Transaction Cost Analysis

# Technical Documentation:

1. Introduction   
   This project aims to provide post trade transaction cost analysis whose output can be represented as graphs or as csv files. The entire user interface is in the form of a lucid, easy to use webpage. This is an effective way of gauging the efficiency of a trade. At present there is no system in place to carry out such analysis.
2. Functional Requirements

* Python 2.7.
* Plotly library for graph plotting.
* Django library for hosting the website.
* Other libraries – multiprocessing, json, socket, csv

1. Implementation
2. User Interface
3. A neatly designed webpage serves as the User Interface.
4. The user needs to enter details – date, type of analysis (algorithm) performed, name of portfolio, the scrip (instrument).
5. The required graphs pertaining to the kind of analysis are then prepared. Click on the hyperlink to display any graph as one graph is displayed at a time on the webpage.
6. Analysis
   * 1. All execution starts from the CAnalysisEngine class.
     2. An abstract analysis class (CAnalysisAbstract) is created detailing the methods that need to be present (defined) in for all classes carrying out analysis. This serves as a framework for all analysis classes.
     3. CAnalysisAbstract has the following abstract methods:-
        1. GetData - Used to fetch the required data.
        2. AnalyseData - Used to run the analyses.
        3. VisualizeData - Used to visualize data in the form of charts, graphs or tables.
     4. Currently the classes designed can carry out – Volume VWAP analysis, NTWAP Execution analysis and Cash Future Spread analysis.   
        These are the classes carrying out the analyses:-
        1. CSimpleAnalysisVolVwap
        2. CSimpleAnalysisNtwap
        3. CSimpleAnalysisSpread
     5. The modular nature of the code means that any alternate analysis can be carried out just by adding another class detailing the analysis to be performed.
     6. Further, more complex analysis can be carried out by inheriting these existing classes that perform basic analysis thus making it scalable to a large extent.
7. Feeder
8. This is the broad representation of the data being fed into the Analysis Engine.
9. The files are read in accordance with the analysis being performed.
10. Data is collected using different server and client classes for different data (e.g execution data, market trade data) which can run concurrently.
11. Abstract server and abstract client classes are created detailing the methods that need to be present (defined) . These serve as frameworks for all server and client classes.  
    The following server and client classes are defined:-
    1. CDataAbstractServer - CDataBaseClient
    2. CDataMarketTradeServer - CDataMarketTradeClient
    3. CDataTwapPortfolioServer - CDataTwapPortfolioClient
    4. CDataPredictedServer - CDataPredictedClient
    5. CDataCashServer - CDataCashClient
    6. CDataFutureServer - CDataFutureClient
    7. CDataStrategyLogServer - CDataStrategyLogClient
12. CDataAbstractServer has the following methods defined:-
    1. Listen - This method binds the socket to the port, which is taken as an argument. Then it starts listening for client’s connection. Finally, it accepts the connection from the client and the connection is established.
    2. ReceiveParams - This method receives the json message containing the parameters from the client and parses it. It then stores the scrip name contained in the parsed json message in a member variable.
    3. SendJson - This method sends the row-wise data one by one in the form of json messages to the client.
13. CDataBaseClient has the following methods defined:-
    1. Connect - This method connects to the server taking the port as an argument.
    2. StoreData -
    3. SendParams -
    4. ReceiveData -
    5. Disconnect -
    6. StartClient -
14. The data retrieved is stored then stored using different classes defined for different data. A data abstract class is also created which would have member variables common to all the data classes.  
    A class (CFormatData) is also defined which is used to convert row-wise data stored in the form of a list into separate lists column wise and store it.  
    The following data classes are defined:-
    1. CDataAbstract
    2. CDataMarketTrade
    3. CDataTwapPortfolio
    4. CDataPredicted
    5. CDataCash
    6. CDataFuture
    7. CDataStrategyLog
15. The modular nature of the code means that any the retrieval of data for any alternate analysis can be carried out just by adding another class detailing the data that needs to be fed.
16. Utilities and Common Functions
17. Utils classes are defined which contain methods used to do all the required calculations.  
    The following Utils classes are defined:-
    1. CUtilsVolVwap  
       It has the following methods:-
       * GetVolWindow - Method to get volume of trade in one time window.
       * GetVwapWindow - Method to get Vwap in a given time window.
       * GetPercentVol - Method to get list of percentage of trade volume.
       * GetPercentFraction - Method to convert fraction of daily trade volume into percentage.
       * GetVolTotal - Method to get Volume traded for entire day for different time windows as a list.
       * GetVwapTotal - Method to get Vwap for entire day for different time windows as a list.
       * GetPartialTimeWindowList - Method to get time window liat from the point where the portfolio started trading
    2. CUtilsSpread  
       It has the following methods:-
       * GetLastPriceWindow - Method to get last traded price and populate the list of last traded prices.
       * GetLastPriceTotal - Method to get the list of last traded prices.
    3. CUtilsNtwap   
       It has the following methods:-
       * GetTimeStamp - Method to extract timestamp (string) from a row of data when reading a strategy log file.
       * GetPrice - Method to extract price from a row of data when reading a strategy log file.
       * GetStrategy - Method to extract strategy name from a row of data when reading a strategy log file.
       * FormatRow - Method to convert a row of data in strategy log files into a list.
       * GetStrategyList - Method to populate executed strategies list.
       * GetMovingAverages - Method to get list of moving averages when a portfolio data object is passed as an argument.
18. A CSpread class is created which is used to store the calculated data pertaining to cash-future spread.
19. A CCommonFunctions class is created which contains methods used frequently in various modules of the code.  
    It has the following methods:-
    1. GetDateTime - Returns datetime object when date and time are passed as arguments.
    2. GetDateTimeStrategyLog - Returns datetime object in case of reading a log file.
    3. GetDateTimePortfolio - Returns datetime object in case of reading a csv file containing portfolio execution data.
    4. GetDateTimeList - Used to populate the datetime list and return it.
    5. GetDateTimeListPortfolio - Used to populate the datetime list when reading data from a csv file containing portfolio execution data and return it.
    6. CompareTime - Used to compare two datetime objects. Returns -1 if datetime 1 < datetime 2 and 1 otherwise.
    7. FormatTime - Returns time in “hh:mm:ss” format (string). Converts “hhmmss” (string) into the above format.
    8. FormatDate - Returns date in “yyyymmdd” format (string). Converts “ddmmyyyy” (string) into the above format.
    9. GetWindowList - Returns two lists. One of them contains time windows (string) and the other contains mid points of the time windows (string).
    10. FindOccurences -
    11. ParseBuffer -
    12. FormatDateJSON -
20. A CMapping class is created which is used to map RIC to symbols and vice versa.
21. Plotter
    1. A CPlotFrameWork class is created which contains all the methods pertaining to plot/tabulate data. They are created in such a way that one needs to just pass the lists corresponding to the axes as arguments to obtain the graph. The other parameters are set as default within these methods (for e.g auto\_open is set as False within these methods).  
       It has the following methods:-
       1. GetBarChart - Method to get bar chart
       2. GetScatterPlot - Method to get scatter plot
       3. GetLayout - Method to get plot layout
       4. GetFigure - Method to get figure object
       5. GetPlot - Method to plot using figure object
    2. A CCustomPlot class is created which uses the methods defined in CPlotFramework to prepare the graphs. It prepares these graphs according to the type of analysis.  
       It has the following methods:-
       1. StartPlot - Method to pass required arguments to plotting methods.
       2. Market\_vs\_portfolio\_vwap - Method to plot Market vs Portfolio (VWAP).
       3. Market\_vs\_portfolio\_vol\_vwap - Method to plot Market vs Portfolio (Volume and VWAP).
       4. Market\_vs\_predicted\_vol - Method to plot Market vs Predicted (Volume)
       5. Market\_vs\_predicted\_vol\_diff - Method to plot Market vs Predicted (Volume Difference).
       6. Market\_vs\_orderCompletion\_price - Method to plot Market vs Order Completion Price.
       7. CF\_Spread - Method to plot Cash and Future spread data along with the minimum of both.
       8. NTWAP\_Execution - Method to plot NTWAP execution data along with market data.

# Functional Documentation:

1. Introduction  
   Post trade TCA is used to evaluate the performance of a strategy or trader. This kind of evaluation helps in identifying flaws which can be improved upon. It can be used with real time data feed as well to know the performance of the strategy in so far in the day. At present, there is no in-house facility to address this need.
2. Functional Requirement  
   There is a need to evaluate the performance of any strategy and benchmark it. A strategy which works fine now may start decaying any time in the future. There is a need to whether the strategy has executed trades at the right time. These things make a post trade TCA engine a requirement any trading firm cannot compromise with.  
   This TCA engine helps in analyzing the performance of the strategies by displaying graphs and charts in a very dynamic UI. It can be improved further to do a lot of other things. For e.g. it can be extended to run various other kinds of analyses, a backtesting engine can also be added to backtest the strategies and further a portfolio manager can also be developed and plugged into this device. Because of the ‘plug and play’ architecture, one just has to develop these additional components separately and plug it into the existing architecture.
3. User Manual  
   1) Open the dedicated web-page.  
   2) Select the date over which you want to run the analysis on.  
   3) Select the type of analysis from the drop-down menu.

4) Select the desired portfolio from the drop down menu containing the portfolios registered under the traders.  
5) Select the desired derivative from the drop down menu containing the derivatives under that portfolio.  
6) Click on “Generate Report” button and the graphs will be prepared.  
7) Click on the respective links to display the graphs.