Computer Simulations and Risk Assessment – Lecture 3

Fall 2019

Brandeis International Business School

Course Information - Schedule

Class Date	Text Chapters
Aug. 30, 2019 – L1	• Course Introduction/Python Installation • Introduction to Quantitative Finance Career • Python basics
Sep. 6, 2019 – L2	• Advanced Python Topics
Sep. 13, 2019 – L3	• Advanced Python Topics
Sep. 20, 2019 – L4	• Sourcing and handling Data • Stylized financial data analysis using Python
Sep. 27, 2019 – L5	• Value at Risk
Oct. 4, 2019 – L6	• Conditional Value at Risk (Expected Shortfall) + Mid-term Review
Oct. 11, 2019	• Mid-term
Oct. 18, 2019 – L7	• Modeling Volatility I
Oct. 25, 2019 – L8	• Modeling Volatility II
Nov. 1, 2019 – L9	• Practical application case Studies I
Nov. 8, 2019 – L10	• Practical application case Studies II
Nov. 15, 2019 – L11	• Back Testing + Conditional risk prediction
Nov. 22, 2019 – L12	• Research project presentation
Dec. 6, 2019 – L13	• Final Review

Advanced Python Programming

- Data types and structures
- Charting

Python Data Types/Structures

- Python has following commonly used Data Types/Structures:
  - Numbers
  - String
  - List
  - Tuple
  - Series
  - Dictionary
  - Set
  - Datetime
  - Dataframe
- Additional data types store text, integer or single-precision values;
- or a combination of related data in a single variable.

A note on referencing data

- The Python and NumPy indexing operators [] and attribute operator '.' provide quick and easy access to data structures across a wide range of use cases:
- E.g, A[0,1] reference the element at first row, second column of A
- A.color reference the attribute color in variable A
- More to follow on these later this lecture

Python Data Types – Datetimes

class `datetime.`**`date`**

An idealized naive date, assuming the current Gregorian calendar always was, and always will be, in effect. Attributes: `year`, `month`, and `day`.

class `datetime.`**`time`**

An idealized time, independent of any particular day, assuming that every day has exactly 24*60*60 seconds (there is no notion of "leap seconds" here). Attributes: `hour`, `minute`, `second`, `microsecond`, and `tzinfo`.

class `datetime.`**`datetime`**

A combination of a date and a time. Attributes: `year`, `month`, `day`, `hour`, `minute`, `second`, `microsecond`, and `tzinfo`.

class `datetime.`**`timedelta`**

A duration expressing the difference between two `date`, `time`, or `datetime` instances to microsecond resolution.

class `datetime.`**`tzinfo`**

An abstract base class for time zone information objects. These are used by the `datetime` and `time` classes to provide a customizable notion of time adjustment (for example, to account for time zone and/or daylight saving time).

class `datetime.`**`timezone`**

A class that implements the `tzinfo` abstract base class as a fixed offset from the UTC.

- Example

  from datetime import datetime
  datetime.now().isoformat(timespec='minutes')
  dt = datetime(2015, 1, 1, 12, 30, 59, 0)
  datetime(2018, 3, 18)

Python Data Types – Characters and Strings

- Creating character strings
  - *myString = 'Hello, world'*
  - *otherString = 'You''re right'*

- Creating and referencing an array of strings
  - *arr = np.chararray((3, 2))*
  - *arr[0,0] ='aa'*

Python Data Types – Series

- Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index

- The basic method to create a Series is to call:
  - s = pd.Series(data, index=index)

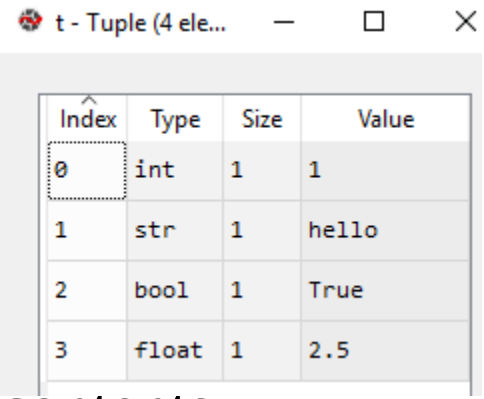
- Series acts very similarly to a ndarray, and is a valid argument to most NumPy functions
  - S[0]
  - S[:3]

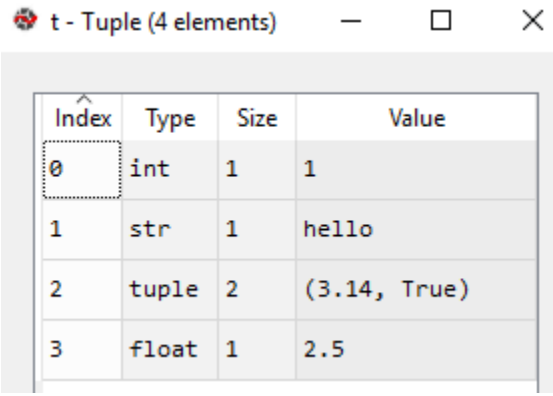
- Elements of a series can not be modified

- By taking a single column of a dataframe it creates a series datatype

Python Data Types – Tuple

- A tuple is a sequence of immutable Python objects, which means tuples they cannot be changed once defined
- Tuples are declared by using brackets () that contains different types of data in one variable. Or you can define it without the () at all.
- Use comma(,) to separate different components of a list
- Example
  - t = (1, 'hello ', True , 2.5)
  - t1 = (1, 'hello ', (3.14 , True), 2.5) # tuple with another tuple as one of its element

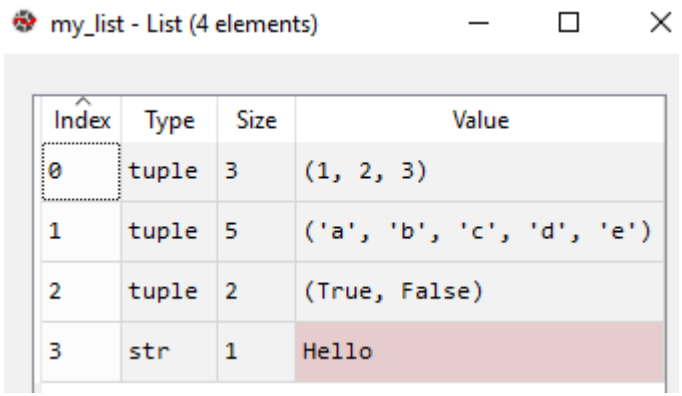




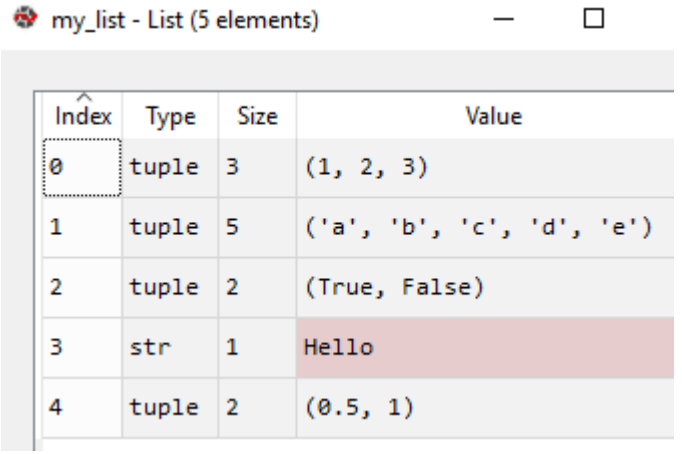
- Handle Tuple contents
  - Indexing using square bracket []: *t1[2][1] # referencing tuples - gives "True"*

Python Data Types – List

- Lists are declared by using brackets [] that contains different types of data in one variable
- Use comma(,) to separate different components of a list
- Example
  - my_list = [(1, 2, 3), ('a', 'b', 'c', 'd', 'e'), (True, False), 'Hello']
  - This actually creates a list with three tuples and one string as its content



Index	Type	Size	Value
0	tuple	3	(1, 2, 3)
1	tuple	5	('a', 'b', 'c', 'd', 'e')
2	tuple	2	(True, False)
3	str	1	Hello

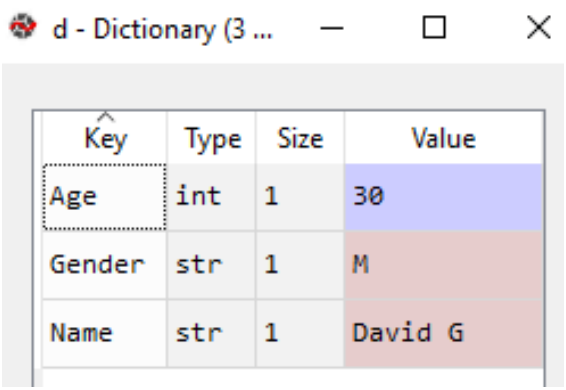


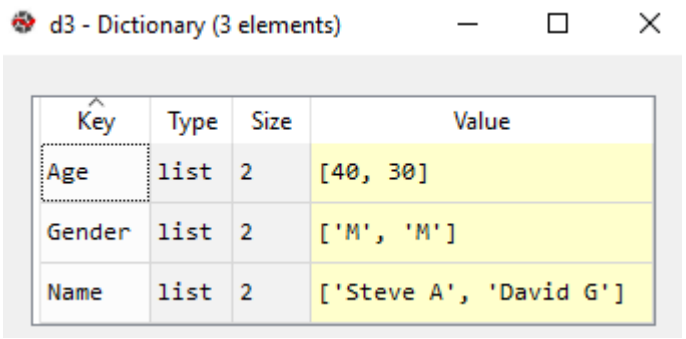
Index	Type	Size	Value
0	tuple	3	(1, 2, 3)
1	tuple	5	('a', 'b', 'c', 'd', 'e')
2	tuple	2	(True, False)
3	str	1	Hello
4	tuple	2	(0.5, 1)

- Handle List contents
  - Indexing using square bracket []: *my_list[2][1]* → False
  - Add contents to a list: *my_list.append ((0.5,1))]* → add a new element with data type tuple to my_list
  - Delete an element from a list*: del my_list[-1]* → delete the last element

Python Data Types – Dictionary

- A dictionary is declared by using brackets {} that contains a number of items of ‘keys’ and associated ‘values’. Each key is separated from its value by a colon (:), the items are separated by commas
- Example
  - d = {'Name' : 'David G', 'Gender' : 'M', 'Age' : 30} # define a dictionary
  - d3 = {'Name' : ['Steve A', 'David G'], 'Gender' : ['M','M'], 'Age' : [40, 30]} # dict with list





- Handle Dictionary contents
  - d['Age’] # referencing the values of the ‘Age’ key
  - d['Age'] = 40 # change element
  - Add a new key to a dictionary: d['School'] = 'MIT'; # Add new entry
  - Delete an key from a dictionary: del d['Name']; # remove entry with key 'Name'

Copy by reference or value in Python

- Python treats all basic data types e.g. single character, float, number, similarly as other programming languages. These data types are copied by value (on assignment using the "=").
  - a = 1
  - b = a
  - a = 2
  - print a, b => output 2 1
- However, for data types such as array, list, dataframe etc., the data is copied by reference (on assignment using the "="). This means the original variable will be modified if the new copied variable is modified
  - a = [1, 2, 3, 4]
  - b = a
  - a[0] = 0
  - print b # output [0, 2, 3, 4]

Copy by reference or value in Python

- The way to avoid the original copy of data be modified?
  - b=a.copy()
- Not sure about what data type you have? Try *type(s1)* where s1 is the name of the variable
- Here is a good place to explain the difference between '==' and 'is', by trying the following code:
  - a=[1 2 3]
  - b=a
  - c=a.copy()
  - 'b==a' gives True
  - 'b is a' gives True
  - 'c==a' gives True
  - 'c is a' gives False

Python Data Structure – Dataframe

- **DataFrame** is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object.
- DataFrame accepts many different kinds of input:
  - Dict of 1D ndarrays, lists, dicts, or Series
  - 2-D numpy.ndarray
  - Structured or record ndarray
  - A Series
  - Another DataFrame
- Along with the data, you can optionally pass index (row labels) and columns (column labels) arguments.

Python Data Structure – Dataframe

- Example: creating a dataframe

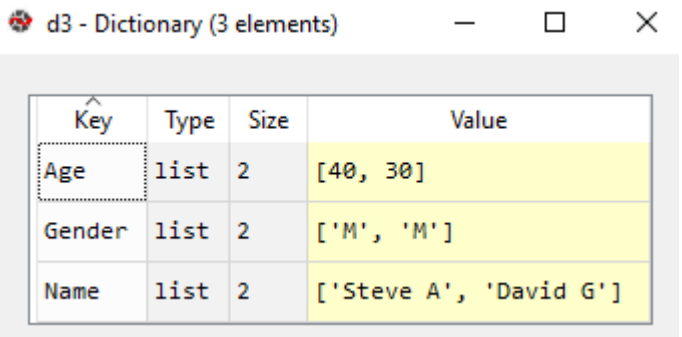
```
np.random.seed(1234)
data = np.random.randn(5, 2)  # 5x2 matrix of N(0, 1) random draws
dates = pd.date_range('28/12/2010', periods=5) # five random dates
# define the dataframe
df = pd.DataFrame(data, columns=('price', 'weight'), index=dates)
```

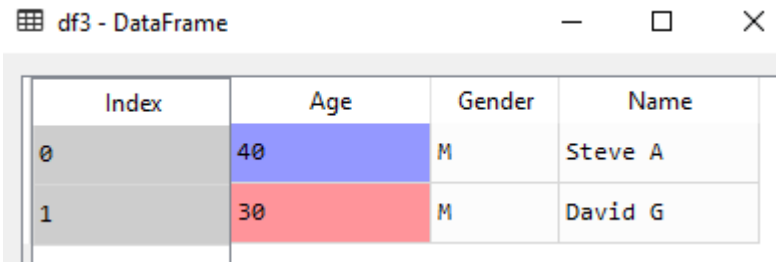
- Example: manipulating a dataframe
  - *df1=pd.DataFrame(df, columns=['price']) # creating a new dataframe, taking only the price column from df*
  - *df2 = df.loc['2010-12-30':] # creating a new dataframe, taking only a portion of the rows*
  - *df3 = df.loc[df.index[0:2]] # taking only the first two rows of df, but use index*
  - *df5 = df.iloc[0:2] # taking only the first two rows of df, use integer row number. iloc's i means using integer number for row/column*
  - *df6 = df.iloc[:,0:1] # take only the first column of df, using integer column numbers*
  - *df4 = df.copy() # creates a copy*
  - *df4['weightedPrice'] = df['price'] * df['weight'] #creates a new column*
  - *df4['Hiflag'] = df['weight'] > 0.1 # add a boolean flag column*
  - *del df4['Hiflag'] # delete the added flag column*
  - *df4.drop(df4.index[0:2], axis=0) # delete the first two rows*

Converting a Dictionary to a Dataframe

- A dictionary can be converted into a dataframe using the pandas method DataFrame
- Example

  import pandas as pd
  df3 = pd.DataFrame(d3)





Indexing Pandas Data (Dataframe) using 'iloc'

- "iloc" in pandas is used to select rows and columns of a dataframe by number
- Example

  df5 = df.iloc[0:2] # taking only the first two rows of df, use integer row number.
  df6 = df.iloc[:,0:1] # take only the first column of df, using integer column numbers
  df7 = df.iloc[0:2,0:2] # take only the first two row and first two column of df
- When only one row is selected, Python outputs a Series datatype instead of a Dataframe. To force output to be a Dataframe, use double [[ ]] for indexing:

  df5a = df.iloc[[2]]

Indexing Pandas Data (Dataframe) using 'loc'

- The Pandas "loc" indexer can be used with DataFrames for two different use cases:
  a) Selecting rows by label/index
  b) Selecting rows with a boolean / conditional lookup

- Example

df2 = df.loc['2010-12-30':] # creating a new dataframe, taking only a portion of the rows starting from a certain date
df3 = df.loc[df.index[0:2]] # taking only the first two rows of df, but use index
df4b = df4.loc['2010-12-30':,'weight':'Hiflag']
df4c = df4.loc['2010-12-30':,['weight','Hiflag']]

- Boolean / Logical indexing using .loc

 # slicing - boolean indexing using .loc
df4f = df4.loc[df.price>0,:]
df4d = df4.loc[df.price>0,'price']

Multi-Indexing

- Hierarchical / Multi-level indexing enables sophisticated data analysis and manipulation, especially for working with higher dimensional data.
- It enables you to store and manipulate data with an arbitrary number of dimensions in lower dimensional data structures like Series (1d) and DataFrame (2d)
- The MultiIndex object is the hierarchical analogue of the standard Index object which typically stores the axis labels in pandas objects

		return1	return2
date	ticker		
3/1/1994	JPM	-0.341555	-1.273598
	GooG	1.601131	1.117483
	GS	1.484033	1.570661
9/1/1994	JPM	0.508370	0.036174
	GooG	-0.024200	-0.921021
	GS	0.339376	-0.681391

More on Indexing and selecting Pandas Data

- https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html
- Multi-indexing
  https://pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html#advanced
- Group by

Plotting methods associated with Series/Dataframes

- Example: simple line plot of all data in a dataframe/Series

```
Ret_Dow.plot(kind='hist', bins=bins, normed=True,
             alpha=0.5, color='blue')
```

- Example: areaplot of data in a dataframe

```
pw = pd.DataFrame(portfolios)
pw.columns = df.columns.values
pw.index = returns
pw.plot.area()
```

More on Python Data Structures

https://pandas.pydata.org/pandas-docs/stable/dsintro.html

Python Graphics – line plots and bar charts

```
# line plots
figure_count = 1
plt.figure(figure_count)

plt.plot(ret1)
plt.ylabel('R(t)')

# bar chart
figure_count = figure_count+1
plt.figure(figure_count)
plt.hist( ret1, normed=True, bins=50,
histtype='stepfilled', alpha=0.5, label='ret')
plt.legend(loc='upper left',  bbox_to_anchor=(0.05,
0.9), shadow=True, ncol=1)
xmin, xmax = -0.1, 0.1
plt.xlim( (xmin, xmax) )
plt.xlabel('returns')
plt.ylabel('frequency')
```

Useful Python Charting Resources

https://www.datacamp.com/community/tutorials/matplotlib-tutorial-python

https://python-graph-gallery.com/

https://matplotlib.org/users/pyplot_tutorial.html

https://plot.ly/python/

Downloading data from various sources

- Sample code (L3_DownloadData.py) to download data from the following sources:
  - Yahoo finance for individual stock/ETF/Mutual fund price data
  - Ken French website for Fama-French historical factor return data
  - St. Louis Fed for Macro data

- Get yourself familiar with the code and learn how to change the code to get information on
  - Other stocks/ETF/Indices
  - Other factor returns
  - Other Macro data

Getting Macro data from Fred

- Get data for the given name from the St. Louis FED (FRED)

  pandas_datareader.data .DataReader(symbols, start=None, end=None,)

where

symbols – symbols of dataseries you want to download from the FRED website. E.g., 'CPIAUCSL' for US CPI

start – start date for data

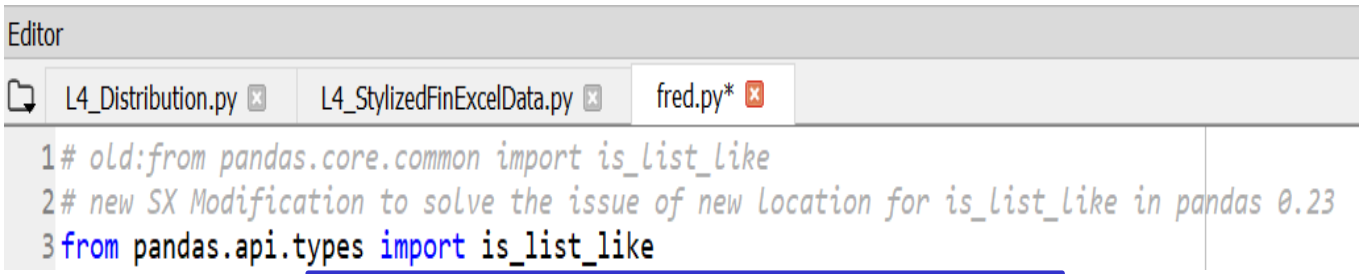
end – end date for data

- This function returns a DataFrame
- See datareader documentation pdf file for more details

Bug correction for code getting data from Fred

- Because of the fred package was changed in Pandas 0.23 which came from the latest anaconda installation, an error will occur when using Datareader to get data from Fred
- How to fix it?
  - You can open up fred.py by clicking the purple <module> link following the error message.
  - Or you can also open the file by going to where it is. For me, Fred.py is located at C:\ProgramData\Anaconda3\Lib\site-packages\pandas_datareader

Because the `is_list_like` is moved to pandas.api.types, I change the fred.py file which is highlighted in the picture. I replace `from pandas.core.common import is_list_like` with `from pandas.api.types import is_list_like`, and it works.



Getting Security level data from Yahoo Finance

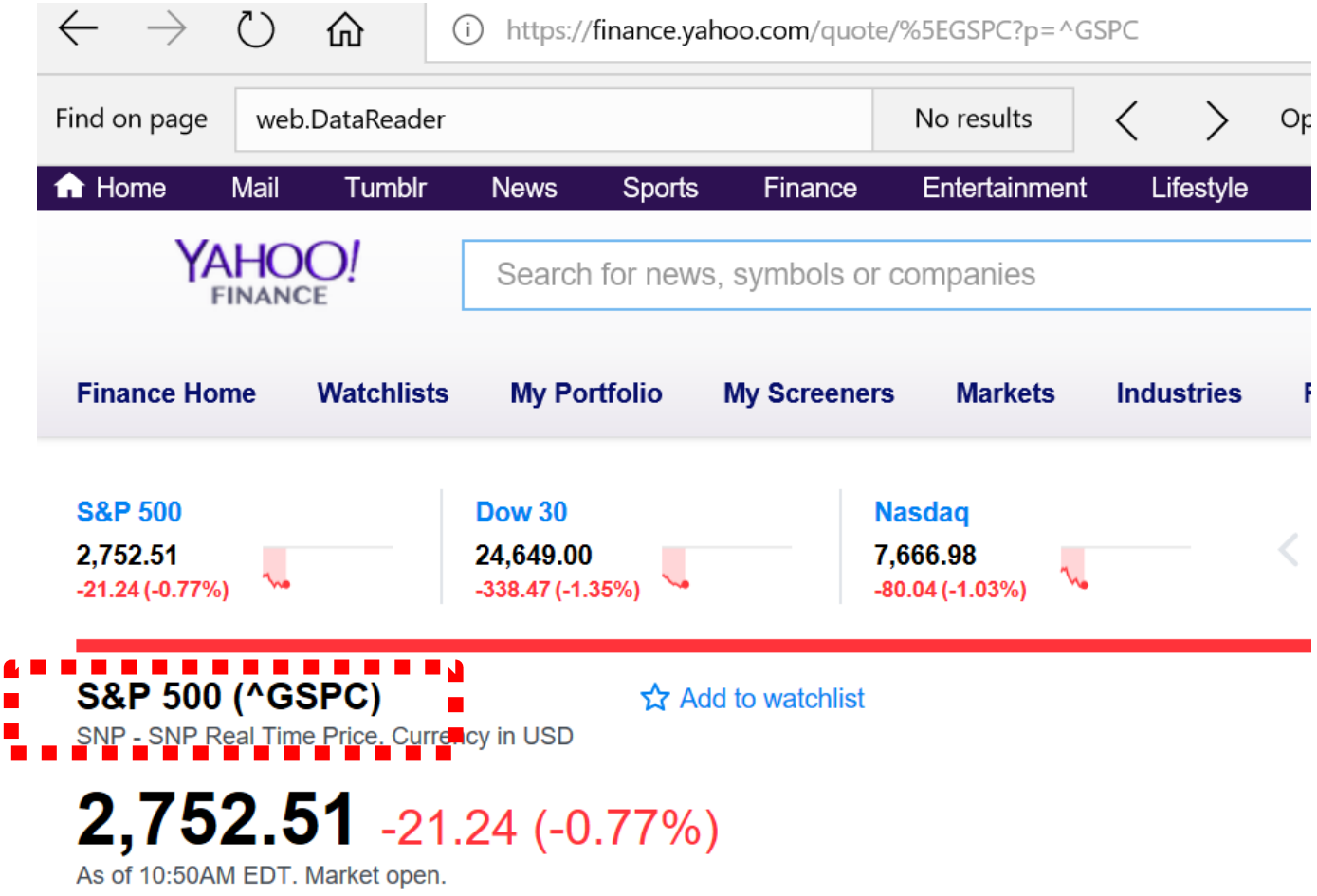
- Get data for the given name from the yahoo finance website

```
data = pdr.get_data_yahoo(
        tickers = ["SPY", "IWM", "..."], # tickers list (single tickers accepts a string as well)
        start = "2017-01-01", # start date (YYYY-MM-DD / datetime.datetime object)
                # (optional, defaults is 1950-01-01
        end = "2017-04-30", # end date (YYYY-MM-DD / datetime.datetime object)
                # (optional, defaults is Today)
        as_panel = False,  # return a multi-index dataframe
                # (optional, default is Panel, which is deprecated)
        group_by = 'ticker', # group by ticker (to access via data['SPY'])
                # (optional, default is 'column')
        auto_adjust = True, # adjust all OHLC automatically
                # (optional, default is False)
        actions = True,  # download dividend + stock splits data
                # (optional, default is None)
                # options are:
                #   - True (returns history + actions)
                #   - 'only' (actions only)
    threads = 10  # How may threads to use?
```

- Note: fix-yahoo-finance was renamed to yfinance
- More details at https://pypi.org/project/fix-yahoo-finance/

How to find the symbol from Yahoo Finance

- Go to yahoo.finance.com, from the search bar, type in name you want, such as s&p500, click then find the symbol from the page that came up



How to find the symbol from Fred

- Go to https://fred.stlouisfed.org/, from the search bar, type in name you want, such as us cpi, click the item you want then find the symbol from the page that came up

How to study the codes given to you – an example

- Let's go over an example **L3_FrameDataManipulation.py**
- This code does the following tasks (Important to understand this first before dig into any code!)
  - Read in data from two existing Excel files:
    - The two files contain time series data that we will actually use in later parts of the course.
      - One contains historical flags indicating each month in history whether it is a RiskOn (good for equity) or RiskOff (bad for equity) month
      - The other contains historical factor returns for US Equity
    - The dates formats are different, one is YYYYMM, one is YYYY-MM-DD
  - Merge the two data sets together
    - First convert the YYYYMM dates into the YYYY-MM-DD format, assuming the missing days are at month end
  - Calculate conditional mean and covariance of the factor returns, given the regime flags
    - Separate the data set into two: one set for Risk-on time periods and one for risk-off
  - Convert the calculated results into Dataframes and export them into excel files to be used later in this course! And put some formatting into the excel file!

Example: How to study the codes given to you

1. Understand what does 'read_excel' function do
   - Highlight and press control i
   - 'header=3, index_col = 0' defines where does the reading begins from inside the excel file
   - Double click on variable 'df_Factor' from the variable explorer window and check out the content
2. Understand what does 'FactorDateWDay = FactorDate.apply(addMthEndDaytoYearMonth, axis=1)' do
   - Google 'python .apply' -> Apply a function along an axis of the DataFrame
   - Axis=1 applies the function to the rows
3. Understand what does the code: 'calendar.monthrange(year, month)' do
   - Highlight the portion of the code and hit 'control i', it will brings up simple help information on the function under the help window
   - Set a breakpoint in the function where this part of code exist and debug through it.
   - During the debugging, where in the function addMthEndDaytoYearMonth , type in calendar.monthrange(year, month) in the console and see what is the output
   - In google, type in 'python calendar.monthrange'

Output: Conditional Analytics

- Conditional expected returns for Risk-on and Risk-off regimes:

Mean Monthly Regime Returns					
	Mkt-RF	SMB	HML	RF	MoM
RiskOn	1.46	0.18	0.18	0.23	0.34
RiskOff	-2.19	0.05	0.25	0.22	1.28

- Conditional covariance matrix for Risk-on and Risk-off regimes:

RiskOn Monthly Covariance Matrix					
	Mkt-RF	SMB	HML	RF	MoM
Mkt-RF	12.15	2.08	-2.02	-0.02	-2.44
SMB	2.08	10.82	-2.81	-0.03	2.16
HML	-2.02	-2.81	7.22	-0.05	-4.82
RF	-0.02	-0.03	-0.05	0.04	0.12
MoM	-2.44	2.16	-4.82	0.12	20.44

RiskOff Monthly Covariance Matrix					
	Mkt-RF	SMB	HML	RF	MoM
Mkt-RF	27.53	5.55	-2.39	-0.10	-12.31
SMB	5.55	8.79	-1.76	-0.10	-4.29
HML	-2.39	-1.76	15.52	0.26	4.93
RF	-0.10	-0.10	0.26	0.04	0.01
MoM	-12.31	-4.29	4.93	0.01	30.71

Example: How to study the codes given to you

4. Understand what does df_merged = pd.concat(frames) and df_merged_C = pd.concat(frames, axis=1, join='inner') do
   - An inner join only take rows of the two dataframes where the index (dates in this case) are identical
   - An outer join creates a resulting dataframe where both rows with shared dates and rows unique to each individual dataframe are kept
5. Understand what does 'mean_ret_RiskOn = df_merged_C_RiskOn.mean()' do
   - Calling the method mean associated with a dataframe
   - Google 'DataFrame.mean'
6. Try out the following methods associated with dataframes:
   - .shift: a=df_merged_C_RiskOn['SMB'].shift(-1)

7. Try yourself…
   - What does 'np.vstack' do?
   - What does 'df_mean = pd.DataFrame(mean_data, columns=Name_Factors,' do?

Basic Probability Theory

- **Basic Probability Theory and related coding in Python**

Random Variables

- Random Variables
  - A random variable maps the outcome of a random phenomena to a unique numerical value
- Value and Probability of a random variable

$$x_i, p_i$$

- Example: Coin toss
  - x_i = (Head, Tail)
  - p_i = (0.5, 0.5)
- Example: Dice
  - x_i = (1, 2, 3, 4, 5, 6)
  - p_i = (1/6, 1/6, 1/6, 1/6, 1/6, 1/6)

Continuous Random Variables

- Discrete vs. Continuous
  - Discrete: Coin, Dice etc.
  - Continuous: Uniform, Normal, Lognormal etc.
- Continuous examples
  - Uniform
  - Normal
  - Log Normal
  - Student-t

Describing Random Variables

- Probability Distribution
  - Pdf (histogram) and
  - Cumulative distribution function

- Statistics
  - mean
  - Variance
  - Skewness
  - Kurtosis
  - Etc.

Stats – Expected Value

- Expected Value

$$E(X) = \sum_{i=1}^{N} p_i x_i$$

- For the case of throwing a dice:
  - E(X) = 1/6*1 + 1/6*2 + 1/6*3 + 1/6*4 + 1/6*5 +1/6*6 = 3.5
  -

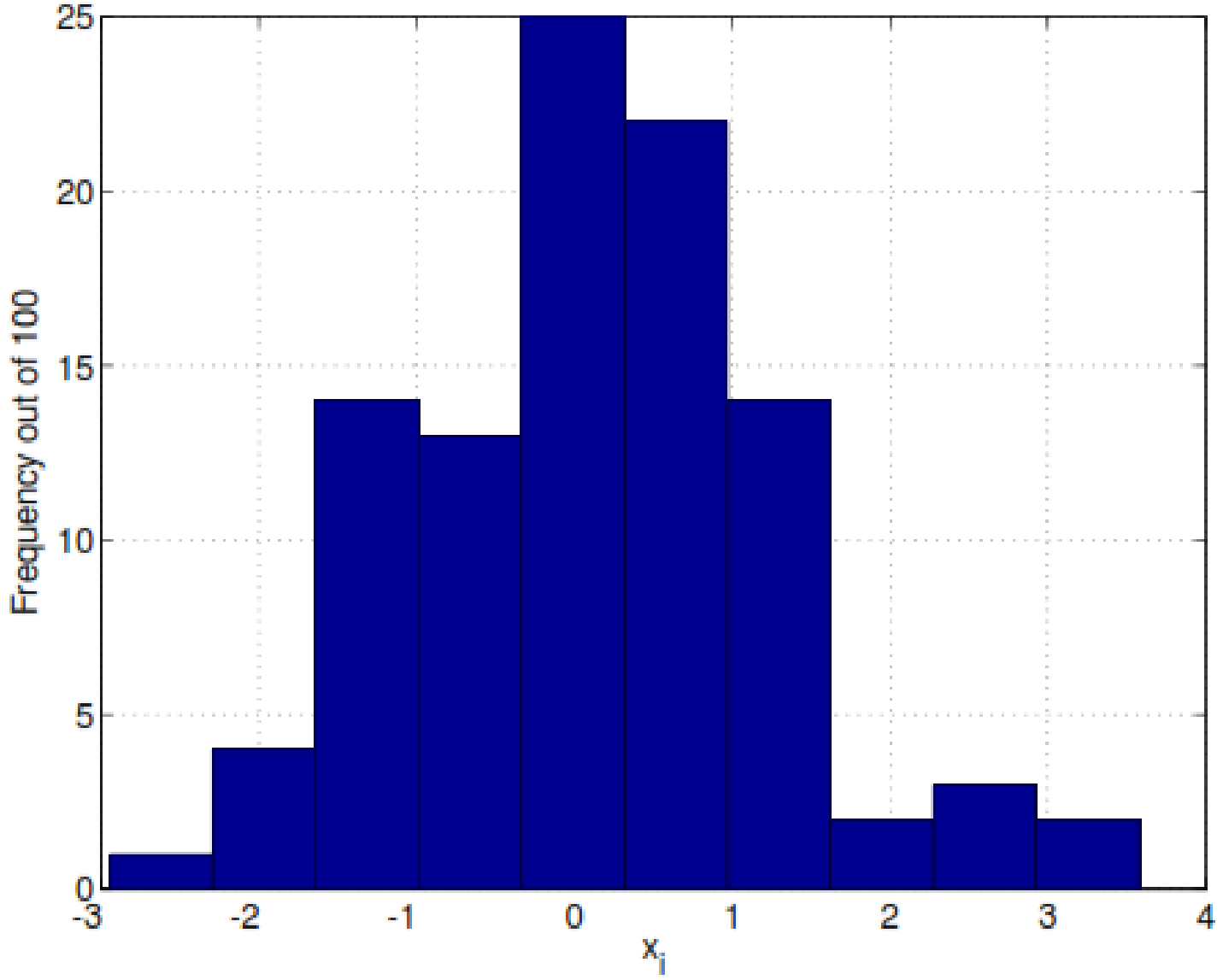
Stats – Sample Mean

- Sample mean, with equal probability

$$\hat{m} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

- Median:
  - Pr(X<median) = 0.5
  - $fraction(x_i$<median) = 0.5

Histogram



Probability Density Function

- F( ) = Cumulative density function (cdf)
- f ( ) = probability density function (pdf)

$$F(a) = Pr(X \leq a) = \int_{-\infty}^{+\infty} f(x)dx$$

$$f(x) = \frac{dF(x)}{dx}$$

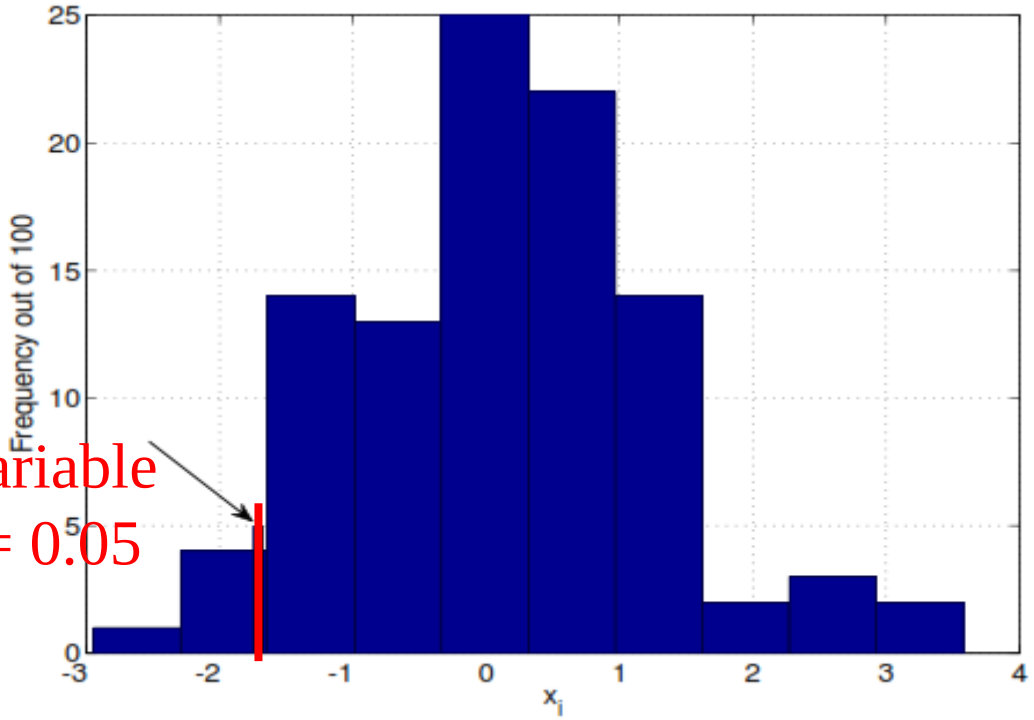
Stats – Quantile

- Quantile α

$$q_\alpha: \Pr(X<q_\alpha)= \alpha$$

e.g., $q_{0.5}$=median

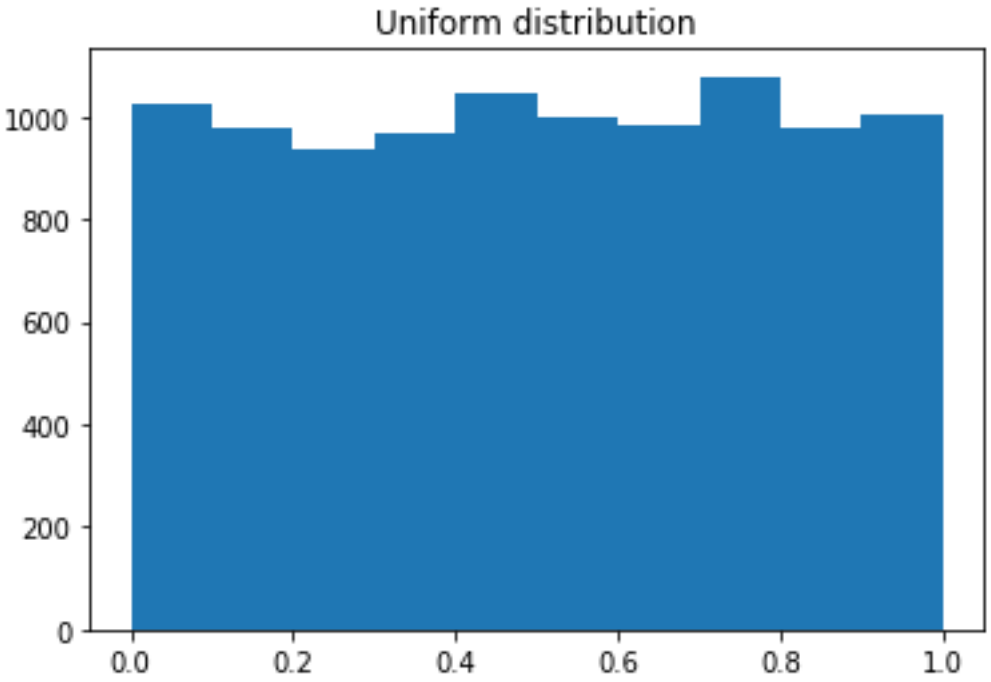
- Quantile 0.05, $q_{0.05}$:



$q_{0.05}$ is the x variable that make cdf = 0.05

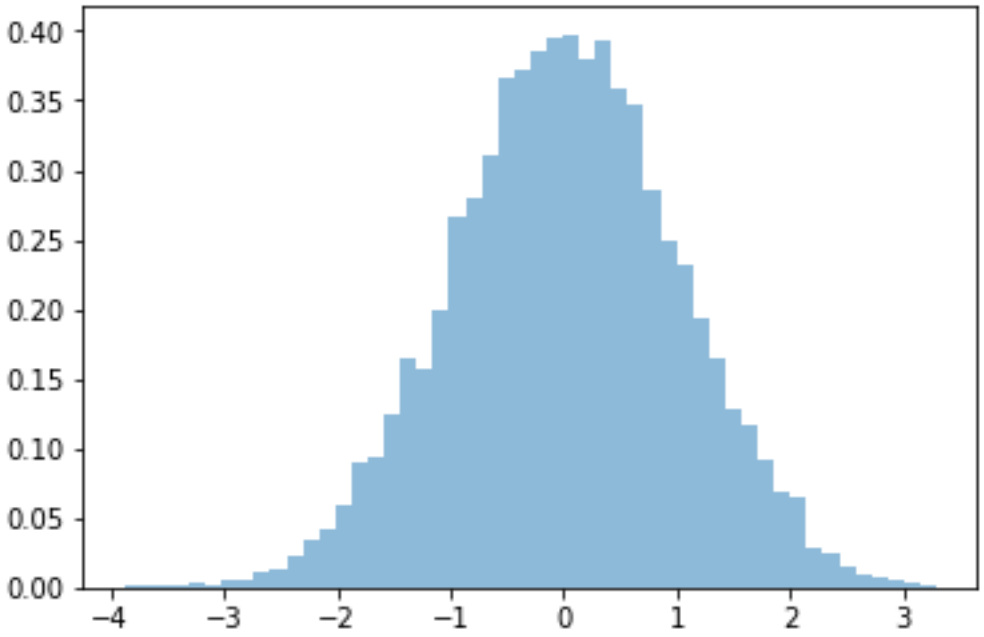
Uniform Distribution

- np.random.rand(m,n) in python
  - Produce a sample of m×n matrix of random numbers between [0,1]
- matplotlib.pyplot.hist(np.random.rand(10000,1)) gives



Normal Distribution

- np.random.normal(0.0, 1.0, (m,n)) in python
  - Produce a sample of m×n matrix of normally distributed random numbers with mean=0, and std=1
- matplotlib.pyplot.hist((np.random.randn(10000, 1)) plots the normal distribution



Normal Density Function

- Normal distribution is fully defined by two parameters

$mean$

Standard deviation

- Normal pdf, given mean μ and σ:

$$pdf = f(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp[-\frac{1}{2}(\frac{x-\mu}{\sigma})^2]$$

$$cdf = F(x,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}}\int_{-\infty}^{x}\exp\left[-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2\right]dt$$

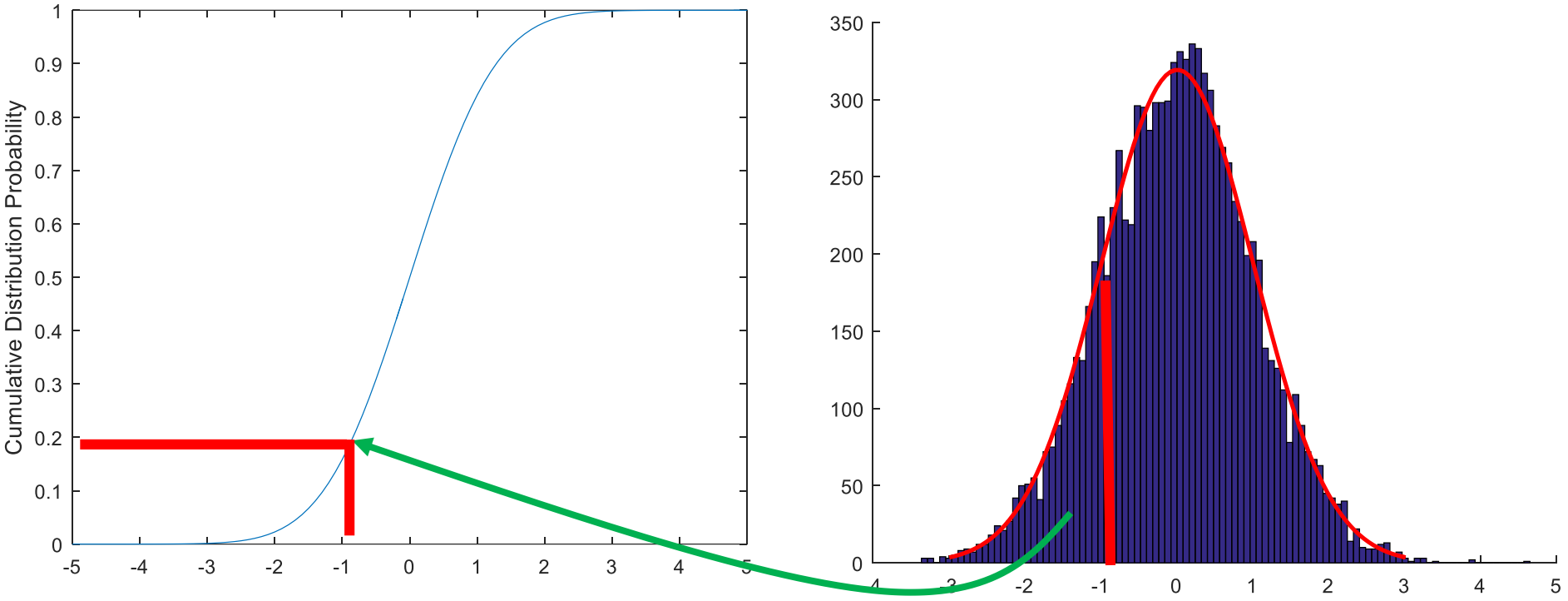
- For $N(\mu = 0, \sigma^2 = 1)$

$$\emptyset(x) = \frac{1}{\sqrt{2\pi}}\exp\left[-\frac{1}{2}x^2\right]$$

Example of CDF

Python code:

```
from scipy.stats import norm
Randomdata = np.random.randn(10000, 1)  # 10000x1 matrix of N(0, 1) random draws
Randomdata_sorted = sorted(Randomdata) # sort the data
cdf_Rand = norm.cdf(Randomdata_sorted, loc=0, scale=1) # create cdf
```



Appendix

Stats – Variance and Standard Deviation

- Variance:

$$m = E(X) = \sum_{i=1}^{N} p_i x_i$$

$$var(X) = \sum_{i=1}^{N} p_i (x_i - m)^2$$

$$std(X) = \sqrt{var(X)}$$

- Sample Variance

$$\hat{m} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

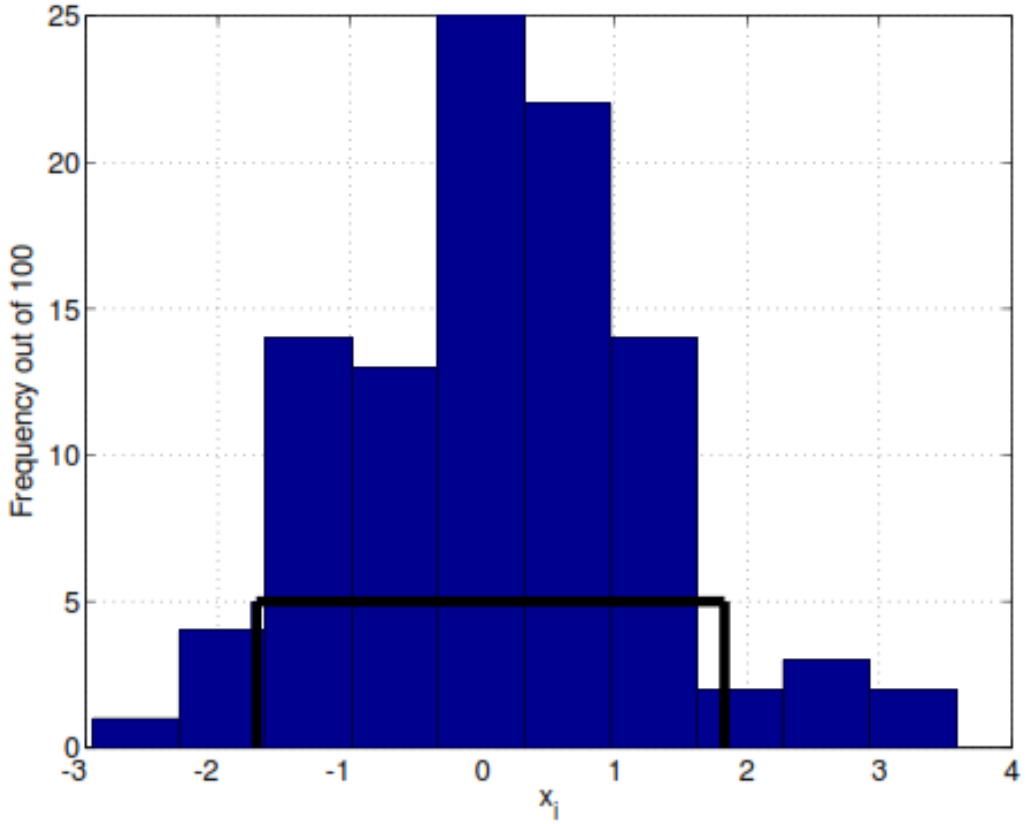
$$var(X) = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \hat{m})^2$$

Stats – Quantile Ranges

- Quantile Ranges:

$$q_{0.95} - q_{0.05}$$

$$q_{0.75} - q_{0.25} = \text{Interquartile range}$$



Centered Moments

First moment: $\mu_X = E(X) = \int_{-\infty}^{\infty} xf(x)dx$

Mth moment: $E(X - \mu_X)^m = \int_{-\infty}^{\infty} (x - \mu_X)^m f(x)dx$

Skewness = $\frac{E(X-\mu_X)^3}{\sigma^3} = \frac{1}{\sigma^3}\int_{-\infty}^{\infty} (x - \mu_X)^3 f(x)dx$

Kurtosis = $\frac{E(X-\mu_X)^4}{\sigma^4} = \frac{1}{\sigma^4}\int_{-\infty}^{\infty} (x - \mu_X)^4 f(x)dx$

Sample Estimates

$$\text{Mean: } \hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\text{Variance: } \hat{\sigma}^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \hat{\mu})^2$$

$$\text{Skewness: } = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \hat{\mu})^3 \,/\hat{\sigma}^3$$

$$\text{Kurtosis: } = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \hat{\mu})^4 \,/\hat{\sigma}^4$$

For Normal distributions:

$$\text{Mean: } \hat{\mu}$$

$$\text{Variance: } \hat{\sigma}^2$$

$$\text{Skewness: } = 0$$

$$\text{Kurtosis: } = 3$$

Central Limit Theorem

- For any random variable X, if the variance of X exists, then:

$$Y = \sum_{i=1}^{N} x_i$$

follows a normal distribution if N is large

- This means that the mean of the X also follows a normal distribution

Log Normal Distribution

- If Y=Log(X) is normally distributed then $X = e^Y$ is log-normally distributed:
- Example: Asymmetric distribution for stock price, $\mu = 0.0, \sigma = 0.5$