In [1]:

```
# Python 3.7
# Analysing the data for usage of app and website for a perticular company
```

In [2]:

```
# Usual imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [3]:

```
%matplotlib inline
```

In [4]:

```
customers = pd.read_csv('Ecommerce Customers')
```

In [5]:

```
# Check out the info() method
customers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
Email                   500 non-null object
Address                 500 non-null object
Avatar                  500 non-null object
Avg. Session Length     500 non-null float64
Time on App             500 non-null float64
Time on Website         500 non-null float64
Length of Membership    500 non-null float64
Yearly Amount Spent     500 non-null float64
dtypes: float64(5), object(3)
memory usage: 31.3+ KB
```

In [6]:

```
# Let's replace the 'space' in the column names with an 'underscore'
customers.columns = customers.columns.str.replace(' ','_')
```

In [7]:

```
# One way to get the columns which are only float or int type are is to use describe()
 method
customers.describe()
```

Out[7]:

	Avg._Session_Length	Time_on_App	Time_on_Website	Length_of_Membership	Yearly_
count	500.000000	500.000000	500.000000	500.000000	
mean	33.053194	12.052488	37.060445	3.533462	
std	0.992563	0.994216	1.010489	0.999278	
min	29.532429	8.508152	33.913847	0.269901	
25%	32.341822	11.388153	36.349257	2.930450	
50%	33.082008	11.983231	37.069367	3.533975	
75%	33.711985	12.753850	37.716432	4.126502	
max	36.139662	15.126994	40.005182	6.922689	

In [8]:

```
# Use the column list from the dataframe derived out of describe() method
customers.describe().columns
```

Out[8]:

```
Index(['Avg._Session_Length', 'Time_on_App', 'Time_on_Website',
       'Length_of_Membership', 'Yearly_Amount_Spent'],
      dtype='object')
```

In [9]:

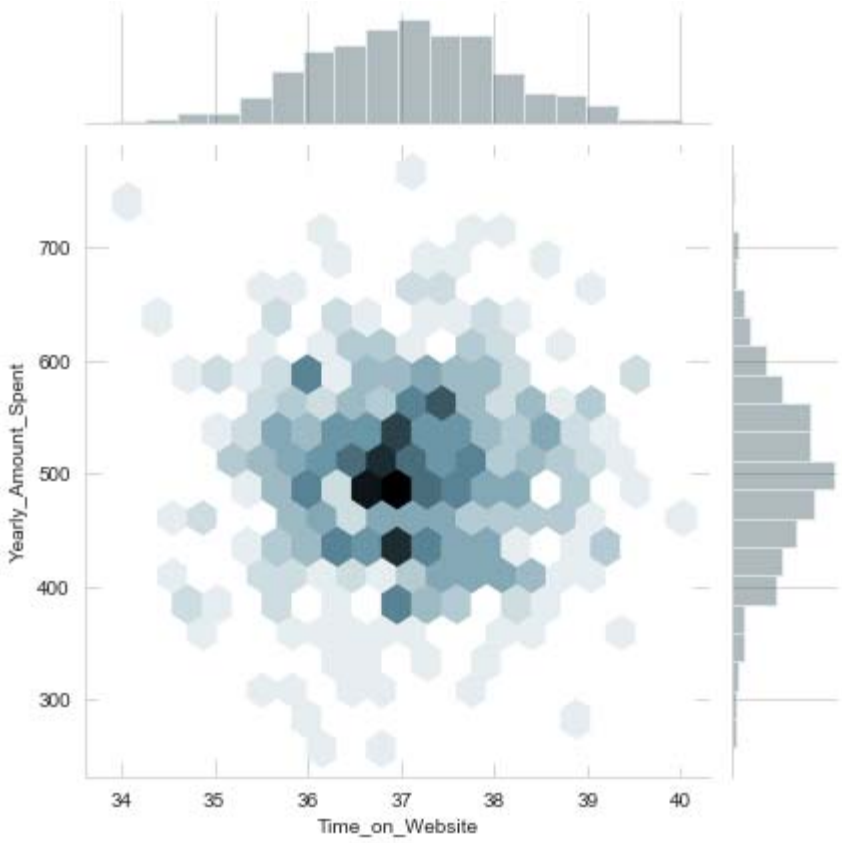
```
# Settings for a format in Seaborn
sns.set_palette("GnBu_d")
sns.set_style('whitegrid')
```

In [10]:

```
# Let's make a jointplot to visualize the correlation between 'Time on Website'
sns.jointplot(data=customers, x='Time_on_Website',y='Yearly_Amount_Spent',kind='hex' )
```

Out[10]:

```
<seaborn.axisgrid.JointGrid at 0x230a81ff358>
```

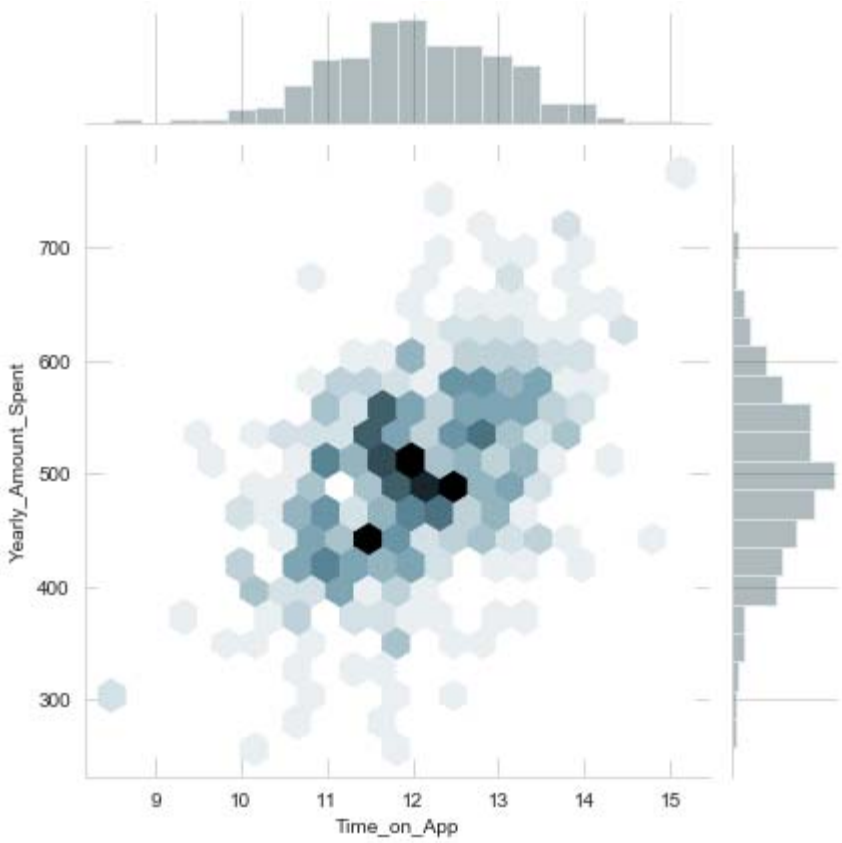


In [11]:

```
# Let's make a jointplot between 'Time_on_App' and 'Yearly_Amount_Spent'
sns.jointplot(data=customers, x='Time_on_App', y='Yearly_Amount_Spent', kind='hex',)
```

Out[11]:

```
<seaborn.axisgrid.JointGrid at 0x230a857b438>
```

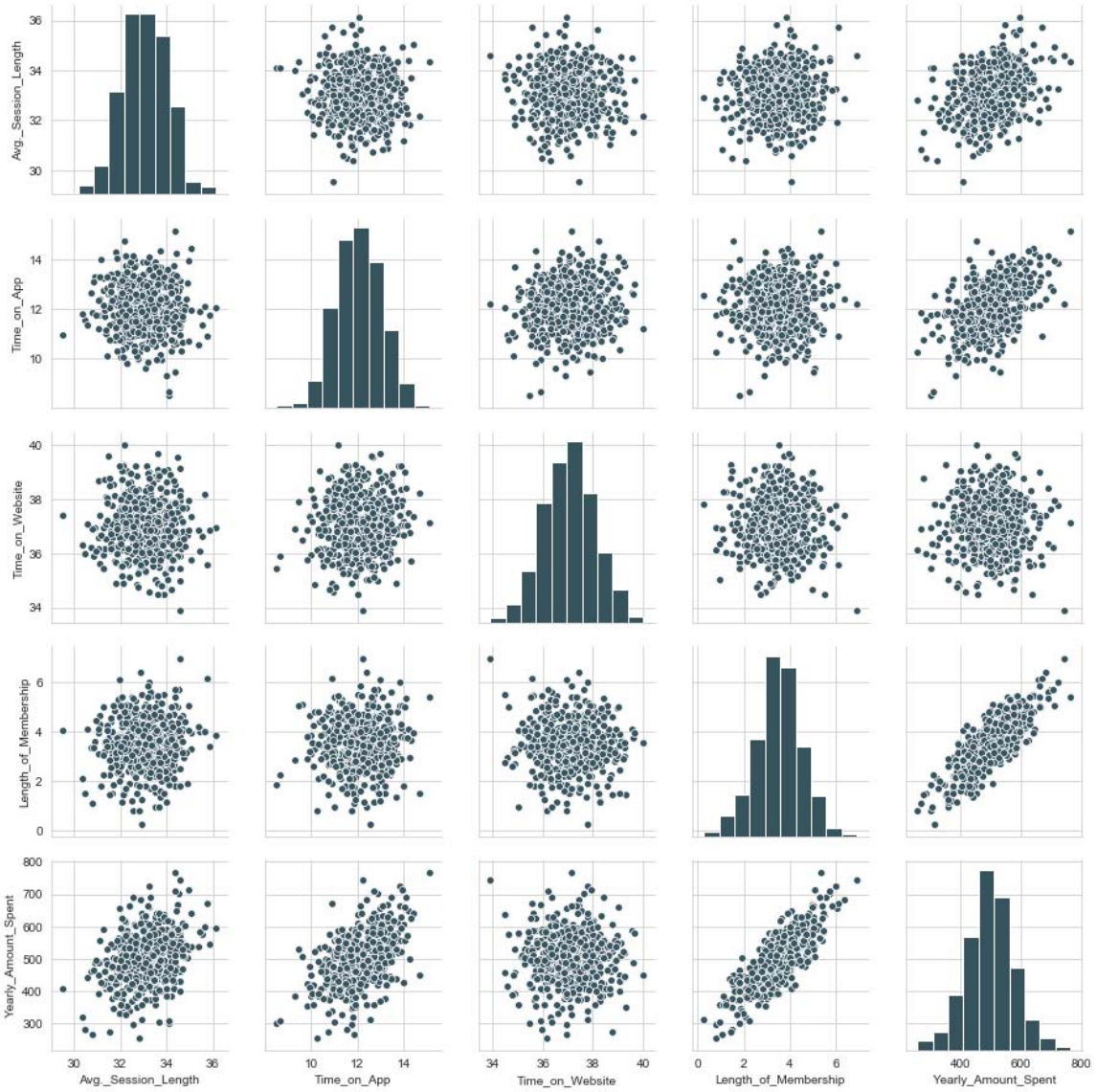


In [12]:

```
sns.pairplot(data=customers)
```

Out[12]:

```
<seaborn.axisgrid.PairGrid at 0x230a878ea90>
```

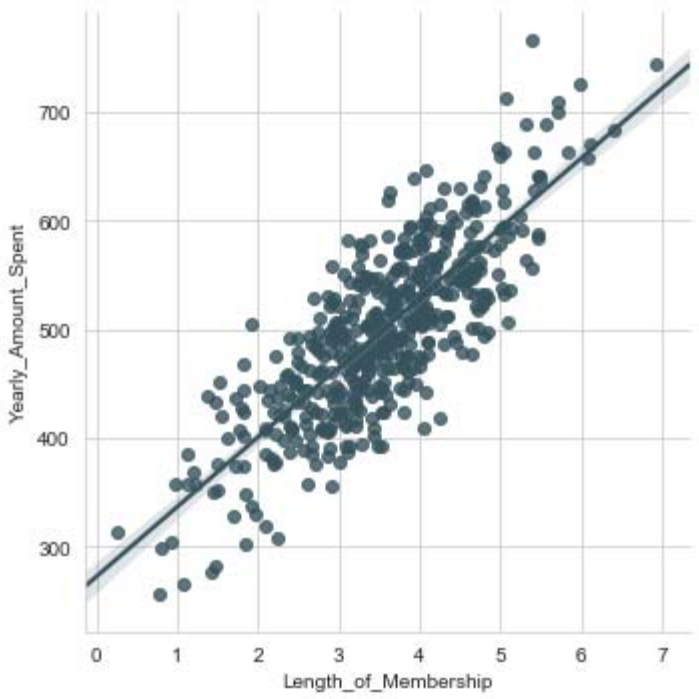


In [13]:

```
# From the above plot, clearly the length_of_membership is linearly correlated to 'Year
ly_amount_Spent'
# Let's explore this a bit
# Use seaborns linear model plot method (lmplot()) to correlate these two columns
sns.lmplot(data=customers, x='Length_of_Membership', y='Yearly_Amount_Spent')
```

Out[13]:

```
<seaborn.axisgrid.FacetGrid at 0x230a90b0128>
```



ML

In [14]:

```
# Let's get 'Yearly_Amount_Spent' as the target
y = customers['Yearly_Amount_Spent']
# Let's get other columns as variables affecting the target
X = customers[['Avg._Session_Length', 'Time_on_App', 'Time_on_Website',
       'Length_of_Membership']]
```

In [15]:

```
from sklearn.model_selection import train_test_split
```

In [16]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
01)
```

In [22]:

```
# Step to train the model on our data
from sklearn.linear_model import LinearRegression
# Instantiate the model
lm = LinearRegression()
# Train the model
lm.fit(X_train, y_train)
```

Out[22]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=F
alse)
```

In [23]:

```
# Print out the coefficients of the model
print('The intercept is: ', lm.intercept_)
print('The set of co-efficients is: ', lm.coef_)
```

```
The intercept is:  -1047.9327822502387
The set of co-efficients is:  [25.98154972 38.59015875  0.19040528 61.2790
9654]
```

Predicting the test data

In [24]:

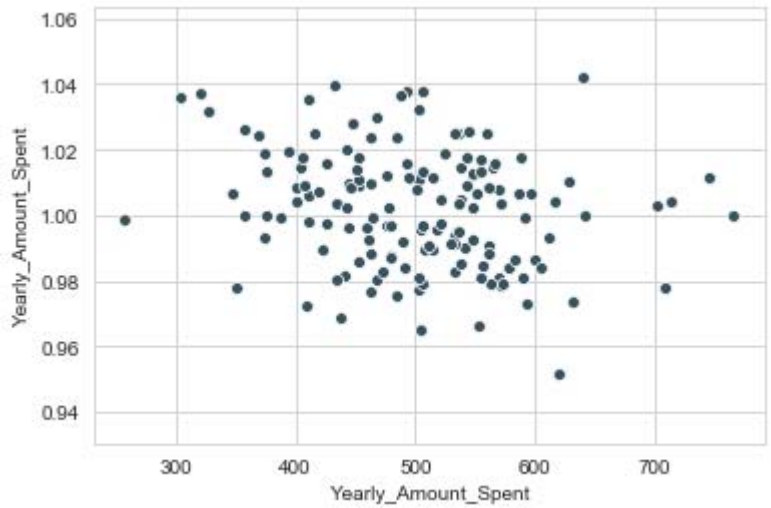
```
pred_values = lm.predict(X_test)
```

In [25]:

```
# Analysing the accuracy of the model
# method # 1
pred_ratio = pred_values/y_test
sns.scatterplot(x=y_test, y=pred_ratio)
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x230ab1588d0>
```

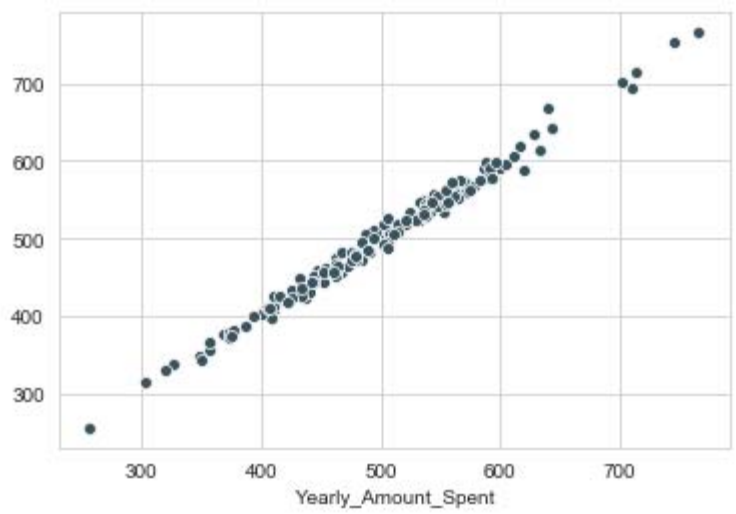


In [26]:

```
# Analysing the accuracy of the model
# method # 2
sns.scatterplot(x=y_test, y=pred_values)
```

Out[26]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x230ab1bc5c0>
```



Evaluating the model

In [27]:

```
from sklearn import metrics

print('Mean Absolute Error(MAE): ', metrics.mean_absolute_error(y_test, pred_values))
print('Mean Squared Error(MAE): ', metrics.mean_squared_error(y_test, pred_values))
print('Root Mean Squared Error(MAE): ', np.sqrt(metrics.mean_squared_error(y_test, pred
_values)))
```

```
Mean Absolute Error(MAE):  7.228148653430853
Mean Squared Error(MAE):  79.81305165097487
Root Mean Squared Error(MAE):  8.933815066978656
```

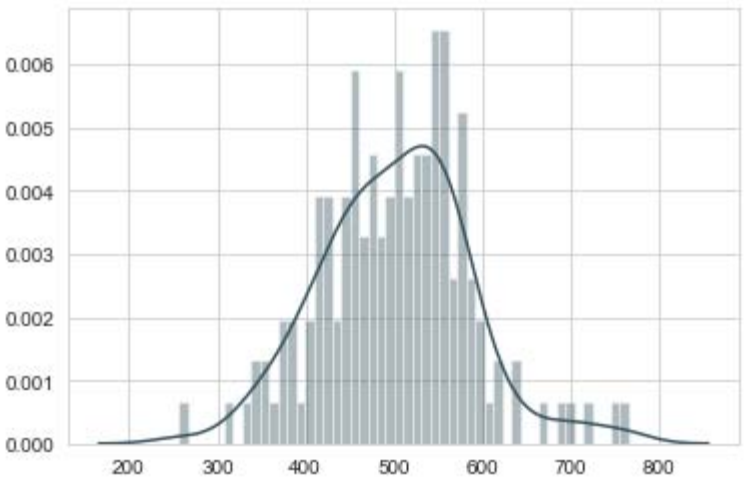
Residuals

In [28]:

```
sns.distplot(pred_values, bins=50)
```

Out[28]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x230ab225128>
```



In [29]:

```
plt.hist(pred_values, bins=50)
```

Out[29]:

```
(array([ 1.,  0.,  0.,  0.,  0.,  1.,  0.,  1.,  2.,  2.,  1.,  3.,  3.,
         1.,  3.,  6.,  6.,  3.,  6.,  9.,  5.,  7.,  5.,  6.,  9.,  6.,
         7.,  7., 10., 10.,  4.,  8.,  4.,  3.,  1.,  2.,  0.,  2.,  0.,
         0.,  1.,  0.,  1.,  1.,  0.,  1.,  0.,  0.,  1.,  1.]),
 array([256.28674001, 266.47152336, 276.65630671, 286.84109006,
        297.02587341, 307.21065676, 317.39544011, 327.58022346,
        337.76500682, 347.94979017, 358.13457352, 368.31935687,
        378.50414022, 388.68892357, 398.87370692, 409.05849027,
        419.24327362, 429.42805697, 439.61284032, 449.79762367,
        459.98240702, 470.16719037, 480.35197372, 490.53675707,
        500.72154042, 510.90632377, 521.09110712, 531.27589048,
        541.46067383, 551.64545718, 561.83024053, 572.01502388,
        582.19980723, 592.38459058, 602.56937393, 612.75415728,
        622.93894063, 633.12372398, 643.30850733, 653.49329068,
        663.67807403, 673.86285738, 684.04764073, 694.23242408,
        704.41720743, 714.60199078, 724.78677414, 734.97155749,
        745.15634084, 755.34112419, 765.52590754]),
 <a list of 50 Patch objects>)
```