## Project Description

The goal of this project is to visualize income and expenses. The output visualization should give the viewer an understanding of :

1. Their income over time (past and future if desired)
2. How income is impacted by best-case and worst-case scenarios

This project was inspired by an argument with my girlfriend We were talking about getting engaged! This was a great topic, except we were arguing about money : I didn’t think we had enough money; she thought we had plenty of money (a story as old as time). The main point of argument was not about 1) how much we *should* spend on an engagement ring (although I wanted to spend less), or 2) how much money we have *right now.* We argued about how much money we will have in the future.

Our future finances are complicated by a lot of things :

1. Engagement ring
2. Wedding expenses
3. School if I want to go back to get my masters
4. Current and future levels of income
5. If I need a new car, health insurance, health issues, feelings of security, putting money in retirement, leisure spending […] etc..

Because we were mostly worried about future spending, I wanted to make a graph to give me an idea of what my future disposable income would actually look like. Therefore I wanted the following features :

1. Be able to graph past expenses
2. Be able to graph future projected expenses
   1. Graph future expenses with a level of uncertainty. I should be able to give best case and worst case scenarios AKA if a wedding costs $12,000 or $17,000 (or if school is $15,000 a year or $25,000 a year). This will be the third axis – probability (the other axes will be time and money)
3. Be intuitive to create new incomes/expenses
4. Gracefully handle one-time incomes and continuous incomes
5. Look presentation-worthy
6. Be able to parse historical data – aka take in a spreadsheet of historical bank transactions to make historical graphing easy

Knowing this, lets get started.

## Core Modules

My code is broken into three core modules : (2) classes and (1) plotting function.

### Income Class

**Income**(income, period, time\_start, time\_end, best\_case=None, worst\_case=None, proba=None, one\_time=False, tax=False, effective\_tax\_rate=0.2)

The income class models cash inflow and outflow. I will call a cash inflow or outflow a transaction for simplicity. The class can handle transactions that occur over time (yearly incomes), or at one time (single purchases). It optionally accepts a probability matrix for projecting worst/best case scenarios and supports taxes for incomes.

Parameters

* income : Transaction value you expect - define this per-period
* period : the time over the transaction happens. Format days/(365 days). Example, for monthly rent input 30.5 / 365
* time\_start : start time for transaction (datetime.date object)
* time\_end : end time for transaction (datetime.date object)
* best\_case : best case transaction value scenario. income = best\_case if best\_case is defined.
* worst\_case : worst case transaction value scenario
* proba : probability distribution between best and worst case. Type list or np.array of shape (10,). Structure [1, p(n-1) .. 0]. Intended to give the graph a different shape. If best\_case is defined and proba=None, proba will be a linear gradient between [1 ... 0]. Try np.linspace(0,1,num=11)
* one\_time : if this is a one\_time expense/income. In this case use time\_start to signify the date of expense. If one\_time = True, then only time\_start will be used in the run() method
* tax : enable tax calculation for income source. Uses effective\_tax\_rate to define a percentage. If you want to inflate expenses by a certain percent, use a negative value (ex -0.08 for 8% sales tax). The user should calculate other non-tax factors (401K, IRA, savings plans, other payments etc.) before using this class for typical earned income.
* effective\_tax\_rate : your effective tax rate (default 20%)

Methods

**Income.run**(run\_time)

* returns a gradient across proba which defines the expected transaction value during a single day. This method is called by the plotting function across a range of days (explained later). The income class methods should not need to be called by the user except for troubleshooting or verification. The primary job of this method is to return the income gradient if it is called within the class instances defined income period, or 0 if it is called outside of its time period.

**Income.** **calc\_derivative** (run\_time)

* called by the run() method. It uses a start date and end date to find the income gradient across a time. The result is returned to run(), which is the expected transaction value during a single day.

### Income\_Import Class

**Income**(path)

Parameters

* path: Directory to a csv file. Python will import the .csv file into a pandas dataframe and perform counting operations based on this data

The income\_Import class is useful for importing transactions from bank statements or ledgers. This class will also average transaction “types” or “categories” based on bank statements. This is useful for projecting future expenses based on past actual transactions.

Methods

**Income\_Import.** **create\_transaction\_objects**(value\_col=’Amount’, date\_col=’Date’)

* Creates Income class instances based on the file passed to the class initializer. The required rows of the bank records .csv file are <value\_col> and <date\_col>. These columns should hold the transaction amounts, and transaction dates for each transaction. The output is a list (iterable) of income classes. This is useful for passing to the plotting function. The plotting function will iterate over these income classes, and plot their contributions to (+) or (-) cash flow.

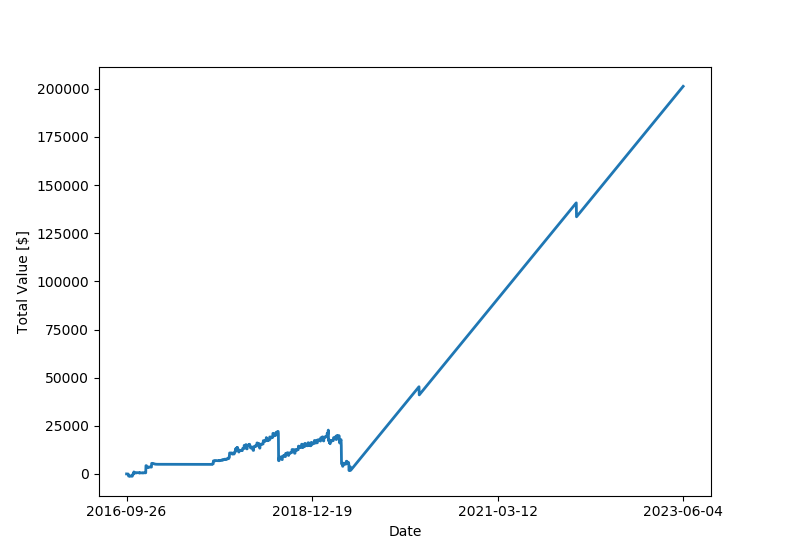
**Income\_Import.** **categorize\_transactions** (cat\_col='Category', date\_col='Date', by\_year=True)

* Outputs a dictionary of {category : value} based on the .csv file passed to the class initializer. The required rows of the bank records .csv file are <cat\_col>, <value\_col>, and <date\_col>. <cat\_col> is a column describing each transaction instances category (many banks export this information). <value\_col> again is the transaction amount. <date\_col> is the transaction date. The optional parameter by\_year indicates if the user wants transactions categorized by year, or over the entire period in the .csv file. This might be useful for more accurately projecting future expenses.

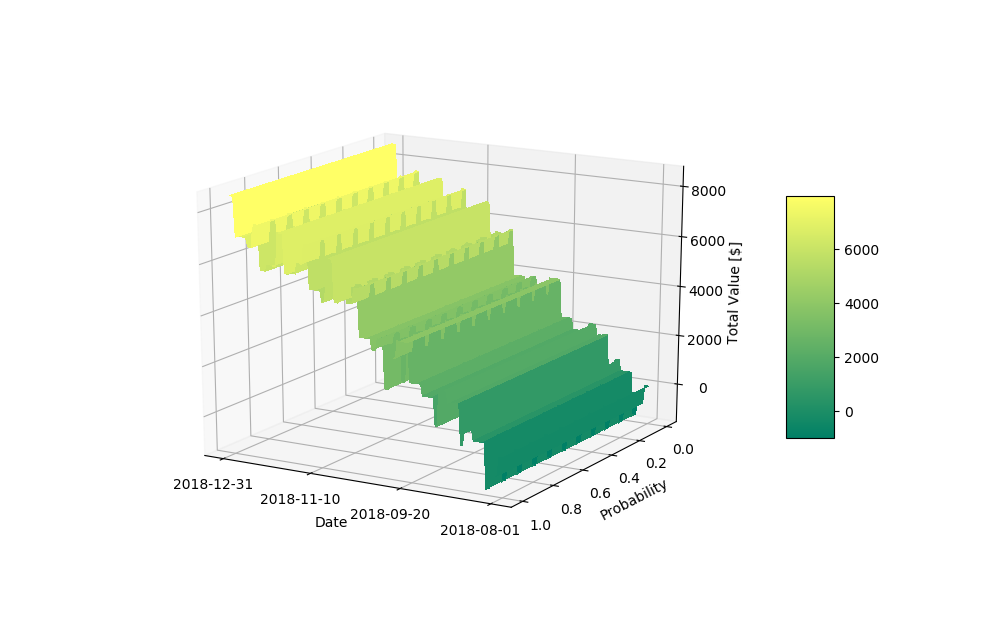
## Usage Example

1. Create custom income objects.
2. Import bank transaction .csv file
3. Create income objects from bank transactions
4. Append (extend) custom income objects and imported income objects
5. Pass the objects to the plotting function

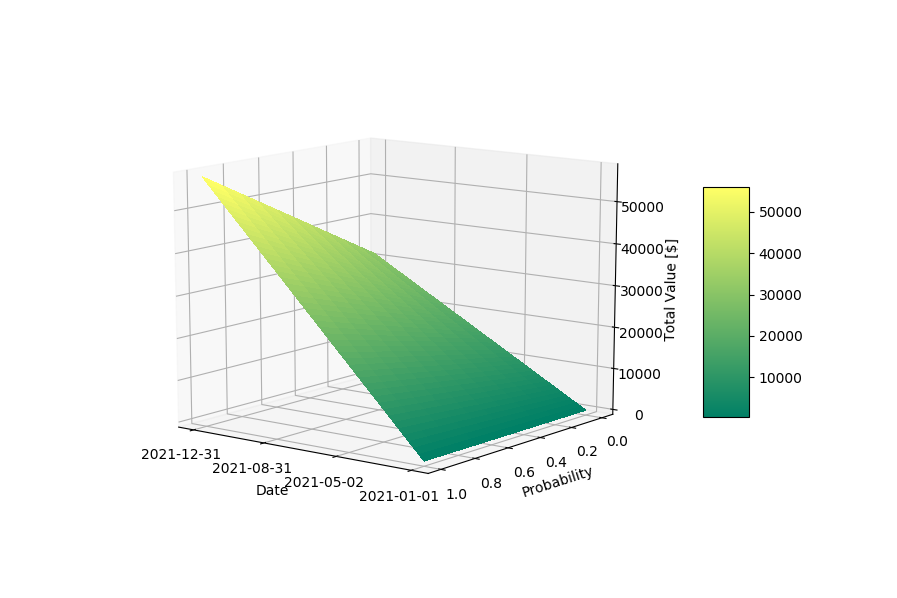




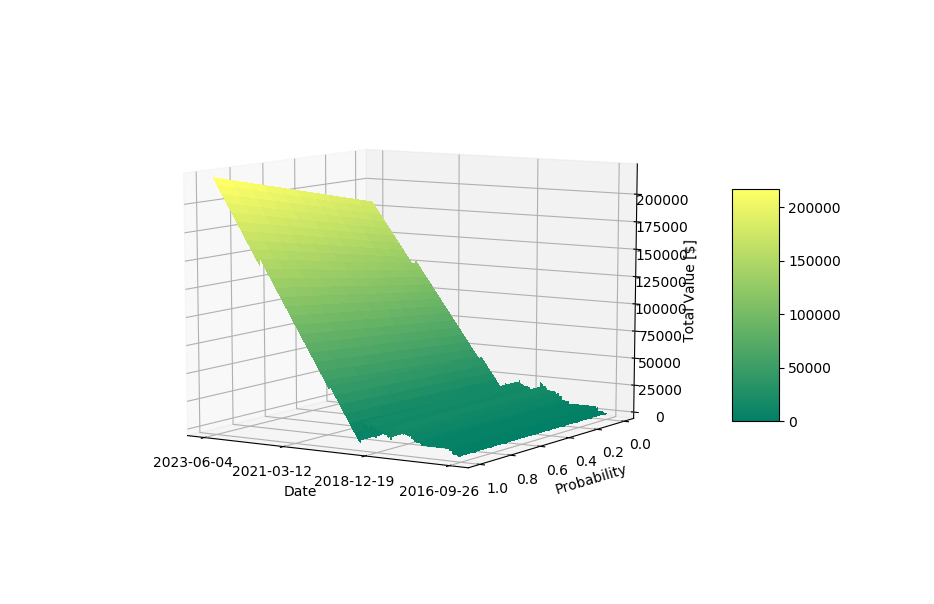
A 2D image is created as well as a 3D image.



Transactions on a 1-day time period can make the graph appear “Choppy”. This is especially true for large transactions relative to the current account balance.



This figure shows the “probability” axis of the 3D image. Income can be projected in the worst case (0 probability) or best case (1 probability). Different distributions can be input besides a linear distribution.



Transaction periods over long time frames appear smoother. I did not account for day-to-day expenses in this graph, which is why it goes sharply up after 2019.