Australian Stock Price Prediction

Phase 1: Data Preperation and Visualization

Group Name : Project Group 51

Name	Student ID
Hishikesh Phukan	s4031214
Ashif Parwez	s4021513

Table of Contents

Introduction

Dataset Source

The dataset used in this project is adapted from Kaggl(Bellet, 2020). It contains historical stock price for top 100 companies registered with Australian Stock Exchange. The data span from 2015 to 2020 for the companies and trades that take place at that that span of time. It includes daily opening, closing, highest, lowest prices and volumes of trades that took place in the market. This comprehensive dataset serves as a reliable foundation for analyzing stock market trends, conducting preprocessing tasks, and applying visualization techniques for stock prediction modeling.

Dataset Details

The dataset contains historical stock price records from the top 100 listed firms from 2015 to 2020. Every record corresponds to the trading activity on the shares of one company on a particular trading day. The major fields in the dataset are the date, company name or symbol, open price, and closing price. The fields are important in analyzing the behavior of stocks and the trend in the market using time intervals. The dataset includes records from more than one sector; hence, the dataset gives an overview of diversified performance through stocks in the market. Due to its granularity on a daily basis, the dataset will contain thousands of records and hence is appropriate to be used in time-series analysis, trend visualizations, and predictive modeling. The dataset could be required to undergo some preprocessing like handling null values and duplicates and formatting the date fields into proper datetime types prior to performing an analysis. The structured and comprehensive dataset offers a powerful foundation to deploy stock price predictive models and experiment with visualization strategies to understand past trading behavior.

In [10]:
```
#Importing the required libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import pandas as pd
from scipy.stats import zscore
import warnings
import datetime
warnings.filterwarnings("ignore", category=FutureWarning)
import calendar
import os
from mpl_toolkits.mplot3d import Axes3D
```

In [11]:
```
#Setting our files folder
folder_path =
```

```
'/Users/hishikeshphukan/Library/CloudStorage/OneDrive-
RMITUniversity/Machine Learning/Assignment/Datasets/archive'
```

In [12]:
```
#Selecting all the csv files
csv_files = [f for f in os.listdir(folder_path) if
f.endswith('.csv')]
```

In [13]:
```
#Intilializing an empty list to store the Dataframes
dfs =[]

#Looping through each csv files
for file in csv_files:
    file_path = os.path.join(folder_path, file)
    df = pd.read_csv(file_path)  # Reading the CSV file into a
DataFrame
    df['Company'] = os.path.splitext(file)[0] # Adding a new
column to the data frame containg the name of the company
    dfs.append(df)  # Appending the DataFrame to the list

#Concatenate all data Frames into one
combined_df = pd.concat(dfs, ignore_index = True)

#Show the combined DataFrame
print(combined_df)
```

```
             Date     Open     High      Low    Close  Adj Close
Volume  \
0       2002-11-27  1.88150  1.88150  1.88150  1.88150   1.122931
0.0
1       2002-11-28  1.64632  1.64632  1.58046  1.59928   0.954495
914630.0
2       2002-11-29  1.62750  1.64632  1.62750  1.64632   0.982570
355407.0
3       2002-12-02  1.64632  1.64632  1.63691  1.64632   0.982570
329088.0
4       2002-12-03  1.64632  1.64632  1.62750  1.62750   0.971337
278797.0
...            ...      ...      ...      ...      ...        ...
...
432883  2020-03-26  7.00000  7.00000  6.10000  6.35000   6.350000  3
919333.0
432884  2020-03-27  6.51000  6.85000  6.36000  6.36000   6.360000  6
649497.0
432885  2020-03-30  6.35000  6.58000  6.28000  6.58000   6.580000  3
138798.0
432886  2020-03-31  6.95000  7.18000  6.67000  6.86000   6.860000  4
507417.0
432887  2020-04-01  7.20000  7.48000  6.96000  7.20000   7.200000  3
444163.0

       Company
0          WOR
1          WOR
2          WOR
3          WOR
4          WOR
...        ...
432883     CHC
432884     CHC
432885     CHC
432886     CHC
432887     CHC

[432888 rows x 8 columns]
```

In [14]:
```
market = combined_df
```

In [15]:
```
market
```

Out[15]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2002-11-27	1.88150	1.88150	1.88150	1.88150	1.122931	0.0
1	2002-11-28	1.64632	1.64632	1.58046	1.59928	0.954495	914630.0
2	2002-11-29	1.62750	1.64632	1.62750	1.64632	0.982570	355407.0
3	2002-12-02	1.64632	1.64632	1.63691	1.64632	0.982570	329088.0
4	2002-12-03	1.64632	1.64632	1.62750	1.62750	0.971337	278797.0
...	...	...	...	...	...	...	...
432883	2020-03-26	7.00000	7.00000	6.10000	6.35000	6.350000	3919333.0
432884	2020-03-27	6.51000	6.85000	6.36000	6.36000	6.360000	6649497.0
432885	2020-03-30	6.35000	6.58000	6.28000	6.58000	6.580000	3138798.0
432886	2020-03-31	6.95000	7.18000	6.67000	6.86000	6.860000	4507417.0
432887	2020-04-01	7.20000	7.48000	6.96000	7.20000	7.200000	3444163.0

432888 rows × 8 columns

Dataset feature

The following tabulated form are the faeture of the data set which we collected from the kaggle website

In [18]:

```
from tabulate import tabulate

table = [['Name','Data Type','Units','Description'],
         ['Date', 'Date', 'dd/mm/yyyy', 'Market Opening Day'],
        ['Open', 'Numerical', 'AUD' 'Market opening price'],
         ['High', 'Numerical', 'AUD', 'Market highset price for
that day'],
         ['Low', 'Numerical', 'AUD', 'Market lowest price for that
day'],
         ['Close', 'Numerical', 'AUD', 'Market closing price'],
         ['Adj Close', 'Numerical', 'AUD', 'Adjusted Closing Price
including dividends'],
```

```
        ['Volume', 'Numerical', 'NA', 'Trading Volumes'],
        ['Compnay', 'Categorical', 'NA', 'Name of the company']]

print(tabulate(table, headers='firstrow', tablefmt='fancy_grid'))
```

```
╒═══════════╤═══════════════╤═══════════════════════════════╤═══════════════════════════════╕
│ Name      │ Data Type     │ Units                         │ Description                   │
╞═══════════╪═══════════════╪═══════════════════════════════╪═══════════════════════════════╡
│ Date      │ Date          │ dd/mm/yyyy                    │ Market Opening Day            │
├───────────┼───────────────┼───────────────────────────────┼───────────────────────────────┤
│ Open      │ Numerical     │ AUDMarket opening price       │                               │
├───────────┼───────────────┼───────────────────────────────┼───────────────────────────────┤
│ High      │ Numerical     │ AUD                           │ Market highset price for that day │
├───────────┼───────────────┼───────────────────────────────┼───────────────────────────────┤
│ Low       │ Numerical     │ AUD                           │ Market lowest price for that day │
├───────────┼───────────────┼───────────────────────────────┼───────────────────────────────┤
│ Close     │ Numerical     │ AUD                           │ Market closing price          │
├───────────┼───────────────┼───────────────────────────────┼───────────────────────────────┤
│ Adj Close │ Numerical     │ AUD                           │ Adjusted Closing Price including dividends │
├───────────┼───────────────┼───────────────────────────────┼───────────────────────────────┤
│ Volume    │ Numerical     │ NA                            │ Trading Volumes               │
├───────────┼───────────────┼───────────────────────────────┼───────────────────────────────┤
│ Compnay   │ Categorical   │ NA                            │ Name of the company           │
╘═══════════╧═══════════════╧═══════════════════════════════╧═══════════════════════════════╛
```

The dataset includes five years' worth of stock price history on the leading 100 companies (2015–2020). Every row is one trading day's data on one company. Following is the detailed description of each column in the dataset by the name of the column, type of data, units of measurement (if any), and an informal description of its significance.

1. Date
    - Data Type: `datetime64`
    - Unit: Not Applicable
    - Description: This column denotes the date when the trading activity

occurred. It is important to arrange the data set in chronological order and to carry out time-series operations. Having this column in the correct datetime form enables features like time-based indexation, filtering, resampling (weekly/monthly mean), and visualizing several trendlines through time.

2. Company name / Stock symbol
    - Data Type: `string` or `object`
    - Units: Not applicable
    - Description: This column includes the name or symbol of the firm whose share data is being accounted for. It is used to identify each firm in the data set uniquely. Without this column, we could not group by company and examine the performance on a firm-by-firm basis, particularly when comparing trends or performance among different firms or industry sectors.
3. Open Price
    - Data Type: `float64`
    - Units: Australian Dollars
    - Description: Opening price describes the price on which the stock initially traded when the market opened on this specific day. It is an important indicator and reflects the investor mood at the very beginning of the trading session and can be used to compare with the closing price to see if the stock appreciated or depreciated throughout the day. It is frequently applied in day trading and short-term predictions.
4. Closing price
    - Data Type: `float64`
    - Units: Australian Dollars
    - Description: The closing price is the last trade price registered on a given stock when the trading day closes. It is one of the most popular tools used in stock analysis and is usually regarded as better than the opening price because it is stable and reflects the final agreement among buyers and sellers. It is also applied to compute several financial metrics such as daily returns, moving averages, and technical patterns.
5. Adjusted Close Price
    - Data Type: `float64`
    - Units: Australian Dollars
    - Description: Adjusted close price takes into account corporate action like stock splits, dividends, and rights offerings. It more accurately reflects the value of a stock over time and is applied in long-term investment evaluation and calculation of total return. Not all datasets provide this column, but it is incredibly helpful with time-series modeling.

6. Volume
    - Date Type: `int64`
    - Units: Number of shares exchanged
    - Description: This figure reflects the quantity of shares that were sold and purchased on any trading day for any given stock. It is an indicator of market liquidity and activity. It usually accompanies high price volatility and important news or events with high trading volume. Volume is extensively applied in technical analysis to validate trends or reversals.
7. High
    - Data Type: `float64`
    - Unit: Australian Dollars
    - Description: The highest price hit during the trading period is depicted by this column. It is very important when it comes to identifying resistance levels, analyzing volatility and patterns in price movement. It assists technical analysis indicators such as moving averages and Bollinger Bands.
8. Low
    - Data Type: `float64`
    - Unit: Australian Dollars
    - Description: This column indicates the lowest price available during the period of trading. It is necessary to identify support levels, interpret market patterns, and measure risk levels. It is usually applied together with the High and the Close price to compute volatility and study patterns with candles.

Target Feature

The **Adjusted Closing Price (AUD)** is a more advanced and insightful measure of the value of a stock over the period. In comparison to the raw closing price, it takes into account corporate-related developments like dividends, stock splits, and rights issues to better reflect past price action. The adjustment will benefit investors and analysts alike by removing artificial price fluctuations due to these events so that an accurate trend and forecast can be performed. Based on adjusted closing price utilization, we are able to create sound trading strategies, find long-term cycles, and develop more confident predictive modeling. The benefit of this function is that it enables past price history to make sense so that we are able to better observe and assess stock performance on an ongoing basis. In the highly volatile financial world where accuracy is paramount, using the adjusted closing price in Australian Dollars allows us to make informed and data-supported investment choices and to neutralize the noise in market abnormalities.

Goals and Objectives

Goals

The main objective within this project is to create an advanced predictive model that effectively anticipates the Adjusted Closing Price (AUD) of a stock using past market data. The Adjusted Closing Price offers an advanced version of the closing price with corporate actions like dividends, stock splits, and issuance of new stock incorporated into its calculation. The use of these corporate actions ensures more accurate and consistent representation of stock value through time and hence makes it an imperative parameter among investors, traders, and financial analysts. Relying on advanced data science and machine learning methods, this project intends to advance market forecast capabilities, reduce risks in investments, and deliver actionable insights to decision-makers.

Objectives

To accurately forecast the **Adjusted Closing Price (AUD)**, this project will first undertake **data collection and preprocessing**. Historical stock market data will be collected and includes **Open, High, Low, Close, Volume, and Adjusted Closing Price**. Care will be given when handling missing values, outliers, and inconsistencies to ensure data quality. Further to enhance model performance, columns will be converted to suitable data types, specifically **datetime** to cater to time-series analysis. Numerical features will be scaled or normalized as necessary.

The following task is **Exploratory Data Analysis (EDA)** to find important trends and relationships in the dataset. Movements in the stock price will be graphed using trend lines, moving averages, and time-series plots. Correlations among distinct market variables will be analyzed to identify the effect on the adjusted closing price. EDA will reveal patterns and insights to inform feature engineering and model selection.

Feature engineering will be important to make the model more predictive. Additional features like **moving averages, volatility indicators, and the Relative Strength Index (RSI)** will be created. Lag features will be formed to model the price movements with dependency on the time factor. Macroeconomic indicators like inflation rates, interest rates, and market sentiment will be added if available to make the model more robust.

For **training and model selection**, several machine learning algorithms will be experimented with, ranging from **Linear Regression to Decision Trees, Random Forest to XGBoost, and Long Short-Term Memory Networks**

(LSTMs) to perform forecasting on time-series data. Historical data will be used to train the models and hyperparameters tuned to optimize predictive performance. **Cross-validation** methods will be used to avoid the problem of overfitting and to ensure the model performs well on unseen data.

After the models are trained, **their performance will be assessed** through metrics like **Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R^2)**. The model with the best performance will be chosen on the basis of its capacity to reduce error and offer credible predictions. In order to make it applicable in real life, the model will be tried out on unseen data. To make the forecasts available and beneficial, the project will provide **deployment and visualisation**. A user-friendly interface or dashboard will be created to show forecasted stock prices and trends. The dashboard will provide real-time updating of data to enable investors to make informed choices based on the most current predictions.

Lastly, **future enhancements and risk assessment** will be addressed. Stock prices are influenced by market volatility, economic fluctuations, and unexpected events, so the model limitations will be examined. In an effort to make the predictions more accurate, more advanced deep learning methods like **transformers** and **reinforcement learning** will be considered. The model will be constantly updated and improved with evolving data so that it stays current and credible under the existing and unpredictable market conditions.

Data Cleaning and Preprocessing

In this section we have discuss the prelimanry steps of cleaning the data and preprocessing udertaken for this project.

Data Cleaning Steps

- Dropping Simillar Columns
- Relabelling Columns
- Datatype conversion
- Mutating new columns
- Finding and replacing null variables
- Finding and replacing outliers
- Random Sampling

At first lets look into the columns

In [30]:
```
print(market.columns)
```

```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volum
e',
       'Company'],
      dtype='object')
```

Dropping or Creating Columns

We are not using the time serises model for our prediction. We will extract the month and year from there. As stock follow monthly, quarterly, or yearly patterns. In additon for any time-based analysis we can see how price vary over time.

In [33]:
```
market['Date'] = pd.to_datetime(market['Date']) # Converting the
date column to date type from string
market['Month'] = market['Date'].dt.month # Extracting the month
from the date and making a new column
market['Year'] = market['Date'].dt.year # Extracting the year from
the date and making a new column
market.drop(columns=['Date'], inplace=True) # Dropping the date
column
```

In [34]:
```
market
```

Out[34]:

	Open	High	Low	Close	Adj Close	Volume	Compar
0	1.88150	1.88150	1.88150	1.88150	1.122931	0.0	WC
1	1.64632	1.64632	1.58046	1.59928	0.954495	914630.0	WC
2	1.62750	1.64632	1.62750	1.64632	0.982570	355407.0	WC
3	1.64632	1.64632	1.63691	1.64632	0.982570	329088.0	WC
4	1.64632	1.64632	1.62750	1.62750	0.971337	278797.0	WC
...	...	...	...	...	...	...	
432883	7.00000	7.00000	6.10000	6.35000	6.350000	3919333.0	CH
432884	6.51000	6.85000	6.36000	6.36000	6.360000	6649497.0	CH
432885	6.35000	6.58000	6.28000	6.58000	6.580000	3138798.0	CH
432886	6.95000	7.18000	6.67000	6.86000	6.860000	4507417.0	CH
432887	7.20000	7.48000	6.96000	7.20000	7.200000	3444163.0	CH

432888 rows × 9 columns

Relabelling the columns

Some of the columns are not labelled properly. Thus, we will label the columns with all lower cases and will remove the white space

In [37]:
```
#Stripping the white spaces at the begining and ending of the
column names
market.columns = market.columns.str.strip()

# Mapping the column name and assigning the new name
columns_mappings = {
    'Open' : 'open',
    'High' : 'high',
    'Low' : 'low',
    'Close' : 'close',
    'Adj Close' : 'adj_close',
    'Volume' : 'volume',
    'Company' : 'company',
    'Month' : 'month',
    'Year' : 'year'
}

# Renaming the columns
market = market.rename(columns = columns_mappings)
market.sample(10, random_state =1000)
```

Out[37]:

	open	high	low	close	adj_close	volume
383208	0.511100	0.511100	0.51110	0.51110	0.511100	0.0
40264	6.386140	6.404260	6.36803	6.38614	3.809678	4585322.0
45531	1.704220	1.704220	1.62542	1.62542	1.368064	18343.0
358737	3.056510	3.132920	3.05651	3.13292	1.409125	767412.0
98968	26.450001	26.629999	26.27000	26.40000	19.610800	2161826.0
244791	5.148570	5.177440	5.07158	5.09083	3.239263	3266665.0
155431	1.255770	1.270610	1.24589	1.26566	0.705400	5548982.0
413634	4.443400	4.448110	4.34906	4.35849	3.446635	2708101.0
234731	5.850000	5.920000	5.80500	5.84000	4.784865	980150.0
293353	1.782000	1.790000	1.74000	1.75000	1.021783	184211.0

Data type conversion

Lets check for the data types of the columns

In [40]:
```
#Checking the data types of the columns
market.dtypes
```

Out[40]:
```
open         float64
high         float64
low          float64
close        float64
adj_close    float64
volume       float64
company       object
month          int32
year           int32
dtype: object
```

Here, we need to change the month and year to object.

In [42]:
```
#Changing the month to object
month_str = {1 : "January",
             2 : "Feburary",
             3 : "March",
             4 : "April",
             5 : "May",
             6 : "June",
             7 : "July",
             8 : "August",
             9 : "September",
             10 : "October",
             11 : "November",
             12 : "December"
            }

market['month'] = market['month'].map(month_str)
```

In [43]:
```
market
```

Out[43]:

	open	high	low	close	adj_close	volume	compar
0	1.88150	1.88150	1.88150	1.88150	1.122931	0.0	WC
1	1.64632	1.64632	1.58046	1.59928	0.954495	914630.0	WC
2	1.62750	1.64632	1.62750	1.64632	0.982570	355407.0	WC
3	1.64632	1.64632	1.63691	1.64632	0.982570	329088.0	WC
4	1.64632	1.64632	1.62750	1.62750	0.971337	278797.0	WC
...	...	...	...	...	...	...	
432883	7.00000	7.00000	6.10000	6.35000	6.350000	3919333.0	CH
432884	6.51000	6.85000	6.36000	6.36000	6.360000	6649497.0	CH
432885	6.35000	6.58000	6.28000	6.58000	6.580000	3138798.0	CH
432886	6.95000	7.18000	6.67000	6.86000	6.860000	4507417.0	CH
432887	7.20000	7.48000	6.96000	7.20000	7.200000	3444163.0	CH

432888 rows × 9 columns

In [44]:
```
#Changing the Year into object/str
market['year'] = market['year'].astype(str)
```

In [45]:
```
market.dtypes
```

Out[45]:
```
open          float64
high          float64
low           float64
close         float64
adj_close     float64
volume        float64
company        object
month          object
year           object
dtype: object
```

Lets examine the unique values in the object columns or categorical columns to verify our data and chenges if we need

In [47]:
```
from IPython.display import display, HTML
display(HTML('<b>Table 1: Summary of categorical features</b>'))
market.describe(include='object').T
```

Table 1: Summary of categorical features

Out[47]:

	count	unique	top	freq
company	432888	100	AMP	5160
month	432888	12	March	38175
year	432888	21	2019	25300

Lets check all the unique values of the categorical values

In [49]:
```
# To see all unique values for categorical data types
categoricalColumns =
market.columns[market.dtypes==object].tolist()

for col in categoricalColumns:
    print('Unique values for ' + col)
    print(market[col].unique())
    print('')
```

```
Unique values for company
['WOR' 'OSH' 'QBE' 'DXS' 'FMG' 'FLT' 'SUN' 'WBC' 'BXB' 'REA' 'SVW' '
DMP'
 'BLD' 'RHC' 'QAN' 'CGF' 'CIM' 'VEA' 'WOW' 'A2M' 'AST' 'SCG' 'ORI' '
MQG'
 'STO' 'GOZ' 'AMC' 'AZJ' 'COH' 'CWY' 'CWN' 'ALQ' 'ANN' 'WES' 'EVN' '
SKI'
 'S32' 'ANZ' 'GMG' 'CCL' 'TAH' 'CAR' 'FPH' 'TWE' 'ALU' 'COL' 'AMP' '
BEN'
 'NST' 'ALX' 'BPT' 'WPL' 'BHP' 'VCX' 'SHL' 'SEK' 'JHX' 'TCL' 'ALL' '
SPK'
 'BSL' 'CBA' 'SGP' 'AWC' 'IAG' 'RMD' 'ILU' 'CTX' 'TLS' 'SGR' 'NCM' '
APT'
 'ASX' 'SYD' 'CSL' 'TPM' 'LLC' 'CPU' 'ORG' 'MGR' 'APA' 'MPL' 'FBU' '
GPT'
 'IPL' 'AGL' 'QUB' 'XRO' 'SAR' 'NAB' 'IEL' 'RIO' 'WTC' 'BOQ' 'JBH' '
DOW'
 'HVN' 'MFG' 'SOL' 'CHC']

Unique values for month
['November' 'December' 'January' 'Feburary' 'March' 'April' 'May' 'J
une'
 'July' 'August' 'September' 'October']

Unique values for year
['2002' '2003' '2004' '2005' '2006' '2007' '2008' '2009' '2010' '201
1'
 '2012' '2013' '2014' '2015' '2016' '2017' '2018' '2019' '2020' '200
0'
 '2001']
```

Finding and Dealing with NULL Values

Lets look for any null values in our data frame

In [52]:
```
market.isna().sum()
```

Out[52]:
```
open          1527
high          1527
low           1527
close         1527
adj_close     1527
volume        1527
company          0
month            0
year             0
dtype: int64
```

Here we have found missing values in multiple columns with high number of missing values. To deal with this NA values we can fill the mean of the price for each company

In [54]:
```
market['open'] = market.groupby('company')
['open'].transform(lambda x: x.fillna(x.mean()))
market['high'] = market.groupby('company')
['high'].transform(lambda x: x.fillna(x.mean()))
market['low'] = market.groupby('company')['low'].transform(lambda
x: x.fillna(x.mean()))
market['close'] = market.groupby('company')
['close'].transform(lambda x: x.fillna(x.mean()))
market['adj_close'] = market.groupby('company')
['adj_close'].transform(lambda x: x.fillna(x.mean()))
market['volume'] = market.groupby('company')
['volume'].transform(lambda x: x.fillna(x.mean()))
```

In [55]:
```
market.isna().sum()
```

Out[55]:
```
open          0
high          0
low           0
close         0
adj_close     0
volume        0
company       0
month         0
year          0
dtype: int64
```

Finding duplicates and dealing with duplicates

Lets examine for any duplicate entries in our data

In [58]:
```
market[market.duplicated()]
```

Out[58]:

	open	high	low	close	adj_close	volum
21	16.219582	16.443239	15.987237	16.214716	12.621522	1.167133e+0
43	16.219582	16.443239	15.987237	16.214716	12.621522	1.167133e+0
64	1.674540	1.674540	1.674540	1.674540	0.999412	0.000000e+0
103	16.219582	16.443239	15.987237	16.214716	12.621522	1.167133e+0
107	16.219582	16.443239	15.987237	16.214716	12.621522	1.167133e+0
...	...	...	...	...	...	
429379	4.174460	4.174460	4.174460	4.174460	2.236731	0.000000e+0
429478	6.023420	6.023420	5.947170	5.947170	3.186572	1.326250e+0
430040	1.124630	1.124630	1.010260	1.029320	0.590181	1.429400e+0
430043	1.124630	1.124630	1.067440	1.067440	0.612038	3.317700e+0
430326	4.954785	5.019156	4.886877	4.955436	3.712646	9.144845e+0

4270 rows × 9 columns

As we can see sone duplicate entries in our data lets get rid of it.

In [60]:
```
market = market.drop_duplicates()
```

Now we have dropped the duplicate entries.

In [62]:
```
market[market.duplicated()]
```

Out[62]:

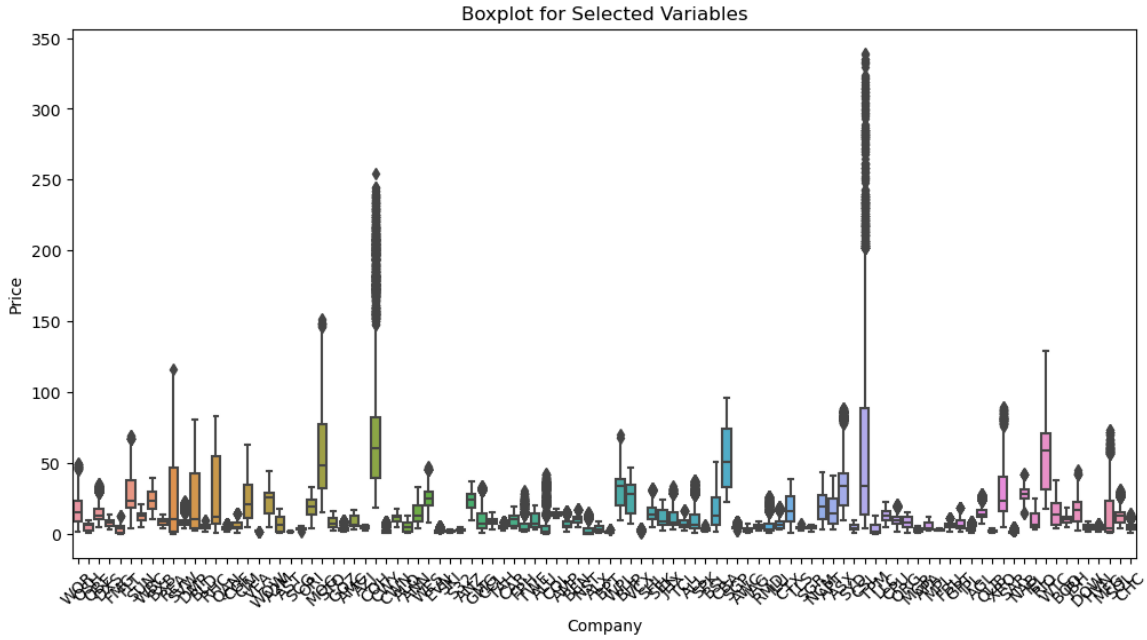
	open	high	low	close	adj_close	volume	company	month	year

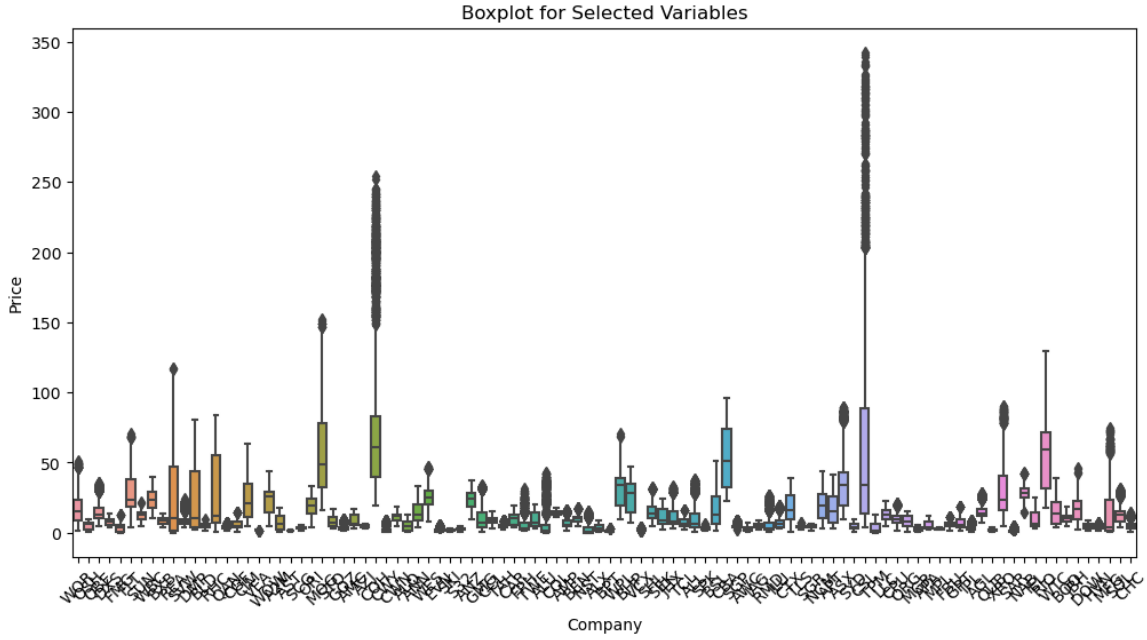
Finding and Dealing with Outliers

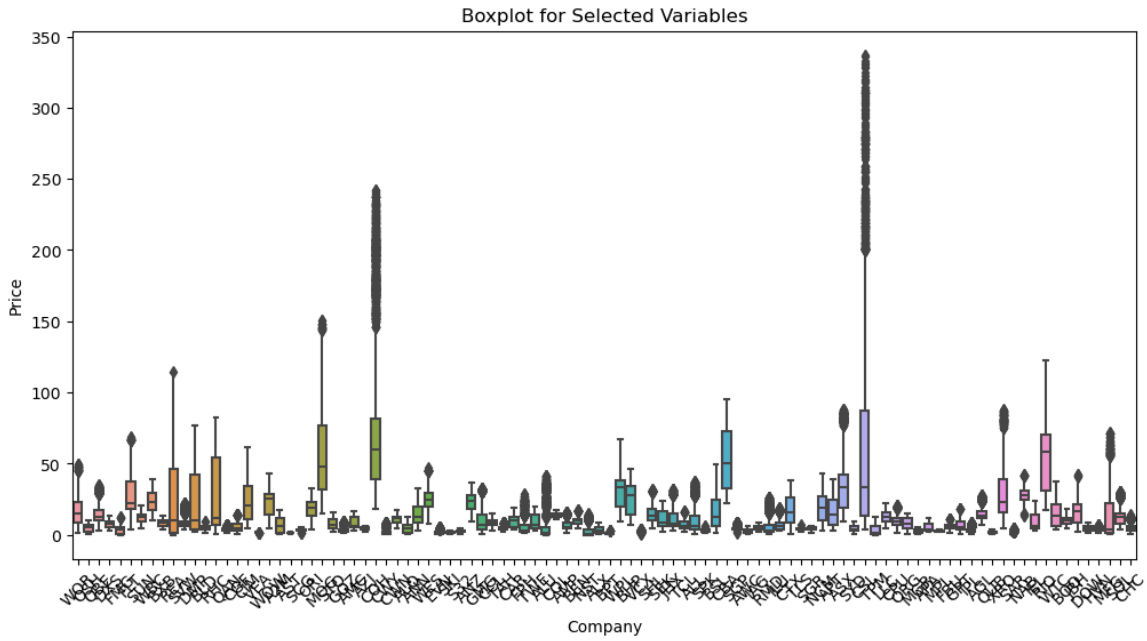
Lets examine for the outliers if there are any outliers we need to handle it

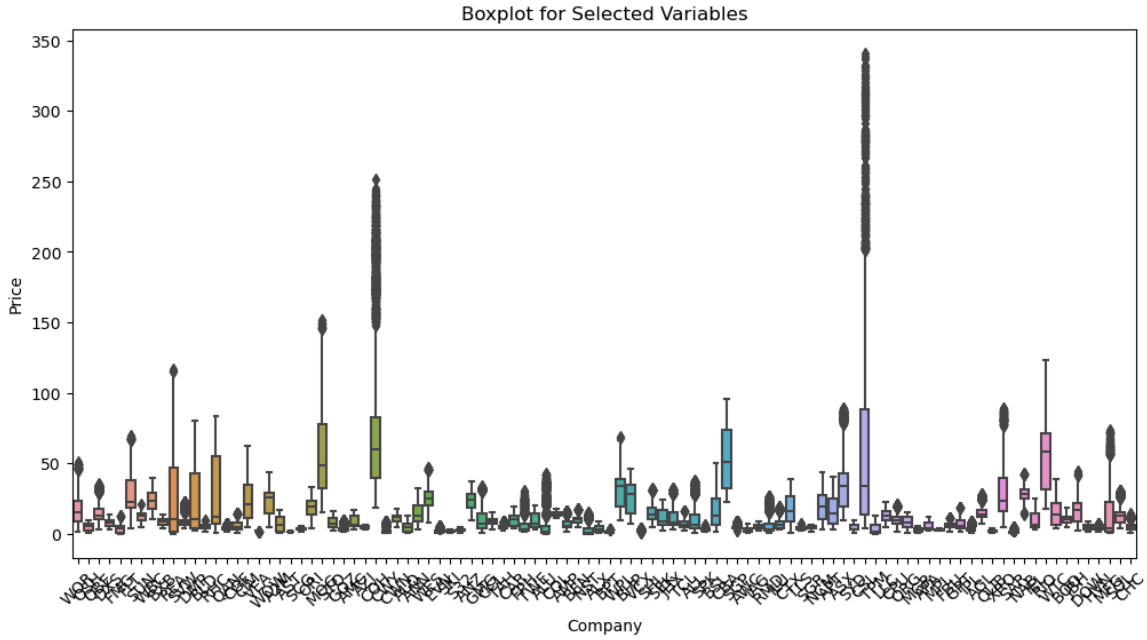
In [65]:
```
variables = ['open', 'high', 'low', 'close', 'adj_close',
'volume']
```

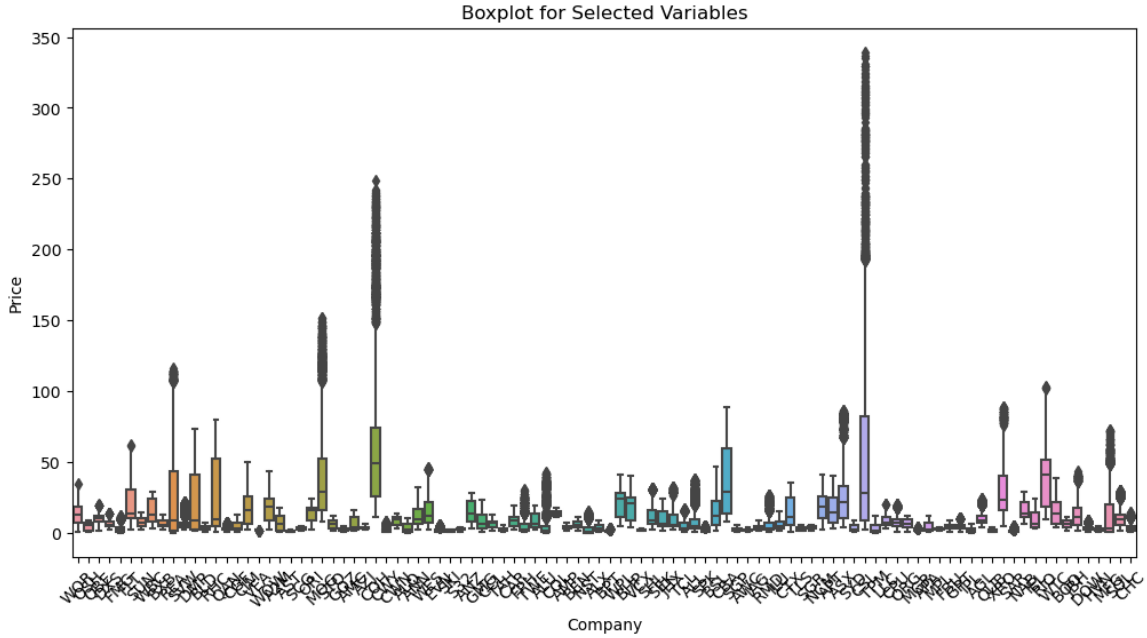
In [66]:
```
for var in variables:
    plt.figure(figsize=(12, 6))
    sns.boxplot(x = 'company', y = var, data = market)
    plt.xticks(rotation=45)
    plt.title("Boxplot for Selected Variables")
    plt.xlabel("Company")
    plt.ylabel("Price")
    plt.show()
```

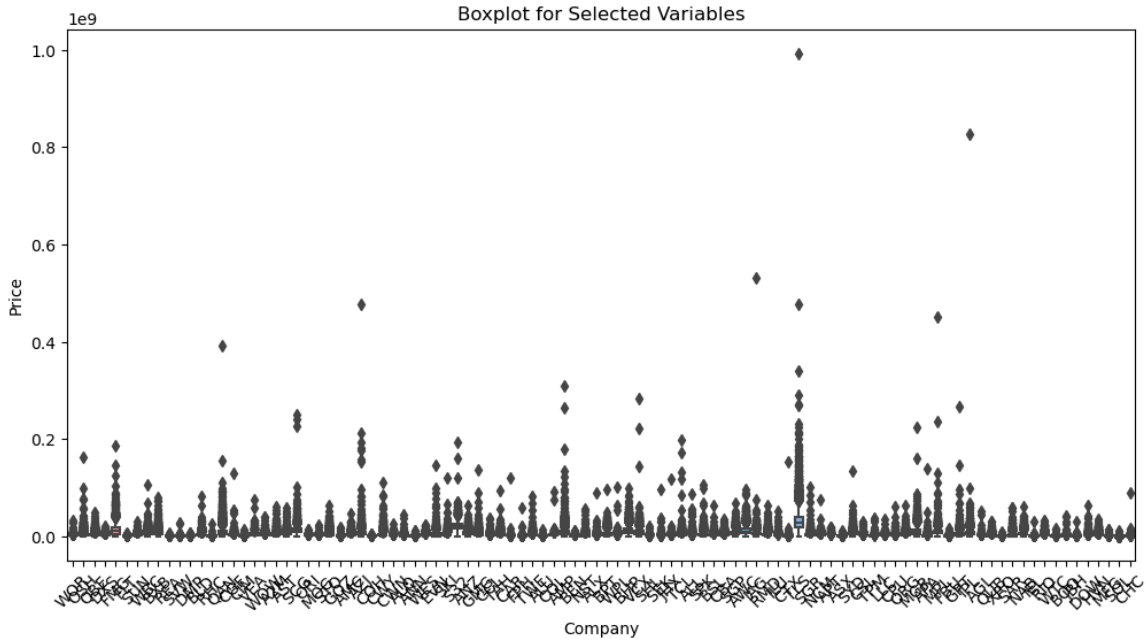

Boxplot for Selected Variables
Price
Company


Boxplot for Selected Variables
Price
Company


Boxplot for Selected Variables
Price
Company



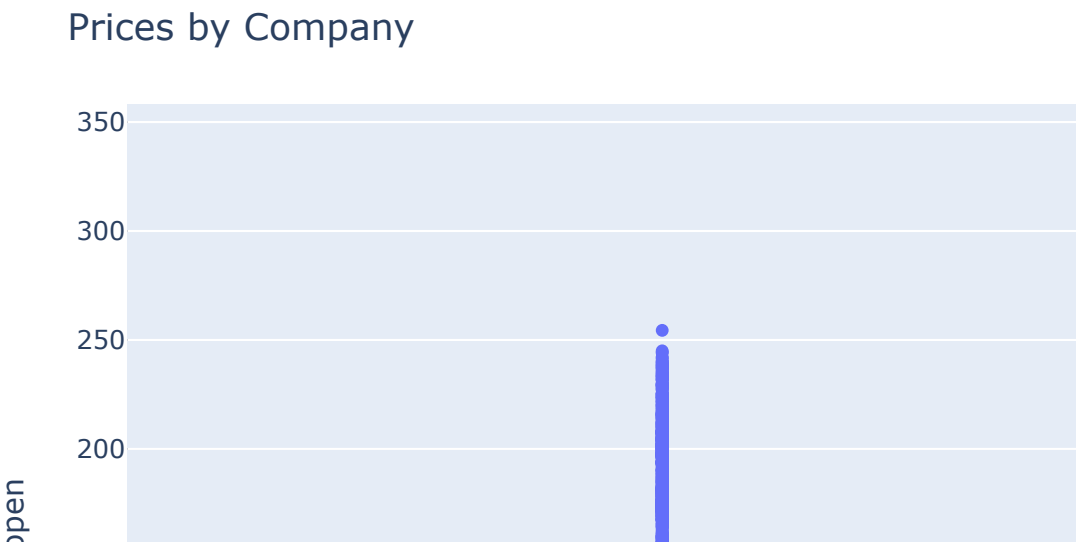


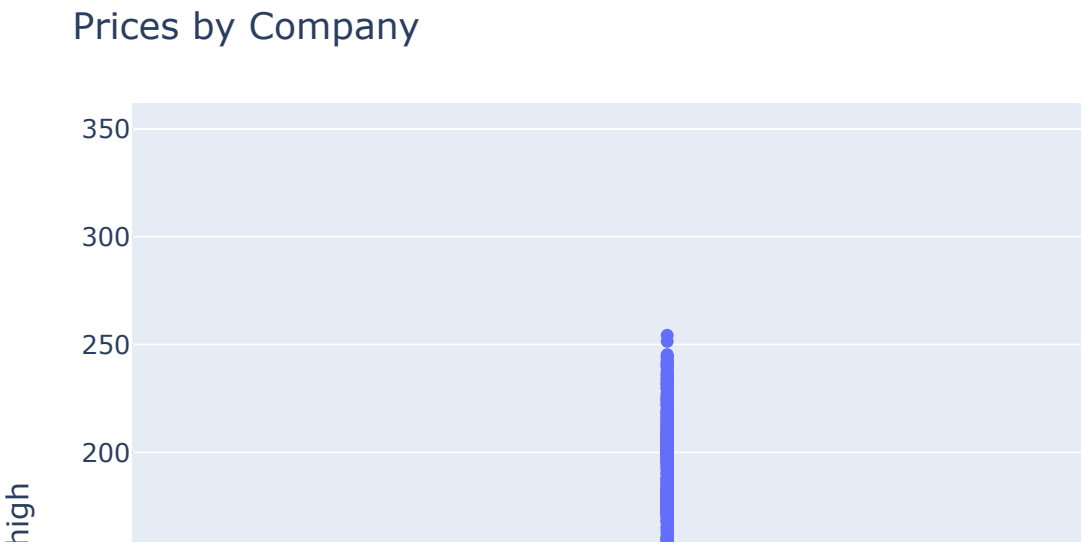


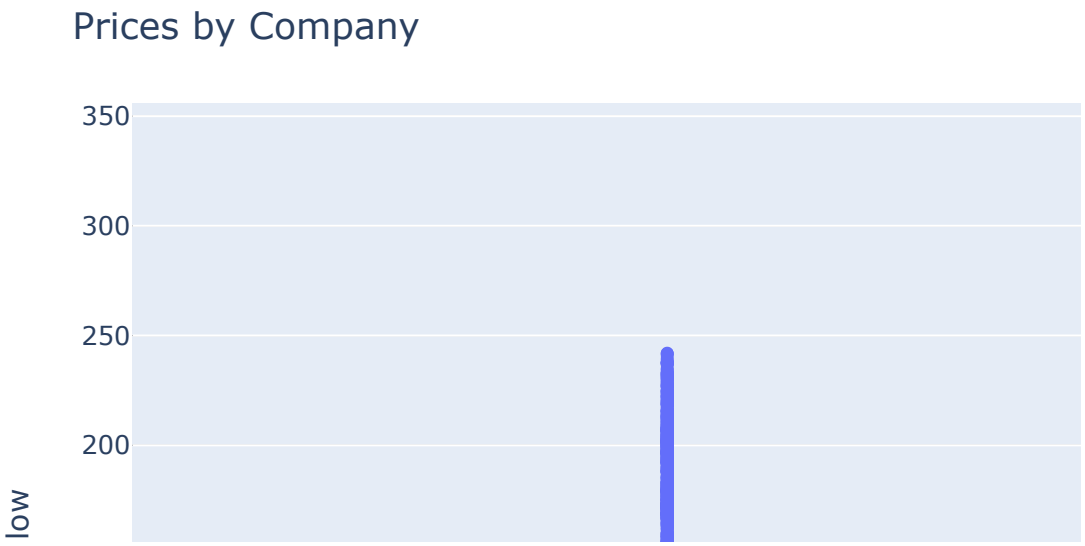
From this its hard to read our graph lets create an iterative boxplot for the

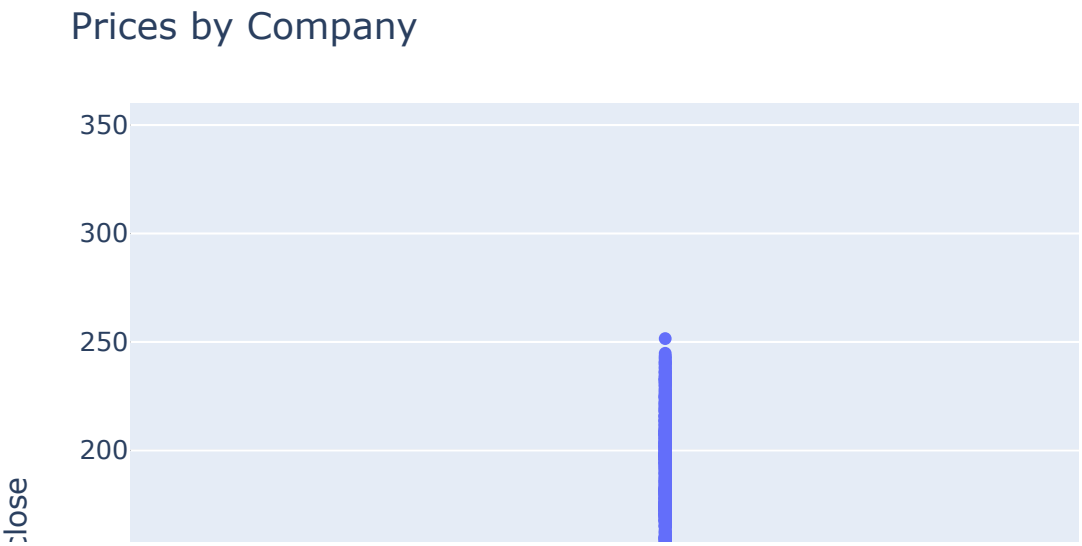
variables

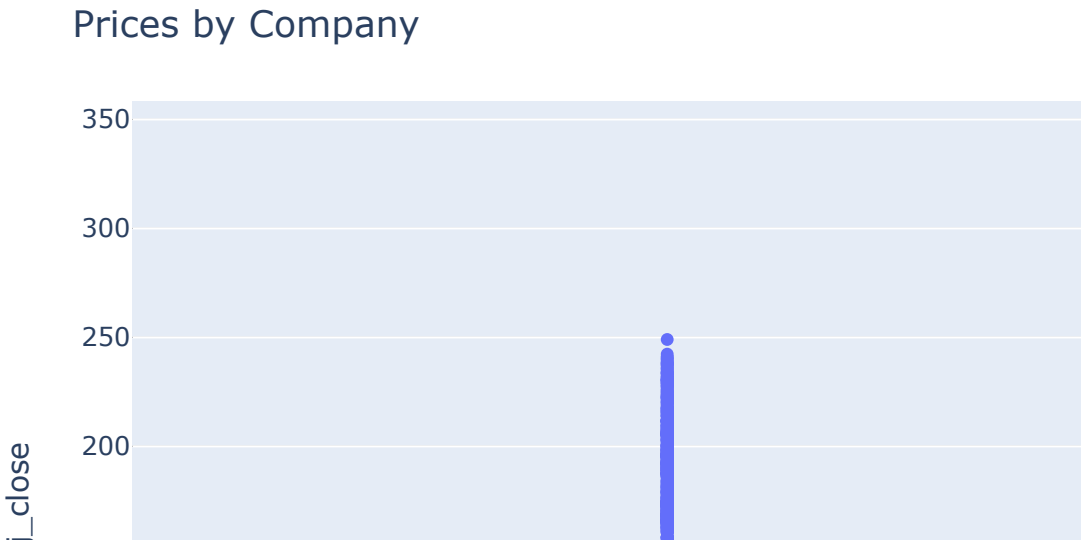
In [68]:
```
import plotly.express as px

for var in variables:
    fig = px.box(market, x = 'company', y = var)
    fig.update_layout(title = 'Prices by Company')
    fig.show()
```

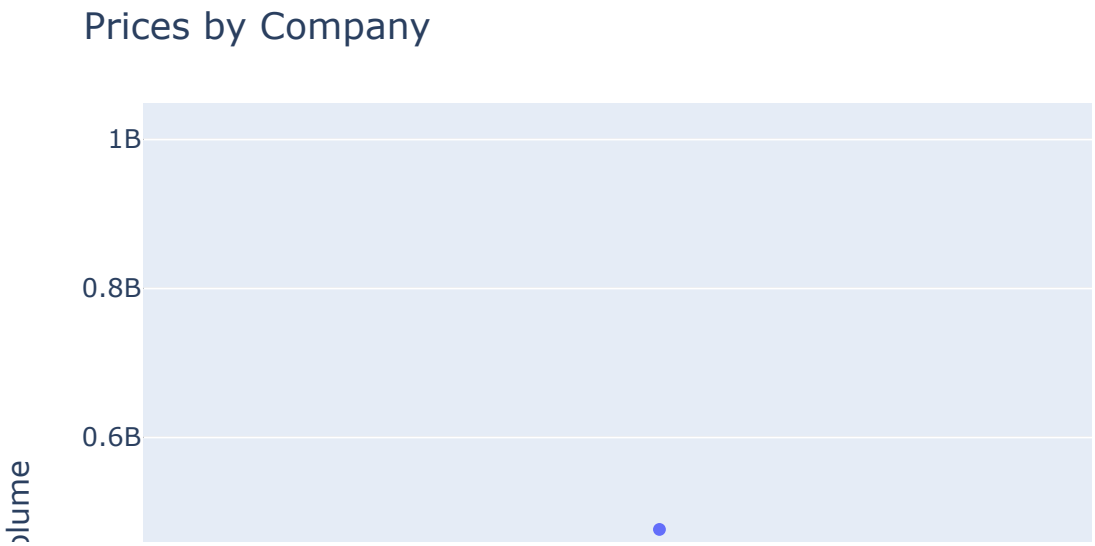



Prices by Company
350
300
250
200
high


Prices by Company
350
300
250
200
low


Prices by Company
350
300
250
200
lose


Prices by Company
350
300
250
200
i_close



From the above we can see there are outliers to deal with the outliers we will do the transformation of the data. We will transforms our columns into lograthmic transformation.

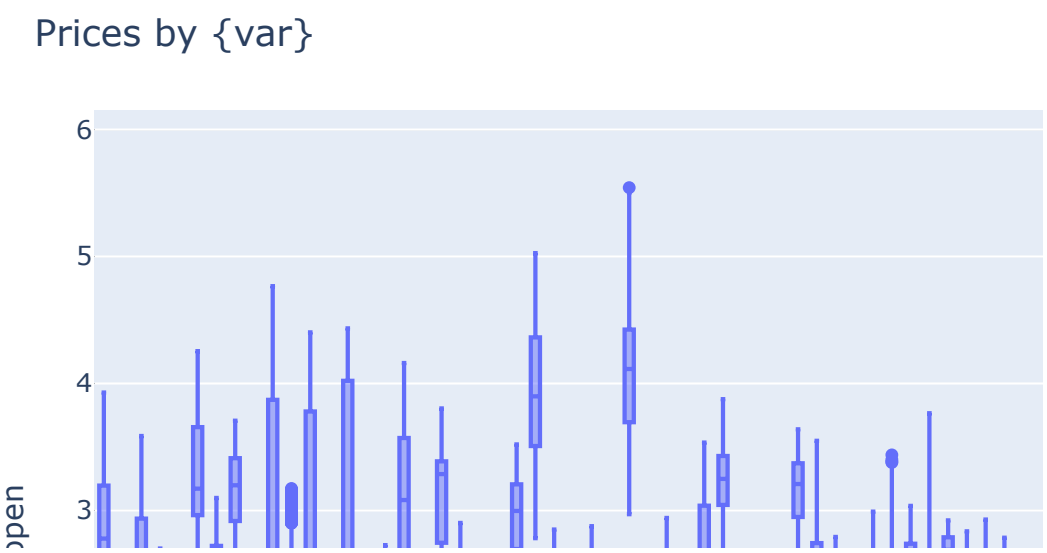
In [70]:
```
import numpy as np

for var in variables:
    market[var] = market.groupby('company')[var].transform(lambda
x: np.log1p(x))
```

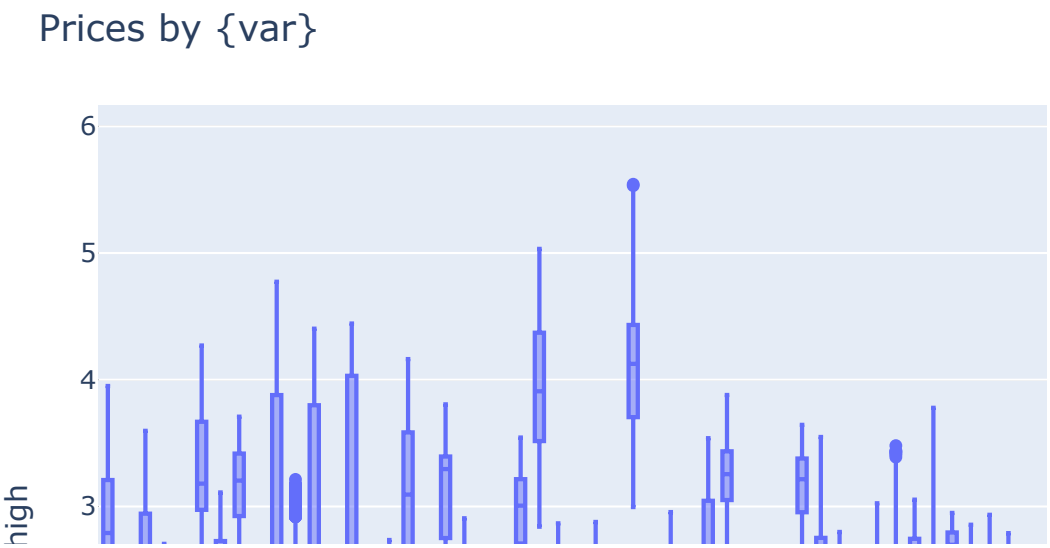
```
/var/folders/2m/rsqhpvzs6bd3f66nsg5d2tb40000gn/T/ipykernel_16982/423
9373838.py:4: SettingWithCopyWarning:


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/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py

/var/folders/2m/rsqhpvzs6bd3f66nsg5d2tb40000gn/T/ipykernel_16982/423
9373838.py:4: SettingWithCopyWarning:
```

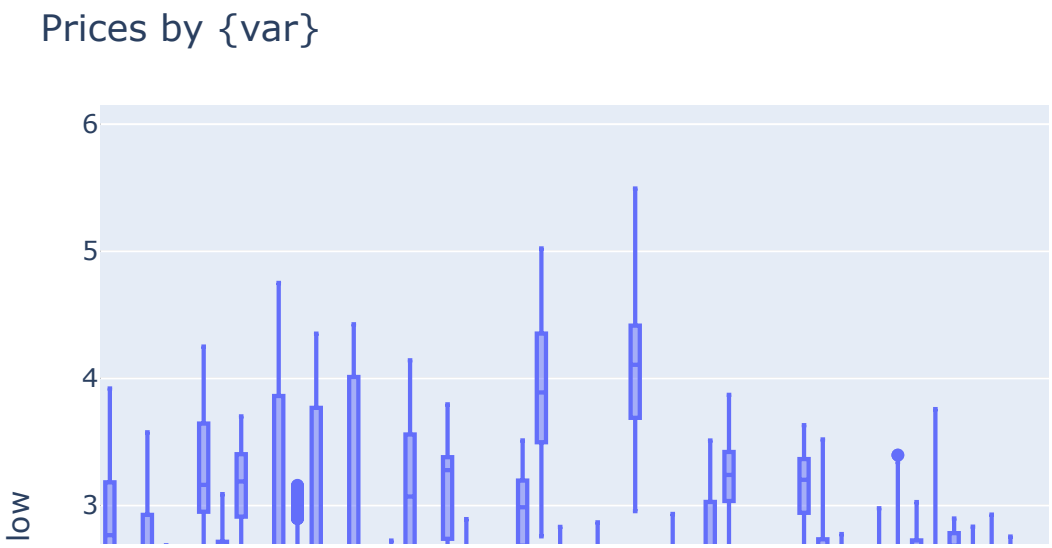
```
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py

/var/folders/2m/rsqhpvzs6bd3f66nsg5d2tb40000gn/T/ipykernel_16982/423
9373838.py:4: 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/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py

/var/folders/2m/rsqhpvzs6bd3f66nsg5d2tb40000gn/T/ipykernel_16982/423
9373838.py:4: 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/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py

/var/folders/2m/rsqhpvzs6bd3f66nsg5d2tb40000gn/T/ipykernel_16982/423
9373838.py:4: 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/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py

/var/folders/2m/rsqhpvzs6bd3f66nsg5d2tb40000gn/T/ipykernel_16982/423
9373838.py:4: 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/pand
as-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
py
```

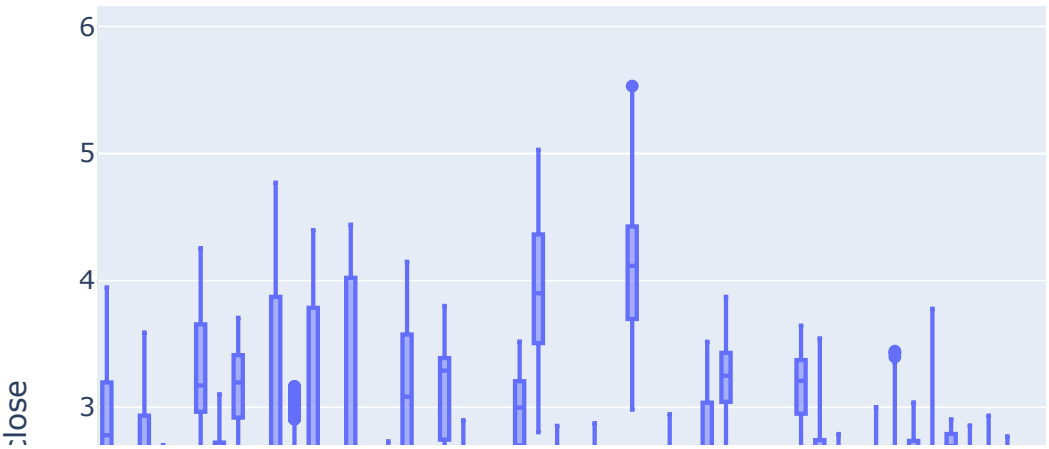
Lets visualize our data once again using boxplot

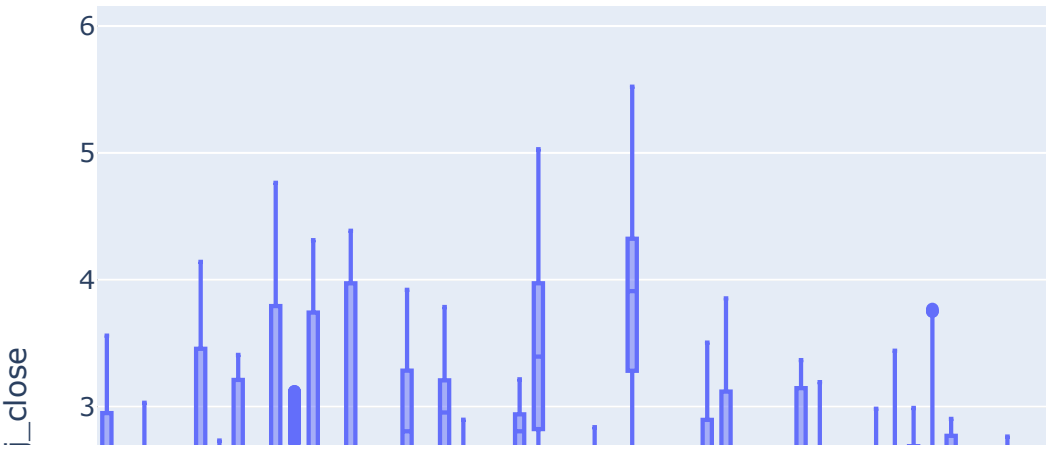
In [72]:
```
for var in variables:
    fig = px.box(market, x = 'company', y = var)
    fig.update_layout(title = 'Prices by {var}')
    fig.show()
```

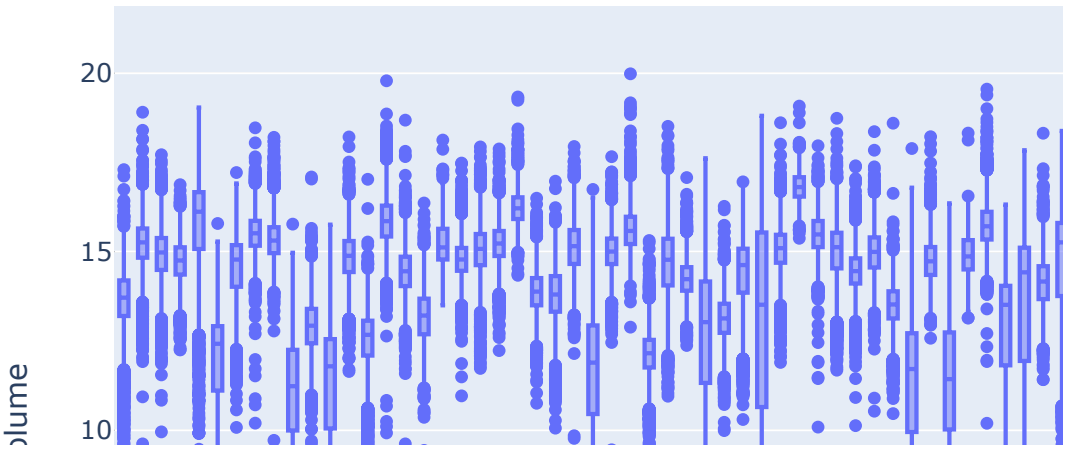



Prices by {var}
6
5
4
3
high


Prices by {var}
low
6
5
4
3


Prices by {var}
6
5
4
3


Prices by {var}
6
5
4
3
i_close



Lets create a descriptive statistics of the numerical variables for each comapny

In [74]:
```
grouped_stats = market.groupby('company').describe()
print(grouped_stats)
```

```
          open                                                          \
         count      mean       std       min       25%       50%
75%
company
A2M      1269.0  1.763899  0.826764  0.385262  1.015231  1.934416
2.564949
AGL      5143.0  2.707674  0.265816  2.101731  2.556324  2.696153
2.860385
ALL      5147.0  2.133525  0.685718  0.636577  1.613430  1.996060
2.656757
ALQ      5139.0  1.574064  0.638281  0.518806  0.926831  1.780966
2.132982
ALU      4729.0  1.261398  1.014557  0.078811  0.357674  0.879627
1.806648
...         ...       ...       ...       ...       ...       ...
...
WOR      4385.0  2.651490  0.685203  0.936971  2.290583  2.778179
```

```
3.195595
WOW      5148.0  3.091722  0.451242  1.732146  2.745481  3.286299
3.386760
WPL      5145.0  3.378239  0.446956  2.327872  3.023980  3.543276
3.683467
WTC      1007.0  2.581842  0.593608  1.483875  1.931521  2.687847
3.108614
XRO      1872.0  3.245025  0.596213  1.695616  2.834536  3.188417
3.709417

                    high           ... adj_close           volume
\
              max   count      mean  ...       75%       max   count
company                             ...
A2M      2.901422  1269.0  1.776289  ...  2.564180  2.897568  1269.0
AGL      3.378952  5143.0  2.715448  ...  2.540187  3.230077  5143.0
ALL      3.662792  5147.0  2.145428  ...  2.358652  3.655581  5147.0
ALQ      2.631867  5139.0  1.582983  ...  1.962802  2.392426  5139.0
ALU      3.765609  4729.0  1.270111  ...  1.653319  3.769450  4729.0
...           ...     ...       ...  ...       ...       ...     ...
WOR      3.928703  4385.0  2.664113  ...  2.949887  3.560184  4385.0
WOW      3.803324  5148.0  3.099326  ...  3.207941  3.785829  5148.0
WPL      4.269178  5145.0  3.386649  ...  3.355401  3.740033  5145.0
WTC      3.665355  1007.0  2.600034  ...  3.106003  3.672441  1007.0
XRO      4.504244  1872.0  3.257766  ...  3.711375  4.499810  1872.0


              mean       std  min        25%        50%        75%
max
company
A2M      15.000475  1.061623  0.0  14.621974  15.077057  15.493659
17.932876
AGL      14.009426  1.412156  0.0  13.776951  14.156892  14.489767
17.735879
ALL      14.175836  1.310066  0.0  13.813363  14.279590  14.695308
18.280646
ALQ      12.610216  2.114220  0.0  11.324219  13.025139  14.163663
17.614996
ALU      10.746235  3.136984  0.0  10.021315  11.429555  12.736880
16.352990
...            ...       ...  ...        ...        ...        ...
...
WOR      13.524428  1.307151  0.0  13.197447  13.710725  14.193944
17.297348
WOW      14.707503  1.250505  0.0  14.464158  14.780520  15.103383
17.481047
WPL      14.402136  1.286774  0.0  14.171572  14.493484  14.820545
18.421124
WTC      12.842697  1.220000  0.0  12.217429  12.999990  13.547082
16.510464
XRO      11.162262  1.476160  0.0  10.081550  10.869520  12.495376
15.932117

[100 rows x 48 columns]
```

Its hard to read the table lets flatten this

In [76]:
```
grouped_stats.columns = ['_'.join(col).strip() for col in
grouped_stats.columns.values]
grouped_stats.reset_index(inplace=True)
print(grouped_stats)
```

```
   company  open_count  open_mean  open_std  open_min  open_25%  ope
n_50%  \
0      A2M      1269.0   1.763899  0.826764  0.385262  1.015231  1.9
34416
1      AGL      5143.0   2.707674  0.265816  2.101731  2.556324  2.6
96153
2      ALL      5147.0   2.133525  0.685718  0.636577  1.613430  1.9
96060
3      ALQ      5139.0   1.574064  0.638281  0.518806  0.926831  1.7
80966
4      ALU      4729.0   1.261398  1.014557  0.078811  0.357674  0.8
79627
..     ...         ...        ...       ...       ...       ...
...
95     WOR      4385.0   2.651490  0.685203  0.936971  2.290583  2.7
78179
96     WOW      5148.0   3.091722  0.451242  1.732146  2.745481  3.2
86299
97     WPL      5145.0   3.378239  0.446956  2.327872  3.023980  3.5
43276
98     WTC      1007.0   2.581842  0.593608  1.483875  1.931521  2.6
87847
99     XRO      1872.0   3.245025  0.596213  1.695616  2.834536  3.1
88417

    open_75%  open_max  high_count  ...  adj_close_75%  adj_close_ma
x  \
0   2.564949  2.901422      1269.0  ...       2.564180       2.89756
8
1   2.860385  3.378952      5143.0  ...       2.540187       3.23007
7
2   2.656757  3.662792      5147.0  ...       2.358652       3.65558
1
3   2.132982  2.631867      5139.0  ...       1.962802       2.39242
6
4   1.806648  3.765609      4729.0  ...       1.653319       3.76945
0
..       ...       ...         ...  ...            ...
...
95  3.195595  3.928703      4385.0  ...       2.949887       3.56018
4
96  3.386760  3.803324      5148.0  ...       3.207941       3.78582
9
97  3.683467  4.269178      5145.0  ...       3.355401       3.74003
3
98  3.108614  3.665355      1007.0  ...       3.106003       3.67244
1
99  3.709417  4.504244      1872.0  ...       3.711375       4.49981
0

    volume_count  volume_mean  volume_std  volume_min  volume_25%  v
```

```
olume_50%  \
0         1269.0    15.000475    1.061623         0.0   14.621974
15.077057
1         5143.0    14.009426    1.412156         0.0   13.776951
14.156892
2         5147.0    14.175836    1.310066         0.0   13.813363
14.279590
3         5139.0    12.610216    2.114220         0.0   11.324219
13.025139
4         4729.0    10.746235    3.136984         0.0   10.021315
11.429555
..           ...          ...          ...         ...         ...
...
95        4385.0    13.524428    1.307151         0.0   13.197447
13.710725
96        5148.0    14.707503    1.250505         0.0   14.464158
14.780520
97        5145.0    14.402136    1.286774         0.0   14.171572
14.493484
98        1007.0    12.842697    1.220000         0.0   12.217429
12.999990
99        1872.0    11.162262    1.476160         0.0   10.081550
10.869520

    volume_75%  volume_max
0    15.493659   17.932876
1    14.489767   17.735879
2    14.695308   18.280646
3    14.163663   17.614996
4    12.736880   16.352990
..         ...         ...
95   14.193944   17.297348
96   15.103383   17.481047
97   14.820545   18.421124
98   13.547082   16.510464
99   12.495376   15.932117

[100 rows x 49 columns]
```

In [77]:
```
market.shape
```

Out[77]:
```
(428618, 9)
```

Random Sampling

For ease out the compution we will we will randomize our cleaned data.

In [80]:
```
market = market.sample(n=428618, random_state=999)
market.shape
market.sample(5, random_state=999)
```

Out[80]:

	open	high	low	close	adj_close	volume	cor
369830	1.481605	1.488400	1.449269	1.449269	1.232188	15.836917	
85850	3.292018	3.295326	3.267430	3.267430	2.861718	15.124052	
325683	5.293154	5.293305	5.280917	5.289731	5.272877	13.190031	
66309	2.944439	2.950212	2.942331	2.948641	2.768000	12.374857	
218258	3.785491	3.793077	3.783023	3.784817	3.354581	13.980044	

In [ ]:

```
```

In [ ]:

```
```

Data Exploration and Visualisation

After thorough checks of the columns we are considering our data is cleaned. Now, lets explore the data and visualize as well

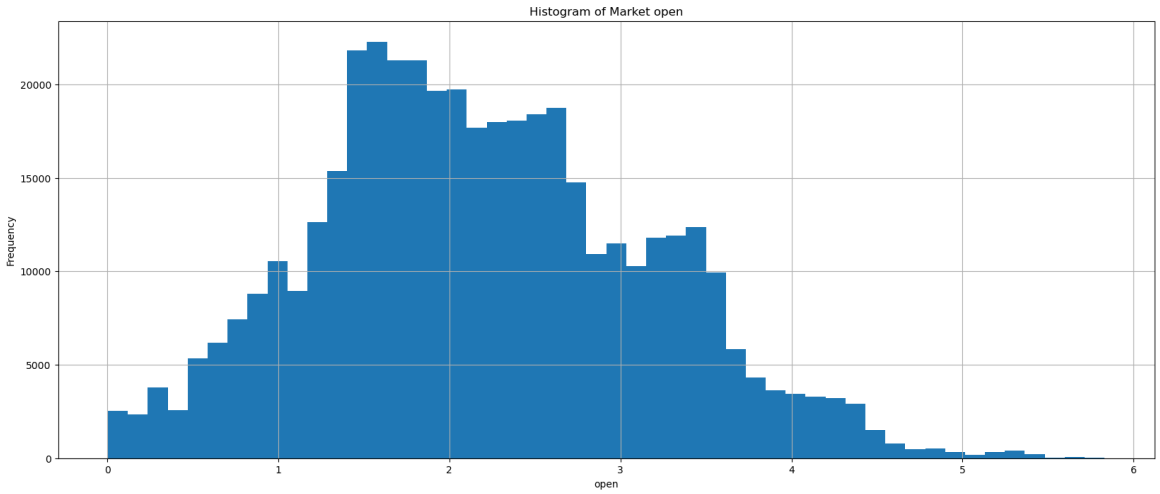
Univariate Visualisation

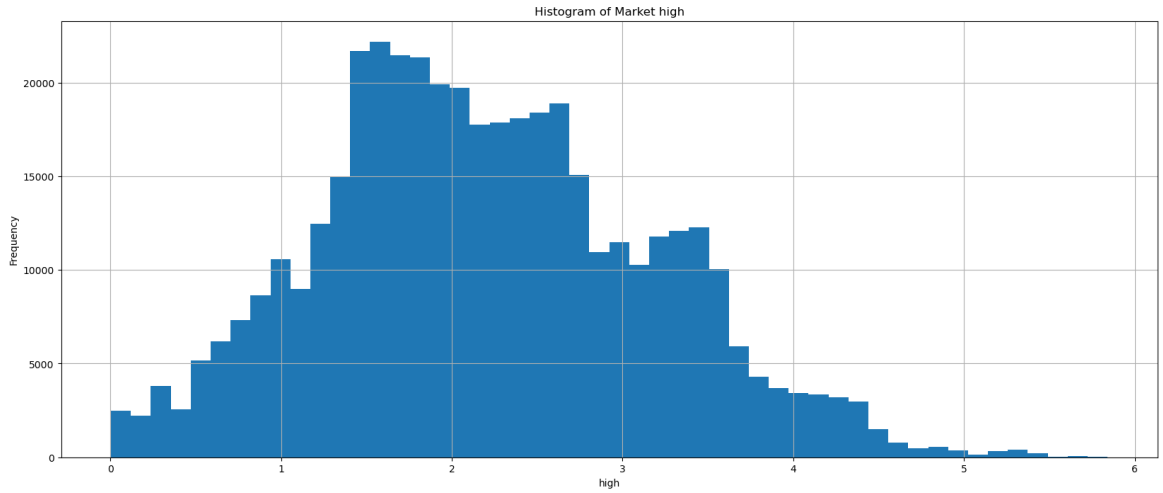
At first lets visualize the numerical varaiable

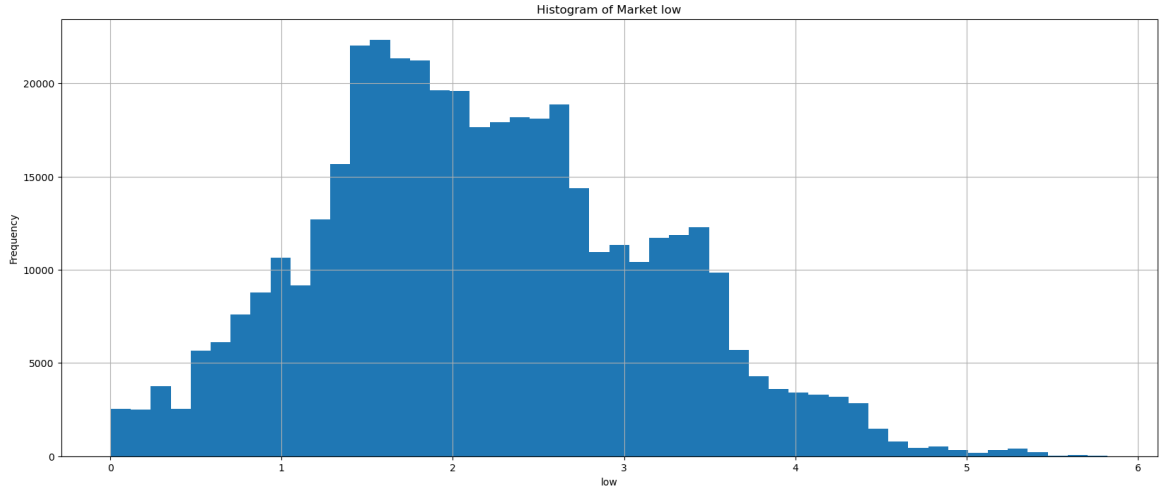
Histogram of Numeric Variable

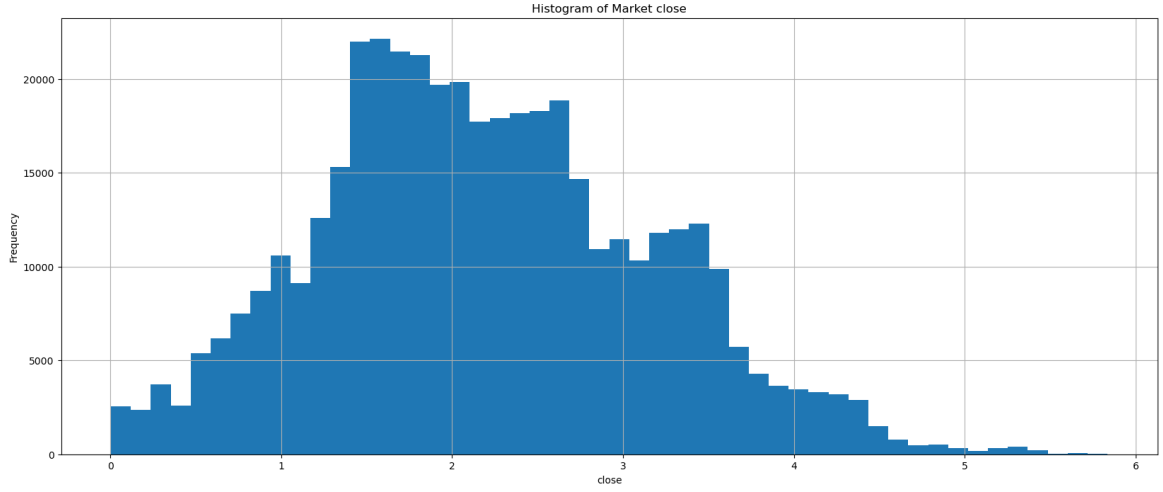
In [86]:

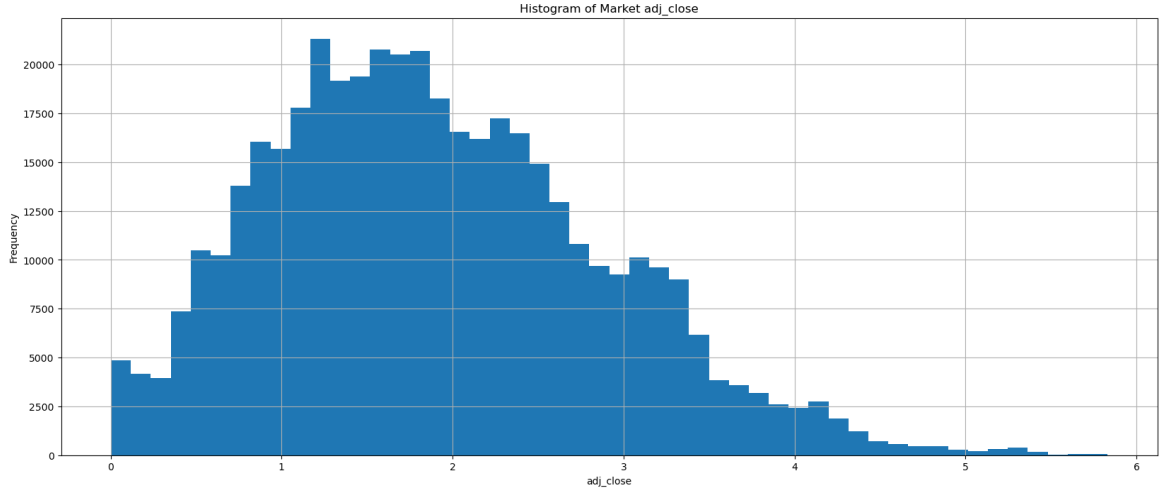
```
for var in variables:
    plt.figure(figsize=(20, 8))
    market[var].hist(bins=50)
    plt.title(f'Histogram of Market {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```

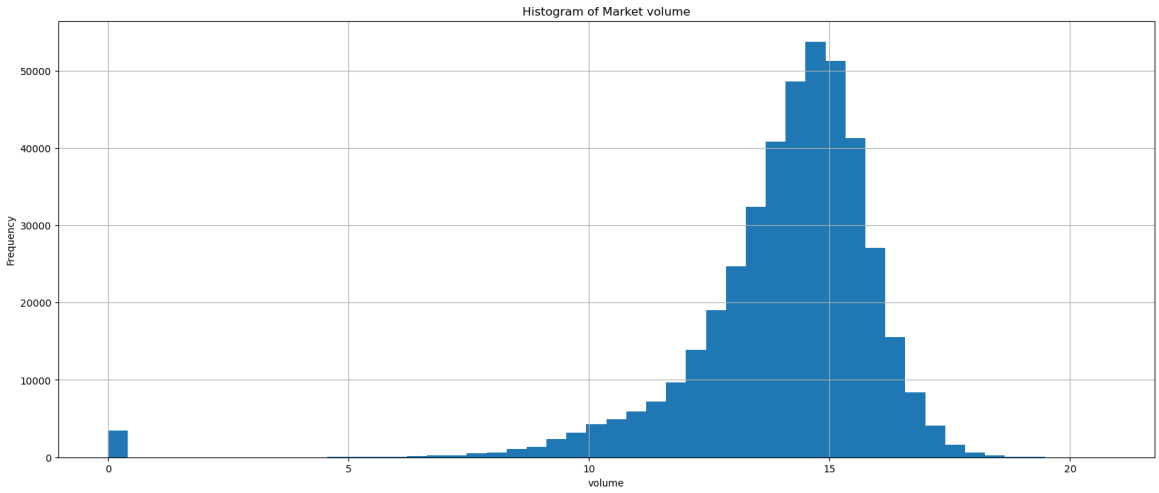



Histogram of Market high
Frequency
high
0
5000
10000
15000
20000
0
1
2
3
4
5
6


Histogram of Market low
Frequency
low
0
5000
10000
15000
20000
0
1
2
3
4
5
6


Histogram of Market close
Frequency
close
0
5000
10000
15000
20000
0
1
2
3
4
5
6


Histogram of Market adj_close
Frequency
adj_close
0
2500
5000
7500
10000
12500
15000
17500
20000
0
1
2
3
4
5
6

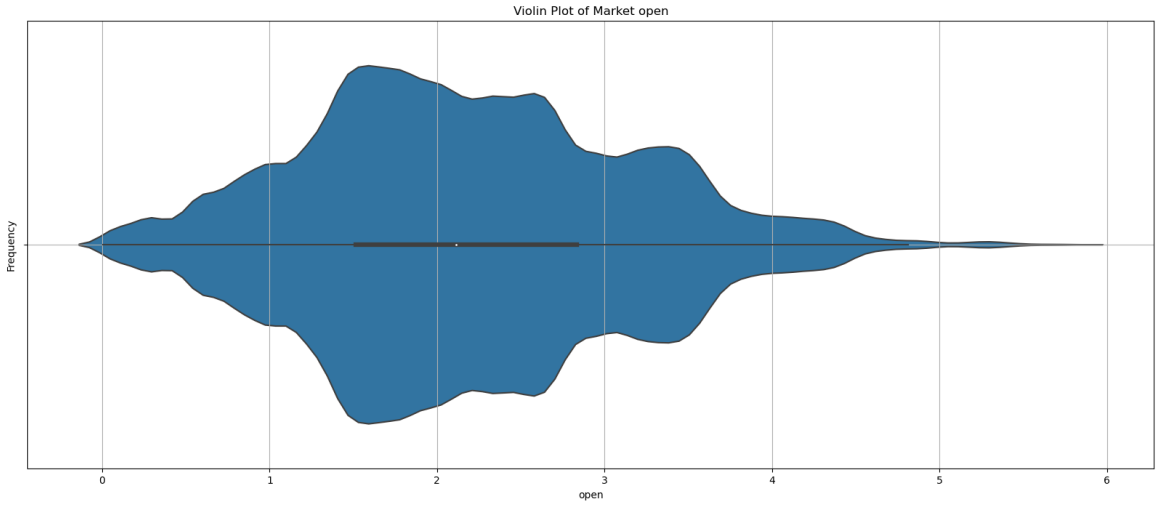


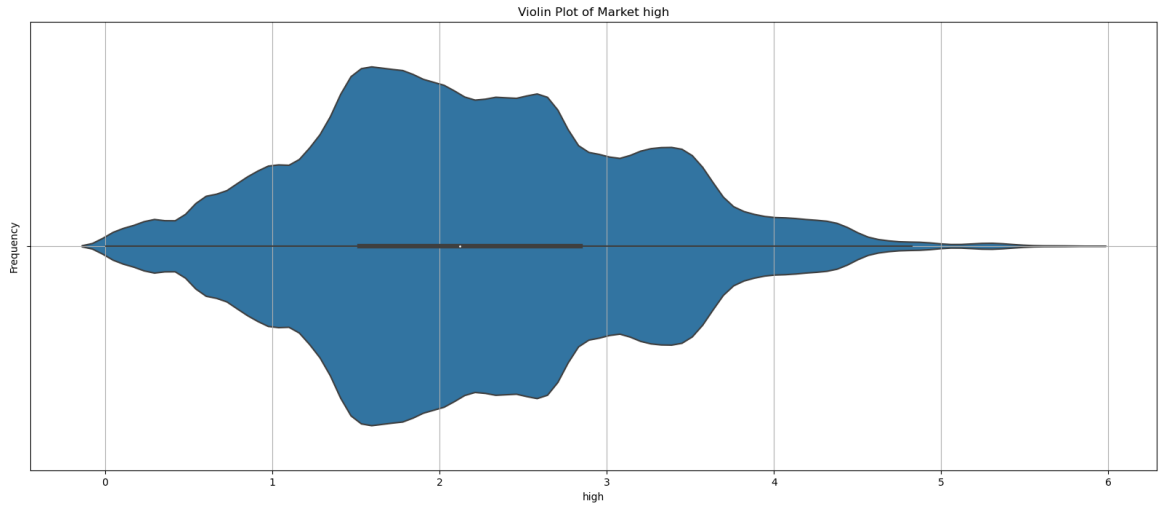
From above graph we can verify that our data has almost no ouliers expcept te volume features. We achieved this by doinf lograthimic transformation.

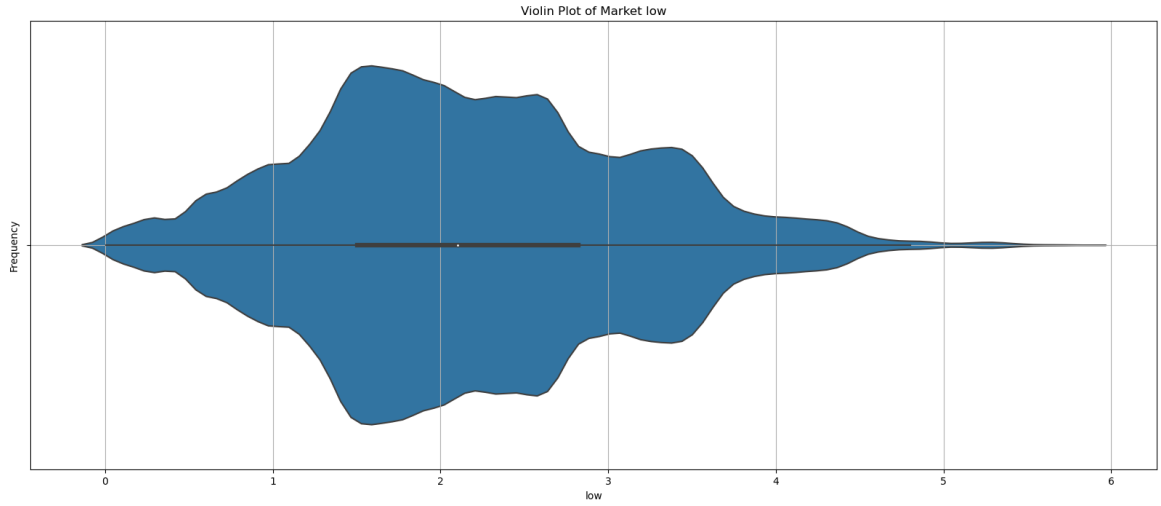
The graph for `open`, `high`, `low`, `close` graphs are symmetrical. However, the graph of `adj_close` and `volume` are right and left skwed data.

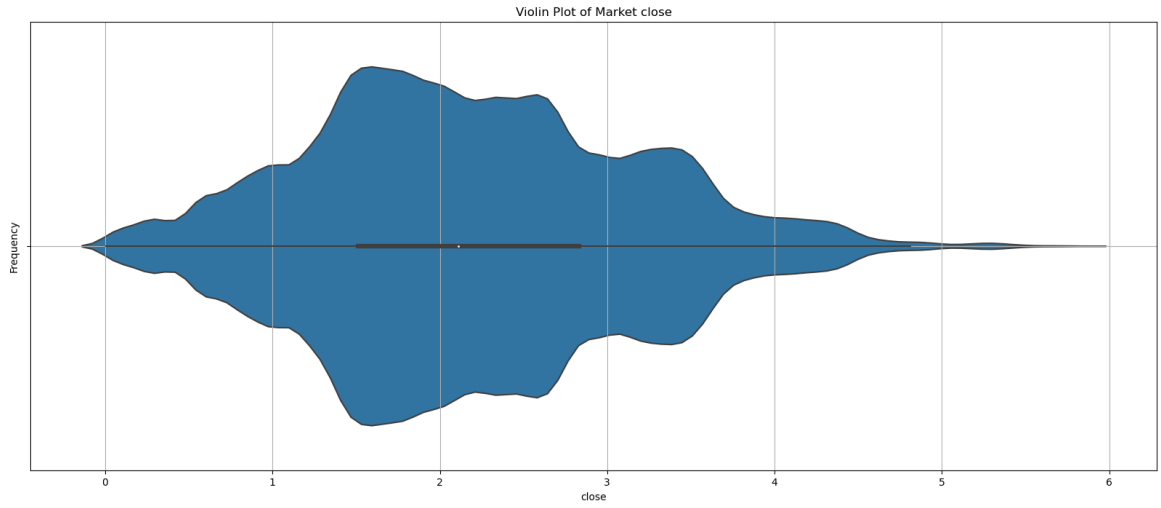
Violin Plot of Numeric Variables

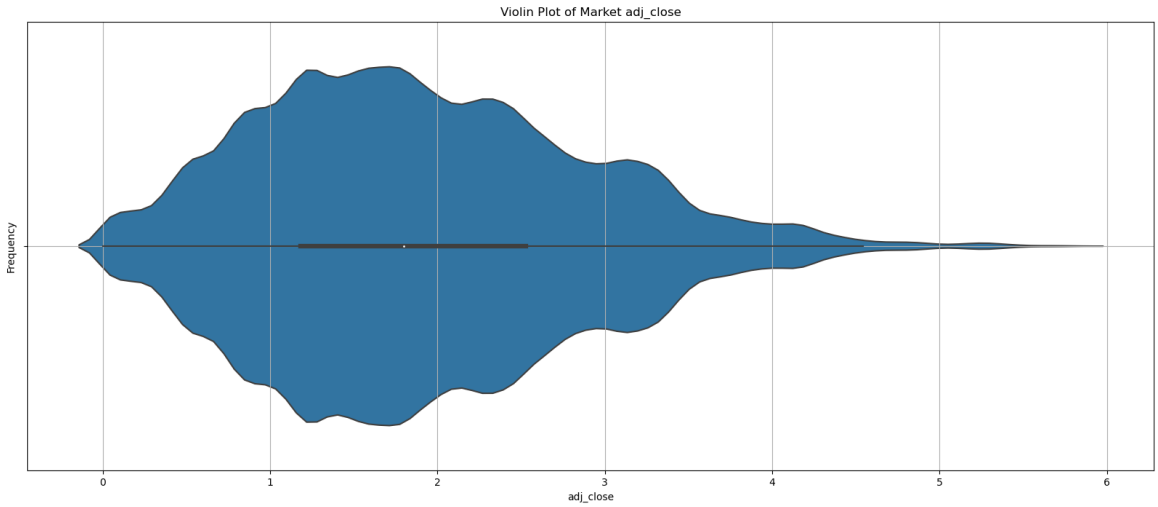
In [89]:
```
for var in variables:
    plt.figure(figsize = (20, 8))
    sns.violinplot(x = market[var])
    plt.title(f'Violin Plot of Market {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```

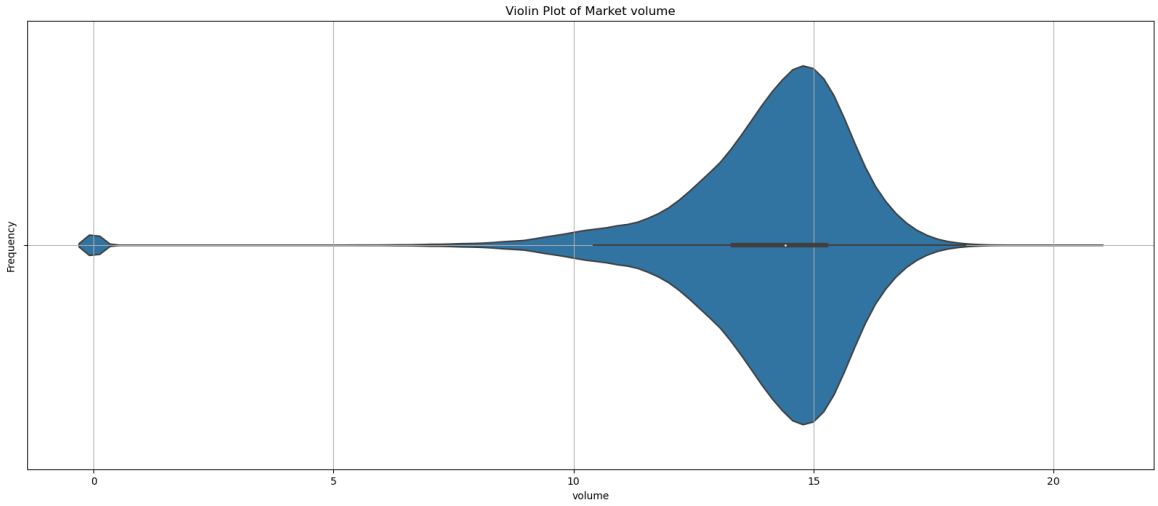



Violin Plot of Market high
Frequency
0
1
2
3
4
5
6
high


Violin Plot of Market low
Frequency
0
1
2
3
4
5
6
low


Violin Plot of Market close
Frequency
0
1
2
3
4
5
6
close

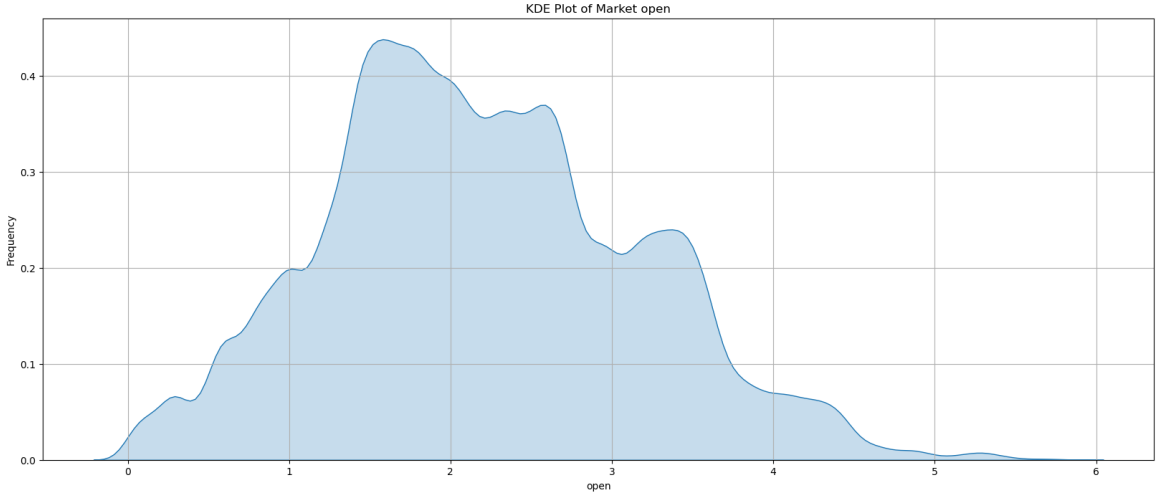


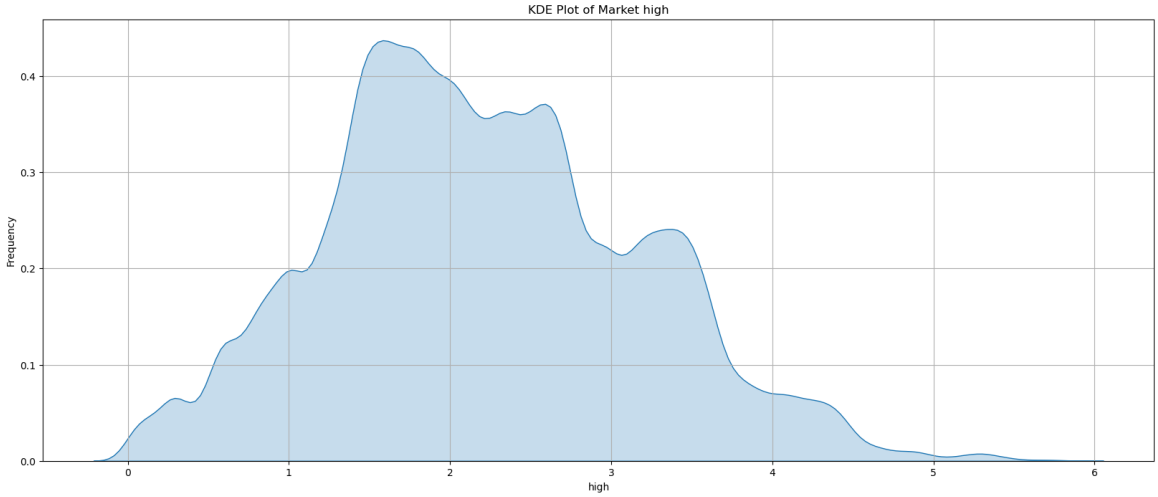


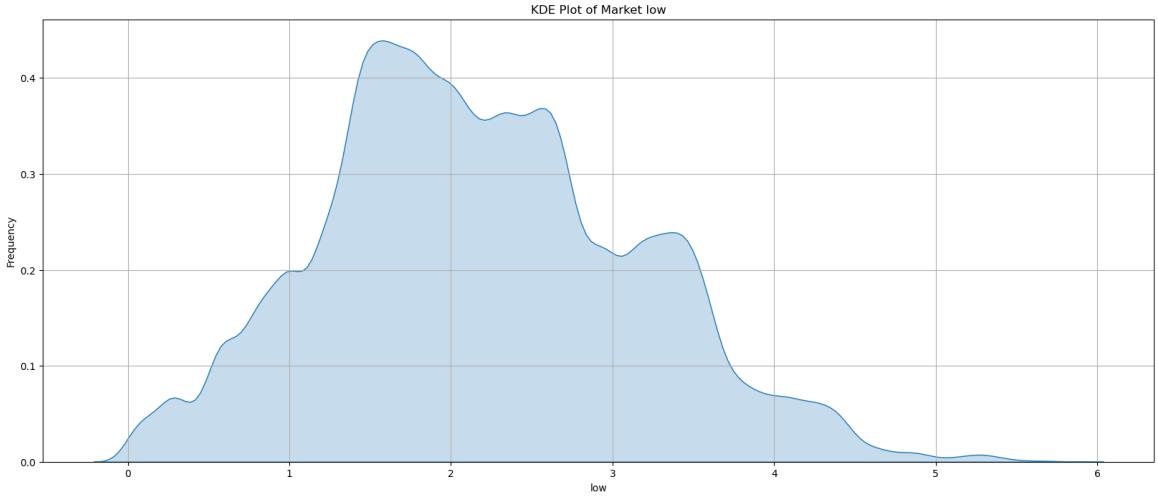
The violin plots show varied distributions for each variable, where var1 is (unimodal/bimodal) with some outliers, var2 is (uniform/normal) and is centered at (mean/median) with no significant outliers, and var3 is (skewed/symmetric) with the peak at (specific value) and (few/many) outliers. The distributions convey details regarding the concentration and variability of data, expressing trends such as (skewness/normal distribution). The spread and outliers show areas that may be explored further or are indicative of some patterns within the data set.

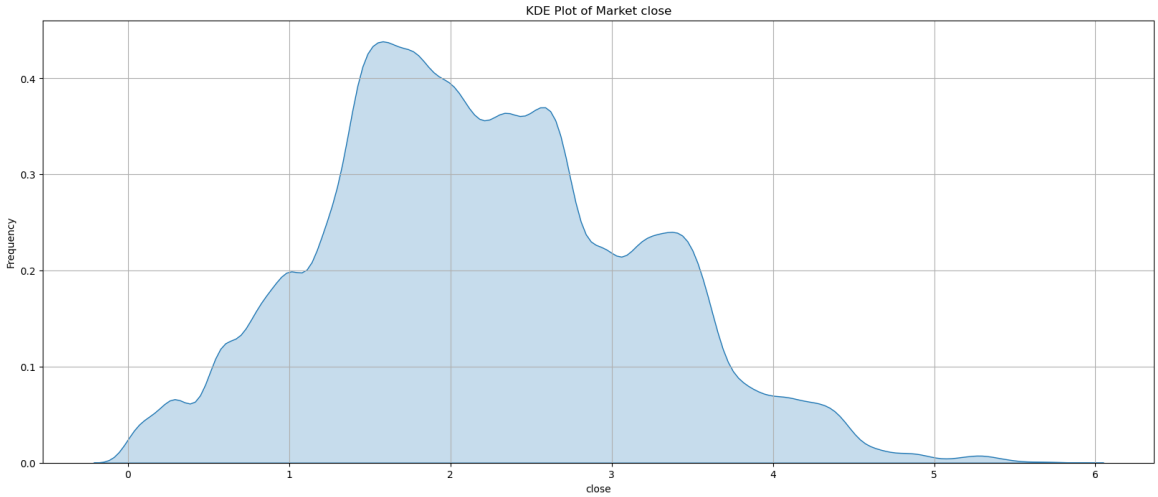
KDE Plot of Numeric Variables

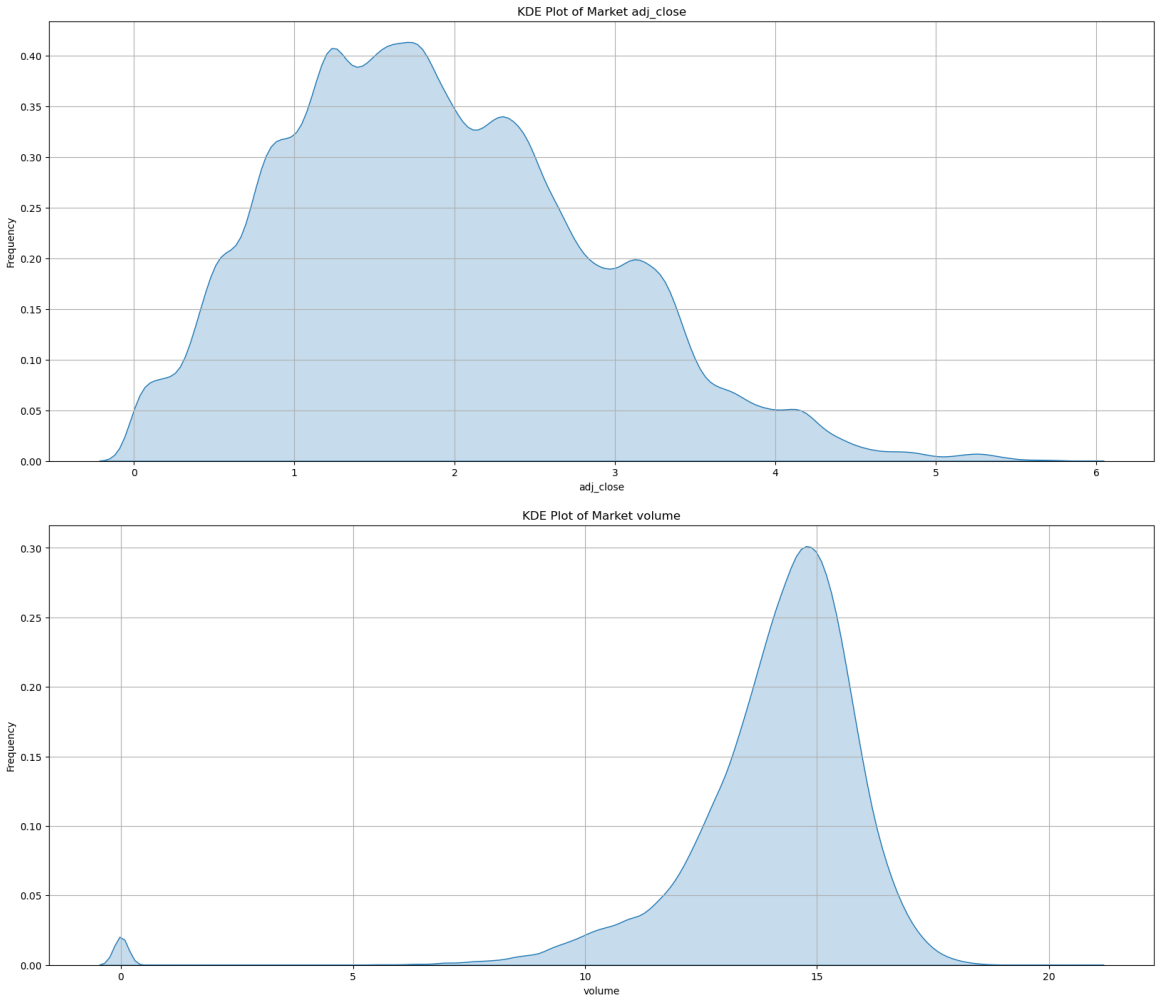
In [92]:
```
for var in variables:
    plt.figure(figsize = (20, 8))
    sns.kdeplot(market[var], shade = True)
    plt.title(f'KDE Plot of Market {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```


KDE Plot of Market open
Frequency
0.0
0.1
0.2
0.3
0.4
0
1
2
3
4
5
6
open


KDE Plot of Market high
Frequency
0.0
0.1
0.2
0.3
0.4
0
1
2
3
4
5
6
high


KDE Plot of Market low
Frequency
0.0
0.1
0.2
0.3
0.4
0
1
2
3
4
5
6
low

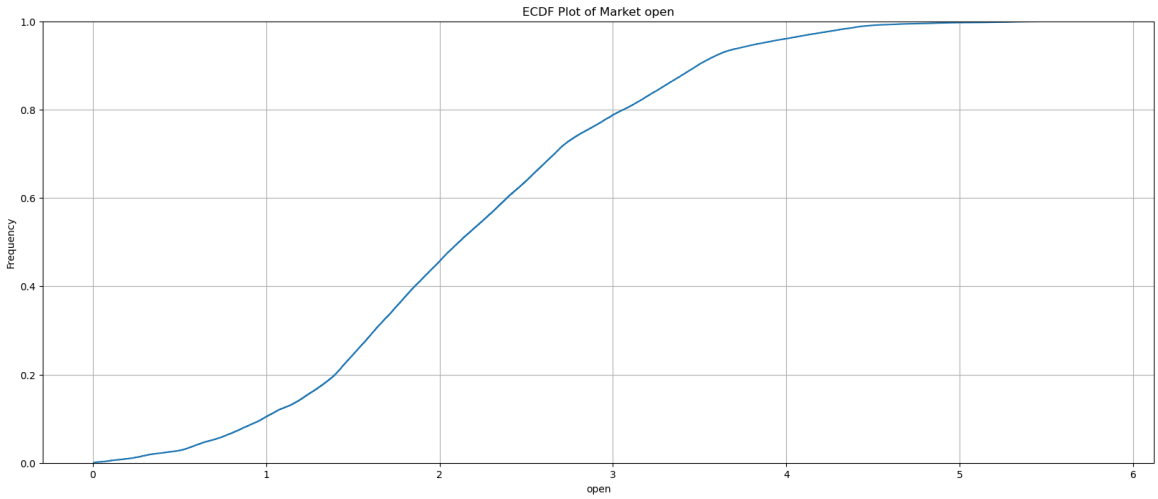

KDE Plot of Market close
Frequency
0.0
0.1
0.2
0.3
0.4
0
1
2
3
4
5
6
close

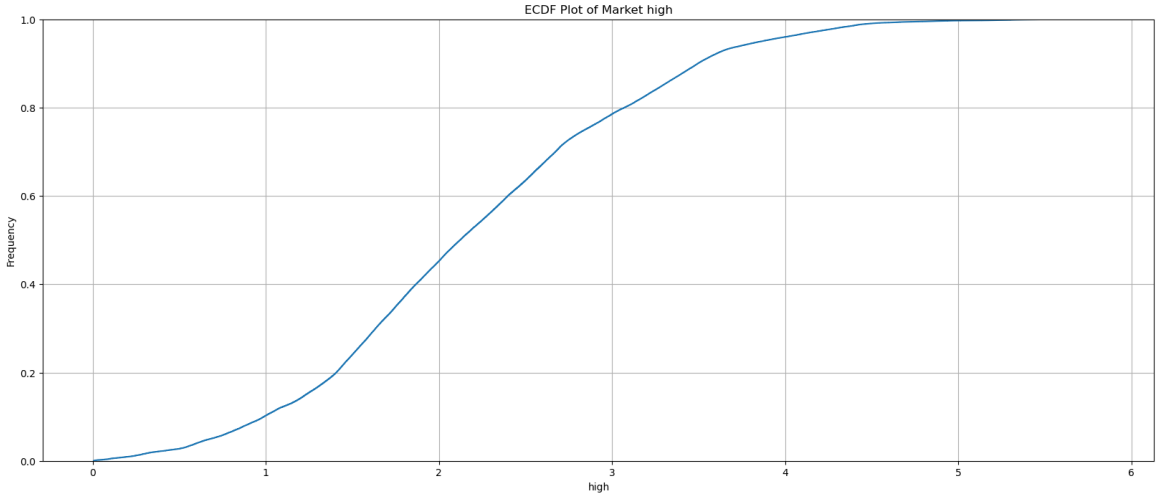


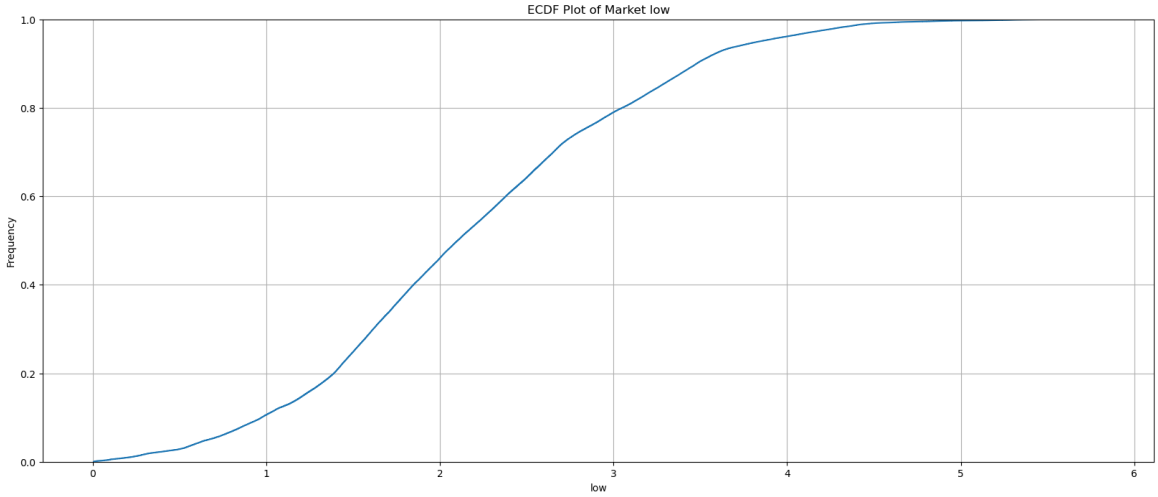
The KDE plot for each variable is a plot of the estimated probability density of the distribution of the data. The plot is a view into the (smooth/peaked) distribution, showing areas of high frequency. It shows the (skewness/symmetry) of the data along with concentration of the values and (narrow/wide) spread showing variability.

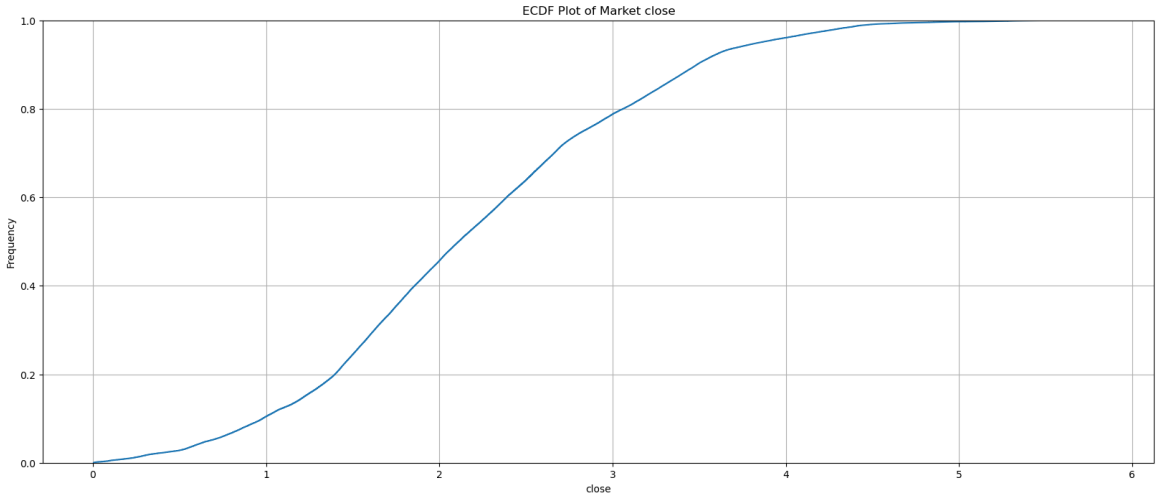
Empirical Cumulative Distribution Function Plot for Numerical Variables

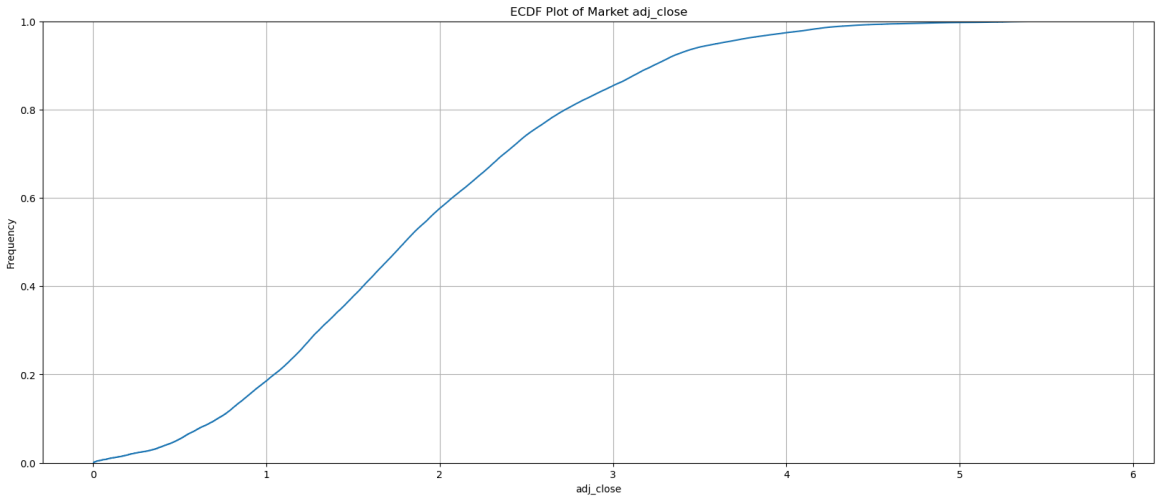
In [95]:
```
for var in variables:
    plt.figure(figsize = (20, 8))
    sns.ecdfplot(market[var])
    plt.title(f'ECDF Plot of Market {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```

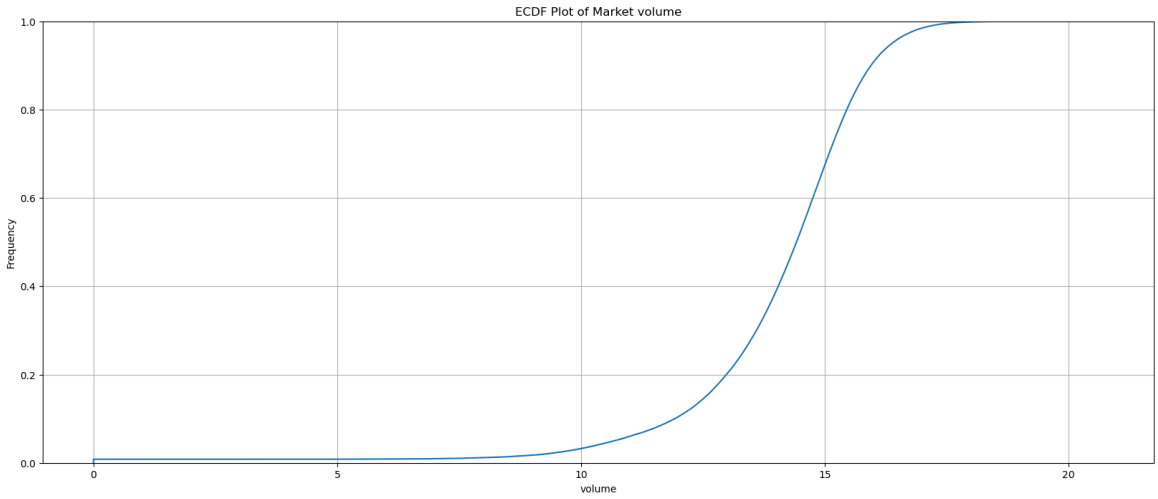

ECDF Plot of Market open
Frequency
open
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6


ECDF Plot of Market high
Frequency
high
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6


ECDF Plot of Market low
Frequency
low
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6


ECDF Plot of Market close
Frequency
close
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6



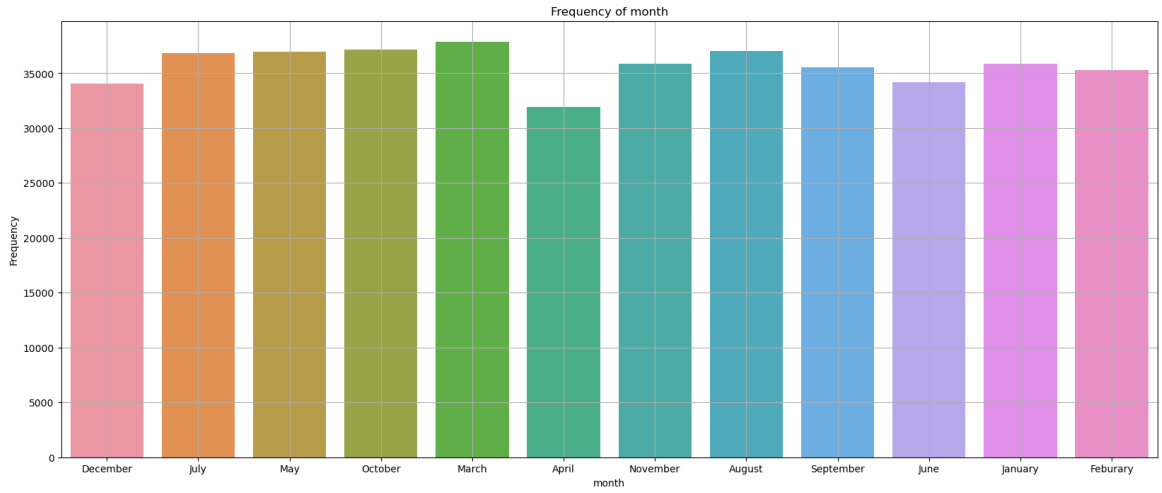


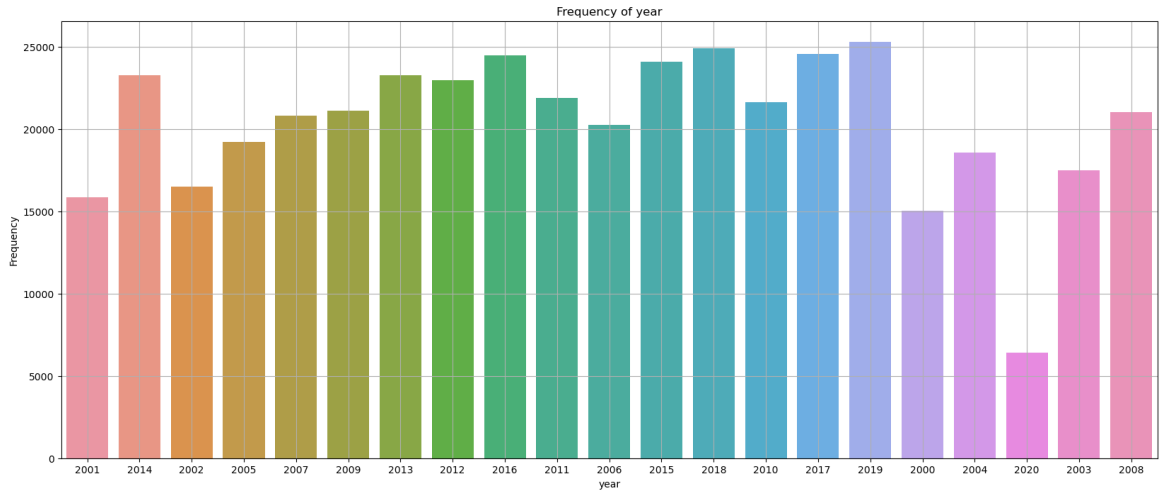
Apart from illustrating the distribution (spread/range) of the distribution and where data points cluster, the ECDF plot for each variable also illustrates the cumulative distribution of the data, illustrating the proportion of values less than or equal to each point. It also illustrates whether the data is (uniform/skewed), a steep curve illustrating regions of high density.

Now lets visualize the categorical variable

Bar Plot/ Count Plot of Categorical Variables

In [99]:
```
categorical = ['month', 'year']
for var in categorical:
    plt.figure(figsize = (20, 8))
    sns.countplot(x = var, data = market)
    plt.title(f'Frequency of {var}')
    plt.xlabel(var)
    plt.ylabel('Frequency')
    plt.grid(True)
    plt.show()
```

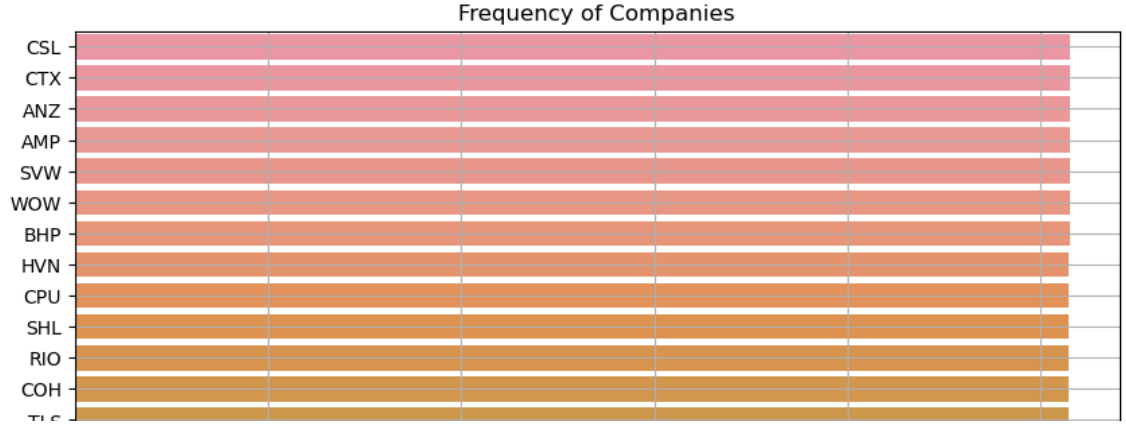


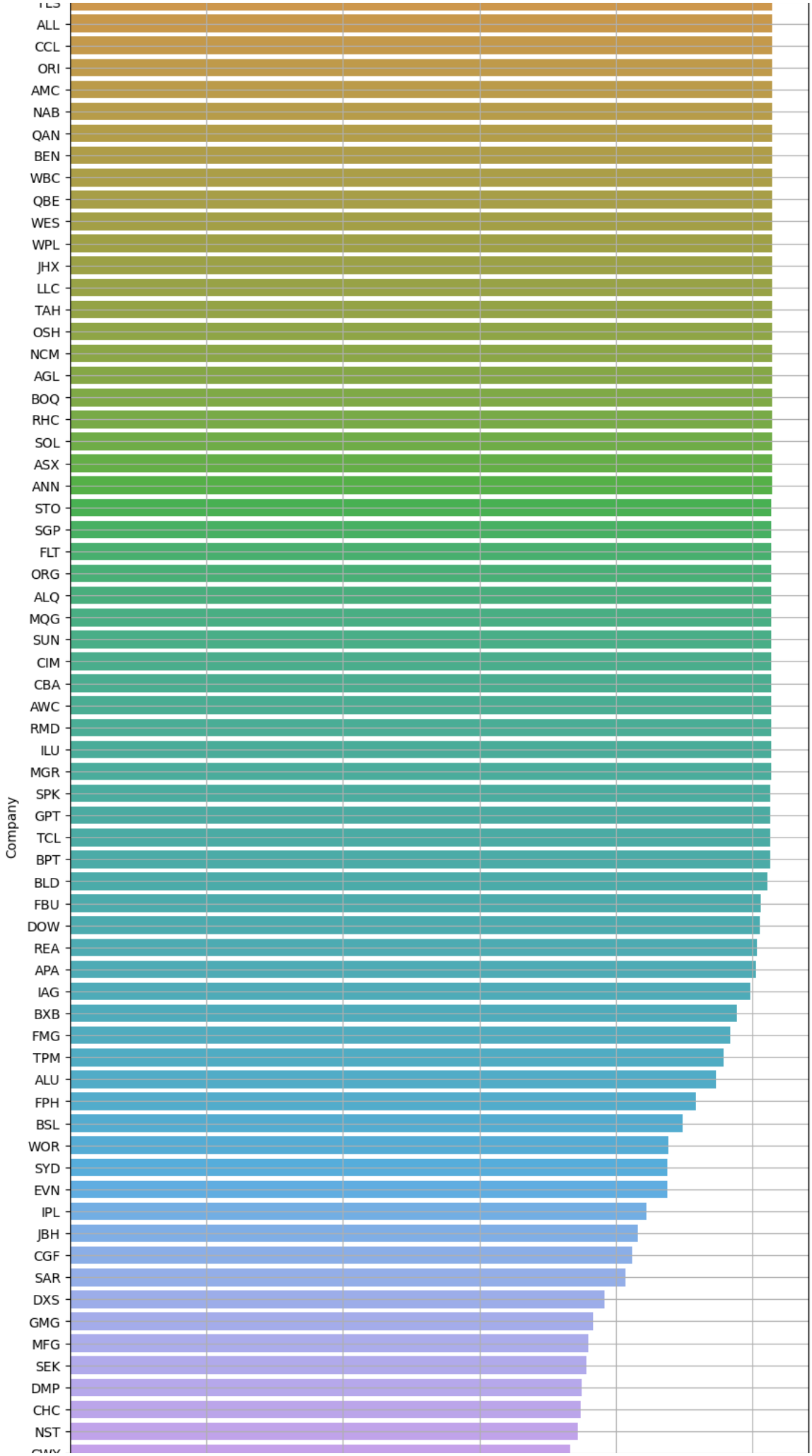


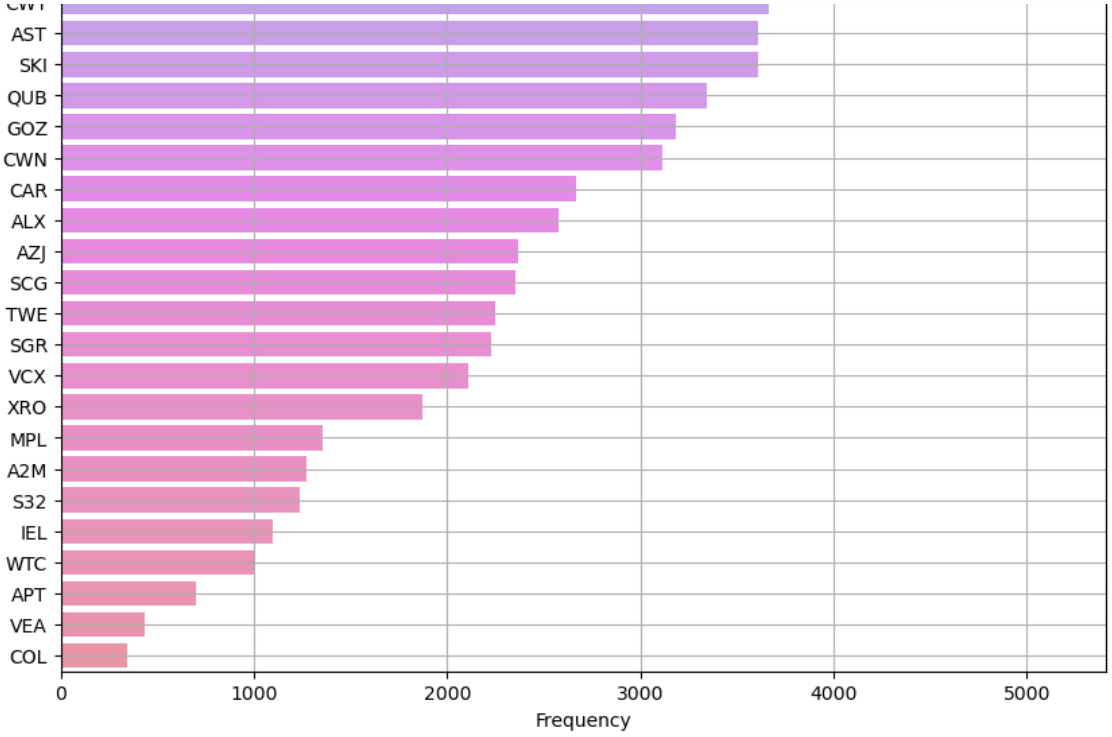
The count plot for each categorical variable shows the frequency of each category in the data. It highlights the distribution of data over different categories, with (some categories) representing the highest and lowest counts. The plot provides a visual representation of how the data is distributed in these categorical values.

In [101…

```
plt.figure(figsize=(10, 30))
sns.countplot(y='company', data = market, order =
market['company'].value_counts().index)
plt.title('Frequency of Companies')
plt.ylabel('Company')
plt.xlabel('Frequency')
plt.grid(True)
plt.show()
```




Company
ALL
CCL
ORI
AMC
NAB
QAN
BEN
WBC
QBE
WES
WPL
JHX
LLC
TAH
OSH
NCM
AGL
BOQ
RHC
SOL
ASX
ANN
STO
SGP
FLT
ORG
ALQ
MQG
SUN
CIM
CBA
AWC
RMD
ILU
MGR
SPK
GPT
TCL
BPT
BLD
FBU
DOW
REA
APA
IAG
BXB
FMG
TPM
ALU
FPH
BSL
WOR
SYD
EVN
IPL
JBH
CGF
SAR
DXS
GMG
MFG
SEK
DMP
CHC
NST

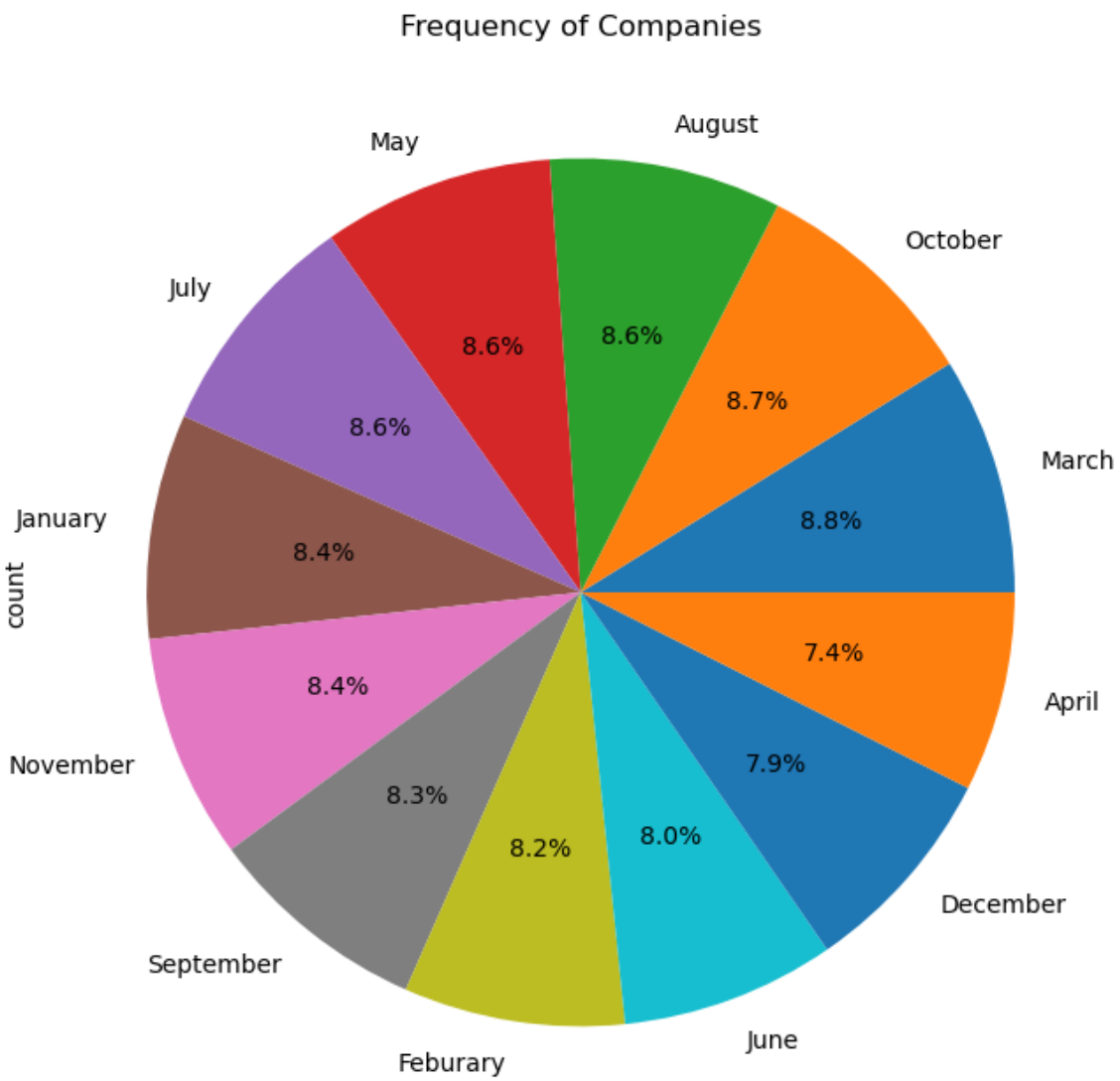


Companies are ordered based on the count of occurrences within the count plot, which gives the frequency each company has within the data. With largest and smallest frequency having (some businesses), it indicates the range of businesses. Which of the companies drive the data and which have a smaller number of observations can be inferred from the plot.

Pie Chart for Trade per Month

In [104…

```
plt.figure(figsize = (20, 8))
market['month'].value_counts().plot.pie(autopct='%1.1f%%')
plt.title('Frequency of Companies')
plt.show()
```

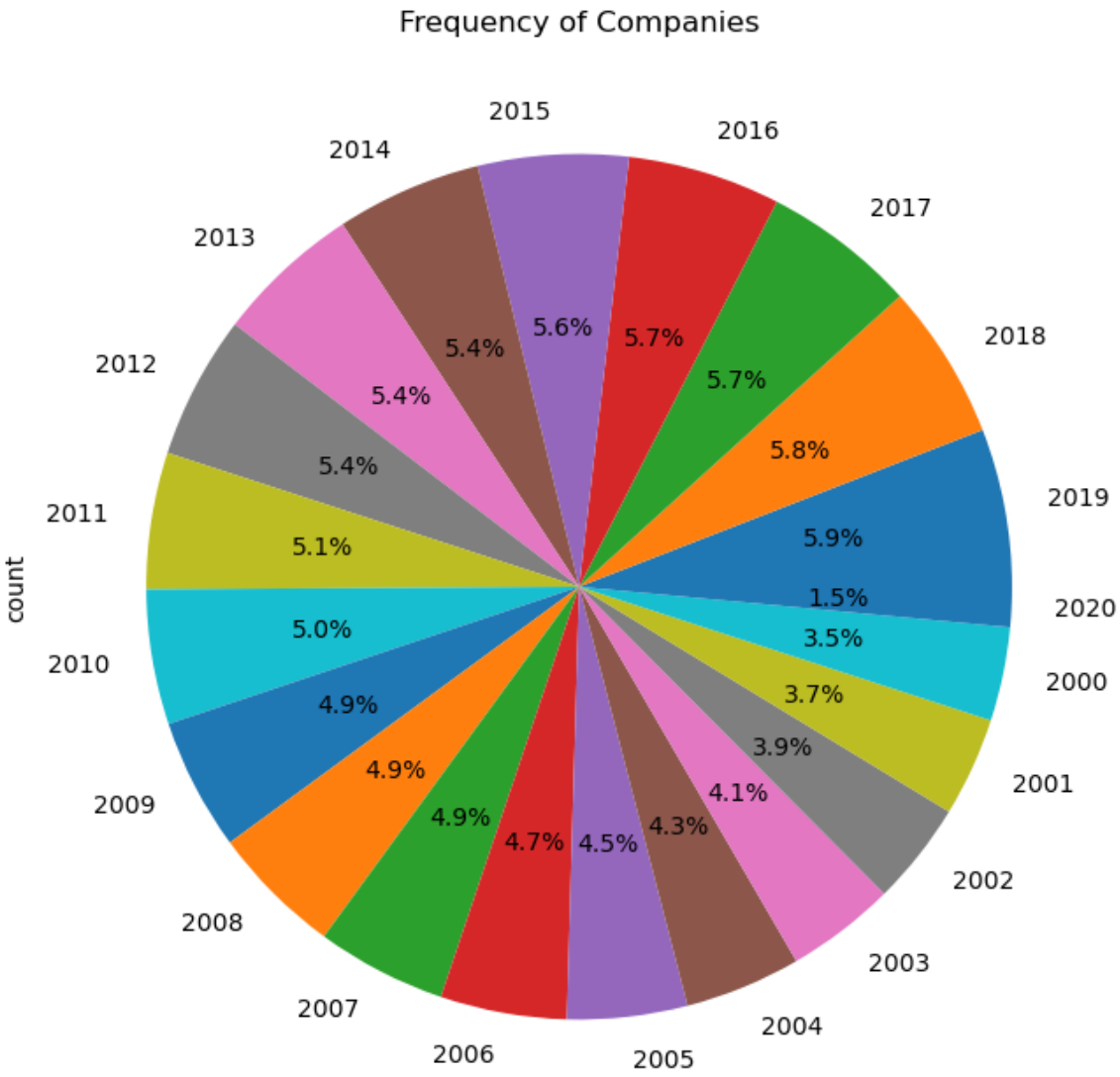


The breakdown of the "month" variable, presenting the percentage of frequency of each month, is presented in the pie chart. All of these wedges depict the percentage of information for the corresponding month, highlighting the most occurring and the least occurring months. The relative frequency of data by months is presented graphically in the chart.

Pie chart for Trades per Year

In [107…

```
plt.figure(figsize = (20, 8))
market['year'].value_counts().plot.pie(autopct='%1.1f%%')
plt.title('Frequency of Companies')
plt.show()
```



The pie chart shows the frequency distribution of the "year" variable as a percentage of occurrences by year. It highlights the relative frequency of data by year, and every segment in the chart represents a specific year. This makes it easier to identify the most frequent and least frequent years in the dataset.

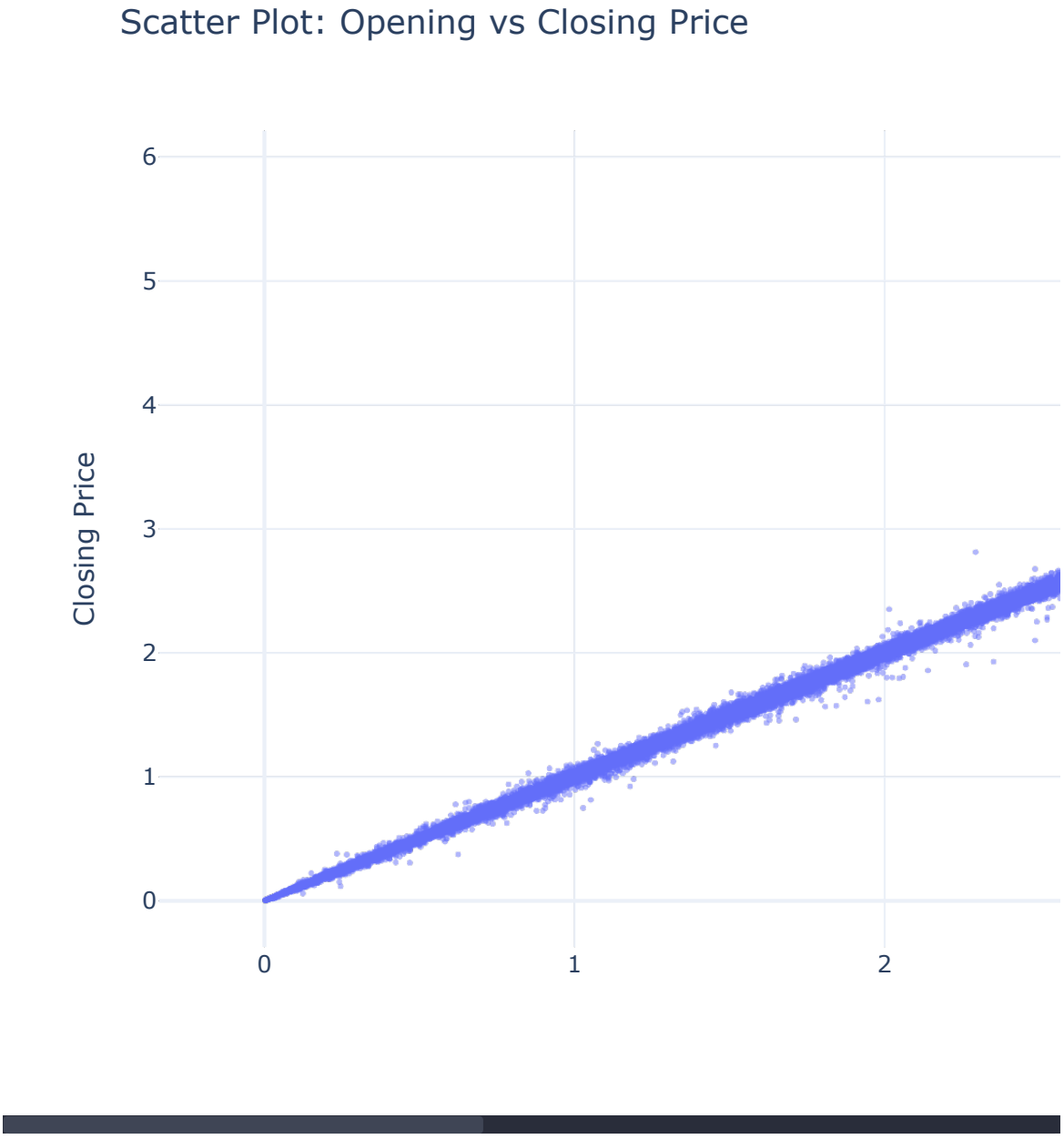
Two Variable Visualisation

Scatter Plot for Opening and Closing Price

In [111…

```
fig = px.scatter(
    market,
    x='open',
    y='close',
    title='Scatter Plot: Opening vs Closing Price',
    labels={'open': 'Opening Price', 'close': 'Closing Price'},
    opacity=0.5,  # makes overlapping points easier to see
    height=600,
    width=1200
)
```

```
fig.update_traces(marker=dict(size=3))  # smaller dots for better
performance
fig.update_layout(template='plotly_white')
fig.show()
```



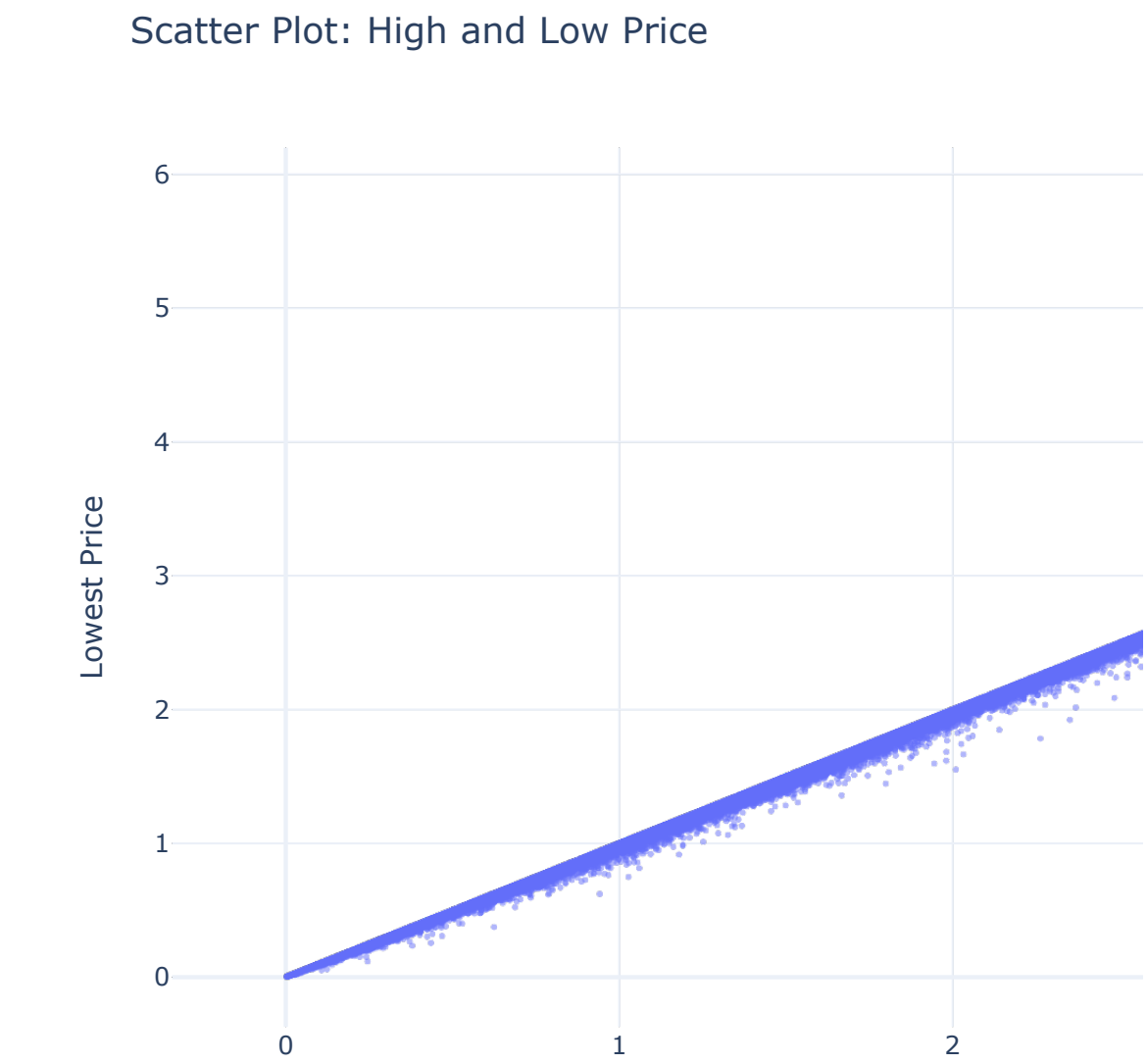
Since each point corresponds to a pair of values, the scatter plot shows the relationship between the opening and closing prices. It highlights any outliers or trends in the data by revealing the connection between the opening and closing prices. The use of smaller markers improves legibility, and the graphic does a great job of displaying the distribution and density of data points.

Scatter Plot for High and Low Price

In [114…

```
fig = px.scatter(
    market,
    x='high',
    y='low',
```

```
    title='Scatter Plot: High and Low Price',
    labels={'high': 'Highest Price', 'low': 'Lowest Price'},
    opacity=0.5,  # makes overlapping points easier to see
    height=600,
    width=1200
)

fig.update_traces(marker=dict(size=3))  # smaller dots for better
performance
fig.update_layout(template='plotly_white')
fig.show()
```

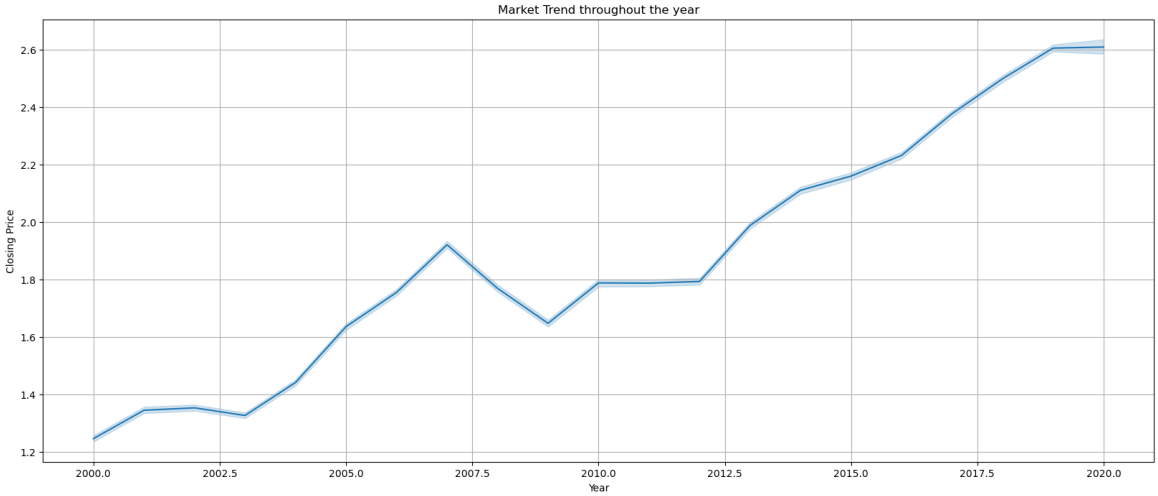


The scatter plot illustrates the distribution of values and how the values are dispersed, with the smaller markers giving clearer perspectives, particularly in areas where points overlap. It illustrates the difference between the highest and lowest prices, where every point is a pair of values. It also illustrates the connection between the high and low prices and whether there is any pattern or discrepancy in the data.

Line Plot for Showing the Trend

In [117…

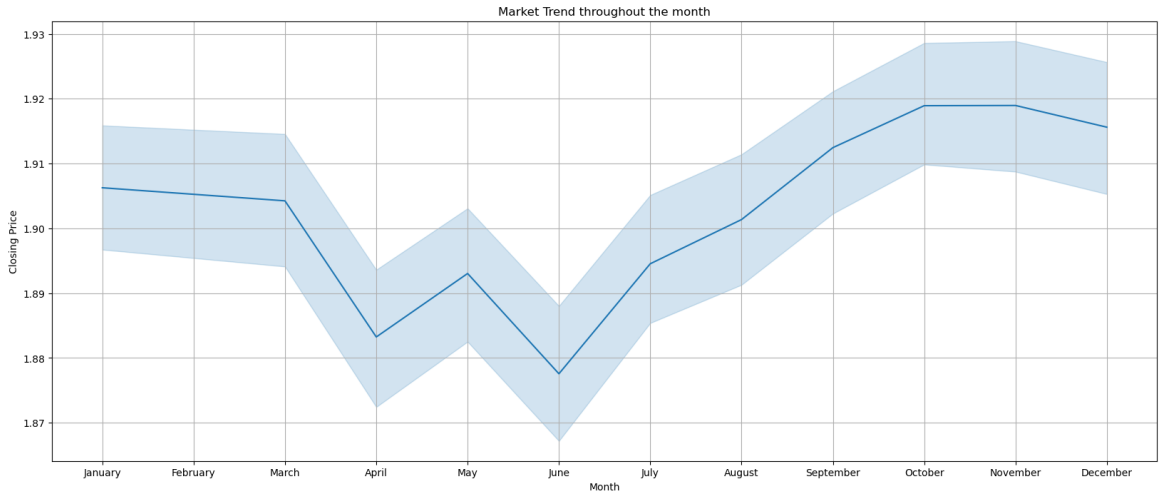
```
market['year'] = pd.to_numeric(market['year'], errors='coerce')
plt.figure(figsize = (20, 8))
sns.lineplot(x='year', y='adj_close', data=market)
plt.title(f'Market Trend throughout the year')
plt.xlabel('Year')
plt.ylabel('Closing Price')
plt.grid(True)
plt.show()
```



Each point on the line plot represents the closing price for every year, and it indicates the trend of the adjusted closing price over time. It indicates the historical price fluctuations in the closing of the market, pointing out any trends upward or downward. The plot gives a good picture of how well the market is doing, and the grid makes it easy to compare price fluctuations over time.

In [119…

```
month_order = ['January', 'February', 'March', 'April', 'May',
'June',
               'July', 'August', 'September', 'October',
'November', 'December']

market['month'] = pd.Categorical(market['month'],
categories=month_order, ordered=True)

plt.figure(figsize = (20, 8))
sns.lineplot(x='month', y='adj_close', data=market)
plt.title(f'Market Trend throughout the month')
plt.xlabel('Month')
plt.ylabel('Closing Price')
plt.grid(True)
plt.show()
```

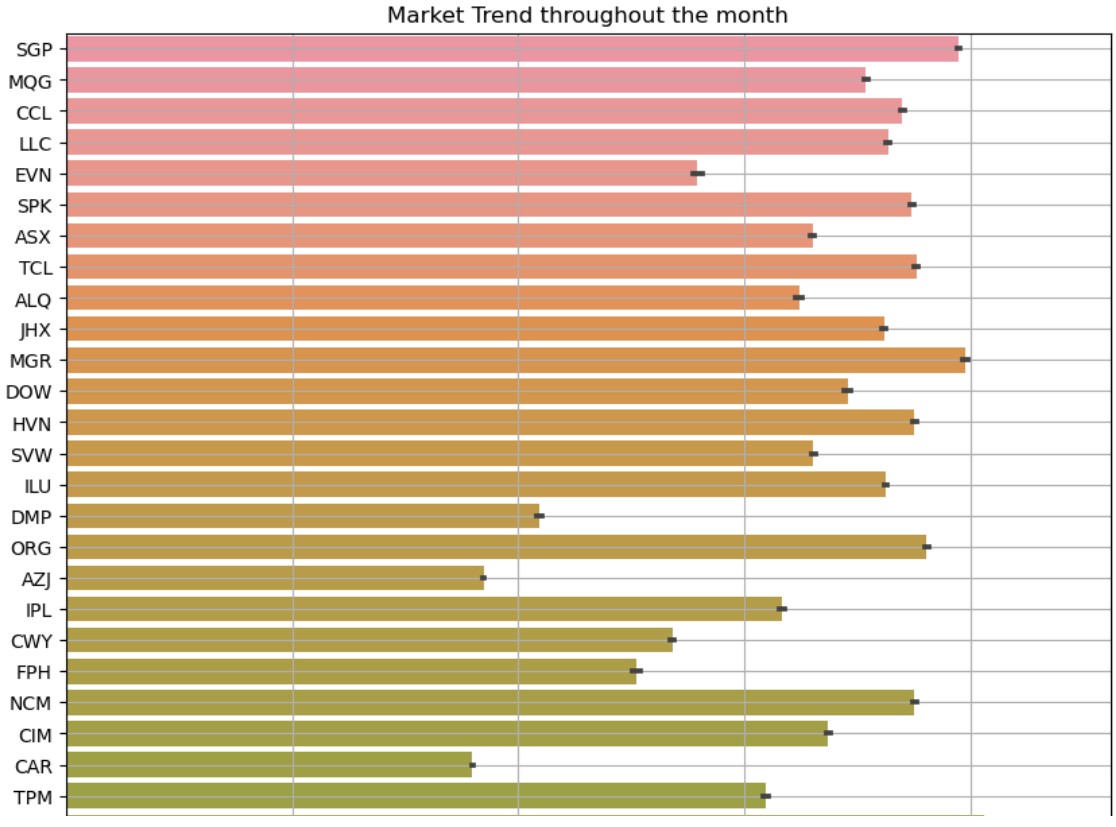


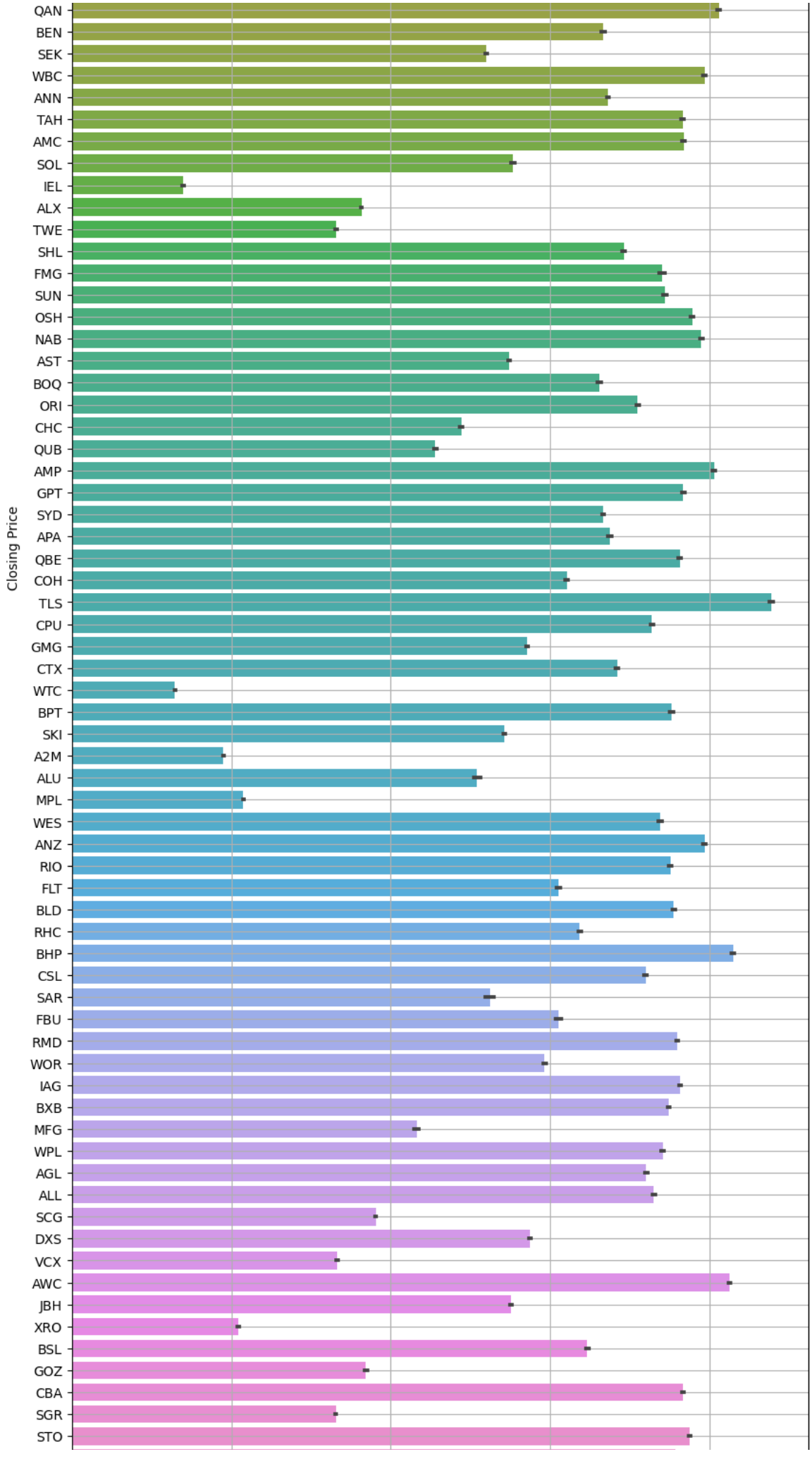
Each point on the line plot is the closing price of the respective month, and it represents the adjusted closing price pattern over the year. It identifies any seasonal pattern or trend by highlighting the closing price changes of the market over the year. A grid facilitates comparing the price movement among the months, and the plot provides a clear image of monthly variations in closing prices.

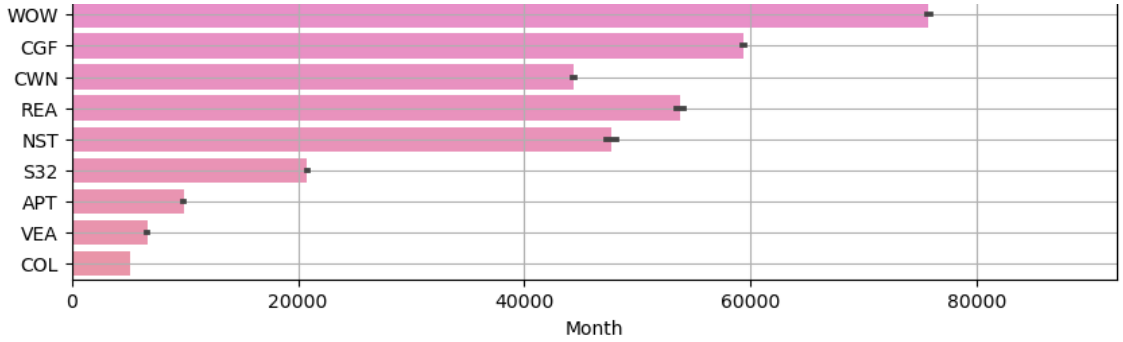
Bar plot for Trades by compnies

In [122…

```
plt.figure(figsize = (10, 30))
sns.barplot(y='company', x='volume', data=market, estimator='sum')
plt.title(f'Market Trend throughout the month')
plt.xlabel('Month')
plt.ylabel('Closing Price')
plt.grid(True)
plt.show()
```




Closing Price
QAN
BEN
SEK
WBC
ANN
TAH
AMC
SOL
IEL
ALX
TWE
SHL
FMG
SUN
OSH
NAB
AST
BOQ
ORI
CHC
QUB
AMP
GPT
SYD
APA
QBE
COH
TLS
CPU
GMG
CTX
WTC
BPT
SKI
A2M
ALU
MPL
WES
ANZ
RIO
FLT
BLD
RHC
BHP
CSL
SAR
FBU
RMD
WOR
IAG
BXB
MFG
WPL
AGL
ALL
SCG
DXS
VCX
AWC
JBH
XRO
BSL
GOZ
CBA
SGR
STO



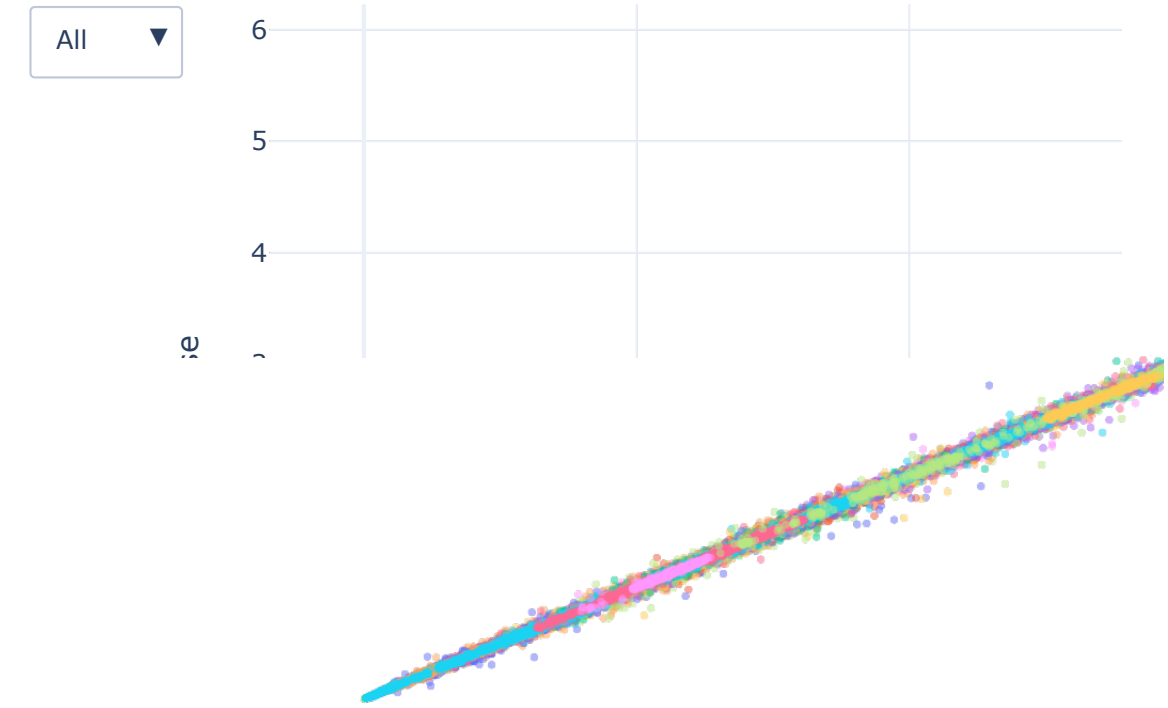
The total trade volume of each firm is shown by the bar plot, and it shows the total volume of all firms in the data. It presents a clear distinction among all firms by showing those with the largest and smallest volumes of trade. Trends of trading activity are shown in the plot, and volume differences between firms are simpler to compare using the grid.

Three variable Visualisation

Scatter Plot of open and high based on companies

In [126…

```
fig = px.scatter(
    market,
    x='open',
    y='close',
    color='company',
    hover_data=['company', 'year'],
    title='Interactive Scatter Plot by Company',
    opacity=0.5
)

fig.update_traces(marker=dict(size=4))
fig.update_layout(
    template='plotly_white',
    legend_title_text='Company',
    showlegend=False,
    updatemenus=[{
        "buttons": [
            {"label": "All", "method": "restyle", "args":
[{"visible": [True]*len(market['company'].unique())}]}
        ] + [
            {"label": comp, "method": "restyle", "args":
[{"visible": [company == comp for company in
market['company'].unique()]}]}
            for comp in market['company'].unique()[:20]  # Only
include first 20 for performance
        ],
        "direction": "down",
        "showactive": True,
    }]
)

fig.show()
```



The color-coded interactive scatter plot displays the correspondence between opening and closing prices. Through adjustment of visibility, it is simple to compare businesses, and a hovering point indicates detailed information for a business and a year. With the facility to look at individual businesses in detail to analyze, the plot offers an interactive means of examining industry trends.

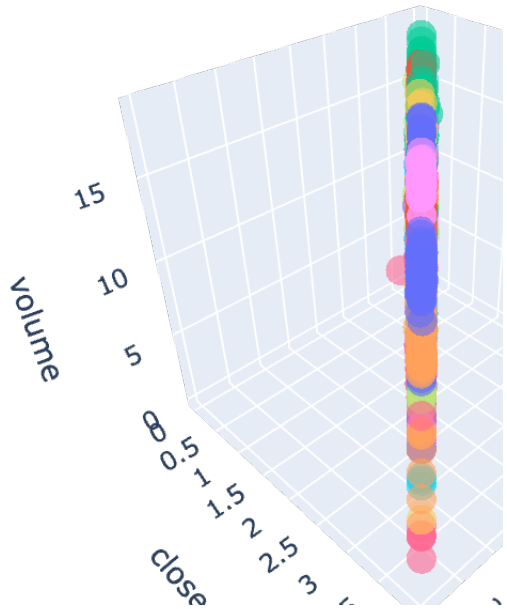
3D Scatter Plot

Plotting 3D scatter plot with volume, open and close price in their respective axis and to optimize we categorise it by year.

In [130…
```
sampled = market.sample(n=100000, random_state=42)
```

In [131…
```
fig = px.scatter_3d(
    sampled,
    x='open',
    y='close',
    z='volume',
    color='company',
    animation_frame='year',
```

```
    opacity=0.6
)
fig.show()
```



Color-coded by company, the 3D scatter plot shows the interaction between the opening price, closing price, and volume. It also has an animation feature, which enables viewers to watch how things have changed over time. This interactive map creates a multidimensional portrait of market trends by making it simple to visualize how different variables change over time.

Literature Review

Literature Review: Market Trends and Price Dynamics: It is necessary that investors, analysts, and policy makers understand price trends, trade volumes, and seasonal trends. Considerable study and professional work on market trend analysis has been done in the last few decades, especially in financial markets. This review of literature discusses major concepts and research evidence on market data analysis, such as visualization and interpretation of variables like opening prices, closing prices, trade volumes, and seasonaliy .

1. Market Trend Analysis and Price Dynamics

One of the most important areas of financial study is market trend analysis. A number of studies have stressed the value of examining the direction of asset price movements over time in order to forecast future performance and determine trading opportunities. Line graphs and scatter graphs, which show the relationship between opening and closing prices, are two of the major mechanisms used to study price trends. Stock market trends are influenced by psychological as well as economic fundamentals, as per Shiller's (2003) research. Finding out if a market is trending higher or downward can be eased by charting price movements through time on line plots.

Furthermore, there is enough evidence of the connection between market sentiment, closing prices, and opening prices. Gervais and Odean (2001) believe that psychological aspects like herd behavior and overconfidence can influence individual investors and result in market volatility. The correlation between these two variables, which commonly proxy market efficiency and investor sentiment, is more interpretable with scatter plots, such as the one showing the relationship between closing and opening prices.

2. Distribution of Market Data: Violin and KDE Plots:

It is useful to know the distribution of market data in order to identify outliers, central tendencies, and variation of price points. Violin plots and Kernel Density Estimation (KDE) plots are popular estimators of the distribution of market data and give information about the density of the data and identify regions of high concentration.

The understanding of the variability and dispersion of important variables, such as the adjusted closing price, becomes feasible with the use of violin plots. Violin plots give a clearer picture of the distribution of values by combining the advantages of density plots and box plots (Hintze & Nelson, 1998). The horizontal width shows density of values, or where the majority of the data points are. Whether a market variable is skewed or symmetric—something that might show underlying market conditions or investor behavior—is easier to determine with violin plots.

Besides, KDE plots are utilized to estimate the probability density of a continuous random variable, depict the distribution form, and identify patterns like multimodality, skewness, or uniformity. Based on Silverman (1986), KDE plots provide a natural and smoother characterization of the data than histograms do, which makes them particularly useful in market trend analysis. KDE plots are also commonly utilized in financial data analysis to spot high-activity regions or to find possible market inefficiencies.

3. Cumulative Distribution and Seasonal Trends

For financial markets, ECDF plots can identify trends in the markets like rising or falling market volatility or the persistence of specific price trends. A further critical way of graphically displaying data from markets is to utilize Empirical Cumulative Distribution Function (ECDF) plots which illustrate a variable's cumulative distribution, that is, the relative proportion of data points less than or equal to some value. ECDF plots are especially helpful for viewing the accumulation of values over time and allow analysts to compare the relative frequency of different outcomes.

Pie charts and count plots can be useful for showing the frequency distribution of categorical data like months or years and can give a good idea of when specific trends or habits are most common. Studies like Fama (1965) put forward the fact that seasonality must be noticed in the behavior of stock prices, where there could be an increase or a decrease in the trading volumes of some months (like the end of the accounting year). Seasonality is critical in the market activity pattern, and pie charts and count plots can be very helpful in representing market data distribution with regard to the various segments of time, whether months or years.

4. Company Performance and Trading Volume Analysis

Market liquidity and investors' interest could be interpreted using appropriate analysis of trading volumes, which could be graphed using bar plots by firm or time. Kyle (1985) has suggested that trading volumes typically are greater when market liquidity is greater, and low volumes might indicate inefficiency or higher risk. If the trading volumes are examined in combination with price change, one would be able to interpret the trend of the market and verify if price change is speculation-based or based on and Hall/CRC Press. https://doi.org/10.1201/9780429240963

Summary

Several visualizations are utilized in market data analysis to help in understanding the trends and inter-connections in the data set. Distribution and frequency of the variables were shown by violin plots, which also portrayed outliers and areas of concentration. Densities and patterns were highlighted by KDE plots, which presented a smooth assessment of the data distribution. The overall value distribution was depicted through ECDF plots that provided information on the global spread of data. The categories and distributions of categorical variables, such as month and year, were illustrated through pie charts and count plots.

The interactions between prominent price levels were revealed through the scatter plots of closing and opening prices, as well as high and low prices.

Also, the line plots revealed seasonal trends and market movements through the illustration of the movement of the market over both months and years. Though the interactive scatter plot allowed for users to examine the data based on company, the bar plot revealed the aggregate trade volume by several companies. Lastly, the 3D scatter plot provided an interactive visualization of the data by animating changes in opening price, closing price, and volumes over time.

Conclusion

The market data was clearly revealed through the visualizations, and they also displayed such crucial information as changes in seasonality, volume of trades, and price movement. The underlying distributions and fluctuations in the important market factors were revealed through the application of violin plots and KDE plots. While the interactive bar plot and bar plot detected company-specific trends and performance, line plots and scatter plots were interested in the price relationships and how these were changing over time. A dynamical element was provided by the 3D scatter plot, which enabled one to examine several variables over time. In total, these depictions provide a clearer picture of market activity and assist in the ability to discern trends, patterns, and areas for future study.

References

In [139…
```
market.to_csv("Cleaned.csv", index = False)
```


1. Davis, L. L. B. (2021). Axes3D — ProPlot documentation. Readthedocs.io. https://proplot.readthedocs.io/en/v0.7.0/api/proplot.axes.Axes3D.html
2. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., & Gérard-Marchant, P. (2020). Array Programming with NumPy. Nature, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2
3. Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90–95.
4. Mckinney, W. (2010). Data Structures for Statistical Computing in Python. https://pub.curvenote.com/01908378-3686-7168-a380-d82bbf21c799/public/mckinney-57fc0d4e8a08cd7f26a4b8bf468a71f4.pdf
5. Pedregosa, F., Buitinck, L., Louppe, G., Grisel, O., Varoquaux, G., & Mueller, A. (2015). Scikit-learn. GetMobile: Mobile Computing and Communications, 19(1), 29–33. https://doi.org/10.1145/2786984.2786995

6. Waskom, M. (2021). Seaborn: Statistical Data Visualization. Journal of Open Source Software, 6(60), 3021. https://doi.org/10.21105/joss.03021
7. Fama, E. F. (1965). The behavior of stock-market prices. The Journal of Business, 38(1), 34–105. https://doi.org/10.1086/294743
8. Gervais, R., & Odean, T. (2001). Learning to be overconfident. Review of Financial Studies, 14(1), 1–27. https://doi.org/10.1093/rfs/14.1.1
9. Hintze, J. L., & Nelson, R. D. (1998). Violin plots: A box plot-density trace synergism. The American Statistician, 52(2), 181–184. https://doi.org/10.1080/00031305.1998.10480556
10. Jensen, M. C. (1978). Some anomalous evidence regarding market efficiency. Journal of Financial Economics, 6(2–3), 95–101. https://doi.org/10.1016/0304-405X(78)90018-1
11. Kyle, A. S. (1985). Continuous auctions and insider trading. Econometrica, 53(6), 1315–1335. https://doi.org/10.2307/1913225
12. Shiller, R. J. (2003). From efficient markets theory to behavioral finance. The Journal of Economic Perspectives, 17(1), 83–104. https://doi.org/10.1257/089533003321164967
13. Silverman, B. W. (1986). Density estimation for statistics and data analysis. Chapman and Hall/CRC Pr

In [ ]: