In [2]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
# Now some of Visualisation libraries
import matplotlib.pyplot as plt
import seaborn as sns
```

Data & EDA

In [3]:
```
df= pd.DataFrame(pd.read_csv("Stock_Price_data_set.csv"))
df.head()
```

Out[3]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	11896100
1	2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	12595800
2	2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	8981500
3	2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	9306700
4	2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	16906900

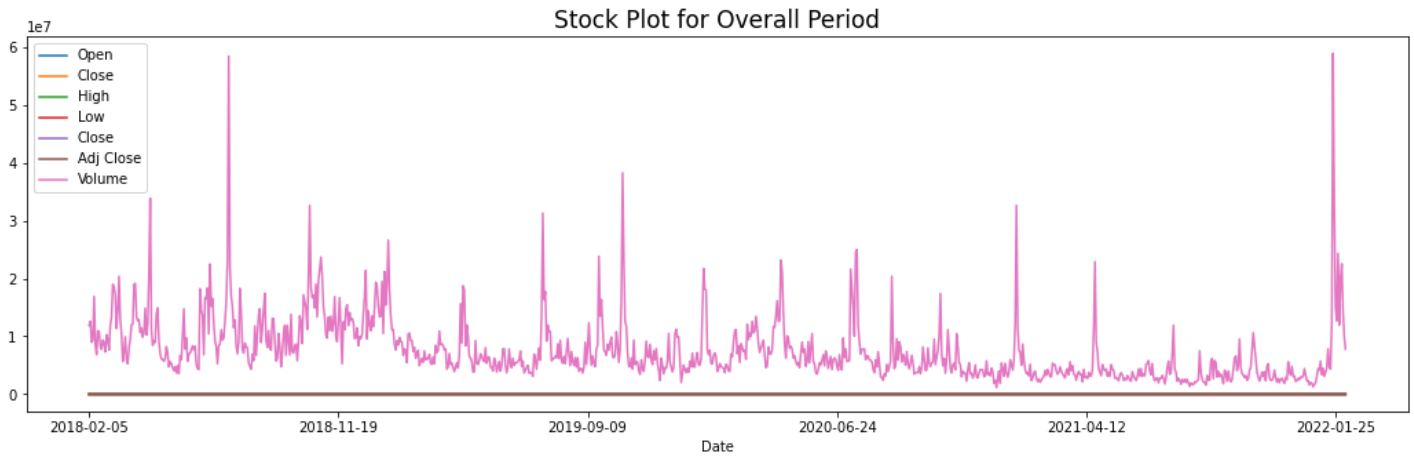
In [4]:
```
# setting date to index
df.set_index('Date', inplace=True)
df.head()
```

Out[4]:

Date	Open	High	Low	Close	Adj Close	Volume
2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	11896100
2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	12595800
2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	8981500
2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	9306700
2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	16906900

In [6]:
```
df[['Open', 'Close', 'High', 'Low', 'Close', 'Adj Close', 'Volume']].plot(figsize=(18,5))
plt.title("Stock Plot for Overall Period", fontsize=17)
```

Out[6]:
```
Text(0.5, 1.0, 'Stock Plot for Overall Period')
```



Top 5 Dates with Highest Stock Price

In [7]:
```
a = df.sort_values(by='High',ascending= False).head(5)
a['High']
```

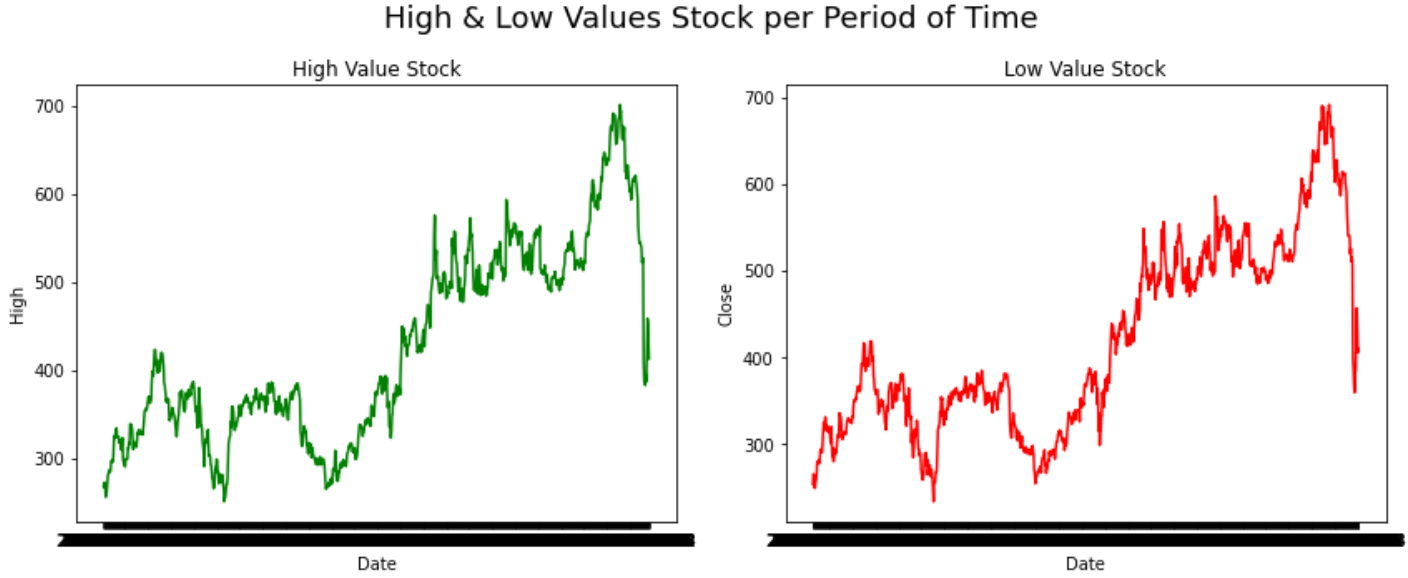
Out[7]:
```
Date
2021-11-17    700.989990
2021-11-19    694.159973
2021-11-18    691.739990
2021-10-29    690.969971
2021-11-01    689.969971
Name: High, dtype: float64
```

Top 5 Dates with Lowest Stock Price

In [8]:
```
b = df.sort_values(by='Low',ascending= True).head(5)
b['Low']
```

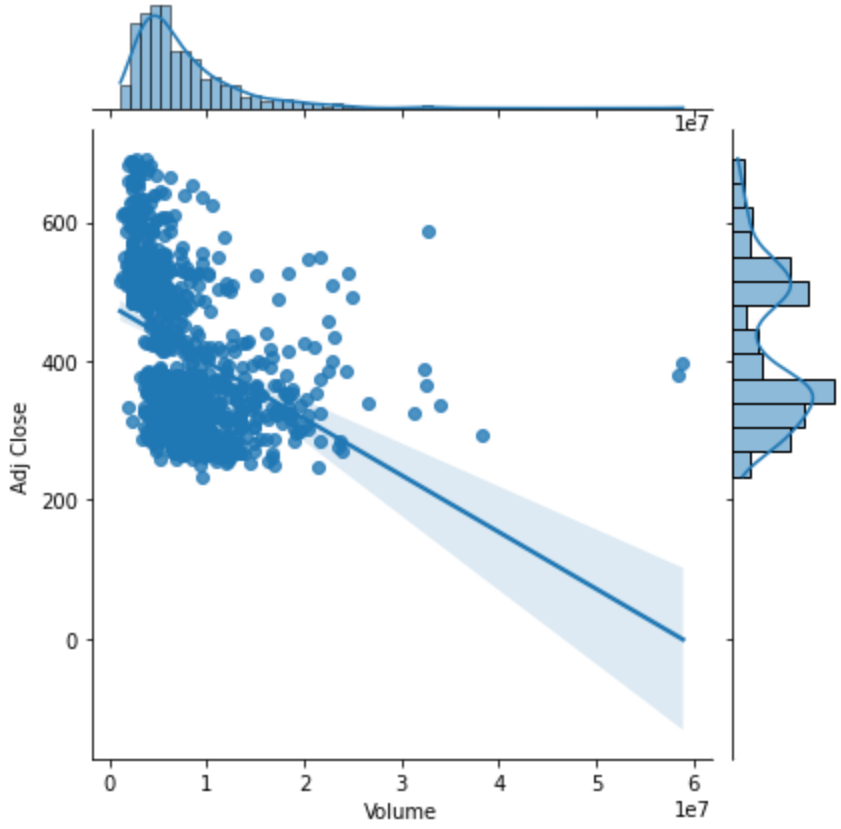
Out[8]:
```
Date
2018-12-26    231.229996
2018-12-24    233.679993
2018-02-09    236.110001
2018-12-27    240.100006
2018-12-21    241.289993
Name: Low, dtype: float64
```

In [9]:
```
fig,axes= plt.subplots(nrows=1,ncols=2, sharex=True, figsize=(12,5))
fig.suptitle('High & Low Values Stock per Period of Time',fontsize=18)
sns.lineplot(ax= axes[0], y=df['High'],x=df.index, color='green')
axes[0].set_title('High Value Stock')
sns.lineplot(ax= axes[1], y=df['Close'], x=df.index, color='red')
axes[1].set_title('Low Value Stock')
plt.tight_layout()
plt.show()
```

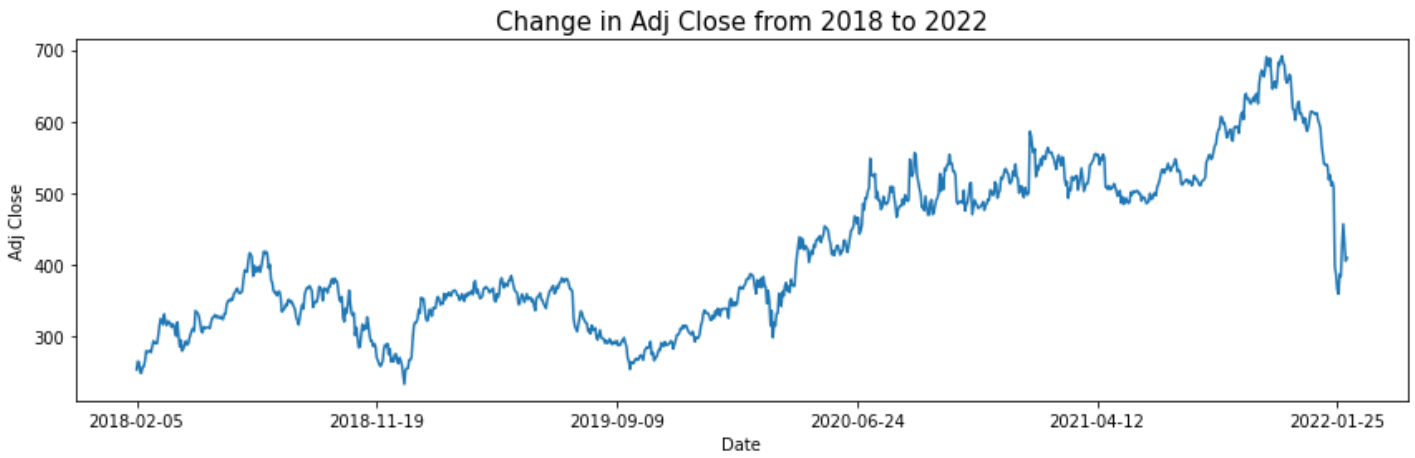


In [10]:
```
sns.jointplot(x='Volume', y='Adj Close', data=df, kind='reg')
```

Out[10]:
```
<seaborn.axisgrid.JointGrid at 0x1b3099b3e80>
```



In [11]:
```
plt.figure(figsize=(12,4))
df['Adj Close'].plot()
plt.xlabel('Date', fontsize= 10)
plt.ylabel('Adj Close',fontsize= 10)
plt.title('Change in Adj Close from 2018 to 2022',fontsize= 15)
plt.tight_layout()
```



Stock Daily Returns

In [13]:
```
df['Daily_returns'] = df['Adj Close'].pct_change()
#Pandas dataframe.pct_change()
#function calculates the percentage change
#between the current and a prior element.
df.head()
```

Out[13]:

Date	Open	High	Low	Close	Adj Close	Volume	Daily_returns
2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	11896100	NaN
2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	12595800	0.045072
2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	8981500	-0.004366
2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	9306700	-0.054657
2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	16906900	-0.002519

Best Day of Stock

In [14]:
```
df[df['Daily_returns']==df['Daily_returns'].max()]['Daily_returns']
```

Out[14]:
```
Date
2021-01-20    0.168543
Name: Daily_returns, dtype: float64
```

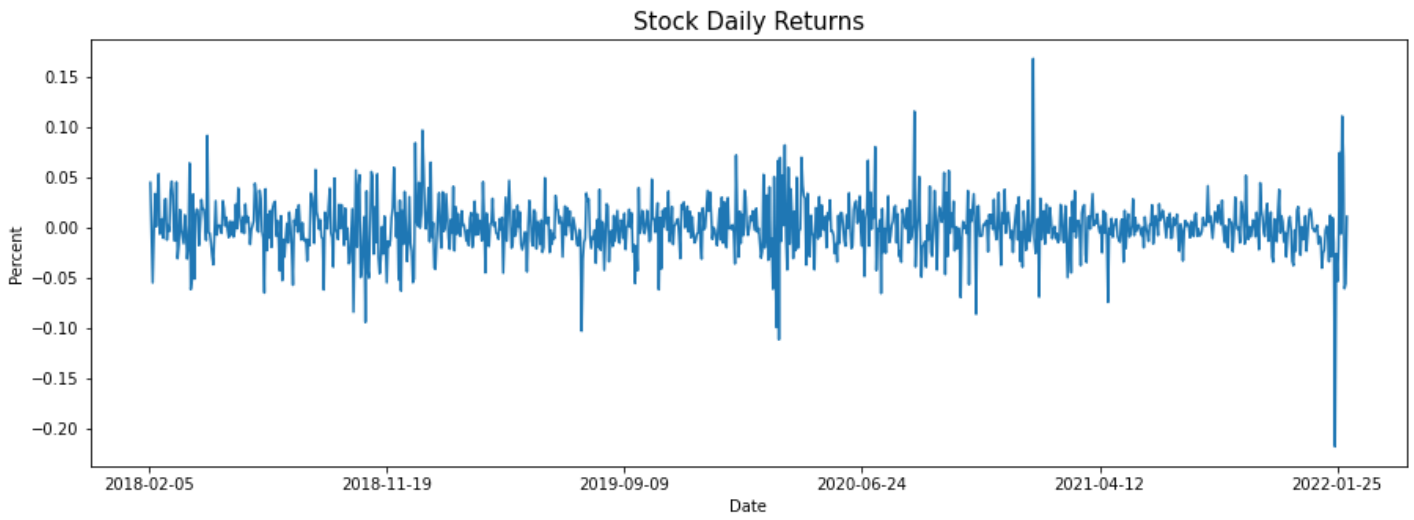
Worst Day of Stock

In [15]:
```
df[df['Daily_returns']==df['Daily_returns'].min()]['Daily_returns']
```

Out[15]:
```
Date
2022-01-21   -0.217905
Name: Daily_returns, dtype: float64
```

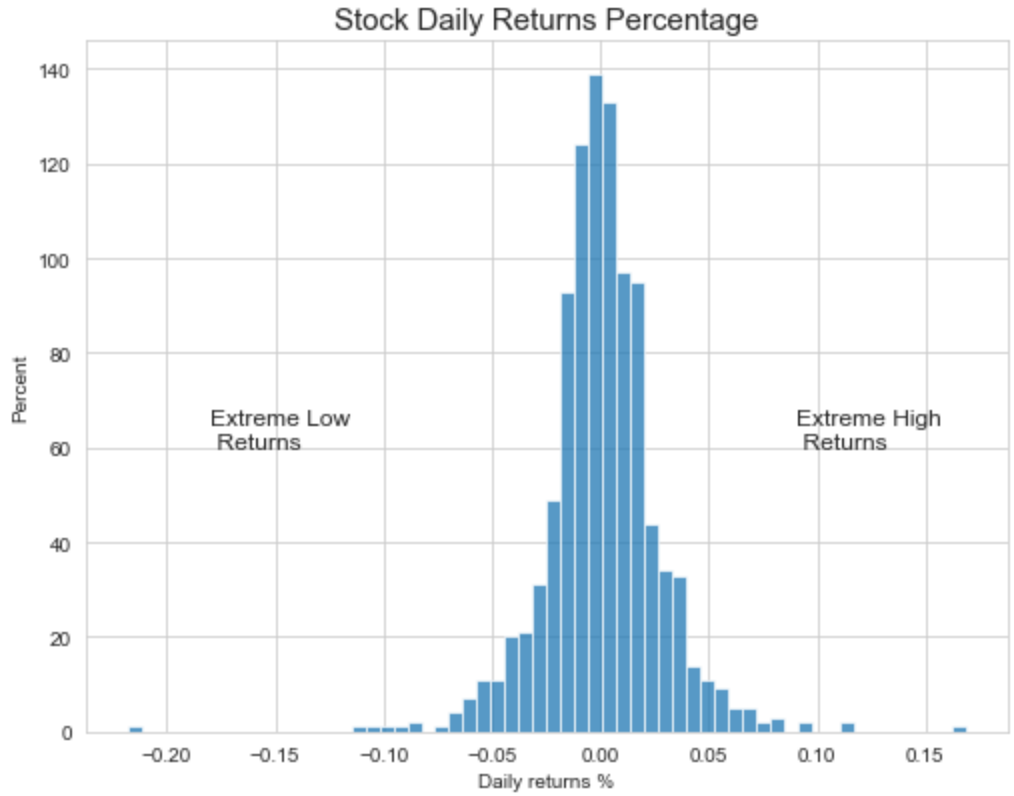
In [16]:
```
plt.figure(figsize=(15,5))
df['Daily_returns'].plot()
plt.xlabel("Date")
plt.ylabel("Percent")
plt.title("Stock Daily Returns",fontsize= 15 )
```

Out[16]:
```
Text(0.5, 1.0, 'Stock Daily Returns')
```



In [17]:

```
sns.set_style('whitegrid')
fig = plt.figure(figsize=(8,6))
ax1 = fig.add_axes([0.1,0.1,0.8,0.8])
sns.histplot(data= df['Daily_returns'], bins=60)
ax1.set_xlabel("Daily returns %")
ax1.set_ylabel("Percent")
ax1.set_title("Stock Daily Returns Percentage",fontsize= 15 )
ax1.text(-0.18,60,"Extreme Low\n Returns",fontsize= 12)
ax1.text(0.09,60,"Extreme High\n Returns", fontsize= 12)
plt.show()
```



Stock Cumulative Returns

In [18]:

```
df['Cum_returns']= (df['Daily_returns']+1).cumprod()
df.head()
```

Out[18]:

	Open	High	Low	Close	Adj Close	Volume	Daily_returns	Cum_returns

	Date	Open	High	Low	Close	Adj Close	Volume	Daily_returns	Cum_returns
	Date								
2018-02-05		262.000000	267.899994	250.029999	254.259995	254.259995	11896100	NaN	NaN
2018-02-06		247.699997	266.700012	245.000000	265.720001	265.720001	12595800	0.045072	1.045072
2018-02-07		266.579987	272.450012	264.329987	264.559998	264.559998	8981500	-0.004366	1.040510
2018-02-08		267.079987	267.619995	250.000000	250.100006	250.100006	9306700	-0.054657	0.983639
2018-02-09		253.850006	255.800003	236.110001	249.470001	249.470001	16906900	-0.002519	0.981161

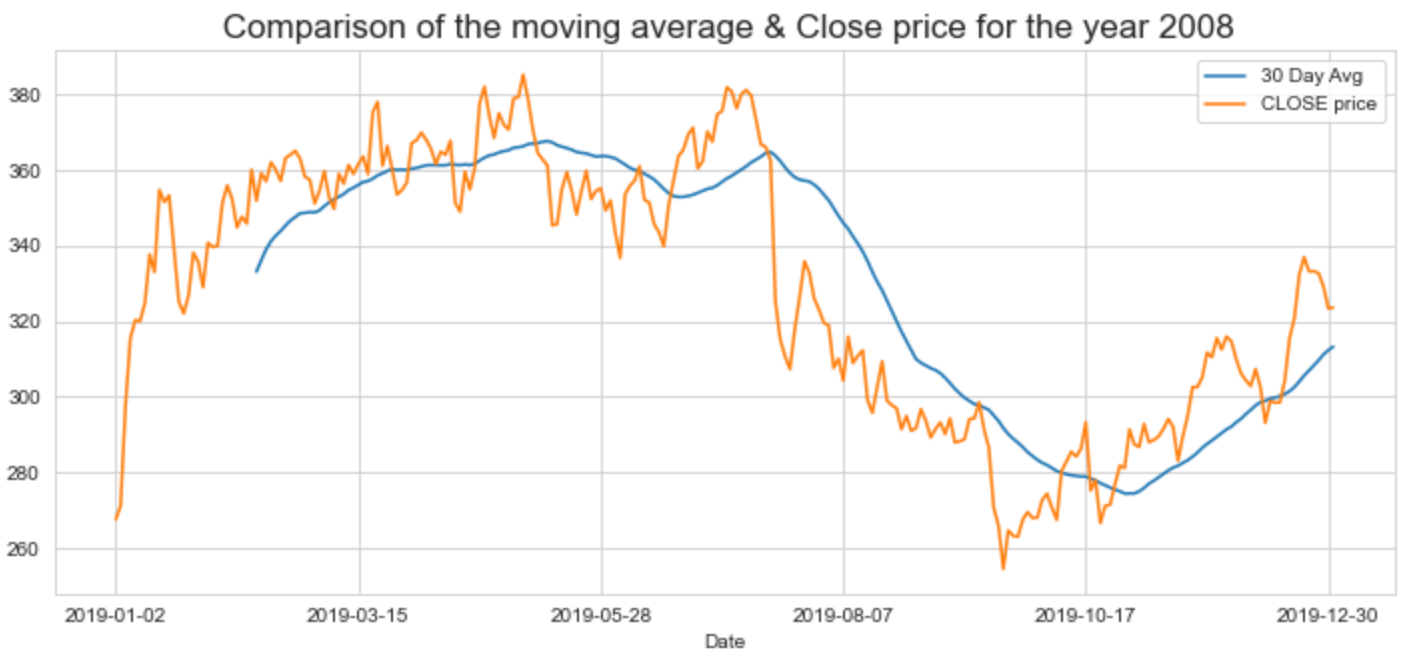
In [19]:
```
sns.set_style('whitegrid')
plt.figure(figsize=(15,5))
df['Cum_returns'].plot()
plt.xlabel("Date")
plt.ylabel("Percent")
plt.title("Stock Cumulative Returns",fontsize= 15 )
plt.legend()
```

Out[19]:
```
<matplotlib.legend.Legend at 0x1b30bee4eb0>
```

Moving Average of Stock

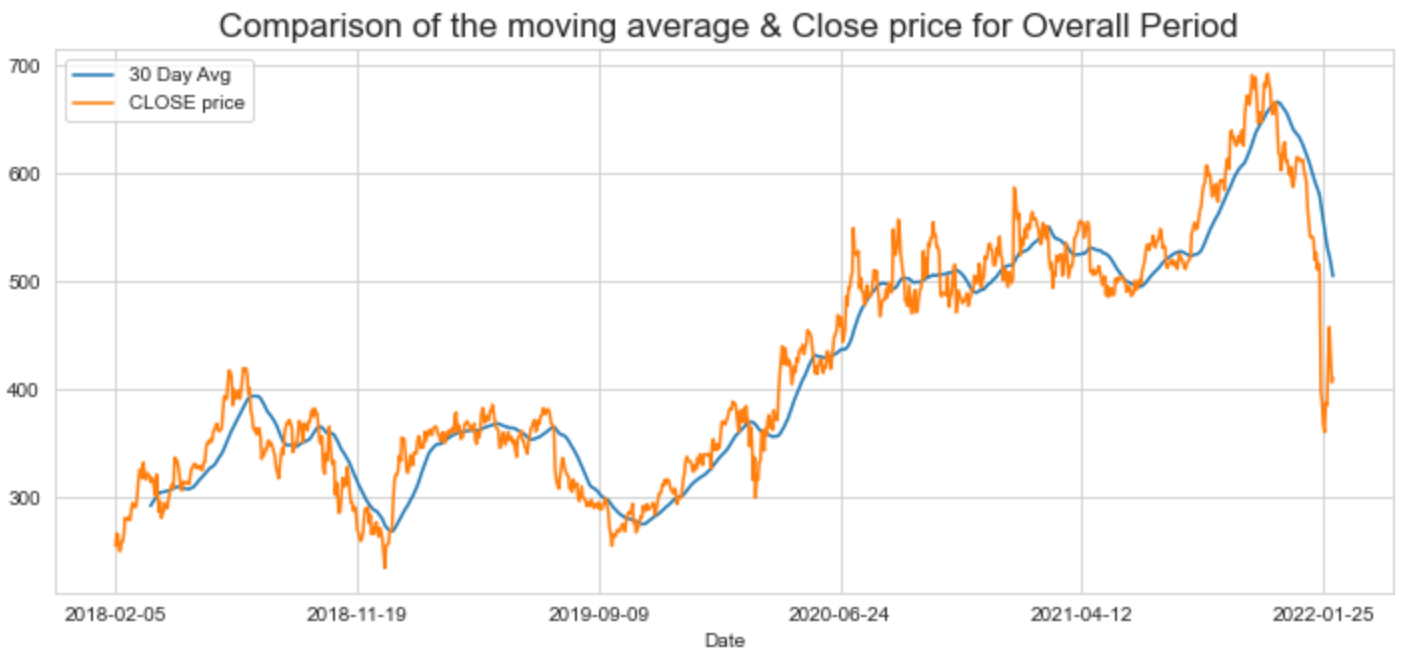
In [20]:
```
sns.set_style('whitegrid')
f= plt.figure(figsize=(12,5))
df['Close'].loc['2019-01-01': '2019-12-31'].rolling(window=30).mean().plot(label='30 Day A
df['Close'].loc['2019-01-01':'2019-12-31'].plot(label='CLOSE price')
plt.title(" Comparison of the moving average & Close price for the year 2008", fontsize=17
plt.legend()
```

Out[20]:
```
<matplotlib.legend.Legend at 0x1b30f793df0>
```



In [21]:
```
f= plt.figure(figsize=(12,5))
df['Close'].rolling(window=30).mean().plot(label='30 Day Avg')
df['Close'].plot(label='CLOSE price')
plt.title(" Comparison of the moving average & Close price for Overall Period", fontsize=1
plt.legend()
```

Out[21]:
```
<matplotlib.legend.Legend at 0x1b30f7ff790>
```



In [22]:
```
df['Mean Avg 30Day'] = df['Close'].rolling(window=30).mean() # MA= mean Average
df['STD 30Day'] = df['Close'].rolling(window=30).std()
```

In [23]:
```
df['Upper Band']= df['Mean Avg 30Day'] + (df['STD 30Day'] *2)
df['Lower Band']= df['Mean Avg 30Day'] - (df['STD 30Day'] *2)
df.head()
```

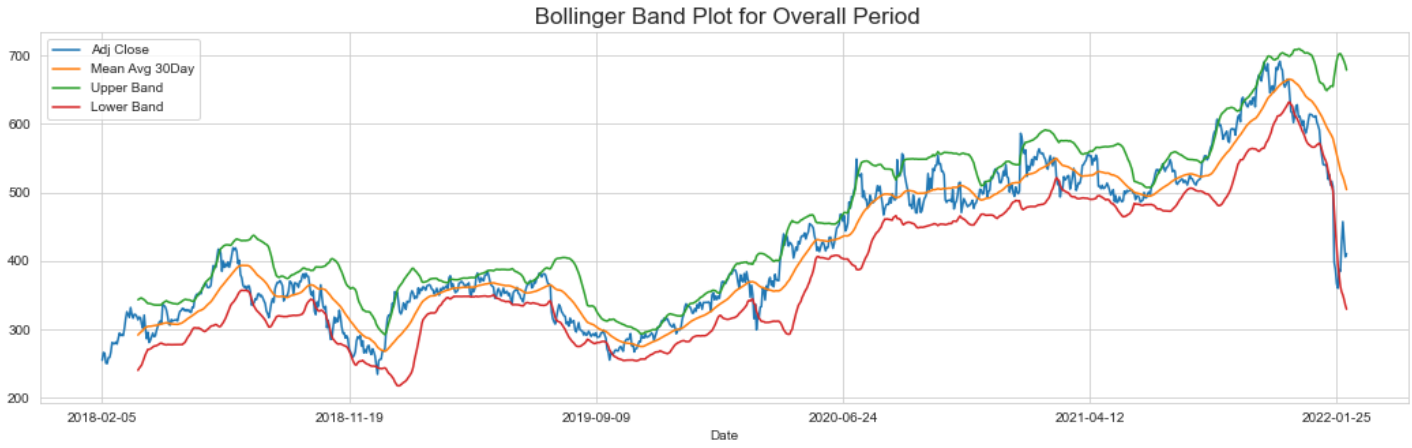
Out[23]:

	Open	High	Low	Close	Adj Close	Volume	Daily_returns	Cum_returns	Mean Avg 30Day	3(

Date	Open	High	Low	Close	Adj Close	Volume	Daily_returns	Cum_returns	Mean Avg 30Day	3(
Date										
2018-02-05	262.000000	267.899994	250.029999	254.259995	254.259995	11896100	NaN	NaN	NaN	
2018-02-06	247.699997	266.700012	245.000000	265.720001	265.720001	12595800	0.045072	1.045072	NaN	
2018-02-07	266.579987	272.450012	264.329987	264.559998	264.559998	8981500	-0.004366	1.040510	NaN	
2018-02-08	267.079987	267.619995	250.000000	250.100006	250.100006	9306700	-0.054657	0.983639	NaN	
2018-02-09	253.850006	255.800003	236.110001	249.470001	249.470001	16906900	-0.002519	0.981161	NaN	

In [24]:
```
df[['Adj Close', 'Mean Avg 30Day', 'Upper Band', 'Lower Band']].plot(figsize=(18,5))
plt.title(" Bollinger Band Plot for Overall Period", fontsize=17)
```

Out[24]:
```
Text(0.5, 1.0, ' Bollinger Band Plot for Overall Period')
```



Train and Test Data

Importing Models

In [29]:
```
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestRegressor
```

In [30]:
```
X= df[['Open', 'High', 'Low', 'Close', 'Volume']]
y= df['Adj Close']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

Scaling of Data

In [31]:
```
scale = StandardScaler()
```

```
X_train = scale.fit_transform(X_train)
X_test = scale.transform(X_test)
```

Applying Model

In [33]:

```
model = RandomForestRegressor(n_estimators=500, random_state=42, max_depth=10)
model.fit(X_train, y_train)
predict = model.predict(X_test)
print(predict)
print(predict.shape)
```

```
[553.97137601 379.24978571 361.31195786 282.40620018 261.1106334
 434.12840093 265.0824694  517.95360648 315.69806911 348.77217258
 520.26088534 321.34475264 489.8786001  298.20136405 510.71931092
 503.11331151 288.72631361 549.49048582 337.49758105 270.21613725
 338.57249553 294.20496034 462.41338583 369.04247224 290.21113387
 502.95015314 498.44456133 297.0068608  351.28284837 363.6167203
 280.9209422  542.83422599 316.88713674 540.81543363 303.79664093
 520.05826545 339.61662634 381.28551173 515.83044348 628.45031635
 482.24322233 297.94551988 305.69776403 264.88771103 515.96472066
 297.15404023 294.31643878 384.10761578 359.67359121 418.47030435
 365.85938115 608.07853329 480.62820511 512.64729849 364.59257334
 419.06538422 375.27641322 360.28349379 370.55831938 361.0713409
 294.90635352 493.30368374 491.94251831 405.99833518 325.16781697
 508.27718568 504.37618136 354.91632678 321.10632151 344.47837739
 656.26341881 351.3631924  263.21120194 267.71782703 427.07612088
 548.08470416 351.77787983 303.83561918 590.93918925 368.45776853
 501.78684241 503.02748522 519.67599252 644.53653508 359.95534932
 336.1220615  391.77168567 302.77127659 357.00125344 519.35087743
 527.40848991 309.41476923 363.1627575  490.89255863 294.19225196
 513.66135308 362.78936552 534.69211551 332.8676565  443.49725987
 541.05014337 553.19686049 309.31586026 548.15530374 452.05088434
 385.82845645 436.34717885 357.11225929 350.51186824 482.55745692
 329.7390701  332.75185819 503.76002028 361.3338861  520.22349697
 524.65766886 486.03727446 488.79588094 540.6653551  387.56502961
 419.45604713 521.92937724 488.78933403 540.83453928 339.12749894
 345.71629702 361.03327788 530.8343916  539.30789134 620.7171178
 313.21508135 263.14604983 566.31760383 524.67746413 345.21691134
 510.78536441 364.64315938 576.45209898 501.12790484 490.7439984
 310.59132662 274.86145693 279.89126007 332.14350624 486.86464932
 392.4666884  452.02181941 349.38724293 612.26053443 595.19043984
 494.58902712 354.4563283  585.63047722 306.92740625 320.09326207
 307.56304514 317.61906777 516.15447085 558.62657848 633.81473983
 353.2712668  606.6075837  447.2246604  493.32583723 530.8679601
 553.8081658  421.73876402 662.62206535 497.2601963  511.43872133
 686.28827204 345.6973585  370.86690081 497.90906118 419.63730533
 352.42087925 355.85935893 649.51702096 367.80520409 326.36526534
 559.15828403 607.08013509 340.86246453 364.12672876 271.39637418
 370.93320142 520.96506822 323.38237867 492.99625222 362.86182446
 361.34076827 339.22020421 524.65171791 378.97258497 367.55261366
 425.05496118 577.37904075 504.59611993 315.1966087  257.36117196
 304.4522007  409.74041822 539.64886353 513.85507973 484.27187012
 348.47783348 299.30129289 351.90459568 302.54505651 680.33380083
 363.90301448 398.43916485 374.61599099 381.1165247  518.85913348
 382.83435491 522.50822021 375.16022635 554.68170593 294.65596733
 482.39689916 605.94490882 394.35236394 365.35310835 364.64892183
 288.68164097 499.15881557 505.78288843 368.06394141 569.46951156
 358.76934094 272.00345144 487.31333488 502.92048821 363.73800911
 300.21388137 497.97266257 318.83638993 339.37978542 484.95048622
 344.49627139 320.00571299 353.53179494 488.29630668 318.04494286
 508.99157687 332.66345233 579.48838692 425.64604432 272.24870082
 546.82372802 366.5582592  302.4490451  510.17202966 264.70409585
 326.37065682 329.19942371 371.66235143 452.53997891 627.90236258
 280.61187013 686.7686079  507.85563171 486.60487111 358.9422981
```

```
 500.65781874 482.61062362 598.22334906 332.89435412 512.2459258
 338.20314363 500.05957767 418.64052632 335.388636   377.6995543
 495.80496987 315.16301493 326.27585116 508.87777796 267.72043028
 326.47247143 472.55505312 343.32626059 331.61348866 540.54677184
 547.87958534 283.31514501 536.68639714 297.83161678 367.7302911
 251.05003551 523.89865034 275.97951229 532.43045925 614.3317342
 533.03198169 377.32039555 485.85218587 546.88182526 410.68753675
 278.80078967 406.71162093 328.57601983]
(303,)
```

Statistical Metrics and Performance Evalution

In [34]:

```
print("Mean Absolute Error:", round(metrics.mean_absolute_error(y_test, predict), 4))
print("Mean Squared Error:", round(metrics.mean_squared_error(y_test, predict), 4))
print("Root Mean Squared Error:", round(np.sqrt(metrics.mean_squared_error(y_test, predict
print("(R^2) Score:", round(metrics.r2_score(y_test, predict), 4))
print(f'Train Score : {model.score(X_train, y_train) * 100:.2f}% and Test Score : {model.s
errors = abs(predict - y_test)
mape = 100 * (errors / y_test)
accuracy = 100 - np.mean(mape)
print('Accuracy:', round(accuracy, 2), '%.')
```

```
Mean Absolute Error: 0.6654
Mean Squared Error: 2.2328
Root Mean Squared Error: 1.4943
(R^2) Score: 0.9998
Train Score : 100.00% and Test Score : 99.98% using Random Tree Regressor.
Accuracy: 99.85 %.
```

Predictions

In [35]:

```
predictions = pd.DataFrame({"Predictions": predict}, index=pd.date_range(start=df.index[-1
predictions.to_csv("Predicted-price-data.csv")

#collecting future days from predicted values
oneyear_df = pd.DataFrame(predictions[:252])
oneyear_df.to_csv("one-year-predictions.csv")
onemonth_df = pd.DataFrame(predictions[:21])
onemonth_df.to_csv("one-month-predictions.csv")
fivedays_df = pd.DataFrame(predictions[:5])
fivedays_df.to_csv("five-days-predictions.csv")
```

In [36]:

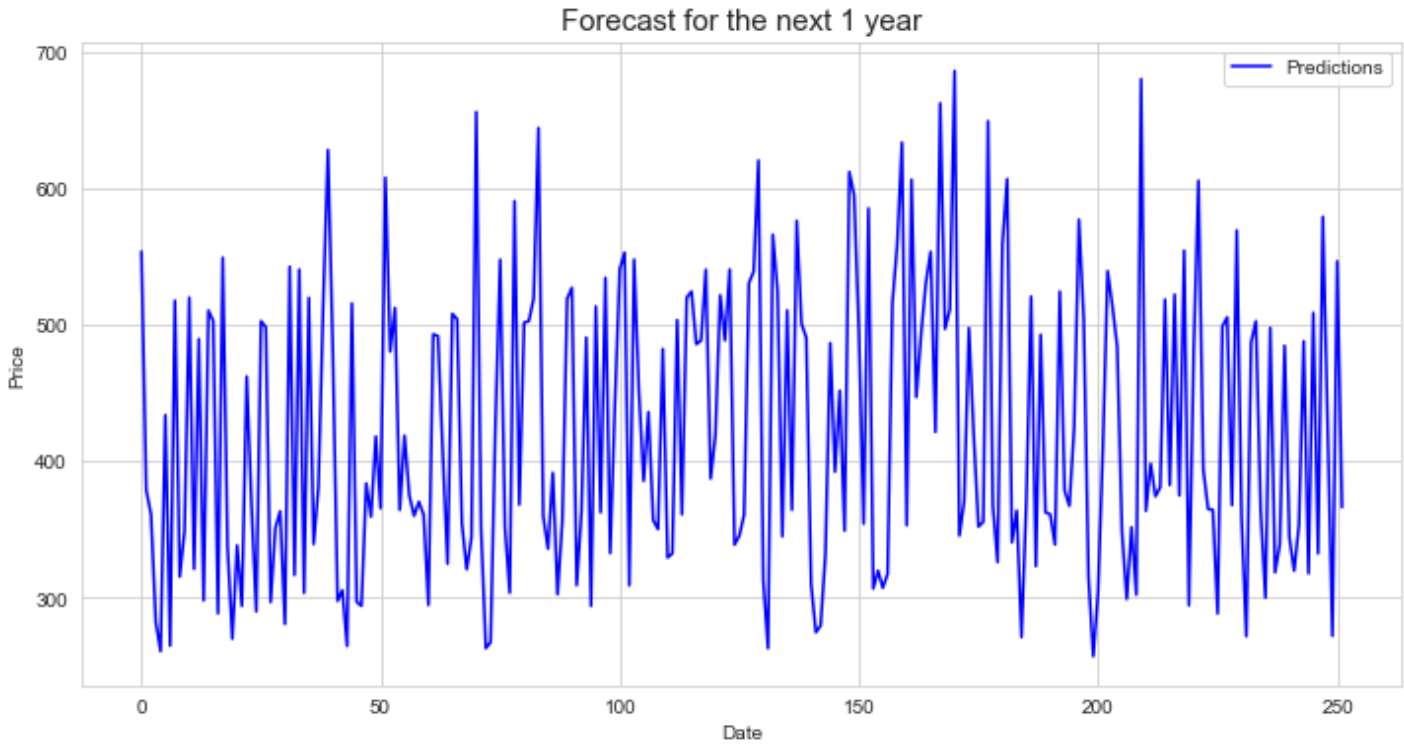
```
print(predictions)
```

```
            Predictions
2022-02-04   553.971376
2022-02-05   379.249786
2022-02-06   361.311958
2022-02-07   282.406200
2022-02-08   261.110633
...                 ...
2022-11-29   546.881825
2022-11-30   410.687537
2022-12-01   278.800790
2022-12-02   406.711621
2022-12-03   328.576020

[303 rows x 1 columns]
```

One Year Prediction

In [37]:
```
oneyear_df_pred = pd.read_csv("one-year-predictions.csv")
buy_price = min(oneyear_df_pred["Predictions"])
sell_price = max(oneyear_df_pred["Predictions"])
oneyear_buy = oneyear_df_pred.loc[oneyear_df_pred["Predictions"] == buy_price]
oneyear_sell = oneyear_df_pred.loc[oneyear_df_pred["Predictions"] == sell_price]
print("Buy Date and Price of Stock")
print(oneyear_buy,'\n')
print("Sell Date and Price of stock")
print(oneyear_sell)
oneyear_df_pred["Predictions"].plot(figsize=(12, 6), color="blue")
plt.title("Forecast for the next 1 year", size=15)
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.show()
```

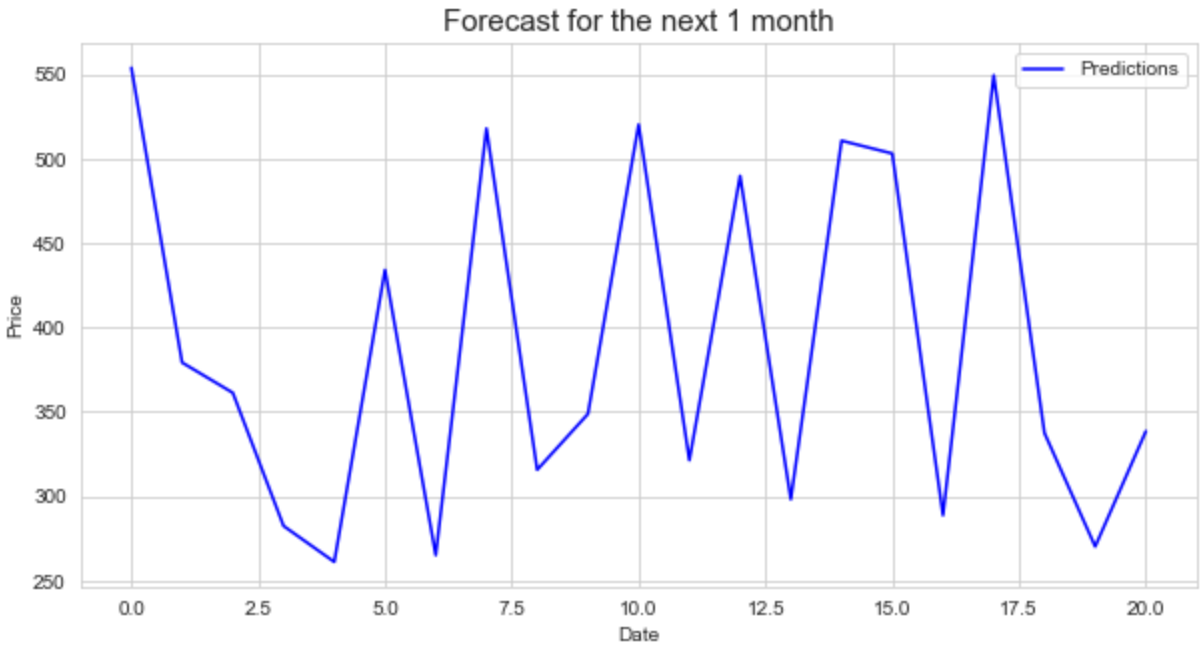
```
Buy Date and Price of Stock
     Unnamed: 0  Predictions
199  2022-08-22   257.361172

Sell Date and Price of stock
     Unnamed: 0  Predictions
170  2022-07-24   686.288272
```



One Month Prediction

In [38]:
```
onemonth_df_pred = pd.read_csv("one-month-predictions.csv")
buy_price = min(onemonth_df_pred["Predictions"])
sell_price = max(onemonth_df_pred["Predictions"])
onemonth_buy = onemonth_df_pred.loc[onemonth_df_pred["Predictions"] == buy_price]
onemonth_sell = onemonth_df_pred.loc[onemonth_df_pred["Predictions"] == sell_price]
print("Buy price and date")
print(onemonth_buy,'\n')
print("Sell price and date")
print(onemonth_sell)
onemonth_df_pred["Predictions"].plot(figsize=(10, 5), color="blue")
plt.title("Forecast for the next 1 month", size=15)
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.show()
```

```
Buy price and date
   Unnamed: 0  Predictions
4  2022-02-08   261.110633

Sell price and date
   Unnamed: 0  Predictions
0  2022-02-04   553.971376
```

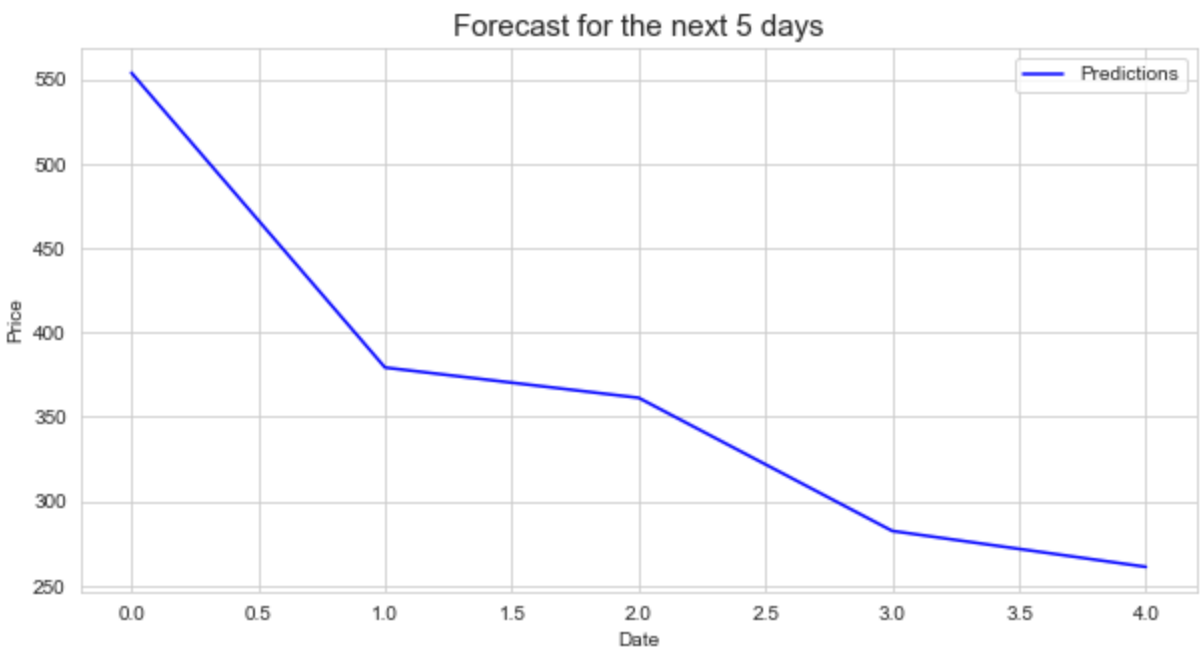


5 Days Prediction

In [39]:

```
fivedays_df_pred = pd.read_csv("five-days-predictions.csv")
buy_price = min(fivedays_df_pred["Predictions"])
sell_price = max(fivedays_df_pred["Predictions"])
fivedays_buy = fivedays_df_pred.loc[fivedays_df_pred["Predictions"] == buy_price]
fivedays_sell = fivedays_df_pred.loc[fivedays_df_pred["Predictions"] == sell_price]
print("Buy price and date")
print(fivedays_buy,'\n')
print("Sell price and date")
print(fivedays_sell)
fivedays_df_pred["Predictions"].plot(figsize=(10, 5), color="blue")
plt.title("Forecast for the next 5 days",size=15)
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend()
plt.show()
```

```
Buy price and date
   Unnamed: 0  Predictions
4  2022-02-08   261.110633

Sell price and date
   Unnamed: 0  Predictions
0  2022-02-04   553.971376
```



Assignment Completed

In [ ]: