1.0*, *norm_ylabel: Optional[str] = None*, *norm_legend_location: Optional[str] = 'upper right'*, *fig_size: Optional[tuple] = (10, 7)*, *bins: Optional[str] = 'sturges'*, *hist_type: Optional[str] = 'bar'*, *stacked: Optional[bool] = False*, *ylabel: Optional[str] = None*, *ylabel_color: Optional[str] = 'black'*, *ylabel_size: Optional[str] = 'medium'*, *ytick_rotation: Optional[int] = 0*, *xlabel: Optional[str] = None*, *xlabel_color: Optional[str] = 'black'*, *xlabel_size: Optional[str] = 'medium'*, *xtick_rotation: Optional[int] = 0*, *title: Optional[str] = 'Histogram'*, *title_size: Optional[str] = 'xx-large'*, *grid: Optional[bool] = True*, *grid_alpha: Optional[float] = 0.75*, *grid_dash_sequence: Optional[tuple] = (3, 3)*, *grid_lineweight: Optional[float] = 0.5*, *legend_fontsize: Optional[str] = 'medium'*, *legend_transparency: Optional[float] = 0.75*, *legend_location: Optional[str] = 'lower right'*)
Creates a Histogram Plot

class plot.**Table**(*data: pandas.core.frame.DataFrame*, *label_lst: Optional[List[str]] = None*, *fig_size: Optional[tuple] = (10, 10)*, *font_size: Optional[str] = 'medium'*, *col_widths: Optional[float] = 0.3*, *row_colors: Optional[str] = None*, *header_colors: Optional[str] = None*, *edge_color: Optional[str] = 'w'*, *sequential_cells: Optional[bool] = None*, *color_map: Optional[str] = 'Greens'*)

Creates a Table Plot

2.4 Regression

class regression.**Regression**(*doc_filter*, *x_column*, *y_column*)

Calculate a linear regression.

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.

```
>>> from Classes.document_filter import DocumentFilter
>>> from Classes.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
↪')
```

This will return a Regression object with regression resault information.

property coefficients
Returns Coefficient Values, if more than one x_column is provided

property confidence
Returns Confidence Values, if more than one x_column is provided

property constant_coefficient
Returns Constant Coefficient, if only one x_column is provided

property ess
Returns Sum of Squared Error

property lower_confidence
Returns Lower Confidence Value, if only one x_column is provided

property mse
Returns Mean Squared Error

property pvalue
Returns P Value or Values

property r2
Returns R Squared

property residuals
Returns residuals

property ssr
Returns Sum of Squared Residuals

property upper_confidence
Returns Upper Confidence Value, if only one x_column is provided

property x_coefficient
Returns X Coefficient, if only one x_column is provided

2.5 Squad

class squad.**Performance**(*original_df*, *map_choice*, *mode_choice*, *uno*)
The Performance class is used to evaluate a players performance on a given map and mode

property map: str
Returns the map selected

property mode: str
Returns the mode selected

property stats: dict
Returns a dict of stats

class squad.**Person**(*original_df*, *uno*, *gamertag*)
The Person class is used to gather all map/mode stats for a given player

property gamertag: str
Returns player gamertag

property rebirth: dict
Returns a dict of all mode stats for Rebirth

property uno: str
Returns player uno

property verdansk
Returns a dict of all mode stats for Verdansk

class squad.**Squad**(*squad_lst*, *original_df*, *uno_name_dic*)
Calculate stats for all maps/modes for each squad memeber.

squad_lst [List[str]] List of gamertags. Include your gamertag in the list.

original_df [pd.DataFrame] Orginal DataFrame for stats to be calculated from.

uno_name_dic [dict] A dict of all gamertags and respective unos.

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

This will calculate and return the stats for all squad memebers.

property squad_df: pandas.core.frame.DataFrame
Returns the dict of results in DataFrame format

property squad_dic: Dict[str, squad.Person]
Returns the dict of results

2.6 User

class user.**User**(*info: Optional[dict] = None*)

Organizes the Users input data.

info [dict] User input dict.

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

CodTrackerID: Optional[str]
Cod Tracker ID for the user

DRIVER_PATH: Optional[str]
Driver path used for Selenium scraping

PASSWORD: Optional[str]
Password for login to Cod Tracker

USERNAME: Optional[str]
Username for login to Cod Tracker

property file_name: str
Returns the file name of the users data

property gamertag: str
Returns the users gamertag

headers: Optional[dict]
Headers from local machine

property repo: str
Returns the directory location of the users data

property squad: List[str]
Returns the users squad gamertags as a list

CHAPTER

THREE

UTILS

This chapter documents the Utils.

3.1 Base

3.2 Build

3.3 Outlier

outlier.**outlier_cooks_distance**(*arr: Optional[numpy.ndarray] = None*, *data: Optional[pandas.core.frame.DataFrame] = None*, *x_column: Optional[str] = None*, *y_column: Optional[str] = None*, *plus: Optional[bool] = True*, *return_df: Optional[bool] = False*) → Union[numpy.ndarray, pandas.core.frame.DataFrame]

Calculate Outliers using Cooks Distance.

arr [np.ndarray] A DataFrame to get data from.

data [pd.DataFrame] A DataFrame to get data from.

x_column: str A column for x variables.

y_column [str] A column for y variables.

plus [bool, default is True] If True, will grab all values above the threshold.

return_df [bool, default is False.] If True, will return a DataFrame.

np.ndarray or pd.DataFrame

outlier.**outlier_distance**(*arr: Optional[numpy.ndarray] = None*, *data: Optional[pandas.core.frame.DataFrame] = None*, *x_column: Optional[str] = None*, *y_column: Optional[str] = None*, *_std: Optional[int] = 3*, *plus: Optional[bool] = True*) → numpy.ndarray

Calculate Outliers using distance measurements.

arr [np.ndarray] An Array to get data from.

data [pd.DataFrame] A DataFrame to get data from.

x_column: str A column for x variables.

y_column [str] A column for y variables.

_std [int, default is 3.] A std threshold.

plus [bool, default is True] If True, will grab all values above the threshold.

np.ndarray

outlier.**outlier_hist**(*arr: Optional[numpy.ndarray] = None*, *data: Optional[pandas.core.frame.DataFrame] = None*, *x_column: Optional[str] = None*, *per: Optional[float] = 0.75*, *plus: Optional[bool] = True*) → numpy.ndarray

Calculate Outliers using Histogram.

arr [np.ndarray] An Array to get data from.

data [pd.DataFrame] A DataFrame to get data from.

x_column [str] A target column.

per [float, default is 0.75.] A percent threshold.

plus [bool, default is True] If True, will grab all values above the threshold.

np.ndarray

outlier.**outlier_knn**(*arr: Optional[numpy.ndarray] = None*, *data: Optional[pandas.core.frame.DataFrame] = None*, *x_column: Optional[str] = None*, *y_column: Optional[str] = None*, *_std: Optional[int] = 3*, *plus: Optional[bool] = True*) → numpy.ndarray

Calculate Outliers using KNN.

arr [np.ndarray] An array to get data from.

data [pd.DataFrame] A DataFrame to get data from.

x_column: str A column for x variables.

y_column [str] A column for y variables.

_std [int, default is 3.] A std threshold.

plus [bool, default is True] If True, will grab all values above the threshold.

np.ndarray

outlier.**outlier_regression**(*arr: Optional[numpy.ndarray] = None*, *data: Optional[pandas.core.frame.DataFrame] = None*, *x_column: Optional[str] = None*, *y_column: Optional[str] = None*, *_std: Optional[int] = 3*, *plus: Optional[bool] = True*) → numpy.ndarray

Calculate Outliers using regression.

arr [np.ndarray] An Array to get data from.

data [pd.DataFrame] A DataFrame to get data from.

x_column: str A column for x variables.

y_column [str] A column for y variables.

_std [int, default is 3.] A std threshold.

plus [bool, default is True] If True, will grab all values above the threshold.

np.ndarray

outlier.**outlier_std**(*arr: Optional[numpy.ndarray] = None*, *data: Optional[pandas.core.frame.DataFrame] = None*, *y_column: Optional[str] = None*, *_std: Optional[int] = 3*, *plus: Optional[bool] = True*) → numpy.ndarray

Calculate Outliers using a simple std value.

arr [np.ndarray] An Array to get data from.

data [pd.DataFrame] A DataFrame to get data from.

y_column [str] A target column.

_std [int, default is 3.] A std threshold.

plus [bool, default is True] If True, will grab all values above the threshold.

np.ndarray

outlier.**outlier_var**(*arr: Optional[numpy.ndarray] = None*, *data: Optional[pandas.core.frame.DataFrame] = None*, *y_column: Optional[str] = None*, *per: Optional[float] = 0.95*, *plus: Optional[bool] = True*) → numpy.ndarray

Calculate Outliers using a simple var value.

arr [np.ndarray] An Array to get data from.

data [pd.DataFrame] A DataFrame to get data from.

y_column [str] A target column.

per [float, default is 0.95.] A percent threshold.

plus [bool, default is True] If True, will grab all values above the threshold.

np.ndarray

3.4 Plots

3.5 Scrape

scrape.**clean_api_data**(*json_object*) → pandas.core.frame.DataFrame

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.

json_object [Json] A lobby json.

pd.DataFrame.

scrape.**connect_to_api**(*_id: str*)

Connect to Cod API to receive lobby information.

_id [str] A matchID str.

Json

Returns a Json of lobby data related to specified matchID.

CHAPTER

FOUR

GLOSSARY

Terms used in this documentation.

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

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

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX