FIT1043 Introduction to Data Science Assignment 2

Diong Chen Xi 32722656

4th September 2021

Introduction

This is the start of my assignment to conduct predictive analytics through machine learning using Python in the Jupyter Notebook environment.

Question 1

1. Introduction

As an investor I would want to invest in people who showed a profile of having a high probability of paying me back. Here I will create a model to classify and predict whether or not the borrower paid back their loan in full.

First I start by importing the library **pandas**, which is an open source date analysis tool for the Python programming language. This library is used to access the data structure such as DataFrame and its functions to read files such as CSV, Excel, etc. Then, I import the library **matplotlib.pyplot** to use as a data visualizer later on in the assignment. I also included the magic line so that the plots created will be in line.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

After importing the library **pandas**, I now have access to the function which allows Python to read CSV files. After reading the specific file, I can store it as a data frame for future usage.

In [3]:
```
loan_data = pd.read_csv('Assignment 2Data/loan_data.csv')
loan_data.shape
```

Out[3]:
```
(9578, 13)
```

After checking, I now know that the file has 9578 rows and 13 columns.

In [4]:
```
loan_data.head()
```

Out[4]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.paid
0	1	0.1189	829.10	11.350407	19.48	737	5639.958333	28854	52.1	0	0	0	0
1	1	0.1071	228.22	11.082143	14.29	707	2760.000000	33623	76.7	0	0	0	0
2	1	0.1357	366.86	10.373491	11.63	682	4710.000000	3511	25.6	1	0	0	0
3	1	0.1008	162.34	11.350407	8.10	712	2699.958333	33667	73.2	1	0	0	0
4	1	0.1426	102.92	11.299732	14.97	667	4066.000000	4740	39.5	0	1	0	0

The above table displays the first five rows of data in the **loan_data.csv** file.

In [5]:
```
loan_data.tail()
```

Out[5]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.pai
9573	0	0.1461	344.76	12.180755	10.39	672	10474.000000	215372	82.1	2	0	0	
9574	0	0.1253	257.70	11.141862	0.21	722	4380.000000	184	1.1	5	0	0	
9575	0	0.1071	97.81	10.596635	13.09	687	3450.041667	10036	82.9	8	0	0	
9576	0	0.1600	351.58	10.819778	19.18	692	1800.000000	0	3.2	5	0	0	
9577	0	0.1392	853.43	11.264464	16.28	732	4740.000000	37879	57.0	6	0	0	

The above table displays the last five rows of data in the **loan_data.csv** file.

In [6]:
```
loan_data.sample(5)
```

Out[6]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delinq.2yrs	pub.rec	not.fully.pai
4205	1	0.1704	713.49	11.512925	12.65	702	4110.000000	6939	31.4	3	0	0	
8873	0	0.1568	350.02	11.225243	8.83	677	1050.000000	9700	36.7	2	0	0	
8091	0	0.1691	154.90	10.373491	27.15	652	3450.041667	2012	100.6	0	0	0	
6812	1	0.1496	519.70	10.518673	14.14	677	4067.000000	7186	55.7	1	0	0	
6815	1	0.1322	109.86	9.798127	23.53	707	1920.041667	12231	86.1	0	0	0	

The above table displays five random rows of data in the **loan_data.csv** file.

In [7]:
```
loan_data.dtypes
```

Out[7]:
```
credit.policy          int64
int.rate             float64
installment          float64
log.annual.inc       float64
dti                  float64
fico                   int64
days.with.cr.line    float64
revol.bal              int64
revol.util           float64
inq.last.6mths         int64
delinq.2yrs            int64
pub.rec                int64
not.fully.paid         int64
dtype: object
```

From the above dataset, we can know that it has 3 columns displaying true/false representative values (i.e. *credit.policy*, *pub.rec* and *not.fully.paid*); and 6 columns displaying float values (i.e. *int.rate*, *installment*, *log.annual.inc*, *dti*, *days.with.cr.line* and *revol.util*); and 4 columns displaying integer values (i.e. *fico*, *revol.bal*, *inq.last.6mths* and *delinq.2yrs*).

In [8]:
```
loan_data.describe()
```

Out[8]:

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delin
count	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9578.000000	9.578000e+03	9578.000000	9578.000000	9578.(

	credit.policy	int.rate	installment	log.annual.inc	dti	fico	days.with.cr.line	revol.bal	revol.util	inq.last.6mths	delin
mean	0.804970	0.122640	319.089413	10.932117	12.606679	710.846314	4560.767197	1.691396e+04	46.799236	1.577469	0.
std	0.396245	0.026847	207.071301	0.614813	6.883970	37.970537	2496.930377	3.375619e+04	29.014417	2.200245	0.
min	0.000000	0.060000	15.670000	7.547502	0.000000	612.000000	178.958333	0.000000e+00	0.000000	0.000000	0.
25%	1.000000	0.103900	163.770000	10.558414	7.212500	682.000000	2820.000000	3.187000e+03	22.600000	0.000000	0.
50%	1.000000	0.122100	268.950000	10.928884	12.665000	707.000000	4139.958333	8.596000e+03	46.300000	1.000000	0.
75%	1.000000	0.140700	432.762500	11.291293	17.950000	737.000000	5730.000000	1.824950e+04	70.900000	2.000000	0.
max	1.000000	0.216400	940.140000	14.528354	29.960000	827.000000	17639.958330	1.207359e+06	119.000000	33.000000	13.

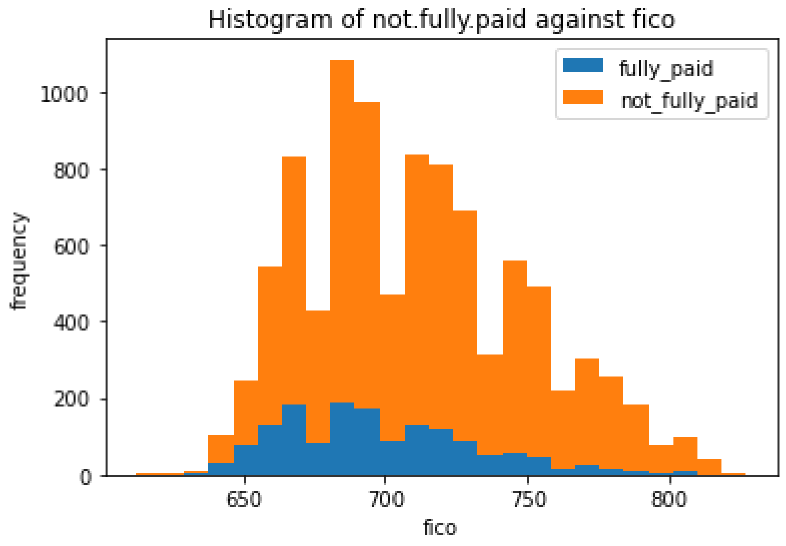
We can also interpret that there are small fluctuations in the interest rates and the credit scores among the borrowers, whereas there are big fluctuations in the installments and revolving balances among the borrowers.

We can also derive that the borrowers have had a credit line of quite a long time overall, and that not much of the borrowers had been 30+ days past due on a payment in the past 2 years nor have derogatory public records.

Next, I create a histogram of the not.fully.paid column on top of each other, one for each not.fully.paid outcome (0 and 1) as per their credit score fico.

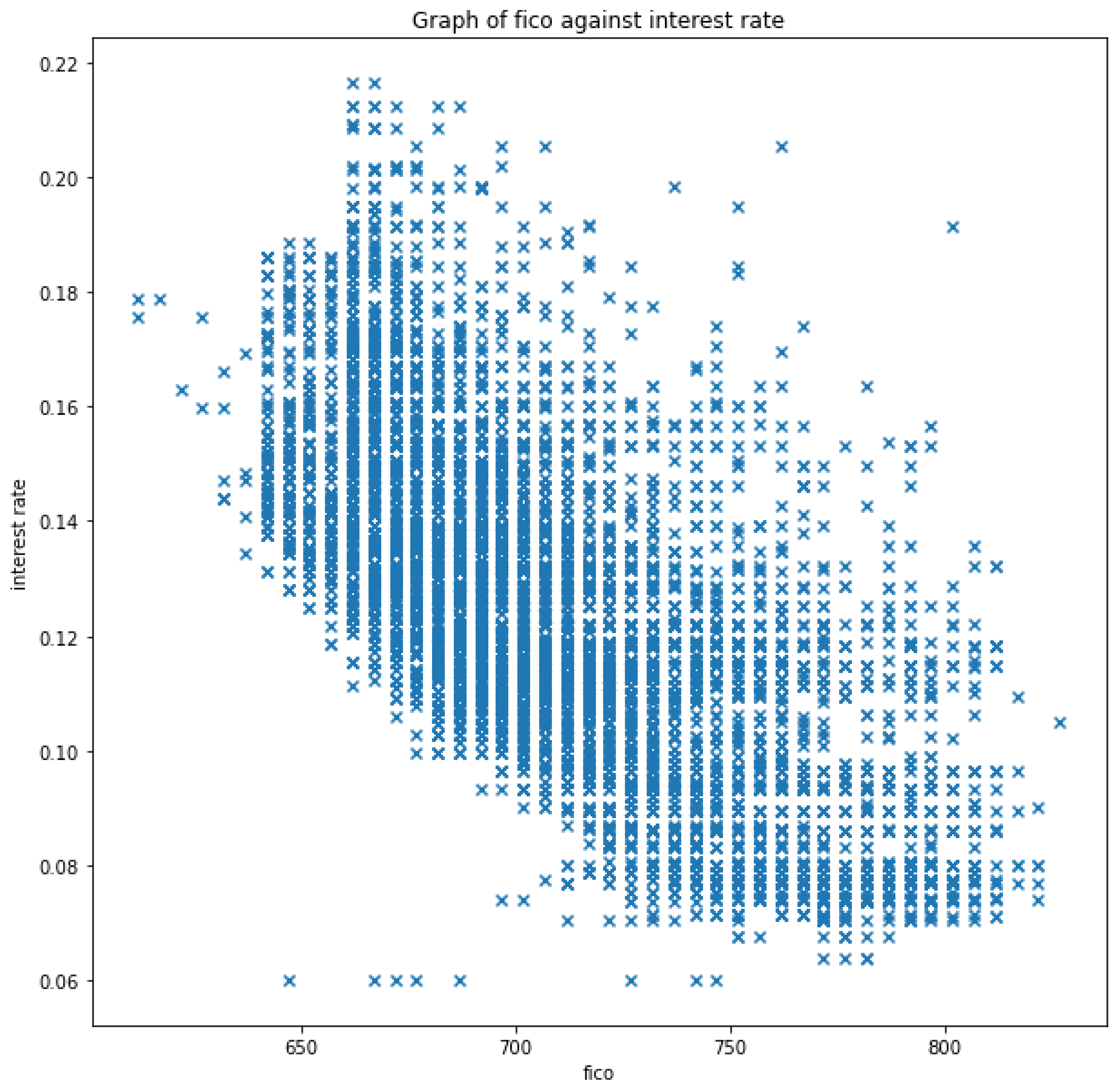
In [9]:
```
fully_paid = loan_data[loan_data['not.fully.paid'] == 1]
not_fully_paid = loan_data[loan_data['not.fully.paid'] == 0]
hist_data = [fully_paid['fico'],not_fully_paid['fico']]
```

In [10]:
```
plt.figure
plt.hist(hist_data, bins = 25, stacked=True)
plt.xlabel('fico')
plt.ylabel('frequency')
plt.title('Histogram of not.fully.paid against fico')
plt.legend(('fully_paid','not_fully_paid'),loc = 'best')
plt.show()
```



Then, I create a scatter plot to show the relationship between fico and interest rate. We can see that as the fico increases, the interest rate has a tendency of decreasing.

```
In [11]:
x = loan_data['fico']
y = loan_data['int.rate']
plt.figure(figsize=[10,10])
plt.scatter(x,y,marker ='x')
plt.title('Graph of fico against interest rate')
plt.xlabel('fico')
plt.ylabel('interest rate')
plt.show()
```



2. Supervised Learning

Supervised machine learning: A method of training algorithms to predict a certain outcome. The machine is fed a set of inputs with their corresponding outputs, and the machine adjusts its algorithm to fit the current data presented. After that, the machine is tested using a pre-divided test dataset to determine its accuracy. Ultimately, a refined algorithm is created to help solve problems.

The notion of labelled data: Data is labelled to introduced expected outcome to a machine. It provides context to the machine so that it can derive a certain expected result from the given raw data. Labelled data serves as an "actual truth" for the machine to base its learning outcome from.

The training and test datasets: Data is split into two different datasets, one for training and one for testing. The training dataset is introduced to the machine as inputs with corresponding outputs to help the machine learn and derive an algorithm for a certain problem. The testing dataset is then introduced to evaluate the accuracy of the machine's current algorithm.

Features: credit.policy, int.rate, installment, log.annual.inc, dti, fico, days.with.cr.line, revol.ba, revol.util, inq.last.6mths, delinq.2yrs, pub.rec

Label: not.fully.paid

In [12]:
```
"""
Features: credit policy, interest rate, installment, annual income, debt-to-income ratio, credit score, the number of days the
borrower has a credit line, revolving balance, revolving line utilization rate, number of inquiries in the last 6 months, the
number of times the borrower had been 30+ days past due on a payment in the past 2 years, number of derogatory public records.
"""
X = loan_data.iloc[:, [0,1,2,3,4,5,6,7,8,9,10,11]].values
"""
Label: Whether the customer has paid back their loan in full.
"""
y = loan_data.iloc[:, 12].values
```

Here I use the sklearn.model_selection.train_test_split function to split my data for training (80 %) and testing (20%).

In [13]:
```
from sklearn.model_selection import train_test_split
```

In [14]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

3. Classification

Binary classification: The task to classify objects into two discrete classes(i.e. dividing data into 2 different groups). For example, the task of sorting images into dogs and cats is a type of binary classification. The images are either classified into dogs or cats.

Multi-class classification: The task to classify objects into more than two discrete classes(i.e. dividing data into more than 2 different groups). For example, the task of sorting different types of fish is a type of multi-class classification. The fish can be sorted into a variety of classes such as tuna, salmon, sardine, etc.

This current problem is a **binary classification** where customers are grouped either into the *will fully pay back their loan* or *will not fully pay back their loan*.

Since the range of values of raw data varies widely, if one of the features has a broad range of values, the outcome will be heavily influenced by this particular feature. Therefore, the range of all features should be normalized in order for each feature to contribute proportionately to the final result. Here I import the StandardScaler function from the sklearn.preprocessing library to aid me with the normalization of data.

In [15]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Training and testing using Decision Tree Algorithm

Here I import the DecisionTreeClassifier function from the sklearn.tree library to conduct training and testing of the dataset using the Decision Tree Algorithm.

In [16]:
```
# Fitting Decision Tree Classification to the Training set
from sklearn.tree import DecisionTreeClassifier
dtclassifier = DecisionTreeClassifier(
 criterion = 'entropy', random_state = 0
)
dtclassifier.fit(X_train, y_train)
```

Out[16]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

I now have a model using the Decision Tree algorithm and the model is called *dtclassifier*. Now, I will conduct the prediction for the label (whether borrowers will pay back or not) using the testing dataset I created above.

In [17]:
```
# Predicting the Test set results
y_pred = dtclassifier.predict(X_test)
```

I now have a set of labels, *y_pred*, that is the output from the prediction of using the testing data. I will now compare this *y_pred* with the actual y_test (the true value) and determine the accuracy of my model's prediction by creating a confusion matrix for visualisation. To do that I import the confusion_matrix function from the sklearn.metrics library.

In [18]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

As the current confusion matrix is displayed as an array, I converted it into a dataframe for better visualisation.

In [19]:
```
pd.DataFrame(cm)
```

Out[19]:

	0	1
0	1378	237
1	221	80

From the confusion matrix, we can see that it has a high number of True Negatives and a small number of True Positives. It also has a fair amount of False Negatives and False Positives.

From there we can derive that this Decision Tree model can predict people who would not pay back their loan accurately, but it has low accuracy on predicting people who would pay back their loan.

Training and testing using the Random Forest Algorithm

Here I import the RandomForestClassifier function from the sklearn.ensemble library to conduct training and testing of the dataset using the Random Forest Algorithm.

In [20]:
```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
rfclassifier = RandomForestClassifier(
    n_estimators = 40,
    criterion = 'entropy',
    random_state = 0
)
rfclassifier.fit(X_train, y_train)
```

```
Out[20]: RandomForestClassifier(criterion='entropy', n_estimators=40, random_state=0)
```

I now have a model using the Random Forest algorithm and the model is called *rfclassifier*. Now, I will conduct the prediction for the label (whether borrowers will pay back or not) using the testing dataset I created above.

```
In [21]: # Predicting the Test set results
         y_pred2 = rfclassifier.predict(X_test)
```

I now have a set of labels, *y_pred2*, that is the output from the prediction of using the testing data. I will now compare this *y_pred2* with the actual y_test (the true value) and determine the accuracy of my model's prediction by creating a confusion matrix for visualisation.

```
In [22]: cm2 = confusion_matrix(y_test, y_pred2)
```

As the current confusion matrix is displayed as an array, I converted it into a dataframe for better visualisation.

```
In [23]: pd.DataFrame(cm2)
```

Out[23]:

	0	**1**
0	1604	11
1	292	9

From the confusion matrix, we can see that it has a high number of True Negatives and a very small number of True Positives and False Negatives. It also has a fair amount of False Positives.

From there we can derive that this Random Forest model can predict people who would not pay back their loan very accurately, but it has almost zero accuracy on predicting people who would pay back their loan.

4. Conclusion

Comparing the performance of Decision Tree Algorithm and Random Forest Algorithm

Here I define some functions to aid with my analysis of the performance between the two different algorithms.

In [24]:
```
def evaluate_cm(cm):
    """
    Converts the values inside the confusion matrix into True Negative(TN), True Positive(TP), False Negative(FN) and
    False Positive(FP).
    """
    TN = cm[0][0]
    TP = cm[1][1]
    FN = cm[1][0]
    FP = cm[0][1]

    return {
        'TN': TN,
        'TP': TP,
        'FN': FN,
        'FP': FP
    }

def evaluate_precision(cm):
    """
    When you predict something positive, how many times they were actually positive.
    """
    vals = evaluate_cm(cm)
    TP = vals['TP']
    FP = vals['FP']
    return TP / (TP + FP)

def evaluate_recall(cm):
    """
    Out of actual positive data, how many times you predicted correctly.
    """
    vals = evaluate_cm(cm)
    TP = vals['TP']
    FN = vals['FN']
    return TP / (TP + FN)

def evaluate_accuracy(cm):
    """
    Out of the overall data, how many predictions were correct.
    """
    vals = evaluate_cm(cm)

    TN, TP, FN, FP = vals.values()

    return (TP+TN)/(TP+TN+FP+FN)
```

Here I declare a dictionary called *scores* to store the analysis of the two algorithms.

In [25]:
```
scores = {}
```

In [26]:
```
scores['DT'] = {
    'cm':cm,
    'precision':evaluate_precision(cm),
    'recall':evaluate_recall(cm),
    'accuracy':evaluate_accuracy(cm)
}
```

In [27]:
```
scores['RF'] = {
    'cm':cm2,
    'precision':evaluate_precision(cm2),
    'recall':evaluate_recall(cm2),
    'accuracy':evaluate_accuracy(cm2)
}
```

In [28]:
```
pd.DataFrame(scores).T
```

Out[28]:

	cm	precision	recall	accuracy
DT	[[1378, 237], [221, 80]]	0.252366	0.265781	0.76096
RF	[[1604, 11], [292, 9]]	0.45	0.0299	0.841858

From the data frame we can see that the Random Forest Algorithm has higher precision and accuracy compared to the Decision Tree Algorithm. However, we can see that the the Decision Tree Algorithm has higher recall than the Random Forest Algorithm. In my opinion, I think that the Decision Tree Algorithm is performing better as it makes more balanced accurate predictions of True Positives and True Negatives than the Random Forest Algorithm.
This concludes the end of Question 1.

Question 2

1. Introduction

An Ecommerce company that is selling clothing online but also have in-store style and clothing advice sessions is trying to decide whether to focus their efforts on their mobile app experience or their website. Here I will try to figure which is the best direction for them to focus on.

First I start by importing the library **pandas**, which is an open source date analysis tool for the Python programming language. This library is used to access the data structure such as DataFrame and its functions to read files such as CSV, Excel, etc. Then, I import the library **matplotlib.pyplot** to use as a data visualizer later on in the assignment. I also included the magic line so that the plots created will be in line.

In [29]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

After importing the library **pandas**, I now have access to the function which allows Python to read CSV files. After reading the specific file, I can store it as a data frame for future usage.

In [31]:
```
customers_shop = pd.read_csv('Assignment 2Data/customers-shop.csv')
customers_shop.shape
```

Out[31]:
```
(500, 6)
```

After checking, I now know that the file has 500 rows and 6 columns.

In [32]:
```
customers_shop.head()
```

Out[32]:

	Customer info-color Avatar	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
0	Violet	34.497268	12.655651	39.577668	4.082621	587.951054
1	DarkGreen	31.926272	11.109461	37.268959	2.664034	392.204933
2	Bisque	33.000915	11.330278	37.110597	4.104543	487.547505
3	SaddleBrown	34.305557	13.717514	36.721283	3.120179	581.852344
4	MediumAquaMarine	33.330673	12.795189	37.536653	4.446308	599.406092

The above table displays the first five rows of data in the **customers-shop.csv** file.

In [33]:
```
customers_shop.tail()
```

Out[33]:

	Customer info-color Avatar	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
495	Tan	33.237660	13.566160	36.417985	3.746573	573.847438
496	PaleVioletRed	34.702529	11.695736	37.190268	3.576526	529.049004
497	Cornsilk	32.646777	11.499409	38.332576	4.958264	551.620146
498	Teal	33.322501	12.391423	36.840086	2.336485	456.469510
499	DarkMagenta	33.715981	12.418808	35.771016	2.735160	497.778642

The above table displays the last five rows of data in the **customers-shop.csv** file.

In [34]:
```
customers_shop.sample(5)
```

Out[34]:

	Customer info-color Avatar	Avg. Session Length	Time on App	Time on Website	Length of Membership	Yearly Amount Spent
138	Sienna	33.547748	10.735363	37.458375	3.863425	476.191413
219	LightGreen	31.736636	10.748534	35.738707	4.835529	496.933446
256	HotPink	34.379394	12.930929	36.360247	3.792712	574.654843
72	Teal	32.386252	10.674653	38.006583	3.401522	418.150081
22	Olive	31.531604	13.378563	38.734006	2.245148	436.515606

The above table displays five random rows of data in the **customers-shop.csv** file.

In [35]:
```
customers_shop.dtypes
```

Out[35]:
```
Customer info-color Avatar     object
Avg. Session Length           float64
Time on App                   float64
Time on Website               float64
Length of Membership          float64
```

```
Yearly Amount Spent           float64
dtype: object
```

From the above dataset, we can know that it has 1 column displaying the color of the customer's Avatar (i.e. *Customer info-color Avatar*; and 5 columns displaying float values (i.e. *Avg. Session Length*, *Time on App*, *Time on Website*, *Length of Membership* and *Yearly Amount Spent*).

In [36]:

```
customers_shop.describe()
```

Out[36]:

	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

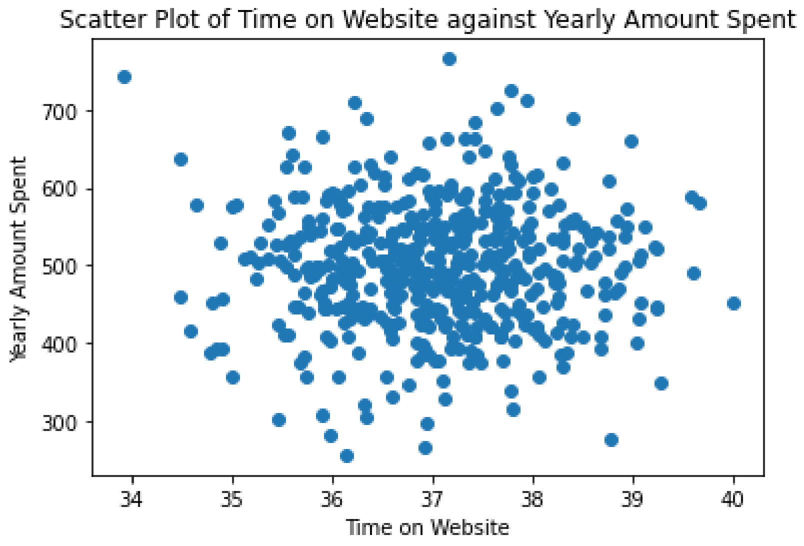
We can interpret that there are small fluctuations in the average in-store advice sessions, the time customers spent on App and the time customers spent on Website.

We can also derive that the shop has been up and running for quite a long time (at least 6 years) and it has been gaining new customers with membership as well.

It is also shown that the shop has a decent amount of yearly income from the mean yearly amount spent of its customers.

In [37]:

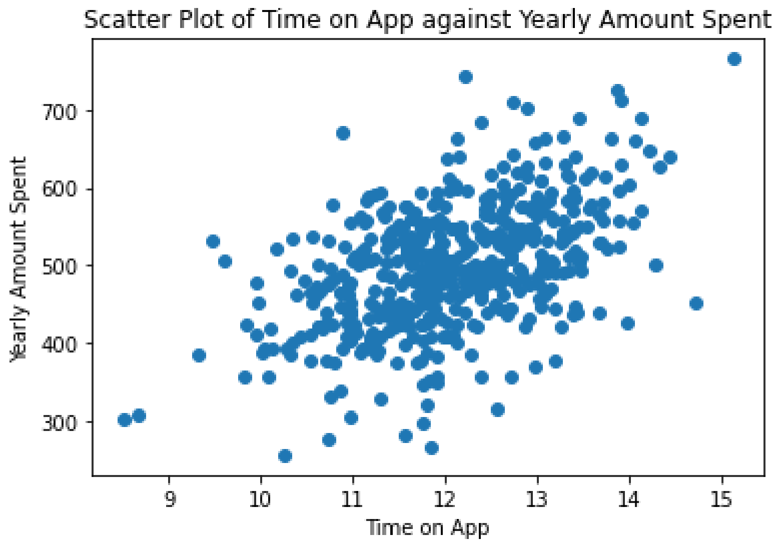
```
x1 = customers_shop['Time on Website']
y1 = customers_shop['Yearly Amount Spent']
plt.scatter(x1,y1)
plt.title('Scatter Plot of Time on Website against Yearly Amount Spent')
plt.xlabel('Time on Website')
plt.ylabel('Yearly Amount Spent')
plt.show()
```



From the scatter plot we can see that the correlation between the time spent on the website and the yearly amount spent by customers do not make much sense. The customer having more time spent on the website does not necessarily indicate that the customer has spent more than a customer having less time spent on the website.

In [38]:

```
x2 = customers_shop['Time on App']
y2 = customers_shop['Yearly Amount Spent']
plt.scatter(x2,y2)
plt.title('Scatter Plot of Time on App against Yearly Amount Spent')
plt.xlabel('Time on App')
plt.ylabel('Yearly Amount Spent')
plt.show()
```



From the scatter plot we can see that the correlation between the time spent on the app and the yearly amount spent by customers makes more sense. The customer having more time spent on the app has an inclination of having a higher yearly amount spent compared to a customer having less time spent on the app.

2. Supervised Learning

In [39]:
```
"""
Features: Time spent on app by the customer, time spent on website by the customer.
"""
X2_app = customers_shop.iloc[:, 2].values
X2_website = customers_shop.iloc[:, 3].values
"""
Label: Yearly amount spent by the customer
"""
y2 = customers_shop.iloc[:, 5].values
```

Here I use the sklearn.model_selection.train_test_split function to split my data for training (70 %) and testing (30%).

The training dataset is used to train the Linear Regression model to learn the patterns of the given dataset and build a model for future predictions. The testing dataset is used to determine the accuracy of the current model to see if it will make accurate predictions. (i.e. it will be a useful prediction model)

In [40]:
```
X_train2, X_test2, y_train2, y_test2 = train_test_split(X2_app, y2, test_size = 0.3, random_state = 0)
X_train3, X_test3, y_train3, y_test3 = train_test_split(X2_website, y2, test_size = 0.3, random_state = 0)
```

In [41]:
```
X_train2 = X_train2.reshape(-1,1)
X_test2 = X_test2.reshape(-1,1)
X_train3 = X_train3.reshape(-1,1)
X_test3 = X_test3.reshape(-1,1)
```

3. Regression

Here I import the LinearRegression function from the sklearn.linear_model library to conduct training and testing of the dataset using Linear Regression. I also imported the numpy library for the LinearRegression function. I also import the mean_squared_error and mean_absolute_error functions from the sklearn.metrics library to calculate the accuracy metrics for the linear regression models later on.

In [42]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
import numpy as np
from math import sqrt
```

In [43]:
```
LR = LinearRegression()
LR.fit(X_train2,y_train2)
```

Out[43]:
```
LinearRegression()
```

In [44]:
```
LR.coef_
```

Out[44]:
```
array([39.1629381])
```

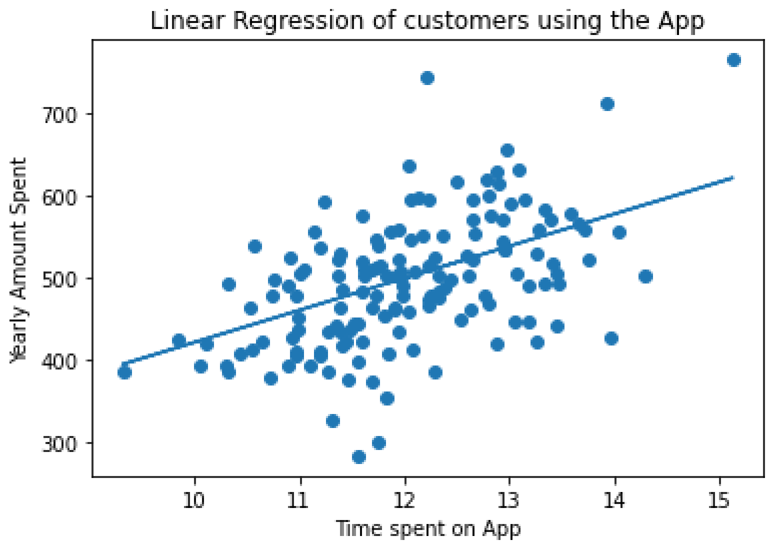
From here we can see that the coefficient of the regression model of time spent using the app is 39.1629381.

Then, the prediction is carried out using the test data on the Linear Regression model that is just trained.

In [45]:
```
#use model to predict the test data
y_prediction = LR.predict(X_test2)
```

In [46]:
```
plt.scatter(X_test2,y_test2)
plt.plot(X_test2,y_prediction)
plt.xlabel('Time spent on App')
plt.ylabel('Yearly Amount Spent')
plt.title('Linear Regression of customers using the App')
```

Out[46]:
```
Text(0.5, 1.0, 'Linear Regression of customers using the App')
```



In [47]:
```
LR.fit(X_train3,y_train3)
```

Out[47]:
```
LinearRegression()
```

In [48]:
```
LR.coef_
```

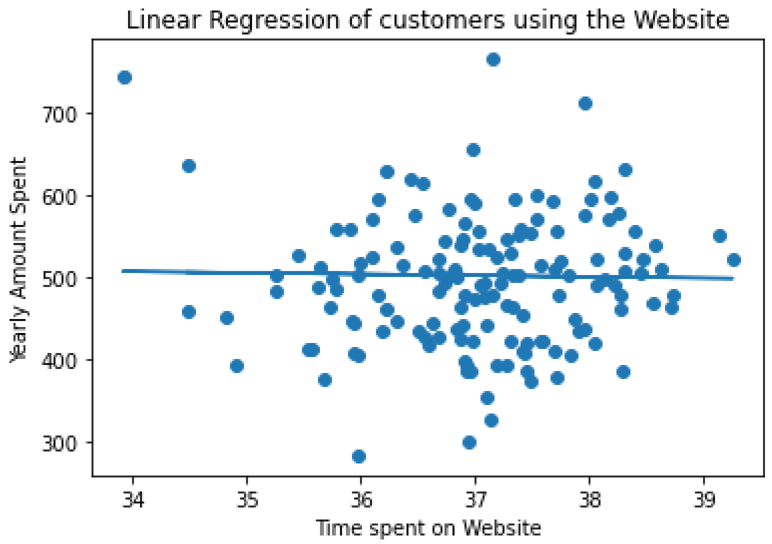
Out[48]:
```
array([-1.58457927])
```

From here we can see that the coefficient of the regression model of time spent using the website is -1.58457927.

In [49]:
```
#use model to predict the test data
y_prediction2 = LR.predict(X_test3)
```

In [50]:
```
plt.scatter(X_test3,y_test3)
plt.plot(X_test3,y_prediction2)
plt.xlabel('Time spent on Website')
plt.ylabel('Yearly Amount Spent')
plt.title('Linear Regression of customers using the Website')
```

Out[50]:
```
Text(0.5, 1.0, 'Linear Regression of customers using the Website')
```



Now, I will calculate the accuracy metrics for the two different linear regression models. I calculated the root mean squared error and the mean absolute error of the two models. The root mean squared error tells me how close my model is to the line of best fit, whereas the mean absolute error tells me the amount of error that is present in the predictions.

In [51]:
```
#accuracy metrics for linear regression model of time spent on app
print(math.sqrt(mean_squared_error(y_test2,y_prediction)))
print(mean_absolute_error(y_test2,y_prediction))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-51-845002720fc7> in <module>
      1 #accuracy metrics for linear regression model of time spent on app
----> 2 print(math.sqrt(mean_squared_error(y_test2,y_prediction)))
      3 print(mean_absolute_error(y_test2,y_prediction))

NameError: name 'math' is not defined
```

```
In [ ]: #accuracy metrics for linear regression model of time spent on website
        print(math.sqrt(mean_squared_error(y_test3,y_prediction2)))
        print(mean_absolute_error(y_test3,y_prediction2))
```

We can see that the linear regression model of time spent on app has a smaller root mean squared error and mean absolute error compared to the linear regression model of time spent on website.

A smaller root mean squared error indicates that the linear regression model of time spent on app has a higher accuracy and a smaller mean absolute error indicates that the linear regression model of time spent on the app has a smaller margin of error compared to the linear regression model of the time spent on website.

4. Conclusion

From the graphs and the accuracy metrics, we can see that the amount of time spent on the app is more important than the amount of time spent on the website for increasing the yearly amount spent. I think that it is better for the company to focus their efforts on their mobile app experience.
This concludes the end of Question 2.

Binary and multi-class classification referenced from https://www.analyticssteps.com/blogs/binary-and-multiclass-classification-machine-learning
Data labelling referenced from https://aws.amazon.com/sagemaker/groundtruth/what-is-data-labeling/
Supervised machine learning referenced from https://www.ibm.com/cloud/learn/supervised-learning
Stacked histogram referenced from https://stackoverflow.com/questions/18449602/create-stacked-histogram-from-unequal-length-arrays