Course: HDIP in Sci in Data Analytics for Business

Module: Data Preparation

Continuous Assessment/DataPrep-CA2

- Student: Eoin Cantwell
- Student Number: sba21230

Title: Analysis and Report of the Tesla data set

The following areas will be looked into;

- Characterisation of the data set
- Application of Data preparation/evaluation methods and EDA visualizations
- PCA implementation
- Explain the "Curse of Dimensionality "
- Conclusions and Findings
- References

Introduction:

Perform an investigation into the data set provided to us that holds certain stock market activity on the company Tesla (TSLA) and do this using python in a Jupyter Notebook.

1.0 Data Characterisation

In [1]:
```
#Importing appropriate libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.ticker as ticker
from datetime import datetime
```

In order for the code to work in python, the relevant libraries are required to be imported so each task can be performed.

In [2]:
```
tesla_df = pd.read_csv("data/TSLA_B.csv")
#Uploading data from CSV file and into a dataframe called tesla_df
```

The Tesla data set we are using is in the csv format, which is an acronym for the full name "Comma Separated Value". The name of my dataframe is called tesla_df. Throughout the entirety of this notebook, lower case will be used in code construction as this is the advised best practice. To the right of the equals sign is how the data is read into Jupyter notebook and it shows where the data is stored on my personal hard drive and the name of the csv file.

1.1 Data Dictionary

Feature	Description
Date	Represents the calendar date of that given days activity
Source	Unknown
Open	The stock price at the opening of the days trading
High	The stocks highest price point on that date
Low	The stocks lowest price point on that date
Close	The stock price at the end of the days trading
Adj Close	The adjusted stock price after companys and hedge funds activity when the stock is not tradeable
Volume	The quantity of stocks traded on that date, ie the number stocks bought and sold

A data dictionary is a record of what each variable/element is defined as. This is an important guide for the user and provides information on the variables/features to a person who opens the notebook for the first time.

In [3]:
```
tesla_df.info()
#Summary of the dataset:  rows,  columns, null values
#Alternative code to see the types is tesla_df.dtypes
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2416 entries, 0 to 2415
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       2416 non-null   object
 1   Source     2413 non-null   object
 2   Open       2416 non-null   float64
 3   High       2416 non-null   float64
 4   Low        2416 non-null   float64
 5   Close      2416 non-null   float64
 6   Adj Close  2384 non-null   float64
 7   Volume     2416 non-null   int64
dtypes: float64(5), int64(1), object(2)
memory usage: 151.1+ KB
```

The above code helps provide a synopsis of key information of the data in the data set, understanding what each data value represents is essential when formulating and creating code.

In [4]:
```
tesla_df.head()
```

Out[4]:

	Date	Source	Open	High	Low	Close	Adj Close	Volume
0	29/06/2010	0.70	19.000000	25.00	17.540001	23.889999	23.889999	18766300
1	30/06/2010	0.40	25.790001	30.42	23.299999	23.830000	23.830000	17187100
2	01/07/2010	0.10	25.000000	25.92	20.270000	21.959999	21.959999	8218800
3	02/07/2010	0.20	23.000000	23.10	18.709999	19.200001	19.200001	5139800
4	06/07/2010	0.60	20.000000	20.00	15.830000	16.110001	16.110001	6866900

The above code provides a table with the data displayed simply. It represents the first 5 rows of data taken from the data set, with the starting point always beginning at 0. If you wanted to see more rows, you insert the number you would like into the brackets and that number of rows will appear when you run the code. The max number of rows you can see together in this dataset is 60.

In [5]:
```
tesla_df.tail()
```

Out[5]:

	Date	Source	Open	High	Low	Close	Adj Close	Volume
2411	28/01/2020	0.10	568.489990	576.809998	558.080017	566.900024	566.900024	11788500
2412	29/01/2020	0.10	575.690002	589.799988	567.429993	580.989990	580.989990	17801500
2413	30/01/2020	0.40	632.419983	650.880005	618.000000	640.809998	640.809998	29005700
2414	31/01/2020	0.60	640.000000	653.000000	632.520020	650.570007	650.570007	15719300
2415	03/02/2020	0.40	673.690002	786.140015	673.520020	780.000000	780.000000	47065000

This is the opposite to the function tesla_df.head(), therefore this gives you the last 5 rows of data in your data set.

In [6]:
```
tesla_df.describe()
```

Out[6]:

	Open	High	Low	Close	Adj Close	Volume
count	2416.000000	2416.000000	2416.000000	2416.000000	2384.000000	2.416000e+03
mean	186.271147	189.578224	182.916639	186.403651	187.252651	5.572722e+06
std	118.740163	120.892329	116.857591	119.136020	118.619900	4.987809e+06
min	16.139999	16.629999	14.980000	15.800000	15.800000	1.185000e+05
25%	34.342498	34.897501	33.587501	34.400002	34.605001	1.899275e+06
50%	213.035003	216.745002	208.870002	212.960007	213.744995	4.578400e+06
75%	266.450012	270.927513	262.102501	266.774994	266.455002	7.361150e+06
max	673.690002	786.140015	673.520020	780.000000	780.000000	4.706500e+07

The above function allows you to gain considerable insights as it uses key statistical formulae's. Information provided is count = number of rows of data in each column, mean is the average, std is the standard deviation, min gives you lowest value, max gives you highest value.

In [7]:
```
tesla_df.corr()
```

Out[7]:

	Open	High	Low	Close	Adj Close	Volume
Open	1.000000	0.999425	0.999575	0.998886	0.998871	0.501762
High	0.999425	1.000000	0.999389	0.999640	0.999635	0.512944
Low	0.999575	0.999389	1.000000	0.999447	0.999437	0.493496
Close	0.998886	0.999640	0.999447	1.000000	1.000000	0.505169
Adj Close	0.998871	0.999635	0.999437	1.000000	1.000000	0.500380
Volume	0.501762	0.512944	0.493496	0.505169	0.500380	1.000000

Shows the correlation between variables, if there is any relationship either positive or negative and can we make out any inferences between the data

In [8]:
```
skewness = tesla_df.skew()
print(skewness)
```

```
Open        -0.014380
High         0.011923
Low         -0.010357
Close        0.016514
Adj Close    0.007839
Volume       2.165242
dtype: float64
```

The above code gives us a review of the skewness of our data, this will show us how flat/curved the data and will potentially highlight if outliers are present or not.

In [9]:
```
tesla_df.shape
```

Out[9]: (2416, 8)

There are 2416 observations with 8 dimensions in the data set, this is another way of understanding how many rows and columns we are dealing with and confirms what we already knew when we ran tesla_df.info().

In [10]:
```
tesla_df['Date']
```

Out[10]:
```
0       29/06/2010
1       30/06/2010
2       01/07/2010
3       02/07/2010
4       06/07/2010
           ...
2411    28/01/2020
2412    29/01/2020
2413    30/01/2020
2414    31/01/2020
2415    03/02/2020
Name: Date, Length: 2416, dtype: object
```

As this is data set involves time series data, it is good to get an understanding of what date the data set starts and finishes at. By using this function we can see that the start of our observed rows of data is 29/06/2010 and the end is 03/02/2020. This is essential information to know as it shows us the period of when the data was taken and the span of the data. We could check for possible information in the news for any price spikes/slumps. From this we can see our data spans just under 10 years and it does not include data from when the Covid-19 pandemic started.

In [11]:
```
tesla_df.count()
```

Out[11]:
```
Date         2416
Source       2413
Open         2416
High         2416
Low          2416
Close        2416
Adj Close    2384
Volume       2416
dtype: int64
```

The above code counts the total values of each feature variable.

In [12]:
```
tesla_df.isnull().sum()
```

Out[12]:
```
Date          0
Source        3
Open          0
High          0
Low           0
Close         0
Adj Close    32
Volume        0
dtype: int64
```

Here we find out about any null values in the data set. Being aware of Null values in a data set is critical to understand as they could play havoc on results if they are not noted and suitably dealt with. From the above we can see there are 3 null values for the feature source and 32 for the feature volume.

In [13]:
```
tesla_df[tesla_df['Source'].isnull()]
```

Out[13]:

	Date	Source	Open	High	Low	Close	Adj Close	Volume
47	03/09/2010	NaN	20.870001	21.299999	20.660000	21.049999	21.049999	434600
224	18/05/2011	NaN	26.100000	26.469999	25.520000	26.350000	26.350000	729500
2408	23/01/2020	NaN	564.250000	582.000000	555.599976	572.200012	572.200012	19651000

This shows us the three columns in "Source" feature that are NA.

In [14]:
```
tesla_df[tesla_df["Adj Close"].isnull()]
```

Out[14]:

	Date	Source	Open	High	Low	Close	Adj Close	Volume
60	23/09/2010	0.70	19.889999	20.139999	19.500000	19.559999	NaN	668100
78	19/10/2010	0.20	20.200001	20.410000	20.000000	20.049999	NaN	245200
96	12/11/2010	0.50	28.250000	30.500000	28.070000	29.840000	NaN	2729100
97	15/11/2010	0.60	30.219999	32.939999	30.219999	30.799999	NaN	2622900
115	10/12/2010	0.70	32.049999	32.919998	31.129999	31.520000	NaN	429400
138	13/01/2011	0.30	26.959999	26.969999	26.160000	26.219999	NaN	723600
159	14/02/2011	0.70	23.639999	24.139999	23.049999	23.080000	NaN	1283100
171	03/03/2011	0.30	24.480000	24.790001	24.059999	24.360001	NaN	640200
172	04/03/2011	0.70	24.480000	24.990000	23.780001	24.950001	NaN	1580100
173	07/03/2011	0.60	24.930000	25.400000	24.700001	24.940001	NaN	2033600
174	08/03/2011	0.50	24.600000	24.959999	24.000000	24.660000	NaN	1399900
175	09/03/2011	0.50	24.660000	24.990000	24.270000	24.719999	NaN	924800
176	10/03/2011	0.30	24.440001	24.490000	23.730000	24.010000	NaN	1017000
195	06/04/2011	0.30	26.990000	27.010000	25.799999	26.490000	NaN	1288300
196	07/04/2011	0.60	26.850000	27.940001	26.450001	27.240000	NaN	2810300
197	08/04/2011	0.10	27.580000	27.600000	26.360001	26.490000	NaN	1946400
198	11/04/2011	0.70	26.469999	26.530001	25.020000	25.270000	NaN	1369400
215	05/05/2011	0.40	27.200001	27.440001	26.170000	26.440001	NaN	1218500
216	06/05/2011	0.30	26.900000	27.700001	26.620001	27.120001	NaN	981700
232	31/05/2011	0.10	29.690001	30.280001	29.549999	30.139999	NaN	3290500
233	01/06/2011	0.70	30.000000	30.100000	28.379999	28.520000	NaN	1529900
1211	22/04/2015	0.40	212.500000	221.880005	211.690002	219.440002	NaN	7863000
1965	19/04/2018	0.40	291.079987	301.010010	288.549988	300.079987	NaN	6090600
1966	20/04/2018	0.40	295.170013	299.980011	289.750000	290.239990	NaN	5627900
1967	23/04/2018	0.20	291.290009	291.619995	282.329987	283.369995	NaN	4893400
1968	24/04/2018	0.30	285.000000	287.089996	278.459991	283.459991	NaN	5685300
1969	25/04/2018	0.60	283.500000	285.160004	277.250000	280.690002	NaN	4013600
1987	21/05/2018	0.20	281.329987	291.489990	281.299988	284.489990	NaN	9182600
1988	22/05/2018	0.50	287.760010	288.000000	273.420013	275.010010	NaN	8945800
2011	25/06/2018	0.10	330.119995	338.470001	327.500000	333.010010	NaN	6931300
2385	18/12/2019	0.40	380.630005	395.220001	380.579987	393.149994	NaN	14121000
2396	06/01/2020	0.20	440.470001	451.559998	440.000000	451.540009	NaN	10133000

This shows us the three columns in the "Adj Close" feature that are NA.

In [15]:
```
Q1 = tesla_df.quantile(0.25)
Q3 = tesla_df.quantile(0.75)
IQR = Q3 - Q1
print(IQR)
```

```
Open         2.321075e+02
High         2.360300e+02
Low          2.285150e+02
Close        2.323750e+02
Adj Close    2.318500e+02
Volume       5.461875e+06
dtype: float64
```

The above code provides the interquartile range.

2.0 Data Cleaning

In [16]:
```
#parse strings to datetime type
tesla_df['Date'] = pd.to_datetime(tesla_df['Date'], infer_datetime_format=True)
indexedDataset = tesla_df.set_index(['Date'])
tesla_df.head(5)
```

Out[16]:

	Date	Source	Open	High	Low	Close	Adj Close	Volume
0	2010-06-29	0.70	19.000000	25.00	17.540001	23.889999	23.889999	18766300
1	2010-06-30	0.40	25.790001	30.42	23.299999	23.830000	23.830000	17187100
2	2010-07-01	0.10	25.000000	25.92	20.270000	21.959999	21.959999	8218800
3	2010-07-02	0.20	23.000000	23.10	18.709999	19.200001	19.200001	5139800
4	2010-07-06	0.60	20.000000	20.00	15.830000	16.110001	16.110001	6866900

This code changes the type of data the feature "Date" represents, it has changed from an object to datetime.This is done so the computer knows the values represent actual dates and they can be worked with in graphs.

In [17]:
```
tesla_df.dtypes
```

Out[17]:
```
Date         datetime64[ns]
Source               object
Open                float64
High                float64
Low                 float64
Close               float64
Adj Close           float64
Volume                int64
dtype: object
```

In [18]:
```
tesla_df = pd.read_csv('data/TSLA_B.csv', parse_dates=[0], index_col=0, squeeze=True)
tesla_df.head()
```

Out[18]:

	Source	Open	High	Low	Close	Adj Close	Volume
Date							
2010-06-29	0.70	19.000000	25.00	17.540001	23.889999	23.889999	18766300
2010-06-30	0.40	25.790001	30.42	23.299999	23.830000	23.830000	17187100
2010-01-07	0.10	25.000000	25.92	20.270000	21.959999	21.959999	8218800
2010-02-07	0.20	23.000000	23.10	18.709999	19.200001	19.200001	5139800
2010-06-07	0.60	20.000000	20.00	15.830000	16.110001	16.110001	6866900

The above line of code changes the index to Date, this is useful to do as it helps with creating code reliant on the date.

In [19]:
```
tesla_df.dtypes
```

Out[19]:
```
Source        object
Open         float64
High         float64
Low          float64
Close        float64
Adj Close    float64
Volume         int64
dtype: object
```

In [20]:
```
#As Source is unknown quantity, the decision was taking to drop the column altogether
to_drop = ['Source']
tesla_df.drop(to_drop, inplace=True, axis=1)
tesla_df.head()
```

Out[20]:

	Open	High	Low	Close	Adj Close	Volume
Date						
2010-06-29	19.000000	25.00	17.540001	23.889999	23.889999	18766300
2010-06-30	25.790001	30.42	23.299999	23.830000	23.830000	17187100
2010-01-07	25.000000	25.92	20.270000	21.959999	21.959999	8218800
2010-02-07	23.000000	23.10	18.709999	19.200001	19.200001	5139800
2010-06-07	20.000000	20.00	15.830000	16.110001	16.110001	6866900

As the feature "Source" is unknown, the decision was made to drop this as it may affect results.

In [21]:
```
#renaming columns into all lower cases
new_names =  {'Date': 'date', 'Open':'open', 'High':'high',
              'Low':'low', 'Close':'close', 'Adj Close':'adj close', 'Volume': 'volum
```

In [22]:
```
tesla_df.rename(columns=new_names, inplace=True)
tesla_df.head()
```

Out[22]:

	open	high	low	close	adj close	volume
Date						
2010-06-29	19.000000	25.00	17.540001	23.889999	23.889999	18766300
2010-06-30	25.790001	30.42	23.299999	23.830000	23.830000	17187100
2010-01-07	25.000000	25.92	20.270000	21.959999	21.959999	8218800
2010-02-07	23.000000	23.10	18.709999	19.200001	19.200001	5139800
2010-06-07	20.000000	20.00	15.830000	16.110001	16.110001	6866900

The above procedure changed all the upper cases of the features on the x axis to lower cases to help with code creation and errors appearing for incorrectly using capitals down the line.

In [23]:
```
tesla_df.isnull().sum()
```

Out[23]:
```
open          0
high          0
low           0
close         0
adj close    32
volume        0
dtype: int64
```

As we have dropped the source column, the only remaining null values are in the adj close column. Now we will remove these.

In [24]:
```
tesla_df.dropna(subset = ['adj close'],how = "any")
```

Out[24]:

	open	high	low	close	adj close	volume
Date						
2010-06-29	19.000000	25.000000	17.540001	23.889999	23.889999	18766300
2010-06-30	25.790001	30.420000	23.299999	23.830000	23.830000	17187100
2010-01-07	25.000000	25.920000	20.270000	21.959999	21.959999	8218800
2010-02-07	23.000000	23.100000	18.709999	19.200001	19.200001	5139800
2010-06-07	20.000000	20.000000	15.830000	16.110001	16.110001	6866900
...	...	...	...	...	...	...
2020-01-28	568.489990	576.809998	558.080017	566.900024	566.900024	11788500
2020-01-29	575.690002	589.799988	567.429993	580.989990	580.989990	17801500
2020-01-30	632.419983	650.880005	618.000000	640.809998	640.809998	29005700
2020-01-31	640.000000	653.000000	632.520020	650.570007	650.570007	15719300
2020-03-02	673.690002	786.140015	673.520020	780.000000	780.000000	47065000

2384 rows × 6 columns

The above function is one of the few procedures that drop the null values. Handling null values is a delicate issue, especially with Time Series Data. Null values should be dealt with differently. Some of the ways are; leaving null values in their place, dropping the rows where null values appear or imputing a figure in place of the na value by using averages.

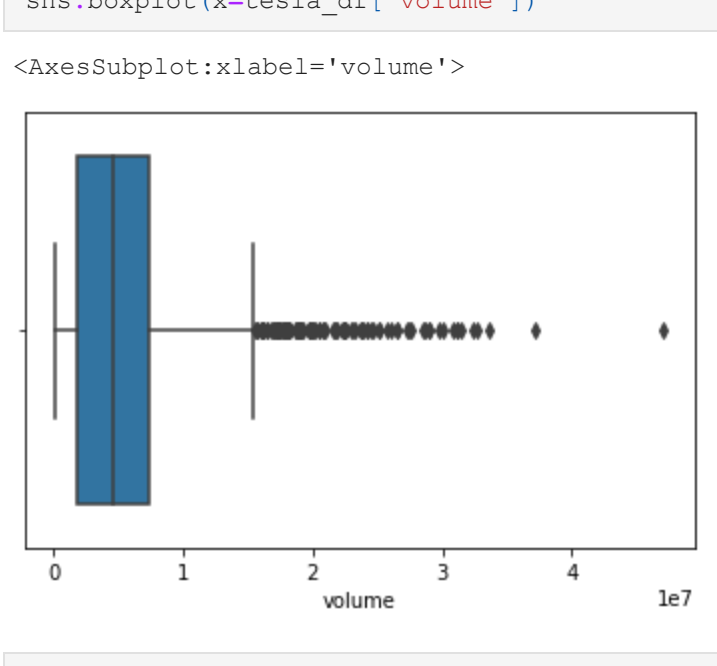
Having researched into null values, it was mentioned that where you have a relatively small few null values, it is advised to drop the rows and that is what was decided to do for this dataset.

3.0 EDA Visualizations

The following boxplot graphs shows us if outliers are present in each feature.

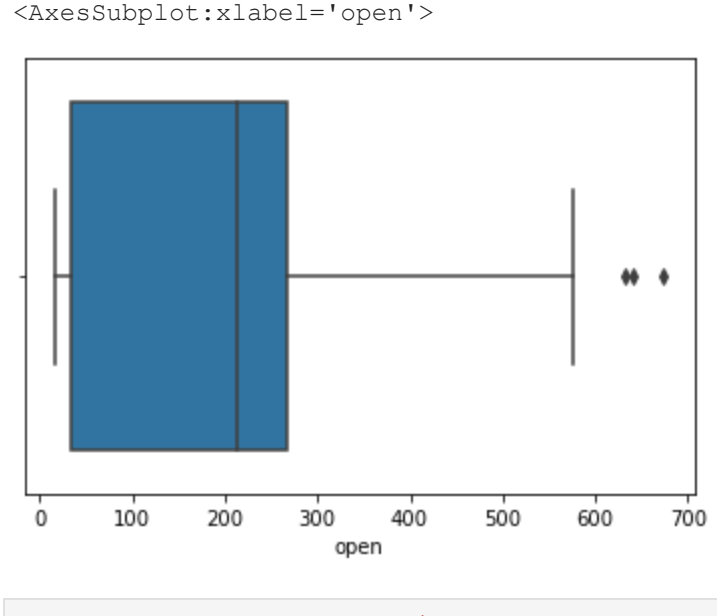
In [25]:
```
sns.boxplot(x=tesla_df['volume'])
```

Out[25]: <AxesSubplot:xlabel='volume'>



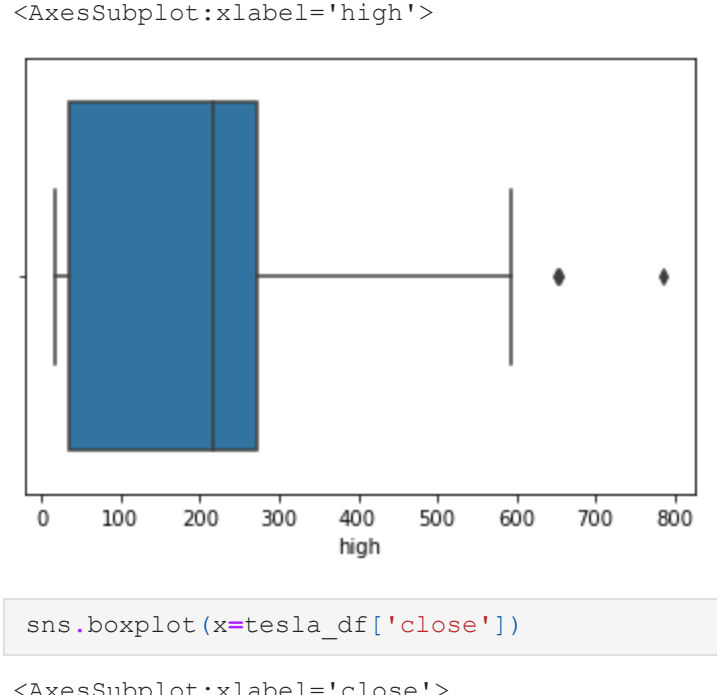
In [26]:
```
sns.boxplot(x=tesla_df['open'])
```

Out[26]: <AxesSubplot:xlabel='open'>



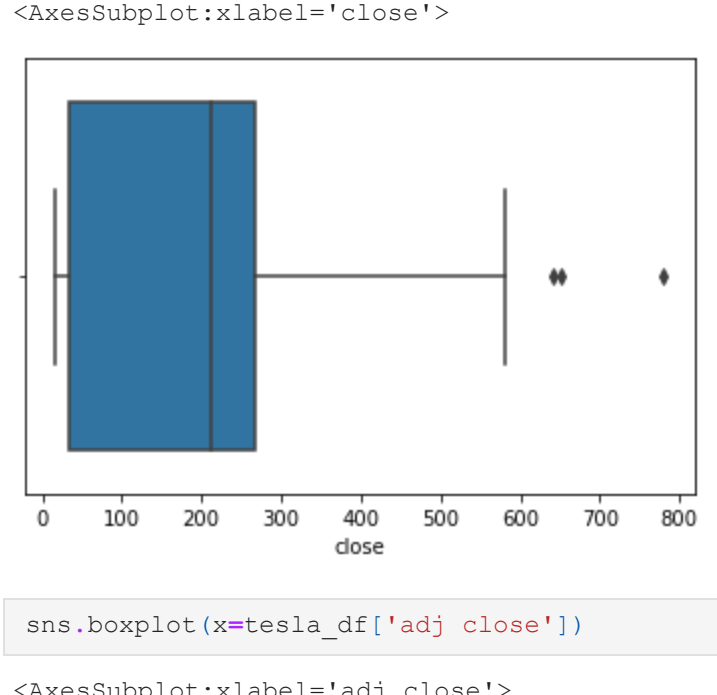
In [27]:
```
sns.boxplot(x=tesla_df['high'])
```

Out[27]: <AxesSubplot:xlabel='high'>



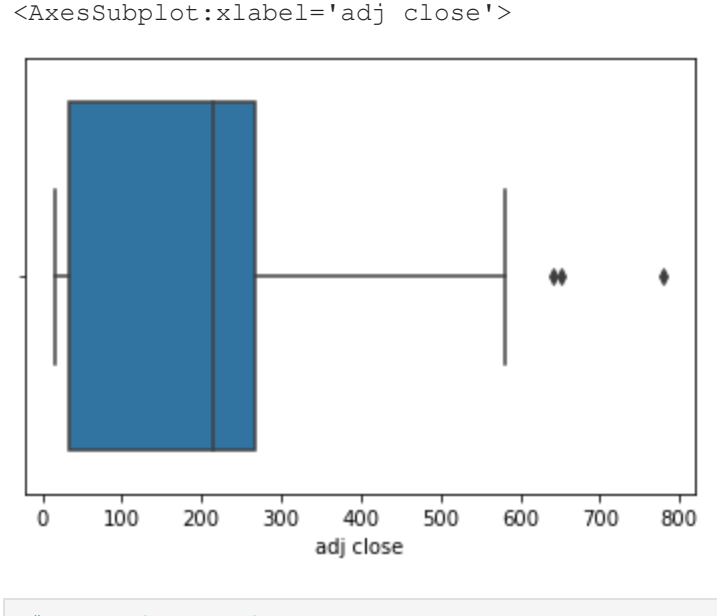
In [28]:
```
sns.boxplot(x=tesla_df['close'])
```

Out[28]: <AxesSubplot:xlabel='close'>

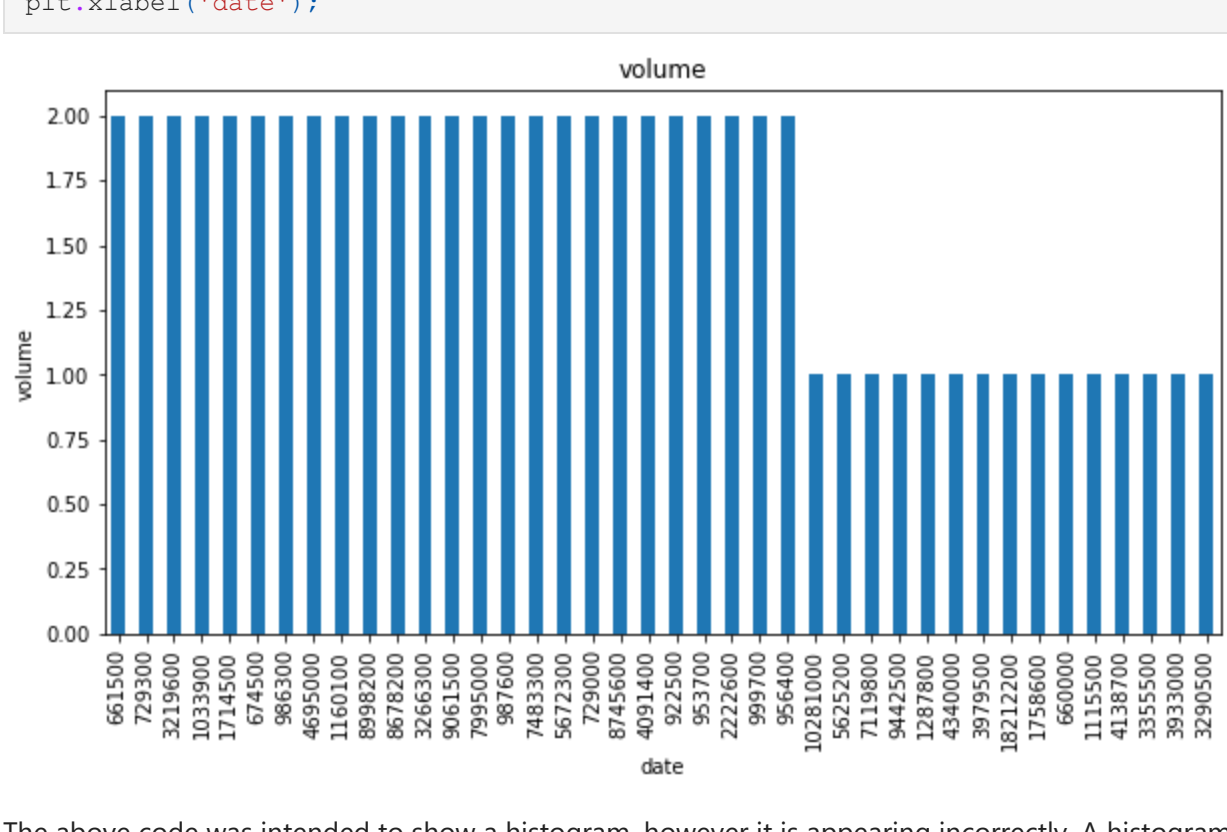


In [29]:
```
sns.boxplot(x=tesla_df['adj close'])
```

Out[29]: <AxesSubplot:xlabel='adj close'>

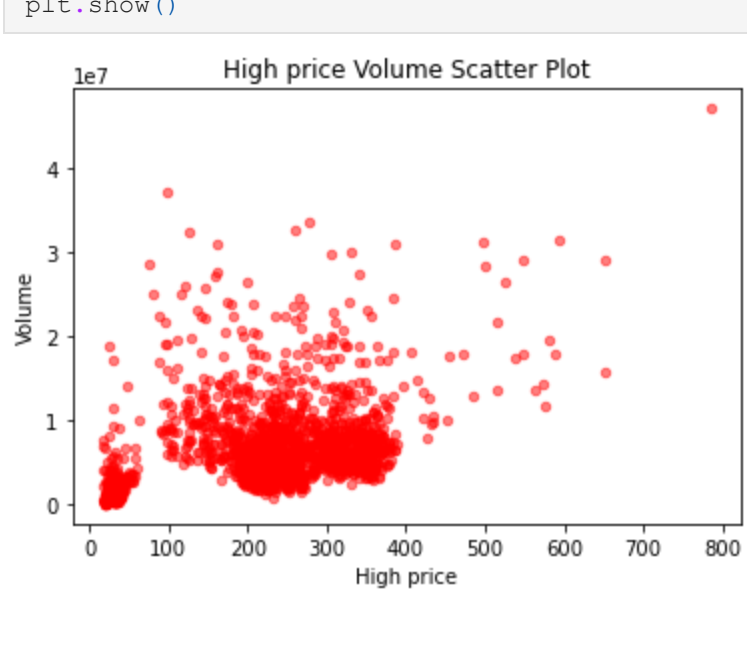


In [30]:
```
# Plotting a Histogram
tesla_df.volume.value_counts().nlargest(40).plot(kind= 'bar', figsize=(10,5))
plt.title('volume')
plt.ylabel ('volume')
plt.xlabel('date');
```



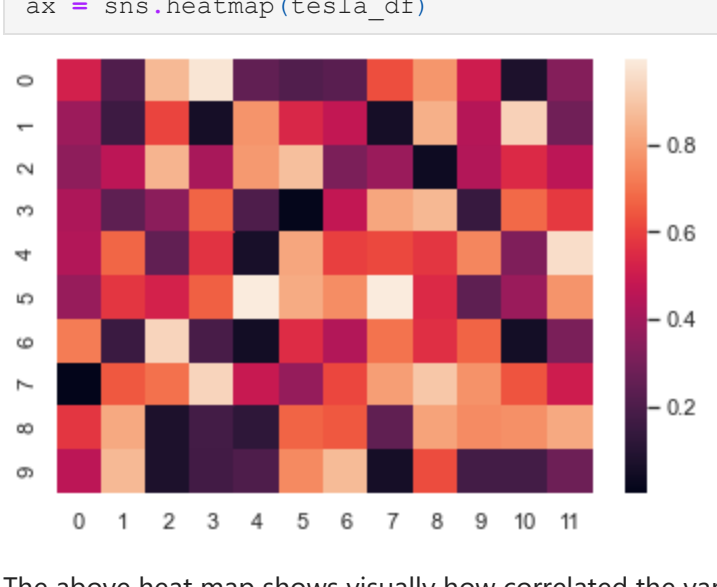
The above code was intended to show a histogram, however it is appearing incorrectly. A histogram helps show a

In [31]:
```
# x = High price, y = Volume
tesla_df.plot(kind='scatter', x='high', y='volume',alpha = 0.5,color = 'red')
plt.xlabel('High price')              # label = name of label
plt.ylabel('Volume')
plt.title('High price Volume Scatter Plot')            # title = title of plot
plt.show()
```



Correlation

In [62]:
```
tesla_df = np.random.rand(10, 12)
ax = sns.heatmap(tesla_df)
```



The above heat map shows visually how correlated the variables are with each other, it helps to uncover if there are relationships or not.

3.1 Understanding Returns

In [32]:
```
tesla_df["adj close-1"] = tesla_df["adj close"].shift(1)
```

This code is done to create a new column and shifts the adj close price of the previous days stock price to one day forward.

In [33]:
```
tesla_df.head()
```

Out[33]:

	open	high	low	close	adj close	volume	adj close-1
Date							
2010-06-29	19.000000	25.00	17.540001	23.889999	23.889999	18766300	NaN
2010-06-30	25.790001	30.42	23.299999	23.830000	23.830000	17187100	23.889999
2010-01-07	25.000000	25.92	20.270000	21.959999	21.959999	8218800	23.830000
2010-02-07	23.000000	23.10	18.709999	19.200001	19.200001	5139800	21.959999
2010-06-07	20.000000	20.00	15.830000	16.110001	16.110001	6866900	19.200001

In [34]:
```
tesla_df['returns_manual'] = (tesla_df['adj close'] / tesla_df['adj close-1']) - 1
```

In [35]:
```
tesla_df.head()
```

Out[35]:

	open	high	low	close	adj close	volume	adj close-1	returns_manual
Date								
2010-06-29	19.000000	25.00	17.540001	23.889999	23.889999	18766300	NaN	NaN
2010-06-30	25.790001	30.42	23.299999	23.830000	23.830000	17187100	23.889999	-0.002511
2010-01-07	25.000000	25.92	20.270000	21.959999	21.959999	8218800	23.830000	-0.078473
2010-02-07	23.000000	23.10	18.709999	19.200001	19.200001	5139800	21.959999	-0.125683
2010-06-07	20.000000	20.00	15.830000	16.110001	16.110001	6866900	19.200001	-0.160937

The above code shows the gain or loss made from the previous day, this helps in the goal of understanding the total returns made.

In [36]:
```
returns = pd.DataFrame()
```

In [37]:
```
returns['Return'] = tesla_df['close'].pct_change()
returns.head()
```

Out[37]:

	Return
Date	
2010-06-29	NaN
2010-06-30	-0.002511
2010-01-07	-0.078473
2010-02-07	-0.125683
2010-06-07	-0.160937

The above .pct_change was used to understand the percentage change of returns made/lost.

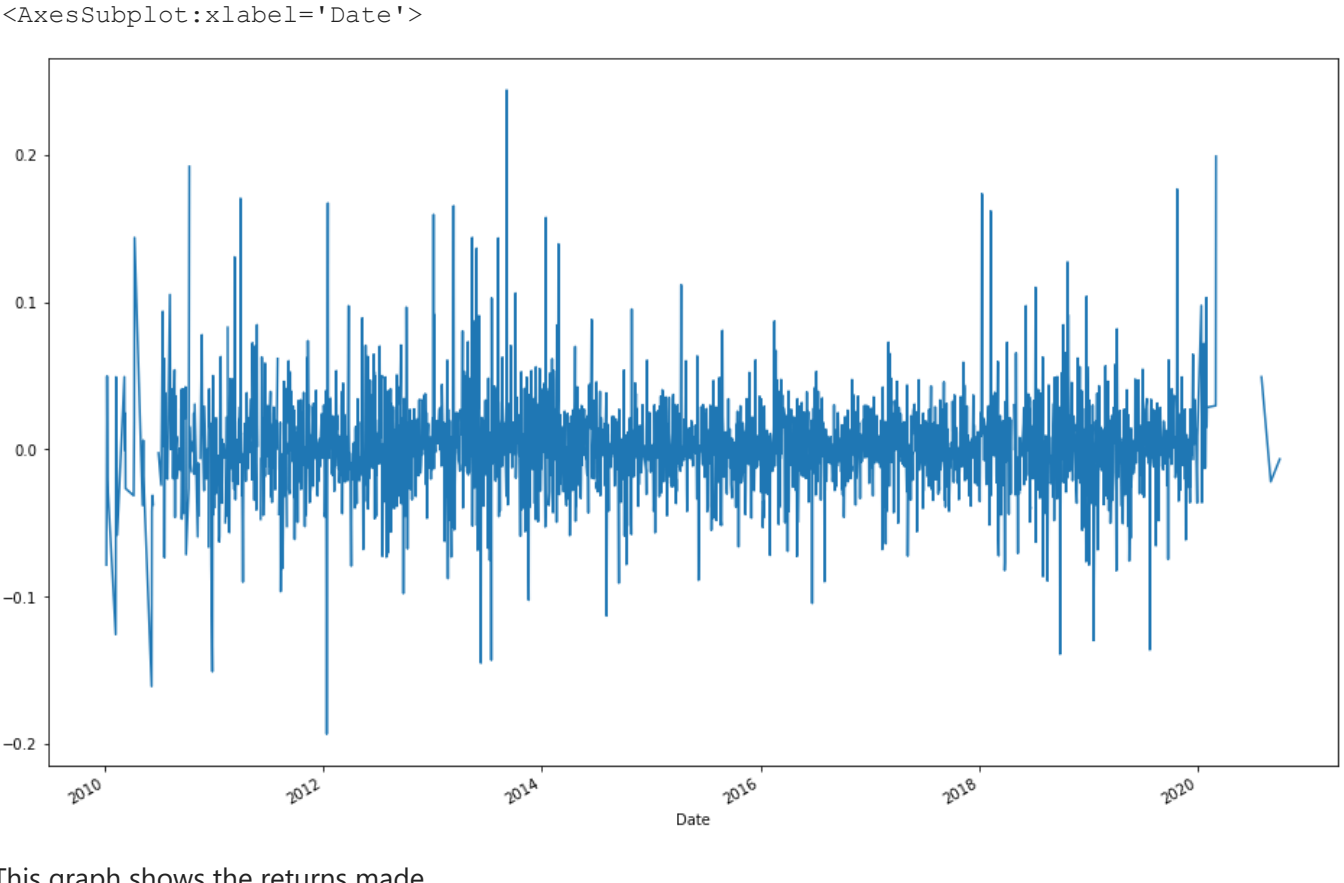
In [38]:
```
tesla_df['returns_pct_change_method'] = tesla_df['adj close'].pct_change(1)
tesla_df.head()
```

Out[38]:

	open	high	low	close	adj close	volume	adj close-1	returns_manual	returns_pct_cha
Date									
2010-06-29	19.000000	25.00	17.540001	23.889999	23.889999	18766300	NaN	NaN	
2010-06-30	25.790001	30.42	23.299999	23.830000	23.830000	17187100	23.889999	-0.002511	
2010-01-07	25.000000	25.92	20.270000	21.959999	21.959999	8218800	23.830000	-0.078473	
2010-02-07	23.000000	23.10	18.709999	19.200001	19.200001	5139800	21.959999	-0.125683	
2010-06-07	20.000000	20.00	15.830000	16.110001	16.110001	6866900	19.200001	-0.160937	

In [39]:
```
tesla_df['returns_manual'].plot(figsize=(16, 10))
```

Out[39]: <AxesSubplot:xlabel='Date'>



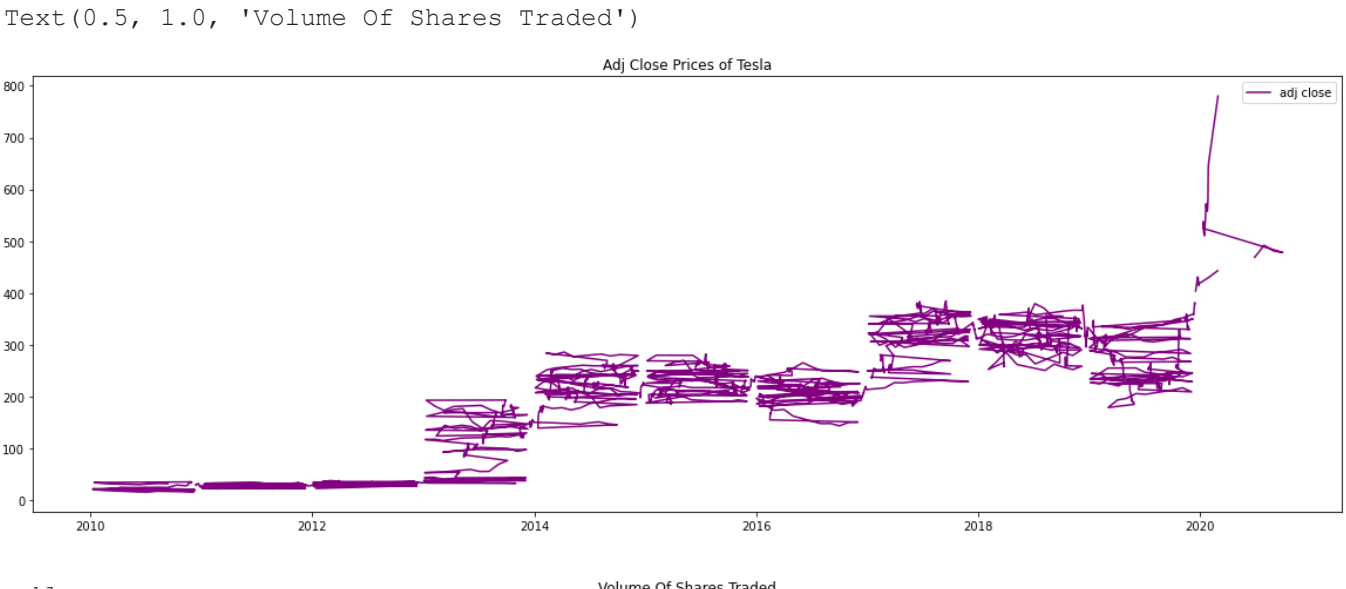
This graph shows the returns made.

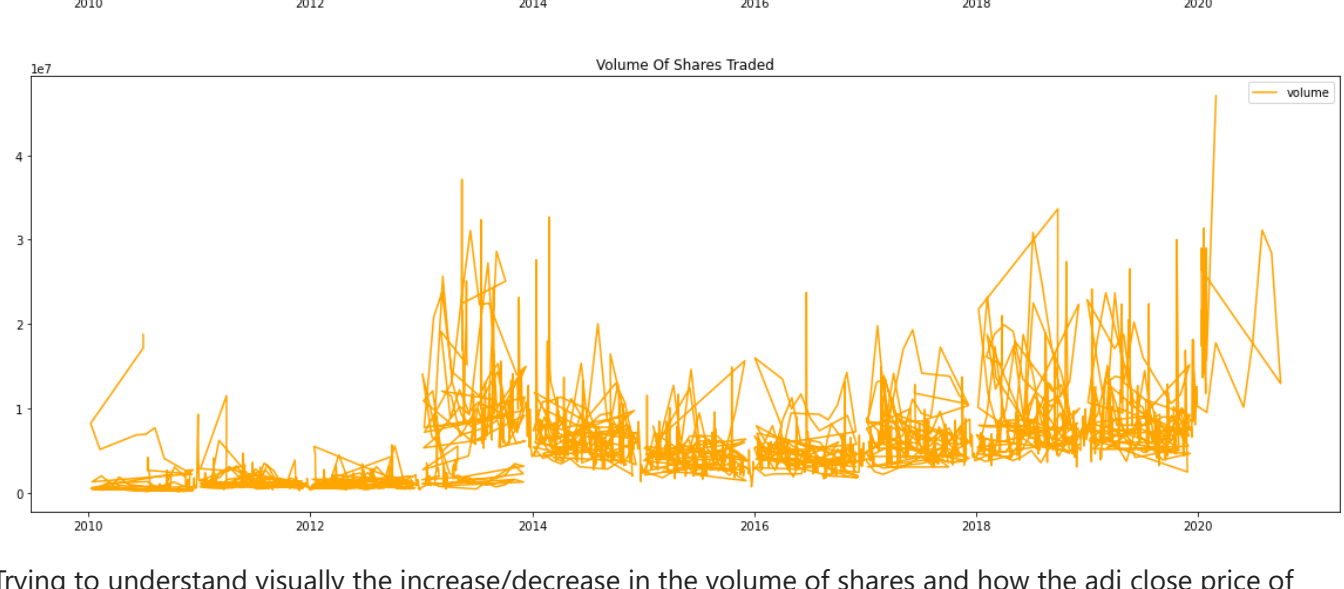
In [40]:
```
# Plot "Adj Close" and "Volume" of Tesla Shares
plt.figure(figsize=(20, 15))

plt.subplot(2,1,1)
plt.plot(tesla_df['adj close'], label='adj close', color="purple")
plt.legend(loc="upper right")
plt.title('Adj Close Prices of Tesla')

plt.subplot(2,1,2)
plt.plot(tesla_df['volume'], label='volume', color="Orange")
plt.legend(loc="upper right")
plt.title('Volume Of Shares Traded')
```

Out[40]: Text(0.5, 1.0, 'Volume Of Shares Traded')



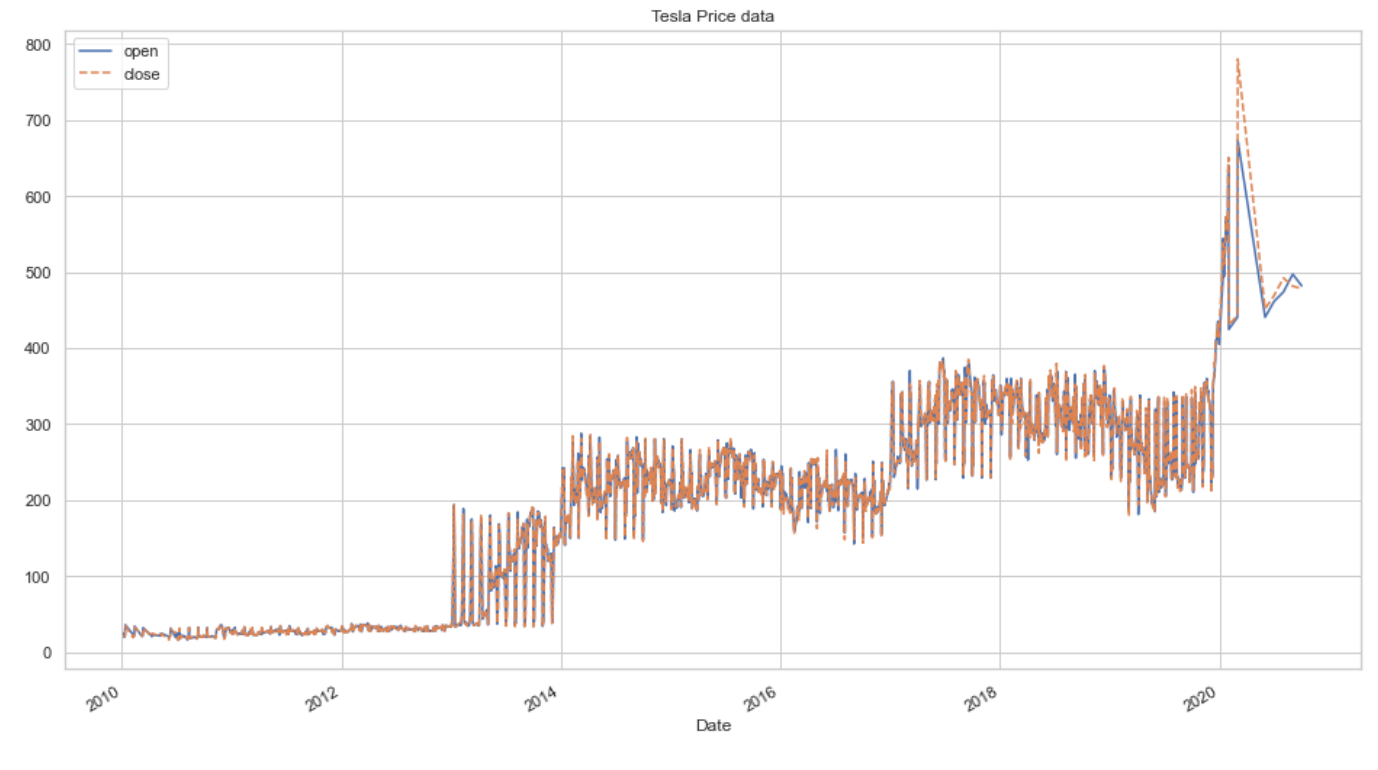


Trying to understand visually the increase/decrease in the volume of shares and how the adj close price of Tesla fluctuates.

In [41]:
```
import seaborn as sns
sns.set_context("poster")
sns.set(rc={"figure.figsize":(16,9.)})
sns.set_style("whitegrid")
```

In [42]:
```
plt.xlabel("Date")
plt.title("Tesla Price data")
tesla_df["open"].plot(style="-")
tesla_df["close"].plot(style="--")
plt.legend(["open","close"],loc="upper left")
```

Out[42]: <matplotlib.legend.Legend at 0x2031c580f70>



Comparing the open vs the close price over the time period.

4. Implement PCA

In [52]:
```
%matplotlib inline
import numpy as np
from sklearn.decomposition import PCA
import seaborn as sns;
import matplotlib.pyplot as plt
sns.set()
```

In [53]:
```
tesla_df.head()
```

Out[53]:

	open	high	low	close	adj close	volume	adj close-1	returns_manual	returns_pc
Date									
2010-06-30	25.790001	30.420000	23.299999	23.830000	23.830000	17187100	23.889999	-0.002511	
2010-01-07	25.000000	25.920000	20.270000	21.959999	21.959999	8218800	23.830000	-0.078473	
2010-02-07	23.000000	23.100000	18.709999	19.200001	19.200001	5139800	21.959999	-0.125683	
2010-06-07	20.000000	20.000000	15.830000	16.110001	16.110001	6866900	19.200001	-0.160937	
2010-07-07	16.400000	16.629999	14.980000	15.800000	15.800000	6921700	16.110001	-0.019243	

In [54]:
```
tesla_df.dtypes
```

Out[54]:
```
open                          float64
high                          float64
low                           float64
close                         float64
adj close                     float64
volume                          int64
adj close-1                   float64
returns_manual                float64
returns_pct_change_method     float64
dtype: object
```

In [55]:
```
from sklearn.preprocessing import StandardScaler
```

In [56]:
```
scaling = StandardScaler()
```

In [57]:
```
scaling.fit_transform(tesla_df[['open', 'high', 'low', 'close', 'adj close', 'volume']
```

Out[57]:
```
array([[-1.37252656, -1.33723823, -1.38705415, -1.38578568, -1.38578568,
         2.32265649],
       [-1.37923098, -1.37475016, -1.41318508, -1.40160579, -1.40160579,
         0.52439735],
       [-1.39620417, -1.39825764, -1.42663864, -1.42495522, -1.42495522,
        -0.09298163],
       ...,
       [ 3.77569745,  3.83490684,  3.74168058,  3.83383174,  3.83383174,
         4.69243734],
       [ 3.840026  ,  3.85257909,  3.86690225,  3.91640089,  3.91640089,
         2.02834377],
       [ 4.12593946,  4.96243221,  4.22048915,  5.01137164,  5.01137164,
         8.31355856]])
```

In [58]:
```
tesla_df.isnull().sum()
```

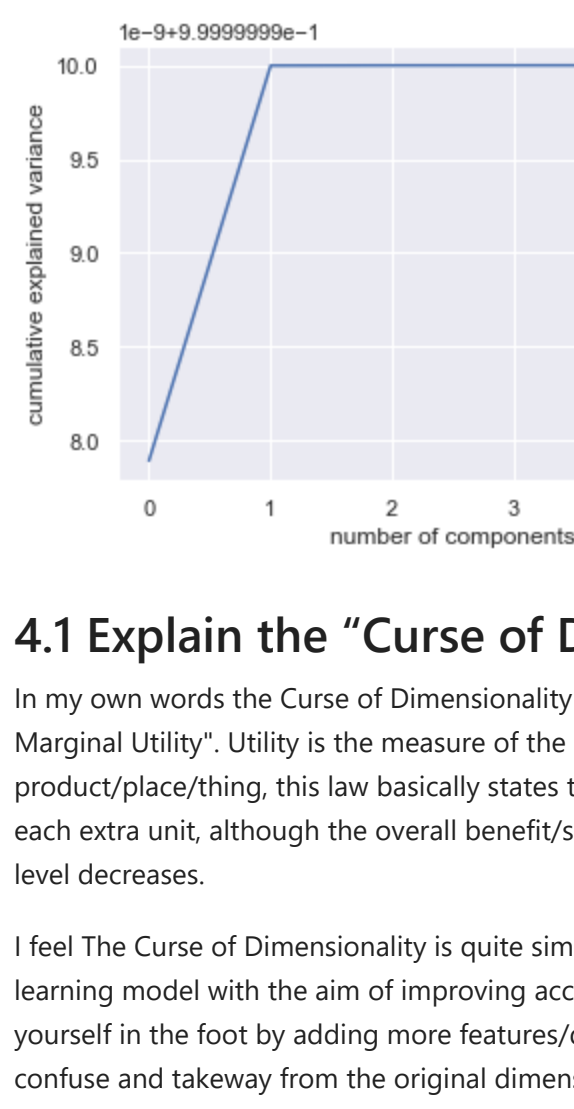
Out[58]:
```
open                          0
high                          0
low                           0
close                         0
adj close                     0
volume                        0
adj close-1                   0
returns_manual                0
returns_pct_change_method     0
dtype: int64
```

In [59]:
```
tesla_df.dropna(subset=['adj close', 'adj close-1', 'returns_manual', 'returns_pct_cha
```

In [60]:
```
tesla_df.isnull().sum()
```

Out[60]:
```
open                          0
high                          0
low                           0
close                         0
adj close                     0
volume                        0
adj close-1                   0
returns_manual                0
returns_pct_change_method     0
dtype: int64
```

In [61]:
```
tsla = tesla_df.drop(["adj close-1", "returns_manual", "returns_pct_change_method"],a
pca = PCA().fit(tsla)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



4.1 Explain the “Curse of Dimensionality ”

In my own words the Curse of Dimensionality is similar to a concept in economics, the "Law of Diminishing Marginal Utility". Utility is the measure of the benefit/satisfaction you receive from a unit of a product/place/thing, this law basically states that as you consume more and more of something, that with each extra unit, although the overall benefit/satisfaction level increases, the unit on unit benefit/satisfaction level decreases.

I feel The Curse of Dimensionality is quite similar to this law, in that as you add dimensions to the machine learning model with the aim of improving accuracy, you come to a point where you could end up shooting yourself in the foot by adding more features/dimensions. The additional dimensions may cause havoc, confuse and takeway from the original dimensions. Therefore you have to weigh up the output and benefit of additional dimensions. Feeding more data into the model might not produce the intended results.

5. Conclusions and Findings of data set

Insights Uncovered

- The data fluctuates a lot over the time period which shows the stock market activity and price did too.