databricks MAS 649 Midterm

MAS 649 Midterm

Group 5

Austin Gravely, Elias Eskind, Arlet Rodriguez, Cesar Diez, Juliano Torii

4/6/21

Step 1: Loading & Splitting Data

Loading in Data

```
btc = spark.read.csv('/FileStore/tables/BTC_USD.csv',inferSchema=True,header=True)
eth = spark.read.csv('/FileStore/tables/ETH_USD.csv',inferSchema=True,header=True)
theta = spark.read.csv('/FileStore/tables/THETA_USD.csv',inferSchema=True,header=True)
```

Viewing Data

```
# Display
display(btc)
display(eth)
display(theta)
```

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	12/10/19	7397.134277	7424.022949	7246.043945	7278.119629	18249031195	-0.00833902	null	null	null	null
2	12/11/19	7277.197754	7324.15625	7195.527344	7217.427246	16350490689	0.0035618	-0.00012668	-0.00833902	null	null
3	12/12/19	7216.73877	7266.639648	7164.741211	7243.134277	18927080224	0.003665581	-0.0000954	0.0035618	-0.00833902	null
4	12/13/19	7244.662109	7293.560547	7227.122559	7269.68457	17125736940	-0.019947322	0.000210891	0.003665581	0.0035618	-0.008
5	12/14/19	7268.902832	7308.836426	7097.208984	7124.673828	17137029730	0.003877782	-0.000107546	-0.019947322	0.003665581	0.0035
6	12/15/19	7124.239746	7181.075684	6924.375977	7152.301758	16881129804	-0.030734342	-0.0000609	0.003877782	-0.019947322	0.0036

Showing all 367 rows.

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	12/10/19	148.179855	148.564468	144.907959	146.267044	6859512025	-0.018179365	null	null	null	null
2	12/11/19	146.320648	147.139206	143.045364	143.608002	7037180049	0.013898961	0.000366346	-0.018179365	null	null
3	12/12/19	143.615662	145.751648	141.436981	145.604004	7890383413	-0.004527733	0.0000533	0.013898961	-0.018179365	null

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
4	12/13/19	145.655685	145.857101	143.746521	144.944748	7264810247	-0.01431936	0.000354816	-0.004527733	0.013898961	-0.018
5	12/14/19	144.953415	145.529083	142.434555	142.869232	7048066973	0.00172016	0.0000598	-0.01431936	-0.004527733	0.0138
6	12/15/19	142.86499	143.925354	139.426956	143.11499	7235153411	-0.066386903	-0.0000297	0.00172016	-0.01431936	-0.004

Showing all 367 rows.

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	12/10/19	0.085608	0.085714	0.073951	0.076597	2773792	0.084311396	null	null	null	null
2	12/11/19	0.076638	0.086088	0.076493	0.083055	2147619	0.032785504	0.000534983	0.084311396	null	null
3	12/12/19	0.083027	0.08921	0.078864	0.085778	3673322	0.05091049	-0.00033724	0.032785504	0.084311396	null
4	12/13/19	0.085842	0.092184	0.084736	0.090145	3681541	-0.016196128	0.000745556	0.05091049	0.032785504	0.0843
5	12/14/19	0.090143	0.090722	0.083036	0.088685	1768192	0.077093082	-0.0000222	-0.016196128	0.05091049	0.0327
6	12/15/19	0.088692	0.098924	0.084474	0.095522	5096621	-0.041770482	0.0000789	0.077093082	-0.016196128	0.0509

Showing all 367 rows.

```
# View Schema
btc.printSchema()
eth.printSchema()
theta.printSchema()
```

```
root
 |-- Date: string (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: long (nullable = true)
 |-- ReturnY: double (nullable = true)
 |-- OCDiff: double (nullable = true)
 |-- Lag1: double (nullable = true)
 |-- Lag2: double (nullable = true)
 |-- Lag3: double (nullable = true)
 |-- Lag4: double (nullable = true)

root
 |-- Date: string (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: long (nullable = true)
```

Dropping N/A's

```
btc2=btc.na.drop()
eth2=eth.na.drop()
theta2=theta.na.drop()
```

Changing 'Date' Column from String to Date Datatype

```
from pyspark.sql.functions import to_date

# Bitcoin
btc2 = btc2.withColumn("Date", to_date(btc2['Date'], "M/d/yy"))
btc2.printSchema()

# Ethereum
eth2 = eth2.withColumn("Date", to_date(eth2['Date'], "M/d/yy"))
eth2.printSchema()

# Theta
theta2 = theta2.withColumn("Date", to_date(theta2['Date'], "M/d/yy"))
theta2.printSchema()
```

```
root
 |-- Date: date (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: long (nullable = true)
 |-- ReturnY: double (nullable = true)
 |-- OCDiff: double (nullable = true)
 |-- Lag1: double (nullable = true)
 |-- Lag2: double (nullable = true)
 |-- Lag3: double (nullable = true)
 |-- Lag4: double (nullable = true)

root
 |-- Date: date (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: long (nullable = true)
```

Splitting data into Train & Test

Bitcoin

```
# Train
train_databtc2 = btc2.limit(316)
display(train_databtc2)

#Test
test_databtc2 = btc2.orderBy(btc2["Date"].desc()).limit(30).orderBy("Date")
display(test_databtc2)
```

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	2019-12-14	7268.902832	7308.836426	7097.208984	7124.673828	17137029730	0.003877782	-0.000107546	-0.019947322	0.003665581	0.0035
2	2019-12-15	7124.239746	7181.075684	6924.375977	7152.301758	16881129804	-0.030734342	-0.0000609	0.003877782	-0.019947322	0.0036
3	2019-12-16	7153.663086	7171.168945	6903.682617	6932.480469	20213265950	-0.042115565	0.000190298	-0.030734342	0.003877782	-0.019
4	2019-12-17	6931.31543	6964.075195	6587.974121	6640.515137	22363804217	0.095819012	-0.000168083	-0.042115565	-0.030734342	0.0038
5	2019-12-18	6647.698242	7324.984863	6540.049316	7276.802734	31836522778	-0.010163598	0.00108054	0.095819012	-0.042115565	-0.030
6	2019-12-19	7277.59082	7346.602539	7041.381836	7202.844238	25904604416	0.002217481	0.000108289	-0.010163598	0.095819012	-0.042

Showing all 316 rows.

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	2020-11-10	15332.35059	15450.3291	15124.95996	15290.90234	25574938143	0.026841941	0.00000229	-0.002701033	-0.009512666	0.0435
2	2020-11-11	15290.90918	15916.26074	15290.00684	15701.33984	29772374934	0.036621327	4.47e-7	0.026841941	-0.002701033	-0.009
3	2020-11-12	15701.29883	16305.00391	15534.77148	16276.34375	34175758344	0.002547553	-0.00000261	0.036621327	0.026841941	-0.002
4	2020-11-13	16276.44043	16463.17773	15992.15234	16317.80859	31599492172	-0.015300457	0.00000594	0.002547553	0.036621327	0.0268
5	2020-11-14	16317.80859	16317.80859	15749.19336	16068.13867	27481710135	-0.007004594	0	-0.015300457	0.002547553	0.0366
6	2020-11-15	16068.13965	16123.11035	15793.53418	15955.58789	23653867583	0.047665021	6.07e-8	-0.007004594	-0.015300457	0.0025

Showing all 30 rows.

Ethereum

```
# Train
train_dataeth2 = eth2.limit(316)
display(train_dataeth2)

#Test
test_dataeth2 = eth2.orderBy(eth2["Date"].desc()).limit(30).orderBy("Date")
display(test_dataeth2)
```

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	2019-12-14	144.953415	145.529083	142.434555	142.869232	7048066973	0.00172016	0.0000598	-0.01431936	-0.004527733	0.0138
2	2019-12-15	142.86499	143.925354	139.426956	143.11499	7235153411	-0.066386903	-0.0000297	0.00172016	-0.01431936	-0.004
3	2019-12-16	143.139526	143.224854	132.456665	133.614029	8992282119	-0.082402575	0.000171413	-0.066386903	0.00172016	-0.014
4	2019-12-17	133.647186	134.011536	121.395081	122.603889	9057166141	0.085546267	0.000248094	-0.082402575	-0.066386903	0.0017
5											

6	2019-12-19	133.05278	134.190643	125.971664	129.321136	9564699140	-0.001972462	-0.000296228	-0.028334179	0.085546267	-0.082

Showing all 316 rows.

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	2020-11-10	444.166382	453.758362	439.600128	449.679626	12090381666	0.029534171	0.00000749	0.012420148	-0.020706923	0.0409
2	2020-11-11	449.679657	473.578857	449.524933	462.960541	14075403511	-0.004223386	6.89e-8	0.029534171	0.012420148	-0.020
3	2020-11-12	462.959534	467.677826	452.072418	461.00528	12877327234	0.029546633	-0.00000218	-0.004223386	0.029534171	0.0124
4	2020-11-13	461.005493	475.217255	457.298248	474.626434	13191505725	-0.030501026	4.62e-7	0.029546633	-0.004223386	0.0295
5	2020-11-14	474.626434	475.161438	452.986084	460.149841	10312037942	-0.027362302	0	-0.030501026	0.029546633	-0.004
6	2020-11-15	460.149902	460.99408	440.254333	447.559082	10308617165	0.027663892	1.33e-7	-0.027362302	-0.030501026	0.0295

Showing all 30 rows.

Theta

```
# Train
train_datatheta2 = theta2.limit(316)
display(train_datatheta2)

#Test
test_datatheta2 = theta2.orderBy(theta2["Date"].desc()).limit(30).orderBy("Date")
display(test_datatheta2)
```

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	2019-12-14	0.090143	0.090722	0.083036	0.088685	1768192	0.077093082	-0.0000222	-0.016196128	0.05091049	0.0327
2	2019-12-15	0.088692	0.098924	0.084474	0.095522	5096621	-0.041770482	0.0000789	0.077093082	-0.016196128	0.0509
3	2019-12-16	0.095568	0.100991	0.088694	0.091532	5611689	0.032360267	0.000481333	-0.041770482	0.077093082	-0.016
4	2019-12-17	0.091532	0.10091	0.089191	0.094494	7840056	0.071464855	0	0.032360267	-0.041770482	0.0770
5	2019-12-18	0.094494	0.104688	0.091284	0.101247	6151333	0.063873497	0	0.071464855	0.032360267	-0.041
6	2019-12-19	0.101247	0.112162	0.100012	0.107714	5951808	-0.01707299	0	0.063873497	0.071464855	0.0323

Showing all 316 rows.

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	2020-11-10	0.641028	0.657987	0.623313	0.641003	19591349	-0.044586375	-0.00000156	-0.0000406	-0.035165933	0.0383
2	2020-11-11	0.641003	0.653566	0.612423	0.612423	18730118	-0.017145013	0	-0.044586375	-0.0000406	-0.035
3	2020-11-12	0.612423	0.626791	0.590405	0.601923	16643669	0.052284096	0	-0.017145013	-0.044586375	-0.000
4	2020-11-13	0.601928	0.634515	0.595535	0.633394	22198608	-0.033803604	0.00000831	0.052284096	-0.017145013	-0.044
5	2020-11-14	0.633394	0.63766	0.595949	0.611983	9700736	-0.013719335	0	-0.033803604	0.052284096	-0.017
6	2020-11-15	0.611983	0.62249	0.59375	0.603587	8911596	0.025005509	0	-0.013719335	-0.033803604	0.0522

Showing all 30 rows.

Transforming Data

```
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols = ['Open',
 'High',
 'Low',
 'Close',
 'Volume',
 'OCDiff',
 'Lag1',
 'Lag2',
 'Lag3',
 'Lag4'], outputCol = "features")


# Bitcoin
outputTrainbtc2 = assembler.transform(train_databtc2)
display(outputTrainbtc2)

outputTestbtc2 = assembler.transform(test_databtc2)
display(outputTestbtc2)

# Ethereum
outputTraineth2 = assembler.transform(train_dataeth2)
display(outputTraineth2)

outputTesteth2  = assembler.transform(test_dataeth2)
display(outputTesteth2)

# Theta
outputTraintheta2 = assembler.transform(train_datatheta2)
display(outputTraintheta2)

outputTesttheta2  = assembler.transform(test_datatheta2)
display(outputTesttheta2)
```

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	2019-12-14	7268.902832	7308.836426	7097.208984	7124.673828	17137029730	0.003877782	-0.000107546	-0.019947322	0.003665581	0.0035
2	2019-12-15	7124.239746	7181.075684	6924.375977	7152.301758	16881129804	-0.030734342	-0.0000609	0.003877782	-0.019947322	0.0036
3	2019-12-16	7153.663086	7171.168945	6903.682617	6932.480469	20213265950	-0.042115565	0.000190298	-0.030734342	0.003877782	-0.019
4	2019-12-17	6931.31543	6964.075195	6587.974121	6640.515137	22363804217	0.095819012	-0.000168083	-0.042115565	-0.030734342	0.0038
5	2019-12-18	6647.698242	7324.984863	6540.049316	7276.802734	31836522778	-0.010163598	0.00108054	0.095819012	-0.042115565	-0.030

6	2019-12-19	7277.59082	7346.602539	7041.381836	7202.844238	25904604416	0.002217481	0.000108289	-0.010163598	0.095819012	-0.042

Showing all 316 rows.

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	2020-11-10	15332.35059	15450.3291	15124.95996	15290.90234	25574938143	0.026841941	0.00000229	-0.002701033	-0.009512666	0.0435
2	2020-11-11	15290.90918	15916.26074	15290.00684	15701.33984	29772374934	0.036621327	4.47e-7	0.026841941	-0.002701033	-0.009
3	2020-11-12	15701.29883	16305.00391	15534.77148	16276.34375	34175758344	0.002547553	-0.00000261	0.036621327	0.026841941	-0.002
4	2020-11-13	16276.44043	16463.17773	15992.15234	16317.80859	31599492172	-0.015300457	0.00000594	0.002547553	0.036621327	0.0268
5	2020-11-14	16317.80859	16317.80859	15749.19336	16068.13867	27481710135	-0.007004594	0	-0.015300457	0.002547553	0.0366
6	2020-11-15	16068.13965	16123.11035	15793.53418	15955.58789	23653867583	0.047665021	6.07e-8	-0.007004594	-0.015300457	0.0025

Showing all 30 rows.

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	2019-12-14	144.953415	145.529083	142.434555	142.869232	7048066973	0.00172016	0.0000598	-0.01431936	-0.004527733	0.0138
2	2019-12-15	142.86499	143.925354	139.426956	143.11499	7235153411	-0.066386903	-0.0000297	0.00172016	-0.01431936	-0.004
3	2019-12-16	143.139526	143.224854	132.456665	133.614029	8992282119	-0.082402575	0.000171413	-0.066386903	0.00172016	-0.014
4	2019-12-17	133.647186	134.011536	121.395081	122.603889	9057166141	0.085546267	0.000248094	-0.082402575	-0.066386903	0.0017
5	2019-12-18	122.656837	133.394165	119.78006	133.092194	11864518321	-0.028334179	0.000431676	0.085546267	-0.082402575	-0.066
6	2019-12-19	133.05278	134.190643	125.971664	129.321136	9564699140	-0.001972462	-0.000296228	-0.028334179	0.085546267	-0.082

Showing all 316 rows.

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	2020-11-10	444.166382	453.758362	439.600128	449.679626	12090381666	0.029534171	0.00000749	0.012420148	-0.020706923	0.0409
2	2020-11-11	449.679657	473.578857	449.524933	462.960541	14075403511	-0.004223386	6.89e-8	0.029534171	0.012420148	-0.020
3	2020-11-12	462.959534	467.677826	452.072418	461.00528	12877327234	0.029546633	-0.00000218	-0.004223386	0.029534171	0.0124
4	2020-11-13	461.005493	475.217255	457.298248	474.626434	13191505725	-0.030501026	4.62e-7	0.029546633	-0.004223386	0.0295

5	2020-11-14	474.626434	475.161438	452.986084	460.149841	10312037942	-0.027362302	0	-0.030501026	0.029546633	-0.004
6	2020-11-15	460.149902	460.99408	440.254333	447.559082	10308617165	0.027663892	1.33e-7	-0.027362302	-0.030501026	0.0295

Showing all 30 rows.

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	2019-12-14	0.090143	0.090722	0.083036	0.088685	1768192	0.077093082	-0.0000222	-0.016196128	0.05091049	0.0327
2	2019-12-15	0.088692	0.098924	0.084474	0.095522	5096621	-0.041770482	0.0000789	0.077093082	-0.016196128	0.0509
3	2019-12-16	0.095568	0.100991	0.088694	0.091532	5611689	0.032360267	0.000481333	-0.041770482	0.077093082	-0.016
4	2019-12-17	0.091532	0.10091	0.089191	0.094494	7840056	0.071464855	0	0.032360267	-0.041770482	0.0770
5	2019-12-18	0.094494	0.104688	0.091284	0.101247	6151333	0.063873497	0	0.071464855	0.032360267	-0.041
6	2019-12-19	0.101247	0.112162	0.100012	0.107714	5951808	-0.01707299	0	0.063873497	0.071464855	0.0323

Showing all 316 rows.

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	2020-11-10	0.641028	0.657987	0.623313	0.641003	19591349	-0.044586375	-0.00000156	-0.0000406	-0.035165933	0.0383
2	2020-11-11	0.641003	0.653566	0.612423	0.612423	18730118	-0.017145013	0	-0.044586375	-0.0000406	-0.035
3	2020-11-12	0.612423	0.626791	0.590405	0.601923	16643669	0.052284096	0	-0.017145013	-0.044586375	-0.000
4	2020-11-13	0.601928	0.634515	0.595535	0.633394	22198608	-0.033803604	0.00000831	0.052284096	-0.017145013	-0.044
5	2020-11-14	0.633394	0.63766	0.595949	0.611983	9700736	-0.013719335	0	-0.033803604	0.052284096	-0.017
6	2020-11-15	0.611983	0.62249	0.59375	0.603587	8911596	0.025005509	0	-0.013719335	-0.033803604	0.0522

Showing all 30 rows.

Final Datasets

Bitcoin

```
# Final Train
trainBtc = outputTrainbtc2.select("features", "ReturnY", "Date")
display(trainBtc)

# Final Test
testBtc = outputTestbtc2.select("features", "ReturnY", "Date")
display(testBtc)

# Displaying
display(trainBtc.describe())
display(testBtc.describe())
```

	features	ReturnY	Date
1	{"vectorType": "dense", "length": 10, "values": [7268.902832, 7308.836426, 7097.208984, 7124.673828, 17137029730, -0.000107546, -0.019947322, 0.003665581, 0.0035618, -0.00833902]}	0.003877782	2019-12-14
2	{"vectorType": "dense", "length": 10, "values": [7124.239746, 7181.075684, 6924.375977, 7152.301758, 16881129804, -0.0000609, 0.003877782, -0.019947322, 0.003665581, 0.0035618]}	-0.030734342	2019-12-15
3	{"vectorType": "dense", "length": 10, "values": [7153.663086, 7171.168945, 6903.682617, 6932.480469, 20213265950, 0.000190298, -0.030734342, 0.003877782, -0.019947322, 0.003665581]}	-0.042115565	2019-12-16
4	{"vectorType": "dense", "length": 10, "values": [6931.31543, 6964.075195, 6587.974121, 6640.515137, 22363804217, -0.000168083, -0.042115565, -0.030734342, 0.003877782, -0.019947322]}	0.095819012	2019-12-17
5	{"vectorType": "dense", "length": 10, "values": [6647.698242, 7324.984863, 6540.049316, 7276.802734, 31836522778, 0.00108054, 0.095819012, -0.042115565, -0.030734342, 0.003877782]}	-0.010163598	2019-12-18
6	{"vectorType": "dense", "length": 10, "values": [7277.59082, 7346.602539, 7041.381836, 7202.844238, 25904604416, 0.000108289, -0.010163598, 0.095819012, -0.042115565, -0.030734342]}	0.002217481	2019-12-19

Showing all 316 rows.

	features	ReturnY	Date
1	{"vectorType": "dense", "length": 10, "values": [15332.35059, 15450.3291, 15124.95996, 15290.90234, 25574938143, 0.00000229, -0.002701033, -0.009512666, 0.043536753, -0.047034084]}	0.026841941	2020-11-10
2	{"vectorType": "dense", "length": 10, "values": [15290.90918, 15916.26074, 15290.00684, 15701.33984, 29772374934, 4.47e-7, 0.026841941, -0.002701033, -0.009512666, 0.043536753]}	0.036621327	2020-11-11
3	{"vectorType": "dense", "length": 10, "values": [15701.29883, 16305.00391, 15534.77148, 16276.34375, 34175758344, -0.00000261, 0.036621327, 0.026841941, -0.002701033, -0.009512666]}	0.002547553	2020-11-12
4	{"vectorType": "dense", "length": 10, "values": [16276.44043, 16463.17773, 15992.15234, 16317.80859, 31599492172, 0.00000594, 0.002547553, 0.036621327, 0.026841941, -0.002701033]}	-0.015300457	2020-11-13
5	{"vectorType": "dense", "length": 10, "values": [16317.80859, 16317.80859, 15749.19336, 16068.13867, 27481710135, 0, -0.015300457, 0.002547553, 0.036621327, 0.026841941]}	-0.007004594	2020-11-14
6	{"vectorType": "dense", "length": 10, "values": [16068.13965, 16123.11035, 15793.53418, 15955.58789, 23653867583, 6.07e-8, -0.007004594, -0.015300457, 0.002547553, 0.036621327]}	0.047665021	2020-11-15

Showing all 30 rows.

	summary	ReturnY
1	count	316
2	mean	0.0030648423259493672

3	stddev	0.03849974767594246
4	min	-0.371695386
5	max	0.181877557

Showing all 5 rows.

	summary	ReturnY
1	count	30
2	mean	0.006498241933333333
3	stddev	0.03387664317332147
4	min	-0.084427067
5	max	0.079678328

Showing all 5 rows.

Ethereum

```
#Final Train
trainEth = outputTraineth2.select("features", "ReturnY", "Date")
display(trainEth)

#Final Test
testEth = outputTesteth2.select("features", "ReturnY", "Date")
display(testEth)

# Displaying
display(trainEth.describe())
display(testEth.describe())
```

	features	ReturnY	Date
1	{"vectorType": "dense", "length": 10, "values": [144.953415, 145.529083, 142.434555, 142.869232, 7048066973, 0.0000598, -0.01431936, -0.004527733, 0.013898961, -0.018179365]}	0.00172016	2019-12-14
2	{"vectorType": "dense", "length": 10, "values": [142.86499, 143.925354, 139.426956, 143.11499, 7235153411, -0.0000297, 0.00172016, -0.01431936, -0.004527733, 0.013898961]}	-0.066386903	2019-12-15
3	{"vectorType": "dense", "length": 10, "values": [143.139526, 143.224854, 132.456665, 133.614029, 8992282119, 0.000171413, -0.066386903, 0.00172016, -0.01431936, -0.004527733]}	-0.082402575	2019-12-16
4	{"vectorType": "dense", "length": 10, "values": [133.647186, 134.011536, 121.395081, 122.603889, 9057166141, 0.000248094, -0.082402575, -0.066386903, 0.00172016, -0.01431936]}	0.085546267	2019-12-17
5	{"vectorType": "dense", "length": 10, "values": [122.656837, 133.394165, 119.78006, 133.092194, 11864518321, 0.000431676, 0.085546267, -0.082402575, -0.066386903, 0.00172016]}	-0.028334179	2019-12-18
6	{"vectorType": "dense", "length": 10, "values": [133.05278, 134.190643, 125.971664, 129.321136, 9564699140, -0.000296228, -0.028334179, 0.085546267, -0.082402575, -0.066386903]}	-0.001972462	2019-12-19

Showing all 316 rows.

	features	ReturnY	Date
1	{"vectorType": "dense", "length": 10, "values": [444.166382, 453.758362, 439.600128, 449.679626, 12090381666, 0.00000749, 0.012420148, -0.020706923, 0.040948144, -0.041797575]}	0.029534171	2020-11-10
2	{"vectorType": "dense", "length": 10, "values": [449.679657, 473.578857, 449.524933, 462.960541, 14075403511, 6.89e-8, 0.029534171, 0.012420148, -0.020706923, 0.040948144]}	-0.004223386	2020-11-11
3	{"vectorType": "dense", "length": 10, "values": [462.959534, 467.677826, 452.072418, 461.00528, 12877327234, -0.00000218, -0.004223386, 0.029534171, 0.012420148, -0.020706923]}	0.029546633	2020-11-12
4	{"vectorType": "dense", "length": 10, "values": [461.005493, 475.217255, 457.298248, 474.626434, 13191505725, 4.62e-7, 0.029546633, -0.004223386, 0.029534171, 0.012420148]}	-0.030501026	2020-11-13
5	{"vectorType": "dense", "length": 10, "values": [474.626434, 475.161438, 452.986084, 460.149841, 10312037942, 0, -0.030501026, 0.029546633, -0.004223386, 0.029534171]}	-0.027362302	2020-11-14
6	{"vectorType": "dense", "length": 10, "values": [460.149902, 460.99408, 440.254333, 447.559082, 10308617165, 1.33e-7, -0.027362302, -0.030501026, 0.029546633, -0.004223386]}	0.027663892	2020-11-15

Showing all 30 rows.

	summary	ReturnY
1	count	316
2	mean	0.0045853581835443025
3	stddev	0.05037579196523408
4	min	-0.423472215
5	max	0.189404015

Showing all 5 rows.

	summary	ReturnY
1	count	30
2	mean	0.0084080907
3	stddev	0.04742481817481735
4	min	-0.090917668
5	max	0.090286337

Showing all 5 rows.

Theta

```
#Final Train
trainTheta = outputTraintheta2.select("features", "ReturnY", "Date")
display(trainTheta)

#Final Test
testTheta = outputTesttheta2.select("features", "ReturnY", "Date")
display(testTheta)

# Displaying
display(trainTheta.describe())
display(testTheta.describe())
```

	features	ReturnY	Date
1	{"vectorType": "dense", "length": 10, "values": [0.090143, 0.090722, 0.083036, 0.088685, 1768192, -0.0000222, -0.016196128, 0.05091049, 0.032785504, 0.084311396]}	0.077093082	2019-12-14
2	{"vectorType": "dense", "length": 10, "values": [0.088692, 0.098924, 0.084474, 0.095522, 5096621, 0.0000789, 0.077093082, -0.016196128, 0.05091049, 0.032785504]}	-0.041770482	2019-12-15
3	{"vectorType": "dense", "length": 10, "values": [0.095568, 0.100991, 0.088694, 0.091532, 5611689, 0.000481333, -0.041770482, 0.077093082, -0.016196128, 0.05091049]}	0.032360267	2019-12-16
4	{"vectorType": "dense", "length": 10, "values": [0.091532, 0.10091, 0.089191, 0.094494, 7840056, 0, 0.032360267, -0.041770482, 0.077093082, -0.016196128]}	0.071464855	2019-12-17
5	{"vectorType": "dense", "length": 10, "values": [0.094494, 0.104688, 0.091284, 0.101247, 6151333, 0, 0.071464855, 0.032360267, -0.041770482, 0.077093082]}	0.063873497	2019-12-18
6	{"vectorType": "dense", "length": 10, "values": [0.101247, 0.112162, 0.100012, 0.107714, 5951808, 0, 0.063873497, 0.071464855, 0.032360267, -0.041770482]}	-0.01707299	2019-12-19

Showing all 316 rows.

	features	ReturnY	Date
1	{"vectorType": "dense", "length": 10, "values": [0.641028, 0.657987, 0.623313, 0.641003, 19591349, -0.00000156, -0.0000406, -0.035165933, 0.038349314, -0.053501138]}	-0.044586375	2020-11-10
2	{"vectorType": "dense", "length": 10, "values": [0.641003, 0.653566, 0.612423, 0.612423, 18730118, 0, -0.044586375, -0.0000406, -0.035165933, 0.038349314]}	-0.017145013	2020-11-11
3	{"vectorType": "dense", "length": 10, "values": [0.612423, 0.626791, 0.590405, 0.601923, 16643669, 0, -0.017145013, -0.044586375, -0.0000406, -0.035165933]}	0.052284096	2020-11-12
4	{"vectorType": "dense", "length": 10, "values": [0.601928, 0.634515, 0.595535, 0.633394, 22198608, 0.00000831, 0.052284096, -0.017145013, -0.044586375, -0.0000406]}	-0.033803604	2020-11-13
5	{"vectorType": "dense", "length": 10, "values": [0.633394, 0.63766, 0.595949, 0.611983, 9700736, 0, -0.033803604, 0.052284096, -0.017145013, -0.044586375]}	-0.013719335	2020-11-14
6	{"vectorType": "dense", "length": 10, "values": [0.611983, 0.62249, 0.59375, 0.603587, 8911596, 0, -0.013719335, -0.033803604, 0.052284096, -0.017145013]}	0.025005509	2020-11-15

Showing all 30 rows.

	summary	ReturnY
1	count	316
2	mean	0.008329582161392403

3	stddev	0.07320447509758738
4	min	-0.453309413
5	max	0.369398964

Showing all 5 rows.

	summary	ReturnY
1	count	30
2	mean	0.0034768952333333334
3	stddev	0.0509510416220618
4	min	-0.107197389
5	max	0.105163087

Showing all 5 rows.

Step 2: ML Algorithm

Importing Required Packages

```
from pyspark.ml.regression import RandomForestRegressor, GBTRegressor, DecisionTreeRegressor, LinearRegression
```

```
dt = DecisionTreeRegressor(labelCol = "ReturnY", featuresCol = "features")
rf = RandomForestRegressor(labelCol = "ReturnY", featuresCol = "features")
gb = GBTRegressor(labelCol = "ReturnY", featuresCol = "features")
```

Training Models

```
# Bitcoin
dt_modelBtc = dt.fit(trainBtc)
rf_modelBtc = rf.fit(trainBtc)
gb_modelBtc = gb.fit(trainBtc)

# Ethereum
dt_modelEth = dt.fit(trainEth)
rf_modelEth = rf.fit(trainEth)
gb_modelEth = gb.fit(trainEth)

# Theta
dt_modelTheta = dt.fit(trainTheta)
rf_modelTheta = rf.fit(trainTheta)
gb_modelTheta = gb.fit(trainTheta)
```

Testing Models

```
# Bitcoin
dt_predictionsBtc = dt_modelBtc.transform(testBtc)
rf_predictionsBtc = rf_modelBtc.transform(testBtc)
gb_predictionsBtc = gb_modelBtc.transform(testBtc)

# Ethereum
dt_predictionsEth = dt_modelEth.transform(testEth)
rf_predictionsEth = rf_modelEth.transform(testEth)
gb_predictionsEth = gb_modelEth.transform(testEth)

# Theta
dt_predictionsTheta = dt_modelTheta.transform(testTheta)
rf_predictionsTheta = rf_modelTheta.transform(testTheta)
gb_predictionsTheta = gb_modelTheta.transform(testTheta)
```

Evaluator

```
from pyspark.ml.evaluation import RegressionEvaluator
evaluator = RegressionEvaluator(labelCol="ReturnY", predictionCol="prediction", metricName="rmse")

# Bitcoin
print('Bitcoin:')
print(evaluator.evaluate(dt_predictionsBtc))
print(evaluator.evaluate(rf_predictionsBtc))
print(evaluator.evaluate(gb_predictionsBtc))

# Ethereum
print('Ethereum:')
print(evaluator.evaluate(dt_predictionsEth))
print(evaluator.evaluate(rf_predictionsEth))
print(evaluator.evaluate(gb_predictionsEth))

# Theta
print('Theta:')
print(evaluator.evaluate(dt_predictionsTheta))
print(evaluator.evaluate(rf_predictionsTheta))
print(evaluator.evaluate(gb_predictionsTheta))
```

```
Bitcoin:
0.035215985893660956
0.03640528574677352
0.03828398855111902
Ethereum:
0.06137721727207424
0.04927536011320538
0.0679509252807517
Theta:
0.055323597787211444
0.048447958485873376
0.05667084581327231
```

Hyperparameter RF

```
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

paramGrid = ParamGridBuilder().addGrid(rf.maxDepth, [2,4,6]).addGrid(rf.maxBins, [20,60]).build()

crossval = CrossValidator(estimator = rf, estimatorParamMaps=paramGrid, evaluator = evaluator, numFolds = 10)


# Training Models
cv_modelTheta = crossval.fit(trainTheta)
cv_modelEth = crossval.fit(trainEth)
cv_modelBtc = crossval.fit(trainBtc)

# RMSE:
# Bitcoin
print('Bitcoin:')
cv_predictionsBtc = cv_modelBtc.transform(testBtc)
print(evaluator.evaluate(cv_predictionsBtc))

# Ethereum
print('Ethereum:')
cv_predictionsEth = cv_modelEth.transform(testEth)
print(evaluator.evaluate(cv_predictionsEth))

# Theta
print('Theta:')
cv_predictionsTheta = cv_modelTheta.transform(testTheta)
print(evaluator.evaluate(cv_predictionsTheta))

# You may need to install mlflow if this chunk is taking forever
# Even so this chunk will take 5 minutes or so
```

```
MLlib will automatically track trials in MLflow. After your tuning fit() call has completed, view the MLflow UI to see logged runs.
MLlib will automatically track trials in MLflow. After your tuning fit() call has completed, view the MLflow UI to see logged runs.
MLlib will automatically track trials in MLflow. After your tuning fit() call has completed, view the MLflow UI to see logged runs.
Bitcoin:
0.03839393151577049
Ethereum:
0.04895113745563109
Theta:
0.05044714477677209
```

Elastic Net Hyperparameter Tuning

```
# Importing Packages
from pyspark.ml.regression import LinearRegression
lr = LinearRegression(labelCol="ReturnY", featuresCol="features")
lrModel = lr.fit(trainTheta)
lrModel

# Training Models
paramGrid = ParamGridBuilder().addGrid(lr.regParam, [0.1, 0.01]).addGrid(lr.elasticNetParam, [0, 0.5, 1]).build()
crossval = CrossValidator(estimator=lr, estimatorParamMaps=paramGrid, evaluator=evaluator, numFolds=10)
cv_modelTheta = crossval.fit(trainTheta)
cv_modelEth = crossval.fit(trainEth)
cv_modelBtc = crossval.fit(trainBtc)

# RMSE:
# Bitcoin
print('Bitcoin:')
cv2_predictionsBtc = cv_modelBtc.transform(testBtc)
print(evaluator.evaluate(cv2_predictionsBtc))

# Ethereum
print('Ethereum:')
cv2_predictionsEth = cv_modelEth.transform(testEth)
print(evaluator.evaluate(cv2_predictionsEth))

# Theta
print('Theta:')
cv2_predictionsTheta = cv_modelTheta.transform(testTheta)
print(evaluator.evaluate(cv2_predictionsTheta))
```

```
MLlib will automatically track trials in MLflow. After your tuning fit() call has completed, view the MLflow UI to see logged runs.
MLlib will automatically track trials in MLflow. After your tuning fit() call has completed, view the MLflow UI to see logged runs.
MLlib will automatically track trials in MLflow. After your tuning fit() call has completed, view the MLflow UI to see logged runs.
Bitcoin:
0.0345924050724973
Ethereum:
0.048332895468184395
Theta:
0.04896190875934517
```

```
# import numpy as np
# print(cv_model.getEstimatorParamMaps()[np.argmax(cv_model.avgMetrics)])
# print(cv_model.bestModel._java_obj.getRegParam())
# print(cv_model.bestModel._java_obj.getElasticNetParam())
```

Basic Linear Regression (for Ethereum since this Model had the Best RMSE)

```
from pyspark.ml.regression import LinearRegression
ethlr = LinearRegression(labelCol = "ReturnY", featuresCol = "features", solver = 'l-bfgs', regParam = 0.2, elasticNetParam = 0.6)

lrModelEth = ethlr.fit(trainEth)
# printing coeff and intercept
print("Coefficients: %s" % str(lrModelEth.coefficients))
print("Intercept: %s" % str(lrModelEth.intercept))

# more details about model
trainingSummary = lrModelEth.summary
trainingSummary.residuals.show()
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("r2: %f" % trainingSummary.r2)
```

```
Coefficients: (10,[],[])
Intercept: 0.004585358183544301
+--------------------+
|           residuals|
+--------------------+
|-0.00286519818354...|
| -0.0709722611835443|
|-0.08698793318354431|
| 0.08096090881645569|
|-0.03291953718354...|
| -0.0065578201835443|
| -0.0118305161835443|
|0.028517978816455694|
| -0.0329781331835443|
|-0.00108808218354...|
| -0.0248021131835443|
|-0.00532408118354...|
|0.002168184816455699|
|0.004125127816455699|
|  0.0455637708164557|
| -0.0203506281835443|
```

```
#RMSE
testResultsEth = lrModelEth.evaluate(testEth)
rmseEth = testResultsEth.rootMeanSquaredError
rmseEth
```

```
Out[25]: 0.04678414493012603
```

Step 3: Testing Best Models on Dataset

Bitcoin

```
# Bitcoin Model with Lowest RMSE = ELASTIC NET HYPERPARAMETER TURNING
cv2_predictionsBtc = cv_modelBtc.transform(testBtc)
rmseBtc = evaluator.evaluate(cv2_predictionsBtc)
rmseBtc
```

```
Out[26]: 0.0345924050724973
```

Ethereum

```
# Ethereum Model with Lowest RMSE = LINEAR REGRESSION
testResultsEth = lrModelEth.evaluate(testEth)
rmseEth = testResultsEth.rootMeanSquaredError
rmseEth
```

```
Out[27]: 0.04678414493012603
```

Theta

```
# Theta Model with Lowest RMSE = RANDOM FOREST
rf_predictionsTheta = rf_modelTheta.transform(testTheta)
rmseTheta = evaluator.evaluate(rf_predictionsTheta)
rmseTheta
```

```
Out[28]: 0.048447958485873376
```

Step 4: Average the RMSE

```
import numpy as np
rmseList = [rmseBtc, rmseEth, rmseTheta]
rmseMean = np.mean(rmseList)
print(rmseMean)
```

```
0.04327483616283224
```

The Avergae RMSE for BTC, ETH, & THETA is: *0.04327*

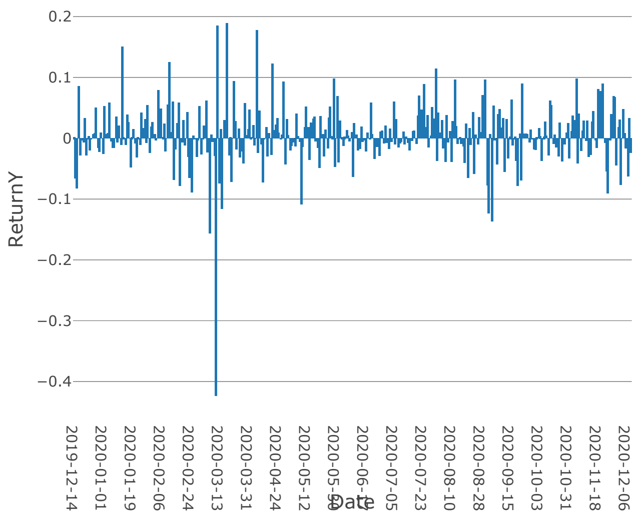
Step 5: Visualizations

Ethereum Graphs

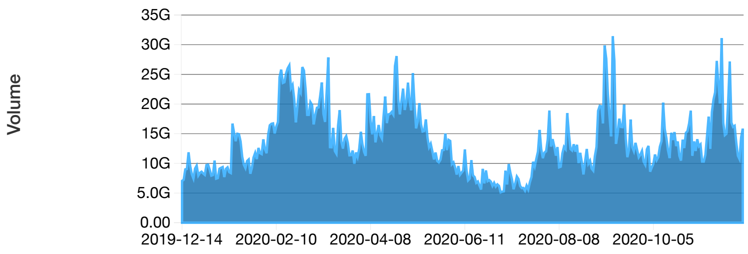
```
print(btc2.filter(btc2["Date"] > "2020-11-09").count())
print(eth2.filter(eth2["Date"] > "2020-11-09").count())
print(theta2.filter(theta2["Date"] > "2020-11-09").count())
```

```
30
30
30
```

```
display(eth2.select("ReturnY", "Date"))
```




```
display(eth2.select("Volume", "Date"))
```




```
display(eth2.select("Close","Low", "High", "Date"))
```

Candlestick Graphs (Extra)

```
import plotly.graph_objects as go
```

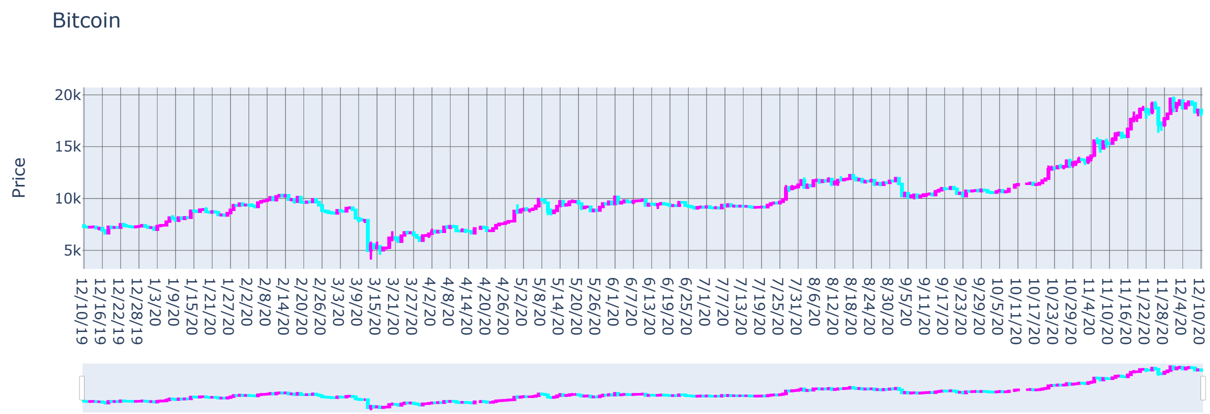
Bitcoin

```
#Converting dataframe from Pyspark to Pandas
df_btc = btc.toPandas()

fig = go.Figure(data=[go.Candlestick(x=df_btc['Date'],
              open=df_btc['Open'],
              high=df_btc['High'],
              low=df_btc['Low'],
              close=df_btc['Close'],
              increasing_line_color= 'magenta', decreasing_line_color= 'cyan')])

fig.update_layout(title='Bitcoin', yaxis_title='Price')

fig.show()
```



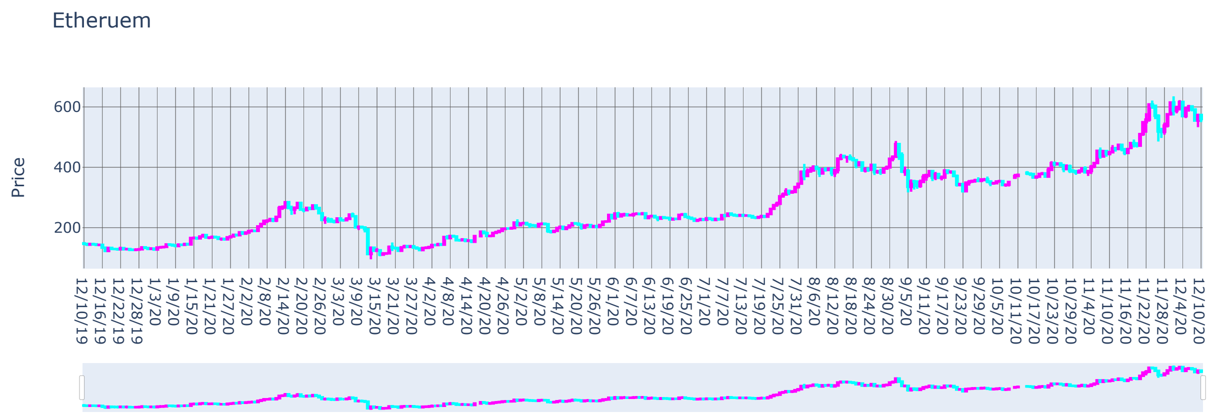
Ethereum

```
#Converting dataframe from Pyspark to Pandas
df_eth = eth.toPandas()

fig = go.Figure(data=[go.Candlestick(x=df_eth['Date'],
              open=df_eth['Open'],
              high=df_eth['High'],
              low=df_eth['Low'],
              close=df_eth['Close'],
              increasing_line_color= 'magenta', decreasing_line_color= 'cyan')])

fig.update_layout(title='Etheruem', yaxis_title='Price')

fig.show()
```



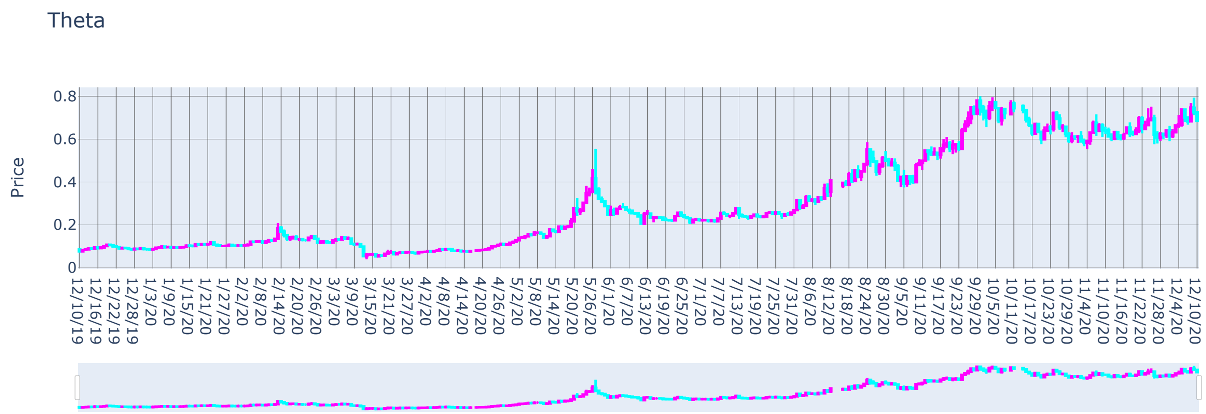
Theta

```
#Converting dataframe from Pyspark to Pandas
df_theta = theta.toPandas()

fig = go.Figure(data=[go.Candlestick(x=df_theta['Date'],
              open=df_theta['Open'],
              high=df_theta['High'],
              low=df_theta['Low'],
              close=df_theta['Close'],
              increasing_line_color= 'magenta', decreasing_line_color= 'cyan')])

fig.update_layout(title='Theta', yaxis_title='Price')

fig.show()
```



Step 6: Options

We chose options 1, 3, & 6 (poster is in the file)

Option 1

Download (from finance.yahoo.com) and test your data on the next month of data for all three currencies.

```
# Load in new data
btcNM = spark.read.csv('/FileStore/tables/BTC_USDNM.csv',inferSchema=True,header=True)
ethNM = spark.read.csv('/FileStore/tables/ETH_USDNM.csv',inferSchema=True,header=True)
thetaNM = spark.read.csv('/FileStore/tables/THETA_USDNM.csv',inferSchema=True,header=True)
```

Pretty much Step 1 again

```
display(btcNM)
btcNM.printSchema()
display(ethNM)
ethNM.printSchema()
display(thetaNM)
thetaNM.printSchema()

# Dropping NAs
btcNM = btcNM.na.drop()
ethNM = ethNM.na.drop()
thetaNM = thetaNM.na.drop()

# Asssembler
assembler = VectorAssembler(inputCols = ['Open',
 'High',
 'Low',
 'Close',
 'Volume',
 'OCDiff',
 'Lag1',
 'Lag2',
 'Lag3',
 'Lag4'], outputCol = "features")

testbtcNM = assembler.transform(btcNM)
testethNM = assembler.transform(ethNM)
testthetaNM = assembler.transform(thetaNM)

# Final Test Data Bitcoin
testbtcNM2 = testbtcNM.select("features", "ReturnY", "Date")
display(testbtcNM2)

# Final Test Data Ethereum
testethNM2 = testethNM.select("features", "ReturnY", "Date")
display(testethNM2)

# Final Test Data Theta
testthetaNM2 = testthetaNM.select("features", "ReturnY", "Date")
display(testthetaNM)
```

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	12/11/2020	18263.92969	18268.45313	17619.5332	18058.9043	18058.9043	0.041240152	-0.0000582	-0.011283218	-0.015572121	0.0127
2	12/12/2020	18051.32031	18919.55078	18046.04102	18803.65625	18803.65625	0.018013867	-0.000420135	0.041240152	-0.011283218	-0.015
3	12/13/2020	18806.76563	19381.53516	18734.33203	19142.38281	19142.38281	0.005446643	0.000165333	0.018013867	0.041240152	-0.011
4	12/14/2020	19144.49219	19305.09961	19012.70898	19246.64453	19246.64453	0.008855135	0.000110182	0.005446643	0.018013867	0.0412
5	12/15/2020	19246.91992	19525.00781	19079.8418	19417.07617	19417.07617	0.097518363	0.0000143	0.008855135	0.005446643	0.0180
6	12/16/2020	19418.81836	21458.9082	19298.31641	21310.59766	21310.59766	0.070132451	0.0000897	0.097518363	0.008855135	0.0054

Showing all 31 rows.

```
root
 |-- Date: string (nullable = true)
 |-- Open: double (nullable = true)
```

```
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: double (nullable = true)
 |-- ReturnY: double (nullable = true)
 |-- OCDiff: double (nullable = true)
 |-- Lag1: double (nullable = true)
 |-- Lag2: double (nullable = true)
 |-- Lag3: double (nullable = true)
 |-- Lag4: double (nullable = true)
```

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	12/11/2020	559.679199	560.376709	537.811646	545.797363	11098819124	0.041718705	0.0000011989	-0.024802032	-0.024064691	0.033
2	12/12/2020	545.578552	573.339417	545.245605	568.567322	8534557897	0.037103585	-0.000401062	0.041718705	-0.024802032	-0.02
3	12/13/2020	568.609863	593.78125	564.565979	589.663208	9070377862	-0.006193432	0.0000748158	0.037103585	0.041718705	-0.02
4	12/14/2020	589.782471	590.492981	577.118408	586.011169	8125837102	0.005707096	0.000202215	-0.006193432	0.037103585	0.041
5	12/15/2020	586.02179	596.247742	580.628784	589.355591	9326645840	0.079453277	0.0000181239	0.005707096	-0.006193432	0.037
6	12/16/2020	589.378662	636.64032	582.039124	636.181824	15817248373	0.010511357	0.0000391446	0.079453277	0.005707096	-0.00

Showing all 31 rows.

```
root
 |-- Date: string (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: long (nullable = true)
 |-- ReturnY: double (nullable = true)
 |-- OCDiff: double (nullable = true)
 |-- Lag1: double (nullable = true)
 |-- Lag2: double (nullable = true)
 |-- Lag3: double (nullable = true)
 |-- Lag4: double (nullable = true)
```

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	12/11/2020	0.681331	0.690951	0.638128	0.681598	23804857	0.015604506	-0.005732896	-0.005310571	-0.054768595	-0.029
2	12/12/2020	0.680686	0.716078	0.680255	0.692234	14859558	0.051263012	-0.001339825	0.015604506	-0.005310571	-0.054
3	12/13/2020	0.692528	0.740943	0.683198	0.72772	20230965	0.031257901	0.000424532	0.051263012	0.015604506	-0.005
4	12/14/2020	0.72813	0.766011	0.717345	0.750467	38059917	0.016097976	0.000563086	0.031257901	0.051263012	0.0156
5	12/15/2020	0.750378	0.773199	0.730398	0.762548	24601690	0.115233402	-0.000118607	0.016097976	0.031257901	0.0512
6	12/16/2020	0.762438	0.853716	0.742709	0.850419	60731400	-0.043594981	-0.000144274	0.115233402	0.016097976	0.0312

Showing all 31 rows.

```
root
 |-- Date: string (nullable = true)
 |-- Open: double (nullable = true)
```

```
|-- High: double (nullable = true)
|-- Low: double (nullable = true)
|-- Close: double (nullable = true)
|-- Volume: integer (nullable = true)
|-- ReturnY: double (nullable = true)
|-- OCDiff: double (nullable = true)
|-- Lag1: double (nullable = true)
|-- Lag2: double (nullable = true)
|-- Lag3: double (nullable = true)
|-- Lag4: double (nullable = true)
```

	features	ReturnY	Date
1	{"vectorType": "dense", "length": 10, "values": [18263.92969, 18268.45313, 17619.5332, 18058.9043, 18058.9043, -0.0000582, -0.011283218, -0.015572121, 0.012705073, -0.045357601]}	0.041240152	12/11/2020
2	{"vectorType": "dense", "length": 10, "values": [18051.32031, 18919.55078, 18046.04102, 18803.65625, 18803.65625, -0.000420135, 0.041240152, -0.011283218, -0.015572121, 0.012705073]}	0.018013867	12/12/2020
3	{"vectorType": "dense", "length": 10, "values": [18806.76563, 19381.53516, 18734.33203, 19142.38281, 19142.38281, 0.000165333, 0.018013867, 0.041240152, -0.011283218, -0.015572121]}	0.005446643	12/13/2020
4	{"vectorType": "dense", "length": 10, "values": [19144.49219, 19305.09961, 19012.70898, 19246.64453, 19246.64453, 0.000110182, 0.005446643, 0.018013867, 0.041240152, -0.011283218]}	0.008855135	12/14/2020
5	{"vectorType": "dense", "length": 10, "values": [19246.91992, 19525.00781, 19079.8418, 19417.07617, 19417.07617, 0.0000143, 0.008855135, 0.005446643, 0.018013867, 0.041240152]}	0.097518363	12/15/2020
6	{"vectorType": "dense", "length": 10, "values": [19418.81836, 21458.9082, 19298.31641, 21310.59766, 21310.59766, 0.0000897, 0.097518363, 0.008855135, 0.005446643, 0.018013867]}	0.070132451	12/16/2020

Showing all 31 rows.

	features	ReturnY	Date
1	{"vectorType": "dense", "length": 10, "values": [559.679199, 560.376709, 537.811646, 545.797363, 11098819124, 0.0000011989, -0.024802032, -0.024064691, 0.033616499, -0.06254294]}	0.041718705	12/11/2020
2	{"vectorType": "dense", "length": 10, "values": [545.578552, 573.339417, 545.245605, 568.567322, 8534557897, -0.000401062, 0.041718705, -0.024802032, -0.024064691, 0.033616499]}	0.037103585	12/12/2020
3	{"vectorType": "dense", "length": 10, "values": [568.609863, 593.78125, 564.565979, 589.663208, 9070377862, 0.0000748158, 0.037103585, 0.041718705, -0.024802032, -0.024064691]}	-0.006193432	12/13/2020
4	{"vectorType": "dense", "length": 10, "values": [589.782471, 590.492981, 577.118408, 586.011169, 8125837102, 0.000202215, -0.006193432, 0.037103585, 0.041718705, -0.024802032]}	0.005707096	12/14/2020
5	{"vectorType": "dense", "length": 10, "values": [586.02179, 596.247742, 580.628784, 589.355591, 9326645840, 0.0000181239, 0.005707096, -0.006193432, 0.037103585, 0.041718705]}	0.079453277	12/15/2020
6	{"vectorType": "dense", "length": 10, "values": [589.378662, 636.64032, 582.039124, 636.181824, 15817248373, 0.0000391446, 0.079453277, 0.005707096, -0.006193432, 0.037103585]}	0.010511357	12/16/2020

Showing all 31 rows.

	Date	Open	High	Low	Close	Volume	ReturnY	OCDiff	Lag1	Lag2	Lag3
1	12/11/2020	0.681331	0.690951	0.638128	0.681598	23804857	0.015604506	-0.005732896	-0.005310571	-0.054768595	-0.029
2	12/12/2020	0.680686	0.716078	0.680255	0.692234	14859558	0.051263012	-0.001339825	0.015604506	-0.005310571	-0.054

3	12/13/2020	0.692528	0.740943	0.683198	0.72772	20230965	0.031257901	0.000424532	0.051263012	0.015604506	-0.005
4	12/14/2020	0.72813	0.766011	0.717345	0.750467	38059917	0.016097976	0.000563086	0.031257901	0.051263012	0.0156
5	12/15/2020	0.750378	0.773199	0.730398	0.762548	24601690	0.115233402	-0.000118607	0.016097976	0.031257901	0.0512
6	12/16/2020	0.762438	0.853716	0.742709	0.850419	60731400	-0.043594981	-0.000144274	0.115233402	0.016097976	0.0312

Showing all 31 rows.

Results

Bitcoin

```
#RMSE
test_resultsbtcNM = cv_modelBtc.transform(testbtcNM2)
rmse_btcNM = evaluator.evaluate(test_resultsbtcNM)
rmse_btcNM

Out[52]: 0.05066156491797875
```

Etereum

```
#RMSE
test_resultsethNM = lrModelEth.evaluate(testethNM2)
rmse_ethNM = test_resultsethNM.rootMeanSquaredError
print(rmse_ethNM)

# Displaying residuals
display(test_resultsethNM.residuals)

0.0669245273541424
```

	residuals
1	0.0371333468164557
2	0.0325182268164557
3	-0.010778790183544301
4	0.0011217378164556993
5	0.0748679188164557
6	0.0059259988164557

Showing all 31 rows.

Theta

```
# Finding rmse
test_resultsthetaNM = rf_modelTheta.transform(testthetaNM2)
rmse_thetaNM = evaluator.evaluate(test_resultsthetaNM)
rmse_thetaNM
```

```
Out[54]: 0.10895255591091357
```

Option 3

Find a single algorithm that works well on all of the 3 crypto-currencies and see how it works on other crypto-currencies (download another crypto currency and predict it for 30 days). You may choose your timeframes for both train and test.

Loading in and Transforming Dogecoin Dataset

```
#Loading in dataset
doge = spark.read.csv('/FileStore/tables/DOGE_USD.csv',inferSchema=True,header=True)
#display(doge)
doge.printSchema()
```

```
root
 |-- Date: string (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: long (nullable = true)
 |-- ReturnY: double (nullable = true)
 |-- OCDIff: double (nullable = true)
 |-- Lag1: double (nullable = true)
 |-- Lag2: double (nullable = true)
 |-- Lag3: double (nullable = true)
 |-- Lag4: double (nullable = true)
```

```
# Dropping na
doge2=doge.na.drop()

# Changing the date datatype to date
from pyspark.sql.functions import to_date
doge2 = doge2.withColumn("Date", to_date(doge2['Date'], "M/d/yy"))
doge2.printSchema()

# Splitting into traingin and testing dataset
train_datadoge2 = doge2.limit(315)
test_datadoge2 = doge2.orderBy(doge2["Date"].desc()).limit(30).orderBy("Date")
```

```
root
 |-- Date: date (nullable = true)
 |-- Open: double (nullable = true)
```

```
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Volume: long (nullable = true)
 |-- ReturnY: double (nullable = true)
 |-- OCDIff: double (nullable = true)
 |-- Lag1: double (nullable = true)
 |-- Lag2: double (nullable = true)
 |-- Lag3: double (nullable = true)
 |-- Lag4: double (nullable = true)
```

```
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols = ['Open',
 'High',
 'Low',
 'Close',
 'Volume',
 'OCDIff',
 'Lag1',
 'Lag2',
 'Lag3',
 'Lag4'], outputCol = "features")

outputTraindoge2 = assembler.transform(train_datadoge2)
outputTestdoge2 = assembler.transform(test_datadoge2)

#Final Train DOGE
trainDoge = outputTraindoge2.select("features", "ReturnY", "Date")
#Final Test DOGE
testDoge = outputTestdoge2.select("features", "ReturnY", "Date")

# Example
display(trainDoge)
```

	features	ReturnY	Date
1	{"vectorType": "dense", "length": 10, "values": [0.001986, 0.002044, 0.00196, 0.002012, 116283369, 0.005538771, 0.007511267, 0.008585859, 0.061662198, -0.007978723]}	0.004473161	2020-04-08
2	{"vectorType": "dense", "length": 10, "values": [0.002013, 0.002033, 0.001985, 0.002021, 109954538, -0.000496771, 0.004473161, 0.007511267, 0.008585859, 0.061662198]}	-0.026224641	2020-04-09
3	{"vectorType": "dense", "length": 10, "values": [0.002021, 0.002023, 0.001891, 0.001968, 122108790, 0, -0.026224641, 0.004473161, 0.007511267, 0.008585859]}	0.011178862	2020-04-10
4	{"vectorType": "dense", "length": 10, "values": [0.001968, 0.002004, 0.001941, 0.00199, 161367396, 0, 0.011178862, -0.026224641, 0.004473161, 0.007511267]}	0	2020-04-11
5	{"vectorType": "dense", "length": 10, "values": [0.001989, 0.002009, 0.001951, 0.00199, 169892709, 0.000502765, 0, 0.011178862, -0.026224641, 0.004473161]}	-0.015577889	2020-04-12
6	{"vectorType": "dense", "length": 10, "values": [0.00199, 0.00199, 0.001894, 0.001959, 167357650, 0, -0.015577889, 0, 0.011178862, -0.026224641]}	0.002552323	2020-04-13

Showing all 315 rows.

```
from pyspark.ml.regression import LinearRegression

dogeLR = LinearRegression(labelCol = "ReturnY", featuresCol = "features", solver = 'l-bfgs', regParam=0.7)

lrModelDoge = dogeLR.fit(trainDoge)
# printing coeff and intercept
print("Coefficients: %s" % str(lrModelDoge.coefficients))
print("Intercept: %s" % str(lrModelDoge.intercept))

# more details about model
trainingSummary = lrModelDoge.summary
trainingSummary.residuals.show()
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("r2: %f" % trainingSummary.r2)
```

```
Coefficients: [-0.00872904356885888,-0.002856347070925378,-0.023427173666502356,0.006829590126776283,-6.325500156720279e-13,0.1331572256838448,0.018137690620592683,0.002616455060123739,0.
005381066165874766,-0.007512860778298973]
Intercept: 0.019318547828731315
+--------------------+
|           residuals|
+--------------------+
|-0.01600446173661...|
|-0.04503513684883108|
|-0.00752012963351...|
|-0.01926347316072...|
|-0.03465522821098...|
|-0.01658094525903293|
|-0.03135388492551737|
| 0.02643488489001497|
|-0.00142392270021...|
|0.007583026405368093|
|-0.00127758509843...|
|  0.1051202702880282|
|-0.04387743945095175|
|-0.02563065645199...|
| 0.05660322632918762|
```

```
#RMSE
testResultsDoge = lrModelDoge.evaluate(testDoge)
rmseDoge = testResultsDoge.rootMeanSquaredError
rmseDoge
```

```
Out[69]: 0.058801133560562255
```

RMSE = .05880

If we were to guess the reason why this model's RMSE is higher than the other cryptocurrencies is because of Dogecoin's extreme volatility.

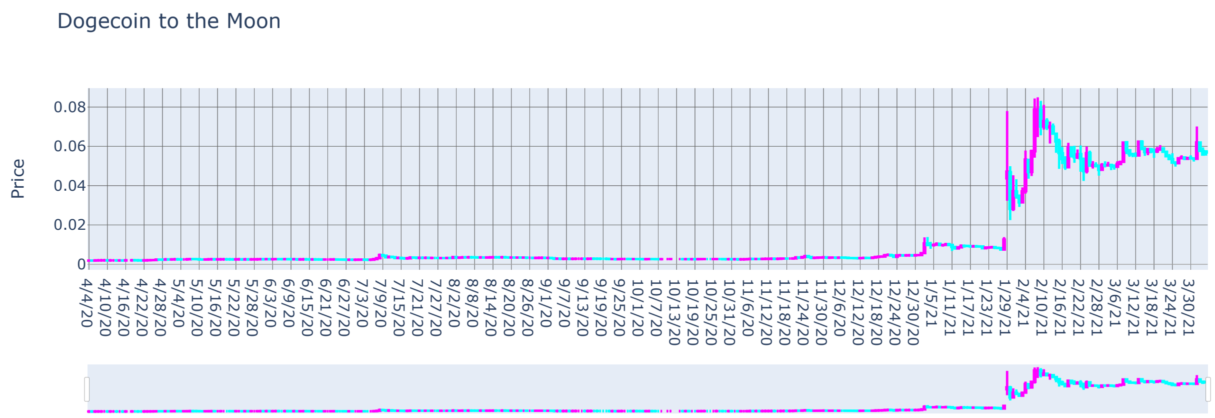
Doge Candlestock Graph

```
#Converting dataframe from Pyspark to Pandas
df_doge = doge.toPandas()

fig = go.Figure(data=[go.Candlestick(x=df_doge['Date'],
              open=df_doge['Open'],
              high=df_doge['High'],
              low=df_doge['Low'],
              close=df_doge['Close'],
              increasing_line_color= 'magenta', decreasing_line_color= 'cyan')])

fig.update_layout(title='Dogecoin to the Moon', yaxis_title='Price')

fig.show()
```



Predicting Next 30 Days

```
+--------------------+------------+--------------------+
|          prediction|     ReturnY|            features|
+--------------------+------------+--------------------+
|0.017156772645292504| 0.027716745|[0.050028,0.05085...|
|0.017384787570601724| 0.022320728|[0.049601,0.05239...|
|0.017819175083312714| 0.188538429|[0.05098,0.052141...|
|0.019341536811785555|-0.063778269|[0.052123,0.06194...|
| 0.01503765038198189|-0.034380496|[0.062104,0.06226...|
| 0.01691020948937858|-0.001535605|[0.057964,0.05861...|
|0.015179627271763176|-0.010175614|[0.055977,0.05669...|
|0.017052440941745736| 0.127969792|[0.055921,0.05698...|
| 0.01915274320746398| -0.06150692|[0.055353,0.06243...|
|0.015159019445515374|-0.025669033|[0.062384,0.06305...|
```

```
|0.016912732960521242| 0.026608044|[0.058531,0.05968...|
|0.015998245081072497|-0.016277919|[0.057086,0.05892...|
|0.017255898986918258|-0.004683191|[0.05861,0.058866...|
|0.017433904559166824| 0.016468292|[0.05764,0.058828...|
| 0.01712202380007528| 0.010852421|[0.057377,0.05972...|
|0.017403371946624505|-0.029290548|[0.058315,0.06063...|
|0.016814181734545592|-0.039696684|[0.05897,0.059521...|
```