Case study- Analyzing Marketing Campaigns with pandas

by DataCamp

Description:

Data analysis in marketing can be used for following methods or the combination of following:

1. Analyse how a marketing campaign performed and what was the conclusion.
2. Analyse based on a new marketing channel, how users interact to a certain offer, etc.
3. A/B testing.

Code:

```
# Import pandas into the environment
import pandas as pd

# Import marketing.csv
marketing = pd.read_csv('marketing.csv')

#to gain more insights about the dataframe:

print(marketing.head())

<script.py> output:
          user_id date_served marketing_channel          variant converted  ...    age_group date_subscribed date_canceled subscribing_chan
    0  a100000029      1/1/18         House Ads  personalization      True  ...   0-18 years          1/1/18           NaN           House 
    1  a100000030      1/1/18         House Ads  personalization      True  ...  19-24 years          1/1/18           NaN           House 
    2  a100000031      1/1/18         House Ads  personalization      True  ...  24-30 years          1/1/18           NaN           House 
    3  a100000032      1/1/18         House Ads  personalization      True  ...  30-36 years          1/1/18           NaN           House 
    4  a100000033      1/1/18         House Ads  personalization      True  ...  36-45 years          1/1/18           NaN           House 

    [5 rows x 12 columns]

print(marketing.describe())

user_id date_served marketing_channel  variant converted  ...    age_group date_subscribed date_canceled subscribing_channel is_retained
count        10037       10021             10022    10037     10022  ...        10037            1856           577                1856
unique        7309          31                 5        2         2  ...            7              31           115                   5
top     a100000882     1/15/18         House Ads  control     False  ...  19-24 years         1/16/18        4/2/18           Instagram
freq            12         789              4733     5091      8946  ...         1682             163            15                 600

[4 rows x 12 columns]

# Check column data types and non-missing values
print(marketing.info())

Data columns (total 12 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   user_id              10037 non-null  object
 1   date_served          10021 non-null  object
 2   marketing_channel    10022 non-null  object
 3   variant              10037 non-null  object
 4   converted            10022 non-null  object
 5   language_displayed   10037 non-null  object
 6   language_preferred   10037 non-null  object
 7   age_group            10037 non-null  object
 8   date_subscribed      1856 non-null   object
 9   date_canceled        577 non-null    object
 10  subscribing_channel  1856 non-null   object
 11  is_retained          1856 non-null   object
dtypes: object(12)
```

```
# Convert is_retained to a boolean
marketing['is_retained'] = marketing['is_retained'].astype('bool')

# Check the data type of is_retained
print(marketing['is_retained'].dtype)
```

Adding new columns that derive information from existing data or based on domain knowledge is known as *Feature Engineering*. Even in relatively simple datasets, there are always new characteristics you could pull out to create a more in-depth analysis.

One of the most critical skills a data scientist needs to learn is how to identify opportunities for feature engineering.

In this exercise, you will add two columns to `marketing`:

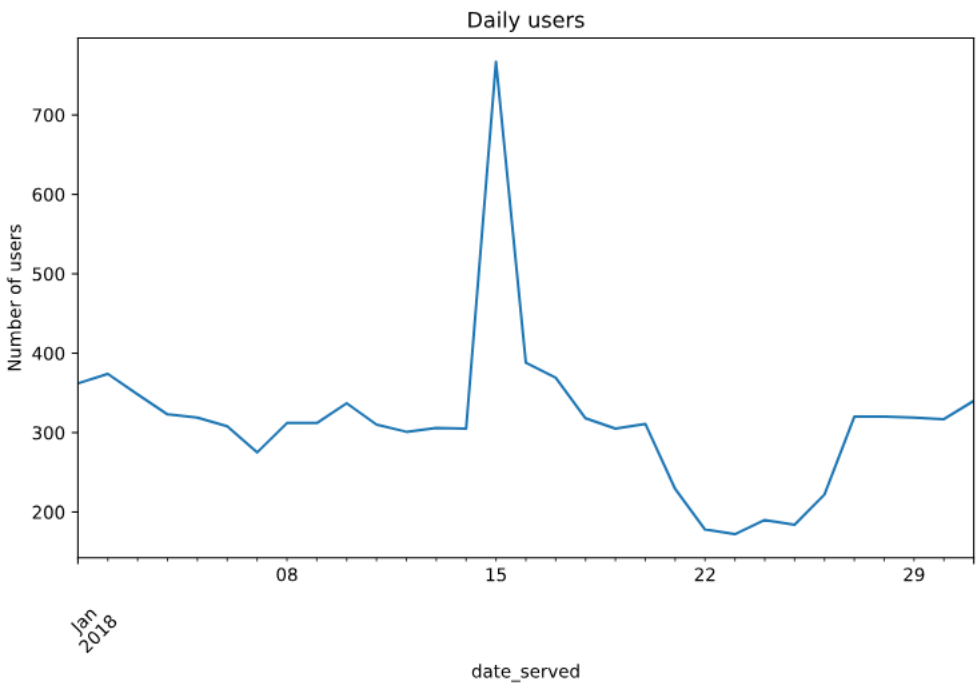
- `channel_code`: represents the numeric value of the subscribing channel
- `is_correct_lang`: conveys whether the ad was shown to the user in their preferred language

```
# Mapping for channels
channel_dict = {"House Ads": 1, "Instagram": 2,
                "Facebook": 3, "Email": 4, "Push": 5}

# Map the channel to a channel code
marketing['channel_code'] = marketing['subscribing_channel'].map(channel_dict)

# Import numpy
import numpy as np

# Add the new column is_correct_lang
marketing['is_correct_lang'] = np.where(marketing['language_preferred'] == marketing['language_displayed'], 'Yes', 'No' )
```

Currently, the date columns in the `marketing` DataFrame are being incorrectly read as objects. We need to convert these columns to date columns to be able to use Python and pandas' robust date manipulation and formatting capabilities.

```
# Import pandas into the environment
import pandas as pd

# Import marketing.csv with date columns
marketing = pd.read_csv('marketing.csv', parse_dates= ['date_served','date_subscribed','date_canceled'])

# Add a DoW column
marketing['DoW'] = marketing['date_subscribed'].dt.dayofweek
```

```
# Group by date_served and count number of unique user_id's
daily_users = marketing.groupby(by= 'date_served')['user_id'].nunique()

# Plot daily_subscribers
daily_users.plot()

# Include a title and y-axis label
plt.title('Daily users')
plt.ylabel('Number of users')

# Rotate the x-axis labels by 45 degrees
plt.xticks(rotation = 45)

# Display the plot
plt.show()
```



Further Analysis:

Common question: was the campaign successful?

Metrics:

1. Conversion Rate
2. Retention Rate

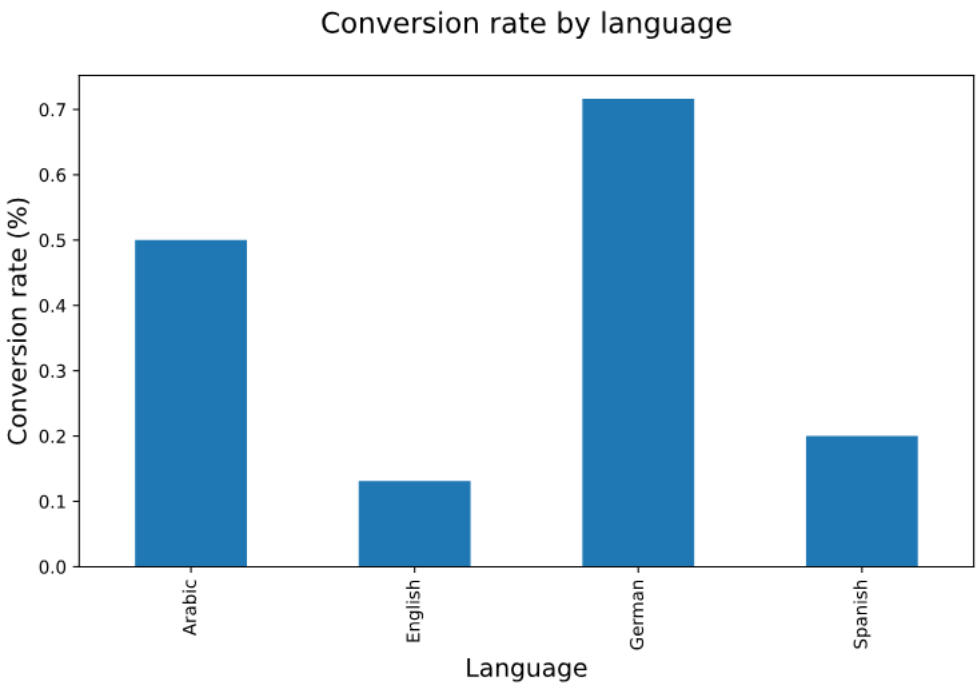
```
# Calculate the number of people we marketed to
total = marketing['user_id'].nunique()

# Calculate the number of people who subscribed
subscribers = marketing[marketing['converted']== True]['user_id'].nunique()

# Calculate the conversion rate
conversion_rate = subscribers/total
print(round(conversion_rate*100, 2), "%")

# conversion rate ~ 14.5%

# Calculate the number of subscribers
total_subscribers = marketing[marketing['converted']== True]['user_id'].nunique()

# Calculate the number of people who remained subscribed
retained = marketing[marketing['is_retained']== True]['user_id'].nunique()

# Calculate the retention rate
retention_rate = retained/total_subscribers
print(round(retention_rate*100, 2), "%")

#retention rate ~ 68.45%

#Customer Segmentation:

#retention rate by different channels
retention = marketing[marketing['is_retained']==True].groupby(by='subscribing_channel')['user_id'].nunique()
subscribers = marketing[marketing['converted']==True].groupby(by='subscribing_channel')['user_id'].nunique()
print(round(retention/subscribers *100, 2), '%')

subscribing_channel
Email        87.58
```

```
Facebook     68.78
House Ads    58.05
Instagram    68.10
Push         70.13
```

```
# Isolate english speakers
english_speakers = marketing[marketing['language_displayed'] == 'English']

# Calculate the total number of English speaking users
total = english_speakers['user_id'].nunique()

# Calculate the number of English speakers who converted
subscribers = english_speakers[english_speakers['converted']== True]['user_id'].nunique()

# Calculate conversion rate
conversion_rate = subscribers/total
print('English speaker conversion rate:', round(conversion_rate*100,2), '%')

#English speaker conversion rate: 13.13 %

# Group by language_displayed and count unique users
total = marketing.groupby(by='language_displayed')['user_id'].nunique()

# Group by language_displayed and count unique conversions
subscribers = marketing[marketing['converted']== True].groupby(by='language_displayed')['user_id'].nunique()

# Calculate the conversion rate for all languages
language_conversion_rate = (subscribers/total)
print(language_conversion_rate)

language_displayed
    Arabic     0.500
    English    0.131
    German     0.716
    Spanish    0.200

# Group by date_served and count unique users
total = marketing.groupby(by='date_served')['user_id'].nunique()

# Group by date_served and count unique converted users
subscribers = marketing[marketing['converted']== True].groupby(by='date_served')['user_id'].nunique()

# Calculate the conversion rate per day
daily_conversion_rate = subscribers/total
print(daily_conversion_rate)

date_served
2018-01-01    0.099
2018-01-02    0.099
2018-01-03    0.103
2018-01-04    0.108
2018-01-05    0.125
2018-01-06    0.114
2018-01-07    0.142
2018-01-08    0.115
2018-01-09    0.125
2018-01-10    0.119
2018-01-11    0.081
2018-01-12    0.076
2018-01-13    0.085
2018-01-14    0.085
2018-01-15    0.113
2018-01-16    0.255
2018-01-17    0.220
2018-01-18    0.091
2018-01-19    0.059
2018-01-20    0.068
2018-01-21    0.087
2018-01-22    0.124
2018-01-23    0.122
2018-01-24    0.116
2018-01-25    0.125
2018-01-26    0.090
2018-01-27    0.066
2018-01-28    0.062
```

```
2018-01-29    0.060
2018-01-30    0.066
2018-01-31    0.053
```

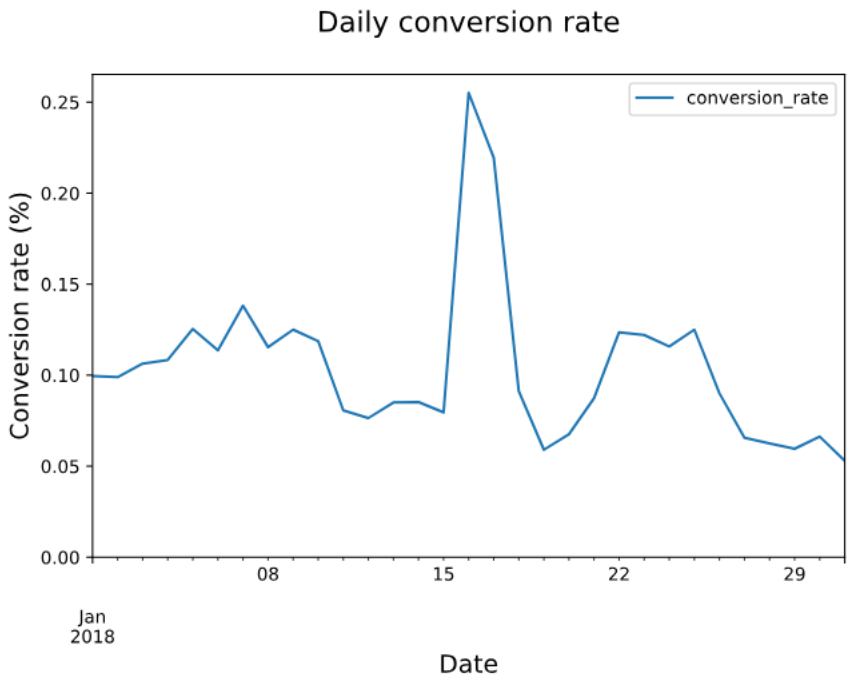
```
# Create a bar chart using language_conversion_rate DataFrame
language_conversion_rate.plot(kind='bar')

# Add a title and x and y-axis labels
plt.title('Conversion rate by language\n', size = 16)
plt.xlabel('Language', size = 14)
plt.ylabel('Conversion rate (%)', size = 14)

# Display the plot
plt.show()
```



Looking at the daily conversion rate is crucial to contextualize whether the conversion rate on a particular day was good or bad. Additionally, looking at conversion rate over time can help to surface trends such as a conversion rate that appears to be going down over time.

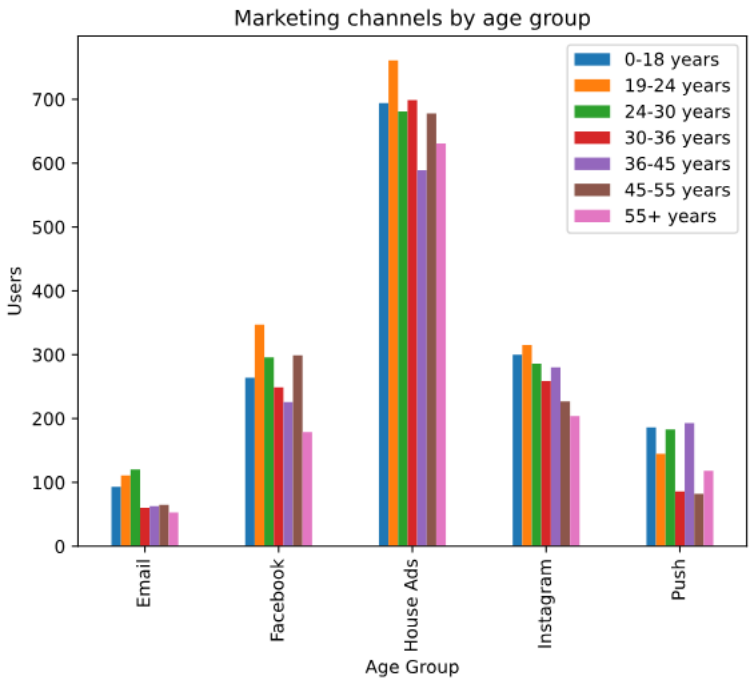
```
# Group by date_served and count unique users
total = marketing.groupby(by='date_served')['user_id'].nunique()

# Group by date_served and calculate subscribers
subscribers = marketing[marketing['converted']==True].groupby(by='date_served')['user_id'].nunique()

# Calculate the conversion rate for all languages
daily_conversion_rates = subscribers/total

# Reset index to turn the results into a DataFrame
daily_conversion_rate = pd.DataFrame(daily_conversion_rates.reset_index())

# Rename columns
daily_conversion_rate.columns = ['date_subscribed', 'conversion_rate']

# Create a line chart using daily_conversion_rate
daily_conversion_rate.plot('date_subscribed','conversion_rate')
```

```
plt.title('Daily conversion rate\n', size = 16)
plt.ylabel('Conversion rate (%)', size = 14)
plt.xlabel('Date', size = 14)

# Set the y-axis to begin at 0
plt.ylim(0)

# Display the plot
plt.show()
```



Some marketing stakeholders want to know if their marketing channels are reaching all users equally or if some marketing channels are serving specific age demographics.

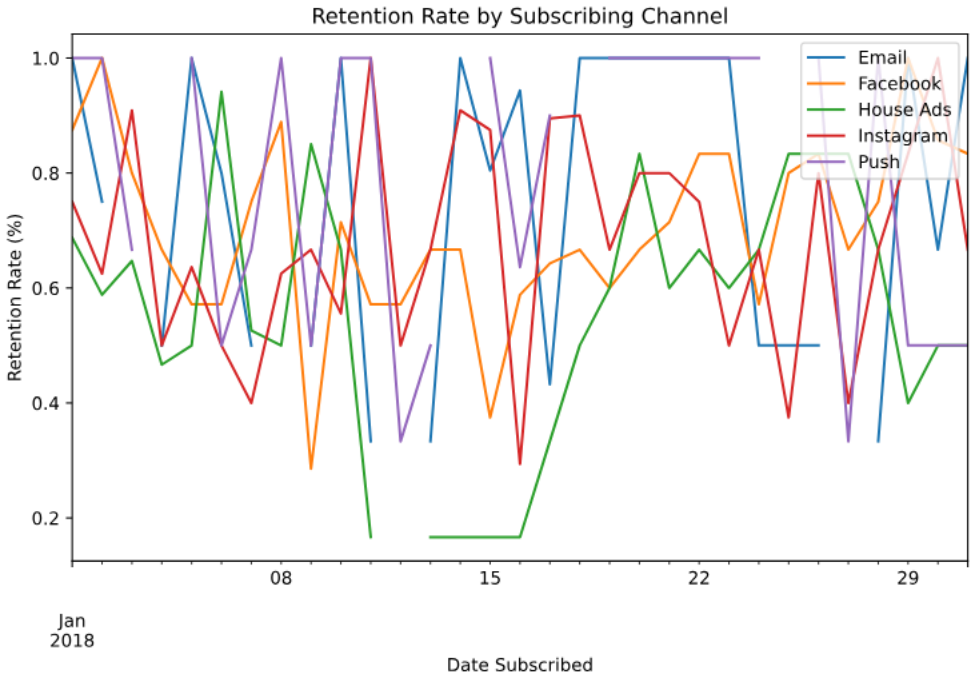
```
#groupby marketing channel & age group.
channel_age = marketing.groupby(['marketing_channel', 'age_group'])['user_id'].count()

# Unstack channel_age and transform it into a DataFrame
channel_age_df = pd.DataFrame(channel_age.unstack(level =1))

# Plot channel_age
channel_age_df.plot(kind = 'bar')
plt.title('Marketing channels by age group')
plt.xlabel('Age Group')
plt.ylabel('Users')
# Add a legend to the plot
plt.legend(loc = 'upper right',
           labels = channel_age_df.columns.values )
plt.show()
```



Stakeholders have begun competing to see whose channel had the best retention rate from the campaign. You must first determine how many subscribers came from the campaign and how many of those subscribers have stayed on the service.

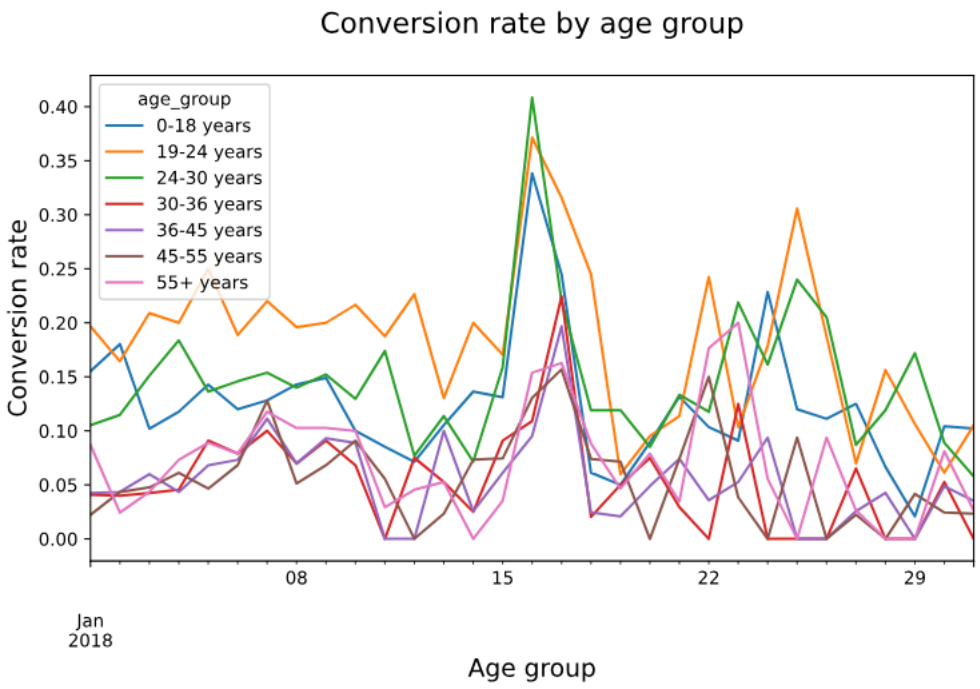
```
# Count the subs by subscribing channel and day
retention_total = marketing.groupby(['date_subscribed',
                                     'subscribing_channel'])['user_id'].nunique()

# Print results
print(retention_total.head())

date_subscribed  subscribing_channel
2018-01-01       Email                 1
                 Facebook              8
                 House Ads            16
                 Instagram             8
                 Push                  3

# Sum the retained subs by subscribing channel and date subscribed
retention_subs = marketing[marketing['is_retained']==True].groupby(['date_subscribed',
                                      'subscribing_channel'])['user_id'].nunique()

# Print results
print(retention_subs.head())

date_subscribed  subscribing_channel
    2018-01-01       Email                 1
                     Facebook              7
                     House Ads            11
                     Instagram             6
                     Push                  3

# Divide retained subscribers by total subscribers
retention_rate = retention_subs/retention_total
retention_rate_df = pd.DataFrame(retention_rate.unstack(level=1))

# Plot retention rate
retention_rate_df.plot()

# Add a title, x-label, y-label, legend and display the plot
plt.title('Retention Rate by Subscribing Channel')
plt.xlabel('Date Subscribed')
plt.ylabel('Retention Rate (%)')
```

```
plt.legend(loc='upper right', labels= retention_rate_df.columns.values)
plt.show()
```



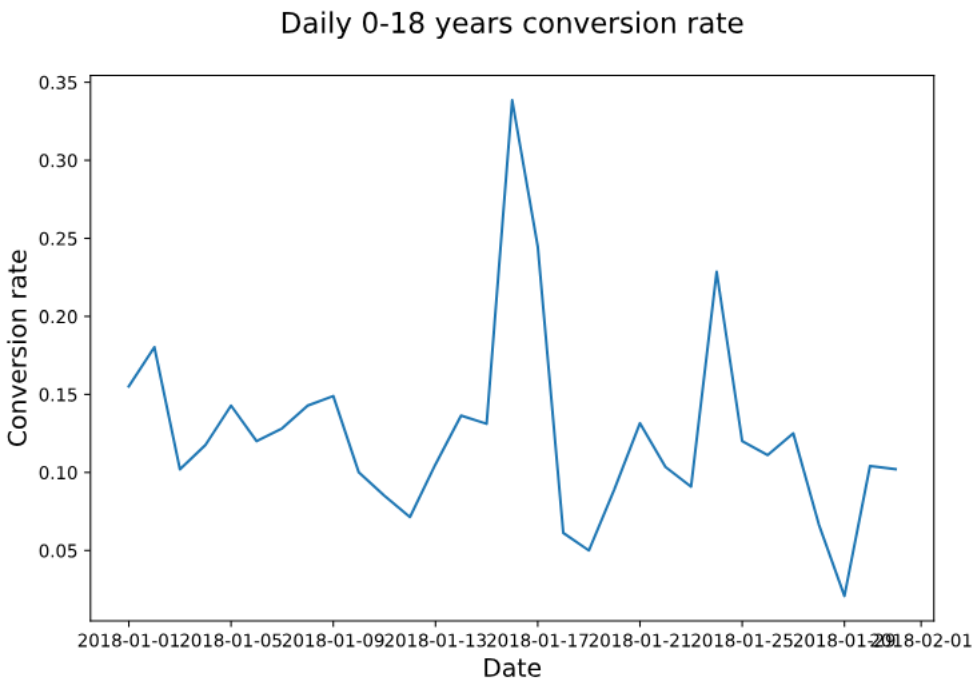
You've been doing a lot of repetitive calculations. Anytime you notice repetition in your work, consider automation. DataFrame will remain the same over time, you can build a function to enable you to calculate conversion rate across any sub-segment you want on the fly.

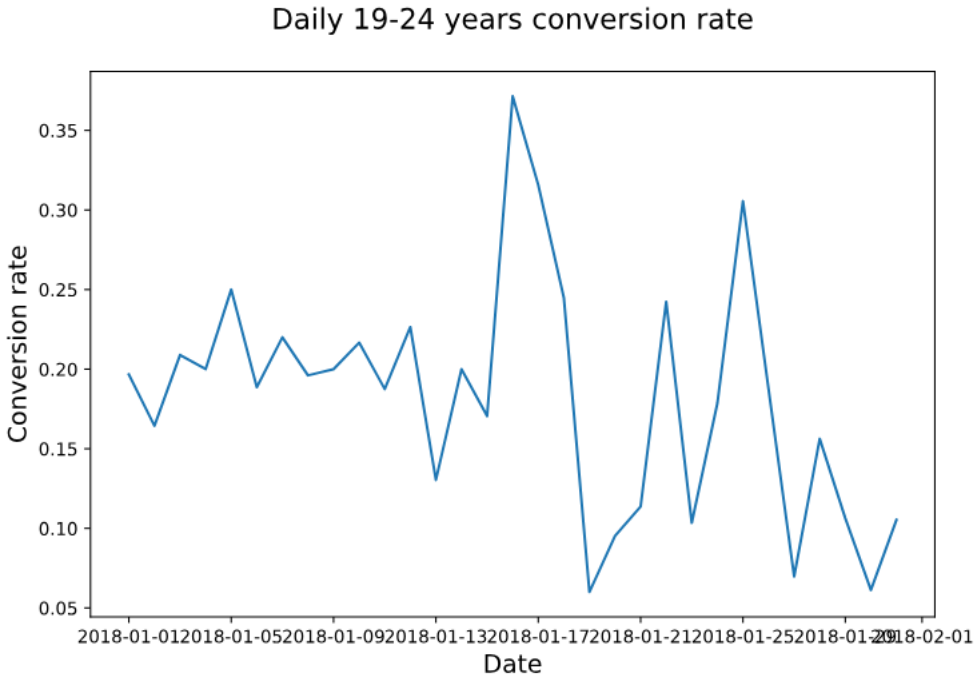
```
def conversion_rate(dataframe, column_names):
    # Total number of converted users
    column_conv = dataframe[dataframe['converted']==True].groupby(column_names)['user_id'].nunique()

    # Total number users
    column_total =   dataframe.groupby(column_names)['user_id'].nunique()

    # Conversion rate
    conversion_rate = column_conv/column_total

    # Fill missing values with 0
    conversion_rate = conversion_rate.fillna(0)
    return conversion_rate

#test the function

# Calculate conversion rate by age_group
age_group_conv = conversion_rate(marketing, ['date_served', 'age_group'])
print(age_group_conv)

# Unstack and create a DataFrame
age_group_df = pd.DataFrame(age_group_conv.unstack(level=1))

# Visualize conversion by age_group
age_group_df.plot()
plt.title('Conversion rate by age group\n', size = 16)
plt.ylabel('Conversion rate', size = 14)
plt.xlabel('Age group', size = 14)
plt.show()
```

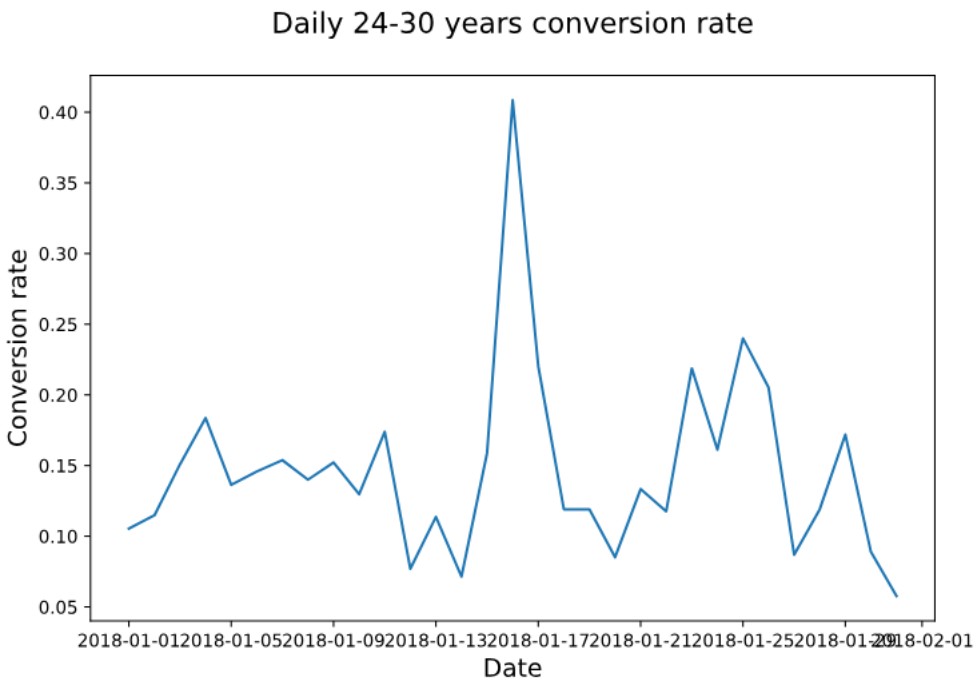


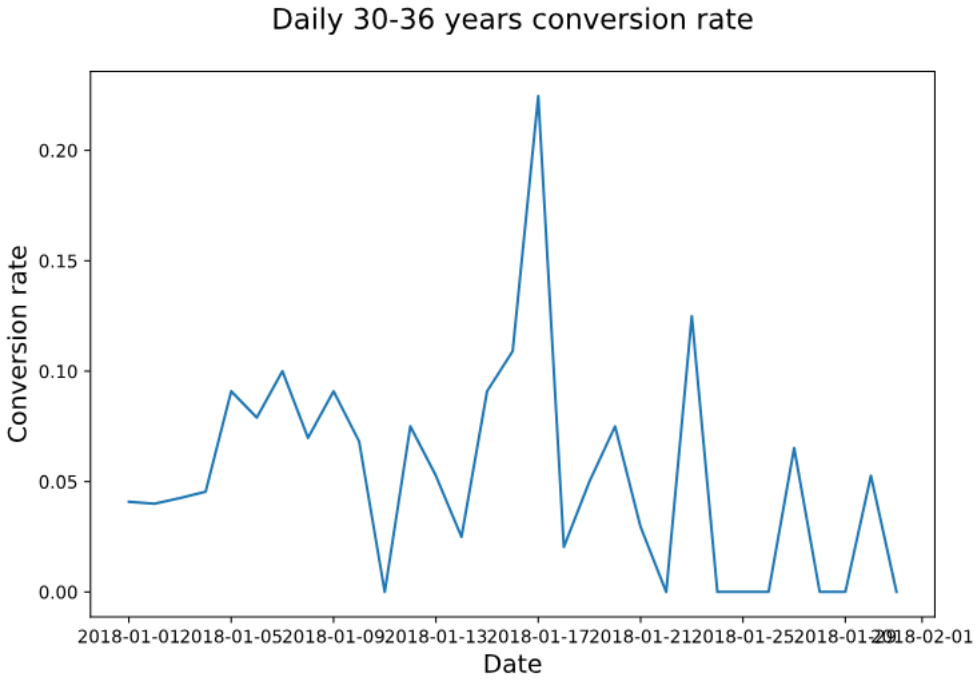

```
#Created function to analyse the marketing channels individually

def plotting_conv(dataframe):
    for column in dataframe:
        # Plot column by dataframe's index
        plt.plot(dataframe.index, dataframe[column])
        plt.title('Daily ' + str(column) + ' conversion rate\n',
                  size = 16)
        plt.ylabel('Conversion rate', size = 14)
        plt.xlabel('Date', size = 14)
        # Show plot
        plt.show()
        plt.clf()

#Testing both our functions:
# Calculate conversion rate by date served and age group
age_group_conv = conversion_rate(marketing, ['date_served','age_group'])

# Unstack age_group_conv and create a DataFrame
age_group_df = pd.DataFrame(age_group_conv.unstack(level=1))

# Plot the results
plotting_conv(age_group_df)
```

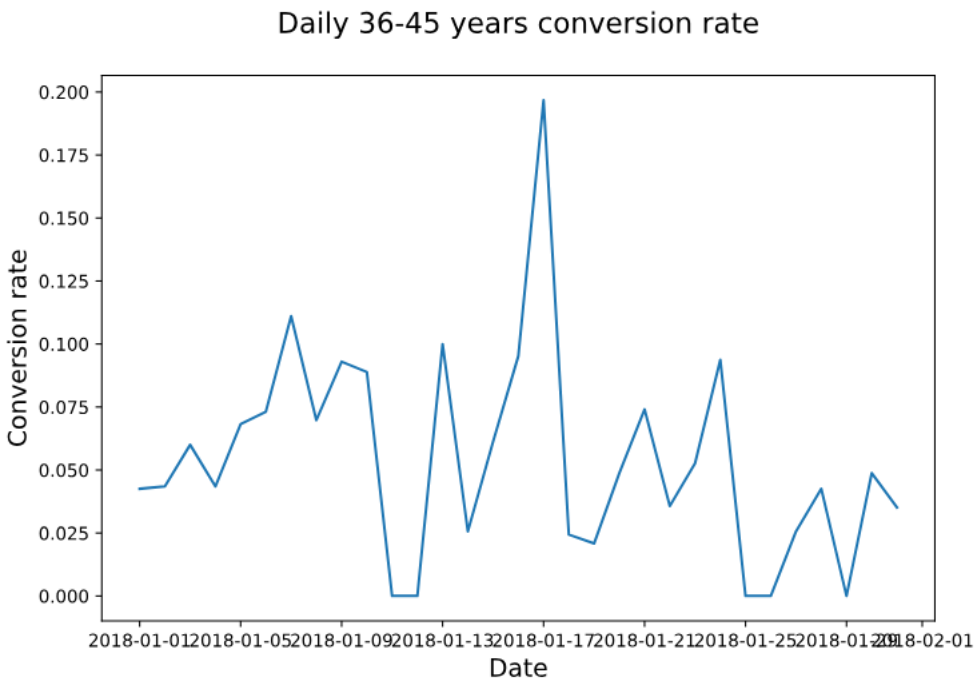
Solution: gives out graphs for all the age groups individually based on conversion rates on each day.

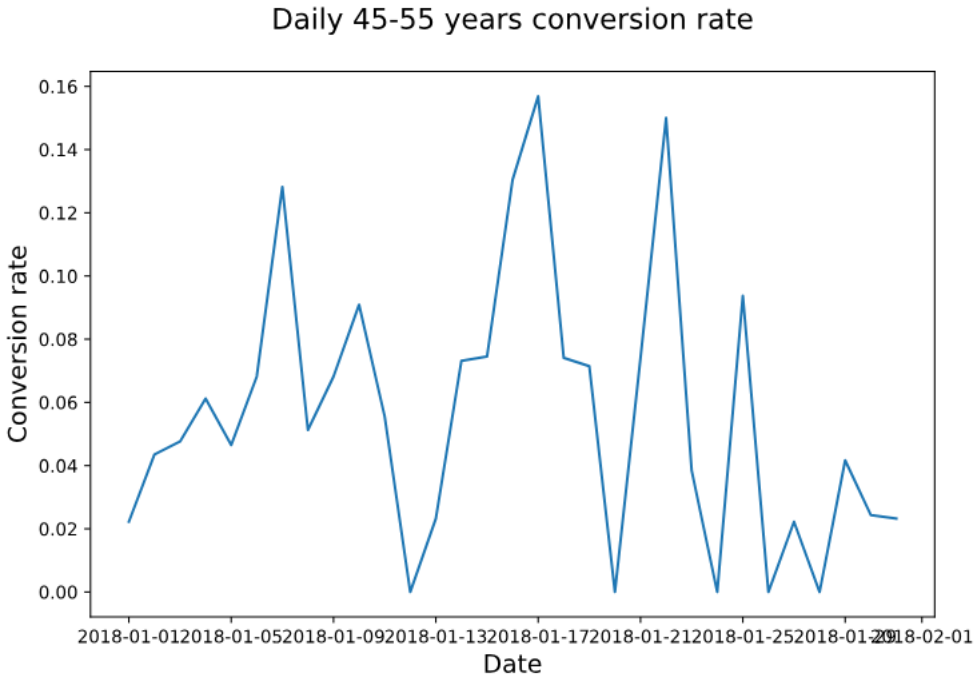

Daily 0-18 years conversion rate
Conversion rate
0.35
0.30
0.25
0.20
0.15
0.10
0.05
2018-01-01
2018-01-05
2018-01-09
2018-01-13
2018-01-17
2018-01-21
2018-01-25
2018-01-29
2018-02-01
Date

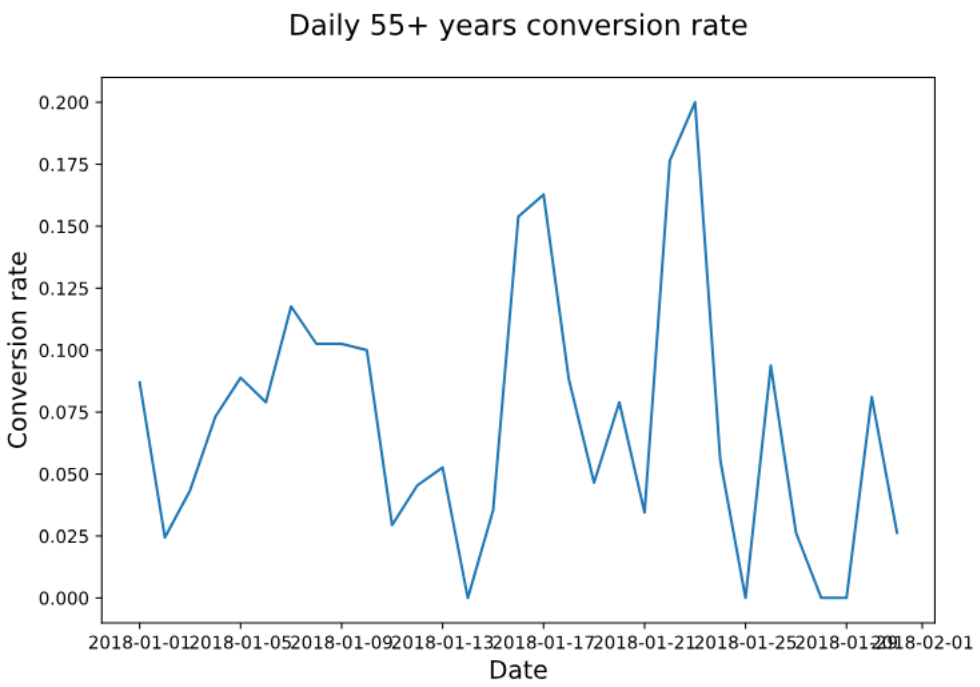

Daily 19-24 years conversion rate
Conversion rate
0.35
0.30
0.25
0.20
0.15
0.10
0.05
2018-01-01
2018-01-05
2018-01-09
2018-01-13
2018-01-17
2018-01-21
2018-01-25
2018-01-29
2018-02-01
Date


Daily 24-30 years conversion rate
Conversion rate
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
2018-01-01
2018-01-05
2018-01-09
2018-01-13
2018-01-17
2018-01-21
2018-01-25
2018-01-29
2018-02-01
Date


Daily 30-36 years conversion rate
Conversion rate
0.20
0.15
0.10
0.05
0.00
2018-01-01
2018-01-05
2018-01-09
2018-01-13
2018-01-17
2018-01-21
2018-01-25
2018-01-29
2018-02-01
Date

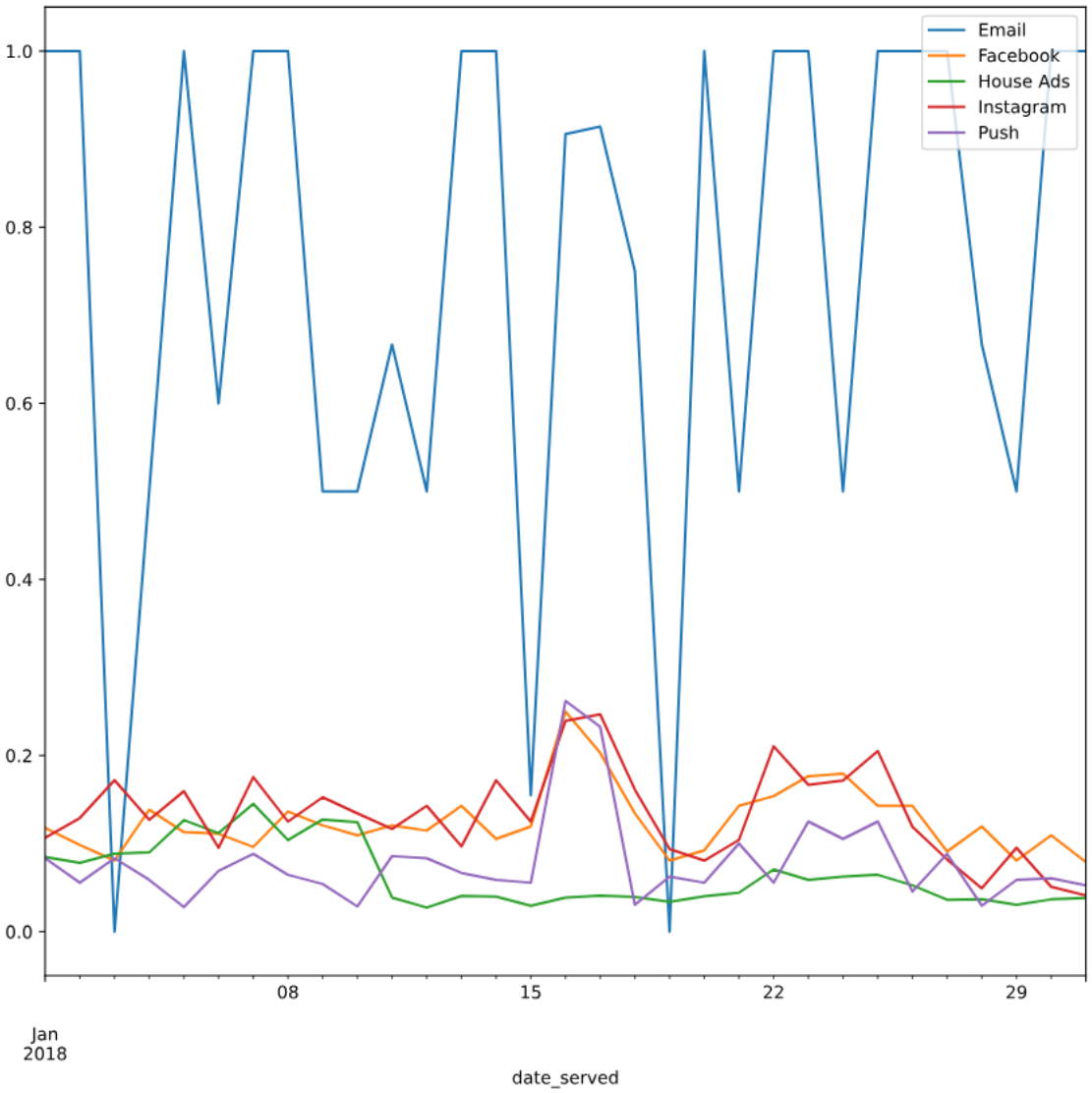

Daily 36-45 years conversion rate
Conversion rate
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
2018-01-01
2018-01-05
2018-01-09
2018-01-13
2018-01-17
2018-01-21
2018-01-25
2018-01-29
2018-02-01
Date


Daily 45-55 years conversion rate
Conversion rate
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
2018-01-01
2018-01-05
2018-01-09
2018-01-13
2018-01-17
2018-01-21
2018-01-25
2018-01-29
2018-02-01
Date



Use your `conversion_rate()` function on `marketing` to determine conversion rate by `'date_served'` and `'marketing_channel'`.

```
# Calculate conversion rate by date served and channel
daily_conv_channel = conversion_rate(marketing, ['date_served','marketing_channel'])

print(daily_conv_channel.head())

date_served  marketing_channel
2018-01-01   Email                1.000
             Facebook             0.118
             House Ads            0.085
             Instagram            0.107
             Push                 0.083

# Unstack daily_conv_channel with level equal to one and convert the result into a DataFrame

daily_conv_channel = pd.DataFrame(daily_conv_channel.unstack(level = 1))

# Plot results of daily_conv_channel
daily_conv_channel.plot()
plt.legend(loc='upper right', labels= daily_conv_channel.columns.values)
plt.show()
```



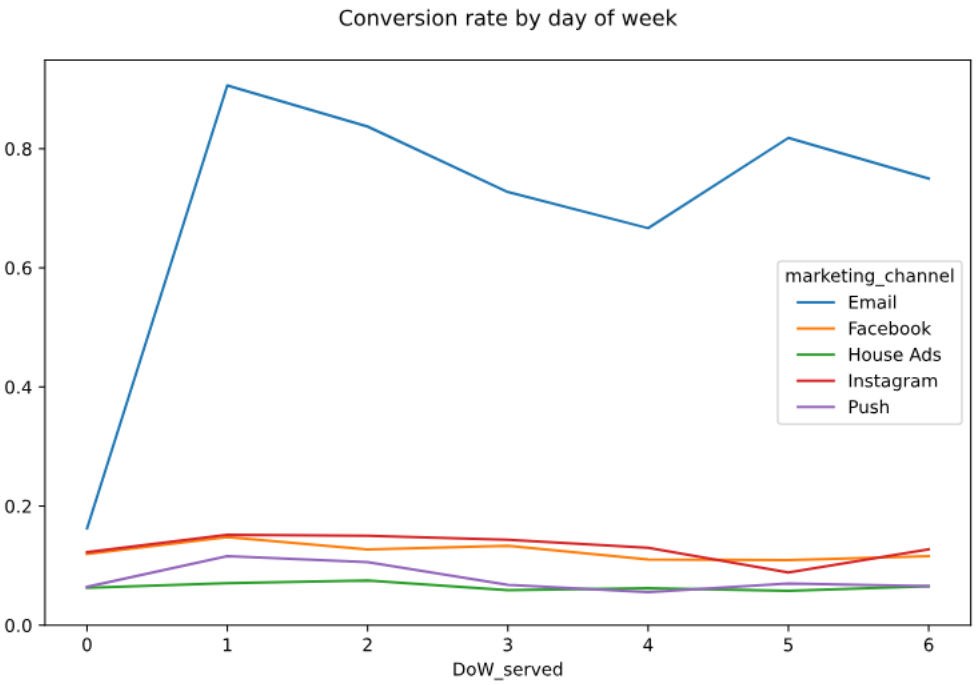
Result: A sudden decrease in conversion rate via house ads on Jan 11.

Now that you have confirmed that house ads conversion has been down since January 11, you will try to identify potential causes for the decrease.

As a data scientist supporting a marketing team, you will run into fluctuating metrics all the time. It's vital to identify if the fluctuations are due to expected shifts in user behavior (i.e., differences across the day of the week) versus a larger problem in technical implementation or marketing strategy.

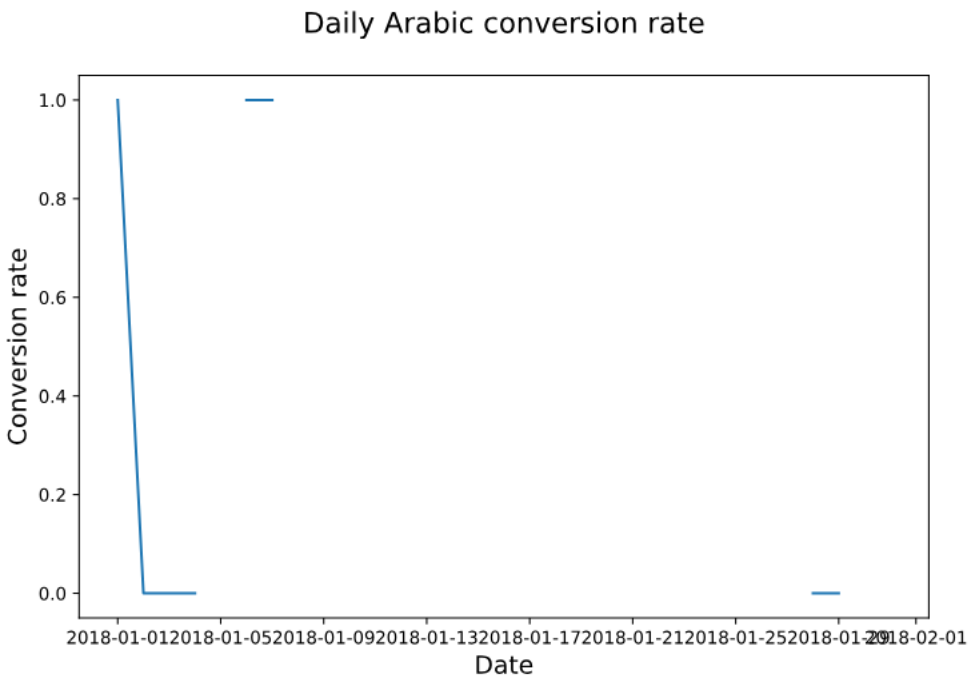
```
# Add day of week column to marketing
marketing['DoW_served'] = marketing['date_served'].dt.dayofweek

# Calculate conversion rate by day of week
DoW_conversion = conversion_rate(marketing, ['DoW_served', 'marketing_channel'])


# Unstack channels
DoW_df = pd.DataFrame(DoW_conversion.unstack(level=1))

# Plot conversion rate by day of week
DoW_df.plot()
plt.title('Conversion rate by day of week\n')
```

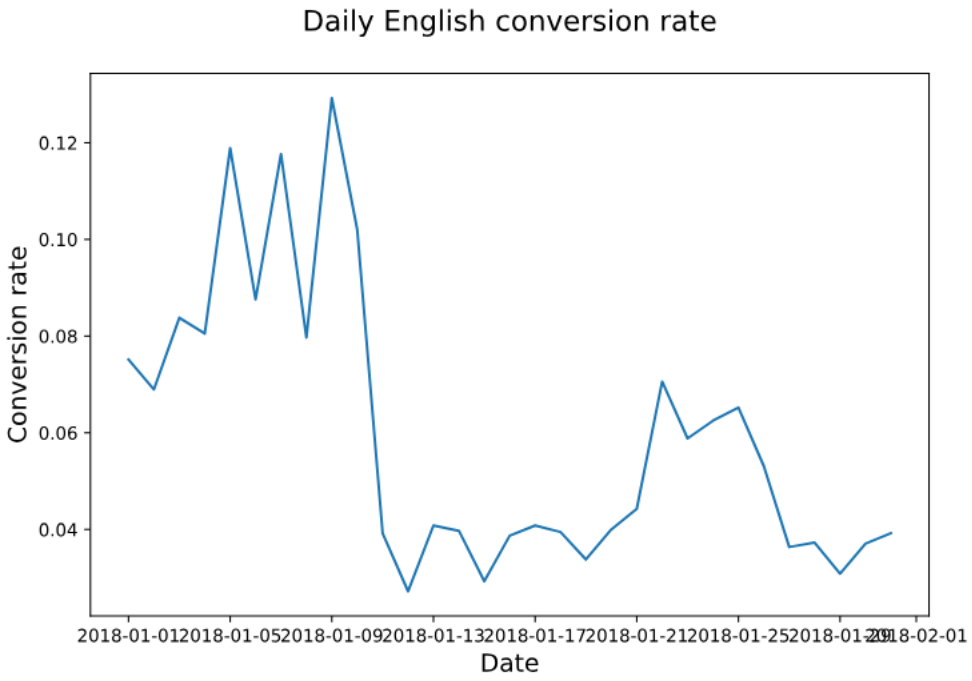
```
plt.ylim(0)
plt.show()
```

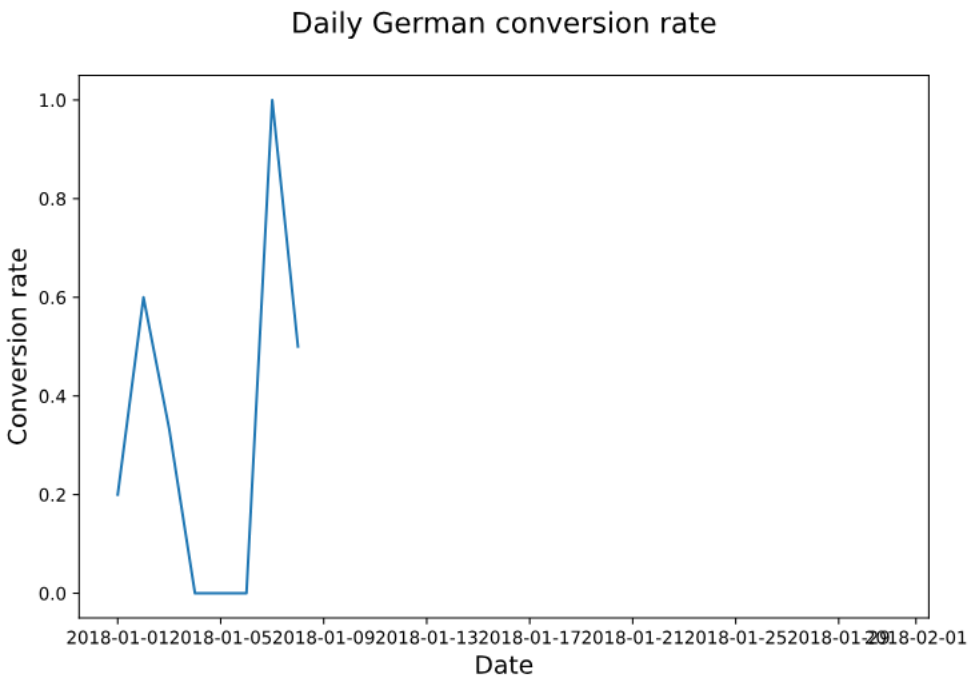


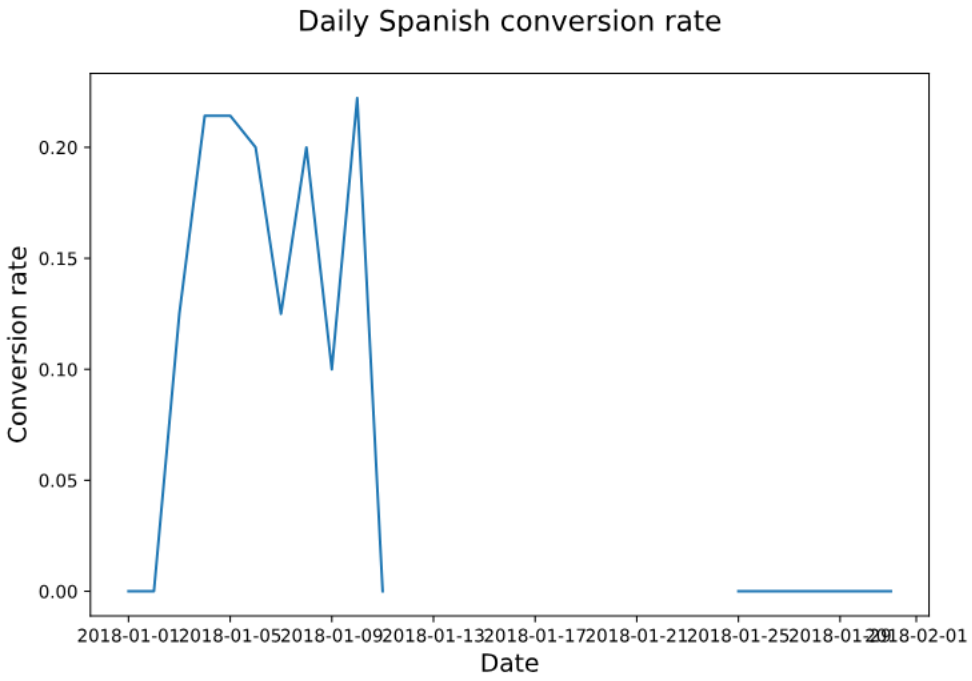
Now that you've ruled out natural fluctuations across the day of the week a user saw our marketing assets as they cause for decreasing house ads conversion, you will take a look at conversion by language over time. Perhaps the new marketing campaign does not apply broadly across different cultures.

```
# Isolate the rows where marketing channel is House Ads
house_ads = marketing[marketing['marketing_channel']=='House Ads']

# Calculate conversion by date served, and language displayed
conv_lang_channel = conversion_rate(house_ads, ['date_served', 'language_displayed'])

# Unstack conv_lang_channel
conv_lang_df = pd.DataFrame(conv_lang_channel.unstack(level=1))

# Use your plotting function to display results
plotting_conv(conv_lang_df)
```


Daily Arabic conversion rate
Conversion rate
1.0
0.8
0.6
0.4
0.2
0.0
2018-01-01
2018-01-05
2018-01-09
2018-01-13
2018-01-17
2018-01-21
2018-01-25
2018-01-29
2018-02-01
Date


Daily English conversion rate
Conversion rate
0.12
0.10
0.08
0.06
0.04
2018-01-01
2018-01-05
2018-01-09
2018-01-13
2018-01-17
2018-01-21
2018-01-25
2018-01-29
2018-02-01
Date





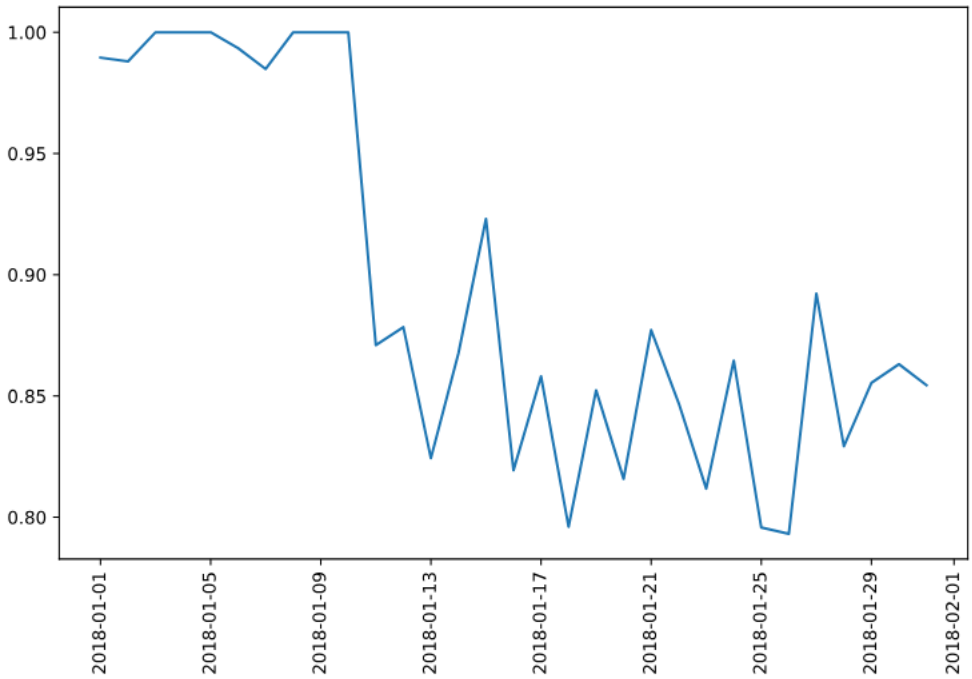
It is vital that you not only say "looks like there's a language problem," but instead identify what the problem is specifically so that the team doesn't repeat their mistake.

```
# Add the new column is_correct_lang
house_ads['is_correct_lang'] = np.where(
    house_ads['language_preferred'] == house_ads['language_displayed'],
    'Yes',
    'No')
```

```
# Groupby date_served and correct_language
language_check = house_ads.groupby(by=['date_served','is_correct_lang'])['user_id'].count()

# Unstack language_check and fill missing values with 0's
language_check_df = pd.DataFrame(language_check.unstack(level=1)).fillna(0)

# Print results
print(language_check_df)
```

```
is_correct_lang    No    Yes
date_served
2018-01-01        2.0  189.0
2018-01-02        3.0  247.0
2018-01-03        0.0  220.0
2018-01-04        0.0  168.0
2018-01-05        0.0  160.0
2018-01-06        1.0  151.0
2018-01-07        2.0  130.0
2018-01-08        0.0  154.0
2018-01-09        0.0  157.0
2018-01-10        0.0  170.0
2018-01-11       20.0  135.0
2018-01-12       18.0  130.0
2018-01-13       26.0  122.0
2018-01-14       20.0  131.0
2018-01-15       16.0  192.0
2018-01-16       28.0  127.0
2018-01-17       21.0  127.0
2018-01-18       31.0  121.0
2018-01-19       22.0  127.0
2018-01-20       28.0  124.0
2018-01-21       14.0  100.0
2018-01-22       13.0   72.0
2018-01-23       16.0   69.0
2018-01-24       13.0   83.0
2018-01-25       19.0   74.0
2018-01-26       24.0   92.0
2018-01-27       18.0  149.0
2018-01-28       28.0  136.0
2018-01-29       24.0  142.0
2018-01-30       23.0  145.0
2018-01-31       23.0  135.0
```

```
# Divide the count where language is correct by the row sum
language_check_df['pct'] = language_check_df['Yes']/language_check_df.sum(axis=1)

# Plot and show your results
plt.plot(language_check_df.index.values, language_check_df['pct'])
plt.show()
```



Now that you've determined that language is, in fact, the issue with House Ads conversion, stakeholders need to know how many subscribers they lost as a result of this bug.

you will index non-English language conversion rates against English conversion rates in the time period before the language bug arose.

```
# Calculate pre-error conversion rate
house_ads_bug = house_ads[house_ads['date_served'] < '2018-01-11']
lang_conv = conversion_rate(house_ads_bug,['language_displayed'])

# Index other language conversion rate against English
spanish_index = lang_conv['Spanish']/lang_conv['English']
arabic_index = lang_conv['Arabic']/lang_conv['English']
german_index = lang_conv['German']/lang_conv['English']

print("Spanish index:", spanish_index)
print("Arabic index:", arabic_index)
print("German index:", german_index)

Spanish index: 1.681924882629108
Arabic index: 5.045774647887324
German index: 4.485133020344287
```

To understand the true impact of the bug, it is crucial to determine how many subscribers we would have expected had there been no language error. This is crucial to understanding the scale of the problem and how important it is to prevent this kind of error in the future.

In this step, you will create a new DataFrame that you can perform calculations on to determine the expected number of subscribers. This DataFrame will include how many users prefer each language by day. Once you have the DataFrame, you can begin calculating how many subscribers you would have expected to have had the language bug not occurred.

```
# Group house_ads by date and language
converted = house_ads.groupby(['date_served','language_preferred']).agg({'user_id':'nunique', 'converted':'sum'})

# Unstack converted
converted_df = pd.DataFrame(converted.unstack(level=1))

#lets just see the df
```

```
print(converted_df.head())

user_id                        converted
language_preferred  Arabic English German Spanish    Arabic English German Spanish
date_served
2018-01-01             2.0   171.0    5.0    11.0       2.0    13.0    1.0     0.0
2018-01-02             3.0   200.0    5.0    10.0       0.0    14.0    3.0     0.0
2018-01-03             2.0   179.0    3.0     8.0       0.0    15.0    1.0     1.0
2018-01-04             2.0   149.0    2.0    14.0       0.0    12.0    0.0     3.0
2018-01-05             NaN   143.0    1.0    14.0       NaN    17.0    0.0     3.0
```

Now that you've created an index to compare English conversion rates against all other languages, you will build out a DataFrame that will estimate what daily conversion rates should have been if users were being served the correct language.

An expected conversion DataFrame named `converted` has been created for you grouping `house_ads` by date and preferred language. It contains a count of unique users as well as the number of conversions for each language, each day.

For example, you can access the number of Spanish-speaking users who received house ads using `converted[('user_id','Spanish')]`.

```
# Create English conversion rate column for affected period
converted['english_conv_rate'] = converted.loc['2018-01-11':'2018-01-31']['converted','English']

# Create expected conversion rates for each language
converted['expected_spanish_rate'] = converted['english_conv_rate']* spanish_index
converted['expected_arabic_rate'] = converted['english_conv_rate'] * arabic_index
converted['expected_german_rate'] = converted['english_conv_rate'] * german_index

# Multiply number of users by the expected conversion rate
converted['expected_spanish_conv'] = converted['expected_spanish_rate']* converted[('user_id', 'Spanish')]/100
converted['expected_arabic_conv'] = converted['expected_arabic_rate']* converted[('user_id', 'Arabic')]/100
converted['expected_german_conv'] = converted['expected_german_rate']*converted[('user_id', 'German')]/100
```

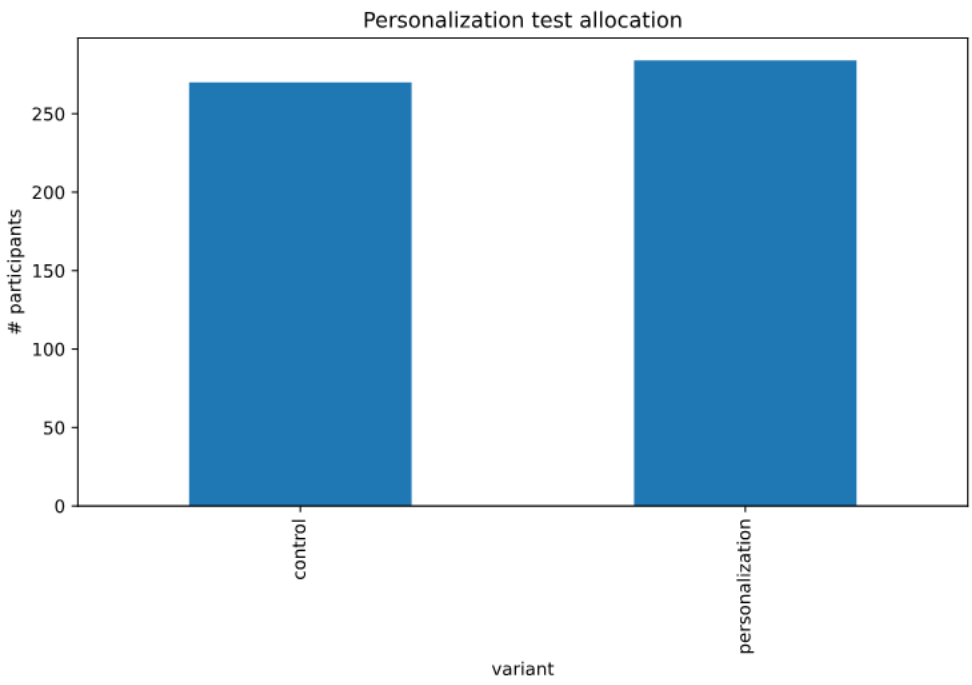
```
# Use .loc to slice only the relevant dates
converted = converted.loc['2018-01-11':'2018-01-31']

# Sum expected subscribers for each language
expected_subs = converted['expected_spanish_conv'].sum() + converted['expected_arabic_conv'].sum() + converted['expected_german_conv'].sum(

# Calculate how many subscribers we actually got
actual_subs = converted[('converted','Spanish')].sum() + converted[('converted','Arabic')].sum() + converted[('converted','German')].sum()

# Subtract how many subscribers we got despite the bug
lost_subs = expected_subs - actual_subs
print(lost_subs)

#answer: 32.14414319248826
```

A/B Testing:

The email portion of this campaign was actually run as an A/B test. Half the emails sent out were generic upsells to your product while the other half contained personalized messaging around the users' usage of the site.

Before you begin analyzing the results, you will check to ensure users were allocated equally to the test and control groups.

```
# Subset the DataFrame
email = marketing[marketing['marketing_channel']=='Email']

# Group the email DataFrame by variant
alloc = email.groupby(by=['variant'])['user_id'].nunique()

# Plot a bar chart of the test allocation
alloc.plot(kind='bar')
plt.title('Personalization test allocation')
plt.ylabel('# participants')
plt.show()
```



Now that we know allocation is relatively even let's look at the conversion rate for the control and personalization. Since we chose conversion rate as our key metrics for this test, it is highly important that we evaluate whether or not conversion was higher in the personalization treatment compared with the control.

```
# Group marketing by user_id and variant
subscribers = email.groupby(['user_id',
                             'variant'])['converted'].max()
subscribers_df = pd.DataFrame(subscribers.unstack(level=1))

# Drop missing values from the control column
control = subscribers_df['control'].dropna()

# Drop missing values from the personalization column
personalization = subscribers_df['personalization'].dropna()

print('Control conversion rate:', control.mean())
print('Personalization conversion rate:', personalization.mean())

<script.py> output:
    Control conversion rate: 0.2814814814814815
    Personalization conversion rate: 0.3908450704225352
```

Calculating Lift = (treatment conversion rate- control conversion rate) / control conversion rate

T-test to calculate significance:

If p vaue <0.05 then the test is running at more than 95% confidence level.

T-test in Python

```
from scipy.stats import ttest_ind

t = ttest_ind(control, personalized)

print(t)
```

```
Ttest_indResult(statistic=-2.7343299447505074,
                pvalue=0.006451487844694175)
```

```
def lift(a,b):
    # Calcuate the mean of a and b
    a_mean = np.mean(a)
    b_mean = np.mean(b)

    # Calculate the lift using a_mean and b_mean
    lift = a_mean- b_mean/ b_mean

    return str(round(lift*100, 2)) + '%'

# Print lift() with control and personalization as inputs
print(lift(control, personalization))

<script.py> output:
    -71.85%
```

Affects of A/B testing on customer segmentation:

We observed that your personalization experiment is highly statistically significant. However, when running experiments, it is important to check how new features are affecting specific demographics. Sometimes features that are highly appealing to one group are less appealing to others.

```
def ab_segmentation(segment):
  # Build a for loop for each subsegment in marketing
  for subsegment in np.unique(marketing[segment].values):
      print(subsegment)

      # Limit marketing to email and subsegment
      email = marketing[(marketing['marketing_channel'] == 'Email') & (marketing[segment] == subsegment)]

      subscribers = email.groupby(['user_id', 'variant'])['converted'].max()
      subscribers = pd.DataFrame(subscribers.unstack(level=1))
      control = subscribers['control'].dropna()
      personalization = subscribers['personalization'].dropna()

      print('lift:', lift(control, personalization))
      print('t-statistic:', stats.ttest_ind(control, personalization) , '\n\n')
```

This segment variable inputted can either be any column such as language, or age group.

```
# Use ab_segmentation on language displayed
ab_segmentation('language_displayed')
```

```
Arabic
lift: 50.0%
t-statistic: Ttest_indResult(statistic=-0.5773502691896255, pvalue=0.5795840000000001)


English
lift: 39.0%
t-statistic: Ttest_indResult(statistic=-2.2183598646203166, pvalue=0.026991701290720815)


German
lift: -1.62%
t-statistic: Ttest_indResult(statistic=0.1910083418078718, pvalue=0.8494394170062678)


Spanish
lift: 166.67%
t-statistic: Ttest_indResult(statistic=-2.3570226039551585, pvalue=0.040156718110477524)
```

```
# Use ab_segmentation on age group
ab_segmentation('age_group')

0-18 years
lift: 121.4%
t-statistic: Ttest_indResult(statistic=-2.966044912142211, pvalue=0.0038724494391297226)


19-24 years
lift: 106.24%
t-statistic: Ttest_indResult(statistic=-3.03179438478667, pvalue=0.0030623836114689134)


24-30 years
lift: 161.19%
t-statistic: Ttest_indResult(statistic=-3.861539544326876, pvalue=0.00018743381094867337)


30-36 years
lift: -100.0%
t-statistic: Ttest_indResult(statistic=3.1859064644147996, pvalue=0.0023238487431765137)


36-45 years
lift: -85.23%
t-statistic: Ttest_indResult(statistic=2.4317901279318503, pvalue=0.01797568600978829)


45-55 years
lift: -72.22%
t-statistic: Ttest_indResult(statistic=2.065499127317933, pvalue=0.043062339688201196)


55+ years
lift: -100.0%
t-statistic: Ttest_indResult(statistic=3.3265654564203397, pvalue=0.0016358623456360435)
```