Bitcoin

August 21, 2023

1 Bitcoin Data Analysis

1.1 Time Series Forcasting

[106]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import yfinance as yf
from datetime import datetime
from statsmodels import tsa
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.stattools import kpss
```

1.2 Overview

we are doing predictions for Bitcoin's future price, we are performing a 30-day forecasting period for the bitcoins price based on the past year's price fluctuations. each time that this file loads it will automatically download the latest Bitcoin data from Yahoo financial API which we can easily access with "yfinance" package in python. keeping that in mind that based on the changes in the Bitcoin price our models may or may not need parameter adjustments to get the most accurate results.

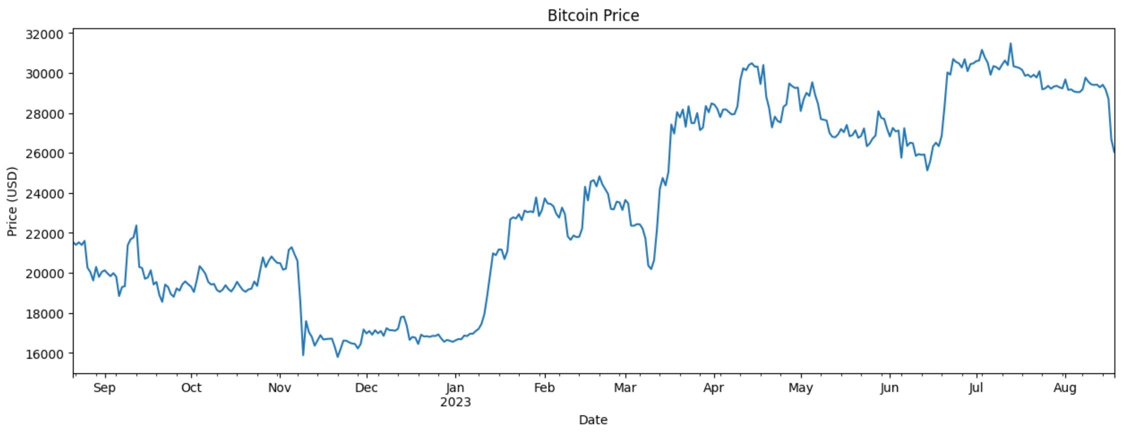
[107]:
```
# KPSS test Function from statsmodels package
def kpss_test(timeseries):
    print("Results of KPSS Test:")
    kpsstest = kpss(timeseries, regression="c", nlags="auto")
    kpss_output = pd.Series(
        kpsstest[0:3], index=["Test Statistic", "p-value", "Lags Used"]
    )
    for key, value in kpsstest[3].items():
        kpss_output["Critical Value (%s)" % key] = value
    print(kpss_output)
```

[108]:
```
bitcoin_data
```

[108]:
```
                   Price
Date
2022-08-21  21534.121094
2022-08-22  21398.908203
2022-08-23  21528.087891
2022-08-24  21395.019531
2022-08-25  21600.904297
…                    …
2023-08-14  29408.443359
2023-08-15  29170.347656
2023-08-16  28701.779297
2023-08-17  26664.550781
2023-08-18  26049.556641

[363 rows x 1 columns]
```

We begin with importing the data from Yahoo Finance and plotting the price of Bitcoin in the past year which started at around 21000 and now it is about 26000 with a gradual increase over the year. the data is not seasonal but it looks very cyclical and there is a trend.

[109]:
```python
from datetime import timedelta

bitcoin_data = yf.download('BTC-USD',
                           start=datetime.now()- timedelta(days=365),
                           end=datetime.now(),
                           progress=False)
bitcoin_data.reset_index(inplace=True)

bitcoin_data = bitcoin_data[['Date', 'Close']]
bitcoin_data.columns = ['Date', 'Price']
bitcoin_data.set_index('Date', inplace=True)

bitcoin_data['Price'].plot(figsize=(15, 5))
plt.title('Bitcoin Price')
plt.ylabel('Price (USD)')
plt.show()
```

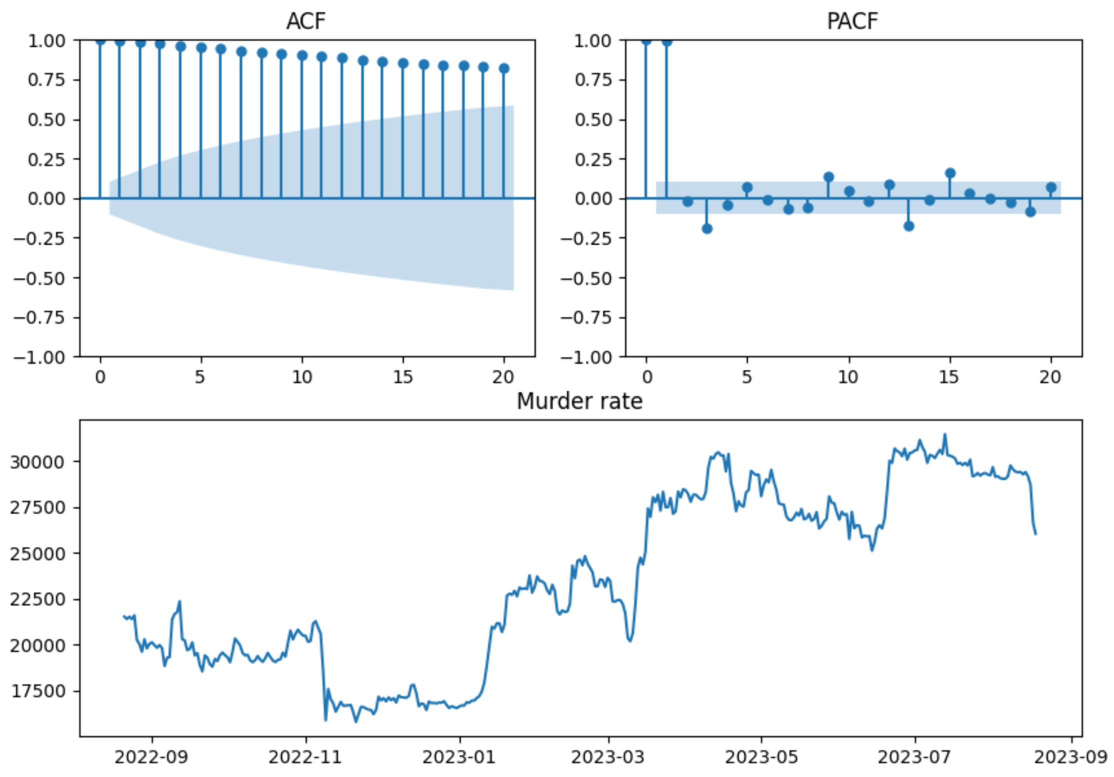


1.3 ARIMA Model

to check the data for cyclicality and the trend, we plot ACF and PACF to analyze our data visually we can see that the ACF plot shows that the data is highly cyclical as we assumed earlier. Hence it means that we need to do differencing on our data to make it stationary for building an ARIMA model.

[110]:
```python
plt.figure(figsize=(10,7))
ax1 = plt.subplot(212)
ax1.plot(bitcoin_data)
ax1.set_title('Murder rate')
ax2 = plt.subplot(221)
ax2=plot_acf(bitcoin_data, lags=20, ax=plt.gca(), title='ACF')
ax3 = plt.subplot(222)
ax3=plot_pacf(bitcoin_data, lags=20, ax=plt.gca(), title='PACF')
```

```
/opt/homebrew/lib/python3.11/site-packages/statsmodels/graphics/tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```



we can say by looking at the data that data is not stationary but we run the KPSS test to see for sure that the test statistic is higher than critical value which we reject the H_0.

[111]:
```
kpss_test(bitcoin_data)
```

```
Results of KPSS Test:
Test Statistic             2.550118
p-value                    0.010000
Lags Used                 11.000000
Critical Value (10%)       0.347000
Critical Value (5%)        0.463000
Critical Value (2.5%)      0.574000
Critical Value (1%)        0.739000
dtype: float64
```

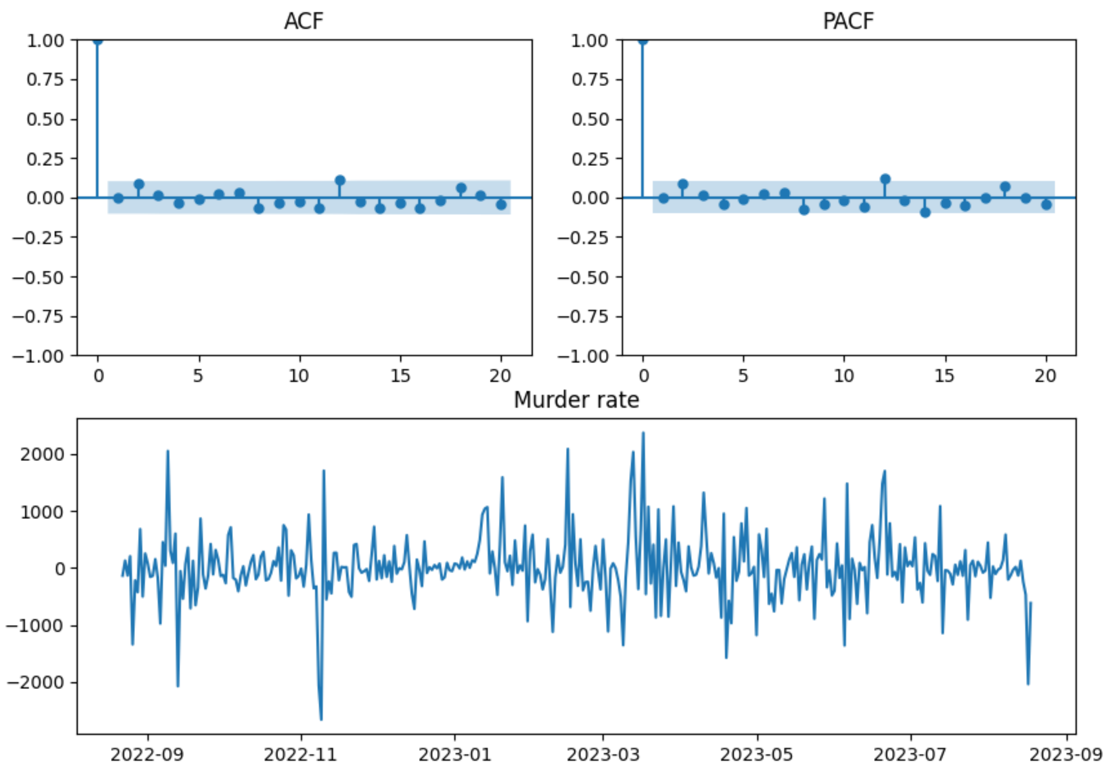
```
/opt/homebrew/lib/python3.11/site-packages/statsmodels/tsa/stattools.py:2018:
InterpolationWarning: The test statistic is outside of the range of p-values
available in the
look-up table. The actual p-value is smaller than the p-value returned.

  warnings.warn(
```

so the measure we take next is to take differencing of our current data to make it stationary for the ARIMA model. and the we plot ACF and PACF to see the correlations between lags. and we can see the data is stationary and there is no correlation between the variables.

```
[112]: bitcoin_data_diff=bitcoin_data.diff().dropna()
```

```
[113]: plt.figure(figsize=(10,7))
       ax1 = plt.subplot(212)
       ax1.plot(bitcoin_data_diff)
       ax1.set_title('Murder rate')
       ax2 = plt.subplot(221)
       ax2=plot_acf(bitcoin_data_diff, lags=20, ax=plt.gca(), title='ACF')
       ax3 = plt.subplot(222)
       ax3=plot_pacf(bitcoin_data_diff, lags=20, ax=plt.gca(), title='PACF')
```

```
/opt/homebrew/lib/python3.11/site-packages/statsmodels/graphics/tsaplots.py:348:
FutureWarning: The default method 'yw' can produce PACF values outside of the
[-1,1] interval. After 0.13, the default will change tounadjusted Yule-Walker
('ywm'). You can use this method now by setting method='ywm'.
  warnings.warn(
```



we again will perform KPSS test to the test statistic which is lower than 10% critical value and the p-value is high so we retain the H_0 and accept that data is staitionary.

```
[114]: kpss_test(bitcoin_data_diff)
```

```
Results of KPSS Test:
Test Statistic           0.104941
p-value                  0.100000
Lags Used                4.000000
Critical Value (10%)     0.347000
Critical Value (5%)      0.463000
Critical Value (2.5%)    0.574000
Critical Value (1%)      0.739000
dtype: float64
```

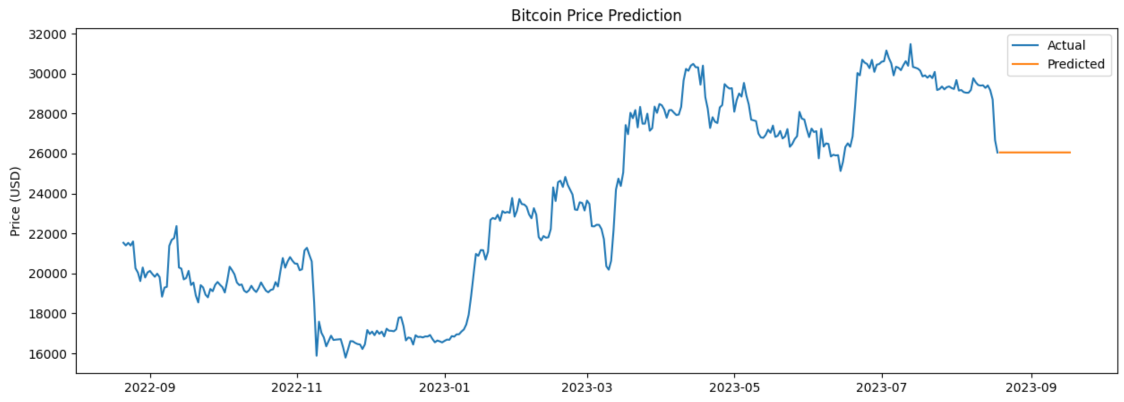
```
/opt/homebrew/lib/python3.11/site-packages/statsmodels/tsa/stattools.py:2022:
InterpolationWarning: The test statistic is outside of the range of p-values
available in the
look-up table. The actual p-value is greater than the p-value returned.

  warnings.warn(
```

next, we want to do the ARIMA Model, and because there are no correlations between values using both ACF and PACF after first differencing choose P D Q as respectively 1,1,0 and build the model. and do a forecasting of 30 dates after today. we print the actual forecasted values here to see and also make a chart to visualize the forecasted values.

[115]:
```
# Perform time series analysis using ARIMA model
model = ARIMA(bitcoin_data,
              order=(1, 1, 0))
model_fit = model.fit()

# Make future predictions
future_predictions = pd.DataFrame(model_fit.forecast(steps=30))
print(future_predictions)

# Plot the future predictions
plt.figure(figsize=(15,5))
plt.plot(bitcoin_data, label='Actual')
#plt.plot(model_fit.fittedvalues)
plt.plot(future_predictions, label='Predicted')
plt.title('Bitcoin Price Prediction')
plt.ylabel('Price (USD)')
plt.legend()
plt.show()
```

```
            predicted_mean
2023-08-19    26055.138663
2023-08-20    26055.087998
2023-08-21    26055.088458
2023-08-22    26055.088453
2023-08-23    26055.088454
2023-08-24    26055.088454
2023-08-25    26055.088454
```

```
2023-08-26    26055.088454
2023-08-27    26055.088454
2023-08-28    26055.088454
2023-08-29    26055.088454
2023-08-30    26055.088454
2023-08-31    26055.088454
2023-09-01    26055.088454
2023-09-02    26055.088454
2023-09-03    26055.088454
2023-09-04    26055.088454
2023-09-05    26055.088454
2023-09-06    26055.088454
2023-09-07    26055.088454
2023-09-08    26055.088454
2023-09-09    26055.088454
2023-09-10    26055.088454
2023-09-11    26055.088454
2023-09-12    26055.088454
2023-09-13    26055.088454
2023-09-14    26055.088454
2023-09-15    26055.088454
2023-09-16    26055.088454
2023-09-17    26055.088454

/opt/homebrew/lib/python3.11/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/opt/homebrew/lib/python3.11/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
/opt/homebrew/lib/python3.11/site-
packages/statsmodels/tsa/base/tsa_model.py:471: ValueWarning: No frequency
information was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```



and for the model summary we have sigma 2 as the coefficient which is significant and the auto regression lag 1 which is not significant.

[116]:
```
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  Price   No. Observations:                  363
Model:                 ARIMA(1, 1, 0)   Log Likelihood               -2817.965
Date:                Mon, 21 Aug 2023   AIC                           5639.931
Time:                        14:25:05   BIC                           5647.714
Sample:                    08-21-2022   HQIC                          5643.025
                         - 08-18-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0091      0.035     -0.257      0.797      -0.078       0.060
sigma2       3.39e+05   1.57e+04     21.594      0.000    3.08e+05     3.7e+05
================================================================================
===
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):
232.96
Prob(Q):                              0.94   Prob(JB):
0.00
Heteroskedasticity (H):               0.92   Skew:
0.10
Prob(H) (two-sided):                  0.63   Kurtosis:
6.92
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

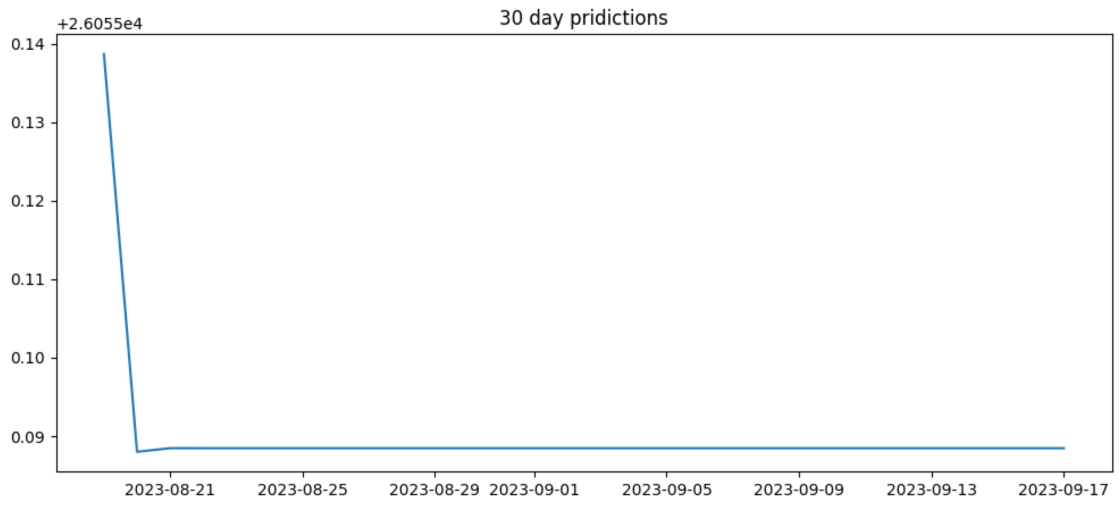
the value of root mean squer error is also as follows:

[117]:
```
print("RMSE for ARIMA Model: ", np.sqrt((model_fit.resid**2).mean()))
```

```
RMSE for ARIMA Model:  1270.666433436001
```

[118]:
```
plt.figure(figsize=(12,5))
plt.plot(future_predictions)
plt.title('30 day pridictions')
```

[118]:
```
Text(0.5, 1.0, '30 day pridictions')
```



now we want to check and see which model parameters does Auto Arima model from the package "pmdarima" chooses for an optimal model.

[119]:
```
from pmdarima.arima import auto_arima

AAmodel=auto_arima(bitcoin_data,
                   start_P=0,
                   start_q=0,
                   max_d=2)
print(AAmodel.arparams)
```

```
<bound method ARIMA.arparams of ARIMA(order=(0, 1, 0), scoring_args={},
suppress_warnings=True,
      with_intercept=False)>
```

The model auto arima chose is the order (0,1,0) which did not consider lag 1 autoregression as significant which was true and we can see the value of AIC is slightly lower than our model. The value of RMSE is also the same as our model.

[120]:
```
print(AAmodel.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  363
Model:               SARIMAX(0, 1, 0)   Log Likelihood               -2817.984
Date:                Mon, 21 Aug 2023   AIC                           5637.968
Time:                        14:25:06   BIC                           5641.860
Sample:                    08-21-2022   HQIC                          5639.515
                         - 08-18-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
```

```
------------------------------------------------------------------------------
sigma2      3.372e+05   1.46e+04     23.131      0.000    3.09e+05    3.66e+05
================================================================================
===
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):
227.04
Prob(Q):                              0.91   Prob(JB):
0.00
Heteroskedasticity (H):               0.92   Skew:
0.10
Prob(H) (two-sided):                  0.64   Kurtosis:
6.87
================================================================================
===

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-
step).
```

[121]:
```python
print("RMSE for ARIMA Model: ", np.sqrt((AAmodel.resid()**2).mean()))
```

```
RMSE for ARIMA Model:  1270.6800427498222
```

1.4 Test for Seasonality

We already know that our there is no seasonality in Bitcoin's price change in the past year but we are seeing a repetitive pattern in our data which brings us to the point that we wanted to try and see what is the pattern in our data. So we performed an exponential smoothing model using additive trend and seasonality and let the algorithm estimate the parameters, doing seasonal periods of less than 150 days. we stored the RMSE for each model for later comparison and see which seasonal period gives us the lowest RMSE in term of seasonality. and we plot and print the result to see the lowest value which is 116.

[122]:
```python
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.seasonal import STL

rmse=[]
aic=[]
for k in range(3,150):
    modelexp=ExponentialSmoothing(bitcoin_data,
                                  trend='add',
                                  seasonal='add',
                                  damped_trend=True,
                                  initialization_method='estimated',
                                  seasonal_periods=k).fit()

    rmse.append(np.sqrt((modelexp.resid**2).mean()))
```