Week_8_Project

Arun Prasad Patel

Nepal Stock Exchange (NEPSE)-BasedProjects NEPSE Stock Price Prediction

Objective: Predict thefutureprices of stocks listedonNEPSE.

Data: Historical stockprices (availablefromNEPSEwebsiteor APIs).

Tasks: Normalizeprices, handlemissingdata, anduseLinear Regressionor RandomForest Regressor topredict stockprices

Data was collectedfromnepse alpha site

- Dailynepsepricedatawascollectedfrom2019Decto2024Dec
- It containsdayHighprice, Lowprice, Openprice, Closeprice, date, changepercentage andvolume

Four models are usedtopredict the future closingprice

- MovingAverage
- Linear RegressionModel
- MultipleRegressionModel
- RandomForest Regressor Model

In [55]:
```
import pandasas pd
import numpyas np
import matplotlib.pyplotas plt
from sklearn.preprocessingimport StandardScaler
from sklearn.model_selectionimport train_test_split
from sklearn.metricsimport mean_squared_erroras mse
from sklearn.metricsimport r2_score
from sklearn.linear_modelimport LinearRegression
data=
pd.read_csv("data.csv")
data.head(
)
```

Out[55]:

	Symbol	Date	Open	High	Low	Close	Percent Change	Volume
0	NEPSE	2024-12-05	2752.38	2764.29	2717.34	2734.93	-0.57%	8,369,687,402.72
1	NEPSE	2024-12-04	2795.98	2795.78	2747.65	2750.87	-0.89%	9,132,468,906.77
2	NEPSE	2024-12-03	2772.74	2793.03	2751.31	2775.85	0.64%	8,521,219,299.67
3	NEPSE	2024-12-02	2749.27	2773.47	2746.47	2758.04	0.72%	10,541,746,564.07
4	NEPSE	2024-12-01	2762.30	2764.85	2707.94	2738.06	-0.36%	8,832,225,892.82

Exploratory Data Analysis

- CandleStickpatternof recent 100days

In [56]:
```
import mplfinance as mpf
data=
data.head(100)
data['Date']=
pd.to_datetime(data['Date'])
data['Volume']= data['Volume'].str.replace(',',
'').astype(float)
data.set_index('Date',inplace=True)# Set 'Date' as index
data= data[['Open','High','Low','Close','Volume']]
# Plot candlestick chart using mplfinance
fig, axes=
mpf.plot(
    data
    ,
    type='candle'
    ,
    volume=True
    ,
    title="NEPSE Candlestick Chart of 100 days",
    style='yahoo'
    ,
    ylabel="Price"
    ,
    ylabel_lower="Volume"
    ,
    returnfig=True
    ,
    warn_too_much_data=100
)

# Invert the x-axis (dates)
axes[0].invert_xaxis()# Invert the x-axis of the first axis (candlestick chart)

# Show the plot
plt.show(
)
```

NEPSE Candlestick Chart of 100 days

Changingdateintodatetimeformat

In [57]:
```
data= pd. read_csv( 'data.csv')
data[ "Date"] =
pd. to_datetime( data[ 'Date'] )
data. info(
)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1128 entries, 0 to 1127
Data columns (total 8 columns):
 # Column           Non-Null Count Dtype
- - - - - - - - -   - - - - - - - - - - - - - - - - - - -
 0 Symbol           1128 non-null object
 1 Date             1128 non-null datetime64[ns]
 2 Open             1128 non-null float64
 3 High             1128 non-null float64
 4 Low              1128 non-null float64
 5 Close            1128 non-null float64
 6 Percent Change 1128 non-null object
 7 Volume           1128 non-null object
dtypes: datetime64[ns](1), float64(4), object(3)
memory usage: 70.6+ KB
```

In [58]:
```
data= pd. DataFrame( data)
data. describe(
)
```

Out[58]:

	Date	Open	High	Low	Close
count	1128	1128.000000	1128.000000	1128.000000	1128.000000
mean	2022-07-02 20:06:22.978723584	2165.339071	2184.024947	2142.533327	2161.999140
min	2019-12-08 00:00:00	1131.920000	1135.370000	1125.810000	1135.370000
25%	2021-05-02 18:00:00	1908.567500	1926.862500	1888.872500	1907.935000
50%	2022-07-03 12:00:00	2078.000000	2093.210000	2059.245000	2072.005000
75%	2023-09-05 12:00:00	2596.872500	2620.352500	2548.730000	2585.002500
max	2024-12-05 00:00:00	3208.530800	3227.110000	3178.250000	3198.190000
std	NaN	473.228330	476.801759	465.756438	470.101723

MovingAverage

- It isatechnical toolsthat investor usedtodeterminethetrendandpriceof stocksin future.
- It usespreviousdaysaveragepricetodeterminethepriceof next days.
- A7dayswindowsizeistakenheretodeterminethenext dayprice.

In [59]:

```
import plotly.graph_objects as go
window_size = 7
data['Moving_Avg'] = data['Close'].rolling(window=window_size).mean()
# Create traces for the plot
trace_original = go.Scatter(
    x=data['Date'],
    y=data['Close'],
    mode='lines',
    name='Original'
)
trace_moving_avg = go.Scatter(
    x=data['Date'],
    y=data['Moving_Avg'],
    mode='lines',
    name=f'{window_size}-Day Moving Average'
)
# Layout for the plot
layout = go.Layout(
    title=f'{window_size}-Day Moving Average of Closing Prices',
    xaxis=dict(title='Date'),
    yaxis=dict(title='Close Price')
)
```

```
fig= go.Figure(data=[trace_original, trace_moving_avg],
layout=layout)
fig.show(
)
valid_indices=
~np.isnan(data['Moving_Avg'])
actual= data.loc[valid_indices,
'Close']
moving_avg= data.loc[valid_indices,
'Moving_Avg'].values
```

```
# Calculate mean squared error
mse= mse(actual, moving_avg)
print(f'Mean Squared Error of the Moving Average: {mse}')
rmse_value=
np.sqrt(mse)
r2_value= r2_score(actual, moving_avg)


print(f"Root Mean Squared Error (RMSE): {rmse_value}")
print(f"R-squared (R²):
{r2_value}")
```

```
Mean Squared Error of the Moving Average: 2530.8692685143988
Root Mean Squared Error (RMSE): 50.30774561152983
R-squared (R²): 0.9885020365752958
```

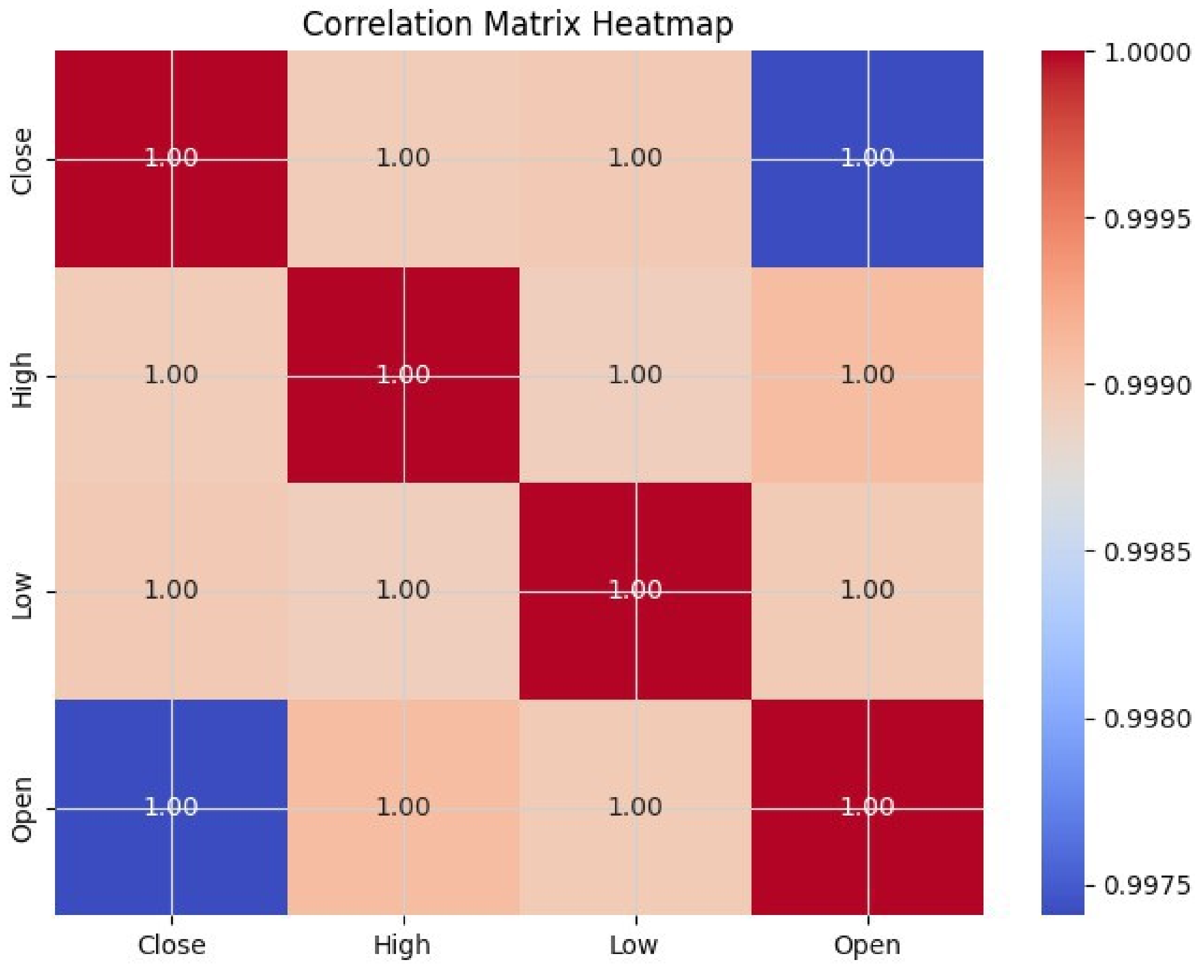
In [60]:
```
total_number_of_days=(data.Date.max() -
data.Date.min()).days
print("total number of days=", total_number_of_days)
```

```
total number of days= 1824
```

Here, Correlationbetweenclosingprice, openingprice, high price andlowprice is mappedusedheat map

In [61]:
```
import seabornas sns
from sklearn.preprocessingimport MinMaxScaler
scaler= MinMaxScaler()
scaler.fit(data[['Close']
])
data['Close']=
scaler.transform(data[['Close']])
scaler.fit(data[['High']
])
data['High']=
scaler.transform(data[['High']])
scaler.fit(data[['Open']
])
data['Open']=
scaler.transform(data[['Open']])

print(data[["Close",'High',"Low",'Open']].corr(
))
plt.figure(figsize=(8,6))# Set figure
size
sns.heatmap(data[["Close",'High',"Low",'Open']].corr(),annot=True,
cmap='coolwarm'
plt.title("Correlation Matrix
Heatmap")
plt.show(
)
```

```
           Close      High       Low      Open
Close 1.000000 0.998953 0.998977 0.997406
High 0.998953 1.000000 0.998919 0.999096
Low 0.998977 0.998919 1.000000 0.998959
Open 0.997406 0.999096 0.998959 1.000000
```



This shows that there is a highcorrelationbetweenthe high, low, close andopenprice.

Linear Regression Model

- Linear Regressionmodel hasequationy=mx+c
  - y: Thepredictedvalue(thescaled'Close' values).
  - x: Theindependent variable(thedayindex).
  - m: slopeof thelinear model
  - c: theintercept
- Dataisscaledbetween0and1toimprovethemodel performance
- Split intotraining(80%) andtesting(20%) sets.
- Alinear regressionmodel istrainedonthetrainingdata.
- Predictionsaremadeandcomparedagainst theactual test data.
- Performancemetrics(MSE, RMSE, R^2) evaluatethemodel.
- Avisualizationcontraststheactual vs. predictedvaluesfor trainingdata.

In [62]:
```
from sklearn.metricsimport mean_squared_erroras mse

data=pd. read_csv('data.csv
')
```

```
scaler= MinMaxScaler( )
scaler.fit( data[ [ 'Close']
] )
data[ 'Close'] =
scaler.transform( data[ [ 'Close'] ] )


X= np.array( data.index).reshape( - 1,
1)
y=
data[ 'Close']
X_train, X_test, y_train, y_test= train_test_split(
     X, y, test_size=0.2, shuffle=False,
     random_state=42
)
model= LinearRegression( )
model.fit( X_train,
y_train)
y_pred=
model.predict( X_test)
# Trace for actual data points (scatter plot)
trace0= go.Scatter(
     x=X_train.T[ 0
     ] ,
     y=y_train
     ,
     mode='markers'
     ,
     name="actual
     "
)
# Trace for predicted data points (line plot)
trace1= go.Scatter(
     x=X_train.T[ 0
     ] ,
     y=model.predict( X_train) .
     T,
     mode='lines'
     ,
     name="predicted
     "
)
# Combine both traces into a list
predicted_data= [trace0, trace1]
layout= go.Layout(
     title="Actual vs Predicted Data",
     xaxis=dict( title='Day',
     autorange='reversed') ,
     yaxis=dict( title='Target
•    ')
)
plot= go.Figure( data=predicted_data,
layout=layout)
plot.show(
)
mse_value= mse( y_test, y_pred)
rmse_value=
np.sqrt( mse_value)
r2_value= r2_score( y_test, y_pred)
```

```
print(f"Mean Squared Error (MSE): {mse_value}")
print(f"Root Mean Squared Error (RMSE): {rmse_value}")
print(f"R-squared (R²):
{r2_value}")
```

Mean Squared Error (MSE): 0.3109412041107992
Root Mean Squared Error (RMSE): 0.5576210219412456
R-squared (R²): -9.649145799000943

Thereisaalot of error inthismodel makingit irrelevant for predictingtheprice.

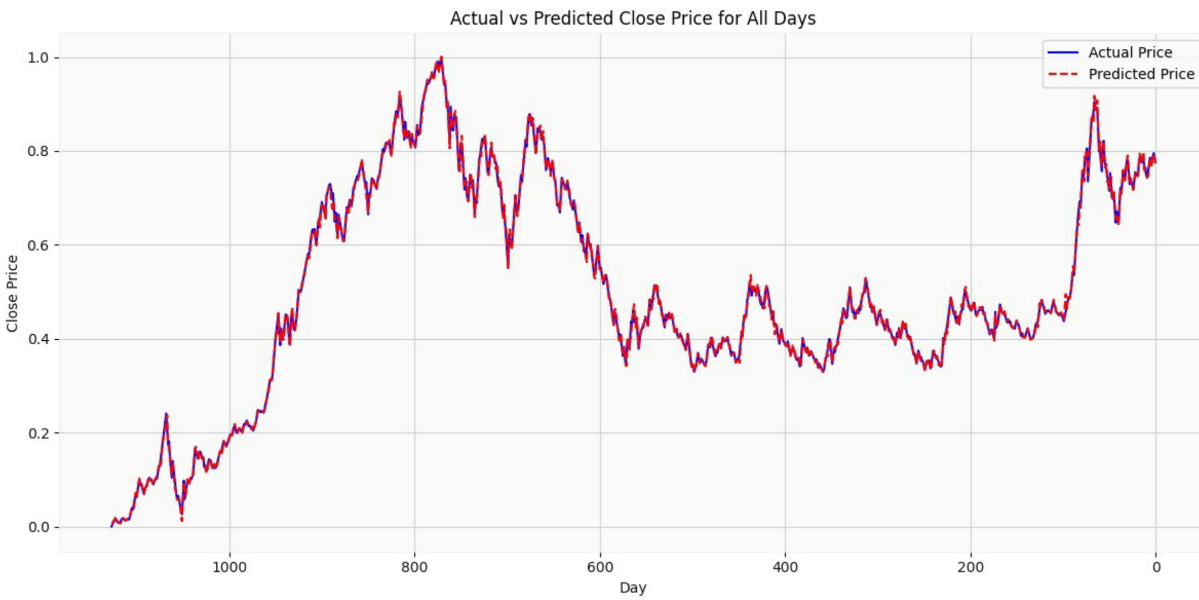
Multiple Linear Regression model

It usesmorethanoneindependant variablesfor predictingadependant variables.

- It usesHigh, low, openpricetopredict thecloseprice.
- Themodel split dataset intotrain(80%) andtest (20%).

In [63]:
```python
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import r2_score


scaler= MinMaxScaler()
scaler.fit(data[['Close']
])
data['Close']=
scaler.transform(data[['Close']])
scaler.fit(data[['High']
])
data['High']=
scaler.transform(data[['High']])
scaler.fit(data[['Open']
])
data['Open']=
scaler.transform(data[['Open']])
X= data[['Open','High','Low']]# Independent variables
y= data['Close']# Dependent variable

X_train,X_test,y_train,y_test= train_test_split(X,y,test_size=0.2,random_sta
model= LinearRegression()
model.fit(X_train,
y_train)
y_pred=
model.predict(X_test)


mse_value= mse(y_test,y_pred)
r2_value= r2_score(y_test,y_pred)
model.fit(X,
y)
y_pred_full=
model.predict(X)


plt.figure(figsize=(12,
6))
plt.plot(data.index,y,color='blue',label='Actual
Price')
plt.plot(data.index,y_pred_full,color='red',linestyle='--',label='Predicted
Pri
plt.xlabel('Day'
)
plt.gca().invert_xaxis
()
plt.ylabel('Close
Price')
plt.title('Actual vs Predicted Close Price for All Days')
plt.legend(
)
plt.tight_layout(
)
```

```
plt.show()


print("Model Coefficients:", model.coef_)
print("Intercept:", model.intercept_)
print("Mean Squared Error (MSE):", mse_value)
print("R-squared:", r2_value)
```



Model Coefficients: [-8.06393657e-01 9.42776311e-01 4.22032540e-04]
Intercept: -0.47788042456870206
Mean Squared Error (MSE): 3.206797429870102e-05
R-squared: 0.9993654540440245

RandomForest Regressor

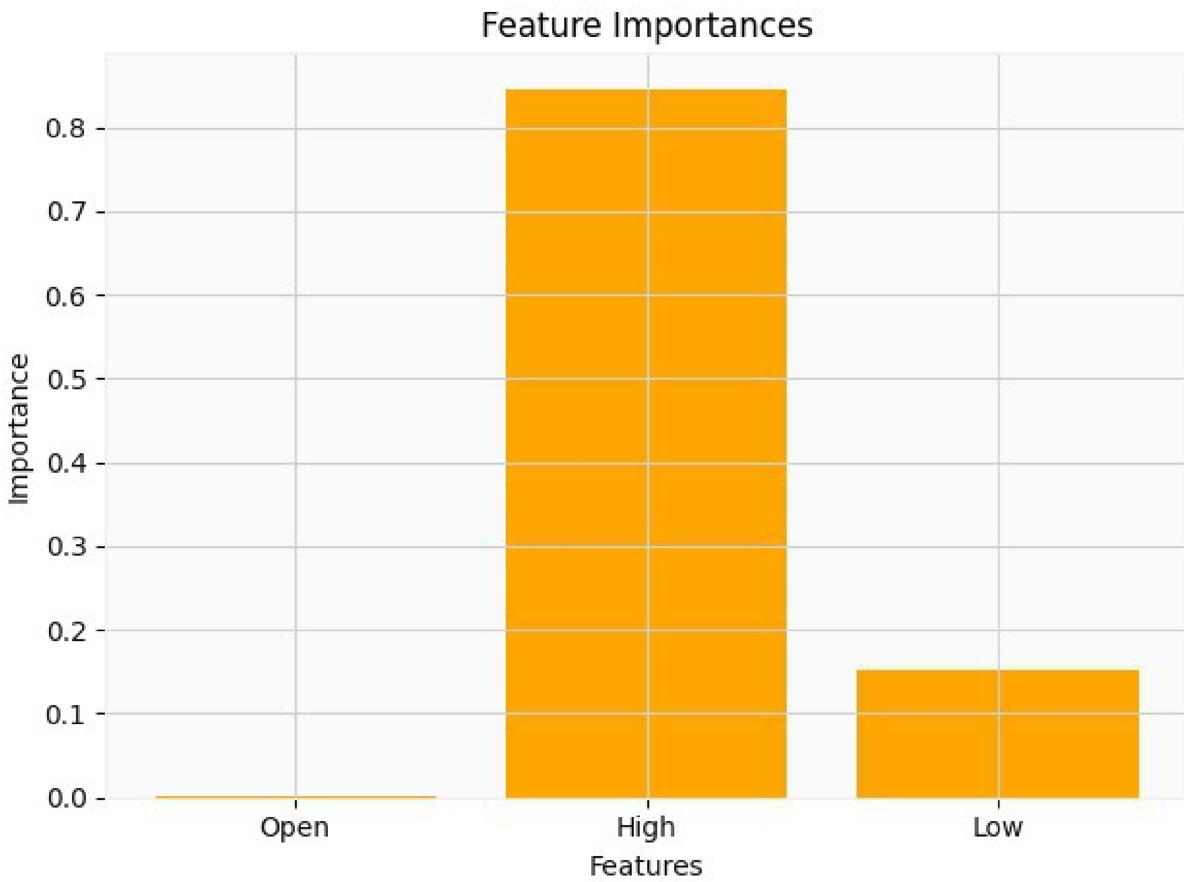
- RandomForest Regressor usesmulitpledecisiontodecidetheoutcomeandchosesthe best decisiontree
- Adataset isdividedintotrain(80%) andtest(20%) data.

In [64]:
```
from sklearn.ensembleimport RandomForestRegressor
from sklearn.model_selectionimport train_test_split, TimeSeriesSplit
features= ['Open', 'High', 'Low']
target= 'Close'
scaler= MinMaxScaler( )
data[ features] =
scaler. fit_transform( data[ features] )
data[ target] =
scaler. fit_transform( data[ [ target] ] )
X=
data[ features]
y=
data[ target]

X_train, X_test, y_train, y_test= train_test_split(
     X, y, test_size=0.2, shuffle=False,
     random_state=42
)

rf_model= RandomForestRegressor( n_estimators=100, max_depth=10, random_state=42)
rf_model. fit ( X_train,
y_train)

# Predict on test data
y_pred_rf=
rf_model. predict( X_test)

# Evaluate the model
mse_value= mse( y_test, y_pred_rf)
```

```
r2_value= r2_score(y_test,y_pred_rf)

# Print evaluation metrics
```

```
print("Mean Squared Error (MSE):", mse_value)
print("R-squared (R²):", r2_value)

# Predict on full data for visualization
y_pred_full=
rf_model.predict(X)

# Plot actual vs predicted prices
plt.figure(figsize=(12,
6))
plt.plot(data.index, y, label="Actual Price",
color='blue')
plt.plot(data.index, y_pred_full, label="Predicted Price", color='red',
linestyle='
plt.axvline(X_train.index[-1], color='green', linestyle='--', label='Train-Test
Spl
plt.xlabel('Day'
)
plt.gca().invert_xaxis
()
plt.ylabel('Close
Price')
plt.title('Actual vs Predicted Close
Price')
plt.legend(
)
plt.tight_layout(
)
plt.show(
)

# Feature importance visualization
feature_importances= rf_model.feature_importances_
plt.bar(features, feature_importances,
color='orange')
plt.title('Feature
Importances')
plt.xlabel('Features'
)
plt.ylabel('Importance'
)
plt.tight_layout(
)
plt.show(
)
```


```
Mean Squared Error (MSE): 0.035239214499173326
R-squared (R²): -0.2068761813575659
```



Conclusion

This Python code serves the purpose of exploring data and developing models through the application of different machine learning approaches. It starts by getting data from a CSV file and preparing it. Exploratory Data Analysis (EDA) involves making candlestick charts and looking at correlation matrices. Four models Moving Average, Linear Regression, Multiple Linear Regression, and Random Forest Regressor are used to predict stock prices. The Moving Average method makes data clearer by averaging it over a 7-day period. Linear Regression and Multiple Regression models help predict prices. We use measures like MSE, RMSE, and R^2 to check how good the predictions are. The Random Forest Regressor is strong because it uses decision trees and helps us understand which features are important. The study looks at how stock prices are related to each other and checks how well each model can predict future prices. However, mistakes in some models show that they may not be very reliable for making accurate predictions. The script shows a complete analysis of stock prices for NEPSE. It focuses on creating models and visualizations to understand market trends and make predictions.