In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:

```
df=pd.read_csv(r"C:\Users\HP\OneDrive\Documents\prices-split-adjusted.csv")
df
```

Out[3]:

	date	symbol	open	close	low	high	volume
0	2016-01-05	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0
1	2016-01-06	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0
2	2016-01-07	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0
3	2016-01-08	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0
4	2016-01-11	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0
...	...	...	...	...	...	...	...
851259	2016-12-30	ZBH	103.309998	103.199997	102.849998	103.930000	973800.0
851260	2016-12-30	ZION	43.070000	43.040001	42.689999	43.310001	1938100.0
851261	2016-12-30	ZTS	53.639999	53.529999	53.270000	53.740002	1701200.0
851262	2016-12-30	AIV	44.730000	45.450001	44.410000	45.590000	1380900.0
851263	2016-12-30	FTV	54.200001	53.630001	53.389999	54.480000	705100.0

851264 rows × 7 columns

In [4]:

```
df.head()
```

Out[4]:

	date	symbol	open	close	low	high	volume
0	2016-01-05	WLTW	123.430000	125.839996	122.309998	126.250000	2163600.0
1	2016-01-06	WLTW	125.239998	119.980003	119.940002	125.540001	2386400.0
2	2016-01-07	WLTW	116.379997	114.949997	114.930000	119.739998	2489500.0
3	2016-01-08	WLTW	115.480003	116.620003	113.500000	117.440002	2006300.0
4	2016-01-11	WLTW	117.010002	114.970001	114.089996	117.330002	1408600.0

In [5]:

```
df.describe()
```

Out[5]:

	open	close	low	high	volume
count	851264.000000	851264.000000	851264.000000	851264.000000	8.512640e+05
mean	64.993618	65.011913	64.336541	65.639748	5.415113e+06
std	75.203893	75.201216	74.459518	75.906861	1.249468e+07
min	1.660000	1.590000	1.500000	1.810000	0.000000e+00
25%	31.270000	31.292776	30.940001	31.620001	1.221500e+06
50%	48.459999	48.480000	47.970001	48.959999	2.476250e+06
75%	75.120003	75.139999	74.400002	75.849998	5.222500e+06
max	1584.439941	1578.130005	1549.939941	1600.930054	8.596434e+08

In [6]:

```
for i in df.columns:
  print(i,"\t-\t", df[i].isna().mean()*100)
```

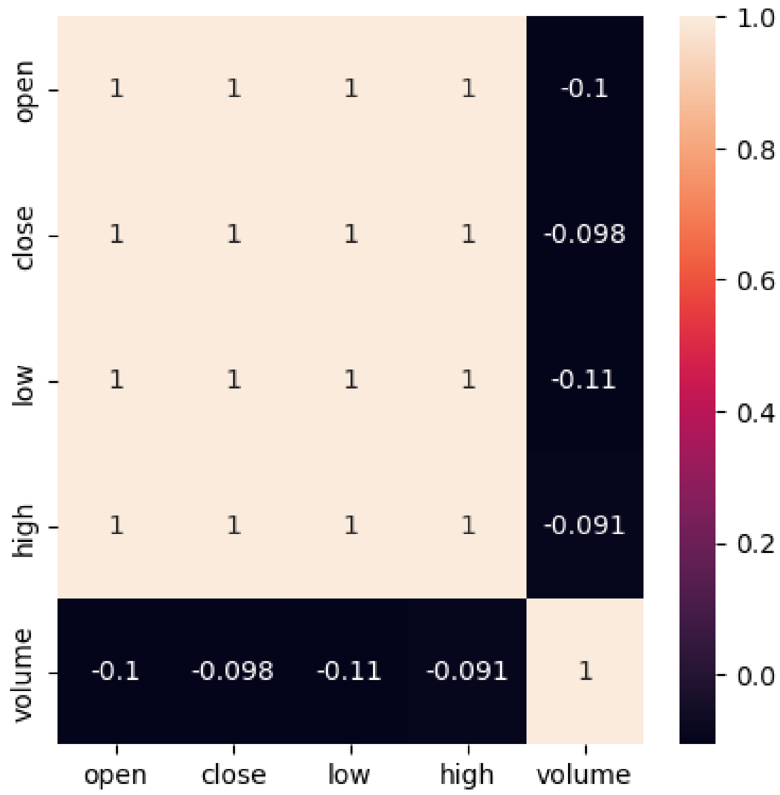
```
date    -       0.0
symbol  -       0.0
open    -       0.0
close   -       0.0
low     -       0.0
high    -       0.0
volume  -       0.0
```

In [7]:

```
df = df[df['symbol']=='AAP']   # Choosin stock values for any company

cormap = df.corr()
fig, ax = plt.subplots(figsize=(5,5))
sns.heatmap(cormap, annot = True)
```

Out[7]:

```
<AxesSubplot:>
```



In [8]:

```
def get_corelated_col(cor_dat, threshold):
  # Cor_data to be column along which corelation to be measured
  #Threshold be the value above which of corelation to considered
  feature=[]
  value=[]

  for i ,index in enumerate(cor_dat.index):
    if abs(cor_dat[index]) > threshold:
      feature.append(index)
      value.append(cor_dat[index])

  df = pd.DataFrame(data = value, index = feature, columns=['corr value'])
  return df
```

In [9]:

```
top_corelated_values = get_corelated_col(cormap['close'], 0.60)
top_corelated_values
```

Out[9]:

	corr value
open	0.999382
close	1.000000
low	0.999615
high	0.999737

In [10]:

```
df = df[top_corelated_values.index]
df.head()
```

Out[10]:

	open	close	low	high
253	40.700001	40.380001	40.360001	41.040001
720	40.299999	40.139999	39.720001	40.310001
1188	40.049999	40.490002	40.049999	40.779999
1656	39.549999	40.480000	39.549999	40.540001
2124	40.250000	40.639999	40.110001	40.820000

In [11]:

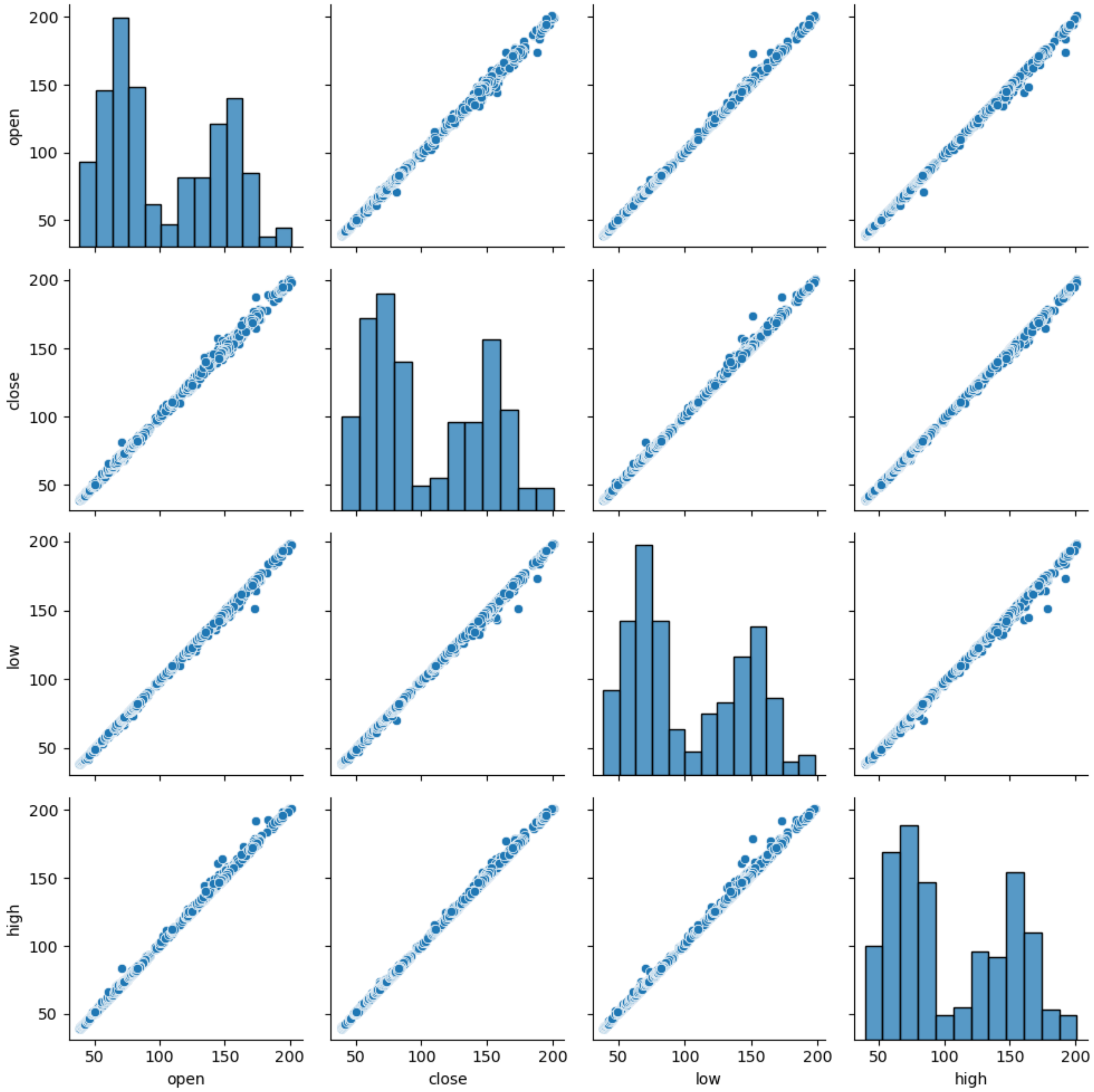
```
df.shape
```

Out[11]:

```
(1762, 4)
```

In [12]:

```
sns.pairplot(df)
plt.tight_layout()
```



In [131]:

```
X = df.drop(['close'], axis=1)
y = df['close']
y
```

Out[131]:

```
253        40.380001
720        40.139999
1188       40.490002
1656       40.480000
2124       40.639999
             ...
848766    170.889999
849266    171.839996
849766    170.419998
850266    170.279999
850766    169.119995
Name: close, Length: 1762, dtype: float64
```

In [132]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X.head()
```

Out[132]:

	open	low	high
0	0.012001	0.012392	0.010256
1	0.009539	0.008387	0.005746
2	0.008000	0.010452	0.008649
3	0.004923	0.007323	0.007167
4	0.009231	0.010827	0.008897

In [133]:

```
from sklearn.model_selection import train_test_split

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

Acc = []
```

In [134]:

```
from sklearn.linear_model import LinearRegression

# model training

model_1 = LinearRegression()
model_1.fit(X_train, y_train)
```

Out[134]:

```
LinearRegression()
```

In [135]:

```
# prediction
y_pred_1 = model_1.predict(X_test)
pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_1})
pred_df.head()
```

Out[135]:

	Actual	Predicted
675111	173.660004	173.682489
675608	171.919998	172.593759
676105	172.000000	171.182789
676602	187.789993	187.980305
677099	187.029999	188.440838

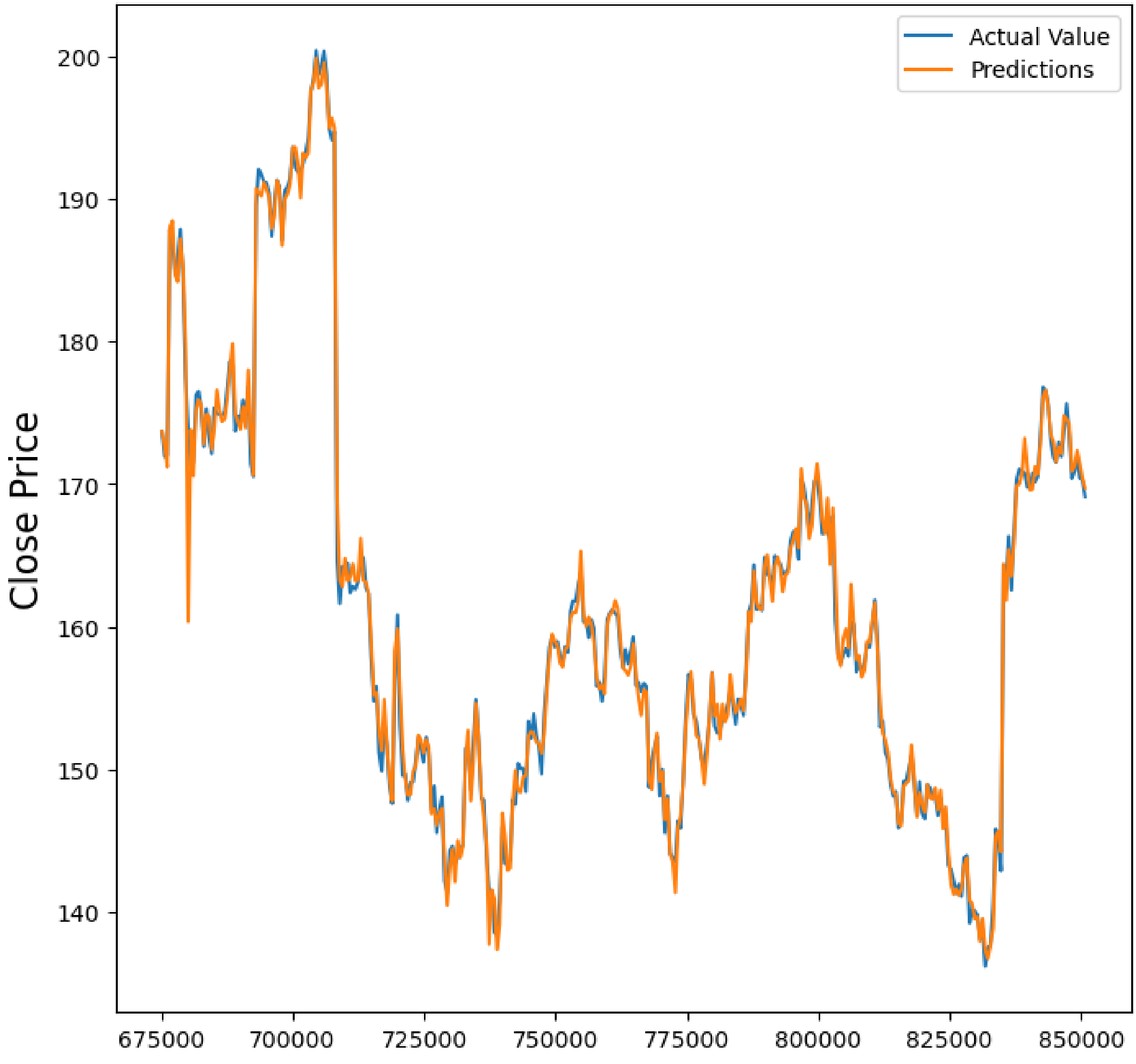
In [136]:

```
# Measure the Accuracy Score

from sklearn.metrics import r2_score

print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_1)))
Acc.append(r2_score(y_test, y_pred_1))
```

```
Accuracy score of the predictions: 0.9931342019332019
```

In [137]:

```
plt.figure(figsize=(8,8))
plt.ylabel('Close Price', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



In [138]:

```
from keras.models import Sequential
from keras.layers import Dense

def regressor(inp_dim):

    model = Sequential()

    model.add(Dense(20, input_dim=inp_dim, kernel_initializer='normal', activation='relu'))
    model.add(Dense(25, kernel_initializer='normal', activation='relu'))
    model.add(Dense(10, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))
    model.compile(loss='mean_squared_error', optimizer='adam')

    return model
```

In [139]:

```
model_2 = regressor(inp_dim=3)
model_2.fit(X_train, y_train, epochs=70, validation_split=0.2)
```

```
Epoch 30/70
36/36 [==============================] - 0s 5ms/step - loss: 2.4381 - val_loss:
19.4836
Epoch 31/70
36/36 [==============================] - 0s 4ms/step - loss: 2.1004 - val_loss:
18.3121
Epoch 32/70
36/36 [==============================] - 0s 4ms/step - loss: 1.8216 - val_loss:
14.1593
Epoch 33/70
36/36 [==============================] - 0s 4ms/step - loss: 1.5828 - val_loss:
12.2309
Epoch 34/70
36/36 [==============================] - 0s 4ms/step - loss: 1.3829 - val_loss:
10.7492
Epoch 35/70
36/36 [==============================] - 0s 4ms/step - loss: 1.2131 - val_loss:
9.1283
Epoch 36/70
```

In [140]:

```
y_pred_2 = model_2.predict(X_test)
```

```
12/12 [==============================] - 0s 2ms/step
```

In [141]:

```
pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_2.flatten()})
pred_df.head()
```

Out[141]:

	Actual	Predicted
675111	173.660004	174.169403
675608	171.919998	172.486450
676105	172.000000	170.709076
676602	187.789993	179.633133
677099	187.029999	188.241119

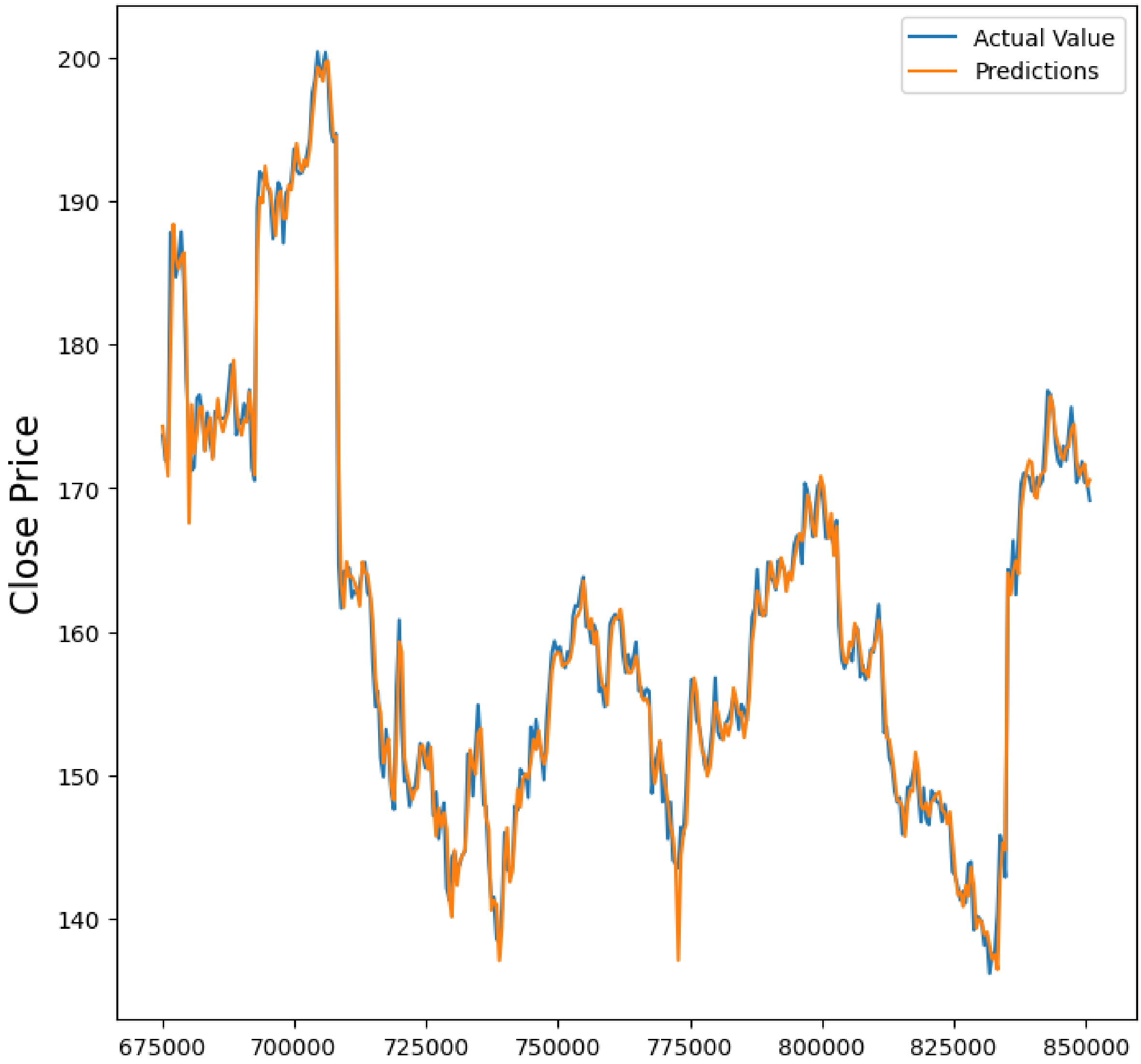
In [24]:

```
# Measure the Accuracy Score

from sklearn.metrics import r2_score

print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_2)))
Acc.append(r2_score(y_test, y_pred_2))
```

```
Accuracy score of the predictions: 0.9880037047512 34
```

In [25]:

```
plt.figure(figsize=(8,8))
plt.ylabel('Close Price', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



In [26]:

```
X_train = np.array(X_train).reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = np.array(X_test).reshape(X_test.shape[0], X_test.shape[1], 1)
```

In [27]:

```
from tensorflow.keras import Sequential,utils
from tensorflow.keras.layers import Flatten, Dense, Conv1D, MaxPool1D, Dropout

def reg():

    model = Sequential()

    model.add(Conv1D(32, kernel_size=(3,), padding='same', activation='relu', input_shape = (X_tı
    model.add(Conv1D(64, kernel_size=(3,), padding='same', activation='relu'))
    model.add(Conv1D(128, kernel_size=(5,), padding='same', activation='relu'))

    model.add(Flatten())

    model.add(Dense(50, activation='relu'))
    model.add(Dense(20, activation='relu'))
    model.add(Dense(units = 1))

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

    return model
```

In [28]:

```
model_3 = reg()
model_3.fit(X_train, y_train, epochs=100, validation_split=0.2)
```

```
Epoch 62/100
36/36 [==============================] - 0s 7ms/step - loss: 0.4262 - val_loss:
2.0515
Epoch 63/100
36/36 [==============================] - 0s 7ms/step - loss: 0.4391 - val_loss:
1.5805
Epoch 64/100
36/36 [==============================] - 0s 7ms/step - loss: 0.4408 - val_loss:
1.5751
Epoch 65/100
36/36 [==============================] - 0s 7ms/step - loss: 0.4508 - val_loss:
2.1747
Epoch 66/100
36/36 [==============================] - 0s 7ms/step - loss: 0.5008 - val_loss:
1.5656
Epoch 67/100
36/36 [==============================] - 0s 7ms/step - loss: 0.4448 - val_loss:
1.6339
Epoch 68/100
36/36 [==============================] - 0s 7ms/step - loss: 0.5152 - val_loss:
```

In [29]:

```
y_pred_3 = model_3.predict(X_test)
```

```
12/12 [==============================] - 0s 3ms/step
```

In [30]:

```
pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_3.flatten()})
pred_df.head()
```

Out[30]:

	Actual	Predicted
675111	173.660004	173.509415
675608	171.919998	171.910919
676105	172.000000	170.124786
676602	187.789993	179.451736
677099	187.029999	187.614029

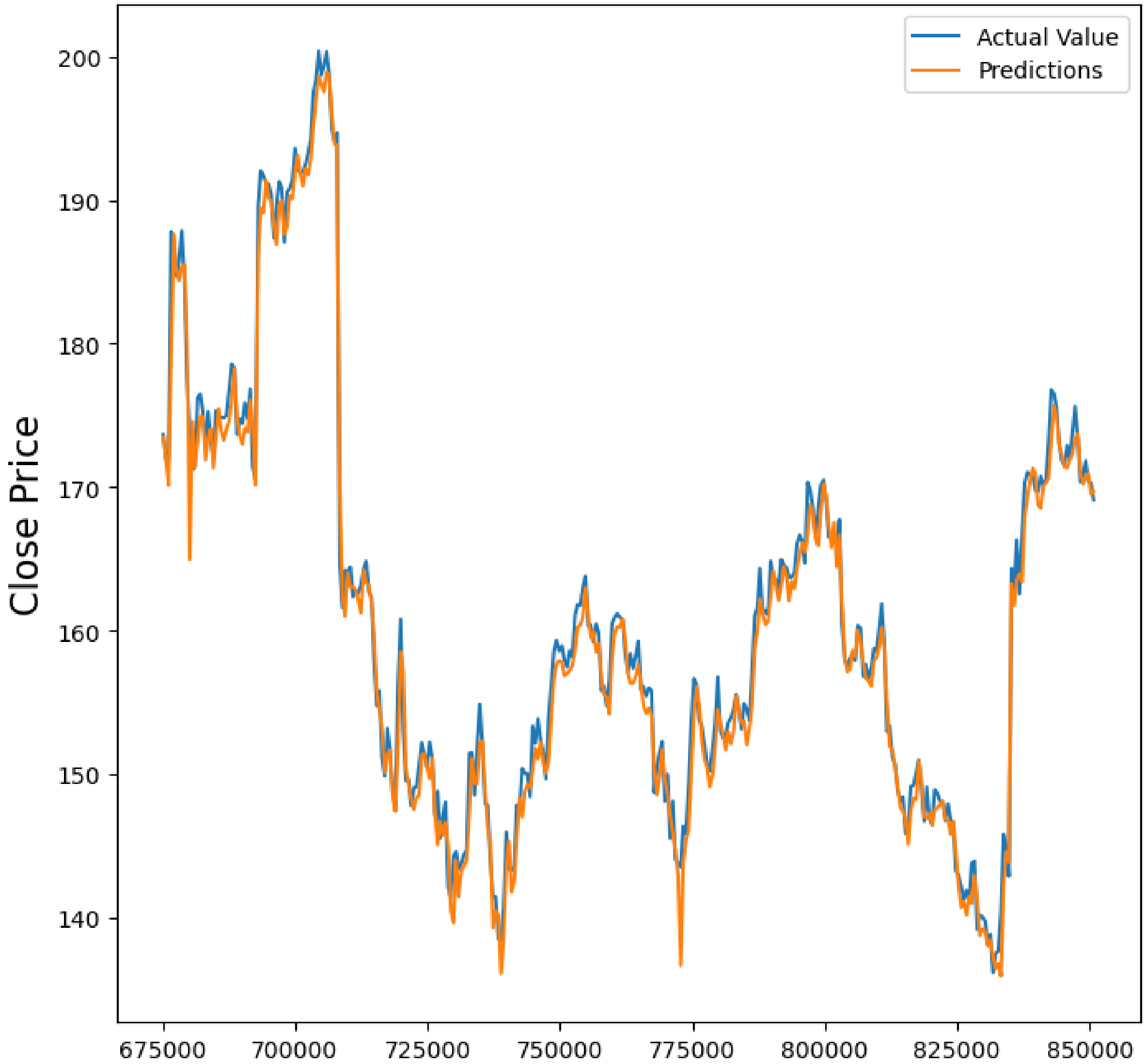
In [31]:

```
from sklearn.metrics import r2_score

print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_3)))
Acc.append(r2_score(y_test, y_pred_3))
```

```
Accuracy score of the predictions: 0.9860817939641268
```

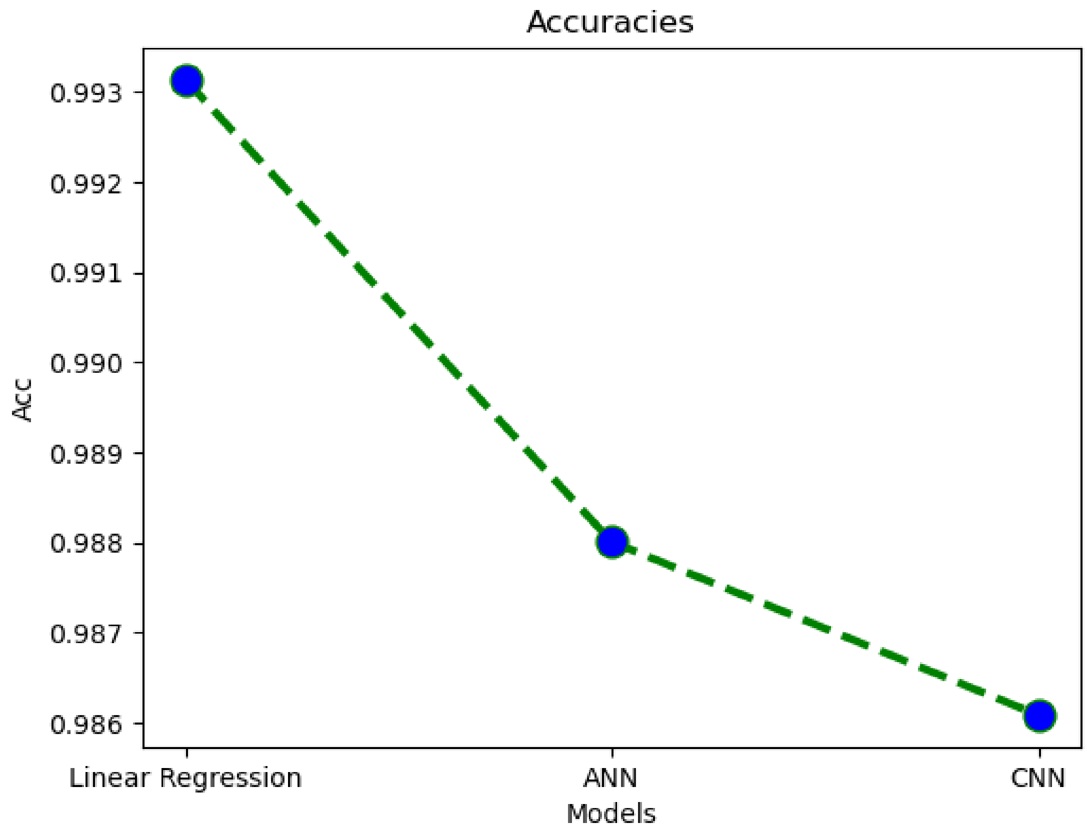
In [32]:

```
plt.figure(figsize=(8,8))
plt.ylabel('Close Price', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



In [33]:

```
plt.plot(range(3), Acc, color='green', linestyle='dashed', linewidth = 3,
         marker='o', markerfacecolor='blue', markersize=12)
plt.ylabel('Acc')
plt.xlabel('Models')
plt.title("Accuracies")
plt.xticks(range(3), ['Linear Regression', 'ANN', 'CNN'])
plt.show()
```



In [34]:

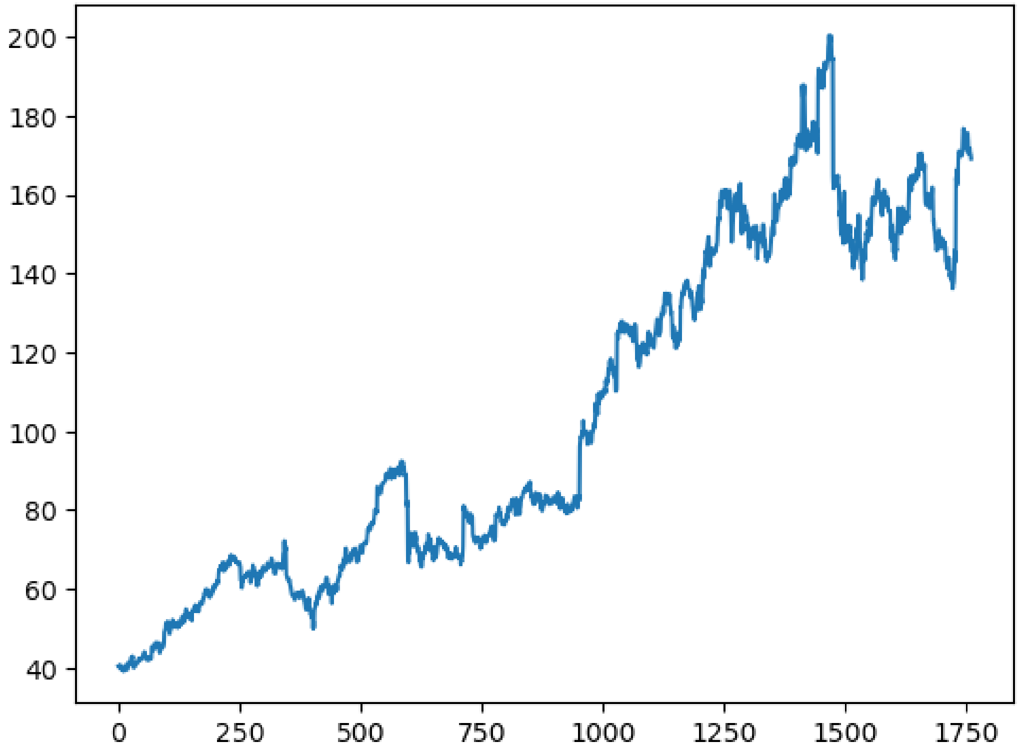
```
close = df.reset_index()['close']
close.head()
```

Out[34]:

```
0    40.380001
1    40.139999
2    40.490002
3    40.480000
4    40.639999
Name: close, dtype: float64
```

In [35]:

```
plt.plot(close)
plt.show()
```



In [36]:

```
time_step = 30
X, y = [], []

for i in range(len(close)-time_step-1):
    X.append(close[i:(i+time_step)])
    y.append(close[(i+time_step)])

X = np.array(X)
y = np.array(y)
```

In [37]:

```
X[:5]
```

Out[37]:

```
array([[40.380001, 40.139999, 40.490002, 40.48    , 40.639999, 40.240002,
        39.540001, 40.09    , 39.560001, 39.310001, 39.5     , 39.16    ,
        39.23    , 39.740002, 40.5     , 40.549999, 40.59    , 39.77    ,
        39.450001, 40.490002, 41.189999, 41.189999, 40.93    , 40.720001,
        40.810001, 41.57    , 42.330002, 42.549999, 42.810001, 42.630001],
       [40.139999, 40.490002, 40.48    , 40.639999, 40.240002, 39.540001,
        40.09    , 39.560001, 39.310001, 39.5     , 39.16    , 39.23    ,
        39.740002, 40.5     , 40.549999, 40.59    , 39.77    , 39.450001,
        40.490002, 41.189999, 41.189999, 40.93    , 40.720001, 40.810001,
        41.57    , 42.330002, 42.549999, 42.810001, 42.630001, 42.880001],
       [40.490002, 40.48    , 40.639999, 40.240002, 39.540001, 40.09    ,
        39.560001, 39.310001, 39.5     , 39.16    , 39.23    , 39.740002,
        40.5     , 40.549999, 40.59    , 39.77    , 39.450001, 40.490002,
        41.189999, 41.189999, 40.93    , 40.720001, 40.810001, 41.57    ,
        42.330002, 42.549999, 42.810001, 42.630001, 42.880001, 40.150002],
       [40.48    , 40.639999, 40.240002, 39.540001, 40.09    , 39.560001,
        39.310001, 39.5     , 39.16    , 39.23    , 39.740002, 40.5     ,
        40.549999, 40.59    , 39.77    , 39.450001, 40.490002, 41.189999,
        41.189999, 40.93    , 40.720001, 40.810001, 41.57    , 42.330002,
        42.549999, 42.810001, 42.630001, 42.880001, 40.150002, 40.       ],
       [40.639999, 40.240002, 39.540001, 40.09    , 39.560001, 39.310001,
        39.5     , 39.16    , 39.23    , 39.740002, 40.5     , 40.549999,
        40.59    , 39.77    , 39.450001, 40.490002, 41.189999, 41.189999,
        40.93    , 40.720001, 40.810001, 41.57    , 42.330002, 42.549999,
        42.810001, 42.630001, 42.880001, 40.150002, 40.       , 40.240002]])
```

In [38]:

```
y[:5]
```

Out[38]:

```
array([42.880001, 40.150002, 40.       , 40.240002, 40.220001])
```

In [39]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X = scaler.fit_transform(X)
pd.DataFrame(X).head()
```

Out[39]:

	0	1	2	3	4	5	6	7	8	9
0	0.007567	0.006079	0.008250	0.008188	0.009180	0.006699	0.002357	0.005769	0.002481	0.000930
1	0.006079	0.008250	0.008188	0.009180	0.006699	0.002357	0.005769	0.002481	0.000930	0.002109
2	0.008250	0.008188	0.009180	0.006699	0.002357	0.005769	0.002481	0.000930	0.002109	0.000000
3	0.008188	0.009180	0.006699	0.002357	0.005769	0.002481	0.000930	0.002109	0.000000	0.000434
4	0.009180	0.006699	0.002357	0.005769	0.002481	0.000930	0.002109	0.000000	0.000434	0.003598

5 rows × 30 columns

In [40]:

```
#now lets split data in test train pairs

from sklearn.model_selection import train_test_split

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

Acc = []
```

In [41]:

```
X_train_ = X_train.reshape(X_train.shape[0],X_train.shape[1],1)
X_test_ = X_test.reshape(X_test.shape[0],X_test.shape[1],1)
```

In [42]:

```
from tensorflow.keras.layers import LSTM

def Reg():
    model = Sequential()

    model.add(LSTM(70, return_sequences=True, input_shape=(30,1)))
    model.add(LSTM(70, return_sequences=True))
    model.add(LSTM(70))
    model.add(Dense(1))

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

    return model
```

In [43]:

```
model_1 = reg()
model_1.fit(X_train_, y_train, epochs=100, validation_split=0.2)
```

```
Epoch 62/100
35/35 [==============================] - 1s 17ms/step - loss: 3.0753 - val_loss:
14.6485
Epoch 63/100
35/35 [==============================] - 1s 18ms/step - loss: 3.3569 - val_loss:
13.6079
Epoch 64/100
35/35 [==============================] - 1s 16ms/step - loss: 2.9476 - val_loss:
24.2553
Epoch 65/100
35/35 [==============================] - 1s 18ms/step - loss: 3.0484 - val_loss:
16.5000
Epoch 66/100
35/35 [==============================] - 1s 20ms/step - loss: 3.8433 - val_loss:
19.1853
Epoch 67/100
35/35 [==============================] - 1s 28ms/step - loss: 2.9231 - val_loss:
10.1443
Epoch 68/100
35/35 [==============================] - 1s 24ms/step - loss: 3.4222 - val_loss:
```

In [44]:

```
y_pred_1 = model_1.predict(X_test_)
```

```
11/11 [==============================] - 0s 5ms/step
```

In [45]:

```
pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_1.flatten()})
pred_df.head()
```

Out[45]:

	Actual	Predicted
0	184.690002	189.099777
1	185.770004	189.631439
2	187.839996	190.265686
3	184.449997	191.880127
4	177.539993	191.474457

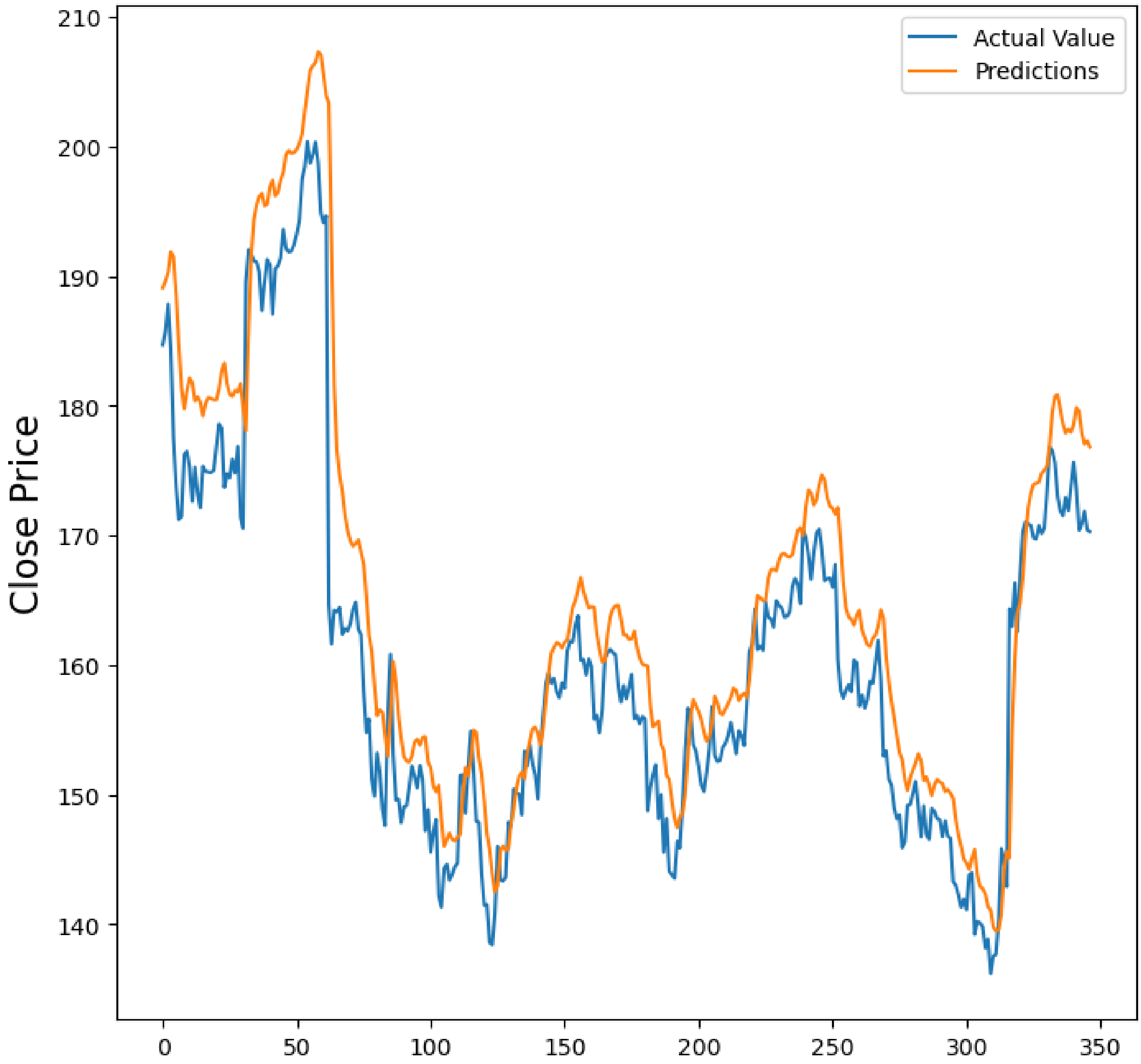
In [46]:

```
from sklearn.metrics import r2_score

print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_1)))
Acc.append(r2_score(y_test, y_pred_1))
```

```
Accuracy score of the predictions: 0.8206824182074275
```

In [47]:

```
plt.figure(figsize=(8,8))
plt.ylabel('Close Price', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



In [48]:

```
model_2 = regressor(inp_dim=30)
model_2.fit(X_train, y_train, epochs=100, validation_split=0.2)
```

```
Epoch 2/100
35/35 [==============================] - 0s 5ms/step - loss: 6252.0913 - val_los
s: 22096.3926
Epoch 3/100
35/35 [==============================] - 0s 4ms/step - loss: 6061.7490 - val_los
s: 20366.2168
Epoch 4/100
35/35 [==============================] - 0s 4ms/step - loss: 5323.5698 - val_los
s: 14838.9824
Epoch 5/100
35/35 [==============================] - 0s 4ms/step - loss: 3572.4983 - val_los
s: 5204.6543
Epoch 6/100
35/35 [==============================] - 0s 4ms/step - loss: 1268.8938 - val_los
s: 46.9312
Epoch 7/100
35/35 [==============================] - 0s 4ms/step - loss: 277.6032 - val_los
s: 1716.5090
Epoch 8/100
```

In [49]:

```
y_pred_2 = model_2.predict(X_test)
```

```
11/11 [==============================] - 0s 6ms/step
```

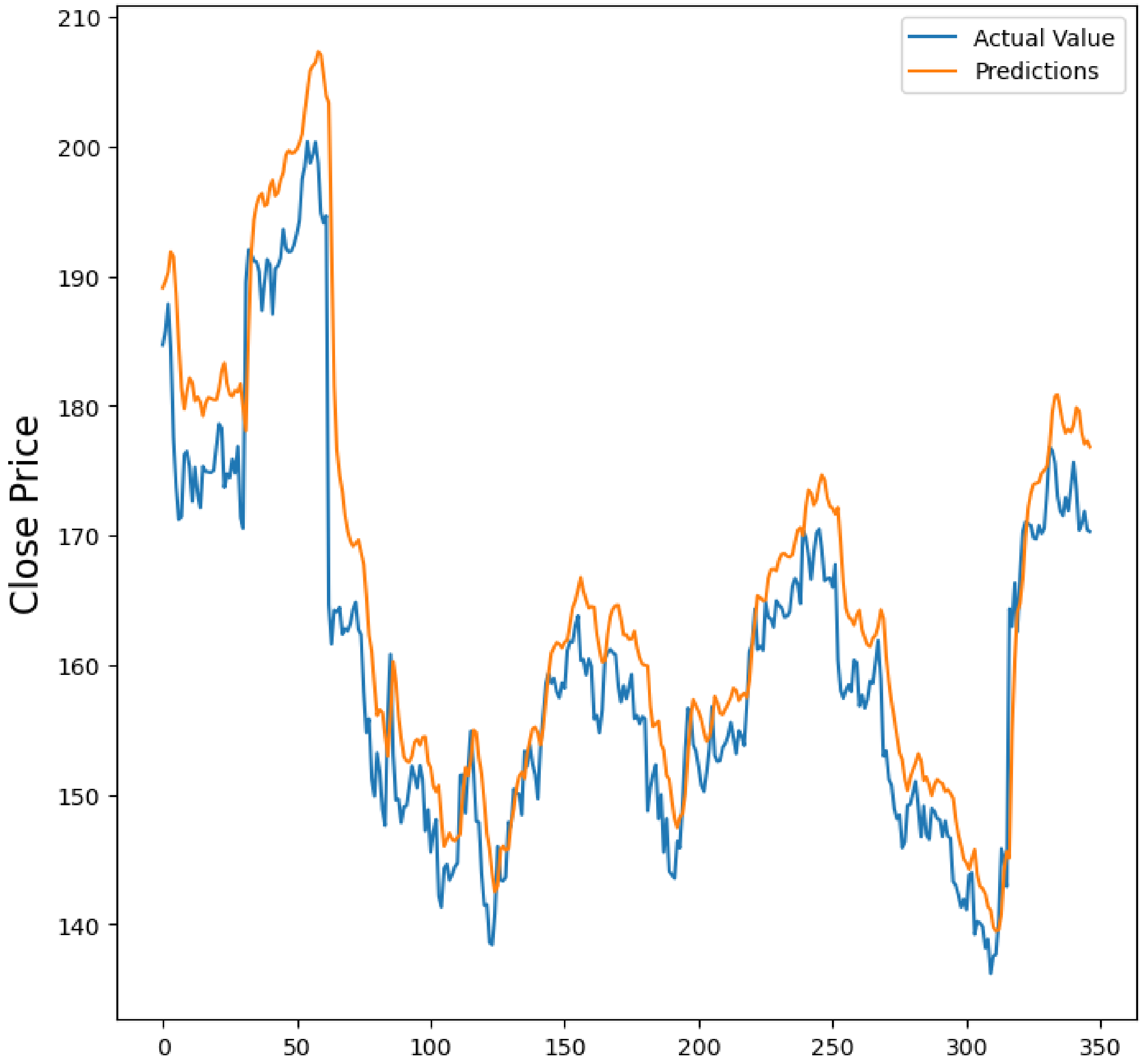
In [50]:

```
from sklearn.metrics import r2_score

print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_2)))
Acc.append(r2_score(y_test, y_pred_2))
```

```
Accuracy score of the predictions: 0.7745443223206158
```

In [51]:

```
plt.figure(figsize=(8,8))
plt.ylabel('Close Price', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



In [52]:

```
model_3 = reg()
model_3.fit(X_train_, y_train, epochs=100, validation_split=0.2)
```

```
Epoch 1/100
35/35 [==============================] - 2s 26ms/step - loss: 3082.6414 - val_lo
ss: 159.3010
Epoch 2/100
35/35 [==============================] - 1s 18ms/step - loss: 97.4133 - val_los
s: 154.4462
Epoch 3/100
35/35 [==============================] - 1s 20ms/step - loss: 16.1196 - val_los
s: 55.8053
Epoch 4/100
35/35 [==============================] - 1s 18ms/step - loss: 15.7579 - val_los
s: 44.1400
Epoch 5/100
35/35 [==============================] - 1s 30ms/step - loss: 16.4674 - val_los
s: 63.8375
Epoch 6/100
35/35 [==============================] - 1s 20ms/step - loss: 14.8252 - val_los
s: 42.9316
Epoch 7/100
```

In [53]:

```
y_pred_3 = model_3.predict(X_test_)
```

```
11/11 [==============================] - 0s 7ms/step
```

In [54]:

```
pred_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_3.flatten()})
pred_df.head()
```

Out[54]:

	Actual	Predicted
0	184.690002	185.972168
1	185.770004	188.472427
2	187.839996	189.712585
3	184.449997	190.770874
4	177.539993	190.902878

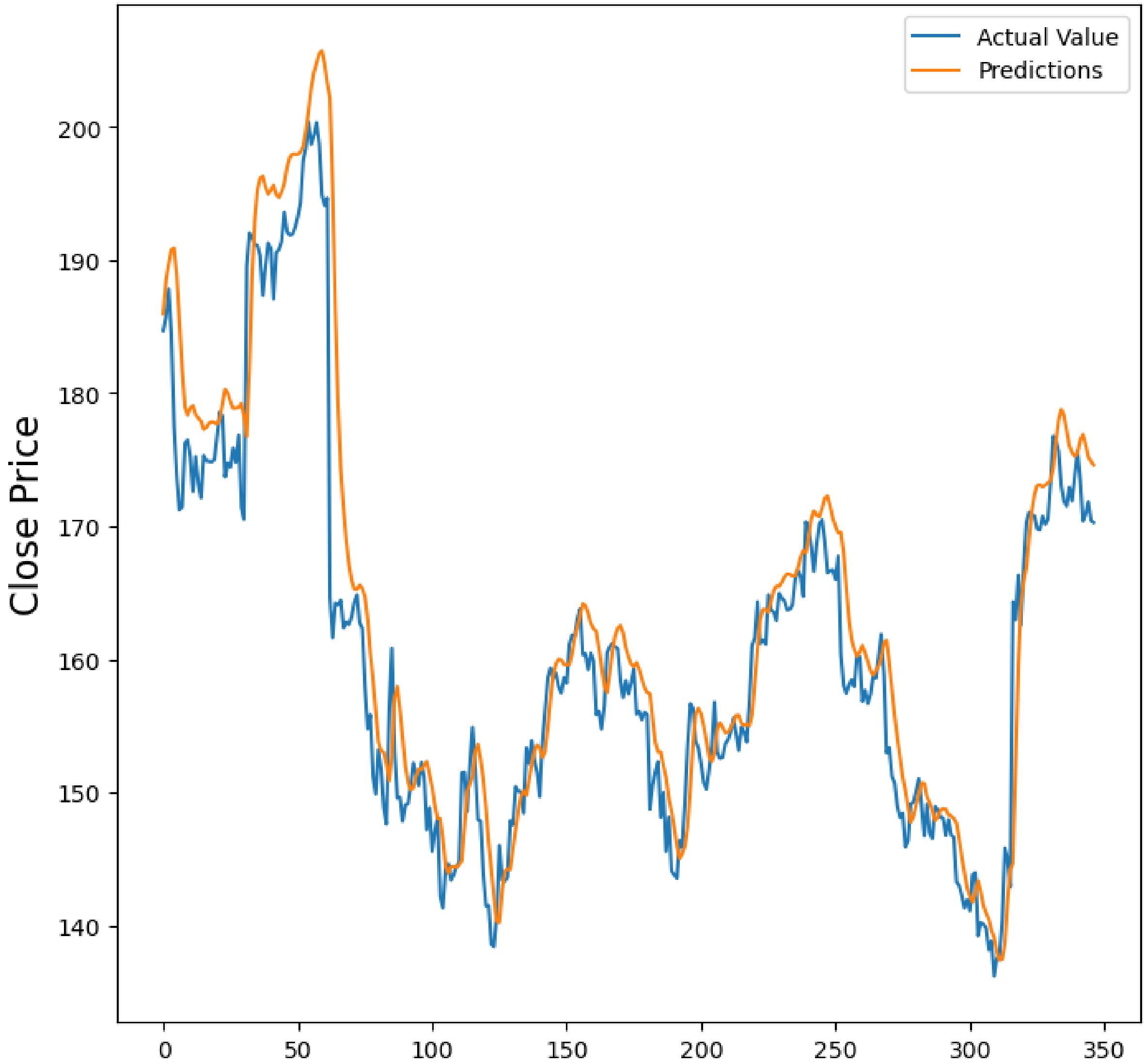
In [55]:

```
from sklearn.metrics import r2_score

print("Accuracy score of the predictions: {0}".format(r2_score(y_test, y_pred_3)))
Acc.append(r2_score(y_test, y_pred_3))
```

```
Accuracy score of the predictions: 0.8681395346500993
```

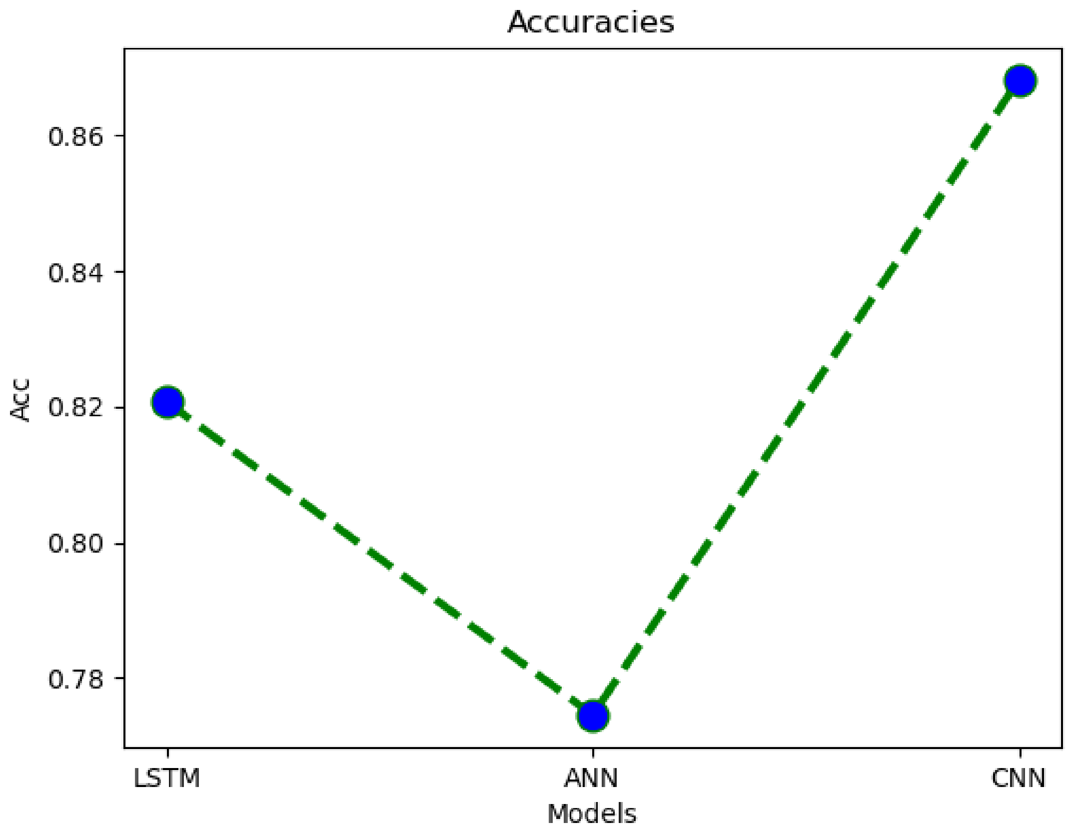
In [56]:

```
plt.figure(figsize=(8,8))
plt.ylabel('Close Price', fontsize=16)
plt.plot(pred_df)
plt.legend(['Actual Value', 'Predictions'])
plt.show()
```



In [57]:

```
plt.plot(range(3), Acc, color='green', linestyle='dashed', linewidth = 3,
         marker='o', markerfacecolor='blue', markersize=12)
plt.ylabel('Acc')
plt.xlabel('Models')
plt.title("Accuracies")
plt.xticks(range(3), ['LSTM', 'ANN', 'CNN'])
plt.show()
```



FROM HERE

In [61]:

```
#COMPUTING CORRELATIONS
corr_matrix = df.corr()
corr_matrix
```

Out[61]:

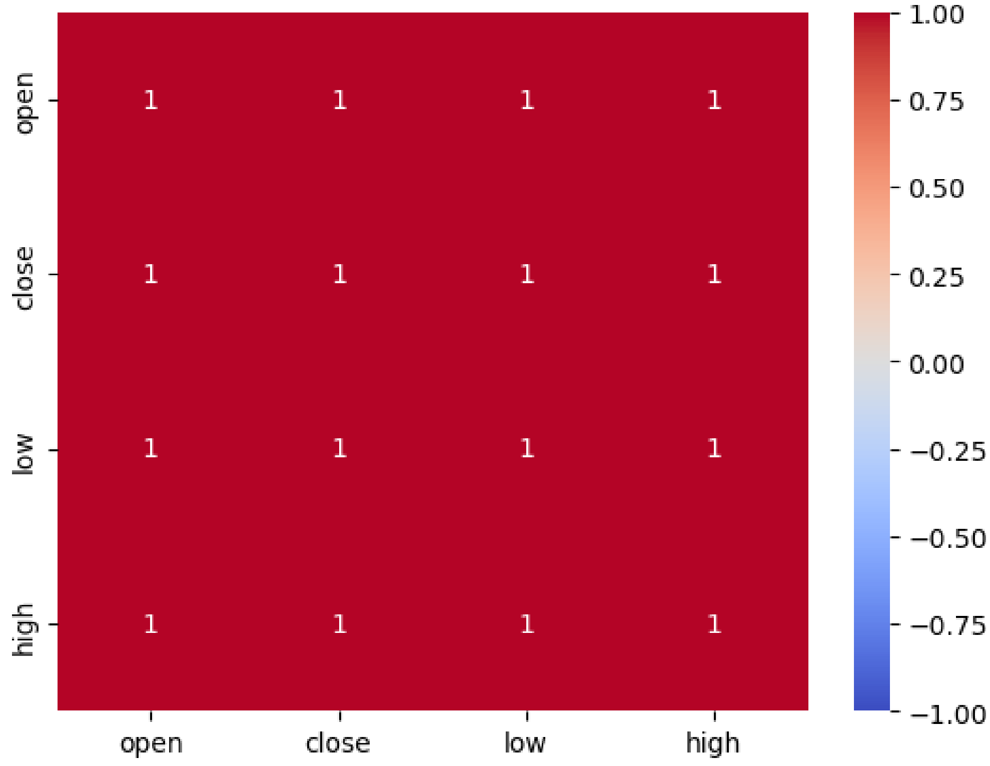
	open	close	low	high
open	1.000000	0.999382	0.999691	0.999628
close	0.999382	1.000000	0.999615	0.999737
low	0.999691	0.999615	1.000000	0.999475
high	0.999628	0.999737	0.999475	1.000000

In [62]:

```
# plot correlation matrix as heatmap(VISUALIZING CORRELATIONS)
sns.heatmap(corr_matrix, cmap='coolwarm', annot=True, vmin=-1, vmax=1)
```

Out[62]:

<AxesSubplot:>



In [65]:

```
print(corr_matrix.loc['open', 'close'])
```

0.9993817292161483

In [69]:

```
#skewness
close_skew = df['close'].skew()
open_skew = df['open'].skew()
low_skew = df['low'].skew()
high_skew = df['high'].skew()
print('Close Skewness:', close_skew)
print('Open Skewness:', open_skew)
print('Low Skewness:', low_skew)
print('High Skewness:', high_skew)
```

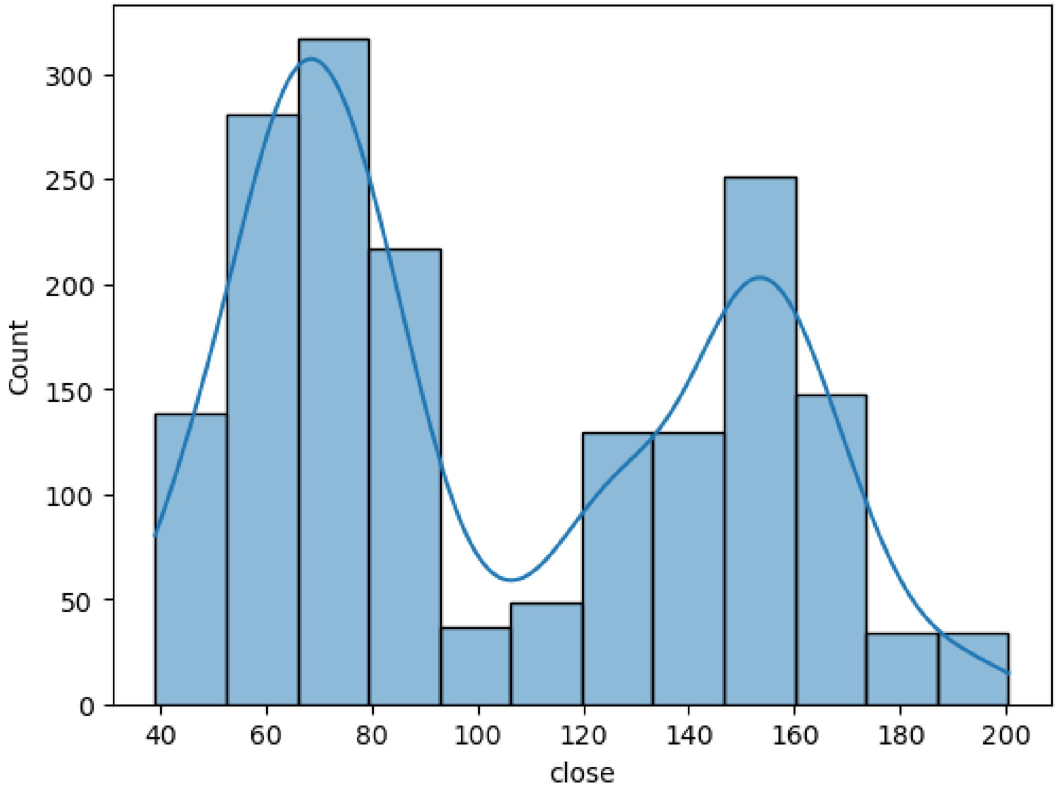
```
Close Skewness: 0.3219049667575204
Open Skewness: 0.3214634068842909
Low Skewness: 0.32028839822816646
High Skewness: 0.32183524318806644
```

In [70]:

```
sns.histplot(data=df, x='close', kde=True)
```

Out[70]:

<AxesSubplot:xlabel='close', ylabel='Count'>

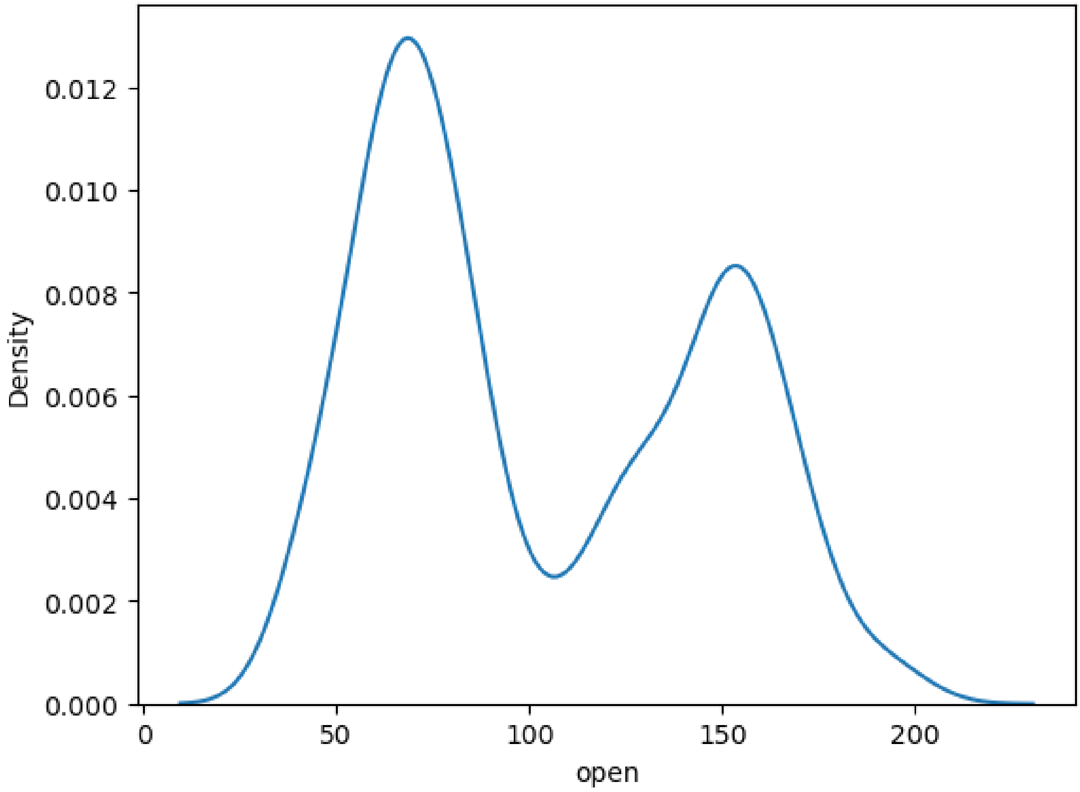


In [71]:

```
sns.kdeplot(data=df, x='open')
```

Out[71]:

<AxesSubplot:xlabel='open', ylabel='Density'>



In [72]:

```
# interpret skewness
if close_skew > 0:
    print('The close variable is positively skewed.')
elif close_skew < 0:
    print('The close variable is negatively skewed.')
else:
    print('The close variable is normally distributed.')
```

The close variable is positively skewed.

In [82]:

```
# split the data into features and target variable (Decision tree)(close-target; open-categorical
from sklearn.preprocessing import LabelEncoder, StandardScaler
X = df.drop('close', axis=1)
y = df['close']
# encode categorical variables
encoder = LabelEncoder()
X['open'] = encoder.fit_transform(X['open'])
# fill missing values
X.fillna(X.mean(), inplace=True)
# scale the features
scaler = StandardScaler()
X = scaler.fit_transform(X)
# split the data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [87]:

```
# create the decision tree classifier object
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
# hyperparameters to tune
param_grid = {'max_depth': [3, 5, 7],
              'min_samples_split': [2, 5, 10],
              'min_samples_leaf': [1, 2, 4]}
# grid search cross-validation
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(dtc, param_grid, cv=5)
```

In [95]:

```
from sklearn.tree import DecisionTreeRegressor

# Create a decision tree regressor object
tree_regressor = DecisionTreeRegressor()

# Fit the decision tree regressor to the training data
tree_regressor.fit(X_train, y_train)
```

Out[95]:

```
DecisionTreeRegressor()
```

In [92]:

```
# Make predictions on test data
y_pred = tree_regressor.predict(X_test)
```

In [97]:

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV

# Create a decision tree regressor object
tree_regressor = DecisionTreeRegressor()

# Define the parameter grid for grid search
param_grid = {
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 3]
}

# Create the grid search object
grid_search = GridSearchCV(tree_regressor, param_grid, cv=5)

# Fit the grid search object to the training data
grid_search.fit(X_train, y_train)

# Retrieve the best parameters
best_params = grid_search.best_params_

# Create the final decision tree model with the best hyperparameters
final_model = DecisionTreeRegressor(
    max_depth=best_params['max_depth'],
    min_samples_split=best_params['min_samples_split'],
    min_samples_leaf=best_params['min_samples_leaf']
)

# Fit the final model to the training data
final_model.fit(X_train, y_train)
```

Out[97]:

```
DecisionTreeRegressor(max_depth=7, min_samples_leaf=2, min_samples_split=4)
```

In [105]:

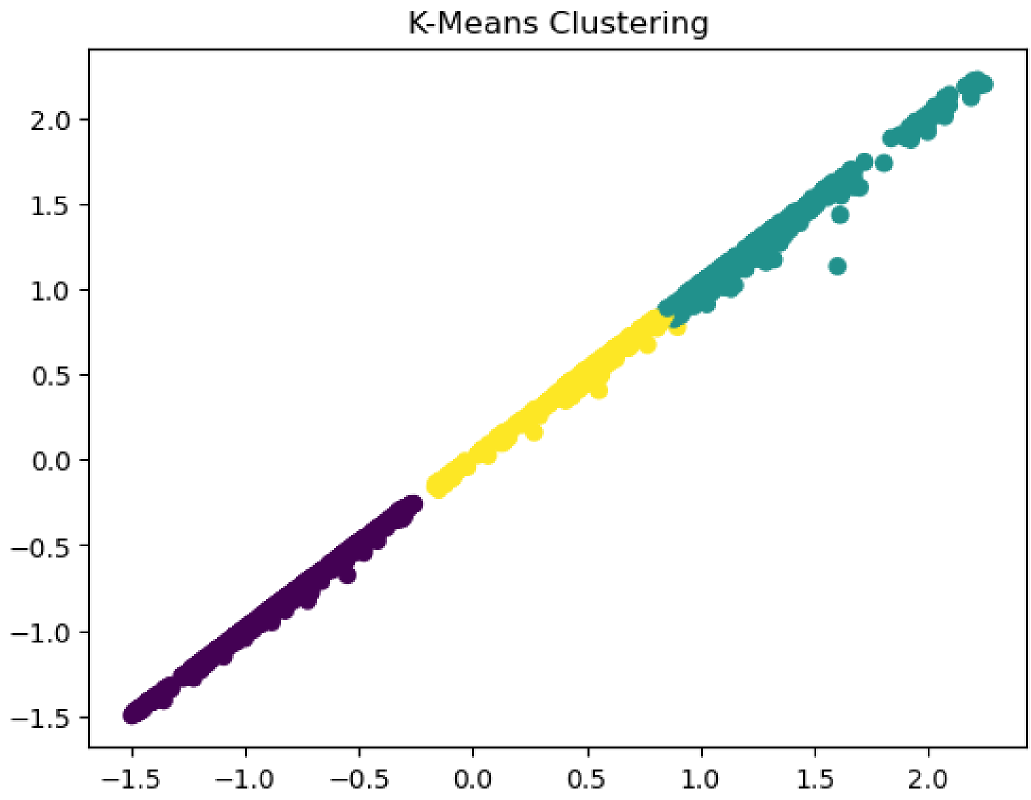
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Make predictions on test data
y_pred = final_model.predict(X_test)

# Compute regression evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
print("Mean absolute error is: ",mae)
print("Mean squared error is: ",mse)
print("R-squared is: ",r2)
```

```
Mean absolute error is:  0.7171359521452576
Mean squared error is:  1.1151255161598734
R-squared is:  0.9993997218908506
```

In [144]:

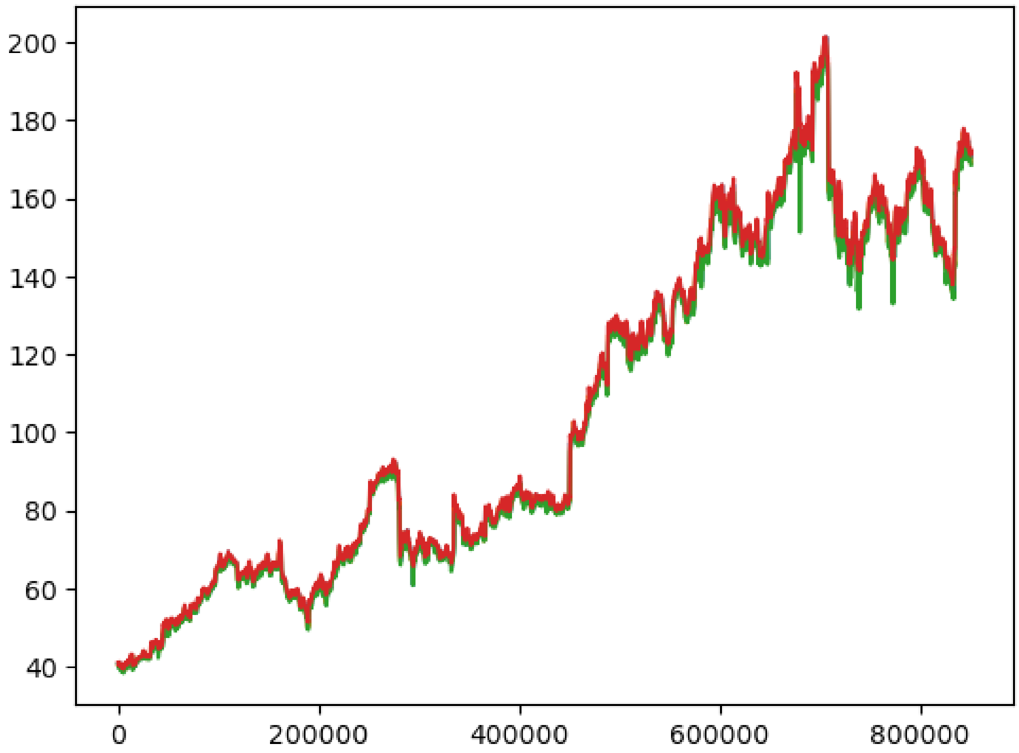
```
#K-means
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
X = df.drop('close', axis=1)
# preprocess the data by scaling the features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
# choose the number of clusters
k = 3
# create the KMeans clustering object
kmeans = KMeans(n_clusters=k, random_state=42)
# fit the algorithm to the data and obtain the cluster labels
labels = kmeans.fit_predict(X_scaled)
# evaluate the performance of the clustering using silhouette score
score = silhouette_score(X_scaled, labels)
print('Silhouette score:', score)
# visualize the results
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels)
plt.title('K-Means Clustering')
plt.show()
```

```
Silhouette score: 0.659542045645491
```



In [153]:

```
#Timeseries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
plt.plot(df)
plt.show()
```



In [3]:

```
pip install --upgrade pip
```

```
Requirement already satisfied: pip in c:\users\hp\anaconda3\ancon\lib\site-package
s (22.2.2)
Collecting pip
  Downloading pip-23.1.2-py3-none-any.whl (2.1 MB)
     ---------------------------------------- 2.1/2.1 MB 1.1 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 22.2.2
    Uninstalling pip-22.2.2:
      Successfully uninstalled pip-22.2.2
Successfully installed pip-23.1.2
Note: you may need to restart the kernel to use updated packages.
```

In [4]:

```
pip install --no-cache-dir mlxtend
```

```
Collecting mlxtendNote: you may need to restart the kernel to use updated package
s.

  Downloading mlxtend-0.22.0-py2.py3-none-any.whl (1.4 MB)
     ---------------------------------------- 1.4/1.4 MB 4.5 MB/s eta 0:00:00
Requirement already satisfied: scipy>=1.2.1 in c:\users\hp\anaconda3\ancon\lib\sit
e-packages (from mlxtend) (1.9.1)
Requirement already satisfied: numpy>=1.16.2 in c:\users\hp\anaconda3\ancon\lib\si
te-packages (from mlxtend) (1.23.5)
Requirement already satisfied: pandas>=0.24.2 in c:\users\hp\anaconda3\ancon\lib\s
ite-packages (from mlxtend) (1.4.4)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\hp\anaconda3\ancon
\lib\site-packages (from mlxtend) (1.0.2)
Requirement already satisfied: matplotlib>=3.0.0 in c:\users\hp\anaconda3\ancon\li
b\site-packages (from mlxtend) (3.5.2)
Requirement already satisfied: joblib>=0.13.2 in c:\users\hp\anaconda3\ancon\lib\s
ite-packages (from mlxtend) (1.1.0)
Requirement already satisfied: setuptools in c:\users\hp\anaconda3\ancon\lib\site-
packages (from mlxtend) (63.4.1)
Requirement already satisfied: cycler>=0.10 in c:\users\hp\anaconda3\ancon\lib\sit
e-packages (from matplotlib>=3.0.0->mlxtend) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\hp\anaconda3\ancon\li
b\site-packages (from matplotlib>=3.0.0->mlxtend) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\hp\anaconda3\ancon\li
b\site-packages (from matplotlib>=3.0.0->mlxtend) (1.4.2)
Requirement already satisfied: packaging>=20.0 in c:\users\hp\anaconda3\ancon\lib
\site-packages (from matplotlib>=3.0.0->mlxtend) (21.3)
Requirement already satisfied: pillow>=6.2.0 in c:\users\hp\anaconda3\ancon\lib\si
te-packages (from matplotlib>=3.0.0->mlxtend) (9.2.0)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\hp\anaconda3\ancon\lib
\site-packages (from matplotlib>=3.0.0->mlxtend) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\hp\anaconda3\ancon
\lib\site-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\hp\anaconda3\ancon\lib\sit
e-packages (from pandas>=0.24.2->mlxtend) (2022.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\hp\anaconda3\ancon
\lib\site-packages (from scikit-learn>=1.0.2->mlxtend) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\hp\anaconda3\ancon\lib\site-pa
ckages (from python-dateutil>=2.7->matplotlib>=3.0.0->mlxtend) (1.16.0)
Installing collected packages: mlxtend
Successfully installed mlxtend-0.22.0
```

In [5]:

```
#Association rule mining
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules
```

In [16]:

```
dataset = [['open', 'close', 'low'],
           ['open', 'volume', 'high'],
           ['open', 'close', 'low', 'high', 'volume'],
           ['open', 'volume', 'low', 'close', 'high'],
           ['close', 'low', 'high', 'volume']]
te = TransactionEncoder()
te_ary = te.fit_transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
print(df)
# Step 2: Support Calculation
min_support = 0.4
freq_items = apriori(df, min_support=min_support, use_colnames=True)
print(freq_items)
# Step 3: Frequent Itemset Generation
min_threshold = 0.7
assoc_rules = association_rules(freq_items, metric="confidence", min_threshold=min_threshold)
print(assoc_rules)
# Step 4: Rule Generation
min_confidence = 0.8
rules = association_rules(freq_items, metric="confidence", min_threshold=min_confidence)
print(rules)
# Step 5: Rule Evaluation
rules["lift"] = association_rules(freq_items, metric="lift", min_threshold=min_confidence)["lift'
rules["conviction"] = association_rules(freq_items, metric="conviction", min_threshold=min_confi
print(rules)
```

```
   close   high    low   open  volume
0   True  False   True   True   False
1  False   True  False   True    True
2   True   True   True   True    True
3   True   True   True   True    True
4   True   True   True  False    True
    support            itemsets
0       0.8             (close)
1       0.8              (high)
2       0.8               (low)
3       0.8              (open)
4       0.8            (volume)
5       0.6       (high, close)
6       0.8        (close, low)
7       0.6       (close, open)
8       0.6     (close, volume)
9       0.6         (high, low)
10      0.6        (high, open)
11      0.8      (high, volume)
```

In [25]:

```
#Locality-sensitive hashing
X = np.random.rand(100, 10)
from sklearn.random_projection import SparseRandomProjection
rp = SparseRandomProjection(n_components=5)
# Transform the data using the random projection
X_rp = rp.fit_transform(X)
X_rp
```

Out[25]:

```
array([[-0.00893743,  0.02802295,  0.05954404, -0.25215123, -0.97969252],
       [-0.10157833,  0.15425237, -0.23710551, -0.58155089, -0.50146989],
       [-0.26888246,  0.7466086 , -0.25253755, -0.54278773, -0.95001067],
       [-0.76547213,  0.92453109, -0.30168417, -0.63579927, -0.71176313],
       [-0.2132047 ,  0.72373173, -0.01468971, -1.15824927, -0.87445205],
       [-0.63688853,  0.69153872, -0.57086425, -0.32678388, -0.81327409],
       [-0.38089537,  0.95459264, -0.69370915,  0.05562309, -1.59281984],
       [-0.2516888 ,  0.46232538,  0.05245795, -0.85781803, -0.15839716],
       [-0.01170683,  0.6990972 ,  0.46028483, -0.415864  , -0.94354021],
       [-0.09304659,  0.23264894, -0.6165143 , -0.03651614, -0.14970285],
       [-0.44664074,  0.49817116, -0.23350027,  0.20335505, -1.16599803],
       [-0.11009092,  0.57995656, -0.11633803, -0.92340283, -0.92206884],
       [-0.7351686 ,  1.02740042,  0.20106924, -0.54125348,  0.24316932],
       [-0.36591537,  1.10601017,  0.63846698, -0.78556162, -0.5587942 ],
       [-0.5852557 ,  1.30028745, -0.15152852, -0.69957649, -0.59883068],
       [-0.43208351,  0.65968718, -0.23788166, -0.40299102, -0.41058953],
       [-0.71267887,  1.25719934, -0.07280254, -0.52598227, -0.61312189],
       [-0.73310829,  0.77368495, -0.2584726 , -0.22036931, -0.54510241],
       [-0.11848165,  0.62852424, -0.43795221, -0.33605381, -0.49261325],
       [-0.10228279,  0.1601532 ,  0.13291006, -0.28851429, -1.04501296],
       [-0.42207975,  0.93113427,  0.14509041, -0.92744452, -0.26654873],
       [-0.10340316,  0.80994474,  0.39487168, -0.58801493, -0.20023699],
       [-0.67632262,  1.01382612, -0.18693943,  0.44195837, -0.94379002],
       [-0.40534182,  0.51154394, -0.01923345, -0.88123629,  0.05345514],
       [-0.29254982,  0.59796368,  0.05782815, -0.28971646, -0.4364229 ],
       [-0.17763021,  0.65424729,  0.03832217,  0.19705265, -1.01084553],
       [-0.43671935,  1.11781488, -0.38320147, -0.60525362, -1.31338568],
       [-0.30857834,  1.02278991, -0.43480447,  0.10991007, -1.23021052],
       [-0.04822639,  0.58555942, -0.14838622, -0.73491405, -1.35384302],
       [-0.39158882,  1.13343331, -0.04094888, -0.32332416, -0.93214079],
       [-0.53344268,  1.11156919,  0.23768846, -0.42751308, -0.17945805],
       [-0.29345757,  0.75368309,  0.17983582, -0.52402287, -0.91021569],
       [-0.38084899,  0.99957132,  0.16389361, -0.49232353, -0.76267865],
       [-0.5880033 ,  0.76738081,  0.16791112, -0.30030429, -0.83789066],
       [-0.14456713,  0.60380907,  0.58990912, -0.1285485 , -0.78651661],
       [-0.40116456,  0.43821497,  0.44518472, -0.63001889, -0.64825967],
       [-0.36182359,  0.50036189, -0.4800447 , -0.20436374, -1.3146845 ],
       [-0.46110568,  0.5675078 ,  0.15739276, -0.64338177, -1.21045739],
       [-0.38165364,  0.81747117,  0.30151752, -0.9059856 ,  0.00585634],
       [-0.37690131,  1.02380944,  0.11162654, -1.10498992, -0.42320301],
       [-0.08928064,  0.3247933 ,  0.21588843, -0.64410766, -0.93464591],
       [-0.11369346,  0.41393274,  0.14608968, -0.23953399, -1.52100354],
       [-0.42949419,  1.07668656, -0.47394007, -0.05715454, -0.27738104],
       [-0.70674952,  1.07994897,  0.0398614 , -0.67198275, -0.70502129],
```