Predicting Bitcoin Prices via Mathematical and Financial Models: A Study of the Time-Series Analysis of the seasonal, trend, and residual components, A comprehensive use of STL Model, AutoReg, ARIMA and SARIMAX

Boulouma A., 2023

The Seasonal decomposition of time series (STL) by R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning (1990)

Seasonal decomposition of time series (STL) is a statistical technique used to decompose a time series into its seasonal, trend, and residual components. STL is particularly useful for analyzing time series data with non-linear and non-stationary trends, and for identifying seasonal patterns in the data.

The STL algorithm works by first smoothing the time series with a moving average to remove the high-frequency noise. The smoothed series is then decomposed into its seasonal, trend, and residual components using a process called loess regression. The seasonal component represents the periodic pattern in the data, such as weekly or monthly fluctuations. The trend component represents the long-term trend in the data, while the residual component represents the random fluctuations or noise that cannot be explained by the seasonal or trend components.

The STL algorithm can be represented mathematically as follows:

Let y be a time series of length n, and let S, T, and R be the seasonal, trend, and residual components, respectively. The STL algorithm can be formulated as:

Smooth the time series y with a moving average of window length m to obtain a smoothed series s. Compute the seasonal component S by subtracting the seasonal subseries from the smoothed series s. The seasonal subseries is obtained by averaging the values of y at each seasonal position (e.g., for monthly data, the seasonal subseries for January is the average of all January values in the data set). Compute the trend component T by applying loess regression to the detrended series. The detrended series is obtained by subtracting the seasonal component S from the smoothed series s. Compute the residual component R by subtracting the seasonal component S and the trend component T from the original series y.

The seasonal, trend, and residual components can be combined to obtain a reconstructed time series that closely approximates the original time series.

Given a time series y_t for $t = 1, 2, \ldots, T$, the STL method decomposes it into three components: trend T_t, seasonal S_t and remainder R_t as follows:

- **Loess Smoothing**: The first step in STL is to extract the trend component by applying a loess smoother to the original time series. Let m be the degree of the polynomial used in the loess smoother. The smoothed values, $\hat{T}_t$, are given by:

$$\hat{T}t = \ell_t + \sum j = 1^m b_j L_j(t)$$

where ℓ_t is the local polynomial regression estimate of the trend at time t, b_j are the smoothing parameters, and $L_j(t)$ are the jth degree Legendre polynomials evaluated at time t.

- **Detrending**:

The detrended values, y_t^*, are obtained by subtracting the smoothed values from the original time series:

$$y_t^* = y_t - \hat{T}_t$$

- **Seasonal Smoothing**:

The seasonal component is extracted by applying a seasonal smoother to the detrended values. Let s be the length of the seasonal period, and q be the degree of the polynomial used in the seasonal smoother. The seasonal values, $\hat{S}_t$, are given by:

$$\hat{S}t = \frac{1}{K} \sum j = 1^K y_{t+(j-1)s}^*$$

where $K = \lfloor \frac{T}{s} \rfloor$ is the number of seasonal periods in the time series.

- **Deseasonalizing**:

The deseasonalized values, y_t^{**}, are obtained by dividing the detrended values by the seasonal values:

$$y_t^{**} = \frac{y_t^*}{\hat{S}_t}$$

- **Residual Smoothing**:

The remainder component is obtained by applying a smoother to the deseasonalized values. Let r be the degree of the polynomial used in the residual smoother. The smoothed residuals, $\hat{R}_t$, are given by:

$$\hat{R}t = \sum j = 1^r c_j \epsilon_{t-j}$$

where c_j are the smoothing parameters, and ϵ_{t-j} are the residuals at time $t - j$.

- **Reconstruction**:

The final step in STL is to add the trend, seasonal and remainder components back together to obtain the reconstructed values, $\hat{y}_t$:

$$\hat{y}_t = \hat{T}_t + \hat{S}_t \cdot y_t^{**} + \hat{R}_t$$

Evaluation

The Root Mean Squared Error (RMSE) is a metric used to measure the differences between predicted and actual values. It is calculated as the square root of the mean of the squared differences between the predicted and actual values:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

where y_i is the actual value, $\hat{y}_i$ is the predicted value, and n is the number of samples.

The Akaike Information Criterion (AIC) is a metric used to compare statistical models. It measures the relative quality of a model based on its likelihood and the number of parameters used in the model:

$$\text{AIC} = 2k - 2\ln(\hat{L})$$

where k is the number of parameters in the model and $\hat{L}$ is the maximum likelihood estimate of the model.

The Bayesian Information Criterion (BIC), also known as the Schwarz criterion, is another metric used to compare statistical models. It is similar to the AIC, but places a stronger penalty on models with more parameters:

$$\text{BIC} = k\ln(n) - 2\ln(\hat{L})$$

where n is the number of samples.

The Hannan-Quinn Information Criterion (HQIC) is a modification of the AIC that is more appropriate for small sample sizes. It is defined as:

$$\text{HQIC} = 2k\ln(\ln(n)) - 2\ln(\hat{L})$$

where n is the number of samples.

Time-Series Analysis

Autoregressions

Autoregression is a statistical model that uses past observations of a time series to predict future values. The AutoReg model is a specific type of autoregression that permits models with deterministic terms, seasonal dummies, custom deterministic terms, exogenous variables, and omission of selected lags. The dynamics of an autoregressive model are given by:

$$y_t = \alpha + \sum_{i=1}^{p} \beta_i y_{t-i} + \epsilon_t$$

where y_t is the value of the time series at time t, α is a constant term, β_i are the autoregressive coefficients, p is the order of the autoregression, and ϵ_t is assumed to be a white noise process.

AutoReg allows for the inclusion of additional components in the model. These components are:

Deterministic terms

A deterministic term is a trend component that is not driven by the underlying data. The following deterministic terms are available in AutoReg:

- n: No deterministic term
- c: Constant (default)
- ct: Constant and time trend
- t: Time trend only

The dynamics of a model with a deterministic term are given by:

$$y_t = \alpha + \beta_1 t + \sum_{i=2}^{p} \beta_i y_{t-i} + \epsilon_t$$

where t is the time index.

Seasonal dummies

Seasonal dummies are binary variables that indicate whether a given observation belongs to a particular season. AutoReg allows for the inclusion of seasonal dummies where the period of the time series is denoted by s (e.g., $s = 12$ for monthly data). The dynamics of a model with seasonal dummies are given by:

$$y_t = \alpha + \sum_{i=1}^{s-1} \gamma_i I_{t-i \equiv s} + \sum_{i=1}^{p} \beta_i y_{t-i} + \epsilon_t$$

where $I_{t-i \equiv s}$ is an indicator function that equals 1 if $t - i$ is a multiple of s.

Custom deterministic terms

AutoReg also allows for the inclusion of custom deterministic terms through the use of a DeterministicProcess object. The dynamics of a model with custom deterministic terms are given by:

$$y_t = \alpha + \beta_1 t + \sum_{i=2}^{p} \beta_i y_{t-i} + \phi_t' \gamma_t + \epsilon_t$$

where ϕ_t is the design matrix for the custom deterministic terms, and γ_t is the associated vector of coefficients.

Exogenous variables

AutoReg can include exogenous variables in the model. Exogenous variables are additional features that are not part of the time series being modeled, but are believed to be related to the time series. The dynamics of a model with exogenous variables are given by:

$$y_t = \alpha + \sum_{i=1}^{p} \beta_i y_{t-i} + x_t' \gamma + \epsilon_t$$

SARIMAX

SARIMAX (Seasonal Autoregressive Integrated Moving Average with Exogenous Variables) is a statistical method used for time series forecasting. It is an extension of the ARIMA (Autoregressive Integrated Moving Average) model, which allows for the inclusion of exogenous variables. The SARIMAX model takes into account the seasonal patterns in the data, which may occur over a fixed period of time.

The general form of a SARIMAX model is:

$$SARIMAX(p, d, q)(P, D, Q)_s$$

where:

- p: the order of the autoregressive (AR) component
- d: the degree of differencing
- q: the order of the moving average (MA) component
- P: the order of the seasonal autoregressive (SAR) component
- D: the degree of seasonal differencing
- Q: the order of the seasonal moving average (SMA) component
- s: the length of the seasonal cycle

The exogenous variables can be included in the model as additional predictors. The SARIMAX model can be used to forecast future values of the time series, taking into account both the autoregressive and moving average components, as well as the seasonal patterns and exogenous variables.

References:

- Brockwell, P. J., & Davis, R. A. (2016). Introduction to time series and forecasting. Springer.
- Hyndman, R. J., & Athanasopoulos, G. (2018). Forecasting: principles and practice (2nd ed.). OTexts.
- Taylor, S. J., & Letham, B. (2018). Forecasting at scale. The American Statistician, 72(1), 37-45.

Analysis of the model

Strengths:

- STL is a well-established time series decomposition method that has been widely used in various applications.
- STL can effectively handle time series data with different types of seasonality, trend, and noise components.
- The method is flexible and allows for the adjustment of parameters such as the length of the seasonal window and the degree of smoothing.
- The decomposition results can provide insights into the underlying patterns of the time series, which can be useful in forecasting and anomaly detection.
- STL can handle missing values and outliers in the time series data.

Weaknesses:

- STL is computationally expensive, especially for long and high-frequency time series data.
- The method may not work well for non-stationary time series data with complex patterns, such as abrupt changes in trend or seasonality.
- The decomposition results may be sensitive to the choice of parameters, such as the degree of smoothing and the seasonal window length.
- The method assumes that the seasonal pattern is fixed and deterministic, which may not be true for some time series data.

Opportunities:

- The popularity of time series analysis in various industries and applications creates opportunities for further development and improvement of STL.
- The increasing availability of computing resources can help to overcome the computational challenges of STL and make it more accessible for large-scale time series data analysis.
- The use of STL in combination with other time series analysis techniques, such as machine learning models, can enhance the forecasting accuracy and predictive power of the method.

Threats:

- The development of new time series decomposition methods that are more computationally efficient and can handle more complex time series patterns may reduce the competitiveness of STL.
- The increasing availability of machine learning models for time series analysis may reduce the demand for traditional time series decomposition methods like STL.
- The potential changes in the underlying patterns of the time series data due to external factors, such as economic downturns or global pandemics, may affect the accuracy and reliability of the STL decomposition results.

References

- STL has been introduced by R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning in their paper "STL: A Seasonal-Trend Decomposition Procedure Based on Loess", published in the Journal of Official Statistics, Vol. 6, No. 1, 1990.

- Hyndman, R. J., & Athanasopoulos, G. (2018). Forecasting: Principles and Practice. OTexts.
- STL Statsmodel: https://www.statsmodels.org/dev/examples/notebooks/generated/stl_decomposition.html (https://www.statsmodels.org/dev/examples/notebooks/generated/stl_decomposition.html)
- Autoregression: https://www.statsmodels.org/dev/examples/notebooks/generated/autoregressions.html#Forecasting (https://www.statsmodels.org/dev/examples/notebooks/generated/autoregressions.html#Forecasting)

Implementation

In [3]:

```
from btc_analysis import *
from btc_data import *
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from statsmodels.tsa.seasonal import STL
from sklearn.metrics import mean_squared_error
from math import sqrt
# Load the data
df = btc_df = clean_and_transform_data(read_data("datasets/btc.csv"), read_data(

# Set the date column as the index
df.set_index('time', inplace=True)

# df.columns
# Index(['Price', 'hash_rate', 'transaction_volume', 'mining_difficulty','inflat
```

```
/var/folders/gc/m0hv5jzd1kn2nrn5y5w241g00000gn/T/ipykernel_16183/26
48613530.py:18: DtypeWarning: Columns (146) have mixed types.Specif
y dtype option on import or set low_memory=False.
  df = btc_df = clean_and_transform_data(read_data("datasets/btc.cs
v"), read_data("datasets/btc_google_trend.csv"))
```

In [2]:

```
df.columns
```

Out[2]:

```
Index(['Price', 'hash_rate', 'transaction_volume', 'mining_difficul
ty',
       'inflation_rate', 'bitcoin_trend'],
      dtype='object')
```

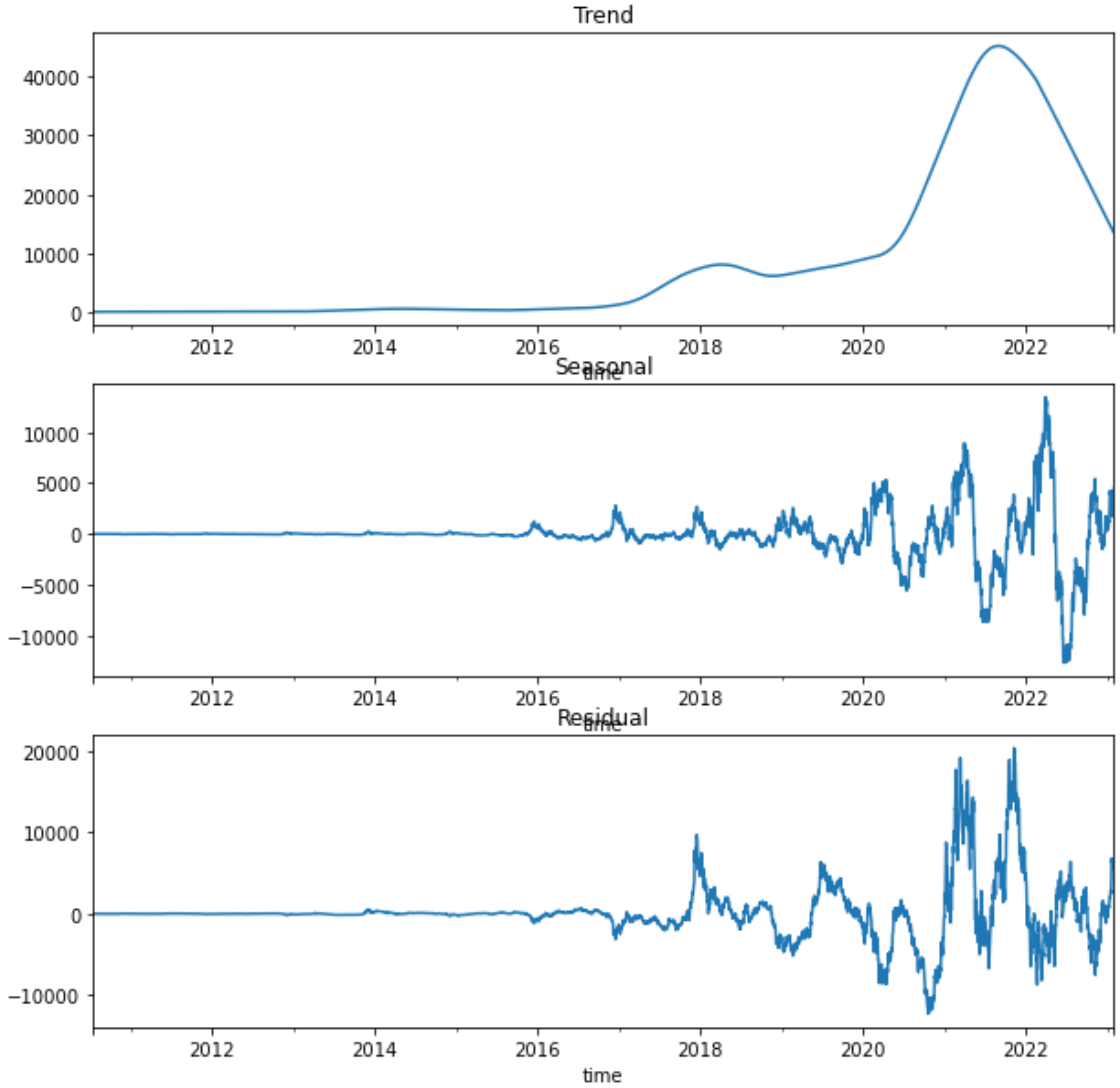
Build the model

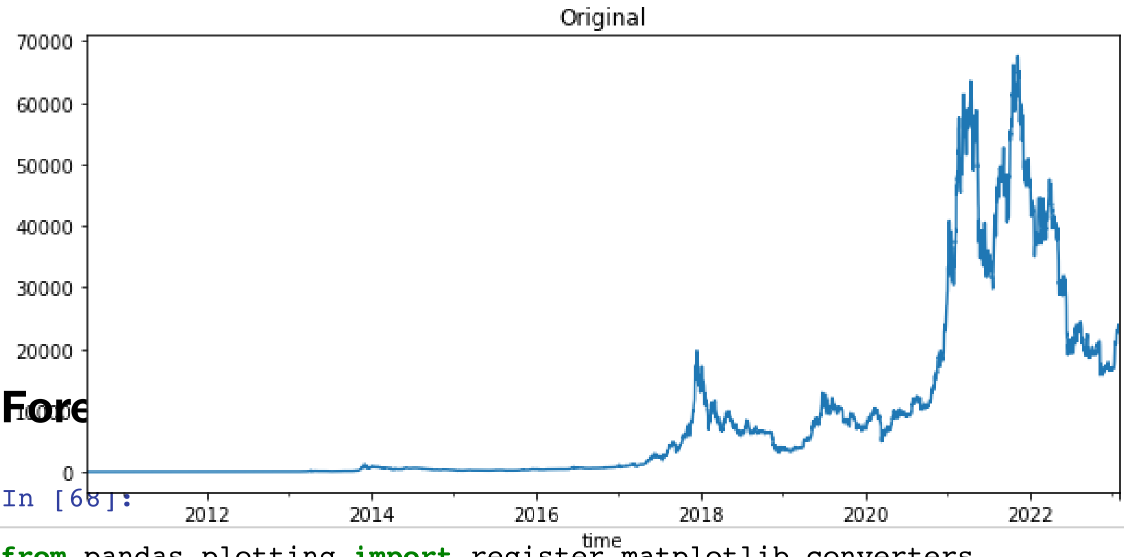
In [26]:

```
def stl_model(df, freq):
    """ Build the STL model for time series decomposition

    Args:
        df (dataframe): the dataframe containing the time series data
        freq (int): the frequency of the time series data, examples: 7 for weekl
    """
    # Decompose the time series into trend, seasonal, and residual components
    # decomposition = STL(df, period=freq).fit()
    decomposition = STL(df.squeeze(), period=freq).fit()
    trend = decomposition.trend
    seasonal = decomposition.seasonal
    residual = decomposition.resid

    # Plot the trend, seasonal, and residual components
    fig, axes = plt.subplots(3, 1, figsize=(10, 10))
    trend.plot(ax=axes[0])
    axes[0].set_title('Trend')
    seasonal.plot(ax=axes[1])
    axes[1].set_title('Seasonal')
    residual.plot(ax=axes[2])
    axes[2].set_title('Residual')
    plt.show()

    # Plot the original and reconstructed time series
    df_reconstructed = trend + seasonal + residual
    fig, axes = plt.subplots(2, 1, figsize=(10, 10))
    df.plot(ax=axes[0])
    axes[0].set_title('Original')
    df_reconstructed.plot(ax=axes[1])
    axes[1].set_title('Reconstructed')
    plt.show()

    return df_reconstructed
```

In [27]:

```
stl_model = stl_model(df['Price'], 365)
```

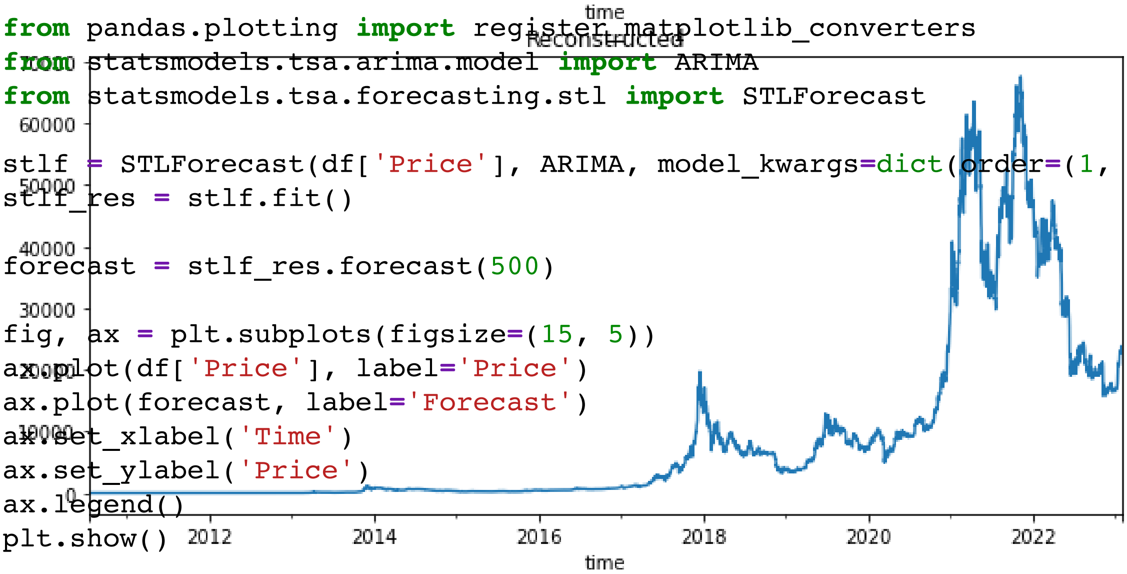




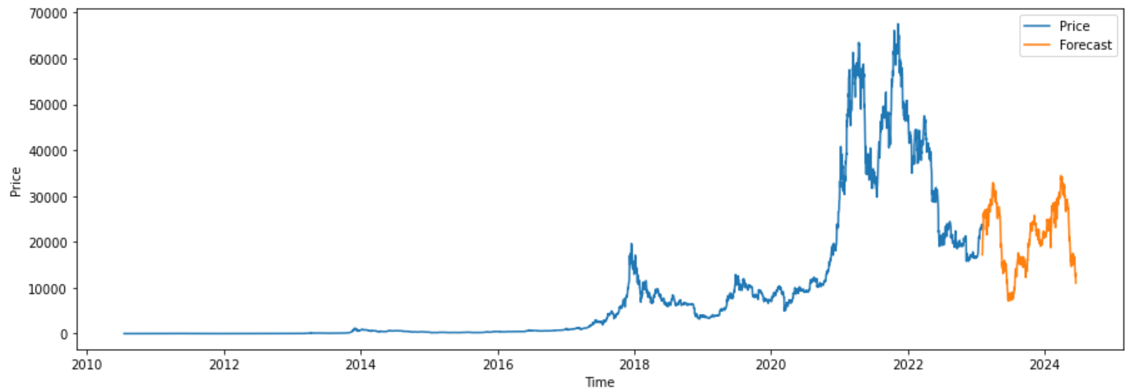
Fore

In [66]:

```
from pandas.plotting import register_matplotlib_converters
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.forecasting.stl import STLForecast

stlf = STLForecast(df['Price'], ARIMA, model_kwargs=dict(order=(1, 1, 0)), trend=
stlf_res = stlf.fit()

forecast = stlf_res.forecast(500)

fig, ax = plt.subplots(figsize=(15, 5))
ax.plot(df['Price'], label='Price')
ax.plot(forecast, label='Forecast')
ax.set_xlabel('Time')
ax.set_ylabel('Price')
ax.legend()
plt.show()
```




```
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
```



The STLForecast function is a time series forecasting method that combines the Seasonal-Trend decomposition using Loess (STL) with an ARIMA (Autoregressive Integrated Moving Average) model. The STL method decomposes the time series into three components: a seasonal component, a trend component, and a remainder component. The ARIMA model is then applied to the remainder component to capture any remaining autocorrelation in the data.

Let's break down the function arguments:

- `df[col]` : This is the time series data that we want to forecast. The col argument specifies which column of the data frame df to use.
- `ARIMA` : This argument specifies that we want to use an ARIMA model for the time series forecasting.
- `model_kwargs=dict(order=(1, 1, 0), trend="t")` : This is a dictionary of keyword arguments that are passed to the ARIMA model. In this case, we are specifying that the ARIMA model should have an order of (1, 1, 0) which means that the model has one autoregressive term, one differencing term, and no moving average terms. We are also specifying that the model should include a linear trend.
- `period=365` : This argument specifies the length of the seasonal cycle in the data. In this case, we are assuming that the data has a yearly seasonal pattern with a period of 365 days.

Overall, the STLForecast function combines the STL decomposition method with an ARIMA model to capture both the seasonal and non-seasonal components of the time series. This allows for accurate forecasting of future values, taking into account both short-term and long-term trends in the data.

In [64]:

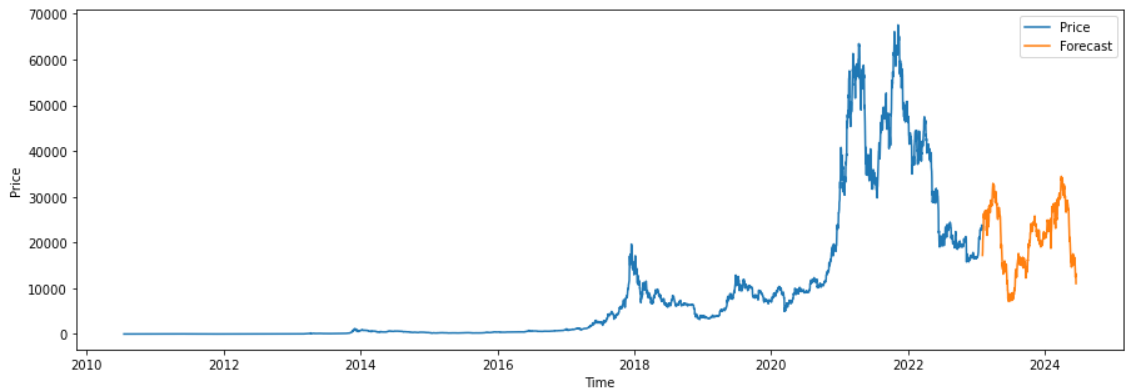
```
print(stlf_res.summary())
```

```
                    STL Decomposition and SARIMAX Results
==============================================================
===============
Dep. Variable:                        y   No. Observations:
4582
Model:                  ARIMA(1, 1, 0)   Log Likelihood
-35182.447
Date:                 Tue, 28 Feb 2023   AIC
70370.894
Time:                         17:36:58   BIC
70390.183
Sample:                     07-18-2010   HQIC
70377.684
                          - 02-01-2023
Covariance Type:                   opg
==============================================================
===============
                 coef    std err          z      P>|z|      [0.
025      0.975]
```

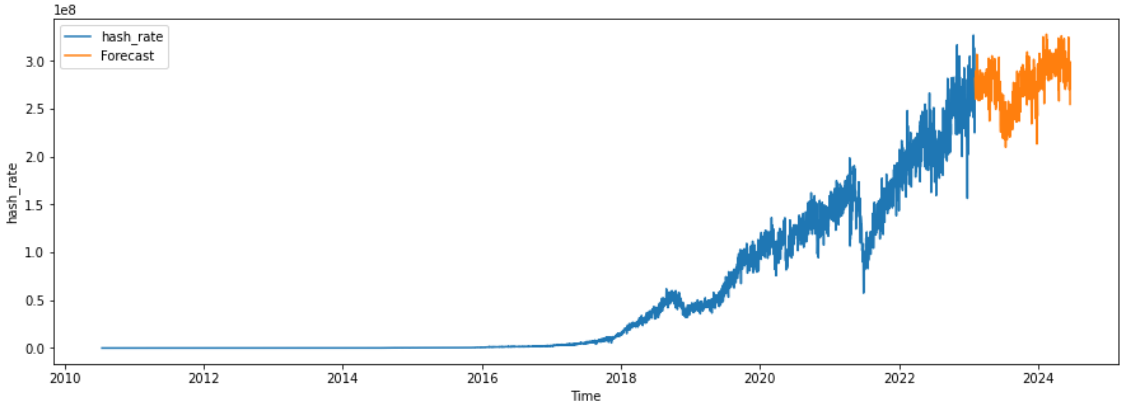
In [70]:

```
from pandas.plotting import register_matplotlib_converters
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.forecasting.stl import STLForecast

for col in df.columns:
    stlf = STLForecast(df[col], ARIMA, model_kwargs=dict(order=(1, 1, 0), trend=
    stlf_res = stlf.fit()

    forecast = stlf_res.forecast(500)

    fig, ax = plt.subplots(figsize=(15, 5))
    ax.plot(df[col], label=col)
    ax.plot(forecast, label='Forecast')
    ax.set_xlabel('Time')
    ax.set_ylabel(col)
    ax.legend()
    plt.show()
```

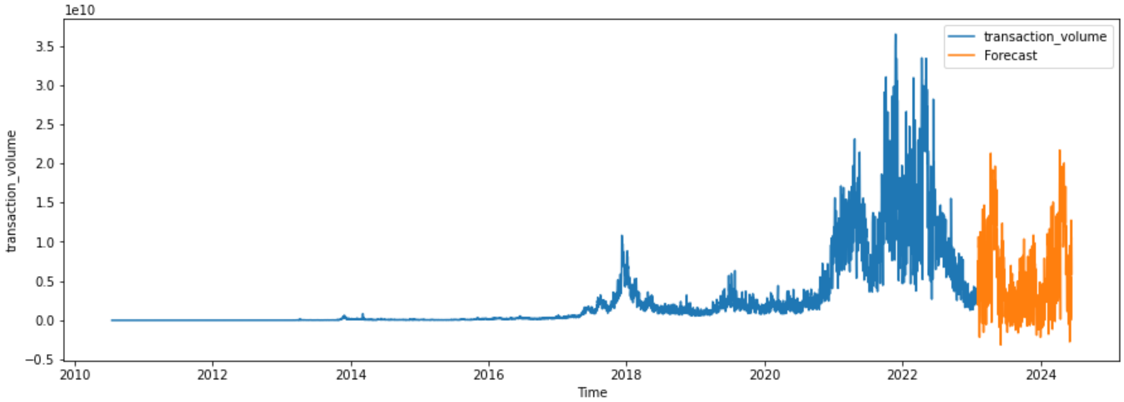
```
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
```



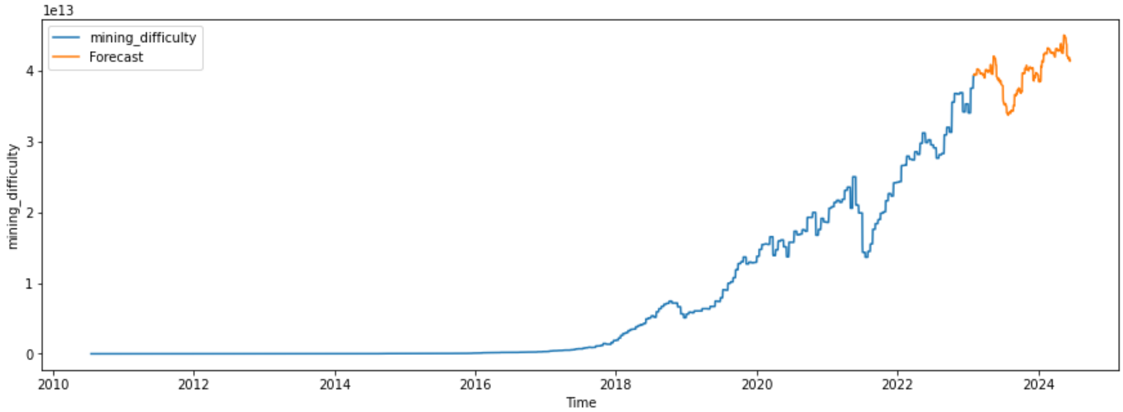

```
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
```



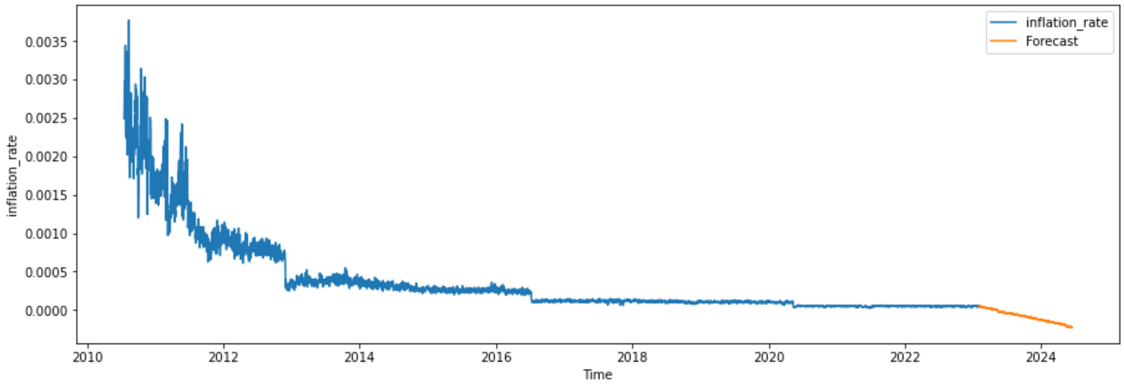

```
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
```



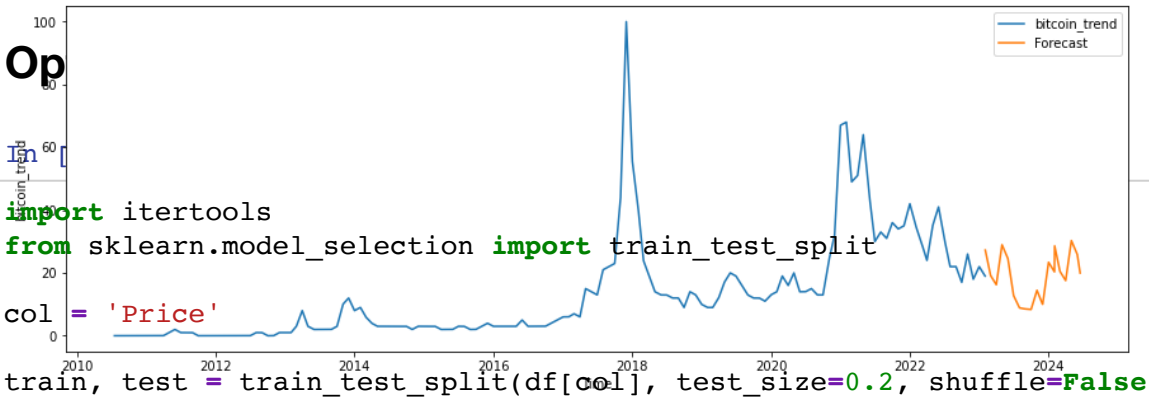

```
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
```




```
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/base/model.py:566: ConvergenceWarning: Maximum Likelihood optimiz
ation failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```




```
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
/Users/macbook/opt/anaconda3/lib/python3.9/site-packages/statsmodel
s/tsa/base/tsa_model.py:524: ValueWarning: No frequency information
was provided, so inferred frequency D will be used.
  warnings.warn('No frequency information was'
```



Op

In [

```
import itertools
from sklearn.model_selection import train_test_split

col = 'Price'
train, test = train_test_split(df[col], test_size=0.2, shuffle=False)

# we specify the range of hyperparameters you want to search over. In this case,

p_range = range(0, 3)
q_range = range(0, 3)
r_range = range(0, 3)

param_combinations = list(itertools.product(p_range, q_range, r_range))
```

In [82]:

```
def evaluate_stlf_model(train_data, test_data, model_kwargs, period):
    model = STLForecast(train_data, ARIMA, model_kwargs=model_kwargs, period=per
    res = model.fit()
    forecast = res.forecast(len(test_data))
    error = test_data - forecast
    rmse = np.sqrt(np.mean(error**2))
    return rmse
```

In [83]:

```
params
```

Out[83]:

```
(0, 0, 0)
```

In [84]:

```
best_rmse = float('inf')
best_params = None

for params in param_combinations:
    model_kwargs = dict(order=params, trend="t")
    rmse = evaluate_stlf_model(train, test, model_kwargs, period=365)
    if rmse < best_rmse:
        best_rmse = rmse
        best_params = params
```