```
import nltk
from nltk.stem.wordnet import WordNetLemmatizer
#from nltk.stem import WordNetLemmatizer
from nltk.corpus import twitter_samples, stopwords
from nltk.tag import pos_tag
from nltk.tokenize import word_tokenize
from nltk import FreqDist, classify, NaiveBayesClassifier
import numpy as np
import re, string, random
import pandas as pd
import sklearn
from sklearn.model_selection import train_test_split
from google.colab import drive
drive.mount('/content/drive')
import matplotlib.pyplot as plt
import math

from sklearn.ensemble import RandomForestRegressor
%matplotlib inline
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler
#!pip install hvplot
from xgboost import XGBRegressor
from sklearn.linear_model import ElasticNet, LinearRegression
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

```
Mounted at /content/drive
```

```
nltk.download('stopwords')
#nltk.download('wordnet')
nltk.download('averaged_perceptron_tagger')
nltk.download('punkt')
nltk.download('twitter_samples')
nltk.download('all')
```

```
[nltk_data]    |     /root/nltk_data...
[nltk_data]    |   Unzipping models/word2vec_sample.zip.
[nltk_data]    | Downloading package wordnet to /root/nltk_data...
[nltk_data]    | Downloading package wordnet2021 to /root/nltk_data...
[nltk_data]    | Downloading package wordnet2022 to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/wordnet2022.zip.
[nltk_data]    | Downloading package wordnet31 to /root/nltk_data...
[nltk_data]    | Downloading package wordnet_ic to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/wordnet_ic.zip.
[nltk_data]    | Downloading package words to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/words.zip.
[nltk_data]    | Downloading package ycoe to /root/nltk_data...
[nltk_data]    |   Unzipping corpora/ycoe.zip.
[nltk_data]    |
[nltk_data]  Done downloading collection all
True
```

```
def remove_noise(tweet_tokens, stop_words = ()):

    cleaned_tokens = []

    for token, tag in pos_tag(tweet_tokens):
        token = re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+#]|[!*\(\),]|'\
                       '(?:%[0-9a-fA-F][0-9a-fA-F]))+','', token)
        token = re.sub("(@[A-Za-z0-9_]+)","", token)

        if tag.startswith("NN"):
            pos = 'n'
        elif tag.startswith('VB'):
            pos = 'v'
        else:
            pos = 'a'

        lemmatizer = WordNetLemmatizer()
        token = lemmatizer.lemmatize(token, pos)

        if len(token) > 0 and token not in string.punctuation and token.lower() not in stop_words:
            cleaned_tokens.append(token.lower())
    return cleaned_tokens
```

```
Name = "META"
stocks_df = pd.read_csv('/content/drive/MyDrive/stock_price_prediction/'+Name+'(stockdata).csv')
tweets_df = pd.read_csv('/content/drive/MyDrive/stock_price_prediction/'+Name+'2022.csv', dtype="string", encoding='latin1')
```

```
stocks_df
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2020-07-01	228.500000	239.000000	227.559998	237.550003	237.550003	43399700
1	2020-07-02	239.000000	240.000000	232.610001	233.419998	233.419998	30633600
2	2020-07-06	233.759995	240.399994	232.270004	240.279999	240.279999	26206200
3	2020-07-07	239.410004	247.649994	238.820007	240.860001	240.860001	27887800
4	2020-07-08	238.110001	246.990005	236.589996	243.580002	243.580002	29791300
...	...	...	...	...	...	...	...
499	2022-06-24	161.729996	170.250000	161.300003	170.160004	170.160004	68736000
500	2022-06-27	171.320007	171.750000	168.009995	169.490005	169.490005	29174600
501	2022-06-28	169.899994	171.300003	160.610001	160.679993	160.679993	27744500
502	2022-06-29	163.570007	166.330002	160.320007	163.940002	163.940002	28595200
503	2022-06-30	162.149994	165.229996	158.490005	161.250000	161.250000	35250600

504 rows × 7 columns

```
col1 = tweets_df.columns[0]
tweets_df.rename(columns={col1:'Date'}, inplace=True)
```

```
tweets_df['Date'] = pd.to_datetime(tweets_df['Date'])
stocks_df['Date'] = pd.to_datetime(stocks_df['Date'])
```

```
tweets_df
```

	Date	0	1	2	3	
0	2020-07-03	@yvngreaper14 @EASPORTSFIFA If saint max wins ...	@Guillermo76484 @SethAbramson @TheRickWilson M...	OjalÃ¡ algÃºn dÃa @SoyHugoOT2020 se meta en e...	Entre pegar um bebÃª e um anÃ£o no colo, eu pe...	ð ´Es acompaÃ±
1	2020-07-04	Minha meta pra agora Ã© atualizar tudo que tem...	@NA2ARENO Jajaja acÃ¡ Naza adhieren que meta c...	Y por favor no se les olvide que hoy estamos c...	Meta cumplida, hablar con tu ahora, amor impos...	ð Favela ver
2	2020-07-05	@G2BarbeQ el mejor rapero de la historia, porf...	Deria estar orgulloso segun mis amigos pero si...	ReportÃ¡ndome a la #ARMYStreamBattle_D1 con el...	@khale_esi96 Sono ad inizio quinto episodio e ...	gente essa bu
3	2020-07-06	@Damihibibere Muy bueno el hilo y el mensaje q...	@joseq311 @MA_palaC Una nueva meta, tengo que ...	@IndianaOhmz @BenjaminGladst1 @GerbusJames @do...	ð ¤£ð ¤£ð ¤£ð ð ð	@Joaoopospicl
4	2020-07-08	Disjointed veer back to upgrades: Each mech ha...	Mi meta es llegar al punto que me critiquen po...	@Andrrewwwwwwwww Ø§Ù Ù meta analysis Ù Ù Ø¯Ø±...	@qorygore gausah make kalung anti corona,kita ...	meta isso no seu per
...	...	...	...	...	...	
375	2022-06-20	@cabralzinmh @psci_ @Botafogo conta pra mim qu...	@Otherside_Meta loser scam	meu deus...eu deveria estar fazendo meta de le...	@ido_meta Dear sir, many people want to buy wa...	@LaPosta_Ecu La s
376	2022-06-24	@st3f4ny_ MINHA META SER ASSIM	@Jaci_DAO we still on degen mint meta ser @Min...	@robocell @techgirl1908 I hope it does. Thread...	@meta_arabic Ù Ø§Ø²Ù Ù Ù	@pitchulir
377	2022-06-25	@Ziilverk @LuzuGames Espera que no le meta uno...	@DaJenus @thetokensite @banana_ballers @Wonder...	@firel0w Meta	@Guilher23599093 @hiroxzfn @yDiamondd Simplici...	ã ½ã ã ã ®ä¸ ã ®é
378	2022-06-27	@TurtleBoii @wethewolfies @CDAcrypto @Meta_Tsu...	@Bleeding1224 @JeffTay68778958 @AR0D93 @meta_r...	@ErikaLima Chegar na meta e dobrar a meta. Soc...	@thecuteza Yo tranquilo babe, porque se que ah...	#12101: Meta bc
379	2022-06-28	Y los cargos de diplomacias, de seremis, alcal...	@ciaosonororaaa metÃ senza e metÃ con	UPGRADING SOME META GUNS! #WARZONE #WARZONEKIL...	@PhilipR08575643 @Texas_jeep__guy @Disney @Mic...	@Victoria6_s

380 rows × 195 columns

```
#drop days from tweets df that are on weekends and holidays because the stock market is only open on weekdays
tweets_dropped1 = pd.merge(stocks_df["Date"], tweets_df, on="Date")
```

```
#drop days from tweets df that are on weekends and holidays because the stock market is only open on weekdays
tweets_dropped = pd.merge(stocks_df["Date"], tweets_df, on="Date").drop(columns=["Date"])
```

```
tweets_dropped.head()
```

	0	1	2	3	4	
0	@Damihibibere Muy bueno el hilo y el mensaje q...	@joseq311 @MA_palaC Una nueva meta, tengo que ...	@IndianaOhmz @BenjaminGladst1 @GerbusJames @do...	ð ¤£ð ¤£ð ¤ £ð ð ð	@Joaoopospich Meta os dreads	@APL_Fear What ev is the meta lol I have
1	Disjointed veer back to upgrades: Each mech ha...	Mi meta es llegar al punto que me critiquen po...	@Andrrewwwwwwww Ø§Ù Ù meta analysis Ù Ù Ø¯Ø±...	@qorygore gausah make kalung anti corona,kita ...	meta isso no seu perfil e veja quantas pessoas...	@Pedrorrez Meta / es
2	Sweeps x The Meta Team: ASUS 165Hz Gaming Moni...	@Blazt Octane was doing this with the Grau ear...	a meta Ã© chegar em guriri p trocar as tranÃ§a...	Quando o seu coraÃ§Ã£o estiver apertado diante...	@falloutplays @Ms5000Watts We have seen pulse ...	i remember 14 yo r wanting to find E meta
3	@RyanJMonaco1 @Safarooniee Probably best playe...	two ppl down at 5 cyphers against spider see w...	@ElenaGarzaM El riesgo radica en la susceptibi...	sÃ© que quieres que te meta a mi mundo pero no	@ViejitodelHoyo Se que los sueÃ±os no se cump...	Minha meta de vi https://t.co/j8dTPoRH

```
#process tweets into sentiment values for a given day
tweets_tokenized = tweets_dropped.applymap(lambda x: word_tokenize(x) if not pd.isnull(x) else x)
tweets_noiseless = tweets_tokenized.applymap(lambda y: remove_noise(y) if not pd.isnull([y]).any() else y)
```

confused... VERGA, ...

```
#tweets_noiseless = pd.DataFrame(tweets_noiseless)
tweets_noiseless
```

	0	1	2	3	4	5
0	[damihibibere, muy, bueno, el, hilo, y, el, me...	[joseq311, ma_palac, una, nueva, meta, tengo, ...	[indianaohmz, benjamingladst1, gerbusjames, do...	[ð ¤£ð ¤£ð ¤ £ð ð ð]	[joaoopospich, meta, os, dread]	[apl_fear, what, even, be, the, meta, lol, i, ...
1	[disjointed, veer, back, to, upgrade, each,	[mi, meta, e, llegar, al, punto, que, me, crit...	[andrrewwwwwwww, ø§ù ù , meta, analysis, ù ù ,...	[qorygore, gausah, make, kalung, anti,	[meta, isso, no, seu, perfil, e, veja, quantas...	[pedrorrez, meta, ã©, essa]

```
from nltk.tokenize.treebank import TreebankWordDetokenizer
twd =TreebankWordDetokenizer()
```

```
reconstructedSentence = tweets_noiseless.applymap(lambda z: twd.detokenize(z) if not pd.isnull([z]).any() else z)
```

```
reconstructedSentence
```

	0	1	2	3	4	5
0	damihibibere muy bueno el hilo y el mensaje qu...	joseq311 ma_palac una nueva meta tengo que ir ...	indianaohmz benjamingladst1 gerbusjames doctor...	ð ¤£ð ¤£ð ¤ £ð ð ð	joaoopospich meta os dread	apl_fear what even be the meta lol i haven't b...

```
df1 = pd.DataFrame()
```

1 al punto que me

```
import pandas as pd
from nltk.sentiment.vader import SentimentIntensityAnalyzer
sent = SentimentIntensityAnalyzer()
```

sweeps x the quando o seu

```
df1 = reconstructedSentence.applymap(lambda x: round(sent.polarity_scores(x)['compound'], 2) if not pd.isnull([x]).any() else x)
```

gaming monit... diante... pulse rif... and...

```
df1
```

	0	1	2	3	4	5	6	7	8	9	...	184	185	186	187	188	189	19
0	0.34	0.00	0.00	0.00	-0.46	0.42	-0.40	0.00	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	Na
1	-0.44	-0.30	0.00	-0.32	-0.32	0.00	0.00	0.00	0.0	0.4	...	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	0.00	0.48	0.00	0.00	0.42	0.67	0.88	0.00	0.2	0.42	...	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	0.64	0.54	-0.68	-0.30	-0.79	0.00	0.00	-0.25	-0.3	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	0.00	-0.32	-0.30	0.00	0.00	-0.30	0.00	-0.30	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	Na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
256	-0.30	0.00	0.00	-0.36	-0.30	0.00	0.61	0.00	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	Na
257	0.00	0.13	0.00	0.00	0.00	0.00	-0.30	0.00	-0.3	0.2	...	NaN	NaN	NaN	NaN	NaN	NaN	Na
258	0.00	0.00	-0.42	0.00	0.00	0.00	0.00	0.00	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	Na
259	0.00	0.69	0.00	0.00	0.00	0.46	-0.30	-0.30	0.0	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN	Na
260	-0.53	0.00	0.00	0.30	0.00	-0.30	0.00	-0.54	0.0	0.44	...	NaN	NaN	NaN	NaN	NaN	NaN	Na

261 rows × 194 columns

```
sentiment_score = df1.mean(axis=1)
```

robocell techgirl1908

```
df2 = pd.DataFrame(sentiment_score)
df2 = pd.concat([tweets_dropped1['Date'],df2],axis=1)
df2
```

	Date	0
0	2020-07-06	-0.033500
1	2020-07-08	-0.015750
2	2020-07-09	0.147000
3	2020-07-13	-0.187500
4	2020-07-14	-0.061833
...	...	...
256	2022-06-14	-0.004000
257	2022-06-16	0.025000
258	2022-06-24	0.009000
259	2022-06-27	0.001579
260	2022-06-28	0.025263

261 rows × 2 columns

```
prelim = pd.merge(stocks_df.drop(columns=["High", "Low", "Close"]), df2, on='Date', how='left')
prelim = prelim.replace(np.nan,0)
prelim.columns = ['Date', 'Open', 'Adj Close', 'Volume', 'Sentiment_score']
```

```
def condition(x):
  if x > 0:
    return "Positive"
  elif x==0:
    return "neutral"
  else:
    return 'Negative'

prelim['com_score'] = prelim['Sentiment_score'].apply(condition)
prelim['com_score'] = prelim['com_score'].replace({'Positive':1,'neutral':2,'Negative':3})
prelim.head()
```

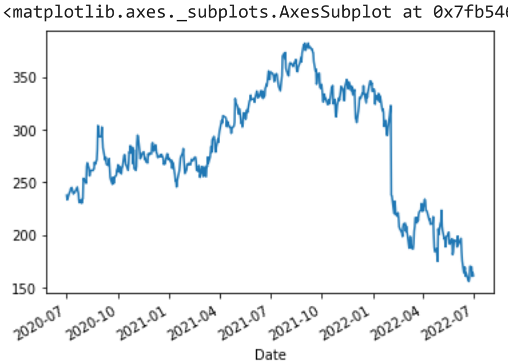
	Date	Open	Adj Close	Volume	Sentiment_score	com_score
0	2020-07-01	228.500000	237.550003	43399700	0.00000	2
1	2020-07-02	239.000000	233.419998	30633600	0.00000	2
2	2020-07-06	233.759995	240.279999	26206200	-0.03350	3
3	2020-07-07	239.410004	240.860001	27887800	0.00000	2
4	2020-07-08	238.110001	243.580002	29791300	-0.01575	3

```
prelim = prelim.set_index('Date')
prelim.head()
```

Date	Open	Adj Close	Volume	Sentiment_score	com_score
2020-07-01	228.500000	237.550003	43399700	0.00000	2
2020-07-02	239.000000	233.419998	30633600	0.00000	2
2020-07-06	233.759995	240.279999	26206200	-0.03350	3
2020-07-07	239.410004	240.860001	27887800	0.00000	2
2020-07-08	238.110001	243.580002	29791300	-0.01575	3

```
prelim['Adj Close'].plot()
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fb54656be20>




```
prelim["Pct_change"] = prelim["Adj Close"].pct_change()
prelim.dropna(inplace = True)
prelim.head()
```

Date	Open	Adj Close	Volume	Sentiment_score	com_score	Pct_change
2020-07-02	239.000000	233.419998	30633600	0.00000	2	-0.017386
2020-07-06	233.759995	240.279999	26206200	-0.03350	3	0.029389
2020-07-07	239.410004	240.860001	27887800	0.00000	2	0.002414
2020-07-08	238.110001	243.580002	29791300	-0.01575	3	0.011293
2020-07-09	245.000000	244.500000	22174900	0.14700	1	0.003777

```
prelim.columns
```

```
Index(['Open', 'Adj Close', 'Volume', 'Sentiment_score', 'com_score',
       'Pct_change'],
      dtype='object')
```

```
prelim = prelim.drop(['Open','Volume'],axis='columns')
```

```
prelim
```

	Adj Close	Sentiment_score	com_score	Pct_change
Date				
2020-07-02	233.419998	0.000000	2	-0.017386
2020-07-06	240.279999	-0.033500	3	0.029389
2020-07-07	240.860001	0.000000	2	0.002414
2020-07-08	243.580002	-0.015750	3	0.011293
2020-07-09	244.500000	0.147000	1	0.003777
...	...	...	...	...
2022-06-24	170.160004	0.009000	1	0.071874
2022-06-27	169.490005	0.001579	1	-0.003937
2022-06-28	160.679993	0.025263	1	-0.051980
2022-06-29	163.940002	0.000000	2	0.020289
2022-06-30	161.250000	0.000000	2	-0.016408

503 rows × 4 columns

```
def window_data(df, window, feature_col_number1, feature_col_number2, feature_col_number3, target_col_number):
    # Create empty lists "X_close", "X_polarity", "X_volume" and y
    X_Sscore = []
    X_pct = []
    X_Cscore = []
    y = []
    for i in range(len(df) - window):

        # Get close, ts_polarity, tw_vol, and target in the loop
        Sscore = df.iloc[i:(i + window), feature_col_number1]
        ts_pct = df.iloc[i:(i + window), feature_col_number2]
        tw_Cscore = df.iloc[i:(i + window), feature_col_number3]
        target = df.iloc[(i + window), target_col_number]

        # Append values in the lists
        X_Sscore.append(Sscore)
        X_pct.append(ts_pct)
        X_Cscore.append(tw_Cscore)
        y.append(target)

    return np.hstack((X_Sscore,X_pct,X_Cscore)), np.array(y).reshape(-1, 1)
```

```
window_size = 4

feature_col_number1 = 0
feature_col_number2 = 1
feature_col_number3 = 2
target_col_number = 0
X, y = window_data(prelim, window_size, feature_col_number1, feature_col_number2, feature_col_number3, target_col_number)
```

```
y
```

```
       [188.070007],
       [184.110001],
       [186.990005],
       [180.949997],
       [174.949997],
       [205.729996],
       [200.470001],
       [211.130005],
       [212.029999],
       [223.410004],
       [208.279999],
       [203.770004],
       [196.210007],
       [197.649994],
       [188.740005],
       [191.240005],
       [198.619995],
       [200.039993],
       [202.619995],
       [192.240005],
       [191.289993],
       [193.539993],
       [196.229996],
       [181.279999],
       [183.830002],
       [191.630005],
       [195.130005],
       [193.639999],
       [188.639999],
       [198.860001],
       [190.779999],
       [194.25    ],
       [195.649994],
       [196.639999],
       [184.      ],
       [175.570007],
       [164.259995],
       [163.729996],
       [169.350006],
       [160.869995],
       [163.740005],
       [157.050003],
       [155.850006],
       [158.75    ],
       [170.160004],
       [169.490005],
       [160.679993],
       [163.940002],
       [161.25    ]])
```

```
X
```

```
array([[233.419998, 240.279999, 240.860001, ...,   3.      ,   2.      ,
          3.      ],
       [240.279999, 240.860001, 243.580002, ...,   2.      ,   3.      ,
          1.      ],
       [240.860001, 243.580002, 244.5     , ...,   3.      ,   1.      ,
          2.      ],
       ...,
       [155.850006, 158.75    , 170.160004, ...,   2.      ,   1.      ,
          1.      ],
       [158.75    , 170.160004, 169.490005, ...,   1.      ,   1.      ,
          1.      ],
       [170.160004, 169.490005, 160.679993, ...,   1.      ,   1.      ,
          2.      ]])
```

```
# Use 90% of the data for training and the remainder for testing
X_split = int(0.9 * len(X))
y_split = int(0.9 * len(y))

X_train = X[: X_split]
X_test = X[X_split:]
y_train = y[: y_split]
y_test = y[y_split:]
```

```
X_train.shape
```

```
(449, 12)
```

```
# Use the MinMaxScaler to scale data between 0 and 1.
x_train_scaler = MinMaxScaler()
```

```
x_test_scaler = MinMaxScaler()
y_train_scaler = MinMaxScaler()
y_test_scaler = MinMaxScaler()

# Fit the scaler for the Training Data
x_train_scaler.fit(X_train)
y_train_scaler.fit(y_train)

# Scale the training data
X_train = x_train_scaler.transform(X_train)
y_train = y_train_scaler.transform(y_train)

# Fit the scaler for the Testing Data
x_test_scaler.fit(X_test)
y_test_scaler.fit(y_test)

# Scale the y_test data
X_test = x_test_scaler.transform(X_test)
y_test = y_test_scaler.transform(y_test)
```

Random Forest Regressor Model

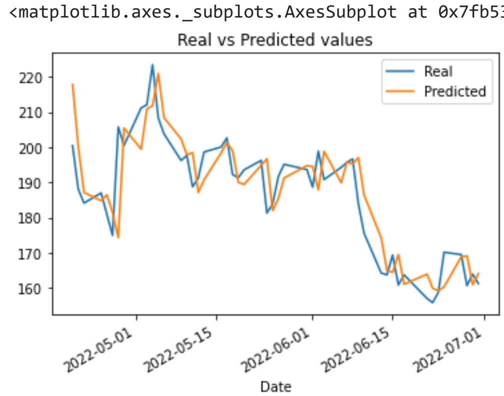
```
model = RandomForestRegressor()
model.fit(X_train, y_train.ravel())
predicted = model.predict(X_test)
#Evaluating the model
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predicted)))
print('R-squared :', metrics.r2_score(y_test, predicted))
predicted_prices = y_test_scaler.inverse_transform(predicted.reshape(-1, 1))
real_prices = y_test_scaler.inverse_transform(y_test.reshape(-1, 1))

stocks = pd.DataFrame({
    "Real": real_prices.ravel(),
    "Predicted": predicted_prices.ravel()
}, index = prelim.index[-len(real_prices): ])

stocks.head()
```

```
Root Mean Squared Error: 0.12486031107297929
R-squared : 0.736888766845539
```

Date	Real	Predicted
2022-04-20	200.419998	217.790349
2022-04-21	188.070007	200.122932
2022-04-22	184.110001	187.114271
2022-04-25	186.990005	184.772214
2022-04-26	180.949997	186.455013

```
stocks.plot(title = "Real vs Predicted values")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb53dbda7f0>
```



XGBOOST Model

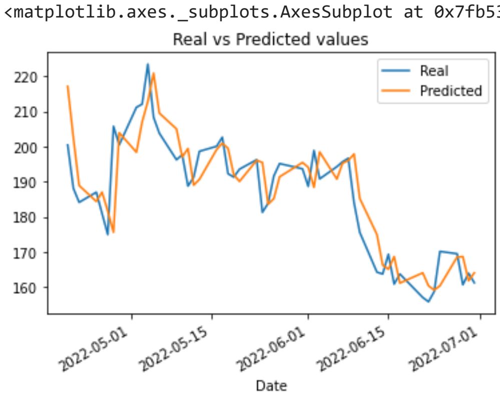
```
# Create the XG Boost regressor instance
model = XGBRegressor()
model.fit(X_train, y_train.ravel())

predicted = model.predict(X_test)

# Evaluating the model
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predicted)))
print('R-squared :', metrics.r2_score(y_test, predicted))

predicted_prices = y_test_scaler.inverse_transform(predicted.reshape(-1, 1))
real_prices = y_test_scaler.inverse_transform(y_test.reshape(-1, 1))

# Create a DataFrame of Real and Predicted values
stocks = pd.DataFrame({
    "Real": real_prices.ravel(),
    "Predicted": predicted_prices.ravel()
}, index = prelim.index[-len(real_prices): ])
stocks.head()
```

```
[12:51:52] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear is now deprecated in fa
Root Mean Squared Error: 0.12453823849465348
R-squared : 0.7382443876907439
```

Date	Real	Predicted
2022-04-20	200.419998	217.084625
2022-04-21	188.070007	202.412445
2022-04-22	184.110001	188.893524
2022-04-25	186.990005	184.360672
2022-04-26	180.949997	187.016449

```
stocks.plot(title = "Real vs Predicted values")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb53d7241c0>
```



Linear regression

```
model = LinearRegression()
model.fit(X_train, y_train.ravel())

predicted = model.predict(X_test)

# Evaluating the model
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predicted)))
print('R-squared :', metrics.r2_score(y_test, predicted))

predicted_prices = y_test_scaler.inverse_transform(predicted.reshape(-1, 1))
real_prices = y_test_scaler.inverse_transform(y_test.reshape(-1, 1))

# Create a DataFrame of Real and Predicted values
stocks = pd.DataFrame({
    "Real": real_prices.ravel(),
    "Predicted": predicted_prices.ravel()
```

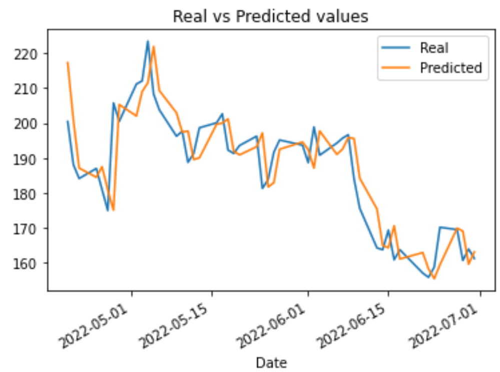
```
}, index = prelim.index[-len(real_prices): ])
stocks.head()
```

```
Root Mean Squared Error: 0.12452163940802427
R-squared : 0.7383141592658282
```

Date	Real	Predicted
2022-04-20	200.419998	217.231648
2022-04-21	188.070007	200.833810
2022-04-22	184.110001	187.097157
2022-04-25	186.990005	184.471166
2022-04-26	180.949997	187.484352

```
stocks.plot(title = "Real vs Predicted values")
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fb53d675cd0>



Deep Learning Model

LSTM RNN model

```
# 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))


# Define the LSTM RNN model.
model = Sequential()

number_units = 9
dropout_fraction = 0.2

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

model.add(Dropout(dropout_fraction))

# Layer 2
# The return_sequences parameter needs to set to True every time we add a new LSTM layer, excluding the final layer.
model.add(LSTM(units=number_units, return_sequences=True))
model.add(Dropout(dropout_fraction))

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

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

```
model.compile(optimizer="adam", loss="mean_squared_error")
```

```
model.summary()
```

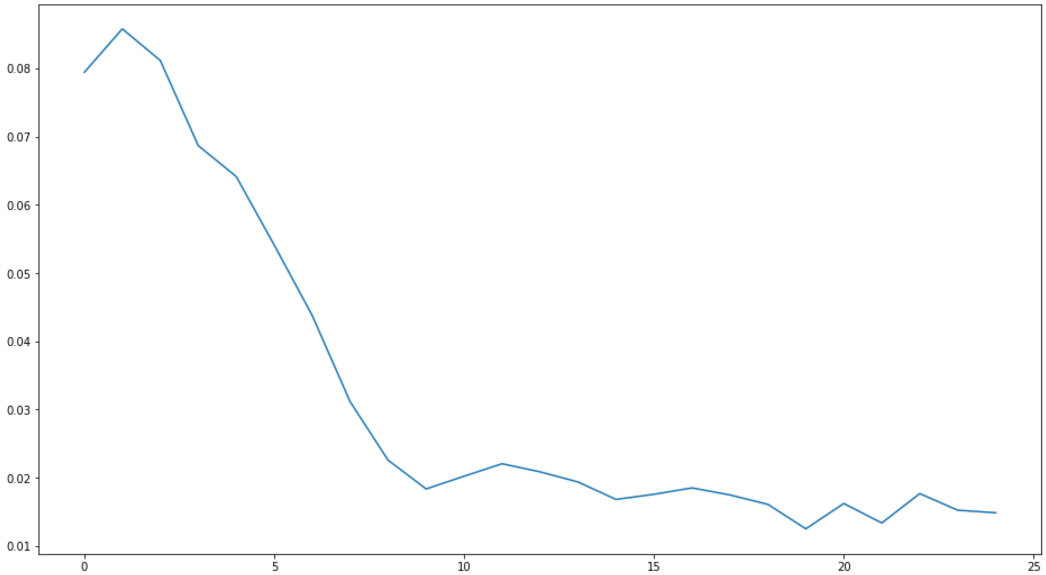
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 12, 9)             396

 dropout (Dropout)           (None, 12, 9)             0

 lstm_1 (LSTM)               (None, 12, 9)             684

 dropout_1 (Dropout)         (None, 12, 9)             0

 lstm_2 (LSTM)               (None, 9)                 684

 dropout_2 (Dropout)         (None, 9)                 0

 dense (Dense)               (None, 1)                 10

=================================================================
Total params: 1,774
Trainable params: 1,774
Non-trainable params: 0
_________________________________________________________________
```

```
history = model.fit(X_train, y_train, epochs=25, shuffle=False, batch_size=8, verbose=1)
```

```
Epoch 1/25
57/57 [==============================] - 10s 8ms/step - loss: 0.0794
Epoch 2/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0858
Epoch 3/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0811
Epoch 4/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0687
Epoch 5/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0641
Epoch 6/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0541
Epoch 7/25
57/57 [==============================] - 0s 8ms/step - loss: 0.0438
Epoch 8/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0311
Epoch 9/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0226
Epoch 10/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0183
Epoch 11/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0202
Epoch 12/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0220
Epoch 13/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0209
Epoch 14/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0194
Epoch 15/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0168
Epoch 16/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0176
Epoch 17/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0185
Epoch 18/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0175
Epoch 19/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0161
Epoch 20/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0125
Epoch 21/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0162
Epoch 22/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0134
Epoch 23/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0177
Epoch 24/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0152
Epoch 25/25
57/57 [==============================] - 0s 7ms/step - loss: 0.0149
```

```
plt.figure(figsize=(16,9))
plt.plot(history.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7fb4bf7c9550>]
```



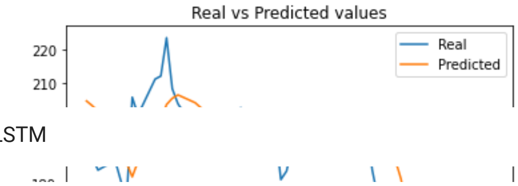

```
model.evaluate(X_test, y_test)
predicted = model.predict(X_test)
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predicted)))
print('R-squared :', metrics.r2_score(y_test, predicted))

predicted_prices = y_test_scaler.inverse_transform(predicted)
real_prices = y_test_scaler.inverse_transform(y_test.reshape(-1, 1))

stocks = pd.DataFrame({
    "Real": real_prices.ravel(),
    "Predicted": predicted_prices.ravel()
}, index = prelim.index[-len(real_prices): ])
stocks.head()
```

```
2/2 [==============================] - 1s 8ms/step - loss: 0.0234
2/2 [==============================] - 1s 12ms/step
Root Mean Squared Error: 0.1530620991216355
R-squared : 0.6046098053688124
```

	Real	Predicted
Date		
2022-04-20	200.419998	204.616928
2022-04-21	188.070007	203.235001
2022-04-22	184.110001	201.507187
2022-04-25	186.990005	197.820145
2022-04-26	180.949997	191.673935

```
stocks.plot(title = "Real vs Predicted values")
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fb4bf77b400>



BiLSTM

```
model = tf.keras.models.Sequential([
   tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(200, return_sequences=True),
                              input_shape=(X_train.shape[1], 1)),
     tf.keras.layers.Dense(20, activation='tanh'),
     tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(150)),
     tf.keras.layers.Dense(20, activation='tanh'),
     tf.keras.layers.Dense(20, activation='tanh'),
     tf.keras.layers.Dropout(0.25),
     tf.keras.layers.Dense(units=1),
 ])
model.compile(optimizer='adam', loss='mse')
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirectiona  (None, 12, 400)          323200
 l)

 dense_1 (Dense)             (None, 12, 20)            8020

 bidirectional_1 (Bidirectio  (None, 300)              205200
 nal)

 dense_2 (Dense)             (None, 20)                6020

 dense_3 (Dense)             (None, 20)                420

 dropout_3 (Dropout)         (None, 20)                0

 dense_4 (Dense)             (None, 1)                 21

=================================================================
Total params: 542,881
Trainable params: 542,881
Non-trainable params: 0
_________________________________________________________________
```

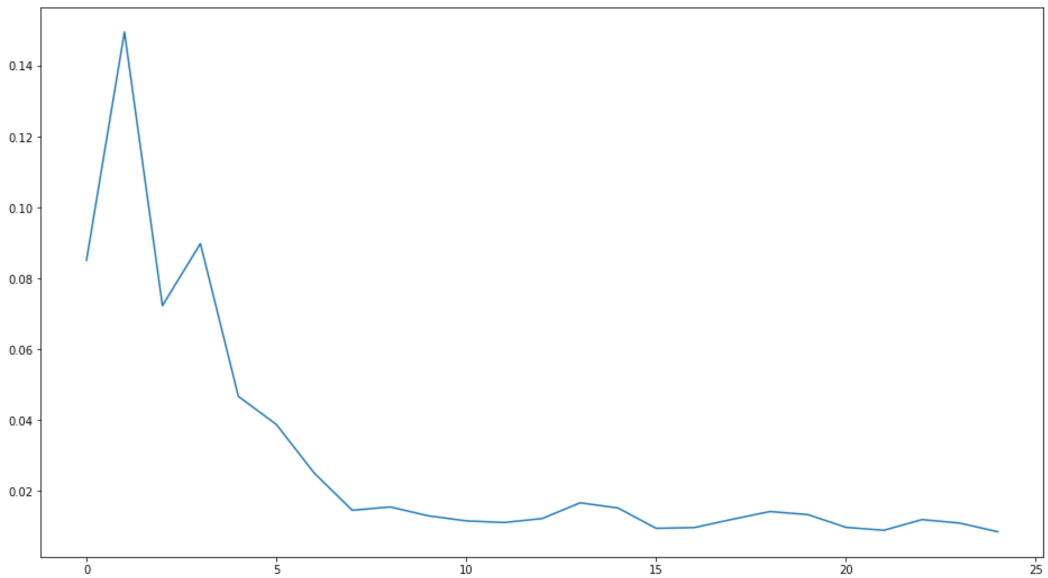
```
history = model.fit(X_train, y_train, epochs=25, shuffle=False, batch_size=32, verbose=1)
```

```
Epoch 1/25
15/15 [==============================] - 5s 12ms/step - loss: 0.0851
Epoch 2/25
15/15 [==============================] - 0s 10ms/step - loss: 0.1495
Epoch 3/25
15/15 [==============================] - 0s 10ms/step - loss: 0.0723
Epoch 4/25
15/15 [==============================] - 0s 11ms/step - loss: 0.0898
Epoch 5/25
15/15 [==============================] - 0s 10ms/step - loss: 0.0467
Epoch 6/25
15/15 [==============================] - 0s 10ms/step - loss: 0.0388
Epoch 7/25
15/15 [==============================] - 0s 11ms/step - loss: 0.0250
Epoch 8/25
15/15 [==============================] - 0s 10ms/step - loss: 0.0145
Epoch 9/25
15/15 [==============================] - 0s 10ms/step - loss: 0.0155
Epoch 10/25
15/15 [==============================] - 0s 10ms/step - loss: 0.0130
Epoch 11/25
15/15 [==============================] - 0s 10ms/step - loss: 0.0116
Epoch 12/25
15/15 [==============================] - 0s 10ms/step - loss: 0.0111
Epoch 13/25
15/15 [==============================] - 0s 11ms/step - loss: 0.0122
Epoch 14/25
15/15 [==============================] - 0s 12ms/step - loss: 0.0167
Epoch 15/25
15/15 [==============================] - 0s 11ms/step - loss: 0.0152
Epoch 16/25
```

```
15/15 [==============================] - 0s 10ms/step - loss: 0.0095
Epoch 17/25
15/15 [==============================] - 0s 11ms/step - loss: 0.0097
Epoch 18/25
15/15 [==============================] - 0s 10ms/step - loss: 0.0120
Epoch 19/25
15/15 [==============================] - 0s 10ms/step - loss: 0.0142
Epoch 20/25
15/15 [==============================] - 0s 11ms/step - loss: 0.0133
Epoch 21/25
15/15 [==============================] - 0s 11ms/step - loss: 0.0097
Epoch 22/25
15/15 [==============================] - 0s 11ms/step - loss: 0.0089
Epoch 23/25
15/15 [==============================] - 0s 10ms/step - loss: 0.0119
Epoch 24/25
15/15 [==============================] - 0s 13ms/step - loss: 0.0109
Epoch 25/25
15/15 [==============================] - 0s 11ms/step - loss: 0.0085
```

```
plt.figure(figsize=(16,9))
plt.plot(history.history['loss'])
```

```
[<matplotlib.lines.Line2D at 0x7fb47daf65b0>]
```



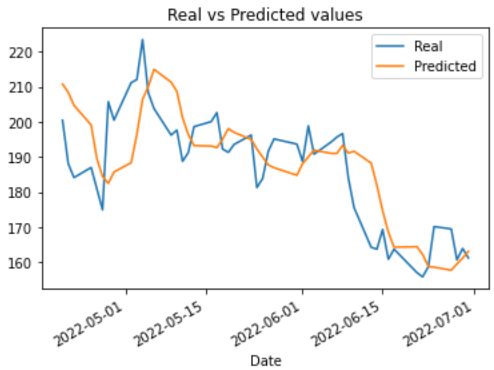

```
model.evaluate(X_test, y_test)
predicted = model.predict(X_test)
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predicted)))
print('R-squared :', metrics.r2_score(y_test, predicted))

predicted_prices = y_test_scaler.inverse_transform(predicted)
real_prices = y_test_scaler.inverse_transform(y_test.reshape(-1, 1))

stocks = pd.DataFrame({
    "Real": real_prices.ravel(),
    "Predicted": predicted_prices.ravel()
}, index = prelim.index[-len(real_prices): ])
stocks.head()
```

```
2/2 [==============================] - 1s 10ms/step - loss: 0.0267
2/2 [==============================] - 1s 9ms/step
Root Mean Squared Error: 0.1633065390553847
R-squared : 0.5499117092144672
```

Real Predicted

```
stocks.plot(title = "Real vs Predicted values")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb47daba1f0>
```



✓ 0s completed at 6:22 PM