# UofM Fintech Group Project 2

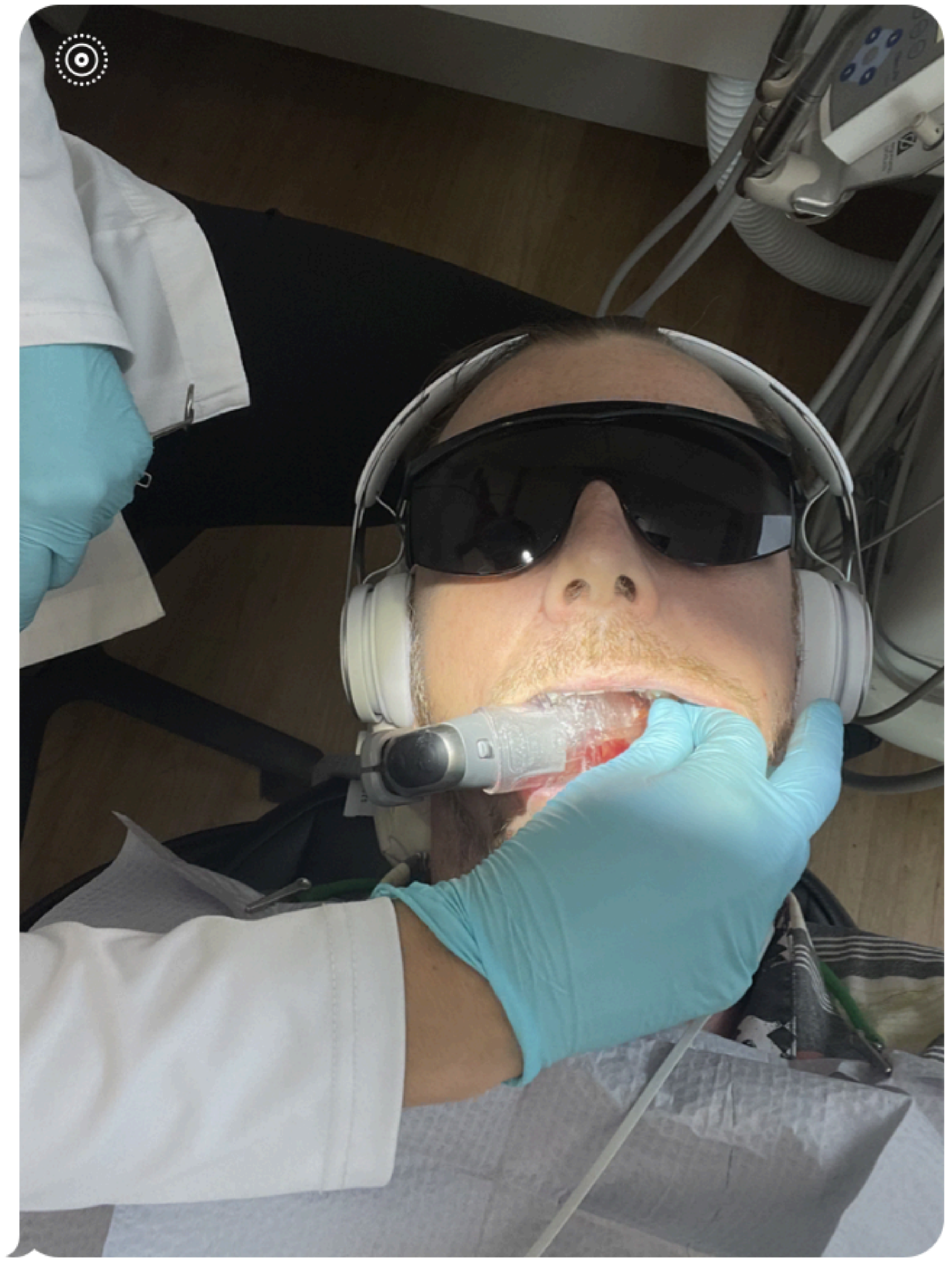
## LSTM, GRU with Ethereum price prediction and forecast.

we will see if you gonna present

or how you going to present lol

kyle plathe

I'll do my best



KP

by Kyle Plathe, Richard Melvin & Meek Msaki, 2022

# Table of Contents

# Hypothesis

# Hypothesis

*Our motivation and summary…*

- We use machine learning tools to predict the price of ethereum from historical data, economic indicators, and community sentiment on ethereum specifically from twitter.
- We test this hypothesis by building LSTM and GRU models

# 1. Data Collection

Describing what kinds of data we needed and where we find it.

# Our Sources

- Rich -Twint Protocol - Collecting tweet data
- Meek - Owlracle API - Getting Gas price history
- Kyle - Kaggle - ETH to USD Historical Data
- Kyle - FRED - Personal Savings Percentage
- Kyle - Market Watch - S&P 500 Historical Data

# 2. Data Cleanup

Preparing our data for our models

# Twint Protocol

Richard Melvin

- **Purpose:** Obtain data for testing the hypothesis that community sentiment predicts Ethereum price.
- **Method:** "Scrape" Twitter for tweets containing keyword, "Ethereum".
  - Twint project: https://github.com/twintproject/twint
- **Result:** 4.3 million tweets were collected (Nov 2017 to Dec 2020)

# Twint Protocol Flow Chart

## Twint
Install from github
Requires: nest_asyncio

```
git clone --depth=1
 https://github.com/twintproject/twint.git
cd twint
pip install . -r requirements.txt
pip install nest_asyncio
```

## Packages
twint
nest_asyncio
pandas

```
import twint
import nest_asyncio
nest_asyncio.apply()
import pandas as pd
```

## Configure Twint

```
c - twint.Config()
```

## Search
"Ethereum"
Nov '17 to Dec '20
English language
Pandas format

```
c.Search = "Ethereum"
c.Since = '2017-11-01'
c.Until = '2020-12-31'
c.Lang = 'en'
c.Pandas = TRUE
```

## Dataframe
date
tweet

```
def twint_to_pd(columns):
    returntwint.output.pandas.Tweets_df[columns]

twint_to_pd(["date", "tweet"])
```

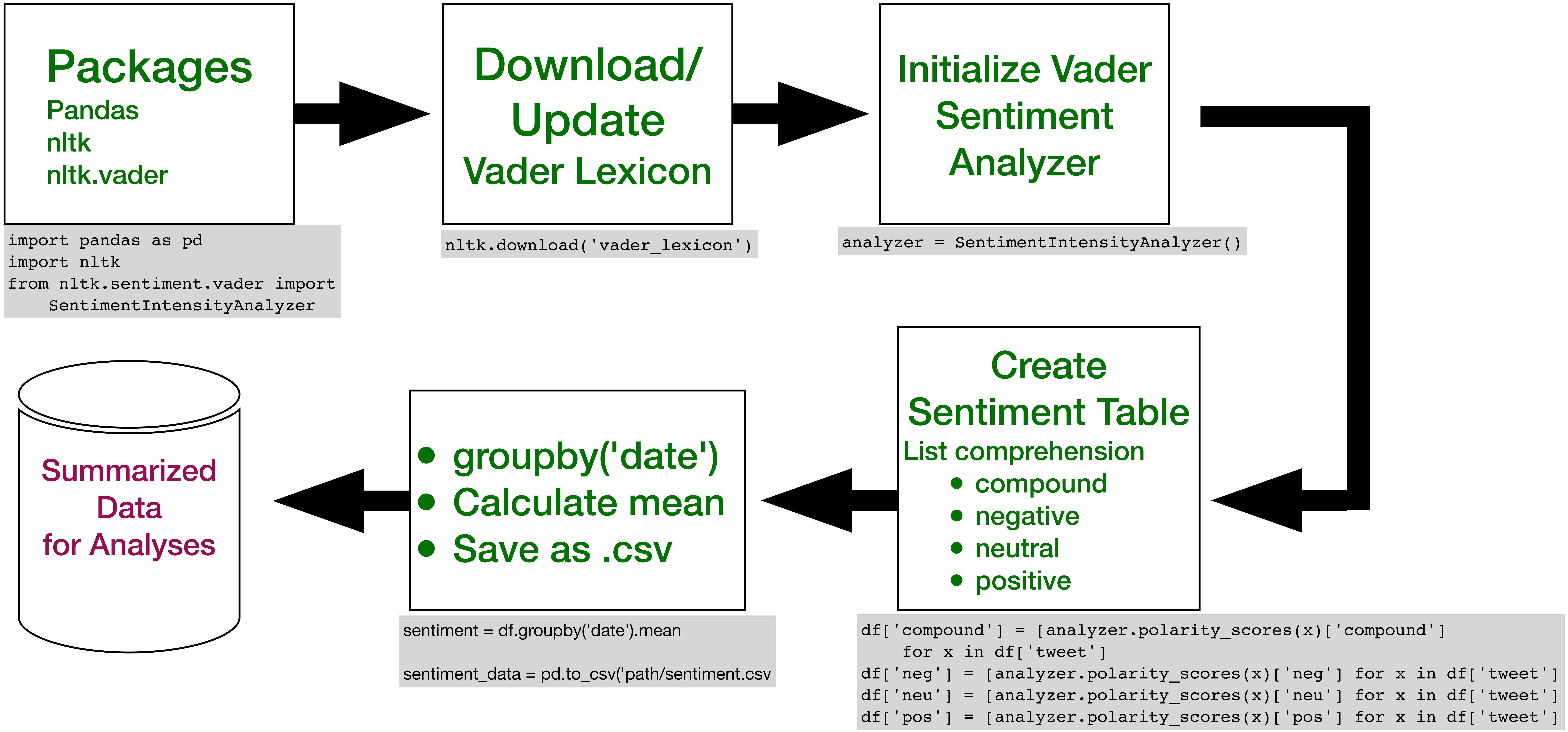
!!!

- Time consuming.
  - connection breaks.
  - restarts.
- Unlimited data
- Iterative function will improve.

# 3. Sentiment Analysis

- **Purpose:** Extract sentiment from Tweet texts.
- **Method:** Natural Language Toolkit (*nltk*), ***Vader*** sentiment analysis
- **Result:** Sentiment of 4.3 million tweets was quantified and summarized as the mean for each day.
    - Degree of positive or negative sentiment.

# Vader Protocol Flowchart



# Kyle's Data

- Using CVS files data from Kaggle ETH historical data, data from FRED - Personal Savings Percentage and historical data from the S&P 500 from Market Watch. I concatenated the 3 sets of data.

- One problem was that the Savings Percentage data was missing days from the month.

Using resample "D" for day and ffill for forward fill.

```
# Fill in the missing data days of the month
Personal_Daily_Savings = Personal_Savings_df.resample("D").ffill()
Personal_Daily_Savings.head()
```

| | Personal Saving % |
|---|---|
| **Date** | |
| **2014-07-01** | 7.2 |
| **2014-07-02** | 7.2 |
| **2014-07-03** | 7.2 |
| **2014-07-04** | 7.2 |
| **2014-07-05** | 7.2 |

# 4. LSTM Model - Meek

Using our data with LSTM (Long Short-Term Memory) model

[1]:
```python
# Initial imports
import pandas as pd
import numpy as np
import hvplot.pandas

# Import dependancies
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras import layers
```



our imports

gas price historical data from csv

Eth price historical + other features data from csv

```python
# Set the random seed for reproducibility
# Note: This is for the homework solution, but it is good practice to comment␣
↪this out and run multiple experiments to evaluate your model
from numpy.random import seed
seed(72)
from tensorflow import random
random.set_seed(72)
```

[ ]:
```python
# Upload eth_gas_prices.csv, eth_features, eth_sentiment_history data to Colab
from google.colab import files
csv_file = files.upload()
```

```
<IPython.core.display.HTML object>
```

[2]:
```python
# Read in data
gas_df = pd.read_csv("eth_gas_prices_history.csv", index_col = "date",␣
↪infer_datetime_format=True, parse_dates=True)

# Set index to a datetime format
gas_df.index =  pd.to_datetime(gas_df.index)

# # View first 5 rows of gas_df
gas_df.head()
```

[2]:

| | open | close | low | high | avgGas |
|---|---|---|---|---|---|
| date | | | | | |
| 2022-07-05 | 22.37 | 30.70 | 10.92 | 92.03 | 89140.499588 |
| 2022-07-04 | 30.92 | 21.04 | 8.46 | 159.32 | 89253.609177 |
| 2022-07-03 | 14.06 | 42.13 | 7.20 | 76.55 | 92493.795505 |
| 2022-07-02 | 50.78 | 32.84 | 5.00 | 320.81 | 92896.649370 |
| 2022-07-01 | 12.74 | 47.59 | 7.49 | 103.23 | 91907.403785 |

[3]:
```python
# Load the historical closing prices for Bitcoin
eth_df = pd.read_csv('eth_features.csv', index_col="Date",␣
↪infer_datetime_format=True, parse_dates=True)

# Sort index
eth_df = eth_df.sort_index(ascending=False)
eth_df = pd.DataFrame(eth_df, index = eth_df.index)
eth_df.index = eth_df.index.strftime("%Y-%m-%d")

# Set index to a datetime format
eth_df.index = pd.to_datetime(eth_df.index)

eth_df.head()
```

[3]:
```
                   Open         High          Low        Close       Volume  \
Date
2022-03-25  3109.523438  3182.826660  3097.624268  3122.535889  16882068480
2022-03-24  3031.060791  3118.387695  3012.326660  3108.062012  18070503166
2022-03-23  2973.145020  3036.752197  2933.306641  3031.067139  16008767658
2022-03-22  2897.774170  3040.382813  2892.544434  2973.131104  16830539230
2022-03-21  2860.103271  2954.556641  2838.250488  2897.976563  15206116098

            Personal Saving %  Close/Last
Date
2022-03-25                5.0     4543.06
2022-03-24                5.0     4520.16
2022-03-23                5.0     4456.24
2022-03-22                5.0     4511.61
2022-03-21                5.0     4461.18
```

[4]:
```python
# Read eth_sentiment_history
sentiment_df = pd.read_csv('eth_sentiment_history.csv', index_col="date",␣
 ↪infer_datetime_format=True, parse_dates=True)
sentiment_df = sentiment_df.drop(columns=["Day of Year", "id", "Unnamed: 0"])

sentiment_df
```

[4]:
```
              nlikes  nretweets  compound       neg       neu       pos
date
2020-01-01  3.508475   0.855932  0.097053  0.032445  0.905449  0.062110
2020-01-02  4.185213   0.843540  0.101582  0.052678  0.872564  0.074755
2020-01-03  2.815929   0.701327  0.105520  0.039138  0.892830  0.068028
2020-01-04  2.978692   0.728444  0.086918  0.049989  0.880994  0.069022
2020-01-05  2.992630   1.079687  0.104279  0.045839  0.886108  0.068048
…                …          …         …         …         …         …
2017-12-25  2.453402   1.475008  0.193739  0.021307  0.885513  0.093183
2017-12-26  1.373333   1.226667  0.152753  0.015047  0.915843  0.069110
2017-12-27  3.701015   1.931156  0.149329  0.020644  0.908841  0.070512
2017-12-29  2.524540   0.911043  0.222387  0.018279  0.887650  0.094071
2017-12-30  1.250290   0.890309  0.188613  0.019919  0.896495  0.083584

[1049 rows x 6 columns]
```

[5]:
```python
# Join historical data for ethereum, gas prices and twitter sentiment with␣
 ↪highest gas fees
df = eth_df.join([gas_df, sentiment_df], how="inner")

# Sort index
df = df.sort_index(ascending=False)
df
```

Joining all dataframes

- eth_df
- gas_df
- sentiment_df



[5]:
```
                  Open        High         Low       Close       Volume  \
2020-12-30  731.472839  754.303223  720.988892  751.618958  17294574210
2020-12-29  730.358704  737.952881  692.149414  731.520142  18710683199
2020-12-28  683.205811  745.877747  683.205811  730.397339  24222565862
2020-12-24  584.135620  613.815186  568.596375  611.607178  14317413703
2020-12-23  634.824585  637.122803  560.364258  583.714600  15261413038
…                    …           …           …           …            …
2017-11-15  337.963989  340.911987  329.812988  333.356995    722665984
2017-11-14  316.763000  340.177002  316.763000  337.631012   1069680000
2017-11-13  307.024994  328.415009  307.024994  316.716003   1041889984
2017-11-10  320.670990  324.717987  294.541992  299.252991    885985984
2017-11-09  308.644989  329.451996  307.056000  320.884003    893249984

            Personal Saving %  Close/Last   open  close  low   high  \
2020-12-30               14.0     3732.04  47.15   74.0  1.0  205.0
2020-12-29               14.0     3727.04  68.31   68.2  9.0  178.0
2020-12-28               14.0     3735.36  55.00   45.0  9.0  153.0
2020-12-24               14.0     3703.06  40.00   34.7  1.0  202.0
2020-12-23               14.0     3690.01  33.00   88.0  1.0  579.0
…                           …           …      …      …    …      …
2017-11-15                7.0     2564.62   0.10   20.0  0.1   60.0
2017-11-14                7.0     2578.87  30.00    1.0  0.1   35.0
2017-11-13                7.0     2584.84   4.00    1.0  0.1   50.0
2017-11-10                7.0     2582.30   8.00    0.0  0.0   34.0
2017-11-09                7.0     2584.62   4.00   20.0  0.1   60.0

                  avgGas    nlikes  nretweets  compound       neg       neu  \
2020-12-30  76337.160077  5.751632   1.361131  0.215897  0.023943  0.880747
2020-12-29  72782.380459  5.482190   1.015992  0.181098  0.029384  0.883803
2020-12-28  69025.670836  6.617657   1.088464  0.159243  0.035662  0.876452
2020-12-24  76968.816529  5.794193   1.053289  0.197436  0.030146  0.875286
2020-12-23  71480.326482  6.123096   1.344860  0.203225  0.032519  0.871497
…                      …         …          …         …         …         …
2017-11-15  57898.769541  3.261106   2.833403  0.098748  0.029318  0.903320
2017-11-14  77198.609591  2.898046   2.153568  0.155619  0.030826  0.877397
2017-11-13  68732.650465  3.146906   3.628248  0.103907  0.029216  0.901205
2017-11-10  62459.741726  2.460854   2.444617  0.074516  0.053909  0.869239
2017-11-09  51382.905682  2.580446   2.625297  0.086666  0.041577  0.887715

                 pos
2020-12-30  0.095315
2020-12-29  0.086815
2020-12-28  0.087886
2020-12-24  0.094560
2020-12-23  0.095986
…                  …
2017-11-15  0.067360
```

```
2017-11-14  0.091771
2017-11-13  0.069577
2017-11-10  0.076842
2017-11-09  0.070710

[691 rows x 18 columns]
```

[6]:
```python
# This function accepts the column number for the features (X) and the target␣
↪(y)
# It chunks the data up with a rolling window of Xt-n to predict Xt
# It returns a numpy array of X any y
def window_data(df, window, feature_col_number, target_col_number):
    X = []
    y = []
    for i in range(len(df) - window - 1):
        features = df.iloc[i:(i + window), feature_col_number]
        target = df.iloc[(i + window), target_col_number]
        X.append(features)
        y.append(target)
    return np.array(X), np.array(y).reshape(-1, 1)
```

[7]:
```python
# Predict Closing Prices using a 10 day window of previous closing prices
# Then, experiment with window sizes anywhere from 1 to 10 and see how the␣
↪model performance changes
window_size = 10

# Column index 0 is the 'Open' column for eth_df
# Column index 3 is the `close` column for eth_df
feature_column = 3
target_column = 3
X, y = window_data(df, window_size, feature_column, target_column)
```

selecting column 3 as our column for prediction eth closing prices

[8]:
```python
# Use 70% of the data for training and the remainder for testing
split = int(0.7 * len(X))
X_train = X[: split]
y_train = y[: split]

X_test = X[split:]
y_test = y[split:]
```

[9]:
```python
from sklearn.preprocessing import MinMaxScaler
# Use the MinMaxScaler to scale data between 0 and 1.
scaler = MinMaxScaler()
scaler.fit(X_train)

# Scale thr X_train and X_test sets
X_train = scaler.transform(X_train)
```



**LSTM layers**

```python
X_test = scaler.transform(X_test)

# fit the MinMaxScaler object with the target daya
scaler.fit(y_train)

# Scale the y_train and y_test sets
y_train = scaler.transform(y_train)
y_test = scaler.transform(y_test)
```

[10]:
```python
# Reshape the features for the model
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))
```

[11]:
```python
# Build the LSTM model.
# The return sequences need to be set to True if you are adding additional LSTM␣
↪layers, but
# You don't have to do this for the final layer.
# Note: The dropouts help prevent overfitting
# Note: The input shape is the number of time steps and the number of indicators
# Note: Batching inputs has a different input shape of Samples/TimeSteps/
↪Features

# Define model
lstm_model = Sequential()

# Initial model setup
number_units = 30
dropout_fraction = 0.2

# Layer 1
lstm_model.add(LSTM(
    units=number_units,
    return_sequences=True,
    input_shape=(X_train.shape[1], 1))
    )
lstm_model.add(Dropout(dropout_fraction))

# Layer 2
lstm_model.add(LSTM(units=number_units, return_sequences=True))
lstm_model.add(Dropout(dropout_fraction))

# Layer 3
lstm_model.add(LSTM(units=number_units))
lstm_model.add(Dropout(dropout_fraction))

# Output layer
lstm_model.add(Dense(1))
```

[12]:
```
# Compile the model
lstm_model.compile(optimizer="adam", loss="mean_squared_error")
```

[13]:
```
# Summarize the model
lstm_model.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 10, 30)            3840

 dropout (Dropout)           (None, 10, 30)            0

 lstm_1 (LSTM)               (None, 10, 30)            7320

 dropout_1 (Dropout)         (None, 10, 30)            0

 lstm_2 (LSTM)               (None, 30)                7320

 dropout_2 (Dropout)         (None, 30)                0

 dense (Dense)               (None, 1)                 31

=================================================================
Total params: 18,511
Trainable params: 18,511
Non-trainable params: 0
_________________________________________________________________
```

[14]:
```
# Train the model
# Experiement with the batch size, but a smaller batch size is recommended
lstm_model.fit(X_train, y_train, epochs=10, shuffle=False, batch_size=100,␣
↪verbose=1)
```

```
Epoch 1/10
5/5 [==============================] - 4s 14ms/step - loss: 0.0912
Epoch 2/10
5/5 [==============================] - 0s 14ms/step - loss: 0.0596
Epoch 3/10
5/5 [==============================] - 0s 15ms/step - loss: 0.0368
Epoch 4/10
5/5 [==============================] - 0s 17ms/step - loss: 0.0233
Epoch 5/10
5/5 [==============================] - 0s 14ms/step - loss: 0.0208
Epoch 6/10
5/5 [==============================] - 0s 15ms/step - loss: 0.0164
Epoch 7/10
```



**LSTM model evaluation 91% accuracy**

```
5/5 [==============================] - 0s 23ms/step - loss: 0.0128
Epoch 8/10
5/5 [==============================] - 0s 27ms/step - loss: 0.0116
Epoch 9/10
5/5 [==============================] - 0s 36ms/step - loss: 0.0097
Epoch 10/10
5/5 [==============================] - 0s 26ms/step - loss: 0.0091
```

[14]: <keras.callbacks.History at 0x7fb009447250>

[15]:
```
# Evaluate the model
lstm_model.evaluate(X_test, y_test, verbose = 0)
```

[15]: 0.08968404680490494

[16]:
```
# Make some predictions
predicted = lstm_model.predict(X_test)
```

[17]:
```
# Recover the original prices instead of the scaled version
predicted_prices = scaler.inverse_transform(predicted)
real_prices = scaler.inverse_transform(y_test.reshape(-1, 1))
```

[18]:
```
# Create a DataFrame of Real and Predicted values
eth_price_prediction_lstm_df = pd.DataFrame({
    "real eth price": real_prices.ravel(),
    "predicted eth price": predicted_prices.ravel()
}, index = df.index[-len(real_prices): ])
eth_price_prediction_lstm_df.head()
```

[18]:

| | real eth price | predicted eth price |
|---|---|---|
| 2018-10-08 | 227.981995 | 208.017929 |
| 2018-10-05 | 229.255005 | 208.443634 |
| 2018-10-04 | 227.600998 | 209.198181 |
| 2018-10-03 | 222.218002 | 210.331802 |
| 2018-10-02 | 220.488998 | 211.785172 |

[19]:
```
# Plot the real vs predicted eth values as a line chart
hvplot.extension('bokeh')

eth_price_prediction_lstm_df.hvplot.line(title = "actual eth price vs.␣
↪predicted eth prices with lstm").opts(width = 800)
```

[19]:
```
:NdOverlay   [Variable]
   :Curve   [index]   (value)
```


actual eth price vs. predicted eth prices with lstm
1200
1000
800
600
400
200
1/2018
3/2018
5/2018
7/2018
9/2018
Variable
real eth price
predicted eth price

# 5. GRU Model - KYLE

Using our data with GRU (Gated Recurrent Unit)  model

ETH GRU Prediction

July 7, 2022

[1]:
```
# Import dependencies
import numpy as np
from numpy import newaxis
import pandas as pd
from keras.layers.core import Dense, Activation, Dropout
from keras.layers.recurrent import LSTM, GRU
from keras.models import Sequential
from keras import optimizers
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
```

[2]:
```
# Enter in how many steps we will enroll the network.

Enrol_window = 100

print ('enroll window set to', Enrol_window )
```

```
enroll window set to 100
```

[3]:
```
# Support functions
sc = MinMaxScaler(feature_range=(0,1))
def load_data(datasetname, column, seq_len, normalise_window):
    # A support function to help prepare datasets for an RNN/LSTM/GRU
    data = datasetname.loc[:,column]

    sequence_length = seq_len + 1
    result = []
    for index in range(len(data) - sequence_length):
        result.append(data[index: index + sequence_length])

    if normalise_window:
        #result = sc.fit_transform(result)
        result = normalise_windows(result)

    result = np.array(result)

    #Last 10% is used for validation test, first 90% for training
```



Support functions:

- load_data ( )
- normalise_windows ( )
- predict_sequence_full ( )
- predict_sequences_multiple ( )
- plot_results ( )
- plot_results_multiple ( )

```
    row = round(0.9 * result.shape[0])
    train = result[:int(row), :]
    np.random.shuffle(train)
    x_train = train[:, :-1]
    y_train = train[:, -1]
    x_test = result[int(row):, :-1]
    y_test = result[int(row):, -1]

    x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
    x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

    return [x_train, y_train, x_test, y_test]

def normalise_windows(window_data):
    # A support function to normalize a dataset
    normalised_data = []
    for window in window_data:
        normalised_window = [((float(p) / float(window[0])) - 1) for p in␣
↪window]
        normalised_data.append(normalised_window)
    return normalised_data

def predict_sequence_full(model, data, window_size):
    #Shift the window by 1 new prediction each time, re-run predictions on new␣
↪window
    curr_frame = data[0]
    predicted = []
    for i in range(len(data)):
        predicted.append(model.predict(curr_frame[newaxis,:,:])[0,0])
        curr_frame = curr_frame[1:]
        curr_frame = np.insert(curr_frame, [window_size-1], predicted[-1],␣
↪axis=0)
    return predicted

def predict_sequences_multiple(model, data, window_size, prediction_len):
    #Predict sequence of <prediction_len> steps before shifting prediction run␣
↪forward by <prediction_len> steps
    prediction_seqs = []
    for i in range(int(len(data)/prediction_len)):
        curr_frame = data[i*prediction_len]
        predicted = []
        for j in range(prediction_len):
            predicted.append(model.predict(curr_frame[newaxis,:,:])[0,0])
            curr_frame = curr_frame[1:]
            curr_frame = np.insert(curr_frame, [window_size-1], predicted[-1],␣
↪axis=0)
        prediction_seqs.append(predicted)
```

```
    return prediction_seqs

def plot_results(predicted_data, true_data):
    fig = plt.figure(facecolor='white')
    ax = fig.add_subplot(111)
    ax.plot(true_data, label='True Data')
    plt.plot(predicted_data, label='Prediction')
    plt.legend()
    plt.show()

def plot_results_multiple(predicted_data, true_data, prediction_len):
    fig = plt.figure(facecolor='white')
    ax = fig.add_subplot(111)
    ax.plot(true_data, label='True Data')
    #Pad the list of predictions to shift it in the graph to it's correct start
    for i, data in enumerate(predicted_data):
        padding = [None for p in range(i * prediction_len)]
        plt.plot(padding + data, label='Prediction')
        plt.legend()
    plt.show()

print ('Support functions defined')
```

```
Support functions defined
```

[15]:
```
# Upload CSV file
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

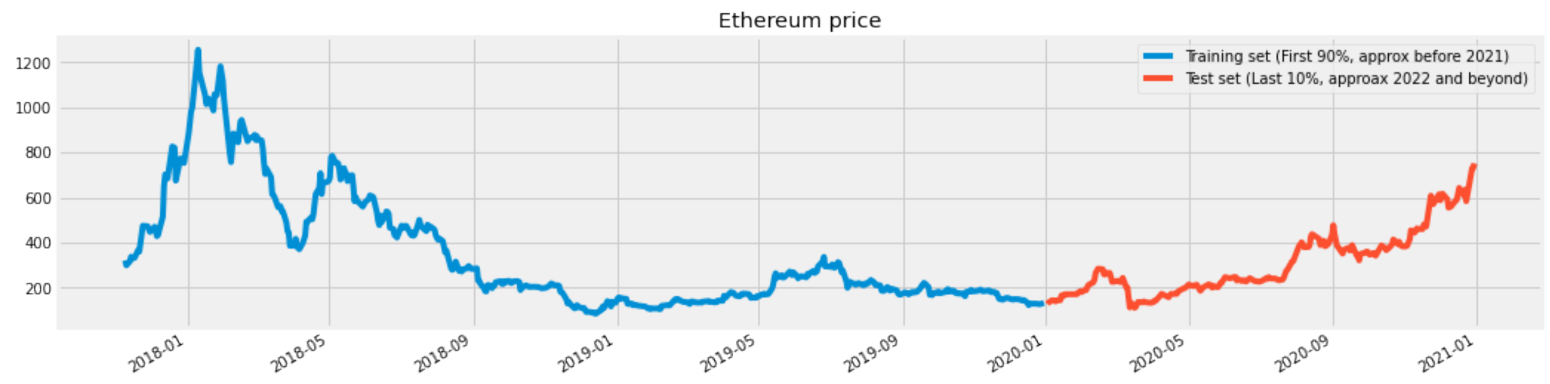
```
Saving eth_all_features.csv to eth_all_features (3).csv
```

[26]:
```
# Read Eth data
dataset = pd.read_csv('eth_all_features.csv', index_col='Unnamed: 0',␣
 ↪parse_dates=True)
dataset.tail()
```

[26]:
```
                  Open        High         Low       Close      Volume  \
2017-11-15  337.963989  340.911987  329.812988  333.356995   722665984
2017-11-14  316.763000  340.177002  316.763000  337.631012  1069680000
2017-11-13  307.024994  328.415009  307.024994  316.716003  1041889984
2017-11-10  320.670990  324.717987  294.541992  299.252991   885985984
2017-11-09  308.644989  329.451996  307.056000  320.884003   893249984

            Personal Saving %  Close/Last  open  close  low  high  \
2017-11-15                7.0     2564.62   0.1   20.0  0.1  60.0
2017-11-14                7.0     2578.87  30.0    1.0  0.1  35.0
2017-11-13                7.0     2584.84   4.0    1.0  0.1  50.0
2017-11-10                7.0     2582.30   8.0    0.0  0.0  34.0
2017-11-09                7.0     2584.62   4.0   20.0  0.1  60.0

                  avgGas    nlikes  nretweets  compound       neg       neu  \
2017-11-15  57898.769541  3.261106   2.833403  0.098748  0.029318  0.903320
2017-11-14  77198.609591  2.898046   2.153568  0.155619  0.030826  0.877397
2017-11-13  68732.650465  3.146906   3.628248  0.103907  0.029216  0.901205
2017-11-10  62459.741726  2.460854   2.444617  0.074516  0.053909  0.869239
2017-11-09  51382.905682  2.580446   2.625297  0.086666  0.041577  0.887715

                 pos
2017-11-15  0.067360
2017-11-14  0.091771
2017-11-13  0.069577
2017-11-10  0.076842
2017-11-09  0.070710
```

[31]:
```
# Prepare the dataset, note that the eth price data will be normalized between␣
 ↪0 and 1
# A label is the thing we're predicting
# A feature is an input variable, in this case ethereum price
# Selected 'Close' (eth price at closing) attribute for prices. Let's see what␣
 ↪it looks like

feature_train, label_train, feature_test, label_test = load_data(dataset,␣
 ↪'Close', Enrol_window, True)

dataset["Close"][:'2019'].plot(figsize=(16,4),legend=True)
dataset["Close"]['2020':].plot(figsize=(16,4),legend=True) # 20% is used for␣
 ↪training data which is approx 2022 data
plt.legend(['Training set (First 80%, approx before 2021)','Test set (Last 20%,␣
 ↪approax 2022 and beyond)'])
plt.title('Ethereum price')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:9: FutureWarning:
Value based partial slicing on non-monotonic DatetimeIndexes with non-existing
keys is deprecated and will raise a KeyError in a future Version.
  if __name__ == '__main__':
```




[32]:
```
# The same LSTM model I would like to test, lets see if the sinus prediction
↪results can be matched

model = Sequential()
model.add(GRU(50, return_sequences=True, input_shape=(feature_train.
↪shape[1],1)))
model.add(Dropout(0.2))
model.add(GRU(100, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1, activation = "linear"))

model.compile(loss='mse', optimizer='adam')

print ('model compiled')

print (model.summary())
```

```
model compiled
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_3 (GRU)                 (None, 100, 50)           7800

 dropout_4 (Dropout)         (None, 100, 50)           0

 gru_4 (GRU)                 (None, 100)               45300

 dropout_5 (Dropout)         (None, 100)               0

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 53,201
Trainable params: 53,201
Non-trainable params: 0
```



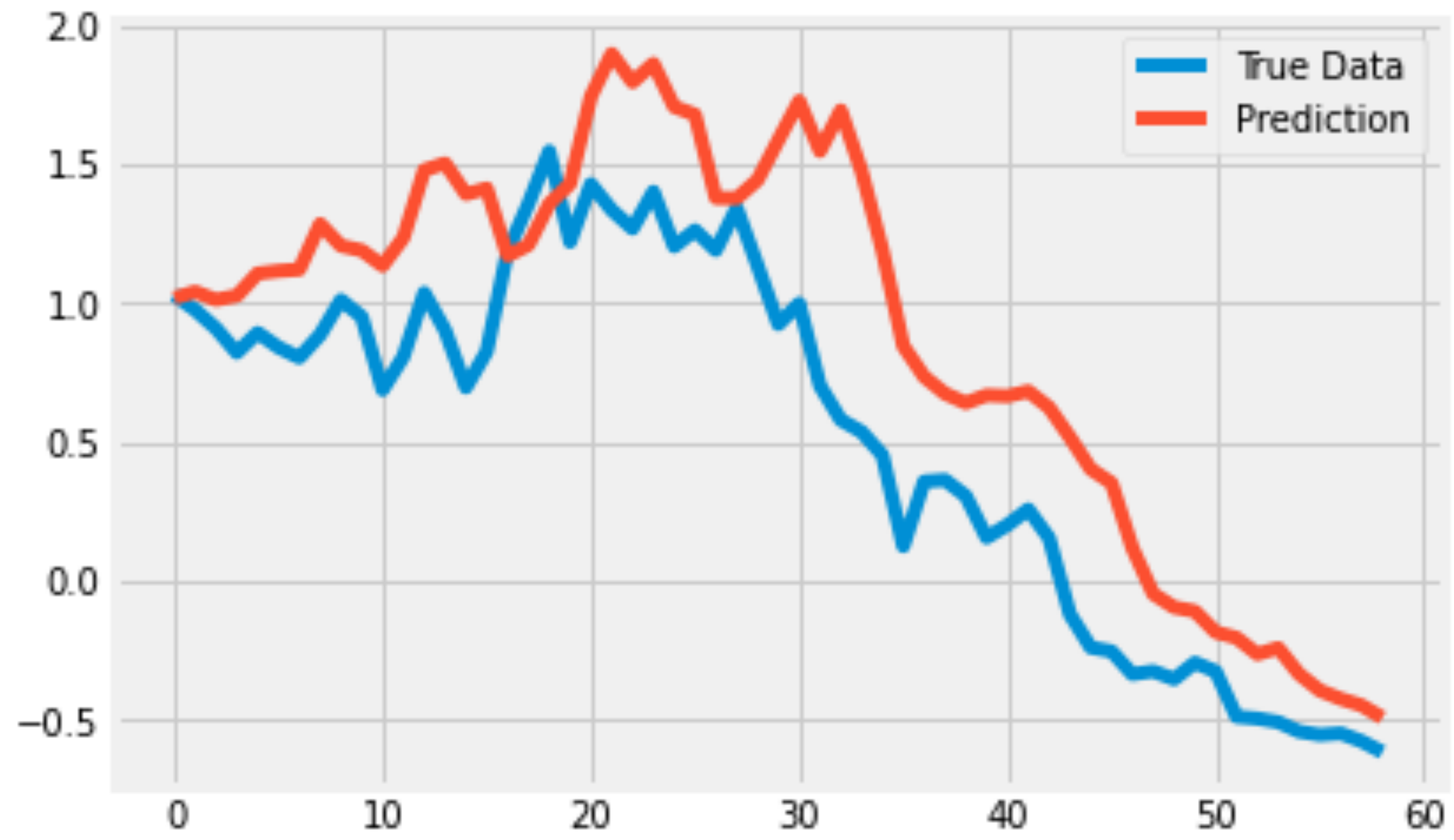
```
_________________________________________________________________
None
```

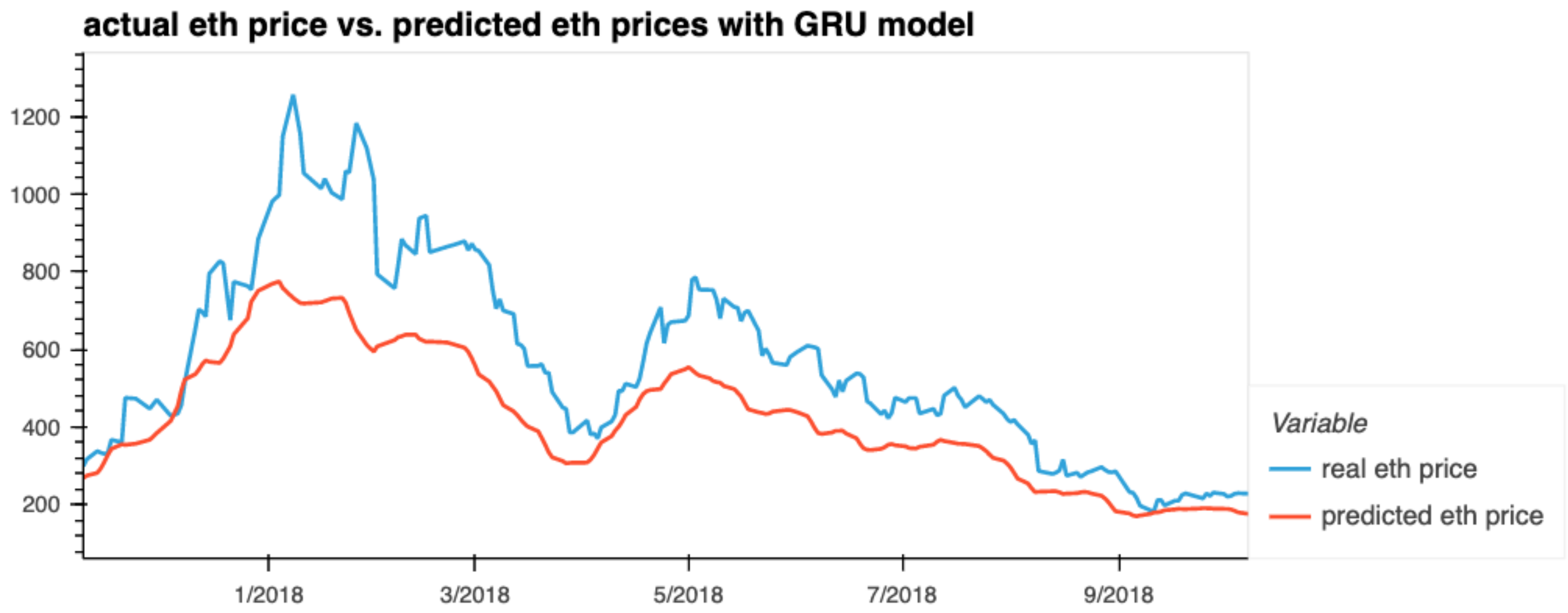
[33]:
```
#Train the model
model.fit(feature_train, label_train, batch_size=512, epochs=5, validation_data
↪= (feature_test, label_test))
```

```
Epoch 1/5
2/2 [==============================] - 8s 937ms/step - loss: 1.0042 - val_loss:
0.3466
Epoch 2/5
2/2 [==============================] - 1s 320ms/step - loss: 0.6146 - val_loss:
0.1761
Epoch 3/5
2/2 [==============================] - 1s 309ms/step - loss: 0.3265 - val_loss:
0.0893
Epoch 4/5
2/2 [==============================] - 1s 314ms/step - loss: 0.1323 - val_loss:
0.1065
Epoch 5/5
2/2 [==============================] - 1s 290ms/step - loss: 0.0755 - val_loss:
0.2019
```

[33]: <keras.callbacks.History at 0x7fd6d570e290>

[34]:
```
# Model and predict the eth price
predicted_eth_price = model.predict(feature_test)
plot_results(predicted_eth_price,label_test)
```






[26]:
```
# Evaluate the model
gru_model.evaluate(X_test, y_test, verbose = 0)
```

[26]: 0.10654260218143463

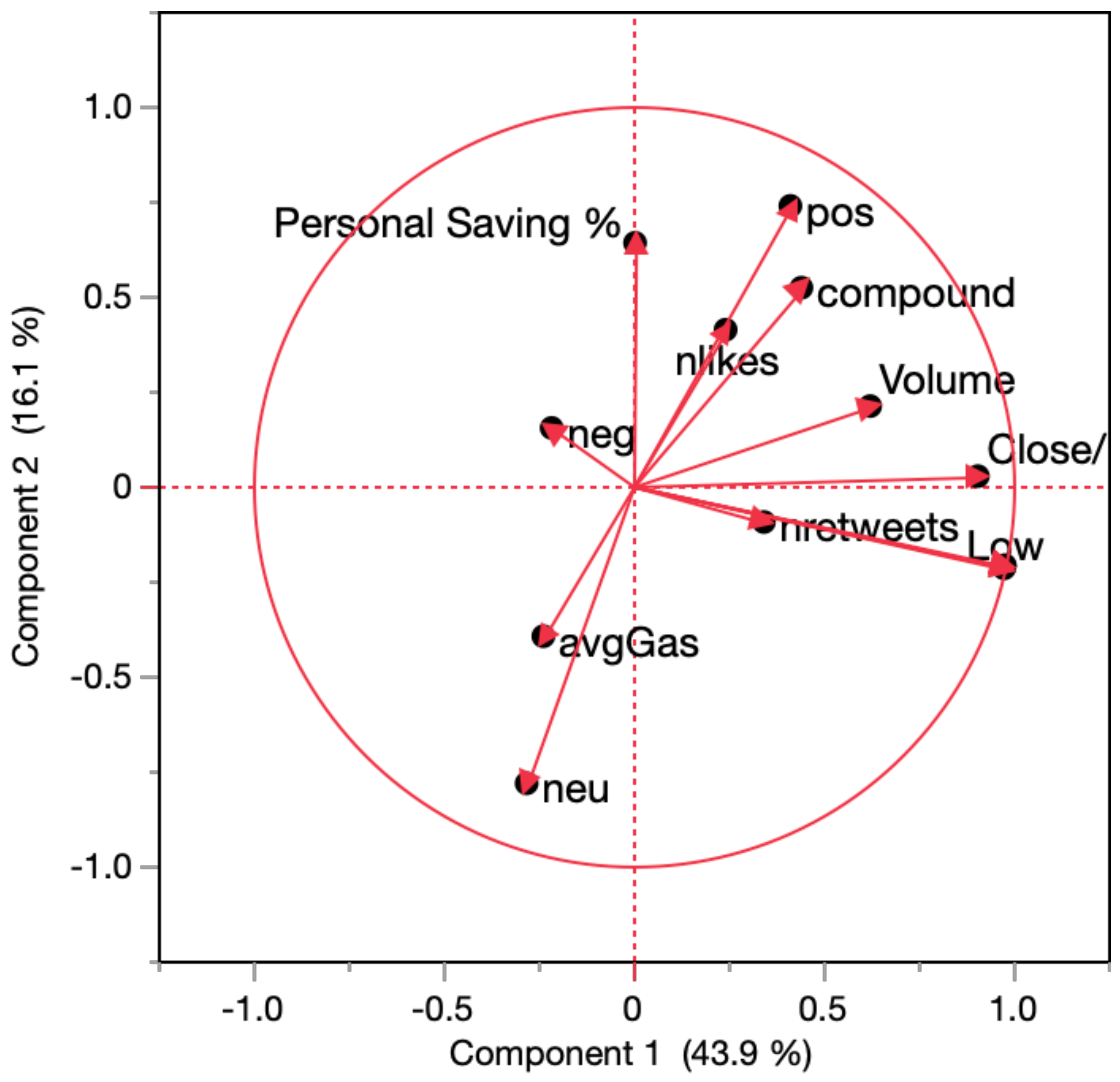
# 6. PCA - Rich

Using PCA analysis on our data

# Principal Components Analysis (PCA)

- Goal: Determine the combination of features that explain most of the variation in a data set.

- In our case PCA may help to understand which, if any, Twitter sentiment values contribute to variation in Ethereum price.




| Prin. Comp. | Eigenvalue | Variation Percent | Variation Cum Percent |
|---|---|---|---|
| 1 | 6.7127 | 43.930 | 43.930 |
| 2 | 2.4661 | 16.139 | 60.069 |
| 3 | 1.8951 | 12.402 | 72.470 |
| 4 | 1.3600 | 8.900 | 81.371 |
| 5 | 1.1768 | 7.701 | 89.072 |
| 6 | 0.6942 | 4.543 | 93.615 |
| 7 | 0.6601 | 4.320 | 97.935 |
| 8 | 0.2237 | 1.464 | 99.398 |
| 9 | 0.0881 | 0.576 | 99.975 |
| 10 | 0.0028 | 0.019 | 99.993 |
| 11 | 0.0008 | 0.005 | 99.998 |
| 12 | 0.0003 | 0.002 | 100.000 |

# Principal Components Analysis (PCA)

**Loading Matrix**

| | Prin1 | Prin2 | Prin3 |
|---|---|---|---|
| Open | 0.97643 | -0.21404 | -0.01543 |
| High | 0.97621 | -0.21732 | -0.00646 |
| Low | 0.97813 | -0.20903 | -0.03619 |
| Close | 0.97767 | -0.21221 | -0.01988 |
| Adj Close | 0.97767 | -0.21221 | -0.01988 |
| Volume | 0.62399 | 0.20969 | 0.54697 |
| avgGas | -0.23572 | -0.39608 | -0.00971 |
| nlikes | 0.24375 | 0.41078 | 0.44950 |
| nretweets | 0.34453 | -0.09548 | -0.09425 |
| compound | 0.44309 | 0.52135 | -0.66219 |
| neg | -0.21386 | 0.15225 | 0.79418 |
| neu | -0.27968 | -0.78327 | -0.04183 |
| pos | 0.41416 | 0.73666 | -0.39819 |
| Personal Saving % | 0.00546 | 0.63940 | 0.22501 |
| Close/Last | 0.90720 | 0.02537 | 0.31987 |

Conclusions:

- Features identified in Principal Component 1 are likely to be better predictors of Ethereum closing price.
- Twitter sentiment likely has little predictive value.

# Principal Components Analysis (PCA)

## SAS code (default to old knowledge... time constraints)

```
Principal Components(
        Y(
                :Open, :High, :Low, :Close, :Adj Close, :Volume, :avgGas, :nlikes,
                :nretweets, :compound, :neg, :neu, :pos, :Personal Saving %, :"Close/Last"n
        ),
        Estimation Method( "Default" ),
        "on Correlations",
        Eigenvalues( 1 ),
        Loading Matrix( 1 ),
        Factor Analysis( "PC", "SMC", 0, "Varimax" ),
        SendToReport(
                Dispatch( {}, "Loading Matrix", OutlineBox, {Select} ),
                Dispatch(
                        {"Factor Analysis: Principal Axis / Varimax"},
                        "Prior Communality Estimates:SMC",
                        OutlineBox,
                        {Close( 0 )}
                ),
                Dispatch(
                        {"Factor Analysis: Principal Axis / Varimax"},
                        "Eigenvalues of the Reduced Correlation Matrix",
                        OutlineBox,
                        {Close( 0 )}
                ),
                Dispatch(
                        {"Factor Analysis: Principal Axis / Varimax"},
                        "Unrotated Factor Loading",
                        OutlineBox,
                        {Close( 0 )}
                )
        )
);
```

# 7. Summary

Summary of our project

# 8. Postmortem

# Postmoterm

*Difficulties that arose*

Our difficulties and how we dealt with them

- Data collection and cleaning, time consuming
- We ask for help - TAs, Instructor & Reached out to api company on telegram

Additional questions that came up that we didn’t answer?

- Undestanding model evaluation

What would we research next if we had more time?

- Crypto exploits data and how it affects eth price movements
- Emoji Sentiment analysis

# 9. Questions

Open floor Q&A with the audience

# thank you,

Special thanks to

- Erik
- Deborah
- Owracle Staff
- US