ujjwal-singh-project-3

June 12, 2024

Meta Stock Price Prediction

0.0.1 Institution Name: Vigor Council

0.0.2 Guidance Under: Dr. B.P. Sharma

0.0.3 Intern Name: Ujjwal Singh

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
df=pd.read_csv("META.csv")
```

[3]:
```
df.head()
```

[3]:
```
         Date        Open        High         Low       Close   Adj Close  \
0  28/10/2021  312.989990  325.519989  308.109985  316.920013  316.584106
1  29/10/2021  320.190002  326.000000  319.600006  323.570007  323.227051
2  01/11/2021  326.040009  333.450012  326.000000  329.980011  329.630280
3  02/11/2021  331.380005  334.790009  323.799988  328.079987  327.732269
4  03/11/2021  327.489990  332.149994  323.200012  331.619995  331.268524

     Volume
0  50806800
1  37059400
2  31518900
3  28353000
4  20786500
```

[4]:
```
df.columns
```

[4]:
```
Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'],
dtype='object')
```

[5]:
```
df.shape
```

[5]:
```
(633, 7)
```

[6]:
```
df.isnull().sum()
```

[6]:
```
Date         0
Open         0
High         0
Low          0
Close        0
Adj Close    0
Volume       0
dtype: int64
```

[7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 633 entries, 0 to 632
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       633 non-null    object
 1   Open       633 non-null    float64
 2   High       633 non-null    float64
 3   Low        633 non-null    float64
 4   Close      633 non-null    float64
 5   Adj Close  633 non-null    float64
 6   Volume     633 non-null    int64
dtypes: float64(5), int64(1), object(1)
memory usage: 34.7+ KB
```

[8]:
```
df['Date']=pd.to_datetime(df["Date"], format='%d/%m/%Y')
```

[9]:
```
Date_column=df.columns[0]
```

[10]:
```
df.set_index(Date_column, inplace=True)
```

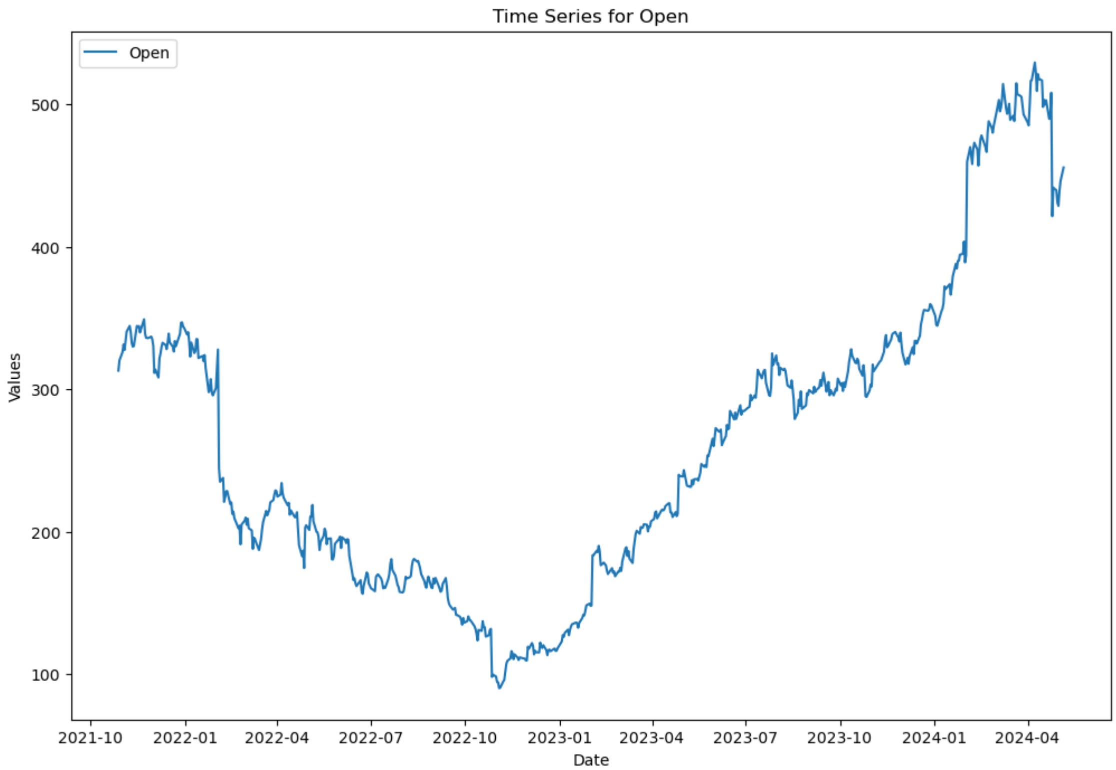
[11]:
```
df.head()
```

[11]:
```
                  Open        High         Low       Close   Adj Close  \
Date
2021-10-28  312.989990  325.519989  308.109985  316.920013  316.584106
2021-10-29  320.190002  326.000000  319.600006  323.570007  323.227051
2021-11-01  326.040009  333.450012  326.000000  329.980011  329.630280
2021-11-02  331.380005  334.790009  323.799988  328.079987  327.732269
2021-11-03  327.489990  332.149994  323.200012  331.619995  331.268524

              Volume
Date
2021-10-28  50806800
```

```
2021-10-29   37059400
2021-11-01   31518900
2021-11-02   28353000
2021-11-03   20786500
```

[12]:

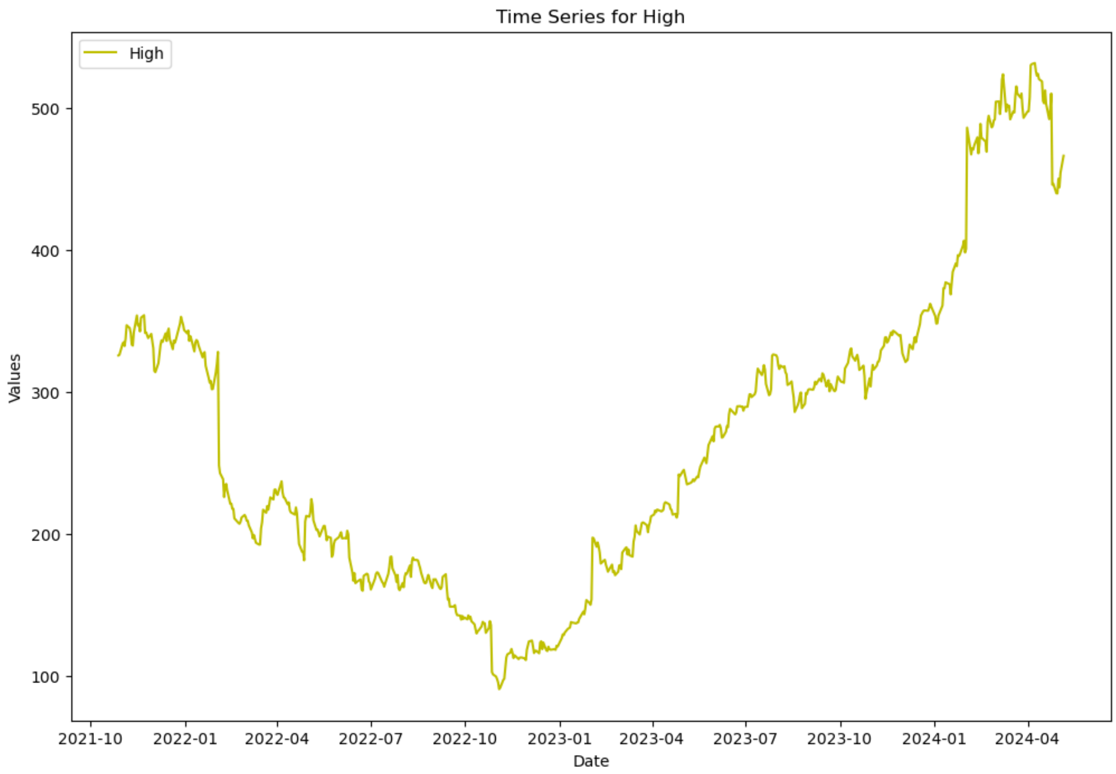
```
plt.figure(figsize=(12,8))
column="Open"
plt.plot(df.index, df[column], label=column)

plt.xlabel('Date')
plt.ylabel('Values')
plt.title(f'Time Series for {column}')
plt.legend()
plt.show()
```



[13]:

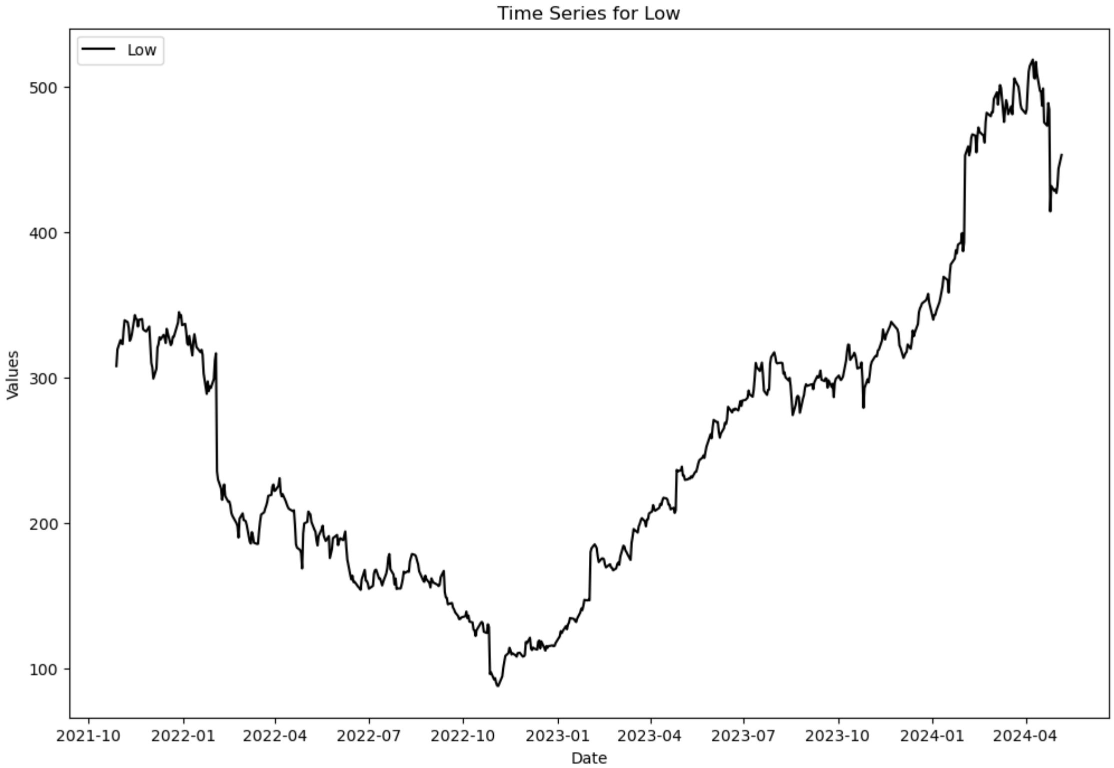
```
plt.figure(figsize=(12,8))
column="High"
plt.plot(df.index, df[column], label=column,color='y')

plt.xlabel('Date')
plt.ylabel('Values')
plt.title(f'Time Series for {column}')
```

```
plt.legend()
plt.show()
```



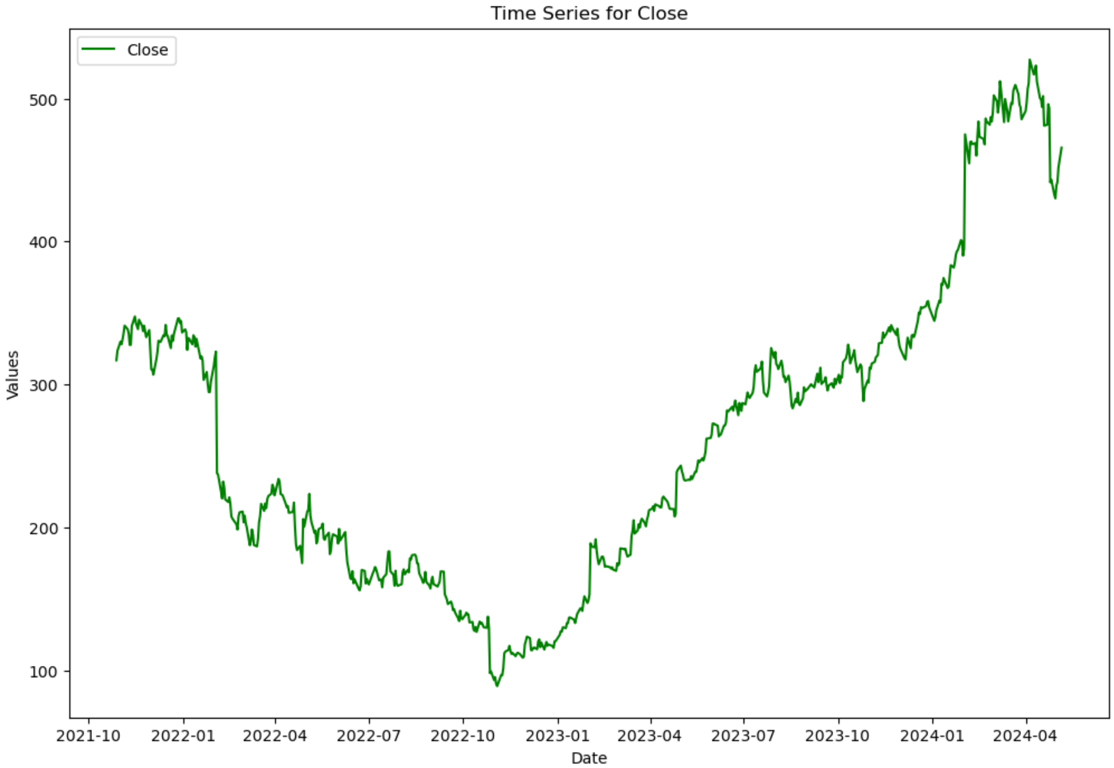
[14]:

```
plt.figure(figsize=(12,8))
column="Low"
plt.plot(df.index, df[column], label=column, color='k')

plt.xlabel('Date')
plt.ylabel('Values')
plt.title(f'Time Series for {column}')
plt.legend()
plt.show()
```

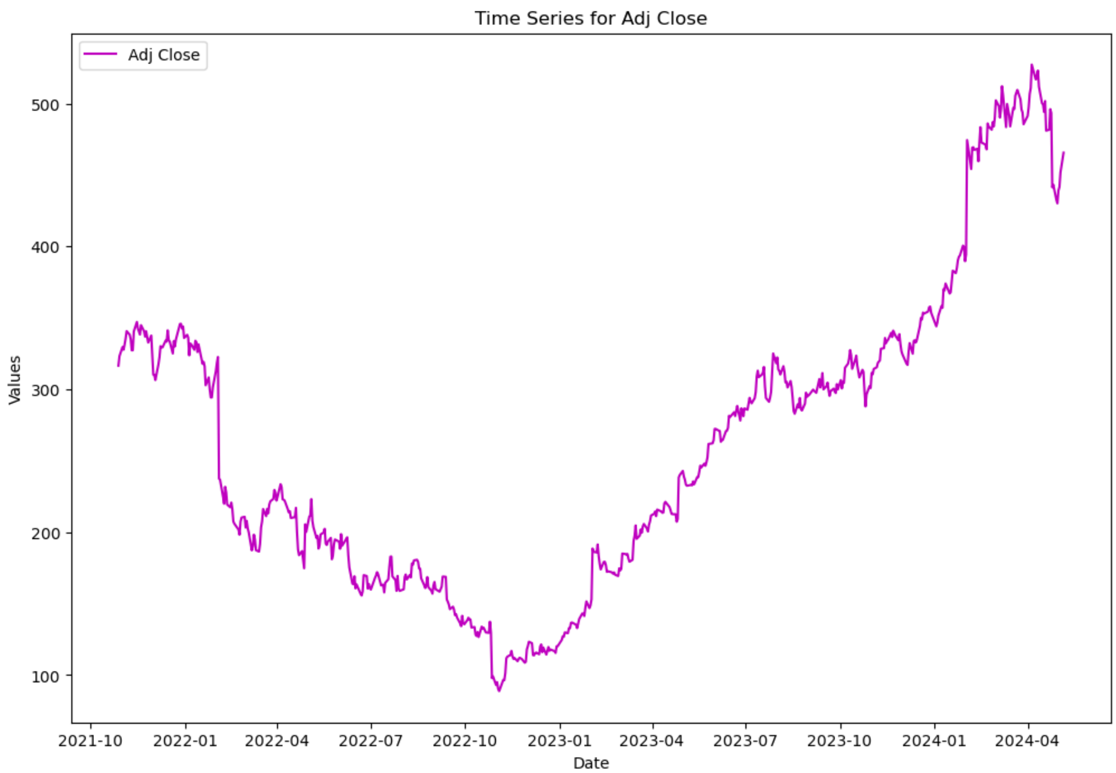


[15]:

```
plt.figure(figsize=(12,8))
column="Close"
plt.plot(df.index, df[column], label=column,color='g')

plt.xlabel('Date')
plt.ylabel('Values')
plt.title(f'Time Series for {column}')
plt.legend()
plt.show()
```



[16]:
```
plt.figure(figsize=(12,8))
column="Adj Close"
plt.plot(df.index, df[column], label=column,color='m')

plt.xlabel('Date')
plt.ylabel('Values')
plt.title(f'Time Series for {column}')
plt.legend()
plt.show()
```



[17]:
```
pip install mplfinance pandas
```

```
Requirement already satisfied: mplfinance in c:\users\kirti\anaconda3\lib\site-
packages (0.12.10b0)Note: you may need to restart the kernel to use updated
packages.

Requirement already satisfied: pandas in c:\users\kirti\anaconda3\lib\site-
packages (2.0.3)
Requirement already satisfied: matplotlib in c:\users\kirti\anaconda3\lib\site-
packages (from mplfinance) (3.7.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
c:\users\kirti\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
c:\users\kirti\anaconda3\lib\site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in
c:\users\kirti\anaconda3\lib\site-packages (from pandas) (2023.3)
Requirement already satisfied: numpy>=1.21.0 in
c:\users\kirti\anaconda3\lib\site-packages (from pandas) (1.24.3)
Requirement already satisfied: six>=1.5 in c:\users\kirti\anaconda3\lib\site-
packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Requirement already satisfied: contourpy>=1.0.1 in
c:\users\kirti\anaconda3\lib\site-packages (from matplotlib->mplfinance) (1.0.5)
Requirement already satisfied: cycler>=0.10 in
```

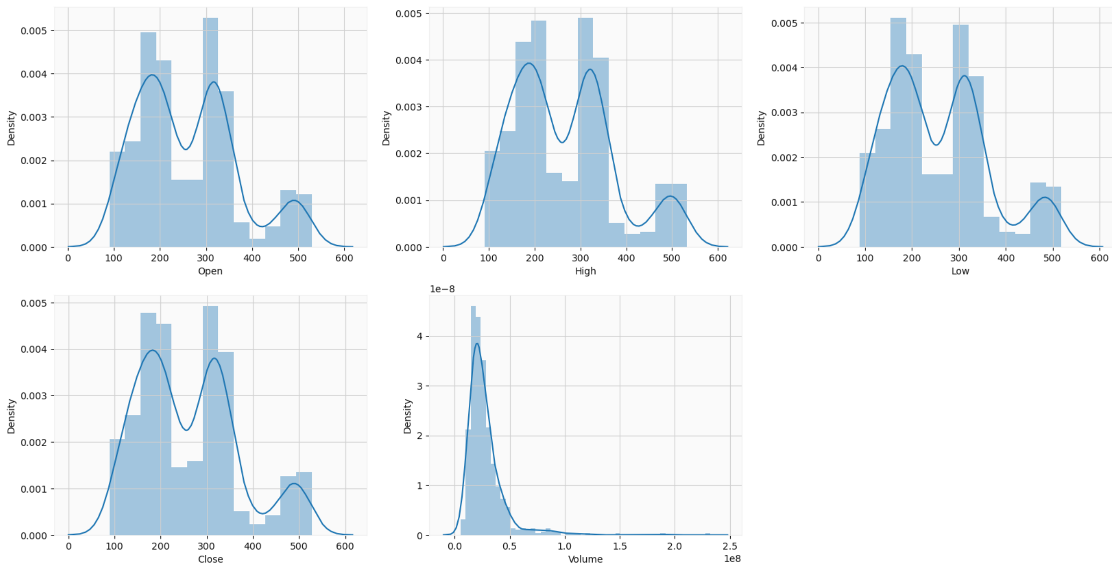
```
c:\users\kirti\anaconda3\lib\site-packages (from matplotlib->mplfinance)
(0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
c:\users\kirti\anaconda3\lib\site-packages (from matplotlib->mplfinance)
(4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in
c:\users\kirti\anaconda3\lib\site-packages (from matplotlib->mplfinance) (1.4.4)
Requirement already satisfied: packaging>=20.0 in
c:\users\kirti\anaconda3\lib\site-packages (from matplotlib->mplfinance) (23.1)
Requirement already satisfied: pillow>=6.2.0 in
c:\users\kirti\anaconda3\lib\site-packages (from matplotlib->mplfinance)
(10.2.0)
Requirement already satisfied: pyparsing<3.1,>=2.3.1 in
c:\users\kirti\anaconda3\lib\site-packages (from matplotlib->mplfinance) (3.0.9)
```

[18]:
```python
import mplfinance as mpf
import warnings
warnings.filterwarnings("ignore")
```

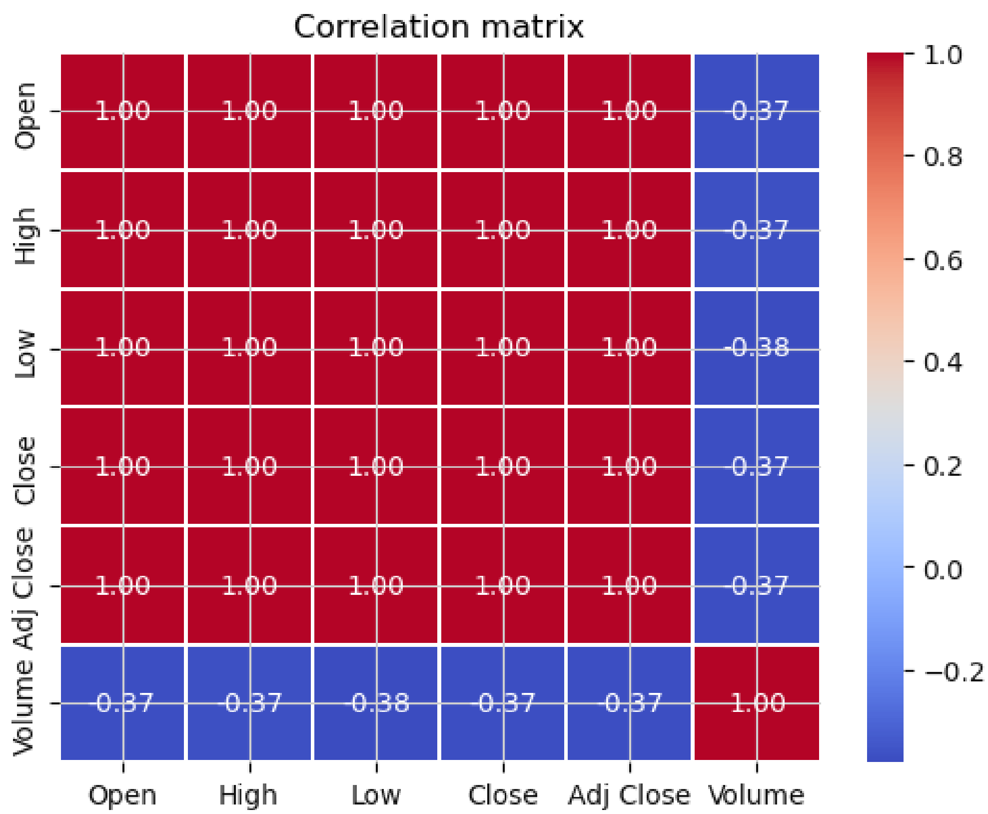
[19]:
```python
plt.figure(figsize=(25,6))
mpf.plot(data=df, type='candle', volume=True, style='yahoo')
```

```
<Figure size 2500x600 with 0 Axes>
```

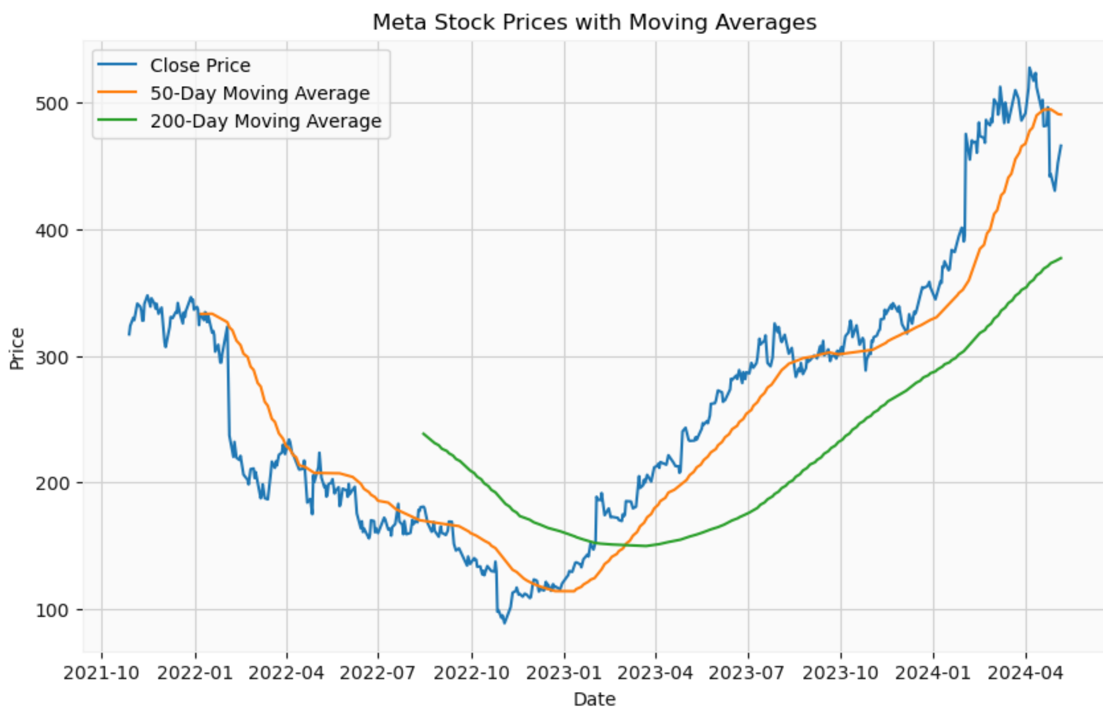
[20]:
```
features = ['Open', 'High', 'Low', 'Close', 'Volume']

plt.subplots(figsize=(20,10))

for i, col in enumerate(features):
    plt.subplot(2,3,i+1)
    sns.distplot(df[col], kde=True)
plt.show()
```



[21]:
```
correlation_matrix=df.corr()
sns.heatmap(correlation_matrix, cmap="coolwarm", annot=True, fmt='.2f',
 ↪linewidths=.25)
plt.title("Correlation matrix")
plt.show()
```



[22]:
```
df['MA50'] = df['Close'].rolling(window=50).mean()
df['MA200'] = df['Close'].rolling(window=200).mean()

plt.figure(figsize=(10, 6))
plt.plot( df['Close'], label='Close Price')
plt.plot( df['MA50'], label='50-Day Moving Average')
plt.plot( df['MA200'], label='200-Day Moving Average')
plt.xlabel('Date')
plt.ylabel('Price')
plt.title('Meta Stock Prices with Moving Averages')
plt.legend()
plt.show()
```



[23]:
```
df=df.drop(["MA50","MA200"], axis=1)
```

0.0.4 Machine Learning Models¶

These models can be applied to regression tasks where the goal is to predict a continuous target variable based on one or more input features.

1.Linear Regression
2.Ridge Regression
3.Lasso Regression
4.ElasticNet Regression
5.Support Vector Machines (SVM) with kernel functions like linear, polynomial, or RBF
6.Decision Trees (and ensemble methods like Random Forests)
7.Gradient Boosting Machines (GBM) and its variants like XGBoost, LightGBM, and CatBoost
8.Neural Networks (e.g., Multi-layer Perceptron, Convolutional Neural Networks for image data, Recurrent Neural Networks for sequential data)
9.K-Nearest Neighbors Regression(KNN)

0.0.5 Suitable metrics

1.Mean Absolute Error (MAE)
2.Mean Squared Error (MSE)
3.Root Mean Squared Error (RMSE)
4.Mean Absolute Percentage Error (MAPE)
5.R-squared (R^2)

6.Adjusted R-squared

1 Using Linear Regression

[24]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error,mean_absolute_error
```

[25]:
```
X=df.drop('Adj Close', axis=1)
y=df['Adj Close']
```

[26]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.3,␣
 ↪random_state=42)
```

[27]:
```
print('X Training Shape:', X_train.shape)
print('y Training Shape:', y_train.shape)
print('X Testing Shape:', X_test.shape)
print('y Testing Shape:', y_test.shape)
```

```
X Training Shape: (443, 5)
y Training Shape: (443,)
X Testing Shape: (190, 5)
y Testing Shape: (190,)
```

[28]:
```
model=LinearRegression()
model
```

[28]:
```
LinearRegression()
```

[29]:
```
model.fit(X_train, y_train)
```

[29]:
```
LinearRegression()
```

[30]:
```
model.intercept_
```

[30]:
```
-0.20678568149986631
```

[31]:
```
model.coef_
```

[31]:
```
array([ 7.64454021e-03, -3.98022156e-03, -7.23159977e-03,  1.00335347e+00,
        4.31480664e-10])
```

[32]:
```
y_pred=model.predict(X_test)
y_pred
```

[32]:
```
array([171.84560411, 129.79973625, 246.59788953, 191.38828965,
       308.41809773, 328.91001157, 169.27526613, 207.48877787,
```

```
334.46323582, 158.53635026, 111.18130445, 334.20369297,
317.90381691, 198.27416881, 472.98708908, 216.30167272,
329.70112328,  88.71677445, 141.27426537, 174.70425589,
203.55949194, 117.81781239, 331.52620501, 438.86162709,
326.92855877, 223.07660157, 209.17526729, 113.92908556,
429.85157322, 111.68679373, 299.89017818, 301.37886971,
132.59866151, 454.45936071, 195.98392525, 173.18349606,
186.27864662, 501.45569125, 175.36863825, 177.72390232,
215.85644666, 334.97233113, 202.76472145, 294.04653037,
172.20098893, 165.14552966, 116.87502707, 162.83142342,
205.93866925, 200.45321227, 156.94357382, 119.54064759,
327.47093615, 163.25207865, 505.2283313 , 483.8293056 ,
190.99649647, 320.27118286, 233.2760649 , 329.50388767,
501.98273836, 312.30793678, 178.11340152, 345.9310479 ,
129.51630146, 121.36208633, 319.2957724 , 263.37323684,
354.55425142, 333.93801224, 358.39878179, 310.14142448,
210.25153988, 116.66463464, 169.13117323, 334.42266439,
370.18734672, 219.97902062, 218.6242831 ,  96.26771509,
162.66853375, 299.88272396, 281.56798152, 286.79169288,
160.12088049, 509.28436906, 511.8612973 , 182.85612852,
157.83141034, 166.43453556, 115.68323459, 193.79322586,
329.53589169, 191.38999548, 219.32512118, 114.26742064,
199.56426449, 101.28151882, 132.76387623, 122.20604054,
294.06483514, 137.28640759, 201.95659021, 346.07392199,
133.56778356, 207.36664839, 299.96940306, 231.59954042,
369.44678042, 140.19679666, 301.46359284, 168.7175905 ,
227.81958765, 188.45182357, 181.472698  , 217.06914819,
505.68365238, 242.94426155, 393.86413683, 160.15969798,
113.01585765, 205.1126129 , 132.66563539, 149.31572412,
170.34070206, 486.74036938, 205.77631727, 340.843856  ,
229.61760141, 213.91864504, 323.73337958, 394.48495483,
489.86128947, 158.93493685, 316.71099802, 147.07863959,
471.45450437, 135.14216191, 126.72193027, 246.46992429,
191.04846855, 129.81684215, 327.54702644, 342.677413  ,
219.34667983, 346.84698598, 322.81065291, 340.61591894,
318.55574504, 511.61922312, 151.27289127, 167.73498301,
211.25314406, 210.78919234, 116.91282254, 286.529857  ,
201.84230749, 310.41345419, 305.4781015 , 491.05543698,
183.22404069, 180.65389869, 273.1117357 , 293.892356  ,
199.78778565, 353.83167052, 109.64777796, 129.62192843,
327.40300343, 336.71287368, 491.57658048, 170.16037272,
499.43722956, 117.88381967, 203.27648183, 120.22548459,
285.01778726, 495.27240329, 136.15381573,  90.60158394,
473.06596818, 516.66593128, 164.47044325, 136.20852356,
196.38039101, 220.80268754, 187.39964   , 304.55769083,
236.883367  , 158.89085567])
```

[33]:
```
r2=r2_score(y_pred,y_test) ## Higher values indicate better model performance,␣
↪with 1 meaning perfect prediction
print("R² for Linear Regression Model is", r2)
```

```
R² for Linear Regression Model is 0.9999989713428837
```

[34]:
```
mse=mean_squared_error(y_pred,y_test) ## Lower values indicates better model␣
↪preformance
print("MSE for Linear Regression Model is", mse)
```

```
MSE for Linear Regression Model is 0.013177732400519863
```

[35]:
```
mea=mean_absolute_error(y_test, y_pred)  ## Lower value indicates better model␣
↪performance
print("Mean Absolute Error for Linear Regression is ", mea)
```

```
Mean Absolute Error for Linear Regression is  0.08363125599643
```

2 Using XgBoost Model

[36]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\kirti\anaconda3\lib\site-
packages (2.0.3)
Requirement already satisfied: numpy in c:\users\kirti\anaconda3\lib\site-
packages (from xgboost) (1.24.3)
Requirement already satisfied: scipy in c:\users\kirti\anaconda3\lib\site-
packages (from xgboost) (1.11.1)
Note: you may need to restart the kernel to use updated packages.
```

[37]:
```
from xgboost import XGBRegressor
```

[38]:
```
model2=XGBRegressor()
model2
```

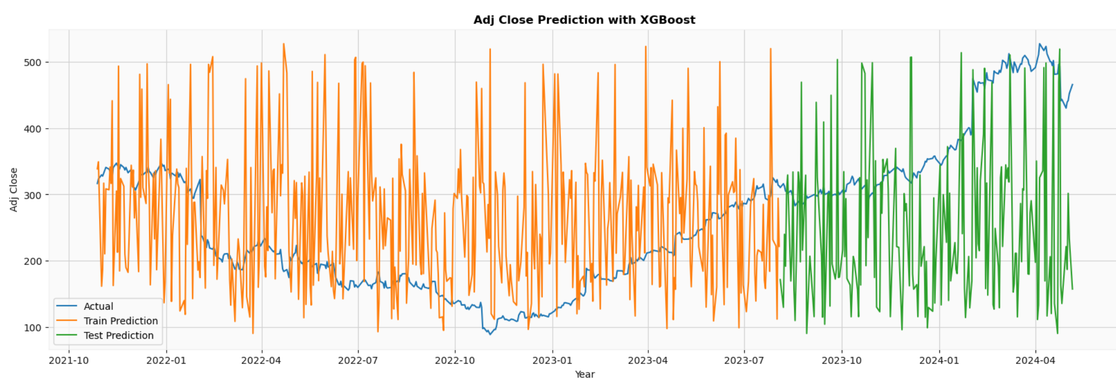
[38]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, …)
```

[39]:
```
model2.fit(X_train, y_train)
```

[39]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, …)
```

[40]:
```
# Make predictions
y_pred_train = model2.predict(X_train)
y_pred_test = model2.predict(X_test)
```

[41]:
```
# Visualize the predictions
plt.figure(figsize=(20, 6))
plt.plot(df.index, df['Adj Close'], label='Actual')
plt.plot(df.index[:len(y_train)], y_pred_train, label='Train Prediction')
plt.plot(df.index[len(y_train):], y_pred_test, label='Test Prediction')
plt.xlabel('Year')
plt.ylabel('Adj Close')
plt.title('Adj Close Prediction with XGBoost', fontweight='bold')
plt.legend()
plt.show()
```



[42]:
```
r2=r2_score(y_train, y_pred_train)
print(r2)
```

```
0.9999985831646808
```

[43]:
```
r2=r2_score(y_test, y_pred_test)
print("R² for XGBoost Model is", r2)
```

```
R² for XGBoost Model is 0.9994371565478546
```

[44]:
```
mean_error=mean_squared_error(y_test, y_pred_test)
print("MSE of XGBoost Model is ", mean_error)
```

```
MSE of XGBoost Model is  7.211095237967466
```

3 Using Support Vector Regressor

[45]:
```
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
```

[46]:
```
svr=SVR(kernel='rbf')
svr
```

[46]:
```
SVR()
```

[47]:
```
svr.fit(X_train, y_train)
```

[47]:
```
SVR()
```

[48]:
```
y_pred_train = svr.predict(X_train)
y_pred_test = svr.predict(X_test)
```

[49]:
```
train_mse=mean_squared_error(y_train, y_pred_train)
test_mse=mean_squared_error(y_test, y_pred_test)
train_r2=r2_score(y_train, y_pred_train)
test_r2=r2_score(y_test, y_pred_test)

print("Training MSE: " ,train_mse)
print("Test MSE: ", test_mse)
print("Train R²: ", train_r2)
print("Test R²", test_r2)
```

```
Training MSE:  9112.31089323682
Test MSE:  10472.741332157315
Train R²:  0.1611369057498142
Test R² 0.1825771688908605
```

[50]:
```
y_pred=svr.predict(X_test)
r2 = r2_score(y_test, y_pred)
print("R² for Support Vector Regressor Model is", r2)
```

```
R² for Support Vector Regressor Model is 0.1825771688908605
```

4 Using Random Forest Regressor model

[51]:
```
from sklearn.ensemble import RandomForestRegressor
```

[52]:
```
rm=RandomForestRegressor()
rm
```

[52]:
```
RandomForestRegressor()
```

[53]:
```
rm.fit(X_train, y_train)
```

[53]:
```
RandomForestRegressor()
```

[54]:
```
y_pred=rm.predict(X_test)
y_pred
```

[54]:
```
array([173.11981739, 128.4682894 , 243.46288574, 191.49511703,
       308.6599035 , 328.25641521, 169.46769407, 208.32056675,
       334.60937617, 158.06678842, 111.91425679, 334.10540912,
       318.17112346, 198.0639494 , 470.31970758, 215.93309006,
       329.82874163,  96.26086666, 142.01512193, 174.5194339 ,
       204.37156081, 117.05480409, 331.7681974 , 441.33302655,
       326.51676095, 223.44982028, 209.73077462, 116.34955175,
       437.99585937, 108.20139744, 299.58913045, 301.3657434 ,
       133.1404324 , 460.2586171 , 195.95608711, 173.75224616,
       186.51900234, 502.21279687, 176.77993223, 178.28803253,
       215.76846509, 334.66321857, 204.03341833, 294.35549059,
       173.04859391, 163.77403531, 116.57221582, 162.94161499,
       206.15106885, 199.30853042, 156.542111  , 119.07006527,
       327.89489879, 162.92992838, 503.29199816, 483.86998168,
       190.21357627, 320.16779413, 234.86640201, 329.93593349,
       499.86680016, 312.36598191, 176.62649473, 347.92933836,
       128.72291975, 121.32877046, 319.84723655, 266.87113957,
       353.97563368, 333.89412921, 355.3973119 , 310.37518043,
       210.16841178, 117.39194582, 169.21456175, 334.69348256,
       365.6982803 , 220.81161853, 218.31586117,  99.22192384,
       162.98307082, 299.55546367, 278.19363708, 286.69351682,
       160.44166634, 508.02280332, 507.43020156, 182.74370679,
       158.1028499 , 167.2093852 , 115.79663877, 193.70527458,
       329.90516595, 191.52288727, 218.24633472, 115.33123245,
       199.97832075,  98.38091523, 131.08061807, 123.56869251,
       294.13102035, 136.07871873, 202.58855284, 346.99073276,
       133.27149024, 206.87639605, 299.81229221, 232.79989585,
       369.59455077, 140.60921235, 301.42777293, 168.49132713,
       228.2135646 , 188.13948467, 181.21962139, 216.5724077 ,
       508.83470261, 240.99140578, 394.08756449, 159.98535249,
       113.92202733, 204.76344124, 132.18584505, 150.23000289,
```

```
170.43507112, 484.35797713, 205.4313322 , 340.87063536,
227.90768924, 216.04797057, 323.13225387, 396.92982461,
490.34416771, 158.87612585, 316.58500585, 146.37758745,
468.68248685, 135.64797219, 127.12382142, 247.51498683,
191.09763922, 128.98094863, 327.81528424, 343.50323443,
221.3701261 , 346.67226997, 322.73257353, 340.58454311,
318.48328648, 510.49850477, 151.80253551, 168.0221244 ,
211.21999526, 210.80783545, 116.61467072, 287.86836941,
202.99272334, 311.06854574, 305.4761803 , 491.98776778,
183.79289558, 180.40918037, 271.70590492, 293.79667101,
199.40422894, 352.78959461, 110.18958687, 129.41538464,
327.80359476, 336.5963633 , 492.94329592, 170.21949944,
499.66480094, 115.75608108, 204.66824552, 120.14812222,
285.1176868 , 496.04640213, 137.77661686,  96.39652271,
472.03171282, 518.38340016, 163.43339576, 136.71873802,
196.07655878, 219.88310127, 187.43303263, 304.91947564,
235.74746813, 158.22731696])
```

[55]:
```
r2=r2_score(y_test, y_pred)
print("R² for Random Forest Regressor Model is: ", r2)
```

```
R² for Random Forest Regressor Model is:  0.9998142074646916
```

[56]:
```
mse=mean_squared_error(y_test, y_pred)
print("Mean Squared Error for Random Forest Regressor is : ", mse)
```

```
Mean Squared Error for Random Forest Regressor is :  2.3803557836653466
```

5 Using K-Nearest Neighbour Regressor model

[57]:
```
from sklearn.neighbors import KNeighborsRegressor
```

[58]:
```
#scaler=StandardScaler()
#X_train=scaler.fit_transform(X_train)
#x_test=scaler.fit_transform(X_test)
knn=KNeighborsRegressor(n_neighbors=4)
knn.fit(X_train, y_train)
```

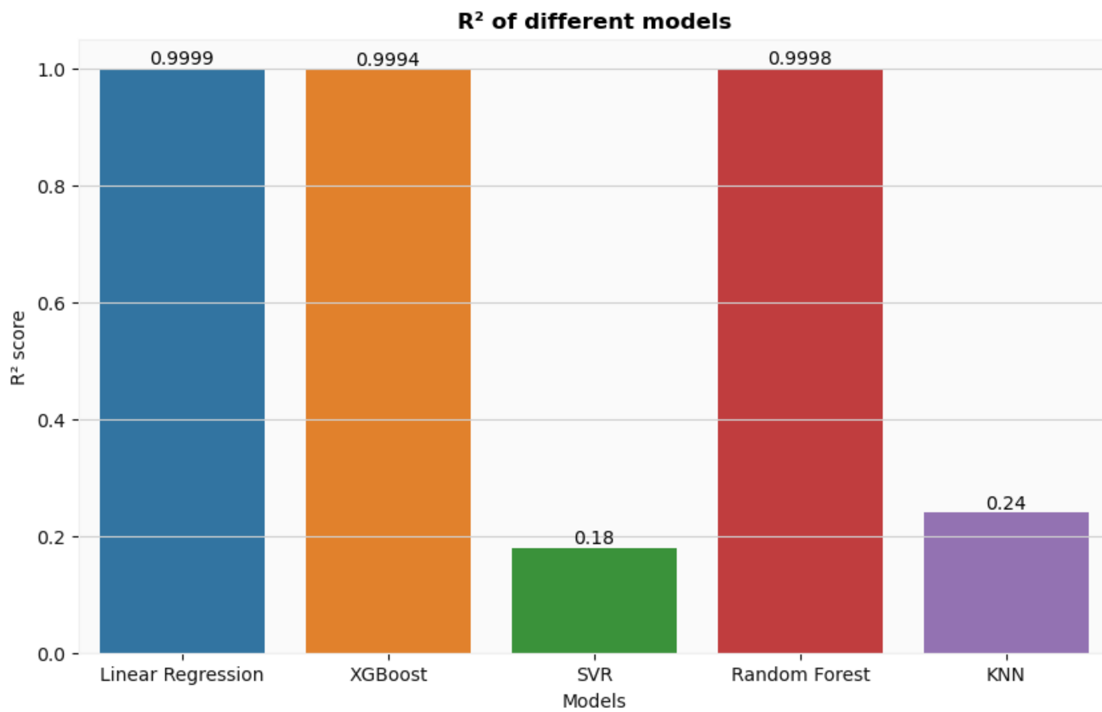
[58]:
```
KNeighborsRegressor(n_neighbors=4)
```

[59]:
```
y_pred=knn.predict(X_test)
```

[60]:
```
r2=r2_score(y_test, y_pred)
mse=mean_squared_error(y_test, y_pred)
print("R² for KNN Regressor Model is : ", r2)
print("Mean Squared Error for KNN Regressor is : ", mse)
```

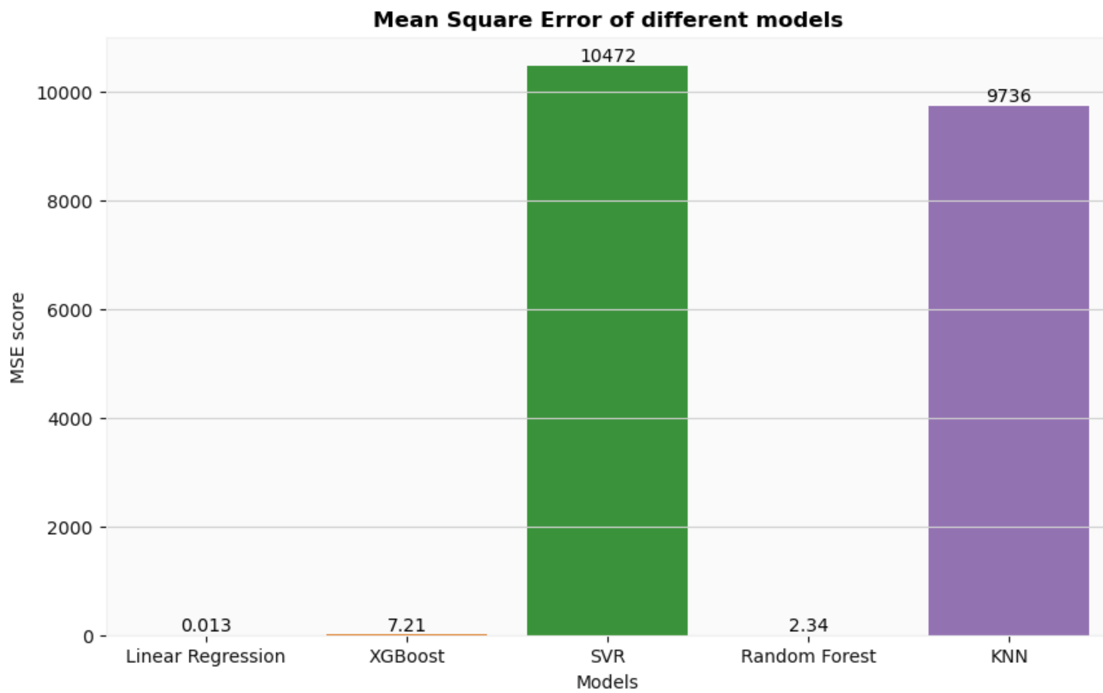
```
R² for KNN Regressor Model is :  0.240033349289411646
Mean Squared Error for KNN Regressor is :  9736.616530790738
```

[61]:
```
models=["Linear Regression","XGBoost","SVR","Random Forest","KNN"]
r_square=[0.9999, 0.9994, 0.18, 0.9998, 0.24]

plt.figure(figsize=(10,6))
ax=sns.barplot(x=models, y=r_square)
for bar in ax.containers:
    ax.bar_label(bar)
plt.title("R² of different models", fontweight='bold')
plt.xlabel("Models")
plt.ylabel("R² score")
plt.show()
```



[62]:
```
models=["Linear Regression","XGBoost","SVR","Random Forest","KNN"]
mean_square=[0.013, 7.21, 10472, 2.34, 9736]

plt.figure(figsize=(10,6))
ax=sns.barplot(x=models, y=mean_square)
for bar in ax.containers:
    ax.bar_label(bar)
plt.title("Mean Square Error of different models", fontweight='bold')
plt.xlabel("Models")
```

```python
plt.ylabel("MSE score")
plt.show()
```



6 Conclusion

The anaysis indicates that Linear Regression is most suitable model for the predication of the stock prices. Linear Regression Model has highest R^2 score .9999 as compared to Random Forest (.9998), XGBoost (.9994).

It also indicates that SVR and KNN are not the suitable models for such dataset.

Another metrics, Mean Square Error also indicates that Linear Regression has lowest MSE (0.013) making it best suitable model