December 21, 2020

Apple Stock

Intrudction

Here is a data set about Apple Stock between the years 2010-2020. In this data set, I would like to explore a couple of features about that stock by using Time Series Analysis (TSA) method to analyze the data and Time Series Forecasting (TSF) to predict future values based on previously observed values.

Data Parameters:

- Date
- Close/Last - Stock price at the closing of the trading day.
- Volume - Number of shares traded during the trading day.
- Open - Stock price at the opening of the trading day.
- High - Maximum stock price during the trading day.
- Low - Minimum stock price during the trading day

In [3]:
```
import math
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
from pandas.plotting import lag_plot
import matplotlib.pyplot as plt
plt.style.use('ggplot')
from numpy.random import normal, seed
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```

Reading the data and defining the date as the index of the data.

In [4]:
```
apple = pd.read_csv(r"C:\Users\Matan\Documents\Python\apple_stocks.csv",index_col="Date", parse_dates=[
"Date"])
apple
```

Out[4]:

Date	Close/Last	Volume	Open	High	Low
2020-02-28	$273.36	106721200	$257.26	$278.41	$256.37
2020-02-27	$273.52	80151380	$281.1	$286	$272.96
2020-02-26	$292.65	49678430	$286.53	$297.88	$286.5
2020-02-25	$288.08	57668360	$300.95	$302.53	$286.13
2020-02-24	$298.18	55548830	$297.26	$304.18	$289.23
...	...	...	...	...	...
2010-03-05	$31.2786	224647427	$30.7057	$31.3857	$30.6614
2010-03-04	$30.1014	89591907	$29.8971	$30.1314	$29.8043
2010-03-03	$29.9043	92846488	$29.8486	$29.9814	$29.7057
2010-03-02	$29.8357	141486282	$29.99	$30.1186	$29.6771
2010-03-01	$29.8557	137312041	$29.3928	$29.9286	$29.35

2518 rows × 5 columns

The data is ready.

Let's see the type of each parameter:

In [5]:
```
apple.dtypes
```

Out[5]:
```
 Close/Last    object
 Volume         int64
 Open          object
 High          object
 Low           object
dtype: object
```

As we can see, we have 4 strings parameters. In order to "work" with the data, I have to transform those types to floats.
I will use the "replace" function to transform those details.

In [6]:
```
columns = apple.columns
for column in columns:
    if column != ' Volume':
        apple[column] = apple[column].apply(lambda x: x.replace('$', '')
                                            if isinstance(x, str) else x).astype(float)
apple
```

Out[6]:

Date	Close/Last	Volume	Open	High	Low
2020-02-28	273.3600	106721200	257.2600	278.4100	256.3700
2020-02-27	273.5200	80151380	281.1000	286.0000	272.9600
2020-02-26	292.6500	49678430	286.5300	297.8800	286.5000
2020-02-25	288.0800	57668360	300.9500	302.5300	286.1300
2020-02-24	298.1800	55548830	297.2600	304.1800	289.2300
...	...	...	...	...	...
2010-03-05	31.2786	224647427	30.7057	31.3857	30.6614
2010-03-04	30.1014	89591907	29.8971	30.1314	29.8043
2010-03-03	29.9043	92846488	29.8486	29.9814	29.7057
2010-03-02	29.8357	141486282	29.9900	30.1186	29.6771
2010-03-01	29.8557	137312041	29.3928	29.9286	29.3500

2518 rows × 5 columns

In [7]:
```
apple.dtypes
```

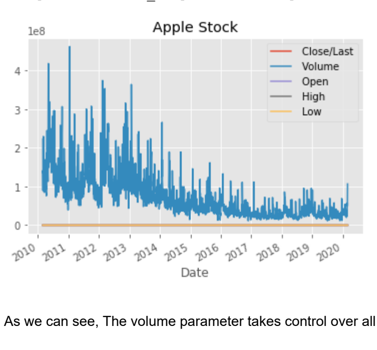
Out[7]:
```
 Close/Last    float64
 Volume          int64
 Open          float64
 High          float64
 Low           float64
dtype: object
```

Now the data is ready for "work".

Let's put the data set on the plot and let's see what we got:

In [8]:
```
apple.plot(title='Apple Stock')
```

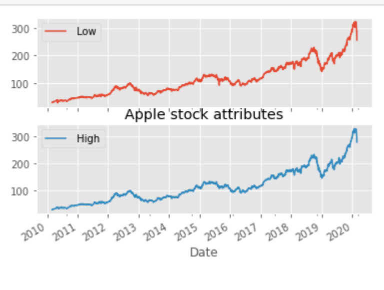
Out[8]: `<matplotlib.axes._subplots.AxesSubplot at 0x1c756879790>`



As we can see, The volume parameter takes control over all the rest due to the fact his numbers are much higher than the others.

here are two graphs, where we can see the maximum stock in each day and the minimum.

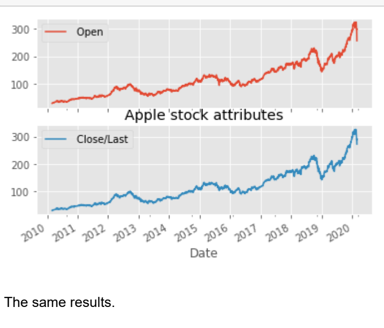
In [9]:
```
apple[[" Low", " High"]].plot(subplots=True)
plt.title('Apple stock attributes')
plt.show()
```



Both graphs show us the same tendency.

Now let's do the same thing but to the "Open" and the "Close/Last" Parameters.

In [10]:
```
apple[[" Open", " Close/Last"]].plot(subplots=True)
plt.title('Apple stock attributes')
plt.show()
```

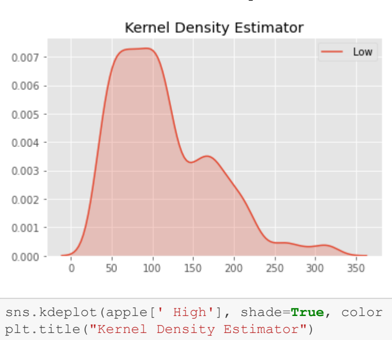


The same results.

In this part, I would like to see the density of a couple of parameters.

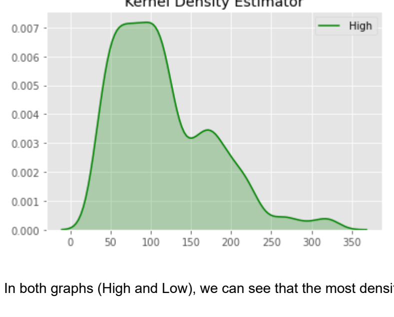
In [11]:
```
sns.kdeplot(apple[' Low'], shade=True)
plt.title("Kernel Density Estimator")
```

Out[11]: `Text(0.5, 1.0, 'Kernel Density Estimator')`



In [12]:
```
sns.kdeplot(apple[' High'], shade=True, color = 'green')
plt.title("Kernel Density Estimator")
```

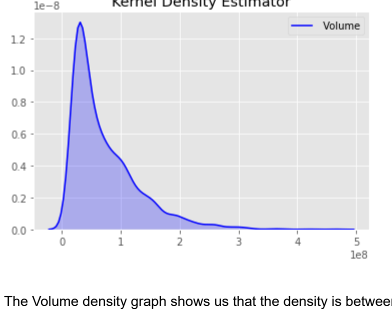
Out[12]: `Text(0.5, 1.0, 'Kernel Density Estimator')`



In both graphs (High and Low), we can see that the most density centralized between 50-110 stock price.

In [13]:
```
sns.kdeplot(apple[' Volume'], shade=True, color = 'blue')
plt.title("Kernel Density Estimator")
```

Out[13]: `Text(0.5, 1.0, 'Kernel Density Estimator')`



The Volume density graph shows us that the density is between 0 - 10,000,000 traded in each day.

Resampling

Now, I would like to use resampling function to aggregate data into a defined time period (year in our case). Institutions can then see an overview of stock prices and make decisions according to these trends.

Presesnting the mean of each parameter in every year:

In [14]:
```
apple.resample(rule = 'A').mean()
```

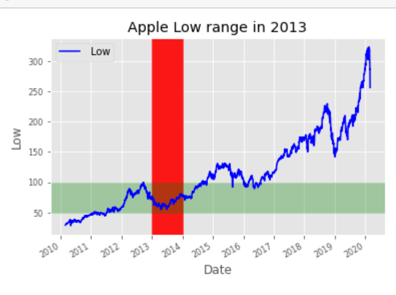
Out[14]:

Date	Close/Last	Volume	Open	High	Low
2010-12-31	38.555780	1.456230e+08	38.562765	38.915179	38.113978
2011-12-31	52.000600	1.225162e+08	52.008761	52.489069	51.471089
2012-12-31	82.292795	1.315986e+08	82.378916	83.117889	81.411874
2013-12-31	67.519240	1.011518e+08	67.589696	68.234103	66.892427
2014-12-31	92.264550	6.292055e+07	92.219755	93.012600	91.484314
2015-12-31	120.039861	5.164626e+07	120.169524	121.241458	118.862948
2016-12-31	104.604008	3.826767e+07	104.507698	105.427063	103.690212
2017-12-31	150.551056	2.700270e+07	150.450849	151.406005	149.487555
2018-12-31	189.053426	3.376838e+07	189.105534	190.994052	187.183420
2019-12-31	208.255933	2.806072e+07	207.869079	209.831616	206.273003
2020-12-31	311.609500	3.725590e+07	310.763725	314.766247	307.831602

A quick look at this result shows us that the stock price has grown almost every year except in 2013 and 2016. The company needs to understand why in those specific years the stock prices fell. In the Volume parameter, we see a decline as the years progress except in 2012.

Marking the low range in 2013 in the Low stock price parameter:

In [15]:
```
ax = apple[[" Low"]].plot(color='blue',fontsize=8)
ax.set_xlabel('Date')
ax.set_ylabel('Low')
ax.axvspan('2013-01-01','2013-12-31', color='red', alpha=0.9)
ax.axhspan(50, 100, color='green',alpha=0.3)
plt.title("Apple Low range in 2013")
plt.show()
```



Now let's see the standard deviation in each year:

In [16]:
```
apple.resample(rule = 'A').std()
```

Out[16]:

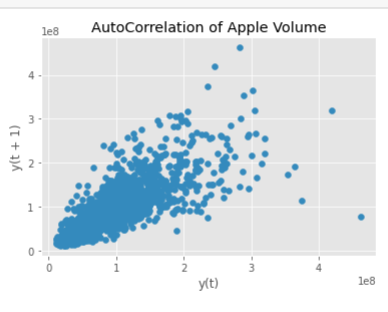
Date	Close/Last	Volume	Open	High	Low
2010-12-31	4.475987	6.050353e+07	4.483032	4.457806	4.550502
2011-12-31	3.703405	5.390431e+07	3.709986	3.764427	3.651114
2012-12-31	9.568140	5.775241e+07	9.647191	9.670320	9.514481
2013-12-31	6.412522	4.383644e+07	6.428731	6.364592	6.439075
2014-12-31	13.371212	2.935145e+07	13.382454	13.557438	13.206757
2015-12-31	7.683447	2.105904e+07	7.689131	7.462732	7.917445
2016-12-31	7.640743	1.703818e+07	7.586135	7.554403	7.680734
2017-12-31	14.621212	1.122725e+07	14.749725	14.822639	14.479358
2018-12-31	20.593860	1.451238e+07	20.484412	20.585083	20.439975
2019-12-31	34.538970	1.080146e+07	34.373587	34.485513	34.305493
2020-12-31	13.199674	1.627718e+07	14.136637	11.608984	14.485227

As we can see, In the stock price parameters the standard deviation range is higher the most in 2017-2019.

AutoCorrelation

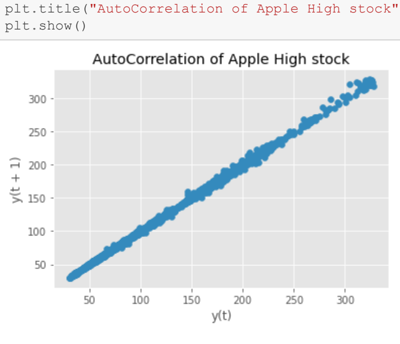
In this part, I will use autocorrelation plot. An autocorrelation plot is very useful for a time series analysis. This is because autocorrelation is a way of measuring and explaining the internal association between observations in a time series. We can check how strong an internal correlation is in an given amount of time.

In [17]:
```
lag_plot(apple[" Volume"])
plt.title("AutoCorrelation of Apple Volume")
plt.show()
```



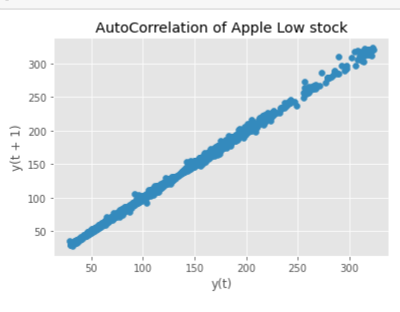
At the beginning of this plot, we see that the scatter is very density but as we progress the plot becomes more scattered. A reason for that is maybe because that when you invest more money in the stock you can lose more too.

In [18]:
```
lag_plot(apple[" High"])
plt.title("AutoCorrelation of Apple High stock")
plt.show()
```



We can see full correlation between the high stock of day and the one after.

In [19]:
```
lag_plot(apple[" Low"])
plt.title("AutoCorrelation of Apple Low stock")
plt.show()
```



Same.

AutoRegressive (AR) Model

An autoregressive (AR) model is a representation of a type of random process. The autoregressive model specifies that the output variable depends linearly on its own previous values. The notation AR(p) indicates an autoregressive model of order p. The AR(p) model is defined as:

$$X_n = c + \sum_{i=1}^{p} a_i X_{n-1} + e_n$$

Where:

1. C is a constant.
2. e_n called “white noise”.
3. a_i are the coeffients (parameters) of the model.

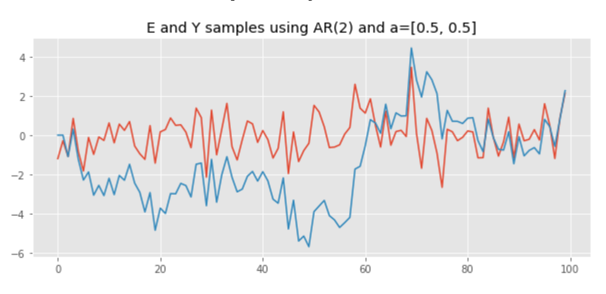
I will use in the AR(2) process with using 0.5 and 0.5 a parameters, which means that the previous two terms and the noise term contribute to the output.

In [20]:
```
SAMPLES = 100
e = np.random.randn(SAMPLES)
```

In [21]:
```
def ar_2(size, p, constant, noise):
    x = np.zeros(size)
    for i in range(p, SAMPLES):
        x[i] = constant[0]*x[i-2] + constant[1]*x[i-1] + e[i]
    return x
```

In [22]:
```
a = [0.5, 0.5]
y = ar_2(SAMPLES, len(a), a, e)
plt.figure(figsize=(10, 4))
plt.plot(range(SAMPLES), e, label="e")
plt.plot(range(SAMPLES), y, label="y")
plt.title("E and Y samples using AR(2) and a=[0.5, 0.5]")
```

Out[22]: `Text(0.5, 1.0, 'E and Y samples using AR(2) and a=[0.5, 0.5]')`



As we can see, At first the model did identify the trends but with a small downward error. however, as the graph progressing we see the prediction become more and more accurate to the real details.

Principal Components Ansalysis (PCA)

In this section, I will use PCA method. Principal Component Analysis (PCA) is one of the most popular dimensionality reduction methods which transforms the data by projecting it to a set of orthogonal axes. It works by finding the eigenvectors and eigenvalues of the covariance matrix of the dataset. The Eigenvectors are called as the “Principal Components” of the dataset.

Normalizing the data:

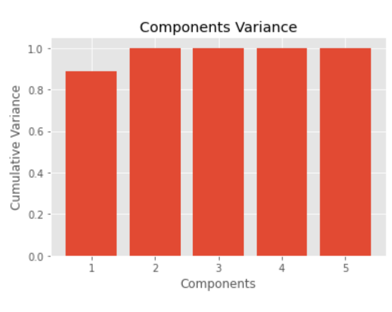
In [23]:
```
sc = StandardScaler() # (Xi - mean) / std
normalized_data = sc.fit_transform(apple)
```

Using the PCA method and displaying a graph that describes to me by how many components I can present the data optimally:

In [24]:
```
pca = PCA()
pca_data = pca.fit_transform(normalized_data)
```

In [25]:
```
plt.bar(range(1, len(pca.explained_variance_ratio_)+1), np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('Components')
plt.ylabel('Cumulative Variance')
plt.xticks(range(1, len(pca.explained_variance_ratio_)+1))
plt.title("Components Variance")
plt.plot()
```

Out[25]: `[]`



In [26]:
```
pd.DataFrame({"Variance": pca.explained_variance_ratio_}, index=range(1, len(pca.explained_variance_rat
io_) + 1))
```

Out[26]:

	Variance
1	0.886148
2	0.113732
3	0.000070
4	0.000041
5	0.000009

As we can see, I can use only two componets to present the 99% of the data.

In [27]:
```
pca = PCA(n_components=2)
pca_data = pca.fit_transform(normalized_data)
components = pd.DataFrame(pca.components_, columns = apple.columns)
components
```

Out[27]:

	Close/Last	Volume	Open	High	Low
0	0.471263	-0.334207	0.471201	0.471016	0.471519
1	0.166691	0.942469	0.168010	0.172408	0.161289

In [28]:
```
pd.DataFrame(pca_data).plot(title="2D Apple Stock")
```

Out[28]: `<matplotlib.axes._subplots.AxesSubplot at 0x1c756dd0d00>`

As the graph shows, the red graph declining as the graph continue and the blue graph looks stable except at the end of it (noises).