Published in Towards Data Science

Chris Price Follow

Jan 4, 2021 · 15 min read · Listen

Save

Estimating Causal Effects on Financial Time-Series with Causal Impact BSTS

A brief introduction to Google's Causal Impact library in Python & its utility in estimating causal effects on financial time-series.

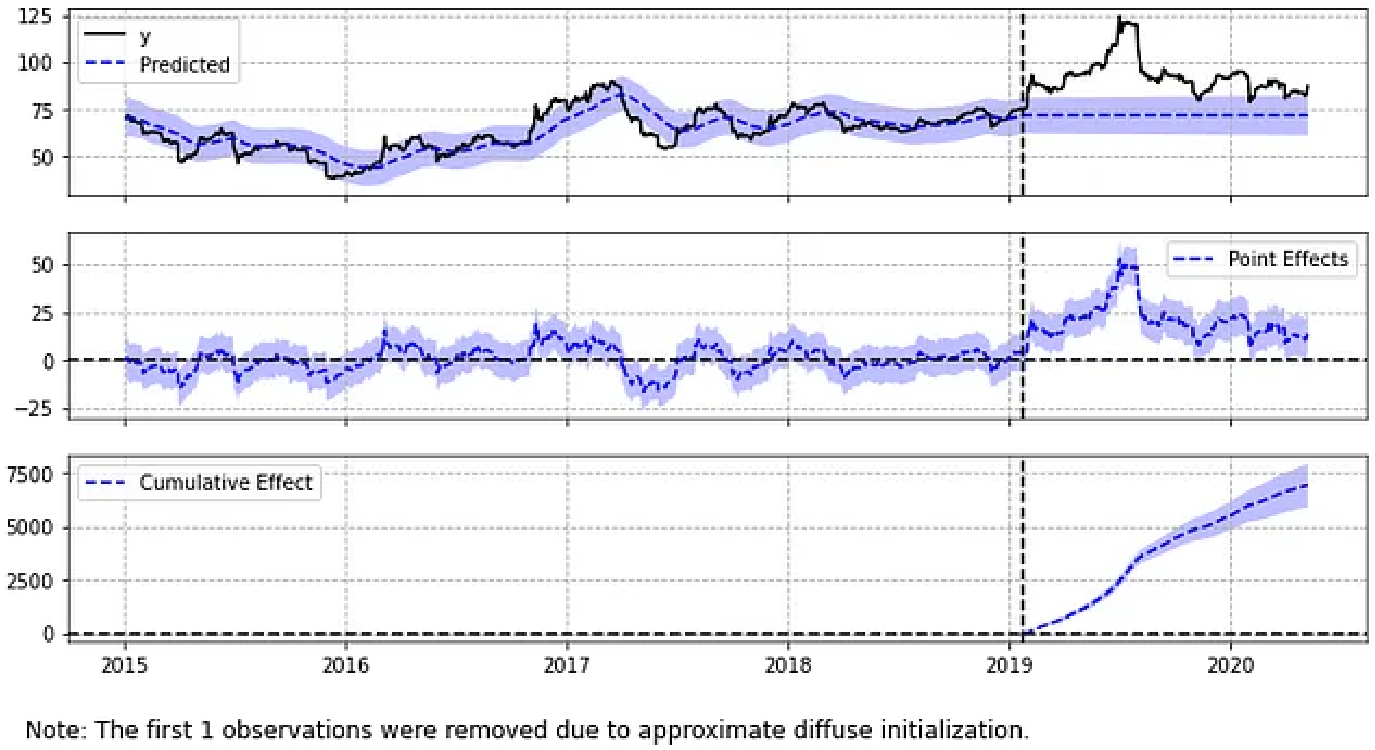


Image by Author

Introduction

The ability to quantify and explicate the impact of a known event on a dependent variable is a data science skill whose utility applies to innumerable disciplines. The impact of such analysis, however, and its ability to influence a business decision (notwithstanding the veracity of the analysis itself!) is only as a good as the Data Scientist/Analyst/Quant's ability to rationalise the underlying choice of model and its decision, as well as the requisite domain expertise and understanding of the dependent variable.

Google's Causal Impact library (implemented in both R and Python) can help us accomplish such a task in a very short space of time while providing methods that enable the user to fully explain the underlying modelling process and the model's decision.

In this article, we are going to briefly explore how we can implement a Causal Impact model to estimate the effect of the Vale dam collapse on the spot price of Iron Ore. You can find all of the code found in this article here.

What is Causal Impact?

Google's Causal Impact library provides a very straightforward implementation of a Structural Time-Series model that estimates the effect of a 'designed' intervention on a target time-series. This effect is measured by analysing the differences between the *expected* and the *observed* behaviour — specifically, the model generates a **forecast counterfactual** i.e. the *expected* observations of how the dependent variable *might* have evolved after the event *had the event not occurred.*

How does it work?

Originally developed as an R package, Causal Impact works by fitting a Bayesian Structural Time Series (BSTS) model to a set of target and control time series observations, and subsequently performs posterior inference on the counterfactual.

For reference, structural time-series models are state-space models for time-series data, and can be defined in terms of the following pair of equations:

$$
\begin{array}{ll}
(1) & y_t = Z_t^T \alpha_t + + \beta X_t + G_t \epsilon_t \\
(2) & \alpha_{t+1} = T_t \alpha_t + R_t \eta_t
\end{array}
$$

$$
\begin{gathered}
where \\
\epsilon_t \sim N(0, \sigma_t^2) \\
and \\
\eta_t \sim N(0, Q_t)
\end{gathered}
$$

Structural Time-Series Model Definition. Image by Author

In the above expressions:

- Eq 1. is the observation equation. This links the observed data y_t to a latent d-dimensional state vector, α_t.
- Equation 2. is the state equation; it governs the evolution of the state vector α_t through time. In other words, the alpha variable refers to the "state" of the time-series, and y_t is a linear combination of the states, plus a linear regression with some explanatory covariates, X, plus some epsilon, ϵ, of noise that is normally distributed about a mean of 0.
- *Epsilon_t* and *eta_t* are independent of all other unknowns.

At this stage, it is worth noting one of the key differences between BSTS models and traditional statistical/deep learning variants:

Traditional time series forecasting architectures such as Linear Regression models, which estimate their coefficients via Maximum Likelihood Estimation, or, on the more powerful end of the scale, an LSTM which learns a function that maps a sequence of past observations as input to an output observation. BSTS models, on the other hand, employ a probabilistic approach to modelling a time series problem, namely, they return a posterior predictive distribution over which we can sample to provide not only a forecast **but also a means of quantifying model uncertainty.**

Why is this useful?

Bayesian Structural-Time Series models are particularly powerful models in that they can generalise to a very large class of time series models such as ARIMA, SARIMAX, Holt-Winters to name but a few. You can observe, in the above expression, how we can

Example - Second-Order AutoRegressive Process:

Consider an example where we want to model a time series's temporal structure (autocorrelation). We can model the time series as a second-order autoregressive process AR(2), by adjusting the expressions above!:

Definition of an AR(2) process:

$$y_t = \phi 1 y_{t-1} + \phi 2 y_t - 2 + \epsilon_t,$$
$$where \qquad \epsilon_t \sim N(0, \sigma^2)$$

2nd Order AutoRegressive Process Definition. Image by Author

In state-space form:

$$y_t = [1, 0]\alpha_t$$

$$\alpha_{t+1} = \begin{bmatrix} \phi_1 & \phi_2 \\ 1 & 0 \end{bmatrix} \alpha_t + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \eta t$$

Where:

$$Z_t \equiv Z = [1 \quad 0]$$

and:

$$T_t \equiv T = \begin{bmatrix} \phi_1 & \phi_2 \\ 1 & 0 \end{bmatrix}$$
$$R_t \equiv R = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$
$$\eta_t \equiv \epsilon_{t+1} \sim N(0, \sigma^2)$$

AR(2) pr 242 7 y Author

In light of this flexibility to choose any time series model you want to fit your data, one can quickly see how powerful these models can be.

But this sounds like a lot of work?

Well, yes, but **fear not.** The wonderful thing about Causal Impact, should you require an expedient initial analysis, is that you **don’t have to** explicitly define *any* of the model’s structural components.

If this is indeed the case, and you do not specify a model as in input, a local level model is built by default and is one that estimates the salient structural components of your time series for you. With the local level model, the target time-series, y, is defined as:

$$y_t = \mu_t + \gamma_t + \beta X_t + \epsilon_t$$
$$\mu_{t+1} = \mu_t + \eta_{\mu t}$$

Local Level Model. Image by Author

Here, a given point in time is modelled as a random-walk component, mu_t (also known as the local level component). **Trend and seasonal components**, gamma_t, are modelled as unobserved components. The trend is modelled as a fixed intercept and the seasonal components using trigonometric functions with fixed periodicities and harmonics. For a more detailed mathematical explanation, one can refer to the Causal

Impact documentation **here**, and the **Statsmodels state-space seasonal documentation**, whose logic is followed by that of the Python variant of Causal Impact.

In summary, our implementation (in Python) therefore is reduced to a one-line expression!:

```
ci_model = CausalImpact(target, pre_period, post_period)
```

For those interested in how to build a Bayesian Structural Time-Series in detail using TensorFlow you can see **how in this article:**

https://towardsdatascience.com/structural-time-series-forecasting-with-tensorflow-probability-iron-ore-mine-production-897d2334c72b

Otherwise, let’s take a look at Causal Impact in action.

The Problem

We are going to explore how we can implement Causal Impact in estimating the effect of the Vale dam collapse on the spot price of Iron Ore. Whilst this event does not constitute a ‘designed’ intervention, utility still exists in the financial world in providing estimates of price moves in response to future events of a similar nature. Moreover, this is intended to serve as a demonstration of the utility of Google’s Causal Impact package in estimating the impact of an event on a response time-series. As a reminder, the link to the Python colab notebook can be found **here**.

Vale Dam Incident

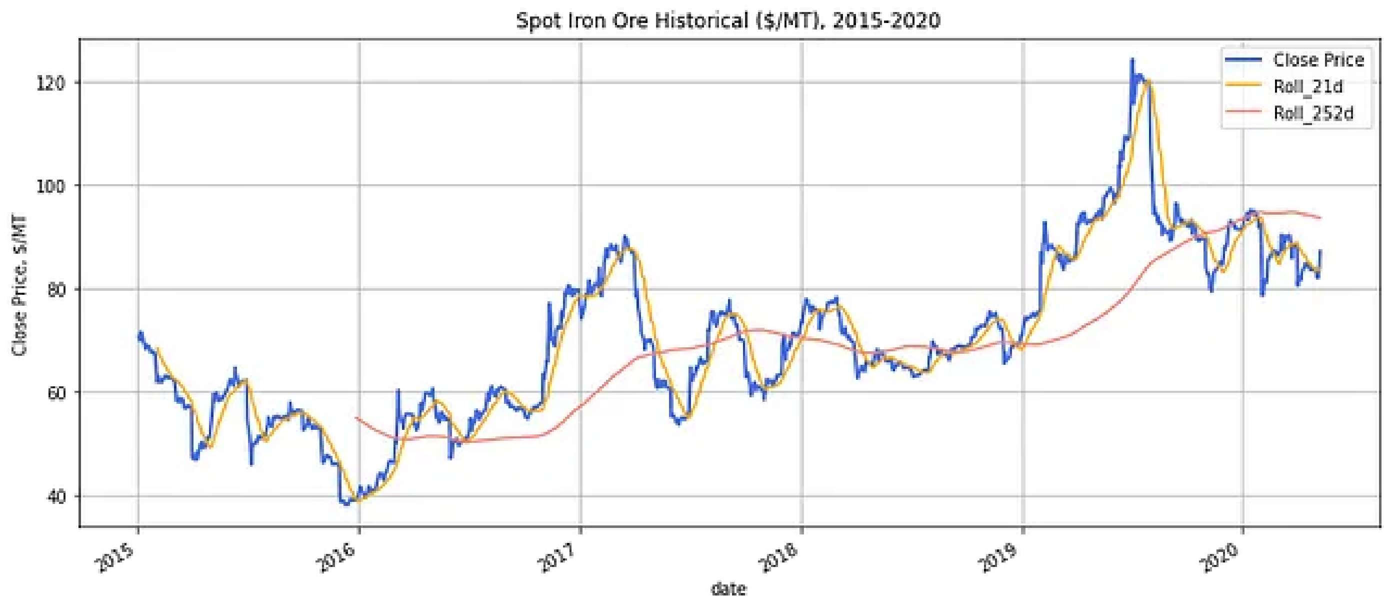
In this instance, we are estimating the impact of the Vale dam incident that occurred at Vale’s (the worlds largest producer of Iron ore) Córrego do Feijão mine on the 25th January 2019, on the spot price of Iron ore.

Caveat: Whilst modelling the outcome of any financial time-series is generally a highly-complicated, non-linear problem requiring far greater consideration and technical application than we are demonstrating, the intention of this demonstration is to illustrate the utility and expediency at which CI can perform such an analysis, and its interpretability — a pre-requisite quality in any business environment.

Spot Iron Ore Price Data

We begin by acquiring our spot iron ore price data, then plotting the close price of the spot iron ore time-series. We also create a 21 and 252 day rolling average to give us a directional steer on the current price movements:

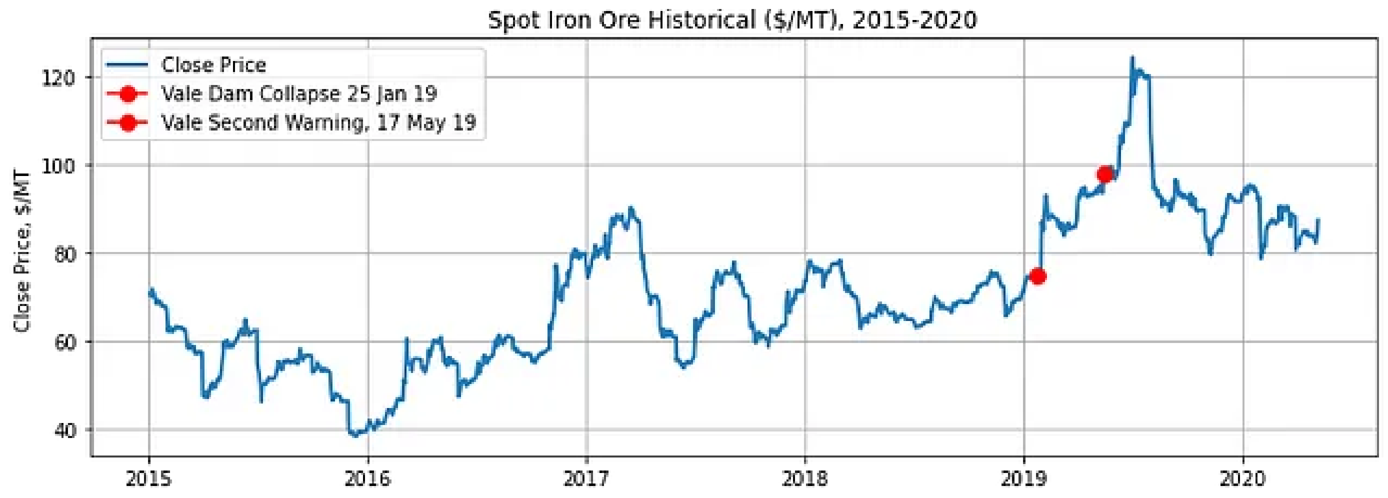
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from causalimpact import CausalImpact
from statsmodels.tsa.seasonal import seasonal_decompose

# Get spot prices
df = pd.read_excel(
    '/Users/CMP/OneDrive/PriceCurves/IO_Spot_2015010120200501.xlsx',
    index_col='date'
)
# Plot close price, 21d and 252 roll avg.
df['close'].plot(lw=2., figsize=(14,6), label='Close
Price',c='royalblue')
df['close'].rolling(21).mean().plot(lw=1.5,
label='Roll_21d',c='orange')
df['close'].rolling(252).mean().plot(lw=1.5, label='Roll_252d',
c='salmon')
plt.title('Spot Iron Ore Historical ($/MT), 2015-2020')
plt.ylabel('Close Price, $/MT')
plt.grid(); plt.legend()
```



Iron Ore Spot Price and 21d/252d Rolling Average, 2015–2020. Image by Author

Vale Dam Event

So far so good. Prior to defining our pre-event and post-event periods, we can get a better idea of the magnitude of the event itself visually by marking the date of the event and plotting it:



Annotated Spot Iron Ore Close Price. Image by Author

In the above chart, you can observe two events:

1. The **first** data point indicates the date of the Vale dam collapse.
2. The **second** data-point/event was a warning from Vale regarding the stability of one of its tailings dams.

Before we start — Model Assumptions:

As with all forecast problems, it is imperative that we fully consider the assumptions made by any model before applying it to our problem.

1. **Control Set**: In the case of Causal Impact, the model assumes that a set control time series exists that was ***not*** itself affected by the event. In the context of our problem, this assumption is not strictly relevant.

2. **External Covariates:** An additional assumption that **is** relevant, however, is that the model assumes that the relationship between external covariates and the target time series, as established during the pre-period, remains stable throughout the post-period. In the context of our problem, it is highly likely that features may be predictive of the price of iron ore **would likely also be affected by the event.**

3. **Model Priors:** The prior distributions for the unobserved states differ depending on whether you have chosen to implement Causal Impact in R, or Python. The R variant of Causal Impact sets the prior for the unobserved state based on the first observation of the target series, and the variance of the dataset. Statsmodels, on the other hand, uses a diffuse (uniform) prior. This shouldn't result in any major differences between the two but it is worth bearing in mind (see observations, criticisms & further analysis for the key differences between the R and Python packages).

Default Model

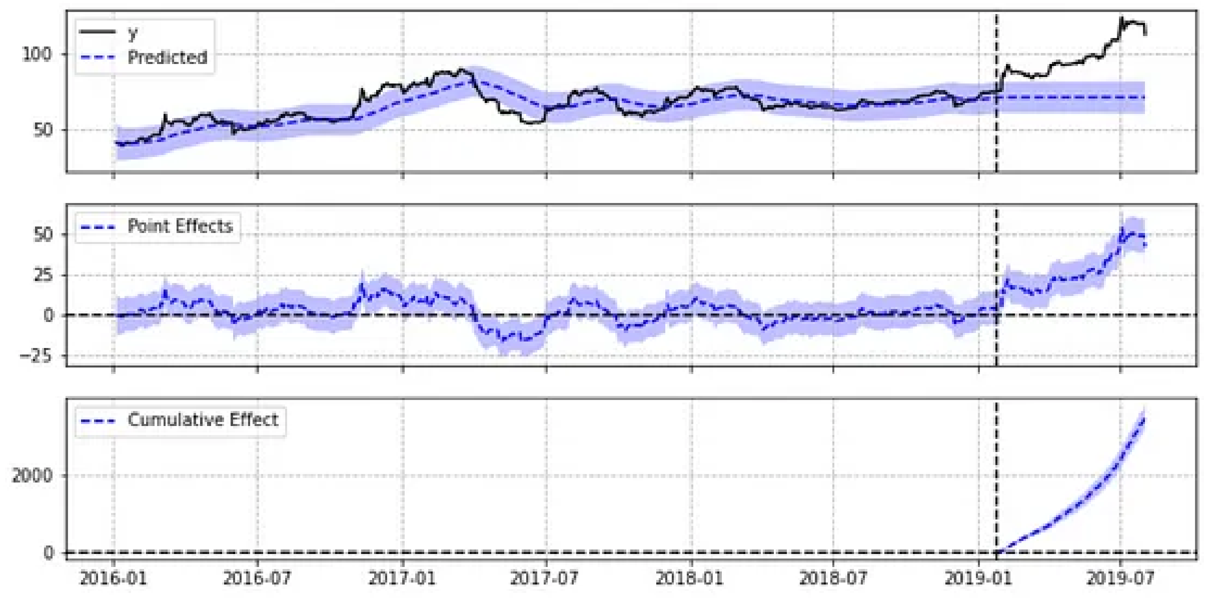
With the basic information above, let's fit the basic model to our spot price data, and examine the output. Implementation simple and straightforward:

```
# Define training data - period prior to the event
pre_period = ['2016-01-04', '2019-01-24']

# Define post-event period - i.e. time AFTER the event occurred.
post_period = ['2019-01-25', '2019-08-01']

# Instantiate CI model.
ci = CausalImpact(df['close'], pre_period, post_period)
```

Note: Causal Impact can (handily) accept date strings when specifying the bounds of our pre and post-event periods.

Fitting the Causal Impact model to our data returns an object whose methods when invoked, allow us to check model results, fitted parameters, etc. For now, we shall plot the results:

```
[ ]   ci.plot(figsize=(12, 6))
```



```
Note: The first 1 observations were removed due to approximate diffuse initialization.
```

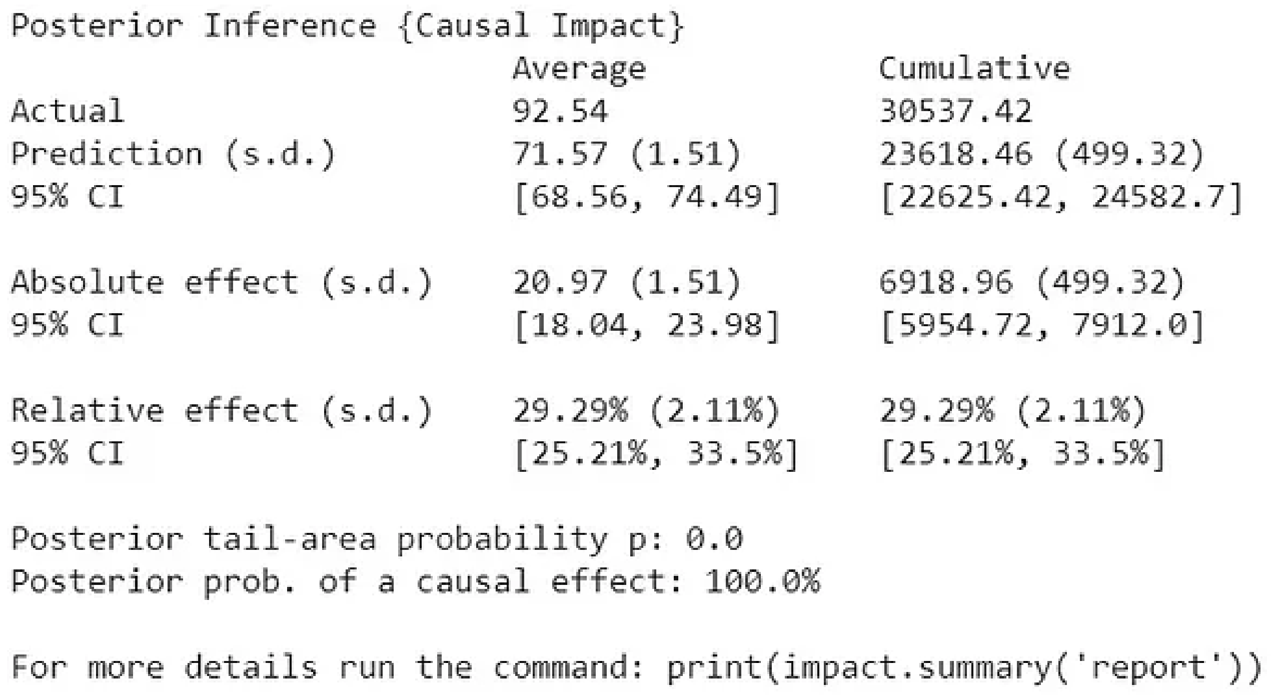
Causal Impact results, including forecast counterfactual, pointwise and cumulative effects. Image by Author.

As you can observe in the above chart, by calling the `.plot()` method on our CI object we can access the results of the fitting process. By default, the plot method renders three separate charts:

1. The **observed** 'post-event' time series and fitted model's forecast counterfactual
2. The ***pointwise*** causal effect, as estimated by the model. This is the difference between the observed outcome and the predicted outcome.
3. The **cumulative** effect.

Additionally, by invoking the CI object's `.summary()` method, we yield a convenient summary report:

```
print(ci.summary())
```



```
Posterior Inference {Causal Impact}
                          Average            Cumulative
Actual                    92.54              30537.42
Prediction (s.d.)         71.57 (1.51)       23618.46 (499.32)
95% CI                    [68.56, 74.49]     [22625.42, 24582.7]

Absolute effect (s.d.)    20.97 (1.51)       6918.96 (499.32)
95% CI                    [18.04, 23.98]     [5954.72, 7912.0]

Relative effect (s.d.)    29.29% (2.11%)     29.29% (2.11%)
95% CI                    [25.21%, 33.5%]    [25.21%, 33.5%]

Posterior tail-area probability p: 0.0
Posterior prob. of a causal effect: 100.0%

For more details run the command: print(impact.summary('report'))
```

Causal Impact Model: Summary Report. Image by Author.

Evaluation of Results

Examination of the above output reveals the results of the fitted model:

- **The Average** column refers to the average price (over the time period) during the post-intervention period.
- The **Cumulative** column is the sum of the individual daily observations — not so useful in our case, however very useful if the dependant variable is a metric whose cumulative sum is part of your experiment; additional sales, clicks, etc.

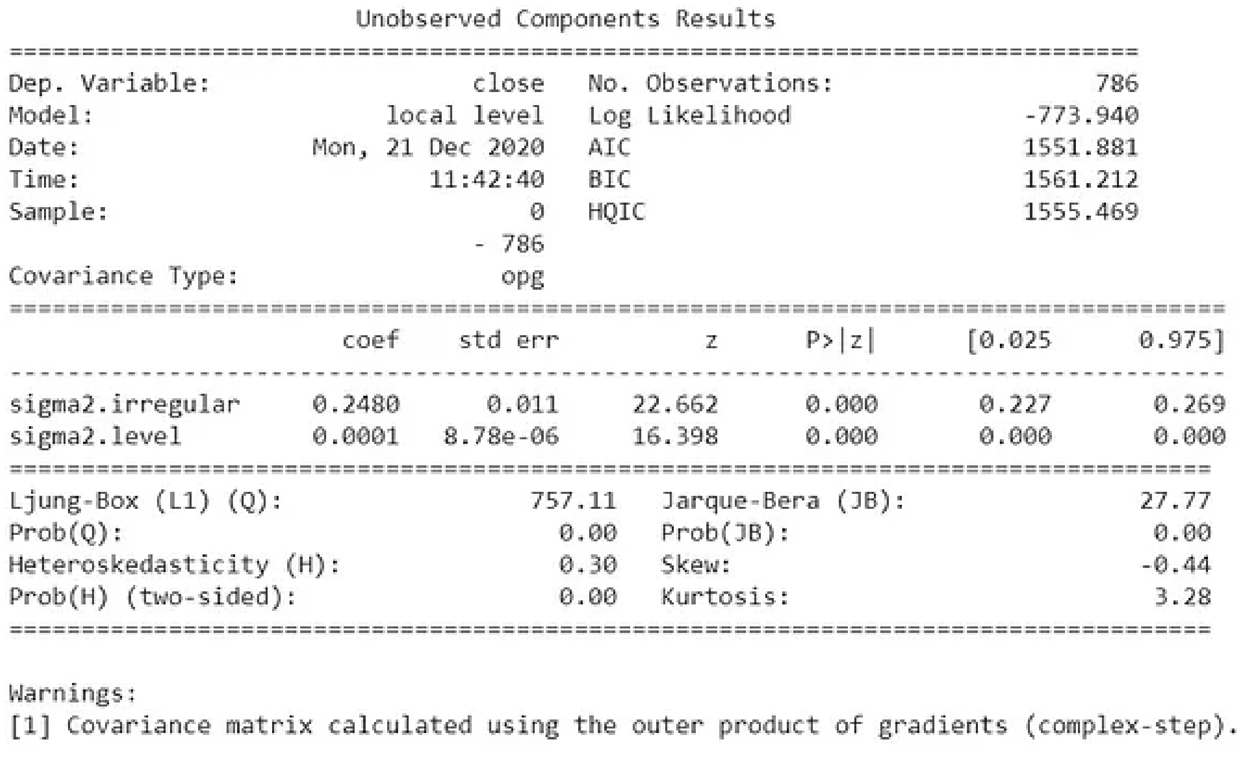
A cursory glance at the forecast counterfactual and the point-wise effect suggests that an event of this magnitude has had a significant effect on the spot price. Indeed, the model asserts, with confidence, that the Vale dam incident had an absolute causal effect of $21, varying from $18.04 to $23.98. It is also important to note the p-value here (< .05) to understand whether the observed behaviour is statistically significant or simply occurred by chance.

Fitted Model Diagnostics

It is immediately apparent, however, that our default model's forecast counterfactual doesn't look terribly convincing. For the most part, it would appear directionally accurate, however, it has clearly failed to capture the salient price movements, and the forecast appears to lag behind the observed spot price.

We can check the fitted model's parameters and diagnostics to assess whether the model conforms to its underlying statistical assumptions:

```
ci.trained_model.summary()
```



```
                       Unobserved Components Results
==============================================================================
Dep. Variable:                  close   No. Observations:                  786
Model:                    local level   Log Likelihood                -773.940
Date:                Mon, 21 Dec 2020   AIC                           1551.881
Time:                        11:42:40   BIC                           1561.212
Sample:                             0   HQIC                          1555.469
                                - 786
Covariance Type:                  opg
====================================================================================
                       coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------
sigma2.irregular     0.2480      0.011     22.662      0.000       0.227       0.269
sigma2.level         0.0001   8.78e-06     16.398      0.000       0.000       0.000
===================================================================================
Ljung-Box (L1) (Q):                 757.11   Jarque-Bera (JB):                27.77
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.30   Skew:                            -0.44
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.28
===================================================================================

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

Fig. 1: Fitted model results and parameters. Image by Author.

```
# Plot residuals diagnostics
_ = ci.trained_model.plot_diagnostics(figsize=(14,6))
```

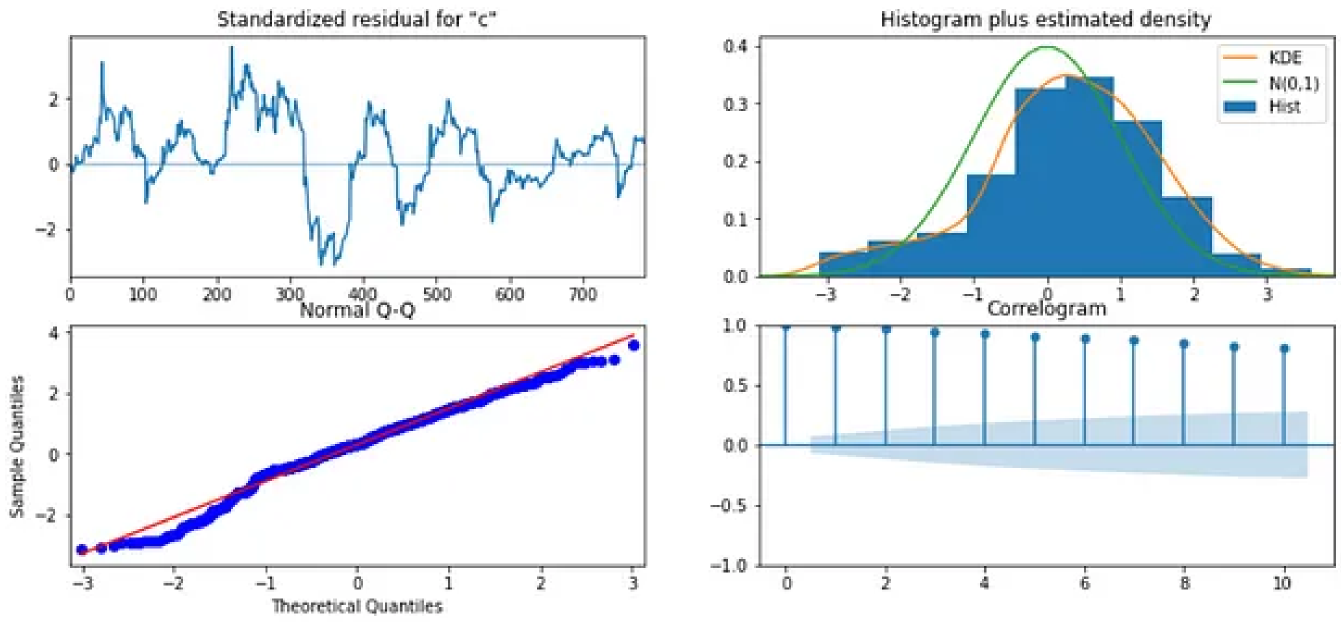


Fig.2: Fitted model's residuals. Image by Author.

Examination of Figure.1 above shows the parameters of the fitted model. A full explanation of the individual results is beyond the scope of this article, however the salient points, namely the model components; sigma2.irregular and sigma2.level and their coefficients show how weakly predictive they are of our target, the spot price of iron ore.

Indeed, if we consider figure 2 and the plot of the residuals we can examine the magnitude of the model errors, 'Standardized residual for c'. We can also observe that the errors follow a distinctly non-normal distribution, and exhibit strong autocorrelation. Nevertheless, we have effectively established a **baseline model** to estimate the effect of the event on our target variable.

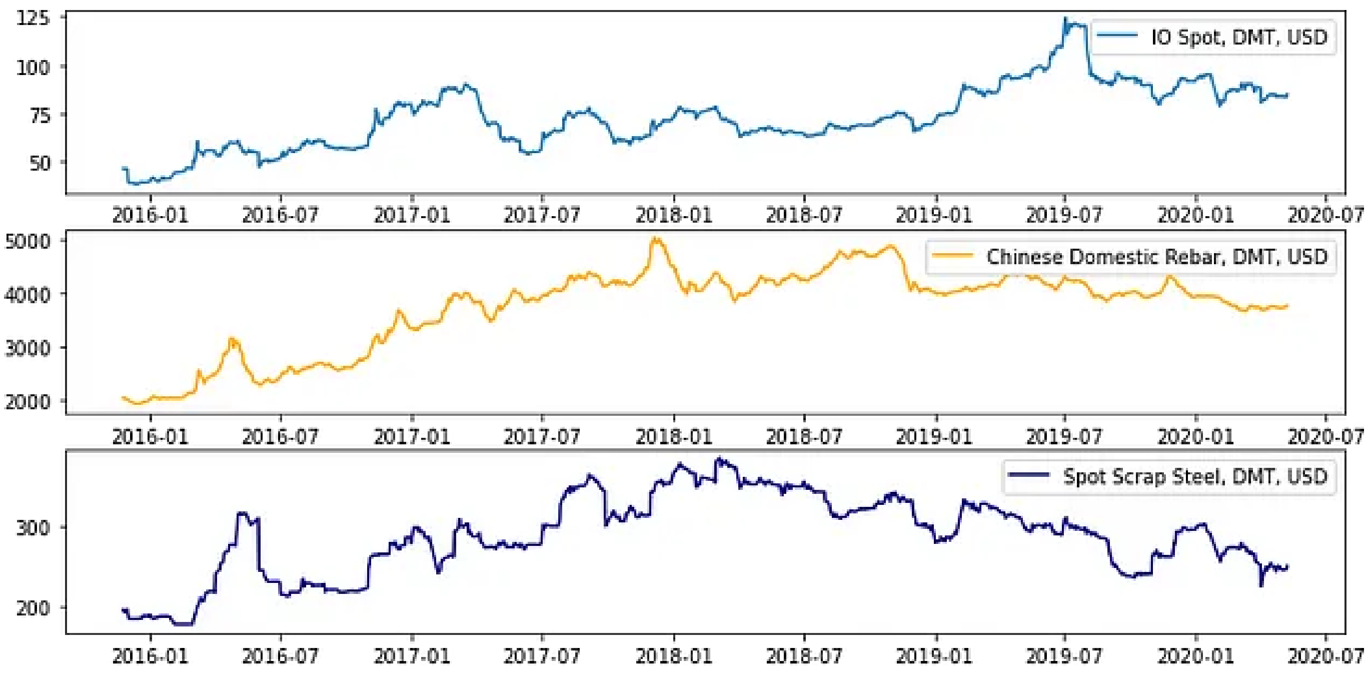
Customised Model

We have just implemented our first Causal Impact model and estimated the causal effect of the Vale dam incident on our spot iron ore data. So far, we've let the package decide how to construct a time-series model for our spot price data, and found, as a result of the fitted model's diagnostics, that we cannot be confident of the inferred effect.

As mentioned above, a useful quality of structural time-series models is their modularity, affording us the flexibility to model individual behavioural dynamics of our time series such as seasonality, for example. The Causal Impact module offers several options that enable us to accomplish this, which we will exploit in an attempt to improve our inferred counterfactual.

Seasonal Model & Model with Exogenous Features (Covariates)

In an attempt to improve our model and the forecast counterfactual, we will employ our domain expertise and adjust our model to include a known seasonal component frequently manifest in spot iron ore price action, and incorporate two features that exhibit a known linear correlation with the spot price of iron ore: spot steel scrap and Chinese domestic steel reinforcing bar (rebar):



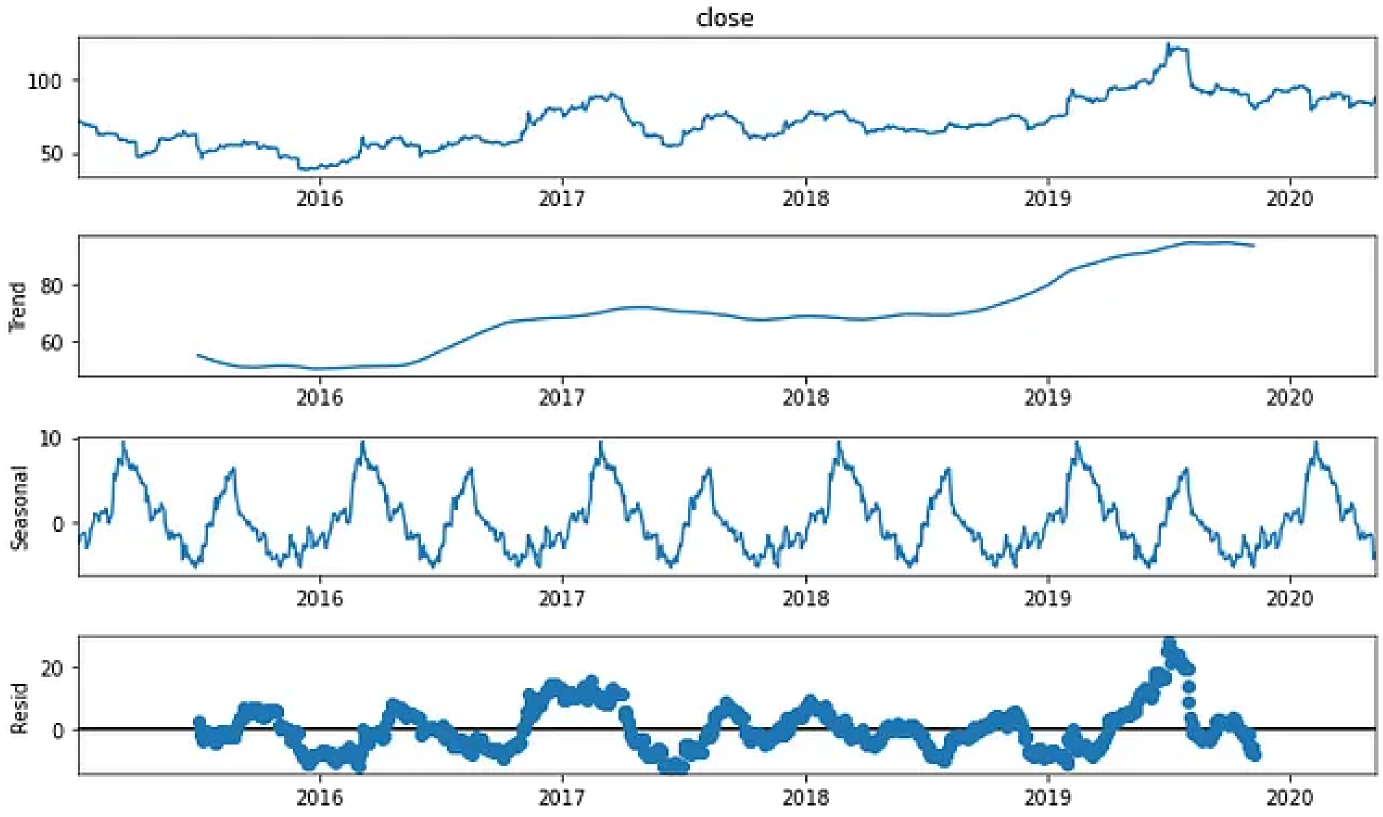
Spot Iron Ore price curve with the addition of our two new features (covariates); rebar and steel scrap. Image by Author.

Adding Seasonal Components

We will begin by trialling the addition of a **known** seasonal component to our model. For many bulk commodities, **prices tend to rise in the Chinese summer and winter** ahead of peak periods of construction in the spring and autumn. The same behaviour can typically be observed with that of iron ore. For reference, a seasonal component can be described as a pattern that repeats itself with periodicity i.e. the signal is periodic.

By decomposing our time series into its constituent structural components, we observe the degree to which the aforementioned seasonality affects the spot price:

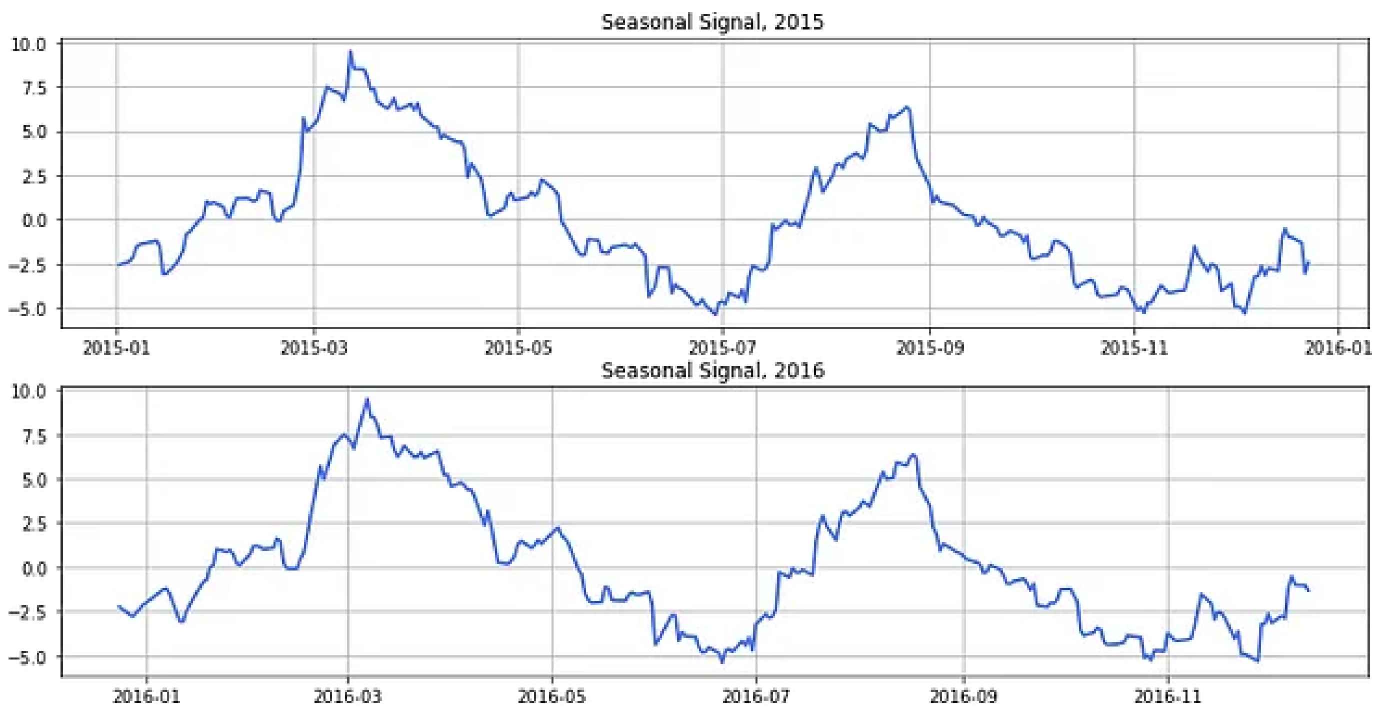
```
s_dc = seasonal_decompose(df['close'], model='additive',
period=252).plot()
```



Spot Iron Ore time-series, decomposed. Image by Author.

Note: Statsmodels seasonal_decompose performs a naive decomposition of our time series — more sophisticated approaches would, and should typically be employed, particularly as our time-series is a financial one. For the purposes of this article and demonstrating how to add these as components to our model, however, this shall suffice.

Isolating a few examples corroborates our prior belief/domain expertise: we can see that the seasonal component's frequency and amplitude correspond with the Chinese summer and winter:



Isolated seasonal component observations, 2015 & 2016. Image by Author.

It would appear that the signal has a rough periodicity of 146 days, and a harmonic of 1, although these are crude interpretations. We can observe that the amplitude of the winter peak in both examples is of lesser magnitude than that of the summer, therefore the signal itself is not strictly symmetric. For now, we will proceed on our assumptions for the purposes of demonstration.

Incorporating seasonal components in Causal Impact is very straightforward: the Causal Impact class accepts a list of dictionaries containing the periodicity of each seasonal signal, and, if known, the harmonics:

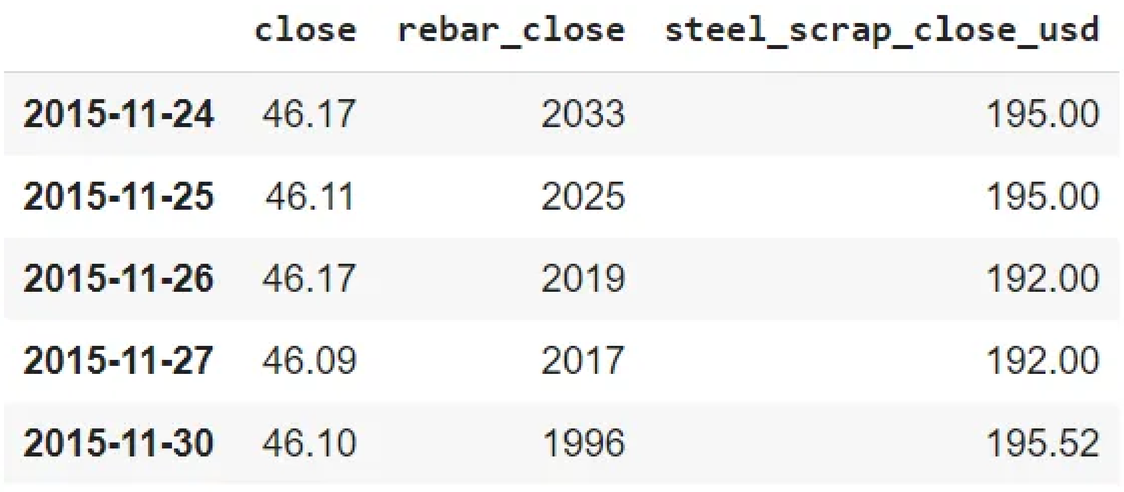
```
# Example - Adding seasonal components to a CI model

ci = CausalImpact(
    df['close'],
    pre_period,
    post_period,
    nseasons=[{'period': 146, 'harmonics': 1}]
)
```

Adding Exogenous Covariates

We can now add the external covariates to our model, spot steel scrap price and Chinese domestic reinforcing bar.

Again, adding external covariates to our Causal Impact model is simple. These can be passed as a Pandas data frame along with our target variable. As per the source code, Causal Impact expects the target/label column to be in the first index (0). All subsequent columns are registered as external covariates:

```
df_1 = pd.concat([df, steel_scrap_df, rebar_df], join='inner', axis=1)
```



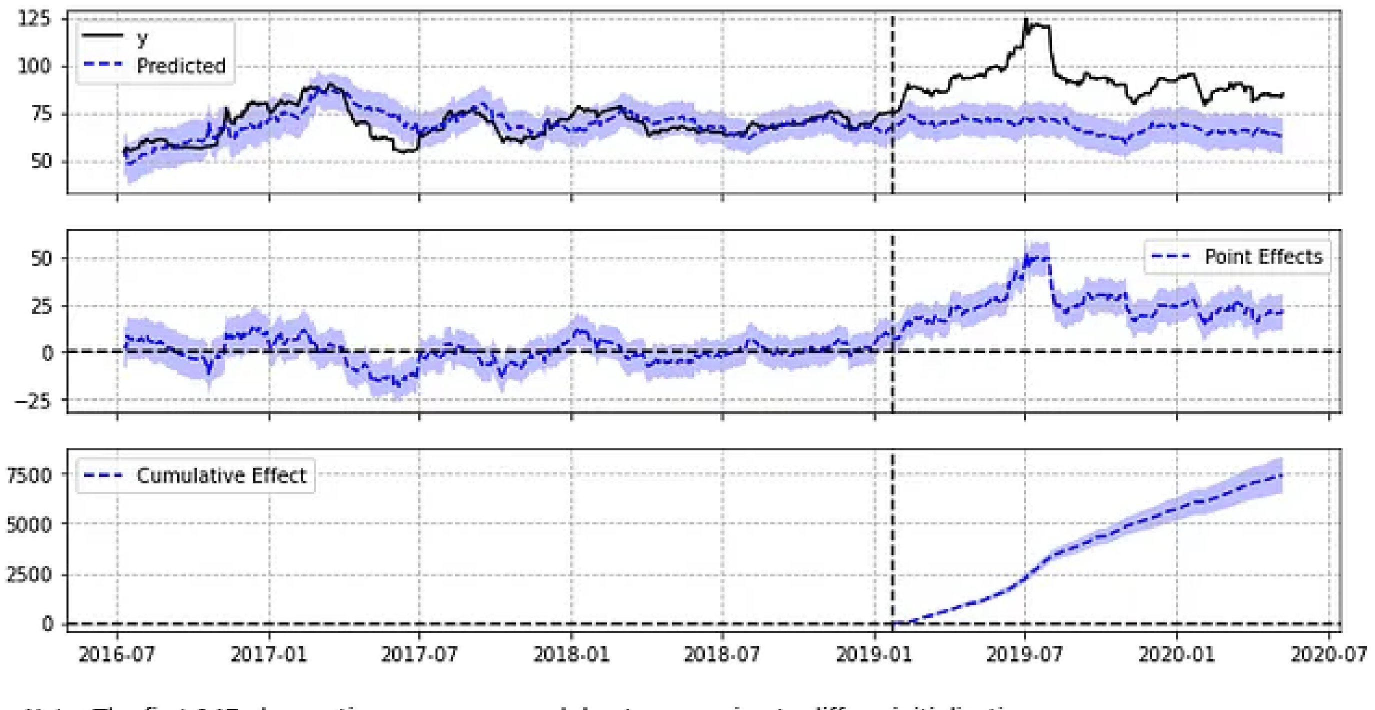
	close	rebar_close	steel_scrap_close_usd
2015-11-24	46.17	2033	195.00
2015-11-25	46.11	2025	195.00
2015-11-26	46.17	2019	192.00
2015-11-27	46.09	2017	192.00
2015-11-30	46.10	1996	195.52

Our price curves DataFrame. Image by Author.

We then pass this data frame to our Causal Impact model, along with the same pre and post-period event definitions and our seasonal component. Note that we will not specify the harmonics of our seasonal component*:

```
# Fit new seasonal + beta coef variant
ci_1 = CausalImpact(df_1, pre_period, post_period, nseasons=
[{'period': 146}])

# Plot result
ci_1.plot(figsize=(12, 6))
```

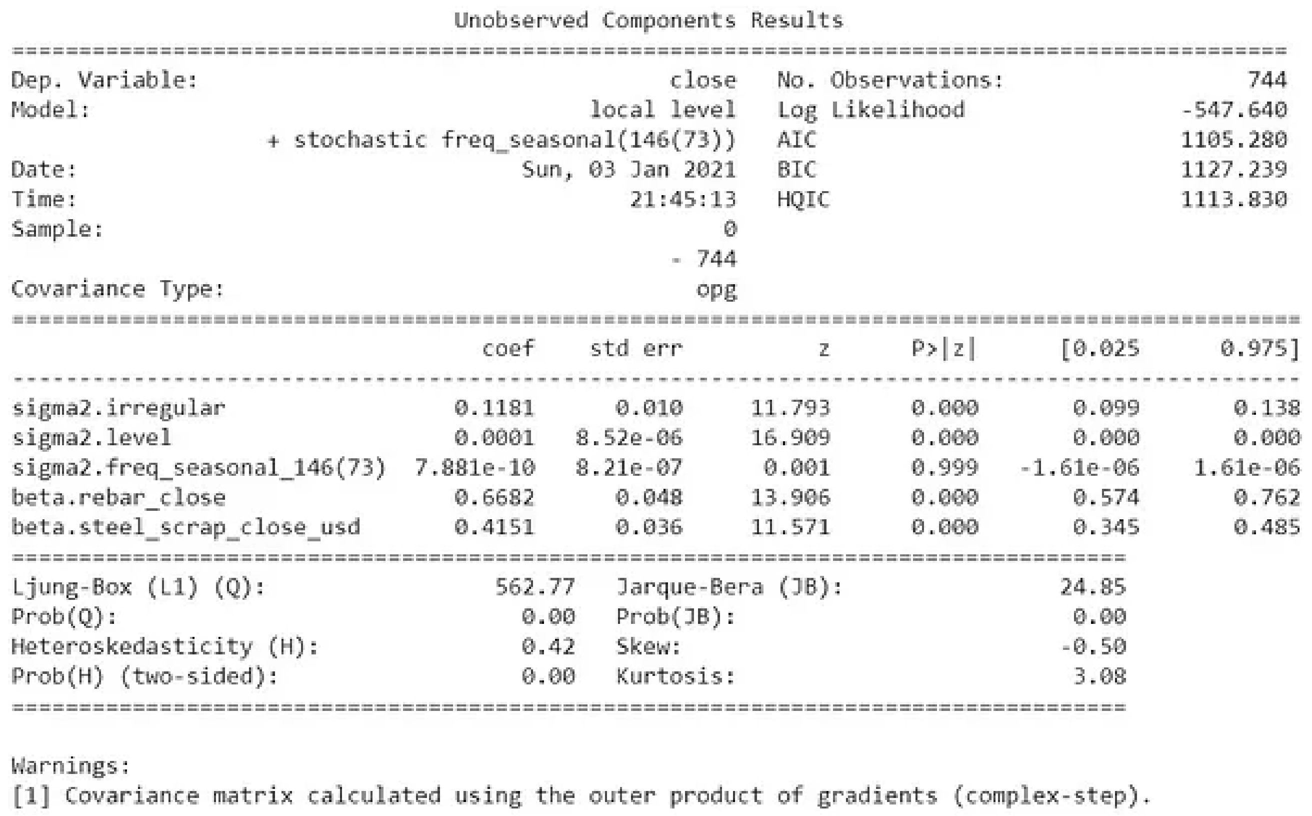
**model defaults to calculating this as math.floor(periodicity / 2))*



Note: The first 147 observations were removed due to approximate diffuse initialization.

Forecast counterfactual: seasonal + external covariate CI model variant. Image by Author.

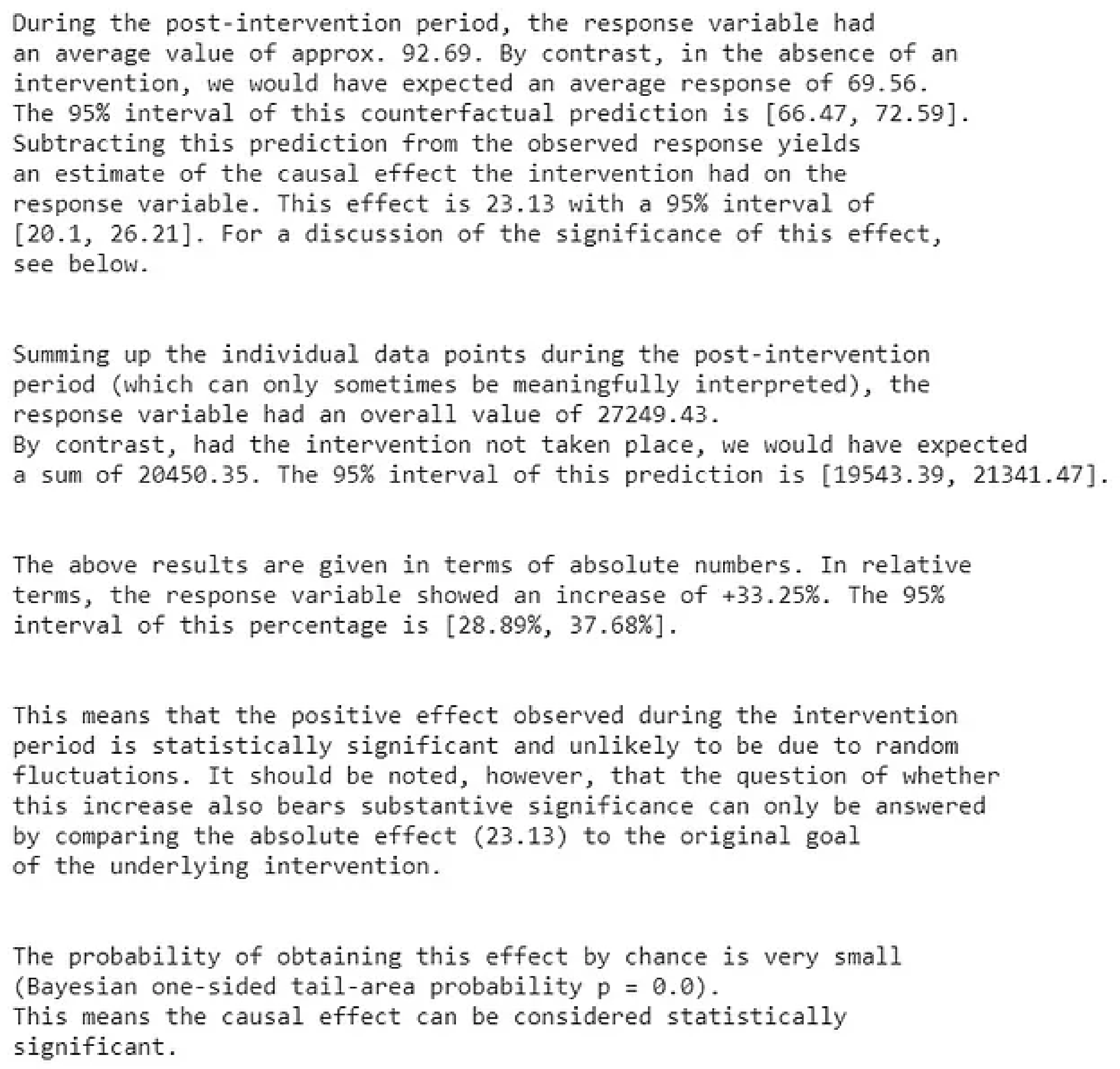
Upon inspecting the counterfactual of our new model, we can observe a far more credible result. It would still appear as though we need to account for the temporal (autocorrelative) structure in the time series and can include this in a subsequent model, but the result would appear credible for a first pass. Upon checking the fitted model's parameters, we can see that our seasonal component and it's definition needs re-evaluating, whereas our external covariates explain a lot of what is observed in our response variable:



```
                           Unobserved Components Results
===============================================================================================
Dep. Variable:                                     close   No. Observations:             744
Model:                                       local level   Log Likelihood            -547.640
                 + stochastic freq_seasonal(146(73))   AIC                       1105.280
Date:                                   Sun, 03 Jan 2021   BIC                       1127.239
Time:                                           21:45:13   HQIC                      1113.830
Sample:                                                0
                                                   - 744
Covariance Type:                                     opg
================================================================================================
                                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------------------------
sigma2.irregular               0.1181      0.010     11.793      0.000       0.099       0.138
sigma2.level                   0.0001   8.52e-06     16.909      0.000       0.000       0.000
sigma2.freq_seasonal_146(73) 7.881e-10   8.21e-07      0.001      0.999   -1.61e-06    1.61e-06
beta.rebar_close               0.6682      0.048     13.906      0.000       0.574       0.762
beta.steel_scrap_close_usd     0.4151      0.036     11.571      0.000       0.345       0.485
==================================================================================
Ljung-Box (L1) (Q):                 562.77   Jarque-Bera (JB):                24.85
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.42   Skew:                            -0.50
Prob(H) (two-sided):                  0.00   Kurtosis:                         3.08
==================================================================================

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

Seasonal + external covariate model — fitted parameters. Image by Author.

Finally, we can have our model estimate the impact as a result of the Vale dam incident on our target with the useful method, `.summary('report')` that returns a detailed explanation of the observed effects:



```
During the post-intervention period, the response variable had
an average value of approx. 92.69. By contrast, in the absence of an
intervention, we would have expected an average response of 69.56.
The 95% interval of this counterfactual prediction is [66.47, 72.59].
Subtracting this prediction from the observed response yields
an estimate of the causal effect the intervention had on the
response variable. This effect is 23.13 with a 95% interval of
[20.1, 26.21]. For a discussion of the significance of this effect,
see below.


Summing up the individual data points during the post-intervention
period (which can only sometimes be meaningfully interpreted), the
response variable had an overall value of 27249.43.
By contrast, had the intervention not taken place, we would have expected
a sum of 20450.35. The 95% interval of this prediction is [19543.39, 21341.47].


The above results are given in terms of absolute numbers. In relative
terms, the response variable showed an increase of +33.25%. The 95%
interval of this percentage is [28.89%, 37.68%].


This means that the positive effect observed during the intervention
period is statistically significant and unlikely to be due to random
fluctuations. It should be noted, however, that the question of whether
this increase also bears substantive significance can only be answered
by comparing the absolute effect (23.13) to the original goal
of the underlying intervention.


The probability of obtaining this effect by chance is very small
(Bayesian one-sided tail-area probability p = 0.0).
This means the causal effect can be considered statistically
significant.
```

Seasonal + external covariate model — report. Image by Author

Here, we can see that the model concludes, with significance, that the incident resulted in a +33% increase in the spot price of iron ore — and logically, therefore, had the incident not occurred, then the spot price would have traded 33% lower during this period.

Conclusion

In this article, we have learned how Google's Causal Impact package can be used to estimate the causal effect of an intervention on an observed time-series. Specifically, we implemented a CI model to estimate the effect of the Vale dam incident on the spot price of Iron Ore.

Furthermore, we have learned:

- What Bayesian Structural Time-Series Models are, their capabilities and to a degree, their limitations (see below).
- What Google's Causal Impact library is, and what it does.
- How to implement a basic/default Causal Impact model.
- How to implement a more sophisticated variant, by defining and adding known seasonal components and linearly correlated exogenous variables as linear covariates.
- How to evaluate our fitted model's results.

Observations, Criticisms & Further Analysis

- **Parameter Estimation:** Depending on whether you implement Causal Impact in R or Python, you may find your models return disparate results. This is as a result of the different estimation methods employed by the respective libraries: The Python variant employs a Statsmodels Unobserved Components implementation to model the target time-series and whose parameters are estimated using maximum likelihood. The R variant employs the BSTS library whose parameters are estimated with Bayesian MCMC.
- **Seasonality:** See limitations mentioned above under 'Adding seasonal component'
- **Models Architectural Ability**: It is a good idea to examine how well the target variable can be predicted *before the intervention,* by running `CausalImpact()` on a **contrived intervention**. This acts as a sanity check as to the model's ability to capture the relevant structure in the data following this imaginary intervention. Logically, we would expect **not** to find a significant effect, i.e., counterfactual estimates and actual data should agree reasonably close.
- **Model Assumptions:** see 'model assumptions' above. Further note that we did not model for autocorrelation, inherent in nearly all financial time-series.

- **Custom models:** Please note that did not cover how to define custom models in this article, which would be verbose, and out of scope. However, a parameter of the Causal Impact class, `model` , enables us to pass any Statsmodels state-space model as an argument, illustrating the power and flexibility of the CI library.

Many thanks for taking the time to read this article and as always, I welcome all constructive criticisms and comments.

References

1. CausalImpact 1.2.1, Brodersen et al., Annals of Applied Statistics (2015), http://google.github.io/CausalImpact/"

2. https://www.ft.com/content/8452e078-7880-11e9-bbad-7c18c0ea0201

3. https://www.businessinsider.com.au/iron-ore-price-seasonality-2018-1

Bayesian Machine Learning Data Science Time Series Forecasting Time Series Analysis

Bayesian Statistics