Collaboration in Higher Education for Digital Transformation in European Business

Data Analysis for Financial Markets

(Practical Case Study)

Jan Budík, budikjan@gmail.com,
Brno University of Technology/Czech Republic

This working paper is a result of the EU Erasmus Plus project "Collaboration in Higher Education for Digital Transformation of Corporate Businesses" (CHEDTEB) and essential ideas come from the intensive exchange with IT experts and company managers of the working groups "Big Data" and "Blockchain technology". Further information about Big Data, algorithms, AI and Blockchain technology can be found on our project webpage.

http://www.chedteb.eu/

The author is grateful for the lively discussions within various Big Data workshops and would like to thank in particular following colleagues for their valuable ideas, and suggestions:
Jan Luhan, Martin Fridrich, Petr Novotný – all of Brno University of Technology/Czech Republic,
Viire Täks, Sherif Sakr both from Tartu University/Estonia,
Rainer Lenz, Bernd Kleinheyer, Tahir Lushi, Margareta Teodorescu - all of University of Applied Sciences Bielefeld/Germany

Co-funded by the Erasmus+ Programme of the European Union

Content

Data Analysis for Financial Markets

This document shows how to work with financial data and how to analyze them with appropriate technologies.

How to analyze data for Financial Markets?

Technologies

What technologies do we need and what could we use?

Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level build in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

(Source: https://www.python.org/doc/essays/blurb/)

Pandas

Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way toward this goal. Pandas is well suited for many different kinds of data: -Tabular data with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet -Ordered and unordered (not necessarily fixed-frequency) time series data. -Arbitrary matrix data (homogeneously typed or heterogeneous) with row and column labels -Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into a pandas data structure

(Source: https://pandas.pydata.org/pandas-docs/stable/getting_started/overview.html)

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery. For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

(Source: https://matplotlib.org/)

Plotly

Plotly provides online graphing, analytics, and statistics tools for individuals and collaboration, as well as scientific graphing libraries for Python, R, MATLAB, Perl, Julia, Arduino, and REST.

(Source: https://plot.ly/#/)

Seaborn

Seaborn is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with Pandas data structures.

Seaborn offers these functionalities: -A dataset-oriented API for examining relationships between multiple variables -Specialized support for using categorical variables to show observations or aggregate statistics -Options for visualizing univariate or bivariate distributions and for comparing them between subsets of data -Automatic estimation and plotting of linear regression models for different kinds dependent variables -Convenient views onto the overall structure of complex datasets -High-level abstractions for structuring multi-plot grids that let you easily build complex visualizations -Concise control over matplotlib figure styling with several built-in themes -Tools for choosing color palettes that faithfully reveal patterns in your data

Seaborn aims to make visualization a central part of exploring and understanding data. Its dataset-oriented plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots.

(Source: https://seaborn.pydata.org/introduction.html)

Jupyter notebook

Project Jupyter is a non-profit, open-source project, born out of the IPython Project in 2014 as it evolved to support interactive data science and scientific computing across all programming languages. Jupyter will always be 100% open-source software, free for all to use and released under the liberal terms of the modified BSD license.

(Source: https://jupyter.org/about)

Quantopian

Quantopian is a crowd-sourced quantitative investment firm that inspires talented people from around the world to write investment algorithms. Quantopian provides capital, education, data, a research environment, and a development platform to algorithm authors (quants). It offers license agreements for algorithms that fit investment strategy, and the licensed authors are paid based on their strategy's individual performance. Quantopian provides everything a quant needs to create a strategy and profit from it. Quantopian is committed to the open source software community.

(Source: https://www.quantopian.com/about)

Zipline

Zipline is a Pythonic algorithmic trading library. It is an event-driven system for backtesting. Zipline is currently used in production as the backtesting and live-trading engine powering Quantopian – a free, community-centered, hosted platform for building and executing trading strategies. Zipline tries to get out of your way so that you can focus on algorithm development. Many common statistics like moving average and linear regression can be readily accessed from within a user-written algorithm. The input of historical data and the output of performance statistics are based on Pandas DataFrames to integrate nicely into the existing PyData ecosystem. You can use libraries like matplotlib, scipy, statsmodels, and sklearn to support development, analysis, and visualization of state-of-the-art trading systems.

(Source: https://www.zipline.io/)

Pyfolio

Pyfolio is a Python library for performance and risk analysis of financial portfolios developed by Quantopian Inc. It works well with the Zipline open source backtesting library. At the core of pyfolio is a so-called tear sheet which consists of various individual plots that provide a comprehensive image of the performance of a trading algorithm.

(Source: https://github.com/quantopian/pyfolio)

Data sources

Where could we find data?

Morningstar

Morningstar is a Chicago-based investment research firm that compiles and analyzes fund, stock and general market data. They also provide an extensive line of internet, software and print-based products for individual investors, financial advisors and institutional clients. The research reaches all corners of the world, including North America, Europe, Australia, and Asia. Among its many offerings, Morningstar's comprehensive, one-page mutual and exchange-traded fund (ETF) reports are widely used by investors to determine the investment quality of the more than 2,000 funds. Morningstar is a respected and reliable source of independent investment analysis for all levels of fund and stock investors, ranging from inexperienced beginners to sophisticated experts. This extensive line of products empowers various financial professionals, including individual investors, financial advisors, asset managers, retirement plan providers, and institutional investors. The data and research provided by Morningstar include insights on investment offerings, managed investment products, publicly listed companies, and real time market data. Its website includes free information on individual funds and stocks.

(Source: https://www.investopedia.com/terms/m/morningstarinc.asp)

Quandl

Quandl has a vast collection of free and open data collected from a variety of organizations: central banks, governments, multinational organizations and more. You can use it without payment and with few restrictions. Some data on Quandl is premium and can only be accessed with a subscription. Subscriptions are a la carte: You can subscribe to the individual premium datasets you need and nothing more. Subscriptions can be canceled at any time. All premium datasets benefit from Quandl's free data-usage features: full API access, integration with our many libraries and tools, downloads in any format, multiple export and visualization options, and more. Most premium datasets come with extensive free samples. While free data is suitable for experimentation and exploration, we strongly recommend using premium data for professional applications.

(Source: https://docs.quandl.com/docs)

Yahoo

Source: https://finance.yahoo.com/

Barchart

https://www.barchart.com/ondemand/free-market-data-api

Use case 1 - Charts

This use case shows how to plot a price chart of 10 commodities which are possible to trade on an electronic exchange.

We'll analyze a commodity future contract, what is a legal agreement to buy or sell a commodity at a predetermined price at a specified time in the future.

You can find more information about future contract here:

https://www.investopedia.com/terms/f/futurescontract.asp

Step #1 - Import of important Libraries

Pandas - library for data analysis https://pandas.pydata.org/
Matplotlib - library for visualization https://matplotlib.org
Numpy - library for scientific computing https://www.numpy.org/
Quantopian - library for financial market analysis
https://www.quantopian.com/help

In [XX]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from quantopian.research.experimental import continuous_future
from quantopian.research.experimental import history
```

Step #2 - Date Range Definition

In [XX]:

```
start_date='2010-12-04'
end_date='2018-10-30'
```

Step #3 - Commodities Definition

Each asset has a specific and unique shortcut which is called "Ticker". We will work only with 10 commodities:

CL - Light Sweet Crude Oil
NG - Natural Gas
GC - Gold
XB - RBOB Gasoline Futures
SY - Soybeans
LC - Live Cattle
HG - Copper High Grade
SV - Silver
PL - Platinum
SB – Sugar

You can find more information about tickers here:

https://www.investopedia.com/terms/t/tickersymbol.asp

In [XX]:

```
commodities = ['CL','NG','GC','XB','SY', 'LC', 'HG', 'SV', 'PL', 'SB']
```

Step #4 - Continuous Contract Definition Based on Volume

Future contract is based on a specific delivery date when contract expires.

You can find more information about the continuous contract here:

https://www.premiumdata.net/support/futurescontinuous.php

In [XX]:

```
CL = continuous_future(commodities[0], offset=0, roll='volume')
NG = continuous_future(commodities[1], offset=0, roll='volume')
GC = continuous_future(commodities[2], offset=0, roll='volume')
XB = continuous_future(commodities[3], offset=0, roll='volume')
SY = continuous_future(commodities[4], offset=0, roll='volume')
LC = continuous_future(commodities[5], offset=0, roll='volume')
HG = continuous_future(commodities[6], offset=0, roll='volume')
SV = continuous_future(commodities[7], offset=0, roll='volume')
PL = continuous_future(commodities[8], offset=0, roll='volume')
SB = continuous_future(commodities[9], offset=0, roll='volume')
```

Step #5 - Historical Data Download

Historical price change is usually saved as open, high, low and close (OHLC) for specific time period.

You can find more information about OHLC here:

https://www.investopedia.com/terms/o/ohlcchart.asp

In [XX]:

```
CLdata = history(CL, fields='close_price', start=start_date, end=end_date, frequency='d
aily')
CLdataDropNa = CLdata.dropna(how='all')
NGdata = history(NG, fields='close_price', start=start_date, end=end_date, frequency='d
aily')
NGdataDropNa = NGdata.dropna(how='all')
GCdata = history(GC, fields='close_price', start=start_date, end=end_date, frequency='d
aily')
GCdataDropNa = GCdata.dropna(how='all')
XBdata = history(XB, fields='close_price', start=start_date, end=end_date, frequency='d
aily')
XBdataDropNa = XBdata.dropna(how='all')
SYdata = history(SY, fields='close_price', start=start_date, end=end_date, frequency='d
aily')
SYdataDropNa = SYdata.dropna(how='all')
LCdata = history(LC, fields='close_price', start=start_date, end=end_date, frequency='d
aily')
LCdataDropNa = LCdata.dropna(how='all')
HGdata = history(HG, fields='close_price', start=start_date, end=end_date, frequency='d
aily')
HGdataDropNa = HGdata.dropna(how='all')
SVdata = history(SV, fields='close_price', start=start_date, end=end_date, frequency='d
aily')
SVdataDropNa = SVdata.dropna(how='all')
PLdata = history(PL, fields='close_price', start=start_date, end=end_date, frequency='da
ily')
PLdataDropNa = PLdata.dropna(how='all')
SBdata = history(SB, fields='close_price', start=start_date, end=end_date, frequency='d
aily')
SBdataDropNa = SBdata.dropna(how='all')
```

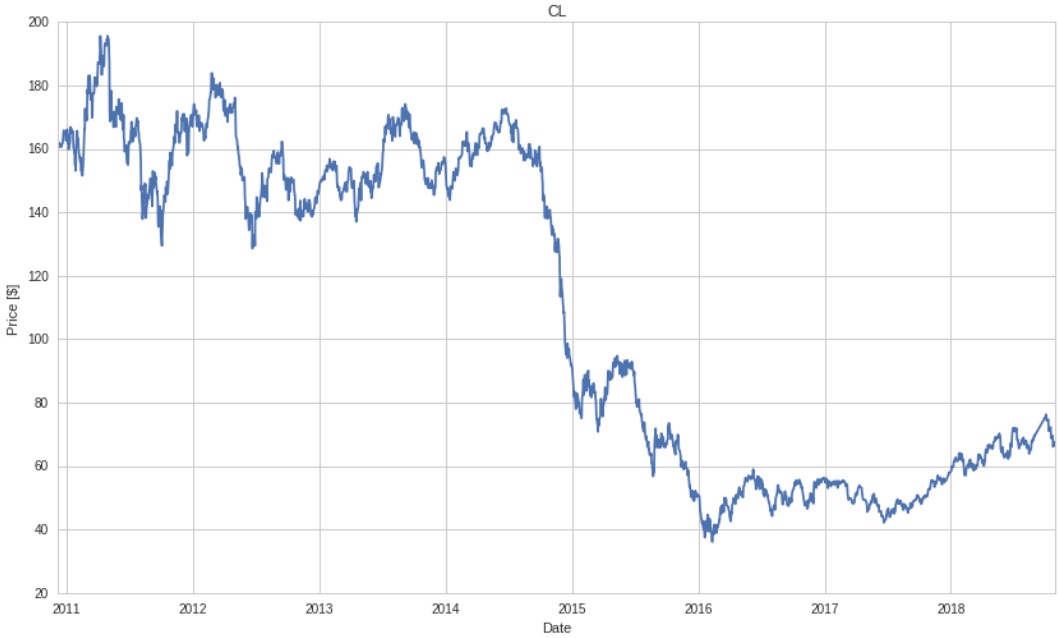
Step #6 - Plot Charts

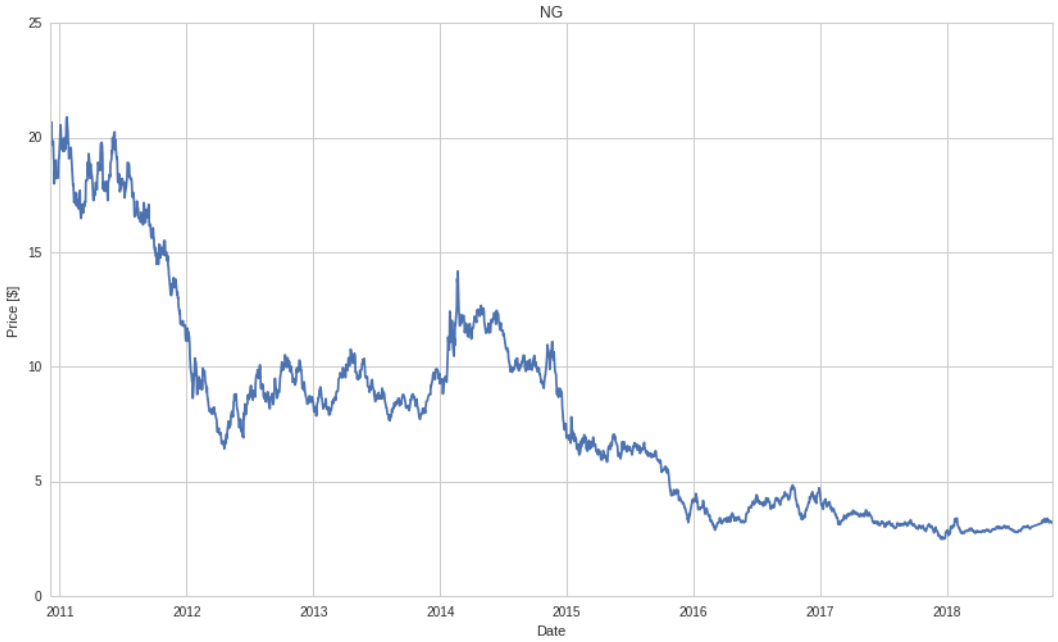
In [XX]:

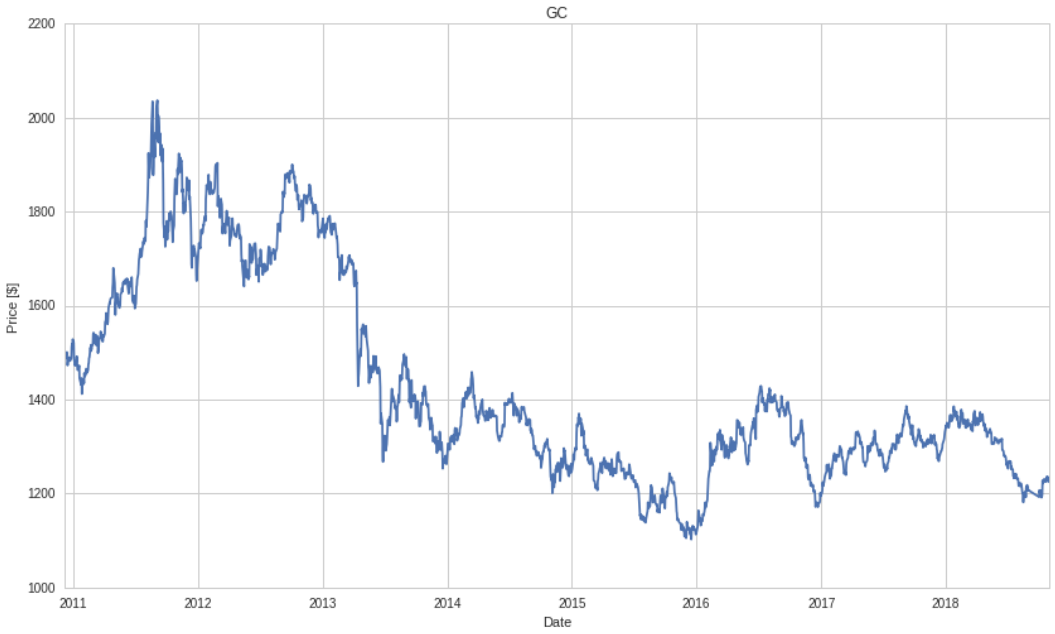
```
plt.plot(CLdataDropNa)
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[0])

plt.figure()
plt.plot(NGdataDropNa)
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[1])

plt.figure()
plt.plot(GCdataDropNa)
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[2])

plt.figure()
plt.plot(XBdataDropNa)
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[3])

plt.figure()
plt.plot(SYdataDropNa)
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[4])

plt.figure()
plt.plot(LCdataDropNa)
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[5])

plt.figure()
plt.plot(HGdataDropNa)
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[6])

plt.figure()
plt.plot(SVdataDropNa)
plt.xlabel('Date')
plt.ylabel('Price [$]')
```

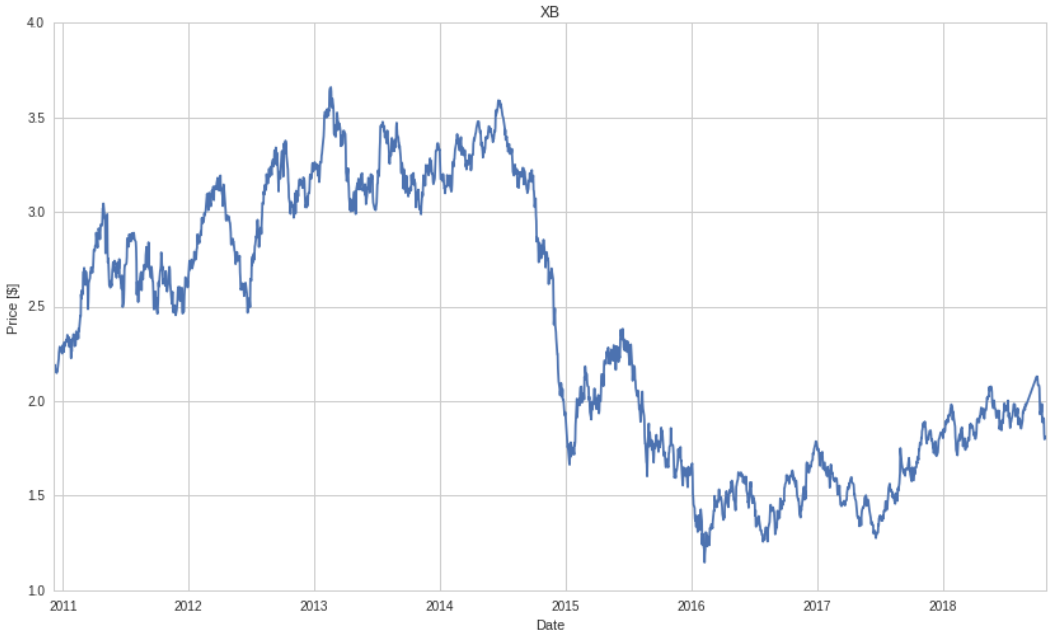
```
plt.title(commodities[7])

plt.figure()
plt.plot(PLdataDropNa)
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[8])

plt.figure()
plt.plot(SBdataDropNa)
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[9])
```

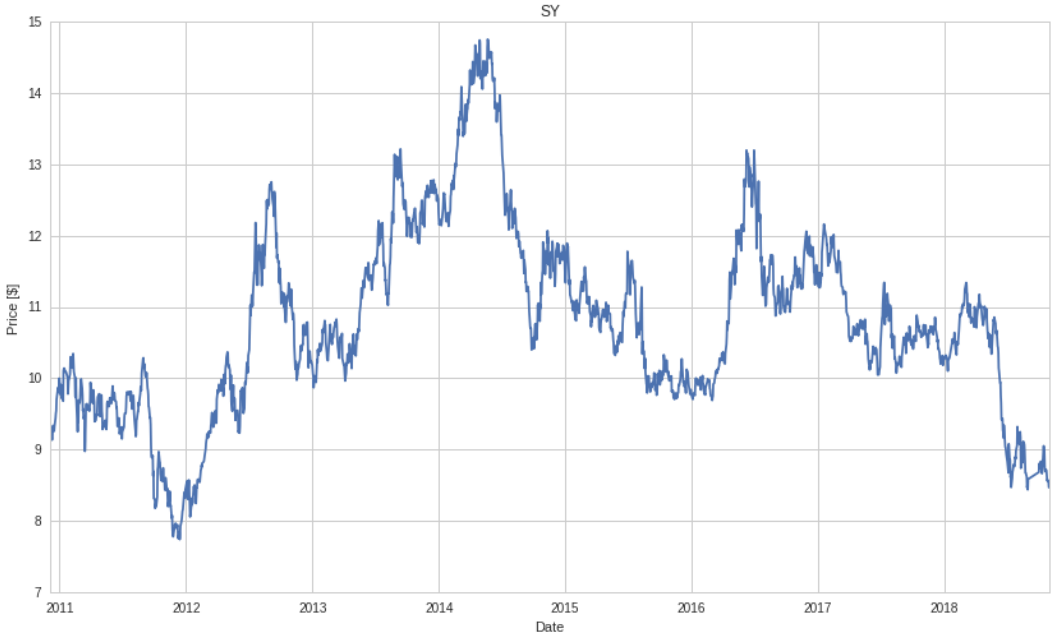
Out [XX]: <matplotlib.text.Text at 0x7fc64d759e50>

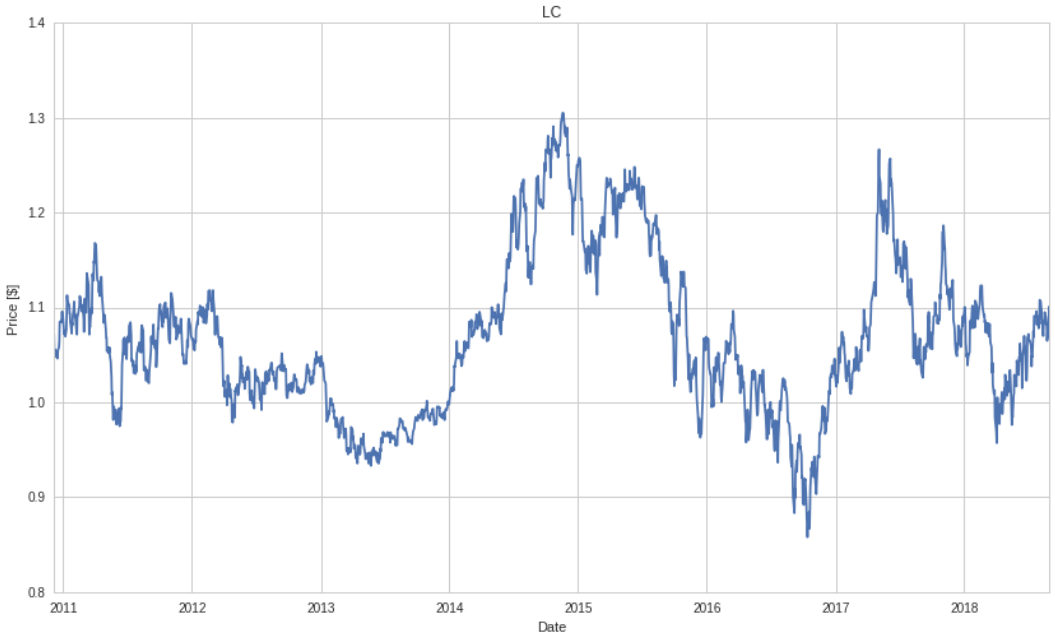


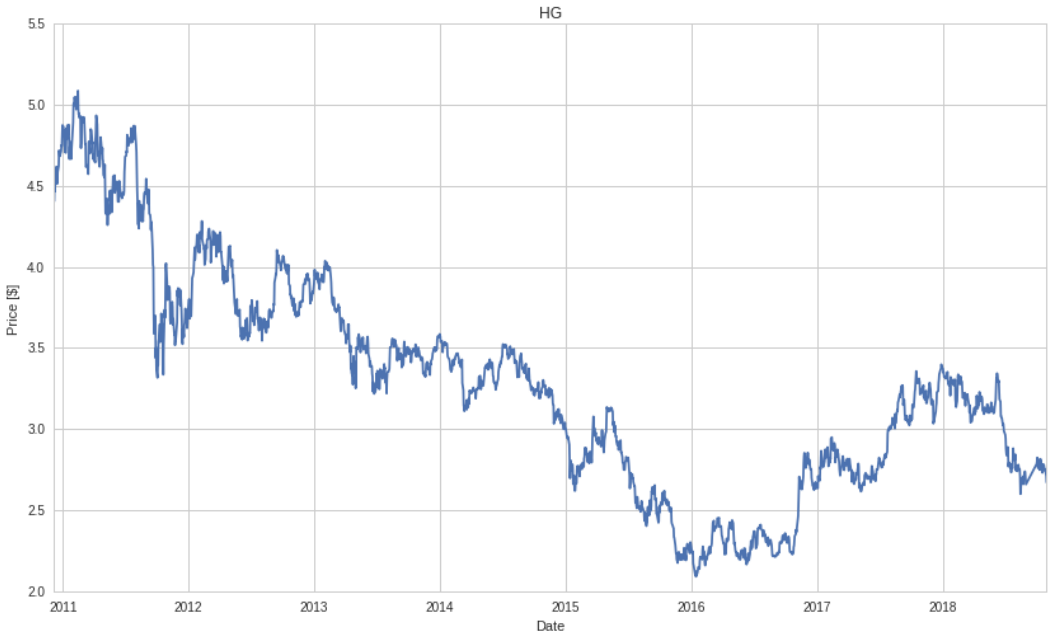

NG
Price [$]
25
20
15
10
5
0
2011
2012
2013
2014
2015
2016
2017
2018
Date

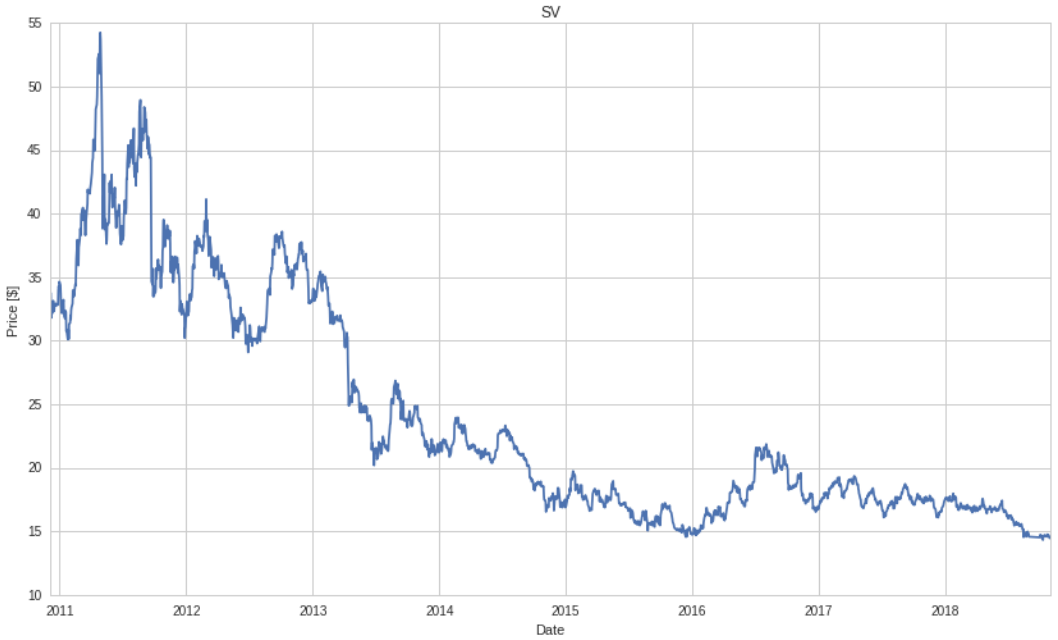

GC
Price [$]
2200
2000
1800
1600
1400
1200
1000
2011
2012
2013
2014
2015
2016
2017
2018
Date


XB
Price [$]
4.0
3.5
3.0
2.5
2.0
1.5
1.0
2011
2012
2013
2014
2015
2016
2017
2018
Date

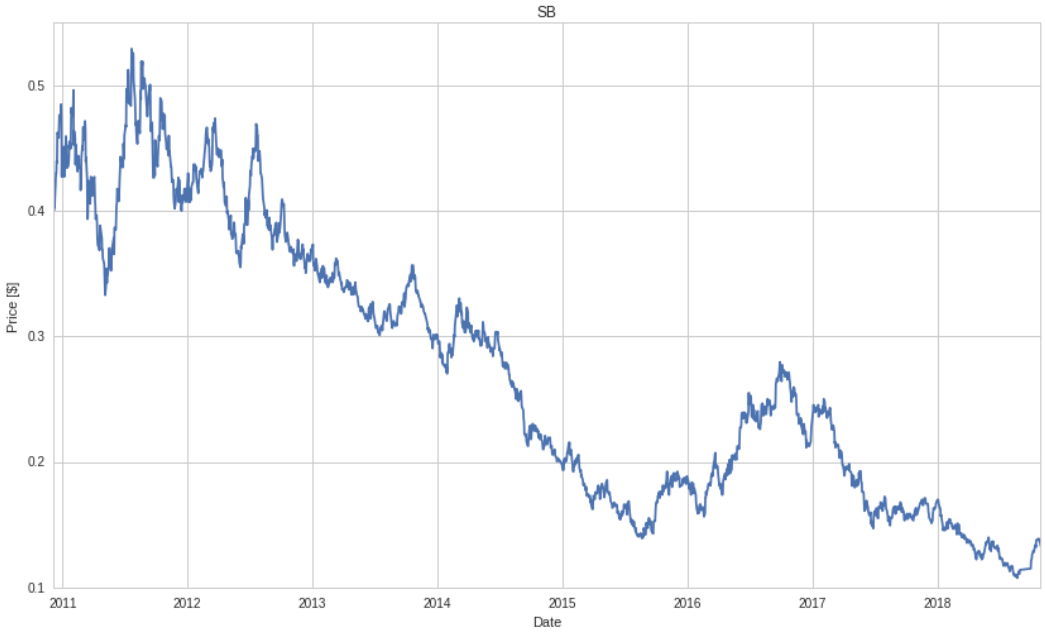

SY
Price [$]
15
14
13
12
11
10
9
8
7
2011
2012
2013
2014
2015
2016
2017
2018
Date


LC
Price [$]
Date
1.4
1.3
1.2
1.1
1.0
0.9
0.8
2011
2012
2013
2014
2015
2016
2017
2018


HG
Price [$]
Date
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
2011
2012
2013
2014
2015
2016
2017
2018


SV
55
50
45
40
35
30
25
20
15
10
Price [$]
2011
2012
2013
2014
2015
2016
2017
2018
Date


PL
2200
2000
1800
1600
1400
1200
1000
800
Price [$]
2011
2012
2013
2014
2015
2016
2017
2018
Date


SB
Price [$]
0.5
0.4
0.3
0.2
0.1
2011
2012
2013
2014
2015
2016
2017
2018
Date

Use case 2 - Technical Analysis

This use case shows how to implement moving average (technical analysis) to chart. There are two primary methods used to analyze financial assets and make investment decisions: fundamental analysis and technical analysis. Fundamental analysis involves analyzing a company's financial statements to determine the fair value of the business, while technical analysis assumes that a security's price already reflects all publicly-available information and instead focuses on the statistical analysis of price movements.

Moving average is a technical indicator that can help you to determine trend and smooth price action.

You can find more information about analysis here:

https://www.investopedia.com/university/technical/

You can find more information about moving average here:

https://www.investopedia.com/terms/m/movingaverage.asp

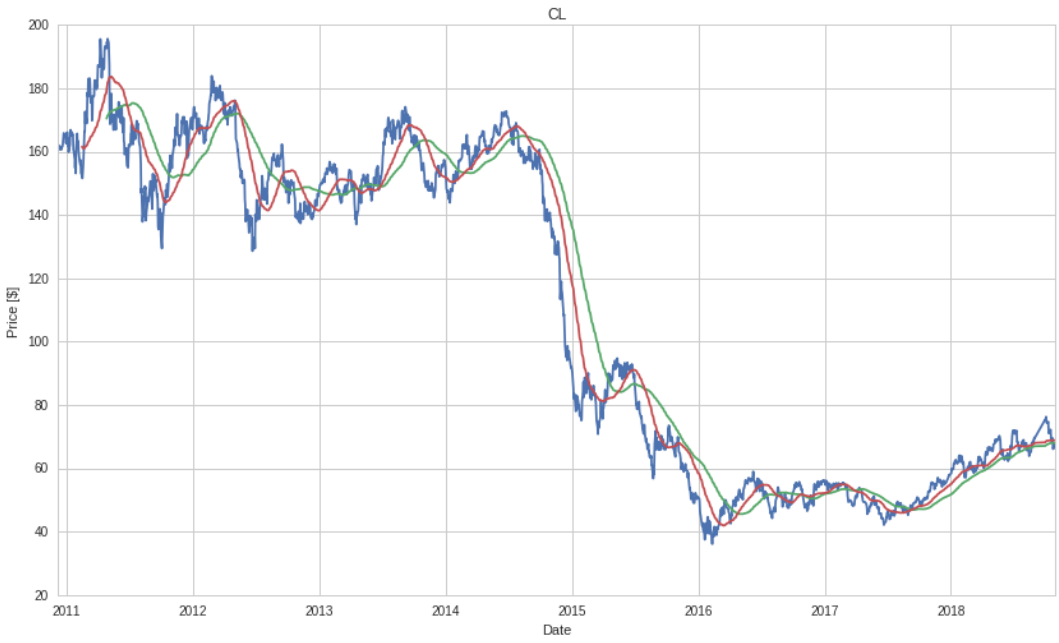
In [XX]:

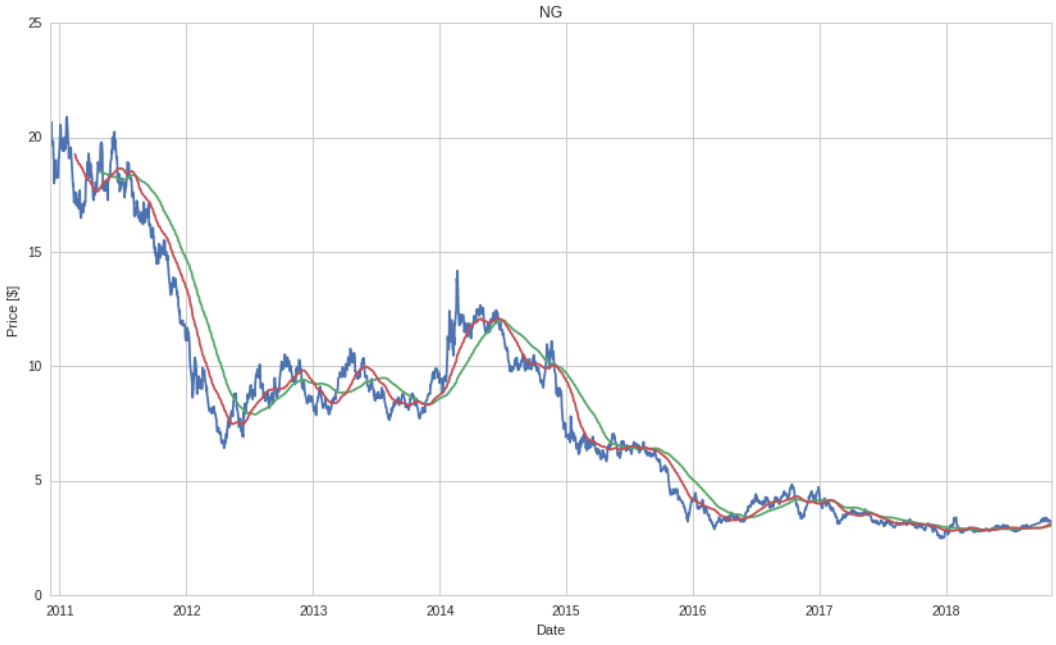
```
plt.plot(CLdataDropNa)
plt.plot(CLdataDropNa.rolling(window=100).mean())
plt.plot(CLdataDropNa.rolling(window=50).mean())
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[0])

plt.figure()
plt.plot(NGdataDropNa)
plt.plot(NGdataDropNa.rolling(window=100).mean())
plt.plot(NGdataDropNa.rolling(window=50).mean())
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[1])

plt.figure()
plt.plot(GCdataDropNa)
plt.plot(GCdataDropNa.rolling(window=100).mean())
plt.plot(GCdataDropNa.rolling(window=50).mean())
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[2])
```

```
plt.figure()
plt.plot(XBdataDropNa)
plt.plot(XBdataDropNa.rolling(window=100).mean())
plt.plot(XBdataDropNa.rolling(window=50).mean())
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[3])

plt.figure()
plt.plot(SYdataDropNa)
plt.plot(SYdataDropNa.rolling(window=100).mean())
plt.plot(SYdataDropNa.rolling(window=50).mean())
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[4])

plt.figure()
plt.plot(LCdataDropNa)
plt.plot(LCdataDropNa.rolling(window=100).mean())
plt.plot(LCdataDropNa.rolling(window=50).mean())
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[5])

plt.figure()
plt.plot(HGdataDropNa)
plt.plot(HGdataDropNa.rolling(window=100).mean())
plt.plot(HGdataDropNa.rolling(window=50).mean())
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[6])

plt.figure()
plt.plot(SVdataDropNa)
plt.plot(SVdataDropNa.rolling(window=100).mean())
plt.plot(SVdataDropNa.rolling(window=50).mean())
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[7])

plt.figure()
plt.plot(PLdataDropNa)
plt.plot(PLdataDropNa.rolling(window=100).mean())
plt.plot(PLdataDropNa.rolling(window=50).mean())
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[8])
```

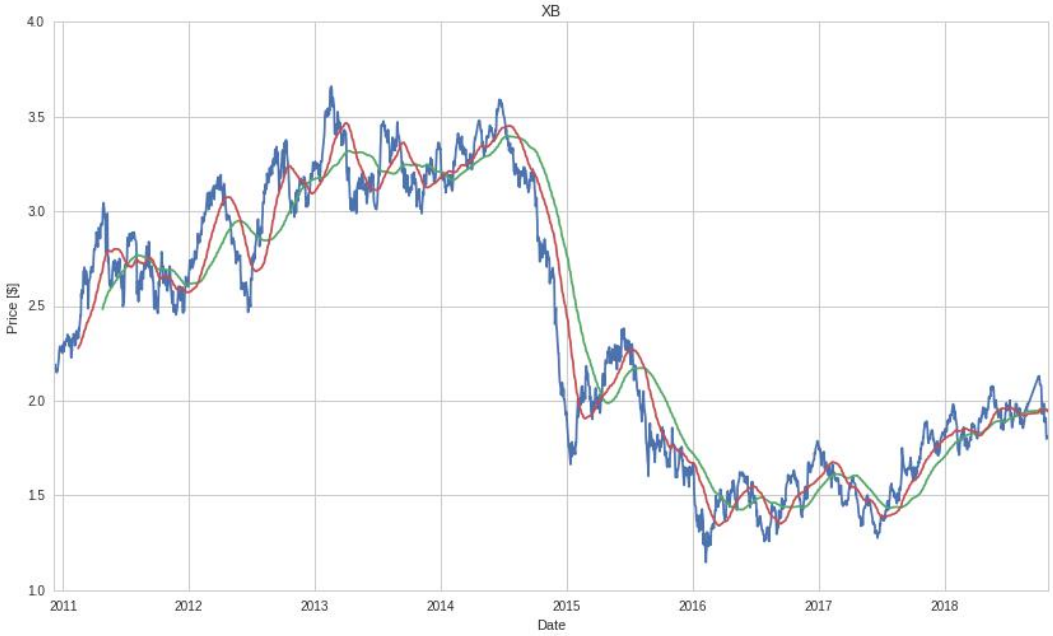
```
plt.figure()
plt.plot(SBdataDropNa)
plt.plot(SBdataDropNa.rolling(window=100).mean())
plt.plot(SBdataDropNa.rolling(window=50).mean())
plt.xlabel('Date')
plt.ylabel('Price [$]')
plt.title(commodities[9])
```

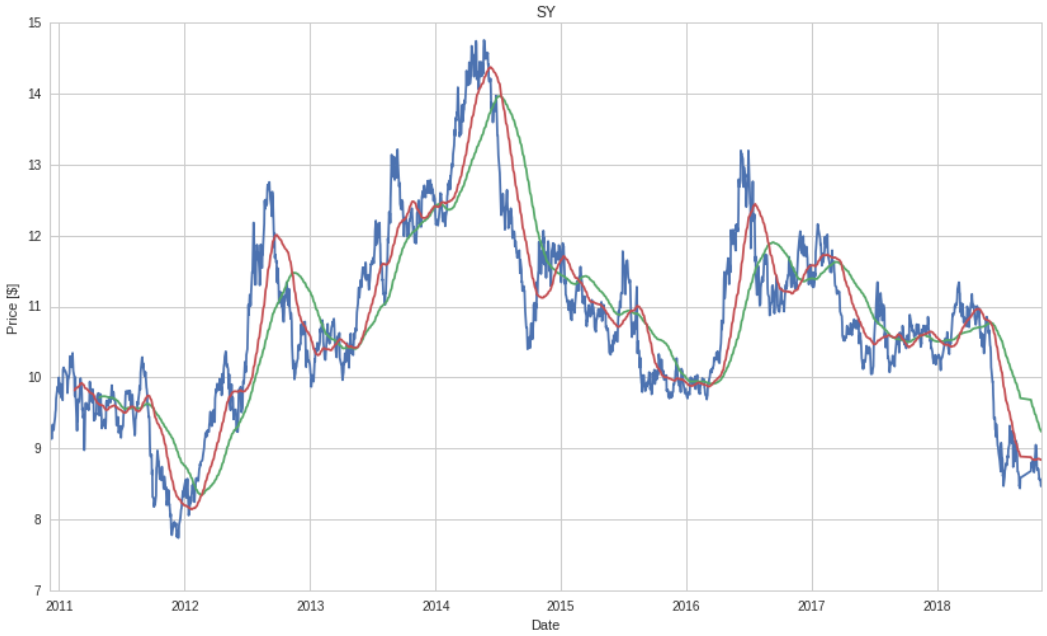
Out[20]: <matplotlib.text.Text at 0x7fc64d012a90>

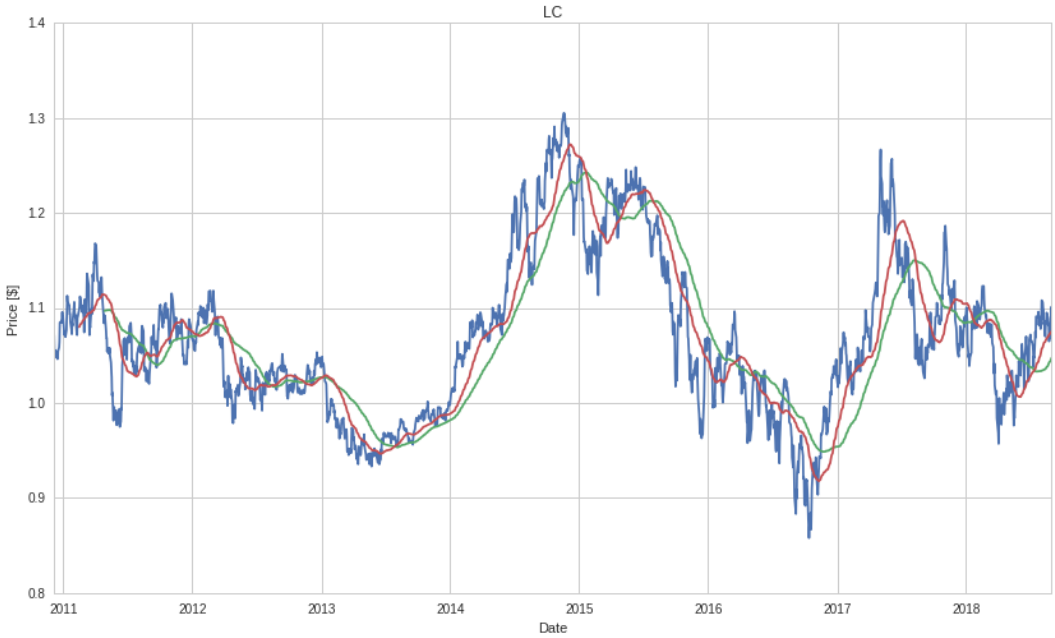


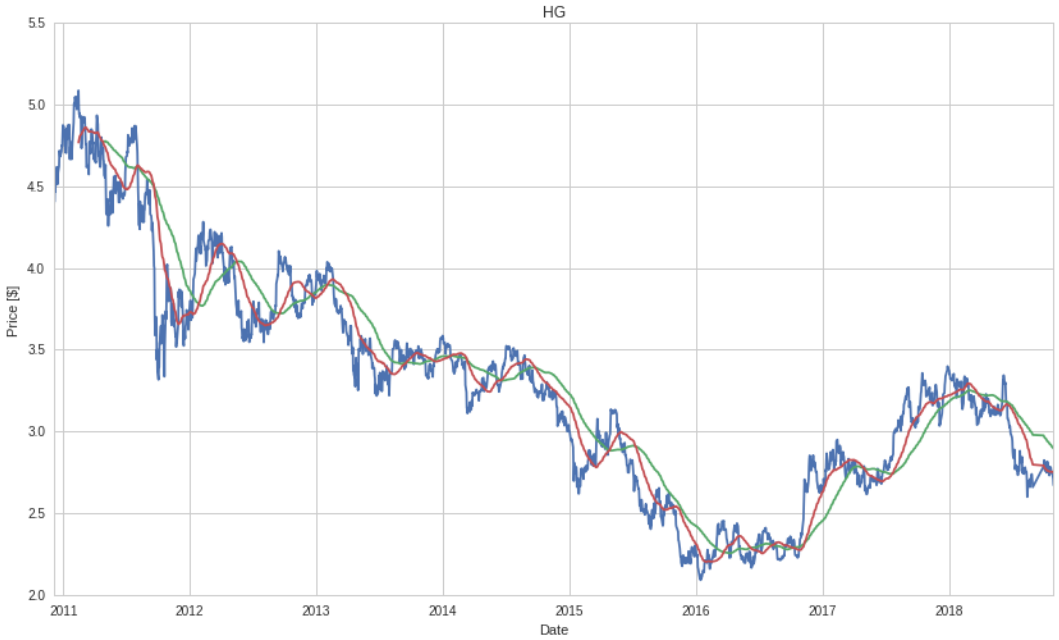


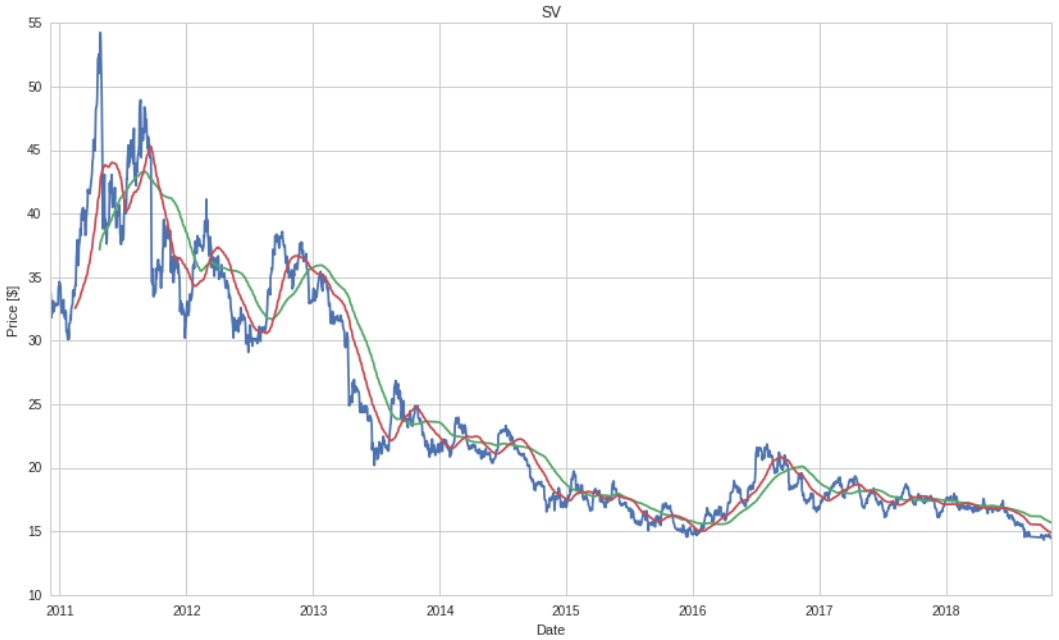

GC
Price [$]
Date
2200
2000
1800
1600
1400
1200
1000
2011
2012
2013
2014
2015
2016
2017
2018


XB
Price [$]
Date
4.0
3.5
3.0
2.5
2.0
1.5
1.0
2011
2012
2013
2014
2015
2016
2017
2018

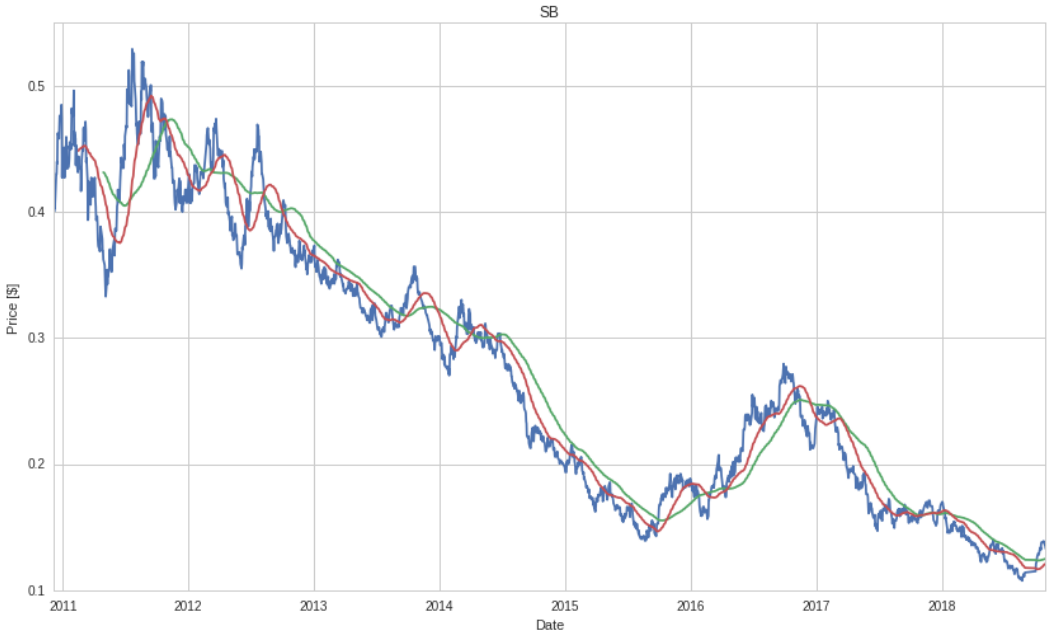

SY
Price [$]
Date
15
14
13
12
11
10
9
8
7
2011
2012
2013
2014
2015
2016
2017
2018


LC
Price [$]
Date
1.4
1.3
1.2
1.1
1.0
0.9
0.8
2011
2012
2013
2014
2015
2016
2017
2018


HG
Price [$]
Date
5.5
5.0
4.5
4.0
3.5
3.0
2.5
2.0
2011
2012
2013
2014
2015
2016
2017
2018


SV
Price [$]
Date
55
50
45
40
35
30
25
20
15
10
2011
2012
2013
2014
2015
2016
2017
2018


PL
2200
2000
1800
1600
1400
1200
1000
800
Price [$]
2011
2012
2013
2014
2015
2016
2017
2018
Date


SB
0.5
0.4
0.3
0.2
0.1
Price [$]
2011
2012
2013
2014
2015
2016
2017
2018
Date

Use case 3 - Statistics

This use case show performance statistics of each commodity. We are analyzing daily percentage performance - mean, standard deviation, maximum and minimum value.

Step #1 - Daily Percentage Changes

You can find more information here:

https://www.investopedia.com/ask/answers/how-do-you-calculate-percentage-gain-or-loss-investment/

In [XX]:

```
CLdata_pct = CLdata.pct_change()*100
NGdata_pct = NGdata.pct_change()*100
GCdata_pct = GCdata.pct_change()*100
XBdata_pct = XBdata.pct_change()*100
SYdata_pct = SYdata.pct_change()*100
LCdata_pct = LCdata.pct_change()*100
HGdata_pct = HGdata.pct_change()*100
SVdata_pct = SVdata.pct_change()*100
PLdata_pct = PLdata.pct_change()*100
SBdata_pct = SBdata.pct_change()*100
```

Step #2 - Mean

You can find more information here:

https://www.investopedia.com/terms/m/mean.asp

In [XX]:

```
CLmean = np.nanmean(CLdata_pct.values[1:])
NGmean = np.nanmean(NGdata_pct.values[1:])
GCmean = np.nanmean(GCdata_pct.values[1:])
XBmean = np.nanmean(XBdata_pct.values[1:])
SYmean = np.nanmean(SYdata_pct.values[1:])
LCmean = np.nanmean(LCdata_pct.values[1:])
HGmean = np.nanmean(HGdata_pct.values[1:])
SVmean = np.nanmean(SVdata_pct.values[1:])
```

```
PLmean = np.nanmean(PLdata_pct.values[1:])
SBmean = np.nanmean(SBdata_pct.values[1:])
```

Step #3 - Standard Deviation

You can find more information here:

https://www.investopedia.com/terms/s/standarddeviation.asp

In [XX]:

```
CLstd = np.nanstd(CLdata_pct.values[1:])
NGstd = np.nanstd(NGdata_pct.values[1:])
GCstd = np.nanstd(GCdata_pct.values[1:])
XBstd = np.nanstd(XBdata_pct.values[1:])
SYstd = np.nanstd(SYdata_pct.values[1:])
LCstd = np.nanstd(LCdata_pct.values[1:])
HGstd = np.nanstd(HGdata_pct.values[1:])
SVstd = np.nanstd(SVdata_pct.values[1:])
PLstd = np.nanstd(PLdata_pct.values[1:])
SBstd = np.nanstd(SBdata_pct.values[1:])
```

Step #4 - Maximum / Minimum

In [XX]:

```
CLmax = np.nanmax(CLdata_pct.values[1:])
NGmax = np.nanmax(NGdata_pct.values[1:])
GCmax = np.nanmax(GCdata_pct.values[1:])
XBmax = np.nanmax(XBdata_pct.values[1:])
SYmax = np.nanmax(SYdata_pct.values[1:])
LCmax = np.nanmax(LCdata_pct.values[1:])
HGmax = np.nanmax(HGdata_pct.values[1:])
SVmax = np.nanmax(SVdata_pct.values[1:])
PLmax = np.nanmax(PLdata_pct.values[1:])
SBmax = np.nanmax(SBdata_pct.values[1:])
```

```
CLmin = np.nanmin(CLdata_pct.values[1:])
NGmin = np.nanmin(NGdata_pct.values[1:])
GCmin = np.nanmin(GCdata_pct.values[1:])
XBmin = np.nanmin(XBdata_pct.values[1:])
SYmin = np.nanmin(SYdata_pct.values[1:])
LCmin = np.nanmin(LCdata_pct.values[1:])
HGmin = np.nanmin(HGdata_pct.values[1:])
SVmin = np.nanmin(SVdata_pct.values[1:])
PLmin = np.nanmin(PLdata_pct.values[1:])
SBmin = np.nanmin(SBdata_pct.values[1:])
```

Step #5 - Statistical Data Created by Pandas Dataframe

You can find more information here:

https://www.geeksforgeeks.org/python-pandas-dataframe/

In [XX]:

```
df = pd.DataFrame({
   'Mean' : [CLmean, NGmean, GCmean, XBmean, SYmean, LCmean, HGmean, SVmean,
PLmean, SBmean],
   'Standard Deviation' : [CLstd, NGstd, GCmean, XBstd, SYstd, LCstd, HGstd, SVstd, PLst
d, SBstd],
   'Max' : [CLmax, NGmax, GCmax, XBmax, SYmax, LCmax, HGmax, SVmax, PLmax, SB
max],
   'Min' : [CLmin, NGmin, GCmin, XBmin, SYmin, LCmin, HGmin, SVmin, PLmin, SBmin],
},index=['Light Sweet Crude Oil', 'Natural Gas', 'Gold', 'RBOB Gasoline Futures','Soybeans'
,'Live Cattle','Copper High Grade','Silver','Platinum', 'Sugar'])
df
```

Out[12]:

	Max	Mean	Min	Standard Deviation
Light Sweet Crude Oil	10.229008	-0.025291	-8.239388	1.955324
Natural Gas	15.432546	-0.061063	-12.324416	2.453626
Gold	4.748879	-0.006201	-8.392102	-0.006201
RBOB Gasoline Futures	7.973606	0.006678	-9.160752	1.790043
Soybeans	5.326121	0.003466	-6.001218	1.255924

	Max	Mean	Min	Standard Deviation
Live Cattle	3.251232	0.004824	-2.997275	0.999764
Copper High Grade	6.369098	-0.015849	-7.046517	1.336737
Silver	6.510819	-0.026394	-13.593540	1.755619
Platinum	4.320099	-0.033689	-5.529984	1.175626
Sugar	7.803650	-0.040690	-9.816339	1.815274

Step #6 - Correlation between Crude Oil and Other Commodities

Correlation is a very important statistical value on financial markets. Each rational investor should create a diversified investment portfolio which contains non-correlated assets.

You can find more information about correlation here:

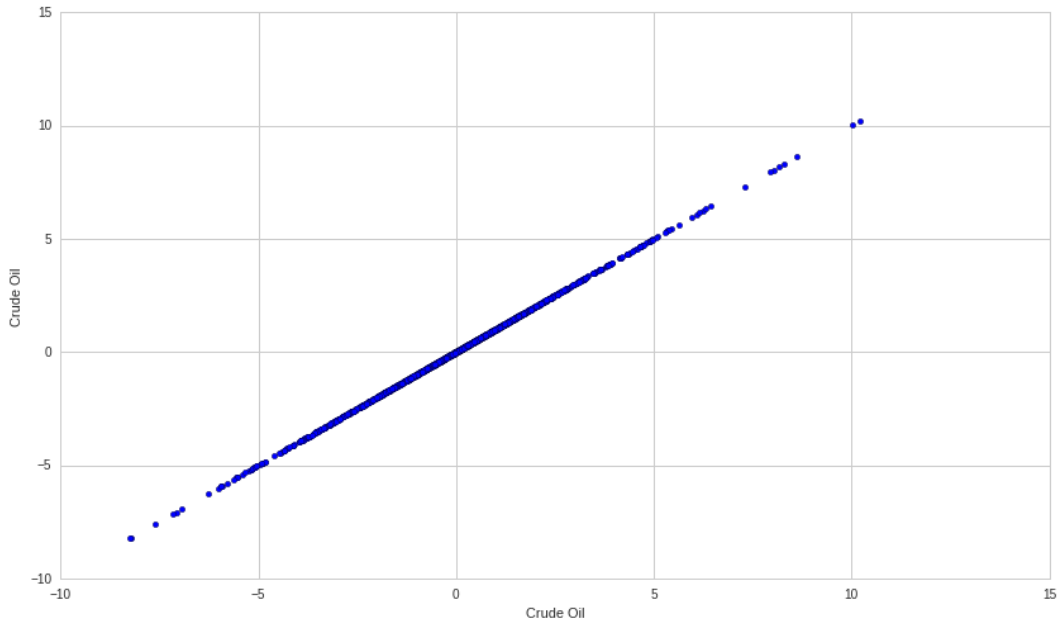
https://www.investopedia.com/terms/c/correlation.asp

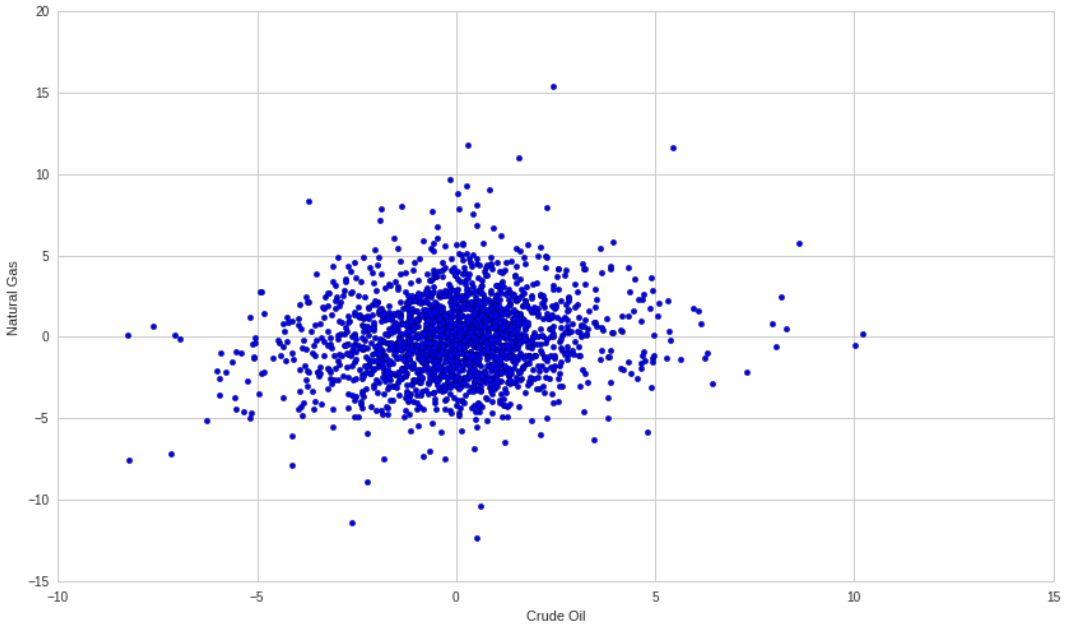
In [XX]:

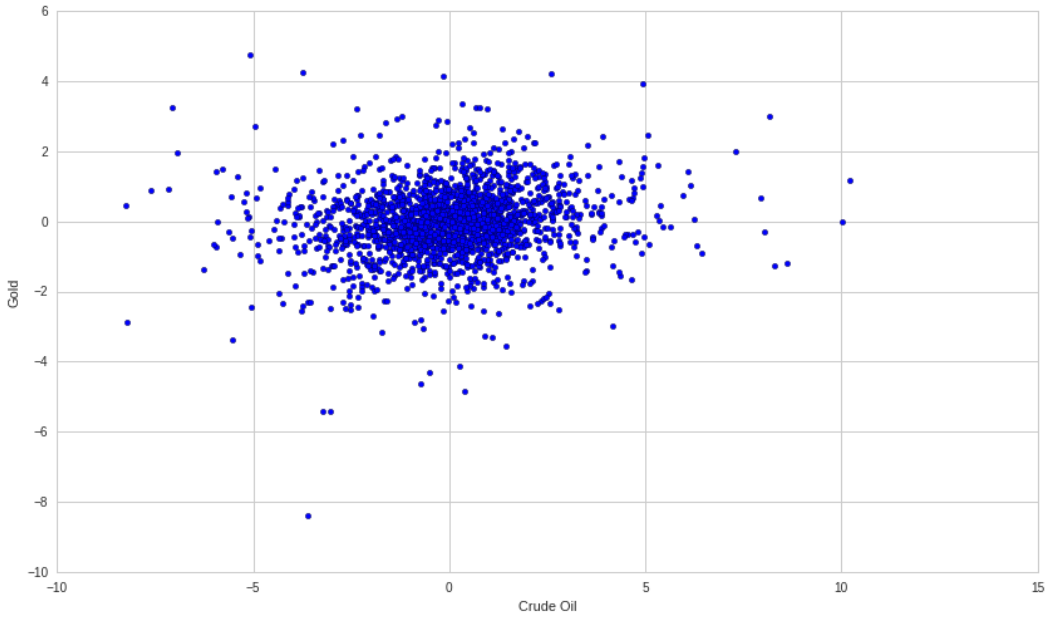
```
plt.scatter(CLdata_pct.values[1:], CLdata_pct.values[1:])
plt.xlabel('Crude Oil')
plt.ylabel('Crude Oil')

plt.figure()
plt.scatter(CLdata_pct.values[1:], NGdata_pct.values[1:])
plt.xlabel('Crude Oil')
plt.ylabel('Natural Gas')

plt.figure()
plt.scatter(CLdata_pct.values[1:], GCdata_pct.values[1:])
plt.xlabel('Crude Oil')
plt.ylabel('Gold')

plt.figure()
plt.scatter(CLdata_pct.values[1:], XBdata_pct.values[1:])
plt.xlabel('Crude Oil')
plt.ylabel('RBOB Gasoline Futures')
```

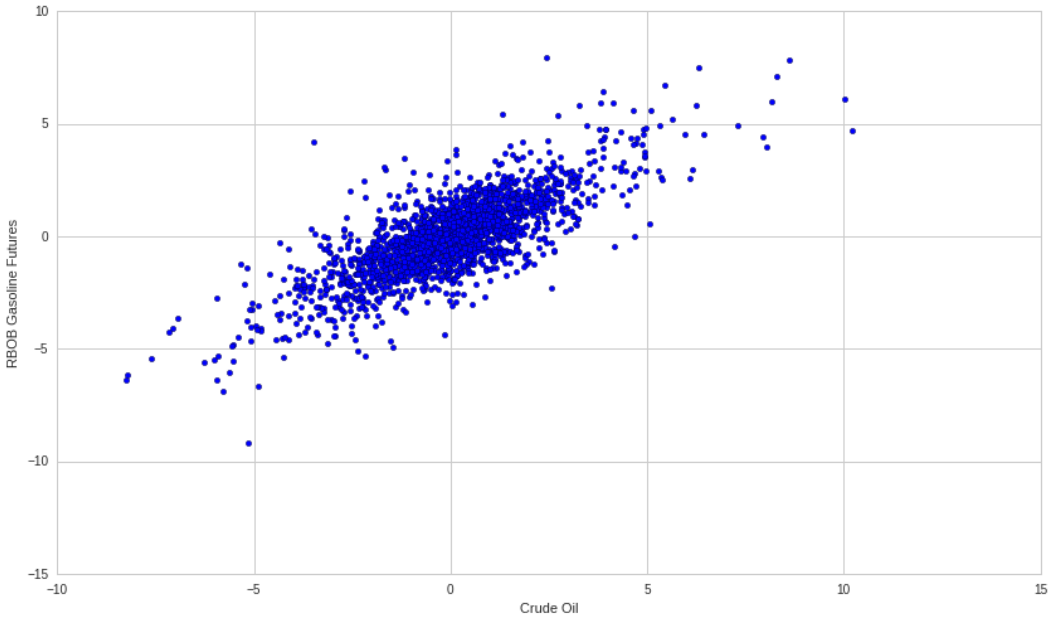
```
plt.figure()
plt.scatter(CLdata_pct.values[1:], SYdata_pct.values[1:])
plt.xlabel('Crude Oil')
plt.ylabel('Soybeans')

plt.figure()
plt.scatter(CLdata_pct.values[1:],LCdata_pct.values[1:])
plt.xlabel('Crude Oil')
plt.ylabel('Live Cattle')

plt.figure()
plt.scatter(CLdata_pct.values[1:], SVdata_pct.values[1:])
plt.xlabel('Crude Oil')
plt.ylabel('Silver')

plt.figure()
plt.scatter(CLdata_pct.values[1:], PLdata_pct.values[1:])
plt.xlabel('Crude Oil')
plt.ylabel('Platinum')

plt.figure()
plt.scatter(CLdata_pct.values[1:], SBdata_pct.values[1:])
plt.xlabel('Crude Oil')
plt.ylabel('Sugar')
```

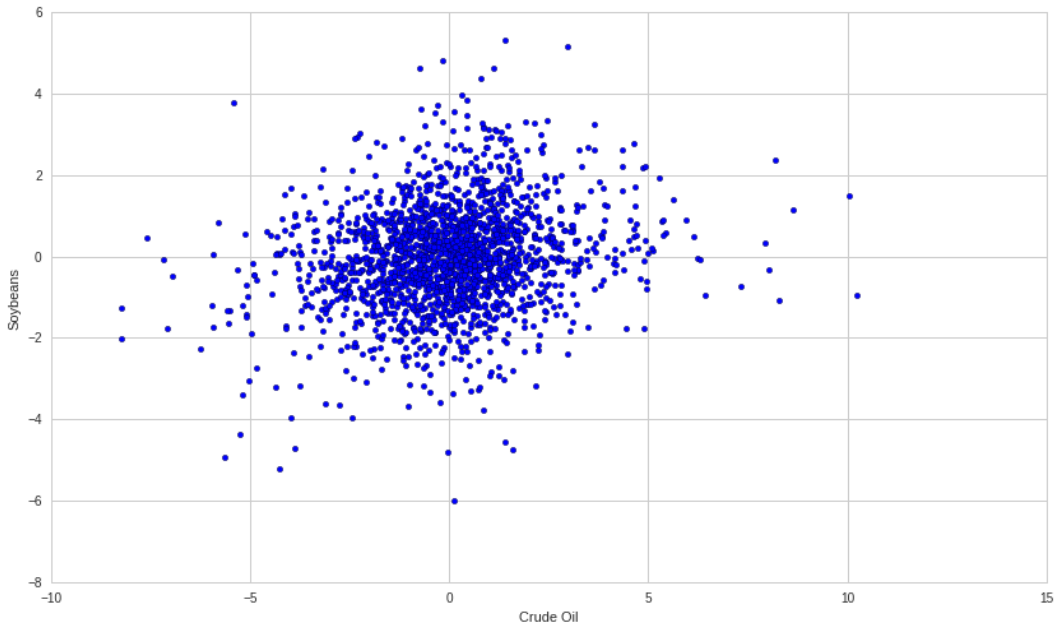
Out[26]: <matplotlib.text.Text at 0x7fc661c0f610>

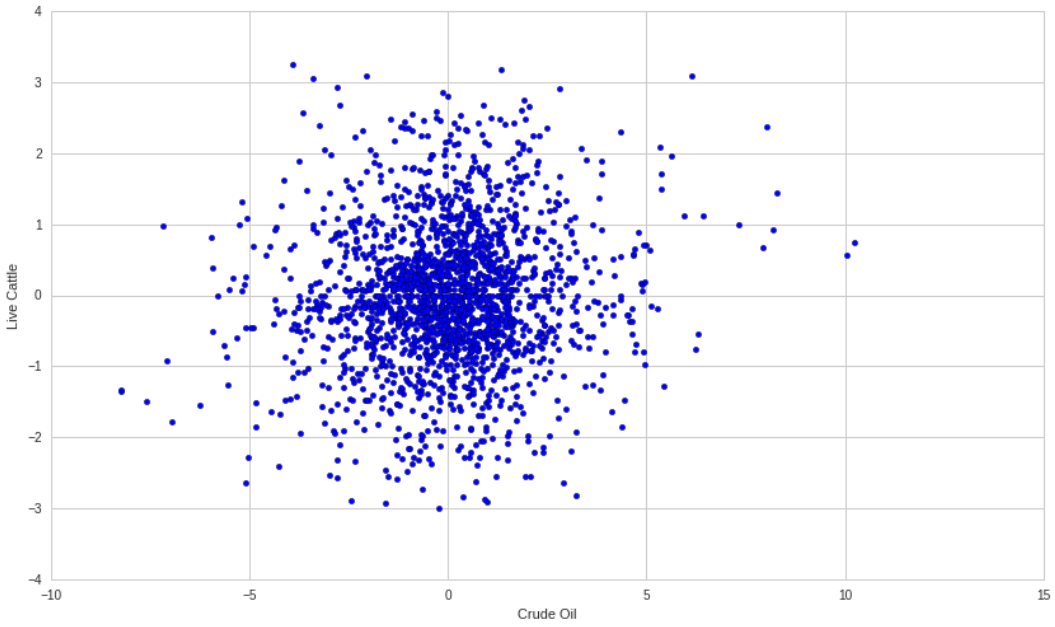


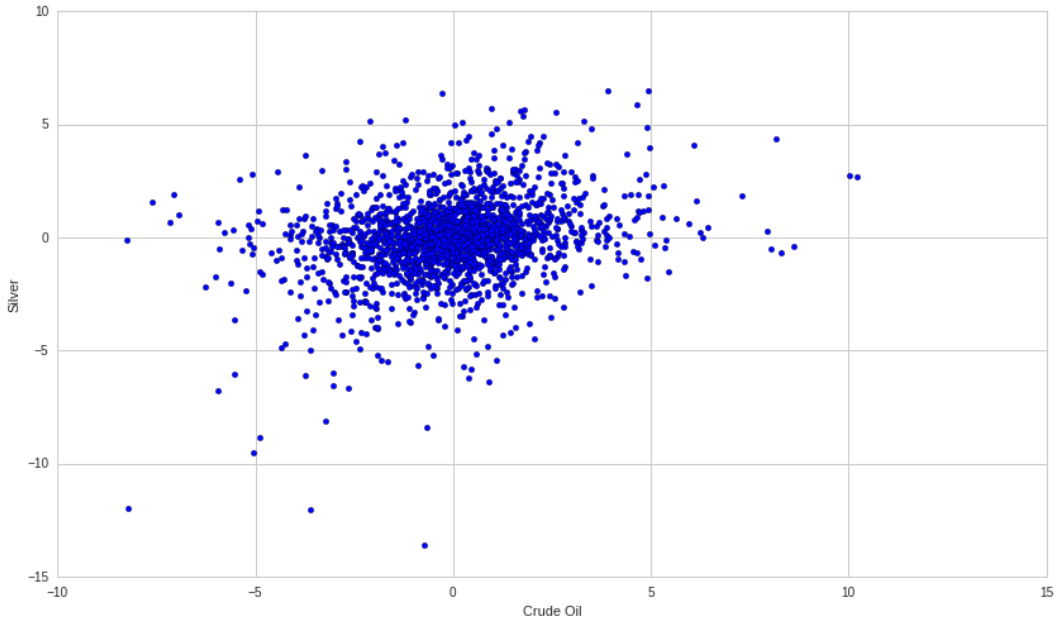

Natural Gas
Crude Oil
20
15
10
5
0
−5
−10
−15
−10
−5
0
5
10
15

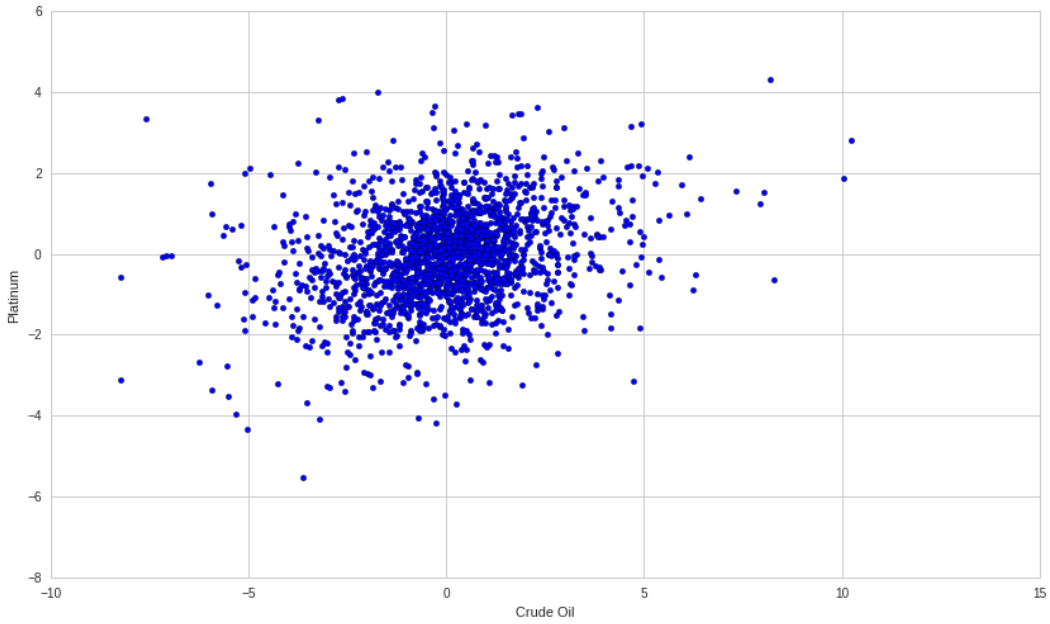

Gold
Crude Oil
6
4
2
0
−2
−4
−6
−8
−10
−10
−5
0
5
10
15

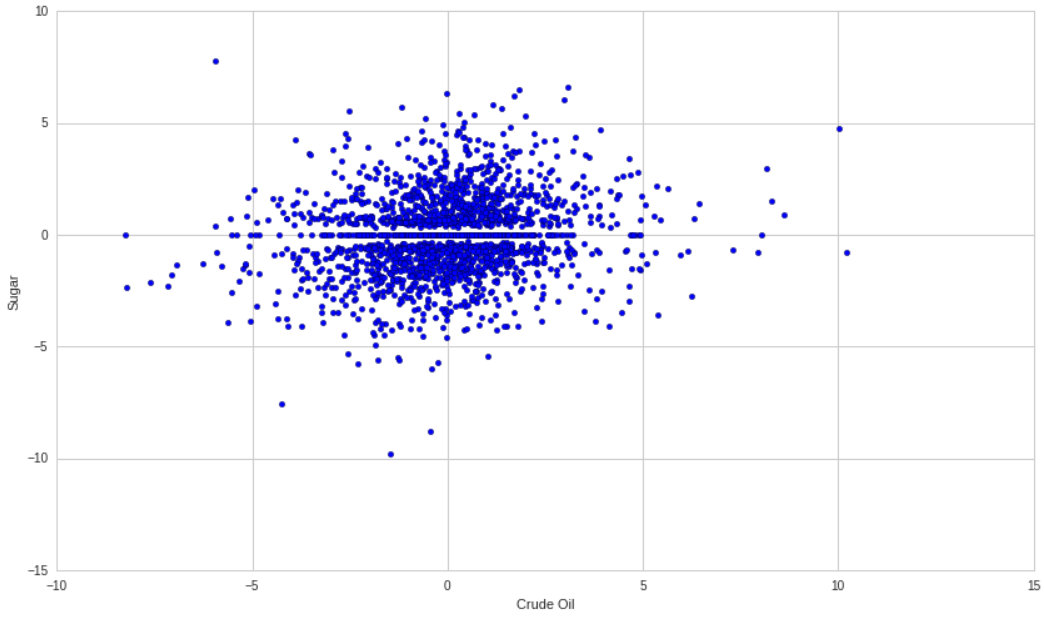

RBOB Gasoline Futures
Crude Oil
10
5
0
−5
−10
−15
−10
−5
0
5
10
15


Soybeans
Crude Oil
6
4
2
0
−2
−4
−6
−8
−10
−5
0
5
10
15


Live Cattle
Crude Oil
4
3
2
1
0
−1
−2
−3
−4
−10
−5
0
5
10
15


Silver
Crude Oil
10
5
0
−5
−10
−15
−10
−5
0
5
10
15


Platinum
Crude Oil
6
4
2
0
−2
−4
−6
−8
−10
−5
0
5
10
15


Sugar
Crude Oil
10
5
0
−5
−10
−15
−10
−5
0
5
10
15

Use case 4 - Investment Strategy Analysis

The main assumption of investment strategy should be a positive performance on historical data. With Quantopian it is possible to develop an exact investment algorithm and easily analyze performance on historical data with very low granularity. Quantopian supports 1-minute granularity for each US stock for free, so you can get a very precise historical performance report of investment strategy.

In this use case we'll show a detailed analysis of investment strategy which bought TESLA in January 2017.

You can find more information about testing on historical data(backtesting) here:

https://www.investopedia.com/terms/b/backtesting.asp

Step #1 - Download Daily Performances for TESLA

We will download TESLA returns and SPY returns as benchmark. SPY is ETF fund that tracks a market-cap-weighted index of US large- and midcap stocks selected by the S&P Committee.

You can find more information about ETF here:

https://www.investopedia.com/terms/e/etf.asp

You can find more information about SPY here:

https://www.etf.com/SPY

In [XX]:

```
import pyfolio as pf

stock = 'TSLA'
start_day = '2010-01-01'
end_day = '2019-06-11'

tesla_rets = get_pricing(stock, start_date=start_day, end_date=end_day, frequency
='daily', fields='close_price').pct_change()[1:]
spy_rets = get_pricing('SPY', start_date=start_day, end_date=end_day, frequency='
daily', fields='close_price').pct_change()[1:]
```

Step #2 – Performance Metrics

You can find performance metrics in the table below. These metrics are usually used for measuring portfolio performance. You can find here e.g. cummulative returns, annual returns, maximum drawdown etc.

You can find more information about metrics here:

https://www.quantopian.com/posts/pyfolio-a-new-python-library-for-performance-and-risk-analysis

In [XX]:

```
pf.show_perf_stats(tesla_rets, spy_rets)
```

Start date	2010-01-05
End date	2019-06-11
Total months	113

	Backtest
Annual return	26.1%
Cumulative returns	789.2%
Annual volatility	51.7%
Sharpe ratio	0.73
Calmar ratio	0.49
Stability	0.80
Max drawdown	-53.5%
Omega ratio	1.14
Sortino ratio	1.11
Skew	NaN
Kurtosis	NaN
Tail ratio	1.05
Daily value at risk	-6.4%
Alpha	0.19
Beta	1.29

Step #3 - Drawdown

A drawdown is a capital decrease during an investment period. A drawdown is usually quoted as the percentage between the peak and the subsequent trough. This value shows how much money you can lose during an investment period.

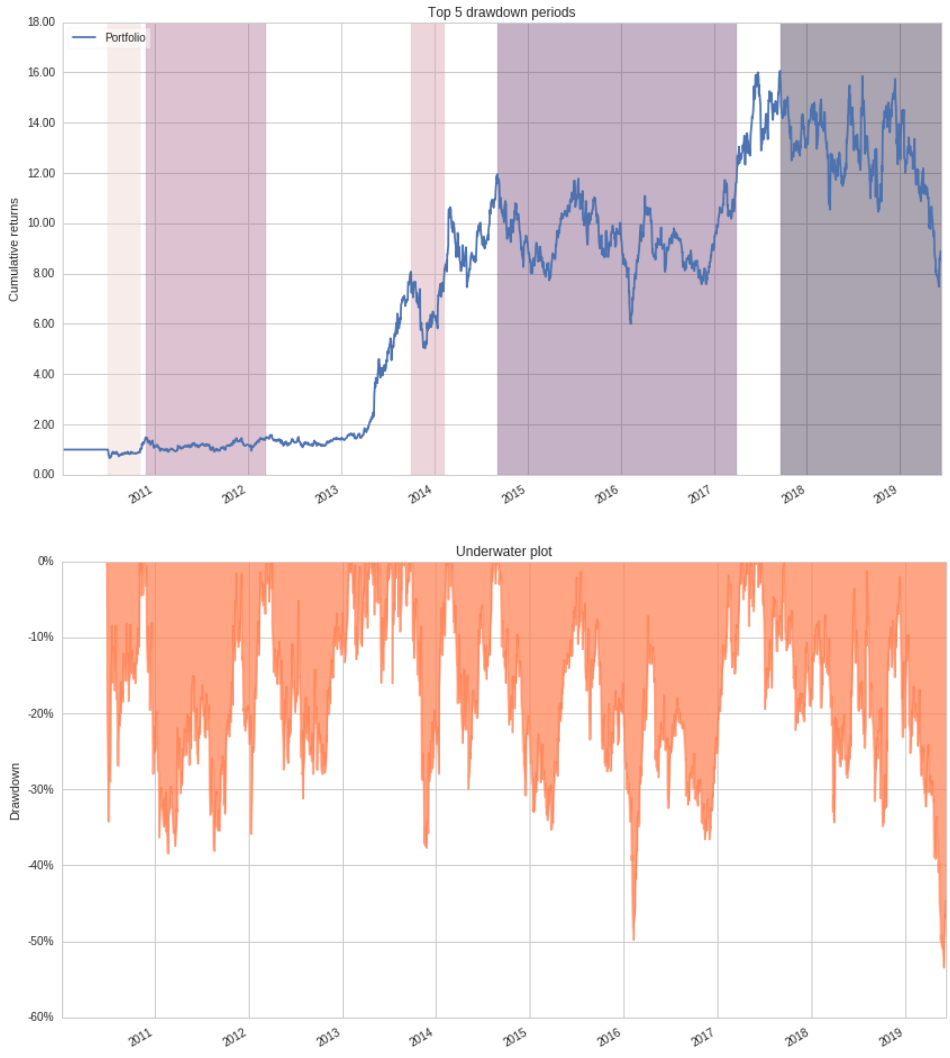
You can find more information about drawdown here:

https://www.investopedia.com/terms/d/drawdown.asp

In [XX]:

```
import matplotlib.pyplot as plt
pf.plot_drawdown_periods(tesla_rets, top=5)
plt.figure()
pf.plot_drawdown_underwater(tesla_rets)
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x7f64c7957cd0>



Step #4 - Annual Returns

Annual return is the return an investment provides over a period of time, expressed as a time-weighted annual percentage. In this use case it's returns per year.

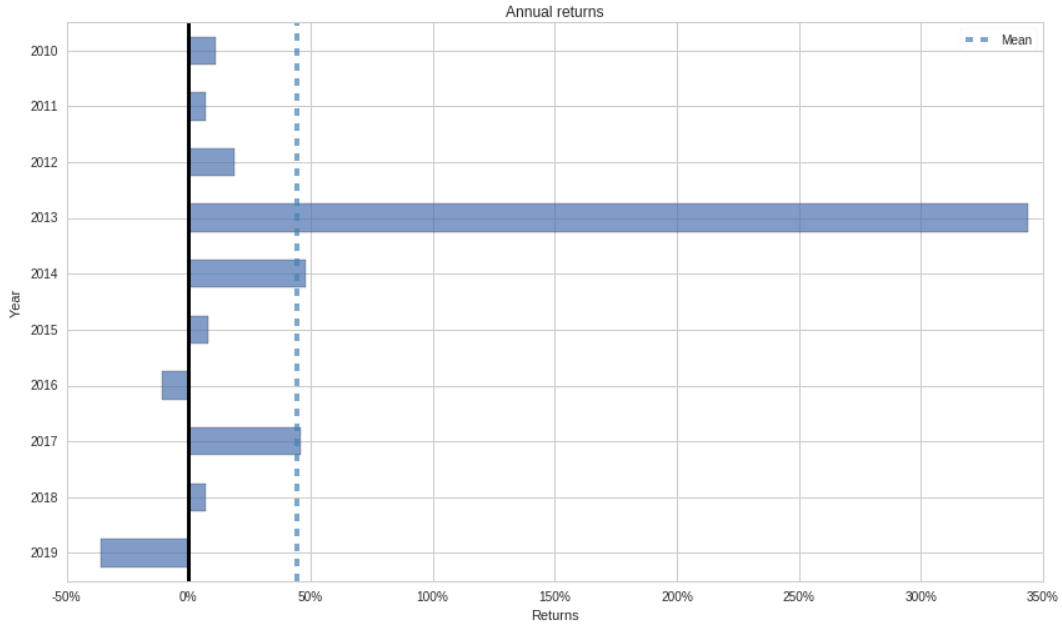
You can find more information about annual returns here:

https://www.investopedia.com/terms/a/annual-return.asp

In [XX]:

```
pf.plot_annual_returns(tesla_rets)
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7f64c6736dd0>



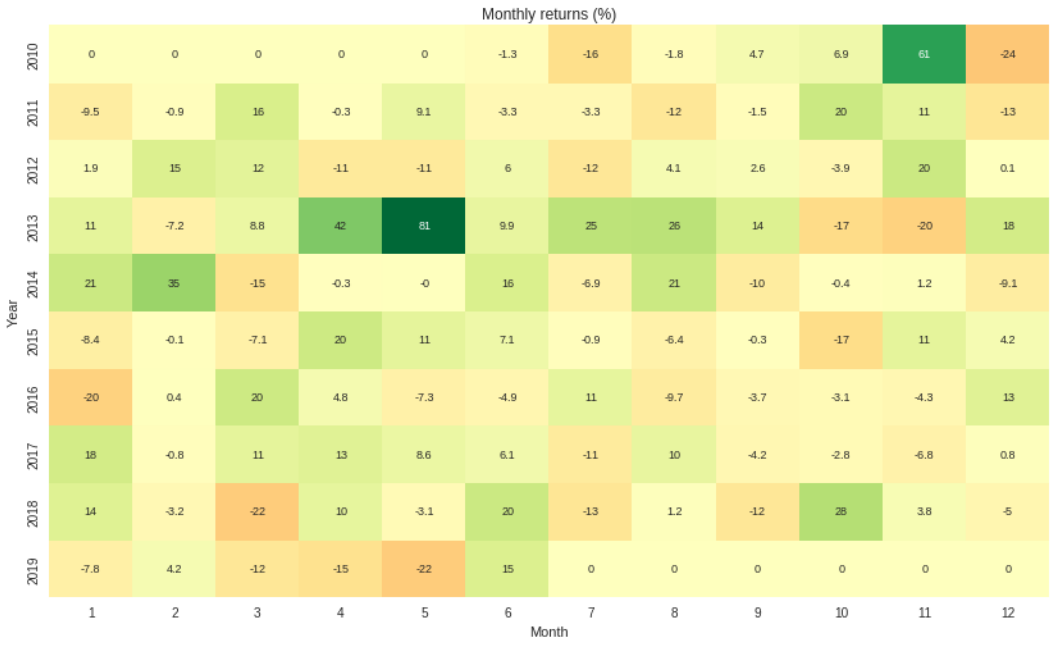
Step #5 - Monthly Returns (%)

Monthly returns heatmap shows stock performance (%) per each month.

In [XX]:

```
pf.plot_monthly_returns_heatmap(tesla_rets)
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7f64c5af67d0>



Step #5 - Cummulative Returns

A cummulative return on an investment is the aggregate amount that the investment has gained or lost over time.

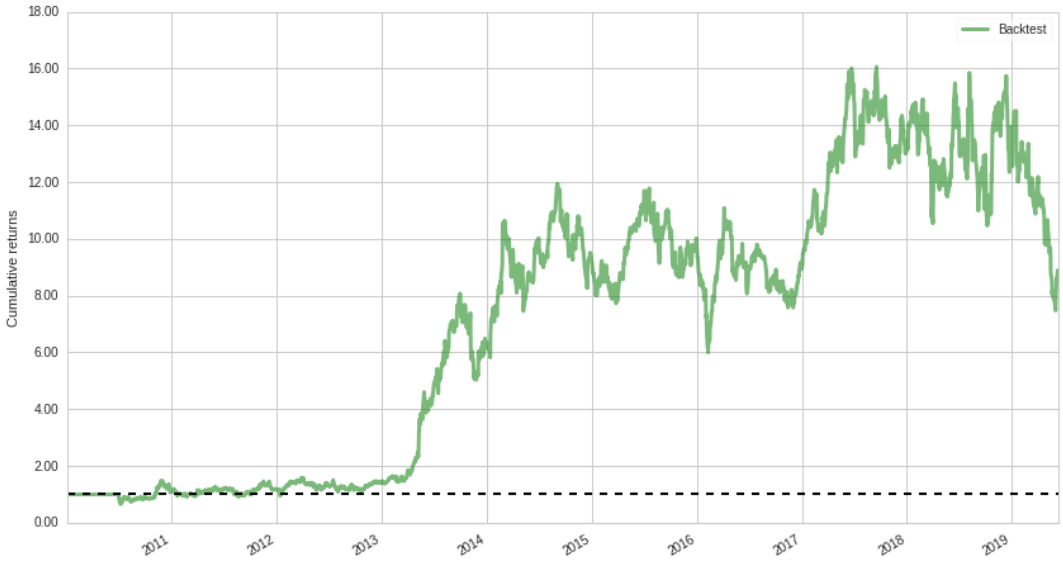
You can find more information about cummulative returns here:

https://www.investopedia.com/terms/c/cumulativereturn.asp

In [XX]:

```
pf.plot_rolling_returns(tesla_rets)
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7f64c58a17d0>



How to Replicate this Study (research)

Try it yourself!

1) Create free Quantopian account:
https://www.quantopian.com/users/sign_up?redirect_to=/posts

2) Login to Quantopian:

https://www.quantopian.com/signin?return_to=/posts

3) Click on Research/Notebooks.

4) Click on blue button with "+" and create new notebook.

5) Copy all source code above.

6) Check this video - how to work with notebook:

https://www.youtube.com/watch?v=W-TlWzwM208

7) Help with all Quantopian functions:

https://www.quantopian.com/help

8) Quantopian tutorials:

https://www.quantopian.com/tutorials/getting-started

Enjoy your results!