DSC680_Project1_jMadsen

April 9, 2023

[1]:
```python
# Justin Madsen

# import the boys
from datetime import datetime, timedelta
from prophet import Prophet
from prophet.plot import plot_plotly, plot_components_plotly,␣
↪plot_cross_validation_metric
from prophet.diagnostics import cross_validation, performance_metrics
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pandas_datareader.data as pdr
import plotly.graph_objects as go
import yfinance as yf
```

[2]:
```python
# yahoofinance changes made datareader brick, so the yahoofinance library␣
↪included a function to bypass
# the changes to the website.
yf.pdr_override()

# create a variable that hold's today's date, then convert to a d/t group
today = datetime.today().strftime('%Y-%m-%d')

# here we use datareader to ingest the historic ethereum prices
eth_ingest = pdr.get_data_yahoo(['ETH-USD'], start=datetime(2016, 1, 1),␣
↪end=today)
```

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

[3]:
```python
# did it work?
eth_ingest.head(2)
```

[3]:
```
                  Open        High         Low       Close   Adj Close  \
Date
2017-11-09  308.644989  329.451996  307.056000  320.884003  320.884003
2017-11-10  320.670990  324.717987  294.541992  299.252991  299.252991

               Volume
```

```
Date
2017-11-09   893249984
2017-11-10   885985984
```

[4]:
```
# let's save the data as a csv incase the yfinance page changes and we can't␣
 ↪pull the data
eth_ingest.to_csv('eth.csv')
```

[5]:
```
# reingest the data
eth_df = pd.read_csv('eth.csv')
```

[6]:
```
# did it work?
eth_df.head(2)
```

[6]:
```
         Date        Open        High         Low       Close   Adj Close  \
0  2017-11-09  308.644989  329.451996  307.056000  320.884003  320.884003
1  2017-11-10  320.670990  324.717987  294.541992  299.252991  299.252991

      Volume
0  893249984
1  885985984
```

[7]:
```
# how much data are we looking at?
eth_df.shape
```

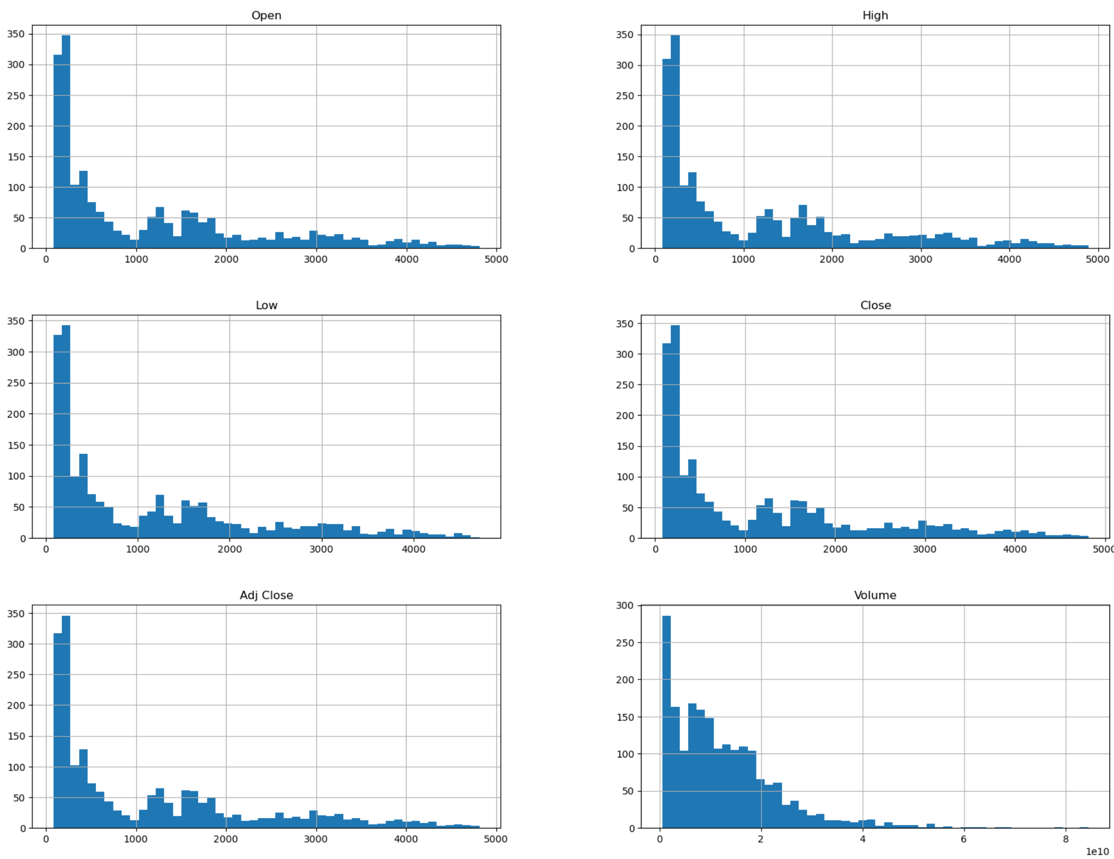
[7]:
```
(1977, 7)
```

[8]:
```
# let's take a look at what the data looks like
eth_df.describe().T
```

[8]:
```
            count          mean           std           min           25%  \
Open       1977.0  1.153107e+03  1.163814e+03  8.427969e+01  2.173270e+02
High       1977.0  1.189362e+03  1.199071e+03  8.534274e+01  2.221826e+02
Low        1977.0  1.112480e+03  1.123251e+03  8.282989e+01  2.096381e+02
Close      1977.0  1.153654e+03  1.163301e+03  8.430830e+01  2.171830e+02
Adj Close  1977.0  1.153654e+03  1.163301e+03  8.430830e+01  2.171830e+02
Volume     1977.0  1.267252e+10  1.058283e+10  6.217330e+08  4.709988e+09

                    50%           75%           max
Open       5.893787e+02  1.746926e+03  4.810071e+03
High       6.085830e+02  1.806539e+03  4.891705e+03
Low        5.685964e+02  1.691658e+03  4.718039e+03
Close      5.896632e+02  1.752045e+03  4.812087e+03
Adj Close  5.896632e+02  1.752045e+03  4.812087e+03
Volume     1.029222e+10  1.774097e+10  8.448291e+10
```

[9]:
```python
# let's plot all the data to look at distributions
eth_df.hist(bins=50, figsize=(20,15))
plt.savefig('eth_hist.png')
plt.show()
```



Close and Adj close look pretty similar. Let's do some comparison.

[10]:
```python
# np.where is useful to create these comparisons in a new column
eth_df['diff'] = np.where(eth_df['Close'] != eth_df['Adj Close'], 'Yes', "No")
```

[11]:
```python
# did it work?
eth_df.head(2)
```

[11]:
```
         Date        Open        High         Low       Close   Adj Close  \
0  2017-11-09  308.644989  329.451996  307.056000  320.884003  320.884003
1  2017-11-10  320.670990  324.717987  294.541992  299.252991  299.252991

      Volume diff
0  893249984   No
1  885985984   No
```

Let's check to see if the Close and Adj Close ever differ

[12]:
```
# value_counts() counts each different type of entry in whatever column we␣
↪specify
eth_df['diff'].value_counts()
```

[12]:
```
No    1977
Name: diff, dtype: int64
```

Since Close and Adj Close are the same, we can go ahead and drop Adj Close. This column adds no value and just takes up space.

[13]:
```
# we can drop the diff column since we already did the comparison
eth_df = eth_df.drop(['Adj Close', 'diff'], axis = 1)
eth_df.head(2)
```

[13]:

	Date	Open	High	Low	Close	Volume
0	2017-11-09	308.644989	329.451996	307.056000	320.884003	893249984
1	2017-11-10	320.670990	324.717987	294.541992	299.252991	885985984

[14]:
```
# let's check the types of the objects
eth_df.dtypes
```

[14]:
```
Date       object
Open      float64
High      float64
Low       float64
Close     float64
Volume      int64
dtype: object
```

Date is an object. Let's go ahead and convert that to an actual datetime group

[15]:
```
eth_df['Date'] = pd.to_datetime(eth_df['Date'], format='%Y-%m-%d')
eth_df.dtypes
```

[15]:
```
Date      datetime64[ns]
Open             float64
High             float64
Low              float64
Close            float64
Volume             int64
dtype: object
```

[16]:
```
# numeric_only is used in order to test for correlation with dates since it's␣
↪not a numerical value
corr_matrix = eth_df.corr(numeric_only=False)
corr_matrix
```

[16]:

	Date	Open	High	Low	Close	Volume
Date	1.000000	0.629282	0.626550	0.633534	0.629352	0.503962
Open	0.629282	1.000000	0.999189	0.998236	0.997744	0.528284
High	0.626550	0.999189	1.000000	0.998055	0.998857	0.538204
Low	0.633534	0.998236	0.998055	1.000000	0.998891	0.509392
Close	0.629352	0.997744	0.998857	0.998891	1.000000	0.525440
Volume	0.503962	0.528284	0.538204	0.509392	0.525440	1.000000

[17]:
```
# open, high, low, close are all almost a flat 1.0 in correlation together.
# building a model on those would be ill advised. Let's do date and close.
input_df = eth_df[['Date', 'Close']]
```

[18]:
```
# fbprophet requires columns to be named ds and y, so let's rename them in a␣
↪new df
input_df = input_df.rename(columns = {'Date': 'ds', 'Close': 'y'})
```

[19]:
```
# did it work?
input_df.head(2)
```

[19]:

	ds	y
0	2017-11-09	320.884003
1	2017-11-10	299.252991

[20]:
```
# here we'll set an x and y based off the column data
x = input_df['ds']
y = input_df['y']

# using plotly, let's create the figure
fig = go.Figure()

# add the line for history of the price
fig.add_trace(go.Scatter(x=x, y=y))

# let's add a title
fig.update_layout(title_text="Etherum Price History")
```

[21]:
```
# here we'll save the image
fig.write_image('eth_price.png')
```

[22]:
```
# create the model using Prophet, and use multiplicative as the seasonality␣
↪since this is a time series model
forecast_model = Prophet(seasonality_mode="multiplicative")
forecast_model.fit(input_df)
```

```
16:07:10 - cmdstanpy - INFO - Chain [1] start processing
16:07:10 - cmdstanpy - INFO - Chain [1] done processing
```

[22]:
```
<prophet.forecaster.Prophet at 0x1c88b9313c0>
```

[23]:
```
# let's set a forecast for 1 year
future_dates = forecast_model.make_future_dataframe(periods = 365)
future_dates.tail()
```

[23]:
```
             ds
2337 2024-04-03
2338 2024-04-04
2339 2024-04-05
2340 2024-04-06
2341 2024-04-07
```

[24]:
```
# Now let's create the forecast off those dates
forecast = forecast_model.predict(future_dates)

# let's grab the forecast dates and their predictions.
forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].tail()
```

[24]:
```
             ds       yhat   yhat_lower   yhat_upper
2337 2024-04-03  27.729676 -2252.649441  2156.760446
2338 2024-04-04  23.604549 -2326.711371  2164.035329
2339 2024-04-05  19.530385 -2257.031090  2123.253921
2340 2024-04-06  15.624360 -2250.241658  2106.686575
2341 2024-04-07  11.718896 -2275.002692  2178.200608
```

[25]:
```
# using datetime, let's create an object that uses the next day
next_day = (datetime.today() + timedelta(days=1)).strftime('%Y-%m-%d')

# what's the forecast for tomorrow?
forecast[forecast['ds'] == next_day]['yhat'].item()
```

[25]: 1388.6476583136105

[26]:
```
# let's plot the forecast
fig2 = plot_plotly(forecast_model, forecast)
fig2
```

[27]:
```
# save the image
fig.write_image('eth_forecast.png')
```

[28]:
```
# now let's plot the trend line, as well as the deviation by year and week
plot_components_plotly(forecast_model, forecast)
```

[29]:
```
# let's get some cross validation
forecast_model_cv = cross_validation(forecast_model, initial="730 days",␣
 ↪period='180 days', horizon='730 days')
forecast_model_cv
```

```
  0%|          | 0/3 [00:00<?, ?it/s]
```

```
16:07:11 - cmdstanpy - INFO - Chain [1] start processing
16:07:11 - cmdstanpy - INFO - Chain [1] done processing
16:07:11 - cmdstanpy - INFO - Chain [1] start processing
16:07:11 - cmdstanpy - INFO - Chain [1] done processing
16:07:11 - cmdstanpy - INFO - Chain [1] start processing
16:07:12 - cmdstanpy - INFO - Chain [1] done processing
```

[29]:

	ds	yhat	yhat_lower	yhat_upper	y	cutoff
0	2020-04-14	118.989870	54.240048	184.604011	157.596390	2020-04-13
1	2020-04-15	120.661759	58.886126	186.419713	153.286896	2020-04-13
2	2020-04-16	122.608373	58.868049	183.331217	172.157379	2020-04-13
3	2020-04-17	126.267111	69.642778	188.718630	171.638580	2020-04-13
4	2020-04-18	131.997180	69.274435	195.063494	186.914001	2020-04-13
…	…	…	…	…	…	…
2185	2023-04-04	6600.230564	4463.294266	8748.476759	1871.005127	2021-04-08
2186	2023-04-05	6628.469267	4535.749458	8778.256973	1909.114014	2021-04-08
2187	2023-04-06	6630.266560	4521.616140	8813.544810	1872.922607	2021-04-08
2188	2023-04-07	6735.758681	4525.768108	8903.723588	1865.636108	2021-04-08
2189	2023-04-08	6860.667871	4649.362796	9109.353520	1849.498169	2021-04-08

[2190 rows x 6 columns]

[30]:

```python
# assign the performance to an object
forecast_model_performance = performance_metrics(forecast_model_cv)
forecast_model_performance
```

[30]:

	horizon	mse	rmse	mae	mape	mdape	\
0	73 days	1.001598e+05	316.480292	223.292154	0.256312	0.236747	
1	74 days	1.054573e+05	324.741970	228.719168	0.260638	0.238435	
2	75 days	1.107734e+05	332.826450	234.132244	0.265034	0.240154	
3	76 days	1.149114e+05	338.985863	238.993898	0.268755	0.243701	
4	77 days	1.188438e+05	344.737241	243.923212	0.272543	0.247385	
..	…	…	…	…	…	…	
653	726 days	1.078257e+07	3283.682126	2727.939734	1.392605	1.038883	
654	727 days	1.081923e+07	3289.260225	2733.217138	1.391730	1.037952	
655	728 days	1.084691e+07	3293.464426	2735.884740	1.390520	1.037937	
656	729 days	1.087631e+07	3297.924856	2738.301868	1.388662	1.037323	
657	730 days	1.091672e+07	3304.046622	2741.854614	1.387753	1.036980	

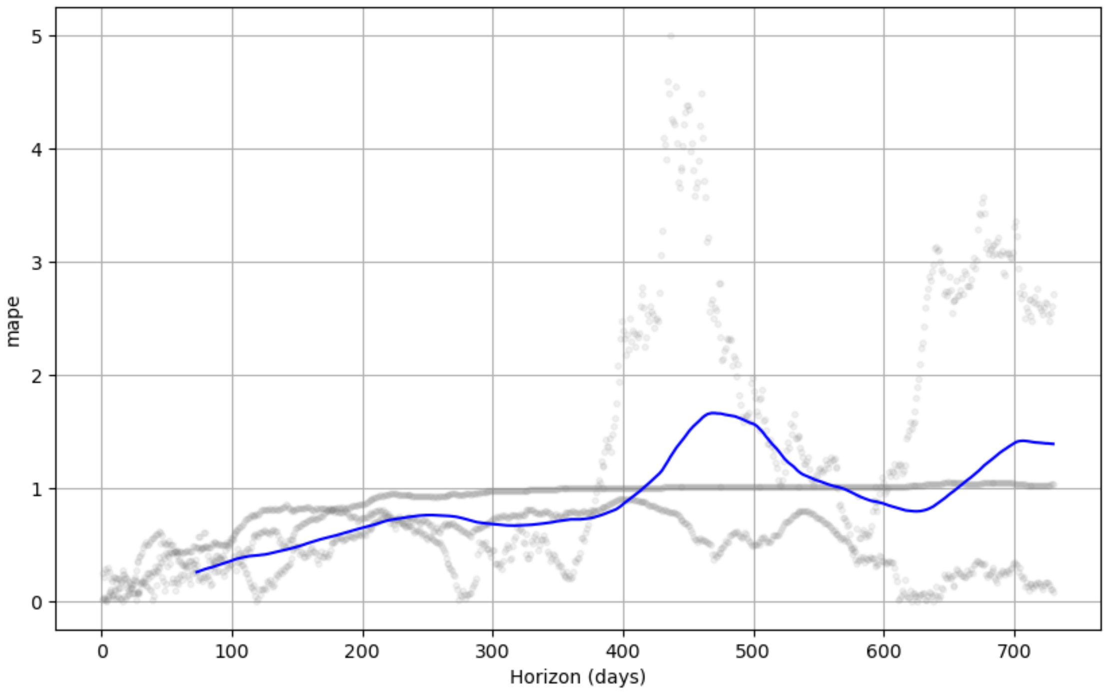
	smape	coverage
0	0.302470	0.342466
1	0.306826	0.328767
2	0.311235	0.315068
3	0.315055	0.301370
4	0.319008	0.287671
..	…	…
653	1.149865	0.333333
654	1.149765	0.333333

```
655  1.148956  0.333333
656  1.147820  0.333333
657  1.146761  0.333333

[658 rows x 8 columns]
```

Here we can see the errors greatly increase the further down the forecast window we go. This is expected as it isn't pulling in new data, and only has a few years of data to go off of.

[31]:
```
# plot the cv values
fig = plot_cross_validation_metric(forecast_model_cv, metric = 'mape')
```



1 references

How to install fbprophet on Windows - https://stackoverflow.com/a/64878241

Facebook.Prophet documentation - https://facebook.github.io/prophet/docs/diagnostics.html

https://github.com/facebook/prophet/blob/main/python/prophet/forecaster.py

[ ]: