```
# Installing required libraries
!pip install cryptocmd
!pip install yfinance
!pip install keras-tuner

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from cryptocmd import CmcScraper
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import TimeSeriesSplit
from sklearn.metrics import mean_squared_error
import yfinance as yf
import tensorflow as tf
from tensorflow import keras
import keras_tuner
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, Bidirectional, Conv1
from kerastuner.tuners import BayesianOptimization
from sklearn.model_selection import RandomizedSearchCV
from keras.wrappers.scikit_learn import KerasRegressor
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/cola
Collecting cryptocmd
  Downloading cryptocmd-0.6.1-py3-none-any.whl (8.5 kB)
Collecting tablib
  Downloading tablib-3.4.0-py3-none-any.whl (45 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 45.5/45.5 kB 5.9 MB/s eta 0:0
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/pyth
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/di
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3
Installing collected packages: tablib, cryptocmd
Successfully installed cryptocmd-0.6.1 tablib-3.4.0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/cola
Requirement already satisfied: yfinance in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.10/di
Requirement already satisfied: appdirs>=1.4.4 in /usr/local/lib/python3.10/
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: cryptography>=3.3.2 in /usr/local/lib/python
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/pyt
Requirement already satisfied: pandas>=1.3.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.10/d
Requirement already satisfied: requests>=2.26 in /usr/local/lib/python3.10/
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/d
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.10/dist
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/di
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/pyt
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/pyth
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/di
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/cola
Collecting keras-tuner
  Downloading keras_tuner-1.3.5-py3-none-any.whl (176 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 176.1/176.1 kB 16.8 MB/s eta 0:
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-p
Collecting kt-legacy
  Downloading kt_legacy-1.0.5-py3-none-any.whl (9.6 kB)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/di
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/pyth
Installing collected packages: kt-legacy, keras-tuner
Successfully installed keras-tuner-1.3.5 kt-legacy-1.0.5
<ipython-input-2-7518a9eeda10>:20: DeprecationWarning: `import kerastuner`
  from kerastuner.tuners import BayesianOptimization
```

```
# Scrape Bitcoin historical data
scraper = CmcScraper("BTC", "01-01-2014","31-03-2023")
bitcoin_df = scraper.get_dataframe()

# Scrape stock market data
stock_data = yf.download("SPY", start="2014-01-01", end="2023-03-31")
stock_df = stock_data["Adj Close"].to_frame().reset_index().rename(columns={"Adj

# Merge data and handle NA using interpolation
merged_df = bitcoin_df.merge(stock_df, on="Date", how="inner")
merged_df.set_index("Date", inplace=True)
merged_df.index = pd.to_datetime(merged_df.index)
merged_df.interpolate(method="time", inplace=True)

# Normalize features separately
scaler_btc = MinMaxScaler()
merged_df["Close"] = scaler_btc.fit_transform(merged_df[["Close"]])
scaler_stock = MinMaxScaler()
merged_df["stock_price"] = scaler_stock.fit_transform(merged_df[["stock_price"]]
scaler_volume = MinMaxScaler()
merged_df["Volume"] = scaler_volume.fit_transform(merged_df[["Volume"]])
```

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

```
merged_df = merged_df.sort_values(by = ['Date'])
merged_df.head()
```

Date	Open	High	Low	Close	Volume	Market Cap	stock_p
2014-01-02	773.440002	820.309998	767.210022	0.009264	0.000092	9.781074e+09	0.0
2014-01-03	802.849976	834.150024	789.119995	0.009506	0.000090	9.980135e+09	0.0
2014-01-06	936.049988	1017.119995	905.710022	0.011503	0.000226	1.162053e+10	0.0
2014-							

```
# Prepare data
def prepare_data(df, feature_columns, target_column, n_past, n_future):
    x_data, y_data = [], []
    for i in range(n_past, len(df) - n_future + 1):
        x_data.append(df[feature_columns].iloc[i - n_past:i].values)
        y_data.append(df[target_column].iloc[i:i + n_future].values)
    return np.array(x_data), np.array(y_data)

n_past = 30
n_future = 1
feature_columns = ["Close", "stock_price","Volume"]
target_column = "Close"
x_data, y_data = prepare_data(merged_df, feature_columns, target_column, n_past,

# Split into train and test sets
# Set the cutoff date
train_date = "2022-03-01"
# Calculate the index of the train_date
train_date_index = merged_df.index.get_loc(pd.Timestamp(train_date), method='nea

# Calculate the train_size based on the date
train_size = len(merged_df.loc[:train_date]) - n_past

x_train, x_test = x_data[:train_size], x_data[train_size:]
y_train, y_test = y_data[:train_size], y_data[train_size:]
```

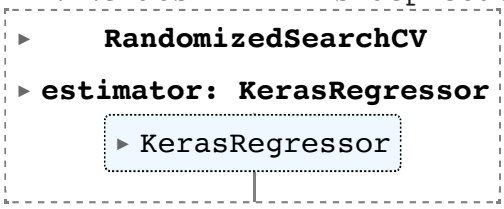
```
<ipython-input-5-03e0b99bd8bb>:19: FutureWarning: Passing method to Datetim
  train_date_index = merged_df.index.get_loc(pd.Timestamp(train_date), meth
```

```
print("x_train shape:", x_train.shape)
print("y_train shape:", y_train.shape)
print("x_test shape:", x_test.shape)
print("y_test shape:", y_test.shape)
```

```
x_train shape: (2025, 30, 3)
y_train shape: (2025, 1)
x_test shape: (272, 30, 3)
y_test shape: (272, 1)
```

```
# LSTM & Hyperparameter tuning
def create_model(learning_rate=0.001, dropout_rate=0.2, neurons=50):
  model = Sequential()
  model.add(LSTM(neurons, activation="tanh", input_shape=(n_past, len(feature_co
  model.add(Dropout(dropout_rate))
  model.add(LSTM(neurons, activation="tanh", return_sequences=False))
  model.add(Dropout(dropout_rate))
  model.add(Dense(n_future))
  optimizer = tf.keras.optimizers.Adam(lr=learning_rate)
  model.compile(optimizer=optimizer, loss="mse")
  return model

model = KerasRegressor(build_fn=create_model, verbose=0)

param_dist = {
    'batch_size': [32, 64],
    'epochs': [10],
    'learning_rate': [0.01, 0.001],
    'dropout_rate': [0.2, 0.4],
    'neurons': [25, 50]
}

random_search = RandomizedSearchCV(estimator=model, param_distributions=param_di
random_search.fit(x_train, y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
<ipython-input-8-fa507ef1ec86>:18: DeprecationWarning: KerasRegressor is de
  model = KerasRegressor(build_fn=create_model, verbose=0)
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_ra
```




```
print("Best parameters: ", random_search.best_params_)
best_model = random_search.best_estimator_.model
```

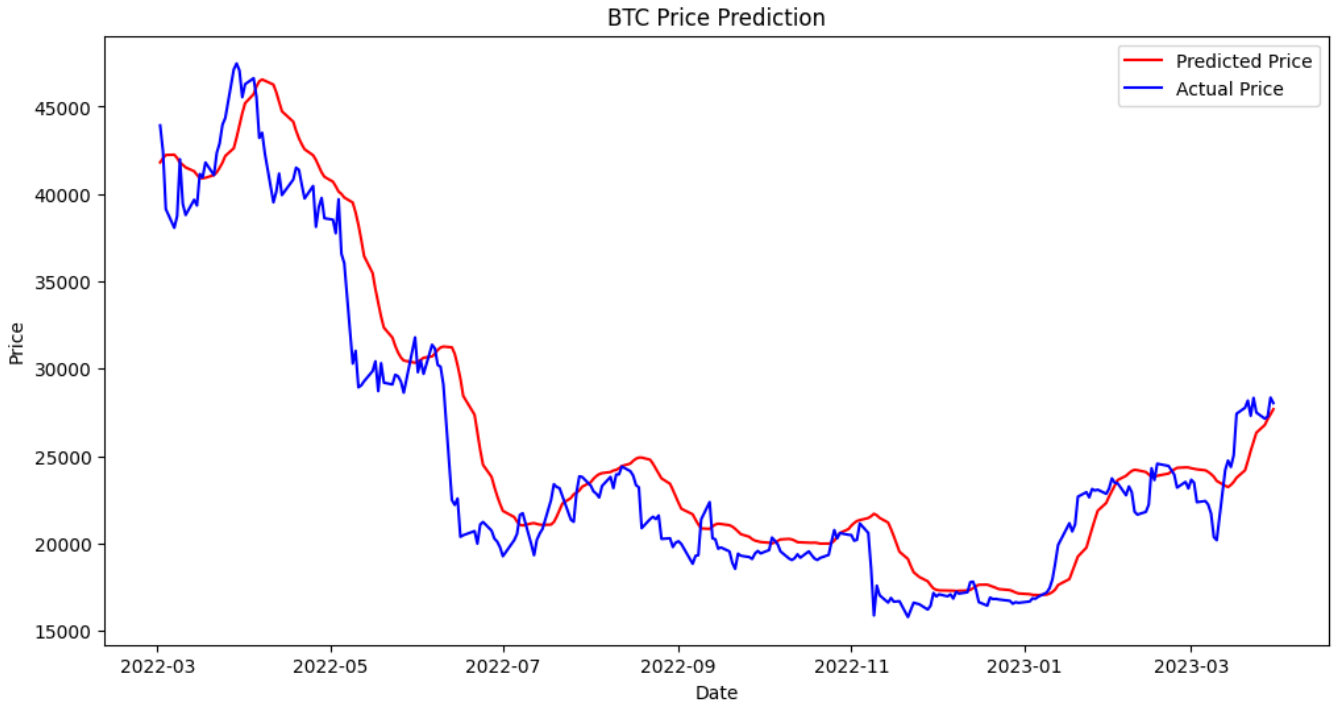
```
Best parameters:  {'neurons': 25, 'learning_rate': 0.01, 'epochs': 10, 'dro
```

```
# Make predictions
y_pred = best_model.predict(x_test)
```

```
9/9 [==============================] - 1s 3ms/step
```

```
# Invert the scaling for predictions
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
y_pred_actual = scaler_btc.inverse_transform(y_pred)
y_test_actual = scaler_btc.inverse_transform(y_test)
# Evaluate the model
mse = mean_squared_error(y_test_actual, y_pred_actual)
mae = mean_absolute_error(y_test_actual, y_pred_actual)
r2 = r2_score(y_test_actual, y_pred_actual)

print("Mean Squared Error: {:.2f}".format(mse))
print("Mean Absolute Error: {:.2f}".format(mae))
print("R2 Score: {:.2f}".format(r2))

# Visualize the results
plt.figure(figsize=(12, 6))
plt.plot(merged_df.index[-len(y_pred_actual):], y_pred_actual, label="Predicted
plt.plot(merged_df.index[-len(y_test_actual):], y_test_actual, label="Actual Pri
plt.xlabel("Date")
plt.ylabel("Price")
plt.legend(loc="best")
plt.title("Daily Price Prediction")
plt.show()
```

```
Mean Squared Error: 7076554.57
Mean Absolute Error: 1901.04
R2 Score: 0.90
```



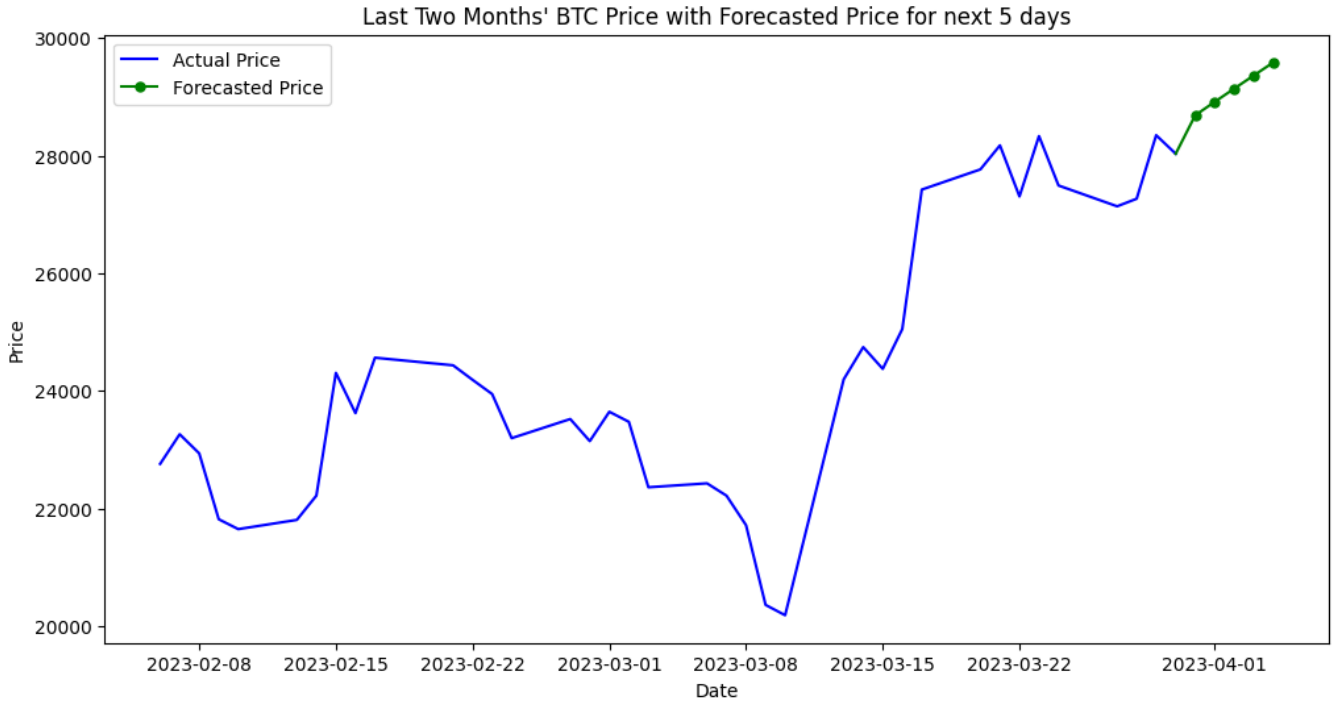
```
# Forecast for the next day
x_forecast = merged_df[feature_columns].values[-n_past:]
x_forecast = x_forecast.reshape((1, n_past, len(feature_columns)))

# Predict the future price
y_forecast = best_model.predict(x_forecast)

# Invert the scaling for the forecasted price
forecasted_price_actual = scaler_btc.inverse_transform(y_forecast)

print(forecasted_price_actual)
```

```
1/1 [==============================] - 0s 19ms/step
[[27957.908]]
```

```
# Create dataframes for actual prices with dates as their index
actual_price_df = pd.DataFrame(scaler_btc.inverse_transform(merged_df[["Close"]]

# Forecast for the next 5 days
x_forecast = merged_df[feature_columns].values[-n_past:]
forecasted_prices = []
forecasted_dates = []
for i in range(5):
    x_forecast = x_forecast.reshape((1, n_past, len(feature_columns)))
    y_forecast = best_model.predict(x_forecast)
    forecasted_price_actual = scaler_btc.inverse_transform(y_forecast)
    forecasted_prices.append(forecasted_price_actual[0][0])
    next_day = merged_df.index[-1] + pd.Timedelta(days=i+1)
    forecasted_dates.append(next_day)
    new_row = np.array([y_forecast[0][0], x_forecast[0][-1][1], x_forecast[0][-1
    x_forecast = np.append(x_forecast, new_row)
    x_forecast = x_forecast[-n_past * len(feature_columns):].reshape((n_past, le

# Add the forecasted prices to a new dataframe
forecasted_price_df = pd.DataFrame(forecasted_prices, index=forecasted_dates, co

# Limit the data to last two months
last_two_months = forecasted_price_df.index[-1] - pd.DateOffset(months=2)
actual_price_df_last_two_months = actual_price_df[last_two_months:]
forecasted_price_df_last_two_months = forecasted_price_df[last_two_months:]

# Visualize the last two months of actual vs forecasted data
plt.figure(figsize=(12, 6))
plt.plot(actual_price_df_last_two_months.index, actual_price_df_last_two_months[
plt.plot(forecasted_price_df_last_two_months.index, forecasted_price_df_last_two

# Joining the forecasted line with the actual prices
```

```
last_actual_price = actual_price_df_last_two_months["Actual Price"].iloc[-1]
first_forecasted_price = forecasted_price_df_last_two_months["Forecasted Price"]
plt.plot([actual_price_df_last_two_months.index[-1], forecasted_price_df_last_tw

plt.xlabel("Date")
plt.ylabel("Price")
plt.legend(loc="best")
plt.title("Forecasted Price for next 5 days")
plt.show()
```

```
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
```



Colab paid products - Cancel contracts here