```
import yfinance as yf
# BTC-USD
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt



from datetime import datetime
end = datetime.now()
start = datetime(end.year-10, end.month, end.day)
stock = "BTC-USD" #"BTC-USD"
bit_coin_data = yf.download(stock, start, end)
bit_coin_data.head()
```

```
[**********100%**********]  1 of 1 completed
```

Price	Close	High	Low	Open	Volume
Ticker	BTC-USD	BTC-USD	BTC-USD	BTC-USD	BTC-USD
Date					
2015-03-16	290.592987	294.112000	285.684998	285.684998	21516100
2015-03-17	285.505005	292.364990	284.373993	290.595001	21497200
2015-03-18	256.299011	285.335999	249.869995	285.066986	57008000
2015-03-19	260.928009	264.243988	248.636002	255.880005	52732000
2015-03-20	261.748993	264.847992	259.161987	260.955994	18456700

Next steps: Generate code with bit_coin_data | View recommended plots | New interactive sheet

```
bit_coin_data.describe()
```

Price	Close	High	Low	Open	Volume
Ticker	BTC-USD	BTC-USD	BTC-USD	BTC-USD	BTC-USD
count	3654.000000	3654.000000	3654.000000	3654.000000	3.654000e+03
mean	22085.216336	22554.606491	21544.947113	22063.794495	2.010180e+10
std	24437.316022	24940.749305	23864.274863	24420.657468	2.087890e+10
min	210.494995	223.832993	199.567001	210.067993	1.060090e+07
25%	3617.863342	3676.000000	3560.231323	3608.554382	1.943640e+09
50%	10271.604492	10451.468262	10013.020020	10262.078613	1.653543e+10
75%	35866.427734	37232.797852	34711.041016	35849.711914	3.107478e+10
max	106146.265625	109114.882812	105291.734375	106147.296875	3.509679e+11

```
bit_coin_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 3654 entries, 2015-03-16 to 2025-03-16
```

```
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   (Close, BTC-USD)   3654 non-null   float64
 1   (High, BTC-USD)    3654 non-null   float64
 2   (Low, BTC-USD)     3654 non-null   float64
 3   (Open, BTC-USD)    3654 non-null   float64
 4   (Volume, BTC-USD)  3654 non-null   int64
dtypes: float64(4), int64(1)
memory usage: 171.3 KB
```

```
bit_coin_data.isna().sum()
```

Price	Ticker	0
Close	BTC-USD	0
High	BTC-USD	0
Low	BTC-USD	0
Open	BTC-USD	0
Volume	BTC-USD	0

```
Closing_price = bit_coin_data[['Close']]
Closing_price
```

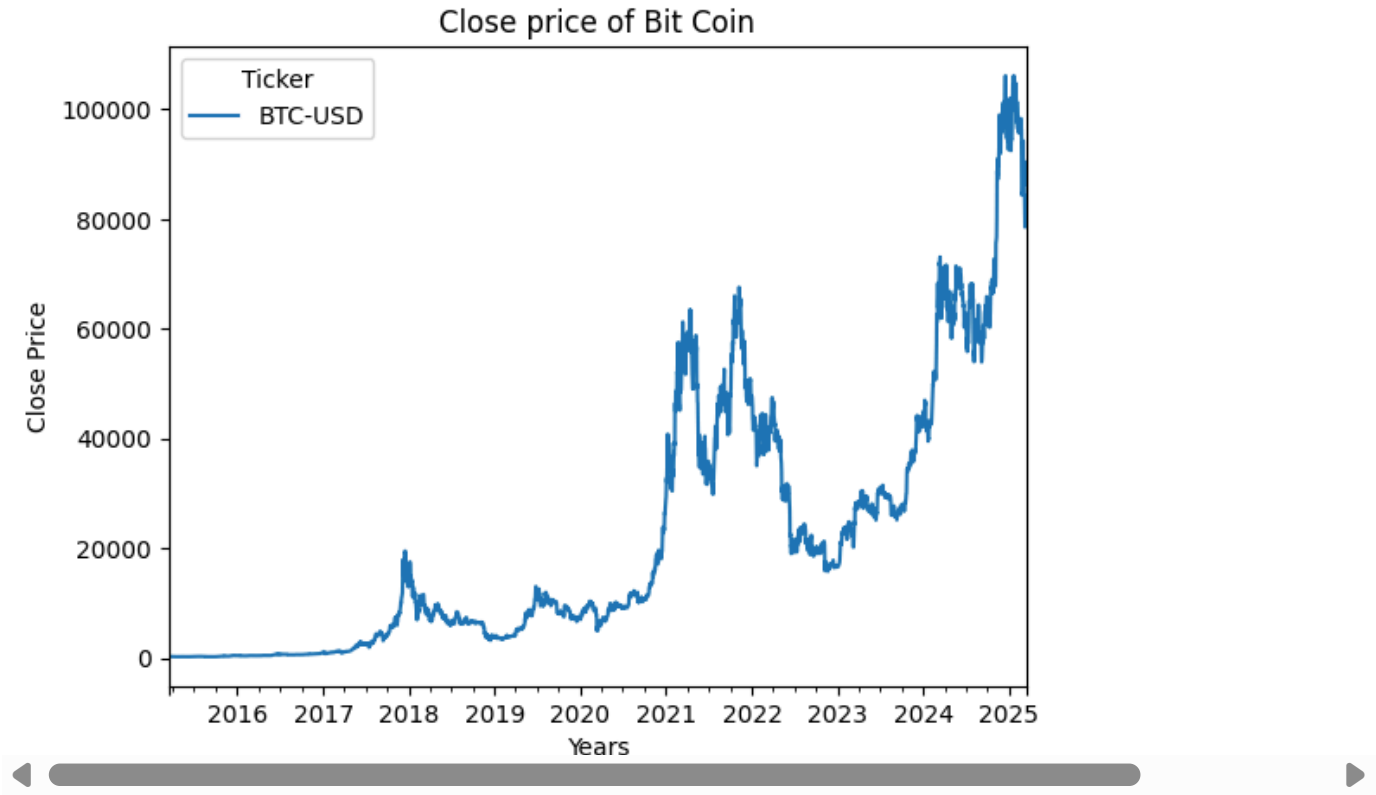
Price	Close
Ticker	BTC-USD
Date	
2015-03-16	290.592987
2015-03-17	285.505005
2015-03-18	256.299011
2015-03-19	260.928009
2015-03-20	261.748993
...	...
2025-03-12	83722.359375
2025-03-13	81066.703125
2025-03-14	83969.101562
2025-03-15	84343.109375
2025-03-16	83110.046875

3654 rows × 1 columns

Next steps: Generate code with Closing_price | View recommended plots | New interactive sheet

```
plt.figure(figsize=(15,5))
Closing_price['Close'].plot()
plt.xlabel('Years')
plt.ylabel('Close Price')
plt.title('Close price of Bit Coin')
```

```
Text(0.5, 1.0, 'Close price of Bit Coin')
<Figure size 1500x500 with 0 Axes>
```



1,2,3,4,5,6,7,8,9,10

MA 5 days for each value(CP) = null, null, null, null, 3, 4, 5...

```
[None, None, None, None, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0]
```

```
(2+3+4+5+6)//5
```

```
4
```

```
for i in range(2015,2025):
    print(list(Closing_price.index.year).count(i))
```

```
291
366
365
365
365
366
365
365
365
366
```

```
Closing_price['MA_for_365_days'] = Closing_price['Close'].rolling(365).mean()
Closing_price['MA_for_365_days'].head()
```

```
<ipython-input-18-388365a217d8>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.
  Closing_price['MA_for_365_days'] = Closing_price['Close'].rolling(365).mean()
```

Date	MA_for_365_days
2015-03-16	NaN
2015-03-17	NaN
2015-03-18	NaN
2015-03-19	NaN
2015-03-20	NaN

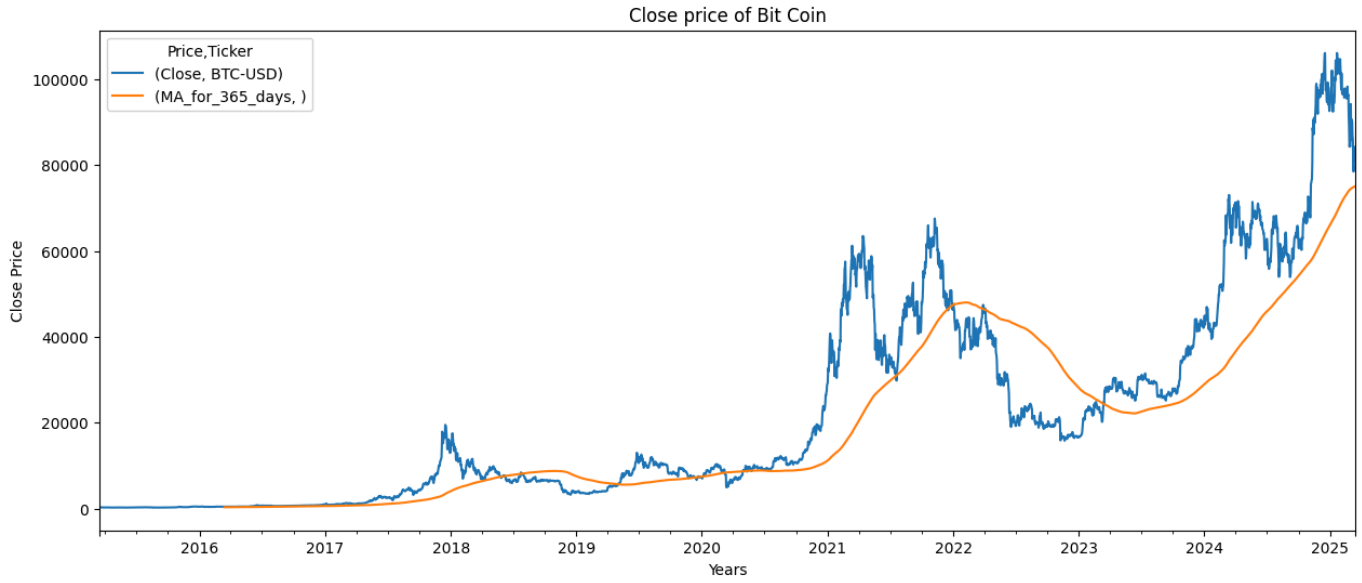
dtype: float64

```
Closing_price['MA_for_365_days'][0:365].tail()
```

Date	MA_for_365_days
2016-03-10	NaN
2016-03-11	NaN
2016-03-12	NaN
2016-03-13	NaN
2016-03-14	304.792123

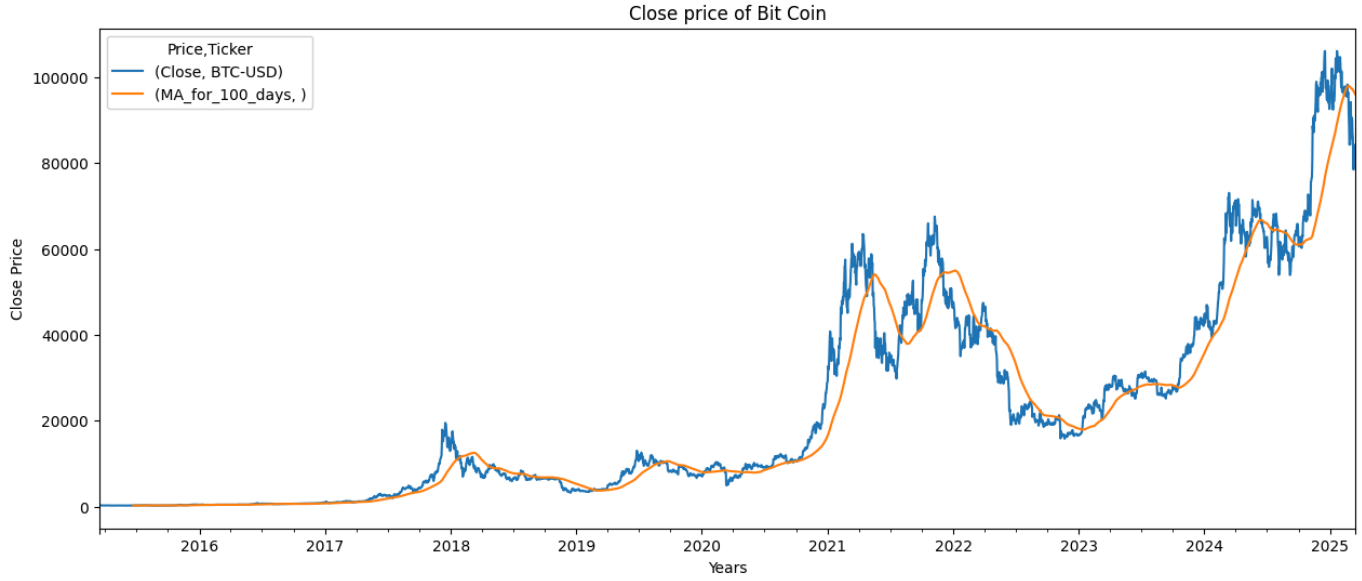
```
plt.figure()
Closing_price[['Close','MA_for_365_days']].plot(figsize=(15,6))
plt.xlabel('Years')
plt.ylabel('Close Price')
plt.title('Close price of Bit Coin')
```

```
Text(0.5, 1.0, 'Close price of Bit Coin')
<Figure size 640x480 with 0 Axes>
```




```
Closing_price['MA_for_100_days'] = Closing_price['Close'].rolling(100).mean()
Closing_price['MA_for_100_days'].head()
plt.figure()
Closing_price[['Close','MA_for_100_days']].plot(figsize=(15,6))
plt.xlabel('Years')
plt.ylabel('Close Price')
plt.title('Close price of Bit Coin')
```

```
<ipython-input-21-75ccff927d8d>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.
  Closing_price['MA_for_100_days'] = Closing_price['Close'].rolling(100).mean()
Text(0.5, 1.0, 'Close price of Bit Coin')
<Figure size 640x480 with 0 Axes>
```



```
# 1 to 100 ==> 101 day
# 2 to 101 (100 days) ==> 102 day
# 3 to 102 (100 days) ==> 103rd day future days close price using our model


from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler(feature_range=(0,1))
scaled_data = scaler.fit_transform(Closing_price[['Close']].values)
scaled_data
```

```
array([[7.56099582e-04],
       [7.08070648e-04],
       [4.32375352e-04],
       ...,
       [7.90654621e-01],
       [7.94185136e-01],
       [7.82545418e-01]])
```

```
scaled_data.shape
```

```
(3654, 1)
```

```
x_data = []
y_data = []

base_days = 100

for i in range(base_days,len(scaled_data)):
    x_data.append(scaled_data[i-base_days:i])
    y_data.append(scaled_data[i])

x_data, y_data = np.array(x_data), np.array(y_data)


x_data[0], y_data[0]
```

```
            [2.34566652e-04],
            [2.81802944e-04],
            [2.67869849e-04],
            [2.87513912e-04],
            [2.51237192e-04],
            [2.51293798e-04],
            [2.52870296e-04],
            [2.54050256e-04],
            [2.51104965e-04],
            [2.15696794e-04],
            [1.85914609e-04],
            [1.17344686e-04],
            [1.44502641e-04],
            [1.45172851e-04],
            [1.30541459e-04],
            [1.36469491e-04],
            [1.42765830e-04],
            [1.16919918e-04],
            [1.69857671e-04],
            [1.75134508e-04],
            [1.72821688e-04],
            [1.81336357e-04],
            [1.83951081e-04],
            [2.06795094e-04],
            [2.17565833e-04],
            [2.48527979e-04],
            [3.81363244e-04],
            [3.66155844e-04],
            [3.63541120e-04],
            [3.21997069e-04],
            [3.27717543e-04],
            [3.15747976e-04],
            [3.44501297e-04],
            [3.19070791e-04]]),
      array([0.00028338]))
```

```
len_train = int(len(x_data)*0.9)
x_train = x_data[:len_train]
y_train = y_data[:len_train]

x_test = x_data[len_train:]
y_test = y_data[len_train:]


print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(3198, 100, 1)
(3198, 1)
(356, 100, 1)
(356, 1)
```

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

model = Sequential()

model.add(LSTM(128, return_sequences = True, input_shape = (x_train.shape[1],1)))

model.add(LSTM(64, return_sequences = False))
```

```
model.add(Dense(25))

model.add(Dense(1))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `i
  super().__init__(**kwargs)
```

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 100, 128)	66,560
lstm_1 (LSTM)	(None, 64)	49,408
dense (Dense)	(None, 25)	1,625
dense_1 (Dense)	(None, 1)	26

Total params: 117,619 (459.45 KB)

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

model.fit(x_train, y_train, batch_size = 5, epochs=10)
```

```
Epoch 1/10
640/640 ━━━━━━━━━━━━━━━━━━━━ 110s 162ms/step - loss: 0.0013
Epoch 2/10
640/640 ━━━━━━━━━━━━━━━━━━━━ 131s 145ms/step - loss: 2.6343e-04
Epoch 3/10
640/640 ━━━━━━━━━━━━━━━━━━━━ 137s 138ms/step - loss: 2.1779e-04
Epoch 4/10
640/640 ━━━━━━━━━━━━━━━━━━━━ 141s 136ms/step - loss: 1.6983e-04
Epoch 5/10
640/640 ━━━━━━━━━━━━━━━━━━━━ 88s 137ms/step - loss: 1.3800e-04
Epoch 6/10
640/640 ━━━━━━━━━━━━━━━━━━━━ 142s 137ms/step - loss: 1.4221e-04
Epoch 7/10
640/640 ━━━━━━━━━━━━━━━━━━━━ 92s 144ms/step - loss: 1.2167e-04
Epoch 8/10
640/640 ━━━━━━━━━━━━━━━━━━━━ 138s 137ms/step - loss: 1.1220e-04
Epoch 9/10
640/640 ━━━━━━━━━━━━━━━━━━━━ 88s 138ms/step - loss: 1.3175e-04
Epoch 10/10
640/640 ━━━━━━━━━━━━━━━━━━━━ 142s 138ms/step - loss: 1.2336e-04
<keras.src.callbacks.history.History at 0x7fd2cee6ae10>
```

```
predictions = model.predict(x_test)
predictions
```

```
       [0.99052715],
       [0.98834455],
       [0.9672879 ],
       [0.9633743 ],
       [0.95644236],
       [0.9817919 ],
       [0.9900609 ],
       [0.9646407 ],
       [0.94912636],
       [0.92079335],
       [0.9621051 ],
       [0.9225947 ],
       [0.91288644],
       [0.91433746],
       [0.9140391 ],
       [0.9137866 ],
       [0.91417587],
       [0.92390263],
       [0.9058812 ],
       [0.9292032 ],
       [0.914622  ],
       [0.9245756 ],
       [0.9246407 ],
       [0.91006845],
       [0.90716475],
       [0.9052796 ],
       [0.9167415 ],
       [0.9332405 ],
       [0.9089659 ],
       [0.9153577 ],
       [0.91211575],
       [0.86289084],
       [0.83965826],
       [0.79809505],
       [0.80468094],
       [0.8020696 ],
       [0.81922877],
       [0.9019349 ],
       [0.8139139 ],
       [0.829913  ],
       [0.8646862 ],
       [0.85576916],
       [0.8231749 ],
       [0.8191924 ],
       [0.76447135],
       [0.7466268 ],
       [0.79108435],
       [0.7986892 ],
       [0.7716017 ],
       [0.80162615],
       [0.8046728 ]], dtype=float32)
```

```
inv_predictions = scaler.inverse_transform(predictions)
inv_predictions
```

```
            [102680.88 ],
            [102266.29 ],
            [101531.95 ],
            [104217.375],
            [105093.36 ],
            [102400.445],
            [100756.92 ],
            [ 97755.445],
            [102131.836],
            [ 97946.27 ],
            [ 96917.82 ],
            [ 97071.54 ],
            [ 97039.93 ],
            [ 97013.18 ],
            [ 97054.414],
            [ 98084.83 ],
            [ 96175.72 ],
            [ 98646.35 ],
            [ 97101.68 ],
            [ 98156.125],
            [ 98163.016],
            [ 96619.3  ],
            [ 96311.69 ],
            [ 96111.984],
            [ 97326.21 ],
            [ 99074.04 ],
            [ 96502.49 ],
            [ 97179.62 ],
            [ 96836.18 ],
            [ 91621.5  ],
            [ 89160.336],
            [ 84757.305],
            [ 85454.984],
            [ 85178.35 ],
            [ 86996.125],
            [ 95757.664],
            [ 86433.086],
            [ 88127.97 ],
            [ 91811.69 ],
            [ 90867.055],
            [ 87414.16 ],
            [ 86992.27 ],
            [ 81195.35 ],
            [ 79304.98 ],
            [ 84014.625],
            [ 84820.25 ],
            [ 81950.71 ],
            [ 85131.375],
            [ 85454.125]], dtype=float32)

inv_y_test = scaler.inverse_transform(y_test)


inv_predictions[:5], inv_y_test[:5]

(array([[71379.914],
            [71404.88 ],
            [70818.805],
            [72115.43 ],
            [71232.21 ]], dtype=float32),
     array([[69987.8359375],
            [69455.34375  ],
            [70744.953125 ],
            [69892.828125 ],
            [69645.3046875]]))
```

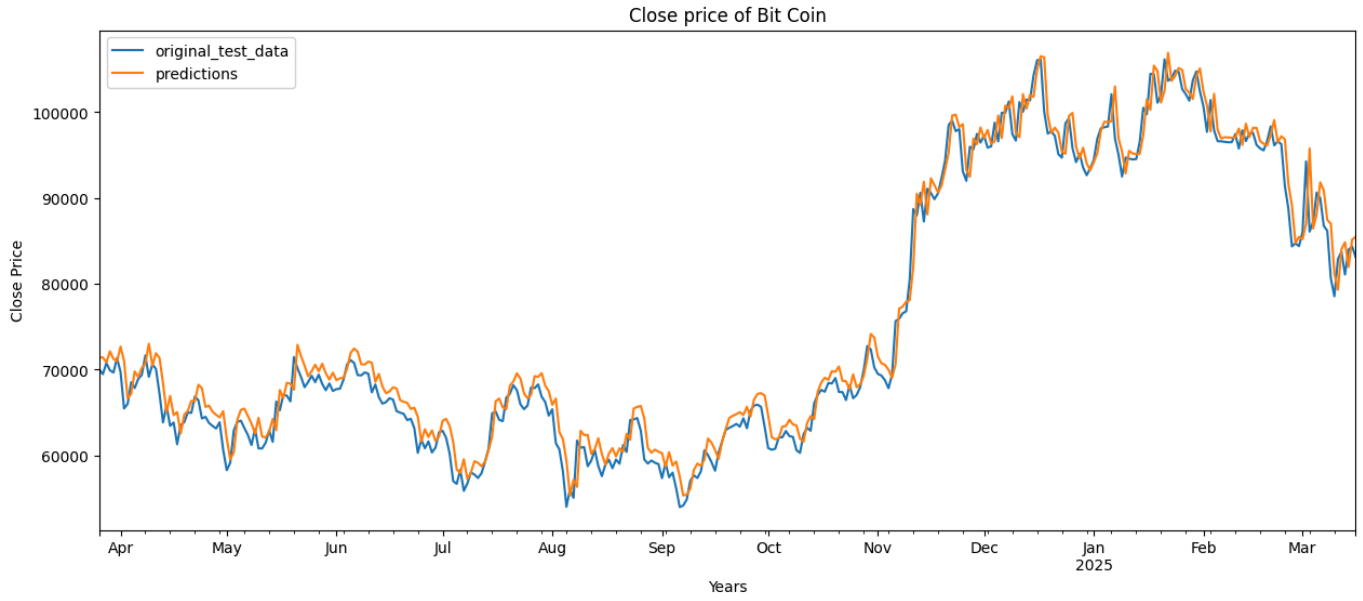
```
plotting_data = pd.DataFrame({
  'original_test_data': inv_y_test.reshape(-1),
'predictions': inv_predictions.reshape(-1),
},
index = Closing_price.index[len_train+100:]
)

plotting_data.head()
```

Date	original_test_data	predictions
2024-03-26	69987.835938	71379.914062
2024-03-27	69455.343750	71404.882812
2024-03-28	70744.953125	70818.804688
2024-03-29	69892.828125	72115.429688
2024-03-30	69645.304688	71232.210938

Next steps: Generate code with `plotting_data` | View recommended plots | New interactive sheet

```
plt.figure()
plotting_data.plot(figsize=(15,6))
plt.xlabel('Years')
plt.ylabel('Close Price')
plt.title('Close price of Bit Coin')
```

```
Text(0.5, 1.0, 'Close price of Bit Coin')
<Figure size 640x480 with 0 Axes>
```




```
last_100 = Closing_price[['Close']].tail(100)
last_100.tail()
```

Price	Close
Ticker	BTC-USD
Date	
2025-03-12	83722.359375
2025-03-13	81066.703125
2025-03-14	83969.101562
2025-03-15	84343.109375
2025-03-16	83110.046875

```
last_100 = scaler.fit_transform(last_100['Close'].values.reshape(-1,1)).reshape(1,-1,1)
last_100
```

```
        [0.93705444],
        [0.81688247],
        [0.85045405],
```

```
            [0.31469949],
            [0.43787377],
            [0.41390659],
            [0.29733443],
            [0.27603826],
            [0.07492646],
            [0.        ],
            [0.15681065],
            [0.18795935],
            [0.09178963],
            [0.19689466],
            [0.21043867],
            [0.16578557]]])
```

```
last_100.shape
```

```
(1, 100, 1)
```

```
day101 = model.predict(last_100)
day101
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step
array([[0.18490803]], dtype=float32)
```

```
scaler.inverse_transform(day101)
```

```
array([[83638.1]], dtype=float32)
```

```
# tmr day 101 ==> past 100
#     day 102 ==> day101 + past 99 days
#     day 103 ==> day 102 + + day 101 + past 98 days
```

```
def predict_future(no_of_days, prev_100):
    future_predictions = []

    # Convert prev_100 to a NumPy array (if not already)
    prev_100 = np.array(prev_100, dtype=np.float32)

    for i in range(no_of_days):
        # Predict the next value
        next_day = model.predict(prev_100)

        # Inverse transform to get actual scale
        next_day_scaled = scaler.inverse_transform(next_day)

        # Append to the list of predictions
        future_predictions.append(next_day_scaled[0][0])

        # Update prev_100 by shifting left and adding new prediction
        prev_100 = np.roll(prev_100, shift=-1, axis=1)
        prev_100[0, -1, 0] = next_day  # Add new predicted value

    return future_predictions
```

```
last_100 = Closing_price[['Close']].tail(100).values  # Extract the last 100 prices
last_100 = scaler.transform(last_100.reshape(-1, 1)).reshape(1, 100, 1)  # Normalize and reshape



no_of_days = 10
future_results = predict_future(no_of_days, last_100)
print(future_results)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 87ms/step
<ipython-input-58-e25a51b5d089>:19: DeprecationWarning: Conversion of an array with ndim > 0 to a scala
  prev_100[0, -1, 0] = next_day  # Add new predicted value
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 86ms/step
<ipython-input-58-e25a51b5d089>:19: DeprecationWarning: Conversion of an array with ndim > 0 to a scala
  prev_100[0, -1, 0] = next_day  # Add new predicted value
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step
<ipython-input-58-e25a51b5d089>:19: DeprecationWarning: Conversion of an array with ndim > 0 to a scala
  prev_100[0, -1, 0] = next_day  # Add new predicted value
<ipython-input-58-e25a51b5d089>:19: DeprecationWarning: Conversion of an array with ndim > 0 to a scala
  prev_100[0, -1, 0] = next_day  # Add new predicted value
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 104ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step<ipython-input-58-e25a51b5d089>:19: DeprecationWarning: Conversion
  prev_100[0, -1, 0] = next_day  # Add new predicted value
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step
<ipython-input-58-e25a51b5d089>:19: DeprecationWarning: Conversion of an array with ndim > 0 to a scala
  prev_100[0, -1, 0] = next_day  # Add new predicted value
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step
<ipython-input-58-e25a51b5d089>:19: DeprecationWarning: Conversion of an array with ndim > 0 to a scala
  prev_100[0, -1, 0] = next_day  # Add new predicted value
<ipython-input-58-e25a51b5d089>:19: DeprecationWarning: Conversion of an array with ndim > 0 to a scala
  prev_100[0, -1, 0] = next_day  # Add new predicted value
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 109ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 97ms/step<ipython-input-58-e25a51b5d089>:19: DeprecationWarning: Conversion
  prev_100[0, -1, 0] = next_day  # Add new predicted value
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 135ms/step
[83638.1, 83888.664, 84148.17, 84407.84, 84667.67, 84927.95, 85189.0, 85451.125, 85714.65, 85979.89]
<ipython-input-58-e25a51b5d089>:19: DeprecationWarning: Conversion of an array with ndim > 0 to a scala
  prev_100[0, -1, 0] = next_day  # Add new predicted value
```

```
future_results = np.array(future_results).reshape(-1,1)
plt.figure()
pd.DataFrame(future_results).plot(figsize=(15,5), marker='o')
for i in range(len(future_results)):
    plt.text(i,future_results[i],int(future_results[i][0]))
plt.xlabel('Future days')
plt.ylabel('Close price')
plt.title("Future Close price of Bit coin")
```

```
Text(0.5, 1.0, 'Future Close price of Bit coin')
<Figure size 640x480 with 0 Axes>
```