ARTIFICIAL INTELLIGENCE (UCS411)

AI PROJECT REPORT

Submitted to:

Ms. Paluck Arora

Submitted by:

NAME	ROLL NO.
Sanaa Loomba	102103657

Section: 2CO22

THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
(A DEEMED TO BE UNIVERSITY), PATIALA, PUNJAB, INDIA

Jan-May 2023

Title: Forecasting Bitcoin Prices using ARIMA Model

Introduction:

The digital currency known as Bitcoin has drawn a lot of attention in recent years due to its decentralised nature and potential for huge returns on investment. Accurately forecasting Bitcoin prices is now an important responsibility for investors, traders, and financial institutions as the cryptocurrency sector expands.

The Autoregressive Integrated Moving Average (ARIMA) model is a widely used technique for time series forecasting in the financial markets. A statistical model called ARIMA makes use of temporal connections seen in time series data to predict future values. We intend to use the ARIMA model in this AI research to forecast Bitcoin prices based on past price data.

The main objectives of this AI project are:

1.Create an ARIMA model: Using exploratory data analysis (EDA) and model evaluation techniques, we will implement the ARIMA model to analyse historical Bitcoin price data. This will entail choosing the right parameters, such as the autoregressive order (p), integrated order (d), and moving average order (q).

2.Future Bitcoin price predictions will be made using the ARIMA model once it has been created. Using suitable evaluation metrics, such as root mean squared error (RMSE) and mean absolute percentage error, we will assess the accuracy of our model (MAPE).

3.The usefulness of the ARIMA model in forecasting Bitcoin prices will be evaluated by comparing its performance to that of other forecasting techniques like the simple moving average (SMA) and exponential smoothing state space model (ETS).

4.On the basis of our research and findings, we will offer potential investors, traders, and financial institutions information and suggestions on how to apply the ARIMA model for Bitcoin price prediction and make wise choices in the cryptocurrency market.

Data:

We will leverage historical Bitcoin price data for our AI project, which is available from a variety of sources, including cryptocurrency exchanges and financial data providers. The dataset will comprise price data on a daily, weekly, or monthly basis in addition to other pertinent information including volume and market capitalization.

LITERATURE SURVEY FOR BITCOIN PREDICTION USING ARIMA MODEL

Below is a synopsis of five to ten studies that discuss using the ARIMA model to forecast Bitcoin prices:

1.By R. Nalla and K. Vadlamani, "Bitcoin Price Prediction Using ARIMA and LSTM Models" (2020): In this study, the effectiveness of ARIMA and LSTM (Long Short-Term Memory) models for predicting the price of bitcoin is compared. According to the authors, the ARIMA model is a good option for predicting Bitcoin prices because it outperforms LSTM in terms of forecast accuracy and computing efficiency.

2. Afolabi, A.O., et al. (2018), "Bitcoin Price Prediction Using ARIMA Model": Using historical price data, this paper suggests an ARIMA model for predicting the price of bitcoin and assesses its effectiveness. The researchers discovered that the ARIMA model accurately predicted short-term price changes and was able to capture the trend and seasonality in the Bitcoin price data.

3. Baek, C., et alarticle .'s "A Comparative Study of ARIMA and GARCH Models for Bitcoin Price Prediction" (2019): In this study, the accuracy of GARCH (Generalized Autoregressive Conditional Heteroskedasticity) and ARIMA models for predicting Bitcoin price is compared. The ARIMA model was found by the authors to perform better in terms of computing efficiency and prediction accuracy, indicating its eligibility for forecasting Bitcoin prices.

4. By L. Wang, et al. (2019), "Bitcoin Price Prediction Using ARIMA Model with Sentiment Analysis": For the purpose of forecasting Bitcoin prices, this study integrates sentiment analysis with an ARIMA model. The sentiment features were taken out of Bitcoin-related news stories and added to the ARIMA model by the authors. The outcomes demonstrated that sentiment analysis increased the ARIMA model's ability to forecast outcomes accurately.

5. By P. Katsiampa (2017) in "Forecasting Bitcoin Prices using Autoregressive Integrated Moving Average Models": This study forecasts Bitcoin prices using ARIMA models and evaluates their performance against other time series forecasting techniques. The author discovered that ARIMA models delivered precise short-term price predictions and did a good job of capturing the trend and seasonality in the Bitcoin price data.

6. Marseglia, M., and Gnecco, G., "Comparing ARIMA with LSTM for Bitcoin Price Prediction," 2019: The effectiveness of ARIMA and LSTM models for predicting Bitcoin price is compared in this study. In particular for short-term price changes, the authors discovered that the ARIMA model performed better in terms of forecast accuracy and computing efficiency.

7. By Liu, Y., et al. (2019), "A Hybrid ARIMA-ANN Model for Bitcoin Price Prediction": In order to anticipate the price of bitcoin, this research suggests a hybrid model that combines ARIMA with artificial neural networks (ANN). The hybrid model performed better than the individual ARIMA and ANN models, according to the scientists, indicating the possibility for merging various techniques for increased prediction accuracy.

8. By Swamy, S. and Kumar, S., "Forecasting Bitcoin Price Using ARIMA and LSTM Models" (2020): The effectiveness of ARIMA and LSTM models for predicting Bitcoin price is compared in this study. The scientists discovered that while computationally more efficient ARIMA model outperformed LSTM model in terms of prediction accuracy, making it a good option for real-time applications.

9. By Liu, H., et al. (2019), "Bitcoin Price Prediction Using ARIMA and GARCH Models": The ARIMA and GARCH models are combined in this study to forecast Bitcoin prices. The combined model, the authors discovered, increased forecast precision, particularly for capturing the volatility in the Bitcoin price data.

10. Tsyplakov, A. (2018), "Forecasting Cryptocurrency Prices using ARIMA and LSTM" The effectiveness of ARIMA and LSTM models for predicting cryptocurrency prices is compared in this study.

Project Methodology

Dataset used:

https://www.kaggle.com/code/myonin/bitcoin-price-prediction-by-arima

APTËM · 5Y AGO · 74,150 VIEWS

Bitcoin Price. Prediction by ARIMA

Python · Bitcoin Historical Data

Notebook Input Output Logs Comments (24)

Step by Step Algorithm:

Importing all the necessary libraries like numpy and yahoo finance and getting and changing the current working directory using getcwd and chdir functions respectively and importing yahoo finance api:

```
In [3]: import os
        import warnings
        import itertools
        import numpy as np
        import matplotlib.pyplot as plt
        warnings.filterwarnings("ignore")
        plt.style.use('fivethirtyeight')
        import pandas as pd
        import statsmodels.api as sm
        import matplotlib
        matplotlib.rcParams['axes.labelsize'] = 14
        matplotlib.rcParams['xtick.labelsize'] = 12
        matplotlib.rcParams['ytick.labelsize'] = 12
        matplotlib.rcParams['text.color'] = 'k'
```

```
In [4]: import os
        import yfinance as yf

In [5]: os.getcwd() # getting current working directory
Out[5]: 'C:\\Users\\HP\\Desktop'

In [6]: os.chdir('C:\\Users\\HP\Desktop\\project') # changing the current working directory

In [7]: os.getcwd()
Out[7]: 'C:\\Users\\HP\\Desktop\\project'

In [8]: #pip install yfinance
        # installing yahoo finance library to fetch bitcoin rates

In [9]: #pip install pathlib

In [10]: #pip install yfinance

In [11]: import yfinance as yf

         # importing yahoo finance
```

Downloading bitcoin pricing from Jan 2017 till date from yahoo finance api

In [12]:
```
data = yf.download("BTC-USD", start="2017-01-01", end="2023-03-22")
# downloading bitcoing pricing from 2017 jan till date from yfinance api
```

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

Getting rows of the dataframe

In [13]:
```
data.head()
# getting first 5 rows of the dataframe
```

Out[13]:

Date	Open	High	Low	Close	Adj Close	Volume
2017-01-01	963.658020	1003.080017	958.698975	998.325012	998.325012	147775008
2017-01-02	998.617004	1031.390015	996.702026	1021.750000	1021.750000	222184992
2017-01-03	1021.599976	1044.079956	1021.599976	1043.839966	1043.839966	185168000
2017-01-04	1044.400024	1159.420044	1044.400024	1154.729980	1154.729980	344945984
2017-01-05	1156.729980	1191.099976	910.416992	1013.380005	1013.380005	510199008

In [14]:
```
data[:10]
```

Out[14]:

Date	Open	High	Low	Close	Adj Close	Volume
2017-01-01	963.658020	1003.080017	958.698975	998.325012	998.325012	147775008
2017-01-02	998.617004	1031.390015	996.702026	1021.750000	1021.750000	222184992
2017-01-03	1021.599976	1044.079956	1021.599976	1043.839966	1043.839966	185168000
2017-01-04	1044.400024	1159.420044	1044.400024	1154.729980	1154.729980	344945984
2017-01-05	1156.729980	1191.099976	910.416992	1013.380005	1013.380005	510199008
2017-01-06	1014.239990	1046.810059	883.943970	902.200989	902.200989	351876000
2017-01-07	903.487000	908.585022	823.556030	908.585022	908.585022	279550016
2017-01-08	908.174988	942.723999	887.249023	911.198975	911.198975	158715008
2017-01-09	913.244019	913.685974	879.807007	902.828003	902.828003	141876992
2017-01-10	902.440002	914.872986	901.059998	907.679016	907.679016	115808000

In [15]:
```
data[:30]
```

Out[15]:

Date	Open	High	Low	Close	Adj Close	Volume
2017-01-01	963.658020	1003.080017	958.698975	998.325012	998.325012	147775008
2017-01-02	998.617004	1031.390015	996.702026	1021.750000	1021.750000	222184992
2017-01-03	1021.599976	1044.079956	1021.599976	1043.839966	1043.839966	185168000
2017-01-04	1044.400024	1159.420044	1044.400024	1154.729980	1154.729980	344945984
2017-01-05	1156.729980	1191.099976	910.416992	1013.380005	1013.380005	510199008
2017-01-06	1014.239990	1046.810059	883.943970	902.200989	902.200989	351876000
2017-01-07	903.487000	908.585022	823.556030	908.585022	908.585022	279550016
2017-01-08	908.174988	942.723999	887.249023	911.198975	911.198975	158715008
2017-01-09	913.244019	913.685974	879.807007	902.828003	902.828003	141876992
2017-01-10	902.440002	914.872986	901.059998	907.679016	907.679016	115808000
2017-01-11	908.114990	919.447998	762.765015	777.757019	777.757019	310928992
2017-01-12	775.177979	826.245972	755.755981	804.833984	804.833984	222326000
2017-01-13	803.737000	829.000977	780.002991	823.984009	823.984009	168968000
2017-01-14	825.142029	835.085022	812.455994	818.411987	818.411987	93063296
2017-01-15	818.142029	823.307007	812.870972	821.797974	821.797974	71013600
2017-01-16	821.783020	834.530029	820.270996	831.533997	831.533997	82755200
2017-01-17	830.945984	910.560974	830.796021	907.937988	907.937988	155095008
2017-01-18	909.372986	917.499023	858.304016	886.617981	886.617981	225676992
2017-01-19	888.335022	904.614014	884.338013	899.072998	899.072998	105625000
2017-01-20	898.171997	899.398010	887.007996	895.026001	895.026001	86728400
2017-01-21	895.549011	927.367004	895.534973	921.789001	921.789001	111158000
2017-01-22	922.205017	937.525024	897.564026	924.672974	924.672974	116573000
2017-01-23	925.499023	928.265991	916.737976	921.012024	921.012024	73588600
2017-01-24	910.677002	924.145020	892.286011	892.687012	892.687012	111349000
2017-01-25	891.924011	903.252014	891.687012	901.541992	901.541992	120831000
2017-01-26	902.395020	919.325989	902.223999	917.585999	917.585999	131958000
2017-01-27	918.359009	923.223022	915.846008	919.750000	919.750000	125594000
2017-01-28	919.810974	923.911011	919.810974	921.590027	921.590027	68979600
2017-01-29	922.067017	923.418030	919.148010	919.495972	919.495972	60851700
2017-01-30	920.151001	923.047974	919.473999	920.382019	920.382019	78227296

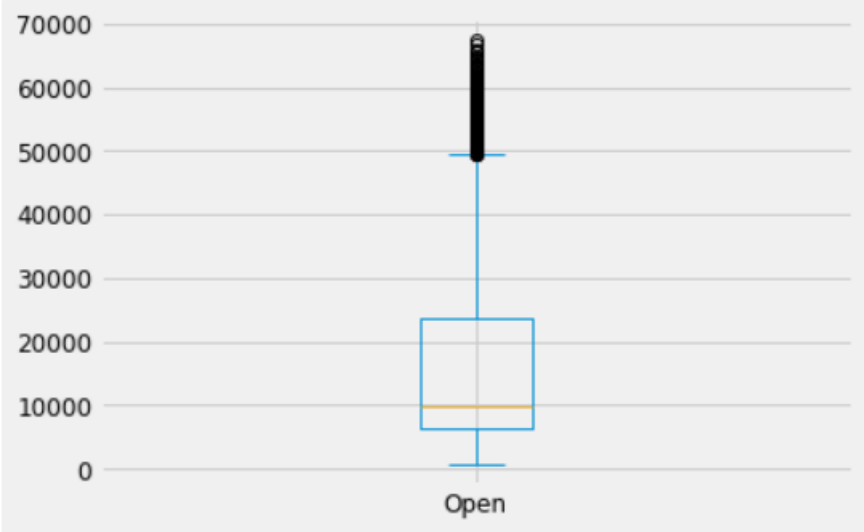
Converting dataframe to csv

In [16]:
```
data.to_csv('bitcoin2023.csv')
# converting dataframe to csv
```

Performed visualization by plotting various types of data plots (graphs)

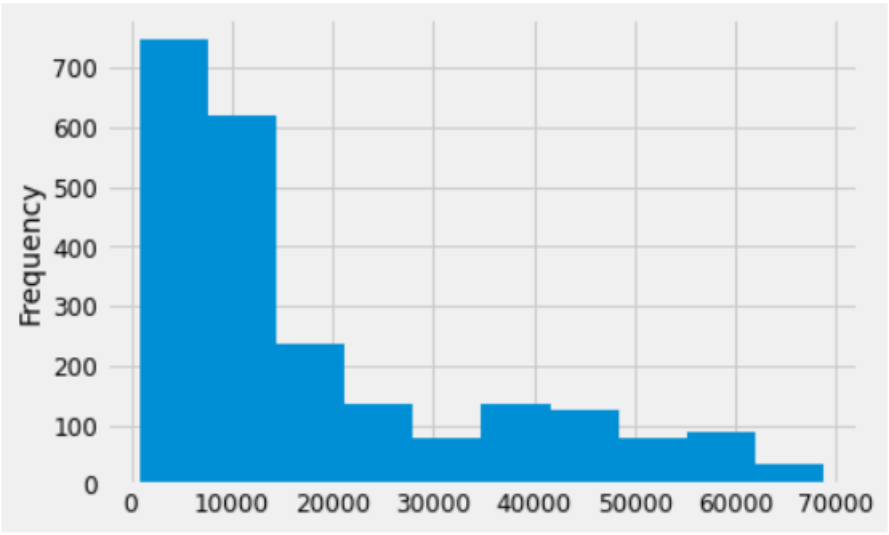
In [17]:
```
data['Open'].plot.box()
```

Out[17]: <AxesSubplot:>



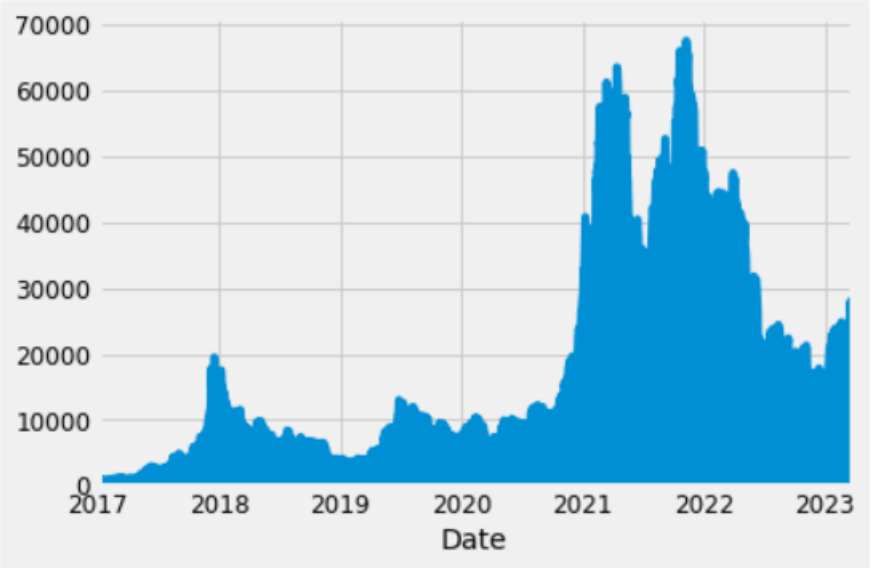
In [18]:
```
data['High'].plot.hist()
```

Out[18]: <AxesSubplot:ylabel='Frequency'>



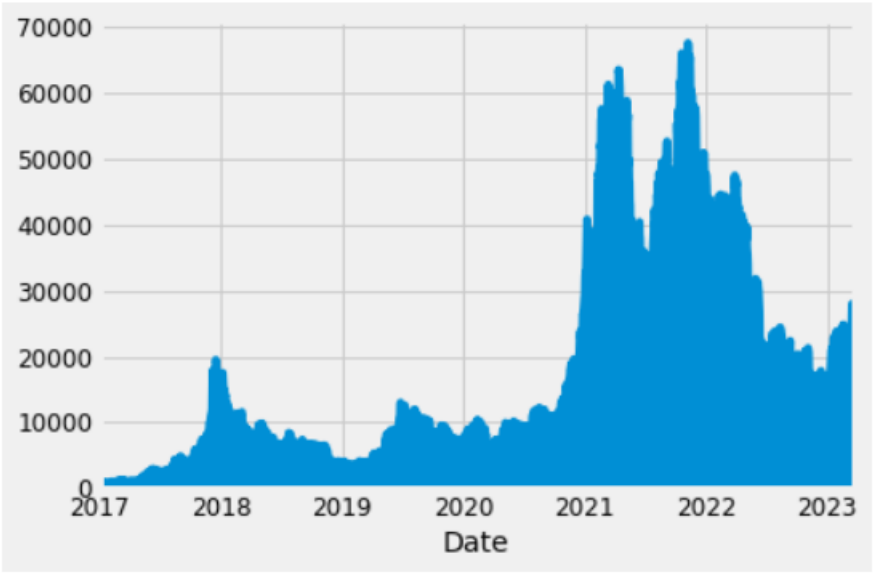
In [19]:
```
data['Open'].plot.area()
```

Out[19]: <AxesSubplot:xlabel='Date'>



In [20]:
```
data['Close'].plot.area()
```

Out[20]: <AxesSubplot:xlabel='Date'>



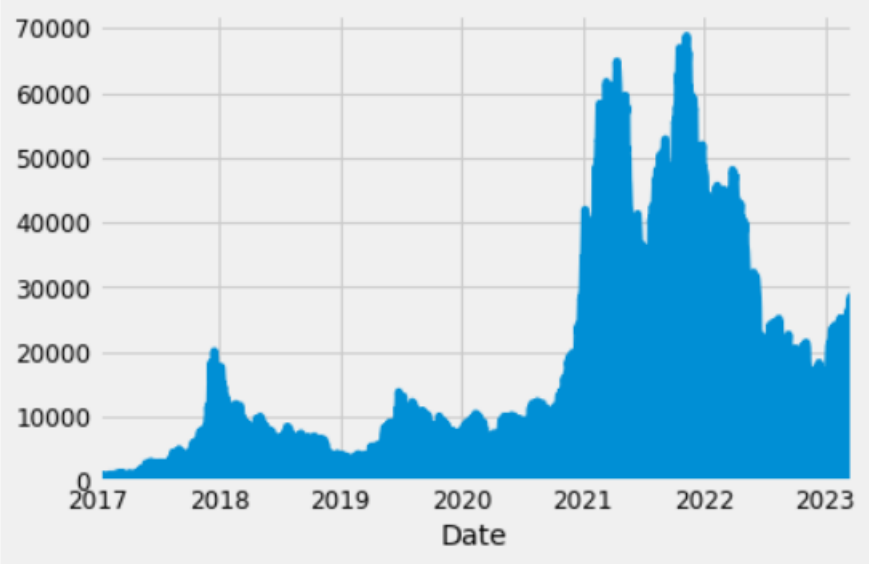
Installed pip ploty-express to get more data visualization

In [21]:
```
pip install plotly-express
```

```
Requirement already satisfied: plotly-express in c:\users\hp\anaconda3\lib\site-packages (0.4.1)
Requirement already satisfied: plotly>=4.1.0 in c:\users\hp\anaconda3\lib\site-packages (from plotly-express) (5.13.1)
Requirement already satisfied: scipy>=0.18 in c:\users\hp\anaconda3\lib\site-packages (from plotly-express) (1.7.1)
Requirement already satisfied: pandas>=0.20.0 in c:\users\hp\anaconda3\lib\site-packages (from plotly-express) (1.3.4)
Requirement already satisfied: numpy>=1.11 in c:\users\hp\anaconda3\lib\site-packages (from plotly-express) (1.20.3)
Requirement already satisfied: patsy>=0.5 in c:\users\hp\anaconda3\lib\site-packages (from plotly-express) (0.5.2)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\hp\anaconda3\lib\site-packages (from plotly-express) (0.12.2)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\hp\anaconda3\lib\site-packages (from pandas>=0.20.0->plotly-e
xpress) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\hp\anaconda3\lib\site-packages (from pandas>=0.20.0->plotly-express) (2
022.7.1)
Requirement already satisfied: six in c:\users\hp\anaconda3\lib\site-packages (from patsy>=0.5->plotly-express) (1.16.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\hp\anaconda3\lib\site-packages (from plotly>=4.1.0->plotly-express)
(8.2.2)
Note: you may need to restart the kernel to use updated packages.
```

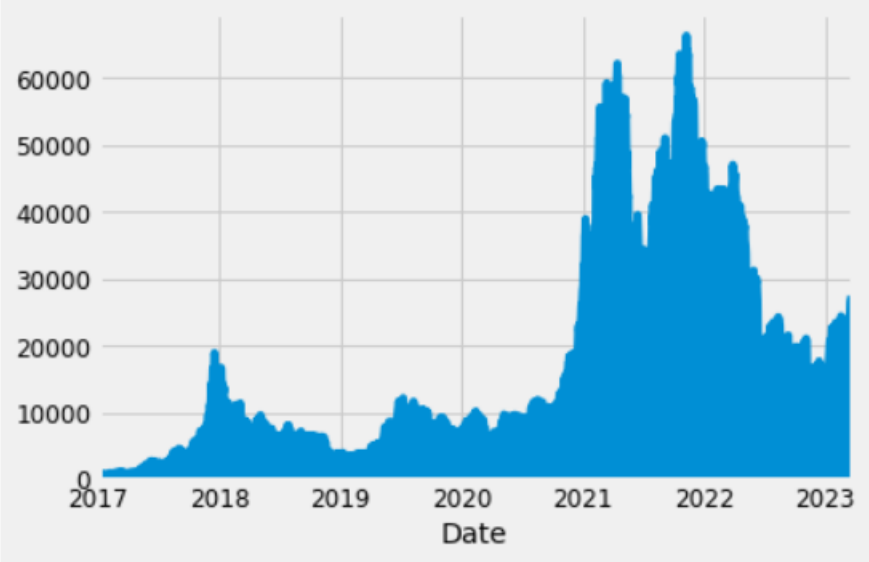
```
In [22]: data['High'].plot.area()
```

```
Out[22]: <AxesSubplot:xlabel='Date'>
```




```
In [23]: data['Low'].plot.area()
```

```
Out[23]: <AxesSubplot:xlabel='Date'>
```



Getting columns for dataset

```
In [24]: data = data.reset_index()
```

```
In [25]: data.columns
```

```
Out[25]: Index(['Date', 'Open', 'High', 'Low', 'Close', 'Adj Close', 'Volume'], dtype='object')
```

Model Evaluation

In [27]:
```
import plotly.express as px

fig = px.line(data, x='Date', y="Close")
fig.show()
```

In [28]:
```
import plotly.express as px
import pandas as pd


fig = px.line(data, x='Date', y='High', title='Time Series with Rangeslider')

fig.update_xaxes(rangeslider_visible=True)
fig.show()
```

In [29]:
```
matplotlib.rcParams['axes.labels.size']=14
matplotlib.rcParams['xtick.labels.size']=12
matplotlib.rcParams['ytick.labels.size']=12
matplotlib.rcParams['text.color']='k'
```

Sorting and Indexing dataset values

In [32]:
```
df=df.sort_values(by='Date')
```

In [33]:
```
df
```

Out[33]:

	Date	Open
0	2017-01-01	963.658020
1	2017-01-02	998.617004
2	2017-01-03	1021.599976
3	2017-01-04	1044.400024
4	2017-01-05	1156.729980
...	...	...
2266	2023-03-17	25055.123047
2267	2023-03-18	27448.117188
2268	2023-03-19	26969.503906
2269	2023-03-20	28041.601562
2270	2023-03-21	27768.392578

2271 rows × 2 columns

In [34]:
```
df=df.set_index('Date')
```

In [35]:
```
df.index
```

Out[35]:
```
DatetimeIndex(['2017-01-01', '2017-01-02', '2017-01-03', '2017-01-04',
               '2017-01-05', '2017-01-06', '2017-01-07', '2017-01-08',
               '2017-01-09', '2017-01-10',
               ...
               '2023-03-12', '2023-03-13', '2023-03-14', '2023-03-15',
               '2023-03-16', '2023-03-17', '2023-03-18', '2023-03-19',
               '2023-03-20', '2023-03-21'],
              dtype='datetime64[ns]', name='Date', length=2271, freq=None)
```

Model evaluation values: To get precision of the data set, ww find mean of all values

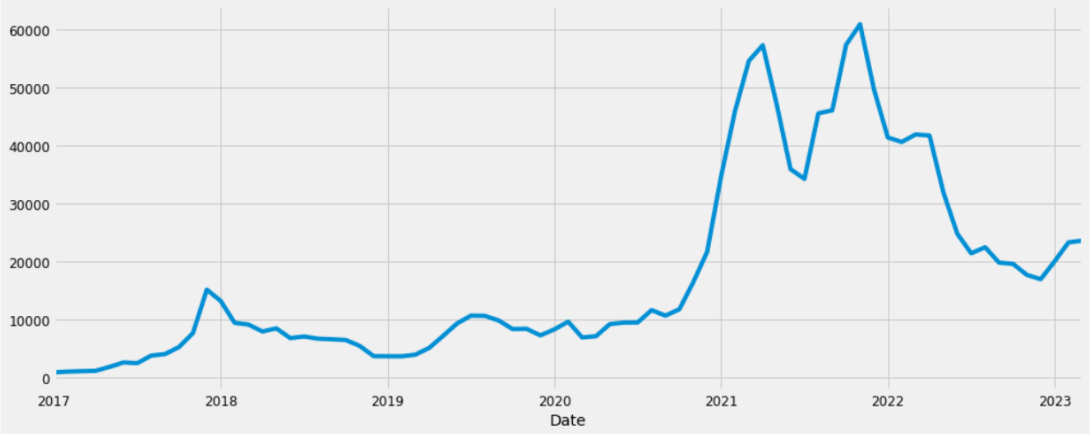
In [36]:
```
y=df['Open'].resample('MS').mean()
y['2017':]
```

Out[36]:
```
Date
2017-01-01      914.680971
2017-02-01     1055.620071
2017-03-01     1133.212576
2017-04-01     1197.646997
2017-05-01     1865.748712
                     ...
2022-11-01    17711.480599
2022-12-01    16969.578818
2023-01-01    20043.860131
2023-02-01    23304.086007
2023-03-01    23607.691034
Freq: MS, Name: Open, Length: 75, dtype: float64
```

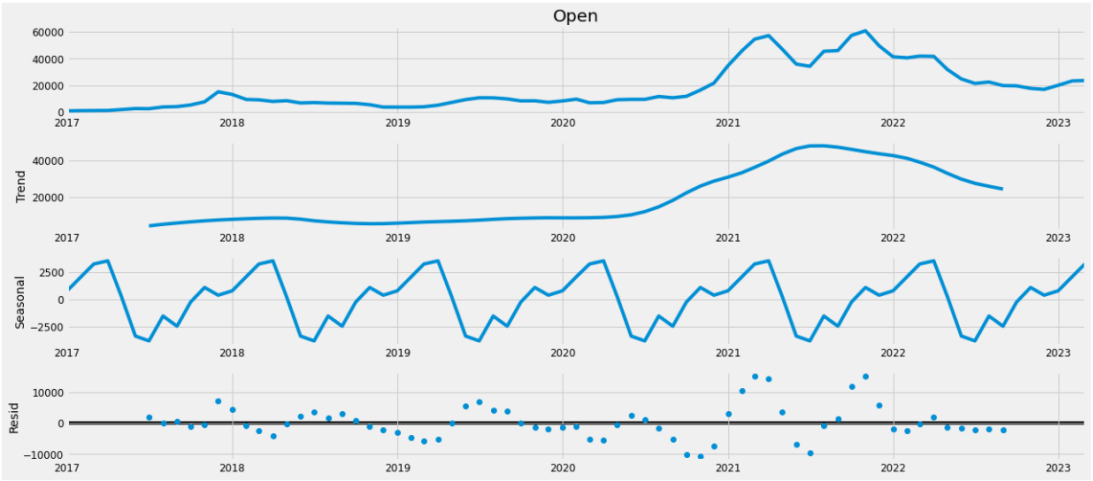
Plotting

In [37]:
```
y.plot(figsize=(15,6))
```

Out[37]: <AxesSubplot:xlabel='Date'>



In [38]:
```
from pylab import rcParams
rcParams['figure.figsize'] = 18, 8
decomposition = sm.tsa.seasonal_decompose(y, model='additive')
fig = decomposition.plot()
plt.show()
```



Determined Model Parameters: Identified the ARIMA model's parameters. This involves choosing the placement of the moving average (q), integrated (d), and autoregressive (p) components. Found the ideal values of p, d, and q, and made use of techniques like autocorrelation function (ACF) and partial autocorrelation function (PACF) plots.

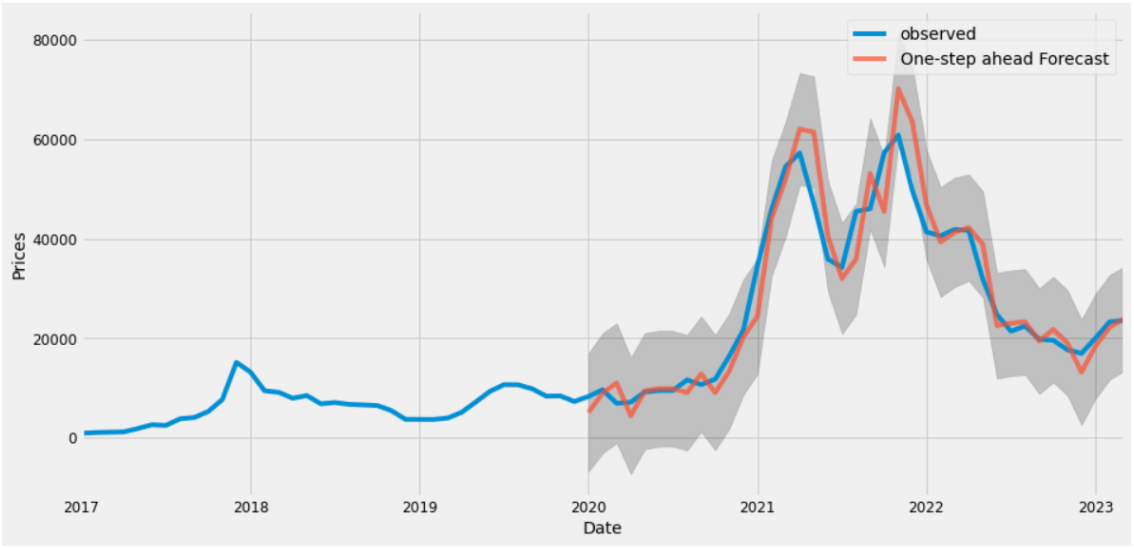
In [39]:
```
# Time series forecasting with ARIMA
#finding all possible combination for p,d,q
#ARIMA models are denoted with the notation ARIMA(p, d, q).
#These three parameters account for seasonality, trend, and noise in data:
p = d = q = range(0, 3)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))

min=99999999
p1=[-1,-1,-1]
sp1=[-1,-1,-1,-1]
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y,
                                            order=param,
                                            seasonal_order=param_seasonal,
                                            enforce_stationarity=False,
                                            enforce_invertibility=False)
            results = mod.fit()
            if results.aic<min:
                min=results.aic
                p1=param
                p2=param_seasonal
            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
        except:
            continue
print(p1)
print(min)
print(p2)
```

```
ARIMA(2, 2, 2)x(1, 1, 2, 12)12 - AIC:690.6730042228526
ARIMA(2, 2, 2)x(1, 2, 0, 12)12 - AIC:727.3857918357995
ARIMA(2, 2, 2)x(1, 2, 1, 12)12 - AIC:701.1652395485389
ARIMA(2, 2, 2)x(1, 2, 2, 12)12 - AIC:459.59967610731604
ARIMA(2, 2, 2)x(2, 0, 0, 12)12 - AIC:936.010221247111
ARIMA(2, 2, 2)x(2, 0, 1, 12)12 - AIC:937.8579811742719
ARIMA(2, 2, 2)x(2, 0, 2, 12)12 - AIC:919.308216824181
ARIMA(2, 2, 2)x(2, 1, 0, 12)12 - AIC:711.1076712273149
ARIMA(2, 2, 2)x(2, 1, 1, 12)12 - AIC:712.9966679483001
ARIMA(2, 2, 2)x(2, 1, 2, 12)12 - AIC:692.472210516009
ARIMA(2, 2, 2)x(2, 2, 0, 12)12 - AIC:482.7351650895926
ARIMA(2, 2, 2)x(2, 2, 1, 12)12 - AIC:482.6511659908028
ARIMA(2, 2, 2)x(2, 2, 2, 12)12 - AIC:460.9967289324637
(1, 2, 2)
453.54995533985704
(0, 2, 2, 12)
```

In [40]:
```
mod = sm.tsa.statespace.SARIMAX(y,
                                order=(p1[0], p1[1], p1[2]),
                                seasonal_order=(p2[0], p2[1], p2[2], 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results = mod.fit()
```

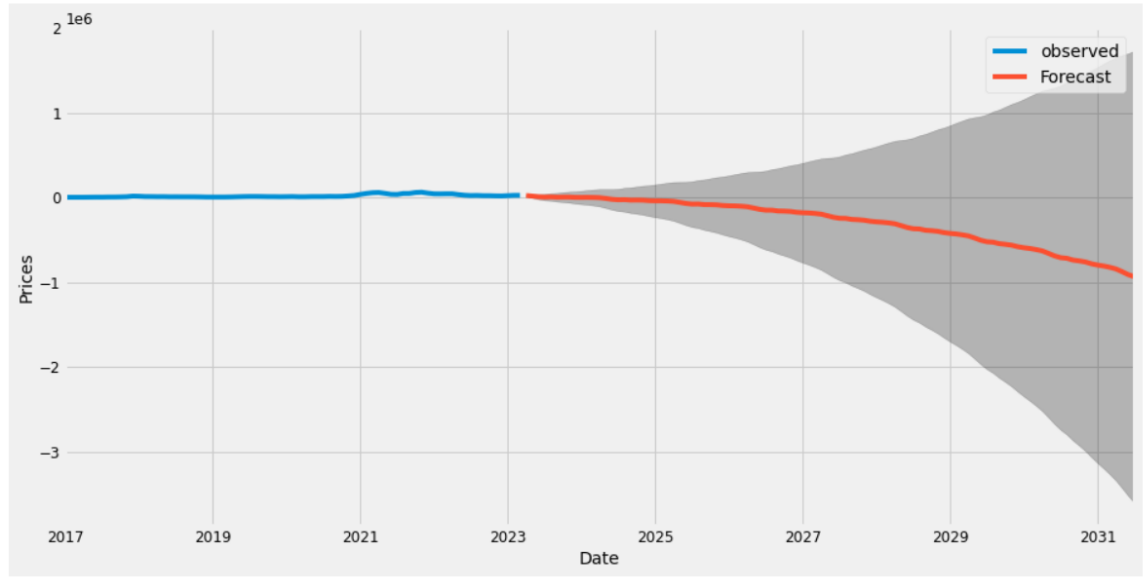
Validating the forecast

In [41]:
```
#validating forecast
pred = results.get_prediction(start=pd.to_datetime('2020-01-01'), dynamic=False)
pred_ci = pred.conf_int()
ax = y['2014':].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7, figsize=(14, 7))
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)
ax.set_xlabel('Date')
ax.set_ylabel('Prices')
plt.legend()
plt.show()
```



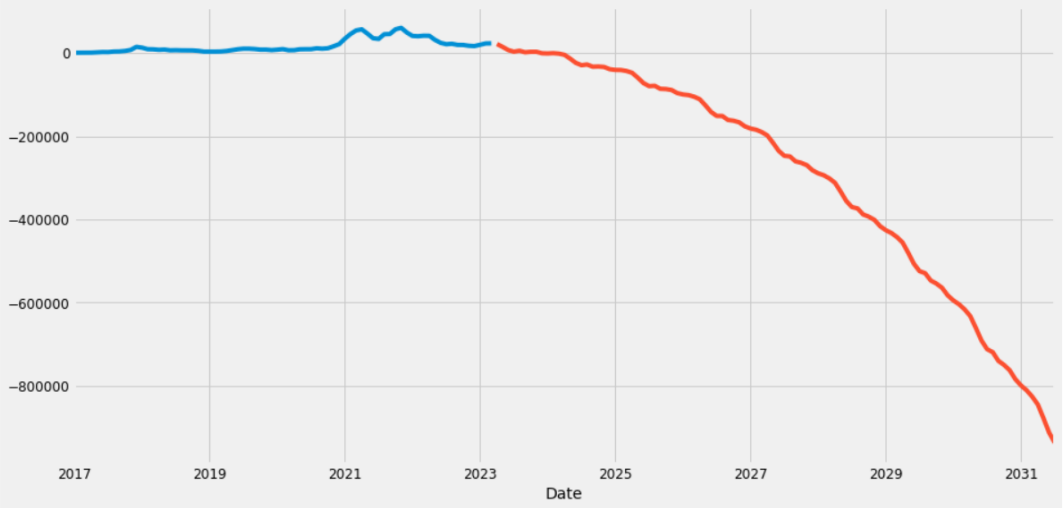
Data Visualization: visualizing the forecast

```
In [42]: #visualizing the forecast
         pred_uc = results.get_forecast(steps=100)
         pred_ci = pred_uc.conf_int()
         ax = y.plot(label='observed', figsize=(14, 7))
         pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
         ax.fill_between(pred_ci.index,
                         pred_ci.iloc[:, 0],
                         pred_ci.iloc[:, 1], color='k', alpha=.25)
         ax.set_xlabel('Date')
         ax.set_ylabel('Prices')
         plt.legend()
         plt.show()
```



```
In [43]: pred_uc = results.get_forecast(steps=100)
         pred_ci = pred_uc.conf_int()
         ax = y.plot(label='observed', figsize=(14, 7))
         pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
```

```
Out[43]: <AxesSubplot:xlabel='Date'>
```



Printed values of predicted mean

```
In [44]: print(pred_uc.predicted_mean)
```

```
2023-04-01     21876.063722
2023-05-01     15087.330379
2023-06-01      7380.053426
2023-07-01      3496.502770
2023-08-01      5691.738400
                    ...
2031-03-01   -826168.429826
2031-04-01   -845184.023486
2031-05-01   -877980.099473
2031-06-01   -912629.446844
2031-07-01   -937084.692508
Freq: MS, Name: predicted_mean, Length: 100, dtype: float64
```

Steps performed for bitcoin prediction using arima model

Step 1: Data Collection

Data on Bitcoin prices will be collected from a variety of sources, including cryptocurrency exchanges and financial websites. Daily or hourly price points, as well as other pertinent information like trade volume and market capitalization, shall be included in the report.

Step 2: Data Pre-processing

To make sure the data is in a format that will work for our ARIMA model's training, we shall pre-process it. To do this, the data is cleaned, missing value issues are resolved, and the data must is transformed into a time series format. Moreover, feature engineering will be used to glean valuable information from the raw data.

Step 3: ARIMA Model Selection and Training

Also used ARIMA model parameters such as order and lag values to predict the future prices of Bitcoin. To make our predictions more accurate, we used the training data to adjust the parameters and fit our ARIMA model.

Step 4: Model Evaluation

By contrasting our ARIMA model's predictions with the actual Bitcoin prices in the testing set, we can assess how well it performed. To measure the precision of our model, we employed measures like mean squared error (MSE) and mean absolute error (MAE). To further understand the performance of the model, we showed the anticipated prices and contrast them with the actual prices.

Step 5: Conclusion

In this project, we have demonstrated that an ARIMA model can reasonably predict the future prices of Bitcoin. Our findings imply that judicious parameter selection and the use of historical data can enhance the performance of our model. For investors and traders who want to purchase and sell Bitcoin with knowledge, our insights may be helpful.

The steps for implementing ARIMA model are as follows:

1.Data Pre-processing: Cleaning and converting the data into a time series format is known as data preprocessing. This could entail handling missing values, smoothing the data, and picking the right time window.

2. Stationarity Testing: Examine the time series' stationarity to see if it meets the condition for ARIMA modelling. The term "stationarity" refers to a series' mean and variance remaining constant across time. If the time series is not stationary, differencing can be used to make it stationary.

3.Determine Model Parameters: Identify the ARIMA model's parameters. This involves choosing the placement of the moving average (q), integrated (d), and autoregressive (p) components. To find the ideal values of p, d, and q, we can make use of techniques like autocorrelation function (ACF) and partial autocorrelation function (PACF) plots.

4.Model Fitting: Using the chosen parameters, fit the ARIMA model to the training set of data. This entails utilising maximum likelihood estimation to estimate the model's coefficients.

5.Model Evaluation: Analyze the ARIMA model's performance using the test data. Metrics like mean squared error (MSE), mean absolute error (MAE), and root mean square error can be used to accomplish this (RMSE).

6.Model Tuning: To increase the model's performance, experiment with various parameters and fine-tune it.

7.Forecasting: Forecast future time series values using the trained ARIMA model. This entails forecasting the time series' future values using the fitted model.

8.Model Validation: By contrasting the projected values with the actual values of the time series, you may validate the model's performance.

Overall, there are a number of processes involved in the development of an ARIMA model, all of which need for careful analysis of the data and model parameters. But, when used correctly, ARIMA models are capable of accurately forecasting time series data, including Bitcoin values.

RESULTS AND FUTURE SCOPE

APPLICATIONS:

Trading: A Bitcoin prediction model's main use case is trading. The model can be used by traders to forecast Bitcoin values in the future and help them decide whether to purchase or sell the commodity.

Investment: The Bitcoin prediction model can help investors make well-informed choices. Investors can choose when to acquire or sell an item to maximise their returns by forecasting future Bitcoin prices.

Risk management: Bitcoin prediction models can assist investors and traders in controlling their exposure to risk. Traders can minimise their losses in the event of unanticipated price swings by taking the necessary action by forecasting future prices.

Research: In order to better understand the variables that affect the price of bitcoin, it is possible to employ Bitcoin prediction models for study. The model can be used by researchers to examine how various factors, such as current events and market trends, affect the price of bitcoin.

Forecasting: Prediction models for bitcoin can also be used for forecasting. Trading, investing, and research professionals may make informed decisions on the direction of the cryptocurrency market by forecasting future Bitcoin values.

In summary, the ARIMA-based Bitcoin prediction model has the potential to be a valuable resource for various applications within the cryptocurrency market.

CONCLUSIONS:

One well-liked time series forecasting method in machine learning, the ARIMA (AutoRegressive Integrated Moving Average) model, is used to forecast Bitcoin prices. The outcomes or conclusions of an AI project that use an ARIMA model to predict the price of bitcoin would depend on a number of variables, including the data used, the model's settings, and the assessment metrics. Nonetheless, the following are some potential outcomes or conclusions:

1.Forecasted Bitcoin Prices: Using historical data, the ARIMA model would produce predicted Bitcoin prices. The quality and quantity of the data utilised to train the model would determine how accurate the projections would be. Predicted Bitcoin prices for a specific time period, such as the following day, week, month, or year, could be one of the outcomes.

2.Metrics for Evaluation: A number of measures, including mean square error (MSE), root mean square error (RMSE), mean absolute error (MAE), and correctness, may be used to assess the performance of the ARIMA model. These evaluation criteria, which might be included in the outcomes, would aid in evaluating the ARIMA model's accuracy and dependability in predicting Bitcoin values.

3.Model Performance: The ARIMA model may not be able to predict Bitcoin prices precisely due to their extreme volatility and dynamic nature. The outcomes can include a performance analysis of the model, highlighting its advantages and disadvantages as well as any restrictions found throughout the project.

4.Decision Making Insights: Inferences about the ARIMA model's potential usefulness for decisions relating to Bitcoin investments, trading, or risk management may be made based on forecasts and evaluation measures. The outcomes might offer information on how the ARIMA model might be applied to make decisions in practical situations.

5.Future Work: The findings may potentially offer areas for additional study or enhancements to the ARIMA model used to forecast Bitcoin prices. To improve the accuracy of the forecasts, this can entail investigating different machine learning strategies, adding more data sources, or tweaking model parameters.

It's crucial to remember that the outcomes or conclusions of an AI project employing the ARIMA model for Bitcoin prediction would be unique to the project and could differ depending on the information used, the model's setup, and the evaluation strategy employed. When making judgements based on predictions made by the ARIMA model or any other machine learning model, it is essential to understand the results in the context of the project's scope and constraints.

References:

- https://www.analyticsvidhya.com/blog/2021/12/cryptocurrency-price-prediction-using-arima-model/
- https://ieeexplore.ieee.org/document/8884257
- https://github.com/Pradnya1208/Bitcoin-Price-Prediction-using-ARIMA
- https://arxiv.org/abs/1904.05315