In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:

```
df = pd.read_csv("USA_housing.csv")
df.head()
```

Out[2]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05	USNS Raymond\nFPO AE 09386

In [3]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Avg. Area Income             5000 non-null   float64
 1   Avg. Area House Age          5000 non-null   float64
 2   Avg. Area Number of Rooms    5000 non-null   float64
 3   Avg. Area Number of Bedrooms 5000 non-null   float64
 4   Area Population              5000 non-null   float64
 5   Price                        5000 non-null   float64
 6   Address                      5000 non-null   object 
```

```
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

In [4]:
```
df.describe() # seven summary
```

Out[4]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

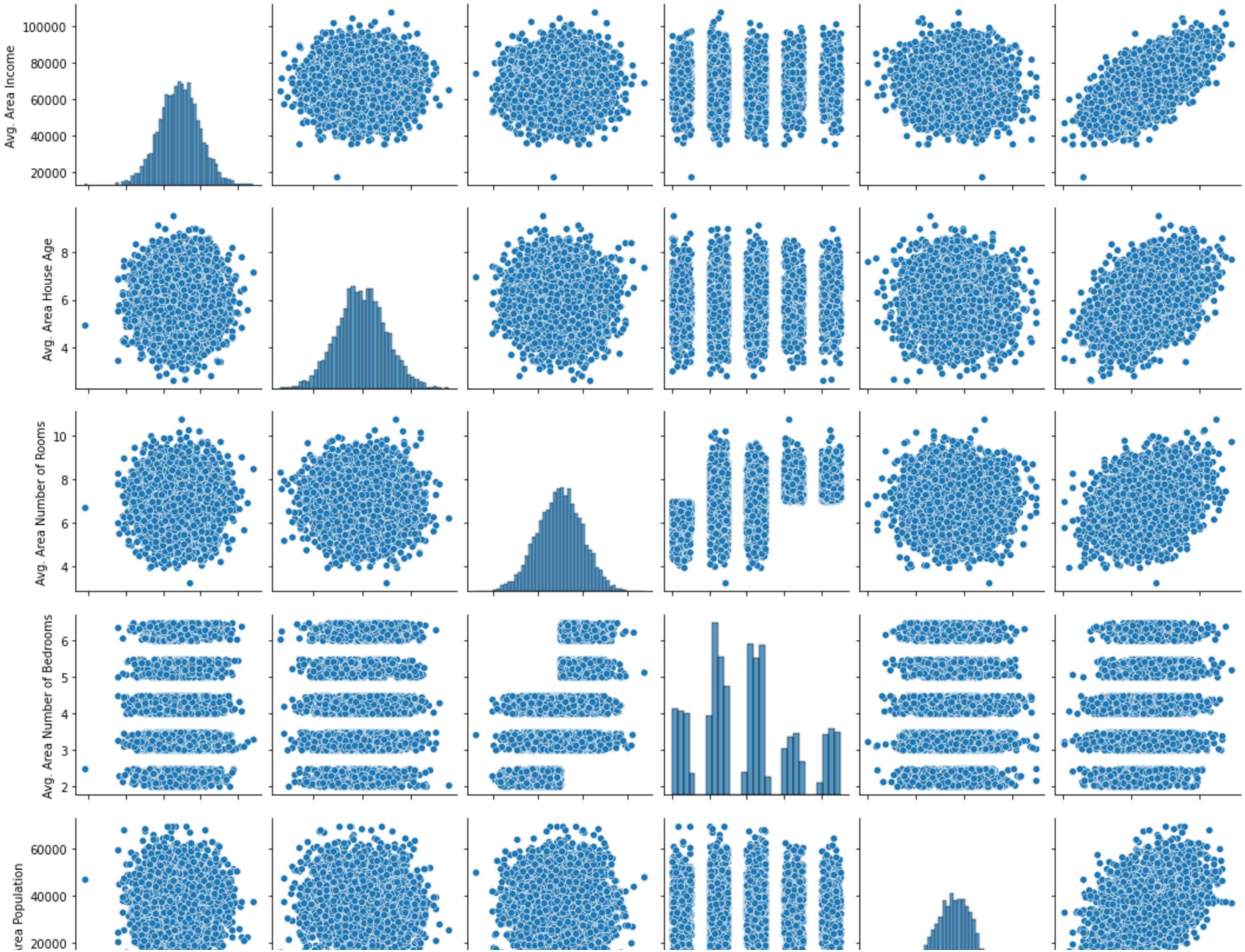
In [5]:
```
#to get column names
df.columns
```

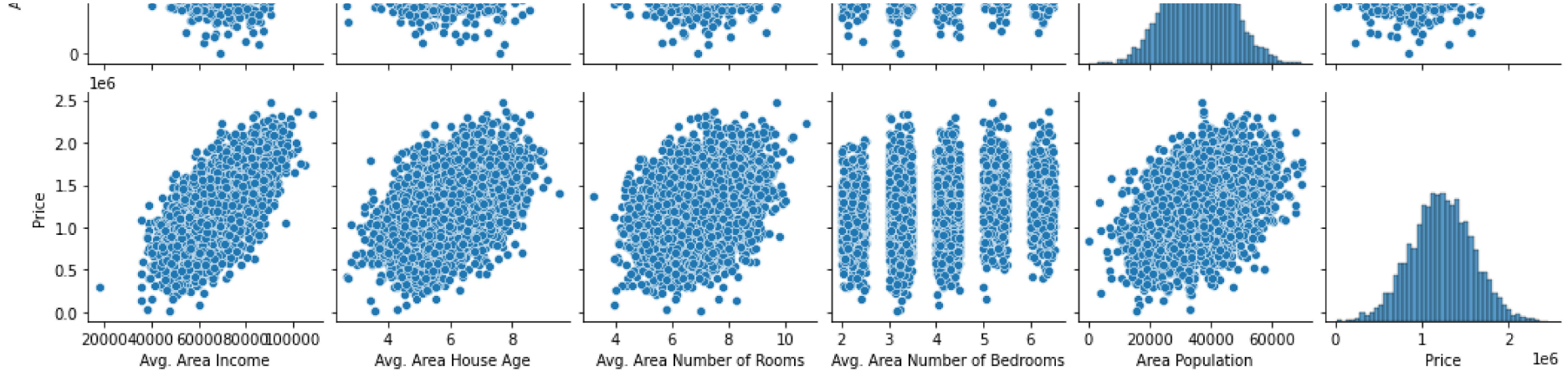
Out[5]:
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

Exploratory Data Analysis

In [6]:
```
sns.pairplot(df)
```

Out[6]:
```
<seaborn.axisgrid.PairGrid at 0x1ba2531b2b0>
```

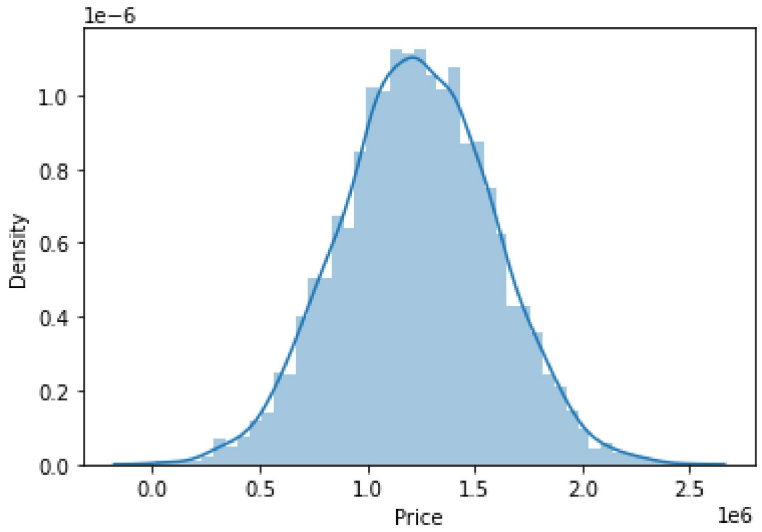

Avg. Area Income
Avg. Area House Age
Avg. Area Number of Rooms
Avg. Area Number of Bedrooms
rea Population
100000
80000
60000
40000
20000
8
6
4
10
6
4
6
5
4
3
2
60000
40000
20000



In [7]:
```
sns.distplot(df['Price'])
```

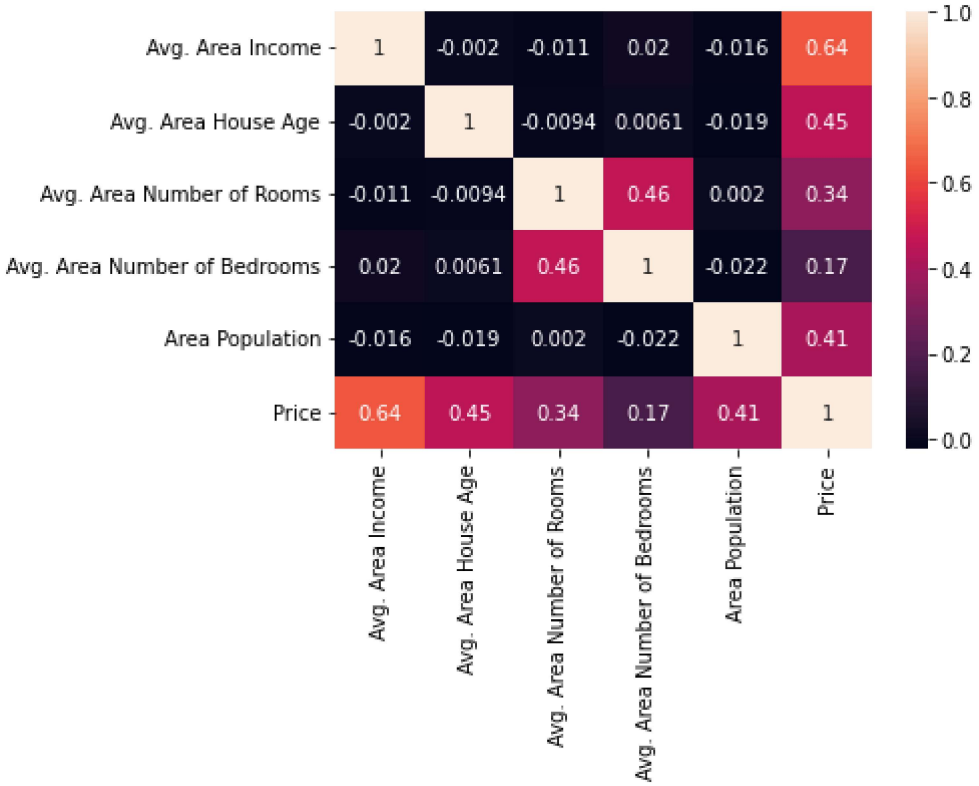
```
C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wil
l be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[7]:
```
<AxesSubplot:xlabel='Price', ylabel='Density'>
```



In [8]:
```
#correlation
sns.heatmap(df.corr(),annot=True)
```

Out[8]: <AxesSubplot:>



Training the Linear Regression model

In [9]:
```
X = df[['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population']]
y = df['Price']
```

In [10]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.4,random_state=104)
```

In [11]:
```
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model
```

Out[11]:
```
LinearRegression()
```

In [12]:
```
model.fit(X_train,y_train)
```

Out[12]:
```
LinearRegression()
```

In [13]:
```
print(model.intercept_)
```

```
-2618630.8922425937
```

In [14]:
```
coeff_df = pd.DataFrame(data=model.coef_,index=X.columns,columns=['coefficient'])
coeff_df
```

Out[14]:

	coefficient
Avg. Area Income	21.427042
Avg. Area House Age	166431.895241
Avg. Area Number of Rooms	120987.471341
Avg. Area Number of Bedrooms	175.296618
Area Population	14.980431

In [15]:
```
#y(price) = inter+ax1+bx2+cx3+dx4+ex5
```

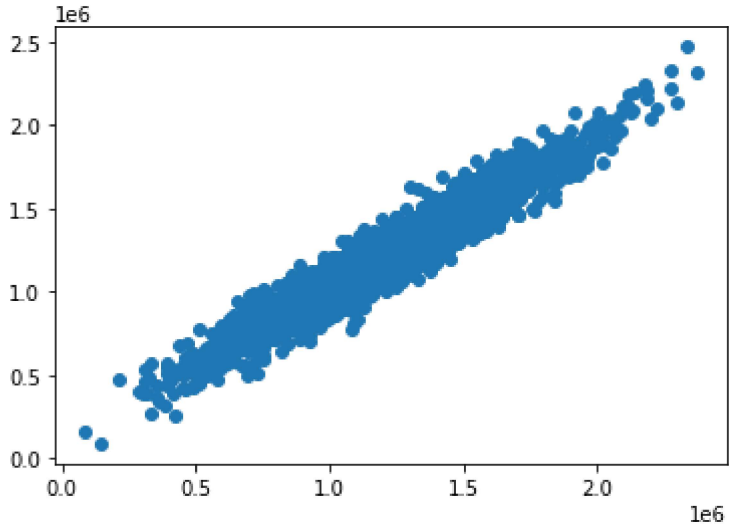
In [16]:
```
y_pred = model.predict(X_test)
```

In [17]:
```
y_pred
```

Out[17]:
```
array([ 507466.94358569, 1525954.35939005,  995897.30753247, ...,
       1128073.03974403, 1480471.83736272, 1061536.60457408])
```

In [18]:
```
plt.scatter(y_test,y_pred)
```

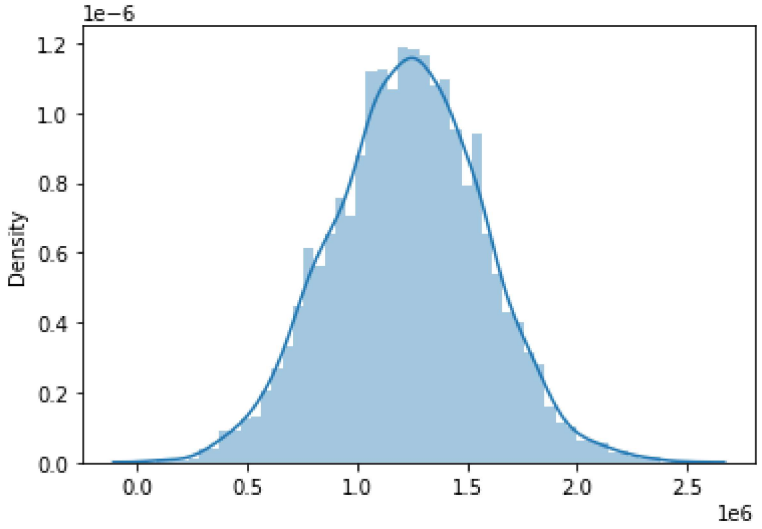
Out[18]:
```
<matplotlib.collections.PathCollection at 0x1ba2a72a550>
```



In [19]:
```
sns.distplot((y_test,y_pred),bins=50)
```

```
C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and wil
l be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility)
or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[19]:
```
<AxesSubplot:ylabel='Density'>
```



In [20]:
```
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MAE: 79967.56466188218
MSE: 10008809400.804377
RMSE: 100044.03730759957
```

In [ ]:

Boston.csv dataset

We will take the Housing dataset which contains information about different houses in Boston.The objective is to predict the value of prices of the house using the given features.

CRIM: Per capita crime rate by town ZN: Proportion of residential land zoned for lots over 25,000 sq.ft INDUS: Proportion of non-retail business acres per town CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise) NOX: Nitric oxide concentration (parts per 10 million) RM: Average number of rooms per dwelling AGE: Proportion of owner-occupied units built prior to 1940 DIS: Weighted distances to five Boston employment centers RAD: Index of accessibility to radial highways TAX: Full-value property tax rate per $10,000 PTRATIO: Pupil-teacher ratio by town

B: 1000(Bk — 0.63)^2, where Bk is the proportion of [people of African American descent] by town LSTAT: Percentage of lower status of the population MEDV: Median value of owner-occupied homes in $1000s

In [21]:
```
data = pd.read_csv("Boston.csv")
data
```

Out[21]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV	CAT. MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0	0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6	0
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	1
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	1
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	22.4	0
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6	0
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9	0
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0	0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	11.9	0

506 rows × 15 columns

In [22]:
```
data.columns
```

Out[22]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'MEDV', 'CAT. MEDV'],
      dtype='object')
```

In [23]:
```
data=data.drop('CAT. MEDV',axis=1)
```

In [24]:
```
data.shape
```

Out[24]:
```
(506, 14)
```

In [25]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

In [26]:
```
data.describe()
```

Out[26]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11

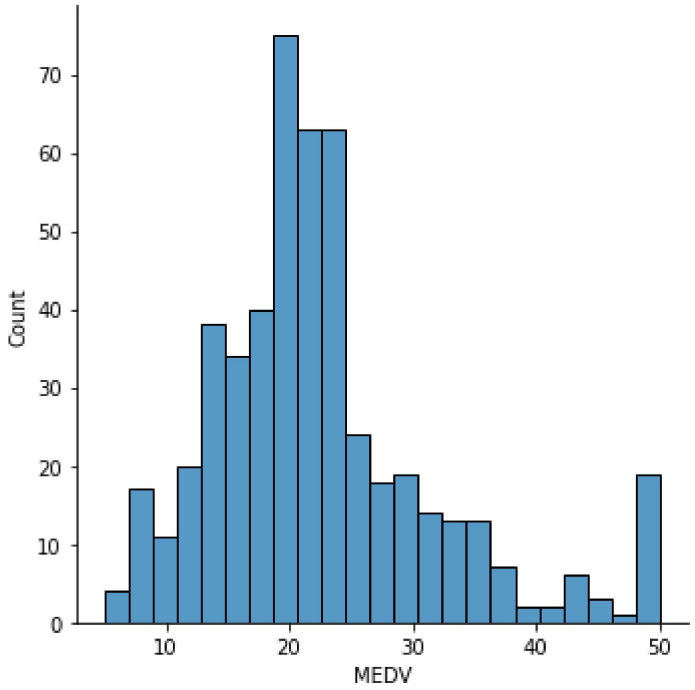
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37

In [27]:
```
data.isnull().sum()
```

Out[27]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

In [28]:
```
sns.displot(data['MEDV']) #to check the distribution of target variable
```

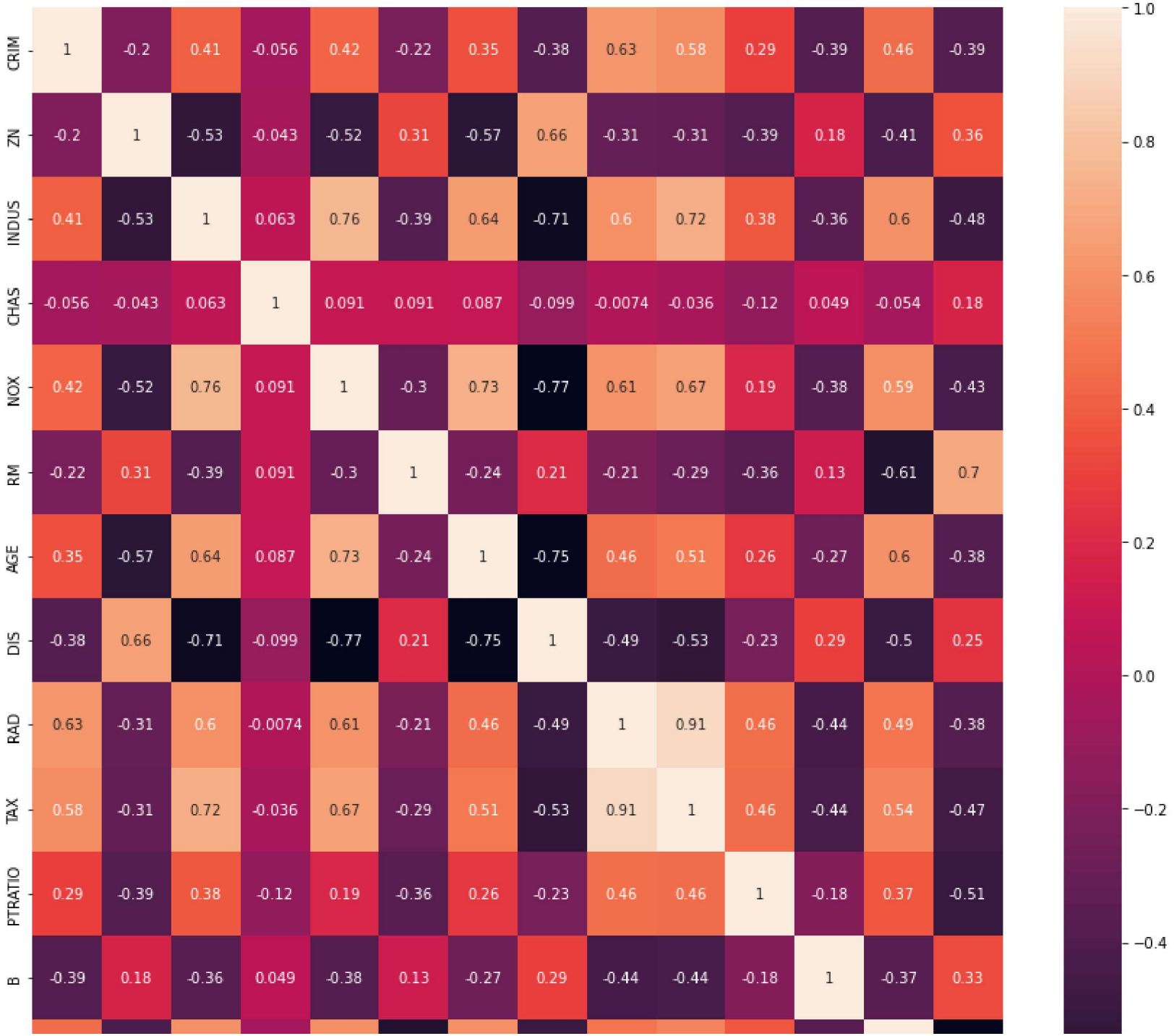
Out[28]:
```
<seaborn.axisgrid.FacetGrid at 0x1ba299056a0>
```

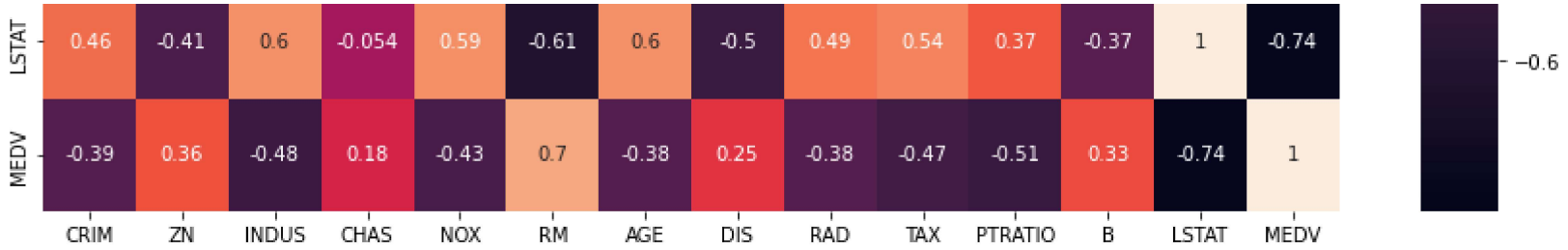


we create a correlation matrix that measures the linear relationships between the variables.The correlation coefficient ranges from -1 to 1. If the value is close to 1, it means that there is a strong positive correlation between the two variables. When it is close to -1, the variables have a strong negative correlation.

In [29]:
```
plt.subplots(figsize=(15,15))
sns.heatmap(data.corr(),annot=True)
```

Out[29]:
```
<AxesSubplot:>
```

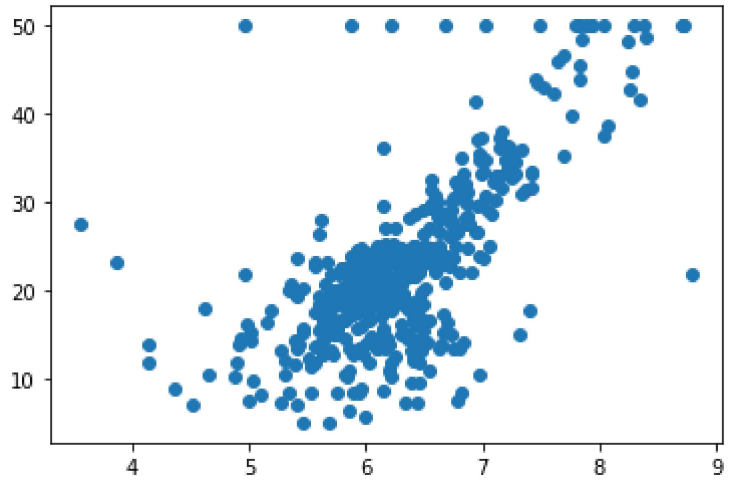

CRIM 1 -0.2 0.41 -0.056 0.42 -0.22 0.35 -0.38 0.63 0.58 0.29 -0.39 0.46 -0.39
ZN -0.2 1 -0.53 -0.043 -0.52 0.31 -0.57 0.66 -0.31 -0.31 -0.39 0.18 -0.41 0.36
INDUS 0.41 -0.53 1 0.063 0.76 -0.39 0.64 -0.71 0.6 0.72 0.38 -0.36 0.6 -0.48
CHAS -0.056 -0.043 0.063 1 0.091 0.091 0.087 -0.099 -0.0074 -0.036 -0.12 0.049 -0.054 0.18
NOX 0.42 -0.52 0.76 0.091 1 -0.3 0.73 -0.77 0.61 0.67 0.19 -0.38 0.59 -0.43
RM -0.22 0.31 -0.39 0.091 -0.3 1 -0.24 0.21 -0.21 -0.29 -0.36 0.13 -0.61 0.7
AGE 0.35 -0.57 0.64 0.087 0.73 -0.24 1 -0.75 0.46 0.51 0.26 -0.27 0.6 -0.38
DIS -0.38 0.66 -0.71 -0.099 -0.77 0.21 -0.75 1 -0.49 -0.53 -0.23 0.29 -0.5 0.25
RAD 0.63 -0.31 0.6 -0.0074 0.61 -0.21 0.46 -0.49 1 0.91 0.46 -0.44 0.49 -0.38
TAX 0.58 -0.31 0.72 -0.036 0.67 -0.29 0.51 -0.53 0.91 1 0.46 -0.44 0.54 -0.47
PTRATIO 0.29 -0.39 0.38 -0.12 0.19 -0.36 0.26 -0.23 0.46 0.46 1 -0.18 0.37 -0.51
B -0.39 0.18 -0.36 0.049 -0.38 0.13 -0.27 0.29 -0.44 -0.44 -0.18 1 -0.37 0.33
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4



The prices increase as the value of RM increases linearly.

In [30]:
```
plt.scatter(data['RM'],data['MEDV'])
```

Out[30]:
```
<matplotlib.collections.PathCollection at 0x1ba2b6e4ee0>
```



In [31]:
```
X = pd.DataFrame(np.c_[data['LSTAT'], data['RM']], columns = ['LSTAT','RM'])
y = data['MEDV']
```

In [32]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=5)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(404, 2)
(102, 2)
(404,)
(102,)
```

In [33]:
```
from sklearn.linear_model import LinearRegression
boston=LinearRegression()
```

In [34]:
```
boston.fit(X_train,y_train)
```

Out[34]:
```
LinearRegression()
```

In [35]:
```
# model evaluation for training set
from sklearn import metrics
y_train_predict = boston.predict(X_train)
rmse = (np.sqrt(metrics.mean_squared_error(y_train, y_train_predict)))
r2 = metrics.r2_score(y_train, y_train_predict)
```

In [36]:
```
print("The model performance for training set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))++
print("\n")
```

```
The model performance for training set
--------------------------------------
RMSE is 5.6371293350711955
R2 score is 0.6300745149331701
```

In [37]:
```
# model evaluation for testing set
y_test_predict = boston.predict(X_test)
rmse = (np.sqrt(metrics.mean_squared_error(y_test, y_test_predict)))
r2 = metrics.r2_score(y_test, y_test_predict)
```

In [38]:
```
print("The model performance for testing set")
print("--------------------------------------")
print('RMSE is {}'.format(rmse))
print('R2 score is {}'.format(r2))
```

```
The model performance for testing set
--------------------------------------
RMSE is 5.137400784702911
R2 score is 0.6628996975186953
```

problem statement: You just got some contract work with an Ecommerce company based in New York City that sells clothing online but they also have in-store style and clothing advice sessions. Customers come in to the store, have sessions/meetings with a personal stylist, then they can go home and order either on a mobile app or website for the clothes they want.

The company is trying to decide whether to focus their efforts on their mobile app experience or their website. They've hired you on contract to help them figure it out! Let's get started!

In [39]:
```
customers = pd.read_csv("Ecommerce Customers")
```

In [40]:
```
customers.head()
```

Out[40]:

	Email	Address	Avatar	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
0	mstephenson@fernandez.com	835 Frank Tunnel\nWrightmouth, MI 82180-9605	Violet	34.497268	12.655651	39.577668	4.082621	587.951054
1	hduke@hotmail.com	4547 Archer Common\nDiazchester, CA 06566-8576	DarkGreen	31.926272	11.109461	37.268959	2.664034	392.204933
2	pallen@yahoo.com	24645 Valerie Unions Suite 582\nCobbborough, D...	Bisque	33.000915	11.330278	37.110597	4.104543	487.547505
3	riverarebecca@gmail.com	1414 David Throughway\nPort Jason, OH 22070-1220	SaddleBrown	34.305557	13.717514	36.721283	3.120179	581.852344

	Email	Address	Avatar	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
4	mstephens@davidson-herman.com	14023 Rodriguez Passage\nPort Jacobville, PR 3...	MediumAquaMarine	33.330673	12.795189	37.536653	4.446308	599.406092

We'll work with the Ecommerce Customers csv file from the company. It has Customer info, suchas Email, Address, and their color Avatar. Then it also has numerical value columns:

- Avg. Session Length: Average session of in-store style advice sessions.
- Time on App: Average time spent on App in minutes
- Time on Website: Average time spent on Website in minutes
- Length of Membership: How many years the customer has been a member.

In [41]:
```
customers.describe()
```

Out[41]:

	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
count	500.000000	500.000000	500.000000	500.000000	500.000000
mean	33.053194	12.052488	37.060445	3.533462	499.314038
std	0.992563	0.994216	1.010489	0.999278	79.314782
min	29.532429	8.508152	33.913847	0.269901	256.670582
25%	32.341822	11.388153	36.349257	2.930450	445.038277
50%	33.082008	11.983231	37.069367	3.533975	498.887875
75%	33.711985	12.753850	37.716432	4.126502	549.313828
max	36.139662	15.126994	40.005182	6.922689	765.518462

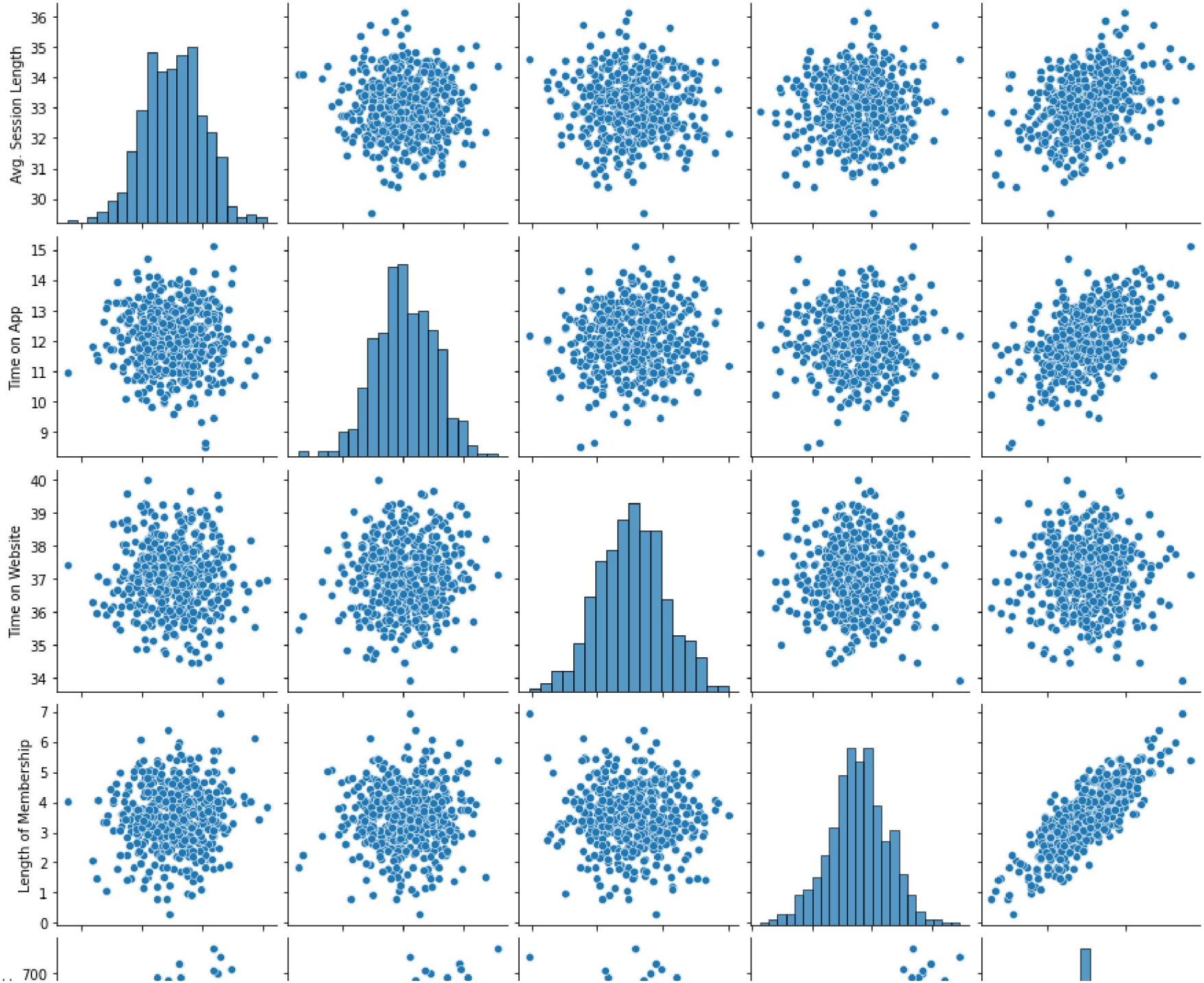
In [42]:
```
customers.info()
```

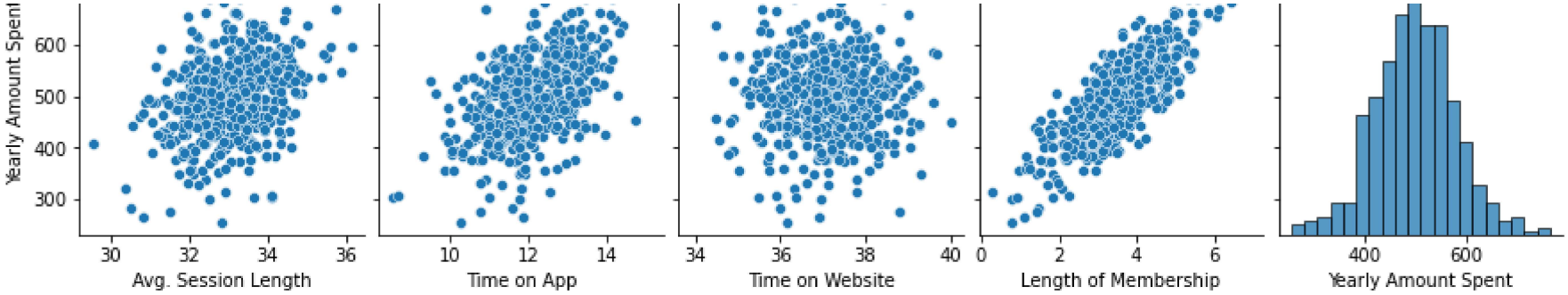
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
```

```
---  ------                --------------  -----
 0   Email                 500 non-null    object
 1   Address               500 non-null    object
 2   Avatar                500 non-null    object
 3   Avg. Session Length   500 non-null    float64
 4   Time on App           500 non-null    float64
 5   Time on Website       500 non-null    float64
 6   Length of Membership  500 non-null    float64
 7   Yearly Amount Spent   500 non-null    float64
dtypes: float64(5), object(3)
memory usage: 31.4+ KB
```

In [43]:
```
sns.pairplot(customers)
```

Out[43]:
```
<seaborn.axisgrid.PairGrid at 0x1ba2b0aa340>
```


Avg. Session Length
Time on App
Time on Website
Length of Membership
36
35
34
33
32
31
30
15
14
13
12
11
10
9
40
39
38
37
36
35
34
7
6
5
4
3
2
1
0
700



From this pairplot we observe that the length of membership is more correlated with the yearly amount spent

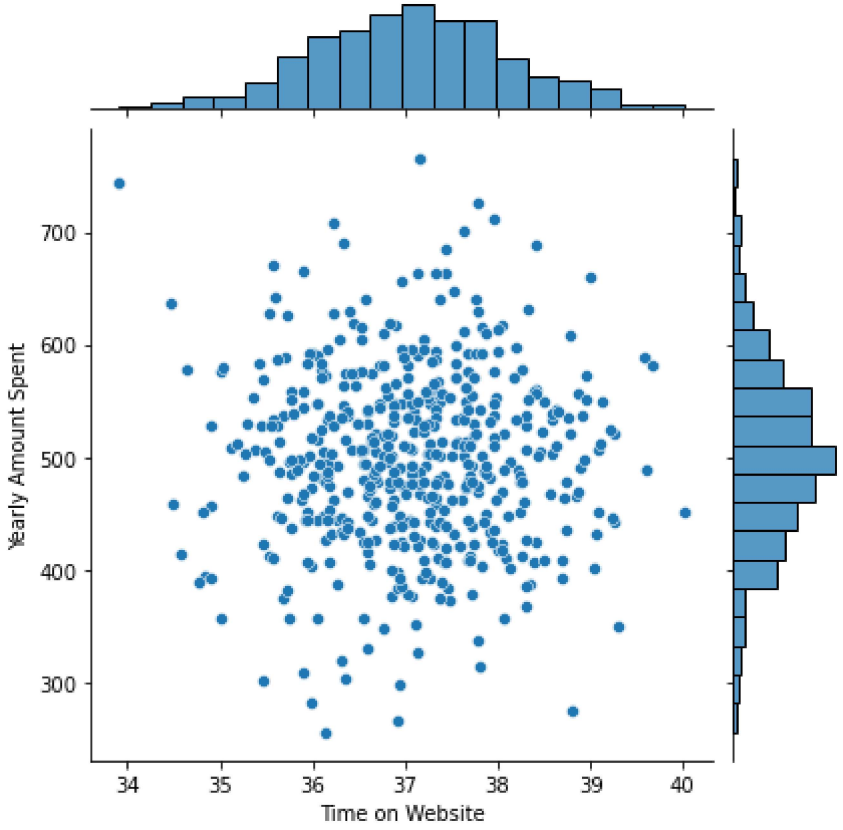
In [44]:
```
sns.heatmap(customers.corr(),cmap="YlGnBu",annot=True)
```

Out[44]:
```
<AxesSubplot:>
```

Use seaborn to create a jointplot to compare the Time on Website and Yearly Amount Spent columns.similarly with time spent on app

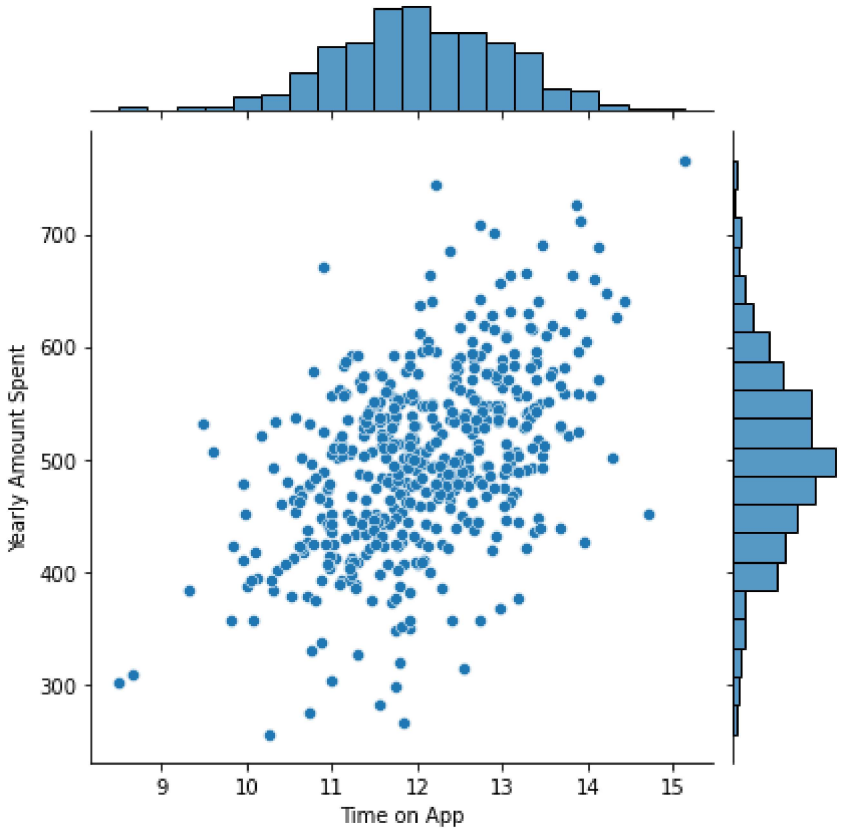
In [45]:
```
sns.jointplot(x='Time on Website',y='Yearly Amount Spent',data=customers)
```

Out[45]:
```
<seaborn.axisgrid.JointGrid at 0x1ba2b0aa8e0>
```



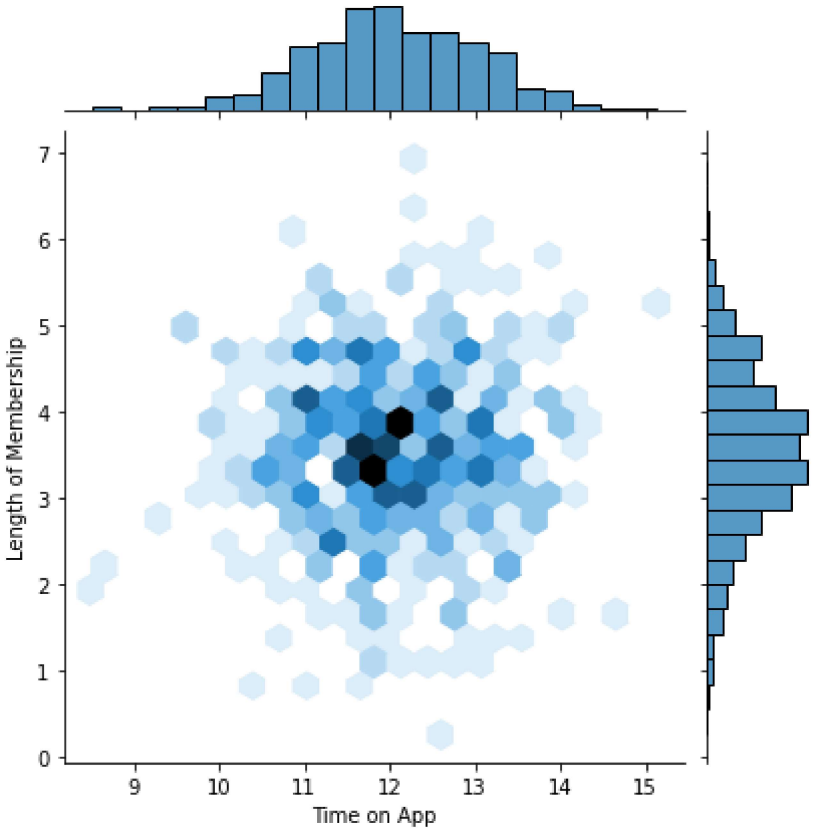
In [46]:
```
sns.jointplot(x='Time on App',y='Yearly Amount Spent',data=customers)
```

Out[46]:
```
<seaborn.axisgrid.JointGrid at 0x1ba2d7110a0>
```



In [47]:
```
sns.jointplot(x='Time on App',y='Length of Membership',kind='hex',data=customers)
```

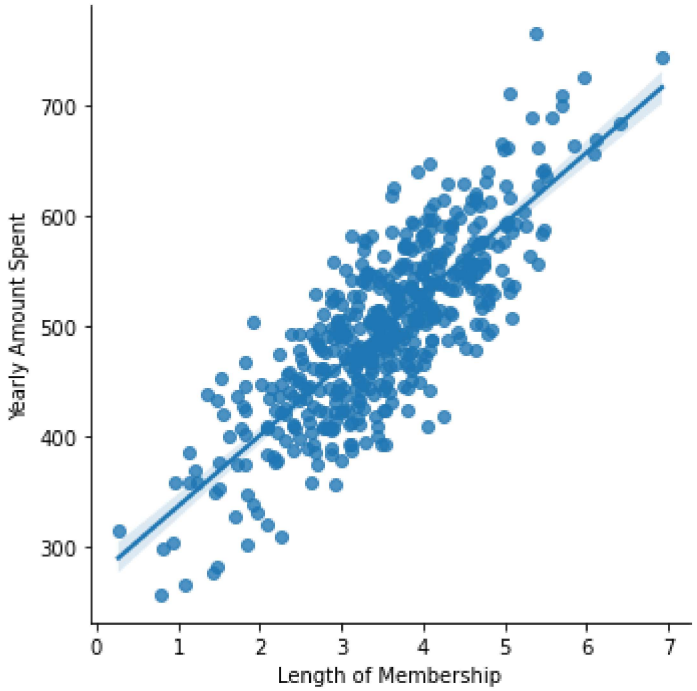
Out[47]:
```
<seaborn.axisgrid.JointGrid at 0x1ba2d83aa00>
```



Create a linear model plot (using seaborn's lmplot) of Yearly Amount Spent vs. Length of Membership.

In [48]:
```
sns.lmplot(x='Length of Membership',y='Yearly Amount Spent',data=customers)
```

Out[48]:
```
<seaborn.axisgrid.FacetGrid at 0x1ba2b09ffd0>
```



In [51]:
```
y = customers['Yearly Amount Spent']
X = customers[['Avg. Session Length', 'Time on App','Time on Website', 'Length of Membership']]
```

In [52]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

In [53]:
```
lm = LinearRegression()
lm.fit(X_train,y_train)
```

Out[53]:
```
LinearRegression()
```

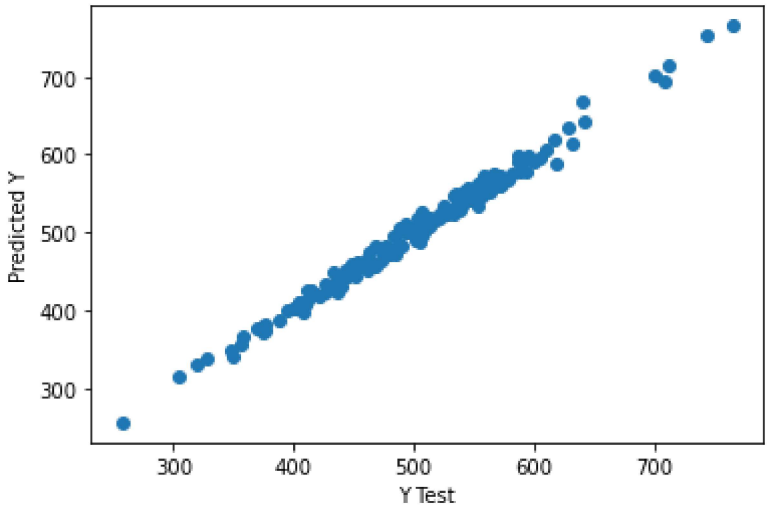
In [54]:
```
# The coefficients
print('Coefficients: \n', lm.coef_)
```

```
Coefficients:
 [25.98154972 38.59015875  0.19040528 61.27909654]
```

In [55]:
```
predictions = lm.predict( X_test)
```

In [56]:
```
plt.scatter(y_test,predictions)
plt.xlabel('Y Test')
plt.ylabel('Predicted Y')
```

Out[56]:
```
Text(0, 0.5, 'Predicted Y')
```



In [57]:
```
# calculate these metrics by hand!
from sklearn import metrics

print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 7.2281486534308295
MSE: 79.81305165097442
RMSE: 8.933815066978632
```

In [58]:
```
coeffecients = pd.DataFrame(lm.coef_,X.columns)
coeffecients.columns = ['Coeffecient']
coeffecients
```

Out[58]:

	Coeffecient
Avg. Session Length	25.981550
Time on App	38.590159
Time on Website	0.190405
Length of Membership	61.279097

Interpreting the coefficients:

- Holding all other features fixed, a 1 unit increase in **Avg. Session Length** is associated with an **increase of 25.98 total dollars spent.**
- Holding all other features fixed, a 1 unit increase in **Time on App** is associated with an **increase of 38.59 total dollars spent**.
- Holding all other features fixed, a 1 unit increase in **Time on Website** is associated with an **increase of 0.19 total dollars spent**.
- Holding all other features fixed, a 1 unit increase in **Length of Membership** is associated with an **increase of 61.27 total dollars spent.**

Do you think the company should focus more on their mobile app or on their website?

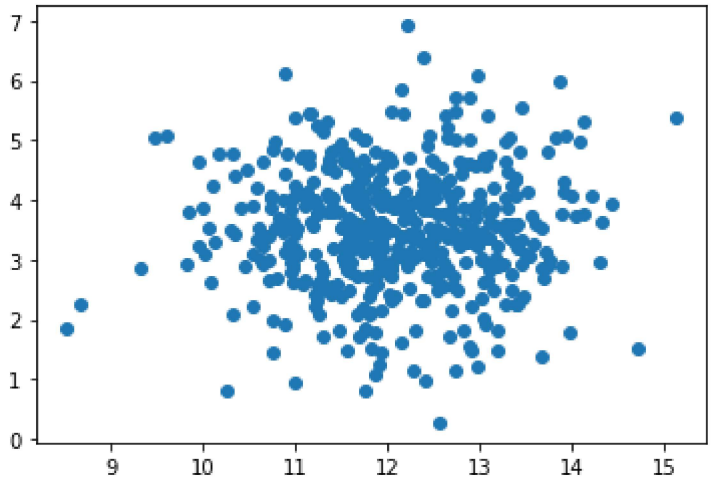
This is tricky, there are two ways to think about this: Develop the Website to catch up to the performance of the mobile app, or develop the app more since that is what is working better. This sort of answer really depends on the other factors going on at the company, you would probably want to explore the relationship between Length of Membership and the App or the Website before coming to a conclusion!

In [59]:
```
customers.columns
```

Out[59]:
```
Index(['Email', 'Address', 'Avatar', 'Avg. Session Length', 'Time on App',
       'Time on Website', 'Length of Membership', 'Yearly Amount Spent'],
      dtype='object')
```

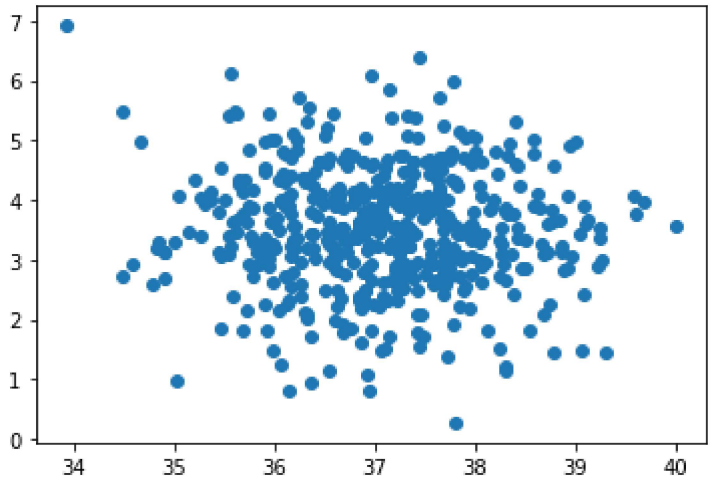
In [60]:
```
plt.scatter(customers['Time on App'],customers['Length of Membership'])
```

Out[60]:
```
<matplotlib.collections.PathCollection at 0x1ba2ec10430>
```



In [61]:
```
plt.scatter(customers['Time on Website'],customers['Length of Membership'])
```

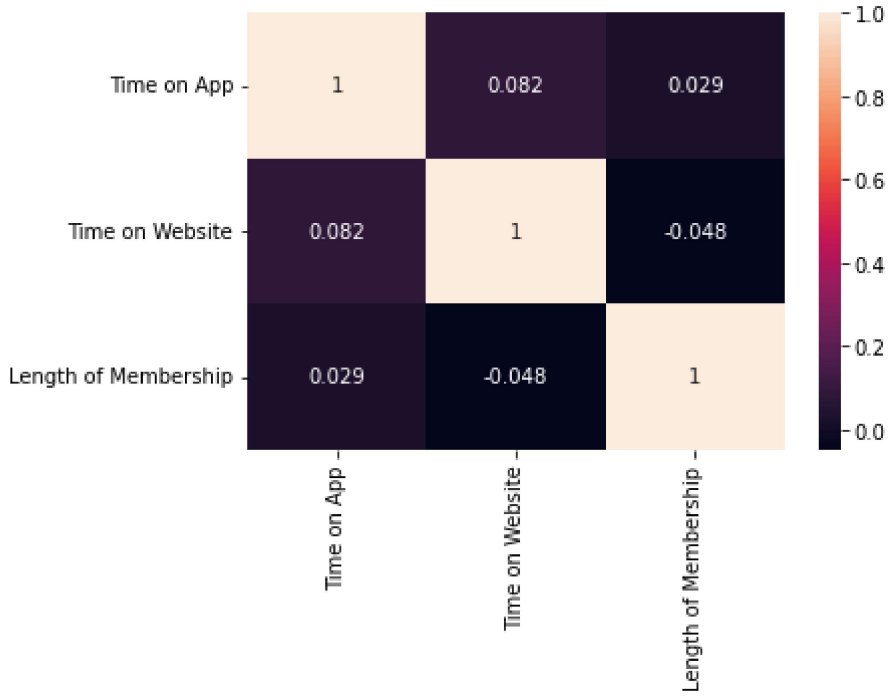
Out[61]:
```
<matplotlib.collections.PathCollection at 0x1ba2ec7a820>
```



In [66]:
```
df_corr = customers[['Time on App','Time on Website', 'Length of Membership']]
```

In [67]:
```
sns.heatmap(df_corr.corr(),annot=True)
```

Out[67]: <AxesSubplot:>



In [ ]: