UNIVERSITY OF WESTMINSTER

7FNCE041W – Computational Methods for Finance

Basic Concepts of Python

FINTECH WITH BUSINESS ANALYTICS MSC 2023-2024, SEMESTER 1

Versha Sandesh, Student ID: 20395827

INDIVIDUAL AUTHENTIC ASSESSMENT

Submission Date: 10 January 2024

Word Count: 2344 (excluding cover page, table of contents, formulas, quantitative answers, and references)

Contents

1. A Python Library 2

1.1. How to use Python Library 2

Executing command without 'Library' 2

1.2. Features of Python essential Libraries 3

a. NumPy 3

b. Pandas 4

c. Matplotlib 6

d. yfinance 7

1.3. Comparison between the four libraries 10

2. Volatility 10

2.1. Implied Volatility and Historical Volatility 10

a. Implied Volatility (IV) 10

b. Historical Volatility (HV) 11

c. Comparison between Implied and Historical Volatility 11

2.2. Newton-Raphson Iteration 12

2.3. Newton-Raphson Iteration using Python 13

2.4. Implied Volatility of Amazon's Option 14

2.5. Annual Historical Volatility of Amazon's Stock Return 16

Comparison of Amazon's Implied and Historical Volatility 16

3. Binomial Tree Option Pricing 17

3.1. Calculation: u, d, and the risk-neutral probability p for a four-step tree 17

a. Up factor 17

b. Down factor 17

c. Risk-neutral Probability 18

3.2. Four-step Binomial Tree for European Call Option 18

3.3. Binomial Tree in Python 20

3.4. Comparison of Results 21

References 23

1. A Python Library

In Python, a library refers to a bundle of code, encompassing a diverse set of built-in modules written in C. The standard library includes modules for system functionalities like file I/O, ranging from integral components of the language to specialized modules (Python, n.d.). It includes everything from modules that are as much part of the Python language as the types and statements defined by the language specification, to obscure modules that are probably useful only to a small number of programs (Lundh, 2001).

The standard library, containing hundreds of modules for performing common tasks, comes bundled with installation of Python (Cepalia, n.d.). Importing these modules into programs enhances efficiency by leveraging well-tested and optimized code. Python libraries span various domains, including data manipulation (NumPy, pandas, Matplotlib), web development (Flask, Django), machine learning (TensorFlow, PyTorch), and data visualization (Seaborn, Plotly), showcasing the language's flexibility and extensibility.

1.1. How to use Python Library

If a library is not part of the Python standard library or have not been installed previously, then first install the required library before using them with '**pip**' (Python's package installer) function.

```
In [ ]: pip install library_name #Use library name, e.g. yfinance
```

If Python software is installed on the computer via 'Anaconda' distribution, essential data analysis packages are pre-installed within Python such as NumPy or any of the major packages, like pandas, Scikit-Learn, etc.

To use a library in Python, import it into the script using the '**import**' keyword.

```
In [ ]: import library_name #call a library e.g. NumPy, Pandas, yfinance
```

Executing command without 'Library'

If library is not installed, the following error appears:

```
ModuleNotFoundError: No module named 'library_name'
```

If a function or module from a library is used without importing it, Python will raise an error as below:

```
NameError: name 'library_name' is not defined
```

1.2. Features of Python essential Libraries

a. NumPy

NumPy (Numerical Python) is an open-source Python library that to work with numerical data in Python. The NumPy API is used extensively in Pandas, SciPy, Matplotlib, scikit-learn, scikit-image and most other data science and scientific Python packages.

The NumPy library features multidimensional array and matrix data structures, primarily the homogeneous n-dimensional array object called 'ndarray'. NumPy empowers efficient operations on arrays, supporting a broad range of high-level mathematical functions. Crucial for scientific computing and data analysis, NumPy facilitates tasks such as mathematical operations, logical computations, shape manipulation, sorting, selection, I/O, discrete Fourier transforms, basic linear algebra, statistical operations, random simulation, and more. (NumPy, n.d.).

To **install** NumPy in Python, use command:

```
conda install numpy
```

Or

```
pip install numpy
```

To access NumPy and its functions, **import** it using:

```
import numpy as np
```

The '**np**' is a common convention for NumPy, recommended for improved code readability.

Creating a NumPy array:

NumPy gives a range of ways to create arrays and manipulate data inside them, distinguishing itself from Python lists by providing a faster array object. While a Python list can contain different data types within a single list, all the elements in a NumPy array should be homogeneous.

To create a NumPy array, use the function **np.array()**.

```
import numpy as np
a = np.array([1, 2, 3]) #to create an array of three elements
print (a)
```

```
[1 2 3]
```

Note that in NumPy, indexing starts at 0. That means to access the first element in the array, access the element "0".

```
import numpy as np
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print (a[0]) #prints first element in the array
```

```
[1 2 3 4]
```

Example: NumPy

```
In [1]: import numpy as np

        # Example: Computing the mean and standard deviation of an array
        data = np.array([1, 2, 3, 4, 5])
        mean_value = np.mean(data)
        std_dev = np.std(data)

        print("Mean:", mean_value)
        print("Standard Deviation:", std_dev)
```

```
Mean: 3.0
Standard Deviation: 1.4142135623730951
```

b. Pandas

Pandas is a widely-used library for efficient data manipulation and analysis, offering essential structures like DataFrame and Series for seamless handling of tabular data. With Pandas, you can easily load, clean, filter, and transform financial datasets (Chin, 2023).

Pandas facilitates big data analysis, statistical inference, and the manipulation of messy datasets, providing tools for various data formats (CSV, Excel, SQL, etc.) and offers powerful data manipulation functions like filtering, grouping, merging, and reshaping. For example, easy handling of missing data (represented as NaN) in floating and non-floating-point data; insertion and deletion of from DataFrame and higher dimensional objects; powerful group by functionality for performing split-apply-combine operations on data sets.

Pandas are used in conjunction with other libraries that are used for data science. It is built on the top of the NumPy library which means that a lot of structures of NumPy are used or replicated in Pandas. The data produced by Pandas are often used as input for plotting functions of Matplotlib, statistical analysis in SciPy, and machine learning algorithms in Scikit-learn (GeeksforGeeks, 2024).

To **install** pandas in Python, use command:

```
pip install pandas
```

Pandas are usually **imported** in conjunction with NumPy, as follows:

```
import numpy as np
import pandas as pd
```

Where '**pd**' is conveniently used as short for pandas.

Pandas provides two types of classes for handling data: (pandas, n.d.)

1. **Series:** a one-dimensional labelled array holding data of any type, such as, integers, strings, Python objects, etc.

```
In [ ]: import pandas as pd

        # Calling Series constructor
        s = pd.Series()
        print(s)
```

2. **DataFrame:** a two-dimensional data structure that holds data like a two-dimensional array or a table with rows and columns.

```
In [ ]: import pandas as pd

        # Calling DataFrame constructor
        df = pd.DataFrame()
        print(df)
```

Following are some of the many extensive functionality operations of pandas in Python:

1. Creating DataFrames:

```
# Creating a DataFrame from a dictionary
df = pd.DataFrame({'column1': [1, 2, 3], 'column2': ['a', 'b', 'c']})
```

2. Reading Data:

```
In [ ]: # Reading from a CSV file
        df = pd.read_csv('filename.csv')
```

3. Viewing Data:

```
In [ ]: # Displaying the first few rows of the DataFrame
        print(df.head())

        # Displaying basic information about the DataFrame
        print(df.info())

        # Descriptive statistics of the DataFrame
        print(df.describe())
```

Example: pandas

```
In [2]: import pandas as pd

        # Example: Creating a DataFrame and performing basic operations
        data = {'Name': ['Alice', 'Bob', 'Charlie'],
                'Age': [25, 30, 35],
                'Salary': [50000, 60000, 75000]}

        df = pd.DataFrame(data)

        # Display the DataFrame
        print("DataFrame:")
        print(df)

        # Perform basic operations
        average_salary = df['Salary'].mean()
        print("\nAverage Salary:", average_salary)
```

```
DataFrame:
      Name  Age  Salary
0    Alice   25   50000
1      Bob   30   60000
2  Charlie   35   75000

Average Salary: 61666.666666666664
```

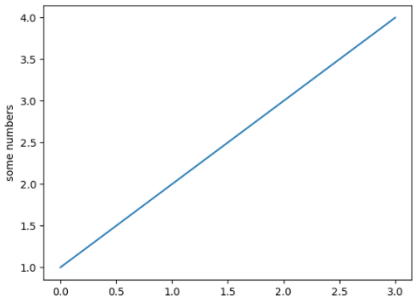
c. Matplotlib

Matplotlib is a powerful 2D plotting library, facilitating the creation of diverse visualizations in Python, like to plot stock prices, performance metrics, or trend analysis (Chin, 2023). It supports various plot types, including line plots, scatter plots, bar plots, histograms, etc.

matplotlib.pyplot is a collection of functions, where each pyplot function is highly customizable, allowing users to control almost every aspect of the plot: e.g., for creating figures, plotting areas, lines, and adding labels, etc. (matplotlib, n.d.)

To generate visualizations with matplotlib, **import** the library after installing (if needed):

```
In [ ]: import matplotlib.pyplot as plt

        plt.plot([1, 2, 3, 4])
        plt.ylabel('some numbers')
        plt.show()
```

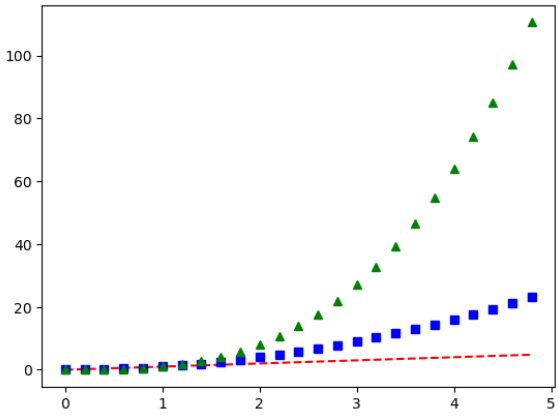


Where '**plt**' is converted to short for matplotlib.pyplot.

Formatting the style of plot

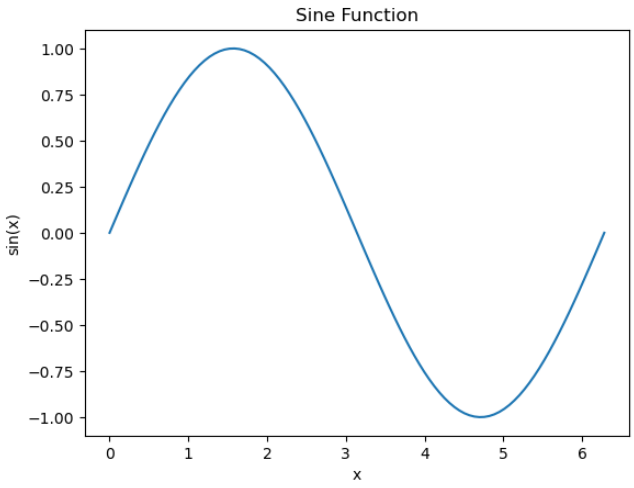
The example below illustrates one of the essential features of matplotlib, by plotting several lines with different format styles in one function call using arrays.

```
In [6]: import numpy as np

        # evenly sampled time at 200ms intervals
        t = np.arange(0., 5., 0.2)

        # red dashes, blue squares and green triangles
        plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
        plt.show()
```



Example: matplotlib

```
In [7]: import matplotlib.pyplot as plt

        # Example: Plotting a simple line chart
        x = np.linspace(0, 2 * np.pi, 100)
        y = np.sin(x)

        # Plotting
        plt.plot(x, y)
        plt.title('Sine Function')
        plt.xlabel('x')
        plt.ylabel('sin(x)')
        plt.show()
```



d. yfinance

yfinance is a Python library simplifying the download financial data from Yahoo Finance. It provides aan easy way to access a wide range of financial data for a given stock symbol, including historical prices, financial statements, and other information (Meier, 2023).

To use yfinance, first **install** it by using 'pip' function then **import** it, defining as 'yf' for better code readability, as below:

```
In [ ]: pip install yfinance==0.2.28
```

```
In [ ]: import yfinance as yf

        data = yf.download("AAPL", start="2020-01-01", end="2021-01-01")
        print(data.head())
```

It can be used to obtain financial data of a single stock or multiple tickers, as shown below:

1. Accessing historical data of Microsoft Inc.

```
In [ ]: import yfinance as yf

        msft = yf.Ticker("MSFT") # to access the ticker data, e.g. Microsoft Inc.

        # get all stock info
        msft.info

        # get recent data
        recent_data = yf.download("MSFT", period="5d")

        # get historical market data
        hist = msft.history(period="1mo")

        # show actions (dividends, splits, capital gains)
        msft.actions
        msft.dividends
        msft.splits
        msft.capital_gains  # only for mutual funds & etfs
```

2. Accessing Historical data of multiple stocks.

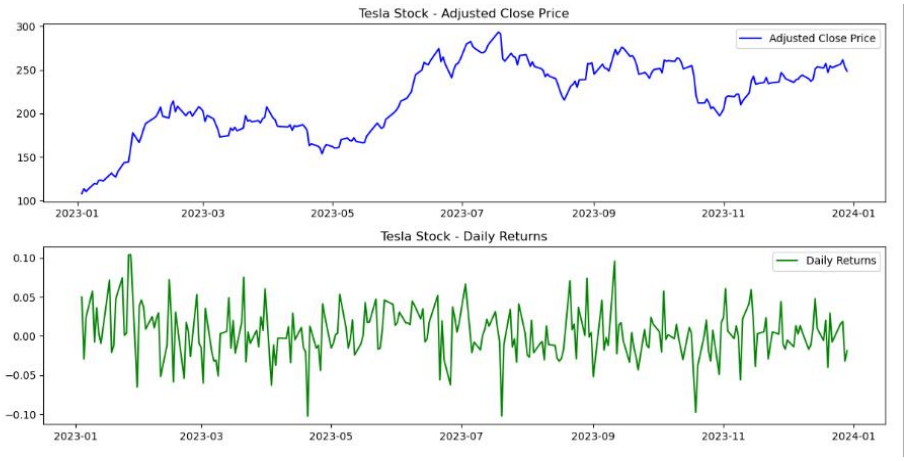
```
In [ ]: import yfinance as yf

        tickers = yf.Tickers('msft aapl goog')

        # access each ticker using (example)
        tickers.tickers['MSFT'].info
        tickers.tickers['AAPL'].history(period="1mo")
        tickers.tickers['GOOG'].actions
```

Example: yfinance

The example below integrates four Python libraries in one program. It utilizes **yfinance** to download Tesla stock data from Yahoo Finance, calculates daily returns with NumPy, creates a new DataFrame with selected columns using Pandas, and then plots the adjusted closing price and daily returns with Matplotlib.

In [12]:
```
import yfinance as yf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Fetching Tesla stock data using yfinance
ticker = 'TSLA'
start_date = '2023-01-01'
end_date = '2023-12-31'
tesla_data = yf.download(ticker, start=start_date, end=end_date)

# Displaying the first few rows of the data
print(tesla_data.head())

# Calculating daily returns using numpy
tesla_data['Daily_Return'] = np.log(tesla_data['Adj Close'] / tesla_data['Adj Close'].shift(1))

# Creating a new DataFrame with selected columns
selected_data = tesla_data[['Adj Close', 'Daily_Return']]

# Plotting adjusted closing price and daily returns using matplotlib
plt.figure(figsize=(12, 6))

# Plotting Adjusted Close Price
plt.subplot(2, 1, 1)
plt.plot(selected_data['Adj Close'], label='Adjusted Close Price', color='blue')
plt.title('Tesla Stock - Adjusted Close Price')
plt.legend()

# Plotting Daily Returns
plt.subplot(2, 1, 2)
plt.plot(selected_data['Daily_Return'], label='Daily Returns', color='green')
plt.title('Tesla Stock - Daily Returns')
plt.legend()

# Displaying the plots
plt.tight_layout()
plt.show()
```

```
[**********100%%**********]  1 of 1 completed
                 Open        High         Low       Close   Adj Close  \
Date
2023-01-03  118.470001  118.800003  104.639999  108.099998  108.099998
2023-01-04  109.110001  114.589996  107.519997  113.639999  113.639999
2023-01-05  110.510002  111.750000  107.160004  110.339996  110.339996
2023-01-06  103.000000  114.389999  101.809998  113.059998  113.059998
2023-01-09  118.959999  123.519997  117.110001  119.769997  119.769997

              Volume
Date
2023-01-03  231402800
2023-01-04  180389000
2023-01-05  157986300
2023-01-06  220911100
2023-01-09  190284000
```



1.3. Comparison between the four libraries

Commonalities:

These libraries are often used together in data science and analysis workflows. Pandas seamlessly integrates with NumPy, facilitating easy conversion between NumPy arrays and Pandas DataFrames. Additionally, Matplotlib is frequently paired with Pandas for effective data visualization. This integration enhances the overall utility of these libraries in diverse aspects of data science and analysis.

Differences:

NumPy is primarily focused on numerical computing and array manipulation. pandas are designed for data manipulation and analysis. Matplotlib specializes in plotting and visualization, while yfinance is specific to financial data retrieval.

2. Volatility

Volatility is a statistical measure of the dispersion of returns for a security or market index, indicating risk and potential reward. In options trading, both parties bet on the volatility of the underlying security. Even though there are several ways to measure volatility, options traders generally work with two metrics: implied volatility and historical volatility (Hayes, 2022).

2.1. Implied Volatility and Historical Volatility

a. Implied Volatility (IV)

Implied volatility measures the market's future volatility expectations for a financial instrument, such as a stock, index, or option. It is derived from option prices reflecting collective market opinions on potential future price movements.

Measuring Implied Volatility

Implied volatility is typically calculated using option pricing models, with the Black-Scholes model being one of the most well-known. In the context of options, it's the input value producing a theoretical option price matching the current market price.

For European options, the implied volatility of a call option equals that of a put option with the same strike price and maturity, called put-call parity (Hull, 2021).

$$p_{BS} + S_0 e^{-qT} = c_{BS} + K e^{-rT}$$

Where, p_{BS} and c_{BS} are the values of European put and call options calculated using the Black-Scholes-Merton model, both having the same strike price K, and time to maturity T. S_0 is the current price of the underlying asset, with a dividend yield q and r is the risk-free interest rate for maturity T.

b. Historical Volatility (HV)

Historical volatility assesses past price movements of an asset, calculated from its historical prices over a specific period. It provides a statistical measure of asset's past variability, indicating historical volatility.

Measuring Historical Volatility

Historical (annualized) volatility is calculated using historical price data, by taking standard deviation of the logarithmic returns of the asset's price over a specific period. The formula for historical volatility from standard deviation is derived as:

$$\text{Standard deviation (s)} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(ui - \bar{u})}$$

Where $u_i = \ln(\frac{Si}{Si-1})$, for i = 1, 2, …, n, S_i is the stock price at the end of i^{th} interval

$\bar{u}$ is the simple average of u_i

n is the number of historical price observations (trading days)

Given the distribution of log returns $\ln(\frac{ST}{S})$ as:

$$N((\mu - \frac{\sigma^2}{2})T, \sigma^2T$$

Where μ is the expected return on stock per year, and σ is the volatility of the stock return per year. Therefore, the standard deviation (s) can be written as:

$$s = \sqrt{\sigma^2 T}$$

hence, the historical volatility is:

$$\hat{\sigma} = \frac{s}{\sqrt{T}}$$

Where, T is the time interval in years between two stock prices = $\frac{n}{252}$ (usually assumed 252 trading days per year).

c. Comparison between Implied and Historical Volatility

Similarities

1. Both implied and historical volatility measure the extent of variation in the price of a financial instrument.

2. Expressed as percentages, they indicate the level of risk or uncertainty associated with the asset's price movements.

Differences

1. Implied volatility is forward-looking, reflecting market expectations, especially in options pricing. Historical volatility is backward-looking, offering a statistical measure of past price movements.
2. Implied volatility is derived from market options prices, while historical volatility is calculated based on past price movements.
3. Implied volatility is often used assess the market's expectation of future price swings. Whereas historical can be employed in risk management and portfolio optimization.

2.2. Newton-Raphson Iteration

The Newton-Raphson iteration is an iterative numerical method for finding roots of a real-valued function, commonly used in finance to estimate implied volatility, especially for options. Applied to the Black-Scholes option pricing model, it involves the derivative of the option price with respect to volatility (Vega).

For a non-dividend European call option, the Black-Scholes formula is:

$c = S_0 * N(d_1) - K e^{-rT} * N(d_2)$

Where:

c: call option price

S_0: current stock price

K: strike price

r: risk-free rate

T: time to expiration

$N(d_1)$ and $N(d_2)$ are cumulative distribution functions of the standard normal distribution.

$$d_1 = \frac{\ln\left(\frac{So}{K}\right) + (r + \frac{\sigma^2}{2})T}{\sigma\sqrt{T}} \qquad d_2 = d_1 - \sigma\sqrt{T}$$

However, it is not possible to invert the Black-Scholes model to express σ as a function of S, K, r, T, and c. Therefore, Newton-Raphson iteration search procedure is used to estimate the implied volatility (σ). It's an effective numerical method for solving the non-linear equation involved in the Black-Scholes option pricing model.

Steps to estimate implied volatility with iteration:

1. Start with the initial guess for the implied volatility, σ_0
2. Calculate the theoretical call option price using the Black-Scholes formula.
3. Find the difference between theoretical and market call option prices.
4. Update implied volatility guess using formula: $\sigma_{i+1} = \sigma_i - \frac{f(\sigma_i)-c}{f'(\sigma_i)}$

   Where, f is the European option formula, and $f'(\sigma_i)$ denotes the first derivative of f (option price) with respect to σ (Vega).
5. Repeat the iteration to produce successively better estimates σ_1, σ_2, σ_3, …, until the difference $[f(\sigma_i) - c]$ becomes close to zero or less than tolerance.

2.3. Newton-Raphson Iteration using Python

Step 1: Import the necessary Python libraries.

In [13]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as si #library to simplify scientific computing tasks
import yfinance as yf
import os
```

Step 2: Apply Newton-Raphson iteration method.

Define a function called "**newton_vol_call**" to compute the implied volatility using Black Scholes Model for European Call Option. Use 1000 as max iteration; tolerance 0.000001 and initial volatility as 25% (sigma = 0.25).

The Newton-Raphson Method to estimate Implied Volatility

In [14]:
```
def newton_vol_call(S, K, T, C, r):

    #S: spot price
    #K: strike price
    #T: time to maturity
    #C: Call value
    #r: risk free rate
    #sigma: volatility of underlying asset

    MAX_ITERATIONS = 1000
    tolerance = 0.000001

    sigma = 0.25

    for i in range(0, MAX_ITERATIONS):
        d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
        d2 = (np.log(S / K) + (r - 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
        price = S * si.norm.cdf(d1, 0.0, 1.0) - K * np.exp(-r * T) * si.norm.cdf(d2, 0.0, 1.0)
        vega = S * np.sqrt(T) * si.norm.pdf(d1, 0.0, 1.0)

        diff = C - price

        if (abs(diff) < tolerance):
            return sigma
        else:
            sigma = sigma + diff/vega

        # print(i,sigma,diff)

    return sigma
```

Step 3: Assign values to remaining parameters of the European call option to calculate Implied Volatility.

```
In [23]: # Example call option parameters
         call_price_market = 10.0
         S0 = 100.0                  # current stock price
         K = 100.0                   # option strike price
         T = 1.0                     # time to expiration
         r_interest_rate = 0.01      # assume a risk-free rate of 1%

         impvol = newton_vol_call(100, 100, 1/12, 10, 0.01)
         print('The implied volatility is', round(impvol*100,2) , '% for the one-month call with strike $ 100' )

         The implied volatility is 86.73 % for the one-month call with strike $ 100
```

2.4. Implied Volatility of Amazon's Option

Step 1: Import relevant libraries.

```
In [24]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import scipy.stats as si
         import yfinance as yf
         import os
```

Step 2: Retrieve Amazon's stock data to python and extract last traded price as spot price, 149.10.

```
In [2]: AMZN = yf.download("AMZN", start="2023-01-10", end="2024-01-09")

        [**********100%%**********]  1 of 1 completed
```

```
In [3]: S = AMZN['Adj Close'][-1]
        print('The spot price is $', round(S,2), '.')

        The spot price is $ 149.1 .
```

Step 3: Define the command to compute implied volatility using Newton-Raphson iteration, with maximum 1000 iterations, tolerance of 0.000001 and initial volatility (sigma) as 25%.

Implied Volatility of Amazon Stock

```
In [29]: def newton_vol_call(S, K, T, C, r):

             #S: spot price
             #K: strike price
             #T: time to maturity
             #C: Call value
             #r: risk free rate
             #sigma: volatility of underlying asset

             MAX_ITERATIONS = 1000
             tolerance = 0.000001

             sigma = 0.25

             for i in range(0, MAX_ITERATIONS):
                 d1 = (np.log(S / K) + (r + 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
                 d2 = (np.log(S / K) + (r - 0.5 * sigma ** 2) * T) / (sigma * np.sqrt(T))
                 price = S * si.norm.cdf(d1, 0.0, 1.0) - K * np.exp(-r * T) * si.norm.cdf(d2, 0.0, 1.0)
                 vega = S * np.sqrt(T) * si.norm.pdf(d1, 0.0, 1.0)

                 diff = C - price

                 if (abs(diff) < tolerance):
                     return sigma
                 else:
                     sigma = sigma + diff/vega

                 # print(i,sigma,diff)

             return sigma
```

Step 3: Access Amazon option data chain in Python for options maturing on January 19, 2024.

```
In [32]: AMAZON = yf.Ticker("AMZN")
         opt = AMAZON.option_chain('2024-01-19') # downlaod Amazon option data for maturity at 19th January 2024
         opt.calls
```

Out[32]:

	contractSymbol	lastTradeDate	strike	lastPrice	bid	ask	change	percentChange	volume	openInterest	impliedVolatility	inTheMoney	co
0	AMZN240119C00045000	2024-01-09 20:13:55+00:00	45.0	106.00	106.25	106.55	4.599998	4.536488	3	325	2.531254	True	
1	AMZN240119C00050000	2024-01-08 19:48:01+00:00	50.0	98.93	101.20	101.65	0.000000	0.000000	16	1928	2.460941	True	
2	AMZN240119C00052000	2024-01-08 14:36:16+00:00	52.0	93.95	99.25	99.60	0.000000	0.000000	2	3291	2.375004	True	
3	AMZN240119C00053000	2024-01-03 14:32:17+00:00	53.0	97.25	98.25	98.65	0.000000	0.000000	31	1865	2.437504	True	
4	AMZN240119C00054000	2023-12-26 14:36:00+00:00	54.0	99.67	97.25	97.60	0.000000	0.000000	100	1301	2.296879	True	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
351	AMZN240119C05000000	2022-06-03 17:49:58+00:00	5000.0	37.09	33.05	41.05	-1.979999	-5.067826	7	474	13.468996	False	
352	AMZN240119C05100000	2022-06-03 14:41:34+00:00	5100.0	31.87	30.30	38.05	0.930000	3.005819	2	108	13.215090	False	
353	AMZN240119C05200000	2022-06-03 13:51:00+00:00	5200.0	34.20	27.75	35.30	3.200001	10.322583	1	437	12.976809	False	
354	AMZN240119C05300000	2022-06-02 13:36:26+00:00	5300.0	29.00	25.65	32.95	0.000000	0.000000	1	698	12.776369	False	
355	AMZN240119C05400000	2022-06-03 19:12:26+00:00	5400.0	26.00	23.60	30.65	-3.850000	-12.897823	23	2045	12.574465	False	

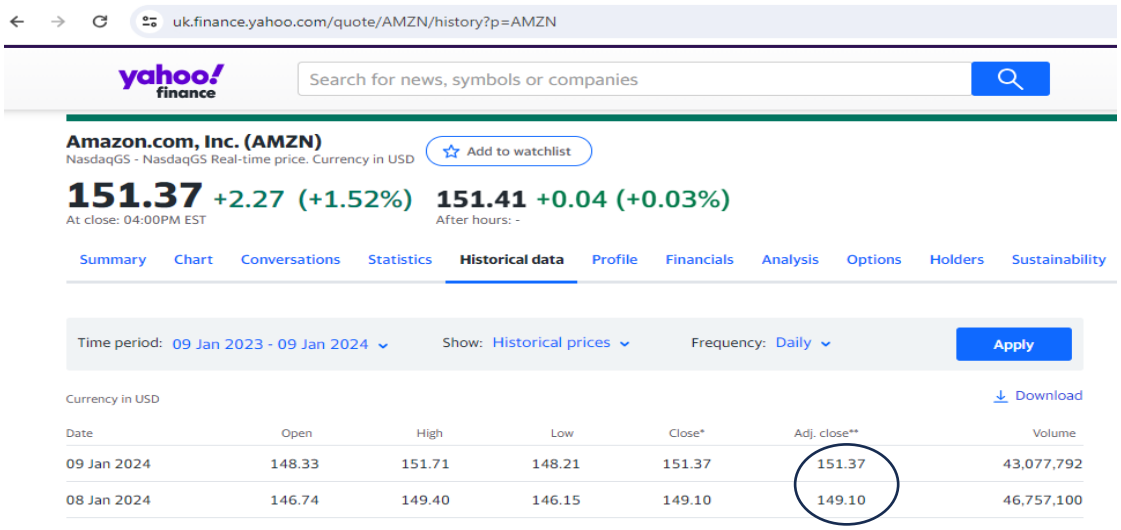
356 rows × 14 columns

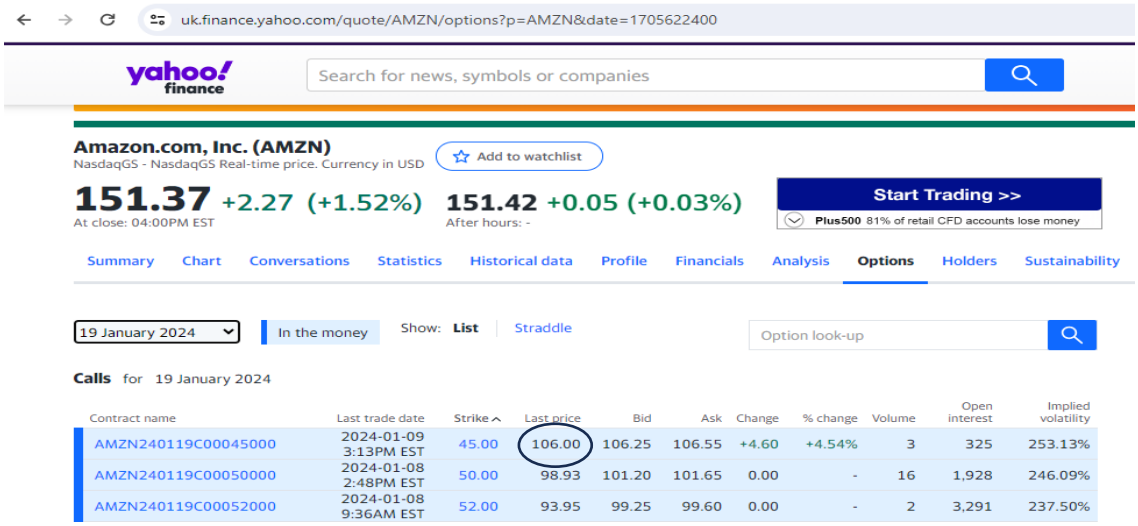
Step 4: Calculate the Implied Volatility, given that strike price K = $165, expiry date is 19th January 2024, risk-free rate of 5.47%, and time to maturity of 10 days from current date (09th January 2024). Spot price and call price have been obtained from yfinance.

```
In [43]: # Amazon call option parameters
         call_price_market = 106       # market call option price
         S0 = 149.1                    # current stock price
         K = 165.0                     # option strike price
         T = 10                        # time to expiration until 19th January 2024, n/252 trading days in year
         r_interest_rate = 0.0547      # a 3-month T-bill rate (annualized risk-free rate) of 5.47%


         impvol = newton_vol_call(S, 165, 10/252, float(opt.calls.lastPrice[opt.calls.strike == 165.00]), 0.0547)
         print('The implied volatility is', round(impvol*100,2) , '% for 10 days call with strike $ 165.00')
```

```
The implied volatility is 25.12 % for 10 days call with strike $ 165.00
```

Below are the screenshots for Spot price and call option price of Amazon taken from yfinance:





Source: (Yahoo Finance, n.d.)

2.5. Annual Historical Volatility of Amazon's Stock Return

The Historical Volatility of Amazon stock by using python is calculated as below:

```
In [24]: import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         import scipy.stats as si
         import yfinance as yf
         import os
```

Historical Volatility of Amazon Stock

```
In [26]: AMZN = yf.download("AMZN", start="2023-01-10", end="2024-01-09")
         [**********100%%**********]  1 of 1 completed

In [27]: S = AMZN['Adj Close'][-1]
         print('The spot price is $', round(S,2), '.')
         The spot price is $ 149.1 .

In [28]: log_return = np.log(AMZN['Adj Close'] / AMZN['Adj Close'].shift(1))
         vol_h = np.sqrt(252) * log_return.std()                    # 252 trading days in a year
         print('The annualised historical volatility is', round(vol_h*100,2), '%')
         The annualised historical volatility is 32.78 %
```

Comparison of Amazon's Implied and Historical Volatility

When implied volatility is significantly higher than the average historical levels, options premiums are assumed to be overvalued. Elevated premiums benefit option writers, who can capitalize on inflated premiums in high implied volatility scenarios. Conversely, option buyers gain an advantage when implied volatility is significantly lower than historical volatility, indicating undervalued premiums (Hayes, 2022).

In the Amazon example, historical volatility (32.78%) is higher than implied volatility (25.12%), indicating that market may be pricing options with a lower expectation of future volatility compared to recent historical price movements.

Options with lower implied volatility may be viewed as undervalued, presenting an opportunity for traders to buy options at a perceived discount. This anticipation is based on the expectation that future price movements could surpass what is currently reflected in the options' prices. The disparity between implied and historical volatility may also be influenced by upcoming events, such as earnings reports or significant news, where traders foresee an opportunity to position themselves before an anticipated spike in volatility.

3. Binomial Tree Option Pricing

A binomial tree considers two outcomes at each stage, creating branching possibilities for evolving stock prices with either an upward or downward movement. To construct the tree, calculate the up factor, down factor, and risk-neutral probability.

3.1. Calculation: u, d, and the risk-neutral probability p for a four-step tree

Considering a non-dividend-paying stock

Spot Price (S) = $100

Volatility (σ) = 20%

Risk-free rate (r) = 5% per annum

Time to maturity (T) = one year

Nodes (n) = 4 (four 3-month periods)

Time step (Δt) = $\frac{T}{n} = \frac{1}{4}$ = 0.25

a. Up factor

$$u = e^{\sigma\sqrt{\Delta t}}$$

r = risk-free rate = 5% or 0.05

σ = volatility = 20% or 0.2

Δt = 0.25 for each node

$$u = e^{0.2*0.25} = 1.105$$

b. Down factor

$$d = \frac{1}{u}$$

$$d = \frac{1}{1.105} = 0.905$$

c. Risk-neutral Probability

Probability of up factor:

$$p_u = \frac{e^{r\Delta t} - d}{u-d}$$

$$p_u = \frac{e^{0.05*0.25} - 0.905}{1.105 - 0.905} = 0.538 = 54\%$$

Probability of down factor:

$$p_d = 1 - p_u$$

$$p_d = 1 - 0.54 = 0.46 = 46\%$$

Therefore, the up factor is 1.105, the down factor is 0.905, and the risk-neutral probabilities are 54% for an up movement and 46% for a down movement.

3.2. Four-step Binomial Tree for European Call Option

From the previous section, the parameters to construct a four-step tree is as follows:

Spot Price = $100

Strike price (K) = $100

u = 1.105

d = 0.905

p = 54%

Stock Price at Time-steps:

Node 0:

S_0 = $100

Node 1:

S_0*u = $100 * 1.105 = $110.50, S_0*d = $100 * 0.905 = $90.50

Node 2:

S_0*u*u = $100 * 1.105 * 1.105 = $122.10, S_0*u*d = $100 * 1.105 * 0.905 = $100

S_0*d*u = $100 * 0.905 * 1.105 = $100, S_0*d*d = $100 * 0.905 * 0.905 = $81.90

Node 3:

S_0*u*u*u = $100 *$(1.105)^3$ = $134.92, S_0*u*u*d = $100 * $(1.105)^2$ * 0.905 = $110.50

S_0*u*d*u =$100*1.105*0.905*1.105= $110.50, S_0*u*d*d = $100*1.105*$(0.905)^2$ = $90.50

$S_0*d*u*u = \$100*0.905*(1.105)^2 = \$110.50$, $S_0*d*u*d = \$100*0.905*1.105*0.905 = \$90.50$

$S_0*d*d*u = \$100*(0.905)^2*1.105 = \$90.50$, $S_0*d*d*d = \$100*(0.905)^3 = \$74.12$

Node 4:

$S_0*u*u*u*u = \$100 * (1.105)^4 = \$149.09$, $S_0*u*u*u*d = \$100*(1.105)^3*0.905 = \$122.11$

$S_0*u*u*d*u = \$100*(1.105)^3*0.905 = \$122.11$, $S_0*u*u*d*d = \$100*(1.105)^2*(0.905)^2 = \$100$

$S_0*u*d*u*u = \$100*(1.105)^3*0.905 = \$122.11$, $S_0*u*d*u*d = \$100*(1.105)^2*(0.905)^2 = \$100$

$S_0*u*d*d*u = \$100*(1.105)^2*(0.905)^2 = \$100$, $S_0*u*d*d*d = \$100*1.105*(0.905)^3 = \$81.90$

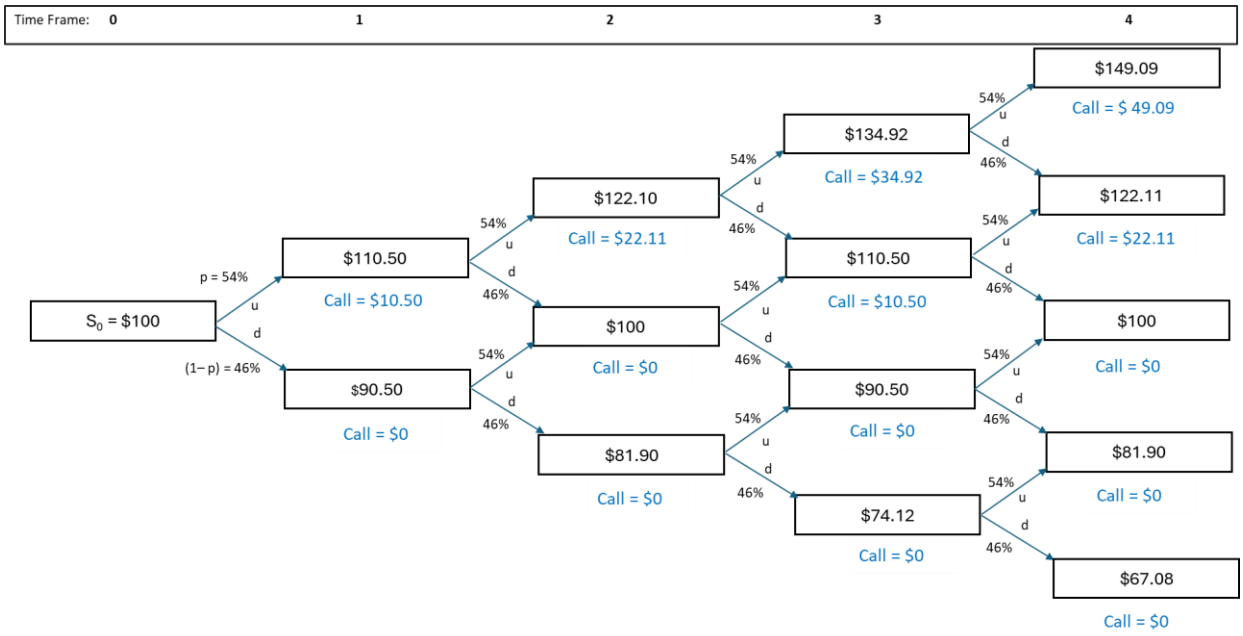
$S_0*d*u*u*u = \$100*0.905*(1.105)^3 = \$122.11$, $S_0*d*u*u*d = \$100*(1.105)^2*(0.905)^2 = \$100$

$S_0*d*u*d*u = \$100*(0.905)^2*(1.105)^2 = \$100$, $S_0*d*u*d*d = \$100*(0.905)^3*1.105 = \$81.90$

$S_0*d*d*u*u = \$100*(0.905)^2*(1.105)^2 = \$100$, $S_0*d*d*u*d = \$100*(0.905)^3*1.105 = \$81.90$

$S_0*d*d*d*u = \$100*(0.905)^3*1.105 = \$81.90$, $S_0*d*d*d*d = \$100*(0.905)^4 = \$67.08$

If the stock price at any time-step exceeds the strike price ($S_T > \$100$), the call option will be exercised. For example, at a stock price of \$149.09 (node 4,1), the call option will be exercised with a payoff of \$49.09. However, if S_T is less than strike price of \$100, the call option will not be exercised, resulting in a 0 payoff.



The option price (c) is considered equal to its expected payoff in a risk-neutral world, discounted at the risk-free rate (Hull, 2021). Therefore,

$$c = e^{-r\Delta t}[pf_u + (1-p)f_d]$$

Where,

r is the risk-free rate = 5%

T is the time to maturity = 3/12 = 1/4 = 0.25

p is the risk neutral probability = 54%

payoff of call option is [Max(S_i – K, 0)

f_u is payoff if the stock price goes up = S_0*u – K = \$100*(1.105) – \$100 = \$10.50

f_d is payoff if the stock price goes down = S_0*d – K = =\$100*0.905 - \$100 = \$0

Hence, $c = e^{-0.05*3/12}[54\%*\$10.50 + 46\%*\$0] = \$5.6$

3.3. Binomial Tree in Python

Step 1: To calculate the binomial tree in Python, import the numpy and os libraries, and define the parameters:

Binomial Tree in Python

```
In [32]: import numpy as np
         import os
```

```
In [33]: S0 = 100                # spot stock price
         K = 100                 # strike
         T = 0.25                # maturity
         r = 0.05                # risk free rate
         sigma = 0.2             # diffusion coefficient or volatility
         N = 4                   # number of periods or number of time steps
         payoff = "call"         # payoff
```

Step 2: Calculate the up factor and down factor:

```
In [34]: dT = float(T) / N                              # Delta t
         u = np.exp(sigma * np.sqrt(dT))                # up factor
         d = 1.0 / u                                    # down factor
         print(f"Up factor (u): {u}")
         print(f"Down factor (d): {d}")
```

```
Up factor (u): 1.0512710963760241
Down factor (d): 0.9512294245007139
```

Step 3: Calculate the stock prices at each node:

```
In [35]: S = np.zeros((N + 1, N + 1))
         S[0, 0] = S0
         z = 1
         for t in range(1, N + 1):
             for i in range(z):
                 S[i, t] = S[i, t-1] * u
                 S[i+1, t] = S[i, t-1] * d
             z += 1
         S
```

```
Out[35]: array([[100.        , 105.12710964, 110.51709181, 116.18342427,
                 122.14027582],
                [  0.        ,  95.12294245, 100.        , 105.12710964,
                 110.51709181],
                [  0.        ,   0.        ,  90.4837418 ,  95.12294245,
                 100.        ],
                [  0.        ,   0.        ,   0.        ,  86.07079764,
                  90.4837418 ],
                [  0.        ,   0.        ,   0.        ,   0.        ,
                  81.87307531]])
```

Step 4: Calculate risk-neutral probability

```
In [36]: a = np.exp(r * dT)    # risk free compound return
         p = (a - d)/ (u - d)  # risk neutral up probability
         q = 1.0 - p           # risk neutral down probability
         p

Out[36]: 0.5187884451462663
```

Step 6: Calculate call option payoff at maturity and then at each node:

```
In [37]: S_T = S[:,-1]
         V = np.zeros((N + 1, N + 1))
         if payoff =="call":
             V[:,-1] = np.maximum(S_T-K, 0.0)
         elif payoff =="put":
             V[:,-1] = np.maximum(K-S_T, 0.0)
         V

Out[37]: array([[ 0.        ,  0.        ,  0.        ,  0.        , 22.14027582],
                [ 0.        ,  0.        ,  0.        ,  0.        , 10.51709181],
                [ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ],
                [ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ],
                [ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ]])
```

```
In [38]: # for European Option
         for j in range(N-1, -1, -1):
             for i in range(j+1):
                 V[i,j] = np.exp(-r*dT) * (p * V[i,j + 1] + q * V[i + 1,j + 1])
         V

Out[38]: array([[ 4.37533143,  7.11074523, 11.14014275, 16.4954365 , 22.14027582],
                [ 0.        ,  1.45477235,  2.81294936,  5.43912186, 10.51709181],
                [ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ],
                [ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ],
                [ 0.        ,  0.        ,  0.        ,  0.        ,  0.        ]])
```

Step 7: Calculate the discounted European Call option price:

```
In [39]: print('European ' + payoff, str( V[0,0]))

         European call 4.375331430623895
```

3.4. Comparison of Results

The call option price with manual computation is $5.60, while Python computes it as $4.375. The difference in value is due to rounding-off of values at each step in manual calculation, Python's floating-point arithmetic ensures higher precision.

In practice, using programming languages like Python for financial calculations, especially involving complex mathematical models like the binomial tree, has several advantages:

1. **Precision and Accuracy:** Python uses floating-point arithmetic that provides higher precision compared to manual calculations. Rounding errors in manual calculations can accumulate, leading to discrepancies, especially in iterative processes like the binomial tree.

2. **Scalability:** Python code allows to easily scale the calculations by changing parameters, such as the number of time steps (n), without introducing errors associated with manual recalculations.
3. **Readability and Maintainability:** Python code is more readable and easier to maintain than a manual calculation, especially for complex financial models.
4. **Error Reduction:** Automation using Python reduces the chances of human errors, which are more likely to occur in manual calculations.
5. **Experimentation:** Python allows for easy experimentation with different parameter values, facilitating sensitivity analysis and scenario testing.

While manual calculations are valuable for understanding the principles and mechanics of financial models, automation through programming becomes essential for real-world applications, where precision, efficiency, and scalability are crucial. Python, with its extensive libraries and tools, provides an excellent platform for quantitative finance and risk analysis.

References

Cepalia, A., n.d. *Scripts, Modules, Packages, and Libraries.* [Online]
Available at: https://realpython.com/lessons/scripts-modules-packages-and-libraries/
[Accessed 08 January 2024].

Chin, C., 2023. *Python with Pandas, Matplotlib, and NumPy: Your Trusty Companions in Investment.* [Online]
Available at: https://python.plainenglish.io/python-with-pandas-matplotlib-and-numpy-your-trusty-companions-in-investment-dcd50eadd391
[Accessed 09 January 2024].

GeeksforGeeks, 2024. *Pandas Introduction.* [Online]
Available at: https://www.geeksforgeeks.org/introduction-to-pandas-in-python/
[Accessed 09 January 2024].

Hayes, A., 2022. *Implied Volatility vs. Historical Volatility: What's the Difference?.* [Online]
Available at: https://www.investopedia.com/articles/investing-strategy/071616/implied-vs-historical-volatility-main-differences.asp#:~:text=Implied%20volatility%2C%20as%20its%20name,indexes%20to%20calculate%20price%20changes.
[Accessed 09 January 2024].

Hull, J. C., 2021. Binomial Trees. In: *Options, Futures and Other Derivatives.* England: s.n., pp. 296-297.

Hull, J. C., 2021. Volatility Smiles and Volatility Surfaces. In: *Options, Futures and Other Derivatives.* England: The Journal of Finance, pp. 451-452.

Lundh, F., 2001. Python Standard Library. In: *An Annotated Reference for Python 2.0.* s.l.:O'Reilly & Associates, Inc..

matplotlib, n.d. *Pyplot tutorial.* [Online]
Available at: https://matplotlib.org/stable/tutorials/pyplot.html
[Accessed 09 January 2024].

Meier, H., 2023. *Python for Finance Part 1: Yahoo Finance.* [Online]
Available at: https://www.linkedin.com/pulse/python-finance-part-1-yahoo-henry-meier/
[Accessed 09 January 2024].

NumPy, n.d. *NumPy: the absolute basics for beginners.* [Online]
Available at: https://numpy.org/doc/stable/user/absolute_beginners.html#welcome-to-numpy
[Accessed 09 January 2024].

pandas, n.d. *10 minutes to Pandas.* [Online]
Available at: https://pandas.pydata.org/docs/user_guide/10min.html
[Accessed 09 January 2024].

Python, n.d. *The Python Standard Library.* [Online]
Available at: https://docs.python.org/3/library/index.html
[Accessed 08 January 2024].

Van Rossum, G. & Drake, F. L., 2003. *An Introduction to Python.* 2.2.2 ed. Bristol: Network Theory Limited.

Yahoo Finance, n.d. *Amazon.com, Inc. (AMZN).* [Online]
Available at: https://uk.finance.yahoo.com/quote/AMZN/options?p=AMZN
[Accessed 09 January 2024].