Snapchat Political Ads

This project uses political ads data from Snapchat, a popular social media app. Interesting questions to consider include:

- What are the most prevalent organizations, advertisers, and ballot candidates in the data? Do you recognize any?
- What are the characteristics of ads with a large reach, i.e., many views? What may a campaign consider when maximizing an ad's reach?
- What are the characteristics of ads with a smaller reach, i.e., less views? Aside from funding constraints, why might a campaign want to produce an ad with a smaller but more targeted reach?
- What are the characteristics of the most expensive ads? If a campaign is limited on advertising funds, what type of ad may the campaign consider?
- What groups or regions are targeted frequently? (For example, for single-gender campaigns, are men or women targeted more frequently?) What groups or regions are targeted less frequently? Why? Does this depend on the type of campaign?
- Have the characteristics of ads changed over time (e.g. over the past year)?
- When is the most common local time of day for an ad's start date? What about the most common day of week? (Make sure to account for time zones for both questions.)

Getting the Data

The data and its corresponding data dictionary is downloadable here (https://www.snap.com/en-US/political-ads/). Download both the 2018 CSV and the 2019 CSV.

The CSVs have the same filename; rename the CSVs as needed.

Note that the CSVs have the exact same columns and the exact same data dictionaries (`readme.txt`).

Cleaning and EDA

- Concatenate the 2018 CSV and the 2019 CSV into one DataFrame so that we have data from both years.
- Clean the data.
  - Convert `StartDate` and `EndDate` into datetime. Make sure the datetimes are in the correct time zone.
- Understand the data in ways relevant to your question using univariate and bivariate analysis of the data as well as aggregations.

Hint 1: What is the "Z" at the end of each timestamp?

Hint 2: `pd.to_datetime` will be useful here. `Series.dt.tz_convert` will be useful if a change in time zone is needed.

Tip: To visualize geospatial data, consider Folium (https://python-visualization.github.io/folium/) or another geospatial plotting library.

Assessment of Missingness

Many columns which have `NaN` values may not actually have missing data. How come? In some cases, a null or empty value corresponds to an actual, meaningful value. For example, `readme.txt` states the following about `Gender` :

> Gender - Gender targeting criteria used in the Ad. If empty, then it is targeting all genders

In this scenario, an empty `Gender` value (which is read in as `NaN` in pandas) corresponds to "all genders".

- Refer to the data dictionary to determine which columns do **not** belong to the scenario above. Assess the missingness of one of these columns.

Hypothesis Test / Permutation Test

Find a hypothesis test or permutation test to perform. You can use the questions at the top of the notebook for inspiration.

Summary of Findings

Introduction

For this project we decided to choose the ads.ipynb notebook. We chose this one because Snapchat is an app we use and we were interested in where this project was going to take us with the dataset. Some questions that we were discussing while looking through the dataset were: More money spent on ads results in more impressions?, Ads during important political dates have more impressions?, Less money spent has less impressions? (and vice versa)?, What are the characteristics/gender/runtime/payed by/ of the most expensive ads?, EndDate dependent on Spend?, EndDate dependent on Impressions?, groupby country code and see: mean, median, mode, max, min etc?. We would say that the quality of this dataset is very good for the 2018 and somewhat good for the 2019. Only somewhat good because the data we are working with is relatively new but not real time, let's say we download the 2019 dataset and tomorrow they push an update to it then we would be using the old dataset. Using a current version of the dataset would help us draw more precise conclusions about the current data. The missing ness in this dataset takes investigation to look at since in the readme certain columns have meaningful NaN values while some do not. For example, in the 'EndDate' column we are left in the dark about why there are NaN values and we may need to run tests to determine what kind of missing ness it is. We came to the conclusion that the null hypothesis that we will be pursuing is: The money spend in 2018 and 2019 is the same on average? With a Alt Hypothesis of: Money spent in 2019 is higher because it is closer to election year. This dataset is really helpful in the question we are wanting to pursue. Mainly because we took a look at the data then asked a question, doing this helped us become as specific and exploratory as possible to answer our question.

Cleaning and EDA

~ Cleaning ~

Since we're interesting in how the characteristics of ads change from 2018 to 2019. We need to clean column 'StartDate' and 'EndDate' first. We need first convert them to datetime object first, then convert them to year only. We also need to convert the column name of 'StartDate' and 'EndDate' to 'StartYear' and 'EndYear' respectively, since we're focusing on the year only.

~ EDA ~

To explore the data we first focused on the univariate analysis of the column 'Spend'. 'Spend''s column consisted of integers that indicate the "Amount (In USD) spent by the advertiser over the campaign (up to the current date)". We computed the: mean, median, total, and max of 'Spend'. Then we took at look at these statistics with a bivariate approach. We were looking at these statistics by year broken into the 2 years. By plotting the kde graph of the spending with 2019 as the orange line and 2018 as the blue line we can see that

in 2018 there was a lot more around $0 then there were in 2019. This is possibly due to the fact that 2019 isn't over yet and there may be more people/companies that buy ads in the coming months. However, if we look further we can see that the mean of the 2018 versus 2019 is significantly higher, this as well can be due to many things, some may include: 2019 is closer to an election year than 2018, 2019 isn't completely through yet and there are more opportunities for companies to purchase ad space, etc. With the mean being higher we were interested in the sum, and the sum is 5x the amount from 2018! We strongly believe that this is due to the upcoming election year (2020). The same is true for the max per each year: 2019 Max is 19 times 2018's and 2019 isn't even over yet! In the bar graphs we tried to put into perspective the amount of differences between the 2 years.

Then we moved on to the column of 'Impressions'. In the univariate analysis we did the same thing, we calculated the: mean, median, sum, and max. For this we did the same thing, we broke it up by year and look at how the 'Impressions' were based on the 'StartYear'. Again, we grouped by 'StartYear' and took a look at the kde graph of 'Impression'. We noticed that in 2018 there was once again a huge spike around 0. But just like Spend we noticed the trend that the 2019 statistics are higher: The average impression in 2018 was 411219 and in 2019 738327, keep in mind that 2019 is not over and this number is actively going up. However, something that was interesting was the median in 2018 was much higher than 2019. This may be because: 2019 hasn't seen all the ads that it will see because there are still a few months left and that the median is low and the mean is so high because the few people that spend a lot on ads spend a really big amount which has their impressions skyrocket. Moving along to the total impressions per year we see that 2019 is nearly 7 times as much as 2018. Once again there is reason to believe that this number is so high because we are getting closer to election year and the politicians who are in the running are buying these ads.

To take this a step further we did bivariate analysis of the 'Spend' and 'Impressions' columns. We plotted and saved the statistics to determine the dependency. In this case we got that the r value of close to one which shows a strong correlation… we decided to take a deeper look. In both years we see that the correlation is very strong. We also took a look at the spread of 'Spend' and 'Impressions' which showed that the spread of 'Spend' of 2019 was significantly higher than that of 2018 as well as for 'Impressions'. Going deeper into this we wanted to explore if It was necessarily related to our Election Year thought so we took the 'Spend' and broke It up by country. With this we can in fact see that the most of the money is being spent in the USA, which made us want to believe that our thought was true. This holds true with 'Impressions' as well, a huge majority of the impressions came from the United States. However when we weren't looking at the maxes and when we turned to the means we see that the mean for the United States is pretty low compared to the other countries, which was odd since the most impressions was in the US.

Finally, we did some aggregate analysis. Once again we looked at the statistics: count, mean, median, sum, max, and min. But we did this for: StartYear, CandidateBallotInformation, and CountryCode.

Assessment of Missingness

First, we go through 'readme.txt' to have a better understanding of each column, specifically what does null value in each column indicates. We found out that while most of the columns don't have missing data despite having null values, 'EndDate' is one of the few that does contain missing data. Therefore, we decide to start from looking in-depth at the values in 'EndDate'

~ Analyzing column 'EndDate' (NMAR or not?) ~

We first try to come up with a valid reason why end date might be missing. We thought that maybe the ads is still running. Therefore, we thought that 'EndDate' is NMAR and we need another that will tell us whether the ads is still running or not. However, we revisit 'readme.txt' and realize that the end date is predetermined and so even it's still running, the end date will still show a future date. Then we decide to subset the dataframe by

missingness in 'EndDate' and try to see if there's a pattern in other columns. We then notice that candidate who has end date missing tends to pay less. We then decide to run a permutation test of columns of 'EndDate' and 'Spend'.

~ Permutation Test: Part 1 ~

We want to determine if missingness of end date is MAR dependent on the amount of money spent. Our null hypothesis in this test is that there isn't a significant difference in the distribution of money spent and end date, and our alternative hypothesis in this test is that there is a significant difference.

We first plot these two distributions (see below) and notice a huge difference in distribution. Therefore, we use ks-statistic in our permutation test. We find out our observed test statistic, which is around 0.35. After that, we run 1000 trials. Each trial we randomly shuffle the 'Spent' column and randomly assign to an end date and compute the ks statistic for that trial. At the end we plot a histogram of the distribution of these ks-statistics we get and see where does our observed-test statistic lands.

It turns out our observed test statistic is a very extreme value under the distrbution of randomly generated statistics. We get a p-value of 0. Therefore, we reject the null hypothesis and conclude that there is a signification difference, as indicated in the plot, in the distribution of money spent and end date.

~ Permutation Test: Part 2 ~

Now we want to determine if missingness of end date is MAR dependent on the location type. Out null hypothesis is that there isn't a significant difference in the distribution of type of location and end date, and our alternative hypothesis is that there is a significant difference.

We first plot these two into a bar chart. Then we use TVD to find out the observed test statistic. After that, we again run 1000 trials. Each trial we randomly shuffle the 'LocationType' column ad assign it to an end date and compute TV. At the end we plot a histogram of distribution of these tvd-statistics and see where does our observed test statistic lands.

It turns out we get a p-value of 0.885, which suggests that there's isn't a significant difference in the distribution of location type and end date, and we should keep our null hypothesis.

Hypothesis Test

In our hypothesis test, we're interested in the distribution of spending in 2018 and 2018, respectively. We want to know if there is a significant difference between the average amount of money spent in 2018 and the amount of money spent in 2019. Therefore, we decide to conduct a permutation test. Our null hypothesis is that there is no significant difference between them, and out alternative hypothesis is that there is a significant difference between them, and that they come from different distributions. We use difference in means as our test statistics and set the significant level at 0.05.

We then run the test. First we compute the observed test statistic, then we randomly shuffle one of the year's spending column and compute the new test statistic. We do this for 1000 times and plot the distribution and see where the observed test statistic lands. We got a p-value of 0.088. This means even though the p-value is small, we still keep out null hypothesis.

Overall I think this is a good way to answer our question. Because when we compute the mean spending in 2018 and 2019 respectively, we saw a huge difference and we were almost 100 percent confident that there's a significant difference. We see a huge increase in average money spent, which might be due to the fact that a lot of countries election happen in 2020. However, we can not hastily conclude that, and our permutation confirms that. There must be other factors (not in other columns probably) that can explain this increase.

Code

In [130]:

```
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import seaborn as sns
%matplotlib inline
%config InlineBackend.figure_format = 'retina'  # Higher resolution figures

ad_2018 = pd.read_csv('PoliticalAds2018.csv')
ad_2019 = pd.read_csv('PoliticalAds2019.csv')
```

Cleaning and EDA

In [131]:

```
# TODO

ads = pd.concat([ad_2018, ad_2019], ignore_index=True)

# Clean the date column by converting them to useable data
ads['StartDate'] = pd.to_datetime(ads['StartDate'])
ads['StartDate'] = ads['StartDate'].apply(lambda x: x.year)
ads['EndDate'] = pd.to_datetime(ads['EndDate'])
ads['EndDate'] = ads['EndDate'].apply(lambda x: x.year)

# Rename the column
ads.rename(columns = {'StartDate': 'StartYear', 'EndDate': 'EndYear'}, inplace=True)
```

In [132]:

```
''' First we want to conduct univariate analysis, we will explore some columns in de

# Find out the mean, median, total, and max of spending
average_spending = ads['Spend'].mean()
median_spending = ads['Spend'].median()
total_spending = ads['Spend'].sum()
max_spending = ads['Spend'].max()

print('Average spending: ' + str(average_spending) + ' USD')
print('Median spending: ' + str(median_spending) + ' USD')
print('Total spending: ' + str(total_spending) + ' USD')
print('Max spending: ' + str(max_spending) + ' USD')
```

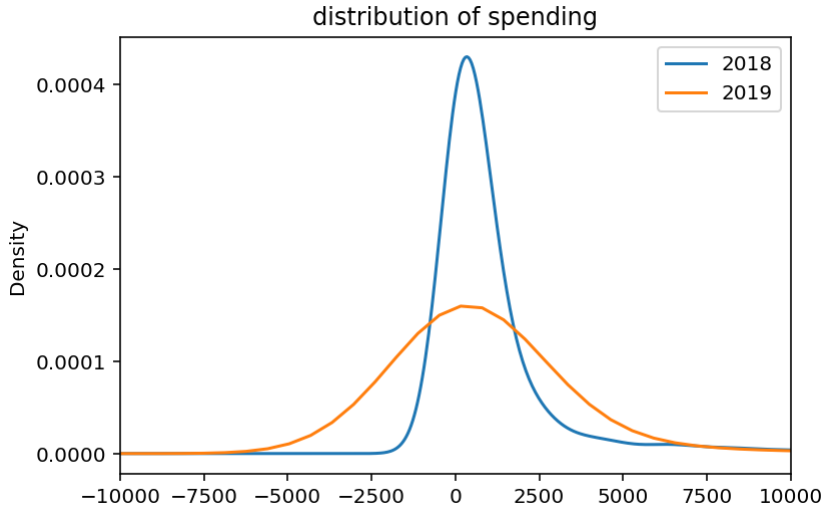
```
Average spending: 1564.2646079321828 USD
Median spending: 184.0 USD
Total spending: 5166766 USD
Max spending: 319467 USD
```

In [133]:

```
# Plot the distribution of spending over the year
ads.groupby('StartYear')['Spend'].plot(kind='kde', legend=True, title='distribution
plt.xlim(-10000, 10000)
```

Out[133]:

```
(-10000, 10000)
```



In [134]:

```
# Find out average spending over the year
mean_spending_over_time = ads.groupby('StartYear')['Spend'].mean()
mean_spending_over_time
```

Out[134]:

```
StartYear
2018    1147.830882
2019    1672.223027
Name: Spend, dtype: float64
```

In [135]:

```
# Find out median spending over the year
median_spending_over_time = ads.groupby('StartYear')['Spend'].median()
median_spending_over_time
```

Out[135]:

```
StartYear
2018    375
2019    148
Name: Spend, dtype: int64
```

In [136]:

```
# Find out total spending over the year
total_spending_over_time = ads.groupby('StartYear')['Spend'].sum()
total_spending_over_time
```

Out[136]:

```
StartYear
2018     780525
2019    4386241
Name: Spend, dtype: int64
```

In [137]:

```
# Find out max spending over the year
max_spending_over_time = ads.groupby('StartYear')['Spend'].max()
max_spending_over_time
```

Out[137]:

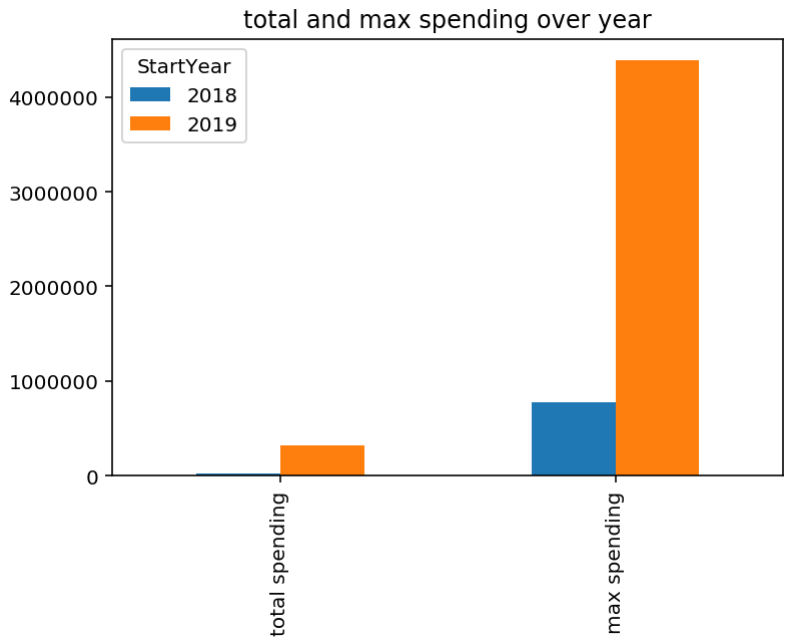
```
StartYear
2018     23742
2019    319467
Name: Spend, dtype: int64
```

In [167]:

```
# Plot the bar chart of mean and median over the year

spending_over_time_index1 = ['total spending', 'max spending']

spending_over_time_table1 = pd.concat([max_spending_over_time, total_spending_over_t

spending_over_time_table1.columns = spending_over_time_index1

spending_over_time_table1.T.plot(kind='bar', title='total and max spending over yea
```

Out[167]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fce51de66d8>
```

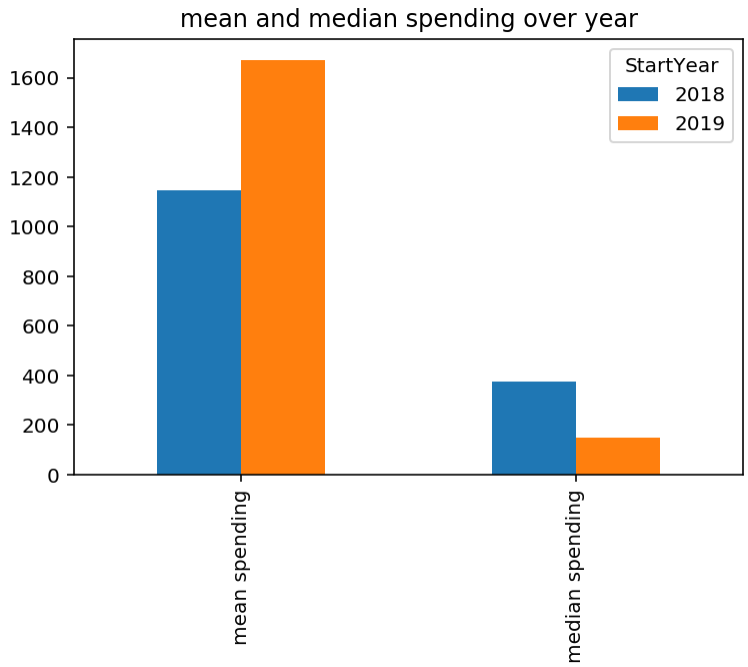


In [168]:

```
# Plot the bar chart of mean and median over time

spending_over_time_index2 = ['mean spending', 'median spending']

spending_over_time_table2 = pd.concat([mean_spending_over_time, median_spending_over

spending_over_time_table2.columns = spending_over_time_index2

spending_over_time_table2.T.plot(kind='bar', title='mean and median spending over ye
```

Out[168]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fce51db10b8>
```



In [140]:

```
''' Then we look in-depth at column of impressions '''

# Find out the mean, median, total, and max impressions over time
average_impression = ads['Impressions'].mean()
median_impression = ads['Impressions'].median()
total_impression = ads['Impressions'].sum()
max_impression = ads['Impressions'].max()

print('Average impression: ' + str(average_impression))
print('Median impression: ' + str(median_impression))
print('Total impression: ' + str(total_impression))
print('Max impression: ' + str(max_impression))
```

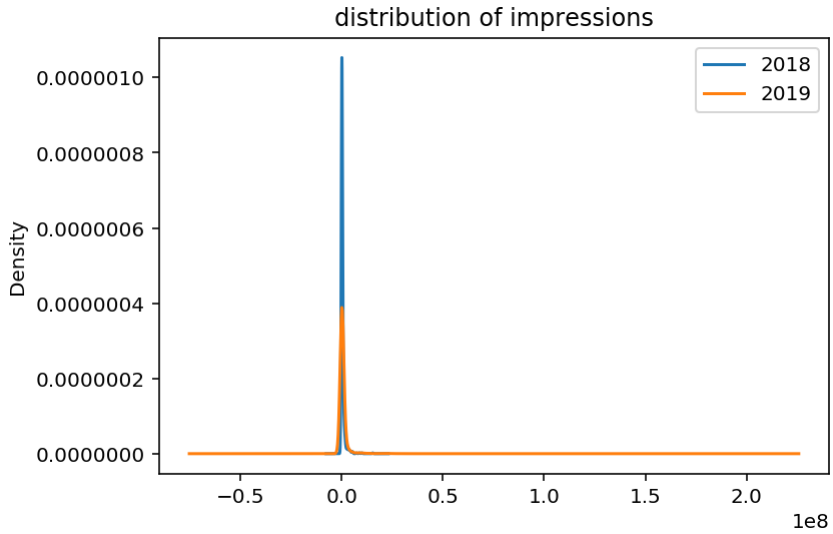
```
Average impression: 670984.5304268847
Median impression: 68914.0
Total impression: 2216261904
Max impression: 150532010
```

In [141]:

```
# Plot the distribution of impressions over the year
ads.groupby('StartYear')['Impressions'].plot(kind='kde', legend=True, title='distrib
```

Out[141]:

```
StartYear
2018    AxesSubplot(0.125,0.125;0.775x0.755)
2019    AxesSubplot(0.125,0.125;0.775x0.755)
Name: Impressions, dtype: object
```

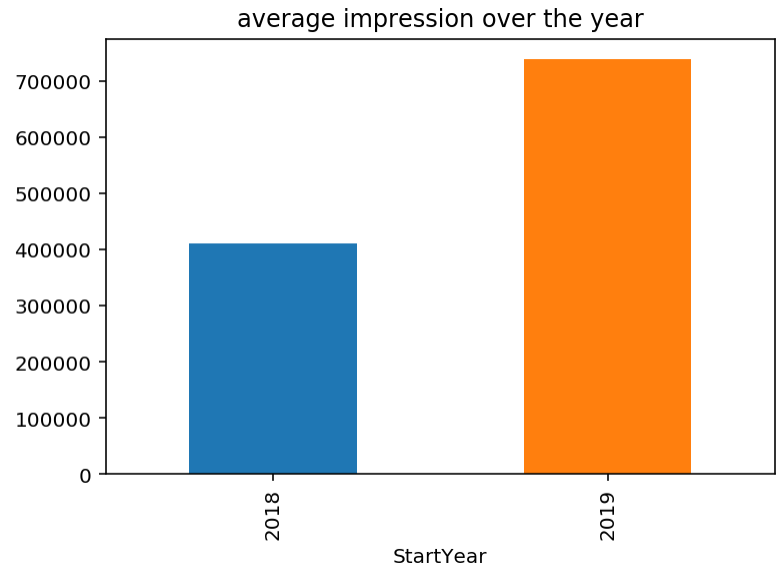


In [142]:

```
# Find out average impression over the year
mean_impression_over_time = ads.groupby('StartYear')['Impressions'].mean()
mean_impression_over_time.plot(kind='bar', title='average impression over the year')
mean_impression_over_time
```

Out[142]:

```
StartYear
2018    411219.861765
2019    738327.258101
Name: Impressions, dtype: float64
```

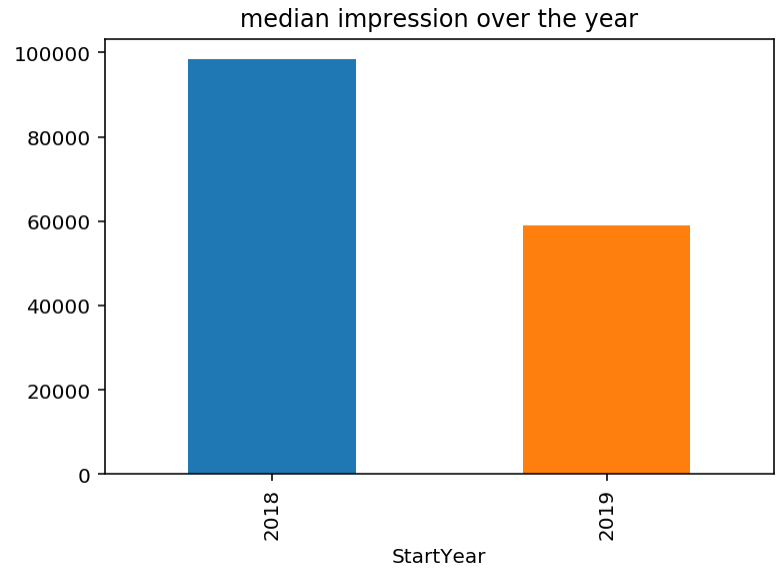


In [143]:

```
# Find out median impression over the year
median_impression_over_time = ads.groupby('StartYear')['Impressions'].median()
median_impression_over_time.plot(kind='bar', title='median impression over the year
median_impression_over_time
```

Out[143]:

```
StartYear
2018    98335.5
2019    59081.0
Name: Impressions, dtype: float64
```

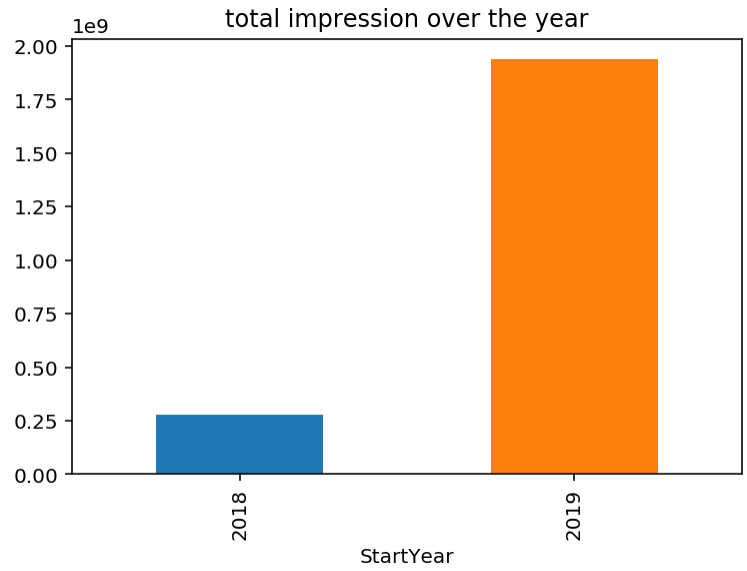


In [144]:

```
# Find out total impression over the year
total_impression_over_time = ads.groupby('StartYear')['Impressions'].sum()
total_impression_over_time.plot(kind='bar', title='total impression over the year')
total_impression_over_time
```

Out[144]:

```
StartYear
2018     279629506
2019    1936632398
Name: Impressions, dtype: int64
```

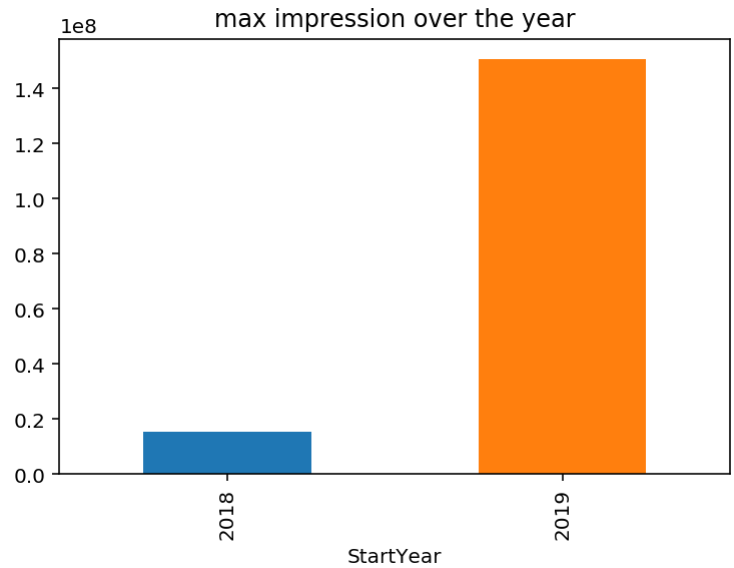


In [145]:

```
# Find out max impression over the year
max_impression_over_time = ads.groupby('StartYear')['Impressions'].max()
max_impression_over_time.plot(kind='bar', title='max impression over the year')
max_impression_over_time
```

Out[145]:

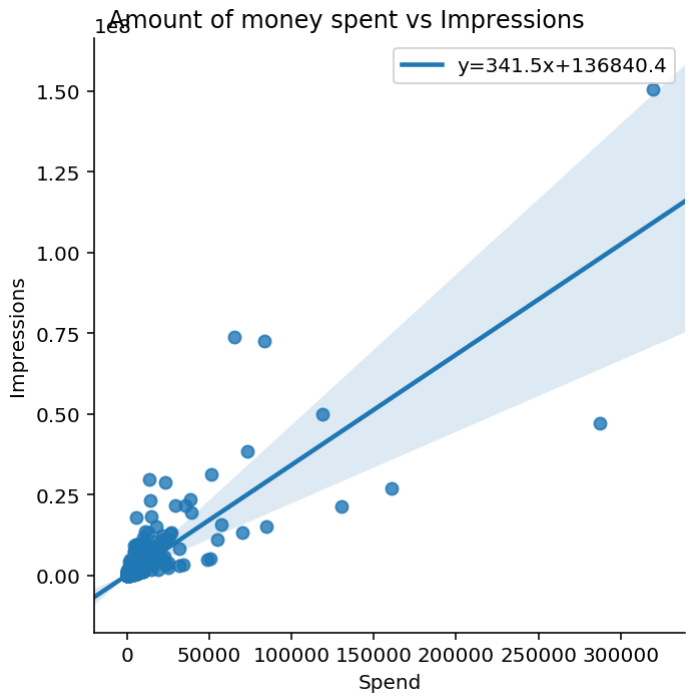
```
StartYear
2018     15322232
2019    150532010
Name: Impressions, dtype: int64
```



In [146]:

```
''' Now we want to conduct bivariate analysis, we will explore some pair of columns
from scipy import stats

# First we would look to look at the relationship between spend and impressions, doe

slope, intercept, r_value, p_value, std = stats.linregress(ads['Spend'], ads['Impres

spend_vs_impression = sns.lmplot(x='Spend', y='Impressions', data=ads,
                        line_kws={'label':"y={0:.1f}x+{1:.1f}".format(slope,intercept

spend_vs_impression.fig.suptitle('Amount of money spent vs Impressions')

plt.legend()

print('Slope: ' + str(slope))
print('Intercept: ' + str(intercept))
print('r value: ' + str(r_value))
print('p value: ' + str(p_value))
print('standard deviation: ' + str(std))
```

```
/opt/conda/lib/python3.6/site-packages/scipy/stats/stats.py:1713: Futu
reWarning: Using a non-tuple sequence for multidimensional indexing is
deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In the future
this will be interpreted as an array index, `arr[np.array(seq)]`, whic
h will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval

Slope: 341.4665931225225
Intercept: 136840.42401414376
r value: 0.8369545641181783
p value: 0.0
standard deviation: 3.8862175405860255
```



In [147]:

```
# It seems like there's a high correlation between money spent and impression. Now w
spend_vs_impression_by_year = sns.lmplot(x='Spend', y='Impressions',
                                         data=ads, hue='StartYear',
                                         height=6, aspect=2)

spend_vs_impression_by_year.fig.suptitle('Amount of money spent vs Impressions over
plt.legend()

#
slope_2018, intercept_2018, r_2018, p_2018, std_2018 = stats.linregress(ads[ads['Sta
                                                                        ads[ads['Sta

print('Linear regression (2018): ' + 'y = ' + str(slope_2018) + 'x + ' + str(interce
print('r_value (2018): ' + str(r_2018))

slope_2019, intercept_2019, r_2019, p_2019, std_2019 = stats.linregress(ads[ads['Sta
                                                                        ads[ads['Sta

print('Linear regression (2019): ' + 'y = ' + str(slope_2019) + 'x + ' + str(interce
print('r_value (2019): ' + str(r_2019))
```

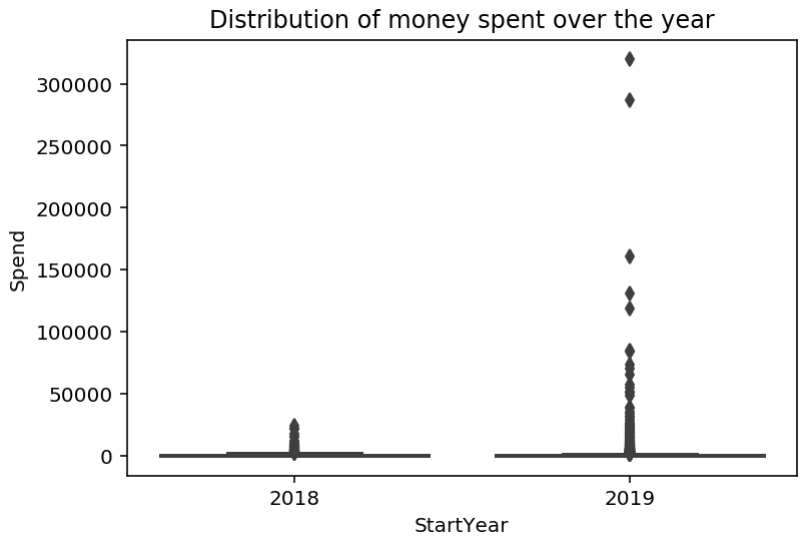
```
Linear regression (2018): y = 338.46412350981024x + 22720.288231625513
r_value (2018): 0.7746013501263307
Linear regression (2019): y = 341.36835810987714x + 167483.228957596
r_value (2019): 0.8377891441990378
```

In [148]:

```
# Now we would like to examine spread of 'Spend' and 'Impressions' over the year by

sns.boxplot(x='StartYear', y='Spend', data=ads).set_title('Distribution of money spe
```

Out[148]:

```
Text(0.5, 1.0, 'Distribution of money spent over the year')
```

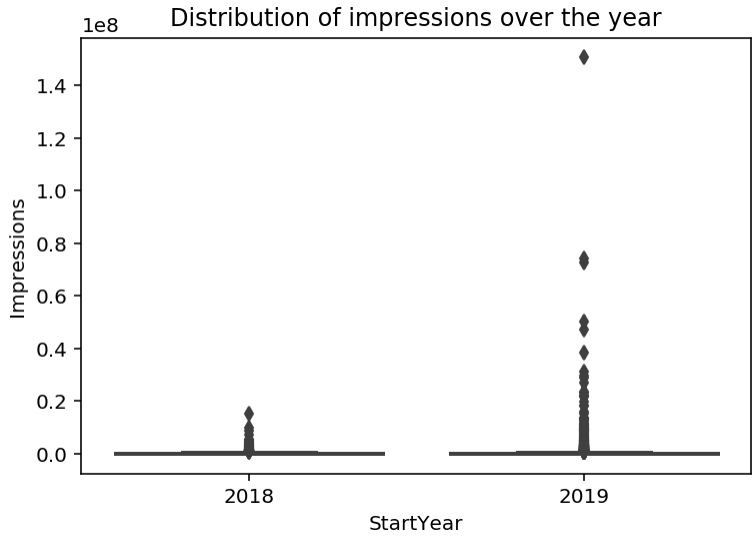


In [149]:

```
# It looks like the outlier is too far away for us to see the boxplot!
# Let's try 'Impressions'
sns.boxplot(x='StartYear', y='Impressions', data=ads).set_title('Distribution of imp

# It's not useful too! Boxplot might not be optimal here. Both data are too spread
```

Out[149]:

```
Text(0.5, 1.0, 'Distribution of impressions over the year')
```

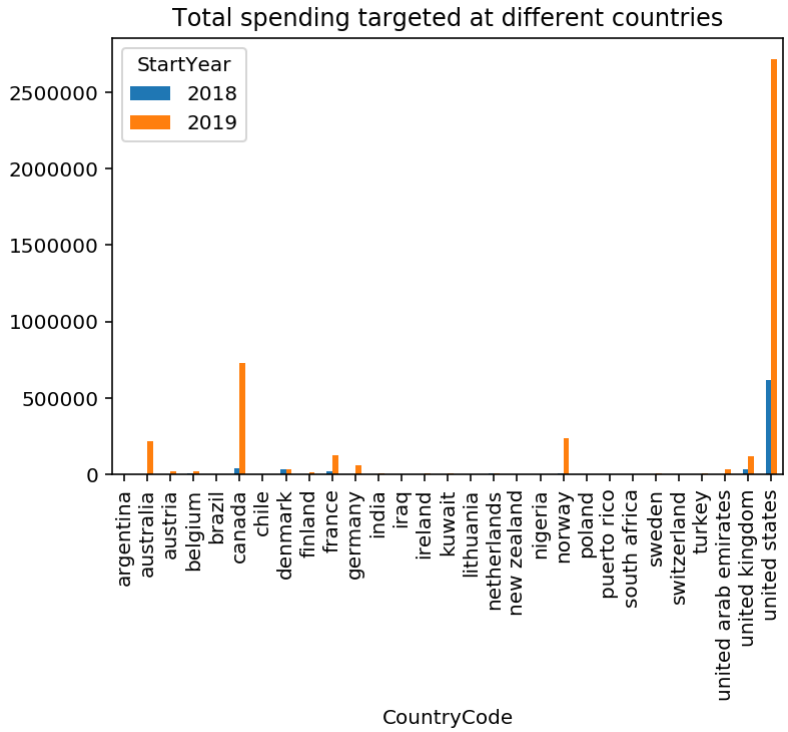


In [150]:

```
# Let's plot some pivot table

# Plot the total spending on different countries
(ads.pivot_table(index='CountryCode', columns='StartYear', values='Spend', aggfunc=
    .plot(kind='bar', title = 'Total spending targeted at different countries'))
```

Out[150]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fce52262b38>
```

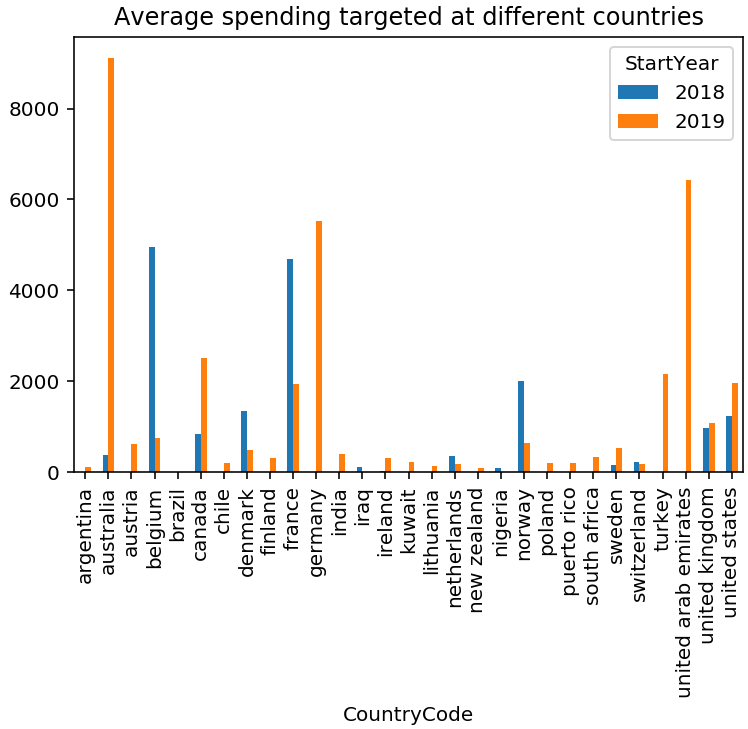


In [151]:

```
# Plot the mean spending on different countries
(ads.pivot_table(index='CountryCode', columns='StartYear', values='Spend', aggfunc=
    .plot(kind='bar', title = 'Average spending targeted at different countries'))
```

Out[151]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fce52189400>
```

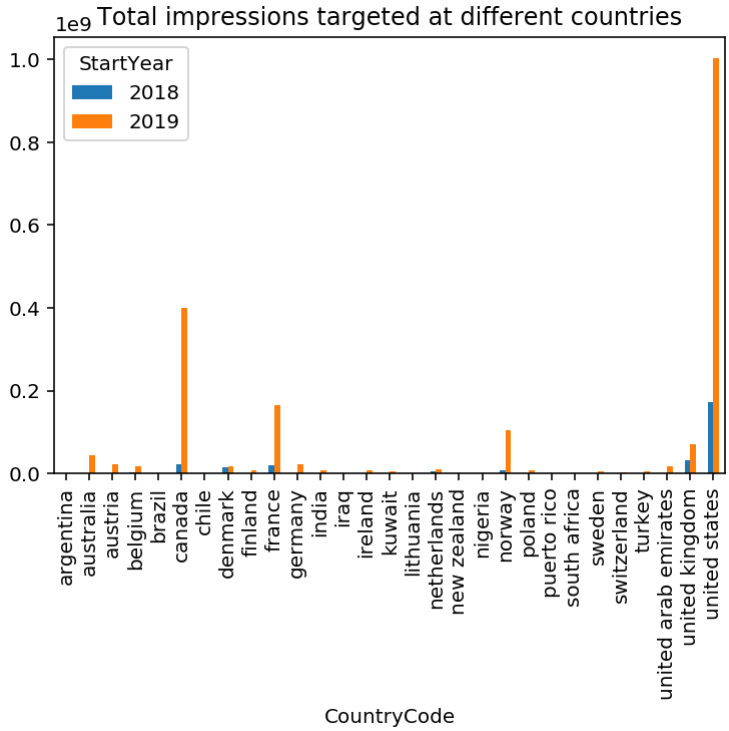


In [152]:

```
# Plot the total impression on different countries
(ads.pivot_table(index='CountryCode', columns='StartYear', values='Impressions', agg
    .plot(kind='bar', title = 'Total impressions targeted at different countries'))
```

Out[152]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fce52042860>
```

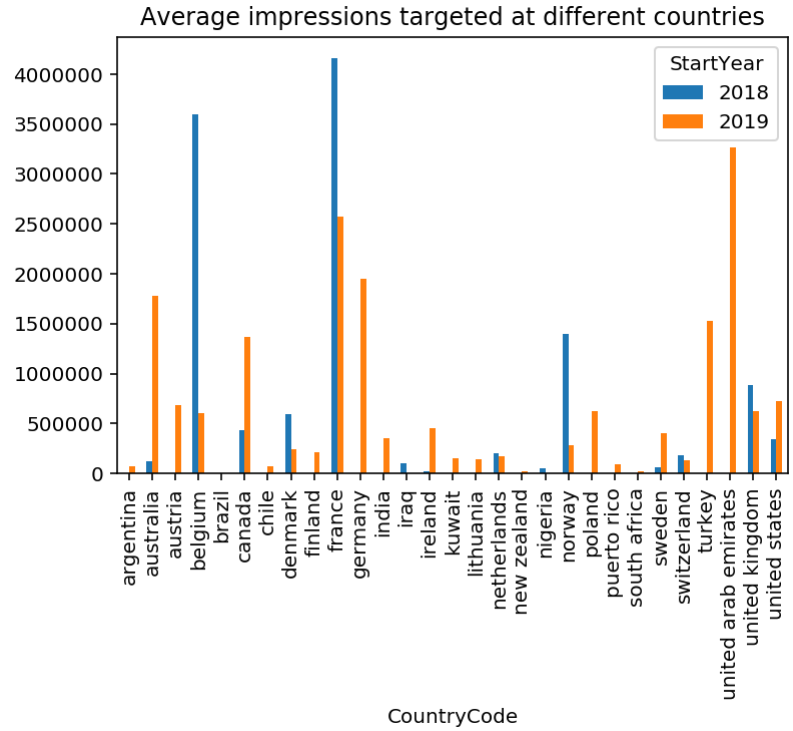


In [153]:

```
# Plot the average impression on different countries
(ads.pivot_table(index='CountryCode', columns='StartYear', values='Impressions', agg
    .plot(kind='bar', title = 'Average impressions targeted at different countries')
```

Out[153]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fce9c2b12e8>
```



In [154]:

```
# Let's do some aggregate analysis
# We want to revisit spending over the year
ads.groupby('StartYear')['Spend'].agg(['count', 'mean', 'median', 'sum', 'max', 'mi
```

Out[154]:

StartYear	count	mean	median	sum	max	min
2018	680	1147.830882	375	780525	23742	0
2019	2623	1672.223027	148	4386241	319467	0

In [155]:

```
# Let's also look at impressions over the year
ads.groupby('StartYear')['Impressions'].agg(['count', 'mean', 'median', 'sum', 'max
```

Out[155]:

StartYear	count	mean	median	sum	max	min
2018	680	411219.861765	98335.5	279629506	15322232	3
2019	2623	738327.258101	59081.0	1936632398	150532010	1

In [156]:

```
# Let's look at top 10 total spending on candidate
(ads.groupby('CandidateBallotInformation')['Spend']
    .agg(['count', 'mean', 'median', 'sum', 'max', 'min'])
    .sort_values(by='sum', ascending=False)
    .head(10))
```

Out[156]:

CandidateBallotInformation	count	mean	median	sum	max	min
Get out the vote	92	181.000000	94.0	16652	1054	4
Warren for President	7	1985.000000	1025.0	13895	5745	9
The Labour Party	2	5174.500000	5174.5	10349	9999	350
Pays Big Waukegan	1	10146.000000	10146.0	10146	10146	10146
Early Voting	1	8500.000000	8500.0	8500	8500	8500
General Election	20	414.450000	249.5	8289	3524	89
Pete for America	4	1624.500000	1031.5	6498	3750	685
Teen Driver Safety	9	363.777778	227.0	3274	1009	63
Make It Legal Florida	2	1527.500000	1527.5	3055	1821	1234
Norbert Hofer	2	1250.000000	1250.0	2500	2000	500

In [157]:

```
# Let's also look at the total impressions top 10 candidates get
(ads.groupby('CandidateBallotInformation')['Impressions']
    .agg(['count', 'mean', 'median', 'sum', 'max', 'min'])
    .sort_values(by='sum', ascending=False)
    .head(10))
```

Out[157]:

CandidateBallotInformation	count	mean	median	sum	max	min
The Labour Party	2	5.220946e+06	5220946.5	10441893	10259185	182708
Norbert Hofer	2	2.864504e+06	2864504.0	5729008	4980733	748275
General Election	20	2.455840e+05	151572.5	4911680	1438239	57130
Make It Legal Florida	2	1.364620e+06	1364620.5	2729241	1640314	1088927
Early Voting	1	2.637803e+06	2637803.0	2637803	2637803	2637803
Get out the vote	92	2.828696e+04	18295.5	2602400	245065	111
Teen Driver Safety	9	2.296919e+05	146002.0	2067227	656515	37676
Boris Johnson	6	3.032958e+05	193669.0	1819775	724536	101457
Warren for President	7	2.020224e+05	52444.0	1414157	632638	3091
Pays Big Waukegan	1	1.134746e+06	1134746.0	1134746	1134746	1134746

In [158]:

```
# Let's also look at which country has advertiser spend most money on
(ads.groupby('CountryCode')['Spend']
    .agg(['count', 'mean', 'median', 'sum', 'max', 'min'])
    .sort_values(by='sum', ascending=False)
    .head(10))
```

Out[158]:

CountryCode	count	mean	median	sum	max	min
united states	1882	1771.821467	122.5	3334568	319467	0
canada	340	2270.476471	513.0	771962	51295	0
norway	377	657.185676	147.0	247759	22976	0
australia	32	6933.031250	252.0	221857	85000	9
united kingdom	148	1049.770270	272.0	155366	31688	0
france	69	2146.550725	707.0	148112	17968	0
denmark	97	715.505155	395.0	69404	6000	4
germany	11	5521.363636	2059.0	60735	25000	0
united arab emirates	5	6438.000000	4298.0	32190	15000	1812
belgium	28	907.178571	116.0	25401	5399	0

In [159]:

```
# Let's also look at which country clicks the most time
(ads.groupby('CountryCode')['Impressions']
    .agg(['count', 'mean', 'median', 'sum', 'max', 'min'])
    .sort_values(by='sum', ascending=False)
    .head(10))
```

Out[159]:

CountryCode	count	mean	median	sum	max	min
united states	1882	6.247765e+05	36608.5	1175829404	150532010	1
canada	340	1.235479e+06	270788.0	420062691	31309945	155
france	69	2.687826e+06	736403.0	185459979	18228091	103
norway	377	2.923706e+05	57565.0	110223734	9299868	2
united kingdom	148	6.877054e+05	183981.5	101780394	10259185	3
australia	32	1.369518e+06	120837.0	43824586	15245241	3845
denmark	97	3.384829e+05	223919.0	32832844	2121387	2067
austria	34	6.802651e+05	269706.5	23129014	6109078	13
germany	11	1.952039e+06	1350616.0	21472425	8958891	119
belgium	28	7.125235e+05	112664.5	19950657	4273887	488

Assessment of Missingness

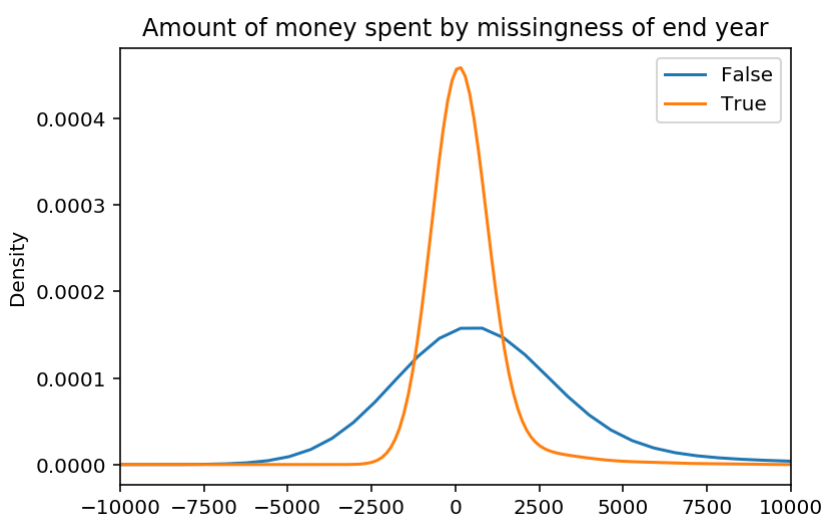
In [160]:

```
# TODO

from scipy.stats import ks_2samp

n_repetitions = 1000

# Plot the distribution of the missingness of 'EndDate', or 'null' and 'non-null'
(
    ads
    .assign(is_null=ads.EndYear.isnull())
    .groupby('is_null')
    .Spend
    .plot(kind='kde', legend=True, title='Amount of money spent by missingness of er

)
plt.xlim(-10000, 10000)

# Find out the observed test statistic
ads['endyear_is_null'] = ads['EndYear'].isnull()
display(ads.groupby('endyear_is_null')['Spend'].mean())


is_null = ads.loc[ads['EndYear'].isnull(), 'Spend']
is_not_null = ads.loc[ads['EndYear'].notnull(), 'Spend']
obs = ks_2samp(is_null, is_not_null).statistic
obs

# Conduct the permutation test and find out p-value
ks_list = []
for _ in range(n_repetitions):
    shuffled_col = ads['Spend'].sample(replace=False, frac=1).reset_index(drop=True)
    shuffled_table = ads.assign(**{'shuffled_spent': shuffled_col})
    grps = shuffled_table.groupby('endyear_is_null')['shuffled_spent']
    ks = ks_2samp(grps.get_group(True), grps.get_group(False)).statistic

    ks_list.append(ks)

pval = np.mean(ks_list > obs)
pval
```

```
endyear_is_null
False    1842.582546
True      441.234756
Name: Spend, dtype: float64
```

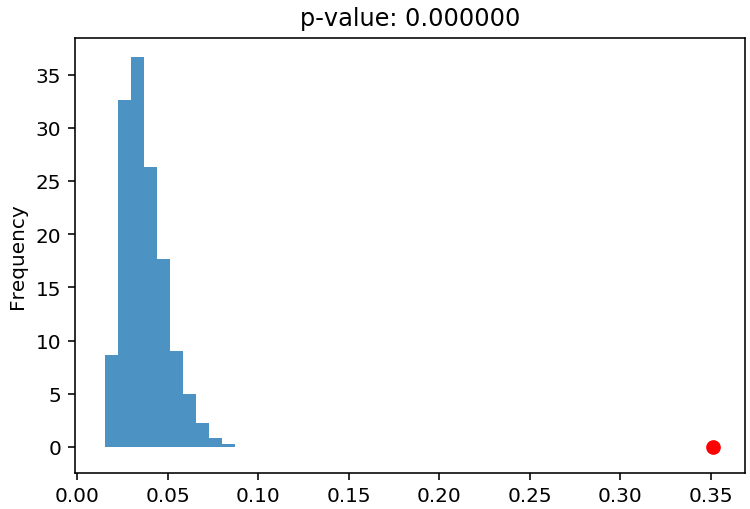
Out[160]:

```
0.0
```



In [161]:

```
# Plot the histogram of the test statistics and see how far the observed test statis
pd.Series(ks_list).plot(kind='hist', density=True, alpha=0.8, title='p-value: %f' %
plt.scatter(obs, 0, color='red', s=40);
```

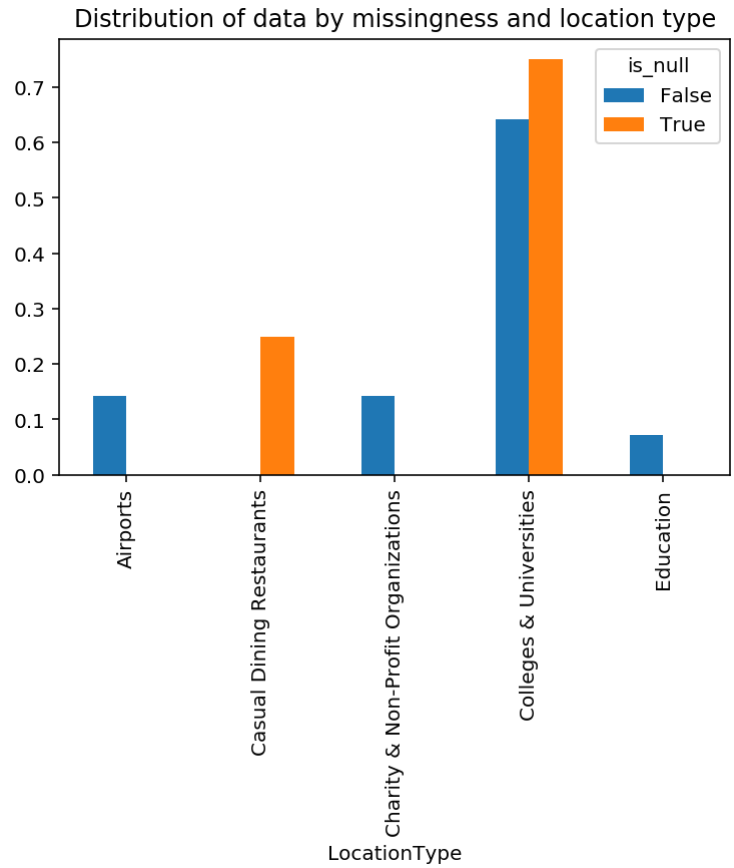


In [162]:

```
distr = (
    ads
    .assign(is_null=ads.EndYear.isnull())
    .pivot_table(index='is_null', columns='LocationType', aggfunc='size')
    .apply(lambda x: x / x.sum(), axis = 1)
)

# Plot a bar showing the counts of data by missingness and start date
distr.T.plot(kind='bar', title='Distribution of data by missingness and location typ

# Determine the observed test statistic
obs = distr.diff().iloc[-1].abs().sum() / 2

# Conduct the permutation test and find out the test statistics
tvds = []
for _ in range(n_repetitions):

    shuffled_col = ads['LocationType'].sample(replace=False, frac=1).reset_index(dro
    shuffled = ads.assign(**{'shuffled_location_type': shuffled_col})
    shuffled = (shuffled.pivot_table(index='endyear_is_null', columns='shuffled_loca
                .apply(lambda x: x / x.sum(), axis = 1))

    tvd = shuffled.diff().iloc[-1].abs().sum() / 2

    tvds.append(tvd)

p_val2 = np.mean(tvds >= obs)
p_val2
```

Out[162]:

```
0.882
```



In [163]:

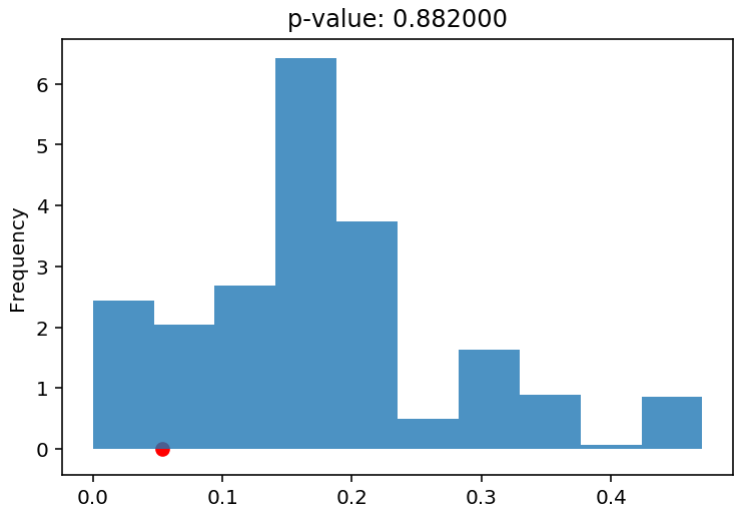
```
p_val2
```

Out[163]:

```
0.882
```

In [164]:

```
# Plot the histogram of the test statistics and see where the observed test statist
pd.Series(tvds).plot(kind='hist', density=True, alpha=0.8, title='p-value: %f' % p_v
plt.scatter(obs, 0, color='red', s=40);
```



Hypothesis Test

In [165]:

```
# TODO

# Create a table which contains only 'StartYear' and 'Spend'
startyear_spending_table = ads[['StartYear', 'Spend']]

# Determine the observed test statistic
obs_stat = mean_spending_over_time.diff().iloc[-1]

differences_list = []


for _ in range(n_repetitions):

    # Shuffled spending
    shuffled_spending = startyear_spending_table['Spend'].sample(replace=False, frac

    # Assign the shuffled column
    original_and_shuffled = startyear_spending_table.assign(**{'Shuffled spending':

    # Groupby StartYear to find out the mean of each year
    group_means = original_and_shuffled.groupby('StartYear').mean().loc[:, 'Shuffle

    # Compute the group differences (which is our test-statistic)
    diff_in_mean = group_means.diff().iloc[-1]

    # Append the test statistic
    differences_list.append(diff_in_mean)

hyp_test_p_value = np.mean(differences_list >= obs_stat)
hyp_test_p_value
```

Out[165]:

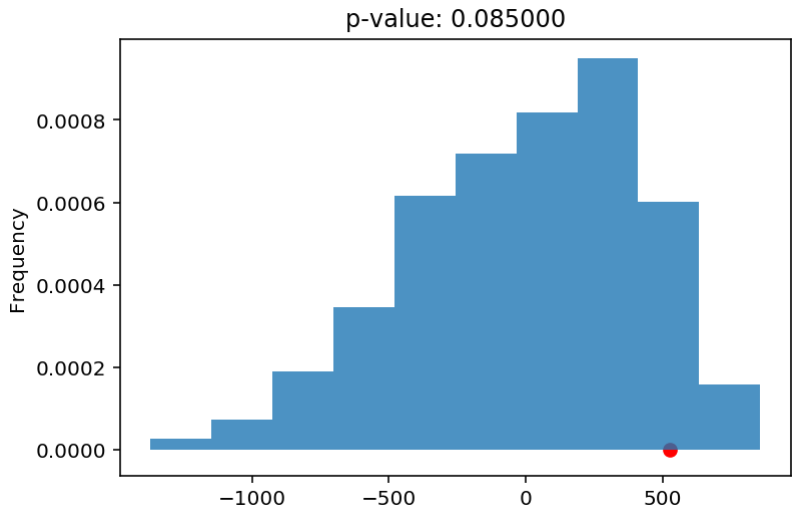
```
0.085
```

In [166]:

```
pd.Series(differences_list).plot(kind='hist', density=True, alpha=0.8, title='p-valu
plt.scatter(obs_stat, 0, color='red', s=40)
```

Out[166]:

```
<matplotlib.collections.PathCollection at 0x7fce51e70fd0>
```



In [ ]:

In [ ]: