

# MSIN0143 Coursework Group Q2

## Building an Analytical Notebook in Python

Predicting Taiwan Semiconductor Manufacturing Company Stock (NYSE: TSM) using historical stock data

21073058 Daniel Silalahi Insert SN Name Insert SN Name Insert SN Name

Word Count: XXXX

### ✓ Background Information

- TSMC Company
- Why TSMC: talk about use of AI, need for high-powered semiconductors
- More Context
- Why other companies stock prices also (Nvidia, S&P 500, Taiwan Index)

Since we need GenAI component, we can say that ChatGPT suggested Nvidia, s&p etc. for various reasons (e.g. partnership, overall economic indicators etc.)

```
1 # Downloading Necessary Libraries
2 import yfinance as yf
3 import pandas as pd
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import shap
7 from sklearn.preprocessing import MinMaxScaler
8 from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
9 from sklearn.model_selection import train_test_split
10 from sklearn.inspection import permutation_importance
11 from statsmodels.tsa.api import VAR
12 import seaborn as sns
13 from tensorflow.keras.models import Sequential
14 from tensorflow.keras.layers import LSTM, Dense
```

Here we import data from yfinance

It is important to keep most of the data, as some missing values in one stock may not mean that other stocks have those missing values.

```

1 # Import TSM (Taiwan Semiconductor Manufacturing Company), NVDA (Nvidia), TAIEX (Taiwan Stock Index), and .INX (US S&P 500 Index)
2
3 # Create a list of stock tickers
4 tickers = ["TSM", # Taiwan Semiconductor Manufacturing Company
5            "NVDA", # Nvidia
6            "^TWII", # Taiwan Stock Exchange Index
7            "^GSPC"] # S&P 500
8
9 # Fetch historical data from Yahoo Finance
10 data = yf.download(tickers, start='2014-12-01', end='2024-12-01')
11
12 # Align all stock data to closing time for consistency
13 data = data['Adj Close']
14 data = data.sort_index() # Sort by date
15 data.columns = ["TSM", "NVDA", "TAIEX", "S&P 500"] # Rename columns
16
17 # Show data head
18 data.head()

```

🔄 [\*\*\*\*\*100%\*\*\*\*\*] 4 of 4 completed

	TSM	NVDA	TAIEX	S&P 500
<b>Date</b>				
<b>2014-12-01 00:00:00+00:00</b>	0.493978	17.410347	2053.439941	9117.667969
<b>2014-12-02 00:00:00+00:00</b>	0.494698	17.129534	2066.550049	9034.749023
<b>2014-12-03 00:00:00+00:00</b>	0.507420	17.402758	2074.330078	9175.217773
<b>2014-12-04 00:00:00+00:00</b>	0.502859	17.319269	2071.919922	9225.068359
<b>2014-12-05 00:00:00+00:00</b>	0.505740	17.258553	2075.370117	9206.528320

It is important to keep most of the data, as some missing values in one stock may not mean that other stocks have those missing values.

Before we choose a method to interpolate missing data, we need to find out the average missing percentage and average time between missing values. For example, we could use a forward or linear fill to fill data if there is a low average time between missing values, meaning that the missing values are spread throughout. But a more complex method with moving averages may be needed if there are large gaps

```

1 # Checking for missing values
2 missing_values = data.isna().sum().sum()
3 total_values = data.size
4 missing_percentage = (missing_values / total_values) * 100
5 print(missing_percentage)
6
7 # Check for average number of days between missing values
8
9 # Firstly, identify missing values
10 missing_dates = data.isna()

```

```

11 # Create a list of missing gaps, where each value is the number of days between valid data
12 missing_gaps = []
13
14 # Iterate over dataset
15 i = 0
16 while i < len(data):
17     # Check if the current day is missing
18     if missing_dates.iloc[i].any():
19         missing_count = 1 # Start counting the missing day
20         # Iterate over the next days until we find a valid day
21         while i + 1 < len(data) and missing_dates.iloc[i + 1].any():
22             missing_count += 1
23             i += 1
24         # After finding a valid day, store the gap (number of missing days)
25         missing_gaps.append(missing_count)
26     i += 1
27
28 average_gap = sum(missing_gaps) / len(missing_gaps)
29 print(f"Average number of consecutive missing days: {average_gap}")

```

→ 3.8342967244701347  
Average number of consecutive missing days: 1.5316455696202531

Because average distance is only 1.53 days, we have chosen to move forward with linear fill, as it allows more insight than forward fill by bridging the missing gap between valid data

```

1 # Data Cleaning
2 # Interpolate missing values
3 data = data.interpolate(method='linear')

```

## ✓ Exploratory Data Analysis

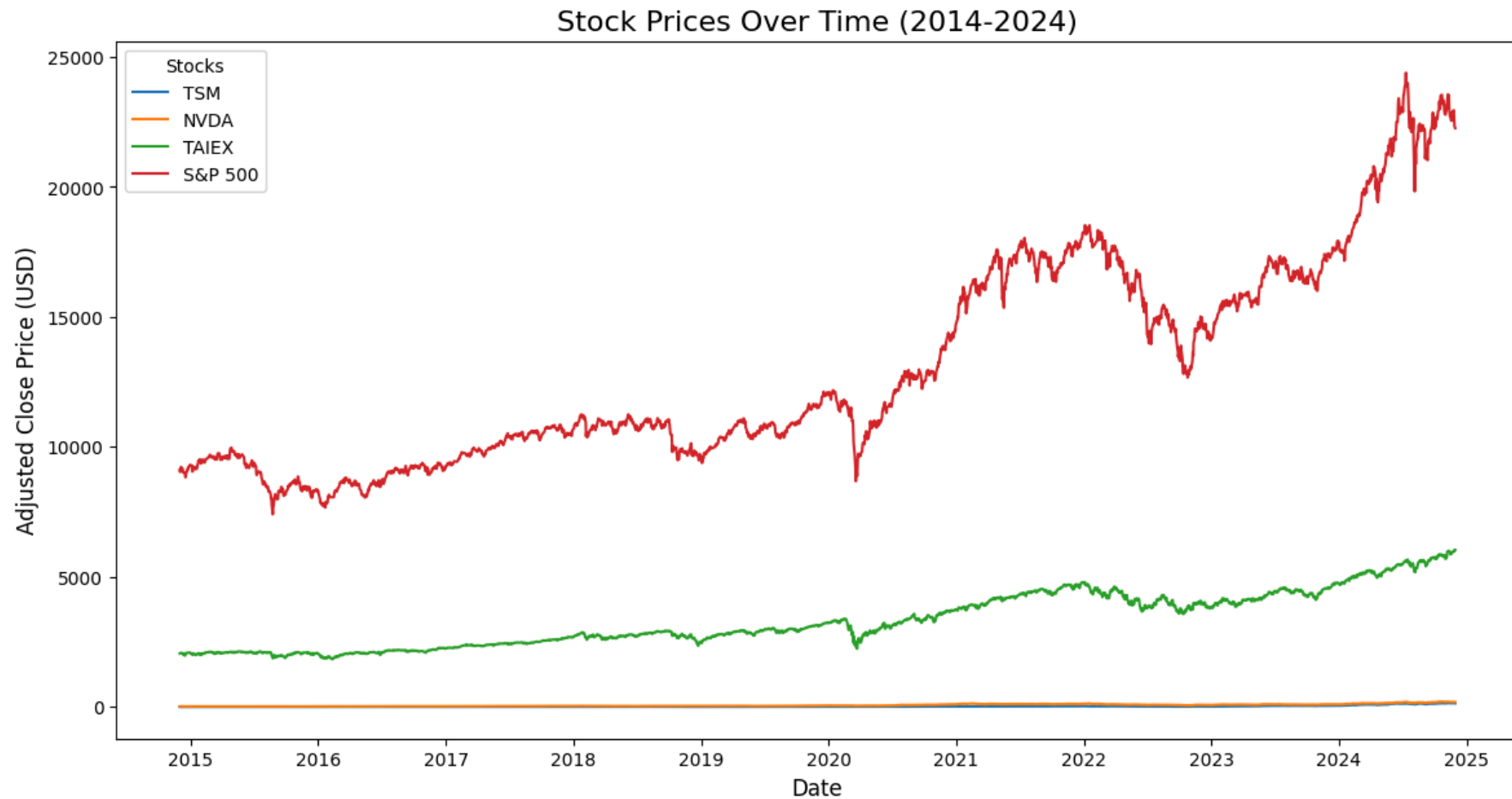
Firstly, we plot stock prices over time...cont...

```

1 # Plot the stock prices over time
2 plt.figure(figsize=(14, 7))
3
4 for column in data.columns:
5     plt.plot(data.index, data[column], label=column)
6
7 # Add title and labels
8 plt.title('Stock Prices Over Time (2014–2024)', fontsize=16)
9 plt.xlabel('Date', fontsize=12)
10 plt.ylabel('Adjusted Close Price (USD)', fontsize=12)

```

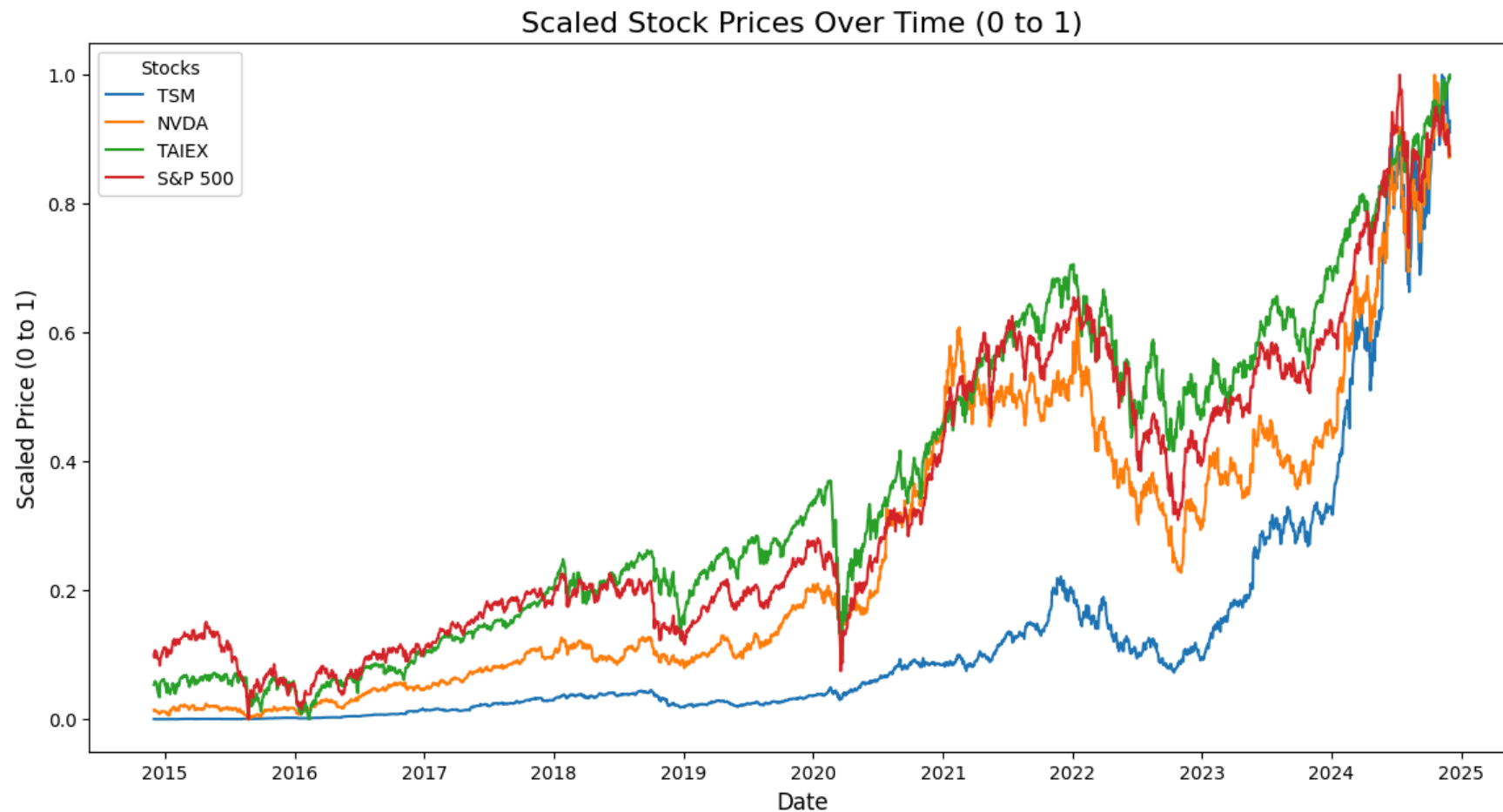
```
11
12 # Add legend and show plot
13 plt.legend(title="Stocks", fontsize=10)
14 plt.show()
```



However, we can see that very large difference, making it hard to view any relationships. Therefore, to improve legibility, we will use min-max scaling to normalise the data across columns. This will also make it easier for the model to gauge relationships across independent variables (counts as feature engineering), as well as allowing us to better analyse the coefficients of models later on.

```
1 # Initialise scaler
2 scaler = MinMaxScaler()
3
4 # Scale data, ensuring that we scale per column and not as a whole
```

```
5 scaled_data = scaler.fit_transform(data.values)
6
7 # Create a new scaled dataframe
8 scaled_df = pd.DataFrame(scaled_data, index=data.index, columns=data.columns)
9
10 # Create plot
11 plt.figure(figsize=(14, 7))
12
13 for column in scaled_df.columns:
14     plt.plot(scaled_df.index, scaled_df[column], label=column)
15
16 plt.title('Scaled Stock Prices Over Time (0 to 1)', fontsize=16)
17 plt.xlabel('Date', fontsize=12)
18 plt.ylabel('Scaled Price (0 to 1)', fontsize=12)
19
20 plt.legend(title="Stocks", fontsize=10)
21 plt.show()
```



Write description of data...better for model analysis as well as visualisations in terms of legibility as compared to percentage growth/change and coefficient analysis

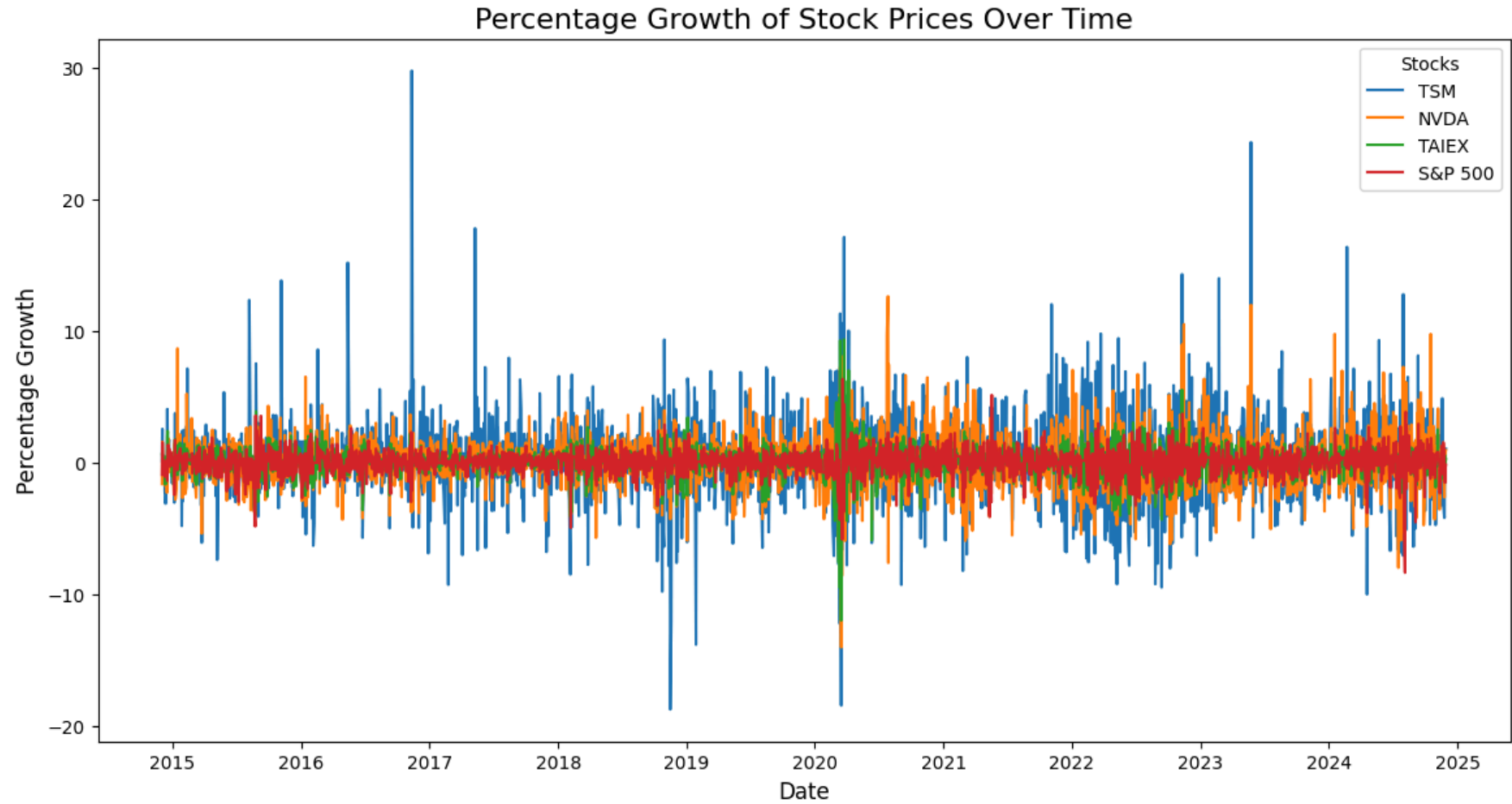
Furthermore, we can also conduct feature engineering on the scaled dataset by converting it into percentage growth to discover trends in growth over time

```
1 # Convert Scaled data into percentage data
2 growth_df = data.pct_change() * 100
3
4 # Drop NA values as first and last day will not have growth data
5 growth_df.dropna(inplace=True)
```

```
1 # Display the first few rows of the growth DataFrame
2 print(growth_df.head())
3
4 # Plot the percentage growth data
5 plt.figure(figsize=(14, 7))
6
7 for column in growth_df.columns:
8     plt.plot(growth_df.index, growth_df[column], label=column)
9
10 plt.title('Percentage Growth of Stock Prices Over Time', fontsize=16)
11 plt.xlabel('Date', fontsize=12)
12 plt.ylabel('Percentage Growth', fontsize=12)
13
14 plt.legend(title="Stocks", fontsize=10)
15 plt.show()
```



	TSM	NVDA	TAIEX	S&P 500
Date				
2014-12-02 00:00:00+00:00	0.145797	-1.612910	0.638446	-0.909432
2014-12-03 00:00:00+00:00	2.571612	1.595046	0.376474	1.554761
2014-12-04 00:00:00+00:00	-0.898804	-0.479743	-0.116190	0.543318
2014-12-05 00:00:00+00:00	0.572792	-0.350573	0.166522	-0.200975
2014-12-08 00:00:00+00:00	-1.281435	-1.055385	-0.725657	-0.209420



Description of graph above

```
1 # Summary of dataframe
2
3 # Generate summary statistics for the growth_df (percentage growth)
4 growth_summary = growth_df.describe()
5 print(growth_summary)
```

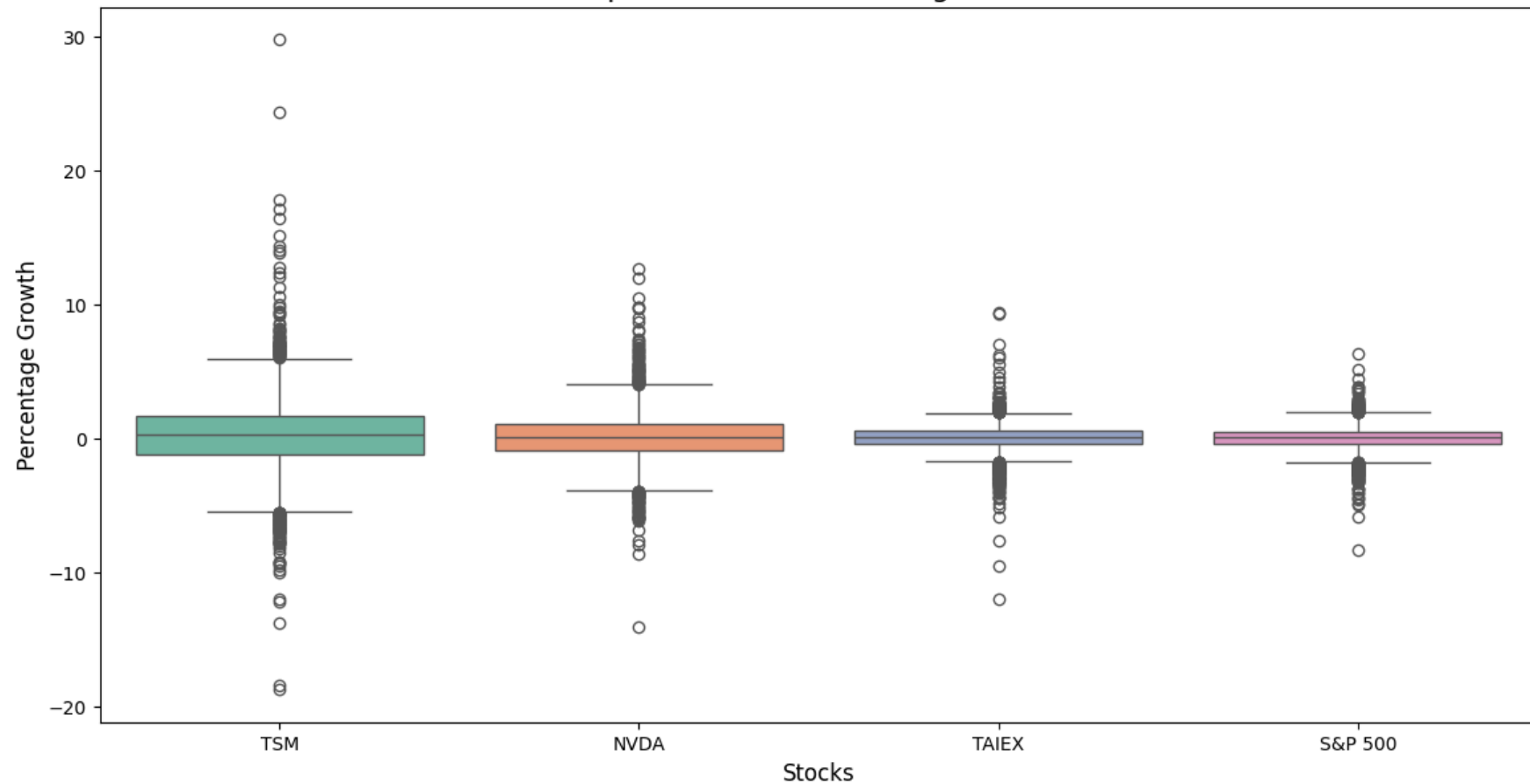


```
6
7 # Create a boxplot
8 plt.figure(figsize=(14, 7))
9
10 sns.boxplot(data=growth_df, palette='Set2')
11
12 plt.title('Boxplot of Stock Percentage Growth', fontsize=16)
13 plt.xlabel('Stocks', fontsize=12)
14 plt.ylabel('Percentage Growth', fontsize=12)
15
16 plt.show()
```



	TSM	NVDA	TAIEX	S&P 500
count	2594.000000	2594.000000	2594.000000	2594.000000
mean	0.261651	0.110122	0.047608	0.038967
std	2.990879	1.956378	1.098220	0.952059
min	-18.755883	-14.034062	-11.984055	-8.351980
25%	-1.183236	-0.925033	-0.358334	-0.402289
50%	0.253720	0.078642	0.054201	0.085508
75%	1.704492	1.068566	0.559971	0.538112
max	29.806735	12.652218	9.382774	6.367105

Boxplot of Stock Percentage Growth

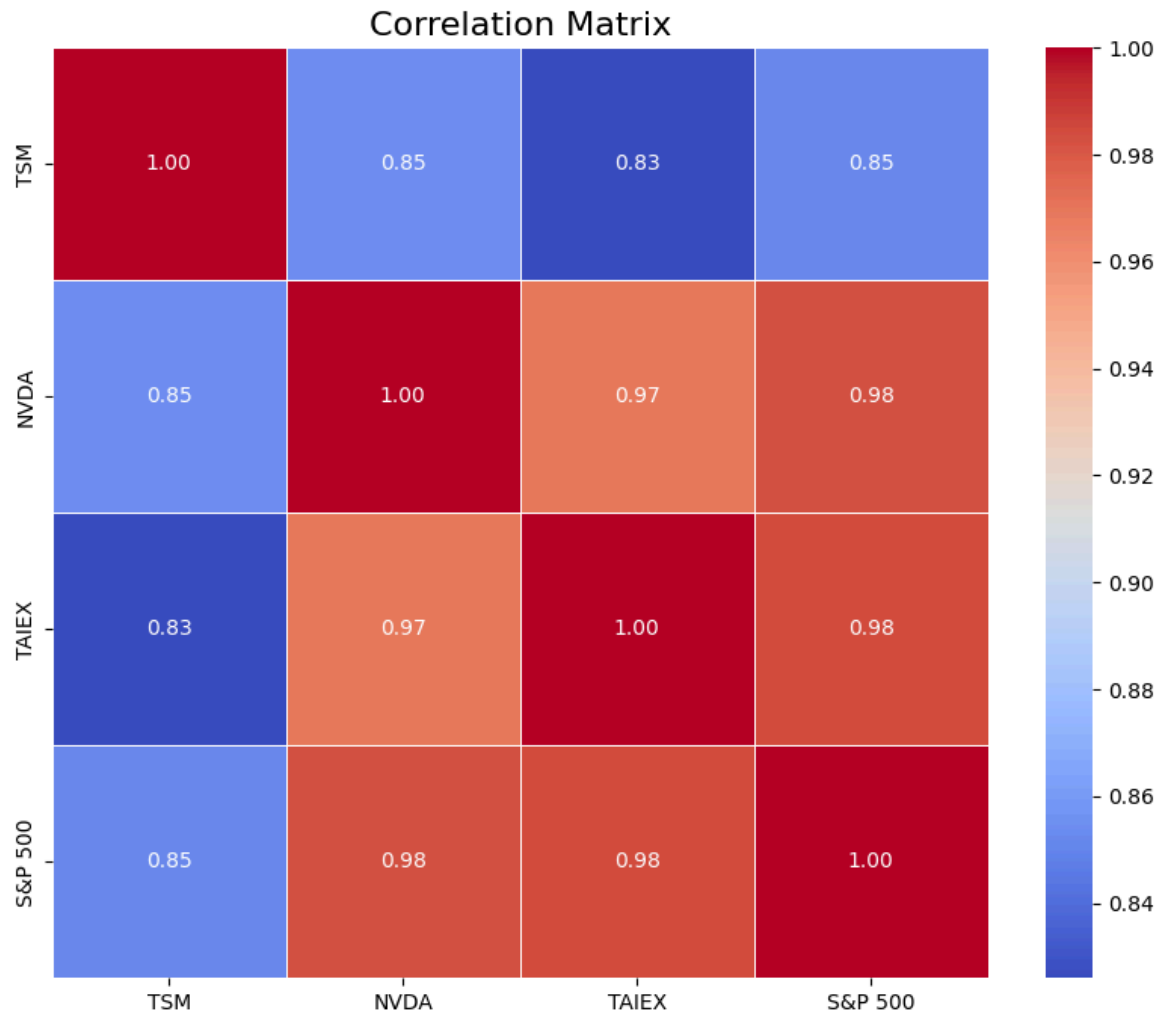


Explanation of summary statistics. Summary statistics of growth was chosen instead of the scaled data as scaled data always rises and there is no valuable statistical insight from this...

Description of boxplot. Explanation of why we wont remove outliers (Stock price is known to be very impacted by general public perception and other factors, wide swings in price are very common etc.)

Furthermore, we will create a correlation matrix to explore relationships between variables, check for multicollinearity, and provide insight for feature selection

```
1 # Calculate the correlation matrix
2 correlation_matrix = scaled_df.corr()
3
4 # Plot the correlation matrix as a heatmap
5 plt.figure(figsize=(10, 8))
6 sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
7 plt.title('Correlation Matrix', fontsize=16)
8 plt.show()
```



Explanation of Correlation matrix, we decide to keep NVDA, TAIEX, and S&P as they indicate a strong positive relationship.

Other variables exhibit multicollinearity, however (insert reason to keep them as stock prices move as a whole market usually etc)

## ✓ Model Selection and Training

We have chosen to move forward with LSTM and ARIMA model, as these are both time series models etc.

Training will also include TSM as a predictor, this is popular with stock prediction etc (include source)

Explain brief steps of model

- TSM, NVDA, S&P... used as predictor for future TSM results

Also include GenAI component here to determine models

Firstly, we need to create a single step sampler function

Sourced used for coding: <https://www.geeksforgeeks.org/multivariate-time-series-forecasting-with-lstms-in-keras/>  
<https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/>

```
1 def singleStepSampler(df, window):
2     xRes = []
3     yRes = []
4     for i in range(0, len(df) - window):
5         res = []
6         for j in range(0, window):
7             r = []
8             for col in df.columns:
9                 # Features for each column
10                r.append(df[col][i + j])
11            res.append(r)
12        xRes.append(res)
13        yRes.append(df['TSM'].iloc[i + window]) # Target is TSM
14    return np.array(xRes), np.array(yRes)
```

Below we split the data into test and training groups (insert source on why 85:15 split is good for stock data)

```
1 # Define the time window for LSTM
2 window_size = 20
3
4 # Prepare the dataset using the singleStepSampler function
5 xVal, yVal = singleStepSampler(scaled_df, window_size)
```

```

6
7 # Split the dataset using train_test_split
8 # Shuffle is set as false as it is a time series data
9 X_train, X_test, y_train, y_test = train_test_split(xVal, yVal, test_size=0.15, random_state=42, shuffle=False)

```

⚡ <ipython-input-25-928a42128651>:10: FutureWarning: Series.\_\_getitem\_\_ treating keys as positions is deprecated. In a future version, integer keys will always be treated as integer positions, selecting from obj.iloc. Please use its explicit method .append(df[col][i + j])

## Define and Train LSTM Model

```

1 # Define LSTM model using default parameters
2 multivariate_lstm = Sequential([
3     LSTM(128, activation='tanh', input_shape=(window_size, X_train.shape[2])),
4     Dense(1) # However, we need to tweak this paramater as we need it for our continuous data
5 ])
6
7 # Set at default values as per brief
8 multivariate_lstm.compile(optimizer='adam', loss='mean_squared_error')
9
10 history = multivariate_lstm.fit(X_train, y_train, epochs=1, batch_size=32, validation_split=0, verbose=1)
11
12 # Predict
13 predicted_values = multivariate_lstm.predict(X_test)
14
15 # Reload the data with the date index
16 scaled_df.index = pd.to_datetime(scaled_df.index)
17
18 # Create a dataframe of predictions and actuals
19 d = {
20     'Predicted Price': predicted_values.flatten(),
21     'Actual Price': y_test,
22 }
23 d = pd.DataFrame(d)
24 d.index = scaled_df.index[-len(y_test):] # Assigning the correct date index
25
26 # Convert scaled data back to actual prices
27
28 # Align y_test (Scaled TSM) indicies with actual TSM indicies to figure out ratio
29 actual_close_prices = data['TSM'].iloc[-len(y_test):].values
30 ratios = actual_close_prices / y_test
31
32 # Apply ratios to d
33 d['Predicted Price'] = d['Predicted Price'] * ratios
34 d['Actual Price'] = d['Actual Price'] * ratios

```

⚡ /usr/local/lib/python3.10/dist-packages/keras/src/layers/rnn/rnn.py:204: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using the `add` method, the input shape is inferred from the first batch. When using the `call` method, the input shape is inferred from the first argument. Please use the `input\_shape` argument to the `add` method or the `call` method. When using the `call` method, the input shape is inferred from the first argument. Please use the `input\_shape` argument to the `add` method or the `call` method.

```

69/69 ————— 4s 25ms/step - loss: 0.0010
13/13 ————— 1s 21ms/step

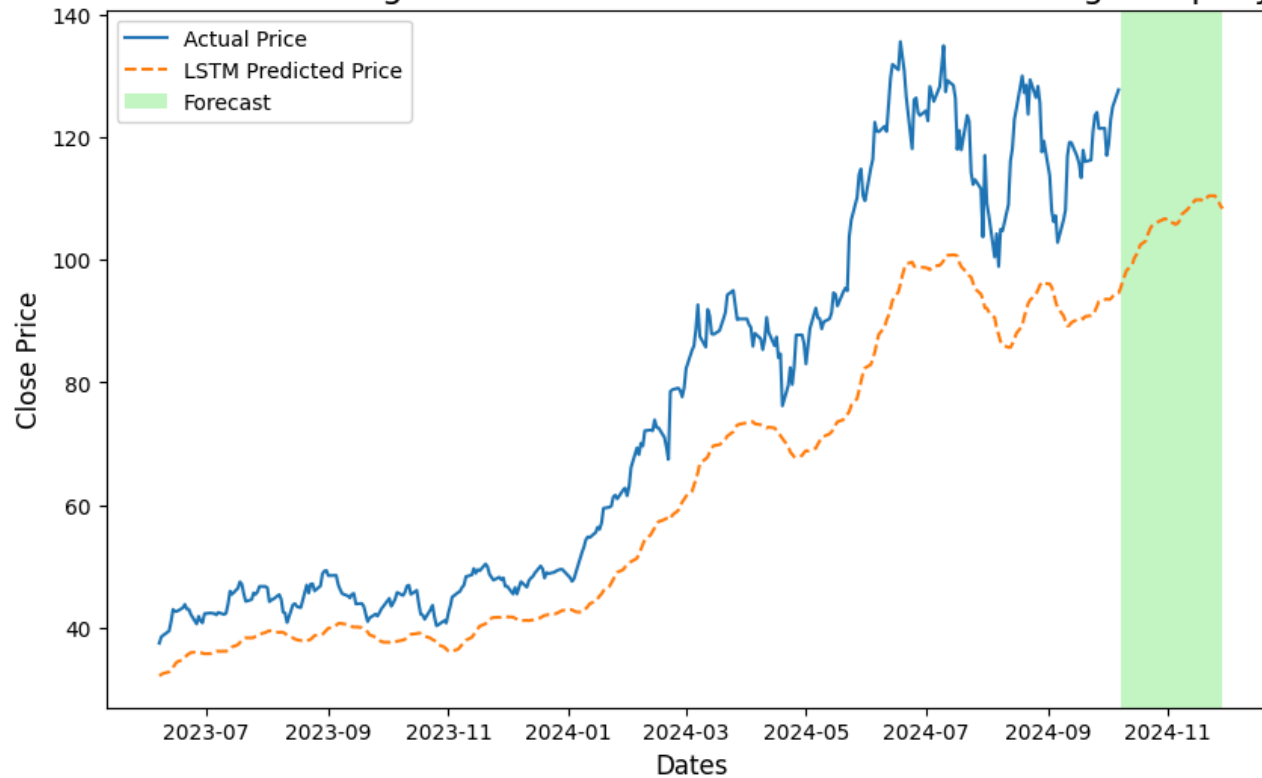
```

## Plotting LSTM Model

```
1 # Plot forecast with Dates on X-axis
2 fig, ax = plt.subplots(figsize=(10, 6))
3
4 highlight_start = int(len(d) * 0.9) # Highlight 10% for forecasting
5 highlight_end = len(d) - 1 # Adjusted to stay within bounds
6
7 # Plot the actual values
8 plt.plot(d['Actual Price'][:highlight_start], label='Actual Price')
9
10 # Plot predicted values with a dashed line
11 plt.plot(d['Predicted Price'], label='LSTM Predicted Price', linestyle='--')
12
13 # Highlight the forecasted portion with a different color
14 plt.axvspan(d.index[highlight_start], d.index[highlight_end], facecolor='lightgreen', alpha=0.5, label='Forecast')
15
16 plt.title('LSTM Forecasting for Taiwan Semiconductor Manufacturing Company', fontsize=16)
17 plt.xlabel('Dates', fontsize=12)
18 plt.ylabel('Close Price', fontsize=12)
19 ax.legend()
20 plt.show()
```



## LSTM Forecasting for Taiwan Semiconductor Manufacturing Company



Interpretation of model. Talk about how we rescaled the data for better legibility (dont worry about the model looking bad, it is because as per coursework brief, we need to use default parameters)

Multivariate ARIMA model (Vector Autoregression)

Code Sourced from : <https://www.geeksforgeeks.org/python-arima-model-for-time-series-forecasting/> <https://www.geeksforgeeks.org/vector-autoregression-var-for-multivariate-time-series/> <https://www.cloudzilla.ai/dev-education/multivariate-time-series-using-auto-arma/#understanding-the-arma-model> [https://www.analyticsvidhya.com/blog/2018/09/multivariate-time-series-guide-forecasting-modeling-python-codes/#Dealing\\_With\\_a\\_Multivariate\\_Time\\_Series\\_%E2%80%93VAR](https://www.analyticsvidhya.com/blog/2018/09/multivariate-time-series-guide-forecasting-modeling-python-codes/#Dealing_With_a_Multivariate_Time_Series_%E2%80%93VAR)

Splitting into training and test sets as well as scaling as VAR needs stationary data

```
1 data_diff = scaled_df.diff()
2
3 # Splitting
```

```

4 train_size = int(len(scaled_df) * 0.85) # 85% for training, 15% for testin
5 train_data, test_data = scaled_df[:train_size], scaled_df[train_size:]


```

### Define and train ARIMA model

```

1 # Define VAR model with default parameters
2 model = VAR(train_data)
3 # Fit the VAR model using default parameters
4 results = model.fit()
5
6 # Predict using the fitted model
7 predicted = results.forecast(train_data.values[-results.k_ar:], steps=len(test_data))
8
9 # Convert forecast to DataFrame
10 predicted_df = pd.DataFrame(predicted, index=test_data.index, columns=test_data.columns)
11
12 # Prepare data for plotting (comparing predicted vs actual)
13 d_var = {
14     'Predicted Price': predicted_df['TSM'], # Using 'TSM' as the variable to forecast
15     'Actual Price': test_data['TSM']
16 }
17
18 d_var = pd.DataFrame(d_var)
19 d_var.index = test_data.index # Ensure correct Date index
20
21 # Reverse the differencing
22 d_var['Predicted Price'] = np.cumsum(d_var['Predicted Price'].values, axis=0)
23 d_var['Actual Price'] = np.cumsum(d_var['Actual Price'].values, axis=0)
24
25 # Rescale the data (as data is scaled) as previously done
26 actual_prices = data['TSM'].iloc[-len(d_var):].values
27 ratios = actual_prices / d_var['Actual Price'].values
28
29 # Apply ratios to d
30 d_var['Predicted Price'] = d_var['Predicted Price'] * ratios
31 d_var['Actual Price'] = d_var['Actual Price'] * ratios

```

 /usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa\_model.py:473: ValueWarning: A date index has been provided, but it has no associated frequency. self.\_init\_dates(dates, freq)

### Plot ARIMA/VAR