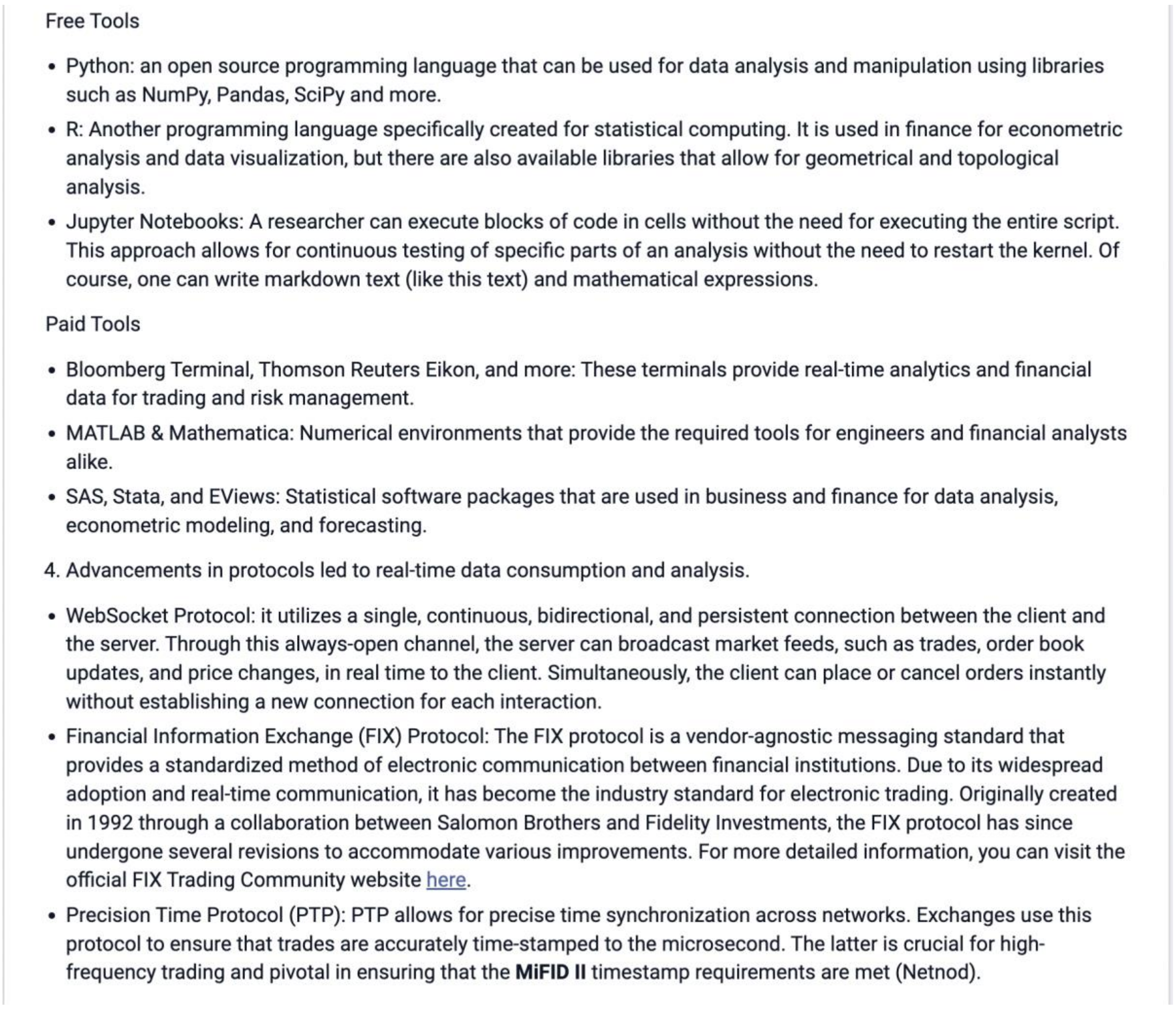
Financial Data best practices:



Free Tools

- Python: an open source programming language that can be used for data analysis and manipulation using libraries such as NumPy, Pandas, SciPy and more.
- R: Another programming language specifically created for statistical computing. It is used in finance for econometric analysis and data visualization, but there are also available libraries that allow for geometrical and topological analysis.
- Jupyter Notebooks: A researcher can execute blocks of code in cells without the need for executing the entire script. This approach allows for continuous testing of specific parts of an analysis without the need to restart the kernel. Of course, one can write markdown text (like this text) and mathematical expressions.

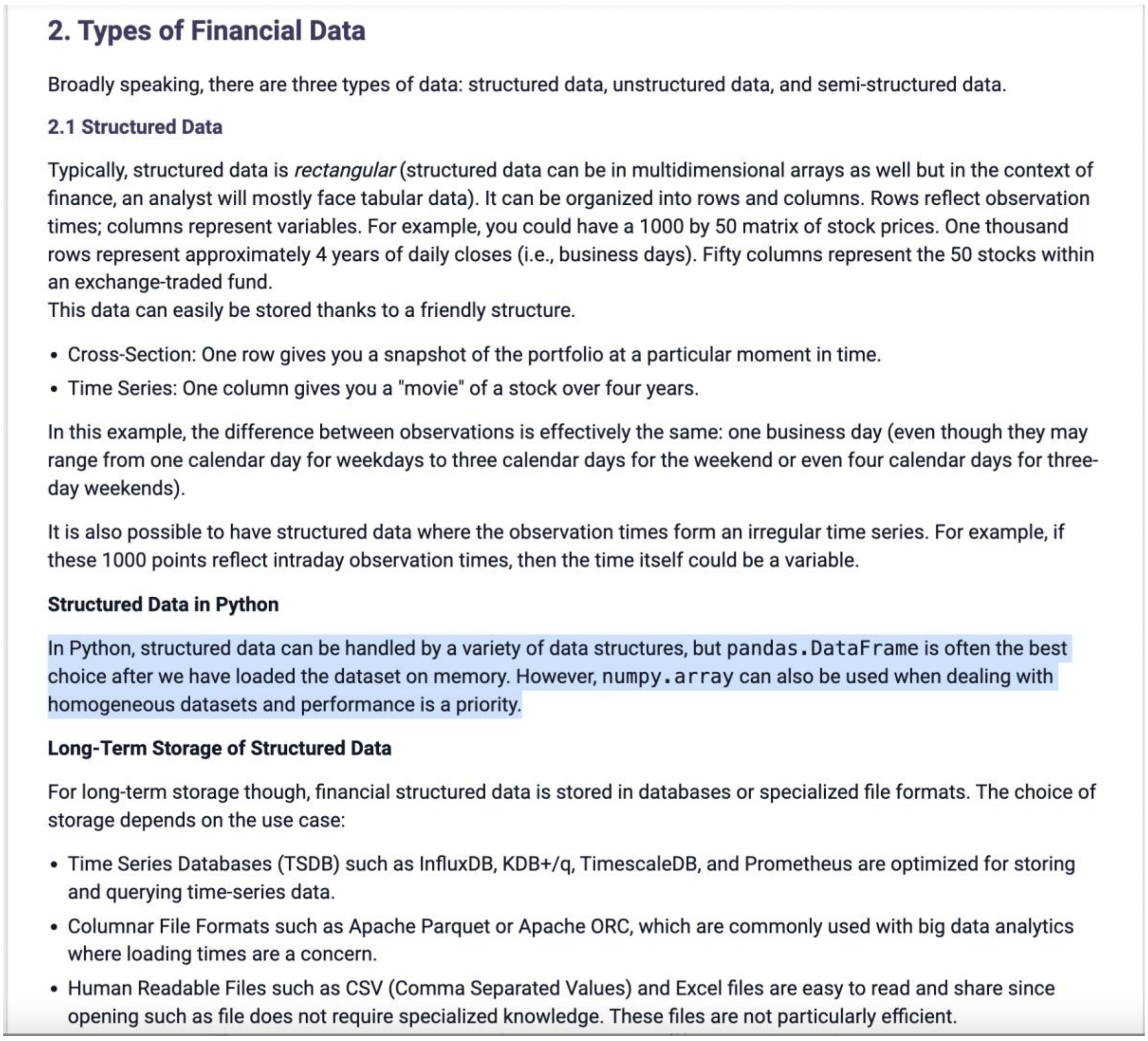
Paid Tools

- Bloomberg Terminal, Thomson Reuters Eikon, and more: These terminals provide real-time analytics and financial data for trading and risk management.
- MATLAB & Mathematica: Numerical environments that provide the required tools for engineers and financial analysts alike.
- SAS, Stata, and EViews: Statistical software packages that are used in business and finance for data analysis, econometric modeling, and forecasting.

4. Advancements in protocols led to real-time data consumption and analysis.

- WebSocket Protocol: it utilizes a single, continuous, bidirectional, and persistent connection between the client and the server. Through this always-open channel, the server can broadcast market feeds, such as trades, order book updates, and price changes, in real time to the client. Simultaneously, the client can place or cancel orders instantly without establishing a new connection for each interaction.
- Financial Information Exchange (FIX) Protocol: The FIX protocol is a vendor-agnostic messaging standard that provides a standardized method of electronic communication between financial institutions. Due to its widespread adoption and real-time communication, it has become the industry standard for electronic trading. Originally created in 1992 through a collaboration between Salomon Brothers and Fidelity Investments, the FIX protocol has since undergone several revisions to accommodate various improvements. For more detailed information, you can visit the official FIX Trading Community website here.
- Precision Time Protocol (PTP): PTP allows for precise time synchronization across networks. Exchanges use this protocol to ensure that trades are accurately time-stamped to the microsecond. The latter is crucial for high-frequency trading and pivotal in ensuring that the **MiFID II** timestamp requirements are met (Netnod).

Data Sources:



2. Types of Financial Data

Broadly speaking, there are three types of data: structured data, unstructured data, and semi-structured data.

2.1 Structured Data

Typically, structured data is *rectangular* (structured data can be in multidimensional arrays as well but in the context of finance, an analyst will mostly face tabular data). It can be organized into rows and columns. Rows reflect observation times; columns represent variables. For example, you could have a 1000 by 50 matrix of stock prices. One thousand rows represent approximately 4 years of daily closes (i.e., business days). Fifty columns represent the 50 stocks within an exchange-traded fund.
This data can easily be stored thanks to a friendly structure.

- Cross-Section: One row gives you a snapshot of the portfolio at a particular moment in time.
- Time Series: One column gives you a "movie" of a stock over four years.

In this example, the difference between observations is effectively the same: one business day (even though they may range from one calendar day for weekdays to three calendar days for the weekend or even four calendar days for three-day weekends).

It is also possible to have structured data where the observation times form an irregular time series. For example, if these 1000 points reflect intraday observation times, then the time itself could be a variable.

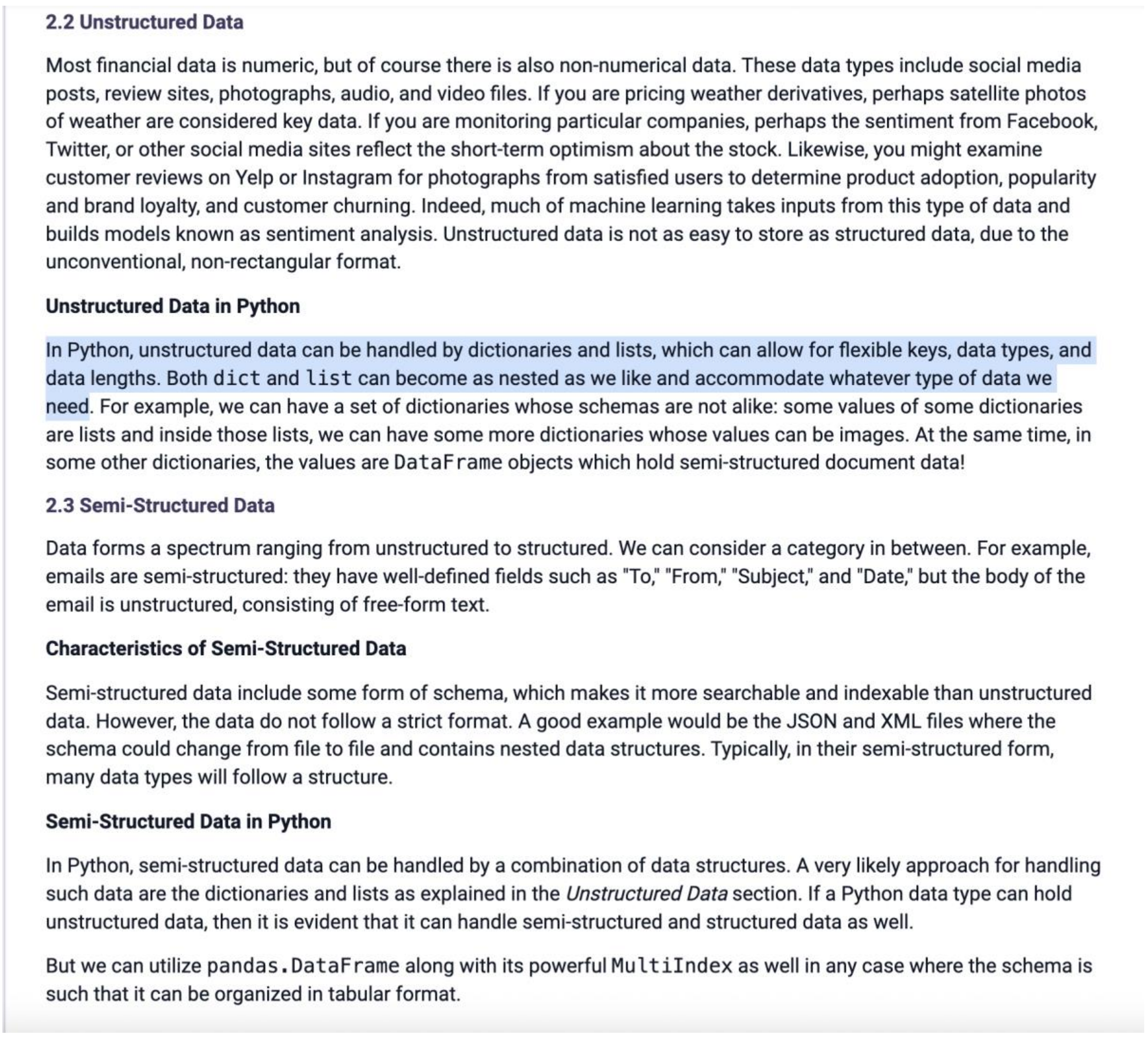
Structured Data in Python

In Python, structured data can be handled by a variety of data structures, but `pandas.DataFrame` is often the best choice after we have loaded the dataset on memory. However, `numpy.array` can also be used when dealing with homogeneous datasets and performance is a priority.

Long-Term Storage of Structured Data

For long-term storage though, financial structured data is stored in databases or specialized file formats. The choice of storage depends on the use case:

- Time Series Databases (TSDB) such as InfluxDB, KDB+/q, TimescaleDB, and Prometheus are optimized for storing and querying time-series data.
- Columnar File Formats such as Apache Parquet or Apache ORC, which are commonly used with big data analytics where loading times are a concern.
- Human Readable Files such as CSV (Comma Separated Values) and Excel files are easy to read and share since opening such as file does not require specialized knowledge. These files are not particularly efficient.



2.2 Unstructured Data

Most financial data is numeric, but of course there is also non-numerical data. These data types include social media posts, review sites, photographs, audio, and video files. If you are pricing weather derivatives, perhaps satellite photos of weather are considered key data. If you are monitoring particular companies, perhaps the sentiment from Facebook, Twitter, or other social media sites reflect the short-term optimism about the stock. Likewise, you might examine customer reviews on Yelp or Instagram for photographs from satisfied users to determine product adoption, popularity and brand loyalty, and customer churning. Indeed, much of machine learning takes inputs from this type of data and builds models known as sentiment analysis. Unstructured data is not as easy to store as structured data, due to the unconventional, non-rectangular format.

Unstructured Data in Python

In Python, unstructured data can be handled by dictionaries and lists, which can allow for flexible keys, data types, and data lengths. Both `dict` and `list` can become as nested as we like and accommodate whatever type of data we need. For example, we can have a set of dictionaries whose schemas are not alike: some values of some dictionaries are lists and inside those lists, we can have some more dictionaries whose values can be images. At the same time, in some other dictionaries, the values are `DataFrame` objects which hold semi-structured document data!

2.3 Semi-Structured Data

Data forms a spectrum ranging from unstructured to structured. We can consider a category in between. For example, emails are semi-structured: they have well-defined fields such as "To," "From," "Subject," and "Date," but the body of the email is unstructured, consisting of free-form text.

Characteristics of Semi-Structured Data

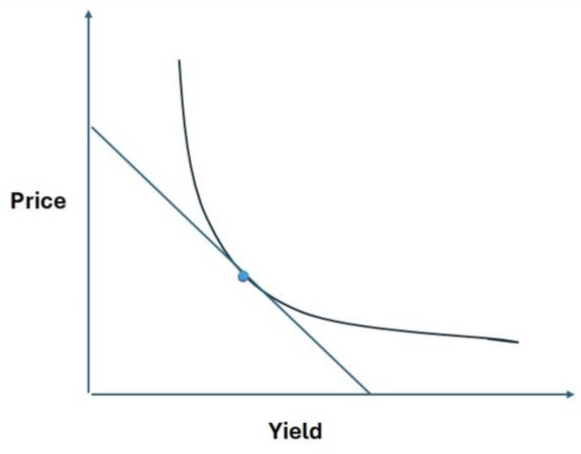
Semi-structured data include some form of schema, which makes it more searchable and indexable than unstructured data. However, the data do not follow a strict format. A good example would be the JSON and XML files where the schema could change from file to file and contains nested data structures. Typically, in their semi-structured form, many data types will follow a structure.

Semi-Structured Data in Python

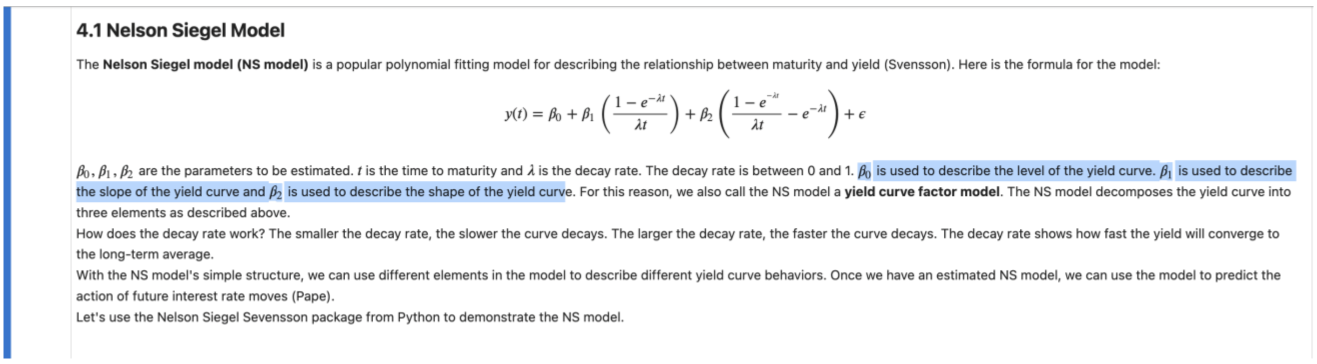
In Python, semi-structured data can be handled by a combination of data structures. A very likely approach for handling such data are the dictionaries and lists as explained in the *Unstructured Data* section. If a Python data type can hold unstructured data, then it is evident that it can handle semi-structured and structured data as well.

But we can utilize `pandas.DataFrame` along with its powerful `MultiIndex` as well in any case where the schema is such that it can be organized in tabular format.

Bond price yield curve:



In Figure 1, we can see that when the yield/interest of a bond increases, the price of the bond will decrease. Conversely, when the yield/interest of a bond decreases, the price of the bond will increase. There is a negative relationship between the bond price and bond yields. However, when yield changes by one unit, the price change varies depending on where the yield level is when yield change happens.
This is another key point to pay attention to in the bond price-yield relationship: the bond price and bond yield do not have a linear relationship; they have a convex relationship. This non-linear relationship between bond price and bond yield has an important implication in interest rate risk management for bond investment. Let's use the following Figure 2 to explain the concept.



4.1 Nelson Siegel Model

The **Nelson Siegel model (NS model)** is a popular polynomial fitting model for describing the relationship between maturity and yield (Svensson). Here is the formula for the model:

$$y(t) = \beta_0 + \beta_1\left(\frac{1-e^{-\lambda t}}{\lambda t}\right) + \beta_2\left(\frac{1-e^{-\lambda t}}{\lambda t} - e^{-\lambda t}\right) + \epsilon$$

$\beta_0, \beta_1, \beta_2$ are the parameters to be estimated. t is the time to maturity and λ is the decay rate. The decay rate is between 0 and 1. β_0 is used to describe the level of the yield curve. β_1 is used to describe the slope of the yield curve and β_2 is used to describe the shape of the yield curve. For this reason, we also call the NS model a **yield curve factor model**. The NS model decomposes the yield curve into three elements as described above.

How does the decay rate work? The smaller the decay rate, the slower the curve decays. The larger the decay rate, the faster the curve decays. The decay rate shows how fast the yield will converge to the long-term average.

With the NS model's simple structure, we can use different elements in the model to describe different yield curve behaviors. Once we have an estimated NS model, we can use the model to predict the action of future interest rate moves (Pape).

Let's use the Nelson Siegel Sevensson package from Python to demonstrate the NS model.

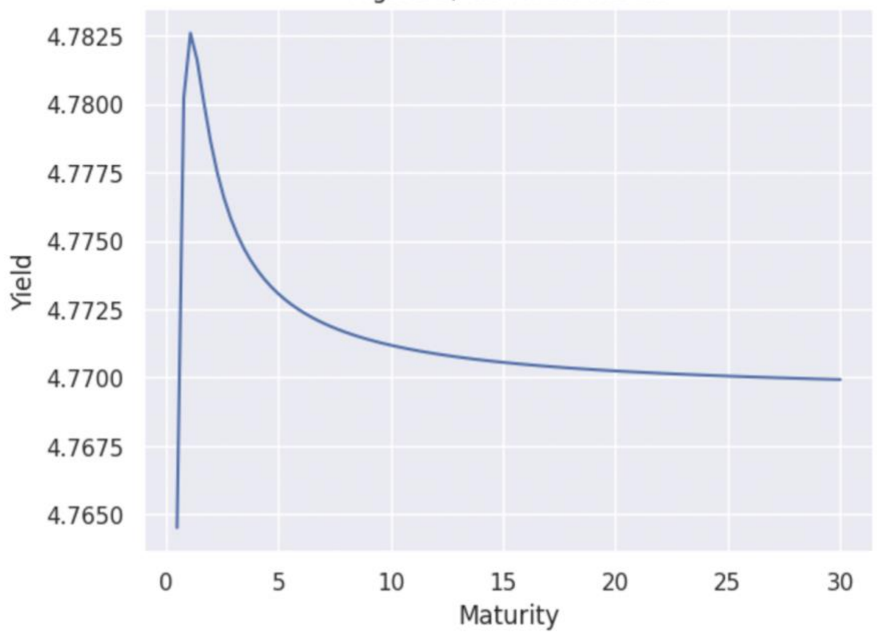
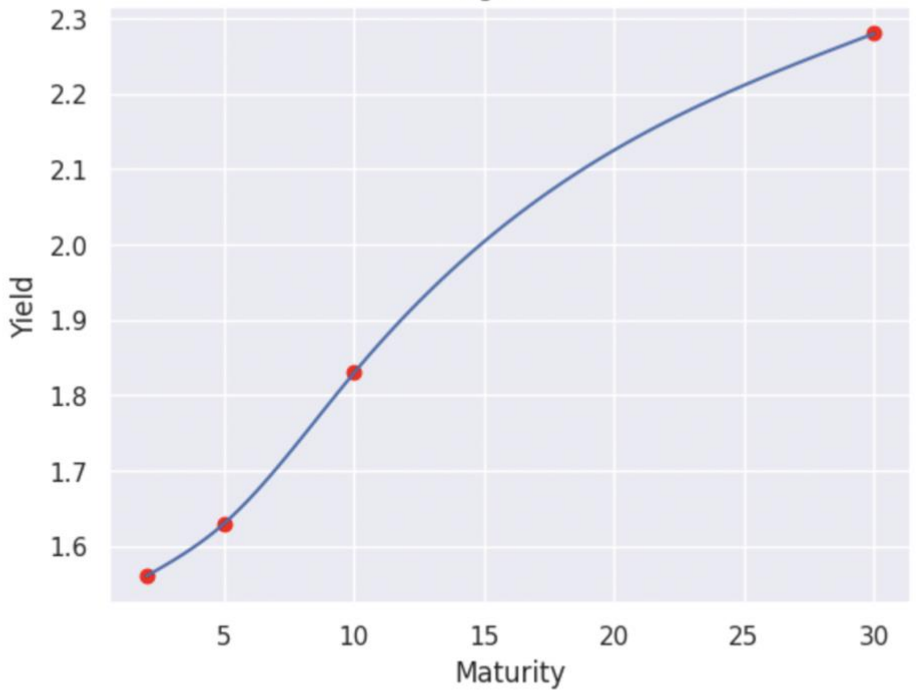


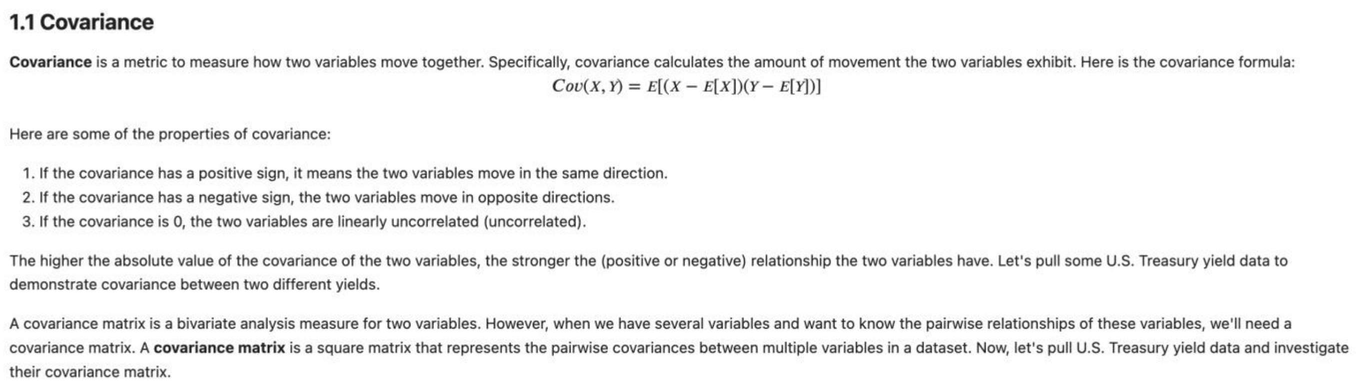
Figure 6, NS Model Result

Cubic spline:

Figure 7



Bivariate Analysis:



1.1 Covariance

Covariance is a metric to measure how two variables move together. Specifically, covariance calculates the amount of movement the two variables exhibit. Here is the covariance formula:

$$Cov(X, Y) = E[(X - E[X])(Y - E[Y])]$$

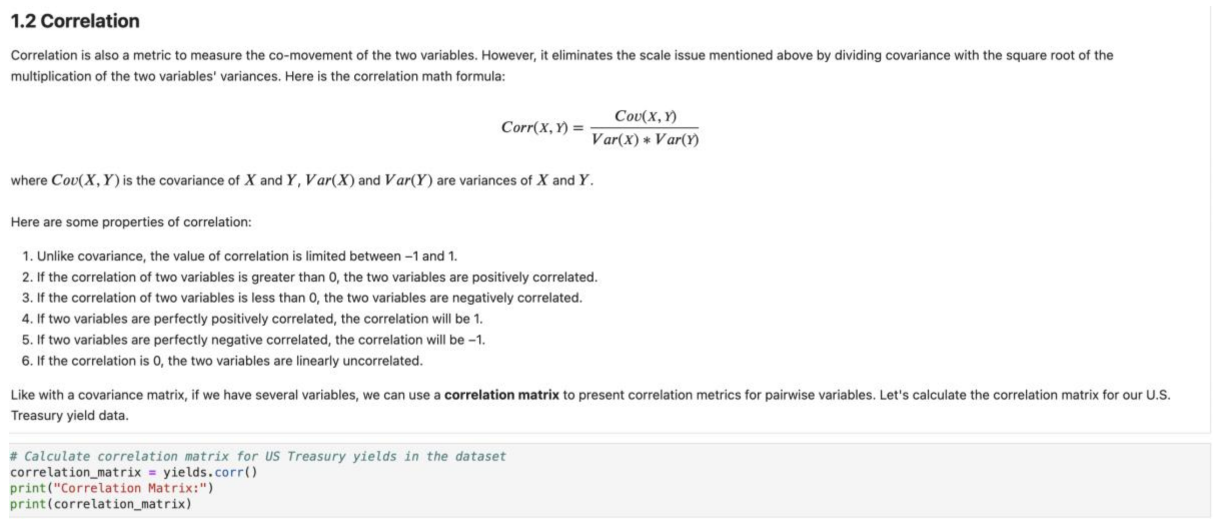
Here are some of the properties of covariance:

1. If the covariance has a positive sign, it means the two variables move in the same direction.
2. If the covariance has a negative sign, the two variables move in opposite directions.
3. If the covariance is 0, the two variables are linearly uncorrelated (uncorrelated).

The higher the absolute value of the covariance of the two variables, the stronger the (positive or negative) relationship the two variables have. Let's pull some U.S. Treasury yield data to demonstrate covariance between two different yields.

A covariance matrix is a bivariate analysis measure for two variables. However, when we have several variables and want to know the pairwise relationships of these variables, we'll need a covariance matrix. A **covariance matrix** is a square matrix that represents the pairwise covariances between multiple variables in a dataset. Now, let's pull U.S. Treasury yield data and investigate their covariance matrix.

However, one downside of covariance is its value changes when the scales of two variables change.



1.2 Correlation

Correlation is also a metric to measure the co-movement of the two variables. However, it eliminates the scale issue mentioned above by dividing covariance with the square root of the multiplication of the two variables' variances. Here is the correlation math formula:

$$Corr(X, Y) = \frac{Cov(X, Y)}{Var(X) * Var(Y)}$$

where $Cov(X, Y)$ is the covariance of X and Y, $Var(X)$ and $Var(Y)$ are variances of X and Y.

Here are some properties of correlation:

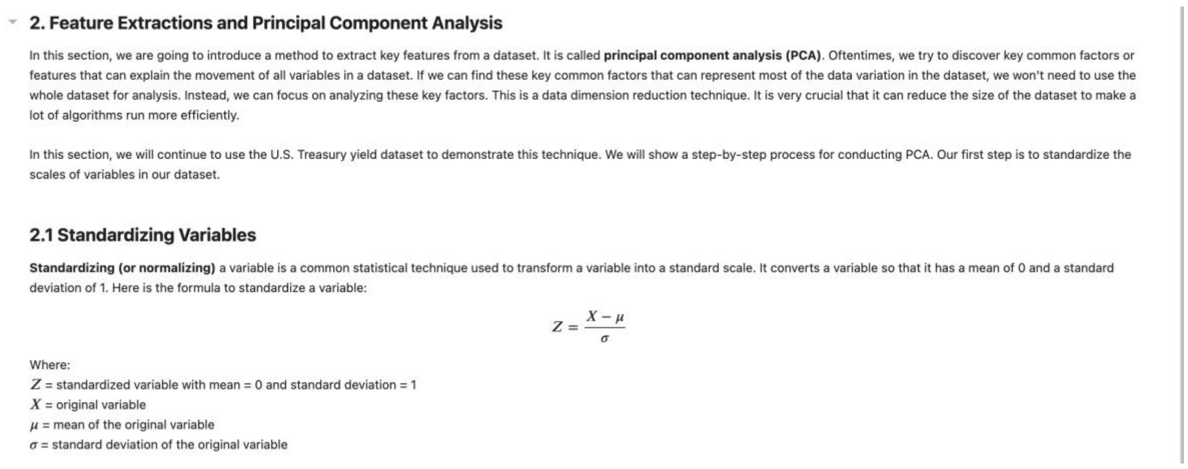
1. Unlike covariance, the value of correlation is limited between −1 and 1.
2. If the correlation of two variables is greater than 0, the two variables are positively correlated.
3. If the correlation of two variables is less than 0, the two variables are negatively correlated.
4. If two variables are perfectly positively correlated, the correlation will be 1.
5. If two variables are perfectly negative correlated, the correlation will be −1.
6. If the correlation is 0, the two variables are linearly uncorrelated.

Like with a covariance matrix, if we have several variables, we can use a **correlation matrix** to present correlation metrics for pairwise variables. Let's calculate the correlation matrix for our U.S. Treasury yield data.

```
# Calculate correlation matrix for US Treasury yields in the dataset
correlation_matrix = yields.corr()
print("Correlation Matrix:")
print(correlation_matrix)
```

Should be sqrt (Var(X)) and sqrt (Var(Y))

PCA:



2. Feature Extractions and Principal Component Analysis

In this section, we are going to introduce a method to extract key features from a dataset. It is called **principal component analysis (PCA)**. Oftentimes, we try to discover key common factors or features that can explain the movement of all variables in a dataset. If we can find these key common factors that can represent most of the data variation in the dataset, we won't need to use the whole dataset for analysis. Instead, we can focus on analyzing these key factors. This is a data dimension reduction technique. It is very crucial that it can reduce the size of the dataset to make a lot of algorithms run more efficiently.

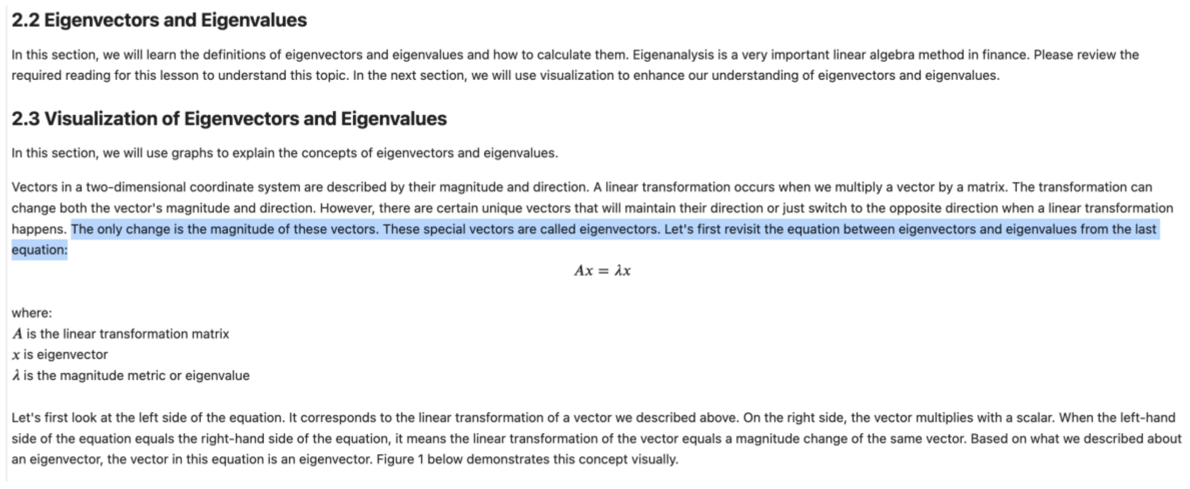
In this section, we will continue to use the U.S. Treasury yield dataset to demonstrate this technique. We will show a step-by-step process for conducting PCA. Our first step is to standardize the scales of variables in our dataset.

2.1 Standardizing Variables

Standardizing (or normalizing) a variable is a common statistical technique used to transform a variable into a standard scale. It converts a variable so that it has a mean of 0 and a standard deviation of 1. Here is the formula to standardize a variable:

$$Z = \frac{X - \mu}{\sigma}$$

Where:
Z = standardized variable with mean = 0 and standard deviation = 1
X = original variable
μ = mean of the original variable
σ = standard deviation of the original variable



2.2 Eigenvectors and Eigenvalues

In this section, we will learn the definitions of eigenvectors and eigenvalues and how to calculate them. Eigenanalysis is a very important linear algebra method in finance. Please review the required reading for this lesson to understand this topic. In the next section, we will use visualization to enhance our understanding of eigenvectors and eigenvalues.

2.3 Visualization of Eigenvectors and Eigenvalues

In this section, we will use graphs to explain the concepts of eigenvectors and eigenvalues.

Vectors in a two-dimensional coordinate system are described by their magnitude and direction. A linear transformation occurs when we multiply a vector by a matrix. The transformation can change both the vector's magnitude and direction. However, there are certain unique vectors that will maintain their direction or just switch to the opposite direction when a linear transformation happens. The only change is the magnitude of these vectors. These special vectors are called eigenvectors. Let's first revisit the equation between eigenvectors and eigenvalues from the last equation:

$$Ax = \lambda x$$

where:
A is the linear transformation matrix
x is eigenvector
λ is the magnitude metric or eigenvalue

Let's first look at the left side of the equation. It corresponds to the linear transformation of a vector we described above. On the right side, the vector multiplies with a scalar. When the left-hand side of the equation equals the right-hand side of the equation, it means the linear transformation of the vector equals a magnitude change of the same vector. Based on what we described about an eigenvector, the vector in this equation is an eigenvector. Figure 1 below demonstrates this concept visually.

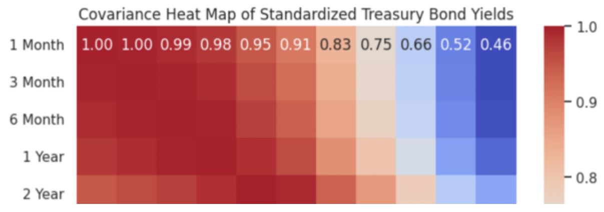
2.4 Derive Principal Components

The next step of PCA is to find the covariance matrix of the standardized dataset. For a standardized dataset, its covariance matrix will be the same as the correlation matrix of the pre-standardized dataset. In the following Python code, we first import the necessary packages for matrix manipulation and for eigenvector/eigenvalue calculation. Then, we will get the covariance matrix for the standardized data and draw a heatmap for the covariance matrix.

```
[19]: import numpy as np
      from numpy import linalg as LA
```

```
[20]: # Calculate covariance matrix of the standardized dataset
      std_data_cov = standardized_data.cov()
```

```
[21]: # Draw a heatmap of the covariance matrix
      plt.figure(figsize=(8, 6))
      sns.heatmap(std_data_cov, annot=True, cmap='coolwarm', fmt=".2f")
      plt.title('Covariance Heat Map of Standardized Treasury Bond Yields')
      plt.show()
```



From the above output, we can see that the eigenvalues and corresponding eigenvectors are ordered in descending order by the values of eigenvalues. In PCA, we also call eigenvectors **loadings**. We will see why this is important later.

We can view this collection of all eigenvectors as a linear transformation matrix to the standardized dataset. The transformed data will have a very interesting feature that we will introduce soon. Let's transform the standardized data with eigenvectors first.

```
# Transform standardized data with Loadings
principal_components = standardized_data.dot(eigenvectors)
principal_components.columns = ["PC_1","PC_2","PC_3","PC_4","PC_5","PC_6","PC_7","PC_8","PC_9","PC_10","PC_11"]
principal_components.head()
```

	PC_1	PC_2	PC_3	PC_4	PC_5	PC_6	PC_7	PC_8	PC_9	PC_10	PC_11
2001-07-31	4.608202	-0.918184	0.138746	-0.223360	0.013690	0.063247	0.008572	0.006278	0.002652	0.017945	-0.002084
2001-08-01	4.665170	-0.950595	0.102492	-0.220182	0.013318	0.054096	0.014505	0.002266	0.000391	0.027474	-0.006491
2001-08-02	4.766161	-1.009754	0.054519	-0.230244	0.021179	0.064157	0.017119	0.011726	-0.002439	0.025895	-0.003628
2001-08-03	4.807092	-1.038602	0.027695	-0.224234	0.025575	0.063723	0.018644	0.020406	0.009374	0.017296	0.002344
2001-08-06	4.781112	-1.044855	0.048161	-0.228694	0.013594	0.051783	0.009102	0.010319	0.003817	0.021904	-0.007042

The above result shows 11 transformed variables. In PCA, they are called **principal components**. Remember we mentioned earlier that the eigenvalues from PCA are in descending order. These principal components are also presented in the same order as their corresponding eigenvalues. For example, PC_1 corresponds to the first eigenvalue, which is 9.22. PC_2 corresponds to the second highest eigenvalue, which is 1.63.

The most important feature of PCA is **the leading principal components can explain higher portions of the variance of the dataset than the rest of the principal components**. For example, PC_1 can explain more variance in the standardized data than PC_2. But by how much? The corresponding eigenvalue for PC_1 is the variance of the whole data explained by PC_1. It is 9.22 in this case. By the same logic, PC_2 explains 1.63 of the total variance of the dataset. We can sum up all the eigenvalues to get the total variance of the data. From the total variance of the data, we can also calculate the percentage of variance contribution each principal component catches. Here is the Python code.

```
# Put data into a DataFrame
df_eigval = pd.DataFrame({"Eigenvalues":eigenvalues}, index=range(1,12))

# Work out explained proportion
df_eigval["Explained proportion"] = df_eigval["Eigenvalues"] / np.sum(df_eigval["Eigenvalues"])
#Format as percentage
df_eigval.style.format({"Explained proportion": "{:.2%}"})
```

	Eigenvalues	Explained proportion
1	9.222362	83.84%
2	1.633410	14.85%
3	0.117348	1.07%
4	0.014675	0.13%
5	0.005372	0.05%
6	0.003701	0.03%
7	0.001613	0.01%
8	0.000697	0.01%
9	0.000412	0.00%
10	0.000247	0.00%
11	0.000162	0.00%

From the above table, we can see PC_1 can explain almost 84% of the variance in the standardized data. PC_2 can explain almost 15% of the variance in the standardized data. The first two leading principal components can explain almost 99% of the variance in the dataset. The rest of the 9 principal components only explain 1% of the variance of the data. Hence, to make data analysis more efficient without losing too much information, we can just use the first two principal components for analysis instead of all 11 principal components. Due to this special feature, we also call PCA a data-dimension reduction technique.

3.1 Value at Risk (VaR)

Value at Risk (VaR) is a statistical metric to measure the potential maximum loss of an investment or a portfolio at a given time period under certain a confidence level. For example, when a stock portfolio has a VaR of $1 milliondringadayat95$1 million in a day.
VaR is an easy-to-understand risk metric and can be used to compare risks across different asset classes. VaR is usually used in risk management for portfolio management or regulartory reporting. It is also part of the metrics to set risk cap for traders. VaR can also be applied for capital allocation.

3.2 VaR for a Simple Treasury Bond Portfolio

We will demonstrate how to calculate VaR for a simple Treasury bond portfolio in this section. This simple bond portfolio will only consist of 2-year Treasury bonds, 5-year Treasury bonds, and 10-year Treastury bonds. First, let's create a dataset with the yields of these three bonds and calculate the daily yield percentage change in these bonds.

```
[ ]: # Create a dataset with 3 Treasury bond yields and calculate the yield changes
     var_dataset = yields[["2 Year","5 Year","10 Year"]]
     var_yield_chng_dataset = var_dataset.pct_change()
     var_yield_chng_dataset = var_yield_chng_dataset.dropna()
     var_yield_chng_dataset.head()
```

Next, we will prepare the dataset. We need to standardize the dataset first.

```
[ ]: # Standardize the dataset
     var_yield_chng_dataset_means = var_yield_chng_dataset.mean()
     var_yield_chng_dataset_stds = var_yield_chng_dataset.std()
     var_yld_chng_stnd_data = (var_yield_chng_dataset - var_yield_chng_dataset_means) / var_yield_chng_dataset_stds
```

Now we can calculate the eigenvectors and eigenvalues of the standardized dataset.

Next, we will calculate bond sensitivities in the portfolio. We assume the bond durations are the same as their maturity for simplicity. Then, we can calculate the portfolio value changes and VAR.

```
# Calculate portfolio sensitivities (assuming duration = maturity for simplicity)
sensitivities = np.array([maturity * amount for maturity, amount in portfolio.items()])

# Calculate portfolio value changes
portfolio_changes = (var_yield_chng_dataset*sensitivities) @ selected_components

# Calculate VaR
confidence_level = 0.95  # 95% VaR
var = -np.percentile(portfolio_changes, 100 * (1 - confidence_level))

print(f"1-day 95% VaR: ${var:,.2f}")

# Display summary statistics
print("\nSummary Statistics:")
print(f"Portfolio Value: ${sum(portfolio.values()):,.2f}")
print(f"VaR as % of Portfolio Value: {var / sum(portfolio.values()) * 100:.3f}%")
```

```
1-day 95% VaR: $458,248.59

Summary Statistics:
Portfolio Value: $5,000,000.00
VaR as % of Portfolio Value: 9.165%
```

The above result shows that the 1-day VaR at 95% confidence level for our simple Treasury bond portfolio is $458,249. It is about 9% of the total portfolio value. The above example demonstrates how to use the feature extraction method to reduce the portfolio dataset and use the smaller dataset to calculate VaR.