Moduł wykresów – dokumentacja

Słowem wstępnym, oczywiście Python nie posiada czegoś takiego jak specyfikatory dostępu (to mnie przyznam dość zaskoczyło na początku), dlatego jeśli używam sformułowań „publiczne”, „prywatne” oznacza to „należy to traktować jako publiczne/prywatne”.

Jeśli nie napisano czy jakaś funkcja coś zwraca to nic nie zwraca.

Jeśli coś nie ma za sobą nawiasów to nie jest metodą, tylko polem.

Klasa ChartData

Klasa ta reprezentuje dane, które są rysowane na wykresie typu Chart lub CompareChart. Dane te tworzone są na podstawie obiektu klasy FinancialObject z modułu dataParser. Klasa ChartData odpowiada za pobranie odpowiedniego wycinka tych danych, transformację do formatu zrozumiałego dla funkcji matplotliba oraz za obliczanie wskaźników dla danych. W przypadku danych dla CompareChart przy okazji przekształcamy dane na procentowe zmiany od wartości odniesienia.

Konstruktor

__init__(self, finObj, start=None, end=None, step='monthly', compare=False)

finObj – obiekt klasy FinancialObject, z którego pobieramy dane

start, end – obiekty klasy datetime, które określają początkową i końcową datę

step – 'daily', 'weekly', 'monthly' w zależności od tego jakie chcemy zagęszczenie wartości

compare – czy dane mają być dla wykresu CompareChart (True) czy zwykłego Chart (False)

Metody i pola publiczne

open – lista z wartościami kursów otwarcia

close - lista z wartościami kursów zamknięcia

high - lista z wartościami maksymalnymi

low - lista z wartościami minimalnymi

volume – lista z wartościami wolumenu

date – lista obiektów klasy datetime (wartości osi x)

name – nazwa instrumentu finansowego (string)

quotes – lista krotek (time, open, close, high, low) – potrzebna, bo w tym formacie danych potrzebuje funkcja biblioteczna candlestick w wykresie

corrupted - True w przypadku wykrycia błędnych danych (tablice nierównej długości, start>=end), False jeśli wszystko ok

Metody i pola prywatne

fullArray – tablica przechowująca wszystkie (do poprawienia! Trzeba ustawić jakieś sensowne maksymalne duration do wskaźników np. 200 i pobierać tylko tyle ile naprawdę trzeba) wartości z FinancialObjecta. Potrzebne są one do obliczania wskaźników

getEarlierValues(self, length, type='close'): - Funkcja używana do wskaźników, które potrzebują wartości z okresu wcześniejszego niż dany okres (czyli chyba do wszystkich). Jeśli wcześniejsze wartości istnieją, są one pobierane z tablicy self.fullArray. W przeciwnym wypadku kopiujemy wartość początkową na wcześniejsze wartości (inaczej mowiąc, zakładamy, że przed debiotem kurs spółki był stały).

length = ilość dodatkowych wartości, które potrzebujemy

zwraca listę o długości o length dłuższej niż pierwotnie miała odpowiednia tablica

metody o nazwach odpowiadających nazwom wskaźników (CCI, SMA, ...) - służą do pobrania wartości odpowiedniego wskaźnika z uwzględnieniem pobrania wcześniejszych danych. Zwracają one tablicę z obliczonymi wartościami wskaźnika o długości równej pozostałych tablic (date, open, low, ...)

Klasa Chart

Ta klasa odpowiada za podstawowy typ wykresu (z dynamicznie zmieniającymi się wskaźnikami, oscylatorem, wolumenem, rysowaniem po wykresie itd.). Dziedziczy po klasie FigureCanvas (biblioteczna z matplotliba), która to z kolei może być użyta jako widget Qt.

Konstruktor

__init__(self, parent, finObj=None, width=8, height=6, dpi=100) -Tworzy domyślny wykres (liniowy z wolumenem, bez wskaźników) dla podanych danych. Domyślny rozmiar to 800x600 pixli

parent – to, w czym umieszczamy wykres (pytajcie Dawida, on się zna na Qt)

finObj – obiekt klasy FinancialObject, który ma być reprezentowany na wykresie

width, height – wysokość/szerokość w calach (lol)

dpi – ilość pixli na cal. Zatem wymiar wykresu to width*dpi na height*dpi. Chociaż to i tak nie ma znaczenia, bo wykres dopasowuje się do parenta (chyba, cokolwiek znaczy QtGui.QSizePolicy.Expanding)

Metody i pola publiczne

setData(self, finObj, start=None, end=None, step='daily') – ustawia model danych dla wykresu, po czym go odświeża. Parametry – patrz dokumentacja ChartData

setGrid(self, grid) - włącza lub wyłącza rysowanie grida

grid – True (grid on) / False (grid off)

setMainType(self, type) - Ustawiamy typ głównego wykresu

type – typ wykresu jako string ('point','line','candlestick','none')

updatePlot(self) – odświeża wszystkie podwykresy

addVolumeBars(self) – dodaje wolumen do wykresu

rmVolumeBars(self) – ukrywa wolumen

def setScaleType(self,type) - Ustawia skalę liniową lub logarytmiczną na głównym wykresie.

type – 'linear' lub 'log'.

setMainIndicator(self, type): - ustawiamy, jaki wskaźnik chcemy wyświetlać na głównym wykresie

type - 'SMA', 'WMA, 'EMA', 'bollinger', dowolna inna wartość oznacza żaden

setOscPlot(self, type) – ustawiamy, jaki oscylator chcemy wyświetlać

type – 'momentum', 'CCI', 'RSI', 'ROC', 'williams', dowolna inna wartość oznacza żaden

setDrawingMode(self, mode) – określamy, czy można rysować na wykresie

mode – True = można rysować, False = nie można rysować

drawLine(self, x0,y0,x1,y1) – dodaje do wykresu dodatkową linię (do zaznaczania trendów i formacji)

x0, y0 współrzędne (na wykresie) początku linii

x1, y1 współrzędne (na wykresie) końca linii

clearLines(self) – czyści wszystkie dodatkowe linie z wykresu (trendy, formacje i to co sobie user ręcznie namalował)

drawRectangle(self, x, y, width, height, colour='blue', lwidth = 2.0, lstyle = 'dashed') – rysuje prostokąt na wykresie (używane do zaznaczania formacji świecowych i luk)

x, y, width, height – wymiary jako współrzędne **na wykresie** a nie na ekranie

reszta - http://matplotlib.sourceforge.net/api/artist_api.html#matplotlib.patches.Rectangle

clearRectangles(self) – usuwa prostokąty narysowane na wykresie

Metody i pola prywatne

TBA :)