## Prathik Mohan Chanchala - MSc. Data Science, PRN:23039142546.

# Data Cleaning and Initial EDA of Bahrain's BAX Index for Sem-IV Time Series Project

## Getting Data from Path

```
import pandas as pd

df_bax = pd.read_csv(r'/content/drive/MyDrive/PRN23039142546/BAX_2010_2025_for_analysis.csv')

df_bax.head()
```

| | Date | Price | Open | High | Low | Vol. | Change % |
|---|---|---|---|---|---|---|---|
| **0** | 05/24/2010 | 1,482.42 | 1,491.98 | 1,491.98 | 1,482.42 | 926.98K | -0.64% |
| **1** | 05/25/2010 | 1,454.85 | 1,482.42 | 1,482.42 | 1,454.85 | 1.66M | -1.86% |
| **2** | 05/26/2010 | 1,472.29 | 1,456.50 | 1,472.29 | 1,454.85 | 1.50M | 1.20% |
| **3** | 05/27/2010 | 1,453.82 | 1,472.29 | 1,478.07 | 1,453.82 | 2.48M | -1.25% |
| **4** | 05/30/2010 | 1,455.16 | 1,453.82 | 1,462.04 | 1,453.72 | 5.91M | 0.09% |

Next steps: Generate code with df_bax | View recommended plots | New interactive sheet

## Check Data Types

```
df_bax.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3659 entries, 0 to 3658
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Date      3659 non-null   object
 1   Price     3659 non-null   object
 2   Open      3659 non-null   object
 3   High      3659 non-null   object
 4   Low       3659 non-null   object
 5   Vol.      3640 non-null   object
 6   Change %  3659 non-null   object
dtypes: object(7)
memory usage: 200.2+ KB
```

## Conversion to datetime, Other columns to float, Volume column removal of K and M.

```
# INITIAL OBSERVATION------------>

# Date is object type → Needs to be converted to datetime.

# Price, Open, High, Low, Vol., Change % are all object type - Should be float.

# Vol. has missing values (19 missing).

# All columns are incorrectly typed as object — typical issue when numbers have commas, percentage signs, or are stored as text.
```

```
# DATE OBJECT CONVERTED TO DATETIME DATATYPE

df_bax['Date'].dtype
```

```
dtype('O')
```

```
df_bax['Date'] = pd.to_datetime(df_bax['Date'])
df_bax['Date'].dtype
```

```
dtype('<M8[ns]')
```

```
# NUMERIC COLUMNS HAVE commas, percent signs, etc. so convertING them properly.

df_bax.columns
```

```
Index(['Date', 'Price', 'Open', 'High', 'Low', 'Vol.', 'Change %'], dtype='object')
```

```
# Volume column has K and M as text to indicate killo and Million. This has to be fixed before converting it to numeric, else NaN in Volume.

def convert_volume(val):
  if isinstance(val,str):
    val = val.replace(',','').strip()
    if 'K' in val:
      return float(val.replace('K',''))*1_000
    elif 'M' in val:
      return float(val.replace('M',''))*1_000_000
    elif val == '-':
      return None
    else:
      return float(val)
    return val

df_bax['Vol.'] = df_bax['Vol.'].apply(convert_volume)

```

```
cols_to_clean = ['Price', 'Open', 'High', 'Low', 'Change %']

# Remove commas and percent signs, then convert to float
```

```
4
5 for col in cols_to_clean:
6   df_bax[col] = df_bax[col].str.replace(',','', regex=False)
7   df_bax[col] = df_bax[col].str.replace('%','', regex=False)
8   df_bax[col] = pd.to_numeric(df_bax[col], errors='coerce')
```

```
1 df_bax.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3659 entries, 0 to 3658
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Date      3659 non-null   datetime64[ns]
 1   Price     3659 non-null   float64
 2   Open      3659 non-null   float64
 3   High      3659 non-null   float64
 4   Low       3659 non-null   float64
 5   Vol.      3640 non-null   float64
 6   Change %  3659 non-null   float64
dtypes: datetime64[ns](1), float64(6)
memory usage: 200.2 KB
```

```
1 df_bax.head()
```

| | Date | Price | Open | High | Low | Vol. | Change % |
|---|---|---|---|---|---|---|---|
| **0** | 2010-05-24 | 1482.42 | 1491.98 | 1491.98 | 1482.42 | 926980.0 | -0.64 |
| **1** | 2010-05-25 | 1454.85 | 1482.42 | 1482.42 | 1454.85 | 1660000.0 | -1.86 |
| **2** | 2010-05-26 | 1472.29 | 1456.50 | 1472.29 | 1454.85 | 1500000.0 | 1.20 |
| **3** | 2010-05-27 | 1453.82 | 1472.29 | 1478.07 | 1453.82 | 2480000.0 | -1.25 |
| **4** | 2010-05-30 | 1455.16 | 1453.82 | 1462.04 | 1453.72 | 5910000.0 | 0.09 |

Next steps: Generate code with df_bax | View recommended plots | New interactive sheet

## Set date as Index and Sort

```
1 df_bax.set_index('Date', inplace=True)
2 df_bax.sort_index(inplace=True)
3 df_bax.head()
```

| Date | Price | Open | High | Low | Vol. | Change % |
|---|---|---|---|---|---|---|
| **2010-05-24** | 1482.42 | 1491.98 | 1491.98 | 1482.42 | 926980.0 | -0.64 |
| **2010-05-25** | 1454.85 | 1482.42 | 1482.42 | 1454.85 | 1660000.0 | -1.86 |
| **2010-05-26** | 1472.29 | 1456.50 | 1472.29 | 1454.85 | 1500000.0 | 1.20 |
| **2010-05-27** | 1453.82 | 1472.29 | 1478.07 | 1453.82 | 2480000.0 | -1.25 |
| **2010-05-30** | 1455.16 | 1453.82 | 1462.04 | 1453.72 | 5910000.0 | 0.09 |

Next steps: Generate code with df_bax | View recommended plots | New interactive sheet

## Check Frequency of dates

```
1 print(df_bax.index.min())
2 print(df_bax.index.max())
3 print(df_bax.index.freq)
```

```
2010-05-24 00:00:00
2025-03-27 00:00:00
None
```

```
1 df_bax.index.to_series().diff().value_counts()
```

| Date | count |
|---|---|
| **1 days** | 2848 |
| **3 days** | 702 |
| **4 days** | 35 |
| **2 days** | 28 |
| **6 days** | 20 |
| **5 days** | 17 |
| **7 days** | 7 |
| **17 days** | 1 |

**dtype:** int64

```
1 # Leaving gaps as it is as models will work it out.
```

```
1 # Bahrain index trades from Sunday to Thrusday
```

## Duplicates and Missing Values

```
1 # Check for duplicate dates
2
```

```
duplicate_dates = df_bax.index.duplicated().sum()
print(f"Number of Duplicate Dates: {duplicate_dates}")
```

```
Number of Duplicate Dates: 0
```

```
# Check for entire duplicate rows

duplicate_rows = df_bax.duplicated().sum()
print(f"Number of duplicate rows: {duplicate_rows}")
```

```
Number of duplicate rows: 8
```

```
# No duplicate date in index, but 8 days have same duplicate data or rows data or all column data. That is fine. Because dates are unique, those rows reflect the real market
```

```
df_bax.head()
```

| Date | Price | Open | High | Low | Vol. | Change % |
|---|---|---|---|---|---|---|
| **2010-05-24** | 1482.42 | 1491.98 | 1491.98 | 1482.42 | 926980.0 | -0.64 |
| **2010-05-25** | 1454.85 | 1482.42 | 1482.42 | 1454.85 | 1660000.0 | -1.86 |
| **2010-05-26** | 1472.29 | 1456.50 | 1472.29 | 1454.85 | 1500000.0 | 1.20 |
| **2010-05-27** | 1453.82 | 1472.29 | 1478.07 | 1453.82 | 2480000.0 | -1.25 |
| **2010-05-30** | 1455.16 | 1453.82 | 1462.04 | 1453.72 | 5910000.0 | 0.09 |

Next steps: Generate code with df_bax | View recommended plots | New interactive sheet

```
# Missing Values

df_bax.isnull().sum()
```

| | 0 |
|---|---|
| **Price** | 0 |
| **Open** | 0 |
| **High** | 0 |
| **Low** | 0 |
| **Vol.** | 19 |
| **Change %** | 0 |

**dtype:** int64

```
# Forward filling for 19 missing volume data

df_bax['Vol.'] = df_bax['Vol.'].ffill()
```

```
df_bax.isnull().sum()
```

| | 0 |
|---|---|
| **Price** | 0 |
| **Open** | 0 |
| **High** | 0 |
| **Low** | 0 |
| **Vol.** | 0 |
| **Change %** | 0 |

**dtype:** int64

```
# @title
df_bax.head()
```

| Date | Price | Open | High | Low | Vol. | Change % |
|---|---|---|---|---|---|---|
| **2010-05-24** | 1482.42 | 1491.98 | 1491.98 | 1482.42 | 926980.0 | -0.64 |
| **2010-05-25** | 1454.85 | 1482.42 | 1482.42 | 1454.85 | 1660000.0 | -1.86 |
| **2010-05-26** | 1472.29 | 1456.50 | 1472.29 | 1454.85 | 1500000.0 | 1.20 |
| **2010-05-27** | 1453.82 | 1472.29 | 1478.07 | 1453.82 | 2480000.0 | -1.25 |
| **2010-05-30** | 1455.16 | 1453.82 | 1462.04 | 1453.72 | 5910000.0 | 0.09 |

Next steps: Generate code with df_bax | View recommended plots | New interactive sheet

```
# prompt: export to csv

df_bax.to_csv(r'/content/drive/MyDrive/PRN23039142546/df_bax_cleaned_till_outliers.csv')
df_bax.to_csv(r'/content/drive/MyDrive/PRN23039142546/df_bax_cleaned_to_view_outliers.csv')

# If no outliers - use original dataset above df_bax_cleaned_till_outliers.csv
```

## Prathik Mohan Chanchala - MSc. Data Science, PRN:23039142546.

## Data Cleaning and Initial EDA of Bahrain's BAX Index for Sem-IV Time Series Project

## Outliers - Visualize

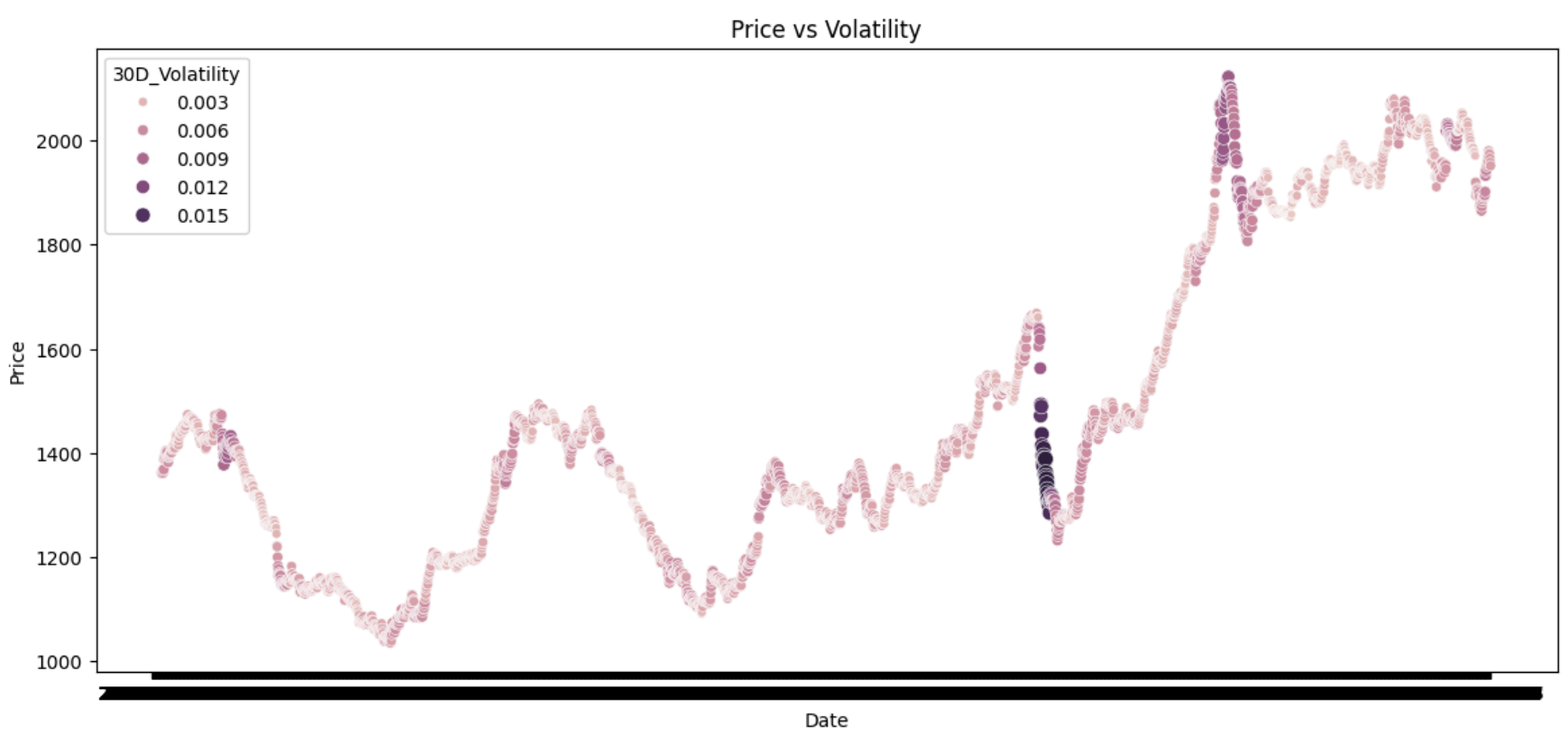
```
# Quick descriptive stats

df_outliers = pd.read_csv(r'/content/drive/MyDrive/PRN23039142546/df_bax_cleaned_to_view_outliers.csv')

df_outliers.describe()
```

| | Price | Open | High | Low | Vol. | Change % |
|---|---|---|---|---|---|---|
| **count** | 3659.000000 | 3659.000000 | 3659.000000 | 3659.000000 | 3.659000e+03 | 3659.000000 |
| **mean** | 1465.173528 | 1465.170697 | 1468.364788 | 1461.654263 | 3.112879e+06 | 0.008623 |
| **std** | 290.889921 | 290.899656 | 291.483077 | 290.544380 | 5.454037e+06 | 0.491530 |
| **min** | 1035.300000 | 1035.260000 | 1040.930000 | 1026.230000 | 1.000000e+04 | -5.820000 |
| **25%** | 1262.710000 | 1262.655000 | 1264.830000 | 1259.205000 | 9.763550e+05 | -0.200000 |
| **50%** | 1397.250000 | 1397.750000 | 1402.690000 | 1393.550000 | 1.950000e+06 | 0.000000 |
| **75%** | 1633.645000 | 1630.420000 | 1638.720000 | 1620.830000 | 3.705000e+06 | 0.220000 |
| **max** | 2122.510000 | 2168.590000 | 2168.590000 | 2119.220000 | 1.832500e+08 | 3.760000 |

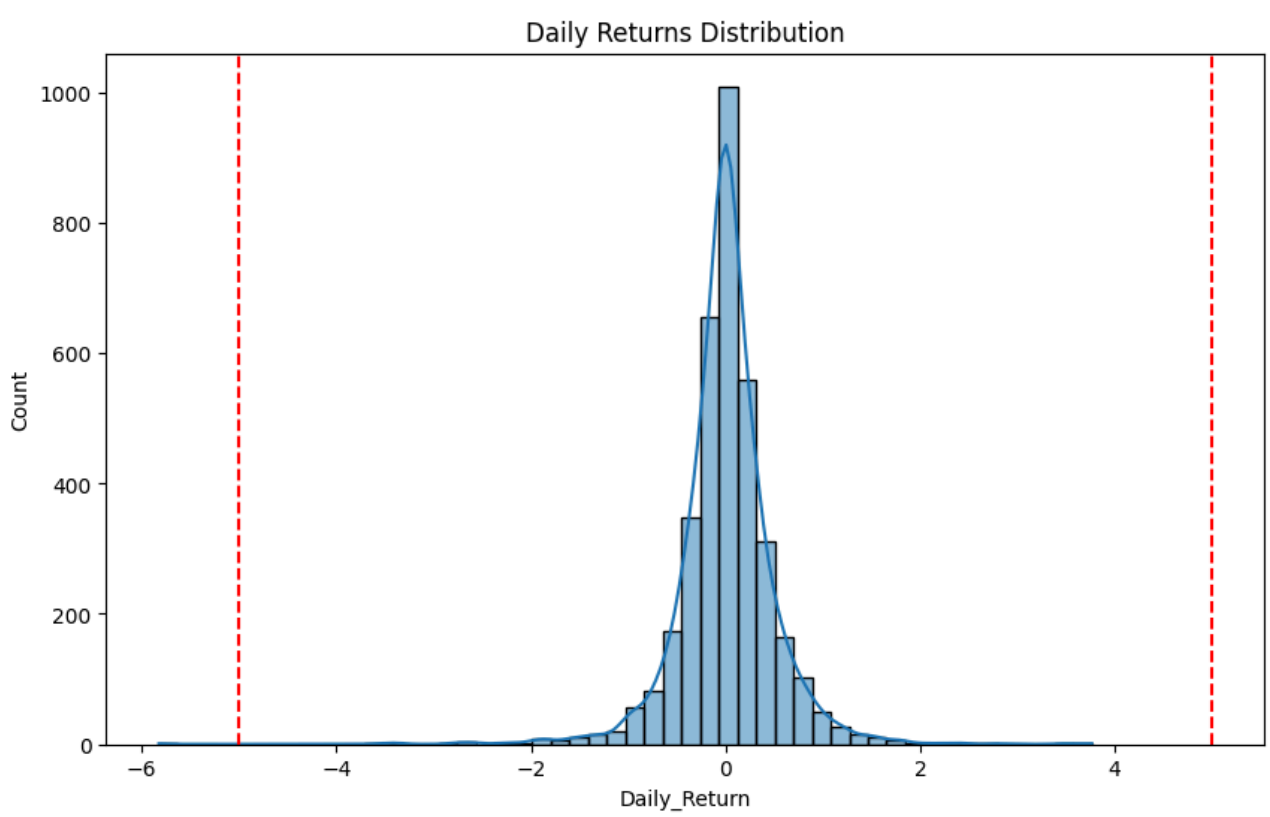
```
# Visual outlier check - effective visualizations using Seaborn that align with financial data analysis:

# Line plot - Price over time

import seaborn as sns
import matplotlib.pyplot as plt
```

```
# 1. Time Series Plot (Most Important)
# Checks: Sudden spikes/drops, data gaps, inconsistent patterns
# Why: Identifies temporal anomalies in context

# Price over Time

plt.figure(figsize=(14, 6))
sns.lineplot(data=df_outliers, x='Date', y='Price')
plt.title('Price Trend Analysis')
plt.ylabel('Price')
plt.axhline(y=df_outliers['Price'].mean(), color='r', linestyle='--', label='Mean')
plt.legend()
plt.show()
```

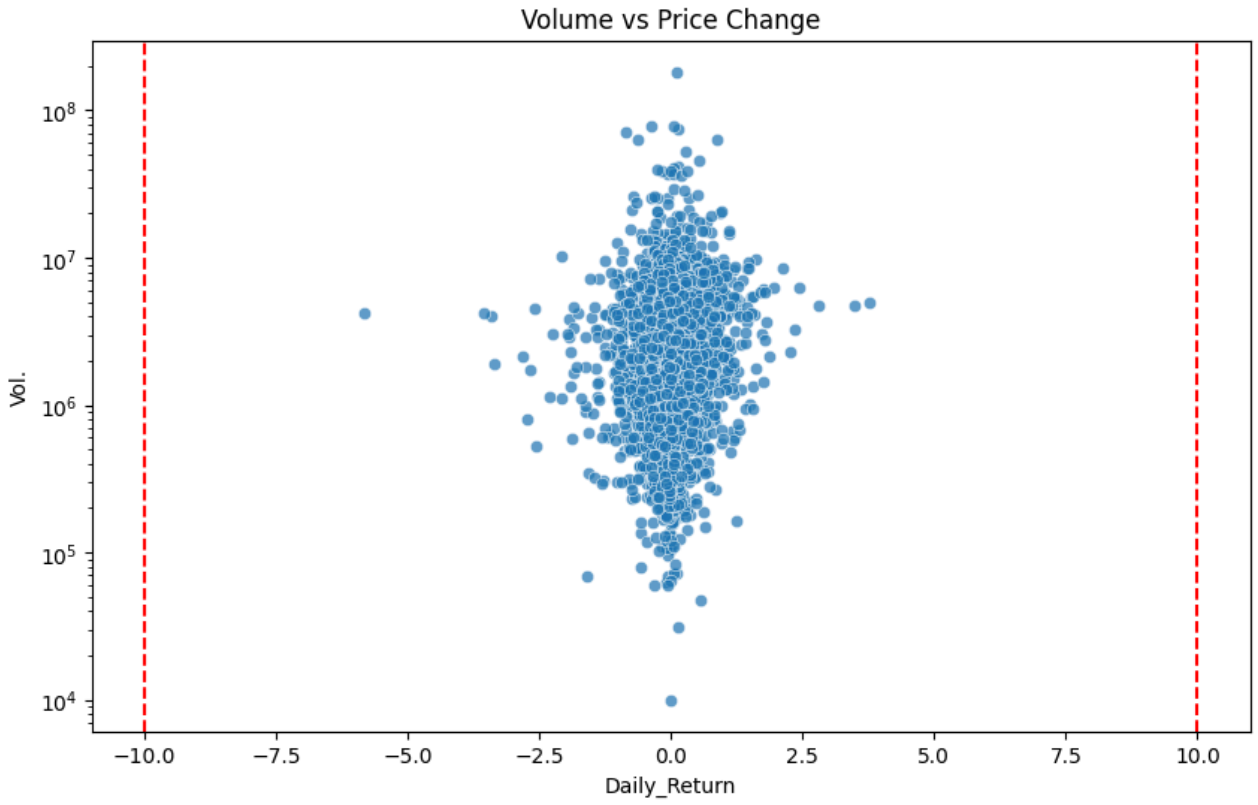

```
# 2. Volatility-Adjusted Price Plot
# Checks: Outliers relative to recent volatility

# Why: Accounts for changing market conditions

df_outliers['30D_Volatility'] = df_outliers['Price'].pct_change().rolling(30).std()
plt.figure(figsize=(14, 6))
sns.scatterplot(data=df_outliers, x='Date', y='Price', size='30D_Volatility', hue='30D_Volatility')
plt.title('Price vs Volatility')
plt.show()
```




```
# 3. Daily Returns Distribution
# Checks: Abnormal price changes (e.g., >10% daily moves)

# Why: Extreme returns often indicate data errors or market events

df_outliers['Daily_Return'] = df_outliers['Price'].pct_change() * 100
plt.figure(figsize=(10, 6))
sns.histplot(data=df_outliers, x='Daily_Return', kde=True, bins=50)
plt.axvline(5, color='r', linestyle='--')
plt.axvline(-5, color='r', linestyle='--')
plt.title('Daily Returns Distribution')
plt.show()
```




```
# 4. Trading Volume vs Price Change
# Checks: Unusual volume-price relationships

# Why: Spikes on low volume = potential errors; spikes on high volume = market events

plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_outliers, x='Daily_Return', y='Vol.', alpha=0.7)
plt.yscale('log')  # Log scale for better visualization
plt.axvline(10, color='r', linestyle='--')
plt.axvline(-10, color='r', linestyle='--')
plt.title('Volume vs Price Change')
plt.show()
```

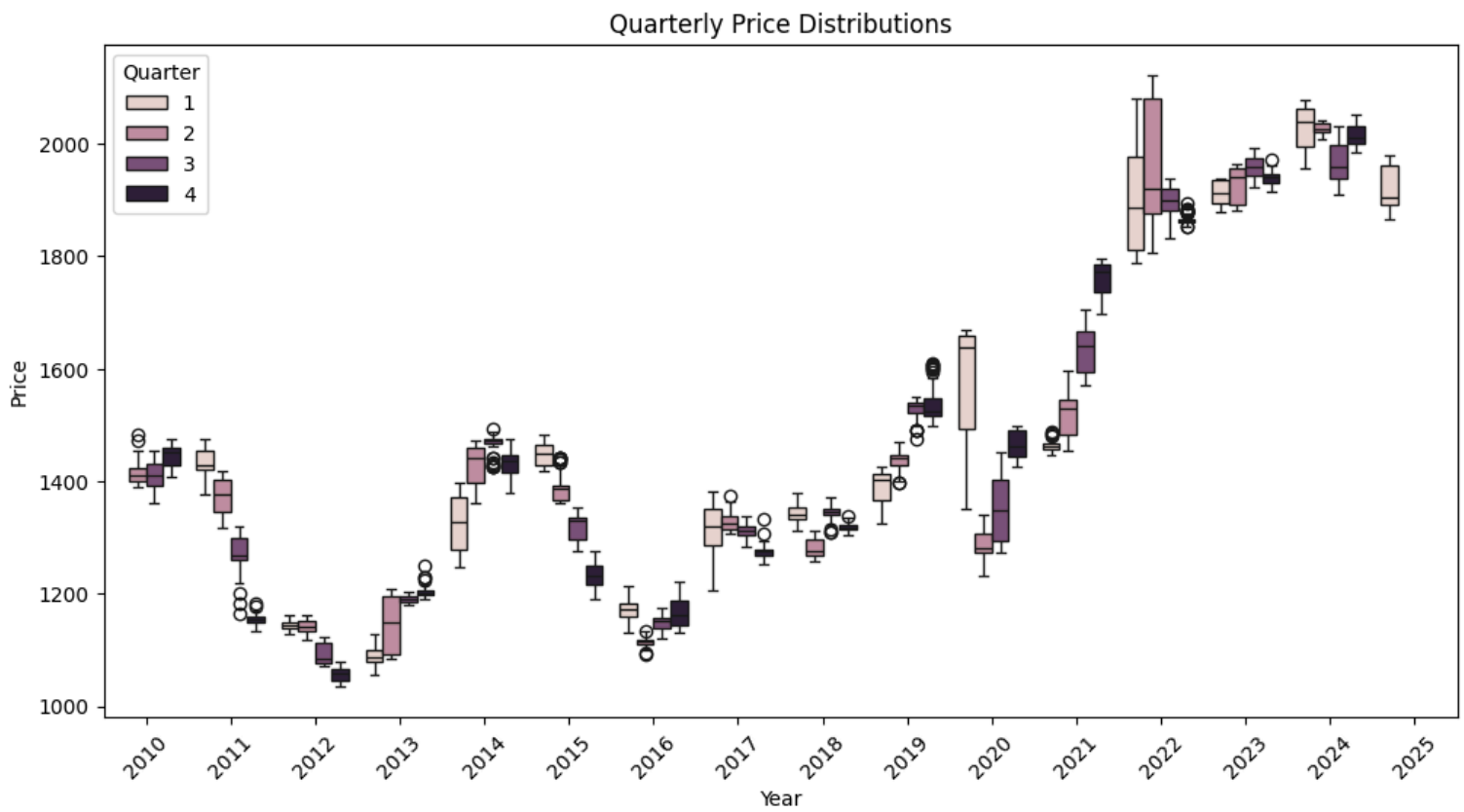



```
df_outliers.head()
```

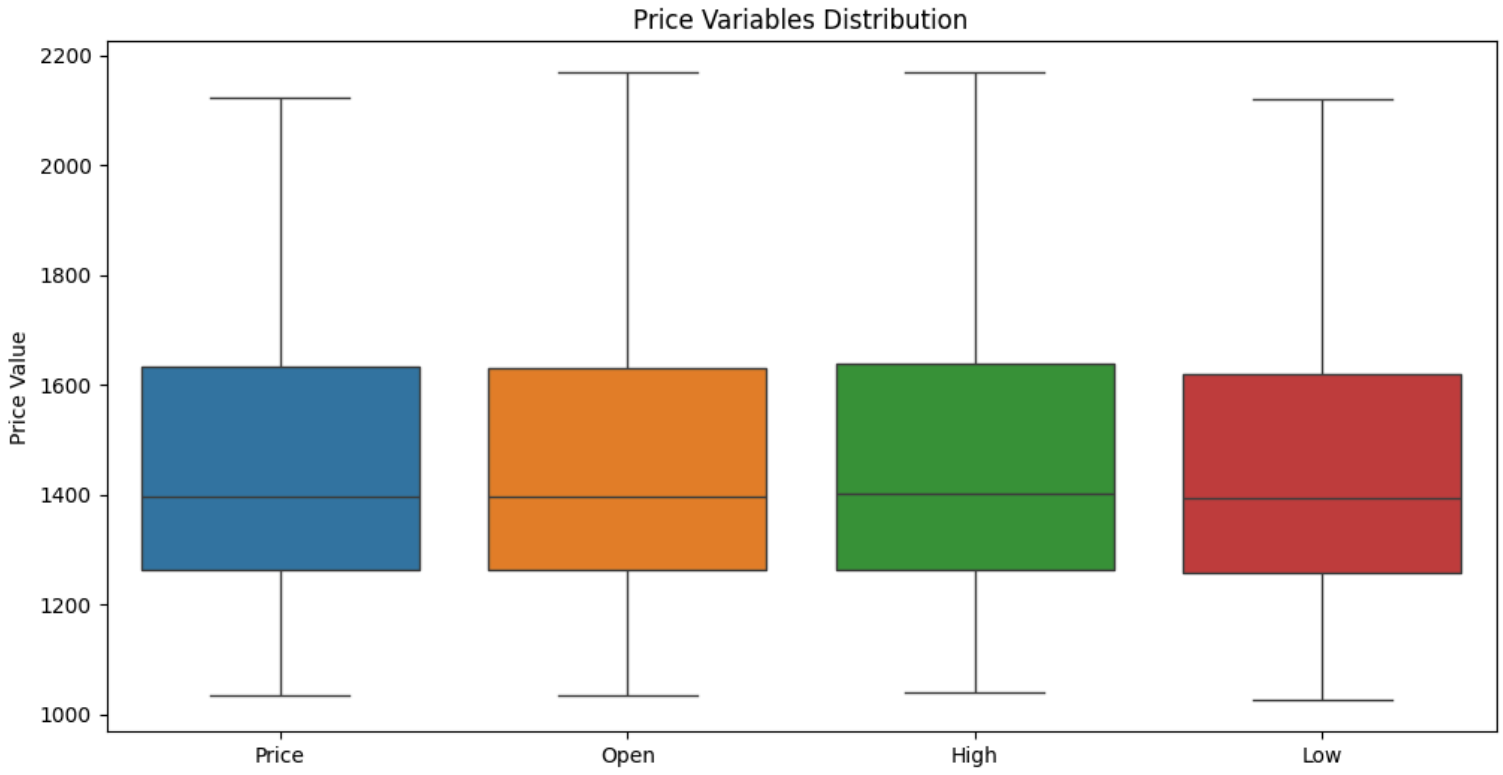
| | Date | Price | Open | High | Low | Vol. | Change % | 30D_Volatility | Daily_Return |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 2010-05-24 | 1482.42 | 1491.98 | 1491.98 | 1482.42 | 926980.0 | -0.64 | NaN | NaN |
| 1 | 2010-05-25 | 1454.85 | 1482.42 | 1482.42 | 1454.85 | 1660000.0 | -1.86 | NaN | -1.859797 |
| 2 | 2010-05-26 | 1472.29 | 1456.50 | 1472.29 | 1454.85 | 1500000.0 | 1.20 | NaN | 1.198749 |
| 3 | 2010-05-27 | 1453.82 | 1472.29 | 1478.07 | 1453.82 | 2480000.0 | -1.25 | NaN | -1.254508 |
| 4 | 2010-05-30 | 1455.16 | 1453.82 | 1462.04 | 1453.72 | 5910000.0 | 0.09 | NaN | 0.092171 |

Next steps: Generate code with `df_outliers` | View recommended plots | New interactive sheet

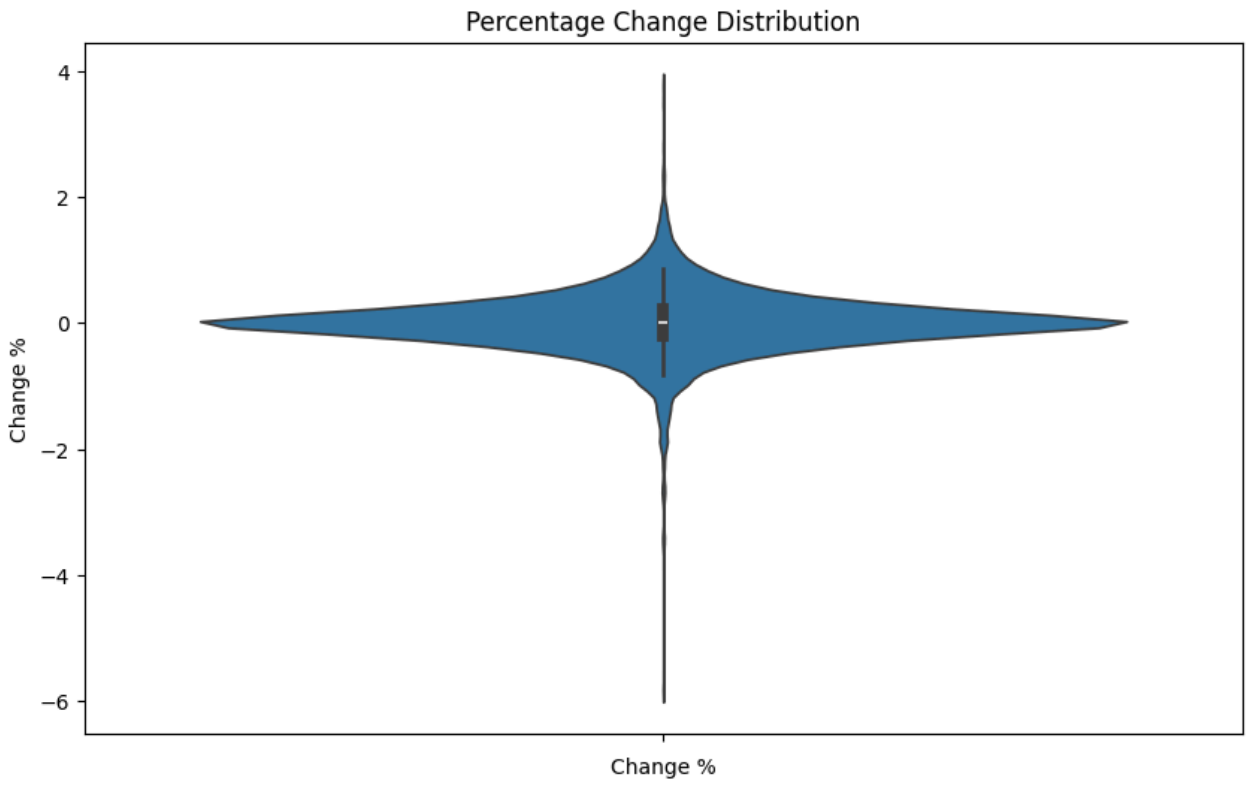
```
# 5. Box Plots (Per Quarter/Year)
# Checks: Seasonal outliers

# Why: Identifies values inconsistent with period norms
df_bax_2 = df_outliers.copy()
df_bax_2.reset_index(inplace=True)

# Convert 'Date' column to datetime
df_bax_2['Date'] = pd.to_datetime(df_bax_2['Date'])

df_bax_2['Year'] = df_bax_2['Date'].dt.year
df_bax_2['Quarter'] = df_bax_2['Date'].dt.quarter
plt.figure(figsize=(12, 6))
sns.boxplot(data=df_bax_2, x='Year', y='Price', hue='Quarter')
plt.title('Quarterly Price Distributions')
plt.xticks(rotation=45)
plt.show()
```



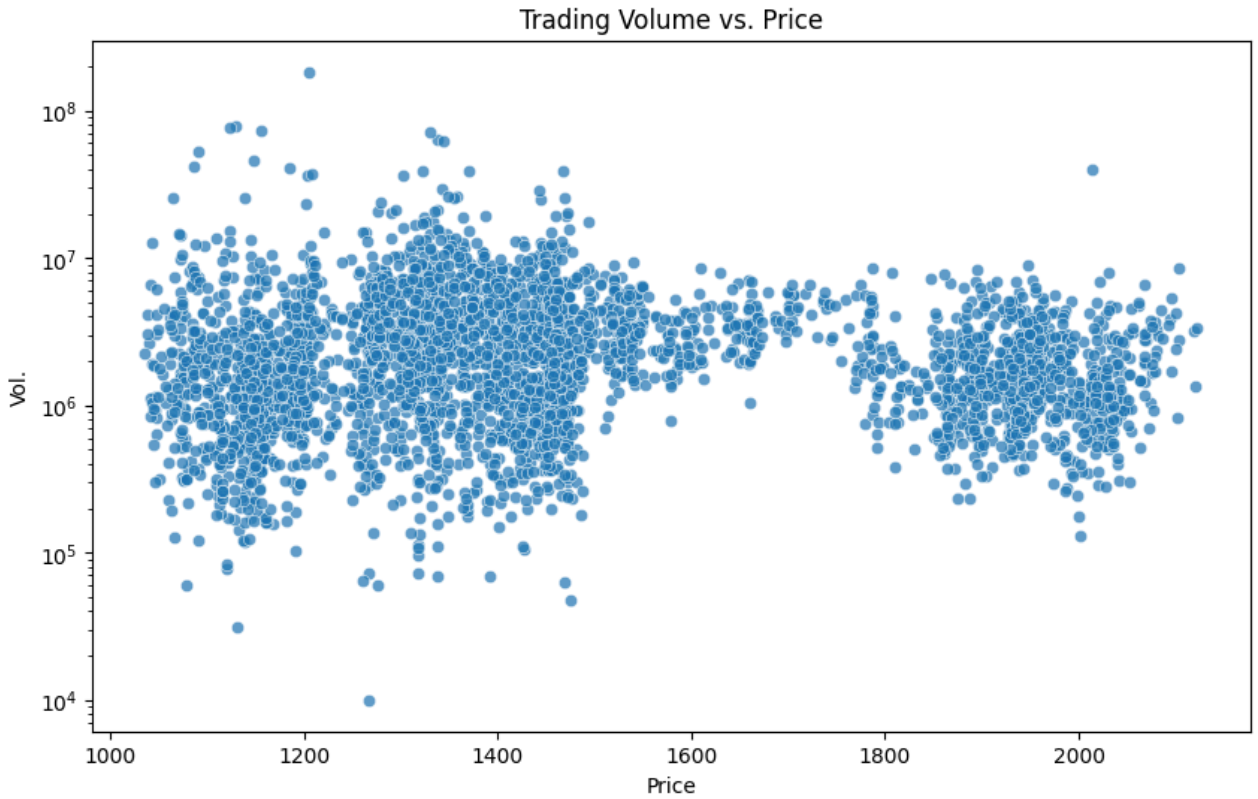

```
plt.figure(figsize=(12, 6))
sns.boxplot(data=df_outliers[['Price', 'Open', 'High', 'Low']])
plt.title('Price Variables Distribution')
plt.ylabel('Price Value')
plt.show()
```




```
plt.figure(figsize=(10, 6))
sns.violinplot(data=df_outliers['Change %'])
plt.title('Percentage Change Distribution')
plt.xlabel('Change %')
plt.show()
```

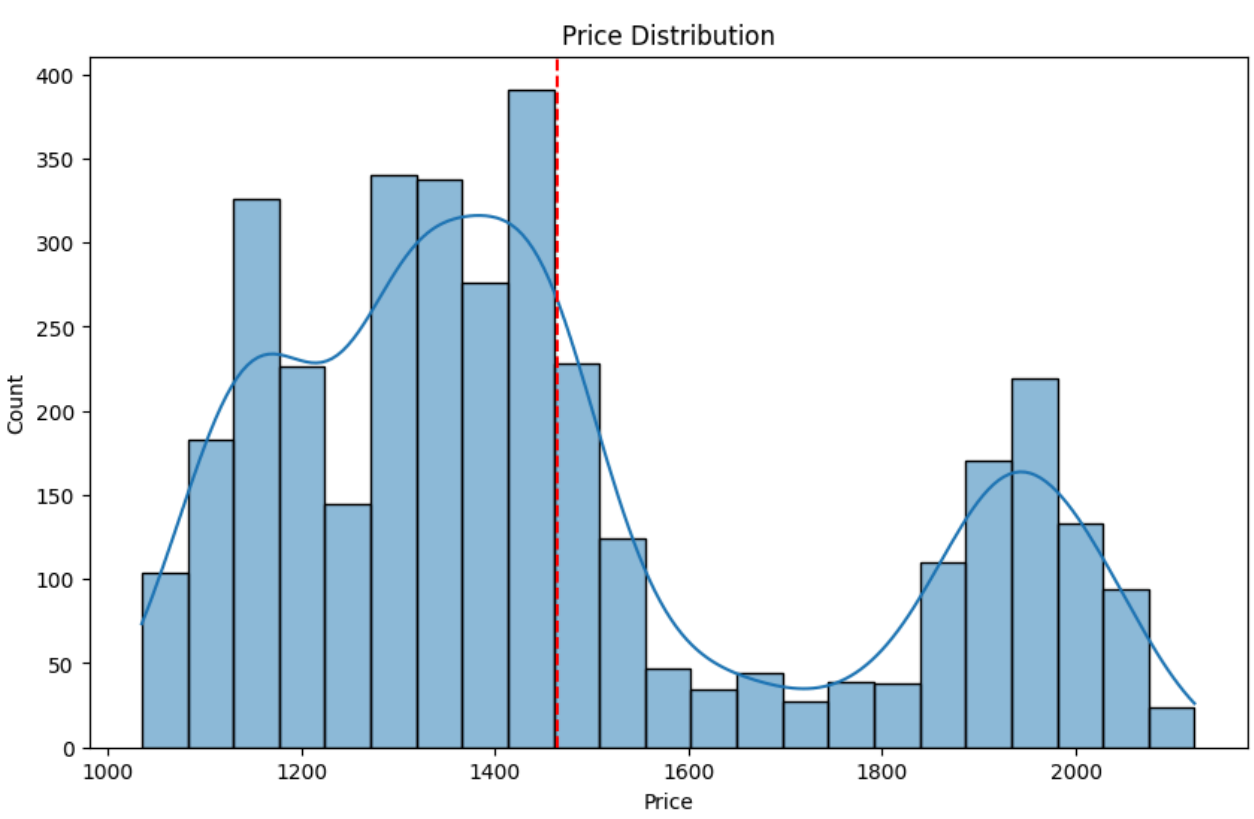



```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_outliers, x='Price', y='Vol.', alpha=0.7)
plt.title('Trading Volume vs. Price')
plt.yscale('log')  # Log scale for better visualization
plt.show()
```




```
plt.figure(figsize=(10, 6))
sns.histplot(df_outliers['Price'], kde=True)
plt.axvline(df_outliers['Price'].mean(), color='r', linestyle='--')
```

```
plt.title('Price Distribution')
plt.show()
```




```
# Key Financial Outlier Checks:

from scipy import stats
z_scores = stats.zscore(df_outliers['Change %'])
change_outliers = df_outliers[(abs(z_scores) > 3)]
change_outliers
```

| | Date | Price | Open | High | Low | Vol. | Change % | 30D_Volatility | Daily_Return |
|---|---|---|---|---|---|---|---|---|---|
| **1** | 2010-05-25 | 1454.85 | 1482.42 | 1482.42 | 1454.85 | 1660000.0 | -1.86 | NaN | -1.859797 |
| **7** | 2010-06-02 | 1415.65 | 1421.44 | 1422.81 | 1409.23 | 3880000.0 | -1.94 | NaN | -1.943604 |
| **35** | 2010-07-12 | 1389.28 | 1376.80 | 1389.28 | 1376.80 | 1020000.0 | 1.51 | 0.006517 | 1.514000 |
| **181** | 2011-02-07 | 1472.84 | 1450.76 | 1472.84 | 1450.76 | 5480000.0 | 1.52 | 0.005841 | 1.521961 |
| **193** | 2011-02-24 | 1432.61 | 1466.25 | 1466.25 | 1432.09 | 795560.0 | -2.73 | 0.007855 | -2.726834 |
| **198** | 2011-03-03 | 1377.36 | 1409.79 | 1409.79 | 1375.66 | 1130000.0 | -2.30 | 0.009066 | -2.300343 |
| **205** | 2011-03-14 | 1395.06 | 1417.22 | 1417.22 | 1395.06 | 341960.0 | -1.56 | 0.009166 | -1.563625 |
| **209** | 2011-03-20 | 1391.81 | 1414.44 | 1414.44 | 1391.81 | 68410.0 | -1.60 | 0.009732 | -1.599926 |
| **343** | 2011-09-25 | 1221.73 | 1225.49 | 1225.49 | 1221.73 | 587170.0 | -1.87 | 0.004373 | -1.869864 |
| **345** | 2011-09-27 | 1200.16 | 1220.08 | 1220.08 | 1200.16 | 901630.0 | -1.63 | 0.005159 | -1.632680 |
| **347** | 2011-09-29 | 1165.75 | 1183.25 | 1183.25 | 1164.55 | 883910.0 | -1.48 | 0.006058 | -1.478977 |
| **528** | 2012-06-13 | 1116.99 | 1134.60 | 1135.26 | 1116.99 | 644520.0 | -1.55 | 0.004014 | -1.552089 |
| **718** | 2013-03-25 | 1085.40 | 1115.23 | 1115.23 | 1085.40 | 1720000.0 | -2.67 | 0.006606 | -2.674785 |
| **907** | 2013-12-31 | 1248.86 | 1232.31 | 1248.86 | 1225.62 | 9830000.0 | 1.61 | 0.004451 | 1.609334 |
| **965** | 2014-03-25 | 1356.42 | 1377.65 | 1377.86 | 1355.91 | 7260000.0 | -1.54 | 0.006768 | -1.541030 |
| **1042** | 2014-07-13 | 1463.78 | 1441.24 | 1463.78 | 1441.24 | 1340000.0 | 1.57 | 0.004152 | 1.570979 |
| **1216** | 2015-04-01 | 1426.59 | 1429.21 | 1429.21 | 1425.88 | 979310.0 | -1.61 | 0.004745 | -1.612447 |
| **1226** | 2015-04-15 | 1393.94 | 1434.12 | 1437.68 | 1393.94 | 2120000.0 | -2.80 | 0.006859 | -2.801718 |
| **1411** | 2016-01-19 | 1175.76 | 1196.27 | 1196.27 | 1173.25 | 1120000.0 | -1.71 | 0.005440 | -1.714496 |
| **1462** | 2016-03-30 | 1139.95 | 1162.17 | 1162.17 | 1133.43 | 1330000.0 | -1.91 | 0.005566 | -1.911941 |
| **1505** | 2016-05-31 | 1111.56 | 1094.41 | 1111.56 | 1094.41 | 5460000.0 | 1.57 | 0.003894 | 1.567054 |
| **1624** | 2016-11-28 | 1177.15 | 1194.28 | 1194.28 | 1177.15 | 1800000.0 | -1.61 | 0.005724 | -1.613105 |
| **1659** | 2017-01-23 | 1274.36 | 1259.55 | 1280.42 | 1259.55 | 4780000.0 | 2.79 | 0.006120 | 2.792521 |
| **1661** | 2017-01-25 | 1300.35 | 1277.93 | 1302.52 | 1277.93 | 6120000.0 | 1.75 | 0.006607 | 1.754400 |
| **1681** | 2017-02-22 | 1342.75 | 1328.74 | 1343.16 | 1328.74 | 3050000.0 | 1.72 | 0.007119 | 1.721944 |
| **1707** | 2017-03-30 | 1355.99 | 1379.72 | 1381.84 | 1339.90 | 4230000.0 | -1.76 | 0.006147 | -1.759076 |
| **1713** | 2017-04-09 | 1374.74 | 1353.36 | 1374.74 | 1353.36 | 4130000.0 | 1.58 | 0.006059 | 1.579772 |
| **1889** | 2017-12-31 | 1331.71 | 1307.95 | 1331.71 | 1307.95 | 6210000.0 | 1.97 | 0.005696 | 1.966264 |
| **2264** | 2019-07-07 | 1537.05 | 1517.01 | 1537.05 | 1517.01 | 5830000.0 | 1.71 | 0.004232 | 1.710561 |
| **2311** | 2019-09-18 | 1489.37 | 1513.93 | 1514.19 | 1481.90 | 2910000.0 | -1.62 | 0.004341 | -1.622268 |
| **2383** | 2020-01-05 | 1575.67 | 1597.86 | 1597.86 | 1575.67 | 3050000.0 | -2.26 | 0.005451 | -2.256754 |
| **2423** | 2020-03-01 | 1604.56 | 1660.48 | 1660.48 | 1600.51 | 1900000.0 | -3.37 | 0.006690 | -3.367701 |
| **2424** | 2020-03-02 | 1640.95 | 1604.56 | 1640.95 | 1604.50 | 2300000.0 | 2.27 | 0.007940 | 2.267911 |
| **2428** | 2020-03-08 | 1562.64 | 1617.26 | 1617.26 | 1562.39 | 3980000.0 | -3.41 | 0.010085 | -3.408971 |
| **2429** | 2020-03-09 | 1471.62 | 1561.84 | 1561.84 | 1468.69 | 4250000.0 | -5.82 | 0.014425 | -5.824758 |
| **2432** | 2020-03-12 | 1436.36 | 1473.51 | 1473.51 | 1432.91 | 4170000.0 | -3.55 | 0.015764 | -3.552747 |
| **2433** | 2020-03-15 | 1414.58 | 1436.36 | 1446.37 | 1414.58 | 3910000.0 | -1.52 | 0.015838 | -1.516333 |
| **2437** | 2020-03-19 | 1408.58 | 1374.90 | 1409.22 | 1372.32 | 6240000.0 | 2.45 | 0.016766 | 2.449633 |
| **2440** | 2020-03-24 | 1360.29 | 1389.39 | 1390.06 | 1359.42 | 10180000.0 | -2.09 | 0.016853 | -2.089511 |
| **2441** | 2020-03-25 | 1385.06 | 1361.39 | 1390.62 | 1361.31 | 3630000.0 | 1.82 | 0.017423 | 1.820935 |
| **2443** | 2020-03-29 | 1362.41 | 1384.12 | 1384.12 | 1362.41 | 2310000.0 | -1.90 | 0.017585 | -1.898789 |
| **2455** | 2020-04-14 | 1307.58 | 1284.99 | 1307.58 | 1284.99 | 1440000.0 | 1.76 | 0.016241 | 1.757990 |
| **2514** | 2020-07-12 | 1313.53 | 1286.33 | 1313.53 | 1286.33 | 8440000.0 | 2.11 | 0.004529 | 2.114543 |
| **2536** | 2020-08-17 | 1348.52 | 1334.22 | 1348.52 | 1332.82 | 4210000.0 | 1.52 | 0.006491 | 1.517661 |
| **2544** | 2020-09-01 | 1405.62 | 1380.89 | 1405.62 | 1380.89 | 5890000.0 | 1.79 | 0.006288 | 1.790874 |
| **2629** | 2021-01-05 | 1456.05 | 1483.30 | 1483.30 | 1452.53 | 3310000.0 | -1.84 | 0.005754 | -1.837120 |
| **2664** | 2021-02-23 | 1460.46 | 1470.26 | 1470.26 | 1458.57 | 4620000.0 | -1.84 | 0.004595 | -1.836953 |
| **2850** | 2021-11-28 | 1744.64 | 1778.69 | 1778.69 | 1743.76 | 2900000.0 | -1.94 | 0.005038 | -1.944099 |
| **2905** | 2022-02-16 | 1898.94 | 1865.54 | 1898.94 | 1865.54 | 2780000.0 | 1.79 | 0.004812 | 1.790366 |
| **2910** | 2022-02-23 | 1964.32 | 1945.53 | 1972.65 | 1938.54 | 2160000.0 | 1.86 | 0.005911 | 1.863212 |
| **2918** | 2022-03-07 | 2062.18 | 1998.68 | 2063.56 | 1998.68 | 4770000.0 | 3.48 | 0.008511 | 3.482572 |
| **2924** | 2022-03-15 | 1969.37 | 2005.30 | 2005.30 | 1969.37 | 1810000.0 | -1.79 | 0.010313 | -1.791752 |
| **2932** | 2022-03-27 | 2080.43 | 2039.17 | 2085.37 | 2039.17 | 3240000.0 | 2.34 | 0.011215 | 2.342067 |
| **2959** | 2022-05-10 | 1988.28 | 2027.92 | 2027.92 | 1984.74 | 3030000.0 | -1.95 | 0.008168 | -1.954712 |
| **2961** | 2022-05-12 | 1959.67 | 2023.52 | 2024.26 | 1959.67 | 4570000.0 | -2.58 | 0.009418 | -2.583948 |
| **2966** | 2022-05-19 | 1922.14 | 1962.96 | 1962.96 | 1922.14 | 1110000.0 | -2.09 | 0.008879 | -2.086995 |
| **3524** | 2024-08-28 | 1958.06 | 1932.21 | 1959.30 | 1932.21 | 951780.0 | 1.55 | 0.005847 | 1.549121 |
| **3537** | 2024-09-17 | 2017.98 | 1944.76 | 2019.26 | 1944.76 | 4960000.0 | 3.76 | 0.008326 | 3.764989 |
| **3617** | 2025-01-13 | 1919.48 | 1968.21 | 1968.21 | 1919.11 | 531540.0 | -2.57 | 0.004967 | -2.566445 |
| **3644** | 2025-02-19 | 1932.21 | 1901.52 | 1932.54 | 1901.52 | 1760000.0 | 1.61 | 0.006927 | 1.613972 |

```
# Price Validation:

# Negative prices? (should never happen)
```

```
# Prices deviating >5% from (Open+High+Low)/3

df_outliers['Avg_Price'] = (df_outliers['Open'] + df_outliers['High'] + df_outliers['Low']) / 3
df_outliers['Price_Deviation'] = abs(df_outliers['Price'] - df_outliers['Avg_Price']) / df_outliers['Avg_Price'] * 100
potential_errors = df_outliers[df_outliers['Price_Deviation'] > 5]
potential_errors
```

| Date | Price | Open | High | Low | Vol. | Change % | 30D_Volatility | Daily_Return | Avg_Price | Price_Deviation |
|---|---|---|---|---|---|---|---|---|---|---|

## Plot 4 Time Series Components - trend, seasonality, cyclicality, and noise.

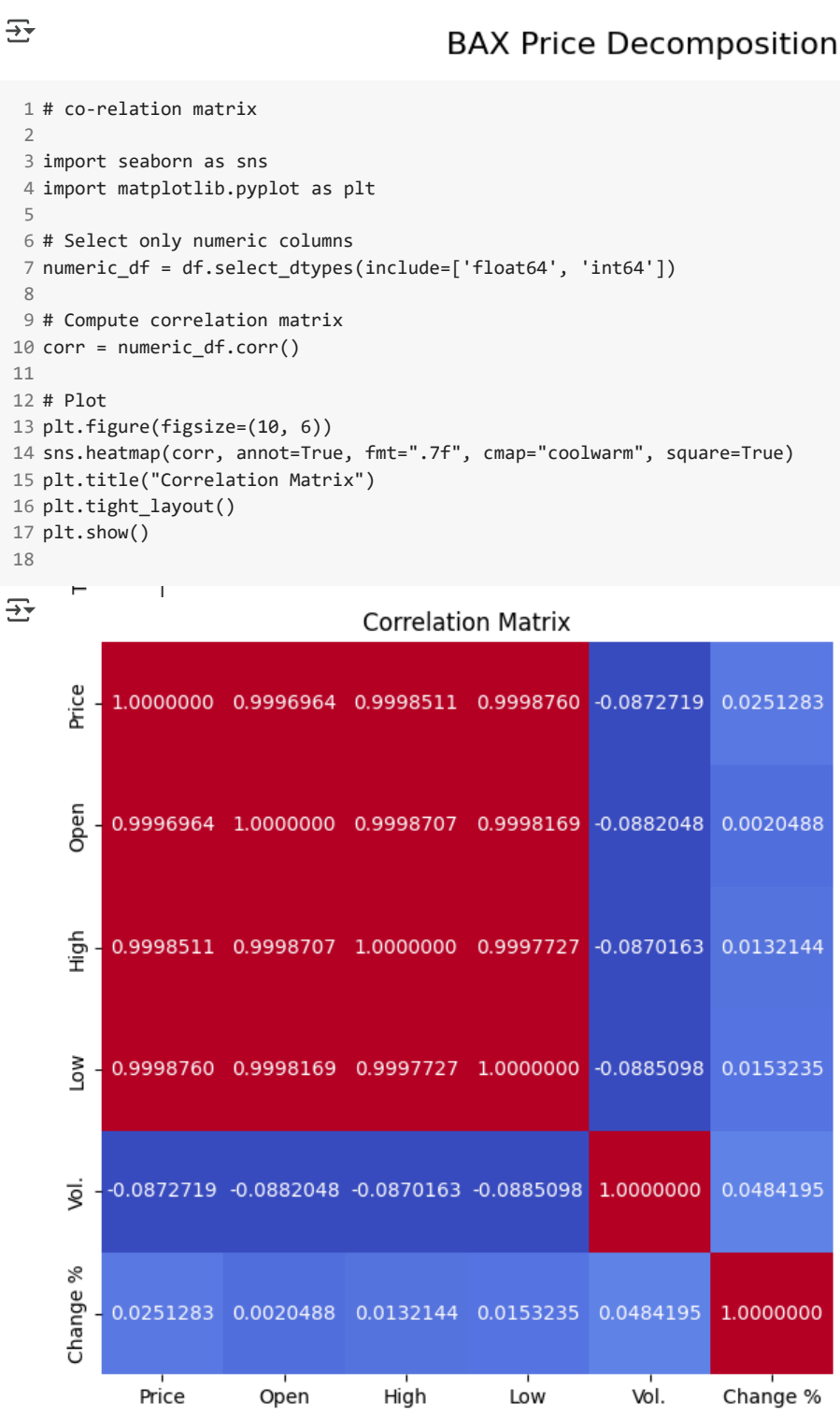
```
#df = pd.read_csv(r'/content/drive/MyDrive/PRN23039142546/df_bax_cleaned_till_outliers.csv')
df_bax.head()
```

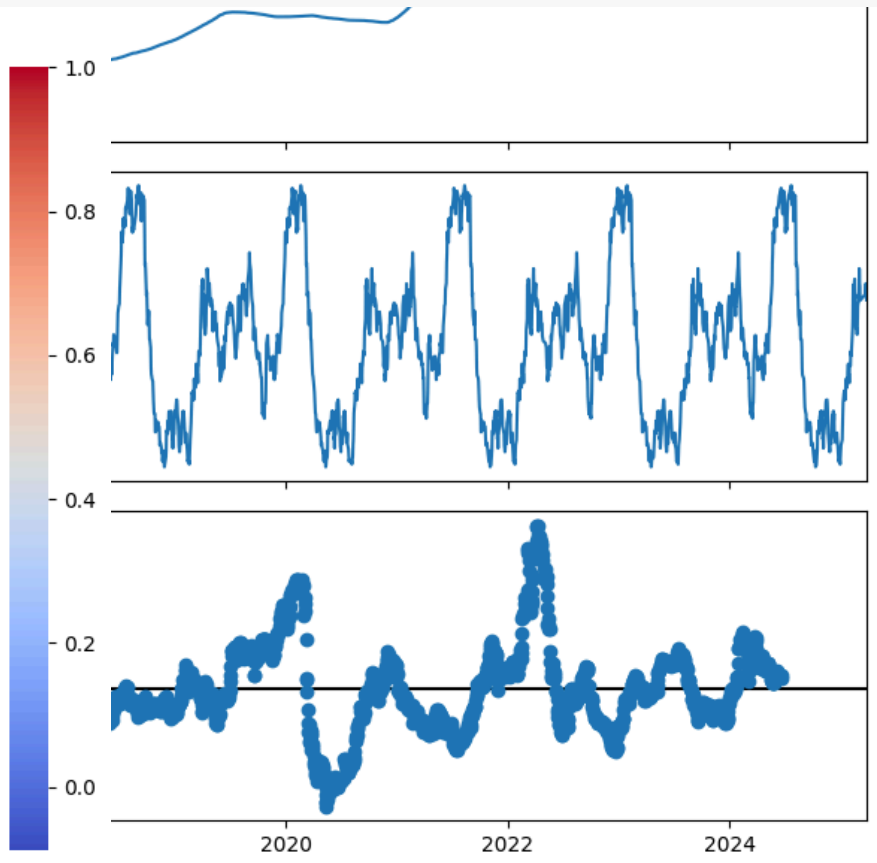
| Date | Price | Open | High | Low | Vol. | Change % |
|---|---|---|---|---|---|---|
| **2010-05-24** | 1482.42 | 1491.98 | 1491.98 | 1482.42 | 926980.0 | -0.64 |
| **2010-05-25** | 1454.85 | 1482.42 | 1482.42 | 1454.85 | 1660000.0 | -1.86 |
| **2010-05-26** | 1472.29 | 1456.50 | 1472.29 | 1454.85 | 1500000.0 | 1.20 |
| **2010-05-27** | 1453.82 | 1472.29 | 1478.07 | 1453.82 | 2480000.0 | -1.25 |
| **2010-05-30** | 1455.16 | 1453.82 | 1462.04 | 1453.72 | 5910000.0 | 0.09 |

Next steps: Generate code with df_bax | View recommended plots | New interactive sheet

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose

# 1. Load data
df = pd.read_csv(r"/content/drive/MyDrive/PRN23039142546/df_bax_cleaned_till_outliers.csv")
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)

# 2. Decompose time series (assuming 'Price' is target)
# result = seasonal_decompose(df['Price'], model='multiplicative', period=30)  # Adjust period if needed
result = seasonal_decompose(df['Price'], model='additive', period=360)  # Adjust period if needed

# 3. Plot
plt.rcParams.update({'figure.figsize': (12, 10)})
result.plot()
plt.suptitle("BAX Price Decomposition - Trend, Seasonality, Residuals", fontsize=16)
plt.tight_layout()
plt.show()

```

```
# co-relation matrix

import seaborn as sns
import matplotlib.pyplot as plt

# Select only numeric columns
numeric_df = df.select_dtypes(include=['float64', 'int64'])

# Compute correlation matrix
corr = numeric_df.corr()

# Plot
plt.figure(figsize=(10, 6))
sns.heatmap(corr, annot=True, fmt=".7f", cmap="coolwarm", square=True)
plt.title("Correlation Matrix")
plt.tight_layout()
plt.show()

```




```
# Stationary check - Augmented Dickey-Fuller (ADF) Test

from statsmodels.tsa.stattools import adfuller

result = adfuller(df['Price'].dropna())
print(f"ADF Statistic: {result[0]}")
print(f"p-value: {result[1]}")

```

```
ADF Statistic: -0.6937043727986383
p-value: 0.8483735147282065
```

```
#  Interpretation:
# p-value < 0.05 → reject null hypothesis → stationary

# p-value ≥ 0.05 → fail to reject → non-stationary → need differencing
```