Contents

[Summary: 2](#_Toc152686505)

[Analysis.py Documentation 2](#_Toc152686506)

[Function: testFunc(ticker) 2](#_Toc152686507)

[Parameters: 2](#_Toc152686508)

[Returns: 3](#_Toc152686509)

[Function: getDat(ticker) 3](#_Toc152686510)

[Parameters: 3](#_Toc152686511)

[Returns: 3](#_Toc152686512)

[Class: Security 3](#_Toc152686513)

[Attributes: 3](#_Toc152686514)

[Methods: 4](#_Toc152686515)

[Class: Portfolio 7](#_Toc152686516)

[Attributes: 7](#_Toc152686517)

[Methods: 9](#_Toc152686518)

[Class: Comparisons 14](#_Toc152686519)

[Attributes: 15](#_Toc152686520)

[Methods: 15](#_Toc152686521)

[App.py Documentation 17](#_Toc152686522)

[Layout 17](#_Toc152686523)

[User Inputs 17](#_Toc152686524)

[Displayed Objects 18](#_Toc152686525)

[Functions 18](#_Toc152686526)

[get\_portfolio(tickers, ra=1) 18](#_Toc152686527)

[update\_figures(portfolio,comp) 19](#_Toc152686528)

[Callbacks 19](#_Toc152686529)

[User Input Callback 19](#_Toc152686530)

[Download Callback 21](#_Toc152686531)

# Summary:

Goal of this project was to make a tool that retrieved stock data and automate the analysis of it to help the user make investment decisions.

Stonker does this by:

* Using the AlphaVantage API for stock history.
* Performing descriptive statistics on the stocks.
* Performing commonly accepted regressions on the stocks to gauge performance.
  + CAPM
  + Fama French three factor
  + Fama French five factor
* Displaying output in an attractive manner.
* Giving the user an easy way to adjust simple parameters.

There is still clear room for growth such as:

* Speeding up the tool by reducing the number of API calls.
* Giving the user the ability to store and modify existing portfolios.
* Adding more performance metrics.
* Tweaking the layout to make things more attractive.

Some notes:

* Statistics will not reflect the lifetime statistic of a particular stock for a couple reasons.
  + AlphaVantage only provides 20 years of data, so for long lived companies (IBM) stat will only reflect a subset of data available.
  + Since prices are consolidated within one data frame, all securities are cut off to whichever security had the most recent IPO.
    - ex. Security A is 10 years old, security B is 5 years old. Statistics are only going to be measured from the most recent 5 years of security A.
* Monthly RF and Fama French data is retrieved from their website. For a further discussion on what they are and how they are calculated please visit them.
  + <https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html>
* All statistics are annualized.

# Getting Started

* To get started you must go to the AlphaVantage website and apply for an API key
  + <https://www.alphavantage.co/support/#api-key>
* Unpaid will get you monthly closing prices.
  + Note. This will not include stock splits and dividends.
* Paid with get you adjusted monthly closing prices.
  + You may be able to get a free “Paid” license if you apply as an academic.
* Once received you must go to /src/utils/config.py
  + IN QUOTES. Place your API key to the right of: API\_Key =.
    - Line should look like: API\_Key = “DEMOAPI69420”
* If you paid for or somehow received a premium API key, set paid = True
  + Else, set paid = False
* To run simply run the app.py file and wait second.
* Then open a browser and go to your local host. Default is http://127.0.0.1:8050

# Analysis.py Documentation

* Note, this needs updating.
* This file performs the logic, math and data retrieval for App.PY

## Function: testFunc(ticker)

* This function is used for testing and debugging to avoid unnecessary API calls.
* It pulls previously downloaded JSON data from either IBM or AAPL and returns a dictionary of their adjusted closing price.

### Parameters:

* Ticker: str
  + A ticker symbol.

### Returns:

* Dictionary
  + A dictionary of returns from the selected symbol.
    - Key: date
    - Value: adjusted closing stock price.

## Function: getDat(ticker)

* This function utilizes the AlphaVantage API to retrieve a JSON of a selected ticker and return it.

### Parameters:

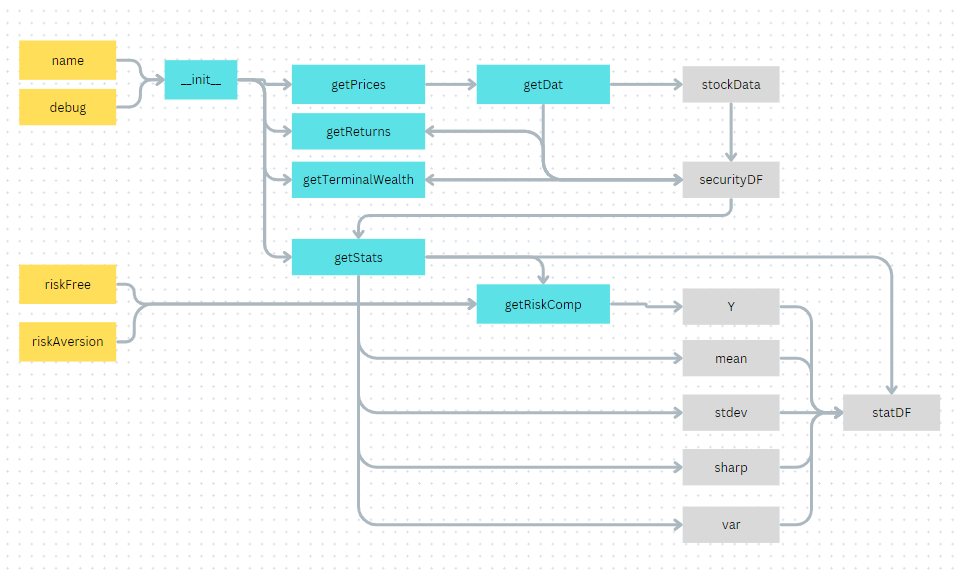
* Ticker: str
  + A ticker symbol.

### Returns:

* .json
  + A .json object that contains all information from the AlphaVantage API call for Monthly Adjusted Returns

## Class: Security

* The main class used. Each ticker that a user wants to inspect will be processed with this class.
* Inheritance: None



### Attributes:

* name: str
  + User defined name
* debug: Bool
  + Used to determine whether or not the testFunc of getDat function is used to retrieve stock information.
* securityDF: Pandas Dataframe
  + A dataframe containing dates of the closing price, closing price, returns and terminal wealth calculations.
* mean: float
  + Annualized Arithmetic Average of returns of the security.
    - Formula: Average returns \* 12
* stdev: float
  + Annualized standard deviation of returns of the security
    - Formula:
      * =Mean return of the security
      * xi=Individual return of the security
      * N=Number of returns in the data frame
* var: float
  + Annualized variance of the security
    - Formula: stdev2
* sharp: float
  + A measure of how much return to be expected per unit of risk
    - Formula:
      * E(r)=Arithmetic mean return of security
      * rf=Risk free rate, assumed to be .02
      * Standard deviation of security
* Y: float
  + A measure indicating how much of one’s portfolio should be invested in a security given their risk aversion.
    - Formula:
      * E(r)=Arithmetic mean return of security
      * rf=Risk free rate, assumed to be .02
      * A=User’s risk aversion
      * =Variance of the security
* statDF: Pandas Dataframe
  + A dataframe containing all of the above statistical measures of a security

### Methods:

#### \_\_init\_\_(self, name, debug=False):

* This method is used to create the security object and is run automatically when called.
* Creates all the Security attributes.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.
* name: str
  + Easily understood name of the object.
* debug: bool
  + Boolean which states whether we are running in debug mode.

##### Returns:

* An object of the Security class.

#### getPrices(self)

* If debug was set to true, this method calls the function getDat
* If not, this method calls the function testFunc
* Either way, it processes the json and returns a data frame with adjusted closing price and the date.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.

##### Returns:

* Security.securityDF: Pandas Dataframe
  + A dataframe with all dates for the index and adjusted price for the column.

#### getReturns(self)

* Processes the Security.securityDF and adds a column for the return.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.

##### Returns:

* Security.securityDF: Pandas Dataframe
  + A dataframe with all dates for the index and adjusted price and returns for the columns.

#### getTerminalWealth(self)

* Further processes the Security.securityDF and adds a column for terminal wealth of the security.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.

##### Returns:

* Security.securityDF: Pandas Dataframe
  + A dataframe with all dates for the index and adjusted price, returns and terminal wealth for the columns.

#### getRiskComp(self, riskFree=.02, riskAversion=1)

* This method takes a user’s defined risk free rate risk aversion and the arithmetic mean of the security and provides us the % of one’s portfolio one should invest in a security given that information.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.
* riskFree: float
  + Risk free rate. Default=.02
* RiskAversion: float
  + How sensitive the user is to risk. Default=1

##### Returns:

* + Y:float
    - The % of one’s portfolio one should invest in a security given the expected return, risk free rate and risk aversion.

#### getStats(self)

* This method uses the returns column of the Security.securityDF to derive descriptive statistics of the security in question.

##### Parameters:

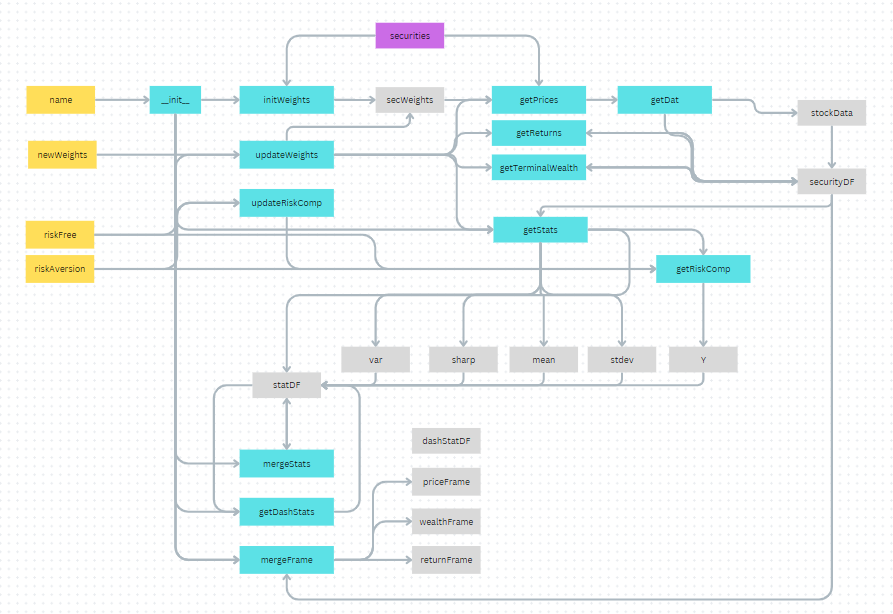
* self: variable
  + Variable which will be used to call this object.

##### Returns:

* securityDF: Pandas Dataframe
  + A dataframe containing dates of the closing price, closing price, returns and terminal wealth calculations.
* mean: float
  + Annualized Arithmetic Average of returns of the security.
    - Formula: Average returns \* 12
* stdev: float
  + Annualized standard deviation of returns of the security
    - Formula:
      * =Mean return of the security
      * xi=Individual return of the security
      * N=Number of returns in the data frame
* var: float
  + Annualized variance of the security
    - Formula: stdev2
* sharp: float
  + A measure of how much return to be expected per unit of risk
    - Formula:
      * E(r)=Arithmetic mean return of security
      * rf=Risk free rate, assumed to be .02
      * Standard deviation of security
* Y: float
  + A measure indicating how much of one’s portfolio should be invested in a security given their risk aversion.
    - Formula:
      * E(r)=Arithmetic mean return of security
      * rf=Risk free rate, assumed to be .02
      * A=User’s risk aversion
      * =Variance of the security
* statDF: Pandas Dataframe
  + A dataframe containing all of the above statistical measures of a security

## Class: Portfolio

* This class consolidates all the disparate securities in question and derives a new Security object with the name “Portfolio” that is the weighted average of the securities in question.
* Inheritance: Security



### Attributes:

* name: str
  + User defined name.
* securities: list
  + A list containing all the Security objects a user is investigating.
* returnFrame: Pandas Dataframe
  + A dataframe containing dates and returns of all securities and the portfolio.
* priceFrame: Pandas Dataframe
  + A dataframe containing dates and adjusted closing prices of all securities and the portfolio.
* wealthFrame: Pandas Dataframe
  + A dataframe containing dates and adjusted terminal wealth of all securities and the portfolio.
* securityDF: Pandas Dataframe
  + A dataframe containing dates of the closing price, closing price, returns and terminal wealth calculations of the portfolio.
  + Derived from the weighted average adjusted closing price of each security in Portfolio.securities
* statDF: Pandas Dataframe
  + A dataframe containing the descriptive statistics of each security and the portfolio.
* dashStatDF: Dictionary
  + This is the stat DF that has been flattened and made so that Dash will display the statDF correctly.
* secWeights: list
  + A list of all the weights of the securities.
* wDict: dictionary
  + A dictionary of all weights of the securities.
  + Key: Security.name
  + Value: Weight
* stdev: float
  + Annualized standard deviation of returns of the security
    - Formula:
      * =Mean return of the security
      * xi=Individual return of the security
      * N=Number of returns in the data frame
* var: float
  + Annualized variance of the security
    - Formula: stdev2
* sharp: float
  + A measure of how much return to be expected per unit of risk
    - Formula:
      * E(r)=Arithmetic mean return of security
      * rf=Risk free rate, assumed to be .02
      * Standard deviation of security
* Y: float
  + A measure indicating how much of one’s portfolio should be invested in a security given their risk aversion.
    - Formula:
      * E(r)=Arithmetic mean return of security
      * rf=Risk free rate, assumed to be .02
      * A=User’s risk aversion
      * =Variance of the security

### Methods:

#### \_\_init\_\_(self, name, securities):

* This method is used to create the Portfolio object and is run automatically when called.
* Creates all the Portfolio attributes.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.
* name: str
  + Easily understood name of the object.
* securities: list
  + List of Security objects to be analyzed.

##### Returns:

* An object of the Portfolio class.

#### initWeights(self)

* Creates default weights for each security equal to 1/# of securities in Portfolio.securities.
* Needed to derive the weighted average of the prices of each security.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.

##### Returns:

* secWeights: list
  + A list of all of the security weights
* wDict: dictionary
  + A dictionary of all weights of the securities.
  + Key: Security.name
  + Value: Weight

#### updateWeights(self, newWeights)

* A method to override the weights created in initWeights.
* It will automatically re-perform all analysis when called.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.
* newWeights: dictionary
  + A dictionary containing the Security.name and new weight of each security we want to change.
  + Key: Security.name
  + Value: Weight

##### Returns:

* returnFrame: Pandas Dataframe
  + A dataframe containing dates and returns of all securities and the portfolio.
* priceFrame: Pandas Dataframe
  + A dataframe containing dates and adjusted closing prices of all securities and the portfolio.
* wealthFrame: Pandas Dataframe
  + A dataframe containing dates and adjusted terminal wealth of all securities and the portfolio.
* securityDF: Pandas Dataframe
  + A dataframe containing dates of the closing price, closing price, returns and terminal wealth calculations of the portfolio.
  + Derived from the weighted average adjusted closing price of each security in Portfolio.securities
* statDF: Pandas Dataframe
  + A dataframe containing the descriptive statistics of each security and the portfolio.
* dashStatDF: Dictionary
  + This is the stat DF that has been flattened and made so that Dash will display the statDF correctly.
* secWeights: list
  + A list of all the weights of the securities.
* wDict: dictionary
  + A dictionary of all weights of the securities.
  + Key: Security.name
  + Value: Weight
* stdev: float
  + Annualized standard deviation of returns of the security
    - Formula:
      * =Mean return of the security
      * xi=Individual return of the security
      * N=Number of returns in the data frame
* var: float
  + Annualized variance of the security
    - Formula: stdev2
* sharp: float
  + A measure of how much return to be expected per unit of risk
    - Formula:
      * E(r)=Arithmetic mean return of security
      * rf=Risk free rate, assumed to be .02
      * Standard deviation of security
* Y: float
  + A measure indicating how much of one’s portfolio should be invested in a security given their risk aversion.
    - Formula:
      * E(r)=Arithmetic mean return of security
      * rf=Risk free rate, assumed to be .02
      * A=User’s risk aversion
      * =Variance of the security

#### getPrices(self)

* Consolidates adjusted closing price of each Security into one data frame.
* For each date, computes the average using each securities weight previously determined by either initWeights or updateWeights.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.

##### Returns:

* securityDF: Pandas Dataframe
  + A dataframe with all dates for the index and adjusted price for the column.

#### getReturns(self) - inherited

* Processes the Security.securityDF and adds a column for the return.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.

##### Returns:

* Security.securityDF: Pandas Dataframe
  + A dataframe with all dates for the index and adjusted price and returns for the columns.

#### getTerminalWealth(self) - inherited

* Further processes the Security.securityDF and adds a column for terminal wealth of the security.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.

##### Returns:

* Security.securityDF: Pandas Dataframe
  + A dataframe with all dates for the index and adjusted price, returns and terminal wealth for the columns.

#### getRiskComp(self, riskFree=.02, riskAversion=1) - inherited

* This method takes a user’s defined risk free rate risk aversion and the arithmetic mean of the security and provides us the % of one’s portfolio one should invest in a security given that information.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.
* riskFree: float
  + Risk free rate. Default=.02
* RiskAversion: float
  + How sensitive the user is to risk. Default=1

##### Returns:

* + Y:float
    - The % of one’s portfolio one should invest in a security given the expected return, risk free rate and risk aversion.

#### getStats(self) - inherited

* This method uses the returns column of the Security.securityDF to derive descriptive statistics of the security in question.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.

##### Returns:

* securityDF: Pandas Dataframe
  + A dataframe containing dates of the closing price, closing price, returns and terminal wealth calculations.
* mean: float
  + Annualized Arithmetic Average of returns of the security.
    - Formula: Average returns \* 12
* stdev: float
  + Annualized standard deviation of returns of the security
    - Formula:
      * =Mean return of the security
      * xi=Individual return of the security
      * N=Number of returns in the data frame
* var: float
  + Annualized variance of the security
    - Formula: stdev2
* sharp: float
  + A measure of how much return to be expected per unit of risk
    - Formula:
      * E(r)=Arithmetic mean return of security
      * rf=Risk free rate, assumed to be .02
      * Standard deviation of security
* Y: float
  + A measure indicating how much of one’s portfolio should be invested in a security given their risk aversion.
    - Formula:
      * E(r)=Arithmetic mean return of security
      * rf=Risk free rate, assumed to be .02
      * A=User’s risk aversion
      * =Variance of the security
* statDF: Pandas Dataframe
  + A dataframe containing all of the above statistical measures of a security

#### mergeFrame(self, colName)

* This method uses the Pandas.merge method to combine columns from Security.securityDF.
* Needed in order to create the Portfolio.PriceFrame, Portfolio.wealthFrame, Portfolio.securityDF, Portfolio.statDF and Portfolio.dashStatDF.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.
* colName: str
  + The name of the column in each Security.securityDF to merge with the Portfolio.securityDF.
    - “Returns”, “Close\_Price”, “T\_Wealth”

##### Returns:

* mFrame: Pandas Dataframe
  + A dataframe containing the returns, closing prices or terminal wealth which overlap dates of all securities and portfolio.

#### mergeStats(self)

* This method joins all the individual Security.statDF to the Portfolio.statDF

##### Parameters:

* self: variable
  + Variable which will be used to call this object.

##### Returns:

* statDF: Pandas Dataframe
  + A dataframe containing the descriptive statistics of each security and the portfolio.

#### getDashStats(self)

* This method takes the Portfolio.statDF and creates a dictionary that is useable by Plotly.Dash for display.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.

##### Returns:

* dashStatDF: Dictionary
  + This is the stat DF that has been flattened and made so that Dash will display the statDF correctly.

#### updateRiskComp(self, rf=.02, ra=1)

* This method changes the users risk aversion level and recalculates Y\* for each security and the portfolio as a whole.

##### Parameters:

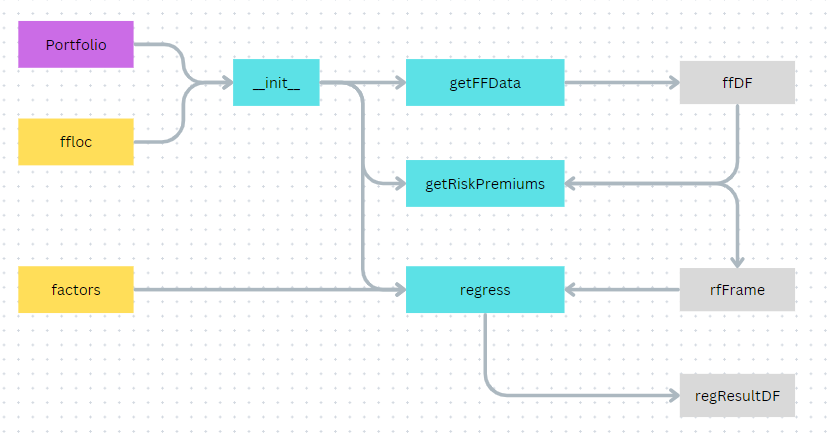
* self: variable
  + Variable which will be used to call this object.
* rf: float
  + Default=.02
  + Risk free rate
* ra: int or float
  + Default = 1
  + User risk aversion level.

##### Returns:

* statDF: Pandas Dataframe
  + A dataframe containing the descriptive statistics of each security and the portfolio.
* dashStatDF: Dictionary
  + This is the stat DF that has been flattened and made so that Dash will display the statDF correctly.

## Class: Comparisons

* This class accesses data provided by Kenneth R. French.
* It then runs regressions on each security and the portfolio
* Securities and portfolio regressed with:
  + Mkt-rf
  + Mkt-rf, SMB, HML
  + Mkt-rf, SMB, HML, RMW, CMA
* For further discussion on what those are and why we do this go to the website below:
* <https://mba.tuck.dartmouth.edu/pages/faculty/ken.french/data_library.html>



### Attributes:

* portfolio: Object of class Portfolio
  + Portfolio object containing relevant data needed for computations.
* ffloc: str
  + Location of French data.
* ffDF: Pandas Dataframe
  + A dataframe containing all the data from Kenneth R. French
* CAPM: Pandas Dataframe
  + Data frame containing the results of the returns of each Security and Portfolio being regressed on Mkt-rf
* ff3: Pandas Dataframe
  + Data frame containing the results of the returns of each Security and Portfolio being regressed on Mkt-rf, SMB, HML
* ff5: Pandas Dataframe
  + Data frame containing the results of the returns of each Security and Portfolio being regressed on Mkt-rf, SMB, HML, RMW, CMA
* rfFrame: Pandas Dataframe
  + A data frame that contains the risk free rate of that month subtracted from the month returns of each Security and Portfolio.

### Methods:

#### \_\_init\_\_(self, portfolio, ffloc= root\_dir/"src"/"assets"/"data"/"ff.csv"):

* This method is used to create the Comparisons object and is run automatically when called.
* Creates all the Comparisons attributes.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.
* portfolio: Object of class Portfolio
  + Portfolio object containing relevant data needed for computations.
* ffloc: str
  + Default= root\_dir/"src"/"assets"/"data"/"ff.csv"):
  + Location of French data.

##### Returns:

* portfolio: Object of class Portfolio
  + Portfolio object containing relevant data needed for computations.
* ffloc: str
  + Location of French data.
* CAPM: Pandas Dataframe
  + Data frame containing the results of the returns of each Security and Portfolio being regressed on Mkt-rf
* ff3: Pandas Dataframe
  + Data frame containing the results of the returns of each Security and Portfolio being regressed on Mkt-rf, SMB, HML
* ff5: Pandas Dataframe
  + Data frame containing the results of the returns of each Security and Portfolio being regressed on Mkt-rf, SMB, HML, RMW, CMA
* rfFrame: Pandas Dataframe
  + A data frame that contains the risk free rate of that month subtracted from the month returns of each Security and Portfolio.

#### getFFData(self)

* Retrieves the data by Kenneth French, stores in dataframe and then formats dataframe.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.

##### Returns:

* ffDF: Pandas Dataframe
  + A dataframe containing all the data from Kenneth R. French

#### getRiskPremiums

* A method used to subtract the monthly risk free rate provided by Kenneth r French from the monthly return of each security and portfolio.
* Needed to for regressions.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.

##### Returns:

* rfFrame: Pandas Dataframe
  + A data frame that contains the risk free rate of that month subtracted from the month returns of each Security and Portfolio.

#### regress(self, factors)

* Performs an OLS regression for the returns of each security and the portfolio with the user defined factors.
* Provides the results in a dataframe.
* Used iteratively in \_\_init\_\_ to create the CAPM, ff3 and ff5 dataframes.

##### Parameters:

* self: variable
  + Variable which will be used to call this object.
* factors: list
  + A list of the factors we want to regress with.
    - Possible: Mkt-rf, SMB, HML, RMW, CMA

##### Returns:

* regResultDF: Pandas Dataframe
  + A dataframe containing the results of the regression

# App.py Documentation

* This code displays data based on user inputs and the Analysis.py file.
* Code is displayed in the users browser.

## Layout

### User Inputs

#### tickers

* Type: dcc.Input
* A space for users to input a list of tickers they would like to examine. Must be separated by commas and not have spaces.

#### stock\_submit

* Type: html.Button
* A button that when pressed triggers the update\_page function

#### userRA

* Type: dcc.Input
* A space for users to input their desired risk aversion level.

#### raSubmit

* Type: html.Button
* A button that when pressed triggers the update\_page function

#### xlsx-button

* Type: html.Button
* A button that when pressed triggers the “xlsxExport” function

### Displayed Objects

#### weightTable

* Type: dash\_table.DataTable
* A table displaying the weights of each security. The cells are editable by the user.
* Upon editing by the user, the update\_page function is called.

##### statTable

* Type: dash\_table.DataTable
* A table displaying the summary statistics of each security and the portfolio overall.

#### twGraph

* Type: dcc.Graph
* A graph showing the terminal wealth over time of each security and portfolio.

#### priceGraph

* Type: dcc.Graph
* A graph showing the price over time of each security and portfolio.

##### CAPMTable

* Type: dash\_table.DataTable
* A table displaying the results of the CAPM Regression.

##### ff3Table

* Type: dash\_table.DataTable
* A table displaying the results of the three factor regression.

##### ff5Table

* Type: dash\_table.DataTable
* A table displaying the results of the five factor regression.

## Functions

### get\_portfolio(tickers, ra=1)

* Since Dash is stateless and doesn’t store objects in memory, we must use this function every time a user changes a value.

#### Parameters

* tickers: str
  + A sting containing the ticker symbols of securities a user would like to investigate.
  + Tickers must be separated by commas and have no spaces.
  + Can be set with the “tickers” input.
* ra: int or float
  + Default = 1
  + The users risk aversion level which can be set with the “userRA” input.

#### Returns

* portfolio
  + An object of class Portfolio
  + Class imported from Analysis.py
* comp
  + An object of class Comparisons
  + Class imported from Analysis.py

### update\_figures(portfolio,comp)

* Function formats the Portfolio and Comparison class attributes to be useable by Dash
* All of the callbacks update the tables and graph so this function exists so that it isn’t copy and pasted underneath each function in each callback.

#### Parameters

* portfolio
  + An object of class Portfolio
  + Class imported from Analysis.py
* comp
  + An object of class Comparisons
  + Class imported from Analysis.py

#### Returns

* wCols: lst
  + list containing a dictionary of the column names and ids in the weight table
* wDat: lst
  + A list containing the the wDict attribute of the passed Portfolio object.
* sFrame: Pandas Dataframe
  + The dashStatDF attribute of the passed Portfolio object.
* price: Plotly Express Line Graph
  + A line graph made from the priceFrame attribute of the passed Portfolio object.
* twealth: Plotly Express Line Graph
  + A line graph made from the wealthFrame attribute of the passed Portfolio object.
* capm: dictionary
  + A dictionary containing made from the data withing the CAPM attribute of the passed Comparisons object.
* ff3
  + A dictionary containing made from the data withing the ff3 attribute of the passed Comparisons object.
* ff5
  + A dictionary containing made from the data withing the ff5 attribute of the passed Comparisons object.

## Callbacks

* Callbacks are used by Dash to listen for triggers from the user and perform some kind of action by calling the function that follows.

### User Input Callback

* This specific callback listens for everything under User Inputs.

#### Functions

##### update\_page(b1,b2,wd,wc,tickers,ra)

* This function listens for what is being triggered by the user.
* It then calls a specific function depending on the trigger.

###### Parameters

* b1: int
  + A counter that returns +1 if the user clicks on the “stock\_submit” button
* b2: int
  + A counter that returns +1 if the user clicks on the “raSubmit” button
* wd: dictionary
  + A dictionary of the data within the weight table.
* wc: dictionary
  + A dictionary of the column ids and names in the weight table
* tickers: lst
  + A list of objects of the Securities class which can be set with the “tickers” input.
* ra: int or float
  + The users risk aversion level which can be set with the “userRA” input

###### Returns

* if “stock\_submit” button triggered callback
  + call to “ticker\_update” function
* else if “raSubmit” button triggered callback
  + call to “Y\_update” function
* else if “weightTable” triggered callback
  + call to “weight\_update” function

##### ticker\_update(tickers = "AAPL,IBM", ra="1")

* Function pulls existing user tickers and risk aversion and creates an object of class Portfolio and an object of class Comparisions.
* It then uses those objects as parameters in the “update\_figures” function to update the dash display.

###### Parameters

* tickers: str
  + A sting containing the ticker symbols of securities a user would like to investigate.
  + Tickers must be separated by commas and have no spaces.
  + Can be set with the “tickers” input.
* ra: int or float
  + Default = 1
  + The users risk aversion level which can be set with the “userRA” input.

###### Returns

* Call to “update\_figures” function

##### Y\_update(tickers="AAPL",ra= "1")

* Function pulls existing user tickers and risk aversion and creates an object of class Portfolio and an object of class Comparisions.
* It then uses those objects as parameters in the “update\_figures” function to update the dash display.

###### Parameters

* tickers: str
  + A sting containing the ticker symbols of securities a user would like to investigate.
  + Tickers must be separated by commas and have no spaces.
  + Can be set with the “tickers” input.
* ra: int or float
  + Default = 1
  + The users risk aversion level which can be set with the “userRA” input.

###### Returns

* Call to “update\_figures” function

##### weight\_update(rows,columns,tickers,ra)

* Function pulls the new weights of the securities as defined by the user, tickers and risk aversion and creates an object of class Portfolio and an object of class Comparisions.
* It then uses those objects as parameters in the “update\_figures” function and “updateWeights” method.

###### Parameters

* rows: dictionary
  + A dictionary of the data within the weight table.
* columns: dictionary
  + A dictionary of the column ids and names in the weight table
* tickers: str
  + A sting containing the ticker symbols of securities a user would like to investigate.
  + Tickers must be separated by commas and have no spaces.
  + Can be set with the “tickers” input.
* ra: int or float
  + Default = 1
  + The users risk aversion level which can be set with the “userRA” input.

###### Returns

* Call to “update\_figures” function

### Download Callback

* This callback listens for a user to click on the export button and then creates an excel spreadsheet of all the information generated.

#### Functions

##### xlsxExport(n\_clicks,rows,columns,tickers,ra)

* Function is triggered when a user clicks on the “xlsx-button” listens for a user to click on the export button and then creates an excel spreadsheet of all the information generated.

###### Parameters

* n\_clicks: int
  + A counter that returns +1 if the user clicks on the “xlsx-button” button.
* rows: dictionary
  + A dictionary of the data within the weight table.
* columns: dictionary
  + A dictionary of the column ids and names in the weight table
* tickers: str
  + A sting containing the ticker symbols of securities a user would like to investigate.
  + Tickers must be separated by commas and have no spaces.
  + Can be set with the “tickers” input.
* ra: int or float
  + Default = 1
  + The users risk aversion level which can be set with the “userRA” input.

###### Returns

* A excel spreadsheet containing all information within the Portfolio and Comparisons objects.