Technion Institution

Presented By: Idan Hasdai

Final Project

Machine Learning and Deep Learning stock Analysis

02 April, 2025

About the Project

In this project, I built a couple of models to forecast the Microsoft (MSFT) stock price using historical data. I built machine learning models in order to predict the stock's price. Then I transformed the stock data into overlapping time windows to create training and validation sets, then developed an LSTM-based neural network to capture the patterns in the data and predict future values. Additionally, I used natural language processing to analyze related news headlines, exploring how market sentiment might influence the stock. This approach integrates deep learning and NLP techniques to deliver a comprehensive financial forecasting solution.

Imports

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import yfinance as yf

import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Input, Embedding, Conv1D, Flatten, BatchNormalization, Bidirectional, GRU,SimpleRNN,GlobalAveragePooling1D,Reshape,Lambda
from tensorflow.keras.losses import huber
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.optimizers import RMSprop

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, accuracy_score, classification_report
from sklearn.ensemble import RandomForestClassifier
from statsmodels.tsa.seasonal import seasonal_decompose
```

These are all the imports I made in order to extract, visualize and work on the data (buillding the models and such)

Choosing a stock and starting to work on it

The stock I chose is Microsoft

```
1 # Step 1: Load stock data
2 stock_data = yf.download('MSFT', start = "2018-03-02", end = "2020-07-18")
3 stock_data.reset_index(inplace=True)
```

After extracting the stock I reset the index in order to set it to the index of my choice, just like I did below

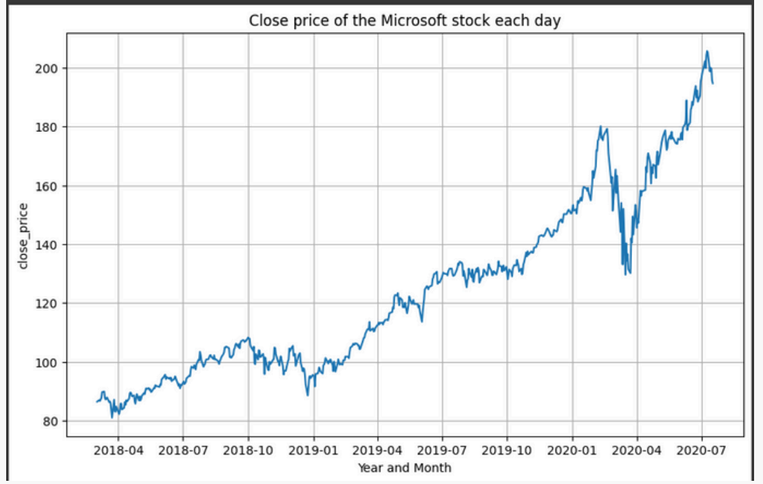
```
1 stock_data.set_index(keys = 'Date', inplace = True)
2 stock_data.head(1)
```

Price	Close	High	Low	Open	Volume
Ticker	MSFT	MSFT	MSFT	MSFT	MSFT
Date					
2018-03-02	86.443405	86.536304	84.408894	85.077775	32830400

Microsoft

Plotting the Close price to see the Trend of the stock

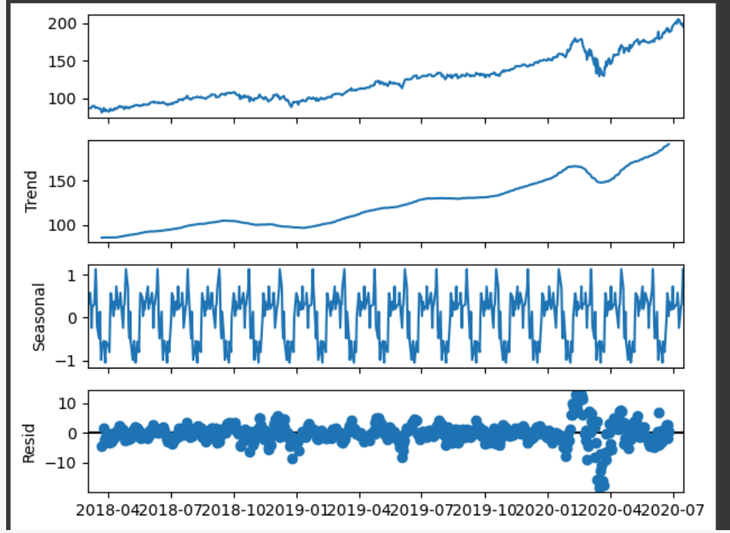
```
1 plt.figure(figsize = (10,6))
2 plt.plot(stock_data.index, stock_data["Close"])
3 plt.title('Close price of the Microsoft stock each day')
4 plt.grid(True)
5 plt.xlabel('Year and Month')
6 plt.ylabel('close_price')
7 plt.show()
```



We can see that overall, the stock has a relatively positive trend except for one point in the middle of 2020 like the entire market at that time

plotting the seasonal decompose of the stock

```
1 result = seasonal_decompose(stock_data['Close'],model = 'additive',period = 31) #period of 31 days
2 result.plot()
3 plt.show()
```



The seasonal Decompose reveals that over the 2018–2020 period, Microsofts' closing price shows a clear upward trend, I have used a period of 31 to show roughly a Month, with the remaining reflecting the day to day market noise.

EDA

Analysis Phase

Visualizing the Five first rows of each column to make sure the Data was loaded correctly

```
1 stock_data.head()
```

Price	Close	High	Low	Open	Volume
Ticker	MSFT	MSFT	MSFT	MSFT	MSFT
Date					
2018-03-02	86.443405	86.536304	84.408894	85.077775	32830400
2018-03-05	86.991493	87.576760	85.709477	85.783791	23901600
2018-03-06	86.694221	87.781149	86.341204	87.641798	22175800
2018-03-07	87.195877	87.270199	85.867408	86.545580	26716100
2018-03-08	87.725418	88.347846	87.112275	87.576775	25887800

Looking at the info of the date set to see the data types of each column and check for nulls

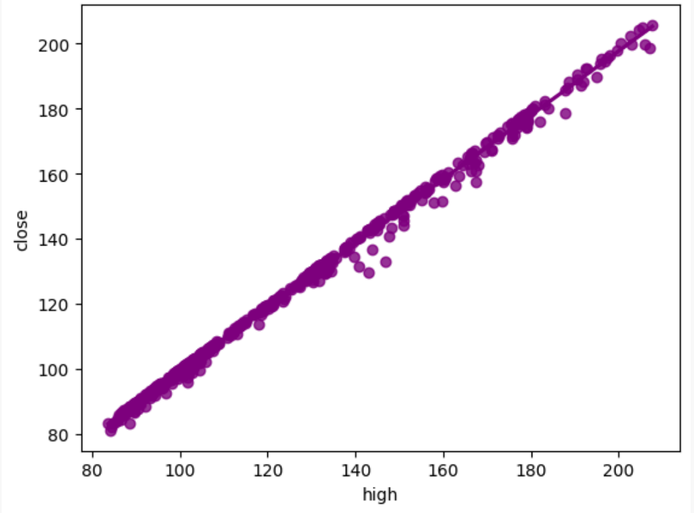
```
1 stock_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 599 entries, 2018-03-02 to 2020-07-17
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   (Close, MSFT)   599 non-null    float64
 1   (High, MSFT)    599 non-null    float64
 2   (Low, MSFT)     599 non-null    float64
 3   (Open, MSFT)    599 non-null    float64
 4   (Volume, MSFT)  599 non-null    int64
dtypes: float64(4), int64(1)
memory usage: 28.1 KB
```

Regplot of the Close and High columns

in this regplot we can see that there is not a big difference between the close and high columns which can indicate that the stock is relatively steady

```
1 sns.regplot(x = 'High', y = 'Close', data = stock_data,color = 'Purple
2 plt.xlabel('high')
3 plt.ylabel('close')
4 plt.show()
```



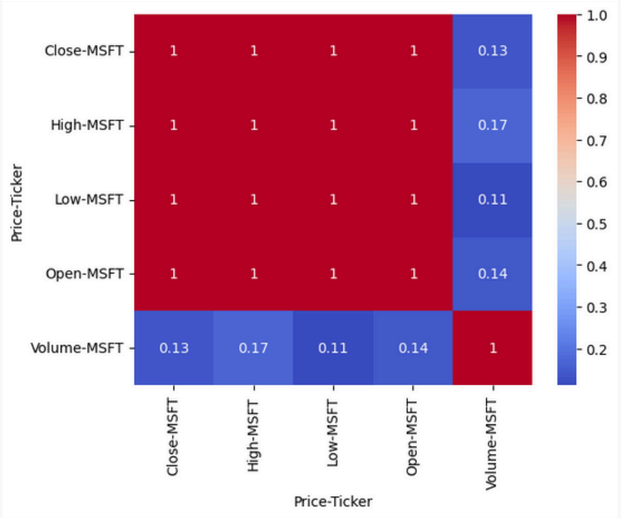
Simplifying the columns names, giving them their First name only

```
1 stock_data.columns = [col[0] for col in stock_data.columns]
```

Correlation Heatmap

I made a heatmap to show the correlation between all the different columns in the stock

```
1 sns.heatmap(stock_data.corr(),annot = True,cmap = 'coolwarm')
2 plt.show()
```



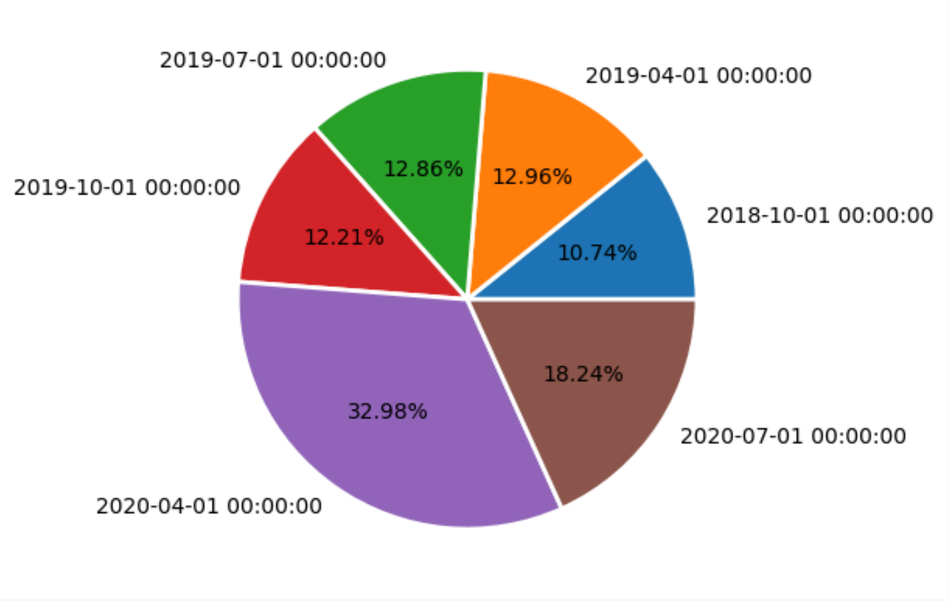
we can see that most of the dataset except for the volume column, is highly correlated to each other, which will probably help us later on with the predictions.

EDA

Pie chart for the volume in each quarter start

This pie chart shows how trading volume is split across these specific periods, with nearly 33% of the stock's volume in only the First quarter of 2020 and the rest distributed among the other quarters in smaller shares. this most likely happened because of the coronavirus and a lot of uncertainty floating across investors.

```
plt.pie(mask['Volume'].squeeze(),
        labels = mask['Volume'].index,
        wedgeprops = {'edgecolor':'white','linewidth':2},
        autopct = '%1.2f%%',
        )
plt.show()
```



Loading the news dataset

I have loaded the financial news dataset and chose one of the three news organization.

I chose reuters as my financial new which I will work with later, for now i have loaded it and will visualize it.

```
!unzip fnews.zip -d fnews
print('-'*20+'\n Upload Completed!!\n'+'-'*20)
```

```
Archive:  fnews.zip
replace fnews/cnbc_headlines.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: fnews/cnbc_headlines.csv
replace fnews/guardian_headlines.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: fnews/guardian_headlines.csv
replace fnews/reuters_headlines.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: y
  inflating: fnews/reuters_headlines.csv
--------------------
 Upload Completed!!
--------------------
```

```
reuters = pd.read_csv('fnews/reuters_headlines.csv')
```

```
reuters.head(3)
```

	Headlines	Time	Description
0	TikTok considers London and other locations fo...	Jul 18 2020	TikTok has been in discussions with the UK gov...
1	Disney cuts ad spending on Facebook amid growi...	Jul 18 2020	Walt Disney has become the latest company to ...
2	Trail of missing Wirecard executive leads to B...	Jul 18 2020	Former Wirecard chief operating officer Jan M...

As we know the dtype of the "Time" column should be day time, and the name should be "Date" to match the stock's datatset. So I changed the name and dtype to pandas day time.

```
[174]  sentences = [len(i) for i in reuters['Headlines']]
```

```
min_len = np.min(sentences)
min_len
```

```
np.int64(20)
```

```
[176]  max_len = np.max(sentences)
       max_len
```

```
np.int64(117)
```

```
[177]  mean_len = np.mean(sentences)
       mean_len
```

```
np.float64(65.2905706438816)
```

```
reuters.rename(columns = {'Time':'Date'},inplace = True)
```

```
reuters.isna().sum()
```

	0
Headlines	0
Date	0
Description	0

dtype: int64

```
reuters['Date'] = pd.to_datetime(reuters['Date'])
```

```
reuters.set_index(keys = 'Date', inplace = True)
reuters.head(1)
```

Date	Headlines	Description
2020-07-18	TikTok considers London and other locations fo...	TikTok has been in discussions with the UK gov...

EDA

1 - Making a Change column

This column represents the change in the Close price for the next day, every day. This column will be useful later on when we will make another column that represent only if the stock went up, down, or stayed the same the following day, we will most likely get a NaN in the final row as we dont have data of the next day

```
#calculating the daily change
stock_data["Change"] = (stock_data["Close"].shift(-1)- stock_data["Close"])
```

2 - Visualizing the new column

```
stock_data.head(1)
```

Date	Close	High	Low	Open	Volume	Change
2018-03-02	86.44339	86.536288	84.408879	85.07776	32830400	0.548088

3 - Making the column that shows if the stock went up down or stayed the same just as I said above, I named the Column label for my prefering.

```
stock_data["label"] =0 #default - no change
stock_data.loc[stock_data["Change"] >0, "label"]= 1
stock_data.loc[stock_data["Change"] <0, "label"]= -1
```

4 - Looking at the new label column

```
stock_data.head(3)
```

Date	Close	High	Low	Open	Volume	Change	label
2018-03-02	86.443390	86.536288	84.408879	85.077760	32830400	0.548088	1
2018-03-05	86.991478	87.576745	85.709462	85.783776	23901600	-0.297256	-1
2018-03-06	86.694221	87.781149	86.341204	87.641798	22175800	0.501656	1

5 - Because We knew we would gete a row with a NaN and a 0 at the end of the dataset, I wanted to visualize it and see what I should do with it, I decided to drop the row with the NaN, as I dont want to fill a real Dataset with incorrect data

```
stock_data[stock_data['label']==0]
```

Date	Close	High	Low	Open	Volume	Change	label
2018-06-18	94.105011	94.338268	92.761451	93.311940	23586000	0.0	0
2019-07-12	131.701767	131.919858	130.857890	131.654370	18936800	0.0	0
2019-12-20	150.259109	151.290049	149.189979	150.201838	53477500	0.0	0
2020-07-17	194.733246	196.806499	193.303072	196.259395	31635300	NaN	0

6

```
stock_data = stock_data.dropna()
```

```
stock_data.isna().sum()
```

	0
Close	0
High	0
Low	0
Open	0
Volume	0
Change	0
label	0

dtype: int64

Machine Learning Models

Prepering the Date for the models

For the features I decided to use all of the columns except for the change and label columns, as the label is my target and the change will give away what I am trying to predict

I did not use the usual train test split method, because I am using a date time type in my features, and it does not fit the train test split of the sk learn library

```
#defining feature and target
X = stock_data.drop(columns = ['Change','label'], axis = 1) #feature
y = stock_data['label'] #target

#spliting it to 80% train 20% test
split_time = int(len(X)*0.8)

#splitting to train and test
X_train,X_test = X[:split_time], X[split_time:]
y_train,y_test = y[:split_time], y[split_time:]
```

I used standard scaler for the featrures, as we have more the one feature and not all the numbers fit each other.

```
#standard sacling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)
```

Machine Learning Models

Building the models

I decided to test Three model, Random Forest, Suppor Vector Machine, and Logistic Regression, to see which of them will work the best, and even see if I will get overfitting in one of them.

```
svm_model = svm.SVC(C = 1,
                    kernel = 'linear',
                    gamma = 'scale',
                    shrinking = True)

rf_model = RandomForestClassifier(min_samples_split = 5,
                                  n_estimators = 200,
                                  max_depth = 5,
                                  max_features = 'sqrt')

lr_model = LogisticRegression(C = 10,
                              max_iter = 100,
                              solver = 'liblinear')
```

Explaining my hyper parameters

Suppor Vector Machine -

- C 1- Controls model flexibility, higher number may causes overfitting
- kernel - I chose a linear Kernel as my data is quite linear so I believed it would work well
- gamma - Adjusts feature influence based on the variance
- shrinking True - Speeds up the model training

Random Forest Classifier -

- min samples split 5 - Minimum samples needed to split a node, also reduces overfitting
- n estimators - Number of trees in the forest, more trees equals more stablity
- max depth - Limits tree growth in order to prevent over fitting
- max features - Uses only some of the features at each step to make the trees more different

Logistic Regresssion -

- C - Higher number mean more flexability and less penalty
- max iter - Maximum optimazation attempts before stopping
- solver - Good for small data sets and binary classification

Comparing the Models

looking at the classification report for all of the models

```
[ ]  1 print(svm_classification)
```

	precision	recall	f1-score	support
-1	0.00	0.00	0.00	50
1	0.58	1.00	0.74	70
accuracy			0.58	120
macro avg	0.29	0.50	0.37	120
weighted avg	0.34	0.58	0.43	120

```
   1 print(lr_classification)
```

	precision	recall	f1-score	support
-1	0.62	0.26	0.37	50
1	0.63	0.89	0.73	70
accuracy			0.62	120
macro avg	0.62	0.57	0.55	120
weighted avg	0.62	0.62	0.58	120

```
[122]  1 print(rf_classification)
```

	precision	recall	f1-score	support
-1	0.47	0.16	0.24	50
1	0.59	0.87	0.71	70
accuracy			0.57	120
macro avg	0.53	0.52	0.47	120
weighted avg	0.54	0.57	0.51	120

Explaining the scores

Precision Score - The precision score shows how many predicted cases were correct, we can asume that the logistic regression worked the best from that aspect out of the 3 models.

Recall Score - shows how many actual cases the model found correctly

F1-Score - Balance between precision and recall, closest to 1 is the best

Overall, Logistic Regression has worked the best

Explaining which model has worked best and why

Logistic Regression - The logistic regression model has worked the best out of the three models, it had the best balance in all three scores.

Support Vector Machine - This model worked well on class 1 but not so much on class -1 which means this model is not so good like the logistic regression

Random Forest Classifier - Just like the svm model this model worked well on class 1 but not so much on class -1

Comparing the Models

Checking the test and train accuracy score to check for overfitting

```
svm_accuracy = accuracy_score(y_test,svm_pred)
rf_accuracy = accuracy_score(y_test,rf_pred)
lr_accuracy = accuracy_score(y_test,lr_pred)

print(f'the logistic regression accuracy score is: {lr_accuracy}')
print(f'the random forest classifier accuracy score is: {rf_accuracy}')
print(f'the svm accuracy score is: {svm_accuracy}')
```

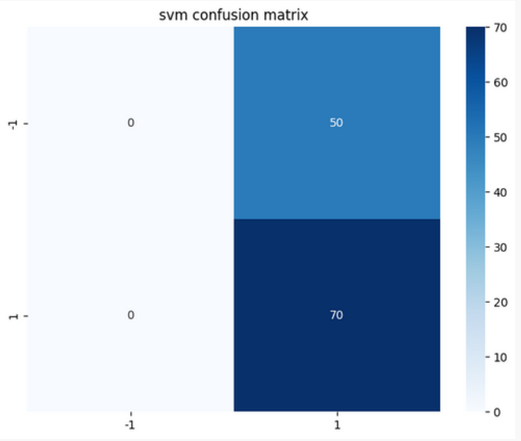
```
the logistic regression accuracy score is: 0.625
the random forest classifier accuracy score is: 0.5583333333333333
the svm accuracy score is: 0.5833333333333334
```

```
[130]
lr_train_accuracy = accuracy_score(y_train,lr_train_pred)
rf_train_accuracy = accuracy_score(y_train,rf_train_pred)
svm_train_accuracy = accuracy_score(y_train,svm_train_pred)
print(f'the logistic regression train accuracy is: {lr_train_accuracy}')
print(f'the random forest classifier train accuracy is: {rf_train_accuracy}')
print(f'the svm train accuracy is: {svm_train_accuracy}')
```

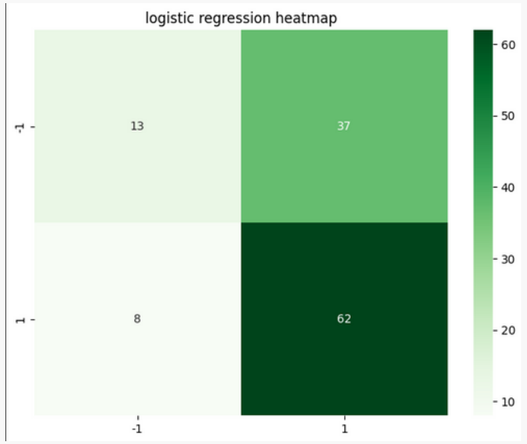
```
the logistic regression train accuracy is: 0.5753138075313807
the random forest classifier train accuracy is: 0.7217573221757322
the svm train accuracy is: 0.5690376569037657
```

Overall we got pretty good results, but we can see the Random Forest model has slight overfitting

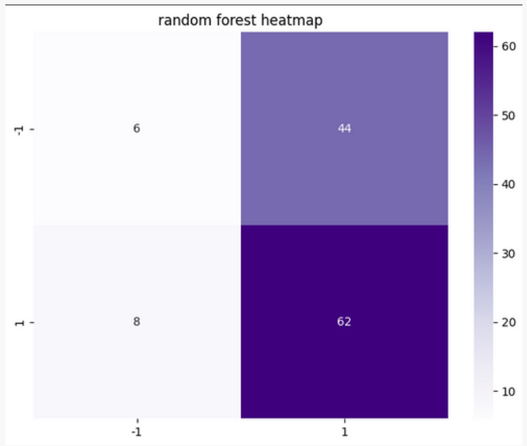
This confusion matrix shows that the SVM model did not predict class -1 correctly at all. It labeled all cases (120 total) as class 1. Therefore, it correctly predicted all 70 instances of class 1 but missed all 50 instances of class -1. This indicates a clear weakness of the SVM model in distinguishing between the two classes.



The Logistic Regression model did a better job distinguishing the two classes. It correctly predicted 13 out of 50 cases for class -1 and 62 out of 70 cases for class 1. Although it still misclassified some instances, it shows a significant improvement over the previous SVM model by predicting both classes more accurately.



This confusion matrix shows moderate performance. The Logistic Regression model correctly predicted 15 out of 50 cases for class -1 and 51 out of 70 cases for class 1. While it correctly identified many instances of class 1, it struggled more with class -1, showing room for improvement.



Deep Learning Models

Splitting the data

This function takes a list of numbers and breaks it into small groups “windows”. Each group of numbers is used to predict the next number in the sequence.

```python
def create_windows(series,window_size):
    X_seq,y_seq = [],[]
    for i in range(len(series)-window_size):
        X_seq.append(series[i:window_size+i])
        y_seq.append(series[window_size+i])
    return np.array(X_seq), np.array(y_seq)
```

Setting the number of windows to 90.
Taking the “Close” column of the stock_data, splitting it into sequences of length 90 (X_seq), and pairs each sequence with the next value (y_seq). This helps in predicting the next closing price from the previous 90 closing prices.

```python
window_size = 90
X_seq,y_seq = create_windows(stock_data['Close'],window_size)
```

spliting my data into two groups: 80% for training the model and 20% for validating it.

```python
split_time = int(len(X_seq)*0.8)
X_train_lstm,X_valid = X_seq[:split_time], X_seq[split_time:]

y_train_lstm,y_valid = y_seq[:split_time], y_seq[split_time:]
```

Building the model

```python
#building my deep learning model
model = tf.keras.models.Sequential([
    Input(shape=(window_size,1)),
    LSTM(128),
    tf.keras.layers.Dense(256,activation = 'relu'),
    tf.keras.layers.Dense(128,activation = 'relu'),
    tf.keras.layers.Dense(64,activation = 'relu'),
    tf.keras.layers.Dense(32,activation = 'relu'),
    tf.keras.layers.Dense(16,activation = 'relu'),
    tf.keras.layers.Dense(1)
])
```

This is my deep learning model, I gave it one big LSTM layer and a couple of Dense layers in decreasing order.

```python
#compiling the model
model.compile(
    loss = 'mae',
    optimizer = tf.keras.optimizers.Adam(learning_rate = 0.005),
    metrics = ['mae']
)
```

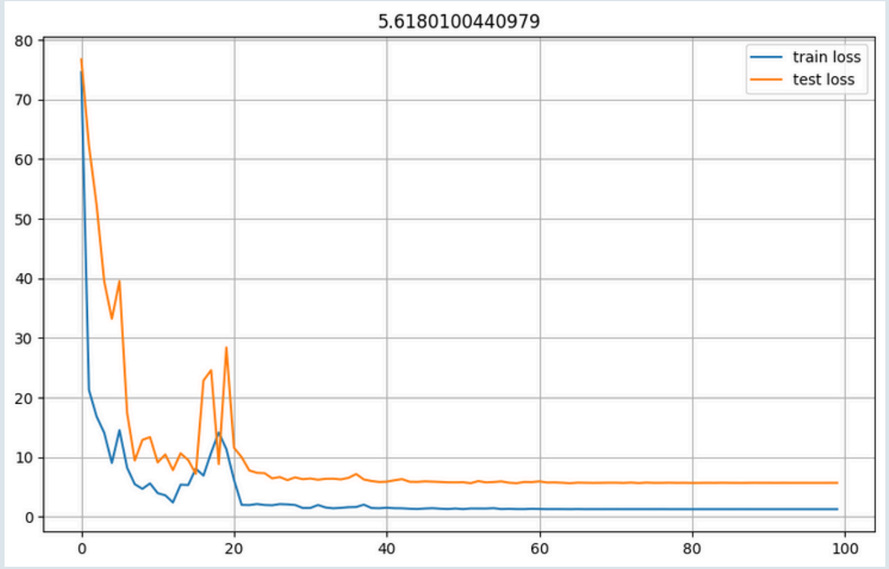
Compiling my model, I gave it mean absolute error as a loss function, as it worked best, and Adam optimizer with a learning rate of 0.005.

Deep Learning Model

Training the model

```
lr_scheduler = ReduceLROnPlateau(
    monitor = 'val_loss',
    factor = 0.5,
    patience = 5,
    verbose = 1,
    min_lr = 1e-6
)
```

I gave my model reduce learning rate on plateau, which reduces the learning rate if the val_loss is not going down for 5 straight epochs




```
#training the model
history = model.fit(
    X_train_lstm,y_train_lstm,
    validation_data = (X_valid,y_valid),
    epochs = 100,
    batch_size = 32,
    callbacks = [lr_scheduler]
)
```

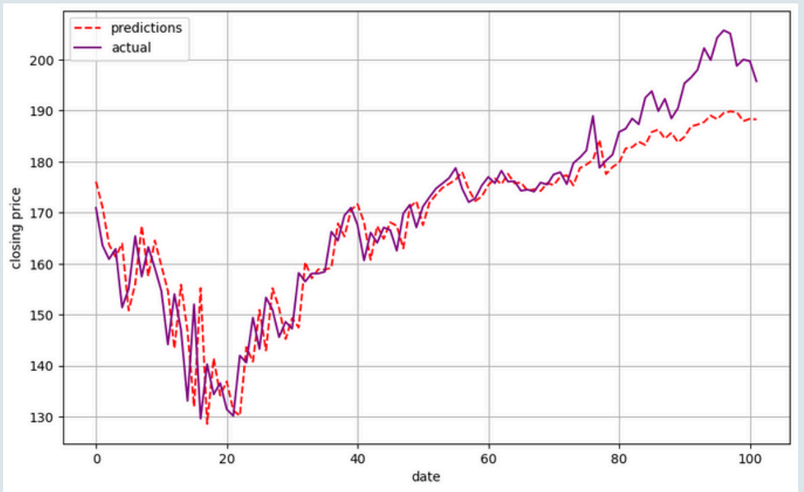
Training the model to see how well it did

made a graph to show that the loss and val loss actually went down by quite a lot although the loss function went down by more which can indicate there's slight overfitting but overall this model looks good.

```
#making a prediction
y_pred_lstm = model.predict(X_valid)
y_pred_lstm = np.squeeze(y_pred_lstm, axis=-1)
time_valid = stock_data.index[window_size:]
```

Made a prediction in order to plot the result

We can see the model predicted the stocks price quite good, except for the end, although it remained a positive trend, which can explain the slight overfitting. Overall this model preformed a lot better than the Machine Learning model



NLP Model

```
[ ]  1 #definind a tokenizer to vectorize all the words in the financial news dataset
     2 tokenizer = Tokenizer(oov_token = '<OOV>')
     3 tokenizer.fit_on_texts(reuters['Headlines']) #tokenizing the headlines column
     4 sequences = tokenizer.texts_to_sequences(reuters['Headlines']) #converting to a list of integers
     5 word_index = tokenizer.word_index
     6 padded = pad_sequences(sequences,maxlen = 500,padding = 'post', truncating = 'post') #making sure all sequences have the same length
```

Made a Tokenizer that takes all the headlines from the retuters financial news I showed earlier and makes them into a sequence

After that I padded them so they all have the same length

NLP model

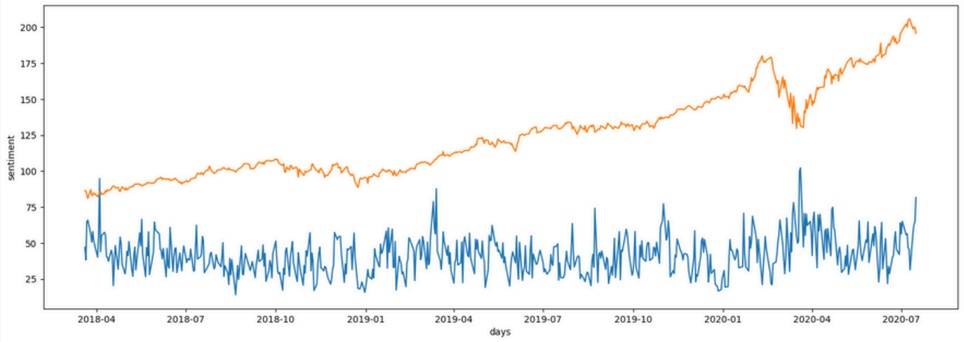
Made a simple model in order to get a sentiment

```
1 NLP_model = Sequential([
2     Embedding(input_dim = len(word_index)+1,output_dim = 128),
3     Dense(32,activation = 'relu'),
4     Dense(1)
5 ])
6
7 NLP_pred = NLP_model.predict(padded)
```

```
[('<OOV>', 1),
 ('say', 2),
 ('china', 3),
 ('trade', 4),
 ('billion', 5),
 ('deal', 6),
 ('source', 7),
 ('new', 8),
 ('ceo', 9),
 ('trump', 10),
 ('coronavirus', 11),
 ('oil', 12),
 ('cut', 13),
 ('exclusive', 14),
 ('share', 15),
 ('sale', 16),
 ('bank', 17),
 ('stock', 18),
 ('talk', 19),
 ('tariff', 20),
 ('market', 21),
 ('boeing', 22),
 ('fed', 23),
 ('million', 24),
 ('plan', 25),
 ('hit', 26),
 ('profit', 27),
 ('wall', 28),
 ('eu', 29),
 ('street', 30)]
```

30 of the most common words

The orange graph is the actual Close price and the Blue one is The sentiment, we can see that even though the sentiment looks more messy, it still has simillar spikes, to the stock.



splitting the data into feature and target but this time with the sentiment we made

```
1 def create_windows_nlp(series,window_size_nlp):
2   X,y = [],[]
3   for i in range(len(series)-window_size):
4     X.append(series[i:i+window_size])
5     y.append(series[i+window_size])
6   return np.array(X), np.array(y)
7
8 window_size_nlp = 90
9
10
11
12 nlp_scaler = StandardScaler()
13 scaled = nlp_scaler.fit_transform(merged_df[['Close','NLP_pred']])
14
15 nlp_X,nlp_y = create_windows_nlp(scaled,window_size_nlp)
16
17 split_time_nlp = int(len(nlp_X)*0.8)
18
19 X_train_nlp, X_valid_nlp = nlp_X[:split_time_nlp], nlp_X[split_time_nlp:]
20 y_train_nlp, y_valid_nlp = nlp_y[:split_time_nlp], nlp_y[split_time_nlp:]
```

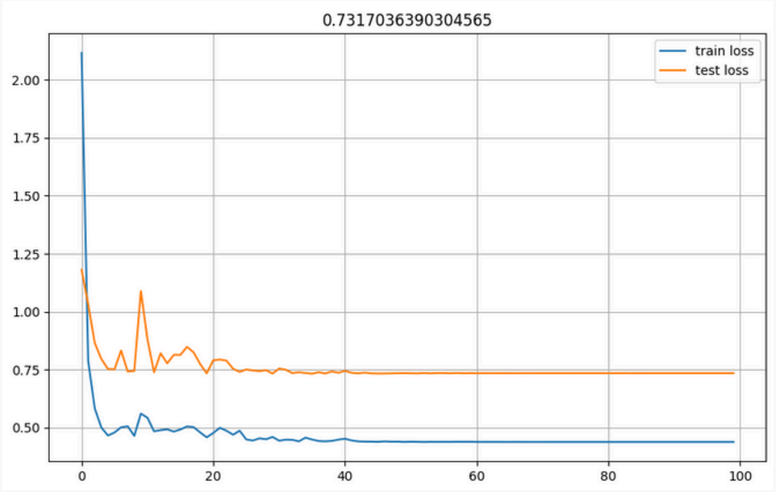
NLP Model

Building the Model

I decided to use the exact same model with an edition of lambda The Lambda layer subtracts 5 from the model's output to correct for a systematic offset in the predicted sentiment values. While this is a shortcut it allowed me to align the output with my expected sentiment scale

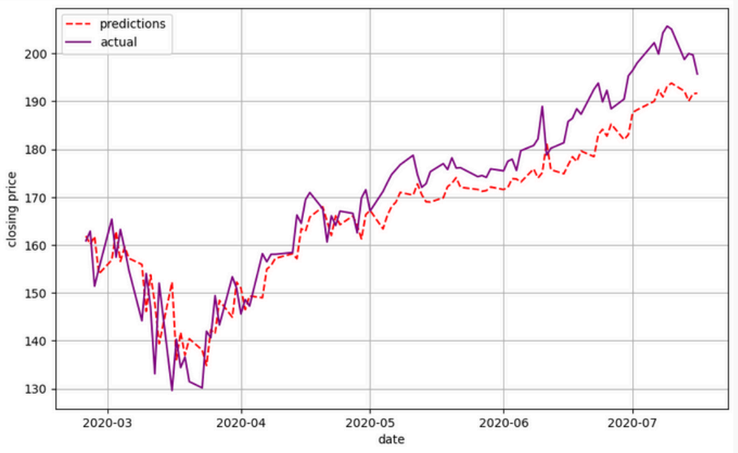
```python
#building my deep learning model
sentiment_model = tf.keras.models.Sequential([
    Input(shape=(window_size_nlp,2)),
    LSTM(128),
    tf.keras.layers.Dense(256,activation = 'relu'),
    tf.keras.layers.Dense(128,activation = 'relu'),
    tf.keras.layers.Dense(64,activation = 'relu'),
    tf.keras.layers.Dense(32,activation = 'relu'),
    tf.keras.layers.Dense(16,activation = 'relu'),
    tf.keras.layers.Dense(1),
    Lambda(lambda x:x-5)
])

#compiling the model
sentiment_model.compile(
    loss = 'mae',
    optimizer = tf.keras.optimizers.Adam(learning_rate = 0.005),
    metrics = ['mae']
)

lr_scheduler = ReduceLROnPlateau(
    monitor = 'val_loss',
    factor = 0.5,
    patience = 5,
    verbose = 1,
    min_lr = 1e-6
)

#training the model
sentiment_history = sentiment_model.fit(
    X_train_nlp,y_train_nlp,
    validation_data = (X_valid_nlp,y_valid_nlp),
    epochs = 100,
    batch_size = 32,
    callbacks = [lr_scheduler]
)
```

The loss function on this model is simillar to the model I have done before with a bigger gap between loss and the vall loss which indicates of a bigger overfitting




```python
#making a prediction
y_pred_nlp = sentiment_model.predict(X_valid_nlp)
y_pred_nlp = nlp_scaler.inverse_transform(np.concatenate([y_pred_nlp,np.zeros_like(y_pred_nlp)],axis = 1))
y_pred_nlp_close = y_pred_nlp[:,0]
time_valid_nlp = merged_df.index[split_time_nlp:]
```

And we can see on the graph of the predictions and the actual data that this model is less accurate, it still shows a positive trend, and is relatively accurate, just not as the one before



NLP Machine Learning Models

Building the Model

For the Machine Learning models I decided to go again with the exact same models and hyper parametrs, to see if we will get more accurate reslut or not

```
svm_model = svm.SVC(C = 1,
                    kernel = 'linear',
                    gamma = 'scale',
                    shrinking = True)

rf_model = RandomForestClassifier(min_samples_split = 5,
                                  n_estimators = 200,
                                  max_depth = 5,
                                  max_features = 'sqrt')

lr_model = LogisticRegression(C = 10,
                              max_iter = 100,
                              solver = 'liblinear')
```

flattening the X in order to make it on dimension array, and converting the y into a single integer and finding the index with the highest value.

```
X_train_nlp_flat = X_train_nlp.reshape(X_train_nlp.shape[0], -1)
y_train_nlp_ml = np.argmax(y_train_nlp, axis=1)
```

did the same to the validation data

```
X_valid_nlp_flat = X_valid_nlp.reshape(X_valid_nlp.shape[0], -1)
y_valid_nlp_ml = np.argmax(y_valid_nlp, axis=1)
```

this is the models accuracy score, surprisingly, all of them achived the same accuracy score, now lets check for overfitting

```
[160]  lr_nlp_pred = lr_model_nlp.predict(X_valid_nlp_flat)
       rf_nlp_pred = rf_model_nlp.predict(X_valid_nlp_flat)
       svm_nlp_pred = svm_model_nlp.predict(X_valid_nlp_flat)

       lr_nlp_acc = accuracy_score(y_valid_nlp_ml,lr_nlp_pred)
       rf_nlp_acc = accuracy_score(y_valid_nlp_ml,rf_nlp_pred)
       svm_nlp_acc = accuracy_score(y_valid_nlp_ml,svm_nlp_pred)

       print(f'logistic regression accuracy is: {lr_nlp_acc}')
       print(f'random forest accuracy is: {rf_nlp_acc}')
       print(f'svm accuracy is: {svm_nlp_acc}')

logistic regression accuracy is: 0.83
random forest accuracy is: 0.83
svm accuracy is: 0.83
```

NLP Machine Learning Models

Checking for overfitting

```
[161]  1 lr_nlp_train_pred = lr_model_nlp.predict(X_train_nlp_flat)
       2 rf_nlp_train_pred = rf_model_nlp.predict(X_train_nlp_flat)
       3 svm_nlp_train_pred = svm_model_nlp.predict(X_train_nlp_flat)
       4
       5 lr_nlp_train_acc = accuracy_score(y_train_nlp_ml,lr_nlp_train_pred)
       6 rf_nlp_train_acc = accuracy_score(y_train_nlp_ml,rf_nlp_train_pred)
       7 svm_nlp_train_acc = accuracy_score(y_train_nlp_ml,svm_nlp_train_pred)
       8
       9 print(f'logistic regression accuracy is: {lr_nlp_train_acc}')
      10 print(f'random forest accuracy is: {rf_nlp_train_acc}')
      11 print(f'svm accuracy is: {svm_nlp_train_acc}')

logistic regression accuracy is: 0.8813131313131313
random forest accuracy is: 0.8762626262626263
svm accuracy is: 0.8762626262626263
```

We can see that we do have slight overfitting in all models, but in the logistic regresssion model its the highest by a little bit, which is actually different from the machine learning model from the beginning of the project.

Even though this model (random forest) had high accuracy, it did not predict class 1 at all, which means this model is not good enough even though it predicted most of the 0 class correctly.

```
1 rf_classification_report = classification_report(y_valid_nlp_ml, rf_nlp_pred)
2 print(rf_classification_report)
```

	precision	recall	f1-score	support
0	0.83	1.00	0.91	83
1	0.00	0.00	0.00	17
accuracy			0.83	100
macro avg	0.41	0.50	0.45	100
weighted avg	0.69	0.83	0.75	100

This model did better than the random forest model but not perfect as it predicted mostly the 0 class correctly

```
1 svm_classification_report = classification_report(y_valid_nlp_ml, svm_nlp_pred)
2 print(svm_classification_report)
```

	precision	recall	f1-score	support
0	0.84	0.99	0.91	83
1	0.50	0.06	0.11	17
accuracy			0.83	100
macro avg	0.67	0.52	0.51	100
weighted avg	0.78	0.83	0.77	100

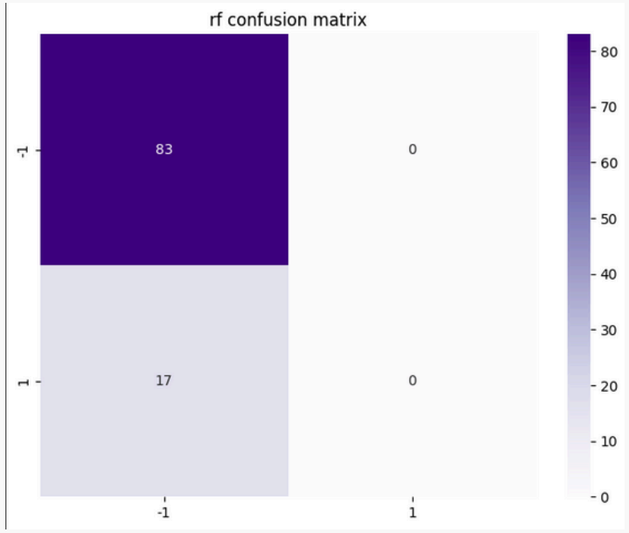
this model did the best in terms of capturing both classes

```
1 lr_classification_report = classification_report(y_valid_nlp_ml, lr_nlp_pred)
2 print(lr_classification_report)
```

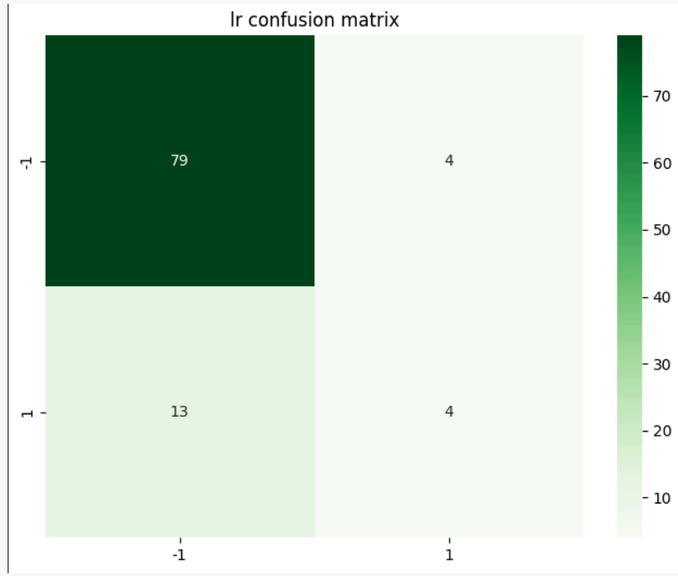
	precision	recall	f1-score	support
0	0.86	0.95	0.90	83
1	0.50	0.24	0.32	17
accuracy			0.83	100
macro avg	0.68	0.59	0.61	100
weighted avg	0.80	0.83	0.80	100

NLP Machine Learning Models

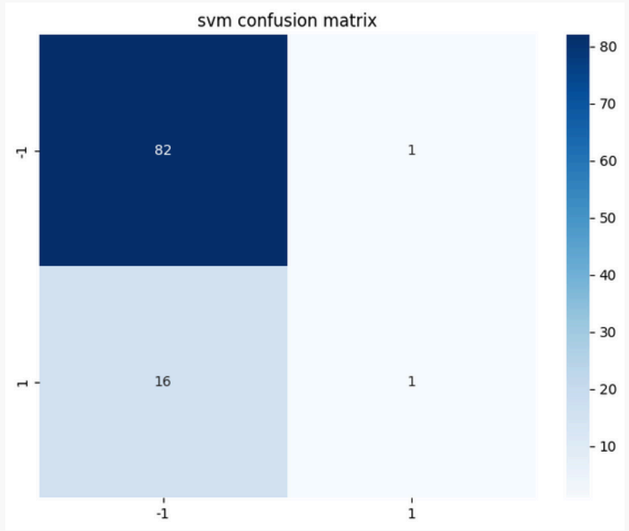
Confusion matrix



This is the confusion matrix for the random forest model, we can see it was the one the succeeded the least as it hasn't predicted class 1 at all



This model did the best in terms of prediction, but its still not as good as I would like it to be



This model did better but not by much, it only predicted one correct class 1

Project Conclusions

Building the Model

In conclusion, my project explored forecasting Microsoft's stock prices using both deep learning and traditional machine learning models. The LSTM model, built on time-windowed historical data, effectively captured the temporal trends in stock prices, while the machine learning models provided complementary insights, especially when integrating market sentiment from news headlines. Despite facing challenges like overfitting and class imbalances, combining these approaches demonstrated a solid framework for financial forecasting, highlighting the potential benefits of leveraging multiple analytical techniques for improved prediction accuracy.

Thank you for your time!

Idan Hasdai

E-MAIL

idanhas3@gmail.com