In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.metrics import mean_absolute_percentage_error
from sklearn.model_selection import train_test_split
import yfinance as yf
import warnings
warnings.filterwarnings('ignore')
```

```
C:\Users\Toral Dave\anaconda3\lib\site-packages\scipy\__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version
1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

In [177]:
```
df = pd.read_excel(r"C:\\Users\Toral Dave\OneDrive - Adani Institute for Education and Research\TRIMESTER-IV\MLATA\Assignment\Final Term\Stock Prices Data.xlsx")
```

In [119]:
```
df
```

Out[119]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2021-09-03	36883.648438	37140.000000	36563.199219	36761.148438	36760.722656	155600.0
1	2021-09-06	36878.250000	36923.648438	36554.449219	36592.351563	36591.925781	88600.0
2	2021-09-07	36558.851563	36685.851563	36151.949219	36468.800781	36468.375000	124500.0
3	2021-09-08	36519.699219	36855.898438	36393.601563	36768.199219	36767.773438	175800.0
4	2021-09-09	36725.500000	36857.199219	36566.699219	36683.199219	36682.773438	83600.0
...	...	...	...	...	...	...	...
491	2023-08-28	44253.648438	44610.398438	44201.449219	44494.648438	44494.648438	163000.0
492	2023-08-29	44655.750000	44673.000000	44429.800781	44495.250000	44495.250000	140300.0
493	2023-08-30	44706.550781	44779.648438	44149.800781	44232.601563	44232.601563	153100.0
494	2023-08-31	44265.851563	44399.648438	43895.050781	43989.148438	43989.148438	705100.0
495	2023-09-01	43996.101563	44568.550781	43830.750000	44436.101563	44436.101563	320800.0

In [120]:
```
df.describe()
```

Out[120]:

	Open	High	Low	Close	Adj Close	Volume
count	495.000000	495.000000	495.000000	495.000000	495.000000	4.950000e+02
mean	39563.160807	39822.913858	39255.565594	39545.088656	39544.796240	3.831669e+06
std	3437.401881	3385.877411	3461.912489	3422.392835	3422.520923	8.080975e+07
min	32393.449219	32889.800781	32155.349609	32617.099609	32616.720703	0.000000e+00
25%	36796.900390	37061.275390	36480.349610	36802.826172	36802.398438	1.486000e+05
50%	39422.300781	39645.199219	39120.648438	39395.351563	39395.351563	1.823000e+05
75%	42645.849610	42724.925781	42322.875000	42559.025390	42558.777344	2.333500e+05
max	46285.851563	46369.500000	45925.898438	46186.898438	46186.898438	1.798102e+09

In [121]:
```
df.head()
```

Out[121]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	2021-09-03	36883.648438	37140.000000	36563.199219	36761.148438	36760.722656	155600.0
1	2021-09-06	36878.250000	36923.648438	36554.449219	36592.351563	36591.925781	88600.0
2	2021-09-07	36558.851563	36685.851563	36151.949219	36468.800781	36468.375000	124500.0
3	2021-09-08	36519.699219	36855.898438	36393.601563	36768.199219	36767.773438	175800.0
4	2021-09-09	36725.500000	36857.199219	36566.699219	36683.199219	36682.773438	83600.0

In [122]:
```
df.tail()
```

Out[122]:

	Date	Open	High	Low	Close	Adj Close	Volume
491	2023-08-28	44253.648438	44610.398438	44201.449219	44494.648438	44494.648438	163000.0
492	2023-08-29	44655.750000	44673.000000	44429.800781	44495.250000	44495.250000	140300.0
493	2023-08-30	44706.550781	44779.648438	44149.800781	44232.601563	44232.601563	153100.0
494	2023-08-31	44265.851563	44399.648438	43895.050781	43989.148438	43989.148438	705100.0
495	2023-09-01	43996.101563	44568.550781	43830.750000	44436.101563	44436.101563	320800.0

In [123]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 496 entries, 0 to 495
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype         
---  ------     --------------  -----         
 0   Date       496 non-null    datetime64[ns]
 1   Open       495 non-null    float64       
 2   High       495 non-null    float64       
 3   Low        495 non-null    float64       
 4   Close      495 non-null    float64       
 5   Adj Close  495 non-null    float64       
 6   Volume     495 non-null    float64       
dtypes: datetime64[ns](1), float64(6)
memory usage: 27.2 KB
```

In [178]:
```
# Dropping null and duplicates
df.dropna(inplace = True)
df.drop_duplicates(inplace = True)
```

In [125]:
```
#df.sort_values(['Date'], ascending=False, inplace = True, ignore_index= True)
```

In [126]:
```
# Set index as Date
df.set_index(['Date'], inplace= True)
```

In [127]:
```
df
```

Out[127]:

Date	Open	High	Low	Close	Adj Close	Volume
2021-09-03	36883.648438	37140.000000	36563.199219	36761.148438	36760.722656	155600.0
2021-09-06	36878.250000	36923.648438	36554.449219	36592.351563	36591.925781	88600.0
2021-09-07	36558.851563	36685.851563	36151.949219	36468.800781	36468.375000	124500.0
2021-09-08	36519.699219	36855.898438	36393.601563	36768.199219	36767.773438	175800.0
2021-09-09	36725.500000	36857.199219	36566.699219	36683.199219	36682.773438	83600.0
...	...	...	...	...	...	...
2023-08-28	44253.648438	44610.398438	44201.449219	44494.648438	44494.648438	163000.0
2023-08-29	44655.750000	44673.000000	44429.800781	44495.250000	44495.250000	140300.0
2023-08-30	44706.550781	44779.648438	44149.800781	44232.601563	44232.601563	153100.0
2023-08-31	44265.851563	44399.648438	43895.050781	43989.148438	43989.148438	705100.0
2023-09-01	43996.101563	44568.550781	43830.750000	44436.101563	44436.101563	320800.0

495 rows × 6 columns

In [176]:
```
#Plot the Close Value
df_date['Close'].plot()
plt.xlabel("Date")
plt.ylabel("Closing Price")
plt.title("Closing Price Over Time", fontsize = 30)
plt.show()
```

1st MODEL

LINEAR REGRESSION MODEL

In [131]:
```
X = df.drop(columns= ['Close', 'Adj Close'])
Y = df['Close']
```

TRAIN_TEST_SPLIT

In [132]:
```
# Splitting into Train and test
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= 0.2, random_state= 0)
```

In [136]:
```
Y_test
```

Out[136]:
```
Date
2022-01-13    38469.949219
2022-09-13    40873.101563
2022-10-25    41122.750000
2023-06-19    43633.750000
2023-01-11    42232.699219
                  ...
2022-05-26    35094.898438
2023-03-03    41251.351563
2021-11-26    36025.500000
2023-06-12    43944.199219
2021-12-02    36508.250000
Name: Close, Length: 99, dtype: float64
```

In [137]:
```
model = LinearRegression()
model.fit(X_train, Y_train)
```

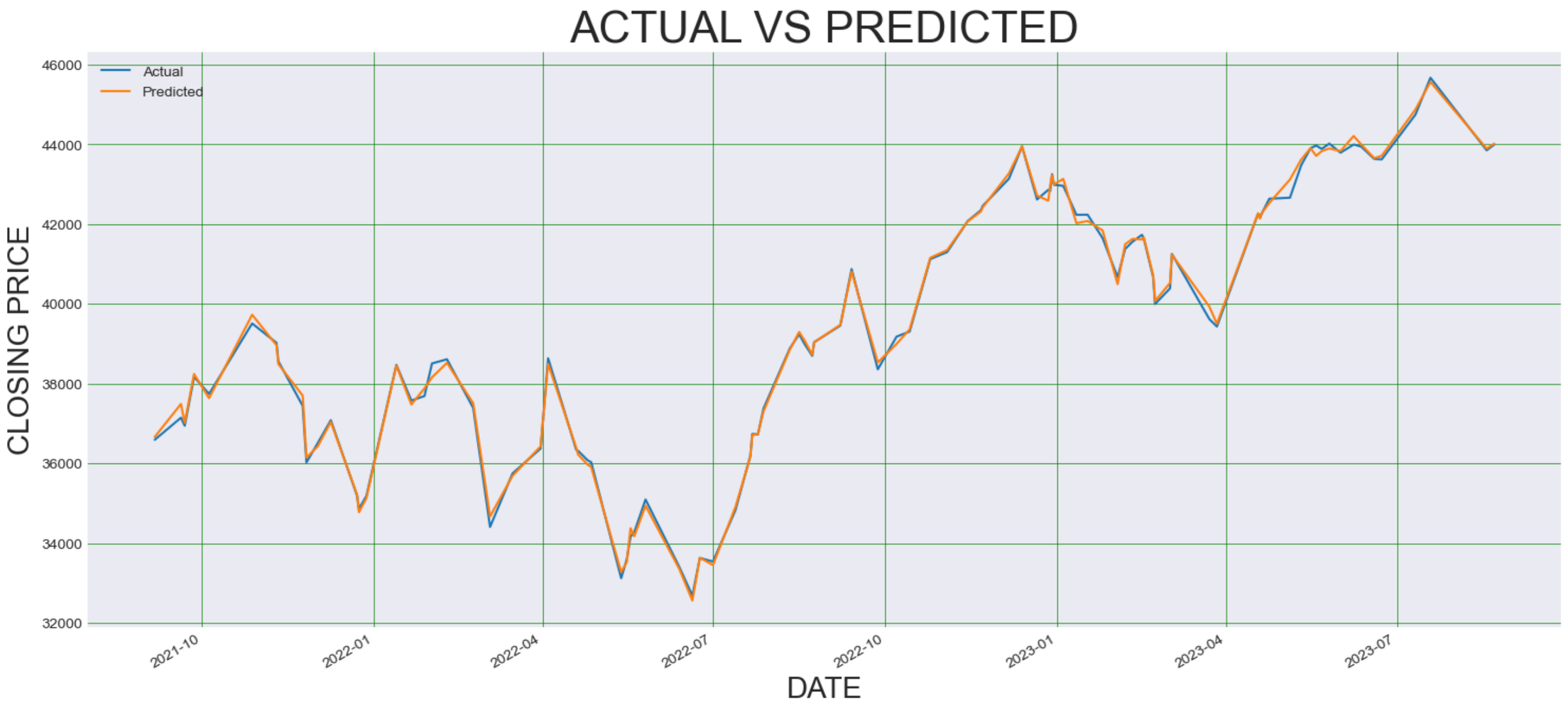
Out[137]:
```
LinearRegression
LinearRegression()
```

In [166]:
```
Y_pred = model.predict(X_test)
Y_pred
```

Out[166]:
```
array([38454.74967451, 40810.07765322, 41154.90181008, 43651.97430626,
       42023.32208335, 38241.22995725, 43945.8102545 , 43892.60114897,
       35906.94338068, 36392.14184814, 39033.01583205, 39476.4474023 ,
       39502.20582673, 41347.91007465, 33527.25247802, 43231.26761068,
       45563.39684834, 41493.63419676, 36220.52907933, 39726.92970821,
       33451.31583931, 40712.89778775, 38724.32354878, 38520.89777555,
       38963.31359637, 44208.84090416, 37636.14722515, 40069.15649843,
       37475.3550283 , 38849.40900868, 39929.42650333, 37510.76792386,
       34668.06827453, 33357.19142311, 40526.85428967, 41629.11114113,
       43270.35782339, 43712.85292915, 40494.96991582, 39075.8117923 ,
       42715.64205705, 38490.9915549 , 42139.84351615, 36736.91365454,
       34775.15629505, 36663.50311321, 34906.51945829, 42530.12674449,
       43618.72616005, 38156.22207092, 41666.84529451, 33272.72975043,
       43831.56680787, 38496.54138499, 35688.82534674, 37016.16032658,
       42967.68793805, 34373.42005753, 37306.33957325, 42076.7100083 ,
       35130.2761893 , 39361.05689893, 43116.79355665, 43831.17504396,
       42418.12530755, 32561.27355467, 37489.94482104, 42276.94516332,
       38993.96943026, 35216.42885521, 36427.83205885, 37039.99122544,
       44873.32633513, 43133.15474244, 34174.02368139, 41614.7376564 ,
       38531.02405843, 37881.28922041, 44015.60063702, 42067.44396296,
       42314.4239657 , 39294.93955932, 43896.14179697, 36172.07376473,
       42991.6677241 , 42586.74980884, 43902.9194993 , 35974.10984961,
       41840.4067417 , 37698.26988622, 36713.51534038, 42268.96620358,
       43711.49735748, 33632.97319417, 34939.3391769 , 41233.13199161,
       36144.78176636, 43998.68208067, 36426.58477843])
```

PLotting Actual vs Predicted

In [167]:
```
df_plot = pd.DataFrame({'Actual': Y_test, 'Predicted': Y_pred})
df_plot.plot(kind='line',figsize=(18,8))
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.title("ACTUAL VS PREDICTED", fontsize = 30)
plt.xlabel("DATE", fontsize = 20)
plt.ylabel("CLOSING PRICE", fontsize = 20)
plt.show()
```



In [168]:
```
mse = mean_squared_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)

print("Mean Absolute Error:", mse)
print("R squared score:", r2)
```

```
Mean Absolute Error: 17367.851482829225
R squared score: 0.9984328677684712
```

In [181]:
```
'''def predict_price(date, open_price):
    date = pd.to_datetime(date).timestamp()
    return model.predict([[date, open_price]])

date = input("Enter date in yyyy-mm-dd format:")
open_price = float(input("Enter openinig price:"))

predicted_close_price = predict_price(date, open_price)

print("The predicted closing price for the date {date_str} and opening price {open_price} is {predicted_close_price}")
'''
```

Out[181]:
```
'def predict_price(date, open_price):\n    date = pd.to_datetime(date).timestamp()\n    return model.predict([[date, open_price]])\n\ndate = input("Enter date in yyyy-mm-dd form
at:")\nopen_price = float(input("Enter openinig price:"))\n\npredicted_close_price = predict_price(date, open_price)\n\nprint("The predicted closing price for the date {date_st
r} and opening price {open_price} is {predicted_close_price}")\n'
```

2nd MODEL

Support Vector Machine

In [144]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
plt.style.use('seaborn-darkgrid')
```

In [174]:
```
# Create predictor variables
df['Open-Close'] = df.Open - df.Close
df['High-Low'] = df.High - df.Low

# Store all predictor variables in a variable x
x = df[['Open-Close', 'High-Low']]
x.head()
```

Out[174]:

Date	Open-Close	High-Low
2021-09-03	122.500000	576.800781
2021-09-06	285.898437	369.199219
2021-09-07	90.050782	533.902344
2021-09-08	-248.500000	462.296875
2021-09-09	42.300781	290.500000

In [142]:
```
# Target variables
y = np.where(df['Close'].shift(-1) > df['Close'], 1, 0)
y
```

Out[142]:
```
array([0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1,
       0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1,
       0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0,
       0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0,
       1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0])
```

In [143]:
```
split_percentage = 0.8
split = int(split_percentage*len(df))

# Train data set
x_train = x[:split]
y_train = y[:split]

# Test data set
x_test = x[split:]
y_test = y[split:]
```

We will use SVC() function from sklearn.svm.SVC library to create our classifier model using the fit() method on the training data set.

In [146]:
```
# Support vector classifier
cls = SVC().fit(x_train, y_train)
```

Strategy implementation

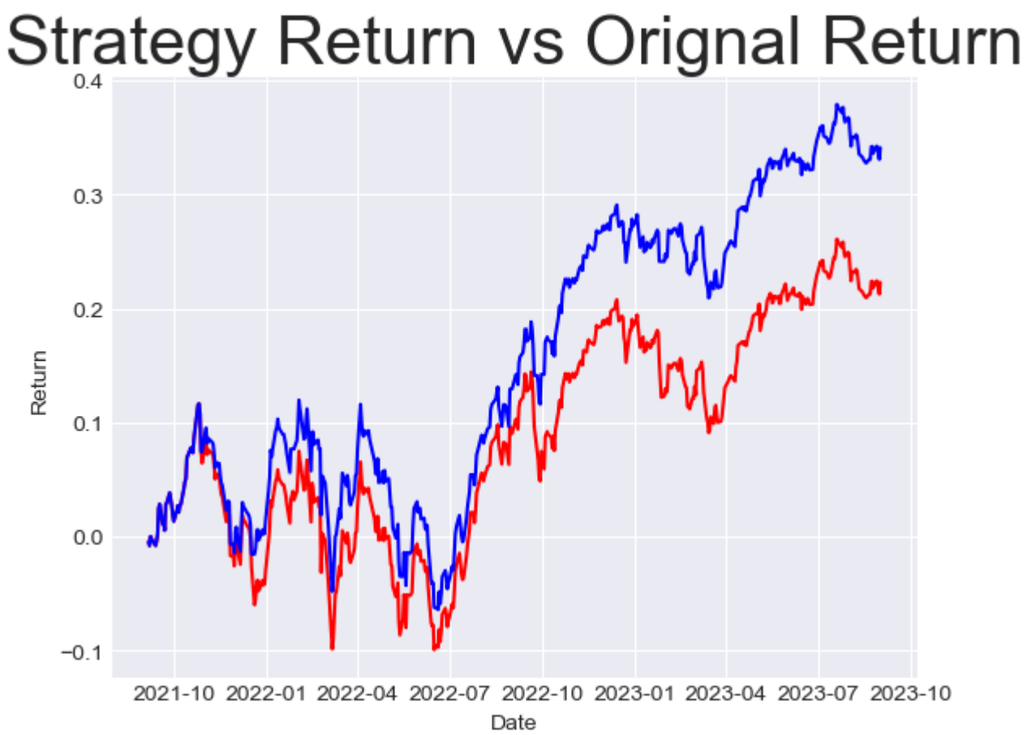
We will predict the signal (buy or sell) using the cls.predict() function.

In [170]:
```
y_pred = cls.predict(x_test)
```

In [152]:
```
df['Predicted_Signal'] = cls.predict(x)

# Calculate daily returns
df['Return'] = df.Close.pct_change()

# Calculate strategy returns
df['Strategy_Return'] = df.Return *df.Predicted_Signal.shift(1)

# Calculate Cumulutive returns
df['Cum_Ret'] = df['Return'].cumsum()

# Plot Strategy Cumulative returns
df['Cum_Strategy'] = df['Strategy_Return'].cumsum()
```

Plotting Strategy Return vs Orignal Return

In [159]:
```
plt.plot(df['Cum_Ret'],color='red')
plt.plot(df['Cum_Strategy'],color='blue')
plt.xlabel("Date")
plt.ylabel("Return")
plt.title("Strategy Return vs Orignal Return", fontsize = 30)
plt.show()
```



In [172]:
```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error:", mse)
print("R squared score:", r2)
```

```
Mean Absolute Error: 0.46464646464646464
R squared score: -0.8679245283018868
```

----------THE END ----------