```
# Installing the required libraries
!pip install cryptocmd
!pip install yfinance

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

!pip install keras-tuner
import keras_tuner
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, Bidirectional
from kerastuner.tuners import BayesianOptimization
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/cola
Requirement already satisfied: cryptocmd in /usr/local/lib/python3.10/dist-
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: tablib in /usr/local/lib/python3.10/dist-pac
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: 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
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: pandas>=1.3.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: cryptography>=3.3.2 in /usr/local/lib/python
Requirement already satisfied: pytz>=2022.5 in /usr/local/lib/python3.10/di
Requirement already satisfied: numpy>=1.16.5 in /usr/local/lib/python3.10/d
Requirement already satisfied: html5lib>=1.1 in /usr/local/lib/python3.10/d
Requirement already satisfied: frozendict>=2.3.4 in /usr/local/lib/python3.
Requirement already satisfied: appdirs>=1.4.4 in /usr/local/lib/python3.10/
Requirement already satisfied: beautifulsoup4>=4.11.1 in /usr/local/lib/pyt
Requirement already satisfied: lxml>=4.9.1 in /usr/local/lib/python3.10/dis
Requirement already satisfied: multitasking>=0.0.7 in /usr/local/lib/python
Requirement already satisfied: requests>=2.26 in /usr/local/lib/python3.10/
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: idna<4,>=2.5 in /usr/local/lib/python3.10/di
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: pycparser in /usr/local/lib/python3.10/dist-
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/cola
Requirement already satisfied: keras-tuner in /usr/local/lib/python3.10/dis
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-
Requirement already satisfied: kt-legacy in /usr/local/lib/python3.10/dist-
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: certifi>=2017.4.17 in /usr/local/lib/python3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/di
```

```
# Scrape cryptocurrency 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

# Create a date range that includes all dates between start and end dates
date_range = pd.date_range(start="2014-01-01", end="2023-03-31")
date_range_df = pd.DataFrame(date_range, columns=["Date"])

# Merge stock_df with the date_range_df
stock_df = stock_df.merge(date_range_df, on="Date", how="outer")
stock_df.sort_values("Date", inplace=True)


# Fill missing values using ffill and bfill methods
stock_df["stock_price"].fillna(method="bfill", inplace=True)
stock_df["stock_price"].fillna(method="ffill", inplace=True)


# Merge the filled stock_df with bitcoin_df
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)

# Add day, week, and month columns
merged_df["day"] = merged_df.index.day
merged_df["week"] = merged_df.index.week
merged_df["month"] = merged_df.index.month
merged_df = merged_df.sort_values(by = ['Date'])
```

```
[*********************100%***********************]  1 of 1 completed
<ipython-input-14-9d4f776bf96d>:31: FutureWarning: weekofyear and week have
  merged_df["week"] = merged_df.index.week
```

```
# 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"]]
```

```
merged_df.head(7)
```

Date	Open	High	Low	Close	Volume	Market Cap	stock
2014-01-01	754.969971	775.349976	754.969971	0.008804	22489400.0	9.403308e+09	0
2014-01-02	773.440002	820.309998	767.210022	0.009264	38489500.0	9.781074e+09	0
2014-01-03	802.849976	834.150024	789.119995	0.009506	37810100.0	9.980135e+09	0
2014-01-04	823.270020	859.510010	801.669983	0.010112	38005000.0	1.047736e+10	0
2014-01-05	858.549988	952.400024	854.520020	0.011210	72898496.0	1.137966e+10	0

```
# Calculate the number of months since the start of the data
merged_df['month_number'] = ((merged_df.index.year - merged_df.index[0].year) *

# Calculate monthly averages
monthly_avg = merged_df.groupby('month_number').mean()
monthly_avg.drop(['day', 'week', 'month'], axis=1, inplace=True)
```

```
monthly_avg.tail(7)
```

month_number	Open	High	Low	Close	Volume	Ma
104	19821.353753	20199.349523	19367.072575	0.291246	3.744241e+10	3.7
105	19616.090194	19870.064232	19380.976069	0.288957	3.090011e+10	3.7
106	17711.480692	18004.313993	17278.805876	0.258540	4.081772e+10	3.3
107	16969.578848	17109.241429	16811.191622	0.248877	1.746312e+10	3.2
108	20038.262513	20460.601251	19862.708316	0.297867	2.228803e+10	3.9
109	23304.085993	23690.400750	22938.975832	0.343180	2.585602e+10	4.4
110	24945.340494	25641.198165	24461.377163	0.370074	2.849354e+10	4.8

```
# 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)
```

```
# Monthly Prediction
# Split into train and test sets
# Set the train_date index
train_month_index = 72
n_past = 20
n_future = 1
feature_columns = ["Close", "stock_price"]
target_column = "Close"
x_data, y_data = prepare_data(monthly_avg, feature_columns, target_column, n_pas

# Calculate the train_size based on the index
train_size_month = train_month_index - n_past

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

```
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: (52, 20, 2)
y_train shape: (52, 1)
x_test shape: (39, 20, 2)
y_test shape: (39, 1)
```

```
# LSTM & Hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV
from keras.wrappers.scikit_learn import KerasRegressor
import tensorflow as tf

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-43-bc25ce849c0c>:17: DeprecationWarning: KerasRegressor is d
  model = KerasRegressor(build_fn=create_model, verbose=0)
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_ra
```

▸ **RandomizedSearchCV**

▸ **estimator: KerasRegressor**

▸ KerasRegressor

```
# Training the model
best_model = random_search.best_estimator_.model
best_model.fit(x_train, y_train, epochs=random_search.best_params_['epochs'], ba
```

```
Epoch 1/10
1/1 [==============================] - 0s 34ms/step - loss: 0.0028
Epoch 2/10
1/1 [==============================] - 0s 26ms/step - loss: 0.0027
Epoch 3/10
1/1 [==============================] - 0s 26ms/step - loss: 0.0033
Epoch 4/10
1/1 [==============================] - 0s 26ms/step - loss: 0.0032
Epoch 5/10
1/1 [==============================] - 0s 25ms/step - loss: 0.0028
Epoch 6/10
1/1 [==============================] - 0s 26ms/step - loss: 0.0023
Epoch 7/10
1/1 [==============================] - 0s 27ms/step - loss: 0.0017
Epoch 8/10
1/1 [==============================] - 0s 27ms/step - loss: 0.0032
Epoch 9/10
1/1 [==============================] - 0s 27ms/step - loss: 0.0024
Epoch 10/10
1/1 [==============================] - 0s 28ms/step - loss: 0.0025
<keras.callbacks.History at 0x7f5a2fd7a2c0>
```

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

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

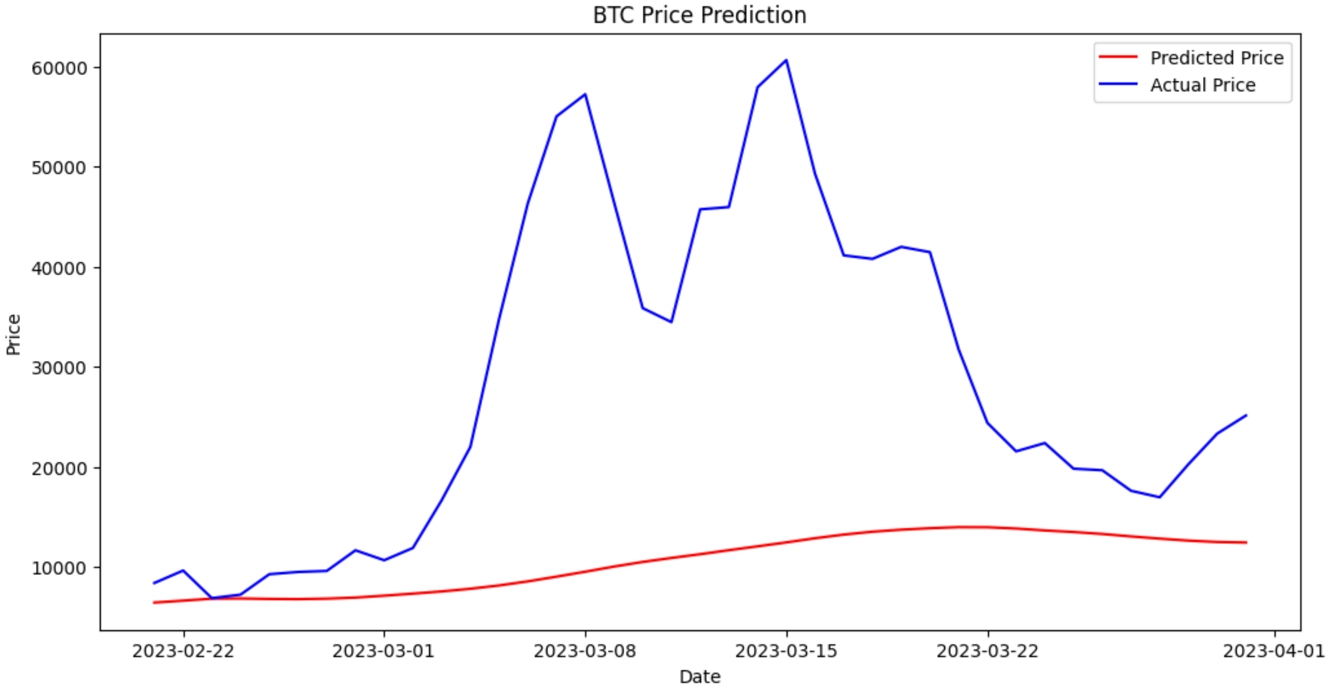
```
# Invert the scaling for predictions
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
y_pred_actual = scaler_btc.inverse_transform(y_pred)
y_test_actual = scaler_btc.inverse_transform(y_test)
# Evaluate the model
mse = mean_squared_error(y_test_actual, y_pred_actual)
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("BTC Price Prediction")
plt.show()
```

```
Mean Squared Error: 554513519.50
Mean Absolute Error: 17952.07
R2 Score: -1.09
```




```
from pandas.tseries.offsets import DateOffset

# Create a function to convert month numbers back to dates
def month_num_to_date(month_num, start_date):
    return start_date + DateOffset(months=month_num)

start_date = pd.Timestamp("2014-01-01")
```

```
# Create dataframes for actual prices with dates as their index
actual_price_df = pd.DataFrame(scaler_btc.inverse_transform(monthly_avg[["Close"
                               index=monthly_avg.index.map(lambda x: month_num_t
                               columns=["Actual Price"])

# Forecast for the next 5 months
x_forecast = monthly_avg[feature_columns].values[-n_past:]
forecasted_prices = []

for i in range(1, 6):  # i starts from 1 to 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_month = monthly_avg.index[-1] + i
    new_row = np.array([y_forecast[0][0], x_forecast[0][-1][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_months = [month_num_to_date(month_num, start_date) for month_num in r
forecasted_price_df = pd.DataFrame(forecasted_prices, index=forecasted_months, c

# Limit the data to last 10 months
last_ten_months = forecasted_price_df.index[-1] - pd.DateOffset(months=10)
actual_price_df_last_ten_months = actual_price_df[last_ten_months:]
forecasted_price_df_last_ten_months = forecasted_price_df[forecasted_price_df.in

# Visualize the last 10 months of actual vs forecasted data
plt.figure(figsize=(12, 6))
plt.plot(actual_price_df_last_ten_months.index, actual_price_df_last_ten_months[
plt.plot(forecasted_price_df_last_ten_months.index, forecasted_price_df_last_ten

# Joining the forecasted line with the actual prices
if not actual_price_df_last_ten_months.empty:
    last_actual_price = actual_price_df_last_ten_months["Actual Price"].iloc[-1]
    first_forecasted_price = forecasted_price_df_last_ten_months["Forecasted Pri
    plt.plot([actual_price_df_last_ten_months.index[-1], forecasted_price_df_las

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

```
1/1 [==============================] - 0s 123ms/step
1/1 [==============================] - 0s 175ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 190ms/step
1/1 [==============================] - 0s 78ms/step
```