ANALYSIS MARKETING CAMPAIGN WITH PANDAS [USING PYTHON]

CONTENTS

PREFACE

In this portfolio I am going to show my analysis which based on translating common business questions into measurable outcomes, including:

1. How did this marketing campaign perform ?
2. Which channel is referring the most subscribers ?
3. Why is a particular channel under performing ?

We are using data that I took from the DataCamp site which is a marketing dataset based on the data of an online subscription business.

The analysis processes will start from:

1. Importing and examining data
2. Pre-processing which includes: feature engineering and resolving errors in data
3. Creating Marketing Metrics with a formula:
   (*). Conversion Rate = number of people who convert / total number of people who we market to
   (*). Retention Rate = number of people who remain subscribed / total number of people who converted
4. Then we make The Customer Segmentation which includes:
   (*). Reaching age_group :
   marketing.groupby(['channel', 'age_group'])['user_id'].count()
5. Afterwards identify the problems and analyzing the impact (Dip in Conversion Rate)
   house_ads = marketing[marketing['channel'] = = 'House Ad']
   language = conversion_rate(house_ads, ['date_served', 'language_ displayed'])
6. Then at the end of the analysis we will use A/B test to understand the true impact of the change. On this section I am going to determine:
   (*). Lift
   (*). T-test

1. IMPORTING AND EXAMINING DATA

since the data is already on *'csv'* type it is really easy to download and save it to my 'visual studio code'. You could find my Python programming language at the last page. At that page you could see how I programme from getting the data on URL, saving dataframe locally to calling the data on my visual studio code. As a result the data looks like this :

```
     user_id date_served marketing_channel  ...  date_canceled subscribing_channel is_retained
0  a100000029      1/1/18         House Ads  ...            NaN           House Ads        True
1  a100000030      1/1/18         House Ads  ...            NaN           House Ads        True
2  a100000031      1/1/18         House Ads  ...            NaN           House Ads        True
3  a100000032      1/1/18         House Ads  ...            NaN           House Ads        True
4  a100000033      1/1/18         House Ads  ...            NaN           House Ads        True

[5 rows x 12 columns]
```

Also we could check in details what data has inside using: print(marketing.info()). Then it will show as follows :

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10037 entries, 0 to 10036
Data columns (total 12 columns):
user_id                10037 non-null object
date_served            10021 non-null object
marketing_channel      10022 non-null object
variant                10037 non-null object
converted              10022 non-null object
language_displayed     10037 non-null object
language_preferred     10037 non-null object
age_group              10037 non-null object
date_subscribed        1856 non-null object
date_canceled          577 non-null object
subscribing_channel    1856 non-null object
is_retained            1856 non-null object
dtypes: object(12)
memory usage: 470.5+ KB
```

We could literally see that DataFrame has 10,037 entries and 15 columns. Which makes the analysis gets easier.

2. PRE PROCESSING

(*). Feature Engineering → adding new, necessary columns

At this part I will add two new columns :

(*). day_of_weeks represent the day of the week as an integer.

(*). is_correct_lang expresses the campaign was shown to the users on their preferred language.

The new look of marketing data frame after adding the new columns :

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

      user_id date_served marketing_channel      ...      is_retained channel_code is_correct_lang
0  a100000029      1/1/18         House Ads      ...             True          1.0             Yes
1  a100000030      1/1/18         House Ads      ...             True          1.0             Yes
2  a100000031      1/1/18         House Ads      ...             True          1.0             Yes
3  a100000032      1/1/18         House Ads      ...             True          1.0             Yes
4  a100000033      1/1/18         House Ads      ...             True          1.0             Yes

[5 rows x 14 columns]
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 10037 entries, 0 to 10036
Data columns (total 14 columns):
user_id                10037 non-null object
date_served            10021 non-null object
marketing_channel      10022 non-null object
variant                10037 non-null object
converted              10022 non-null object
language_displayed     10037 non-null object
language_preferred     10037 non-null object
age_group              10037 non-null object
date_subscribed        1856 non-null object
date_canceled          577 non-null object
subscribing_channel    1856 non-null object
is_retained            1856 non-null object
channel_code           1856 non-null float64
is_correct_lang        10037 non-null object
dtypes: float64(1), object(13)
memory usage: 588.1+ KB
```

Currently, the date columns on 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.

After changing the data type of all date columns on *marketing* data, the data is updated like below :

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10037 entries, 0 to 10036
Data columns (total 15 columns):
user_id                10037 non-null object
date_served            10021 non-null datetime64[ns]
marketing_channel      10022 non-null object
variant                10037 non-null object
converted              10022 non-null object
language_displayed     10037 non-null object
language_preferred     10037 non-null object
age_group              10037 non-null object
date_subscribed        1856 non-null datetime64[ns]
date_canceled          577 non-null datetime64[ns]
subscribing_channel    1856 non-null object
is_retained            1856 non-null object
channel_code           1856 non-null float64
is_correct_lang        10037 non-null object
DoW                    1856 non-null float64
dtypes: datetime64[ns](3), float64(2), object(10)
memory usage: 784.2+ KB
```

3. MARKETING MATRICS

before the conversion_rate and retention rate will be calculated I need to determine how many users are seeing the marketing assets each day. This is crucial to understand how effective our marketing efforts have been over the past month.

I'll group the *marketing* DataFrame by *'date_served'* and 'user_id' then will count the number of unique user Ids :

```
daily_users = marketing.groupby(['date_served'])\
    ['user_id'].nunique()
```

Output :

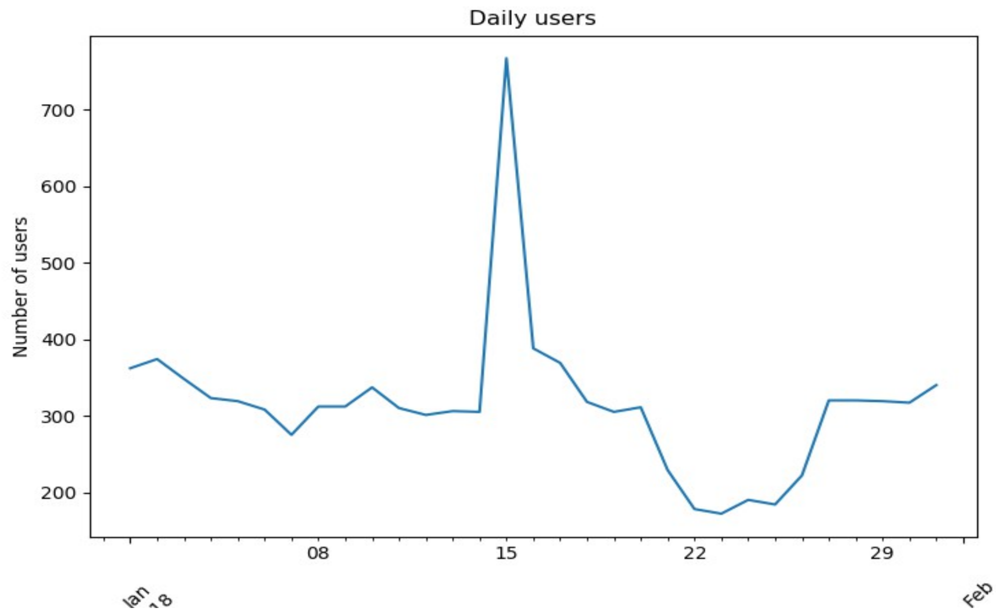
```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

date_served
2018-01-01    362
2018-01-02    374
2018-01-03    348
2018-01-04    323
2018-01-05    319
Name: user_id, dtype: int64
```

daily_user data groupby date_served and user_id

Getting the amount of daily_subscribers is a great first step. It is challenging to interpret daily trends by looking at a table. To make it easier to see the subscriber trends, I will visualize the results using a line plot.

Output:



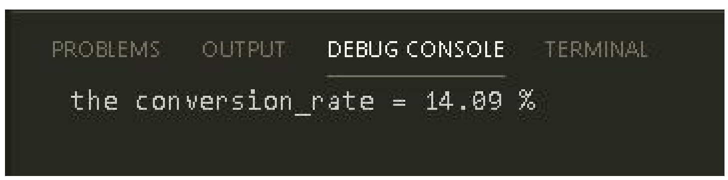
from the line plot it is clearly shown that on 14th January the campaign trend raised up rocketedly. Yet from 15th January to 23th of January it was dropped down significantly. Afterwards, it was begun to slightly move up again.

Now, it is time to determine 'conversion_rate' to evaluate how a marketing campaign performed, and one of the best ways to determine how effective a marketing team was at gaining new customers. (As a reminder, conversion rate is the percentage of the users who saw our marketing assets and subsequently became subscribers).

The formula for conversion rate is:

$$\frac{\text{Number of people who convert}}{\text{Total number of people who we market to}}$$

Output:



Question now: is it a good conversion_rate ?. This depends on every business. Instead, when working with a marketing team this number will help to look at historical data to determine if a conversion_rate is succeded. Furthermore, let's calculate 'retention_rate' that can give us a sense of whether the marketing campaign converted subscribers who were actually interested in the product.

The formula for retention rate is:

$$\frac{\text{Number of people who remain subscribed}}{\text{Total number of people who converted}}$$

Output :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

 retention_rate =  66.8 %
```

likewise conversion_rate, retention_rate helps us to look at historical data that we would expect.

4. **CREATING CUSTOMERS SEGMENTATION**

At this part I'll isolate the data to a specific area. I want to know how effective the marketing campaign was on converting English speakers. After grouping data to only English speaker I then calculate the conversion_rate. Thereafter, I can compare it to the overall conversion rate to gain a sense of how effective the marketing campaign was among this group compared to the overall population. The converting_rate formula is still the same from the previous one.

Output :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

 English speaker conversion rate: 13.13 %
```

Next step, we will compare English to the rest of the languages in the dataset. When we make this comparison across languages, we can get a sense of which languages convert well relative to others.

Output:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

language_displayed
Arabic      50.00
English     13.13
German      71.62
Spanish     20.00
Name: user_id, dtype: float64 %
```

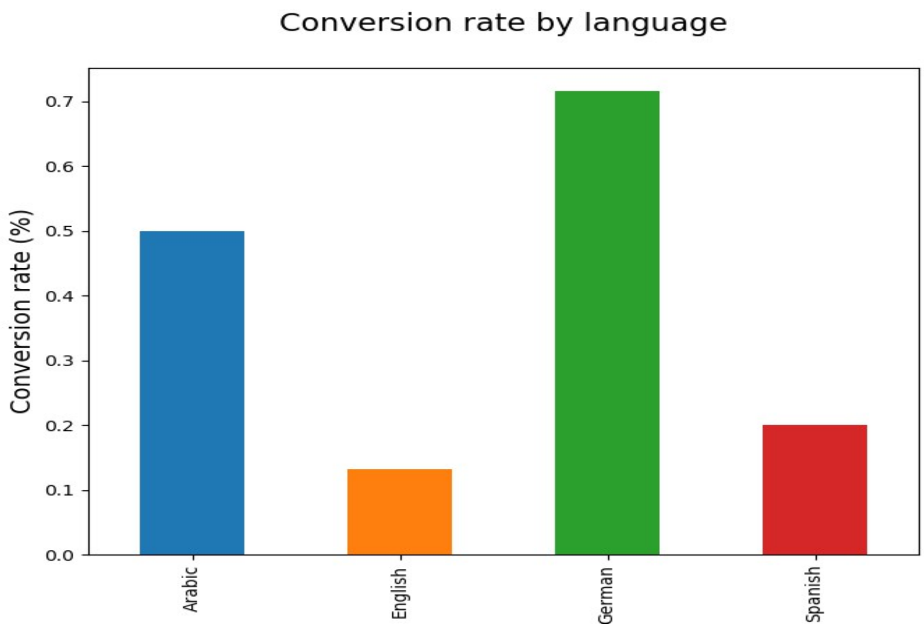
the output shows the conversion rate is much lower for English and Spanish. We'll conduct a deeper investigation into the differences between conversion_rate by language. Let us have a look whether there is any difference in the conversion rate based on when in the month, the users saw the campaign.

Output :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
date_served
2018-01-01    0.099448
2018-01-02    0.098930
2018-01-03    0.103448
2018-01-04    0.108359
2018-01-05    0.125392
2018-01-06    0.113636
2018-01-07    0.141818
2018-01-08    0.115385
2018-01-09    0.125000
2018-01-10    0.118694
2018-01-11    0.080645
2018-01-12    0.076412
2018-01-13    0.084967
2018-01-14    0.085246
2018-01-15    0.113429
2018-01-16    0.255155
2018-01-17    0.219512
2018-01-18    0.091195
2018-01-19    0.059016
2018-01-20    0.067524
2018-01-21    0.087336
2018-01-22    0.123596
2018-01-23    0.122093
2018-01-24    0.115789
2018-01-25    0.125000
2018-01-26    0.090090
2018-01-27    0.065625
2018-01-28    0.062500
2018-01-29    0.059561
2018-01-30    0.066246
2018-01-31    0.052941
Name: user_id, dtype: float64
```

Personally, it is hard to clearly see the essential information by only looking the data set even though it contains useful information. By plotting the data set it is much easier to compare relative conversion_ rates visually. Therefore, by plotting it using bar chart will make things a lot easier.

Output :



This plot shows that German and Arabic speakers have higher conversion rates than English and Spanish speakers.

After knowing how the marketing campaign performs among the languages and getting the significant result it is time for us <u>to understand trends over time</u> by creating a new DataFrame that includes the conversion rate each day. Following essentially the same steps as before when calculated the overall conversion rate, this time also grouping by the date a user subscribed.

<u>Looking at the daily conversion rate is crucial to contextualize whether the conversion rate on a particular day was good or bad</u>. Additionally, <u>looking at conversion rate over time can help to surface trends such as a conversion rate that appears to be going down over time</u>. These kinds of trends are crucial to identify for the marketing stakeholders as early as possible.

I'm going to group the marketing data with 'date_served', 'converted', and 'user_id'. Using the python logical programming as below :

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

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

the I'll calculate the daily_conversion_rate using formula :

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

Output :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

date_served
2018-01-01    0.099448
2018-01-02    0.098930
2018-01-03    0.103448
2018-01-04    0.108359
2018-01-05    0.125392
2018-01-06    0.113636
2018-01-07    0.141818
2018-01-08    0.115385
2018-01-09    0.125000
2018-01-10    0.118694
2018-01-11    0.080645
2018-01-12    0.076412
2018-01-13    0.084967
2018-01-14    0.085246
2018-01-15    0.113429
2018-01-16    0.255155
```

```
2018-01-17    0.219512
2018-01-18    0.091195
2018-01-19    0.059016
2018-01-20    0.067524
2018-01-21    0.087336
2018-01-22    0.123596
2018-01-23    0.122093
2018-01-24    0.115789
2018-01-25    0.125000
2018-01-26    0.090090
2018-01-27    0.065625
2018-01-28    0.062500
2018-01-29    0.059561
2018-01-30    0.066246
2018-01-31    0.052941
Name: user_id, dtype: float64
```

We notice that it is difficult to identify trends using the DataFrame. I will then plot the results to make interpretation easier. It is essential to look at how key metrics changed throughout the campaign. The key metrics can help us catch problems that may have happened during the campaign, such as a bug in the checkout system that led to a dip in conversion toward the end of the campaign. Metrics over time can also surface trends like gaining more subscribers over the weekends or on specific holidays. We will then build upon the daily conversion rate DataFrame 'daily_conversion_rate' that we built previously. Before we can begin visualizing, we need to transform our data into an easier format using pandas DataFrame (pd.DataFrame()) and then we are able to plot with matplotlib.

```
# Reset index to turn the results into a DataFrame
daily_conversion_rate = pd.DataFrame(daily_conversion_rate.reset_index(0))

# Rename columns
daily_conversion_rate.columns = ['date_served', 'conversion_rate']
```

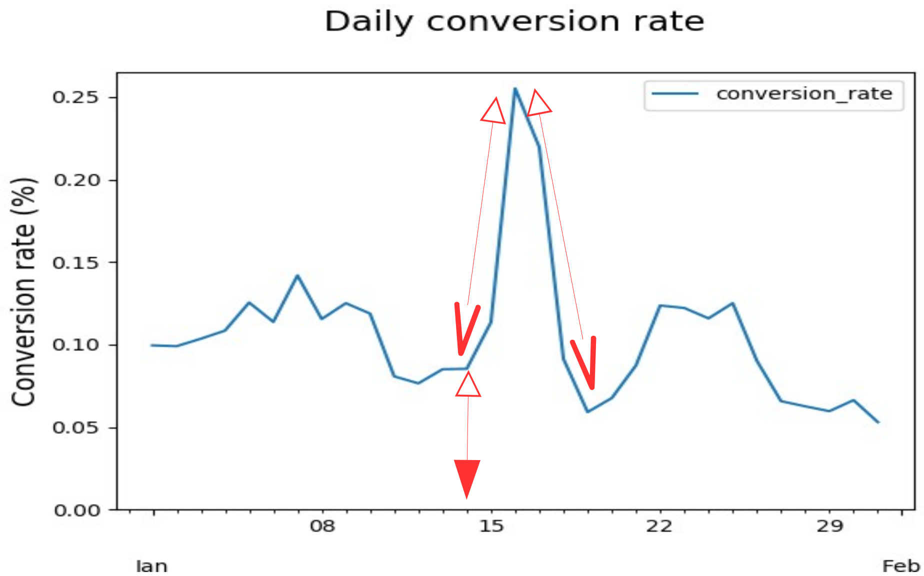
Output :
(1). New DataFrame of 'daily_conversion_rate

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

	date_served	conversion_rate
0	2018-01-01	0.099448
1	2018-01-02	0.098930
2	2018-01-03	0.103448
3	2018-01-04	0.108359
4	2018-01-05	0.125392
5	2018-01-06	0.113636
6	2018-01-07	0.141818
7	2018-01-08	0.115385
8	2018-01-09	0.125000
9	2018-01-10	0.118694
10	2018-01-11	0.080645
11	2018-01-12	0.076412
12	2018-01-13	0.084967
13	2018-01-14	0.085246
14	2018-01-15	0.113429
15	2018-01-16	0.255155
16	2018-01-17	0.219512

16	2018-01-17	0.219512
17	2018-01-18	0.091195
18	2018-01-19	0.059016
19	2018-01-20	0.067524
20	2018-01-21	0.087336
21	2018-01-22	0.123596
22	2018-01-23	0.122093
23	2018-01-24	0.115789
24	2018-01-25	0.125000
25	2018-01-26	0.090090
26	2018-01-27	0.065625
27	2018-01-28	0.062500
28	2018-01-29	0.059561
29	2018-01-30	0.066246
30	2018-01-31	0.052941

(2). Visualization of daily_conversion_rate



As you can see, the conversion rate is relatively steady except for one day in January.

Marketing channels across age groups

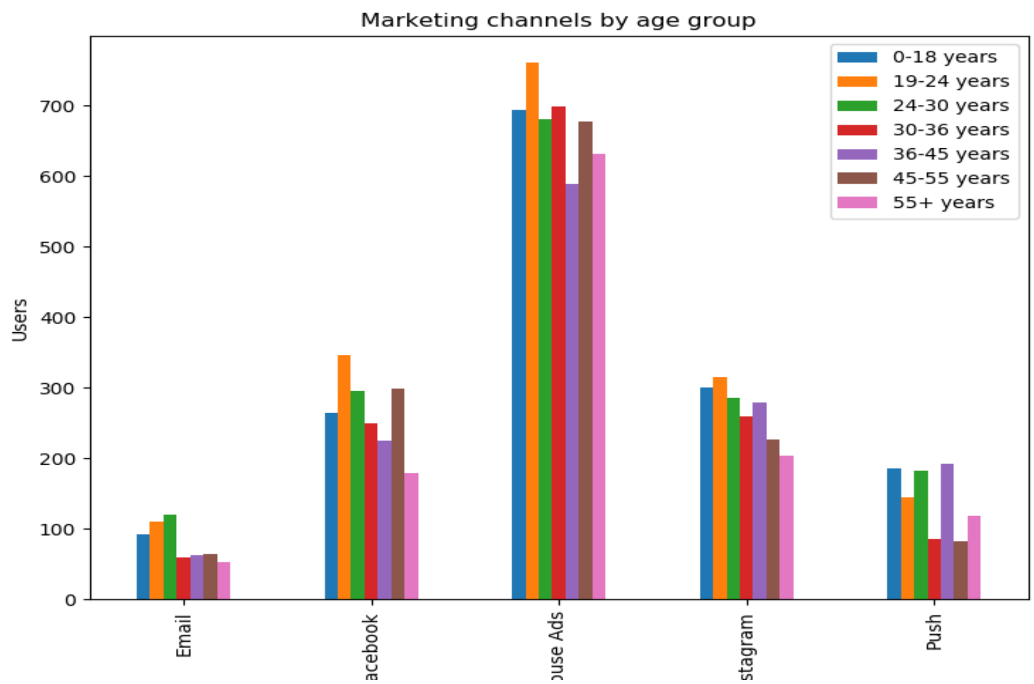
Some marketing stakeholders want to know if their marketing channels are reaching all users equally or if some marketing channels(at this data set: Email, Facebook, House Ads, Instagram, and Push) are serving specific age demographics.

It's common to get requests that require quick analysis and visualization. The better we visualize the results, the more likely we will effectively communicate the findings. I will then create a grouped bar chart showing how many people each marketing channel reached by age group. Using "groupby" and "unstack" pandas in python into marketing data as below :

```
# Grouping the marketing data by 'marketing_channel' and '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))
```

Output :



As you can see, email is not reaching older age groups, and Facebook is not reaching many people under age of 18 years old.

What about the retention rate for each marketing channel daily?. How does it look like ?. We could see the trend from each marketing channel by plotting it with line graph every day. Using pandas groupby and unstack the dataframe as follows.

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

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

# Divide retained subscribers by total subscribers
retention_rate = retention_subs / retention_total
retention_rate_df = pd.DataFrame(retention_rate.unstack(level=1).\
                             reset_index(0))
retention_rate_df['date_subscribed'] = pd.to_datetime(\
                                         retention_rate_df['date_subscribed'])
```

Output :

Now we could see which marketing channel who has the longest retention rate. Those are Facebook and Instagram.

5. DIP IN CONVERSION RATES

What if we notice problem from the marketing campaign. The recent scenario is: the house ads team has become worried about some irregularities they have noticed in conversion rate. The first thing that I could do is determine whether these changes are natural fluctuations or if they require further investigation. To start my investigation I will start by grouping the dataset on 'date_served' and 'marketing channel', and then unstack the grouped data as follows :

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

# Unstack daily_conv_channel and convert it to a DataFrame
daily_conv_channel =  pd.DataFrame(daily_conv_channel.unstack(level = 1))
```

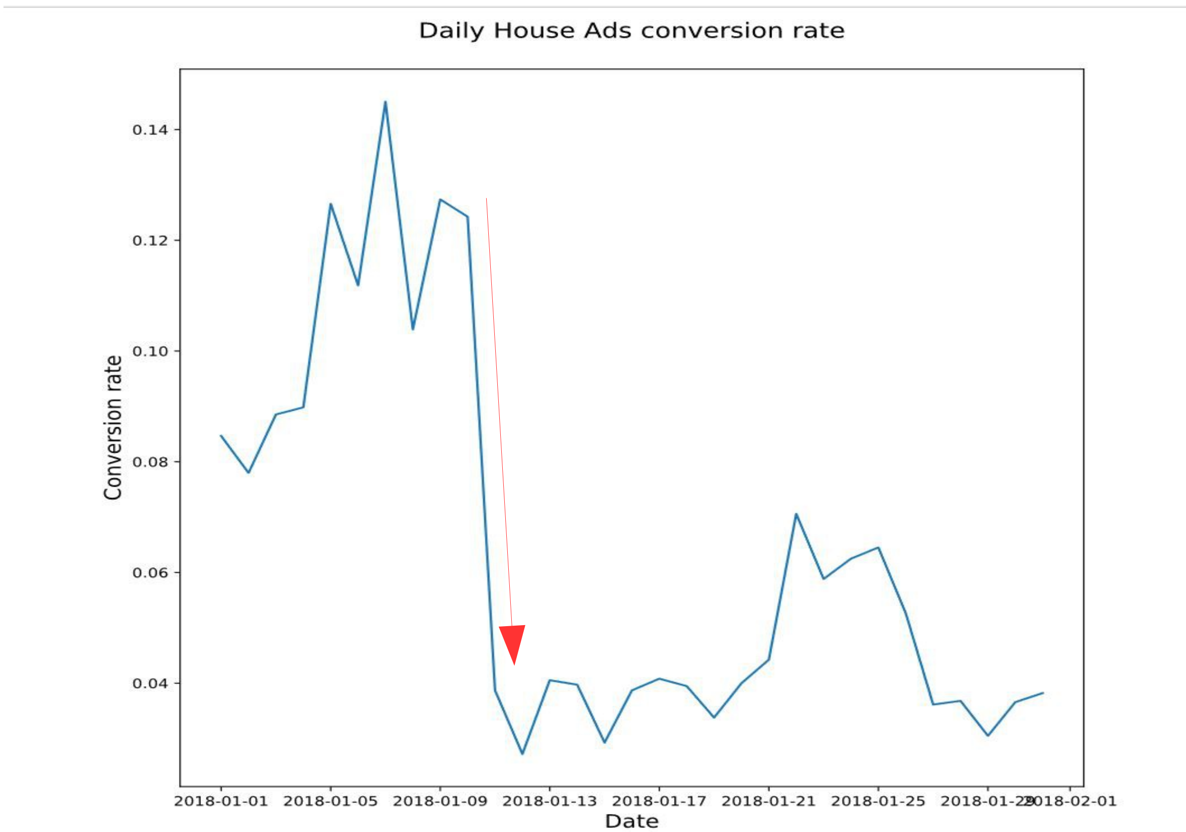
Output :

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

marketing_channel	Email	Facebook	House Ads	Instagram	Push
date_served					
2018-01-01	1.0	0.117647	0.084656	0.106667	0.083333
2018-01-02	1.0	0.098361	0.077982	0.129032	0.055556
2018-01-03	0.0	0.080645	0.088542	0.171875	0.083333
2018-01-04	0.5	0.138462	0.089820	0.126984	0.058824
2018-01-05	1.0	0.112903	0.126582	0.159420	0.027778

It is hard to see the trend that is happening in the House Add by seeing only the data set. Therefore I'll plot the data in order to get a visual result, specifically only take the plot for House Ads.

Ouput :



As on the graph we could see that there is a sudden decrease in conversion rate on January 11.

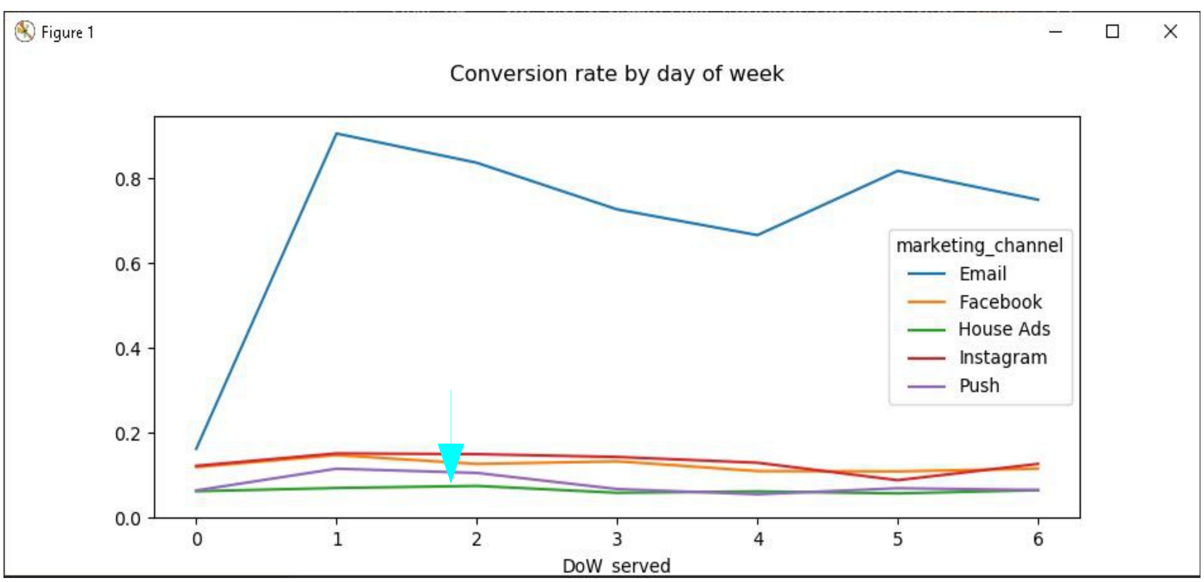
Now, let's try to figure out what might be going on with House Ads' conversion rate. Now that we have confirmed that house ads conversion has been down since January 11, I will try to identify potential causes for the decrease. It's vital to identify if the fluctuations are due to expected shifts in user behavior (i.e. differences across the day of the week) versus a larger problem in technical implementation or marketing strategy. I will begin with checking whether users are more likely to convert on weekends compared with weekdays and determine if that could be the cause for the changing house ads conversion rate. To start my analysis I'll do: grouping by 'date_served', unstacking the data that has been grouped, and plotting it. Using the python language as below:

```
# 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))
```

Output :



As you can see, email is particularly high and may be reflective of a tracking error, but house ads appear stable across the week with a slight

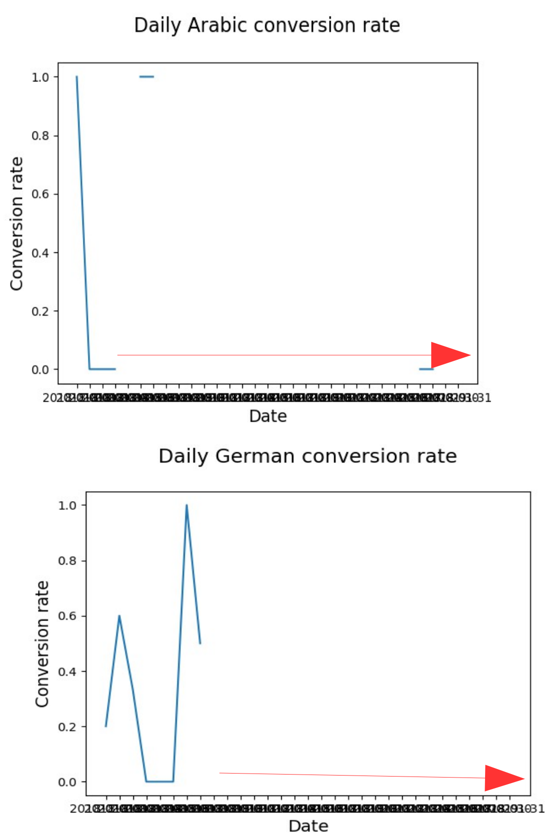
peak on Tuesday of 2nd January 2018. We will investigate further.
Now that I've ruled out natural fluctuations across the day of the week a user saw our marketing assets as they cause for decreasing house ads conversion, I will take a look at conversion by language over time. Probably the new marketing campaign does not apply broadly across different cultures. Ideally, the marketing team will consider cultural differences prior to launching a campaign, but sometimes mistakes are made, and we need to identify the cause. I'll start my analysis with grouping the data set only for House Ads then plotting it to make it easier to see the comparison among laguages. The python language program :

```
# 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))
```

Output :



As we can see, the English conversion rate drops around the 11th January (see the red arrow in Daily English Conversion Rate line graph), and there does not appear to be ads served in other languages (it was in German,and Arabic) for a two week period. We will investigate further.

The house ads team is concerned because they've seen their conversion rate drop suddenly in the past few weeks. In the previous analysis, it confirmed that conversion is down because we noticed a pattern around language preferences. It is vital that we do not only say "looks like there is a language problem" but instead identify what the problem is specifically so that the team does not repeat their mistake.
To begin my analysis i'll group add a new column called “house_ads['is_correct_lang']” with numpy array where the language_displayed equal language_displayed and written as 'Yes' and 'No. Then I'll group the data set, named it 'language_check', by 'date_served', 'is_correct_lang' and 'user_id' and address it to count. Then unstack the language_check dataframe to get a new dataframe so then easily be plotting. Last step is to visualize the findings.

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

# Groupby date_served and correct_language
language_check = house_ads.groupby(['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)
```

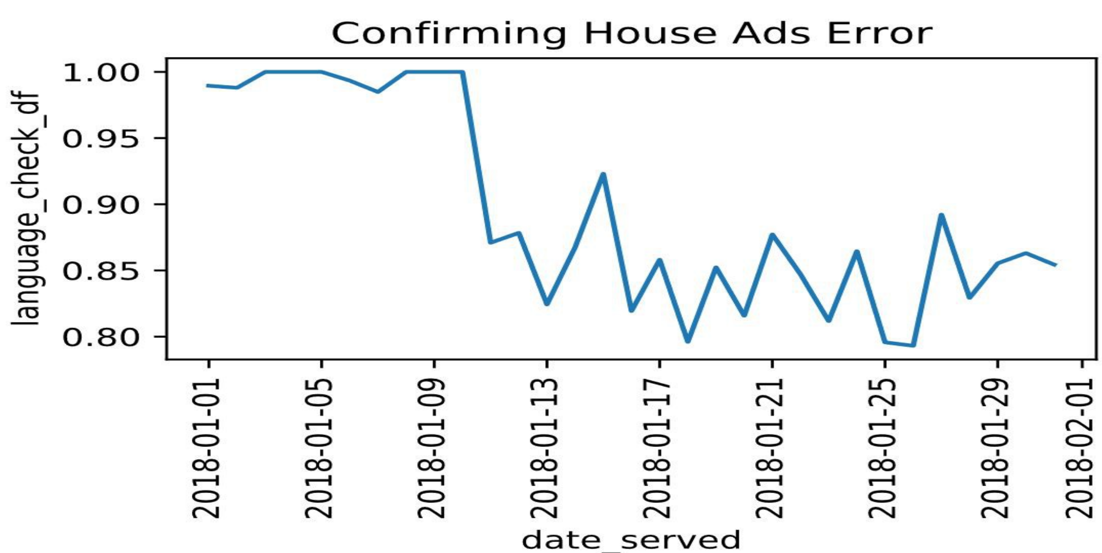
Output:

is_correct_lang date_served	No	Yes
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

Now that I have created a DataFrame that checks whether users see ads in the correct language. Now, let us calculate what percentage of users were not being served ads in the right language and plot the result.

```
# 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)
```

Output :



The line plot shows that house ads have been under performing due to serving all ads in English rather than each user's preferred language.

Now that we have determined that language is, in fact, the issue with House Ads conversion, we need to know how many subscribers we lost as a result of this bug. First step to determine the lost subscribers is I 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)
```

Output :

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

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

Now that the indexes have created for each language compared with English, we can now assess what conversion rate should have been during the rest of the month.

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, a new DataFrame will be created which I can perform calculations on to determine the expected number of subscribers. This

DataFrame will include how many users prefer each language by day. Once we have the DataFrame, we 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))
```

Output :

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

	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
2018-01-06	3.0	136.0	2.0	11.0	3.0	12.0	0.0	2.0
2018-01-07	2.0	117.0	2.0	10.0	2.0	14.0	2.0	1.0
2018-01-08	NaN	138.0	6.0	10.0	NaN	11.0	3.0	2.0
2018-01-09	NaN	147.0	NaN	10.0	NaN	19.0	NaN	1.0
2018-01-10	NaN	147.0	4.0	18.0	NaN	15.0	2.0	4.0
2018-01-11	7.0	133.0	2.0	13.0	0.0	6.0	0.0	0.0
2018-01-12	3.0	129.0	4.0	11.0	0.0	3.0	0.0	1.0
2018-01-13	6.0	121.0	5.0	16.0	0.0	5.0	1.0	0.0
2018-01-14	5.0	131.0	3.0	12.0	0.0	6.0	0.0	0.0
2018-01-15	2.0	189.0	4.0	10.0	0.0	6.0	0.0	0.0
2018-01-16	7.0	127.0	4.0	17.0	0.0	6.0	0.0	0.0
2018-01-17	2.0	126.0	3.0	16.0	0.0	2.0	0.0	4.0
2018-01-18	7.0	121.0	6.0	18.0	0.0	5.0	1.0	0.0
2018-01-19	5.0	126.0	5.0	12.0	1.0	4.0	0.0	0.0
2018-01-20	6.0	124.0	6.0	14.0	1.0	4.0	1.0	0.0
2018-01-21	1.0	99.0	4.0	9.0	0.0	5.0	0.0	0.0
2018-01-22	2.0	72.0	3.0	8.0	1.0	4.0	1.0	0.0
2018-01-23	3.0	69.0	4.0	9.0	0.0	5.0	0.0	0.0
2018-01-24	2.0	83.0	3.0	8.0	0.0	6.0	0.0	0.0
2018-01-25	3.0	75.0	4.0	11.0	0.0	4.0	2.0	0.0
2018-01-26	6.0	89.0	3.0	16.0	0.0	4.0	0.0	2.0
2018-01-27	3.0	148.0	3.0	12.0	1.0	4.0	0.0	1.0
2018-01-28	5.0	134.0	3.0	21.0	0.0	4.0	0.0	2.0
2018-01-29	7.0	138.0	4.0	15.0	2.0	3.0	0.0	0.0
2018-01-30	4.0	139.0	3.0	18.0	0.0	4.0	0.0	2.0
2018-01-31	7.0	130.0	4.0	16.0	1.0	4.0	0.0	1.0

Next, that I have created an index to compare English conversion rates against all other languages, I 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, 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.

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

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

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

It is time to calculate how many subscribers were lost due to mistakenly serving users in English rather than their preferred language. Once the team has an estimate of the impact of this error, they can determine whether it is worth putting additional checks in place to avoid this in the future, but of course, it is worth it to try to prevent errors! In a way, but every choice a company makes requires work and funding. The more information we have, the better we will be able to evaluate the trade-off.

```
# Use .loc to slice only the relevant dates
converted = converted_df.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
```

Output :

32 subscribers may not seem like many, but for a small company this can be vitally important, especially when expanding to new markets.

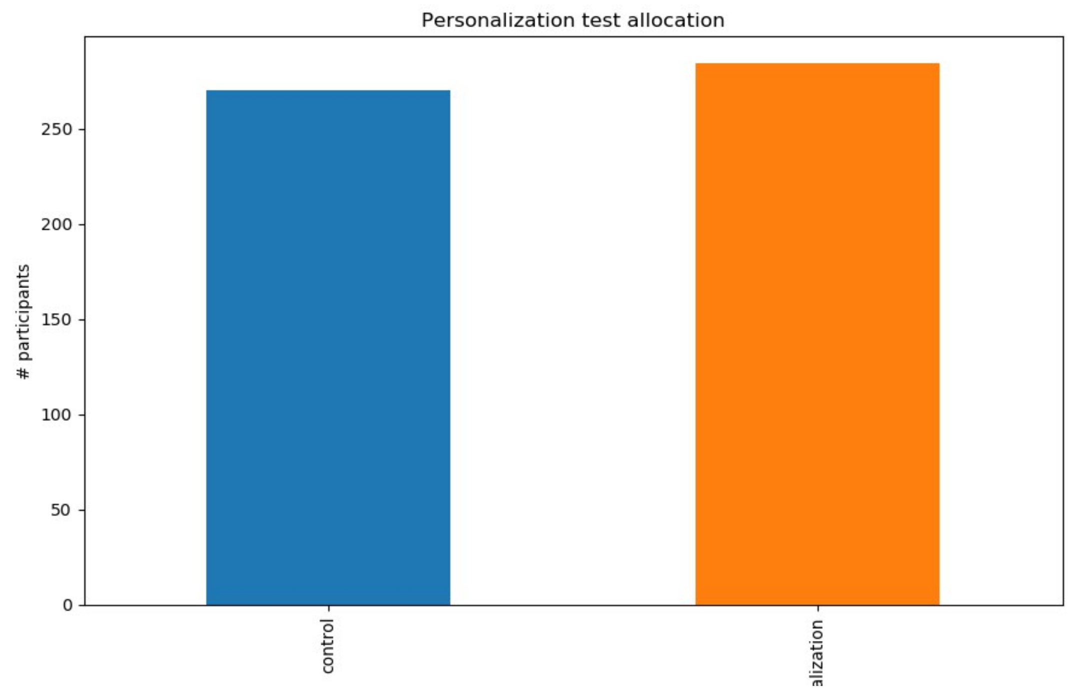
6. PERSONALIZATION A/B TEST

Before we finally determine the lift and A/B test the earliest step that I will take is: Test Allocation. At this step the email portion of this campaign was actually run as an A/B test. Half the emails sent out were generic up-sells to our product while the other half contained personalized messaging around the users' usage of the site.
Before we begin analyzing the results, I will check to ensure users were allocated equally to the test and control groups.
Using the python language as below :

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

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

Output :



There's a slight difference in allocation, but it is within the expected range thus we can continue with our analysis.

Now that we know allocation is relatively even let us 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. While we will dive-in deeper in subsequent exercises, measuring the difference between the key metric in the control and the treatment is the most important part of evaluating the success of an A/B test. Using python language programming as below :

```
# 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()
```

Output :

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL
Control conversion rate: 0.2814814814814815
Personalization conversion rate: 0.3908450704225352
```

We can see that personalization converted users at a higher rate than the control.

Next, I will build a lift function. Lift can be calculated by calculating the difference between the treatment effect (or the mean) of the treatment compared to the treatment effect of the control divided by the treatment effect of the control. The result is the percent difference between the control and treatment. The formula lift as below :

$$\frac{\text{Treatment conversion rate - Control conversion rate}}{\text{Control conversion rate}}$$

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

    # Calculate the lift using a_mean and b_mean
    lift = (b_mean - a_mean) / a_mean
    return str(round(lift*100, 2)) + '%'
```

Output :

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL
Lift =  38.85%
```

As you can see, there's a large lift, but are your results statistically significant?.

Now that we know the personalization variant outperformed the control, it is time for determining whether the result is statistically significant. Remember, statistical significance is vital to understanding whether a test showed a positive result by chance or if it is reflective of a true difference between the variants. This will enable the marketing team to make an informed choice about whether to roll out the feature or not. To prove if it is significant result I am using the python as below:

```
stats.ttest_ind(control, personalization)
```

Output :

```
In [1]: stats.ttest_ind(control, personalization)
Out[1]: Ttest_indResult(statistic=-2.7343299447505074,
    pvalue=0.006451487844694175)
```

from result above the question now is: Is the difference between the control and personalization statistically significant?.
The answer is : Yes, the result are statistically significant with p = 0.006

In the previous step we observed that 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.
Since we want to segment our data multiple times, we will build a function 'ab_segmentation() that analyzes the impact of A/B tests on segments of data that we can reuse each time we want to conduct this kind of analysis. Our function will take in a column name and run through each unique value in that column calculating lift and statistical significance. Using the python format as below :

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

Now that we have generated an ab_segmentation function, it's time to test it out. Often a treatment will not affect all people uniformly. Some people will love a particular marketing campaign while others hate it. We will run through two segments in our data that may be relevant to assessing the impact of our test. The two segments that I want to see the ab_segmentation value are: language_displayed and age_group

Output :

(*). Language Displayed

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
  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)
```

(*). Age Group

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
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=0.00018743381094867335)


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=0.017975686009788286)


45-55 years
lift: -72.22%
t-statistic: Ttest_indResult(statistic=2.065499127317933, pvalue=0.04306233968820196)


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

We found that personalization was extremely effective for younger groups but less effective than the control for older groups. One explanation might be that younger users understand how their data might be used online and think personalization is cool because it gives them more insight into themselves while older people might feel that this is a violation of their privacy.

CONCLUSION

Answering the questions from the beginning chapter:

1. How did this marketing campaign perform ?
   The marketing campaign was not performed as its best since we have found a bug in the marketing segmentation and in the language preference. However, it also performed well in some areas such as age_group in young age and marketing_channels.

2. Which channel is referring the most subscribers ?
   The personalization channel was extremely effective for younger groups. Also, House Ads, Facebook and Instagram are the good choice for broadly spreading the campaign instead of Email, and Push.

3. Why is a particular channel under performing ?
   There are a few factors for some channels did not perform in their best. As I have explained before such as reason of: language preference, culture differentiation, language error, bugs, and even the marketing campaign did not rightly broad.
   From the analysis that we have done we found that there is a bug in English speaker which we assume there is an error language influenced. Also, we figure out that there did not appear to be ads served in other languages (it is in Germany and Arabic language) for a two week period. This can be also happened because of bugs or errors. Or, the marketing team did not have their attention to it so then the campaign had stopped and caused loss from it. This has proven from the 32 subscribers we had lost due to lack of detailed analysis and strategies. Loosing 32 subscribers sounds small but for a small marketing company that could cost them a lot.

All in all, the findings from the analysis that we have done could be a good historical data for the marketing team to succeed the marketing campaign goals. From these analysis they will be more paying attention and will formulate their strategies thoroughly and effectively. Therefore, they could prevent the huge number of error and working productively and effectively.

PYTHON PROGRAMMING LANGUAGE

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

# Import numpy
  import numpy as np

# Import Matplotlib
  import matplotlib.pyplot as plt

# Import Stats
  from scipy import stats

# Import t-test
  from scipy.stats import ttest_ind
```

 # TASK 1 : IMPORTING AND EXAMINING DATA

```
# Import marketing.csv
 marketing = pd.read_csv('C:/Users/Hp ProBook 640/Documents/Analyzing Marketing\
                  Campaigns with Pandas/DATA/marketing1.csv',\
                  parse_dates =['date_served', 'date_subscribed', 'date_canceled'])

# Print the first five rows of the DataFrame
  print(marketing.head())

# Print the statistics of all columns
  print(marketing.describe())

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

 # TASK 2 : PRE PROCESSING

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

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


# 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)


# 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)

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

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

 # TASK 3 : MARKETING MATRICS

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

# Print head of daily_users
  print(daily_users.head())
  print(marketing.info())

# Plot daily_subscribers
  daily_users.plot()

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

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

# Display the plot
  plt.show()

# 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('the conversion_rate =',round(conversion_rate*100, 2), "%")


# 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('retention_rate = ', round(retention_rate*100, 2), "%")
```

TASK 4 : CREATING CUSTOMERS SEGMENTATION

```
# 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), '%')


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

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

# Calculate the conversion rate for all languages
  language_conversion_rate = subscribers/total
  print(round(language_conversion_rate*100, 2))

# Group by date_served and count unique users
```

```
  total = marketing.groupby(['date_served'])['user_id'].nunique()

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

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

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

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

# Reset index to turn the results into a DataFrame
  daily_conversion_rate = pd.DataFrame(daily_conversion_rate.reset_index(0))

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

# Print daily_conversion_rate
  print(daily_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()

# MARKETING CHANNEL ACROSS AGE GROUP
# Group channel age data
  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()

# 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())

# Count 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())

# Divide retained subscribers by total subscribers
  retention_rate = retention_subs / retention_total
  retention_rate_df = pd.DataFrame(retention_rate.unstack(level=1).reset_index(0))\
  retention_rate_df['date_subscribed'] = pd.to_datetime(retention_rate_df\
                                                  ['date_subscribed'])
```

```
# Set the style to 'ggplot'
 plt.style.use('ggplot')

# Create a figure with 2x2 subplot layout
 plt.subplot(3, 2, 1)

# Plot the retention rate for email
 plt.plot(retention_rate_df['Email'], color='blue')
 plt.title('Retention Rate for : Email')
 plt.xlabel('Date Subscribed')
 plt.ylabel('Retention Rate (%)')

# Plot the retention rate for facebook
 plt.subplot(3, 2, 2)
 plt.plot(retention_rate_df['Facebook'], color='red')
 plt.title('Retention Rate for : Facebook')
 plt.xlabel('Date Subscribed')
 plt.ylabel('Retention Rate (%)')

# Plot the enrollmment % of women in Health professions
 plt.subplot(3, 2, 3)
 plt.plot(retention_rate_df['House Ads'], color='green')
 plt.title('Retention Rate for : House Ads')
 plt.xlabel('Date Subscribed')
 plt.ylabel('Retention Rate (%)')

# Plot the enrollment % of women in Education
 plt.subplot(3, 2, 4)
 plt.plot(retention_rate_df['Instagram'], color='yellow')
 plt.title('Retention Rate for : Instagram')
 plt.xlabel('Date Subscribed')
 plt.ylabel('Retention Rate (%)')

# Plot the enrollment % of women in Education
 plt.subplot(3, 2, 5)
 plt.plot(retention_rate_df['Push'], color='purple')
 plt.title('Retention Rate for : Push')
 plt.xlabel('Date Subscribed')
 plt.ylabel('Retention Rate (%)')

# Improve spacing between subplots and display them
 plt.tight_layout()
 plt.show()
```

 # TASK 5 : DIP IN CONVERSION RATES

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

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

# Unstack daily_conv_channel and convert it to a DataFrame
  daily_conv_channel = pd.DataFrame(daily_conv_channel.unstack(level = 1))
# Plot results of daily_conv_channel (just taking the plot on House Ads)
  plotting_conv(daily_conv_channel)

# 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()

# 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)

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

# Add the new column is_correct_lang
  house_ads['is_correct_lang'] = np.where(house_ads['language_displayed'] = =\
```

```
                                house_ads['language_preferred'], 'Yes', 'No')

# Groupby date_served and correct_language
 language_check = house_ads.groupby(['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)

# 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.title('Confirming House Ads Error')
 plt.show()

# Making automated formula
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

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

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

# 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))

# Print the dataframe of converted_df
  print(converted_df)

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

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

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

# Use .loc to slice only the relevant dates
  converted = converted_df.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 (make it to the whole positive number without decimals)
  print('Lost Subscribers = ', round(lost_subs), 'people')
```

 # TASK 6 : PERSONALIZATION A/B TEST

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

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

# 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 the results
  print('Control conversion rate:', np.mean(control))
  print('Personalization conversion rate:', np.mean(personalization))

# Print automated calculation for lift
  def lift(a,b):
  # Calcuate the mean of a and b
    a_mean = np.mean(control)
    b_mean = np.mean(personalization)
```

```
  # Calculate the lift using a_mean and b_mean
    lift = (b_mean - a_mean) / a_mean
    return str(round(lift*100, 2)) + '%'

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

# Evaluating statistical significance
  print(stats.ttest_ind(control, personalization))

# ab_segmentation and lift formula for language_displayed
def ab_segmentation(segment):
  # Build a for loop for each subsegment in marketing
  for subsegment in np.unique( marketing['language_displayed']):
      def lift(a,b):
           # Calcuate the mean of a and b
              a_mean = np.mean(control)
              b_mean = np.mean(personalization)

           # Calculate the lift using a_mean and b_mean
              lift = (b_mean - a_mean) / a_mean
                return str(round(lift*100, 2)) + '%'
      # 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(subsegment)
      print('lift:', lift(control, personalization))
      print('t-statistic:', stats.ttest_ind(control, personalization), '\n\n')

ab_segmentation('language_displayed')

# ab_segmentation and lift formula for age_group
def ab_segmentation(segment):
  # Build a for loop for each subsegment in marketing
  for subsegment in np.unique( marketing['age_group']):
      def lift(a,b):
           # Calcuate the mean of a and b
              a_mean = np.mean(control)
```

```
        b_mean = np.mean(personalization)

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

        return str(round(lift*100, 2)) + '%'
    # 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(subsegment)
    print('lift:', lift(control, personalization))
    print('t-statistic:', stats.ttest_ind(control, personalization), '\n\n')

ab_segmentation('age_group')
```