poker-now-analysis

Release 1.0.1

Peter Rigali

Aug 08, 2023

CONTENTS:

CHAPTER

ONE

INTRO

This is a package for analyzing past performance and player habits from the Poker Now website.

1.1 Usage

In progress…

```
import poker
from poker.poker_class import Poker

# Input past Data folder location.
repo = '\\location of past Data folder'

# Manual grouping. When same players, play from different devices, they will get a␣
↪different unique ID.
# This will group the players.
grouped = [['YEtsj6CMK4', 'M_ODMJ-3Je', 'DZy-22KNBS'],
           ['1_FRcDzJU-', 'ofZ3AjBJdl', 'yUaYOqMtWh', 'EIxKLHzvif'],
           ['3fuMmmzEQ-', 'LRdO6bTCRh', '9fNOKzXJkb'],
           ['FZayb4wOU1', '66rXA9g5yF', 'rM6qlbc77h', 'fy6-0HLhb_'],
           ['48QVRRsiae', 'u8_FUbXpAz'],
           ['Aeydg8fuEg', 'yoohsUunIZ'],
           ['mUwL4cyOAC', 'zGv-6DI_aJ'],
           ]

poker = Poker(repo_location=repo, grouped=grouped)
```

1.2 More Info

In progress…

CHAPTER

TWO

CLASSES

This chapter documents the Classes used in this package.

2.1 Poker

Class object for running the package.

Poker(repo_location, grouped, money_multi):

Calculate stats for all games and players.

Parameters

- **repo_location** (*str*) – Location of data folder.
- **grouped** (*str*) – List of lists, filled with unique player Ids that are related to the same person. *Optional*
- **money_multi** (*int*) – Multiple to divide the money amounts to translate them to dollars *Optional*

Example

```
from poker.poker_class import Poker
repo = 'location of your previous game'
grouped = [['YEtsj6CMK4', 'M_ODMJ-3Je', 'DZy-22KNBS'],
            ['48QVRRsiae', 'u8_FUbXpAz']]
poker = Poker(repo_location=repo, grouped=grouped)
```

Note

Grouped will need to be figured out by the player. The grouped stats are only taken into account within this class

Poker Attributes:

Name	Type	Description
Poker.files	List[str]	Returns list of data files
Poker.matches	dict	Returns list of data files
Poker.players_money_overview	pd.DataFrame	Returns summary info for each player across games
Poker.card_distribution	pd.DataFrame	Returns count and percent for each card that showed up across games
Poker.winning_hand_distribution	pd.DataFrame	Returns count and percent of each type of winning hand across games
Poker.players_history	dict	Collects player stats for all matches and groups based on grouper input

2.2 DocumentFilter

Class for getting data from Poker.

The class_lst can have any of the following:

- Requests
- Approved
- Joined
- MyCards
- SmallBlind
- BigBlind
- Folds
- Calls
- Raises
- Checks
- Wins
- Shows
- Quits
- Flop
- Turn
- River
- Undealt
- StandsUp
- SitsIn
- PlayerStacks

The position_lst can have any of the following:

- 'Pre Flop'
- 'Post Flop'
- 'Post Turn'
- 'Post River'

The win_loss_all can be one of the following:

- 'Win'
- 'Loss'
- 'All'

The column_lst can have any of the following:

- 'All In'
- 'Bet Amount'

- 'Class'
- 'End Time'
- 'From Person'
- 'Game Id'
- 'Player Current Chips'
- 'Player Index'
- 'Player Name'
- 'Player Starting Chips'
- 'Position'
- 'Pot Size'
- 'Previous Time'
- 'Remaining Players'
- 'Round'
- 'Start Time'
- 'Time'
- 'Win'
- 'Win Hand'
- 'Win Stack'
- 'Winner'

DocumentFilter(data):

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

Parameters

- **data** (*Poker*) – Input Poker object to be filtered.
- **game_id_lst** (*Union[List[str], str, None]*) – Game Id filter, default is None. *Optional*
- **player_index_lst** (*Union[List[str], str, None]*) – Player Index filter, default is None. *Optional*
- **player_name_lst** (*Union[List[str], str, None]*) – Player Name filter, default is None. *Optional*
- **class_lst** (*Union[List[str], str, None]*) – Filter by class objects, default is None. *Optional*
- **position_lst** (*Union[List[str], str, None]*) – Filter by position, default is None. *Optional*
- **win_loss_all** (*Union[str, None]*) – Filter by Win, Loss or All, default is None. *Optional*
- **column_lst** (*Union[List[str], str, None]*) – Filter by column name, default is None. *Optional*

Example
None

Note
All inputs, except data, are *Optional* and defaults are set to None. Any str inputs are placed in a list.

DocumentFilter Attributes:

Name	Type	Description
DocumentFilter.df	pd.DataFrame	Returns a DataFrame of requested items
DocumentFilter.game_id_lst	Union[List[str], None]	Returns game id input
DocumentFilter.player_index_lst	Union[List[str], None]	Returns player index input
DocumentFilter.player_name_lst	Union[List[str], None]	Returns player name input
DocumentFilter.class_lst	Union[List[str], None]	Returns class input
DocumentFilter.position_lst	Union[List[str], None]	Returns position input
DocumentFilter.win_loss_all	Union[List[str], None]	Returns win loss or all input
DocumentFilter.column_lst	Union[List[str], None]	Returns column input

2.3 Game

Class object used for getting specific game stats.

`Game(hand_lst, file_id, players_data):`

Calculate stats for a game.

Parameters

- **`hand_lst`** (*`List[dict]`*) – List of dict's from the csv.
- **`file_id`** (*`str`*) – Name of file.
- **`players_data`** (*`dict`*) – A dict of player data.

Example
None

Note
This class is intended to be used internally.

Game Attributes:

Name	Type	Description
Game.file_name	str	Returns name of data file
Game.hands_lst	List[Hand]	Returns list of hands in the game
Game.card_distribution	dict	Returns count of each card that showed up
Game.winning_hand_distribution	dict	Returns count of winning hands
Game.players_data	dict	Returns Player stats for players across hands
Game.game_stats	dict	Returns Mean stats for Game across hands

2.4 Player

Class object used for getting specific player stats.

`Player(player_index):`

Calculate stats for a player.

Parameters

`player_index` (*`str or List[str]`*) – A unique player ID.

Example

None

Note

This class is intended to be used internally.

Player Attributes:

Name	Type	Description
Player.win_percent	dict	Returns player win percent
Player.win_count	dict	Returns player win count
Player.largest_win	dict	Returns players largest win
Player.largest_loss	dict	Returns players largest loss
Player.hand_count	dict	Returns total hand count when player involved
Player.all_in	dict	Returns a dict documenting when the player went all in
Player.player_index	List[str]	Returns player index or indexes
Player.player_name	List[str]	Returns player name or names
Player.player_money_info	dict	Returns a dict of DataFrames documenting player buy-in and loss counts
Player.hand_dic	dict	Returns a dict of DataFrames documenting hands when the player won
Player.card_dic	dict	Returns a dict of DataFrames documenting card appearances
Player.line_dic	dict	Returns a dict with a list of objects where player involved
Player.moves_dic	dict	Returns a players moves on the table
Player.merged_moves	dict	Returns a combined dict of player moves

2.5 Hand

Class object used for getting specific hand stats.

`Hand(lst_hand_objects, file_id, player_dic):`

Organizes a hand with a class and adds the stands to the player_dic.

Parameters

- **`lst_hand_objects`** (*`list`*) – A list of Class Objects connected to a hand.
- **`file_id`** (*`str`*) – Unique file name.
- **`player_dic`** (*`dict`*) – Dict of players.

Example

None

Note

This class is intended to be used internally.

Hand Attributes:

Name	Type	Description
Hand.merged_moves	list	Returns a list of actions as objects
Hand.small_blind	SmallBlind	Returns SmallBlind Class
Hand.big_blind	BigBlind	Returns BigBlind Class
Hand.winner	Wins	Returns Wins Class or list of Wins Classes
Hand.starting_players	dict	Returns dict of name and ID for each player that was present at the hand start
Hand.starting_players_chips	dict	Returns dict of name and stack amount for each player that was present at the hand start
Hand.flop_cards	Flop	Returns Flop Class
Hand.turn_card	Turn	Returns Turn Class
Hand.river_card	River	Returns River Class
Hand.my_cards	MyCards	Returns MyCards Class
Hand.chips_on_board	int	Returns the count of chips on the table
Hand.gini_coef	float	Returns the gini coef for the board
Hand.pot_size_lst	List[int]	Returns pot size over course of hand
Hand.players	dict	Returns dict of player moves
Hand.start_time	TimeStamp	Returns time of first hand item
Hand.end_time	TimeStamp	Returns time of last hand item
Hand.win_stack	Union[int, None]	Returns win amount for the hand
Hand.bet_lst	List[int]	Returns Raise amounts for the hand

2.6 Processor

Class object for holding information from lines.

The following child classes use this framework:

- Requests
- Approved
- Joined
- MyCards
- SmallBlind
- BigBlind
- Folds
- Calls
- Raises
- Checks
- Wins
- Shows
- Quits
- Flop

- Turn
- River
- Undealt
- StandsUp
- SitsIn
- PlayerStacks

`LineAttributes:`

Applies attributes to a respective Class object.

Parameters
`text` (*`str`*) – A line of text from the data.

Example
None

Note
This class is intended to be used internally. All values are set to None or 0 by default.

LineAttributes Attributes:

Name	Type	Description
LineAttributes.text	Union[str, None]	Text input
LineAttributes.player_name	Union[str, None]	Player Name
LineAttributes.player_index	Union[str, None]	Player Id
LineAttributes.stack	Union[int, None]	Amount offered to the table
LineAttributes.position	Union[str, None]	Position of move in relation to table cards being drawn
LineAttributes.winning_hand	Union[str, None]	Winning hand
LineAttributes.cards	Union[str, list, None]	Card or cards
LineAttributes.current_round	Union[int, None]	Round number within the game
LineAttributes.pot_size	Union[int, None]	Size of pot when move happens
LineAttributes.remaining_players	Union[List[str], None]	Players left in hand
LineAttributes.action_from_player	Union[str, None]	Who bet previously
LineAttributes.action_amount	Union[int, None]	Previous bet amount
LineAttributes.all_in	Union[bool, None]	Notes if player when all-in
LineAttributes.game_id	Union[str, None]	File name
LineAttributes.starting_chips	Union[int, None]	Player's chip count at start of hand
LineAttributes.current_chips	Union[int, None]	Player's chip count at time of move
LineAttributes.winner	Union[str, None]	Player Name who wins the hand
LineAttributes.win_stack	Union[int, None]	Amount won at end of hand
LineAttributes.time	TimeStamp	Timestamp of action
LineAttributes.previous_time	TimeStamp	Timestamp of previous action
LineAttributes.start_time	TimeStamp	Timestamp of the start of the hand
LineAttributes.end_time	TimeStamp	Timestamp of the end of the hand

2.7 TSanalysis

Class for Time Series Analysis. The ts_analysis function in Analysis does not compute running values.

TSanalysis:

Calculate Time Series stats for a player.

Parameters

- **data** (*DocumentFilter*) – Input DocumentFilter.
- **upper_q** (*float*) – Upper Quantile percent, default is 0.841. *Optional*
- **lower_q** (*float*) – Lower Quantile percent, default is 0.159. *Optional*
- **window** (*int*) – Rolling window, default is 5. *Optional*

Example

```
>>> from poker.time_series_class import TSanalysis
>>> docu_filter = DocumentFilter(data=poker, player_index_lst=['DZy-
↪22KNBS'])
>>> TSanalysis(data=docu_filter)
```

Note

This class expects a DocumentFilter with only one player_index used.

TSanalysis Attributes:

Name	Type	Description
TSanalysis.ts_hand	pd.DataFrame	Hand Related base data
TSanalysis.ts_hand_mean	pd.DataFrame	Hand Related mean data
TSanalysis.ts_hand_std	pd.DataFrame	Hand Related std data
TSanalysis.ts_hand_median	pd.DataFrame	Hand Related median data
TSanalysis.ts_hand_upper_quantile	pd.DataFrame	Hand Related upper quantile data
TSanalysis.ts_hand_lower_quantile	pd.DataFrame	Hand Related lower quantile data
TSanalysis.ts_position	pd.DataFrame	Position Related base data
TSanalysis.ts_position_mean	pd.DataFrame	Position Related mean data
TSanalysis.ts_position_std	pd.DataFrame	Position Related std data
TSanalysis.ts_position_median	pd.DataFrame	Position Related median data
TSanalysis.ts_position_upper_quantile	pd.DataFrame	Position Related upper quantile data
TSanalysis.ts_position_lower_quantile	pd.DataFrame	Position Related lower quantile data
TSanalysis.ts_class	pd.DataFrame	Class Related base data
TSanalysis.ts_class_mean	pd.DataFrame	Class Related mean data
TSanalysis.ts_class_std	pd.DataFrame	Class Related std data
TSanalysis.ts_class_median	pd.DataFrame	Class Related median data
TSanalysis.ts_class_upper_quantile	pd.DataFrame	Class Related upper quantile data
TSanalysis.ts_class_lower_quantile	pd.DataFrame	Class Related lower quantile data

2.8 Plot Classes

Plot Class objects.

Possible Font Size Strings:

- 'xx-small'
- 'x-small'
- 'small'
- 'medium'
- 'large'
- 'x-large'
- 'xx-large'

Possible Legend Locations:

- 'best'
- 'upper right'
- 'upper left'
- 'lower left'
- 'lower right'
- 'right'
- 'center left'
- 'center right'
- 'lower center'
- 'upper center'
- 'center

2.8.1 Line

`Line(data)`:

Class for Line plots.

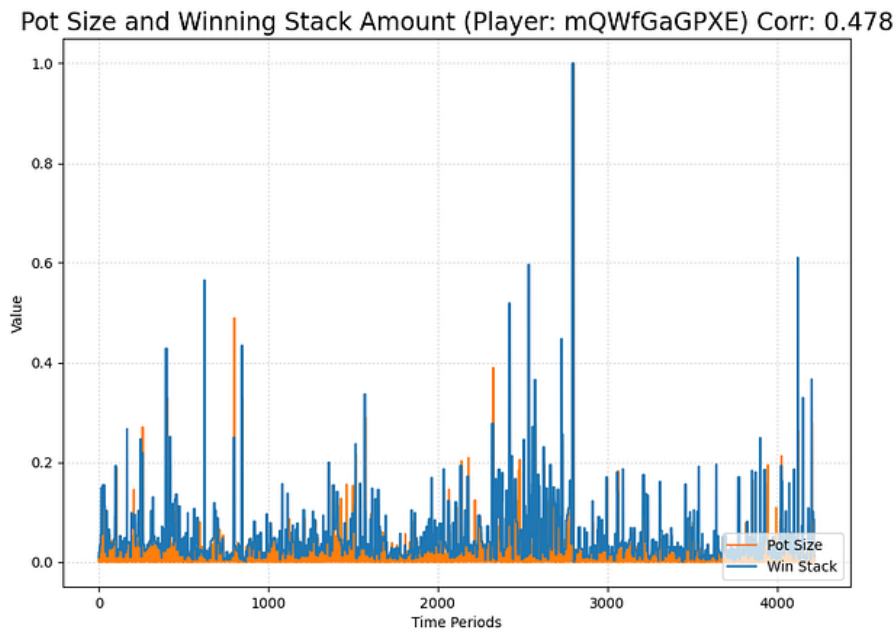
Parameters

- **`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, needs to be same length as label_lst. *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*
- **corr** (*List[str]*) – Pass two strings to return the correlation. *Optional*

Example

```
from poker.plot import Line
Line(data=val[['Pot Size', 'Win Stack']],
     normalize_x=['Pot Size', 'Win Stack'],
     color_lst=['tab:orange', 'tab:blue'],
     title='Pot Size and Winning Stack Amount (Player: ' + key + ')',
     ylabel='Value',
     xlabel='Time Periods',
     corr=['Pot Size', 'Win Stack'])
plt.show()
```



Note

None

Line Attributes:

Name	Type	Description
Line.ax	plt	Returns a Line Plot

2.8.2 Scatter

`Scatter(data)`:

Class for Scatter plots.

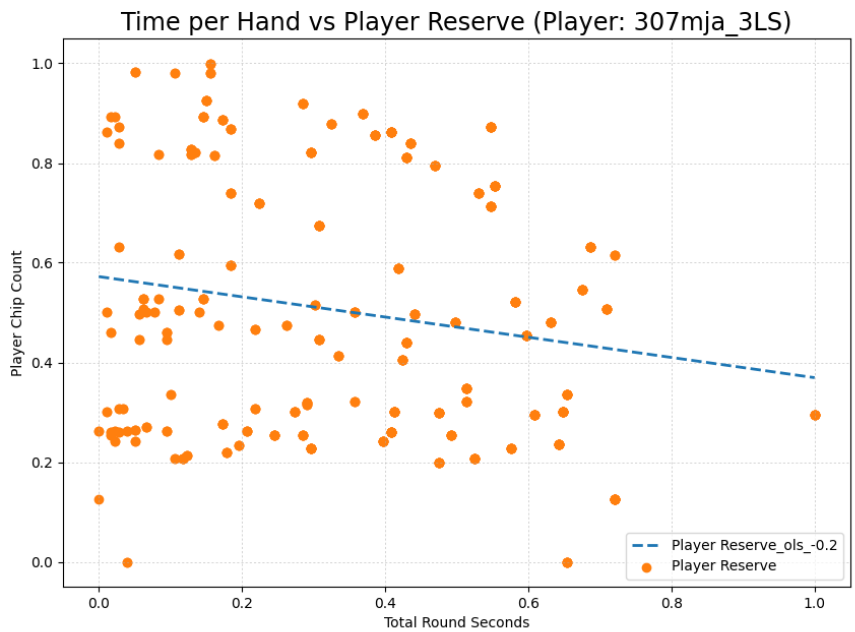
Parameters

- **`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*
- **compare_two** (*List[str]*) – If given will return a scatter comparing two variables, default is None. *Optional*
- **y_limit** (*float*) – If given will limit the y axis, default is None. *Optional*

Example

```
from poker.plot import Scatter
Scatter(data=val,
        compare_two=['Round Seconds', 'Player Reserve'],
        normalize_x=['Round Seconds', 'Player Reserve'],
        color_lst=['tab:orange'],
        regression_line=['Player Reserve'],
        regression_line_color='tab:blue',
        title='Time per Hand vs Player Reserve (Player: ' + key + ')',
        ylabel='Player Chip Count',
        xlabel='Total Round Seconds')
plt.show()
```



Note

Slope of the regression line is noted in he legend.

Scatter Attributes:

Name	Type	Description
Scatter.ax	plt	Returns a Scatter Plot

2.8.3 Histogram

`Histogram(data):`

Class for Histogram plots.

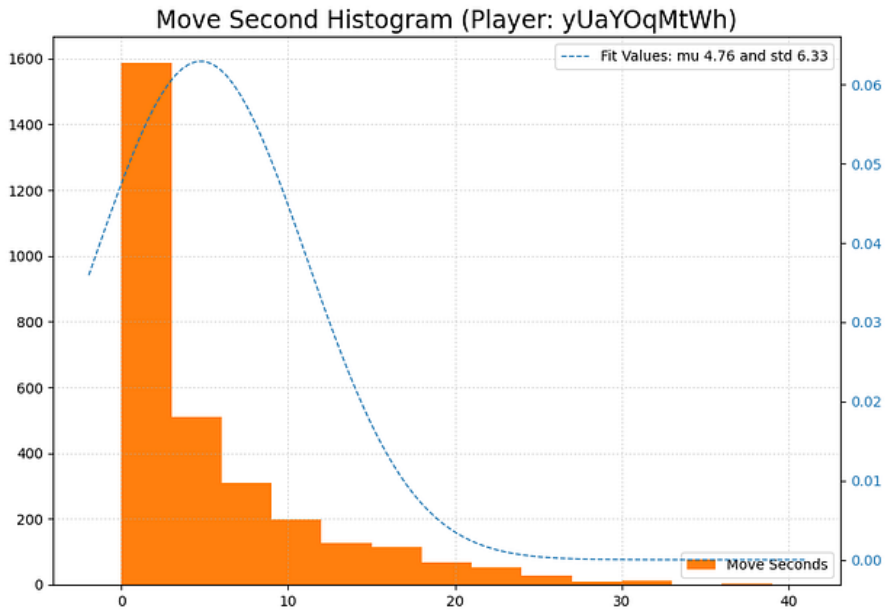
Parameters

- **`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*
- **`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

```
from poker.plot import Histogram
Histogram(data=val,
          label_lst=['Move Seconds'],
          include_norm='Move Seconds',
          title='Move Second Histogram (Player: ' + key + ')')
plt.show()
```



Note

None

Histogram Attributes:

Name	Type	Description
Histogram.ax	plt	Returns a Histogram Plot

CHAPTER

THREE

FUNCTIONS

This chapter documents the Functions used in this package.

3.1 Analysis

One off functions for various analysis. Almost all Analysis functions take either a Game or Player Class Object.

face_card_in_winning_cards(data):

Find what percent of the time a face card is used to win.

Parameters
data (*DocumentFilter*) – Input data.

Returns
A dict of file_id and face card in winning hand percent.

Return type
dict

Example

```
# This function requires Player Stacks and Wins to be included in the␣
↪DocumentFilter.
from poker.poker_class import Poker
from poker.analysis import face_card_in_winning_cards
from poker.document_filter_class import DocumentFilter
repo = 'location of your previous game'
grouped = [['YEtsj6CMK4', 'M_ODMJ-3Je', 'DZy-22KNBS'],
           ['48QVRRsiae', 'u8_FUbXpAz']]
poker = Poker(repo_location=repo, grouped=grouped)
face_card_in_winning_cards(data=DocumentFilter(data=poker, class_lst=[
↪'Player Stacks', 'Wins']))
```

Note
Percent of all Winning Cards = Total all cards and get percent that include a face card. Percent one face in Winning Cards = Percent of all wins hand at least a single face card.

longest_streak(data):

Find the longest winning streak.

Parameters
data (*DocumentFilter*) – Input data.

Returns
Longest streak.

Return type
pd.DataFrame

Example

```
# This function requires Wins to be included in the DocumentFilter.
from poker.poker_class import Poker
from poker.analysis import longest_streak
from poker.document_filter_class import DocumentFilter
repo = 'location of your previous game'
grouped = [['YEtsj6CMK4', 'M_ODMJ-3Je', 'DZy-22KNBS'],
            ['48QVRRsiae', 'u8_FUbXpAz']]
poker = Poker(repo_location=repo, grouped=grouped)
longest_streak(data=DocumentFilter(data=poker, class_lst=['Wins']))
```

Note
DocumentFilter requires class_lst=['Wins']

raise_signal_winning(data):

When a player raises, does that mean they are going to win(?).

Parameters
data (*DocumentFilter*) – Input data.

Returns
A pd.DataFrame with the percent related to each position.

Return type
pd.DataFrame

Example

```
# This function requires Raises to be included in the DocumentFilter.
from poker.poker_class import Poker
from poker.analysis import raise_signal_winning
from poker.document_filter_class import DocumentFilter
repo = 'location of your previous game'
grouped = [['YEtsj6CMK4', 'M_ODMJ-3Je', 'DZy-22KNBS'],
            ['48QVRRsiae', 'u8_FUbXpAz']]
poker = Poker(repo_location=repo, grouped=grouped)
raise_signal_winning(data=DocumentFilter(data=poker, class_lst=['Raises
↪'']))
```

Note
DocumentFilter requires class_lst=['Raises']

small_or_big_blind_win(data):

When a player is small or big blind, does that mean they are going to win(?).

Parameters
data (*DocumentFilter*) – Input data.

Returns
A pd.DataFrame with the percent related to each blind.

Return type
pd.DataFrame

Example

```
# This function requires Small Blind and Big Blind to be included in␣
↪the DocumentFilter.
from poker.poker_class import Poker
from poker.analysis import small_or_big_blind_win
from poker.document_filter_class import DocumentFilter
repo = 'location of your previous game'
grouped = [['YEtsj6CMK4', 'M_ODMJ-3Je', 'DZy-22KNBS'],
            ['48QVRRsiae', 'u8_FUbXpAz']]
poker = Poker(repo_location=repo, grouped=grouped)
small_or_big_blind_win(data=DocumentFilter(data=poker, class_lst=[
↪'Small Blind', 'Big Blind']))
```

Note

DocumentFilter requires class_lst=['Small Blind', 'Big Blind']

player_verse_player(data):

Find how many times and what value a player called or folded related all other players.

Parameters

data (*DocumentFilter*) – Input data.

Returns

A dict of counts and values for each 'Calls', 'Raises', 'Checks', and 'Folds'.

Return type

dict

Example

```
# This function requires 'Calls', 'Raises', 'Checks', and 'Folds' to be␣
↪included in the DocumentFilter.
from poker.poker_class import Poker
from poker.analysis import player_verse_player
from poker.document_filter_class import DocumentFilter
repo = 'location of your previous game'
grouped = [['YEtsj6CMK4', 'M_ODMJ-3Je', 'DZy-22KNBS'],
            ['48QVRRsiae', 'u8_FUbXpAz']]
poker = Poker(repo_location=repo, grouped=grouped)
player_verse_player(data=DocumentFilter(data=poker, class_lst=['Calls',
↪'Raises', 'Checks', 'Folds']))
```

Note

DocumentFilter requires class_lst=['Calls', 'Raises', 'Checks', 'Folds']

bluff_study(data, position_lst):

Compare betting habits when a player is bluffing.

Parameters

- **data** (*DocumentFilter*) – Input data.
- **position_lst** (*Union[List[str], str]*) –

Returns

A pd.DataFrame of counts and values for each position.

Return type

pd.DataFrame

Example

```
# This function requires a single player_index to be included in the␣
↪DocumentFilter.
from poker.poker_class import Poker
from poker.analysis import bluff_study
from poker.document_filter_class import DocumentFilter
repo = 'location of your previous game'
grouped = [['YEtsj6CMK4', 'M_ODMJ-3Je', 'DZy-22KNBS'],
            ['48QVRRsiae', 'u8_FUbXpAz']]
poker = Poker(repo_location=repo, grouped=grouped)
bluff_study(data=DocumentFilter(data=poker, player_index_lst=['DZy-
↪22KNBS']))
```

Note

This function requires a single player_index to be included in the DocumentFilter.

static_analysis(data):

Build a static analysis DataFrame.

Parameters

data (*DocumentFilter*) – Input data.

Returns

A dict of stats.

Return type

dict

Example

```
# This function requires a single player_index to be included in the␣
↪DocumentFilter.
from poker.poker_class import Poker
from poker.analysis import static_analysis
from poker.document_filter_class import DocumentFilter
repo = 'location of your previous game'
grouped = [['YEtsj6CMK4', 'M_ODMJ-3Je', 'DZy-22KNBS'],
            ['48QVRRsiae', 'u8_FUbXpAz']]
poker = Poker(repo_location=repo, grouped=grouped)
static_analysis(data=DocumentFilter(data=poker, player_index_lst=['DZy-
↪22KNBS']))
```

Note

This function requires a single player_index to be included in the DocumentFilter.

pressure_or_hold(data, bet, position):

Check how a player has responded to a bet in the past.

Parameters

- **data** (*DocumentFilter*) – Input data.
- **position** (*str*) – Location in the hand, default is None. *Optional*

Paran bet

Proposed bet amount.

Returns

A dict of Call Counts, Fold Counts, Total Count, and Call Percent.

Return type
dict

Example

```
# This function requires a single player_index to be included in the␣
↪DocumentFilter.
from poker.poker_class import Poker
from poker.analysis import pressure_or_hold
from poker.document_filter_class import DocumentFilter
repo = 'location of your previous game'
grouped = [['YEtsj6CMK4', 'M_ODMJ-3Je', 'DZy-22KNBS'],
            ['48QVRRsiae', 'u8_FUbXpAz']]
poker = Poker(repo_location=repo, grouped=grouped)
pressure_or_hold(data=DocumentFilter(data=poker, player_index_lst=['DZy-
↪22KNBS']), bet=500, position='Pre Flop')
```

Note
None

ts_analysis(data, window):

Build a Time Series DataFrame.

Parameters

- **data** (*DocumentFilter*) – A Player class object.
- **window** (*int*) – Rolling window value, default is 5. *Optional*

Returns
A DataFrame of various moves over time.

Return type
pd.DataFrame

Example

```
# This function requires a single player_index to be included in the␣
↪DocumentFilter.
from poker.poker_class import Poker
from poker.analysis import ts_analysis
from poker.document_filter_class import DocumentFilter
repo = 'location of your previous game'
grouped = [['YEtsj6CMK4', 'M_ODMJ-3Je', 'DZy-22KNBS'],
            ['48QVRRsiae', 'u8_FUbXpAz']]
poker = Poker(repo_location=repo, grouped=grouped)
ts_analysis(data=DocumentFilter(data=poker, player_index_lst=['DZy-
↪22KNBS']))
```

Note
This is a function version of the TSanalysis class.

3.2 Base

One off functions for helping analysis. These are helper functions that were constructed to limit the reliance on other packages. Most take a list, np.ndarray, or pd.Series and return a list of floats or ints.

normalize(data, keep_nan):

Normalize a list between 0 and 1.

Parameters

- **data** (*list, np.ndarray, or pd.Series*) – Input data to normalize.
- **keep_nan** (*bool*) – If True, will maintain nan values, default is False. *Optional*

Returns
Normalized list.

Return type
list

Example

```
from poker.base import normalize
data = [1, 2, 3, None, np.nan, 4]
# keep_nan set to False (default)
test = normalize(data, keep_nan=False) # [0.0, 0.3333333333333333, 0.
↪6666666666666666, 1.0]
# keep_nan set to True
test = normalize(data, keep_nan=False) # [0.0, 0.3333333333333333, 0.
↪6666666666666666, nan, nan, 1.0]
```

Note
If an int or float is passed for keep_nan, that value will be placed where nan's are present.

standardize(data, keep_nan):

Standardize a list with a mean of zero and std of 1.

Parameters

- **data** (*list, np.ndarray, or pd.Series*) – Input data to standardize.
- **keep_nan** (*bool*) – If True, will maintain nan values, default is False. *Optional*

Returns
Standardized list.

Return type
list

Example
None

Note
If an int or float is passed for keep_nan, that value will be placed where nan's are present.

running_mean(data, num):

Calculate the Running Std on *num* interval.

Parameters

- **data** (*list, np.ndarray, or pd.Series*) – Input data.

- **num** (*int*) – Input val used for Running Std window.

Returns
Running mean for a given np.ndarray, pd.Series, or list.

Return type
List[float]

Example

```
from poker.base import running_mean
data = [1, 2, 3, None, np.nan, 4]
test = running_mean(data=data, num=2) # [1.5, 1.5, 1.5, 2.5, 2.75, 2.5]
```

Note
None and np.nan values are replaced with the mean value.

running_std(data, num):

Calculate the Running Std on *num* interval.

Parameters

- **data** (*list, np.ndarray, or pd.Series*) – Input data.
- **num** (*int*) – Input val used for Running Std window.

Returns
Running mean for a given np.ndarray, pd.Series, or list.

Return type
List[float]

Example

```
from poker.base import running_std
data = [1, 2, 3, None, np.nan, 4]
test = running_std(data=data, num=2) # [0.7071067811865476, 0.
↪7071067811865476, 0.7071067811865476,
                                        #  0.7071067811865476, 0.
↪3535533905932738, 0.0]
```

Note
None and np.nan values are replaced with the mean value.

running_median(data, num):

Calculate the Running Median on *num* interval.

Parameters

- **data** (*list, np.ndarray, or pd.Series*) – Input data.
- **num** (*int*) – Input val used for Running median window.

Returns
list.

Return type
List[float]

Example
None

Note
None and np.nan values are replaced with the mean value.

running_percentile(data, num):

Calculate the Running Percentile on *num* interval.

Parameters

- **data** (*list, np.ndarray, or pd.Series*) – Input data.
- **num** (*int*) – Input val used for Running Percentile window.
- **q** (*float*) – Percent of data.

Returns
Running percentile for a given np.ndarray, pd.Series, or list.

Return type
List[float]

Example
None

Note
None and np.nan values are replaced with the mean value.

cumulative_mean(data):

Calculate the Cumulative Mean.

Parameters
data (*list, np.ndarray, or pd.Series*) – Input data.

Returns
Cumulative mean for a given np.ndarray, pd.Series, or list.

Return type
List[float]

Example

```
from poker.base import cumulative_mean
data = [1, 2, 3, None, np.nan, 4]
test = cumulative_mean(data=data) # [0.0, 1.0, 1.5, 2.0, 2.125, 2.2]
```

Note
None and np.nan values are replaced with the mean value.

round_to(data, val, remainder):

Rounds an np.array, pd.Series, or list of values to the nearest value.

Parameters

- **data** (*list, np.ndarray, pd.Series, int, float, or any of the numpy int/float variations*) – Input data.
- **val** (*int*) – Value to round to. If decimal, will be that number divided by.
- **remainder** (*bool*) – If True, will round the decimal, default is False. *Optional*

Returns
Rounded number.

Return type
List[float] or float

Example

```
# With remainder set to True.
lst = [4.3, 5.6]
round_to(data=lst, val=4, remainder=True) # [4.25, 5.5]

# With remainder set to False.
lst = [4.3, 5.6]
round_to(data=lst, val=4, remainder=False) # [4, 4]
```

Note
Single int or float values can be passed.

calc_gini(data):

Calculate the Gini Coef for a list.

Parameters
data (*list, np.ndarray, or pd.Series*) – Input data.

Returns
Gini value.

Return type
float

Example

```
>>> lst = [4.3, 5.6]
>>> calc_gini(data=lst, val=4, remainder=True) # 0.05445544554455435
```

Note
The larger the gini coef, the more consolidated the chips on the table are to one person.

search_dic_values(dic, item):

Searches a dict using the values.

Parameters

- **dic** (*dict*) – Input data.
- **item** (*str, float or int*) – Search item.

Returns
Key value connected to the value.

Return type
str, float or int

Example
None

Note
None

flatten(data, type_used):

Flattens a list of lists and checks the list.

Parameters

- **data** (*list*) – Input data.
- **type_used** (*str*) – Type to search for, default is "str". *Optional*

- **type_used** – Either {str, int, or float}

Returns
Returns a flattened list.

Return type
list

Example
None

Note
Will work when lists are mixed with non-list items.

native_mode(data):

Calculate Mode of a list.

Parameters
data (*list, np.ndarray, or pd.Series*) – Input data.

Returns
Returns the Mode.

Return type
float

Example
None

Note
None

native_median(data):

Calculate Median of a list.

Parameters
data (*list, np.ndarray, or pd.Series*) – Input data.

Returns
Returns the Median.

Return type
float

Example
None

Note
If multiple values have the same count, will return the mean. Median is used if there is an odd number of same count values.

native_mean(data):

Calculate Mean of a list.

Parameters
data (*list, np.ndarray, or pd.Series*) – Input data.

Returns
Returns the mean.

Return type
float

Example
None

Note
None

`native_variance(data, ddof):`

Calculate Variance of a list.

Parameters

- **data** (*list, np.ndarray, or pd.Series*) – Input data.
- **ddof** (*int*) – Set the degrees of freedom, default is 1. *Optional*

Returns
Returns the Variance.

Return type
float

Example
None

Note
None

`native_std(data, ddof):`

Calculate Standard Deviation of a list.

Parameters

- **data** (*list, np.ndarray, or pd.Series*) – Input data.
- **ddof** (*int*) – Set the degrees of freedom, default is 1. *Optional*

Returns
Returns the Standard Deviation.

Return type
float

Example
None

Note
None

`native_sum(data):`

Calculate Sum of a list.

Parameters
data (*list, np.ndarray, or pd.Series*) – Input data.

Returns
Returns the Sum.

Return type
float

Example
None

Note
None

native_max(data):

Calculate Max of a list.

Parameters
data (*list, np.ndarray, or pd.Series*) – Input data.

Returns
Returns the max value.

Return type
float

Example
None

Note
None

unique_values(data, count, order, indexes, keep_nan):

Get Unique values from a list.

Parameters

- **data** (*list, np.ndarray, or pd.Series*) – Input data.
- **count** (*bool*) – Return a dictionary with item and count, default is None. *Optional*
- **order** (*bool*) – If True will maintain the order, default is None. *Optional*
- **indexes** (*bool*) – If True will return index of all similar values, default is None. *Optional*
- **keep_nan** (*bool*) – If True will keep np.nan and None values, converting them to None, default is False. *Optional*

Returns
Returns either a list of unique values or a dict of unique values with counts.

Return type
Union[list, dict]

Example

```
from poker.base import unique_values
data = [1, 2, 3, None, np.nan, 4]
# count set to False (default)
test = normalize(data, keep_nan=False) # [1, 2, 3, 4]
# count set to True
test = normalize(data, keep_nan=False) # {1: 1, 2: 1, 3: 1, 4: 1}
```

Note
Ordered may not appear accurate if viewing in IDE.

native_skew(data):

Calculate Skew of a list.

Parameters
data (*list, np.ndarray, or pd.Series*) – Input data.

Returns
Returns the skew value.

Return type
float

Example
None

Note
None

`native_kurtosis(data):`

Calculate Kurtosis of a list.

Parameters
data (*list, np.ndarray, or pd.Series*) – Input data.

Returns
Returns the kurtosis value.

Return type
float

Example
None

Note
None

`native_percentile(data, q):`

Calculate Percentile of a list.

Parameters

- **data** (*list, np.ndarray, or pd.Series*) – Input data.
- **q** (*float*) – Percentile percent.

Returns
Returns the percentile value.

Return type
float

Example
None

Note
If input values are floats, will return float values.

CHAPTER

FOUR

GLOSSARY

Terms used in this documentation.

All in
This means a player has bet all their chips and if they lose they will be removed from the table.

Flush
Any five cards of the same suit, but not in a sequence.

Four of a Kind
All four cards of the same rank.

Full House
Three of a kind with a pair.

Hand
A hand or round counted when cards are dealt and a winner is decided. A hand may also be used to note the type of cards a player has ie. Straight, Two Pair, etc.

High Card
This notes a player won by having the highest card on the table.

Match
A match is a game played over one sitting.

Pair
Two cards of the same rank.

Pocket
A player is dealt two cards at he start of each round. These cards are known as pocket cards.

Position
There are four positions {Pre Flop, Post Flop, Post Turn, and Post River}. These are used to separate a hand.

Rank
Either number on the card or if a face card (A, K, Q, J)

Royal Flush
A, K, Q, J, 10, all the same suit.

Straight
Three cards of the same rank.

Straight Flush
Five cards in a sequence, all in the same suit.

Texas Hold'em
wiki

Three of a Kind
Three cards of the same rank.

Two Pair
Two different pairs.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- search

INDEX