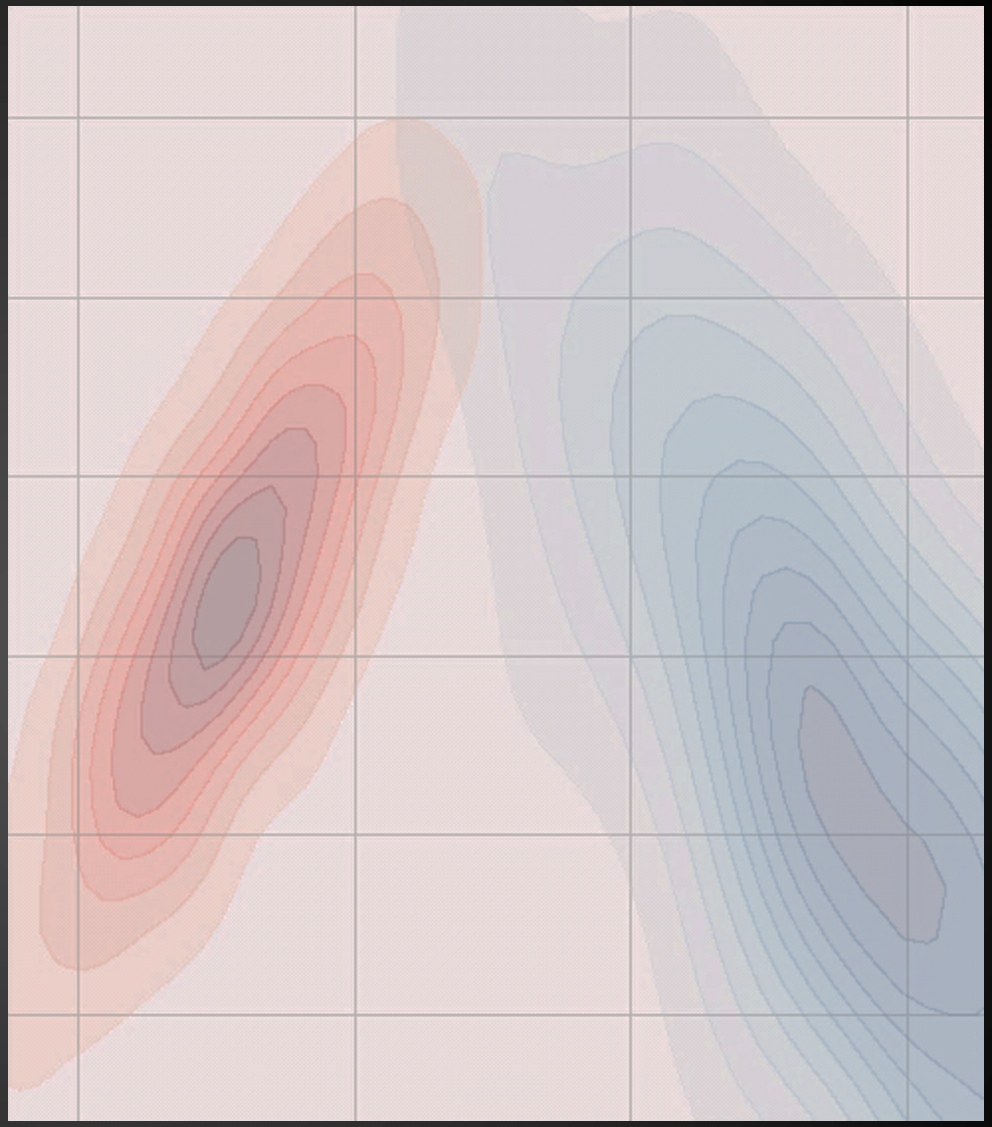
Interrupted Time Series Experiments in Python



Drew Fustin
Automaton Data
drewfustin@gmail.com
@drewfustin

github.com/drewfustin/talks__time-series-experiments

Interrupted Time Series Experiments in Python

- Typical Experimentation in Data Science
- Time Series Experiments / Regression Discontinuity
- Simulating Time Series Data
- ARIMA Algebra and Forecasting with `statsmodels`
- Modeling Interruptions with `PyMC3`

Me

Automaton Data

available for hire!

drew@automatondata.com

PhD, Physics

Data Scientist

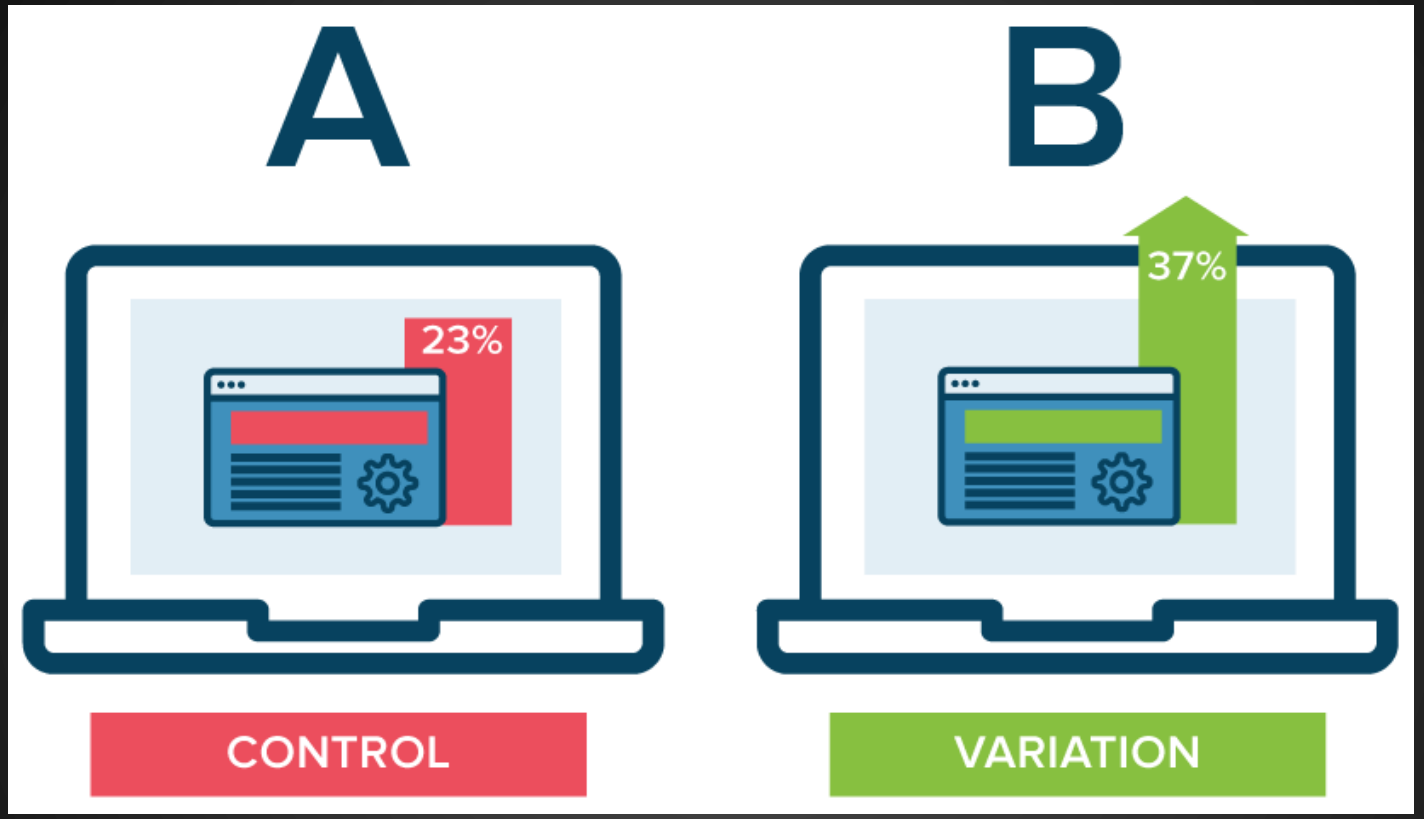
SPOT HERO

GRUBHUB

You?

- ❑ Hypothesis Testing (e.g. A/B Testing)?
- ❑ Time Series Forecasting (e.g. ARIMA)?
- ❑ Probability Distributions (e.g. Poisson)?
- ❑ Markov Chain Monte Carlo (MCMC)?

Typical Experiments



source: Optimizely

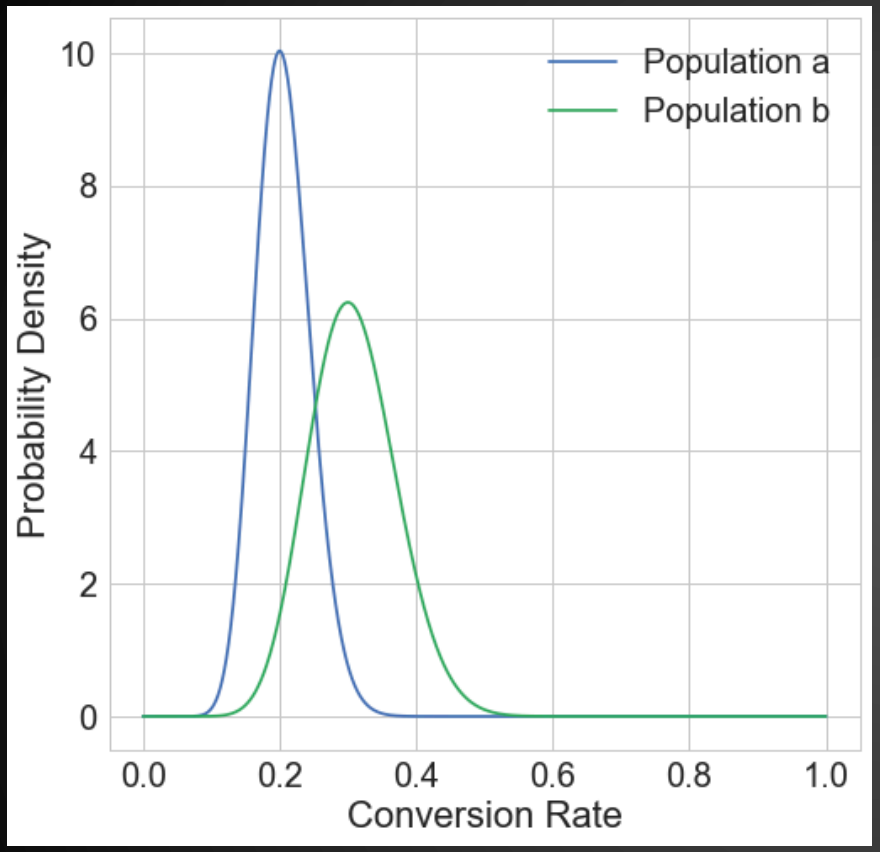
- Select Control/Variant populations (a, b)
- Vary something and measure the change
- Run concurrently to avoid unnecessary variation

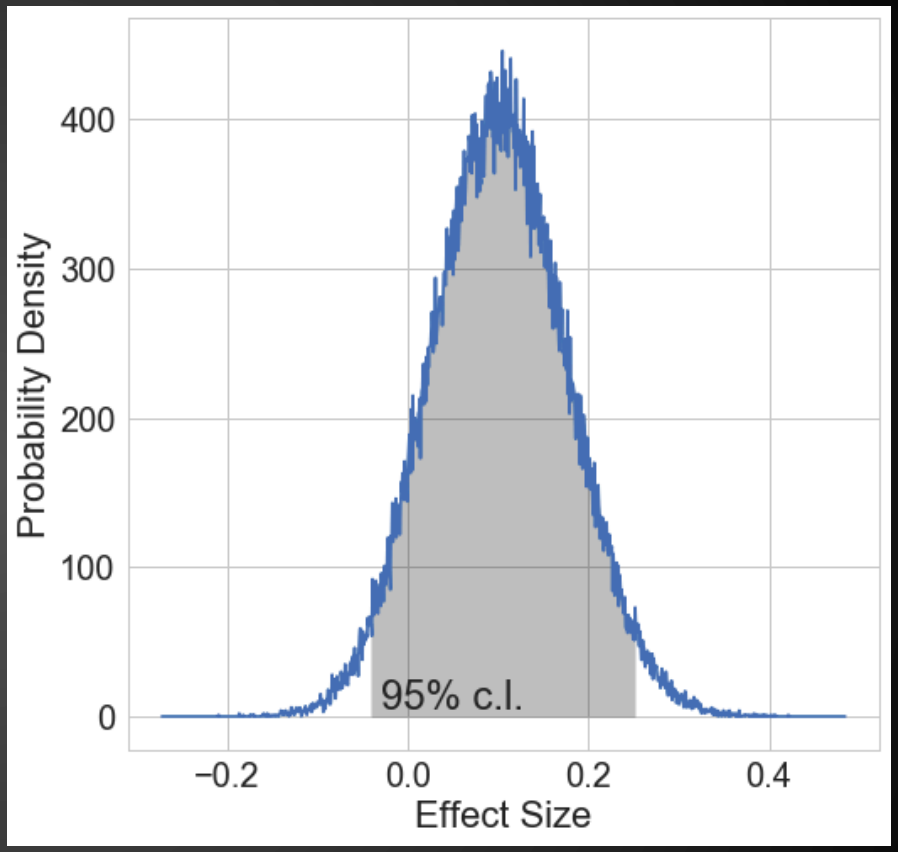
Typical Experiments

e.g. A/B testing conversion rate

```
def conversion_test(successes_a, failures_a, successes_b, failures_b):
    a_distribution = scipy.stats.beta(successes_a + 1, failures_a + 1)
    b_distribution = scipy.stats.beta(successes_b + 1, failures_b + 1)
    effect_size = b_distribution.rvs(10000) - a_distribution.rvs(10000)
    return (effect_size.mean(),
            numpy.percentile(effect_size, (1 - 0.95) / 2 * 100),
            numpy.percentile(effect_size, (1 - (1 - 0.95) / 2) * 100))
```

Typical Experiments



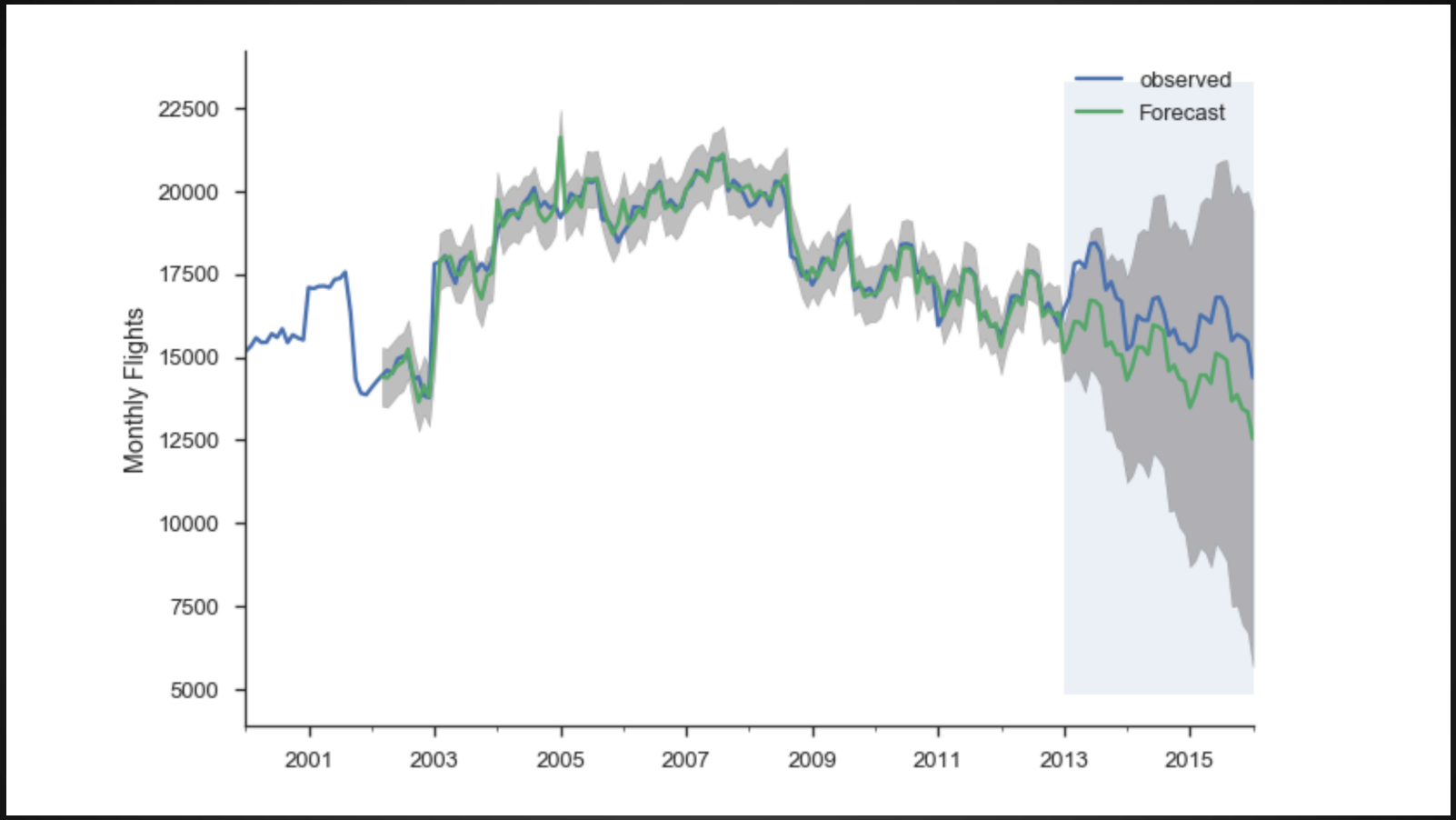


a = 100 observations, 20 successes
b = 50 observations, 15 successes
Effect Size = [-4%, 25%] to 95% confidence

Experimentation is easy!

Experimentation can be hard

Experimentation can be hard

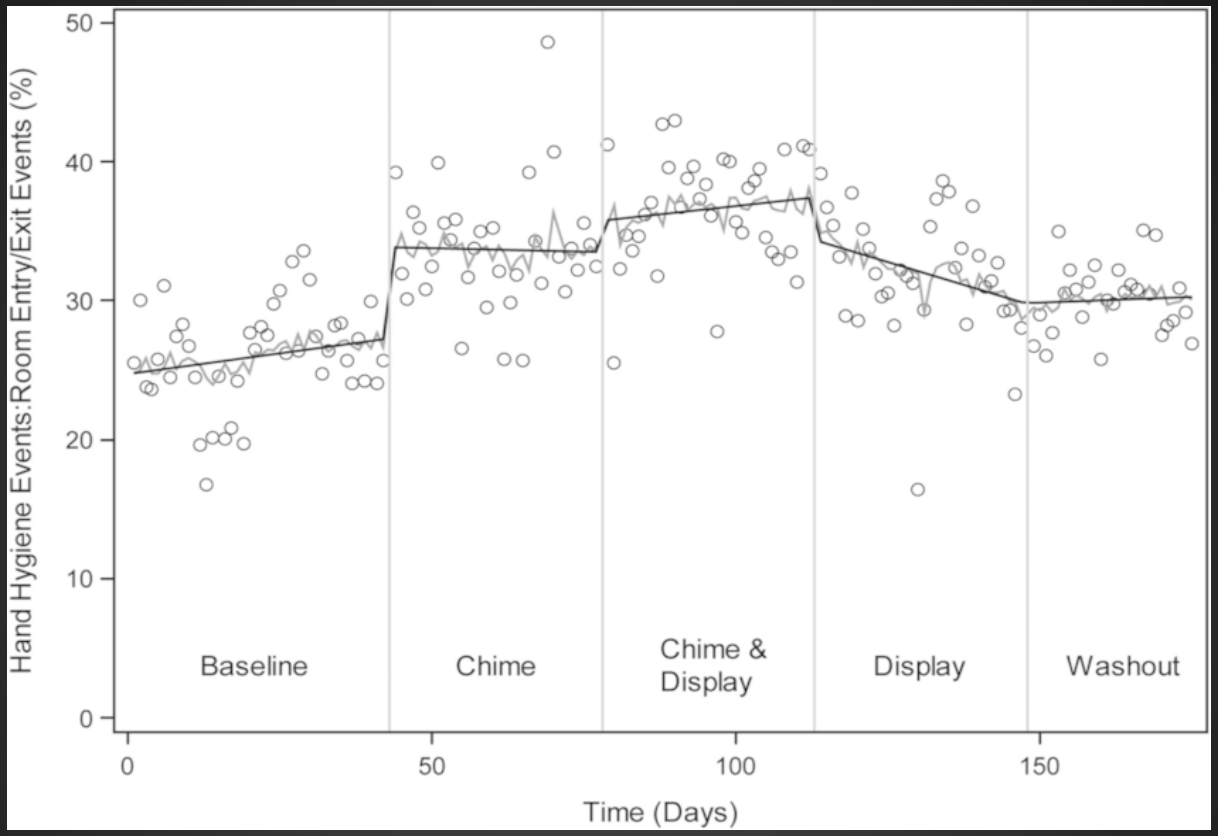


source: Tom Augspurger, tomaugspurger.github.io/modern-7-timeseries

What if you can't experiment on concurrent populations?

Experimentation can be hard

Time Series Experiments

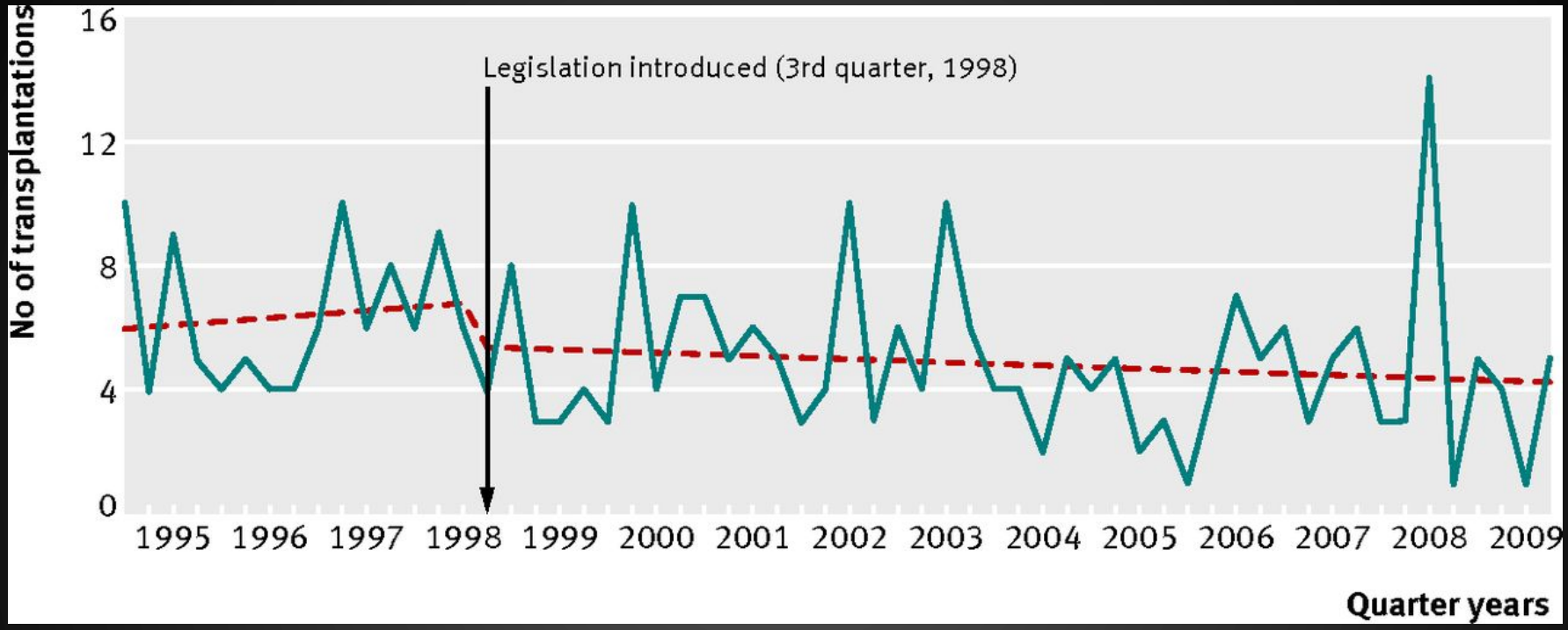


source: Ellison, et al. "A Prospective Controlled Trial of an Electronic Hand Hygiene Reminder System"

- Simply apply variation at a moment in time
- Measure before vs. after

Time Series Experiments

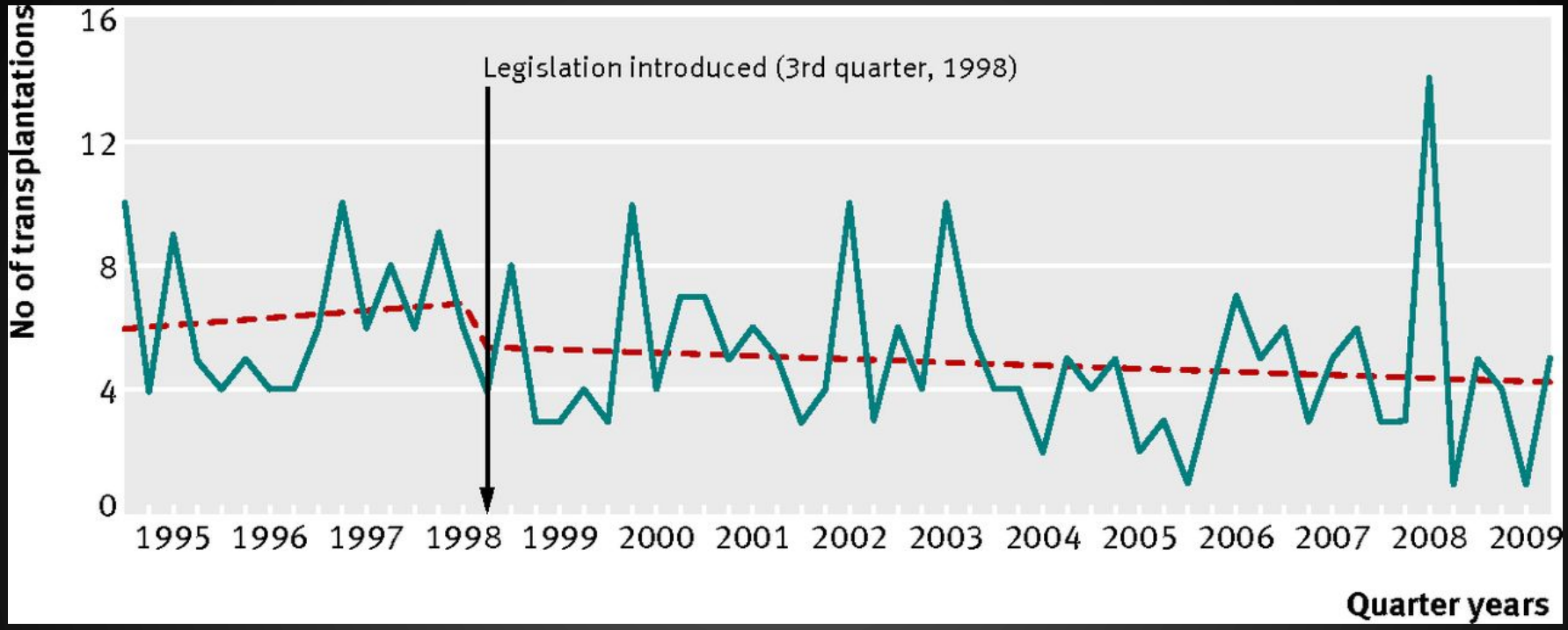
Time Series Experiments



source: Hawton, et al. "Long term effect of reduced pack sizes of paracetamol on poisoning deaths and liver transplant activity in England and Wales: interrupted time series analyses"

- Simply apply variation at a moment in time
- Measure before vs. after

Time Series Experiments



source: Hawton, et al. "Long term effect of reduced pack sizes of paracetamol on poisoning deaths and liver transplant activity in England and Wales: interrupted time series analyses"

- "Simply" apply variation at a moment in time
- "Measure" before vs. after

Time Series Experiments

Let's Give It a Try

Make Some Data

Homogeneous Poisson Point Process

Make Some Data

Homogeneous Poisson Point Process

points distributed randomly in time

Make Some Data

Homogeneous Poisson Point Process

points distributed randomly in time

$$P\left(N(t) = n\right) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$$

$$E\left[N(t)\right] = \lambda t$$

Make Some Data

Homogeneous Poisson Point Process

points distributed randomly in time

$$P\left(N(t) = n\right) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$$

$$E\left[N(t)\right] = \lambda t$$

$$\lambda = \text{constant}$$

Make Some Data

Homogeneous Poisson Point Process

points distributed randomly in time

$$P\left(N(t) = n\right) = \frac{(\lambda t)^n}{n!} e^{-\lambda t}$$

$$E\left[N(t)\right] = \lambda t$$

$$\lambda = \text{constant}$$

note: the time between events that are Poisson distributed follows an Exponential Distribution with rate $1/\lambda$

Make Some Data

```
def hpp(λ, n, t0, freq=None):
    """
    Homogeneous Poisson Point Process

    Parameters
    ----------
    λ : Poisson rate parameter [transactions per day]
    n : number of transactions to produce
    t0 : initial datetime for simulation
    freq : Pandas resample rule for returned time series

    Returns
    -------
    Time series of transactions per freq
    """
    # Times between Poisson-distributed events are exponentially distributed with mean 1 / λ
    transaction_times = pd.TimedeltaIndex(np.cumsum(np.random.exponential(1 / λ, n)), unit='d')
    # Generate non-grouped HPP
    transactions = pd.Series(data=1,
                             index=pd.to_datetime(t0) + transaction_times,
                             name='transactions')
    # Group transactions by freq
    if freq:
        transactions = transactions.resample(freq).sum().fillna(0)[1:-1]

    return transactions
```

Make Some Data

But, events never occur with a constant rate.

Can we let λ vary with time?

Make Some Data

Nonhomogeneous Poisson Point Process

Make Some Data

Nonhomogeneous Poisson Point Process

$$\lambda = \lambda(t)$$

Make Some Data

Nonhomogeneous Poisson Point Process

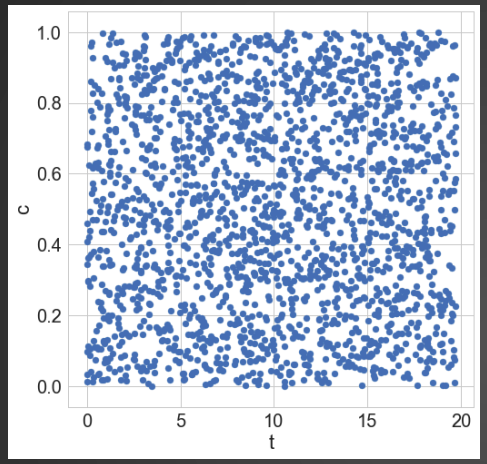
$$\lambda = \lambda(t)$$

Generation:

1. Generate HPP with rate $\lambda_{\mathrm{h}} = \max(\lambda)$
2. Attach a random number c_t bounded by [0, 1] to each point in the HPP process
3. Generate a rate function $r(t)$ bounded by [0, 1] such that $r(t_m) = 1$ for any t_m such that $\lambda(t_m) = \max(\lambda)$
4. Discard any point in the HPP for which $c_t > r(t)$

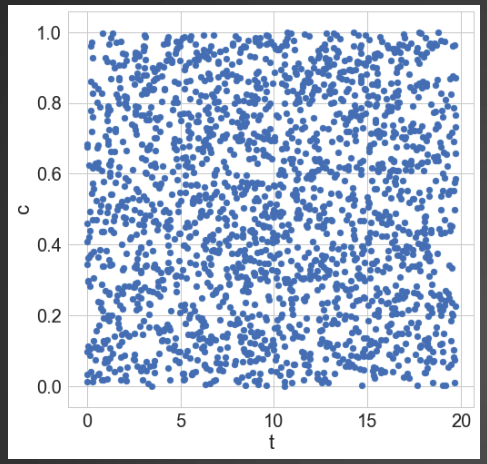
Make Some Data

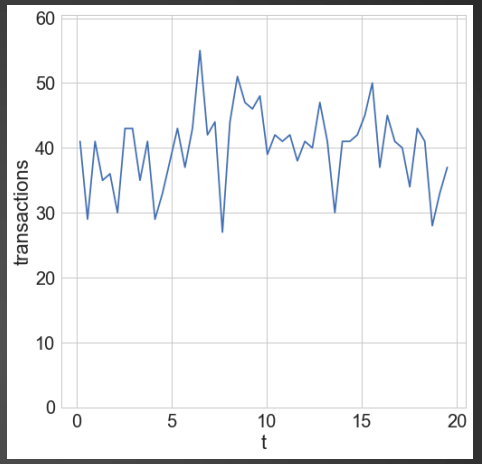
Nonhomogeneous Poisson Point Process



Make Some Data

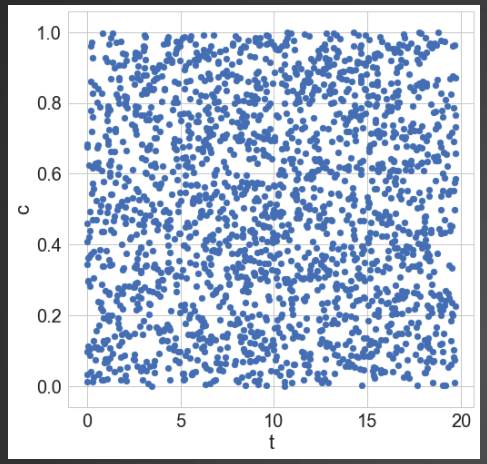
Nonhomogeneous Poisson Point Process

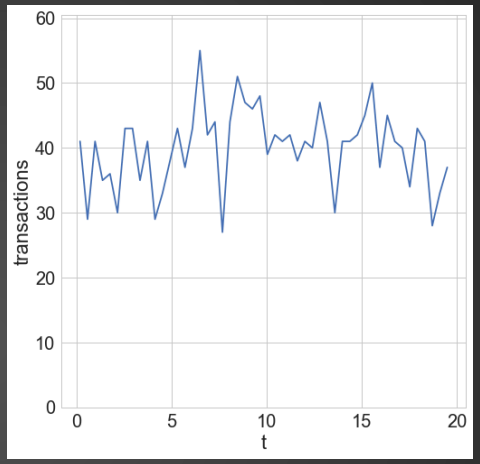


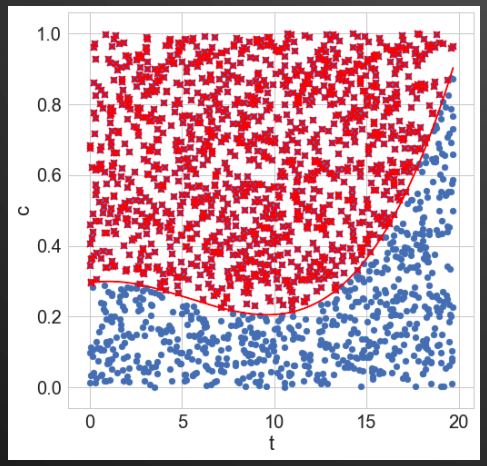


Make Some Data

Nonhomogeneous Poisson Point Process

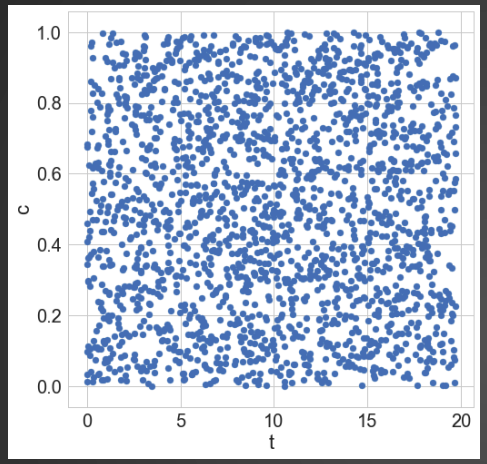


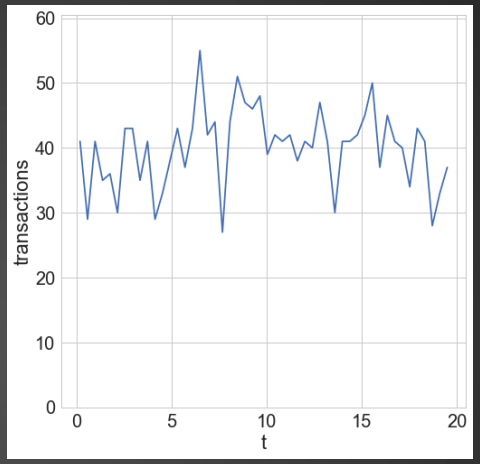


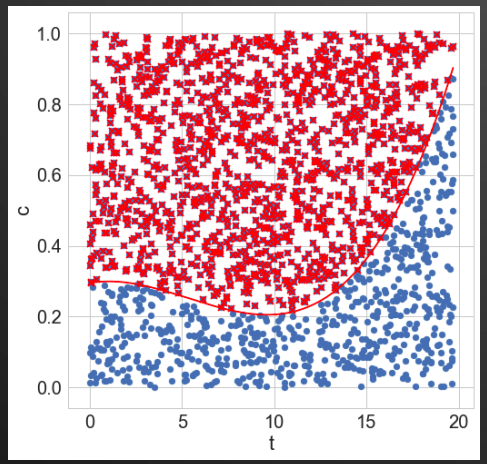


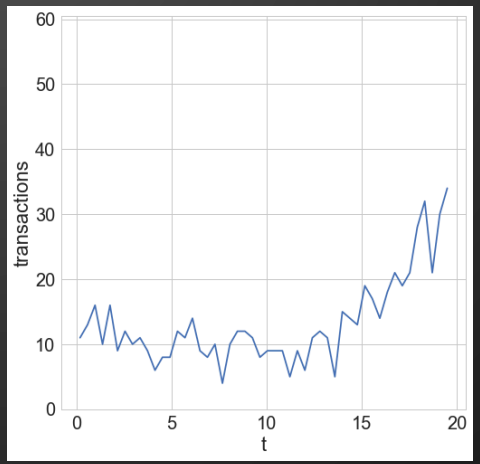
Make Some Data

Nonhomogeneous Poisson Point Process





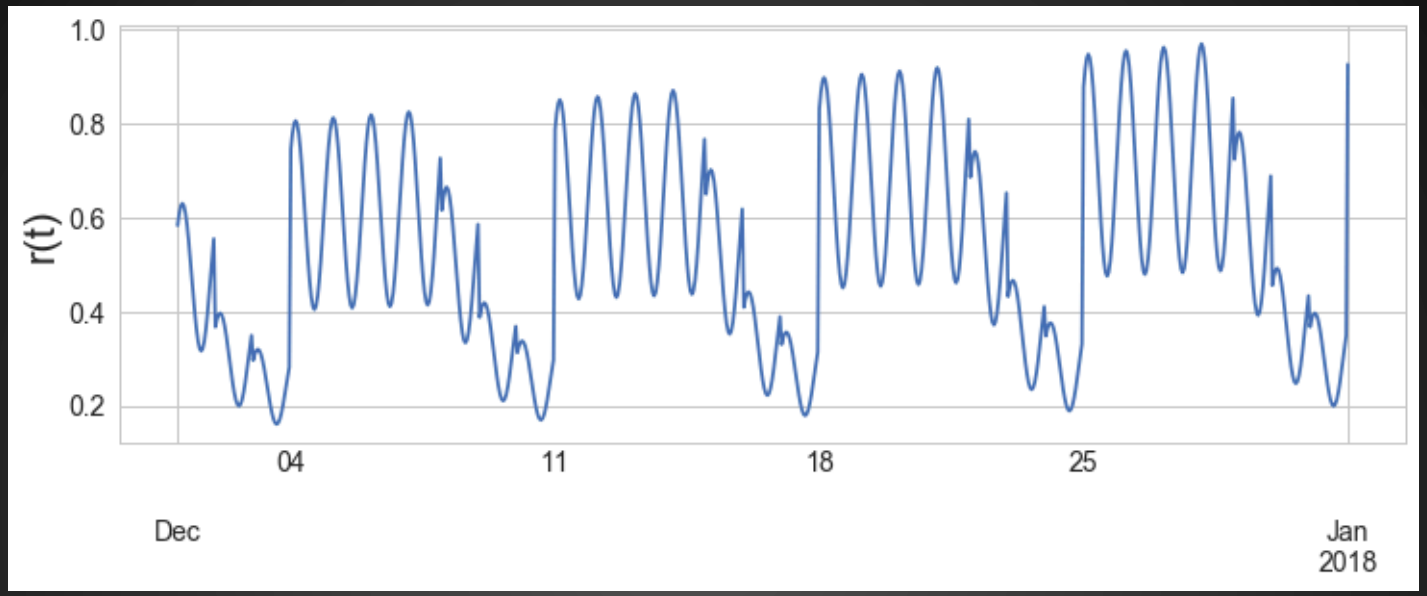


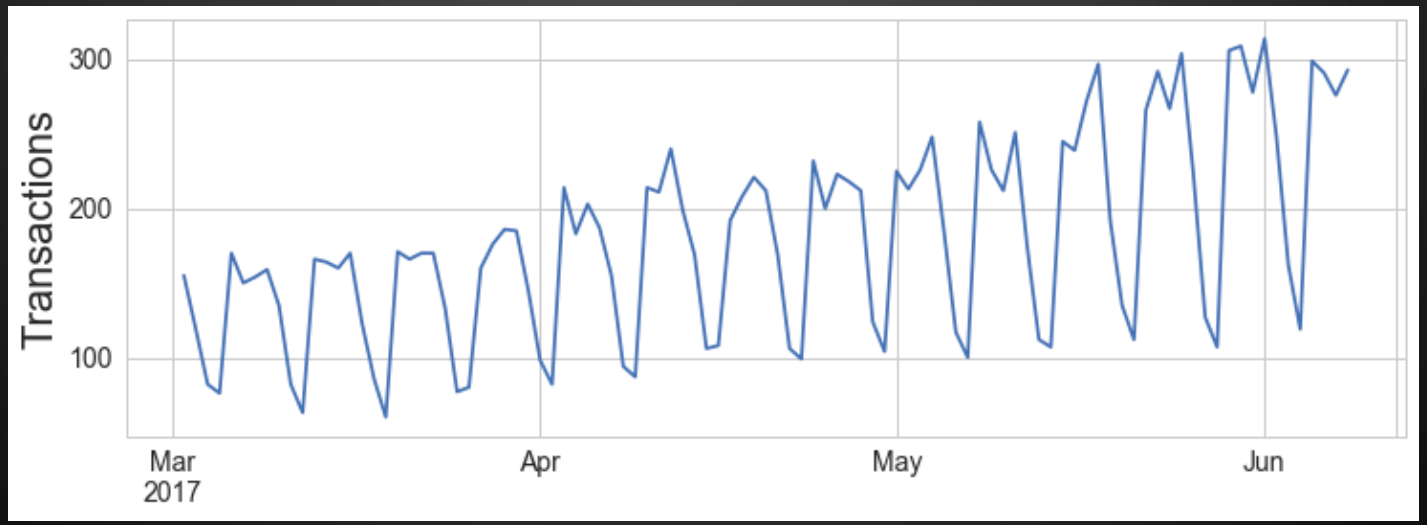


Make Some Data

```
def nhpp(λ, n, t0, freq=None, thinning_components=None):
    """
    Nonhomogeneous Poisson Point Process

    Parameters
    ----------
    λ : Poisson rate parameter [transactions per day] of the bounding Homogeneous Point Process
        Equal to the instantaneous rate of the Nonhomogeneous Point Process when thinning = 1
    n : number of transactions to produce
    t0 : initial datetime for simulation
    freq : Pandas resample rule for returned time series
    thinning_components : components to pass to thinning function

    Returns
    -------
    Time series of transactions per freq
    """
    # Get a non-grouped HPP
    transactions = hpp(λ, n, t0)
    # Thin the HPP
    if thinning_components:
        transactions = transactions[np.random.uniform(size=len(transactions)) <
                                    thinning(t=transactions.index, components=thinning_components)]
    # Group transactions by freq
    if freq:
        transactions = transactions.resample(freq).sum().fillna(0)[1:-1]

    return transactions
```

Make Some Data





Make Some Data

ARIMA Algebra

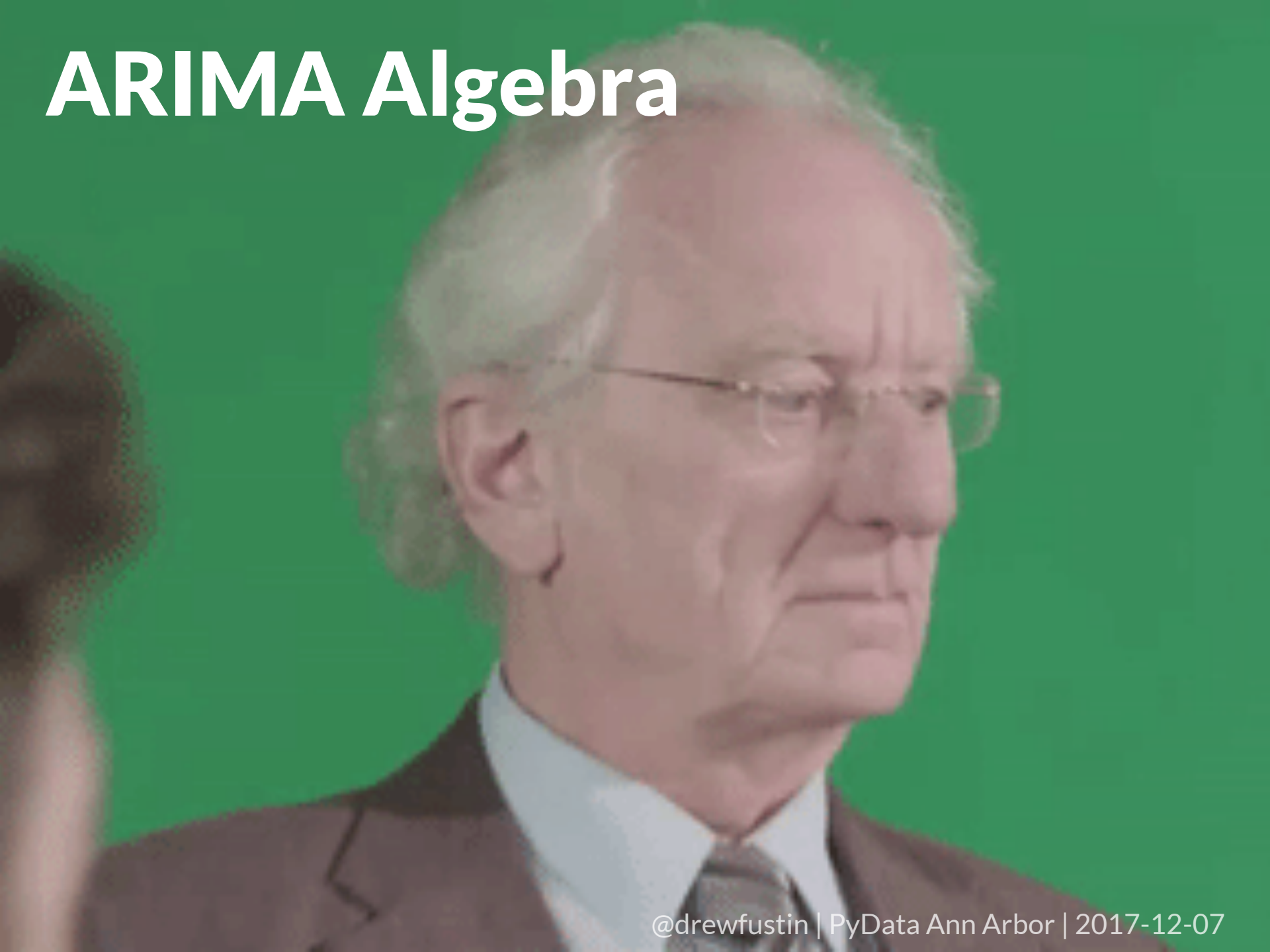
Autoregressive **(AR)** Integrated **(I)** Moving Average **(MA)**

The above can be generalized as follows.

$$\left(1-\sum_{i=1}^{p}\phi_i L^i\right)(1-L)^d X_t = \delta + \left(1+\sum_{i=1}^{q}\theta_i L^i\right)\varepsilon_t.$$

This defines an ARIMA(p,d,q) process with **drift** $\delta/(1 - \Sigma\phi_i)$.

ARIMA Algebra



ARIMA Algebra

$$\text{ARIMA}\,(p, d, q)$$

$$\phi_p(B)\nabla_d(B)X_t = \theta_q(B)\varepsilon_t$$

ARIMA Algebra

$$\text{ARIMA}\,(p, d, q)$$

$$\phi_p(B)\nabla_d(B)X_t = \theta_q(B)\varepsilon_t$$

Autoregressive (AR) [order p]

$$\phi_p\,(B) \equiv \left(1 - \phi_1 B - \phi_2 B^2 - \ldots \phi_p B^p\right)$$

$$\phi_p(B)X_t = \varepsilon_t$$

ARIMA Algebra

$$\text{ARIMA}\,(p, d, q)$$

$$\phi_p(B)\nabla_d(B)X_t = \theta_q(B)\varepsilon_t$$

Autoregressive (AR) [order p]

$$\phi_p\,(B) \equiv \left(1 - \phi_1 B - \phi_2 B^2 - \ldots \phi_p B^p\right)$$

$$\phi_p(B)X_t = \varepsilon_t$$

Integrated (I) [order d]

$$\nabla_d\,(B) \equiv (1 - B)^d$$

$$\nabla_d(B)X_t = \varepsilon_t$$

ARIMA Algebra

$$\text{ARIMA}\,(p, d, q)$$

$$\phi_p(B)\nabla_d(B)X_t = \theta_q(B)\varepsilon_t$$

Autoregressive (AR) [order p]

$$\phi_p\,(B) \equiv \left(1 - \phi_1 B - \phi_2 B^2 - \ldots \phi_p B^p\right)$$

$$\phi_p(B)X_t = \varepsilon_t$$

Integrated (I) [order d]

$$\nabla_d\,(B) \equiv (1 - B)^d$$

$$\nabla_d(B)X_t = \varepsilon_t$$

Moving Average (MA) [order q]

$$\theta_q\,(B) \equiv \left(1 - \theta_1 B - \theta_2 B^2 - \ldots \theta_q B^q\right)$$

$$X_t = \theta_q(B)\varepsilon_t$$

ARIMA Algebra

This generalizes to a Seasonal ARIMA:

$$\text{ARIMA}\,(p,d,q)\,(P,D,Q)_s$$

$$\phi_p(B)\Phi_P(B^s)\nabla_d(B)\nabla_D(B^s)X_t = \theta_q(B)\Theta_Q(B^s)\varepsilon_t$$

ARIMA Algebra

This generalizes to a Seasonal ARIMA:

$$\text{ARIMA}\,(p,d,q)\,(P,D,Q)_s$$

$$\phi_p(B)\Phi_P(B^s)\nabla_d(B)\nabla_D(B^s)X_t = \theta_q(B)\Theta_Q(B^s)\varepsilon_t$$

e.g. Seasonal Moving Average (MA) [order Q]

$$\Theta_Q(B^s) \equiv \left(1 - \Theta_1 B^s - \Theta_2 B^{2s} - \ldots - \Theta_Q B^{Ps}\right)$$

ARIMA Algebra

e.g. ARIMA (1, 1, 0):

$$\phi_1(B)\nabla_1(B)X_t = \theta_0(B)\varepsilon_t$$
$$(1 - \phi_1 B)(1 - B)X_t = (1)\varepsilon_t$$
$$\left(1 - \phi_1 B - B + \phi_1 B^2\right) X_t = \varepsilon_t$$
$$X_t - (1 + \phi_1)X_{t-1} + \phi_1 X_{t-2} = \varepsilon_t$$

ARIMA Algebra

Forecasting with ARIMA

Box-Cox Transformation

power transformation to impose normality on the time series

$$y^{(\lambda)} = \begin{cases} \frac{y^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0 \\ \ln(y) & \text{if } \lambda = 0 \end{cases}$$

```
from utils import poisson_process
from scipy.stats import boxcox

comps = [
    {'function': poisson_process._trend,
     'params': {'percent_increase': 1.0, 'period': 365}},
    {'function': poisson_process._cyclical,
     'params': {'weekday_factor': [1, 1, 1, 1, 1, 0.75, 0.75]}},
    {'function': poisson_process._seasonal,
     'params': {'peak_time': '09:00:00', 'amplitude': 0.75}}]

nhpp = poisson_process.nhpp(λ=200, n=int(1e5), t0='2017-03-01', freq='1D',
                            thinning_components=comps)
nhpp_bc, λ_bc = boxcox(nhpp)
```

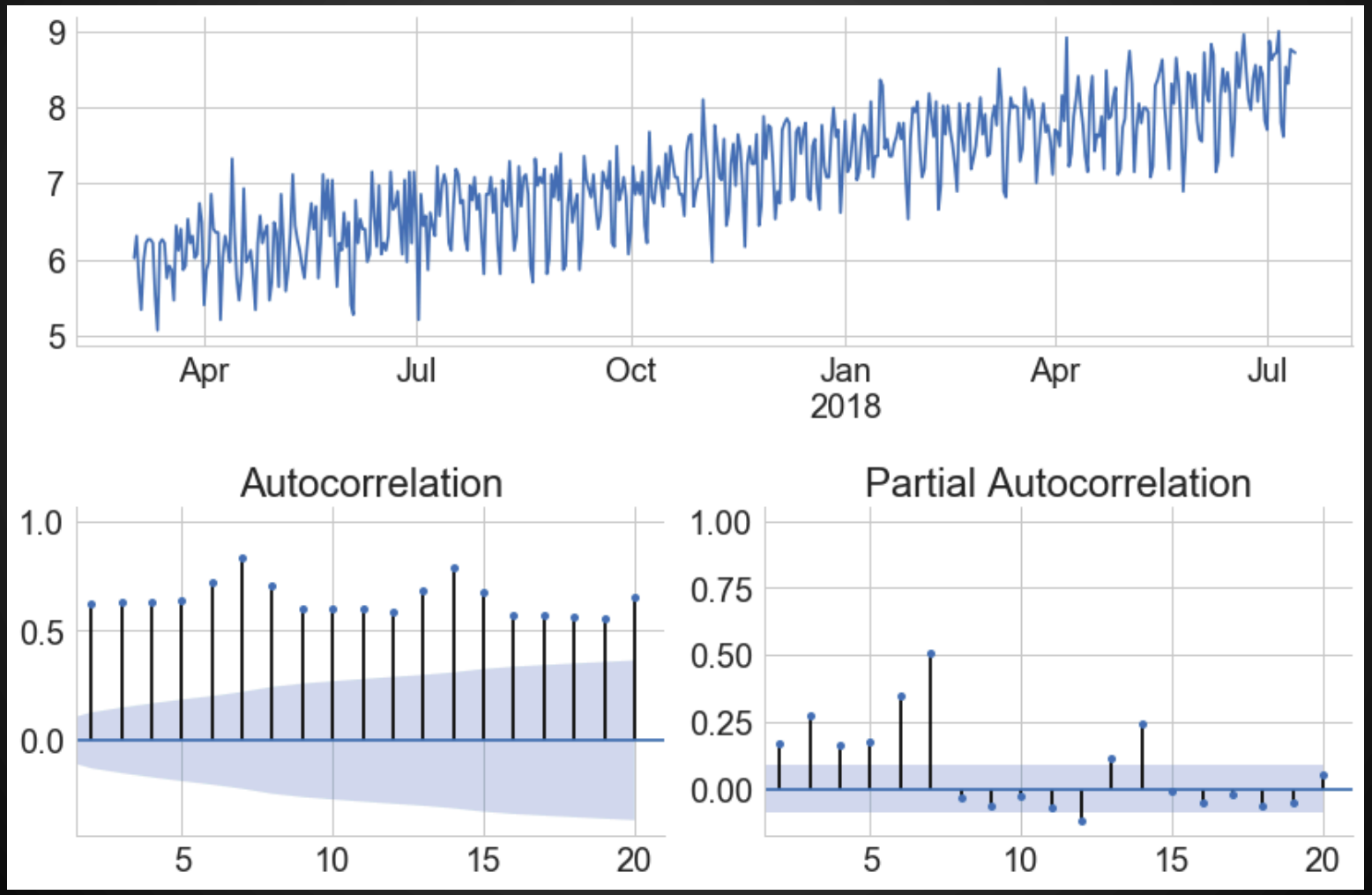
Forecasting with ARIMA

(Partial) Autocorrelation Functions ACF/PACF plots

```
import matplotlib.pyplot as plt
import statsmodels.tsa.api as smt
import seaborn as sns


def tsplot(ts, lags=None):
    """
    Plot the time series with ACF and PACF
        Stolen from Tom Augspurger's fantastic post on pandas and time series forecasting
        https://tomaugspurger.github.io/modern-7-timeseries
    """
    fig = plt.figure()
    layout = (2, 2)
    ts_ax = plt.subplot2grid(layout, (0, 0), colspan=2)
    acf_ax = plt.subplot2grid(layout, (1, 0))
    pacf_ax = plt.subplot2grid(layout, (1, 1))

    ts.plot(ax=ts_ax)
    smt.graphics.plot_acf(ts, lags=lags, ax=acf_ax)
    smt.graphics.plot_pacf(ts, lags=lags, ax=pacf_ax)
    [ax.set_xlim(1.5) for ax in [acf_ax, pacf_ax]]
    sns.despine()
    plt.tight_layout()
    return ts_ax, acf_ax, pacf_ax
```

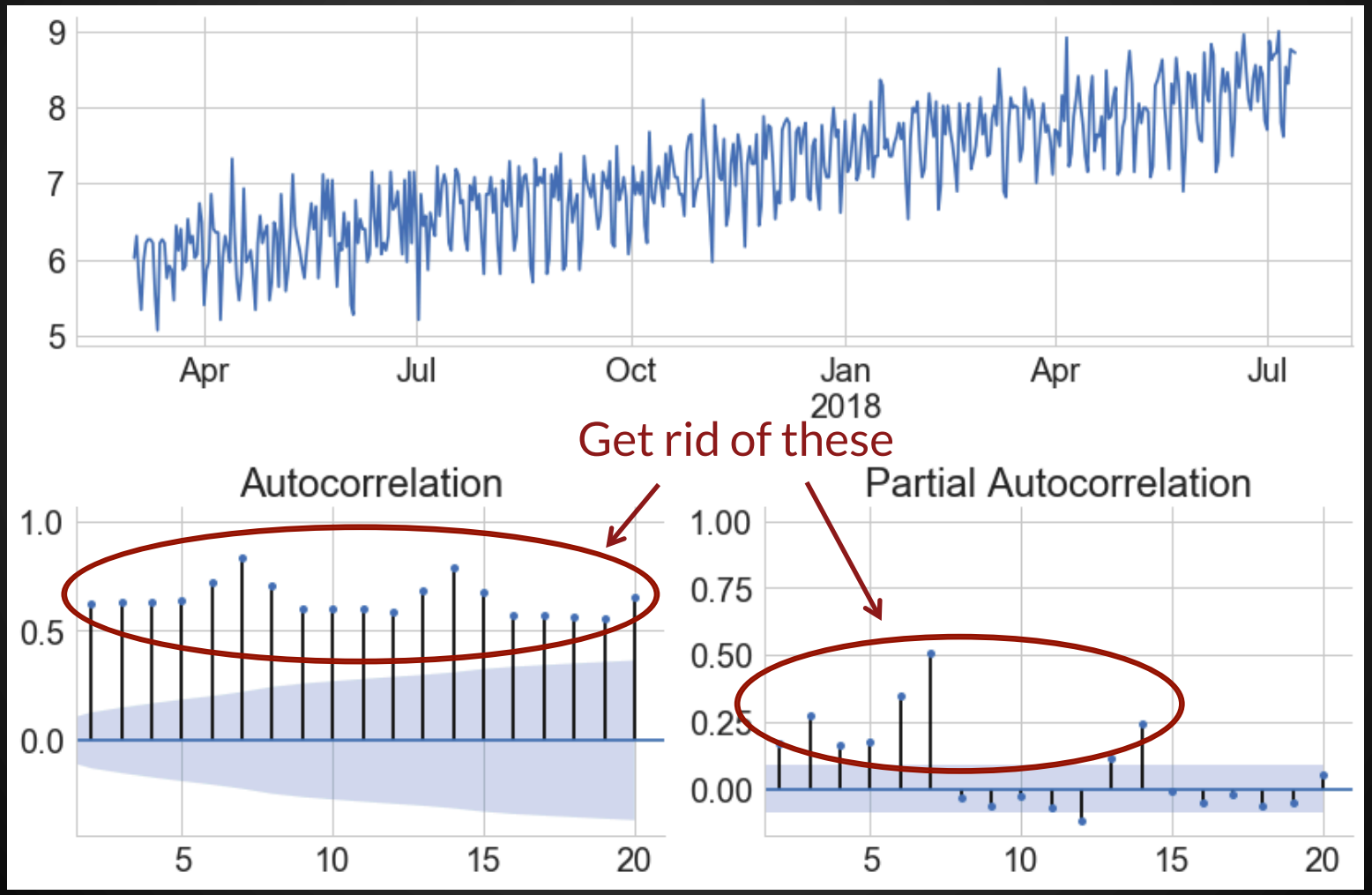
Forecasting with ARIMA

(Partial) Autocorrelation Functions ACF/PACF plots



Forecasting with ARIMA

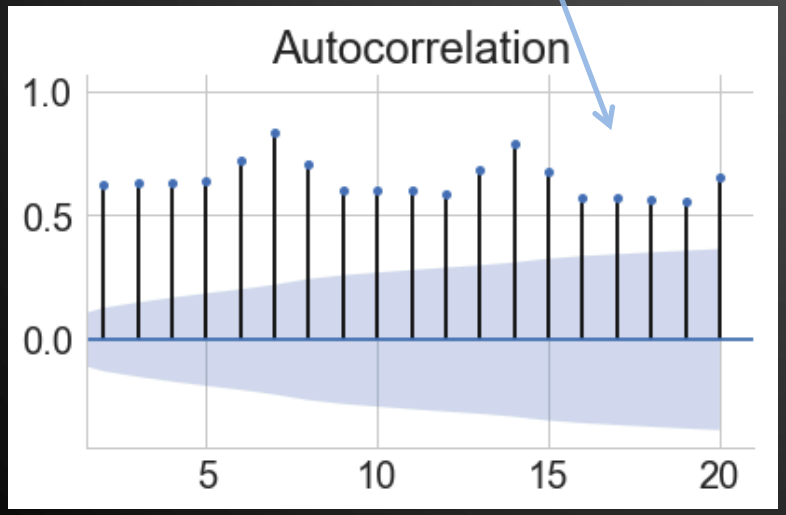
(Partial) Autocorrelation Functions ACF/PACF plots



Forecasting with ARIMA

(Partial) Autocorrelation Functions ACF/PACF plots

ACF being so high for so long
suggests Integrated [d>0]



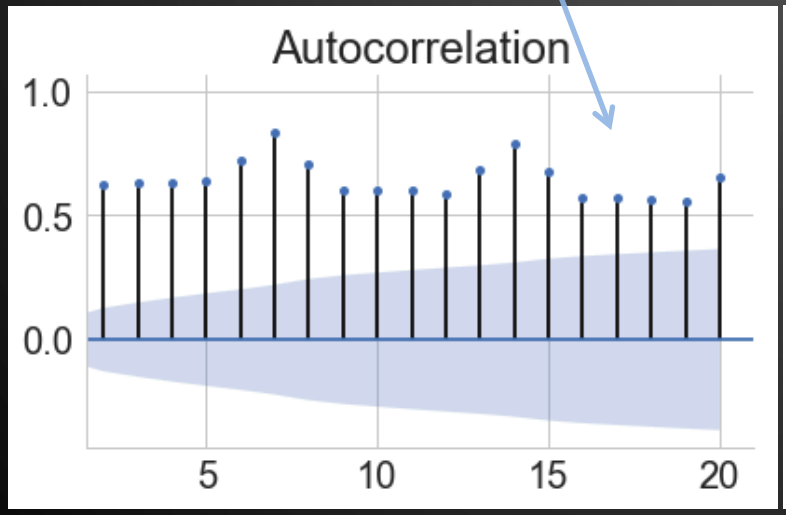
Forecasting with ARIMA

(Partial) Autocorrelation Functions ACF/PACF plots

ACF being so high for so long suggests Integrated [d>0]

as does a test for Stationarity, Adjusted Dickey-Fuller Test

(this is the p-value for the time series being stationary)




```
1  import statsmodels.tsa.api as smt
2  smt.adfuller(nhpp_bc)
```

```
(-1.0652843554507179,
 0.72871575782842646,
 14,
 485,
 {'1%': -3.4439051505128342,
  '10%': -2.5699539005207779,
  '5%': -2.8675177321998131},
 1083.6330094891273)
```

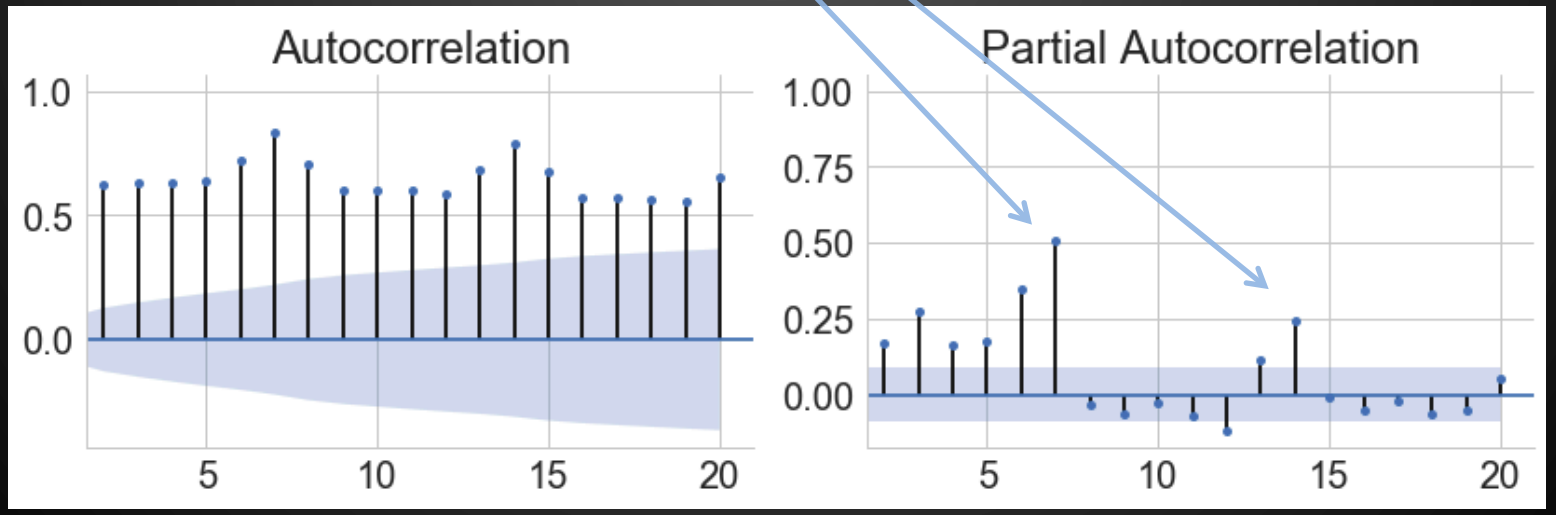
Forecasting with ARIMA

(Partial) Autocorrelation Functions ACF/PACF plots

ACF being so high for so long suggests Integrated [d>0]

suggests Seasonal [s=7]

as does a test for Stationarity, Adjusted Dickey-Fuller Test



Forecasting with ARIMA

(Partial) Autocorrelation Functions ACF/PACF plots

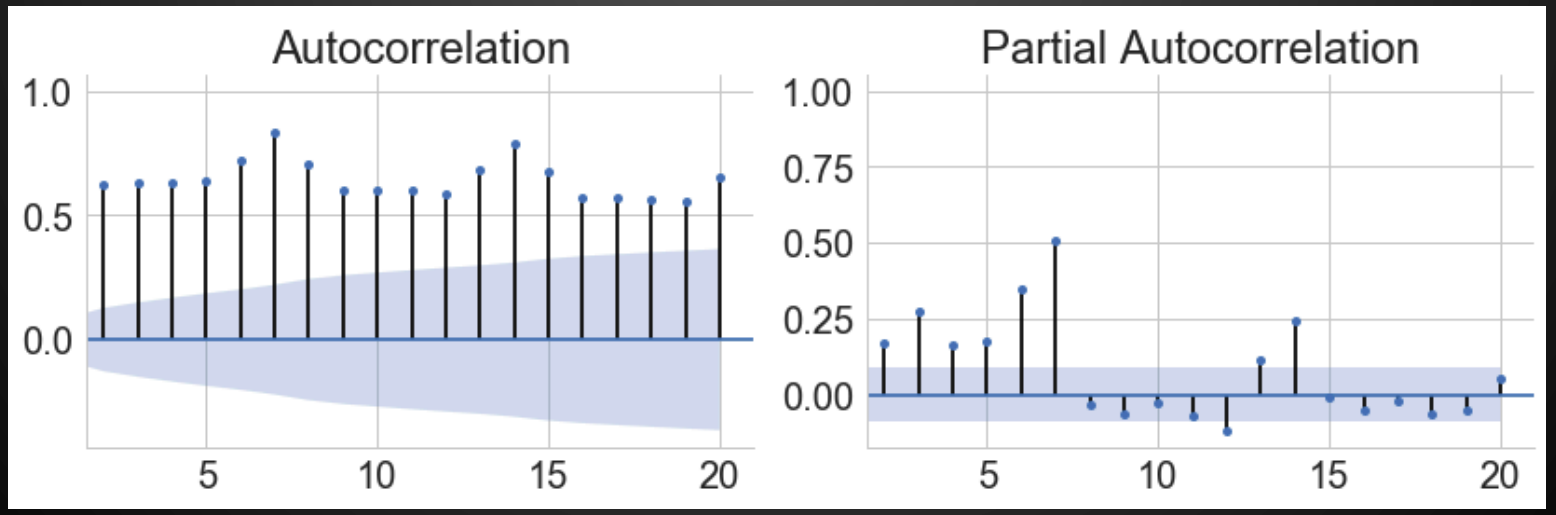
ACF being so high for so long suggests Integrated [d>0]

suggests Seasonal [s=7]

as does a test for Stationarity, Adjusted Dickey-Fuller Test

Try:

$$(p, d, q)\,(P, D, Q)_s = (0, 1, 0)\,(1, 0, 1)_7$$



Forecasting with ARIMA

$$(p, d, q)\,(P, D, Q)_s = (0, 1, 0)\,(1, 0, 1)_7$$


```
import statsmodels.tsa.api as smt

mod = smt.SARIMAX(nhpp_bc, order=(0, 1, 0), seasonal_order=(1, 0, 1, 7))
res = mod.fit()
```

Forecasting with ARIMA

$$(p, d, q)\,(P, D, Q)_s = (0, 1, 0)\,(1, 0, 1)_7$$

```
6  res.summary()
```

Statespace Model Results

Dep. Variable:	transactions	**No. Observations:**	500
Model:	SARIMAX(0, 1, 0)x(1, 0, 1, 7)	**Log Likelihood**	-694.806
Date:	Wed, 06 Dec 2017	**AIC**	1395.612
Time:	23:43:00	**BIC**	1408.256
Sample:	03-02-2017	**HQIC**	1400.574
	- 07-14-2018		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.S.L7	0.9994	0.001	940.764	0.000	0.997	1.001
ma.S.L7	-0.9669	0.026	-37.099	0.000	-1.018	-0.916
sigma2	0.9144	0.059	15.538	0.000	0.799	1.030

Ljung-Box (Q):	189.81	**Jarque-Bera (JB):**	0.01
Prob(Q):	0.00	**Prob(JB):**	1.00
Heteroskedasticity (H):	0.75	**Skew:**	0.00
Prob(H) (two-sided):	0.06	**Kurtosis:**	3.02

Forecasting with ARIMA

$$(p, d, q)\,(P, D, Q)_s = (0, 1, 0)\,(1, 0, 1)_7$$

```
6  res.summary()
```

Statespace Model Results

Dep. Variable:	transactions	**No. Observations:**	500
Model:	SARIMAX(0, 1, 0)x(1, 0, 1, 7)	**Log Likelihood**	-694.806
Date:	Wed, 06 Dec 2017	**AIC**	1395.612
Time:	23:43:00	**BIC**	1408.256
Sample:	03-02-2017	**HQIC**	1400.574
	- 07-14-2018		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.S.L7	0.9994	0.001	940.764	0.000	0.997	1.001
ma.S.L7	-0.9669	0.026	-37.099	0.000	-1.018	-0.916
sigma2	0.9144	0.059	15.538	0.000	0.799	1.030

Ljung-Box (Q):	189.81	**Jarque-Bera (JB):**	0.01
Prob(Q):	0.00	**Prob(JB):**	1.00
Heteroskedasticity (H):	0.75	**Skew:**	0.00
Prob(H) (two-sided):	0.06	**Kurtosis:**	3.02

AIC measures goodness-of-fit but punishes for using more parameters. Smaller AIC implies a better model.

Forecasting with ARIMA

$$(p, d, q)\,(P, D, Q)_s = (0, 1, 0)\,(1, 0, 1)_7$$

```
6  res.summary()
```

Statespace Model Results

Dep. Variable:	transactions	**No. Observations:**	500
Model:	SARIMAX(0, 1, 0)x(1, 0, 1, 7)	**Log Likelihood**	-694.806
Date:	Wed, 06 Dec 2017	**AIC**	1395.612
Time:	23:43:00	**BIC**	1408.256
Sample:	03-02-2017	**HQIC**	1400.574
	- 07-14-2018		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.S.L7	0.9994	0.001	940.764	0.000	0.997	1.001
ma.S.L7	-0.9669	0.026	-37.099	0.000	-1.018	-0.916
sigma2	0.9144	0.059	15.538	0.000	0.799	1.030

Ljung-Box (Q):	189.81	**Jarque-Bera (JB):**	0.01
Prob(Q):	0.00	**Prob(JB):**	1.00
Heteroskedasticity (H):	0.75	**Skew:**	0.00
Prob(H) (two-sided):	0.06	**Kurtosis:**	3.02

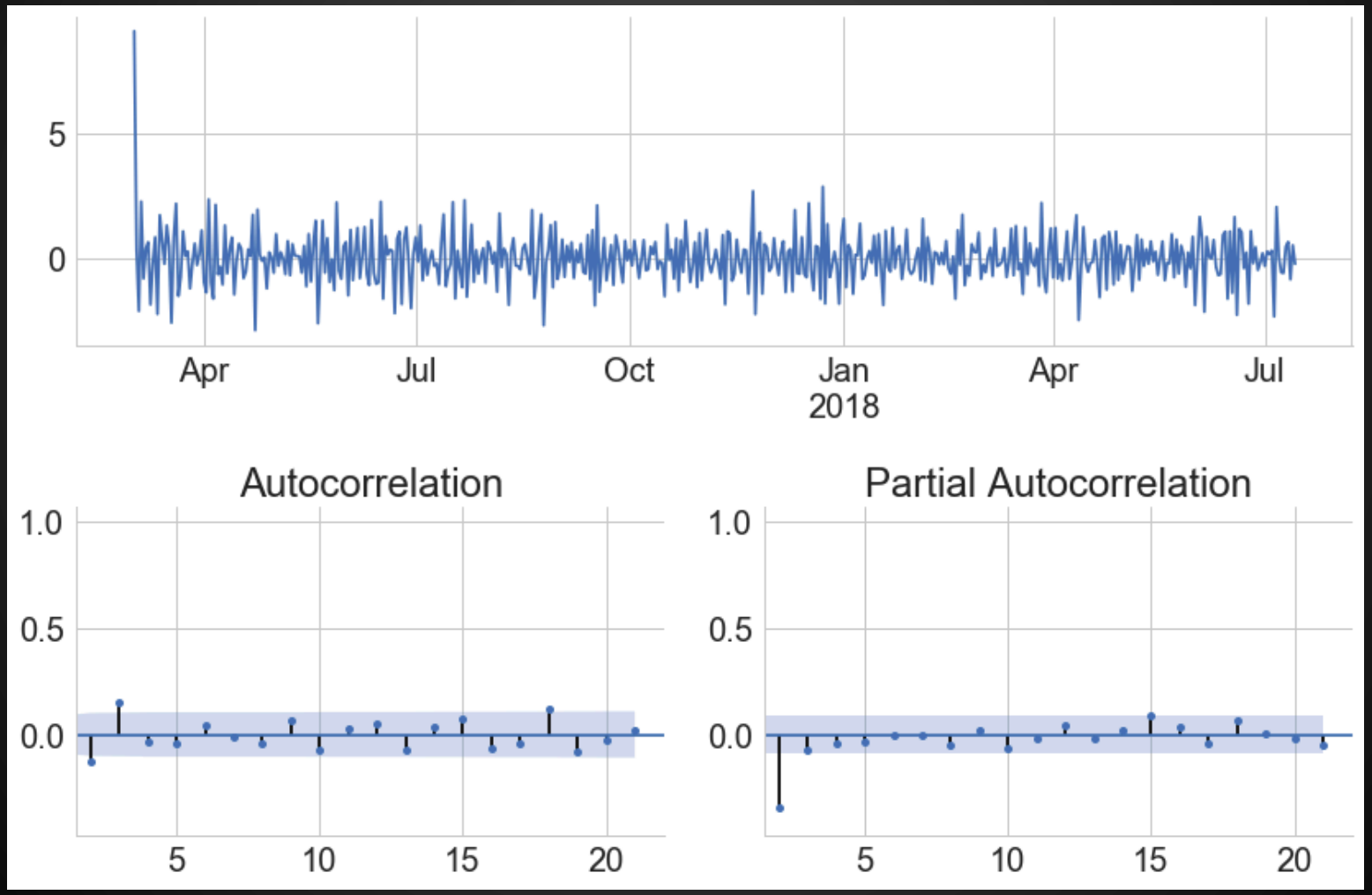
Φ_1

Θ_1

AIC measures goodness-of-fit but punishes for using more parameters. Smaller AIC implies a better model.

Forecasting with ARIMA

$$(p, d, q)\,(P, D, Q)_s = (0, 1, 0)\,(1, 0, 1)_7$$

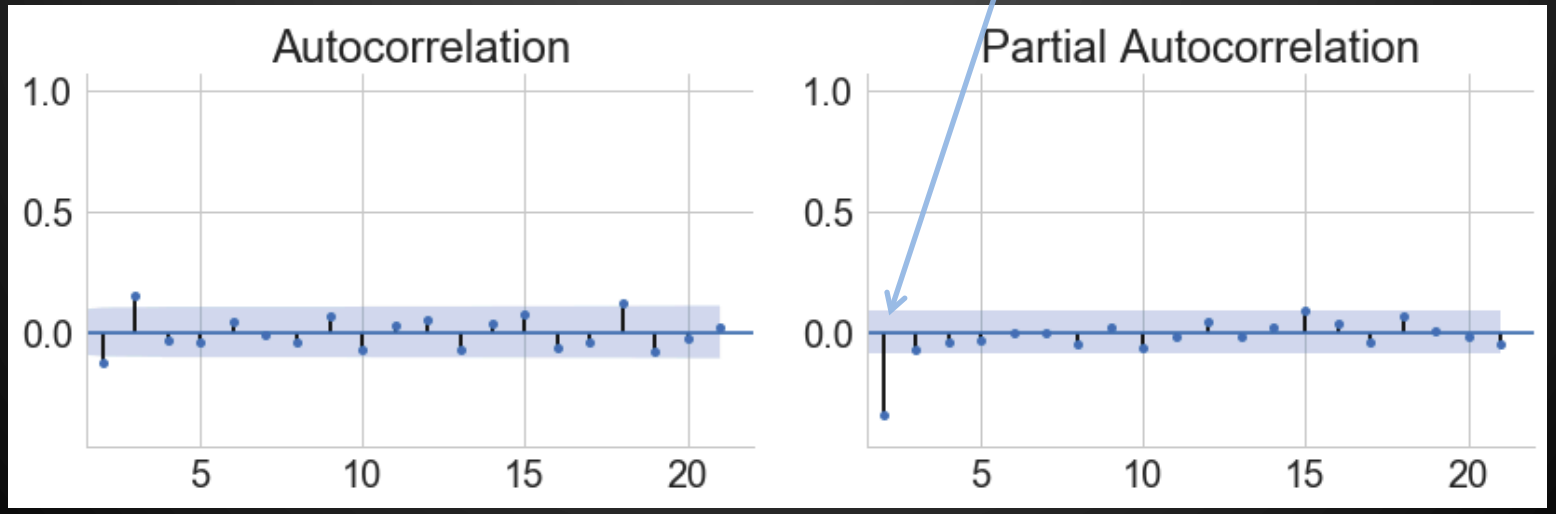


Forecasting with ARIMA

$$(p, d, q)\,(P, D, Q)_s = (0, 1, 0)\,(1, 0, 1)_7$$

PACF being non-zero at 1 but falling quickly after suggests Moving Average [q=1]

MA(1) is memory-less, covariance for steps > 1 should be 0



Forecasting with ARIMA

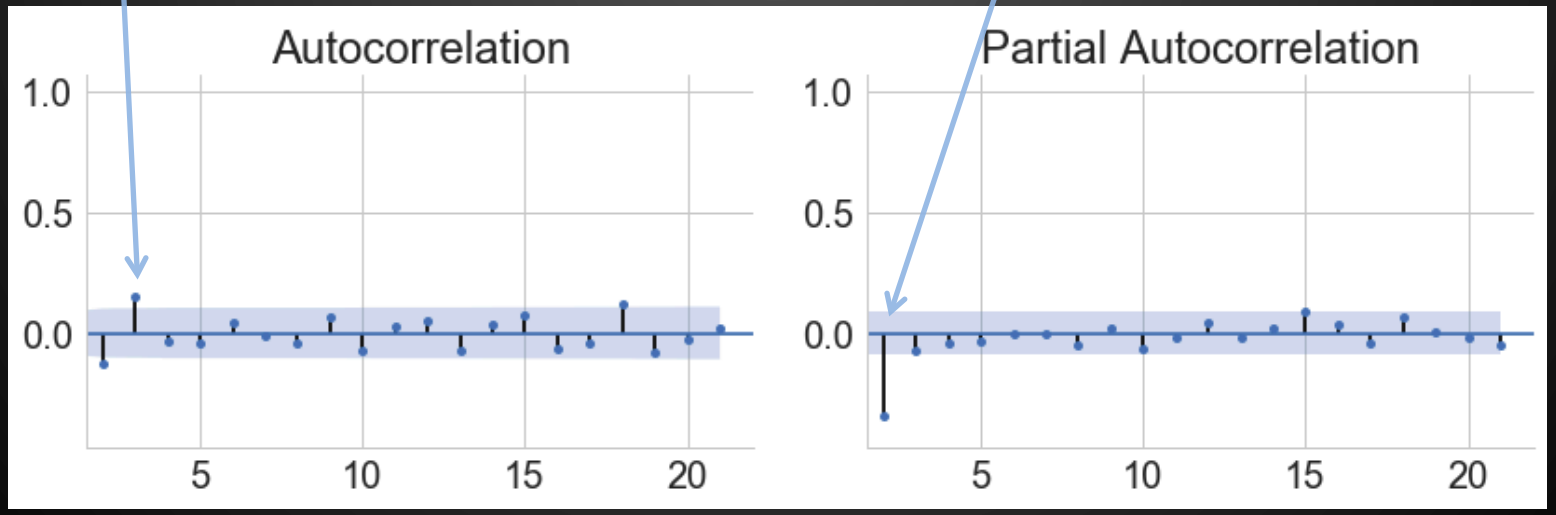
$$(p, d, q)\,(P, D, Q)_s = (0, 1, 0)\,(1, 0, 1)_7$$

ACF being non-zero above 1 suggests Autoregression [p>0]

covariance for steps > 0 decays as ϕ^k

PACF being non-zero at 1 but falling quickly after suggests Moving Average [q=1]

MA(1) is memory-less, covariance for steps > 1 should be 0



Forecasting with ARIMA

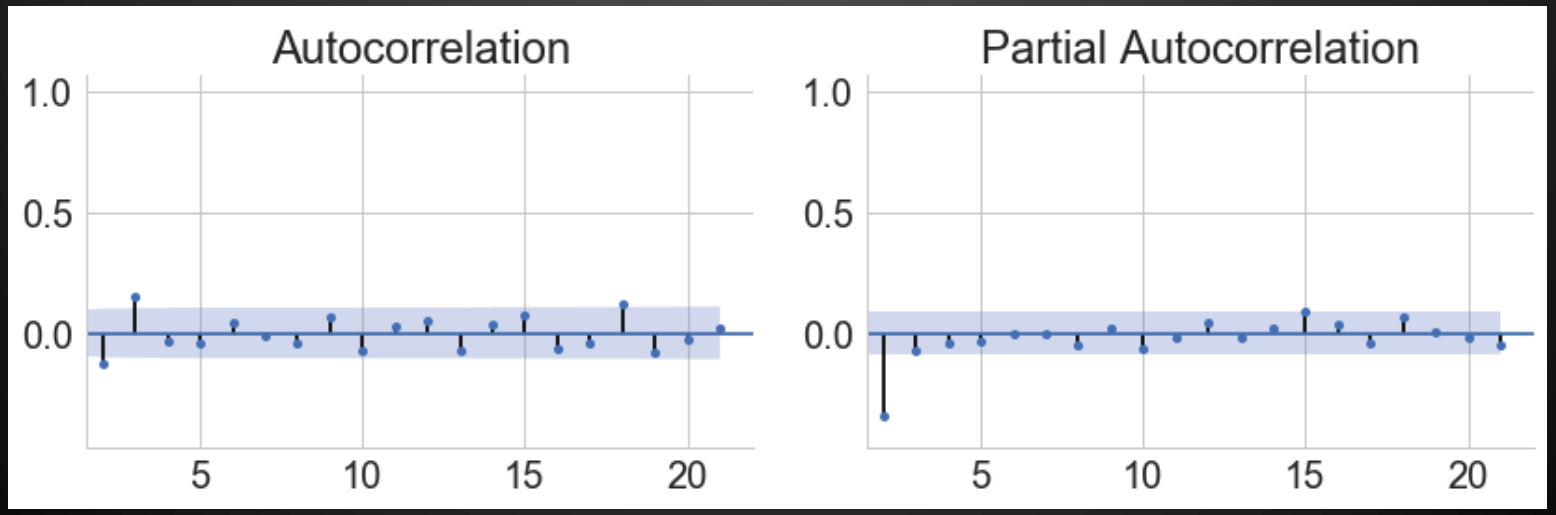
$$(p, d, q)\,(P, D, Q)_s = (0, 1, 0)\,(1, 0, 1)_7$$

ACF being non-zero above 1 suggests Autoregression [p>0]

PACF being non-zero at 1 but falling quickly after suggests Moving Average [q=1]

Try:

$$(p, d, q)\,(P, D, Q)_s = (2, 1, 1)\,(1, 0, 1)_7$$



Forecasting with ARIMA

$$(p, d, q)\,(P, D, Q)_s = (2, 1, 1)\,(1, 0, 1)_7$$

Statespace Model Results

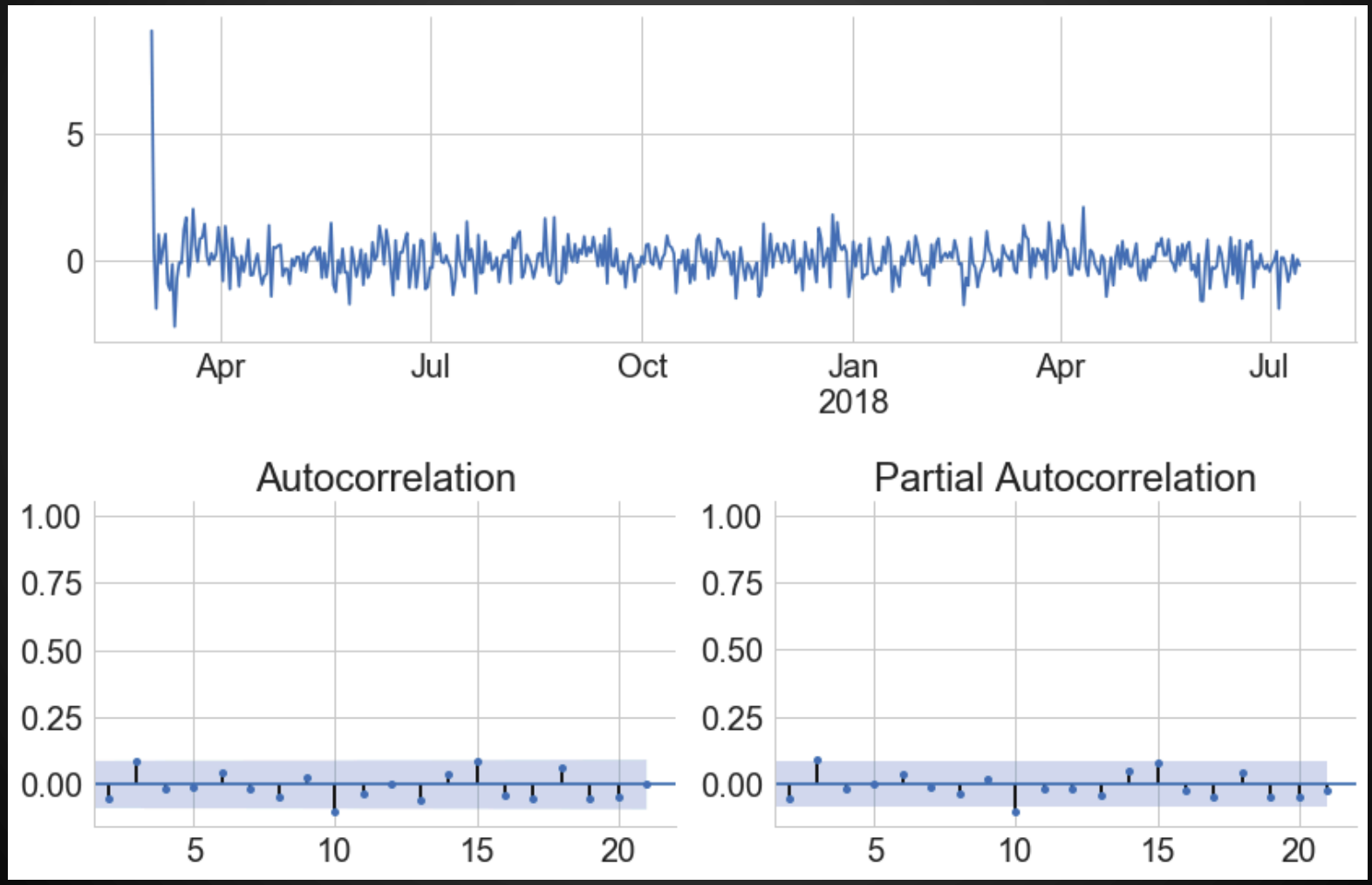
Dep. Variable:	transactions	**No. Observations:**	500
Model:	SARIMAX(2, 1, 1)x(1, 0, 1, 7)	**Log Likelihood**	-514.523
Date:	Wed, 06 Dec 2017	**AIC**	1041.046
Time:	23:56:32	**BIC**	1066.333
Sample:	03-02-2017	**HQIC**	1050.968
	- 07-14-2018		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.0629	0.046	-1.375	0.169	-0.152	0.027
ar.L2	-0.0986	0.049	-2.031	0.042	-0.194	-0.003
ma.L1	-0.9662	0.013	-75.824	0.000	-0.991	-0.941
ar.S.L7	0.9996	0.001	1602.365	0.000	0.998	1.001
ma.S.L7	-0.9670	0.025	-38.726	0.000	-1.016	-0.918
sigma2	0.4381	0.027	16.152	0.000	0.385	0.491

Ljung-Box (Q):	36.22	**Jarque-Bera (JB):**	0.71
Prob(Q):	0.64	**Prob(JB):**	0.70
Heteroskedasticity (H):	0.83	**Skew:**	-0.04
Prob(H) (two-sided):	0.24	**Kurtosis:**	3.16

Forecasting with ARIMA

$$(p, d, q)\,(P, D, Q)_s = (2, 1, 1)\,(1, 0, 1)_7$$

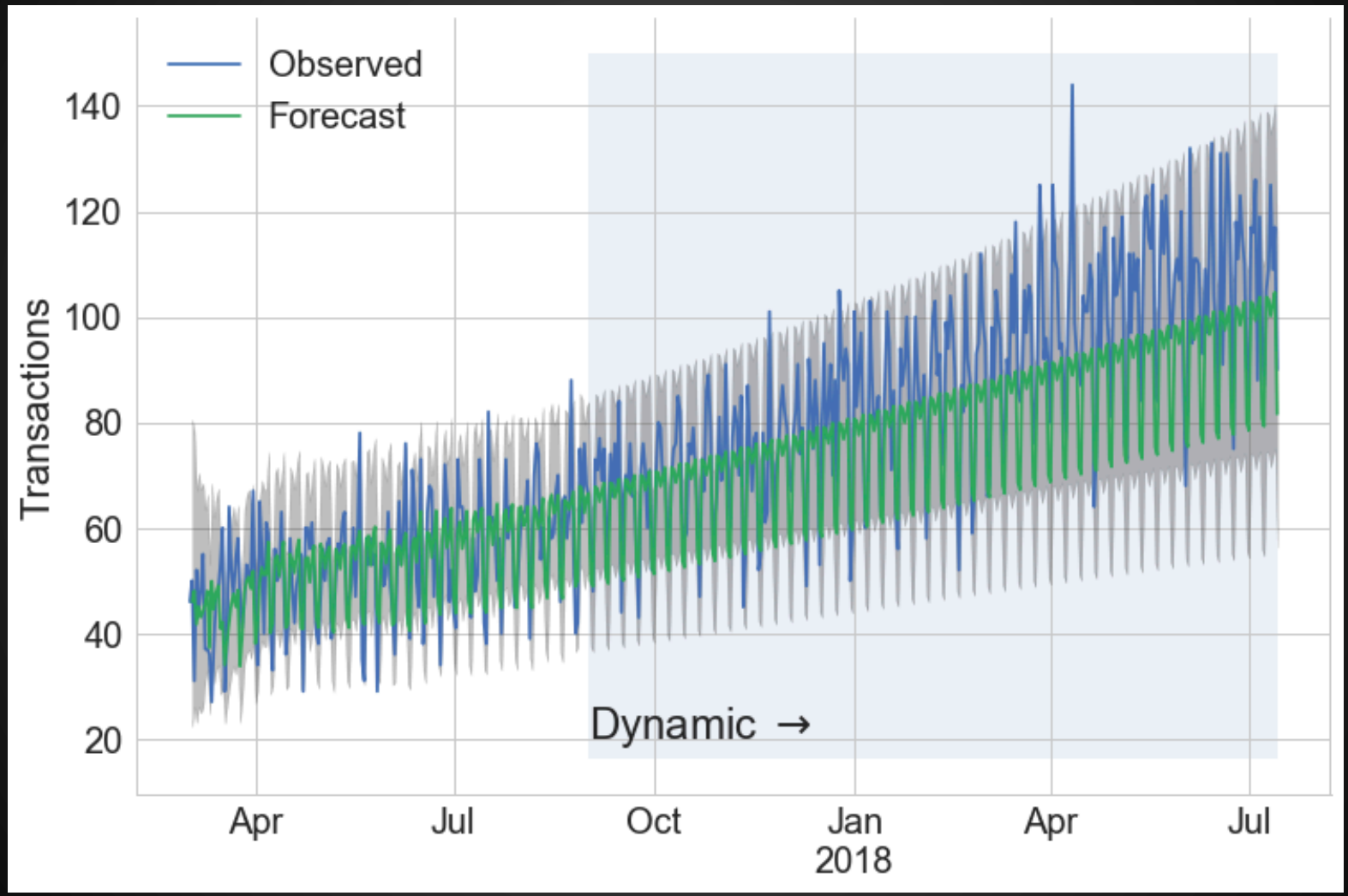


Forecasting with ARIMA

Forecasting with ARIMA

```
# Get the predictions, and forecast beyond dynamic_start
dynamic_start = '2017-09-01'
pred = res.get_prediction(start='2017-03-03', end=max(nhpp.index), dynamic=dynamic_start)

# Undo the Box-Cox transformation
predictions = (λ_bc * pred.predicted_mean + 1) ** (1/ λ_bc)
predictions_bounds = (λ_bc * pred.conf_int() + 1) ** (1 / λ_bc)

# Plot the time series and the forecast
ax = nhpp.plot(label='Observed')
predictions.plot(ax=ax, label='Forecast')
ax.fill_between(predictions.index,
                predictions_bounds.iloc[:, 0],
                predictions_bounds.iloc[:, 1], color='k', alpha=.25)

# Highlight the forecast area
ax.fill_betweenx(ax.get_ylim(), pd.Timestamp(dynamic_start), predictions.index[-1], alpha=0.1)
ax.annotate('Dynamic $\\rightarrow$', (pd.Timestamp(dynamic_start), 20))
ax.set_ylabel('Transactions')
plt.legend()
''
```

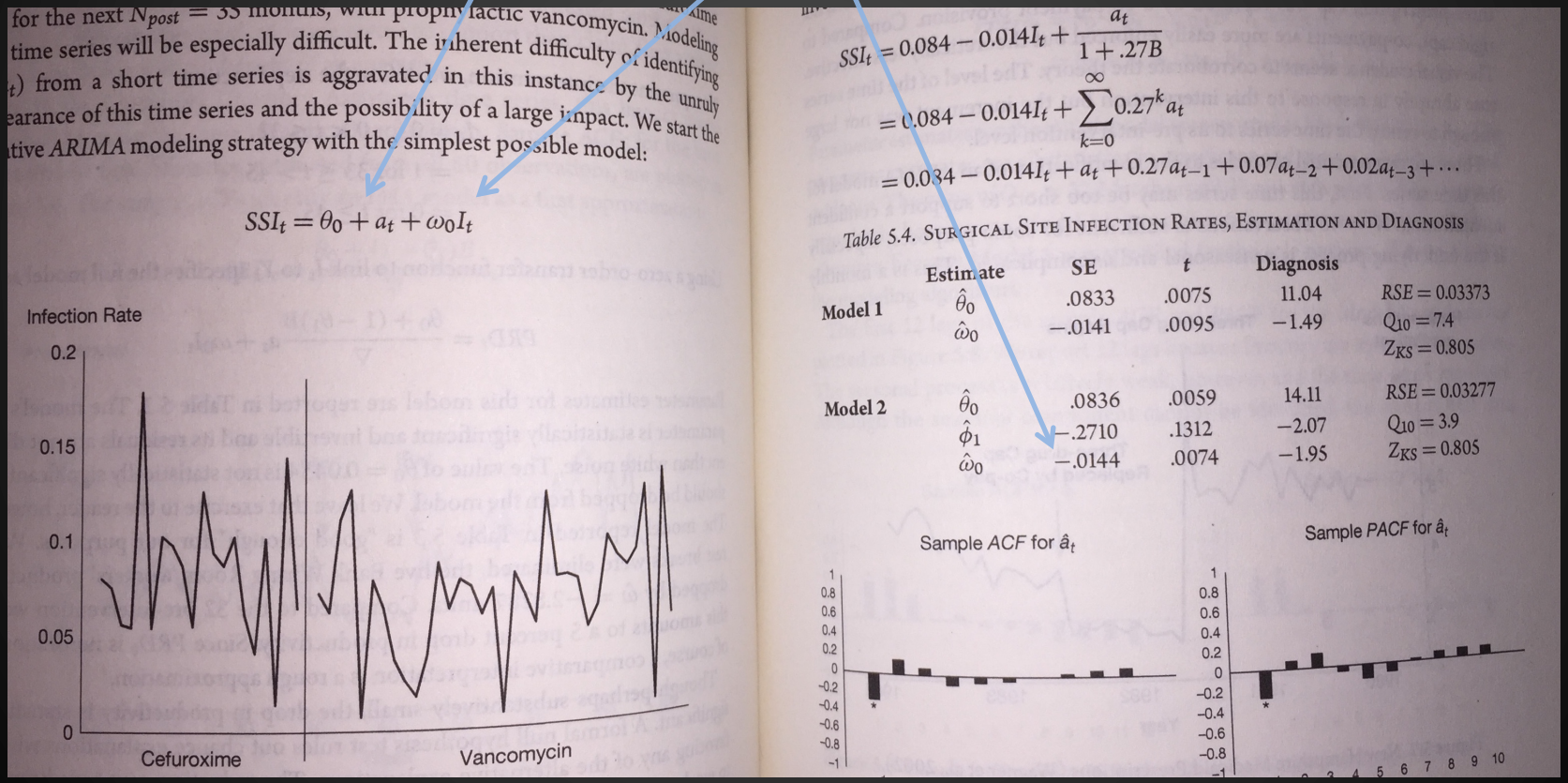
Forecasting with ARIMA



Interrupted Time Series

Introduce variant at time t_{start} of effect size β

Basic idea : Fit ARIMA + effect size



for the next $N_{post} = 35$ months, with prophylactic vancomycin. Modeling time series will be especially difficult. The inherent difficulty of identifying θ_t) from a short time series is aggravated in this instance by the unruly appearance of this time series and the possibility of a large impact. We start the iterative *ARIMA* modeling strategy with the simplest possible model:

$$SSI_t = \theta_0 + a_t + \omega_0 I_t$$

Infection Rate

0.2
0.15
0.1
0.05
0

Cefuroxime

Vancomycin

$$SSI_t = 0.084 - 0.014 I_t + \frac{a_t}{1 + .27B}$$

$$= 0.084 - 0.014 I_t + \sum_{k=0}^{\infty} 0.27^k a_t$$

$$= 0.084 - 0.014 I_t + a_t + 0.27 a_{t-1} + 0.07 a_{t-2} + 0.02 a_{t-3} + \cdots$$

Table 5.4. Surgical Site Infection Rates, Estimation and Diagnosis

	Estimate	SE	t	Diagnosis	
Model 1	$\hat{\theta}_0$	.0833	.0075	11.04	$RSE = 0.03373$
	$\hat{\omega}_0$	−.0141	.0095	−1.49	$Q_{10} = 7.4$
					$Z_{KS} = 0.805$
Model 2	$\hat{\theta}_0$	.0836	.0059	14.11	$RSE = 0.03277$
	$\hat{\phi}_1$	−.2710	.1312	−2.07	$Q_{10} = 3.9$
	$\hat{\omega}_0$	−.0144	.0074	−1.95	$Z_{KS} = 0.805$

Sample *ACF* for $\hat{a}_t$

Sample *PACF* for $\hat{a}_t$

1 0.8 0.6 0.4 0.2 0 −0.2 −0.4 −0.6 −0.8 −1

1 2 3 4 5 6 7 8 9 10

source: McCleary, et al. "Design and Analysis of Time Series Experiments"

Interrupted Time Series

Start with a simple time series, for illustration

```
df = poisson_process.nhpp(
    λ=100, n=10000, t0='2017-03-01', freq='1D',
    thinning_components=[{'function': poisson_process._trend,
                          'params': {'percent_increase': 1.0, 'period': 365}}]).to_frame()
```

Interrupted Time Series

Add a small effect size at some start time

```
df = poisson_process.nhpp(
    λ=100, n=10000, t0='2017-03-01', freq='1D',
    thinning_components=[{'function': poisson_process._trend,
                          'params': {'percent_increase': 1.0, 'period': 365}}]).to_frame()

treatment_start = '2017-05-01'
effect_size = 0.1
df.loc[treatment_start:, 'response'] = (
    np.random.normal(loc=effect_size, scale=effect_size / 2, size=len(df[treatment_start:])))
df['transactions_with_response'] = (df['transactions'] * (1 + df['response'].fillna(0))).astype(int)
```

Interrupted Time Series

Find the ARIMA model that fits best

```
df = poisson_process.nhpp(
    λ=100, n=10000, t0='2017-03-01', freq='1D',
    thinning_components=[{'function': poisson_process._trend,
                          'params': {'percent_increase': 1.0, 'period': 365}}]).to_frame()

treatment_start = '2017-05-01'
effect_size = 0.1
df.loc[treatment_start:, 'response'] = (
    np.random.normal(loc=effect_size, scale=effect_size / 2, size=len(df[treatment_start:])))
df['transactions_with_response'] = (df['transactions'] * (1 + df['response'].fillna(0))).astype(int)

mod = smt.SARIMAX(df.loc[:treatment_start, 'transactions'],
                  order=(0, 1, 1),
                  seasonal_order=(0, 0, 0, 0))
res = mod.fit()
```

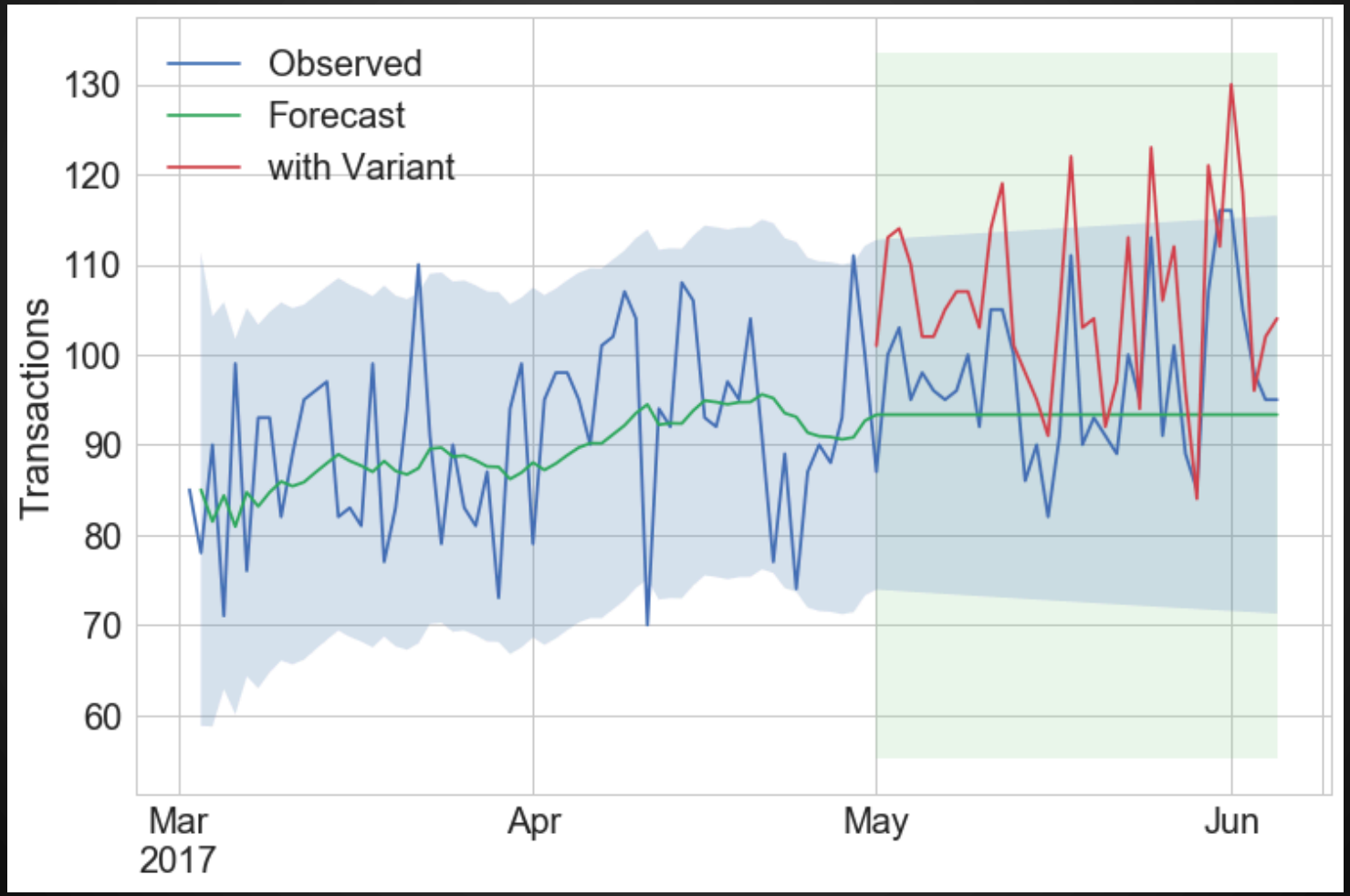
Interrupted Time Series

Get the predictions and confidence bands

```
df = poisson_process.nhpp(
    λ=100, n=10000, t0='2017-03-01', freq='1D',
    thinning_components=[{'function': poisson_process._trend,
                          'params': {'percent_increase': 1.0, 'period': 365}}]).to_frame()

treatment_start = '2017-05-01'
effect_size = 0.1
df.loc[treatment_start:, 'response'] = (
    np.random.normal(loc=effect_size, scale=effect_size / 2, size=len(df[treatment_start:])))
df['transactions_with_response'] = (df['transactions'] * (1 + df['response'].fillna(0))).astype(int)

mod = smt.SARIMAX(df.loc[:treatment_start, 'transactions'],
                  order=(0, 1, 1),
                  seasonal_order=(0, 0, 0, 0))
res = mod.fit()

pred = res.get_prediction(start='2017-03-03', end=max(df.index), dynamic=treatment_start)
df['prediction'] = pred.predicted_mean
df[['prediction_lower', 'prediction_upper']] = pred.conf_int()
```

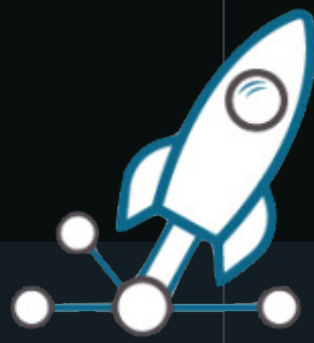
Interrupted Time Series

Our Data Set



Interrupted Time Series

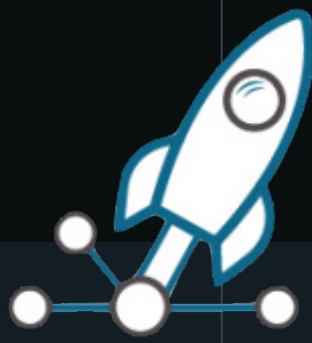
```python
import pymc3 as pm
from theano import scan, shared
import theano.tensor as tt


def build_model(X, treatment_start, treatment_observations):
    time_seen = pd.to_datetime(treatment_start) + pd.DateOffset(treatment_observations - 1)
    y = shared(X[:time_seen].values)
    y_switch = shared(X[:time_seen].index < treatment_start)
    with pm.Model() as i1ma1:
        σ = pm.HalfCauchy('σ', beta=2.)
        θ = pm.Normal('θ', 0., sd=2.)
        β = pm.Normal('β', 0., sd=2.)

        y_adj = tt.switch(y_switch, y, y - tt.dot(y, β))

        # ARIMA (0, 1, 1)
        # ---------------
        # (1 - B) y[t] = (1 - θB) ε[t]
        # y[t] - y[t-1] = ε[t] - θ * ε[t-1]
        # ε[t] = y[t] - y[t-1] - θ * ε[t-1]
        def calc_next(y_lag1, y_lag0, ε, θ):
            return y_lag0 - y_lag1 - θ * ε

        # Initial noise guess -- let's just seed with 0
        ε0 = tt.zeros_like(y_adj)

        ε, _ = scan(fn=calc_next,
                    sequences=dict(input=y_adj, taps=[-1, 0]),
                    outputs_info=[ε0],
                    non_sequences=[θ])

        pm.Potential('like', pm.Normal.dist(0, sd=σ).logp(ε))
    return i1ma1
```



Interrupted Time Series

```python
import pymc3 as pm
from theano import scan, shared
import theano.tensor as tt


def build_model(X, treatment_start, treatment_observations):
    time_seen = pd.to_datetime(treatment_start) + pd.DateOffset(treatment_observations - 1)
    y = shared(X[:time_seen].values)
    y_switch = shared(X[:time_seen].index < treatment_start)
    with pm.Model() as i1ma1:
        σ = pm.HalfCauchy('σ', beta=2.)
        θ = pm.Normal('θ', 0., sd=2.)
        β = pm.Normal('β', 0., sd=2.)

        y_adj = tt.switch(y_switch, y, y - tt.dot(y, β))

        # ARIMA (0, 1, 1)
        # ---------------
        # (1 - B) y[t] = (1 - θB) ε[t]
        # y[t] - y[t-1] = ε[t] - θ * ε[t-1]
        # ε[t] = y[t] - y[t-1] - θ * ε[t-1]
        def calc_next(y_lag1, y_lag0, ε, θ):
            return y_lag0 - y_lag1 - θ * ε

        # Initial noise guess -- let's just seed with 0
        ε0 = tt.zeros_like(y_adj)

        ε, _ = scan(fn=calc_next,
                    sequences=dict(input=y_adj, taps=[-1, 0]),
                    outputs_info=[ε0],
                    non_sequences=[θ])

        pm.Potential('like', pm.Normal.dist(0, sd=σ).logp(ε))
    return i1ma1
```

Case statement:
left = control
right = variant
[effect size = β]

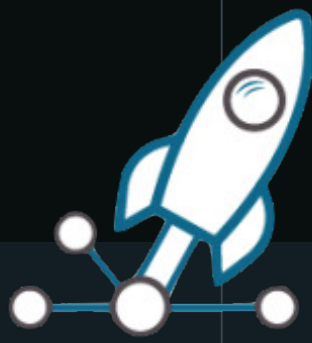


Interrupted Time Series

Case statement:
left = control
right = variant
[effect size = β]

ARIMA (0, 1, 1)

```python
import pymc3 as pm
from theano import scan, shared
import theano.tensor as tt


def build_model(X, treatment_start, treatment_observations):
    time_seen = pd.to_datetime(treatment_start) + pd.DateOffset(treatment_observations - 1)
    y = shared(X[:time_seen].values)
    y_switch = shared(X[:time_seen].index < treatment_start)
    with pm.Model() as i1ma1:
        σ = pm.HalfCauchy('σ', beta=2.)
        θ = pm.Normal('θ', 0., sd=2.)
        β = pm.Normal('β', 0., sd=2.)

        y_adj = tt.switch(y_switch, y, y - tt.dot(y, β))

        # ARIMA (0, 1, 1)
        # ---------------
        # (1 - B) y[t] = (1 - θB) ε[t]
        # y[t] - y[t-1] = ε[t] - θ * ε[t-1]
        # ε[t] = y[t] - y[t-1] - θ * ε[t-1]
        def calc_next(y_lag1, y_lag0, ε, θ):
            return y_lag0 - y_lag1 - θ * ε

        # Initial noise guess -- let's just seed with 0
        ε0 = tt.zeros_like(y_adj)

        ε, _ = scan(fn=calc_next,
                    sequences=dict(input=y_adj, taps=[-1, 0]),
                    outputs_info=[ε0],
                    non_sequences=[θ])

        pm.Potential('like', pm.Normal.dist(0, sd=σ).logp(ε))
    return i1ma1
```

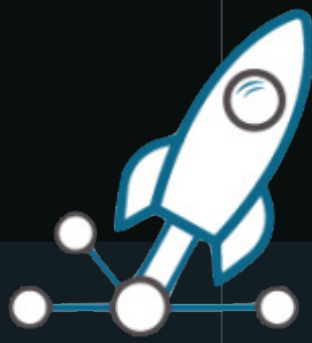


Interrupted Time Series

```python
import pymc3 as pm
from theano import scan, shared
import theano.tensor as tt


def build_model(X, treatment_start, treatment_observations):
    time_seen = pd.to_datetime(treatment_start) + pd.DateOffset(treatment_observations - 1)
    y = shared(X[:time_seen].values)
    y_switch = shared(X[:time_seen].index < treatment_start)
    with pm.Model() as i1ma1:
        σ = pm.HalfCauchy('σ', beta=2.)
        θ = pm.Normal('θ', 0., sd=2.)
        β = pm.Normal('β', 0., sd=2.)

        y_adj = tt.switch(y_switch, y, y - tt.dot(y, β))

        # ARIMA (0, 1, 1)
        # ---------------
        # (1 - B) y[t] = (1 - θB) ε[t]
        # y[t] - y[t-1] = ε[t] - θ * ε[t-1]
        # ε[t] = y[t] - y[t-1] - θ * ε[t-1]
        def calc_next(y_lag1, y_lag0, ε, θ):
            return y_lag0 - y_lag1 - θ * ε

        # Initial noise guess -- let's just seed with 0
        ε0 = tt.zeros_like(y_adj)

        ε, _ = scan(fn=calc_next,
                    sequences=dict(input=y_adj, taps=[-1, 0]),
                    outputs_info=[ε0],
                    non_sequences=[θ])

        pm.Potential('like', pm.Normal.dist(0, sd=σ).logp(ε))
    return i1ma1
```

Case statement:
left = control
right = variant
[effect size = β]

ARIMA (0, 1, 1)

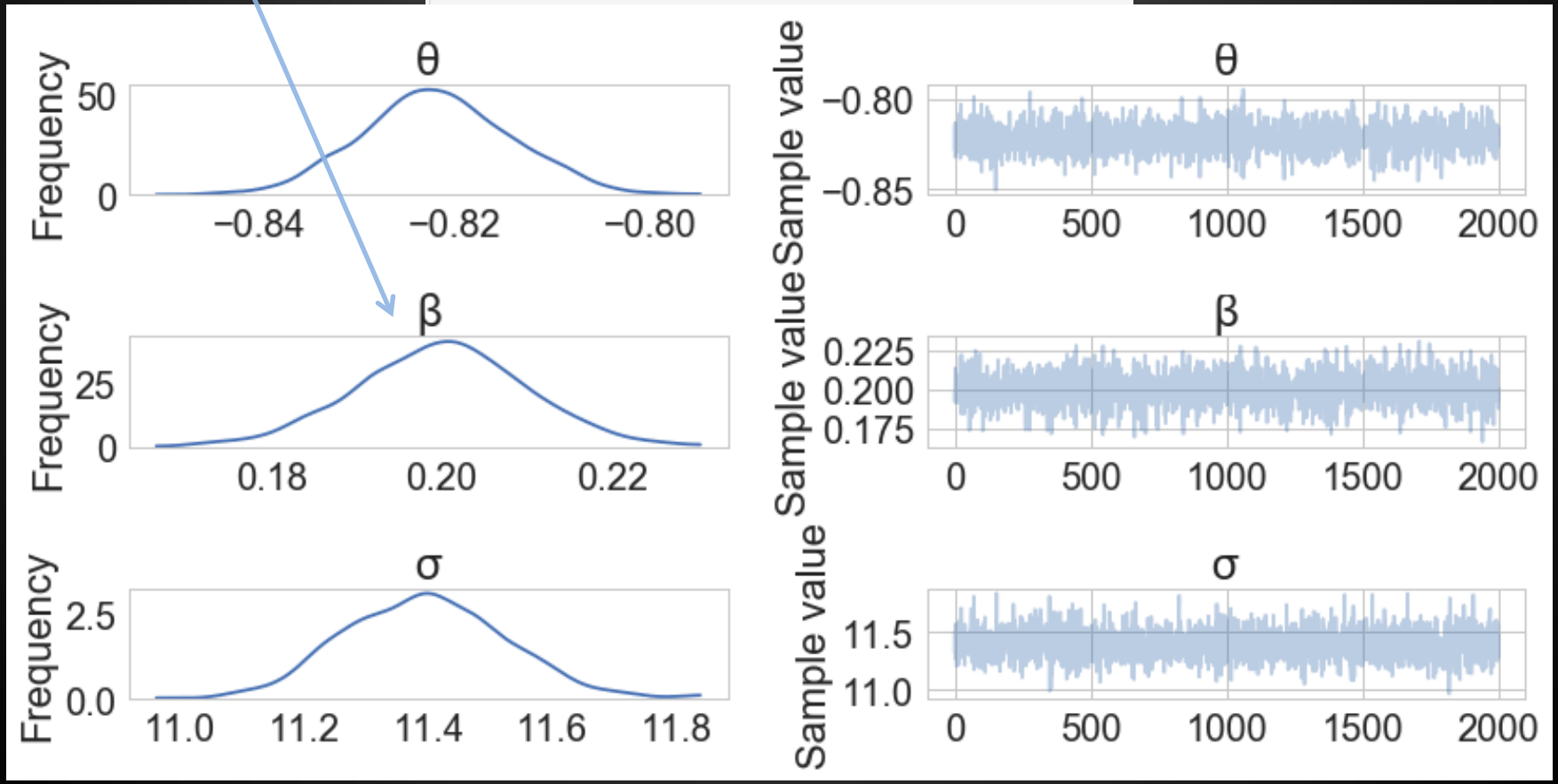


Looping in Theano

Interrupted Time Series

effect size estimation

```
treatment_observations = 2
with build_model(df['transactions_with_response'],
                 treatment_start,
                 treatment_observations):
    trace = pm.sample(2000)

pm.traceplot(trace, combined=True)
```



Interrupted Time Series

Interrupted Time Series Experiments in Python

PhD, Physics

Data Scientist

SPOT HERO

GRUBHUB

Drew Fustin
Automaton Data
available for hire!
drew@automatondata.com

github.com/drewfustin/talks__time-series-experiments