AB_Test

June 30, 2021

1 A|B Testing

1.1 *Installing/Importing all the required lib*

[30]:
```
# Data Processing
import pandas as pd
import numpy as np

# Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Statistics
import statsmodels.stats.api as sms
from statsmodels.stats.proportion import proportions_ztest, proportion_confint
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import StandardScaler
import os
import scipy.stats as stats
%matplotlib inline
```

1.2 *Loading Data into Pandas Dataframe*

[31]:
```
# Importing/Loading the csv file using pandas read_csv() function
raw_data = pd.read_csv("Landing_Page.csv")
```

1.3 *Analyzing Data*

[32]:
```
# Having the look of top first 10 rows of the data frame raw_data
raw_data.head(10)
```

[32]:

	user_id	timestamp	group	landing_page	converted
0	851104	2017-01-21 22:11:48.556739	control	old_page	0
1	804228	2017-01-12 08:01:45.159739	control	old_page	0
2	661590	2017-01-11 16:55:06.154213	treatment	new_page	0
3	853541	2017-01-08 18:28:03.143765	treatment	new_page	0
4	864975	2017-01-21 01:52:26.210827	control	old_page	1
5	936923	2017-01-10 15:20:49.083499	control	old_page	0

```
6   679687  2017-01-19 03:26:46.940749  treatment     new_page          1
7   719014  2017-01-17 01:48:29.539573    control     old_page          0
8   817355  2017-01-04 17:58:08.979471  treatment     new_page          1
9   839785  2017-01-15 18:11:06.610965  treatment     new_page          1
```

- ***EDA, Checking DType of columns and Null values***

[33]:
```python
# Using info() to check dtypes of column
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 294478 entries, 0 to 294477
Data columns (total 5 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   user_id       294478 non-null  int64
 1   timestamp     294478 non-null  object
 2   group         294478 non-null  object
 3   landing_page  294478 non-null  object
 4   converted     294478 non-null  int64
dtypes: int64(2), object(3)
memory usage: 11.2+ MB
```

[34]:
```python
# Using isnull() to check if any column is having any Null values
raw_data.isnull().sum()
```

[34]:
```
user_id         0
timestamp       0
group           0
landing_page    0
converted       0
dtype: int64
```

[35]:
```python
# To get summary statistic of a data set you can use describe()
raw_data.describe()
```

[35]:

	user_id	converted
count	294478.000000	294478.000000
mean	787974.124733	0.119659
std	91210.823776	0.324563
min	630000.000000	0.000000
25%	709032.250000	0.000000
50%	787933.500000	0.000000
75%	866911.750000	0.000000
max	945999.000000	1.000000

[36]:
```python
# But to get summary stats for object columns you have to mention␣
↪describe(include='object')
```

```
raw_data.describe(include=['object'])
```

[36]:

```
                         timestamp      group landing_page
count                       294478     294478       294478
unique                      294478          2            2
top     2017-01-21 17:28:18.088125  treatment     old_page
freq                             1     147276       147239
```

- ***We need to make sure that there are no duplicate users as it can cause biasness to our outcome. To avoid that we need to make sure that we drop duplicate user records from our dataset.***

[37]:

```
# Checking for duplicates by comparing the number of unique values with the␣
↪number of rows
raw_data.shape[0] == raw_data.user_id.nunique()
```

[37]: False

[38]:

```
# Calculating the number of duplicate rows
raw_data.shape[0] - raw_data.user_id.nunique()
```

[38]: 3894

[39]:

```
%%time
# To check how time it going to take to execute this cell. It will help us to␣
↪compare it with other option we have.
# pd.concat() to yield the result for same user with different timestamp
pd.concat(i for _, i in raw_data.groupby("user_id") if len(i) > 1)
```

Wall time: 23.5 s

[39]:

```
        user_id                   timestamp      group landing_page  converted
213114   630052  2017-01-07 12:25:54.089486  treatment     old_page          1
230259   630052  2017-01-17 01:16:05.208766  treatment     new_page          0
22513    630126  2017-01-14 13:35:54.778695  treatment     old_page          0
251762   630126  2017-01-19 17:16:00.280440  treatment     new_page          0
11792    630137  2017-01-22 14:59:22.051308    control     new_page          0
…          …                         …          …          …          …
142354   945703  2017-01-08 19:40:51.169351    control     new_page          0
40370    945797  2017-01-11 03:04:49.433736    control     new_page          1
186960   945797  2017-01-13 17:23:21.750962    control     old_page          0
131756   945971  2017-01-22 12:43:54.087275    control     new_page          0
165143   945971  2017-01-16 10:09:18.383183    control     old_page          0

[7788 rows x 5 columns]
```

[53]:

```
%%time
```

```
# Using duplicate() is another way to get all the duplicates rows but its much␣
↪faster than using pd.concat() method
raw_data[raw_data.duplicated(['user_id'], keep=False)].sort_values("user_id")
```

Wall time: 20 ms

[53]:

	user_id	timestamp	group	landing_page	converted
230259	630052	2017-01-17 01:16:05.208766	treatment	new_page	0
213114	630052	2017-01-07 12:25:54.089486	treatment	old_page	1
22513	630126	2017-01-14 13:35:54.778695	treatment	old_page	0
251762	630126	2017-01-19 17:16:00.280440	treatment	new_page	0
183371	630137	2017-01-20 02:08:49.893878	control	old_page	0
…	…	…	…	…	…
142354	945703	2017-01-08 19:40:51.169351	control	new_page	0
186960	945797	2017-01-13 17:23:21.750962	control	old_page	0
40370	945797	2017-01-11 03:04:49.433736	control	new_page	1
165143	945971	2017-01-16 10:09:18.383183	control	old_page	0
131756	945971	2017-01-22 12:43:54.087275	control	new_page	0

[7788 rows x 5 columns]

[41]:
```
# Users exposed to both the groups which is against the principle of A/B tesing
raw_data[raw_data.duplicated(['user_id','group'], keep=False)].
↪sort_values(by="user_id")
```

[41]:

	user_id	timestamp	group	landing_page	converted
230259	630052	2017-01-17 01:16:05.208766	treatment	new_page	0
213114	630052	2017-01-07 12:25:54.089486	treatment	old_page	1
251762	630126	2017-01-19 17:16:00.280440	treatment	new_page	0
22513	630126	2017-01-14 13:35:54.778695	treatment	old_page	0
183371	630137	2017-01-20 02:08:49.893878	control	old_page	0
…	…	…	…	…	…
99479	945703	2017-01-18 06:39:31.294688	control	old_page	0
186960	945797	2017-01-13 17:23:21.750962	control	old_page	0
40370	945797	2017-01-11 03:04:49.433736	control	new_page	1
165143	945971	2017-01-16 10:09:18.383183	control	old_page	0
131756	945971	2017-01-22 12:43:54.087275	control	new_page	0

[3998 rows x 5 columns]

[42]:
```
# Clearly the are some duplicate rows which we need to remove
# Number of duplicate rows
raw_data[raw_data.duplicated(['user_id'], keep=False)].shape
```

[42]: (7788, 5)

- ***After looking at the above result I found that there are several users who got expose to both old and new landing page. This is violating our principle of A/B testing as we need to have only 2 groups to compare (control and treatment) the***

outcome. Now understanding how the users are divided into groups (control & treatment) and drop the users who are present in both the groups.

[43]:
```python
# To check you can use groupby()
raw_data.groupby(['group', 'landing_page'])['converted'].count()
```

[43]:
```
group      landing_page
control    new_page          1928
           old_page        145274
treatment  new_page        145311
           old_page          1965
Name: converted, dtype: int64
```

[44]:
```python
# Another way you can acheive the same result
pd.crosstab(raw_data['group'], raw_data['landing_page'])
```

[44]:
```
landing_page  new_page  old_page
group
control           1928    145274
treatment       145311      1965
```

- ***As you can see above pd.crosstab display the same information as what we acheived by using groupby() but in a very neat & clear way. I would prefer pd.crosstab() for this kind situation.***

[45]:
```python
# Taking out control group only being exposed to old_page and treatment group␣
↪with new_page
# Taking out user with control and Old_page && treatment with new_page
data = raw_data.loc[(raw_data.group == 'control') & (raw_data.landing_page ==␣
↪'old_page')
                   | (raw_data.group == 'treatment') & (raw_data.landing_page␣
↪== 'new_page')]
```

[46]:
```python
# Just confirming that it is done correctly
data.groupby(['group', 'landing_page'])['converted'].count()
```

[46]:
```
group      landing_page
control    old_page        145274
treatment  new_page        145311
Name: converted, dtype: int64
```

[47]:
```python
data[data.duplicated(['user_id'], keep=False)]
```

[47]:

	user_id	timestamp	group	landing_page	converted
1899	773192	2017-01-09 05:37:58.781806	treatment	new_page	0
2893	773192	2017-01-14 02:55:59.590927	treatment	new_page	0

[19]:
```python
# Dropping the duplicate user_id row using drop_duplicates() and keeping the␣
↪first instance of that user_id. You can drop it from the raw_data but I am␣
↪keeing it as it is for this time and only dropping it from are working data␣
↪set.
#raw_data = raw_data.drop_duplicates(subset='user_id',keep='first')
data = data.drop_duplicates(subset='user_id',keep='first')
```

[20]:
```python
# Just confirming that we have and dropped the duplicate user_id and now number␣
↪of rows should be equal to the number of unique user_id.
data.shape[0] == data.user_id.nunique()
```

[20]:
```
True
```

[21]:
```python
# Another way you can acheive the same result
pd.crosstab(data['group'], data['landing_page'])
```

[21]:
```
landing_page  new_page  old_page
group
control              0    145274
treatment       145310         0
```

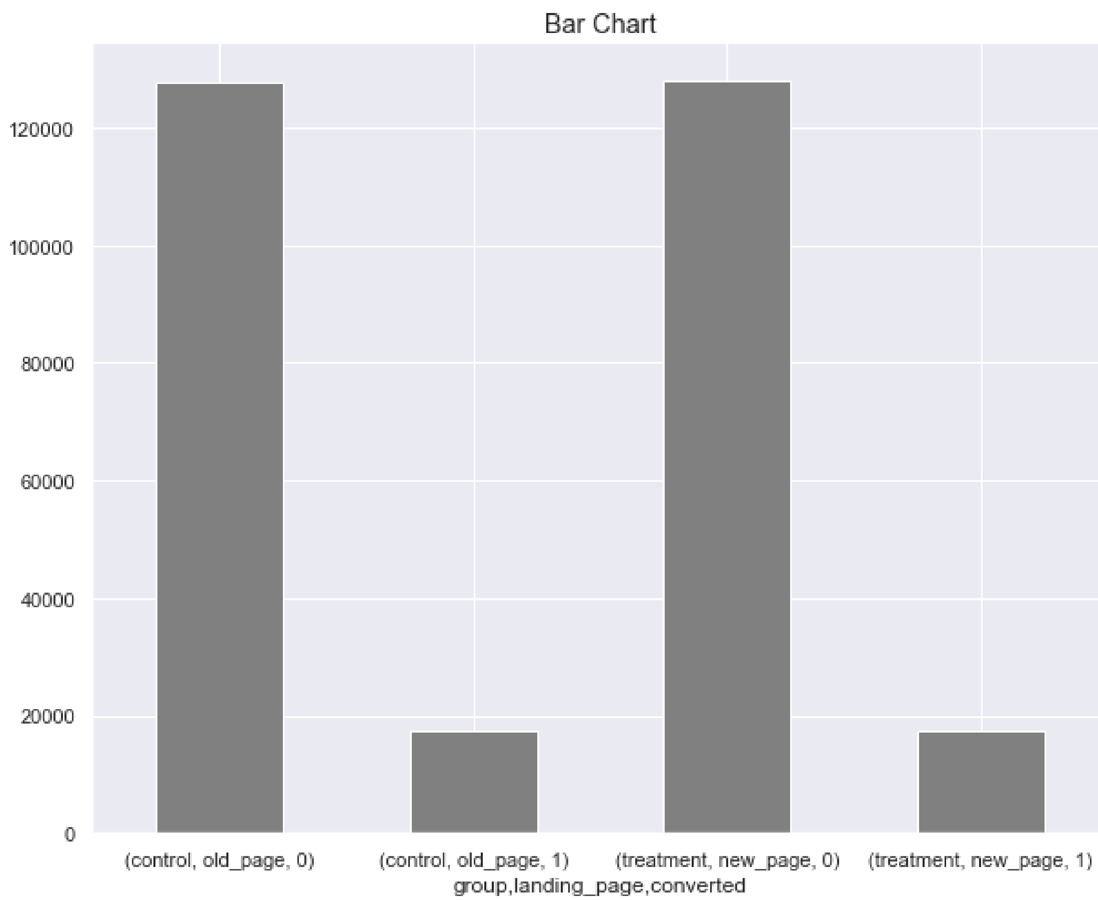
1.4 *Data Visualization*

•

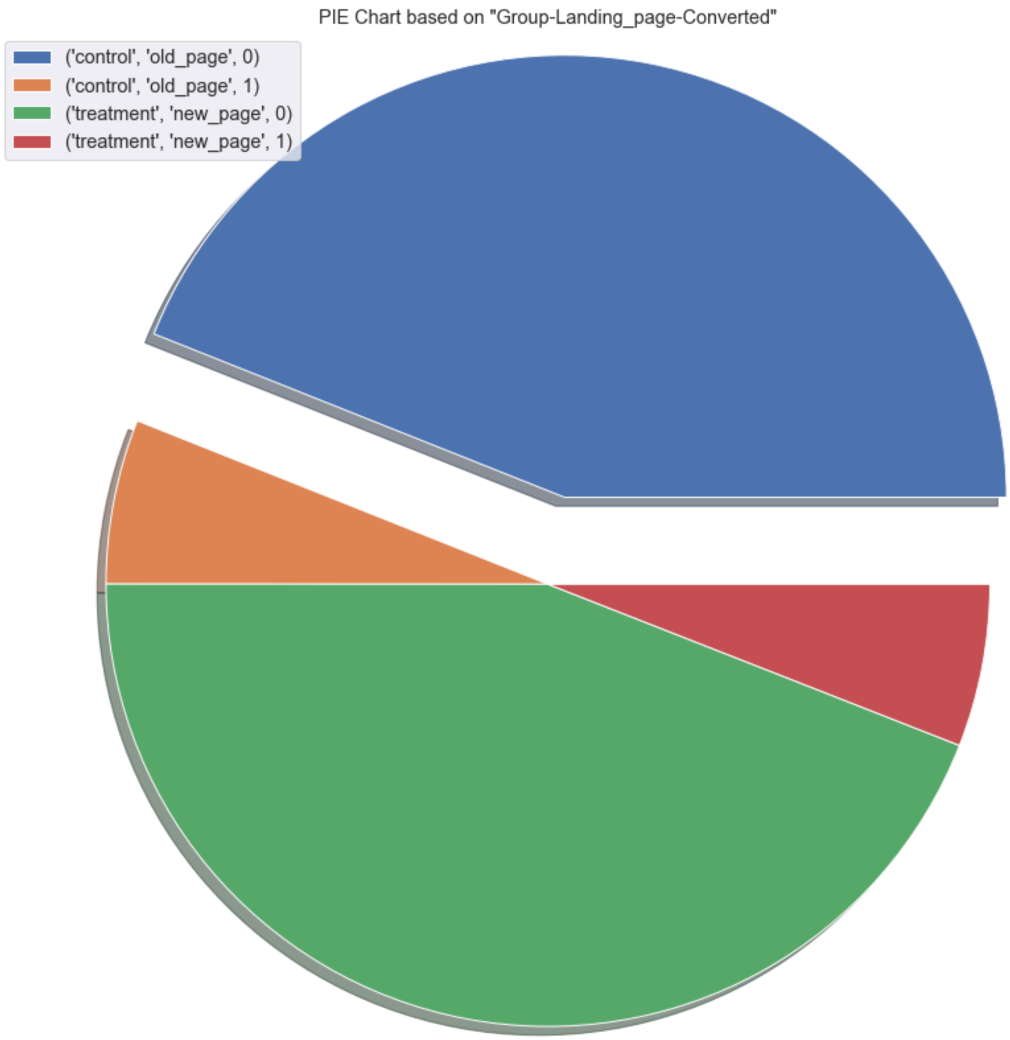
[92]:
```python
all_group = data.groupby(['group','landing_page','converted']).size()
all_group
```

[92]:
```
group      landing_page  converted
control    old_page      0            127785
                         1             17489
treatment  new_page      0            128047
                         1             17264
dtype: int64
```

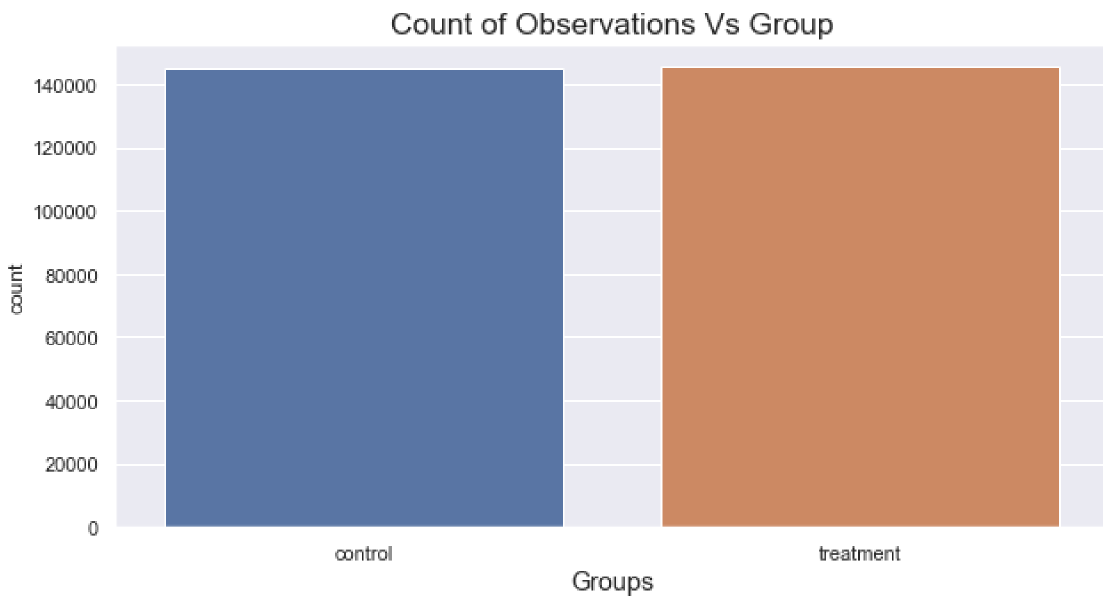
[150]:
```python
plt.figure(figsize=(8,6))
all_group.plot.bar(color='grey')
plt.title('Bar Chart', fontsize='large')
plt.xticks(rotation=0)
plt.show()
```



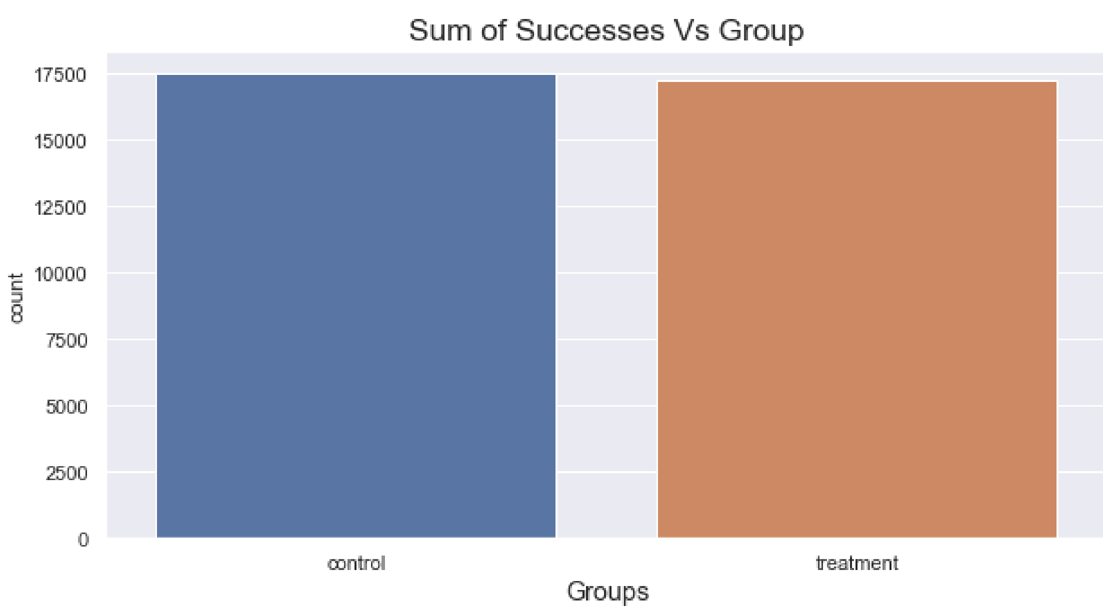
[108]:
```
plt.figure(figsize=(8, 12))
plt.pie(all_group.values, explode = (0.2,0,0,0),shadow = True)
plt.title('PIE Chart based on "Group-Landing_page-Converted"', fontsize='large')
plt.legend(all_group.index, loc = 'upper left',fontsize="large")
plt.show()
```



[154]:
```
sns.set(rc={'figure.figsize':(8,5)})
sns.countplot(x='group', data=data)
plt.title('Count of Observations Vs Group', fontsize = 'x-large')
plt.xlabel('Groups', fontsize = 'large')
plt.show()
plt.close()
```




```
[153]: sns.set(rc={'figure.figsize':(8,5)})
sns.countplot(x='group', data=data[data['converted']==1])
plt.title('Sum of Successes Vs Group', fontsize = 'x-large')
plt.xlabel('Groups', fontsize = 'large')
plt.show()
plt.close()
```



1.5 *Experiment - Hypothesis Testing*

- ***Lets calculate the Conversion Rate:***

Overall

[17]:
```
# Probability of conversion regardless of the page/group
(data.query('converted == 1').converted.count())/data.shape[0]*100
```

[17]: 11.959708724499627

Group - Control

[18]:
```
# Probability of conversion of control/old_page
((data.query('(converted == 1) & (group == "control")').converted.count())/
↪data[data.group == 'control'].shape[0])*100
```

[18]: 12.03863045004612

Group - Treatment

[19]:
```
# Probability of conversion of treatment/new_page
((data.query('(converted == 1) & (group == "treatment")').converted.count())/
↪data[data.group == 'treatment'].shape[0])*100
```

[19]: 11.880806551510565

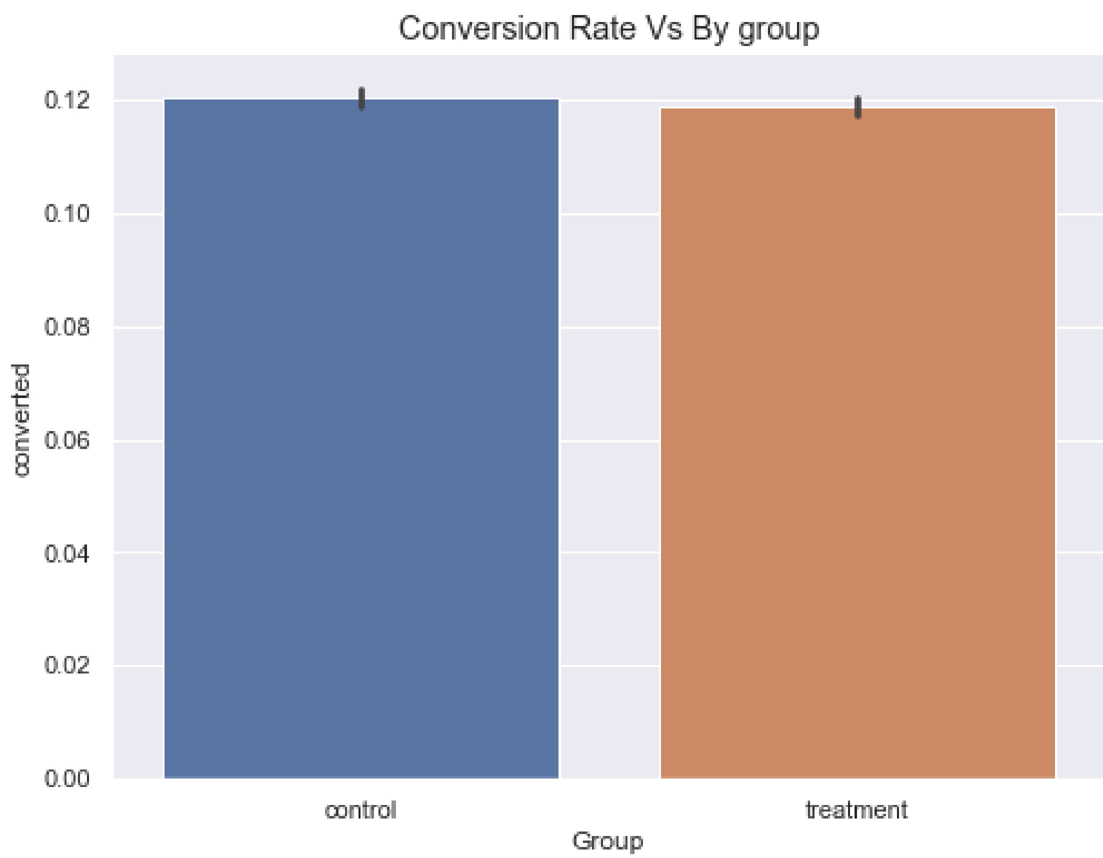
- ***Checking some basic level of statistic of the dataset***

[121]:
```
conv_rates = data.groupby('group')['converted']

# Standard deviation of the converted proportion
std_prop = lambda i: np.std(i, ddof=0)
# Standard error of the converted proportion (std / sqrt(n))
stde_prop = lambda i: stats.sem(i, ddof=0)

conv_rates = conv_rates.agg([np.mean, std_prop, stde_prop])
conv_rates.columns = ['conversion_rate', 'std_deviation', 'std_error']

# Display in float format upto 3 decimal places
conv_rates.style.format('{:.4f}')
```

[121]: <pandas.io.formats.style.Styler at 0x1c6850075b0>

[132]:
```
plt.figure(figsize=(8,6))
sns.barplot(x=data['group'], y=data['converted'])
```

```
plt.title('Conversion Rate Vs By group',fontsize='large')
plt.xlabel('Group', fontsize='medium')
plt.show()
```



- ***Calculating Z_Score and P_Value for our Hypothesis***

$\boldsymbol{H : p = p}$

$\boldsymbol{H : p \quad p}$

[143]:
```
# Importing proportions_ztest and proportion_confint to calculate z_value,␣
↪p_value and Confidence Interval
from statsmodels.stats.proportion import proportions_ztest, proportion_confint

# Dividing data into 2 parts(Control & Treatment) and storing their converted␣
↪values
control_group = data[data['group'] == 'control']['converted']
treatment_group = data[data['group'] == 'treatment']['converted']
```

```
# Total number of successes each group stored as a list.
Total_successes = [control_group.sum(), treatment_group.sum()]

# Total number of observation in each group stored as a list.
Total_Obs = [control_group.count(), treatment_group.count()]

# Applying proportions_ztest() and proportion_confint()
z_score, p_value = proportions_ztest(Total_successes, nobs=Total_Obs)
(low_con, low_treat), (up_con, up_treat) = proportion_confint(Total_successes,␣
↪nobs=Total_Obs, alpha=0.05)
```

[144]:
```
print(f'z_score: {z_score:.3f}')
print(f'p_value: {p_value:.3f}')
print(f'CI 95% - control group: [{low_con:.3f}, {up_con:.3f}]')
print(f'CI 95% - treatment group: [{low_treat:.3f}, {up_treat:.3f}]')
```

```
z_score: 1.312
p_value: 0.190
CI 95% - control group: [0.119, 0.122]
CI 95% - treatment group: [0.117, 0.120]
```

Since our p-value is above the =0.05 threshold and the z value is $1.312 < 1.96$, the null hypothesis, H , cannot be rejected (Failed to reject Null Hypothesis). In other words, both the groups are similar in terms of conversion rate.

Furthermore, we can say with 95% confidence that the treatment conversion rate lies between 11.7% - 12%. Similarly, with the same confidence of 95%, we can say that the control conversion rate lies between 11.9% - 12.2% which is slightly better. This is a further proof that the new design is not likely to be an improvement on the old design.

In conclusion, from the business point of view, the new landing page does not make a convincing case for investment.