Principal Component Analysis Report

Introduction

This report presents the findings from a Principal Component Analysis (PCA) conducted on a diverse financial dataset, including major cryptocurrencies, global stock market indices, and commodities. The primary goal of this analysis was to reduce the dimensionality of the dataset by identifying the underlying, uncorrelated factors that explain the majority of the data's variance

By converting many correlated variables into a smaller set of meaningful components, PCA allows for a more simplified and efficient approach to tasks like portfolio management, risk assessment, and predictive modelling. The analysis revealed that a few key components account for most of the market's movements, providing valuable insight into the major drivers of the data. The following sections detail the methodology, present the results of the PCA, and discuss the implications of the findings.

Methodology

The objective of this analysis was to identify the key underlying factors driving the performance of a diverse set of financial assets. The dataset was compiled from historical daily market data, with all information collected from the Bloomberg Terminal. The raw data consisted of the open, close, and volume prices for 13 distinct datasets, including major domestic and international indices, popular cryptocurrencies, and the stock of Hindustan Unilever.

The dataset spanned a period from January 1, 2023, to June 1, 2025. To ensure a standardized and comparable basis for analysis, the raw historical prices were transformed into daily percentage changes. This process was executed manually in a spreadsheet for all 13 datasets. This transformation was crucial as it normalizes the data, addressing the issue of differing price scales and trends across the various assets.

After calculating the percentage changes, all irrelevant columns were removed, leaving only the date and the newly created percentage change metric. The data was then sorted in ascending order by date to establish a proper time series. The percentage values, which initially included the '%' symbol, were converted into a floating-point format to prepare the dataset for numerical analysis.

Following the initial data cleaning performed in the spreadsheet software, the consolidated dataset was imported into a Python environment for further processing and analysis. The final dataset, now in a pandas Data Frame, contained 13 financial time-series representing daily percentage changes for various assets.

```
#Importing required libraries
import pandas as pd
import glob
import os

folder = "Datasets"
files                                  =
glob.glob(os.path.join(folder,
"*.csv"))
print(files)  # check what files are
picked up
```

```
#Reading all CSV files into a list of Data Frames and parsing the "Date" column as datetime to ensure correct date format
dfs = [pd.read_csv(f, parse_dates=["Date"], dayfirst=True) for f in files]


#Merging all DataFrames on the "Date" column using an outer join to ensure all dates are included
final = dfs[0]
for df in dfs[1:]:
	final = pd.merge(final, df, on="Date", how="outer")

final = final.sort_values("Date").reset_index(drop=True)
print(final.head())


final = final.fillna(0) #treat missing values as 0 / holidays

final.to_csv("final.csv", index=False) # save the final DataFrame
```

The empty values were present in the joined dataset because of timing differences and non-trading days across different markets. When you merge time-series data from various countries and asset classes, not all of them trade on the same days. For example:

- **Holidays:** A public holiday in India (affecting the BSE 500) might be a regular trading day in the US (affecting the S&P 500).
- **Time Zone Differences:** The trading hours of the Nikkei in Japan don't overlap with those of the S&P 500 in the US. This can cause discrepancies in the daily data points.

Consequently, when you align all these time-series on a single date column, a day that has a data point for one asset might not have a corresponding data point for another, resulting in empty or missing values.

Why was percentage Change chosen as the Target column for all datasets?

Principal Component Analysis (PCA) is better applied to percentage changes (returns) rather than raw prices due to stationarity, scale, and interpretability.

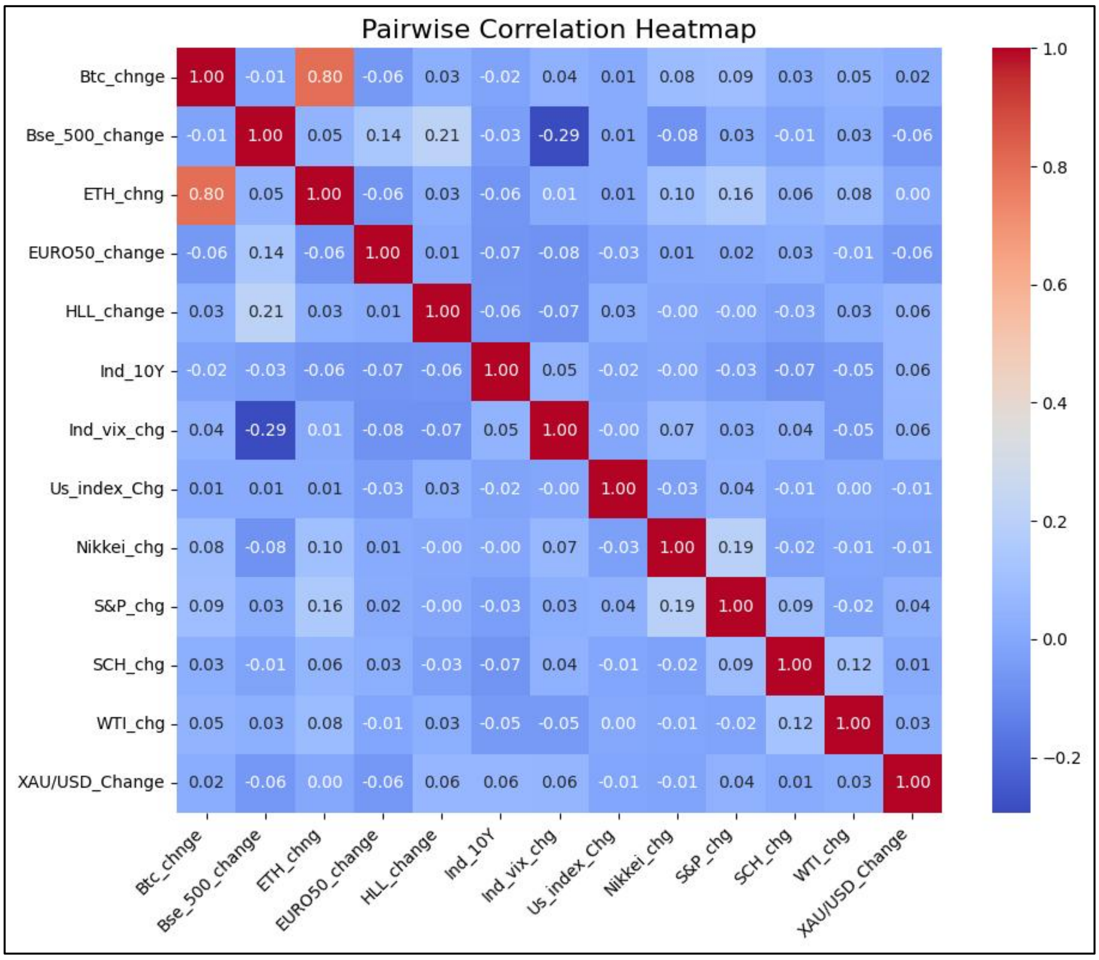
Stationarity: Raw stock prices are typically non-stationary, with trends and increasing variance over time. PCA on such data often captures only these dominant trends. Returns, being generally stationary, focus the analysis on short-term fluctuations and co-movements, which are more informative.

Scale: Raw prices vary widely across assets, causing high-priced stocks to dominate variance-based PCA. Using percentage changes standardizes all assets, allowing PCA to identify components that explain collective variance.

Interpretability: Components from returns are easier to interpret. PC1 often reflects overall market movement, while subsequent components capture sector specific or other underlying factors. Returns isolate meaningful co-movements rather than being dominated by absolute price levels.

```
# Correlation heatmap
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# drop Date and compute correlation
matrix as date is not required for
correlation
corr = df.corr()

plt.figure(figsize=(10, 8))
sns.heatmap(corr,annot=True,
# show numbers
            fmt=".2f",
# format numbers
            cmap="coolwarm",
# red/blue color scheme
            cbar=True,
            square=True)
#plotting
plt.title("Pairwise      Correlation
Heatmap", fontsize=16)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```



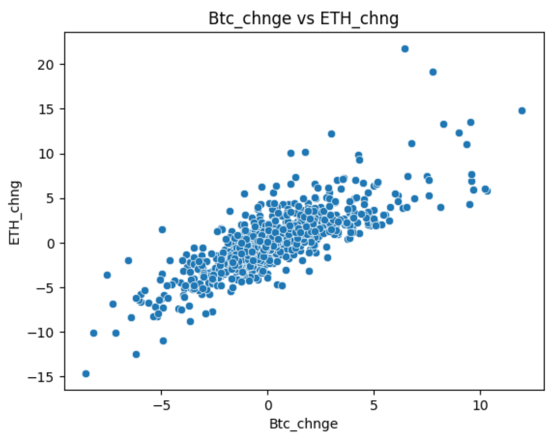
The pairwise correlation analysis reveals distinct patterns among the assets:

- **Cryptocurrencies (BTC and ETH):** Very strong positive correlation (~0.80), indicating they move closely together, as expected.
- **Euro50, US index, SCH):** Low correlations among each other (0.1–0.2 at most). Daily changes are not tightly coupled, even though long-term trends may align.
- **BSE vs. India VIX:** Moderate negative correlation (~–0.29), consistent with the common pattern

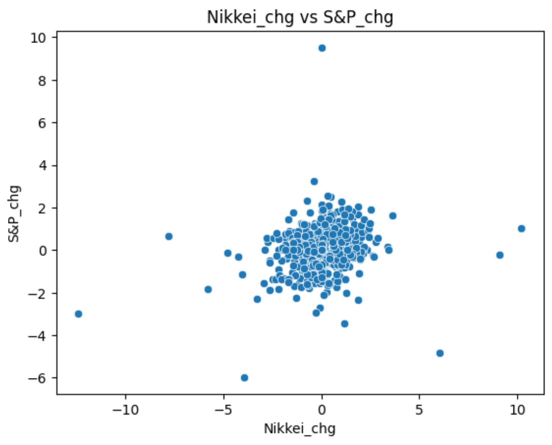
that rising volatility corresponds with falling equity prices.

- **Other assets (Gold, Oil, 10Y yields, HLL):** Near-zero correlations with all other assets, suggesting largely independent daily movements.
- **Diagonal elements:** All equal to 1.0, representing self-correlation.

```
## Instead of all variables, plot only the pairs that showed high correlation in the heatmap in a Scatter plot

import seaborn as sns

import matplotlib.pyplot as plt

# pick only a few strong relationships
pairs = [
    ("Btc_chnge", "ETH_chng"),
    ("Nikkei_chg", "S&P_chg"),
                  ("Bse_500_change", "Ind_vix_chg")
]

for x, y in pairs:
      sns.scatterplot(data=df, x=x, y=y)
    plt.title(f"{x} vs {y}")
    plt.show()
```

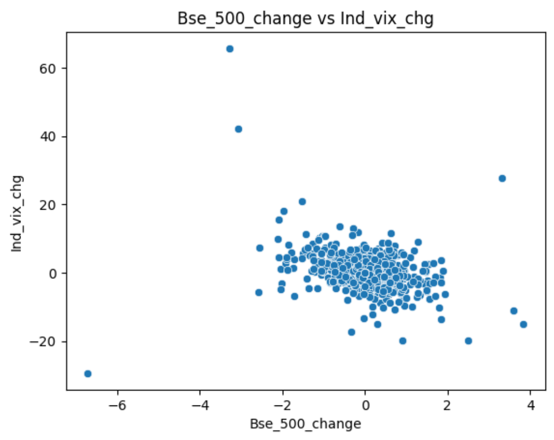


The plot shows a strong positive correlation between the daily percentage change in Bitcoin (Btc_chnge) and the daily percentage change in Ethereum (ETH_chnge). This means that when the price of Bitcoin goes up, the price of Ethereum also tends to go up



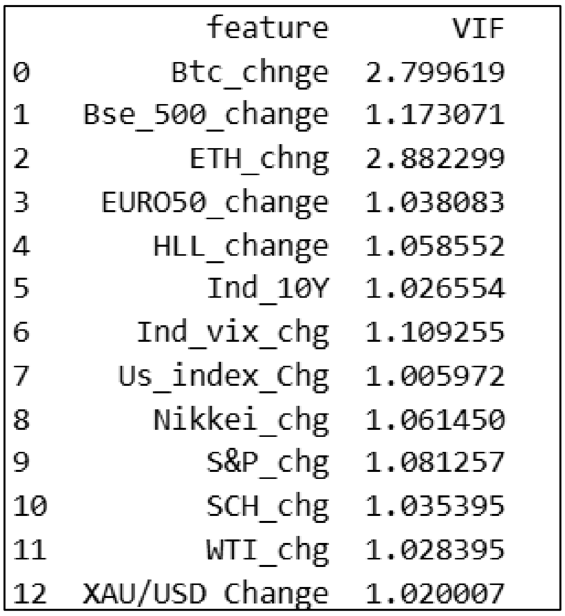
The plot shows a weak positive correlation between the daily percentage change in the Nikkei (Nikkei_chg) and the daily percentage change in the S&P 500 (S&P_chg).

The data points are scattered and do not follow a clear, tight linear pattern. While there is a slight tendency for both indices to move in the same direction, the relationship is not strong



The daily percentage changes of the BSE 500 index and India VIX exhibit a weak negative correlation. Generally, increases in BSE correspond to decreases in VIX and vice versa, reflecting the expected inverse relationship between equity performance and market volatility. However, the data points are widely scattered, particularly for larger changes, indicating that BSE movements are not strong predictors of VIX fluctuations. A few outliers highlight rare instances of extreme market behaviour.

```
# get the VIF values
from
statsmodels.stats.outliers_influenc
e import variance_inflation_factor
import pandas as pd

X = df.fillna(0)   # make sure no NaN
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"]                     =
[variance_inflation_factor(X.values
, i)
                          for  i  in
range(X.shape[1])]
print(vif_data)
```



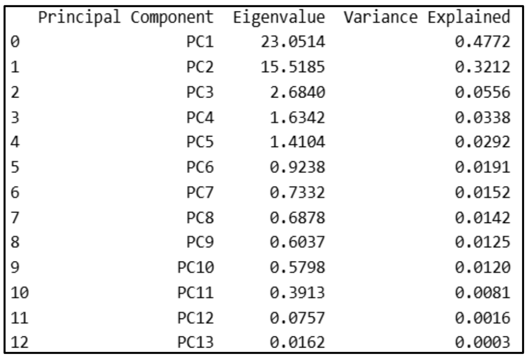
```
           feature       VIF
0        Btc_chnge  2.799619
1   Bse_500_change  1.173071
2         ETH_chng  2.882299
3    EURO50_change  1.038083
4       HLL_change  1.058552
5          Ind_10Y  1.026554
6      Ind_vix_chg  1.109255
7     Us_index_Chg  1.005972
8       Nikkei_chg  1.061450
9          S&P_chg  1.081257
10         SCH_chg  1.035395
11         WTI_chg  1.028395
12  XAU/USD Change  1.020007
```

Most features have VIF values near 1, indicating low correlation and independence. BTC and ETH changes have slightly higher VIFs (2.80 and 2.88), reflecting their positive correlation, but these values are well below the multicollinearity threshold of 5 and are not a concern.

Eigen values and Vectors

```
import pandas as pd
import numpy as np

# Compute covariance matrix

cov_matrix = np.cov(df.T)

#     Compute     eigenvalues     and
eigenvectors

eigenvalues,eigenvectors=
np.linalg.eig(cov_matrix)

# Sort eigenvalues and eigenvectors
in descending order

idx = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]
```

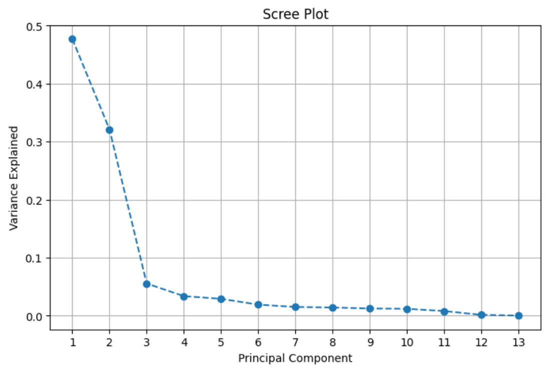
```
# Compute variance explained

variance_explained = eigenvalues /
eigenvalues.sum()

# Create a readable DataFrame

pc_table = pd.DataFrame({
           'Principal Component':
[f'PC{i+1}' for i in
range(len(eigenvalues))],
    'Eigenvalue': eigenvalues,
           'Variance Explained':
variance_explained
}

#show top few components
pc_table = pc_table.round(4)

print(pc_table)
```



```
   Principal Component  Eigenvalue  Variance Explained
0                  PC1     23.0514              0.4772
1                  PC2     15.5185              0.3212
2                  PC3      2.6840              0.0556
3                  PC4      1.6342              0.0338
4                  PC5      1.4104              0.0292
5                  PC6      0.9238              0.0191
6                  PC7      0.7332              0.0152
7                  PC8      0.6878              0.0142
8                  PC9      0.6037              0.0125
9                 PC10      0.5798              0.0120
10                PC11      0.3913              0.0081
11                PC12      0.0757              0.0016
12                PC13      0.0162              0.0003
```

Scree plot for optimal number of components

```
import matplotlib.pyplot as plt
from sklearn.decomposition import
PCA

# Apply PCA

pca = PCA()
pca.fit(df)

# Explained variance

explained_variance =
pca.explained_variance_ratio_

# Scree plot

plt.figure(figsize=(8,5))
plt.plot(range(1,
len(explained_variance)+1),
explained_variance, marker='o',
linestyle='--')
plt.title('Scree Plot')
plt.xlabel('Principal Component')
plt.ylabel('Variance Explained')
plt.xticks(range(1,
len(explained_variance)+1))
plt.grid(True)
plt.show()
```



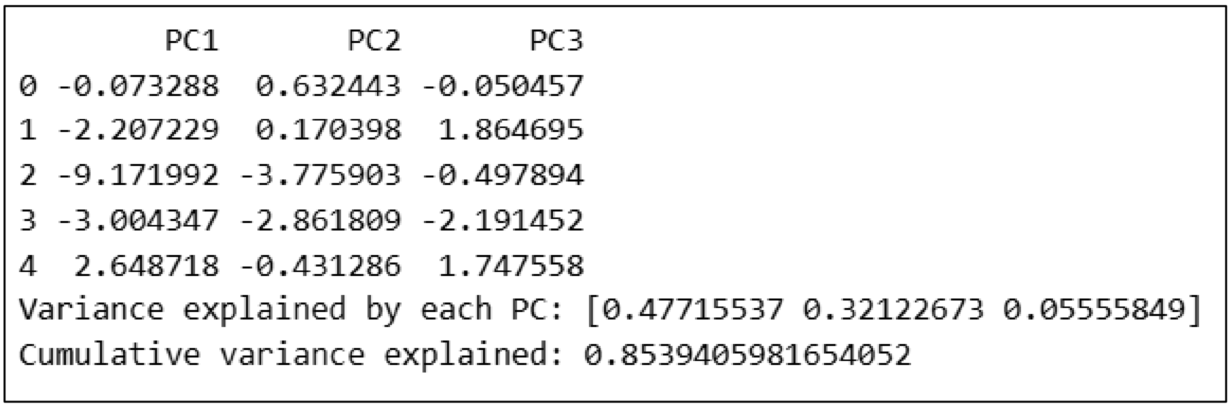
Principal Component 1 (PC1): This component explains the most variance, approximately 47%. It represents the primary underlying factor in the data.

Principal Component 2 (PC2): This component explains about 32% of the variance, a significant drop from PC1 but still substantial.

The "Elbow" Point: The most prominent elbow in this plot occurs after the third principal component. The variance explained drops sharply from PC2 to PC3 (from ~32% to ~5%) and then continues to decline gradually. Therefore, we decide on 3 number of Principal components

Perform Principal Component Analysis to reduce Variables

```python
# Perform PCA to reduce to 3 components
from sklearn.decomposition import PCA

# Number of components
n_components = 3

# Initialize PCA
pca = PCA(n_components=n_components)

# Fit PCA and transform data
X_pca = pca.fit_transform(df)

# Create a DataFrame with the 3 PCs
df_pca = pd.DataFrame(X_pca, columns=[f'PC{i+1}' for i in range(n_components)])

# Optional: check results
print(df_pca.head())

# Variance explained by these 3 PCs
print("Variance explained by each PC:", pca.explained_variance_ratio_)
print("Cumulative variance explained:", sum(pca.explained_variance_ratio_))
```



```
        PC1       PC2       PC3
0 -0.073288  0.632443 -0.050457
1 -2.207229  0.170398  1.864695
2 -9.171992 -3.775903 -0.497894
3 -3.004347 -2.861809 -2.191452
4  2.648718 -0.431286  1.747558
Variance explained by each PC: [0.47715537 0.32122673 0.05555849]
Cumulative variance explained: 0.8539405981654052
```

Interpretation

Principal Component Analysis (PCA) was performed on the dataset comprising 13 variables to reduce dimensionality while retaining most of the variance. The analysis identified three principal components (PCs) that together account for approximately 85% of the total variance.

PC1 captures the largest portion of variance (47.7%), representing the dominant pattern across the original variables. PC2, orthogonal to PC1, accounts for 32.1% of the variance, reflecting secondary variation not explained by the first component. PC3 contributes an additional 5.6% of the variance, capturing minor but meaningful variation. The remaining components each contribute less than 4%, indicating that they mainly represent residual noise rather than significant structural information.

By projecting the data onto these three principal components, the dataset is effectively reduced from 13 dimensions to 3, retaining most of its inherent information. This dimensionality reduction simplifies subsequent analyses and visualizations, while preserving the key patterns and relationships present in the original data.