A Univariate Time Series Analysis and Forecasting on FRED Unemployment Data with Python

A little about Time Series

- Time series analysis comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data.
- Time series forecasting is the use of a model to predict future values based on previously observed values.

In [1]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import numpy as np
import pandas as pd
from sklearn.metrics import f1_score, precision_score, recall_score, confusion_matrix, mean_squared_error, mean_a
bsolute_error, explained_variance_score
```

In [2]:

```
data = pd.read_excel(r"C:\Users\mdmoh\Desktop\fredgraph (2).xls")
```

In [3]:

```
data.head()
```

Out[3]:

	observation_date	BTC
0	2014-09-17	465.864014
1	2014-09-18	456.859985
2	2014-09-19	424.102997
3	2014-09-20	394.673004
4	2014-09-21	408.084991

Data Preprocessing

In [4]:

```
data1 = data.iloc[0:]
data1.head(10)
```

Out[4]:

	observation_date	BTC
0	2014-09-17	465.864014
1	2014-09-18	456.859985
2	2014-09-19	424.102997
3	2014-09-20	394.673004
4	2014-09-21	408.084991
5	2014-09-22	399.100006
6	2014-09-23	402.092010
7	2014-09-24	435.751007
8	2014-09-25	423.156006
9	2014-09-26	411.428986

In [5]:

```
data1.isna()
```

Out[5]:

	observation_date	BTC
0	False	False
1	False	False
2	False	False
3	False	False
4	False	False
...	...	...
2287	False	False
2288	False	False
2289	False	False
2290	False	False
2291	False	False

2292 rows × 2 columns

In [6]:

```
data1.tail(10)
```

Out[6]:

	observation_date	BTC
2282	2020-12-16	19418.818359
2283	2020-12-17	21308.351563
2284	2020-12-18	22806.796875
2285	2020-12-19	23132.865234
2286	2020-12-20	23861.765625
2287	2020-12-21	23474.455078
2288	2020-12-22	22794.039063
2289	2020-12-23	23781.974609
2290	2020-12-24	23240.203125
2291	2020-12-25	24445.125000

In [7]:

```
data1.isna().sum()
```

Out[7]:

```
observation_date    0
BTC                 0
dtype: int64
```

In [8]:

```
data1.shape
```

Out[8]:

```
(2292, 2)
```

In [9]:

```
data1.columns
```

Out[9]:

```
Index(['observation_date', 'BTC '], dtype='object')
```

In [10]:

```
data1.shape
```

Out[10]:

```
(2292, 2)
```

In [11]:

```
data1['obs_date'] = [d.date() for d in data1['observation_date']]
data1['obs_time'] = [d.time() for d in data1['observation_date']]
```

```
C:\Users\mdmoh\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
ng.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
C:\Users\mdmoh\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexi
ng.html#returning-a-view-versus-a-copy
```

In [12]:

```
data1
```

Out[12]:

	observation_date	BTC	obs_date	obs_time
0	2014-09-17	465.864014	2014-09-17	00:00:00
1	2014-09-18	456.859985	2014-09-18	00:00:00
2	2014-09-19	424.102997	2014-09-19	00:00:00
3	2014-09-20	394.673004	2014-09-20	00:00:00
4	2014-09-21	408.084991	2014-09-21	00:00:00
...	...	...	...	...
2287	2020-12-21	23474.455078	2020-12-21	00:00:00
2288	2020-12-22	22794.039063	2020-12-22	00:00:00
2289	2020-12-23	23781.974609	2020-12-23	00:00:00
2290	2020-12-24	23240.203125	2020-12-24	00:00:00
2291	2020-12-25	24445.125000	2020-12-25	00:00:00

2292 rows × 4 columns

In [13]:

```
data5 = data1.drop(['observation_date'], axis=1)
data5
```

Out[13]:

	BTC	obs_date	obs_time
0	465.864014	2014-09-17	00:00:00
1	456.859985	2014-09-18	00:00:00
2	424.102997	2014-09-19	00:00:00
3	394.673004	2014-09-20	00:00:00
4	408.084991	2014-09-21	00:00:00
...	...	...	...
2287	23474.455078	2020-12-21	00:00:00
2288	22794.039063	2020-12-22	00:00:00
2289	23781.974609	2020-12-23	00:00:00
2290	23240.203125	2020-12-24	00:00:00
2291	24445.125000	2020-12-25	00:00:00

2292 rows × 3 columns

In [14]:

```
data5['obs_time'].nunique()
```

Out[14]:

1

In [15]:

```
data6 = data5.drop(['obs_time'], axis=1)
data6
```

Out[15]:

	BTC	obs_date
0	465.864014	2014-09-17
1	456.859985	2014-09-18
2	424.102997	2014-09-19
3	394.673004	2014-09-20
4	408.084991	2014-09-21
...	...	...
2287	23474.455078	2020-12-21
2288	22794.039063	2020-12-22
2289	23781.974609	2020-12-23
2290	23240.203125	2020-12-24
2291	24445.125000	2020-12-25

2292 rows × 2 columns

In [16]:

```
data6.shape
```

Out[16]:

(2292, 2)

In [17]:

```
data6.describe()
```

Out[17]:

	BTC
count	2292.000000
mean	4865.497410
std	4683.042424
min	176.897003
25%	444.617989
50%	3880.410034
75%	8349.262207
max	24445.125000

In [18]:

```
data6.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2292 entries, 0 to 2291
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   BTC       2292 non-null   float64
 1   obs_date  2292 non-null   object 
dtypes: float64(1), object(1)
memory usage: 35.9+ KB
```

In [19]:

```
file_loc = r"C:\Users\mdmoh\Desktop\fredgraph (2).xls"
data7 = pd.read_excel(file_loc, index_col=None, na_values=['NA'], usecols = "L")
print(data7)
```

```
Empty DataFrame
Columns: []
Index: []
```

In [26]:

```
plt.figure(figsize=(12,8))
plt.scatter(data6['obs_date'], data6['BTC '])
plt.xlabel('Dates')
plt.ylabel('Federal Funds')
```

Out[26]:

```
Text(0, 0.5, 'Federal Funds')
```

In [27]:

```
data6 = data6.reset_index(drop=True)
data6.head(2)
```

Out[27]:

	BTC	obs_date
0	465.864014	2014-09-17
1	456.859985	2014-09-18

In [34]:

```
# average the unemployment rate for each month
# use start of each month as the timestamp
y = data13['BTC '].resample('MS').mean()
y['2019':]
```

Out[34]:

```
observation_date
2019-01-01     3709.705645
2019-02-01     3697.178327
2019-03-01     3967.740400
2019-04-01     5136.813314
2019-05-01     7205.208024
2019-06-01     9339.480322
2019-07-01    10691.706055
2019-08-01    10657.745621
2019-09-01     9858.141813
2019-10-01     8382.432129
2019-11-01     8427.103516
2019-12-01     7296.351625
2020-01-01     8318.949597
2020-02-01     9656.215113
2020-03-01     6943.507009
2020-04-01     7150.611328
2020-05-01     9237.761530
2020-06-01     9499.797005
2020-07-01     9519.383852
2020-08-01    11639.097215
2020-09-01    10689.070540
2020-10-01    11793.169449
2020-11-01    16450.121647
2020-12-01    20485.105391
Freq: MS, Name: BTC , dtype: float64
```

Time-series decompositon

It allows us to decompose our time series into three distinct components

- Trend
- Seasonality
- Noise

Time series forecasting with ARIMA model

A Little about Arima Model

- ARIMA stands for Autoregressive Integreted Moving Average
- ARIMA models are denoted with the notation ARIMA(p, d, q)
- These three parameters account for seasonality, trend, and noise in data

In [36]:

```
import itertools

# set the typical ranges for p, d, q
p = d = q = range(0, 2)

#take all possible combination for p, d and q
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]))
```

```
Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

In [37]:

```
# Using Grid Search find the optimal set of parameters that yields the best performance
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y, order = param, seasonal_order = param_seasonal, enforce_stationary
= False,enforce_invertibility=False)
            result = mod.fit()
            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, result.aic))
        except:
            continue
```

```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:1557.9960862922303
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:1533.6637609034035
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:1266.887234956689
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:1265.4249794386938
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:1510.9572008425228

C:\Users\mdmoh\Anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:994: UserWarning: N
on-stationary starting seasonal autoregressive Using zeros as starting parameters.
  warn('Non-stationary starting seasonal autoregressive'

ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:1519.7856103502681
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:1263.8836525896018
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:1266.1955249001371
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:1509.7541400836258
ARIMA(0, 0, 1)x(0, 0, 1, 12)12 - AIC:1449.602119461618
ARIMA(0, 0, 1)x(0, 1, 0, 12)12 - AIC:1209.4635812983247
ARIMA(0, 0, 1)x(0, 1, 1, 12)12 - AIC:1205.9457365704043
ARIMA(0, 0, 1)x(1, 0, 0, 12)12 - AIC:1506.1888957088927

C:\Users\mdmoh\Anaconda3\lib\site-packages\statsmodels\base\model.py:568: ConvergenceWarning: Maximu
m Likelihood optimization failed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)

ARIMA(0, 0, 1)x(1, 0, 1, 12)12 - AIC:1441.5489769682808
ARIMA(0, 0, 1)x(1, 1, 0, 12)12 - AIC:1206.4385661960937
ARIMA(0, 0, 1)x(1, 1, 1, 12)12 - AIC:1207.6455064668683
ARIMA(0, 1, 0)x(0, 0, 0, 12)12 - AIC:1311.7571978556
ARIMA(0, 1, 0)x(0, 0, 1, 12)12 - AIC:1312.0504150262987
ARIMA(0, 1, 0)x(0, 1, 0, 12)12 - AIC:1152.8059263437844
ARIMA(0, 1, 0)x(0, 1, 1, 12)12 - AIC:1122.246643090082
ARIMA(0, 1, 0)x(1, 0, 0, 12)12 - AIC:1312.5265079073017
ARIMA(0, 1, 0)x(1, 0, 1, 12)12 - AIC:1313.997608738376
ARIMA(0, 1, 0)x(1, 1, 0, 12)12 - AIC:1141.7954822611669
ARIMA(0, 1, 0)x(1, 1, 1, 12)12 - AIC:1121.8211466818905
ARIMA(0, 1, 1)x(0, 0, 0, 12)12 - AIC:1305.7952123525272
ARIMA(0, 1, 1)x(0, 0, 1, 12)12 - AIC:1306.4866765412564
ARIMA(0, 1, 1)x(0, 1, 0, 12)12 - AIC:1148.0311426798253
ARIMA(0, 1, 1)x(0, 1, 1, 12)12 - AIC:1119.1407484855888
ARIMA(0, 1, 1)x(1, 0, 0, 12)12 - AIC:1306.9539017538123
ARIMA(0, 1, 1)x(1, 0, 1, 12)12 - AIC:1308.248157067773
ARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:1138.2013654227878
ARIMA(0, 1, 1)x(1, 1, 1, 12)12 - AIC:1119.689041303396
ARIMA(1, 0, 0)x(0, 0, 0, 12)12 - AIC:1335.3318283362435

C:\Users\mdmoh\Anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:963: UserWarning: N
on-stationary starting autoregressive parameters found. Using zeros as starting parameters.
  warn('Non-stationary starting autoregressive parameters'
```

```
ARIMA(1, 0, 0)x(0, 0, 1, 12)12 - AIC:1335.5051810160805
ARIMA(1, 0, 0)x(0, 1, 0, 12)12 - AIC:1171.4265214074812
ARIMA(1, 0, 0)x(0, 1, 1, 12)12 - AIC:1142.3043380370511
ARIMA(1, 0, 0)x(1, 0, 0, 12)12 - AIC:1336.0337701544781
ARIMA(1, 0, 0)x(1, 0, 1, 12)12 - AIC:1337.4454311570146
ARIMA(1, 0, 0)x(1, 1, 0, 12)12 - AIC:1161.546220086455
ARIMA(1, 0, 0)x(1, 1, 1, 12)12 - AIC:1142.2998338941393
ARIMA(1, 0, 1)x(0, 0, 0, 12)12 - AIC:1329.1982214526279
ARIMA(1, 0, 1)x(0, 0, 1, 12)12 - AIC:1329.817986421741
ARIMA(1, 0, 1)x(0, 1, 0, 12)12 - AIC:1165.2986329098271
ARIMA(1, 0, 1)x(0, 1, 1, 12)12 - AIC:1139.0916409920499
ARIMA(1, 0, 1)x(1, 0, 0, 12)12 - AIC:1330.3186025594432
ARIMA(1, 0, 1)x(1, 0, 1, 12)12 - AIC:1331.3913651185176
ARIMA(1, 0, 1)x(1, 1, 0, 12)12 - AIC:1157.095511427168
ARIMA(1, 0, 1)x(1, 1, 1, 12)12 - AIC:1139.9727150346525
ARIMA(1, 1, 0)x(0, 0, 0, 12)12 - AIC:1307.6613159855297
ARIMA(1, 1, 0)x(0, 0, 1, 12)12 - AIC:1308.5075534452187
ARIMA(1, 1, 0)x(0, 1, 0, 12)12 - AIC:1148.5178032248264
ARIMA(1, 1, 0)x(0, 1, 1, 12)12 - AIC:1120.3525784665487
ARIMA(1, 1, 0)x(1, 0, 0, 12)12 - AIC:1308.907401264373
ARIMA(1, 1, 0)x(1, 0, 1, 12)12 - AIC:1310.3060002530246
ARIMA(1, 1, 0)x(1, 1, 0, 12)12 - AIC:1140.1592509149025
ARIMA(1, 1, 0)x(1, 1, 1, 12)12 - AIC:1120.8753327191198
ARIMA(1, 1, 1)x(0, 0, 0, 12)12 - AIC:1307.1400074249495
ARIMA(1, 1, 1)x(0, 0, 1, 12)12 - AIC:1307.5246607718136
ARIMA(1, 1, 1)x(0, 1, 0, 12)12 - AIC:1150.024438096712
ARIMA(1, 1, 1)x(0, 1, 1, 12)12 - AIC:1120.5425975560217
ARIMA(1, 1, 1)x(1, 0, 0, 12)12 - AIC:1308.1308979603502
ARIMA(1, 1, 1)x(1, 0, 1, 12)12 - AIC:1309.3702674210965
ARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:1140.128313210085
ARIMA(1, 1, 1)x(1, 1, 1, 12)12 - AIC:1120.535160902676
```

In [38]:

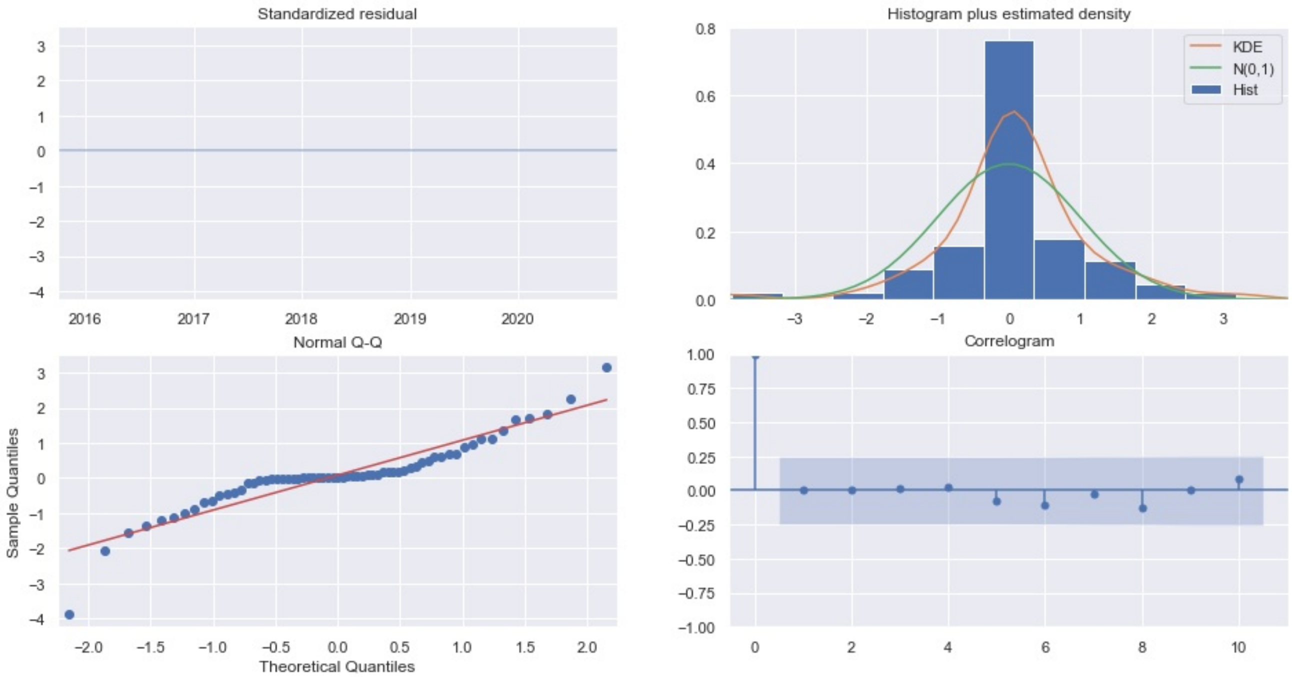
```python
#Fitting the ARIMA model using above optimal combination of p, d, q (optimal means combination at which we got lo
west AIC score)

model = sm.tsa.statespace.SARIMAX(y, order = (0, 1, 1),
                                  seasonal_order = (0, 1, 0, 12)
                                 )
result = model.fit()
print(result.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1          0.3458      0.079      4.393      0.000       0.192       0.500
sigma2      4.498e+06   4.76e+05      9.458      0.000    3.57e+06    5.43e+06
==============================================================================
```

In [39]:

```
#run model diagnostic to investigate any unusual behavior
result.plot_diagnostics(figsize = (16, 8))
plt.show()
```

```
C:\Users\mdmoh\Anaconda3\lib\site-packages\pandas\plotting\_matplotlib\converter.py:256: MatplotlibD
eprecationWarning:
The epoch2num function was deprecated in Matplotlib 3.3 and will be removed two minor releases later
.
  base = dates.epoch2num(dt.asi8 / 1.0e9)
```



It is not perfect, however, our model diagnostics suggests that the model residuals are near normally distributed.

Validating Forecasts

To help us understand the accuracy of our forecasts, we compare predicted rates to real rates of the time series, and we set forecasts to start at 2019–01–01 to the end of the data.

In [40]:

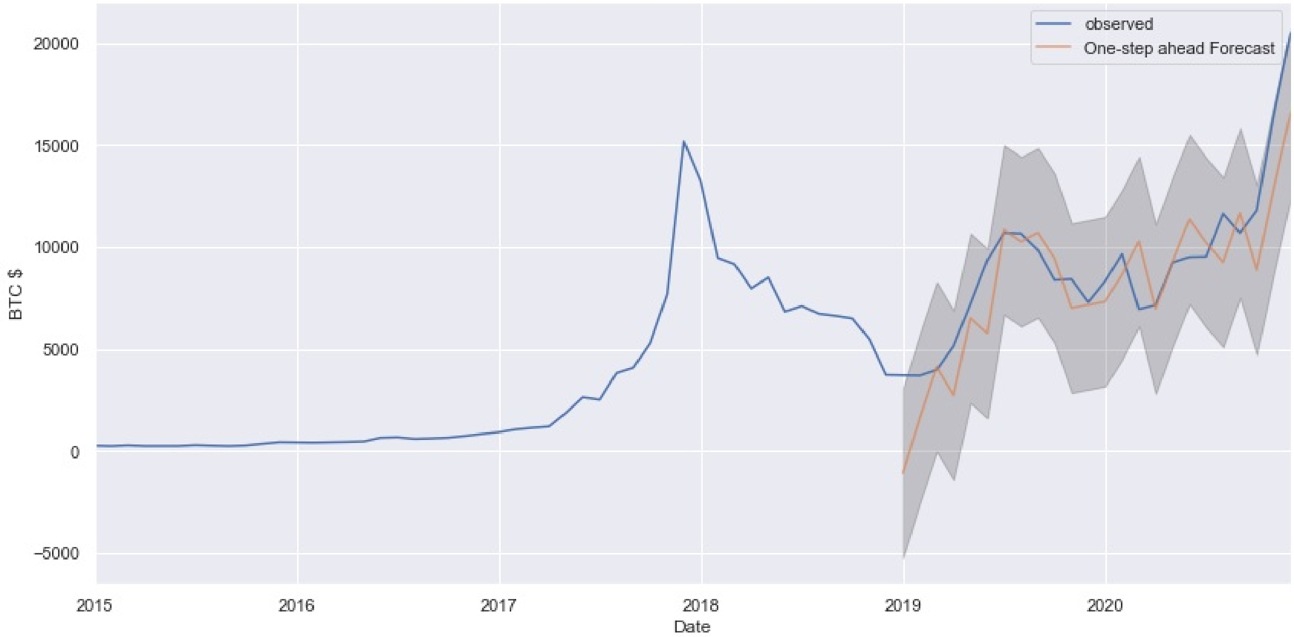
```
prediction = result.get_prediction(start = pd.to_datetime('2019-01-01'), dynamic = False)
prediction_ci = prediction.conf_int()
prediction_ci
```

Out[40]:

	lower BTC	upper BTC
observation_date		
2019-01-01	-5243.439854	3069.939322
2019-02-01	-2538.372130	5775.007046
2019-03-01	-46.366636	8267.012539
2019-04-01	-1431.222549	6882.156627
2019-05-01	2355.503493	10668.882668
2019-06-01	1612.155547	9925.534722
2019-07-01	6689.597304	15002.976480
2019-08-01	6103.902439	14417.281615
2019-09-01	6537.397942	14850.777118
2019-10-01	5283.607212	13596.986388
2019-11-01	2847.569357	11160.948533
2019-12-01	3007.242738	11320.621913
2020-01-01	3168.678441	11482.057617
2020-02-01	4493.277704	12806.656880
2020-03-01	6118.012148	14431.391324
2020-04-01	2804.082198	11117.461374
2020-05-01	5127.956182	13441.335358
2020-06-01	7199.133345	15512.512521
2020-07-01	6053.585686	14366.964862
2020-08-01	5089.848258	13403.227433
2020-09-01	7510.065380	15823.444556
2020-10-01	4718.622919	13032.002094
2020-11-01	8690.041945	17003.421121
2020-12-01	12408.603747	20721.982923

In [41]:

```
#Visualize the forecasting
ax = y['2015':].plot(label = 'observed')
prediction.predicted_mean.plot(ax = ax, label = 'One-step ahead Forecast', alpha = 0.7, figsize = (14, 7))
ax.fill_between(prediction_ci.index, prediction_ci.iloc[:, 0], prediction_ci.iloc[:, 1], color = 'k', alpha = 0.2
)
ax.set_xlabel("Date")
ax.set_ylabel('BTC $')
plt.legend()
plt.show()
```



The line plot is showing the observed values compared to the rolling forecast predictions. Overall, our forecasts align with the true values very well, showing an upward trend starts from the beginning of the year and captured the seasonality toward the end of the year.

Error Analysis

In [42]:

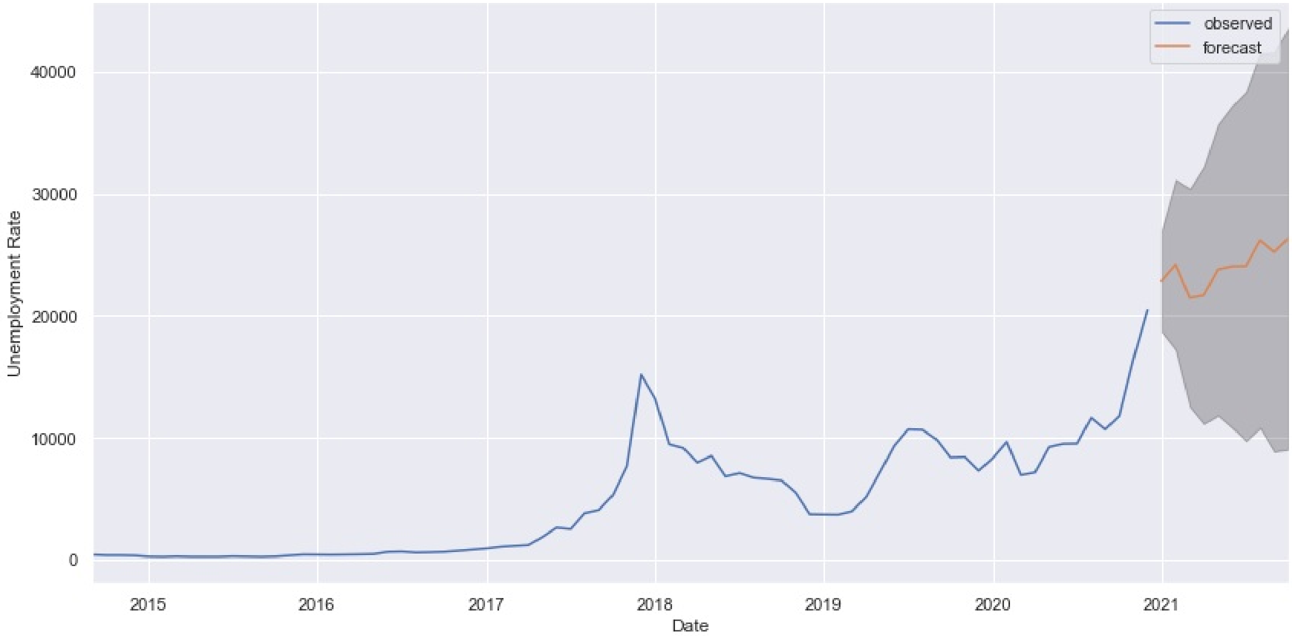
```
# Evaluation metrics are Squared Mean Error(SME) and Root Mean Squared Error(RMSE)
y_hat = prediction.predicted_mean
y_truth = y['2019-01-01':]

mse = ((y_hat - y_truth) ** 2).mean()
rmse = np.sqrt(mse)
print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))
print('The Root Mean Squared Error of our forecasts is {}'.format(round(rmse, 2)))
```

```
The Mean Squared Error of our forecasts is 4626427.06
The Root Mean Squared Error of our forecasts is 2150.91
```

Producing and visualizing forecasts

In [43]:

```
# forcasting for out of sample data
pred_uc = result.get_forecast(steps = 10)
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 = 0.25)
ax.set_xlabel('Date')
ax.set_ylabel('Unemployment Rate')

plt.legend()
plt.show()
```



Summary

- Our model clearly captured unemployment rates' seasonality.
- As we forecast further out into the future, it is natural for us to become less confident in our values.
- This is reflected by the confidence intervals generated by our model, which grow larger as we move further out into the future

In [ ]:

In [ ]:

In [ ]: