Part 2: Data Exploration and Analysis

By Josh Houlding

The purpose of this assignment is to apply Python to the preliminary tasks of data exploration and analysis. To demonstrate completion of this assignment, create a Microsoft Word document with your working code, screenshots of program results, and written answers to questions. Writing should be professional and rigorous, and include scientific/mathematical justification, where appropriate, for all conclusions reached. Upload your final Jupyter notebook and Word document to the digital classroom when complete.

In 250-500 words, answer the following:

1. Import the necessary libraries in Python that will allow you to use data frames, lists, load datasets, and plots.
2. Load your dataset into a data frame using the pandas library, then display the first 10 rows of your data.
3. Perform univariate analysis on your loaded dataset using the describe function. Report the results.
4. Based on the previous step results, spot and interpret any outliers (negative or positive skewness) and/or missing data.
5. For each outlier you spotted in the previous step, plot it into a scatter plot using the seaborn library. Show and interpret the graphs.
6. Perform an outlier treatment to the variables that have outliers. Report and explain the results.
7. Perform missing values imputation to the variables that have missing values. Explain which method you used and why. Report the results.
8. Perform bivariate analysis to study the strength of the correlation among your variables. Hint: use scatter plots or correlation matrix methods to explain the relationship between two variables. Report and interpret the results.
9. Based on the results reported in the previous step, perform data transformation, as needed, to transform a relationship from nonlinear to linear. Explain your decision on the transformation method used here.
10. Show and explain how you would handle qualitative data in your dataset. Hint: use Dummy variables.
11. Test for multicollinearity in the dataset using the variance_inflation_factor() function. Report the VIF values and interpret the results.

Task 1

Import the necessary libraries in Python that will allow you to use data frames, lists, load datasets, and plots.

In [50]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

Task 2

Load your dataset into a data frame using the pandas library, then display the first 10 rows of your data.

In [51]:
```
# Load data
df = pd.read_csv("AAPL.csv")

# Drop useless column "Unnamed: 0"
df.drop(columns={"Unnamed: 0"}, inplace=True)

# Show first 10 rows
df.head(10)
```

Out[51]:

	symbol	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume	divCash	s
0	AAPL	2015-05-27 00:00:00+00:00	132.045	132.260	130.05	130.34	45833246	121.682558	121.880685	119.844118	120.111360	45833246	0.0	
1	AAPL	2015-05-28 00:00:00+00:00	131.780	131.950	131.10	131.86	30733309	121.438354	121.595013	120.811718	121.512076	30733309	0.0	
2	AAPL	2015-05-29 00:00:00+00:00	130.280	131.450	129.90	131.23	50884452	120.056069	121.134251	119.705890	120.931516	50884452	0.0	
3	AAPL	2015-06-01 00:00:00+00:00	130.535	131.390	130.05	131.20	32112797	120.291057	121.078960	119.844118	120.903870	32112797	0.0	
4	AAPL	2015-06-02 00:00:00+00:00	129.960	130.655	129.32	129.86	33667627	119.761181	120.401640	119.171406	119.669029	33667627	0.0	
5	AAPL	2015-06-03 00:00:00+00:00	130.120	130.940	129.90	130.66	30983542	119.908625	120.664274	119.705890	120.406248	30983542	0.0	
6	AAPL	2015-06-04 00:00:00+00:00	129.360	130.580	128.91	129.58	38450118	119.208267	120.332526	118.793582	119.411002	38450118	0.0	
7	AAPL	2015-06-05 00:00:00+00:00	128.650	129.690	128.36	129.50	35626800	118.553986	119.512370	118.286744	119.337280	35626800	0.0	
8	AAPL	2015-06-08 00:00:00+00:00	127.800	129.210	126.83	128.90	52674786	117.770691	119.070039	116.876813	118.784366	52674786	0.0	
9	AAPL	2015-06-09 00:00:00+00:00	127.420	128.080	125.62	126.70	56075420	117.420512	118.028717	115.761770	116.757015	56075420	0.0	

Apple is the company for every entry in this dataset, so `symbol` is redundant and should be removed.

In [52]:
```
# Remove "symbol"
df.drop(columns={"symbol"}, inplace=True)
```

Task 3

Perform univariate analysis on your loaded dataset using the describe function. Report the results.

In [53]:
```
# Describe data
df.describe()
```

Out[53]:

	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume
count	1258.000000	1258.000000	1258.000000	1258.000000	1.258000e+03	1258.000000	1258.000000	1258.000000	1258.000000	1.258000e+03
mean	167.723998	169.230475	166.039780	167.548266	3.500397e+07	162.666715	164.131054	161.028013	162.493082	3.500397e+07
std	56.850796	57.500128	56.006773	56.612707	1.729100e+07	58.733820	59.402842	57.869246	58.494560	1.729100e+07
min	90.340000	91.670000	89.470000	90.000000	1.136204e+07	84.954351	86.205062	84.136216	84.634620	1.136204e+07
25%	116.327500	117.405000	115.602500	116.482500	2.359205e+07	109.484490	110.393556	107.962457	109.135002	2.359205e+07
50%	160.485000	162.080000	158.974250	160.345000	3.064771e+07	154.710645	156.091874	153.054341	154.410017	3.064771e+07
75%	199.785000	201.277500	198.170000	199.520000	4.100487e+07	196.960053	198.428438	195.281553	196.452903	4.100487e+07
max	327.200000	327.850000	323.350000	324.730000	1.622063e+08	326.337147	326.357095	322.497300	323.873661	1.622063e+08

We see that `splitFactor` is always 1 and thus has no predictive power, so we will remove it.

In [54]:
```
# Drop "splitFactor"
df.drop(columns={"splitFactor"}, inplace=True)
```

Task 4

Based on the previous step results, spot and interpret any outliers (negative or positive skewness) and/or missing data.

In [55]:
```
# Find number of rows with missing values
num_missing = df.isna().any(axis=1).sum()
print(f"Number of rows with missing values: {num_missing}")
```

```
Number of rows with missing values: 0
```

It appears there are no missing values in this dataset.

Next, we will create a boxplot for each numeric variable to see which ones, if any, have major outliers that need to be dealt with.

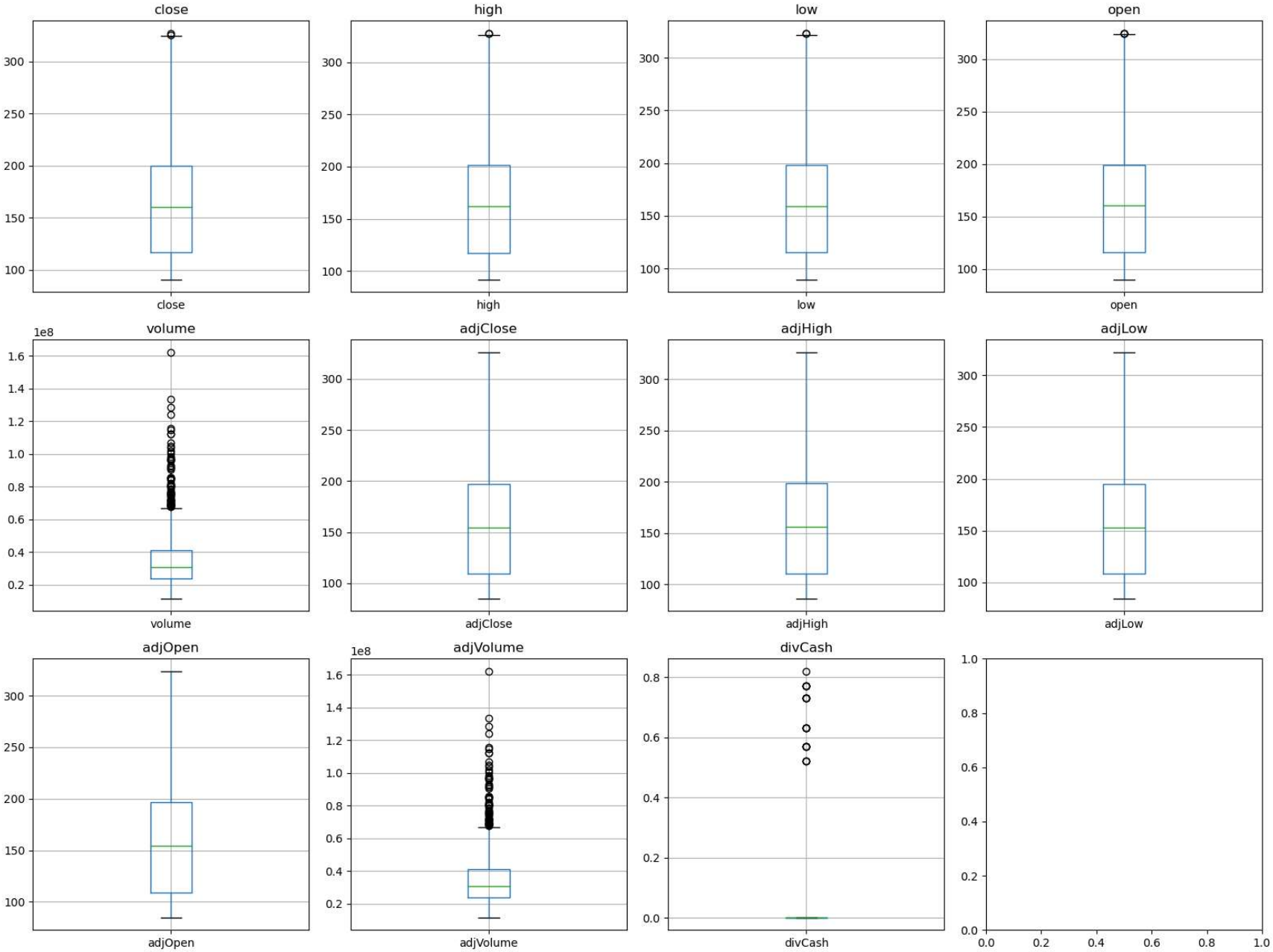
In [56]:

```
# Create 3x4 grid
fig, axes = plt.subplots(3, 4, figsize=(16, 12))
axes = axes.flatten()

numeric_columns = df.drop(columns={"date"})

# Create a boxplot for every numeric column
for i, col in enumerate(numeric_columns.columns):
    df.boxplot(column=col, ax=axes[i])
    axes[i].set_title(col)

# Show the plots
plt.tight_layout()
plt.show()
```



Based on these plots, it looks like only `volume` and `adjVolume` have enough outliers to justify taking action. However, we see that the boxplots for both variables look identical, which is suspicious. Let's check if any entries in the dataframe have different `volume` and `adjVolume` values.

In [57]:
```
# Check for rows with different "volume" and "adjVolume" values
len(df[df["volume"] != df["adjVolume"]])
```

Out[57]:
```
0
```

Apparently, `volume` and `adjVolume` are the same column value-wise, so we can safely ignore `adjVolume` and only focus on `volume`.

Now, let's see which rows contain the `volume` outliers.

In [58]:
```
# Function to find outliers using Z-score method
def find_outliers(data, threshold=3):
    z_scores = stats.zscore(data)
    return data[(z_scores > threshold) | (z_scores < -threshold)]

# Find and display outliers
outlier_indices = find_outliers(df["volume"]).index
volume_outliers = df.loc[outlier_indices]
volume_outliers["volume"]
```

Out[58]:
```
39      115450607
48      124138623
49       98384461
53       97082814
54      101685610
61      128275471
62      162206292
63      103601599
64       96774611
144      96453327
169     133369674
232     114602142
295      92344820
329     112340318
330      90613177
425     111985040
838      96246748
868      91328654
901      95744384
908      91312195
1198    106721230
1207    104618517
1208     92683032
1213    100423346
Name: volume, dtype: int64
```

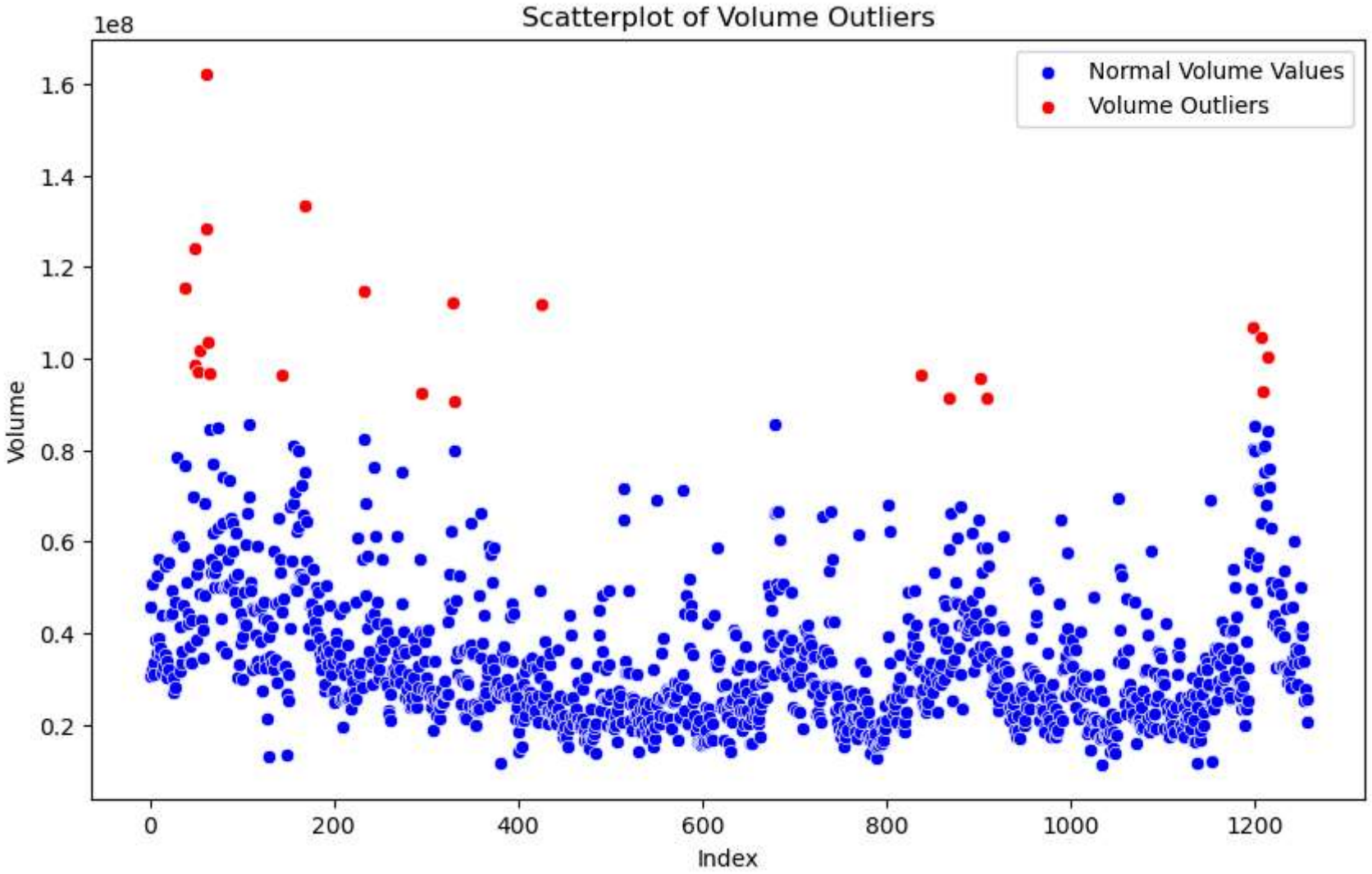
It is plausible that more than 100 million shares were traded in a single day, as we see in some of these outliers, given that Apple had over 16 billion outstanding shares as of July 16th, 2021. These days of high activity might have happened because of earnings announcements, news about new Apple products, potential controversies brewing, or any number of other events that could spur traders to buy or sell shares.

Task 5

For each outlier you spotted in the previous step, plot it into a scatter plot using the seaborn library. Show and interpret the graphs.

In [59]:
```
# Plot "volume" values with outliers highlighted
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x=df.index, y="volume", color="blue", label="Normal Volume Values")
sns.scatterplot(data=volume_outliers, x=volume_outliers.index, y="volume", color="red", label="Volume Outliers")
plt.xlabel("Index")
plt.ylabel("Volume")
```

```
plt.title("Scatterplot of Volume Outliers")
plt.show()
```



We see that there are a few outliers on the high side, but they are not high enough to be nonsensical. Again, these could represent days when info on new Apple hardware or software was announced (WWDC, for example), or major controversies (eg. when stories broke about Apple throttling performance of older devices with weaker batteries).

Task 6

Perform an outlier treatment to the variables that have outliers. Report and explain the results.

The outliers that were identified earlier are well within the realm of possibility, so I believe removing them would negatively affect the analysis. Thus, I have opted to leave them alone.

Task 7

Perform missing values imputation to the variables that have missing values. Explain which method you used and why. Report the results.

In [60]:
```
# Show number of missing values
print(f"Number of rows with missing values: {num_missing}")
```

```
Number of rows with missing values: 0
```

As we found earlier, there are no missing values, and thus nothing needs to be done here either.

Task 8

Perform bivariate analysis to study the strength of the correlation among your variables. Hint: use scatter plots or correlation matrix methods to explain the relationship between two variables. Report and interpret the results.

In [61]:
```
# Get all numeric variables except "adjVolume" since it is redundant
numeric_columns = numeric_columns.drop(columns={"adjVolume"})
numeric_columns.corr()
```

Out[61]:

	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	divCash
close	1.000000	0.999446	0.999473	0.999070	-0.139941	0.999710	0.999118	0.999276	0.998848	0.031930
high	0.999446	1.000000	0.999146	0.999503	-0.124726	0.999251	0.999714	0.999033	0.999328	0.031200
low	0.999473	0.999146	1.000000	0.999488	-0.152665	0.999135	0.998759	0.999703	0.999171	0.031922
open	0.999070	0.999503	0.999488	1.000000	-0.138214	0.998801	0.999148	0.999265	0.999706	0.032833
volume	-0.139941	-0.124726	-0.152665	-0.138214	1.000000	-0.141219	-0.126855	-0.153230	-0.139630	-0.017023
adjClose	0.999710	0.999251	0.999135	0.998801	-0.141219	1.000000	0.999497	0.999522	0.999160	0.032559
adjHigh	0.999118	0.999714	0.998759	0.999148	-0.126855	0.999497	1.000000	0.999226	0.999551	0.031844
adjLow	0.999276	0.999033	0.999703	0.999265	-0.153230	0.999522	0.999226	1.000000	0.999538	0.032560
adjOpen	0.998848	0.999328	0.999171	0.999706	-0.139630	0.999160	0.999551	0.999538	1.000000	0.033402
divCash	0.031930	0.031200	0.031922	0.032833	-0.017023	0.032559	0.031844	0.032560	0.033402	1.000000

We also see a lot of high correlation coefficients, so it is worth checking which variables have a redundant corresponding adjusted column.

In [62]:
```
# Check for rows with different normal and adjusted values
print("Rows with different \"high\" and \"adjHigh\" values:", len(df[df["volume"] != df["adjVolume"]]))
print("Rows with different \"low\" and \"adjLow\" values:", len(df[df["low"] != df["adjLow"]]))
print("Rows with different \"open\" and \"adjOpen\" values:", len(df[df["open"] != df["adjOpen"]]))
print("Rows with different \"close\" and \"adjClose\" values:", len(df[df["close"] != df["adjClose"]]))
```

```
Rows with different "high" and "adjHigh" values: 0
Rows with different "low" and "adjLow" values: 1247
Rows with different "open" and "adjOpen" values: 1247
Rows with different "close" and "adjClose" values: 1247
```

In [63]:
```
# Drop "adjHigh"
numeric_columns.drop(columns={"adjHigh"}, inplace=True)
```

In [64]:
```
# Create new correlation matrix
numeric_columns.corr()
```

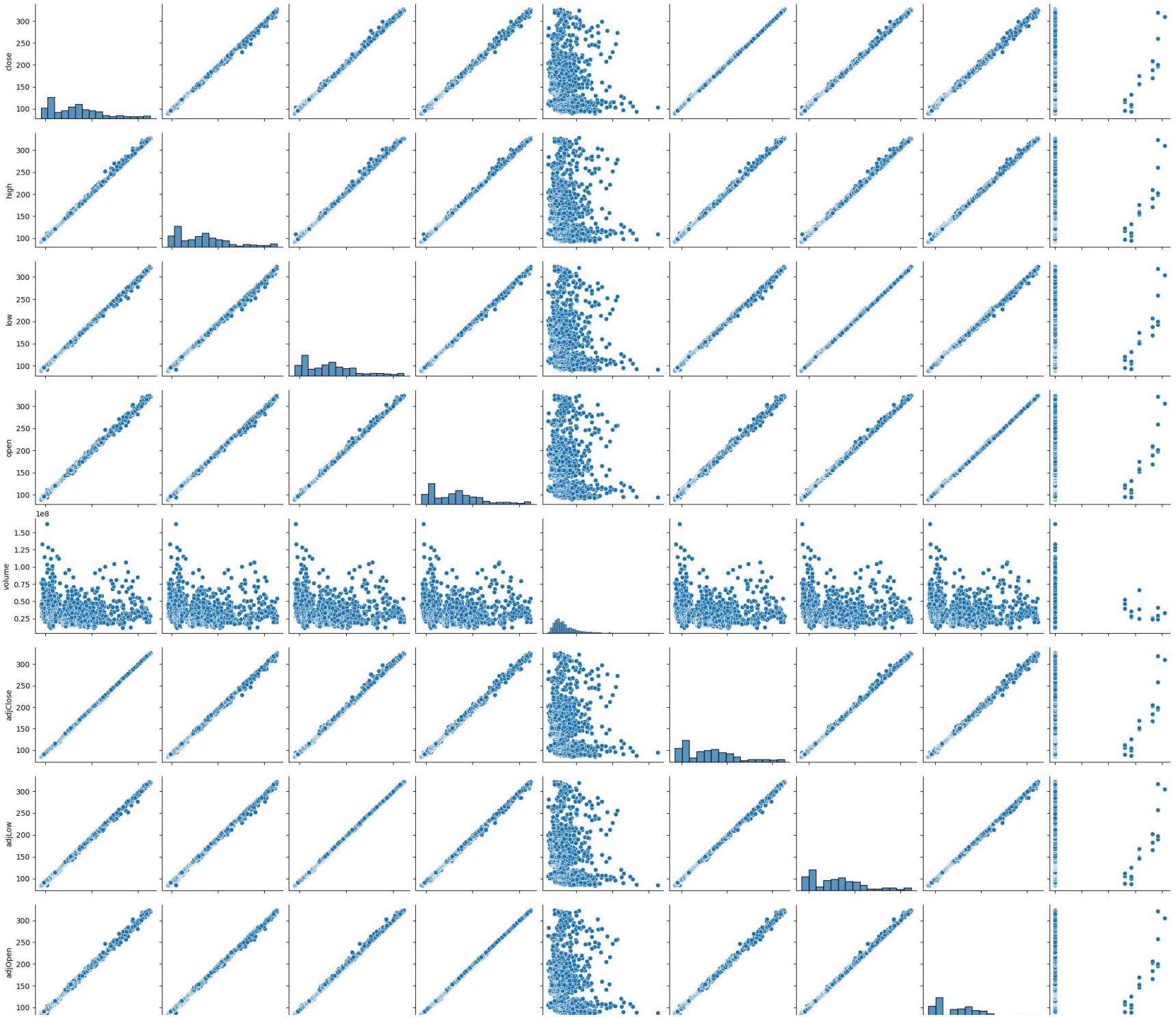
Out[64]:

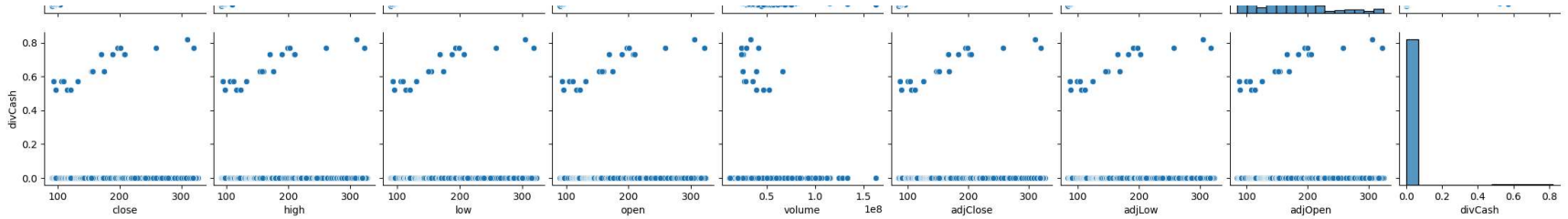
	close	high	low	open	volume	adjClose	adjLow	adjOpen	divCash
close	1.000000	0.999446	0.999473	0.999070	-0.139941	0.999710	0.999276	0.998848	0.031930
high	0.999446	1.000000	0.999146	0.999503	-0.124726	0.999251	0.999033	0.999328	0.031200
low	0.999473	0.999146	1.000000	0.999488	-0.152665	0.999135	0.999703	0.999171	0.031922
open	0.999070	0.999503	0.999488	1.000000	-0.138214	0.998801	0.999265	0.999706	0.032833
volume	-0.139941	-0.124726	-0.152665	-0.138214	1.000000	-0.141219	-0.153230	-0.139630	-0.017023
adjClose	0.999710	0.999251	0.999135	0.998801	-0.141219	1.000000	0.999522	0.999160	0.032559
adjLow	0.999276	0.999033	0.999703	0.999265	-0.153230	0.999522	1.000000	0.999538	0.032560
adjOpen	0.998848	0.999328	0.999171	0.999706	-0.139630	0.999160	0.999538	1.000000	0.033402
divCash	0.031930	0.031200	0.031922	0.032833	-0.017023	0.032559	0.032560	0.033402	1.000000

We still see a lot of very large correlation coefficients (>0.999), so it's clear that multicollinearity could be a large problem during modeling. The only variables that aren't strongly correlated with most other variables are `volume` and `divCash`, by the looks of it.

In [65]:
```
# Show visual representation of correlation matrix
sns.pairplot(numeric_columns)
plt.show()
```

```
C:\Users\jdh10\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```


close
high
low
open
volume
adjClose
adjLow
adjOpen
1e8
1.50
1.25
1.00
0.75
0.50
0.25
300
250
200
150
100



Task 9

Based on the results reported in the previous step, perform data transformation, as needed, to transform a relationship from nonlinear to linear. Explain your decision on the transformation method used here.

Looking at the pairplot, we see that all relationships are linear, even if those between `volume` and other variables have rather weak correlation coefficients. Thus, no transformations are necessary to convert nonlinear relationships to linear ones.

Task 10

Show and explain how you would handle qualitative data in your dataset. Hint: use Dummy variables.

In [66]:
```
df.head(1)
```

Out[66]:

	date	close	high	low	open	volume	adjClose	adjHigh	adjLow	adjOpen	adjVolume	divCash
0	2015-05-27 00:00:00+00:00	132.045	132.26	130.05	130.34	45833246	121.682558	121.880685	119.844118	120.11136	45833246	0.0

There is only one non-numeric variable in my dataset (`date`), and it is not categorical, so dummy variables wouldn't be useful for representing it.

However, let's illustrate the application of dummy variables with an example. Let's say that hypothetically we had a dataset of ice cream flavors and their prices. Suppose the flavors are "vanilla", "chocolate" and "strawberry". We could use dummy variables to represent these flavors.

In [67]:
```
# Define flavors and prices
flavors = ["vanilla", "chocolate", "strawberry"]
prices = [1.99, 2.29, 2.49]

# Assemble dataframe
ice_cream_df = pd.DataFrame({
    "flavors": flavors,
    "prices": prices
})

ice_cream_df.head()
```

Out[67]:

	flavors	prices
0	vanilla	1.99
1	chocolate	2.29
2	strawberry	2.49

In [68]:
```
# Encode "flavors" as dummy variables
flavor_dummies = pd.get_dummies(ice_cream_df["flavors"])
prices = ice_cream_df["prices"]
ice_cream_df = pd.concat([flavor_dummies, prices], axis=1)
ice_cream_df.head()
```

Out[68]:

	chocolate	strawberry	vanilla	prices
0	False	False	True	1.99
1	True	False	False	2.29
2	False	True	False	2.49

Now, we see that we have a binary column for each flavor, `chocolate`, `strawberry` and `vanilla`, each representing whether the current entry is that flavor or not. This "dummification" process is helpful for converting categorical variables into a numeric/boolean form that can be used by a model.

Task 11

Test for multicollinearity in the dataset using the variance_inflation_factor() function. Report the VIF values and interpret the results.

In [71]:
```
numeric_columns.head(1)
```

Out[71]:

	close	high	low	open	volume	adjClose	adjLow	adjOpen	divCash
0	132.045	132.26	130.05	130.34	45833246	121.682558	119.844118	120.11136	0.0

In [70]:
```
# Find VIF values for each numeric variable
vif_data = pd.DataFrame()
vif_data["variable"] = numeric_columns.columns
vif_data["VIF"] = [variance_inflation_factor(numeric_columns.values, i) for i in range(len(numeric_columns.columns))]
vif_data.head()
```

Out[70]:

	variable	VIF
0	close	2.261638e+07
1	high	2.918967e+04
2	low	4.996254e+07
3	open	2.480743e+07
4	volume	1.137087e+01

It appears that `close` and `adjClose`, `low` and `adjLow`, and `open` and `adjOpen` are duplicates of each other, since they don't both show up in the VIF dataframe, probably because of perfect multicollinearity.

However, we see that the normal variables all have very high VIFs, with the lowest being for `volume` at ~11.4. A VIF value over 10 indicates that a variable has strong multicollinearity issues, so as I suspected earlier, multicollinearity is a systematic problem in this dataset. This is a problem that will need to be remedied at a future point if this data is to be suitable for modeling purposes.

References

ChatGPT. (n.d.). https://chat.openai.com/

Bing AI - Bing. (n.d.-b). Bing. https://www.bing.com/search?form=NTPCHB&q=Bing+AI&showconv=1