Assignment 3

Elena Pérez-Ródenas Martínez
REDID: 827222533

October 2021

I, Elena Pérez-Ródenas Martínez, guarantee that this homework is my independent work and I have never copied any part from other resources. Also, I acknowledge and agree with the plagiarism penalty specified in the course syllabus.

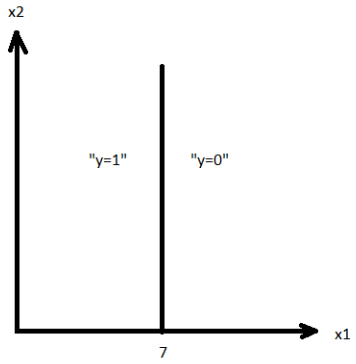
1. Exercise

1.1.

Our estimate for P(y=0|x;θ) is 0.81. Explanation: P(y=0|x;θ) = 1 - P(y = 1| x; θ); the former is 1 - 0.19 = 0.81

Our estimate for P(y=1|x;θ) is 0.19. Explanation: $h_\theta(x) = 0{,}19$

1.2.



$h_\theta(x) = g(\theta^T x)$ and $g(z) = \frac{1}{1+e^{-z}}$

If we look at the sigmoid function we know that we should predict "$y = 1$" if $h_\theta(x) \geq 0{,}5 \Rightarrow h_\theta(x) = g(\theta^T x) \geq 0{,}5$ which happens whenever $\theta^T x \geq 0$ so whenever $7 - x_1 \geq 0 \Rightarrow x_1 \leq 7$

For the same reason we predict "$y = 0$" if $h_\theta < 0{,}5 \Rightarrow h_\theta(x) = g(\theta^T x) < 0{,}5$ which happens whenever $\theta^T x < 0$ so whenever $7 - x_1 < 0 \Rightarrow x_1 > 7$

1.3.

The formula for the cost function in logistic regression is given by $Cost = \frac{-1}{m} \sum_{i=1}^{m} [yloga + (1-y)log(1-a)]]$ where a are our predictions $a = \sigma(w^T x + b) = \sigma(z)$ with $z = w^T x + b$ and $\sigma(x) = \frac{1}{1+e^{-x}}$

So if we have m observations the cost function represents the average of all the errors of each observation.

We will name the Loss error as $L = -[yloga + (1-y)log(1-a)]$

Also $w = \begin{bmatrix} w_1 \\ w_2 \\ ... \\ w_n \end{bmatrix}$

Our goal is to calculate $\frac{\partial Cost}{\partial w}$ so first we will calculate $\frac{\partial L}{\partial w_1}$, $\frac{\partial L}{\partial w_2}$ ...
For simplicity let's calculate $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a} \text{x} \frac{\partial a}{\partial w} = \frac{\partial L}{\partial a} \text{x} \frac{\partial a}{\partial z} \text{x} \frac{\partial z}{\partial w}$

$\frac{\partial L}{\partial a} = \frac{-y}{a} + \frac{(1-y)}{(1-a)}$
$\frac{\partial a}{\partial z} = [(1+e^{-z})^{-1}]' = \frac{e^{-z}}{(1+e^{-z})^2}$ and we know that $a^2 = \frac{1}{(1+e^{-z})^2}$ and $e^{-z} = \frac{(1-a)}{a}$ so
$\frac{\partial a}{\partial z} = \frac{(1-a)}{a} a^2 = a(1-a)$
$\frac{\partial z}{\partial w} = x + 0 = x \Rightarrow$

$\frac{\partial L}{\partial w} = [\frac{-y}{a} + \frac{(1-y)}{(1-a)}]a(1-a)x = [\frac{-y+ay+a-ay}{a(1-a)}]a(1-a)x = \frac{a-y}{a(1-a)}a(1-a)x = (a-y)x$

so $\frac{\partial L}{\partial w_1} = (a-y)x_1$, $\frac{\partial L}{\partial w_2} = (a-y)x_2$, ..., $\frac{\partial L}{\partial w_n} = (a-y)x_n$ so $\frac{\partial L}{\partial W} = (A-Y)X$ where W is the martix of all the w parameters, A is the matrix of all the predictions, Y is the matrix of all the output values and X is the complete input of all the observations.

$\Rightarrow \frac{\partial Cost}{\partial W} = (A-Y)X$

Now let's calculate $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a} \text{x} \frac{\partial a}{\partial z} \text{x} \frac{\partial z}{\partial b}$

$\frac{\partial L}{\partial a} = \frac{(a-y)}{a(1-a)}$, $\frac{\partial a}{\partial z} = a(1-a)$ and $\frac{\partial z}{\partial b} = 1$
so $\frac{\partial L}{\partial b} = \frac{(a-y)}{a(1-a)}a(1-a)1 = (a-y) \Rightarrow \frac{\partial Cost}{\partial b} = (A-Y)$

1.4.

Gradient Descent is a machine learning technique that can help us make predictions. First we set an initial value and then calculate the sum of the squared residuals to see how well our prediction fits the data. Then we will select the prediction that has the less sum of the squared residuals. In order to do that, gradient descent identifies the optimal value by taking big steps when it is far away and baby steps when it's close. We know that because the closer we get to the optimal value, the closer the slope of the curve gets to 0.

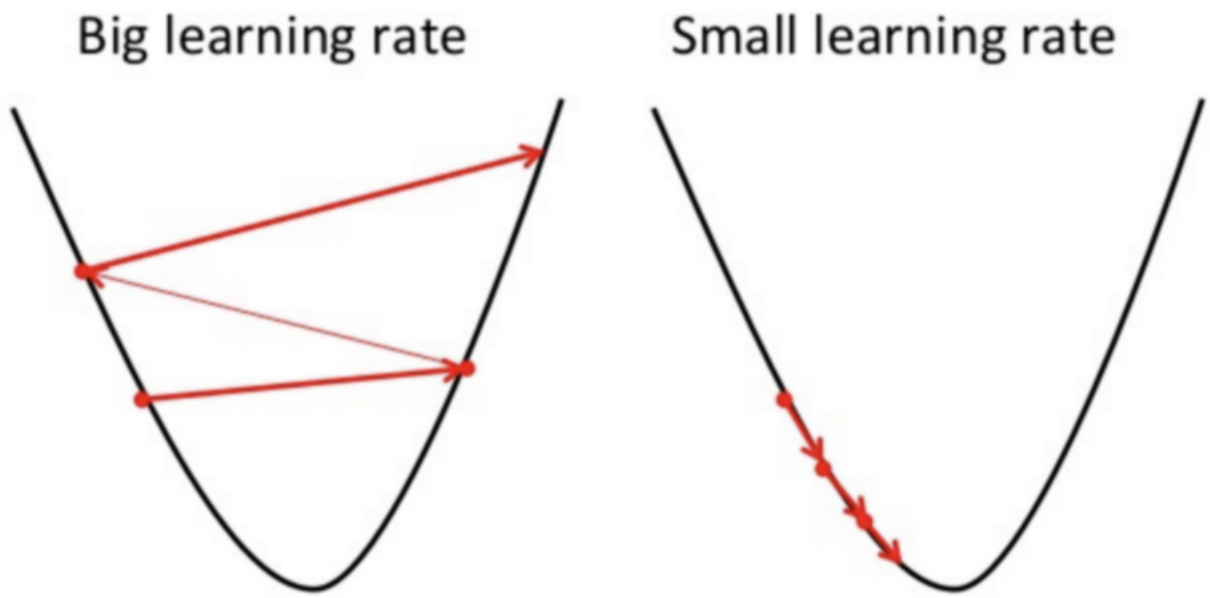
Gradient Descent determines the step size by multiplying the slope by a small number called the learning rate. The new prediction will be the old one minus the step size.

Gradient Descent will stop taking steps when the step size is very close to 0. It also sets a limit on the number of steps.

So the formula for the gradient descent will be $w_{i+1} = w_i - \alpha \frac{\partial Cost}{\partial w_i}$ and since we know from exercise c that $\frac{\partial Cost}{\partial w_i} = (a-y)x$ then $w_{i+1} = w_i - \alpha(a-y)x$

1.5.

Learning rate is used to scale the magnitude of parameter updates during gradient descent. The gradient descent update rule is given by $w_i = w_i - \alpha \frac{\partial Cost}{\partial w_i}$ where α is called the learning rate.
If α is too small the convergence will be slower and if it's too large the cost function may not decrease on every iteration and may not converge.
If the cost function increases with the number of iterations that's a sign that gradient descent is not working. In that case we should use a smaller α. This is because if the learning rate is too large instead of getting closer to the minimum we will get farther from it as it is shown in the picture below:



The key is to find the biggest learning rate that converges so that it's working and fast.

2. Coding exercise

2.1. Task 1

First we will read the csv file with the following code:

```
customers = pd.read_csv('./Ecomm-Customers.csv')
```

In order to print the information regarding the columns we will create a DataFrame df with the different attributes.

```
emails = customers['Email']
addresses = customers['Address']
avatars = customers['Avatar']
avgs = customers['Avg. Session Length']
timesapp = customers['Time on App']
timesweb = customers['Time on Website']
lengths = customers['Length of Membership']
yearlys = customers['Yearly Amount Spent']
df = pd.DataFrame({"Email": emails, "Address": addresses,
                   "Avatar": avatars, "Avg. Session Length": avgs,
                   "Time on App": timesapp, "Time on Website": timesweb,
                   "Length of Membership": lengths, "Yearly Amount Spent": yearlys
                  })
```

Finally we will print the first rows with the command:

```
df.head()
```

The output obtained is the following:

	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
4	mstephens@davidson-herman.com	14023 Rodriguez Passage\nPort Jacobville, PR 3...	MediumAquaMarine	33.330673	12.795189	37.536653	4.446308	599.406092

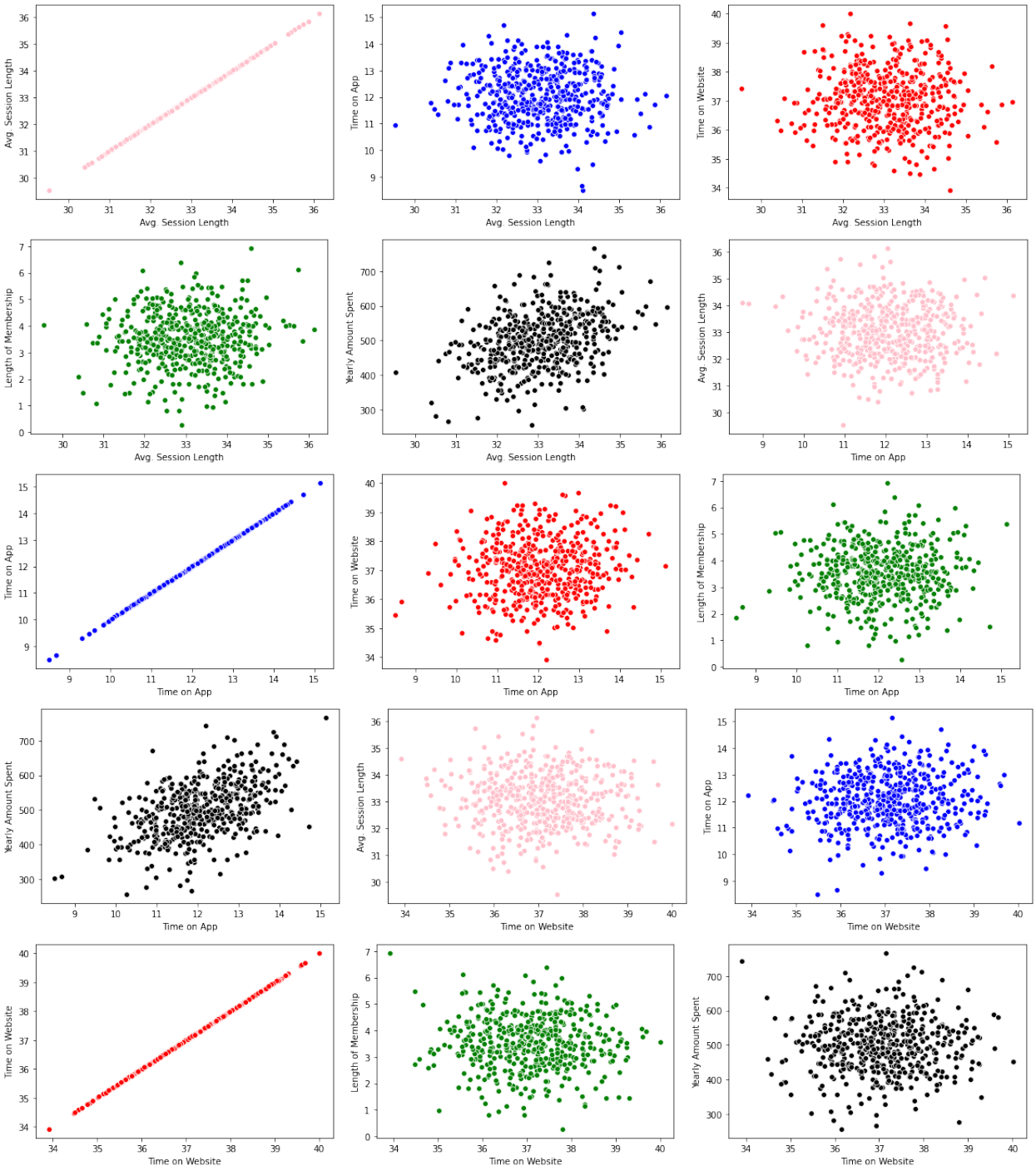
2.2. Task 2

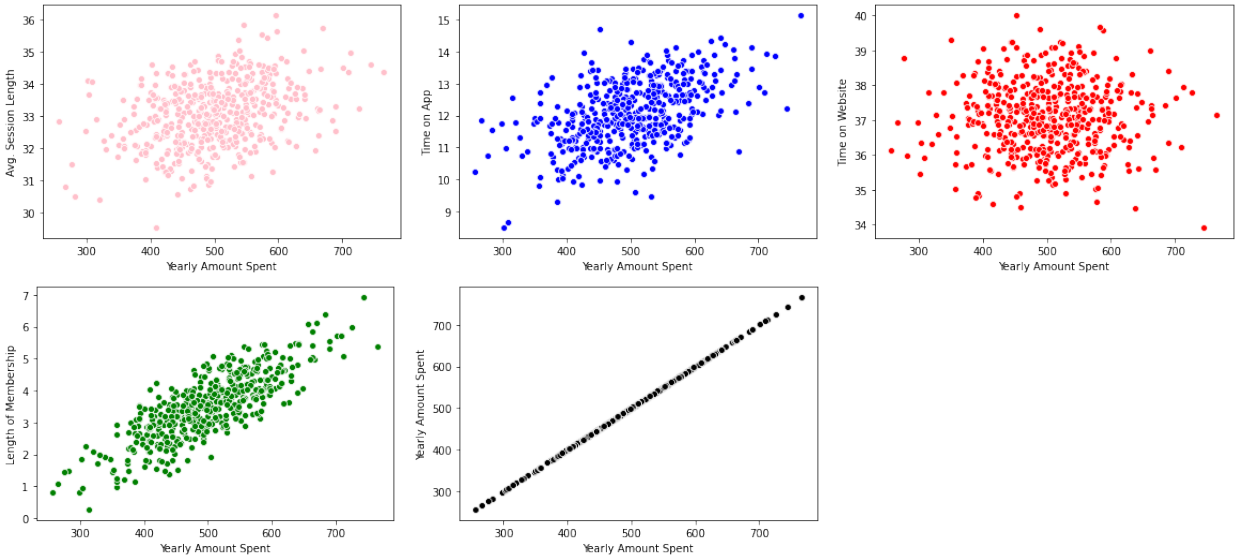
To find the correlation between all the columns in customers in terms of scatter-plots we will make use of sns library and do the scatter-plots of every combination of columns.

```
sns.scatterplot(avgs, avgs, color='pink')
plt.figure()
sns.scatterplot(avgs, timesapp, color='blue')
plt.figure()
sns.scatterplot(avgs, timesweb, color='red')
plt.figure()
sns.scatterplot(avgs, lengths, color='green')
plt.figure()
sns.scatterplot(avgs, yearlys, color='black')
plt.figure()
sns.scatterplot(timesapp, avgs, color='pink')
plt.figure()
sns.scatterplot(timesapp, timesapp, color='blue')
plt.figure()
sns.scatterplot(timesapp, timesweb, color='red')
plt.figure()
sns.scatterplot(timesapp, lengths, color='green')
plt.figure()
sns.scatterplot(timesapp, yearlys, color='black')
plt.figure()
sns.scatterplot(timesweb, avgs, color='pink')
plt.figure()
sns.scatterplot(timesweb, timesapp, color='blue')
plt.figure()
sns.scatterplot(timesweb, timesweb, color='red')
plt.figure()
sns.scatterplot(timesweb, lengths, color='green')
plt.figure()
sns.scatterplot(timesweb, yearlys, color='black')
plt.figure()
sns.scatterplot(yearlys, avgs, color='pink')
plt.figure()
sns.scatterplot(yearlys, timesapp, color='blue')
plt.figure()
sns.scatterplot(yearlys, timesweb, color='red')
plt.figure()
sns.scatterplot(yearlys, lengths, color='green')
plt.figure()
```

```
sns.scatterplot(yearlys, yearlys, color='black')
plt.figure()
```

These are the plots obtained:





As we can see, the data distribution is quite normal and there is a clear correlation between length of membership and yearly amount spent.

Now we will find the correlations using heatmaps. In order to do so we will use sns.heatmap and as a parameter we will create a new dataframe called heatmap1_data with combining the different columns to plot the heatmaps. It's important to take the floor part of each value so that the graph is more clear. We will also divide the Yearly Amount Spent so that the numbers are not that big.

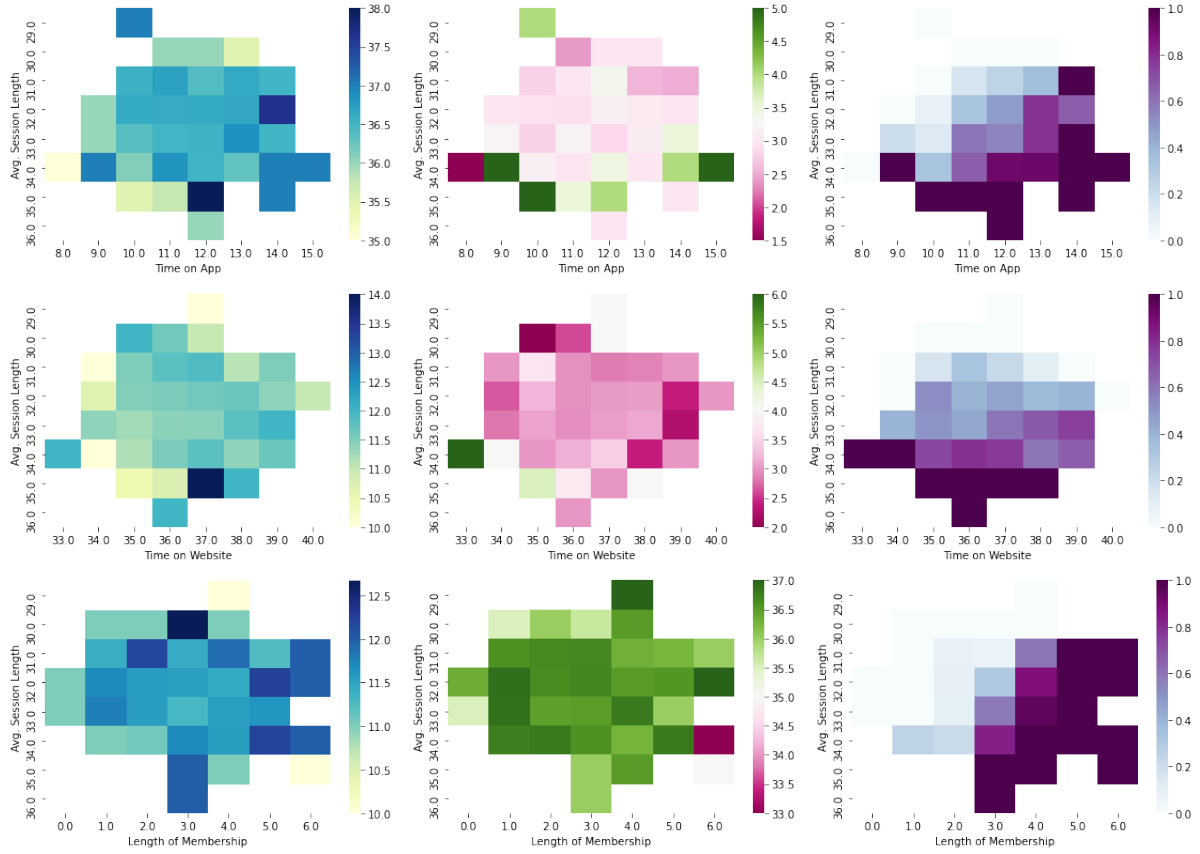
```
customers2 = customers.copy()
cols = customers.columns
for col in cols[3:]:
    customers2[col] = customers[col].apply(np.floor)
customers2['Yearly Amount Spent'] = (customers['Yearly Amount Spent']/500).apply(np.floor)

heatmap1_data = pd.pivot_table(customers2, values='Time on Website',
                      index=['Avg. Session Length'],
                      columns='Time on App')
sns.heatmap(heatmap1_data, cmap="YlGnBu")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Length of Membership',
                      index=['Avg. Session Length'],
                      columns='Time on App')
sns.heatmap(heatmap1_data, cmap="PiYG")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Yearly Amount Spent',
                      index=['Avg. Session Length'],
                      columns='Time on App')
sns.heatmap(heatmap1_data, cmap="BuPu")
plt.figure()

heatmap1_data = pd.pivot_table(customers2, values='Time on App',
                      index=['Avg. Session Length'],
                      columns='Time on Website')
sns.heatmap(heatmap1_data, cmap="YlGnBu")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Length of Membership',
                      index=['Avg. Session Length'],
                      columns='Time on Website')
sns.heatmap(heatmap1_data, cmap="PiYG")
```

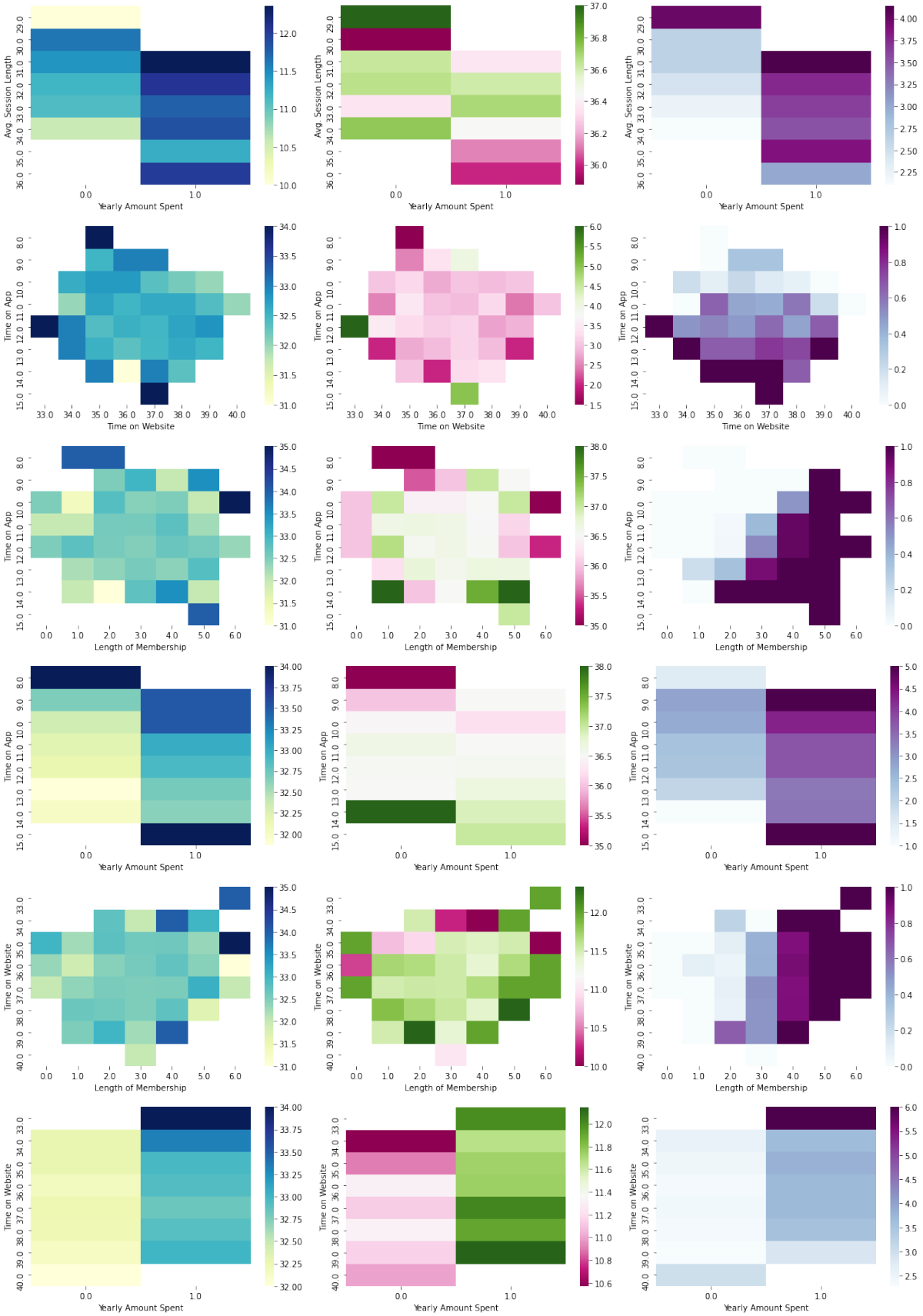
```
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Yearly Amount Spent',
                     index=['Avg. Session Length'],
                     columns='Time on Website')
sns.heatmap(heatmap1_data, cmap="BuPu")
plt.figure()

heatmap1_data = pd.pivot_table(customers2, values='Time on App',
                     index=['Avg. Session Length'],
                     columns='Length of Membership')
sns.heatmap(heatmap1_data, cmap="YlGnBu")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Time on Website',
                     index=['Avg. Session Length'],
                     columns='Length of Membership')
sns.heatmap(heatmap1_data, cmap="PiYG")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Yearly Amount Spent',
                     index=['Avg. Session Length'],
                     columns='Length of Membership')
sns.heatmap(heatmap1_data, cmap="BuPu")
plt.figure()

heatmap1_data = pd.pivot_table(customers2, values='Time on App',
                     index=['Avg. Session Length'],
                     columns='Yearly Amount Spent')
sns.heatmap(heatmap1_data, cmap="YlGnBu")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Time on Website',
                     index=['Avg. Session Length'],
                     columns='Yearly Amount Spent')
sns.heatmap(heatmap1_data, cmap="PiYG")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Length of Membership',
                     index=['Avg. Session Length'],
                     columns='Yearly Amount Spent')
sns.heatmap(heatmap1_data, cmap="BuPu")
plt.figure()

heatmap1_data = pd.pivot_table(customers2, values='Avg. Session Length',
                     index=['Time on App'],
                     columns='Time on Website')
sns.heatmap(heatmap1_data, cmap="YlGnBu")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Length of Membership',
                     index=['Time on App'],
                     columns='Time on Website')
sns.heatmap(heatmap1_data, cmap="PiYG")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Yearly Amount Spent',
                     index=['Time on App'],
                     columns='Time on Website')
sns.heatmap(heatmap1_data, cmap="BuPu")
plt.figure()

heatmap1_data = pd.pivot_table(customers2, values='Avg. Session Length',
```

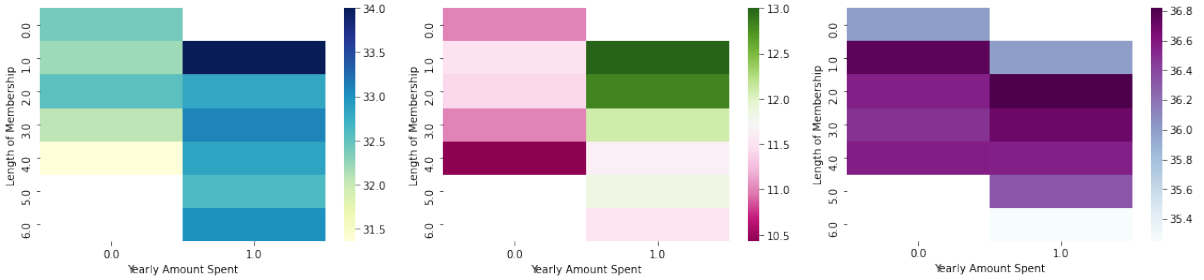
```
                    index=['Time on App'],
                    columns='Length of Membership')
sns.heatmap(heatmap1_data, cmap="YlGnBu")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Time on Website',
                    index=['Time on App'],
                    columns='Length of Membership')
sns.heatmap(heatmap1_data, cmap="PiYG")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Yearly Amount Spent',
                    index=['Time on App'],
                    columns='Length of Membership')
sns.heatmap(heatmap1_data, cmap="BuPu")
plt.figure()

heatmap1_data = pd.pivot_table(customers2, values='Avg. Session Length',
                    index=['Time on App'],
                    columns='Yearly Amount Spent')
sns.heatmap(heatmap1_data, cmap="YlGnBu")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Time on Website',
                    index=['Time on App'],
                    columns='Yearly Amount Spent')
sns.heatmap(heatmap1_data, cmap="PiYG")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Length of Membership',
                    index=['Time on App'],
                    columns='Yearly Amount Spent')
sns.heatmap(heatmap1_data, cmap="BuPu")
plt.figure()

heatmap1_data = pd.pivot_table(customers2, values='Avg. Session Length',
                    index=['Time on Website'],
                    columns='Length of Membership')
sns.heatmap(heatmap1_data, cmap="YlGnBu")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Time on App',
                    index=['Time on Website'],
                    columns='Length of Membership')
sns.heatmap(heatmap1_data, cmap="PiYG")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Yearly Amount Spent',
                    index=['Time on Website'],
                    columns='Length of Membership')
sns.heatmap(heatmap1_data, cmap="BuPu")
plt.figure()

heatmap1_data = pd.pivot_table(customers2, values='Avg. Session Length',
                    index=['Time on Website'],
                    columns='Yearly Amount Spent')
sns.heatmap(heatmap1_data, cmap="YlGnBu")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Time on App',
                    index=['Time on Website'],
                    columns='Yearly Amount Spent')
sns.heatmap(heatmap1_data, cmap="PiYG")
```

```
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Length of Membership',
                      index=['Time on Website'],
                      columns='Yearly Amount Spent')
sns.heatmap(heatmap1_data, cmap="BuPu")
plt.figure()

heatmap1_data = pd.pivot_table(customers2, values='Avg. Session Length',
                      index=['Length of Membership'],
                      columns='Yearly Amount Spent')
sns.heatmap(heatmap1_data, cmap="YlGnBu")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Time on App',
                      index=['Length of Membership'],
                      columns='Yearly Amount Spent')
sns.heatmap(heatmap1_data, cmap="PiYG")
plt.figure()
heatmap1_data = pd.pivot_table(customers2, values='Time on Website',
                      index=['Length of Membership'],
                      columns='Yearly Amount Spent')
sns.heatmap(heatmap1_data, cmap="BuPu")
plt.figure()
```

After trying all the combinations possible we will have all the heatmaps:
These are the plots obtained:




Avg. Session Length
Yearly Amount Spent
Time on App
Time on Website
Length of Membership



From the above heatmap we can see the good degree of correlation between 'length of membership' and 'Yearly amount spent'. Also between 'Yearly amount spent' and the column 'Time on app'. Also lesser degree of correlation with 'Avg. Session length'.

2.3. Task 3

First we will split the customers regarding the independent (x) and dependent (y) variables.

```
x = customers[['Time on App', 'Length of Membership']]
y = customers['Yearly Amount Spent']
```

Then we will train and test them with a test size of 0.3.

```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)
```

We will use this to create a Linear Regression Model lm and fit it.

```
lm = LinearRegression().fit(x_train, y_train)
```

Finally, we will use the model to predict for x_test and store the result.

```
result = lm.predict(x_test)
print('predicted response:', result, sep='\n')
```

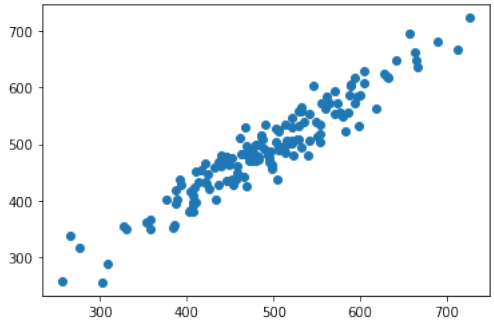
This is the predicted response obtained:
predicted response:
511.31521119 290.25371385 483.87063885 520.23424352 501.39859022
499.28586602 520.66205733 499.62275509 490.07916039 493.25259346
450.17486774 528.39260935 469.65049539 485.85279324 490.49142278
446.15826946 506.13222367 523.6602001 481.75235505 554.12407138
451.18038472 350.62014914 503.88432664 481.60877944 570.56813429
445.65330195 398.99599137 495.35165489 549.09310625 509.59274302
503.02453769 409.49602751 483.27503354 452.19867322 480.76768548
614.76436228 347.82388528 464.44292879 627.68181461 521.32156902
613.9075364 521.35116611 602.09306881 461.65701609 569.17099374
495.87055804 486.90483261 483.3988587 623.99283846 509.25721138
506.07052924 397.64498241 463.15507952 413.36132325 514.22403207
435.05993046 469.8967564 566.86515394 488.84509005 480.18065949
462.31049567 494.80579934 426.91238206 591.50274609 528.50114578
449.89281806 534.67665393 647.62719011 467.74156079 531.28051302
472.10560318 523.54993291 395.8136149 497.14062025 561.47296154
447.84174006 617.95933251 501.29359836 529.10385474 592.95130526
605.93315059 593.48494513 529.16669808 616.69759805 548.15200355
497.78028745 470.46990799 638.95363883 442.48433776 465.65369344
471.60315368 462.3487325 429.61013443 506.92022352 461.72010072
461.57460012 432.62159992 434.72755595 456.9753245 412.5922507
401.34858887 515.14901798 462.05211911 428.19223518 479.58798932
476.76712071 465.08209582 507.66707595 552.31949085 536.39416462
552.49976471 521.43647939 571.38586036 422.61343915 479.68036468

496.26055572 500.01146743 547.24777669 257.68027855 665.94923037
469.48465682 361.65645388 465.52119406 504.17153453 432.80786409
401.93624884 635.70716595 507.54255312 493.57194707 423.23241986
621.52986608 531.80000049 542.98342631 433.45878908 433.29574843
508.17260611 458.33941671 489.89750268 512.25822767 520.44264716
689.50330801 528.73570369 381.27427956 404.40160796 647.52524611
475.50243361 370.26278955 532.9320305 477.91958125 400.66517042

2.4. Task 4

We will plot a scatter plot for 'Actual Values' vs the 'Predicted Values':

```
plt.scatter(y_test, result)
```

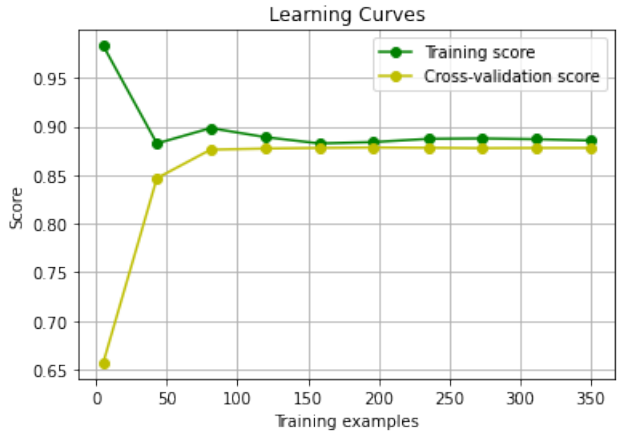


We will create the function plot_lc() to plot the learning curve based on the model developed and the x and y data. We will make use of the function learning_curve from sklearn and then we will name the different components of the plot.

```
def plot_lc(estimator, x, y, train_sizes):
    train_sizes, train_scores, test_scores = learning_curve(estimator, x, y, cv=5, n_jobs=1,
        train_sizes=train_sizes)
    train_scores_mean = np.mean(train_scores, axis=1)
    test_scores_mean = np.mean(test_scores, axis=1)
    plt.grid()
    plt.title("Learning Curves")
    plt.xlabel("Training examples")
    plt.ylabel("Score")
    plt.plot(train_sizes, train_scores_mean, 'o-', color="g", label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="y", label="Cross-validation score")
    plt.legend(loc="best")
    plt.show()
```

After that, we just have to call the function with the right parameters.

```
plot_lc(lm,x,y,np.linspace(5, len(x_train), 10, dtype='int'))
```



As we can see, the learning curve for the linear regression model shows a small gap between the training and validation error, so that variance should be reduced.

2.5. Task 5

Now we will use the functions r2_score, mean_squared_error and explained_variance_score from sklearn to print the R2 Score,Variance and MSE.

```
print("R2 score : %.2f" % r2_score(y_test,result))
print("Mean squared error: %.2f" % mean_squared_error(y_test,result))
v = explained_variance_score(y_test, result)
print ("Variance: %.2f" % v)
```

This is the output:
R2 score : 0.91
Mean squared error: 718.08
Variance: 0.91

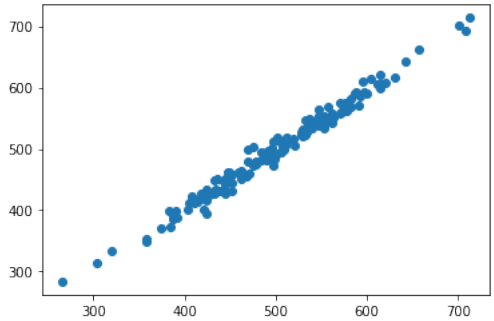
2.6. Task 6

Now we will consider 'Avg. Session Length' as well to build the new model 'lm1' and fit it.

```
x = customers[['Avg. Session Length', 'Time on App', 'Length of Membership']]
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)
lm1 = LinearRegression().fit(x_train, y_train)
```

We will also plot the scatter plot.

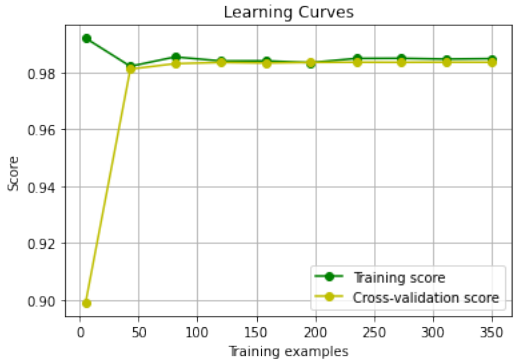
```
result = lm1.predict(x_test)
plt.scatter(y_test, result)
```



If we compare this with the other scatter plot we can see that the points are closer and the distribution is more uniform making a clearer line. That's thanks to the new independent variable.

We will also show the learning curve:

```
plot_lc(lm1,x,y,np.linspace(5, len(x_train), 10, dtype='int'))
```



Now the gap between the training and validation error has been reduced.

```
print("R2 score : %.2f" % r2_score(y_test,result))
print("Mean squared error: %.2f" % mean_squared_error(y_test,result))
v = explained_variance_score(y_test, result)
print ("Variance: %.2f" % v)
```

This is the new output:
R2 score : 0.98
Mean squared error: 103.12
Variance: 0.98

The R2 score and the variance have slightly decreased meanwhile the MSE has decreased a lot. That is what we were looking for when adding a new independent variable.