

GRIP - The Sparks Foundation

Data Science & Business Analytics Intern

Task 7: Stock Market Prediction using Numerical and Textual Analysis

Stock to analyze and predict SENSEX (S&P BSE SENSEX) and to create a hybrid model for stock price/performance prediction using numerical analysis of historical stock prices and sentimental analysis of news headlines.

Tools: Numpy Array, Pandas, Matplotlib, Scikit Learn

Author : vandana

In [1]:

```
pip install yfinance --upgrade --no-cache-dir
```

Collecting yfinance

Downloading yfinance-0.1.70-py2.py3-none-any.whl (26 kB)

Requirement already satisfied: lxml>=4.5.1 in c:\users\vanda\anaconda3\lib\site-packages (from yfinance) (4.6.3)

Requirement already satisfied: requests>=2.26 in c:\users\vanda\anaconda3\lib\site-packages (from yfinance) (2.26.0)

Requirement already satisfied: numpy>=1.15 in c:\users\vanda\anaconda3\lib\site-packages (from yfinance) (1.20.3)

Collecting multitasking>=0.0.7

Downloading multitasking-0.0.10.tar.gz (8.2 kB)

Requirement already satisfied: pandas>=0.24.0 in c:\users\vanda\anaconda3\lib\site-packages (from yfinance) (1.3.4)

Requirement already satisfied: pytz>=2017.3 in c:\users\vanda\anaconda3\lib\site-packages (from pandas>=0.24.0->yfinance) (2021.3)

Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\vanda\anaconda3\lib\site-packages (from pandas>=0.24.0->yfinance) (2.8.2)

Requirement already satisfied: six>=1.5 in c:\users\vanda\anaconda3\lib\site-packages (from python-dateutil>=2.7.3->pandas>=0.24.0->yfinance) (1.16.0)

Requirement already satisfied: idna<4,>=2.5 in c:\users\vanda\anaconda3\lib\site-packages (from requests>=2.26->yfinance) (3.2)

Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\vanda\anaconda3\lib\site-packages (from requests>=2.26->yfinance) (2.0.4)

Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\vanda\anaconda3\lib\site-packages (from requests>=2.26->yfinance) (1.26.7)

Requirement already satisfied: certifi>=2017.4.17 in c:\users\vanda\anaconda3\lib\site-packages (from requests>=2.26->yfinance) (2021.10.8)

Building wheels for collected packages: multitasking

Building wheel for multitasking (setup.py): started

Building wheel for multitasking (setup.py): finished with status 'done'

Created wheel for multitasking: filename=multitasking-0.0.10-py3-none-any.whl size=8500 sha256=3d91c23a1969ebd1e90581410c1c3296ed0f5e4bdb12d91a89cf6b045c1383dd

Stored in directory: C:\Users\vanda\AppData\Local\Temp\pip-ephem-wheel-cache-rt77btfd\wheels\f2\b5\2c\59ba95dcf854e542944c75fe3da584e4e3833b319735a0546c

Successfully built multitasking

Installing collected packages: multitasking, yfinance

Successfully installed multitasking-0.0.10 yfinance-0.1.70

Note: you may need to restart the kernel to use updated packages.

In [2]:

```
pip install altair vega_datasets
```

Collecting altair

Using cached altair-4.2.0-py3-none-any.whl (812 kB)

Collecting vega_datasets

Using cached vega_datasets-0.9.0-py3-none-any.whl (210 kB)

Requirement already satisfied: numpy in c:\users\vanda\anaconda3\lib\site-packages (from altair) (1.20.3)

Requirement already satisfied: entrypoints in c:\users\vanda\anaconda3\lib\site-packages (from altair) (0.3)

Requirement already satisfied: jsonschema>=3.0 in c:\users\vanda\anaconda3\lib\site-packages (from altair) (3.2.0)

Requirement already satisfied: toolz in c:\users\vanda\anaconda3\lib\site-packages (from altair) (0.11.1)

Requirement already satisfied: pandas>=0.18 in c:\users\vanda\anaconda3\lib\site-packages (from altair) (1.3.4)

Requirement already satisfied: jinja2 in c:\users\vanda\anaconda3\lib\site-packages (from altair) (2.11.3)

Requirement already satisfied: attrs>=17.4.0 in c:\users\vanda\anaconda3\lib\site-packages (from jsonschema>=3.0->altair) (21.2.0)

Requirement already satisfied: six>=1.11.0 in c:\users\vanda\anaconda3\lib\site-packages (from jsonschema>=3.0->altair) (1.16.0)

Requirement already satisfied: setuptools in c:\users\vanda\anaconda3\lib\site-packages (from jsonschema>=3.0->altair) (58.0.4)

Requirement already satisfied: pyparsing>=2.4.0 in c:\users\vanda\anaconda3\lib\site-packages (from jsonschema>=3.0->altair) (2.6.2)

Requirement already satisfied: pytz>=2017.3 in c:\users\vanda\anaconda3\lib\site-packages (from pandas>=0.18->altair) (2021.3)

Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\vanda\anaconda3\lib\site-packages (from pandas>=0.18->altair) (2.8.2)

Requirement already satisfied: MarkupSafe>=0.23 in c:\users\vanda\anaconda3\lib\site-packages (from jinja2->altair) (1.1.1)

Installing collected packages: vega-datasets, altair

Successfully installed altair-4.2.0 vega-datasets-0.9.0

Note: you may need to restart the kernel to use updated packages.

In [3]:

```
#import libraries
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

#Altair is a declarative statistical visualization library for Python
import altair as alt

import statsmodels.api as sm

from sklearn.model_selection import ShuffleSplit
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.multioutput import RegressorChain
from sklearn.ensemble import RandomForestRegressor
```

In [4]:


```
#ignoring the warnings
import warnings
warnings.filterwarnings('ignore')
```

In [5]:

```
# import yfinance as yf
# bse_data = yf.download('^BSESN', start='2015-01-01', end='2020-11-03')
# unseenbse_data = yf.download('^BSESN', start='2020-11-03', end='2020-11-04')
```

In [6]:

```
import yfinance as yf
bse_data = yf.download('^BSESN', start='2015-01-01', end='2020-06-30')
#since our Textual Analysis dataset containing news from Times of India News Headlines is o
#So we will assume today is 29th June 2020 and tomorrow is 30th June 2020. And we have to p
#for tomorrow 30th June 2020.
unseenbse_data = yf.download('^BSESN', start='2020-06-30', end='2020-07-01')
```



```
[*****100%*****] 1 of 1 completed
[*****100%*****] 1 of 1 completed
```

In [7]:

```
bse_data.columns
```

Out[7]:

```
Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

In [8]:

```
bse_data.columns
```

Out[8]:

```
Index(['Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

In [9]:

```
bse_data.rename(columns={'Open': 'open', 'High': 'high', 'Low': 'low', 'Close': 'close', 'Adj Close': 'adjclose', 'Volume': 'volume'})
```

In [10]:

```
unseenbse_data.rename(columns={'Open': 'open', 'High': 'high', 'Low': 'low', 'Close': 'close', 'Adj Close': 'adjclose', 'Volume': 'volume'})
```

In [11]:

```
bse_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1345 entries, 2015-01-02 to 2020-06-29
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   open        1345 non-null   float64
1   high        1345 non-null   float64
2   low         1345 non-null   float64
3   close       1345 non-null   float64
4   adjclose    1345 non-null   float64
5   volume      1345 non-null   int64
dtypes: float64(5), int64(1)
memory usage: 73.6 KB
```

In [12]:

```
unseenbse_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1 entries, 2020-06-30 to 2020-06-30
Data columns (total 6 columns):
#   Column      Non-Null Count  Dtype
---  ---
0   open        1 non-null      float64
1   high        1 non-null      float64
2   low         1 non-null      float64
3   close       1 non-null      float64
4   adjclose    1 non-null      float64
5   volume      1 non-null      int64
dtypes: float64(5), int64(1)
memory usage: 56.0 bytes
```

In [13]:

```
bse_data.head()
```

Out[13]:

	open	high	low	close	adjclose	volume
Date						
2015-01-02	27521.279297	27937.470703	27519.259766	27887.900391	27887.900391	7400
2015-01-05	27978.429688	28064.490234	27786.849609	27842.320312	27842.320312	9200
2015-01-06	27694.230469	27698.929688	26937.060547	26987.460938	26987.460938	14100
2015-01-07	26983.429688	27051.599609	26776.119141	26908.820312	26908.820312	12200
2015-01-08	27178.769531	27316.410156	27101.939453	27274.710938	27274.710938	8200

In [14]:

```
bse_data.head()
```

Out[14]:

	open	high	low	close	adjclose	volume
Date						
2015-01-02	27521.279297	27937.470703	27519.259766	27887.900391	27887.900391	7400
2015-01-05	27978.429688	28064.490234	27786.849609	27842.320312	27842.320312	9200
2015-01-06	27694.230469	27698.929688	26937.060547	26987.460938	26987.460938	14100
2015-01-07	26983.429688	27051.599609	26776.119141	26908.820312	26908.820312	12200
2015-01-08	27178.769531	27316.410156	27101.939453	27274.710938	27274.710938	8200

In [15]:

```
unseenbse_data.head()
```

Out[15]:

	open	high	low	close	adjclose	volume
Date						
2020-06-30	35168.300781	35233.910156	34812.800781	34915.800781	34915.800781	18500

In [16]:

```
bse_data.reset_index(inplace=True)
```

In [17]:

```
bse_data.rename(columns={'Date': 'date'}, inplace = True)
```

In [18]:

```
bse_data.head()
```

Out[18]:

	date	open	high	low	close	adjclose	volume
0	2015-01-02	27521.279297	27937.470703	27519.259766	27887.900391	27887.900391	7400
1	2015-01-05	27978.429688	28064.490234	27786.849609	27842.320312	27842.320312	9200
2	2015-01-06	27694.230469	27698.929688	26937.060547	26987.460938	26987.460938	14100
3	2015-01-07	26983.429688	27051.599609	26776.119141	26908.820312	26908.820312	12200
4	2015-01-08	27178.769531	27316.410156	27101.939453	27274.710938	27274.710938	8200

In [19]:

```
unseenbse_data.reset_index(inplace=True)
```

In [20]:

```
unseenbse_data.rename(columns={'Date': 'date'}, inplace = True)
```

In [21]:

```
unseenbse_data.head()
```

Out[21]:

	date	open	high	low	close	adjclose	volume
0	2020-06-30	35168.300781	35233.910156	34812.800781	34915.800781	34915.800781	18500

In [22]:

```
bse_data['date'] = pd.to_datetime(bse_data['date'], format = '%Y%m%d')
```

In [23]:

```
unseenbse_data['date'] = pd.to_datetime(unseenbse_data['date'], format = '%Y%m%d')
```

In [24]:

```
#before moving forward let us calculate first the actual price
unseenbse_data_price = round((unseenbse_data['high'] + unseenbse_data['low'] + unseenbse_data['close'])/3)
unseenbse_data_price
#actual price
```

Out[24]:

```
0    34987.5
dtype: float64
```

Rolling window analysis of time series

Creating 4, 16, 52 week moving average of closing price of BSE index

In [25]:

```
def stock_weekmovingavg(wks, df):
    dateclose_data = pd.DataFrame({'date': df['date'], 'close': df['close']})
    dateclose_data.set_index('date', inplace=True)
    num = wks * 5 #calculating the number of days in the week
    dateclose_data['movingavg'] = dateclose_data['close'].rolling(window=num).mean().shift()
    return dateclose_data.dropna()
```

In [26]:

```
stock_weekmovingavg(4, bse_data).head()
```

Out[26]:

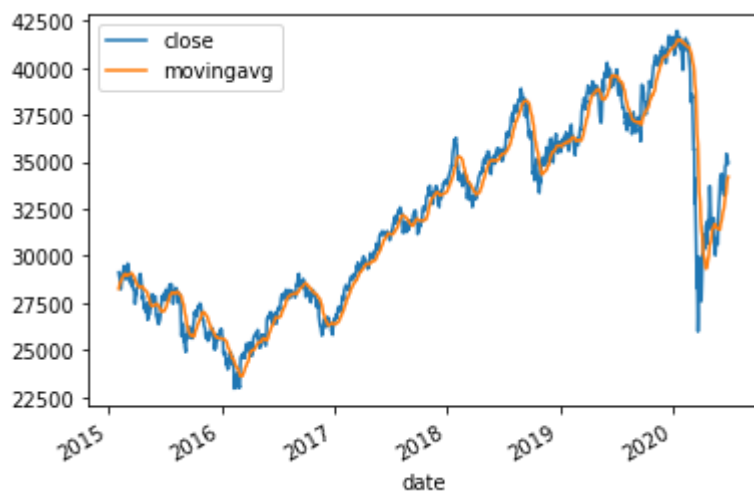
	close	movingavg
date		
2015-02-02	29122.269531	28256.509570
2015-02-03	29000.140625	28318.228027
2015-02-04	28883.109375	28376.119043
2015-02-05	28850.970703	28470.901465
2015-02-06	28717.910156	28568.008984

In [27]:

```
stock_weekmovingavg(4, bse_data).plot()
```

Out[27]:

<AxesSubplot:xlabel='date'>



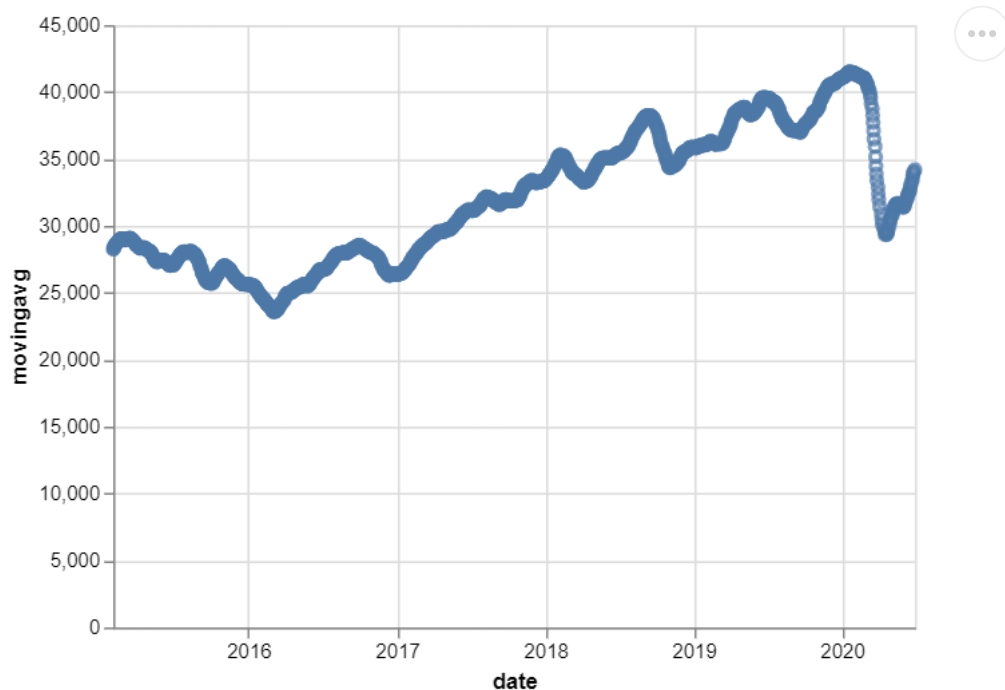
In [28]:

```
altdata_fourweek = stock_weekmovingavg(4, bse_data)
altdata_fourweek.reset_index(inplace=True)
altdata_fourweek.rename(columns={list(altdata_fourweek)[0]: 'date'}, inplace=True)
```

In [29]:

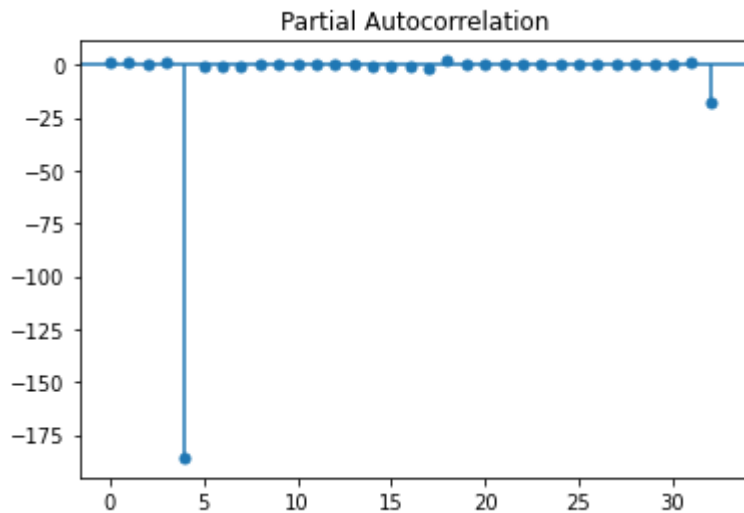
```
alt.Chart(altdata_fourweek).mark_point().encode(  
    x='date',  
    y='movingavg')
```

Out[29]:



In [30]:

```
plotfourweek = altdata_fourweek.filter(['date', 'movingavg'], axis=1) #df.copy()
plotfourweek.index = pd.Index(sm.tsa.datetools.dates_from_range('2015', length=len(altdata_
del plotfourweek['date']
sm.graphics.tsa.plot_pacf(plotfourweek.values.squeeze())
plt.show()
```



In the partial autocorrelation plot above, we have statistically significant partial autocorrelations at lag values 4 and 32. Since it is less than 0 and more than -1 so 4 and 32 represents a perfect negative correlation. While the rest of values are very close to 0 and under the confidence intervals, which are represented as blue shaded regions (which is not vividly seen in the above plot)

In [31]:

```
stock_weekmovingavg(16, bse_data).head()
```

Out[31]:

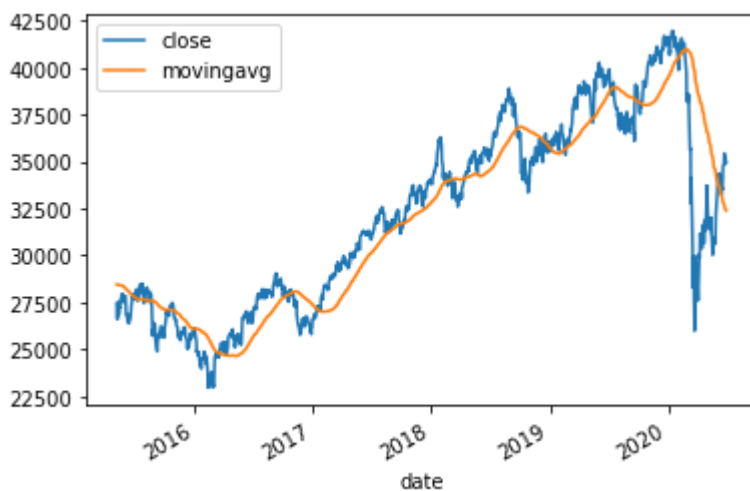
	close	movingavg
date		
2015-05-05	27440.140625	28447.880933
2015-05-06	26717.369141	28442.283936
2015-05-07	26599.109375	28428.222046
2015-05-08	27105.390625	28423.367651
2015-05-11	27507.300781	28425.824780

In [32]:

```
stock_weekmovingavg(16, bse_data).plot()
```

Out[32]:

<AxesSubplot:xlabel='date'>



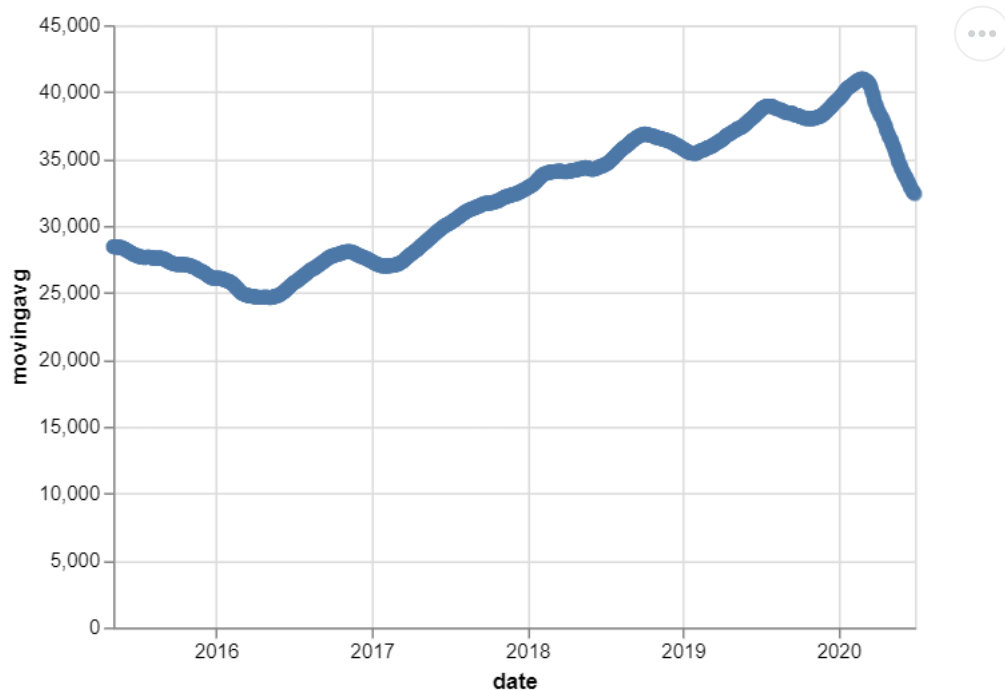
In [33]:

```
altdata_sixteenweek = stock_weekmovingavg(16, bse_data)
altdata_sixteenweek.reset_index(inplace=True)
altdata_sixteenweek.rename(columns={list(altdata_sixteenweek)[0]: 'date'}, inplace=True)
```

In [34]:

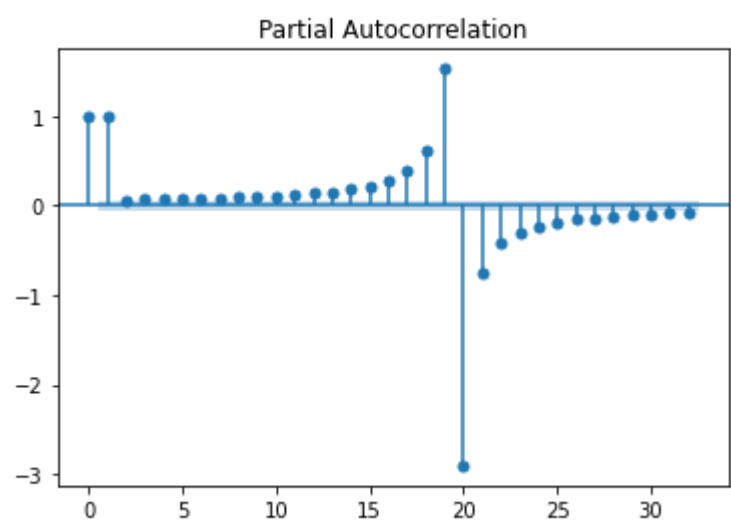
```
alt.Chart(altdata_sixteenweek).mark_point().encode(  
    x='date',  
    y='movingavg')
```

Out[34]:



In [35]:

```
plotsixteenweek = altdata_sixteenweek.filter(['date', 'movingavg'], axis=1) #df.copy()
plotsixteenweek.index = pd.Index(sm.tsa.datetools.dates_from_range('2015', length=len(altda
del plotsixteenweek['date']
sm.graphics.tsa.plot_pacf(plotsixteenweek.values.squeeze())
plt.show()
```



In the partial autocorrelation plot above, we have statistically significant partial autocorrelations at lag values 0, 1, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28. Where 0, 1, 19 represents a perfect positive correlation and 20 represents a perfect negative correlation. While the rest of values are very close to 0 and under the confidence intervals, which are represented as blue shaded regions (which is not vividly seen in the above plot)

In [36]:

```
stock_weekmovingavg(52, bse_data).head()
```

Out[36]:

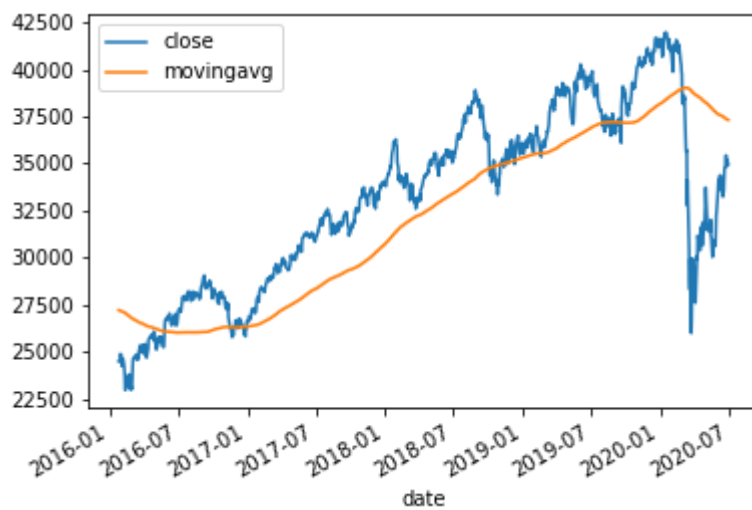
	close	movingavg
date		
2016-01-25	24485.949219	27198.907963
2016-01-27	24492.390625	27185.823535
2016-01-28	24469.570312	27172.939190
2016-01-29	24870.689453	27163.254995
2016-02-01	24824.830078	27155.416031

In [37]:

```
stock_weekmovingavg(52, bse_data).plot()
```

Out[37]:

<AxesSubplot:xlabel='date'>



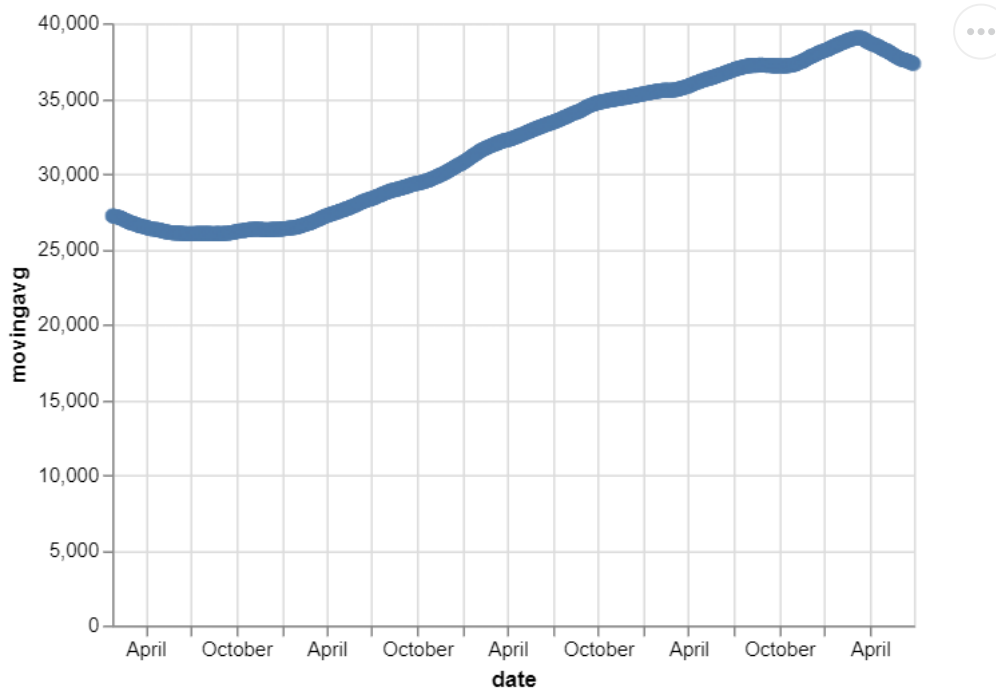
In [38]:

```
altdata_fiftytwoweek = stock_weekmovingavg(52, bse_data)
altdata_fiftytwoweek.reset_index(inplace=True)
altdata_fiftytwoweek.rename(columns={list(altdata_fiftytwoweek)[0]:'date'}, inplace=True)
```

In [39]:

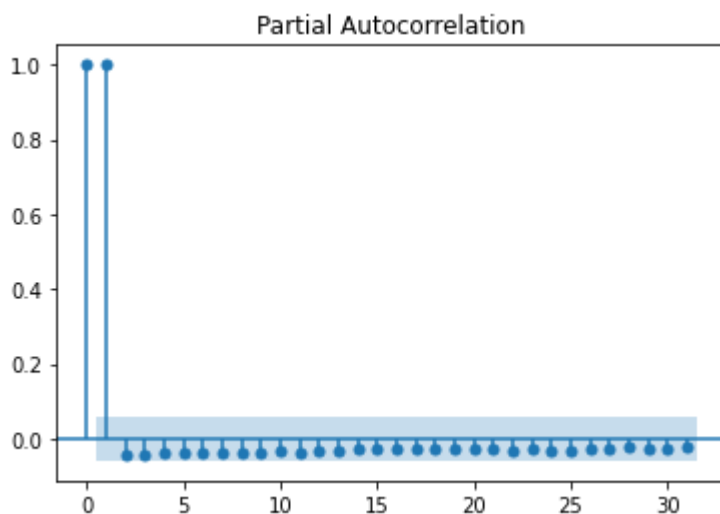
```
alt.Chart(altdata_fiftytwoweek).mark_point().encode(
    x='date',
    y='movingavg')
```

Out[39]:



In [40]:

```
plotfiftytwoweek = altdata_fiftytwoweek.filter(['date', 'movingavg'], axis=1) #df.copy()
plotfiftytwoweek.index = pd.Index(sm.tsa.datetools.dates_from_range('2015', length=len(altd
del plotfiftytwoweek['date']
sm.graphics.tsa.plot_pacf(plotfiftytwoweek.values.squeeze())
plt.show())
```



In the partial autocorrelation plot above, we have statistically significant partial autocorrelations at lag values 0, 1 representing a perfect positive correlation. While the rest of values are very close to 0 and under the confidence intervals, which are represented as blue shaded regions

Creating a rolling window of size 10 and 50 of the BSE index

In [41]:

```
def rollingwindows(days, df):
    data = df.filter(['date', 'open', 'high', 'low', 'close'], axis=1) #df.copy()
    data.set_index('date', inplace=True)
    rollingwindows_data = data.rolling(window=days).mean().shift()
    return rollingwindows_data.dropna()
```

In [42]:

```
rollingwindows(10, bse_data).head()
```

Out[42]:

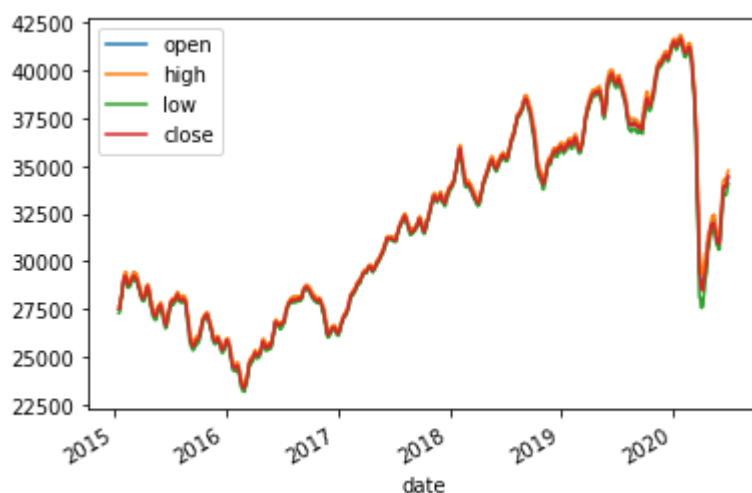
	open	high	low	close
date				
2015-01-16	27515.904883	27657.483008	27279.612891	27479.296484
2015-01-19	27569.437891	27681.345898	27322.217969	27502.695508
2015-01-20	27596.578906	27708.302930	27363.268945	27544.664453
2015-01-21	27659.640820	27821.338867	27502.047852	27724.385352
2015-01-22	27845.606836	28011.988867	27703.692969	27922.389258

In [43]:

```
rollingwindows(10, bse_data).plot()
```

Out[43]:

<AxesSubplot:xlabel='date'>



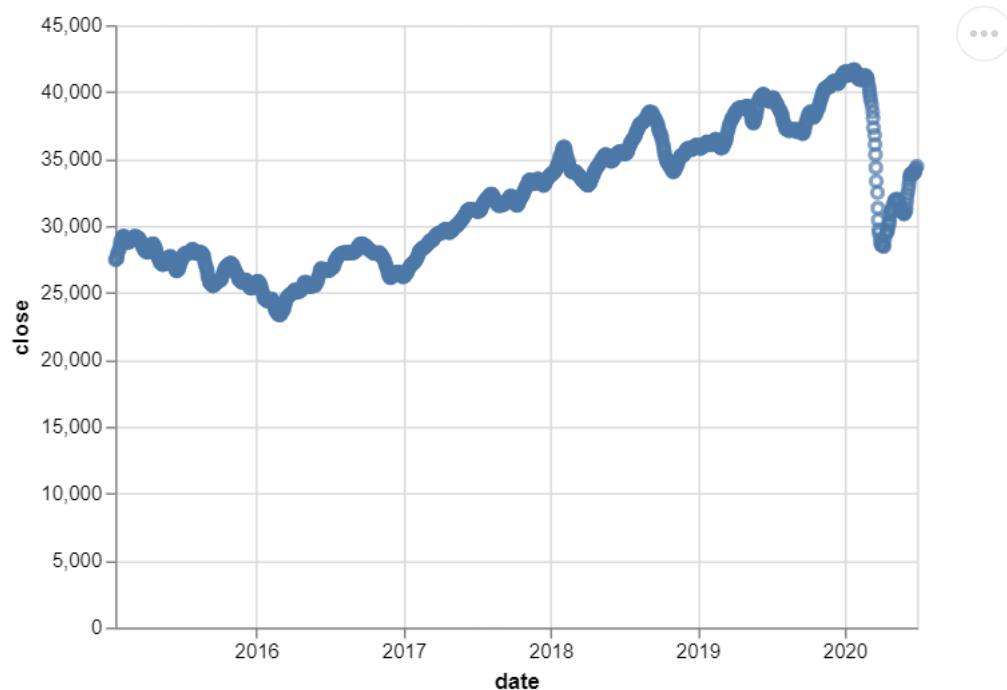
In [44]:

```
altdata_tendays = rollingwindows(10, bse_data)
altdata_tendays.reset_index(inplace=True)
altdata_tendays.rename(columns={list(altdata_tendays)[0]: 'date'}, inplace=True)
```

In [45]:

```
alt.Chart(altdata_tendays).mark_point().encode(  
    x = 'date',  
    y = 'close')
```

Out[45]:



In [46]:

```
rollingwindows(50, bse_data).head()
```

Out[46]:

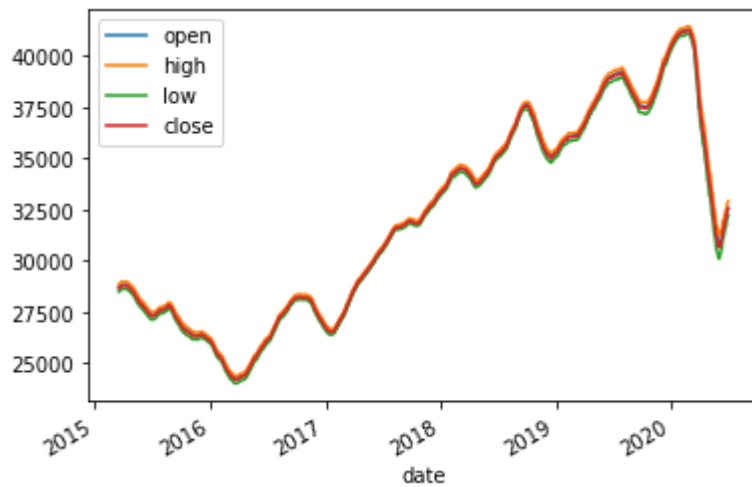
	open	high	low	close
date				
2015-03-18	28721.847383	28863.956836	28502.767422	28670.602461
2015-03-19	28746.759180	28881.346836	28523.317422	28685.286836
2015-03-20	28763.295000	28899.631836	28535.814414	28697.833828
2015-03-23	28778.719180	28915.340430	28561.266406	28723.306211
2015-03-24	28805.396367	28942.011250	28589.022031	28748.970195

In [47]:

```
rollingwindows(50, bse_data).plot()
```

Out[47]:

<AxesSubplot:xlabel='date'>



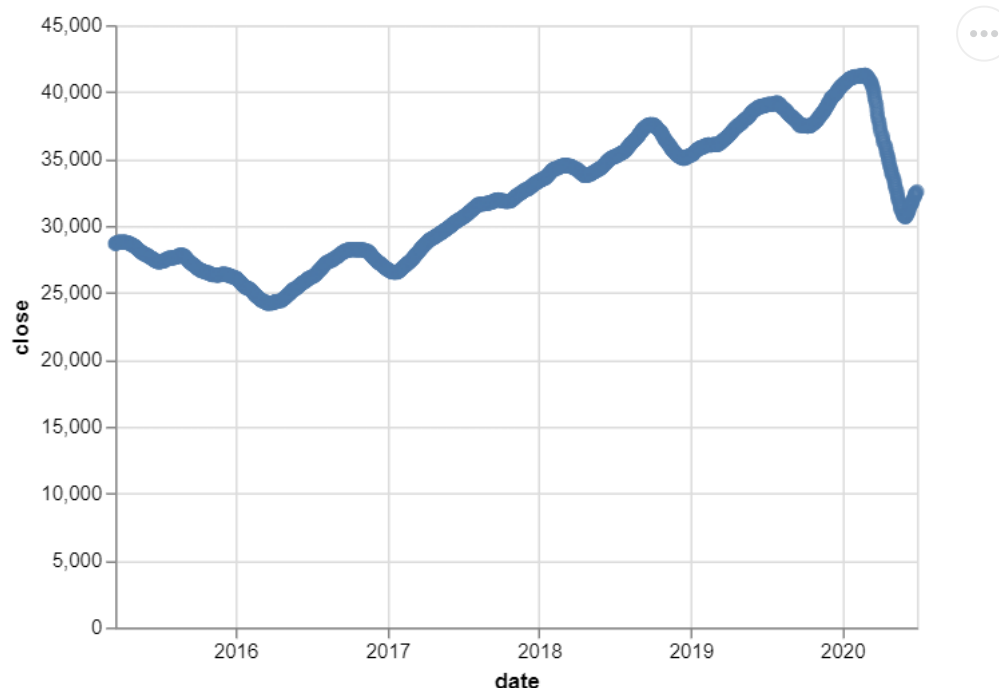
In [48]:

```
altdata_fiftydays = rollingwindows(50, bse_data)
altdata_fiftydays.reset_index(inplace=True)
altdata_fiftydays.rename(columns={list(altdata_fiftydays)[0]: 'date'}, inplace=True)
```

In [49]:

```
alt.Chart(altdata_fiftydays).mark_point().encode(  
    x='date',  
    y='close')
```

Out[49]:



Creating the dummy time series:

Volume shocks : we will be creating a 0/1 dummy-coded boolean time series for shock, based on whether volume traded is 10% higher/lower than previous day. (0/1 dummy-coding is for direction of shock)

In [50]:

```
def boolean_shock(percent, df, col):
    data = df.filter(['date', col], axis=1) #df.copy()
    data.set_index('date', inplace=True)
    data['percentchg'] = (data[col].pct_change()) * 100 #percentage change compare to previo
    data['shock'] = data['percentchg'].apply(lambda x: 1 if x >= percent else 0)
    data.drop(col, axis = 1, inplace = True)
    return data.dropna()
```

In [51]:

```
boolean_shock(10, bse_data, 'volume')
```

Out[51]:

	percentchg	shock
date		
2015-01-05	24.324324	1
2015-01-06	53.260870	1
2015-01-07	-13.475177	0
2015-01-08	-32.786885	0
2015-01-09	35.365854	1
...
2020-06-23	0.398406	0
2020-06-24	5.555556	0
2020-06-25	-7.518797	0
2020-06-26	0.813008	0
2020-06-29	-26.209677	0

1344 rows × 2 columns

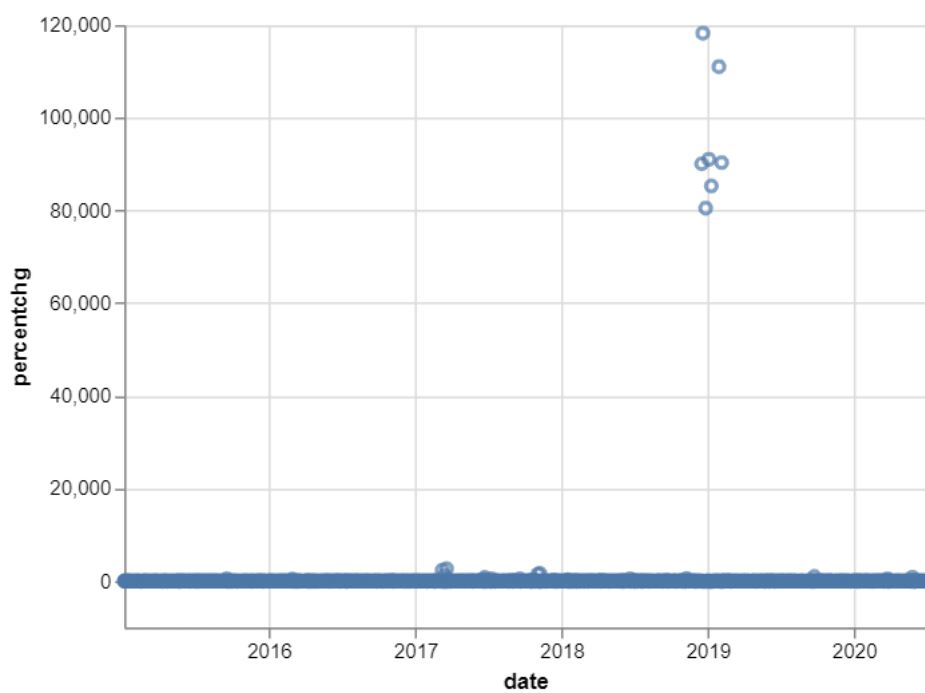
In [52]:

```
altdata_volpercentchg = boolean_shock(10, bse_data, 'volume')
altdata_volpercentchg.reset_index(inplace=True)
altdata_volpercentchg.rename(columns={list(altdata_volpercentchg)[0]: 'date'}, inplace=True)
```

In [53]:

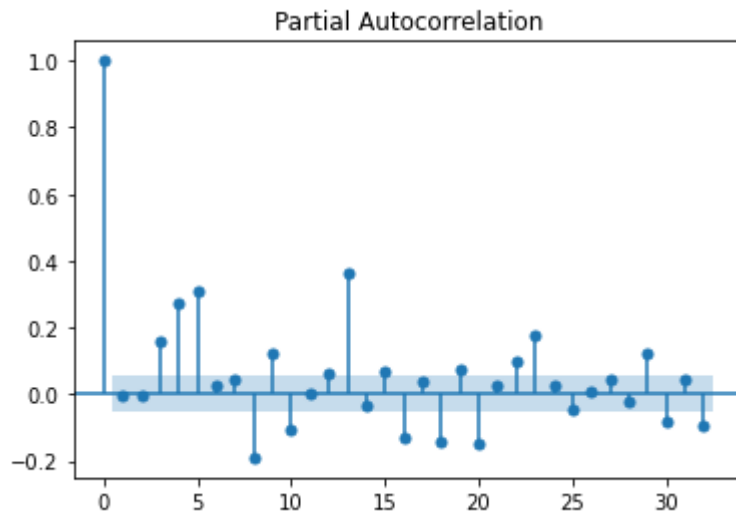
```
alt.Chart(altdata_volpercentchg).mark_point().encode(  
    x='date',  
    y='percentchg')
```

Out[53]:



In [54]:

```
plotvolpercentchg = altdata_volpercentchg.filter(['date', 'percentchg'], axis=1) #df.copy()
plotvolpercentchg.index = pd.Index(sm.tsa.datetools.dates_from_range('2015', length=len(altdata_volpercentchg)))
del plotvolpercentchg['date']
sm.graphics.tsa.plot_pacf(plotvolpercentchg.values.squeeze())
plt.show()
```



In the partial autocorrelation plot above, we have statistically significant partial autocorrelations at lag values 0, 3, 4, 5, 8, 9, 10, 12, 13, 15, 16, 18, 19, 20, 22, 23, 29, 30, 32. And lag value 0 represents a perfect positive correlation. While the rest of values are very close to 0 and under the confidence intervals, which are represented as blue shaded regions

In [55]:

```
boolean_shock(2, bse_data, 'close')
```

Out[55]:

	percentchg	shock
date		
2015-01-05	-0.163440	0
2015-01-06	-3.070360	0
2015-01-07	-0.291397	0
2015-01-08	1.359742	0
2015-01-09	0.673407	0
...
2020-06-23	1.486937	0
2020-06-24	-1.584653	0
2020-06-25	-0.077085	0
2020-06-26	0.944742	0
2020-06-29	-0.596367	0

1344 rows × 2 columns

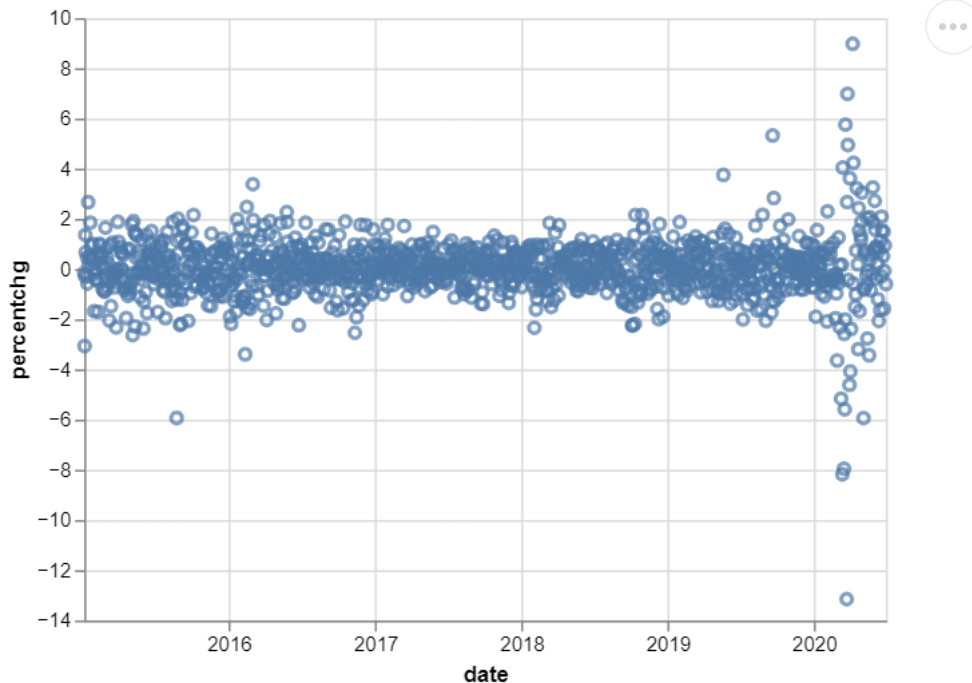
In [56]:

```
altdata_closepercentchg2 = boolean_shock(2, bse_data, 'close')
altdata_closepercentchg2.reset_index(inplace=True)
altdata_closepercentchg2.rename(columns={list(altdata_closepercentchg2)[0]: 'date'}, inplace=True)
```


In [57]:

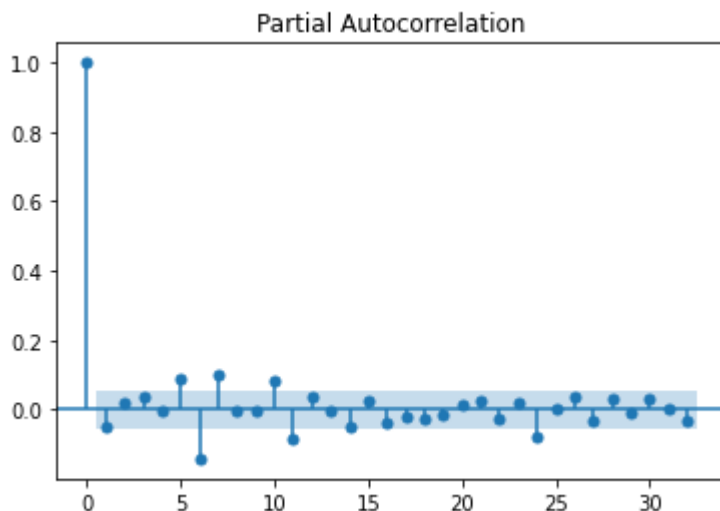
```
alt.Chart(alldata_closepercentchg2).mark_point().encode(
    x='date',
    y='percentchg'
)
```

Out[57]:



In [58]:

```
plotclosepercentchg2 = alldata_closepercentchg2.filter(['date', 'percentchg'], axis=1) #df.
plotclosepercentchg2.index = pd.Index(sm.tsa.dates_from_range('2015', length=len(
del plotclosepercentchg2['date']
sm.graphics.tsa.plot_pacf(plotclosepercentchg2.values.squeeze())
plt.show())
```



In the partial autocorrelation plot above, we have statistically significant partial autocorrelations at lag values 0, 5, 6, 7, 10, 11, 24. And lag value 0 represents a perfect positive correlation.

While the rest of values are very close to 0 and under the confidence intervals, which are

represented as blue shaded regions

In [59]:

```
boolean_shock(10, bse_data, 'close')
```

Out[59]:

	percentchg	shock
date		
2015-01-05	-0.163440	0
2015-01-06	-3.070360	0
2015-01-07	-0.291397	0
2015-01-08	1.359742	0
2015-01-09	0.673407	0
...
2020-06-23	1.486937	0
2020-06-24	-1.584653	0
2020-06-25	-0.077085	0
2020-06-26	0.944742	0
2020-06-29	-0.596367	0

1344 rows × 2 columns

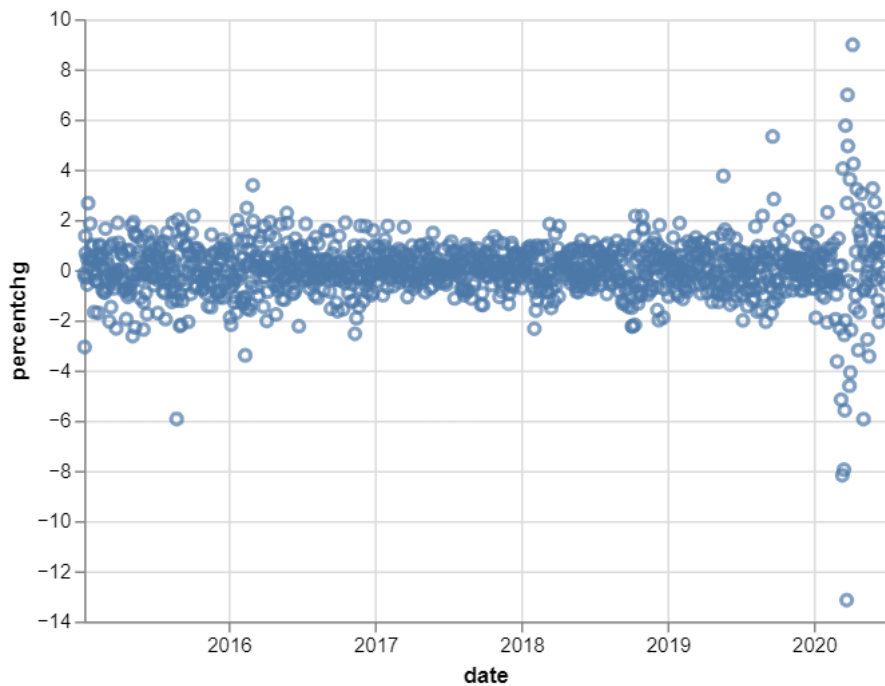
In [60]:

```

altdata_closepercentchg10 = boolean_shock(10, bse_data, 'close')
altdata_closepercentchg10.reset_index(inplace=True)
altdata_closepercentchg10.rename(columns={list(altdata_closepercentchg10)[0]: 'date'}, inplace=True)
alt.Chart(altdata_closepercentchg10).mark_point().encode(
    x='date',
    y='percentchg'
)

```

Out[60]:

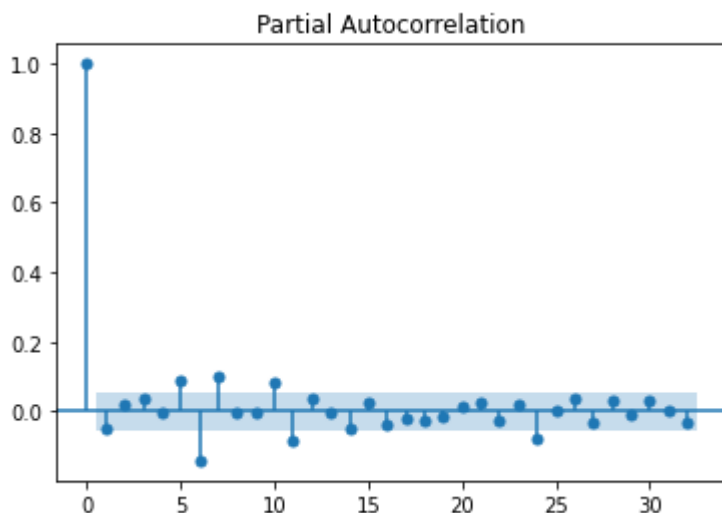


In [61]:

```

plotclosepercentchg10 = altdata_closepercentchg10.filter(['date', 'percentchg'], axis=1) #drop index
plotclosepercentchg10.index = pd.Index(sm.tsa.datetools.dates_from_range('2015', length=len(plotclosepercentchg10)))
del plotclosepercentchg10['date']
sm.graphics.tsa.plot_pacf(plotclosepercentchg10.values.squeeze())
plt.show()

```



In the partial autocorrelation plot above, we have statistically significant partial autocorrelations at lag values 0, 5, 6, 7, 10, 11, 24. And lag value 0 represents a perfect positive correlation. While the rest of values are very close to 0 and under the confidence intervals, which are represented as blue shaded regions

Pricing shock without volume shock

In [62]:

```
def priceboolean_shock(percent, df):
    df['date'] = pd.to_datetime(df['date'])
    data = df.filter(['date', 'high', 'low', 'close'], axis=1) #df.copy()
    data.set_index('date', inplace=True)
    data['priceavg'] = (data['high'] + data['low'] + data['close']) / 3
    data['shock'] = (data['priceavg'].pct_change()) * 100
    data['shock'] = data['shock'].apply(lambda x: 1 if x >= percent else 0)
    data.drop(['high', 'low', 'close'], axis = 1, inplace = True)
    return data
priceboolean_shock(10, bse_data)
```

Out[62]:

	priceavg	shock
date		
2015-01-02	27781.543620	0
2015-01-05	27897.886719	0
2015-01-06	27207.817057	0
2015-01-07	26912.179688	0
2015-01-08	27231.020182	0
...
2020-06-23	35252.093750	0
2020-06-24	35123.486979	0
2020-06-25	34807.830729	0
2020-06-26	35112.162760	0
2020-06-29	34885.312500	0

1345 rows × 2 columns

In [63]:

```
altdata_pricepercentchg = priceboolean_shock(10, bse_data)
altdata_pricepercentchg.reset_index(inplace=True)
altdata_pricepercentchg.rename(columns={list(altdata_pricepercentchg)[0]: 'date'}, inplace=True)
alt.Chart(altdata_pricepercentchg).mark_point().encode(
    x='date',
    y='priceavg'
)
```

Out[63]:

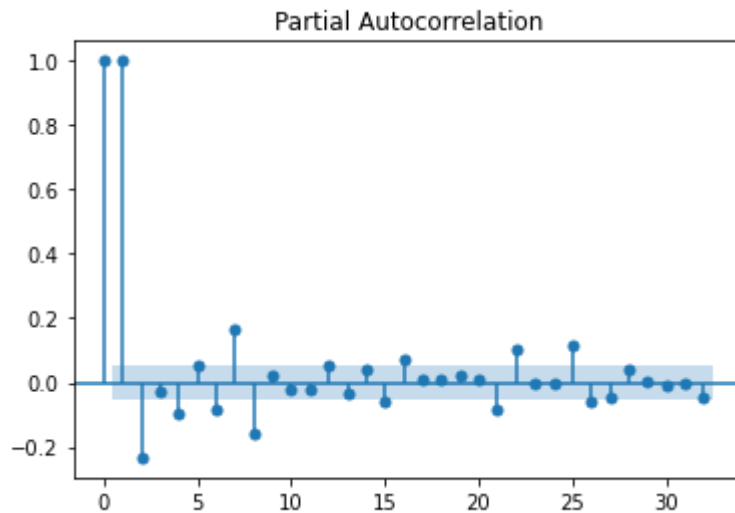


In [64]:

```

plotpricepercentchg = alldata_pricepercentchg.filter(['date', 'priceavg'], axis=1) #df.copy
plotpricepercentchg.index = pd.Index(sm.tsa.datetools.dates_from_range('2015', length=len(a
del plotpricepercentchg['date']
sm.graphics.tsa.plot_pacf(plotpricepercentchg.values.squeeze())
plt.show()

```



In the partial autocorrelation plot above, we have statistically significant partial autocorrelations at lag values 0, 1, 2, 4, 6, 7, 8, 15, 16, 21, 22, 25, 26. And lag values 0, 1 represents a perfect positive correlation. While the rest of values are very close to 0 and under the confidence intervals, which are represented as blue shaded regions

Creating the reverse dummy time series:

Price shocks : we will be creating a 0/1 dummy-coded boolean time series for shock, based on whether closing price at T vs T+1 has a difference > 2%. (0/1 dummy-coding is for direction of shock). This will be reverse of pct_change()

In [65]:

```
def reverseboolean_shock(percent, df, col):  
    data = df.filter(['date', col], axis=1) #df.copy()  
    data.set_index('date', inplace=True)  
    data = data.reindex(index=data.index[::-1])  
    data['percentchg'] = (data[col].pct_change()) * 100  
    data['shock'] = data['percentchg'].apply(lambda x: 1 if x > percent else 0)  
    data.drop(col, axis = 1, inplace = True)  
    data = data.reindex(index=data.index[::-1])  
    return data.dropna()  
reverseboolean_shock(2, bse_data, 'close')
```

Out[65]:

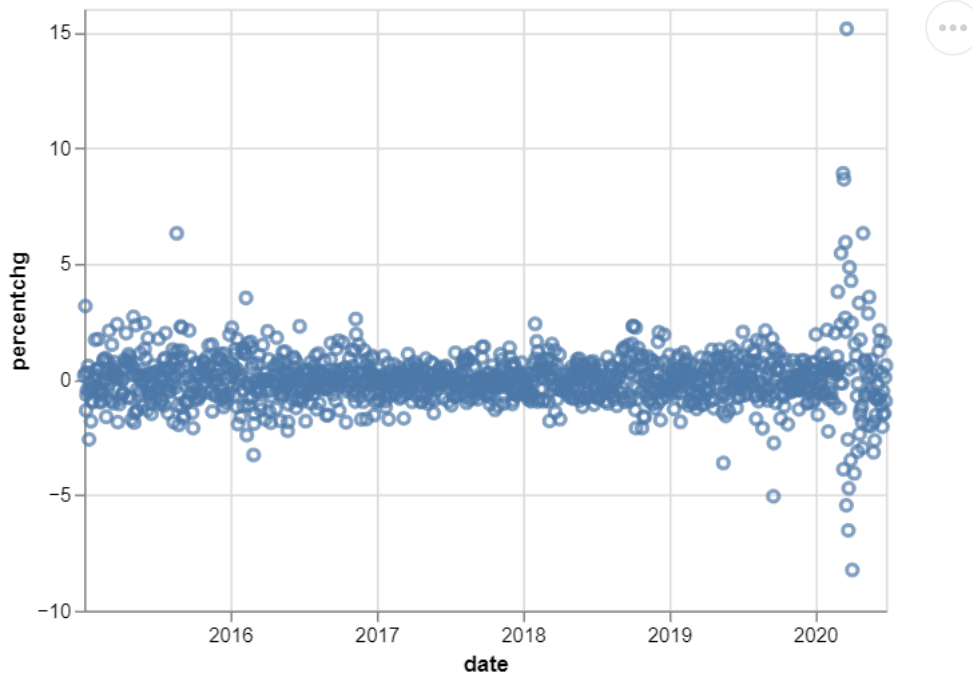
	percentchg	shock
date		
2015-01-02	0.163708	0
2015-01-05	3.167617	1
2015-01-06	0.292249	0
2015-01-07	-1.341501	0
2015-01-08	-0.668903	0
...
2020-06-22	-1.465151	0
2020-06-23	1.610168	0
2020-06-24	0.077145	0
2020-06-25	-0.935900	0
2020-06-26	0.599945	0

1344 rows × 2 columns

In [66]:

```
altdata_closepercentchg = reverseboolean_shock(2, bse_data, 'close')
altdata_closepercentchg.reset_index(inplace=True)
altdata_closepercentchg.rename(columns={list(altdata_closepercentchg)[0]: 'date'}, inplace=True)
alt.Chart(altdata_closepercentchg).mark_point().encode(
    x='date',
    y='percentchg'
)
```

Out[66]:



Pricing black swan : we will be creating a 0/1 dummy-coded boolean time series for shock, based on whether closing price at T vs T+1 has a difference > 5%. (0/1 dummy-coding is for direction of shock). This will be reverse of pct_change()

In [67]:

```
reverseboolean_shock(5, bse_data, 'close')
```

Out[67]:

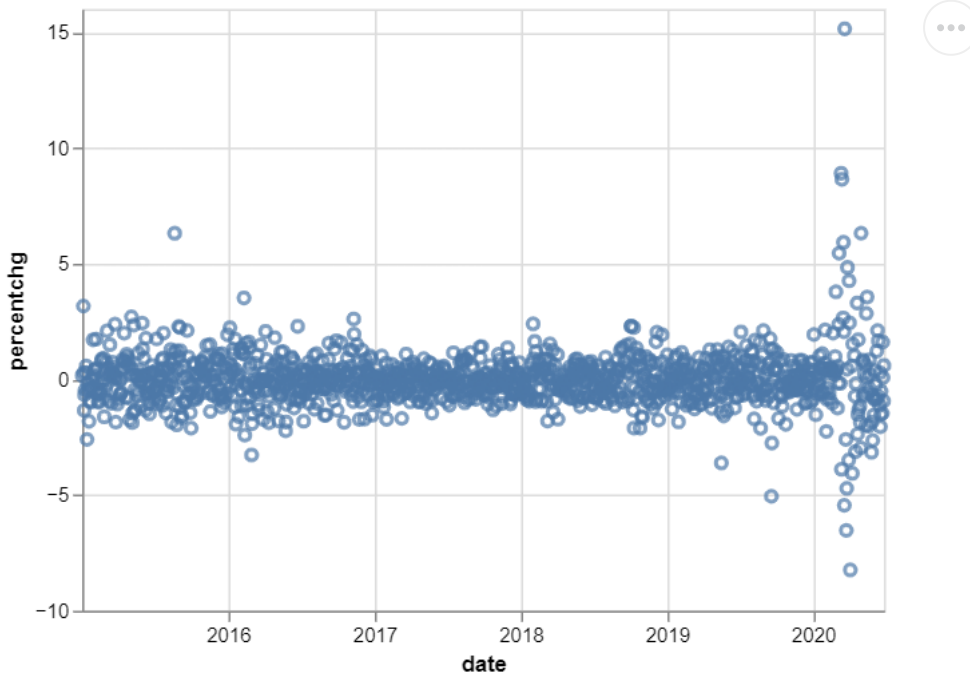
	percentchg	shock
date		
2015-01-02	0.163708	0
2015-01-05	3.167617	0
2015-01-06	0.292249	0
2015-01-07	-1.341501	0
2015-01-08	-0.668903	0
...
2020-06-22	-1.465151	0
2020-06-23	1.610168	0
2020-06-24	0.077145	0
2020-06-25	-0.935900	0
2020-06-26	0.599945	0

1344 rows × 2 columns

In [68]:

```
altdata_closepercentchg5 = reverseboolean_shock(5, bse_data, 'close')
altdata_closepercentchg5.reset_index(inplace=True)
altdata_closepercentchg5.rename(columns={list(altdata_closepercentchg5)[0]: 'date'}, inplace=True)
alt.Chart(altdata_closepercentchg5).mark_point().encode(
    x='date',
    y='percentchg'
)
```

Out[68]:



Pricing shock without volume shock : Now we will be creating a time series for pricing shock without volume shock based on whether price at T vs T+1 has a difference > 2%. (0/1 dummy-coding is for direction of shock). This will be reverse of pct_change()

In [69]:

```
def pricerreverseboolean_shock(percent, df):
    data = df.filter(['date', 'high', 'low', 'close'], axis=1) #df.copy()
    data.set_index('date', inplace=True)
    data = data.reindex(index=data.index[::-1])
    data['reversepriceavg'] = (data['high'] + data['low'] + data['close']) / 3
    data['shock'] = (data['reversepriceavg'].pct_change()) * 100
    data['shock'] = data['shock'].apply(lambda x: 1 if x >= percent else 0)
    data.drop(['high', 'low', 'close'], axis = 1, inplace = True)
    data = data.reindex(index=data.index[::-1])
    return data.dropna()
pricerreverseboolean_shock(2, bse_data)
```

Out[69]:

	reversepriceavg	shock
date		
2015-01-02	27781.543620	0
2015-01-05	27897.886719	1
2015-01-06	27207.817057	0
2015-01-07	26912.179688	0
2015-01-08	27231.020182	0
...
2020-06-23	35252.093750	0
2020-06-24	35123.486979	0
2020-06-25	34807.830729	0
2020-06-26	35112.162760	0
2020-06-29	34885.312500	0

1345 rows × 2 columns

In [70]:

```
altdata_reversepricepercentchg = pricerreverseboolean_shock(2, bse_data)
altdata_reversepricepercentchg.reset_index(inplace=True)
altdata_reversepricepercentchg.rename(columns={list(altdata_reversepricepercentchg)[0]: 'date',
                                              list(altdata_reversepricepercentchg)[1]: 'reversepriceavg'})
alt.Chart(altdata_reversepricepercentchg).mark_point().encode(
    x='date',
    y='reversepriceavg')
```

Out[70]:



Textual Analysis of news from Times of India News Headlines

In [71]:

```
news = pd.read_csv("india-news-headlines.csv")
```

In [72]:

```
#getting the overview of all the columns in the news dataset
news.columns
```

Out[72]:

```
Index(['publish_date', 'headline_category', 'headline_text'], dtype='object')
```

In [73]:

```
#finding the total rows and columns of news dataset
news.shape
```

Out[73]:

```
(3650970, 3)
```

In [74]:

```
#first 5 rows content of the dataset
news.head()
```

Out[74]:

	publish_date	headline_category	headline_text
0	20010102	unknown	Status quo will not be disturbed at Ayodhya; s...
1	20010102	unknown	Fissures in Hurriyat over Pak visit
2	20010102	unknown	America's unwanted heading for India?
3	20010102	unknown	For bigwigs; it is destination Goa
4	20010102	unknown	Extra buses to clear tourist traffic

In [75]:

```
#converting publish_date column to
news['publish_date'] = pd.to_datetime(news['publish_date'], format = '%Y%m%d')
```

In [76]:

```
#first 5 rows content of the dataset
news.head()
```

Out[76]:

	publish_date	headline_category	headline_text
0	2001-01-02	unknown	Status quo will not be disturbed at Ayodhya; s...
1	2001-01-02	unknown	Fissures in Hurriyat over Pak visit
2	2001-01-02	unknown	America's unwanted heading for India?
3	2001-01-02	unknown	For bigwigs; it is destination Goa
4	2001-01-02	unknown	Extra buses to clear tourist traffic

In [77]:

```
#Last 5 rows content of the dataset
news.tail()
```

Out[77]:

	publish_date	headline_category	headline_text
3650965	2022-03-31	city.srinagar	J&K sacks 2 cops; 3 other employees over terro...
3650966	2022-03-31	entertainment.hindi.bollywood	Ranbir Kapoor says 'Rishi Kapoor enjoyed his a...
3650967	2022-03-31	city.trichy	As Covid-19 cases drop to nil in southern dist...
3650968	2022-03-31	city.eroode	Tamil Nadu sees marginal rise of Covid cases w...
3650969	2022-03-31	city.salem	Tamil Nadu sees marginal rise of Covid cases w...

In [80]:

```
#dropping the Unnamed: 0 column
news.drop('publish_date', inplace=True, axis=1)
```

In [82]:

```
#getting brief overview of the dataset - number of columns and rows (shape of dataset), col
news.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3650970 entries, 0 to 3650969
Data columns (total 2 columns):
#   Column          Dtype
---  -
0   headline_category  object
1   headline_text      object
dtypes: object(2)
memory usage: 55.7+ MB
```

In [83]:

```
#finding unique values in headline_category
news['headline_category'].unique()
```

Out[83]:

```
array(['unknown', 'entertainment.hindi.bollywood', 'india', ...,
      'sports.football.euro-2021', 'business.markets.ipo',
      'sports.tokyo-olympics.india-in-tokyo'], dtype=object)
```

In [84]:

```
#checking all the values count (unique values total count)
news['headline_category'].value_counts()
```

Out[84]:

```
india                297491
unknown             209583
city.mumbai         150451
city.delhi           137648
business.india-business 121195
...
nepal-india-earthquake.opinion      8
elections.lok-sabha-elections-2019.tripura.news      8
best-products.home-decor-and-garden.living-room-decor      8
profiles.india-profiles      8
pms-us-visit      8
Name: headline_category, Length: 1041, dtype: int64
```

In [85]:

```
#total unique values count
news['headline_category'].value_counts().count()
```

Out[85]:

```
1041
```

In [86]:

```
#checking all the values count (unique values total count)  
news['headline_text'].value_counts()
```

Out[86]:

Straight Answers

6723

Cartoons

1536

Straight answers

1500

Photogallery

1353

Your say

1012

...

Pak President Asif Ali Zardari at 7RCR; meets Manmohan Singh

1

Why Goa Inc is so happy with CM Manohar Parrikar

1

American Tourister eyes Rs 100 crore from backpacks in 2012

1

Book 'Ek Hota Carver' soon in audio form

1

Ranbir Kapoor says 'Rishi Kapoor enjoyed his alcohol; food and ensured to have at least one meal together with family' 1

Name: headline_text, Length: 3387380, dtype: int64

In [87]:

```
#total unique values count  
news['headline_text'].value_counts().count()
```

Out[87]:

3387380

In [88]:

```
#finding if any null values are present  
news.isnull().sum().sum()
```

Out[88]:

0

In [89]:

```
#finding if any duplicate values are present  
news.duplicated().sum()
```

Out[89]:

109392

In [90]:

```
#rough checking by marking all duplicates as True. Default is first which marks duplicates
news.duplicated(keep=False).sum()
```

Out[90]:

152566

In [91]:

```
#sorting the dataset to delete the duplicates, to make duplicates come together one after a
cols = list(news.columns)
news.sort_values(by=cols, inplace=True, ignore_index=True)
```

In [92]:

```
news[news.duplicated(keep=False)]
```

Out[92]:

	headline_category	headline_text
91	2011-top-stories	Gold for a billion: Abhinav Bindra
92	2011-top-stories	Gold for a billion: Abhinav Bindra
93	2011-top-stories	Gold for a billion: Abhinav Bindra
94	2011-top-stories	Gold for a billion: Abhinav Bindra
95	2011-top-stories	Gold for a billion: Abhinav Bindra
...
3650841	young-india-votes.gallup-poll	Gallup pollmeter: Kolkata
3650864	young-india-votes.news	Ban on opinion polls: Govt seeks AG's view
3650865	young-india-votes.news	Ban on opinion polls: Govt seeks AG's view
3650869	young-india-votes.news	Campaigns go from austere to extravagant
3650870	young-india-votes.news	Campaigns go from austere to extravagant

152566 rows × 2 columns