In [1]:
```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

from sklearn.cluster import KMeans
%matplotlib inline
```

In [2]:
```python
## Loading the raw data in python

df = pd.read_csv('Events Raw Data.csv')

# print the shape
print(df.shape)

#runs the first 5 rows
df.head()
```

(14783, 19)

Out[2]:

	CUSTOMER	Age	Gender	Postcode	District	Constituency	latitude	longitude	Current_Status	Total_Household_Income	How often you attend Entertaining events in a year
0	1	40-49	Female	AL9 7BN	Welwyn Hatfield	Welwyn Hatfield	51.727429	-0.183648	Married with children	50,000 to 74,999	4 Times a Yea
1	2	60-64	Male	AL7 1ZW	Welwyn Hatfield	Welwyn Hatfield	51.804482	-0.195898	Married with children	35,000 to 49,999	3 Times a Yea
2	3	50-59	Male	AB4 5JX	Aberdeenshire	Banff and Buchan	57.692200	-2.022295	Married with children	75,000 to 99,999	3 Times a Yea
3	4	50-59	Female	B24 0DL	Birmingham	Birmingham, Erdington	52.526595	-1.822211	Married without children	35,000 to 49,999	1 Times a Yea
4	5	60-64	Male	E17 5QP	Waltham Forest	Walthamstow	51.593979	-0.040803	Married with children	100,000 to 149,999	3 Times a Yea

Data Preprocessing and EDA

In [3]:
```python
# Check for null entries
df.isnull().sum()
```

Out[3]:
```
CUSTOMER                                                          0
Age                                                               0
Gender                                                            0
Postcode                                                          0
District                                                          0
Constituency                                                      0
latitude                                                          0
longitude                                                         0
Current_Status                                                    0
Total_Household_Income                                            5
How often you attend Entertaining events in a year?               5
Social_Media                                                      5
How many hours are you willing to travel to attend an event?      5
Do you enjoy adrenaline-rush activities?                          5
Are food areas, coffee areas, bars & toilets important to you?    5
What is your favourite attraction from below:                     5
Were you satisfied with the last event you attended with us?      5
Would you recommend our events to other people?                   5
Did you find our events value for money?                          5
dtype: int64
```

In [4]:
```python
# Remove rows with null entries
df.dropna(inplace = True)
print(df.shape)
```

(14778, 19)

In [5]:
```python
# Remove any duplicate entries
df.drop_duplicates(inplace = True)
print(df.shape)
```

(14778, 19)

In [6]:
```python
# investigate elements per feature

for cols in df.columns:
    unique_vals = df[cols].unique()
    nr_unique_vals = len(unique_vals)
    if nr_unique_vals < 10:
        print("Feature ", cols, " has ", nr_unique_vals, " unique values including ", unique_vals)
    else:
        print("Feature ", cols, " has ", nr_unique_vals)
```

```
Feature  CUSTOMER  has  14778
Feature  Age  has  9  unique values including  ['40-49' '60-64' '50-59' '65 or older' '33-39' '26-32' '21-25'
 '17 or younger' '18-20']
Feature  Gender  has  2  unique values including  ['Female' 'Male']
Feature  Postcode  has  14674
Feature  District  has  225
Feature  Constituency  has  358
Feature  latitude  has  13242
Feature  longitude  has  13262
Feature  Current_Status  has  8  unique values including  ['Married with children' 'Married without children'
 'Living with partner'
 'Divorced' 'Single, never married' 'Separated' 'Widowed' 'Step parent']
Feature  Total_Household_Income  has  8  unique values including  ['50,000 to 74,999' '35,000 to 49,999' '75,000 to 99,999'
 '100,000 to 149,999' '150,000 to 199,999' '200,000 or more'
 '20,000 to 34,999' 'Less than 20,000']
Feature  How often you attend Entertaining events in a year?  has  5  unique values including  ['4 Times a Year' '3 Times a Year' '1 Times a Year' '2 Times a Year'
 '5+ Times a Year']
Feature  Social_Media  has  5  unique values including  ['Up to 1 hour' "I don't spend any time at all" 'Half a day'
 'Up to 2 hours' 'More than half a day']
Feature  How many hours are you willing to travel to attend an event?  has  7  unique values including  ['3 hours' '5 hours' '1 hour' '6 hours plus' '4 hours' '2 hours'
 'Depends from the Event']
Feature  Do you enjoy adrenaline-rush activities?  has  7  unique values including  ['Entirely disagree' 'Somewhat disagree' 'Entirely agree'
 'Mostly disagree' 'Neither agree nor disagree' 'Somewhat agree'
 'Mostly agree']
Feature  Are food areas, coffee areas, bars & toilets important to you?  has  7  unique values including  ['Entirely agree' 'Somewhat agree' 'Neither agree nor disagree'
 'Mostly agree' 'Mostly disagree' 'Somewhat disagree' 'Entirely disagree']
Feature  What is your favourite attraction from below:  has  7  unique values including  ['Ice Bar' 'Happy Rollercoaster' 'Crazy rollercoaster ride'
 'Kids Playground' 'Wooden Bar' 'Ice skating' 'Tree climbing']
Feature  Were you satisfied with the last event you attended with us?  has  7  unique values including  ['Somewhat agree' 'Mostly agree' 'Entirely agree'
 'Neither agree nor disagree' 'Mostly disagree' 'Entirely disagree'
 'Somewhat disagree']
Feature  Would you recommend our events to other people?  has  7  unique values including  ['Somewhat Unlikely'
 'Somewhat Likely' 'Mostly Unlikely' 'Very Unlikely'
 'Very Likely' 'Mostly Likely' 'Neither agree nor disagree']
Feature  Did you find our events value for money?  has  7  unique values including  ['Somewhat disagree' 'Entirely disagree' 'Entirely agree' 'Somewhat agree'
 'Mostly disagree' 'Mostly agree' 'Neither agree nor disagree']
```

In [7]:
```python
df.columns
```

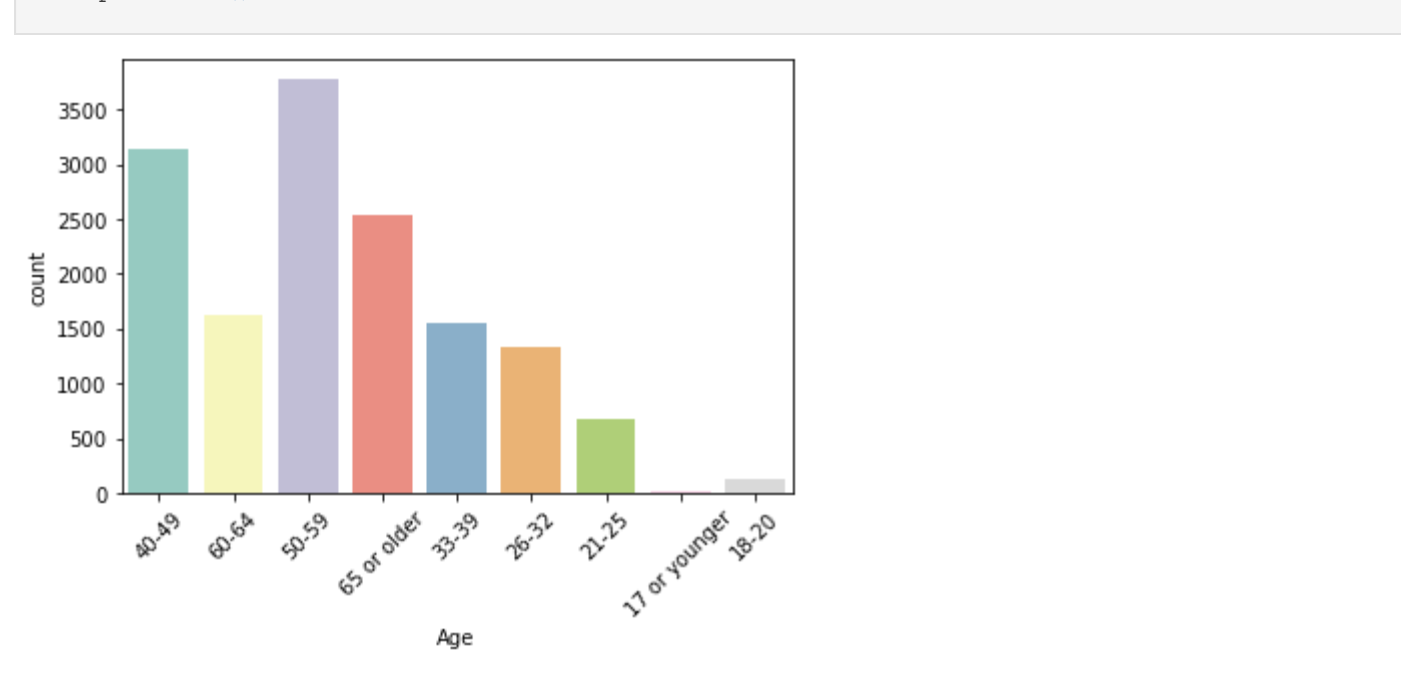
Out[7]:
```
Index(['CUSTOMER', 'Age', 'Gender', 'Postcode', 'District', 'Constituency',
       'latitude', 'longitude', 'Current_Status', 'Total_Household_Income',
       'How often you attend Entertaining events in a year?', 'Social_Media',
       'How many hours are you willing to travel to attend an event?',
       'Do you enjoy adrenaline-rush activities?',
       'Are food areas, coffee areas, bars & toilets important to you?',
       'What is your favourite attraction from below:',
       'Were you satisfied with the last event you attended with us?',
       'Would you recommend our events to other people?',
       'Did you find our events value for money?'],
      dtype='object')
```

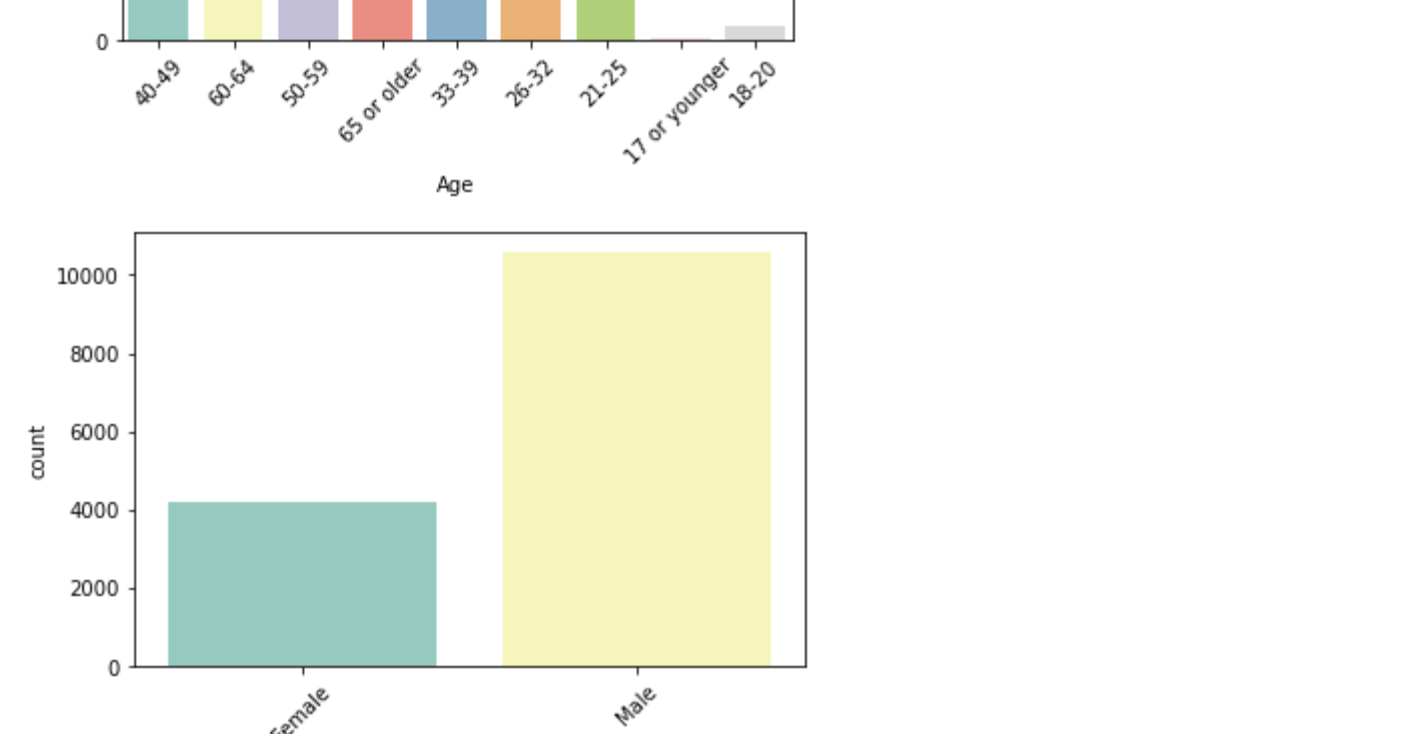
In [8]:
```python
df.head()
```

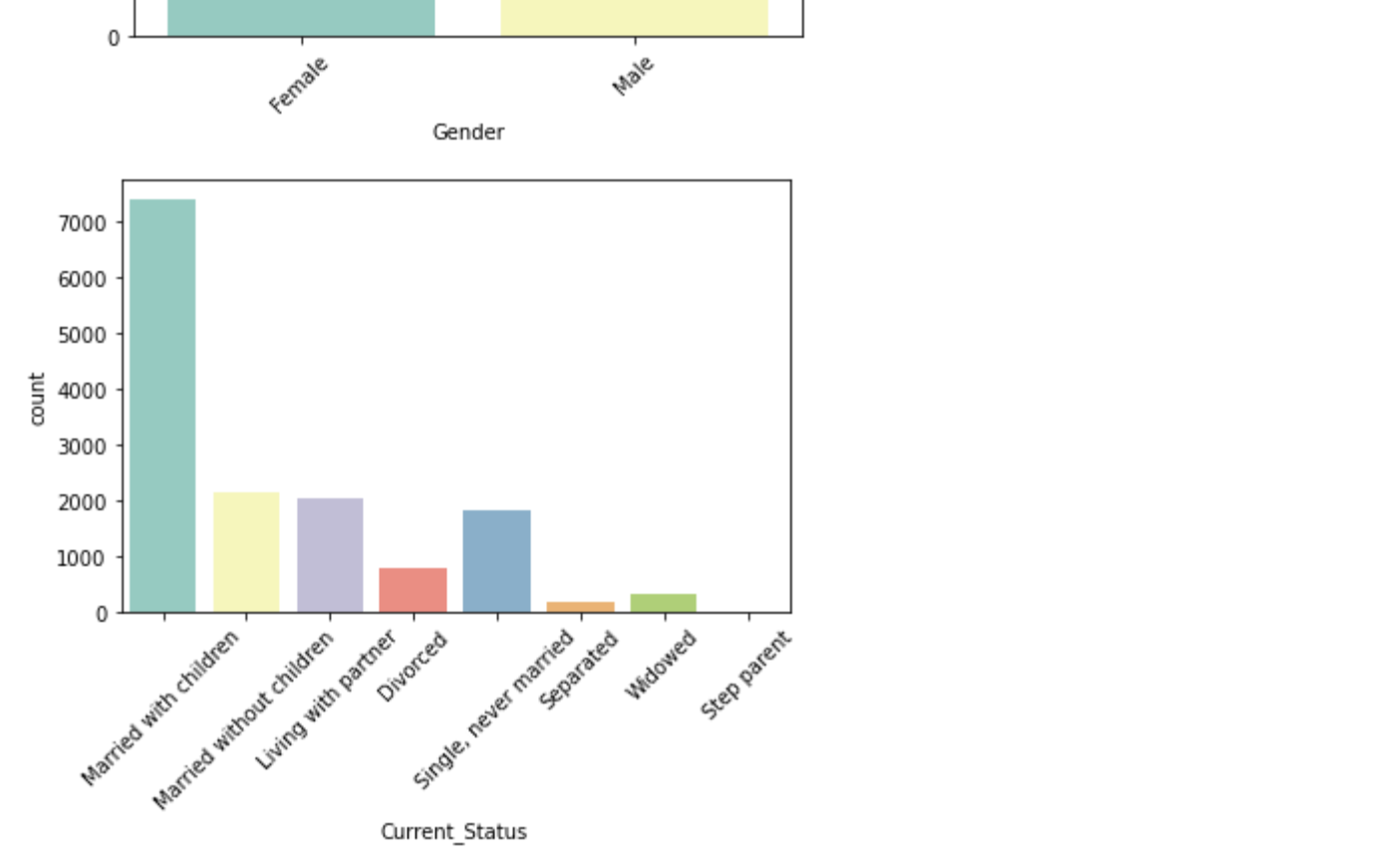
Out[8]:

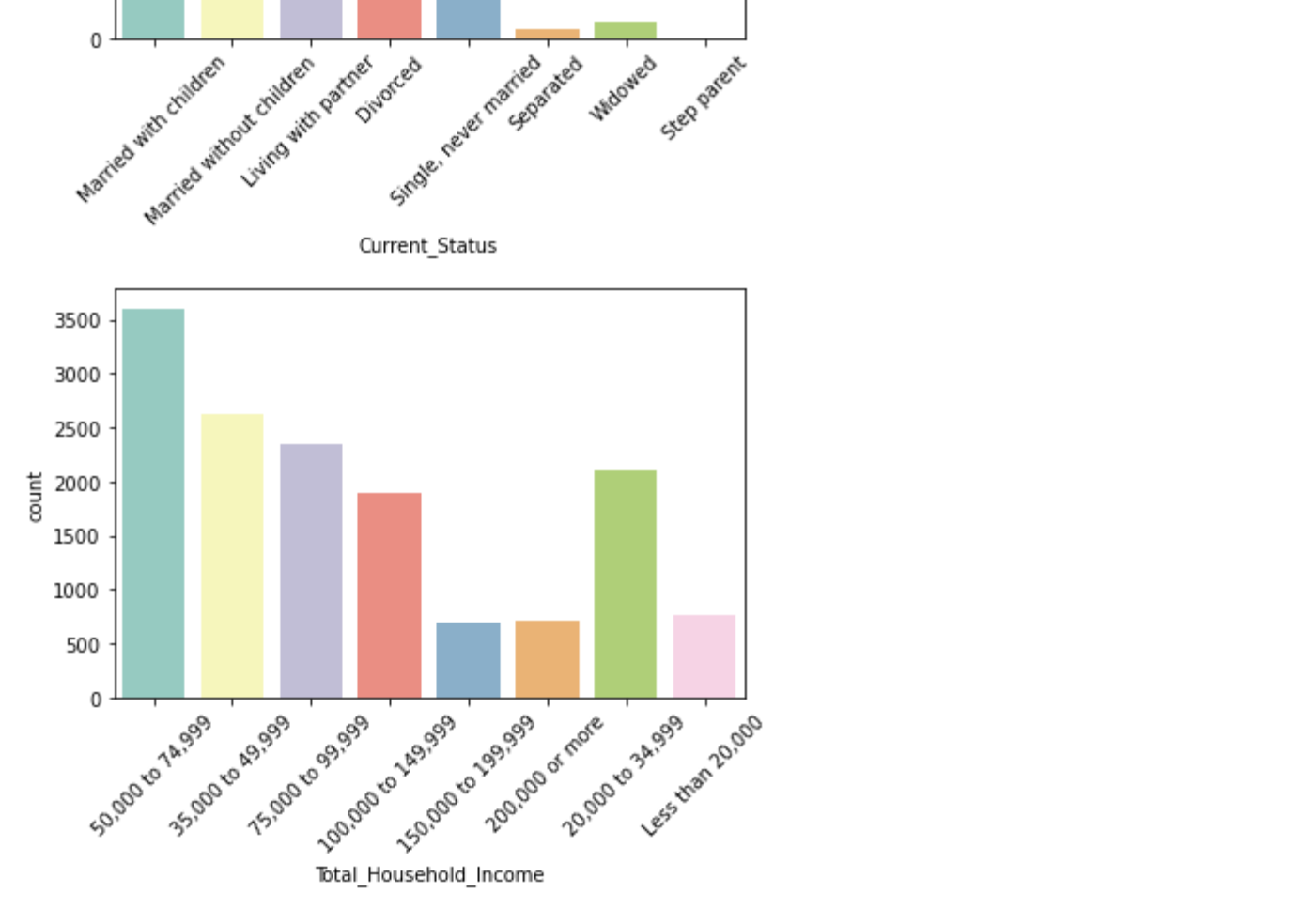
	CUSTOMER	Age	Gender	Postcode	District	Constituency	latitude	longitude	Current_Status	Total_Household_Income	How often you attend Entertaining events in a year
0	1	40-49	Female	AL9 7BN	Welwyn Hatfield	Welwyn Hatfield	51.727429	-0.183648	Married with children	50,000 to 74,999	4 Times a Yea
1	2	60-64	Male	AL7 1ZW	Welwyn Hatfield	Welwyn Hatfield	51.804482	-0.195898	Married with children	35,000 to 49,999	3 Times a Yea
2	3	50-59	Male	AB4 5JX	Aberdeenshire	Banff and Buchan	57.692200	-2.022295	Married with children	75,000 to 99,999	3 Times a Yea
3	4	50-59	Female	B24 0DL	Birmingham	Birmingham, Erdington	52.526595	-1.822211	Married without children	35,000 to 49,999	1 Times a Yea
4	5	60-64	Male	E17 5QP	Waltham Forest	Walthamstow	51.593979	-0.040803	Married with children	100,000 to 149,999	3 Times a Yea

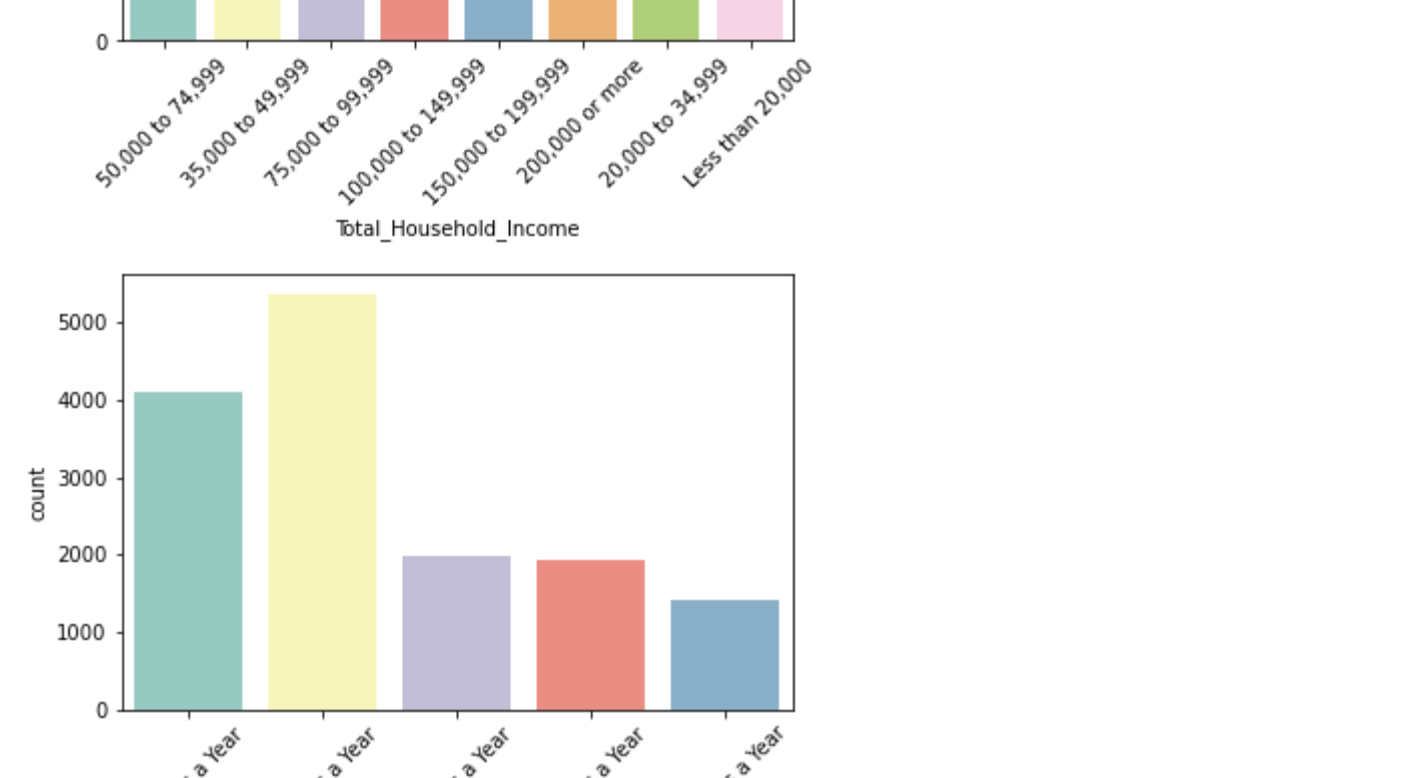
In [9]:
```python
# looping through all the categorical features by our y variable - see if there is relationship

features = ['Age', 'Gender',
            'Current_Status', 'Total_Household_Income',
            'How often you attend Entertaining events in a year?', 'Social_Media',
            'How many hours are you willing to travel to attend an event?',
            'Do you enjoy adrenaline-rush activities?',
            'Are food areas, coffee areas, bars & toilets important to you?',
            'What is your favourite attraction from below:',
            'Were you satisfied with the last event you attended with us?',
            'Would you recommend our events to other people?',
            'Did you find our events value for money?']

for f in features:
    sns.countplot(x = f, data = df, palette = 'Set3')
    plt.xticks(rotation=45)
    plt.show()
```

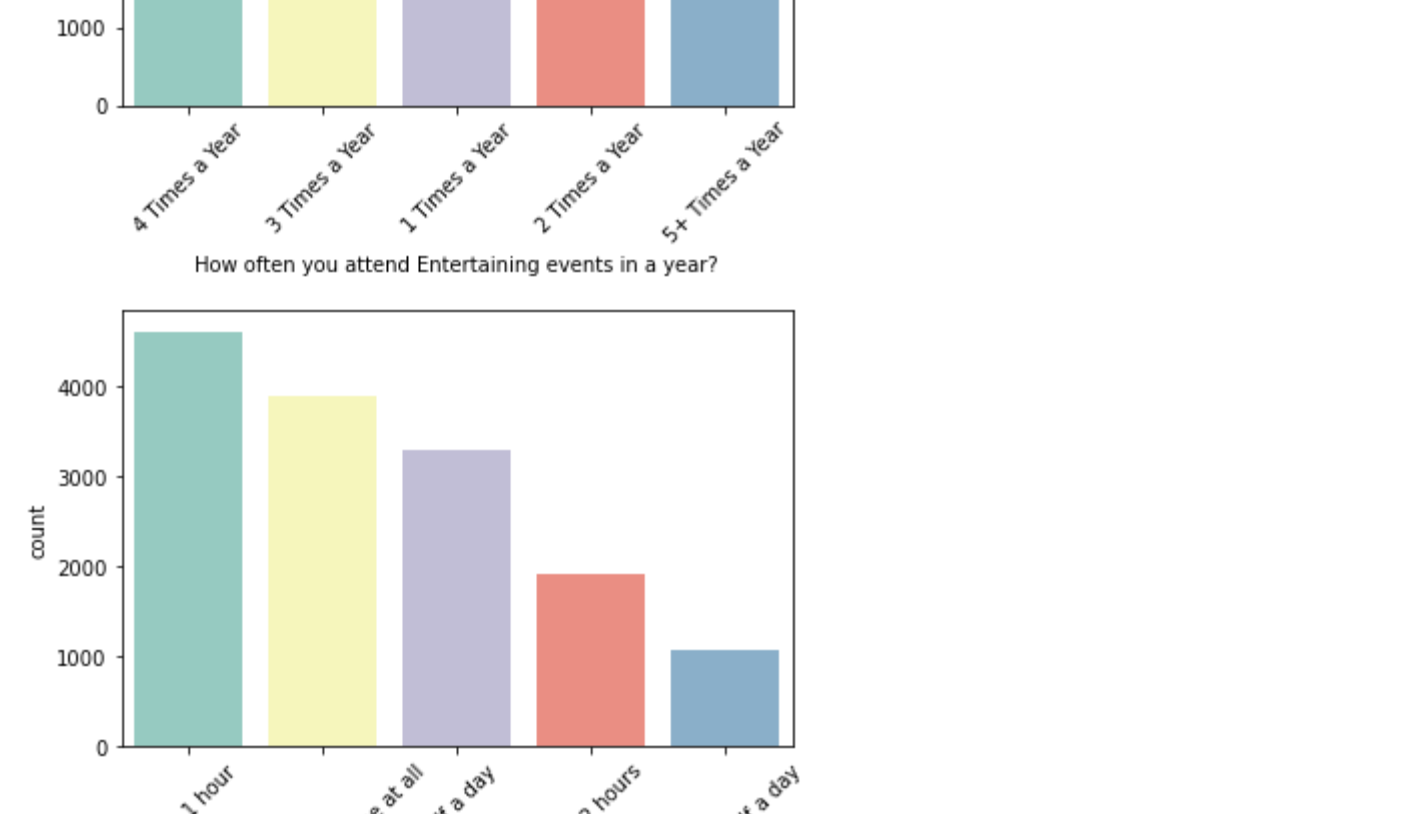


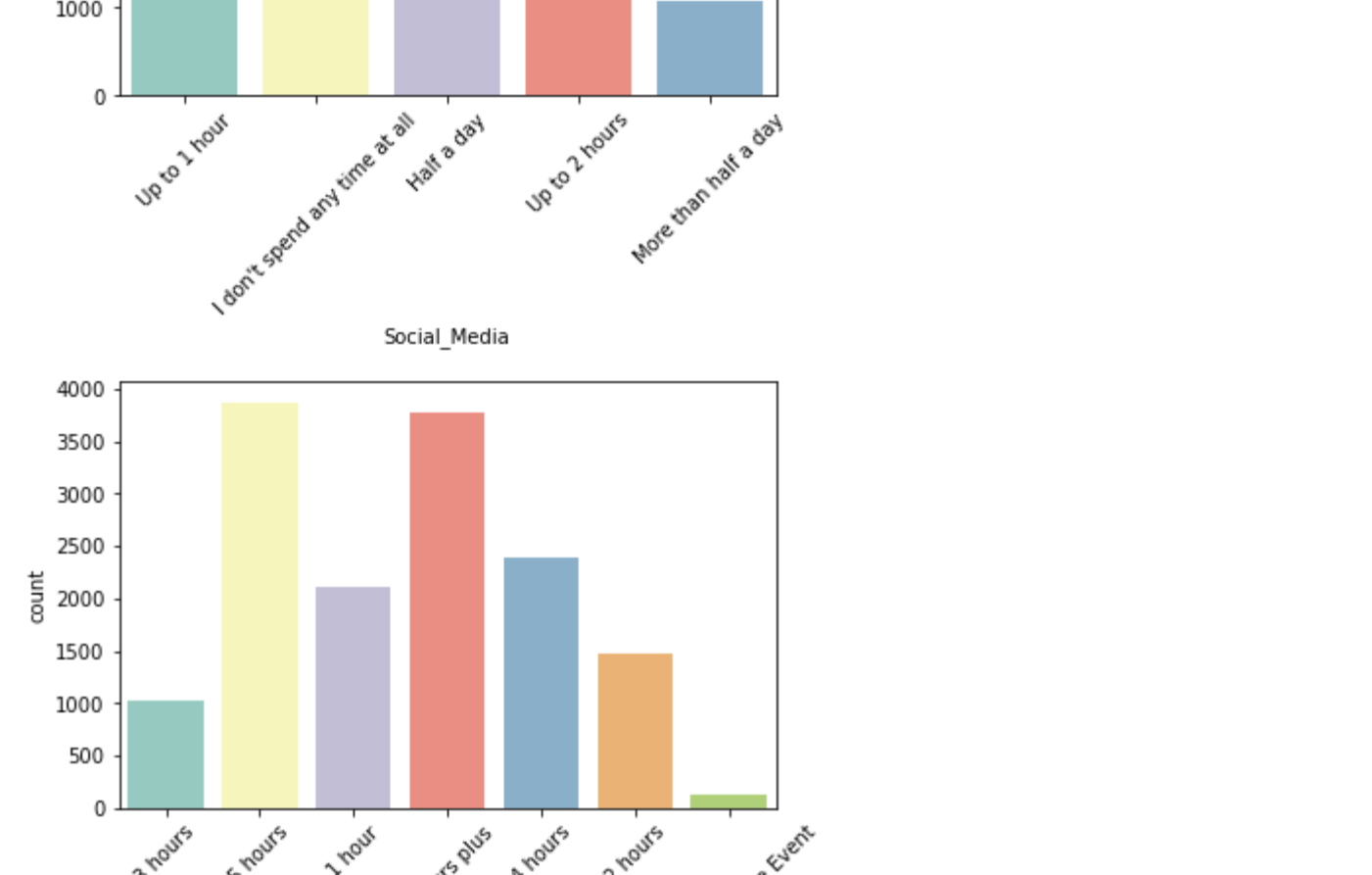


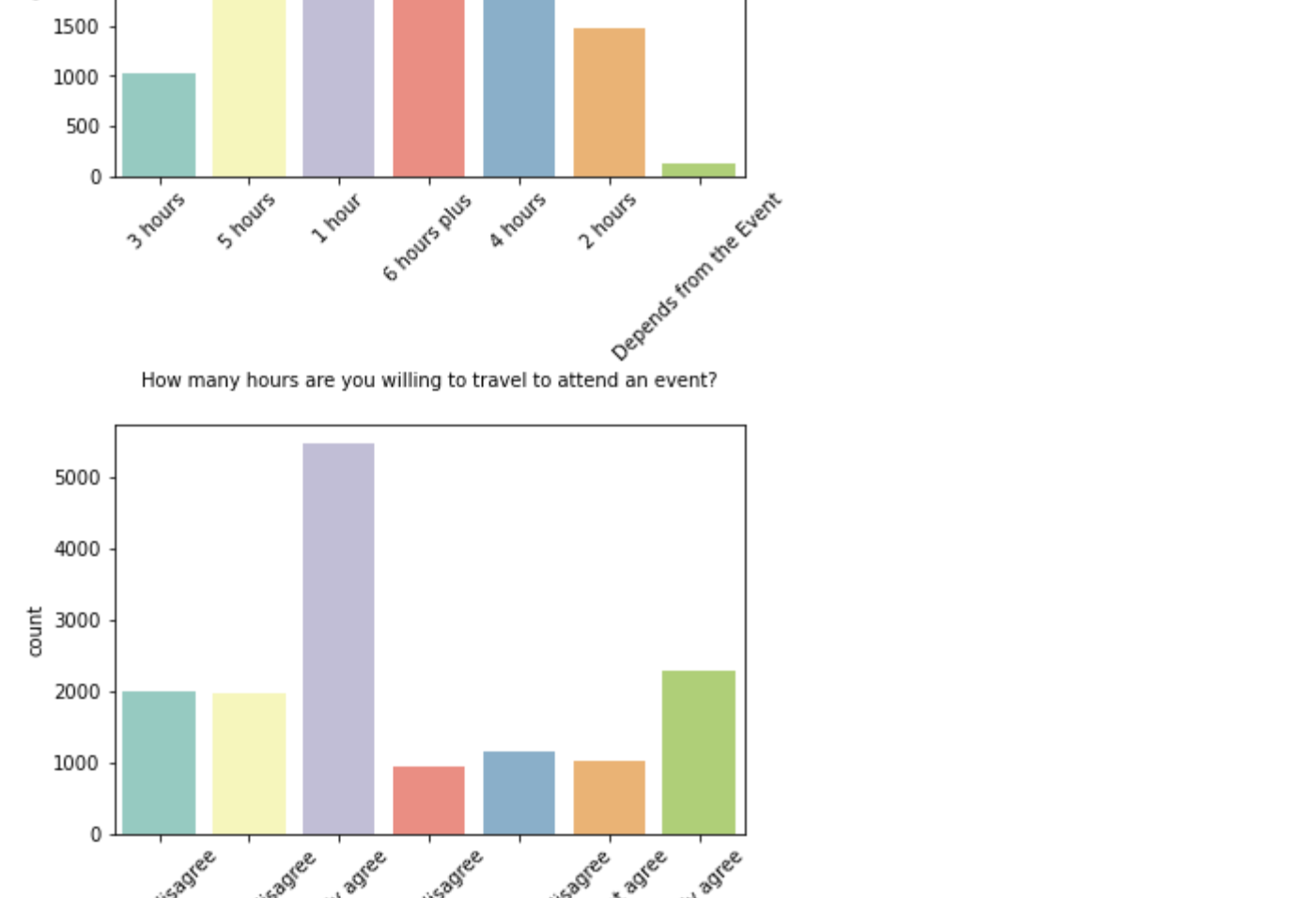


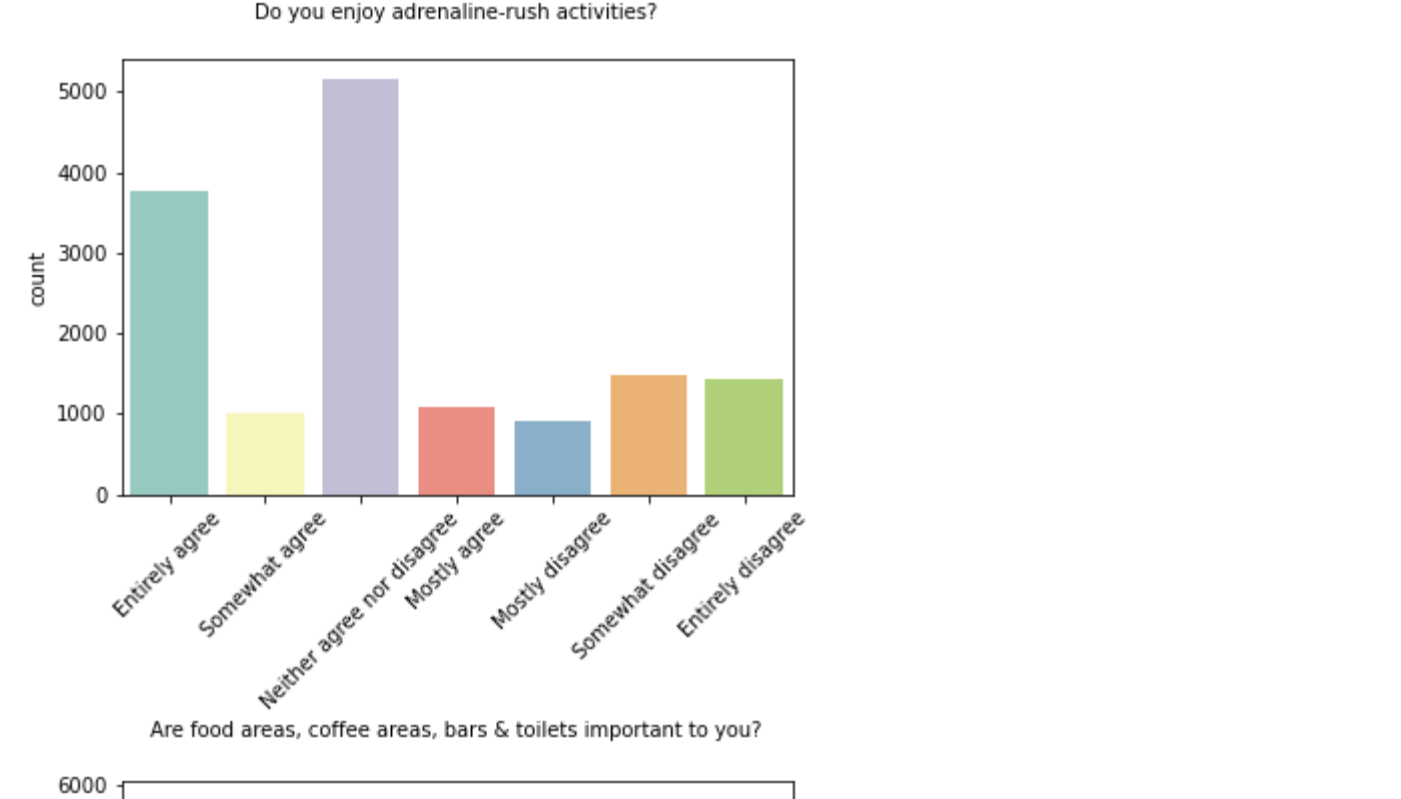


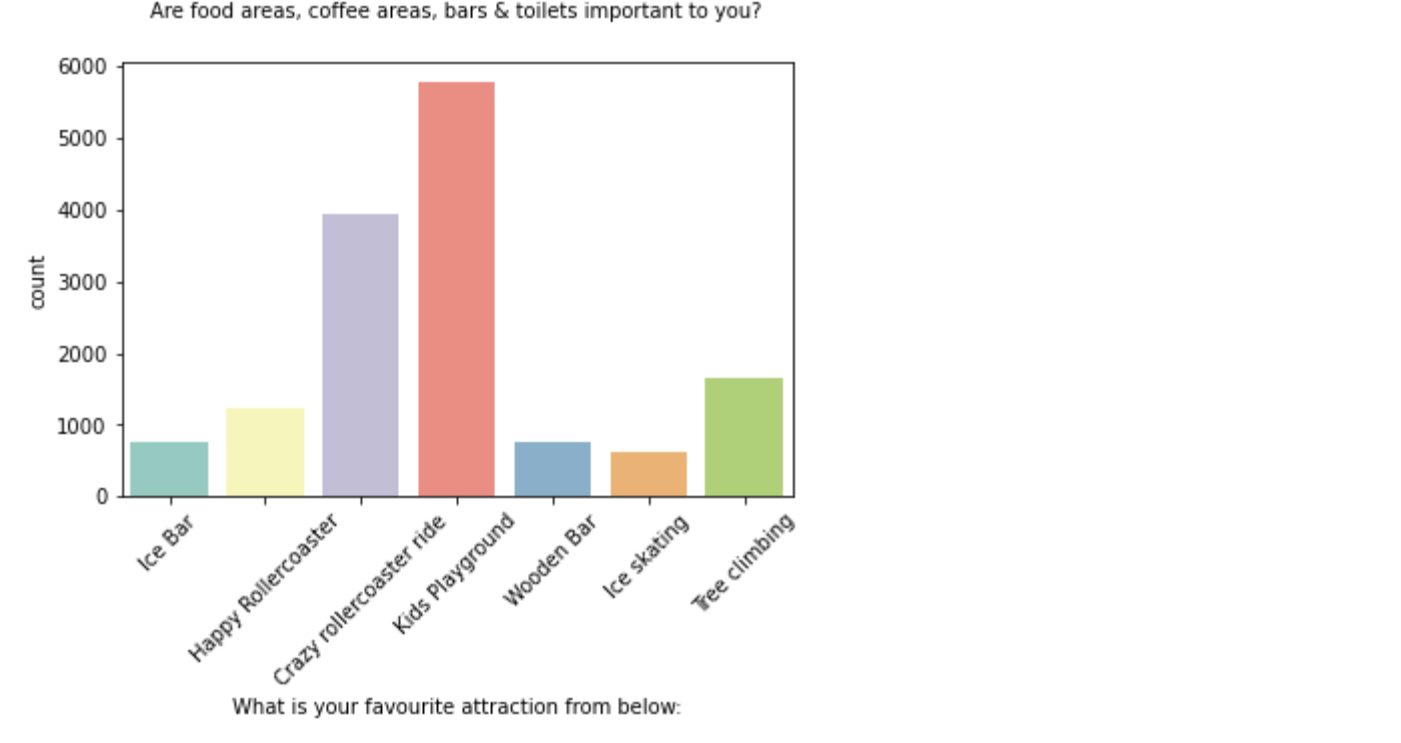


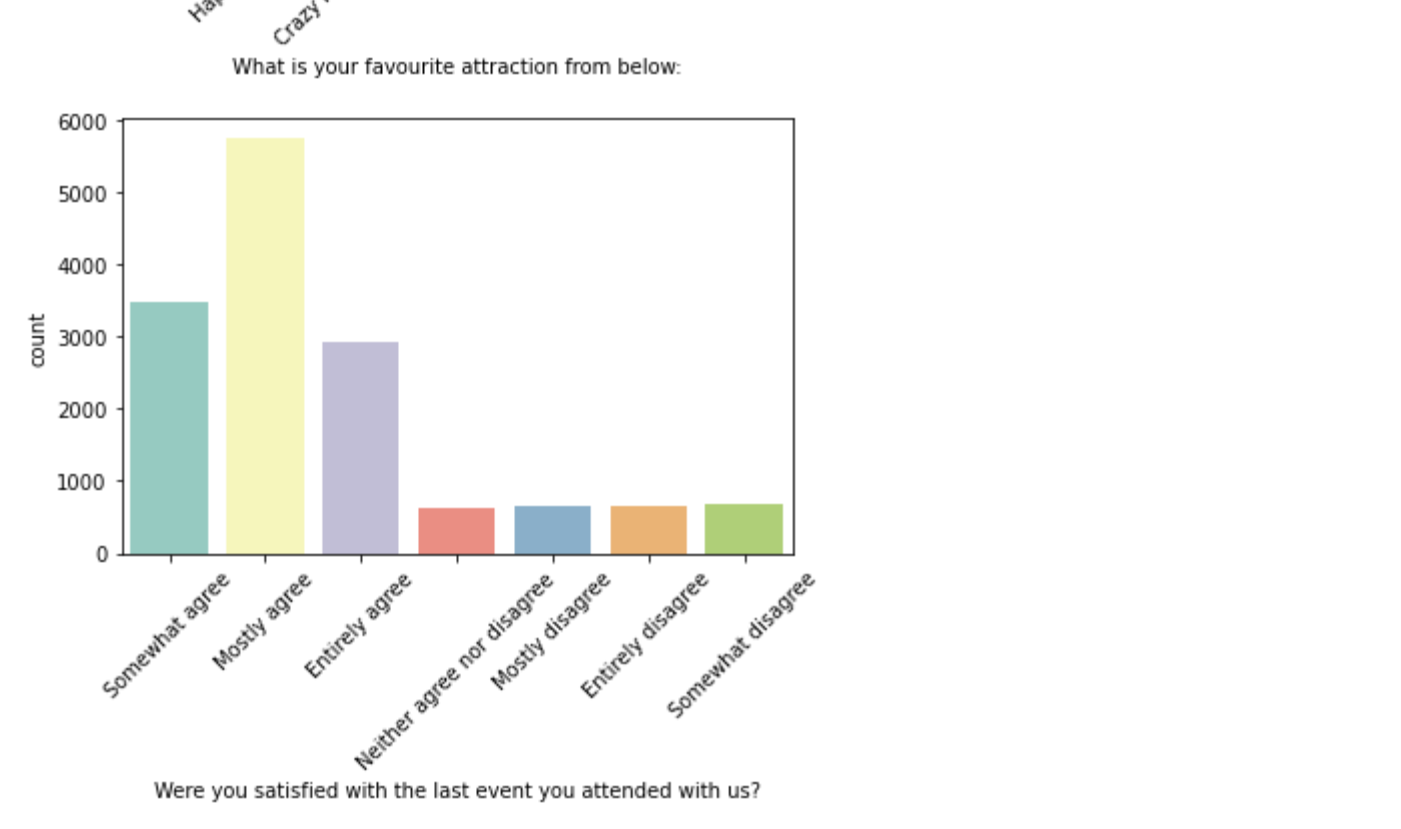


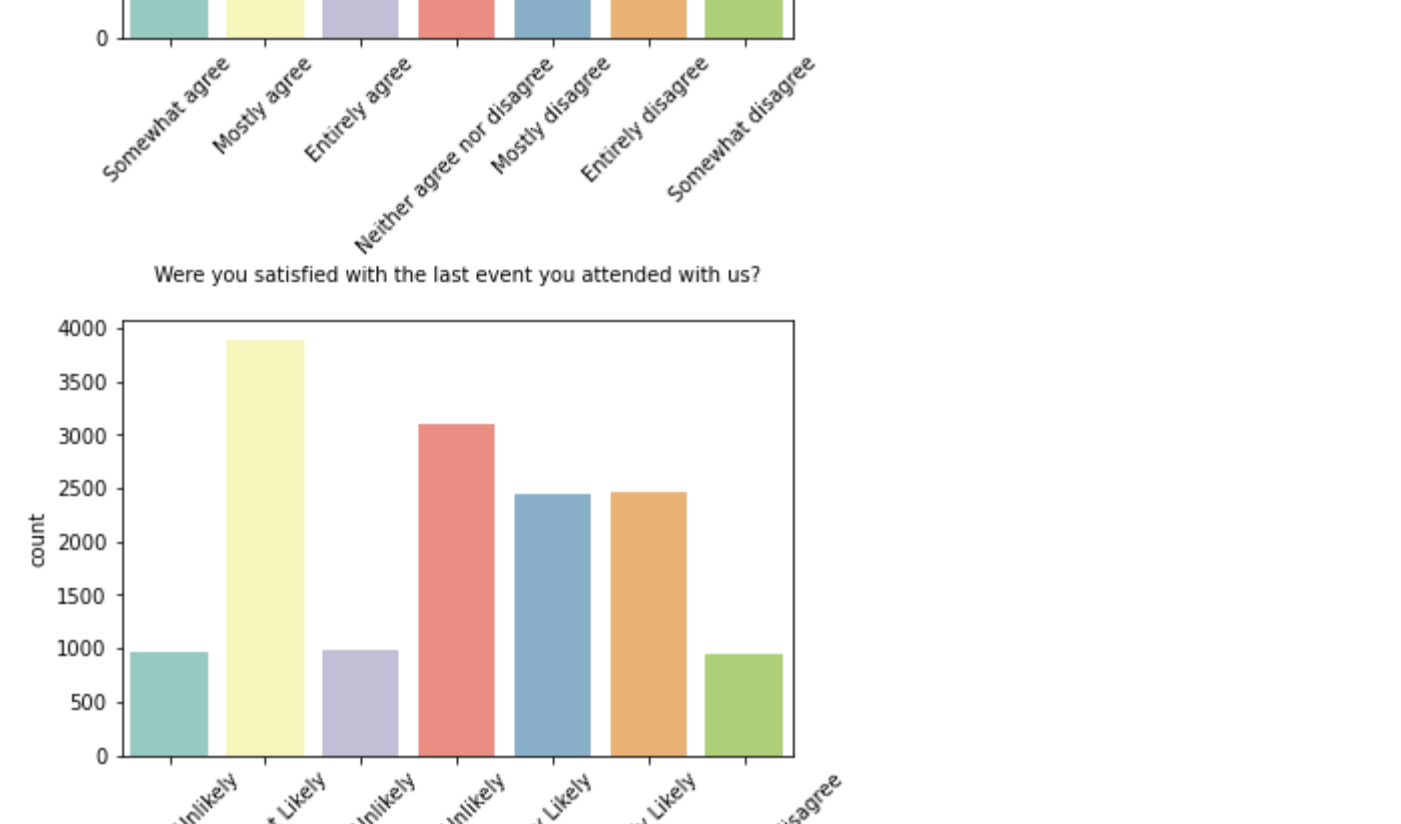












In [10]:
```python
# giving a numeric representation to categorical variables
print(df.shape)
raw_data = df[features]
print(raw_data.shape)
# Making categorical variables into numeric representation
new_raw_data = pd.get_dummies(raw_data, columns = features)

# Notes:
# We can also do this with Label Encoding and OneHotEncoder from the preprocessing library

print(new_raw_data.shape)
# print the shape

new_raw_data.head()
```

```
(14778, 19)
(14778, 13)
(14778, 86)
```

Out[10]:

	Age_17 or younger	Age_18-20	Age_21-25	Age_26-32	Age_33-39	Age_40-49	Age_50-59	Age_60-64	Age_65 or older	Gender_Female	...	Would you recommend our events to other people?_Somewhat Unlikely	Would you recommend our events to other people?_Very Likely
0	0	0	0	0	0	1	0	0	0	1	...	1	0
1	0	0	0	0	0	0	0	1	0	0	...	0	0
2	0	0	0	0	0	0	1	0	0	0	...	0	0
3	0	0	0	0	0	0	1	0	0	1	...	0	0
4	0	0	0	0	0	0	0	1	0	0	...	0	0

5 rows × 86 columns

In [11]:
```python
raw_data.head()
```

Out[11]:

	Age	Gender	Current_Status	Total_Household_Income	How often you attend Entertaining events in a year?	Social_Media	How many hours are you willing to travel to attend an event?	Do you enjoy adrenaline-rush activities?	Are food areas, coffee areas, bars & toilets important to you?	What is your favourite attraction from below:	Were you satisfied with the last event you attended with us?
0	40-49	Female	Married with children	50,000 to 74,999	4 Times a Year	Up to 1 hour	3 hours	Entirely disagree	Entirely agree	Ice Bar	Somewhat agree
1	60-64	Male	Married with children	35,000 to 49,999	3 Times a Year	Up to 1 hour	5 hours	Somewhat disagree	Entirely agree	Happy Rollercoaster	Mostly agree
2	50-59	Male	Married with children	75,000 to 99,999	3 Times a Year	I don't spend any time at all	5 hours	Somewhat disagree	Somewhat agree	Happy Rollercoaster	Mostly agree
3	50-59	Female	Married without children	35,000 to 49,999	1 Times a Year	Half a day	1 hour	Entirely agree	Neither agree nor disagree	Crazy rollercoaster ride	Somewhat agree
4	60-64	Male	Married with children	100,000 to 149,999	3 Times a Year	Up to 1 hour	6 hours plus	Mostly disagree	Neither agree nor disagree	Kids Playground	Entirely agree

Feature Scaling

In [12]:
```python
#Let's check if any feature requires scaling
scale_vars = []
for f in new_raw_data.columns:
    if new_raw_data[f].max() > 1:
        scale_vars.append(f)
scale_vars
```

Out[12]: []

KMeans

In [13]:
```python
# We will start with 5 clusters

kmeans = KMeans(n_clusters=5, random_state=540)
kmeans = kmeans.fit(new_raw_data)

# Prints the clusters it assigned to each observation
print("The clusters are: ", kmeans.labels_)


# Prints the Inertia
print("The Inertia is: ", kmeans.inertia_)
```

```
The clusters are:  [3 3 3 ... 3 1 3]
The Inertia is:  100865.1789273626
```

In [14]:
```python
kmeans.labels_.shape
```

Out[14]: (14778,)

In [15]:
```python
kmeans.cluster_centers_
```

Out[15]:
```
array([[-2.27682456e-18,  1.82186235e-02,  1.24493927e-01,
         1.97368421e-01,  1.07287449e-01,  1.29554656e-01,
         1.81174089e-01,  8.40080972e-02,  1.57894737e-01,
         3.02631579e-01,  6.97368421e-01,  1.34615385e-01,
         2.26720648e-01, -1.22124533e-15,  1.95344130e-01,
         3.03643725e-02,  3.54251012e-01,  2.02429150e-03,
         5.66801619e-01,  2.02429150e-03,  3.03643725e-03,
         4.30161943e-01,  2.28983499e-16,  5.64777328e-01,
         1.11022302e-16,  7.49400542e-16,  3.40005801e-16,
        -4.99600361e-16,  9.98987854e-01,  1.01214575e-03,
         3.33066907e-16, -3.46944695e-16,  3.05311332e-16,
         2.02429150e-03,  9.94939271e-01,  2.02429150e-03,
         1.01214575e-03,  2.02429150e-03,  9.83805668e-01,
         3.33066907e-16,  3.03643725e-03,  5.06072874e-03,
         5.06072874e-03,  1.01214575e-03,  5.63765182e-01,
         2.22044605e-16,  4.36234818e-01,  1.94289029e-16,
         1.80411242e-16, -2.22044605e-16, -2.77555756e-17,
         0.00000000e+00,  1.21457490e-02, -2.08166817e-16,
        -1.66533454e-16,  9.86842105e-01,  1.01214575e-03,
         4.30211422e-16,  1.01214575e-03, -1.38777878e-17,
         3.53883589e-16,  2.02429150e-03,  1.33226763e-15,
         9.96963563e-01,  1.80411242e-16,  3.03643725e-03,
         2.02429150e-03,  1.01214575e-03,  3.03643725e-03,
        -4.85722573e-17,  9.90890688e-01,  2.56739074e-16,
         3.88578059e-16, -1.38777878e-17, -3.60822483e-16,
         9.76720648e-01, -1.66533454e-16,  1.01214575e-02,
         2.22672065e-02,  8.90688259e-02,  6.98380567e-02,
         8.19838057e-02,  8.50202429e-02,  8.09716599e-02,
         9.10931174e-02,  5.02024291e-01],
       [-2.71050543e-18,  3.29597460e-17, -1.17961196e-16,
         1.40271493e-02,  9.90950226e-02,  3.46153846e-01,
         3.43891403e-01,  9.32126697e-02,  1.03619910e-01,
         1.84615385e-01,  8.15384615e-01,  9.04977376e-02,
         6.33484163e-03,  9.86425339e-01,  5.88235294e-03,
        -1.90819582e-17, -2.08166817e-16,  4.52488688e-04,
        -9.02056208e-17,  5.34389140e-01,  2.11764706e-01,
         2.71493213e-03,  2.26244344e-03,  5.42986425e-03,
         8.14479638e-03,  1.08597285e-02,  4.52488688e-03,
        -5.55111512e-16,  2.71493213e-03,  9.94570136e-01,
         2.26244344e-03,  4.52488688e-04,  9.04977376e-04,
         5.18552036e-01,  5.55111512e-17,  4.77828054e-01,
         2.71493213e-03,  1.35746606e-03,  3.61990950e-03,
         3.61990950e-03,  1.43438914e-01,  4.34841629e-01,
         4.12217195e-01,  9.04977376e-04,  1.35746606e-03,
         5.08144796e-01,  9.04977376e-04,  5.42986425e-03,
         7.23981900e-03,  4.52488688e-04,  4.76470588e-01,
         9.92307692e-01,  4.52488688e-04,  9.04977376e-04,
         9.04977376e-04,  2.26244344e-03,  2.71493213e-03,
         4.52488688e-04,  4.52488688e-04,  2.26244344e-03,
         9.04977376e-04, -1.80411242e-16,  9.95475113e-01,
        -2.35922393e-16,  9.04977376e-04,  9.79185520e-01,
         2.26244344e-03,  9.50226244e-03,  1.80995475e-03,
         4.97737557e-03,  2.26244344e-03,  2.77555756e-16,
         2.26244344e-03,  9.04977376e-04,  3.61990950e-03,
         2.71493213e-03,  3.61990950e-03,  2.71493213e-03,
         9.84162896e-01,  9.80995475e-01,  2.26244344e-03,
         6.33484163e-03,  1.35746606e-03,  4.07239819e-03,
         2.26244344e-03,  2.71493213e-03],
       [ 1.63132137e-03,  4.81239804e-02,  1.47634584e-01,
         1.30505710e-01,  8.89070147e-02,  1.59053834e-01,
         2.25938010e-01,  6.85154976e-02,  1.29690049e-01,
         4.37194127e-01,  5.62805873e-01,  2.61011419e-01,
         3.26264274e-03,  2.44698206e-03,  8.15660685e-04,
         5.79119086e-02,  5.69331158e-01,  5.70962480e-03,
         9.95106036e-02, -4.71844785e-16,  1.63132137e-03,
        -3.05311332e-16, -6.93889390e-17, -2.49800181e-16,
         4.46166395e-01,  2.35725938e-01,  3.16476346e-01,
         8.15660685e-04,  4.07830343e-03,  8.15660685e-04,
         8.15660685e-04,  9.93474715e-01,  9.91843393e-01,
         8.15660685e-04,  8.15660685e-04,  2.44698206e-03,
         4.07830343e-03, -3.05311332e-16, -2.49800181e-16,
         2.63677968e-16,  8.15660685e-04,  1.63132137e-03,
         9.97553018e-01,  2.94902991e-17,  9.99184339e-01,
         2.49800181e-16,  8.15660685e-04,  0.00000000e+00,
         1.38777878e-17, -1.66533454e-16, -5.55111512e-17,
         5.55111512e-16,  2.44698206e-03,  1.63132137e-03,
         1.63132137e-03,  5.70146819e-01,  8.15660685e-04,
         4.23327896e-01,  6.28058728e-01,  8.15660685e-04,
         8.15660685e-04,  9.46166395e-02, -6.66133815e-16,
         2.75693312e-01, -1.11022302e-16, -6.93889390e-16,
         7.63278329e-17,  9.97553018e-01,  8.15660685e-04,
         8.15660685e-04,  0.00000000e+00,  8.15660685e-04,
         9.97553018e-01, -2.77555756e-17, -9.71445147e-17,
         4.99600361e-16, -9.71445147e-17,  2.44698206e-03,
        -3.05311332e-16, -5.27355937e-16,  1.38777878e-16,
         8.15660685e-04,  9.97553018e-01,  8.15660685e-04,
         8.32667268e-17,  8.15660685e-04],
       [ 8.24596537e-04,  2.00259159e-03,  1.55495347e-02,
         7.91612675e-02,  1.12616327e-01,  2.11685711e-01,
         2.56567322e-01,  1.25809872e-01,  1.95782778e-01,
         2.49499352e-01,  7.50500648e-01,  1.08375545e-02,
         1.68099894e-01,  6.14913417e-01,  1.80822241e-01,
         3.41618565e-03,  1.76699258e-02,  2.35599010e-04,
         4.00518318e-03,  8.45800448e-02,  2.63870892e-02,
         8.56402403e-02,  2.56802921e-02,  1.35469431e-01,
         3.57757097e-01,  2.39604194e-01,  4.48816115e-02,
         1.26045471e-02,  1.09789139e-01,  3.71539640e-01,
         4.82388974e-01,  2.36777006e-02,  2.83896808e-02,
         3.22888444e-01,  7.30356933e-03,  4.19248439e-01,
         2.22169867e-01,  2.85074803e-02,  5.73683591e-02,
         1.20508894e-01,  2.43255978e-01,  3.42207563e-01,
         1.93191189e-01,  1.49605372e-02,  2.12981505e-01,
         1.02485570e-01,  2.17340087e-01,  1.10142537e-01,
         1.32288844e-01,  1.18035104e-01,  1.06726352e-01,
         1.84356226e-01,  1.65390505e-01,  1.25456473e-01,
         1.05430557e-01,  1.88597008e-01,  1.18741901e-01,
         1.12027329e-01,  1.66921899e-01,  1.42537401e-01,
         8.89386265e-02,  5.79573566e-02,  4.16774650e-01,
         3.75780422e-02,  8.92920250e-02,  8.95276240e-02,
         7.70408764e-02,  5.30097774e-02,  7.75120744e-02,
         7.16220992e-02,  7.50382848e-02,  7.91612675e-02,
         1.45364589e-01,  1.15207916e-01,  1.10495936e-01,
         1.22864884e-01,  1.12969726e-01,  2.87312993e-01,
         1.05783956e-01,  1.44186594e-01,  1.25456473e-01,
         2.10389916e-01,  1.33349040e-01,  1.27341265e-01,
         1.33584639e-01,  1.25692072e-01],
       [ 5.36193029e-04,  1.34048257e-02,  1.27613941e-01,
         1.49597855e-01,  8.68632708e-02,  1.41018767e-01,
         2.06970509e-01,  9.54423592e-02,  1.78552279e-01,
         4.48257373e-01,  5.51742627e-01,  1.31367292e-01,
         2.04289544e-01, -4.66293670e-15,  2.19302949e-01,
         3.05630027e-02,  3.46916890e-01,  3.75335121e-03,
         6.38069705e-02,  5.36193029e-04,  1.66533454e-16,
         5.10455764e-01,  3.81639165e-16,  4.88471850e-01,
         5.36193029e-04,  7.77156117e-16,  9.02056208e-17,
         1.00000000e+00,  1.22124533e-15, -2.94209102e-15,
        -5.55111512e-16,  3.33066907e-16,  9.90348525e-01,
         1.44328993e-15,  6.97050938e-03,  1.07238606e-03,
         1.60857909e-03,  9.98927614e-01, -5.27355937e-16,
         2.91433544e-16,  5.36193029e-04, -1.66533454e-15,
         5.36193029e-04, -3.81639165e-17,  9.99463807e-01,
        -3.05311332e-16,  8.32667268e-17,  6.24500451e-16,
        -4.16333634e-17, -2.22044605e-16,  5.36193029e-04,
        -1.66533454e-15,  3.33066907e-16, -4.85722573e-16,
        -2.91433544e-16,  9.99463807e-01,  1.66533454e-16,
         5.36193029e-04,  9.39410188e-01,  1.71581769e-02,
         1.60857909e-03,  8.57908847e-03,  1.93029491e-02,
         1.23324397e-02,  1.60857909e-03,  3.05311332e-16,
         1.80411242e-16,  2.49800181e-15,  1.38777878e-17,
         1.59594560e-16,  1.00000000e+00,  1.11022302e-16,
         5.36193029e-04,  2.35922393e-16, -5.13478149e-16,
         9.99463807e-01, -2.08166817e-16, -2.22044605e-16,
        -2.49800181e-16, -8.32667268e-16, -4.16333634e-17,
         5.36193029e-04,  3.60822483e-16,  3.88578059e-16,
         8.32667268e-17,  9.99463807e-01]])
```

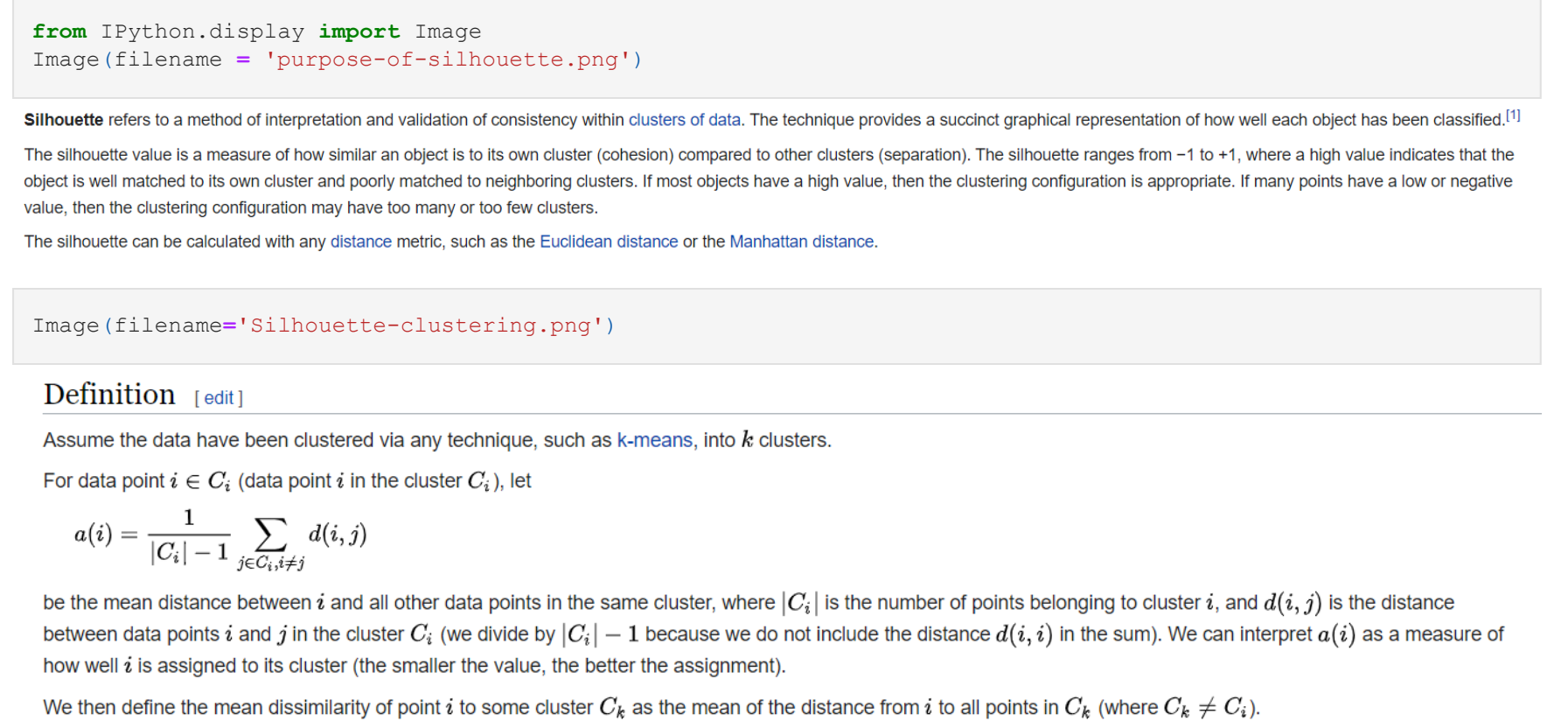
What is Inertia?

Inertia is the within-cluster sum of squares criterion. It is a metric that shows how internally coherent the clusters are.

In [16]:
```python
# How to find the best number of Ks? (Elbow Method)

# Running K means with multiple Ks

no_of_clusters = range(2,20) #[2,3,4,5,6,7,8,9, .... 19]
inertia = []


for f in no_of_clusters:
    kmeans = KMeans(n_clusters=f, random_state=540)
    kmeans = kmeans.fit(new_raw_data)
    u = kmeans.inertia_
    inertia.append(u)
    print("The inertia for :", f, "Clusters is:", u)
```

```
The inertia for : 2 Clusters is: 125619.02972065964
The inertia for : 3 Clusters is: 114905.39784583467
The inertia for : 4 Clusters is: 106337.18007454128
The inertia for : 5 Clusters is: 100865.1789273626
The inertia for : 6 Clusters is: 96432.68621313221
The inertia for : 7 Clusters is: 93814.49649581587
The inertia for : 8 Clusters is: 91864.70809764387
The inertia for : 9 Clusters is: 89725.57903775302
The inertia for : 10 Clusters is: 88540.69184845858
The inertia for : 11 Clusters is: 87662.6673397905
The inertia for : 12 Clusters is: 86632.59301274843
The inertia for : 13 Clusters is: 85755.03117123523
The inertia for : 14 Clusters is: 85026.48115685378
The inertia for : 15 Clusters is: 84355.95041422671
The inertia for : 16 Clusters is: 83688.67739466966
The inertia for : 17 Clusters is: 82952.6898529621
The inertia for : 18 Clusters is: 82284.27523573027
The inertia for : 19 Clusters is: 81797.65052685676
```

In [17]:
```python
plt.figure(figsize = (16, 6))
plt.xlabel('Number of clusters')
plt.ylabel('Inertia Score')
plt.title('SSE for different cluster size [The Elbow Method]')
plt.plot(no_of_clusters, inertia)
plt.grid(True)
plt.show()
```



Clustering Validation Technique (Silhouette)

https://en.wikipedia.org/wiki/Silhouette_(clustering)

https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html#sphx-glr-auto-examples-cluster-plot-kmeans-silhouette-analysis-py

In [18]:
```python
from IPython.display import Image
Image(filename = 'purpose-of-silhouette.png')
```

Out[18]:

Silhouette refers to a method of interpretation and validation of consistency within clusters of data. The technique provides a succinct graphical representation of how well each object has been classified.[1]

The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette ranges from −1 to +1, where a high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. If most objects have a high value, then the clustering configuration is appropriate. If many points have a low or negative value, then the clustering configuration may have too many or too few clusters.

The silhouette can be calculated with any distance metric, such as the Euclidean distance or the Manhattan distance.

In [19]:
```python
Image(filename='Silhouette-clustering.png')
```

Out[19]:

Definition [edit]

Assume the data have been clustered via any technique, such as k-means, into k clusters.

For data point $i \in C_i$ (data point i in the cluster C_i), let

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i,j)$$

be the mean distance between i and all other data points in the same cluster, where $|C_i|$ is the number of points belonging to cluster i, and $d(i,j)$ is the distance between data points i and j in the cluster C_i (we divide by $|C_i| - 1$ because we do not include the distance $d(i,i)$ in the sum). We can interpret $a(i)$ as a measure of how well i is assigned to its cluster (the smaller the value, the better the assignment).

We then define the mean dissimilarity of point i to some cluster C_k as the mean of the distance from i to all points in C_k (where $C_k \neq C_i$).

For each data point $i \in C_i$, we now define

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j)$$

to be the *smallest* (hence the $\min$ operator in the formula) mean distance of i to all points in any other cluster, of which i is not a member. The cluster with this smallest mean dissimilarity is said to be the "neighboring cluster" of i because it is the next best fit cluster for point i.

We now define a *silhouette* (value) of one data point i

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |C_i| > 1$$

and

$$s(i) = 0, \text{ if } |C_i| = 1$$

Which can be also written as:

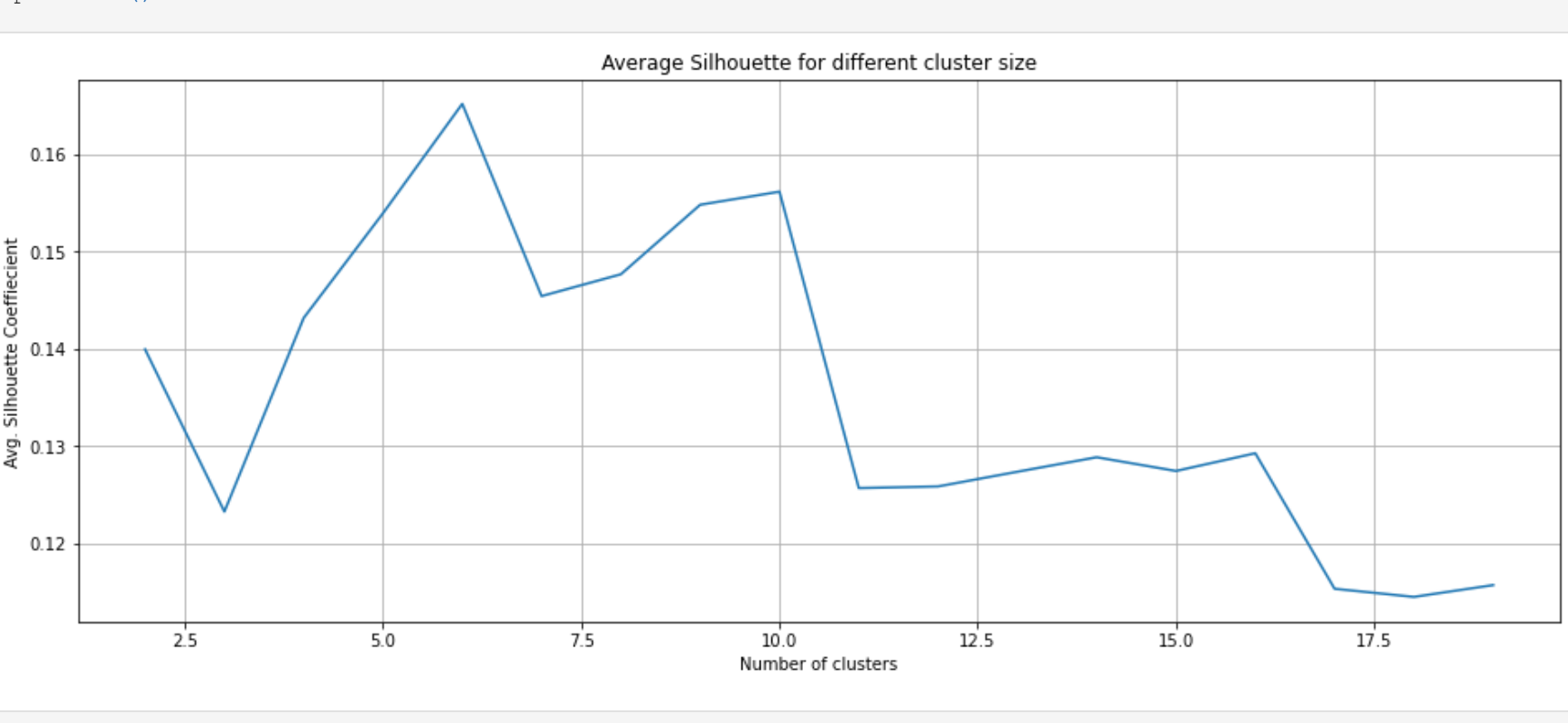
$$s(i) = \begin{cases} 1 - a(i)/b(i), & \text{if } a(i) < b(i) \\ 0, & \text{if } a(i) = b(i) \\ b(i)/a(i) - 1, & \text{if } a(i) > b(i) \end{cases}$$

In [20]:
```python
from sklearn.metrics import silhouette_samples, silhouette_score
avg_silh = []
for n_clusters in no_of_clusters:

    clusterer = KMeans(n_clusters=n_clusters, random_state=540)
    cluster_labels = clusterer.fit_predict(new_raw_data)

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(new_raw_data, cluster_labels)
    avg_silh.append(silhouette_avg)
    print("For n_clusters =", n_clusters,
          "The average silhouette_score is :", silhouette_avg)
```

```
For n_clusters = 2 The average silhouette_score is : 0.13992456890913335
For n_clusters = 3 The average silhouette_score is : 0.12324576035751086
For n_clusters = 4 The average silhouette_score is : 0.14315521902905532
For n_clusters = 5 The average silhouette_score is : 0.15392684972449343
For n_clusters = 6 The average silhouette_score is : 0.16516561674619268
For n_clusters = 7 The average silhouette_score is : 0.14540157050869345
For n_clusters = 8 The average silhouette_score is : 0.1476426951549887
For n_clusters = 9 The average silhouette_score is : 0.15479088794023727
For n_clusters = 10 The average silhouette_score is : 0.15613475214857367
For n_clusters = 11 The average silhouette_score is : 0.1256650017424242
For n_clusters = 12 The average silhouette_score is : 0.12583695890812968
For n_clusters = 13 The average silhouette_score is : 0.12734720267355448
For n_clusters = 14 The average silhouette_score is : 0.12883678582000692
For n_clusters = 15 The average silhouette_score is : 0.12742603200042713
For n_clusters = 16 The average silhouette_score is : 0.12924848853482543
For n_clusters = 17 The average silhouette_score is : 0.1153168121794705
For n_clusters = 18 The average silhouette_score is : 0.1144807741152854
For n_clusters = 19 The average silhouette_score is : 0.11569578562478706
```

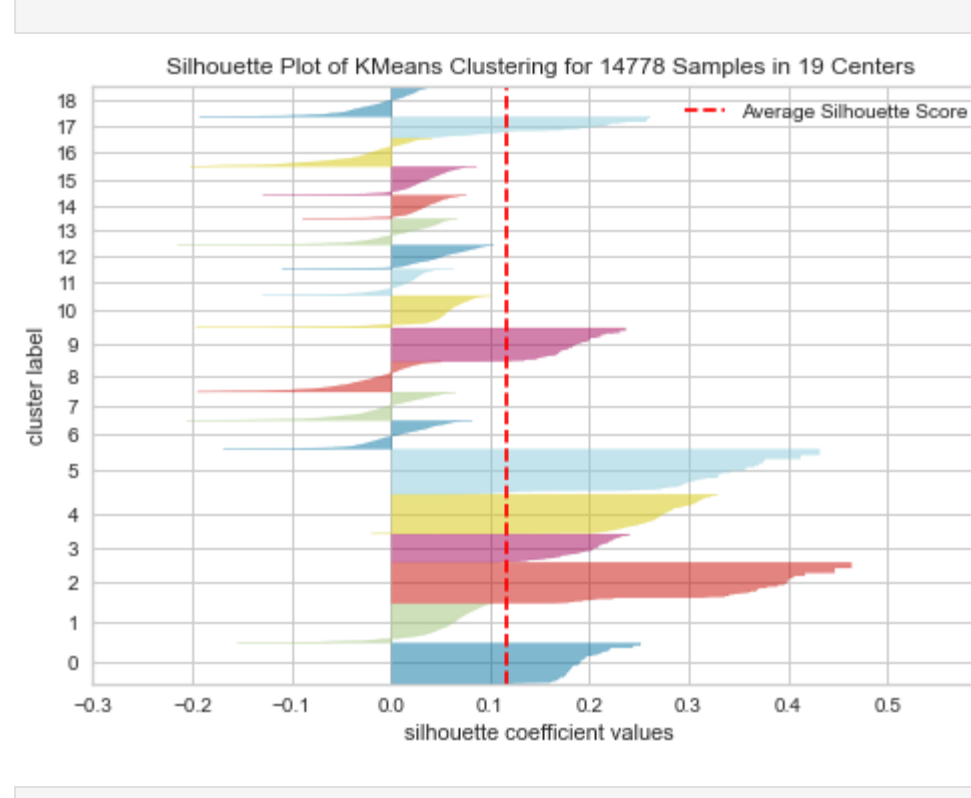
In [21]:

```
plt.figure(figsize = (16, 6))
plt.xlabel('Number of clusters')
plt.ylabel('Avg. Silhouette Coefficient')
plt.title('Average Silhouette for different cluster size')
plt.plot(no_of_clusters, avg_silh)
plt.grid()
plt.show()
```



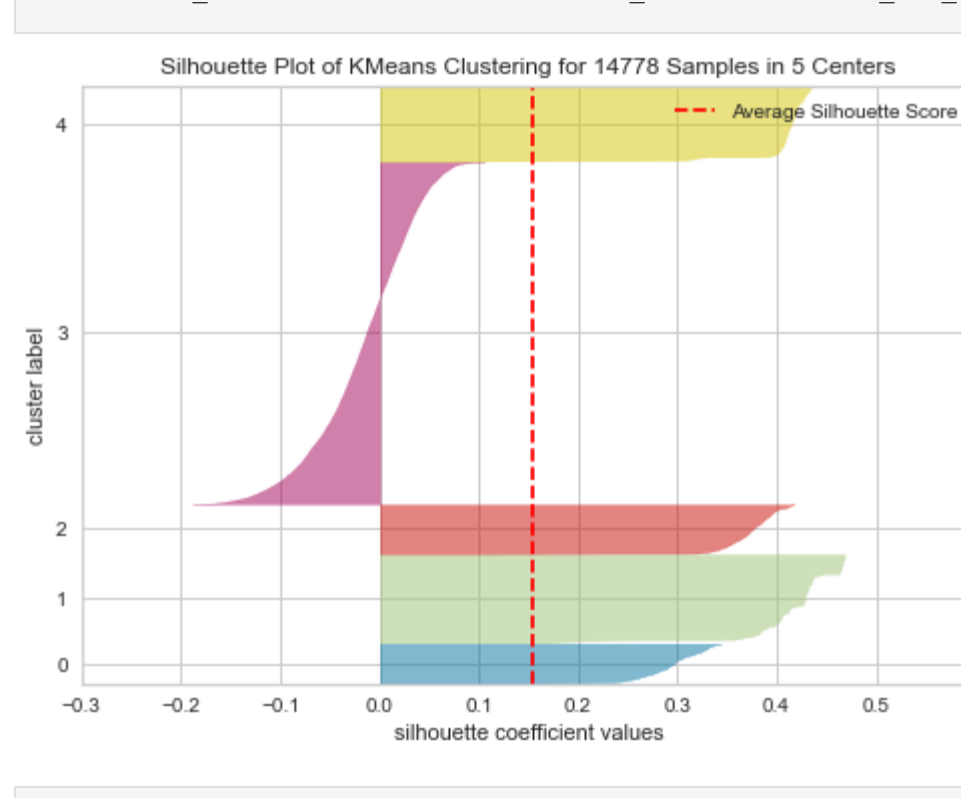
In [22]:

```
from yellowbrick.cluster import SilhouetteVisualizer
visualizer = SilhouetteVisualizer(clusterer, colors='yellowbrick')
visualizer.fit(new_raw_data)
visualizer.show();
```



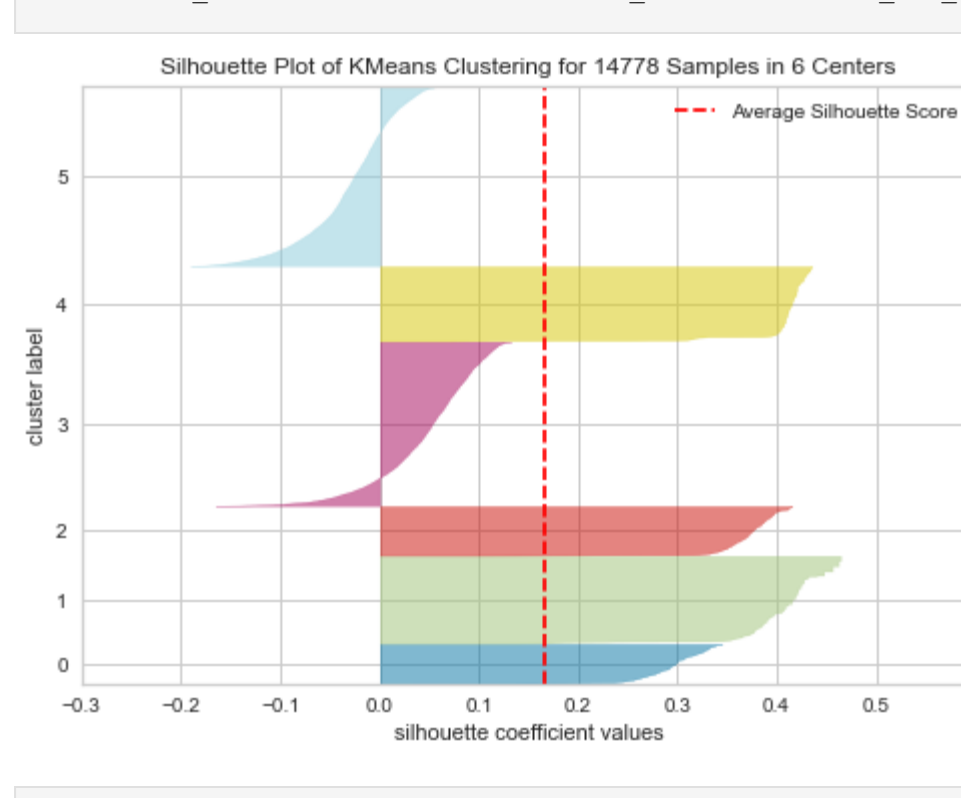
In [23]:

```
from yellowbrick.cluster import silhouette_visualizer
silhouette_visualizer(KMeans(5, random_state=540), new_raw_data, colors='yellowbrick');
```



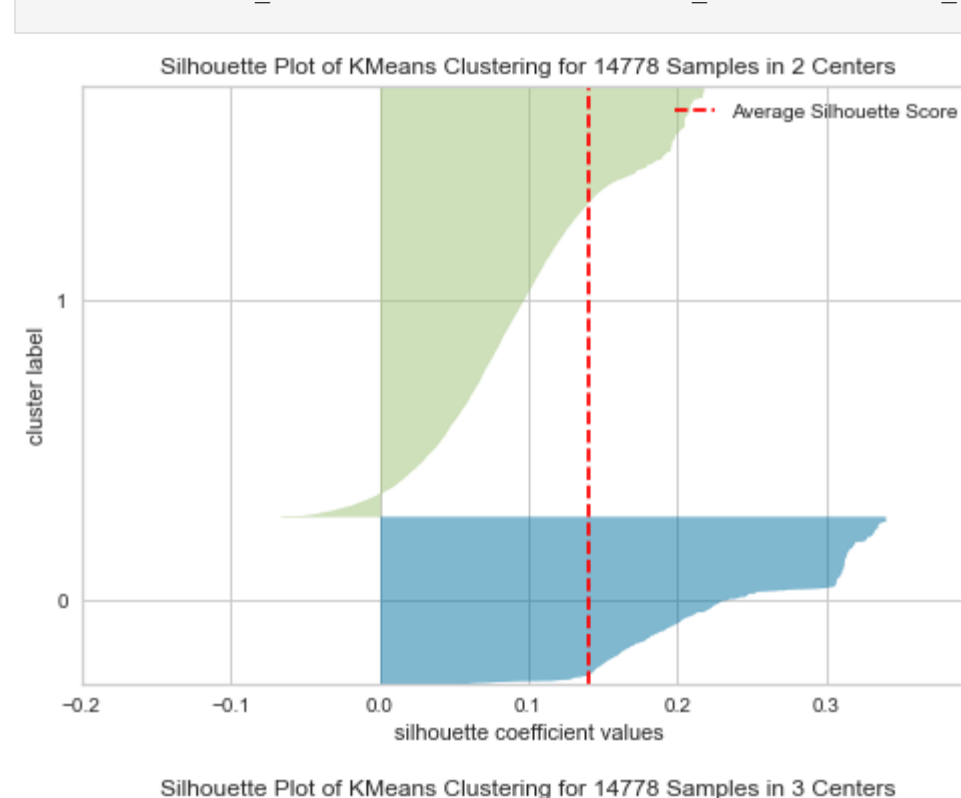
In [24]:

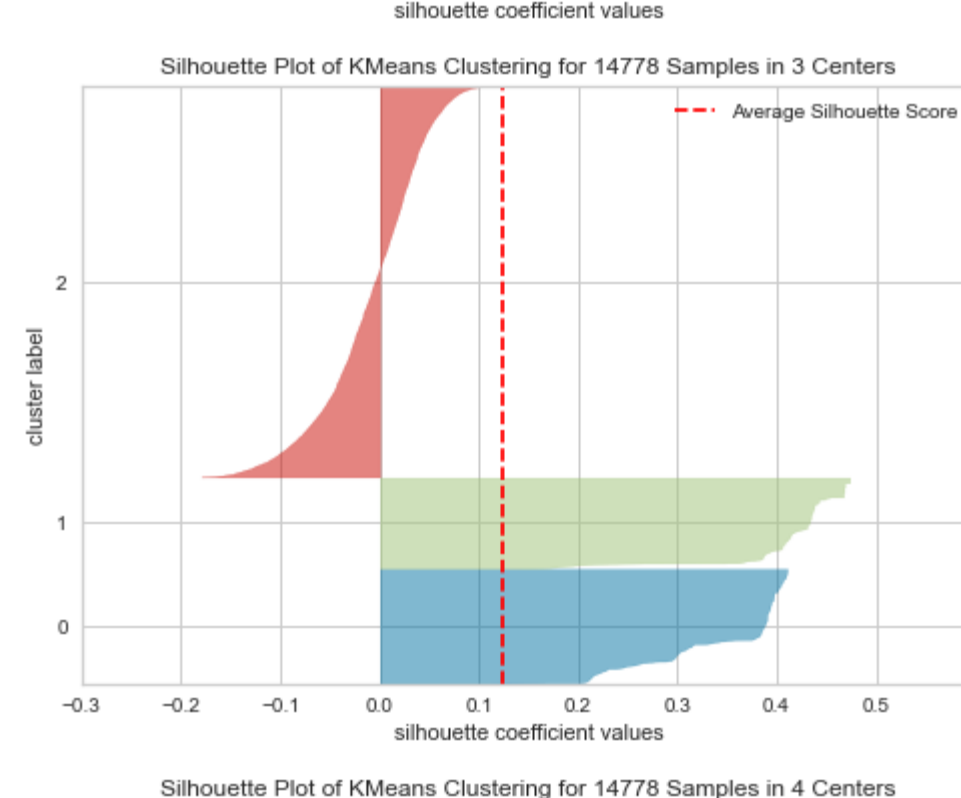
```
silhouette_visualizer(KMeans(6, random_state=540), new_raw_data, colors='yellowbrick');
```

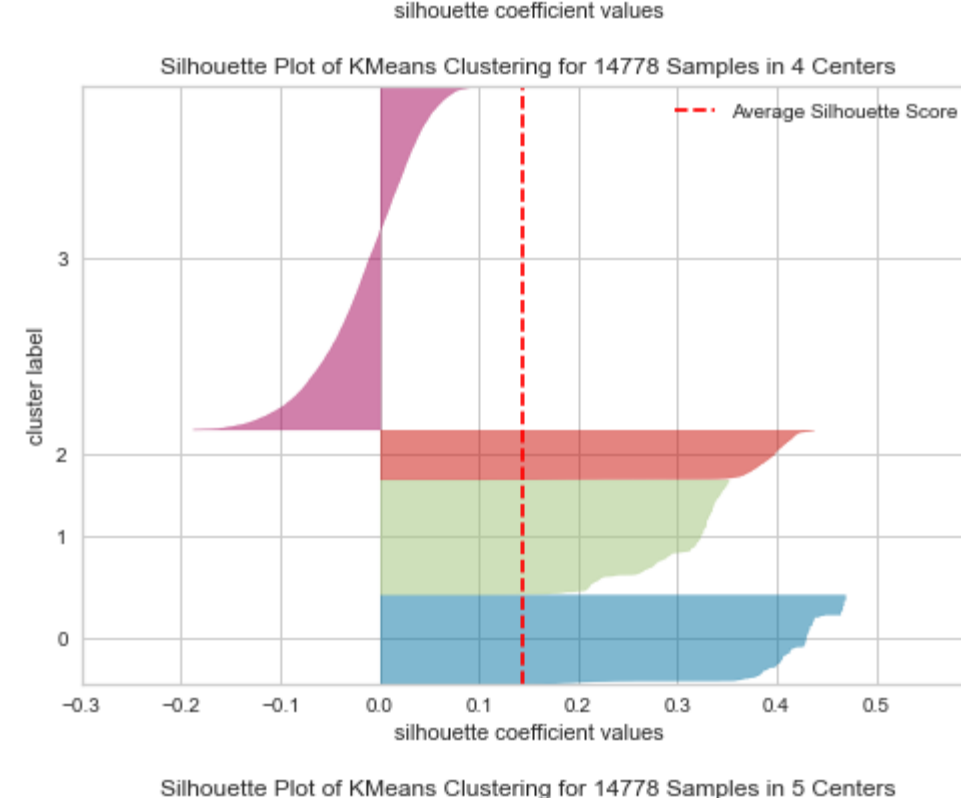


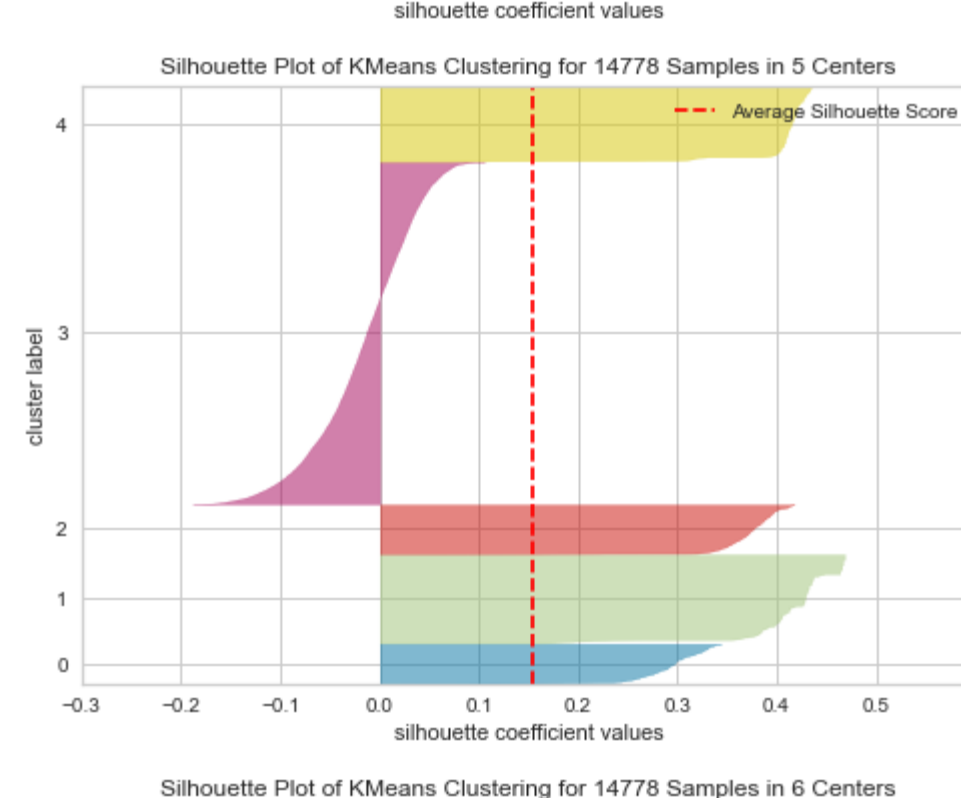
In [25]:

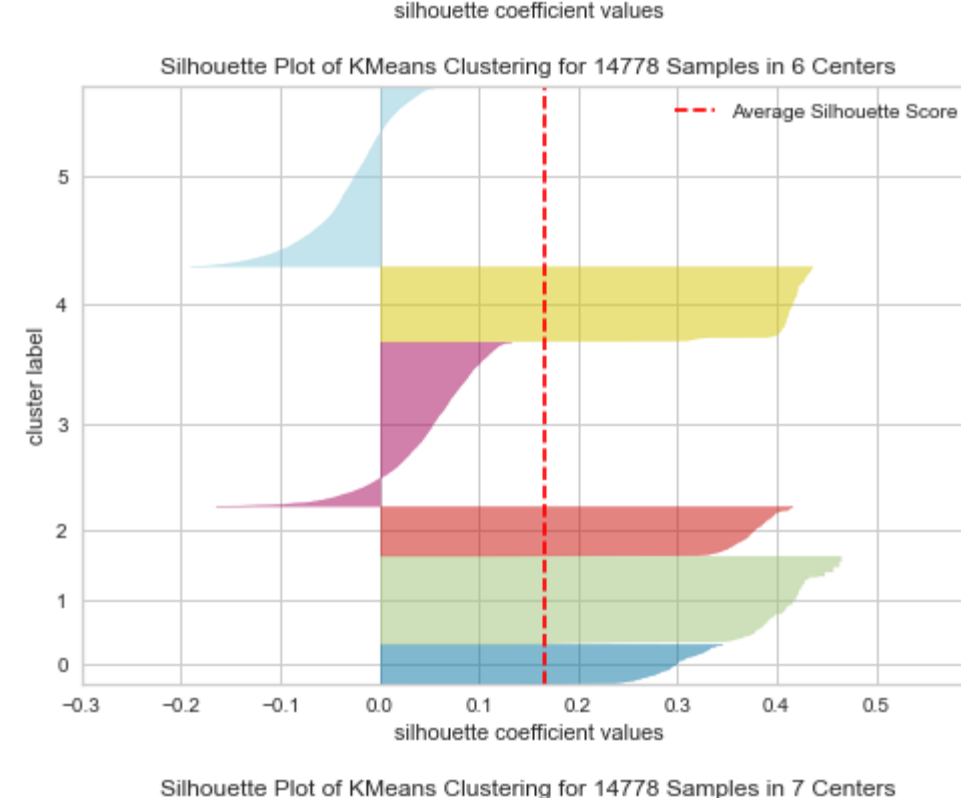
```
x = [2, 3, 4, 5, 6, 7]

for i in x:
    silhouette_visualizer(KMeans(i, random_state=540), new_raw_data, colors='yellowbrick');
```

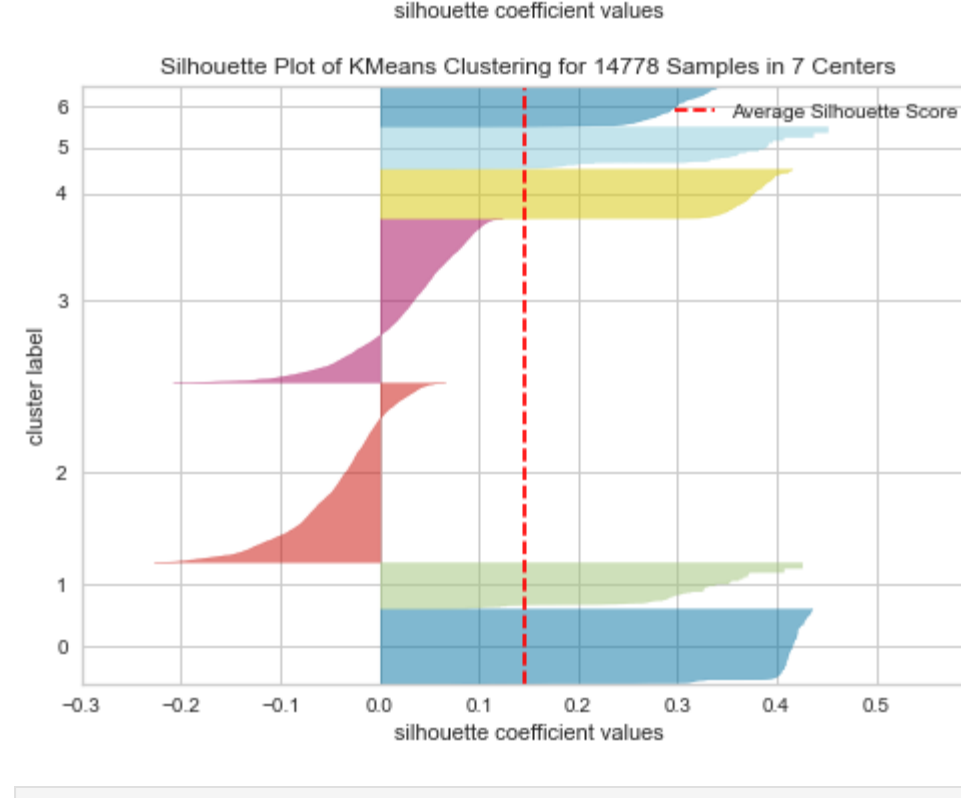












In [ ]: