Group - Room 6

Compiled by:
smck0583 510678236
kong7456 500459007
jdor8613 500490723
mlim2731 500308222

Section 1

Topic:

The topic in question for this analysis is to observe multiple stock exchanges from different countries and thoroughly examine their performance over the past 3 years (2019 to 2021). The stock exchanges chosen for this analysis are:

- The Nasdaq Stock Market (Nasdaq) from the United States
- Financial Times Stock Exchange 100 Index (FTSE100) from the United Kingdom
- Australian Securities Exchange (ASX) from Australia
- Brazil Stock Exchange Index (Bovespa) from Brazil

These respective markets were each chosen for their rich data and diverse values throughout the years with both Nasdaq and FTSE100 ranking in the top 10 stock exchanges in the world by market capitalization. The ASX and Bovespa were also chosen to provide an insight into the economic performance of markets in countries within the Southern Hemisphere, with Bovespa being an example of a stock market not belonging to an Anglospheric country.

This topic was chosen in regards to not only the personal interests of the respective group members but to discover and engage with the financial world through the lens of data analytics. Within our contemporary digital era, data analytics is used more than ever before as firms, governments and researchers begin to recognise and utilise its capabilities in observing and forecasting both trends and correlations within our data rich society. The examination of this topic provides an example and insight into the methods of financial data analysis which would be utilised by data scientists to discover patterns for various firms, thus granting greater awareness into future career experiences catering towards financial data exploration.

By using custom designed programs via the coding language Python as well as unlocking the great wealth of tools and functions of analysis in Microsoft Excel, datasets from the above stock markets were analysed and integrated with the aim of spotting correlations or distinctions across the financial performance of the 4 countries' markets from the beginning of 2019 to the end of the financial year (June 30).

Stakeholder Discussion

The topic in question not only discusses critical changes within internationally vital stock markets during a time of incredible uncertainty but also attempts to reveal potential safe investment opportunities, both of which could be incredibly valuable to a plethora of stakeholder groups. In particular, the results of our report may be relevant to distinct groups such as, governments, central banks, investment and commercial banks, commercial traders and retail traders.

The proposed question aims at gauging the impact of the Covid-19 Pandemic on certain continent leading stock markets (FTSE100, ASX200, BOVESPA, NASDAQ). The results produced would give stakeholders nuanced insight into how the different economies around the world have responded to the pandemic. Thereby, the observations from the summarized data are a proxy for characteristics such as stability, robustness and profitability of the renowned stock markets and the leading businesses featured within them during the tumultuousness of the past 3 years. More specifically, this would lead governments and central banks (such as the United States government and the Federal Reserve) to better understand their economies to help inform policy making. Additionally, it would also aid international banking bodies (such as J.P. Morgan, Morgan Stanley and Goldman Sachs) inform their investment strategies, identify international economic concerns whilst also better understanding which countries and continents have the most security during times of economic downturn. Finally, it could help traders (within hedge funds such as Blackrock, or simply everyday retail traders) make informed investment decisions to minimize exposure to risk.

Discussion of Data

The data collected is largely numerical in nature however, the featured attributes depict the movement of each stock index over the past 2 and a half years allowing for valuable summaries to be created. Specifically, using the collected data, and with automated Python programs, summaries may be made to deduce the growth, variance, and general stability of the index price over the past two and a half years. The data sets and their values can be summarized to form a variety of insights, for instance, the daily percentage change of each index price, the residual between the maximum and minimum index prices in a given period, the number of positive daily growth periods, the standard deviation of the index prices and many more.

Inherently these summaries are limited but, when each summary is compared against one another, the values may be compared to at least partially reveal the robustness and volatility present within each index in the mentioned time period. Thus, allowing for stakeholders to compare each index and this may form an understanding of the health of each country/region compared to one another. Not only would that aid investors and traders to adjust their investment strategies towards different businesses and regions that may be safer investment options during the current economic downturn. Additionally, it may allow governments and central banks to compare their countries stock market performance as a way to potentially indicate a need to change economic policy given disproportionately worse performance (for instance).

Within our current digital age, data analytics forms an essential aspect within every frame of each industry, such as entertainment, science, literature, sports and most importantly, finance. Financial analysis and stock analysis utilises a great amount of data analysis in discovering present correlations within various stock exchanges across the world whilst also contributing towards predicting future trends of prices (such as opening and closing price of each trading day) and precise stock volumes.

With the current COVID-19 pandemic, stakeholders such as governments and independent investors are in disarray over the uncertainty of the pandemic and its impacts on the financial stock market. This analysis unearths the question of whether COVID-19 has had a significant impact upon the financial markets of various countries and how exactly investors would overcome respective impacts in regaining their profits through the financial market. The topic also gives insight into if developed countries such as the United States and Australia are financially stronger and more resilient towards economic crises over developing countries such as Brazil. This explicitly represents the relevance of the issue within our current day and age as the pandemic continues in making its mark across the globe and more investors are leaning towards integrating data analysis in forming decisions over stock exchanges.

Section 2a - FTSE 100 (510678236)

Metadata:

The FTSE100 Stock Index itself, is an index that measures the cumulative performance of the top 100 companies listed on the London Stock Exchange with the largest market capitalization, for this reason the FTSE index is often used to observe the health of the United Kingdom's economy.

To study the FTSE100 index, a data set entitled “Historicaldata.csv” pertaining to the FTSE100 stock index was downloaded from The Wall Street Journal website on the 6th of September 2021 as a text file in comma separated value format (with data values downloaded from the first of January 2019 till the thirtieth of June 2021). Within the data set there are 641 rows and 5 columns. The columns of the file feature the schema of the data which consists of 5 attributes including “date”, “open”, “high”, “low”, “close”, the rows represent a new trading day (of which there are 5 per week excluding national holidays). The data dictionary that describes that format further may be seen below:

Schema Attributes	**Data Type**	**Data Format**	**Explanation**	**Example**
“date”	Ordinal	American Calendar Date Format (month/ date/ year) (with the year expressed in 2 digits)	The “date” attribute refers to the day in which the proceeding attributes were observed, the calendar date itself is expressed in the American format.	1/1/19
“open”	Numeric	Pounds sterling with 2 decimal places	The “open” attribute refers to the first recorded value of the index on that day of trading when the market opened.	2342.90
“high”	Numeric	Pounds sterling with 2 decimal places	The “high” attribute refers to the maximum recorded value of the index on that day.	7843.32
“low”	Numeric	Pounds sterling with 2 decimal places	The “low” attribute refers to the minimum recorded value of the index on that day.	4890.34
“close	Numeric	Pounds sterling with 2 decimal places	Finally, the “close” attribute refers to the final recorded value of the index for that particular	7423.45

			day of trading when the market closed.	

Provenance:

The data was originally created by FTSE Russel, a subsidiary of the London Stock Exchange Group who collected the data pertaining to the stock prices of the FTSE100 companies. The data was then acquired by an intermediary, Factset, who condensed it into a long format with 5 columns and each row representing a different trading day; after this process the data was then purchased by The Wall Street Journal. Finally, the data was displayed in table format on the WSJ Markets website with the additional option of downloading the historical data over a given time period. Under the “Notes and Data Providers” subsection of the website, it specifies the ability for users to freely use the publicly available data (including the FTSE100 historical data), provided on the WSJ Markets website. However, the regulations do specify that the data is only intended for informational uses not for trading and as such WSJ and Factset are not responsible for errors or omissions of the data that may impact the accuracy of the data. As such, the data was then downloaded as a comma separated value text file with data from the 1st of January 2019 till the 6th of September 2021.

Data Evaluation (Strengths & Weaknesses):

The data collected from the WSJ Markets website has clear strengths and weaknesses when attempting to answer the topic proposed in this report. Firstly, and to outline the strengths, the agreed upon time period for analysis of the stock markets is from the beginning of the 2019 financial year till half way through the financial year of 2021. The original WSJ historical data csv contained 642 different days worth of daily data of the FTSE100 index over the specified time frame suggesting that the data set contains all of the information needed to produce accurate summaries of the FTSE index which will reduce any bias or ambiguity when comparing each index. Additionally, the numeric data present within the data set is all standardized in both units and format such that each value is expressed in Pounds Sterling to at least 4 significant figures. The precision and standardization of the data will reduce any additional cleaning required while also ensuring a high degree of accuracy in the summaries produced. Furthermore, the 4 different attributes included within the data set increase the number of summaries and analysis that can be created to better characterize the growth, variance and movement of the FTSE 100 index price. By doing so this can help in the number of ways each data set can be compared, particularly when considering the “high” and “low” columns which may help depict the FTSE 100 index’s daily variance (for example). These comparisons will provide a larger basis within which conclusions about the financial security of each country may be deduced for stakeholders.

Conversely, there are weaknesses with the data set. Firstly, the date is expressed in American format, which in of itself is negligible, however since the data will be integrated with other data sets that do not follow the same format, the format must be changed before integration. Only through changing the format may the indexes be appropriately matched by date and then analysis may take place. Additionally, the closing price which will be heavily relied upon in

producing the summaries of the data are not adjusted. This means that the original data set features a closing price that isn't adjusted for any corporate actions and hence less accurately reflects the inherent value of the index, which may limit the conclusiveness of the summaries when compared to other indexes. Finally, the data was initially downloaded with the data sorted from end value to the beginning value (z to a). However, the other data sets collected were in the opposite order and thus, like the date format, in order to be able to integrate the data sets the order of the data must be reversed and only by doing so may the cleaned data sets be compared to one another.

Testing Data Quality:

In order to test for data quality, the data set must be checked for the format of the data but also the values themselves in comparison to the other data sets to ensure that integration can occur, and also no data value can be missing for any given row in the data set.

To test for any missing values, a python program may be made that checks for the number of days on record for each year. In any given year there should be on average 253 trading days (approximately five trading days per week excluding public holidays). Given that the original data set is supposed to include data from all of 2019, all of 2020 and stop half way through 2021 the number of trading days should be 253 for 2019 and 2020, and 126 or 127 for 2021. Using the following code, the number of days on record may be computed:

```
file = open("FTSE100 (1) (1) (1) (2).csv")

year_21 = 0
year_20 = 0
year_19 = 0

i = 0
for trading_day in file:
    if i > 1:
        trading_day = trading_day.strip("\n")
        trading_day = trading_day.split(",")
        date = trading_day[0]
        if date.endswith("21"):
            year_21 += 1
        if date.endswith("20"):
            year_20 += 1
        if date.endswith("19"):
            year_19 += 1
    i += 1
print(year_21)
print(year_20)
print(year_19)
```

With corresponding output in the terminal:

126
257
257

These results are positive, though they exceed the expected number of days recorded it would suggest that there are sufficient days to make an informed analysis. In the case that there would be missing days, it would then be necessary to inform the group before integration to ensure that the number of recorded days between each data set is standardized such that the data sets are comparable.

Next, to test if any of the data values are missing the following program may be run:

```
import pandas as pd

data = pd.read_csv("FTSE100 (1) (1) (1) (2).csv")
missing_values = data.isnull().sum()
print(missing_values)
```

With corresponding output in the terminal:

```
Date      0
 Open     0
 High     0
 Low      0
 Close    0
dtype: int64
```

This code shows that there are no missing values in any of the 5 attributes across the 641 rows. Therefore, no amendments or additions need to be made to missing data values. However, in the case that values were missing, the date can be readily amended given the date before and after the missing date value, the opening or closing value can be easily inputted since they would correspond to the closing value the day before or the opening value the day after respectively. If either a high or low value would be missing, there are no other data values that would indicate its accurate value and any guesses would be baseless therefore, in the case that either a high or low value is missing the inputted value would be based on what that days non-missing high or low value is.

One final data quality test may be programmed to check if the data values within the data set are within a reasonable range. The following program checks to see if any of the data values exceed the range of 4000 pounds to 8000 pounds (chosen based on the observed maximum and minimums of the data) and will return true if any of the values present exceed the range:

```
file = open("FTSE100 (1) (1) (1) (2).csv")

bool_list = [ ]

i = 0
```

```
for trading_day in file:
    trading_day = trading_day.strip("\n")
    trading_day = trading_day.split(",")
    if i > 0:
        open_td = float(trading_day[1])
        high = float(trading_day[2])
        low = float(trading_day[3])
        close = float(trading_day[4])
        value1 = (4000.0 < open_td < 8000.0)
        value2 = (4000.0 < high < 8000.0)
        value3 = (4000.0 < low < 8000.0)
        value4 = (4000.0 < close < 8000.0)
        bool_list.append(value1)
        bool_list.append(value2)
        bool_list.append(value3)
        bool_list.append(value4)
    i += 1

print(not any(bool_list))
```

Which prints to the terminal the following:

False

Therefore, from this program it may be concluded that the values of the data set are within the reasonable range based on the context of the historical FTSE performance and thus no values need to be changed. In the event that a value exceeded the range the value can either be replaced given the specifications in the previous program explanation or if after an attempt the value cannot be reasonably deduced the row may be deleted for the data set along with the corresponding column in the other data sets collected.

However, one problem was previously identified from observation, that was namely the incorrect date format. In order to change the date format such that the data sets may be integrated, the following python program was used:

```
i = 0

file = open("FTSE100 (1) (1) (1) (2).csv")

for trading_day in file:
    if i == 0:
        new_trading_day = trading_day.rstrip("\n")
        values = new_trading_day.split(",")
        start_values = ",".join(values)

    if i > 0:
        trading_day = trading_day.rstrip('\n')
```

```
        trading_day_elements = trading_day.rsplit(',')
        date_ls = list(trading_day_elements[0])
        if len(date_ls) < 7:
            order = [2,1,0,3,4,5]
            date_ls = [date_ls[i] for i in order]
        elif len(date_ls) < 8:
            order = [2,3,1,0,4,5,6]
            date_ls = [date_ls[i] for i in order]
        elif len(date_ls) == 8:
            order = [3,4,2,0,1,5,6,7]
            date_ls = [date_ls[i] for i in order]
        date_ls = ",".join(date_ls)
        date_ls = date_ls.replace(",", "")
        trading_day_elements[0] = date_ls
        trading_day = ",".join(trading_day_elements)
        with open('save.csv', 'a') as file:
            if i == 1:
                file.write(start_values)
                file.write(f"\n{trading_day}")
    i += 1
```

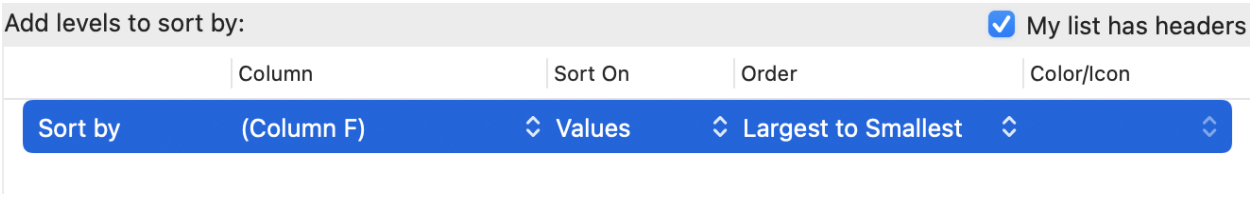
With the corresponding output to a new “save.csv” file:

```
Date, Open, High, Low, Close
1/1/19,6728.13,6752.54,6718.32,6728.13
2/1/19,6728.13,6753.29,6599.48,6734.23
3/1/19,6734.23,6753.14,6685.09,6692.66
4/1/19,6692.66,6850.37,6692.5,6837.42
7/1/19,6837.42,6874.11,6778.01,6810.88
.
.
.
```

It was previously identified that the order of the data set did not fit with the rest of the data sets. Therefore, to correct for this, excel was used to flip the order of the data. First a new unnamed column was amended to the end of the data set that is a series incrementing by 1 each row:

Date	Open	High	Low	Close	
06/30/21	7087.55	7100.71	7010.57	7037.47	1
06/29/21	7072.97	7120.77	7071.86	7087.55	2
06/28/21	7136.07	7136.22	7071.95	7072.97	3
06/25/21	7109.97	7139.08	7109.79	7136.07	4
06/24/21	7074.06	7119.41	7070.9	7109.97	5
06/23/21	7090.01	7129.4	7074.06	7074.06	6
06/22/21	7062.29	7098.67	7061.13	7090.01	7
06/21/21	7017.47	7065.05	6948.63	7062.29	8

Once the new column was created the data set can then be ordered using the “sort” button on excel:

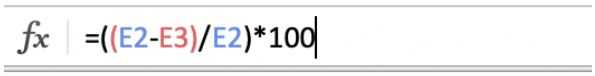


Then one may specify the order in which the data is sorted (with this case being largest to smallest to reverse the order). Which produces the following:

Date	Open	High	Low	Close	
1/1/19	6728.13	6752.54	6718.32	6728.13	641
1/2/19	6728.13	6753.29	6599.48	6734.23	640
1/3/19	6734.23	6753.14	6685.09	6692.66	639
1/4/19	6692.66	6850.37	6692.5	6837.42	638
1/7/19	6837.42	6874.11	6778.01	6810.88	637
1/8/19	6810.88	6901.48	6804.89	6861.6	636
1/9/19	6861.6	6938.31	6861.6	6906.63	635
1/10/19	6906.63	6948.18	6860.82	6942.87	634
1/11/19	6942.87	7001.94	6902.65	6918.18	633

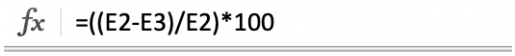
After these data cleaning procedures the resulting data set is ready for analysis and then eventual integration with the other data sets.

Finally, the following excel operations create the percentage change column for the data set in order to match the same columns as the rest of the data sets. Firstly, the formula for the percentage change of the closing FTSE price is created within the first possible row in the data set:



High	Low	Close	
6752.54	6718.32	6728.13	
6753.29	6599.48	6734.23	
6753.14	6685.09	6692.66	
6850.37	6692.5	6837.42	
6874.11	6778.01	6810.88	

When this is complete the bottom right corner of the cell is then double clicked to apply the formula for all of the rest of the rows, resulting in:



High	Low	Close	% `Change
6752.54	6718.32	6728.13	0
6753.29	6599.48	6734.23	-0.0906641
6753.14	6685.09	6692.66	0.61729403
6850.37	6692.5	6837.42	-2.1629666
6874.11	6778.01	6810.88	0.38815811
6901.48	6804.89	6861.6	-0.7446908
6938.31	6861.6	6906.63	-0.6562609
6948.18	6860.82	6942.87	-0.5247132

Analysis:

The following section details three data summaries including a grouped aggregate summary which help describe the characteristics of the data set. Firstly, one of the best indicators of performance of a stock over a certain period is hence, the following summary computes the highest closing price of the FTSE100 index for 2019, 2020 and half of 2021. First to iterate over the each row in a data set an iterating variable i is assigned along with the initial maximum closing price that will soon be compared to the closing price present in each row:

```
i = 0
max_closing_price19 = 0.0
max_closing_price20 = 0.0
max_closing_price21 = 0.0

file= open("FTSE100 (1) (1) (1) (2).csv")

for trading_day in file:
    if i > 0:
        trading_day = trading_day.split(",")
        date = trading_day[0]
        date_numbers = date.split("/")
        if date_numbers[2] == "19":
            max_closing_price19 = max(float(trading_day[4]), max_closing_price19)

        if date_numbers[2] == "20":
            max_closing_price20 = max(float(trading_day[4]), max_closing_price20)

        if date_numbers[2] == "21":
            max_closing_price21 = max(float(trading_day[4]), max_closing_price21)

    i += 1
print("The maximum closing price of The FTSE100 in 2019 was:
{}".format(max_closing_price19))
print("The maximum closing price of The FTSE100 in 2020 was:
{}".format(max_closing_price20))
print("The maximum closing price of The FTSE100 in 2021 was:
{}".format(max_closing_price21))
```

The following was printed into the terminal:

```
The maximum closing price of The FTSE100 in 2019 was: 7686.61
The maximum closing price of The FTSE100 in 2020 was: 7674.56
The maximum closing price of The FTSE100 in 2021 was: 7184.95
```

The code then compares the current maximum price to the closing price of each row and then assigns the new maximum to the maximum between those two values. This result is then computed for each of the three years finally printing the maximum values in the terminal.

The following data summary computes the percentage change of the starting value compared to the ending value of the FTSE100 stock index from the 1st of January 2019 till the 30th of June 2021:

```
i = 0

f = open("FTSE100 (1) (1) (1) (2).csv")

for trading_day in f:
    trading_day = trading_day.strip("\n")
    trading_day = trading_day.split(",")
    if i == 1:
        starting_value = float(trading_day[4])
    elif i == 641:
        ending_value = float(trading_day[4])
    i += 1
percent_change = ((ending_value - starting_value)/ending_value)*100
string_pc = str(("{:.2f}".format(percent_change)))
print("The percentage change of the closing price of the FTSE100 Index from 2019 till present
is: {}%".format(string_pc))
```

The following is printed into the terminal:

The percentage change of the closing price of the FTSE100 Index from 2019 till present is: 4.40%

The code assigns a value to the first recorded close price and the final recorded price and then computes the percentage change between them and finally reports it.

The following code is used to compute the difference between the maximum and minimum closing price for each year:

```
year_dictionary = {}
minimum_21= 20000
maximum_21 = 0
minimum_19= 20000
maximum_19 = 0
minimum_20= 20000
maximum_20 = 0

f = open("FTSE100 (1) (1) (1) (2).csv")

i = 0
```

```
for trading_day in f:
    if i > 1:
        trading_day = trading_day.strip("\n")
        trading_day = trading_day.split(",")
        date = trading_day[0]
        if date.endswith("21"):
            year = 2021
        if date.endswith("20"):
            year = 2020
        if date.endswith("19"):
            year = 2019
        closing_price = float(trading_day[4])
        if year == 2021:
            minimum_21 = min(closing_price, minimum_21)
            maximum_21 = max(closing_price, maximum_21)
        if year == 2020:
            minimum_20 = min(closing_price, minimum_20)
            maximum_20 = max(closing_price, maximum_20)
        if year == 2019:
            minimum_19 = min(closing_price, minimum_19)
            maximum_19 = max(closing_price, maximum_19)
    i += 1
if 2019 not in year_dictionary:
    year_dictionary[2019] = (maximum_19 - minimum_19)
if 2020 not in year_dictionary:
    year_dictionary[2020] = (maximum_20 - minimum_20)
if 2021 not in year_dictionary:
    year_dictionary[2021] = (maximum_21 - minimum_21)

for key in sorted(year_dictionary):
    print("The residual between maximum closing price and minimum closing price in {} is:
{:.2f}".format(str(key), year_dictionary[key]))
```

This code prints the following into the terminal:

The residual between maximum closing price and minimum closing price in 2019 is: 993.95
The residual between maximum closing price and minimum closing price in 2020 is: 2680.67
The residual between maximum closing price and minimum closing price in 2021 is: 777.4

Similar to the first method the program finds the maximum and minimum closing prices in each year then it proceeds to find the difference between the two and store those values in a dictionary. The dictionary is then used to call the years as keys to print the results.

Section 2B - Nasdaq (500459007)

The dataset ^IXIC.csv was used to analyse the Nasdaq stock exchange. It is a text file under the comma-separated value (csv) file format. This dataset was downloaded from *Yahoo! Finance* on the 8th of September, 2021 and contains 630 rows with 7 attributes and it begins from values on the 2nd of February 2019 to the 30th of June 2021.

Attribute	**Meaning**	**Data Type**
Date	Date of trading day	Ordinal in the format date/month/year (DD/MM/YY)
Open	Opening price of each trading day	Numeric under the United States Dollar (USD), rounded to 6 decimal places (6 dp)
High	Highest price at which a stock was traded on that day	Numeric (USD) (6 dp)
Low	Lowest price at which a stock was traded on that day	Numeric (USD) (6 dp)
Close	Closing price of each trading day	Numeric (USD) (6 dp)
Adjusted	Closing price of stocks with respect to other attributes	Numeric (USD) (6 dp)
Volume	Number of shares traded with a stock	Numeric

The data contained in the dataset analysed was originally collected and provided for by ICE Data Services to *Yahoo! Finance* in real time. ICE Data Services, a subsidiary of Intercontinental Exchange, is an American financial services company which operates to provide financial market data towards individual investors and active traders as well as data sharing/providing firms. The data provided is available for free publicly and contains at least medium volume.

Upon viewing the dataset, some strengths and limitations were found. Strengths included the rounding of the price data and values to 6 decimal places, providing accurate and precise information towards the numeric values with no need for further cleaning or extrapolation. The date format was also in the common form of date-month-year (DD/MM/YY), hence no change of format was necessary with the date values in the dataset, unlike the remaining stock exchanges analysed. The dataset also contained an additional ‘adjusted close’ column pertaining to the adjusted closing price of stocks with consideration of other attributes and corporate actions such as dividends or rights offerings. This column provides individual investors and firms alike to understand and comprehend an accurate representation of the stock’s performance. Another strength of the dataset includes the real-time collection of the values in the dataset from ICE Data Services, leaving no ambiguity behind in the validity of the values recorded. Limitations of the dataset includes a lack of information indicating the country’s economic status and current

performance and a lack of consideration of current worldwide economic events such as the COVID-19 pandemic

Through analysing the dataset, some issues checked for were missing values throughout the dataset. Although there were no missing values, this was accounted for through the following code sequence:

```
import pandas as pd

data = pd.read_csv('^IXIC.csv')

print(data.isnull().sum())
```

with the following output indicating the dataset has no (0) missing values in each attribute.

```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

Another potential issue accounted for was to ensure a constant number of rows in the dataset. This was done through the following code sequence:

```
num_rows = 0

for row in open('data.csv'):
    num_rows += 1

print(num_rows)
```

```
[user@sahara ~]$ python num_rows_check.py
630
```

To ensure consistency across the datasets collected by each member of the group, the original dataset was edited to include a new attribute: percentage change. This was done through the following spreadsheet code on Microsoft Excel $= (E2 - B2)/B2 * 100$. This code obtained the difference between the closing and opening prices for each trading day and divided that difference by the opening price. The result was then multiplied by 100 to format the result in percentage form. By filling out the entire column with an autofill function, a new attribute column was created for consistency across datasets.

Some simple analysis done through Python code on the dataset were:

Finding the average percentage change from 2019 to 2021

```
the_dict = {}
first_line = True

for idk in open('data.csv'):
    if first_line:
        first_line = False
    else:
        val = idk.split(',')
        date = val[0]
        year = date.split('/')[2]
        per_change = float(val[7])
        if year in the_dict:
            the_dict[year].append(per_change)
        else:
            the_dict[year] = [per_change]

for key in sorted(the_dict):
    avg = sum(the_dict[key])/len(the_dict[key])
    print(key + ": " + str(round(avg, 4)) + "%")
```

```
[user@sahara ~]$ python per_change.py
19: 0.073%
20: 0.0762%
21: -0.004%
```

Finding the average closing price for each year:

```
the_dict = {}
first_line = True

for idk in open('data.csv'):
    if first_line:
        first_line = False
    else:
        val = idk.split(',')
        date = val[0]
        year = date.split('/')[2]
        close = float(val[4])
        if year in the_dict:
            the_dict[year].append(close)
        else:
            the_dict[year] = [close]

for key in sorted(the_dict):
    avg = sum(the_dict[key])/len(the_dict[key])
    print(key + ": " + "$" + str(round(avg, 2)))
```

```
[user@sahara ~]$ python change.py
19: $7940.13
20: $10201.51
21: $13603.95
```

Finding the highest selling price of a stock for each year:

```
max_price = 0
line_count = 0

for file in open('data.csv'):
    if line_count > 0:
        val = file.split(',')
        date = val[0]
        year = date.split('/')[2]
        high = float(val[2])

        if year == '19':
            high_19 = max(max_price, high)

        if year == '20':
            high_20 = max(max_price, high)

        if year == '21':
            high_21 = max(max_price, high)

    line_count += 1

print("Highest selling price of 2019 = $" + str(round(high_19, 2)))
print("Highest selling price of 2020 = $" + str(round(high_20, 2)))
print("Highest selling price of 2021 = $" + str(round(high_21, 2)))
```

```
[user@sahara ~]$ python highest_high.py
Highest selling price of 2019 = $8975.36
Highest selling price of 2020 = $12902.07
Highest selling price of 2021 = $14526.81
```

Another form of analysis was executed through Microsoft with the codes $= C2 - D2$ and $= E2 - B2$. Both codes correspond to subtracting one value from another. With the first code, the highest price for each trading day was subtracted with the lowest price to obtain the difference between the highest and lowest trading prices for each trading day. The second code obtained the difference between the closing and opening price for each trading day.

Section 2C - ASX 200 (500490723)

The dataset that was acquired was S&P_ASX 200 Historical Data.csv. It is a dataset in the comma-separated values format which pertains to the daily trades of the S&P ASX 200 between 1/01/2019 to 30/06/2021(EOFY 2021) and contains 633 rows of data. This data contains 6 attributes:

Attribute	**Format**	**Meaning**
Date	month/day/year (e.g. Jun 30,2021)	Day of which the trades occurred within normal ASX trading times(10am to 4pm Sydney time)
Open	AUD, rounded to 1dp with 2dp shown, and comma separated per 1000 (e.g. 7,203.90)	The stock's price at the start of the trading day
Close	AUD, rounded to 1dp with 2dp shown, and comma separated per 1000 (e.g. 6,001.20)	The stock's price at the end of the trading day
High	AUD, rounded to 1dp with 2dp shown, and comma separated per 1000 (e.g. 6,303.50)	The stock's highest price at which it was traded on a specific day
Low	AUD, rounded to 1dp with 2dp shown, and comma separated per 1000 (e.g. 7,203.90)	The stock's lowest price at which it was traded on a specific day
Volume	Numeric, rounded to 2dp, with either a M or B suffix for million and billion respectively.	The total amount of shares traded on the day

The original source of this data is the ASX or Australian Securities Exchange Ltd. This data was given to intermediary market makers who then supplied this data to Fusion Media Ltd, the company that runs au.investing.com. I am fully authorised to use the data I acquired as it states on the my sources website, "Our services are provided for free and you are welcome to use the information and tools we provide".

Data limitations and strengths

A notable limitation of this data, which is mentioned on the investing.com website, states "The data contained in this website is not necessarily real-time nor accurate...so prices may not be accurate and may differ from the actual market price, meaning prices are indicative." Inaccuracies are never good for any data set if we want conclusive results, yet it shouldn't have too much of an impact on my results as these inaccuracies are objectively very minute as they will only be a small fraction away from the true values. With limitations in regards to our topic question, there are many variables which are not included that could have a significant impact on the stock prices such as the various impacts of covid as well as the subsequent government spending and subsidies, specifically subsidies for large companies. This boosts their profits and therefore increases the stock price of the asx but this is not correlated to an increased company performance, one of the main variables we are attempting to explore. In regards to strengths of the dataset, the several attributes of the ASX200 index is a very good starting point to help in

the understanding of the relative performance of companies contained in the ASX. It is also extremely consistent with no potential outliers, and the length of 633 rows gives it enough data to be able to adjust, change or remove information for integration without any significant drawbacks. In regards to independent origin, all my peers have chosen different exchanges/indexes from one another and thus we all have our data from independent origin as the source of the data comes from the exchanges themselves.

Data quality and cleaning

Several steps were taken to ensure data quality and cleaning. Firstly I opened my csv in excel to visualise the document I had just downloaded and instantly realised the file was in reverse chronology, which was quickly fixed by making a new column of numbers from 1-633 and sorting the whole document by that column. The column was then removed leaving the original dataset now flipped and in chronological order. The rows of the dataset looked like so:

Jun 30, 2021	7,301.20	7,313.00	7,370.10	7,301.20	675.23M

I used Ctrl+h to find and remove all commas from the data set, as this would otherwise cause several issues when attempting to split each row at the comma to get individual values.Since this doesn't work for only numbers I used excel formatting to remove the commas from rows 2-5. After importing the csv into my local python IDE another issue instantly arose as each row of my data showed as:
"Jan 02 2019","5557.80","5646.40","5652.50","5552.00",350.17M

Therefore numbers would be presented as strings and never integers as the quotation marks were a part of the string. I simply used Ctrl+h again to find and replace all instances of quotations with nothing. For my date, it was displayed as a completely incorrect format (Jan 02 2019)
For this I wrote a few lines of code for each month of the year for replacement. E.g.

```
if “Jan” in date:
        date.replace(“Jan ”, “01/” )
        date.replace(“ “, “/ “)
if “Feb” in date:
        date.replace(“Feb ”, “02/” )
        date.replace(“ “, “/ “)
etc.
```

This changed all months into numbers as well as removing whitespace behind the month and replacing the remaining whitespace with a slash. This code would output dates such as 01/02/2019. Next I needed to swap month with day. This was done with the following python code by splitting by slash and reordering the parts:

```
date.split(“/”)
date[0], date[1] =  date[1], date[0]
date = "/".join(date)
```

As this was my only formatting issue, I saved my dataset with new date values to a new file. After this step I could now check my data. First I set up all the variables:

```
for row in open("asx.csv"):
        if first_line:
                first_line = False

        else:
                new_row = row.rstrip("\n")
                values = new_row.split(",")
                date = values[0]
                open_price = float(values[1])
                high = float(values[2])
                low = float(values[3])
                close_price = float(values[4])
                volume = values[5]
```

I didn't need to check any of the numeric variables for incorrect format as the above code worked(it would error when attempting to turn the value into a float). Then I wrote a simple check for missing values when writing all the values back to a new file:

```
with open(clean_asx.csv', 'a') as f:
        if i == 0:
                f.write(start_vaules)
                i += 1
        save_values = ",".join(values)
        if date and open_price and high and low and close_price and volume:
                f.write(f"\n{save_values}")
```

Since an empty string is falsy in a boolean context and a string is true, if one of the variables ends up being an empty string, this code simply skips the whole line when writing to the clean file. After running this code I had the same 633 rows meaning there were no missing values. Lastly I opened up my new clean csv and manually inspected it in Excel to find no other issues.

Data summaries

My python code which produced the data summaries is as follows:

```
#setting up variables i need to use later
first_line = True
growth_2019 = 0
growth_2020 = 0
growth_2021 = 0
max_volume = 0
i = 0

#opening the file and setting up values as shown previously
for row in open("S&P_ASX 200 Historical Data.csv"):
        if first_line:
                first_line = False
```

```
            new_row = row.rstrip("\n")
            values = new_row.split(",")
            values.append("Change")
            start_vaules = ",".join(values)
    else:
            new_row = row.rstrip("\n")
            values = new_row.split(",")
            date = values[0]
            open_price = float(values[1])
            close_price = float(values[4])
            volume = values[5]
```

#For one of my aggregate summaries I decided to calculate the percent change per day. This was done through a simple calculation where we find the difference from the open and close values, divide by the original price and multiply by 100 to get a percentage. It is then formatted to 2dp and a % is added to the end of the number.

```
    change = ((close_price - open_price)/open_price)*100
    percent_change = ("{:.2f}%".format(change))
```

 # I felt it would also be a good addition to my dataset and the integrated dataset so I wrote some extra code to append this value on the end of each line of my dataset. When initialising my first line values above I added "Change" to my list of attributes.

```
    values.append(percent_change)
    with open('save.csv', 'a') as f:
            if i == 0:
                    f.write(start_vaules)
                    i += 1
            save_values = ",".join(values)
            f.write(f"\n{save_values}")
```

 # For another summary I did a grouped aggregate summary which calculated the percentage change for each year. This was done by checking the date variable and adding the relevant percentage change together.

```
    if "2019" in date:
            growth_2019 += change
    if "2020" in date:
            growth_2020 += change
    if "2021" in date:
            growth_2021 += change
```

 # My third data summary was to find the maximum trading volume. Since my trading volume was not purely a float, I removed the M and B at the end of the volume e.g. 905M -> 905. If a B was removed I would simply remove the B then multiply the number by 1000. These floats could now be compared and if the new float was higher, it would replace the old one.

```
    if "M" in volume:
        volume = volume.replace("M", “”)
    if "B" in volume:
        volume = volume.replace("B", “”)
    V   olume = float(volume) *1000
    if float(volume) > max_volume:
        max_volume = float(volume)


print(f"The 2019 yearly growth was {growth_2019:.2f}%")
print(f"The 2020 yearly growth was {growth_2020:.2f}%")
print(f"The 2021 yearly growth was {growth_2021:.2f}%")
print(f"The maximum trading volume for any one day was {max_volume}M shares.")
```

Running the program gave the following output as well as a new file with a new change variable.

```
$ python data.py
The 2019 yearly growth was 17.59%
The 2020 yearly growth was 2.30%
The 2021 yearly growth was 10.79%
The maximum trading volume for any one day was 2450.0M shares.
```

New file first 5 lines:

```
Date,Open,High,Low,Close,Vol,Change
2/01/2019,5646.4,5652.5,5552,5557.8,350.1M,-1.57%
3/01/2019,5557.8,5646.5,5557.8,5633.4,451.3M,1.36%
4/01/2019,5633.4,5633.4,5568.7,5619.4,475.8M,-0.25%
7/01/2019,5619.4,5716.2,5619.4,5683.2,439.6M,1.14%
```

Section 2D - Bovespa (500308222)

a) The Bovespa dataset contains information regarding the Bovespa Index which is a benchmark index of roughly 84 stocks traded on the *B3 - Brazil Stock Exchange and Over-the-Counter Market,* with information from dates ranging 02/01/2019 to 30/06/2021. The data is a text file in a .csv file format (comma-separated values) and consists of ordinal and numeric values in strings. The dataset 'Bovespa Historical Data.csv' was downloaded from Investing.com on 10/09/2021. The primary source of the data was the B3 Stock Exchange, and Investing.com states that the intermediary party was Market Maker CFDs.
The dataset contains 619 rows and 7 variables, with each row representing a trading day. The price values (Price, Open, High, Low) are in BRL (Brazilian Real).

Data Dictionary:

Variable	Meaning	Data Type
Date	Date when the market was open; Trading day	Ordinal value; string in format Mon DD, YYYY
Price	Index price when market closed for the trading day	Numeric value with 0 d.p.; string
Open	Index price when market opened for the trading day	Numeric value with 0 d.p.; string
High	Highest index price during trading day	Numeric value with 0 d.p.; string
Low	Lowest index price during trading day	Numeric value with 0 d.p.; string
Vol.	Number of index shares involved in transactions for the trading day	Numeric value with 2 d.p.; string; M represents Millions, K represents Thousands
Change %	Percentage change in price as compared to previous trading day	Numeric percentage value with 2 d.p. and a percentage symbol; string

For the purposes of investigating the performance of the Brazilian stock market, the dataset is quite robust as it contains extensive pricing information about the Bovespa Index which accounts for the majority of trading and market capitalization in the Brazilian stock market. However, limitations of the data include:

- Lack of after-hours trading information
- Lack of information regarding Brazil's economic performance (e.g. GDP)
- Lack of information regarding world events (e.g. COVID-19. changing export demand) and political events (e.g. new subsidy and tax laws, new trade

agreements, low poll numbers for President Jair Bolsonaro) and how they have affected market performance

These limitations mean that our analysis will not take all factors which affect market performance into account. The relevance and accuracy of the conclusions and comparisons to other datasets that we draw from the Bovespa dataset will therefore not be fully informed, unable to be weighted to account for all relevant information/events.

Regarding permission to use this data, it is stated in the Investing.com Terms and Conditions that “Fusion Media grants you a non-exclusive, non-transferable and limited personal license to access and use the Services”, their basic services are free and the data is publicly available to access upon creation of an account. (Fusion Media is the body that runs Investing.com)

b) Firstly, the columns in the Bovespa dataset were rearranged and renamed from [Date, Price, Open, High, Low, Vol., Change %] to [Date, Open, High, Low, Close, Volume, Change %] to match the structure found in other datasets. To ensure data quality, Python was used to automatically check that there were no missing values, that the various price columns (Open, High, Low, Close) and the Volume column only consisted of digits/numerical values. Python also automatically checked that the date was in a format identical to or adjacent to the format YYYY-MM-DD by checking the length of the values in the Date Column. As can be seen in the output below, the missing value function printed False indicating that there were no missing values. All of the price-digit functions and the volume function printed False indicating that there were values in each column that did not consist of exclusively digits. The date function printed False which indicated that there were date values that did not follow the YYYY-MM-DD format or a similar format.

```
C:\Users\Marsha Lim\Documents\DATA1001>python Change_date.py
Presence of Missing Values: False
Check all Open values are digits: False
Check all High values are digits: False
Check all Low values are digits: False
Check all Close values are digits: False
Check Date is in correct/correct-adjacent format: False
```

To fix the quality issue in the price columns, Python was used to remove all of the commas, so that only the digits remained. To fix the quality issue in the Volume column, through Python, the Ms and Ks were removed and the entailed amount of 0s were appended to the values. To fix the quality issue in the date column, Python was used to bring the date format from “Month DD, YYYY” to the common standard of “YYYY-MM-DD”.

After the dataset was cleaned and values formatted, another data quality issue that was checked automatically in Python was to ensure that all numeric values (Open, High, Low, Close, Volume) are greater than 0 and the cleaning was done correctly. As the output below shows, there are no missing values in the cleaned dataset and all numeric values are above 0, which shows that there were no errors when cleaning the dataset.

```
C:\Users\Marsha Lim\Documents\DATA1001>python check_clean.py
Presence of Missing Values: False
All Open values >0: True
All High values >0: True
All Low values >0: True
All Close values >0: True
All Volume values >0: True
```

Code used to check and fix raw dataset: (Bovespa_checkrawandclean.py in folder, Change_data.py in CMD output screenshot)

```python
import pandas as pd

training = pd.read_csv(r'C:\Users\Marsha Lim\Documents\DATA1001\Bovespa Historical Data - Raw.csv')
training = training[['Date', 'Open', 'High', 'Low', 'Price', 'Vol.', 'Change %']] # Rearrange columns
training.rename(columns={"Vol.":"Volume", "Price":"Close"}, inplace=True) # Rename Volume and Close
columns

print("Presence of Missing Values: "+str(training.isnull().values.any())) # Check if there are any
missing values
print("Check all Open values are digits: "+str(training['Open'].str.isdigit().values.all())) # Check if
all price values are exclusively digits
print("Check all High values are digits: "+str(training['High'].str.isdigit().values.all()))
print("Check all Low values are digits: "+str(training['Low'].str.isdigit().values.all()))
print("Check all Close values are digits: "+str(training['Close'].str.isdigit().values.all()))
print("Check all Volume values are digits: "+str(training['Close'].str.isdigit().values.all())) # check
if all volume values are exclusively digits

def check_date(df): # Check date format
    check = 0
    date_check_col = df['Date'].str.len()
    for x in date_check_col:
        if x != 10.0:
            check = 1
    return check
print("Check Date is in correct/correct-adjacent format: "+str(check_date(training) == 0))

first_column = training.iloc[:, 0] #Isolate Date column
index = 0
for date in first_column:
    old_date = date.split(" ") #Change Month names to numerical values
    if old_date[0] == "Jan":
        new_date = "01"
    if old_date[0] == "Feb":
        new_date = "02"
    if old_date[0] == "Mar":
        new_date = "03"
    if old_date[0] == "Apr":
        new_date = "04"
    if old_date[0] == "May":
        new_date = "05"
    if old_date[0] == "Jun":
        new_date = "06"
    if old_date[0] == "Jul":
        new_date = "07"
    if old_date[0] == "Aug":
        new_date = "08"
    if old_date[0] == "Sep":
        new_date = "09"
    if old_date[0] == "Oct":
        new_date = "10"
    if old_date[0] == "Nov":
        new_date = "11"
    if old_date[0] == "Dec":
        new_date = "12"
    new_date = old_date[2]+"-"+new_date+"-"+old_date[1].replace(',', '') #Change date format to YYYY-
MM-DD to match datetime64
    training.at[index,'Date'] = new_date # Update Date column
    index+=1


training.iloc[:, 1:5] = training.iloc[:, 1:5].replace(',','', regex= True) # Remove commas in various
price columns values

training['Date'] = pd.to_datetime(training.Date) # Change date column to datetime64 format


index = 0
vol_column = training.iloc[:, 5]  # Isolate Volume Column
for vol in vol_column:            # Volume data is in format X.XXM
    vol = vol.replace('.','')     # Remove decimal place
    if vol[-1] == "M":            # Replace all M with 0000
        vol = vol.replace('M','')
        vol = vol+"0000"
    if vol[-1] == "K":            # Replace all K with 0
        vol = vol.replace('K','')
        vol = vol+"0"
    training.at[index, "Volume"] = vol
    index+=1                    # Volume data is now in format XXX0000

print(training.head())
training.to_csv('Bovespa Historical Data - Cleaned.csv', index = False) # export new dataset to csv
```

Code used to check cleaned dataset: (Bovespa_check_clean.py in folder; check_clean.py in CMD output screenshot)

```python
import pandas as pd

training = pd.read_csv(r'C:\Users\Marsha Lim\Documents\DATA1001\Bovespa Historical Data - Cleaned.csv')
# import clean dataset

print("Presence of Missing Values: "+str(training.isnull().values.any())) # Check if there are any
missing values
print("All Open values >0: "+str((training['Open']>0).all())) # Check all price values > 0
print("All High values >0: "+str((training['High']>0).all()))
print("All Low values >0: "+str((training['Low']>0).all()))
print("All Close values >0: "+str((training['Close']>0).all()))
print("All Volume values >0: "+str((training['Volume']>0).all())) # Check all Volume values > 0
```

c) Data Summaries:

- Grouped-aggregate: Highest trading price per year (within observed period of 2019-01-02 to 2021-06-30)
- Maximum Trading Volume
- Minimum Trading Volume
- Average Trading Volume
- Largest Percentage Increase (1 Day)
- Average Closing Price
- Closing price percentage increase from 2019-01-02 (start of observed period) to 2021-06-30 (end of observed period)

Code Output:

```
C:\Users\Marsha Lim\Documents\DATA1001>python Analysis.py
Highest trading price per year
Highest trading price for 2019: 117803
Highest trading price for 2020: 120150
Highest trading price for 2021: 131190


Maximum Trading Volume: 21770000
Minimum Trading Volume: 424320
Average Trading Volume: 7687018.29
Largest Percentage Increase (1 Day): 13.91%
Average Closing Price: 103743.71
Closing Price Percentage Increase from 2019-01-02 to 2021-06-30: 39.32%
```

Data Summaries Code: (Bovespa_datasummaries.py in folder; Analysis.py in CMD output screenshot)

```python
import pandas as pd
import dateutil

cleaned_dataset = pd.read_csv(r'C:\Users\Marsha Lim\Documents\DATA1001\Bovespa Historical Data -
Cleaned.csv') # Import cleaned dataset

cleaned_dataset["Date"] = cleaned_dataset["Date"].apply(lambda x: dateutil.parser.parse(x)) # Splits
dataset by year
year = cleaned_dataset["Date"].dt.to_period("Y")
yearly_aggregate = cleaned_dataset.groupby([year])

print("Highest trading price per year")
index = 0
for year in yearly_aggregate: # Prints highest value in "High" column per year
    print("Highest trading price for "+str(2019+index)+": "+str(year[1]['High'].max()))
    index+=1
print('\n')

change_col = cleaned_dataset['Change %'] #
change_col = change_col.replace('%','', regex= True).astype(float)
print("Maximum Trading Volume: "+str(cleaned_dataset.iloc[:, 5].max())) # Maximum value in Volume
column
print("Minimum Trading Volume: "+str(cleaned_dataset.iloc[:, 5].min())) # Minimum value in Volume
column
print("Average Trading Volume: "+str(round(cleaned_dataset.iloc[:, 5].mean(),2))) # Average value in
Volume column
print("Largest Percentage Increase (1 Day): "+str(change_col.max())+"%") # Highest value in Change %
Column
print("Average Closing Price: "+str(round(cleaned_dataset.iloc[:, 4].mean(),2))) # Average value in
Close column
print("Closing Price Percentage Increase from 2019-01-02 to 2021-06-30:
"+str(round(((cleaned_dataset.iloc[-1, 4]-cleaned_dataset.iloc[0, 4])/cleaned_dataset.iloc[0,
4])*100,2))+"%") # Percentage increase in closing price from SOP to EOP
```

Section 3 - Data Integration

Date Integration:
The 4 datasets were integrated horizontally, merging on the Date column present in all datasets. After converting all Date column values to datetime64 objects, the datasets were integrated using the pandas.merge function. The merge function automatically removed the values corresponding to dates that did not appear in all datasets. While some information was thus removed, this eliminated the possibility of missing values in the integrated dataset, making future comparison and analysis easier.

Combined dataset schema:
Within the combined dataset, the attributes are the date of each trading day, opening and closing index price for each of the 4 stock exchanges, highest and lowest index price for all exchanges, the volumes for all exchanges and the daily percentage change in closing index of each exchange for each trading day.

Challenges encountered:

- In integrating the data, there was an issue regarding the inconsistent date format across the datasets when importing them into Python. As the date was the column along which we were integrating the datasets, it was important that they were all in the same format. Furthermore, since the dates were not in the YYYY-MM-DD format, when converting them into datetime64 objects, the order would be incorrect and for some values the date and month would be switched. To combat both these issues in one stroke, we used Python to adjust the order and format of the 'Date' column in each dataset to match the YYYY-MM-DD format before converting the values into datetime64 objects.
- Another challenge we encountered was that column names representing certain categories were not consistent. These included the column names representing the "Change %" and "Volume" columns. These columns were renamed in Python to match the other datasets. Furthermore, the "Change %" column was not consistent in formatting, with some datasets not having a % sign at the end, with differing decimal places. Using Python, these columns were rounded to 2 decimal places and "%" was appended, to match the other datasets.
- Finally, since the datasets all largely consisted of the same attributes for differing exchanges, the column names were largely identical. This posed an issue as it would be difficult to differentiate which data belonged to which exchange after the merge was performed. Therefore, the names of the exchange were appended to their column names.

In order to confirm that the merge was completed correctly in Python, we compared the dates of each dataset using the "conditional formatting" function in Excel, which highlighted all the dates which were a unique value. After deleting these values, the" find and select" function was used to highlight all the empty rows which were subsequently deleted. This left 590 rows, the same as the merged dataset which confirmed that the merge was successful in eliminating the values with dates that did not match the other datasets.

Integration Code:

```python
import pandas as pd

bovespa = pd.read_csv(r'Bovespa Historical Data - Cleaned.csv') # Import datasets
ftse = pd.read_csv(r'FTSE100_clean(1).csv')
asx = pd.read_csv(r'asx - cleaned.csv')
ixic = pd.read_csv(r'^IXIC (cleaned).csv')

ftse_date = ftse['Date'] # This code chunk changes the FTSE date format to YYYY-MM-DD to match
datetime64
index = 0
for date in ftse_date:
    date_split = date.split('/')
    if len(date_split[0]) == 1:
        date_split[0] = "0"+date_split[0]
    #print(date_split[0])
    if len(date_split[1]) == 1:
        date_split[1] = "0"+date_split[1]
    date = "20"+date_split[2]+"-"+date_split[1]+"-"+date_split[0]
    ftse.at[index,'Date'] = date
    index+=1

asx_date = asx['Date'] # Change ASX date format to YYYY-MM-DD to match datetime64
index = 0
for date in asx_date:
    date_split = date.split('/')
    if len(date_split[0]) == 1:
        date_split[0] = "0"+date_split[0]
    date = date_split[2]+"-"+date_split[1]+"-"+date_split[0]
    asx.at[index,'Date'] = date
    index+=1

ixic_date = ixic['Date'] # Change IXIC date format to YYYY-MM-DD to match datetime64
index = 0
for date in ixic_date:
    date_split = date.split('/')
    if len(date_split[0]) == 1:
        date_split[0] = "0"+date_split[0]
    date = date_split[2]+"-"+date_split[1]+"-"+date_split[0]
    ixic.at[index,'Date'] = date
    index+=1

ftse.iloc[:,5] = round(ftse.iloc[:, 5],2).astype(str) # Round FTSE Change % column to 2d.p. and add "%"
at end to match
ftse.iloc[:,5] = ftse.iloc[:,5]+"%"

ixic['Change %'] = round(ixic['Change %'],2).astype(str) # Round IXIC Change % column to 2d.p. and add
"%" at end to match
ixic['Change %'] = ixic['Change %']+"%"

asx.rename(columns={"Vol.":"Volume"}, inplace=True) # Change ASX column name Vol. to Volume in ASX
dataset to match
asx.rename(columns={"Change":"Change %"}, inplace=True) # Change ASX column name "Change" to "Change %"
in ASX dataset to match
ftse.rename(columns={" % change":"Change %"}, inplace=True) # Change ASX column name "% change" to
"Change %" in FTSE dataset to match

# Append exchange names to each column
bovespa.columns= [str(col) + '_Bovespa' for col in bovespa.columns]
ftse.columns= [str(col) + '_FTSE' for col in ftse.columns]
asx.columns= [str(col) + '_ASX' for col in asx.columns]
ixic.columns = [str(col) + '_IXIC' for col in ixic.columns]

data_list = [bovespa, ftse, asx, ixic] # Change the first column back to 'Date' for integration
for set in data_list:
    set.rename(columns={set.columns[0]:"Date"}, inplace=True)

bovespa['Date'] = pd.to_datetime(bovespa['Date']) # Change Date values to datetime64
ftse['Date'] = pd.to_datetime(ftse['Date'])
asx['Date'] = pd.to_datetime(asx['Date'])
ixic['Date'] = pd.to_datetime(ixic['Date'])

merge_1 = pd.merge(bovespa, ftse, left_on = 'Date', right_on = 'Date') # Merge Bovespa and FTSE dataset
merge_2 = pd.merge(merge_1, asx, left_on = 'Date', right_on = 'Date') # Merge ASX dataset
merge_3 = pd.merge(merge_2, ixic, left_on = 'Date', right_on = 'Date') # Merge IXIC dataset

merge_3.to_csv('Merged Files.csv', index = False) # Export merged files to CSV
```