

# Supervised Learning for Investment Risk Assessment

**MS-DS, University of Colorado Boulder, Peculiar.D@colorado.edu**

## 1. Introduction

It's difficult for investors to accurately assess the risk of their investments at any given time. They are often bogged down by too many metrics, and market sentiment can be confusing. Traditional risk assessments, such as the Sharpe Ratio and Beta, provide limited insights. This project explores unsupervised machine learning technique, to systematically group stocks into low, medium, and high-risk categories based on key indicators like Beta, volatility, and maximum drawdown. This unsupervised learning approach offers a simplify means of risk assessment.

## 2. Project Goal

This project examines how risk classification evolves over time by experimenting 5 years datasets. The project's results are compared with external risk scores to validate the model's accuracy.

Unlike prior studies that focus on broad financial risk assessment or portfolio diversification, this project tailors K-Means clustering specifically for stock market risk classification. By incorporating multiple financial risk indicators (Beta, volatility, and maximum drawdown) this approach enhances traditional risk assessment methods, offering a better perspective on investment risk.

## 3. Project Data

### 3.1 Data Source

Historical stock closing prices were successfully retrieved from Yahoo Finance using Python for analysis. Initial trials included AAPL (Apple), GOOGL (Google), JPM (JPMorgan Chase), META (Meta Platforms), MSFT (Microsoft), TSLA (Tesla), WMT (Walmart), and XOM (ExxonMobil) and S&P500 (SPY). "SPY" is an Exchange Traded Fund (ETF) that passively tracks the S&P 500 index, which comprises 500 of the largest U.S. companies. The S&P 500 is a commonly used proxy for the overall U.S. stock market performance. Therefore, "SPY" was chosen as the market index for Beta comparisons, allowing for the assessment of individual stock risk relative to the broader market.

To address Yahoo Finance API download limit errors, the dataset was first defined with a three-month timeframe (as of August 2025), followed by a five-year dataset from 2020 to 2025. The dataset was then stored as a CSV file for K-means clustering processing.

The Python script is shown below:

In [128...

```
import numpy as np
import pandas as pd
import yfinance as yf
import warnings

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.cluster import AgglomerativeClustering
from sklearn.decomposition import PCA
from scipy.cluster.hierarchy import dendrogram, linkage

# Load stock data (Example: S&P 500 stocks)
stocks = ['AAPL', 'MSFT', 'TSLA', 'GOOGL', 'JPM', 'XOM', 'WMT', 'META', 'SPY']

start_date = "2020-07-31"
end_date = "2025-08-01"

#####
# This section was commented out as the data was downloaded from Yahoo and saved as
#####
# Download stock data
#data = yf.download(stocks, start=start_date, end=end_date)['Close']

# Save data to CSV
#data.to_csv("stock_data_yahoo.csv")

# To ensure parsing is consistent, specify index_col=0, parse_dates=True.
raw_data = pd.read_csv('stock_data_yahoo.csv', index_col=0, parse_dates=True, date_
```

## 3.2 Data Description

### 3.2.1 First 5 rows of the dataset


The dataset has a time series 'Date' column as the index. The table below displays the first 5 rows of the dataset, obtained using the `head()` function. The dataset contains daily closing stock prices for nine entities: AAPL, GOOGL, JPM, META, MSFT, TSLA, WMT, XOM, and SPY, with each column representing one entity. The Date column serves as the index.

In [129...

```
raw_data.head()
```

Out[129...

	AAPL	GOOGL	JPM	META	MSFT	SPY	TSLA	
Date								
2020-07-31	103.174988	73.953972	84.504875	252.285950	196.417145	304.028687	95.384003	40
2020-08-03	105.774734	73.696022	84.032700	250.585266	207.463821	306.142395	99.000000	40
2020-08-04	106.481102	73.225838	83.551765	248.466904	204.350098	307.324829	99.133331	40
2020-08-05	106.867065	73.513611	85.003304	247.760773	204.014740	309.233612	99.001335	40
2020-08-06	110.595596	74.798904	85.029541	263.832581	207.281799	311.300690	99.305336	40



### 3.2.2 Dataset Summary

`info()` function shows the summary of the five-year dataset and confirms that there are no missing values, as all columns have a non-null count of 1256. As expected, all stock price columns have the data type `float64` to accommodate floating-point numbers.

In [130...

```
raw_data.info()

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1256 entries, 2020-07-31 to 2025-07-31
Data columns (total 9 columns):
 #   Column  Non-Null Count  Dtype
---  -
 0   AAPL    1256 non-null   float64
 1   GOOGL   1256 non-null   float64
 2   JPM     1256 non-null   float64
 3   META    1256 non-null   float64
 4   MSFT    1256 non-null   float64
 5   SPY     1256 non-null   float64
 6   TSLA    1256 non-null   float64
 7   WMT     1256 non-null   float64
 8   XOM     1256 non-null   float64
dtypes: float64(9)
memory usage: 98.1 KB
```

### 3.2.3 Dataset Statistics

`describe()` function presents the descriptive statistics of the dataset. It confirms the complete five-year dataset with 1256 rows for each stock. META exhibit the highest standard deviations, indicating greater price volatility compared to other stocks, like WMT, which has the lowest standard deviation.

In [131...

```
raw_data.describe()
```

Out[131...

	AAPL	GOOGL	JPM	META	MSFT	SPY	
<b>count</b>	1256.000000	1256.000000	1256.000000	1256.000000	1256.000000	1256.000000	1256
<b>mean</b>	167.905860	130.604268	156.695646	350.255804	318.396989	443.649150	243
<b>std</b>	36.273604	31.327491	50.751819	161.194474	82.331232	83.726914	67
<b>min</b>	103.174988	70.049385	81.024651	88.424896	192.454895	301.618561	91
<b>25%</b>	140.274353	104.766687	122.733952	235.781498	247.085953	383.762642	196
<b>50%</b>	165.716248	131.136780	139.942612	312.087860	300.394058	421.030548	237
<b>75%</b>	191.494617	153.929337	190.125843	484.064819	400.295288	510.787209	283
<b>max</b>	258.103729	205.893341	299.630005	773.440002	533.500000	637.099976	479

3.2.4 Dataset Correlation

Using the `corr()` function to generate the correlation matrix of the dataset. It reveals that most stocks have strong positive correlations with each other, particularly with SPY, indicating that their prices tend to move together in line with the overall market trend. SPY exhibits high correlations with all stocks, confirming its role as a reliable market indicator. Notably, MSFT and SPY have the highest correlation (0.96), suggesting that Microsoft's stock price closely mirrors the broader market.

In [132...

```
raw_data.corr()
```

Out[132...

	AAPL	GOOGL	JPM	META	MSFT	SPY	TSLA	WMT	
<b>AAPL</b>	1.000000	0.872149	0.831888	0.759897	0.899263	0.918245	0.494653	0.831585	0.7
<b>GOOGL</b>	0.872149	1.000000	0.882249	0.850855	0.922173	0.942725	0.560523	0.790185	0.5
<b>JPM</b>	0.831888	0.882249	1.000000	0.931153	0.896804	0.967957	0.448948	0.946752	0.6
<b>META</b>	0.759897	0.850855	0.931153	1.000000	0.864529	0.910558	0.367877	0.899927	0.4
<b>MSFT</b>	0.899263	0.922173	0.896804	0.864529	1.000000	0.956914	0.390478	0.828455	0.7
<b>SPY</b>	0.918245	0.942725	0.967957	0.910558	0.956914	1.000000	0.484034	0.915494	0.6
<b>TSLA</b>	0.494653	0.560523	0.448948	0.367877	0.390478	0.484034	1.000000	0.422705	0.1
<b>WMT</b>	0.831585	0.790185	0.946752	0.899927	0.828455	0.915494	0.422705	1.000000	0.6
<b>XOM</b>	0.771971	0.588163	0.615042	0.419617	0.719200	0.689340	0.132739	0.601955	1.0

4. Data Cleaning

4.1 Handling Missing Values

While the preliminary review of the dataset shows no missing values, it is good practice to include a step to handle potential missing data. Stock data can sometimes have missing values due to various reasons (e.g., trading halts, data errors). To address this, we use Mean Imputation, a common technique that replaces missing values with the mean value of the respective feature. This helps maintain the dataset's completeness without significantly distorting the data distribution. `SimpleImputer(strategy="mean")` from scikit-learn is employed to perform mean imputation. However, since no missing values were found, this step will not alter the dataset.

In [133...

```
# Handle missing values using Mean Imputation
imputer = SimpleImputer(strategy="mean")
data_imputed = pd.DataFrame(imputer.fit_transform(raw_data),
                             columns=raw_data.columns, index=raw_data.index)
```

## 5. Feature Selection

The selection of the following key features was finalized for classifying stocks based on risk:

### 5.1 Beta ( $\beta$ )

Beta comes from the Capital Asset Pricing Model (CAPM), developed in the 1960s by William F. Sharpe. Beta measures a stock's price sensitivity to market movements, specifically the returns of a market index (in this case, SPY, which tracks the S&P 500). It quantifies how much a stock's price is expected to move in response to a change in the market. By including Beta as a feature, this project aims to capture and categorize the market risk associated with each stock, enabling investors to make more informed decisions based on their risk tolerance.

$$\beta = \frac{Cov(r_s, r_m)}{Var(r_m)}$$

where:

- $r_s$  = Stock returns
- $r_m$  = Market index returns (e.g., S&P 500)
- $Cov(r_s, r_m)$  = Covariance between stock and market returns
- $Var(r_m)$  = Variance of the market returns
- **If Beta > 1:** Stock is more volatile than the market.
- **If Beta < 1:** Stock is less volatile than the market.
- **If Beta = 1:** Stock moves with the market.

To calculate Beta in Python, we used the `np.cov()` function from the NumPy module to calculate the covariance between each stock's returns and the market returns, as well as the variance of the market returns. Finally, Beta was calculated by dividing the covariance by the market variance.

## 5.2 Volatility ( $\sigma$ )

Volatility was first formalized in the Modern Portfolio Theory (MPT) by Harry Markowitz (1952). Markowitz demonstrated that volatility is a crucial factor in balancing risk and return within a portfolio. Volatility measures how much stock prices fluctuate over time. Higher volatility, the riskier investment and vice versa. By incorporating volatility as a feature, this project aims to capture the inherent price fluctuation risk of each stock.

$$\sigma_{annual} = \sigma_{daily} \times \sqrt{252}$$

where:

- $\sigma_{daily}$  = Standard deviation of daily returns
- 252 is the average number of trading days per year, so this converts daily volatility to an annualized value.
- If volatility = 0.10, this means that 10% price fluctuation per year.

We can use the `std()` and `sqrt()` functions from the NumPy library to calculate annualized volatility.

## 5.3 Maximum Drawdown (MDD)

Maximum Drawdown (MDD) measures the largest percentage decline in value from a peak to a subsequent trough in an investment's price history.

$$MDD = \frac{MaximumValue - MinimumValue}{MaximumValue}$$

It's essentially the "worst-case scenario" or "maximum historical loss" an investor would have experienced within a given period. It provides a clear picture of the potential downside risk of an investment, helping investors understand the magnitude of losses they might face. By adding MDD as a feature, this project aims to capture the historical downside risk associated with each stock.

Stock prices are cumulative in nature, meaning each day's price builds upon the previous price. Since we've computed percentage returns (`pct_change()`) for daily changes, we can sum these up over time to reconstruct the price trend. We can use the cumulative sum function `cumsum()` of percentage returns approximates the stock's movement, though on a different scale.

## 6. Data Preprocessing

## 6.1 Convert Prices to Returns

We convert the raw stock prices into daily percentage changes (returns) using the `pct_change()` function. This step is crucial for the following reasons:

- Beta, volatility, and maximum drawdown are more accurately calculated using returns rather than raw prices. Returns reflect the actual gains or losses experienced by investors, providing a more meaningful measure of risk.
- Using returns allows for a fair comparison of stocks with different price levels.
- By converting price to percentage changes also remove the bias introduced by different price scales, ensuring that all stocks are compared equally.

After calculating the returns, we use `dropna()` to remove any null values. This is necessary because the first row of the returns data will have null values since there is no previous day's price to calculate the percentage change from.

```
In [134... # Calculate percentage returns and drop NA
percentage_returns = raw_data.pct_change().dropna()

# Get market returns
market_index = "SPY" # Use SPY ETF as a proxy for S&P 500
market_index_returns = percentage_returns[market_index]
```

## 6.2 Feature Calculation

After the the percentage returns, proceed to calculate the Beta, Volatility, Max Drawdown for each stock and stored the result in a risk dataframe.

```
In [135... # Function to compute Beta for each stock
def calculate_beta(stock_returns, market_returns):
    cov_matrix = np.cov(stock_returns, market_returns)
    stock_and_market_cov = cov_matrix[0, 1]
    market_var = cov_matrix[1, 1]
    return stock_and_market_cov / market_var

# Compute risk metrics for each stock
risk_data = {}
for stock in stocks:
    stock_return_series = percentage_returns[stock]

    beta = calculate_beta(stock_return_series, market_index_returns)

    # Annualized volatility
    volatility = np.std(stock_return_series) * np.sqrt(252)

    # Maximum Drawdown
    max_drawdown = (stock_return_series.cumsum().max() - stock_return_series.cumsum()

    risk_data[stock] = {"Beta": beta, "Volatility": volatility, "Max Drawdown": max
```

```
# Convert to DataFrame
risk_df = pd.DataFrame.from_dict(risk_data, orient="index")

# Display the table of Risk Dataframe before normalization
risk_df = pd.DataFrame.from_dict(risk_data, orient="index")
print("#####")
print("##### Risk Dataframe BEFORE Normalize #####")
print("#####")
print(risk_df)
print("#####\n")
```

```
#####
##### Risk Dataframe BEFORE Normalize #####
#####
      Beta  Volatility  Max Drawdown
AAPL  1.276181    0.293400    1.068093
MSFT  1.171548    0.266991    1.187671
TSLA  2.011588    0.635830    2.492160
GOOGL 1.247264    0.309792    1.281404
JPM   0.883042    0.255544    1.464502
XOM   0.591282    0.291385    1.735369
WMT   0.489636    0.207802    1.062452
META  1.570949    0.439751    2.383604
SPY   1.000000    0.174909    0.821529
#####
```

## 6.3 Feature Scaling (Normalization)

Feature Scaling (Normalization) is crucial for algorithms like K-Means that rely on distance calculations because features with different scales can disproportionately influence the distance calculations, leading to biased results. Normalization ensures that all features contribute equally to the analysis. Although, it's less crucial with only price values, but scaling can still be beneficial to ensure all features (Beta, Volatility, Max Drawdown) are treated equally and to account for any subtle differences in scale.

This project uses `StandardScaler()` from scikit-learn to standardize the features. This transforms the data to have zero mean and unit variance, effectively bringing all features to a comparable scale.

For each feature  $X$ , `StandardScaler()` applies this formula:  $X_{scaled} = \frac{X - \mu}{\sigma}$

Where:

- $X$  = Original value
- $\mu$  = Mean (average) of the feature
- $\sigma$  = Standard deviation of the feature

After applying `StandardScaler()`, all features will have:

- Mean = 0



- Standard deviation = 1

```
In [136... # Normalize features for K-Means
# Standardize features to ensure equal weight
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
risk_df_scaled = pd.DataFrame(scaler.fit_transform(risk_df), columns=risk_df.columns)

print("#####")
print("##### Risk Dataframe AFTER Normalize #####")
print("#####")
print(risk_df_scaled)
print("#####\n")
```

```
#####
##### Risk Dataframe AFTER Normalize #####
#####
      Beta  Volatility  Max Drawdown
AAPL  0.310311   -0.197959   -0.772950
MSFT  0.075435   -0.398340   -0.558773
TSLA  1.961124    2.400308    1.777701
GOOGL 0.245398   -0.073583   -0.390888
JPM   -0.572193   -0.485203   -0.062940
XOM   -1.227125   -0.213245    0.422210
WMT   -1.455297   -0.847456   -0.783054
META  0.971996    0.912514    1.583265
SPY   -0.309650   -1.097037   -1.214571
#####
```

## 7. Exploratory Data Analysis

ensuring analysis is relevant engineering features data feeding into the actual model.

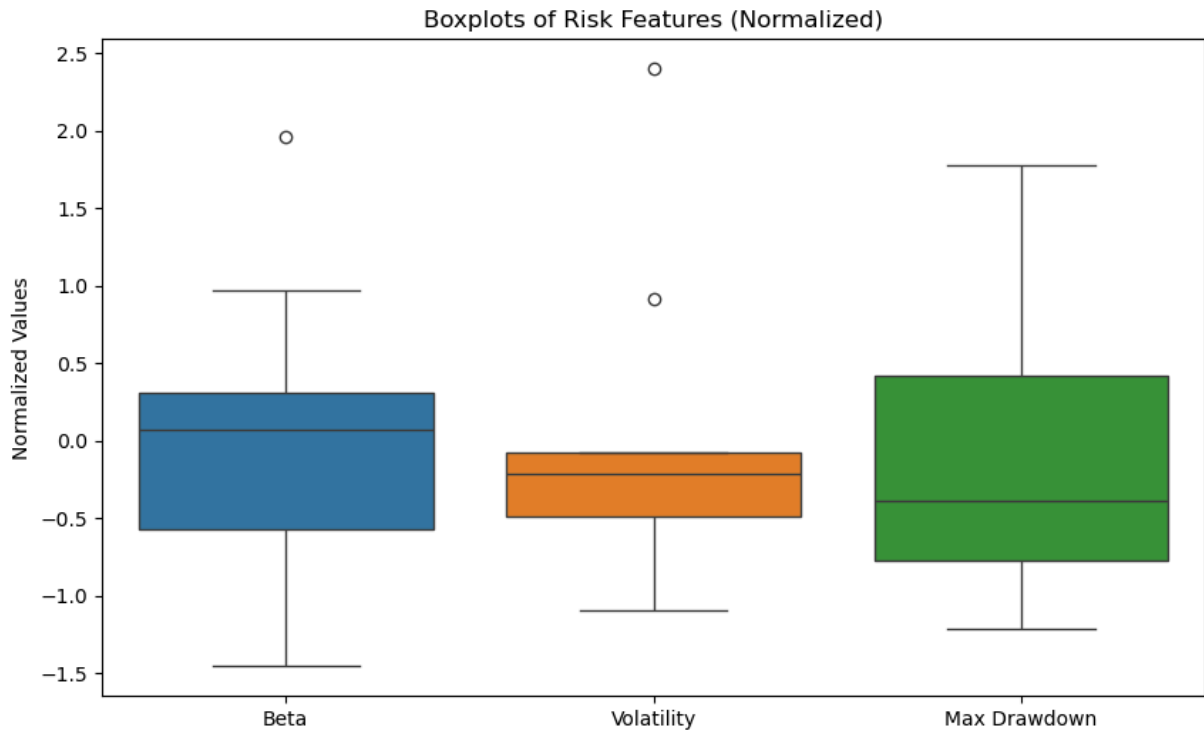
Boxplots to spot outliers (e.g., stocks with extreme volatility). Scatterplots:  $\beta$  vs.  $\sigma$ ,  $\beta$  vs. MDD,  $\sigma$  vs. MDD. Correlation between  $\beta$ ,  $\sigma$ , MDD.

### 7.1 Boxplots for outliers

- The boxplot for Beta shows an outlier around the value 2.0 which corresponds to TSLA with a value of 1.96.
- Volatility has two outliers: around 2.4 and 0.9 which correspond to TSLA (2.40) and META (0.91).
- The boxplot for Max Drawdown has no outliers. The highest value is TSLA at 1.78 and lowest value is SPY at -1.21, which also falls within the whisker range.

```
In [137... # Boxplots for outliers
plt.figure(figsize=(10, 6))
sns.boxplot(data=risk_df_scaled)
plt.title("Boxplots of Risk Features (Normalized)")
```

```
plt.ylabel("Normalized Values")
plt.show()
```

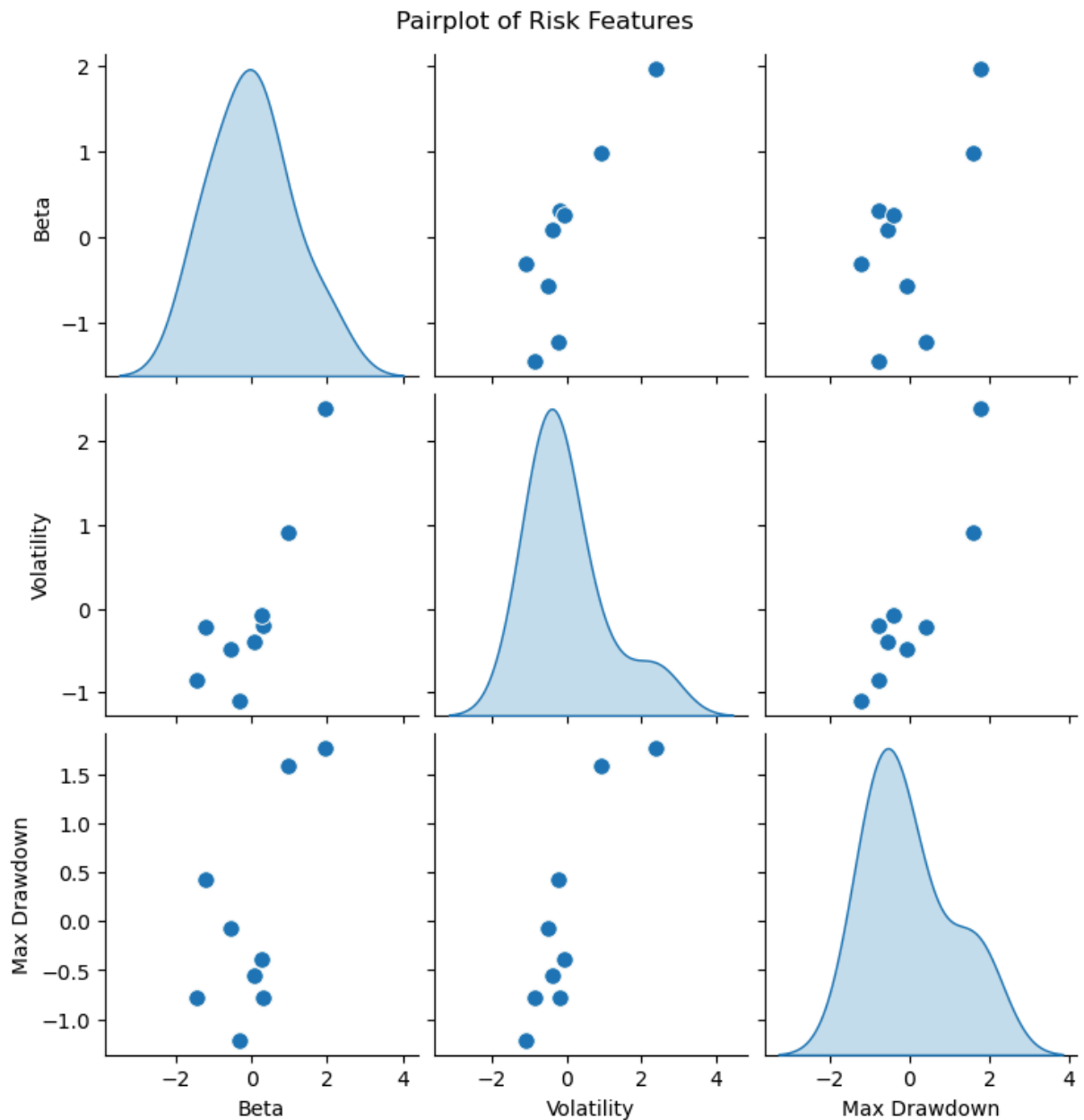


## 7.2 Pairplot and Scatterplot

The diagonal plots show the distribution of each feature. Beta is roughly normal. Volatility and Max Drawdown are skewed to the right which explained the outliers cause by TLSA and META shown in boxplots.

The scatter plots shows the relationships between pairs of features. Beta, Volatility, and Max Drawdown all show a positive linear relationship.

```
In [138... # Pairplot
sns.pairplot(risk_df_scaled, diag_kind="kde", plot_kws={"s": 70})
plt.suptitle("Pairplot of Risk Features", y=1.02)
plt.show()
```



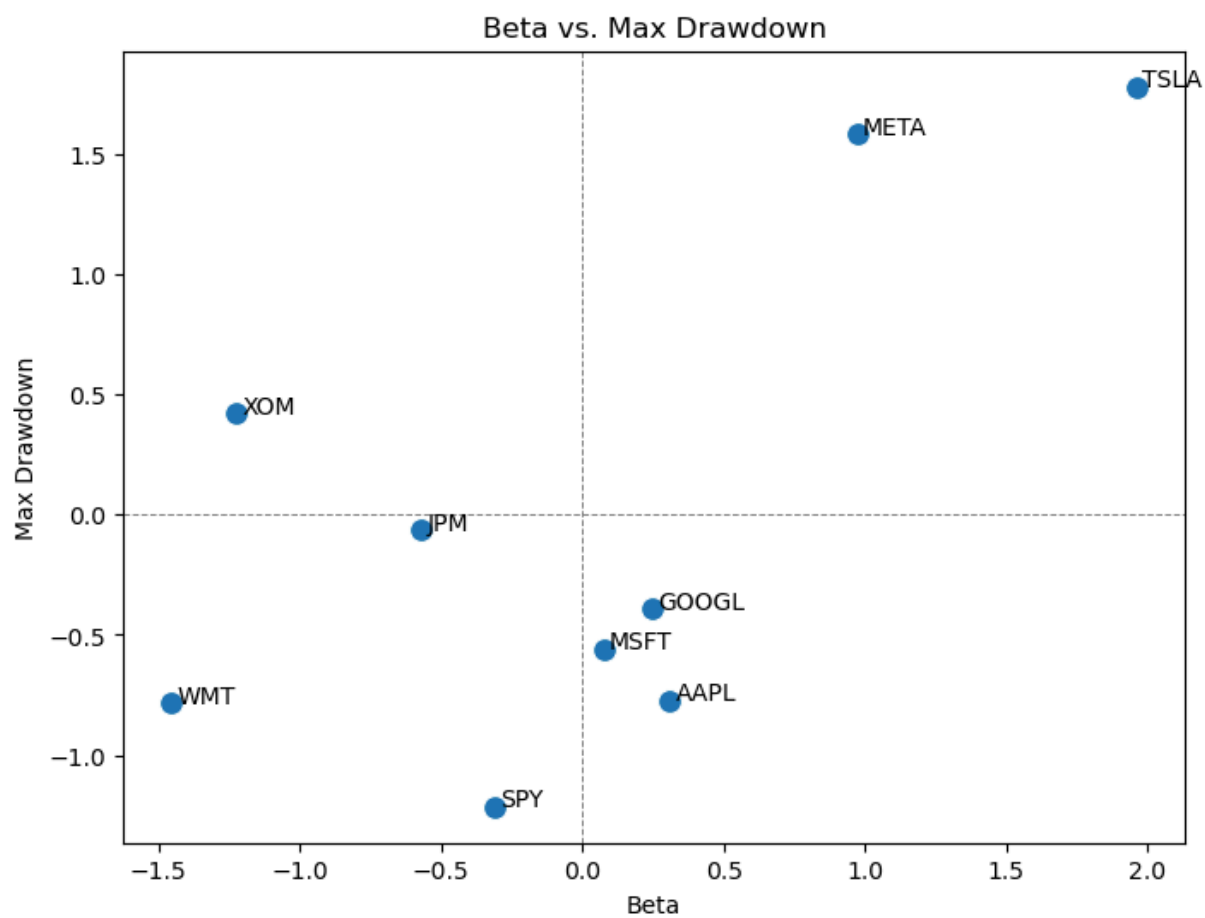
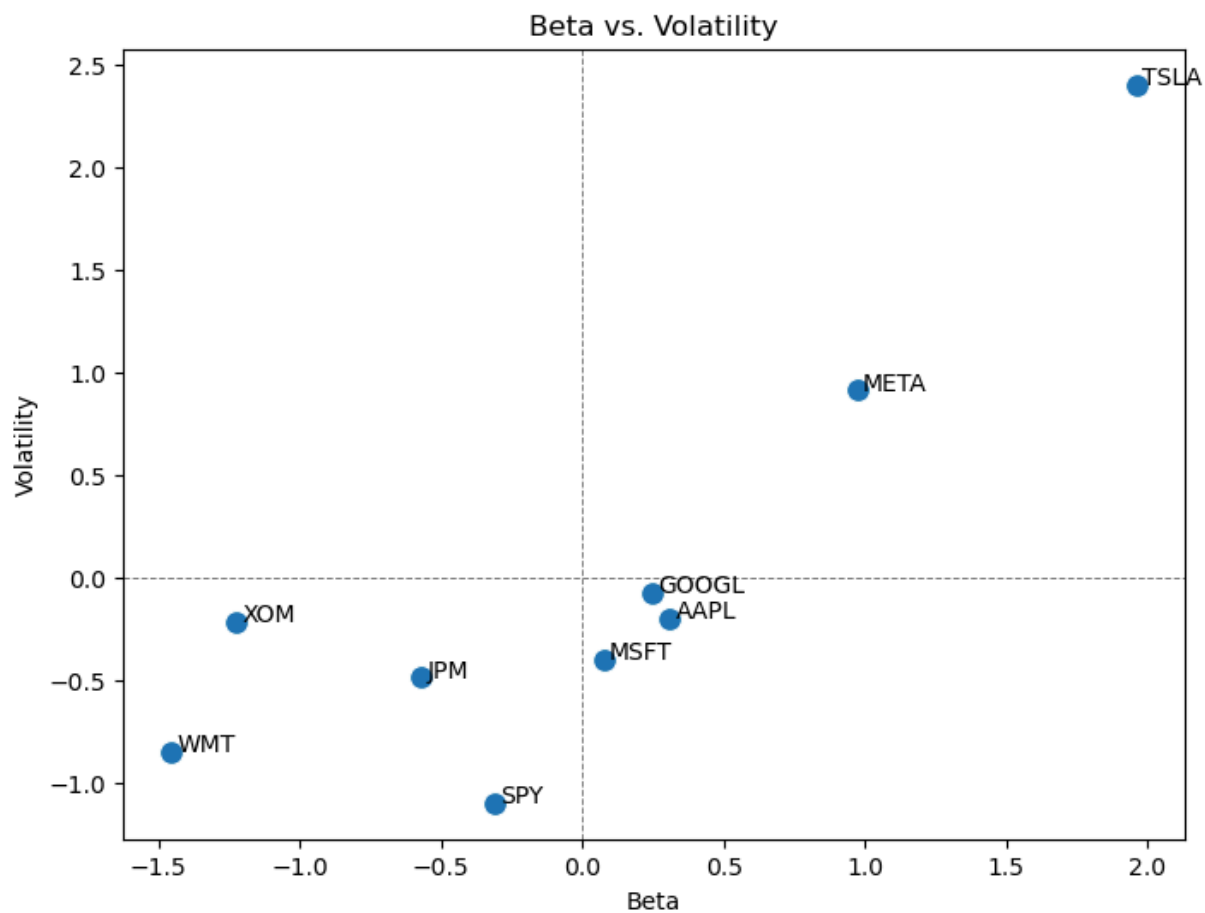
In [139...

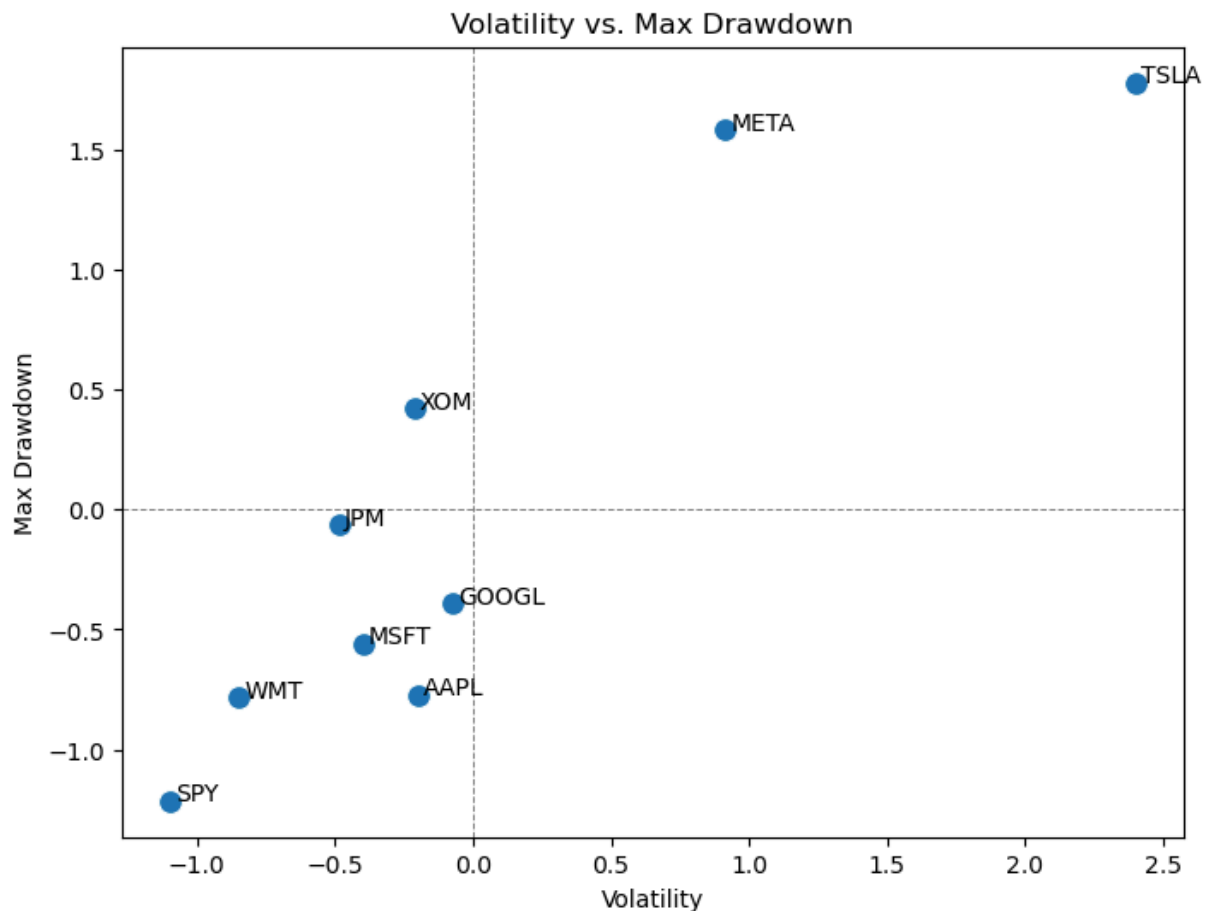
```
# Scatterplots
pairs = [("Beta", "Volatility"), ("Beta", "Max Drawdown"), ("Volatility", "Max Drawdown")]

for x, y in pairs:
    plt.figure(figsize=(8, 6))
    sns.scatterplot(data=risk_df_scaled, x=x, y=y, s=100)

    # annotate stock labels
    for stock in risk_df_scaled.index:
        plt.text(risk_df_scaled.loc[stock, x]+0.02, risk_df_scaled.loc[stock, y], s=10)

    plt.title(f"{x} vs. {y}")
    plt.xlabel(x)
    plt.ylabel(y)
    plt.axhline(0, color="gray", linestyle="--", linewidth=0.7)
    plt.axvline(0, color="gray", linestyle="--", linewidth=0.7)
    plt.show()
```



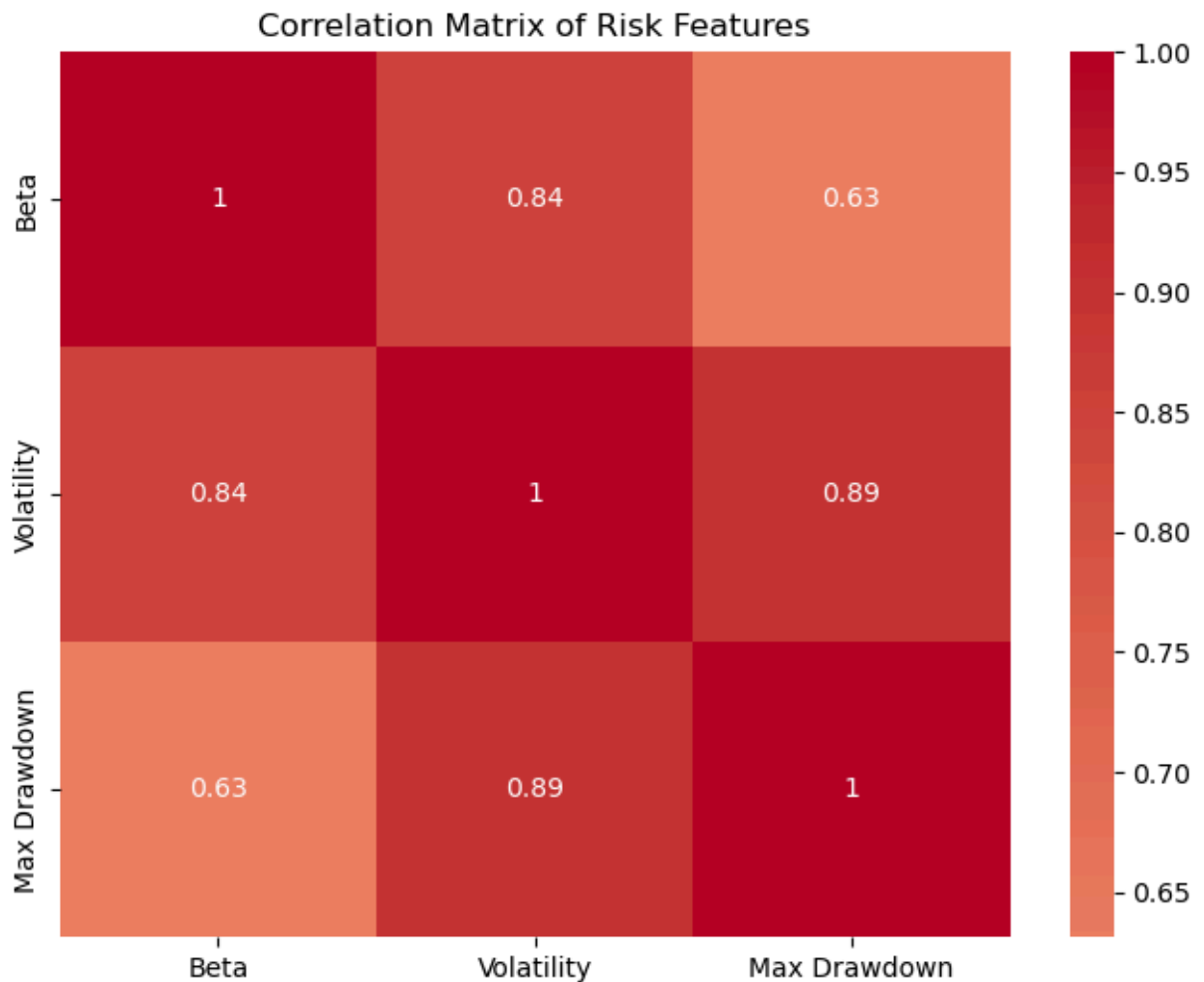


## 7.3 Correlation heatmap

The heatmap, pairplot, and scatterplot all show a strong positive correlation among the three risk features:

- Volatility and Max Drawdown have the strongest relationship. A stock's maximum drawdown tends to increase with its volatility.
- Beta and Volatility also have a strong relationship. High-volatility stocks are expected to have a higher beta.
- Beta and Max Drawdown have the weakest correlation.

```
In [140... # Correlation heatmap
plt.figure(figsize=(8, 6))
corr = risk_df_scaled.corr()
sns.heatmap(corr, annot=True, cmap="coolwarm", center=0)
plt.title("Correlation Matrix of Risk Features")
plt.show()
```



## 7.4 Post EDA data processing - Principal Component Analysis (PCA)

The EDA confirmed that the selected features are not independent. Their high correlation will cause the unsupervised learning cluster to give them more weight, skewing the clusters. Principal Component Analysis (PCA) can correct the dependency issue by combining the features into new, uncorrelated ones. This approach is suppose to produce a more accurate clustering result.

```
In [141... # Principal Component Analysis
# use n_components=3 for 3 features
pca = PCA(n_components=3)

# Fit the PCA model to the normalized data
principal_components = pca.fit_transform(risk_df_scaled)

# Create a new DataFrame with the principal components
pca_df = pd.DataFrame(data = principal_components, columns = ['PC1', 'PC2', 'PC3'],

print("Explained Variance Ratio:")
print(pca.explained_variance_ratio_)
```

```
print("\nPrincipal Components Dataframe:")
print(pca_df)
```

Explained Variance Ratio:

```
[0.86174555 0.12367929 0.01457516]
```

Principal Components Dataframe:

	PC1	PC2	PC3
AAPL	-0.387376	0.755482	0.110087
MSFT	-0.518628	0.450794	-0.066348
TSLA	3.558718	0.140403	0.288992
GOOGL	-0.130835	0.446530	0.043829
JPM	-0.648667	-0.355784	-0.139417
XOM	-0.568823	-1.183178	0.078094
WMT	-1.765195	-0.517076	0.256833
META	1.991097	-0.377443	-0.421015
SPY	-1.530291	0.640272	-0.151055

## 8. Modelling

### 8.1 K-Means Model

K-Means was chosen for this project due to its simplicity and scalability with large datasets. It also produces easily interpretable clusters, which aligns with the project's goal of defining Low, Medium, and High-risk categories.

### 8.2 The Elbow Method for `n_clusters`

The elbow method was used to find the optimal number of clusters. As the plot shows, the "elbow" occurs at two clusters, indicating it's the most efficient grouping.

However, for this project, a more granular risk classification was needed. We chose three clusters to create distinct Low, Medium, and High-risk groups, which aligned with the project's primary goal.

In [142...

```
warnings.filterwarnings("ignore", category=UserWarning)

# Implement the Elbow Method
# A list to hold the sum of squared distances
sum_sq_distance = []

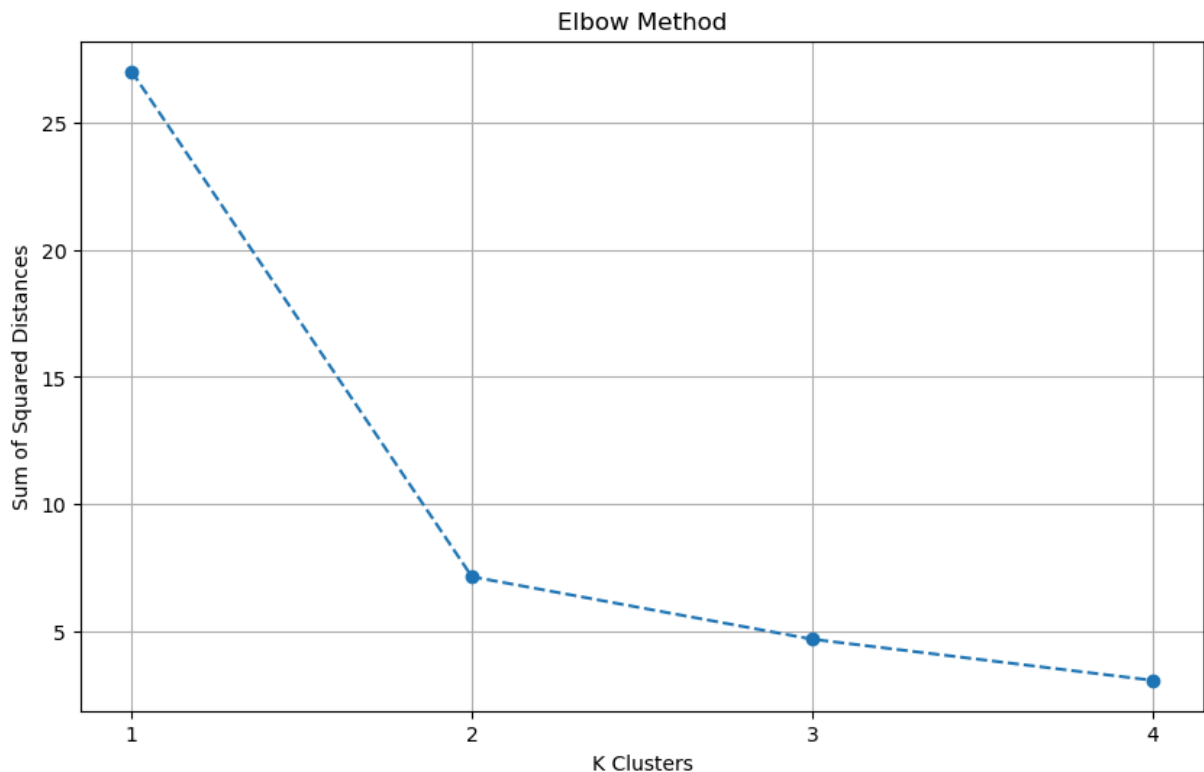
# Range of cluster numbers to test (e.g., from 1 to 10)
k_range = range(1, 5)

# Loop through the range of k values
for k in k_range:
    # Initialize KMeans with the current k
    # random_state=42 ensures reproducibility of the results
    kmeans = KMeans(n_clusters=k, random_state=42)
```

```
# Fit the model to the scaled risk dataframe
kmeans.fit(risk_df_scaled)

# Append to inertia List
sum_sq_distance.append(kmeans.inertia_)

# 3. Plot the Elbow Curve
plt.figure(figsize=(10, 6))
plt.plot(k_range, sum_sq_distance, marker='o', linestyle='--')
plt.title('Elbow Method')
plt.xlabel('K Clusters')
plt.ylabel('Sum of Squared Distances')
plt.xticks(k_range)
plt.grid(True)
plt.show()
```



## 8.3 Final K-Means Clustering

The final K-Means model was configured with the following parameters:

- `n_clusters=3` : Chosen to represent the three risk categories.
- `random_state=42` : This ensures reproducibility by setting a fixed seed for centroid initialization.
- `n_init=10` : The algorithm was run 10 times with different centroid initializations to reduce the risk of a poor clustering outcome.

To validate the model, K-Means was applied to both the normalized data and the PCA-transformed data to assess for any significant differences in the results.



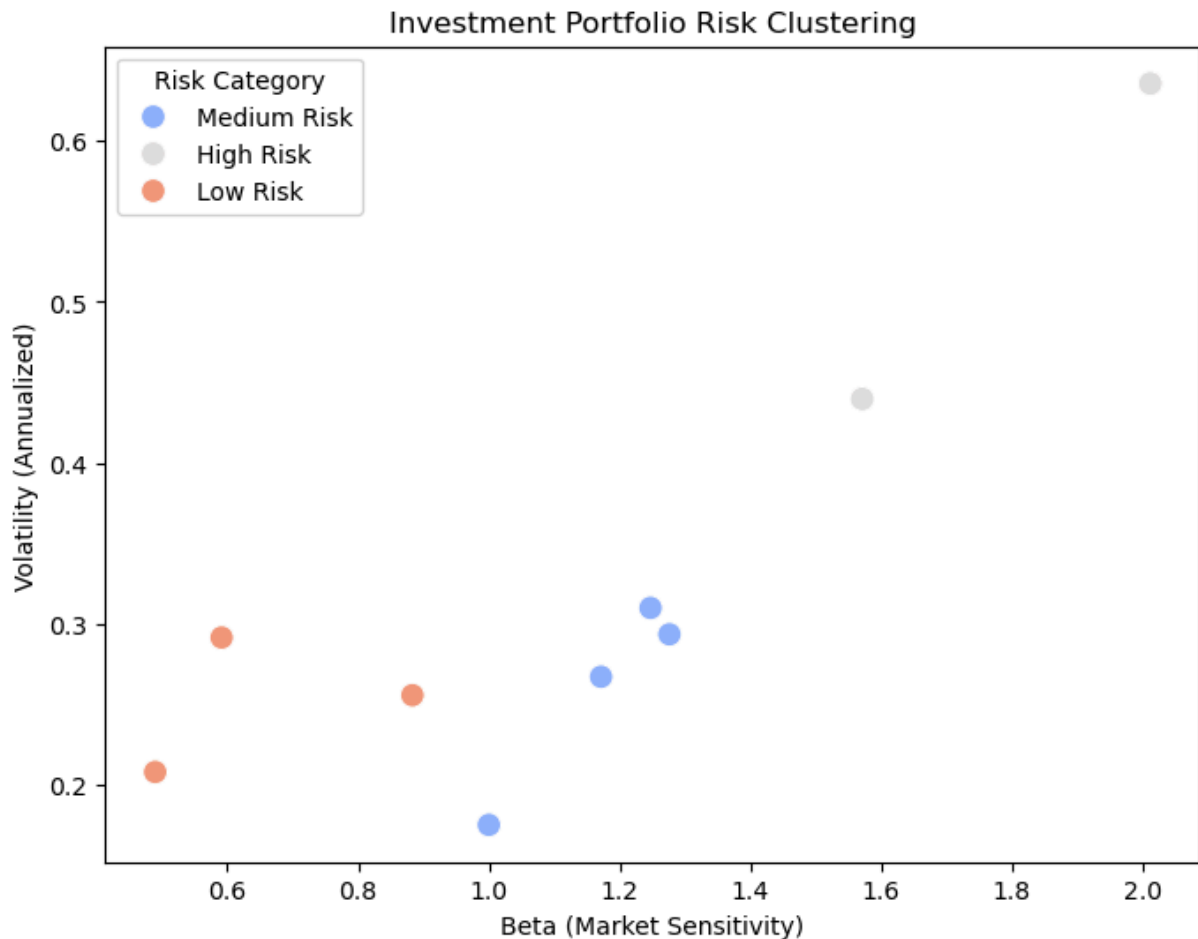
In [143...

```
# Apply K-Means on scaled data
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
risk_df["Risk Cluster"] = kmeans.fit_predict(risk_df_scaled)
risk_df["Risk Cluster PCA"] = kmeans.fit_predict(pca_df)

# Map clusters to risk categories
risk_labels = {0: "Low Risk", 2: "Medium Risk", 1: "High Risk"}
#risk_df["Risk Category"] = risk_df["Risk Cluster"].map(risk_labels)
risk_df["Risk Category"] = risk_df["Risk Cluster"].map(risk_labels)

# Display clustered risk table
print("#####")
print("##### K-means Risk Assessment with and without PCA #####")
print("#####")
print(risk_df[["Risk Cluster", "Risk Cluster PCA", "Risk Category"]])
print("#####")
# Visualization: Cluster Distribution
plt.figure(figsize=(8, 6))
sns.scatterplot(x=risk_df["Beta"], y=risk_df["Volatility"], hue=risk_df["Risk Category"])
plt.xlabel("Beta (Market Sensitivity)")
plt.ylabel("Volatility (Annualized)")
plt.title("Investment Portfolio Risk Clustering")
plt.legend(title="Risk Category")
plt.show()
```

```
#####
##### K-means Risk Assessment with and without PCA #####
#####
Risk Cluster Risk Cluster PCA Risk Category
AAPL          2              2 Medium Risk
MSFT          2              2 Medium Risk
TSLA          1              1 High Risk
GOOGL         2              2 Medium Risk
JPM           0              0 Low Risk
XOM           0              0 Low Risk
WMT           0              0 Low Risk
META          1              1 High Risk
SPY           2              2 Medium Risk
#####
```



## 8.4 Hierarchical Clustering Model

To see if another clustering model would produce a different result than K-Means, a Hierarchical Clustering Model was chosen. It uses the `AgglomerativeClustering` function from sklearn. The parameters for the model are:

- **metric:** The distance metric uses `euclidean`.
- **linkage:** The method for calculating the distance between clusters is `ward` because it minimizes the variance within each cluster, forces the algorithm to group similar data points together.
- **n\_clusters:** Hierarchical Clustering models do not require specifying the number of clusters. However, the initial result of this configuration produced only two clusters.

In [144...

```
# Hierarchical Clustering Model
agg_cluster = AgglomerativeClustering(
    metric="euclidean",
    linkage="ward"
)

# Create a copy of the DataFrame before adding new columns
hierarchical_df_1 = risk_df_scaled.copy()
```

```

hierarchical_labels = agg_cluster.fit_predict(risk_df_scaled)

# Add the cluster labels to DataFrame
hierarchical_df_1["Hierarchical Label"] = hierarchical_labels

# Risk mapping based on analysis
risk_labels = {1: "High Risk", 2: "Medium Risk", 0: "Low Risk"}

# Map the numerical cluster labels to the risk levels
hierarchical_df_1["Risk Level"] = hierarchical_df_1["Hierarchical Label"].map(risk_labels)

print(hierarchical_df_1)

```

	Beta	Volatility	Max Drawdown	Hierarchical Label	Risk Level
AAPL	0.310311	-0.197959	-0.772950	0	Low Risk
MSFT	0.075435	-0.398340	-0.558773	0	Low Risk
TSLA	1.961124	2.400308	1.777701	1	High Risk
GOOGL	0.245398	-0.073583	-0.390888	0	Low Risk
JPM	-0.572193	-0.485203	-0.062940	0	Low Risk
XOM	-1.227125	-0.213245	0.422210	0	Low Risk
WMT	-1.455297	-0.847456	-0.783054	0	Low Risk
META	0.971996	0.912514	1.583265	1	High Risk
SPY	-0.309650	-1.097037	-1.214571	0	Low Risk

To find the optimal number of clusters, a dendrogram can give a visual aid in deciding the `n_clusters`. When a horizontal line across the dendrogram at a distance of about 2.0, it will intersect with three vertical lines for the following groups:

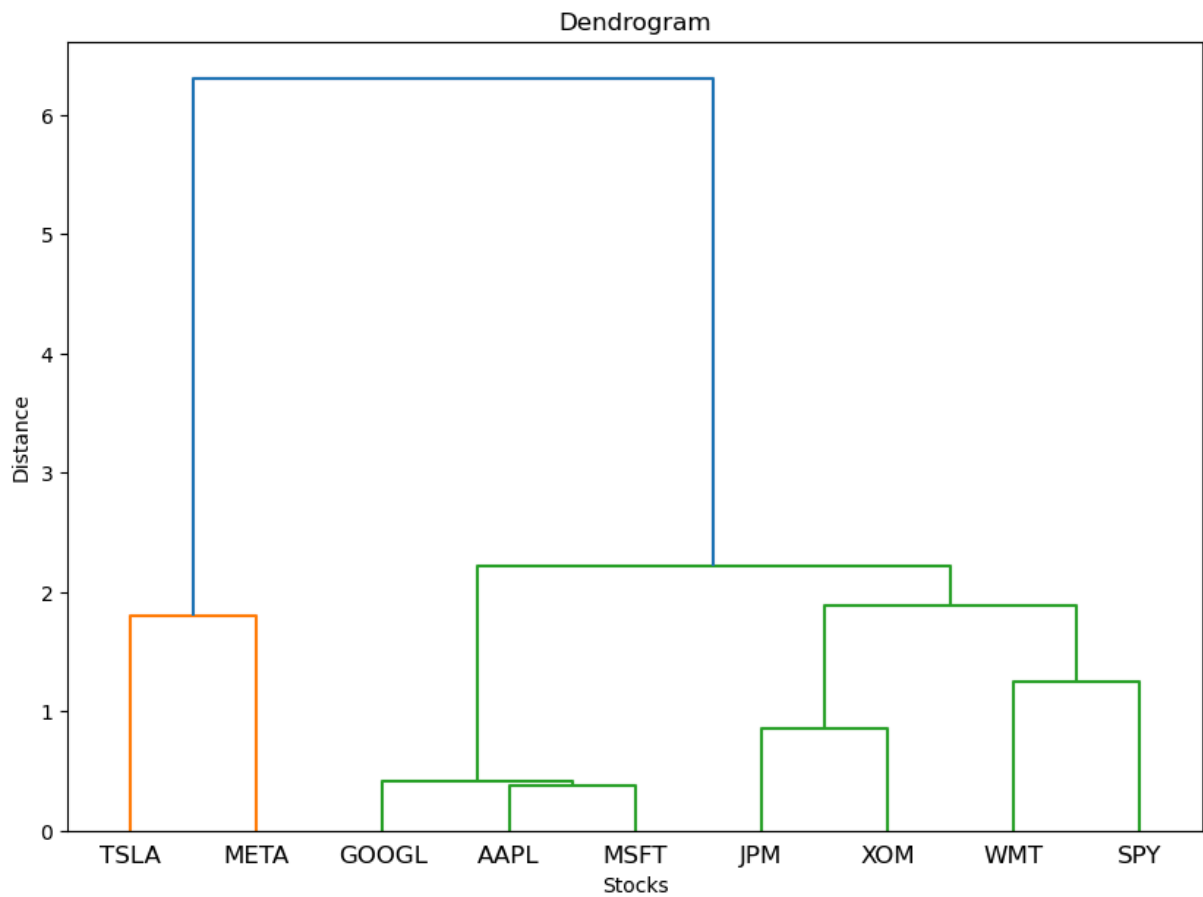
- Cluster 1: TSLA and META
- Cluster 2: GOOGL, AAPL, and MSFT
- Cluster 3: JPM, XOM, WMT, and SPY

Based on the dendrogram, `n_clusters = 3`.

```

In [145... # draw the dendrogram to find the optimal cluster grouping
linked = linkage(risk_df_scaled, method="ward")
plt.figure(figsize=(10, 7))
dendrogram(linked, labels=risk_df_scaled.index)
plt.title("Dendrogram")
plt.xlabel("Stocks")
plt.ylabel("Distance")
plt.show()

```



In [146...

```
# Hierarchical Clustering Model
agg_cluster = AgglomerativeClustering(
    n_clusters=3,
    metric="euclidean",
    linkage="ward"
)

hierarchical_labels = agg_cluster.fit_predict(risk_df_scaled)

# Add the cluster labels to DataFrame
risk_df_scaled["Hierarchical Label"] = hierarchical_labels

# Risk mapping based on analysis
risk_labels = {1: "High Risk", 2: "Medium Risk", 0: "Low Risk"}

# Map the numerical cluster labels to the risk levels
risk_df_scaled["Risk Level"] = risk_df_scaled["Hierarchical Label"].map(risk_labels)

print(risk_df_scaled)
```

	Beta	Volatility	Max Drawdown	Hierarchical Label	Risk Level
AAPL	0.310311	-0.197959	-0.772950	2	Medium Risk
MSFT	0.075435	-0.398340	-0.558773	2	Medium Risk
TSLA	1.961124	2.400308	1.777701	1	High Risk
GOOGL	0.245398	-0.073583	-0.390888	2	Medium Risk
JPM	-0.572193	-0.485203	-0.062940	0	Low Risk
XOM	-1.227125	-0.213245	0.422210	0	Low Risk
WMT	-1.455297	-0.847456	-0.783054	0	Low Risk
META	0.971996	0.912514	1.583265	1	High Risk
SPY	-0.309650	-1.097037	-1.214571	0	Low Risk

## 9. Result and Analysis

### 9.1 Validation Methodology

To validate the K-Means clustering risk assessment, we could compare the result from K-Means clustering of the 5-year dataset with the Morningstar risk rating (Morningstar, Inc., n.d.). Morningstar provides an independent risk measurement using Uncertainty Rating in five categories: Low, Medium, High, Very High, and Extreme. By comparing it to the K-Means clusters (Low, Medium, High), we could determine the accuracy of the K-Means clustering result. This method is the simplest and easily interpretable validation.

### 9.2 Evaluation Metrics

Compare K-Means risk categories (High, Medium, Low Risk) with Morningstar Risk Score:

Morningstar Risk Score	K-Means Risk Category
High, Very High, and Extreme	High Risk
Medium	Medium Risk
Low	Low Risk

Other results are consider as mismatches and require further analyze for the cause.

### 9.3 Evaluation Analysis

Ticker	Morningstar	K-Means (5-Year Dataset)
AAPL	Medium	Medium
GOOGL	Medium	Medium
JPM	NA	Medium
META	High	High
MSFT	Medium	Medium
TSLA	Very High	High

Ticker	Morningstar	K-Means (5-Year Dataset)
WMT	Medium	Low
XOM	High	Medium

The K-Means risk categories were consistent with the Morningstar Risk Score, with a few exceptions:

- JPM: Morningstar did not have a risk score for JPM.
- WMT: Morningstar rated WMT as medium risk due to the threat from e-commerce companies like Amazon to Walmart's traditional retail model.
- XOM: Morningstar rated XOM as high risk because of its exposure to global oil prices. The increasing adoption of electric vehicles and climate change policies, including carbon taxes, could dramatically reduce oil demand and prices.

## 9.4 Normalized data vs PCA data

Both the normalized data and the PCA data gave the same K-means results. This is an unexpected outcome. PCA should change the data's geometry to affect clustering.

However, since the clusters were already distinctively far apart with only nine data points that PCA's transformation didn't change the final groupings.

Even with the same result, PCA is still the correct approach for dealing with correlated features. In larger datasets like 1000 stocks, it would likely produce a significantly different result.

## 9.4 K-Means vs Hierarchical Clustering Model

Both models can produce the same groupings. In this project, the data has a very clear, natural separation. Even though the models work differently, they both find the same best solution. Hierarchical clustering relies on building a tree of relationships, while K-Means uses an iterative process to find cluster centers.

The fact that Hierarchical Clustering returned two groups (TSLA and META in one and the rest in the other) is interesting. In the context of 5-years normalized dataset, this suggests that `AgglomerativeClustering` algorithm sees those two stocks as outliers as "High Risk" group, while all the others are seen as "Low Risk".

# 10. Discussion and Conclusion

## 10.1 Challenges

During implementation, several challenges emerged that required adjustments:

- **Limited Data Retrieval:** A primary challenge was retrieving historical stock data due to API rate limits from data providers. To address this, the data were split into smaller batches, and delays were implemented between downloads to avoid exceeding request limits.
- **K-Means Fine-Tuning:** K-Means clustering proved highly sensitive to feature selection and data scaling. It was therefore important to select the most appropriate risk features (Beta, Volatility, Max Drawdown) and determine the correct data preprocessing steps, such as using `pct_change()`.
- **Optimal Number of Clusters:** Determining the optimal number of risk groups presented a significant challenge. Using two clusters (High vs. Low) would lead to oversimplification, while four or more could have resulted in over-segmentation, complicating interpretation. The Elbow method was used as a simple way to identify a better cluster number, but it still required human interpretation and judgment to decide on the final number of clusters.
- **Alternative Approach:** K-Means requires specifying the number of clusters, which can be challenging and increase the model's sensitivity. Hierarchical Clustering is an alternative that does not require a predefined number of clusters, as it treats each stock as its own cluster and iteratively merges the closest ones. However, Hierarchical Clustering can be computationally expensive for large datasets compared to K-Means.

## 10.2 Key Takeaway

Lessons learned from implementing K-Means clustering for stock risk assessment include:

- **Percentage Changes:** Preprocessing with percentage changes instead of raw price data makes stocks comparable regardless of price level. Furthermore, metrics like Beta, Volatility, and Max Drawdown rely on return distributions, not price values; using raw prices would distort these calculations.
- **Standardization:** Even after converting prices to percentage changes, risk metrics had different numerical ranges. Standardizing them with `StandardScaler()` helped prevent features with larger numerical values from dominating the clustering.
- **Cluster Interpretability:** While K-Means effectively grouped stocks into risk categories, the assignments did not always align perfectly with external risk scores (Morningstar). The discrepancies for stocks like WMT and XOM suggest that Morningstar's ratings are likely based on both qualitative and quantitative data, while this model relied solely on quantitative historical data. For improved accuracy, the K-Means model would need additional quantitative metrics, such as a Risk-Adjusted Performance Score.
- **Definition of Risk:** The primary risk metric used in this project was volatility. However, this differs from the methodology used by Morningstar, whose "Risk Ratings Explained"

article states that its risk measurement includes an "uncertainty" rating. The article also notes, "market risk is about more than volatility... [it] includes the possibility of depressed markets and extreme events" (Morningstar, Inc., n.d.).

Another analytics platform, Portfolios Lab, defines risk using "Risk-Adjusted Performance," which incorporates several ratios such as Sharpe, Sortino, and Calmar. The Sharpe ratio, for example, helps investors understand the return they receive for the level of risk they take. It is also important to note that returns are relative to volatility over different time periods.

## 10.3 Conclusion

This project successfully applied K-Means clustering to categorize stocks into three risk groups. However, its effectiveness is limited by two main factors.

First, K-Means assumes a fixed number of clusters, which may not be ideal. Future research could explore alternative methods like Hierarchical Clustering, which adaptively determines cluster structure.

Second, this project's definition of risk was limited to historical volatility and drawdowns. As external analyses from Morningstar and Portfolio Lab suggest, risk is a much broader concept that includes market uncertainty and is best measured by risk-adjusted performance using metrics like the Sharpe or Sortino ratios.

To provide a more comprehensive risk assessment, future research should incorporate these findings by including additional quantitative metrics. Further validation through historical stress testing and expanding the dataset to include economic indicators and sector-specific risk patterns could also enhance the model's accuracy and adaptability.

## REFERENCES

Datrics AI. (n.d.). K-Means Clustering in Banking: Applications & Examples. Retrieved from <https://www.datrics.ai/articles/how-k-means-clustering-is-transforming-the-banking-sector>

Stock classification using k-means clustering (n.d.). Medium. Retrieved from <https://medium.com/@facujallia/stock-classification-using-k-means-clustering-8441f75363de>

Li, B., Tao, R., Li, M., & Sharma, K. (2022). Identification of Enterprise Financial Risk Based on Clustering Algorithm. *Computational Intelligence and Neuroscience*, 2022, Article ID 1086944. <https://doi.org/10.1155/2022/1086945>

Zhu, Y., & Liu, Q. (2021). Early Warning of Financial Risk Based on K-Means Clustering Algorithm. *ResearchGate*. Retrieved from



[https://www.researchgate.net/publication/349851488\\_Early\\_Warning\\_of\\_Financial\\_Risk\\_Based\\_on\\_Means\\_Clustering\\_Algorithm](https://www.researchgate.net/publication/349851488_Early_Warning_of_Financial_Risk_Based_on_Means_Clustering_Algorithm)

Morningstar, Inc. (n.d.). Morningstar's risk ratings explained. Retrieved from <https://global.morningstar.com/en-gb/personal-finance/morningstar-s-risk-ratings-explained>

Portfolios Lab (n.d.). Example of AAPL Risk-Adjusted Performance. Retrieved from <https://portfolioslab.com/symbol/AAPL>

Tipranks (n.d.). Example of AAPL's Risk Factors. Retrieved from <https://www.tipranks.com/stocks/aapl/risk-factors>

Infront. (n.d.). Infront Advanced Analytics Solution for Wealth Managers, Analysts & Brokers. Retrieved from <https://www.infront.co/global/en/solutions/modules/data-display-and-analytics/analytics.html>