

# Working with time series data in pandas

CUSTOMER ANALYTICS AND A/B TESTING IN PYTHON



**Ryan Grossman**  
Data Scientist, EDO

# Exploratory Data Analysis

- Exploratory Data Analysis (EDA)
- Working with time series data
- Uncovering trends in KPIs over time

# Review: Manipulating dates & times



# Example: Week Two Conversion Rate

- **Week 2 Conversion Rate** Users who subscribe in the second week *after* the free trial
- Users must have:
  - Completed the free trial
  - **Not** subscribed in the first week
  - Had a full second week to subscribe or not

# Using the Timedelta class

- **Lapse Date:** Date the trial ends for a given user

```
import pandas as pd
from datetime import timedelta

# Define the most recent date in our data
current_date = pd.to_datetime('2018-03-17')
# The last date a user could lapse be included
max_lapse_date = current_date - timedelta(days=14)
# Filter down to only eligible users
conv_sub_data = sub_data_demo[
    sub_data_demo.lapse_date < max_lapse_date]
```

# Date differences

- Step 1: Filter to the relevant set of users
- **Step 2: Calculate the time between a users lapse and subscribed dates**

```
# How many days passed before the user subscribed
sub_time = conv_sub_data.subscription_date - conv_sub_data.lapse_date
# Save this value in our dataframe
conv_sub_data['sub_time'] = sub_time
```

# Date components

- Step 1: Filter to the relevant set of users
- Step 2: Calculate the time between a users lapse and subscribed dates
- **Step 3: Convert the `sub_time` from a `timedelta` to an `int`**

```
# Extract the days field from the sub_time  
conv_sub_data['sub_time'] = conv_sub_data.sub_time.dt.days
```

# Conversion rate calculation

```
# filter to users who have did not subscribe in the right window
conv_base = conv_sub_data[(conv_sub_data.sub_time.notnull()) | \
    (conv_sub_data.sub_time > 7)]
total_users = len(conv_base)
```

```
total_subs = np.where(conv_sub_data.sub_time.notnull() & \
    (conv_base.sub_time <= 14), 1, 0)
total_subs = sum(total_subs)
```

```
conversion_rate = total_subs / total_users
```

```
0.0095877277085330784
```



# Parsing dates - on import

```
pandas.read_csv(...,  
    parse_dates=False,  
    infer_datetime_format=False,  
    keep_date_col=False,  
    date_parser=None,  
    dayfirst=False,...)
```

```
customer_demographics = pd.read_csv('customer_demographics.csv',  
    parse_dates=True,  
    infer_datetime_format=True)
```

	uid	reg_date	device	gender	country	age
0	54030035.0	2017-06-29	and	M	USA	19
1	72574201.0	2018-03-05	iOS	F	TUR	22
2	64187558.0	2016-02-07	iOS	M	USA	16
3	92513925.0	2017-05-25	and	M	BRA	41
4	99231338.0	2017-03-26	iOS	M	FRA	59

# Parsing dates - manually

```
pandas.to_datetime(arg, errors='raise', ..., format=None, ...)
```

## strftime

*1993-01-27* -- "%Y-%m-%d"

*05/13/2017 05:45:37* -- "%m/%d/%Y %H:%M:%S"

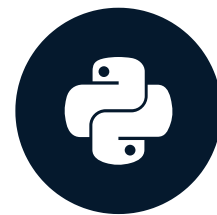
*September 01, 2017* -- "%B %d, %Y"

# Let's practice!

CUSTOMER ANALYTICS AND A/B TESTING IN PYTHON

# Creating time series graphs with matplotlib

CUSTOMER ANALYTICS AND A/B TESTING IN PYTHON



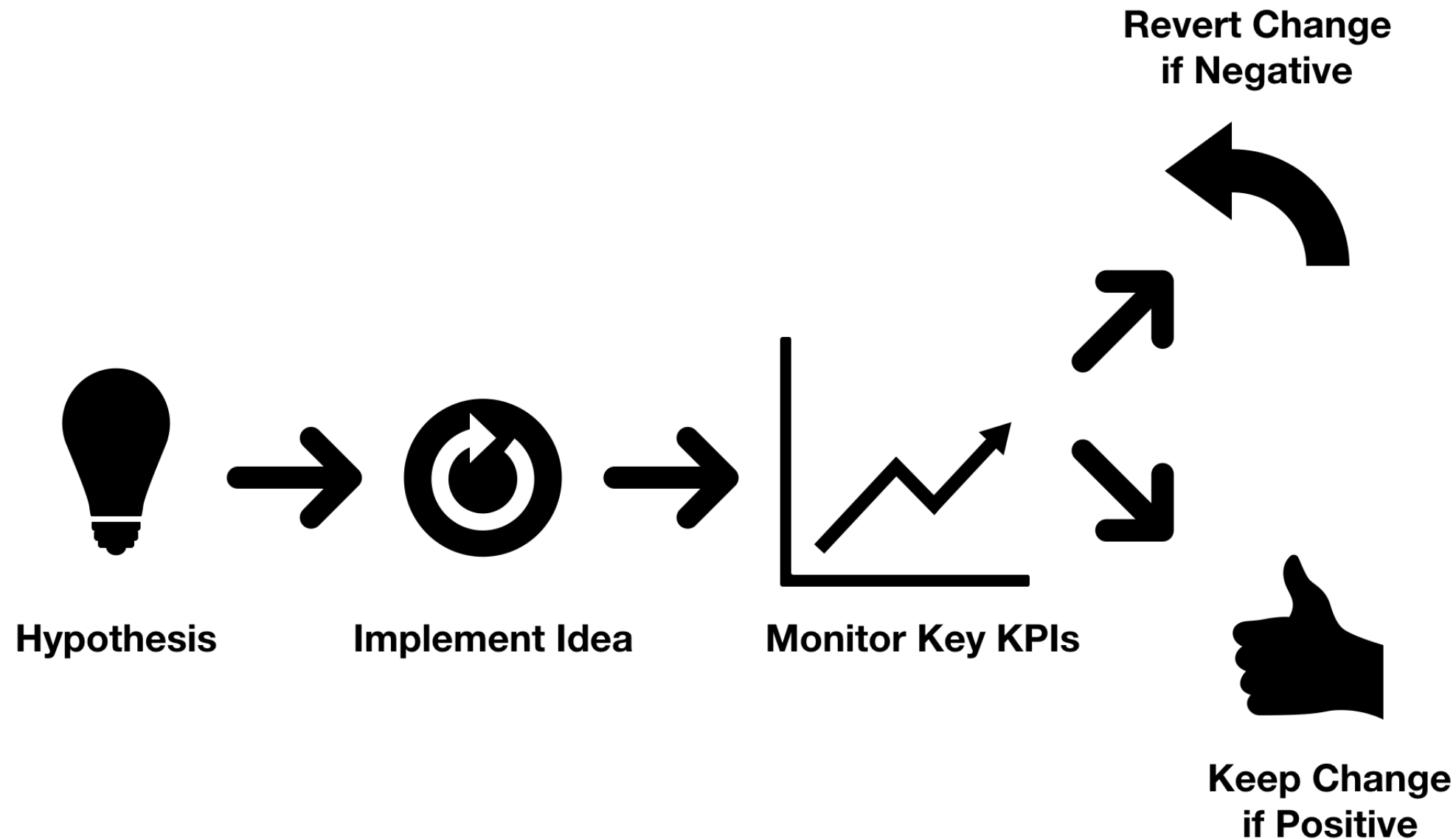
**Ryan Grossman**  
Data Scientist, EDO

# Conversion rate over time

## Useful Ways to Explore Metrics

- By user type
- Over time

# Monitoring the impact of changes



# Week one conversion rate by day

```
import pandas as pd
from datetime import timedelta

# The maximum date in our dataset
current_date = pd.to_datetime('2018-03-17')

# Limit to users who have had a week to subscribe
max_lapse_date = current_date - timedelta(days=7)
conv_sub_data = sub_data_demo[
    sub_data_demo.lapse_date < max_lapse_date]

# Calculate how many days it took the user to subscribe
conv_sub_data['sub_time'] = (conv_sub_data.subscription_date
    - conv_sub_data.lapse_date.dt.days)
```

# Conversion Rate by Day

- The lapse date is the first day a user is eligible to subscribe

```
# Find the conversion rate for each daily cohort
conversion_data = conv_sub_data.groupby(
    by=['lapse_date'], as_index=False
).agg({'sub_time': [gc7]})

# Clean up the dataframe columns
conversion_data.head()
```

```
lapse_date  sub_time
0    2017-09-01    0.224775
1    2017-09-02    0.223749
...
```



# Plotting Daily Conversion Rate

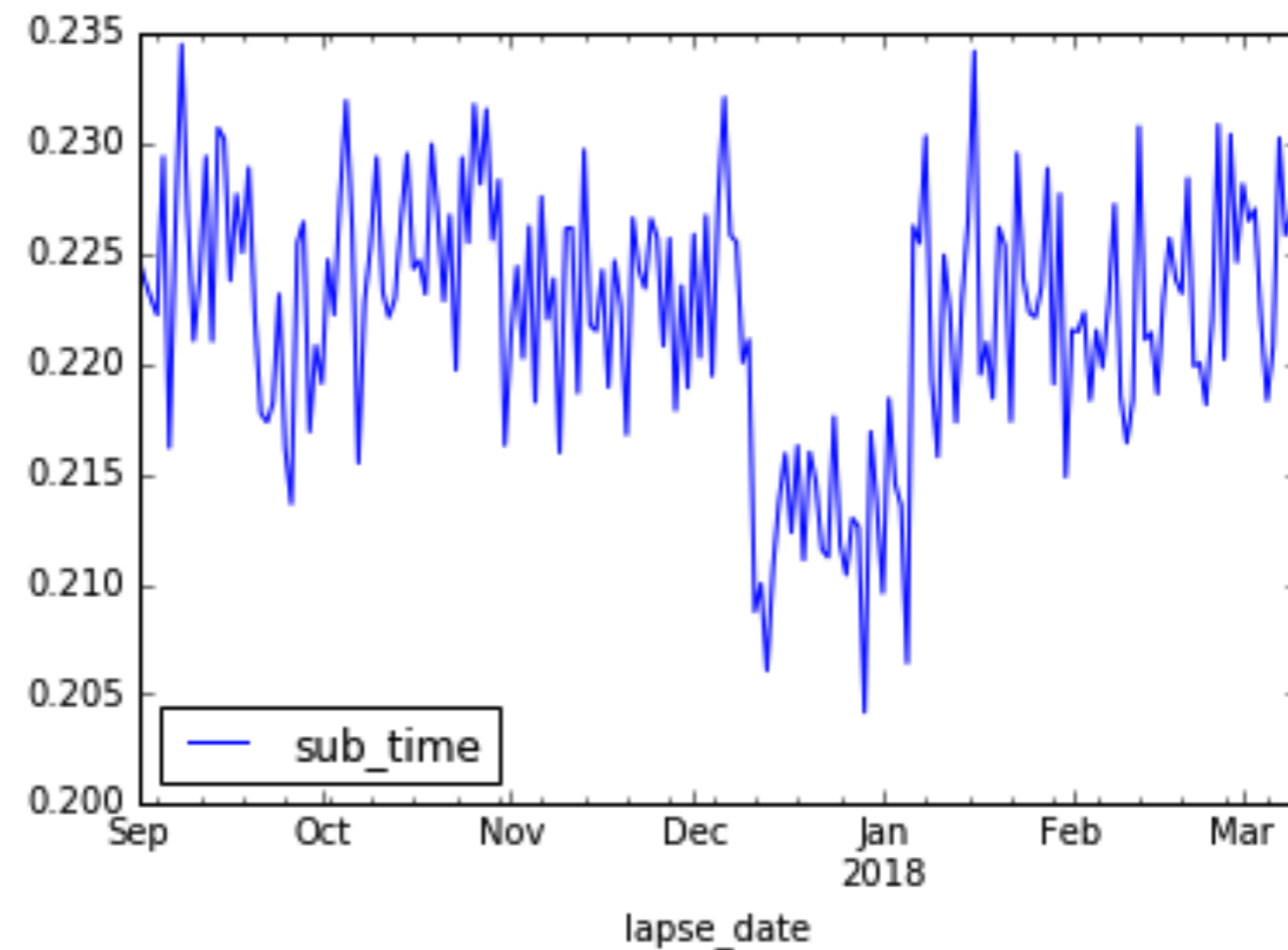
- Use the `.plot()` method to generate graphs of DataFrames

```
# Convert the lapse_date value from a string to a
# datetime value
conversion_data.lapse_date = pd.to_datetime(
    conversion_data.lapse_date
)

# Generate a line graph of the average conversion rate
# for each user registration cohort
conversion_data.plot(x='lapse_date', y='sub_time')
```

# Plotting Daily Conversion Rate

```
# Print the generated graph to the screen  
plt.show()
```



# Trends in different cohorts

- See how changes interact with different groups
- Compare users of different genders
- Evaluate the impact of a change across regions
- See the impact for different devices

# Trends across time and user groups

- Is the holiday dip consistent across different countries?

```
conversion_data.head()
```

- Conversion rate by day, broken out by our top selling countries

	lapse_date	country	sub_time
0	2017-09-01	BRA	0.184000
1	2017-09-01	CAN	0.285714
2	2017-09-01	DEU	0.276119
3	2017-09-01	FRA	0.240506
4	2017-09-01	TUR	0.161905

# Conversion rate by country

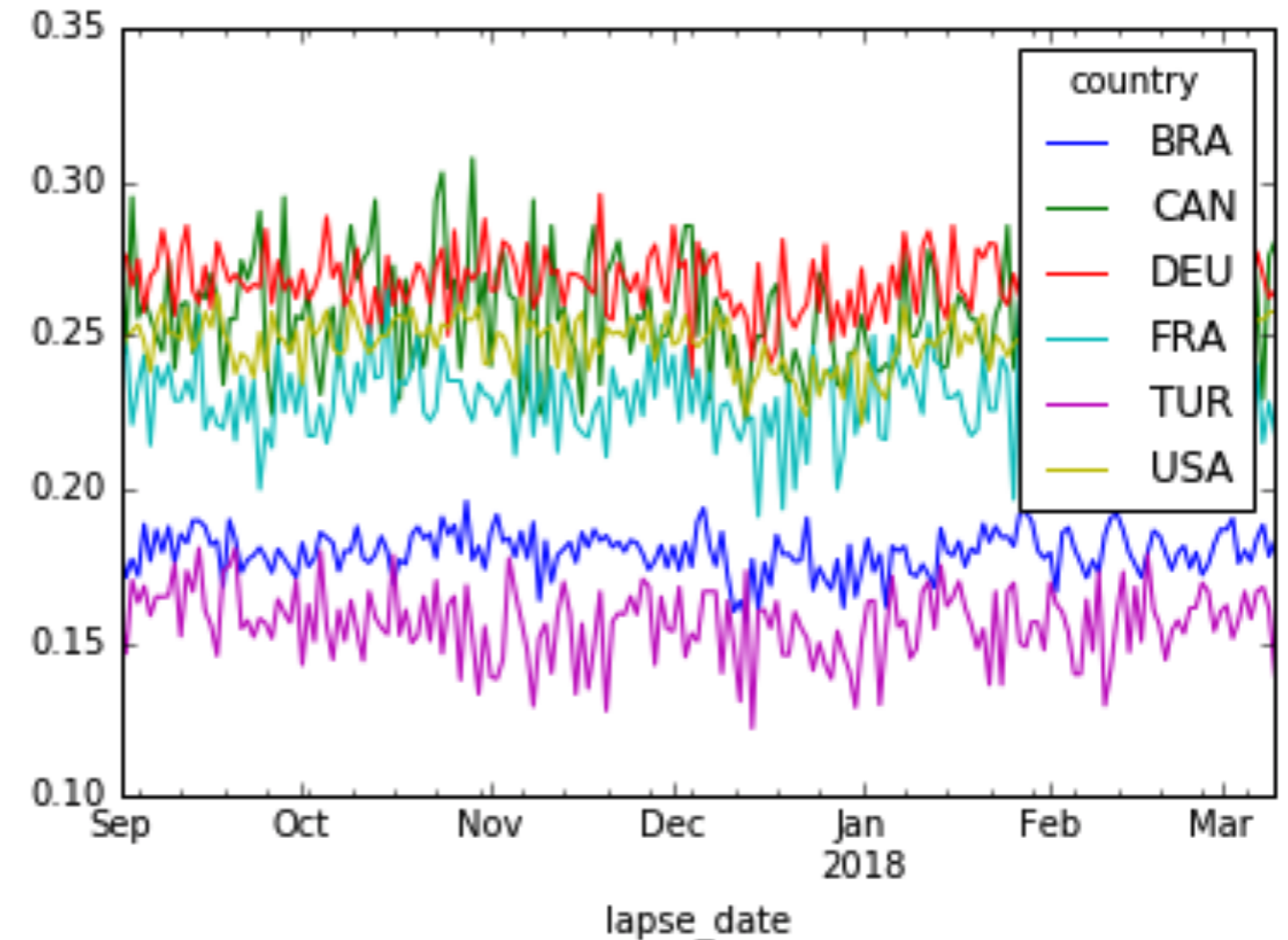
```
# Break out our conversion rate by country
reformatted_cntry_data = pd.pivot_table(
    conversion_data, # dataframe to reshape
    values=['sub_time'], # Our primary value
    columns=['country'], # what to break out by
    index=['reg_date'], # the value to use as rows
    fill_value=0
)
```

lapse_date	BRA	CAN	DEU	...
2017-09-01	0.184000	0.285714	0.276119	...
2017-09-02	0.171296	0.244444	0.276190	...
2017-09-03	0.177305	0.295082	0.266055	...

# Plotting trends in different cohorts

```
# Plot each countries conversion rate
reformatted_cntry_data.plot(
    x='reg_date',
    y=['BRA', 'FRA', 'DEU', 'TUR', 'USA', 'CAN']
)
```

```
plt.show()
```

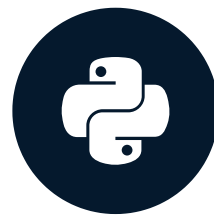


# Let's practice!

CUSTOMER ANALYTICS AND A/B TESTING IN PYTHON

# Understanding and visualizing trends in customer data

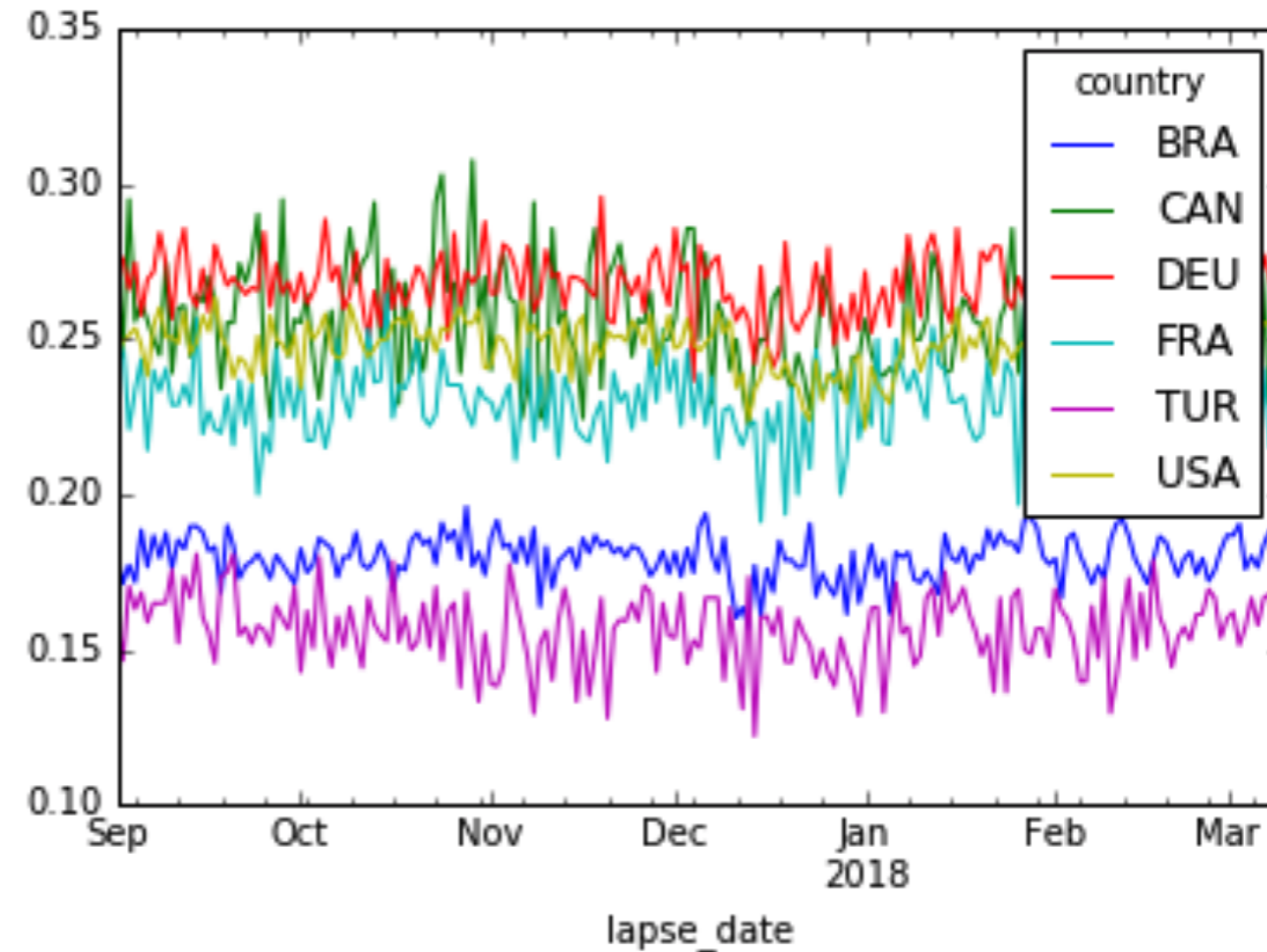
CUSTOMER ANALYTICS AND A/B TESTING IN PYTHON



**Ryan Grossman**  
Data Scientist, EDO



# Further techniques for uncovering trends



# Subscribers Per Day

```
# Find the days-to-subscribe of our loaded usa subs data set
usa_subscriptions['sub_day'] = (usa_subscriptions.sub_date -
    usa_subscriptions.lapse_date).dt.days

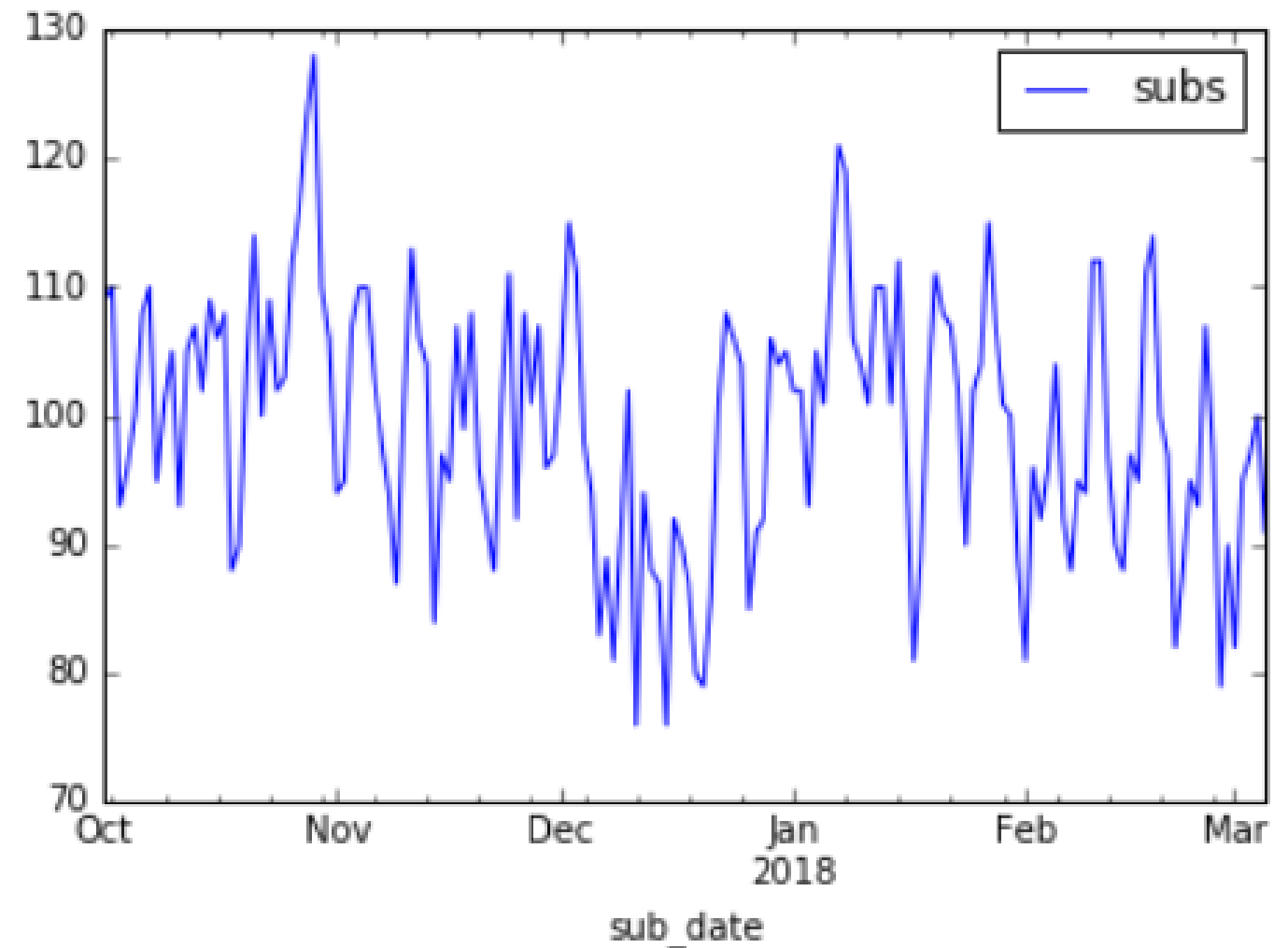
# Filter out those who subscribed in the past week
usa_subscriptions = usa_subscriptions[usa_subscriptions.sub_day <= 7]

# Find the total subscribers per day
usa_subscriptions = usa_subscriptions.groupby(
    by=['sub_date'], as_index = False
).agg({'subs': ['sum']})
```

# Weekly seasonality and our pricing change

```
# plot USA subscribers per day
usa_subscriptions.plot(x='sub_date', y='subs')
plt.show()
```

- **Weekly Seasonality:** Trends following the day of the week
  - Potentially more likely to subscribe on the weekend
  - Seasonality can hide larger trends...the impact of our price change?



# Correcting for seasonality with trailing averages

- **Trailing Average:** smoothing technique that averages over a **lagging window**
  - Reveal hidden trends by **smoothing** out seasonality
  - Average across the period of seasonality
  - 7-day window to smooth weekly seasonality
  - Average out day level effects to produce the average week effect

# Calculating Trailing Averages

- Calculate the rolling average over the USA subscribers data with `.rolling()`
  - Call this on the `Series` of interest
  - `window` : Data points to average
  - `center` : If true set the average at the center of the window

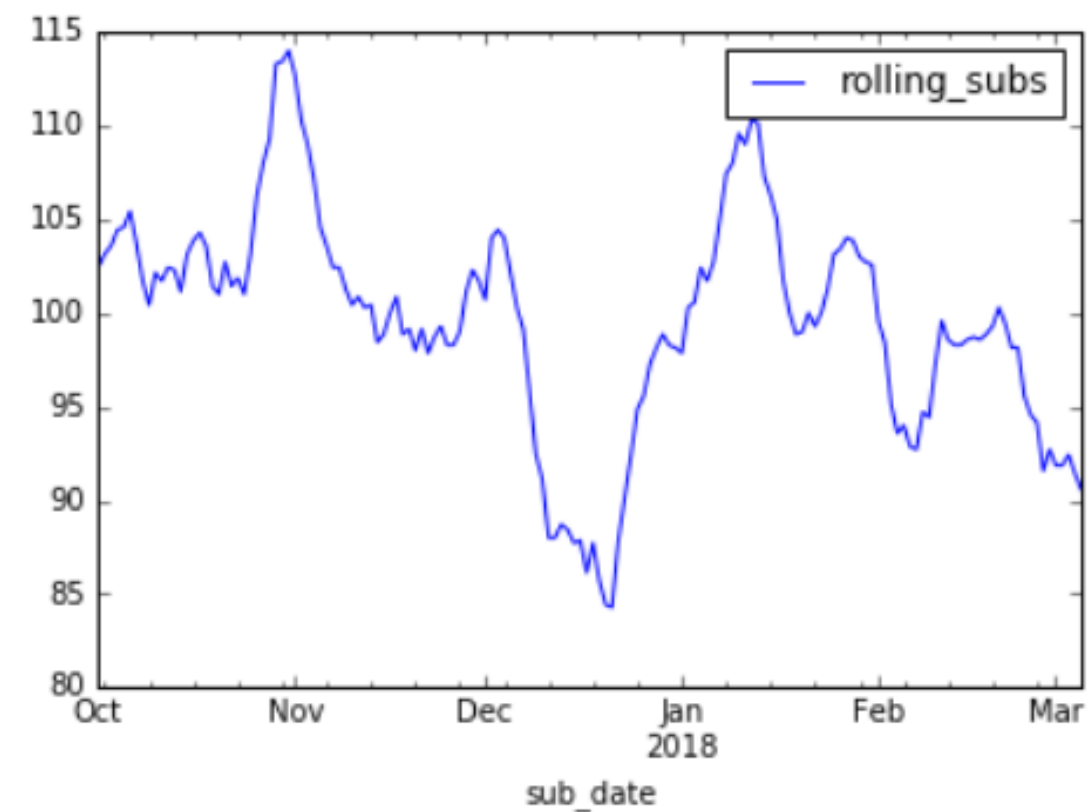
```
# calling rolling on the "subs" Series
rolling_subs = usa_subscriptions.subs.rolling(
    # How many data points to average over
    window=7,
    # Specify to average backwards
    center=False
)
```

# Smoothing our USA subscription data

```
# find the rolling average
usa_subscriptions['rolling_subs']
    = rolling_subs.mean()
usa_subscriptions.tail()
```

sub_date	subs	rolling_subs
2018-03-14	89	94.714286
2018-03-15	96	95.428571
2018-03-16	102	96.142857

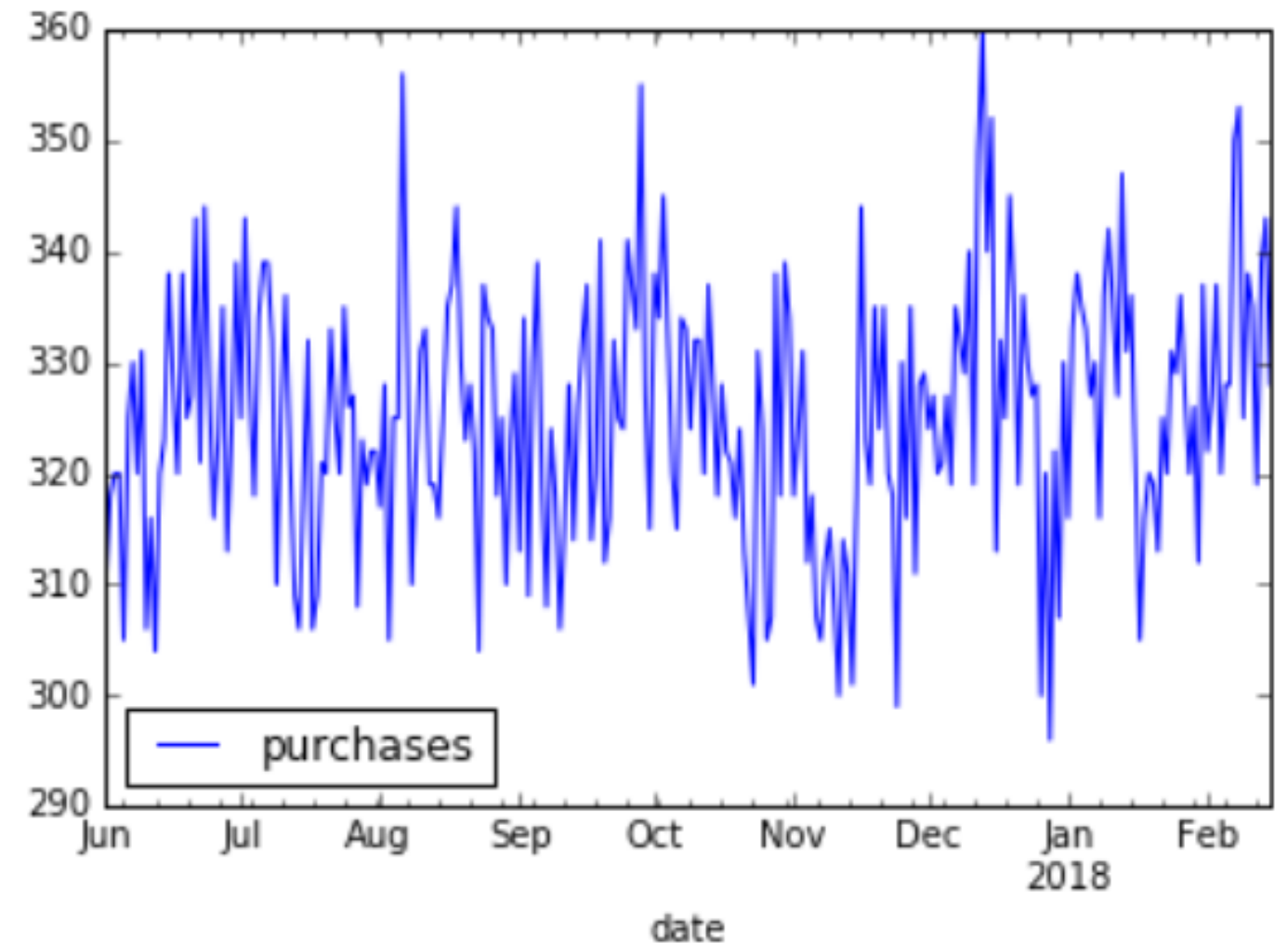
- `.rolling` like `groupby` specifies a grouping of data points
- We still need to calculate a summary over this group (e.g. `.mean()` )



# Noisy data - Highest SKU purchases by date

- **Noisy Data:** data with high variation over time

```
# Load a dataset of our highest sku purchases
high_sku_purchases = pd.read_csv(
    'high_sku_purchases.csv',
    parse_dates=True,
    infer_datetime_format=True
)
# Plot the count of purchases by day of purchase
high_sku_purchases.plot(x='date', y='purchases')
plt.show()
```



# Smoothing with an exponential moving average

- **Exponential Moving Average:** Weighted moving (rolling) average
  - Weights more recent items in the window more
  - Applies weights according to an exponential distribution
  - Averages back to a central trend without masking any recent movements



# Smoothed purchases by date

- `.ewm()` : exponential weighting function
- `span` : Window to apply weights over

```
# Calculate the exp. avg. over our high sku  
# purchase count  
exp_mean = high_sku_purchases.purchases.ewm(  
    span=30)
```

```
# Find the weighted mean over this period  
high_sku_purchases['exp_mean'] = exp_mean.mean()
```

## High Sku Purchase Data



# Summary - Data Smoothing Techniques

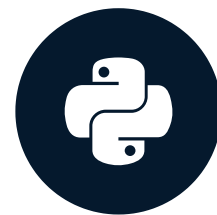
- **Trailing Average:**
  - Smooths seasonality by averaging over the periodicity
- **Exponential Moving Average:**
  - Reveals trends by pulling towards the central tendency
  - Weights the more recent values relative to the window more heavily
- You can use `.rolling()` and `.ewm()` for many more methods of smoothing

# Let's practice!

CUSTOMER ANALYTICS AND A/B TESTING IN PYTHON

# Events and releases

CUSTOMER ANALYTICS AND A/B TESTING IN PYTHON



**Ryan Grossman**  
Data Scientist, EDO

# Exploratory analysis - issues in our ecosystem



# Visualizing the drop in conversion rate (3 Years)

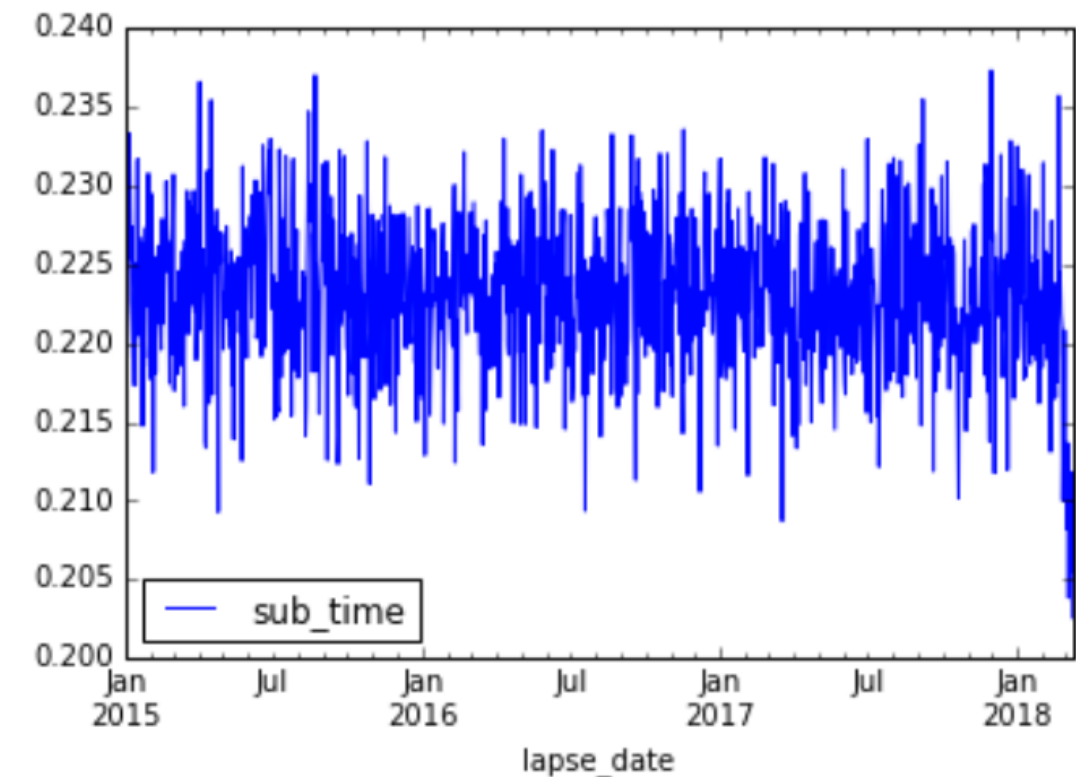
```
import pandas as pd
import matplotlib.pyplot as plt

# Remove users who lapsed within the past week
conv_sub_data = sub_data_demo[
    sub_data_demo.lapse_date <= max_lapse_date]

# Calculate week one conversion rate by lapse date
sub_time = (conv_sub_data.subscription_date -
            conv_sub_data.lapse_date).dt.days
conv_sub_data['sub_time'] = sub_time

conversion_data = conv_sub_data.groupby(
    by=['lapse_date'], as_index=False
).agg({'sub_time': [gc7]})
```

```
# Plot our conversion rate over time
conversion_data.plot()
plt.show()
```



# Visualizing the drop in conversion rate (6 Months)

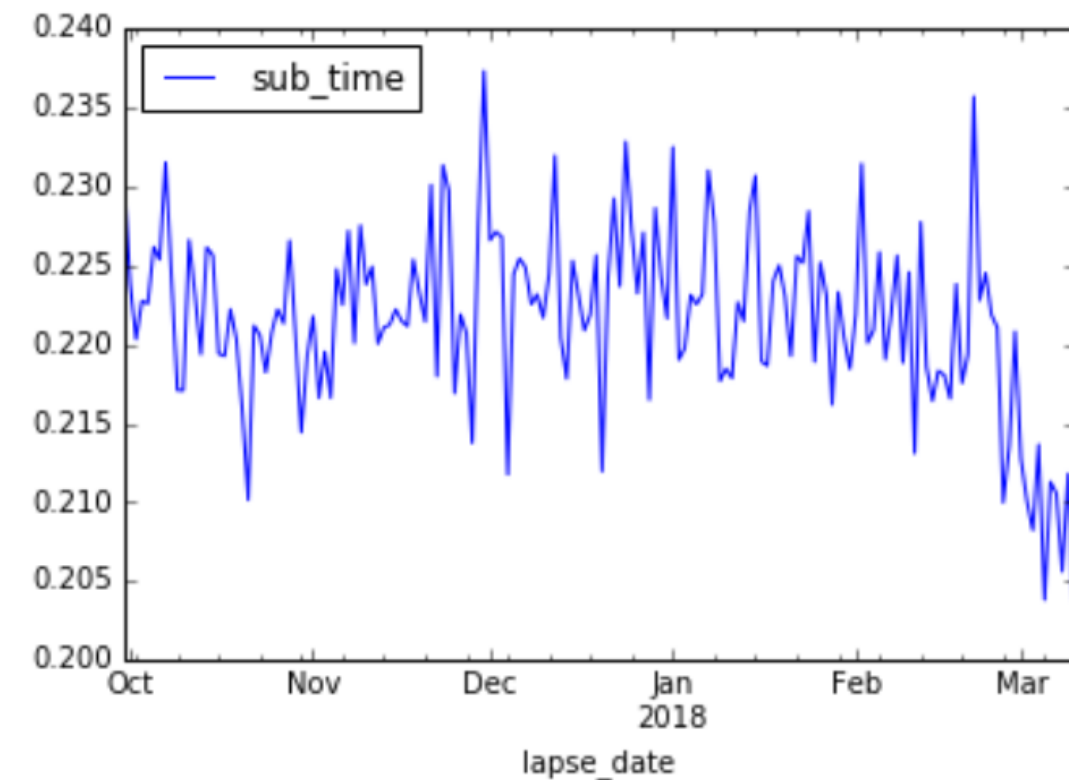
```
# Find the date boundaries to limit our data by
current_date = pd.to_datetime('2018-03-17')

# 6 * 28 to represent the past 6 months
start_date = current_date - timedelta(days=(6*28))

# A mask for our conversion rate data
conv_filter = (
    conversion_data.lapse_date >= start_date) &
    (conversion_data.lapse_date <= current_date)
)

# Filter our conversion rate data
con_data_filt = conversion_data[conv_filter]
```

```
conv_data_filt.plot(x='lapse_date', y='sub_time')
plt.show()
```



# Investigating the conversion rate drop

- Is this drop impacting all users or just a specific cohort
- This could provide clues on what the issue may be
- Ecosystems within our data
  - Distinct countries
  - Specific device (Android or iOS)



# Splitting our data by country and device

```
# After filtering and calculating daily conversion...

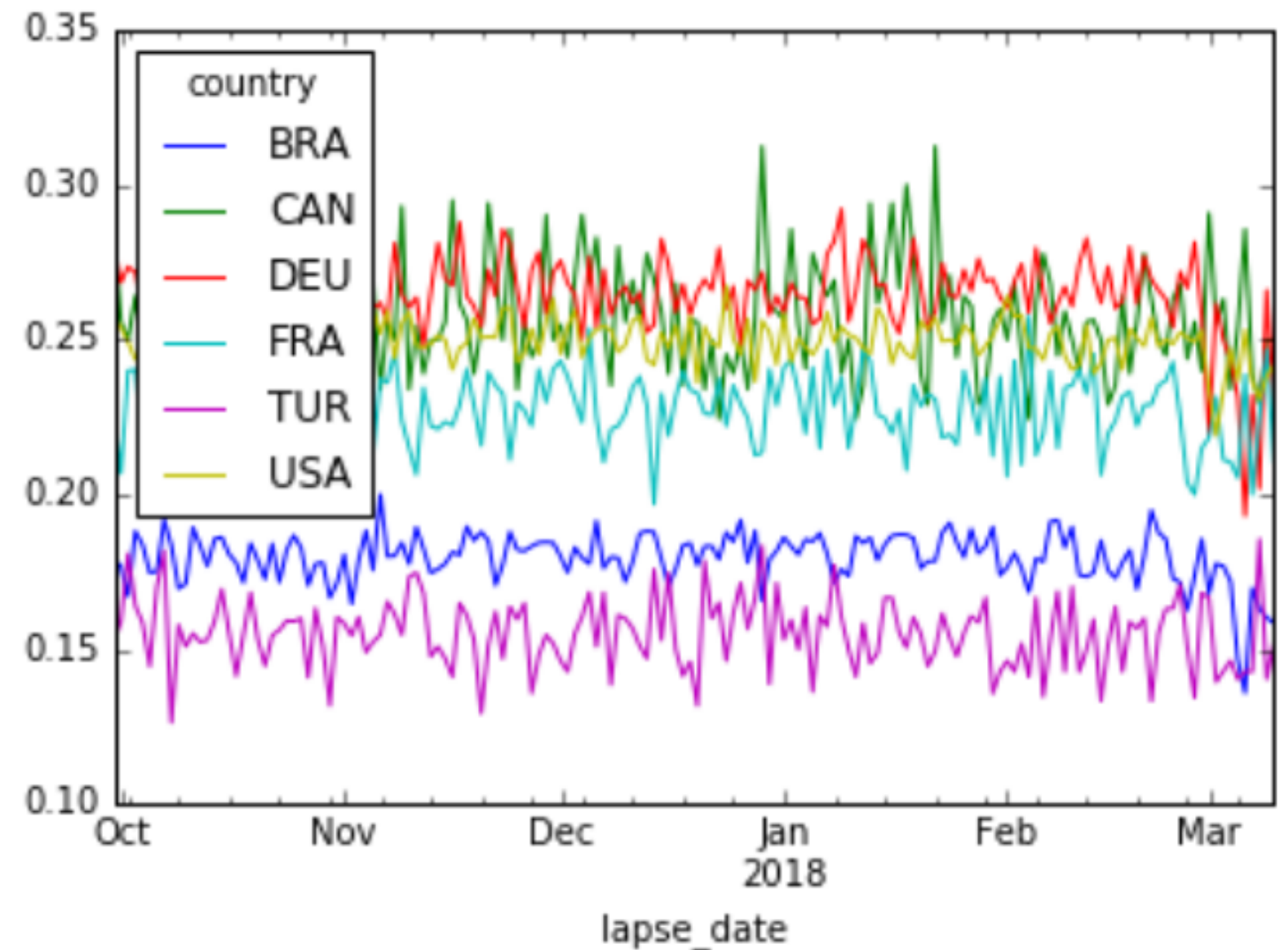
# Pivot the results to have one column per country
conv_data_cntry = pd.pivot_table(
    conv_data_cntry, values=['sub_time'],
    columns=['country'], index=['lapse_date'], fill_value=0
)

...

# Pivot the results to have one column per device
conv_data_dev = pd.pivot_table(
    conv_data_dev, values=['sub_time'],
    columns=['device'], index=['lapse_date'], fill_value=0
)
```

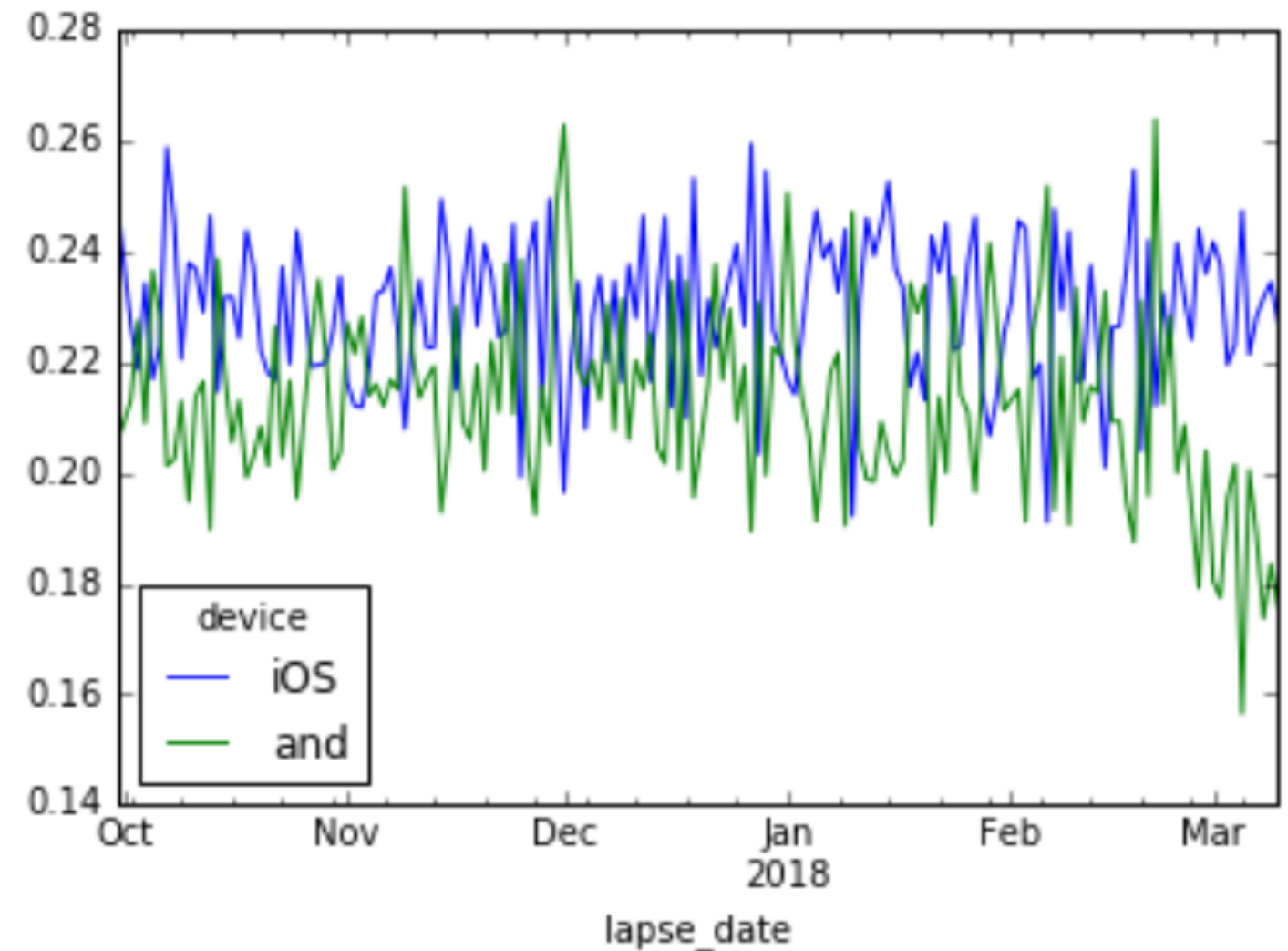
# Breaking out by Country

- All countries experience the drop
- It is most pronounced in Brazil & Turkey
  - Our two most android heavy countries



# Breaking out by Device

- The drop only appears on Android devices



# Annotating datasets

- `events` : Holidays and events impacting user behavior

```
events = pd.read_csv('events.csv')
```

- `releases` : iOS and Android software releases

```
releases = pd.read_csv('releases.csv')  
releases.head()
```

Date	Event
2018-03-14	iOS Release
2018-03-03	Android Release
2018-01-13	iOS Release
2018-01-15	Android Release

# Plotting annotations - events

- `plt.axvline()` : Plots vertical line at the x-intercept
  - `color` : Specify the color of the plotted line
  - `linestyle` : The type of line to plot

```
# Plot the conversion rate trend per device
conv_data_dev.plot(
    x=['lapse_date'], y=['iOS', 'and']
)

# Iterate through the events and plot each one
events.Date = pd.to_datetime(events.Date)
for row in events.iterrows():
    tmp = row[1]
    plt.axvline(
        x=tmp.Date, color='k', linestyle='--'
    )
```

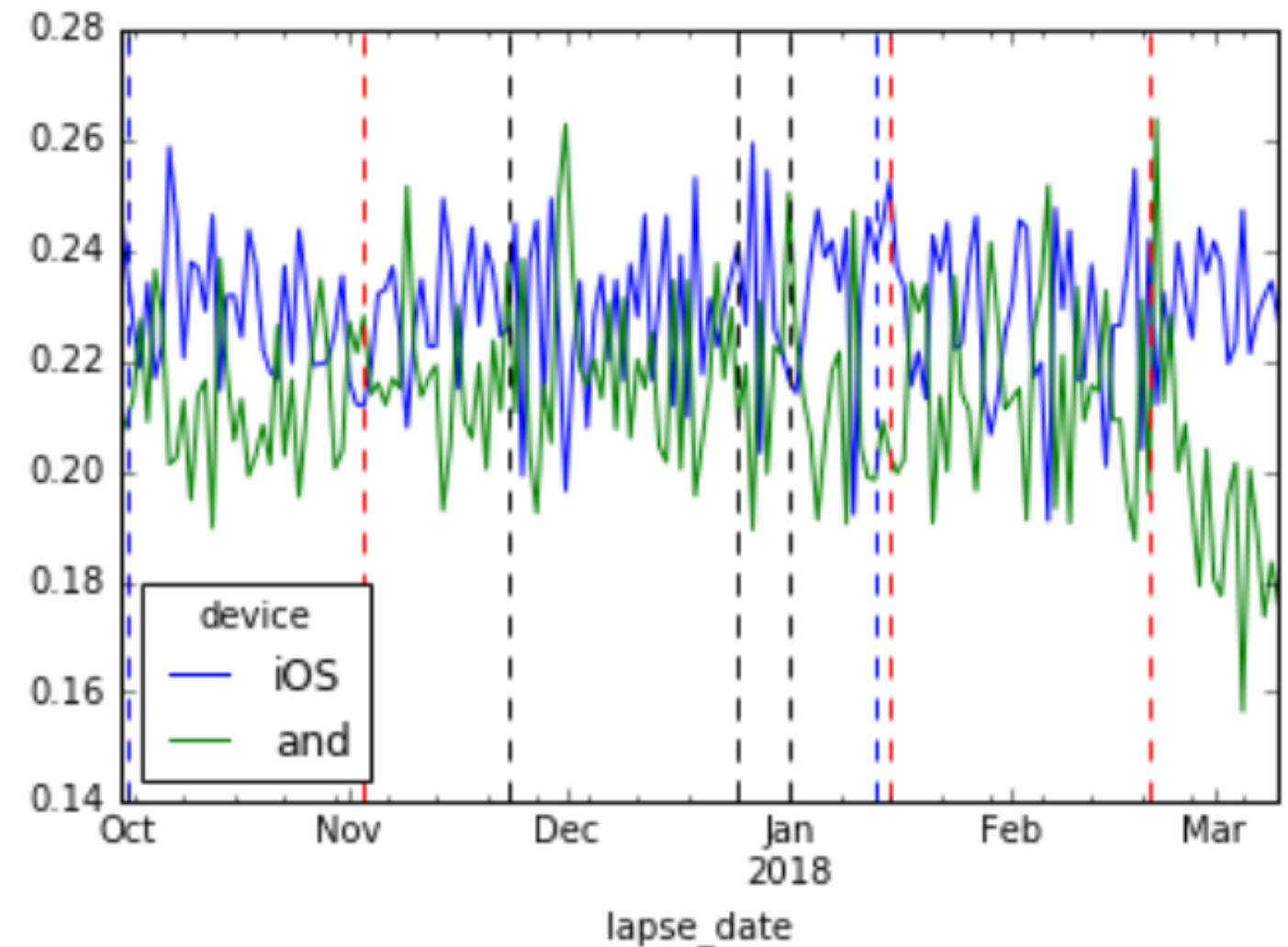
# Plotting annotations - releases

```
# Iterate through the releases and plot each one
releases.Date = pd.to_datetime(releases.Date)
for row in releases.iterrows():
    tmp = row[1]
    # plot iOS releases as a blue lines
    if tmp.Event == 'iOS Release':
        plt.axvline(x=tmp.Date, color='b', linestyle='--')

    # plot Android releases as red lines
    else:
        plt.axvline(x=tmp.Date, color='r', linestyle='--')
plt.show()
```

# Annotated conversion rate graphs

- Android release in Feb/Mar aligns with our dip in conversion rate
- This release may contain a bug impacting the conversion rate!



# Power and limitations of exploratory analysis

- Visualize data over time to uncover hidden trends
  - While useful it has its limitations
- To truly explore relationships in data we need A/B testing



# Let's practice!

CUSTOMER ANALYTICS AND A/B TESTING IN PYTHON