Forecasting Fortunes: Stock Price Prediction

CS3244 Group 07

01

Motivation

Motivation

We empower investors with actionable insights by **forecasting stock price movements** using current data.

Description

Focus: GOOGL stocks

Data: stock name, change in stock price, percentage change in price, opening price, closing price, and volume of stocks transacted

Response variable: price of stock the following day

02

Data Prep

Data Collection

Source: Yahoo Finance API

Table 1

	Date	Open	High	Low	Close	Adj Close	Volume	Company
0	2004-08-19	2.502503	2.604104	2.401401	2.511011	2.511011	893181924	GOOGL
1	2004-08-20	2.527778	2.729730	2.515015	2.710460	2.710460	456686856	GOOGL
2	2004-08-23	2.771522	2.839840	2.728979	2.737738	2.737738	365122512	GOOGL
3	2004-08-24	2.783784	2.792793	2.591842	2.624374	2.624374	304946748	GOOGL
4	2004-08-25	2.626627	2.702703	2.599600	2.652653	2.652653	183772044	GOOGL
...	...	...	...	...	...	...	...	...
4869	2023-12-05	128.949997	132.139999	128.250000	130.990005	130.990005	27384800	GOOGL
4870	2023-12-06	131.440002	131.839996	129.880005	130.020004	130.020004	23576200	GOOGL
4871	2023-12-26	141.589996	142.679993	141.190002	141.520004	141.520004	16780300	GOOGL
4872	2023-12-27	141.589996	142.080002	139.889999	140.369995	140.369995	19628600	GOOGL
4873	2023-12-28	140.779999	141.139999	139.750000	140.229996	140.229996	16045700	GOOGL

Data Collection

Source: Yahoo Finance API

Table 1

IPO

Weekends

	Date	Open	High	Low	Close	Adj Close	Volume	Company
0	2004-08-19	2.502503	2.604104	2.401401	2.511011	2.511011	893181924	GOOGL
1	2004-08-20	2.527778	2.729730	2.515015	2.710460	2.710460	456686856	GOOGL
2	2004-08-23	2.771522	2.839840	2.728979	2.737738	2.737738	365122512	GOOGL
3	2004-08-24	2.783784	2.792793	2.591842	2.624374	2.624374	304946748	GOOGL
4	2004-08-25	2.626627	2.702703	2.599600	2.652653	2.652653	183772044	GOOGL
...	...	...	...	...	...	...	...	...
4869	2023-12-05	128.949997	132.139999	128.250000	130.990005	130.990005	27384800	GOOGL
4870	2023-12-06	131.440002	131.839996	129.880005	130.020004	130.020004	23576200	GOOGL
4871	2023-12-26	141.589996	142.679993	141.190002	141.520004	141.520004	16780300	GOOGL
4872	2023-12-27	141.589996	142.080002	139.889999	140.369995	140.369995	19628600	GOOGL
4873	2023-12-28	140.779999	141.139999	139.750000	140.229996	140.229996	16045700	GOOGL

01/01/2000 – 31/12/2023

Data Collection

Source: Yahoo Finance API
Table 1

	Date	Open	High	Low	Close	Adj Close	Volume	Company
0	2004-08-19	2.502503	2.604104	2.401401	2.511011	2.511011	893181924	GOOGL
1	2004-08-20	2.527778	2.729730	2.515015	2.710460	2.710460	456686856	GOOGL
2	2004-08-23	2.771522	2.839840	2.728979	2.737738	2.737738	365122512	GOOGL
3	2004-08-24	2.783784	2.792793	2.591842	2.624374	2.624374	304946748	GOOGL
4	2004-08-25	2.626627	2.702703	2.599600	2.652653	2.652653	183772044	GOOGL
...	...	...	...	...	...	...	...	...
4869	2023-12-05	128.949997	132.139999	128.250000	130.990005	130.990005	27384800	GOOGL
4870	2023-12-06	131.440002	131.839996	129.880005	130.020004	130.020004	23576200	GOOGL
4871	2023-12-26	141.589996	142.679993	141.190002	141.520004	141.520004	16780300	GOOGL
4872	2023-12-27	141.589996	142.080002	139.889999	140.369995	140.369995	19628600	GOOGL
4873	2023-12-28	140.779999	141.139999	139.750000	140.229996	140.229996	16045700	GOOGL

Daily Adjusted Closing Price

Data Preprocessing

Source: Yahoo Finance API
Table 1

	Date	Open	High	Low	Close	Adj Close	Volume	Company	Tomorrow
0	2004-08-19	2.502503	2.604104	2.401401	2.511011	2.511011	893181924	GOOGL	2.710460
1	2004-08-20	2.527778	2.729730	2.515015	2.710460	2.710460	456686856	GOOGL	2.737738
2	2004-08-23	2.771522	2.839840	2.728979	2.737738	2.737738	365122512	GOOGL	2.624374
3	2004-08-24	2.783784	2.792793	2.591842	2.624374	2.624374	304946748	GOOGL	2.652653
4	2004-08-25	2.626627	2.702703	2.599600	2.652653	2.652653	183772044	GOOGL	2.700450
...	...	...	...	...	...	...	...	...	...
4869	2023-12-05	128.949997	132.139999	128.250000	130.990005	130.990005	27384800	GOOGL	130.020004
4870	2023-12-06	131.440002	131.839996	129.880005	130.020004	130.020004	23576200	GOOGL	136.929993
4871	2023-12-26	141.589996	142.679993	141.190002	141.520004	141.520004	16780300	GOOGL	140.369995
4872	2023-12-27	141.589996	142.080002	139.889999	140.369995	140.369995	19628600	GOOGL	140.229996
4873	2023-12-28	140.779999	141.139999	139.750000	140.229996	140.229996	16045700	GOOGL	139.690002

Data Preprocessing

Source: Yahoo Finance API
Table 2

Date	AC 0	AC 1	AC 2	AC 3	AC 4	AC 5	AC 6	AC 7	AC 8	...	AC 51	AC 52	AC 53	AC 54	AC 55	AC 56	AC 57	AC 58	AC 59	AC Current
2004-11-12	2.511011	2.71046	2.737738	2.624374	2.652653	2.70045	2.656406	2.552803	2.561812	...	4.905656	4.876627	4.796547	4.622122	4.237988	4.318068	4.221722	4.200701	4.58008	4.554555
2004-11-15	2.71046	2.737738	2.624374	2.652653	2.70045	2.656406	2.552803	2.561812	2.508759	...	4.876627	4.796547	4.622122	4.237988	4.318068	4.221722	4.200701	4.58008	4.554555	4.626376
2004-11-16	2.737738	2.624374	2.652653	2.70045	2.656406	2.552803	2.561812	2.508759	2.54029	...	4.796547	4.622122	4.237988	4.318068	4.221722	4.200701	4.58008	4.554555	4.626376	4.317818
2004-11-17	2.624374	2.652653	2.70045	2.656406	2.552803	2.561812	2.508759	2.54029	2.502753	...	4.622122	4.237988	4.318068	4.221722	4.200701	4.58008	4.554555	4.626376	4.317818	4.316817
2004-11-18	2.652653	2.70045	2.656406	2.552803	2.561812	2.508759	2.54029	2.502753	2.542042	...	4.237988	4.318068	4.221722	4.200701	4.58008	4.554555	4.626376	4.317818	4.316817	4.192693
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2023-12-05	131.940002	132.600006	135.800003	136.649994	138.339996	140.419998	141.490005	68.720001	70.458	...	136.970001	138.490005	136.690002	136.410004	137.199997	134.990005	132.529999	131.860001	129.270004	130.990005
2023-12-06	132.600006	135.800003	136.649994	138.339996	140.419998	141.490005	68.720001	70.458	70.337502	...	138.490005	136.690002	136.410004	137.199997	134.990005	132.529999	131.860001	129.270004	130.990005	130.020004
2023-12-26	135.800003	136.649994	138.339996	140.419998	141.490005	68.720001	70.458	70.337502	70.662003	...	136.690002	136.410004	137.199997	134.990005	132.529999	131.860001	129.270004	130.990005	130.020004	141.520004
2023-12-27	136.649994	138.339996	140.419998	141.490005	68.720001	70.458	70.337502	70.662003	71.068497	...	136.410004	137.199997	134.990005	132.529999	131.860001	129.270004	130.990005	130.020004	141.520004	140.369995
2023-12-28	138.339996	140.419998	141.490005	68.720001	70.458	70.337502	70.662003	71.068497	71.014	...	137.199997	134.990005	132.529999	131.860001	129.270004	130.990005	130.020004	141.520004	140.369995	140.229996

Current + Past 60 Days Adj Close Price

Data Preprocessing

Source: Yahoo Finance API
Table 3

Company	Date	Adj Close	Moving Avg	Moving Std	Moving Min	Moving Max	Moving Range	Moving Trend	Tomorrow	Change
GOOGL	2004-11-11	4.580080	3.408296	0.772473	2.502753	4.905656	2.402903	1.171784	4.554555	-0.025525
GOOGL	2004-11-12	4.554555	3.442355	0.777276	2.502753	4.905656	2.402903	1.112200	4.626376	0.071821
GOOGL	2004-11-15	4.626376	3.474287	0.786004	2.502753	4.905656	2.402903	1.152089	4.317818	-0.308558
GOOGL	2004-11-16	4.317818	3.500621	0.787377	2.502753	4.905656	2.402903	0.817197	4.316817	-0.001001
GOOGL	2004-11-17	4.316817	3.528829	0.785768	2.502753	4.905656	2.402903	0.787988	4.192693	-0.124124
...	...	...	...	...	...	...	...	...	...	...
GOOGL	2023-12-04	129.270004	102.493717	31.683017	68.126999	141.490005	73.363006	26.776287	130.990005	1.720001
GOOGL	2023-12-05	130.990005	102.477884	31.668286	68.126999	141.490005	73.363006	28.512121	130.020004	-0.970001
GOOGL	2023-12-06	130.020004	102.434884	31.628419	68.126999	141.490005	73.363006	27.585120	141.520004	11.500000
GOOGL	2023-12-26	141.520004	102.530217	31.739118	68.126999	141.520004	73.393005	38.989787	140.369995	-1.150009
GOOGL	2023-12-27	140.369995	102.592217	31.810451	68.126999	141.520004	73.393005	37.777778	140.229996	-0.139999

Moving metrics of Adj Close prices of past 60 days

Data Storage and Retrieval

```python
import os
from supabase import create_client

# Store url and key values
os.environ['SUPABASE_URL'] = 'https://tdjanfzeomxcvcc
os.environ['SUPABASE_KEY'] = 'eyJhbGciOiJIUzI1NiIsInR

# Retrieve url and key values
supabase_url = os.environ.get('SUPABASE_URL')
supabase_key = os.environ.get('SUPABASE_KEY')

# Create supabase connection
sb = create_client(supabase_url, supabase_key)
```

supabase

Helper functions:

`insert_data(df, tablename)`

`fetch_company_data(company, tablename)`

03

Models

Vanilla Linear Regression

LR equation:

Slopes of hyper-plane

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x} = \theta_d x_d + \cdots + \theta_2 x_2 + \theta_1 x_1 + \theta_0$$

bias (offset)

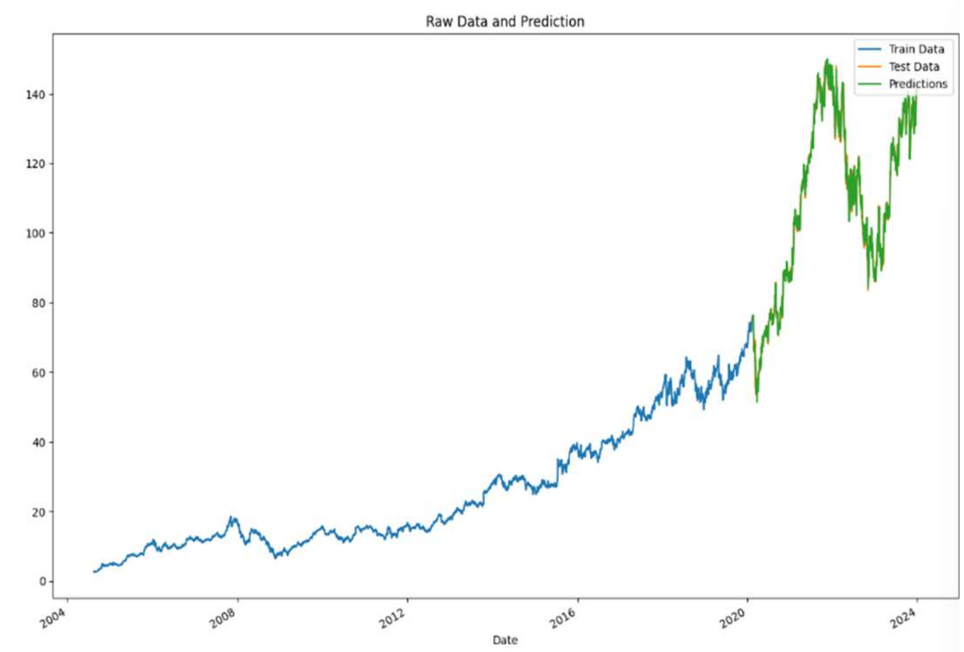
Features from raw data

```python
# Features
X = df.loc[:, df.columns.isin(['Open', 'High', 'Low', 'Volume'])]
y = df.loc[:, ['Adj Close']]
```

Vanilla Linear Regression

	mape	mae	rmse	mse	r2
0	0.565037	0.003915	0.005087	0.000026	0.999078

Plot of Adjusted Close over time



Linear Regression with Time-series Feature Engineering

Implementation of 60-day sliding window

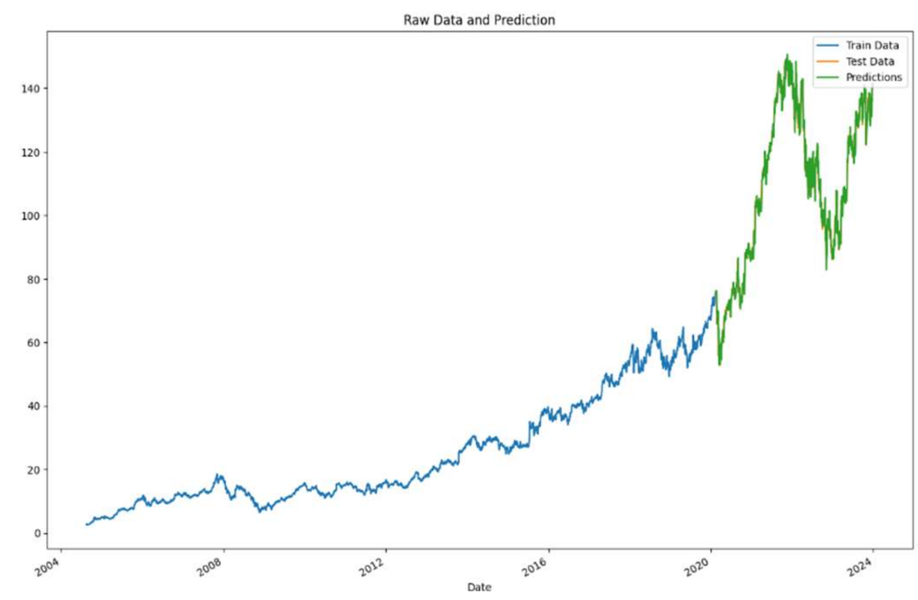
```
def slide(data, X_period, y_period):
  X = []
  y = []
  for i in range(X_period, len(data) - y_period + 1):
    X.append(data[i - X_period:i, 0]) # every data before 60th day
    y.append(data[i + y_period -1: i + y_period, 0]) # data for 60th day
  X = np.array(X)
  y = np.array(y)

  return [X, y]
```

Linear Regression with Time Series Feature Engineering

	mape	mae	rmse	mse	r2
0	1.634051	0.011248	0.015312	0.000234	0.991651

Plot of Adjusted Close over time



Ridge Regression

Penalty term in Ridge Regression

$$min_\theta L_{task}(\theta) \text{ s.t.} ||\theta||_2^2 \leq C \Leftrightarrow min_\theta L_{task}(\theta) + \lambda||\theta||_2^2$$

Ridge Regression with Time Series Feature Engineering

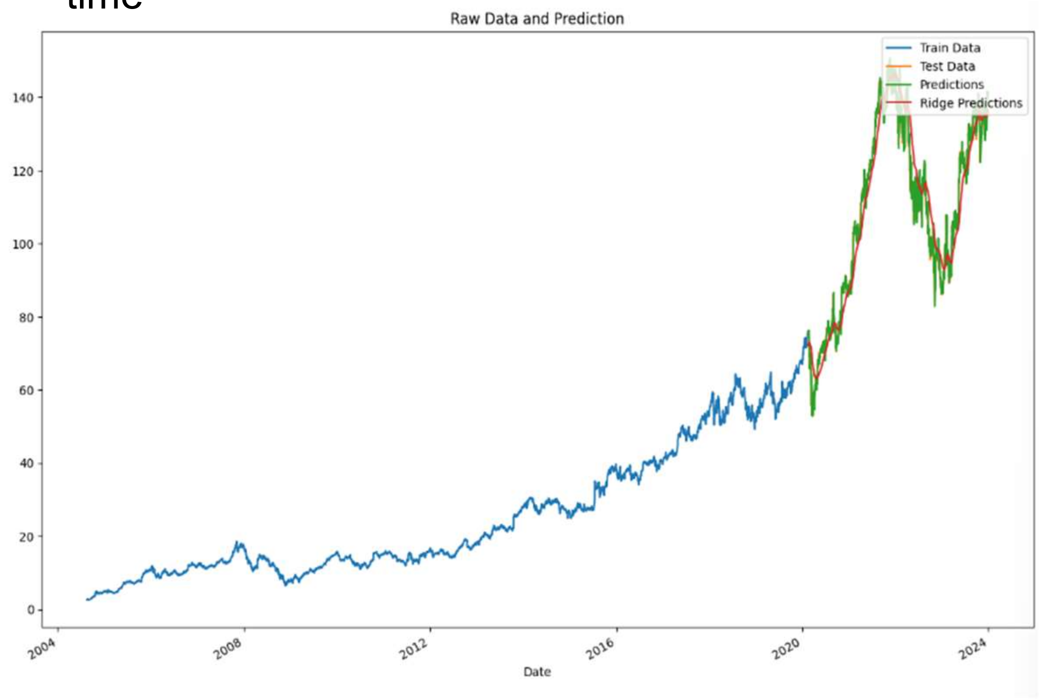
	mape	mae	rmse	mse	r2
0	4.991251	0.033978	0.042427	0.0018	0.935895

High (>10) Variance Inflation Factors indicate high multi-collinearity

	feature	VIF
0	const	2.066107
1	0	1394.496282
2	1	2686.994652
3	2	2687.720912
4	3	2691.031081
...	...	...

	feature	VIF
56	55	2666.049707
57	56	2667.620814
58	57	2669.944515
59	58	2673.422540
60	59	1384.544547

Plot of Adjusted Close over time



Gradient Boosting (Time series split)

GB equation: $\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i)$

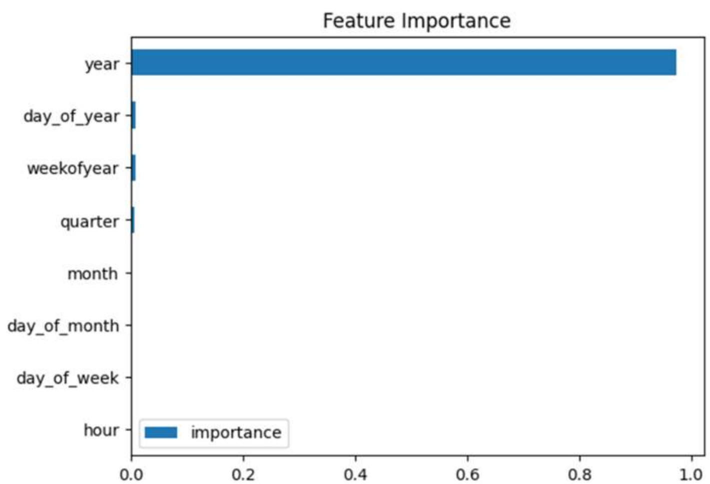
RMSE:

	Time-series split
0	56.254721

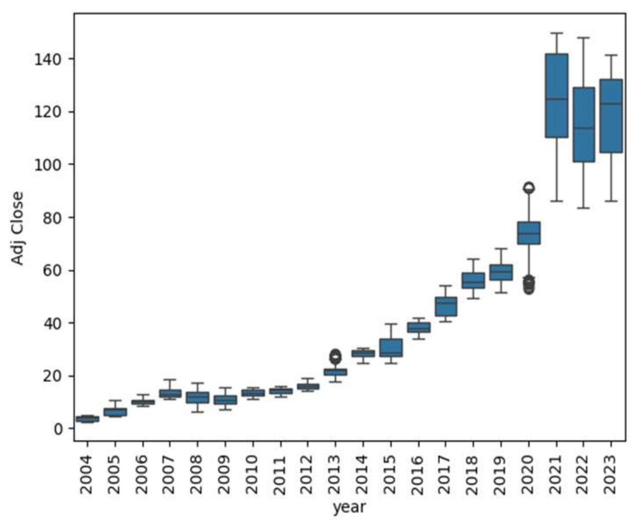
Implementation to feature engineer time of year features

```python
def create_features(df):
    """
    Create time series features based on time series index.
    """
    df = df.copy()
    df['hour'] = df.index.hour
    df['day_of_week'] = df.index.dayofweek
    df['quarter'] = df.index.quarter
    df['month'] = df.index.month
    df['year'] = df.index.year
    df['day_of_year'] = df.index.dayofyear
    df['day_of_month'] = df.index.day
    df['weekofyear'] = df.index.isocalendar().week
    return df
```

Feature Importance Plot shows year is the most important

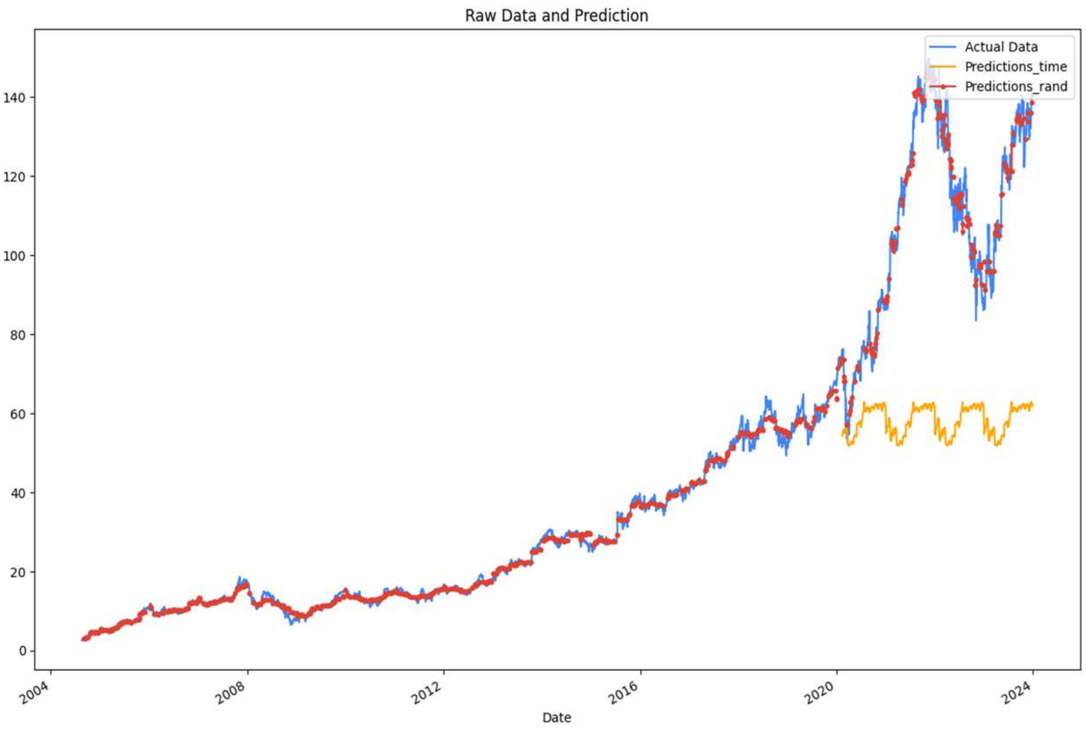


Boxplot of Adjusted Close against year



Gradient Boosting (Random Split)

Predictions made by time series split (yellow) compared to random split (red)

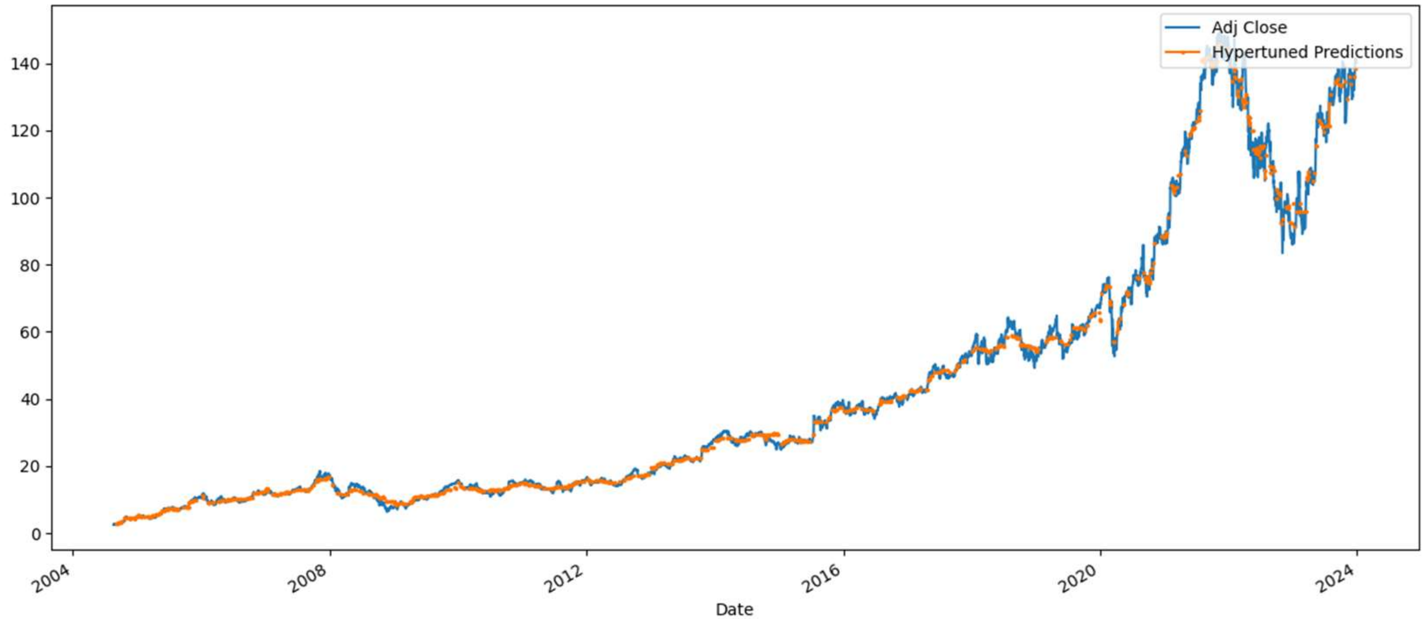


RMSE: Time-series vs Random Split

	Time-series split	Random split
0	56.254721	2.03732

Hypertuned Gradient Boosting

Predictions on Adjusted Close made by hypertuned GB model



Best GB parameters using grid search

```
grid_XGB.best_params_
```

```
{'early_stopping_rounds': 50,
 'learning_rate': 0.1,
 'max_depth': 8,
 'n_estimators': 1000,
 'subsample': 0.4000000000000001}
```

RMSE of the 3 models

	Time-series split	Random split	Random split with hypertuning
0	56.254721	2.03732	1.589624

Seasonal Autoregressive Integrated Moving Average (SARIMA)

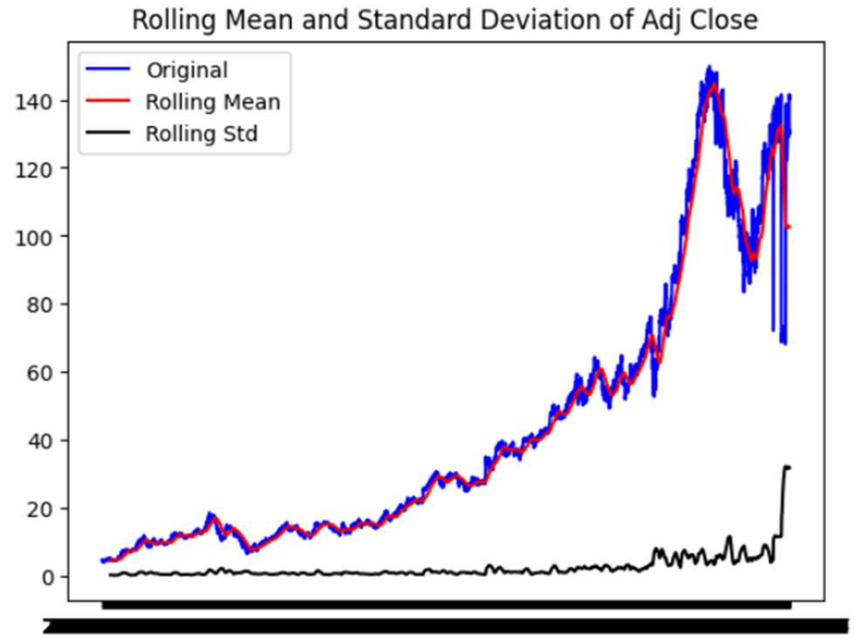
$$\text{ARIMA} \quad \underbrace{(p, d, q)}_{\substack{\uparrow \\ \text{Non-seasonal part} \\ \text{of the model}}} \quad \underbrace{(P, D, Q)_m}_{\substack{\uparrow \\ \text{Seasonal part} \\ \text{of the model}}}$$

p : The number of autoregressive (AR) terms for the non-seasonal component.
d : The degree of differencing for the non-seasonal component.
q : The number of moving average (MA) terms for the non-seasonal component.
P : The number of seasonal autoregressive (SAR) terms.
D : The degree of seasonal differencing
Q : The number of seasonal moving average (SMA) terms.
S : The seasonal period

Augmented Dickey Fuller (ADF) Test

Null hypothesis: The series has a unit root (non-stationary)

Alternate hypothesis: The series has no unit root (stationary)

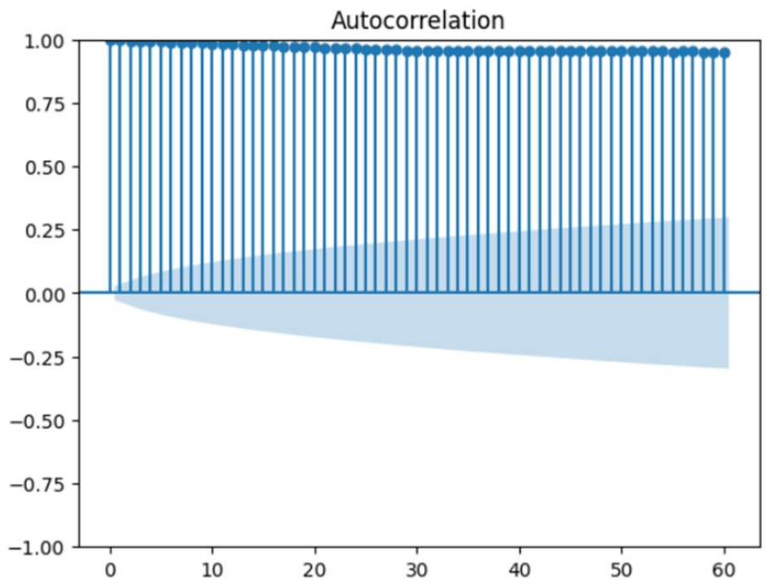



```
Results of dickey fuller test
Test Statistics                  -0.055389
p-value                           0.953705
No. of lags used                 32.000000
Number of observations used    4781.000000
critical value (1%)              -3.431719
critical value (5%)              -2.862145
critical value (10%)             -2.567092
dtype: float64
```

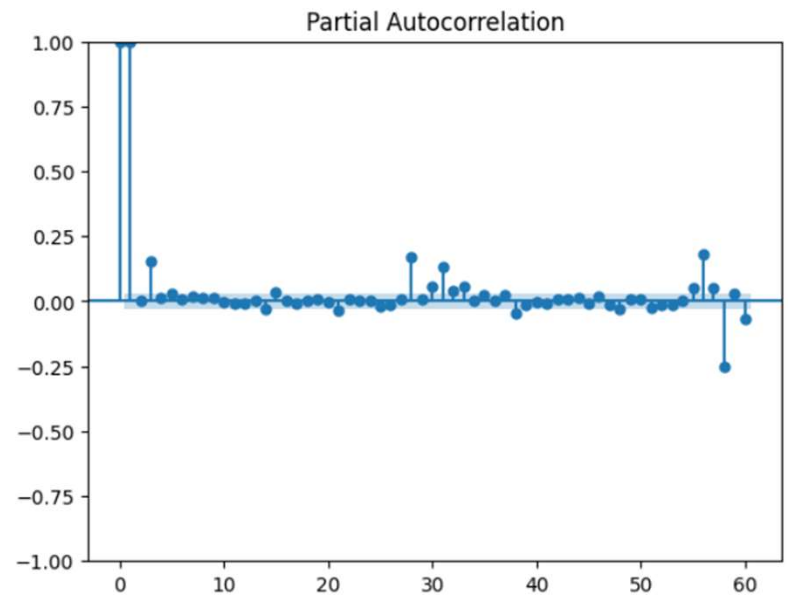
p-value > 0.05, null hypothesis is not rejected

Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF)

ACF Plot



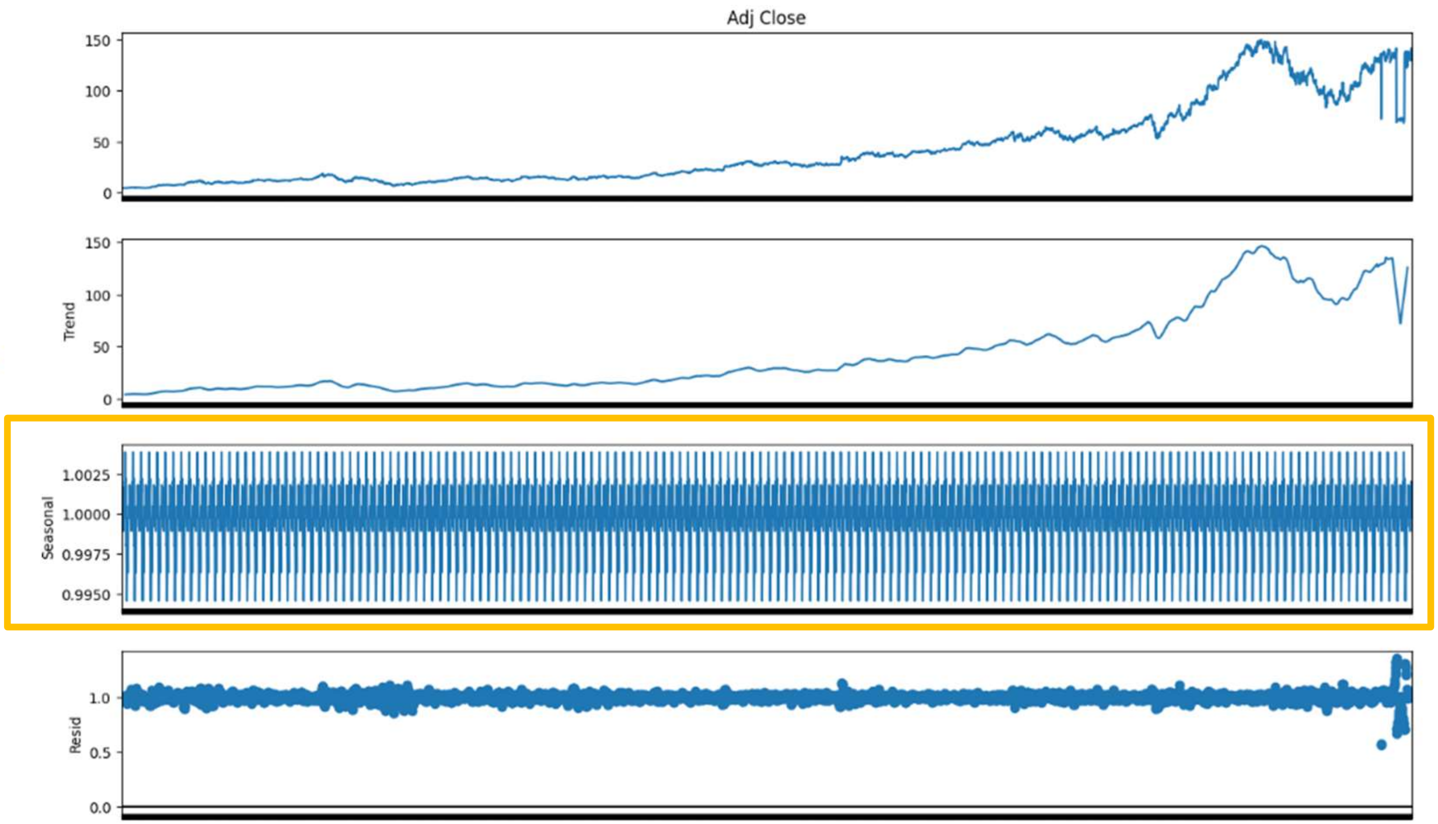
PACF Plot



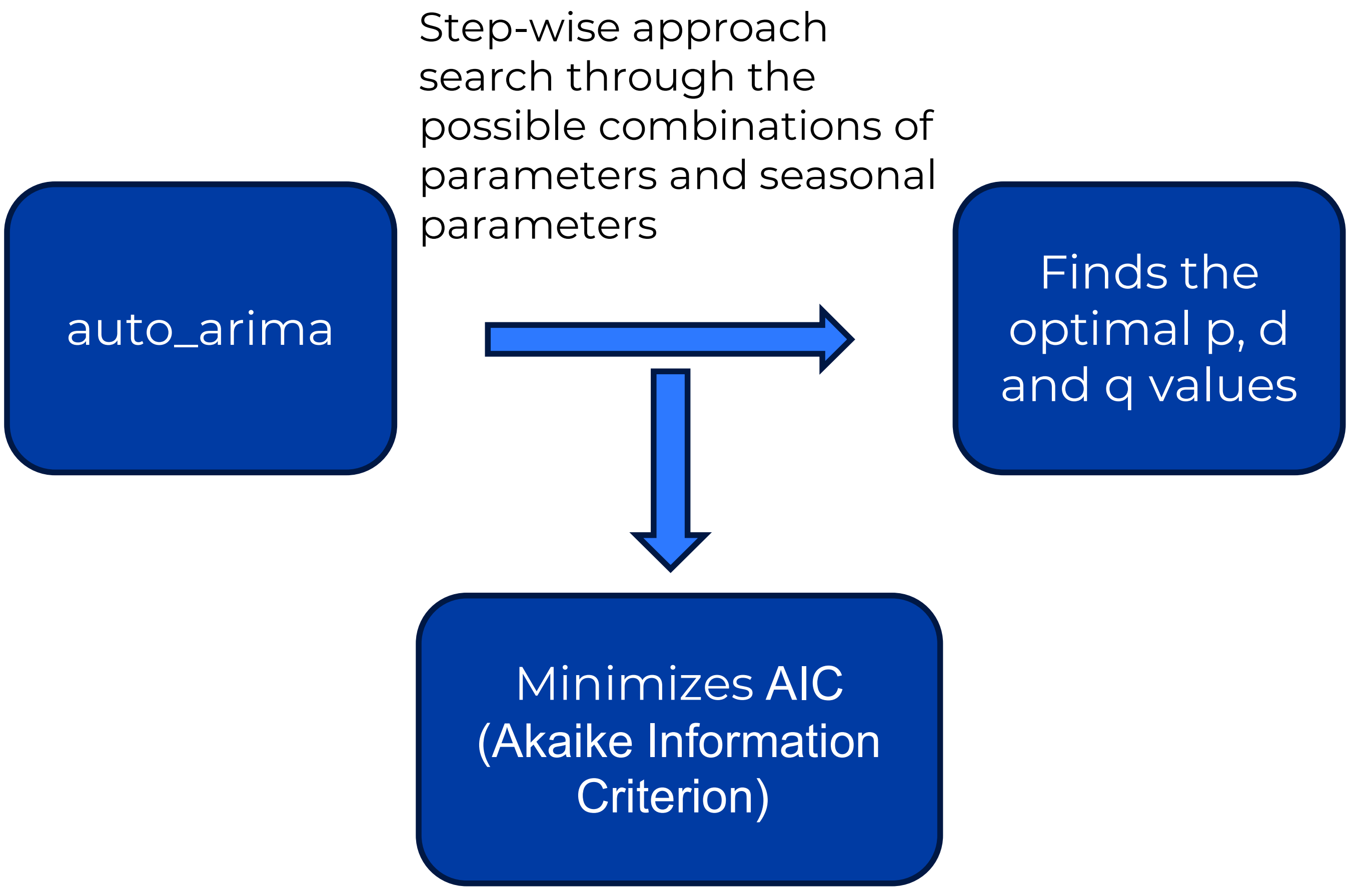
The plots imply that the time series may exhibit a seasonal pattern

Multiplicative Decomposition

$$y_t = S_t \times T_t \times R_t.$$



Model Training

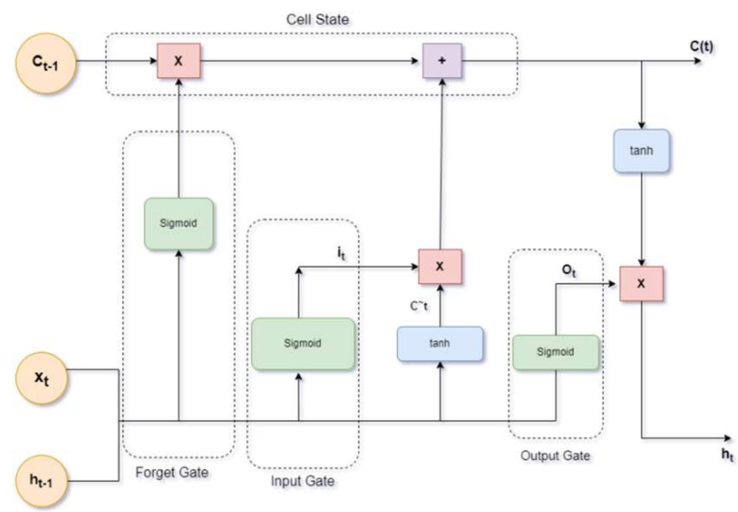


LSTM (Long-Short Term Memory)

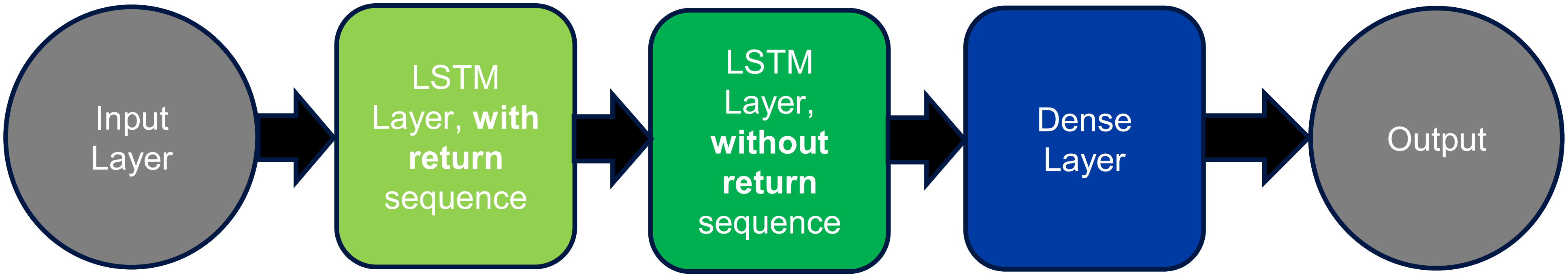
LSTM Architecture

Advantages of LSTM

- **Long-term Dependencies**: LSTM is capable of learning long-term dependencies.
- **Handling of Sequential Data**: It can process data with time steps of different lengths, unlike traditional neural networks.
- **Avoiding Vanishing Gradient Problem**: LSTM networks are designed to avoid the long-term dependency problem or the vanishing gradient problem.



Initial Implementation




```python
def set_model(train, lstm_1, lstm_2, dense):
  model = Sequential([
    keras.Input(shape=(X_train.shape[1], X_train.shape[2])),
    LSTM(lstm_1, return_sequences=True),
    LSTM(lstm_2, return_sequences=False),
    Dense(dense),
    Dense(train.shape[2])
  ])
  model.compile(optimizer=Adam(learning_rate=0.001),
    loss='mean_squared_error', metrics=[RootMeanSquaredError()])
  return model

model = set_model(X_train, 50, 50, 50)
```

X_train.shape[1] = 60 (No of days used as predictor)
X_train.shape[2] = 1 (No of features from original dataset)

Model: "sequential_1"

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 60, 50)	10,400
lstm_3 (LSTM)	(None, 50)	20,200
dense_2 (Dense)	(None, 50)	2,550
dense_3 (Dense)	(None, 1)	51

Total params: 33,201 (129.69 KB)
Trainable params: 33,201 (129.69 KB)
Non-trainable params: 0 (0.00 B)

Hypertuning

We wish to tune the following:

- **lstm_1**: No. of units in first LSTM layer
- **lstm_2**: No. of units in second LSTM layer
- **dense**: No. of neurons in dense layer
- **drop**: rate of dropout in Dropout layer
- **learning_rate**: momentum coefficient in Adam optimizer
- **batch_size**: Number of samples propogated
- **window_size**: No. of days used to predict next day's price

```
class MyHyperModel(keras_tuner.HyperModel):
  def __init__(self):
    self.window_size = 50

  # override
  def build(self, hp):
    lstm_1 = hp.Int('lstm_1', min_value=50, max_value=70, step=10, default=50)
    lstm_2 = hp.Int('lstm_2', min_value=50, max_value=70, step=10, default=50)
    dense = hp.Int('dense', min_value=50, max_value=70, step=10, default=50)
    drop = hp.Float("drop_rate", min_value=1e-1, max_value=5e-1, step=5e-2, default=2e-1)

    model = Sequential([
      keras.Input(shape=(self.window_size, 1)),
      LSTM(units=lstm_1, return_sequences=True),
      LSTM(units=lstm_2, return_sequences=False),
      Dense(units=dense),
      Dropout(rate=drop),
      Dense(1)
    ])

    hp_rates = hp.Float("learning_rate", min_value=1e-4,
    max_value=1e-2, sampling='log', default=1e-3)

    model.compile(optimizer=Adam(learning_rate=hp_rates),
      loss='mean_squared_error', metrics=[RootMeanSquaredError()])
    return model

  # override
  def fit(self, hp, model, scaled, *args, **kwargs):
    hp_batch = hp.Int("batch_size", min_value=1, max_value=15, step=1, default=1)
    hp_window = hp.Int("window_size", min_value=50, max_value=70, step=10, default=60)
    # hp_window = self.window_size
    self.window_size = hp_window
    X_train, y_train, X_val, y_val, X_test, y_test = self.tt_split(scaled, hp_window, 1)
    return model.fit(x=X_train, y=y_train,
      validation_data=(X_val, y_val), batch_size=hp_batch, *args, **kwargs)
```

Executing GridSearch

```
tuner = keras_tuner.GridSearch(
    hypermodel=MyHyperModel(),
    objective=keras_tuner.Objective(name='val_loss', direction='min'),
    max_trials=3,
    seed=1234,
    directory="results",
    project_name="custom_training",
    overwrite=True
  )

tuner.search(scaled=scaled2, epochs=3)
```

```
Trial 3 Complete [00h 02m 49s]
val_loss: 0.0006565555231645703

Best val_loss So Far: 0.0005446489085443318
Total elapsed time: 00h 07m 43s
```

	lstm_1	lstm_2	dense	drop_rate	learning_rate	batch_size	window_size
0	50	50	50	0.02	0.001	1	50

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 70, 50)	10,400
lstm_1 (LSTM)	(None, 50)	20,200
dense (Dense)	(None, 50)	2,550
dropout (Dropout)	(None, 50)	0
dense_1 (Dense)	(None, 1)	51

Total params: 33,201 (129.69 KB)
Trainable params: 33,201 (129.69 KB)
Non-trainable params: 0 (0.00 B)

04

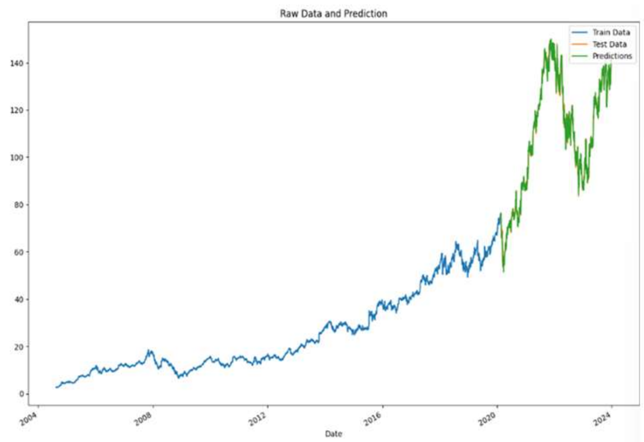
Conclusion

Analysis of Results

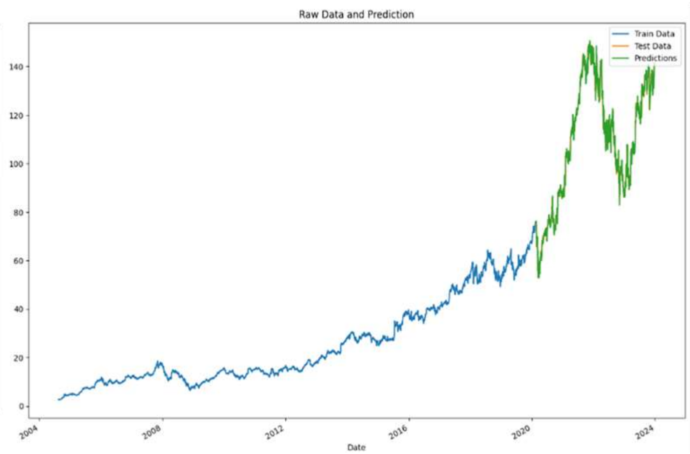
Linear regression

- Vanilla linear regression: overfitting
- Sliding window linear regression: overfitting
- Ridge regression: less overfitting, reduced variance but increased bias

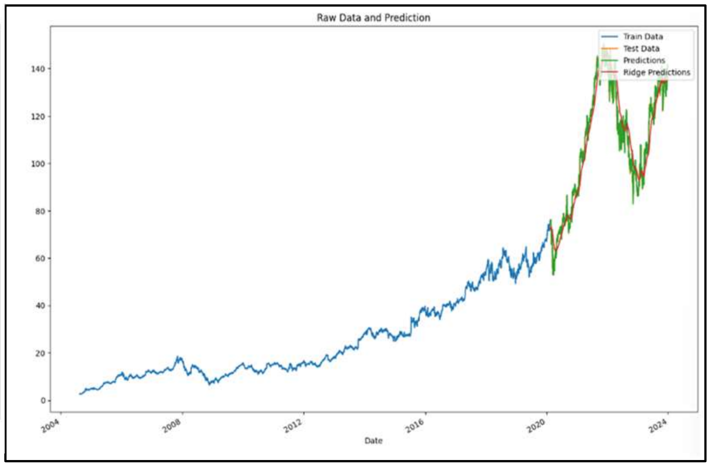
Too much overfitting because of multicollinearity, violates assumption of linear regression



Vanilla

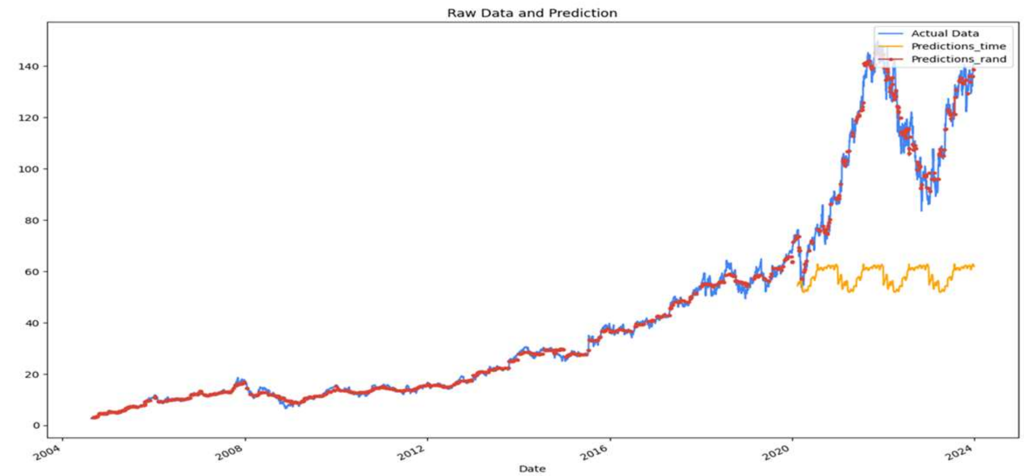


Sliding Window



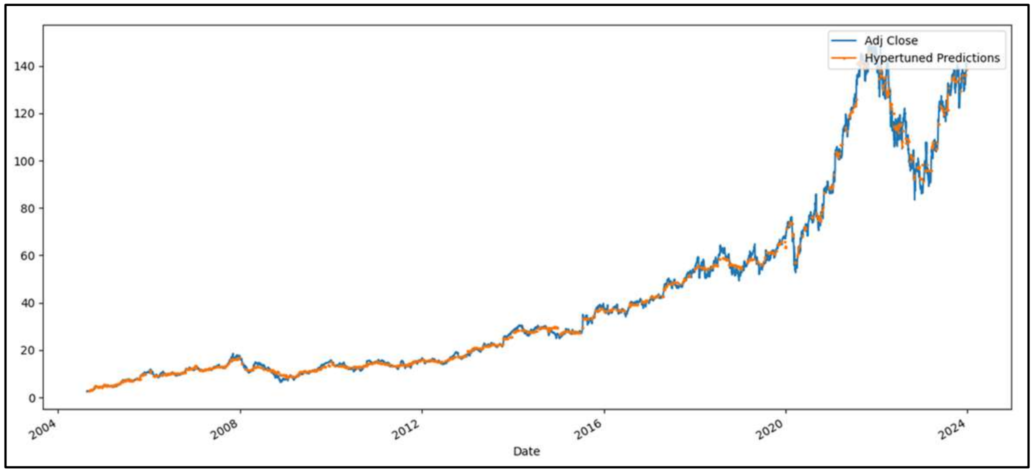
Ridge Regression

Analysis of Results



Gradient boosting

- Time-series split XG boost: larger RMSE, less accurate
- Random split XG boost: smaller RMSE, more accurate, little overfitting

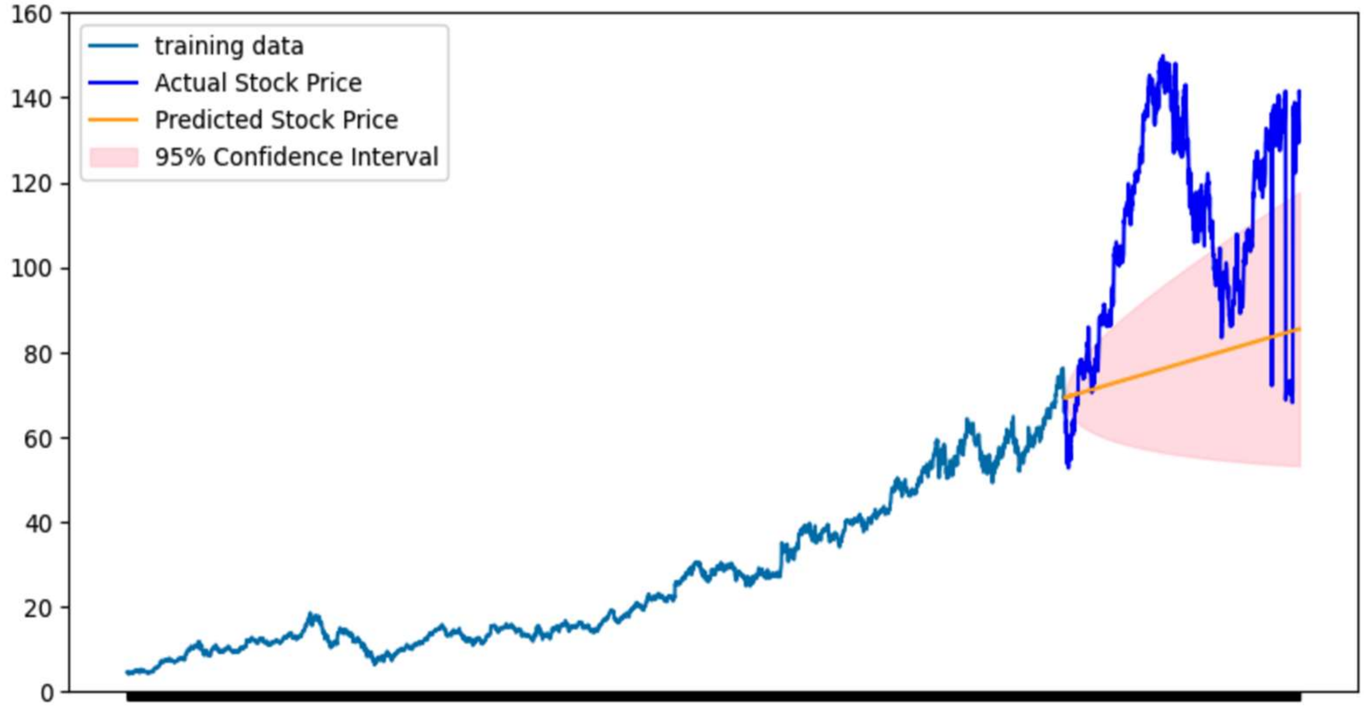


- Random split XG boost with hypertuning: even smaller RMSE

Analysis of Results

SARIMA

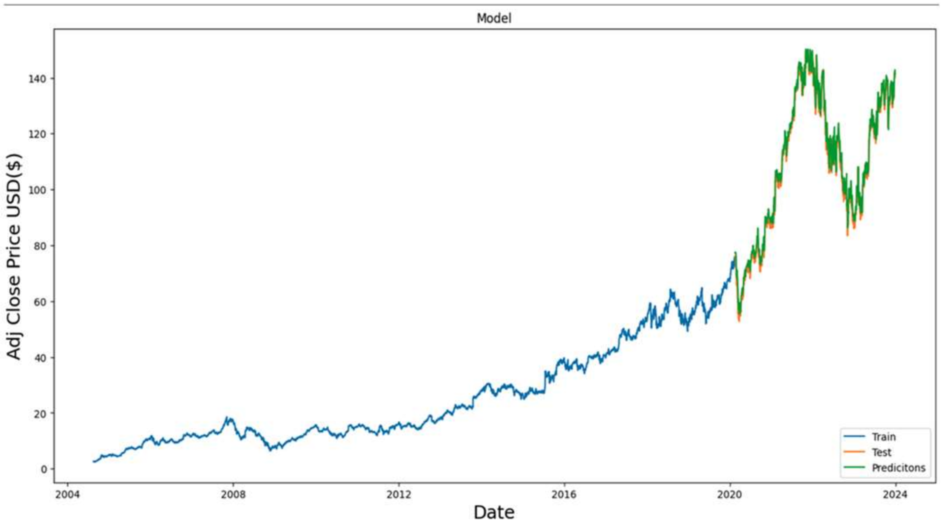
- Underfitting, does not predict the spikes in the 95% confidence interval



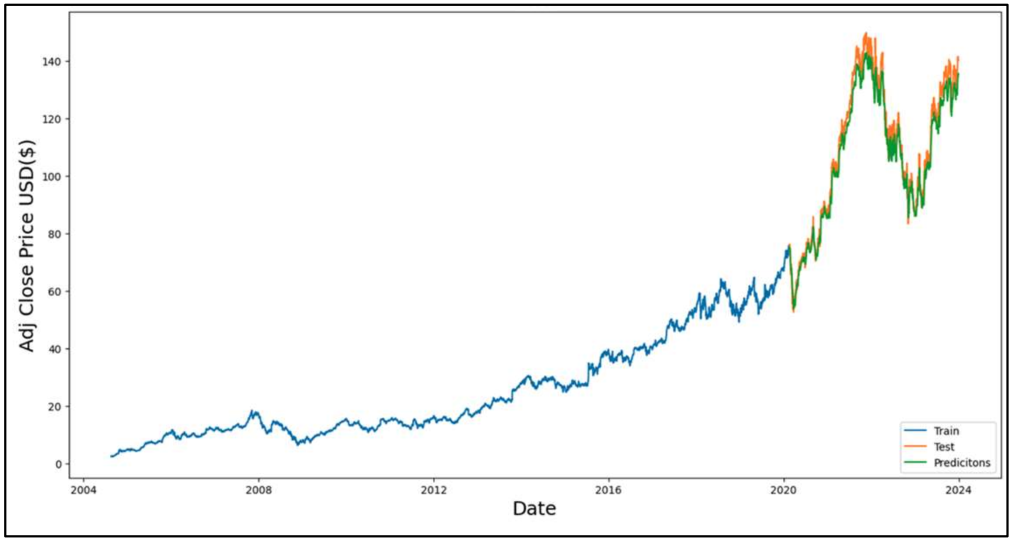
Analysis of Results

LSTM

- sliding window LSTM: accurate prediction, overfitting
- Hypertuned sliding window LSTM: less overfitting, still accurate

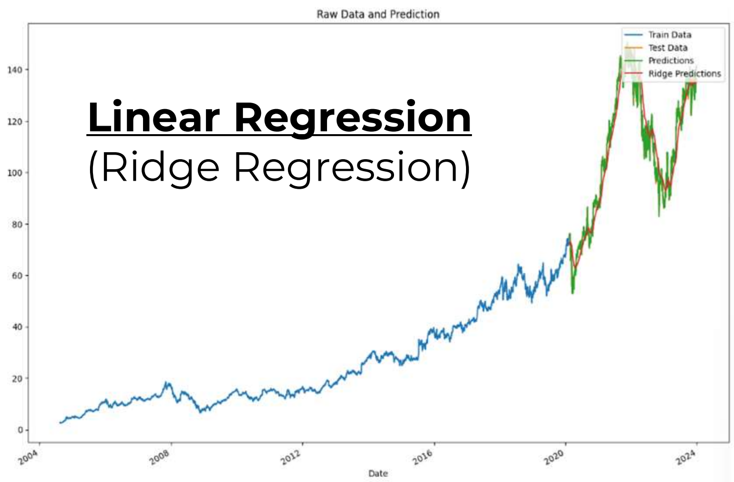


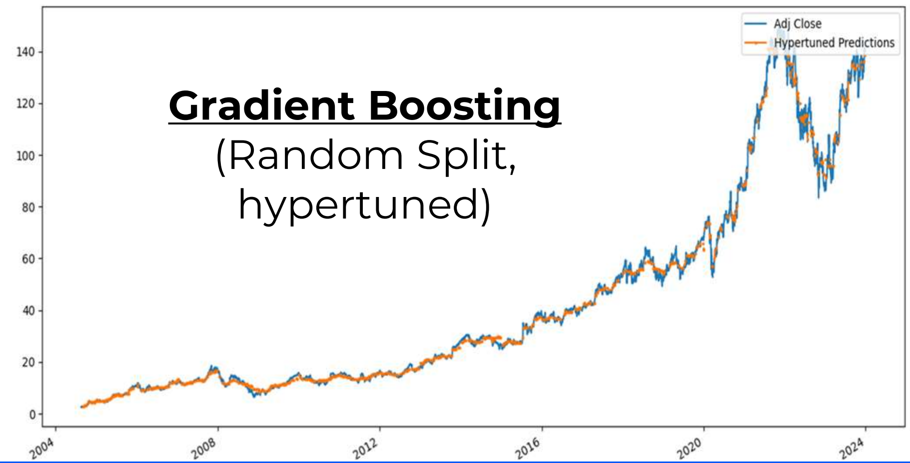
Sliding Window

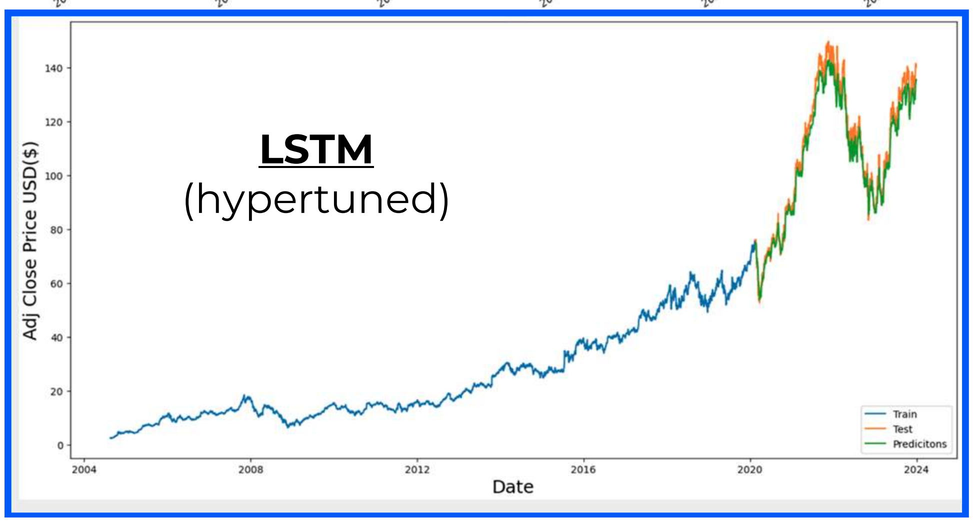


Hypertuned Sliding Window

Analysis of Results







Future Development

Expand to test models to other stocks

- Stocks in Tech Industry
- Stocks in other industries

Compare accuracy and fit of model

Automate periodic data scraping

Thanks