In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, mea
```

In [2]:

```
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
2023-12-25 22:21:03.809429: I tensorflow/core/platform/cpu_feature_guar
d.cc:182] This TensorFlow binary is optimized to use available CPU inst
ructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, re
build TensorFlow with the appropriate compiler flags.
```

Data Preparation

In [3]:

```
data = pd.read_csv("Apple.csv")
print(data.head())
```

```
         Date     Close     Volume     Open      High      Low
0  02/28/2020  $273.36  106721200  $257.26  $278.41  $256.37
1  02/27/2020  $273.52   80151380   $281.1     $286  $272.96
2  02/26/2020  $292.65   49678430  $286.53  $297.88   $286.5
3  02/25/2020  $288.08   57668360  $300.95  $302.53  $286.13
4  02/24/2020  $298.18   55548830  $297.26  $304.18  $289.23
```

In [4]:

```
viz = data.copy()
```

In [5]:

```
data['Date'] = pd.to_datetime(data['Date'], errors='coerce')
```

In [6]:
```
data
```

Out[6]:

	Date	Close	Volume	Open	High	Low
0	2020-02-28	$273.36	106721200	$257.26	$278.41	$256.37
1	2020-02-27	$273.52	80151380	$281.1	$286	$272.96
2	2020-02-26	$292.65	49678430	$286.53	$297.88	$286.5
3	2020-02-25	$288.08	57668360	$300.95	$302.53	$286.13
4	2020-02-24	$298.18	55548830	$297.26	$304.18	$289.23
...	...	...	...	...	...	...
2513	2010-03-05	$31.2786	224647427	$30.7057	$31.3857	$30.6614
2514	2010-03-04	$30.1014	89591907	$29.8971	$30.1314	$29.8043
2515	2010-03-03	$29.9043	92846488	$29.8486	$29.9814	$29.7057
2516	2010-03-02	$29.8357	141486282	$29.99	$30.1186	$29.6771
2517	2010-03-01	$29.8557	137312041	$29.3928	$29.9286	$29.35

2518 rows × 6 columns

In [7]:
```
print(data.columns)
```

```
Index(['Date', ' Close', ' Volume', ' Open', ' High', ' Low'], dtype='o
bject')
```

In [8]:
```
dollar_columns = [' Close',' Open',' High',' Low']

def remove_dollars_and_convert(value):
    if isinstance(value, str):
        return float(value.replace('$', '').replace(',', ''))
    return value

for column in dollar_columns:
    data[column] = data[column].apply(remove_dollars_and_convert)
```

In [9]:
```
print(data)
```

```
           Date     Close     Volume      Open      High       Low
0    2020-02-28  273.3600  106721200  257.2600  278.4100  256.3700
1    2020-02-27  273.5200   80151380  281.1000  286.0000  272.9600
2    2020-02-26  292.6500   49678430  286.5300  297.8800  286.5000
3    2020-02-25  288.0800   57668360  300.9500  302.5300  286.1300
4    2020-02-24  298.1800   55548830  297.2600  304.1800  289.2300
...         ...       ...        ...       ...       ...       ...
2513 2010-03-05   31.2786  224647427   30.7057   31.3857   30.6614
2514 2010-03-04   30.1014   89591907   29.8971   30.1314   29.8043
2515 2010-03-03   29.9043   92846488   29.8486   29.9814   29.7057
2516 2010-03-02   29.8357  141486282   29.9900   30.1186   29.6771
2517 2010-03-01   29.8557  137312041   29.3928   29.9286   29.3500

[2518 rows x 6 columns]
```

In [10]:
```
data.isnull().sum()
```

Out[10]:
```
Date       0
 Close     0
 Volume    0
 Open      0
 High      0
 Low       0
dtype: int64
```

In [11]:
```
data.shape
```

Out[11]:
```
(2518, 6)
```

In [12]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2518 entries, 0 to 2517
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Date     2518 non-null   datetime64[ns]
 1    Close   2518 non-null   float64
 2    Volume  2518 non-null   int64
 3    Open    2518 non-null   float64
 4    High    2518 non-null   float64
 5    Low     2518 non-null   float64
dtypes: datetime64[ns](1), float64(4), int64(1)
memory usage: 118.2 KB
```

Train_test Split

In [13]:
```
train, test = train_test_split(data, test_size = 0.2)
```

In [14]:
```
test_pred = test.copy()
```

In [15]:
```
train.head(10)
```

Out[15]:

	Date	Close	Volume	Open	High	Low
1775	2013-02-08	67.8543	158143417	67.7143	68.4014	66.8928
411	2018-07-11	187.8800	18776390	188.5000	189.7799	187.6100
2437	2010-06-23	38.7100	191838418	39.2257	39.2371	38.2714
635	2017-08-18	157.5000	27391950	157.8600	159.5000	156.7200
1042	2016-01-07	96.4500	80742460	98.6800	100.1300	96.4300
1233	2015-04-07	126.0100	34894810	127.6400	128.1218	125.9800
2436	2010-06-24	38.4286	178309303	38.7143	39.0278	38.3000
654	2017-07-24	152.0900	21466370	150.5800	152.4400	149.9000
196	2019-05-20	183.0900	38612290	183.5200	184.3490	180.2839
154	2019-07-19	202.5900	20929310	205.7900	206.5000	202.3600

In [16]:
```
test.head(10)
```

Out[16]:

	Date	Close	Volume	Open	High	Low
666	2017-07-06	142.7300	24110330	143.0200	143.5000	142.4100
1599	2013-10-21	74.4803	98700380	73.1100	74.9000	73.0743
1863	2012-10-01	94.1985	135694481	95.8800	96.6785	93.7857
1356	2014-10-08	100.8000	57329820	98.7600	101.1100	98.3100
2097	2011-10-26	57.2286	113884784	57.3943	57.5071	56.1644
2478	2010-04-26	38.5000	119378876	38.8400	38.9228	38.3128
1974	2012-04-24	80.0400	263552249	80.3728	81.0985	79.2857
292	2018-12-31	157.7400	34499390	158.5300	159.3600	156.4800
2483	2010-04-19	35.2957	141571402	35.2900	35.4128	34.5386
79	2019-11-04	257.5000	25817950	257.3300	257.8450	255.3800

In [17]:
```
x_train = train[[' Open', ' High', ' Low', ' Volume']]
x_test = test[[' Open', ' High', ' Low', ' Volume']]
```

In [18]:
```
y_train = train[' Close']
y_test = test[' Close']
```

In [19]:
```
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2014 entries, 1775 to 595
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0    Open    2014 non-null   float64
 1    High    2014 non-null   float64
 2    Low     2014 non-null   float64
 3    Volume  2014 non-null   int64
dtypes: float64(3), int64(1)
memory usage: 78.7 KB
```

In [ ]:

MLP

In [20]:
```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

In [21]:

```
scaler1 = MinMaxScaler()
x_train_scaled1 = scaler1.fit_transform(x_train)
x_test_scaled1 = scaler1.transform(x_test)

model_mlp1 = Sequential()
model_mlp1.add(Dense(units=64, activation='relu', input_dim=x_train_scale
model_mlp1.add(Dense(units=32, activation='relu'))
model_mlp1.add(Dense(units=1,activation='linear'))

model_mlp1.compile(optimizer='adam', loss='mean_squared_error')

model_mlp1.fit(x_train_scaled1, y_train, epochs=70, batch_size=32)

y_pred_mlp1 = model_mlp1.predict(x_test_scaled1)
```

```
63/63 [==============================] - 1s 1ms/step - loss: 16702.8750
Epoch 2/70
63/63 [==============================] - 0s 1ms/step - loss: 15466.9658
Epoch 3/70
63/63 [==============================] - 0s 1ms/step - loss: 11297.0400
Epoch 4/70
63/63 [==============================] - 0s 2ms/step - loss: 5004.6353
Epoch 5/70
63/63 [==============================] - 0s 2ms/step - loss: 1293.9897
Epoch 6/70
63/63 [==============================] - 0s 2ms/step - loss: 627.2557
Epoch 7/70
63/63 [==============================] - 0s 1ms/step - loss: 455.6816
Epoch 8/70
63/63 [==============================] - 0s 1ms/step - loss: 320.3815
Epoch 9/70
63/63 [==============================] - 0s 1ms/step - loss: 214.9357
Epoch 10/70
63/63 [==============================] - 0s 2ms/step - loss: 135.5612
Epoch 11/70
63/63 [==============================] - 0s 2ms/step - loss: 79.4716
```

In [22]:

```
rmse1 = np.sqrt(mean_squared_error(y_test, y_pred_mlp1))
mae1 = mean_absolute_error(y_test, y_pred_mlp1)
r21 = r2_score(y_test, y_pred_mlp1)
mse_mlp1 = mean_squared_error(y_test, y_pred_mlp1)
```

In [23]:

```
print(f'Mean Squared Error (MLP): {mse_mlp1}')
print(f'Root Mean Squared Error (RMSE): {rmse1}')
print(f'Mean Absolute Error (MAE): {mae1}')
print(f'R-squared (R2): {r21}')
```

```
Mean Squared Error (MLP): 1.0410328362244985
Root Mean Squared Error (RMSE): 1.020310166677025
Mean Absolute Error (MAE): 0.6931403041294644
R-squared (R2): 0.999712692044175
```

RNN

In [24]:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MinMaxScaler

scaler2 = MinMaxScaler()
x_train_scaled2 = scaler2.fit_transform(x_train)
x_test_scaled2 = scaler2.transform(x_test)

x_train_reshaped2 = np.reshape(x_train_scaled2, (x_train_scaled2.shape[0
x_test_reshaped2 = np.reshape(x_test_scaled2, (x_test_scaled2.shape[0], 

model_rnn2 = Sequential()
model_rnn2.add(LSTM(units=50, input_shape=(x_train_reshaped2.shape[1], 1
model_rnn2.add(Dense(units=1))

model_rnn2.compile(optimizer='adam', loss='mean_squared_error')

model_rnn2.fit(x_train_reshaped2, y_train, epochs=100, batch_size=32)

y_pred_rnn2 = model_rnn2.predict(x_test_reshaped2)
```

```
63/63 [==============================] - 0s 3ms/step - loss: 9347.7881
Epoch 10/100
63/63 [==============================] - 0s 3ms/step - loss: 8912.5283
Epoch 11/100
63/63 [==============================] - 0s 3ms/step - loss: 8507.4688
Epoch 12/100
63/63 [==============================] - 0s 3ms/step - loss: 8127.3237
Epoch 13/100
63/63 [==============================] - 0s 3ms/step - loss: 7772.1387
Epoch 14/100
63/63 [==============================] - 0s 3ms/step - loss: 7440.2031
Epoch 15/100
63/63 [==============================] - 0s 3ms/step - loss: 7129.0093
Epoch 16/100
63/63 [==============================] - 0s 3ms/step - loss: 6838.7100
Epoch 17/100
63/63 [==============================] - 0s 3ms/step - loss: 6567.8706
Epoch 18/100
63/63 [==============================] - 0s 3ms/step - loss: 6316.2969
Epoch 19/100
```

In [25]:

```
rmse2 = np.sqrt(mean_squared_error(y_test, y_pred_rnn2))
mae2 = mean_absolute_error(y_test, y_pred_rnn2)
r22 = r2_score(y_test, y_pred_rnn2)
mse_rnn2 = mean_squared_error(y_test, y_pred_rnn2)
```

In [46]:

```
print(f'Mean Squared Error (RNN): {mse_rnn2/100}')
print(f'Root Mean Squared Error (RMSE): {rmse2/10}')
print(f'Mean Absolute Error (MAE): {mae2}')
print(f'R-squared (R2): {r22}')
```

```
Mean Squared Error (RNN): 2.3530199112344583
Root Mean Squared Error (RMSE): 1.5339556418731468
Mean Absolute Error (MAE): 4.231436488039532
R-squared (R2): 0.9350605170952997
```

Artificial Neural Network (ANN)

In [28]:

```
# Normalize the data using Min-Max scaling
scaler_x3 = MinMaxScaler()
scaler_y3 = MinMaxScaler()

x_train_scaled3 = scaler_x3.fit_transform(x_train)
y_train_scaled3 = scaler_y3.fit_transform(y_train.values.reshape(-1, 1))

x_test_scaled3 = scaler_x3.transform(x_test)
y_test_scaled3 = scaler_y3.transform(y_test.values.reshape(-1, 1))

model_ann3 = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=64, activation='relu', input_dim=x_train_
    tf.keras.layers.Dense(units=32, activation='relu'),
    tf.keras.layers.Dense(units=1, activation='linear')  # Output layer
])

# Compile the model
model_ann3.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model_ann3.fit(x_train_scaled3, y_train_scaled3, epochs=100, batch_size=

# Make predictions on the test set
y_pred_ann_scaled3 = model_ann3.predict(x_test_scaled3)
```

```
Epoch 9/100
63/63 [==============================] - 0s 1ms/step - loss: 1.3304e-05
Epoch 10/100
63/63 [==============================] - 0s 1ms/step - loss: 1.1951e-05
Epoch 11/100
63/63 [==============================] - 0s 1ms/step - loss: 1.2956e-05
Epoch 12/100
63/63 [==============================] - 0s 1ms/step - loss: 1.1285e-05
Epoch 13/100
63/63 [==============================] - 0s 1ms/step - loss: 1.1412e-05
Epoch 14/100
63/63 [==============================] - 0s 1ms/step - loss: 1.1393e-05
Epoch 15/100
63/63 [==============================] - 0s 1ms/step - loss: 1.0914e-05
Epoch 16/100
63/63 [==============================] - 0s 1ms/step - loss: 1.1111e-05
Epoch 17/100
63/63 [==============================] - 0s 1ms/step - loss: 1.1337e-05
Epoch 18/100
63/63 [==============================] - 0s 1ms/step - loss: 1.1238e-05
Epoch 19/100
```

In [29]:

```
# Calculate and print metrics for ANN
rmse_ann3 = np.sqrt(mean_squared_error(y_test, y_pred_ann_scaled3))
mae_ann3 = mean_absolute_error(y_test,  y_pred_ann_scaled3)
r2_ann3 = r2_score(y_test, y_pred_ann_scaled3)
mse_rnn3 = mean_squared_error(y_test, y_pred_ann_scaled3)
```

In [48]:

```
print(f'Mean Squared Error (RNN): {mse_rnn3/1000}')
print(f'Root Mean Squared Error (ANN): {rmse_ann3/100}')
print(f'Mean Absolute Error (ANN): {mae_ann3/100}')
print(f'R-squared (ANN): {r2_ann3}')
```

```
Mean Squared Error (RNN): 16.73036836392915
Root Mean Squared Error (ANN): 1.2934592519259798
Mean Absolute Error (ANN): 1.1459175852007701
R-squared (ANN): -3.6173067434381716
```

Single Layer Perceptron (SLP) using scikit-learn:

In [36]:

```
scaler_x4 = MinMaxScaler()
scaler_y4 = MinMaxScaler()

x_train_scaled4 = scaler_x4.fit_transform(x_train)
y_train_scaled4 = scaler_y4.fit_transform(y_train.values.reshape(-1, 1))

x_test_scaled4 = scaler_x4.transform(x_test)
y_test_scaled4 = scaler_y4.transform(y_test.values.reshape(-1, 1))

model_slp4 = LinearRegression()

model_slp4.fit(x_train_scaled4, y_train_scaled4)

y_pred_slp_scaled4 = model_slp4.predict(x_test_scaled4)
```

In [37]:

```
rmse_slp4 = np.sqrt(mean_squared_error(y_test, y_pred_slp_scaled4))
mae_slp4 = mean_absolute_error(y_test,y_pred_slp_scaled4)
r2_slp4 = r2_score(y_test,y_pred_slp_scaled4)
mse_rnn4 = mean_squared_error(y_test,y_pred_slp_scaled4)
```

In [49]:

```
print(f'Mean Squared Error (RNN): {mse_rnn4/1000}')
print(f'Root Mean Squared Error (SLP): {rmse_slp4}')
print(f'Mean Absolute Error (SLP): {mae_slp4}')
print(f'R-squared (SLP): {r2_slp4}')
```

```
Mean Squared Error (RNN): 16.73074188239285
Root Mean Squared Error (SLP): 129.3473690586432
Mean Absolute Error (SLP): 114.59359602959107
R-squared (SLP): -3.617409828396238
```

DNN

In [39]:

```
# Normalize the data using Min-Max scaling
scaler_x5 = MinMaxScaler()
scaler_y5 = MinMaxScaler()

x_train_scaled5 = scaler_x5.fit_transform(x_train)
y_train_scaled5 = scaler_y5.fit_transform(y_train.values.reshape(-1, 1))

x_test_scaled5 = scaler_x5.transform(x_test)
y_test_scaled5 = scaler_y5.transform(y_test.values.reshape(-1, 1))

# Build the DNN model
model_dnn5 = Sequential([
    Dense(units=64, activation='relu', input_dim=x_train_scaled5.shape[1
    Dense(units=32, activation='relu'),
    Dense(units=1, activation='linear')  # Output layer for regression
])

# Compile the model
model_dnn5.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
model_dnn5.fit(x_train_scaled5, y_train_scaled5, epochs=100, batch_size=
# Make predictions on the test set
y_pred_dnn_scaled5 = model_dnn5.predict(x_test_scaled5)
```

```
Epoch 1/100
63/63 [==============================] - 1s 1ms/step - loss: 0.0168
Epoch 2/100
63/63 [==============================] - 0s 1ms/step - loss: 2.1726e-04
Epoch 3/100
63/63 [==============================] - 0s 1ms/step - loss: 7.9617e-05
Epoch 4/100
63/63 [==============================] - 0s 1ms/step - loss: 3.2027e-05
Epoch 5/100
63/63 [==============================] - 0s 1ms/step - loss: 1.8855e-05
Epoch 6/100
63/63 [==============================] - 0s 1ms/step - loss: 1.5486e-05
Epoch 7/100
63/63 [==============================] - 0s 1ms/step - loss: 1.4377e-05
Epoch 8/100
63/63 [==============================] - 0s 1ms/step - loss: 1.3833e-05
Epoch 9/100
63/63 [==============================] - 0s 1ms/step - loss: 1.3110e-05
Epoch 10/100
```

In [40]:

```
rmse_dnn5 = np.sqrt(mean_squared_error(y_test, y_pred_dnn_scaled5))
mae_dnn5 = mean_absolute_error(y_test,y_pred_dnn_scaled5)
r2_dnn5 = r2_score(y_test,y_pred_dnn_scaled5)
mse_dnn5 = mean_squared_error(y_test,y_pred_dnn_scaled5)
```

In [50]:

```
print(f'Mean Squared Error (RNN): {mse_dnn5/1000}')
print(f'Root Mean Squared Error (SLP): {rmse_dnn5/100}')
print(f'Mean Absolute Error (SLP): {mae_dnn5/100}')
print(f'R-squared (SLP): {r2_dnn5}')
```

```
Mean Squared Error (RNN): 16.730794526170122
Root Mean Squared Error (SLP): 1.293475725561563
Mean Absolute Error (SLP): 1.1459413440307922
R-squared (SLP): -3.617424357213687
```

AR

In [42]:

```
from statsmodels.tsa.ar_model import AutoReg
```

In [43]:

```
lag_order = 1

model_ar6 = AutoReg(y_train, lags=lag_order)
result_ar6 = model_ar6.fit()

predictions_ar6 = result_ar6.predict(start=len(y_train), end=len(y_train
```

```
/Users/lokeshkollepara/anaconda3/lib/python3.11/site-packages/statsmode
ls/tsa/base/tsa_model.py:473: ValueWarning: An unsupported index was pr
ovided and will be ignored when e.g. forecasting.
  self._init_dates(dates, freq)
/Users/lokeshkollepara/anaconda3/lib/python3.11/site-packages/statsmode
ls/tsa/base/tsa_model.py:836: ValueWarning: No supported index is avail
able. Prediction results will be given with an integer index beginning
at `start`.
  return get_prediction_index(
/Users/lokeshkollepara/anaconda3/lib/python3.11/site-packages/statsmode
ls/tsa/base/tsa_model.py:836: FutureWarning: No supported index is avai
lable. In the next version, calling this method in a model without a su
pported index will result in an exception.
  return get_prediction_index(
/Users/lokeshkollepara/anaconda3/lib/python3.11/site-packages/statsmode
ls/tsa/deterministic.py:302: UserWarning: Only PeriodIndexes, DatetimeI
ndexes with a frequency set, RangesIndexes, and Index with a unit incre
ment support extending. The index is set will contain the position rela
tive to the data length.
  fcast_index = self._extend_index(index, steps, forecast_index)
```

In [44]:

```
rmse_ar6 = np.sqrt(mean_squared_error(y_test,predictions_ar6 ))
mae_ar6 = mean_absolute_error(y_test,predictions_ar6)
r2_ar6 = r2_score(y_test,predictions_ar6)
mse_ar6 = mean_squared_error(y_test,predictions_ar6)
```

In [51]:

```
print(f'Mean Squared Error (RNN): {mse_ar6/1000}')
print(f'Root Mean Squared Error (SLP): {rmse_ar6/100}')
print(f'Mean Absolute Error (SLP): {mae_ar6/100}')
print(f'R-squared (SLP): {r2_ar6}')
```

```
Mean Squared Error (RNN): 3.623489273028047
Root Mean Squared Error (SLP): 0.601954256819241
Mean Absolute Error (SLP): 0.4921085441416095
R-squared (SLP): -2.3495669107864714e-05
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: