Customer Status Prediction

Objectives

ExtraaLearn is an initial stage startup that offers programs on cutting-edge technologies to students and professionals to help them upskill/reskill. With a large number of leads being generated on a regular basis, one of the issues faced by ExtraaLearn is to identify which of the leads are more likely to convert so that they can allocate resources accordingly. You, as a data scientist at ExtraaLearn, have been provided the leads data to:

Analyze and build an ML model to help identify which leads are more likely to convert to paid customers. Find the factors driving the lead conversion process. Create a profile of the leads which are likely to convert

1. Importing all the required libraries

- All the important and required dependencies for this project will be imported first

In [143]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import plot_confusion_matrix
```

In [2]:

```
# Loading the dataset
data = pd.read_csv('data.csv')
data.head()
```

Out[2]:

bsite_visits	time_spent_on_website	page_views_per_visit	last_activity	print_media_type1	print_medi
7	1639	1.861	Website Activity	Yes	
2	83	0.320	Website Activity	No	
3	330	0.074	Website Activity	No	
4	464	2.057	Website Activity	No	
4	600	16.914	Email Activity	No	

The ID column will be dropped because it will not make any sense for the machine learning model

In [7]:

```
data = data.drop('ID', axis = 1)
data.head()
```

Out[7]:

	age	current_occupation	first_interaction	profile_completed	website_visits	time_spent_on_websi
0	57	Unemployed	Website	High	7	16
1	56	Professional	Mobile App	Medium	2	
2	52	Professional	Website	Medium	3	3
3	53	Unemployed	Website	High	4	4
4	23	Student	Website	High	4	6

2. Statistical analysis of the data

In [9]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4612 entries, 0 to 4611
Data columns (total 14 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   age                    4612 non-null   int64  
 1   current_occupation     4612 non-null   object 
 2   first_interaction      4612 non-null   object 
 3   profile_completed      4612 non-null   object 
 4   website_visits         4612 non-null   int64  
 5   time_spent_on_website  4612 non-null   int64  
 6   page_views_per_visit   4612 non-null   float64
 7   last_activity          4612 non-null   object 
 8   print_media_type1      4612 non-null   object 
 9   print_media_type2      4612 non-null   object 
 10  digital_media          4612 non-null   object 
 11  educational_channels   4612 non-null   object 
 12  referral               4612 non-null   object 
 13  status                 4612 non-null   int64  
dtypes: float64(1), int64(4), object(9)
memory usage: 504.6+ KB
```

In [10]:
```
data.describe()
```

Out[10]:

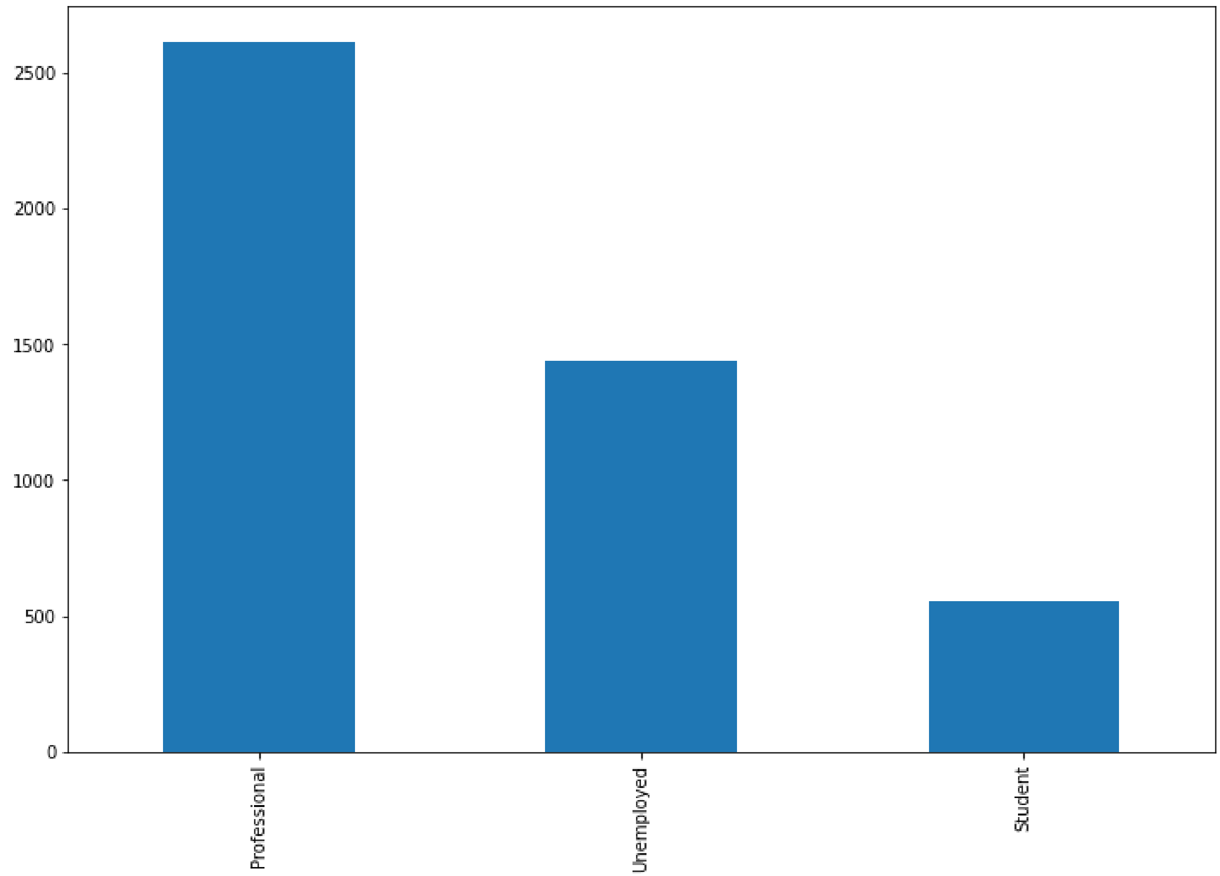
	age	website_visits	time_spent_on_website	page_views_per_visit	status
count	4612.000000	4612.000000	4612.000000	4612.000000	4612.000000
mean	46.201214	3.566782	724.011275	3.026126	0.298569
std	13.161454	2.829134	743.828683	1.968125	0.457680
min	18.000000	0.000000	0.000000	0.000000	0.000000
25%	36.000000	2.000000	148.750000	2.077750	0.000000
50%	51.000000	3.000000	376.000000	2.792000	0.000000
75%	57.000000	5.000000	1336.750000	3.756250	1.000000
max	63.000000	30.000000	2537.000000	18.434000	1.000000

3. Data Exploration

We will explore the whole dataset to see the hidden patterns and insights

- Let's see the occupation of the customers

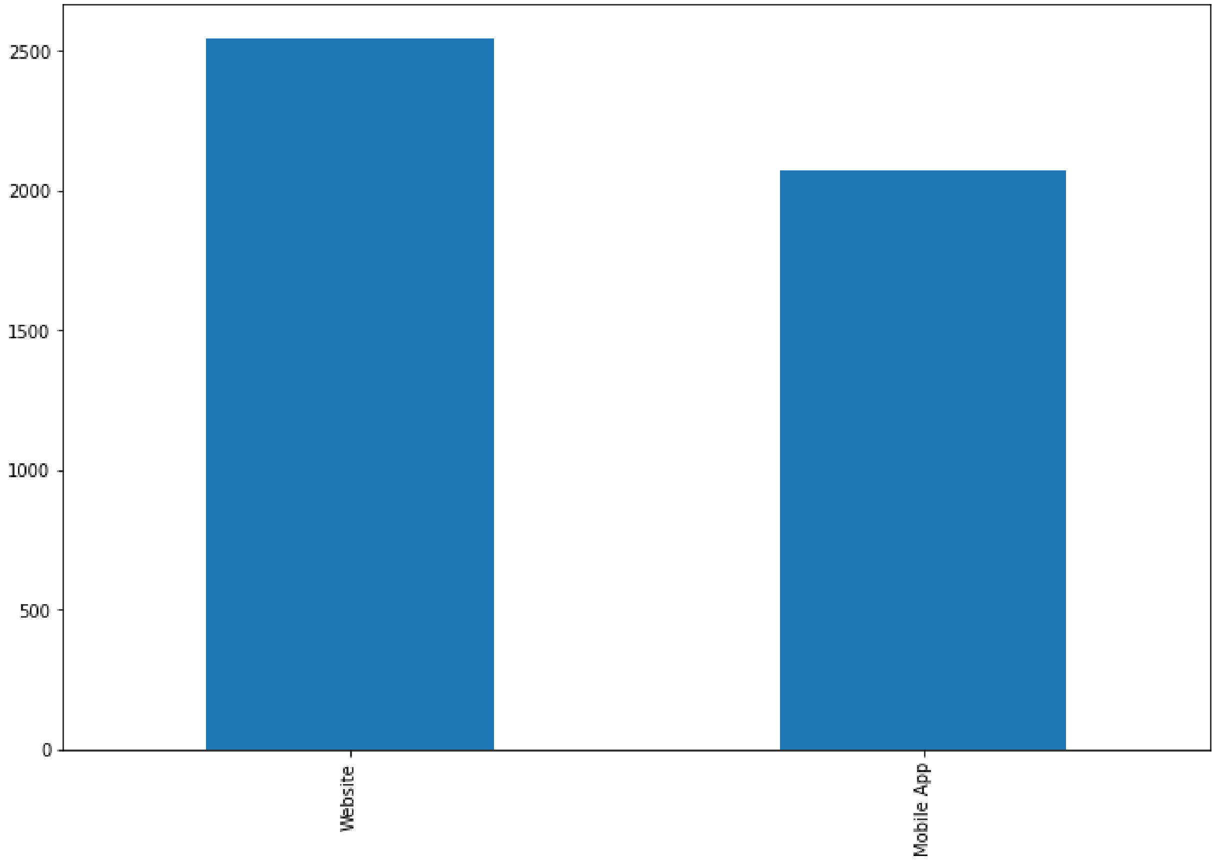
In [36]:
```
data["current_occupation"].value_counts().plot.bar(figsize=(12,8))
plt.show()
```



We can see that we have huge number of customers with the current accupation as Professional

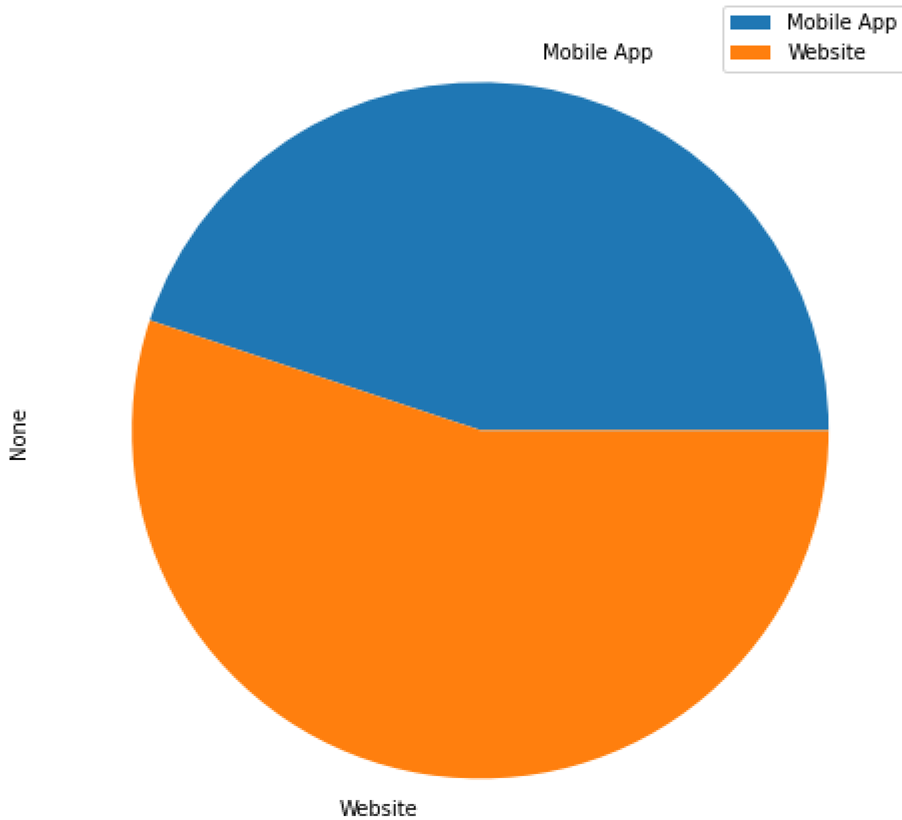
- Lets see how many customers are coming from mobile application and website

In [37]:
```
data["first_interaction"].value_counts().plot.bar(figsize=(12,8))
plt.show()
```



In [38]:
```
data.groupby('first_interaction').size().plot(kind='pie', legend=True, figsize=(1
```

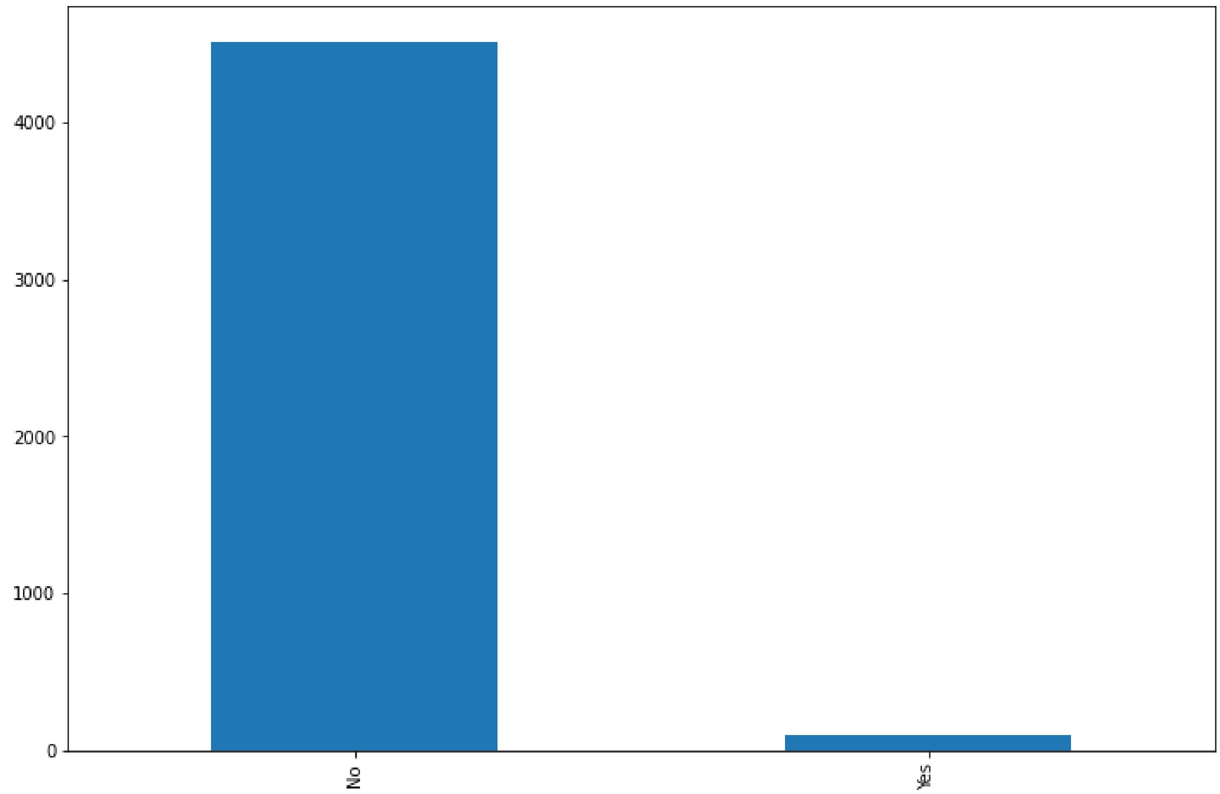
Out[38]: <AxesSubplot:ylabel='None'>



We can see that the number of visitors from website are higher than mobile application

- Now lets see how many customers came through referral and how many are not

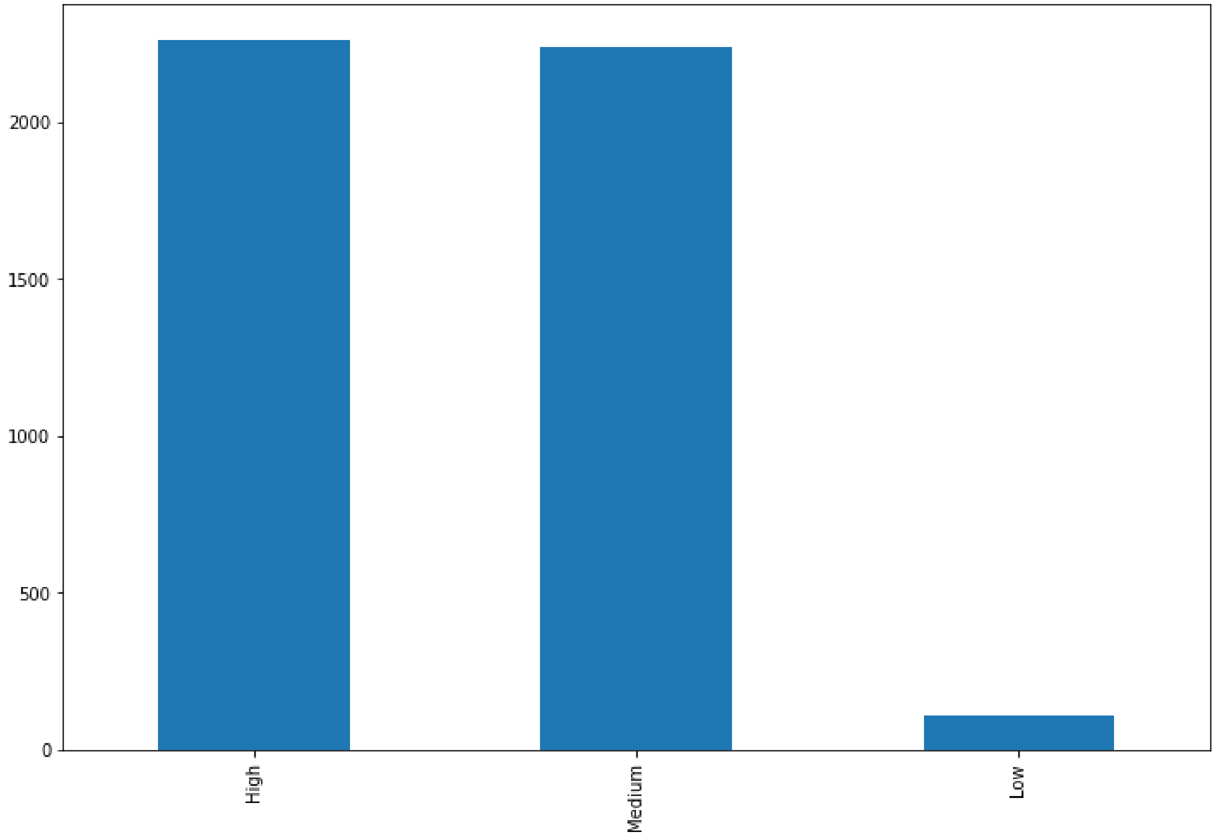
In [39]:
```
data["referral"].value_counts().plot.bar(figsize=(12,8))
plt.show()
```



The above graph shows that only some customers came by referral link

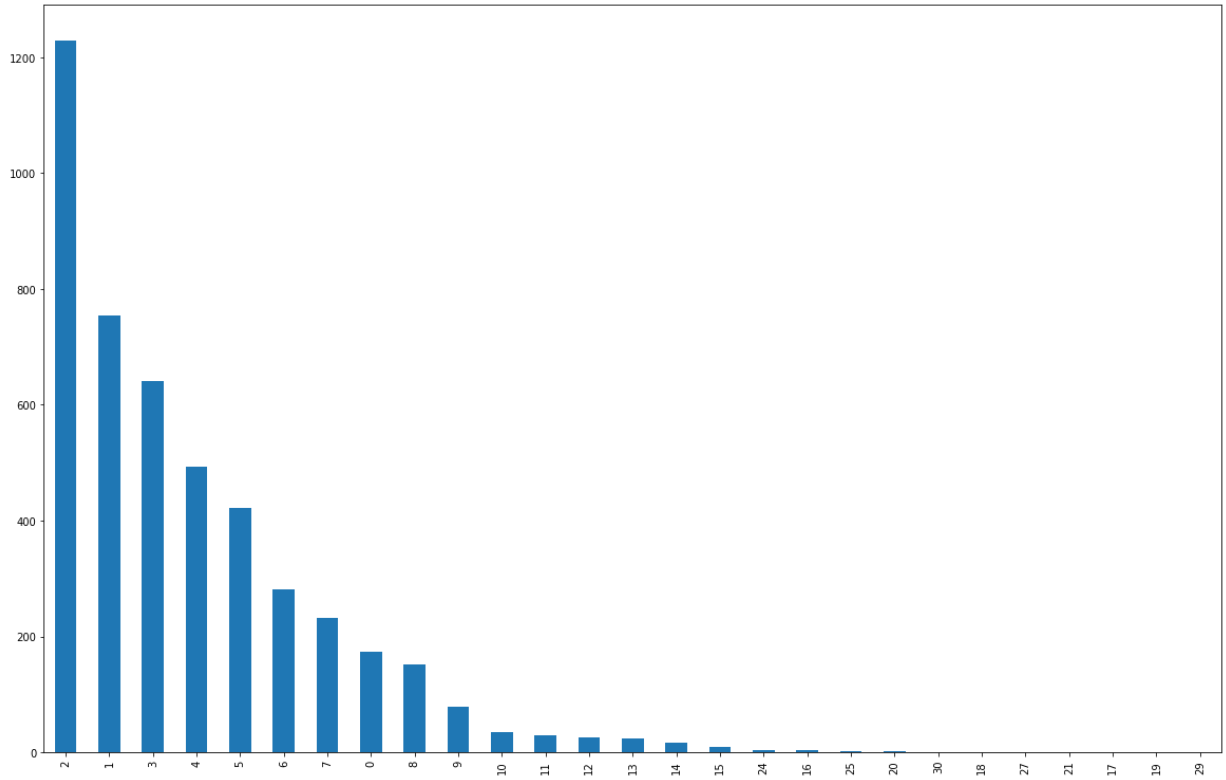
- Lets check What percentage of profile has been filled by the lead on the website/mobile app.

In [41]:
```
data["profile_completed"].value_counts().plot.bar(figsize=(12,8))
plt.show()
```



- Lets check how many time a lead came back and visited again

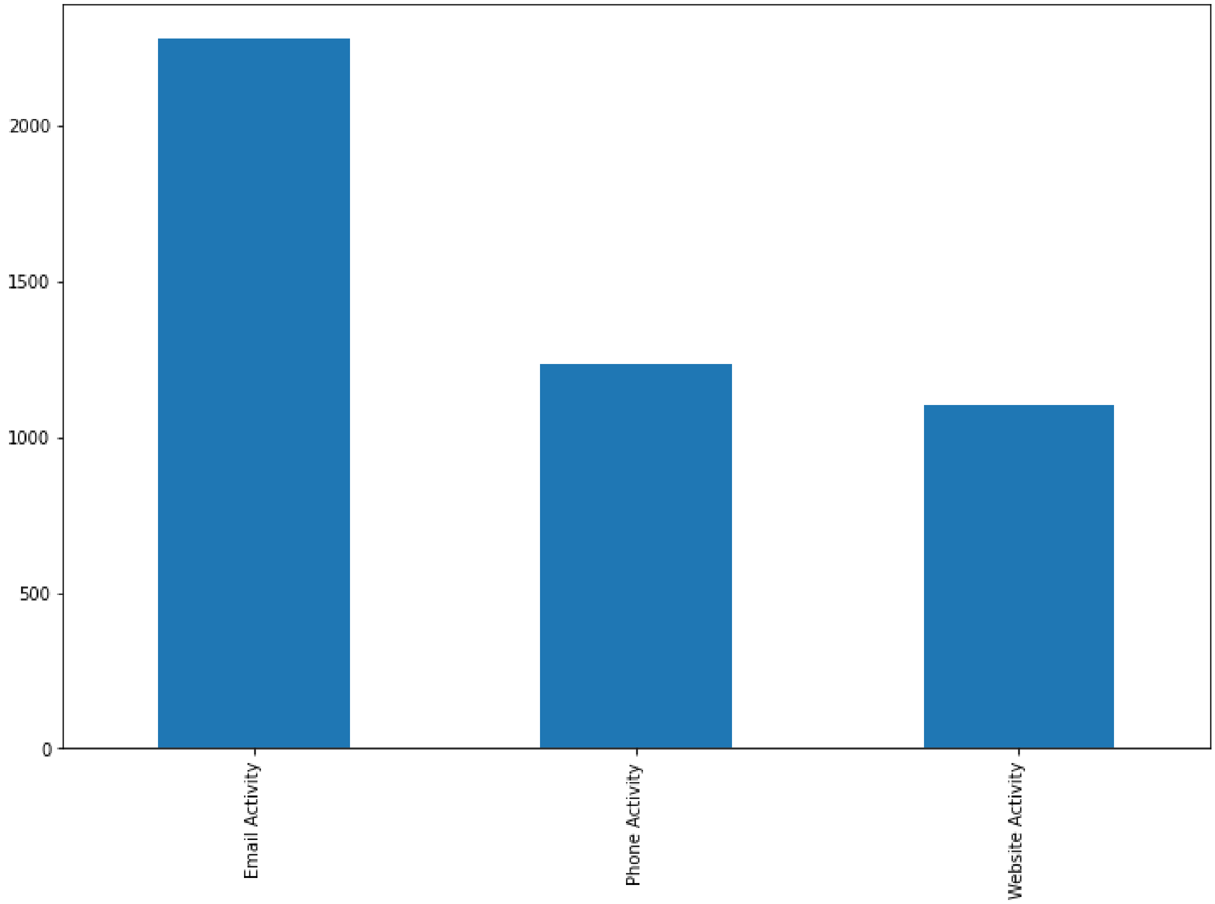
In [44]:
```
data["website_visits"].value_counts().plot.bar(figsize=(20,13))
plt.show()
```



we can see that small amount of leads came and visited again

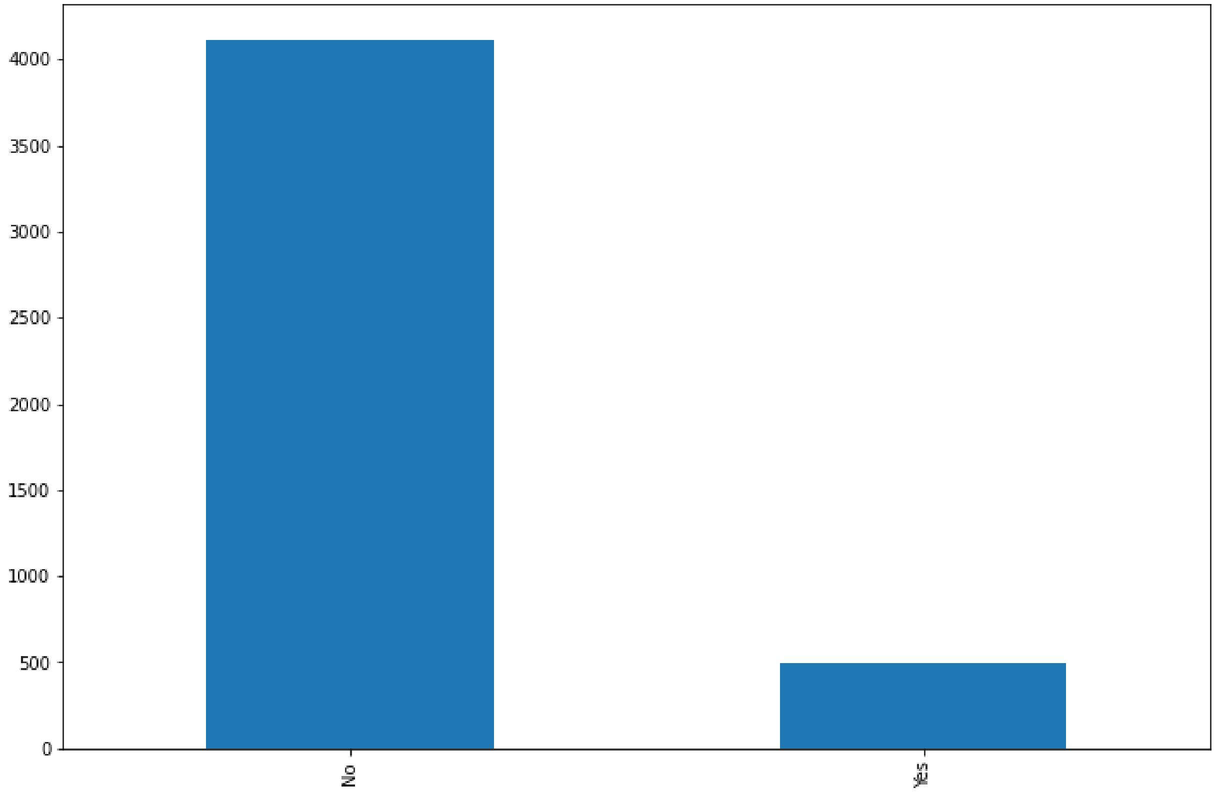
- Lets check the Last interaction between the lead and ExtraaLearn.

In [46]:
```
data["last_activity"].value_counts().plot.bar(figsize=(12,8))
plt.show()
```



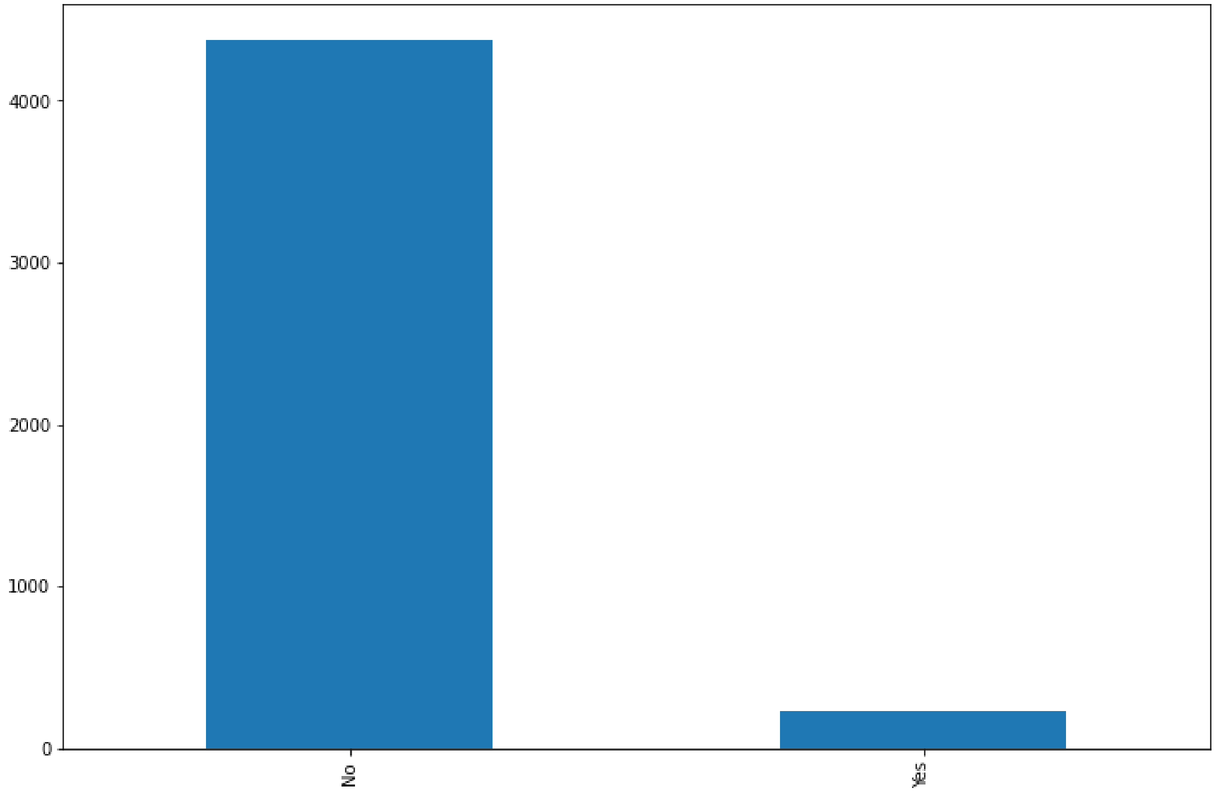
- Lets see whether the lead had seen the ad of ExtraaLearn in the Newspaper.

In [47]:
```
data["print_media_type1"].value_counts().plot.bar(figsize=(12,8))
plt.show()
```



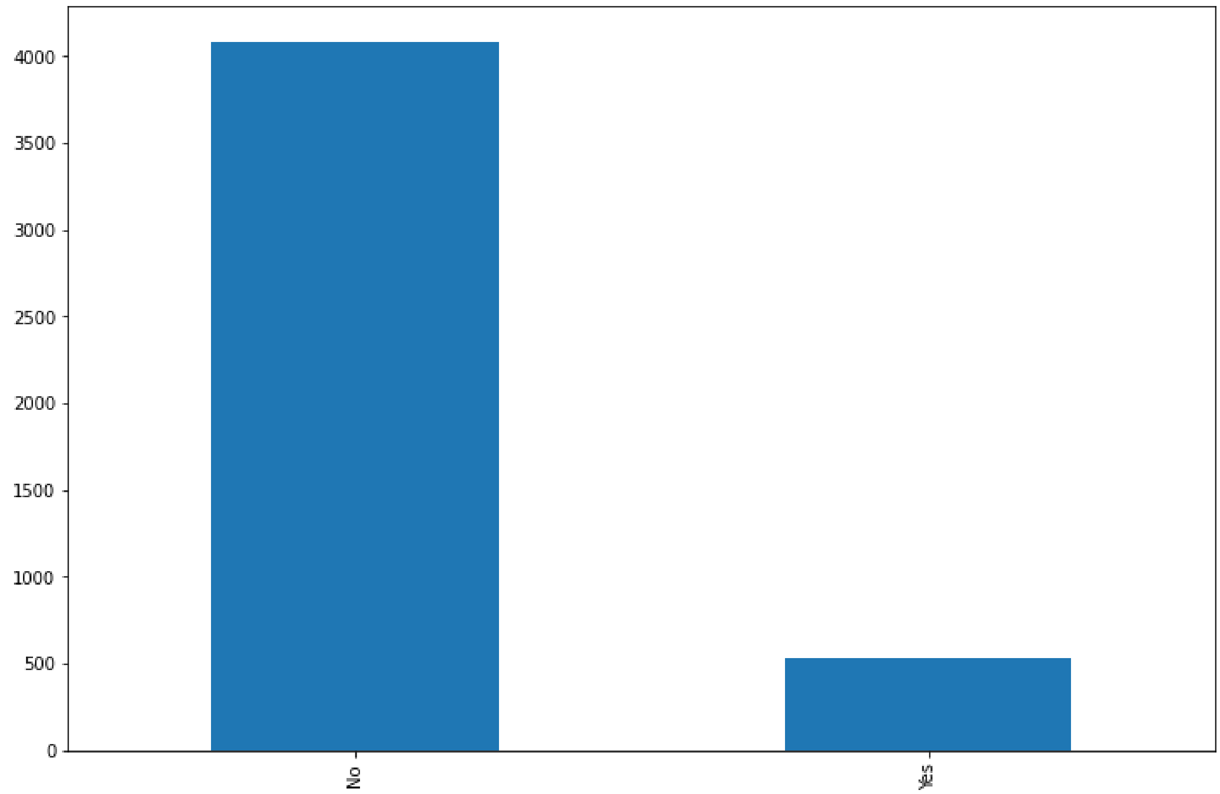
- Lets check whether the lead had seen the ad of ExtraaLearn in the Magazine.

In [48]:
```
data["print_media_type2"].value_counts().plot.bar(figsize=(12,8))
plt.show()
```



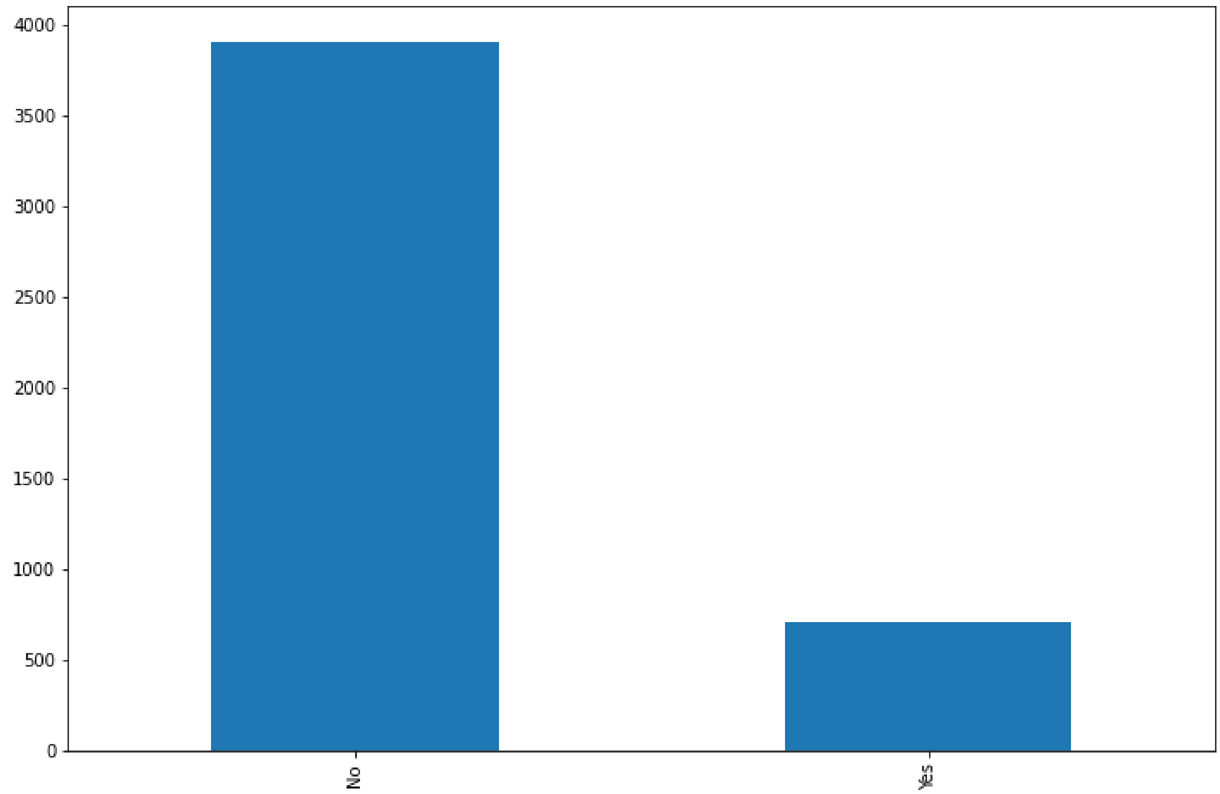
- Let see whether the lead had seen the ad of ExtraaLearn on the digital platforms.

In [57]:
```
data["digital_media"].value_counts().plot.bar(figsize=(12,8))
plt.show()
```



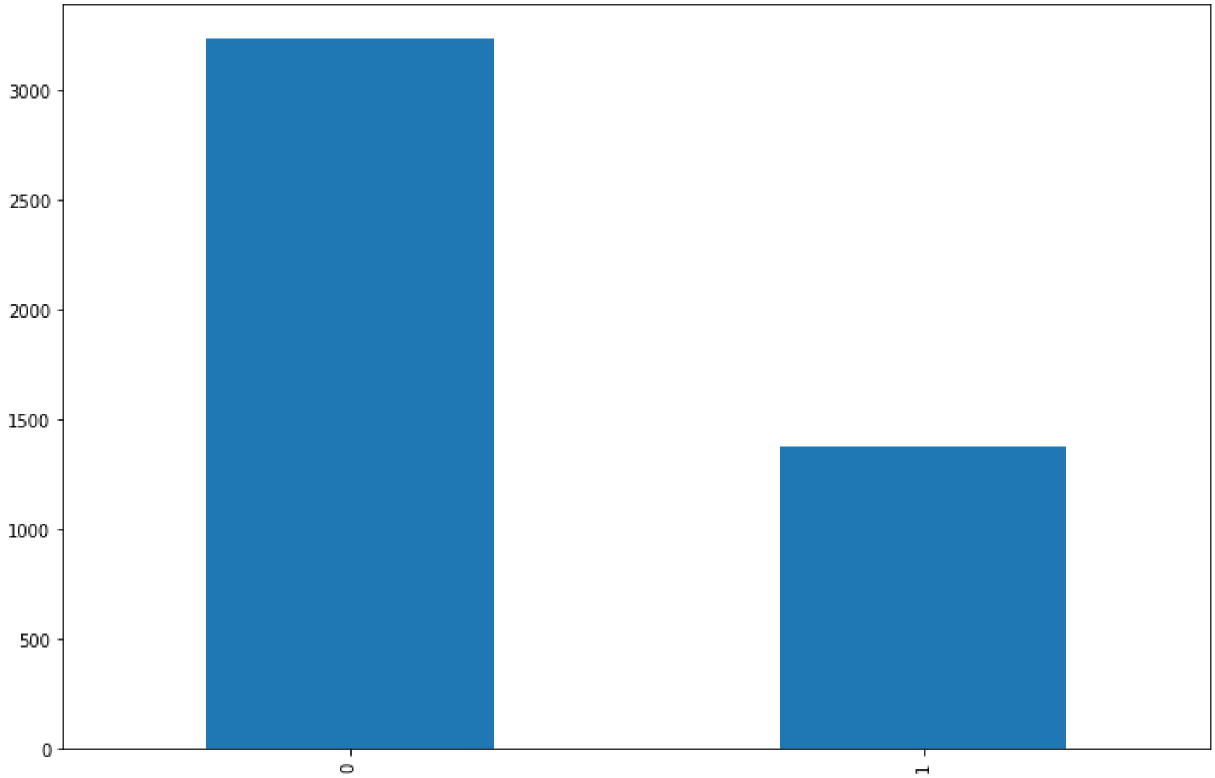
- Lets check whether the lead had heard about ExtraaLearn in the education channels like online forums, discussion threads, educational websites, etc.

In [50]:
```
data["educational_channels"].value_counts().plot.bar(figsize=(12,8))
plt.show()
```



- Lets check how many customers converted

In [52]:
```
data["status"].value_counts().plot.bar(figsize=(12,8))
plt.show()
```



We can see that small number of customers converted to real buyers

4. Data Cleaning

In this section, the data cleaning will be applied on the data to make it prepared for the machine learning

- In our dataset, we have categrocal columns which needs to be converted into numerical columns
- for example, "Current Occupation" have three categorical values, we will convert them into numerical values

In [56]:
```
data.head()
```

Out[56]:

	age	current_occupation	first_interaction	profile_completed	website_visits	time_spent_on_websi
0	57	Unemployed	Website	High	7	16
1	56	Professional	Mobile App	Medium	2	
2	52	Professional	Website	Medium	3	3
3	53	Unemployed	Website	High	4	4
4	23	Student	Website	High	4	6

In [58]:
```
data['current_occupation'].value_counts()
```

Out[58]:
```
Professional    2616
Unemployed      1441
Student          555
Name: current_occupation, dtype: int64
```

As we can see that the current occupation feature have. we will convert this into numerical values and give specific number to each of the category, this is often called encoding

- Professional
- Unemployed
- Student

We will create a function for this

- the function will take the whole dataset
- take each categorical feature and convert it into numerical feature

In [70]:
```
def encode(df):
    columnsToEncode = list(df.select_dtypes(include=['category','object']))
    le = LabelEncoder()
    for feature in columnsToEncode:
        try:
            df[feature] = le.fit_transform(df[feature])
        except:
            print('Error encoding '+feature)
    display(df)
```

In [65]:
```
encode(data)
```

	age	current_occupation	first_interaction	profile_completed	website_visits	time_spent_on_we
0	57	2	1	0	7	
1	56	0	0	2	2	
2	52	0	1	2	3	
3	53	2	1	0	4	
4	23	1	1	0	4	
...	...	...	...	...	...	
4607	35	2	0	2	15	
4608	55	0	0	2	8	
4609	58	0	1	0	2	
4610	57	0	0	2	1	
4611	55	0	1	2	4	

All the categorical features have been converted into numerical features

- lets check the correlation of the dataset

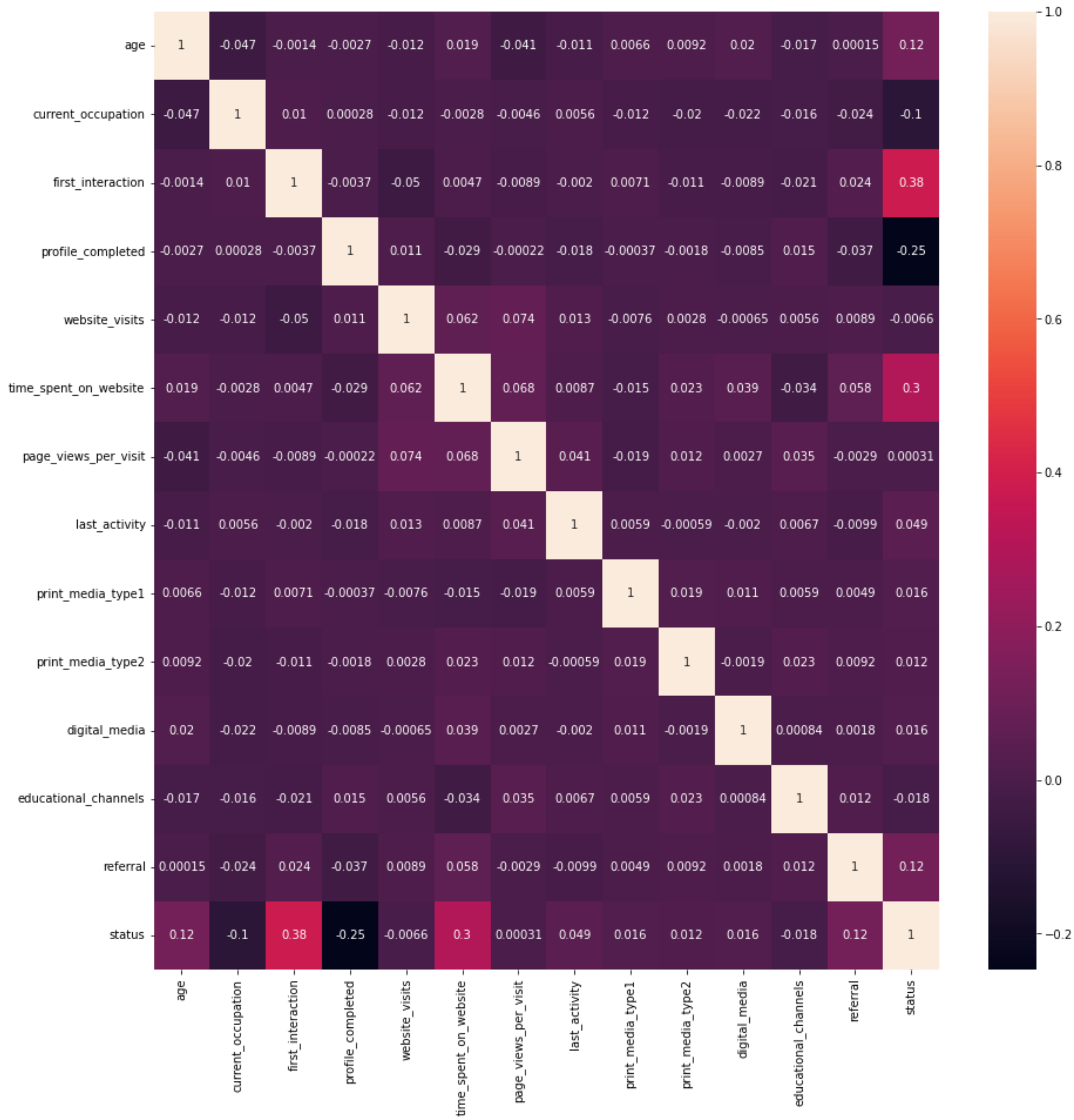
In [81]:
```
cormat = data.corr()
round(cormat,2)
```

Out[81]:

	age	current_occupation	first_interaction	profile_completed	website_visits
age	1.00	-0.05	-0.00	-0.00	-0.01
current_occupation	-0.05	1.00	0.01	0.00	-0.01
first_interaction	-0.00	0.01	1.00	-0.00	-0.05
profile_completed	-0.00	0.00	-0.00	1.00	0.01
website_visits	-0.01	-0.01	-0.05	0.01	1.00
...	...	...	...	...	...
print_media_type2	0.01	-0.02	-0.01	-0.00	0.00
digital_media	0.02	-0.02	-0.01	-0.01	-0.00
educational_channels	-0.02	-0.02	-0.02	0.01	0.01
referral	0.00	-0.02	0.02	-0.04	0.01
status	0.12	-0.10	0.38	-0.25	-0.01

In [78]:
```
plt.figure(figsize = (15,15))
sns.heatmap(cormat, annot = True)
```

Out[78]: <AxesSubplot:>



The heatmap shows that the profile completed feature is negatively correlated with the status

- it means that if profile feature is increasing, the status will decrease and vice versa

5. Model Building

as we have explored the data and cleaned the data

- Now we will go ahead to split the data and train machine learning models
- We will train different models and based on the accuracy, we will select the model
- the selected model will go through hyperpatameter tuning at the end

In [84]:
```
data['status'].value_counts()
```

Out[84]:
```
0    3235
1    1377
Name: status, dtype: int64
```

Our dataset is highly imbalanced

5.1. Data Splitting

In [89]:
```
X = data.drop('status', axis = 1) # lets store all the independent features in X
```

In [92]:
```
y = data['status'] # store the dependent variable in y
```

In [93]:
```
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2, random_
```

We have splitted the data into training and testing sets

- Training data will be used to train the models
- The testing data will be used to evaluate the model

5.2. Logistic Regression

- Lets train the very first model on the data and see how it is performing

In [95]:
```
logistic = LogisticRegression().fit(X_train, y_train)
```

```
C:\Users\hp\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:444:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-
learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regressi
on)
  n_iter_i = _check_optimize_result(
```

In [104]:
```
logistic.score(X_train,y_train)
```

Out[104]:
```
0.7926267281105991
```

In [105]:
```
logistic.score(X_test,y_test)
```

Out[105]:
```
0.7800650054171181
```

Logistic Regression performed good, the accuracy on the testing and training data is good which means that our model is not overfitting the data, which is a good sign.

5.3. Random Forest

lets try the Random Forest model

In [101]:
```
random = RandomForestClassifier().fit(X_train, y_train)
```

In [102]:
```
random.score(X_train,y_train)
```

Out[102]:
```
0.9997289238275956
```

In [103]:
```
random.score(X_test,y_test)
```

Out[103]:
```
0.8483206933911159
```

The Random Forest gave accuracy greater than Logistic Regression

5.4. Decision Tree

Lets move ahead and train the Decision Tree Classifier

In [108]:
```
decision = DecisionTreeClassifier().fit(X_train, y_train)
```

In [112]:
```
decision.score(X_train,y_train)
```

Out[112]:
```
0.9997289238275956
```

In [113]:
```
decision.score(X_test,y_test)
```

Out[113]:
```
0.8320693391115926
```

The accuracy of Decision Tree on the testing data is lower than Random Forest

5.5. Gradient Boosting

we will now train GradientBoostingClassifier

In [116]:
```
gradient = GradientBoostingClassifier().fit(X_train, y_train)
```

In [117]:
```
gradient.score(X_train,y_train)
```

Out[117]:
```
0.8864190837625373
```

In [118]:
```
gradient.score(X_test,y_test)
```

Out[118]:
```
0.8732394366197183
```

Gradient Boosting classifier beats all the above model by giving the highest accuracy on testing data

5.6. Extra Tree Classifier

In [120]:
```
extra = ExtraTreesClassifier().fit(X_train, y_train)
```

In [121]:
```
extra.score(X_train,y_train)
```

Out[121]:
```
0.9997289238275956
```

In [122]:
```
extra.score(X_test,y_test)
```

Out[122]:
```
0.8374864572047671
```

We have trained 5 different machine learning cclassifiers. the highest accuracy is given by Gradient Boosting Classfier

- Lets train Gradient Boosting Classifier again using hyperparameter tuning
- with classification report and confusion metrix

Gradient Boosting with parameters tuning

the highest accuracy is achieved by gradient boosting on testing data

- we will now train it again using different parameters using the GridSearchCv
- classification report
- confusion metrics

In [123]:
```
# Here you will make the list of all possibilities for each of the Hyperparameter
gbc = GradientBoostingClassifier()
parameters = {
    "n_estimators":[5,50,250,500],
    "max_depth":[1,3,5,7,9],
    "learning_rate":[0.01,0.1,1,10,100]
}
```

In [125]:
```
# We are using the GrisSearchCV for cross validation
cv = GridSearchCV(gbc,parameters,cv=5)
cv.fit(X_train, y_train)
```

Out[125]:
```
▸ GridSearchCV
▸ estimator: GradientBoostingClassifier
  ▾ GradientBoostingClassifier
  GradientBoostingClassifier()
```

In [126]:
```
def display(results):
    print(f'Best parameters are: {results.best_params_}')
    print("\n")
    mean_score = results.cv_results_['mean_test_score']
    std_score = results.cv_results_['std_test_score']
    params = results.cv_results_['params']
    for mean,std,params in zip(mean_score,std_score,params):
        print(f'{round(mean,3)} + or -{round(std,3)} for the {params}')
```

In [127]:
```
display(cv)
```

```
Best parameters are: {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators':
250}


0.701 + or -0.001 for the {'learning_rate': 0.01, 'max_depth': 1, 'n_estimato
rs': 5}
0.701 + or -0.001 for the {'learning_rate': 0.01, 'max_depth': 1, 'n_estimato
rs': 50}
0.787 + or -0.009 for the {'learning_rate': 0.01, 'max_depth': 1, 'n_estimato
rs': 250}
0.819 + or -0.01 for the {'learning_rate': 0.01, 'max_depth': 1, 'n_estimator
s': 500}
0.701 + or -0.001 for the {'learning_rate': 0.01, 'max_depth': 3, 'n_estimato
rs': 5}
0.766 + or -0.01 for the {'learning_rate': 0.01, 'max_depth': 3, 'n_estimator
s': 50}
0.85 + or -0.02 for the {'learning_rate': 0.01, 'max_depth': 3, 'n_estimator
s': 250}
0.856 + or -0.018 for the {'learning_rate': 0.01, 'max_depth': 3, 'n_estimato
rs': 500}
0.701 + or -0.001 for the {'learning_rate': 0.01, 'max_depth': 5, 'n_estimato
rs': 5}
0.802 + or -0.015 for the {'learning_rate': 0.01, 'max_depth': 5, 'n_estimato
rs': 50}
0.863 + or -0.015 for the {'learning_rate': 0.01, 'max_depth': 5, 'n_estimato
rs': 250}
0.861 + or -0.013 for the {'learning_rate': 0.01, 'max_depth': 5, 'n_estimato
rs': 500}
0.701 + or -0.001 for the {'learning_rate': 0.01, 'max_depth': 7, 'n_estimato
rs': 5}
0.822 + or -0.009 for the {'learning_rate': 0.01, 'max_depth': 7, 'n_estimato
rs': 50}
0.853 + or -0.014 for the {'learning_rate': 0.01, 'max_depth': 7, 'n_estimato
rs': 250}
0.848 + or -0.013 for the {'learning_rate': 0.01, 'max_depth': 7, 'n_estimato
rs': 500}
0.701 + or -0.001 for the {'learning_rate': 0.01, 'max_depth': 9, 'n_estimato
rs': 5}
0.824 + or -0.008 for the {'learning_rate': 0.01, 'max_depth': 9, 'n_estimato
rs': 50}
0.838 + or -0.009 for the {'learning_rate': 0.01, 'max_depth': 9, 'n_estimato
rs': 250}
0.839 + or -0.008 for the {'learning_rate': 0.01, 'max_depth': 9, 'n_estimato
rs': 500}
0.701 + or -0.001 for the {'learning_rate': 0.1, 'max_depth': 1, 'n_estimator
s': 5}
0.818 + or -0.014 for the {'learning_rate': 0.1, 'max_depth': 1, 'n_estimator
s': 50}
0.828 + or -0.013 for the {'learning_rate': 0.1, 'max_depth': 1, 'n_estimator
s': 250}
0.829 + or -0.009 for the {'learning_rate': 0.1, 'max_depth': 1, 'n_estimator
s': 500}
0.766 + or -0.009 for the {'learning_rate': 0.1, 'max_depth': 3, 'n_estimator
s': 5}
0.857 + or -0.016 for the {'learning_rate': 0.1, 'max_depth': 3, 'n_estimator
```

```
s': 50}
0.851 + or -0.015 for the {'learning_rate': 0.1, 'max_depth': 3, 'n_estimator
s': 250}
0.85 + or -0.016 for the {'learning_rate': 0.1, 'max_depth': 3, 'n_estimator
s': 500}
0.807 + or -0.005 for the {'learning_rate': 0.1, 'max_depth': 5, 'n_estimator
s': 5}
0.857 + or -0.013 for the {'learning_rate': 0.1, 'max_depth': 5, 'n_estimator
s': 50}
0.843 + or -0.009 for the {'learning_rate': 0.1, 'max_depth': 5, 'n_estimator
s': 250}
0.834 + or -0.006 for the {'learning_rate': 0.1, 'max_depth': 5, 'n_estimator
s': 500}
0.829 + or -0.009 for the {'learning_rate': 0.1, 'max_depth': 7, 'n_estimator
s': 5}
0.849 + or -0.008 for the {'learning_rate': 0.1, 'max_depth': 7, 'n_estimator
s': 50}
0.839 + or -0.013 for the {'learning_rate': 0.1, 'max_depth': 7, 'n_estimator
s': 250}
0.838 + or -0.013 for the {'learning_rate': 0.1, 'max_depth': 7, 'n_estimator
s': 500}
0.825 + or -0.006 for the {'learning_rate': 0.1, 'max_depth': 9, 'n_estimator
s': 5}
0.838 + or -0.01 for the {'learning_rate': 0.1, 'max_depth': 9, 'n_estimator
s': 50}
0.841 + or -0.008 for the {'learning_rate': 0.1, 'max_depth': 9, 'n_estimator
s': 250}
0.842 + or -0.013 for the {'learning_rate': 0.1, 'max_depth': 9, 'n_estimator
s': 500}
0.817 + or -0.01 for the {'learning_rate': 1, 'max_depth': 1, 'n_estimators':
5}
0.831 + or -0.009 for the {'learning_rate': 1, 'max_depth': 1, 'n_estimator
s': 50}
0.825 + or -0.011 for the {'learning_rate': 1, 'max_depth': 1, 'n_estimator
s': 250}
0.824 + or -0.01 for the {'learning_rate': 1, 'max_depth': 1, 'n_estimators':
500}
0.855 + or -0.015 for the {'learning_rate': 1, 'max_depth': 3, 'n_estimator
s': 5}
0.816 + or -0.011 for the {'learning_rate': 1, 'max_depth': 3, 'n_estimator
s': 50}
0.812 + or -0.015 for the {'learning_rate': 1, 'max_depth': 3, 'n_estimator
s': 250}
0.813 + or -0.013 for the {'learning_rate': 1, 'max_depth': 3, 'n_estimator
s': 500}
0.837 + or -0.015 for the {'learning_rate': 1, 'max_depth': 5, 'n_estimator
s': 5}
0.818 + or -0.011 for the {'learning_rate': 1, 'max_depth': 5, 'n_estimator
s': 50}
0.824 + or -0.01 for the {'learning_rate': 1, 'max_depth': 5, 'n_estimators':
250}
0.826 + or -0.008 for the {'learning_rate': 1, 'max_depth': 5, 'n_estimator
s': 500}
0.828 + or -0.011 for the {'learning_rate': 1, 'max_depth': 7, 'n_estimator
s': 5}
0.825 + or -0.013 for the {'learning_rate': 1, 'max_depth': 7, 'n_estimator
s': 50}
```

```
0.83 + or -0.015 for the {'learning_rate': 1, 'max_depth': 7, 'n_estimators':
250}
0.838 + or -0.01 for the {'learning_rate': 1, 'max_depth': 7, 'n_estimators':
500}
0.814 + or -0.011 for the {'learning_rate': 1, 'max_depth': 9, 'n_estimator
s': 5}
0.832 + or -0.012 for the {'learning_rate': 1, 'max_depth': 9, 'n_estimator
s': 50}
0.829 + or -0.016 for the {'learning_rate': 1, 'max_depth': 9, 'n_estimator
s': 250}
0.835 + or -0.011 for the {'learning_rate': 1, 'max_depth': 9, 'n_estimator
s': 500}
0.338 + or -0.02 for the {'learning_rate': 10, 'max_depth': 1, 'n_estimator
s': 5}
0.338 + or -0.02 for the {'learning_rate': 10, 'max_depth': 1, 'n_estimator
s': 50}
0.338 + or -0.02 for the {'learning_rate': 10, 'max_depth': 1, 'n_estimator
s': 250}
0.338 + or -0.02 for the {'learning_rate': 10, 'max_depth': 1, 'n_estimator
s': 500}
0.39 + or -0.024 for the {'learning_rate': 10, 'max_depth': 3, 'n_estimator
s': 5}
0.39 + or -0.024 for the {'learning_rate': 10, 'max_depth': 3, 'n_estimator
s': 50}
0.39 + or -0.024 for the {'learning_rate': 10, 'max_depth': 3, 'n_estimator
s': 250}
0.39 + or -0.024 for the {'learning_rate': 10, 'max_depth': 3, 'n_estimator
s': 500}
0.422 + or -0.078 for the {'learning_rate': 10, 'max_depth': 5, 'n_estimator
s': 5}
0.422 + or -0.078 for the {'learning_rate': 10, 'max_depth': 5, 'n_estimator
s': 50}
0.422 + or -0.078 for the {'learning_rate': 10, 'max_depth': 5, 'n_estimator
s': 250}
0.422 + or -0.078 for the {'learning_rate': 10, 'max_depth': 5, 'n_estimator
s': 500}
0.613 + or -0.041 for the {'learning_rate': 10, 'max_depth': 7, 'n_estimator
s': 5}
0.544 + or -0.076 for the {'learning_rate': 10, 'max_depth': 7, 'n_estimator
s': 50}
0.567 + or -0.059 for the {'learning_rate': 10, 'max_depth': 7, 'n_estimator
s': 250}
0.547 + or -0.059 for the {'learning_rate': 10, 'max_depth': 7, 'n_estimator
s': 500}
0.697 + or -0.07 for the {'learning_rate': 10, 'max_depth': 9, 'n_estimator
s': 5}
0.678 + or -0.023 for the {'learning_rate': 10, 'max_depth': 9, 'n_estimator
s': 50}
0.625 + or -0.076 for the {'learning_rate': 10, 'max_depth': 9, 'n_estimator
s': 250}
0.655 + or -0.055 for the {'learning_rate': 10, 'max_depth': 9, 'n_estimator
s': 500}
0.299 + or -0.001 for the {'learning_rate': 100, 'max_depth': 1, 'n_estimator
s': 5}
0.299 + or -0.001 for the {'learning_rate': 100, 'max_depth': 1, 'n_estimator
s': 50}
0.299 + or -0.001 for the {'learning_rate': 100, 'max_depth': 1, 'n_estimator
```

```
s': 250}
0.299 + or -0.001 for the {'learning_rate': 100, 'max_depth': 1, 'n_estimator
s': 500}
0.369 + or -0.049 for the {'learning_rate': 100, 'max_depth': 3, 'n_estimator
s': 5}
0.369 + or -0.049 for the {'learning_rate': 100, 'max_depth': 3, 'n_estimator
s': 50}
0.369 + or -0.049 for the {'learning_rate': 100, 'max_depth': 3, 'n_estimator
s': 250}
0.369 + or -0.049 for the {'learning_rate': 100, 'max_depth': 3, 'n_estimator
s': 500}
0.481 + or -0.057 for the {'learning_rate': 100, 'max_depth': 5, 'n_estimator
s': 5}
0.481 + or -0.057 for the {'learning_rate': 100, 'max_depth': 5, 'n_estimator
s': 50}
0.48 + or -0.057 for the {'learning_rate': 100, 'max_depth': 5, 'n_estimator
s': 250}
0.48 + or -0.057 for the {'learning_rate': 100, 'max_depth': 5, 'n_estimator
s': 500}
0.522 + or -0.054 for the {'learning_rate': 100, 'max_depth': 7, 'n_estimator
s': 5}
0.461 + or -0.075 for the {'learning_rate': 100, 'max_depth': 7, 'n_estimator
s': 50}
0.5 + or -0.037 for the {'learning_rate': 100, 'max_depth': 7, 'n_estimator
s': 250}
0.53 + or -0.045 for the {'learning_rate': 100, 'max_depth': 7, 'n_estimator
s': 500}
0.609 + or -0.048 for the {'learning_rate': 100, 'max_depth': 9, 'n_estimator
s': 5}
0.613 + or -0.097 for the {'learning_rate': 100, 'max_depth': 9, 'n_estimator
s': 50}
0.67 + or -0.015 for the {'learning_rate': 100, 'max_depth': 9, 'n_estimator
s': 250}
0.634 + or -0.045 for the {'learning_rate': 100, 'max_depth': 9, 'n_estimator
s': 500}
```

So the best parameters are : {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 250}

- we will train the model again on these parameters

In [133]:
```
GradientBoosting = GradientBoostingClassifier(learning_rate=0.01, n_estimators=25
```

In [134]:
```
GradientBoosting.fit(X_train, y_train)
```

Out[134]:
```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.01, max_depth=5, n_estimators=250)
```

In [135]:
```
GradientBoosting.score(X_train,y_train)
```

Out[135]: 0.8891298454865817

In [137]:
```
GradientBoosting.score(X_test,y_test)
```

Out[137]:
```
0.8764897074756229
```

In [138]:
```
# Lets do the predictions on the testing data
pred = GradientBoosting.predict(X_test)
pred
```

Out[138]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0,
       1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
      dtype=int64)
```

Classification Report & Confusion Metrics

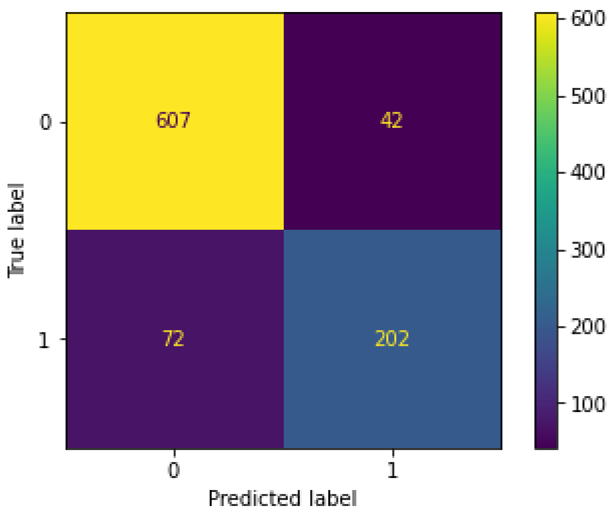
In [145]:
```
print(classification_report(y_test, pred))
plot_confusion_matrix(GradientBoosting, X_test, y_test)
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.89      0.94      0.91       649
           1       0.83      0.74      0.78       274

    accuracy                           0.88       923
   macro avg       0.86      0.84      0.85       923
weighted avg       0.87      0.88      0.87       923
```

```
C:\Users\hp\anaconda3\lib\site-packages\sklearn\utils\deprecation.py:87: Future
Warning: Function plot_confusion_matrix is deprecated; Function `plot_confusion
_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class
methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from
_estimator.
  warnings.warn(msg, category=FutureWarning)
```



END