

# SDS 496 Final Project Report

Aidan Kardan

December 1st, 2024

## Introduction to Dataset and Objectives

**US Stock Market Data:** Includes 11 datasets from Kaggle with AMZN stock data and various predictors, spanning fundamental, technical and raw historical metrics.

**Predictors:** Financial ratios, valuation metrics, technical indicators (e.g., RSI, MACD), and historical OHLCV data.

### Analysis Goals:

**Regression Task:** Predict AMZN stock prices, specifically, the close price, using the stock market data.

**Classification Task:** Predict stock movement (up/down) based on current information.

## Data Extraction and Cleaning

After extracting the 11 datasets, and merging them into one dataframe, there were some issues that arose.

### Data Repetition

#### Rationale for Handling Date Repetition

When combining data from multiple sources, each source contributed unique columns, resulting in multiple rows for the same Date. For any given Date, some columns were populated in one source but not in others, leading to overlapping but incomplete information.

#### Filling Missing Values:

Within all rows for the same date: Forward-fill was used to propagate the last valid value downward. Backward-fill was used to propagate the first valid value upward, addressing gaps at the start of the group.

#### Selecting the First Consolidated Row:

After filling missing values, the first row from each group was selected as the representative entry for that Date.

### **This process ensured:**

Each Date had a single row in the final dataset. All available information across sources was preserved. Missing values were handled methodically, minimizing data loss while maintaining data integrity.

## **Forward-Fill Financial Ratios**

Forward-filling financial ratios is standard in finance, as these metrics are typically updated quarterly after earnings reports. This approach preserves data integrity by propagating the most recent known values, reflecting real-world decision-making where investors rely on current and past information until new data becomes available.

Financial Ratios are also easy to spot in financial data because of their low frequency of occurrence!

## **Missing Information**

Rows missing open, high, low, close, or volume data indicate no trading activity occurred on those days. These rows can be safely removed from the dataset as they do not contribute meaningful information for analysis.

## **Price Column Removal**

The Price column, reflecting stock values at earnings reports, was removed to avoid biases from forward-filling and redundancy with existing daily metrics (Open, High, Low).

## **Target Variable for Analysis**

Using Close price as the target ensures clarity and relevance in our regression and classification analyses.

## **Tailored Analysis to Align with TA Window**

Technical analysis indicators are available from January 2010 to July 2022. Therefore, the analysis is specifically tailored to this time frame to ensure consistency and accuracy.

## **Miscellaneous NA**

Missing macroeconomic data is rare, occurring only on days when updates were not published. Forward-filling addresses these gaps, as these metrics change gradually, and the most recent value remains a reasonable approximation until new data is available. This reflects real-world reliance on the latest information.

At this point, the training and test data are ready for statistical analysis (no longer have any NA values).

## Time Series Data Issues

When dealing with time series data, standard training-test splits with randomization, and Cross-Validation or Bootstrap techniques cannot be used for analysis. To fix this, we will utilize a time-series split/time-series CV, which preserves the integrity of the chronological order of observations by sequentially splitting the data.

This method ensures that future data points are never used to predict past ones, maintaining the temporal structure critical for accurate financial modeling.

## Statistical Analysis

### Exploratory Analysis

In this analysis, I explored a wide range of statistical and machine learning models to address the regression and classification problems. For regression, I utilized Linear Regression, Ridge Regression, Lasso Regression, PCA Regression, Random Forests, Gradient Boosting, Linear SVM, Non-linear SVM, KMeans Clustering, Hierarchical Clustering, and Model-Based (GMM) Clustering. For classification, I focused on Classification Trees.

This diverse selection of models was chosen to handle the inherent complexity of financial data, which is often characterized by high dimensionality, collinearity, non-linearity, and the presence of outliers. Each method was selected for its unique ability to address these challenges, identify patterns, and generalize well to unseen data.

While clustering methods are not directly predictive, as they do not use the response variable during training, they can still provide meaningful insights. If clusters naturally align with variations in the response variable, they can serve as useful approximations, and cluster labels can be incorporated as new predictors in regression models to enhance performance.

To begin, all 97 predictors were included to ensure no information was excluded prematurely. Baseline models were created to evaluate the full dataset before applying statistically backed feature selection methods to reduce dimensionality and prevent overfitting.

The goal of this exploratory analysis was to determine feature importance and identify the best combination of predictors for a mixed model.

A rigorous exploratory analysis was conducted to better understand the data, including:

**Histogram Analysis:** To assess the distribution of the response variable and predictors.

**Scatterplot Analysis:** To visualize relationships between predictors and the response variable.

**Correlation Analysis:** To identify collinear relationships among predictors.

From the histogram analysis, it was evident that the response variable needed to be log-transformed to better conform to normality.

Correlation analysis revealed strong collinearity among financial ratio predictors and similarly among technical analysis-based predictors.

Scatterplots showed that many technical analysis-based predictors exhibited a strong linear relationship with the response variable.

## Feature Importance and Baseline Model Testing

Baseline models were constructed using the full set of predictors, both with the raw response variable and the log-transformed response. Feature importance was evaluated for each model to identify the top predictors contributing to performance. Baseline model performance was assessed using RMSE on the test set, providing a benchmark for further refinement.

### Why RMSE Was Chosen:

RMSE was chosen because I am predicting close prices, and my focus is on the absolute difference between predictions and actual observations. RMSE is expressed in the same units as the close price, making it both intuitive and easy to understand.

### Time Series Cross-Validation (Time Series Split):

To ensure robustness in training and testing, I used time series cross-validation (time series split). This method respects the temporal ordering of the data, ensuring that the training set always precedes the test set and avoids data leakage.

## Mixed Model Development

After analyzing feature importance across methods, I identified the top 10 predictors from each method and selected unique predictors to create an optimal mixed model. This approach ensured that the model leveraged the most informative features across all methods while minimizing redundancy.

The mixed model was then evaluated across all baseline methods to assess its performance relative to the full predictor models. The goal of this step was to determine whether the mixed model, with a reduced yet diverse set of predictors, could maintain or improve predictive accuracy and generalization.

## Final Comparison

The optimally mixed model was compared to the baseline models to assess improvements in prediction accuracy, generalization, and overall model efficiency. Among the models using the log-transformed response variable, Ridge and Lasso Regression demonstrated the lowest RMSE values, effectively managing multicollinearity through regularization. In contrast, models considering all predictors generally outperformed the mixed model in cases where the dataset's complexity required capturing hidden patterns.

### Model-Specific Insights:

**Ridge and Lasso Regression:** These methods performed well with the full predictor set but saw reduced predictive power when applied to the mixed model. This suggests the reduced feature set outweighed the benefits of regularization.

**Random Forest and Gradient Boosting:** Tree-based models benefited significantly from the mixed model, as irrelevant predictors were removed, allowing the models to focus on the most important features.

**Linear SVM and PCA Regression:** Both models showed improved performance with the mixed model, as feature selection reduced noise and emphasized key linear relationships.

## Results

### Full, Baseline Models

The most relevant and important results are highlighted below:

Linear Regression Average RMSE (Log Scale): 66247924.34874252

Linear Regression Average RMSE (Original Scale): inf

PCA Regression Average RMSE (Log Scale): 0.3535938673414937

PCA Regression Average RMSE (Original Scale): 1.424176664354599

Gradient Boosting Average RMSE (Log Scale): 0.24348653960017835

Gradient Boosting Average RMSE (Original Scale): 1.2756891464391382

Random Forest Average RMSE (Log Scale): 0.2407474488995537

Random Forest Average RMSE (Original Scale): 1.2721996992990103

Ridge Regression Average RMSE (Log Scale): 0.04601900778501562

Ridge Regression Average RMSE (Original Scale): 1.047094313711389

Lasso Regression Average RMSE (Log Scale): 0.041607199963150494

Lasso Regression Average RMSE (Original Scale): 1.0424849102056157

Linear SVM Average RMSE (Log Scale): 0.4667938446172881

Linear SVM Average RMSE (Original Scale): 1.5948725779569104

Non-linear SVM Average RMSE (Log Scale): 0.6068300772895748

Non-linear SVM Average RMSE (Original Scale): 1.8346066104355785

KMeans Clustering RMSE for Log Close (Log Scale): 0.23802173305004115

KMeans Clustering RMSE for Log Close (Original Scale): 1.2687367660449103

Hierarchical Clustering RMSE for Log Close (Log Scale): 0.3723168547812369

Hierarchical Clustering RMSE for Log Close (Original Scale): 1.4510926941163582

Model-Based Clustering RMSE for Log Close (Log Scale): 0.23845845586042463

Model-Based Clustering RMSE for Log Close (Original Scale): 1.2692909733396893

## Optimally Mixed Models

Linear Regression RMSE Original Scale: 1.3166661668616408

Ridge RMSE Original Scale: 1.2355178856459195

Lasso RMSE Original Scale: 1.2071336836237347

Random Forest RMSE Original Scale: 1.269863777706667

Gradient Boosting RMSE Original Scale: 1.2753331882329946

Linear SVM RMSE Original Scale: 1.5272381690448305

PCA Regression RMSE Original Scale: 1.2641572788408475

## Conclusion and Future Directions

In this project, I analyzed a wide range of statistical and machine learning models to address regression and classification problems using financial data. The analysis began with an exploratory evaluation of all 97 predictors, identifying strong linear and non-linear relationships with the response variable. This was followed by the development and testing of baseline models across all predictors, using RMSE as the primary

evaluation metric. Feature importance analysis led to the creation of a mixed model, leveraging the most informative predictors from multiple methods.

**Among the models evaluated, Lasso Regression with all predictors emerged as the best-performing model, achieving the lowest RMSE.** This result highlights Lasso's ability to handle multicollinearity while simplifying the model by shrinking irrelevant coefficients to zero. Ridge Regression also performed well, but its lack of inherent feature selection resulted in slightly reduced efficiency.

Tree-based methods like Random Forest and Gradient Boosting showed significant improvements when irrelevant predictors were removed, while PCA Regression and Linear SVM benefited from targeted feature selection to reduce noise.

Time series cross-validation was employed to ensure robust evaluation, respecting the temporal structure of the financial dataset and avoiding data leakage. This methodological rigor provided reliable insights into the predictive power and generalizability of each model.

While the academic objectives of this project have been fully met, this analysis has sparked curiosity about further enhancing stock price prediction through advanced techniques.

In particular, time series analysis (TSA) is a promising avenue to explore. TSA can incorporate lagged relationships, address non-stationarity, and identify temporal patterns to enhance predictive performance. Insights from TSA can be integrated with regression models, combining the strengths of traditional regression with time series dynamics.

Additionally, I plan to investigate Neural Networks (NNs) and Long Short-Term Memory (LSTM) networks to capture the complex, non-linear relationships and sequential dependencies inherent in financial data. Expanding the classification problem by leveraging insights from regression and clustering analyses will also allow for the development of a more comprehensive framework that balances feature importance, temporal trends, and non-linear dynamics.

In summary, this project has demonstrated the value of combining exploratory analysis, feature selection, and robust model evaluation techniques to address the challenges of complex financial datasets. While the current work fulfills its academic goals, the insights gained provide a strong foundation for future exploration and personal growth in advanced modeling techniques.

## Further Statistical Analysis and Results

The full statistical analysis encompassed feature importance selection, model selection, and model evaluation. All steps, including data extraction, cleaning, and preprocessing,

were carefully implemented to ensure robust and reproducible results.

The code used for these tasks will be provided below, the code provides insights into the methodologies and evaluation metrics that shaped the findings presented above.

## Data Extraction from Kaggle Files

```
In [1]: import pandas as pd

# Common file path and list of file endings
common_path = '/Users/aidanashrafi/Downloads/'
file_endings = [
    'market_indicators.csv',
    'AMZN_fundamentals_train.csv',
    'AMZN_fundamentals_test.csv',
    'AMZN_technicals_test.csv',
    'AMZN_technicals_train.csv',
    'revenue_profit.csv',
    'ratios.csv',
    'price_fundamentals.csv',
    'assets_liabilities.csv',
    'price.csv',
    'fundamentals.csv'
]

# Read and combine data
dataframes = [pd.read_csv(f"{common_path}{ending}") for ending in file_endings]
full_data = pd.concat(dataframes, ignore_index=True)
```

## Data Cleaning:

```
In [2]: # Ensure 'Date' is in datetime format
full_data['Date'] = pd.to_datetime(full_data['Date'])
# Remove duplicate columns
full_data = full_data.loc[:, ~full_data.columns.duplicated()]
# Sort the data by 'Date' to ensure chronological order
full_data.sort_values(by='Date', inplace=True)
# Reset the index for a clean DataFrame
full_data.reset_index(drop=True, inplace=True)
# Deleting Duplicate columns after inspection
full_data.drop(columns=['pb_ratio', 'ps_ratio', 'pfcf_ratio', 'price'], inplace=True)

print(f"Number of columns after removing duplicates: {len(full_data.columns)}")

# Group by 'Date', fill missing values, and consolidate rows
full_data = (
    full_data.groupby('Date', as_index=False)
    .apply(lambda group: group.drop(columns=['Date']).ffill().bfill().iloc[0])
    .reset_index(drop=True)
)
```

Number of columns after removing duplicates: 98

```
/var/folders/2p/hbz8h54x5hj828cdgg7jshz0000gn/T/ipykernel_66719/3208626561.py:16: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include_gr oups=False` to exclude the groupings or explicitly select the grouping colum ns after groupby to silence this warning.
    full_data.groupby('Date', as_index=False)
```

## Missing Values

Determine number of missing values for each variable

```
In [3]: missing_summary = full_data.isnull().sum().sort_values(ascending=False)
missing_percentage = (missing_summary / len(full_data)) * 100
print(missing_percentage[missing_percentage > 0])

lt_investments_&_debt           98.536585
return_on_investment             98.507891
price_to_fcf_ratio              98.507891
price_to_sales_ratio             98.507891
long_term_debt_excl._capitalized_leases 98.479197
...
treasury_10_years                0.659971
treasury_5_years                 0.659971
gold                            0.631277
oil                             0.573888
dxy                            0.487805
Length: 97, dtype: float64
```

```
In [4]: # Identify columns with more than 95% missing values
columns_to_ffill = missing_percentage[missing_percentage > 95].index.tolist()
# Apply forward-fill for these columns
full_data[columns_to_ffill] = full_data[columns_to_ffill].ffill()

# If necessary, apply backward-fill
full_data[columns_to_ffill] = full_data[columns_to_ffill].bfill()
# Check the results
print("Columns processed with forward-fill", columns_to_ffill)
```

Columns processed with forward-fill ['lt\_investments\_&\_debt', 'return\_on\_inv estment', 'price\_to\_fcf\_ratio', 'price\_to\_sales\_ratio', 'long\_term\_debt\_exc l.\_capitalized\_leases', 'shares\_outstanding', 'price\_to\_book\_ratio']

```
In [5]: # Check for missing values in core market data
missing_market_data = full_data[['open', 'high', 'low', 'close', 'volume']]
# Remove rows where all four columns are missing
full_data = full_data[~missing_market_data].reset_index(drop=True)
```

```
In [6]: # Split the data into training (2010–2020) and testing (2020–2023) periods
train_data = full_data[(full_data['Date'] >= '2010-01-14') & (full_data['Dat
test_data = full_data[(full_data['Date'] > '2019-12-31') & (full_data['Date']

# Check the results
print("Training Data Range:", train_data['Date'].min(), "-", train_data['Dat
print("Testing Data Range:", test_data['Date'].min(), "-", test_data['Date'])
```

Training Data Range: 2010-01-14 00:00:00 – 2019-12-31 00:00:00  
 Testing Data Range: 2020-01-02 00:00:00 – 2022-06-28 00:00:00

```
In [7]: missing_summary = train_data.isnull().sum().sort_values(ascending=False)
missing_percentage = (missing_summary / len(full_data)) * 100
print(missing_percentage[missing_percentage > 0])
```

gold	0.089928
treasury_5_years	0.089928
treasury_10_years	0.089928
treasury_30_years	0.089928
dxy	0.059952
oil	0.059952
dtype:	float64

```
In [8]: # Missing Data Train
# List of macroeconomic indicators
macroeconomic_data = ['gold', 'treasury_5_years', 'treasury_10_years', 'treasury_30_years', 'dxy', 'oil']

train_data.loc[:, macroeconomic_data] = train_data.loc[:, macroeconomic_data]

missing_summary_train = train_data.isnull().sum().sort_values(ascending=False)
missing_percentage_train = (missing_summary_train / len(full_data)) * 100
print(missing_percentage_train[missing_percentage_train > 0])
```

Series([], dtype: float64)

```
In [9]: missing_summary_test = test_data.isnull().sum().sort_values(ascending=False)
missing_percentage_test = (missing_summary_test / len(full_data)) * 100
print(missing_percentage_test[missing_percentage_test > 0])
```

Series([], dtype: float64)

It is clear that there are no longer missing values in the training or test set, so we can proceed with the analysis.

## Histograms

```
In [10]: # Histograms

import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

# Generate histograms with a normal distribution overlay for numeric variables
for column in train_data.columns:
    if train_data[column].dtype in ['int64', 'float64']: # Only numeric columns
        data = train_data[column].dropna() # Drop NaN values if any
        mean, std = data.mean(), data.std() # Calculate mean and standard deviation
        plt.figure(figsize=(8, 4))

        # Plot histogram
        plt.hist(data, bins=30, density=True, edgecolor='k', alpha=0.7, label=f'{column} Distribution')

        # Overlay normal distribution
        x = np.linspace(data.min(), data.max(), 100)
        y = norm.pdf(x, mean, std)
        plt.plot(x, y, 'r--', label='Normal Distribution')

        plt.title(f'Histogram of {column} with Normal Distribution Overlay')
        plt.xlabel(column)
        plt.ylabel('Density')
        plt.legend()
        plt.show()
```

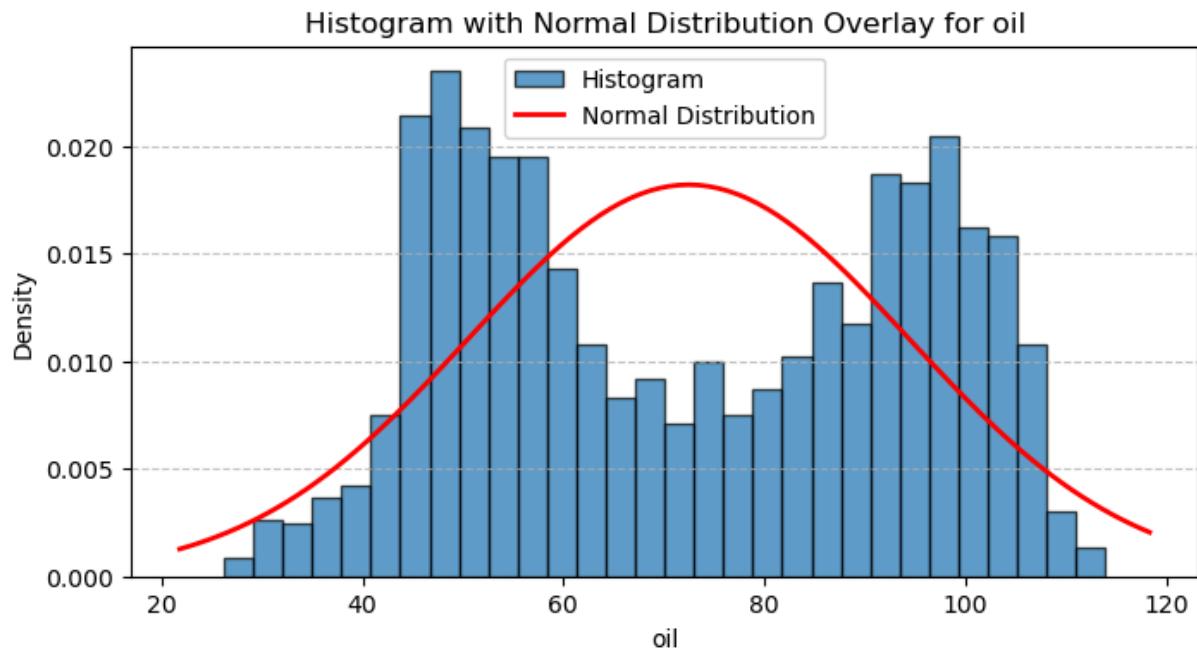
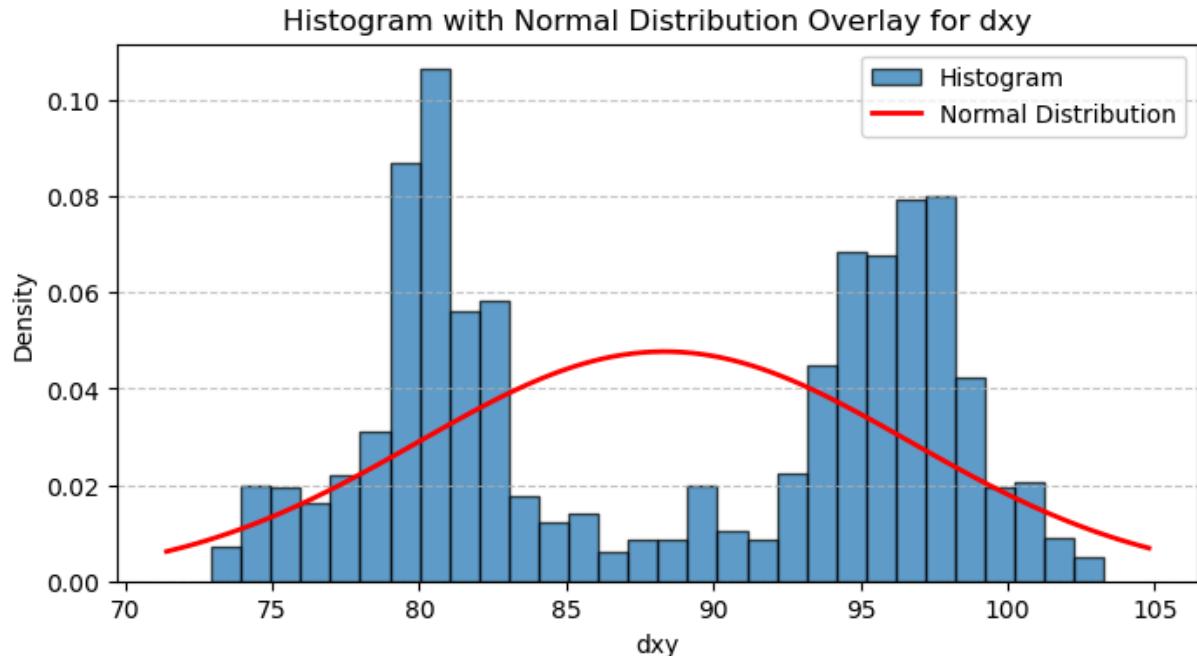
```

xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mean, std)
plt.plot(x, p, 'r', linewidth=2, label='Normal Distribution')

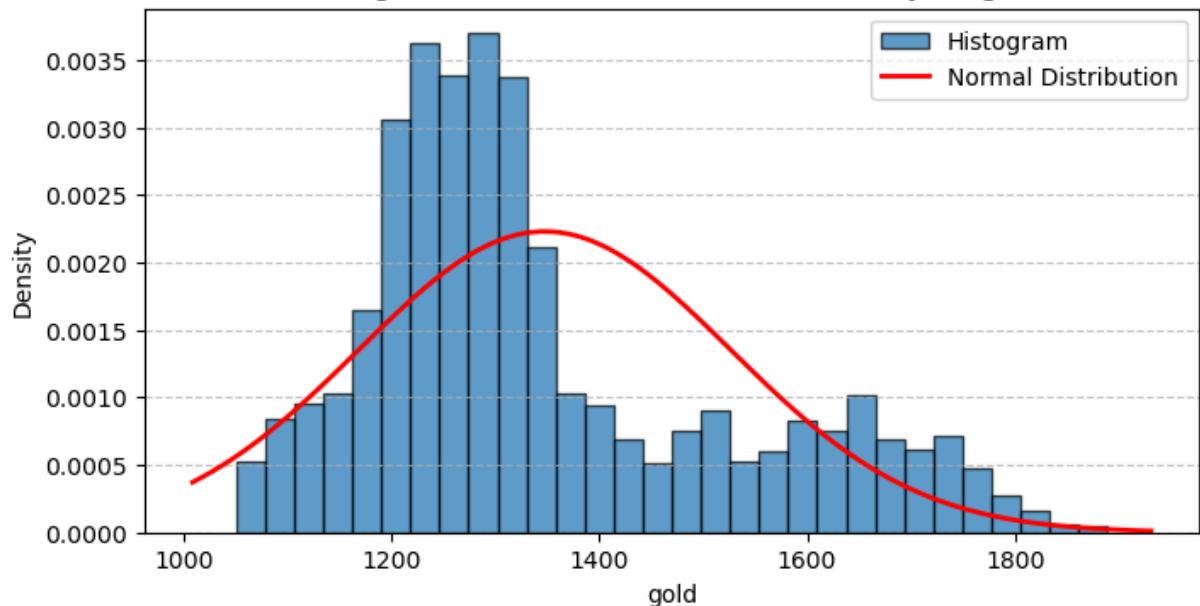
# Add title, labels, and legend
plt.title(f"Histogram with Normal Distribution Overlay for {column}")
plt.xlabel(column)
plt.ylabel("Density")
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()

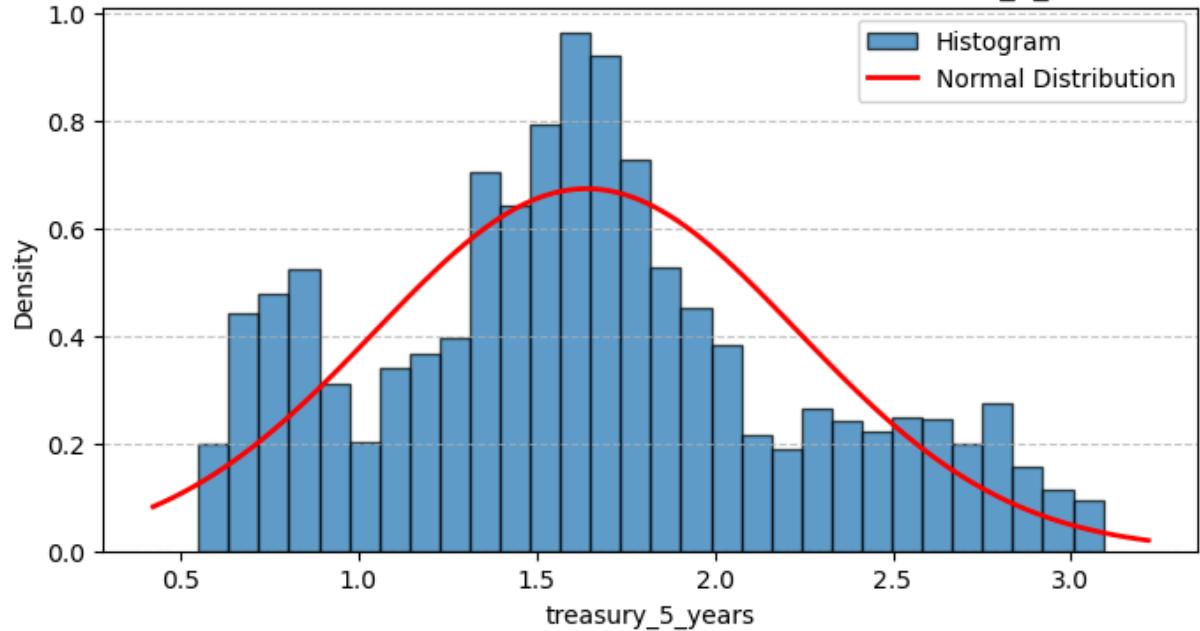
```

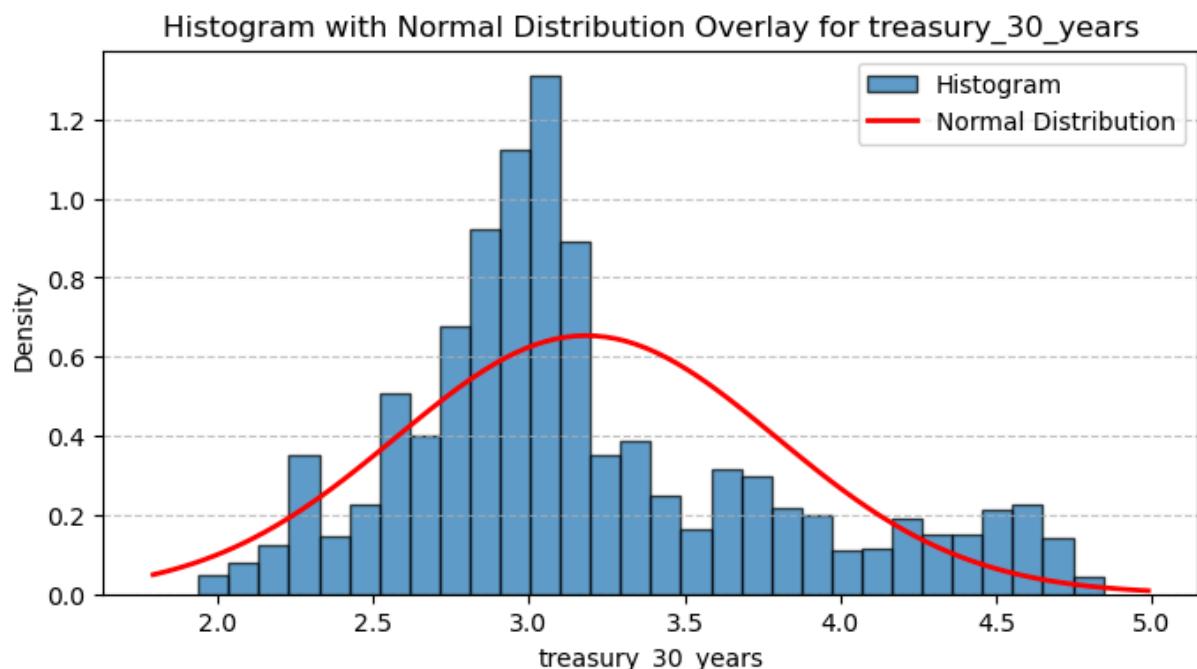
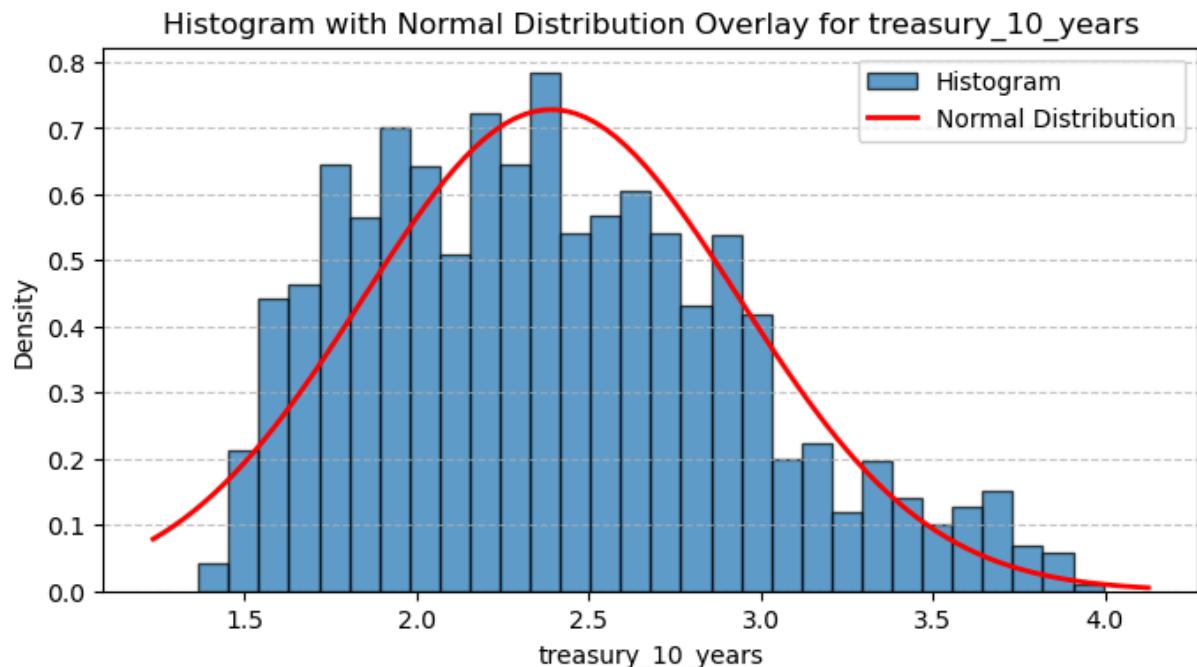


Histogram with Normal Distribution Overlay for gold

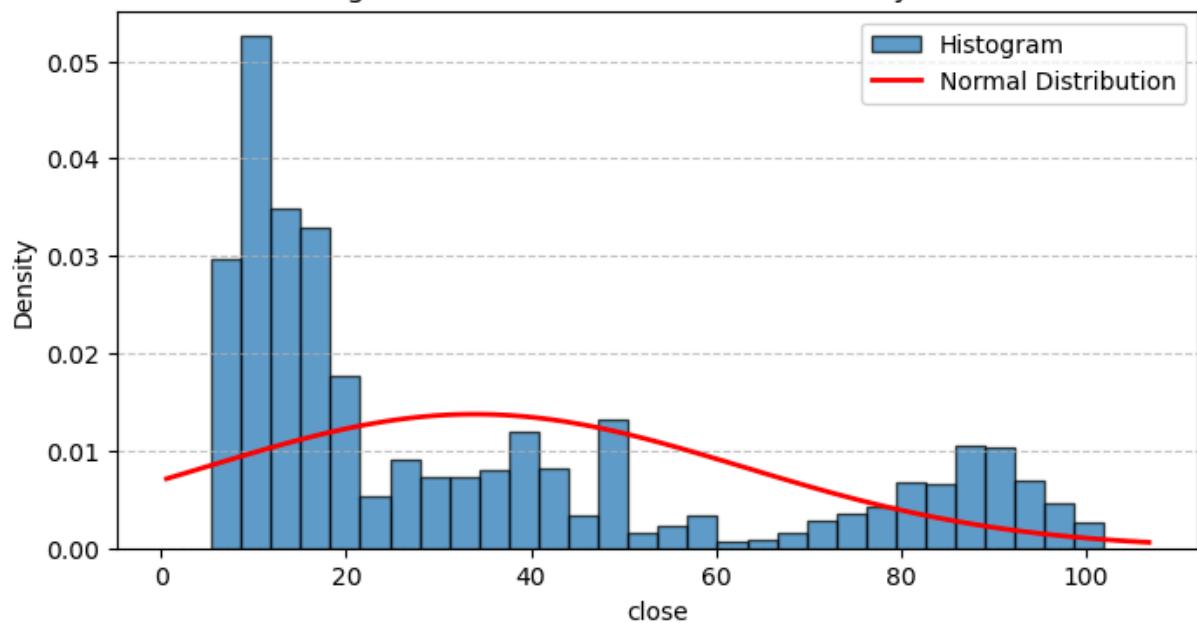


Histogram with Normal Distribution Overlay for treasury\_5\_years

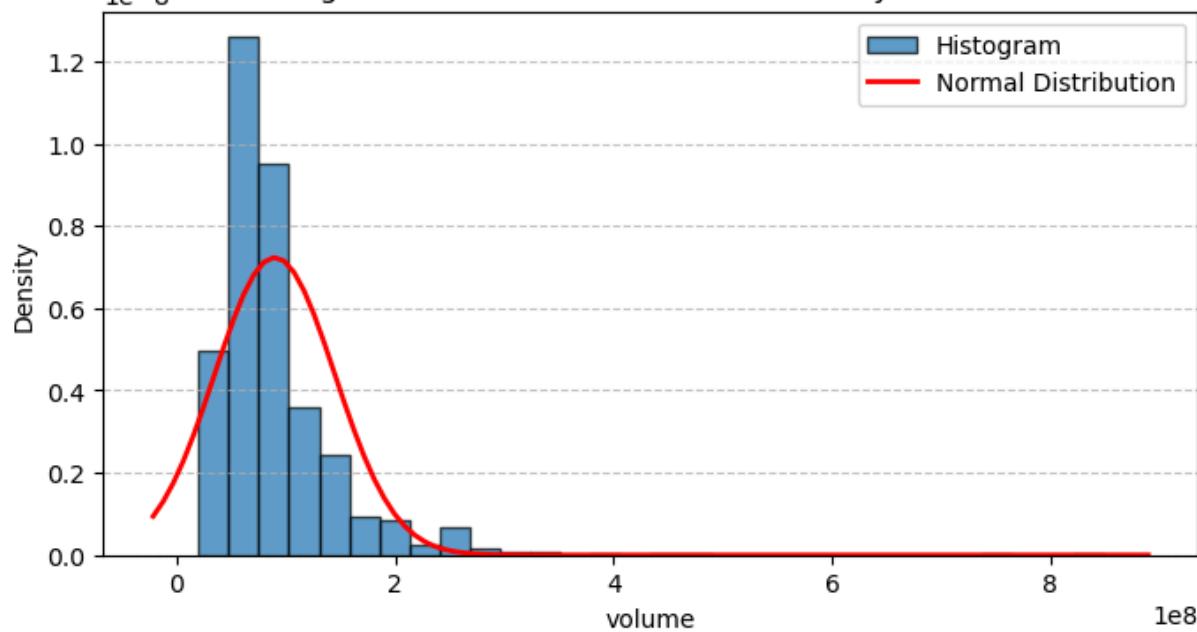


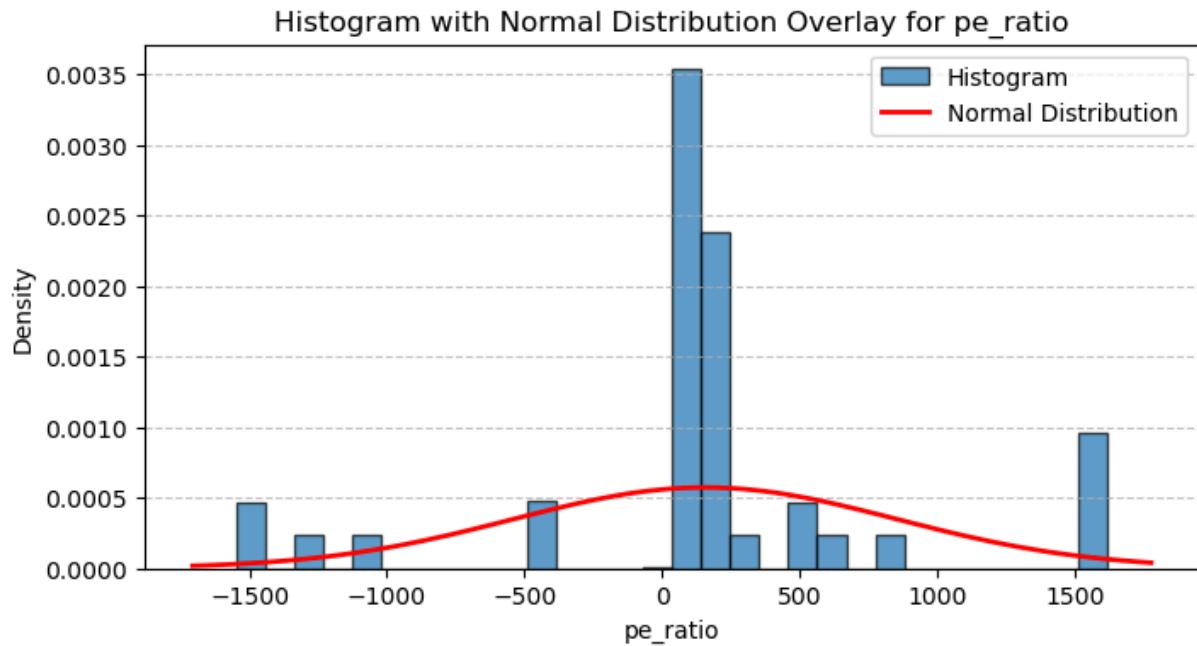
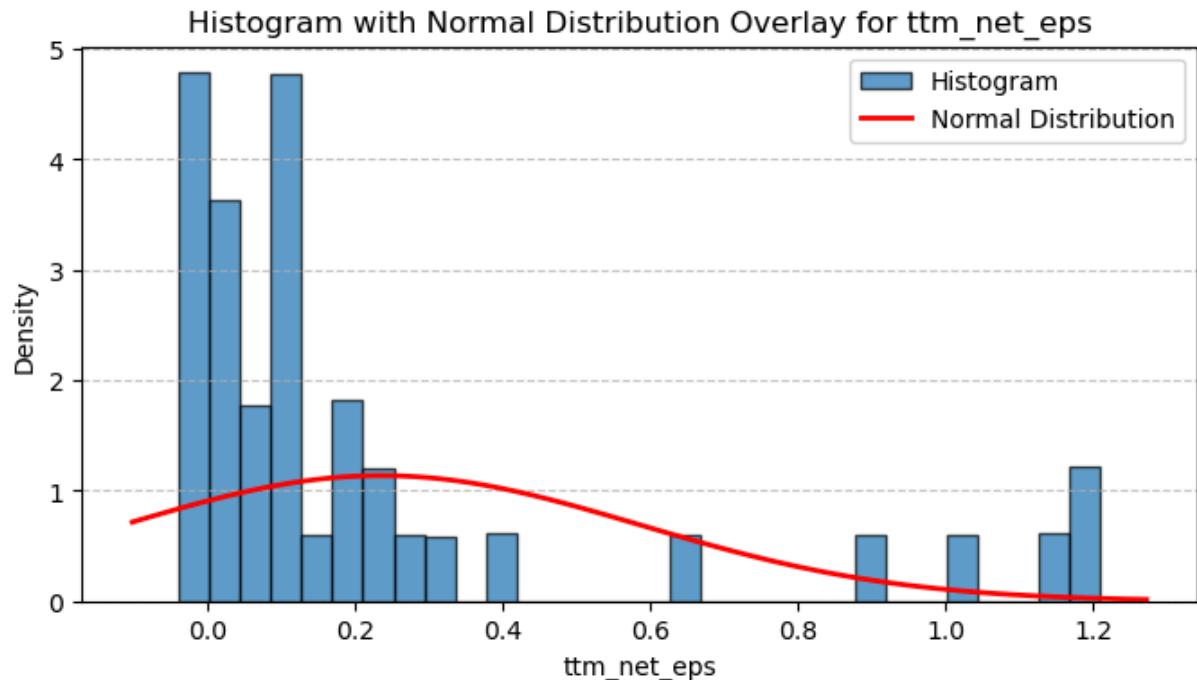


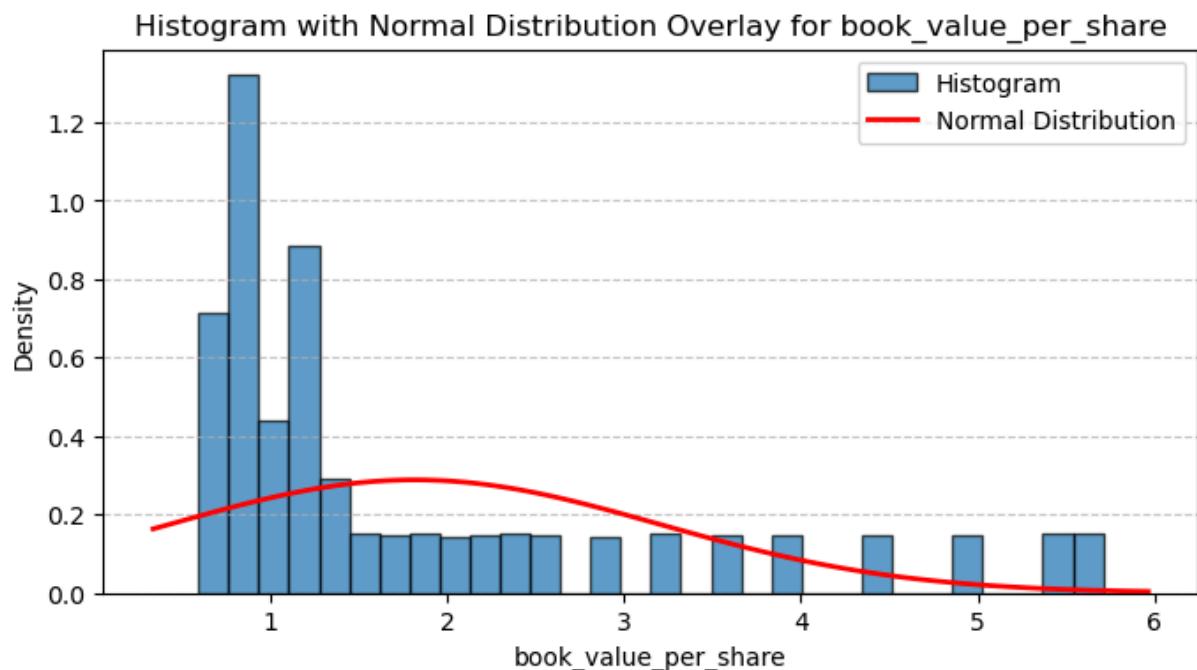
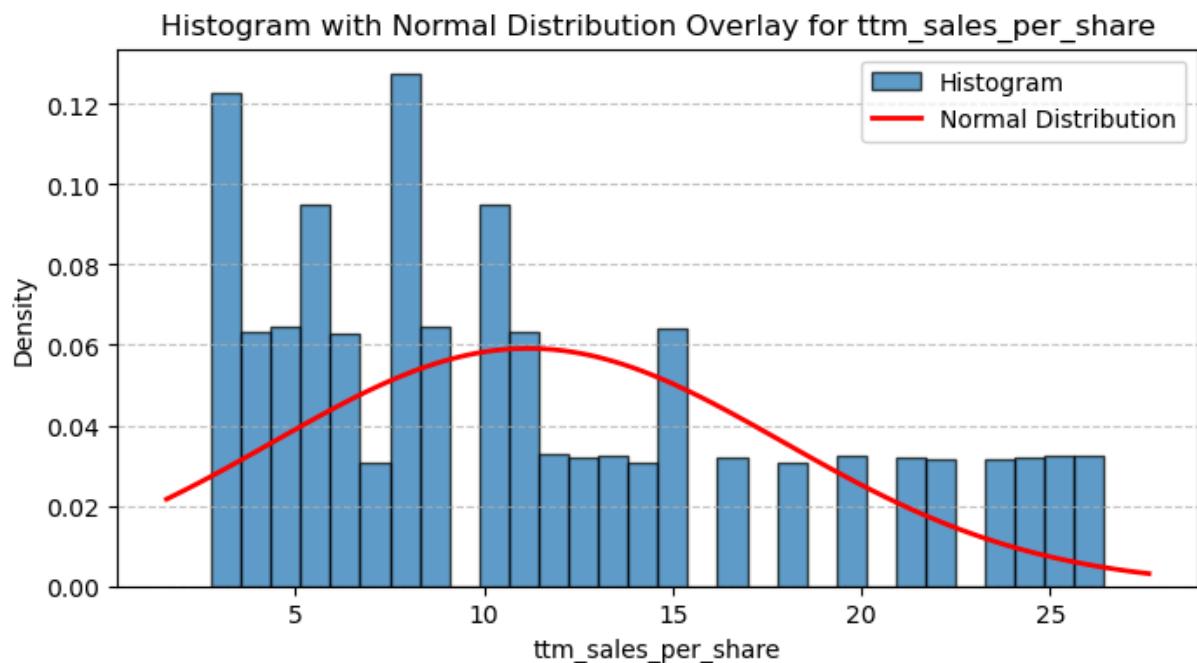
Histogram with Normal Distribution Overlay for close



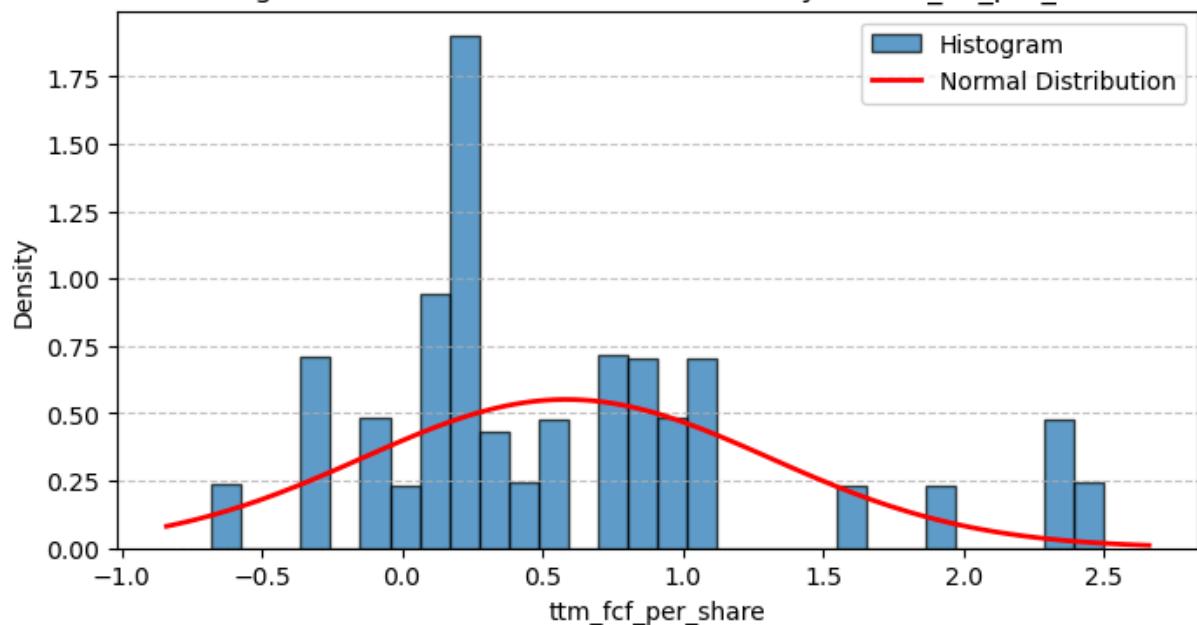
Histogram with Normal Distribution Overlay for volume



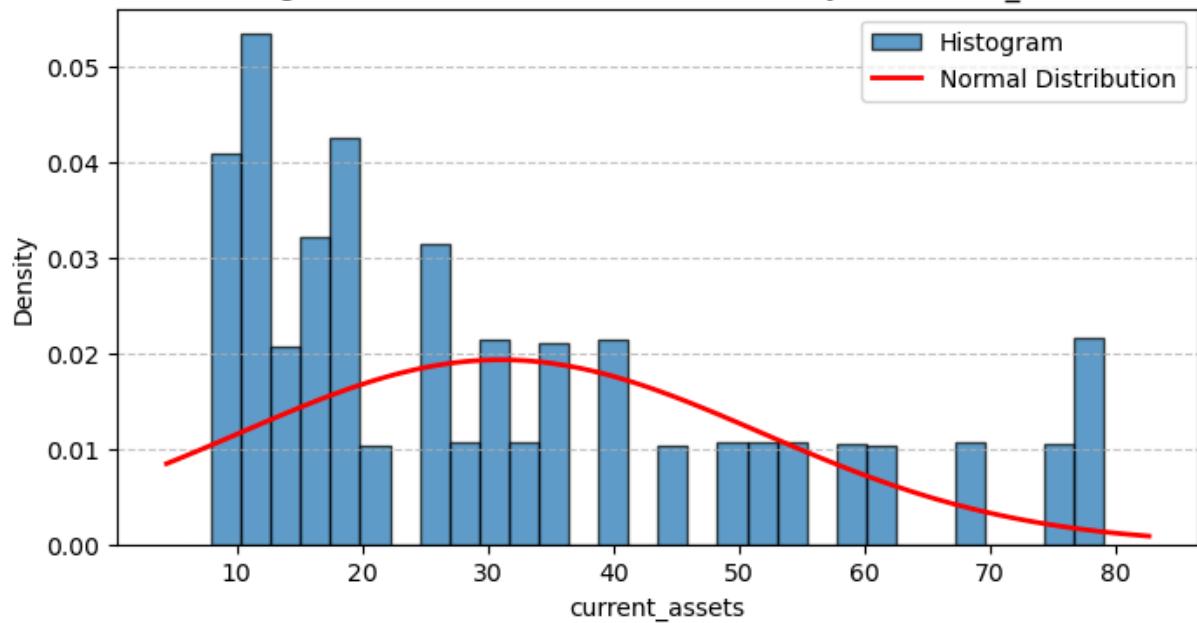


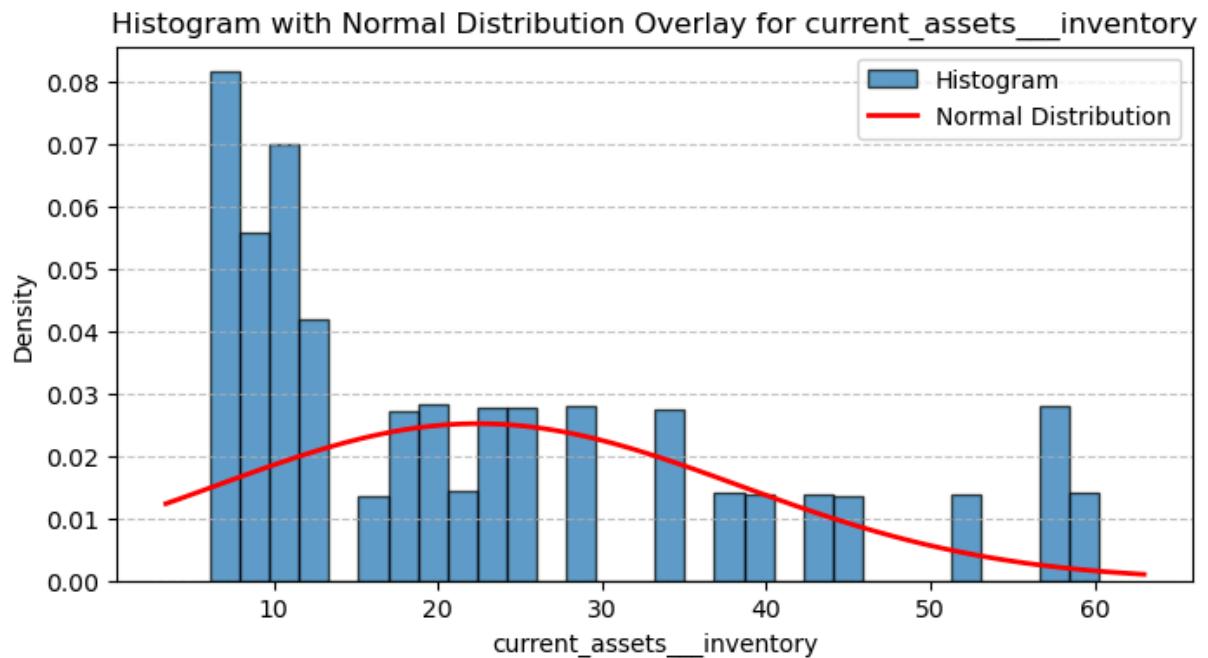
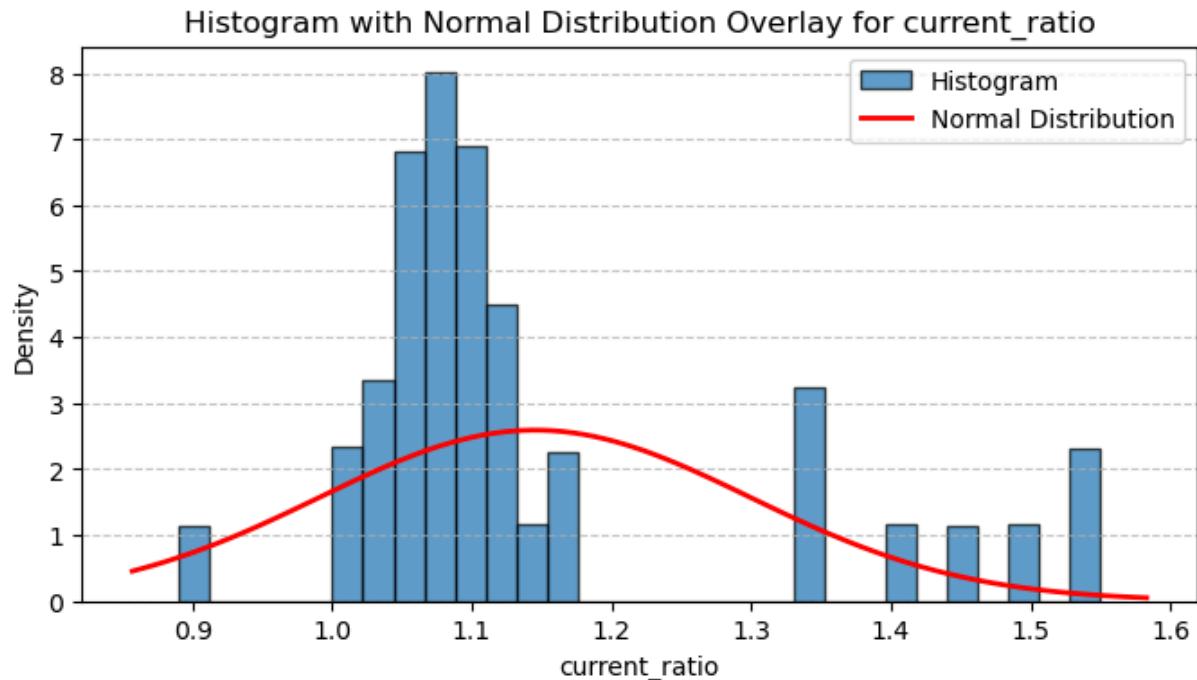


Histogram with Normal Distribution Overlay for ttm\_fcf\_per\_share

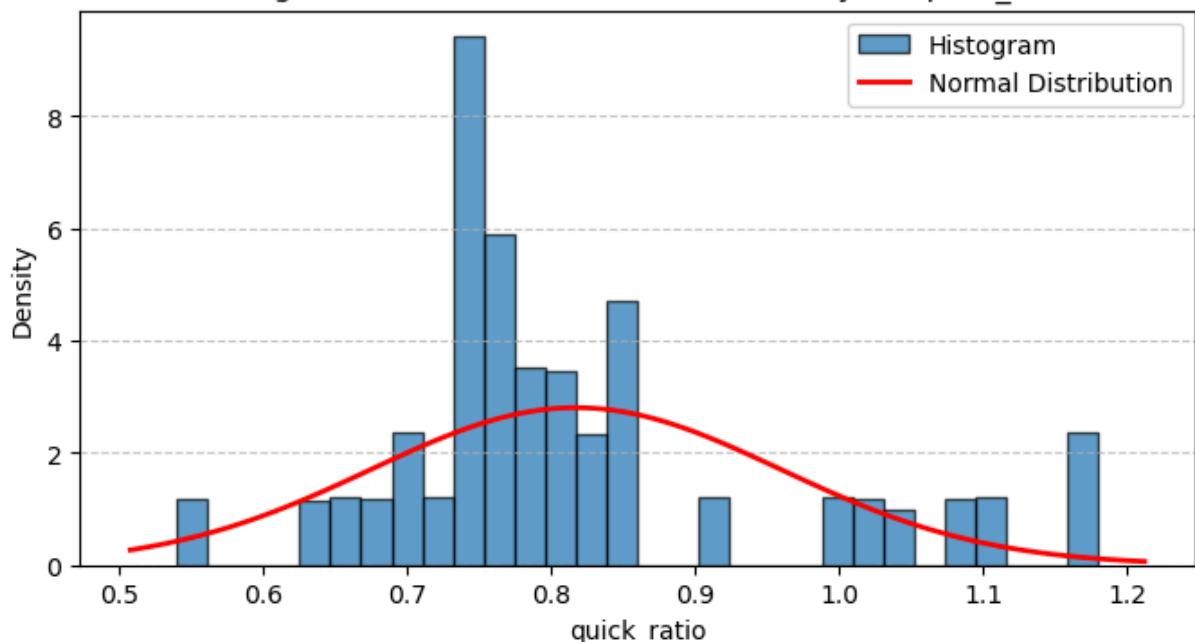


Histogram with Normal Distribution Overlay for current\_assets

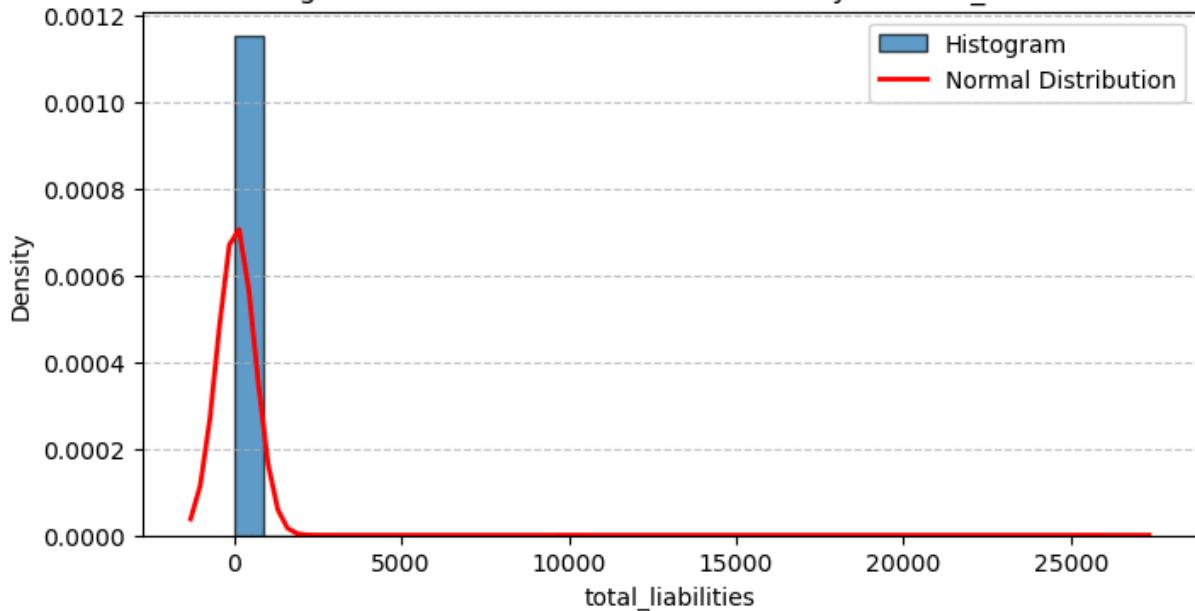


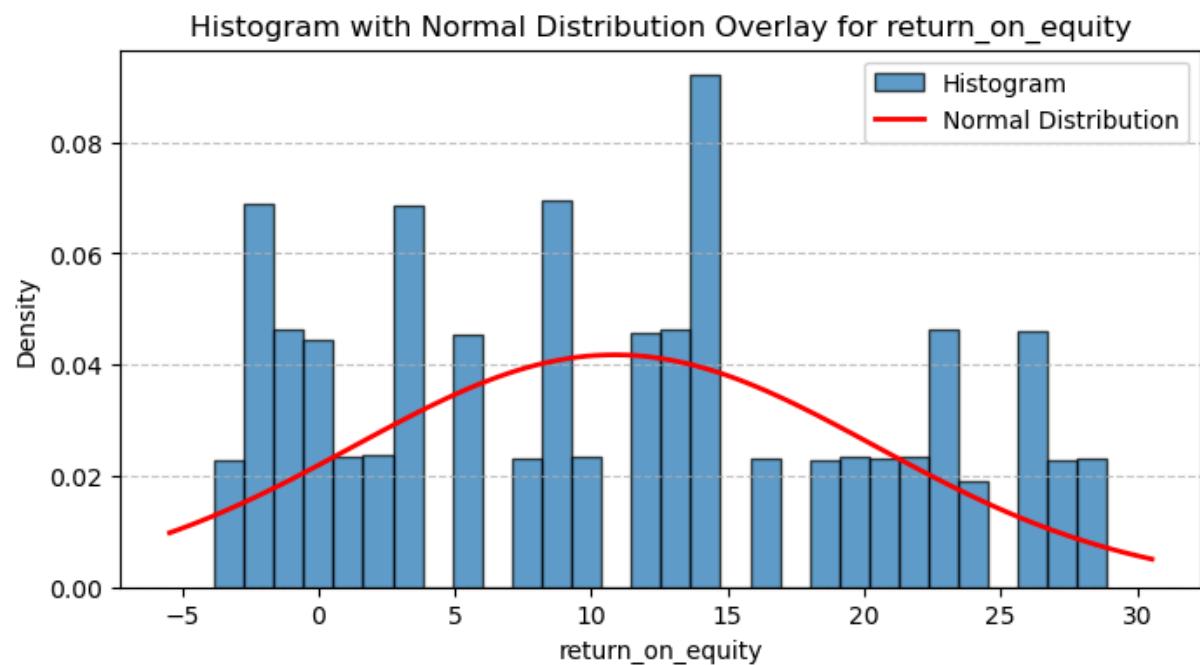
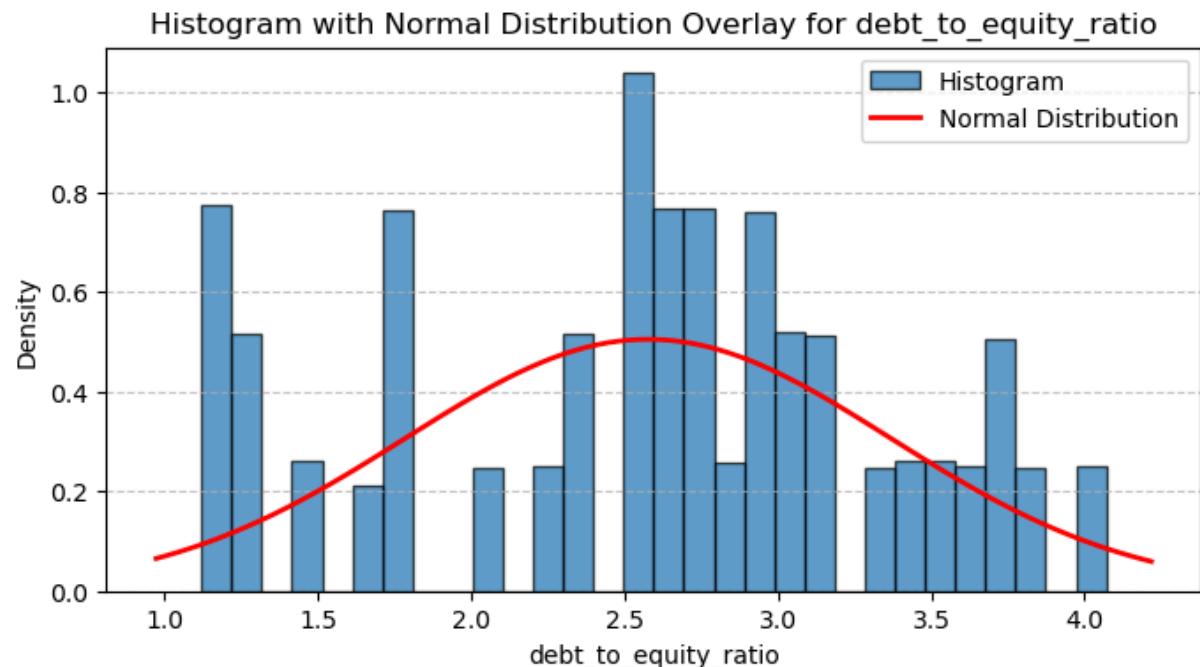


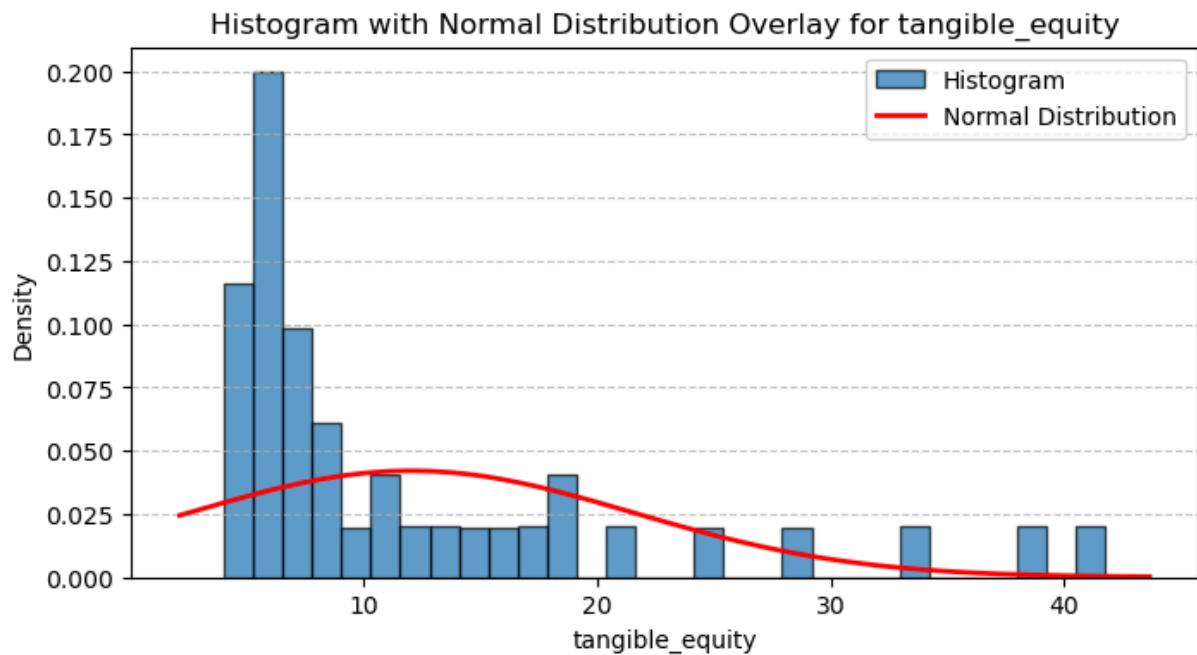
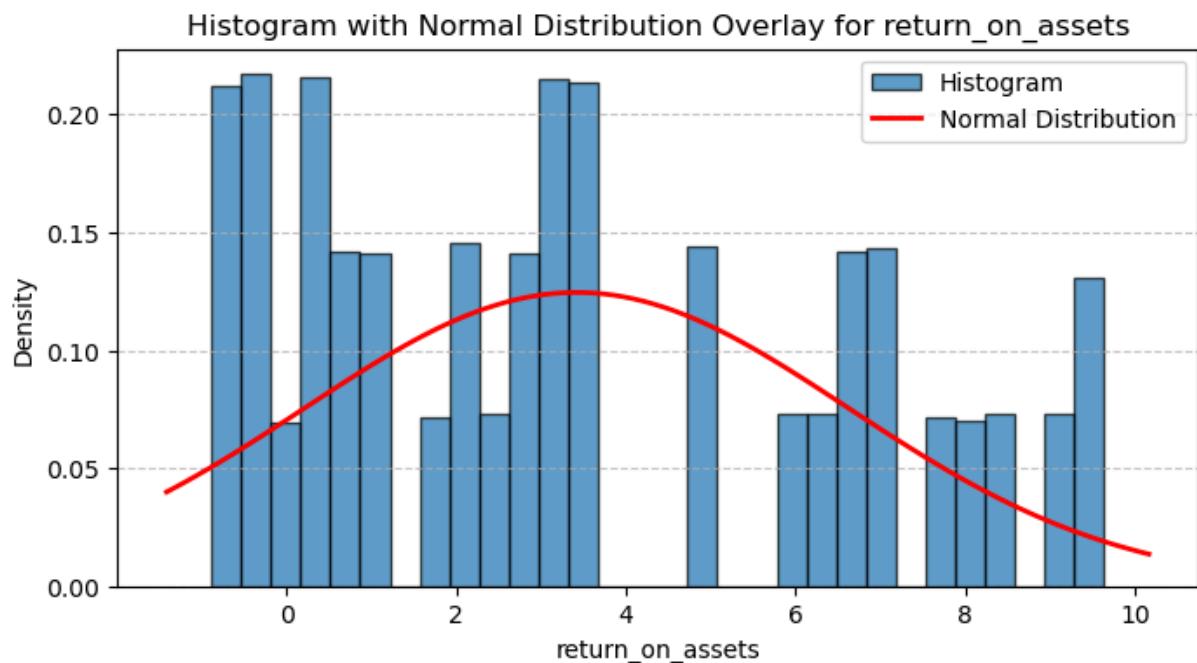
Histogram with Normal Distribution Overlay for quick\_ratio

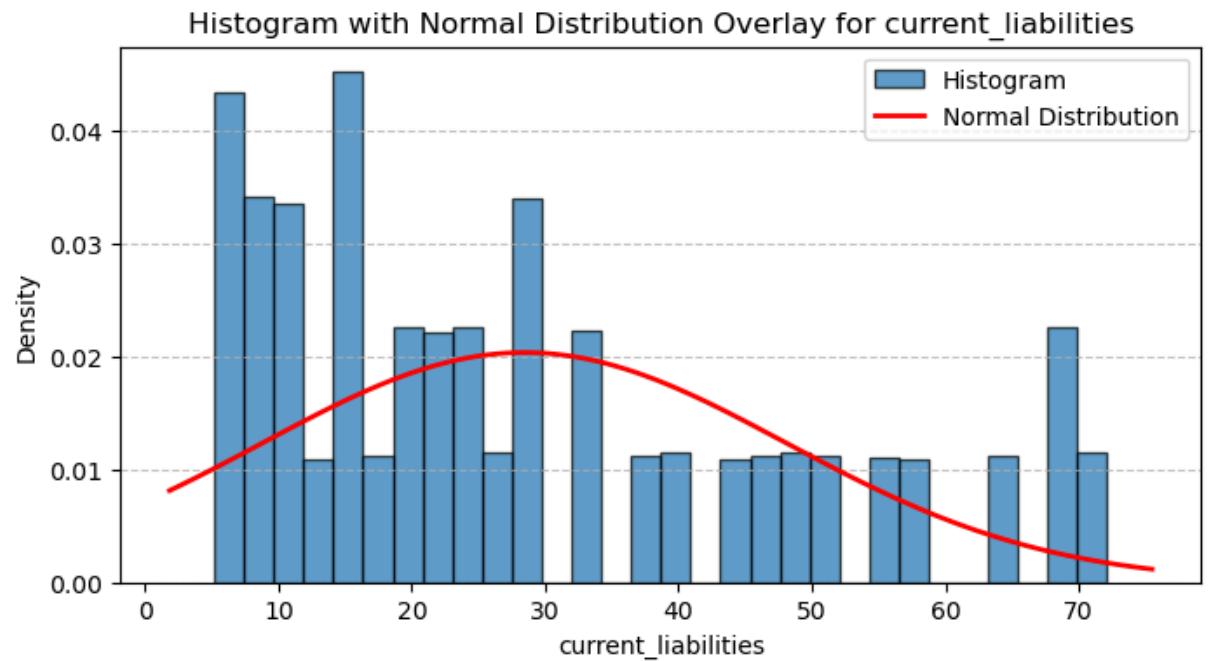
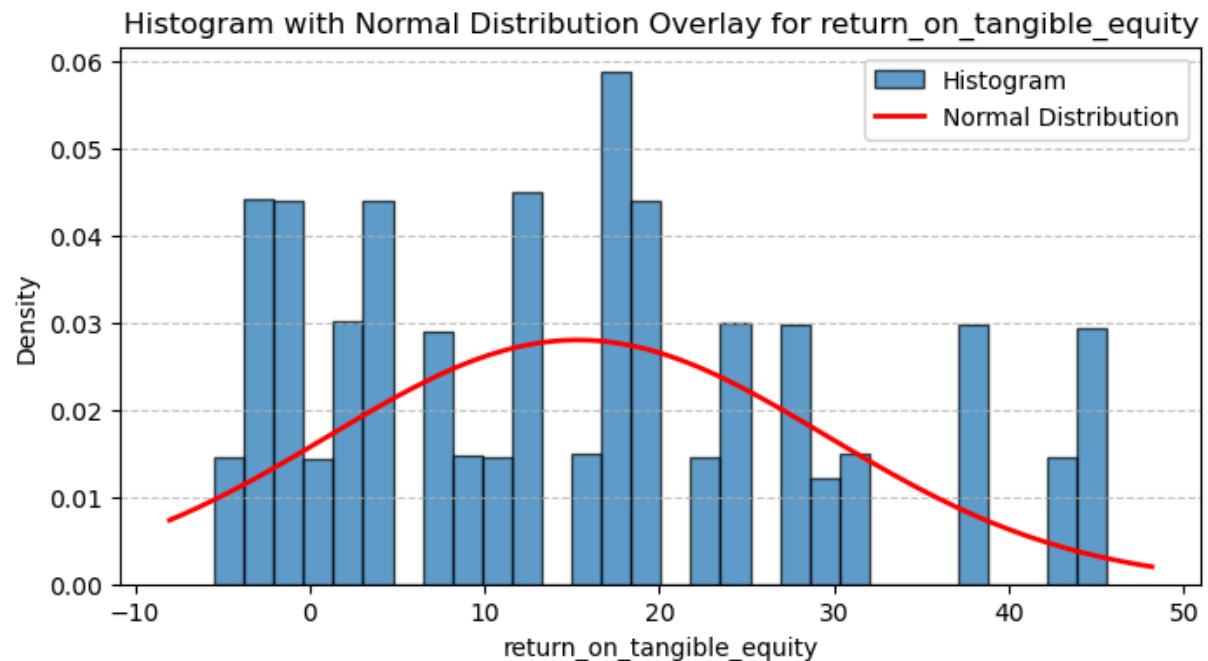


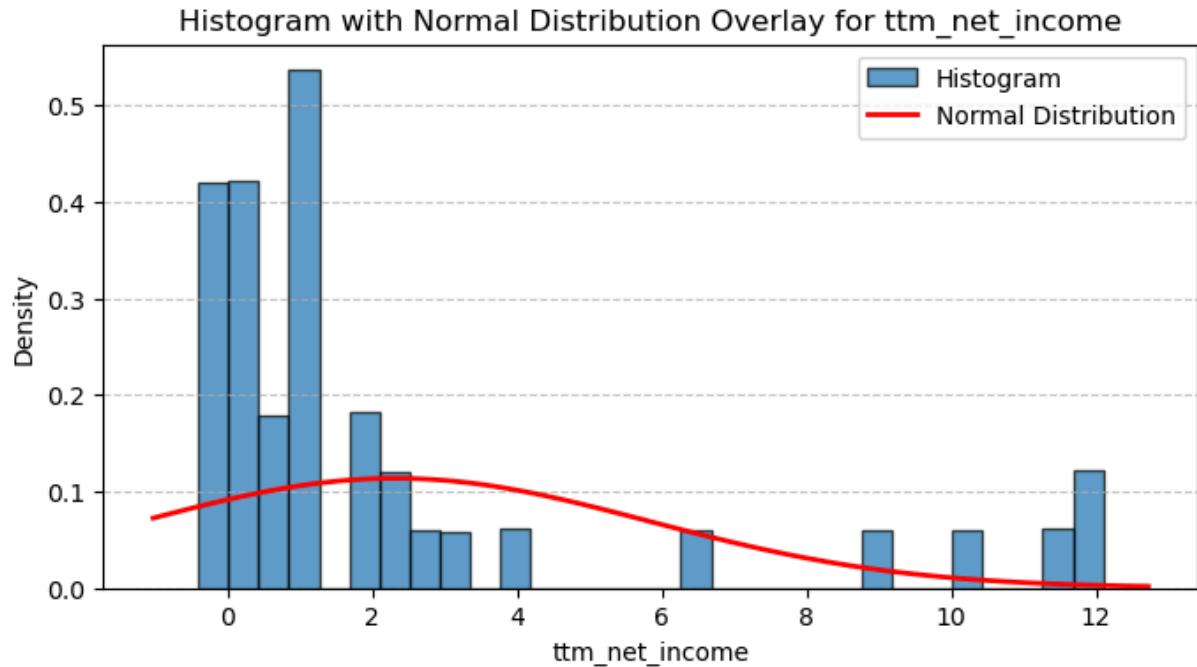
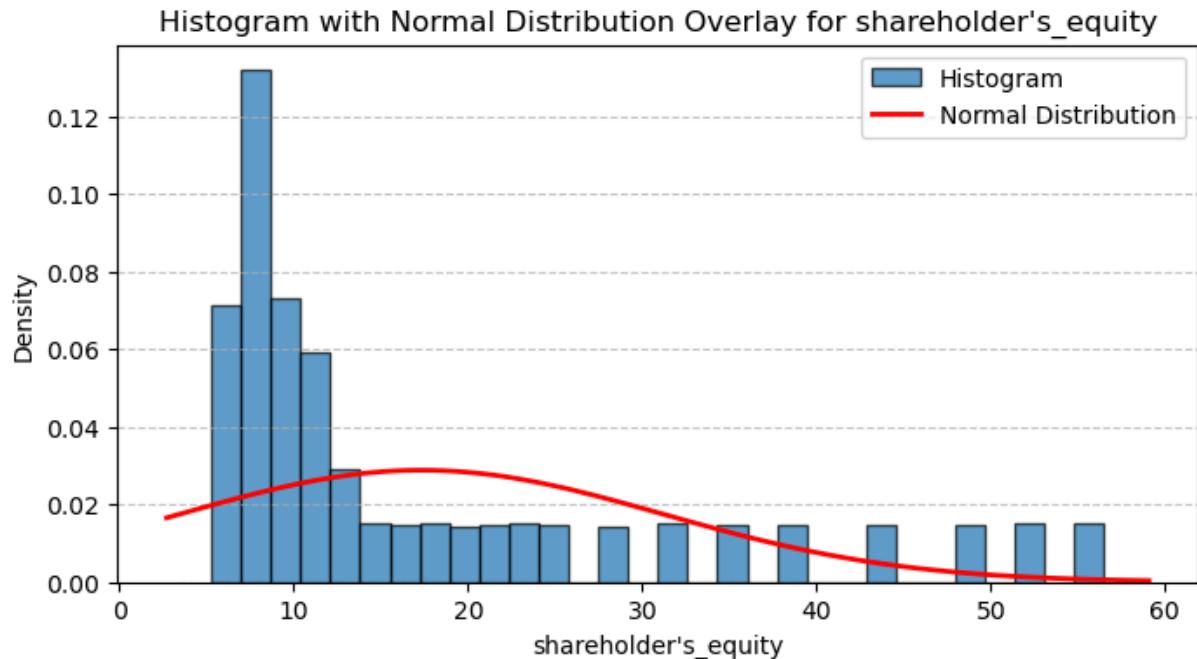
Histogram with Normal Distribution Overlay for total\_liabilities

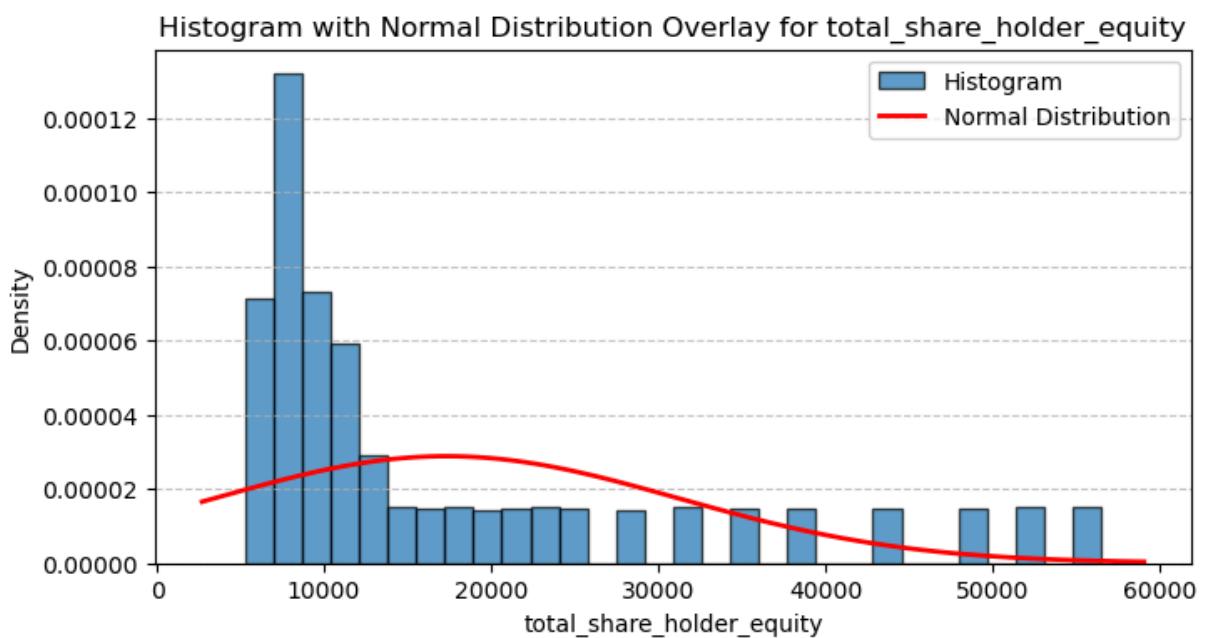
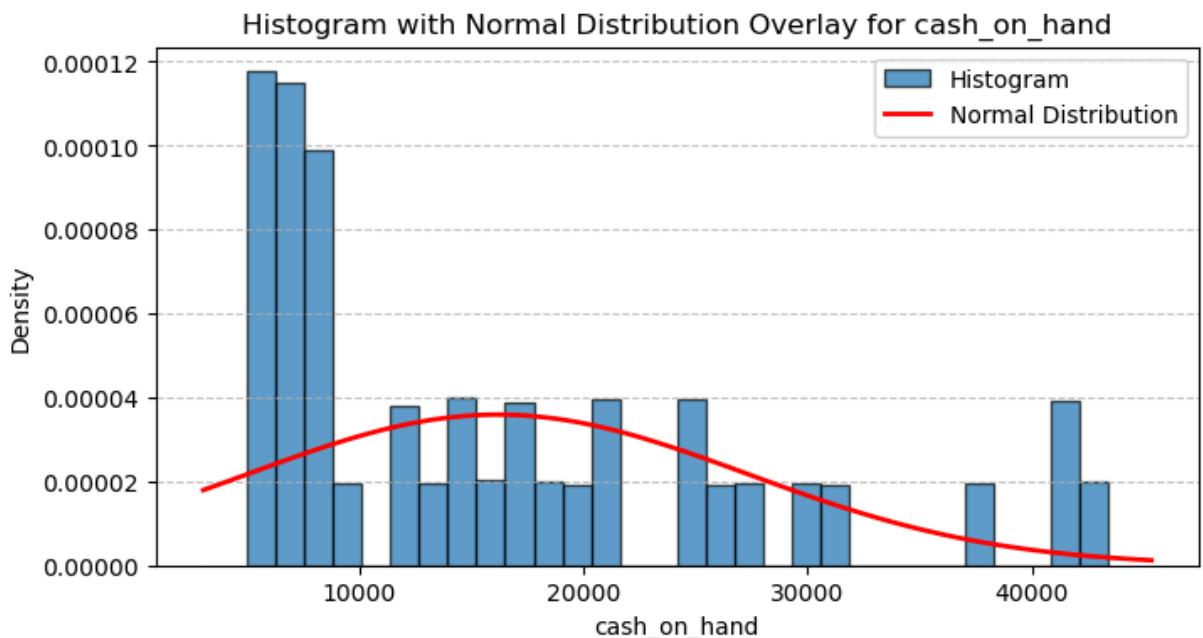


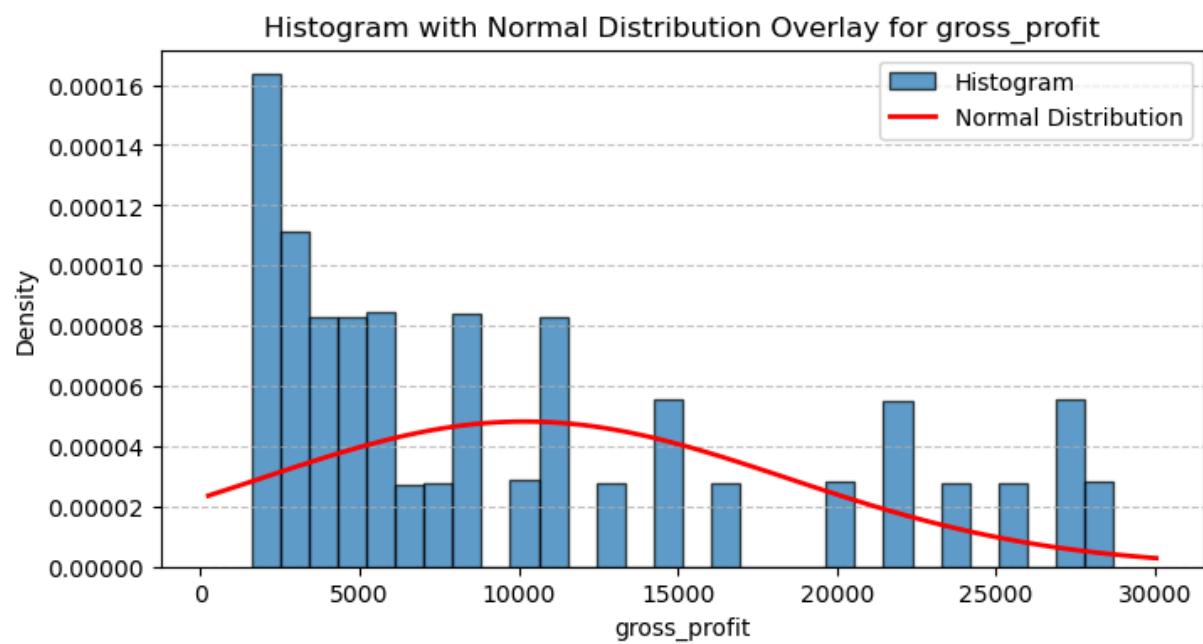
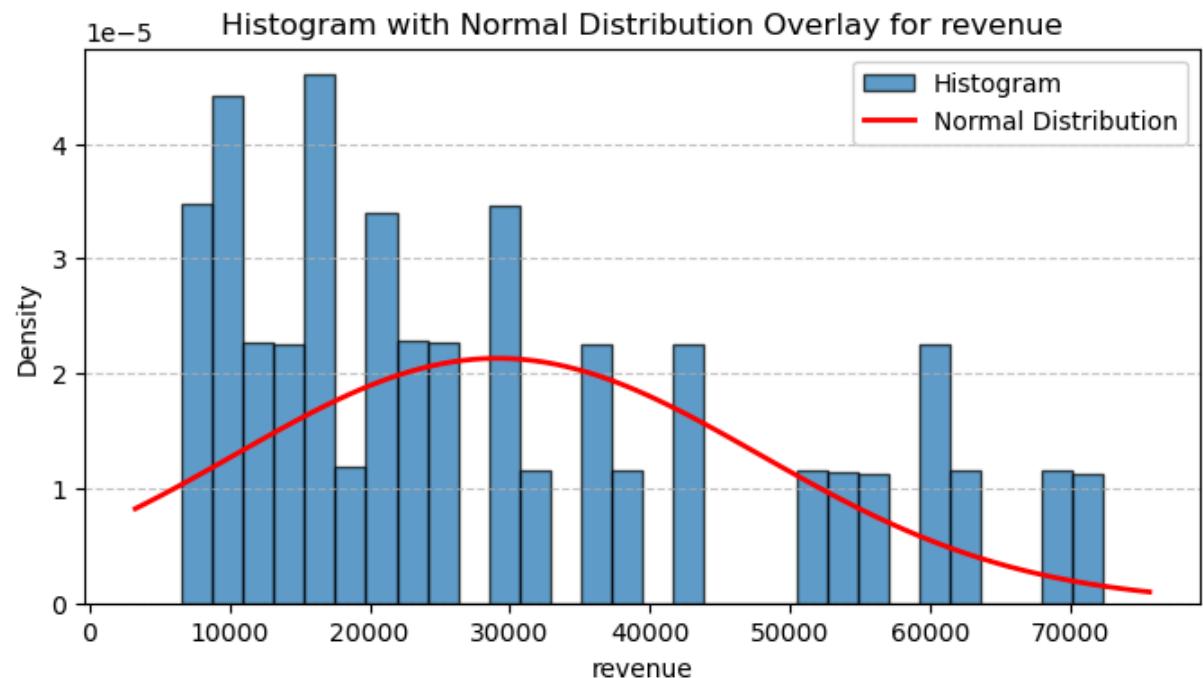


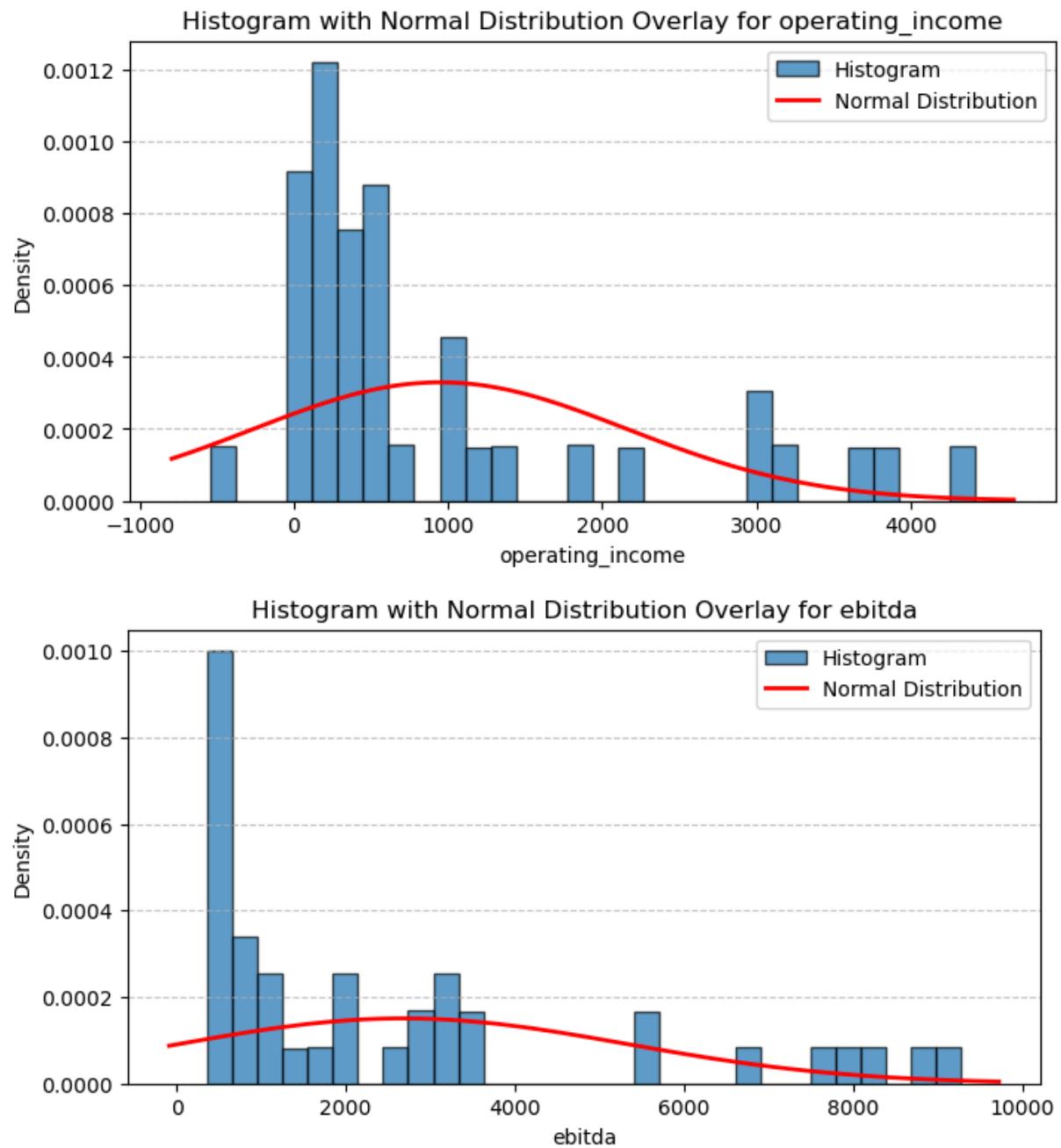


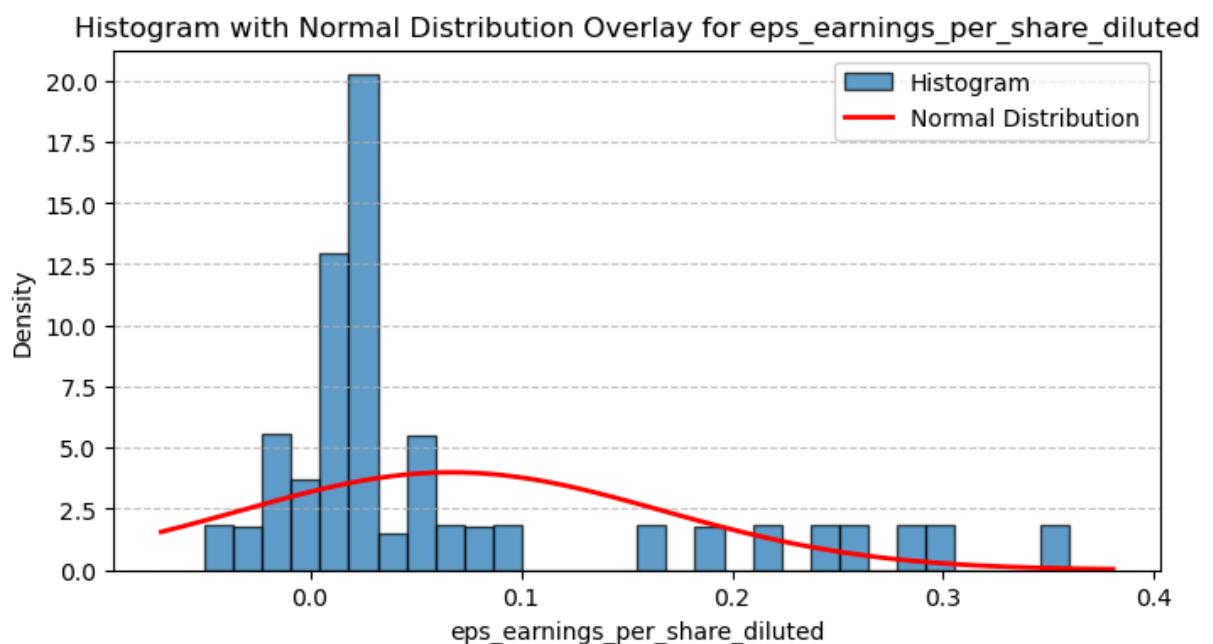
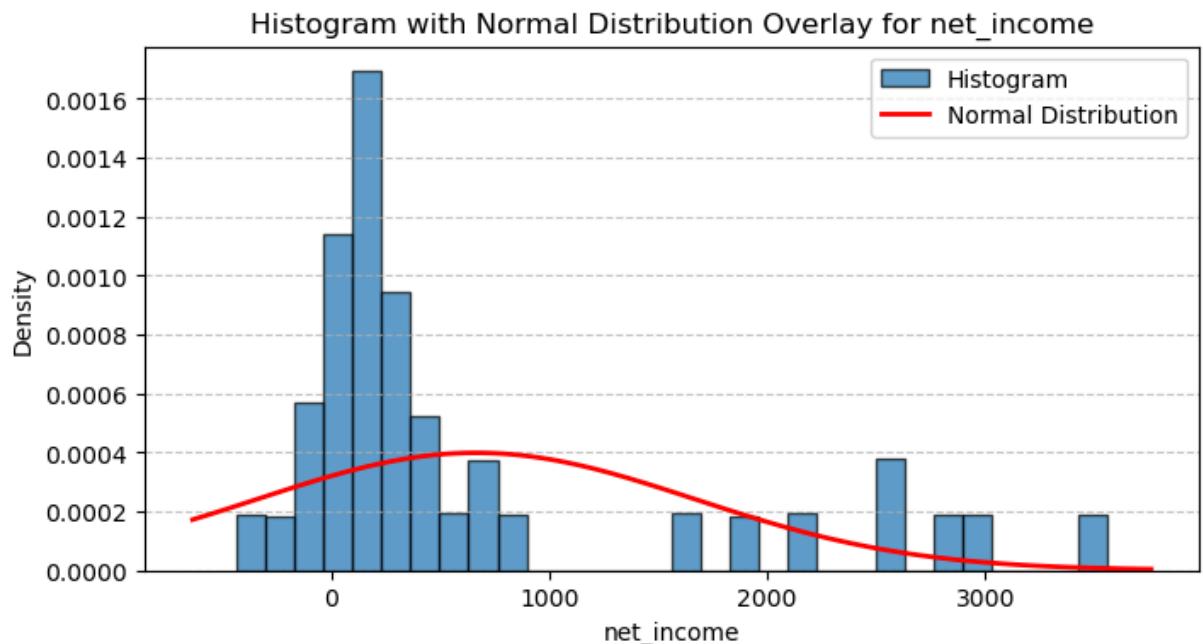


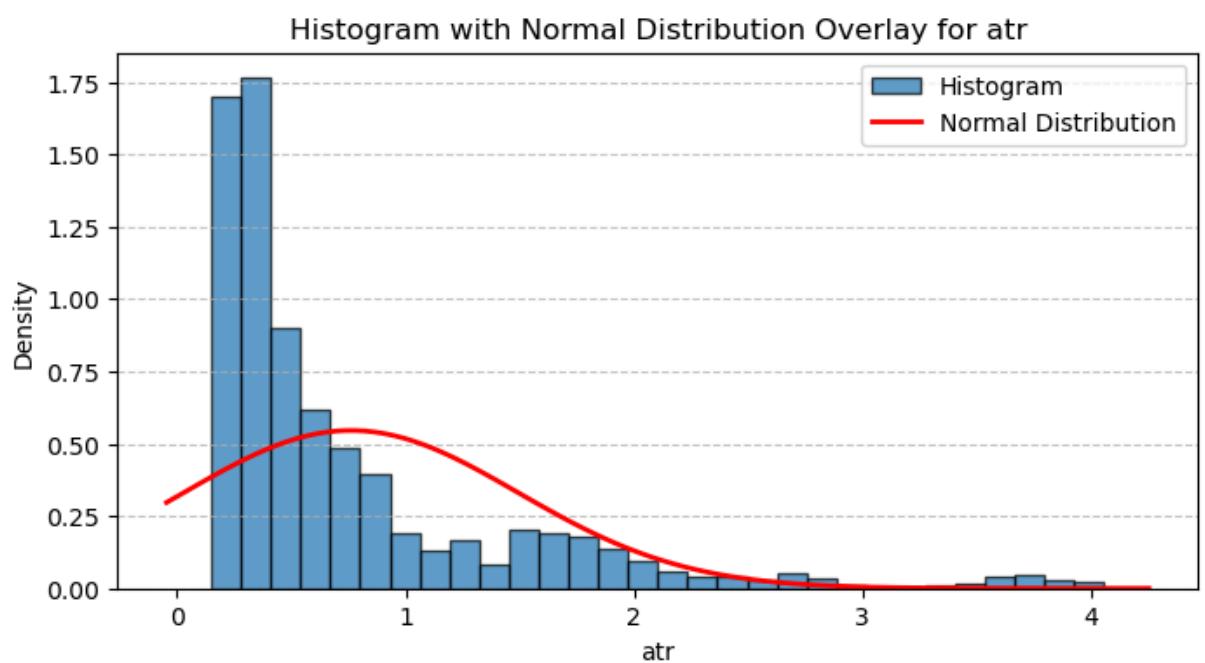
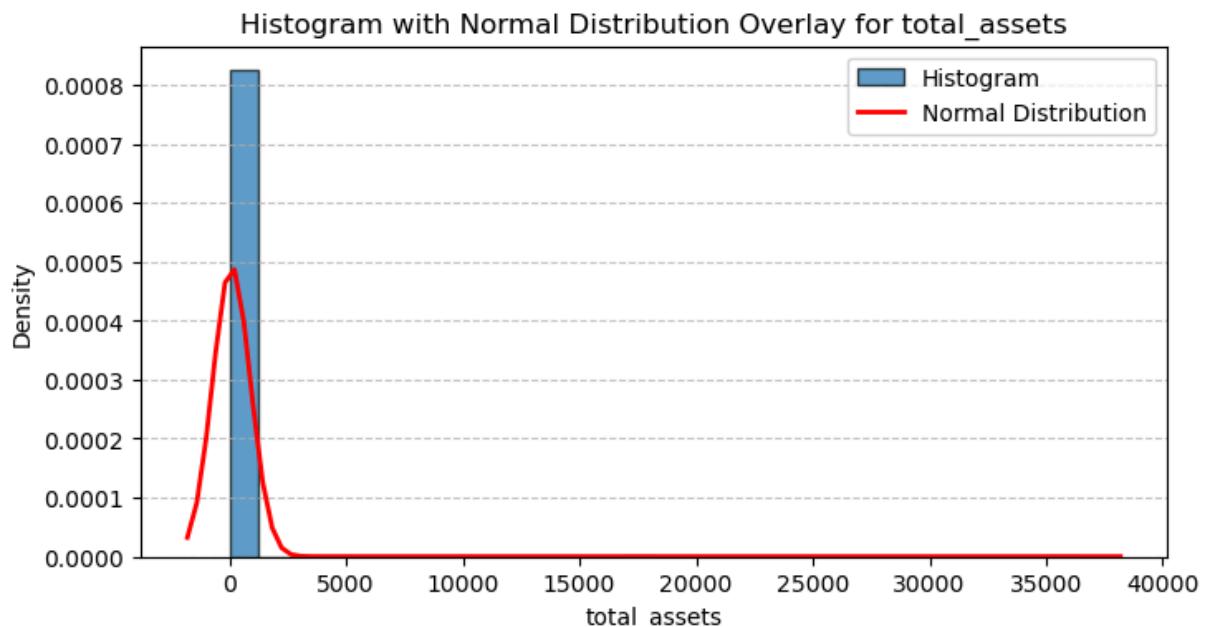


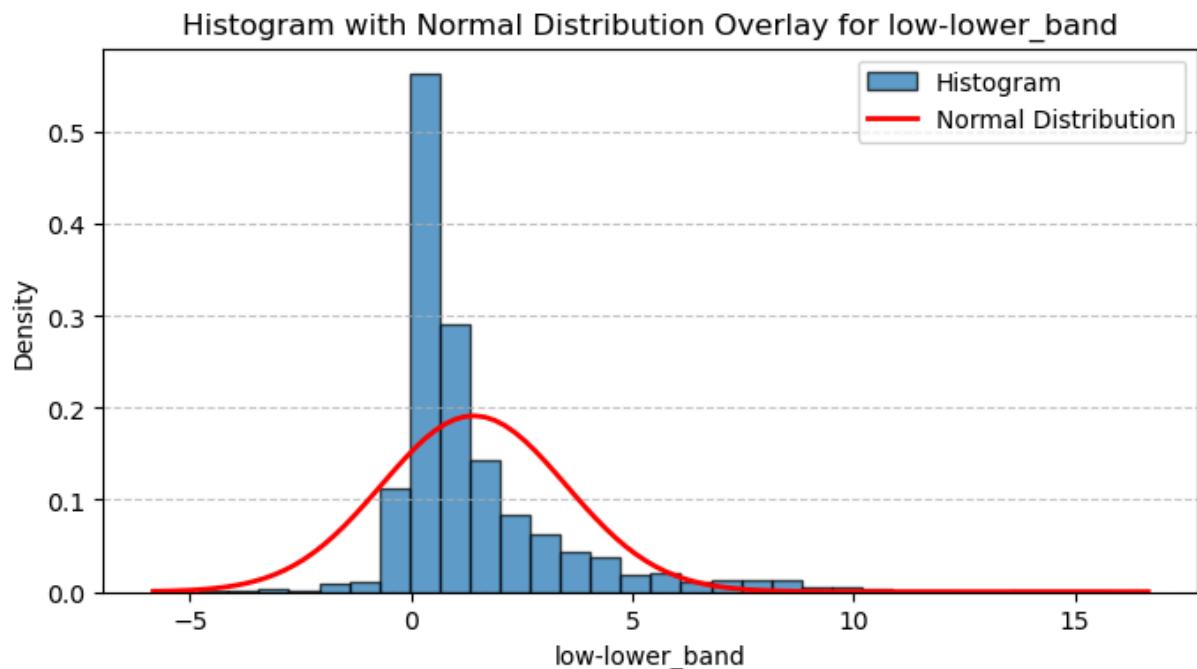
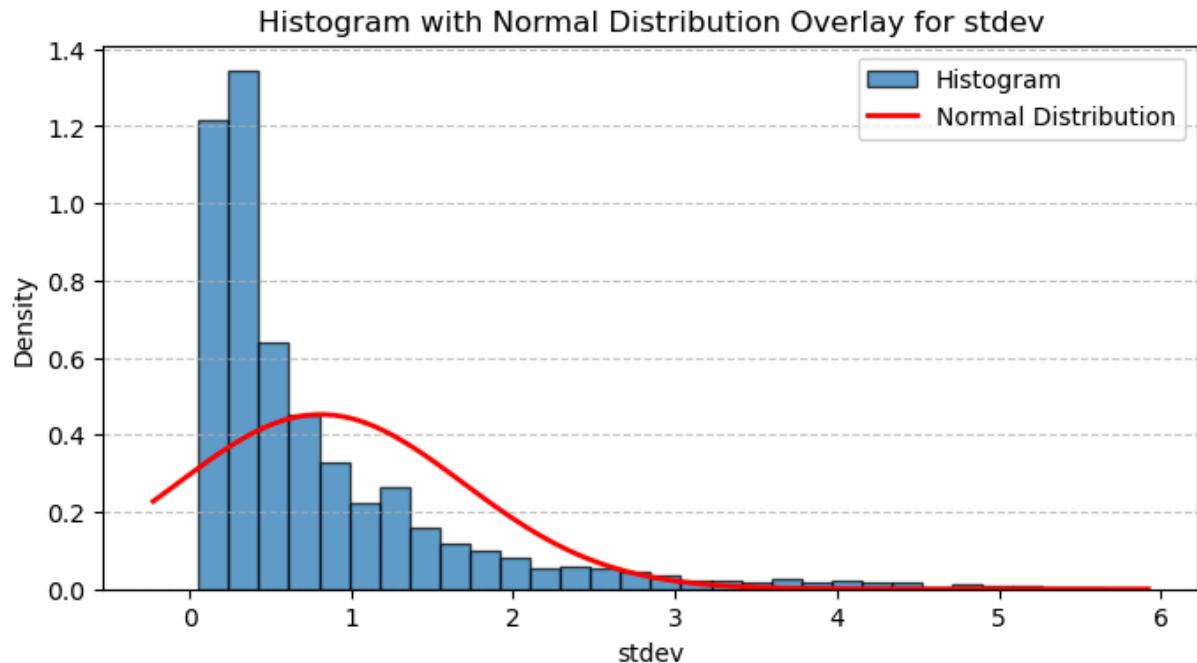


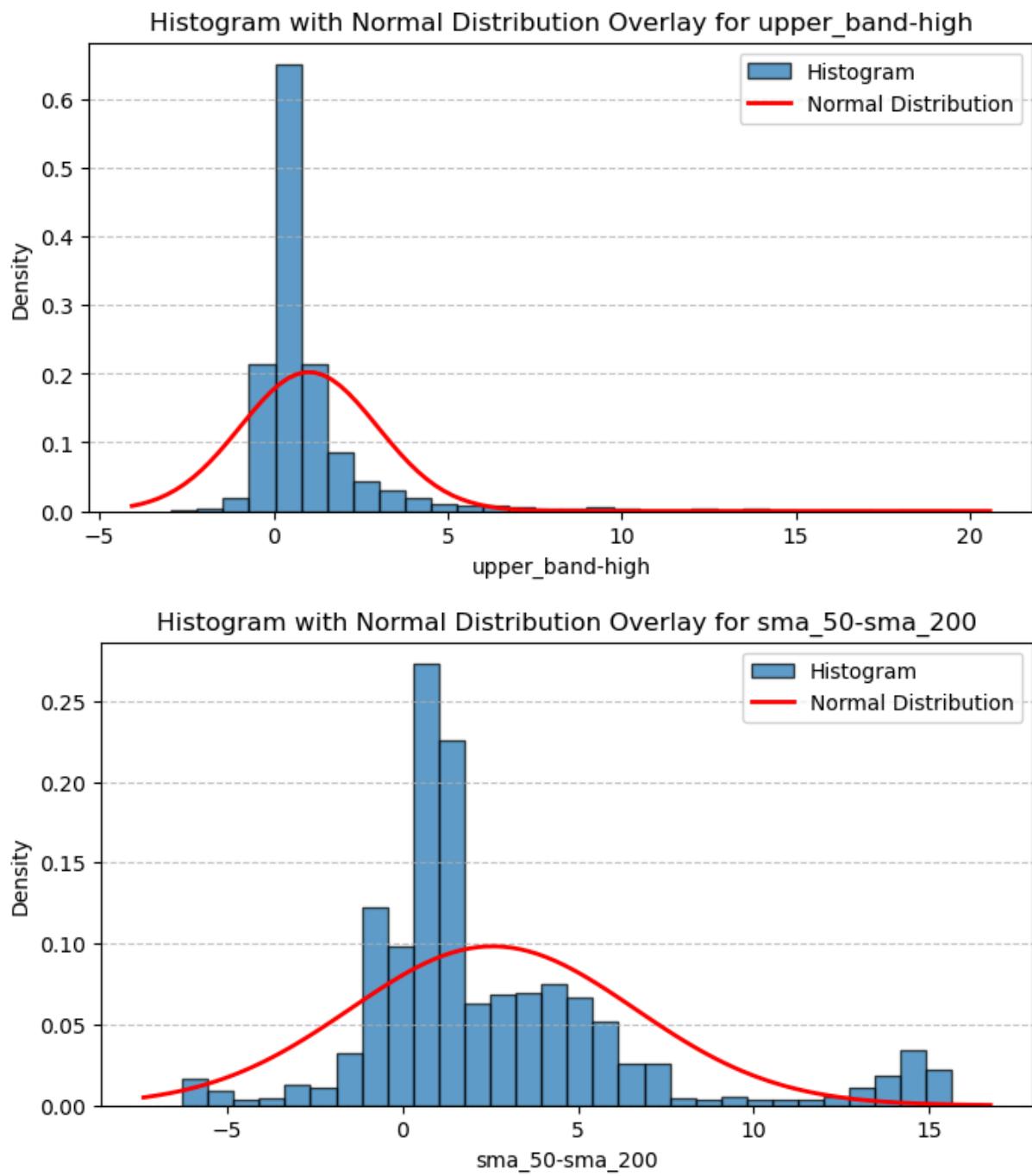


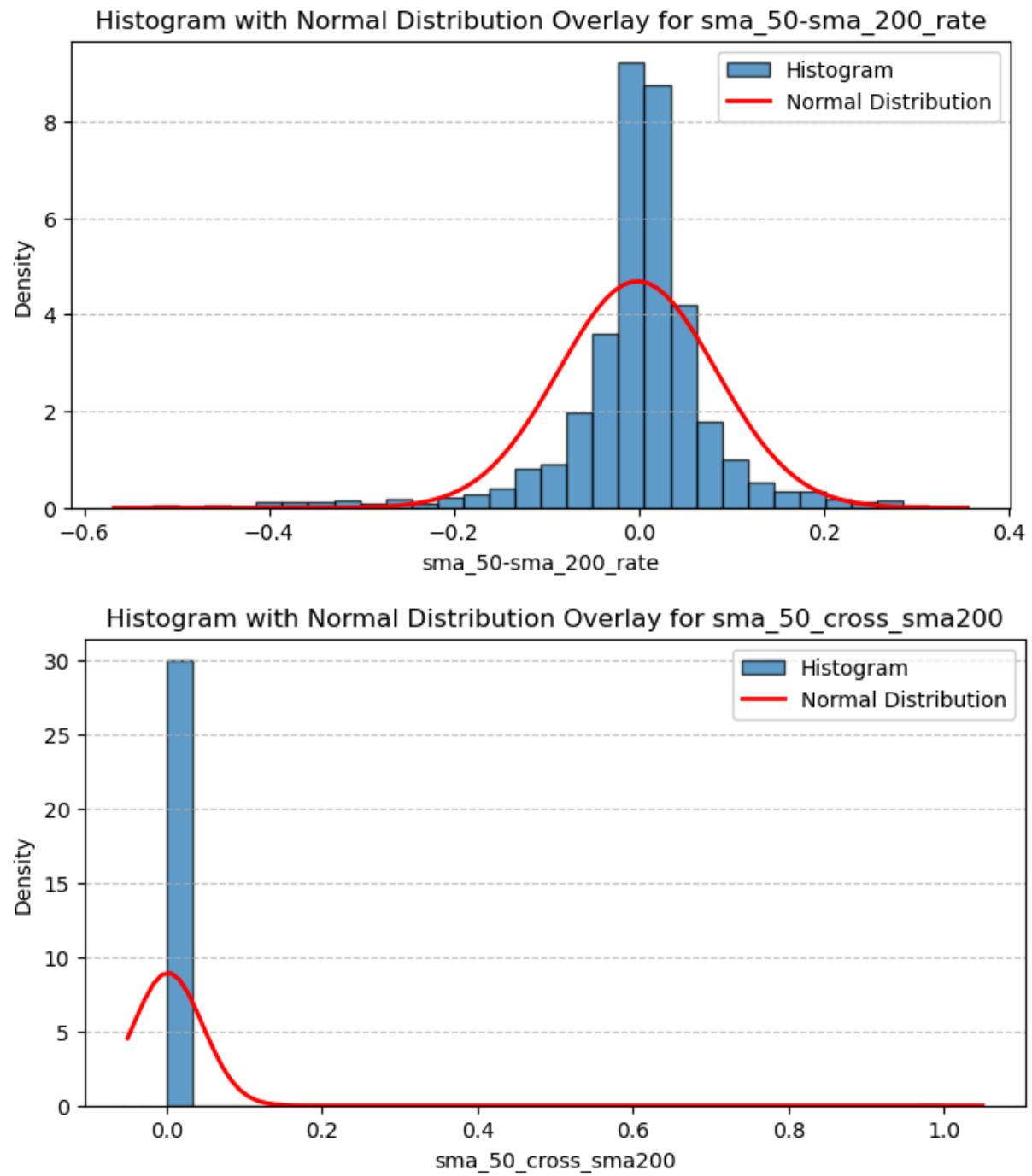


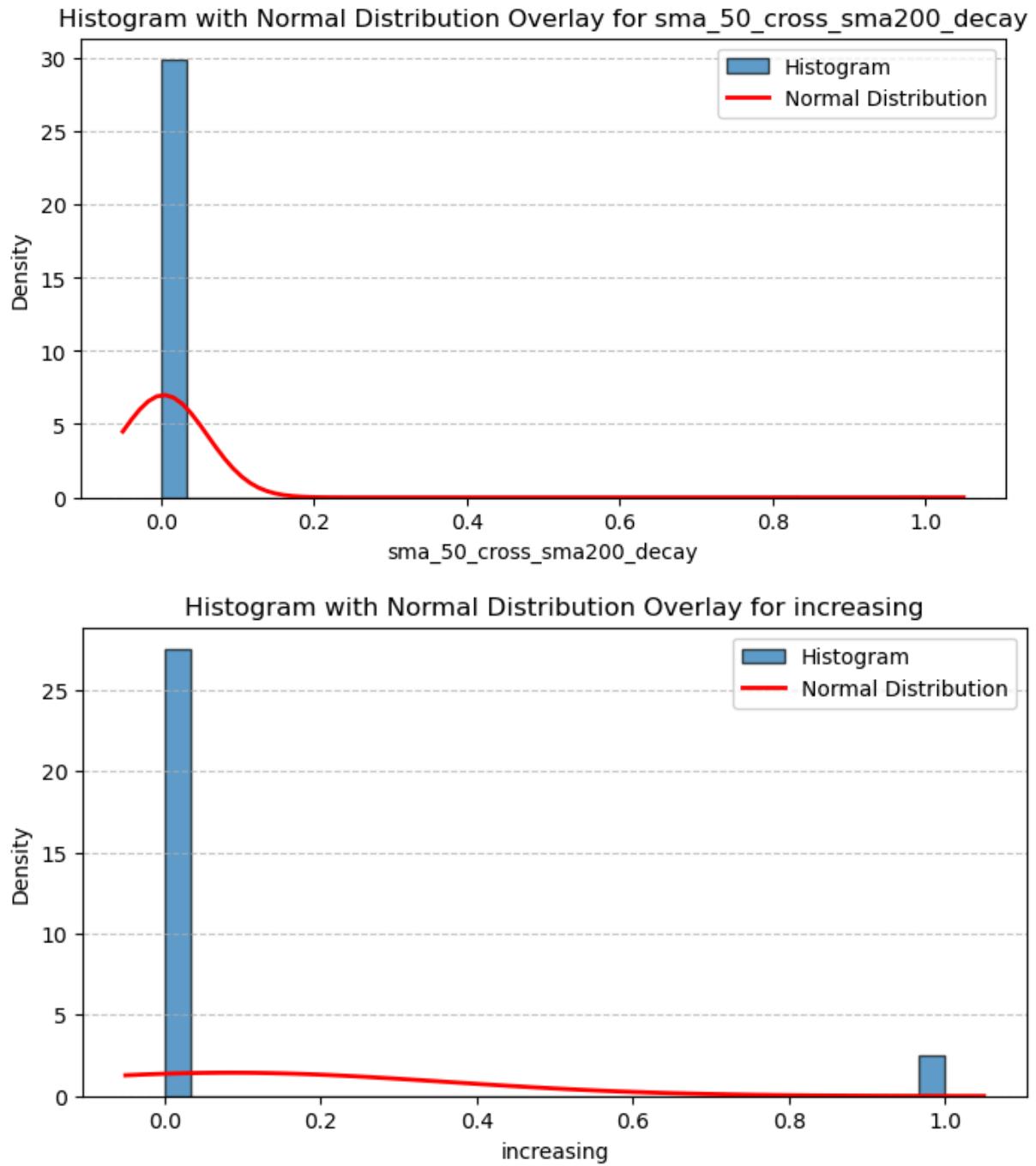


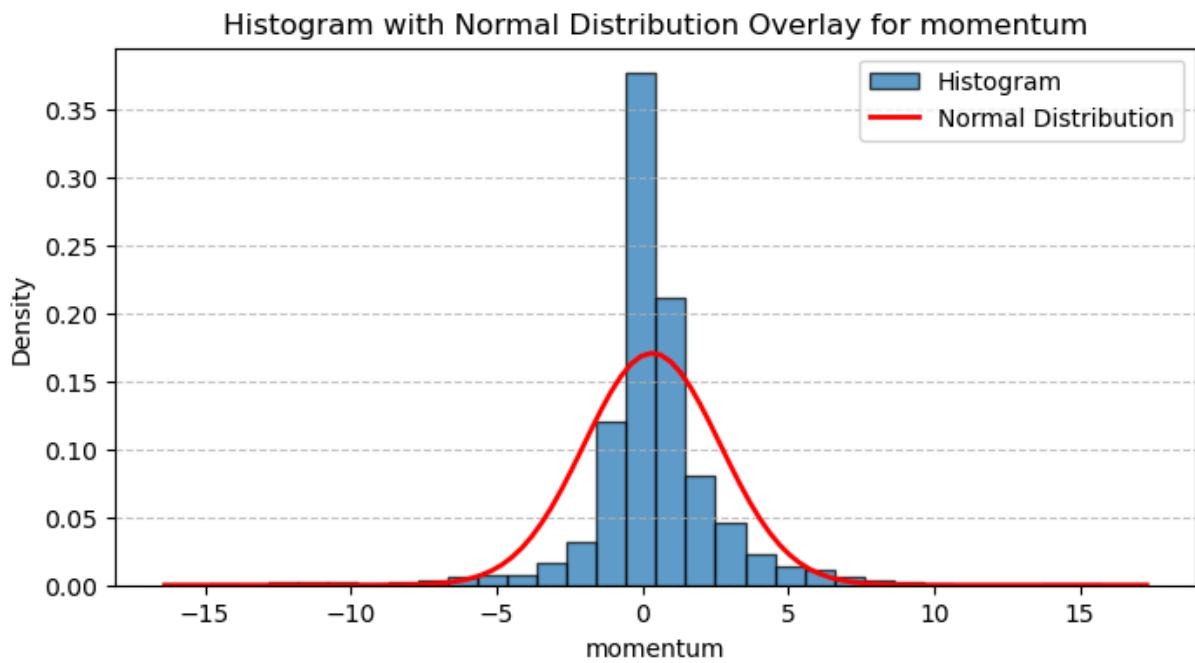
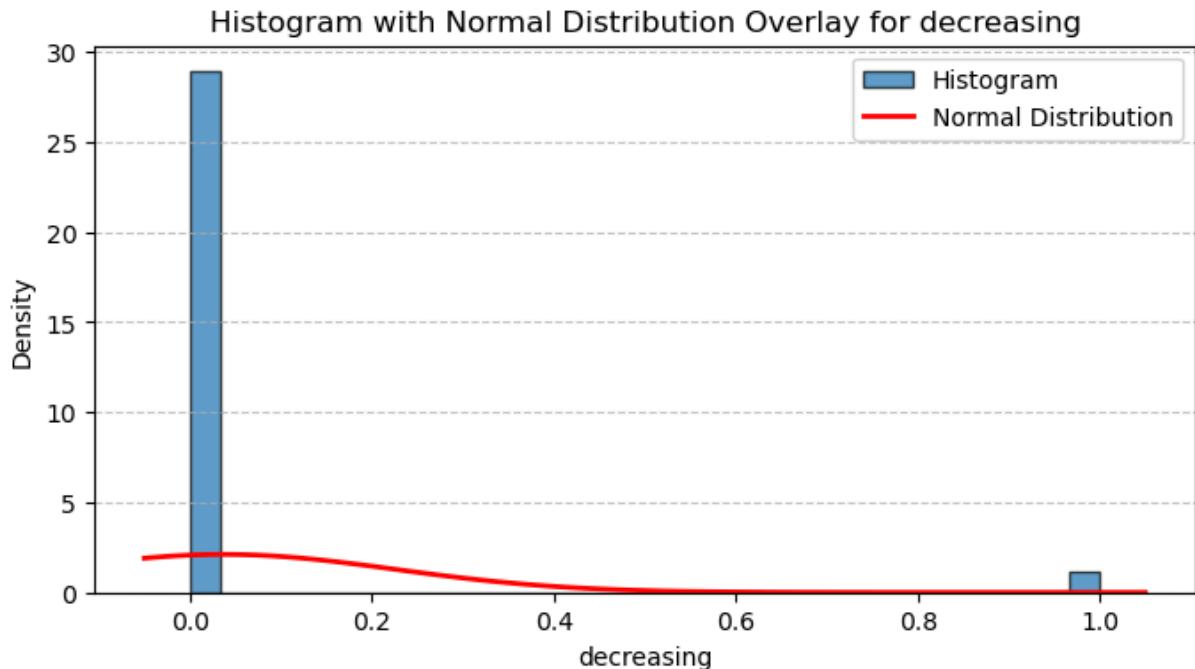




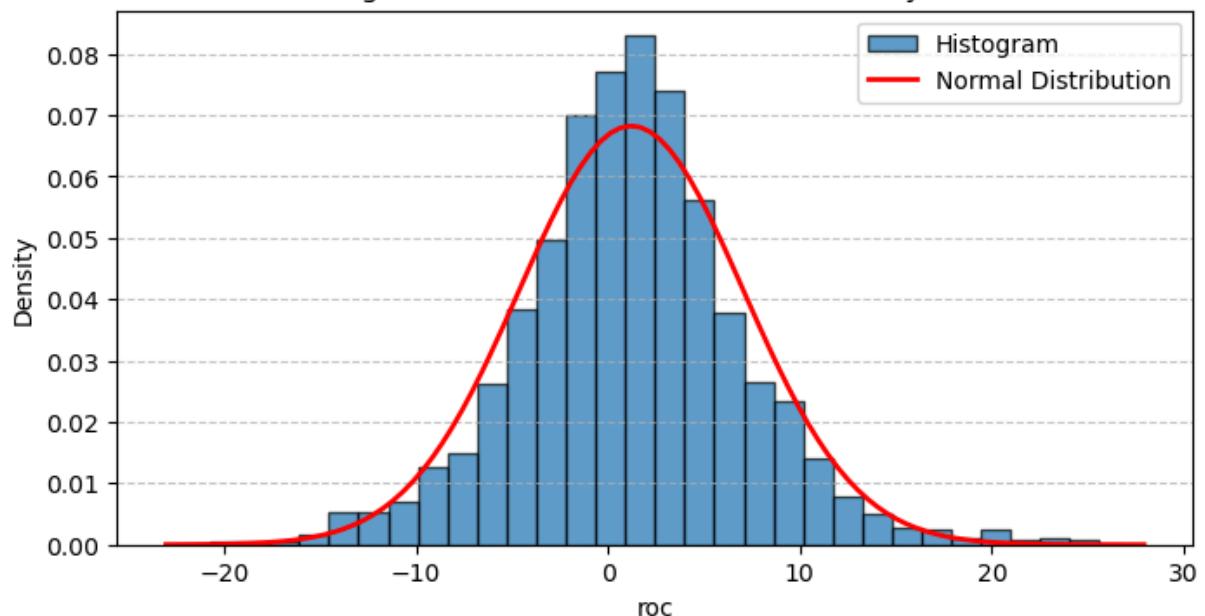




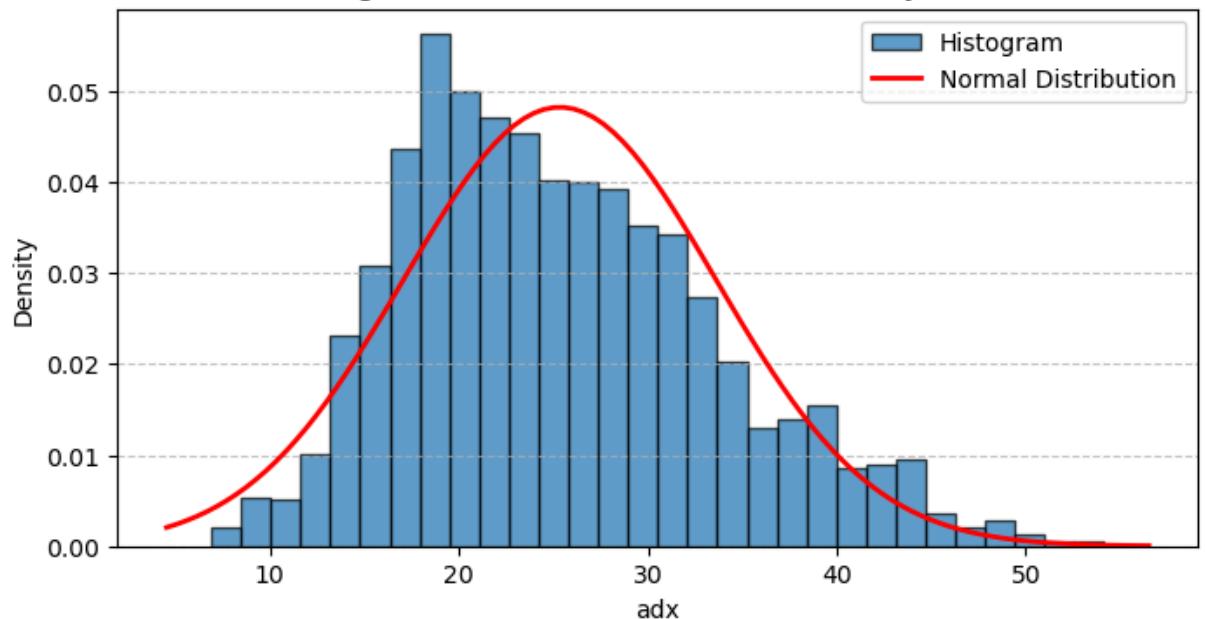


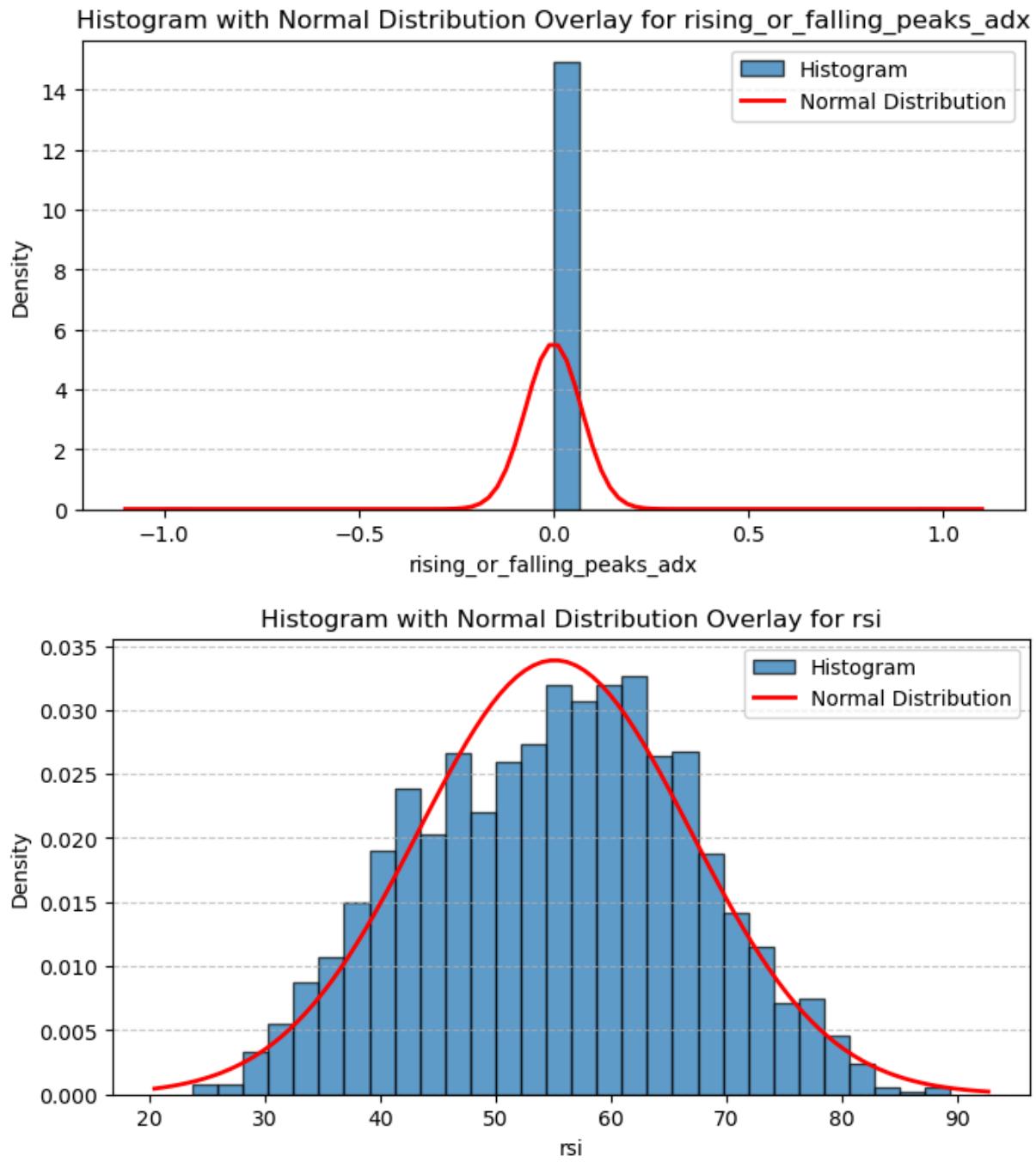


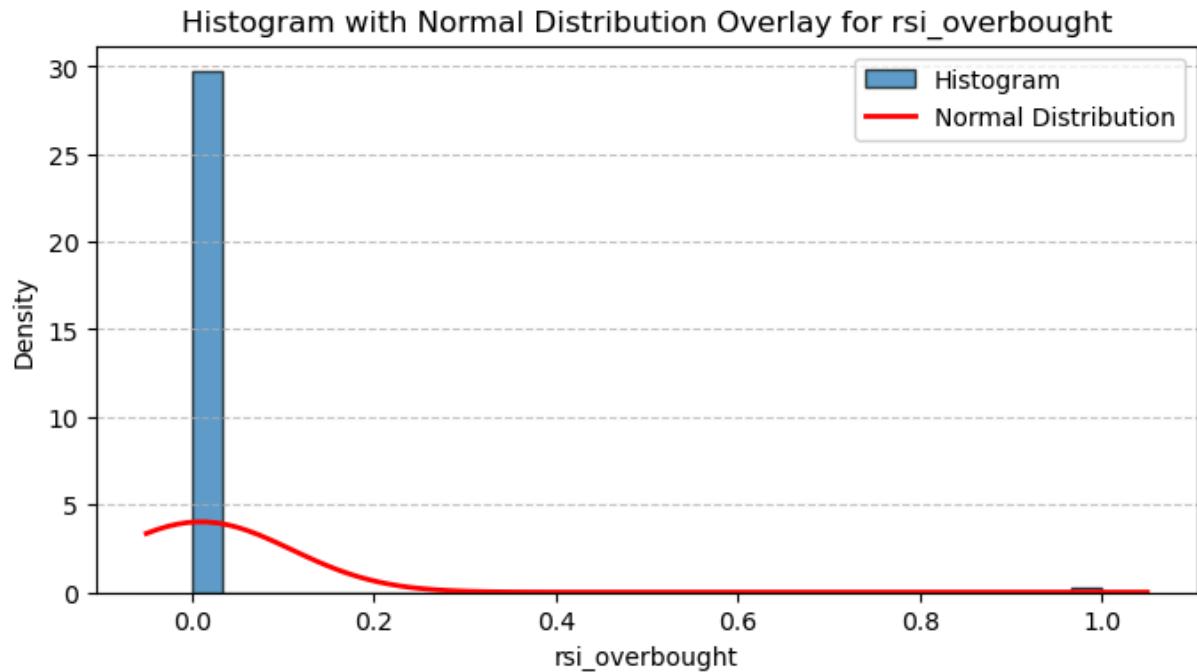
Histogram with Normal Distribution Overlay for roc



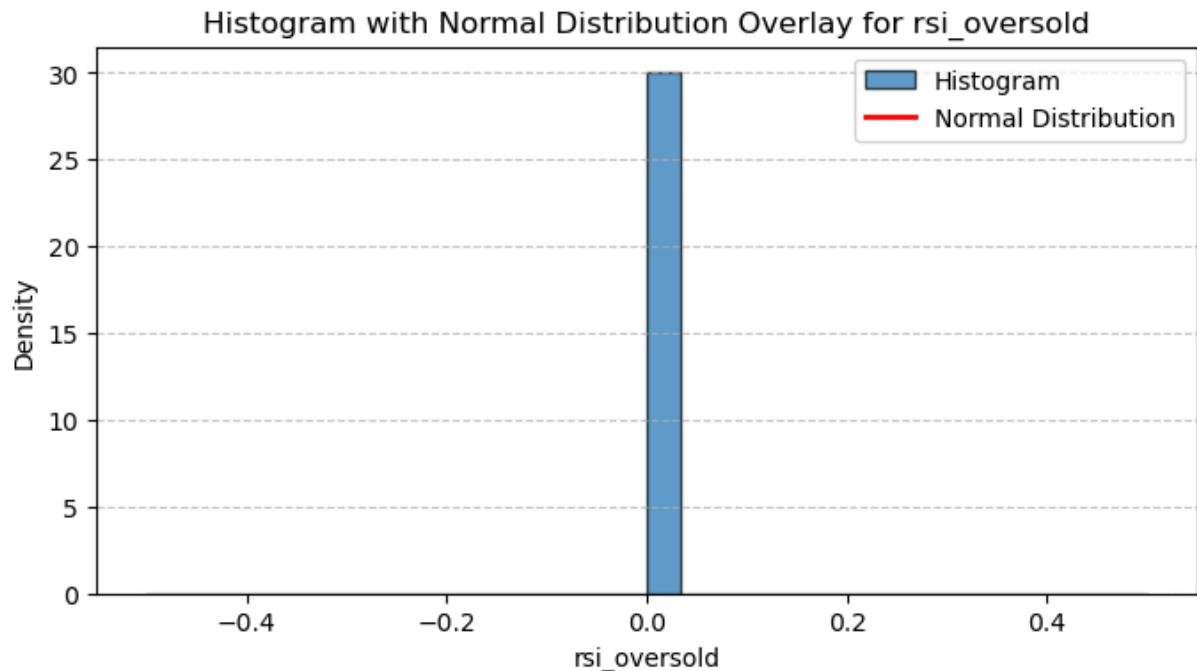
Histogram with Normal Distribution Overlay for adx

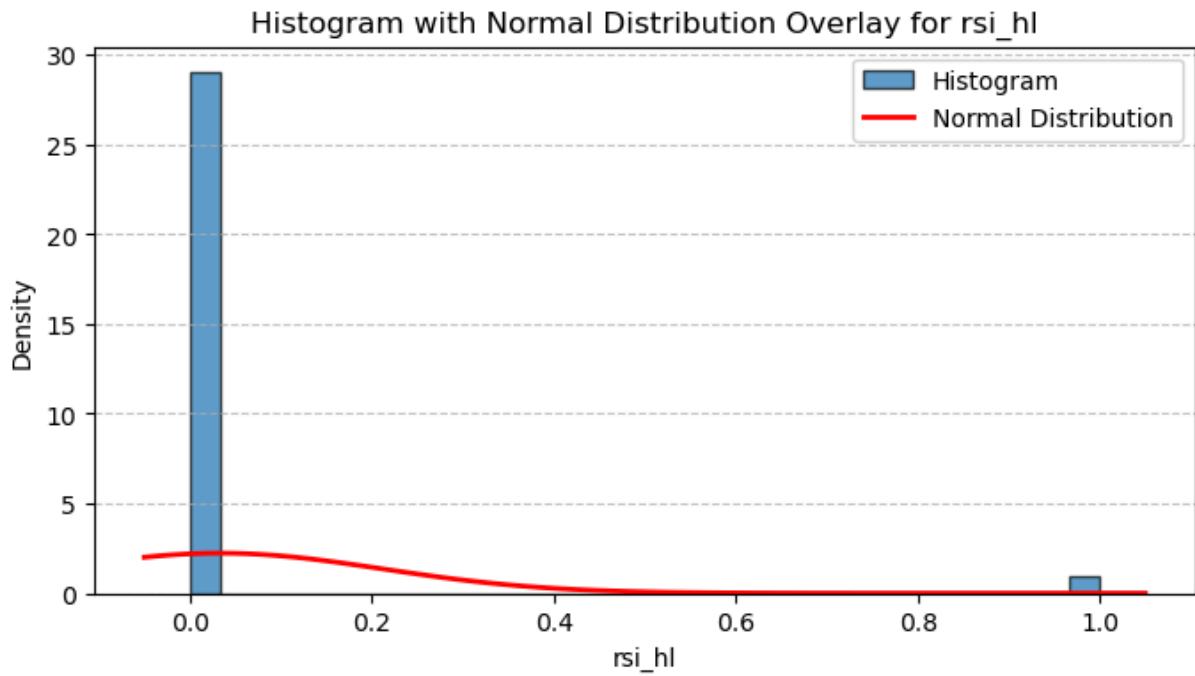
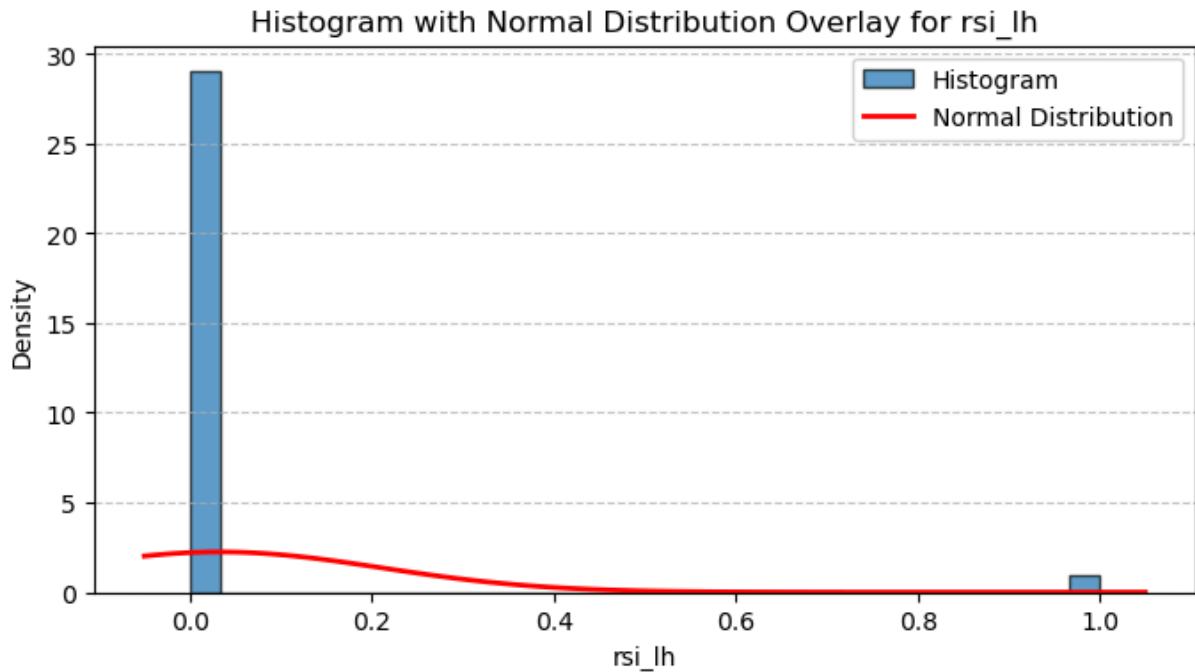


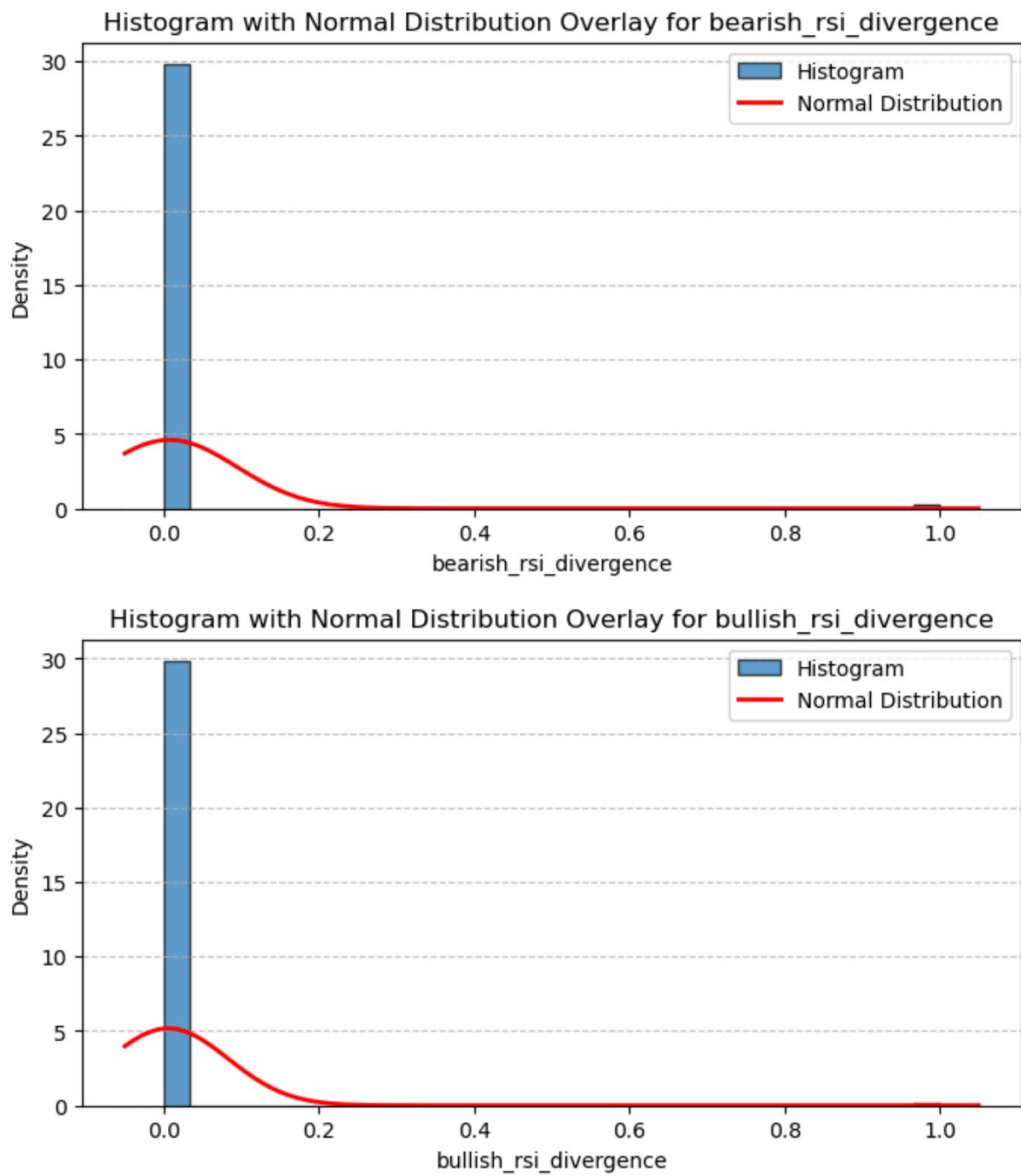


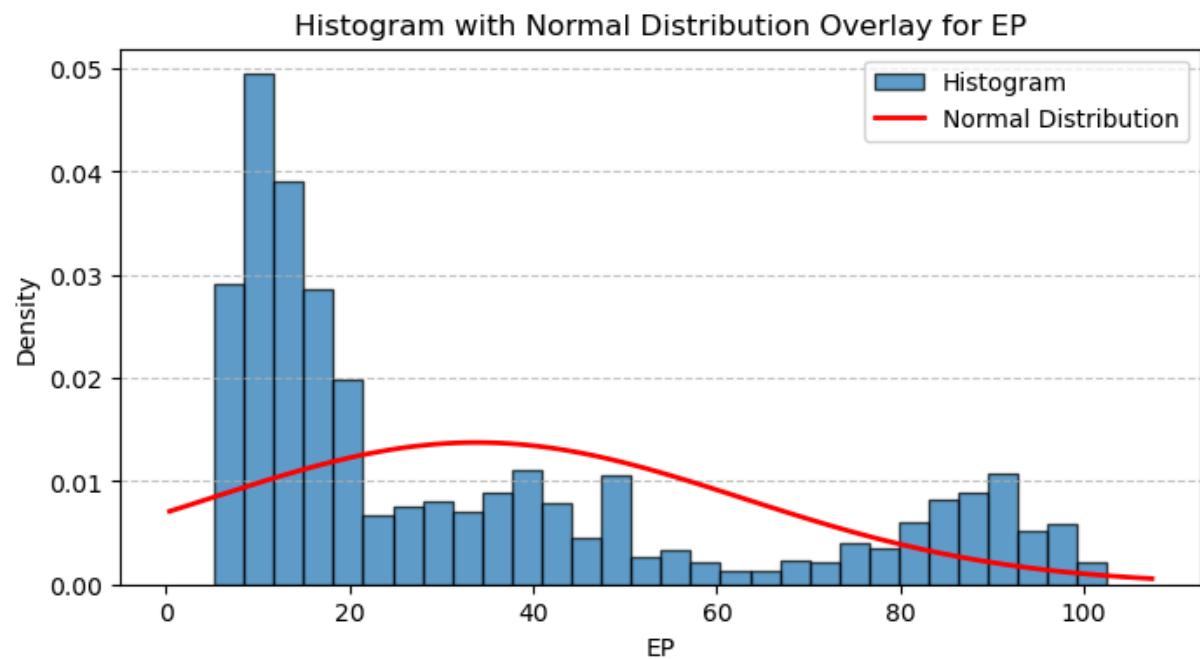
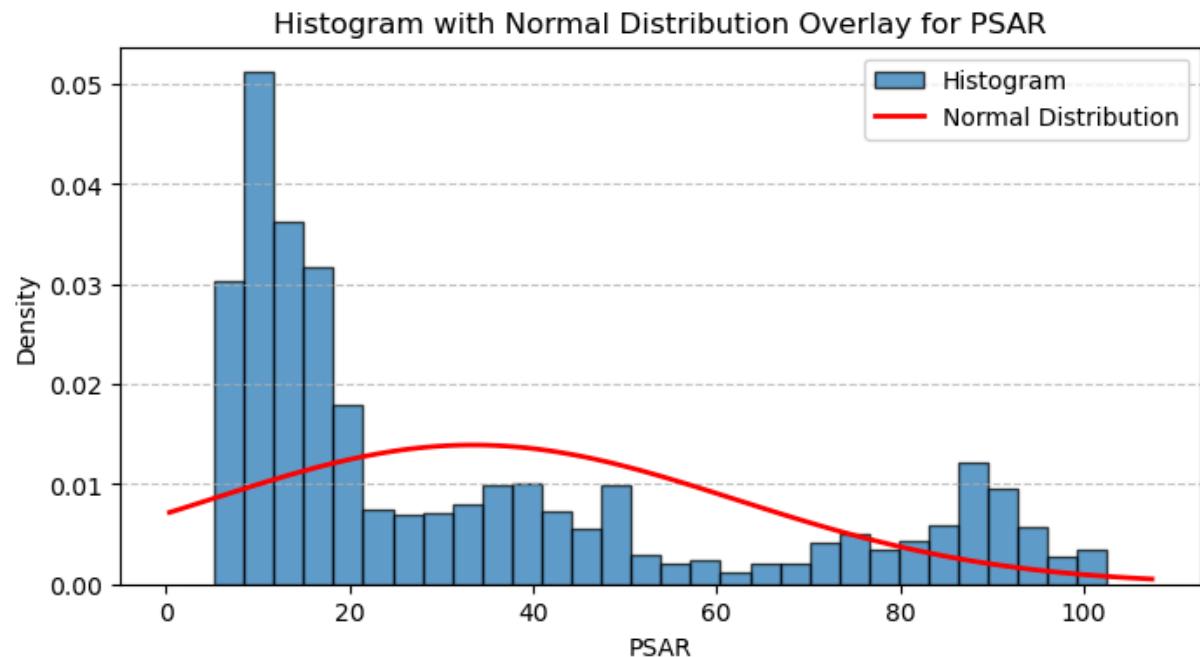


```
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/scipy/stats/_distn_infrastructure.py:1986: RuntimeWarning: divide by zero encountered in divide
x = np.asarray((x - loc)/scale, dtype=dtyp)
```

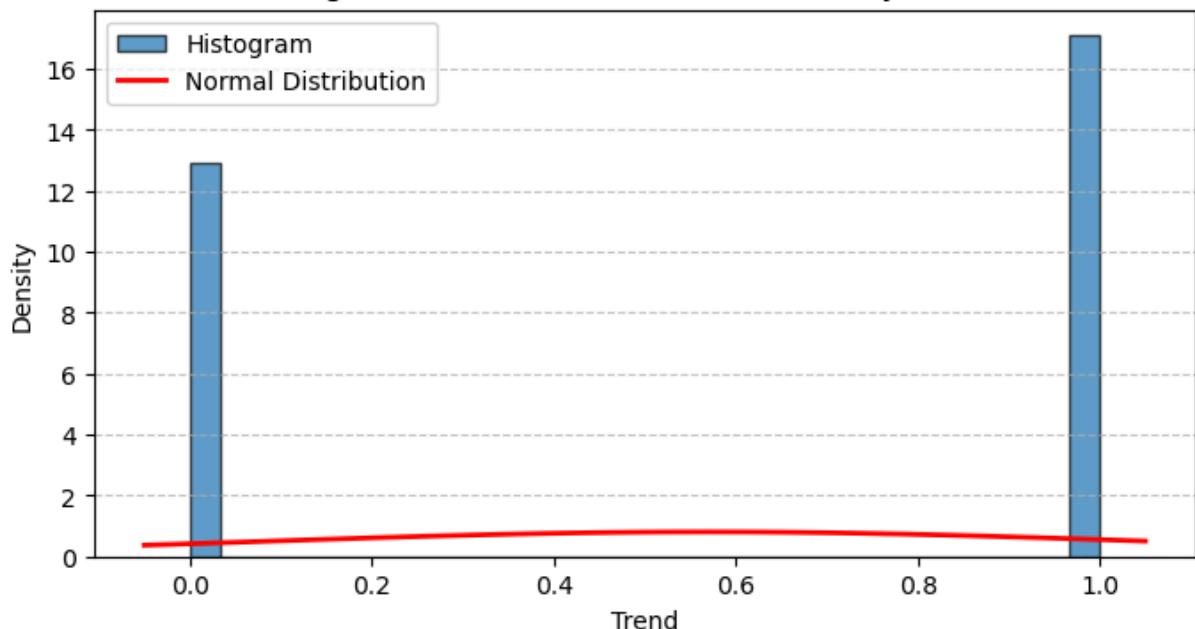




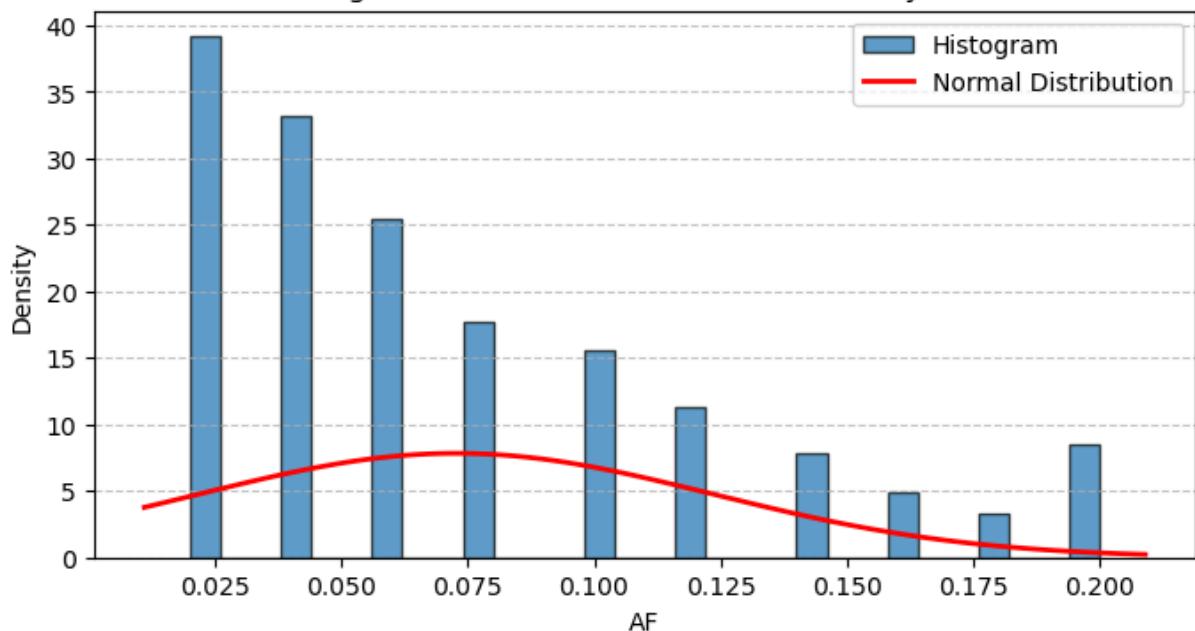




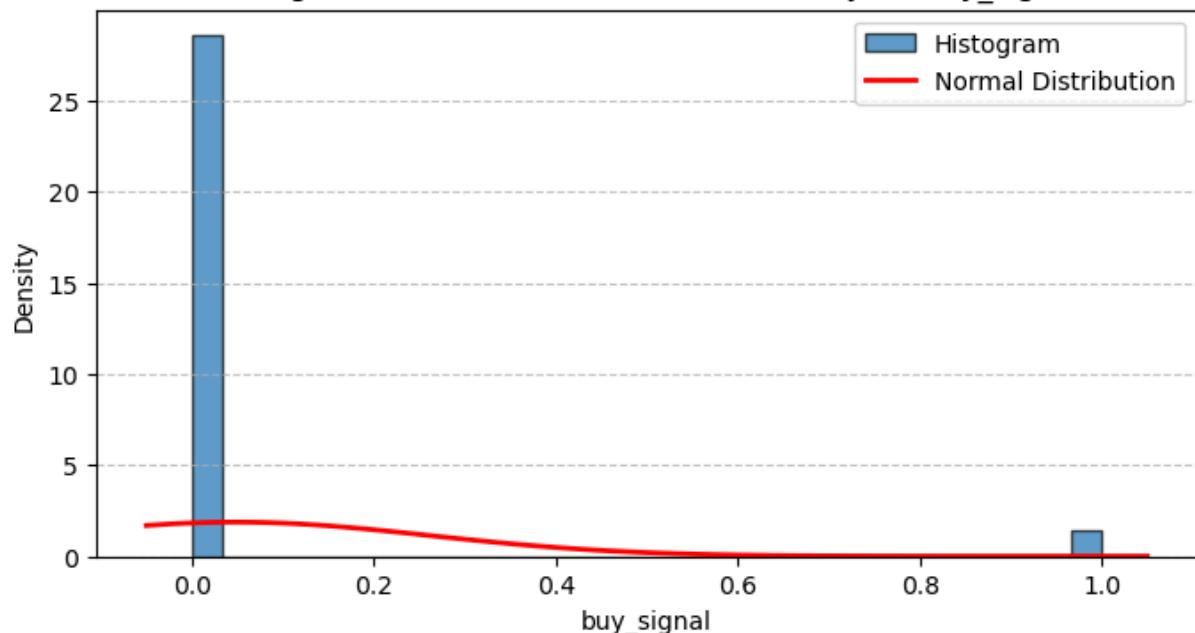
Histogram with Normal Distribution Overlay for Trend



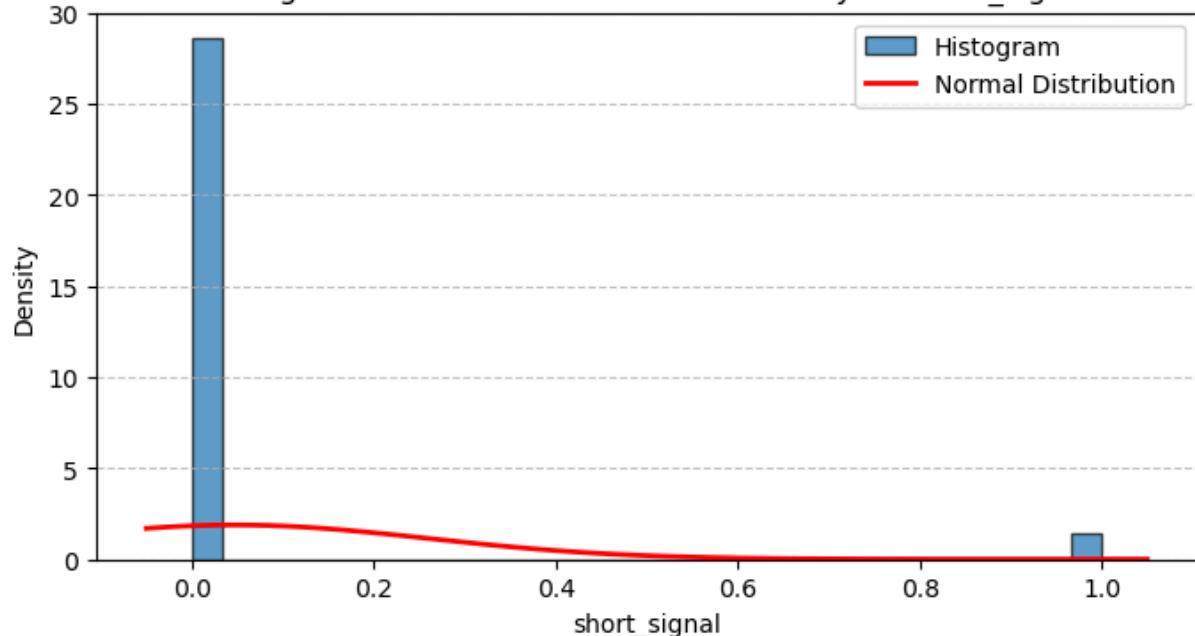
Histogram with Normal Distribution Overlay for AF

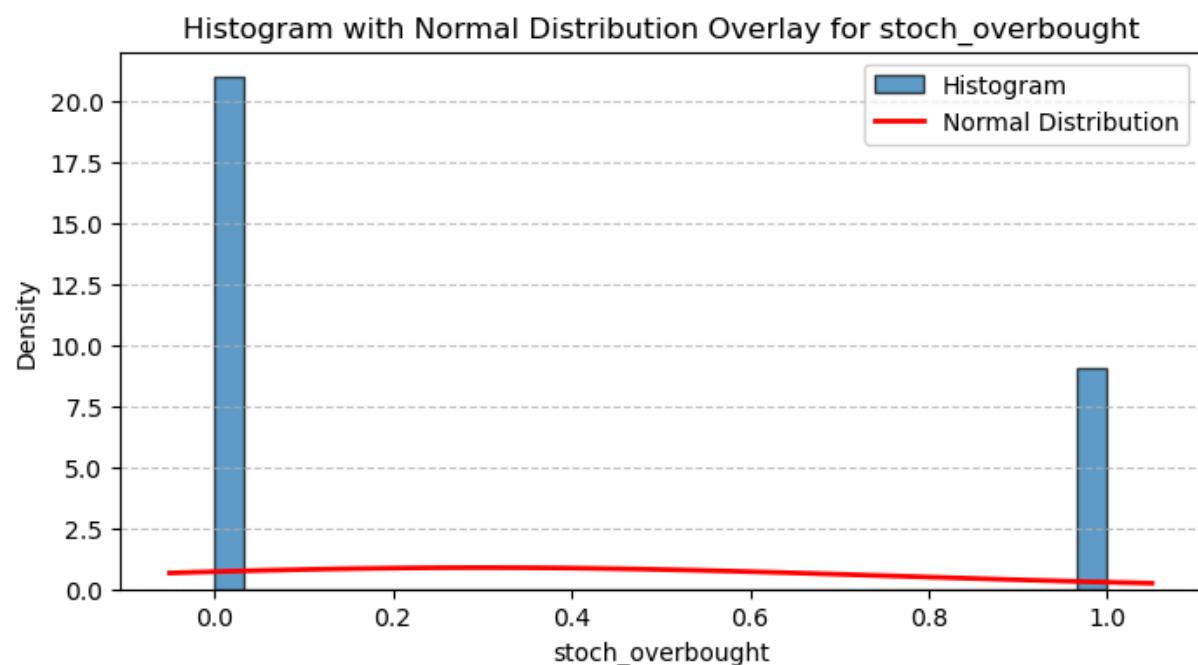
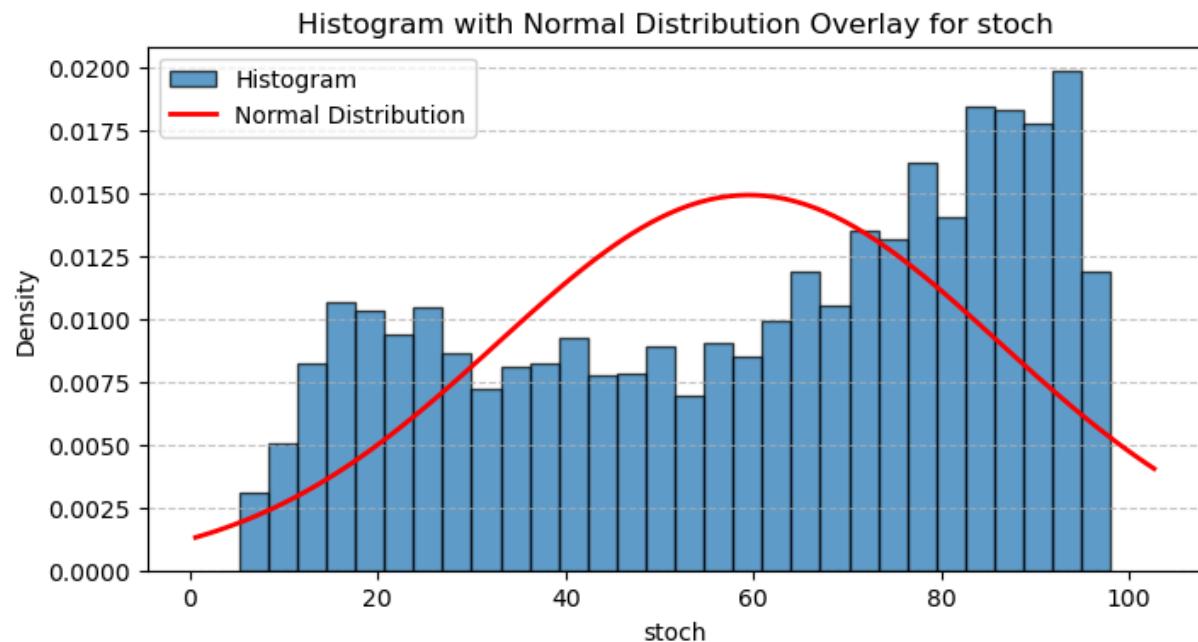


Histogram with Normal Distribution Overlay for buy\_signal

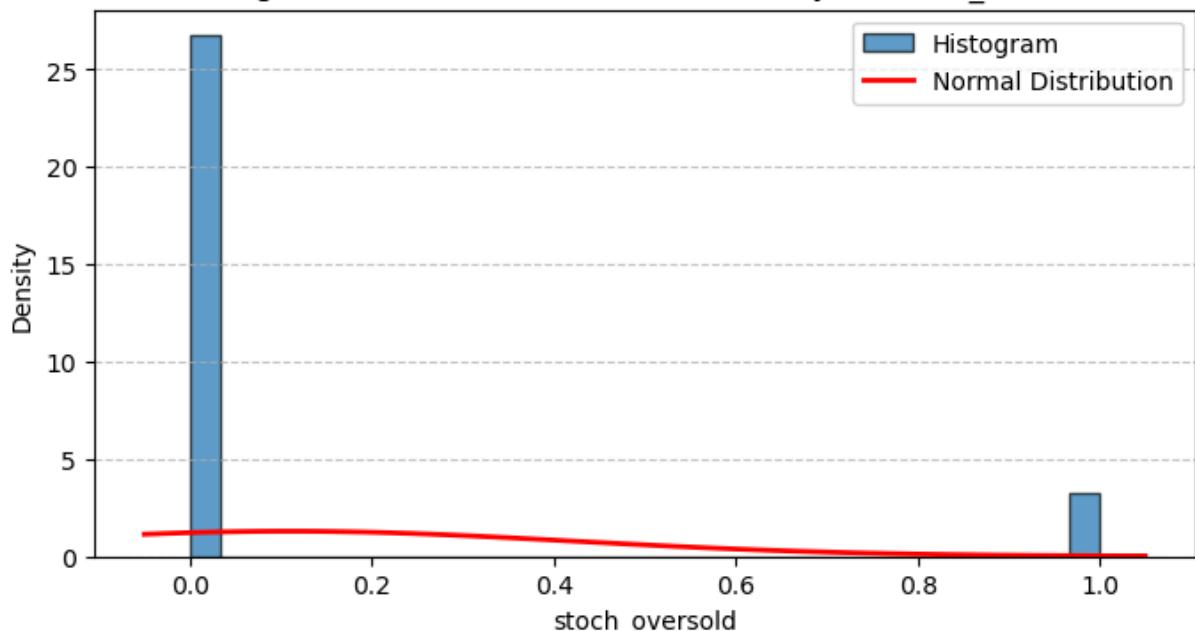


Histogram with Normal Distribution Overlay for short\_signal

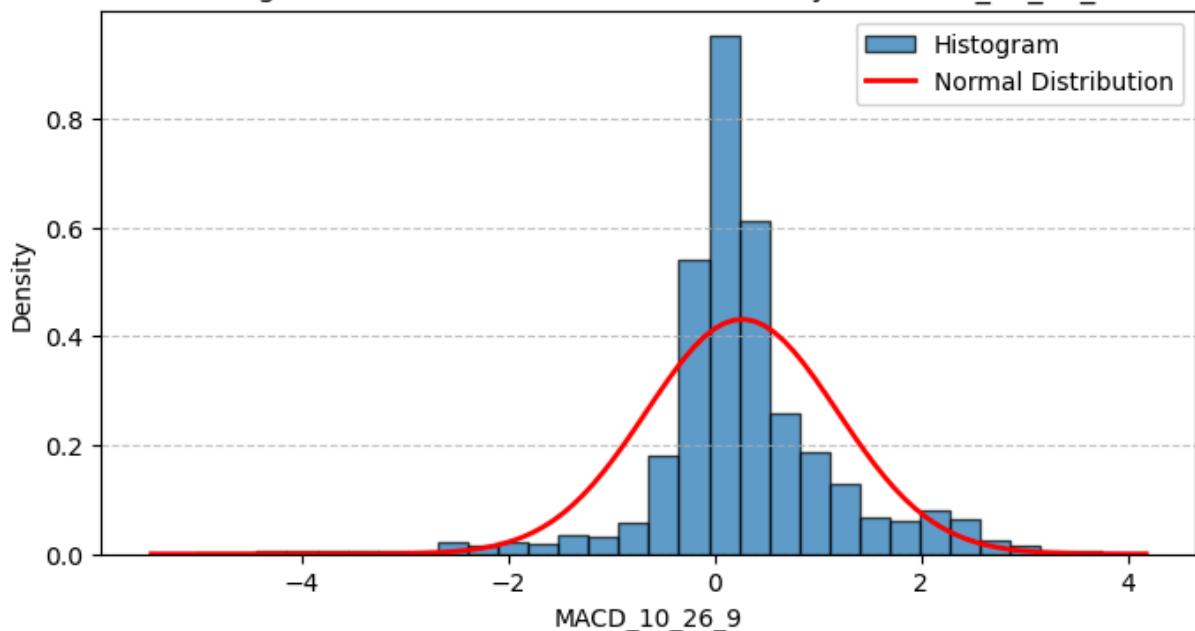




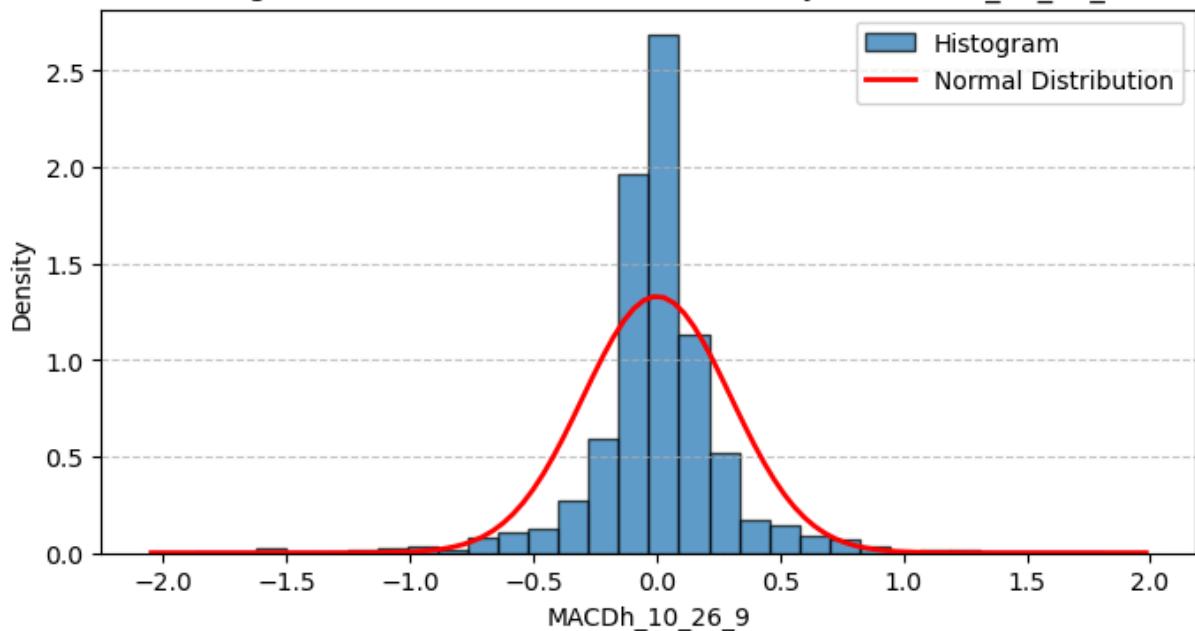
Histogram with Normal Distribution Overlay for stoch\_oversold



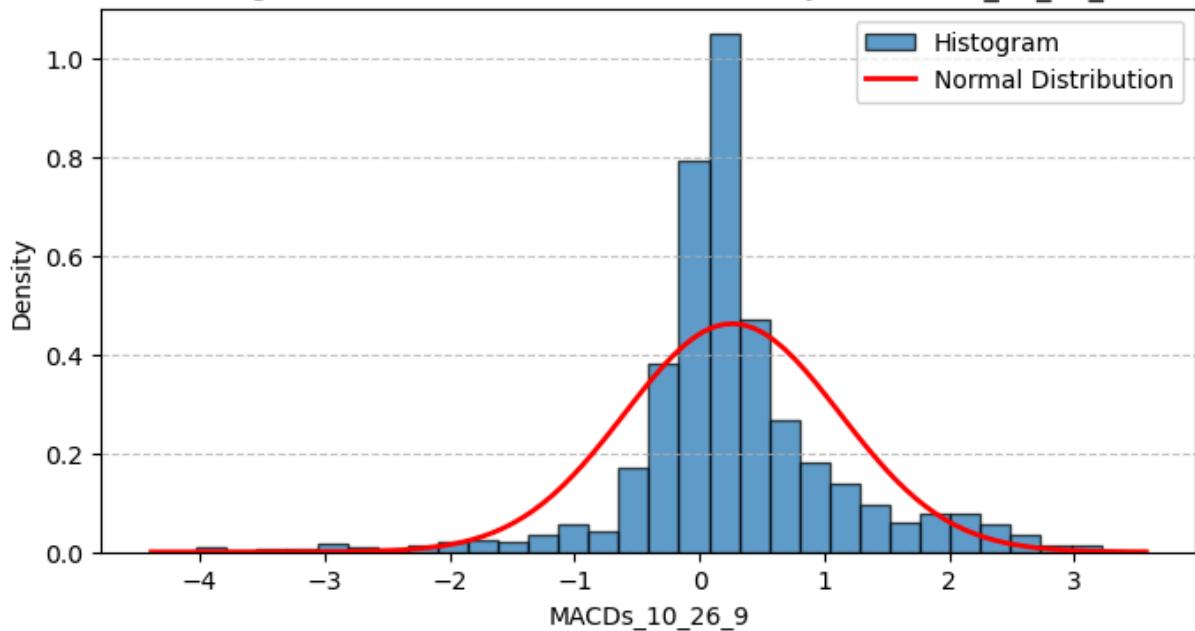
Histogram with Normal Distribution Overlay for MACD\_10\_26\_9

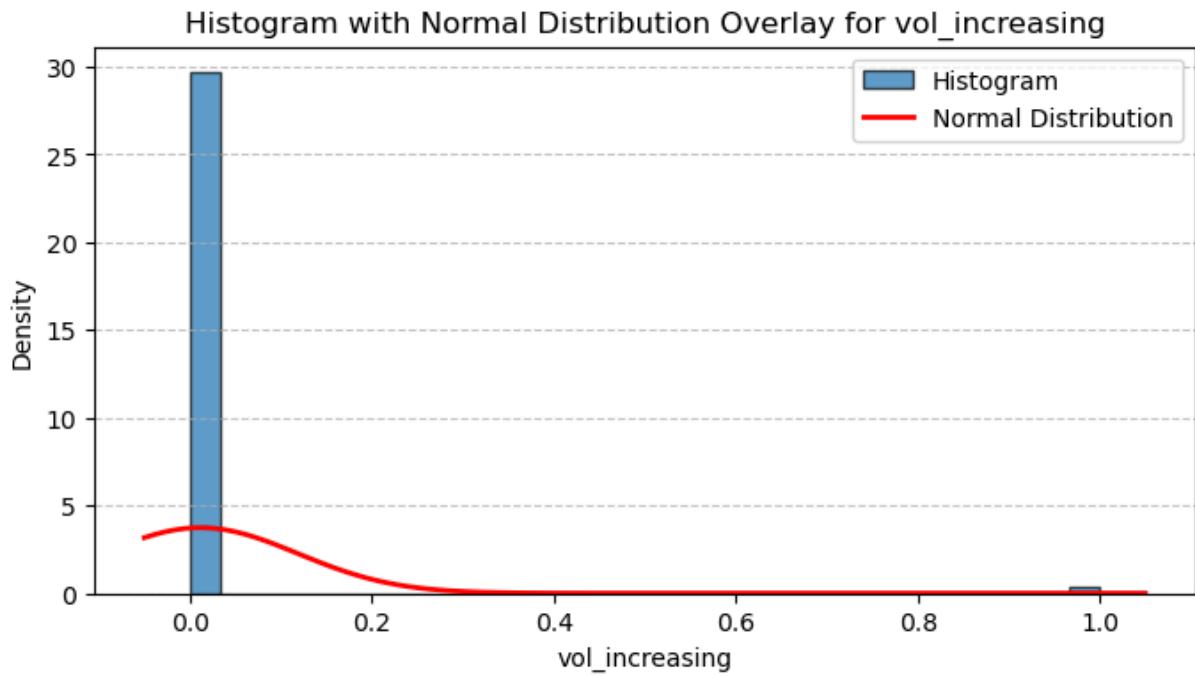
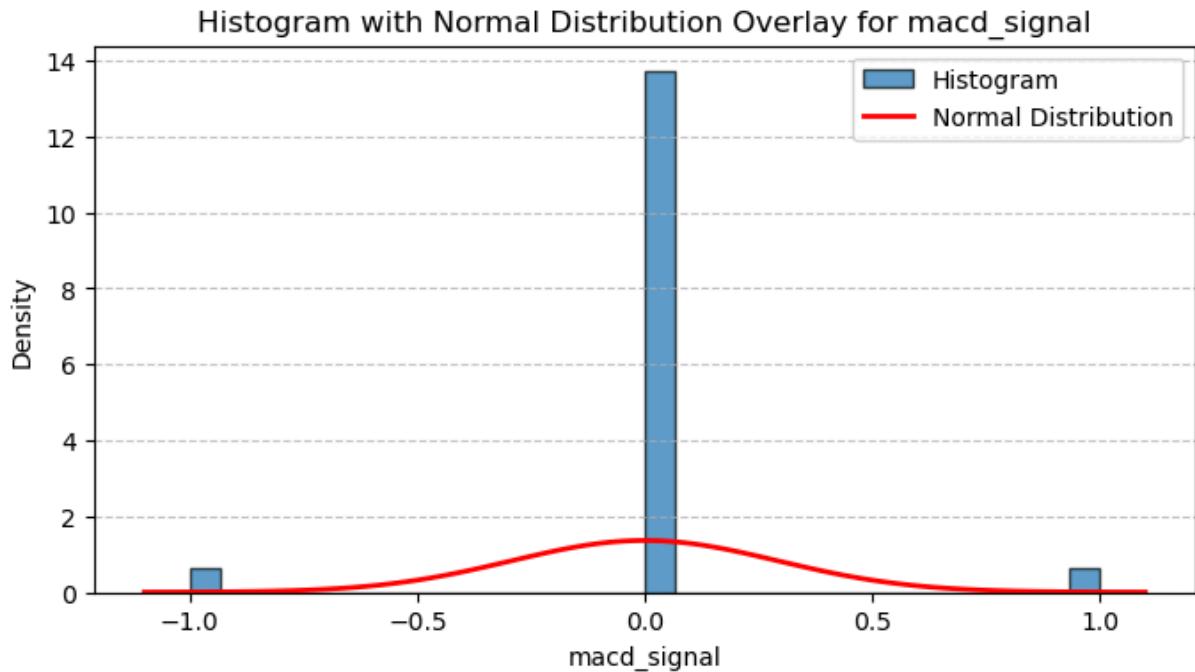


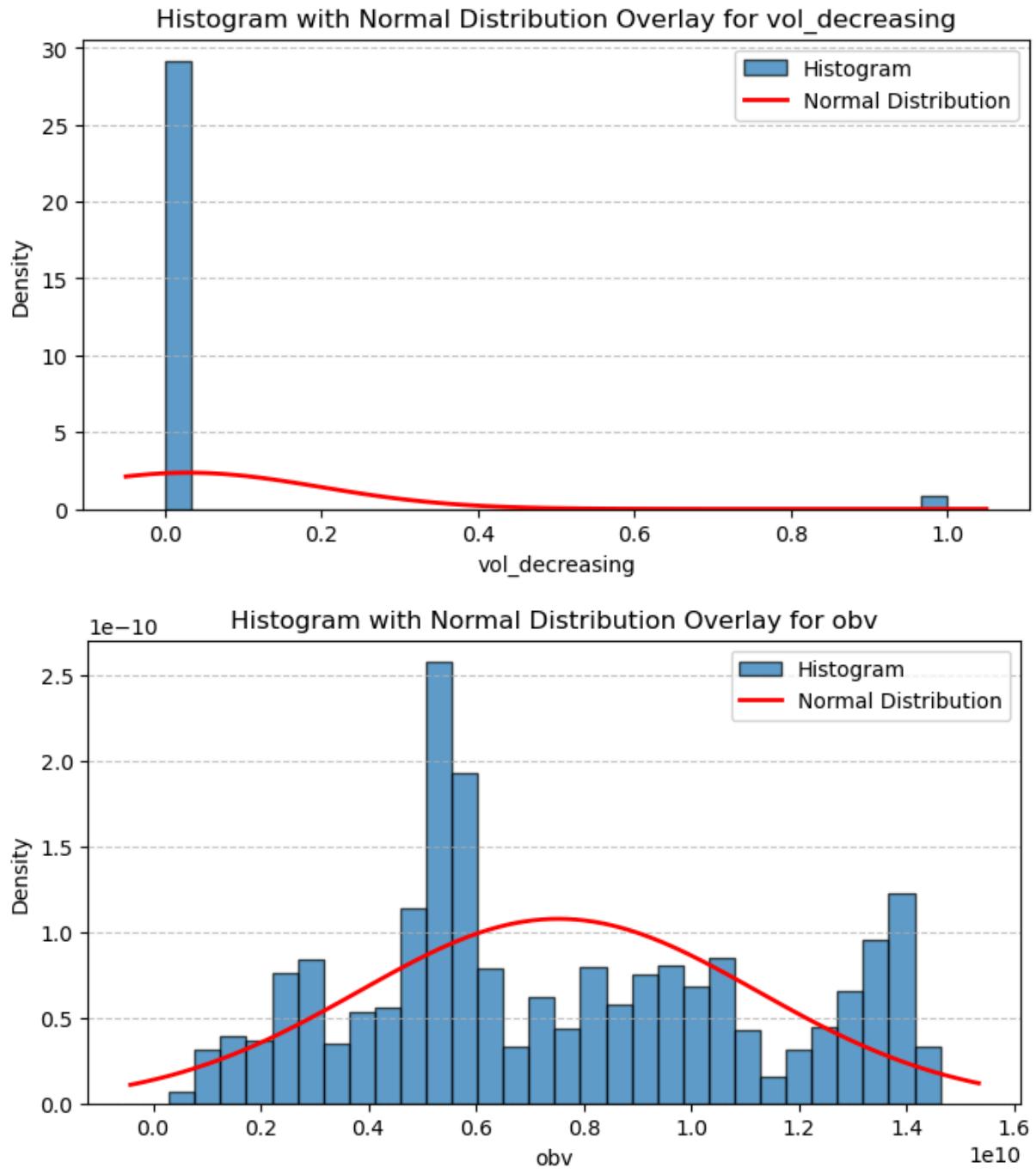
Histogram with Normal Distribution Overlay for MACDh\_10\_26\_9

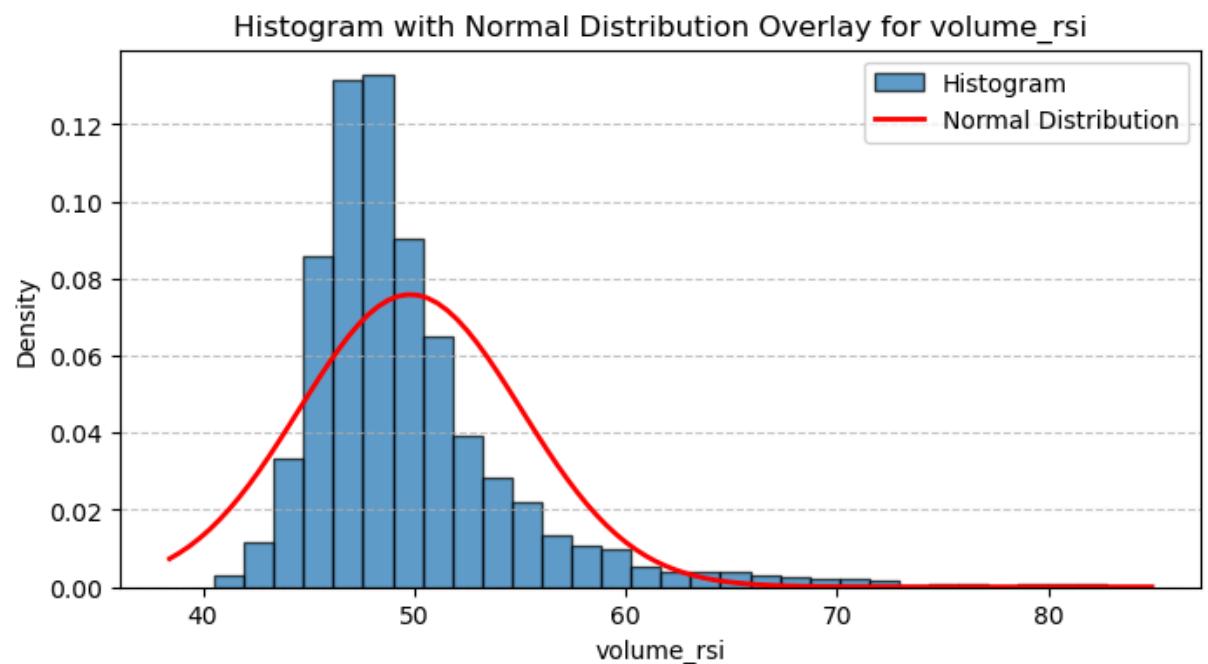
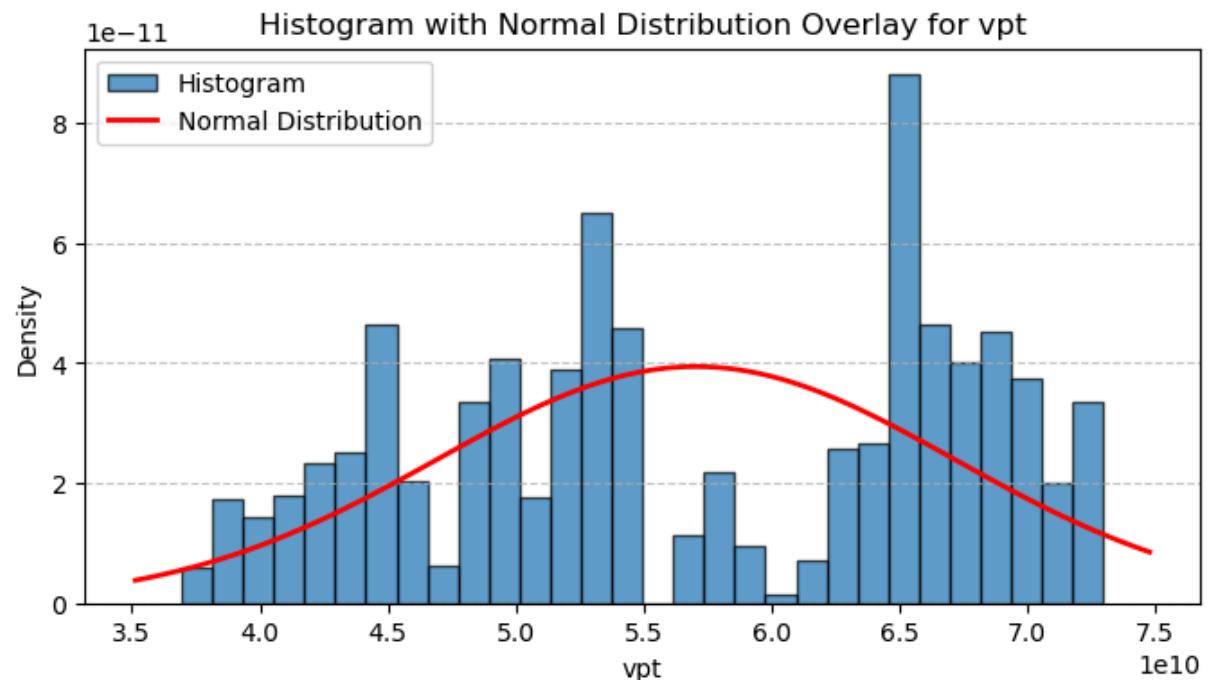


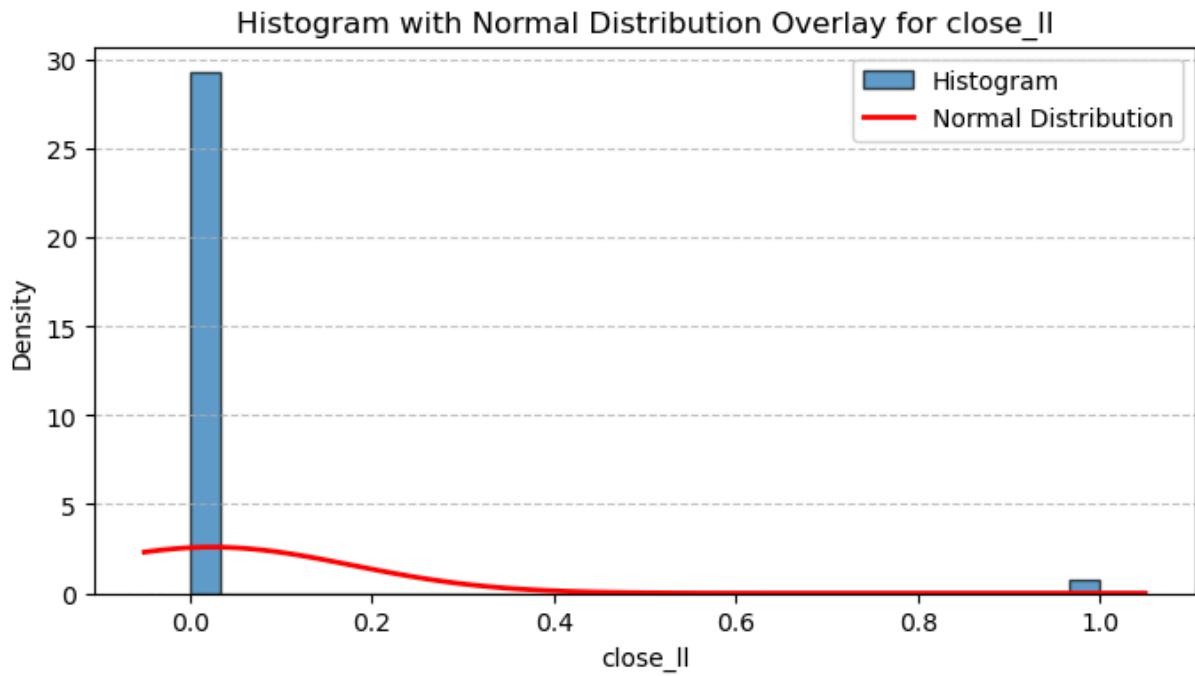
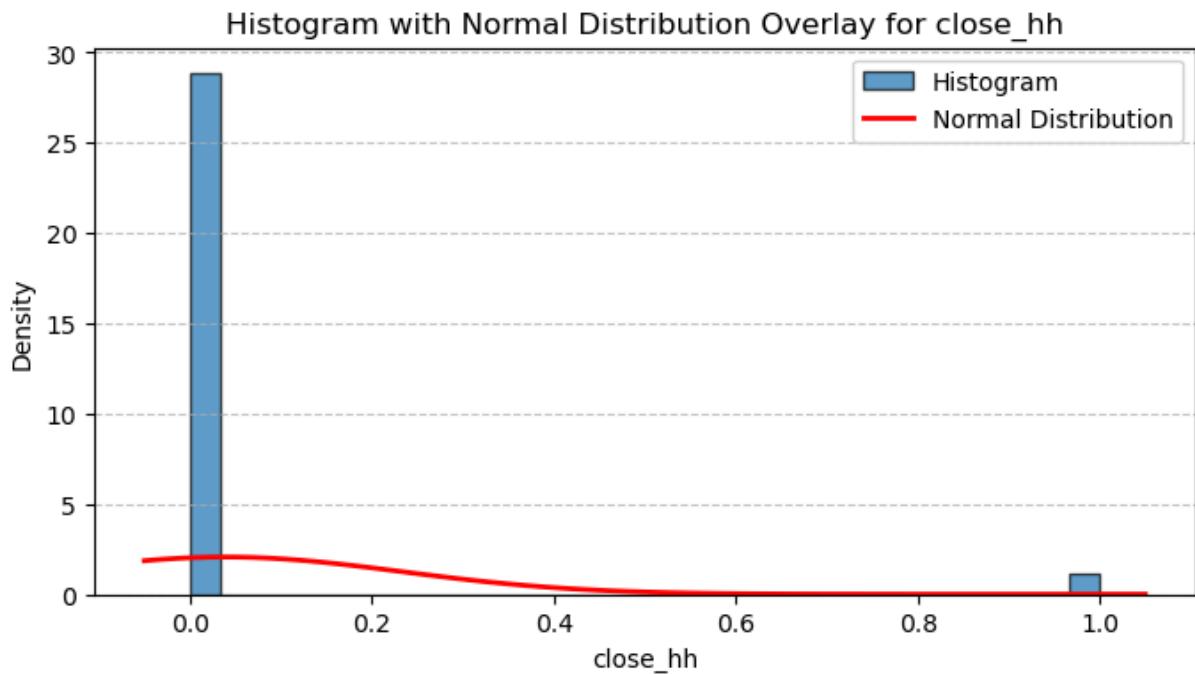
Histogram with Normal Distribution Overlay for MACDs\_10\_26\_9

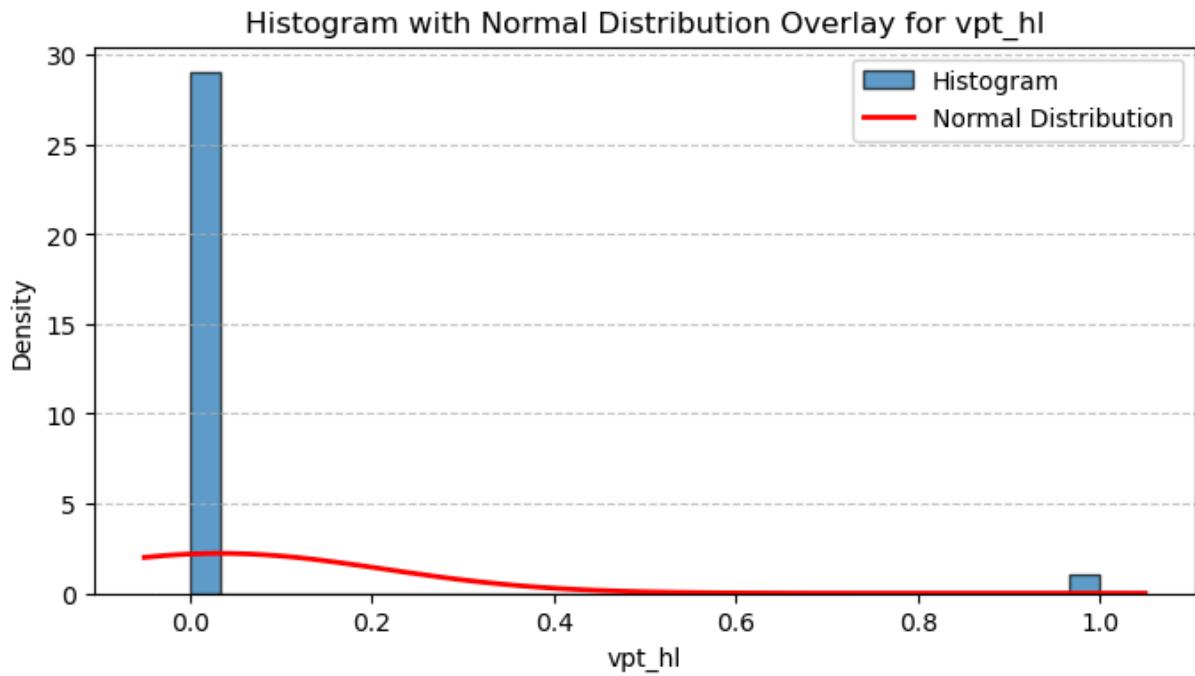
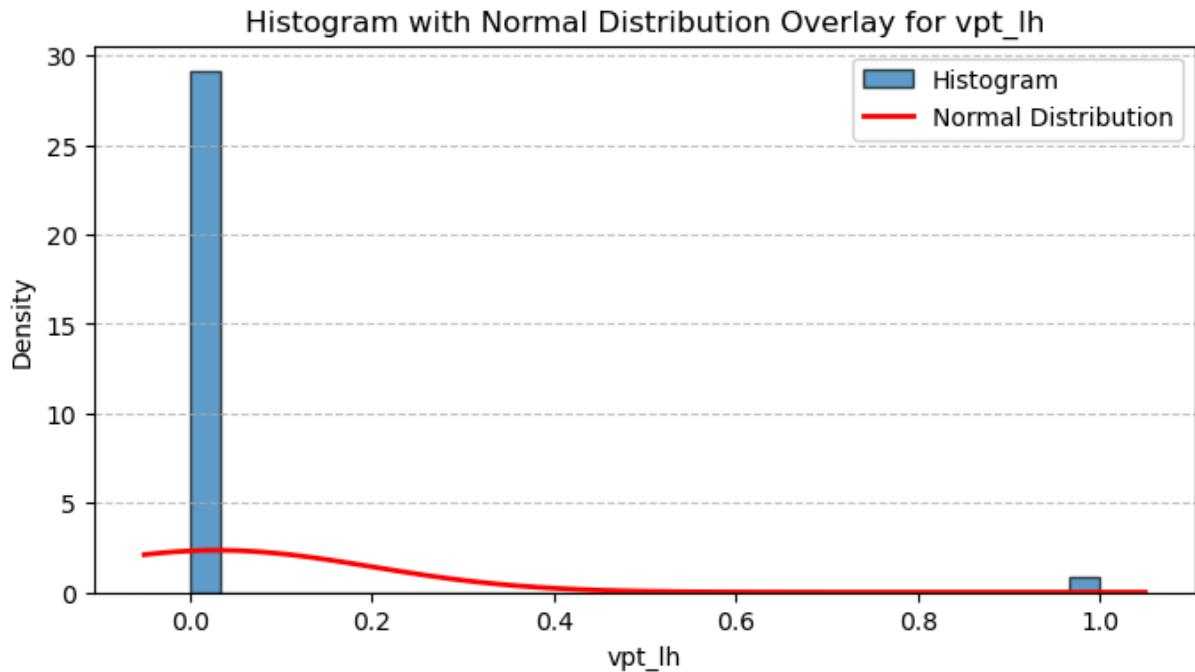


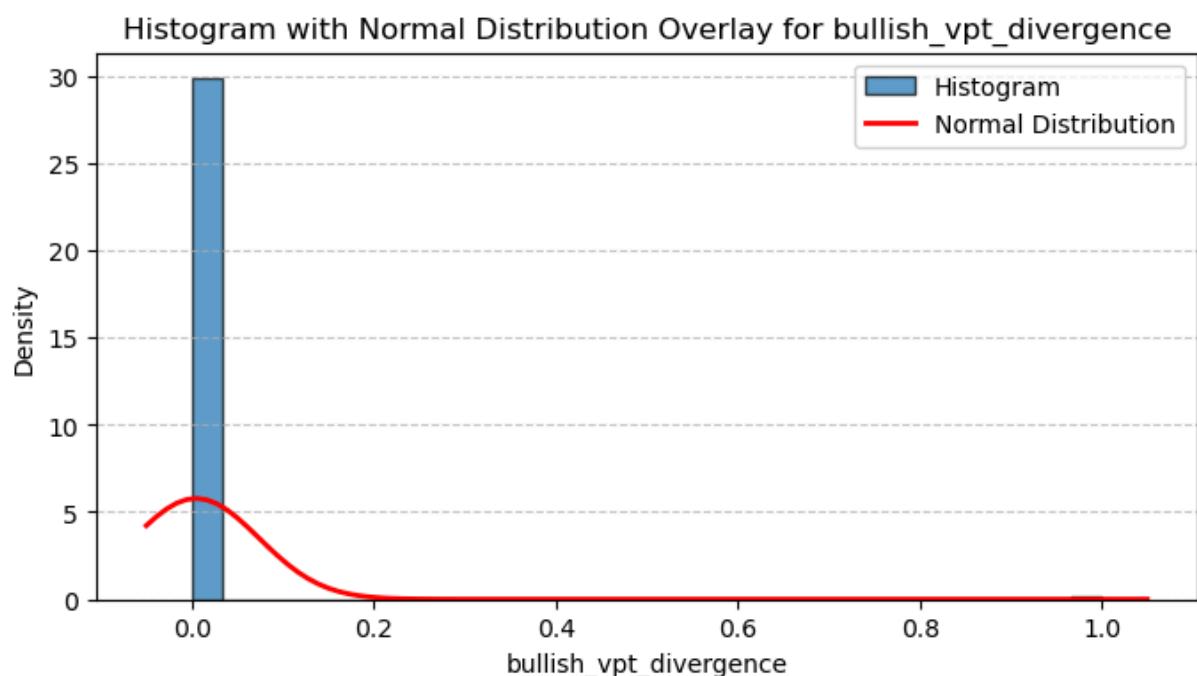
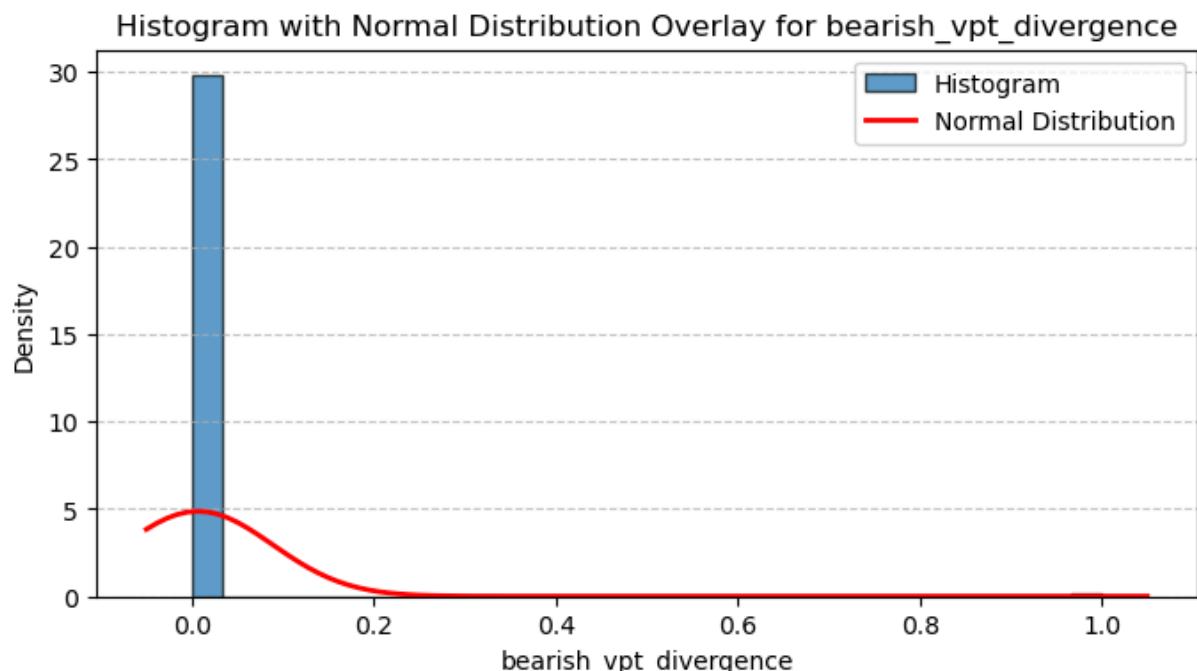


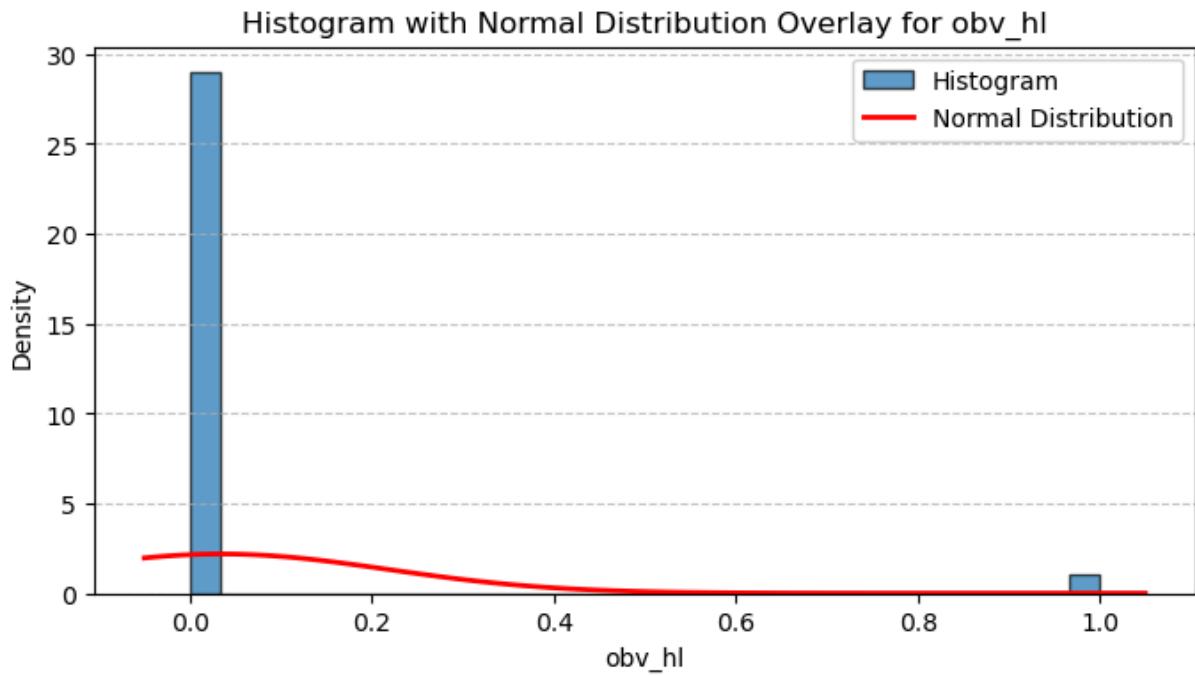
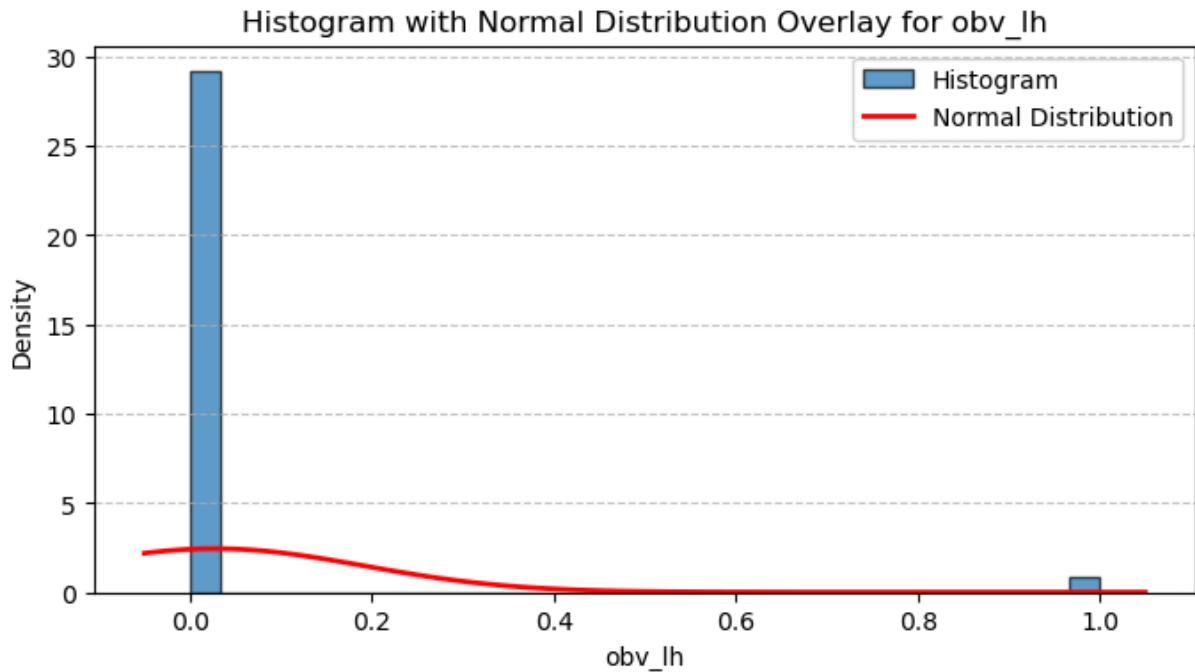


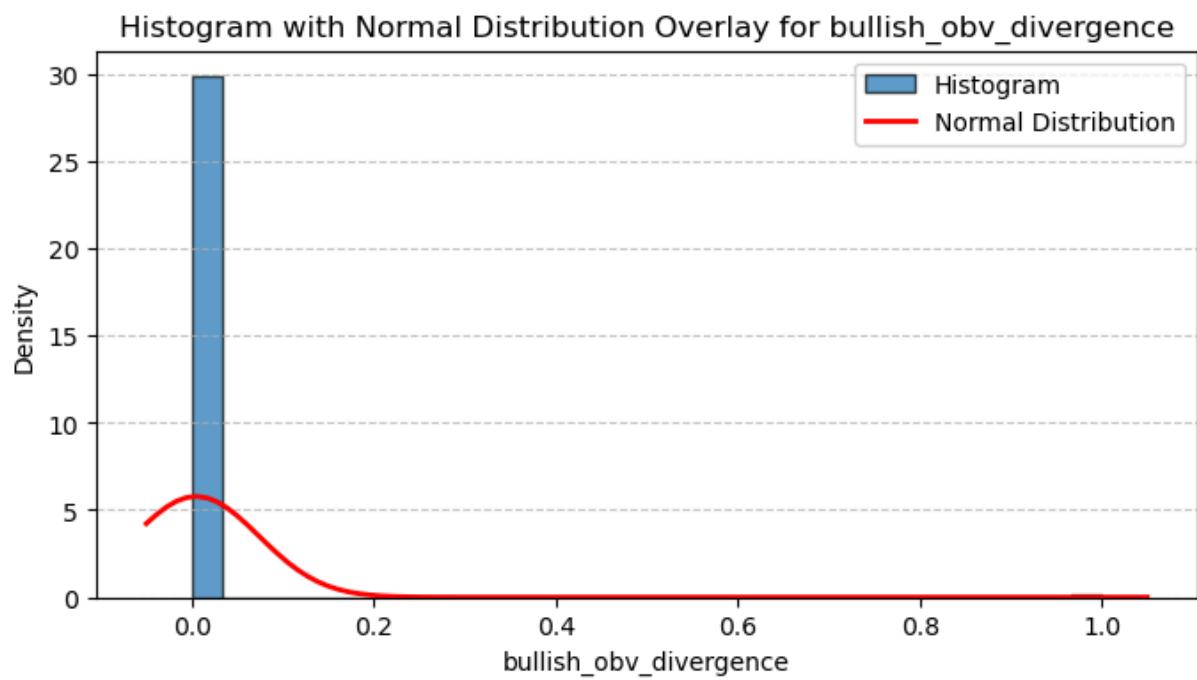
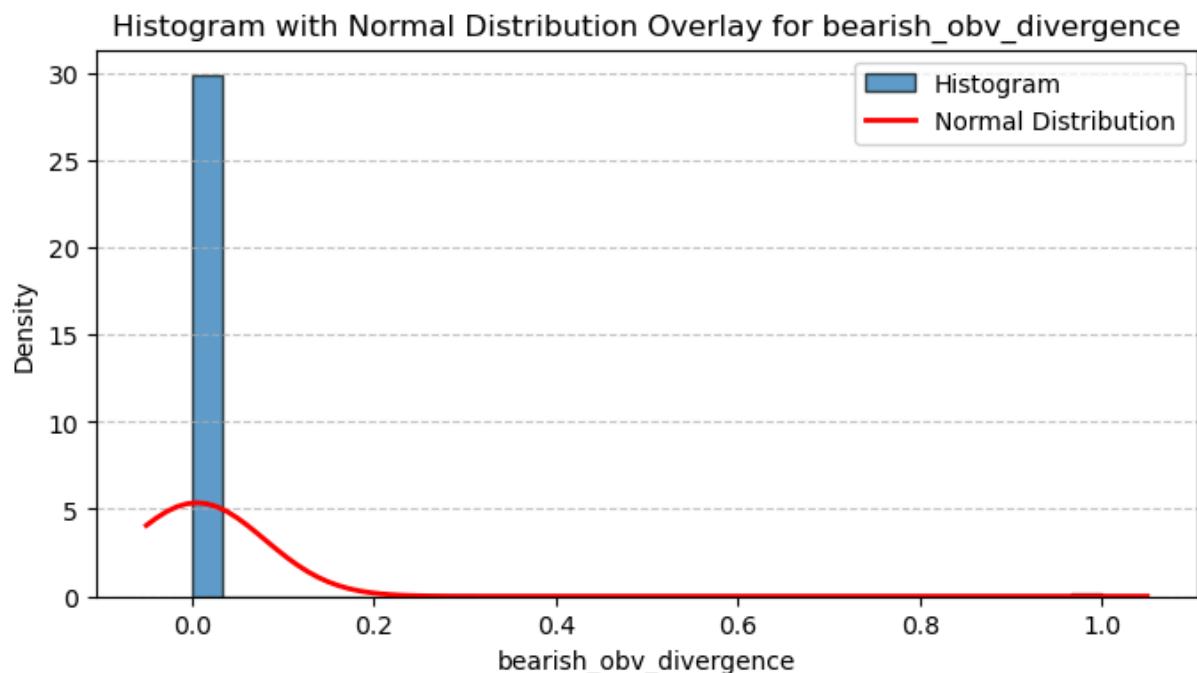


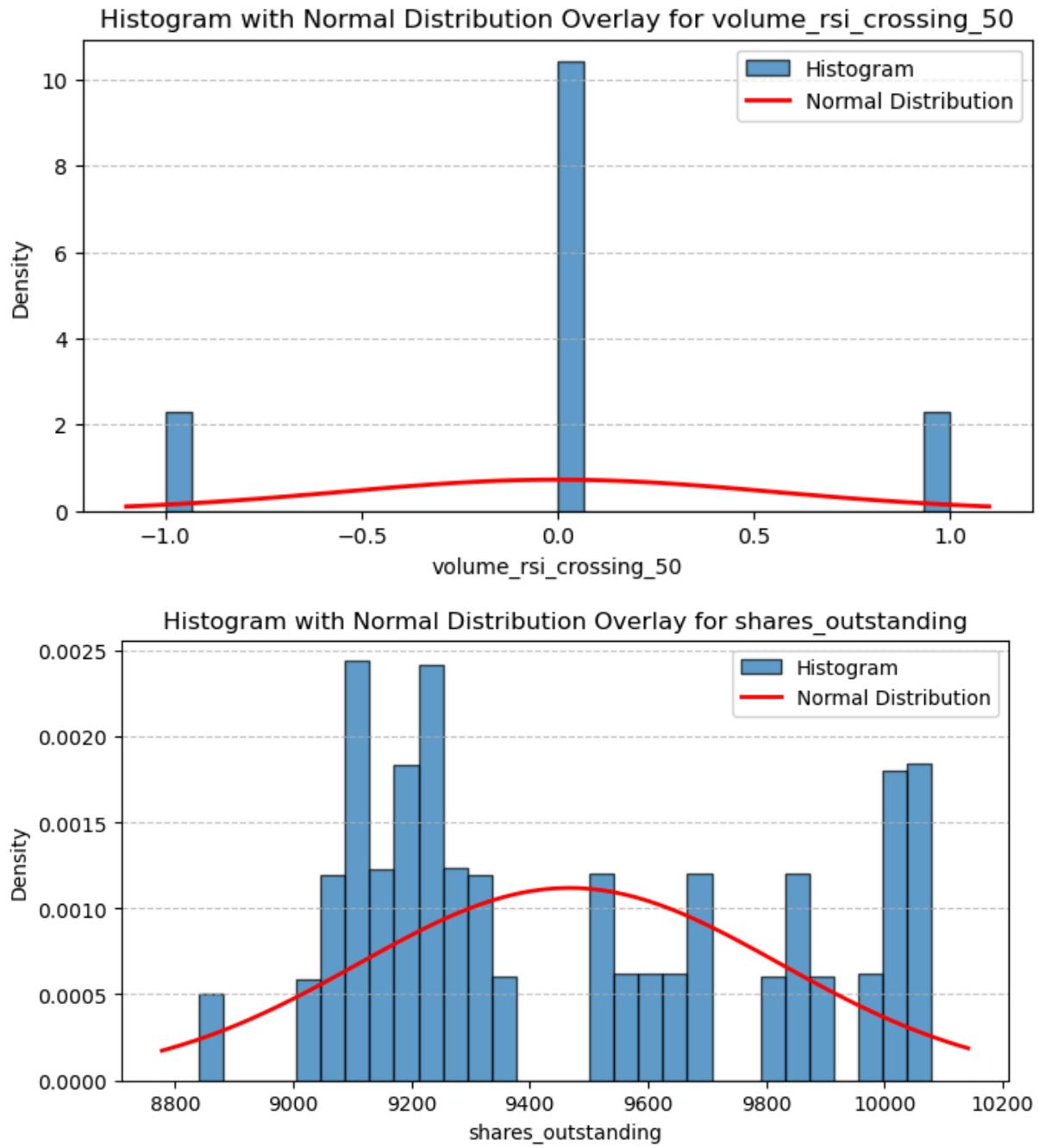


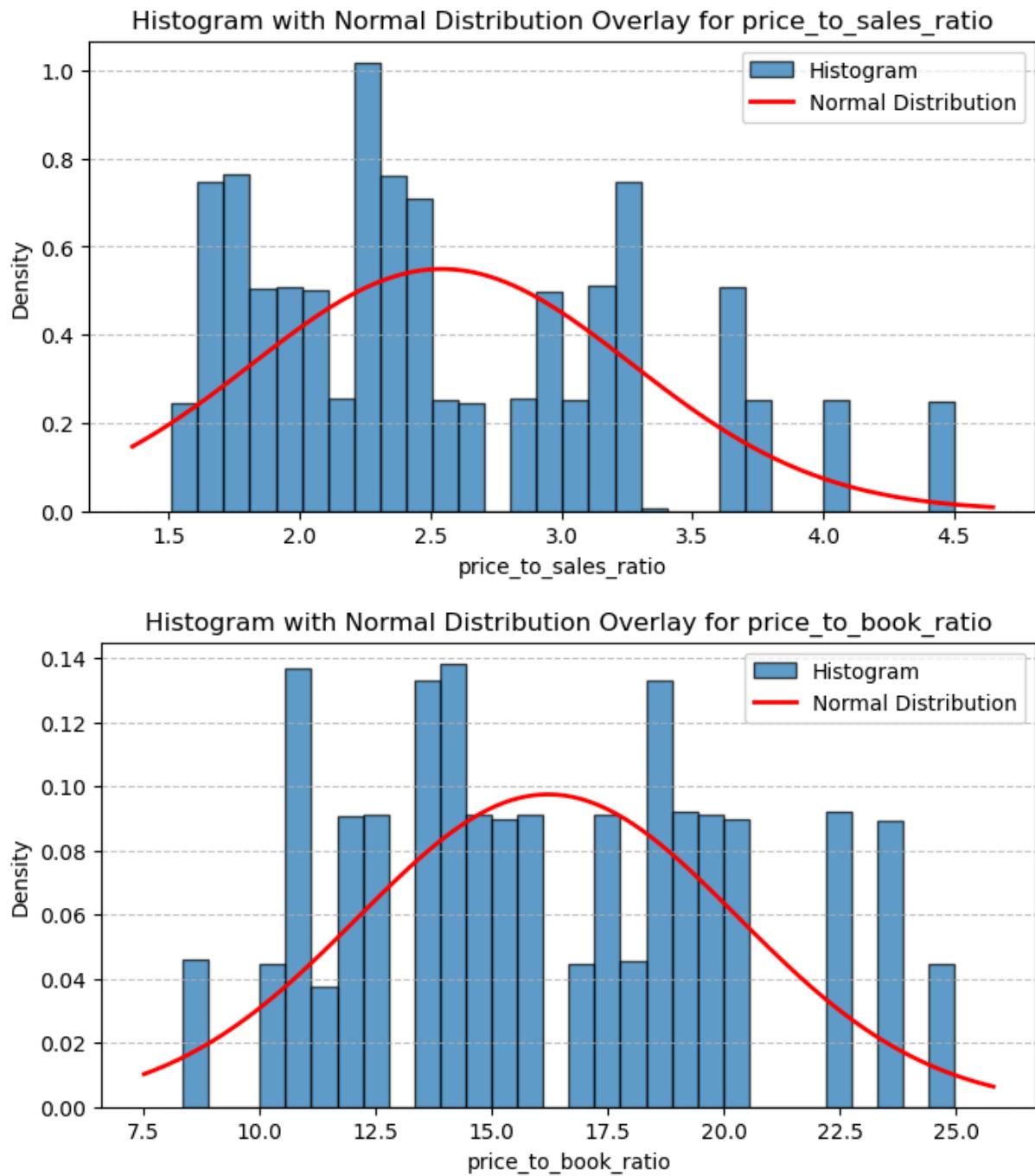




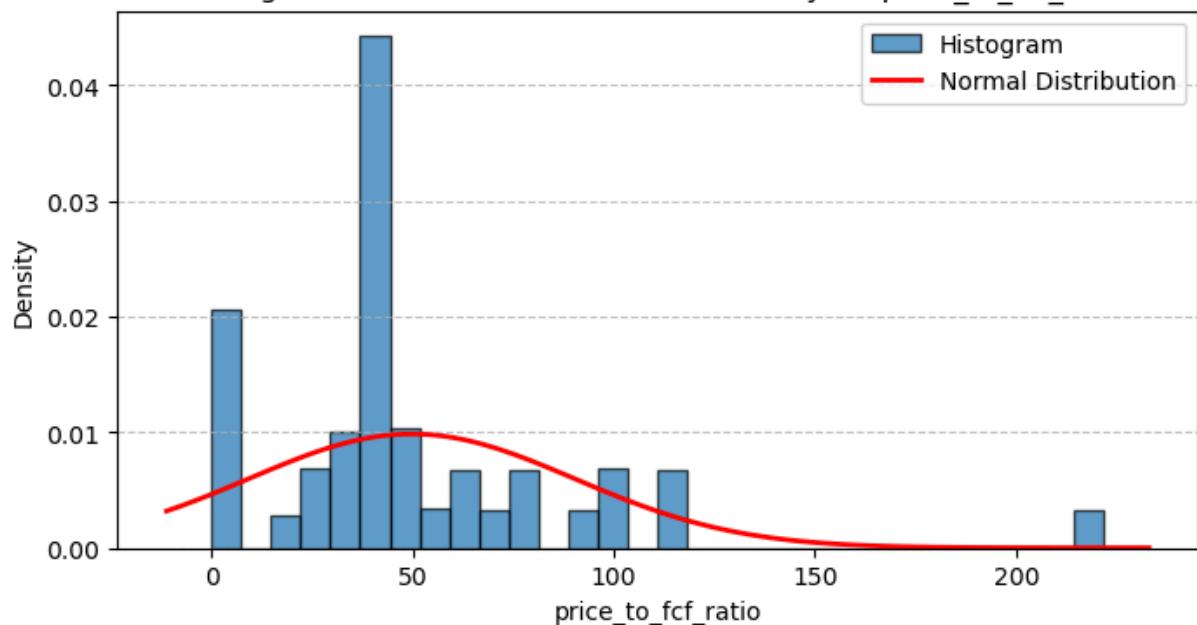




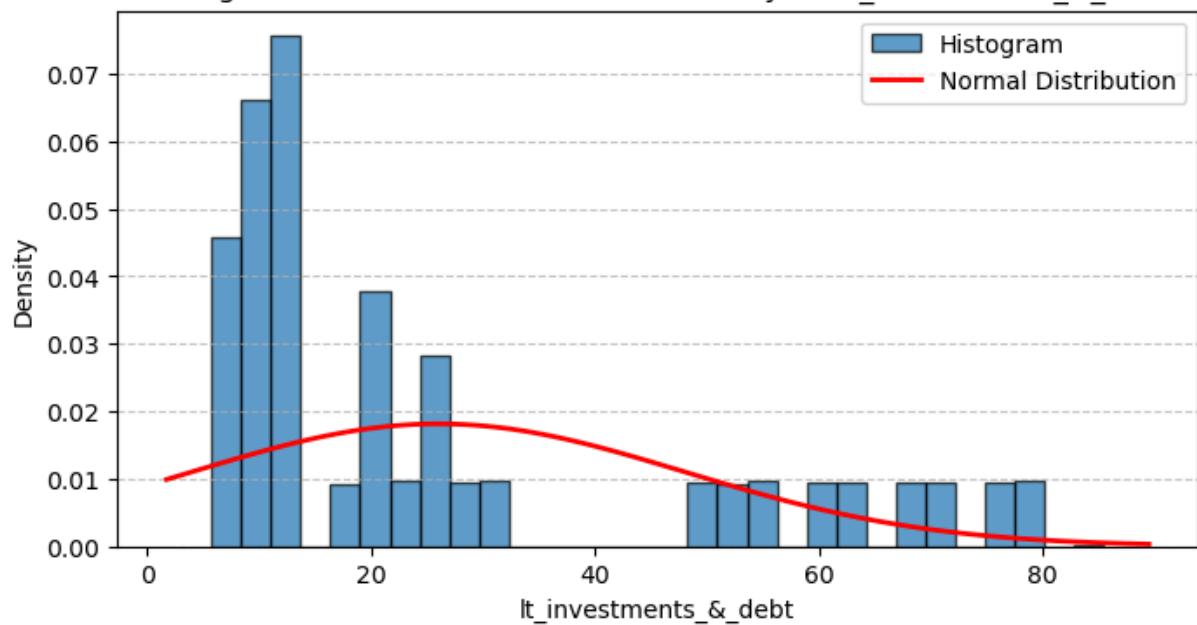


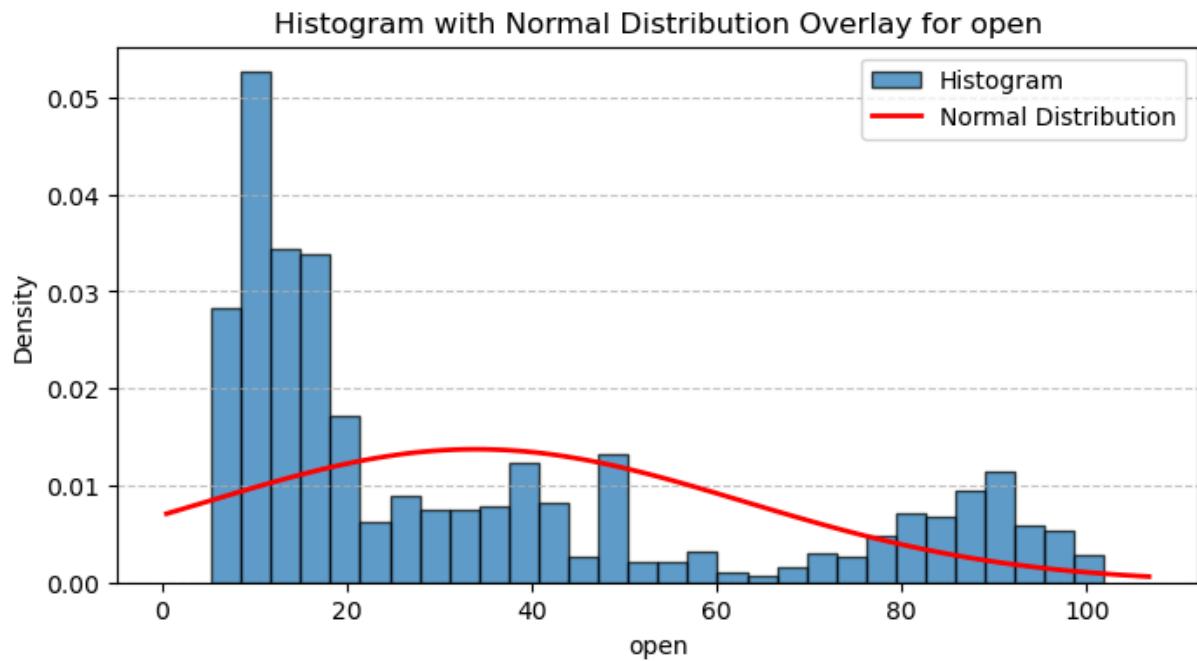
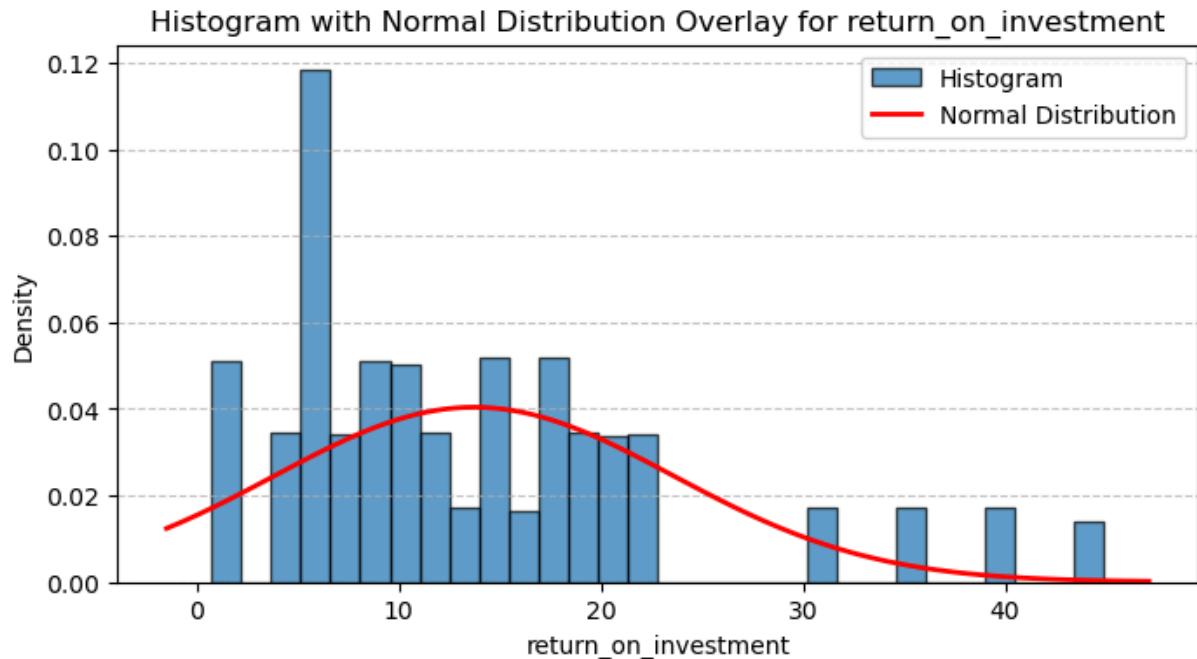


Histogram with Normal Distribution Overlay for price\_to\_fcf\_ratio

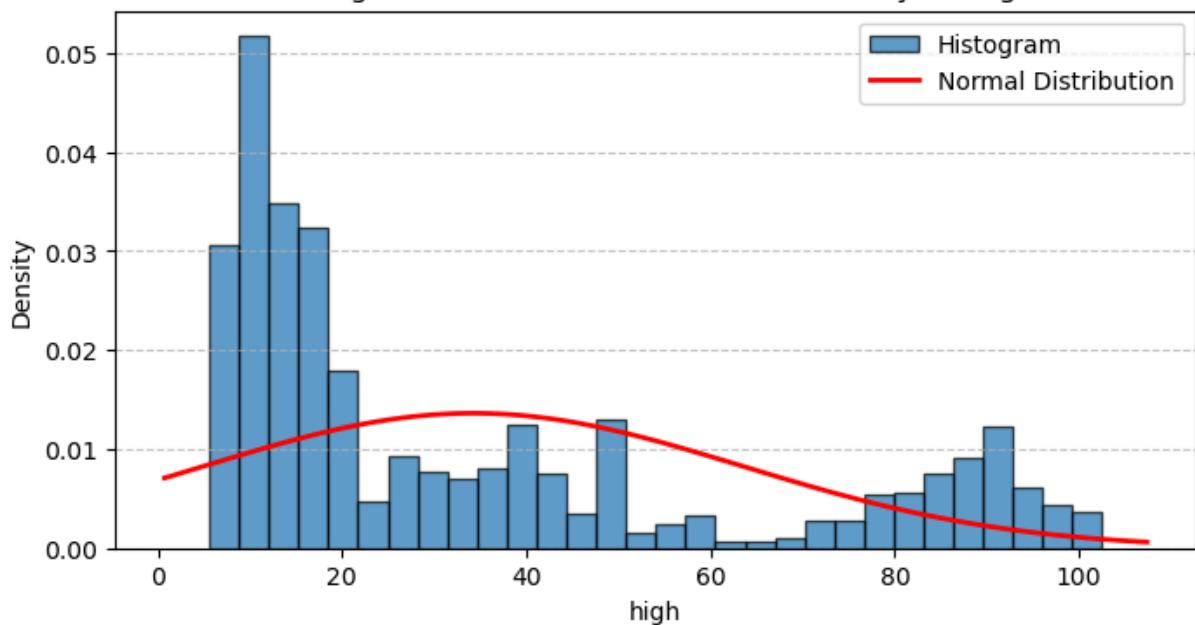


Histogram with Normal Distribution Overlay for lt\_investments\_&amp;\_debt

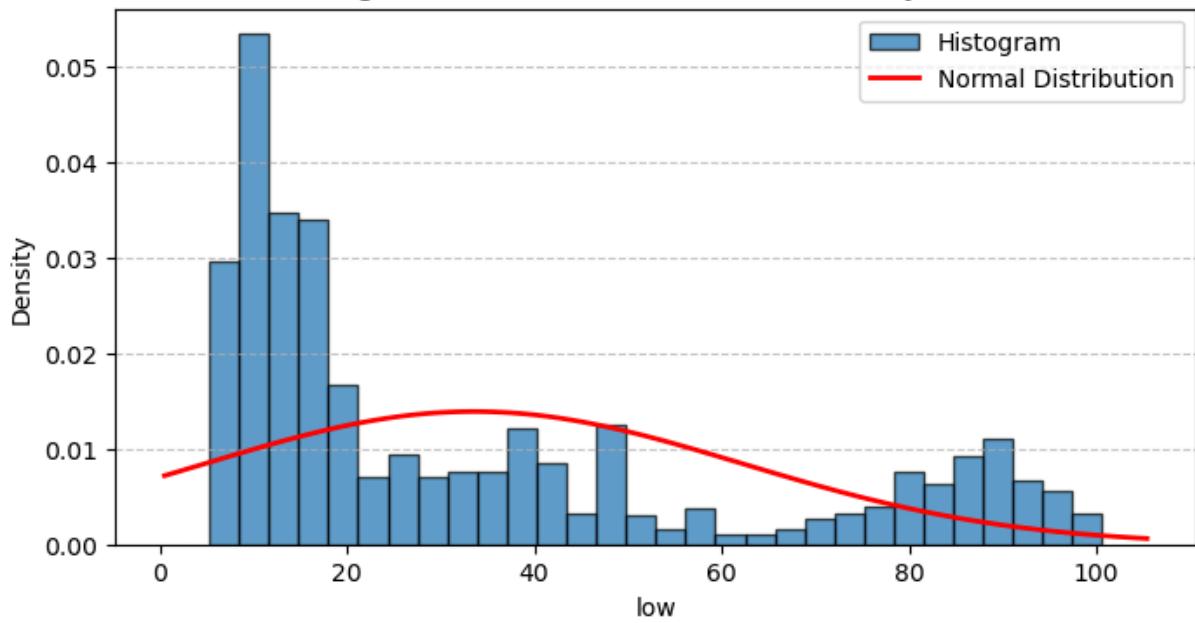


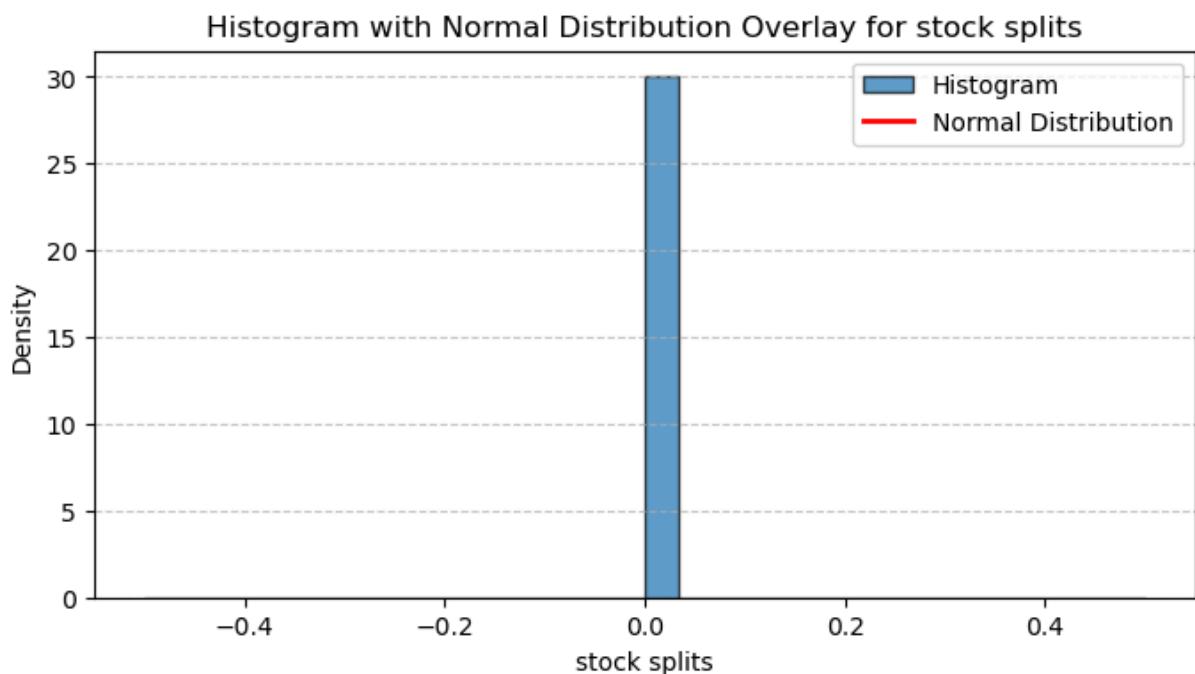
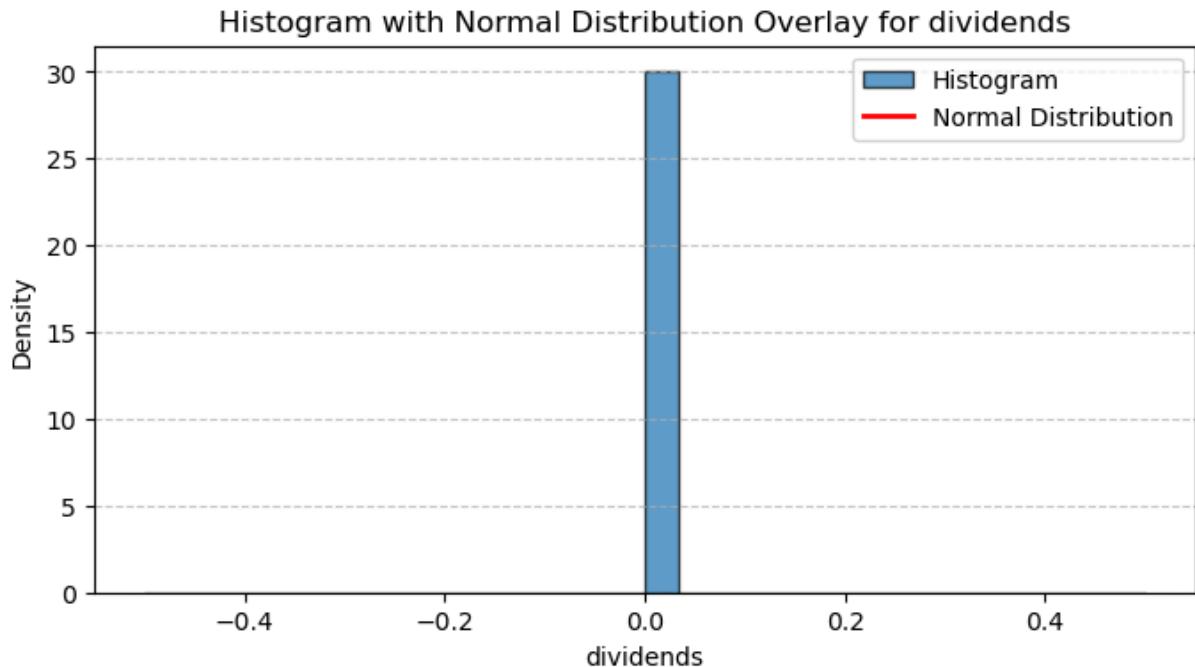


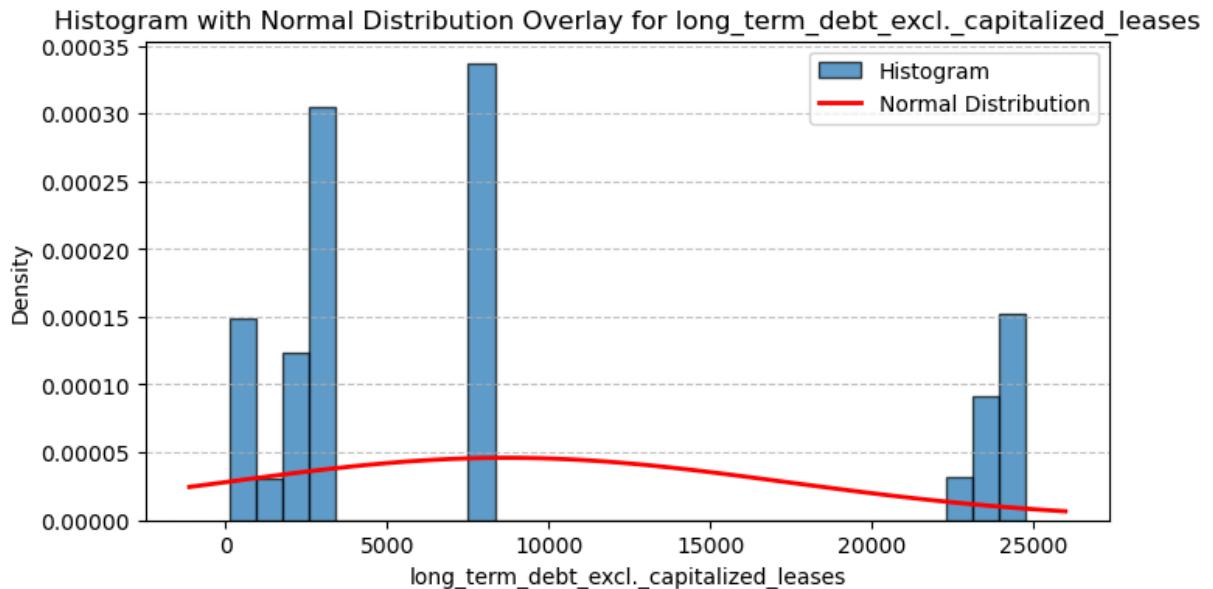
Histogram with Normal Distribution Overlay for high



Histogram with Normal Distribution Overlay for low







The histogram for close shows us that we likely want to log transform the response variable to achieve normality, depending on the model chosen for further analysis.

```
In [11]: import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

# Apply log transformation to the response
train_data['log_close'] = np.log(train_data['close'])

# Calculate mean and standard deviation of the log-transformed response
mean_log_close = np.mean(train_data['log_close'])
std_log_close = np.std(train_data['log_close'])

# Generate points for the normal distribution
x = np.linspace(min(train_data['log_close']), max(train_data['log_close']), 100)
normal_pdf = norm.pdf(x, mean_log_close, std_log_close)

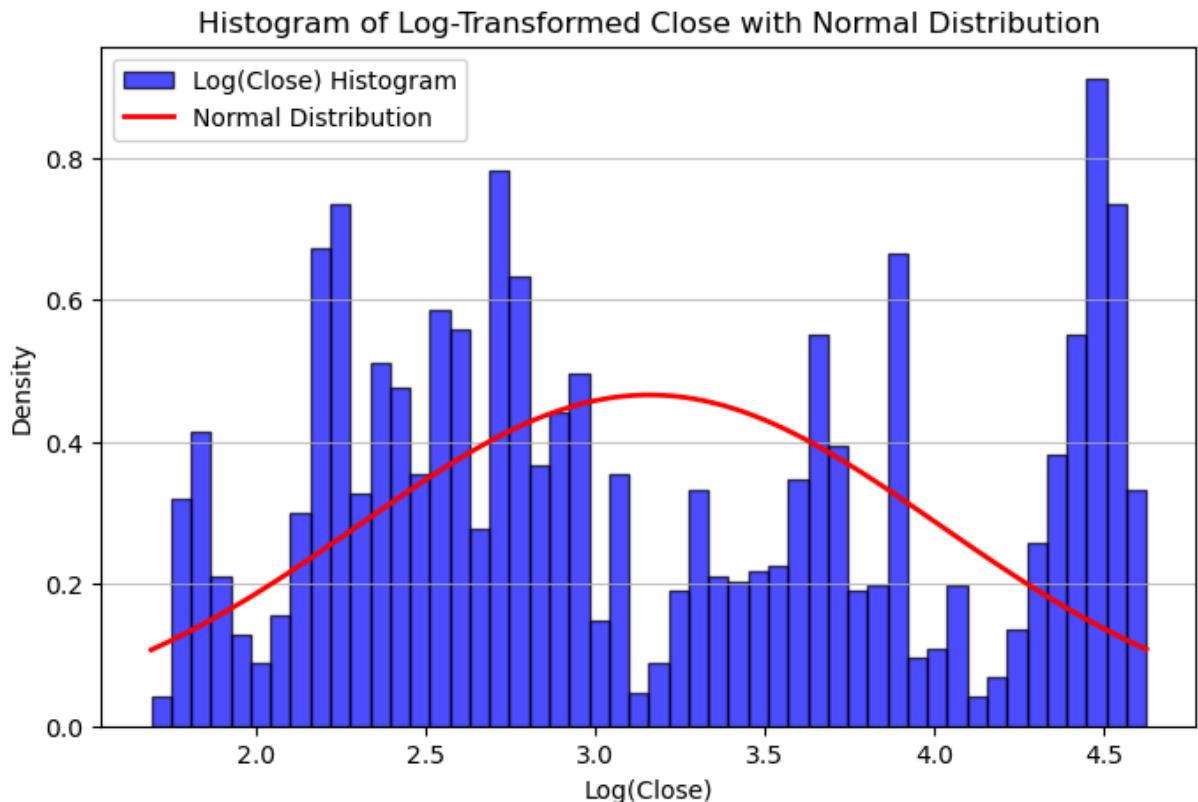
# Plot histogram of the log-transformed response
plt.figure(figsize=(8, 5))
plt.hist(train_data['log_close'], bins=50, alpha=0.7, color='blue', edgecolor='black')

# Overlay the normal distribution
plt.plot(x, normal_pdf, color='red', linewidth=2, label='Normal Distribution')

# Add titles and labels
plt.title("Histogram of Log-Transformed Close with Normal Distribution")
plt.xlabel("Log(Close)")
plt.ylabel("Density")
plt.legend()
plt.grid(axis='y', alpha=0.75)
plt.show()
```

```
/var/folders/2p/hbz8h54x5hj828cdgg7jshz0000gn/T/ipykernel_66719/255999552.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
train_data['log_close'] = np.log(train_data['close'])
```

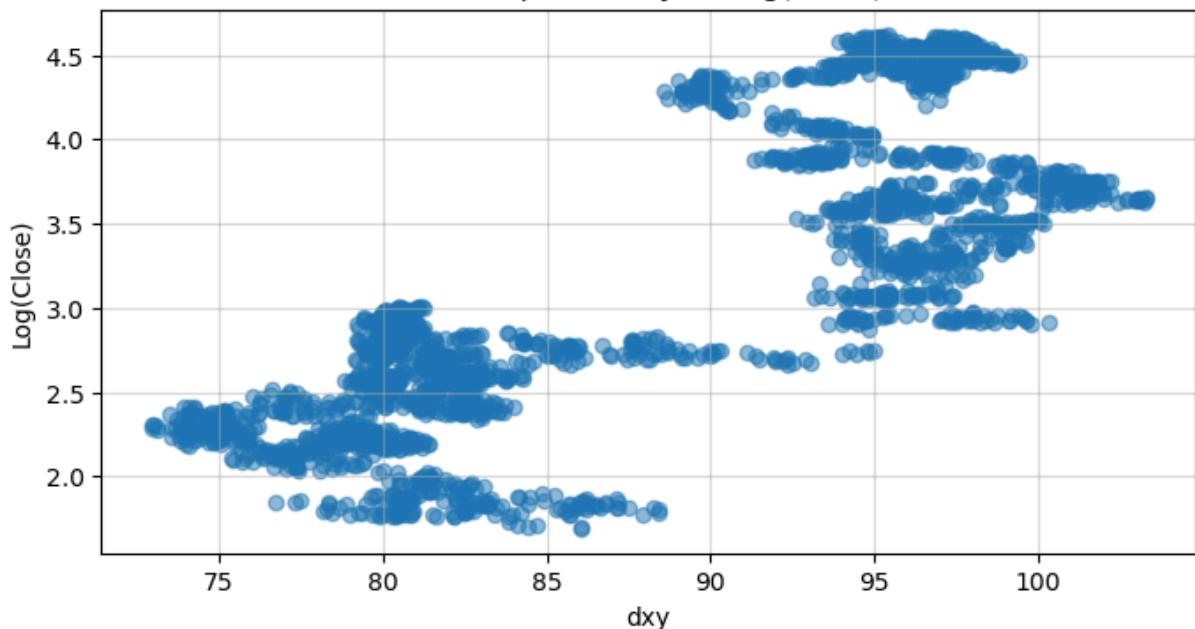


## Scatterplots

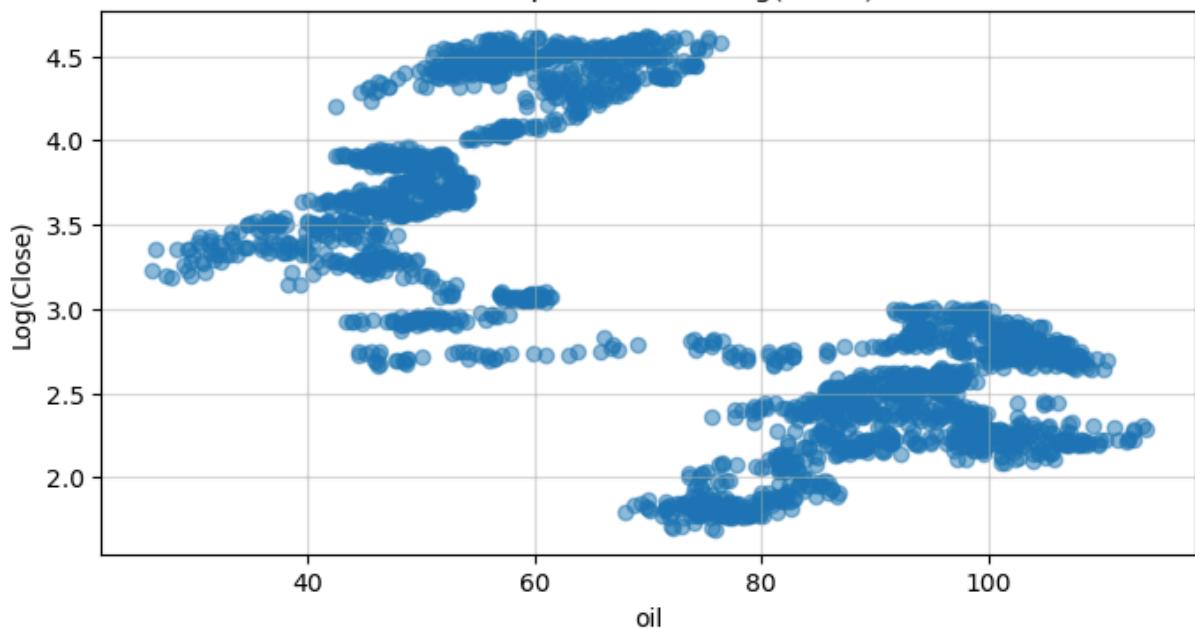
```
In [12]: import matplotlib.pyplot as plt

# Generate scatterplots for each numeric variable with the log-transformed r
for column in train_data.columns:
    if column not in ['close', 'log_close'] and train_data[column].dtype in
        plt.figure(figsize=(8, 4))
        plt.scatter(train_data[column], train_data['log_close'], alpha=0.5)
        plt.title(f"Scatterplot of {column} vs Log(Close)")
        plt.xlabel(column)
        plt.ylabel('Log(Close)')
        plt.grid(alpha=0.5)
        plt.show()
```

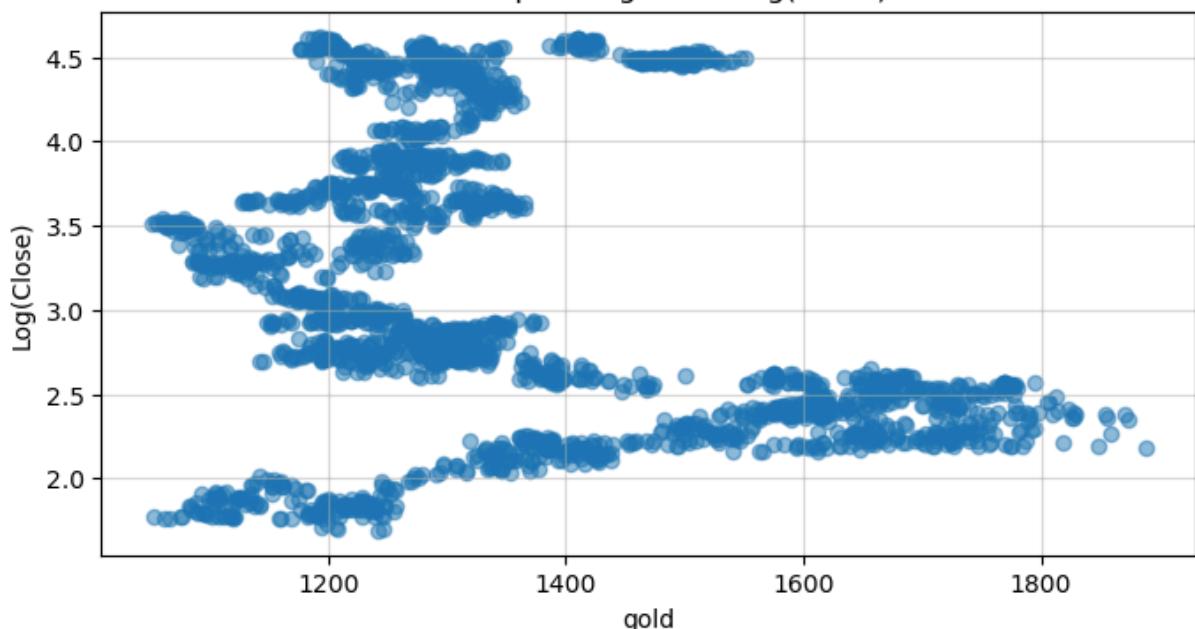
Scatterplot of dxy vs Log(Close)



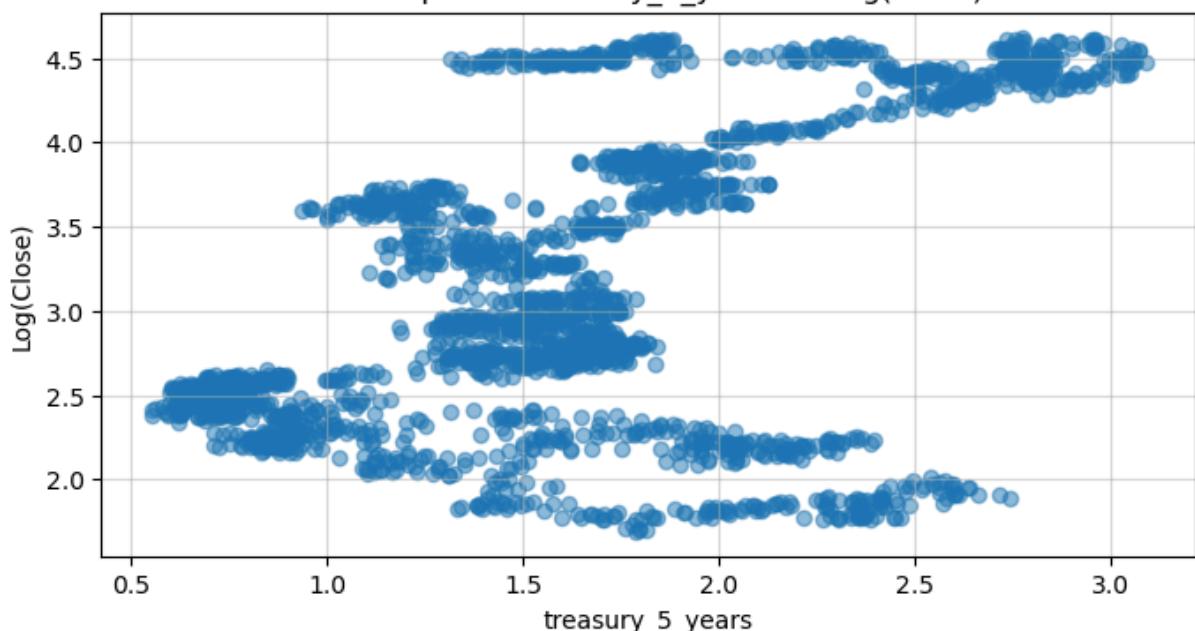
Scatterplot of oil vs Log(Close)



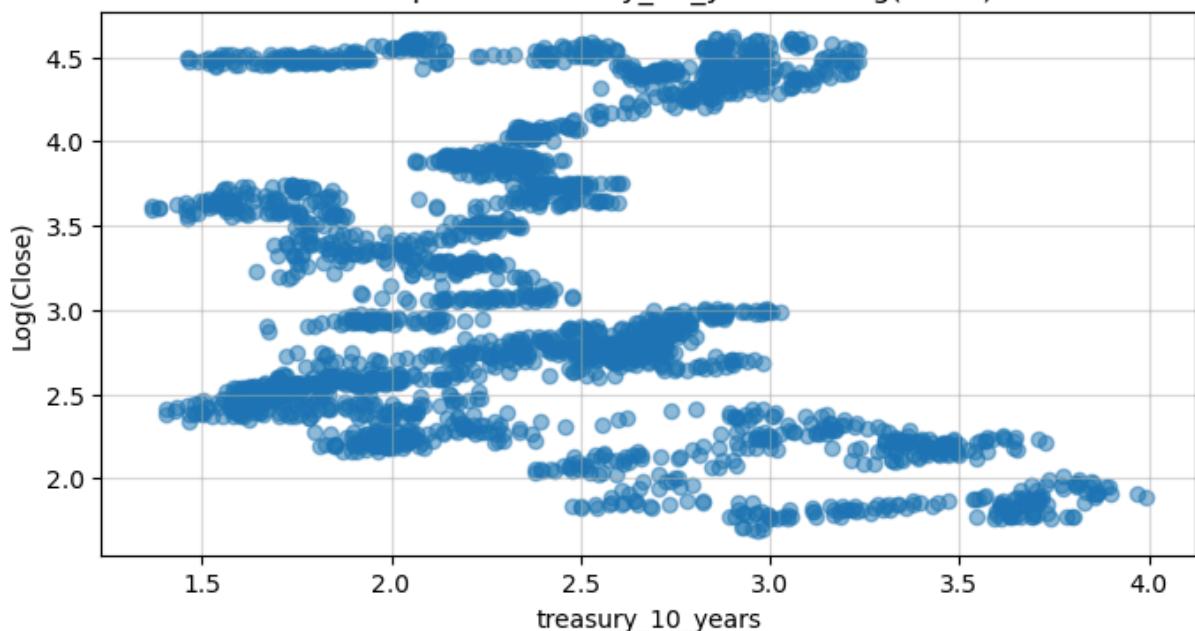
Scatterplot of gold vs Log(Close)



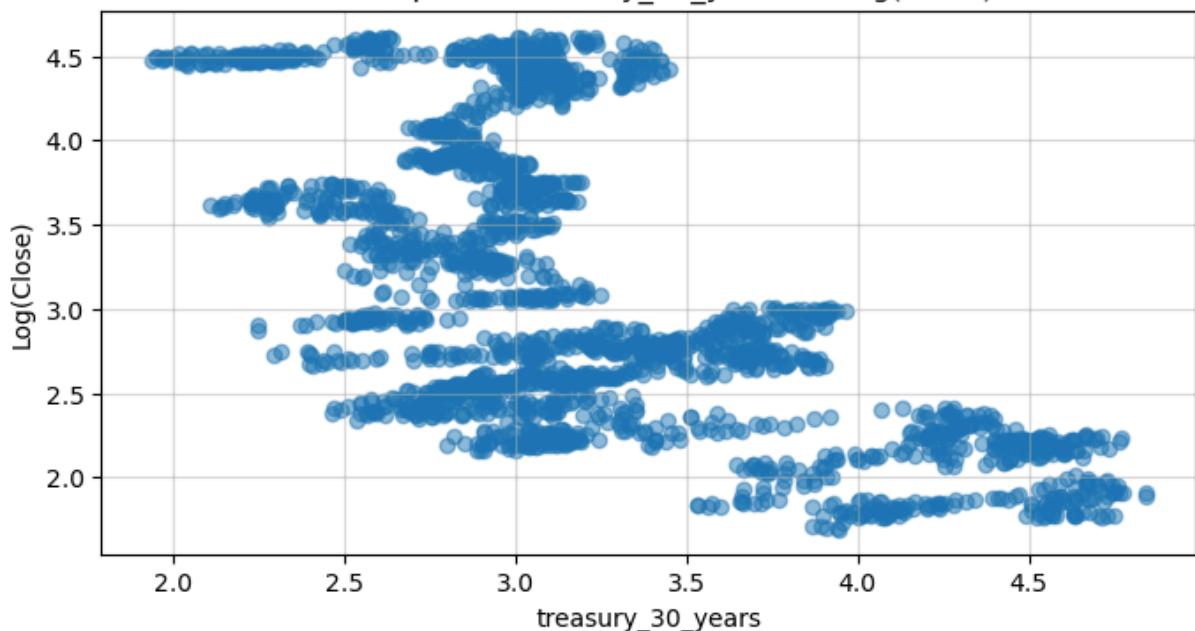
Scatterplot of treasury\_5\_years vs Log(Close)



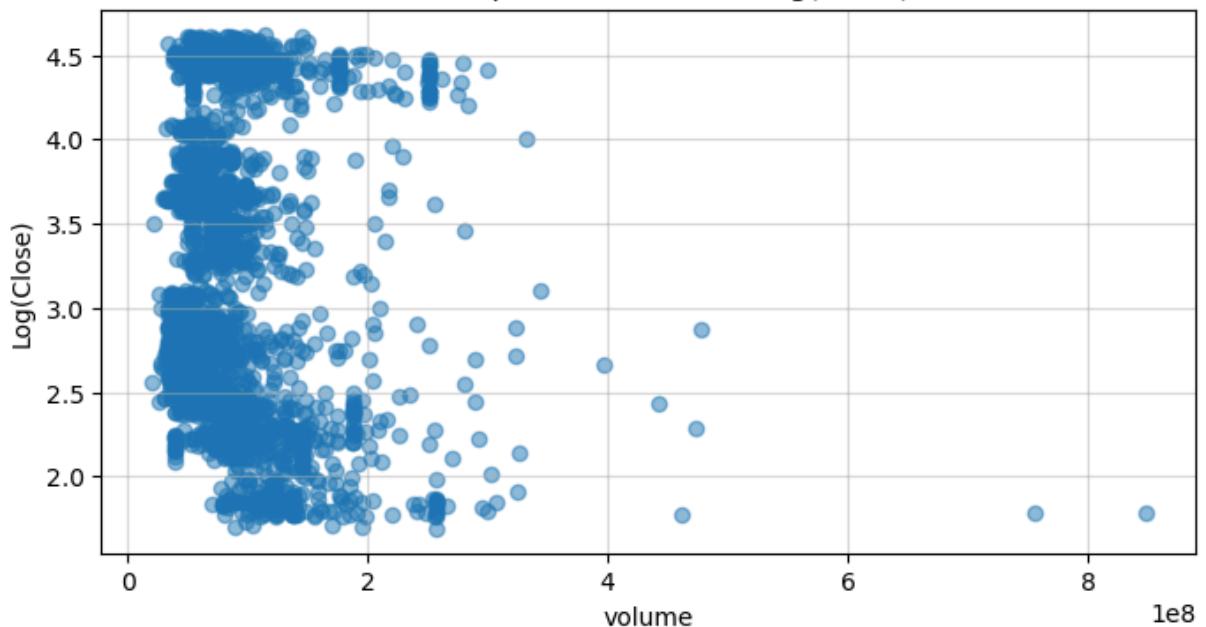
Scatterplot of treasury\_10\_years vs Log(Close)



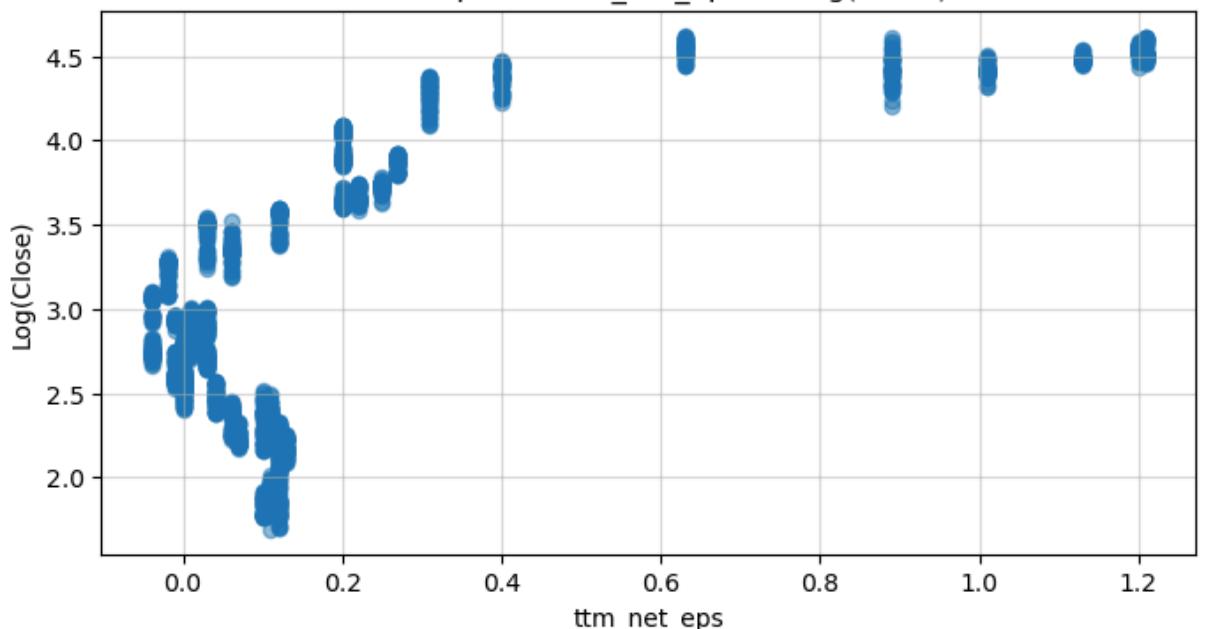
Scatterplot of treasury\_30\_years vs Log(Close)



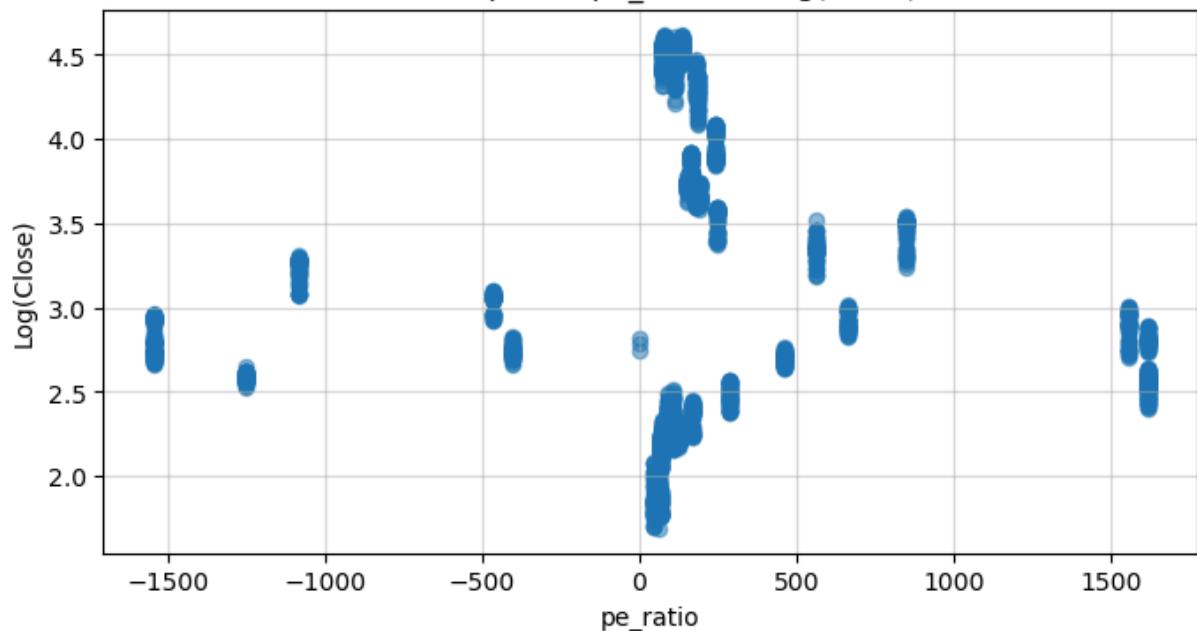
Scatterplot of volume vs Log(Close)



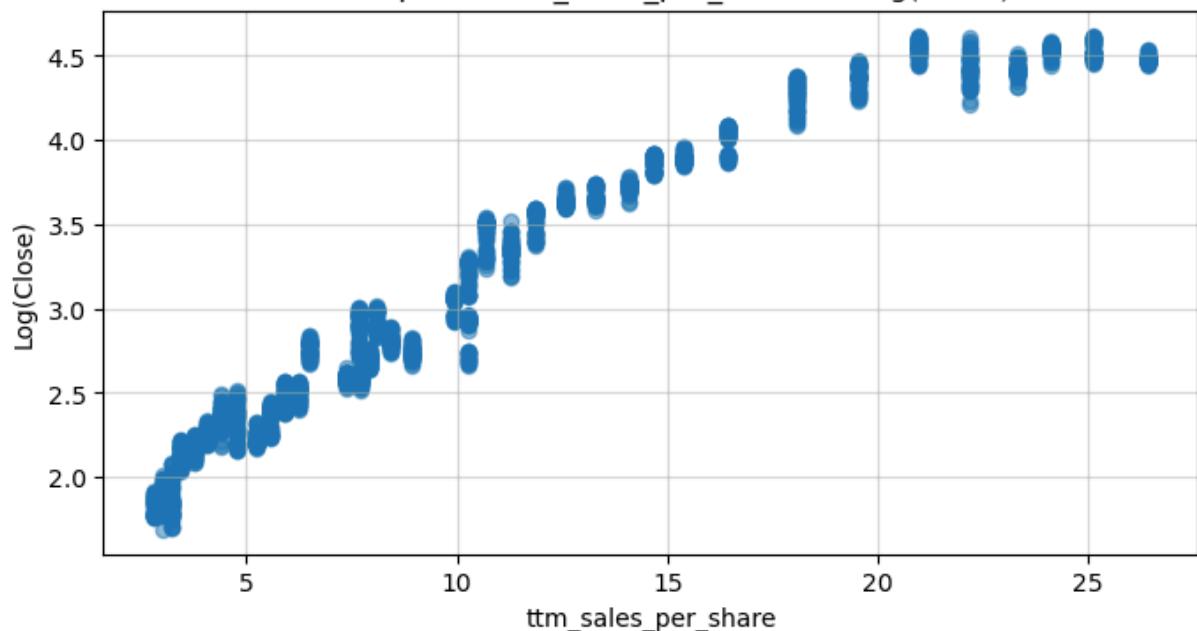
Scatterplot of ttm\_net\_eps vs Log(Close)



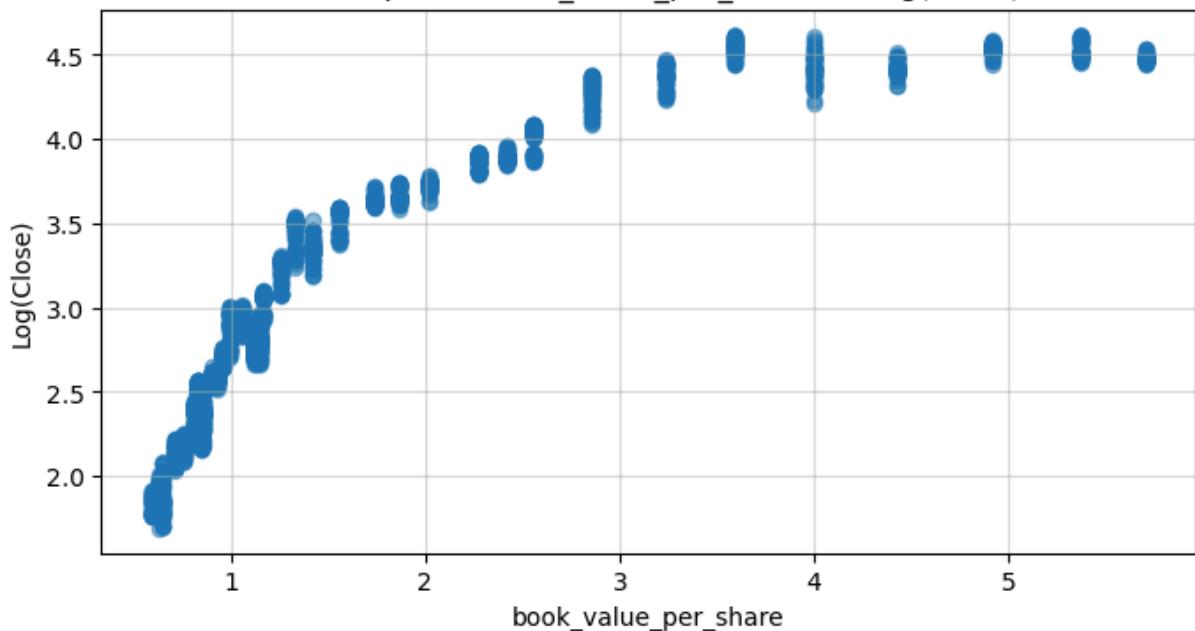
Scatterplot of pe\_ratio vs Log(Close)



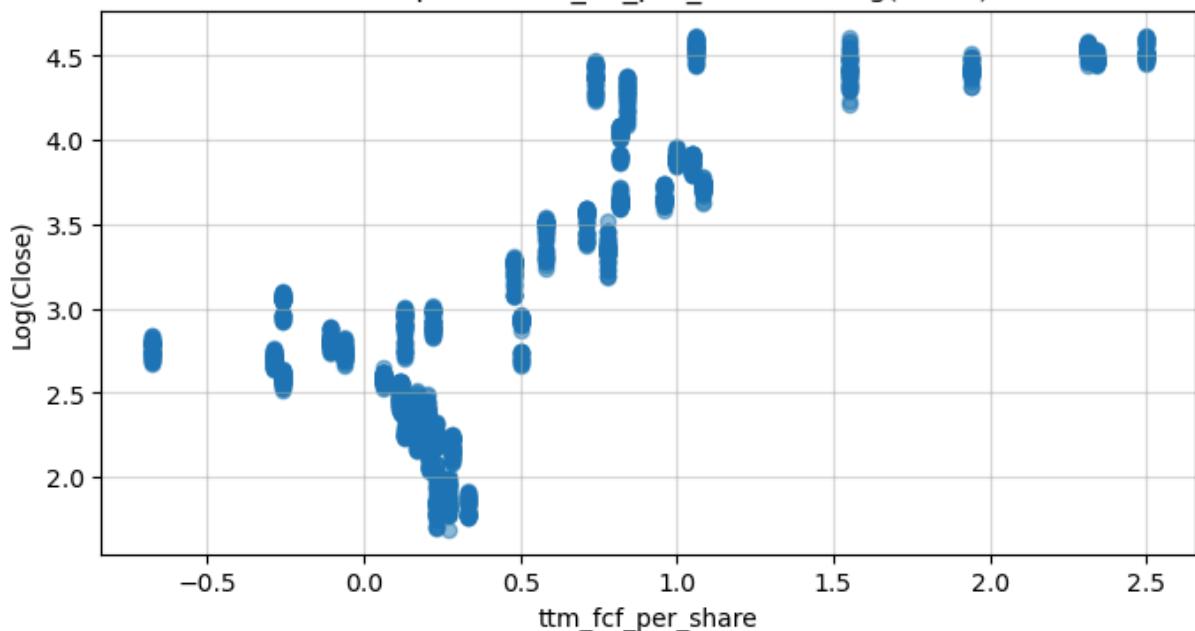
Scatterplot of ttm\_sales\_per\_share vs Log(Close)



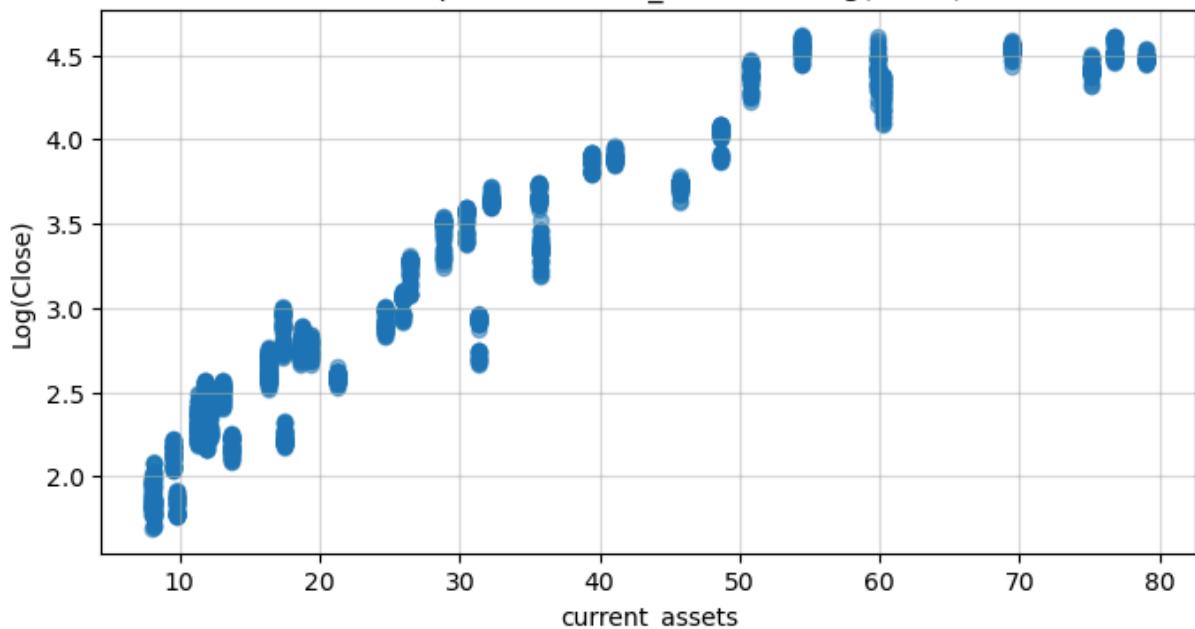
Scatterplot of book\_value\_per\_share vs Log(Close)



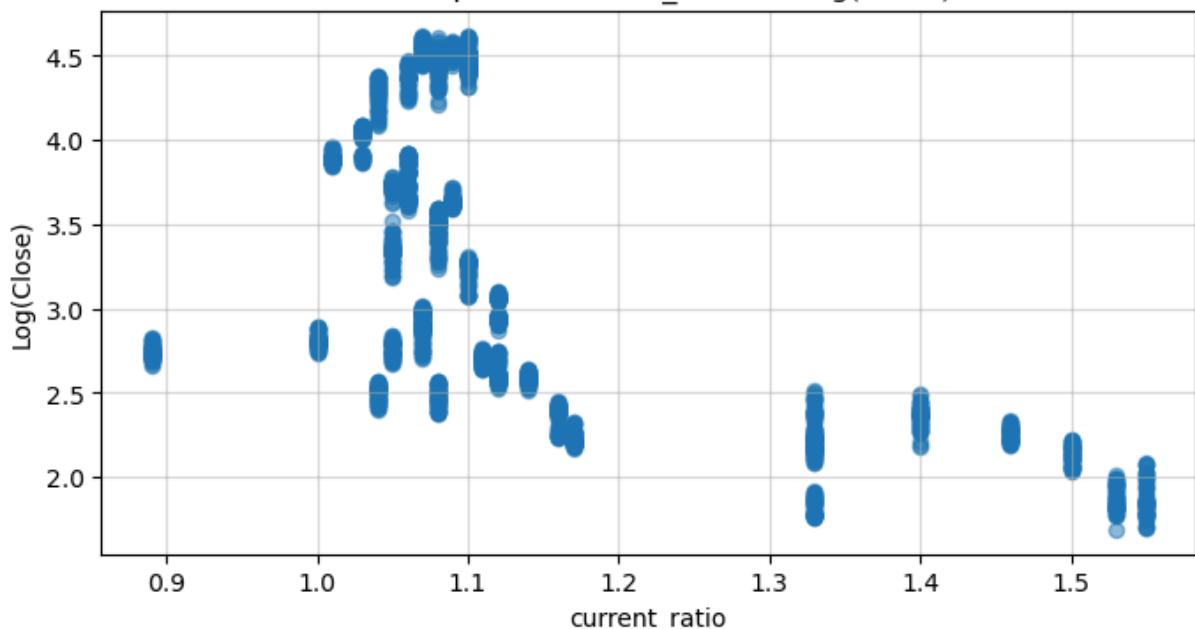
Scatterplot of ttm\_fcf\_per\_share vs Log(Close)

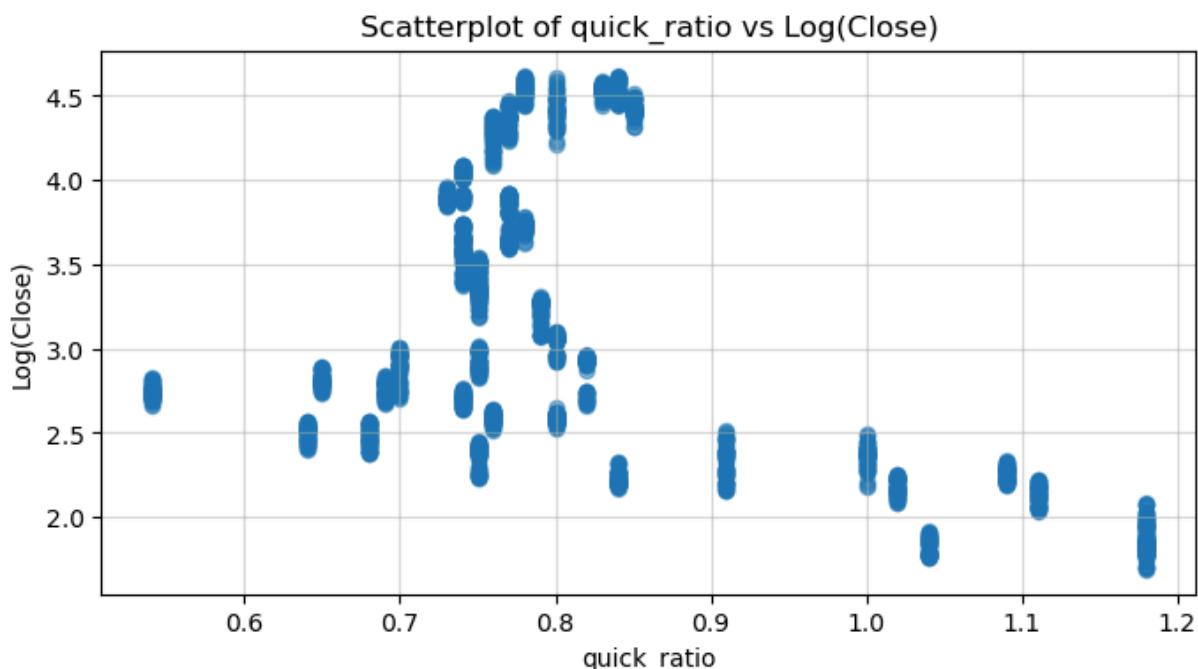
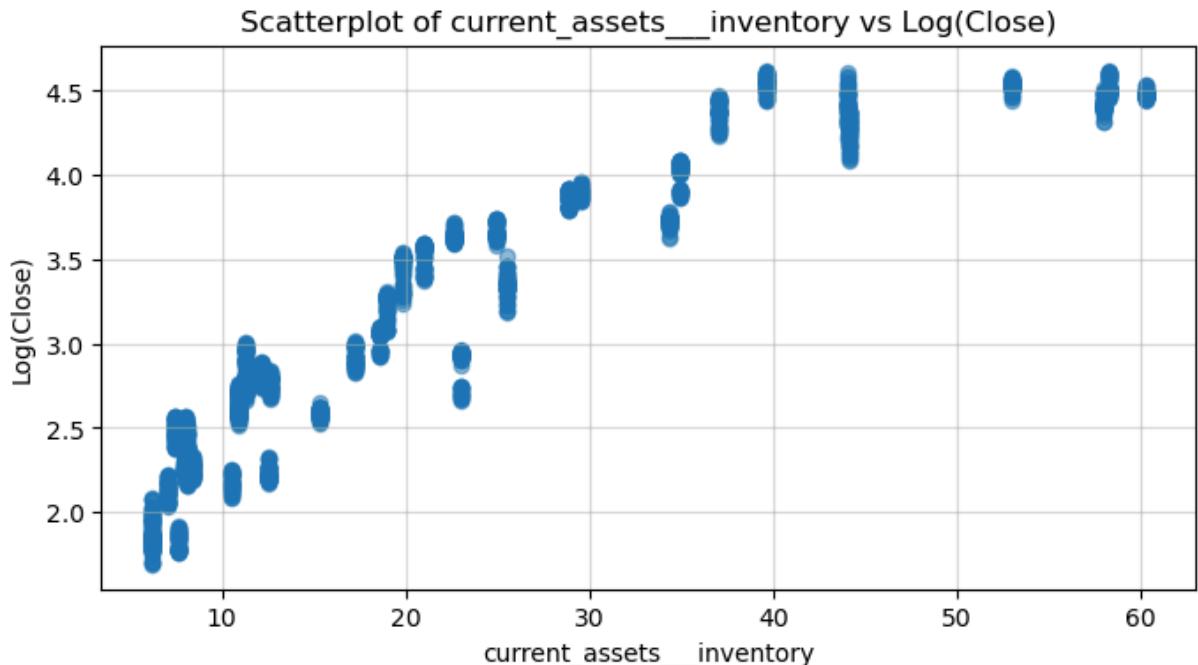


Scatterplot of current\_assets vs Log(Close)

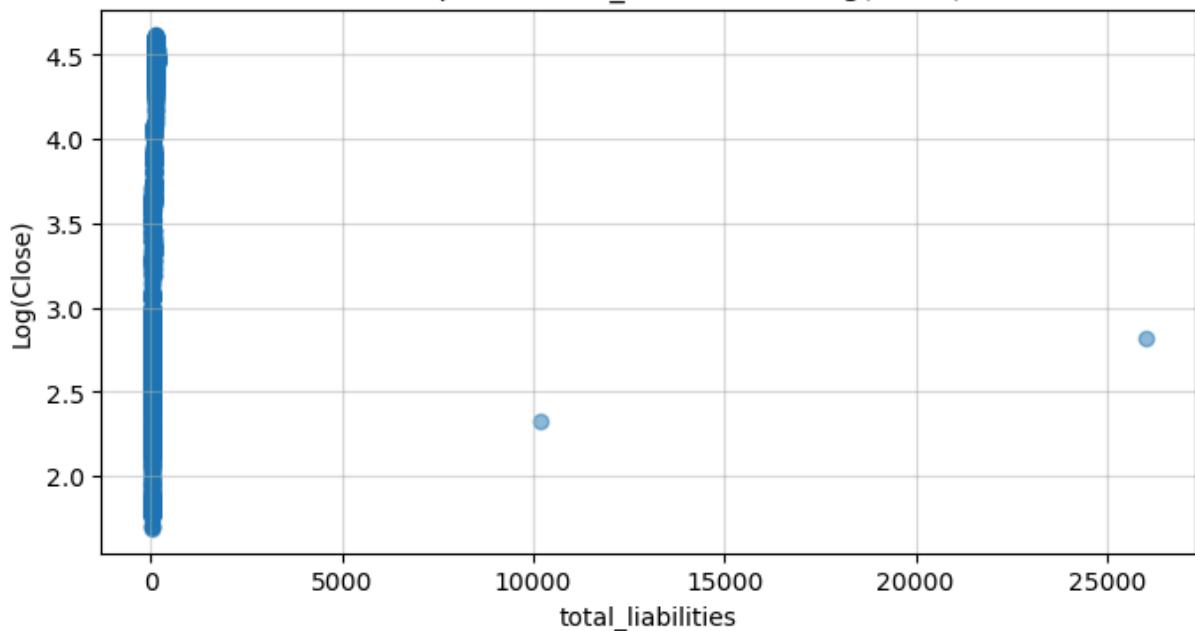


Scatterplot of current\_ratio vs Log(Close)

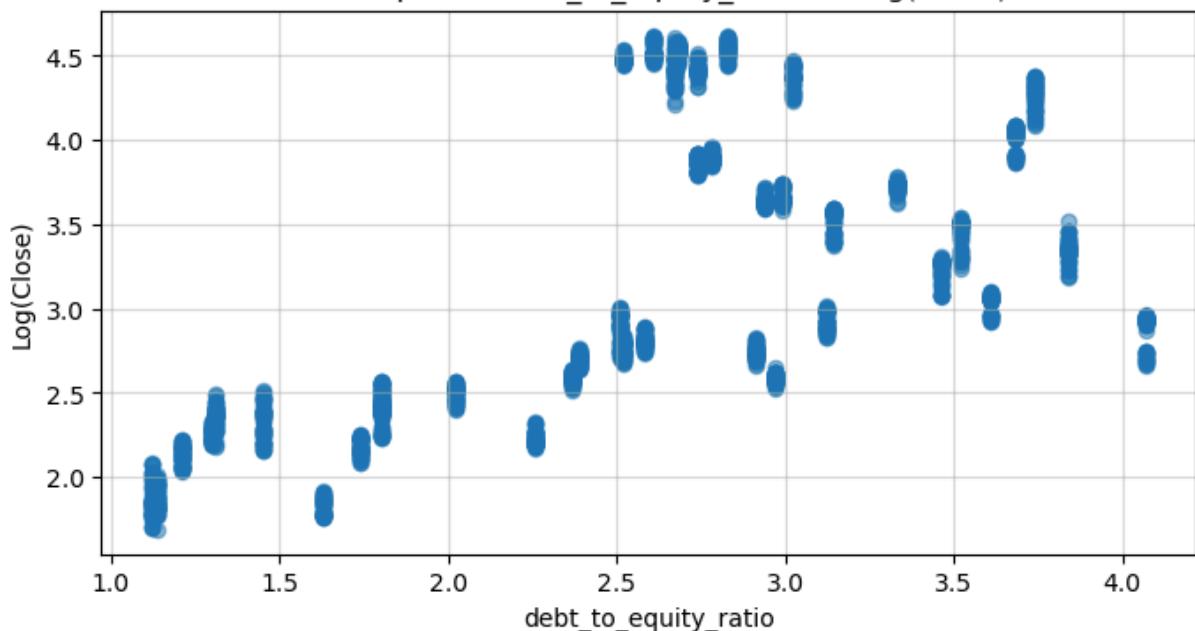




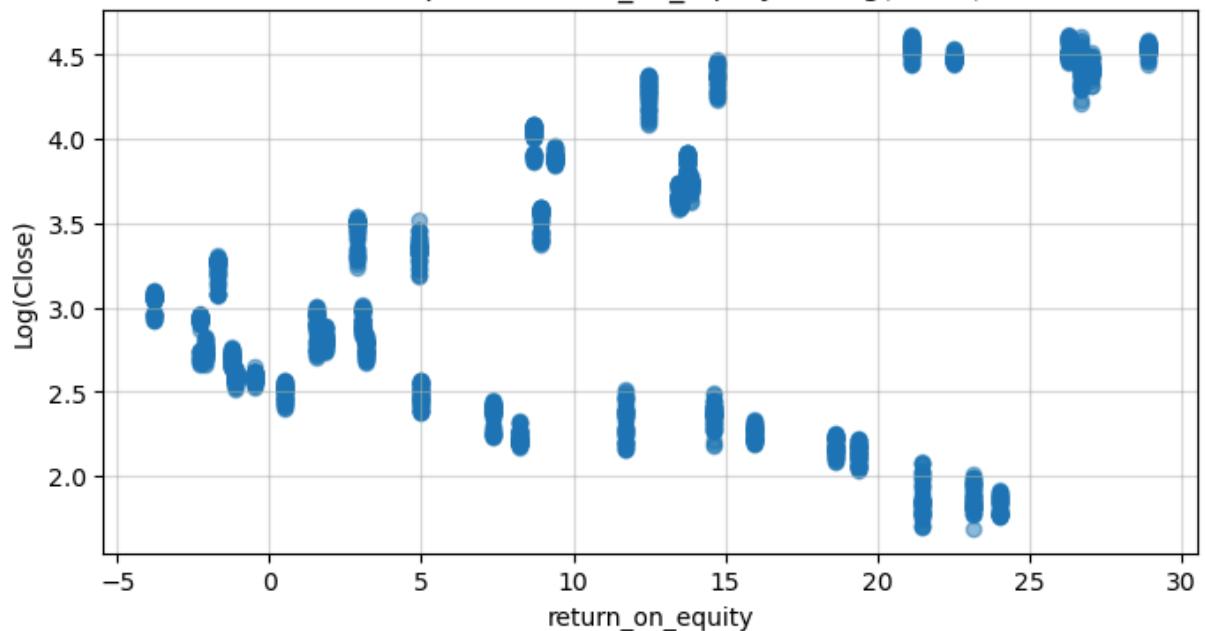
Scatterplot of total\_liabilities vs Log(Close)



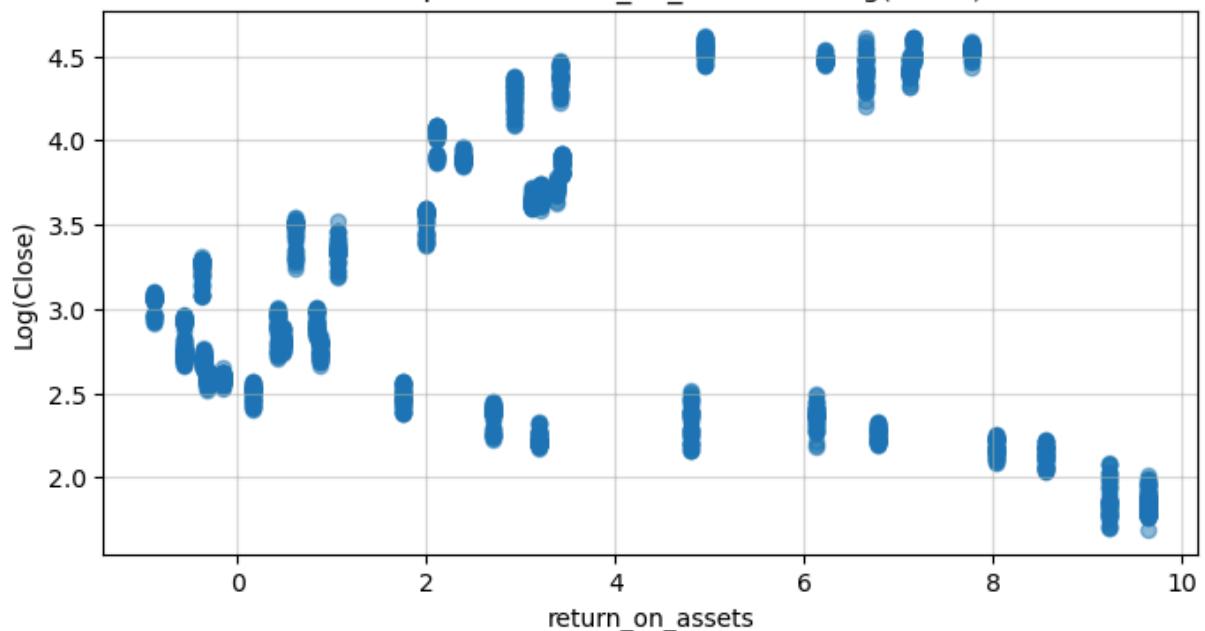
Scatterplot of debt\_to\_equity\_ratio vs Log(Close)



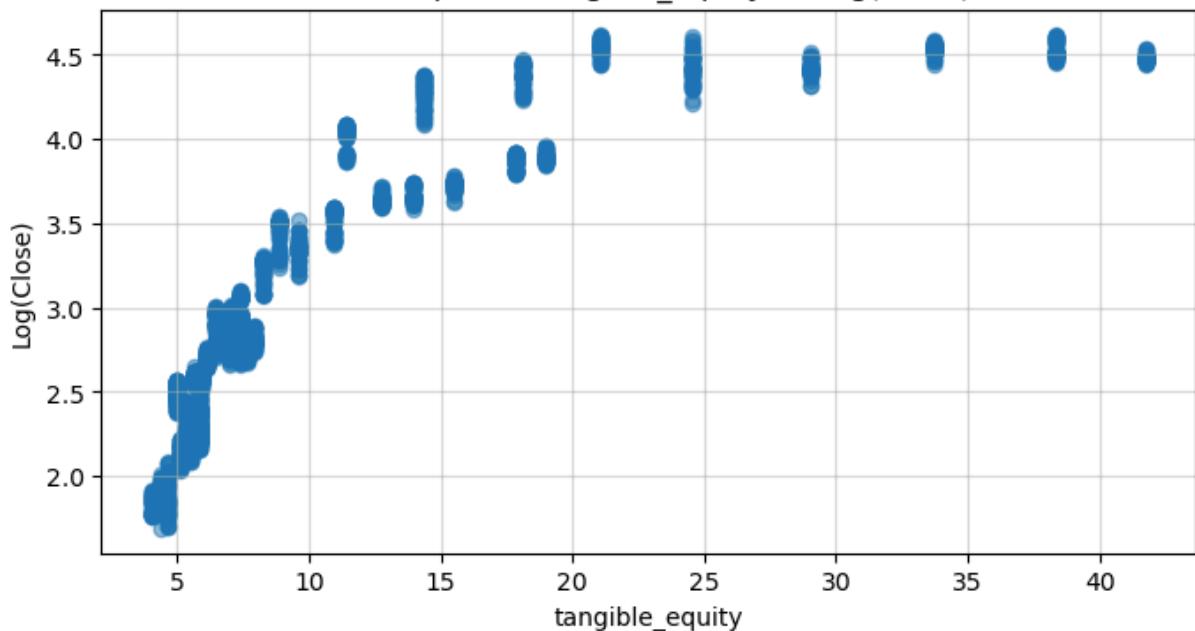
Scatterplot of return\_on\_equity vs Log(Close)



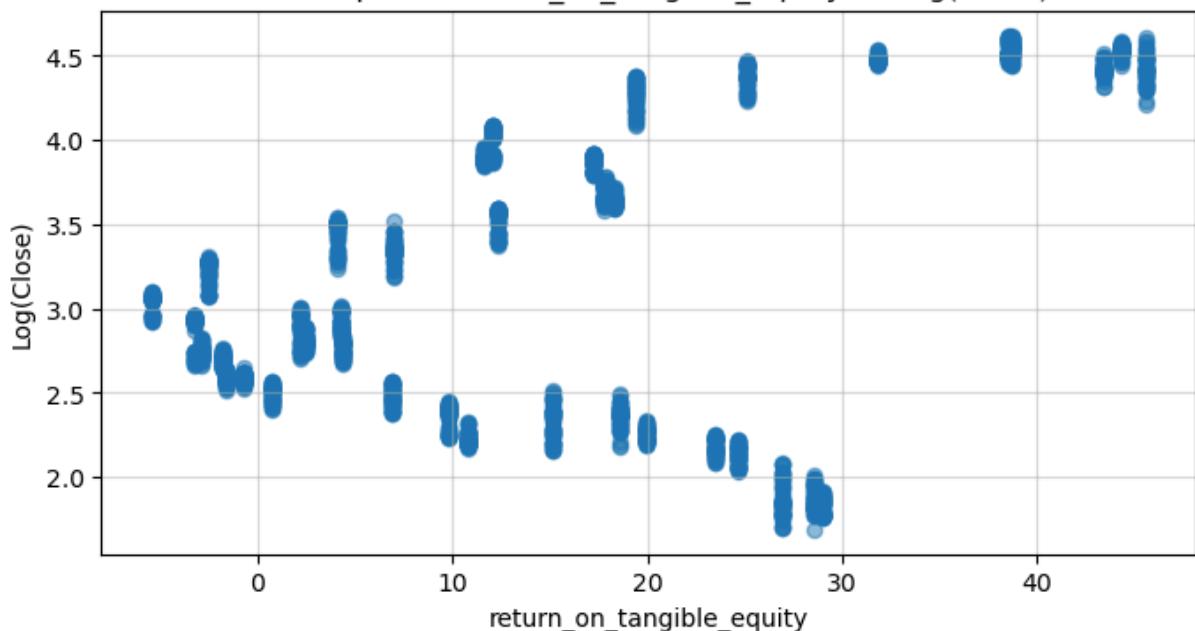
Scatterplot of return\_on\_assets vs Log(Close)



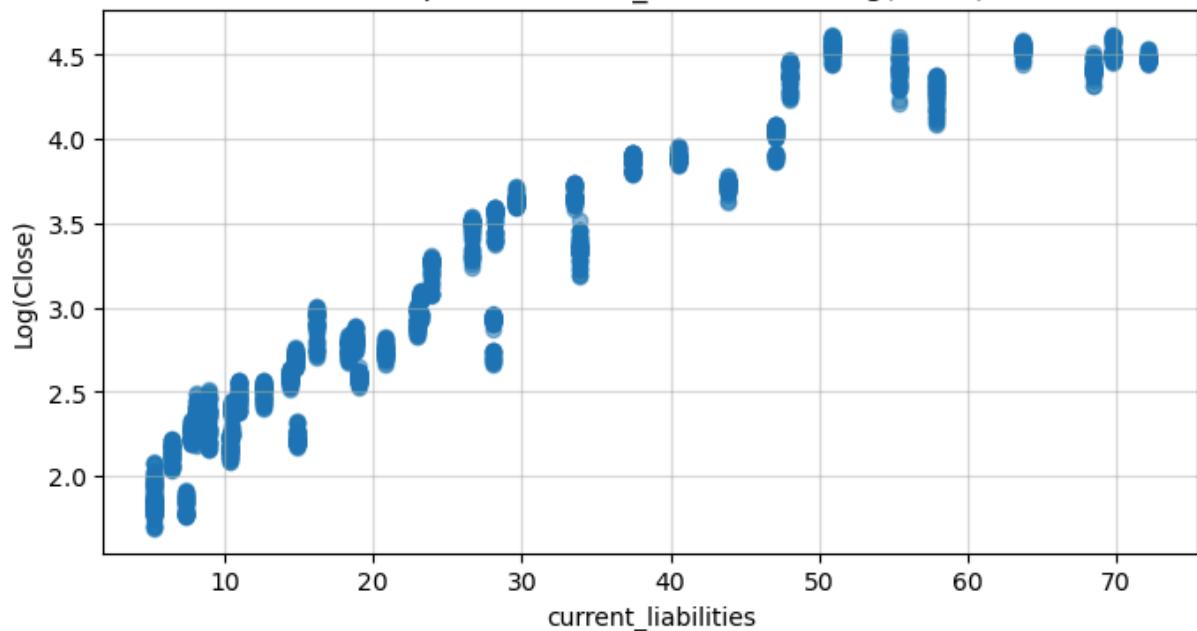
Scatterplot of tangible\_equity vs Log(Close)



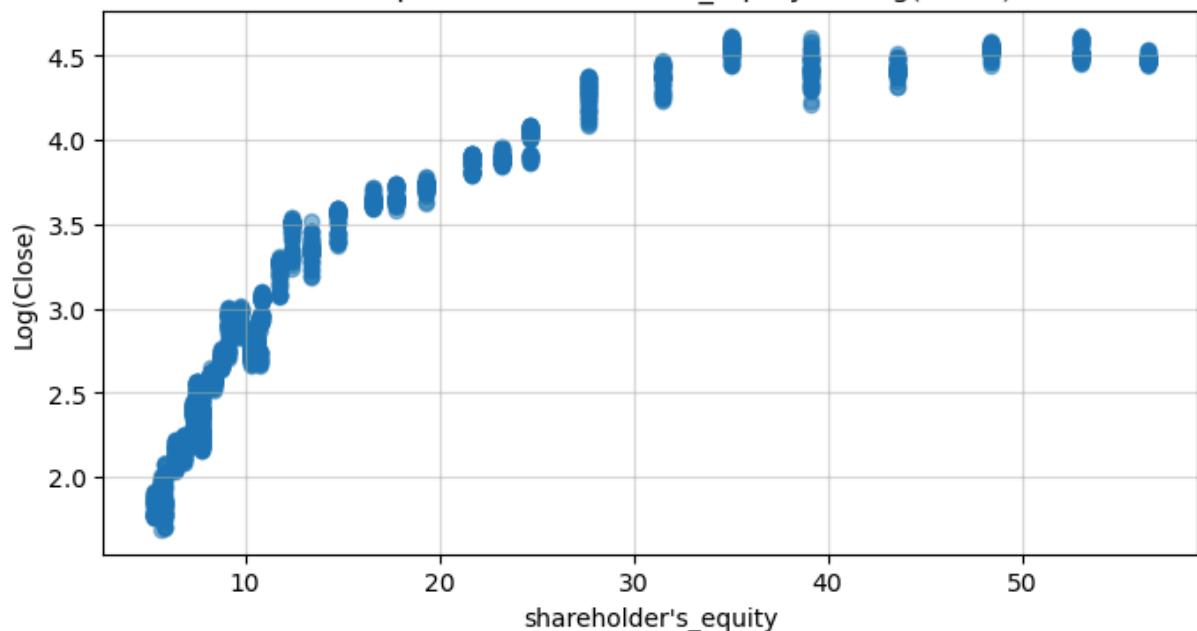
Scatterplot of return\_on\_tangible\_equity vs Log(Close)



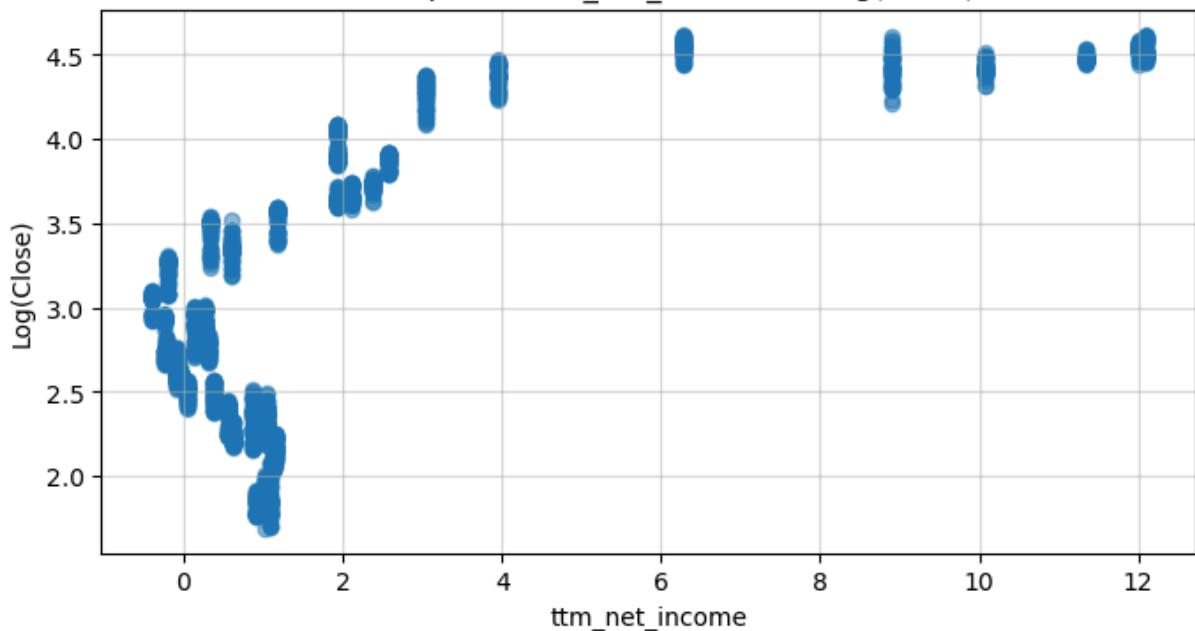
Scatterplot of current\_liabilities vs Log(Close)



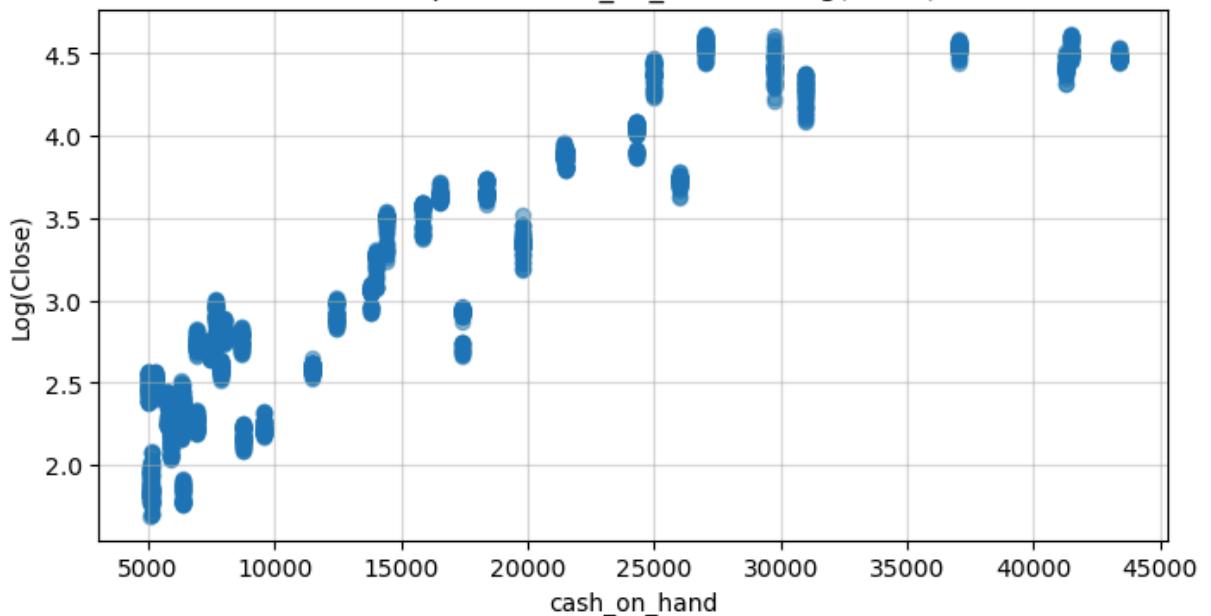
Scatterplot of shareholder's\_equity vs Log(Close)

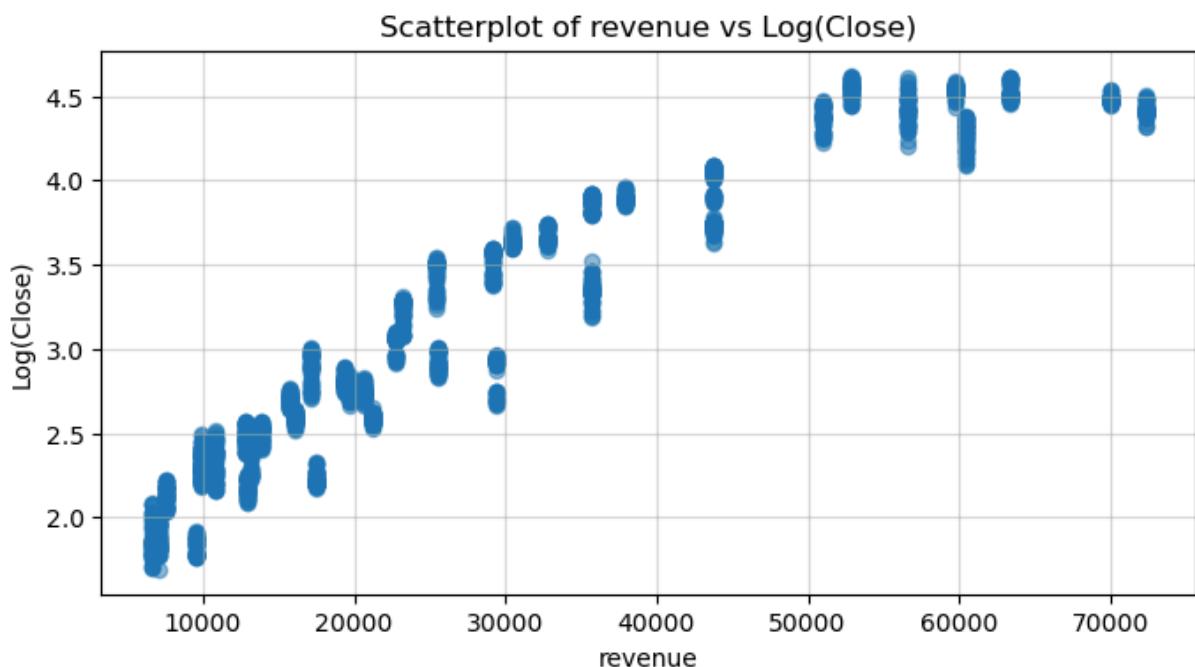
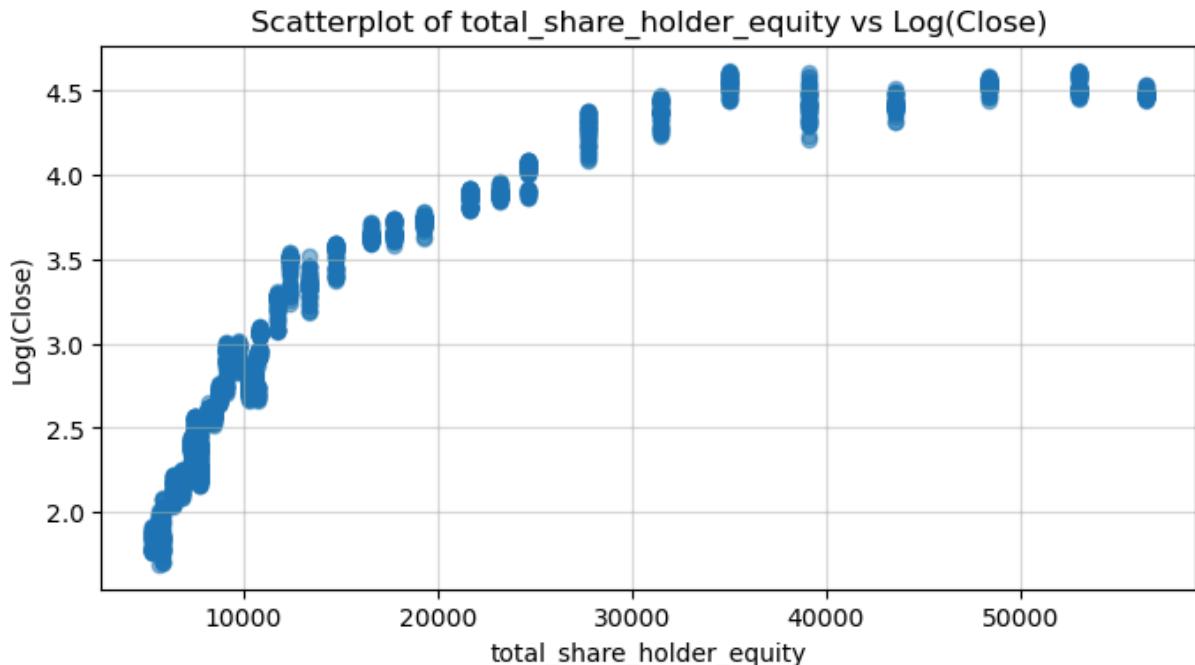


Scatterplot of ttm\_net\_income vs Log(Close)

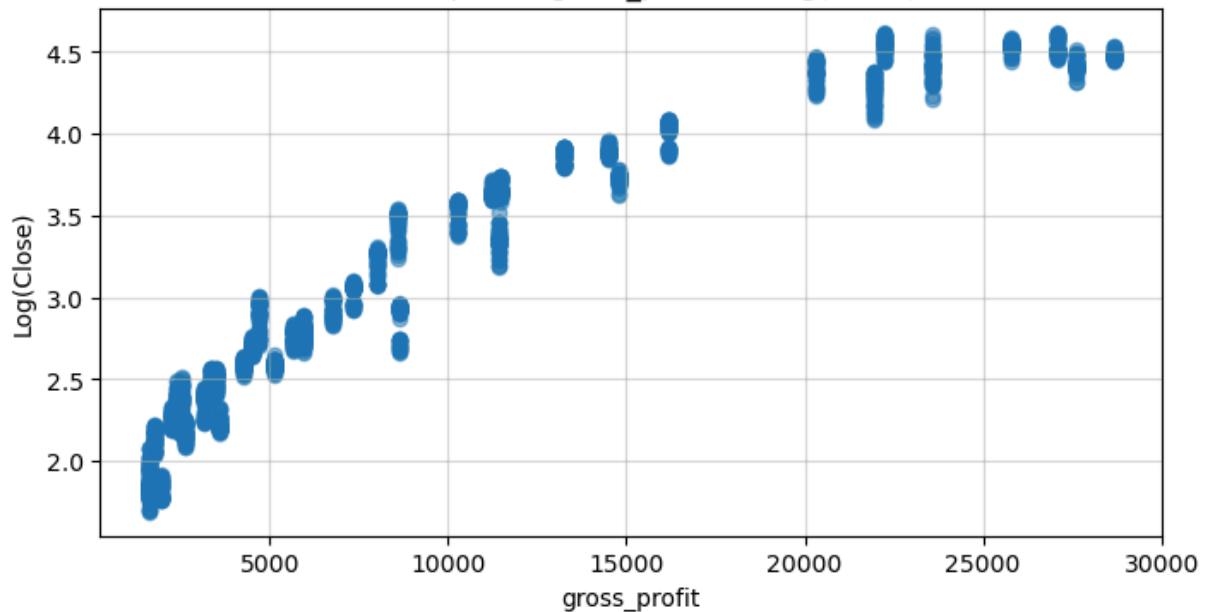


Scatterplot of cash\_on\_hand vs Log(Close)

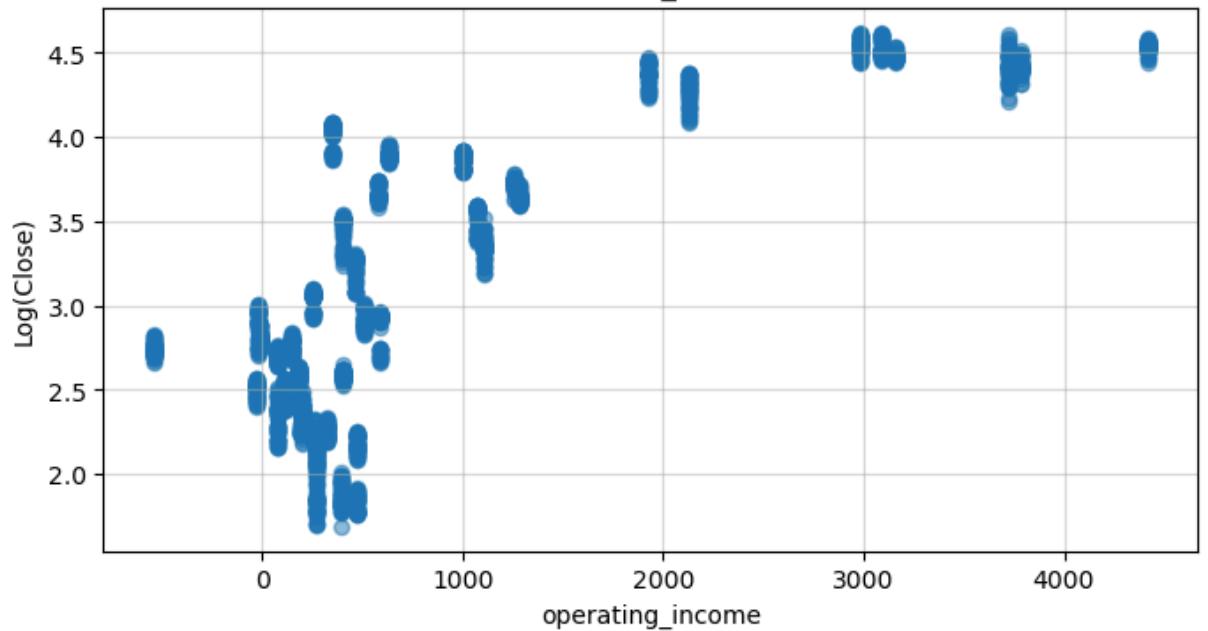




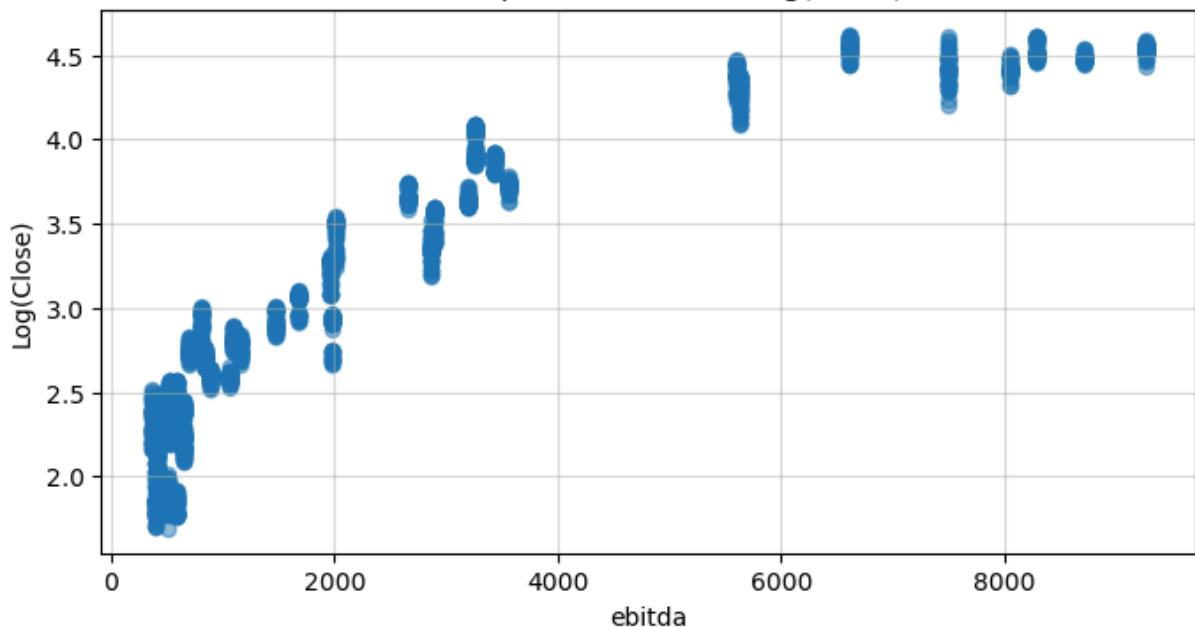
Scatterplot of gross\_profit vs Log(Close)



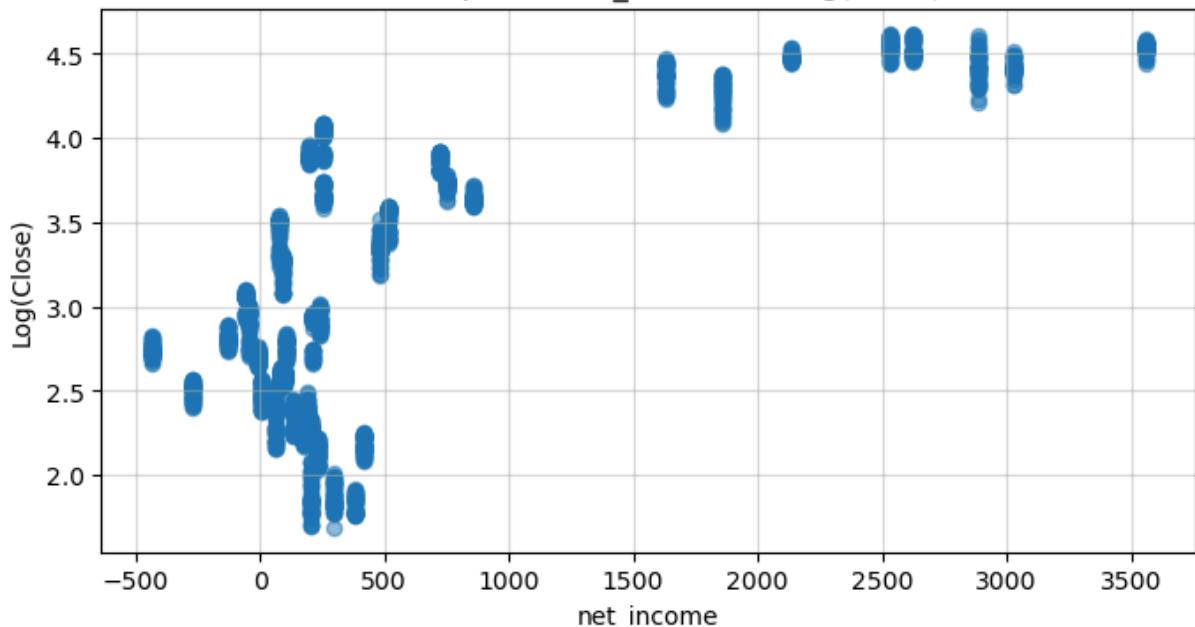
Scatterplot of operating\_income vs Log(Close)

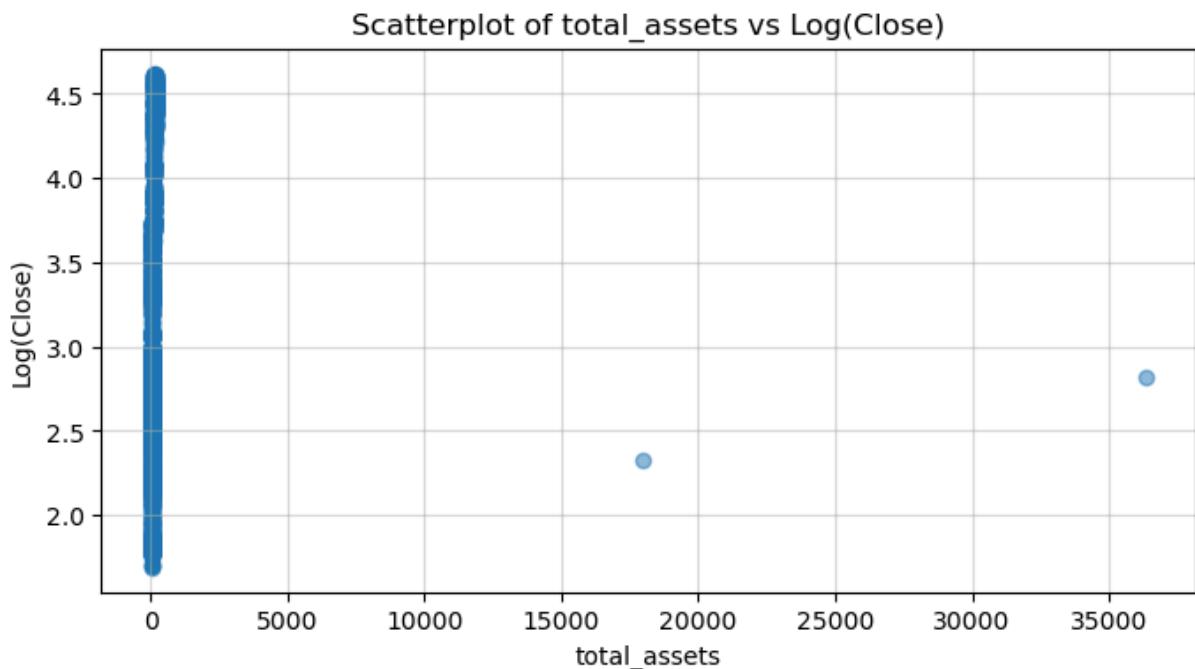
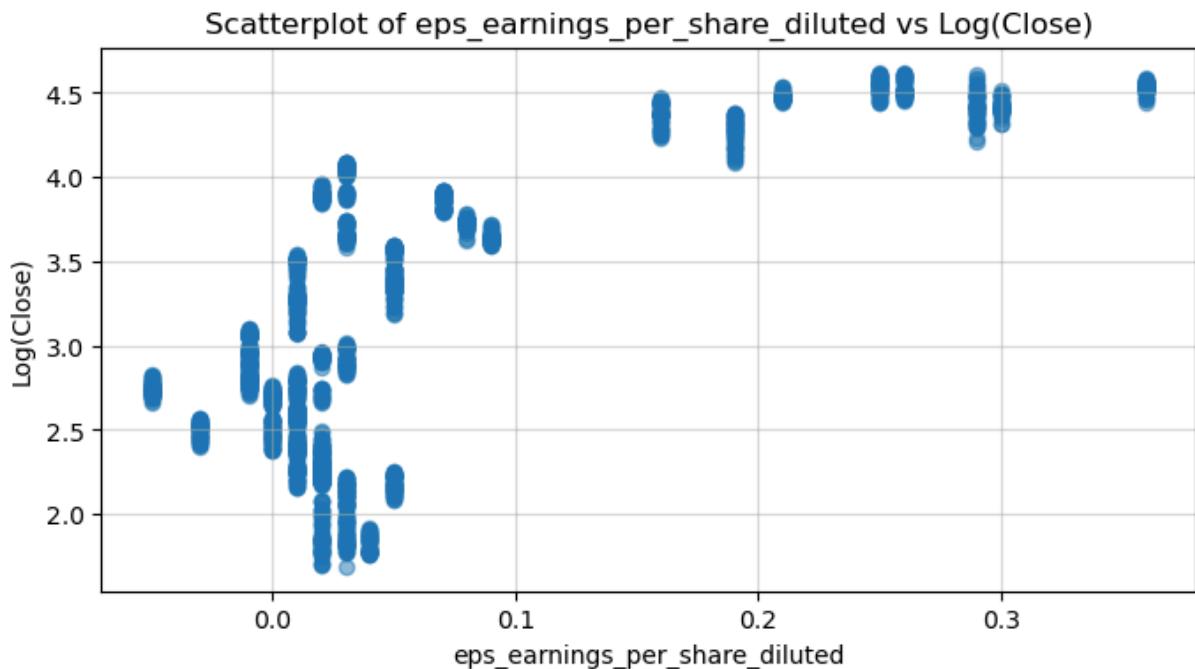


Scatterplot of ebitda vs Log(Close)

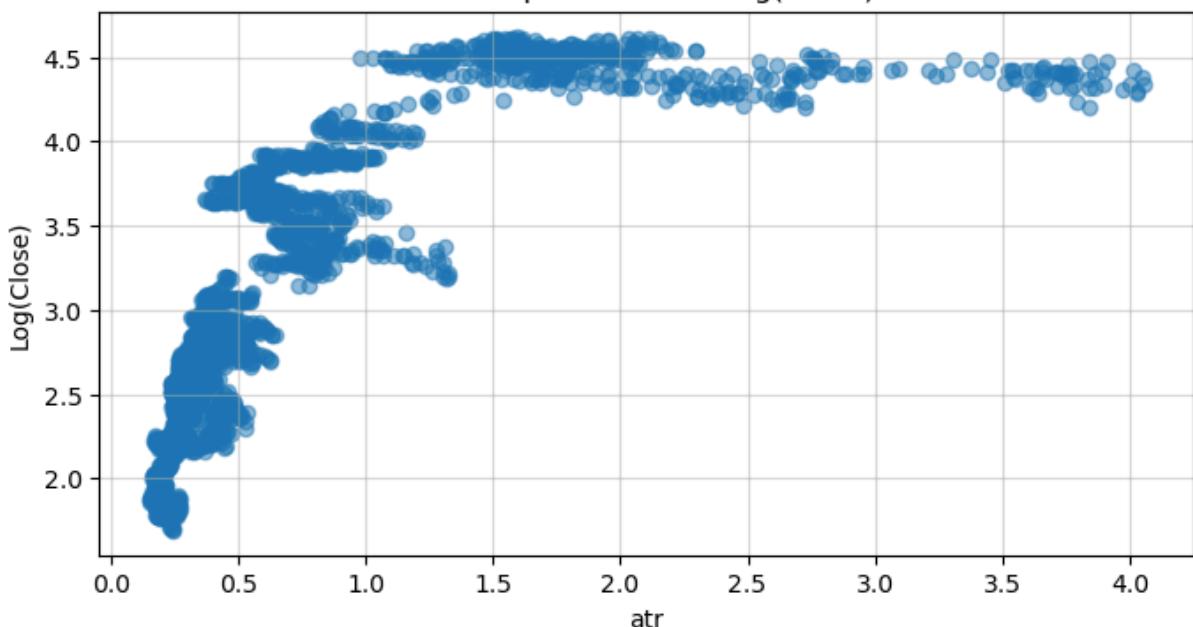


Scatterplot of net\_income vs Log(Close)

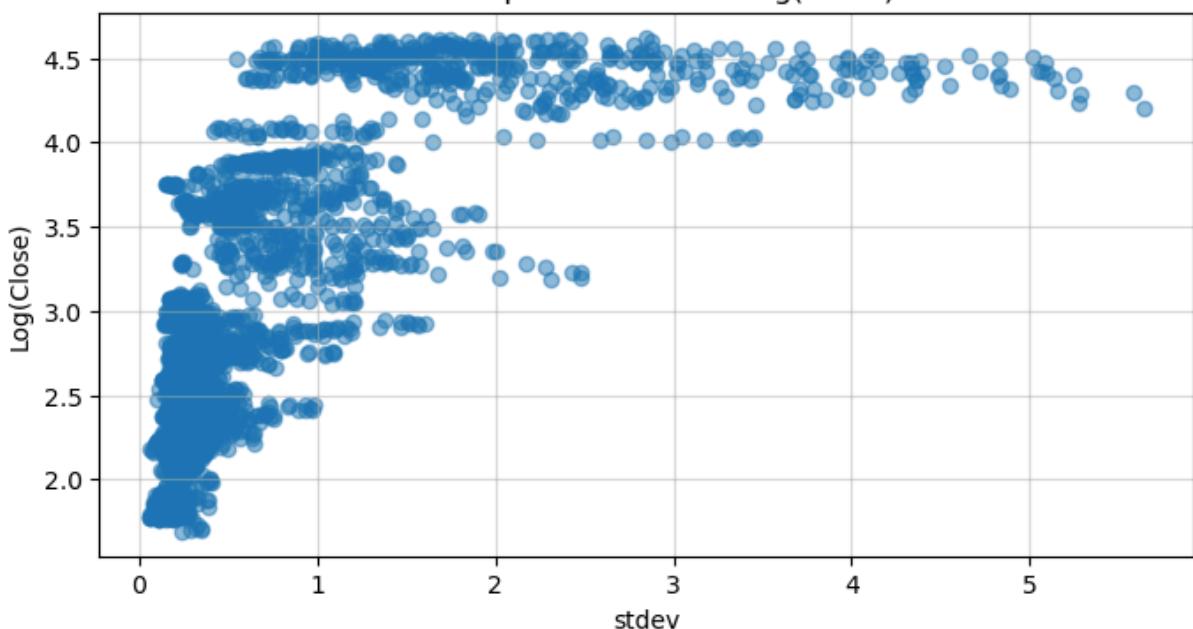




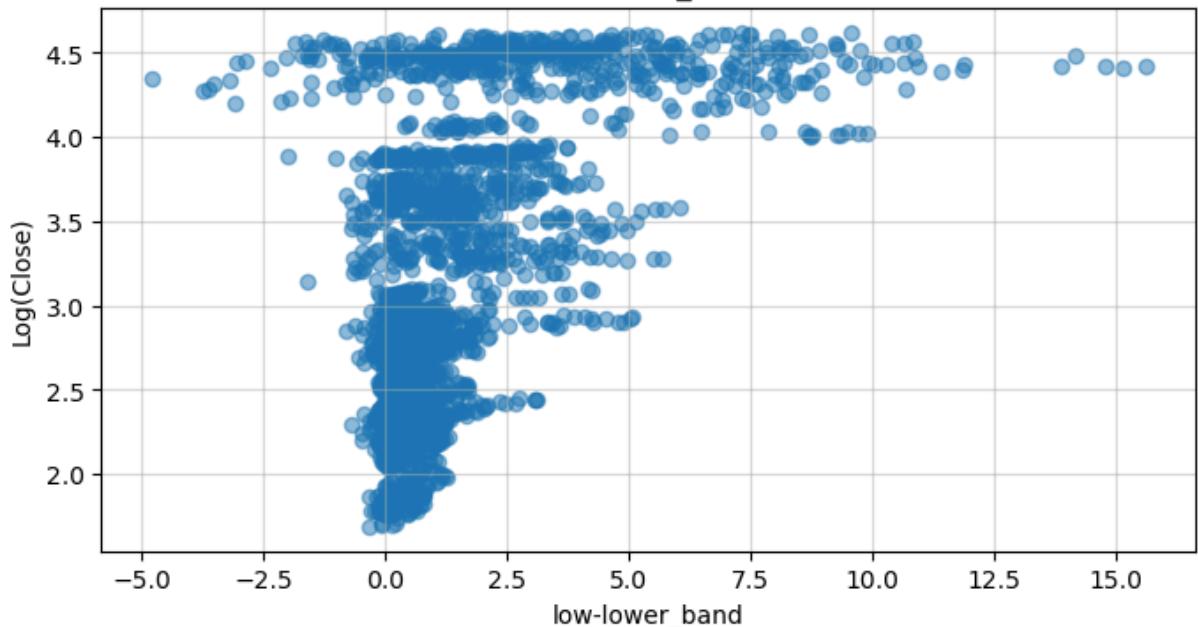
Scatterplot of atr vs Log(Close)



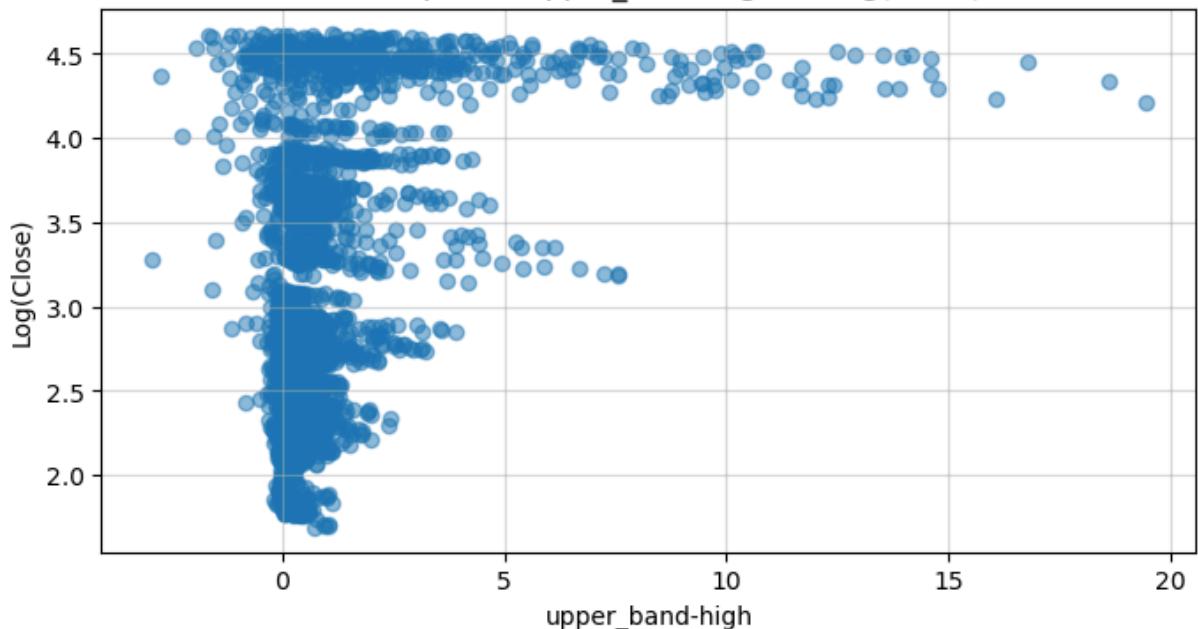
Scatterplot of stdev vs Log(Close)



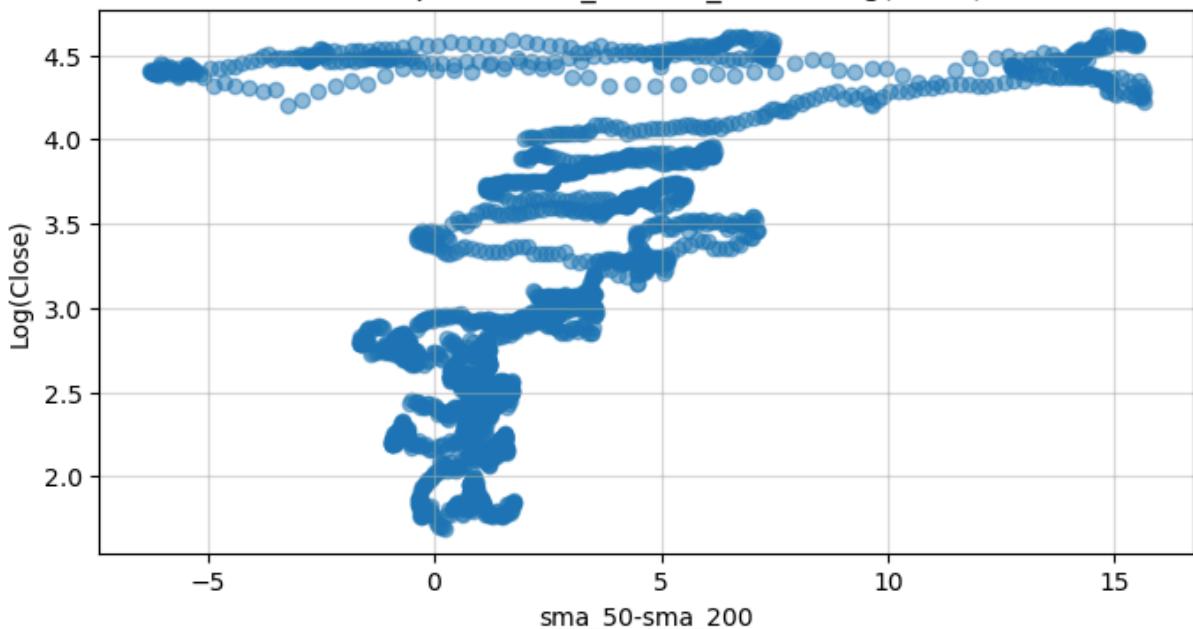
Scatterplot of low-lower\_band vs Log(Close)



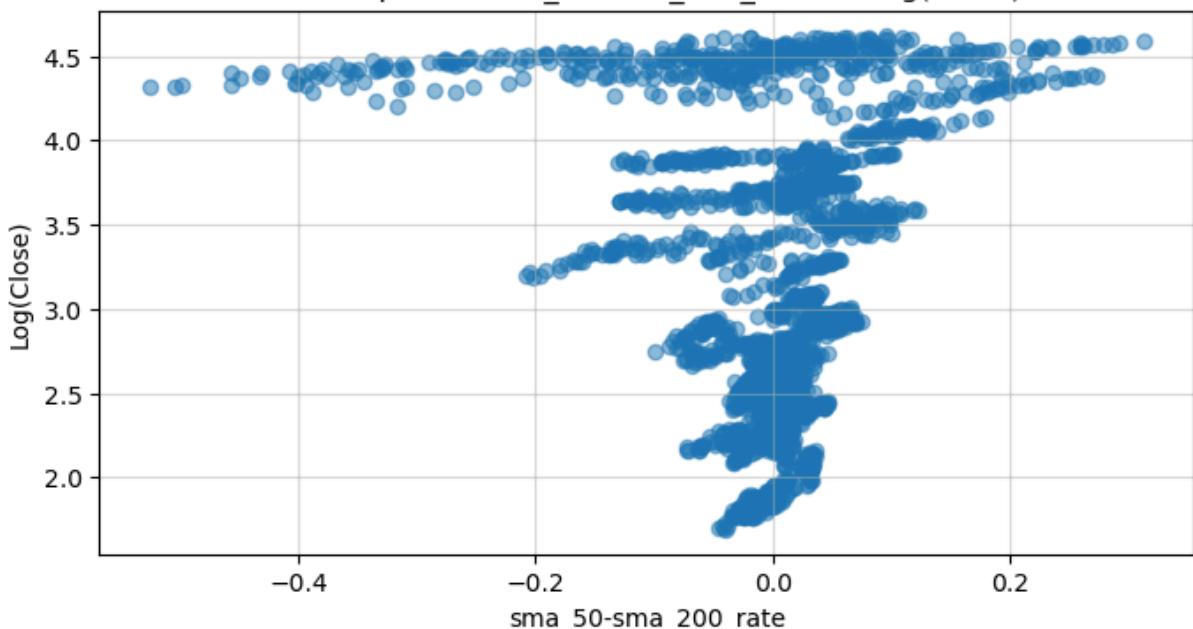
Scatterplot of upper\_band-high vs Log(Close)



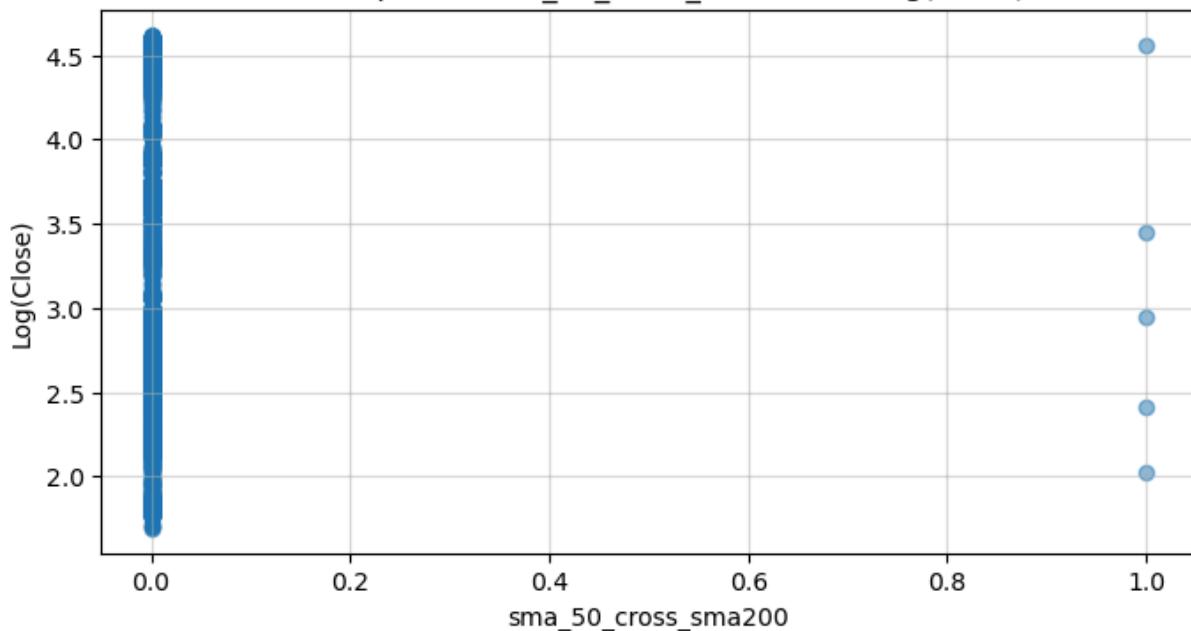
Scatterplot of sma\_50-sma\_200 vs Log(Close)



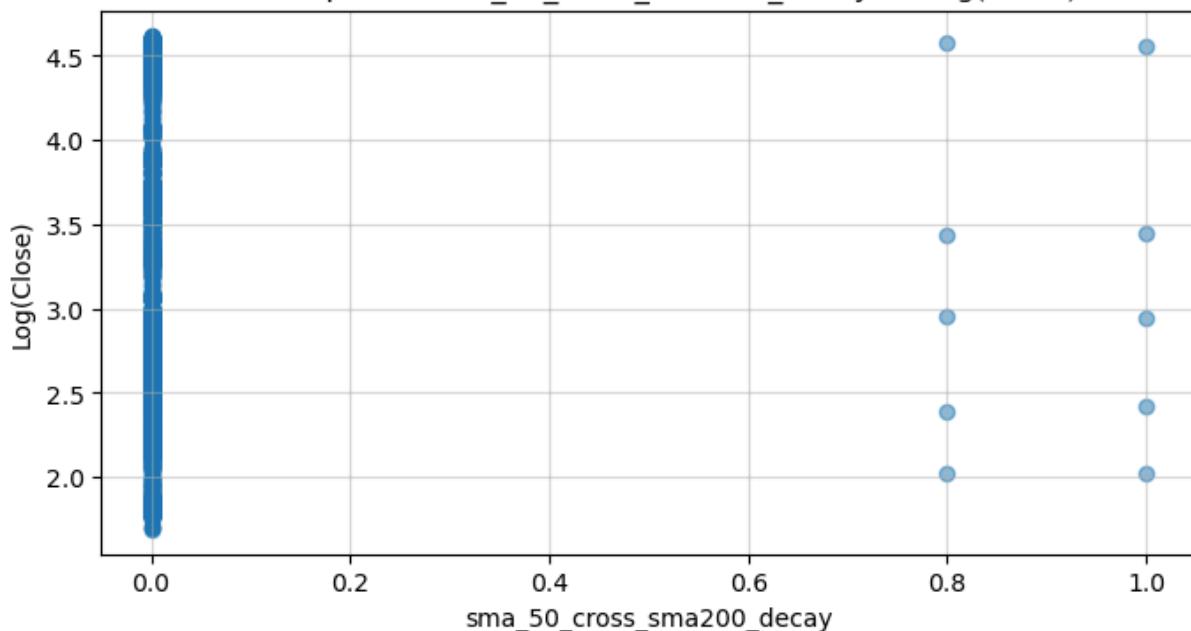
Scatterplot of sma\_50-sma\_200\_rate vs Log(Close)



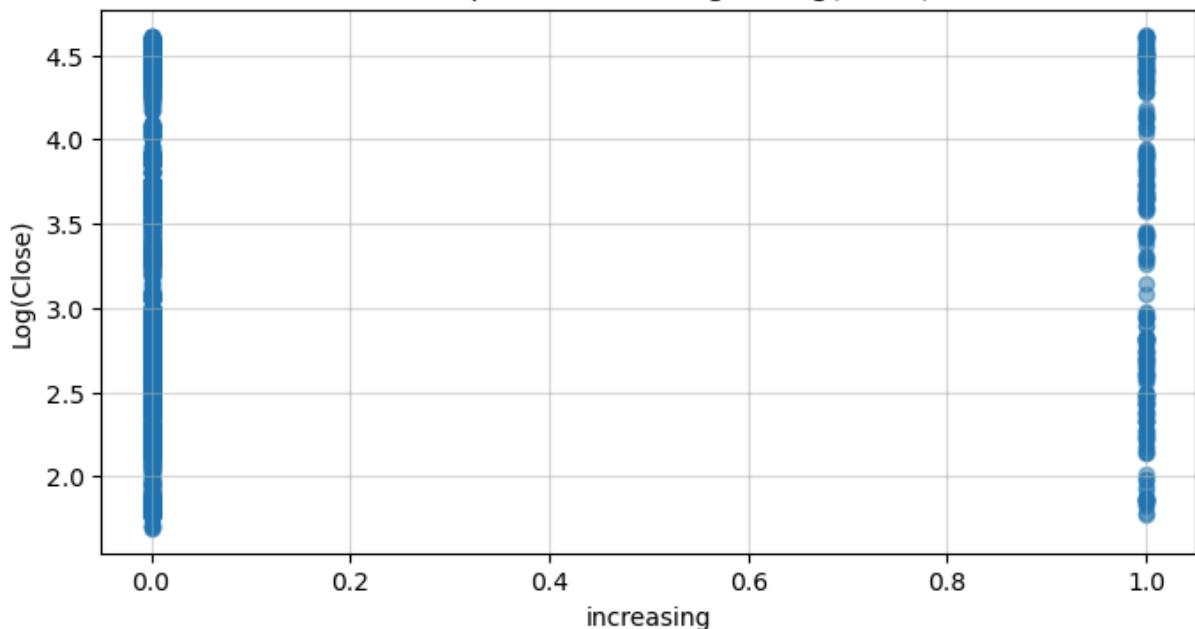
Scatterplot of sma\_50\_cross\_sma200 vs Log(Close)



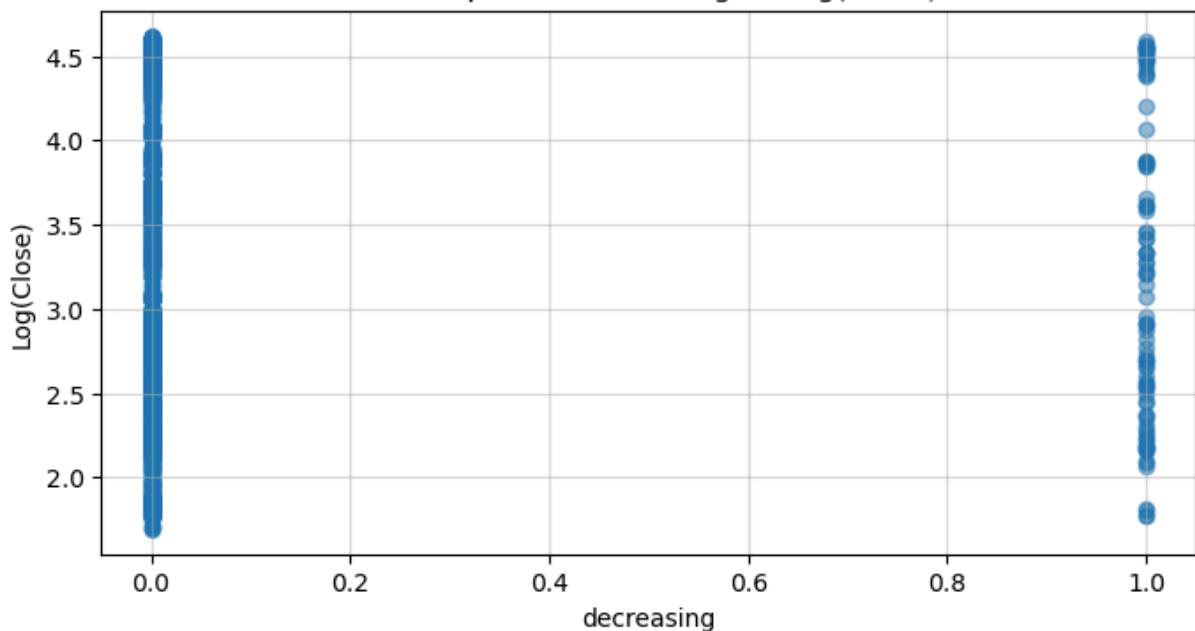
Scatterplot of sma\_50\_cross\_sma200\_decay vs Log(Close)



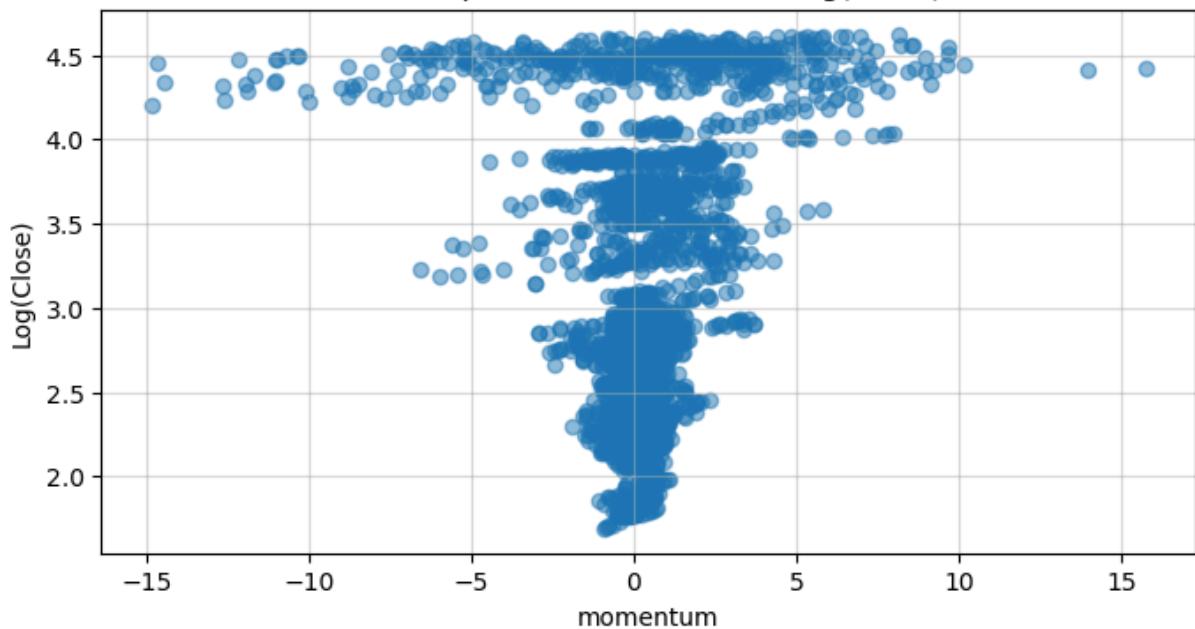
Scatterplot of increasing vs Log(Close)



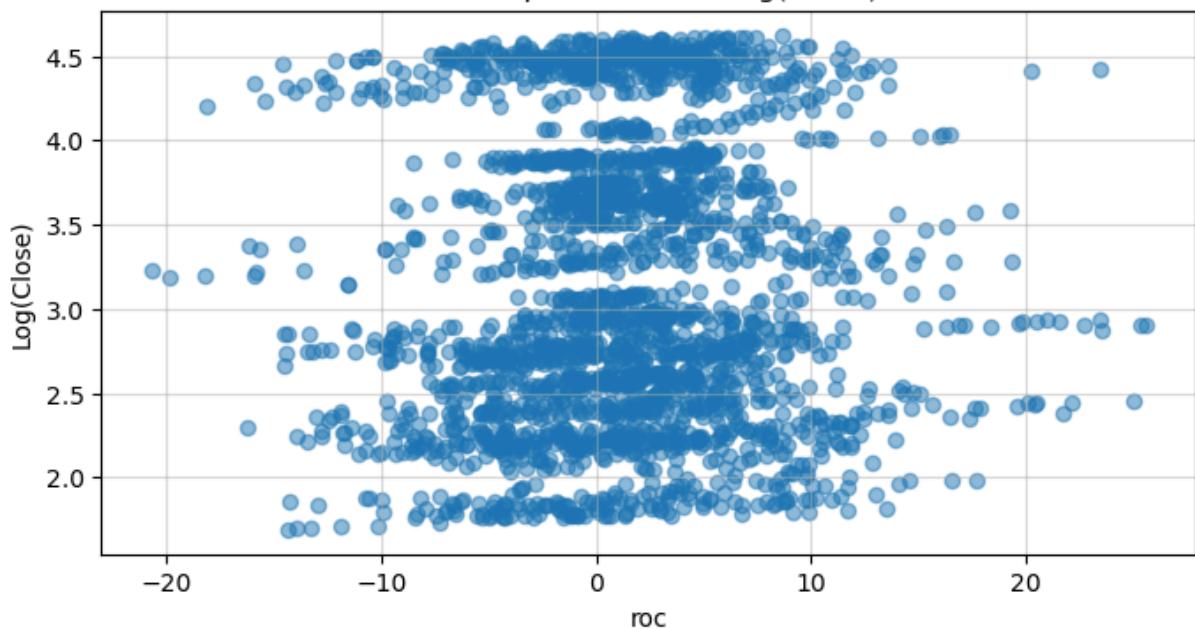
Scatterplot of decreasing vs Log(Close)



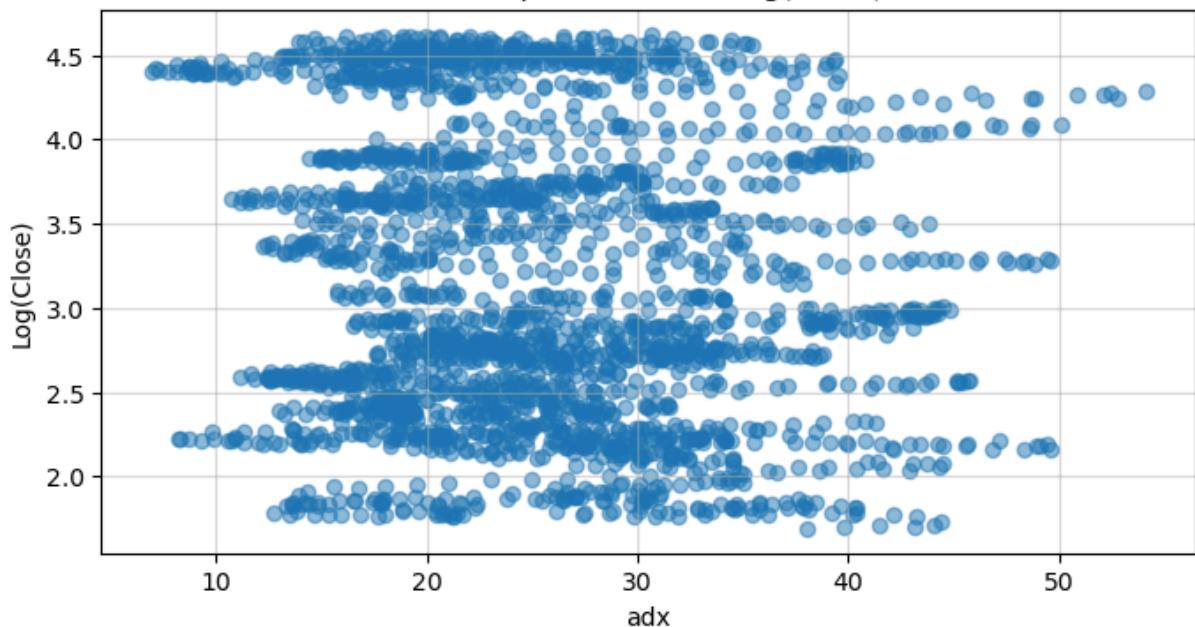
Scatterplot of momentum vs Log(Close)



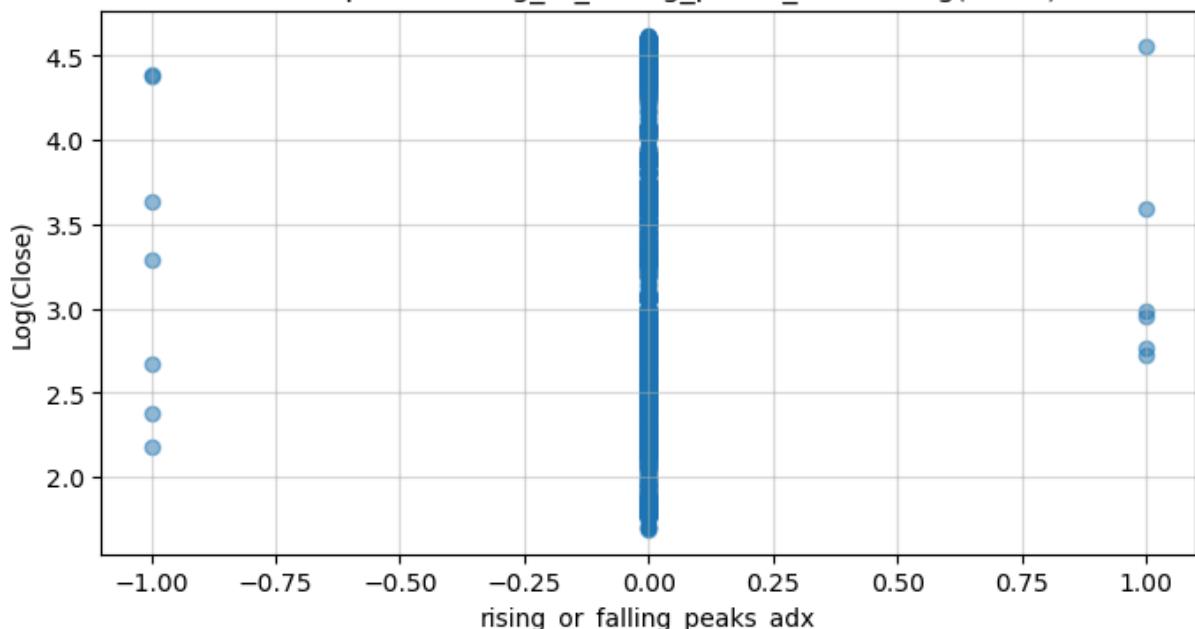
Scatterplot of roc vs Log(Close)



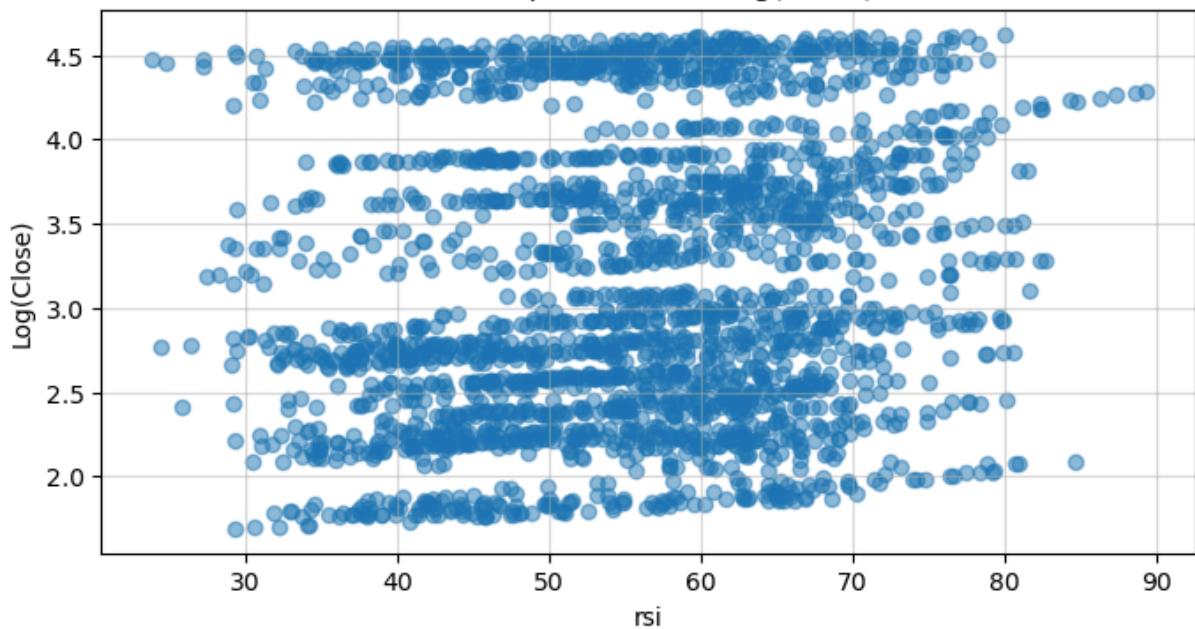
Scatterplot of adx vs Log(Close)



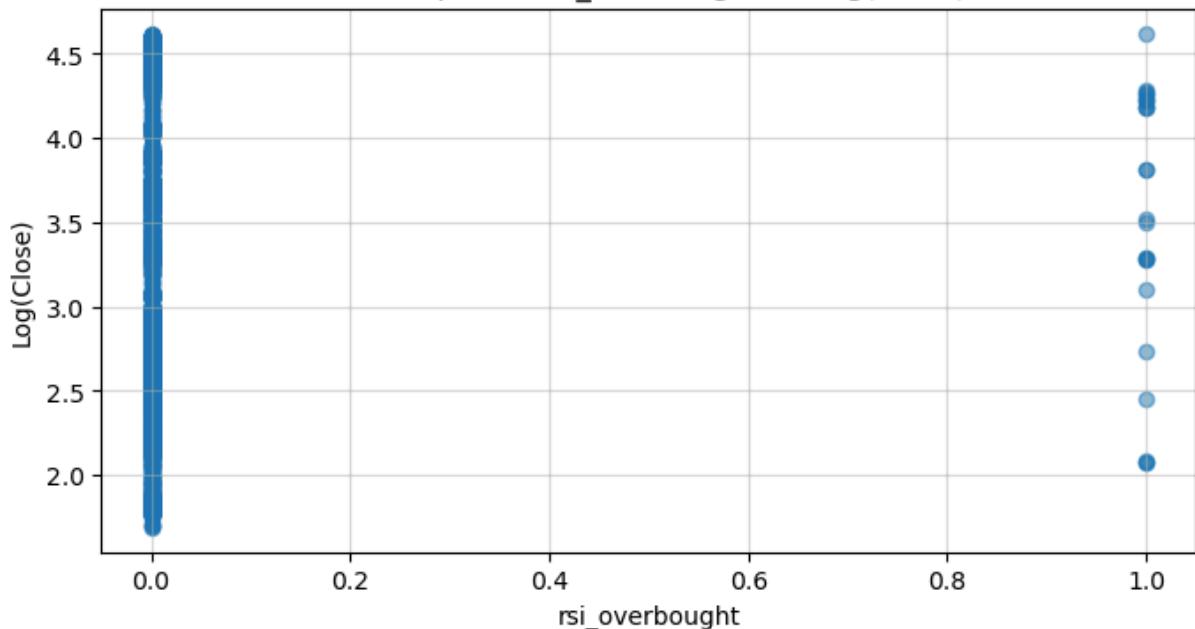
Scatterplot of rising\_or\_falling\_peaks\_adx vs Log(Close)



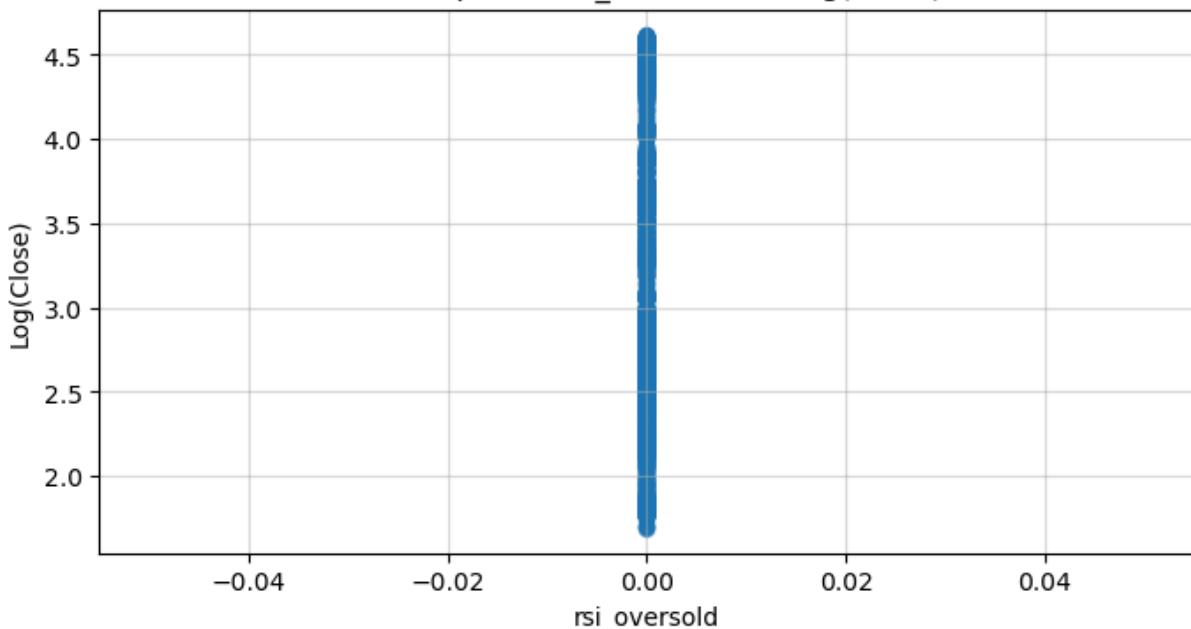
Scatterplot of rsi vs Log(Close)



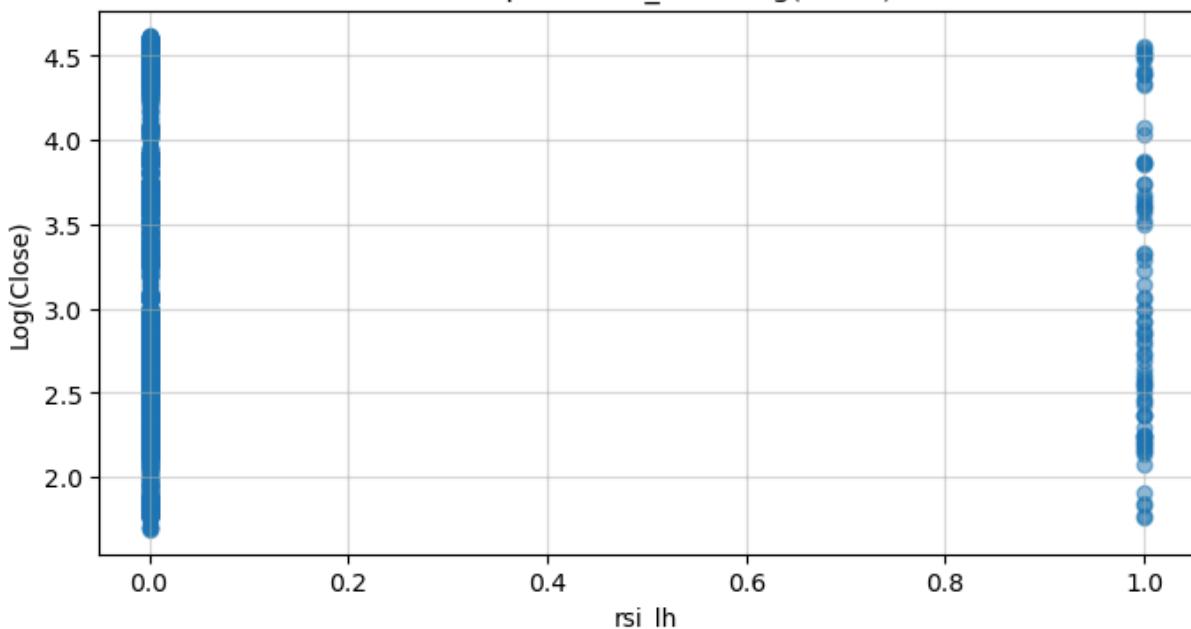
Scatterplot of rsi\_overbought vs Log(Close)



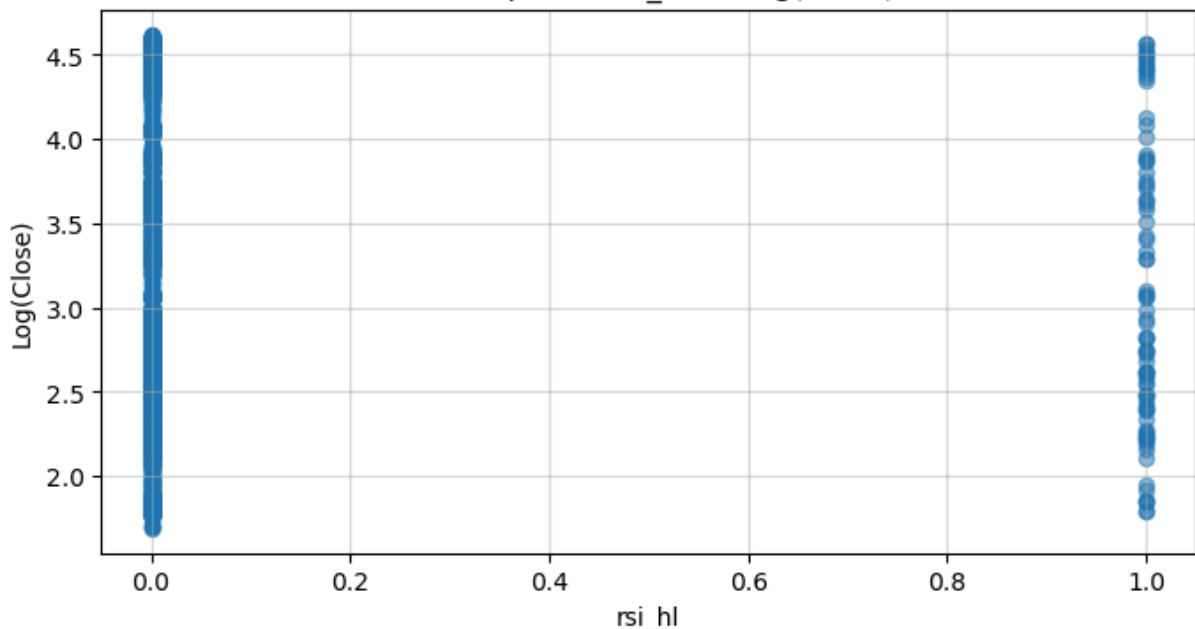
Scatterplot of rsi\_oversold vs Log(Close)



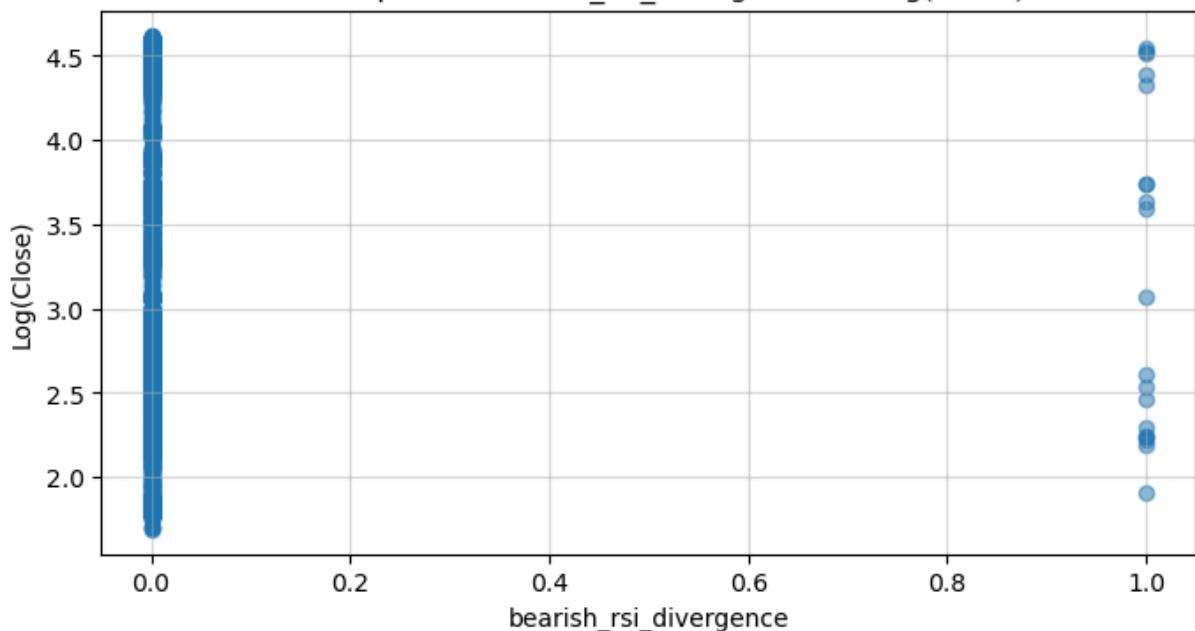
Scatterplot of rsi\_lh vs Log(Close)



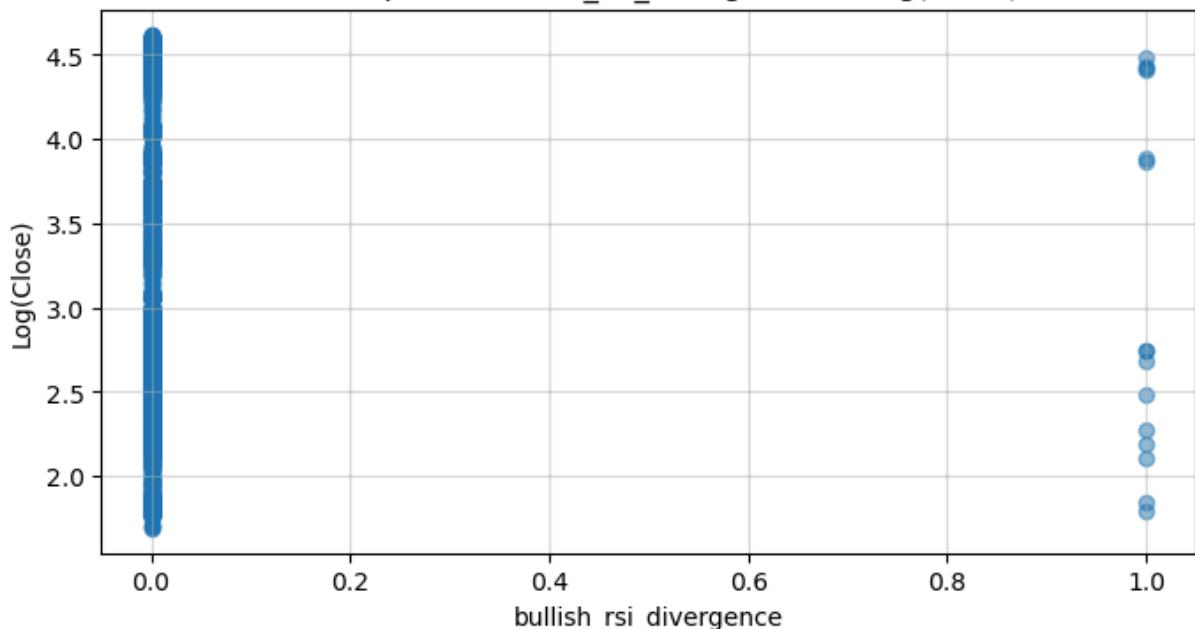
Scatterplot of rsi\_hl vs Log(Close)



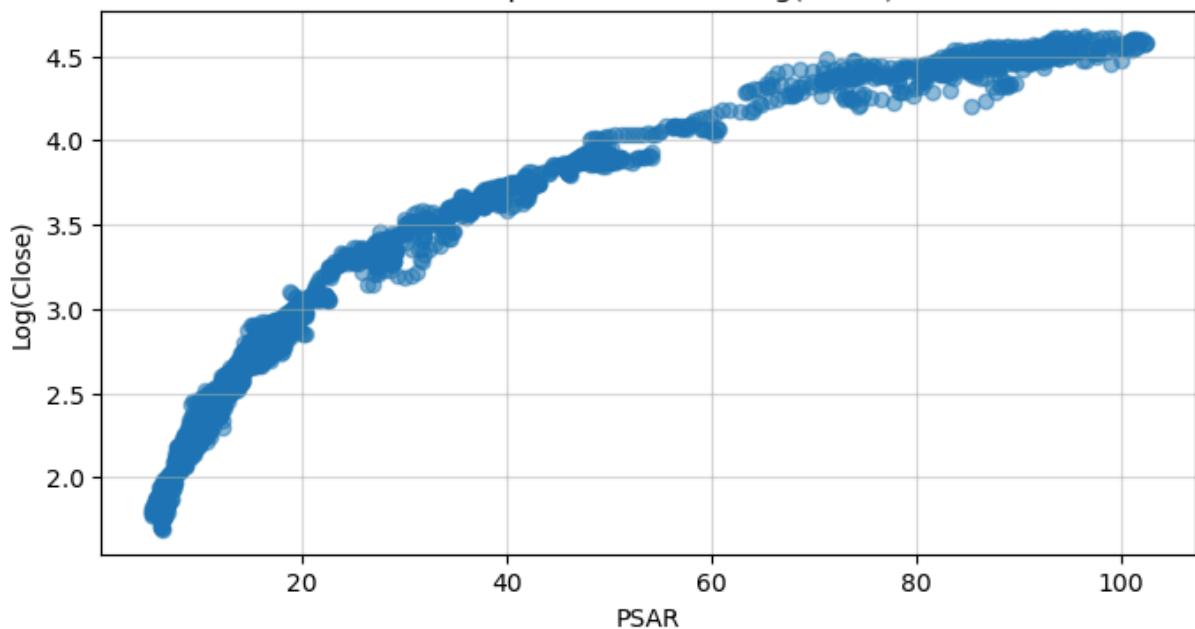
Scatterplot of bearish\_rsi\_divergence vs Log(Close)



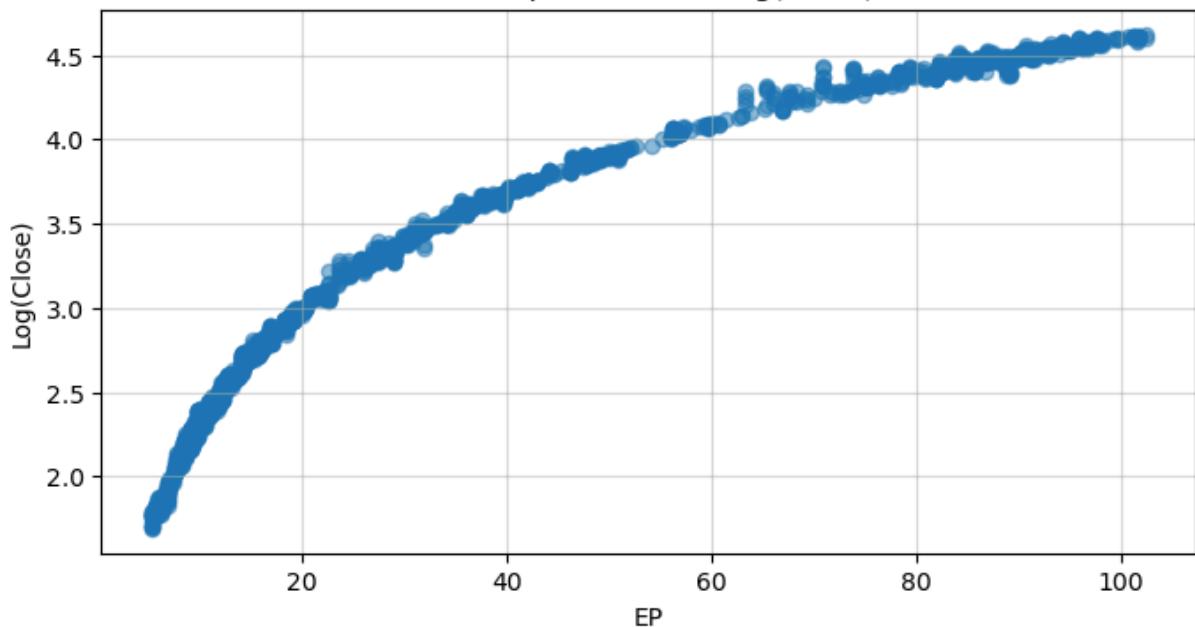
Scatterplot of bullish\_rsi\_divergence vs Log(Close)



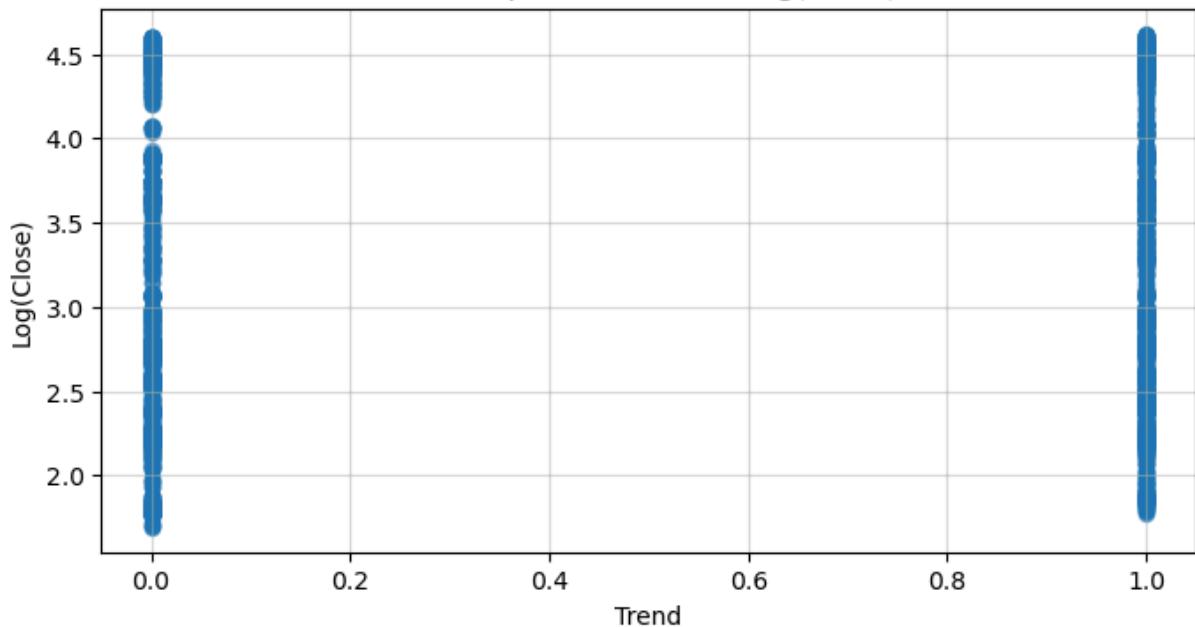
Scatterplot of PSAR vs Log(Close)



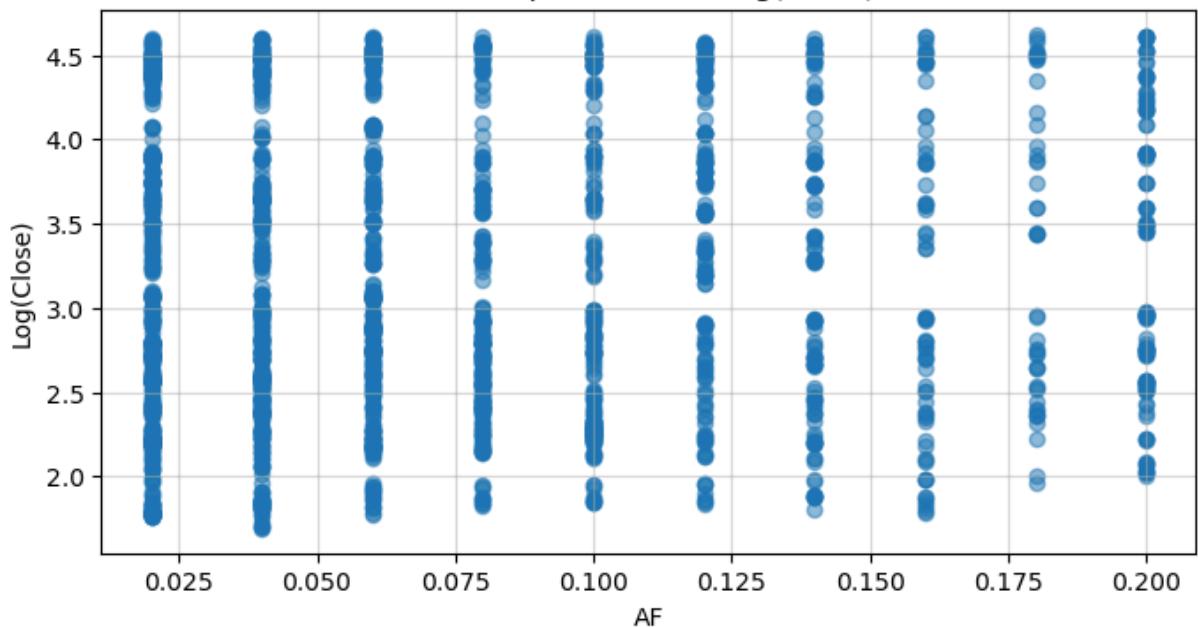
Scatterplot of EP vs Log(Close)



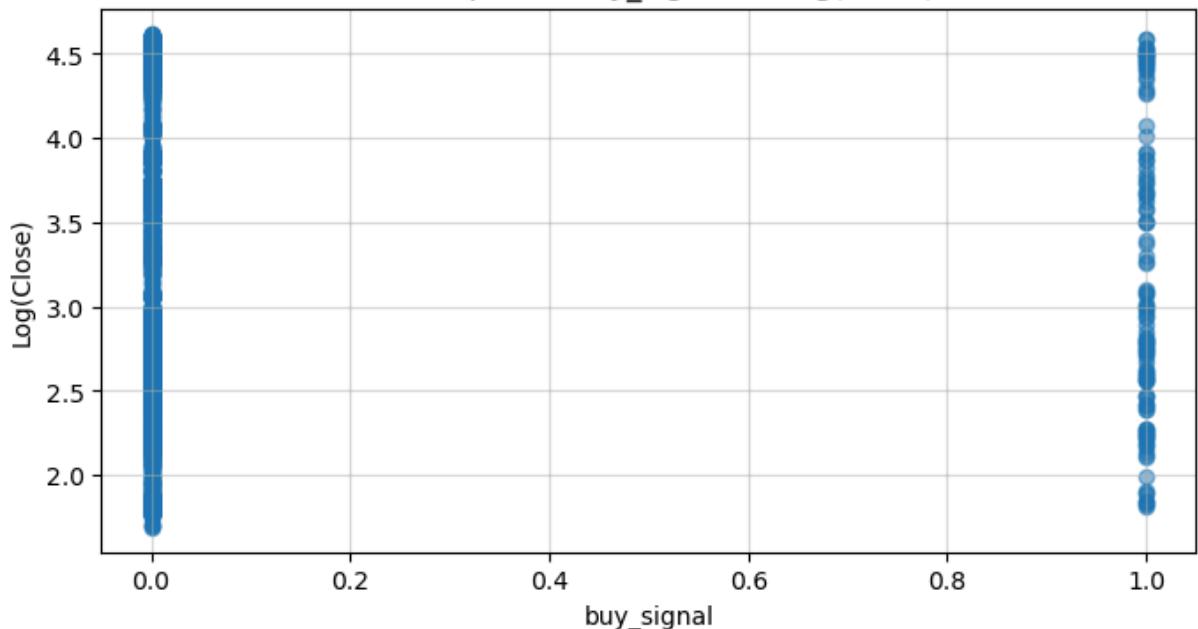
Scatterplot of Trend vs Log(Close)



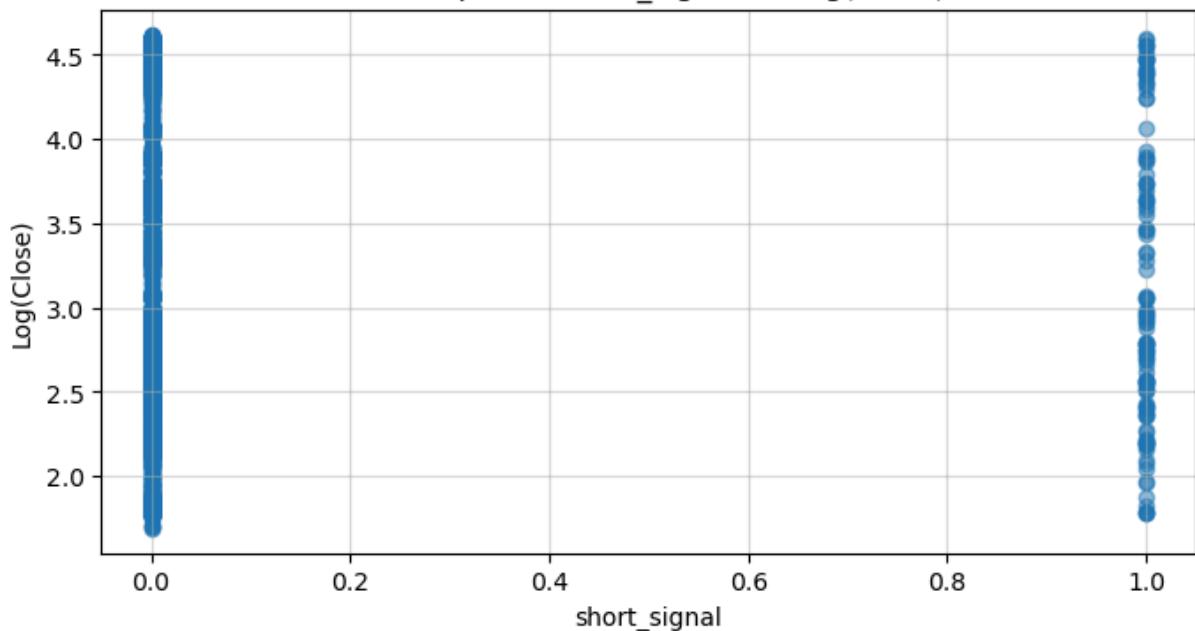
Scatterplot of AF vs Log(Close)



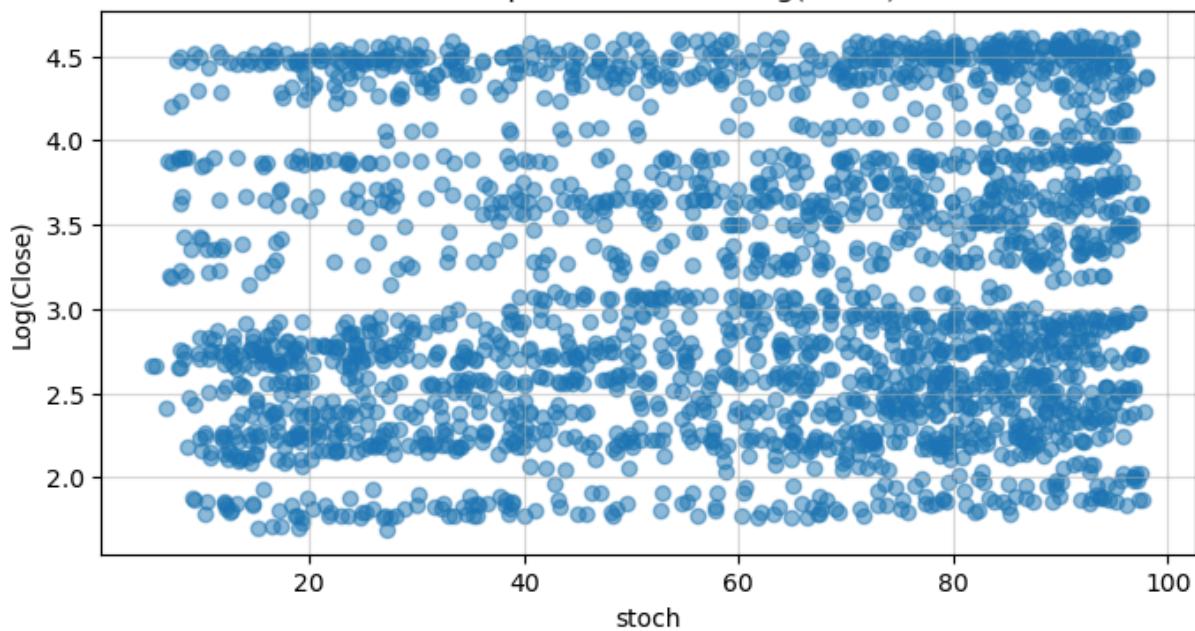
Scatterplot of buy\_signal vs Log(Close)



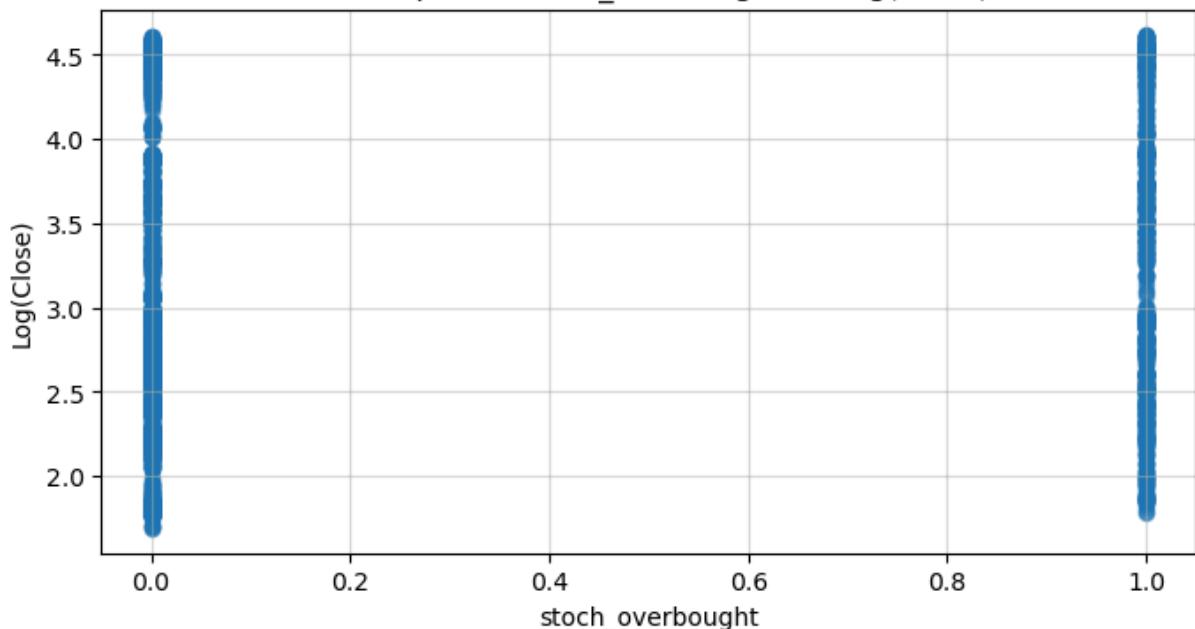
Scatterplot of short\_signal vs Log(Close)



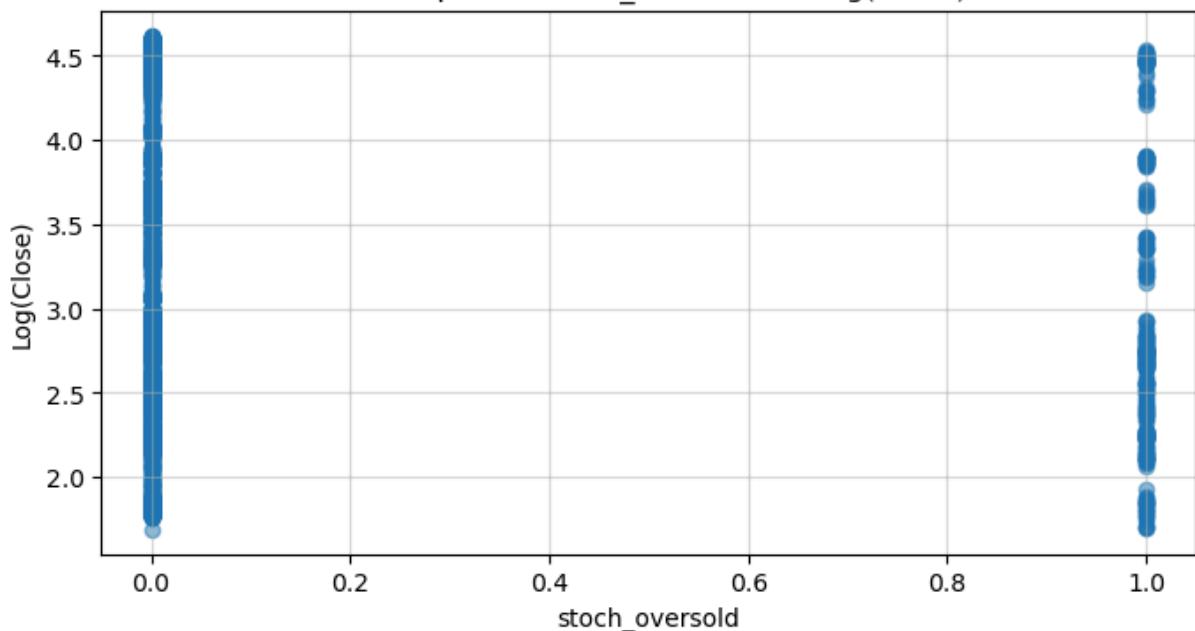
Scatterplot of stoch vs Log(Close)



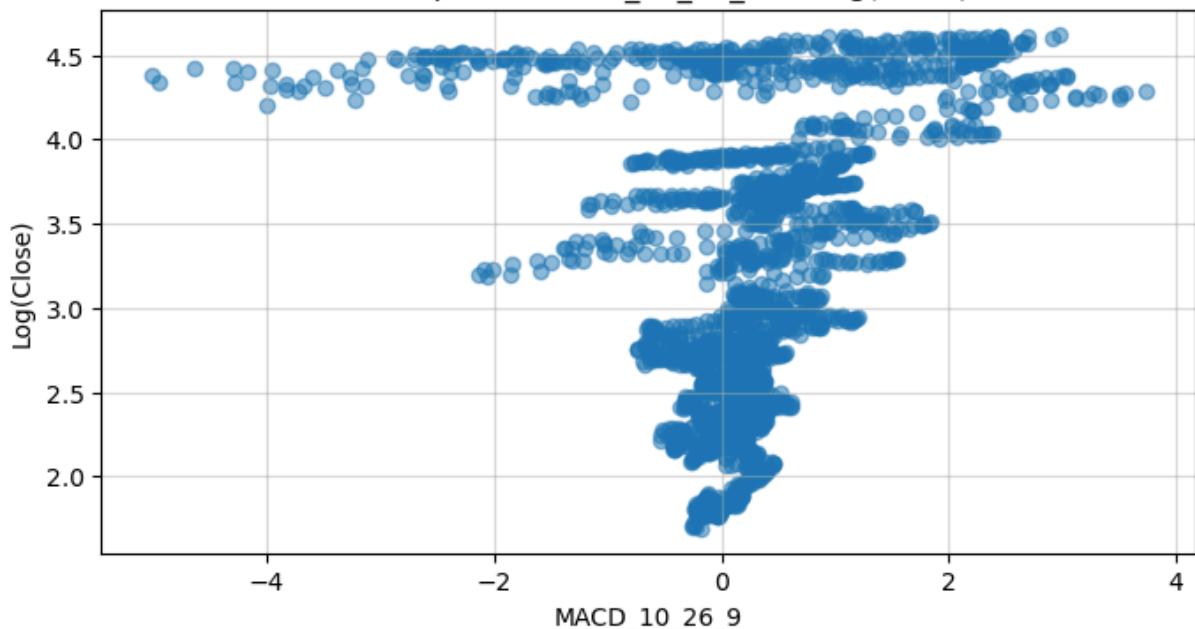
Scatterplot of stoch\_overbought vs Log(Close)



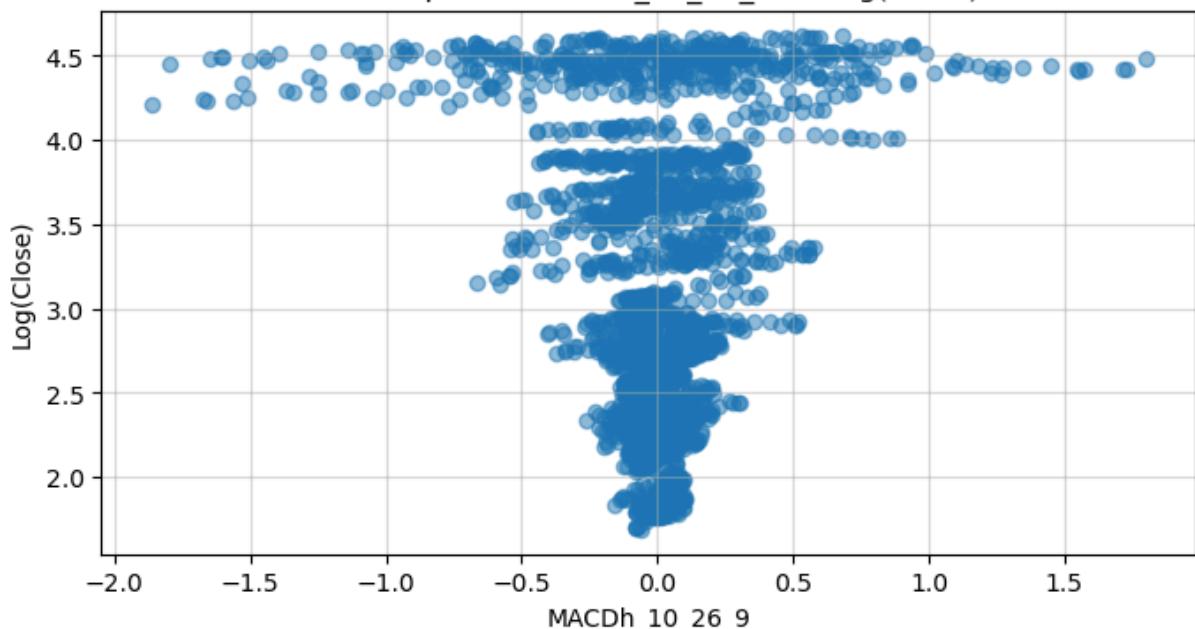
Scatterplot of stoch\_oversold vs Log(Close)



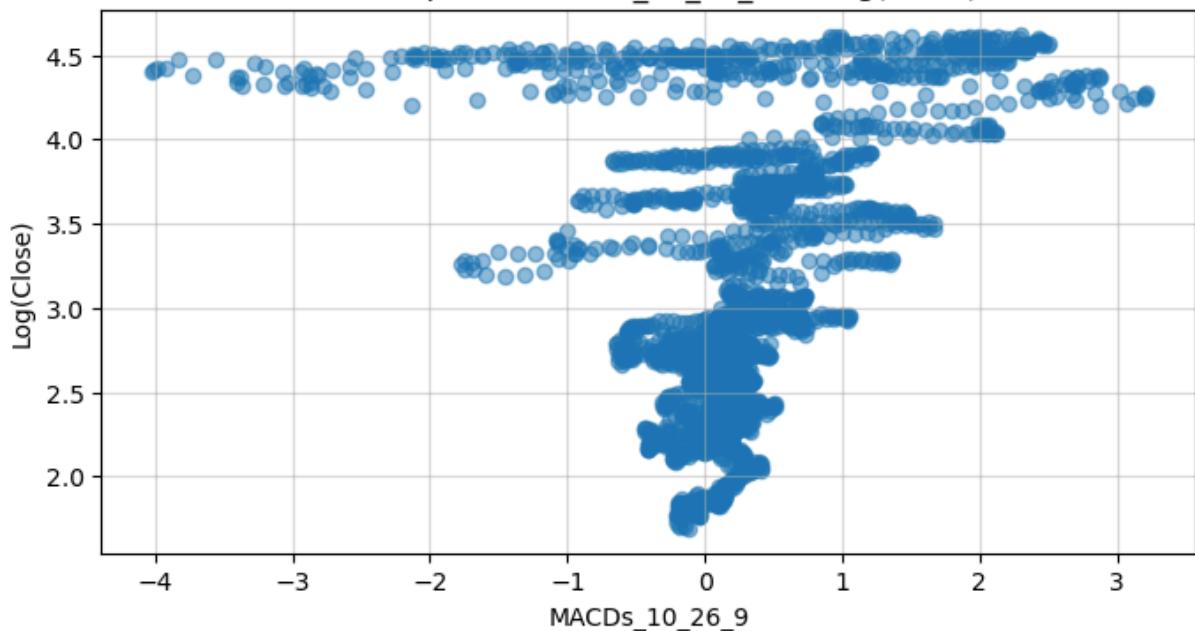
Scatterplot of MACD\_10\_26\_9 vs Log(Close)



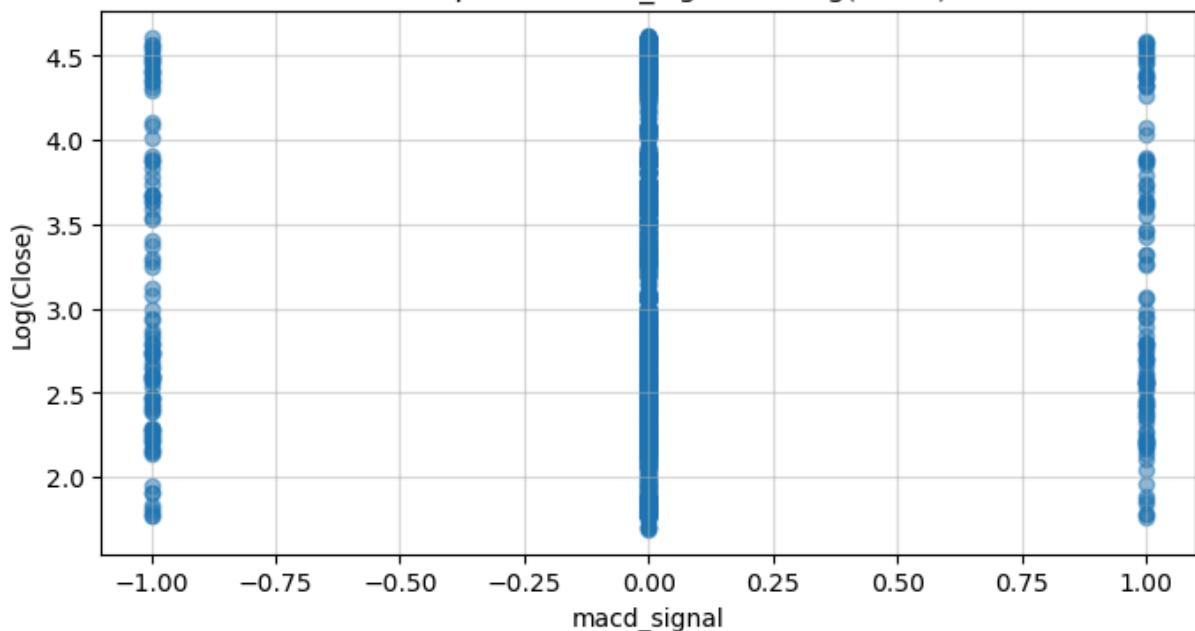
Scatterplot of MACDh\_10\_26\_9 vs Log(Close)



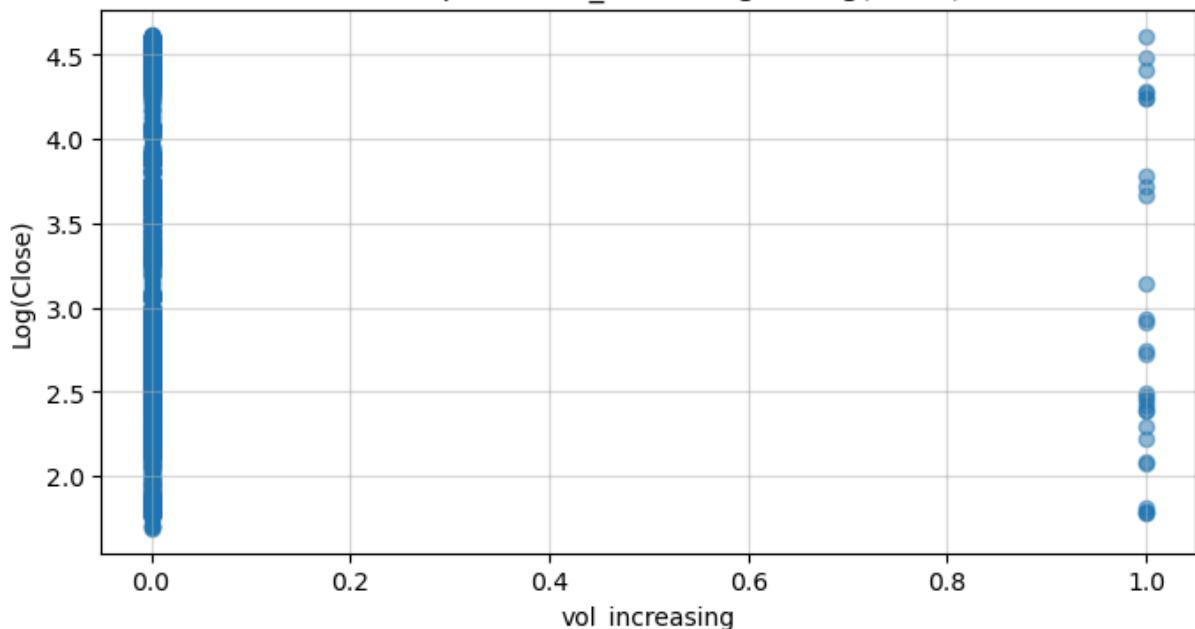
Scatterplot of MACDs\_10\_26\_9 vs Log(Close)



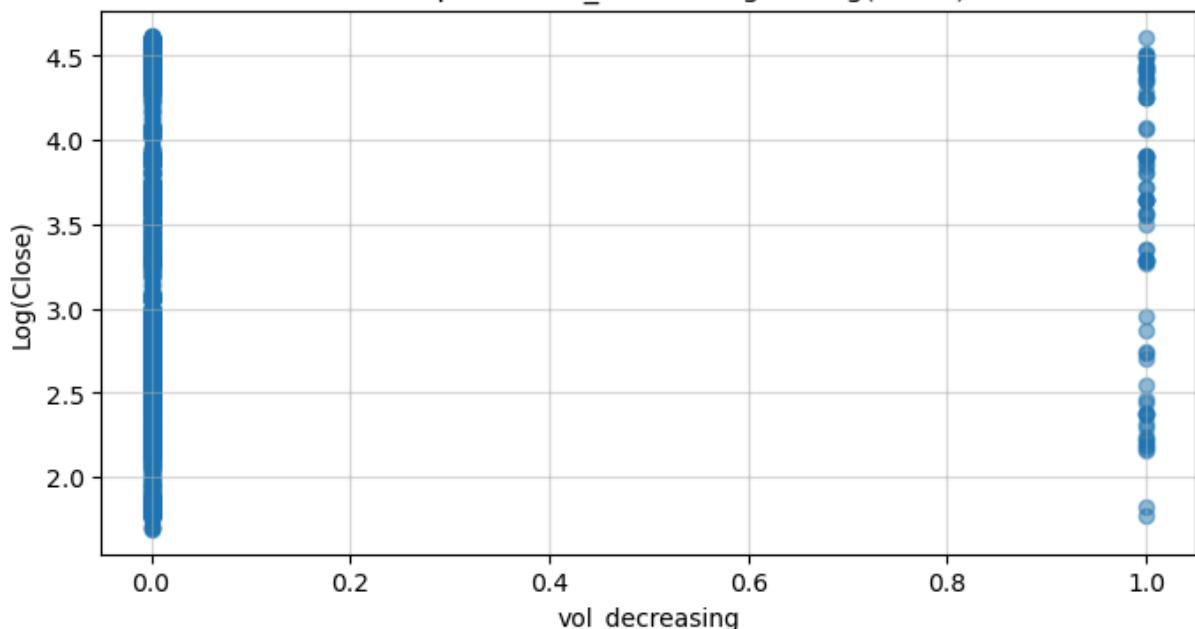
Scatterplot of macd\_signal vs Log(Close)



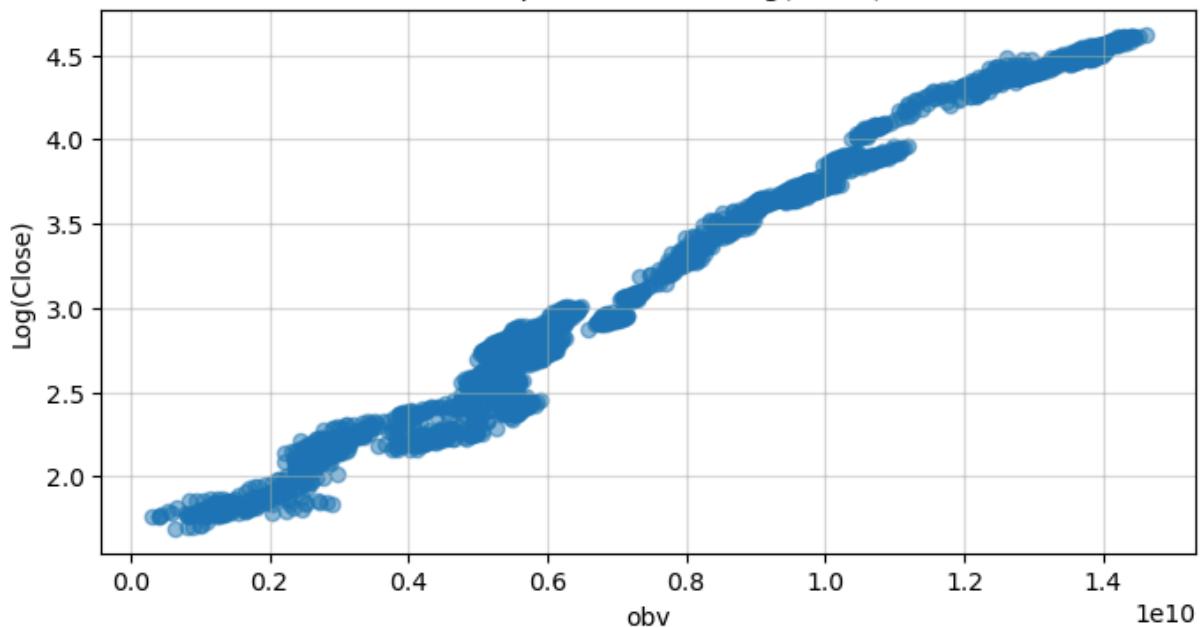
Scatterplot of vol\_increasing vs Log(Close)



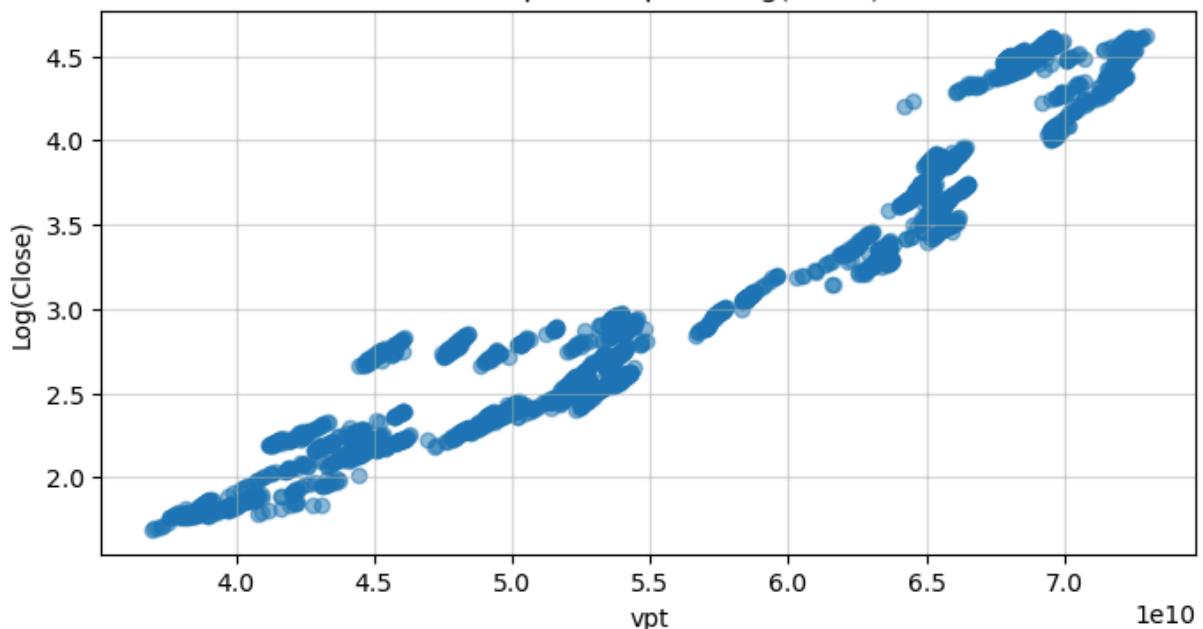
Scatterplot of vol\_decreasing vs Log(Close)



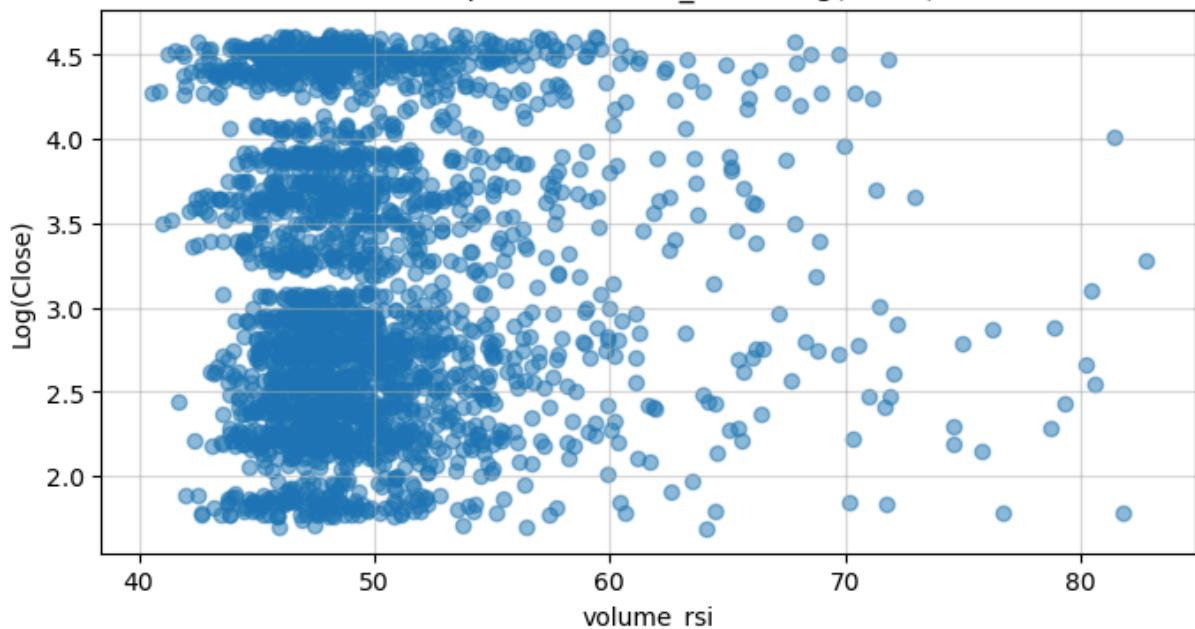
Scatterplot of obv vs Log(Close)



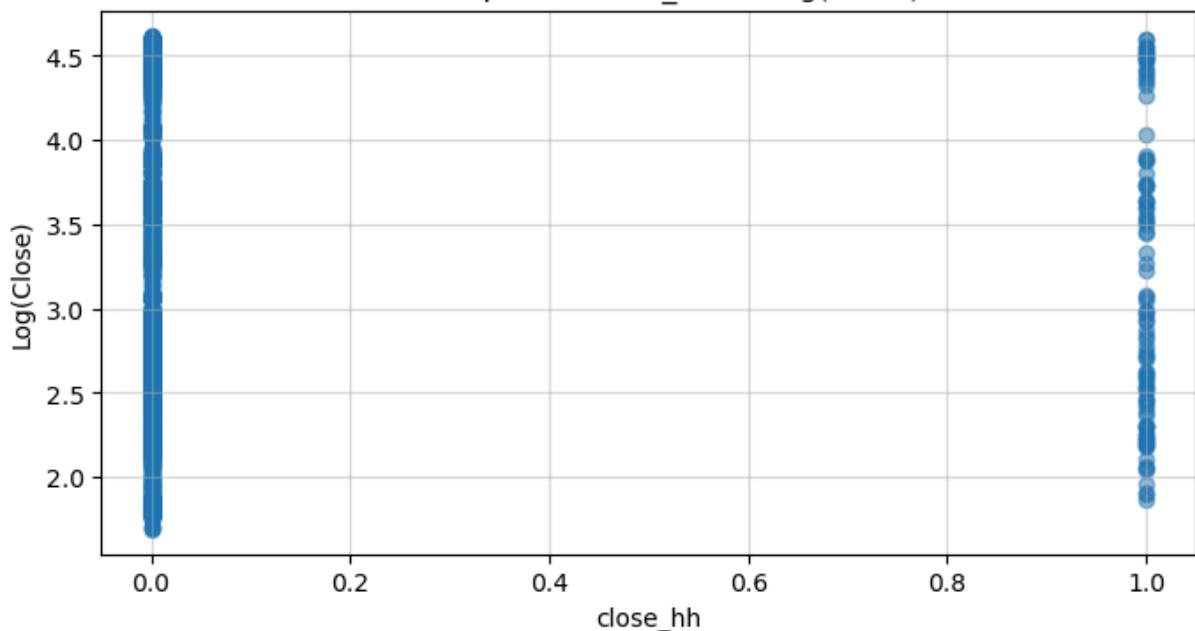
Scatterplot of vpt vs Log(Close)

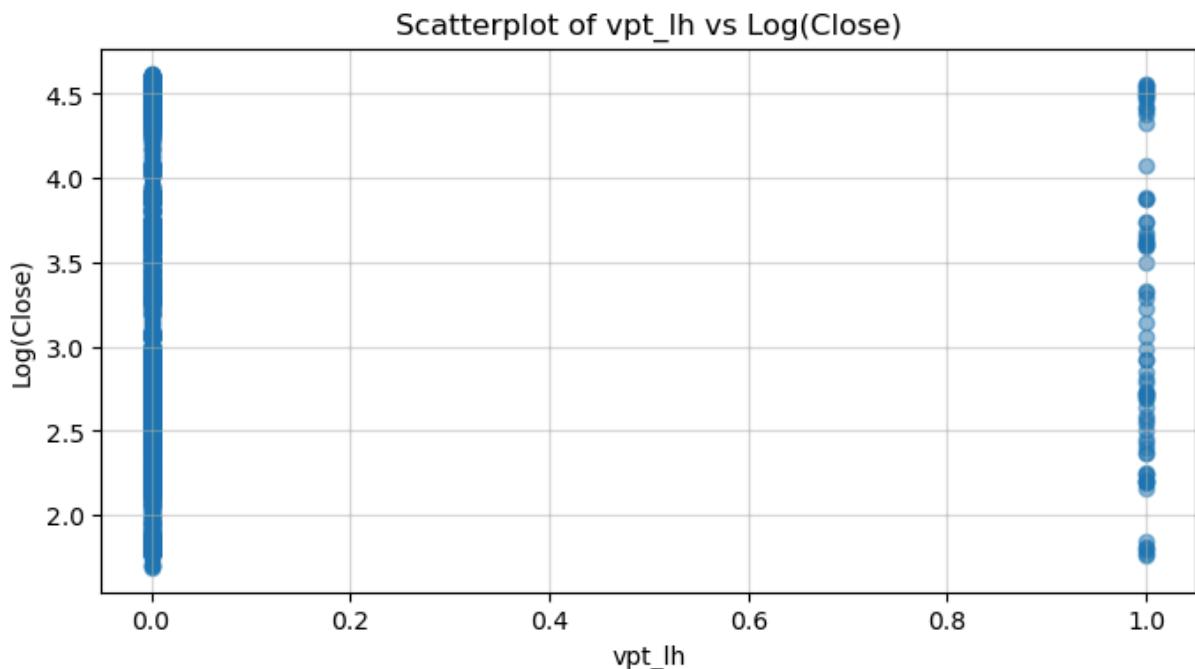
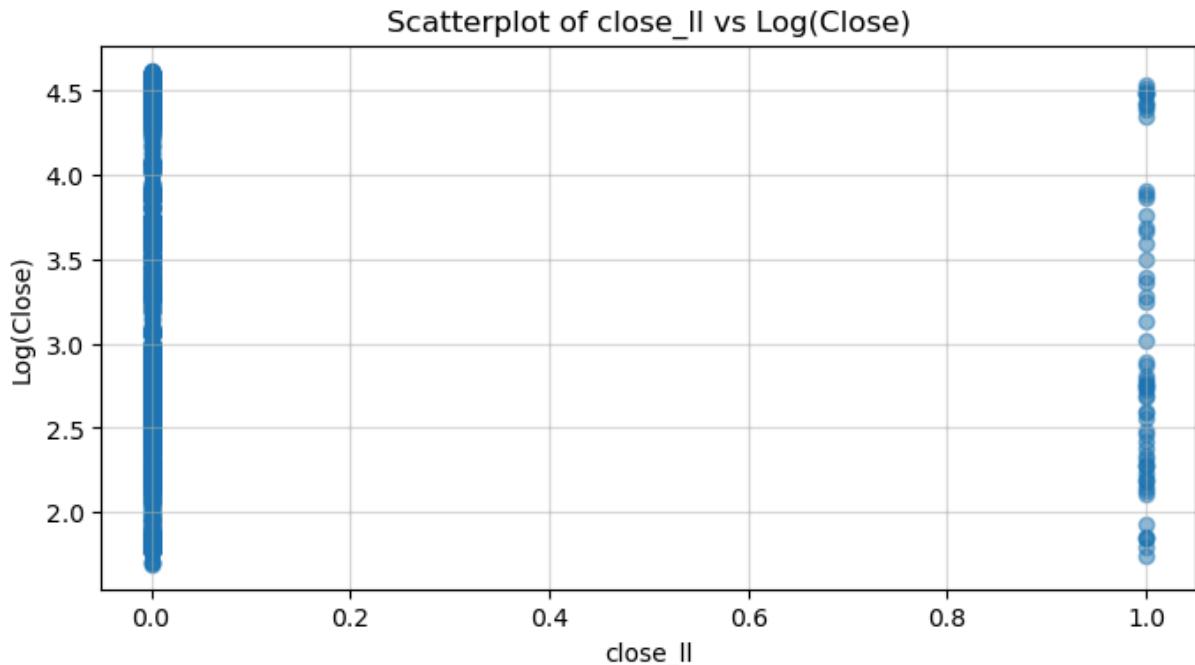


Scatterplot of volume\_rsi vs Log(Close)

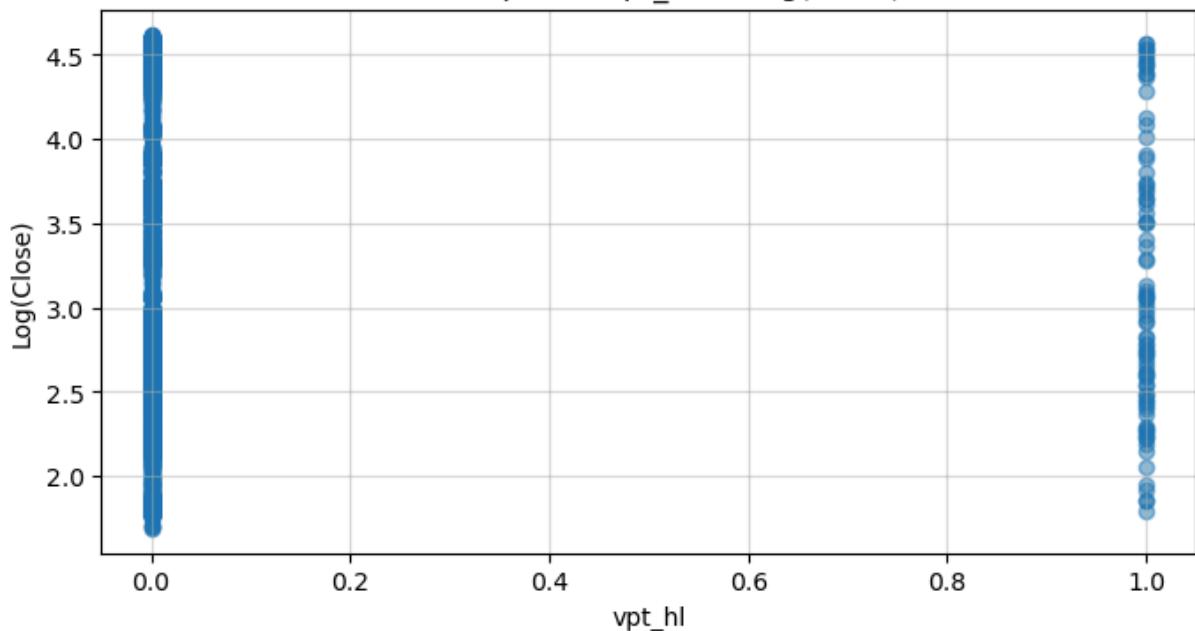


Scatterplot of close\_hh vs Log(Close)

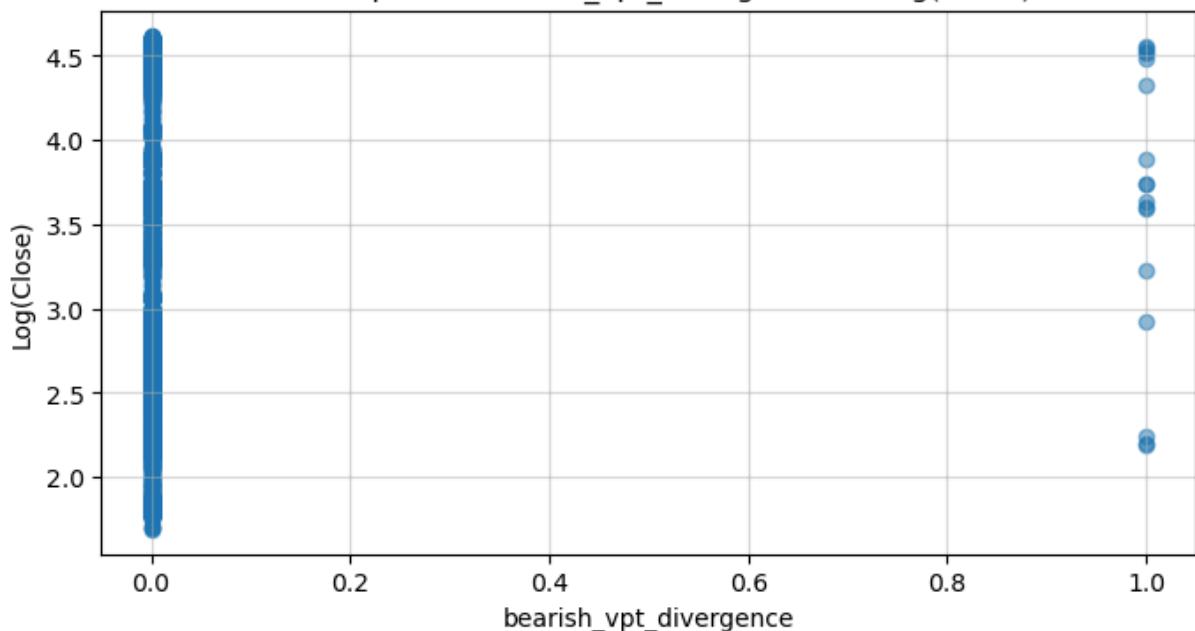




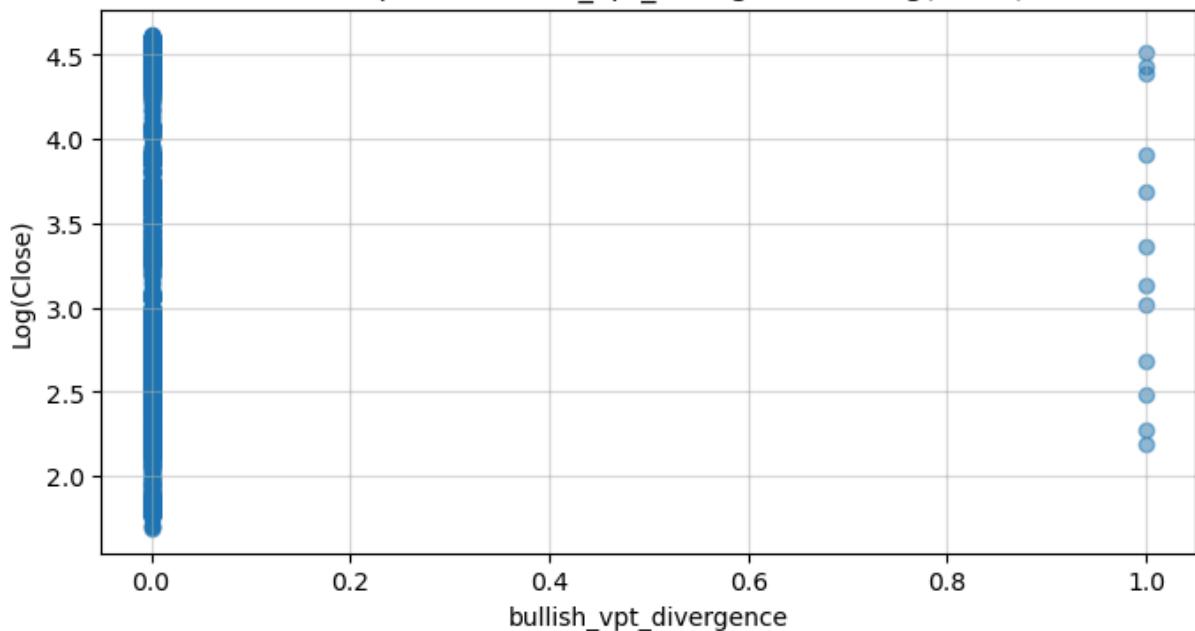
Scatterplot of vpt\_hl vs Log(Close)



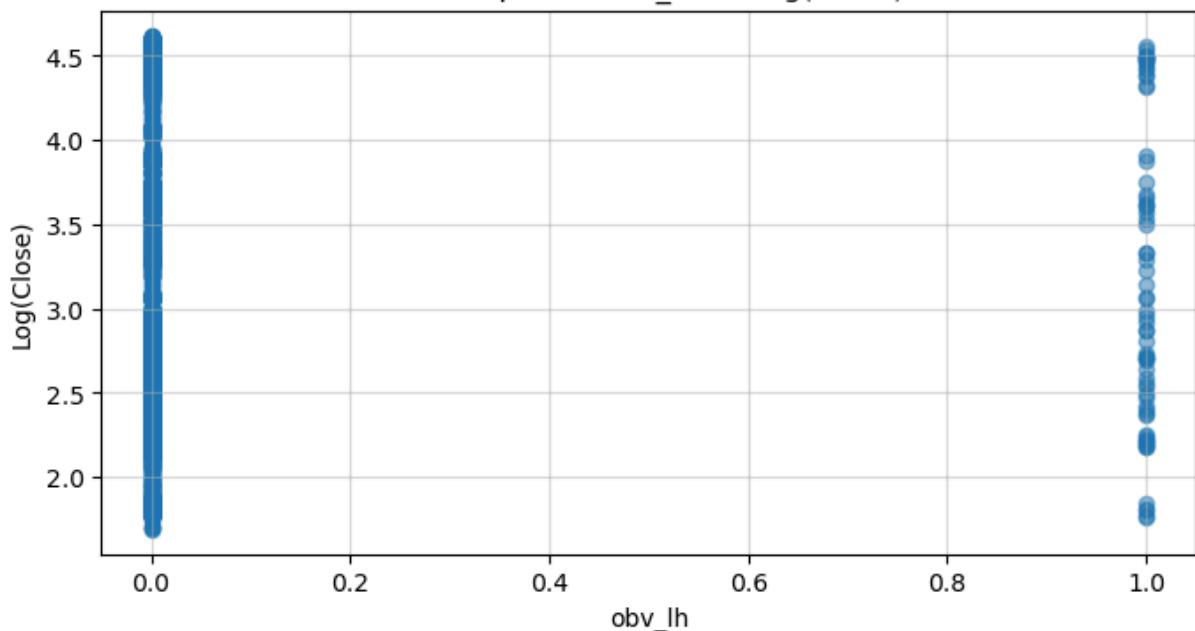
Scatterplot of bearish\_vpt\_divergence vs Log(Close)



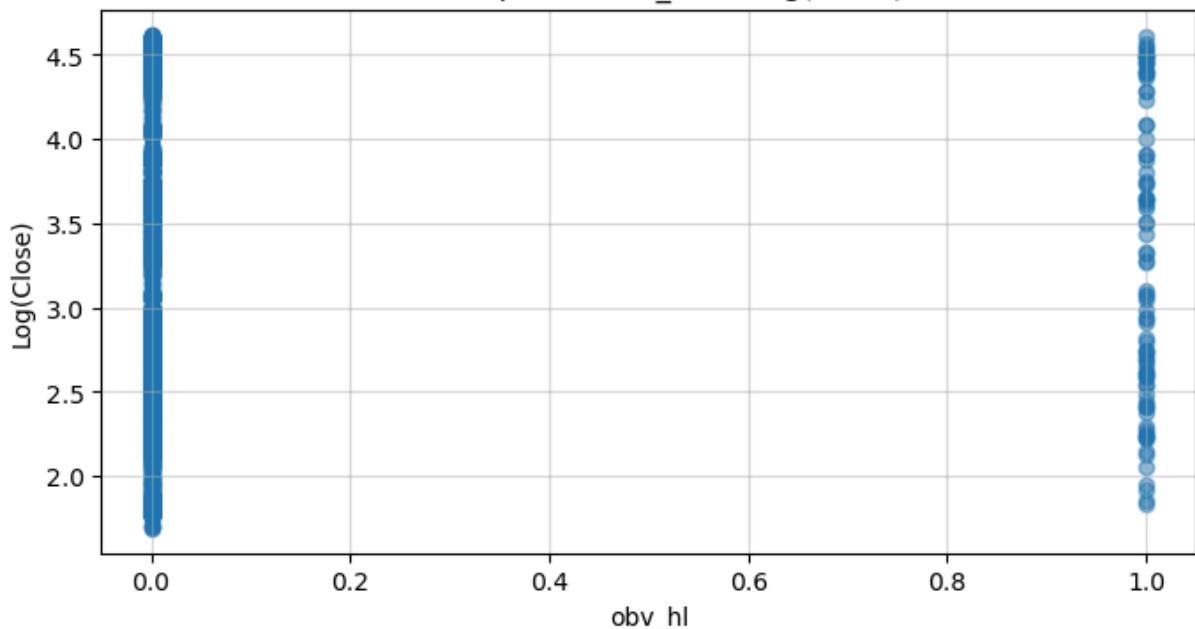
Scatterplot of bullish\_vpt\_divergence vs Log(Close)



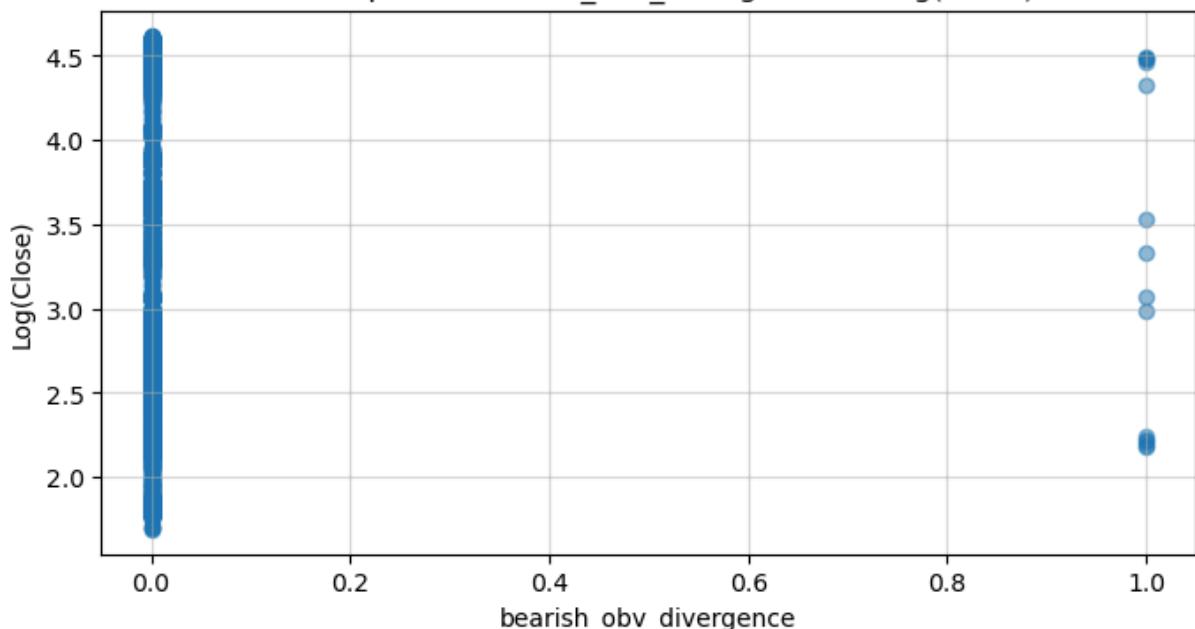
Scatterplot of obv\_lh vs Log(Close)



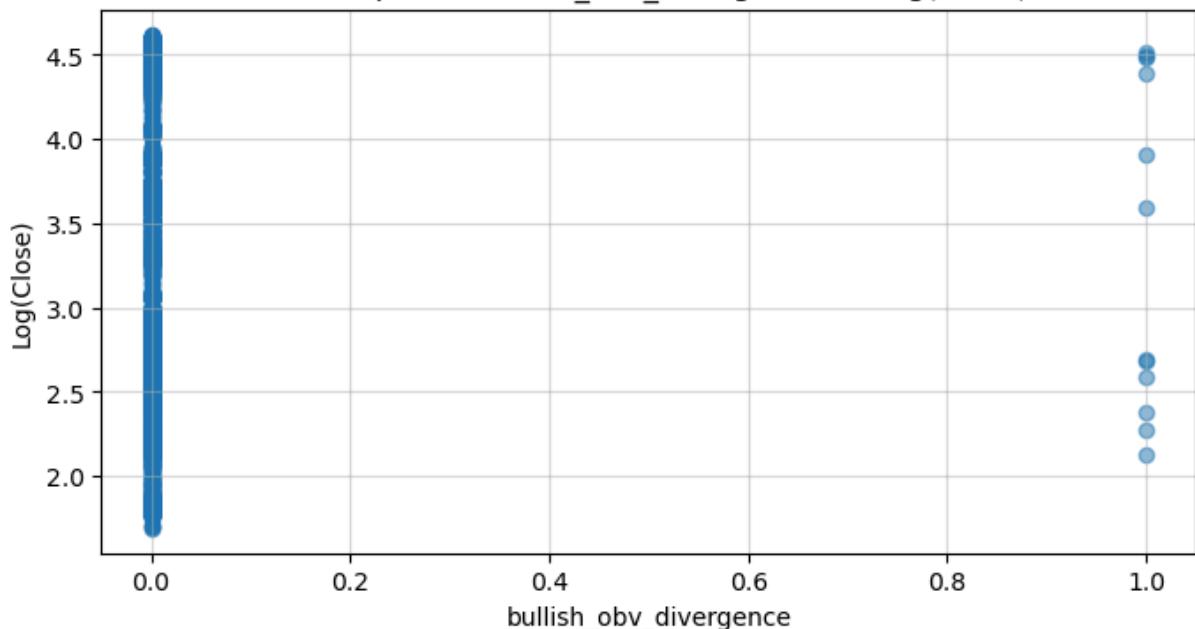
Scatterplot of obv\_hl vs Log(Close)



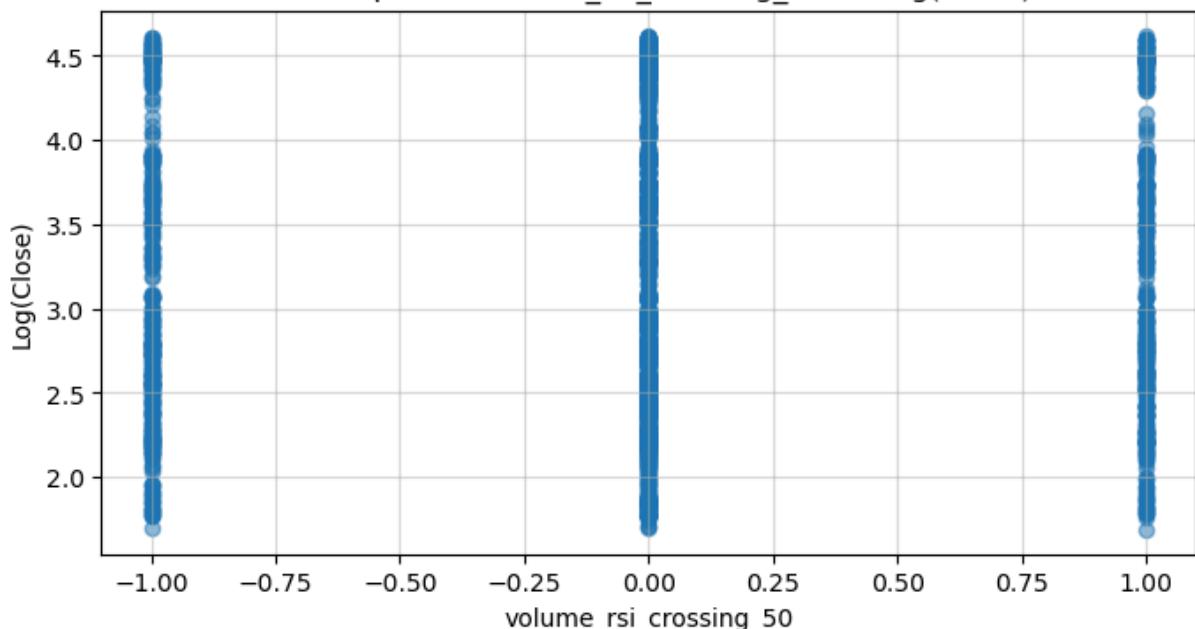
Scatterplot of bearish\_obv\_divergence vs Log(Close)



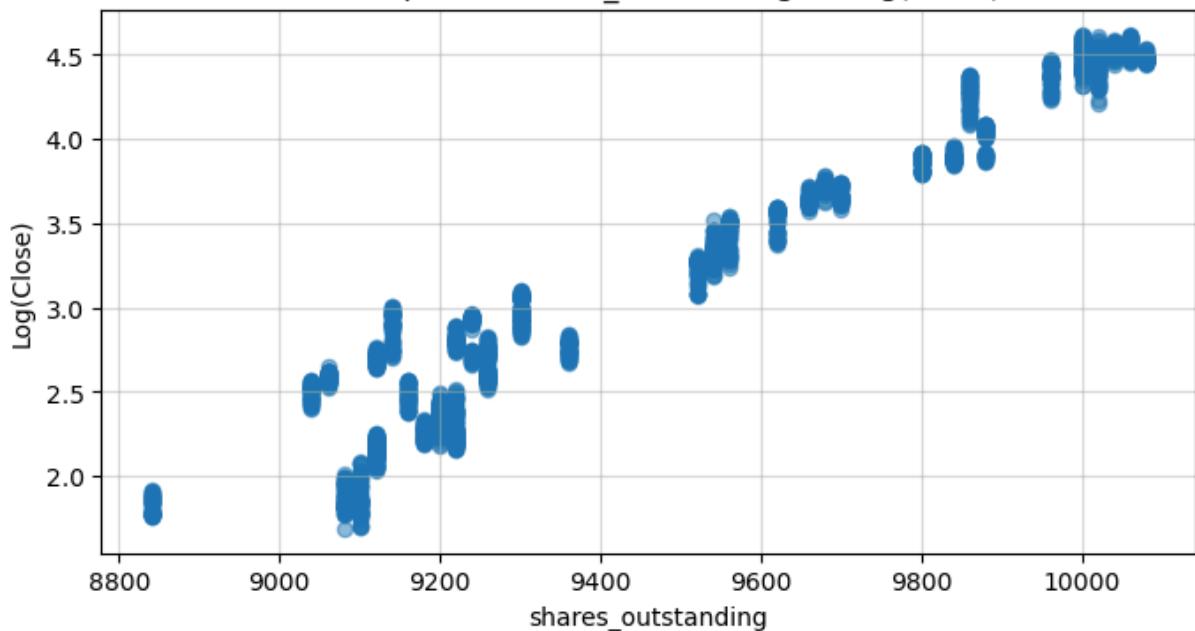
Scatterplot of bullish\_obv\_divergence vs Log(Close)



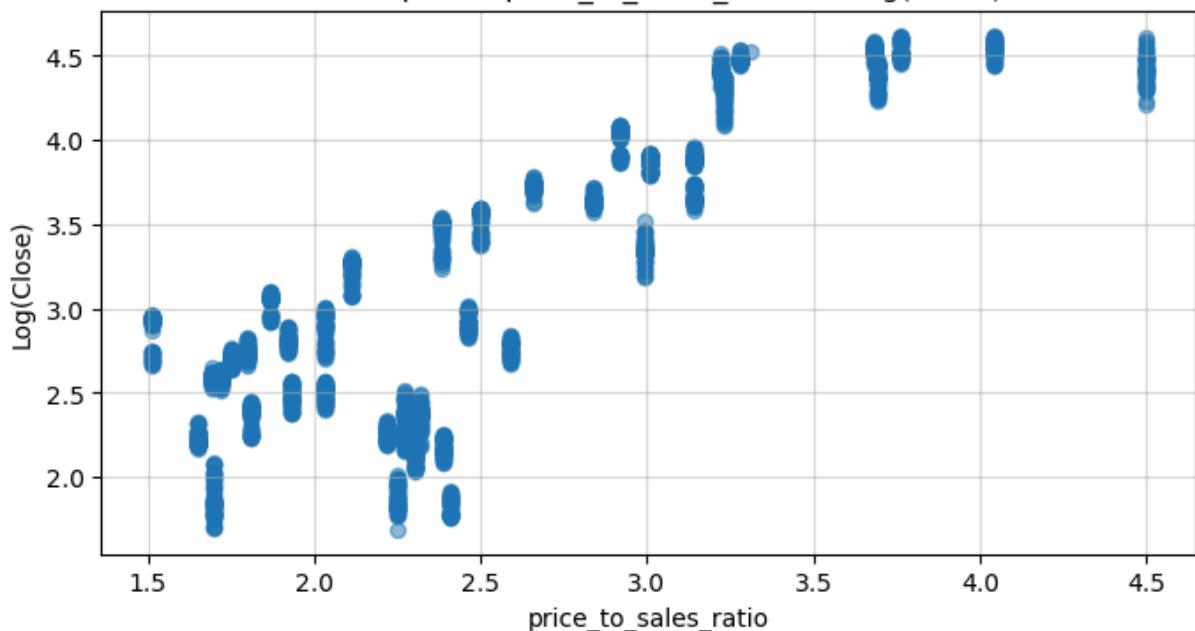
Scatterplot of volume\_rsi\_crossing\_50 vs Log(Close)



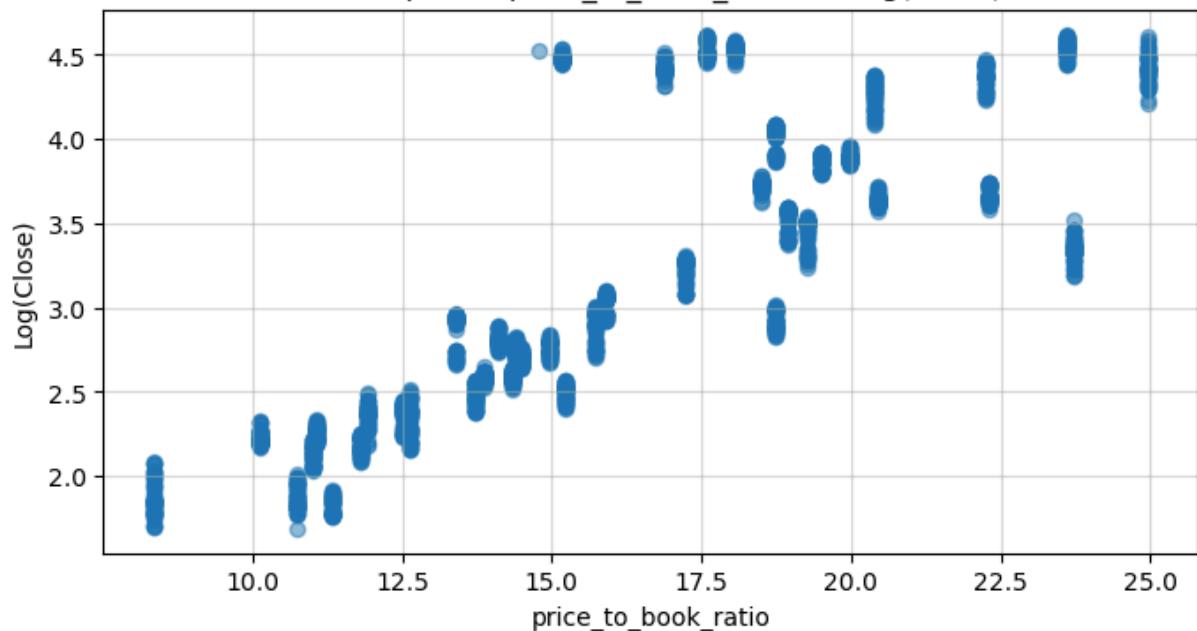
Scatterplot of shares\_outstanding vs Log(Close)



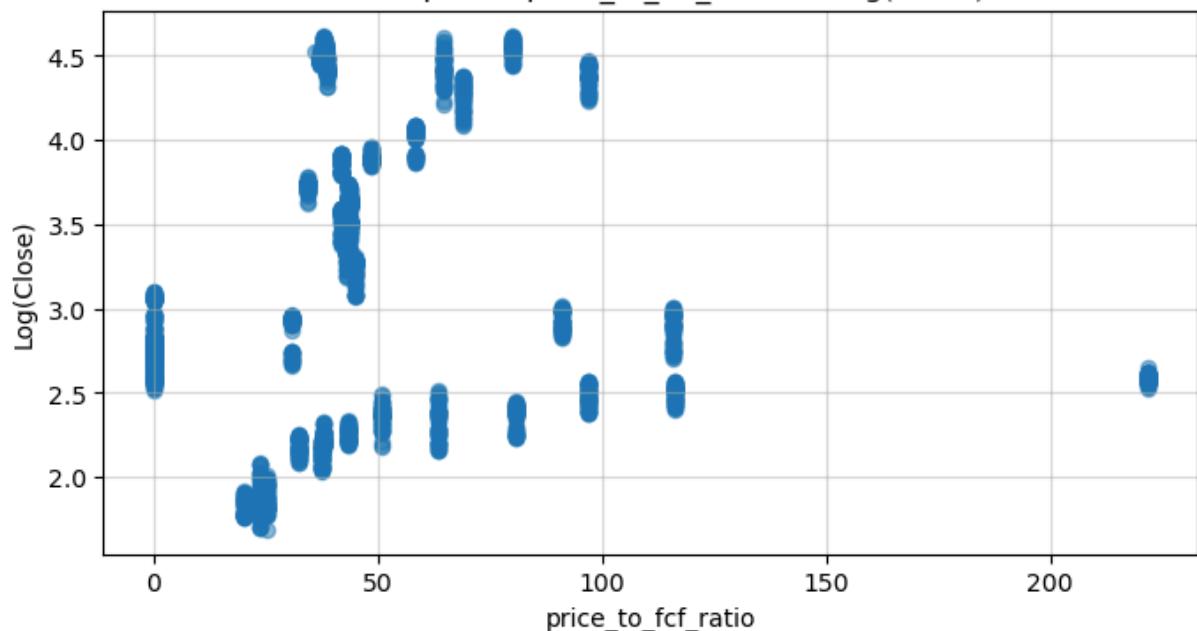
Scatterplot of price\_to\_sales\_ratio vs Log(Close)



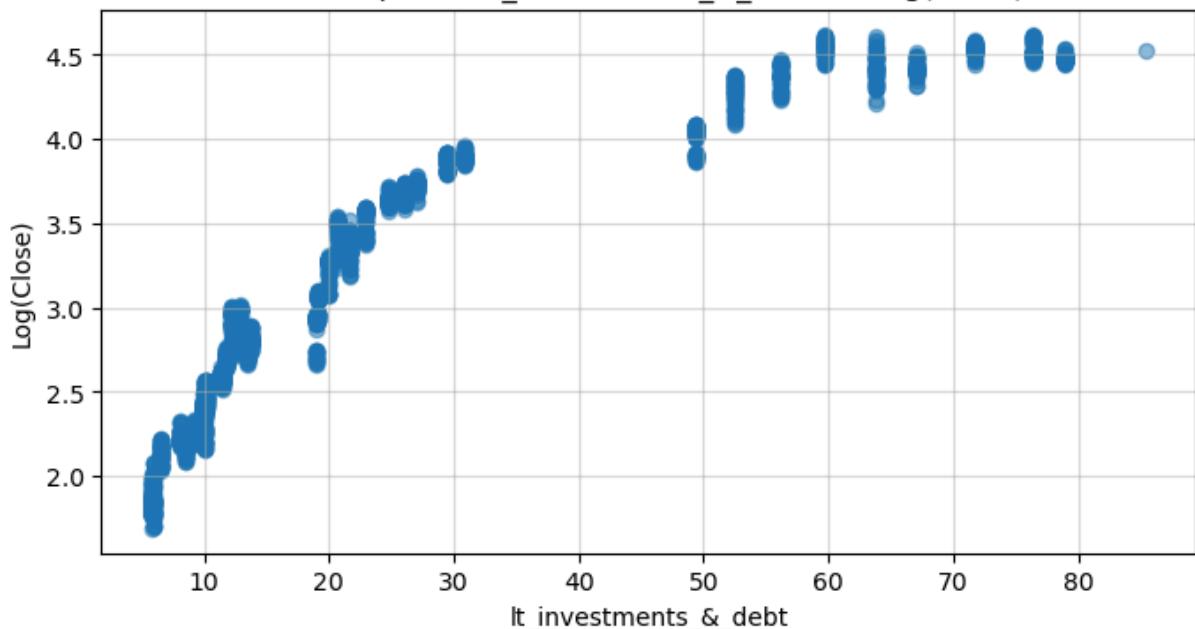
Scatterplot of price\_to\_book\_ratio vs Log(Close)



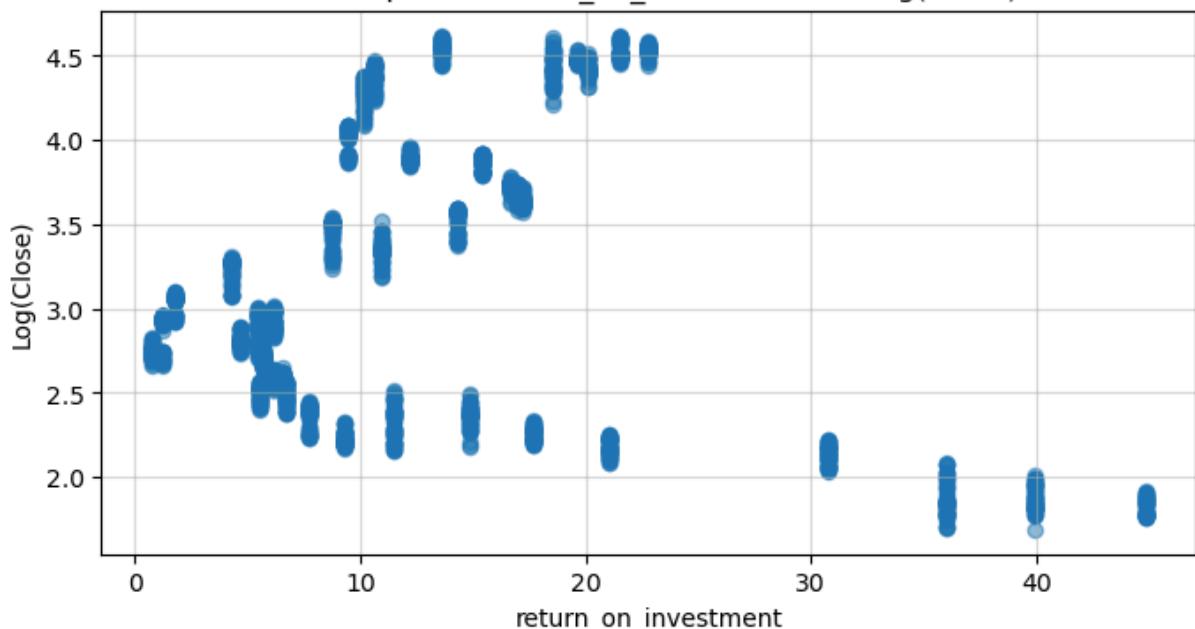
Scatterplot of price\_to\_fcf\_ratio vs Log(Close)



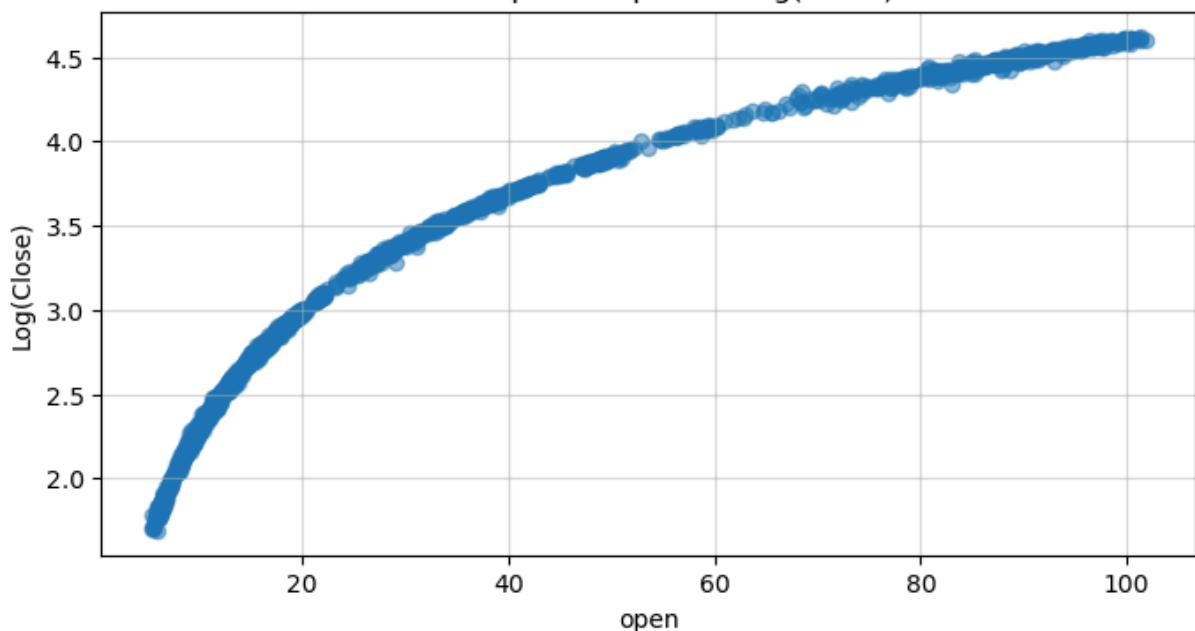
Scatterplot of It\_investments\_&amp;\_debt vs Log(Close)



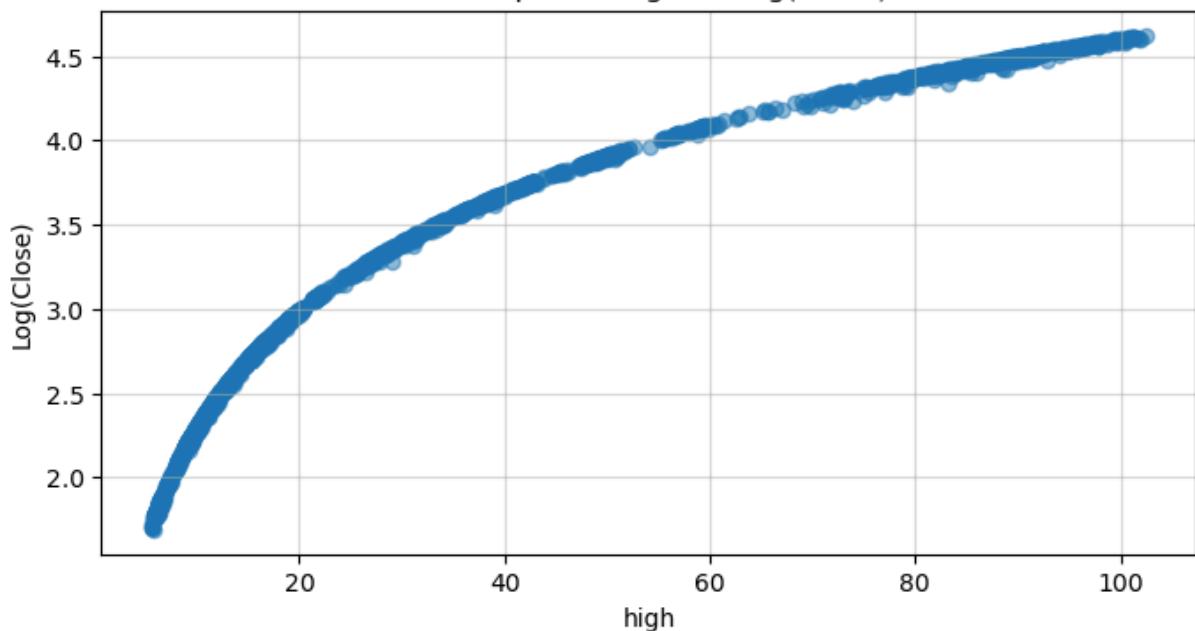
Scatterplot of return\_on\_investment vs Log(Close)



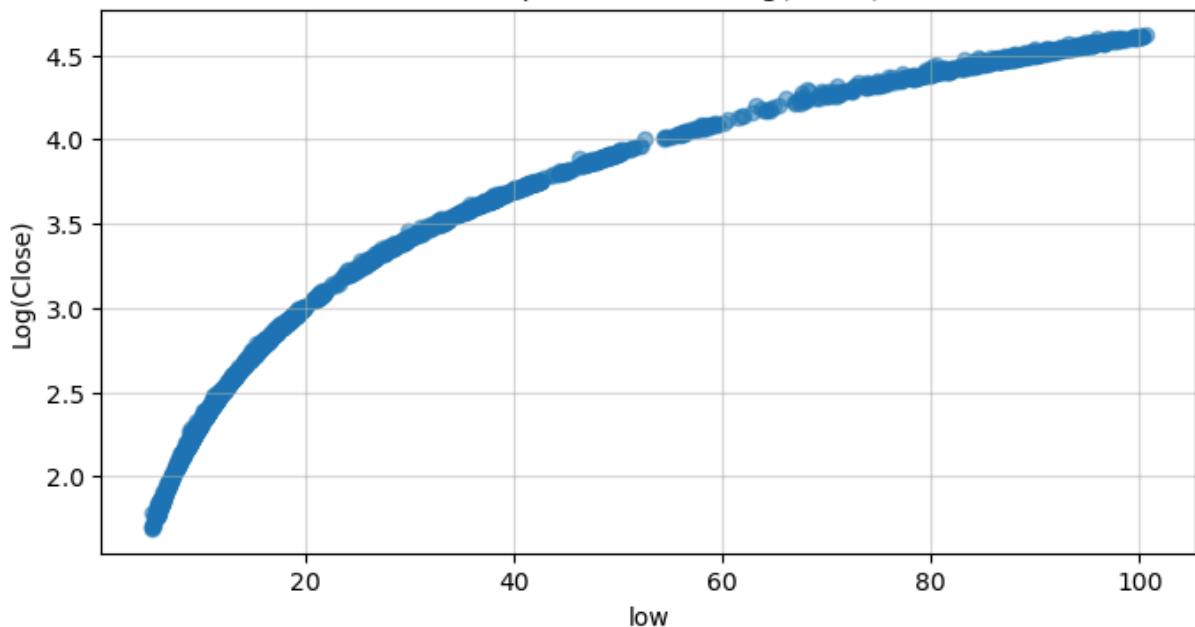
Scatterplot of open vs Log(Close)



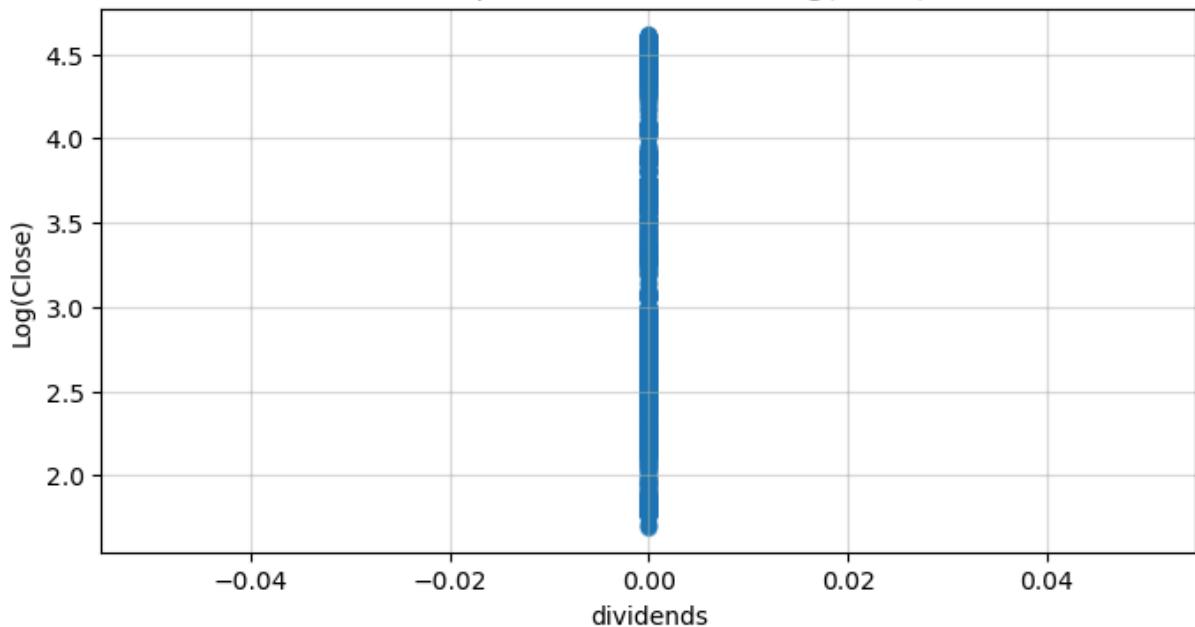
Scatterplot of high vs Log(Close)



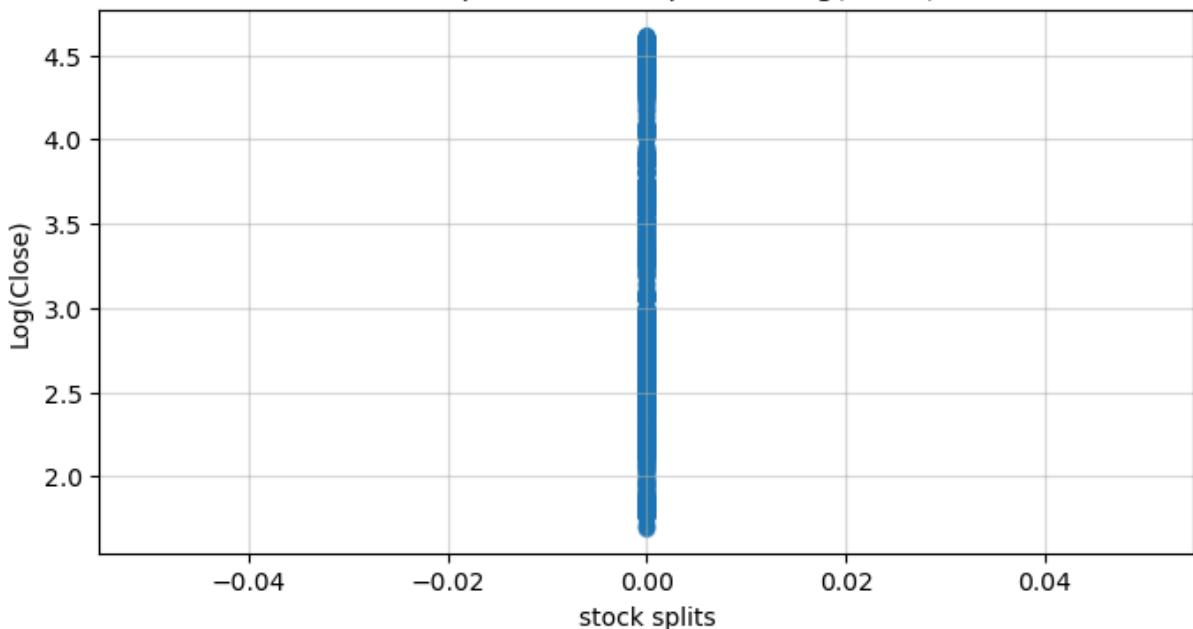
Scatterplot of low vs Log(Close)



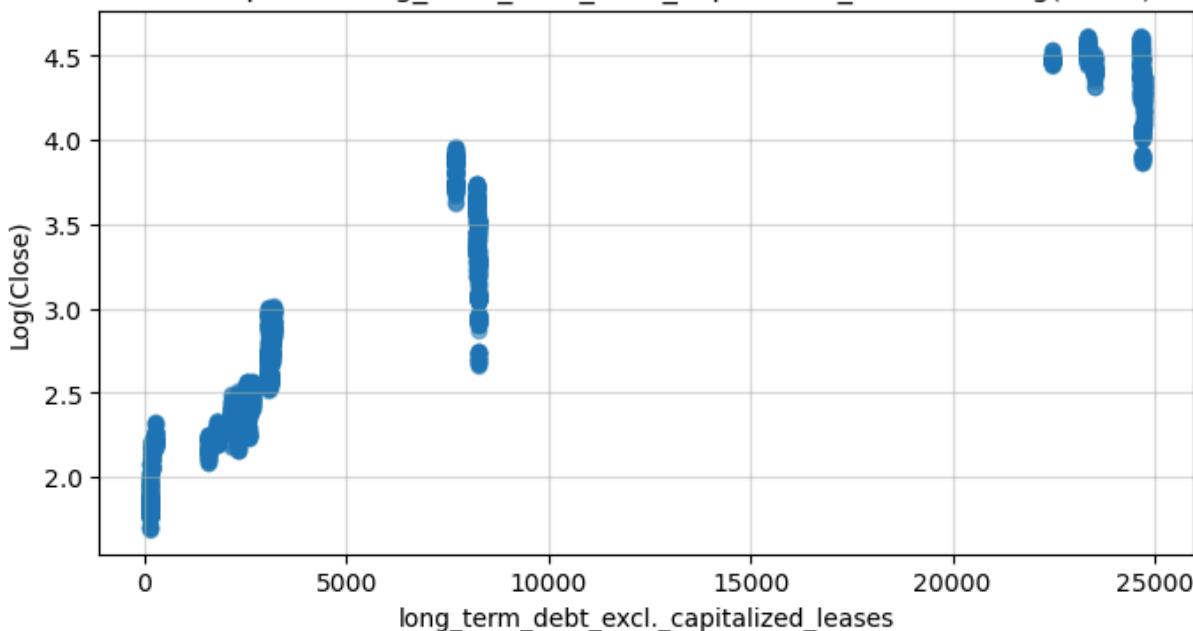
Scatterplot of dividends vs Log(Close)



Scatterplot of stock splits vs Log(Close)



Scatterplot of long\_term\_debt\_excl\_capitalized\_leases vs Log(Close)



## Correlation Analysis

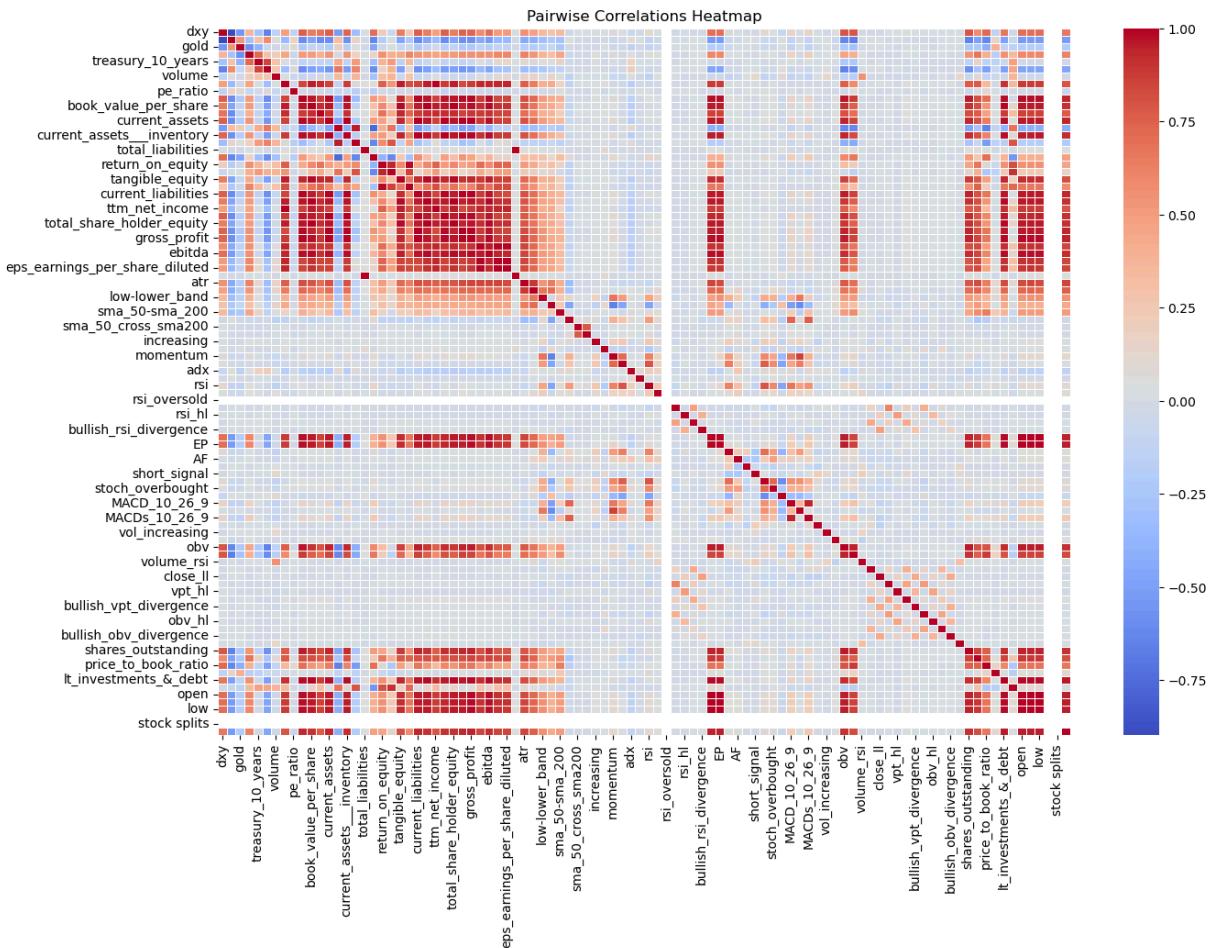
### Pairwise Correlations

```
In [13]: # Pairwise Correlations
import seaborn as sns
import matplotlib.pyplot as plt

# Update numeric data to include log-transformed response
numeric_data = train_data.select_dtypes(include=['int64', 'float64']).drop(['Date'], axis=1)
correlation_matrix = numeric_data.corr()

# Plot the heatmap
```

```
plt.figure(figsize=(15, 10))
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm', fmt='.2f', linewidths=1)
plt.title("Pairwise Correlations Heatmap")
plt.show()
```



## Correlation with Target Transformed Response

```
In [14]: # Calculate the correlation of all numeric predictors with the log-transform
correlation_with_target = numeric_data.corrwith(train_data['log_close']) #

# Set a correlation threshold (e.g., |correlation| >= 0.3)
high_corr_threshold = 0.3

# Filter predictors based on the threshold
selected_features_corr = correlation_with_target[abs(correlation_with_target)

# Display the selected features
print("Features Selected by Correlation with Log(Close):")
print(selected_features_corr)

# Subset the dataset to include only the selected predictors
X_corr_filtered = train_data[selected_features_corr]

# Number of columns in the filtered dataset
num_columns = X_corr_filtered.shape[1]
print(f"Number of predictors selected: {num_columns}")
```

```
Features Selected by Correlation with Log(Close):
Index(['dxy', 'oil', 'treasury_5_years', 'treasury_30_years', 'ttm_net_eps',
       'ttm_sales_per_share', 'book_value_per_share', 'ttm_fcf_per_share',
       'current_assets', 'current_ratio', 'current_assets__inventory',
       'quick_ratio', 'debt_to_equity_ratio', 'return_on_equity',
       'tangible_equity', 'return_on_tangible_equity', 'current_liabilitie
s',
       'shareholder's_equity', 'ttm_net_income', 'cash_on_hand',
       'total_share_holder_equity', 'revenue', 'gross_profit',
       'operating_income', 'ebitda', 'net_income',
       'eps_earnings_per_share_diluted', 'atr', 'stdev', 'low-lower_band',
       'upper_band-high', 'sma_50-sma_200', 'PSAR', 'EP', 'obv', 'vpt',
       'shares_outstanding', 'price_to_sales_ratio', 'price_to_book_ratio',
       'lt_investments_&_debt', 'open', 'high', 'low',
       'long_term_debt_excl._capitalized_leases'],
      dtype='object')
Number of predictors selected: 44
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-package
s/numpy/lib/function_base.py:2897: RuntimeWarning: invalid value encountered
in divide
    c /= stddev[:, None]
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-package
s/numpy/lib/function_base.py:2898: RuntimeWarning: invalid value encountered
in divide
    c /= stddev[None, :]
```

```
In [15]: from sklearn.linear_model import LinearRegression
from sklearn.model_selection import TimeSeriesSplit
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import numpy as np

X = numeric_data
y = train_data['log_close']

# Standardize predictors
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Time series split
tscv = TimeSeriesSplit(n_splits=5)

# Initialize list to store RMSE for each split
rmse_list = []

# Perform time series split and train/test evaluation for full linear regression
for train_index, test_index in tscv.split(X_scaled):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train a linear regression model
    lr_model = LinearRegression()
    lr_model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = lr_model.predict(X_test)
```

```
# Calculate RMSE
rmse = mean_squared_error(y_test, y_pred, squared=False)
rmse_list.append(rmse)

# Average RMSE across all splits
average_rmse_log_scale = sum(rmse_list) / len(rmse_list)

# Convert RMSE from log scale to original scale
average_rmse_original_scale = np.exp(average_rmse_log_scale)
```

```
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
    /var/folders/2p/hbz8h54x5hj828cdgg7jshz0000gn/T/ipykernel_66719/1742660946.py:40: RuntimeWarning: overflow encountered in exp
        average_rmse_original_scale = np.exp(average_rmse_log_scale)
```

In [16]: # Print results

```
print(f"Linear Regression Average RMSE (Log Scale): {average_rmse_log_scale}")
print(f"Linear Regression Average RMSE (Original Scale): {average_rmse_original_scale}")
```

```
Linear Regression Average RMSE (Log Scale): 66247924.34874252
Linear Regression Average RMSE (Original Scale): inf
```

## PCA Analysis with Transformed Target

In [17]:

```
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import TimeSeriesSplit
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import numpy as np
import pandas as pd
```

```

import matplotlib.pyplot as plt

# Prepare data
X = numeric_data # Predictors
y = train_data['log_close'] # Log-transformed response variable

# Standardize predictors
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Perform PCA
pca = PCA()
X_pca = pca.fit_transform(X_scaled)

# Determine explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_[:5] # First 5 components
cumulative_explained_variance = np.cumsum(pca.explained_variance_ratio_[:5])

# Display explained variance for the first 5 components
explained_variance_table = pd.DataFrame({
    "Principal Component": [f"PC{i+1}" for i in range(5)],
    "Explained Variance (%)": explained_variance_ratio * 100,
    "Cumulative Explained Variance (%)": cumulative_explained_variance * 100
})
print("Explained Variance Table (First 5 Components):")
print(explained_variance_table)

```

Explained Variance Table (First 5 Components):

	Principal Component	Explained Variance (%)
0	PC1	34.103314
1	PC2	8.371138
2	PC3	7.031563
3	PC4	3.282007
4	PC5	3.207038

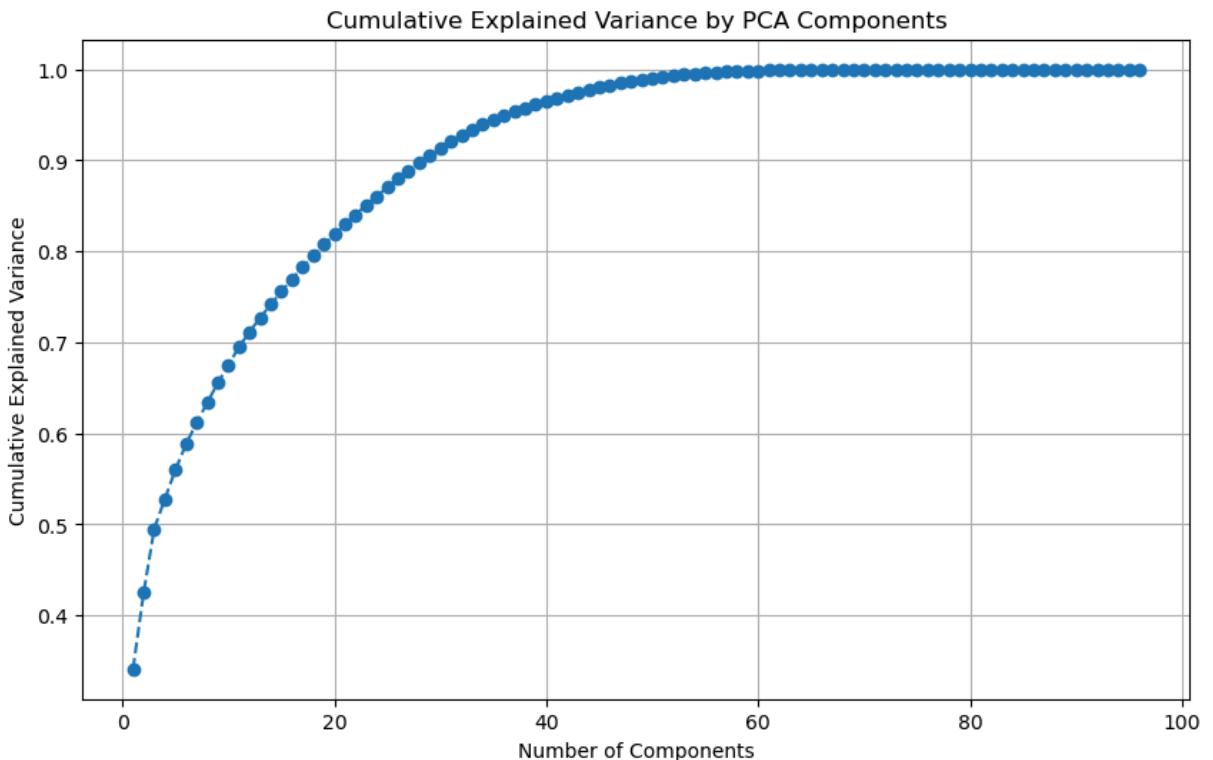
	Cumulative Explained Variance (%)
0	34.103314
1	42.474452
2	49.506015
3	52.788022
4	55.995061

In [18]:

```

# Visualize cumulative explained variance (all components)
cumulative_explained_variance_all = np.cumsum(pca.explained_variance_ratio_)
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(cumulative_explained_variance_all) + 1), cumulative_explained_variance_all)
plt.title("Cumulative Explained Variance by PCA Components")
plt.xlabel("Number of Components")
plt.ylabel("Cumulative Explained Variance")
plt.grid()
plt.show()

```



```
In [19]: # Choose the number of components (e.g., explaining 95% variance)
n_components = np.argmax(cumulative_explained_variance_all >= 0.95) + 1
X_pca_reduced = X_pca[:, :n_components]

# Time series split
tscv = TimeSeriesSplit(n_splits=5)

# Initialize list to store MSPE for each split
rmse_list = []

# Perform time series split and train/test evaluation
for train_index, test_index in tscv.split(X_pca_reduced):
    X_train, X_test = X_pca_reduced[train_index], X_pca_reduced[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train a linear regression model using PCA components
    lr_model = LinearRegression()
    lr_model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = lr_model.predict(X_test)

    # Calculate MSPE
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    rmse_list.append(rmse)
```

```

/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(

```

In [20]: # Average MSPE across all splits  
`average_rmse = sum(rmse_list) / len(rmse_list)`  
`print(f"PCA Regression Average RMSE (Log Scale): {average_rmse}")`  
`print(f"PCA Regression Average RMSE (Original Scale): {np.exp(average_rmse)})`

PCA Regression Average RMSE (Log Scale): 0.3535938673414937  
PCA Regression Average RMSE (Original Scale): 1.424176664354599

In [21]: # import numpy as np  
# import pandas as pd  
  
# Get loadings (coefficients of original variables in principal components)  
loadings = pd.DataFrame(  
 pca.components\_.T,  
 columns=[f"PC{i+1}" for i in range(len(pca.components\_))],  
 index=numeric\_data.columns  
)  
  
# Convert loadings into a DataFrame for readability, focusing on the first 5  
loadings\_df = loadings.iloc[:, :5].reset\_index()  
loadings\_df.rename(columns={'index': 'Variable'}, inplace=True)  
  
# Display the DataFrame  
print("Principal Component Loadings (Top Components):")  
print(loadings\_df)

## Principal Component Loadings (Top Components):

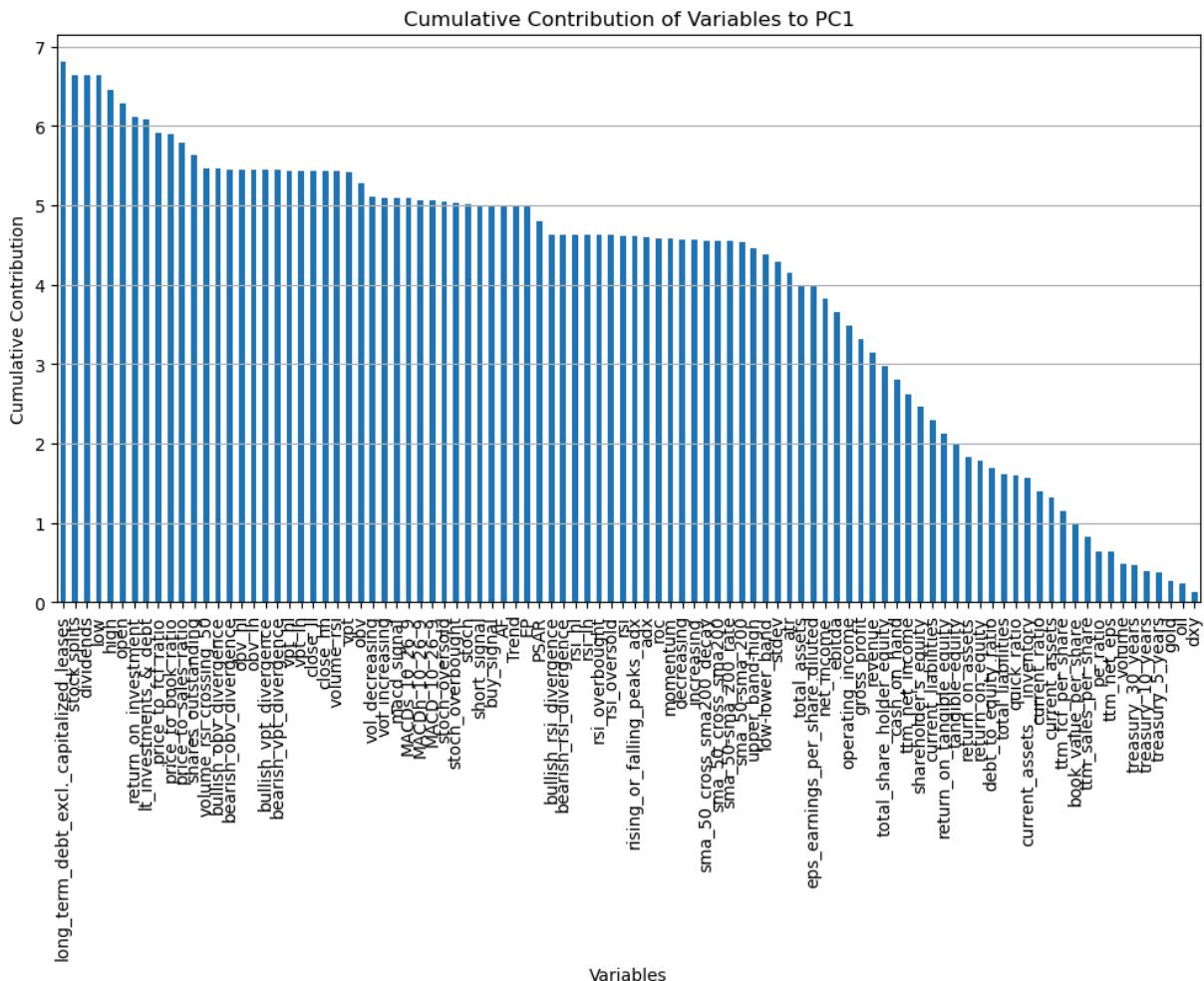
	Variable	PC1	PC2	PC3	\
0	dxy	0.130880	-0.102277	-0.081700	
1	oil	-0.103436	0.080967	0.059561	
2	gold	-0.040273	0.044036	-0.008220	
3	treasury_5_years	0.103547	0.079024	0.113883	
4	treasury_10_years	-0.008793	0.157917	0.184799	
..		...	...	...	...
91	high	0.175624	-0.011934	0.009110	
92	low	0.175518	-0.013856	0.010494	
93	dividends	0.000000	-0.000000	0.000000	
94	stock splits	0.000000	-0.000000	0.000000	
95	long_term_debt_excl._capitalized_leases	0.167644	-0.029869	-0.003339	
	PC4	PC5			
0	-0.072110	-0.041572			
1	0.092678	0.054716			
2	0.322501	0.194545			
3	-0.281680	-0.165491			
4	-0.275893	-0.163471			
..	..	..			
91	-0.017648	0.000300			
92	-0.015160	0.002625			
93	0.000000	-0.000000			
94	0.000000	-0.000000			
95	-0.043982	-0.021789			

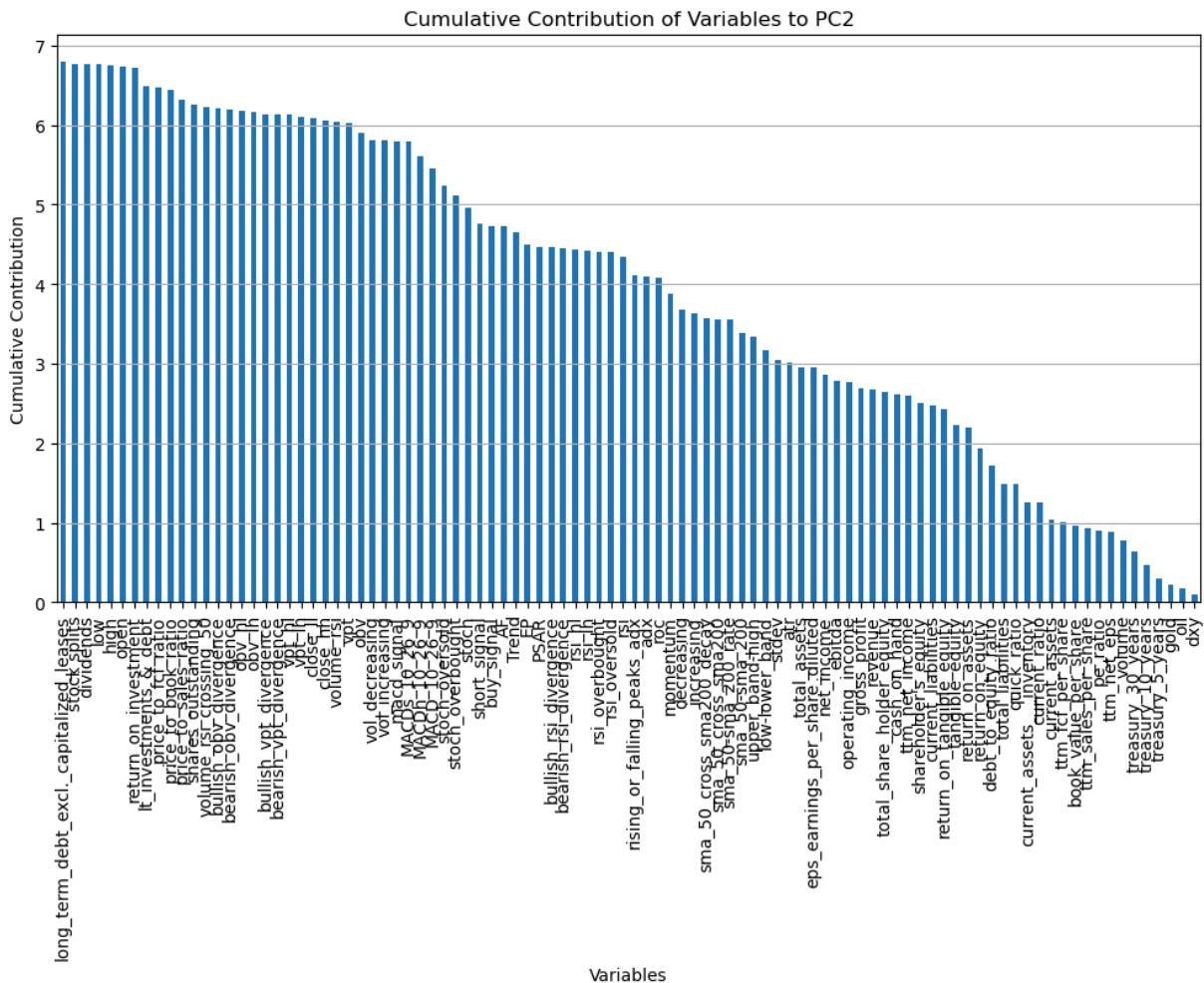
[96 rows x 6 columns]

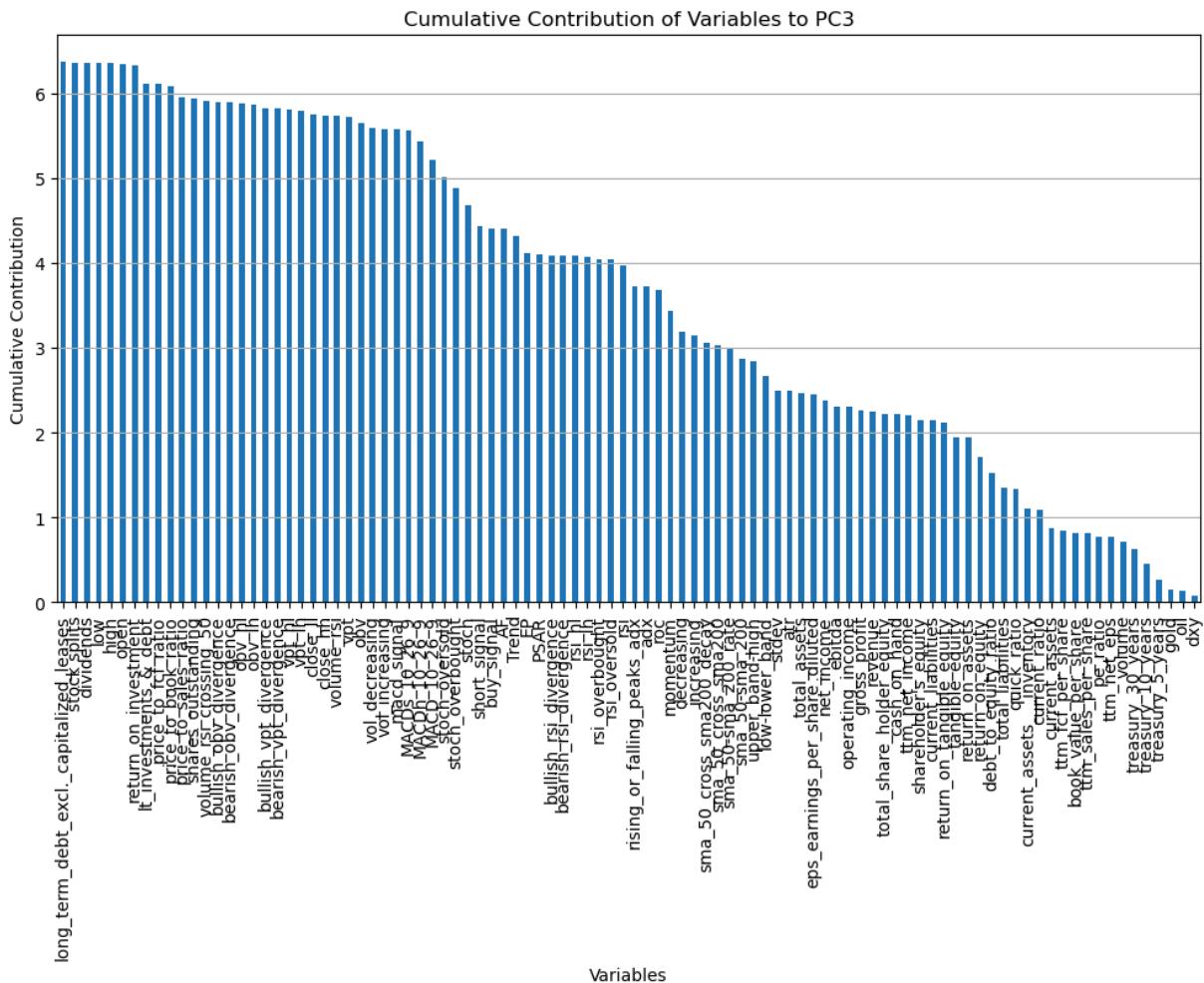
```
In [22]: # Loop through all principal components and plot cumulative contributions
for i in range(loadings.shape[1]): # Number of components
    component_name = f"PC{i+1}"

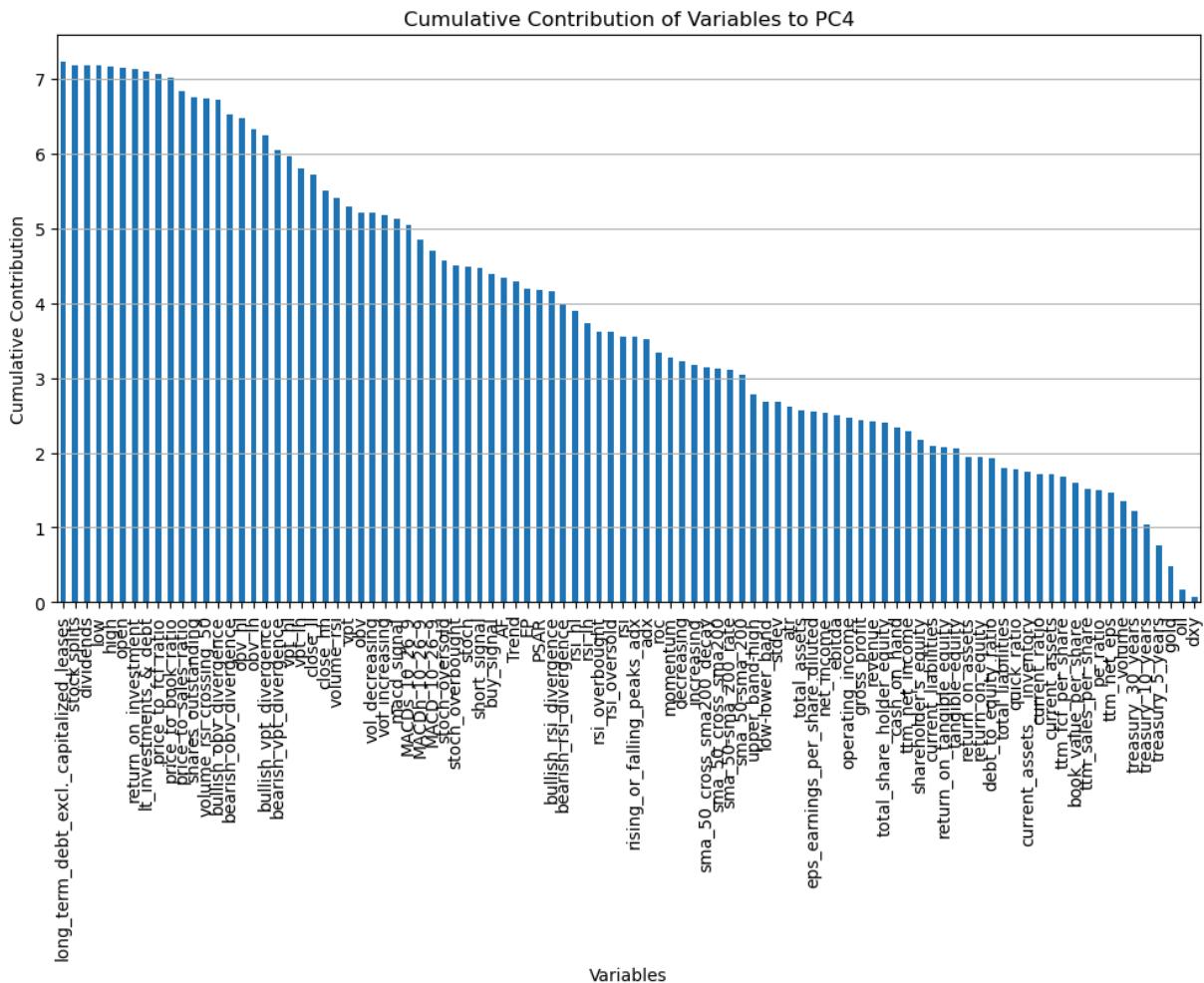
    plt.figure(figsize=(12, 6))
    cumulative_contribution = loadings.abs().cumsum(axis=0) # Cumulative sum
    cumulative_contribution[component_name].sort_values(ascending=False).plot()

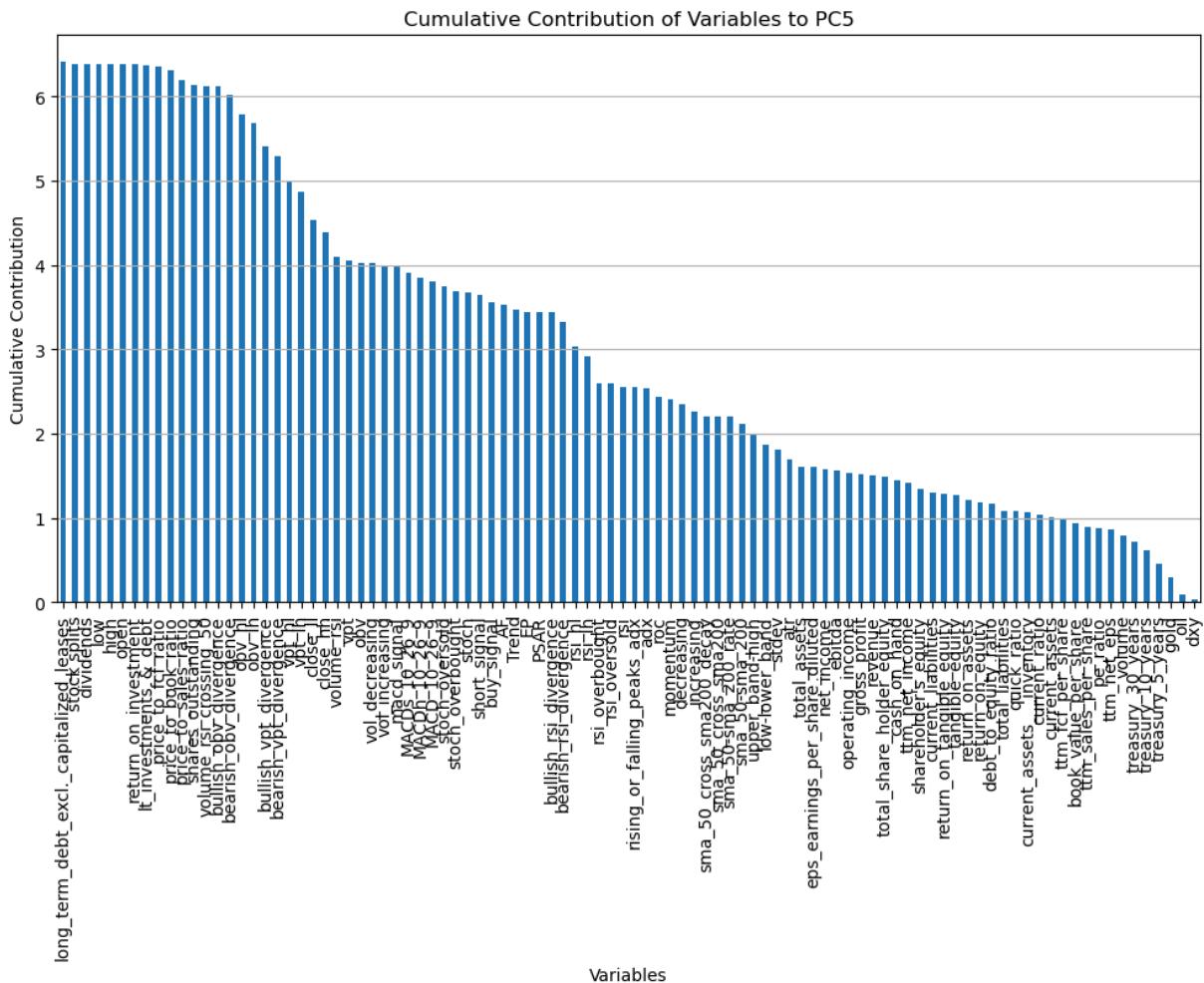
    plt.title(f'Cumulative Contribution of Variables to {component_name}')
    plt.xlabel('Variables')
    plt.ylabel('Cumulative Contribution')
    plt.grid(axis='y')
    plt.show()
```

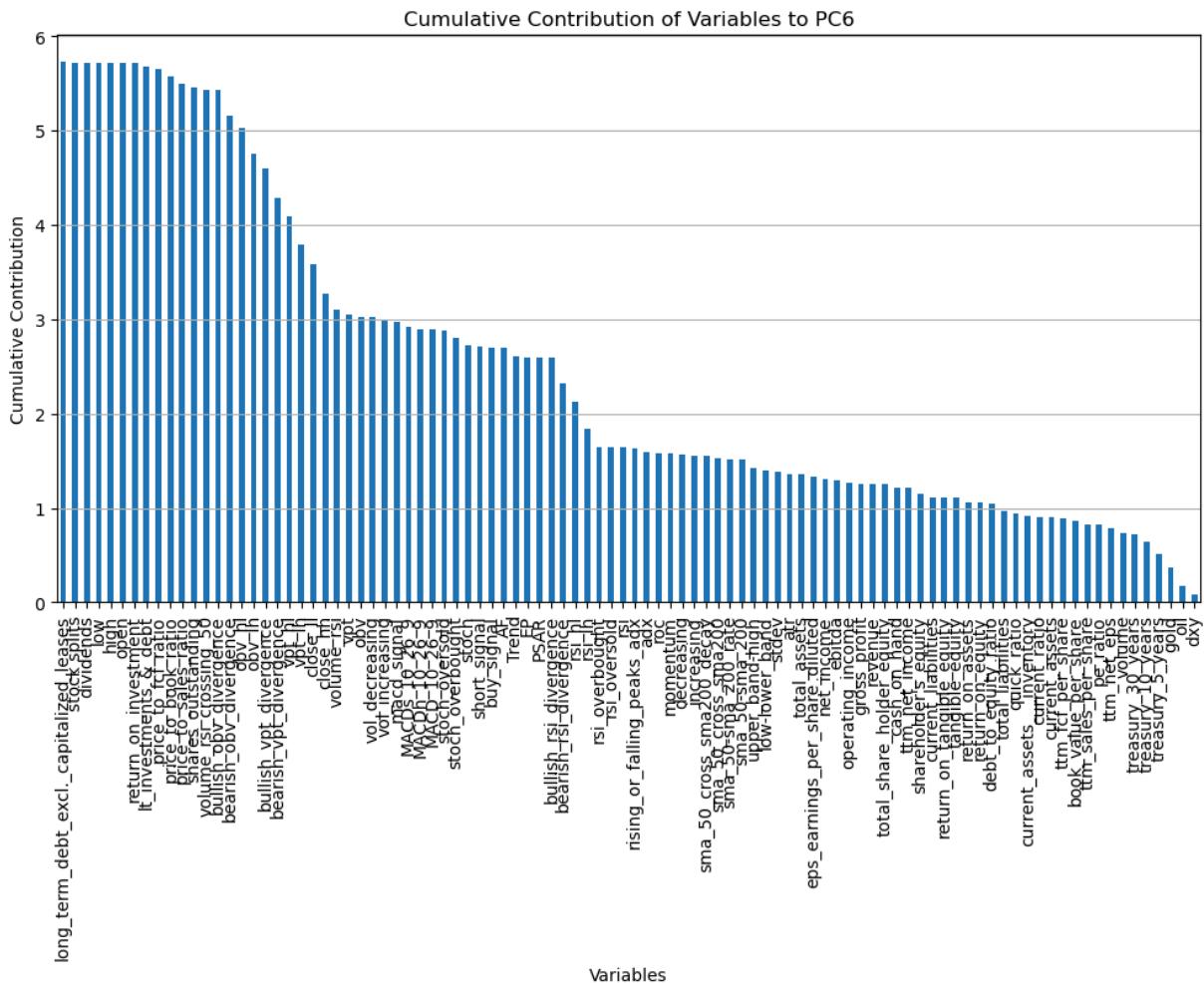




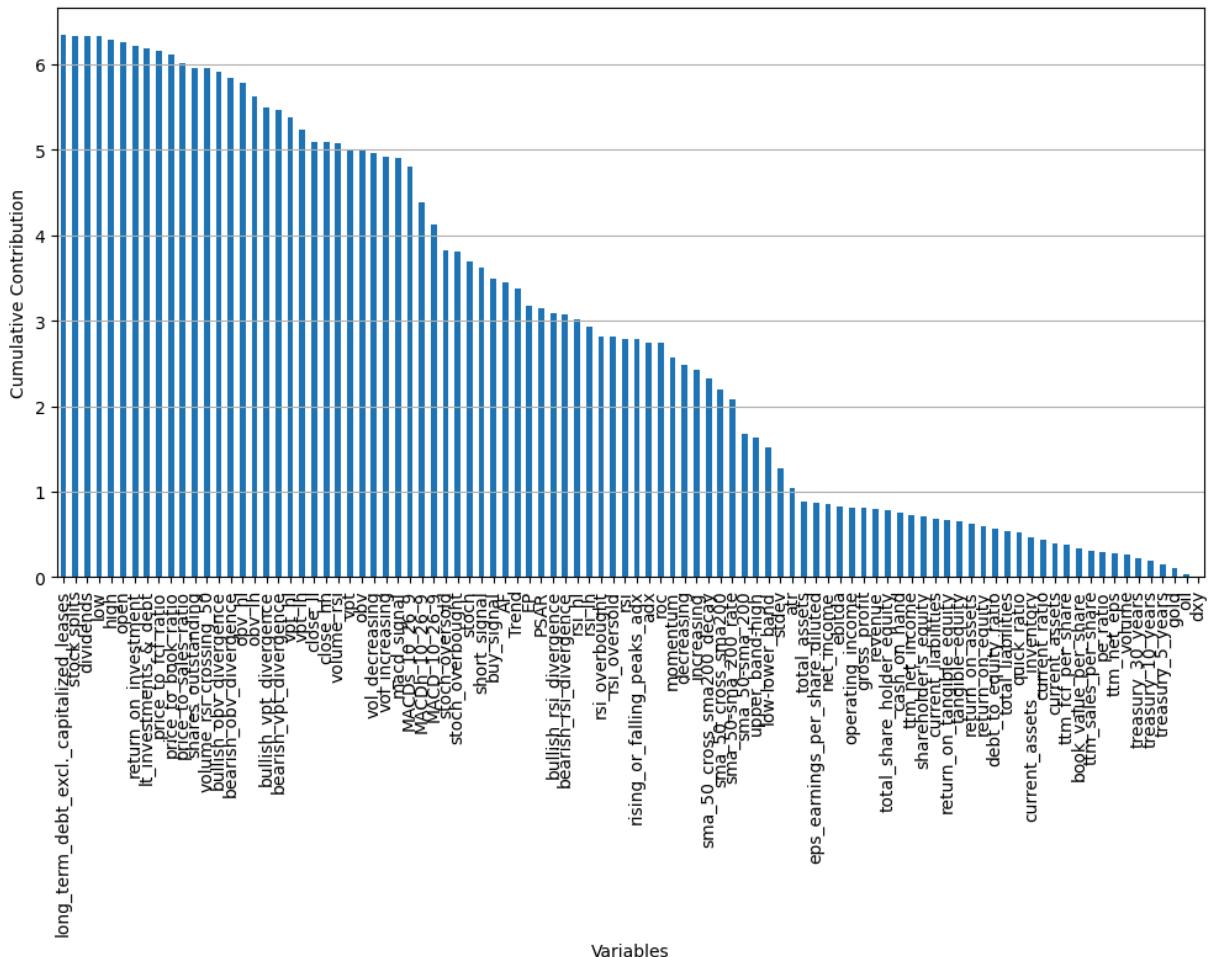


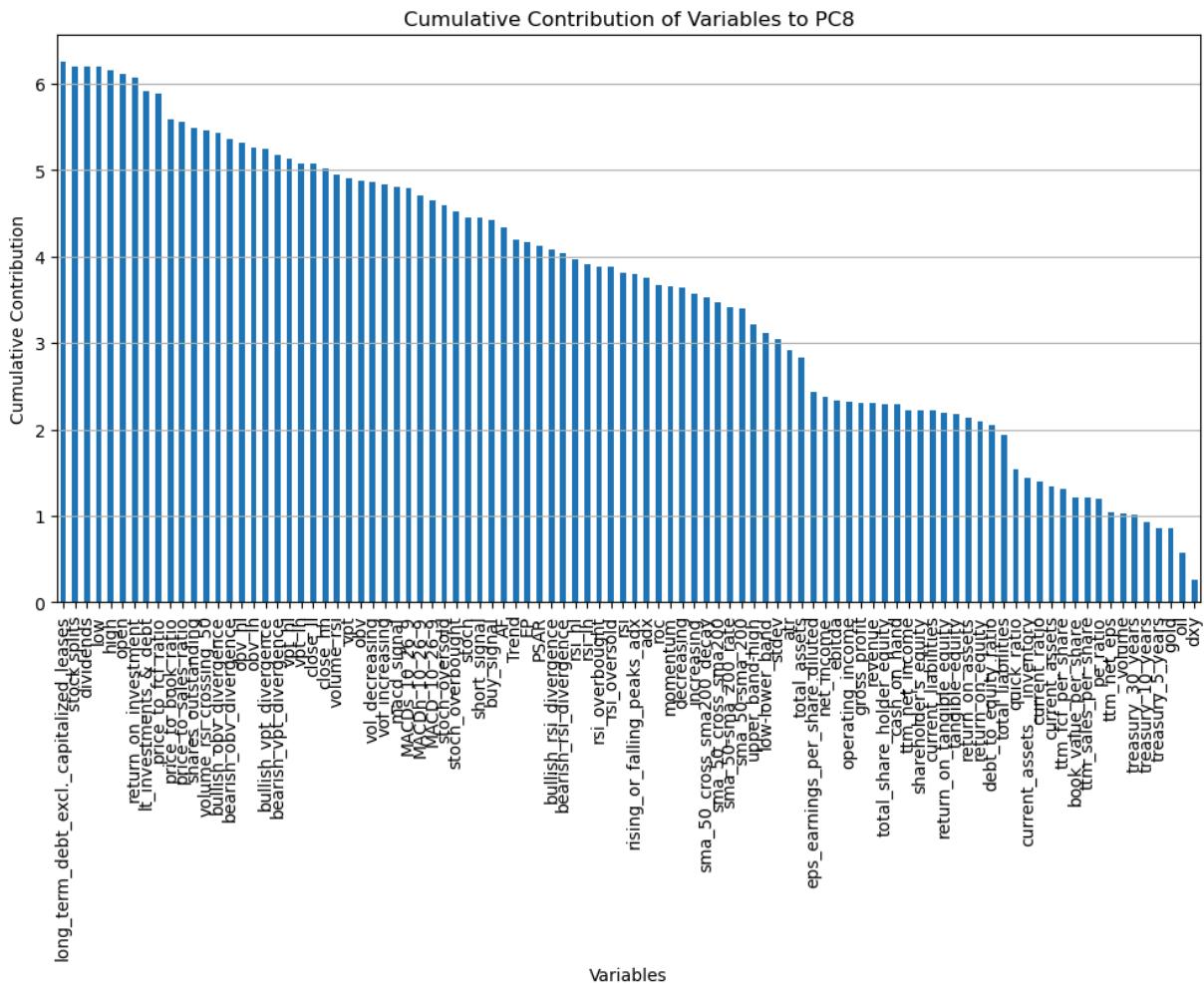




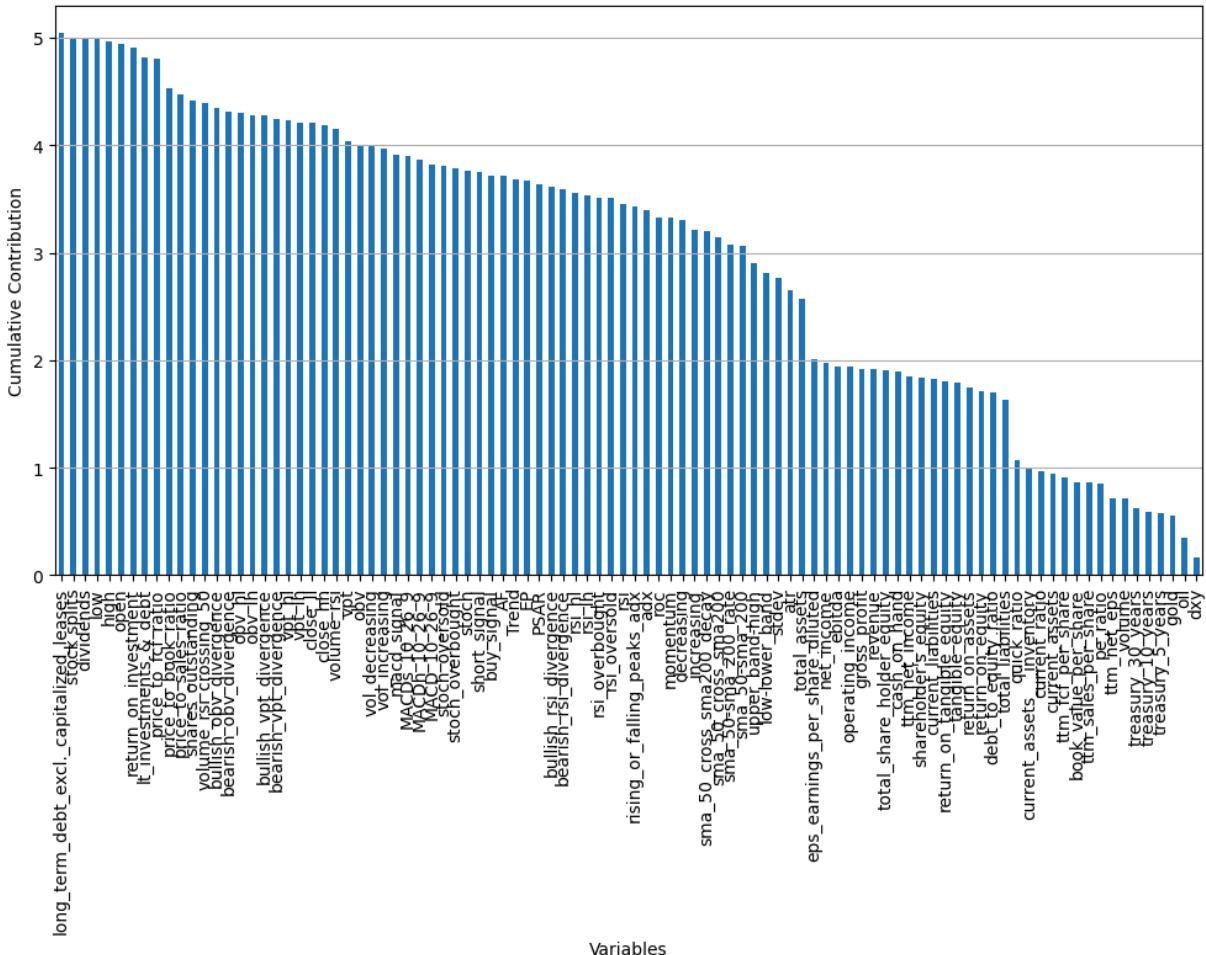


## Cumulative Contribution of Variables to PC7

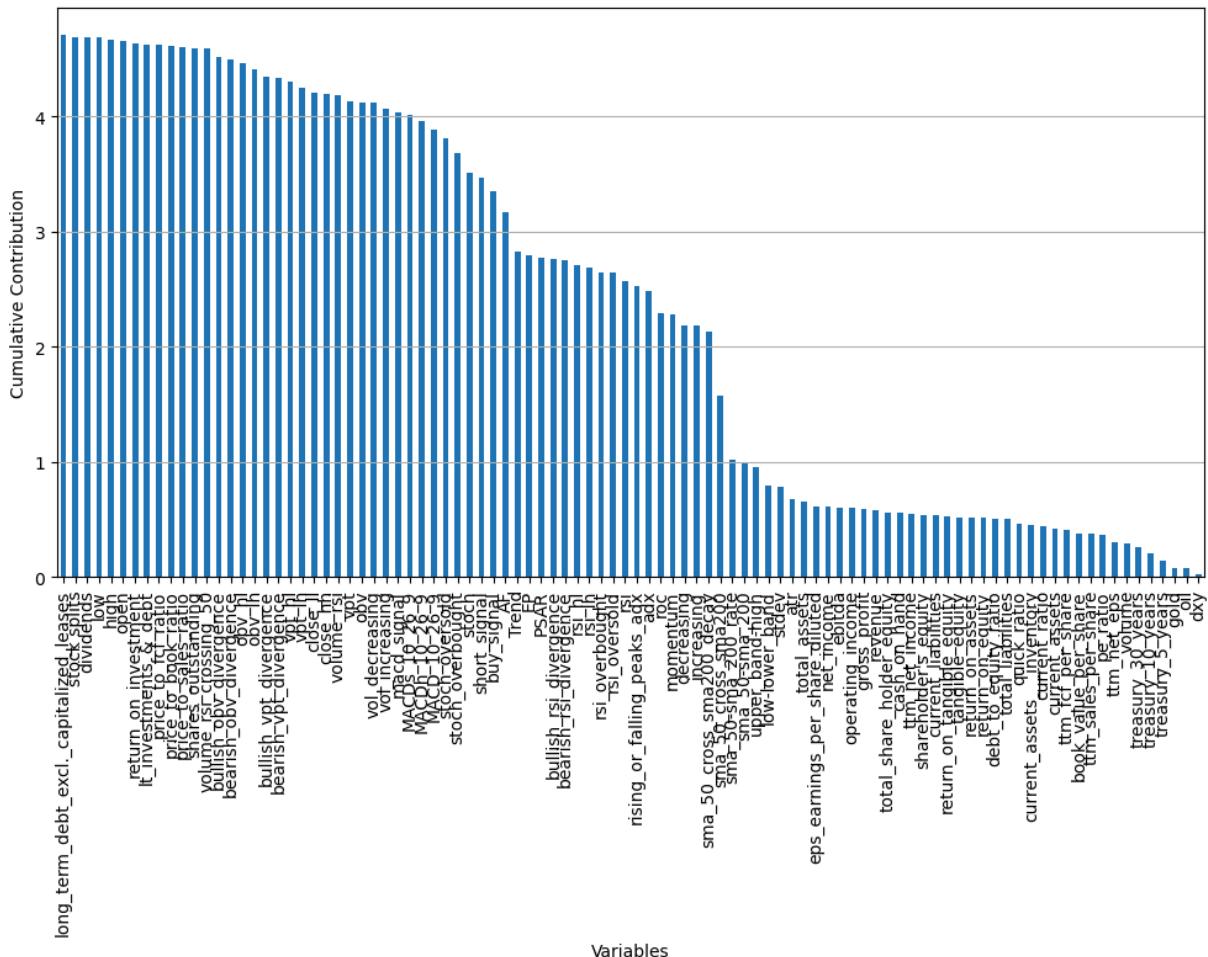


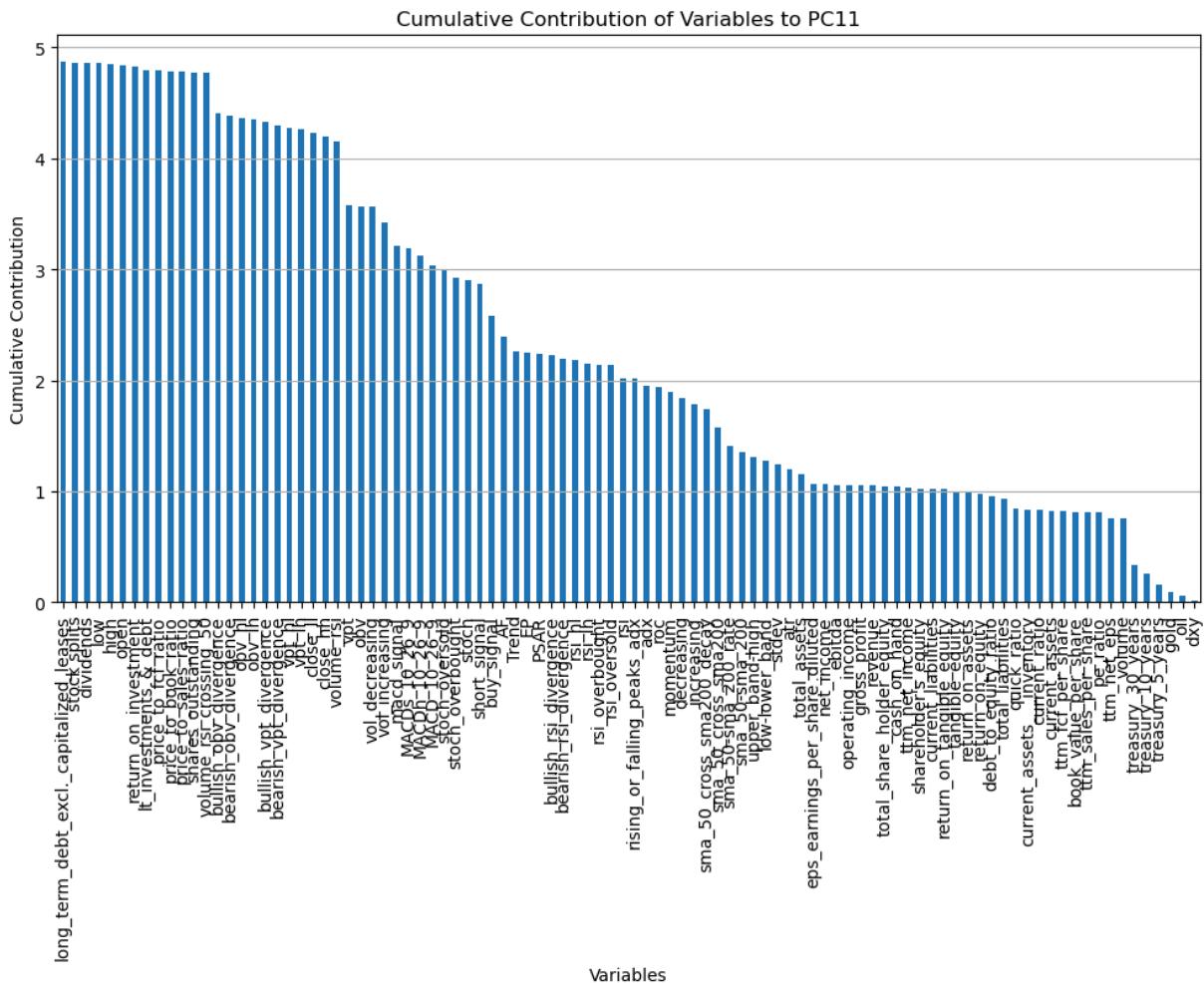


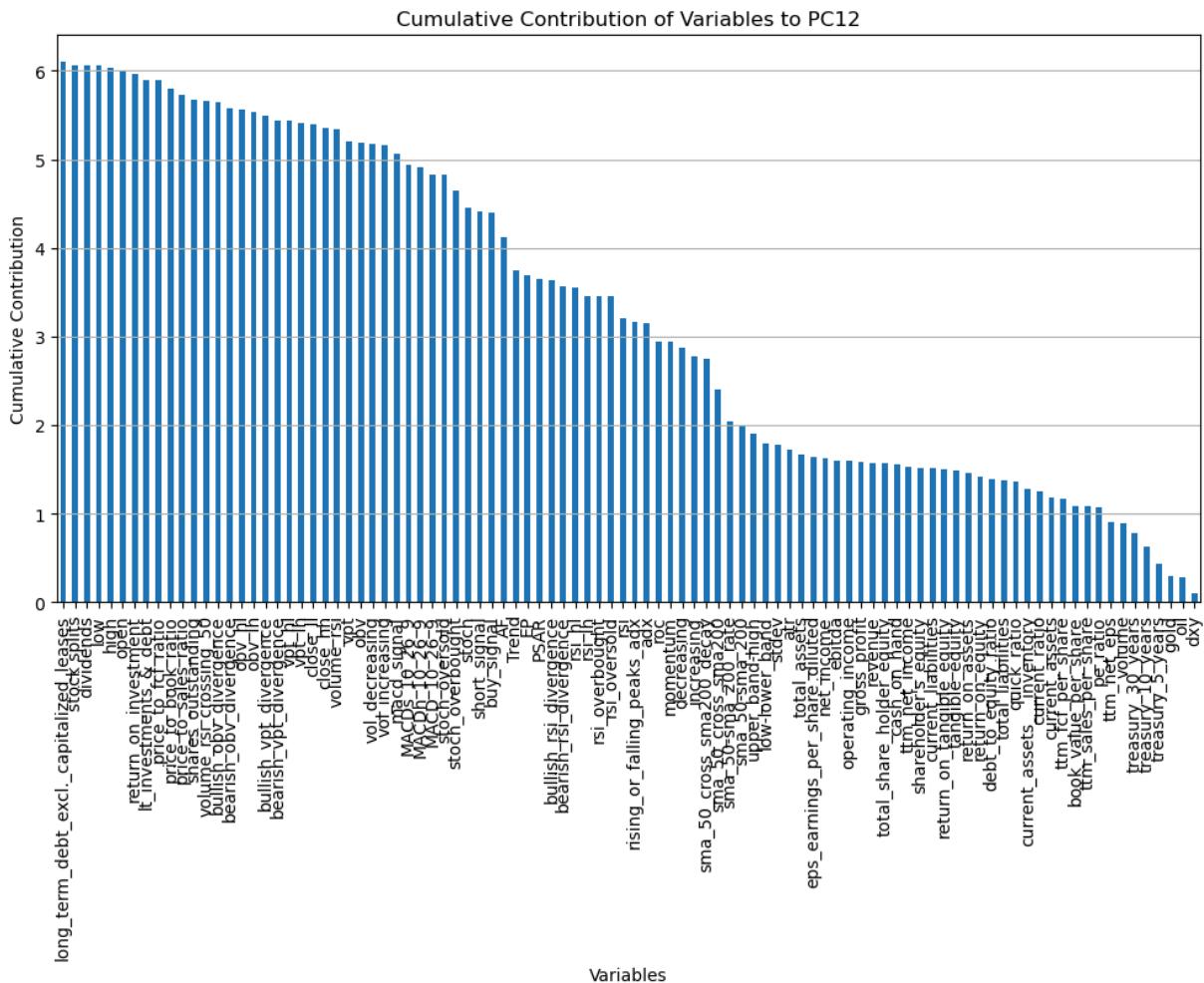
### Cumulative Contribution of Variables to PC9

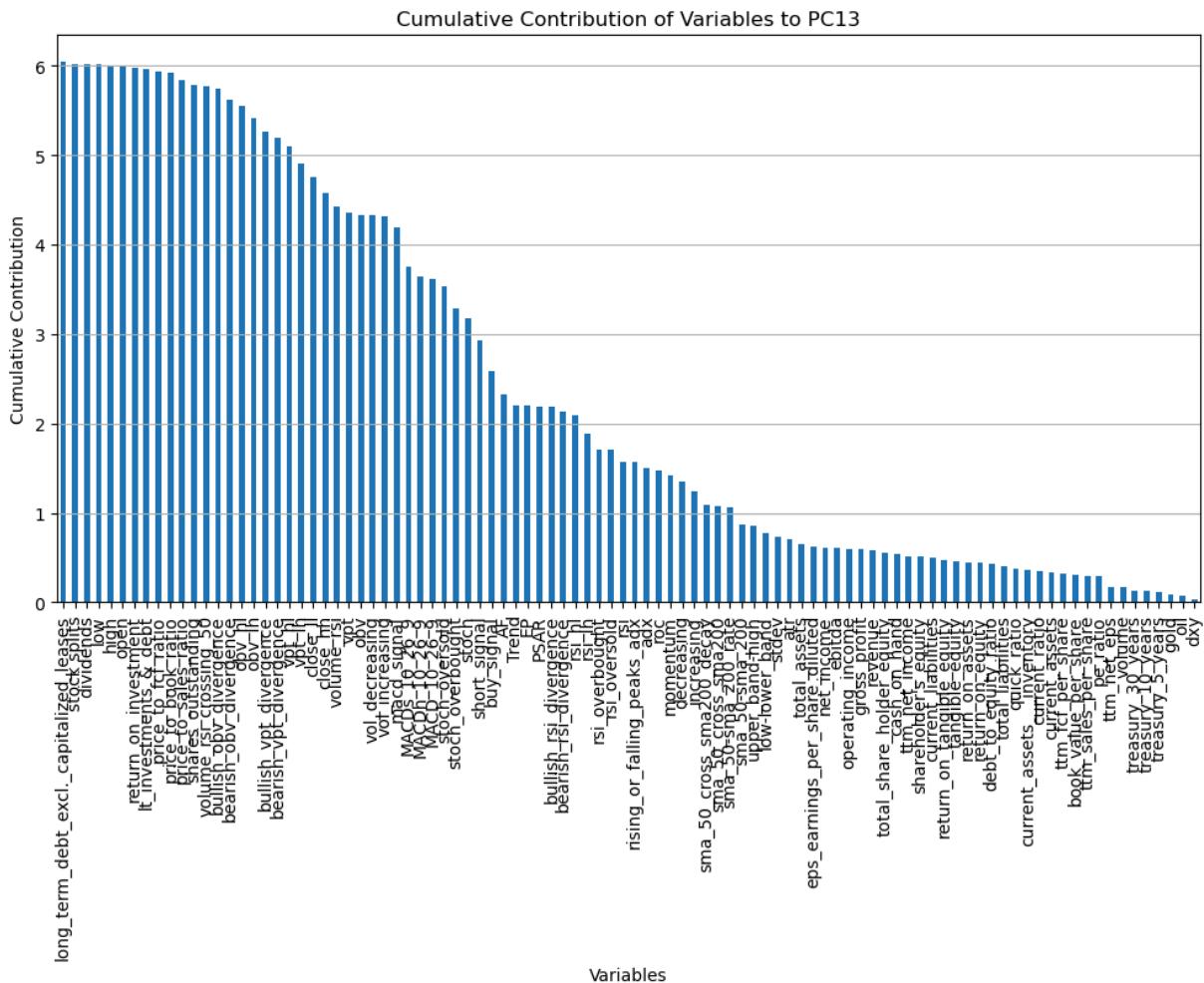


## Cumulative Contribution of Variables to PC10

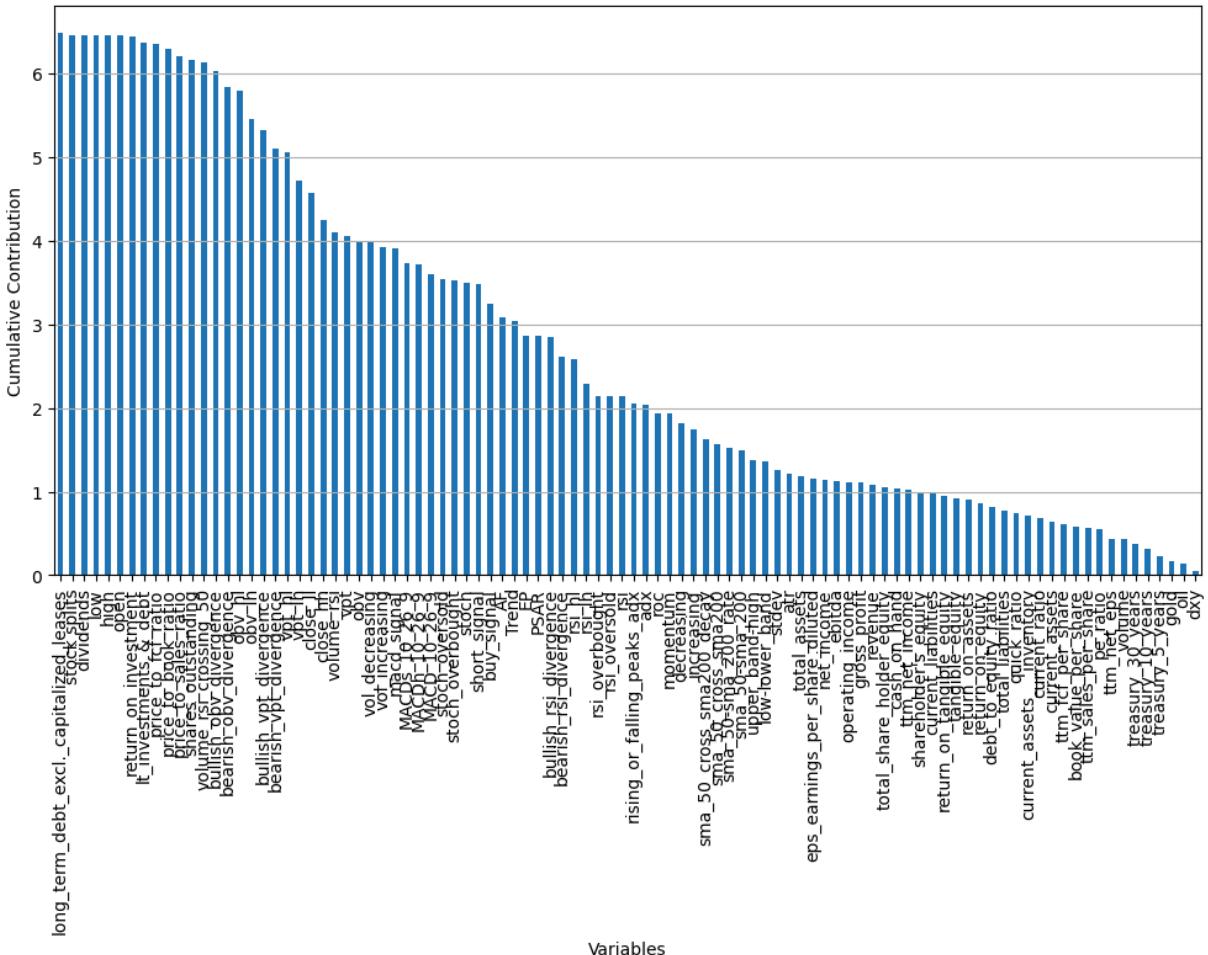


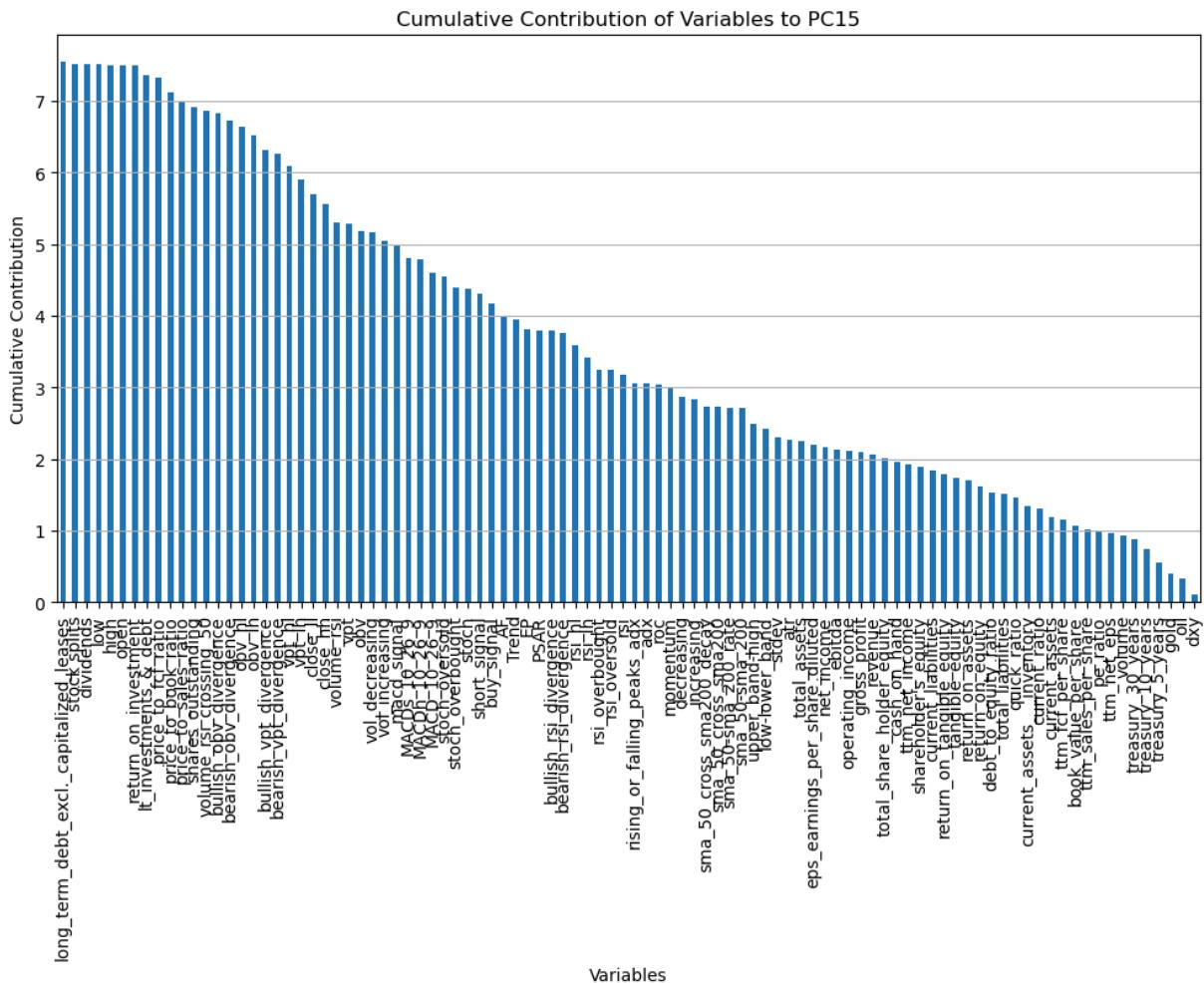


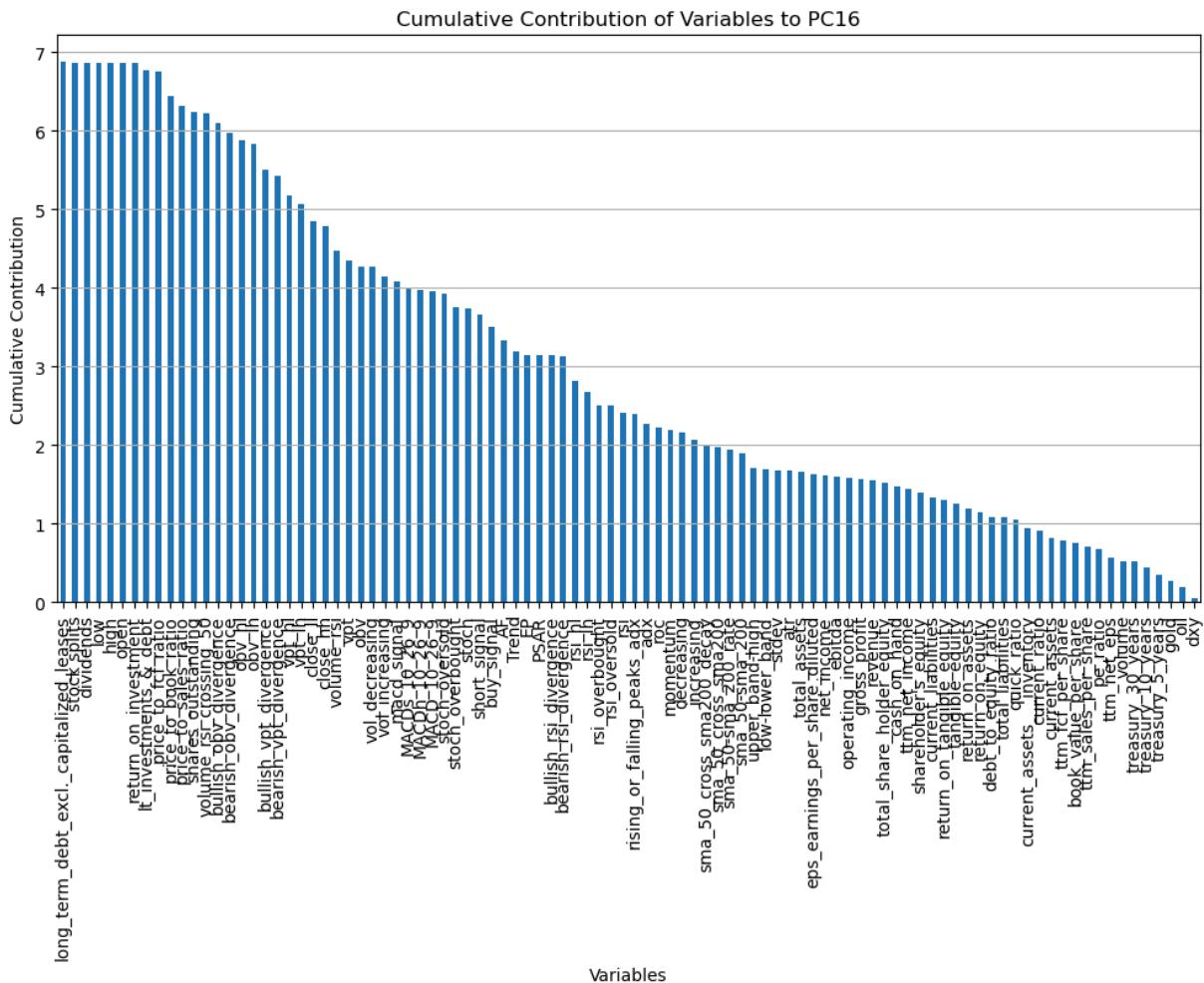




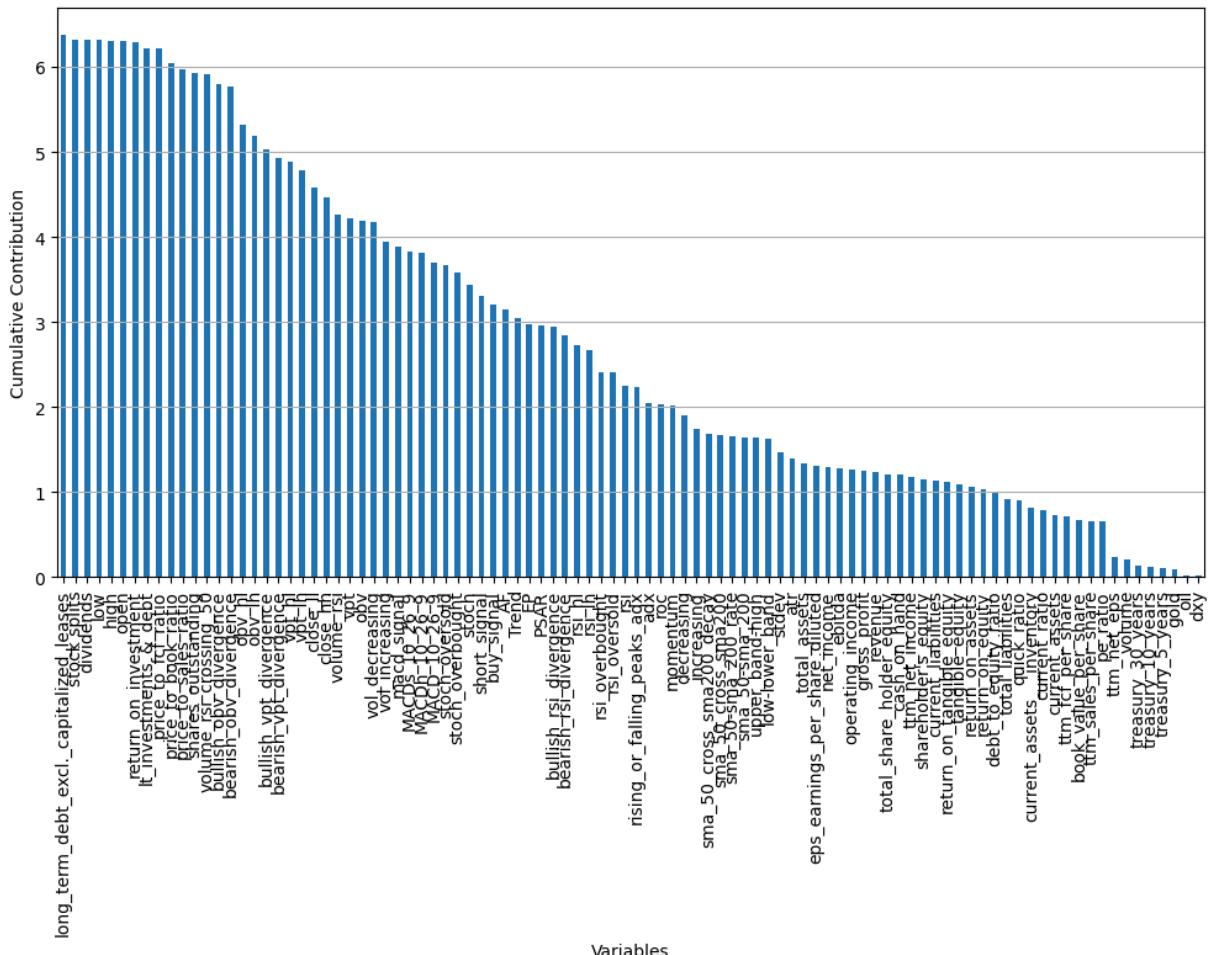
## Cumulative Contribution of Variables to PC14



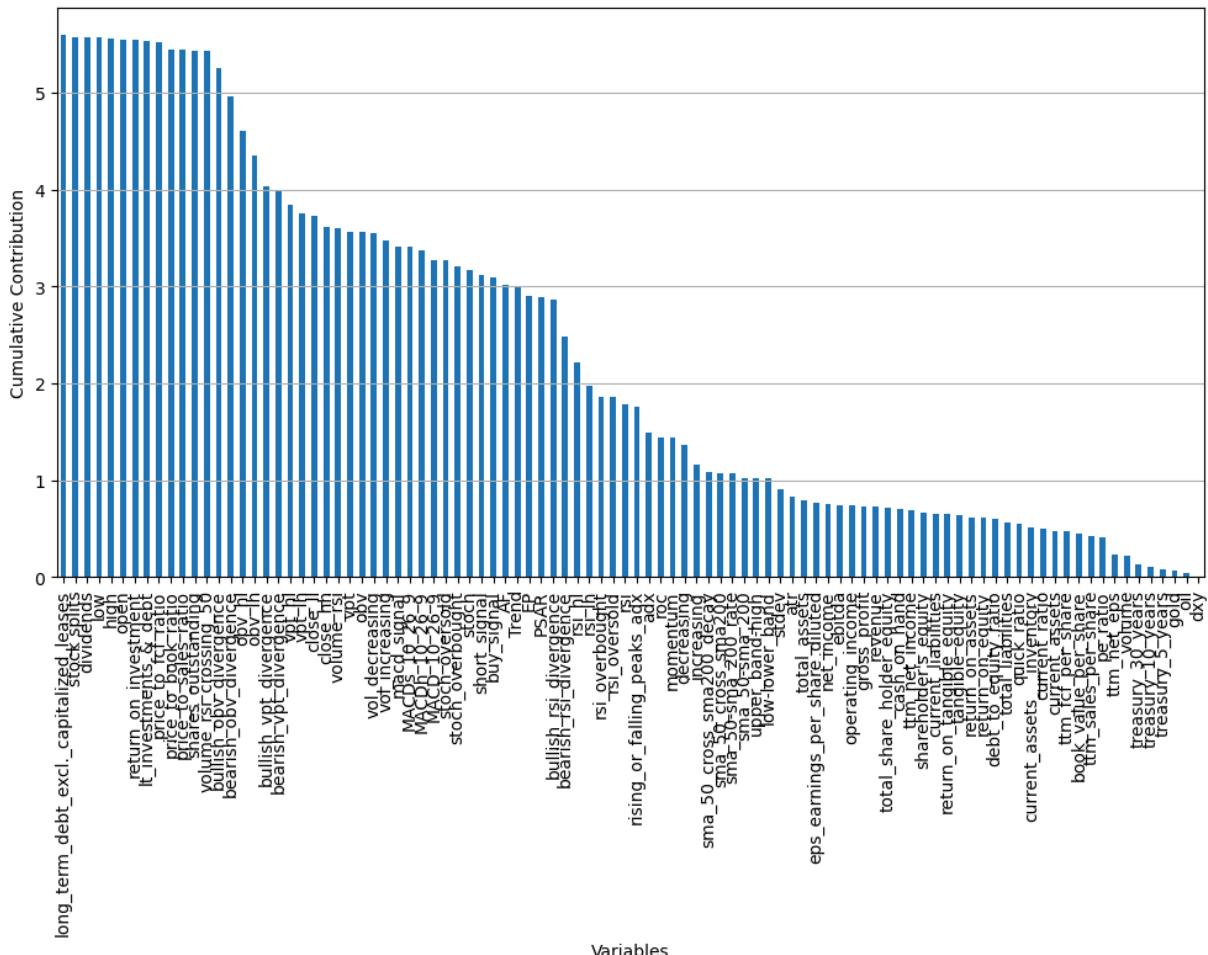




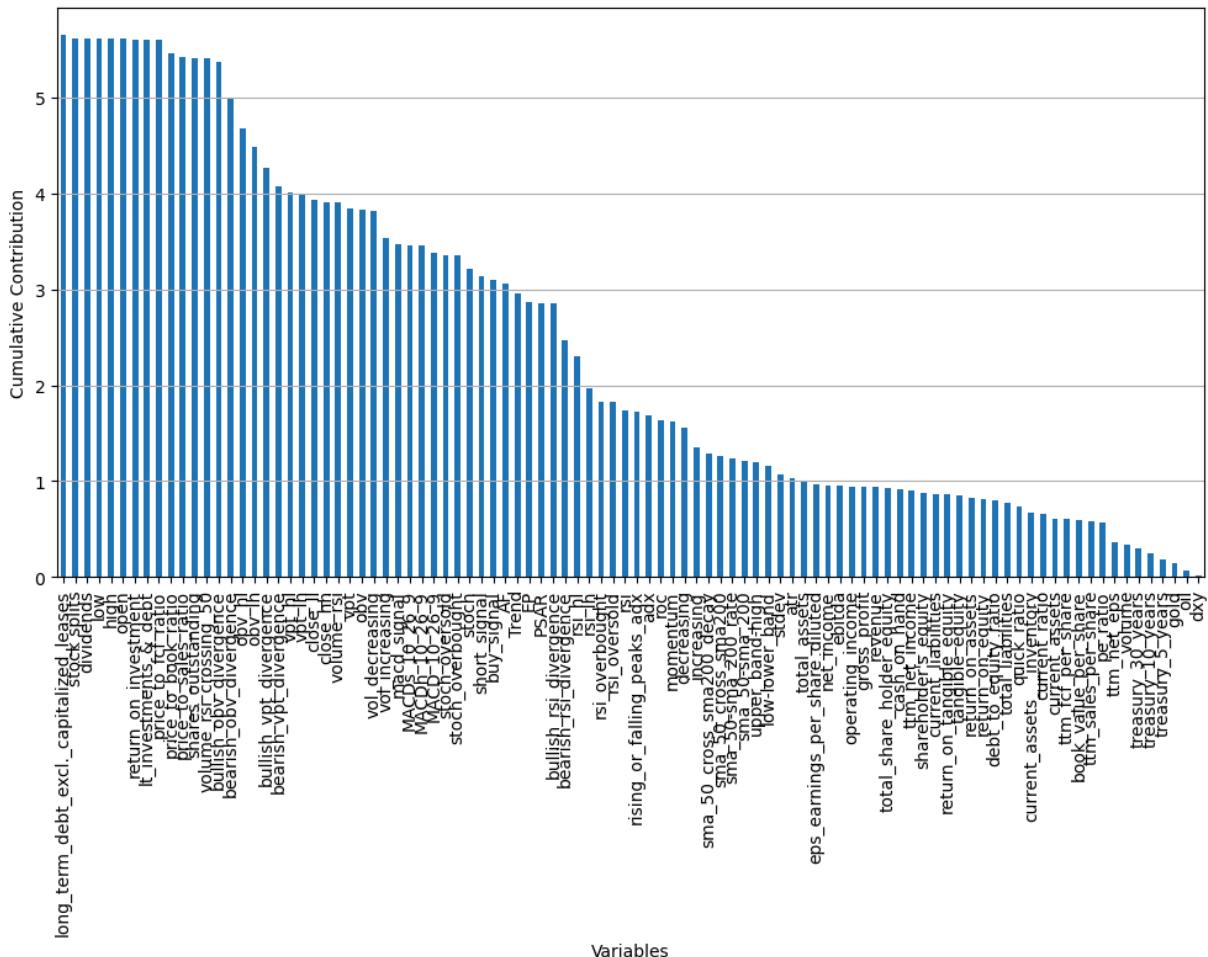
## Cumulative Contribution of Variables to PC17

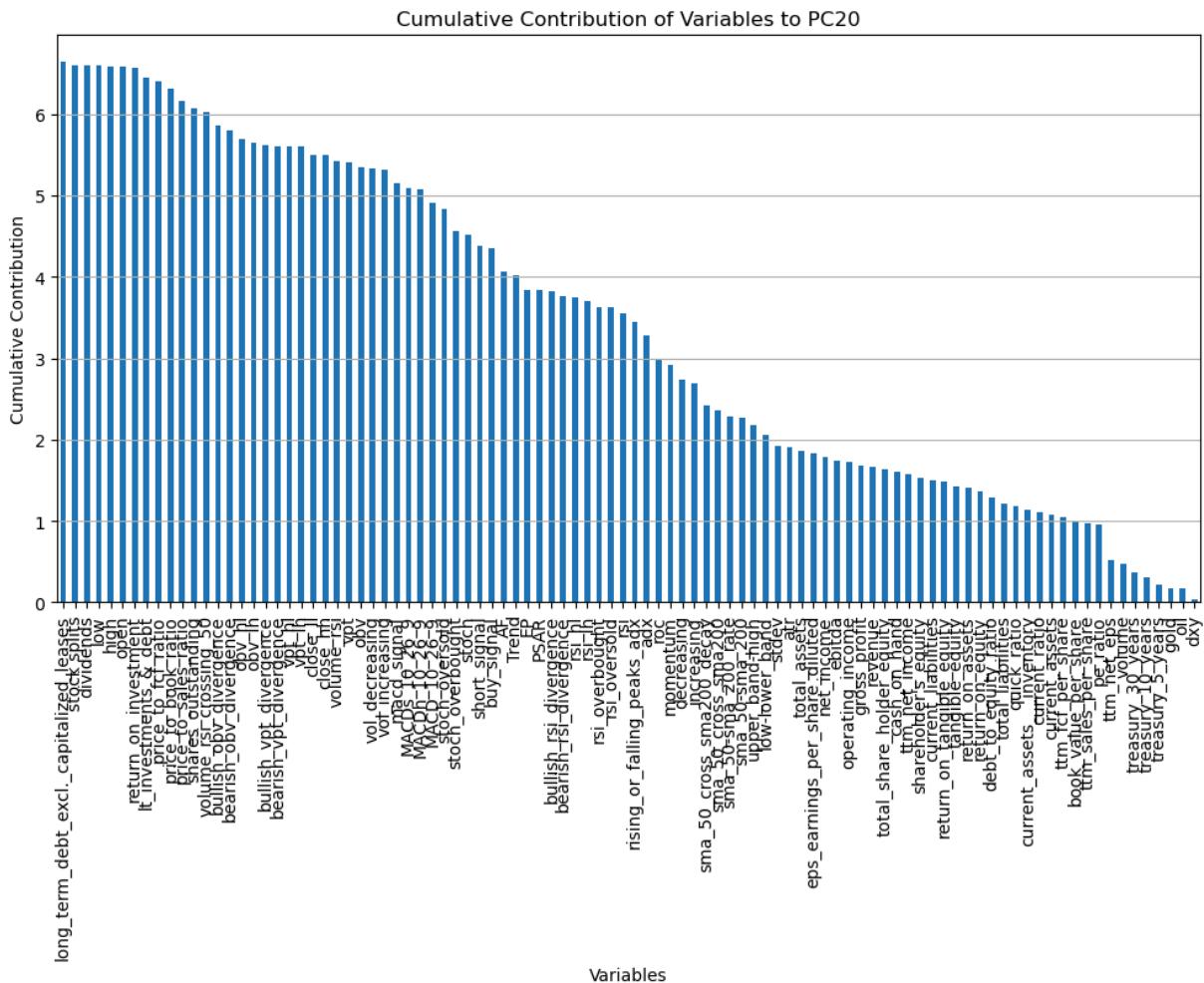


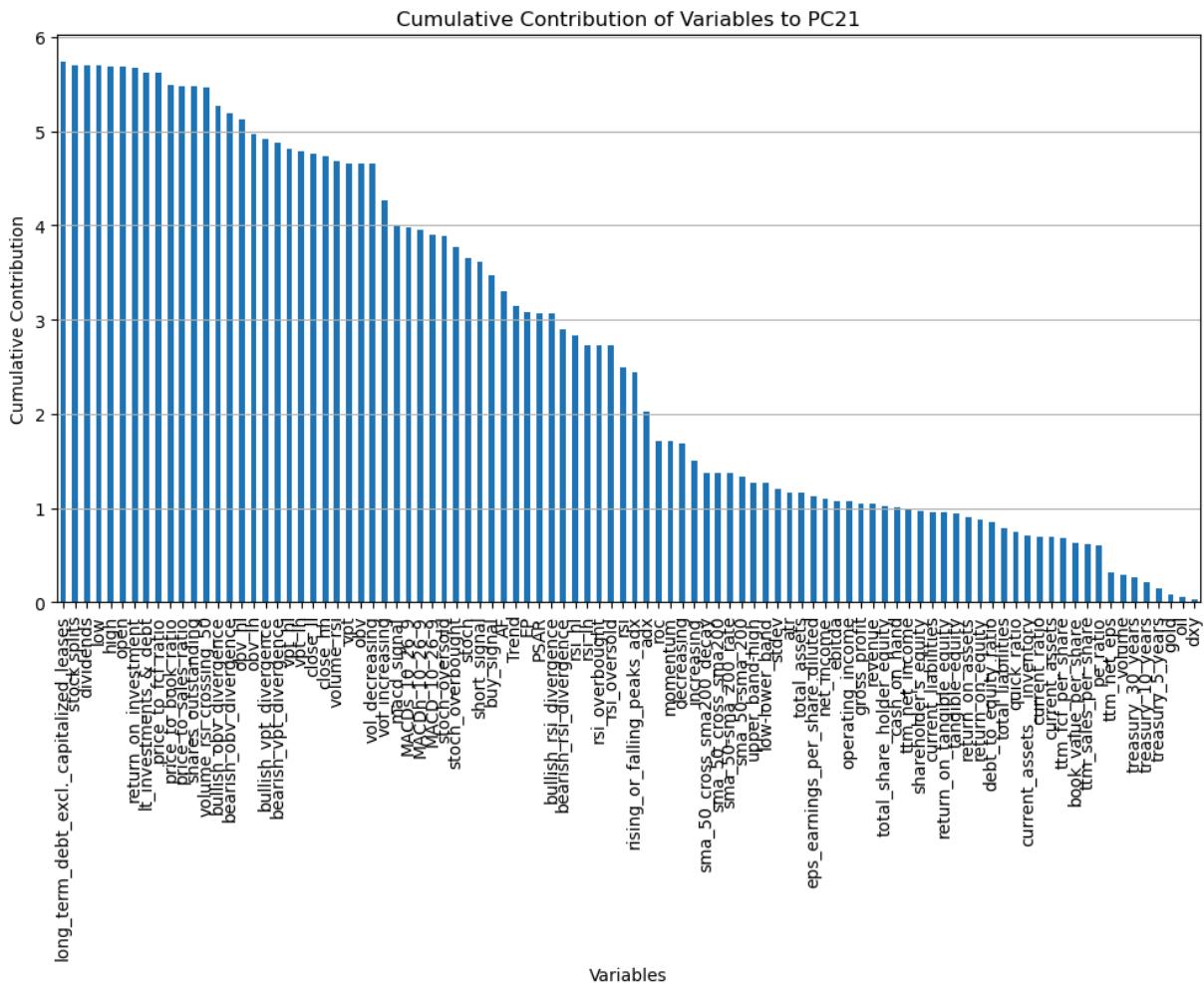
## Cumulative Contribution of Variables to PC18

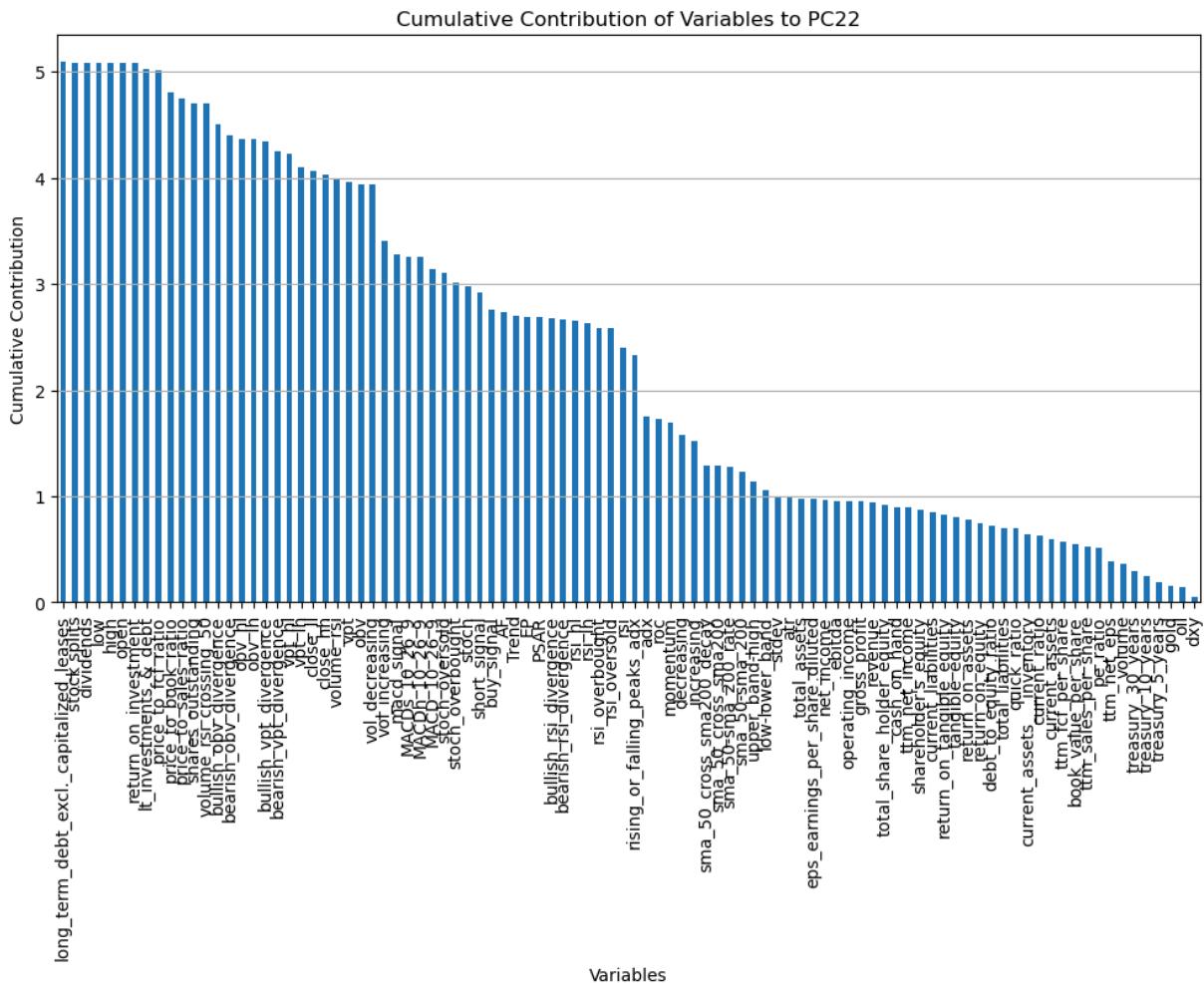


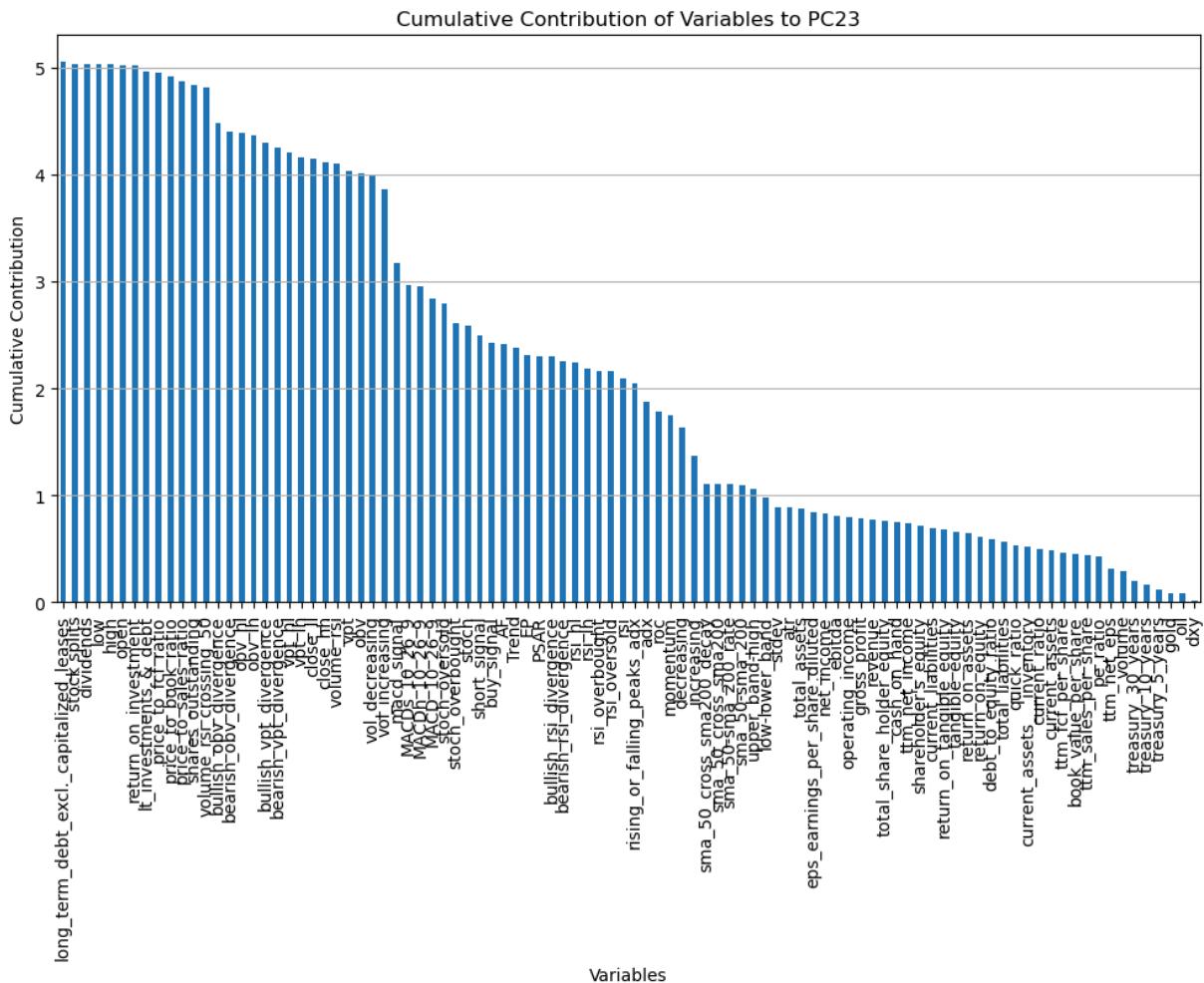
## Cumulative Contribution of Variables to PC19



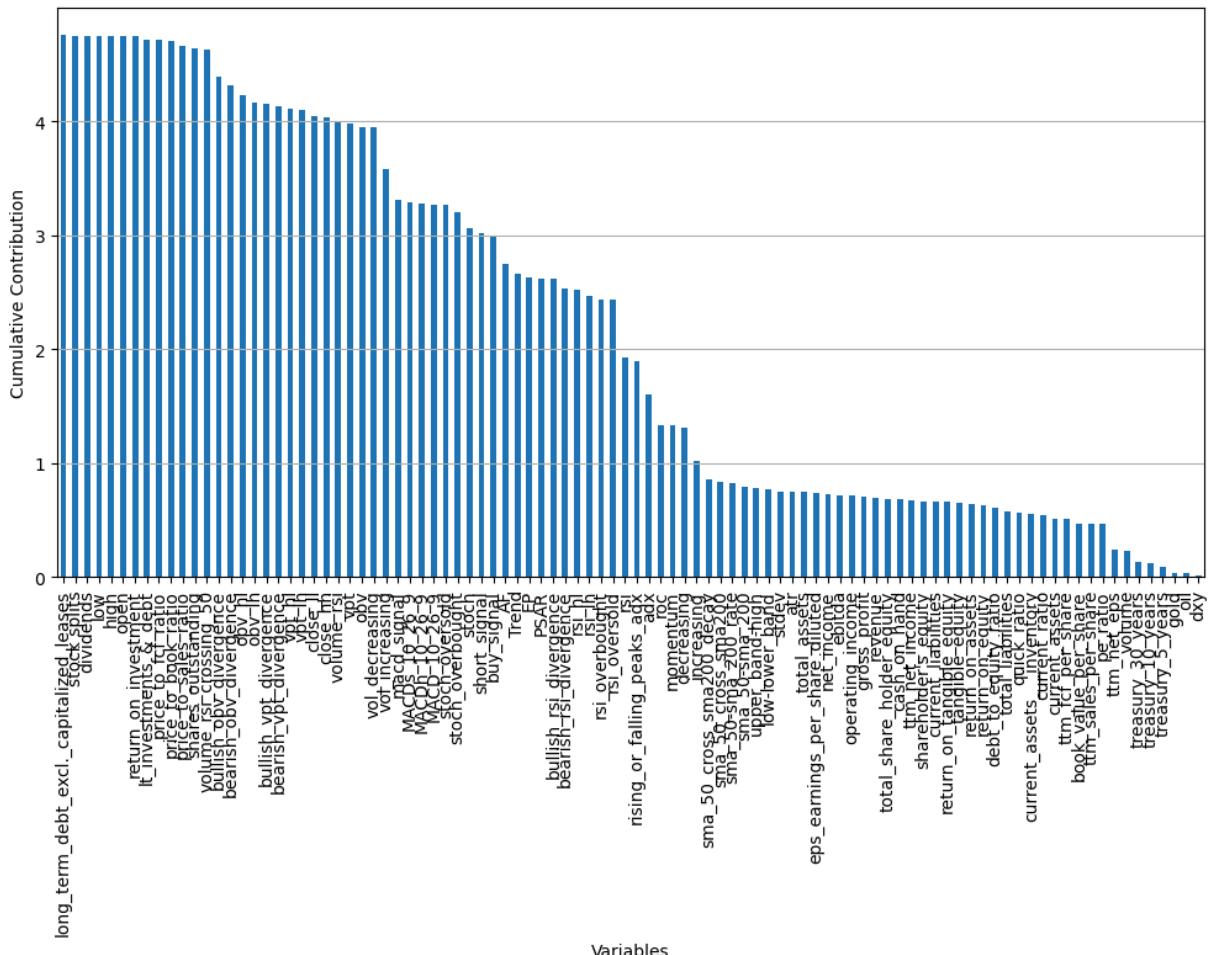




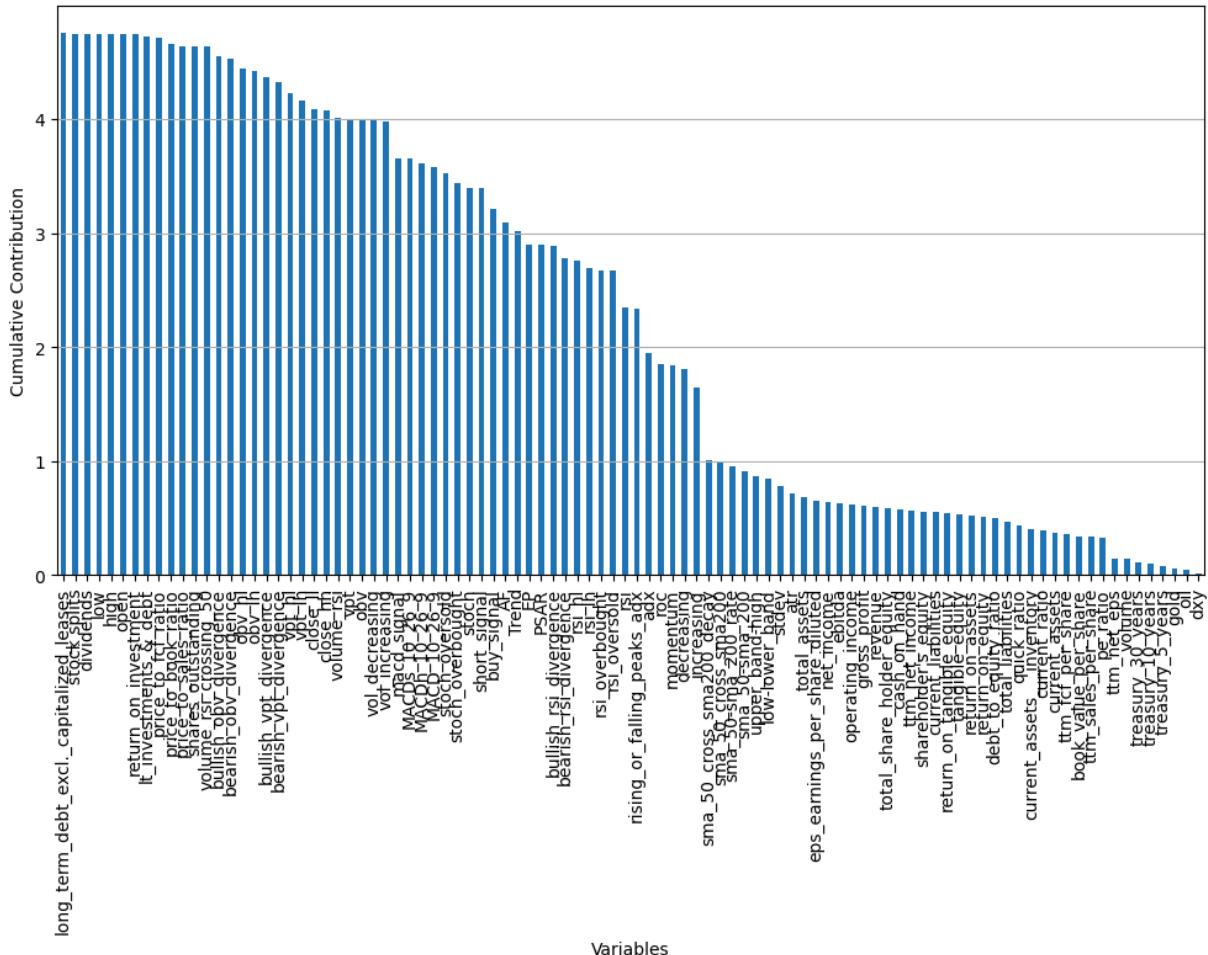




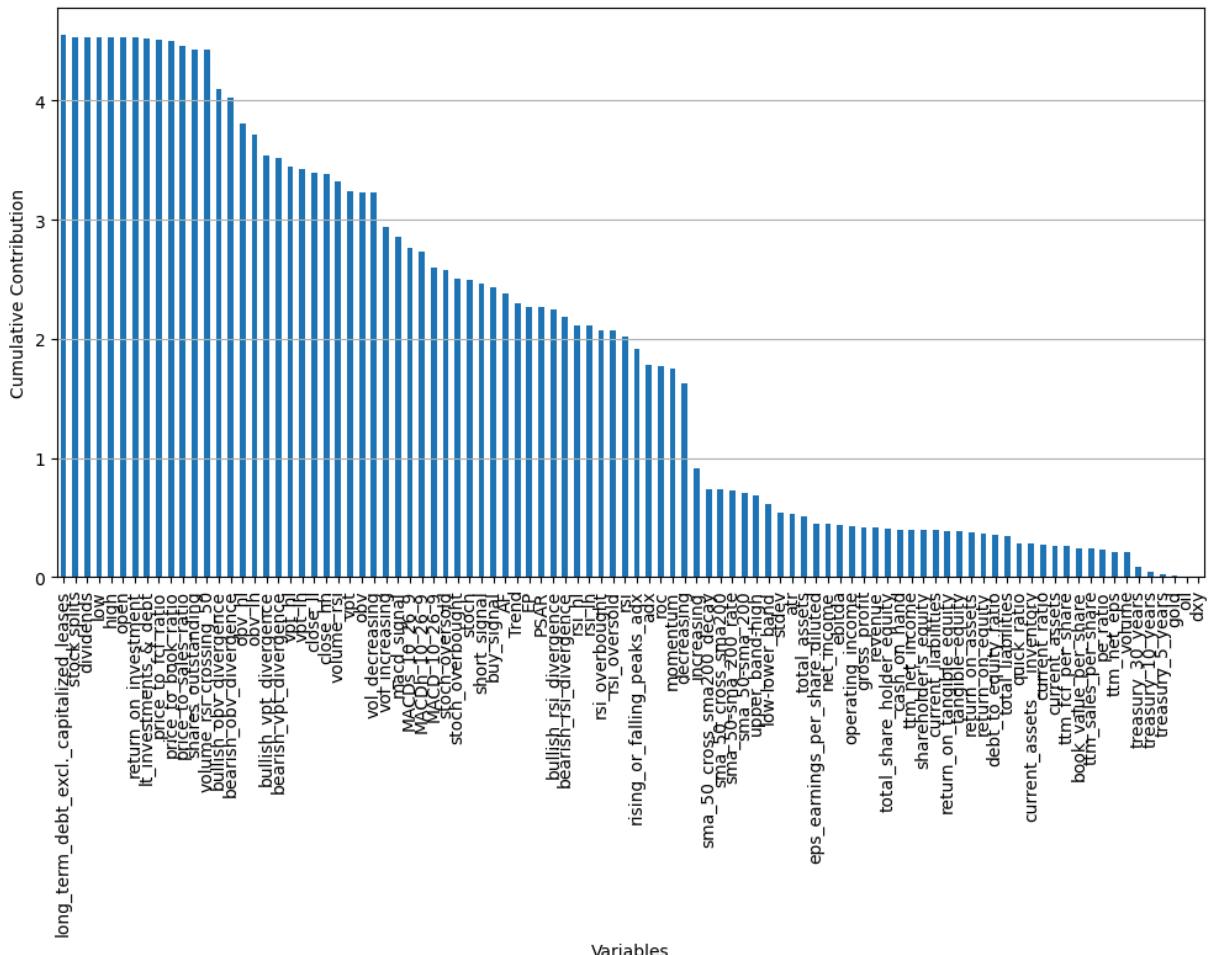
## Cumulative Contribution of Variables to PC24

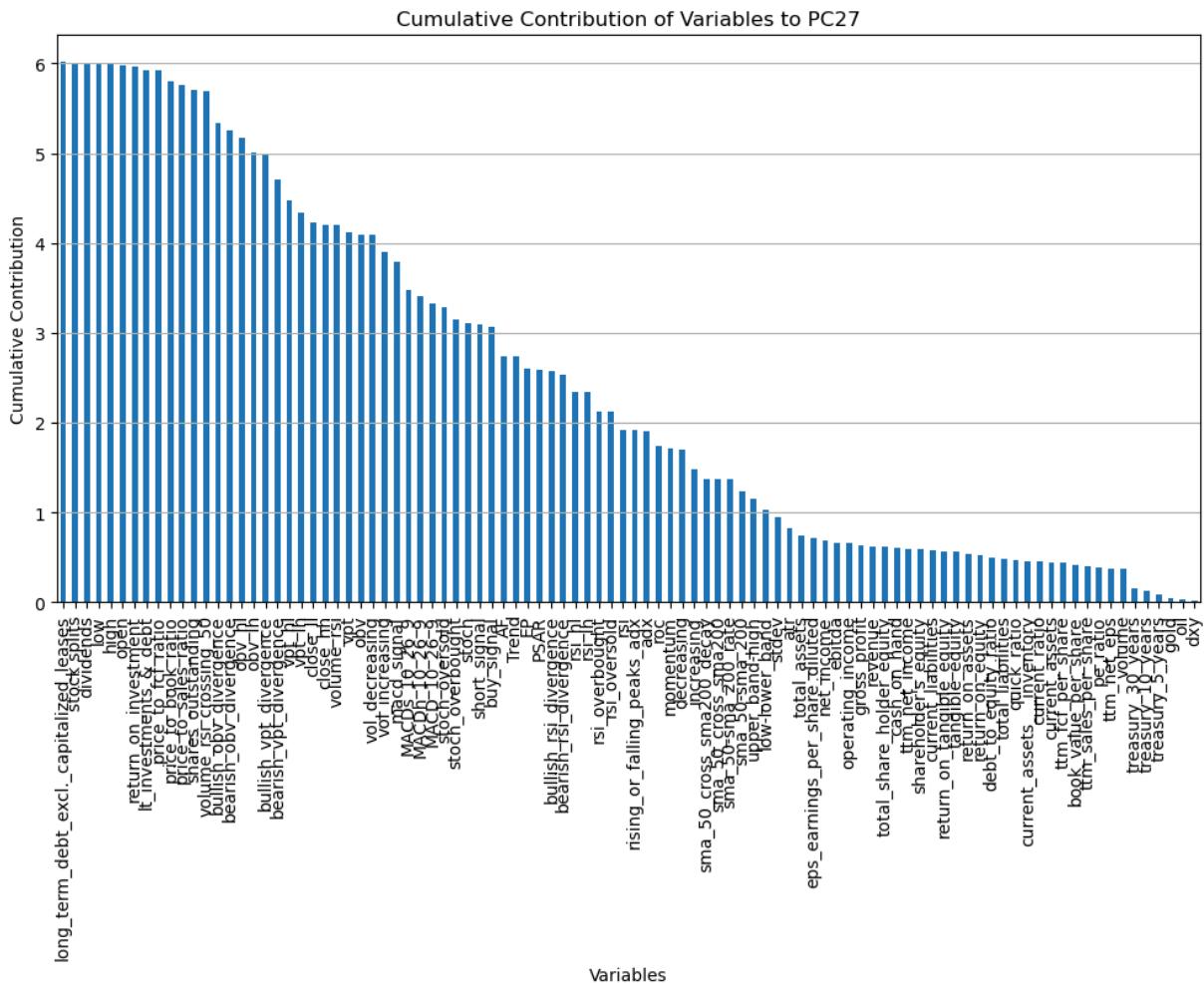


## Cumulative Contribution of Variables to PC25

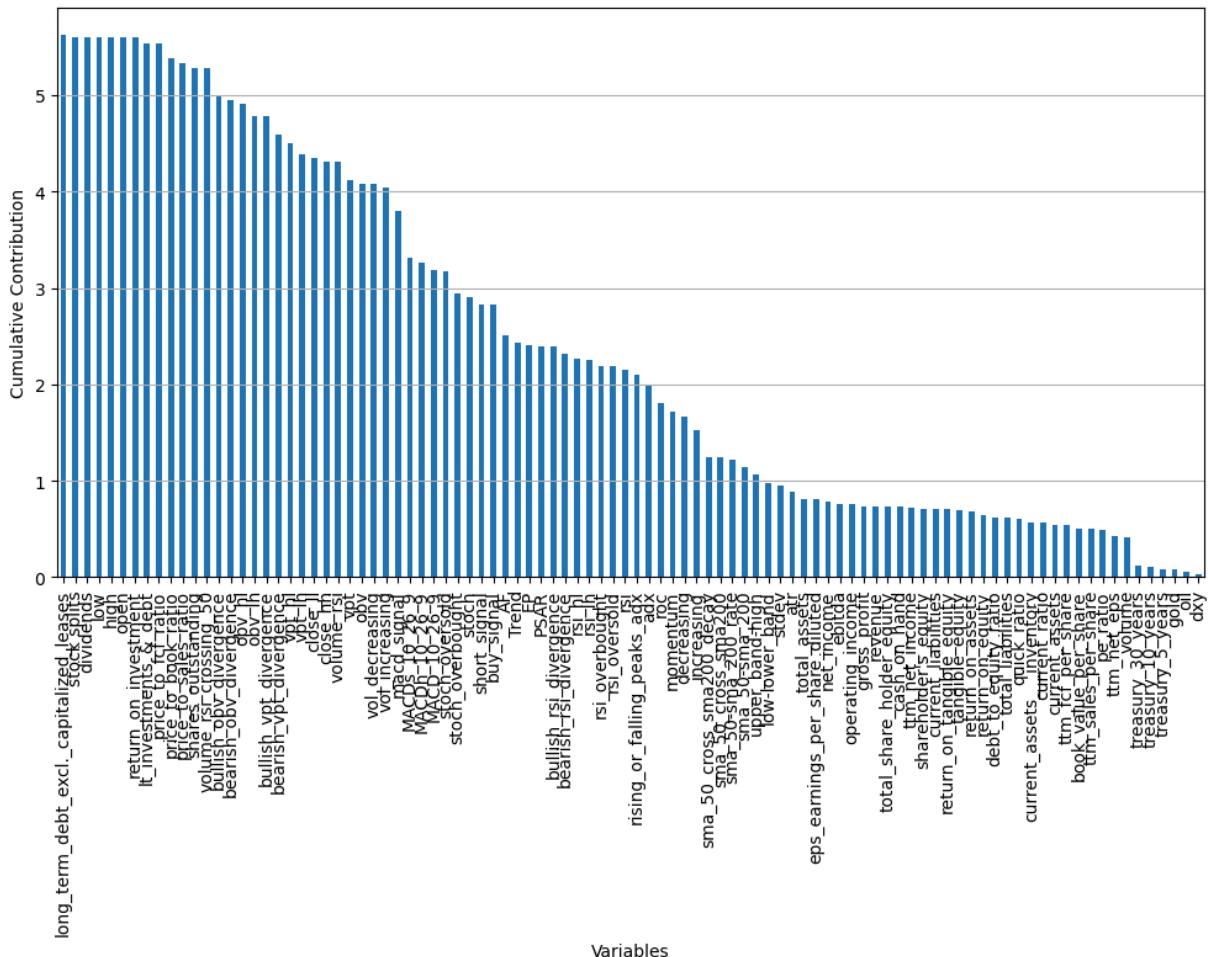


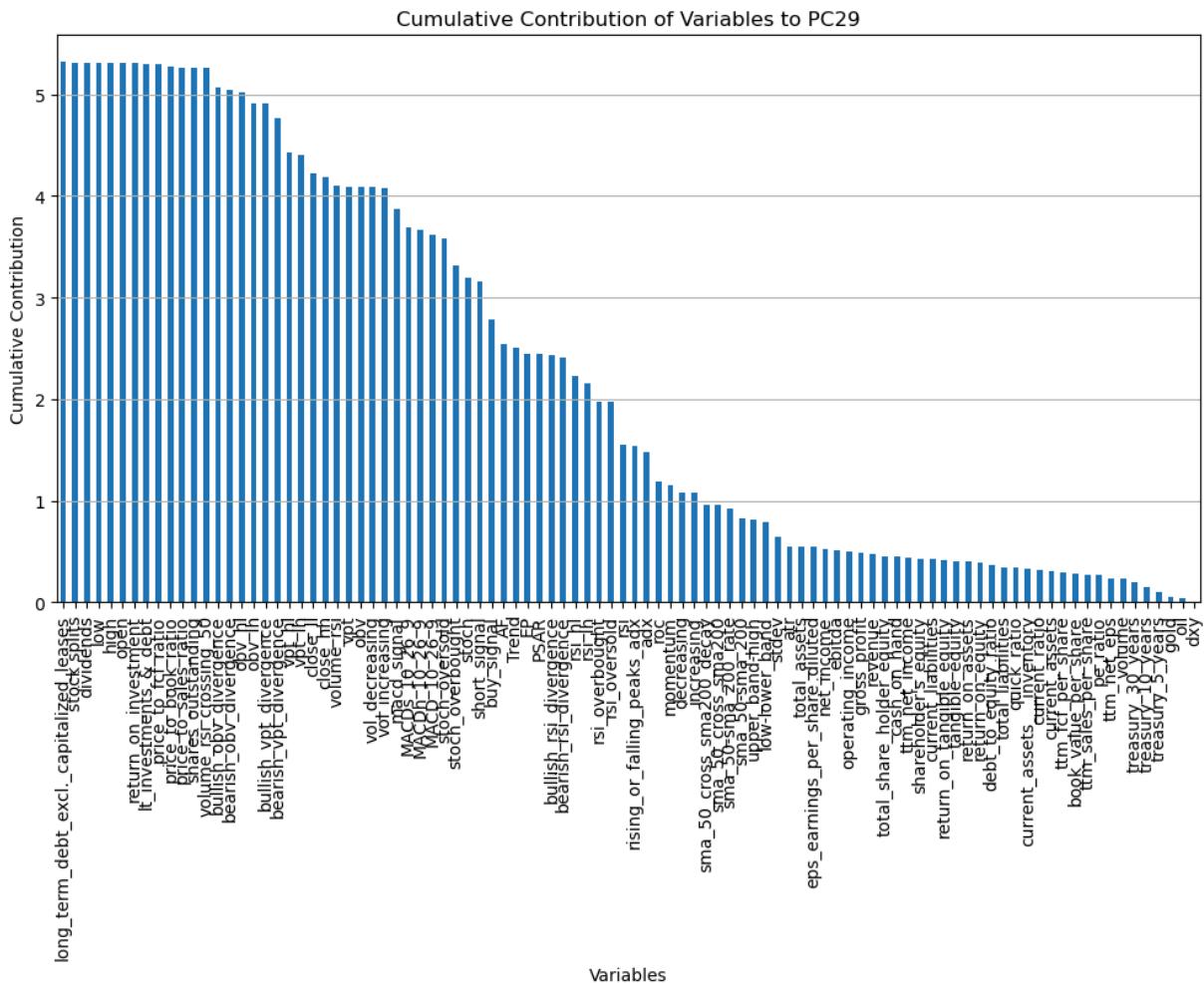
## Cumulative Contribution of Variables to PC26

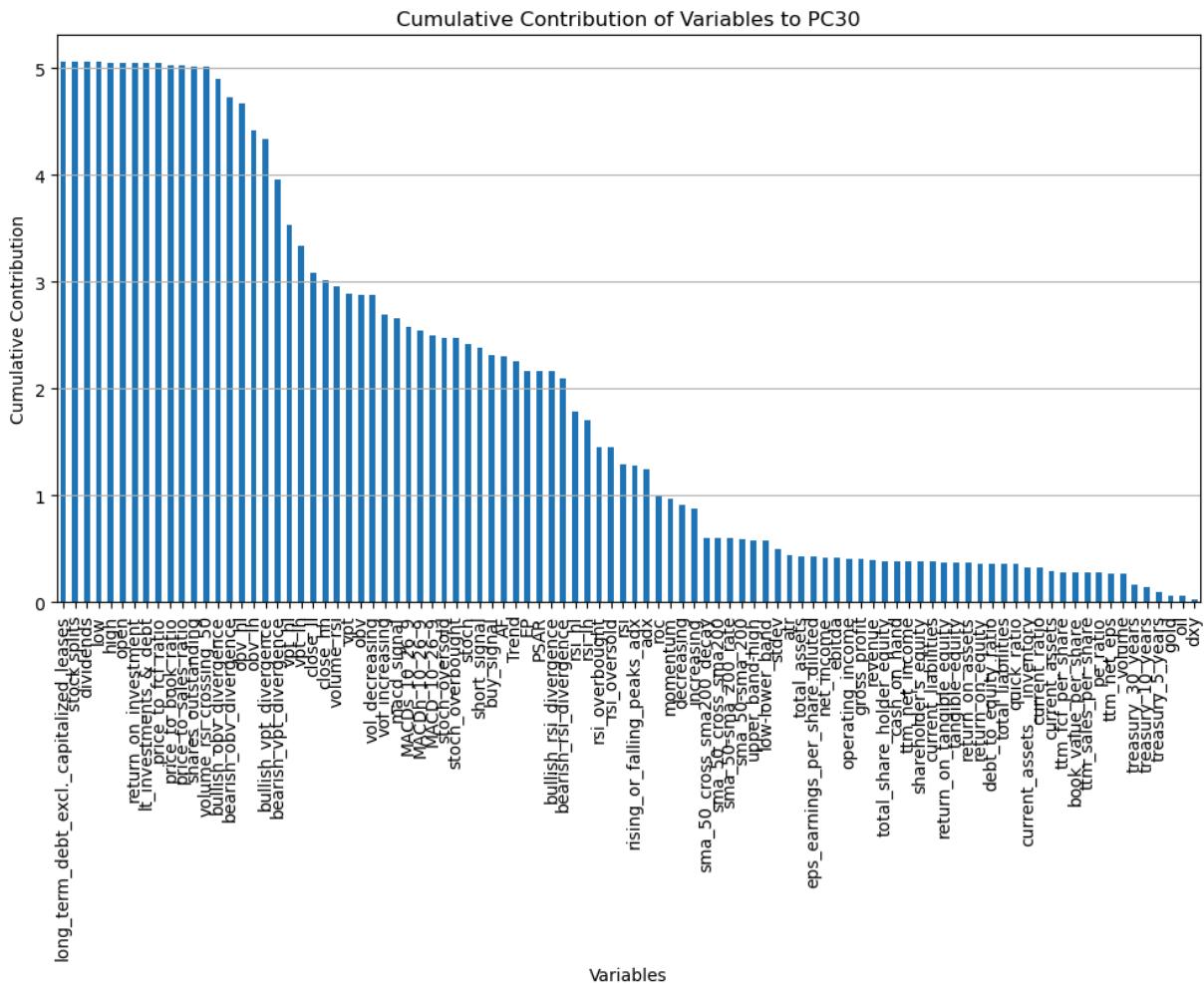


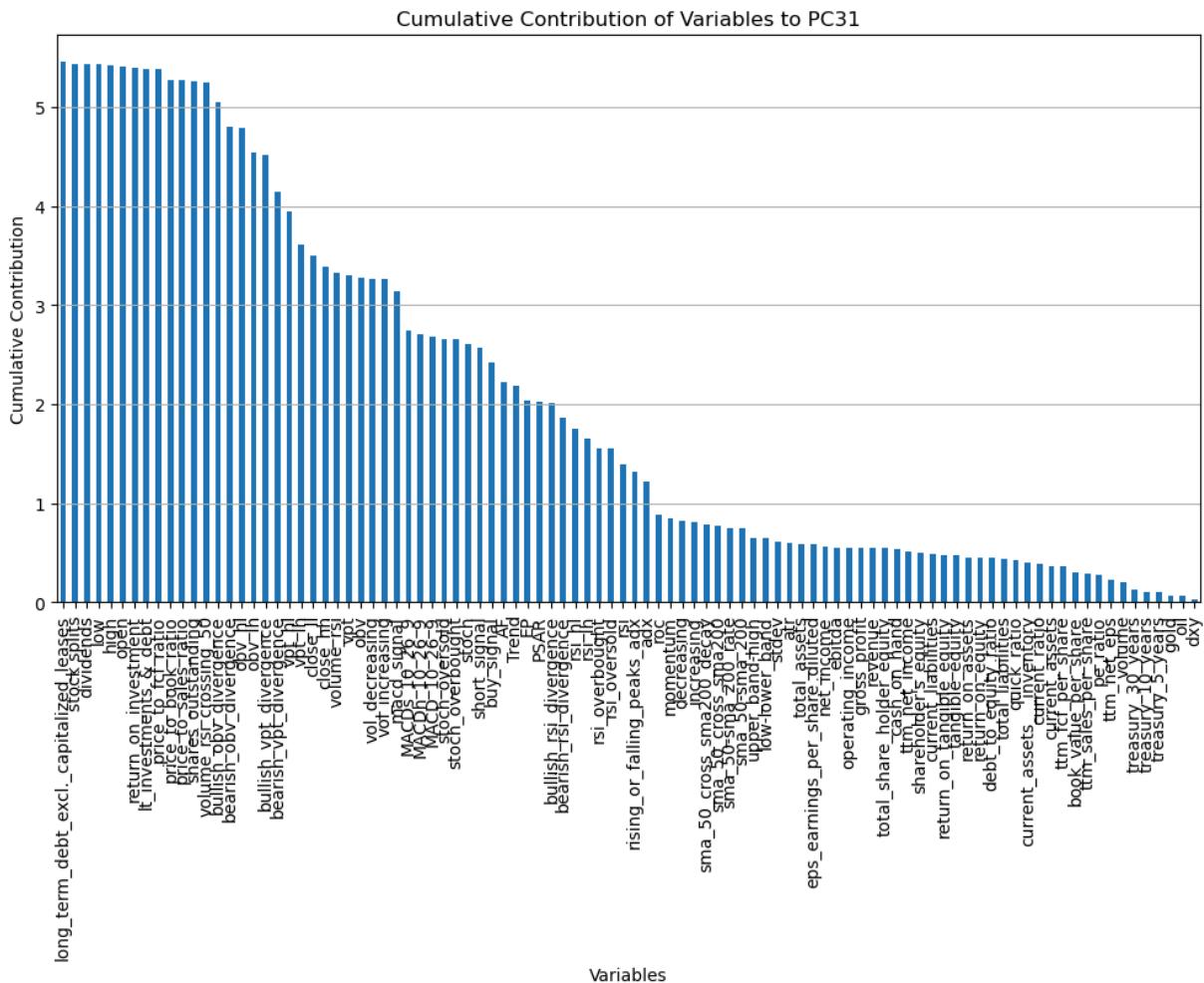


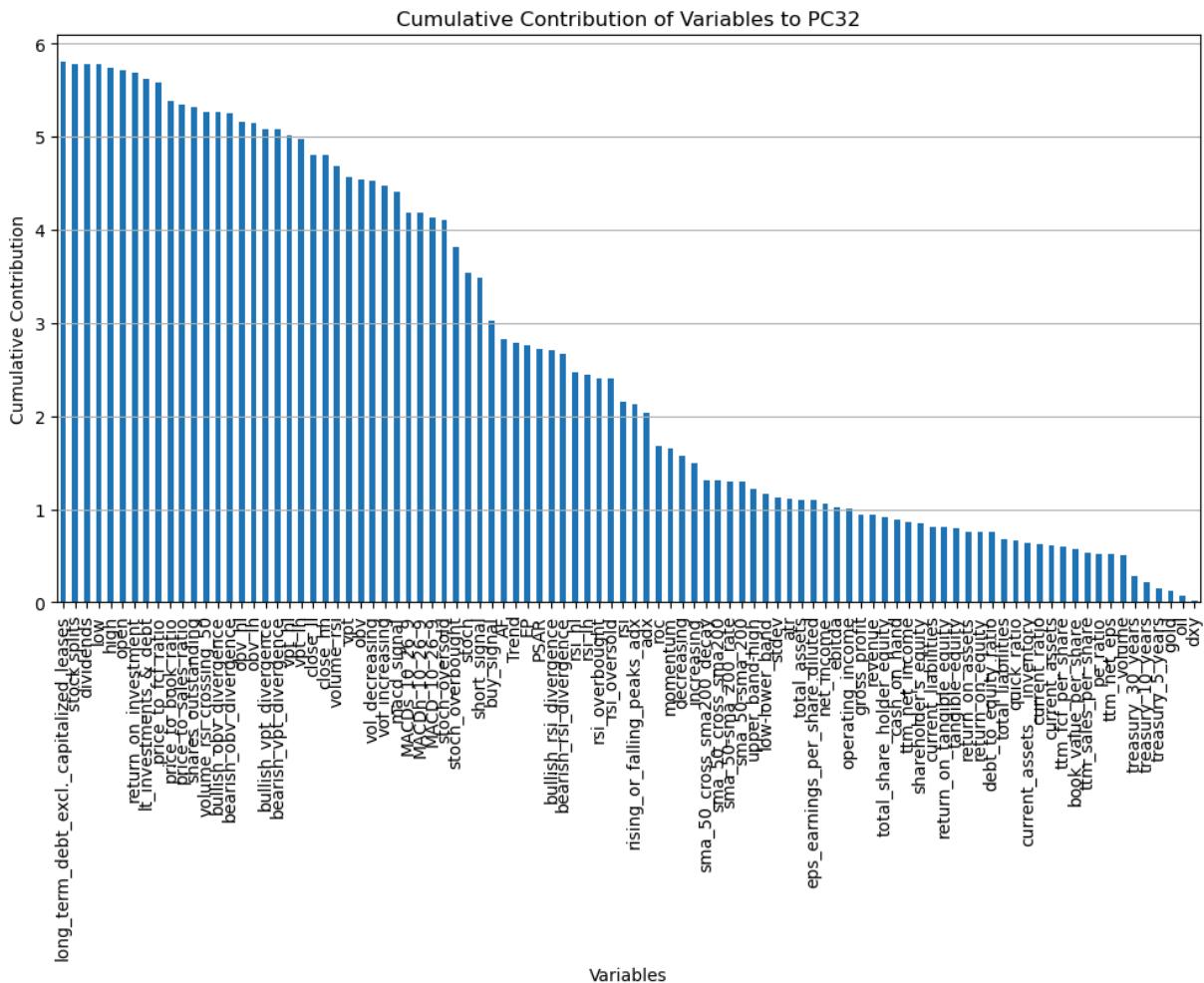
## Cumulative Contribution of Variables to PC28

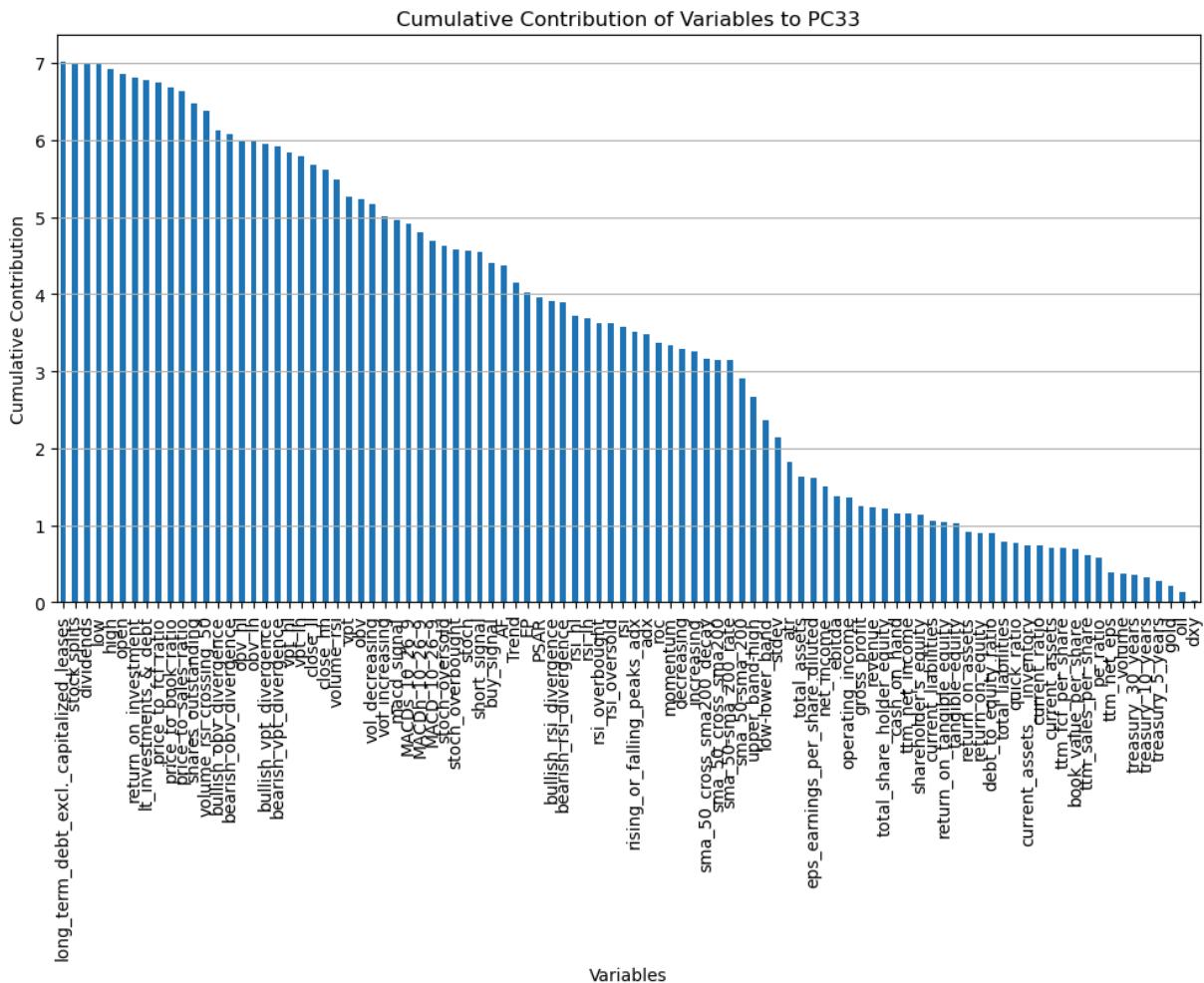




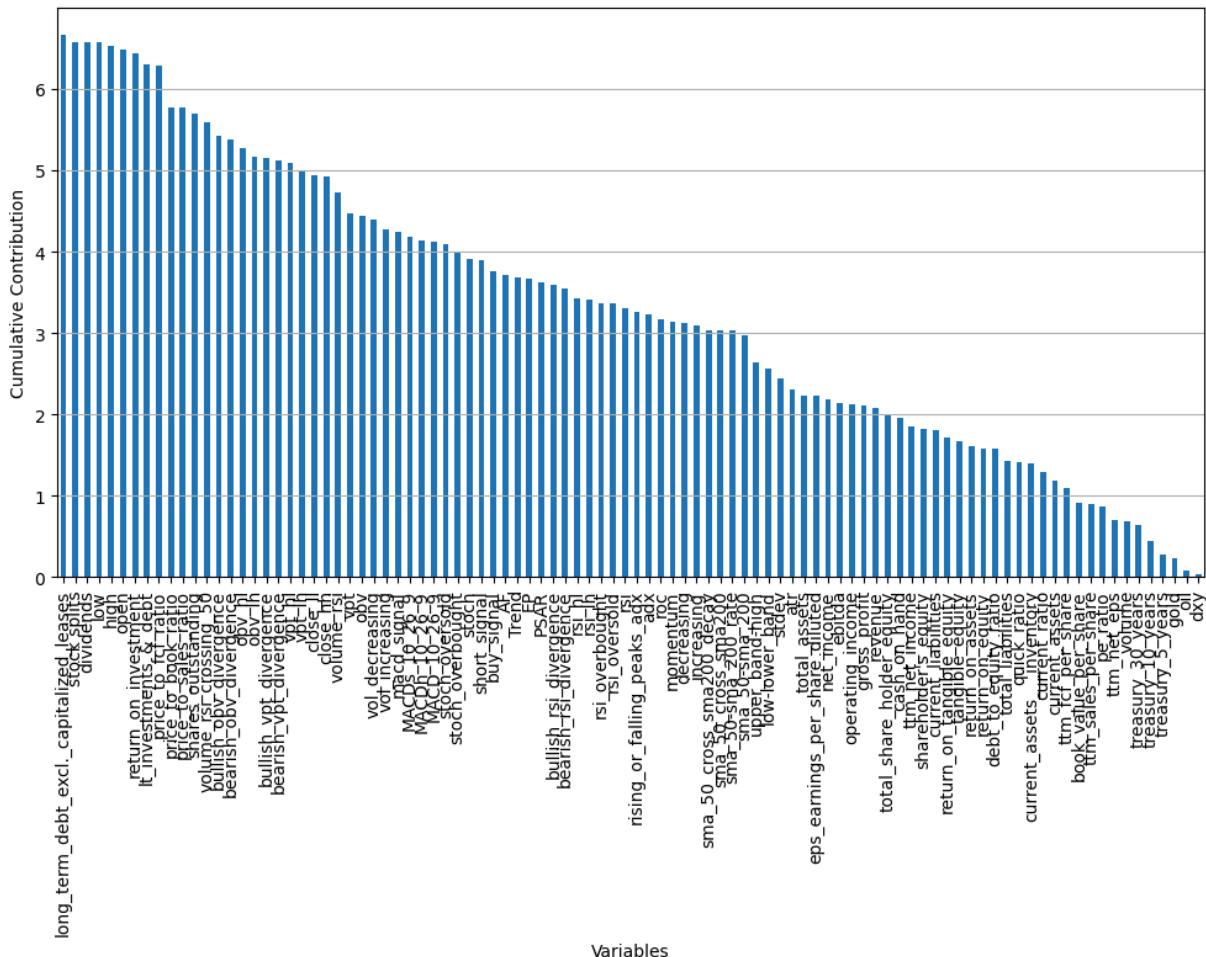


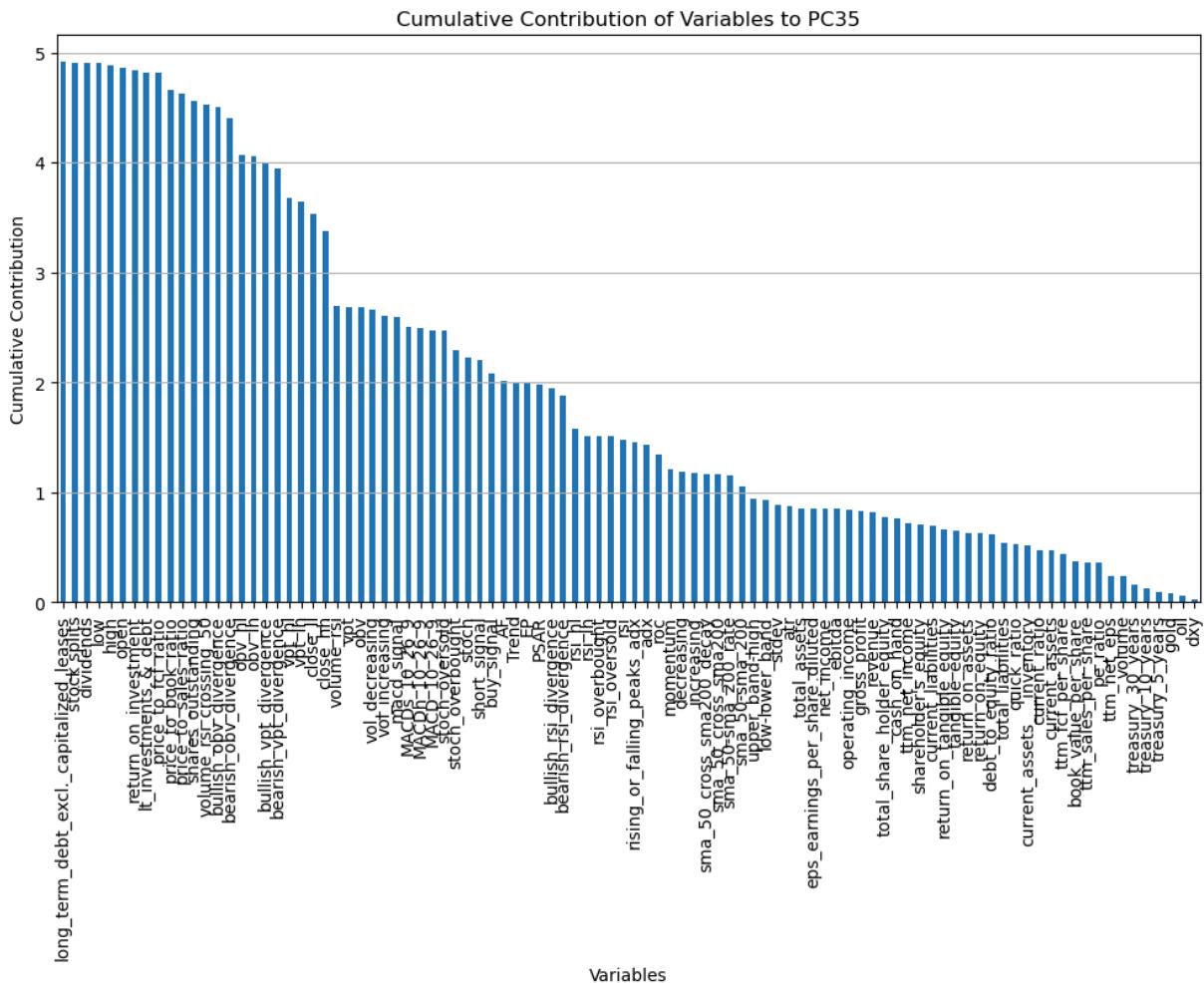




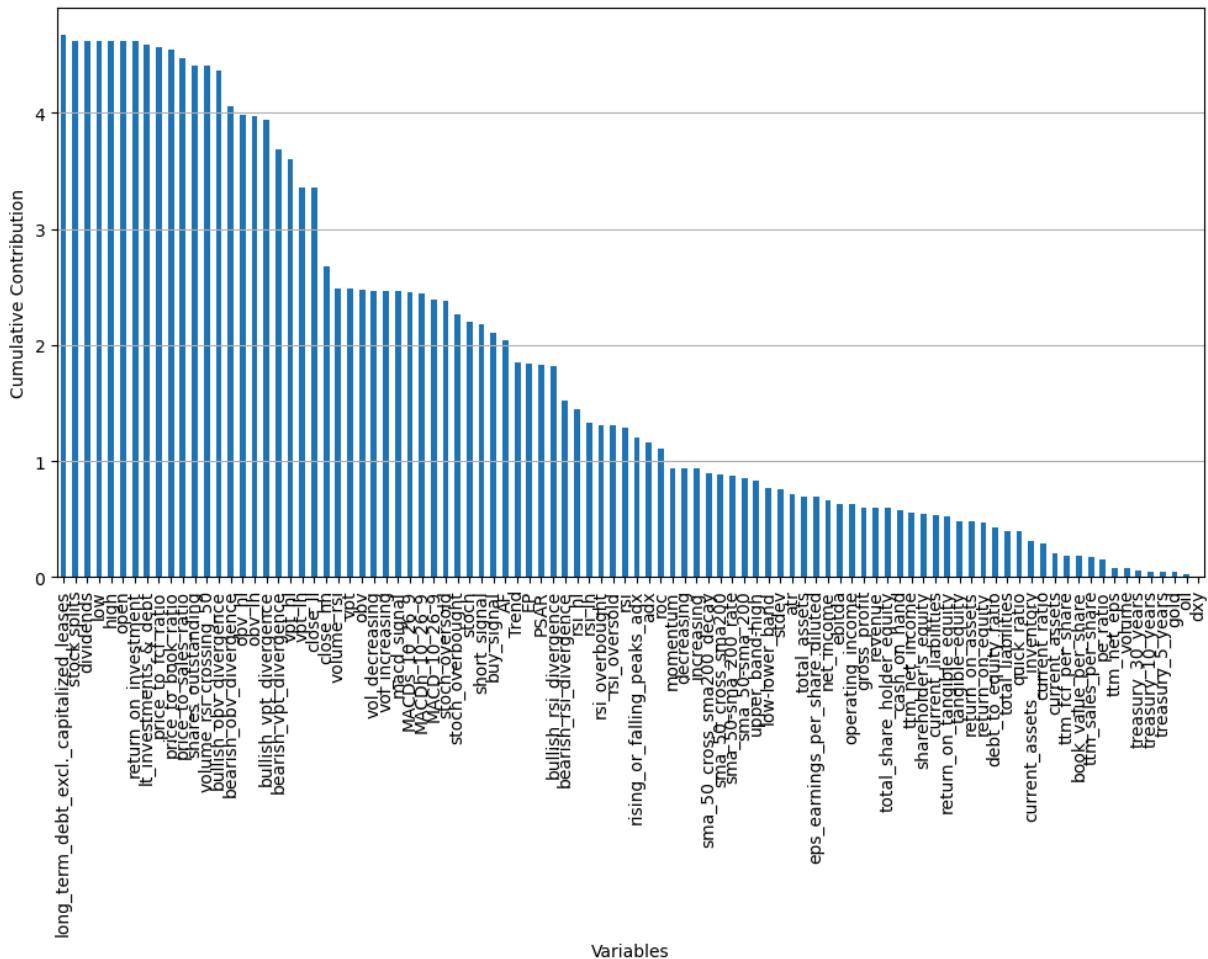


Cumulative Contribution of Variables to PC34

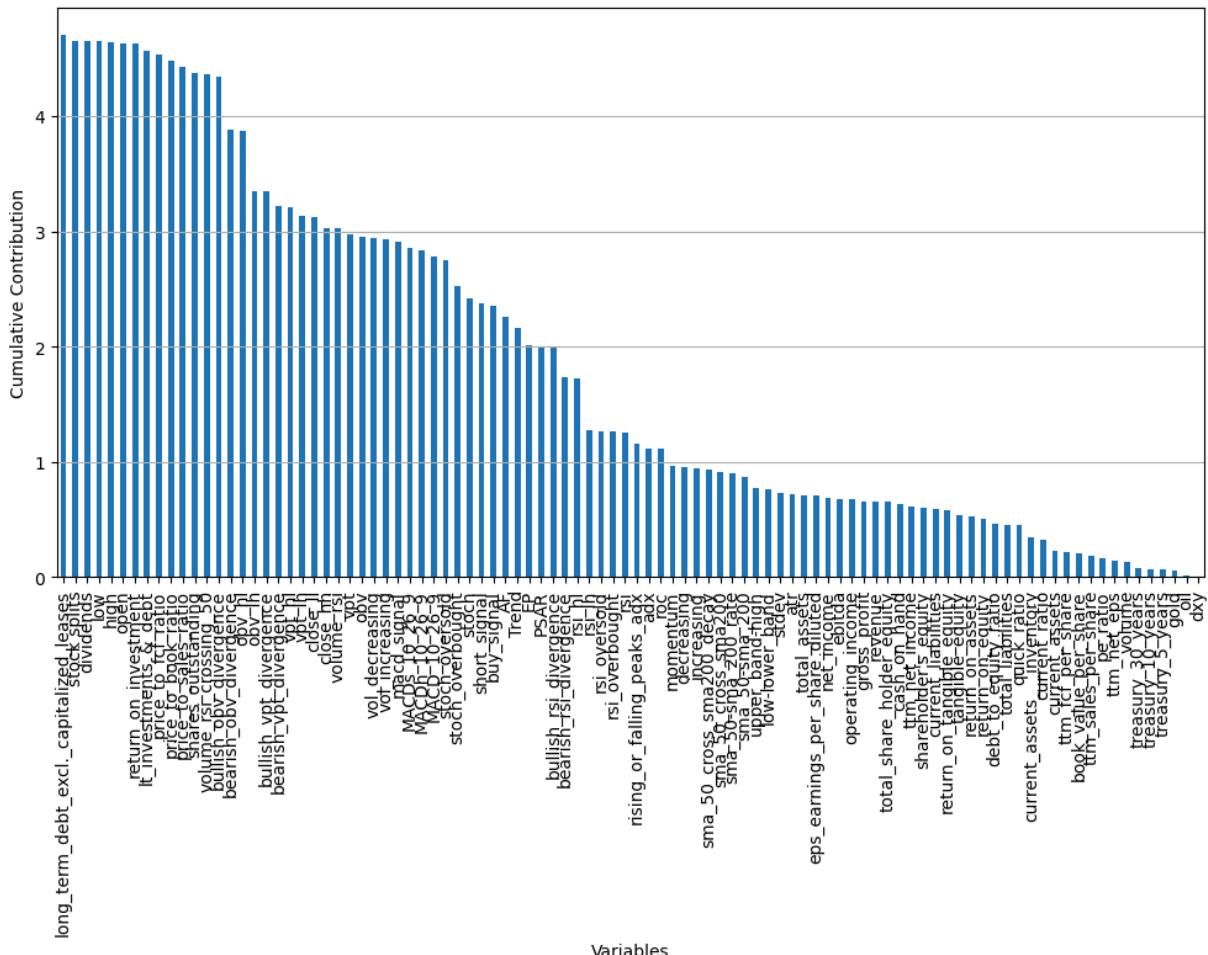


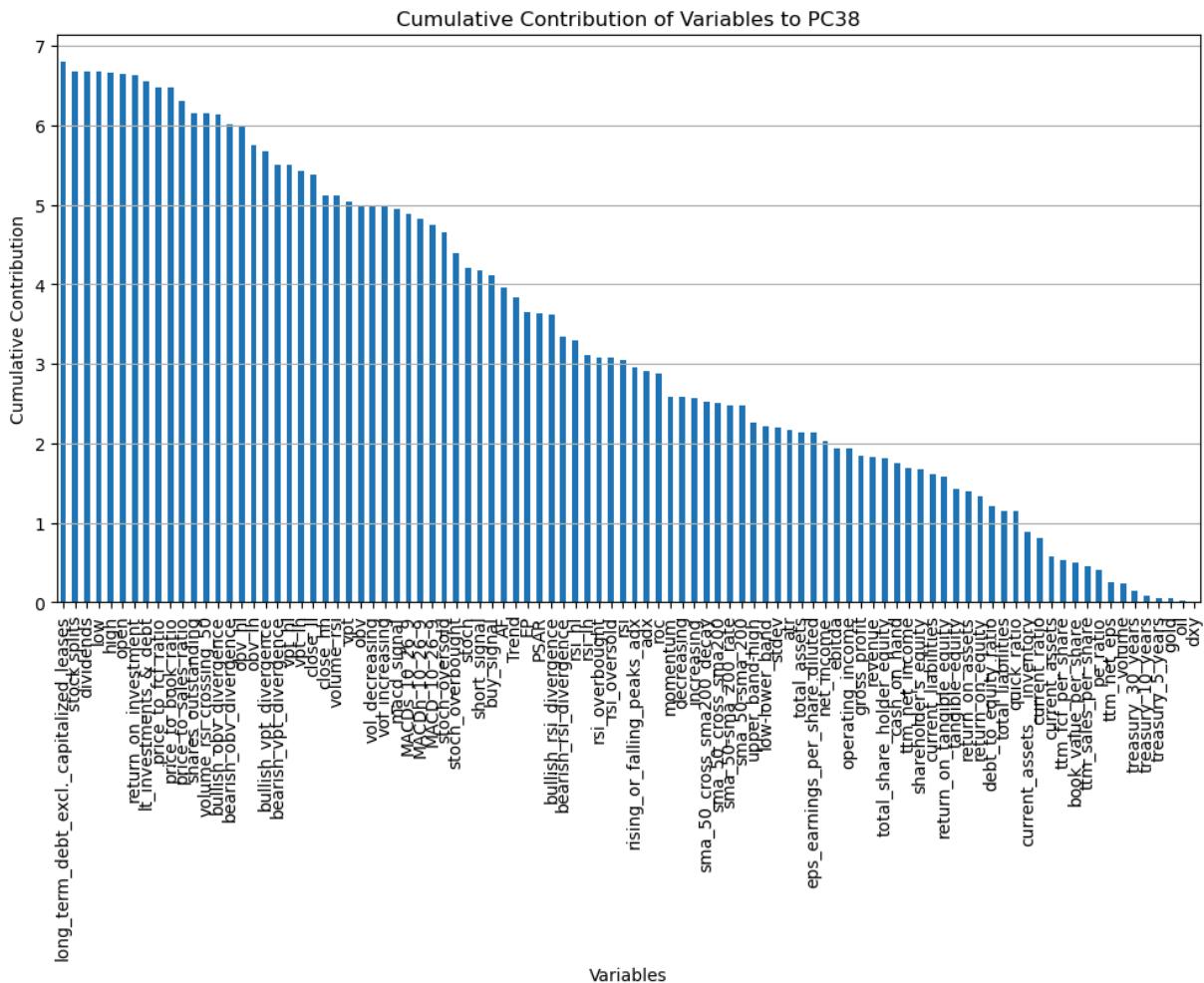


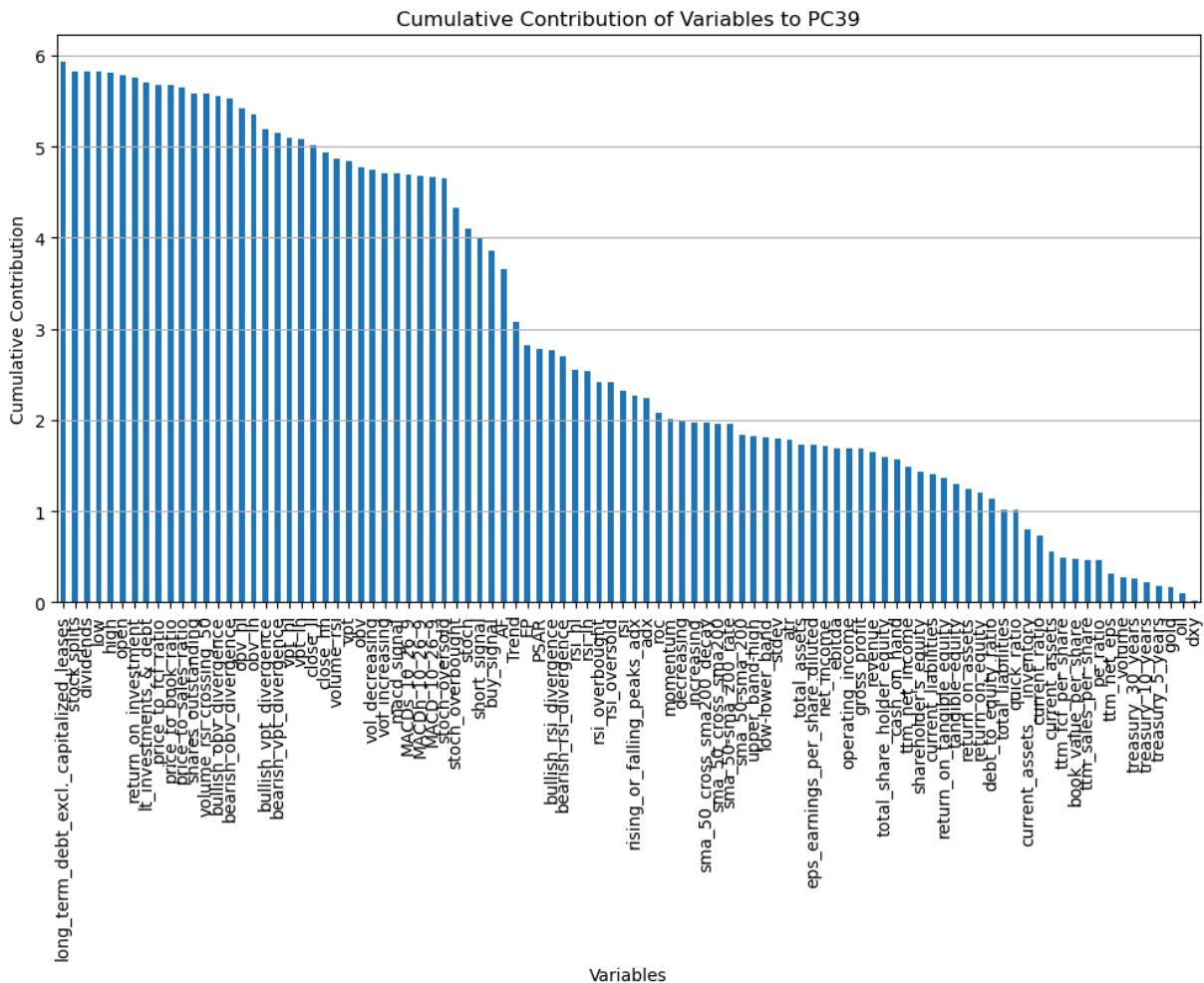
## Cumulative Contribution of Variables to PC36

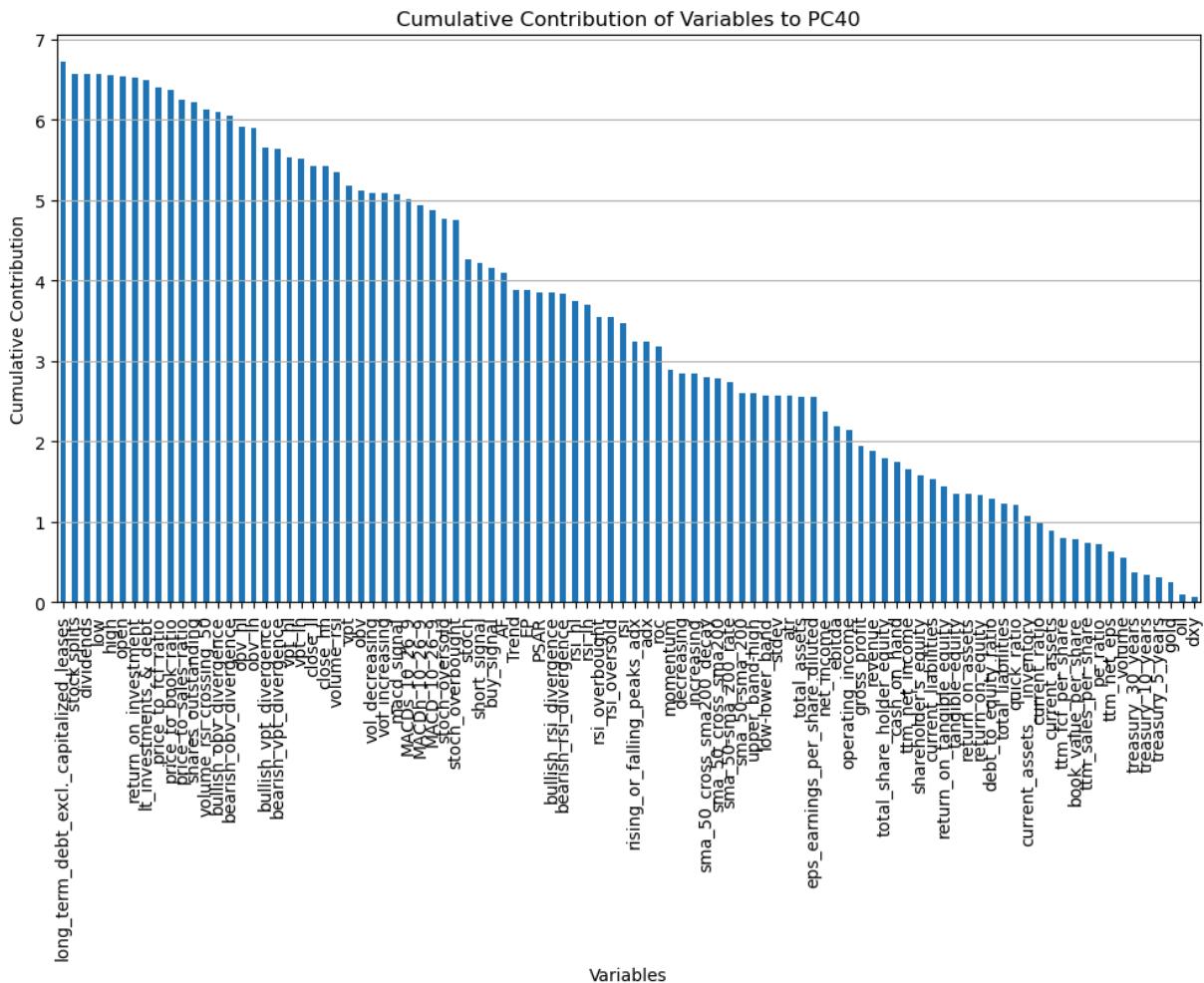


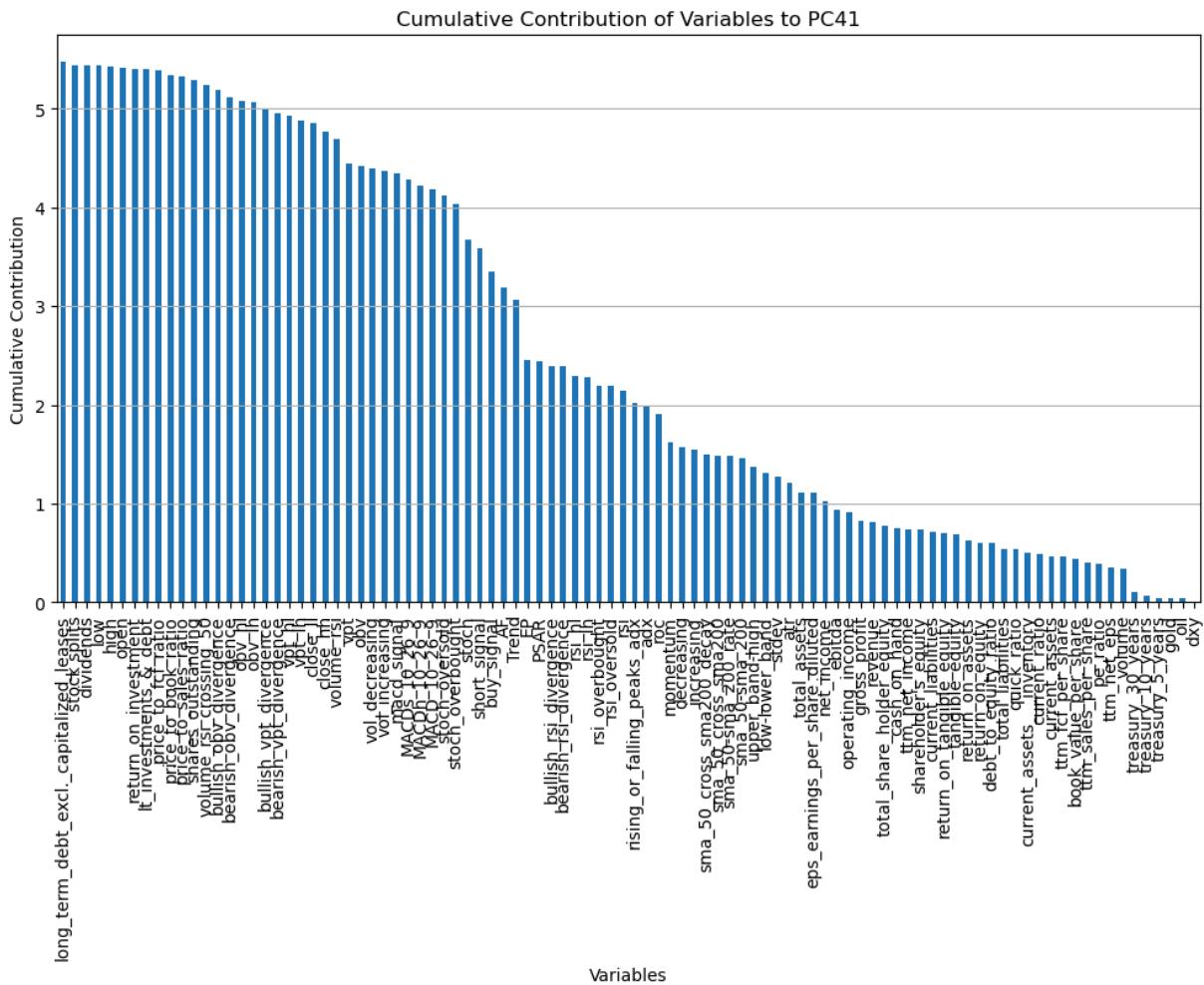
Cumulative Contribution of Variables to PC37



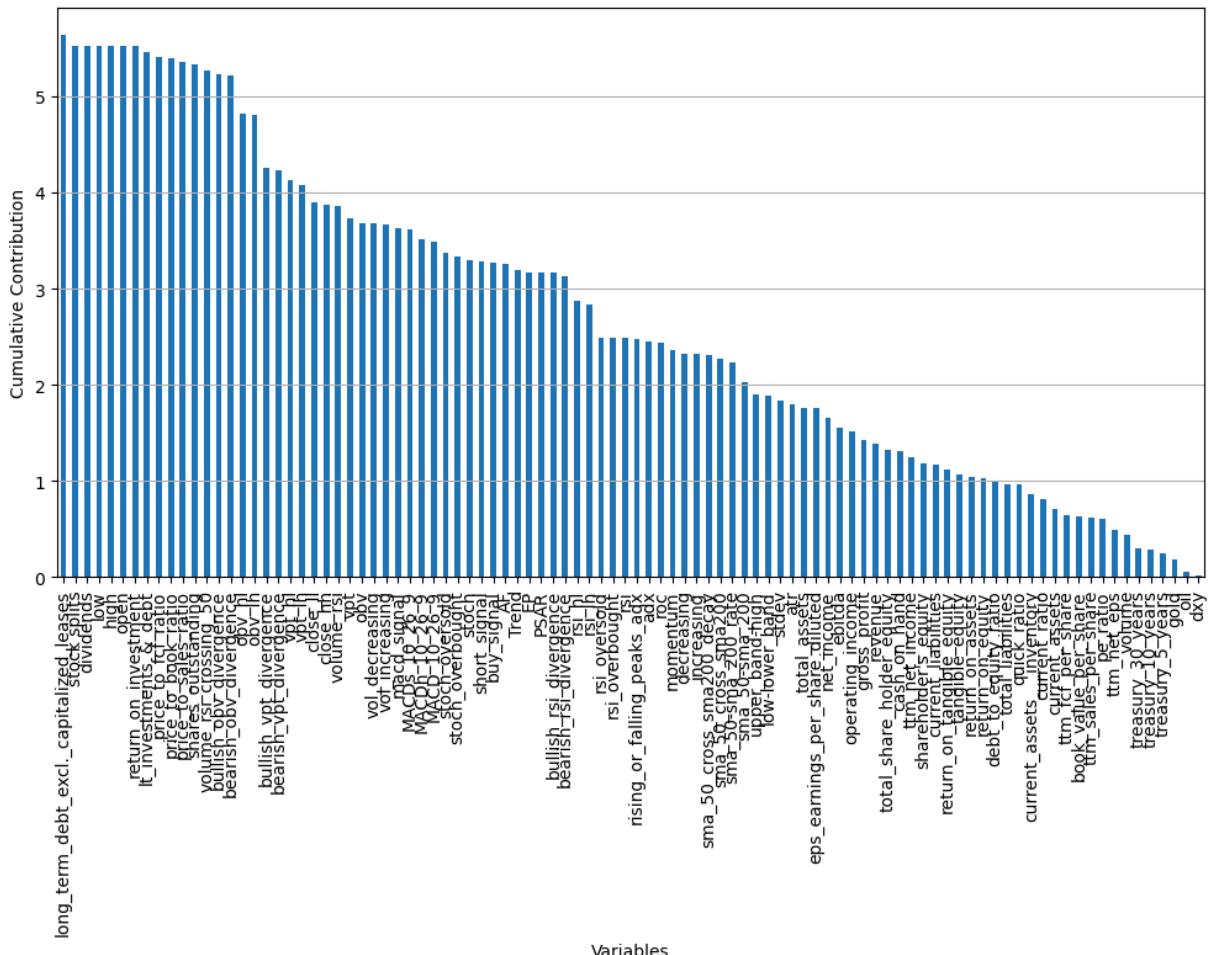




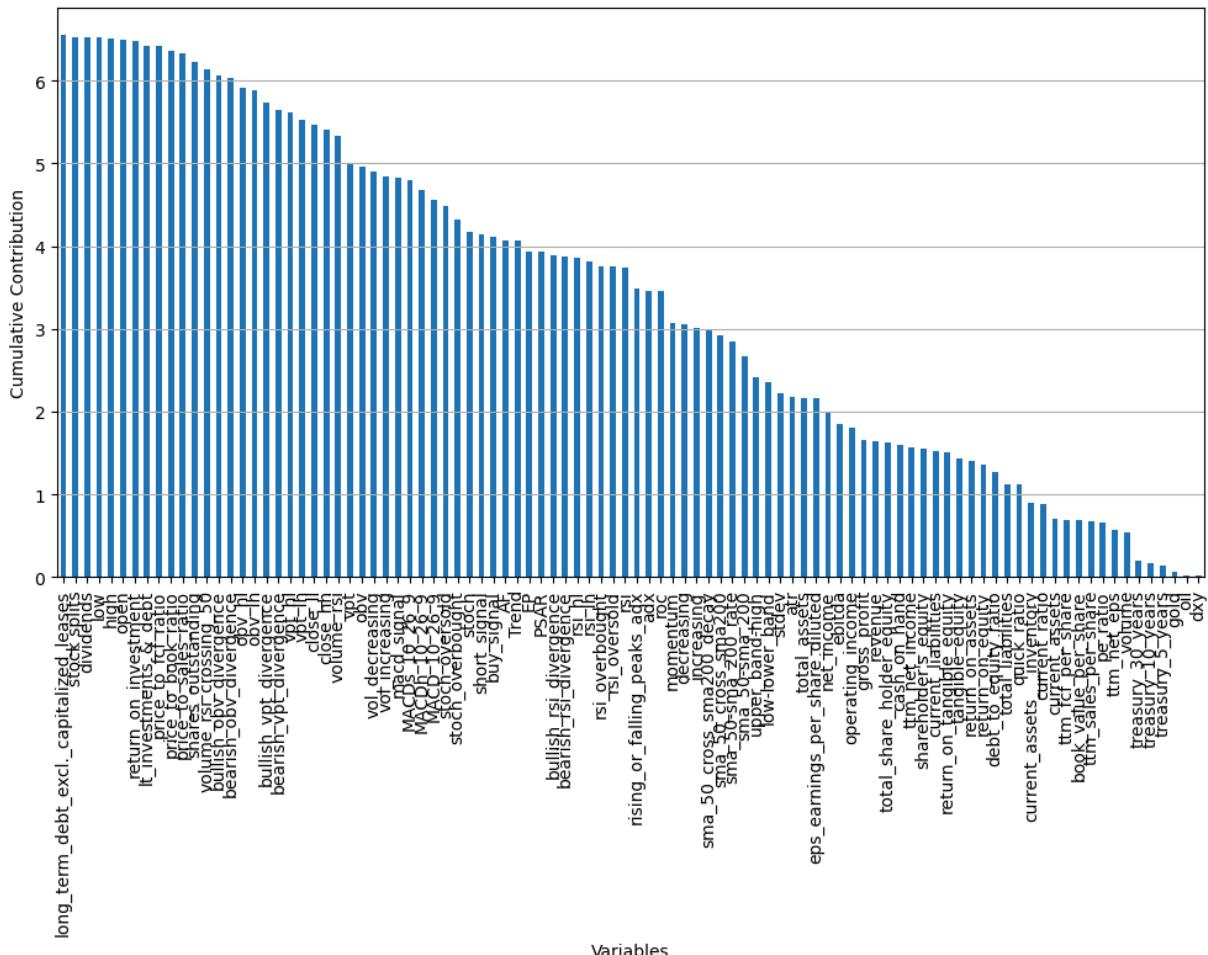


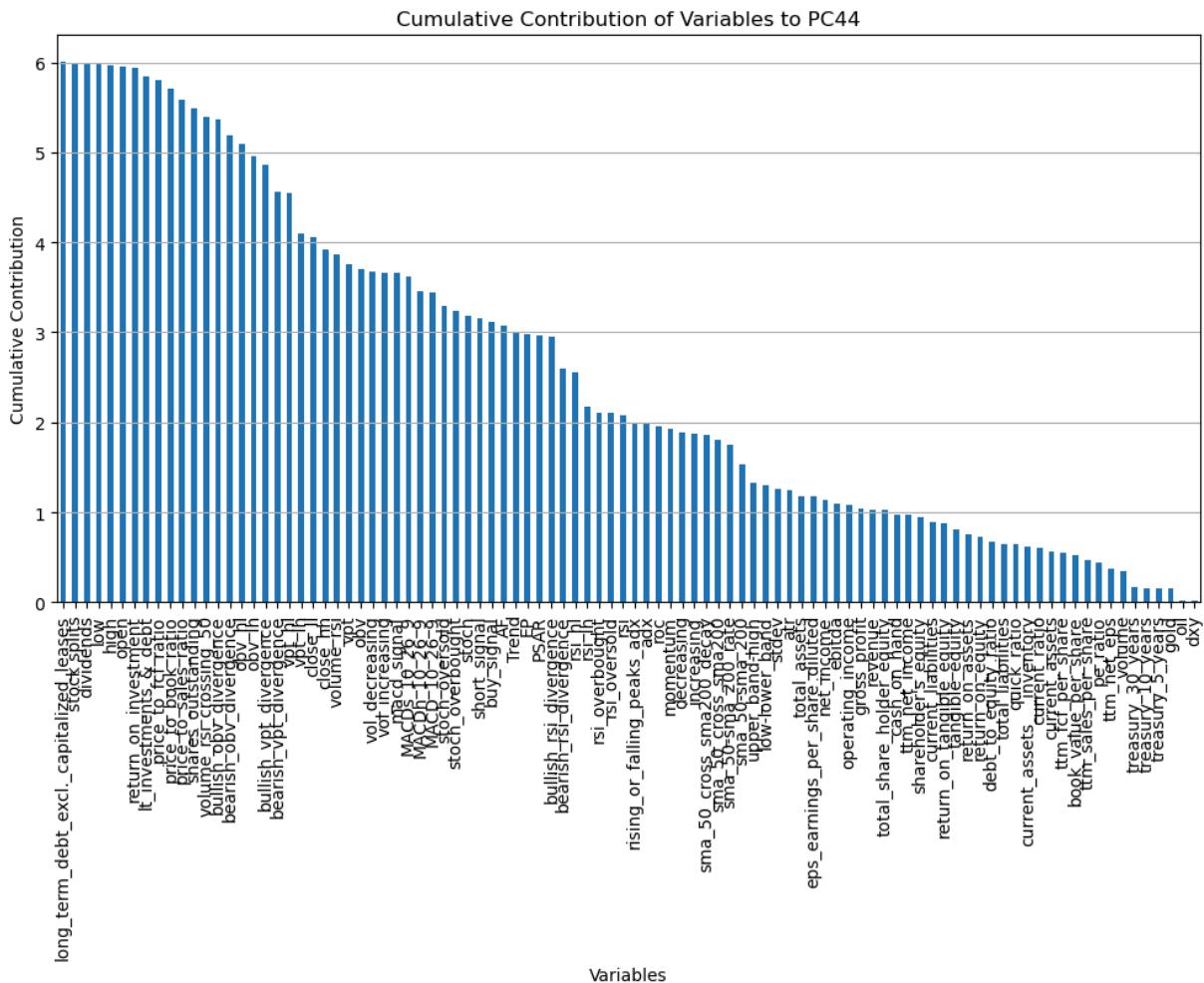


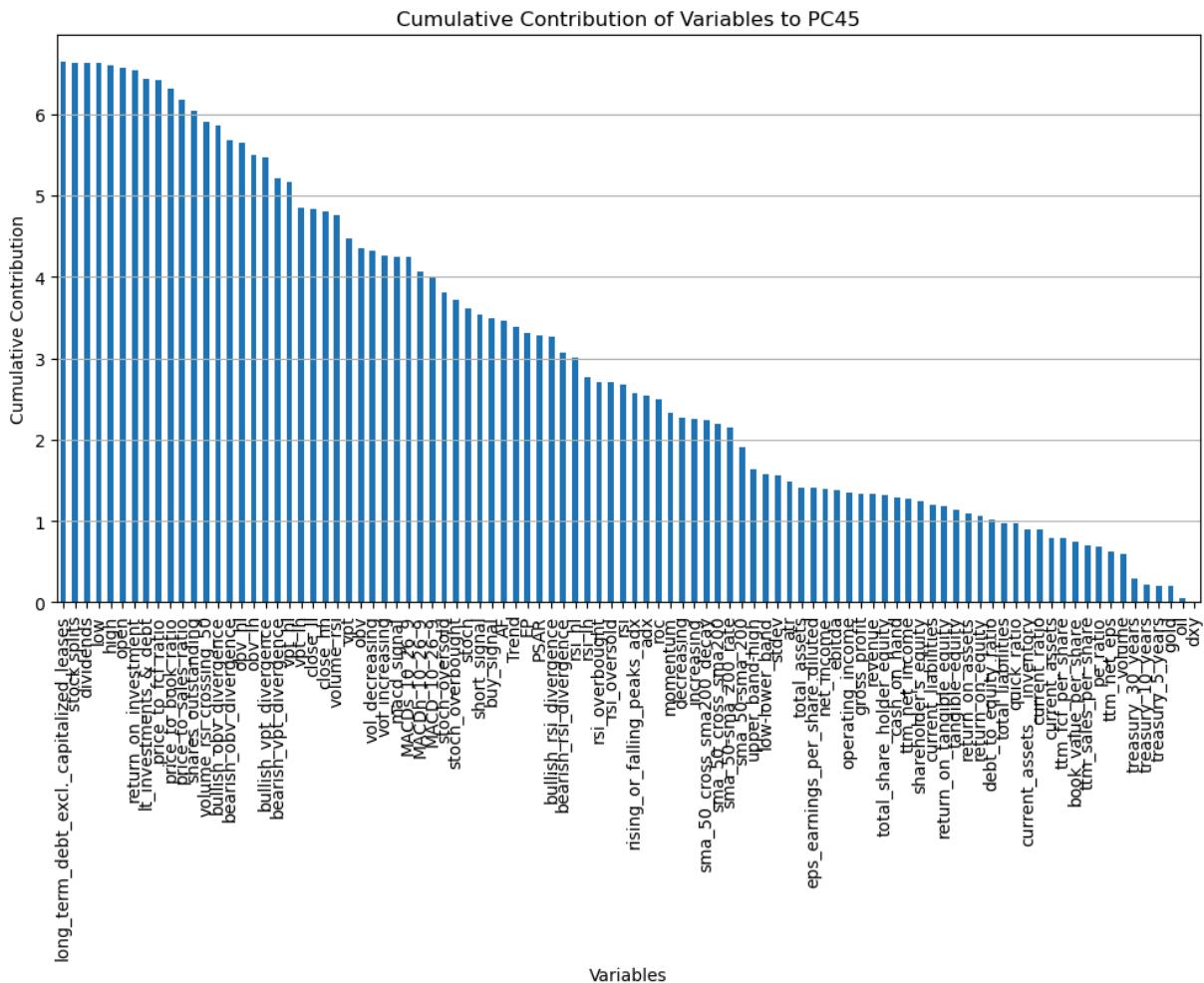
## Cumulative Contribution of Variables to PC42

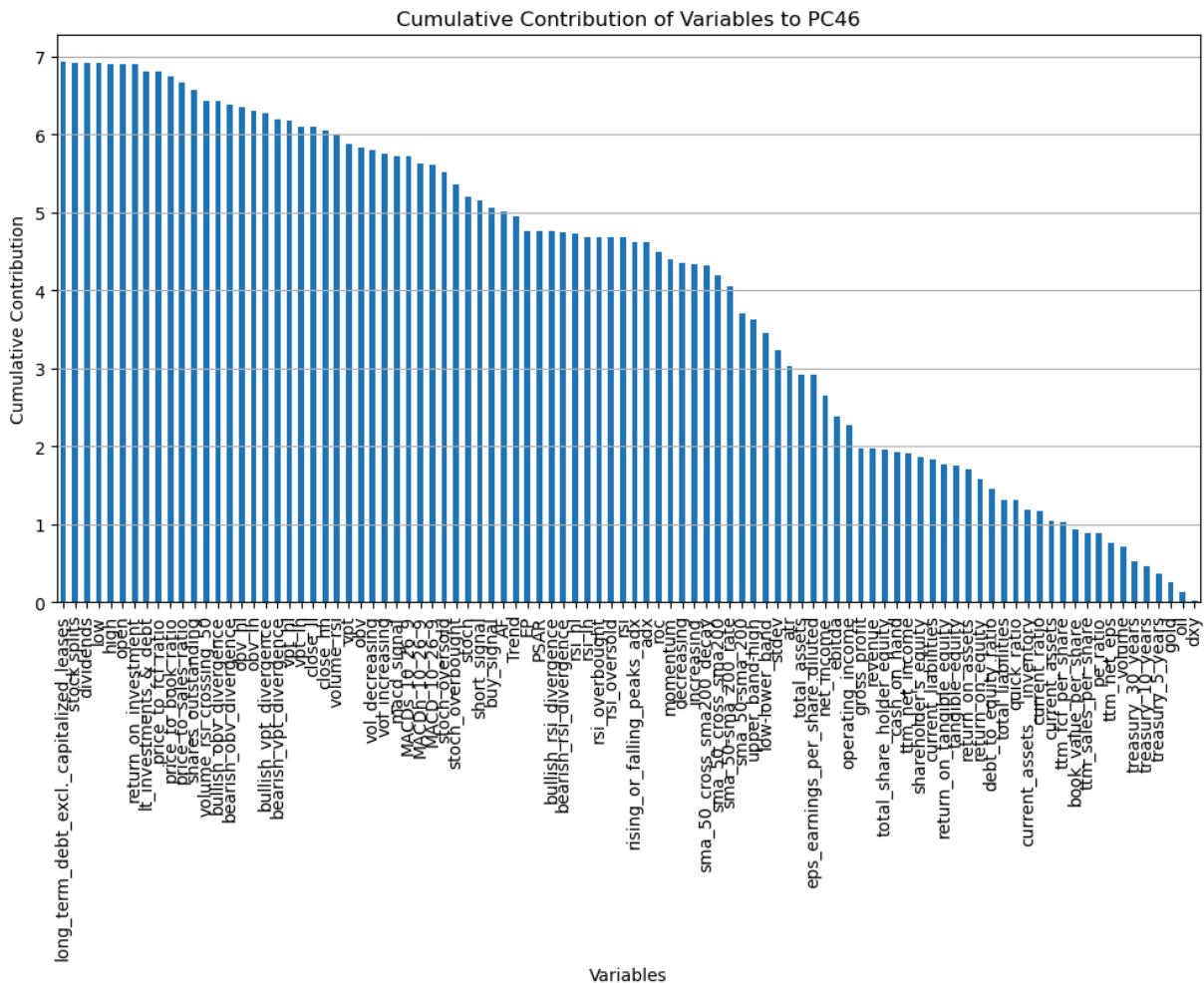


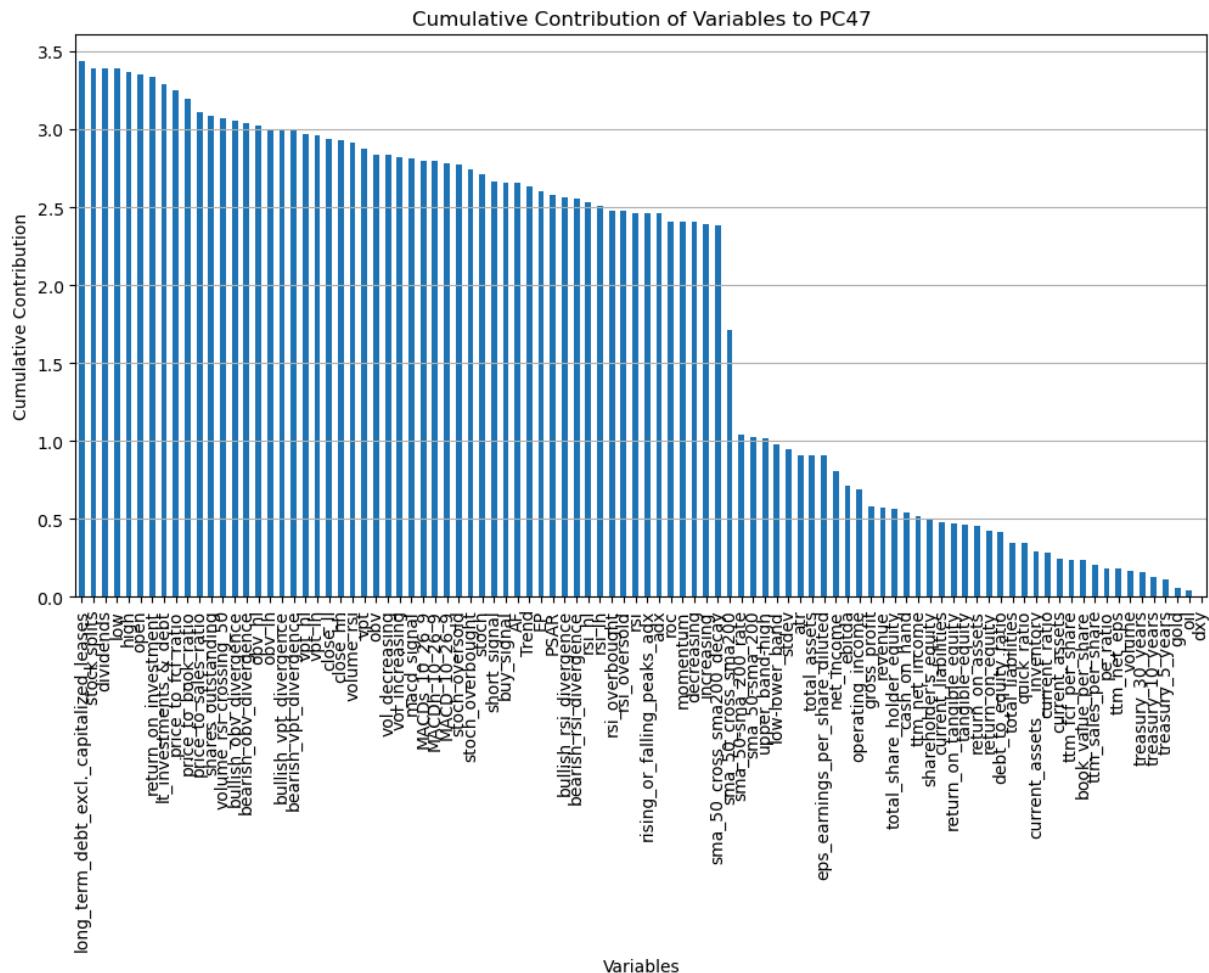
## Cumulative Contribution of Variables to PC43

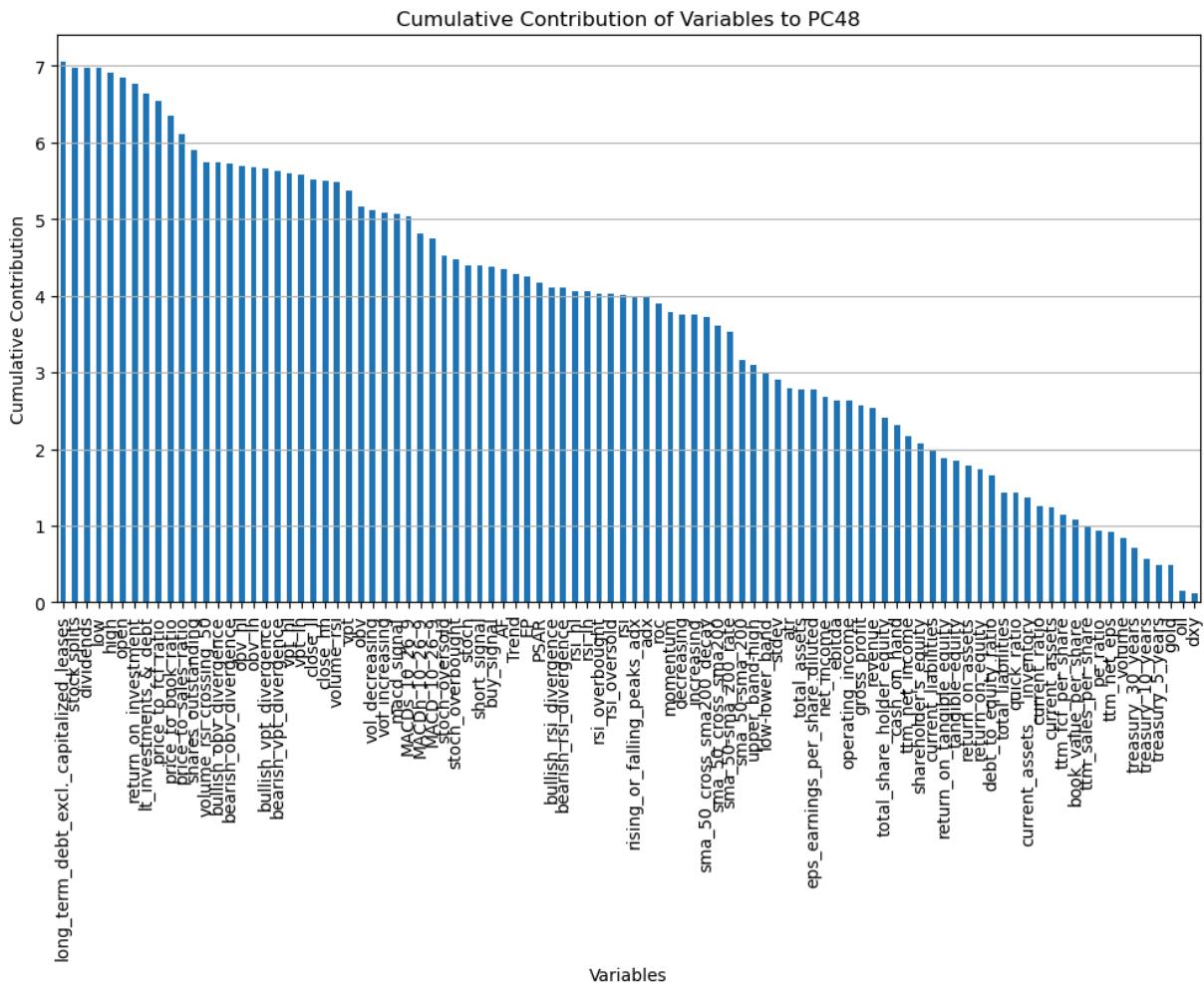




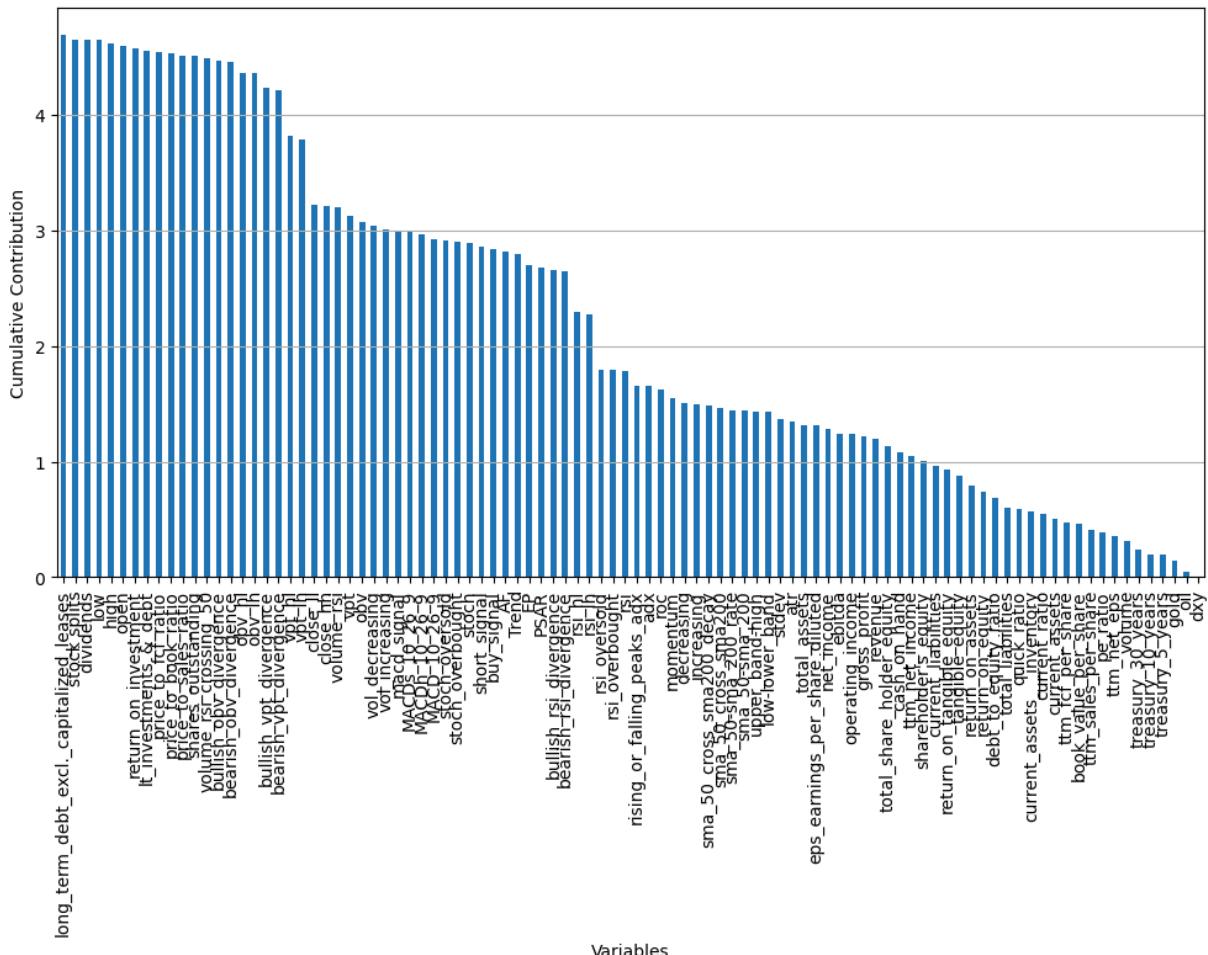


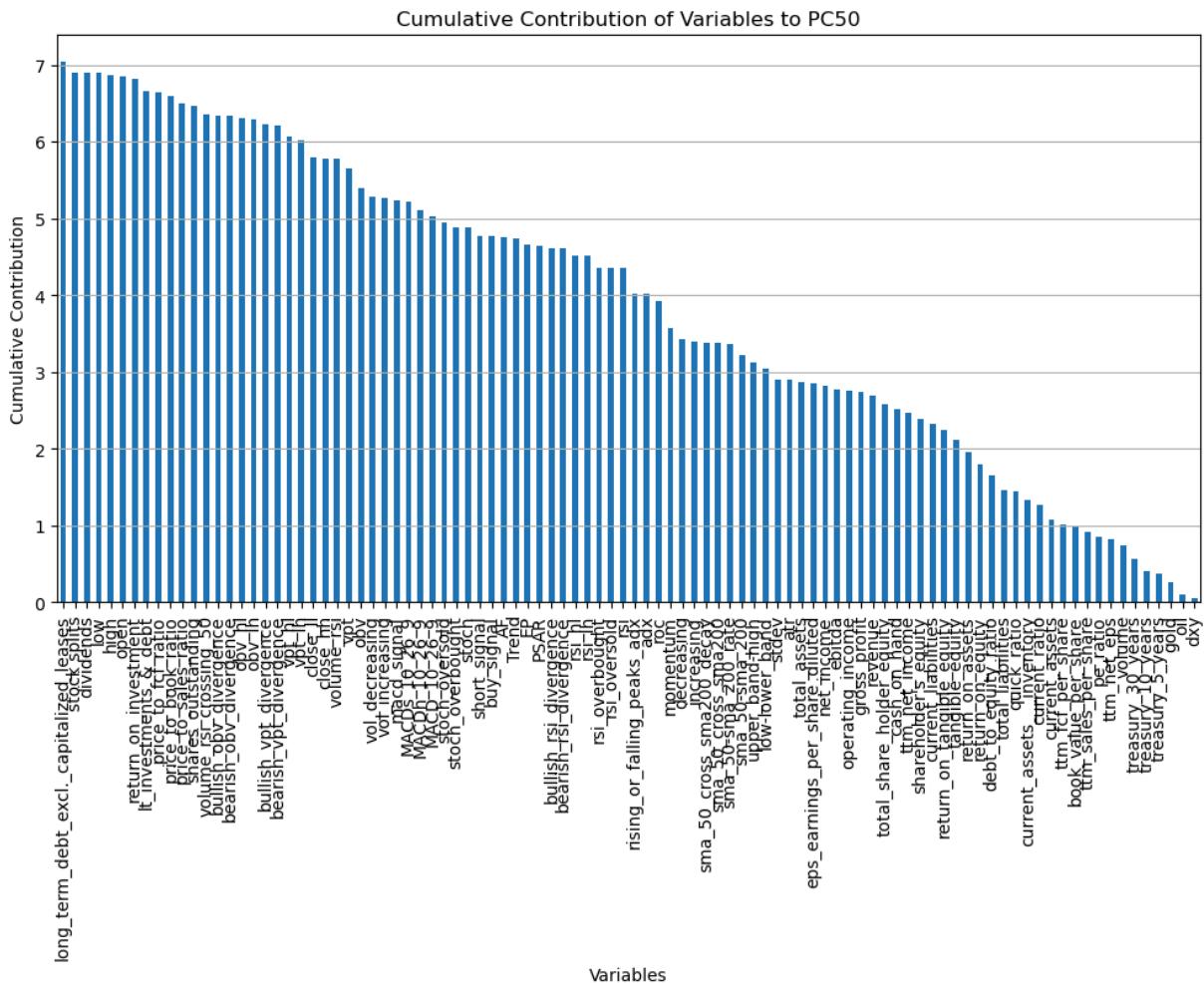




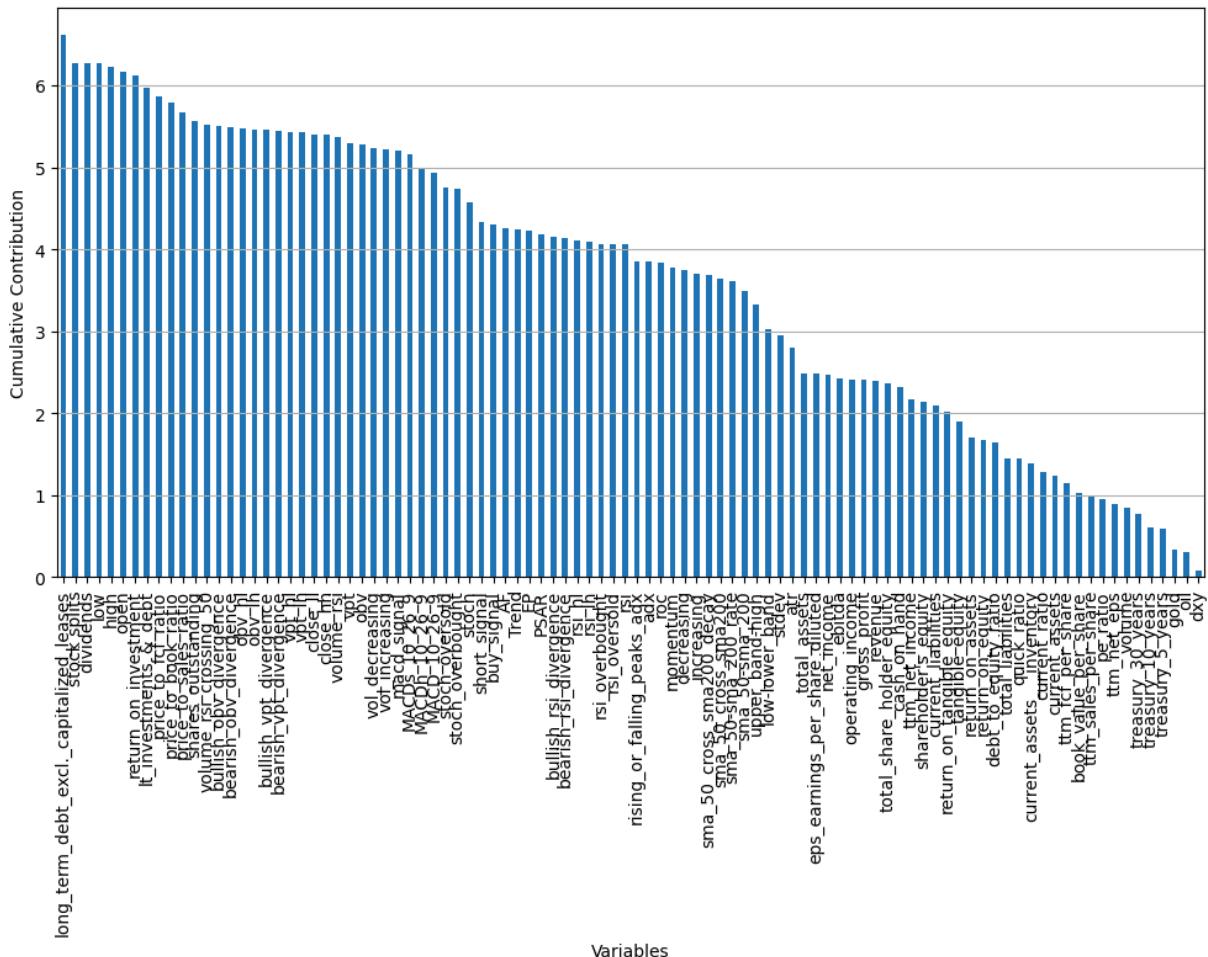


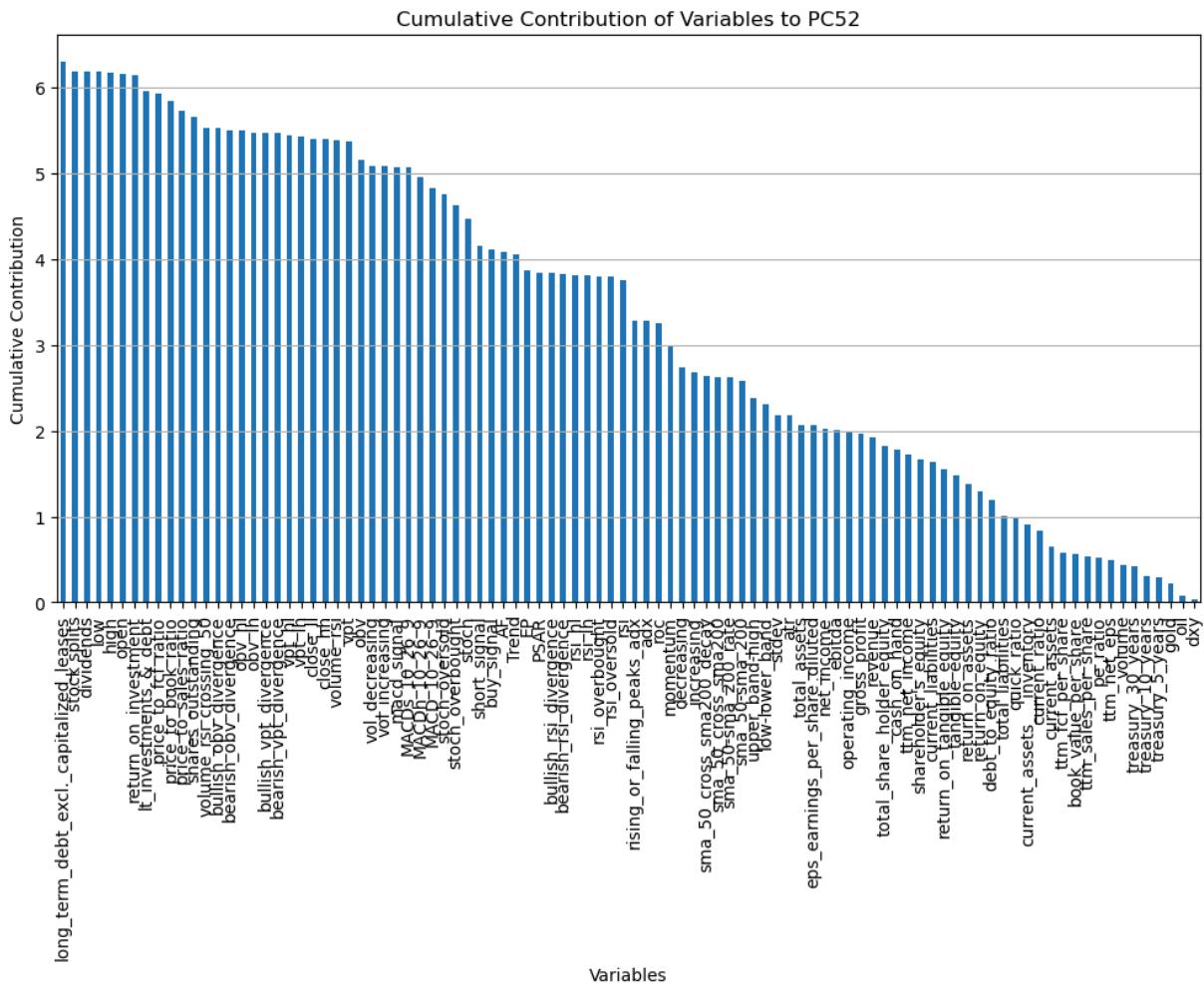
## Cumulative Contribution of Variables to PC49



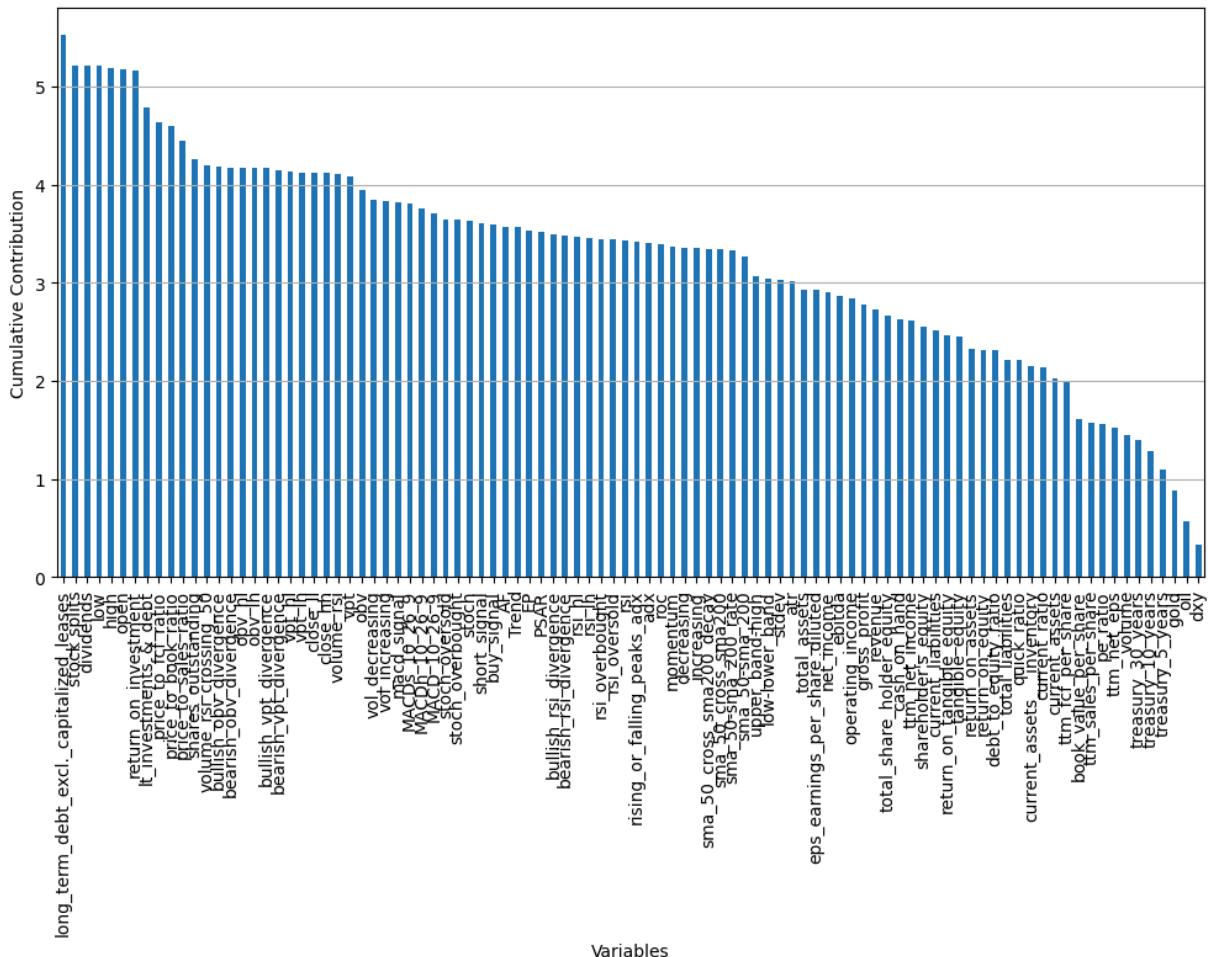


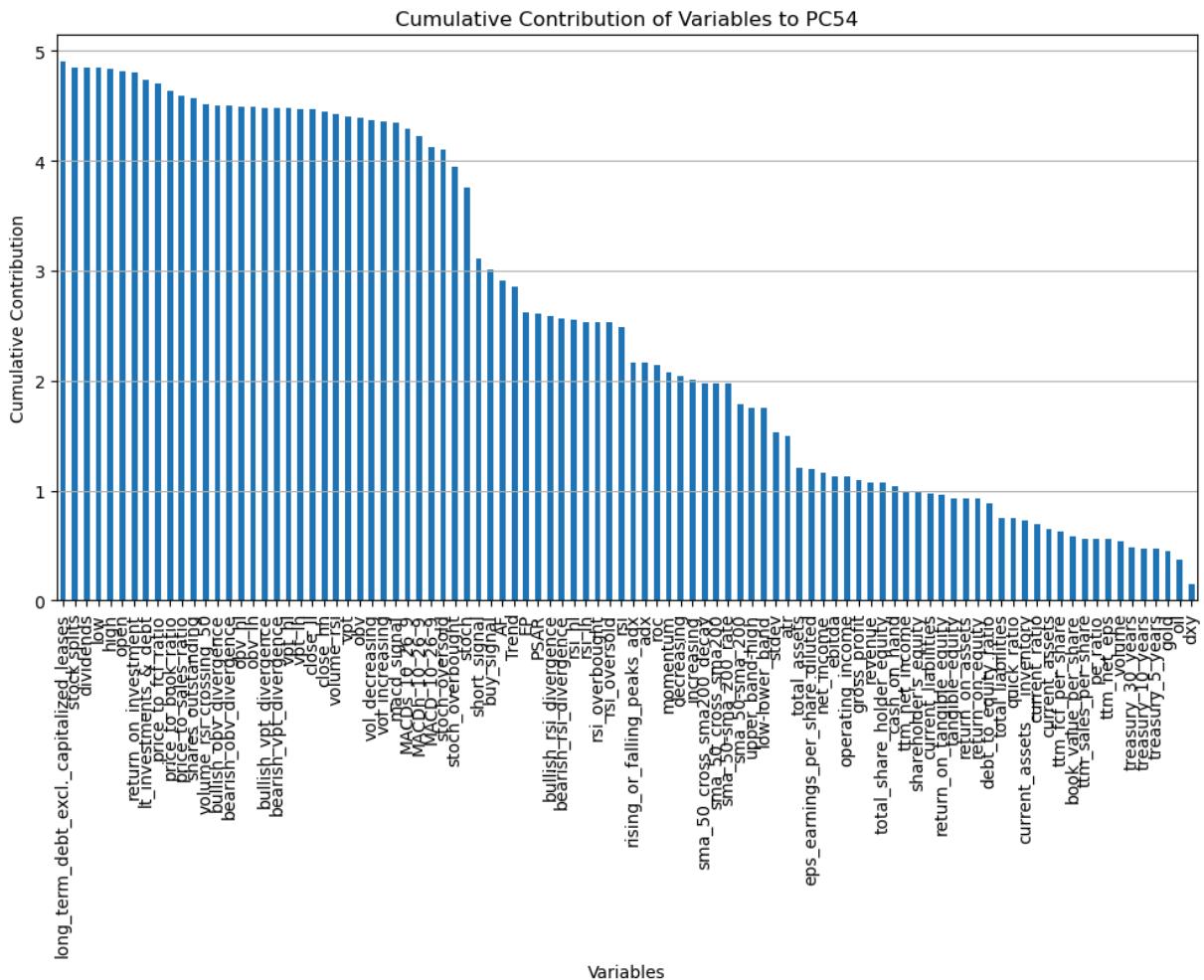
## Cumulative Contribution of Variables to PC51

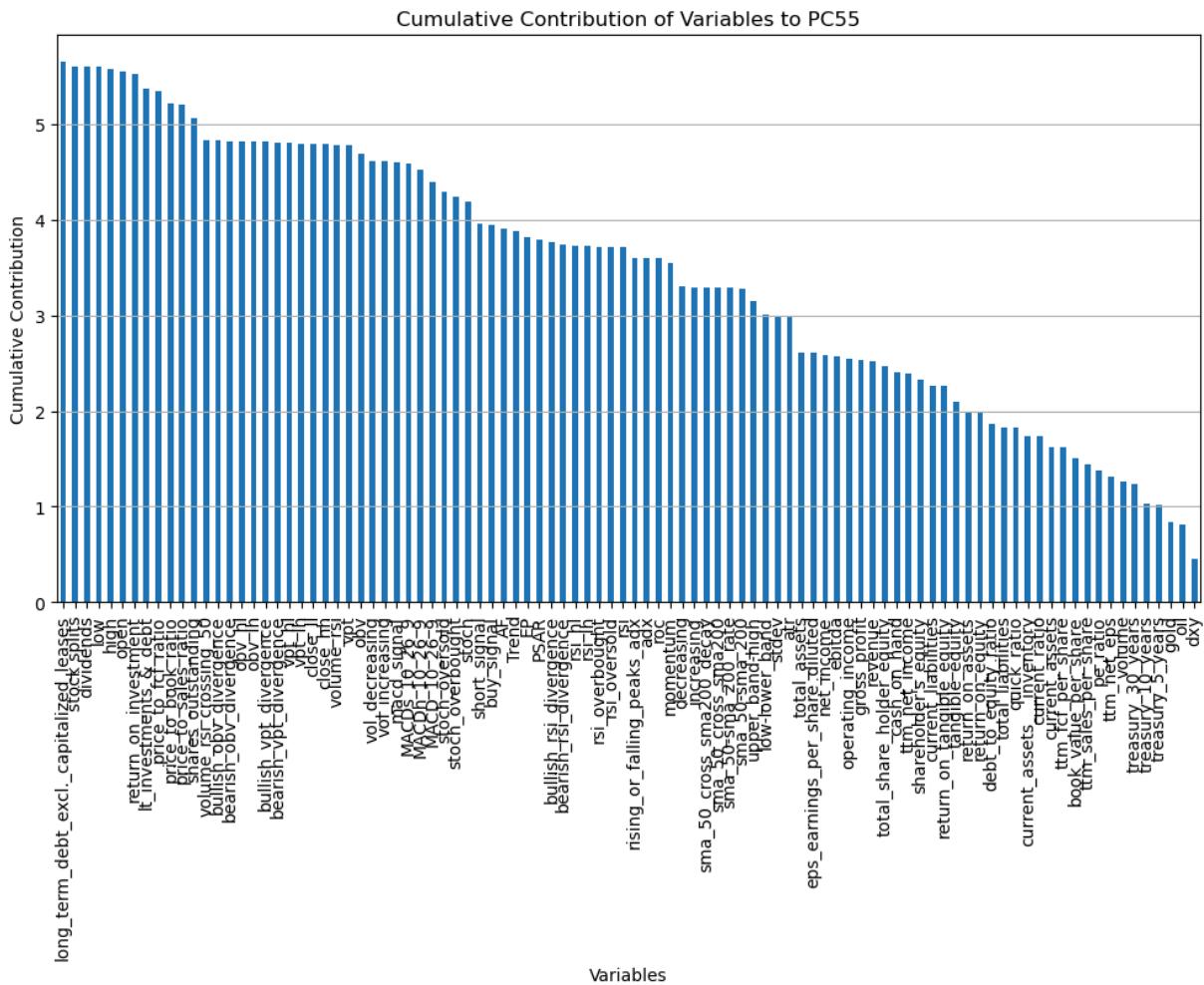




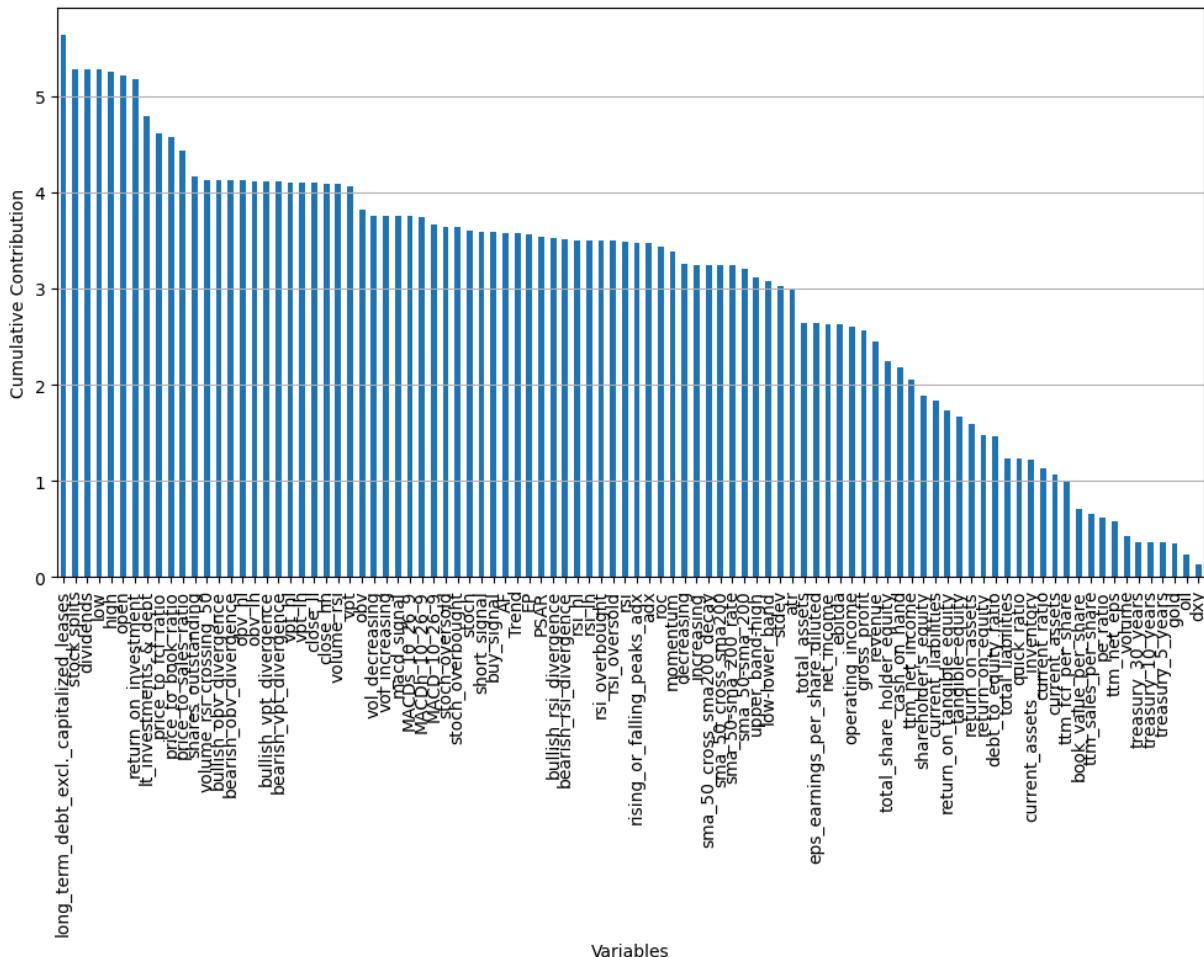
## Cumulative Contribution of Variables to PC53

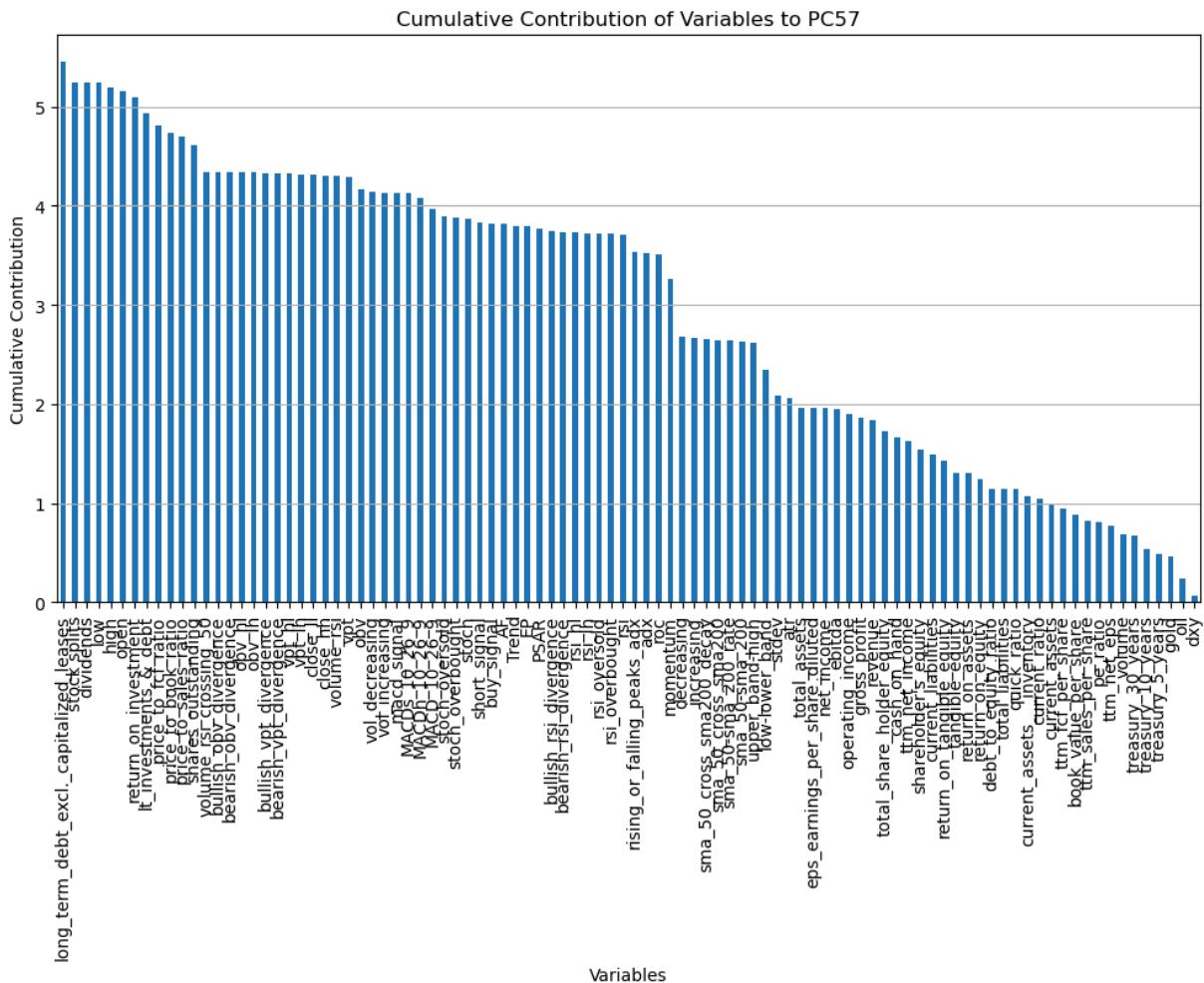




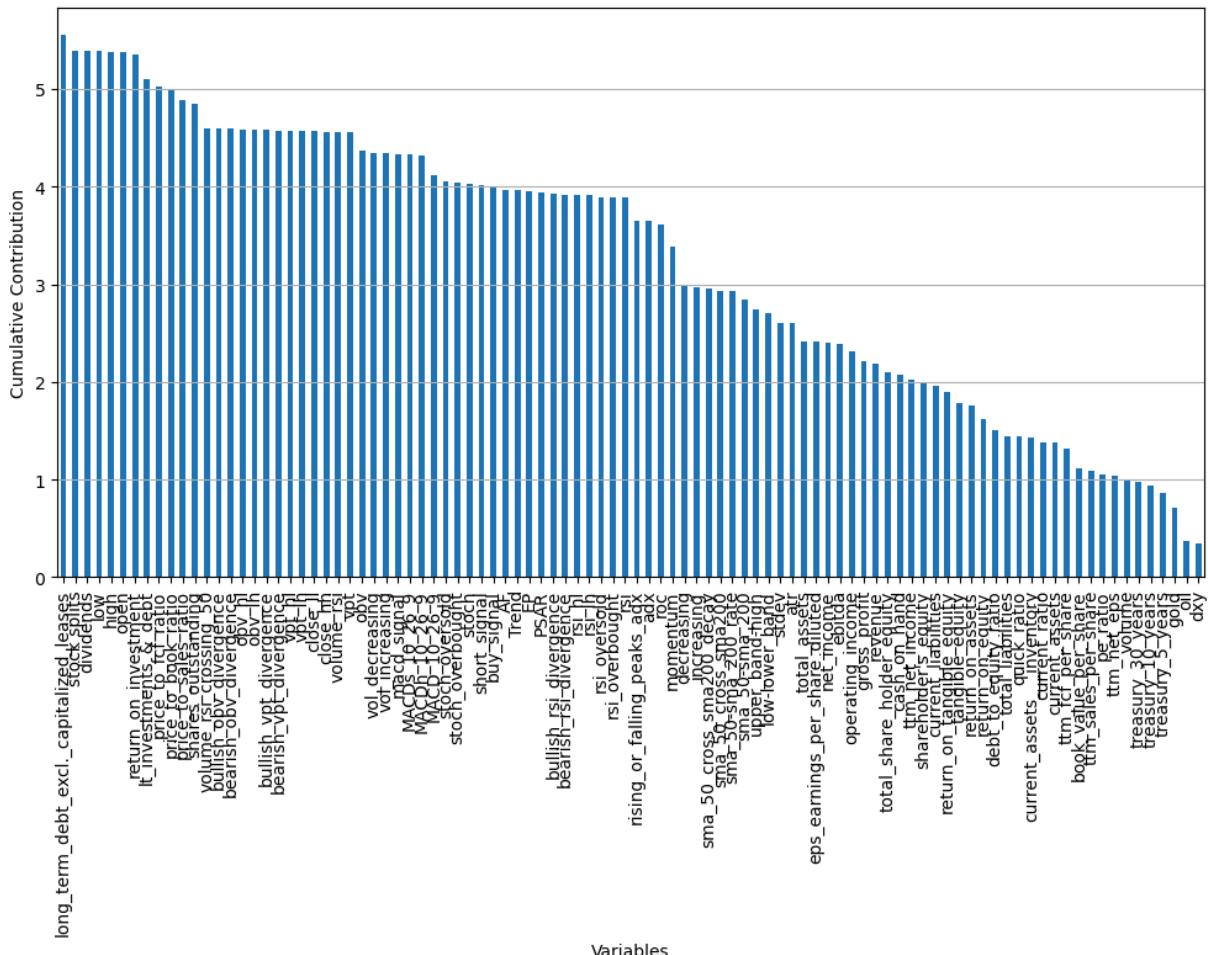


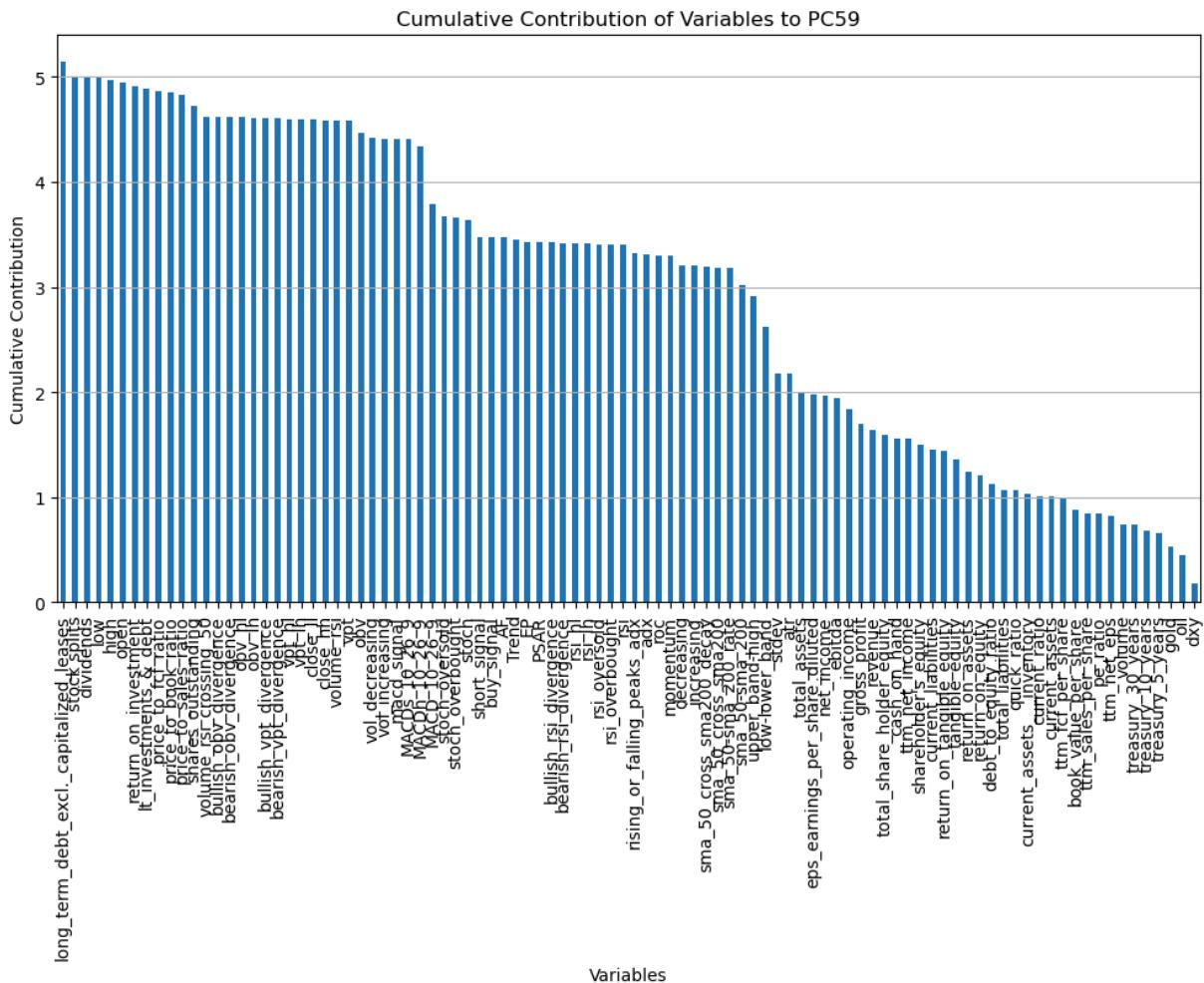
## Cumulative Contribution of Variables to PC56

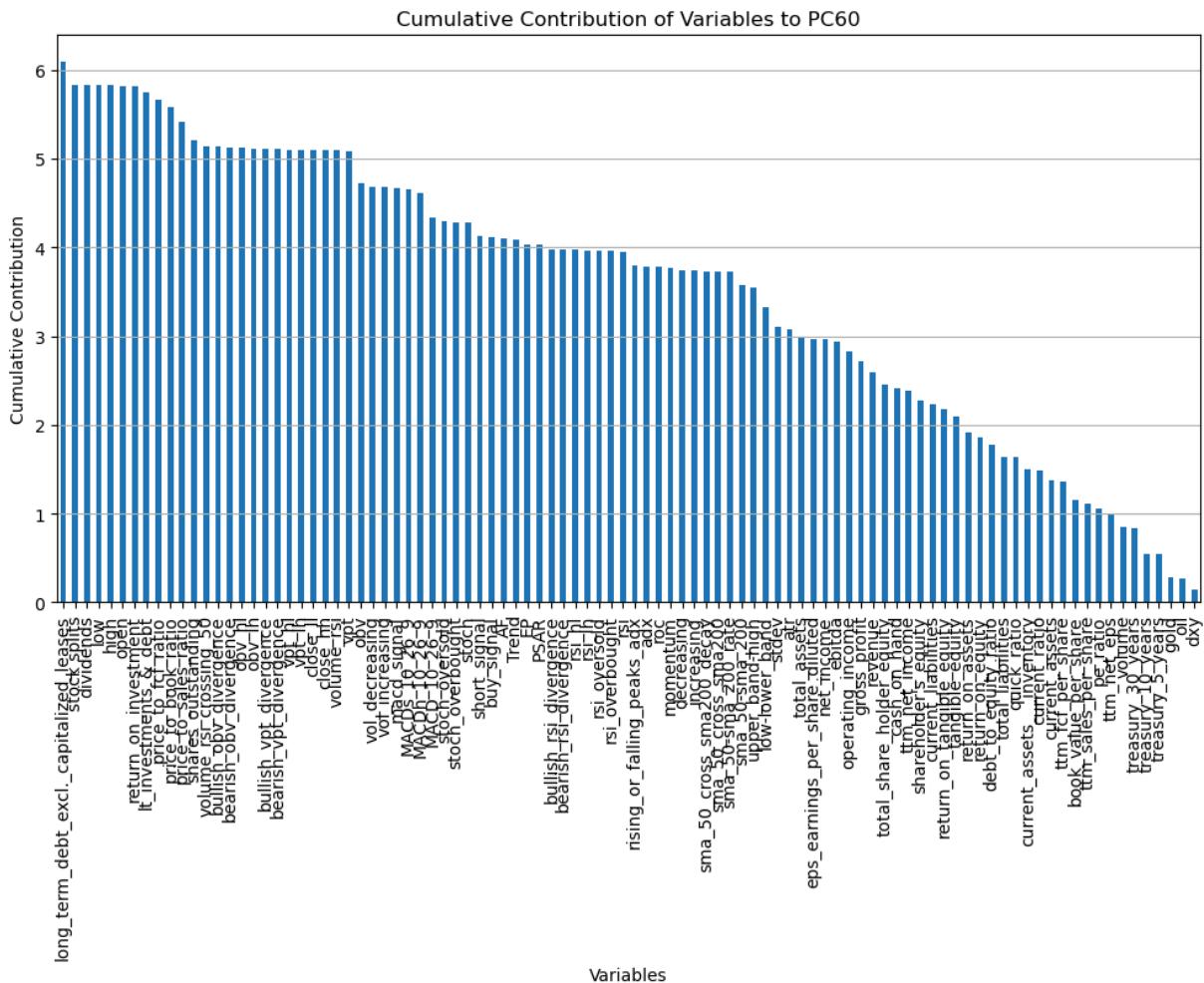


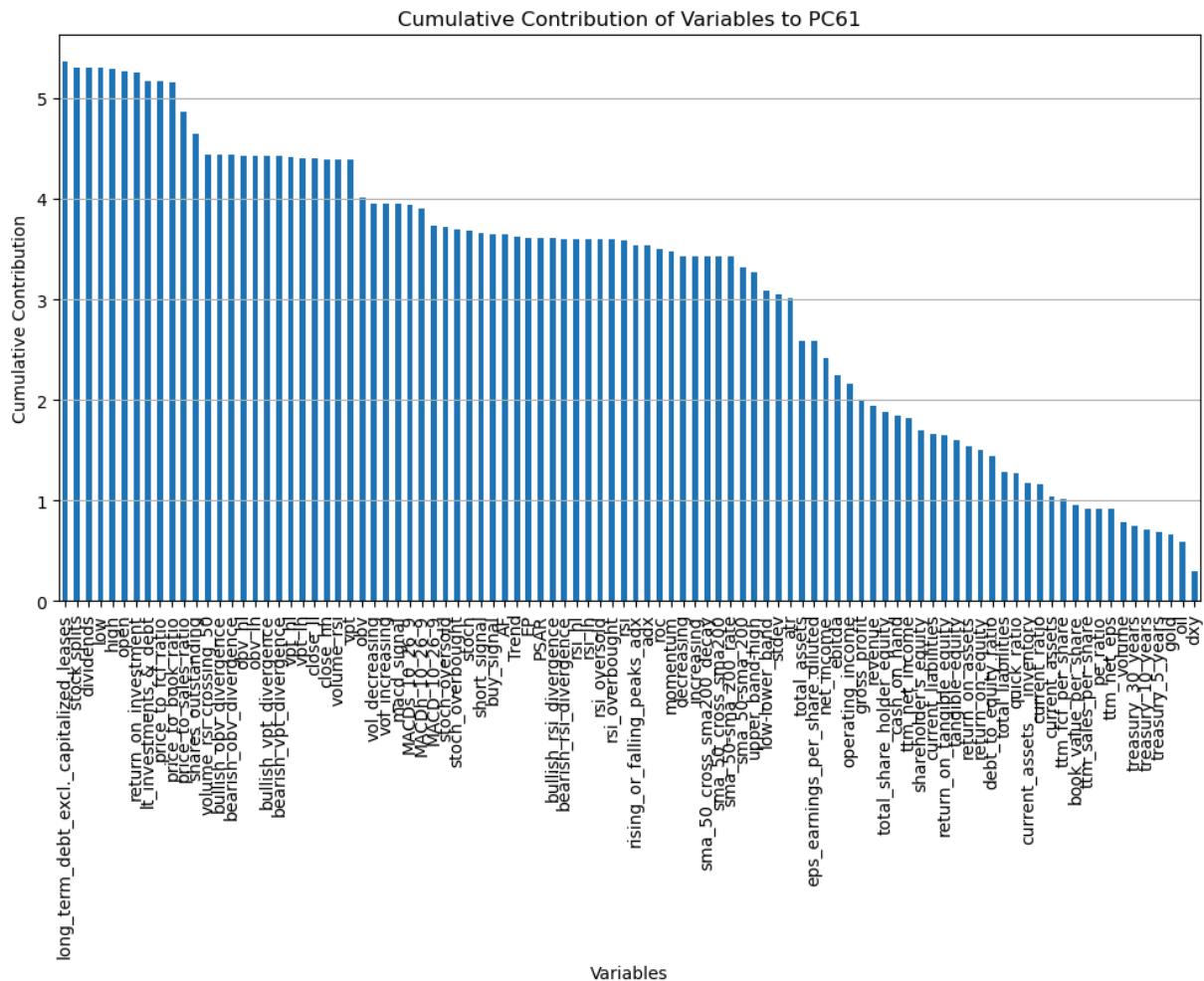


## Cumulative Contribution of Variables to PC58

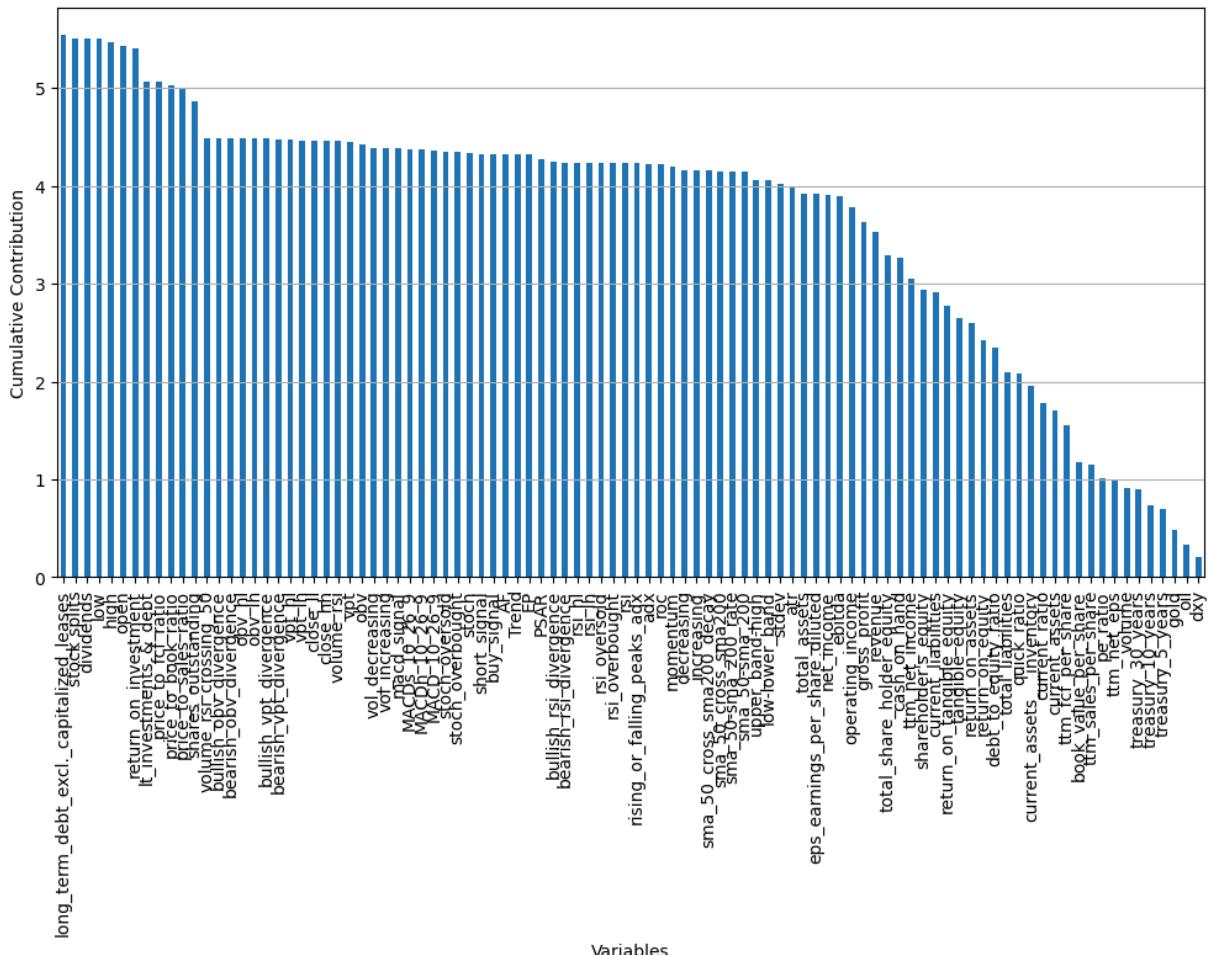




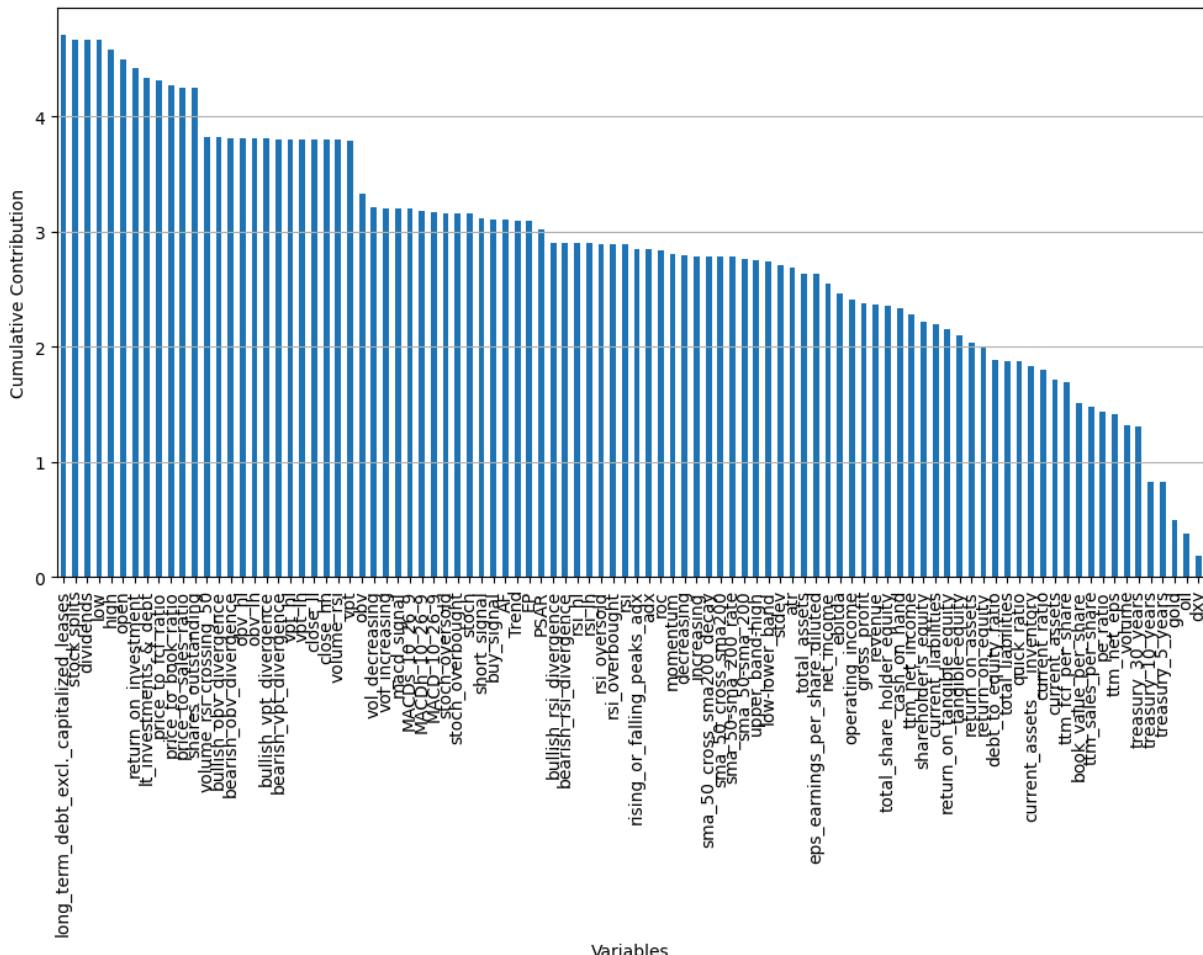


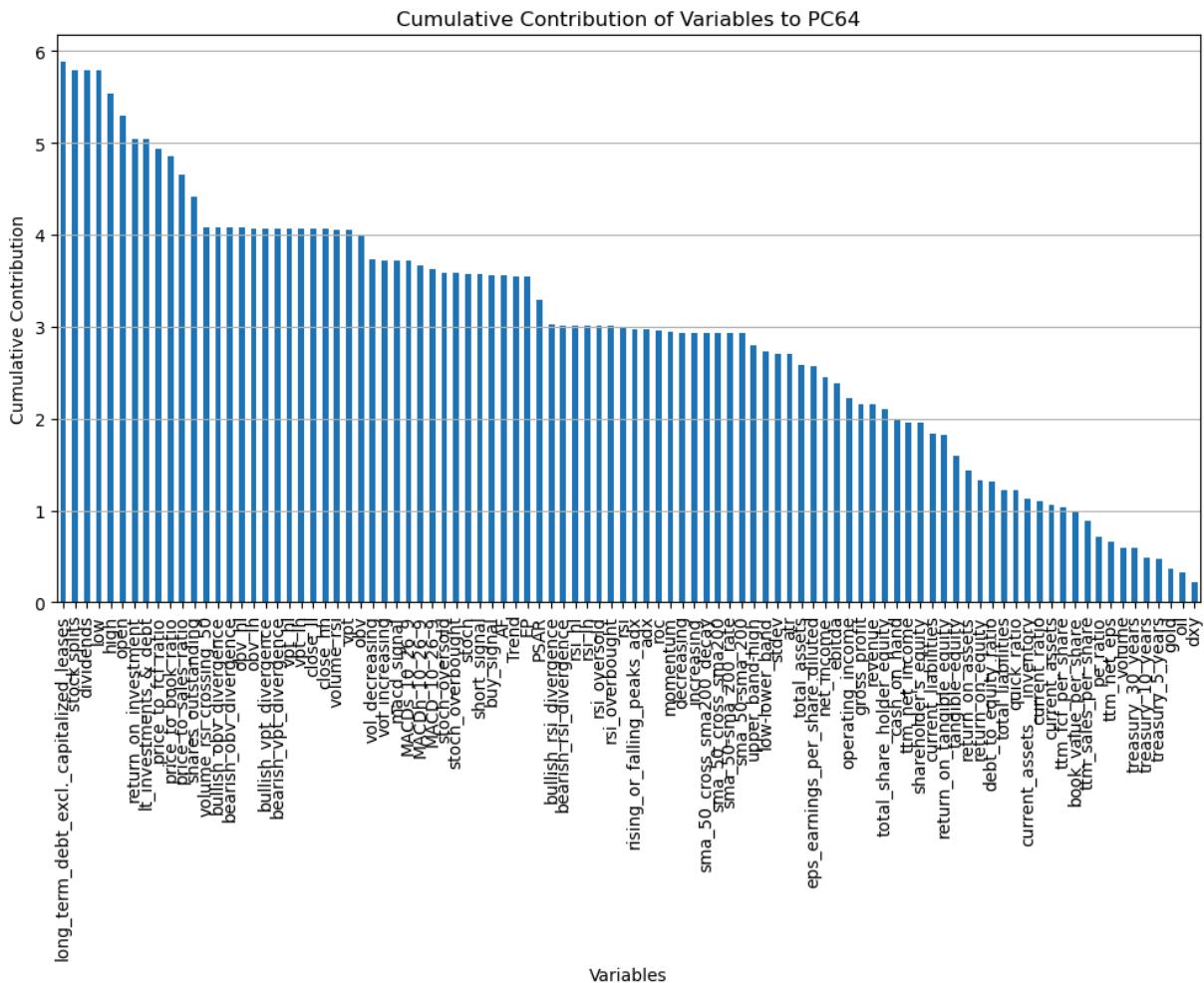


## Cumulative Contribution of Variables to PC62

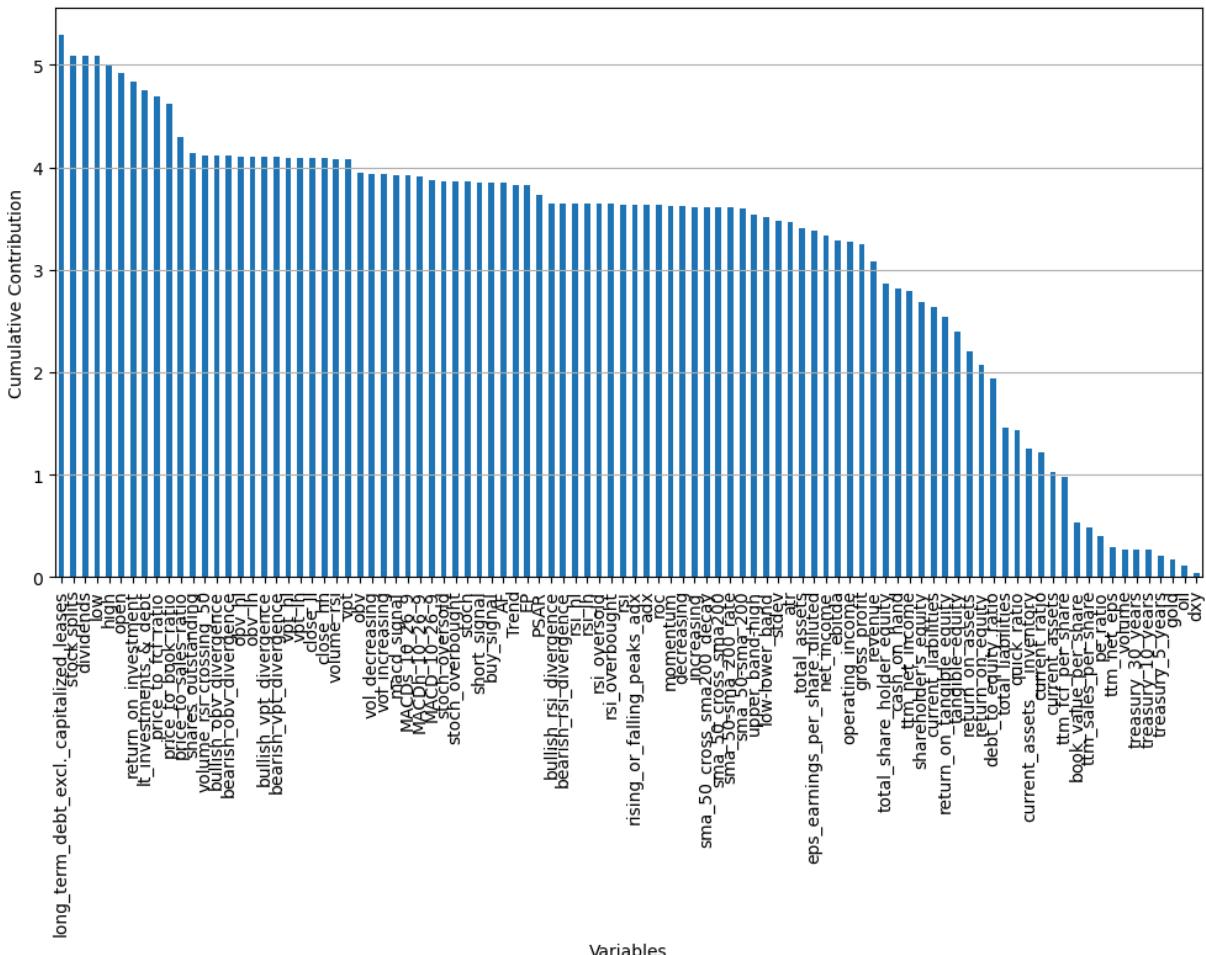


Cumulative Contribution of Variables to PC63

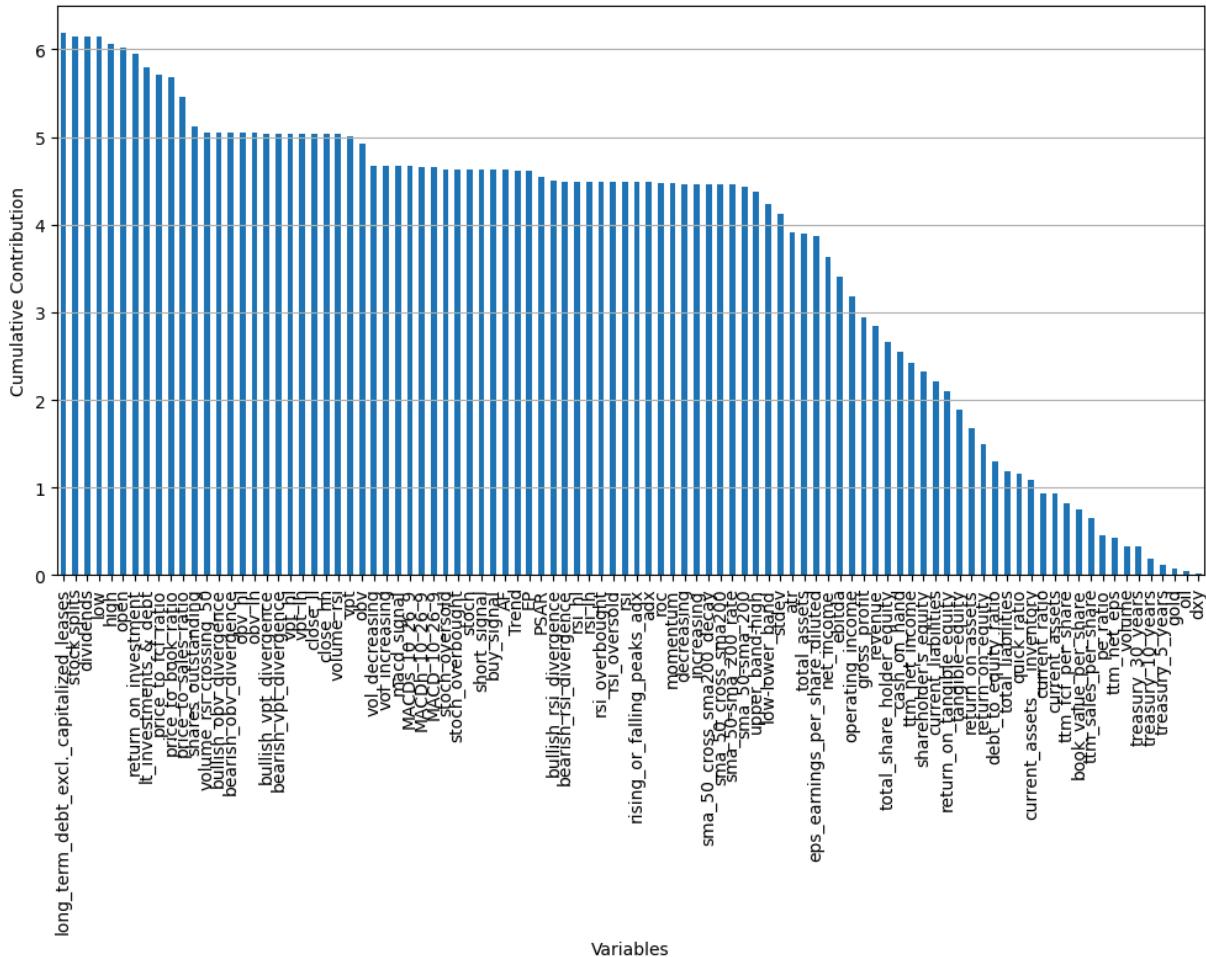




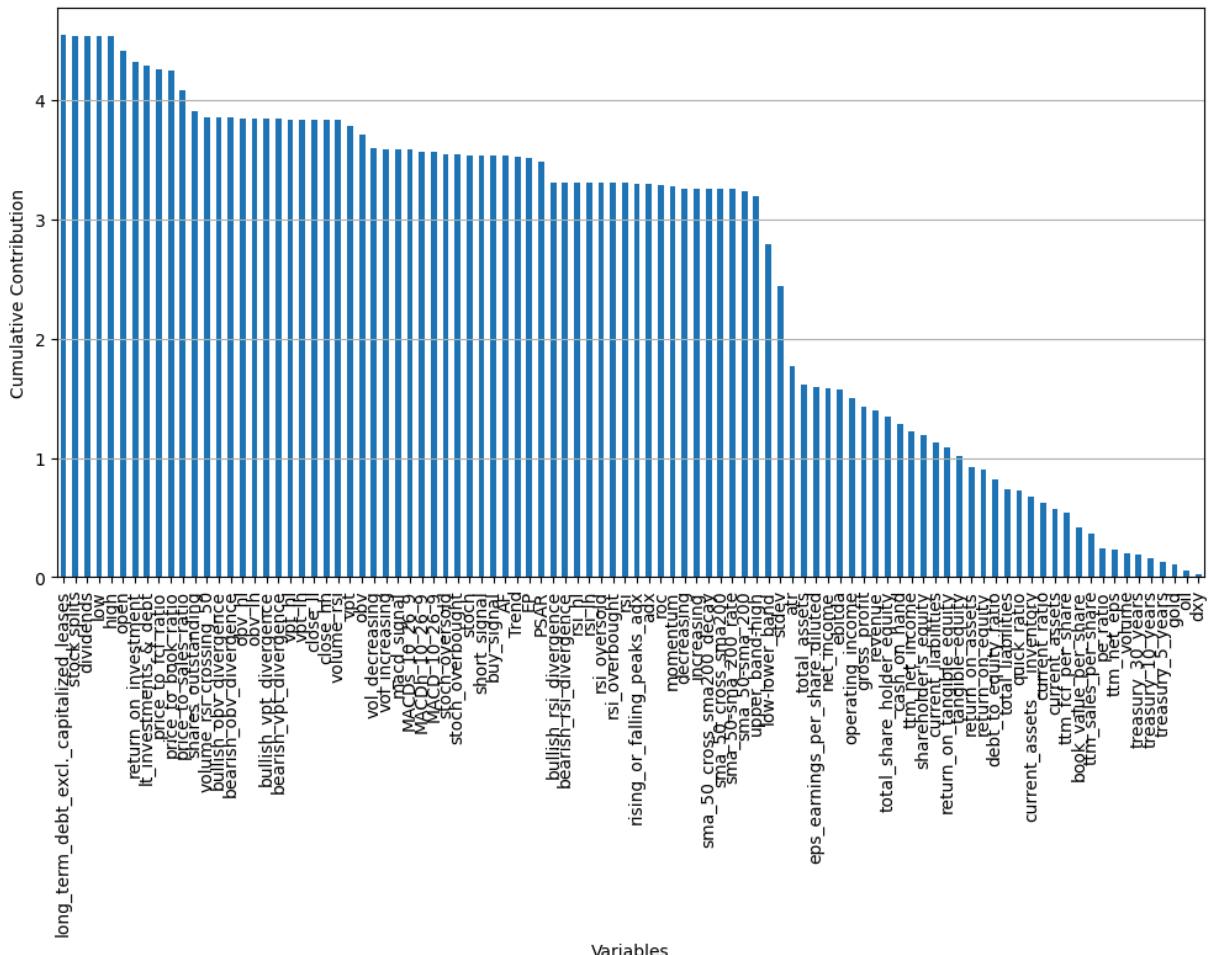
## Cumulative Contribution of Variables to PC65

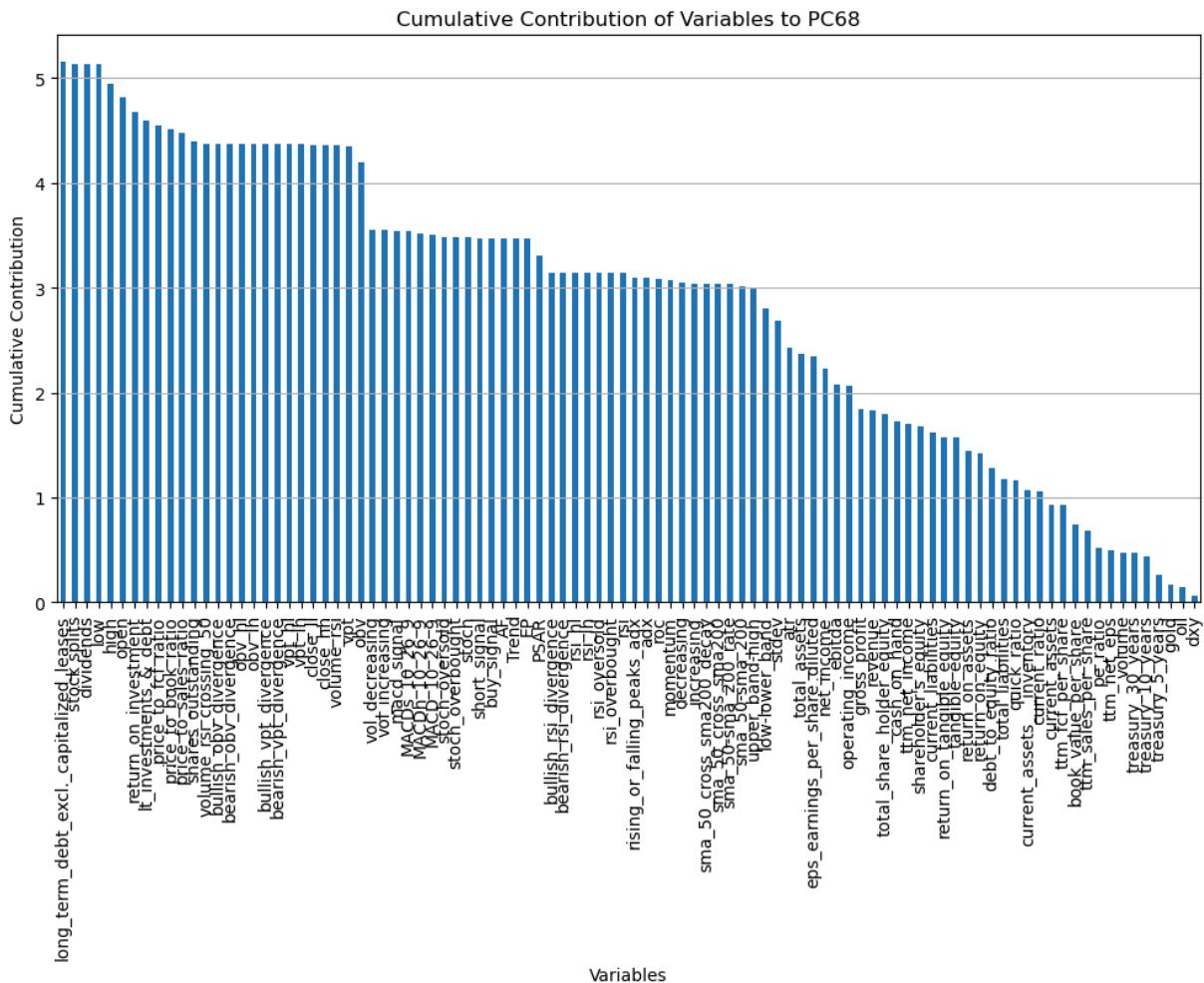


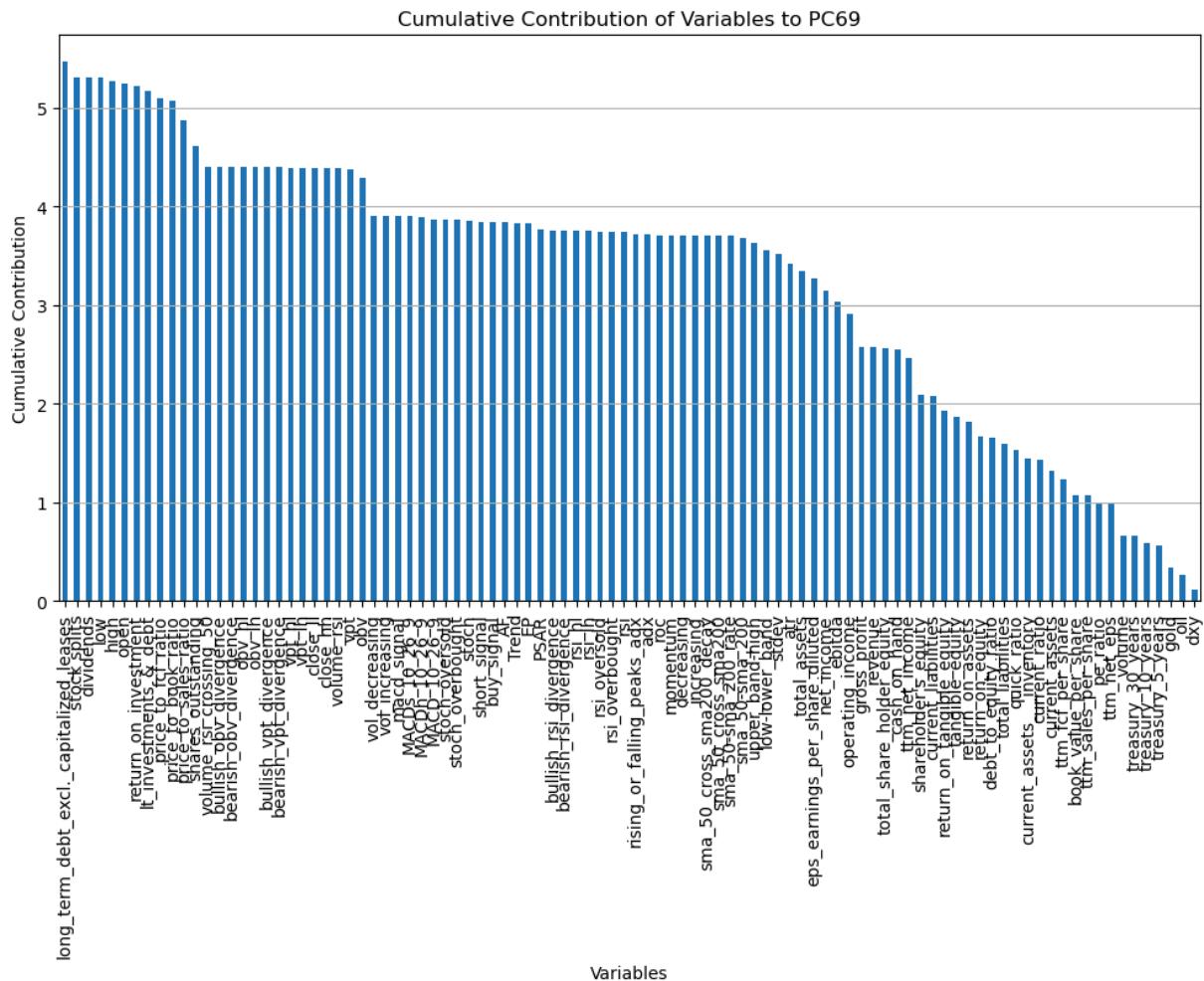
## Cumulative Contribution of Variables to PC66

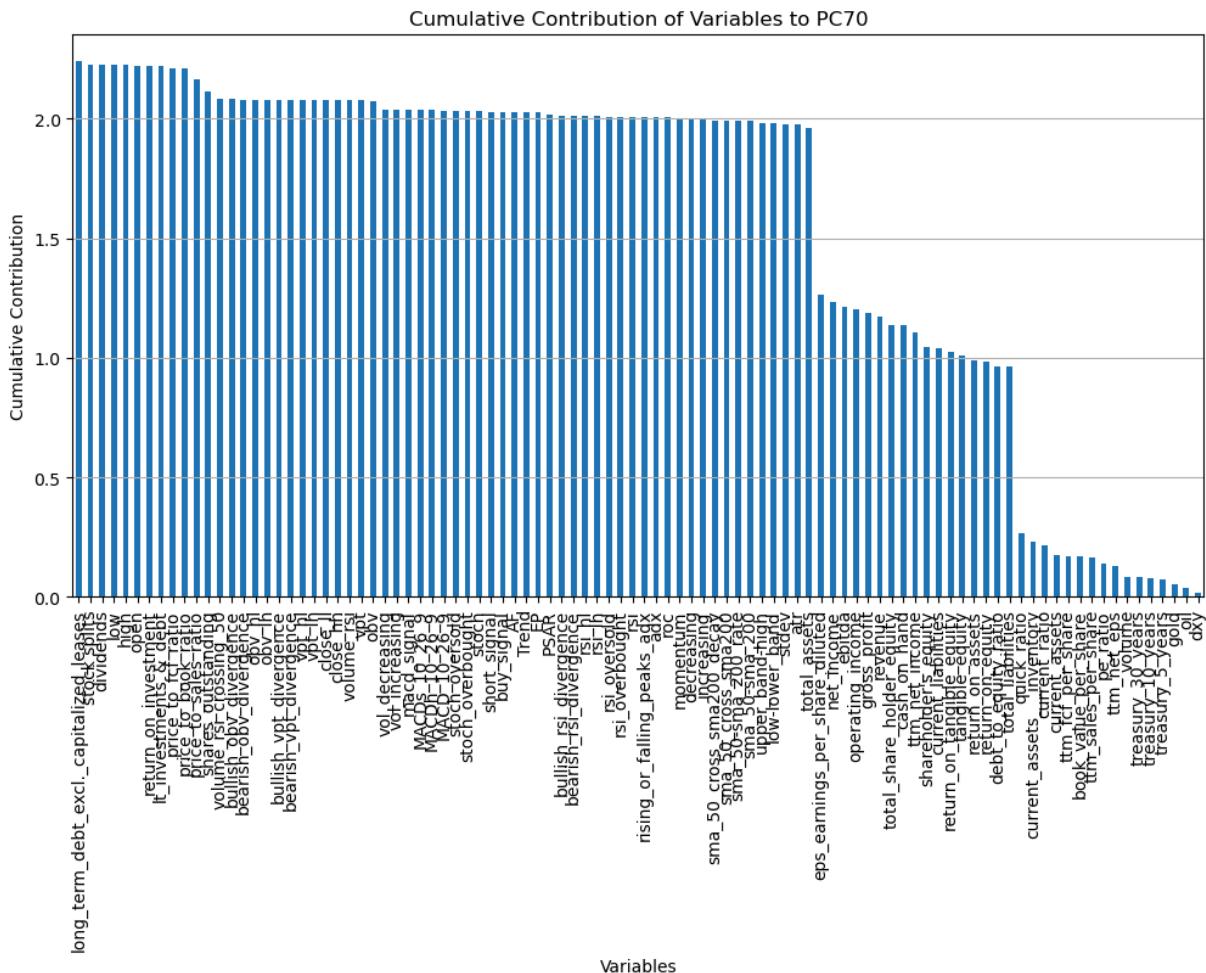


## Cumulative Contribution of Variables to PC67

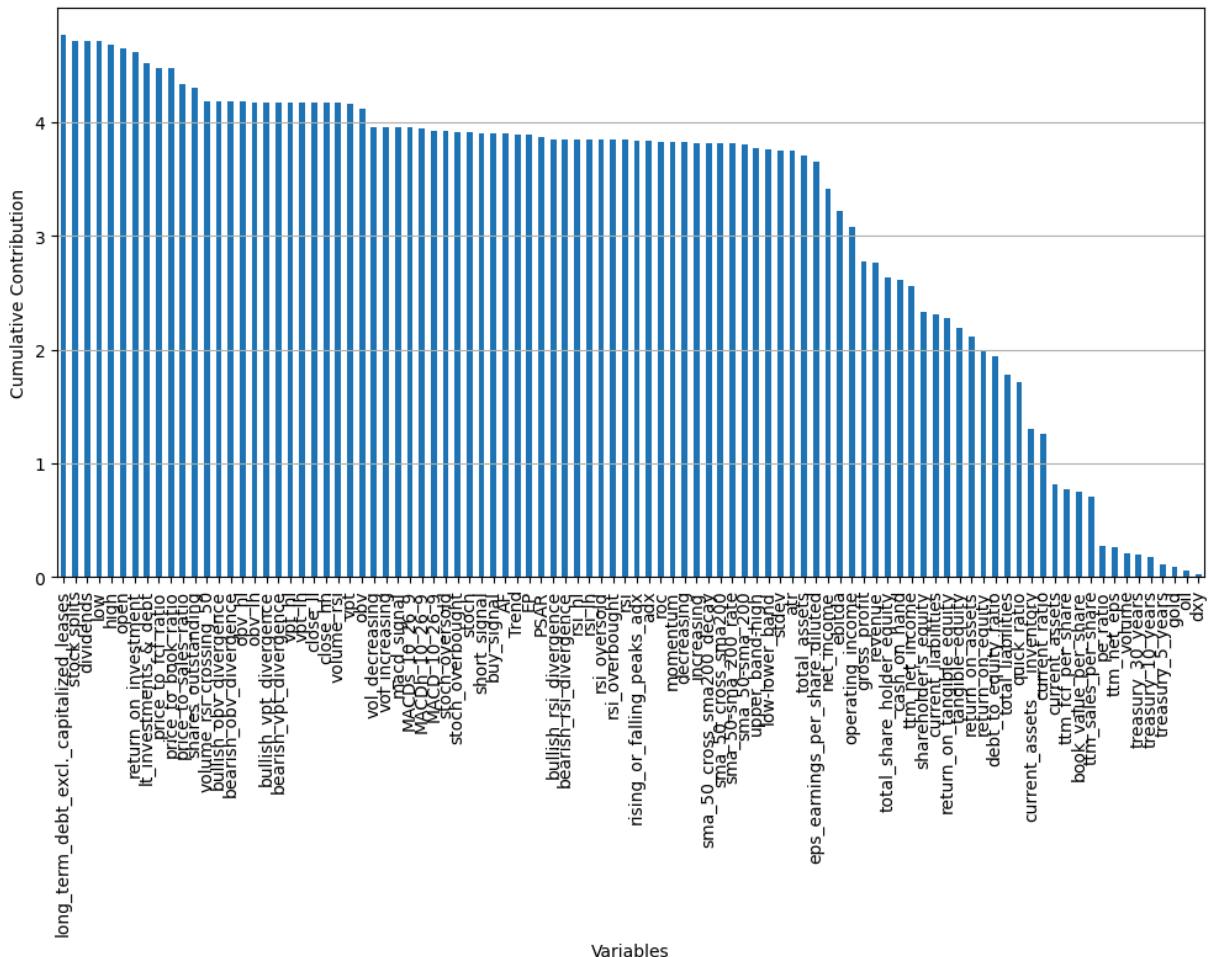




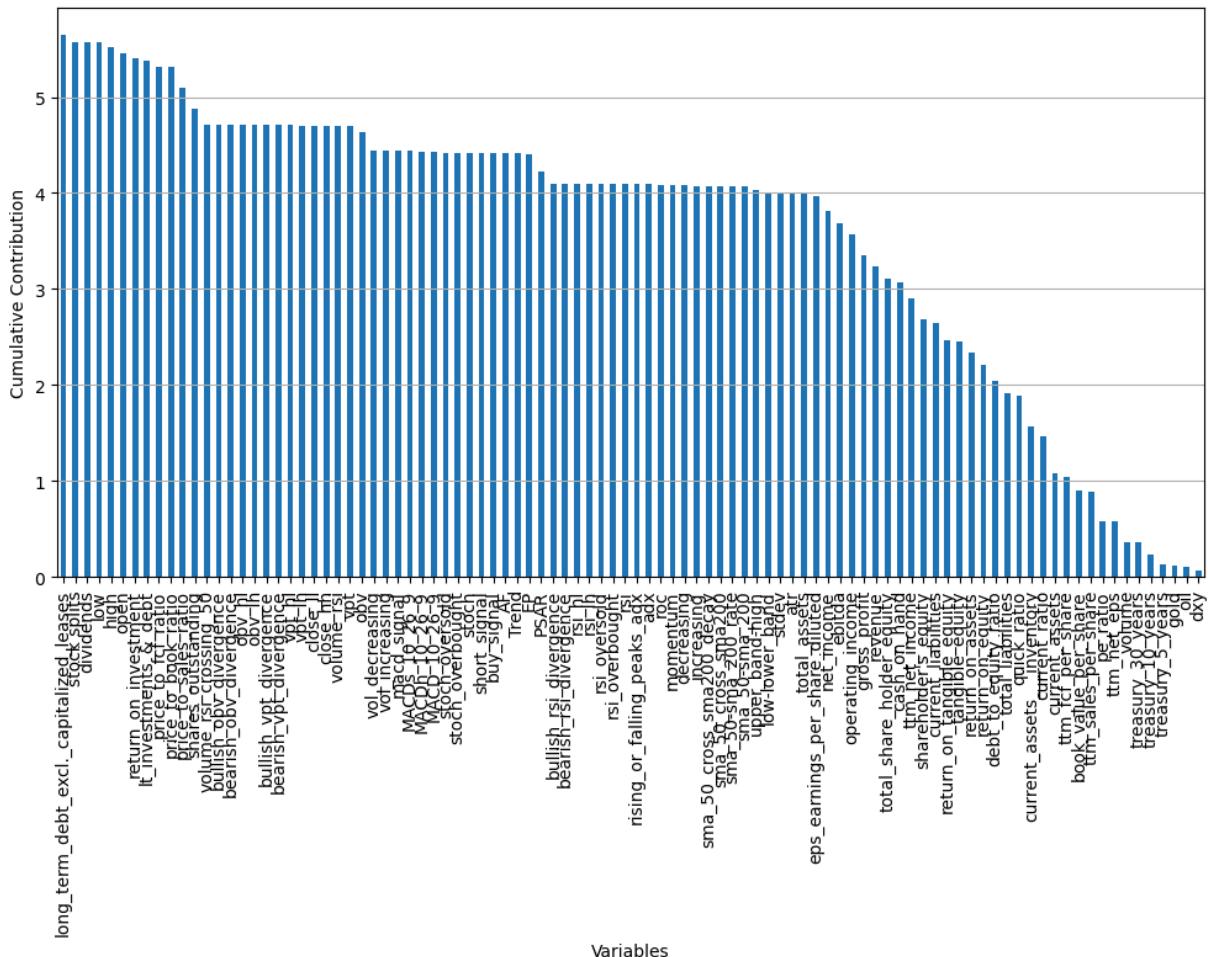


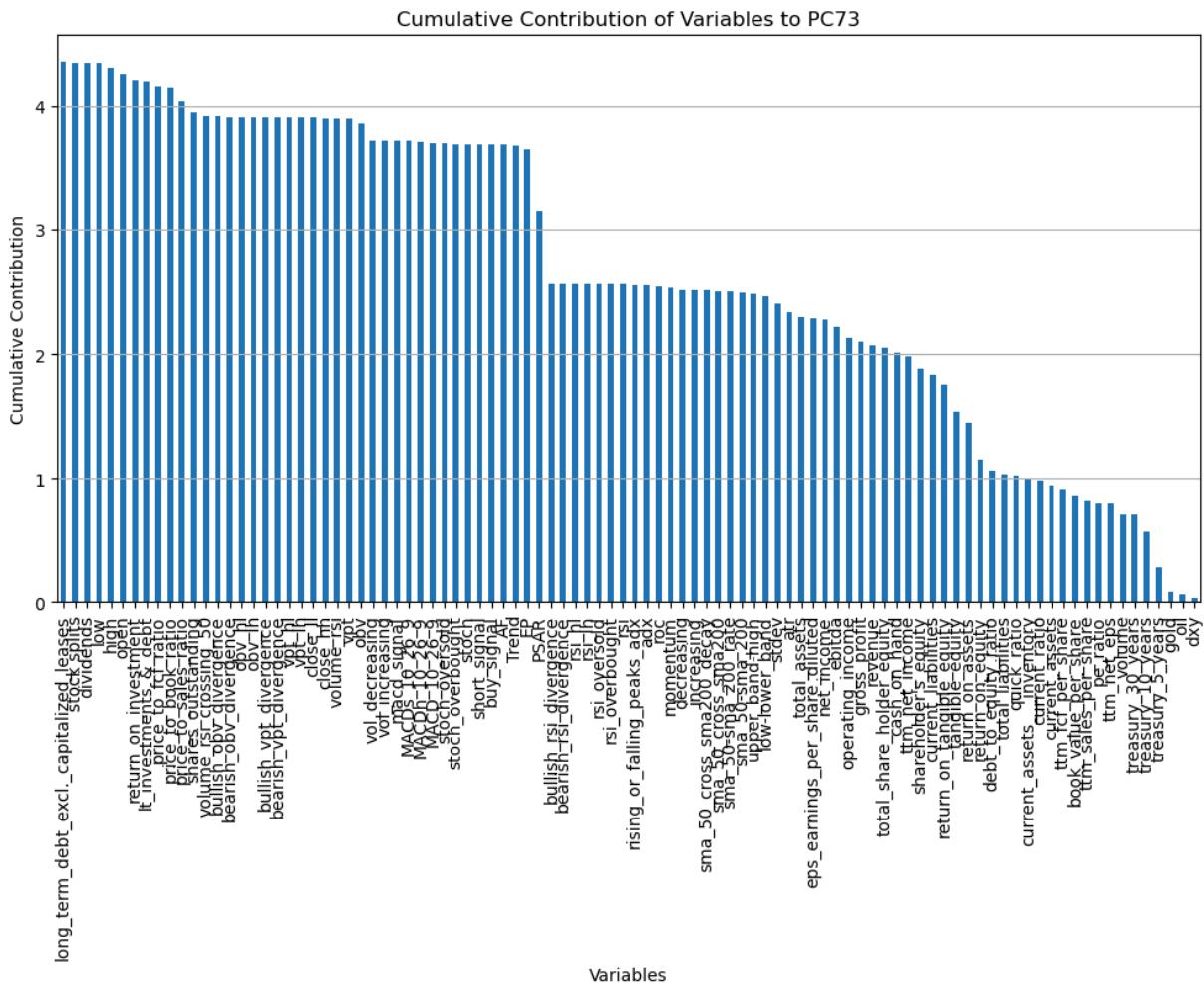


## Cumulative Contribution of Variables to PC71

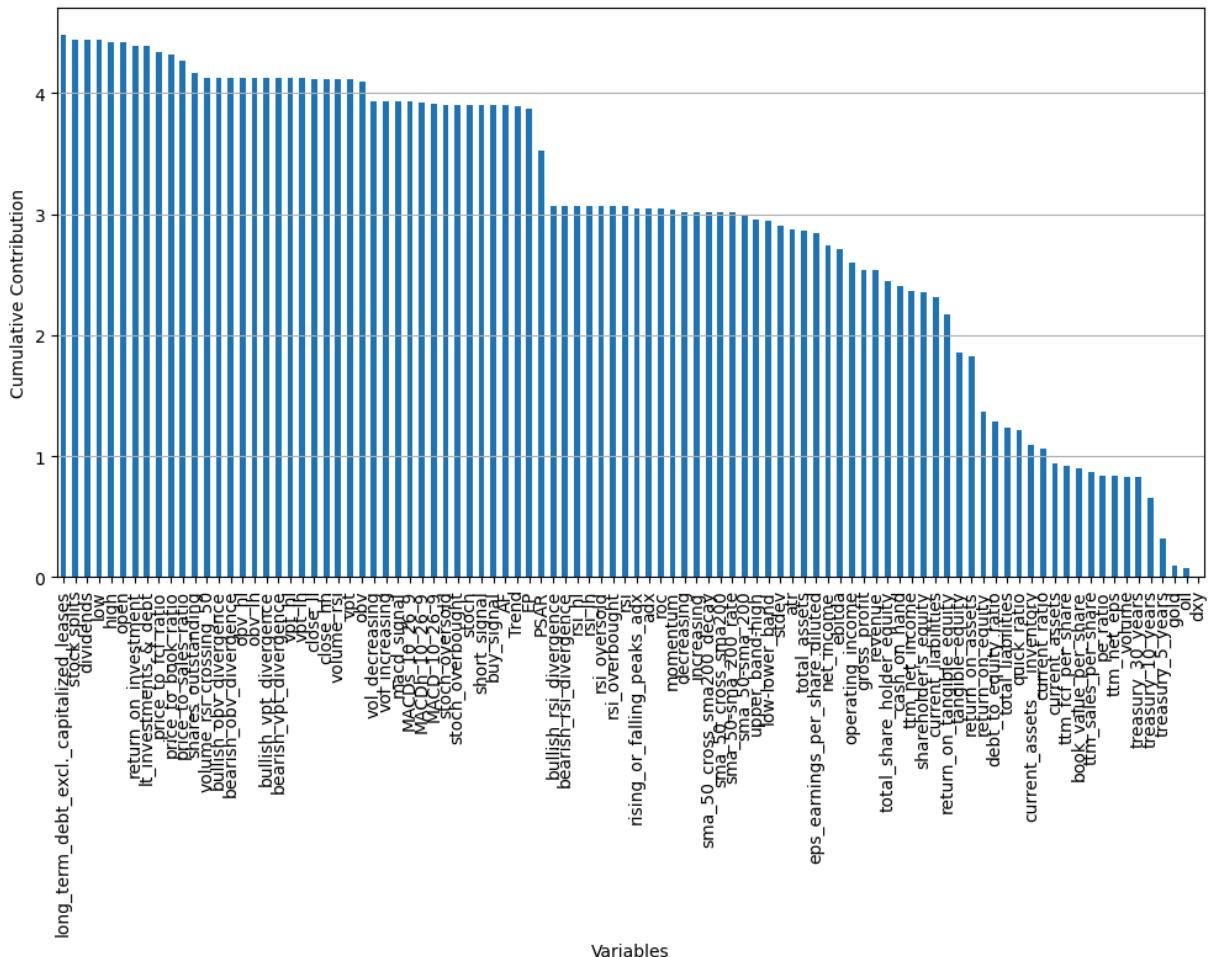


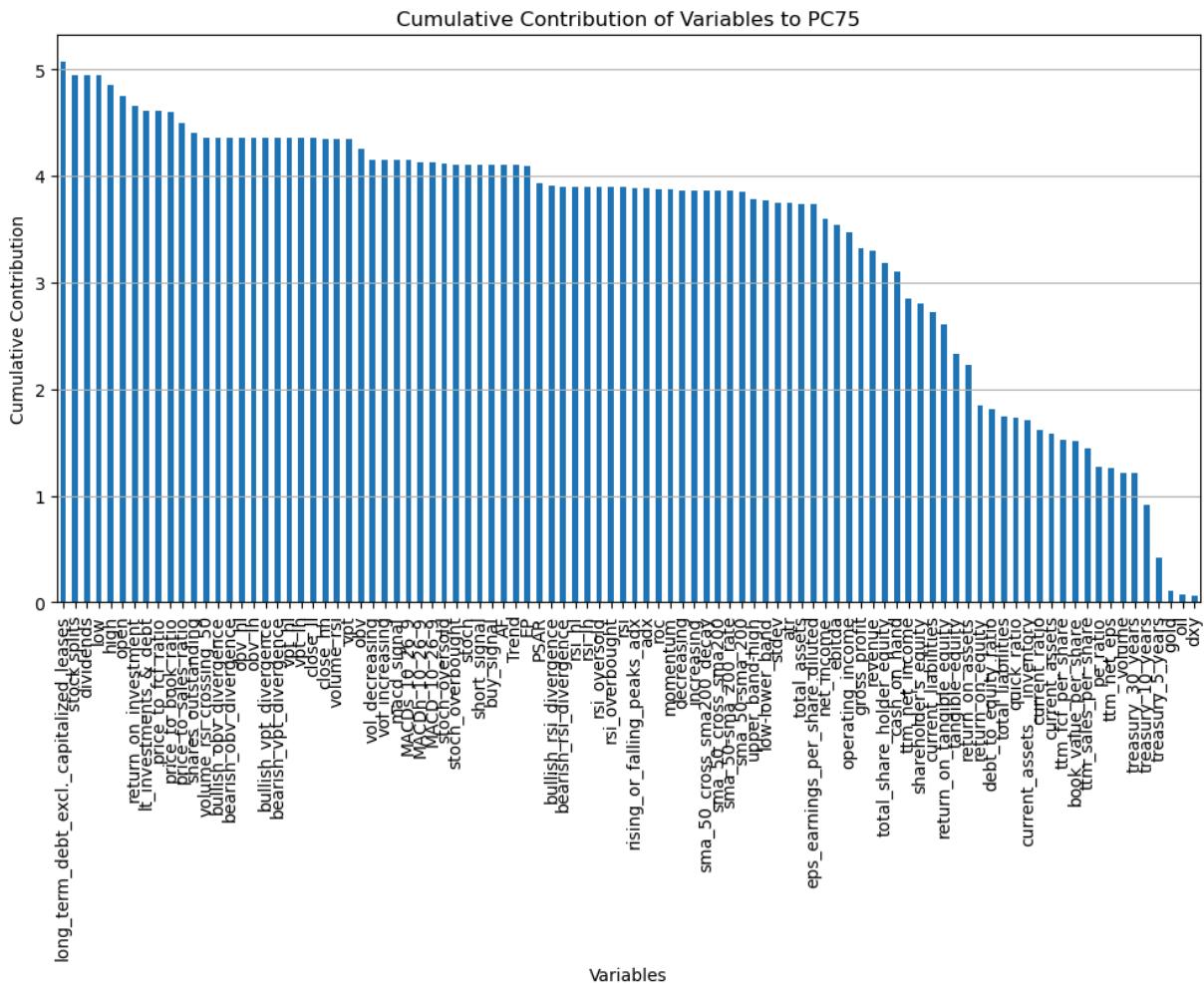
Cumulative Contribution of Variables to PC72

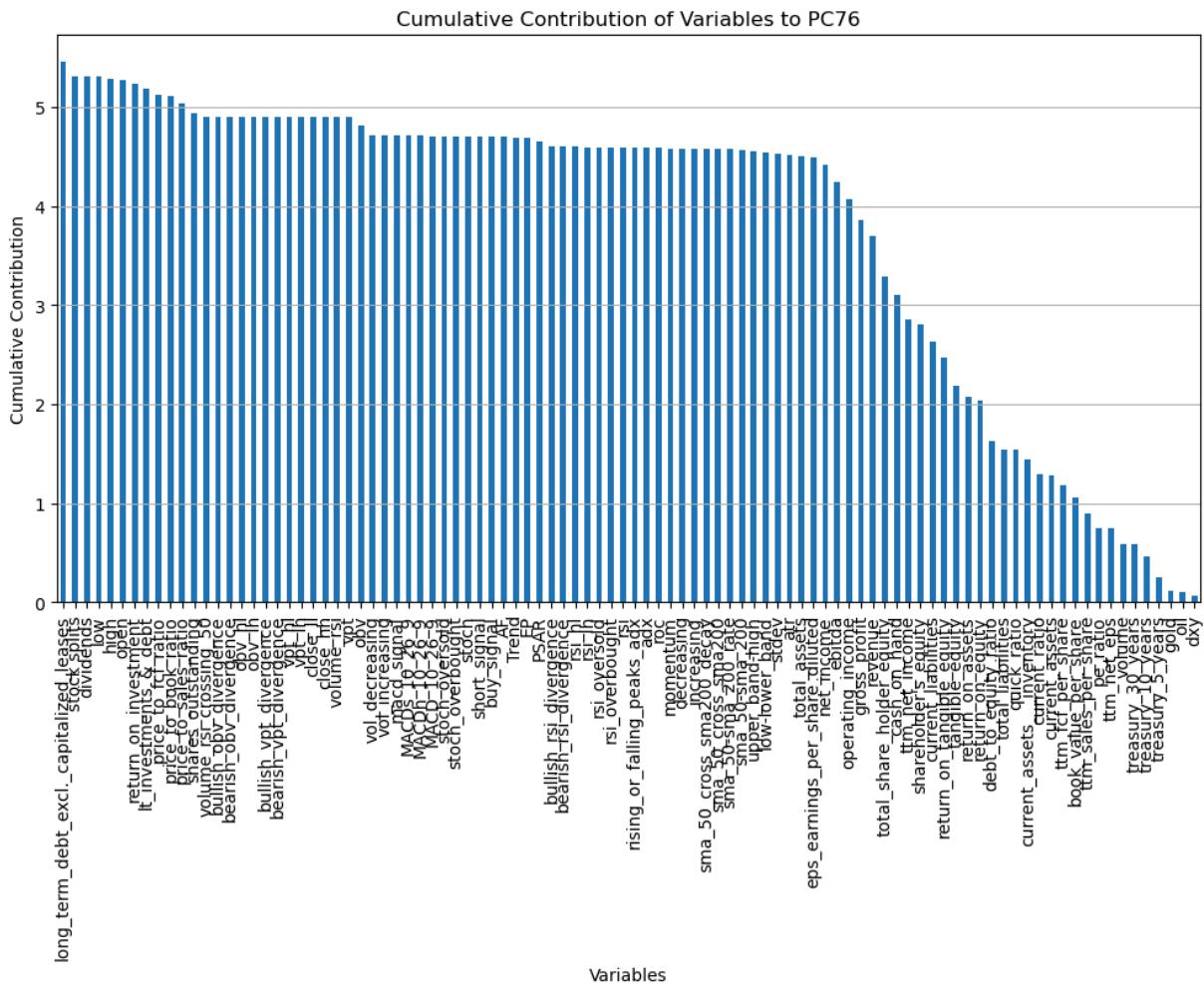


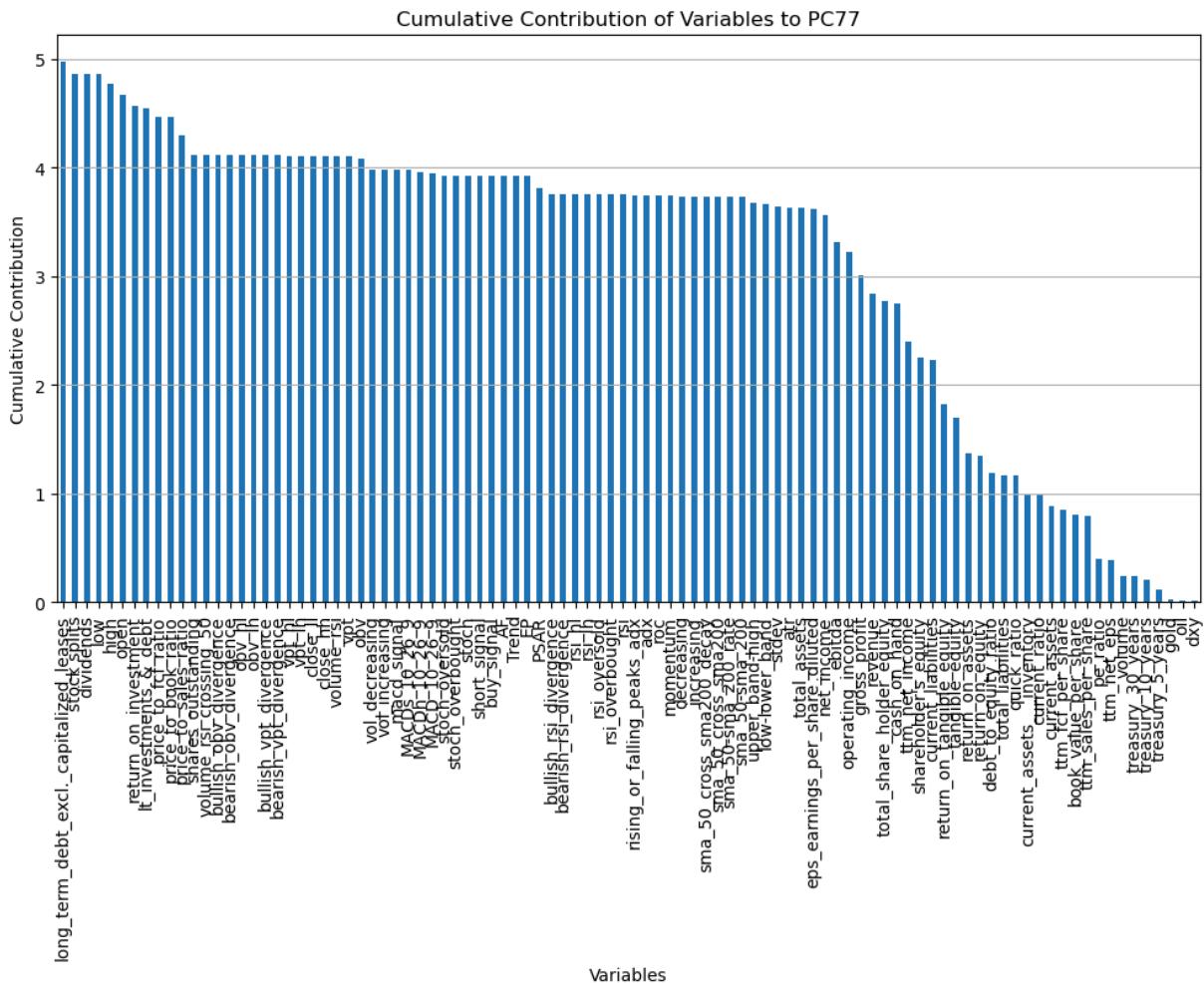


## Cumulative Contribution of Variables to PC74

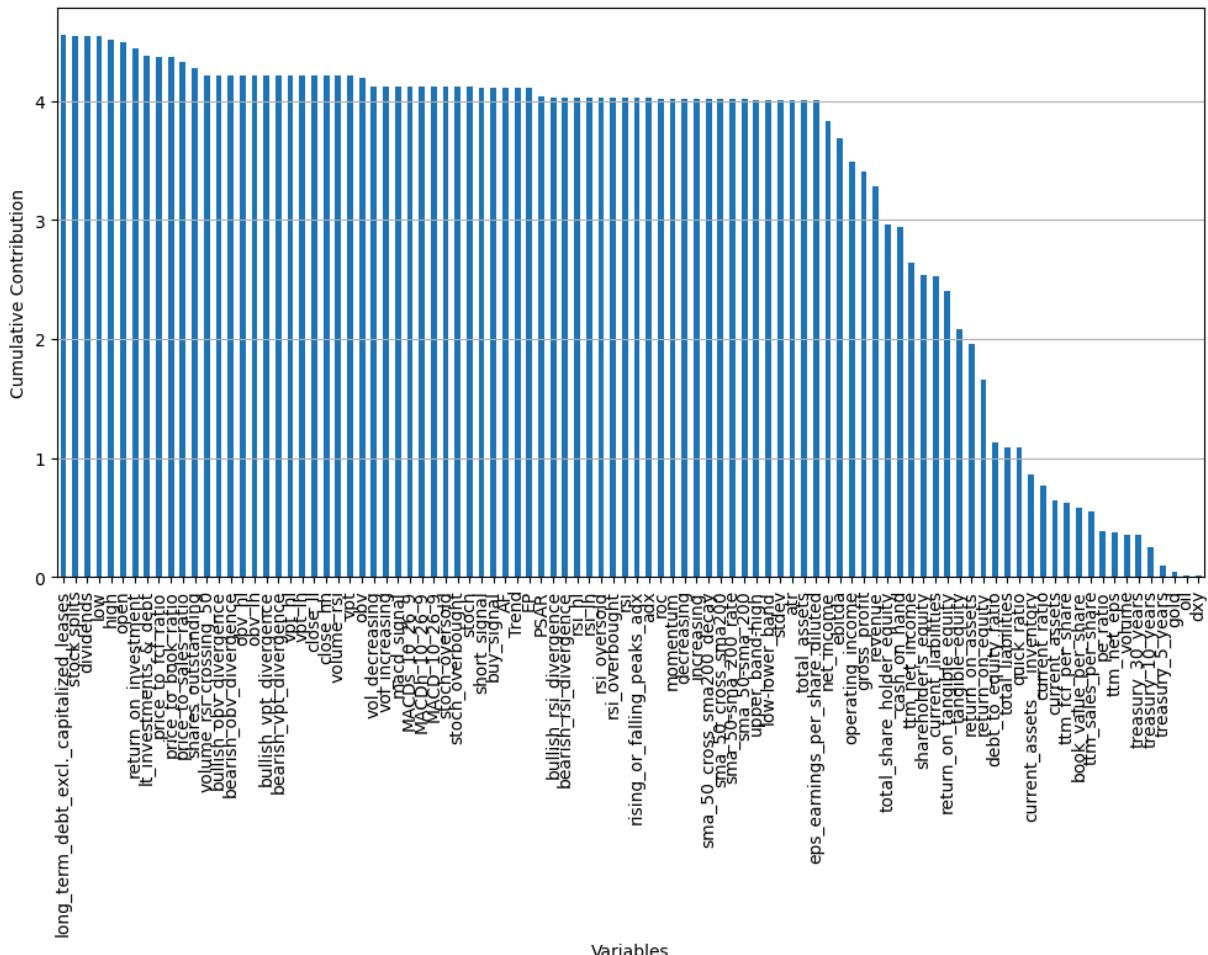


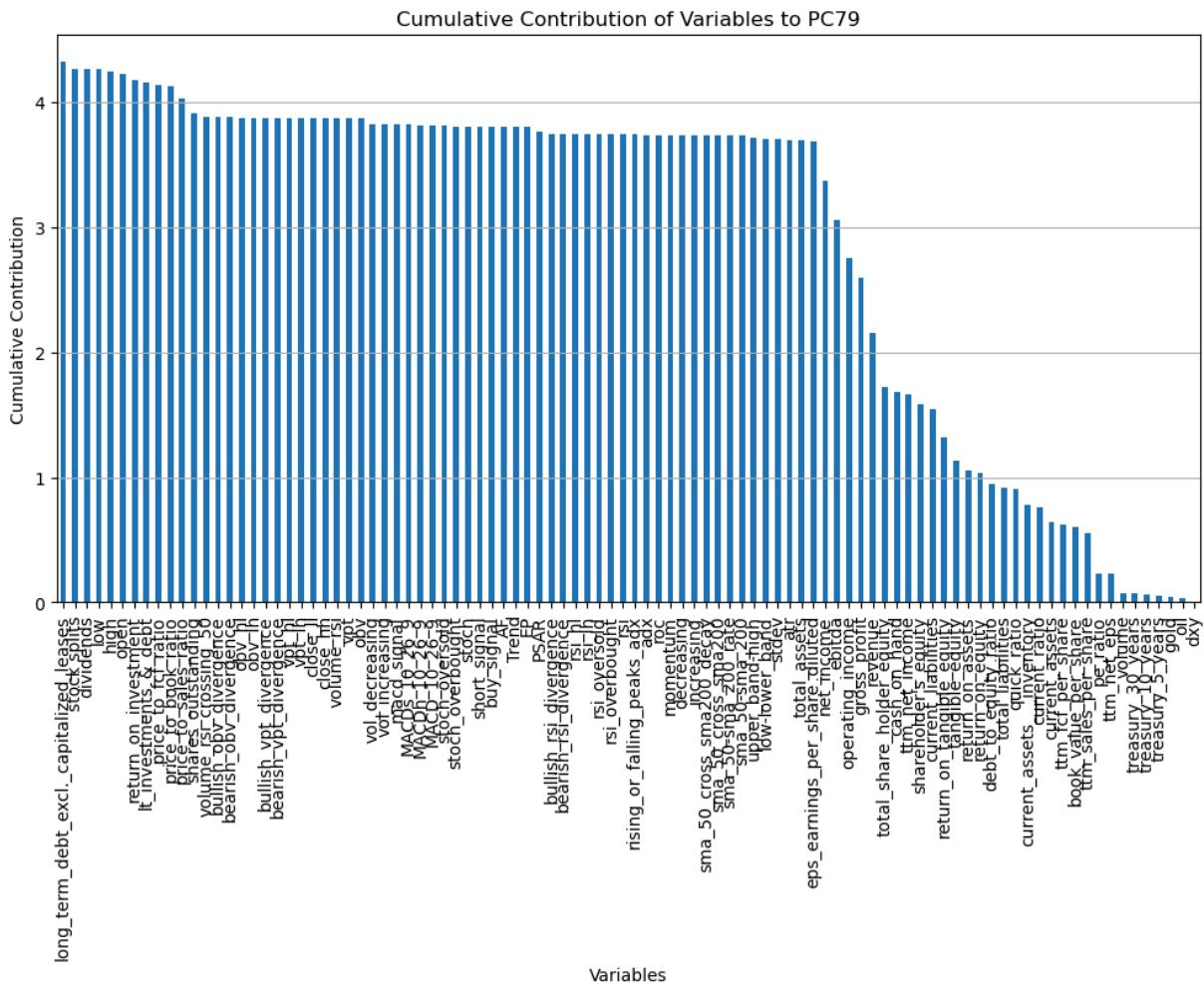


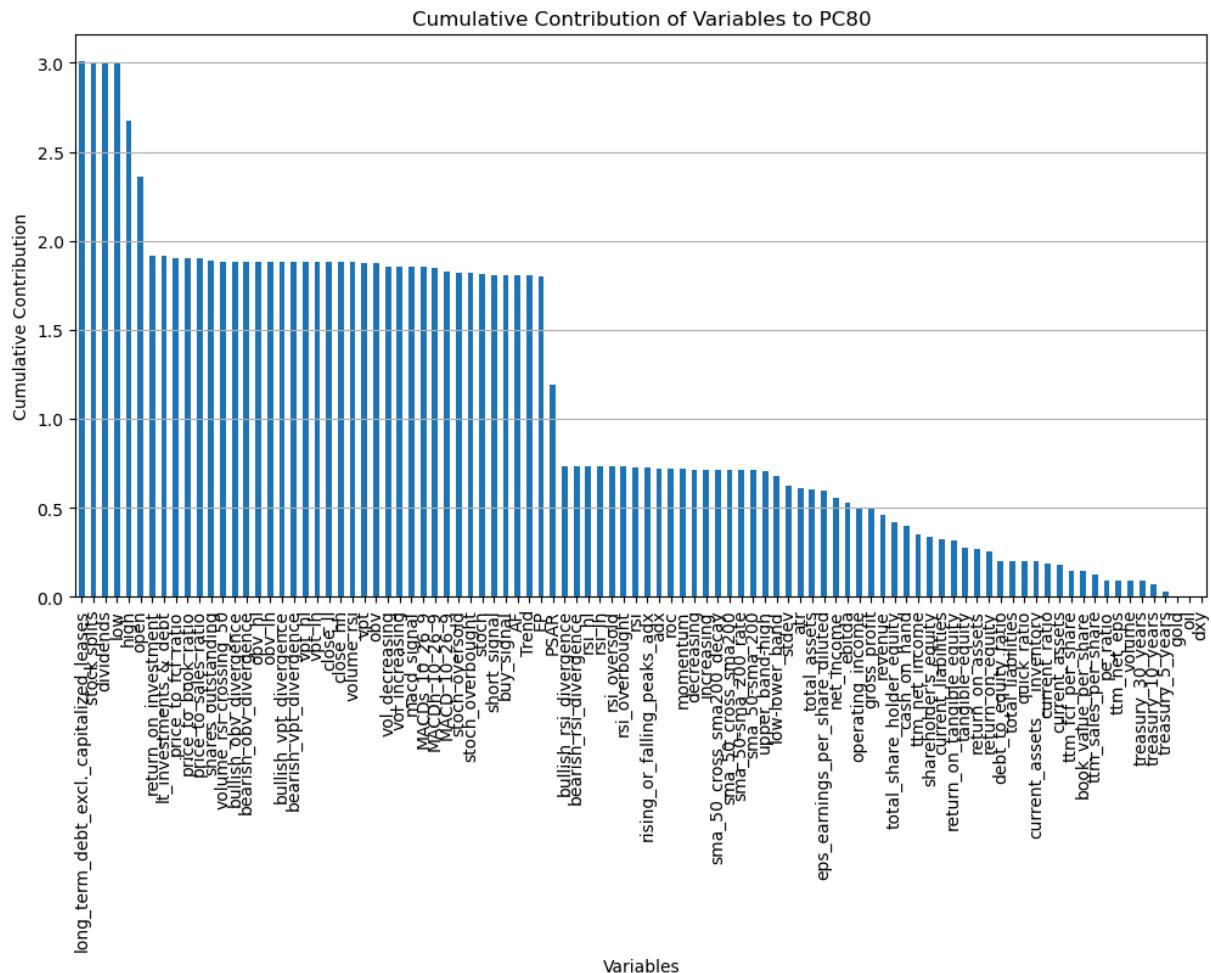




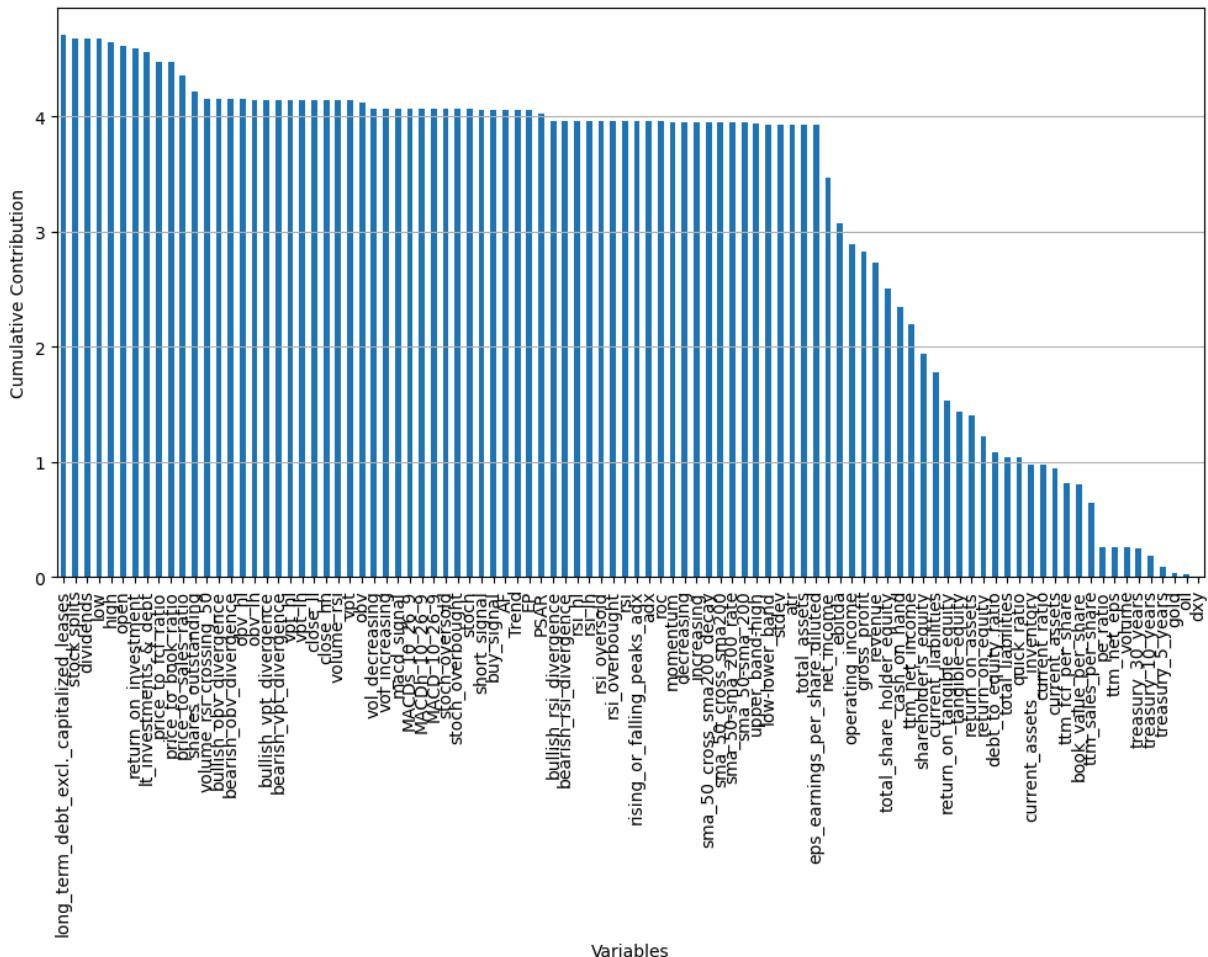
## Cumulative Contribution of Variables to PC78



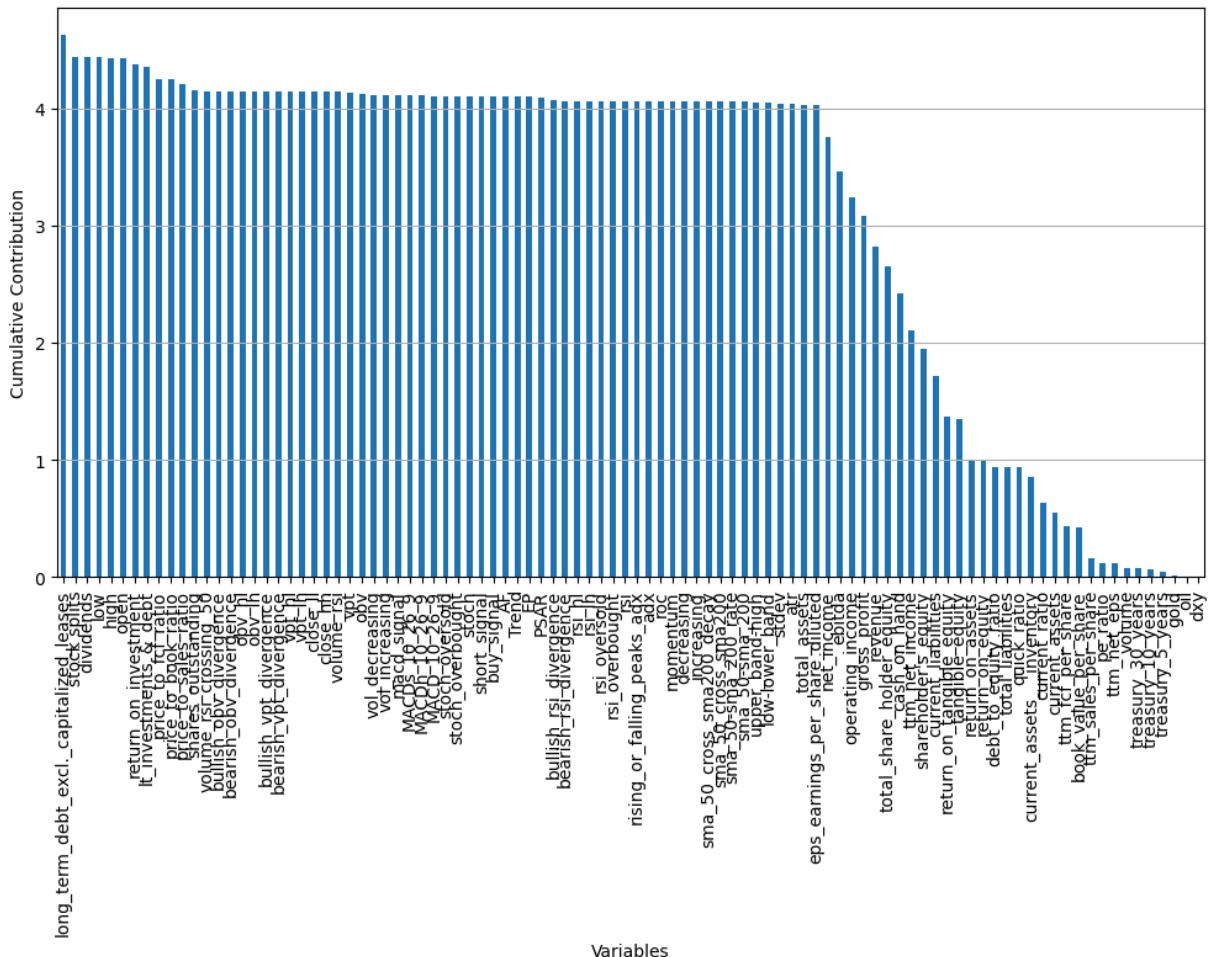


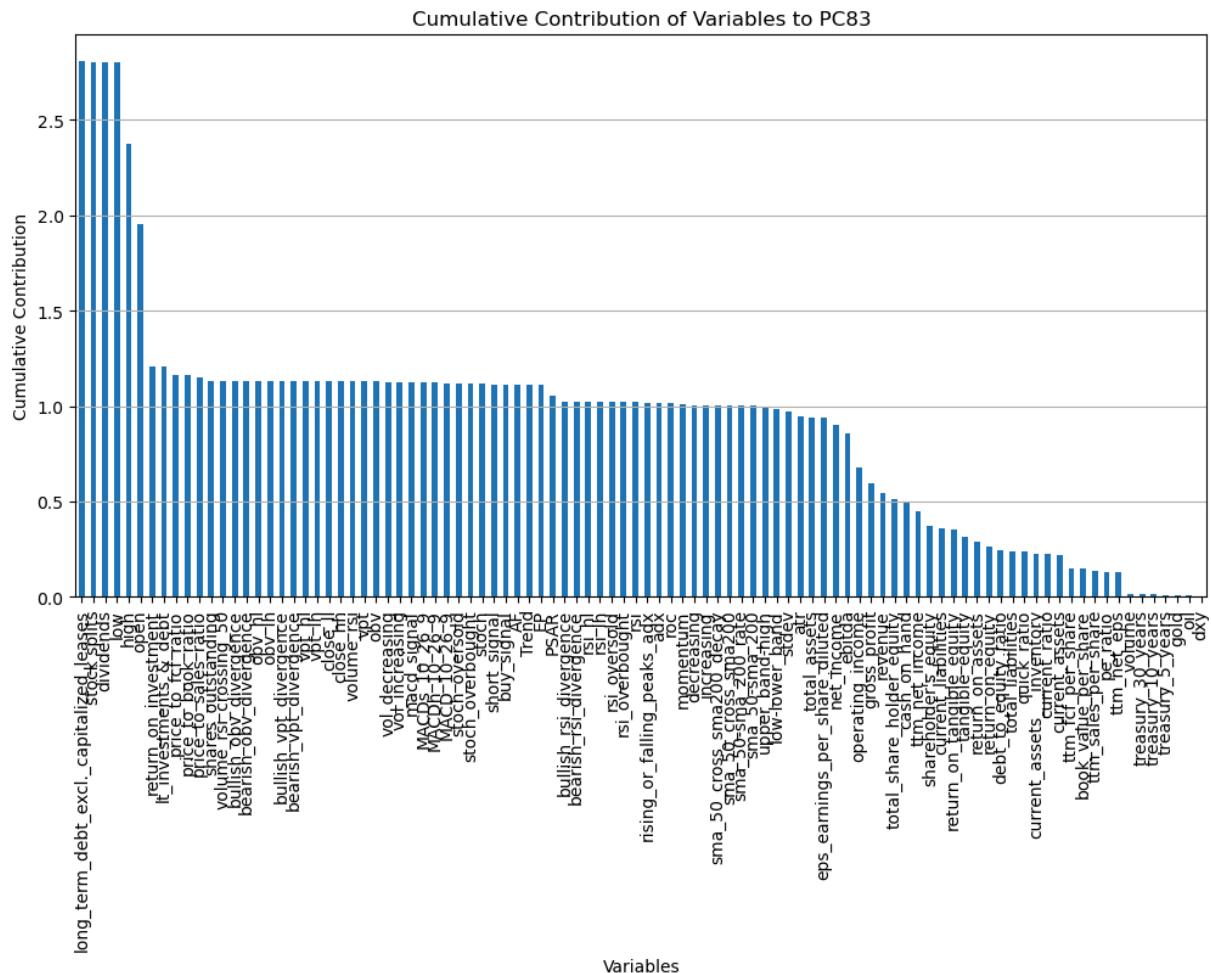


Cumulative Contribution of Variables to PC81

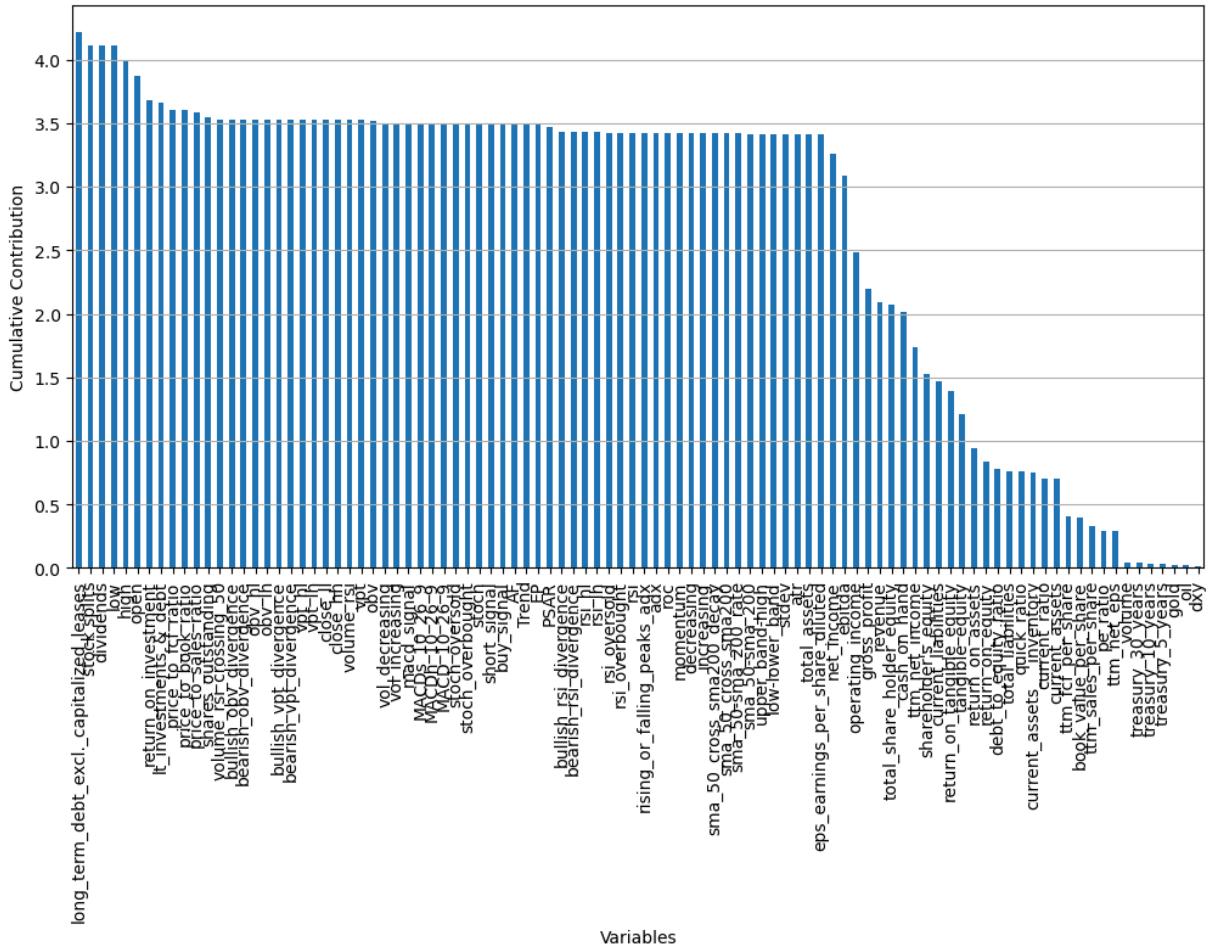


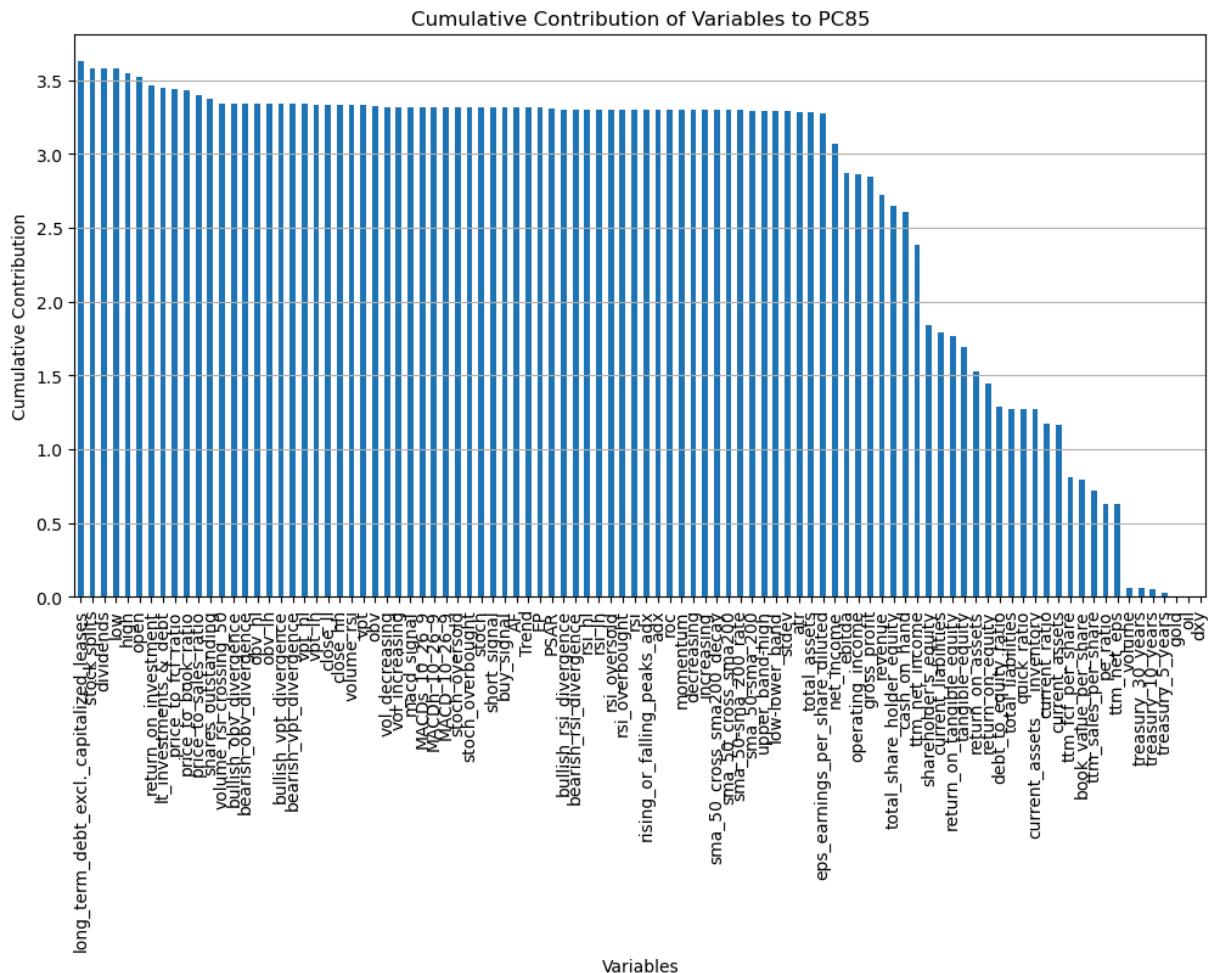
## Cumulative Contribution of Variables to PC82

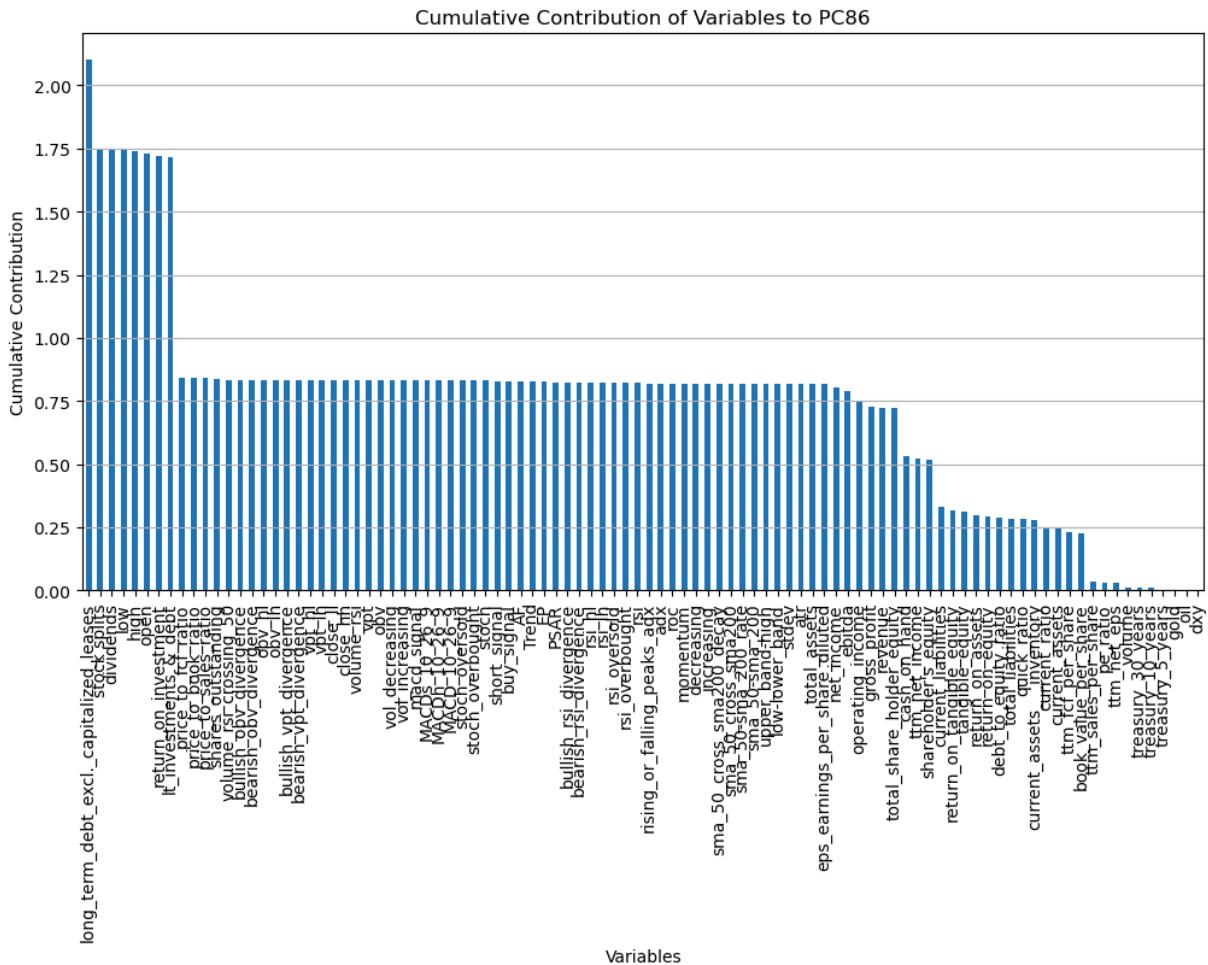


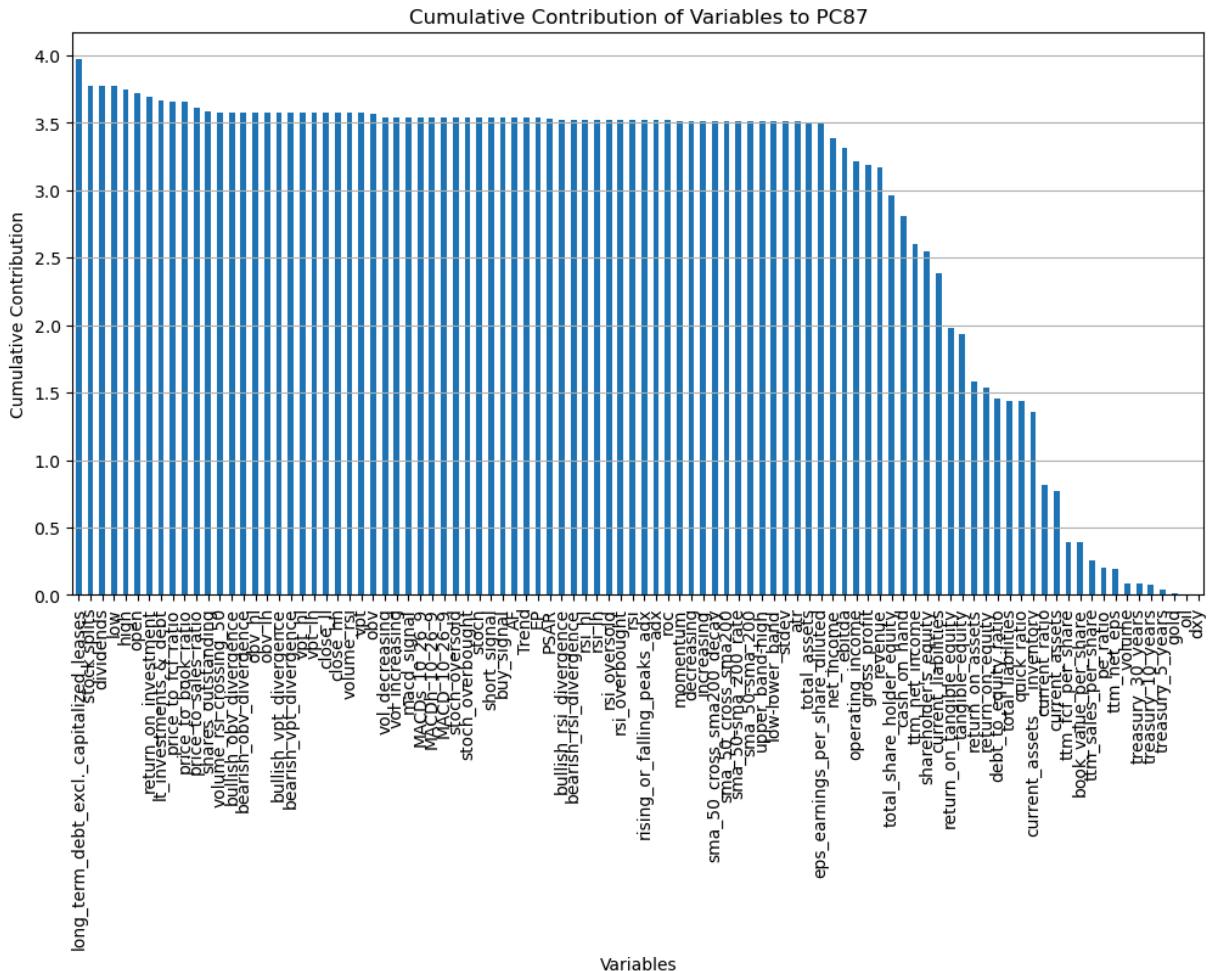


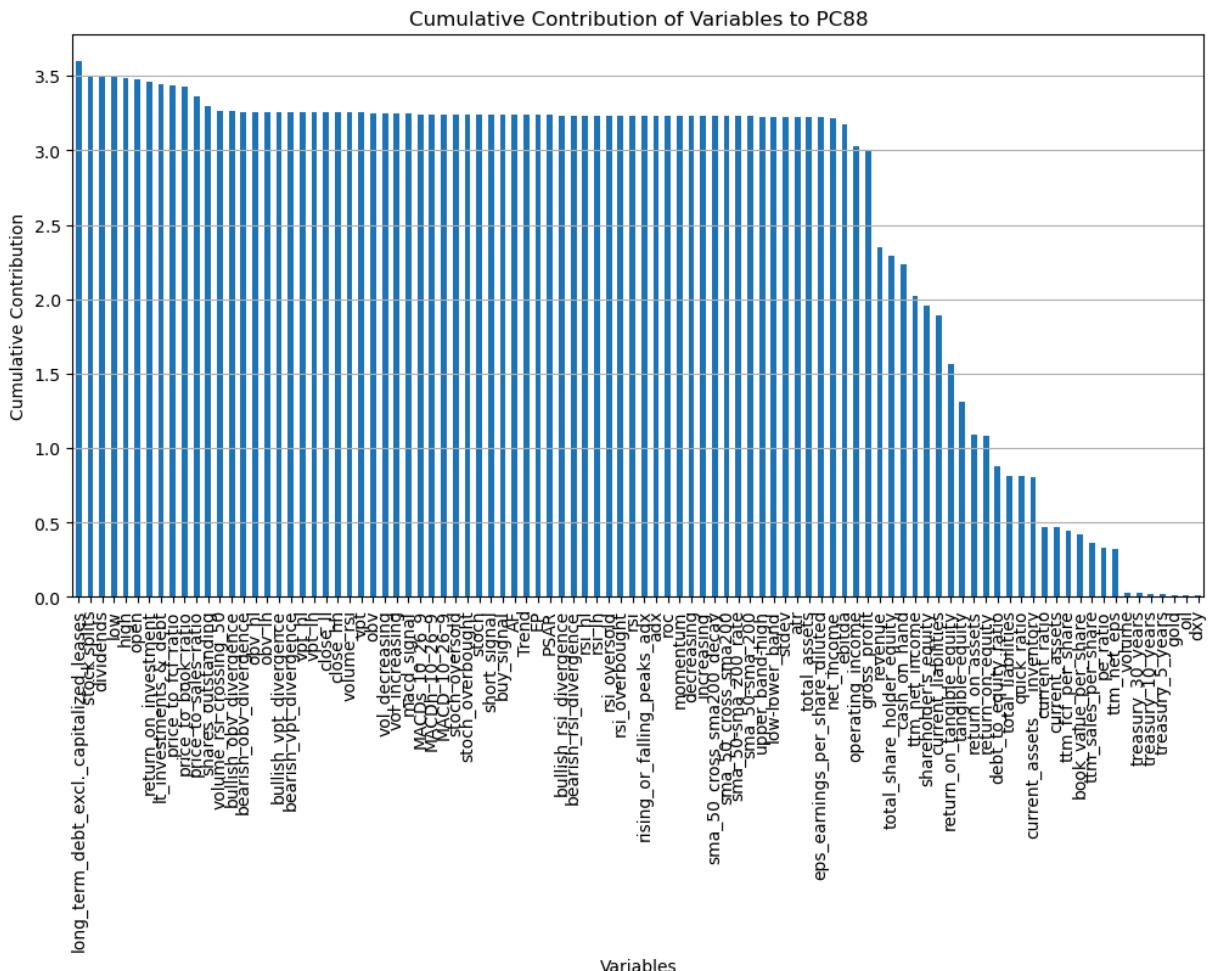
## Cumulative Contribution of Variables to PC84

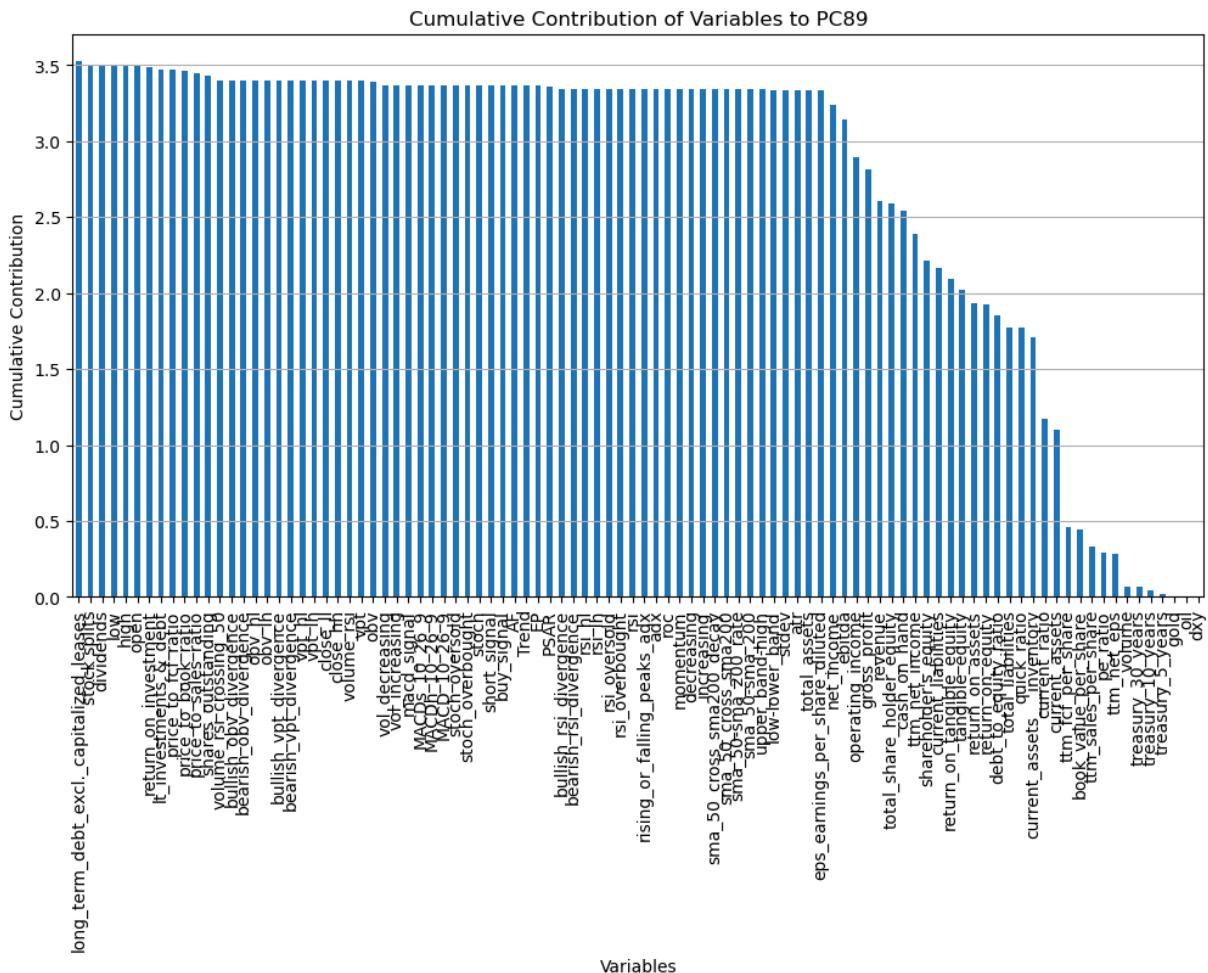


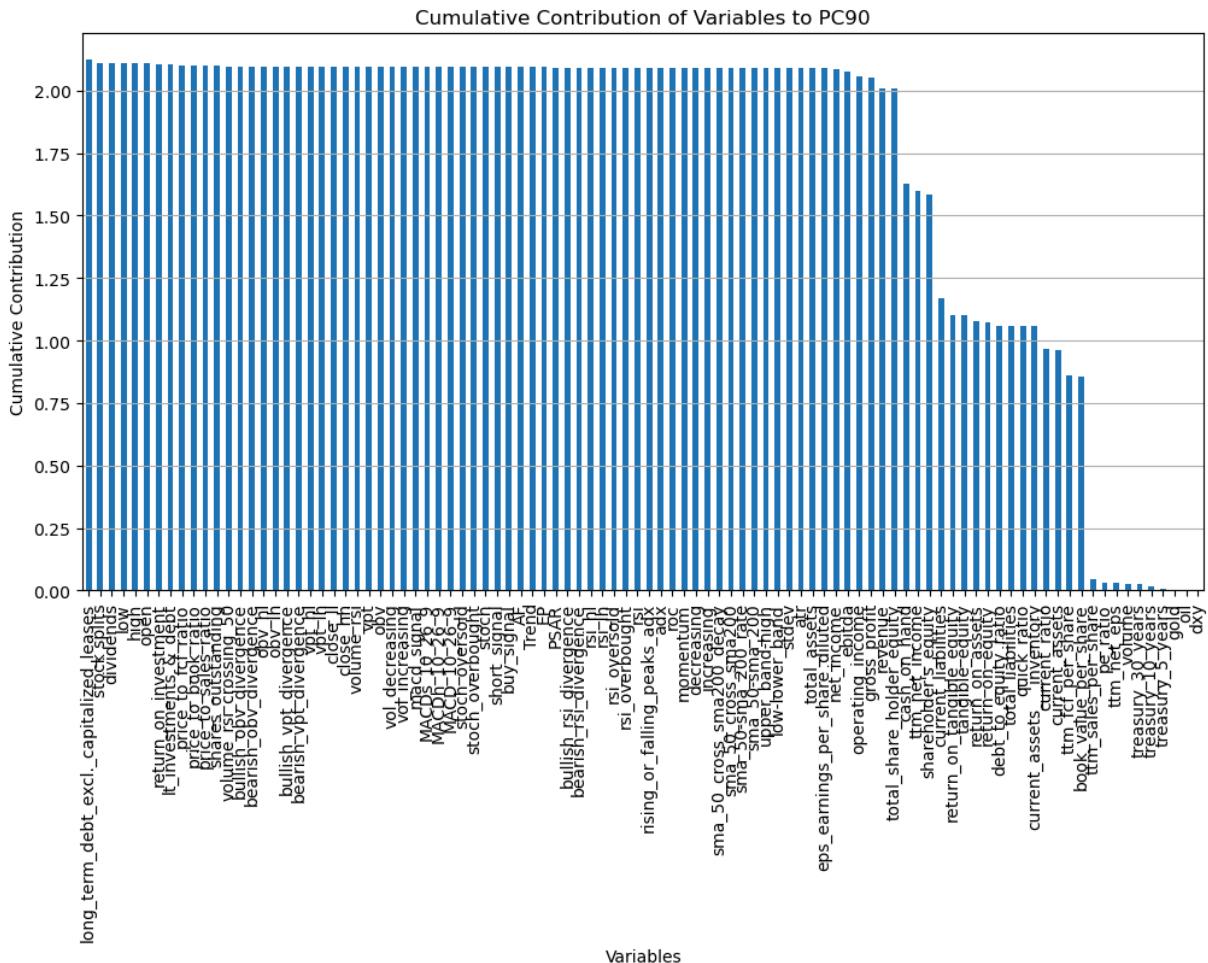


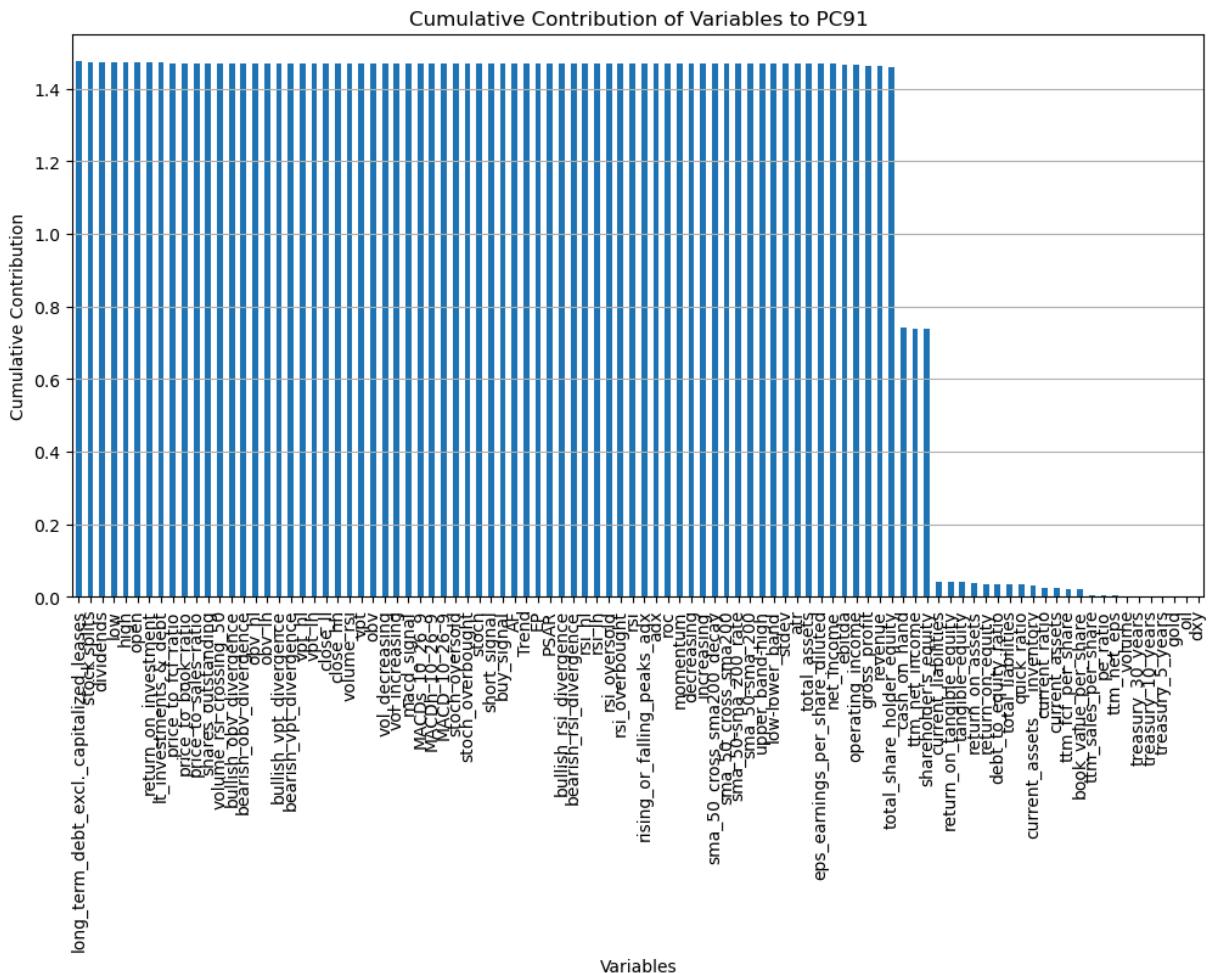


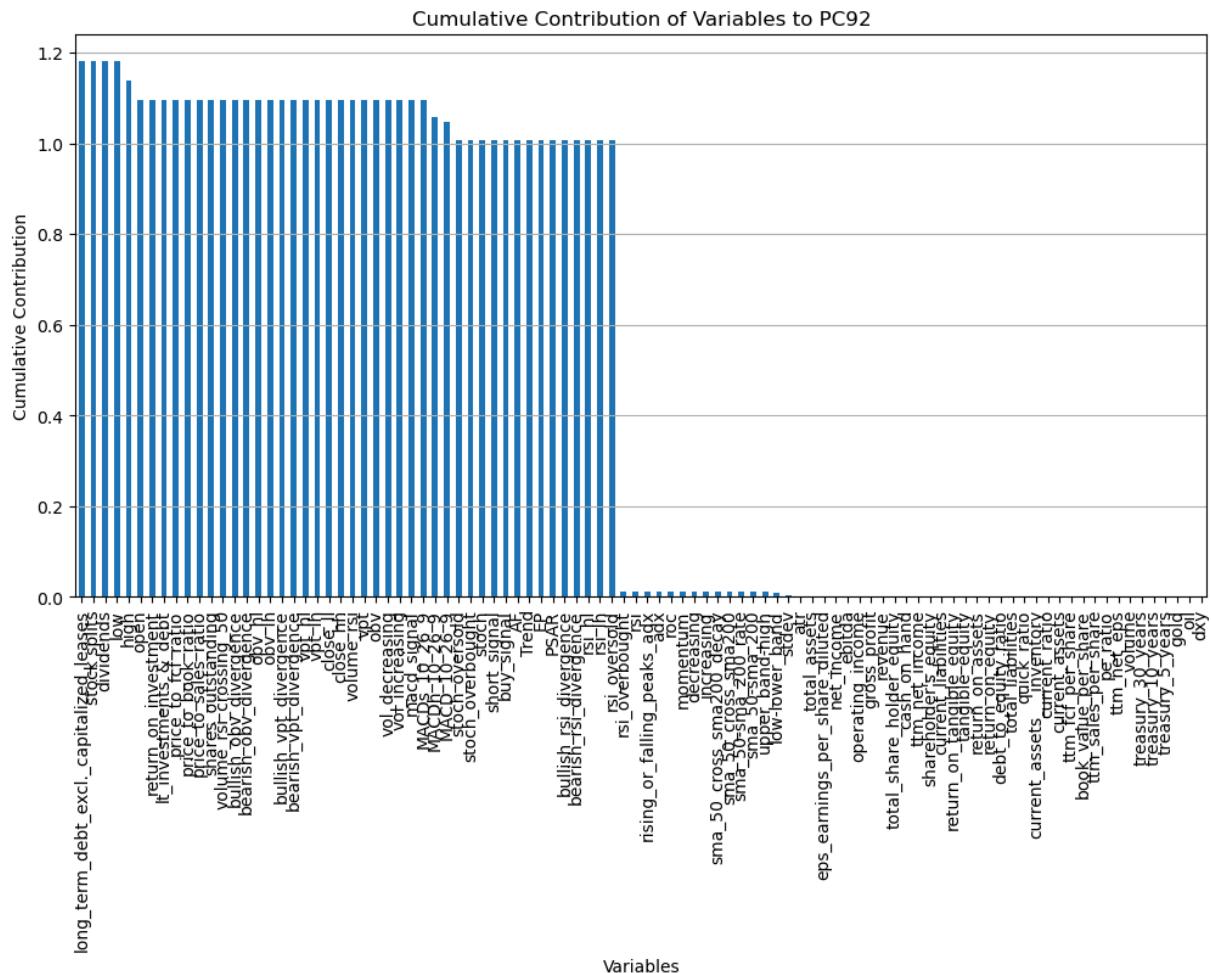


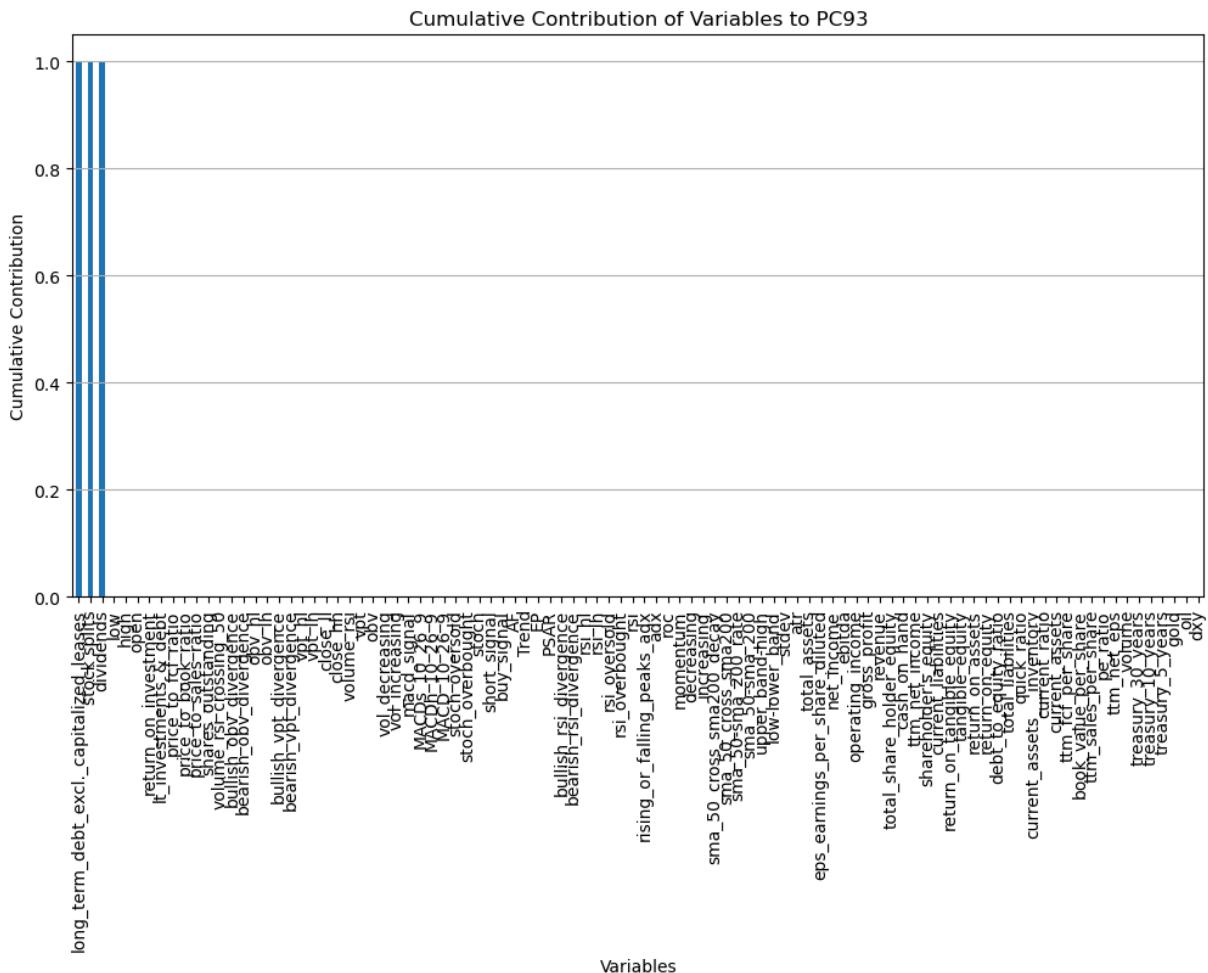


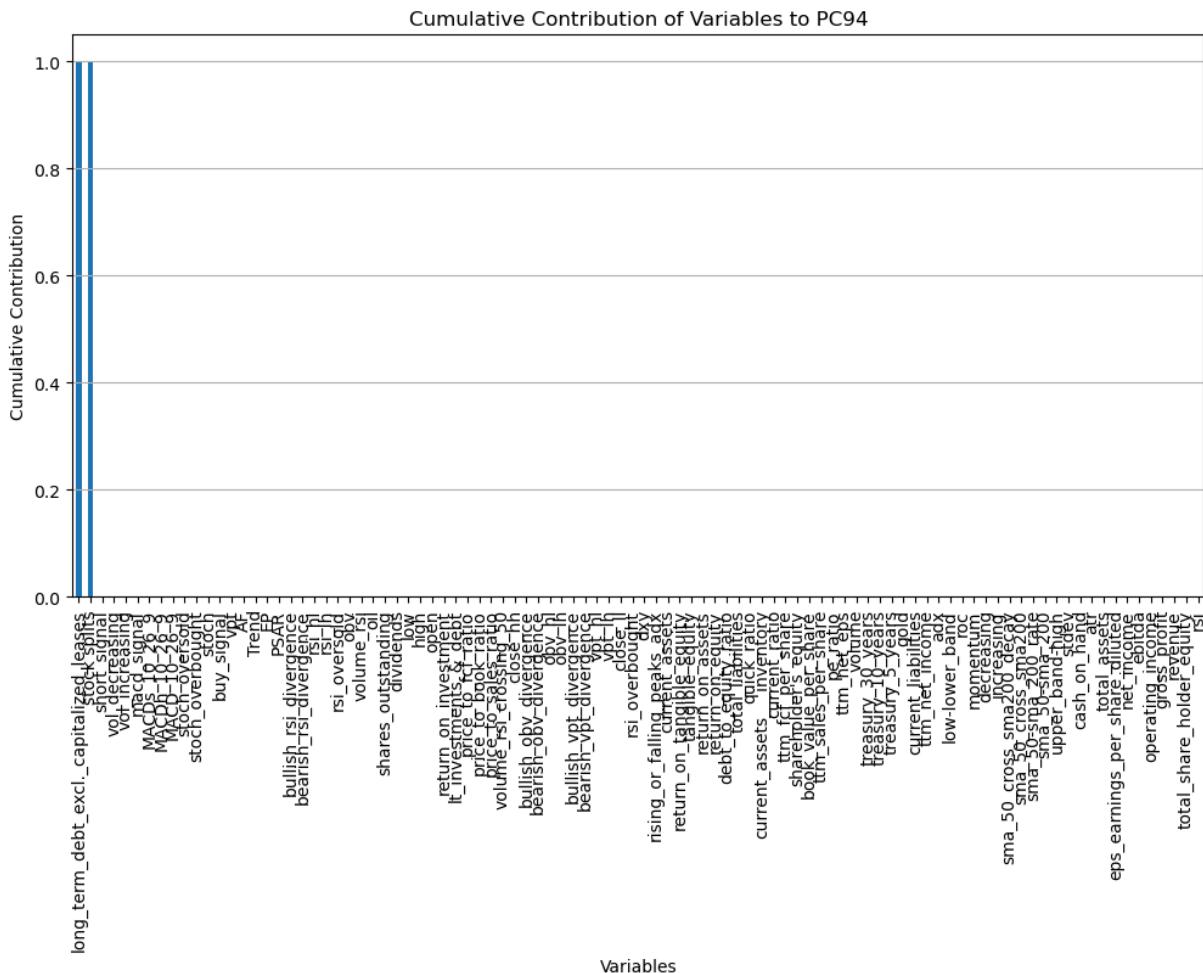


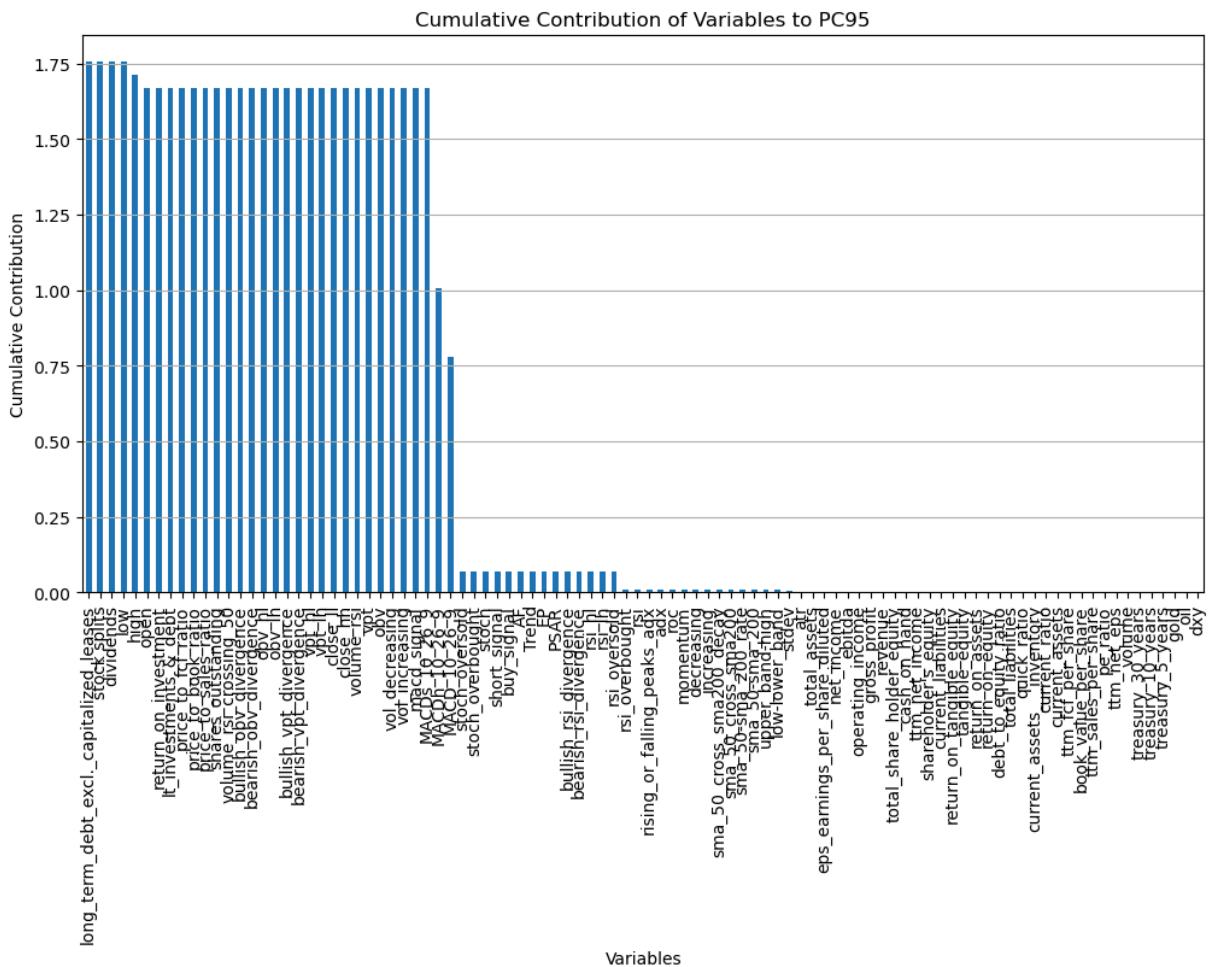


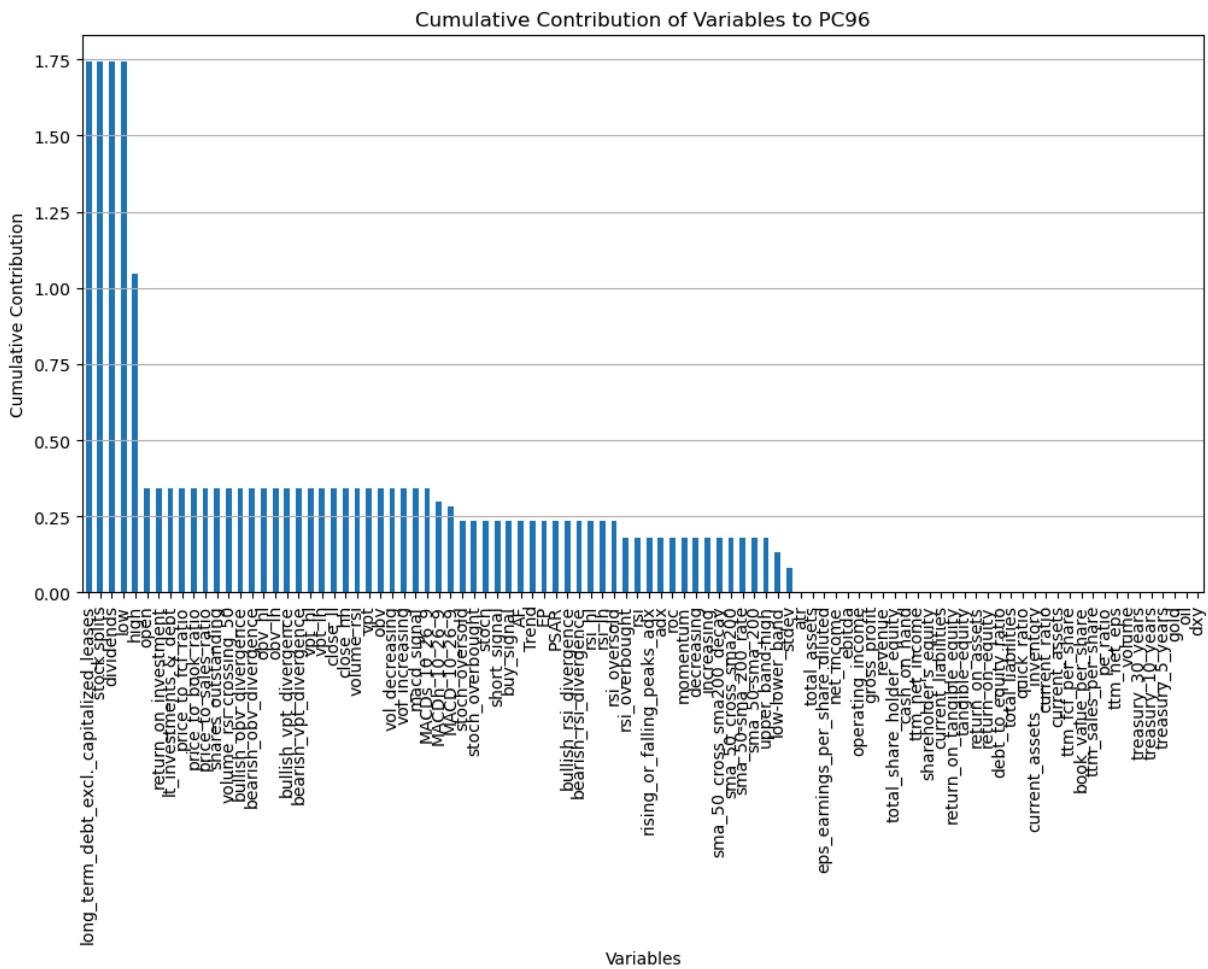












## Gradient Boosting

## Transformed Response

```
In [23]: from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import pandas as pd
import matplotlib.pyplot as plt

# Prepare predictors and response
X = numeric_data.drop(columns=['close', 'log_close'], errors='ignore') # Ex
y = train_data['log_close'] # Log-transformed response variable

# Time series split
tscv = TimeSeriesSplit(n_splits=5)

# Initialize list to store MSPE for each split
rmse_list = []

# Iterate through time series splits and fit Gradient Boosting
for train_index, test_index in tscv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
```

```
# Train Gradient Boosting
gb_model = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb_model.fit(X_train, y_train)

# Predict on the test set
y_pred = gb_model.predict(X_test)

# Calculate MSPE
rmse = mean_squared_error(y_test, y_pred, squared=False)
rmse_list.append(rmse)
```

```
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn()
```

In [24]: # Average MSPE across all splits

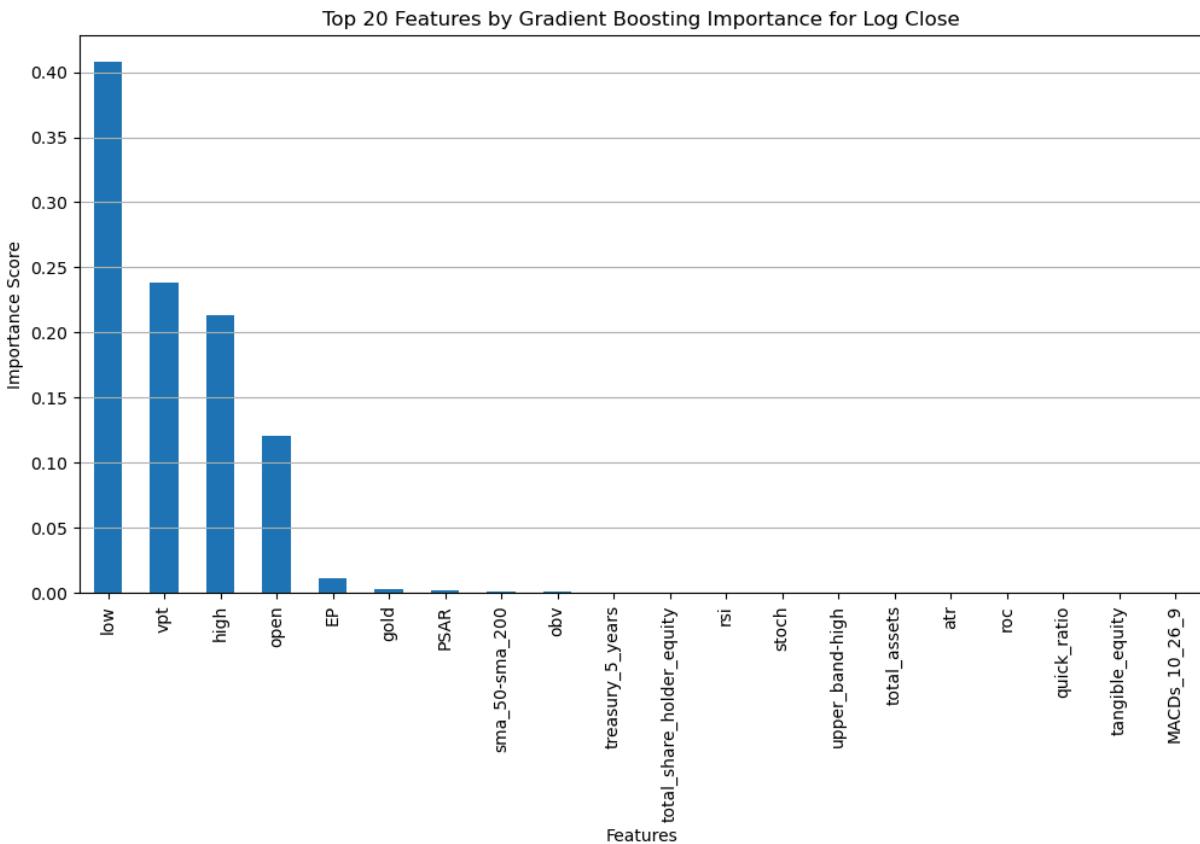
```
average_rmse = sum(rmse_list) / len(rmse_list)
print(f"Gradient Boosting Average RMSE (Log Scale): {average_rmse}")
print(f"Gradient Boosting Average RMSE (Original Scale): {np.exp(average_rmse)}
```

Gradient Boosting Average RMSE (Log Scale): 0.24348653960017835  
 Gradient Boosting Average RMSE (Original Scale): 1.2756891464391382

In [25]: # Feature Importance

```
gb_feature_importance = pd.Series(gb_model.feature_importances_, index=X.columns)
gb_feature_importance.sort_values(ascending=False, inplace=True)

# Plot feature importance
gb_feature_importance.head(20).plot(kind='bar', figsize=(12, 6), title="Top Features")
plt.xlabel('Features')
plt.ylabel('Importance Score')
plt.grid(axis='y')
plt.show()
```



## Normal Close Values

```
In [26]: from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import pandas as pd
import matplotlib.pyplot as plt

# Prepare predictors and response
X = numeric_data.drop(columns=['close', 'log_close'], errors='ignore') # Ex
y = train_data['close'] # Log-transformed response variable

# Time series split
tscv = TimeSeriesSplit(n_splits=5)

# Initialize list to store MSPE for each split
rmse_list = []

# Iterate through time series splits and fit Gradient Boosting
for train_index, test_index in tscv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train Gradient Boosting
    gb_model = GradientBoostingRegressor(n_estimators=100, random_state=42)
    gb_model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = gb_model.predict(X_test)
```

```
# Calculate MSPE
rmse = mean_squared_error(y_test, y_pred, squared=False)
rmse_list.append(rmse)

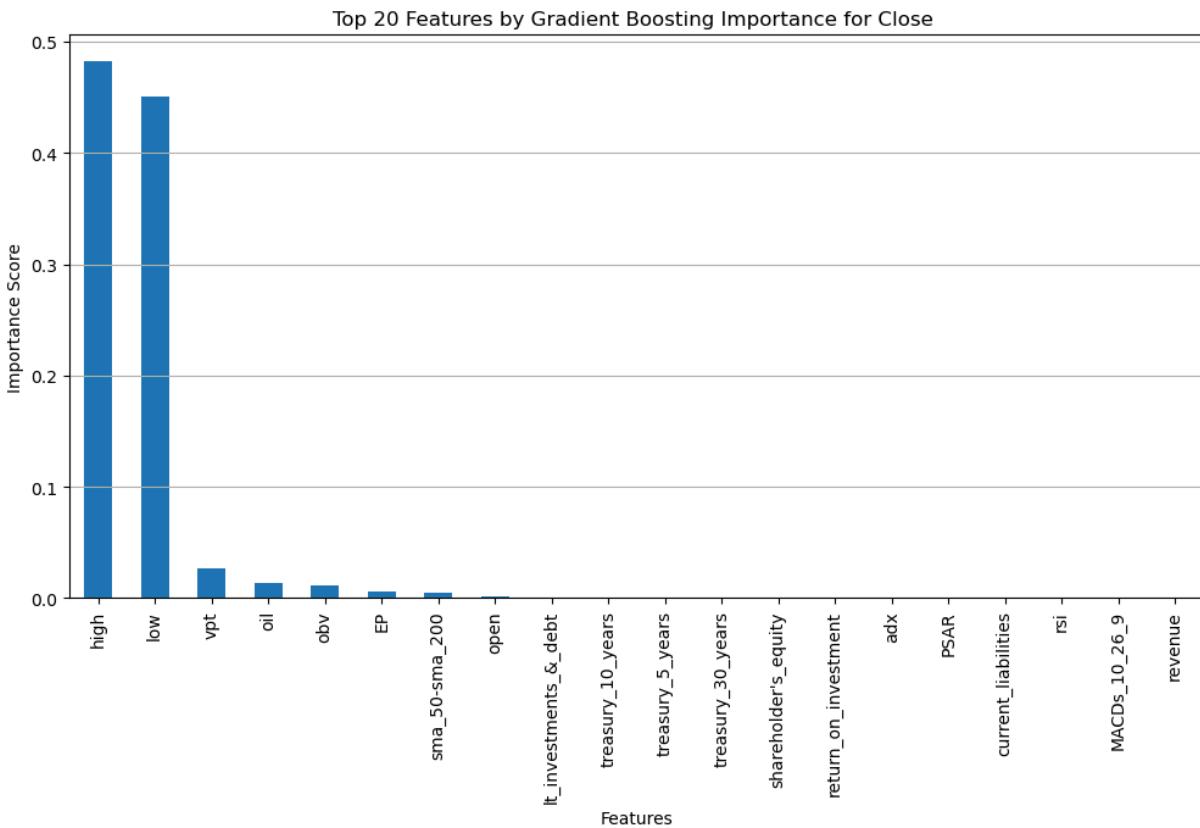
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn()
```

In [27]: # Average RMSE across all splits  
average\_rmse = sum(rmse\_list) / len(rmse\_list)  
print(f"Gradient Boosting Average RMSE: {average\_rmse}")

Gradient Boosting Average RMSE: 9.650645718052829

In [28]: # Feature Importance  
gb\_feature\_importance = pd.Series(gb\_model.feature\_importances\_, index=X.columns)  
gb\_feature\_importance.sort\_values(ascending=False, inplace=True)

# Plot feature importance
gb\_feature\_importance.head(20).plot(kind='bar', figsize=(12, 6), title="Top Features")
plt.xlabel('Features')
plt.ylabel('Importance Score')
plt.grid(axis='y')
plt.show()



## Random Forests

### Transformed Response

```
In [29]: from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import pandas as pd
import matplotlib.pyplot as plt

# Prepare predictors and response
X = numeric_data.drop(columns=['close', 'log_close'], errors='ignore') # Ex
y = train_data['log_close'] # Log-transformed response variable

# Time series split
tscv = TimeSeriesSplit(n_splits=5)

# Initialize list to store MSPE for each split
rmse_list = []

# Iterate through time series splits and fit Random Forest
for train_index, test_index in tscv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train Random Forest
    rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
    rf_model.fit(X_train, y_train)
```

```
# Predict on the test set
y_pred = rf_model.predict(X_test)

# Calculate MSPE (fixed for FutureWarning)
rmse = mean_squared_error(y_test, y_pred, squared=False)
rmse_list.append(rmse)
```

```
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn()
```

In [30]: # Average MSPE across all splits

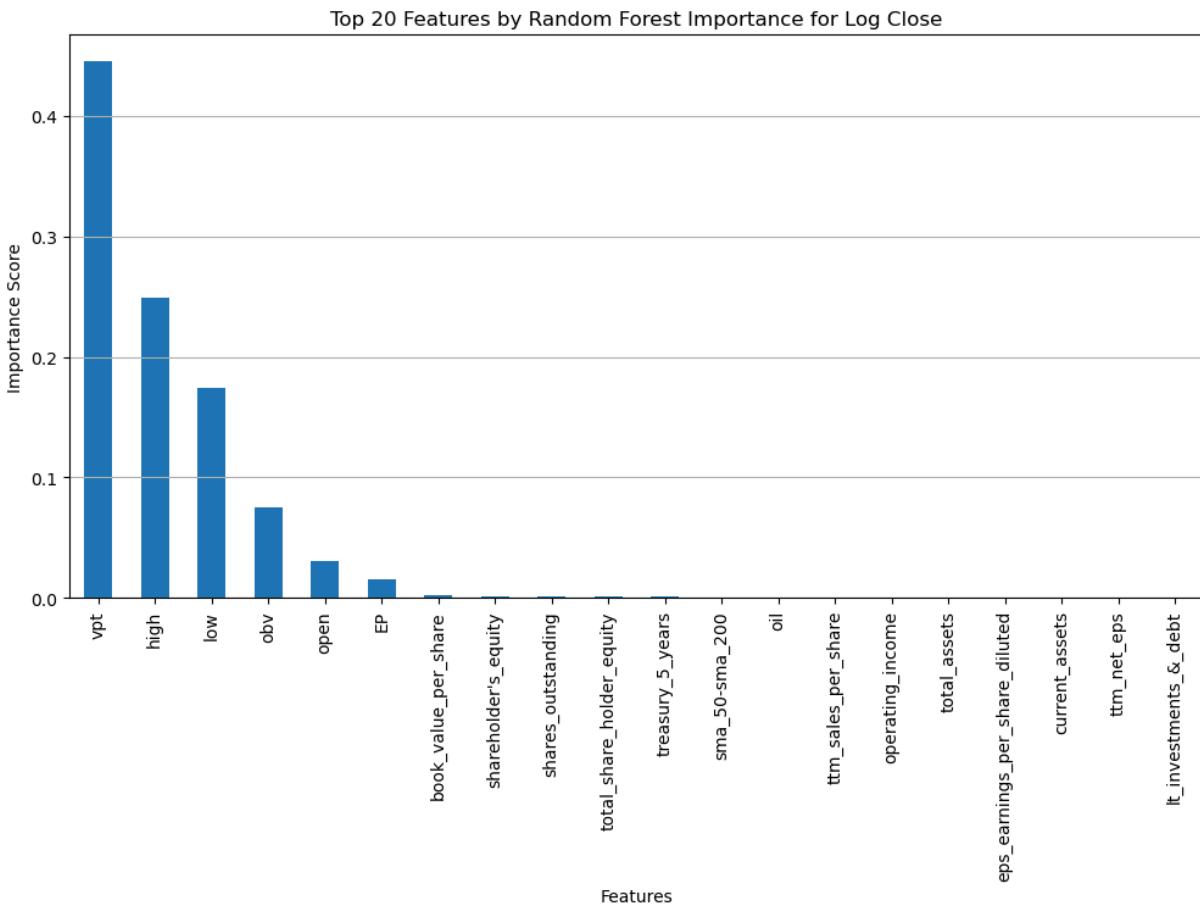
```
average_rmse = sum(rmse_list) / len(rmse_list)
print(f"Random Forest Average RMSE (Log Scale): {average_rmse}")
print(f"Random Forest Average RMSE (Original Scale): {np.exp(average_rmse)}")
```

Random Forest Average RMSE (Log Scale): 0.2407474488995537  
 Random Forest Average RMSE (Original Scale): 1.2721996992990103

In [31]: # Feature Importance

```
rf_feature_importance = pd.Series(rf_model.feature_importances_, index=X.columns)
rf_feature_importance.sort_values(ascending=False, inplace=True)

# Plot feature importance
rf_feature_importance.head(20).plot(kind='bar', figsize=(12, 6), title="Top Features")
plt.xlabel('Features')
plt.ylabel('Importance Score')
plt.grid(axis='y')
plt.show()
```



## Normal Close Values

```
In [32]: from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import pandas as pd
import matplotlib.pyplot as plt

# Prepare predictors and response
X = numeric_data.drop(columns=['close', 'log_close'], errors='ignore') # Exclude log-transformed target variable
y = train_data['close'] # Log-transformed response variable

# Time series split
tscv = TimeSeriesSplit(n_splits=5)

# Initialize list to store MSPE for each split
rmse_list = []

# Iterate through time series splits and fit Random Forest
for train_index, test_index in tscv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train Random Forest
    rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
    rf_model.fit(X_train, y_train)
```

```
# Predict on the test set
y_pred = rf_model.predict(X_test)

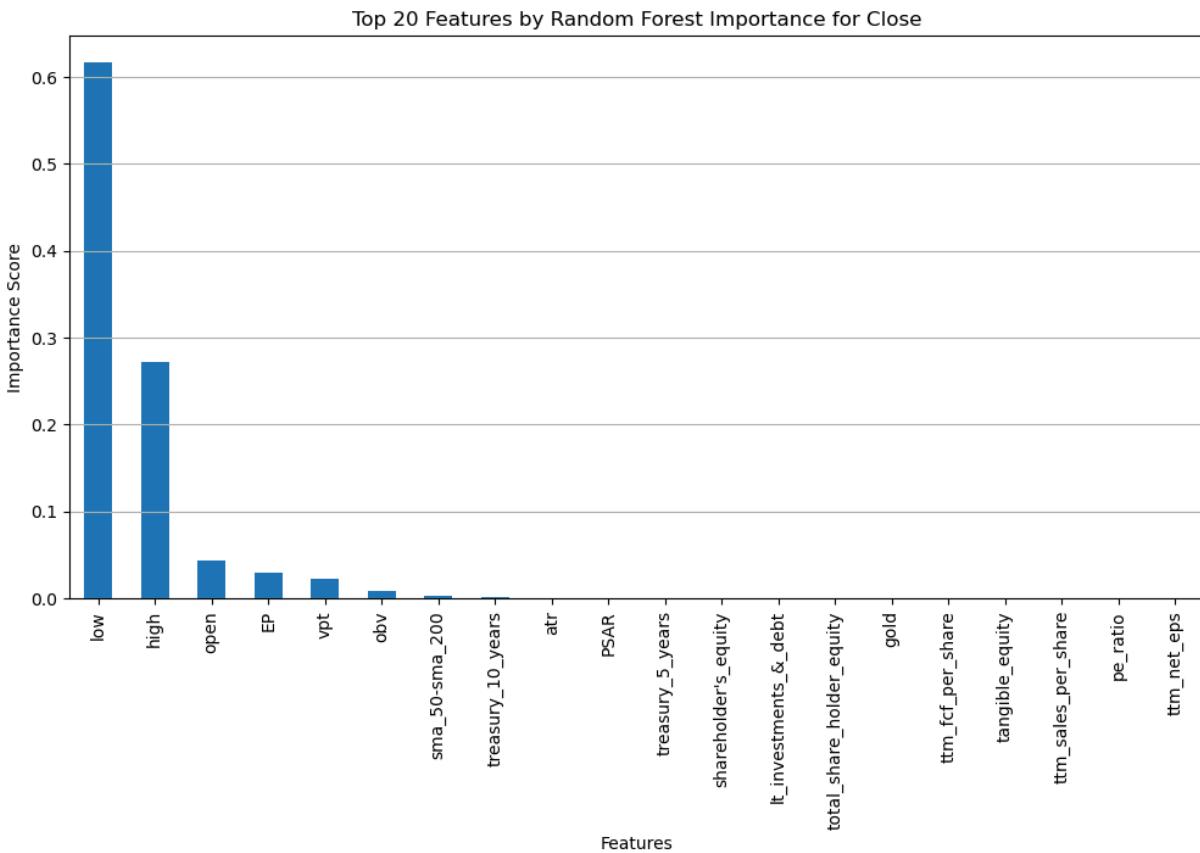
# Calculate MSPE (fixed for FutureWarning)
rmse = mean_squared_error(y_test, y_pred, squared=False)
rmse_list.append(rmse)
```

```
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn()
```

In [33]: # Average MSPE across all splits  
`average_rmse = sum(rmse_list) / len(rmse_list)  
print(f"Random Forest Average RMSE for Close: {average_rmse}")`

Random Forest Average RMSE for Close: 9.562276880884301

In [34]: # Feature Importance  
`rf_feature_importance = pd.Series(rf_model.feature_importances_, index=X.columns)  
rf_feature_importance.sort_values(ascending=False, inplace=True)`  
  
# Plot feature importance  
`rf_feature_importance.head(20).plot(kind='bar', figsize=(12, 6), title="Top Features")  
plt.xlabel('Features')  
plt.ylabel('Importance Score')  
plt.grid(axis='y')  
plt.show()`



## Classification Trees

```
In [35]: from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import accuracy_score
import pandas as pd
import matplotlib.pyplot as plt

# Prepare data for classification (example: predict if 'close' is increasing
train_data['Direction'] = (train_data['close'].diff() > 0).astype(int) # 1

X = numeric_data # Predictors
y = train_data['Direction'] # Binary target variable

# Time series cross-validation
tscv = TimeSeriesSplit(n_splits=5)

# Store feature importance and accuracy for each fold
feature_importances = []
accuracies = []

for train_index, test_index in tscv.split(X):
    # Split data while respecting time order
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train a classification tree
    clf_tree = DecisionTreeClassifier(random_state=42, max_depth=5)
```

```

clf_tree.fit(X_train, y_train)

# Extract and store feature importance
tree_feature_importance = pd.Series(clf_tree.feature_importances_, index=feature_importances.append(tree_feature_importance))

# Evaluate accuracy on the test set
y_pred = clf_tree.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
accuracies.append(accuracy)

# Average feature importance across folds
average_importance = pd.concat(feature_importances, axis=1).mean(axis=1)
average_importance.sort_values(ascending=False, inplace=True)

# Plot the average feature importance
average_importance.head(20).plot(kind='bar', figsize=(12, 6), title="Average Classification Tree Feature Importance (Time Series CV)")
plt.xlabel('Features')
plt.ylabel('Average Importance Score')
plt.grid(axis='y')
plt.show()

```

/var/folders/2p/hbz8h54x5hj828cdgg7jshz0000gn/T/ipykernel\_66719/4217483650.py:8: SettingWithCopyWarning:

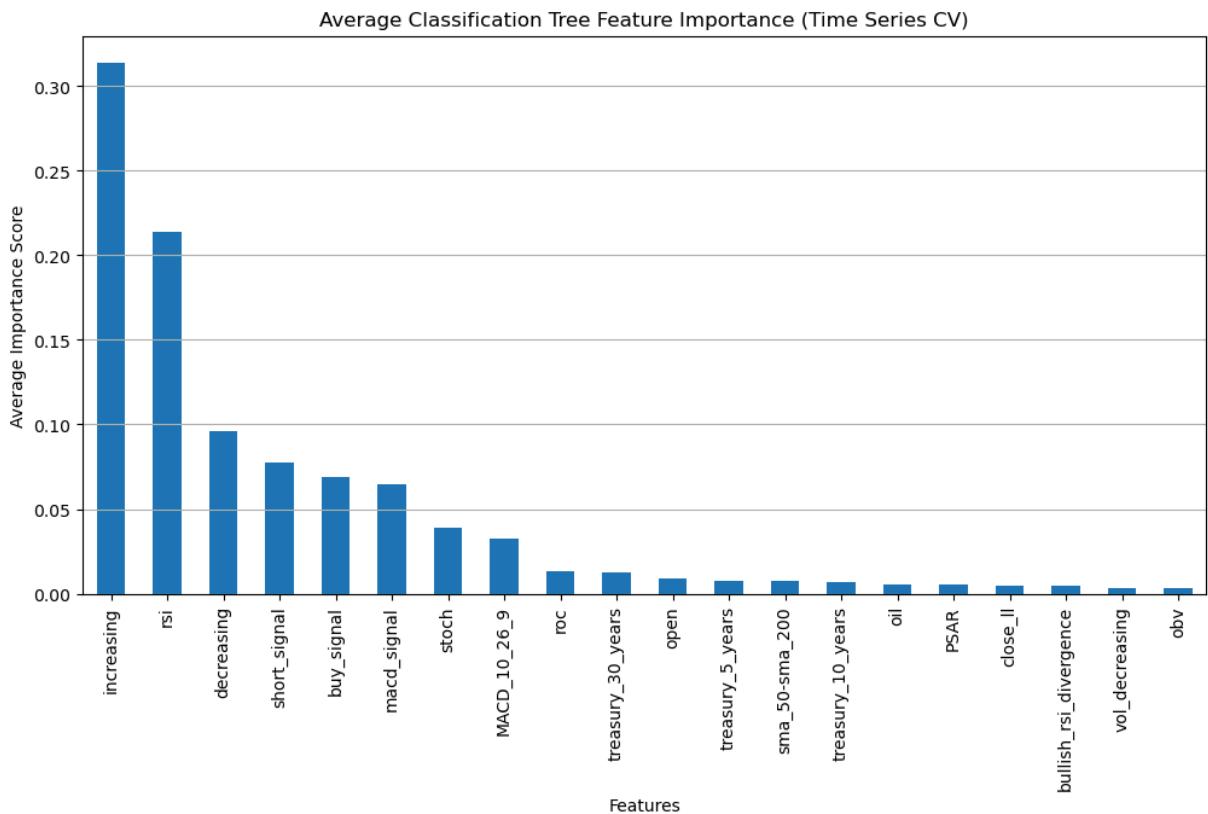
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```

train_data['Direction'] = (train_data['close'].diff() > 0).astype(int) # 1 for increase, 0 for decrease

```



```
In [36]: # Print average feature importance values
print("Average Classification Tree Feature Importance (Time Series CV):")
print(average_importance)
```

```
Average Classification Tree Feature Importance (Time Series CV):
increasing          0.314040
rsi                 0.213709
decreasing          0.096331
short_signal         0.077550
buy_signal           0.069268
...
net_income           0.000000
ebitda               0.000000
operating_income     0.000000
gross_profit         0.000000
long_term_debt_excl._capitalized_leases 0.000000
Length: 96, dtype: float64
```

```
In [37]: # Print average accuracy across folds
print(f"Average Accuracy Across Folds: {sum(accuracies) / len(accuracies):.2f}")
```

Average Accuracy Across Folds: 0.61

## Regularization-Based Methods

### Log Transformed Response

```
In [38]: from sklearn.linear_model import RidgeCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Prepare data
X = numeric_data # Predictors
y = train_data['log_close'] # Response variable

# Standardize predictors
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define time series split
tscv = TimeSeriesSplit(n_splits=5)

# Ridge Regression with Time Series CV
ridge_model = RidgeCV(alphas=[0.1, 1.0, 10.0, 100.0], cv=tscv)
ridge_model.fit(X_scaled, y)

# Predict on entire data (as RidgeCV optimizes internally)
y_pred = ridge_model.predict(X_scaled)

# Calculate MSPE on a manual test split for consistency
rmse_list = []
```

```

for train_index, test_index in tscv.split(X_scaled):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Use the pre-fitted ridge_model to predict
    y_pred = ridge_model.predict(X_test)

    # Calculate MSPE for the current fold
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    rmse_list.append(rmse)

```

```

/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function 'root_mean_squared_error'.
    warnings.warn(

```

In [39]:

```

# Average MSPE across all splits
average_rmse = sum(rmse_list) / len(rmse_list)
print(f'Ridge Regression Average RMSE (Log Scale): {average_rmse}')
print(f'Ridge Regression Average RMSE (Original Scale): {np.exp(average_rmse)}

```

Ridge Regression Average RMSE (Log Scale): 0.04601900778501562  
Ridge Regression Average RMSE (Original Scale): 1.047094313711389

In [40]:

```

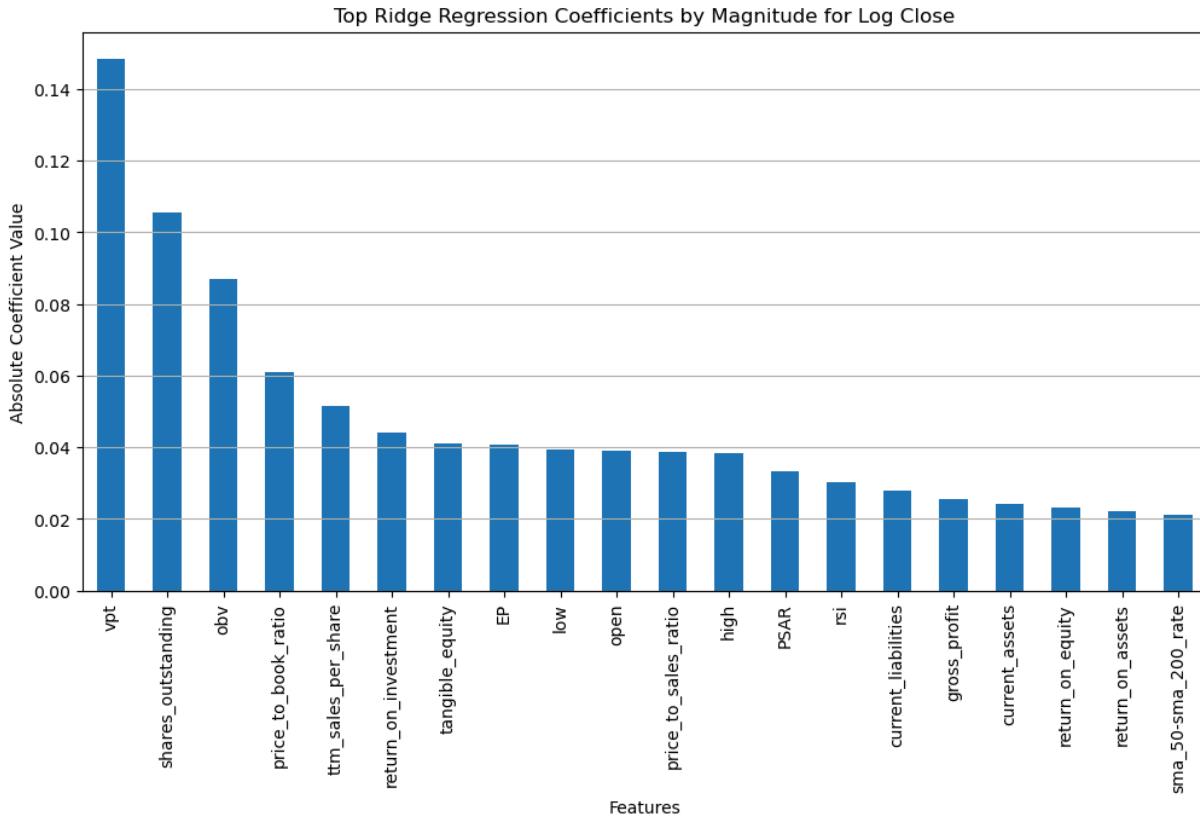
# Extract and sort coefficients by magnitude
ridge_coefficients = pd.Series(np.abs(ridge_model.coef_), index=X.columns)
ridge_coefficients.sort_values(ascending=False, inplace=True)

# Plot top coefficients
ridge_coefficients.head(20).plot(kind='bar', figsize=(12, 6), title="Top Ridge Coefficients")
plt.xlabel('Features')
plt.ylabel('Absolute Coefficient Value')
plt.grid(axis='y')
plt.show()

# Print all coefficients sorted by magnitude

```

```
print("Ridge Regression Coefficients by Magnitude for Log Close:")
print(ridge_coefficients)
```



Ridge Regression Coefficients by Magnitude for Log Close:

```
vpt          0.148525
shares_outstanding  0.105650
obv          0.087053
price_to_book_ratio  0.060799
ttm_sales_per_share  0.051446
...
vpt_hl        0.000113
sma_50_cross_sma200  0.000072
rsi_oversold    0.000000
dividends      0.000000
stock splits    0.000000
Length: 96, dtype: float64
```

## Normal Close Values

```
In [41]: from sklearn.linear_model import RidgeCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Prepare data
X = numeric_data # Predictors
y = train_data['close'] # Response variable
```

```

# Standardize predictors
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define time series split
tscv = TimeSeriesSplit(n_splits=5)

# Ridge Regression with Time Series CV
ridge_model = RidgeCV(alphas=[0.1, 1.0, 10.0, 100.0], cv=tscv)
ridge_model.fit(X_scaled, y)

# Predict on entire data (as RidgeCV optimizes internally)
y_pred = ridge_model.predict(X_scaled)

# Calculate MSPE on a manual test split for consistency
rmse_list = []
for train_index, test_index in tscv.split(X_scaled):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Use the pre-fitted ridge_model to predict
    y_pred = ridge_model.predict(X_test)

    # Calculate MSPE for the current fold
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    rmse_list.append(rmse)

```

```

/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(

```

In [42]: # Average MSPE across all splits

```

average_rmse = sum(rmse_list) / len(rmse_list)
print(f"Ridge Regression Average MSPE for Close: {average_rmse}")

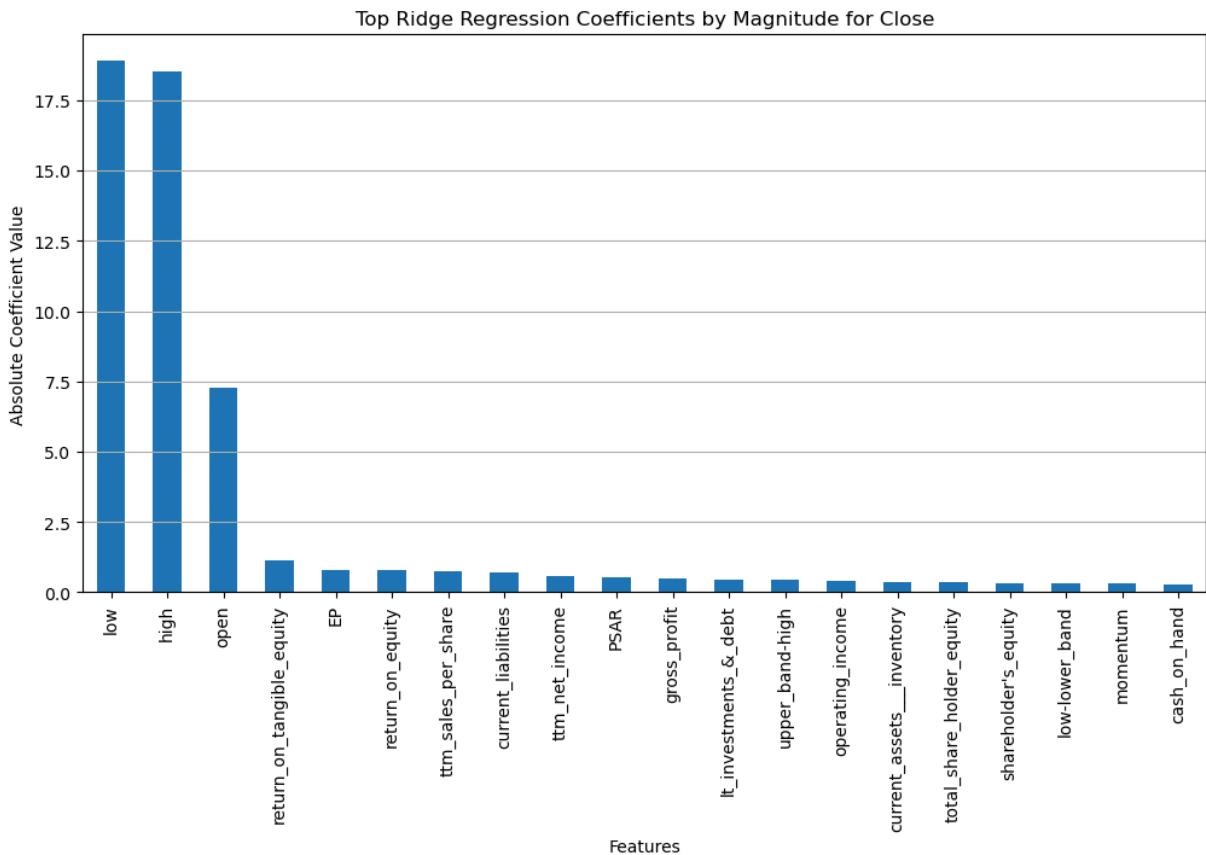
```

Ridge Regression Average MSPE for Close: 0.21423664144996674

```
In [43]: # Extract and sort coefficients by magnitude
ridge_coefficients = pd.Series(np.abs(ridge_model.coef_), index=X.columns)
ridge_coefficients.sort_values(ascending=False, inplace=True)

# Plot top coefficients
ridge_coefficients.head(20).plot(kind='bar', figsize=(12, 6), title="Top Ridge Regression Coefficients by Magnitude")
plt.xlabel('Features')
plt.ylabel('Absolute Coefficient Value')
plt.grid(axis='y')
plt.show()

# Print all coefficients sorted by magnitude
print("Ridge Regression Coefficients by Magnitude:")
print(ridge_coefficients)
```



Ridge Regression Coefficients by Magnitude:

low	18.890171
high	18.511902
open	7.289454
return_on_tangible_equity	1.124937
EP	0.797100
...	...
bullish_rsi_divergence	0.000185
vpt_hl	0.000128
dividends	0.000000
stock_splits	0.000000
rsi_oversold	0.000000

Length: 96, dtype: float64

## Log Transformed Response

In [44]:

```
from sklearn.linear_model import LassoCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Prepare data
X = numeric_data # Predictors
y = train_data['log_close'] # Response variable

# Standardize predictors
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define time series split
tscv = TimeSeriesSplit(n_splits=5)

# Lasso Regression with Time Series CV
lasso_model = LassoCV(alphas=np.logspace(-4, 0, 50), cv=tscv, max_iter=10000)
lasso_model.fit(X_scaled, y)

# Predict on entire dataset (fitted on time series CV)
y_pred = lasso_model.predict(X_scaled)

# Calculate MSPE on manual splits
rmse_list = []
for train_index, test_index in tscv.split(X_scaled):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Use pre-fitted LassoCV model to predict
    y_pred = lasso_model.predict(X_test)

    # Calculate MSPE for current fold
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    rmse_list.append(rmse)
```

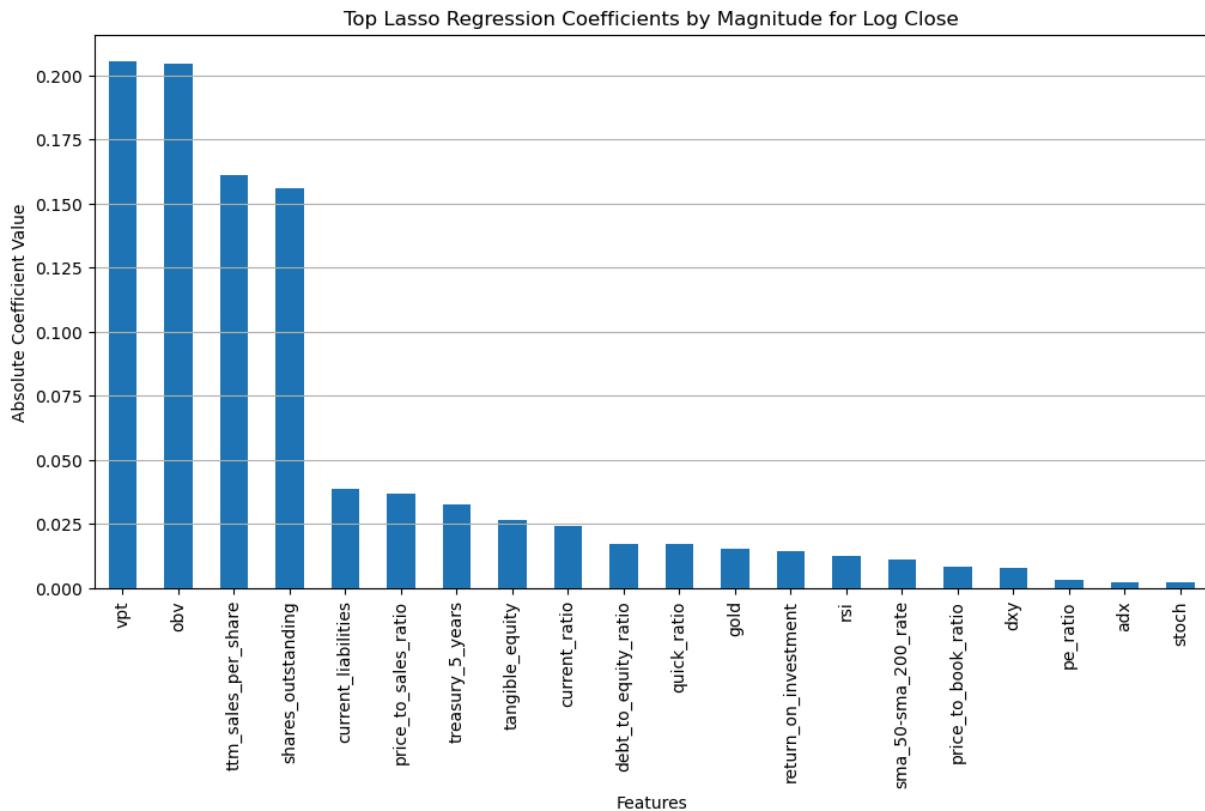
```
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
```

```
In [45]: # Average MSPE across all splits  
average_rmse = sum(rmse_list) / len(rmse_list)  
print(f'Lasso Regression Average RMSE (Log Scale): {average_rmse}')  
print(f'Lasso Regression Average RMSE (Original Scale): {np.exp(average_rmse)}
```

Lasso Regression Average RMSE (Log Scale): 0.041607199963150494  
Lasso Regression Average RMSE (Original Scale): 1.0424849102056157

```
In [46]: # Extract and sort coefficients by magnitude
lasso_coefficients = pd.Series(np.abs(lasso_model.coef_), index=X.columns)
lasso_coefficients.sort_values(ascending=False, inplace=True)

# Plot top coefficients
lasso_coefficients.head(20).plot(kind='bar', figsize=(12, 6), title="Top Lasso Coefficients")
plt.xlabel('Features')
plt.ylabel('Absolute Coefficient Value')
plt.grid(axis='y')
plt.show()
```



```
In [47]: # Print all coefficients sorted by magnitude
print("Lasso Regression Coefficients by Magnitude for Log Close:")
print(lasso_coefficients)
```

Lasso Regression Coefficients by Magnitude for Log Close:

vpt	0.205567
obv	0.204481
ttm_sales_per_share	0.160890
shares_outstanding	0.156044
current_liabilities	0.038405
...	...
sma_50_cross_sma200_decay	0.000000
sma_50_cross_sma200	0.000000
ttm_fcf_per_share	0.000000
sma_50-sma_200	0.000000
long_term_debt_excl._capitalized_leases	0.000000

Length: 96, dtype: float64

```
In [48]: # Identify and print features removed by Lasso (coefficients = 0)
lasso_removed_features = lasso_coefficients[lasso_coefficients == 0]
print("\nFeatures Removed by Lasso (Coefficients = 0):")
print(lasso_removed_features)
```

```
Features Removed by Lasso (Coefficients = 0):
total_liabilities           0.0
volume_rsi                   0.0
book_value_per_share         0.0
vol_decreasing                0.0
macd_signal                   0.0
...
sma_50_cross_sma200_decay    0.0
sma_50_cross_sma200          0.0
ttm_fcf_per_share            0.0
sma_50-sma_200                0.0
long_term_debt_excl._capitalized_leases 0.0
Length: 70, dtype: float64
```

```
In [49]: lasso_selected_features = lasso_coefficients[lasso_coefficients > 0]
print("\nFeatures kept by Lasso:")
print(lasso_selected_features)
```

```
Features kept by Lasso:
vpt                           0.205567
obv                           0.204481
ttm_sales_per_share           0.160890
shares_outstanding             0.156044
current_liabilities           0.038405
price_to_sales_ratio          0.036943
treasury_5_years               0.032723
tangible_equity                 0.026401
current_ratio                  0.023902
debt_to_equity_ratio           0.017276
quick_ratio                     0.017130
gold                            0.015095
return_on_investment           0.014134
rsi                             0.012323
sma_50-sma_200_rate            0.010850
price_to_book_ratio             0.008146
dxy                            0.007581
pe_ratio                        0.003136
adx                            0.002125
stoch                           0.002051
MACDh_10_26_9                  0.001089
volume                          0.000909
stoch_oversold                  0.000796
rsi_overbought                  0.000648
decreasing                       0.000190
bullish_obv_divergence          0.000174
dtype: float64
```

## Normal Close Values

```
In [50]: from sklearn.linear_model import LassoCV
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

```
# Prepare data
X = numeric_data # Predictors
y = train_data['close'] # Response variable

# Standardize predictors
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define time series split
tscv = TimeSeriesSplit(n_splits=5)

# Lasso Regression with Time Series CV
lasso_model = LassoCV(alphas=np.logspace(-4, 0, 50), cv=tscv, max_iter=10000)
lasso_model.fit(X_scaled, y)

# Predict on entire dataset (fitted on time series CV)
y_pred = lasso_model.predict(X_scaled)

# Calculate MSPE on manual splits
rmse_list = []
for train_index, test_index in tscv.split(X_scaled):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Use pre-fitted LassoCV model to predict
    y_pred = lasso_model.predict(X_test)

    # Calculate MSPE for current fold
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    rmse_list.append(rmse)
```

```

/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(

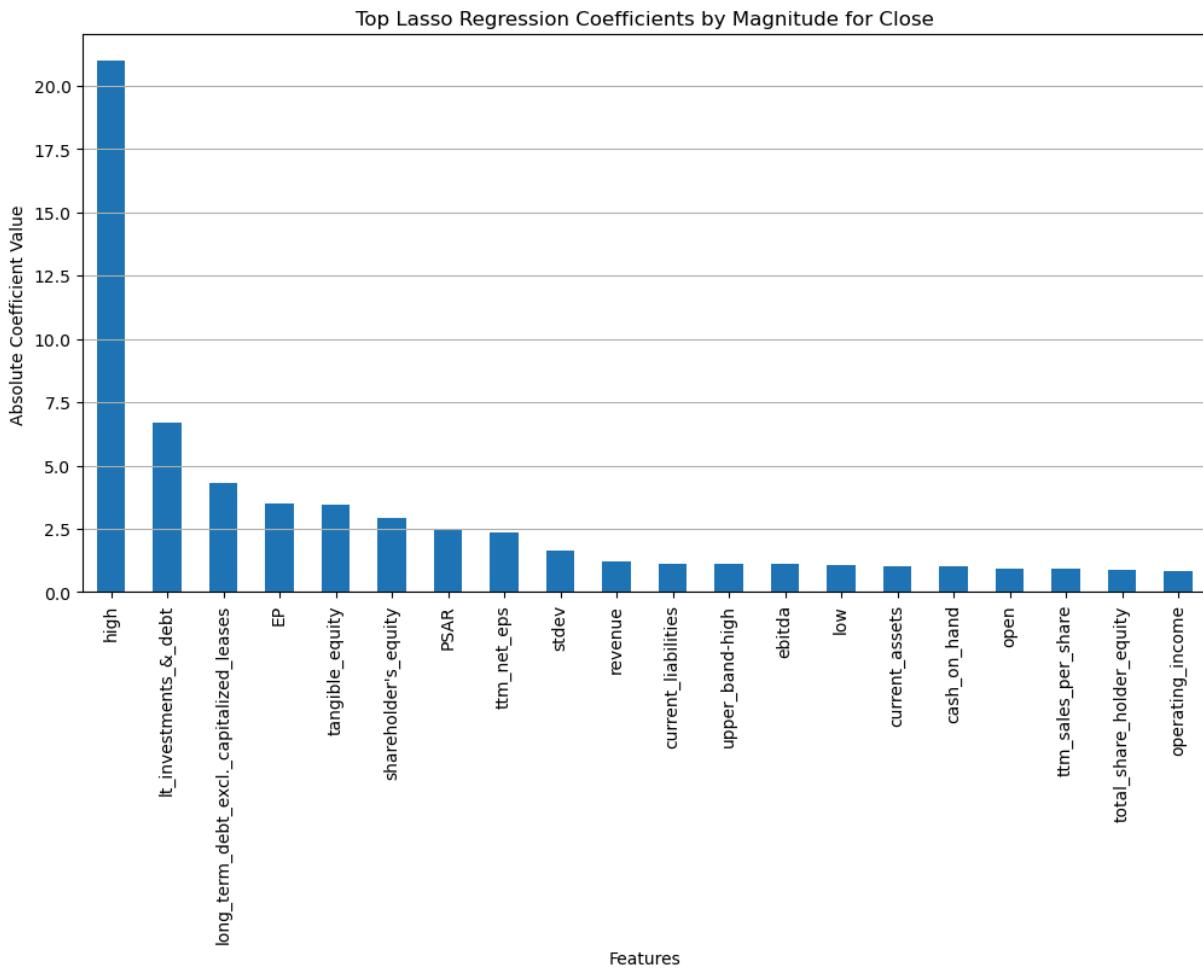
```

In [51]: # Average MSPE across all splits  
`average_rmse = sum(rmse_list) / len(rmse_list)`  
`print(f'Lasso Regression Average RMSE for Close: {average_rmse}')`

Lasso Regression Average RMSE for Close: 0.3056940968266856

In [52]: # Extract and sort coefficients by magnitude  
`lasso_coefficients = pd.Series(np.abs(lasso_model.coef_), index=X.columns)`  
`lasso_coefficients.sort_values(ascending=False, inplace=True)`

# Plot top coefficients  
`lasso_coefficients.head(20).plot(kind='bar', figsize=(12, 6), title="Top Lasso Coefficients")`  
`plt.xlabel('Features')`  
`plt.ylabel('Absolute Coefficient Value')`  
`plt.grid(axis='y')`  
`plt.show()`



```
In [53]: # Print all coefficients sorted by magnitude
print("Lasso Regression Coefficients by Magnitude for Close:")
print(lasso_coefficients)
```

Lasso Regression Coefficients by Magnitude for Close:

high	20.963839
lt_investments_&_debt	6.678839
long_term_debt_excl._capitalized_leases	4.310645
EP	3.503863
tangible_equity	3.456291
...	
sma_50-sma_200	0.000319
rising_or_falling_peaks_adx	0.000014
rsi_oversold	0.000000
dividends	0.000000
stock splits	0.000000

Length: 96, dtype: float64

```
In [54]: # Identify and print features removed by Lasso (coefficients = 0)
lasso_removed_features = lasso_coefficients[lasso_coefficients == 0]
print("\nFeatures Removed by Lasso (Coefficients = 0):")
print(lasso_removed_features)
```

```
Features Removed by Lasso (Coefficients = 0):
```

```
rsi_oversold    0.0
dividends      0.0
stock splits    0.0
dtype: float64
```

```
In [55]: lasso_selected_features = lasso_coefficients[lasso_coefficients > 0]
print("\nFeatures kept by Lasso:")
print(lasso_selected_features)
```

```
Features kept by Lasso:
```

high	20.963839
lt_investments_&_debt	6.678839
long_term_debt_excl._capitalized_leases	4.310645
EP	3.503863
tangible_equity	3.456291
	...
obv_hl	0.001783
vpt_hl	0.001708
close_hh	0.001126
sma_50-sma_200	0.000319
rising_or_falling_peaks_adx	0.000014

Length: 93, dtype: float64

## SVMs

### Linear SVM for feature importance

### Log Transformed Response

```
In [56]: from sklearn.svm import LinearSVR
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Prepare data
X = numeric_data # Predictors
y = train_data['log_close'] # Response variable

# Standardize predictors
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Define time series split
tscv = TimeSeriesSplit(n_splits=5)

# Train a Linear SVM regression model on the full dataset for feature import
svr_model = LinearSVR(random_state=42, max_iter=10000, C=0.1) # Use smaller
svr_model.fit(X_scaled, y)

# Extract and sort coefficients by magnitude
```

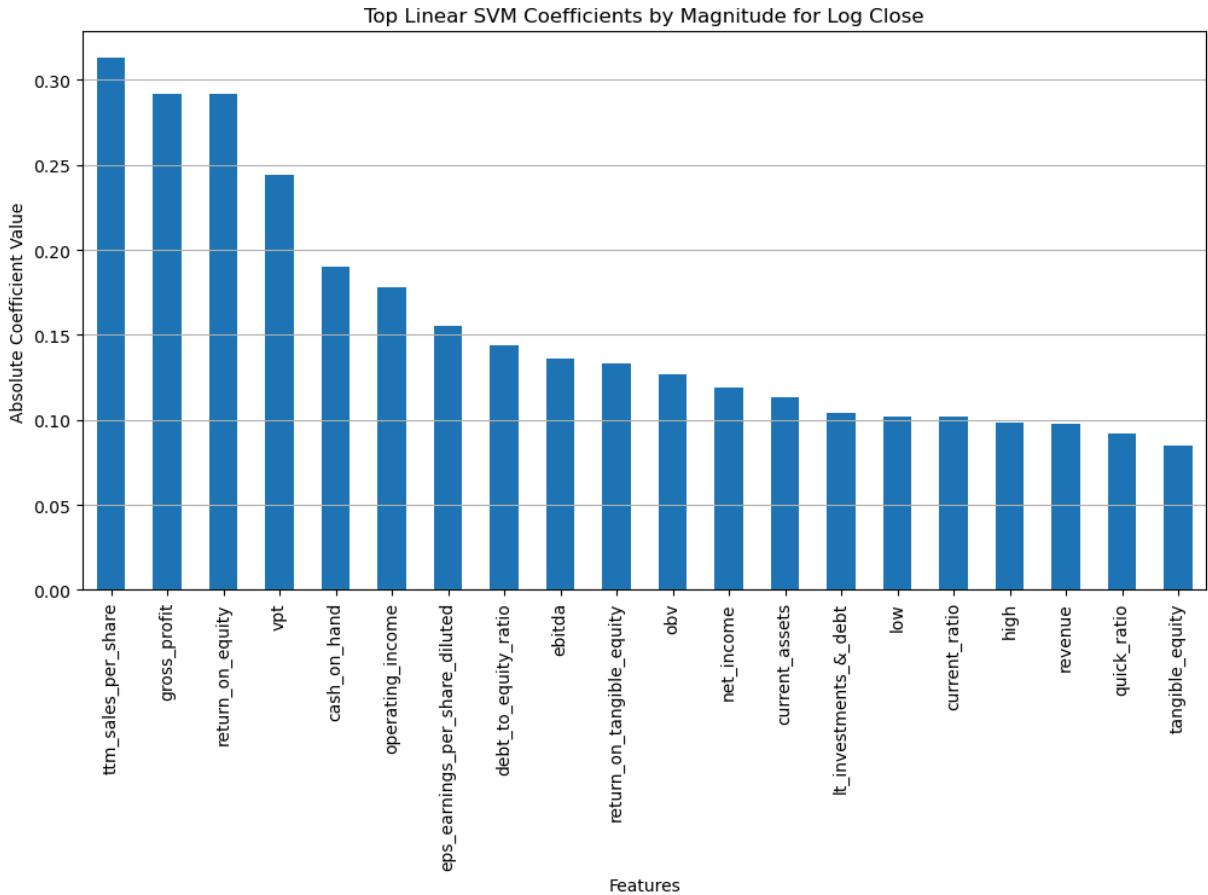
```

svr_feature_importance = pd.Series(np.abs(svr_model.coef_), index=X.columns)
svr_feature_importance.sort_values(ascending=False, inplace=True)

# Plot top coefficients
svr_feature_importance.head(20).plot(kind='bar', figsize=(12, 6), title="Top Linear SVM Coefficients by Magnitude for Log Close")
plt.xlabel('Features')
plt.ylabel('Absolute Coefficient Value')
plt.grid(axis='y')
plt.show()

```

/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/svm/\_base.py:1235: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.  
warnings.warn(



In [57]: # Initialize list to store MSPE for each split  
rmse\_list = []  
  
# Perform time series split and train/test evaluation for MSPE  
for train\_index, test\_index in tscv.split(X\_scaled):  
 X\_train, X\_test = X\_scaled[train\_index], X\_scaled[test\_index]  
 y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]  
  
 # Train a Linear SVM regression model on the current split  
 svr\_model\_split = LinearSVR(random\_state=42, max\_iter=10000, C=0.1)  
 svr\_model\_split.fit(X\_train, y\_train)  
  
 # Predict on the test set  
 y\_pred = svr\_model\_split.predict(X\_test)

```

# Calculate MSPE
rmse = mean_squared_error(y_test, y_pred, squared=False)
rmse_list.append(rmse)

/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/svm/_base.py:1235: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/svm/_base.py:1235: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(

```

In [58]:

```

# Average MSPE across all splits
average_rmse = sum(rmse_list) / len(rmse_list)
print(f"Linear SVM Average RMSE (Log Scale): {average_rmse}")
print(f"Linear SVM Average RMSE (Original Scale): {np.exp(average_rmse)}")

```

Linear SVM Average RMSE (Log Scale): 0.4667938446172881  
 Linear SVM Average RMSE (Original Scale): 1.5948725779569104

In [59]:

```

# Print all coefficients sorted by magnitude
print("Linear SVM Coefficients by Magnitude for Log Close:")

```

```
print(svr_feature_importance)

Linear SVM Coefficients by Magnitude for Log Close:
ttm_sales_per_share      0.312848
gross_profit              0.291984
return_on_equity          0.291503
vpt                        0.243964
cash_on_hand               0.190353
...
vol_increasing            0.000091
macd_signal                0.000010
rsi_oversold               0.000000
dividends                  0.000000
stock splits                0.000000
Length: 96, dtype: float64
```

## Normal Close Values

```
In [60]: from sklearn.svm import LinearSVR
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Prepare data
X = numeric_data # Predictors
y = train_data['close'] # Response variable

# Standardize predictors
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

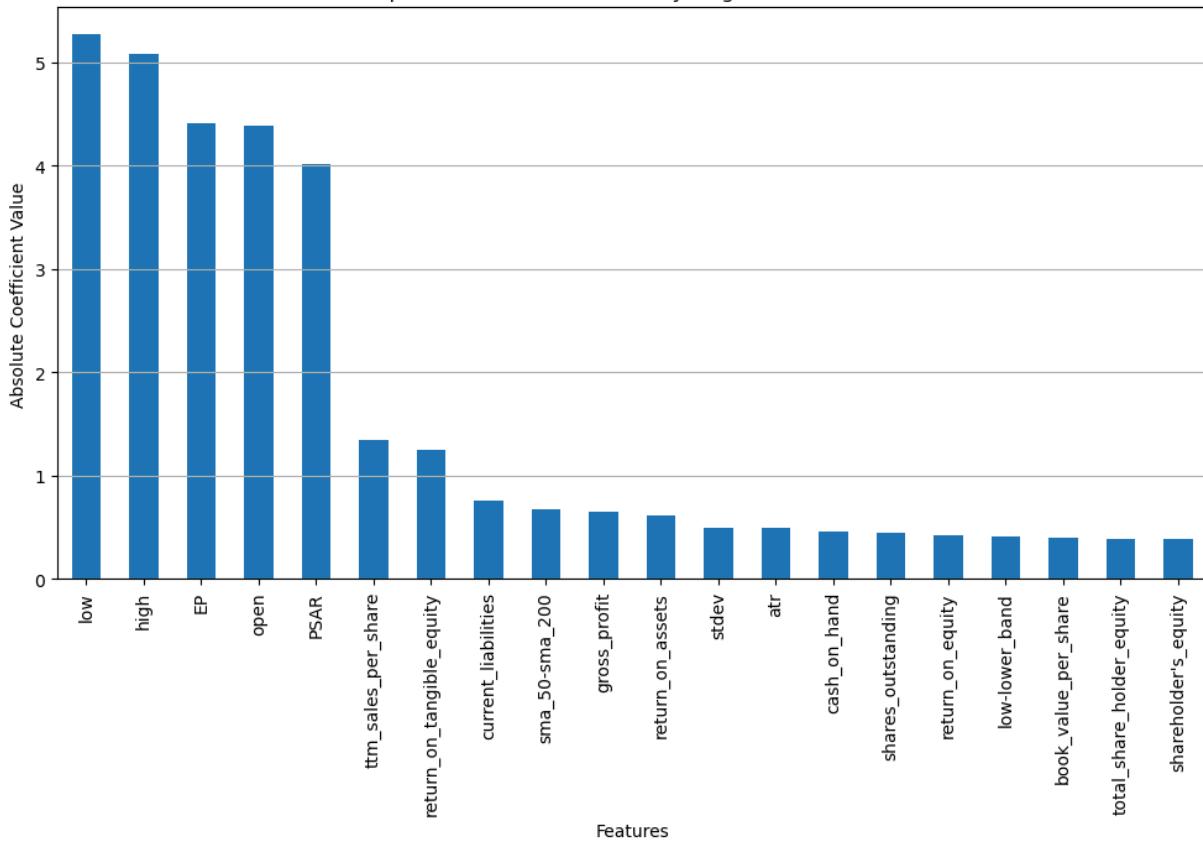
# Define time series split
tscv = TimeSeriesSplit(n_splits=5)

# Train a Linear SVM regression model on the full dataset for feature import
svr_model = LinearSVR(random_state=42, max_iter=10000, C=0.1) # Use smaller
svr_model.fit(X_scaled, y)

# Extract and sort coefficients by magnitude
svr_feature_importance = pd.Series(np.abs(svr_model.coef_), index=X.columns)
svr_feature_importance.sort_values(ascending=False, inplace=True)

# Plot top coefficients
svr_feature_importance.head(20).plot(kind='bar', figsize=(12, 6), title="Top
plt.xlabel('Features')
plt.ylabel('Absolute Coefficient Value')
plt.grid(axis='y')
plt.show()
```

Top Linear SVM Coefficients by Magnitude for Close



```
In [61]: # Initialize list to store MSPE for each split
rmse_list = []

# Perform time series split and train/test evaluation for MSPE
for train_index, test_index in tscv.split(X_scaled):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train a Linear SVM regression model on the current split
    svr_model_split = LinearSVR(random_state=42, max_iter=10000, C=0.1)
    svr_model_split.fit(X_train, y_train)

    # Predict on the test set
    y_pred = svr_model_split.predict(X_test)

    # Calculate MSPE
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    rmse_list.append(rmse)
```

```
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean square d error, use the function'root_mean_squared_error'.
    warnings.warn(
```

```
In [62]: # Average MSPE across all splits  
average_rmse = sum(rmse_list) / len(rmse_list)  
print(f"Linear SVM Average RMSE for Close: {average_rmse}")
```

Linear SVM Average RMSE for Close: 15.221398274445935

```
In [63]: # Print all coefficients sorted by magnitude
print("Linear SVM Coefficients by Magnitude:")
print(svr.feature_importance_)
```

```
Linear SVM Coefficients by Magnitude:  
low                      5.266985  
high                     5.078919  
EP                       4.406339  
open                      4.389974  
PSAR                     4.010959  
...  
bullish_obv_divergence   0.000088  
vpt_hl                   0.000059  
rsi_oversold              0.000000  
dividends                 0.000000  
stock splits                0.000000  
Length: 96, dtype: float64
```

## Non-Linear SVM for feature importance

## Log Transformed Response

```
In [64]: from sklearn.svm import SVR
from sklearn.inspection import permutation_importance
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Standardize predictors
scaler = StandardScaler()
X_scaled = scaler.fit_transform(numeric_data)
y = train_data['log_close'] # Response variable

# Define time series split
tscv = TimeSeriesSplit(n_splits=5)

# Initialize list to store MSPE for each split
rmse_list = []

# Perform time series split and train/test evaluation
for train_index, test_index in tscv.split(X_scaled):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train a non-linear SVM model with RBF kernel
    svr_rbf_model = SVR(kernel='rbf', C=0.1, gamma='scale', epsilon=0.1, max_iter=1000)
    svr_rbf_model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = svr_rbf_model.predict(X_test)

    # Calculate MSPE
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    rmse_list.append(rmse)
```

```

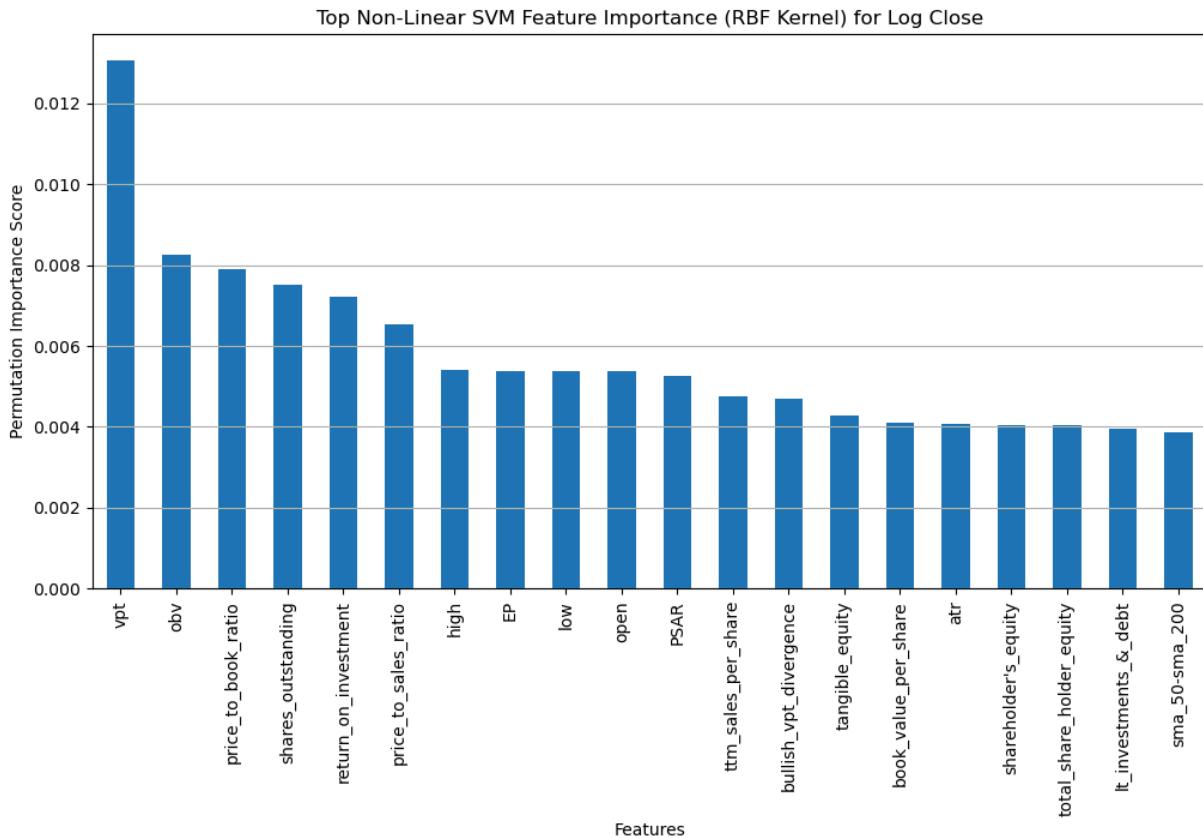
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(

```

In [65]: # Average MSPE across all splits  
`average_rmse = sum(rmse_list) / len(rmse_list)`  
`print(f"Non-linear SVM Average RMSE (Log Scale): {average_rmse}")`  
`print(f"Non-linear SVM Average RMSE (Original Scale): {np.exp(average_rmse)}")`

Non-linear SVM Average RMSE (Log Scale): 0.6068300772895748  
 Non-linear SVM Average RMSE (Original Scale): 1.8346066104355785

In [66]: # Train the model on the full dataset for feature importance  
`svr_rbf_model.fit(X_scaled, y)`  
  
`# Compute permutation importance`  
`perm_importance = permutation_importance(svr_rbf_model, X_scaled, y, n_repeats=10)`  
  
`# Extract feature importance`  
`perm_feature_importance = pd.Series(perm_importance.importances_mean, index=perm_importance.feature_names_in_)`  
`perm_feature_importance.sort_values(ascending=False, inplace=True)`  
  
`# Plot top positive feature importances`  
`perm_feature_importance.head(20).plot(kind='bar', figsize=(12, 6), title="Top 20 Feature Importances")`  
`plt.xlabel('Features')`  
`plt.ylabel('Permutation Importance Score')`  
`plt.grid(axis='y')`  
`plt.show()`



```
In [67]: # Print all feature importances sorted by score
print("Non-Linear SVM (RBF Kernel) Feature Importance:")
print(perm_feature_importance)
```

Non-Linear SVM (RBF Kernel) Feature Importance:

vpt	0.013058
obv	0.008264
price_to_book_ratio	0.007899
shares_outstanding	0.007506
return_on_investment	0.007232
volume_rsi_crossing_50	...
rsi_oversold	0.000352
dividends	0.000000
stock_splits	0.000000
vol_increasing	-0.000107

Length: 96, dtype: float64

```
In [68]: # List features with negative or zero importance
negative_features = perm_feature_importance[perm_feature_importance <= 0]
print("\nFeatures with Negative Importance for Log Close:")
print(negative_features)
```

Features with Negative Importance for Log Close:

rsi_oversold	0.000000
dividends	0.000000
stock_splits	0.000000
vol_increasing	-0.000107

dtype: float64

## Normal Close Values

In [69]:

```
from sklearn.svm import SVR
from sklearn.inspection import permutation_importance
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# Standardize predictors
scaler = StandardScaler()
X_scaled = scaler.fit_transform(numeric_data)
y = train_data['close'] # Response variable

# Define time series split
tscv = TimeSeriesSplit(n_splits=5)

# Initialize list to store MSPE for each split
rmse_list = []

# Perform time series split and train/test evaluation
for train_index, test_index in tscv.split(X_scaled):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Train a non-linear SVM model with RBF kernel
    svr_rbf_model = SVR(kernel='rbf', C=0.1, gamma='scale', epsilon=0.1, max_iter=1000)
    svr_rbf_model.fit(X_train, y_train)

    # Predict on the test set
    y_pred = svr_rbf_model.predict(X_test)

    # Calculate MSPE
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    rmse_list.append(rmse)
```

```

/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(

```

In [70]: # Average MSPE across all splits  
`average_rmse = sum(rmse_list) / len(rmse_list)`  
`print(f"Non-Linear SVM (RBF Kernel) Average RMSE for Close: {average_rmse}")`

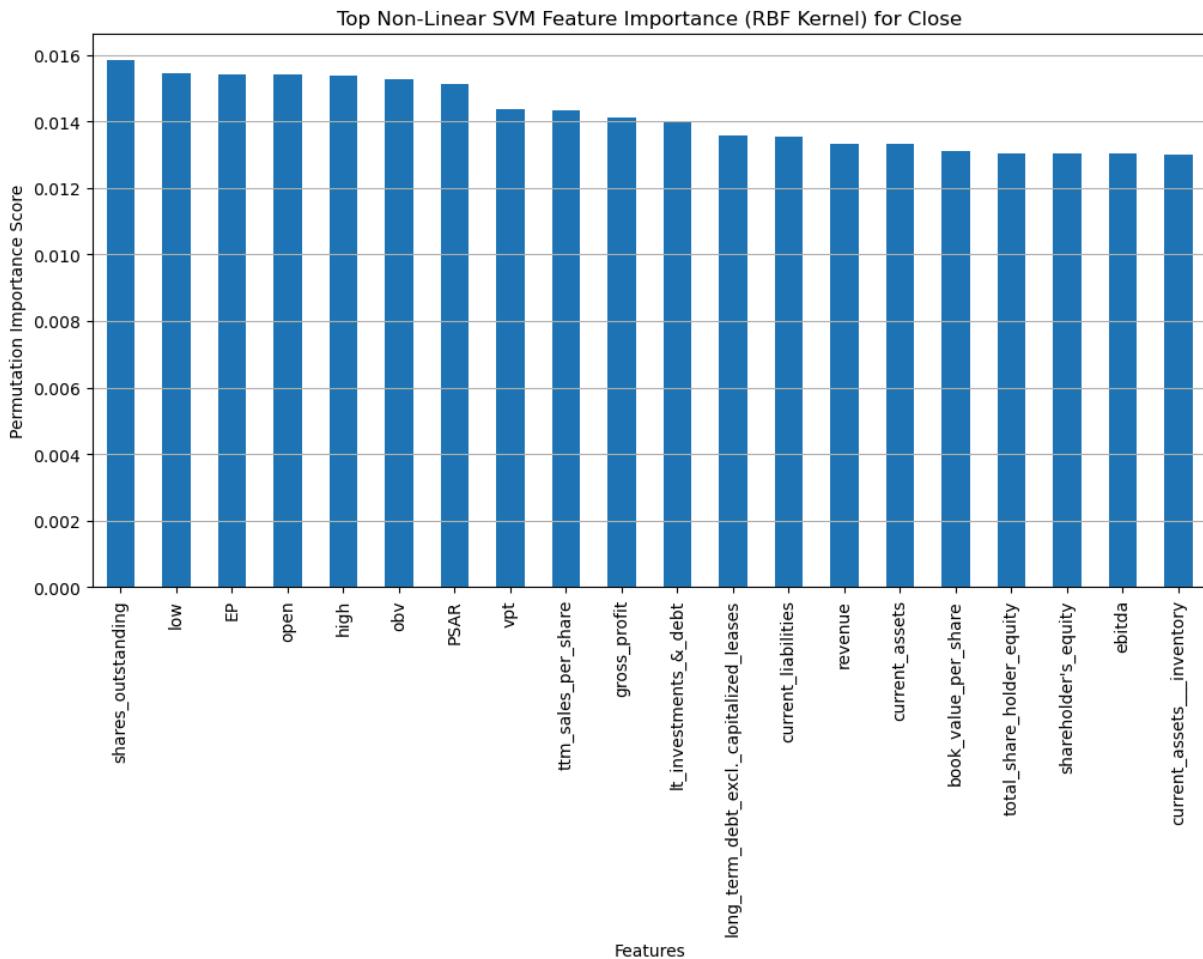
Non-Linear SVM (RBF Kernel) Average RMSE for Close: 24.458283288430287

In [71]: # Train the model on the full dataset for feature importance  
`svr_rbf_model.fit(X_scaled, y)`

# Compute permutation importance  
`perm_importance = permutation_importance(svr_rbf_model, X_scaled, y, n_repeats=10)`

# Extract feature importance  
`perm_feature_importance = pd.Series(perm_importance.importances_mean, index=X_scaled.columns)`  
`perm_feature_importance.sort_values(ascending=False, inplace=True)`

# Plot top positive feature importances  
`perm_feature_importance.head(20).plot(kind='bar', figsize=(12, 6), title="Top 20 Feature Importances")`  
`plt.xlabel('Features')`  
`plt.ylabel('Permutation Importance Score')`  
`plt.grid(axis='y')`  
`plt.show()`



```
In [72]: # Print all feature importances sorted by score
print("Non-Linear SVM (RBF Kernel) Feature Importance for Close:")
print(perm_feature_importance)
```

Non-Linear SVM (RBF Kernel) Feature Importance for Close:

Feature	Importance Score
shares_outstanding	0.015826
low	0.015451
EP	0.015419
open	0.015406
high	0.015378
...	
vol_decreasing	-0.000275
increasing	-0.000496
momentum	-0.001222
sma_50-sma_200_rate	-0.002852
MACDh_10_26_9	-0.003012
Length:	96, dtype: float64

```
In [73]: # List features with negative or zero importance
negative_features = perm_feature_importance[perm_feature_importance <= 0]
print("\nFeatures with Negative Importance for Close:")
print(negative_features)
```

```
Features with Negative Importance for Close:  
dividends           0.000000  
stock splits         0.000000  
rsi_oversold        0.000000  
volume_rsi_crossing_50 -0.000017  
MACDs_10_26_9       -0.000108  
decreasing          -0.000241  
vol_decreasing      -0.000275  
increasing          -0.000496  
momentum            -0.001222  
sma_50-sma_200_rate -0.002852  
MACDh_10_26_9       -0.003012  
dtype: float64
```

## Clustering Methods

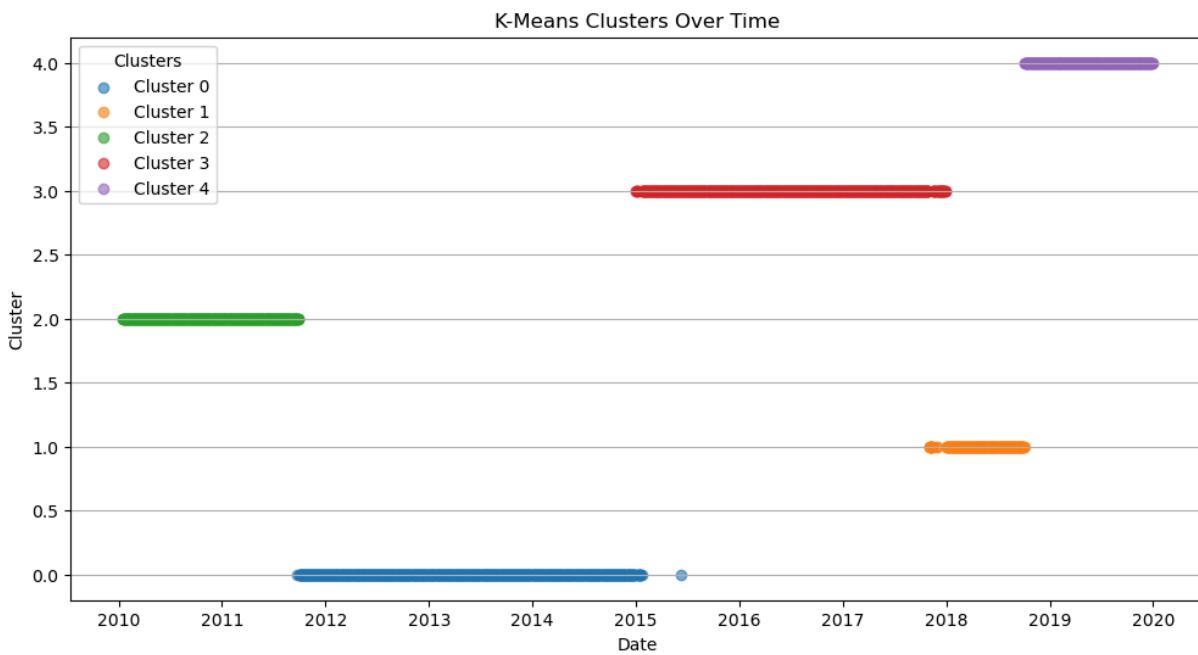
### KMeans Clustering

```
In [74]: from sklearn.cluster import KMeans  
from sklearn.preprocessing import StandardScaler  
import pandas as pd  
import matplotlib.pyplot as plt  
  
# Standardize predictors  
scaler = StandardScaler()  
X_scaled = scaler.fit_transform(numeric_data)  
  
# Perform K-Means clustering  
kmeans = KMeans(n_clusters=5, random_state=42, algorithm="elkan")  
kmeans_clusters = kmeans.fit_predict(X_scaled)  
  
# Ensure 'Date' is in datetime format  
train_data1 = train_data.copy() # Create a copy to avoid SettingWithCopyWarning  
train_data1.loc[:, 'KMeans_Cluster'] = kmeans_clusters  
train_data1.loc[:, 'Date'] = pd.to_datetime(train_data['Date']) # Ensure Date is in datetime format  
  
# Summarize response ('close') by cluster  
kmeans_summary = train_data1.groupby('KMeans_Cluster').mean()  
print("K-Means Cluster Summary (Average Close Price):")  
print(kmeans_summary)
```

K-Means Cluster Summary (Average Close Price):							\		
KMeans_Cluster	Date	dxy	oil	\	gold	treasury_5_years	treasury_10_years	\	
0	2013-05-23 11:26:57.391304192	81.396510	94.435869						
1	2018-05-08 02:43:09.162561536	92.759114	66.316749						
2	2010-11-21 14:38:42.041763328	78.964919	86.520835						
3	2016-06-30 03:59:20.762942720	96.664237	47.538338						
4	2019-05-19 09:55:23.076923136	97.230096	57.347308						
KMeans_Cluster	gold	treasury_5_years	treasury_10_years	\	treasury_30_years	close	volume	ttm_net_eps	\
0	1463.707004	1.167854	2.200315						
1	1280.106404	2.668251	2.841946						
2	1361.824131	1.805842	3.104121						
3	1220.746050	1.583035	2.096290						
4	1361.254486	2.130051	2.310808						
KMeans_Cluster	treasury_30_years	close	volume	ttm_net_eps	\	lt_investments_&_debt	return_on_investment	open	\
0	3.212406	13.896132	7.706319e+07	0.022826		11.518913	6.243696	13.896431	
1	3.048601	80.503919	1.081875e+08	0.439754		55.814828	11.446946	80.513271	
2	4.220958	8.143020	1.297453e+08	0.116102		7.395035	28.956056	8.131974	
3	2.780162	35.655860	7.378753e+07	0.124074		25.622057	10.929183	35.662905	
4	2.712215	88.092816	9.247963e+07	1.091314		71.772500	20.543590	88.158043	
KMeans_Cluster	high	low	dividends	stock splits	\	long_term_debt_excl._capitalized_leases	log_close	Directio	n
0	14.050676	13.726659	0.0	0.0		2771.719807	2.608411	0.50966	
1	81.277005	79.598584	0.0	0.0		24675.118227	4.376728	0.62561	
2	8.247664	8.017571	0.0	0.0		876.981439	2.076140	0.51740	
3	35.950422	35.330703	0.0	0.0		9239.245232	3.523150	0.54359	
4	88.992631	87.134378	0.0	0.0		23441.285256	4.475998	0.51602	
6									

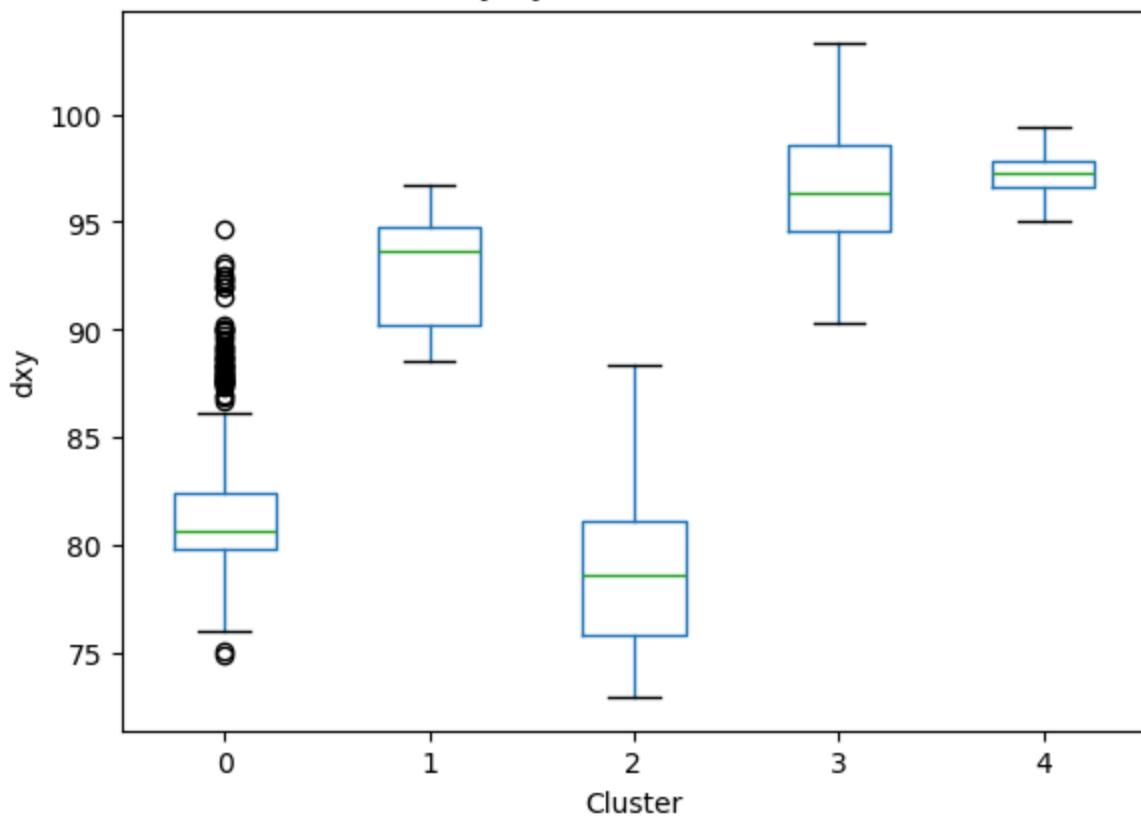
[5 rows x 100 columns]

```
In [75]: # Visualize cluster distribution over time
plt.figure(figsize=(12, 6))
for cluster in sorted(train_data1['KMeans_Cluster'].unique()):
    cluster_dates = train_data1[train_data1['KMeans_Cluster'] == cluster]['Date']
    plt.scatter(cluster_dates, [cluster] * len(cluster_dates), label=f"Cluster {cluster}")
plt.title("K-Means Clusters Over Time")
plt.xlabel("Date")
plt.ylabel("Cluster")
plt.legend(title="Clusters")
plt.grid(axis='y')
plt.show()
```

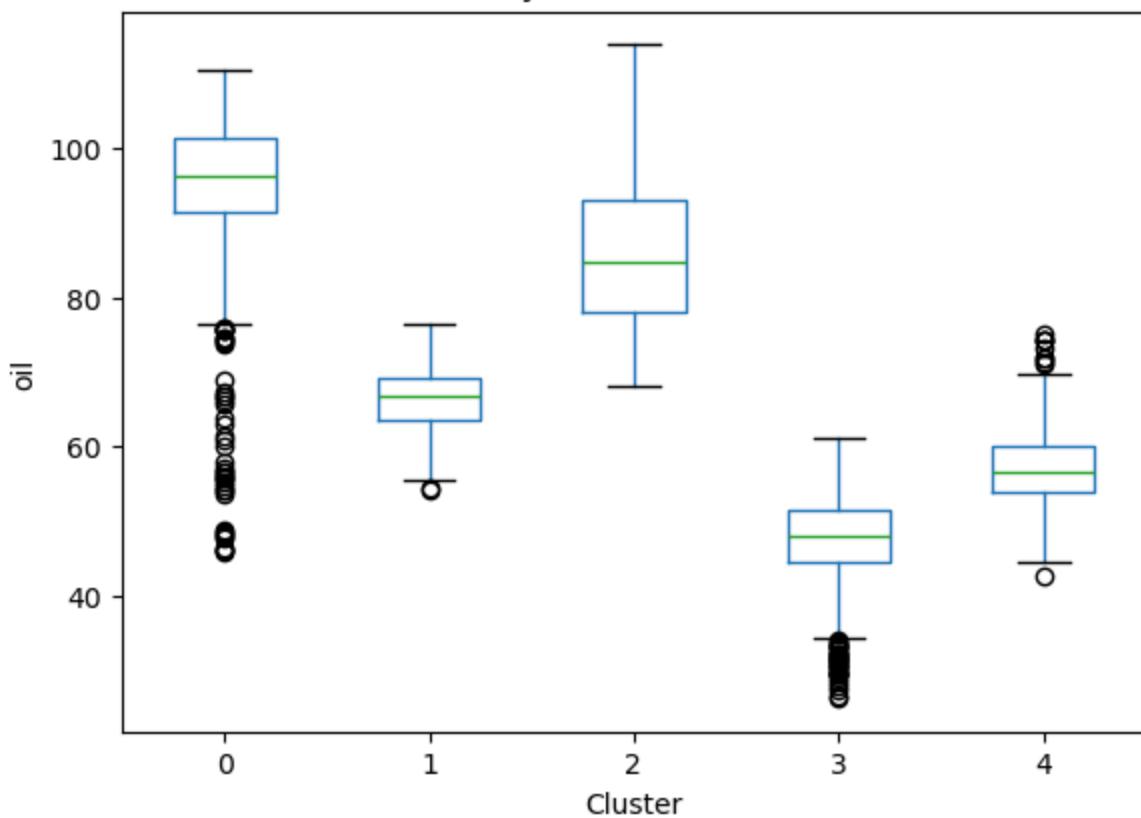


```
In [76]: for feature in numeric_data.columns:
    train_data1.boxplot(column=feature, by='KMeans_Cluster', grid=False)
    plt.title(f'{feature} by K-Means Cluster')
    plt.xlabel("Cluster")
    plt.ylabel(feature)
    plt.show()
```

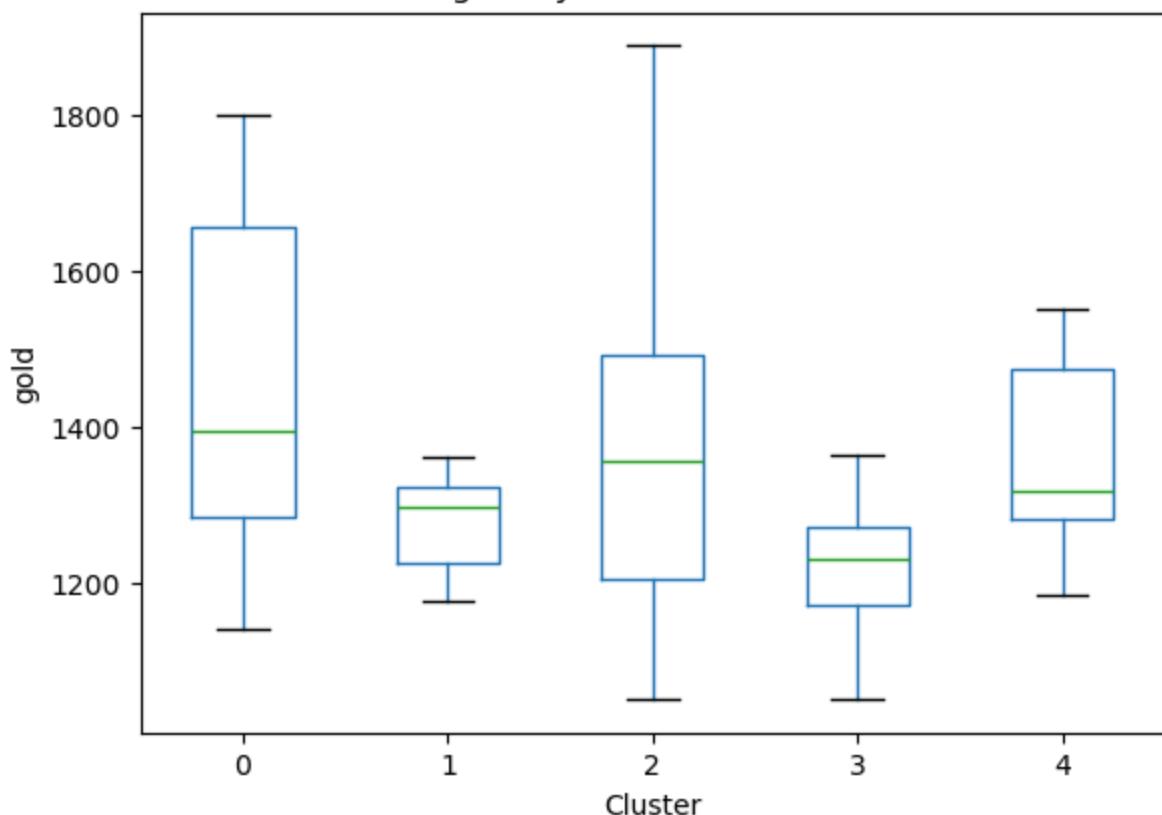
Boxplot grouped by KMeans\_Cluster  
dxy by K-Means Cluster



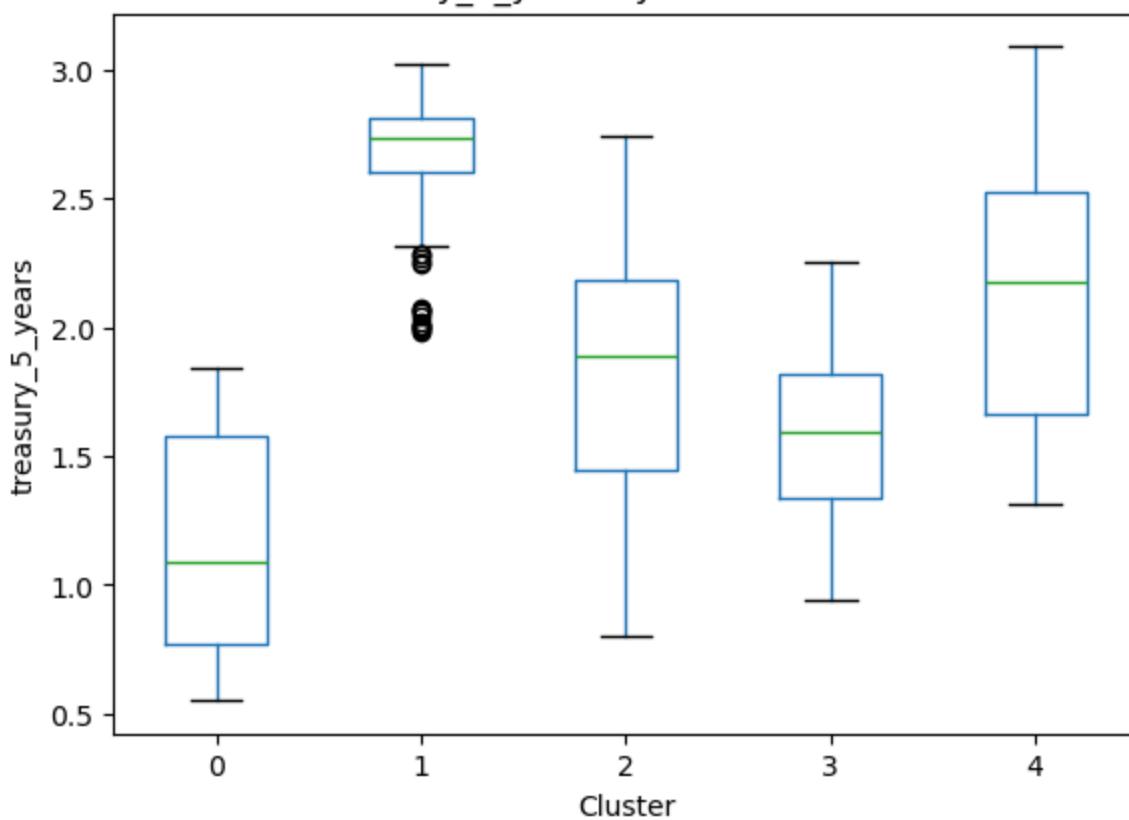
Boxplot grouped by KMeans\_Cluster  
oil by K-Means Cluster



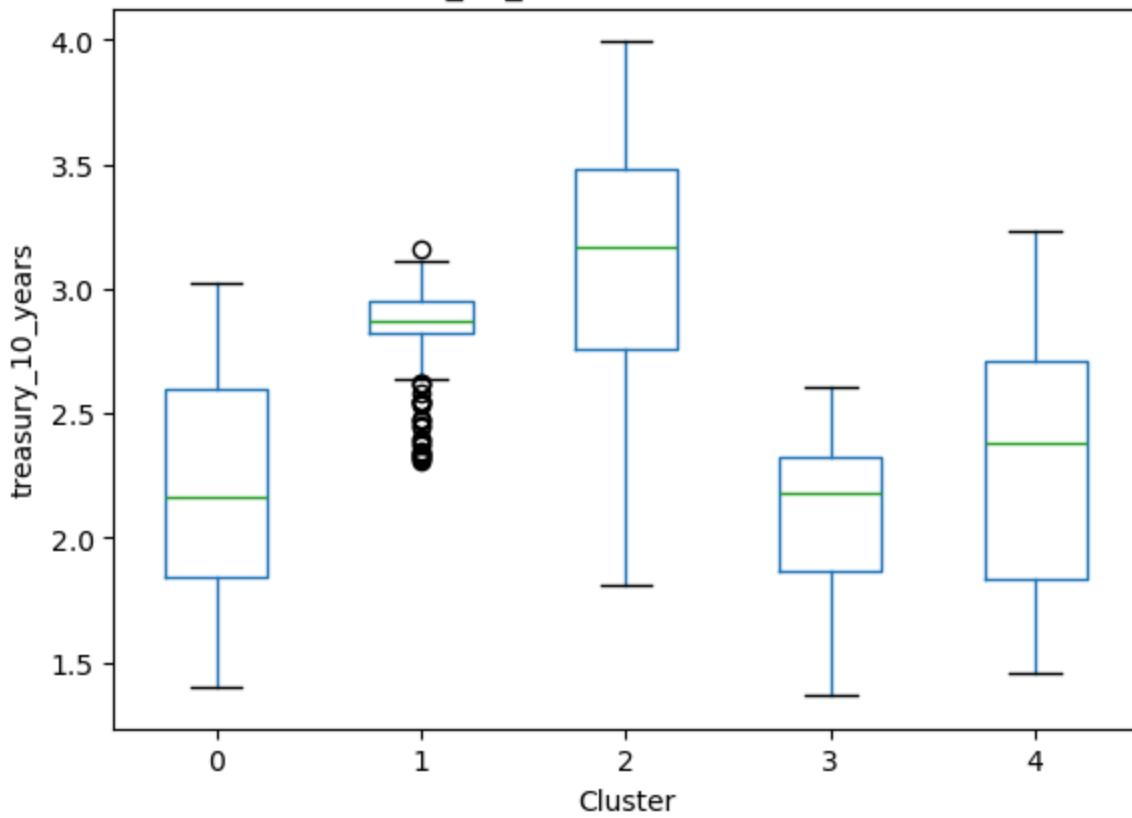
Boxplot grouped by KMeans\_Cluster  
gold by K-Means Cluster



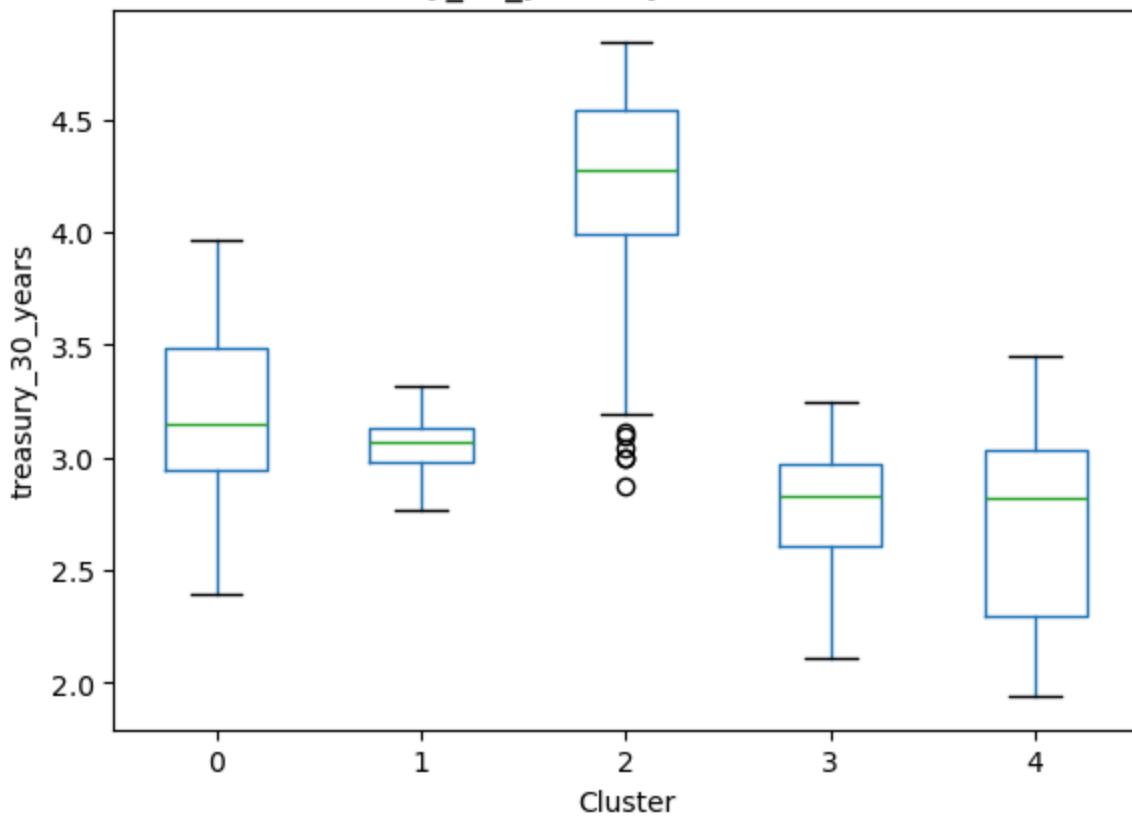
Boxplot grouped by KMeans\_Cluster  
treasury\_5\_years by K-Means Cluster

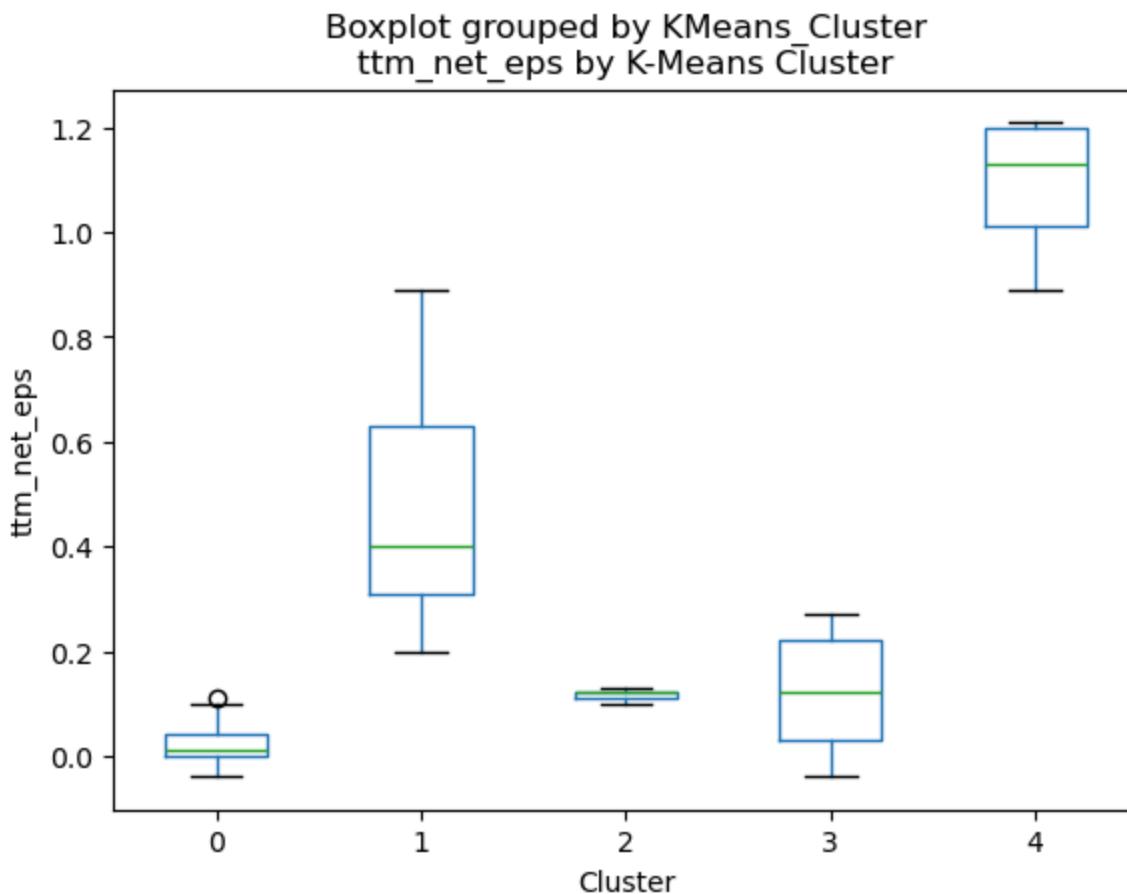
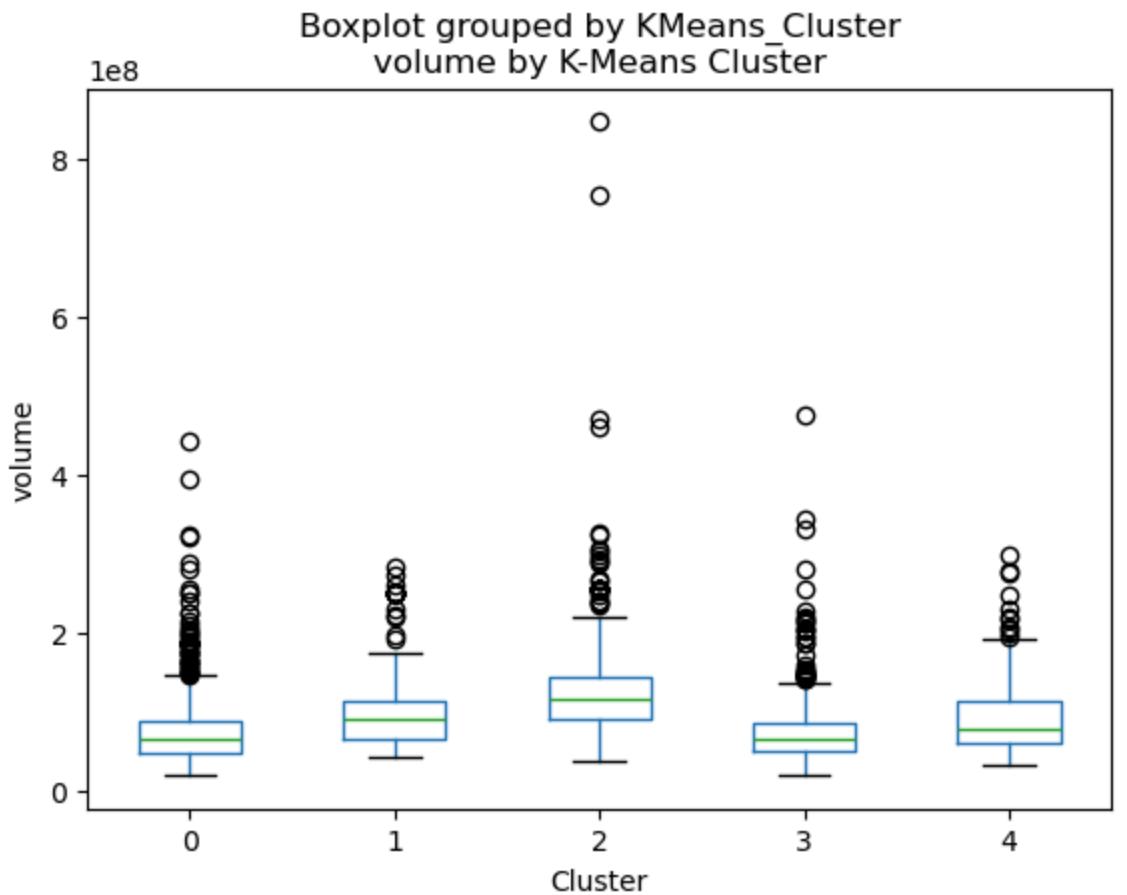


Boxplot grouped by KMeans\_Cluster  
treasury\_10\_years by K-Means Cluster

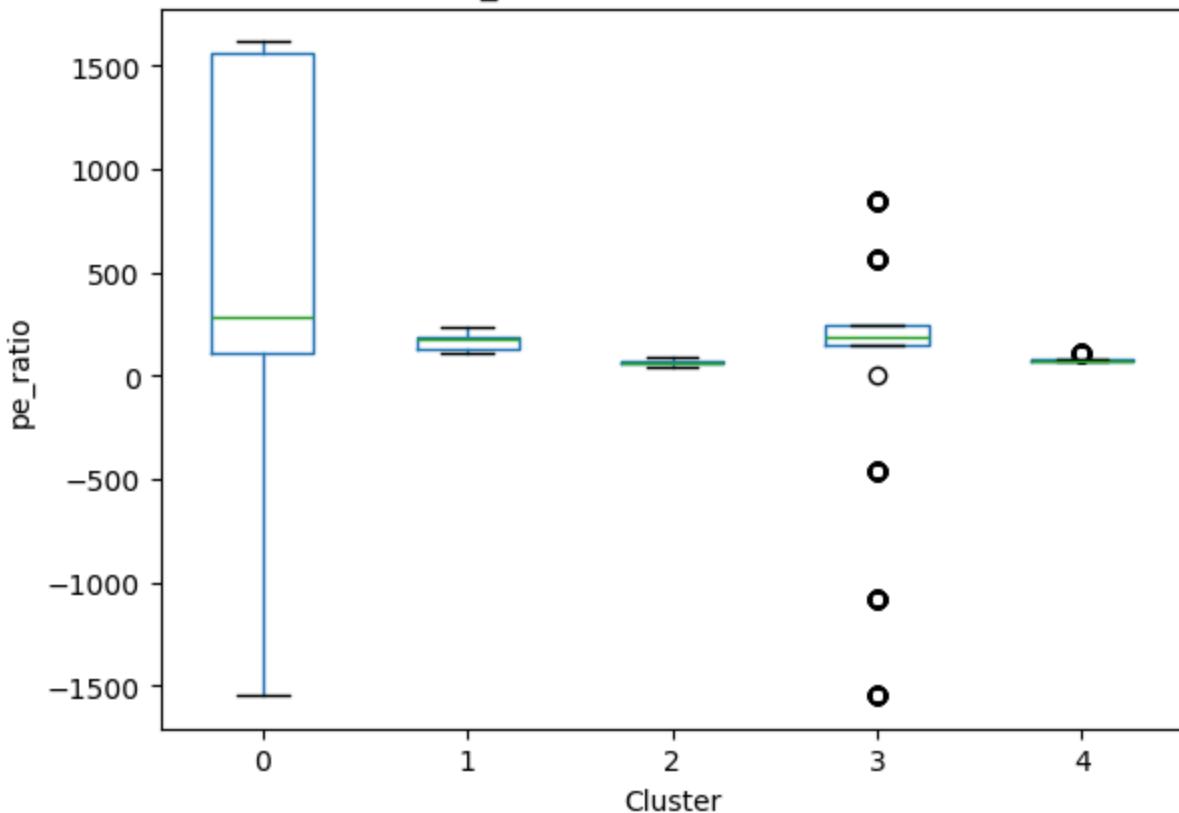


Boxplot grouped by KMeans\_Cluster  
treasury\_30\_years by K-Means Cluster

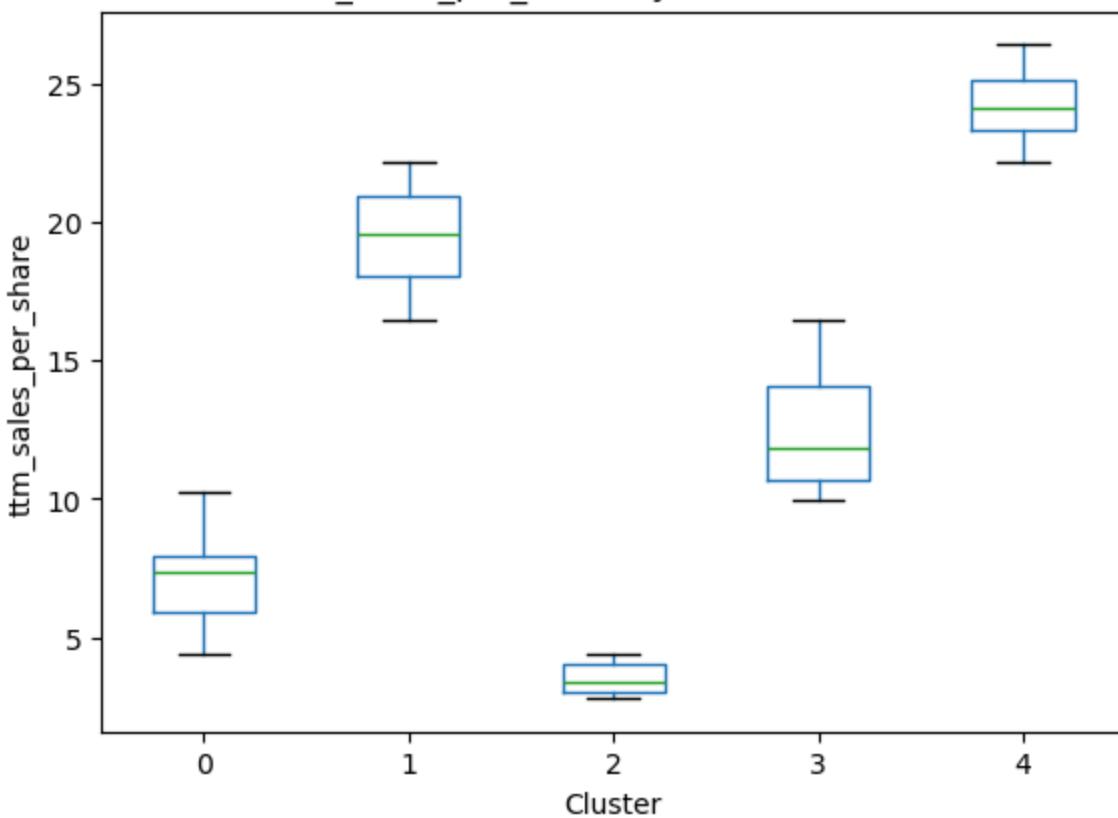




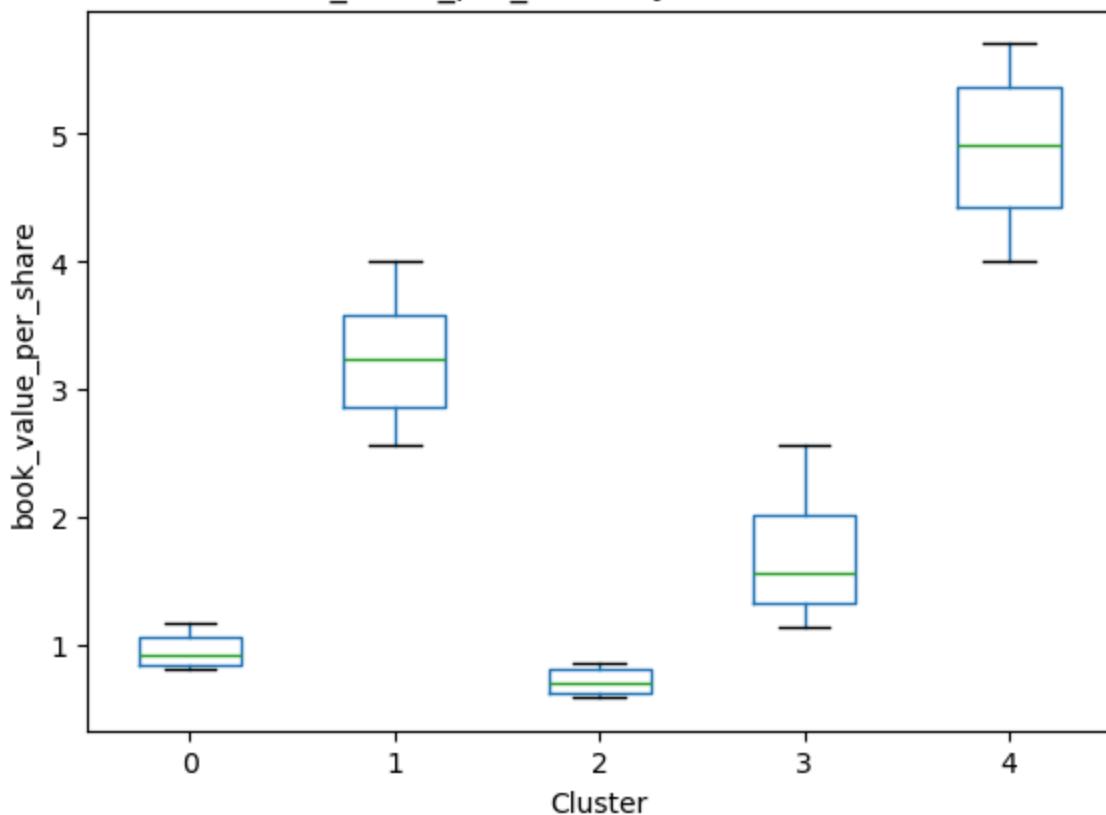
Boxplot grouped by KMeans\_Cluster  
pe\_ratio by K-Means Cluster



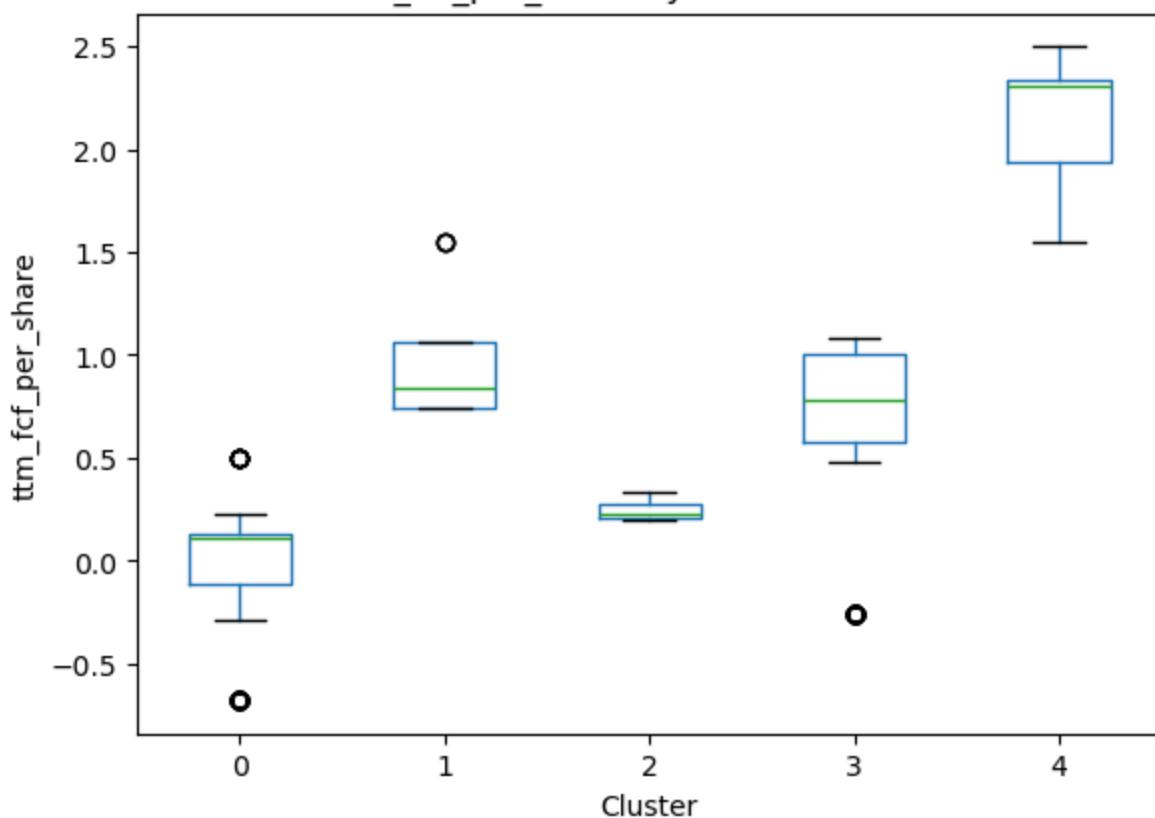
Boxplot grouped by KMeans\_Cluster  
ttm\_sales\_per\_share by K-Means Cluster



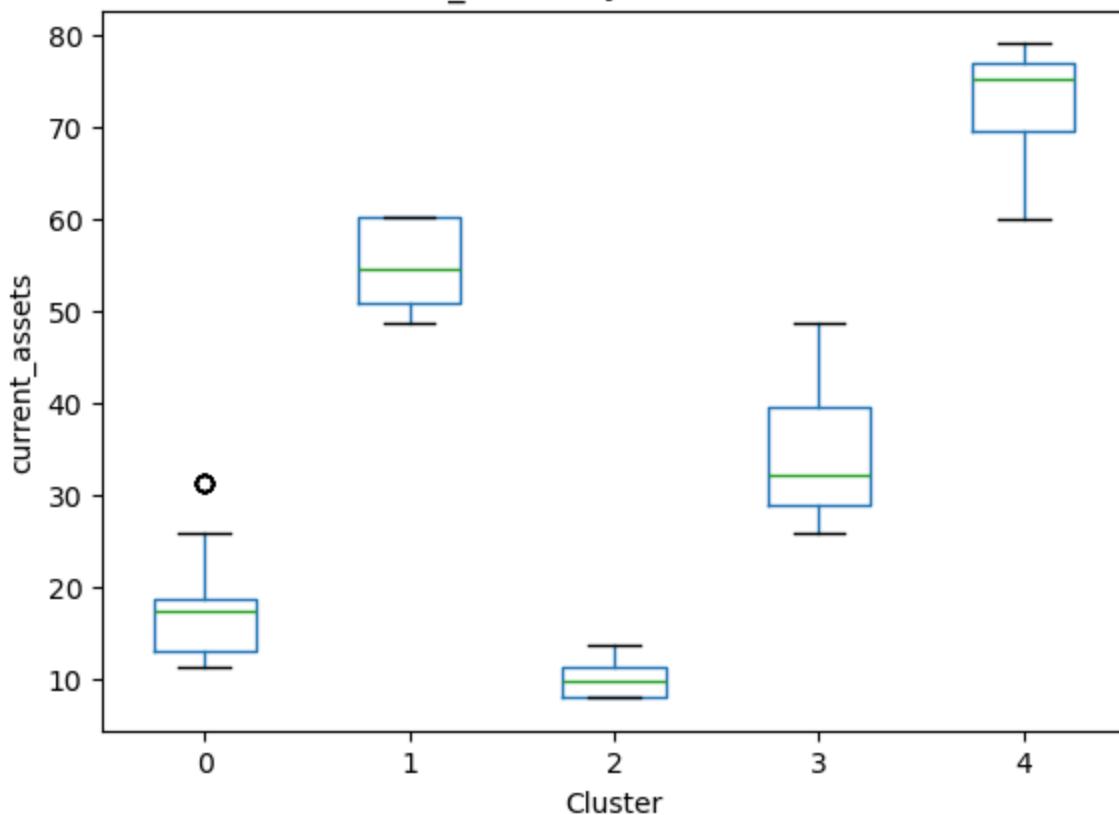
Boxplot grouped by KMeans\_Cluster  
book\_value\_per\_share by K-Means Cluster



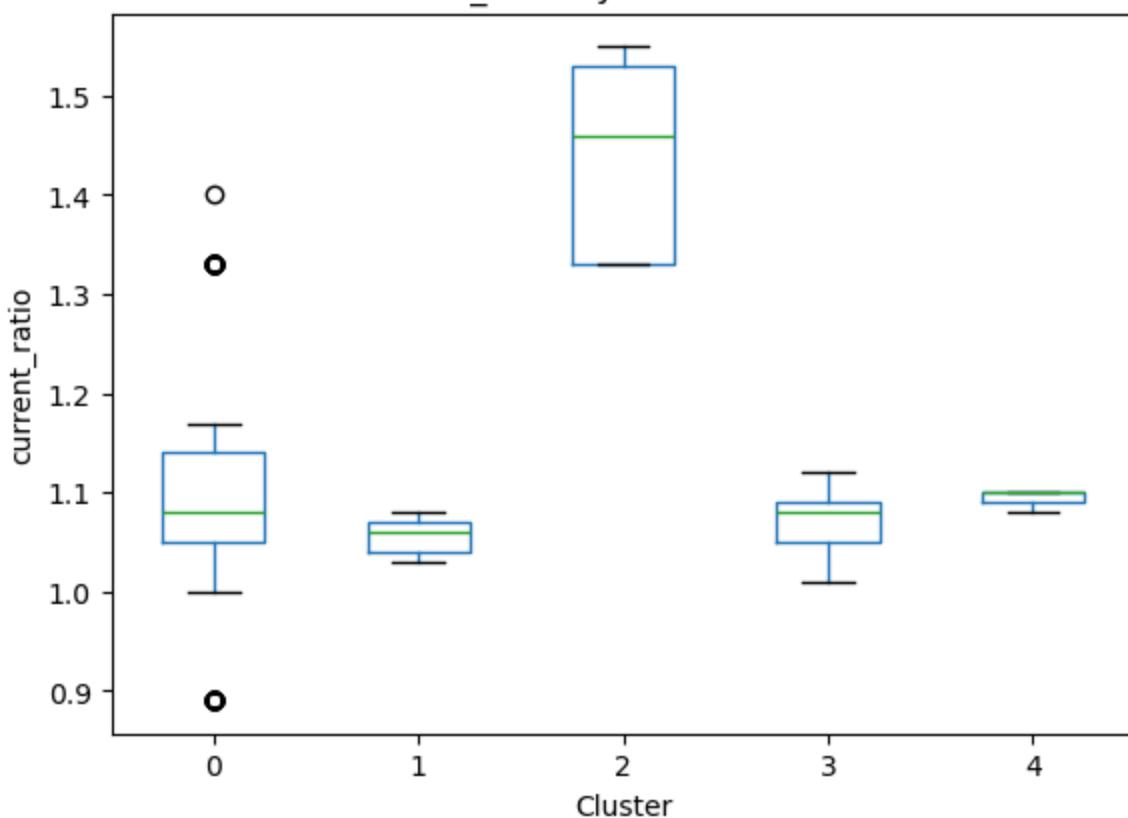
Boxplot grouped by KMeans\_Cluster  
ttm\_fcf\_per\_share by K-Means Cluster

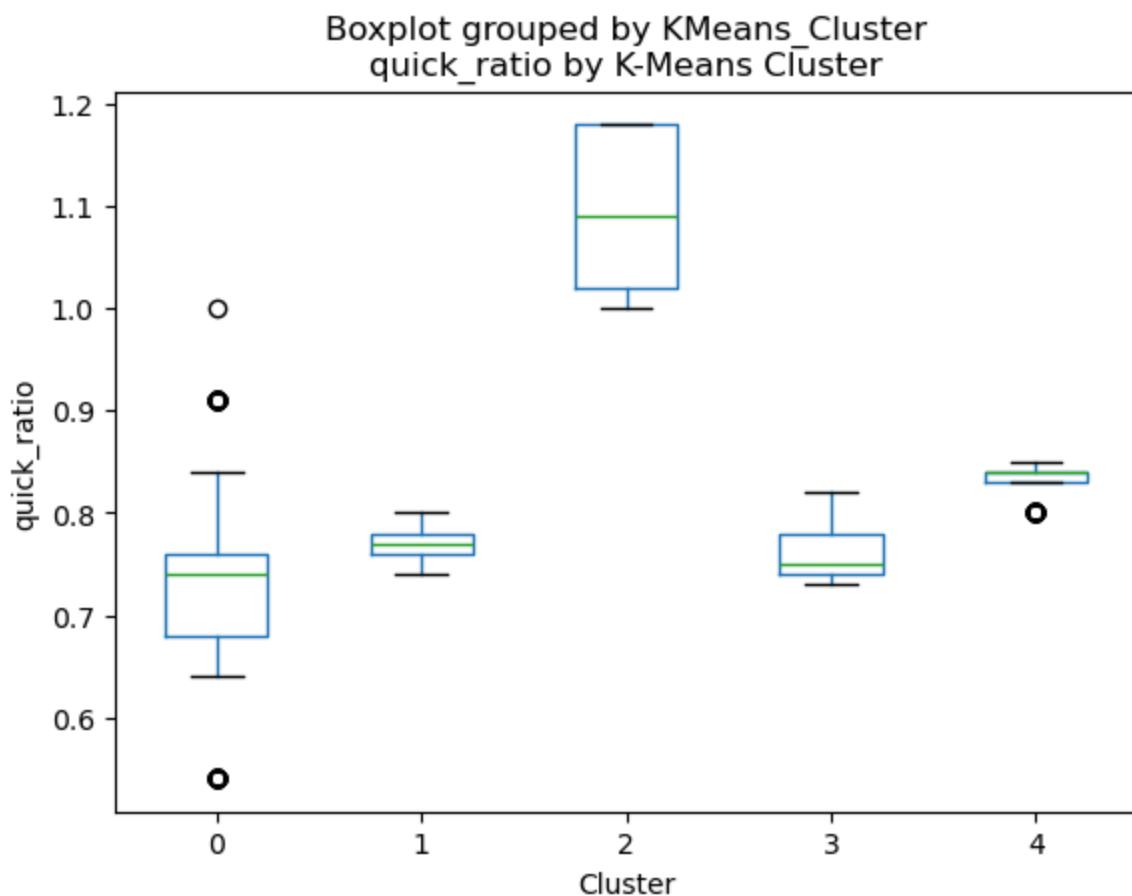
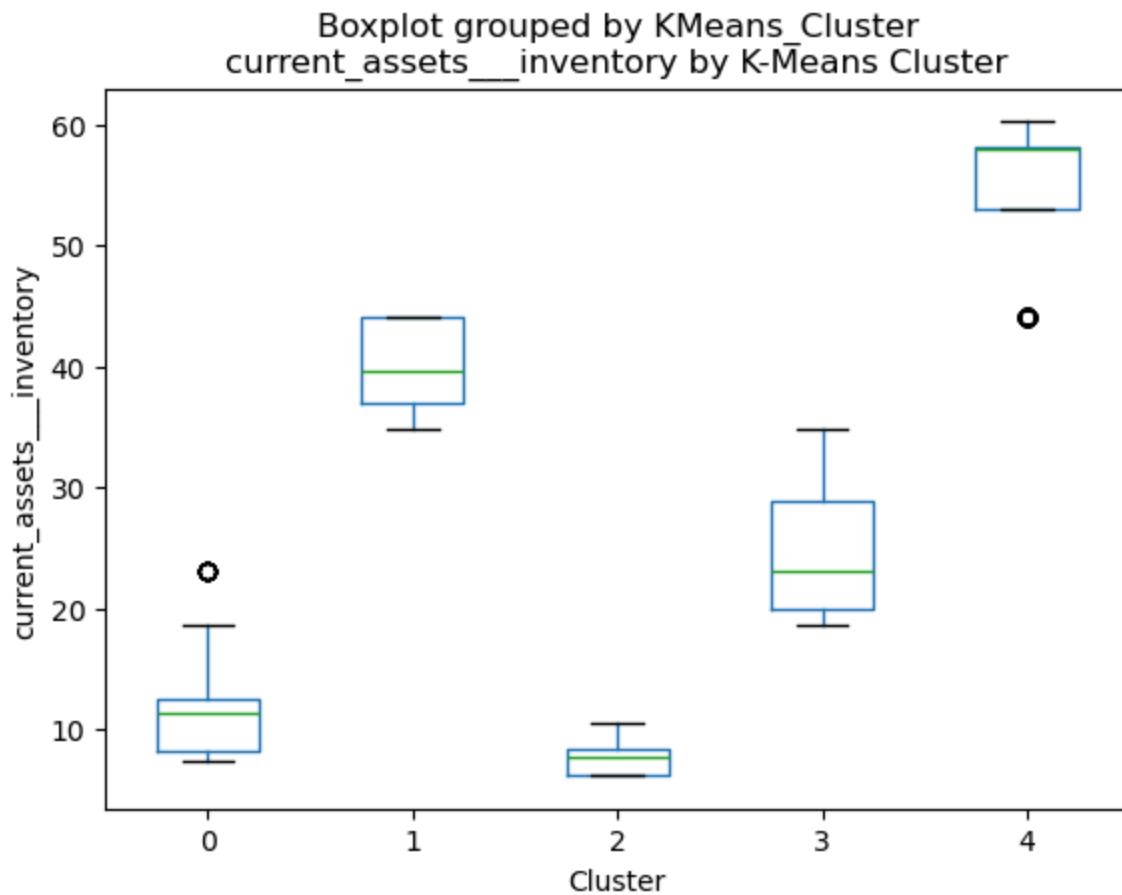


Boxplot grouped by KMeans\_Cluster  
current\_assets by K-Means Cluster

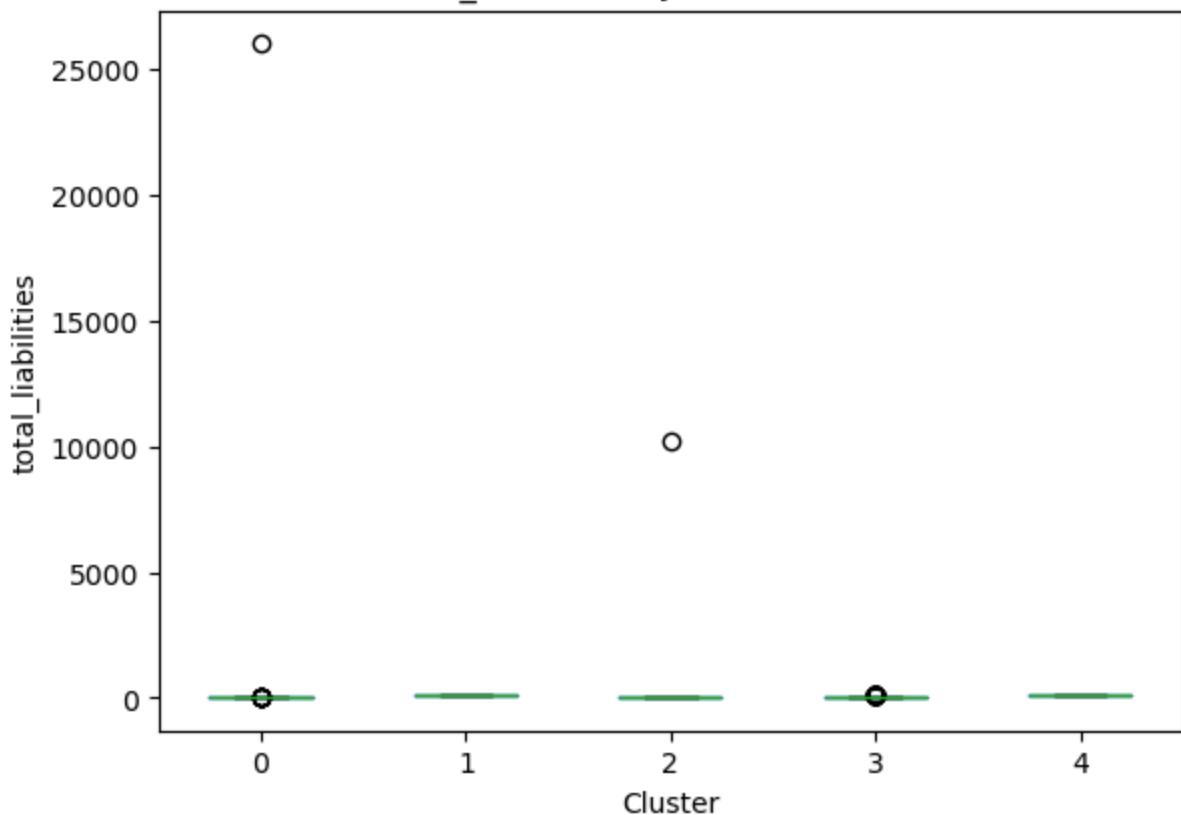


Boxplot grouped by KMeans\_Cluster  
current\_ratio by K-Means Cluster

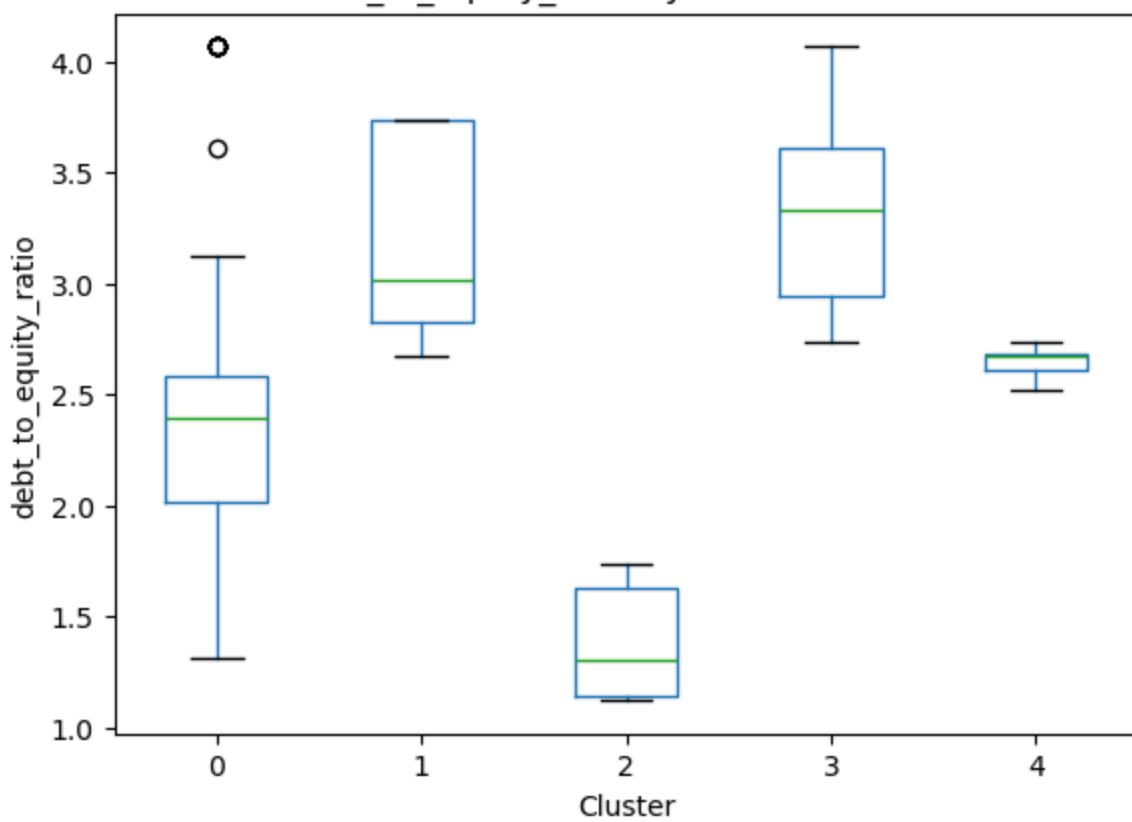




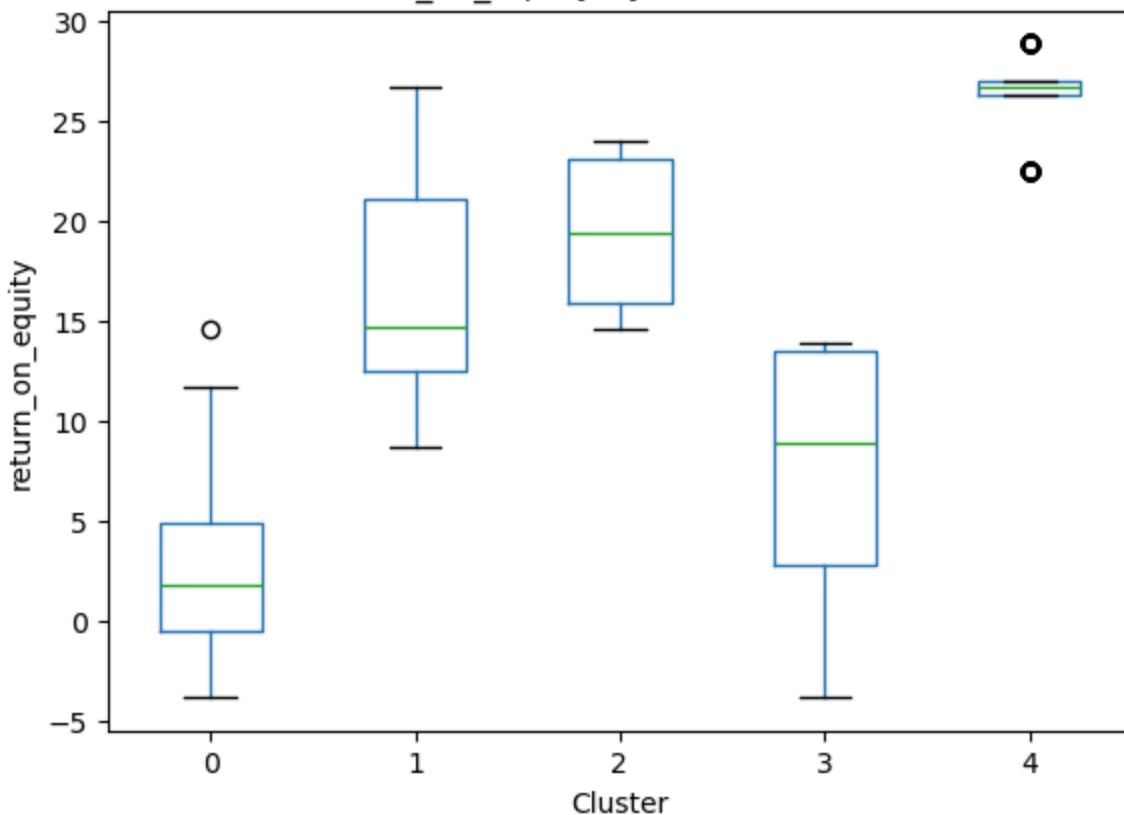
Boxplot grouped by KMeans\_Cluster  
total\_liabilities by K-Means Cluster



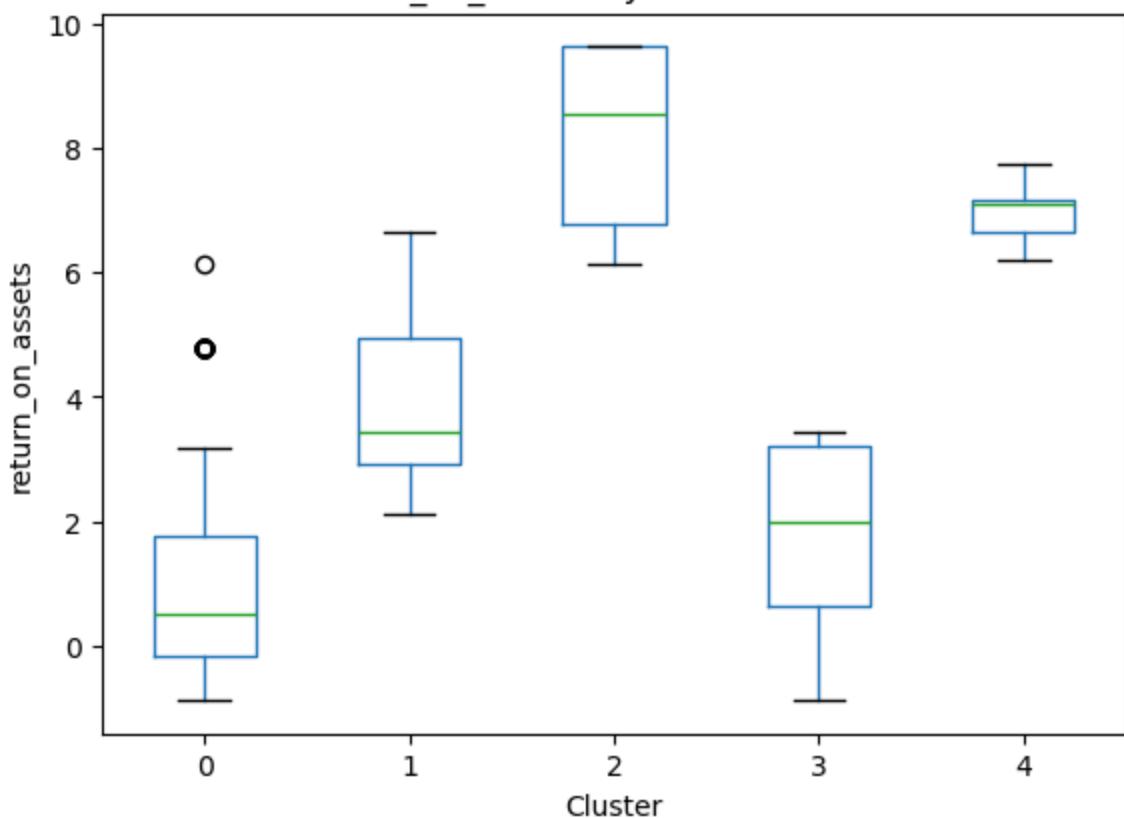
Boxplot grouped by KMeans\_Cluster  
debt\_to\_equity\_ratio by K-Means Cluster



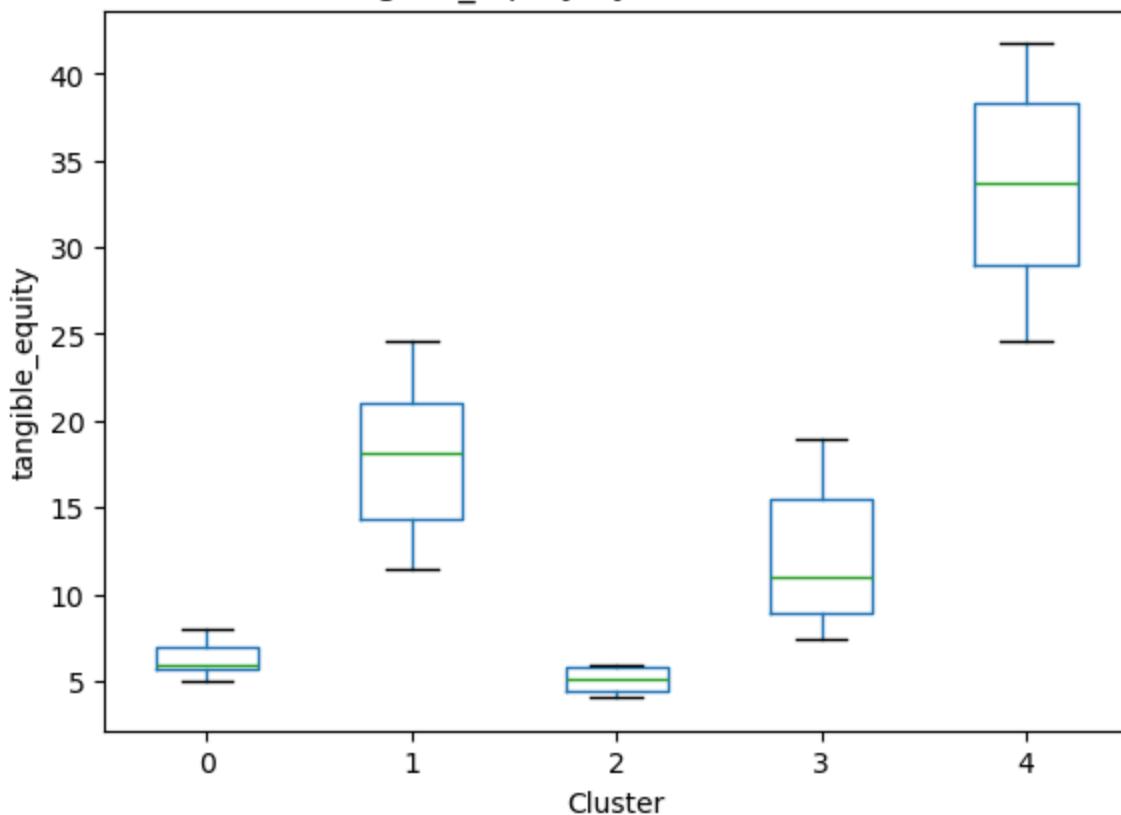
Boxplot grouped by KMeans\_Cluster  
return\_on\_equity by K-Means Cluster



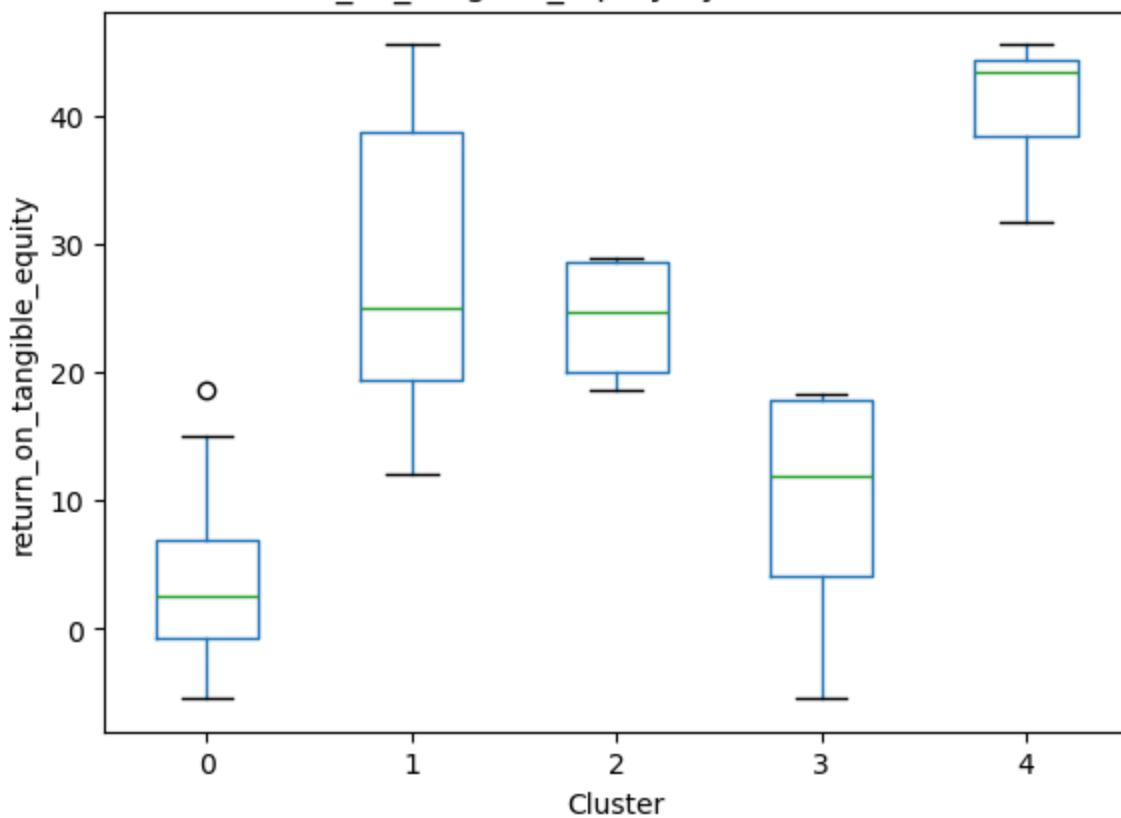
Boxplot grouped by KMeans\_Cluster  
return\_on\_assets by K-Means Cluster



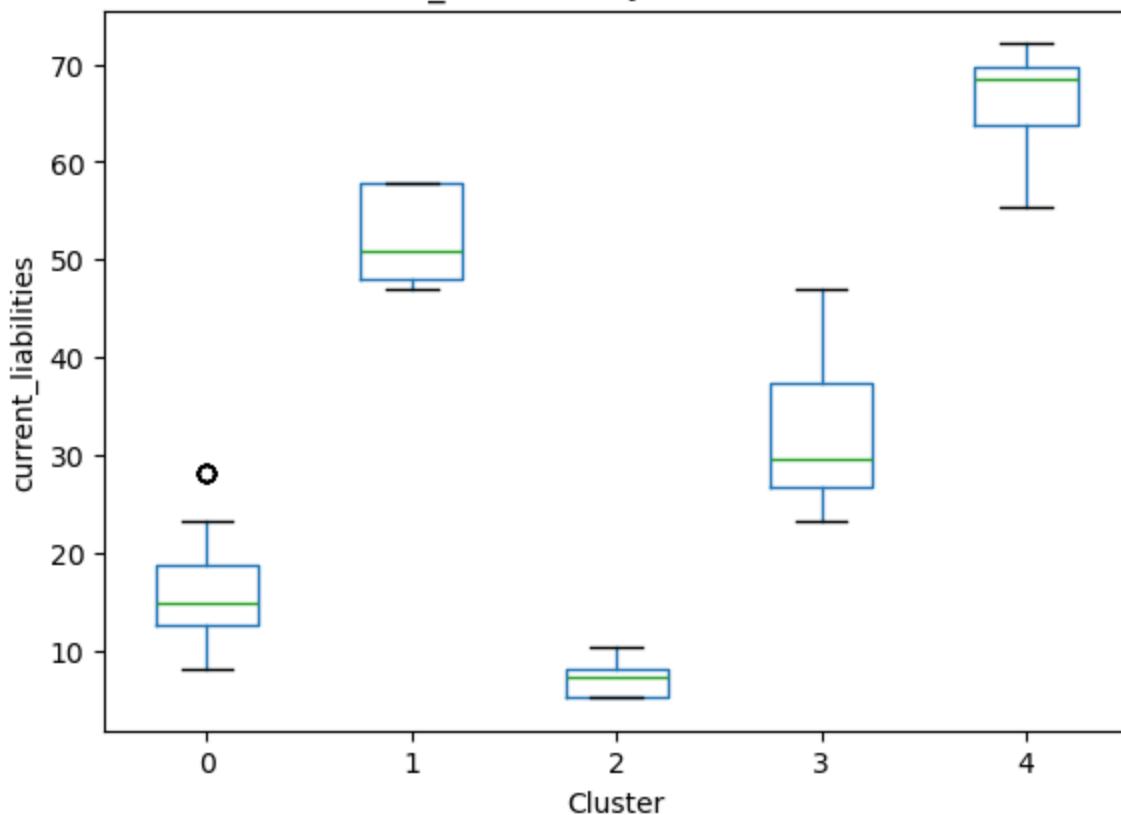
Boxplot grouped by KMeans\_Cluster  
tangible\_equity by K-Means Cluster



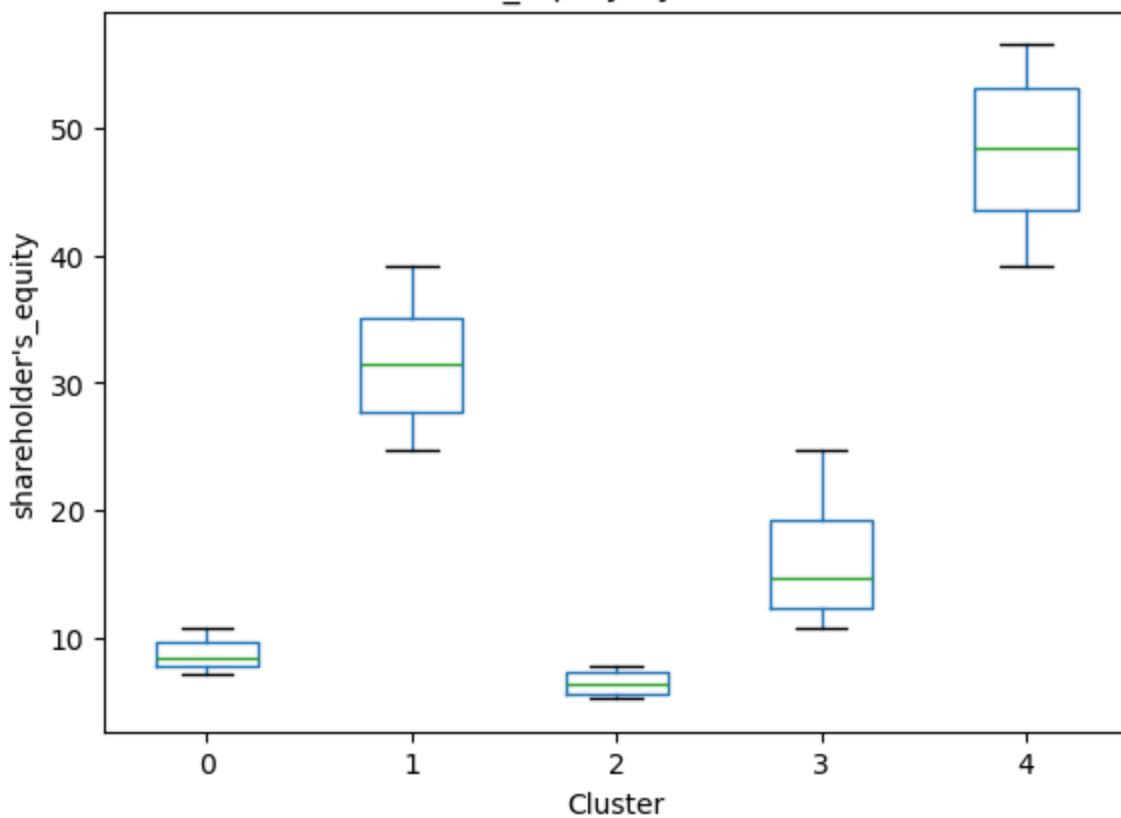
Boxplot grouped by KMeans\_Cluster  
return\_on\_tangible\_equity by K-Means Cluster



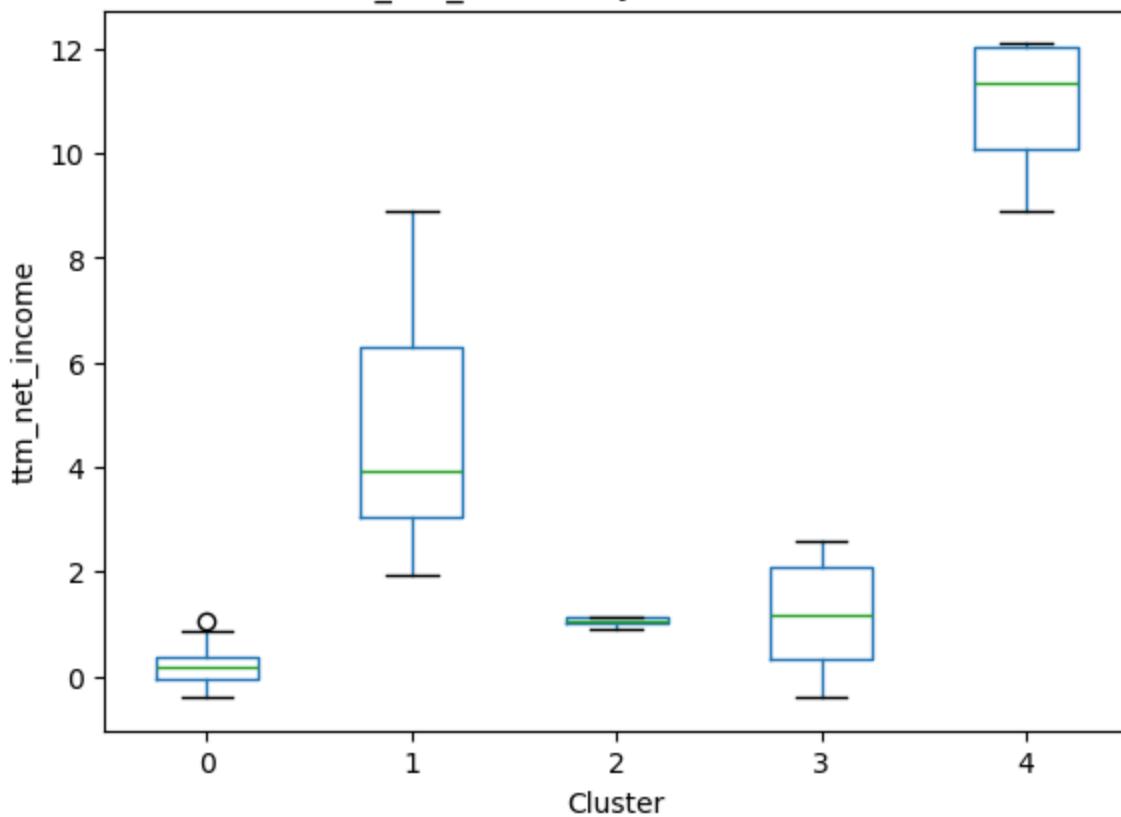
Boxplot grouped by KMeans\_Cluster  
current\_liabilities by K-Means Cluster



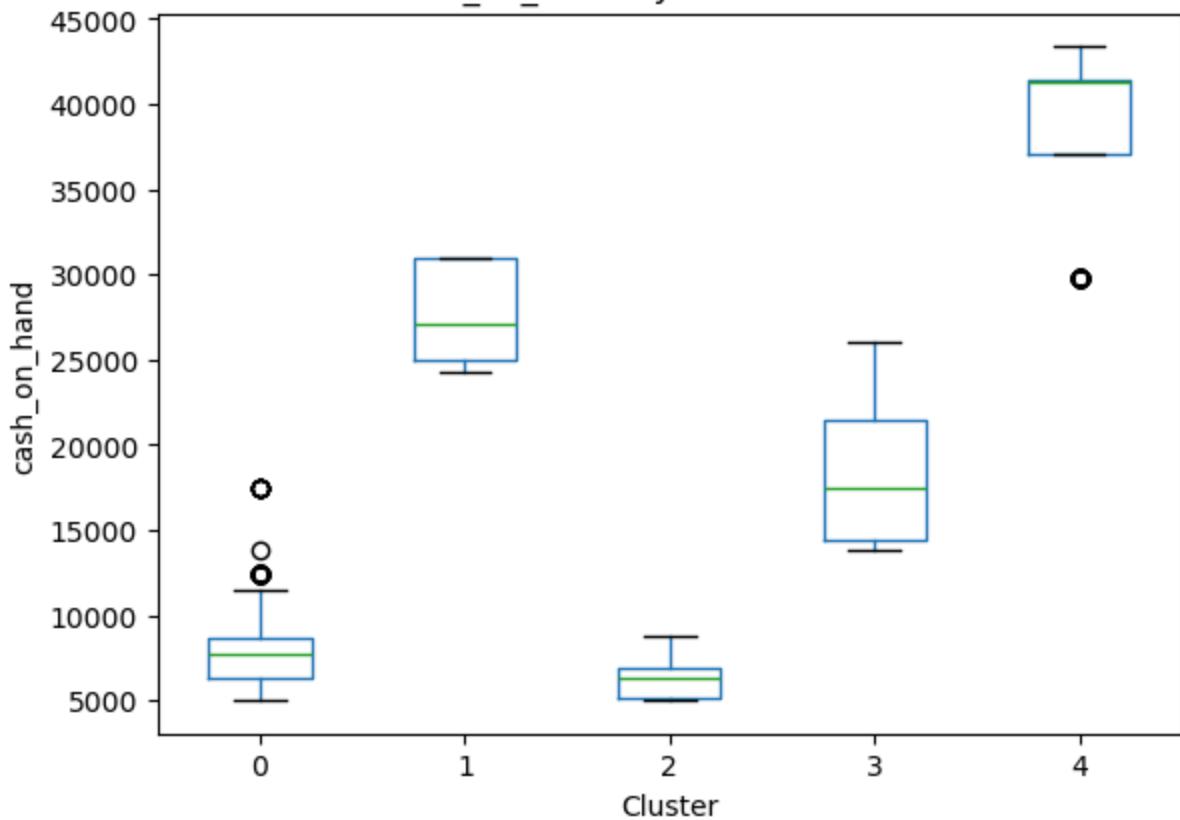
Boxplot grouped by KMeans\_Cluster  
shareholder's\_equity by K-Means Cluster



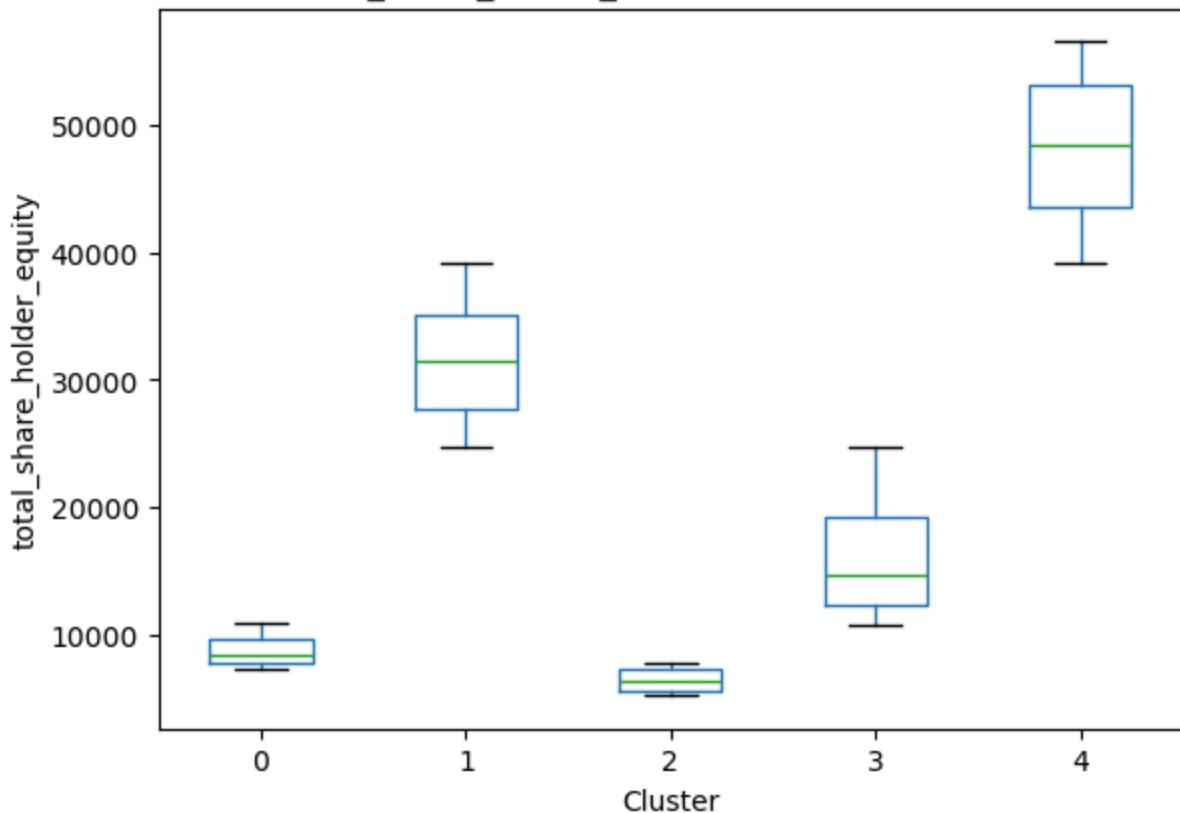
Boxplot grouped by KMeans\_Cluster  
ttm\_net\_income by K-Means Cluster



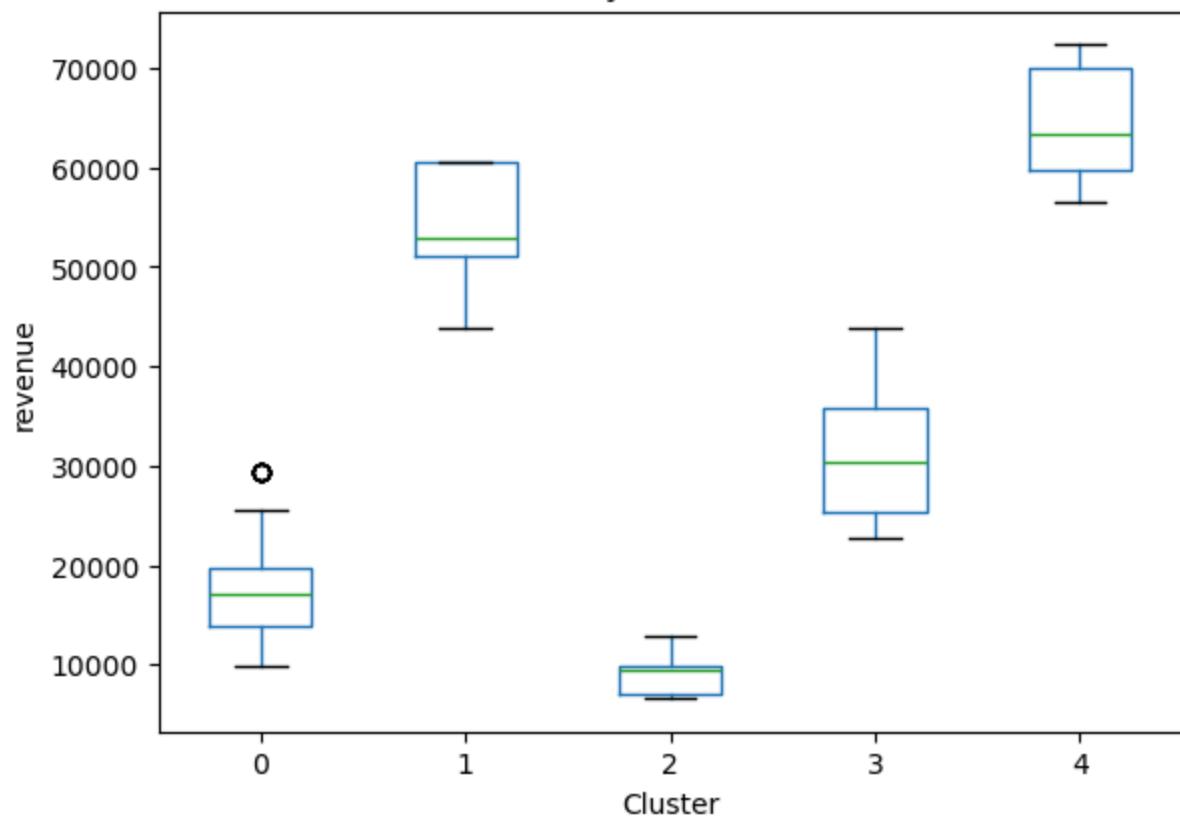
Boxplot grouped by KMeans\_Cluster  
cash\_on\_hand by K-Means Cluster

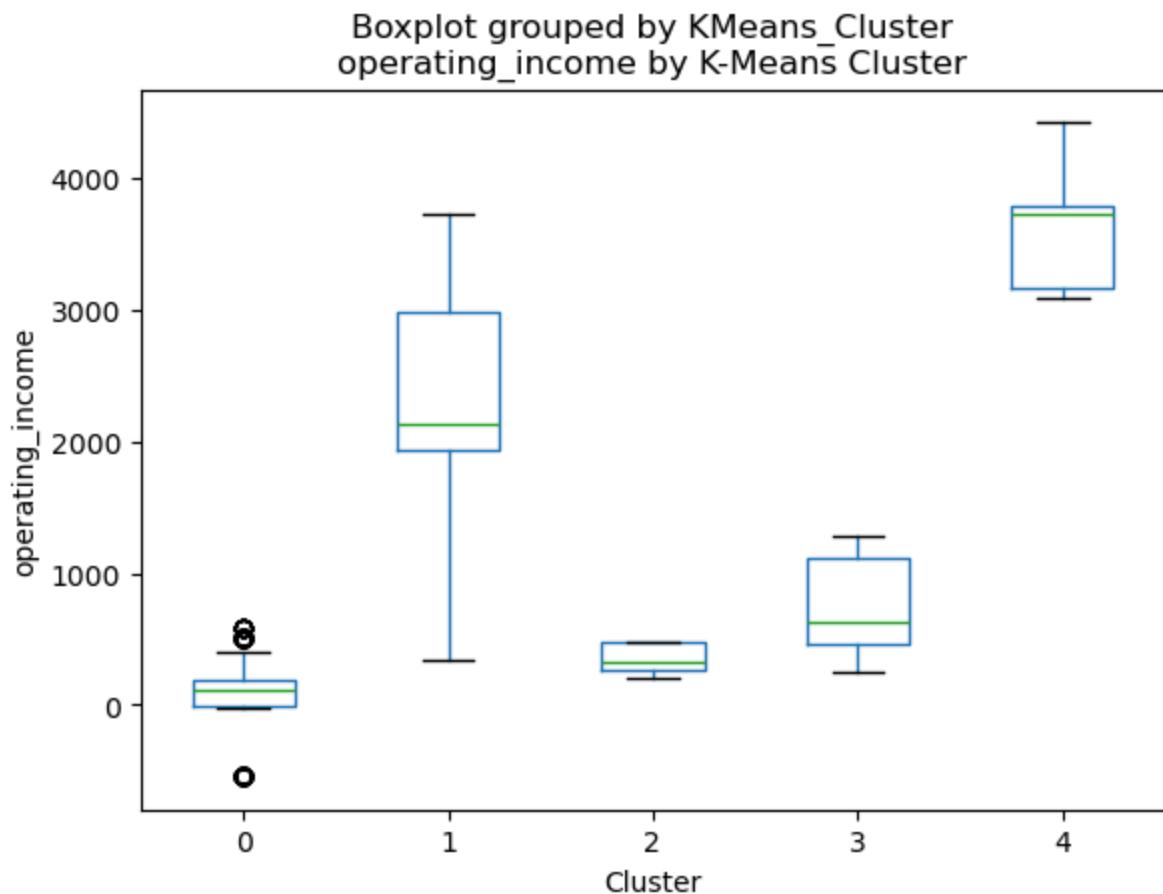
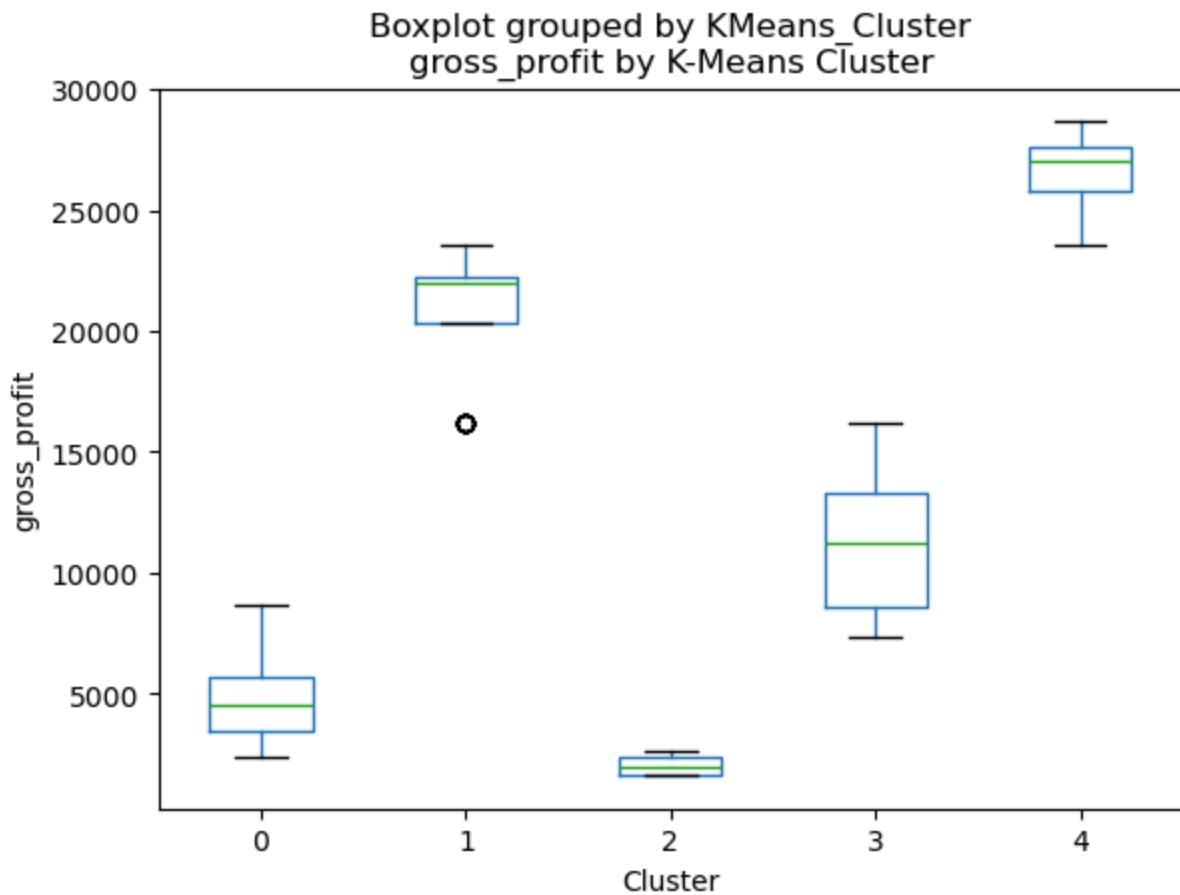


Boxplot grouped by KMeans\_Cluster  
total\_share\_holder\_equity by K-Means Cluster

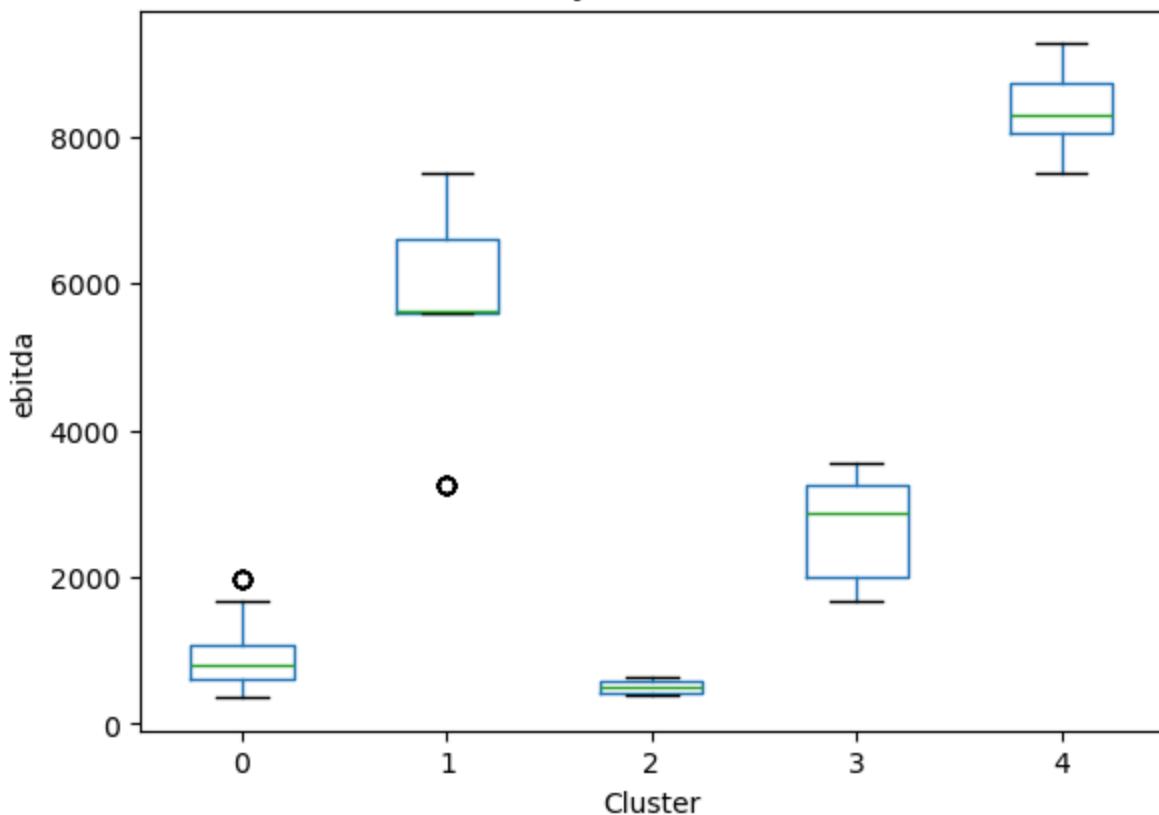


Boxplot grouped by KMeans\_Cluster  
revenue by K-Means Cluster

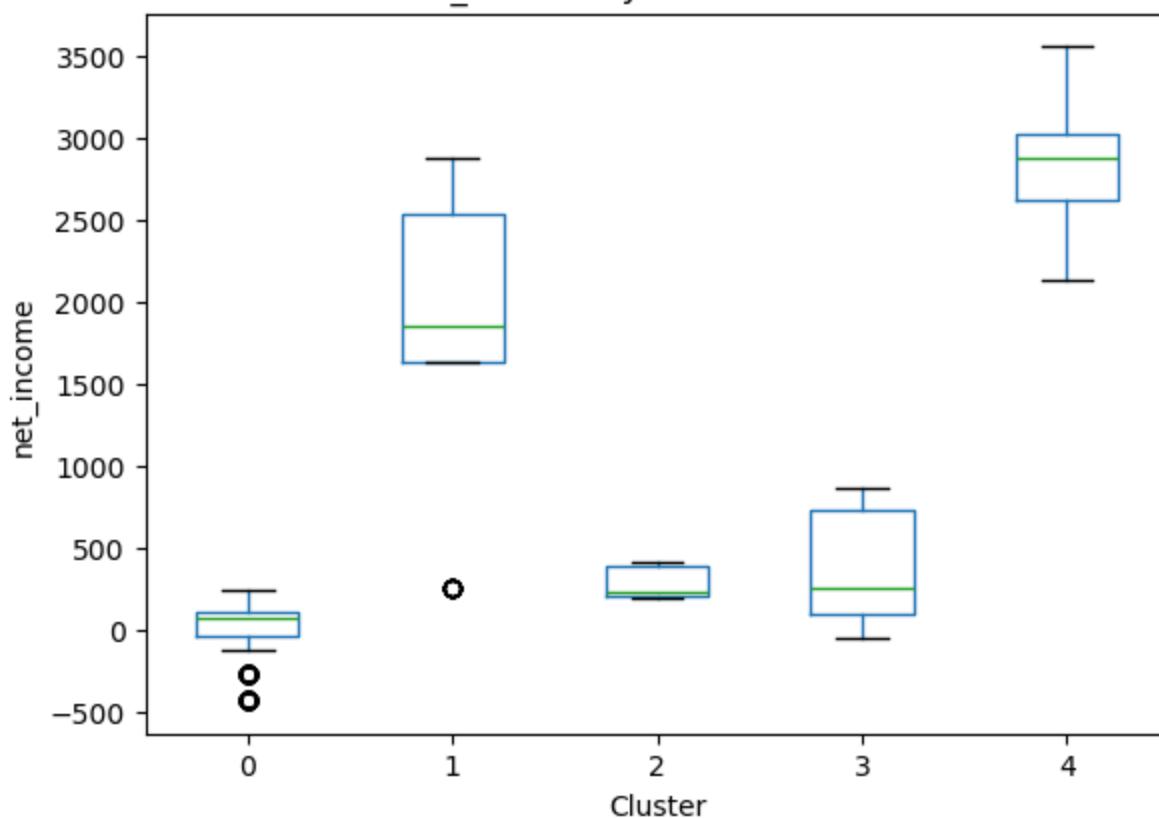




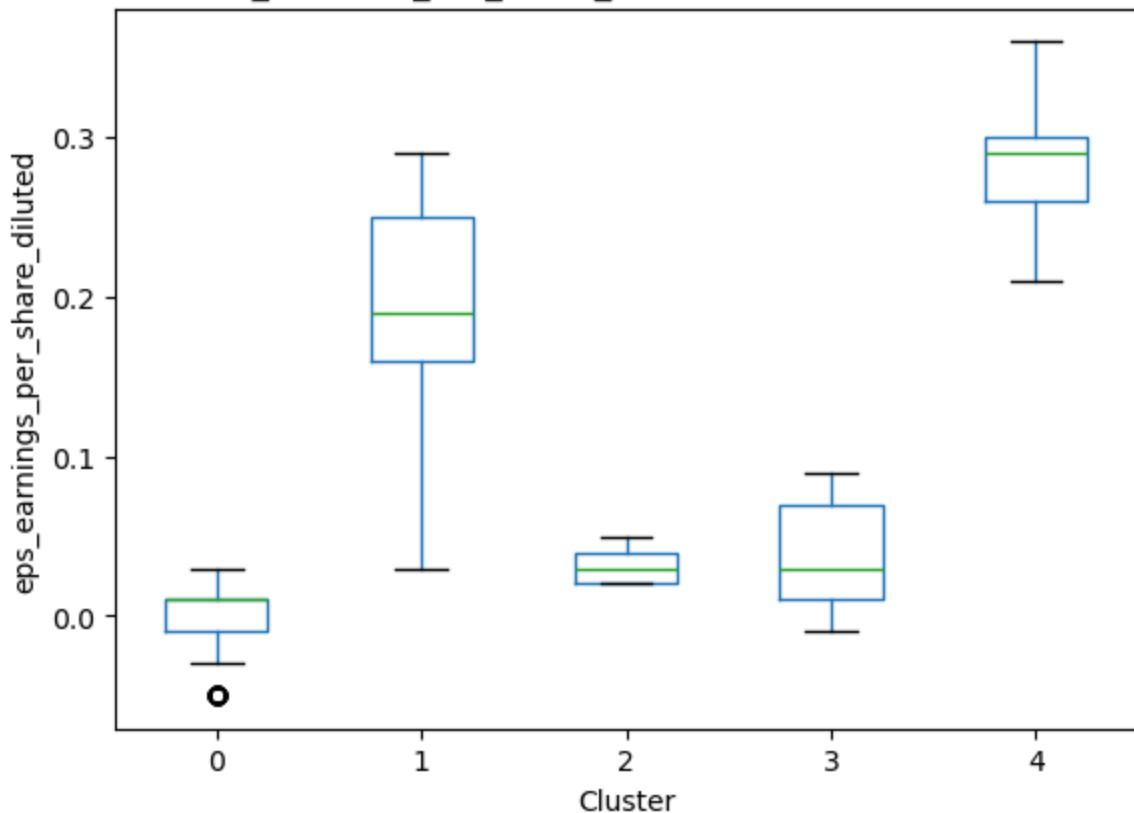
Boxplot grouped by KMeans\_Cluster  
ebitda by K-Means Cluster



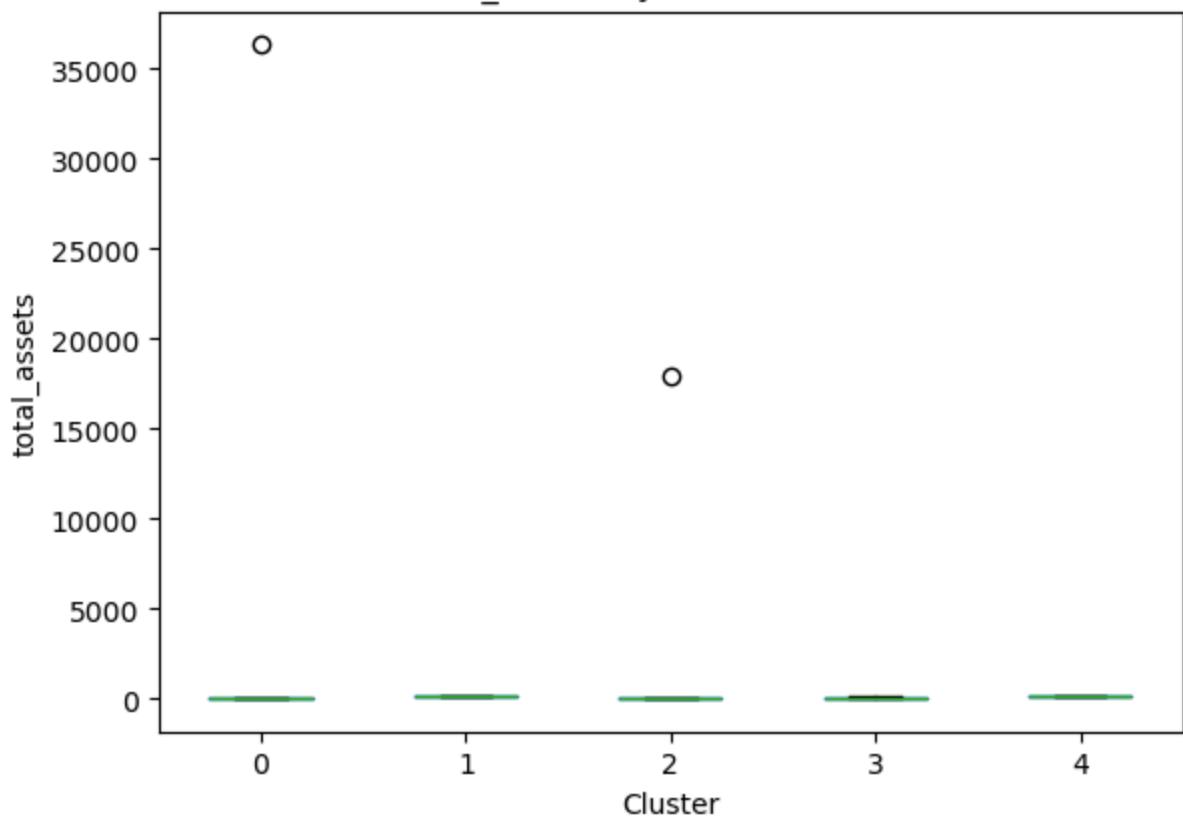
Boxplot grouped by KMeans\_Cluster  
net\_income by K-Means Cluster



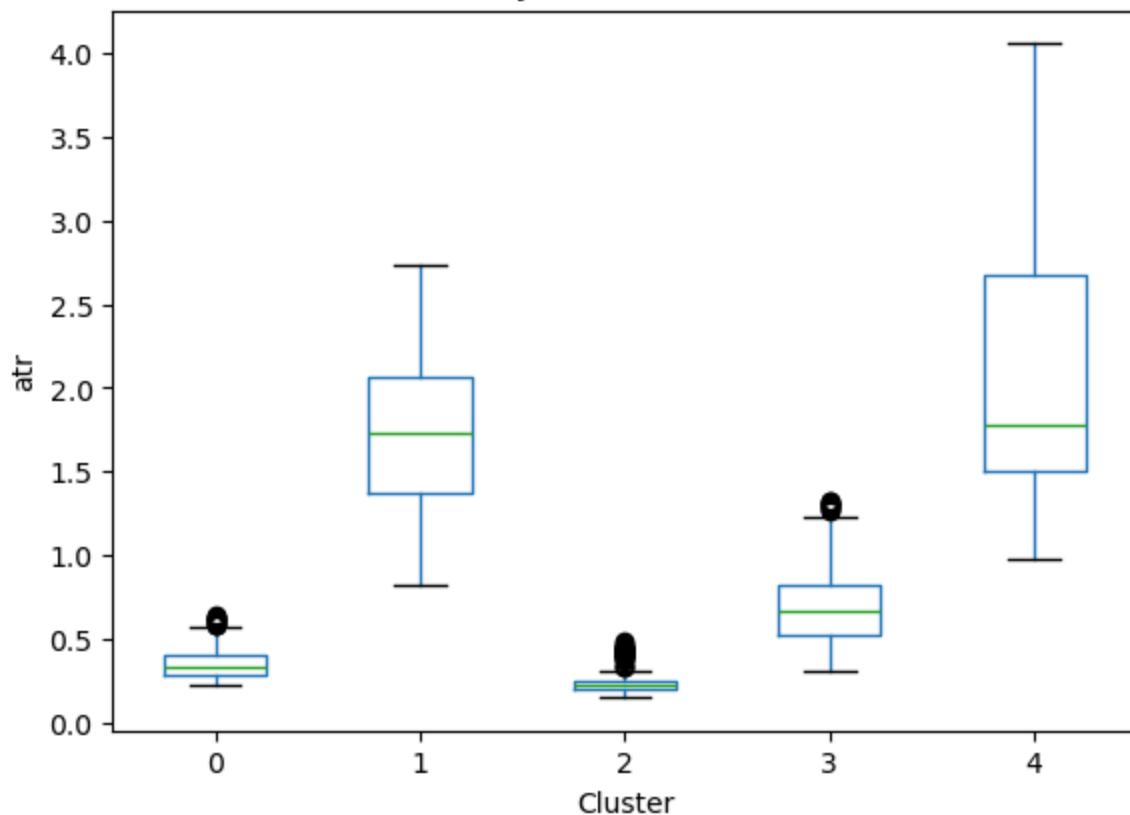
Boxplot grouped by KMeans\_Cluster  
eps\_earnings\_per\_share\_diluted by K-Means Cluster



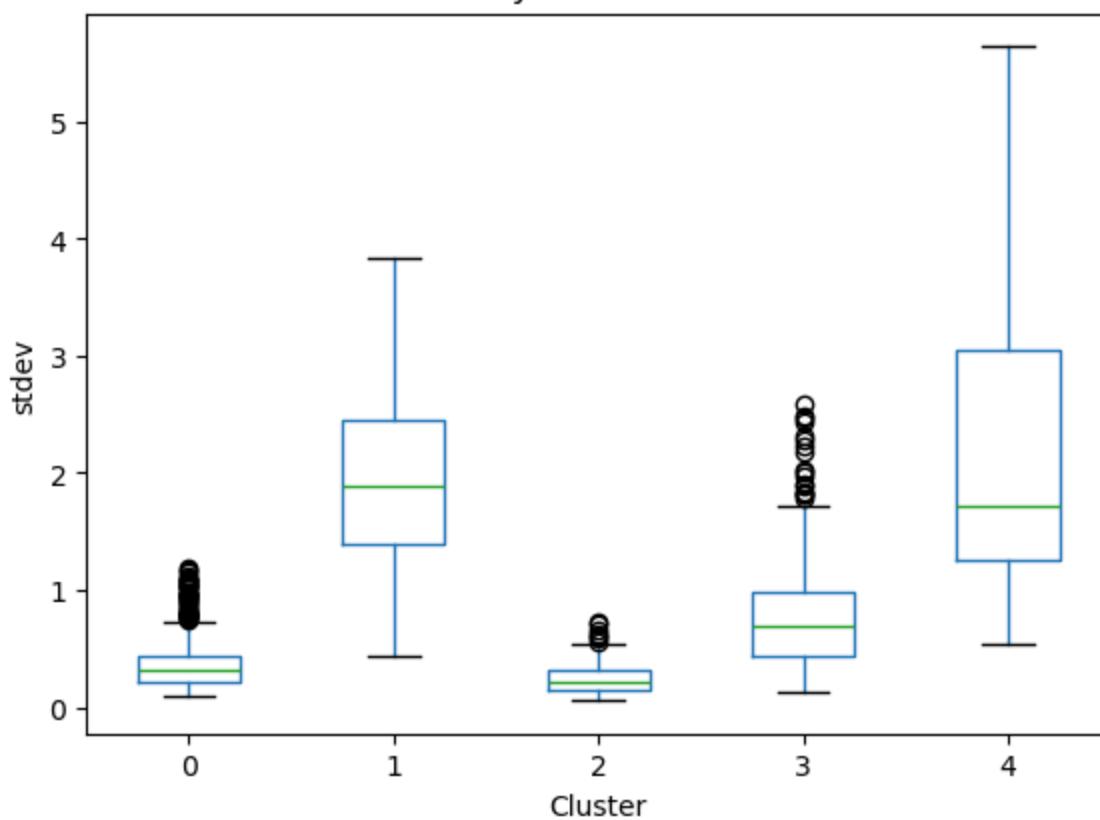
Boxplot grouped by KMeans\_Cluster  
total\_assets by K-Means Cluster



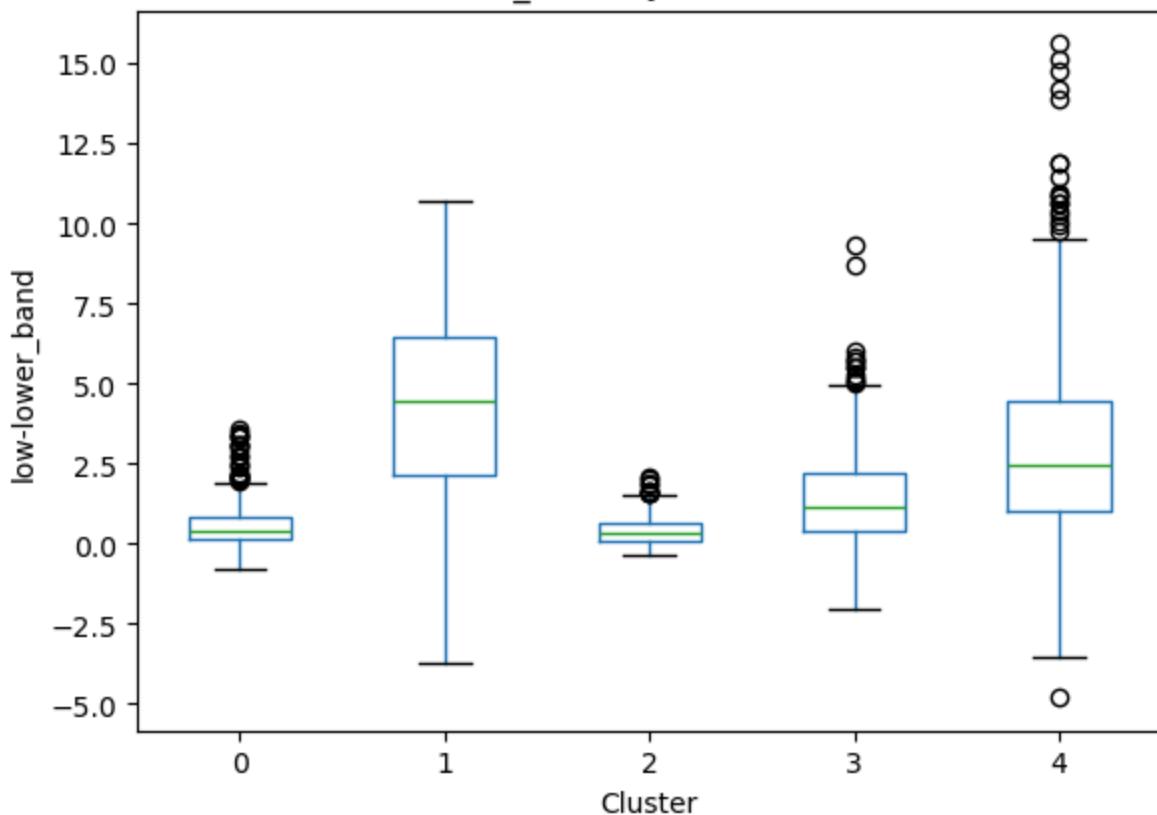
Boxplot grouped by KMeans\_Cluster  
atr by K-Means Cluster



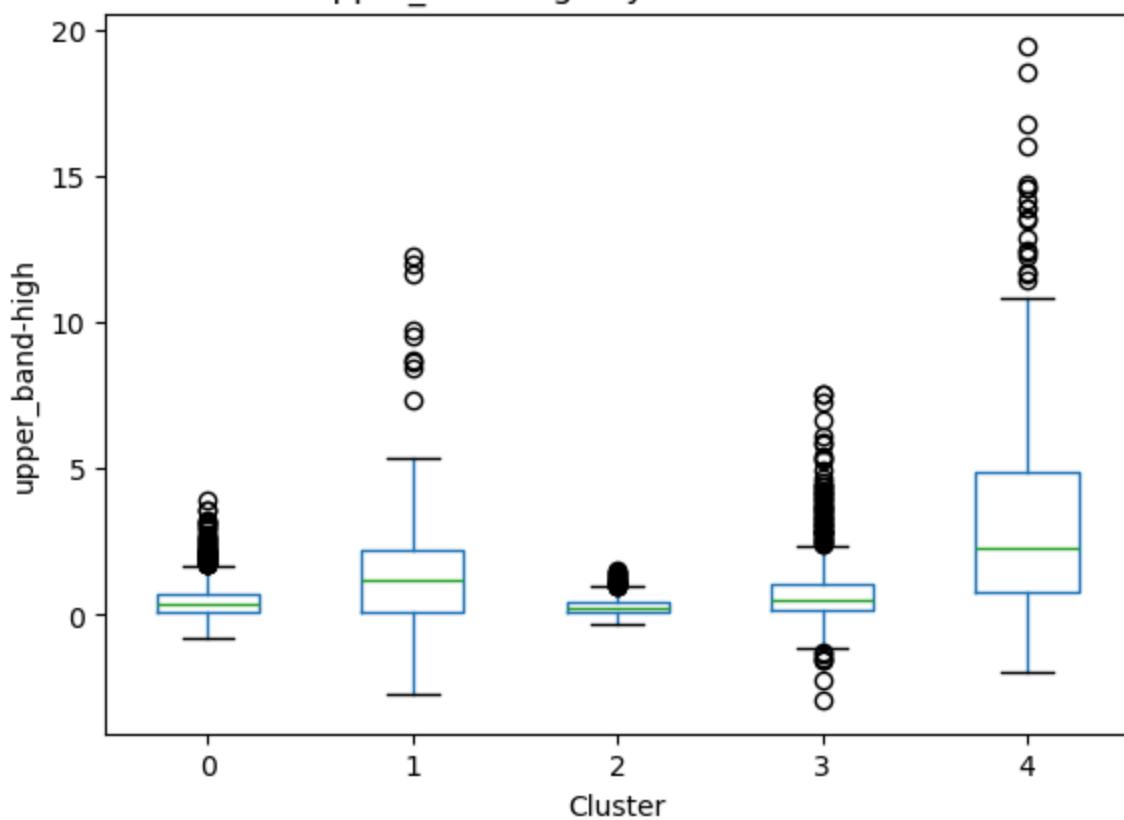
Boxplot grouped by KMeans\_Cluster  
stdev by K-Means Cluster



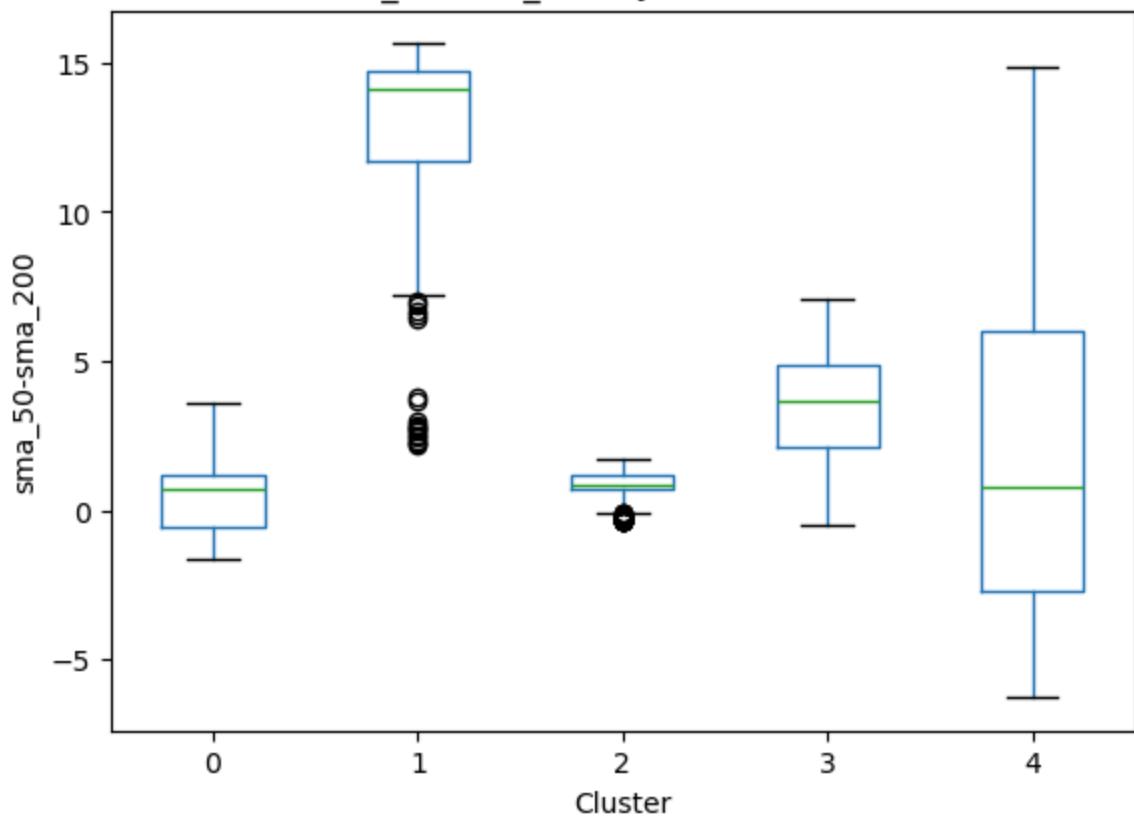
Boxplot grouped by KMeans\_Cluster  
low-lower\_band by K-Means Cluster



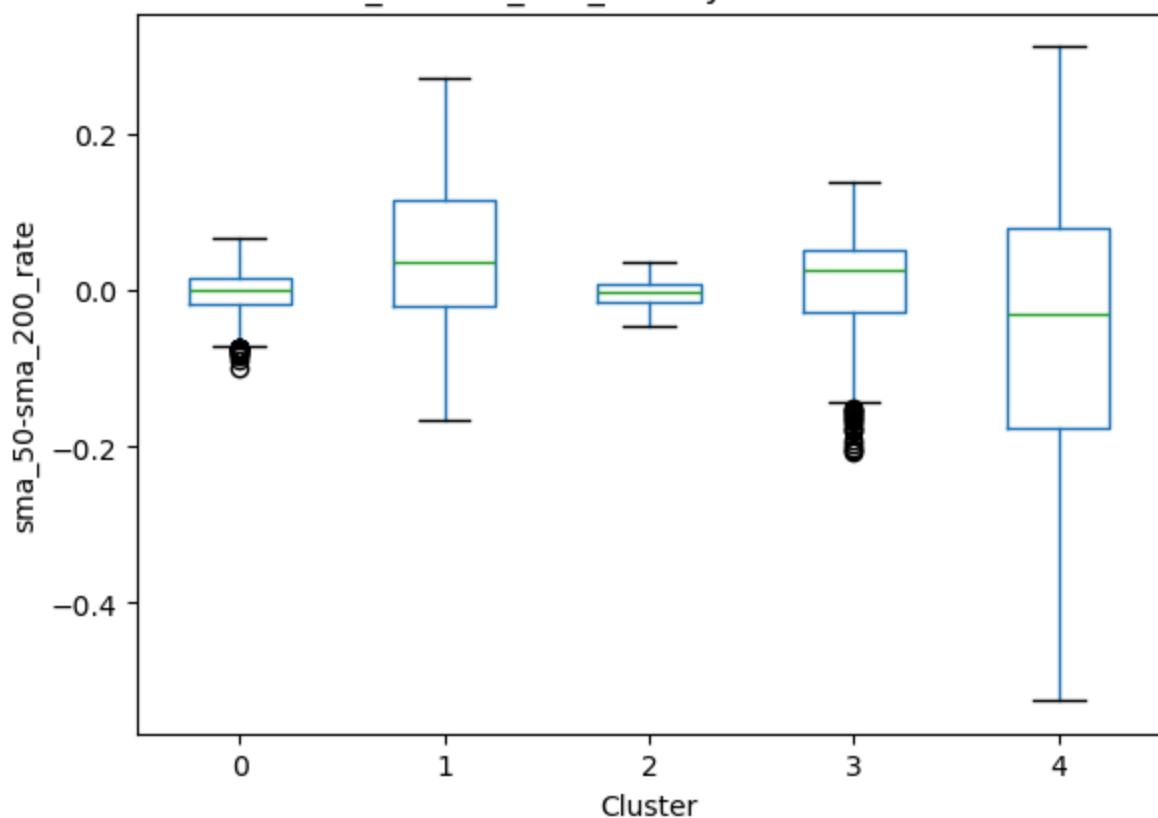
Boxplot grouped by KMeans\_Cluster  
upper\_band-high by K-Means Cluster



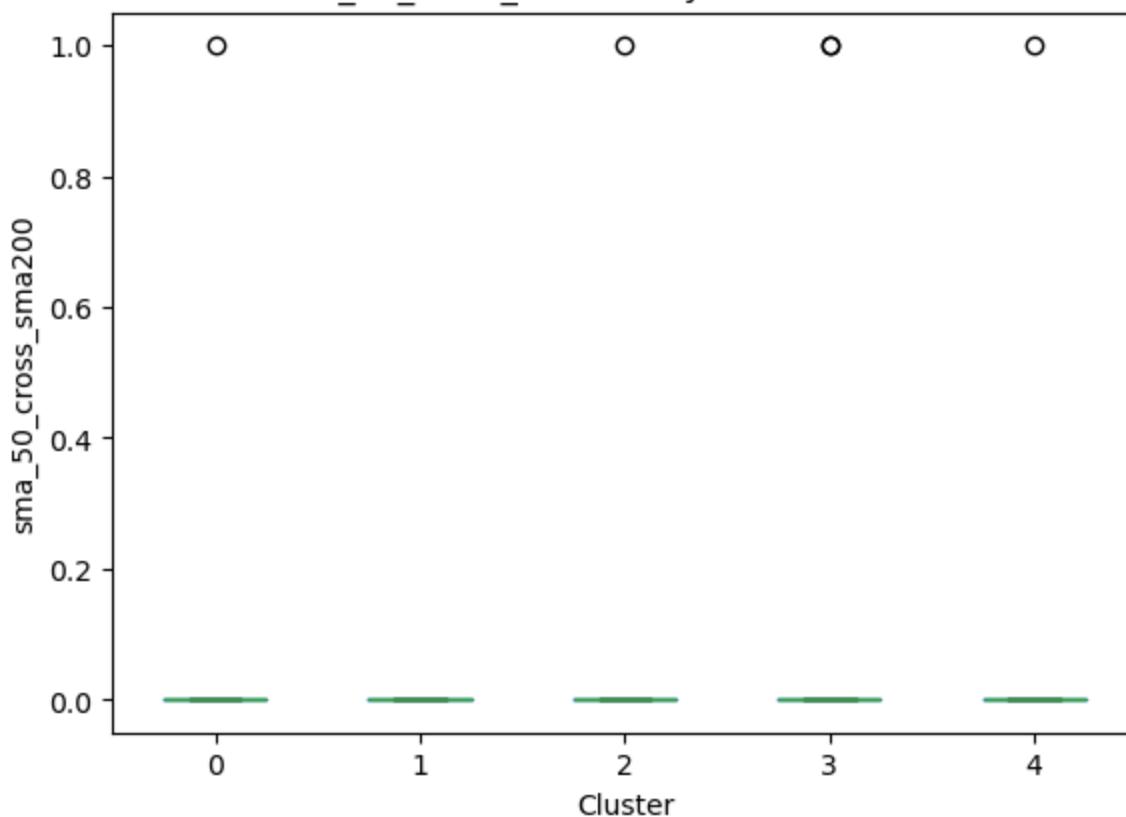
Boxplot grouped by KMeans\_Cluster  
sma\_50-sma\_200 by K-Means Cluster



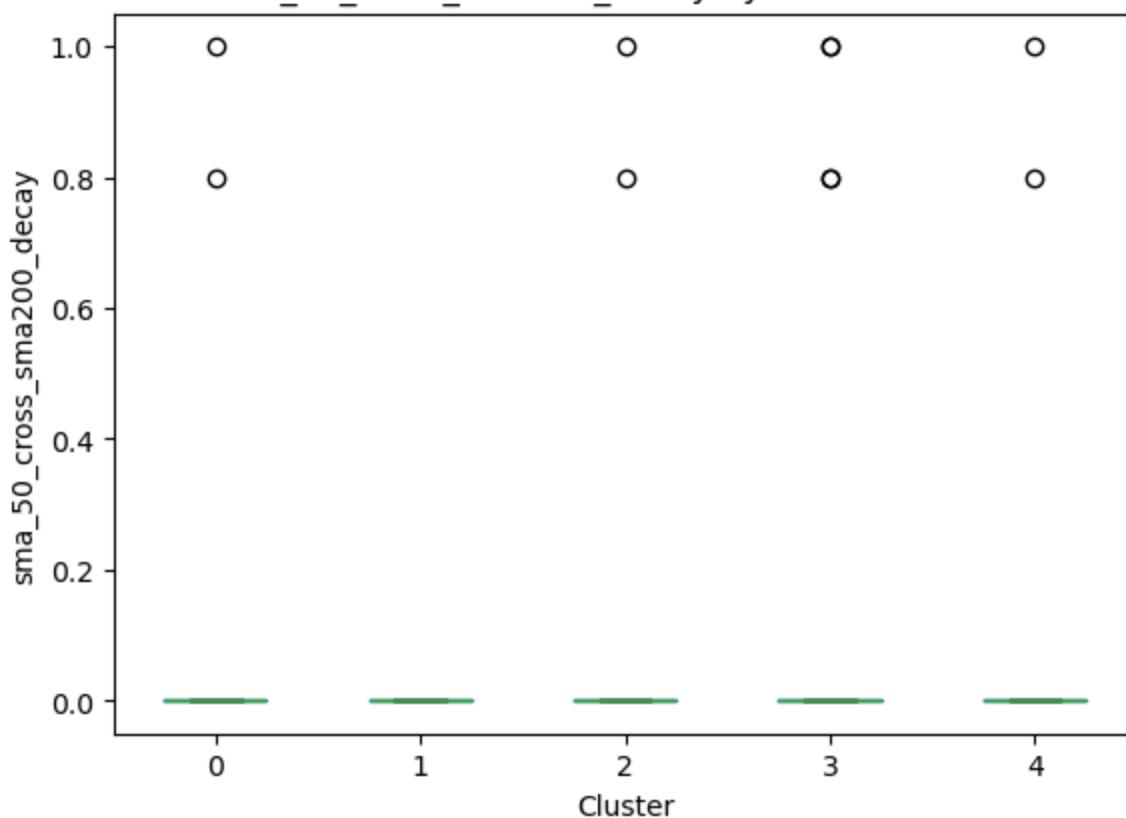
Boxplot grouped by KMeans\_Cluster  
sma\_50-sma\_200\_rate by K-Means Cluster



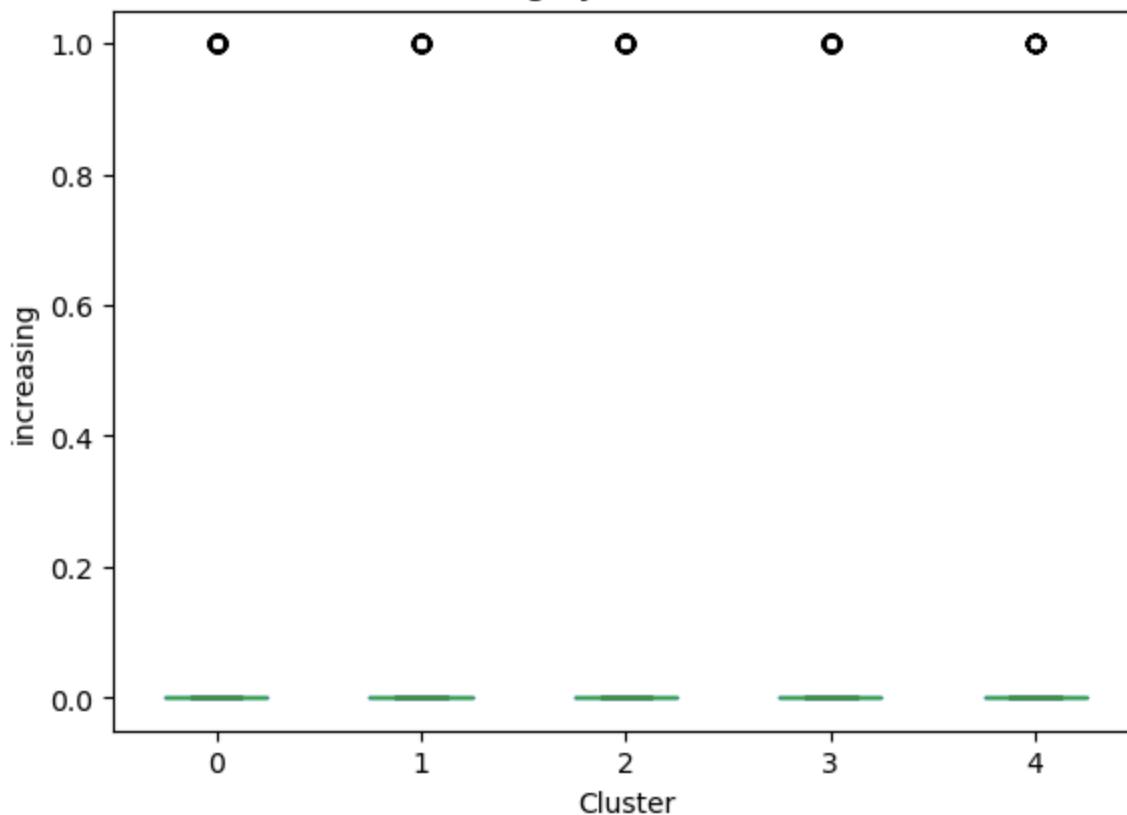
Boxplot grouped by KMeans\_Cluster  
sma\_50\_cross\_sma200 by K-Means Cluster



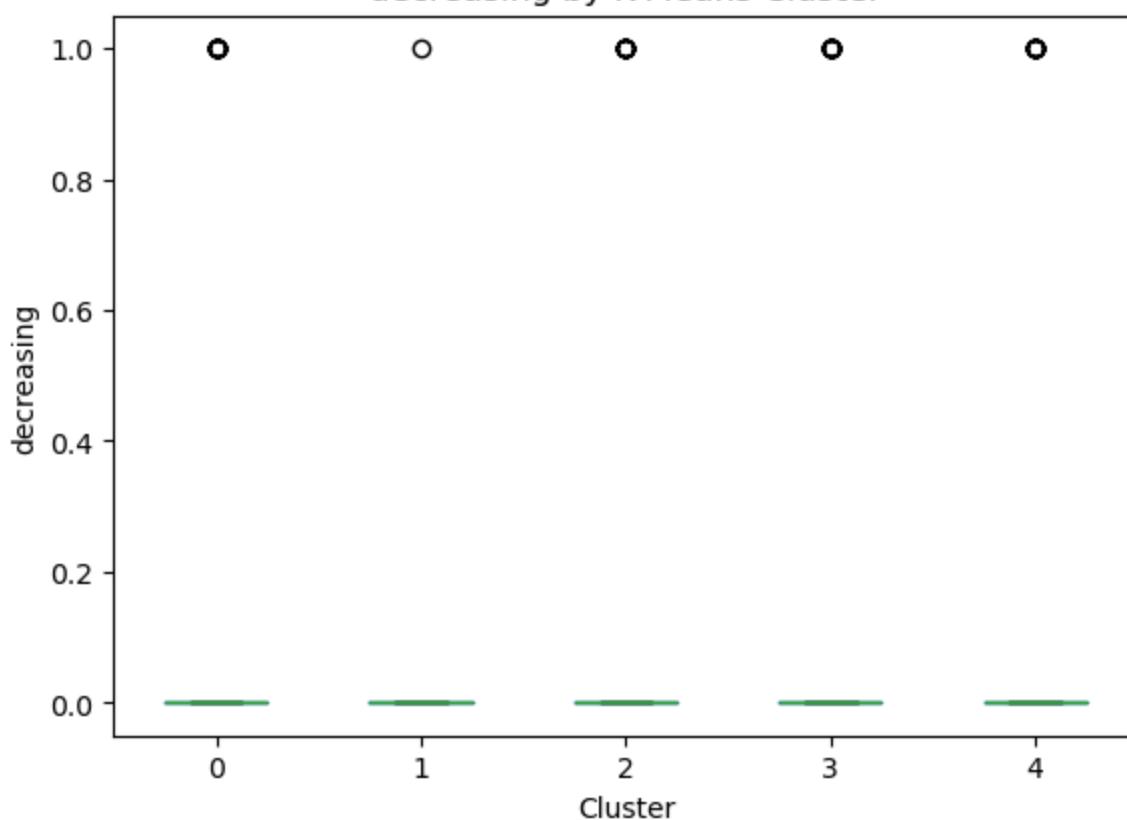
Boxplot grouped by KMeans\_Cluster  
sma\_50\_cross\_sma200\_decay by K-Means Cluster



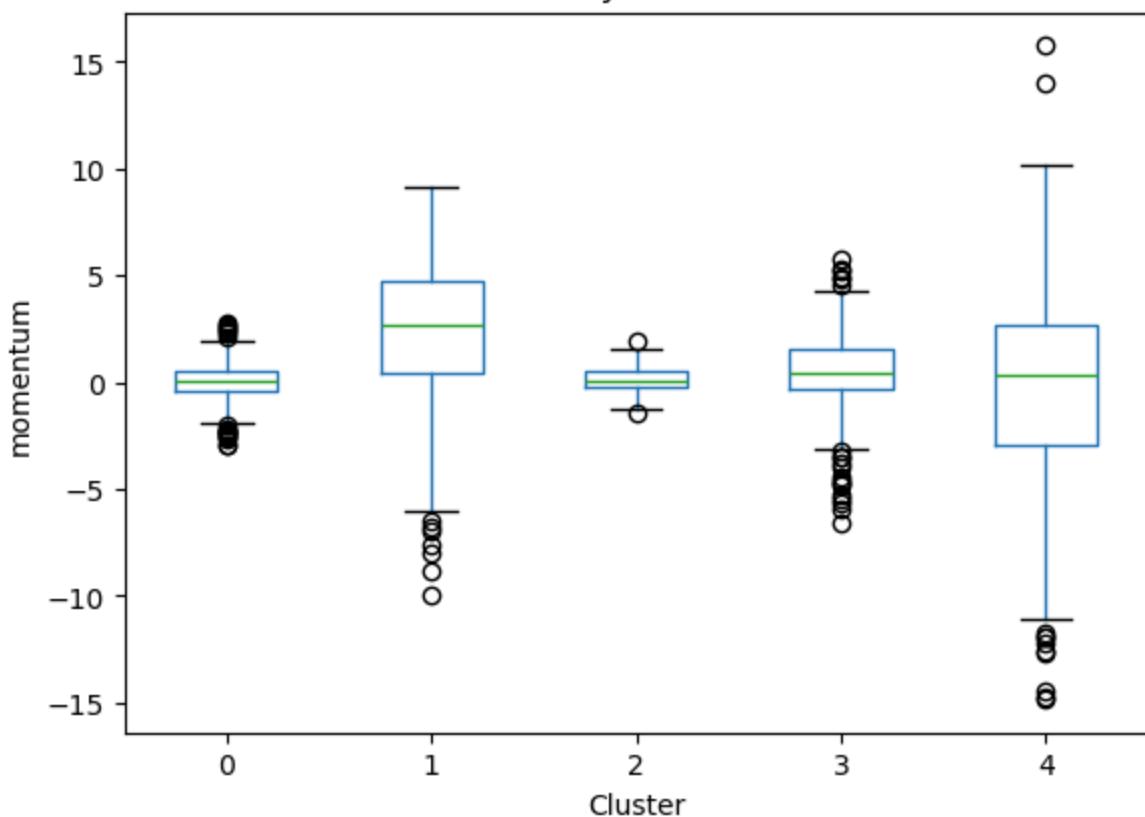
Boxplot grouped by KMeans\_Cluster  
increasing by K-Means Cluster



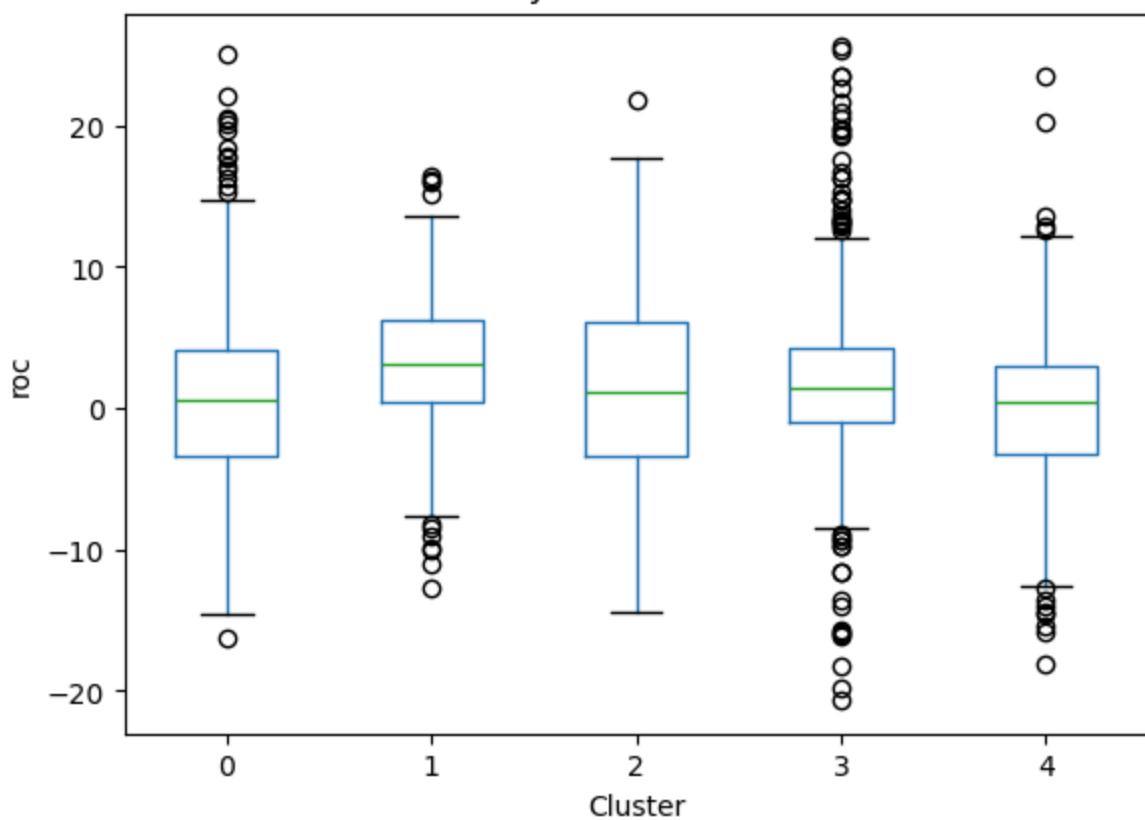
Boxplot grouped by KMeans\_Cluster  
decreasing by K-Means Cluster



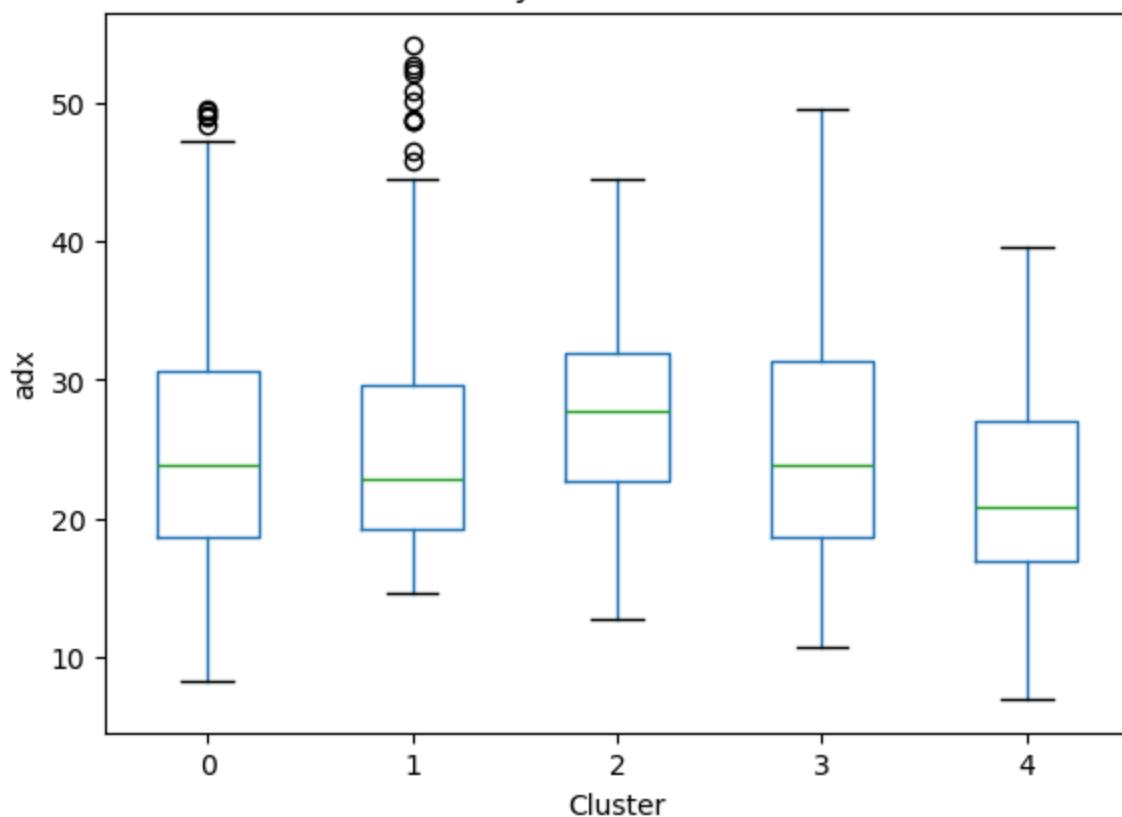
Boxplot grouped by KMeans\_Cluster  
momentum by K-Means Cluster



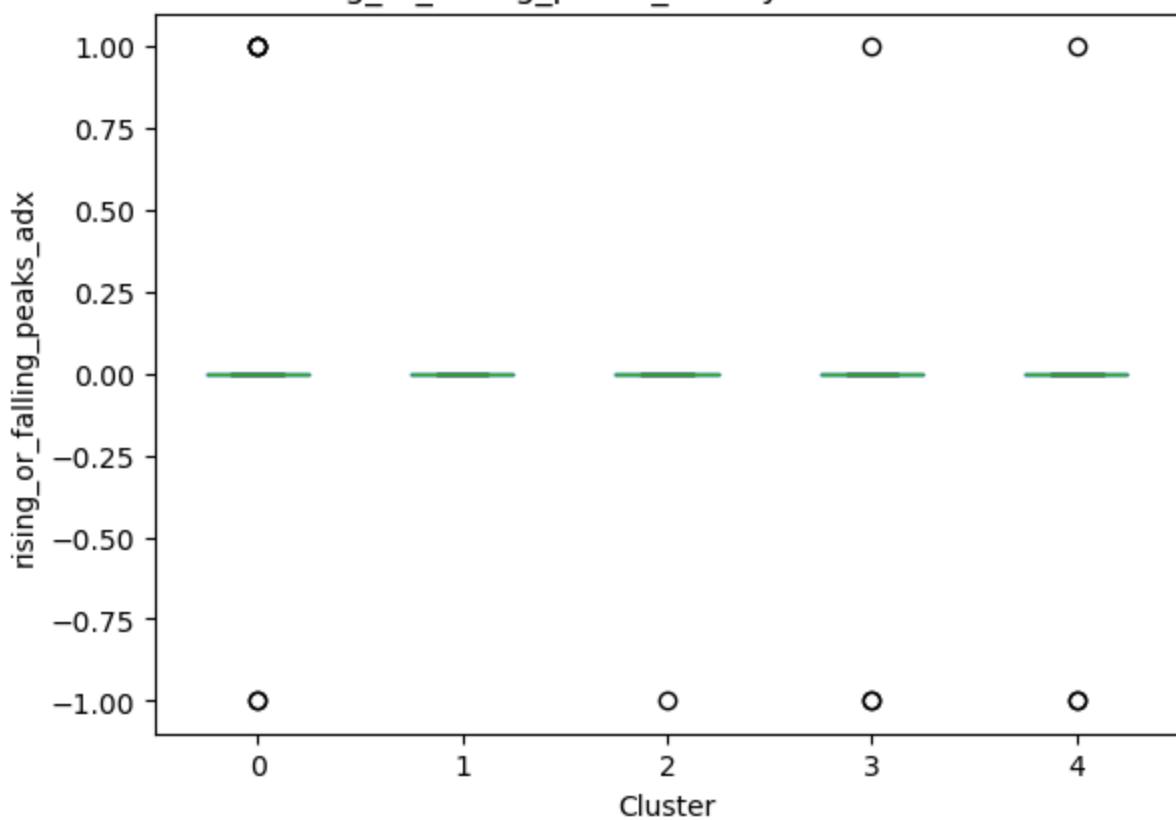
Boxplot grouped by KMeans\_Cluster  
roc by K-Means Cluster



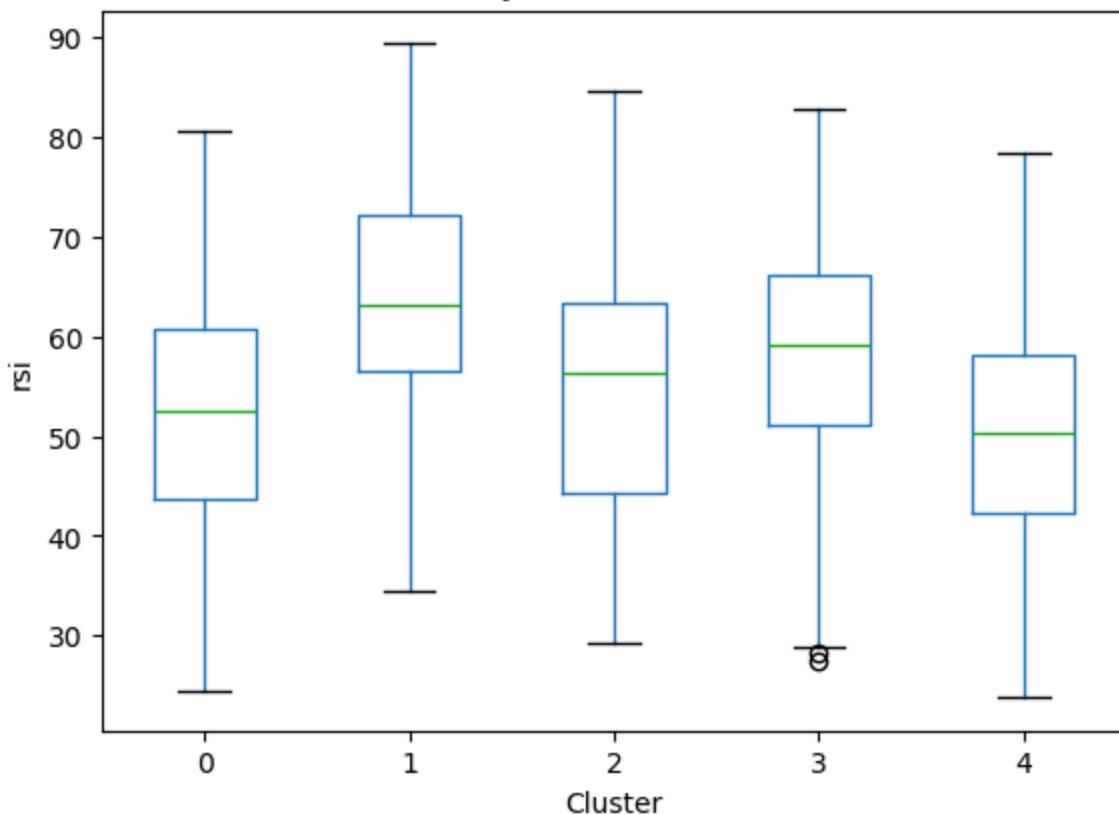
Boxplot grouped by KMeans\_Cluster  
adx by K-Means Cluster



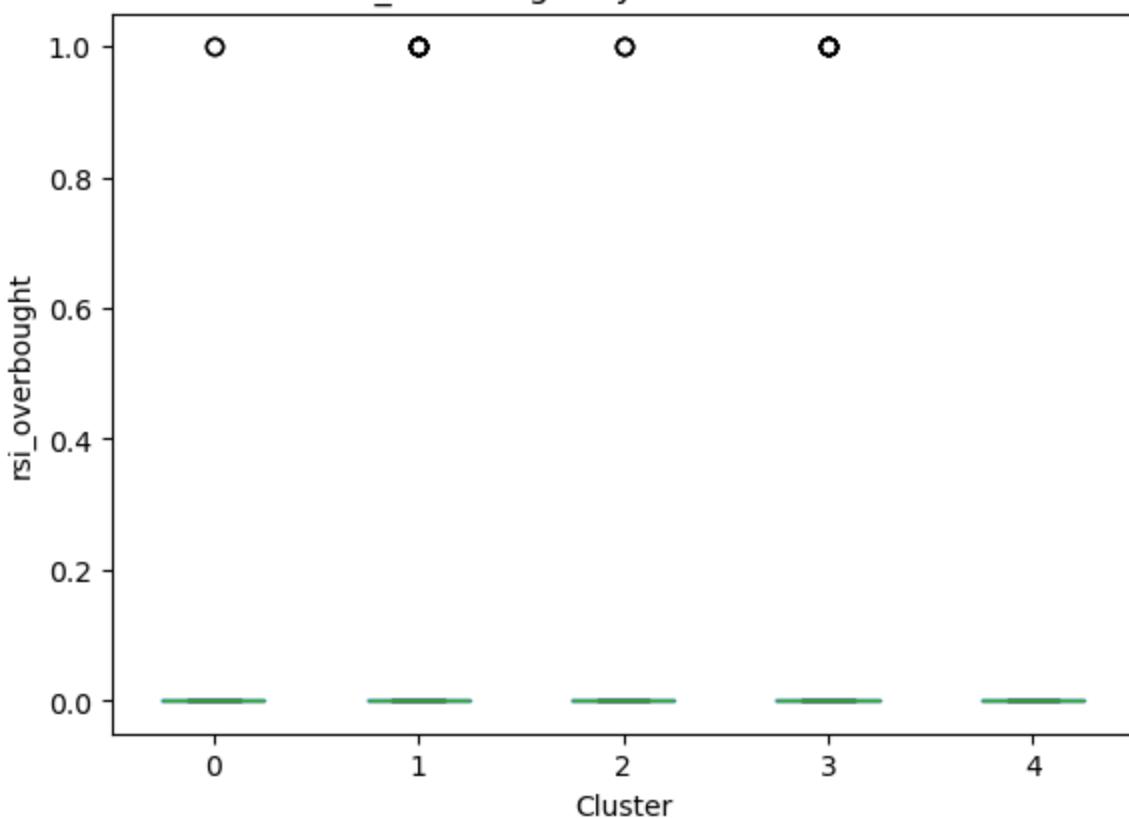
Boxplot grouped by KMeans\_Cluster  
rising\_or\_falling\_peaks\_adx by K-Means Cluster



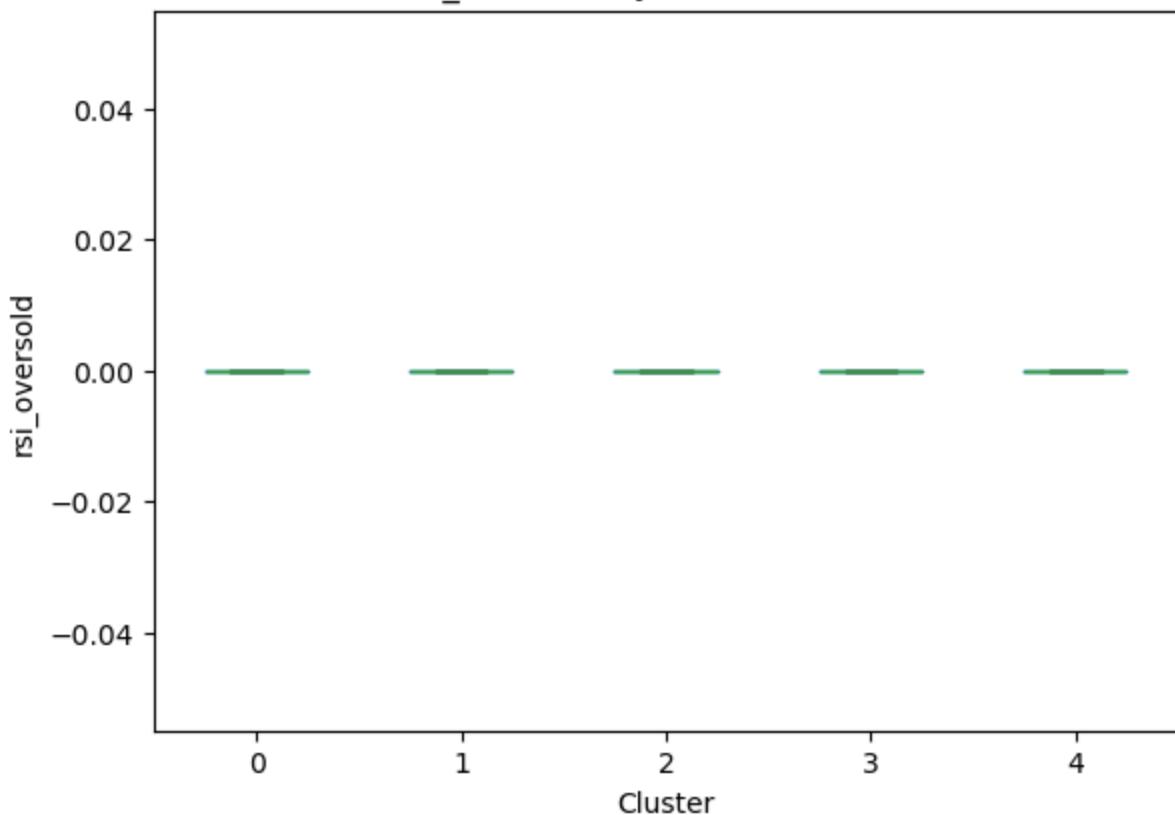
Boxplot grouped by KMeans\_Cluster  
rsi by K-Means Cluster



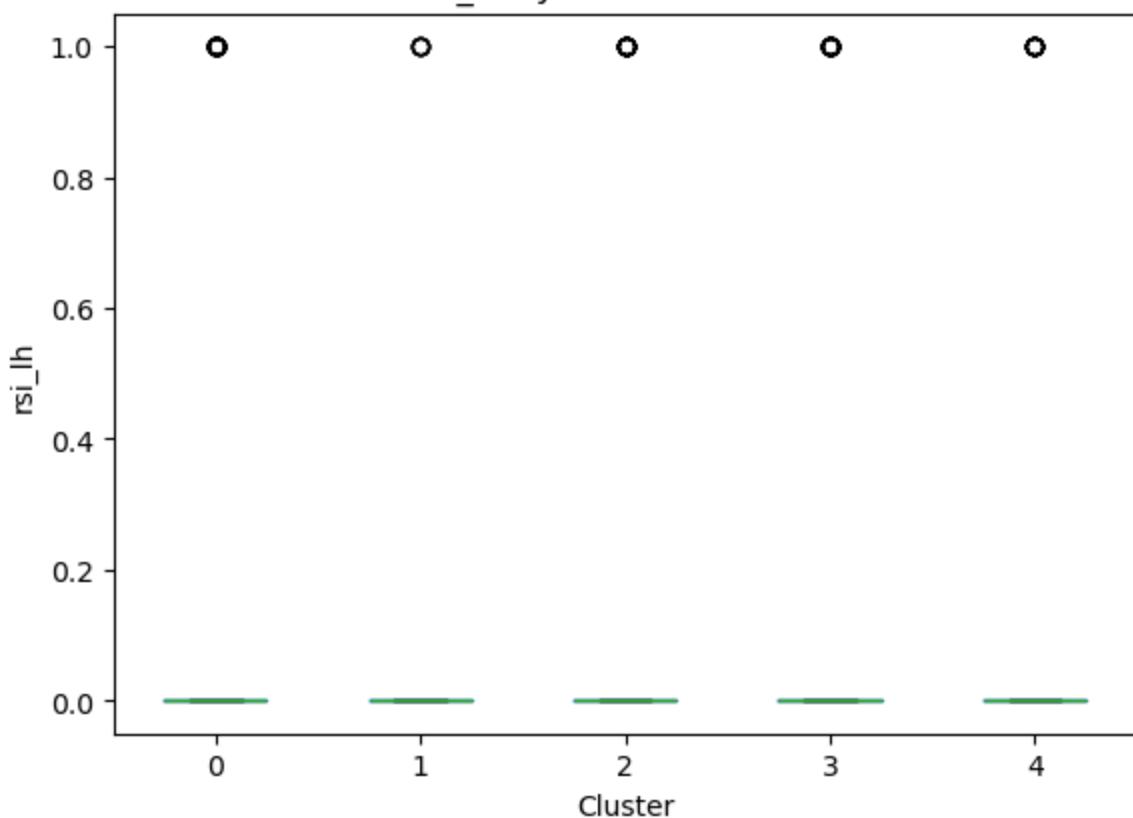
Boxplot grouped by KMeans\_Cluster  
rsi\_overbought by K-Means Cluster



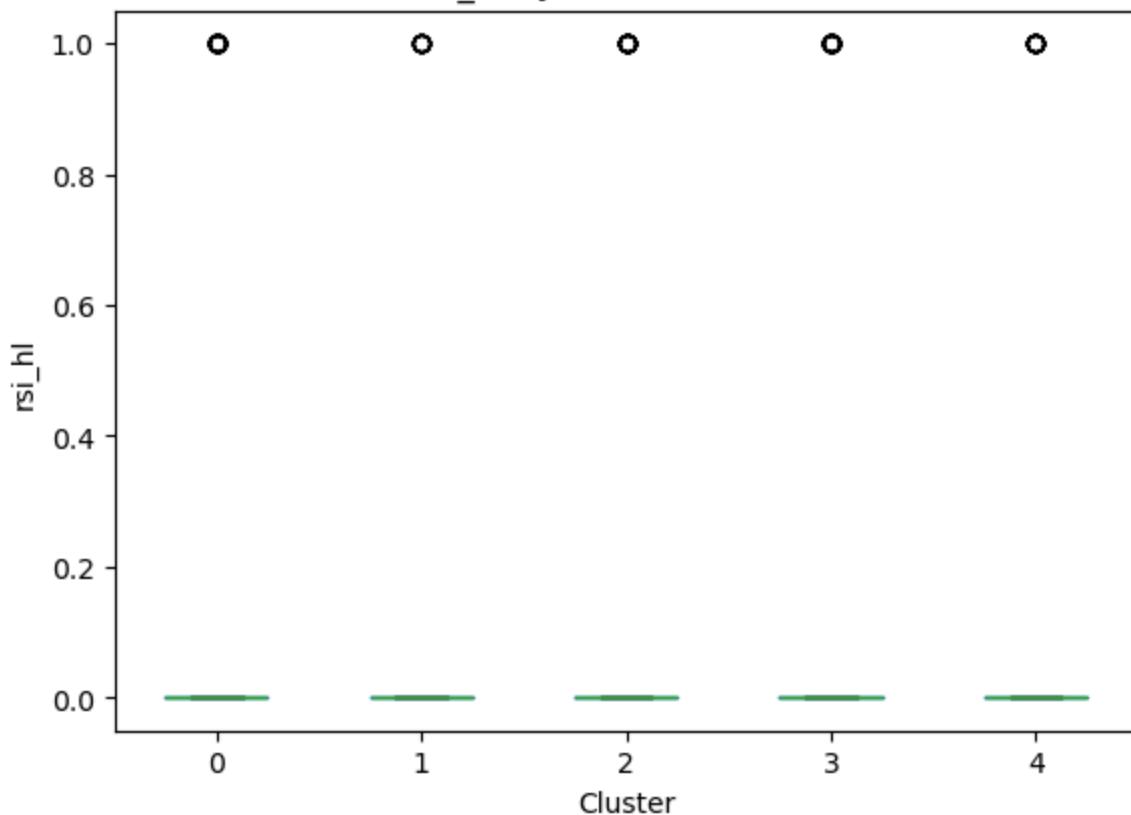
Boxplot grouped by KMeans\_Cluster  
rsi\_oversold by K-Means Cluster



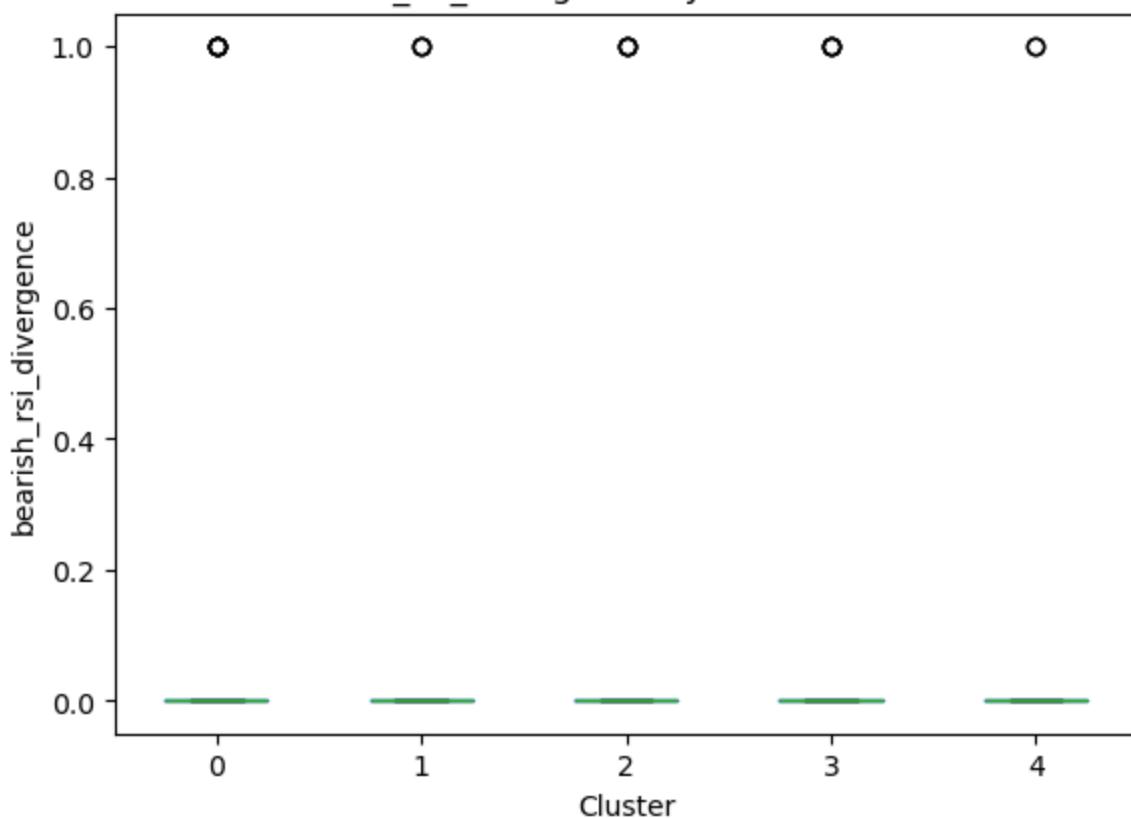
Boxplot grouped by KMeans\_Cluster  
rsi\_lh by K-Means Cluster



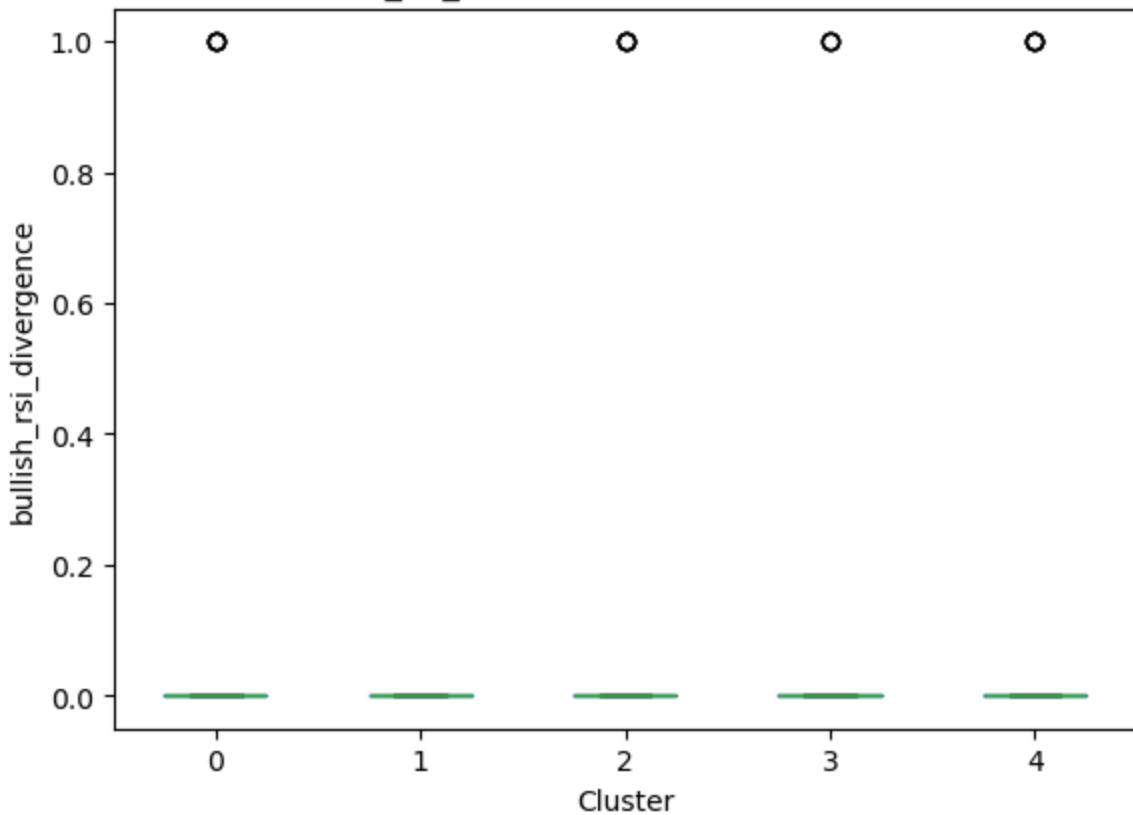
Boxplot grouped by KMeans\_Cluster  
rsi\_hl by K-Means Cluster



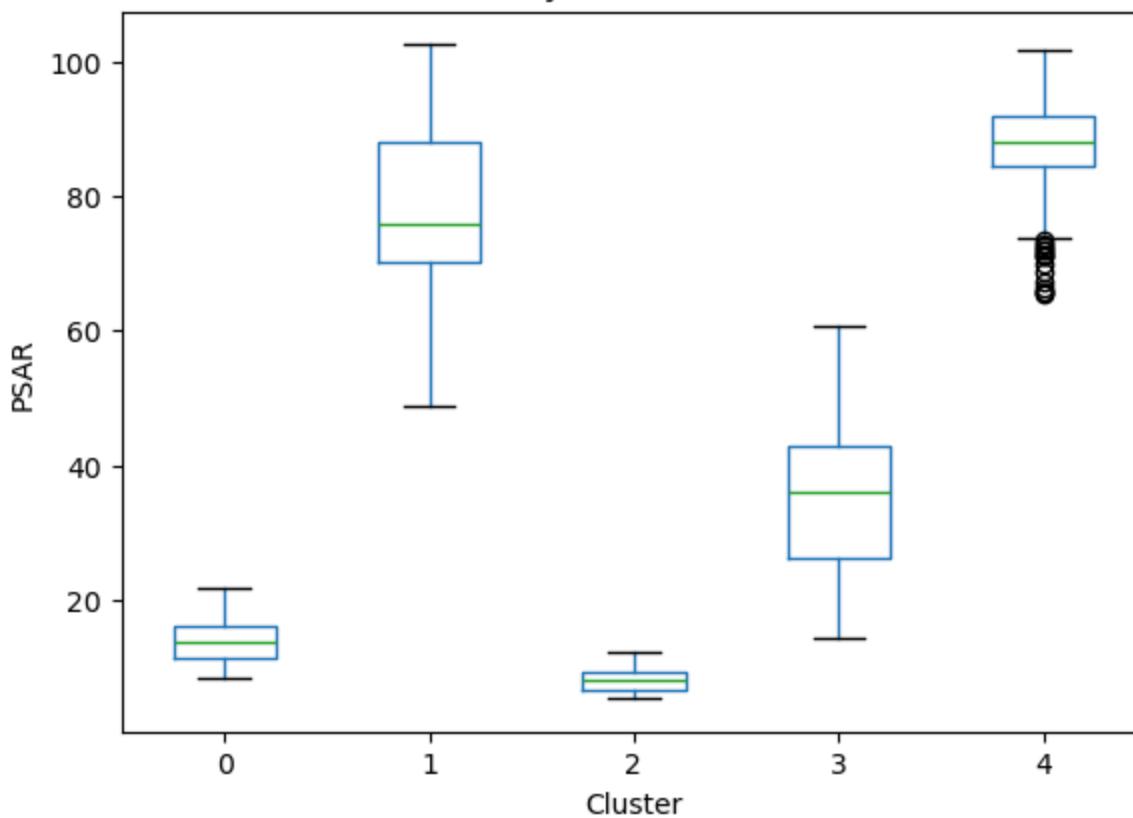
Boxplot grouped by KMeans\_Cluster  
bearish\_rsi\_divergence by K-Means Cluster



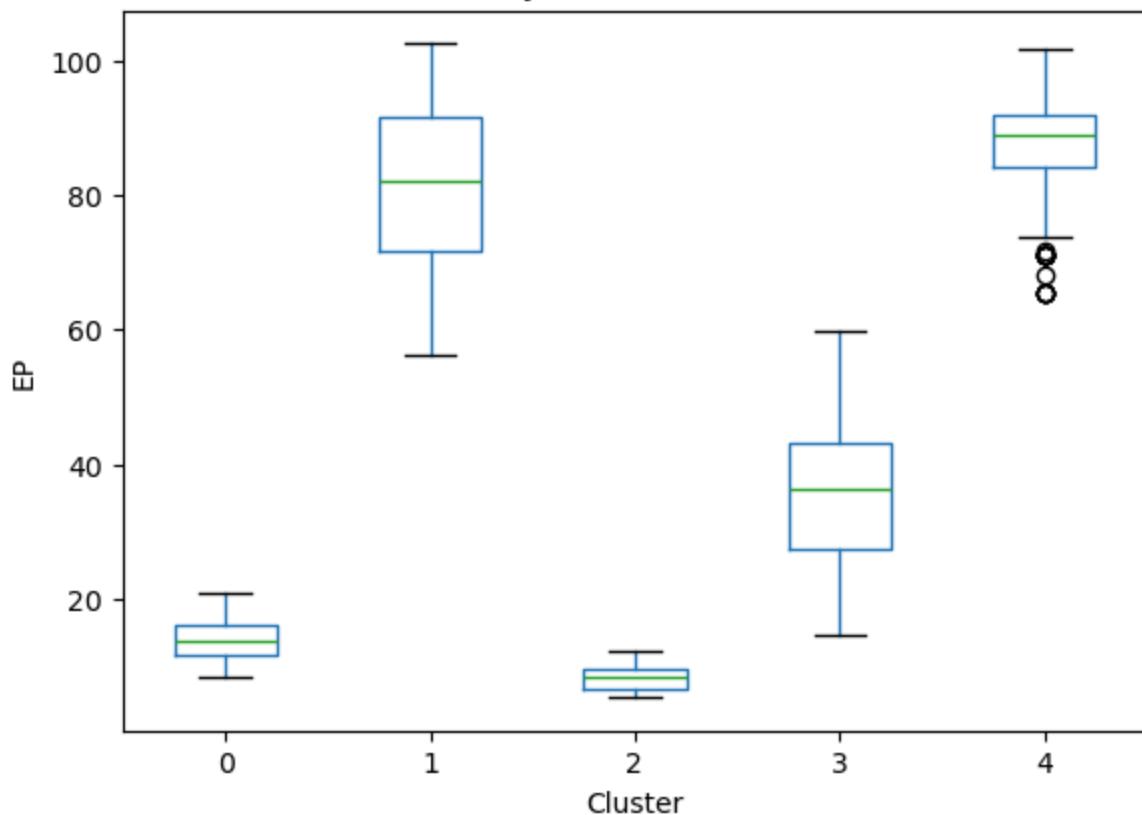
Boxplot grouped by KMeans\_Cluster  
bullish\_rsi\_divergence by K-Means Cluster



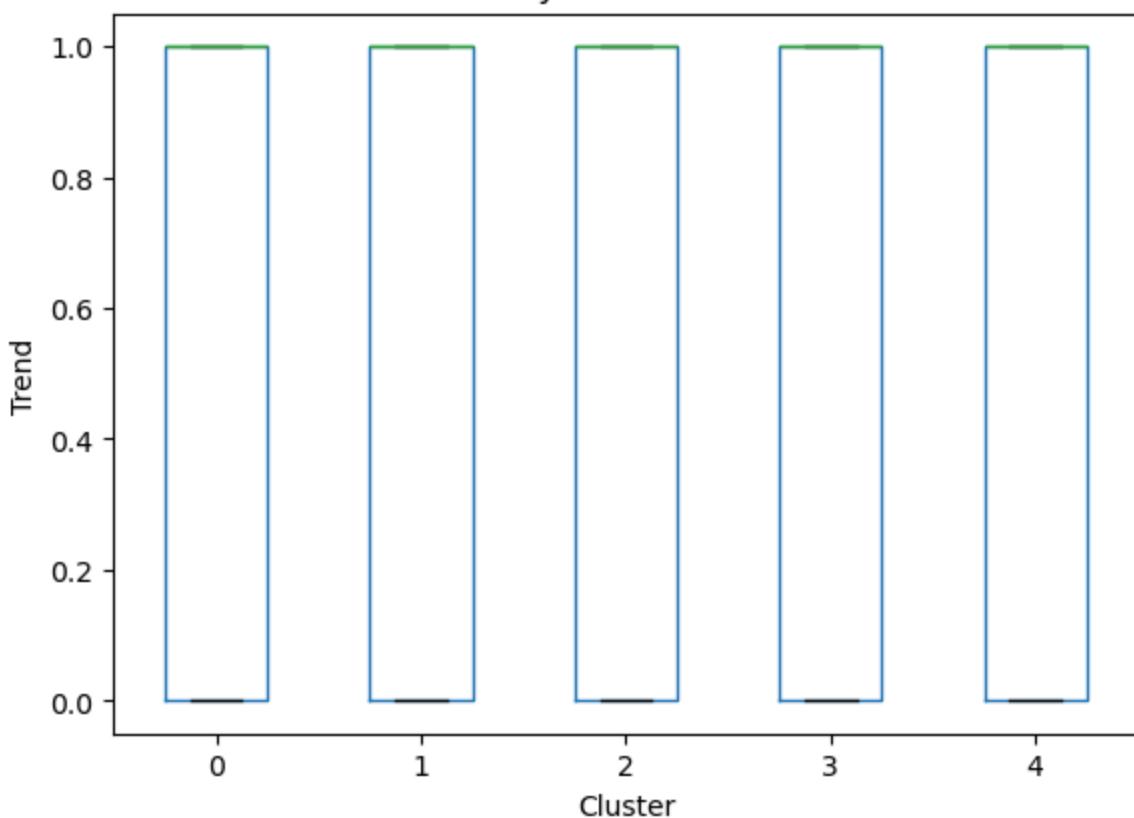
Boxplot grouped by KMeans\_Cluster  
PSAR by K-Means Cluster



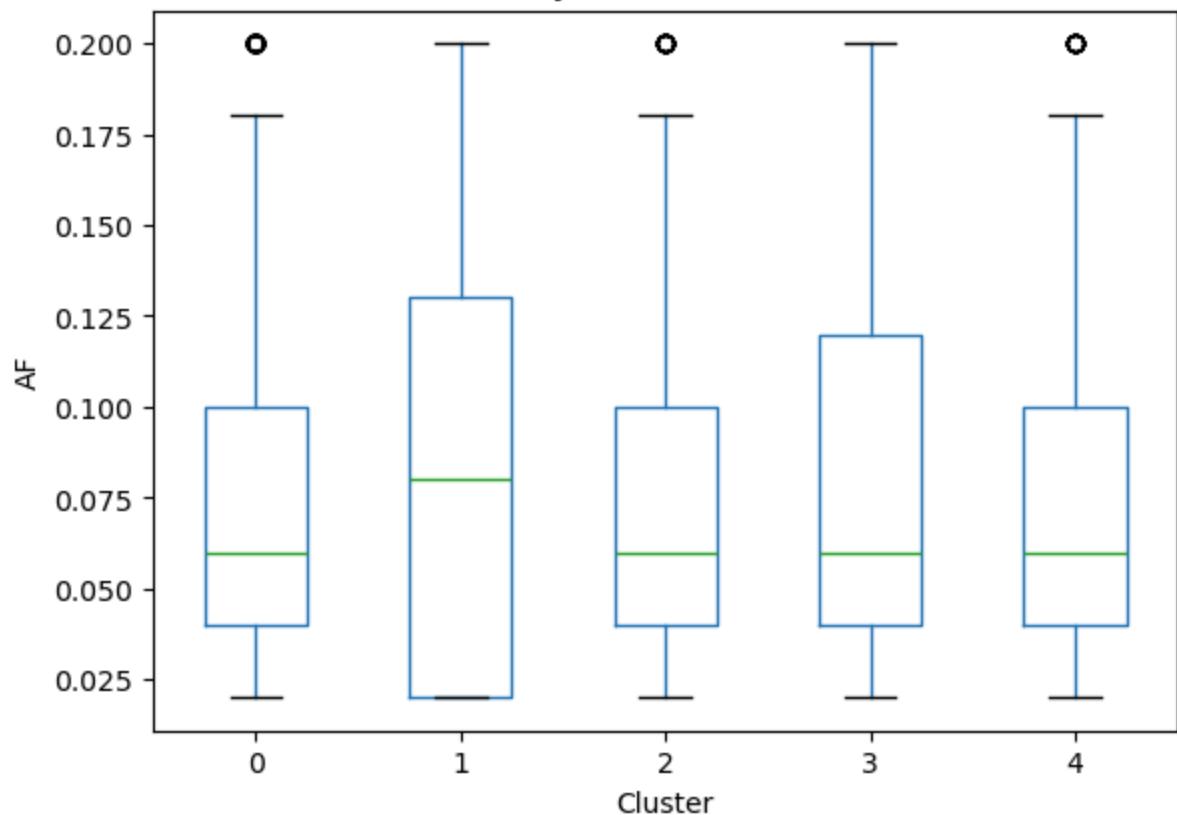
Boxplot grouped by KMeans\_Cluster  
EP by K-Means Cluster



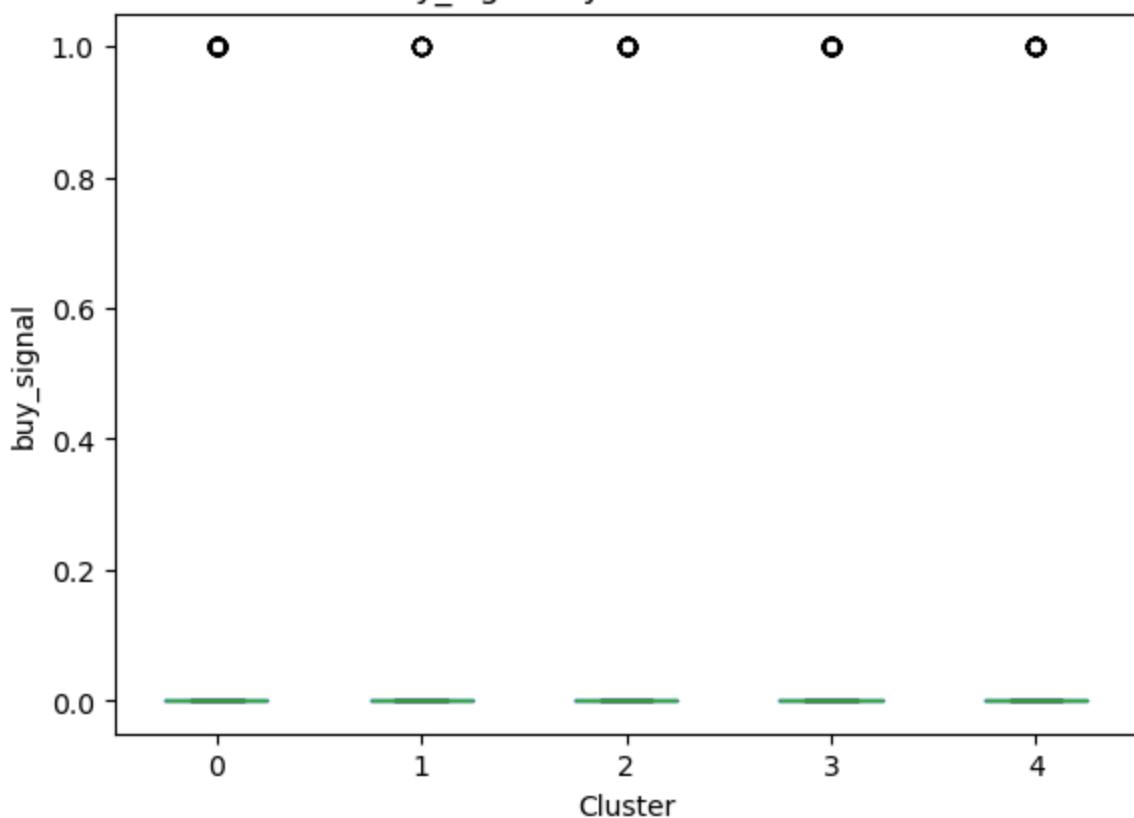
Boxplot grouped by KMeans\_Cluster  
Trend by K-Means Cluster



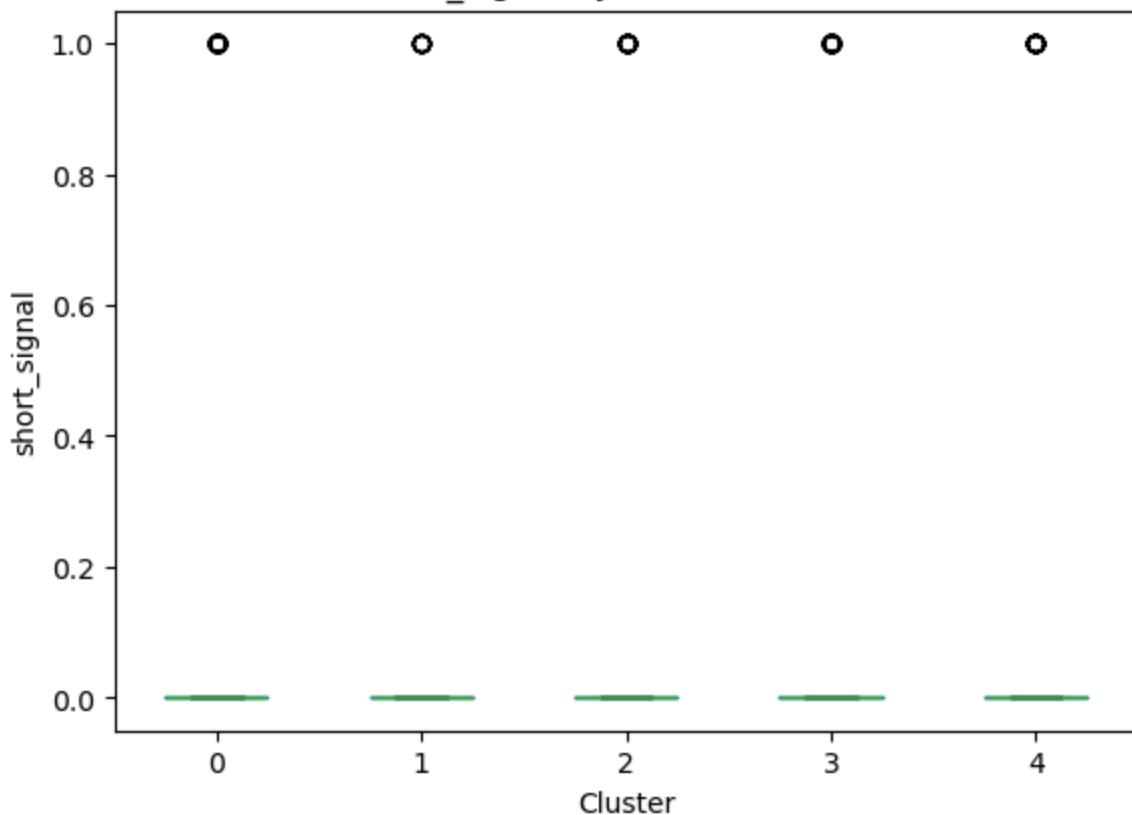
Boxplot grouped by KMeans\_Cluster  
AF by K-Means Cluster



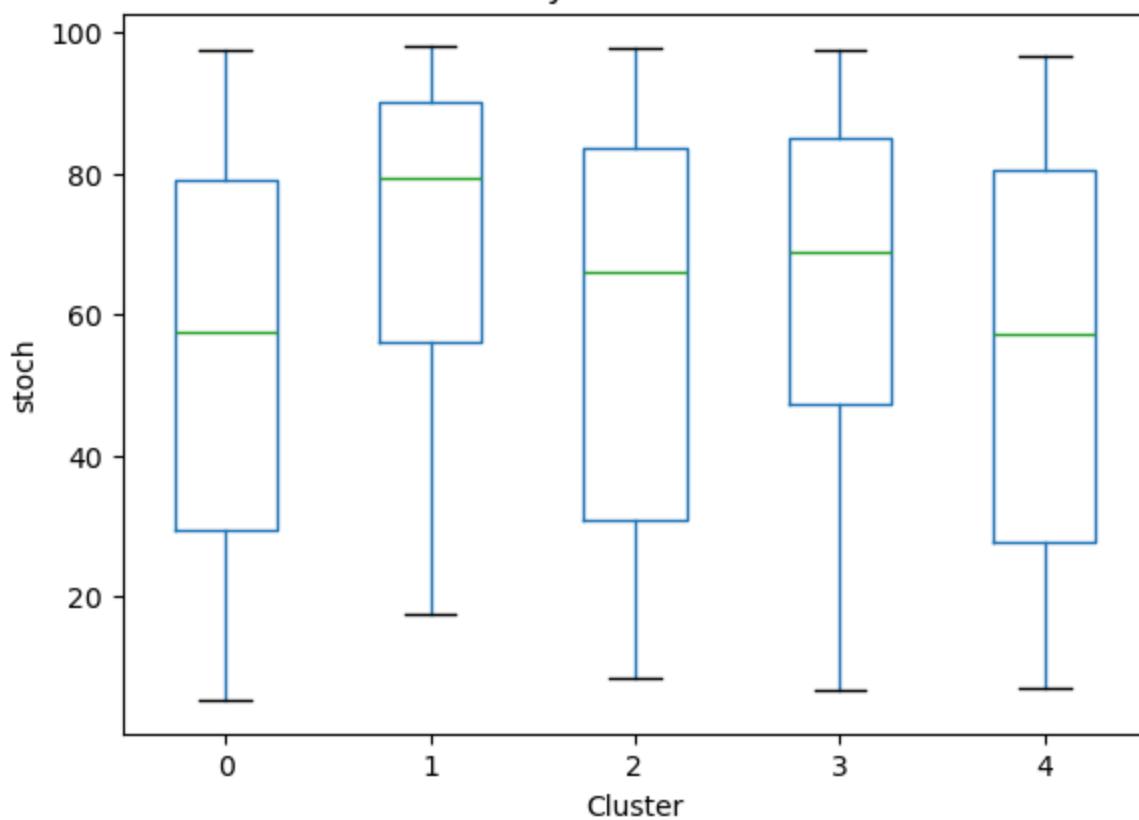
Boxplot grouped by KMeans\_Cluster  
buy\_signal by K-Means Cluster



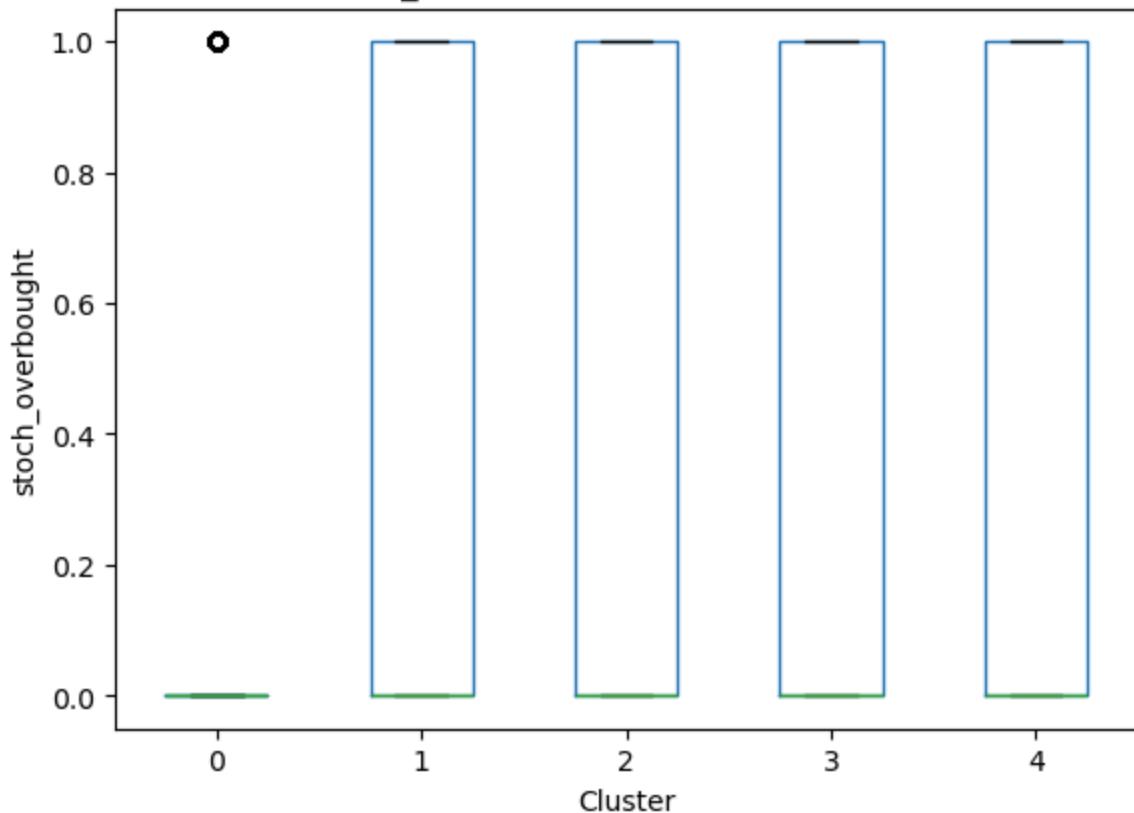
Boxplot grouped by KMeans\_Cluster  
short\_signal by K-Means Cluster



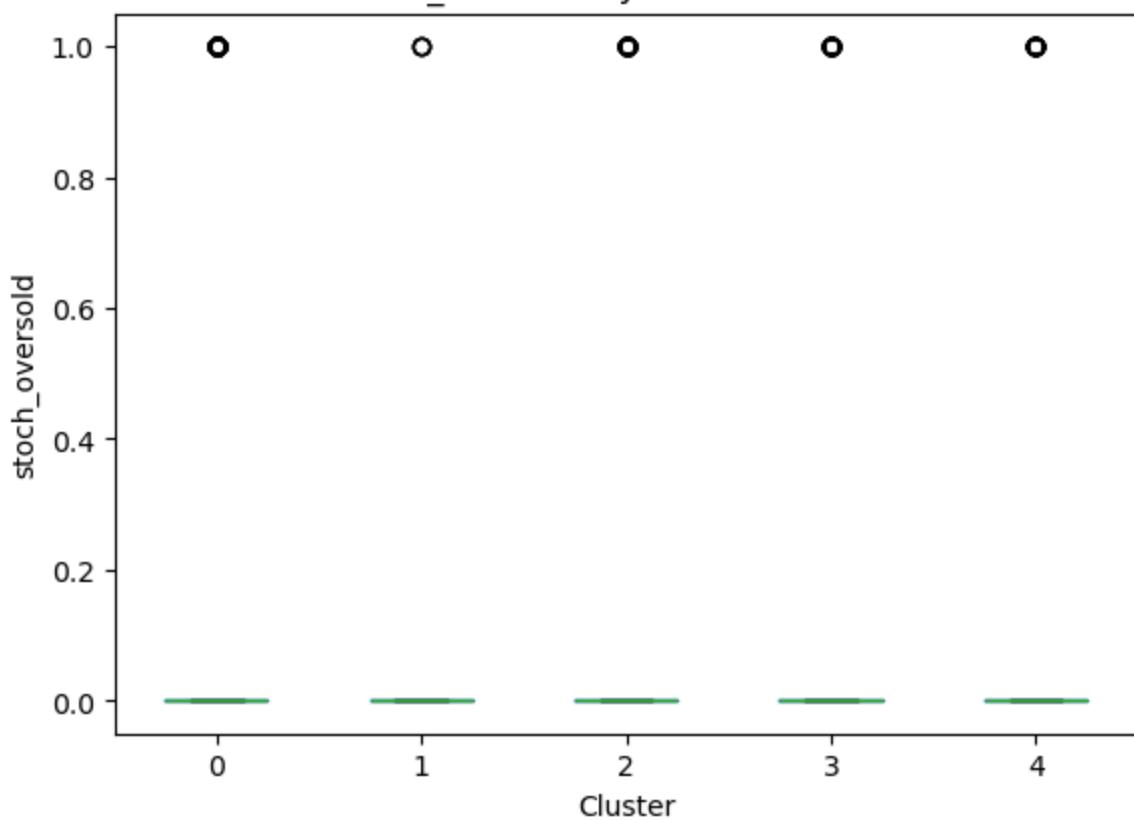
Boxplot grouped by KMeans\_Cluster  
stoch by K-Means Cluster



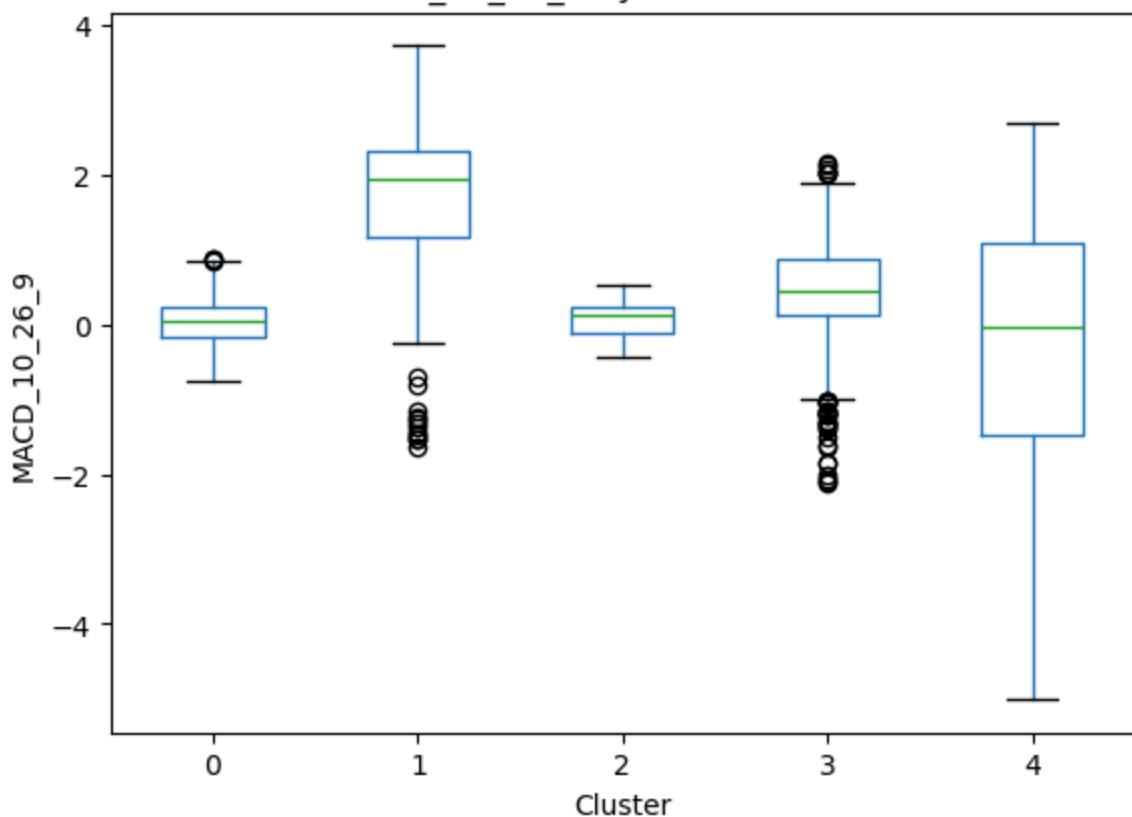
Boxplot grouped by KMeans\_Cluster  
stoch\_overbought by K-Means Cluster



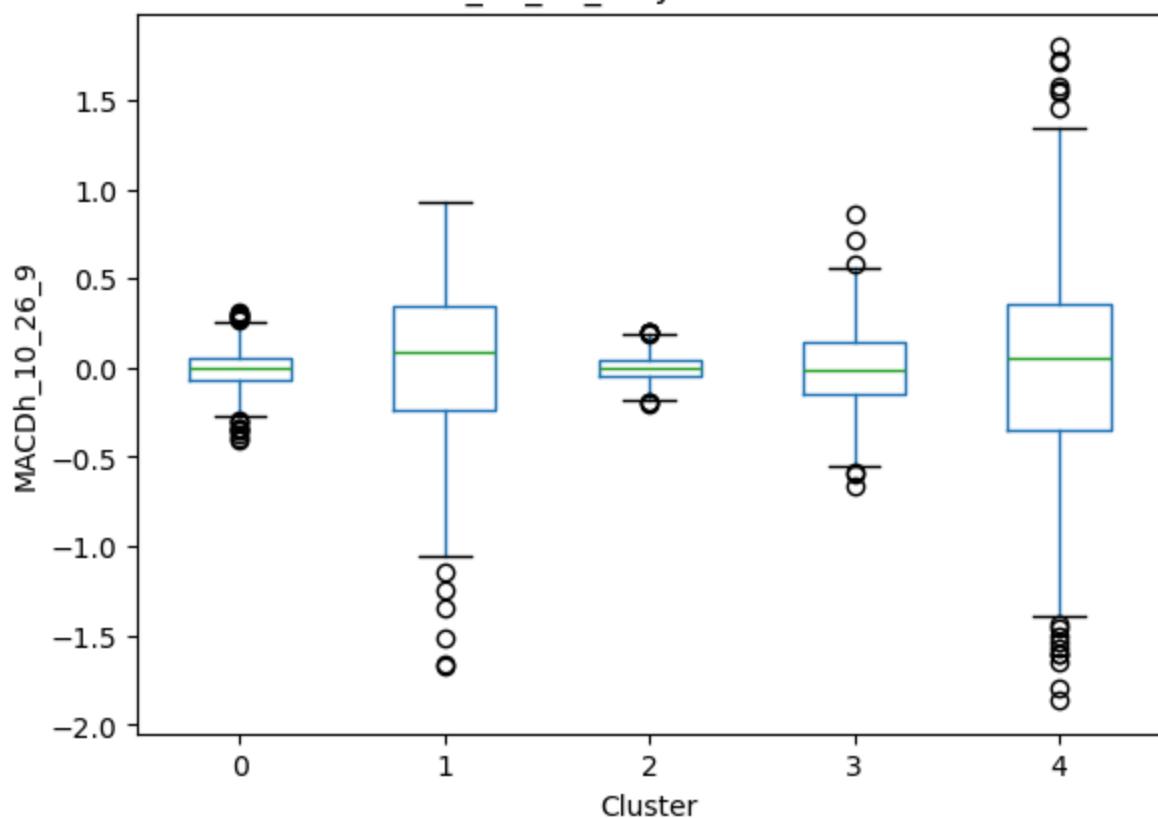
Boxplot grouped by KMeans\_Cluster  
stoch\_oversold by K-Means Cluster



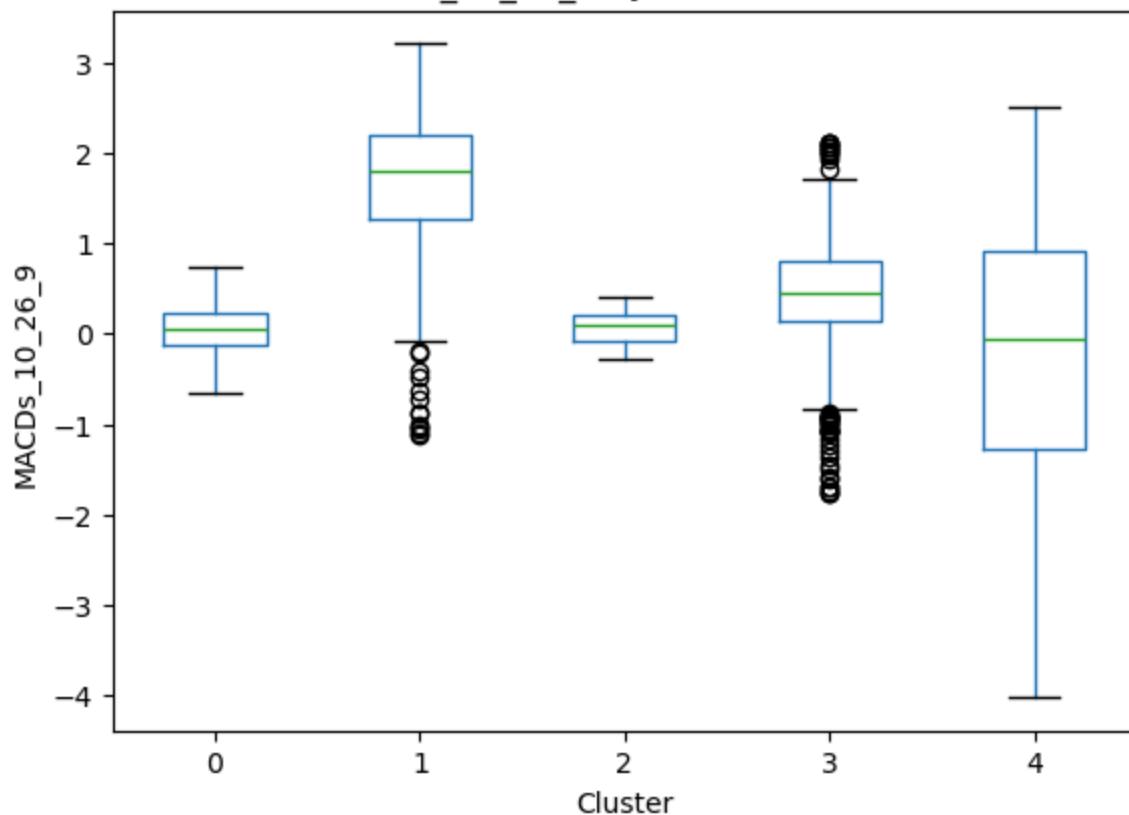
Boxplot grouped by KMeans\_Cluster  
MACD\_10\_26\_9 by K-Means Cluster



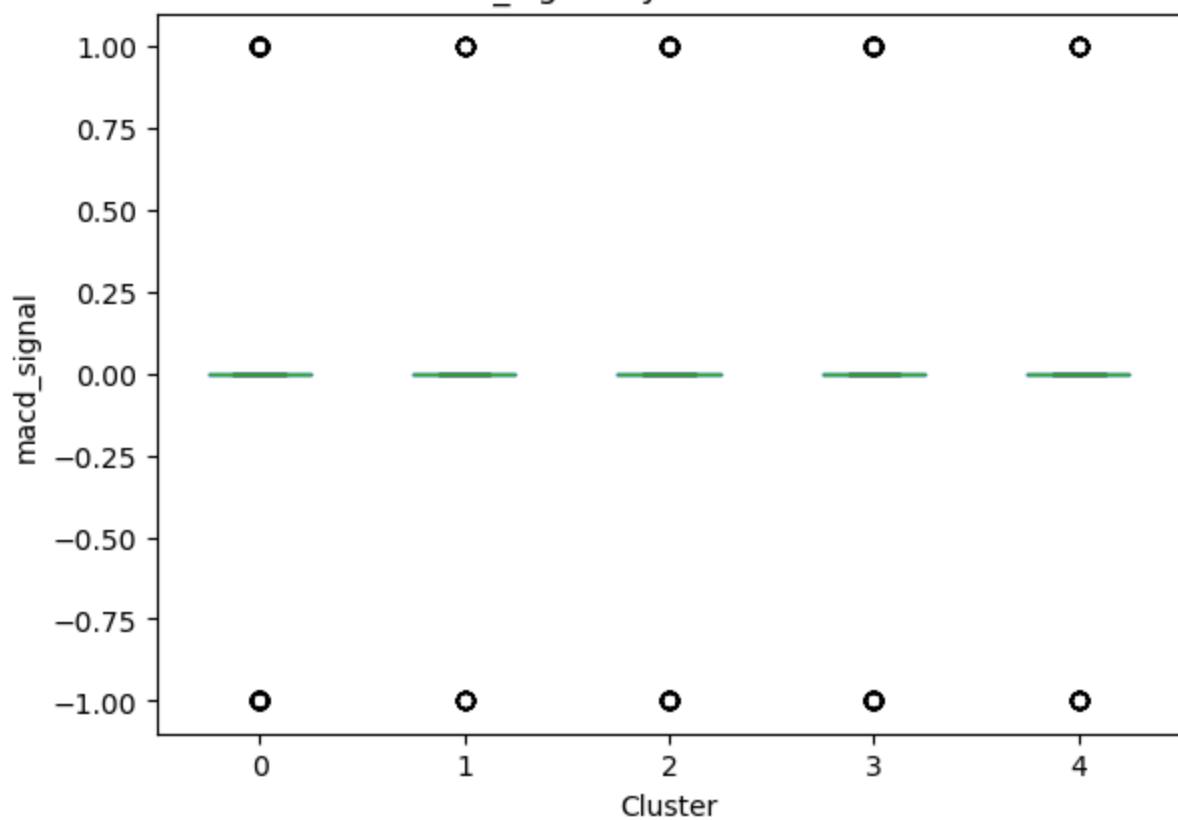
Boxplot grouped by KMeans\_Cluster  
MACDh\_10\_26\_9 by K-Means Cluster



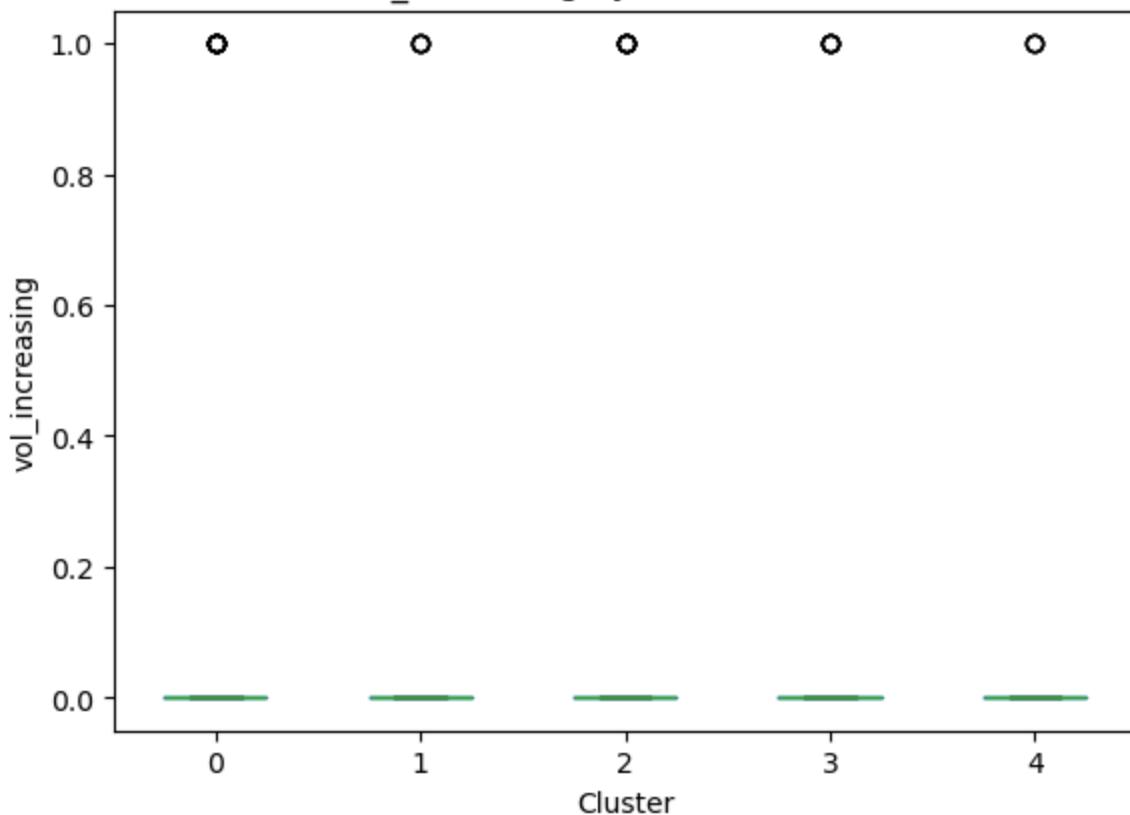
Boxplot grouped by KMeans\_Cluster  
MACDs\_10\_26\_9 by K-Means Cluster



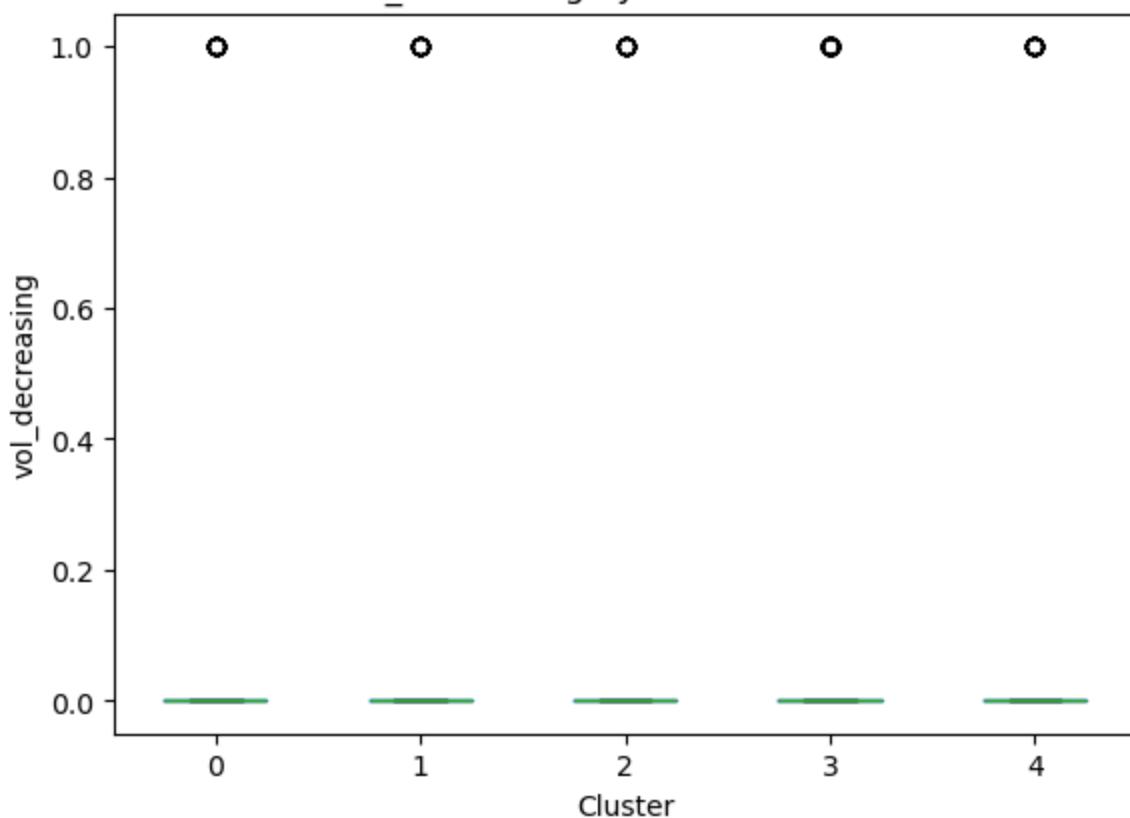
Boxplot grouped by KMeans\_Cluster  
macd\_signal by K-Means Cluster

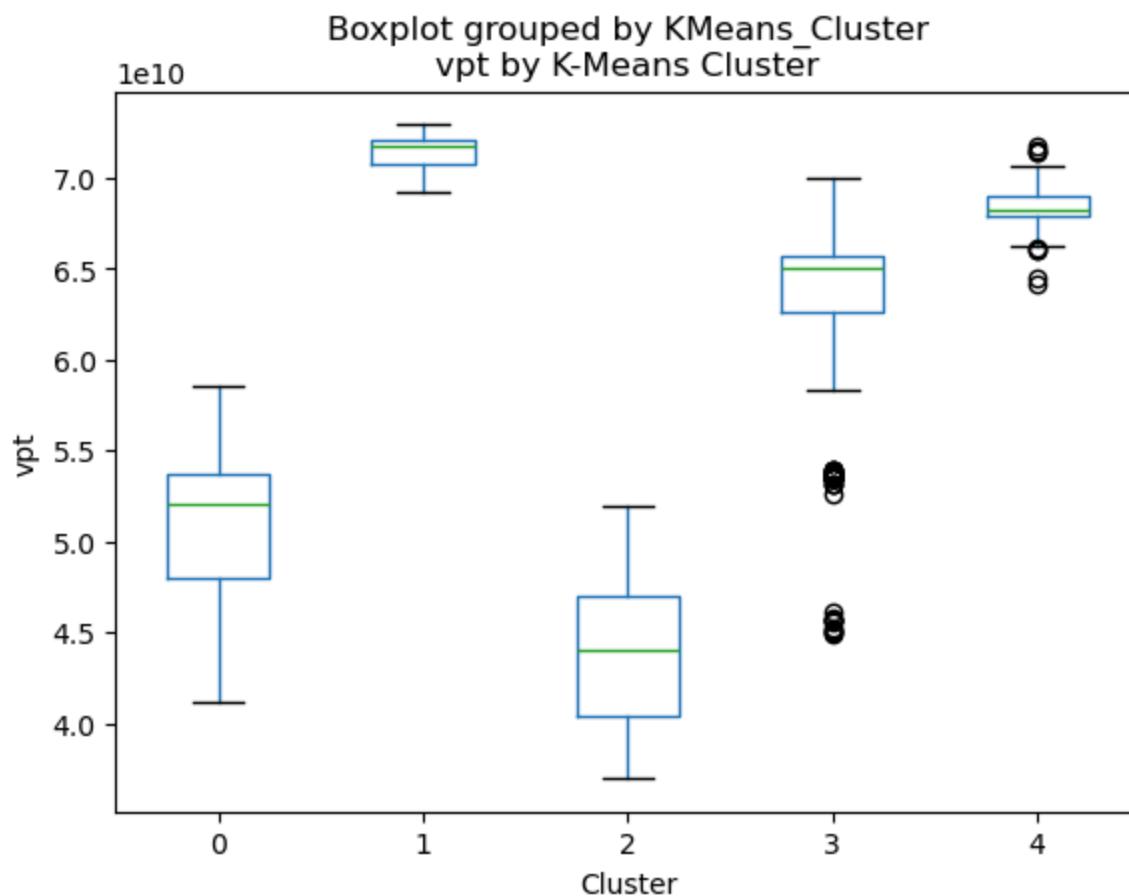
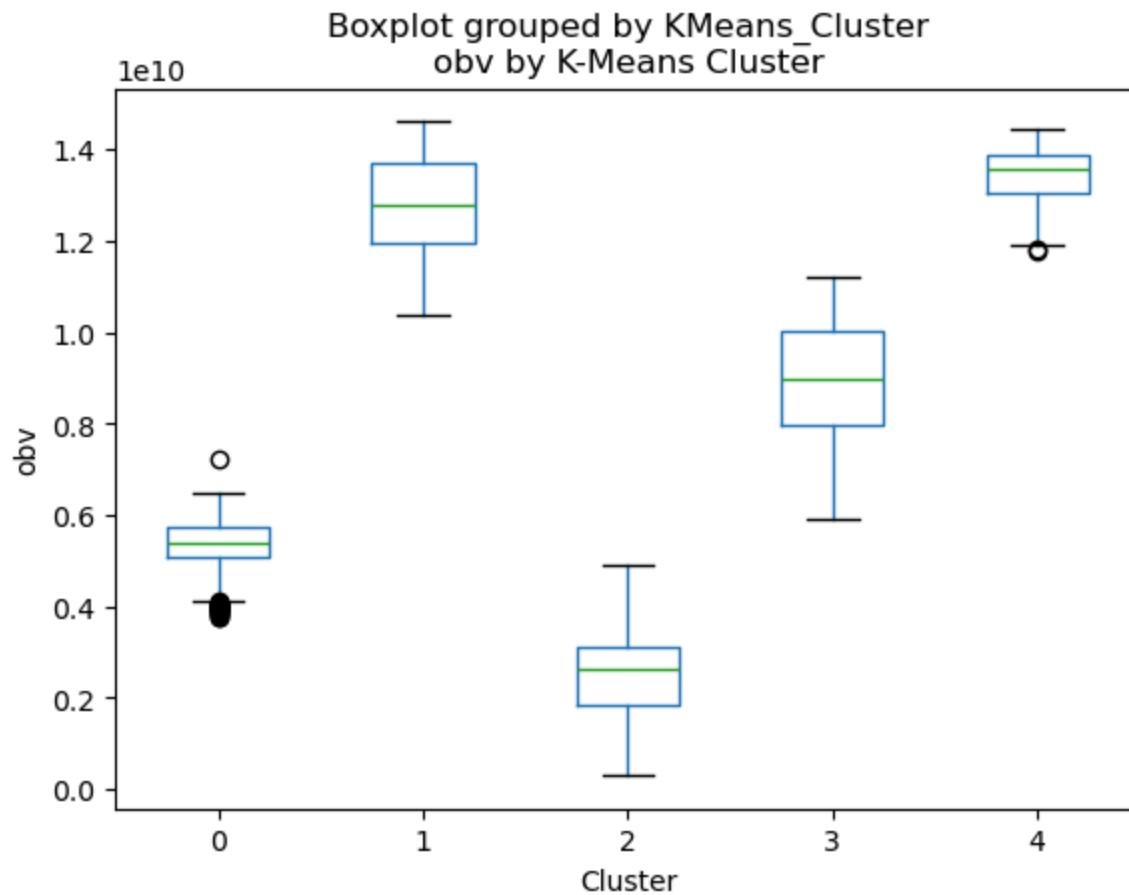


Boxplot grouped by KMeans\_Cluster  
vol\_increasing by K-Means Cluster

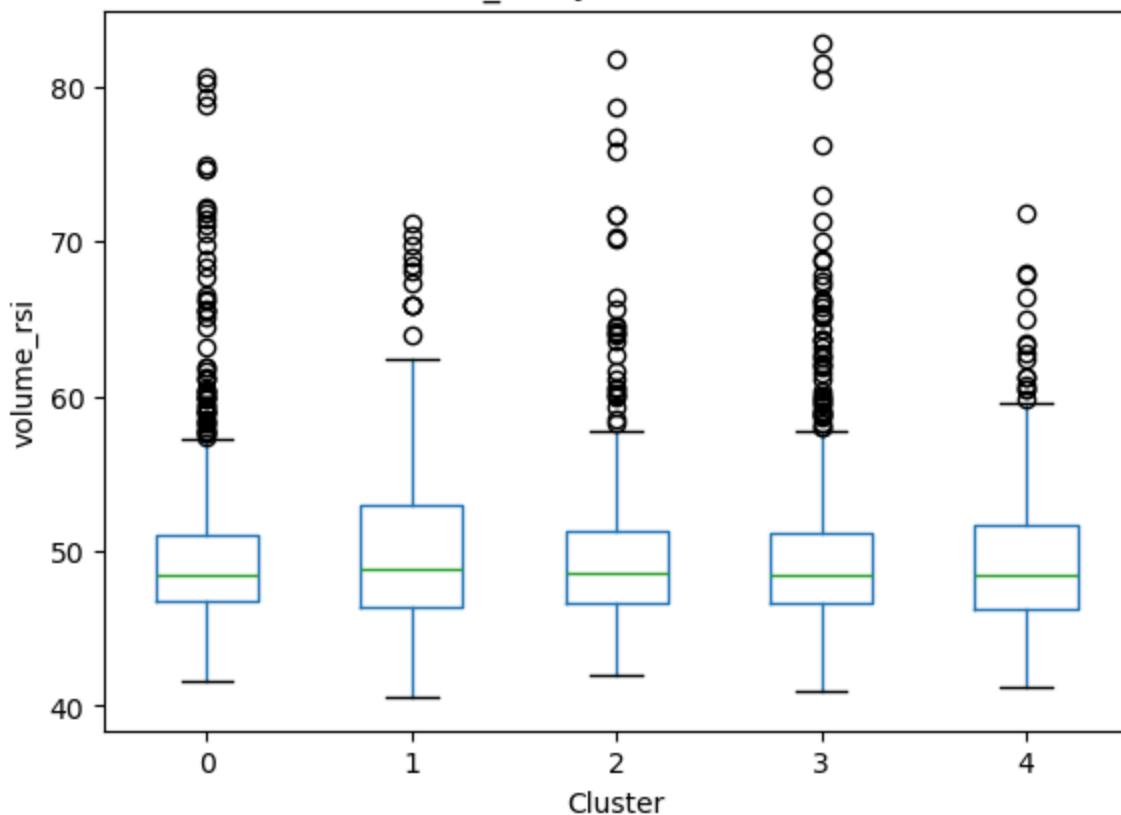


Boxplot grouped by KMeans\_Cluster  
vol\_decreasing by K-Means Cluster

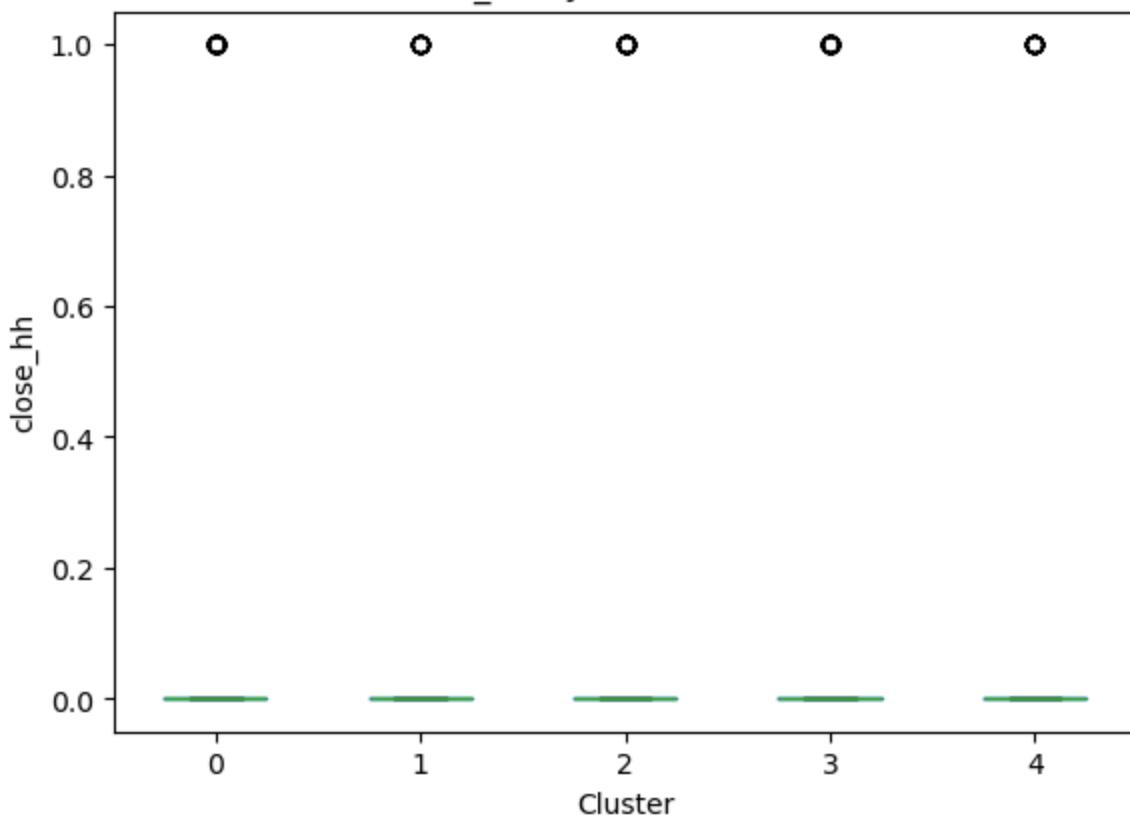




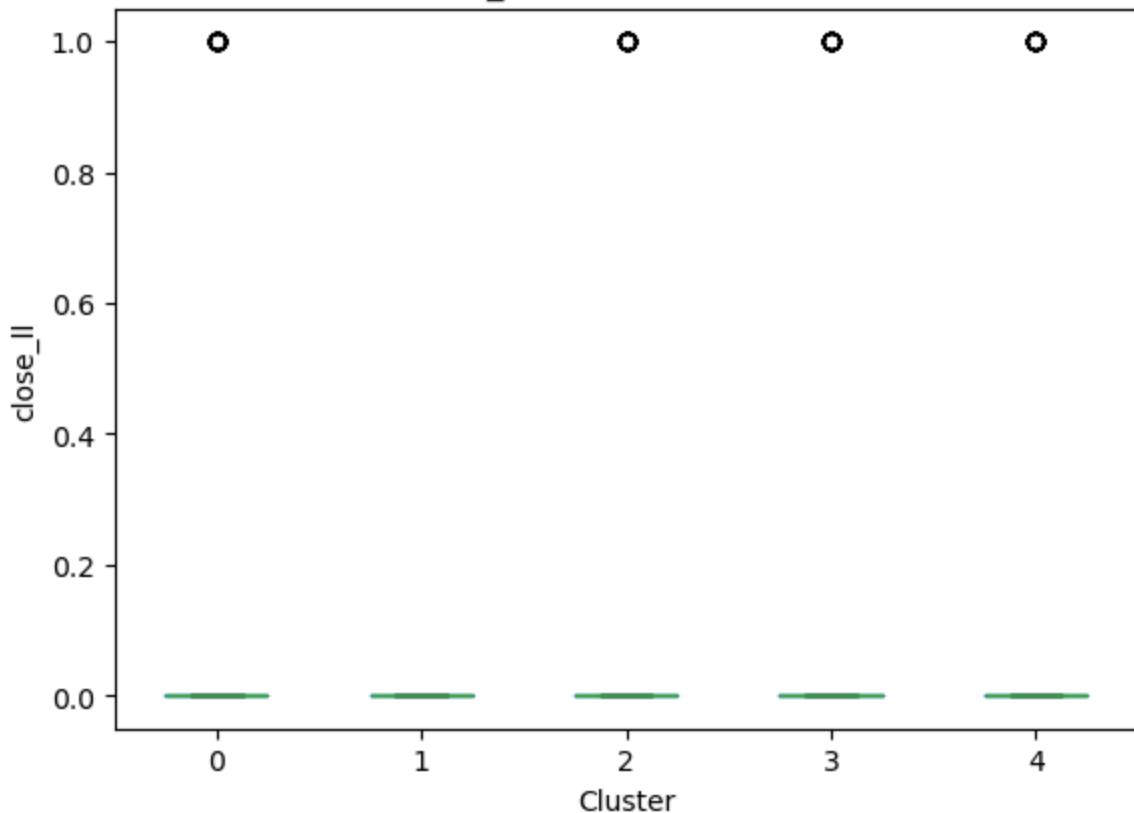
Boxplot grouped by KMeans\_Cluster  
volume\_rsi by K-Means Cluster



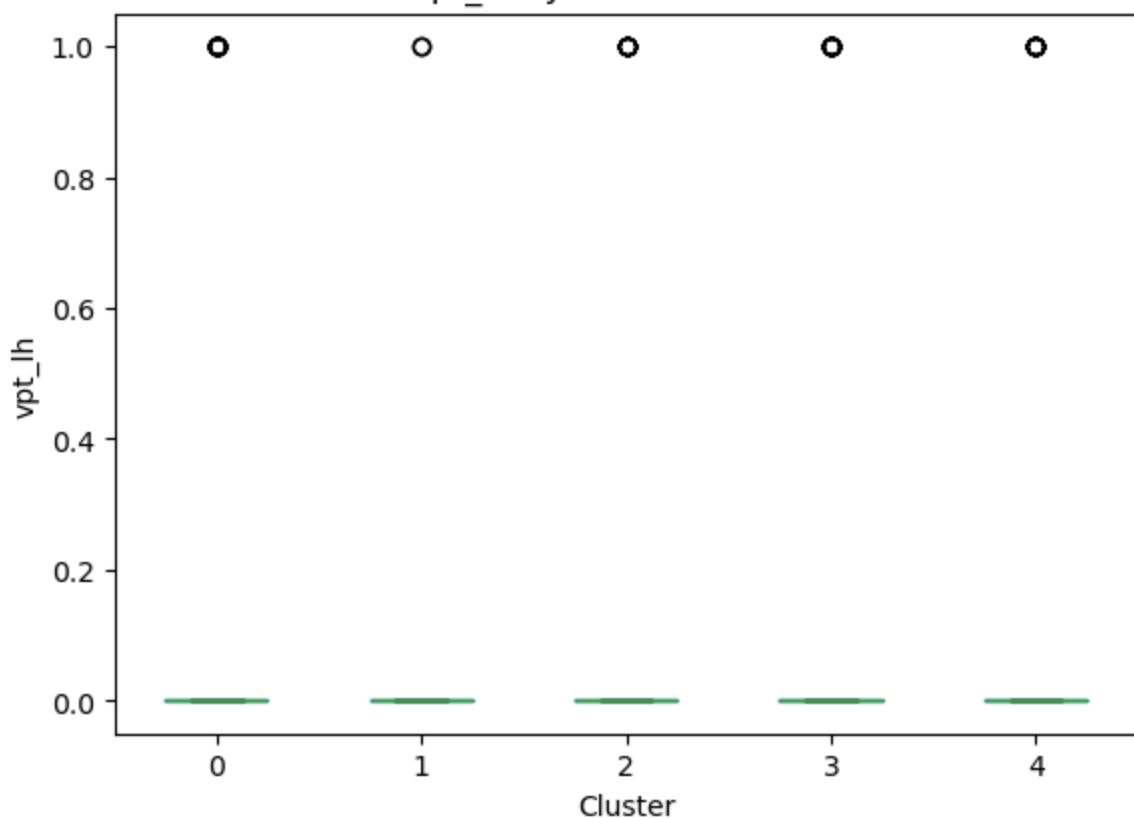
Boxplot grouped by KMeans\_Cluster  
close\_hh by K-Means Cluster



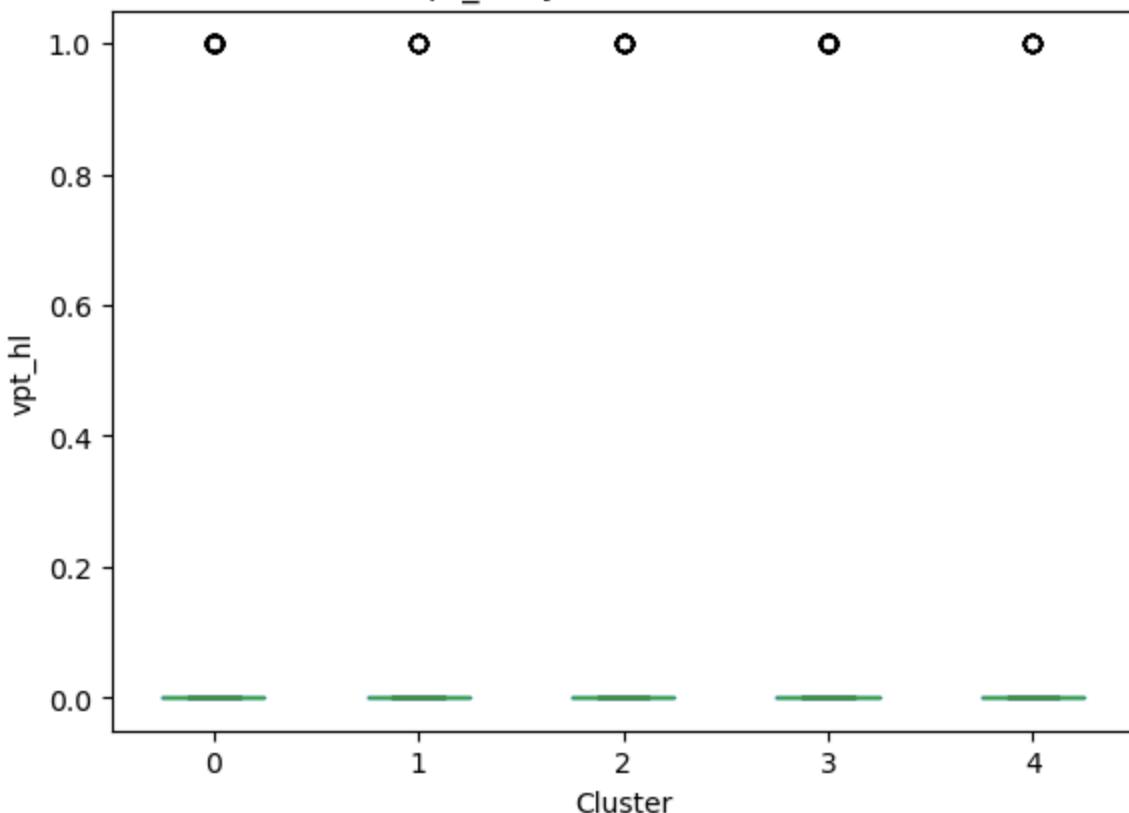
Boxplot grouped by KMeans\_Cluster  
close\_ll by K-Means Cluster



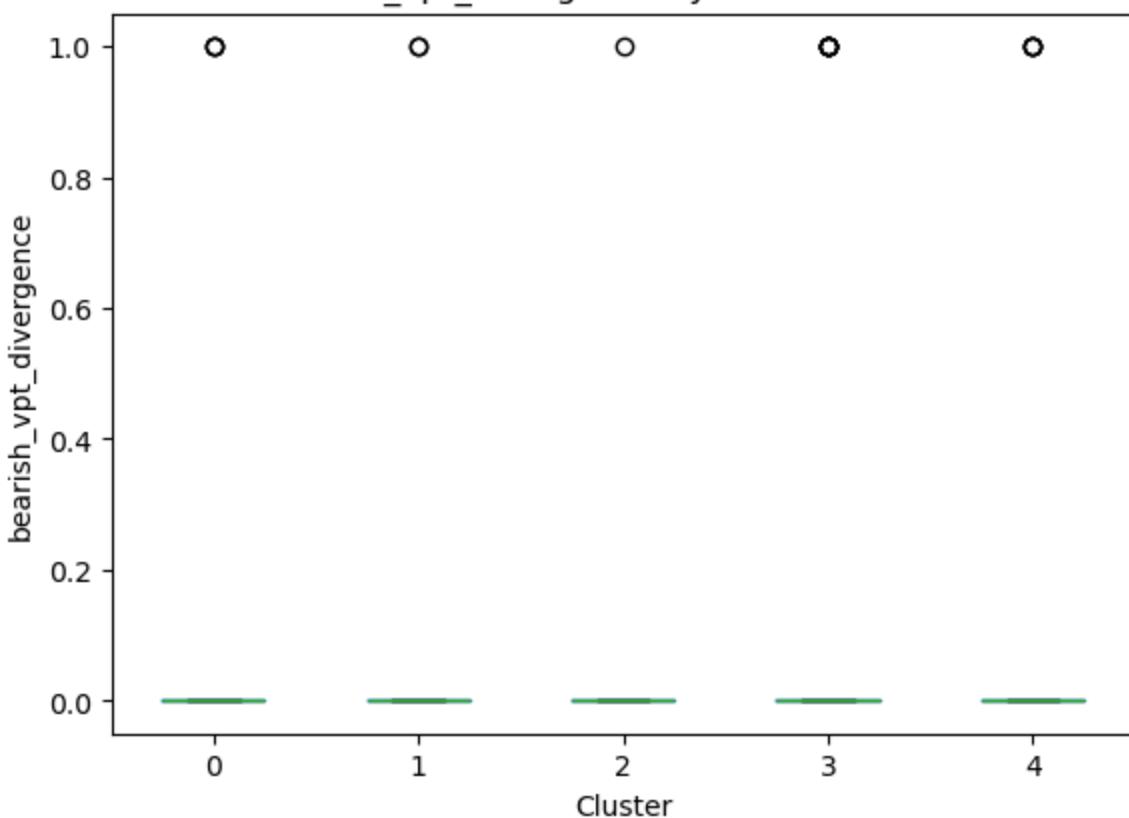
Boxplot grouped by KMeans\_Cluster  
vpt\_ll by K-Means Cluster



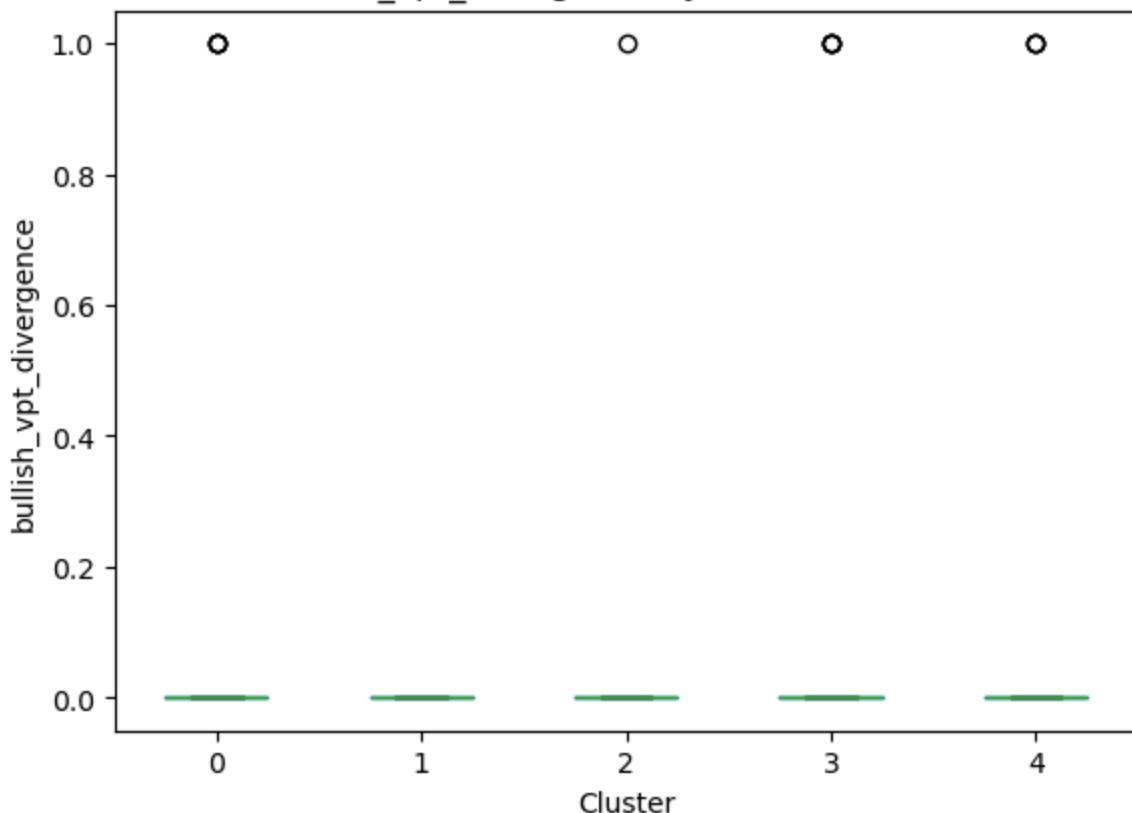
Boxplot grouped by KMeans\_Cluster  
vpt\_hl by K-Means Cluster



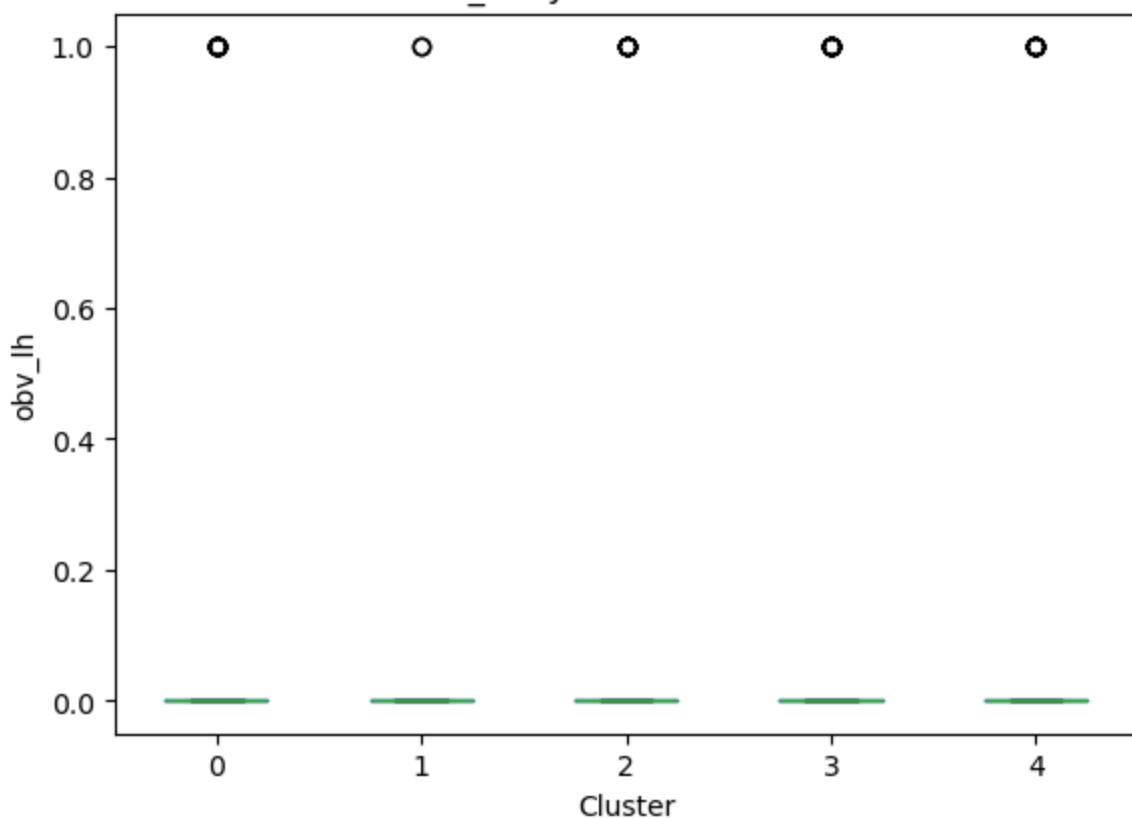
Boxplot grouped by KMeans\_Cluster  
bearish\_vpt\_divergence by K-Means Cluster



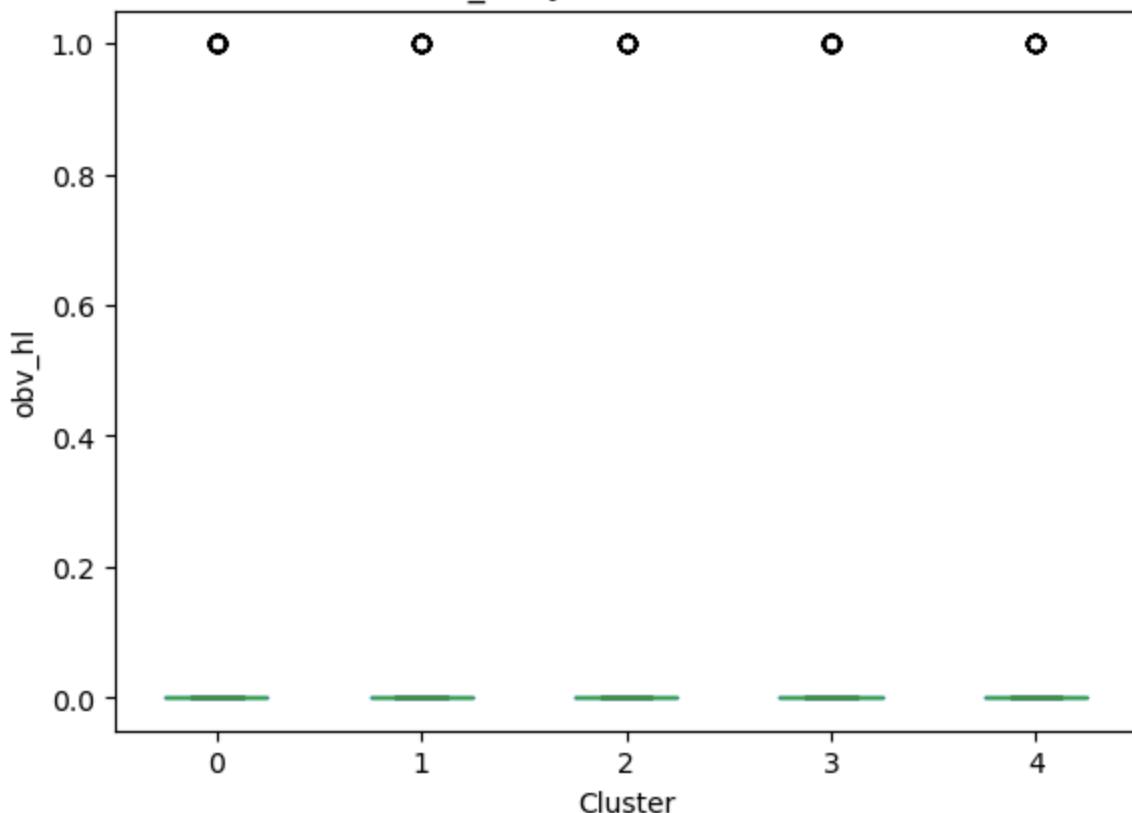
Boxplot grouped by KMeans\_Cluster  
bullish\_vpt\_divergence by K-Means Cluster



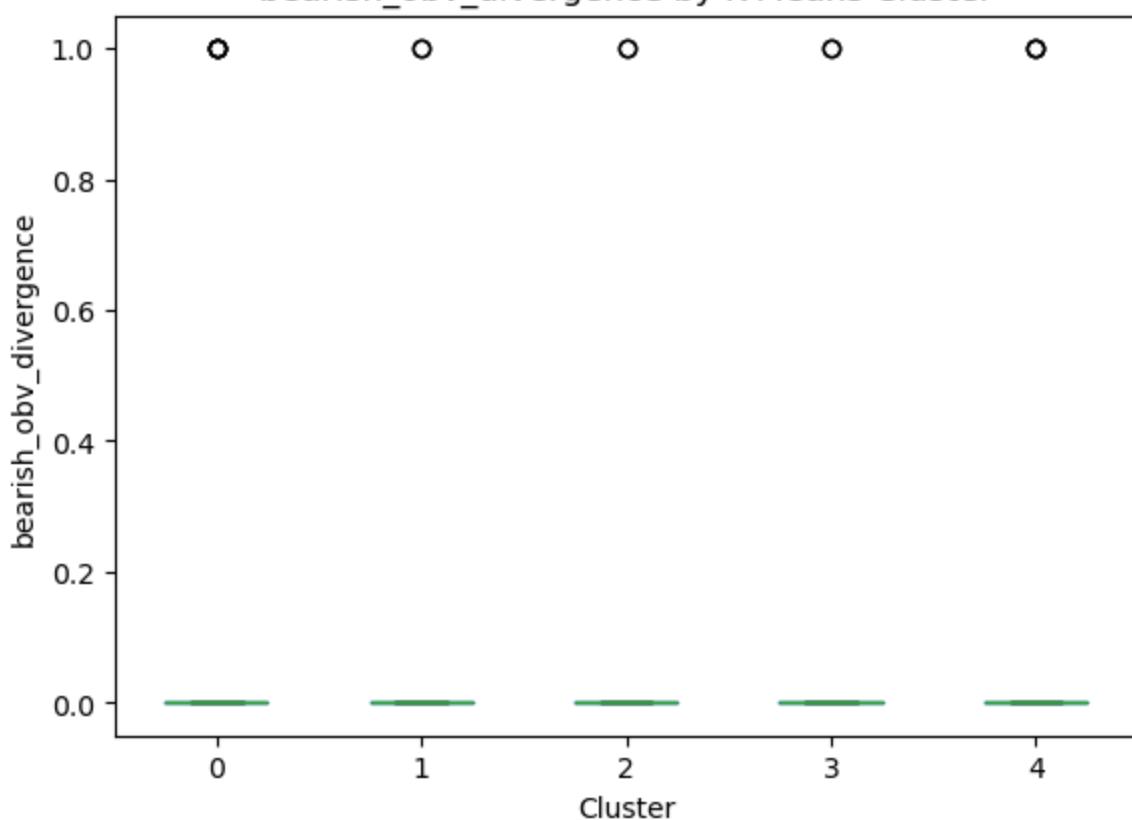
Boxplot grouped by KMeans\_Cluster  
obv\_lh by K-Means Cluster



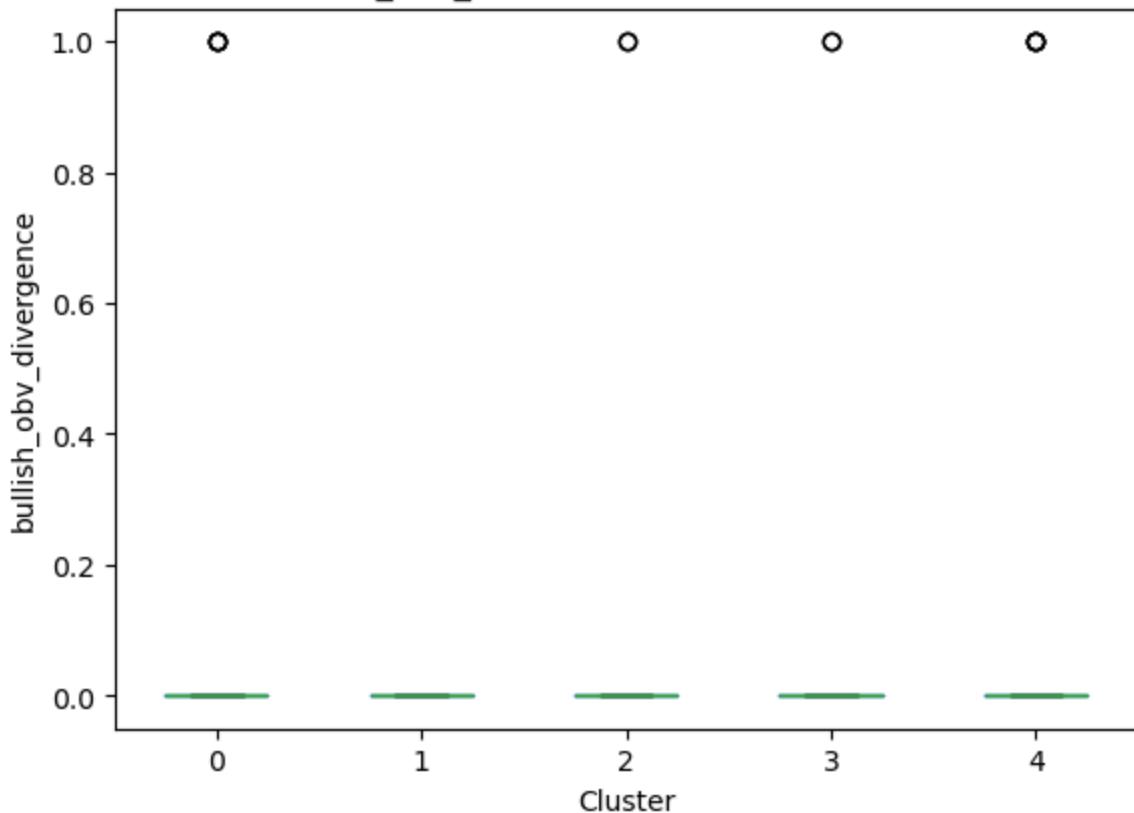
Boxplot grouped by KMeans\_Cluster  
obv\_hi by K-Means Cluster



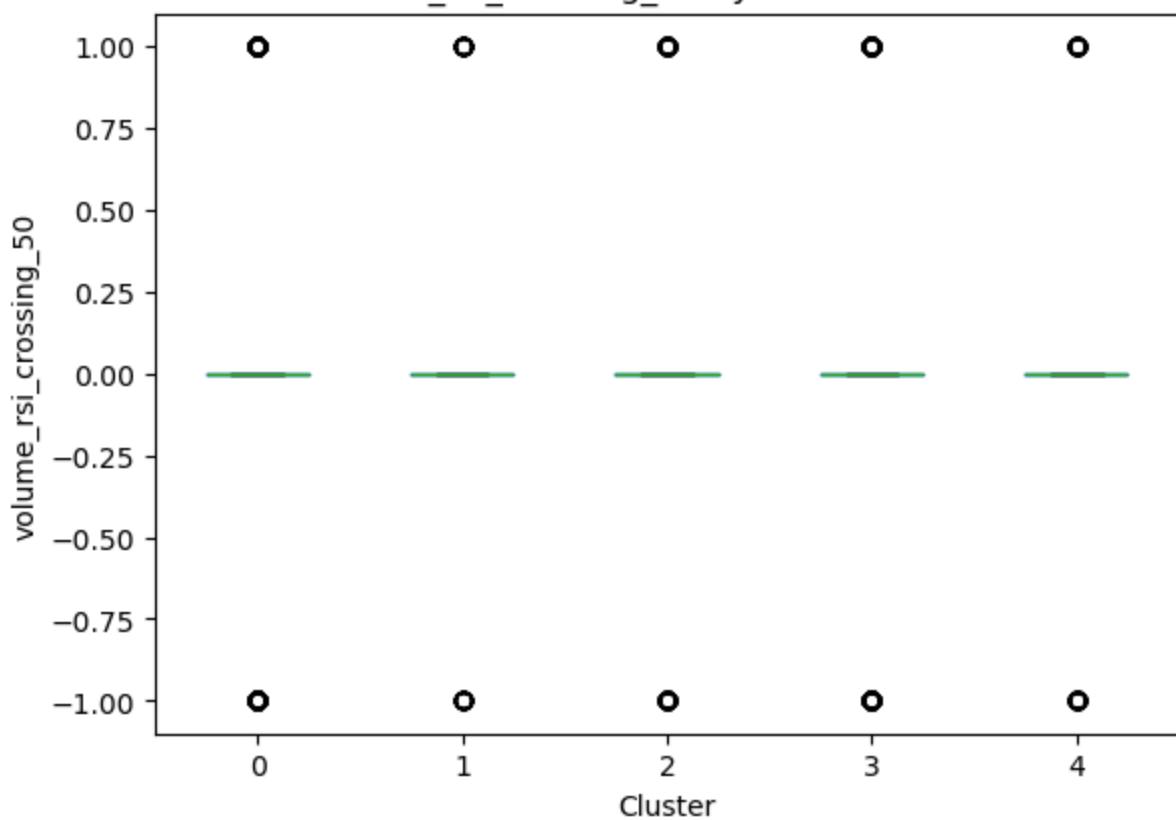
Boxplot grouped by KMeans\_Cluster  
bearish\_obv\_divergence by K-Means Cluster



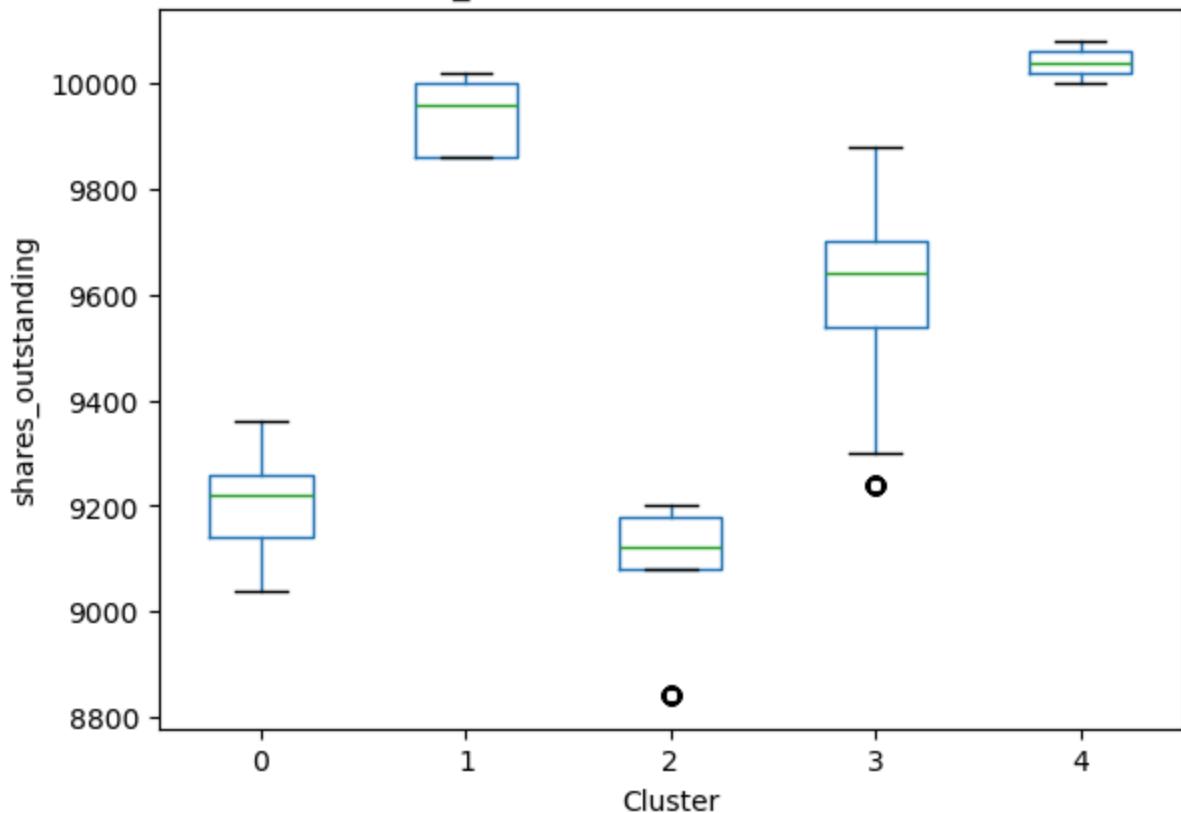
Boxplot grouped by KMeans\_Cluster  
bullish\_obv\_divergence by K-Means Cluster



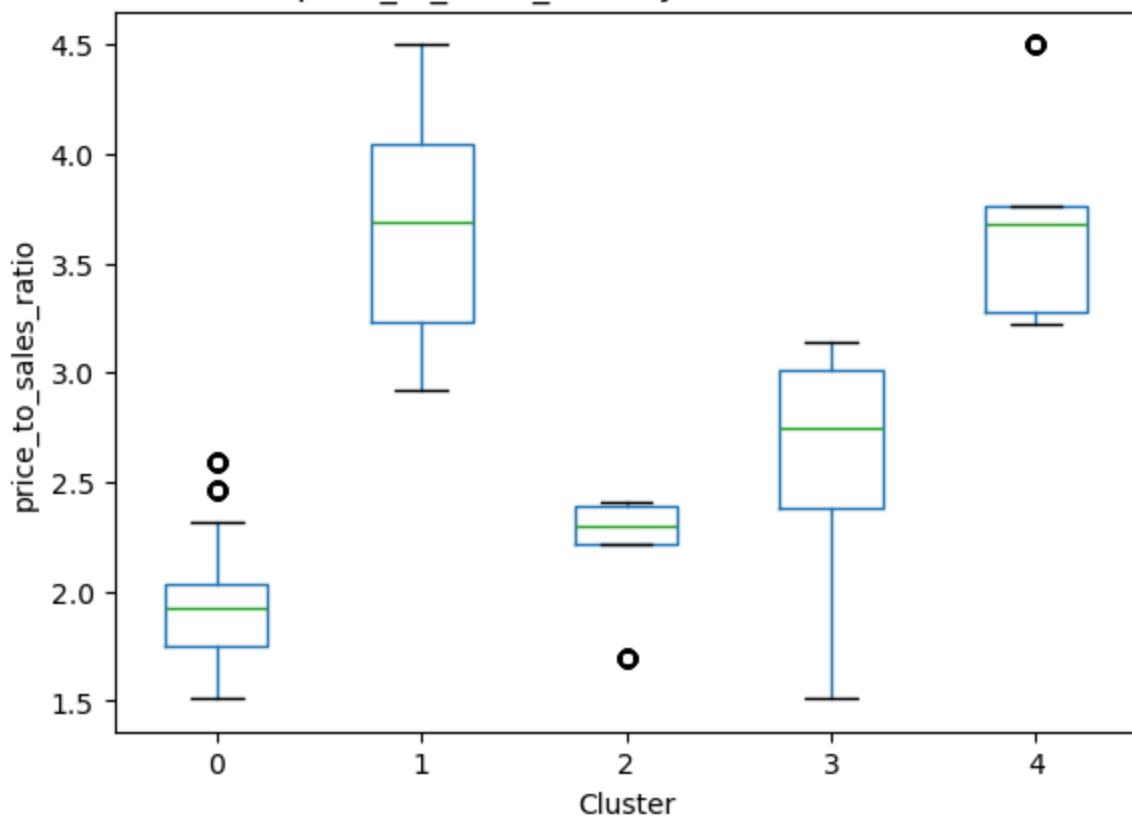
Boxplot grouped by KMeans\_Cluster  
volume\_rsi\_crossing\_50 by K-Means Cluster



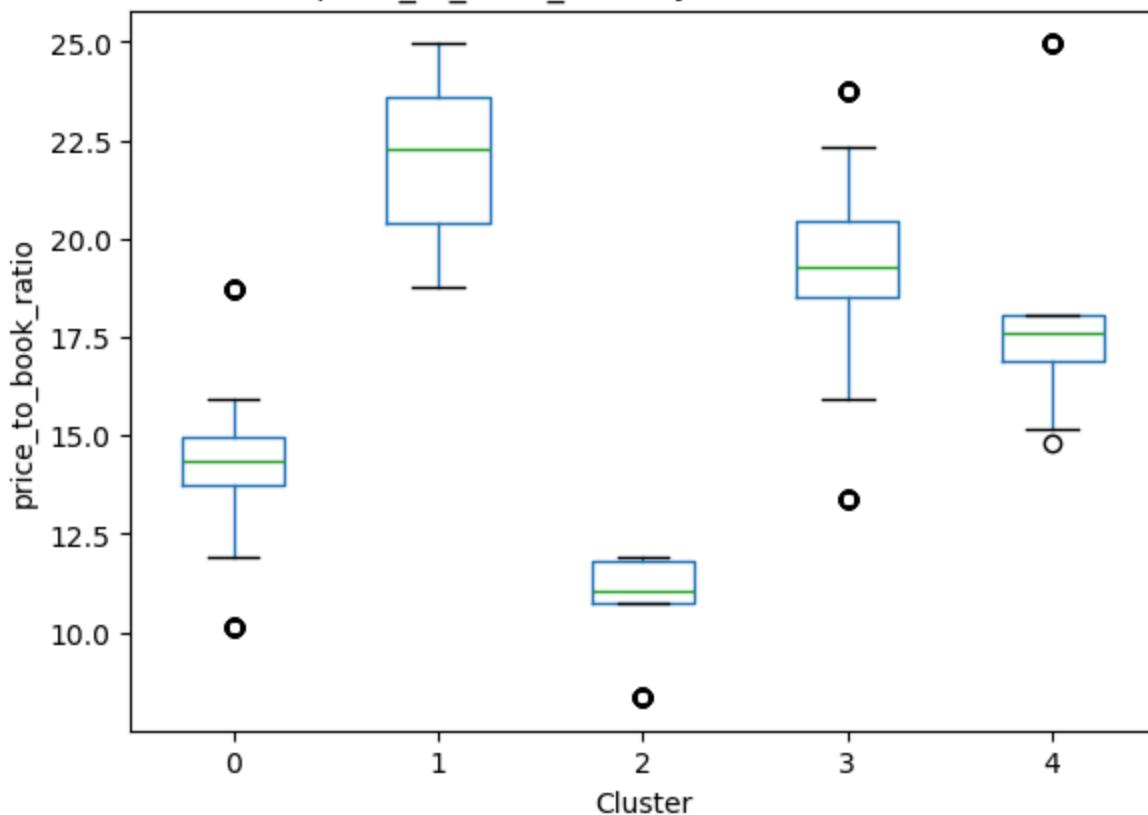
Boxplot grouped by KMeans\_Cluster  
shares\_outstanding by K-Means Cluster



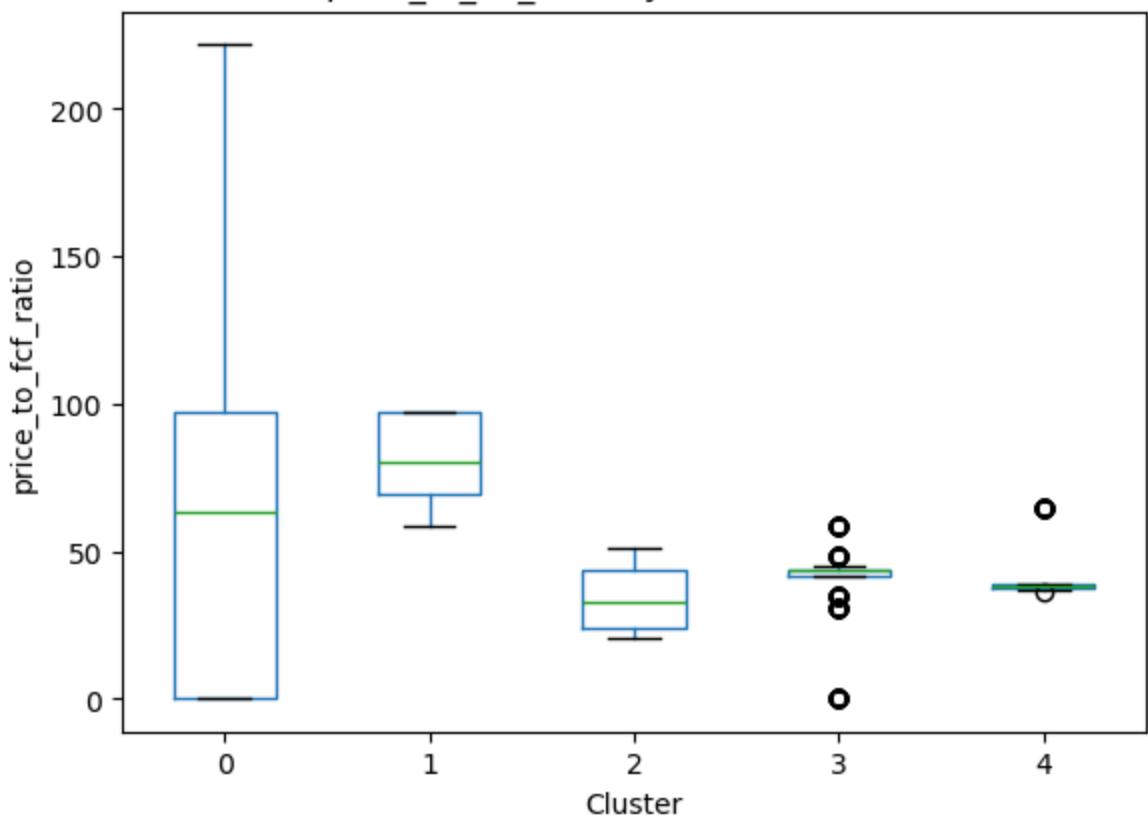
Boxplot grouped by KMeans\_Cluster  
price\_to\_sales\_ratio by K-Means Cluster



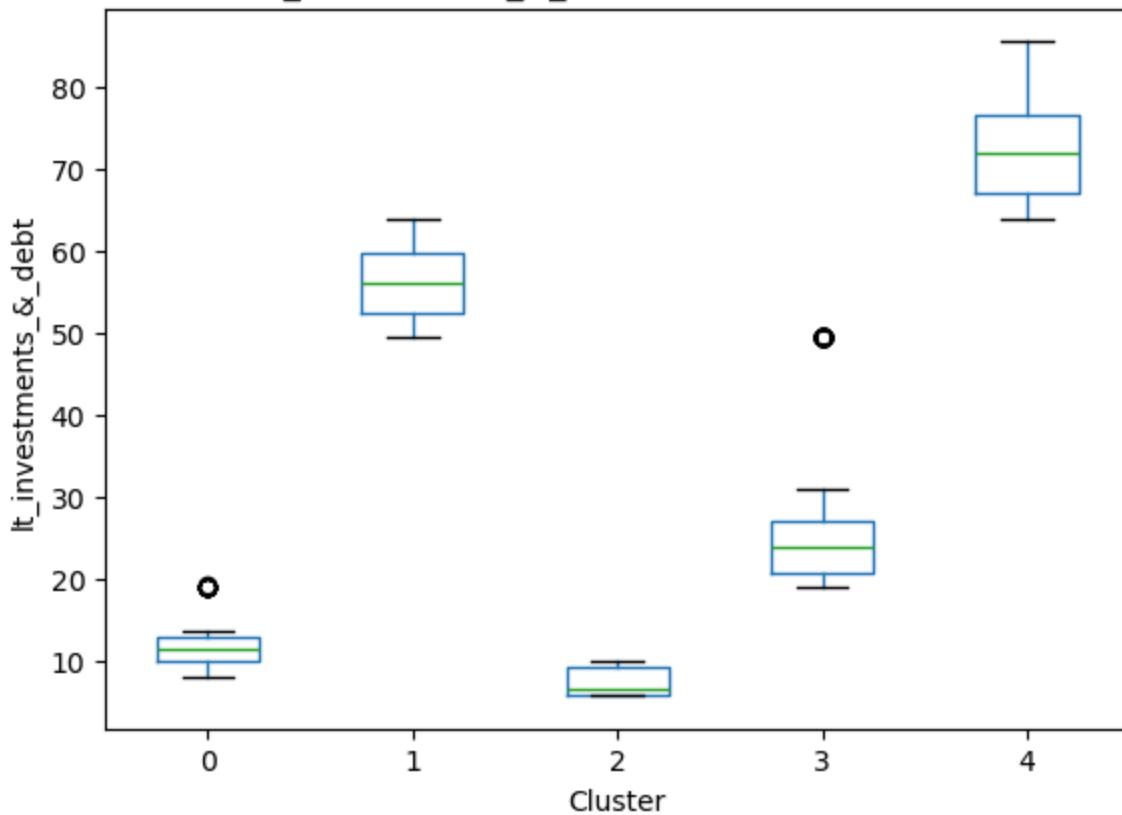
Boxplot grouped by KMeans\_Cluster  
price\_to\_book\_ratio by K-Means Cluster



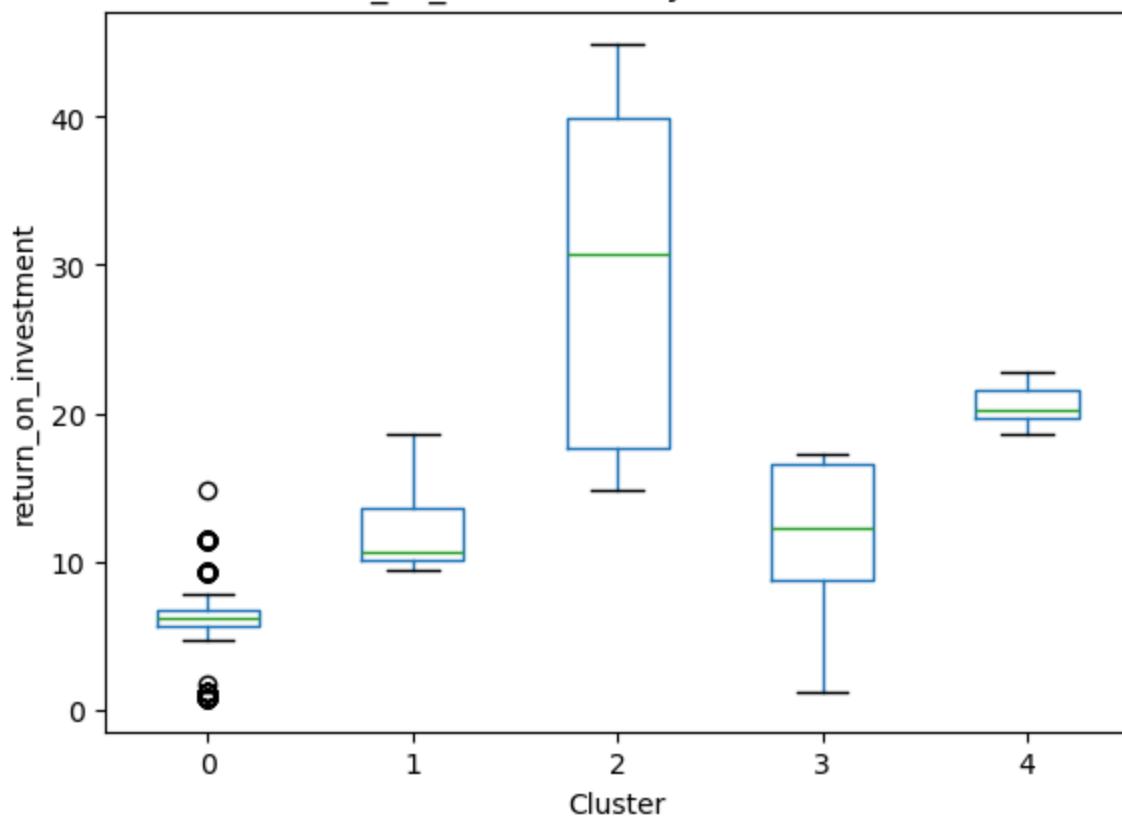
Boxplot grouped by KMeans\_Cluster  
price\_to\_fcf\_ratio by K-Means Cluster



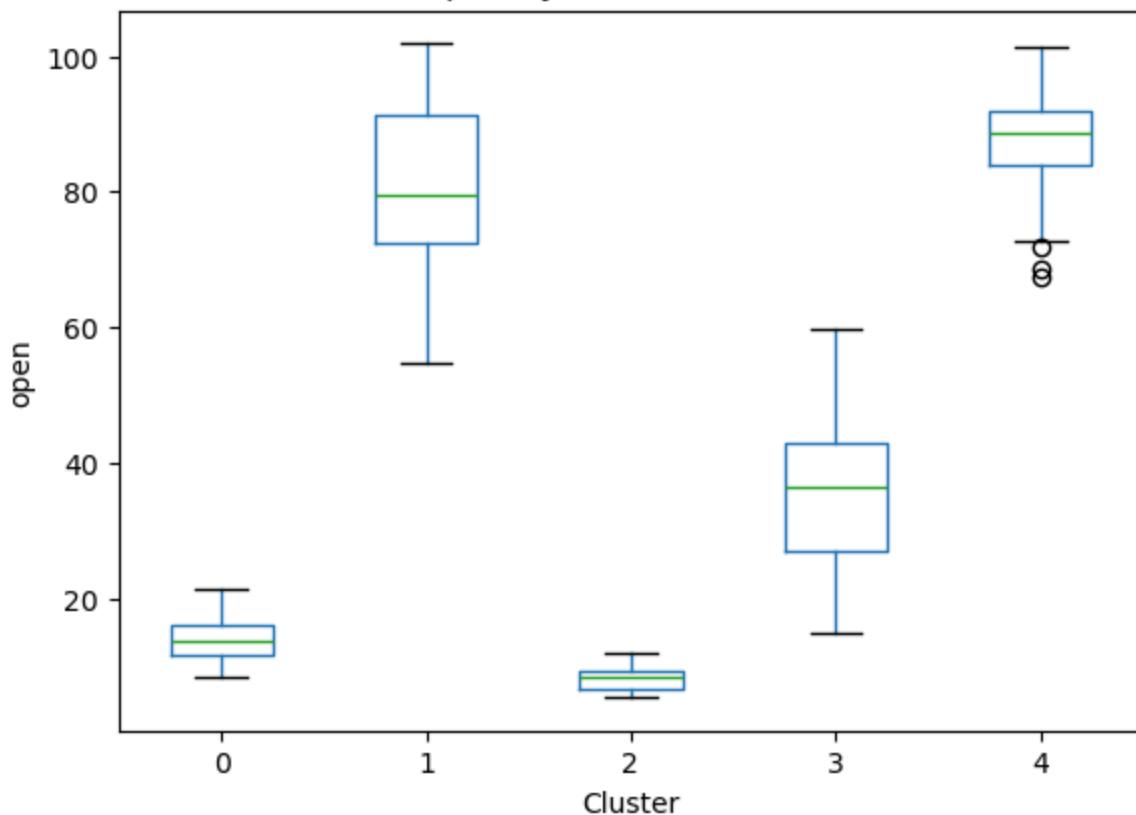
Boxplot grouped by KMeans\_Cluster  
lt\_investments\_&\_debt by K-Means Cluster



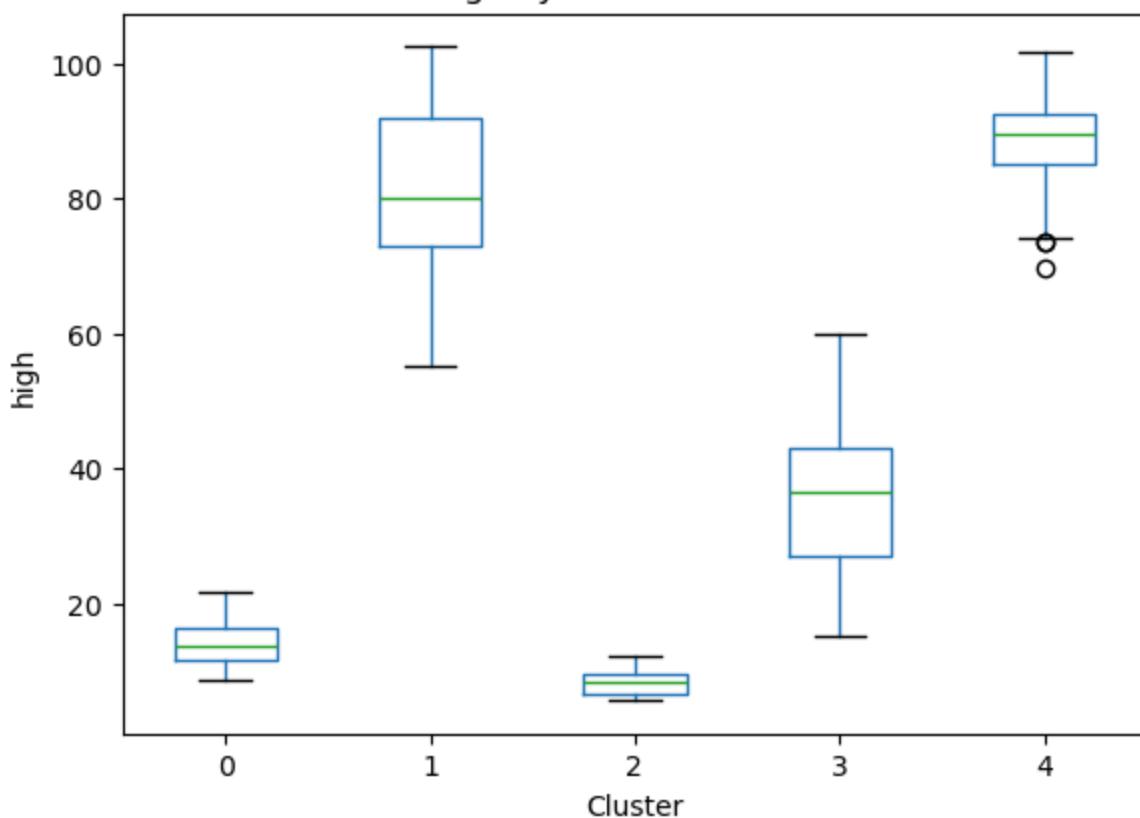
Boxplot grouped by KMeans\_Cluster  
return\_on\_investment by K-Means Cluster



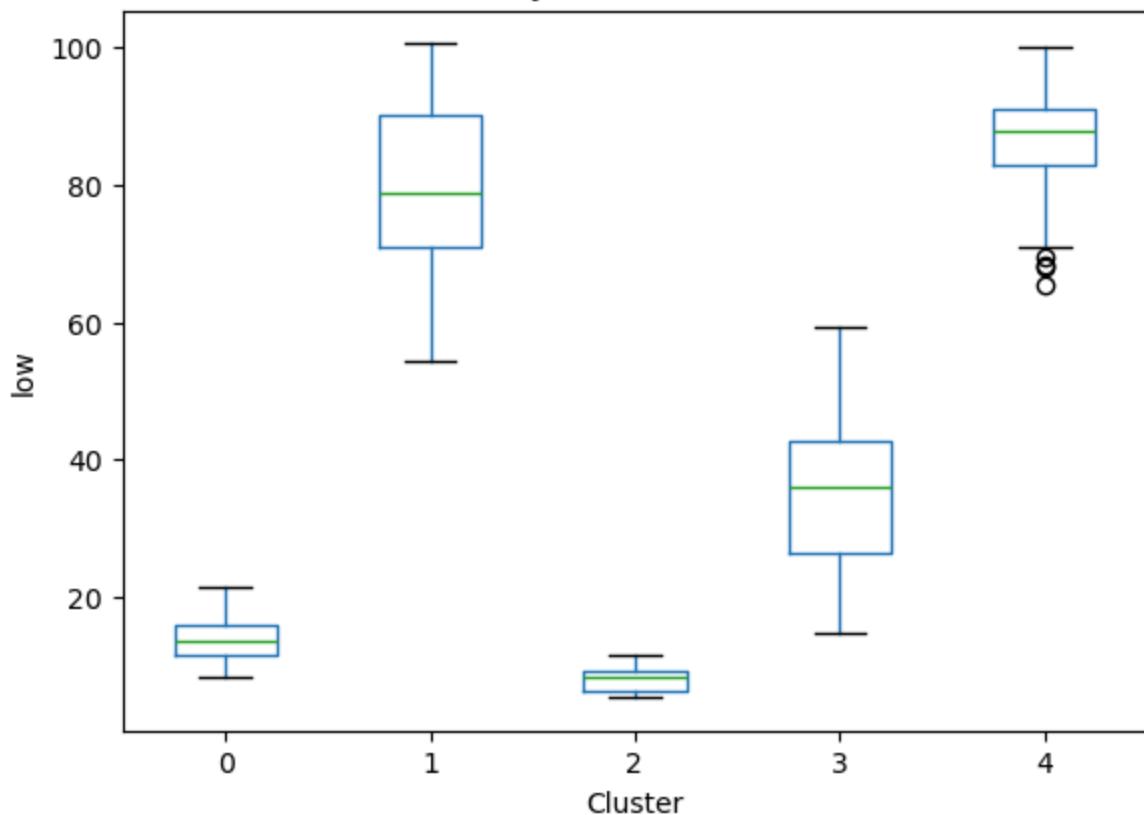
Boxplot grouped by KMeans\_Cluster  
open by K-Means Cluster



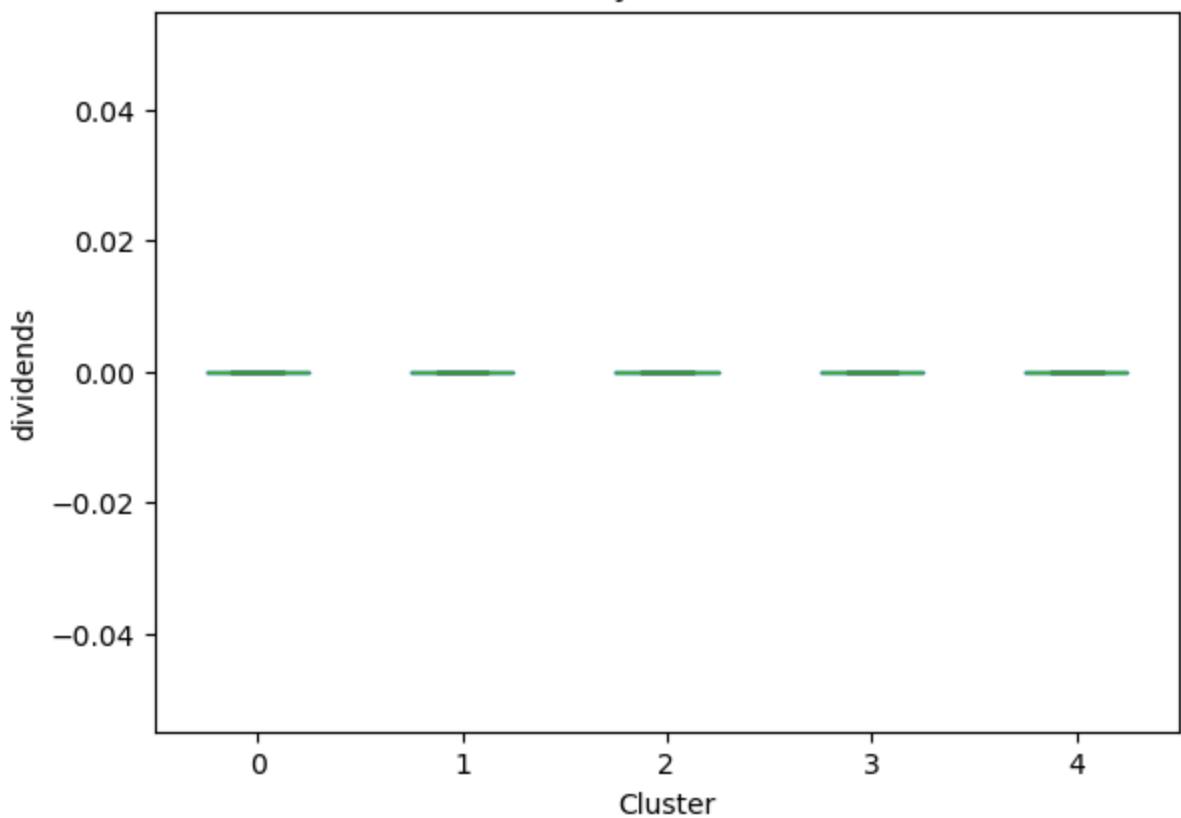
Boxplot grouped by KMeans\_Cluster  
high by K-Means Cluster



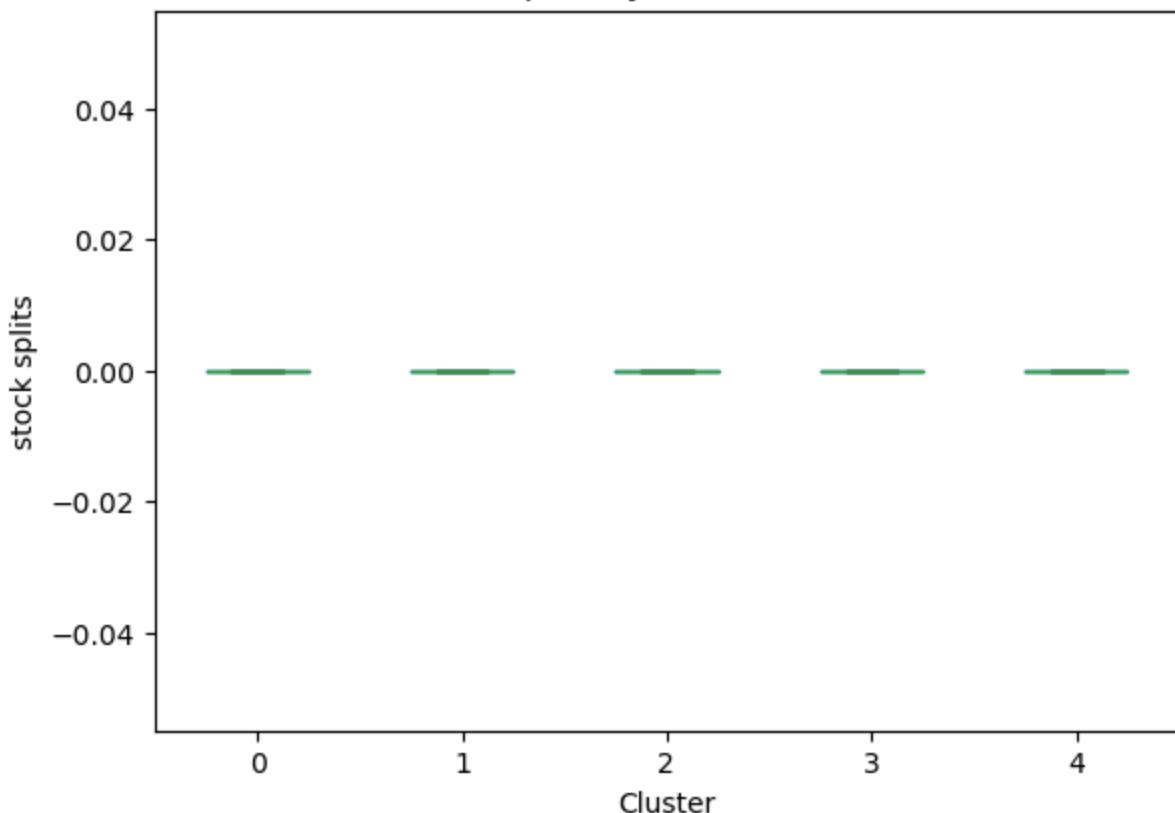
Boxplot grouped by KMeans\_Cluster  
low by K-Means Cluster



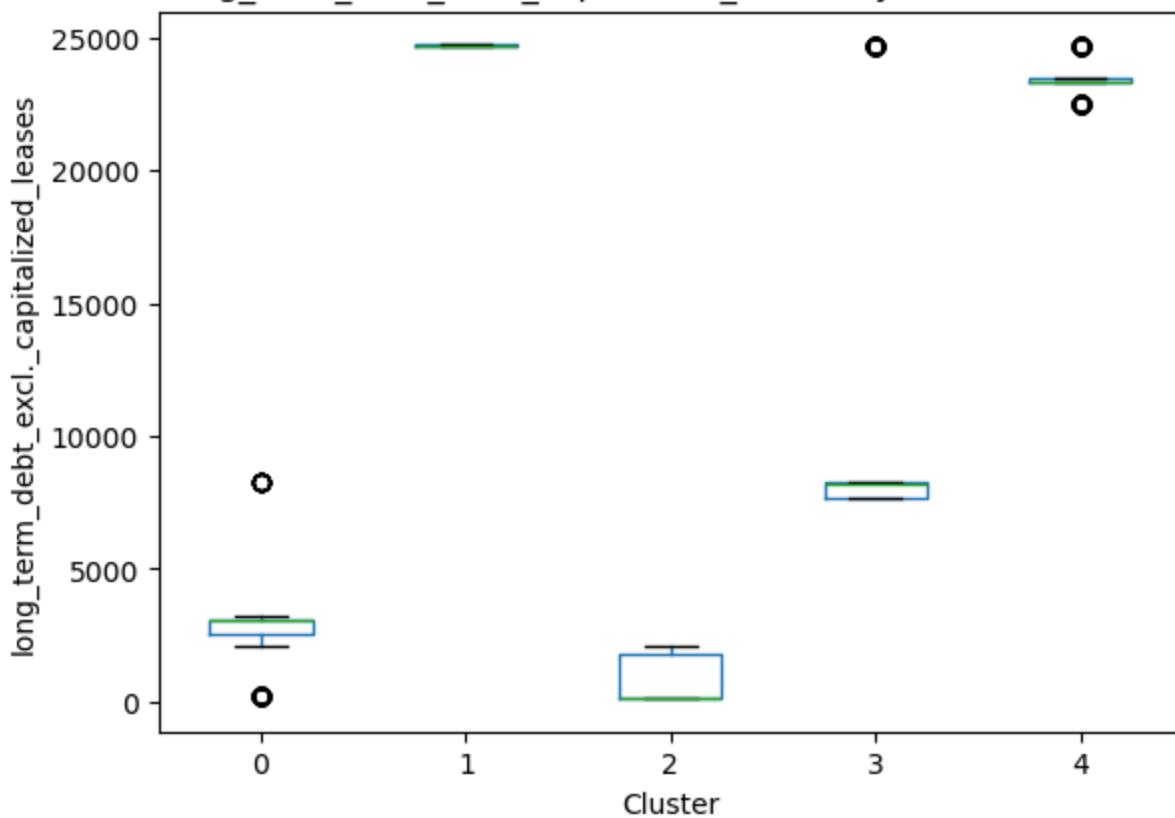
Boxplot grouped by KMeans\_Cluster  
dividends by K-Means Cluster



Boxplot grouped by KMeans\_Cluster  
stock splits by K-Means Cluster



Boxplot grouped by KMeans\_Cluster  
long\_term\_debt\_excl\_capitalized\_leases by K-Means Cluster



## Hierarchical Clustering

In [77]:

```
from scipy.cluster.hierarchy import linkage, fcluster
import pandas as pd

# Perform hierarchical clustering
linked = linkage(X_scaled, method='ward') # Perform clustering

# Extract flat clusters (e.g., 5 clusters)
hierarchical_clusters = fcluster(linked, t=5, criterion='maxclust')

# Add cluster labels to the dataset
train_data2 = train_data.copy()
train_data2.loc[:, 'Hierarchical_Cluster'] = hierarchical_clusters

# Summarize predictors and response ('close') by cluster
hierarchical_summary = train_data2.groupby('Hierarchical_Cluster').mean()
print("Hierarchical Cluster Summary:")
print(hierarchical_summary)
```

## Hierarchical Cluster Summary:

Hierarchical_Cluster	Date	dxy	oil	\	
1	2018-06-08 23:00:21.301775104	93.072900	67.189290		
2	2019-05-09 16:48:29.090909184	97.045993	57.409832		
3	2016-08-09 14:16:34.475138304	95.296446	51.128011		
4	2010-11-04 06:57:04.390243840	79.219146	86.949878		
5	2013-11-08 13:35:18.189581568	84.770111	82.872246		
Hierarchical_Cluster	gold	treasury_5_years	treasury_10_years	\	
1	1274.094673	2.731527	2.893817		
2	1361.315824	2.135461	2.316397		
3	1256.049357	1.639991	2.158414		
4	1332.794512	1.864445	3.165509		
5	1401.086251	1.294250	2.227985		
Hierarchical_Cluster	treasury_30_years	close	volume	ttm_net_ep	
1	3.081325	83.957358	1.049441e+08	0.48798	
8	2.714135	87.604840	9.685454e+07	1.07262	
2	2.844770	40.436028	7.721340e+07	0.19670	
6	4.262280	7.973035	1.333078e+08	0.11542	
3	3.172708	17.280055	7.903597e+07	0.03762	
4	...	...	...		
7	...	...	...		
5	...	...	...		
6	...	...	...		
Hierarchical_Cluster	... lt_investments_&_debt	return_on_investment	\		
1	56.927574	12.027160			
2	71.396296	20.264411			
3	28.773517	13.702284			
4	7.188720	29.886585			
5	14.045739	7.780470			
Hierarchical_Cluster	open	high	low	dividends	\
1	84.021601	84.771648	83.064000	0.0	
2	87.675436	88.523884	86.620239	0.0	
3	40.405607	40.732585	40.055842	0.0	
4	7.952610	8.071326	7.850796	0.0	
5	17.292268	17.470585	17.087115	0.0	
Hierarchical_Cluster	stock_splits	long_term_debt_excl._capitalized_leases			
1	0.0	24640.218935			
2	0.0	23468.814815			
3	0.0	10180.086556			
4	0.0	755.533537			
5	0.0	4156.500427			

Hierarchical_Cluster	log_close	Direction
1	4.423190	0.591716
2	4.469882	0.518519
3	3.611597	0.578269
4	2.056157	0.548780
5	2.745490	0.498719

[5 rows x 100 columns]

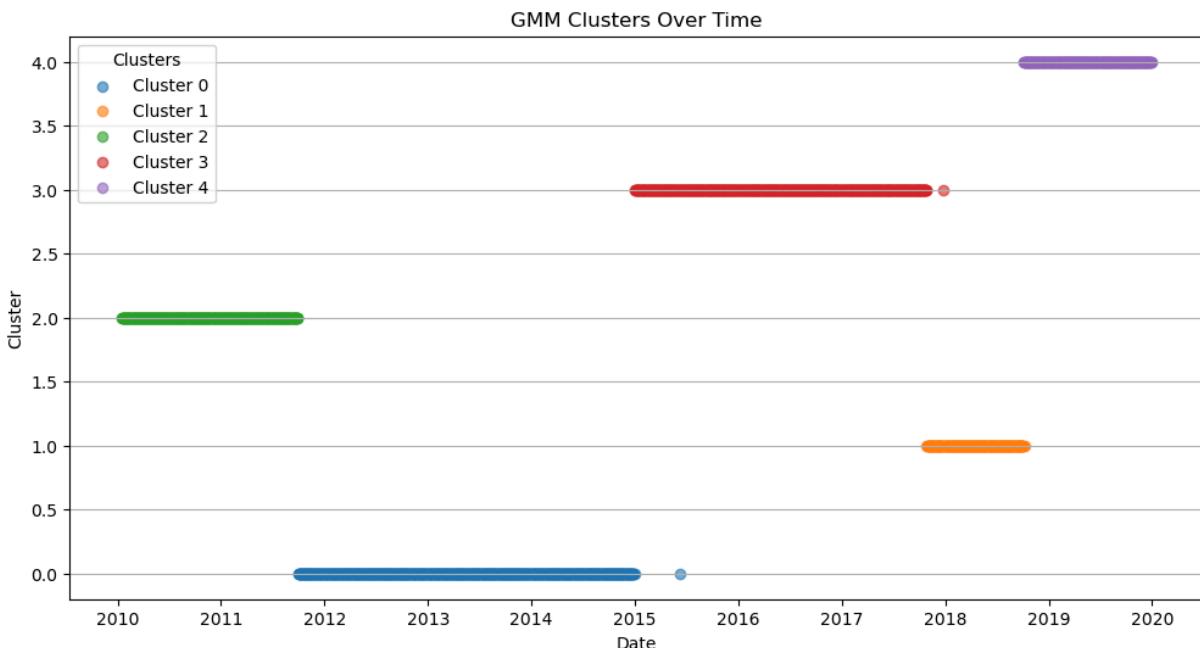
## Model Based Clustering

```
In [78]: from sklearn.mixture import GaussianMixture
import pandas as pd
import matplotlib.pyplot as plt

# Perform Gaussian Mixture Clustering
gmm = GaussianMixture(n_components=5, random_state=42)
gmm_clusters = gmm.fit_predict(X_scaled)

# Add GMM cluster labels to the dataset
train_data3 = train_data.copy()
train_data3.loc[:, 'GMM_Cluster'] = gmm_clusters

# Visualize GMM clusters over time
plt.figure(figsize=(12, 6))
for cluster in sorted(train_data3['GMM_Cluster'].unique()):
    cluster_dates = train_data3[train_data3['GMM_Cluster'] == cluster]['Date']
    plt.scatter(cluster_dates, [cluster] * len(cluster_dates), label=f"Cluster {cluster}")
plt.title("GMM Clusters Over Time")
plt.xlabel("Date")
plt.ylabel("Cluster")
plt.legend(title="Clusters")
plt.grid(axis='y')
plt.show()
```



```
In [79]: # Summarize predictors and response ('close') by cluster
gmm_summary = train_data3.groupby('GMM_Cluster').mean()
print("Gaussian Mixture Model Cluster Summary:")
print(gmm_summary)
```

## Gaussian Mixture Model Cluster Summary:

	Date	dxy	oil	gold
\				
GMM_Cluster				
0	2013-05-17 14:45:28.606356992	81.280220	94.971332	1465.701710
1	2018-04-18 05:49:16.595744768	92.876256	65.150085	1278.402979
2	2010-11-22 07:33:20.000000000	78.963565	86.506921	1362.697686
3	2016-05-31 17:27:38.426966272	96.744551	47.115126	1218.985816
4	2019-05-20 03:05:12.540193024	97.235080	57.290675	1361.814147
	treasury_5_years	treasury_10_years	treasury_30_years	\
GMM_Cluster				
0	1.166368	2.204551	3.221147	
1	2.599098	2.783532	3.015243	
2	1.803449	3.100905	4.217637	
3	1.556171	2.080749	2.775956	
4	2.127071	2.307923	2.710100	
	close	volume	ttm_net_eps	... lt_investments_&_deb
t \				
GMM_Cluster				
0	13.891240	7.684348e+07	0.023081	... 11.43847
2				
1	77.568770	1.023126e+08	0.410043	... 54.99868
1				
2	8.150007	1.298271e+08	0.116088	... 7.40083
3				
3	34.424653	7.438114e+07	0.119073	... 24.50450
8				
4	88.075378	9.247005e+07	1.091961	... 71.79810
3				
	return_on_investment	open	high	low dividend
s \				
GMM_Cluster				
0	6.288753	13.890402	14.044160	13.721925 0.
0				
1	11.216383	77.569564	78.297406	76.715217 0.
0				
2	28.923449	8.139160	8.255052	8.024359 0.
0				
3	10.869607	34.434801	34.713739	34.109520 0.
0				
4	20.550000	88.142476	88.973849	87.116752 0.
0				
	stock_splits	long_term_debt_excl._capitalized_leases	log_clos	
e \				
GMM_Cluster				
0	0.0		2712.063570	2.60781
9				
1	0.0		24679.757447	4.33528
3				
2	0.0		879.884259	2.07691
9				
3	0.0		8553.344101	3.48928

```

4          0.0
4
8

      Direction
GMM_Cluster
0        0.512225
1        0.617021
2        0.516204
3        0.540730
4        0.514469

[5 rows x 100 columns]

```

```
In [80]: from sklearn.metrics import mean_squared_error
import numpy as np
import pandas as pd

# Assume `train_data` has the following cluster labels already added:
# 'KMeans_Cluster', 'Hierarchical_Cluster', 'GMM_Cluster'

# Prepare response variable
y = train_data['log_close']

# Initialize a dictionary to store MSPE for each clustering method
rmse_dict = {}

# Evaluate MSPE for K-Means Clustering
kmeans_means = train_data1.groupby('KMeans_Cluster')['log_close'].transform(
    lambda x: x.mean())
kmeans_rmse = mean_squared_error(y, kmeans_means, squared=False)
rmse_dict['KMeans'] = kmeans_rmse

# Evaluate MSPE for Hierarchical Clustering
hierarchical_means = train_data2.groupby('Hierarchical_Cluster')['log_close'].transform(
    lambda x: x.mean())
hierarchical_rmse = mean_squared_error(y, hierarchical_means, squared=False)
rmse_dict['Hierarchical'] = hierarchical_rmse

# Evaluate MSPE for Model-Based Clustering
gmm_means = train_data3.groupby('GMM_Cluster')['log_close'].transform('mean')
gmm_rmse = mean_squared_error(y, gmm_means, squared=False)
rmse_dict['Model-Based'] = gmm_rmse

```

```
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
/Users/aidanashrafi/opt/anaconda3/envs/mynewbase/lib/python3.9/site-packages/sklearn/metrics/_regression.py:492: FutureWarning: 'squared' is deprecated in version 1.4 and will be removed in 1.6. To calculate the root mean squared error, use the function 'root_mean_squared_error'.
    warnings.warn(
```

In [81]: # Display MSPE for all clustering methods  
print("Clustering RMSEs:")  
for method, rmse in rmse\_dict.items():  
 print(f"{method} Clustering RMSE for Log Close (Log Scale): {rmse}")  
 print(f"{method} Clustering RMSE for Log Close (Original Scale): {np.exp(rmse)})  
 print(f"\n")

Clustering RMSEs:  
KMeans Clustering RMSE for Log Close (Log Scale): 0.23845845586042463  
KMeans Clustering RMSE for Log Close (Original Scale): 1.2692909733396893

Hierarchical Clustering RMSE for Log Close (Log Scale): 0.3723168547812369  
Hierarchical Clustering RMSE for Log Close (Original Scale): 1.4510926941163  
582

Model-Based Clustering RMSE for Log Close (Log Scale): 0.23802173305004115  
Model-Based Clustering RMSE for Log Close (Original Scale): 1.26873676604491  
03

In [82]: from sklearn.linear\_model import LinearRegression, RidgeCV, LassoCV  
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor  
from sklearn.svm import LinearSVR  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import TimeSeriesSplit  
from sklearn.metrics import mean\_squared\_error  
import numpy as np  
import pandas as pd

  
train\_data.loc[:, 'KMeans\_Cluster'] = kmeans\_clusters  
train\_data.loc[:, 'Hierarchical\_Cluster'] = hierarchical\_clusters  
train\_data.loc[:, 'GMM\_Cluster'] = gmm\_clusters  
# Prepare data  
X = train\_data[["low", "vpt", "high", "EP", "gold", "PSAR", "sma\_50-sma\_200",  
 "treasury\_5\_years", "total\_share\_holder\_equity", "open", "bc",  
 "shares\_outstanding", "price\_to\_book\_ratio", "ttm\_sales\_per\_

```

    "current_liabilities", "price_to_sales_ratio", "current_ratio",
    "return_on_equity", "cash_on_hand", "operating_income", "eps",
    "ebitda", "return_on_tangible_equity", "decreasing", "increasing",
    "macd_signal", "stoch", "MACD_10_26_9", "roc", "treasury_30_",
    "KMeans_Cluster"]]

y = train_data["log_close"]

# Standardize features (for models requiring scaling)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Time series cross-validation
tscv = TimeSeriesSplit(n_splits=5)

# Initialize MSPE dictionary
rmse_results = {}

# Linear Regression
linear_rmse = []
linear_model = LinearRegression()
for train_index, test_index in tscv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    linear_model.fit(X_train, y_train)
    y_pred = linear_model.predict(X_test)
    linear_rmse.append(mean_squared_error(y_test, y_pred, squared=False))
rmse_results["Linear Regression"] = np.exp(np.mean(linear_rmse))

# Ridge Regression
ridge_model = RidgeCV(alphas=[0.1, 1.0, 10.0])
ridge_rmse = []
for train_index, test_index in tscv.split(X_scaled):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    ridge_model.fit(X_train, y_train)
    y_pred = ridge_model.predict(X_test)
    ridge_rmse.append(mean_squared_error(y_test, y_pred, squared=False))
rmse_results["Ridge"] = np.exp(np.mean(ridge_rmse))

# Lasso Regression
lasso_model = LassoCV(alphas=np.logspace(-4, 0, 50), max_iter=10000)
lasso_rmse = []
for train_index, test_index in tscv.split(X_scaled):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    lasso_model.fit(X_train, y_train)
    y_pred = lasso_model.predict(X_test)
    lasso_rmse.append(mean_squared_error(y_test, y_pred, squared=False))
rmse_results["Lasso"] = np.exp(np.mean(lasso_rmse))

# Random Forest
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_rmse = []
for train_index, test_index in tscv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]

```

```

y_train, y_test = y.iloc[train_index], y.iloc[test_index]
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
rf_rmse.append(mean_squared_error(y_test, y_pred, squared=False))
rmse_results["Random Forest"] = np.exp(np.mean(rf_rmse))

# Gradient Boosting
gb_model = GradientBoostingRegressor(n_estimators=100, random_state=42)
gb_rmse = []
for train_index, test_index in tscv.split(X):
    X_train, X_test = X.iloc[train_index], X.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    gb_model.fit(X_train, y_train)
    y_pred = gb_model.predict(X_test)
    gb_rmse.append(mean_squared_error(y_test, y_pred, squared=False))
rmse_results["Gradient Boosting"] = np.exp(np.mean(gb_rmse))

# Linear SVM
svr_model = LinearSVR(random_state=42, max_iter=10000, C=0.1)
svr_rmse = []
for train_index, test_index in tscv.split(X_scaled):
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    svr_model.fit(X_train, y_train)
    y_pred = svr_model.predict(X_test)
    svr_rmse.append(mean_squared_error(y_test, y_pred, squared=False))
rmse_results["Linear SVM"] = np.exp(np.mean(svr_rmse))

from sklearn.decomposition import PCA

# PCA Regression
pca_rmse = []
explained_variance_threshold = 0.95 # Set the threshold for explained variance

# Fit PCA on the entire dataset to determine the number of components for 95%
pca_full = PCA().fit(X_scaled)
cumulative_explained_variance = np.cumsum(pca_full.explained_variance_ratio_)
n_components = np.argmax(cumulative_explained_variance >= explained_variance)

# Use the determined number of components for PCA
pca = PCA(n_components=n_components)
pca_model = LinearRegression()

for train_index, test_index in tscv.split(X_scaled):
    # Apply PCA on training and test sets
    X_train, X_test = X_scaled[train_index], X_scaled[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]

    # Fit PCA on training data and transform both training and test data
    X_train_pca = pca.fit_transform(X_train)
    X_test_pca = pca.transform(X_test)

    # Train Linear Regression model on PCA-transformed data
    pca_model.fit(X_train_pca, y_train)
    y_pred = pca_model.predict(X_test_pca)

```

```
# Calculate RMSE
pca_rmse.append(mean_squared_error(y_test, y_pred, squared=False))

# Add PCA Regression RMSE to the results dictionary
rmse_results["PCA Regression"] = np.exp(np.mean(pca_rmse))
```







```
In [83]: # Output MSPE results  
rmse_results
```

```
Out[83]: {'Linear Regression': 1.3166661668616408,
          'Ridge': 1.2355178856459195,
          'Lasso': 1.2071336836237347,
          'Random Forest': 1.269863777706667,
          'Gradient Boosting': 1.2753331882329946,
          'Linear SVM': 1.5272381690448305,
          'PCA Regression': 1.2641572788408475}
```