Gaussian Process Time Series

Estimating the Effect Salary Transparency Has on Statewide Hiring

Authors

- Kaia Lindberg (pkx2ec)
- Will Johnson (hmd9tv)

External Code Reference: https://docs.pymc.io/en/v3/pymc-examples/examples/gaussian_processes/GP-MaunaLoa.html

- Muana Loa C02 Predictions using Gaussian Processes
- PyMC3 Documentation
- Copyright 2018
- The PyMC Development Team

In [104…
```
import arviz as az
import numpy as np
import pandas as pd
import pymc3 as pm
import theano.tensor as tt

from bokeh.io import output_notebook
from bokeh.models import BoxAnnotation, Label, Legend, Span
from bokeh.palettes import brewer
from bokeh.plotting import figure, show

output_notebook()
```

BokehJS 2.4.3 successfully loaded.

In [105…
```
# ******************************************
# ************ CONFIGURATIONS ***************
# ******************************************
%config InlineBackend.figure_format = 'retina'
RANDOM_SEED = 8927
np.random.seed(RANDOM_SEED)
az.style.use("arviz-darkgrid")
```

In [106…
```
# ******************************************
# *************** FUNCTIONS *****************
# ******************************************
def clean_data(filename, header = 13):
```

```
    '''
    Processes data and reformat data from BLS for analysis

    Inputs:
    filename: name of csv file (if in current directory) or full file path
    header: defaults to 13 (header rows in file I looked at) but can update


    Outputs:
    cleaned and reformatted dataframe
    '''
    # Load data, skip header
    df = pd.read_csv(filename, header = header)

    # Re-format data
    df = df.melt(id_vars=["Year"], var_name="Month", value_name="Hires")

    # Concatenate year and month columns
    df['Date'] = df['Month'] + df['Year'].map(str)
    # Drop year/month columns
    df.drop(['Month', 'Year'], axis = 1, inplace = True)
    # Convert to datetime
    df['Date'] = pd.to_datetime(df['Date'])
    # Sort by date to get in chronological order
    df = df.sort_values('Date')

    # Remove dates for which hires is NaN
    # these happen when moving months to the rows
    df = df[df['Hires'].notnull()]


    return df

def dates_to_idx(timelist):
    reference_time = pd.to_datetime("2012-01-01")
    t = (timelist - reference_time) / pd.Timedelta(365, "D")
    return np.asarray(t)

def plot_traces(traces, retain=0):
    '''
    Convenience function:
    Plot traces with overlaid means and values
    '''
    ax = pm.traceplot(traces[-retain:],
                      lines=tuple([(k, {}, v['mean'])
                                   for k, v in pm.summary(traces[-retain:]).

    for i, mn in enumerate(pm.summary(traces[-retain:])['mean']):
        ax[i,0].annotate('{:.2f}'.format(mn), xy=(mn,0), xycoords='data'
                    ,xytext=(5,10), textcoords='offset points', rotation=90
                    ,va='bottom', fontsize='large', color='#AA0022')
```

In [107…

```
# ****************************************
# *********** DATA LOAD/CLEANING ************
# ****************************************
data_monthly = clean_data('BLS_CO_Hires_Nonfarm_2012_to_2022_notSA.csv')
data_monthly.head()
```

Out[107]:

	Hires	Date
0	65.0	2012-01-01
11	58.0	2012-02-01
22	71.0	2012-03-01
33	88.0	2012-04-01
44	113.0	2012-05-01

In [108…

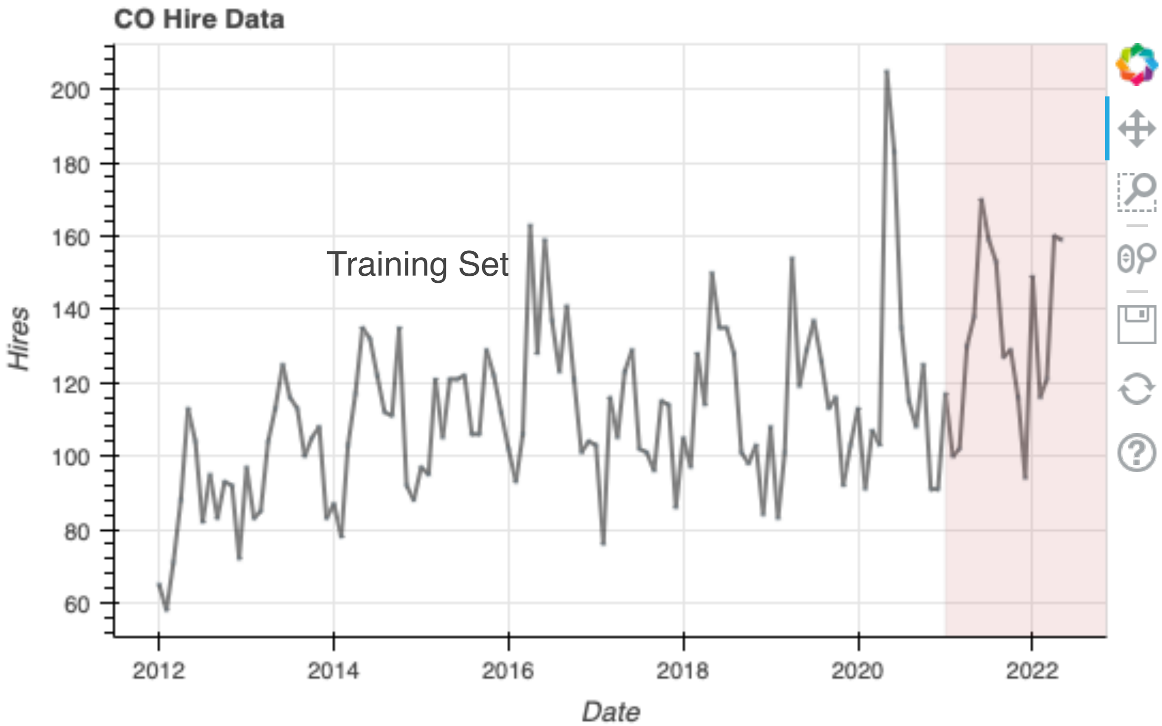
```
data_monthly = data_monthly.set_index('Date')
```

In [109…

```
t = dates_to_idx(data_monthly.index)

# normalize Hire levels
y = data_monthly["Hires"].values
first_hires = y[0]
std_hires = np.std(y)
y_n = (y - first_hires) / std_hires


data_monthly = data_monthly.assign(t=t)
data_monthly = data_monthly.assign(y_n=y_n)
```

In [110…

```
# *****************************************
# ***************** EDA *******************
# *****************************************

# make plot
p = figure(
    x_axis_type="datetime",
    title="CO Hire Data",
    plot_width=550,
    plot_height=350,
)
p.yaxis.axis_label = "Hires"
p.xaxis.axis_label = "Date"
predict_region = BoxAnnotation(
    left=pd.to_datetime("2021-01-01"), fill_alpha=0.1, fill_color="firebrick
)
p.add_layout(predict_region)
ppm400 = Span(location=400, dimension="width", line_color="red", line_dash="
p.add_layout(ppm400)

p.line(data_monthly.index, data_monthly["Hires"], line_width=2, line_color="
p.circle(data_monthly.index, data_monthly["Hires"], line_color="black", alph

train_label = Label(
    x=100,
    y=165,
    x_units="screen",
    y_units="screen",
    text="Training Set",
    render_mode="css",
    border_line_alpha=0.0,
    background_fill_alpha=0.0,
)
test_label = Label(
    x=585,
    y=80,
    x_units="screen",
    y_units="screen",
    text="Test Set",
    render_mode="css",
    border_line_alpha=0.0,
    background_fill_alpha=0.0,
)

p.add_layout(train_label)
p.add_layout(test_label)
show(p)
```



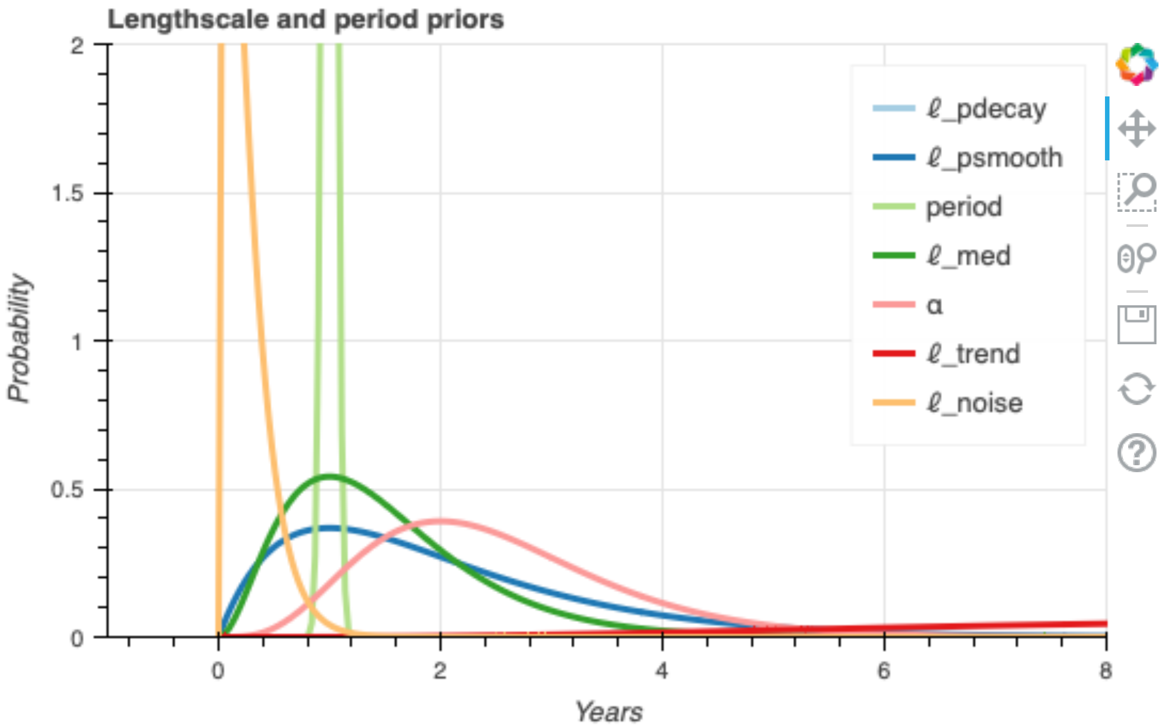
Looking at the above, we notice that in May and June of 2020 the numbers are wildly off trend. This is likely due to reactions to the 2020 pandemic, so for the purposes of our analysis we will ignore these outliers.

In [111…
```
drop_dates = [
    pd.to_datetime('2020-05-01'),
    pd.to_datetime('2020-06-01')
]
data_monthly = data_monthly[data_monthly.index.isin(drop_dates)==False]
```

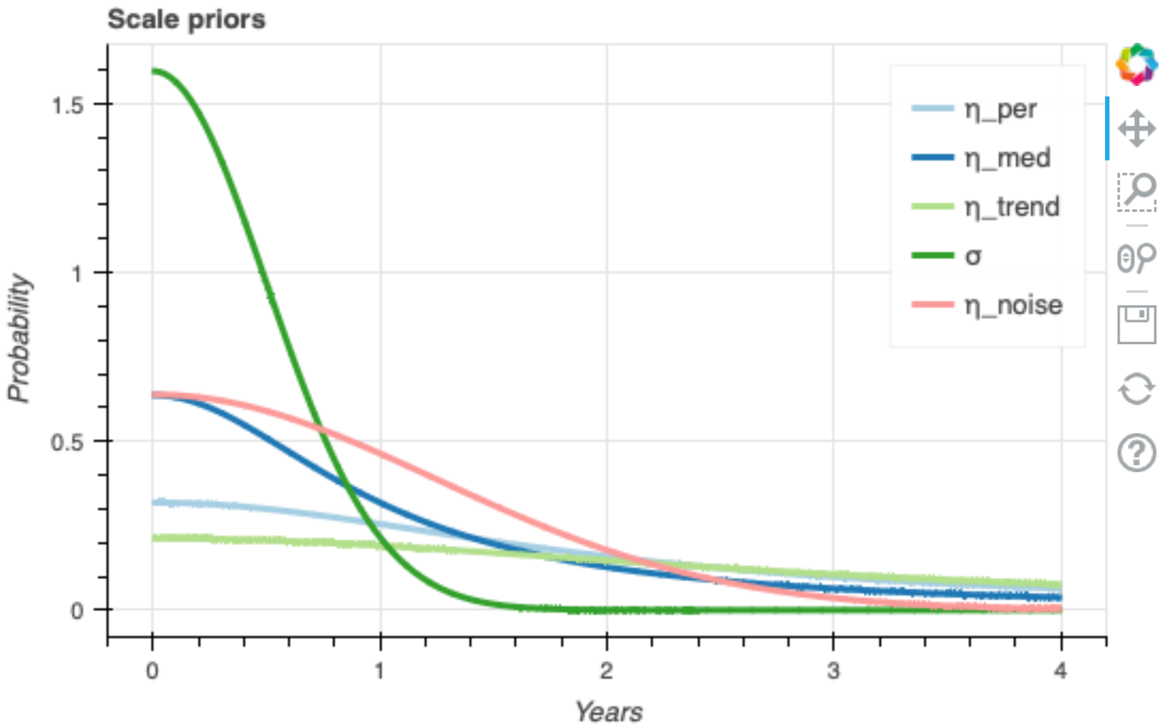
In [112…
```
## split into training and test set
sep_idx = data_monthly.index.searchsorted(pd.to_datetime("2021-01-01"))
data_pre = data_monthly.iloc[: sep_idx + 1, :]
data_post = data_monthly.iloc[sep_idx:, :]
```

In [113…

```
x = np.linspace(0, 150, 5000)
# Priors Params
psmooth_priors = dict(alpha=2, beta=1)
l_noise_params = dict( alpha=1.5, beta=6)
l_med_params = dict(alpha=3, beta=2)
l_trend_params = dict(alpha=4, beta=0.25)
alpha_params = dict(alpha=5, beta=2)
period_params = dict(mu=1.0, sigma=0.05)
decay_params = dict(alpha=10, beta=0.05)

priors = [
    ("ℓ_pdecay", pm.Gamma.dist(**decay_params)),
    ("ℓ_psmooth", pm.Gamma.dist(**psmooth_priors)),
    ("period", pm.Normal.dist(**period_params)),
    ("ℓ_med", pm.Gamma.dist(**l_med_params)),
    ("α", pm.Gamma.dist(**alpha_params)),
    ("ℓ_trend", pm.Gamma.dist(**l_trend_params)),
    ("ℓ_noise", pm.Gamma.dist(**l_noise_params)),
]

colors = brewer["Paired"][7]

p = figure(
    title="Lengthscale and period priors",
    plot_width=550,
    plot_height=350,
    x_range=(-1, 8),
    y_range=(0, 2),
)
p.yaxis.axis_label = "Probability"
p.xaxis.axis_label = "Years"

for i, prior in enumerate(priors):
    p.line(
        x,
        np.exp(prior[1].logp(x).eval()),
        legend_label=prior[0],
        line_width=3,
        line_color=colors[i],
    )
show(p)
```


Lengthscale and period priors
Probability
Years
2
1.5
1
0.5
0
0
2
4
6
8
ℓ_pdecay
ℓ_psmooth
period
ℓ_med
α
ℓ_trend
ℓ_noise

In [114…
```
x = np.linspace(0, 4, 5000)
# Scale Prior Params
n_noise_params = dict(sigma=1.25)
sig_noise_params = dict(sigma=.5)
n_per_params = dict(beta=2)
n_med_params = dict(beta=1.0)
n_trend_params = dict(beta=3)

priors = [
    ("η_per", pm.HalfCauchy.dist(**n_per_params)),
    ("η_med", pm.HalfCauchy.dist(**n_med_params)),
    ("η_trend",pm.HalfCauchy.dist(**n_trend_params)),
    ("σ", pm.HalfNormal.dist(**sig_noise_params)),
    ("η_noise", pm.HalfNormal.dist(**n_noise_params)),
]

colors = brewer["Paired"][5]

p = figure(title="Scale priors", plot_width=550, plot_height=350)
p.yaxis.axis_label = "Probability"
p.xaxis.axis_label = "Years"

for i, prior in enumerate(priors):
    p.line(
        x,
        np.exp(prior[1].logp(x).eval()),
        legend_label=prior[0],
        line_width=3,
        line_color=colors[i],
    )
show(p)
```



Lengthscale Hyperparameters

- ℓ_pdecay: The periodic decay. The smaller this parameter is, the faster the periodicity goes away. We doubt the annual cycle of hiring is going to decrease so we set this to be a large value. The mass of this parameter's probability is between 50 to 150 years out.
- ℓ_psmooth: The smoothness of the periodic component. It controls how "sinusoidal" the periodicity is. We know that because hiring spikes in the first half of the year, plateaus during the summer, and then tends to drop twice (one for thanksgiving and one for new years/holidays), that means this cycle isn't very sinusoidal and should be allowed a few kinks in it's shape. We use a Gamma whose mode is between 1 and 2 years, and has a slightly larger variance, with most of the prior mass from around 0.5 and 2 years.
- period: The period. We put a very strong prior on p and the period that is centered at one. R+W fix p=1, since the hiring cycle is annual.
- ℓ_med: This is the lengthscale for the short to medium long variations. We've decided that our timescale for "medium" should really vary between <1 year and >2 years, keeping the mode at 1 year. This should take into consideration the changes that occur as reaction to economic trends beyond Colorado's control (i.e. the pandemic's effect on hiring or low unemployment figures prior to that). A longer interval than this might miss some of these reactions which is why we keep our prior

in this range.

- α: This is the shape parameter. We keep the mode of this prior below 1 year and increased it's variance relative to a few other priors in order to allow our "medium" term variation to react more strongly to these changes we discussed in ℓ_med.
- ℓ_trend: The lengthscale of the long term trend. This is where we attempt to account for what economists call the "Business Cycle" where macroeconomic trends go up and down usually over a period of 7 to 12 years. Our mode for this prior rests at ~9.5 years and has the widest variance of all our priors.
- ℓ_noise: The lengthscale of the noise covariance. This noise should be very rapid, in the scale of several months rather than several years.

Scale Hyperparameters

For scale hyperparameters, we are using uninformed priors where we pick either HalfCauchy or HalfNormal distributions since they need to stay strictly positive values (no negative time frames). We tend to push these priors towards zero, with the σ and η _noise having the lowest impact on our data and other hyperparameters like the scale of the yearly trend having more impact on our data.

- η_per: Scale of the periodic or seasonal component.
- η_med: Scale of the short to medium term component.
- η_trend: Scale of the long term trend.
- σ: Scale of the white noise.
- η_noise: Scale of correlated, short term noise.

In [115…

```
# pull out normalized data
t = data_pre["t"].values[:, None]
y = data_pre["y_n"].values
```

In [116…

```
with pm.Model() as model:
    # yearly periodic component x long term trend
    η_per = pm.HalfCauchy("η_per", **n_per_params)
    ℓ_pdecay = pm.Gamma("ℓ_pdecay", **decay_params)
    period = pm.Normal("period", **period_params)
    ℓ_psmooth = pm.Gamma("ℓ_psmooth ",**psmooth_priors)
    cov_seasonal = (
        η_per ** 2 * pm.gp.cov.Periodic(1, period, ℓ_psmooth) * pm.gp.cov.Ma
    )
    gp_seasonal = pm.gp.Marginal(cov_func=cov_seasonal)

    # small/medium term irregularities
    η_med = pm.HalfCauchy("η_med", **n_med_params)
    ℓ_med = pm.Gamma("ℓ_med", **l_med_params)
    α = pm.Gamma("α", **alpha_params)
    cov_medium = η_med ** 2 * pm.gp.cov.RatQuad(1, ℓ_med, α)
    gp_medium = pm.gp.Marginal(cov_func=cov_medium)

    # long term trend
    η_trend = pm.HalfCauchy("η_trend", **n_trend_params)
    ℓ_trend = pm.Gamma("ℓ_trend", **l_trend_params)
    cov_trend = η_trend ** 2 * pm.gp.cov.ExpQuad(1, ℓ_trend)
    gp_trend = pm.gp.Marginal(cov_func=cov_trend)

    # noise model
    η_noise = pm.HalfNormal("η_noise", **n_noise_params)
    ℓ_noise = pm.Gamma("ℓ_noise",**l_noise_params)
    σ = pm.HalfNormal("σ", **sig_noise_params)
    cov_noise = η_noise ** 2 * pm.gp.cov.Matern32(1, ℓ_noise) + pm.gp.cov.Wh

    # The Gaussian process is a sum of these three components
    gp = gp_seasonal + gp_medium + gp_trend

    # Since the normal noise model and the GP are conjugates, we use `Margir
    y_ = gp.marginal_likelihood("y", X=t, y=y, noise=cov_noise)

    # this line calls an optimizer to find the MAP
    mp = pm.find_MAP(model = model, include_transformed=True)

    #traces for plotting
    trace = pm.sample(1000,target_accept=0.9)
```

100.00% [72/72 00:00<00:00 logp = -108.8, ||grad|| = 0.11341]

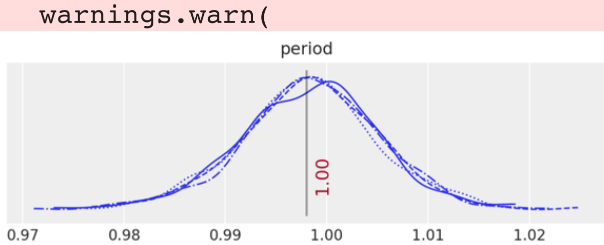
In [117…

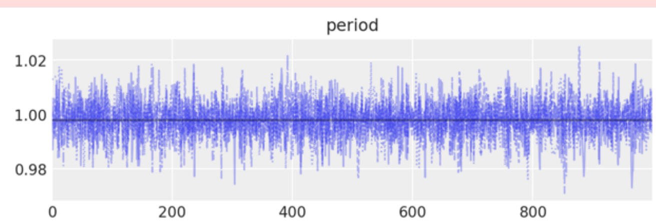
```
# display point estimates for each parameter
sorted([name + ":" + str(mp[name]) for name in mp.keys() if not name.endswit
```

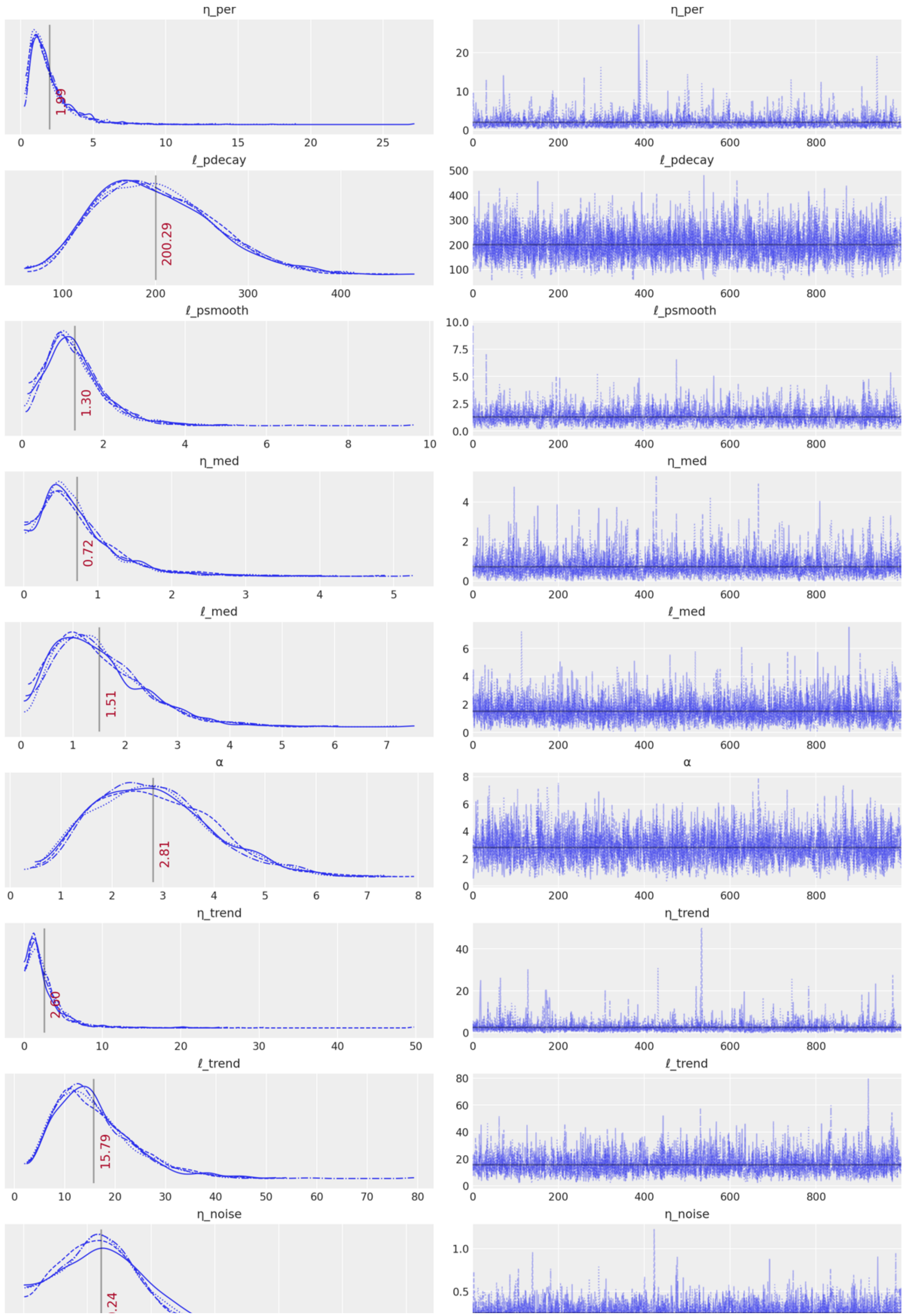
Out[117]:
```
['period:0.9985305090216079',
 'α:2.1055878214217643',
 'η_med:0.3641518848633629',
 'η_noise:0.20386096535551393',
 'η_per:0.7663557156823079',
 'η_trend:1.4491724701798772',
 'σ:0.4948949074570984',
 'ℓ_med:1.0384771256589014',
 'ℓ_noise:0.2573462600797193',
 'ℓ_pdecay:180.29712743415232',
 'ℓ_psmooth :0.6801484120996655',
 'ℓ_trend:9.313645716114364']
```

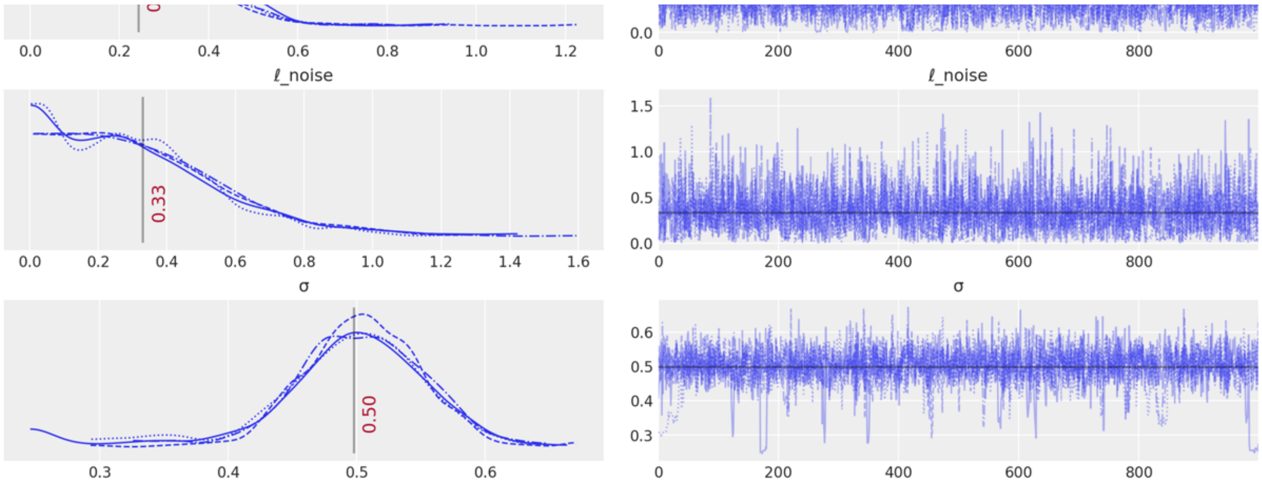
In [118…
```python
# Plot our Posterior Estimates
plot_traces(trace)
```

```
Got error No model on context stack. trying to find log_likelihood in transl
ation.
/Users/will/opt/anaconda3/lib/python3.9/site-packages/arviz/data/io_pymc3_3x
.py:98: FutureWarning: Using `from_pymc3` without the model will be deprecat
ed in a future release. Not using the model will return less accurate and le
ss useful results. Make sure you use the model argument or call from_pymc3 w
ithin a model context.
  warnings.warn(
/var/folders/1s/mg3mq3l17fv3552kc0knnqmc0000gn/T/ipykernel_84824/2978291196.
py:48: DeprecationWarning: The function `traceplot` from PyMC3 is just an al
ias for `plot_trace` from ArviZ. Please switch to `pymc3.plot_trace` or `arv
iz.plot_trace`.
  ax = pm.traceplot(traces[-retain:],
Got error No model on context stack. trying to find log_likelihood in transl
ation.
/Users/will/opt/anaconda3/lib/python3.9/site-packages/arviz/data/io_pymc3_3x
.py:98: FutureWarning: Using `from_pymc3` without the model will be deprecat
ed in a future release. Not using the model will return less accurate and le
ss useful results. Make sure you use the model argument or call from_pymc3 w
ithin a model context.
  warnings.warn(
Got error No model on context stack. trying to find log_likelihood in transl
ation.
Got error No model on context stack. trying to find log_likelihood in transl
ation.
/Users/will/opt/anaconda3/lib/python3.9/site-packages/arviz/data/io_pymc3_3x
.py:98: FutureWarning: Using `from_pymc3` without the model will be deprecat
ed in a future release. Not using the model will return less accurate and le
ss useful results. Make sure you use the model argument or call from_pymc3 w
ithin a model context.
  warnings.warn(
```






η_per
1.99
η_per
ℓ_pdecay
200.29
ℓ_pdecay
ℓ_psmooth
1.30
ℓ_psmooth
η_med
0.72
η_med
ℓ_med
1.51
ℓ_med
α
2.81
α
η_trend
2.60
η_trend
ℓ_trend
15.79
ℓ_trend
η_noise
η_noise



In [119…

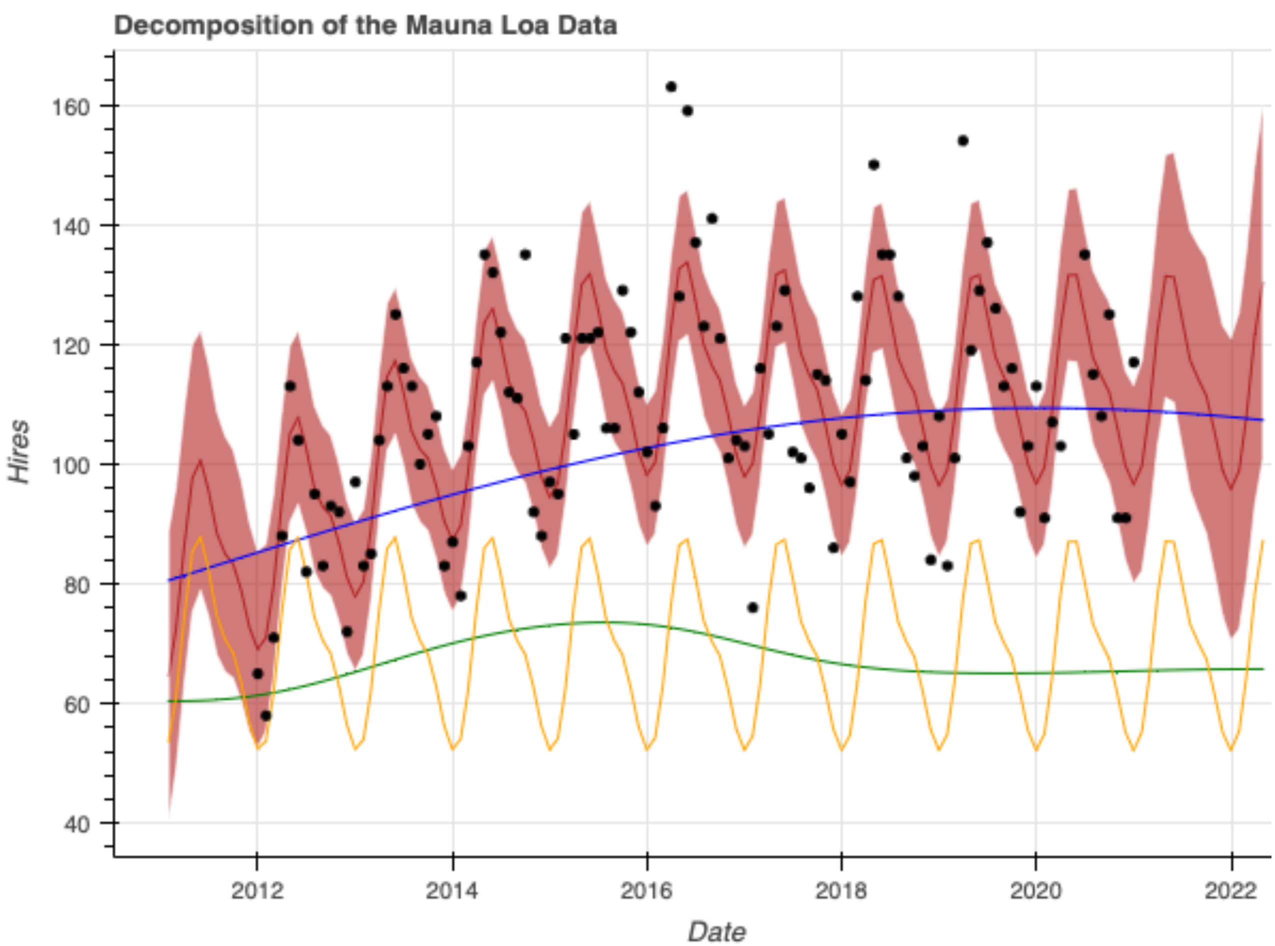
```
# predict at a 1 month granularity
dates = pd.date_range(start="1/1/2011", end="5/1/2022", freq="1M")
tnew = dates_to_idx(dates)[:, None]

print("Predicting with gp ...")
mu, var = gp.predict(tnew, point=mp, diag=True)
mean_pred = mu * std_hires + first_hires
var_pred = var * std_hires ** 2

# make dataframe to store fit results
fit = pd.DataFrame(
    {"t": tnew.flatten(), "mu_total": mean_pred, "sd_total": np.sqrt(var_pre
    index=dates,
)

print("Predicting with gp_trend ...")
mu, var = gp_trend.predict(
    tnew, point=mp, given={"gp": gp, "X": t, "y": y, "noise": cov_noise}, di
)
fit = fit.assign(mu_trend=mu * std_hires + first_hires, sd_trend=np.sqrt(var

print("Predicting with gp_medium ...")
mu, var = gp_medium.predict(
    tnew, point=mp, given={"gp": gp, "X": t, "y": y, "noise": cov_noise}, di
)
fit = fit.assign(mu_medium=mu * std_hires + first_hires, sd_medium=np.sqrt(v

print("Predicting with gp_seasonal ...")
mu, var = gp_seasonal.predict(
    tnew, point=mp, given={"gp": gp, "X": t, "y": y, "noise": cov_noise}, di
)
fit = fit.assign(mu_seasonal=mu * std_hires + first_hires, sd_seasonal=np.sq
print("Done")
```

```
Predicting with gp ...
Predicting with gp_trend ...
Predicting with gp_medium ...
Predicting with gp_seasonal ...
Done
```

In [120…

```
## plot the components
p = figure(
    title="Decomposition of the Mauna Loa Data",
    x_axis_type="datetime",
    plot_width=850,
    plot_height=450,
)
p.yaxis.axis_label = "Hires"
p.xaxis.axis_label = "Date"

# plot mean and 3σ region of total prediction
upper = fit.mu_total + 3 * fit.sd_total
lower = fit.mu_total - 3 * fit.sd_total
band_x = np.append(fit.index.values, fit.index.values[::-1])
band_y = np.append(lower, upper[::-1])

# total fit
p.line(
    fit.index,
    fit.mu_total,
    line_width=1,
    line_color="firebrick",
    legend_label="Total fit",
)
p.patch(band_x, band_y, color="firebrick", alpha=0.6, line_color="white")

# trend
p.line(
    fit.index,
    fit.mu_trend,
    line_width=1,
    line_color="blue",
    legend_label="Long term trend",
)

# medium
p.line(
    fit.index,
    fit.mu_medium,
    line_width=1,
    line_color="green",
    legend_label="Medium range variation",
)

# seasonal
p.line(
    fit.index,
```

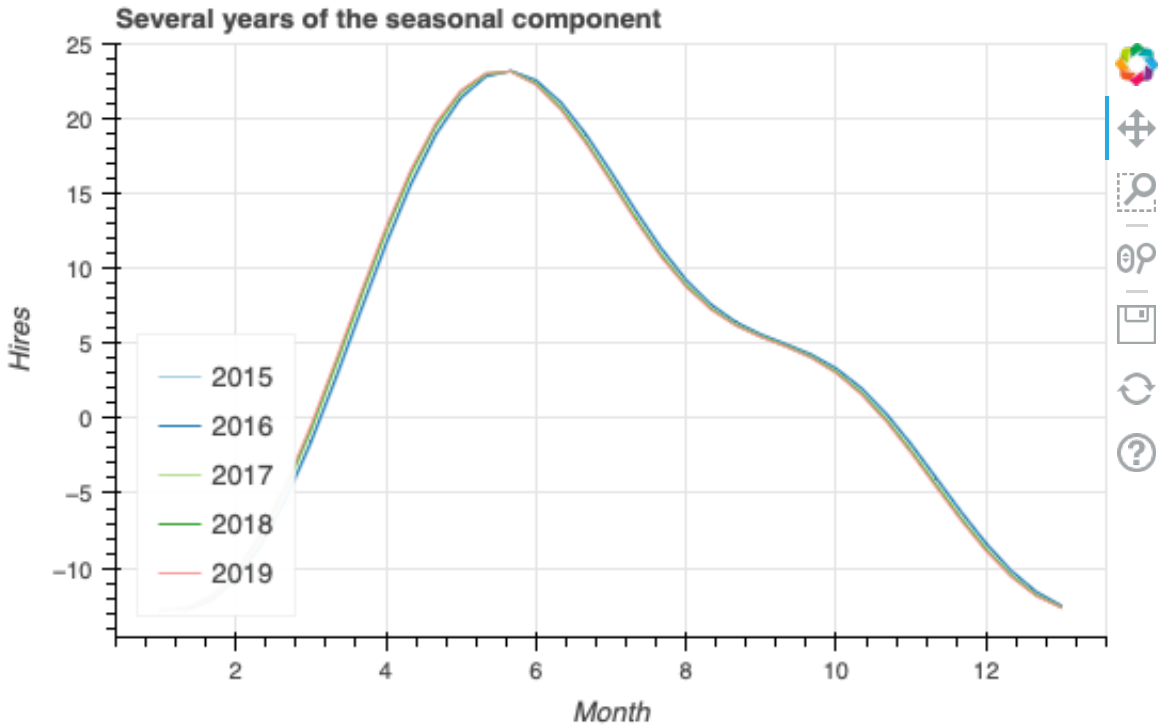
```
    fit.mu_seasonal,
    line_width=1,
    line_color="orange",
    legend_label="Seasonal process",
)
predict_region = BoxAnnotation(
    left=pd.to_datetime("2021-01-01"), fill_alpha=0.1, fill_color="firebrick
)
# true value
p.circle(data_pre.index, data_pre["Hires"], color="black", legend_label="Obs
p.add_layout(p.legend[0], 'right')
show(p)
```



In [121…

```
# plot several years

p = figure(title="Several years of the seasonal component", plot_width=550,
p.yaxis.axis_label = "Hires"
p.xaxis.axis_label = "Month"

colors = brewer["Paired"][5]
years = ["2015", "2016", "2017", "2018", "2019"]

for i, year in enumerate(years):
    dates = pd.date_range(start="1/1/" + year, end="12/31/" + year, freq="10
    tnew = dates_to_idx(dates)[:, None]

    print("Predicting year", year)
    mu, var = gp_seasonal.predict(
        tnew, point=mp, diag=True, given={"gp": gp, "X": t, "y": y, "noise":
    )
    mu_pred = mu * std_hires

    # plot mean
    x = np.asarray((dates - dates[0]) / pd.Timedelta(30, "D")) + 1
    p.line(x, mu_pred, line_width=1, line_color=colors[i], legend_label=year

p.legend.location = "bottom_left"
show(p)
```

```
Predicting year 2015
Predicting year 2016
Predicting year 2017
Predicting year 2018
Predicting year 2019
```



Industry Experience Informative Priors:

The above represents what industry prior knowledge we bring to this analysis (prior work in job market data). This curve reflects tha anual hiring lifecycle with a 12 week delay caused by BLS reporting lags. The typical hiring wave happens early in the year with lots of new jobs and jobseekers looking to find work all at once. After this initial spike, it will typically plateau in the summer time through Labor Day before falling around Thanksgiving and the holidays. We'd expect a more closely fitted curve to have two marked decreases in hires made in November and December with a short spike in between the holidays as well, but our smoothing processes we applied to the seasonal GP is likely why we don't see that above.

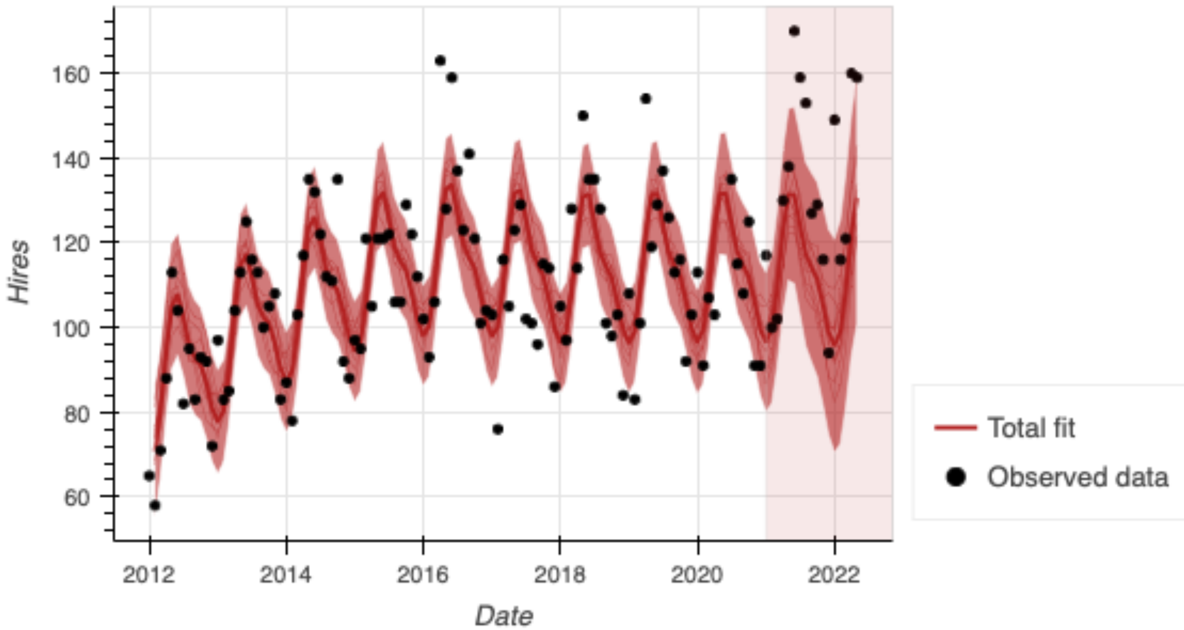
In [122…

```
dates = pd.date_range(start="1/1/2012", end="5/1/2022", freq="1M")
tnew = dates_to_idx(dates)[:, None]

print("Sampling gp predictions ...")
mu_pred, cov_pred = gp.predict(tnew, point=mp)

# draw samples, and rescale
n_samples = 2000
samples = pm.MvNormal.dist(mu=mu_pred, cov=cov_pred, shape=len(mu_pred)).ran
samples = samples * std_hires + first_hires
```

```
Sampling gp predictions ...
```

In [123…

```
### make plot
p = figure(x_axis_type="datetime", plot_width=600, plot_height=300)
p.yaxis.axis_label = "Hires"
p.xaxis.axis_label = "Date"

### plot mean and 2σ region of total prediction
# scale mean and var
mu_pred_sc = mu_pred * std_hires + first_hires
sd_pred_sc = np.sqrt(np.diag(cov_pred) * std_hires ** 2)

upper = mu_pred_sc + 3 * sd_pred_sc
lower = mu_pred_sc - 3 * sd_pred_sc
band_x = np.append(dates, dates[::-1])
band_y = np.append(lower, upper[::-1])

p.line(dates, mu_pred_sc, line_width=2, line_color="firebrick", legend_label
p.patch(band_x, band_y, color="firebrick", alpha=0.6, line_color="white")

# some predictions
idx = np.random.randint(0, samples.shape[0], 10)
p.multi_line(
    [dates] * len(idx),
    [samples[i, :] for i in idx],
    color="firebrick",
    alpha=0.5,
    line_width=0.5,
)
# true value
p.circle(data_monthly.index, data_monthly["Hires"], color="black", legend_la

ppm400 = Span(
    location=400,
    dimension="width",
    line_color="black",
    line_dash="dashed",
    line_width=1,
)
predict_region = BoxAnnotation(
    left=pd.to_datetime("2021-01-01"), fill_alpha=0.1, fill_color="firebrick
)
p.add_layout(predict_region)
p.add_layout(ppm400)
p.add_layout(p.legend[0], 'right')
p.legend.location = "bottom_right"
show(p)
```


Hires
160
140
120
100
80
60
2012
2014
2016
2018
2020
2022
Date
Total fit
Observed data

Takeaways

Given our model's potential range (3 standard deviations from the mean, highlighted in red), we can se that most of our predicted points fall within a reasonable range of our training set. It also appears as though the variance of hiring practices has been increasing over this time period, where the dates after 2017 and up appear to have more and more points outside our predicted range. In future iterations of this, it may be worth implementing a GP term that can account for increasing volatility in the hiring market, assuming we'd want to include that in our model.

Also notice how the range expands the deeper into our test range we get, this is because the further away from our most recent point we estimate, the wider and wider the potential range may be. Towards the end of our test range, the upper and lower bands have expanded to 100k to 160k hires per month, more than twice the size of the variability earlier in the model.

Performance Metrics

To quanitfy our model's performance, we'll observe the root mean squared error (RMSE) and the Mean Absolute Percentage Error (MAPE) below. The former can give us in plain terms how far off our predictions typically are in terms of hires in thousands, so an RMSE of 20 means that our model might typically be off by about 20k hires per month. This metric is typically used for regression models when testing on unseen data. The latter metric, MAPE, is more typically used for time series forecasting, and differs from RMSE in that it gives us error on a percentage scale rather than in unit terms. So a MAPE of 10% would mean that our model is typically off by ±10% of our actual value.

Below, we'll measure the RMSE and the MAPE for both our pre-intervention time period and our post-intervention time period, and note the differences.

In [124…
```
# Sum Total Error
# pre RMSE and MAPE
pred_df = fit[(fit.index<=pd.to_datetime('2021-01-01'))
              & (fit.index>=pd.to_datetime('2011-12-31'))
              & (fit.index.isin([pd.to_datetime('2020-04-30'),pd.to_datetime
pre_pred = pred_df.reset_index()['mu_total']
pre_act = data_monthly[(data_monthly.index.isin(drop_dates)==False)
                       & (data_monthly.index<=pd.to_datetime('2021-01-01'))]
pre_sum_of_squares = ((pre_pred- pre_act)**2).sum()
pre_sum_of_error = (pre_act - pre_pred).sum()
pre_MSE = pre_sum_of_squares/len(pre_pred)
pre_RMSE = np.sqrt(pre_MSE)
pre_RMSE
```

Out[124]: 12.150934709972855

In [125…
```
# MAPE = Mean Absolute Percentage Error
(np.abs((pre_pred - pre_act)/pre_act)).sum()/len(pre_pred)
```

Out[125]: 0.09036824032607339

In [126…
```
pre_sum_of_error
```

Out[126]: 11.473183191370069

In [127…
```
# post RMSE and MAPE

post_df = fit[(fit.index>pd.to_datetime('2021-01-01'))]
post_pred = post_df.reset_index()['mu_total']
post_act = data_monthly[data_monthly.index>pd.to_datetime('2021-01-01')].res
post_sum_of_squares = ((post_act- post_pred)**2).sum()
post_sum_of_error = (post_act - post_pred).sum()
post_MSE = post_sum_of_squares/len(post_pred)
post_RMSE = np.sqrt(post_MSE)
post_RMSE
```

Out[127]: 25.38532075966743

In [128…
```
# MAPE = Mean Absolute Percentage Error
(np.abs(( post_pred - post_act)/post_act)).sum()/len(post_pred)
```

Out[128]: 0.14170218608893065

In [129…
```
post_sum_of_error
```

Out[129]: 302.6929174804121

In [130…
```
post_sum_of_error/pre_sum_of_error*100
```

Out[130]:
```
2638.264485379197
```

Model Accuracy falls after intervention

the RMSE more than doubled from 12k hires to 25k hires before and after, and the MAPE values increased from 9% to 14%. Something else that's interesting to look at is the sum of errors from both periods. If our model was tuned correctly, we should hope that the sum of errors is nearer to zero in our training period as we should ideally have errors both above and below our regression line. In the test period, however, we see that the sum of errors is much further from zero and resoundingly positive, meaning our model is typically under estimating what actually happened.

This much alone is not enough to suggest the new law requiring companies to advertise salaries for open positions actually caused the increase in hires above our expectations. It's possible that something on a macroeconomic scale could be influencing Colorado's hiring market overall that has more to do with the greater US hiring market than Colorado individually.

To more closely analyze this, we'll run a similar Gaussian Process regression model on a separate state that did not have a similar law put in place in the same time frame. This new state we'll use as a natural control group to see how much of the boost in hires post-intervention was actually common across other states.

Control State: Washington

Using the same method and the same priors, let's look at another state that we've identified as a logical control group to compare Coloardo to: Washington State.

The reasons we believe these two states are comparable are as follows:

- Similar Economic Size (GDP Per Capita: WA = $ 86,770 CO = $ 72,653)
- Similar Unemployment Rates (WA = 3.8% CO = 3.4%)
- Similar Population Size(WA = 7.7M, CO = 5.8M)
- Similar Politics (Liberal policies. Washington State has since passed a similar law that has not taken effect).
- Similar Capital Cities (Tech and Manufacturing Industry exposure, majority state density).
- Similar Climate (Evergreen Mountain Forest and plains. Washington has a coast line).

Relative to other states we could compare Colorado to, we believe this one makes the best candidate for a state with similar pre-law conditions but with no law passed reflected in our data.

In [131…

```
# *********** Washington Control Group   ***************
wa_data_monthly = clean_data('BLS_WA2_Hires_Nonfarm_2012_to_2022_notSA.csv')
wa_data_monthly = wa_data_monthly.set_index('Date')
wa_data_monthly = wa_data_monthly[wa_data_monthly.index.isin(drop_dates)==Fa
t = dates_to_idx(wa_data_monthly.index)

# normalize Hire levels
y = wa_data_monthly["Hires"].values
first_hires = y[0]
std_hires = np.std(y)
y_n = (y - first_hires) / std_hires

wa_data_monthly = wa_data_monthly.assign(t=t)
wa_data_monthly = wa_data_monthly.assign(y_n=y_n)

## split into training and test set
sep_idx = wa_data_monthly.index.searchsorted(pd.to_datetime("2021-01-01"))
data_pre = wa_data_monthly.iloc[: sep_idx + 1, :]
data_post = wa_data_monthly.iloc[sep_idx:, :]

# pull out normalized data
t = data_pre["t"].values[:, None]
y = data_pre["y_n"].values
```

```
with pm.Model() as model:
    # yearly periodic component x long term trend
    η_per = pm.HalfCauchy("η_per", **n_per_params)
    ℓ_pdecay = pm.Gamma("ℓ_pdecay", **decay_params)
    period = pm.Normal("period", **period_params)
    ℓ_psmooth = pm.Gamma("ℓ_psmooth ",alpha=2, beta=.75)
    cov_seasonal = (
        η_per ** 2 * pm.gp.cov.Periodic(1, period, ℓ_psmooth) * pm.gp.cov.Ma
    )
    gp_seasonal = pm.gp.Marginal(cov_func=cov_seasonal)

    # small/medium term irregularities
    η_med = pm.HalfCauchy("η_med", **n_med_params)
    ℓ_med = pm.Gamma("ℓ_med", alpha=3, beta=2)
    α = pm.Gamma("α", alpha=3, beta=6)
    cov_medium = η_med ** 2 * pm.gp.cov.RatQuad(1, ℓ_med, α)
    gp_medium = pm.gp.Marginal(cov_func=cov_medium)

    # long term trend
    η_trend = pm.HalfCauchy("η_trend", **n_trend_params)
    ℓ_trend = pm.Gamma("ℓ_trend", alpha=6, beta=.5)
    cov_trend = η_trend ** 2 * pm.gp.cov.ExpQuad(1, ℓ_trend)
    gp_trend = pm.gp.Marginal(cov_func=cov_trend)

    # noise model
    η_noise = pm.HalfNormal("η_noise", **n_noise_params)
    ℓ_noise = pm.Gamma("ℓ_noise",**l_noise_params)
    σ = pm.HalfNormal("σ", **sig_noise_params)
    cov_noise = η_noise ** 2 * pm.gp.cov.Matern32(1, ℓ_noise) + pm.gp.cov.Wh

    # The Gaussian process is a sum of these three components
    gp = gp_seasonal + gp_medium + gp_trend

    # Since the normal noise model and the GP are conjugates, we use `Margin
    y_ = gp.marginal_likelihood("y", X=t, y=y, noise=cov_noise)

    # this line calls an optimizer to find the MAP
    mp = pm.find_MAP(include_transformed=True)
```

100.00% [41/41 00:00<00:00 logp = -119.72, ||grad|| = 544.06]

In [132…

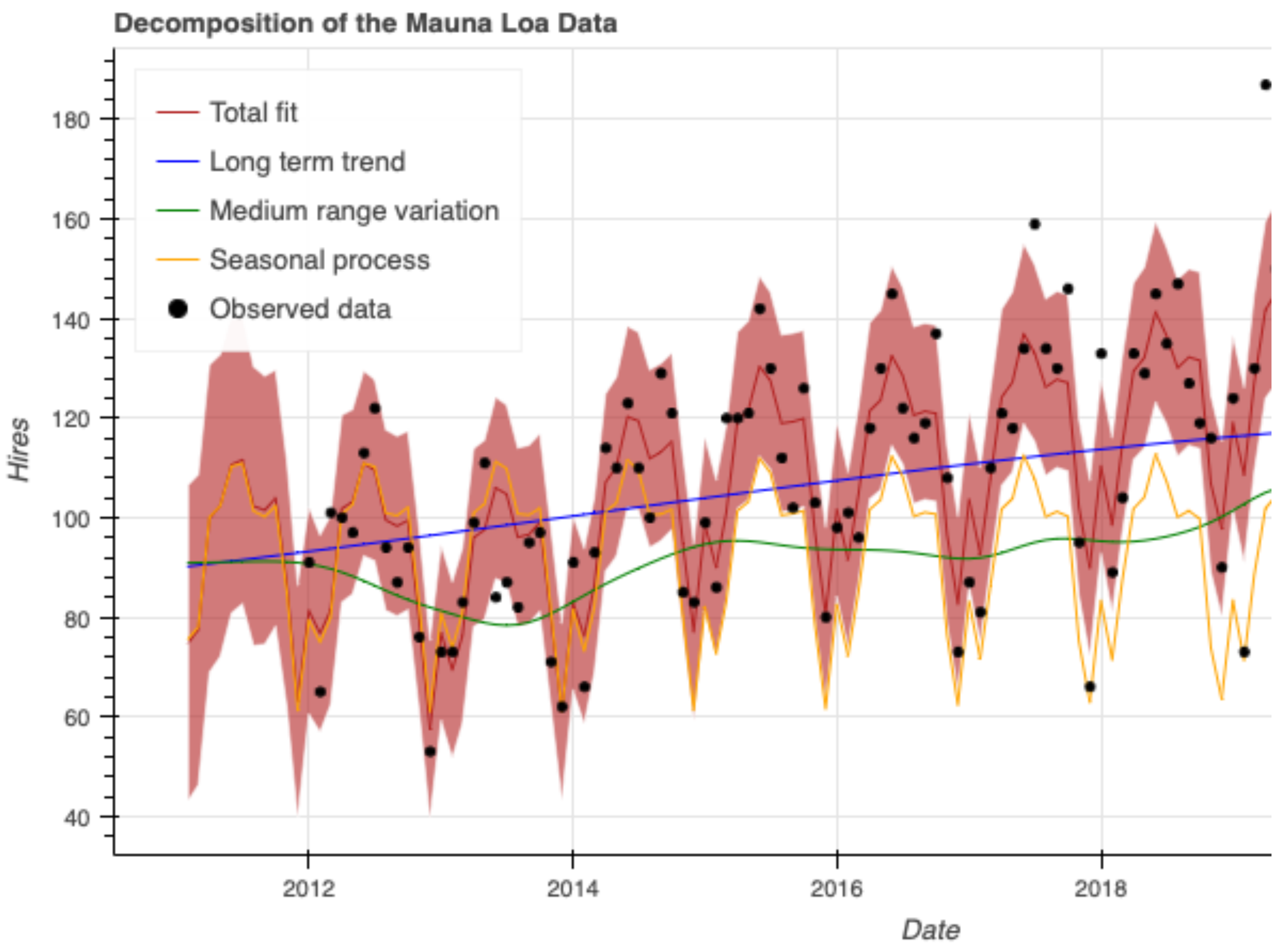
```
# predict at a 1month day granularity
dates = pd.date_range(start="1/1/2011", end="5/1/2022", freq="1M")
tnew = dates_to_idx(dates)[:, None]

print("Predicting with gp ...")
mu, var = gp.predict(tnew, point=mp, diag=True)
mean_pred = mu * std_hires + first_hires
var_pred = var * std_hires ** 2

# make dataframe to store fit results
fit = pd.DataFrame(
    {"t": tnew.flatten(), "mu_total": mean_pred, "sd_total": np.sqrt(var_pre
    index=dates,
)

print("Predicting with gp_trend ...")
mu, var = gp_trend.predict(
    tnew, point=mp, given={"gp": gp, "X": t, "y": y, "noise": cov_noise}, di
)
fit = fit.assign(mu_trend=mu * std_hires + first_hires, sd_trend=np.sqrt(var

print("Predicting with gp_medium ...")
mu, var = gp_medium.predict(
    tnew, point=mp, given={"gp": gp, "X": t, "y": y, "noise": cov_noise}, di
)
fit = fit.assign(mu_medium=mu * std_hires + first_hires, sd_medium=np.sqrt(v

print("Predicting with gp_seasonal ...")
mu, var = gp_seasonal.predict(
    tnew, point=mp, given={"gp": gp, "X": t, "y": y, "noise": cov_noise}, di
)
fit = fit.assign(mu_seasonal=mu * std_hires + first_hires, sd_seasonal=np.sq
print("Done")


# Predict samples
dates = pd.date_range(start="1/1/2012", end="5/1/2022", freq="1M")
tnew = dates_to_idx(dates)[:, None]

print("Sampling gp predictions ...")
mu_pred, cov_pred = gp.predict(tnew, point=mp)

# draw samples, and rescale
n_samples = 2000
samples = pm.MvNormal.dist(mu=mu_pred, cov=cov_pred, shape=len(mu_pred)).ran
samples = samples * std_hires + first_hires
```

```
Predicting with gp ...
Predicting with gp_trend ...
Predicting with gp_medium ...
Predicting with gp_seasonal ...
Done
Sampling gp predictions ...
```

In [133…

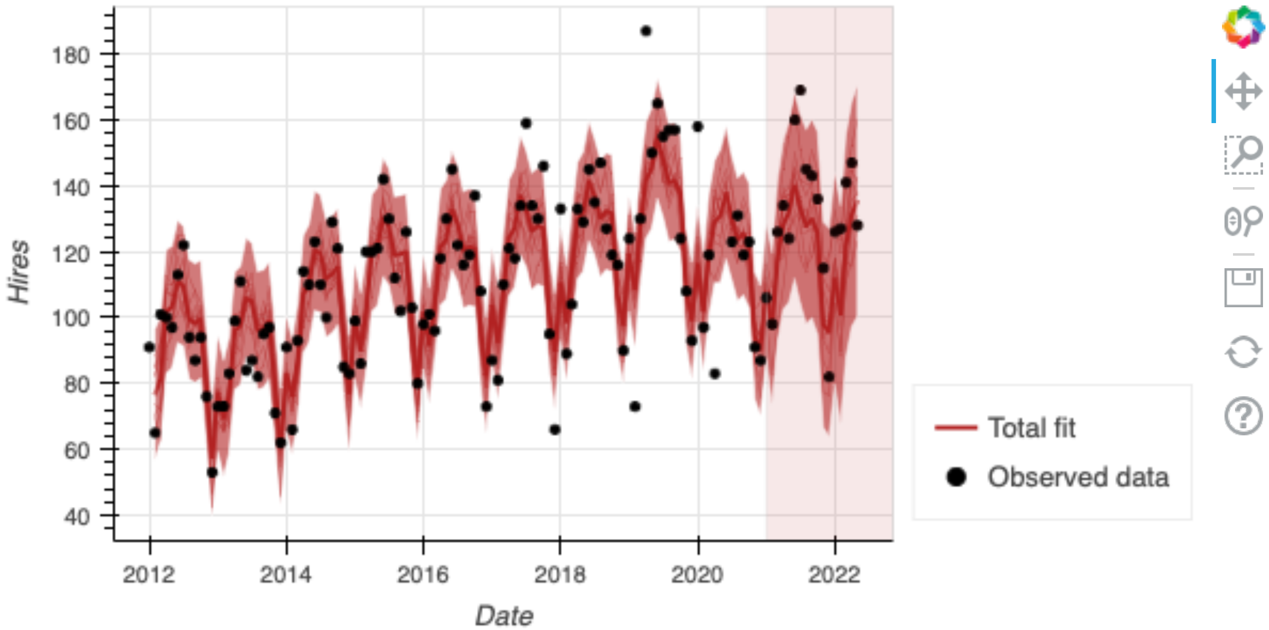
```
## plot the components
p = figure(
    title="Decomposition of the Mauna Loa Data",
    x_axis_type="datetime",
    plot_width=850,
    plot_height=450,
)
p.yaxis.axis_label = "Hires"
p.xaxis.axis_label = "Date"

# plot mean and 3σ region of total prediction
upper = fit.mu_total + 3 * fit.sd_total
lower = fit.mu_total - 3 * fit.sd_total
band_x = np.append(fit.index.values, fit.index.values[::-1])
band_y = np.append(lower, upper[::-1])

# total fit
p.line(
    fit.index,
    fit.mu_total,
    line_width=1,
    line_color="firebrick",
    legend_label="Total fit",
)
p.patch(band_x, band_y, color="firebrick", alpha=0.6, line_color="white")

# trend
p.line(
    fit.index,
    fit.mu_trend,
    line_width=1,
    line_color="blue",
    legend_label="Long term trend",
)

# medium
p.line(
    fit.index,
    fit.mu_medium,
    line_width=1,
    line_color="green",
    legend_label="Medium range variation",
)

# seasonal
p.line(
    fit.index,
    fit.mu_seasonal,
    line_width=1,
    line_color="orange",
    legend_label="Seasonal process",
)
```

```
predict_region = BoxAnnotation(
    left=pd.to_datetime("2021-01-01"), fill_alpha=0.1, fill_color="firebrick
)
p.add_layout(predict_region)
# true value
p.circle(data_pre.index, data_pre["Hires"], color="black", legend_label="Obs
p.legend.location = "top_left"
show(p)
```



In [134…

```
### make plot
p = figure(x_axis_type="datetime", plot_width=600, plot_height=300)
p.yaxis.axis_label = "Hires"
p.xaxis.axis_label = "Date"

### plot mean and 3σ region of total prediction
# scale mean and var
mu_pred_sc = mu_pred * std_hires + first_hires
sd_pred_sc = np.sqrt(np.diag(cov_pred) * std_hires ** 2)

upper = mu_pred_sc + 3 * sd_pred_sc
lower = mu_pred_sc - 3 * sd_pred_sc
band_x = np.append(dates, dates[::-1])
band_y = np.append(lower, upper[::-1])

p.line(dates, mu_pred_sc, line_width=2, line_color="firebrick", legend_label
p.patch(band_x, band_y, color="firebrick", alpha=0.6, line_color="white")

# some predictions
idx = np.random.randint(0, samples.shape[0], 10)
p.multi_line(
    [dates] * len(idx),
    [samples[i, :] for i in idx],
    color="firebrick",
    alpha=0.5,
    line_width=0.5,
)
# true value
p.circle(wa_data_monthly.index, wa_data_monthly["Hires"], color="black", leg

ppm400 = Span(
    location=400,
    dimension="width",
    line_color="black",
    line_dash="dashed",
    line_width=1,
)
predict_region = BoxAnnotation(
    left=pd.to_datetime("2021-01-01"), fill_alpha=0.1, fill_color="firebrick
)
p.add_layout(predict_region)
p.add_layout(ppm400)
p.add_layout(p.legend[0], 'right')


p.legend.location = "bottom_right"
show(p)
```



Washington State Takeaways

Interestingly, Washington State appeared to have a similar trend in the early 2010's from what looks like post-Great_Recession recovery and growth before having Hiring fall in 2020. The annualy hiring cycle looks similar, with a slightly sharper variance around the holidays/end-of-year. It also sees an increase in volatility in hires after 2017, like Colorado did. The numbers of hires in a typical year actually appear similar too, largely due to the similarities in population size between the two states. This helps reinforce our choice of control group.

Below, we'll take another look at RMSE, MAPE, and sum-of-errors as we did for Colorado. If the law had no effect at all, we'd expect see a similar increase in error in our test range as we did in CO.

Performance Metrics

In [135…
```
# Sum Total Error
# pre RMSE and MAPE
pred_df = fit[(fit.index<=pd.to_datetime('2021-01-01'))
              & (fit.index>=pd.to_datetime('2011-12-31'))
              & (fit.index.isin([pd.to_datetime('2020-04-30'),pd.to_datetime
pre_pred = pred_df.reset_index()['mu_total']
pre_act = wa_data_monthly[(wa_data_monthly.index.isin(drop_dates)==False)
                          & (wa_data_monthly.index<=pd.to_datetime('2021-01-01'
pre_sum_of_squares = ((pre_pred- pre_act)**2).sum()
pre_sum_of_error = (pre_act - pre_pred).sum()
pre_MSE = pre_sum_of_squares/len(pre_pred)
pre_RMSE = np.sqrt(pre_MSE)
pre_RMSE
```

Out[135]: 12.64285768762611

In [136…
```
# MAPE = Mean Absolute Percentage Error
(np.abs((pre_pred - pre_act)/pre_act)).sum()/len(pre_pred)
```

Out[136]: 0.08723117070849097

In [137…
```
pre_sum_of_error
```

Out[137]: 7.217443204684223

In [138…
```
# post RMSE and MAPE

post_df = fit[(fit.index>pd.to_datetime('2021-01-01'))]
post_pred = post_df.reset_index()['mu_total']
post_act = wa_data_monthly[wa_data_monthly.index>pd.to_datetime('2021-01-01'
post_sum_of_squares = ((post_act- post_pred)**2).sum()
post_sum_of_error = (post_act - post_pred).sum()
post_MSE = post_sum_of_squares/len(post_pred)
post_RMSE = np.sqrt(post_MSE)
post_RMSE
```

Out[138]: 16.72099344260682

In [139…
```
# MAPE = Mean Absolute Percentage Error
(np.abs(( post_pred - post_act)/post_act)).sum()/len(post_pred)
```

Out[139]: 0.10927395153549926

In [140…
```
172/7.2*100
```

Out[140]: 2388.888888888889

Performance Metric Takeaways

Our Training RMSE was very similar to CO's training RMSE at ~12k hires per month. The training MAPE was lower as well, likely due to the increased variance from the seasonality trend we control for. The training sum of error was about the same as well at net 7k hires per month overall, meaning the model typically under estimates the actual but not by much.

Where our findings are more interesting is in the test period. Again, this is the same post-intervention period but without any intervention this time. In Washington we do see a similar rise in error and that error is usually an underestimation of what actually occured. This can be seen from both the post intervention sum-of-errors being resoundingly positive at 176k net hires. The RMSE and MAPE have both increased as well in this post-intervention period for Washington.

What's interesting is that the *rate* of these increases are significantly lower than what we observed in Colorado. The RMSE for Washington increased by about 32% (from 12k to 16k), while the same metric increased over 108% (from 12k to 25k) in Colorado. Similarly, the sum-of-errors is much higher in Colorado as well by nearly 72% (176k v. 302k). This suggests that while both states underwent a hiring growth period in 2021 and early 2022, Colorado's growth period was more pronounced than Washington State's.

Conclusions

We do not believe there is enough evidence here to confirm that the Salary Transparency law had a beneficial impact on the job market in Colorado. However, given our findings using Gaussian Process time series modeling, we believe there is some evidence to suggest that could be possible and warrants further investigation. Continued research might involve using more states than just Washington to compare Colorado to in order to gain a more complete picture of the macroeconomic forces that affect all states. We could also perform follow up analysis on Washington State once their law has gone into effect in 2023.