Decision Trees and Random Forest Project: Predicting Potential Customers

Project on classification using Decision Tree and Random Forest.

Context

The EdTech industry has been surging in the past decade immensely, and according to a forecast, the Online Education market, would be worth $286.62bn by 2023, with a compound annual growth rate (CAGR) of 10.26% from 2018 to 2023. The modern era of online education has enforced a lot in its growth and expansion beyond any limit. Due to having many dominant features like ease of information sharing, personalized learning experience, transparency of assessment, etc., it is now preferable to traditional education.

The online education sector has witnessed rapid growth and is attracting a lot of new customers. Due to this rapid growth, many new companies have emerged in this industry. With the availability and ease of use of digital marketing resources, companies can reach out to a wider audience with their offerings. The customers who show interest in these offerings are termed as **leads**. There are various sources of obtaining leads for Edtech companies, like:

- The customer interacts with the marketing front on social media or other online platforms.
- The customer browses the website/app and downloads the brochure.
- The customer connects through emails for more information.

The company then nurtures these leads and tries to convert them to paid customers. For this, the representative from the organization connects with the lead on call or through email to share further details.

Objective

LearnIT 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 LearnIT is to identify which of the leads are more likely to convert so that they can allocate the resources accordingly. You, as a data scientist at LearnIT, 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.

Data Description

The data contains the different attributes of leads and their interaction details with LearnIT. The detailed data dictionary is given below.

- **ID:** ID of the lead
- **age:** Age of the lead
- **current_occupation:** Current occupation of the lead. Values include 'Professional', 'Unemployed', and 'Student'
- **first_interaction:** How did the lead first interact with ExtraaLearn? Values include 'Website' and 'Mobile App'
- **profile_completed:** What percentage of the profile has been filled by the lead on the website/mobile app? Values include Low - (0-50%), Medium - (50-75%), High (75-100%)
- **website_visits:** The number of times a lead has visited the website
- **time_spent_on_website:** Total time spent on the website
- **page_views_per_visit:** Average number of pages on the website viewed during the visits
- **last_activity:** Last interaction between the lead and LearnIT
    - **Email Activity:** Seeking details about the program through email, Representative shared information with a lead like a brochure of the program, etc.
    - **Phone Activity:** Had a phone conversation with a representative, had a conversation over SMS with a representative, etc.
    - **Website Activity:** Interacted on live chat with a representative, updated profile on the website, etc.
- **print_media_type1:** Flag indicating whether the lead had seen the ad of LearnIT in the Newspaper
- **print_media_type2:** Flag indicating whether the lead had seen the ad of LearnIT in the Magazine
- **digital_media:** Flag indicating whether the lead had seen the ad of LearnIT on the digital platforms
- **educational_channels:** Flag indicating whether the lead had heard about LearnIT in the education channels like online forums, discussion threads, educational websites, etc.
- **referral:** Flag indicating whether the lead had heard about LearnIT through reference.
- **status:** Flag indicating whether the lead was converted to a paid customer or not.

Importing the necessary libraries and overview of the dataset

In [109]:

```
import warnings
warnings.filterwarnings("ignore")

# Libraries for data manipulation and visualization
```

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

# Algorithms to use
from sklearn.tree import DecisionTreeClassifier

from sklearn import tree

from sklearn.ensemble import RandomForestClassifier

# Metrics to evaluate the model
from sklearn.metrics import confusion_matrix, classification_report, recall_score

from sklearn import metrics

# For hyperparameter tuning
from sklearn.model_selection import GridSearchCV
```

Loading the dataset

In [110]:

```
learn = pd.read_csv("LearnIT.csv")
```

In [111]:

```
# Copying data to another variable to avoid any changes to the original data
data = learn.copy()
```

View the first and the last 5 rows of the dataset

In [112]:

```
data.head()
```

Out[112]:

	ID	age	current_occupation	first_interaction	profile_completed	website_visits	time_spent_on_website	page_views_per_visit	last_activity	print_media
0	EXT001	57	Unemployed	Website	High	7	1639	1.861	Website Activity	
1	EXT002	56	Professional	Mobile App	Medium	2	83	0.320	Website Activity	
2	EXT003	52	Professional	Website	Medium	3	330	0.074	Website Activity	
3	EXT004	53	Unemployed	Website	High	4	464	2.057	Website Activity	
4	EXT005	23	Student	Website	High	4	600	16.914	Email Activity	

In [113]:

```
data.tail()
```

Out[113]:

	ID	age	current_occupation	first_interaction	profile_completed	website_visits	time_spent_on_website	page_views_per_visit	last_activity	print_r
4607	EXT4608	35	Unemployed	Mobile App	Medium	15	360	2.170	Phone Activity	
4608	EXT4609	55	Professional	Mobile App	Medium	8	2327	5.393	Email Activity	
4609	EXT4610	58	Professional	Website	High	2	212	2.692	Email Activity	
4610	EXT4611	57	Professional	Mobile App	Medium	1	154	3.879	Website Activity	
4611	EXT4612	55	Professional	Website	Medium	4	2290	2.075	Phone Activity	

Understand the shape of the dataset

In [114]:

In [114]:
```
data.shape
```

Out[114]:
```
(4612, 15)
```

- The dataset has **4612 rows and 15 columns.**

Check the data types of the columns in the dataset

In [115]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4612 entries, 0 to 4611
Data columns (total 15 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   4612 non-null   object 
 1   age                  4612 non-null   int64  
 2   current_occupation   4612 non-null   object 
 3   first_interaction    4612 non-null   object 
 4   profile_completed    4612 non-null   object 
 5   website_visits       4612 non-null   int64  
 6   time_spent_on_website  4612 non-null   int64  
 7   page_views_per_visit  4612 non-null   float64
 8   last_activity        4612 non-null   object 
 9   print_media_type1    4612 non-null   object 
 10  print_media_type2    4612 non-null   object 
 11  digital_media        4612 non-null   object 
 12  educational_channels  4612 non-null   object 
 13  referral             4612 non-null   object 
 14  status               4612 non-null   int64  
dtypes: float64(1), int64(4), object(10)
memory usage: 540.6+ KB
```

Observations:

- age , website_visits , time_spent_on_website , page_views_per_visit , and status are of numeric type while rest of the columns are of object type.
- There are **no null values** in the dataset.

In [116]:
```
# Checking for duplicate values
data.duplicated().sum()
```

Out[116]:
```
0
```

- There are **no duplicate values** in the data.

Exploratory Data Analysis

Univariate Analysis

Check the statistical summary of numerical variables in the data.

In [117]:
```
data.describe().T
```

Out[117]:

	count	mean	std	min	25%	50%	75%	max
age	4612.0	46.201214	13.161454	18.0	36.00000	51.000	57.00000	63.000
website_visits	4612.0	3.566782	2.829134	0.0	2.00000	3.000	5.00000	30.000
time_spent_on_website	4612.0	724.011275	743.828683	0.0	148.75000	376.000	1336.75000	2537.000
page_views_per_visit	4612.0	3.026126	1.968125	0.0	2.07775	2.792	3.75625	18.434
status	4612.0	0.298569	0.457680	0.0	0.00000	0.000	1.00000	1.000

Observations:

- The average age of leads in the data is about 46 years and the median age is 51 years. This implies that the majority of leads have good work experience and they may be looking for a shift in career or upskill themselves.
- On average, a lead visits the website 3 times. There are some leads who have never visited the website.
- On average, the leads spent 724 seconds or 12 minutes on the website. There's also a large difference between the median and 75th percentile and between 75th percentile and the maximum value which indicates there might be outliers present in this column.
- On average, a lead views 3 pages per visit. The large difference between the 75th percentile and the maximum value suggests that there might be outliers in this column.

In [118]:

```
# Making a list of all categorical variables
cat_col = list(data.select_dtypes("object").columns)

# Printing count of each unique value in each categorical column
for column in cat_col:
    print(data[column].value_counts(normalize = True))
    print("-" * 50)
```

```
EXT001     0.000217
EXT2884    0.000217
EXT3080    0.000217
EXT3079    0.000217
EXT3078    0.000217
           ...
EXT1537    0.000217
EXT1536    0.000217
EXT1535    0.000217
EXT1534    0.000217
EXT4612    0.000217
Name: ID, Length: 4612, dtype: float64
--------------------------------------------------
Professional    0.567216
Unemployed      0.312446
Student         0.120338
Name: current_occupation, dtype: float64
--------------------------------------------------
Website       0.551171
Mobile App    0.448829
Name: first_interaction, dtype: float64
--------------------------------------------------
High      0.490893
Medium    0.485906
Low       0.023200
Name: profile_completed, dtype: float64
--------------------------------------------------
Email Activity      0.493929
Phone Activity      0.267563
Website Activity    0.238508
Name: last_activity, dtype: float64
--------------------------------------------------
No     0.892238
Yes    0.107762
Name: print_media_type1, dtype: float64
--------------------------------------------------
No     0.94948
Yes    0.05052
Name: print_media_type2, dtype: float64
--------------------------------------------------
No     0.885733
Yes    0.114267
Name: digital_media, dtype: float64
--------------------------------------------------
No     0.847138
Yes    0.152862
Name: educational_channels, dtype: float64
--------------------------------------------------
No     0.979835
Yes    0.020165
Name: referral, dtype: float64
--------------------------------------------------
```

Observations:

- Most of the leads are working professionals.
- As expected, the majority of the leads interacted with ExtraaLearn from the website.
- Almost an equal percentage of profile completions are categorized as high and medium that is 49.1% and 48.6%, respectively. Only **2.3%** of the profile completions are categorized as low.
- Approx 49.4% of the leads had their last activity over email, followed by 26.8% having phone activity. This implies that the majority of the leads prefer to communicate via email.
- We can observe that each ID has an equal percentage of values. Let's check the number of unique values in the ID column.

In [119]:

```
# Checking the number of unique values
data["ID"].nunique()
```

Out[119]:

```
4612
```

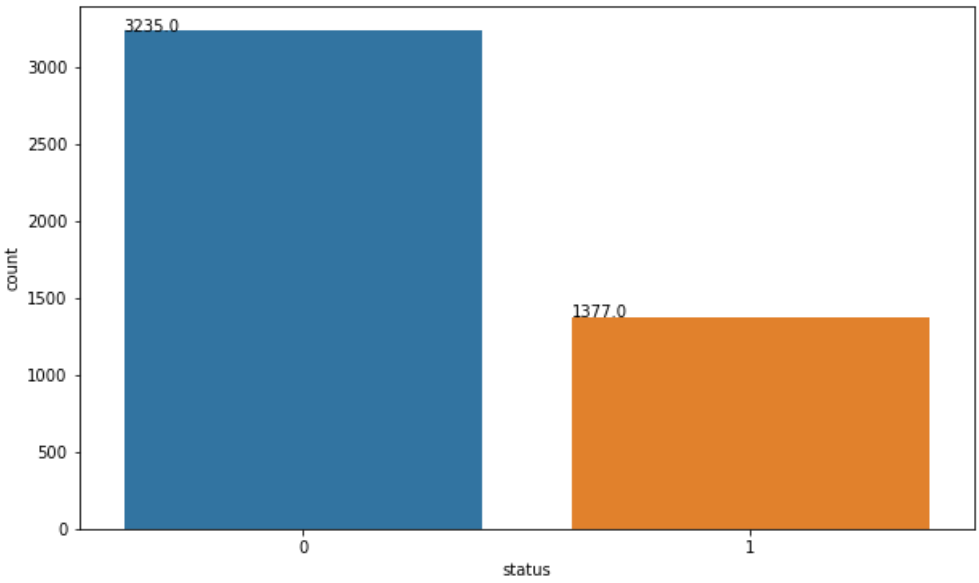
- All the values in the ID column are unique.
- Drop this column as it would not add value to our analysis.

In [120]:

```
# Dropping ID column
data.drop(["ID"], axis = 1, inplace = True)
```

In [121]:

```
plt.figure(figsize = (10, 6))

ax = sns.countplot(x = 'status', data = data)

# Annotating the exact count on the top of the bar for each category
for p in ax.patches:
    ax.annotate('{:.1f}'.format(p.get_height()), (p.get_x(), p.get_height()+0.35))
```



- The above plot shows that number of leads converted are significantly less than number of leads not converted which can be expected.
- The plot indicates that ~**30%** (1377/4612) of leads have been converted.

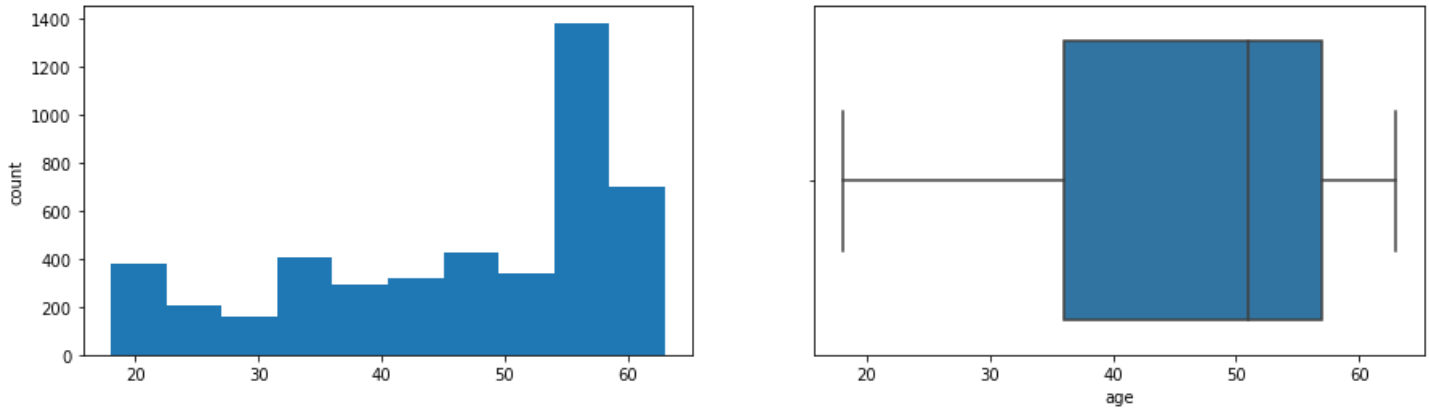
Check the distribution and outliers for numerical columns in the data

distribution plots and box plots

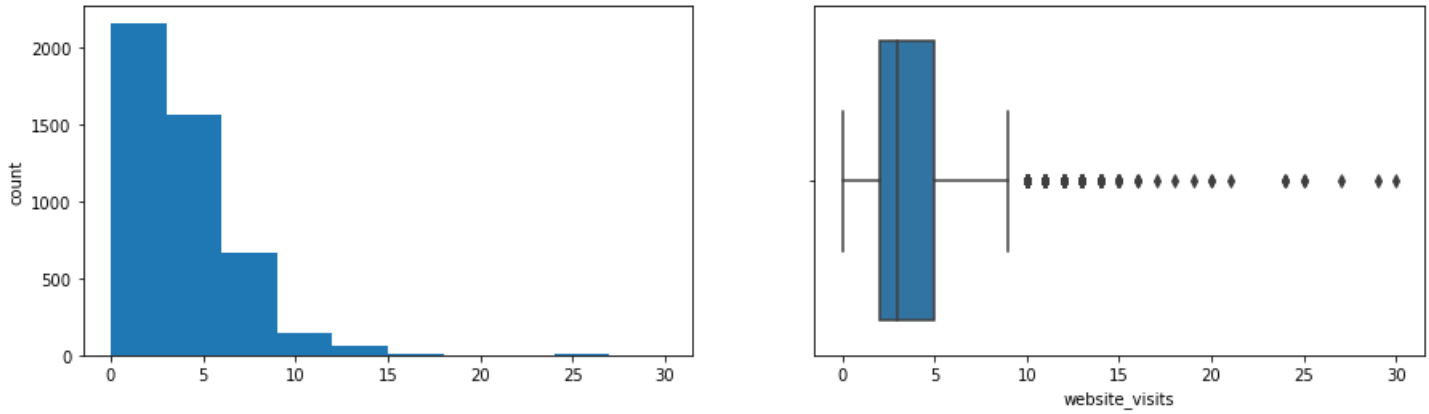
In [122]:

```
for col in ['age', 'website_visits', 'time_spent_on_website', 'page_views_per_visit']:
    print(col)

    print('Skew :',round(data[col].skew(), 2))

    plt.figure(figsize = (15, 4))

    plt.subplot(1, 2, 1)

    data[col].hist(bins = 10, grid = False)

    plt.ylabel('count')

    plt.subplot(1, 2, 2)

    sns.boxplot(x = data[col])

    plt.show()
```

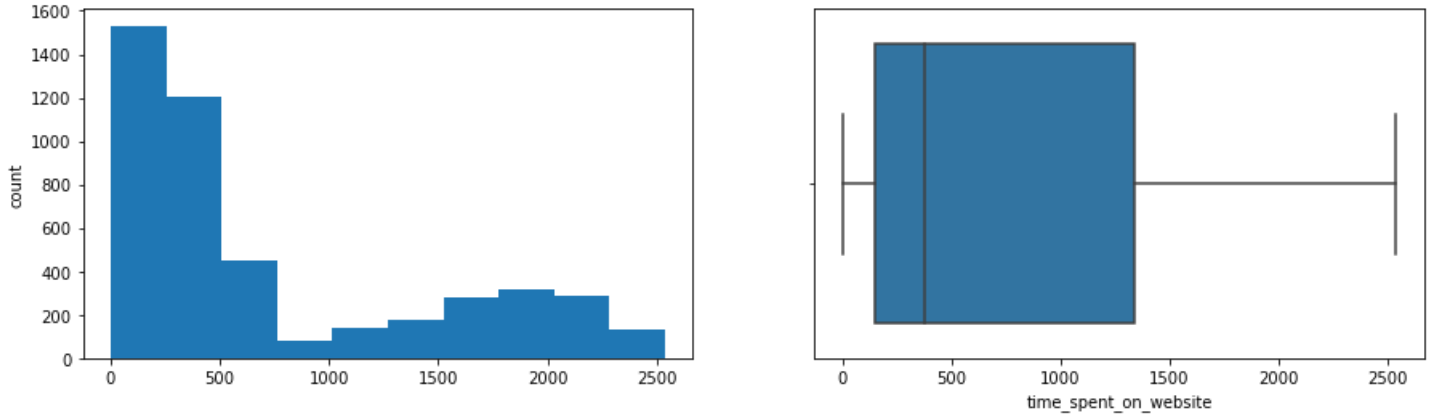
age
Skew : -0.72



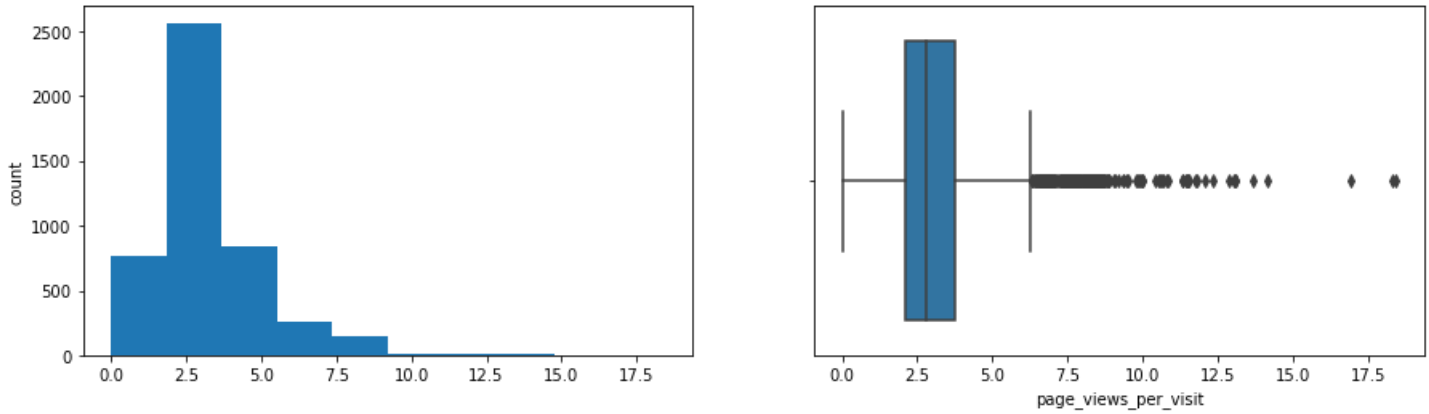
website_visits
Skew : 2.16



time_spent_on_website
Skew : 0.95



page_views_per_visit
Skew : 1.27



**Observations:

-- Age feature has a left skewed distribution, suggesting most of the leads are matured adults between the ages of 35 and 60.

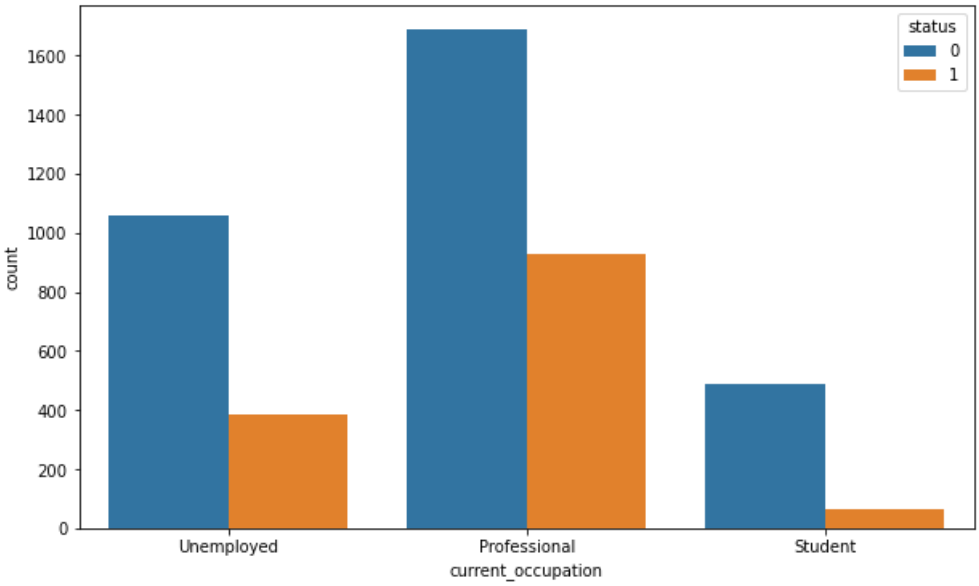
-- website visits, time spent on website and page views per visit have right skewed distributions with outliers except for time spent on website.

Bivariate Analysis

In [123]:

```
plt.figure(figsize = (10, 6))
```

```
sns.countplot(x = 'current_occupation', hue = 'status', data = data)

plt.show()
```



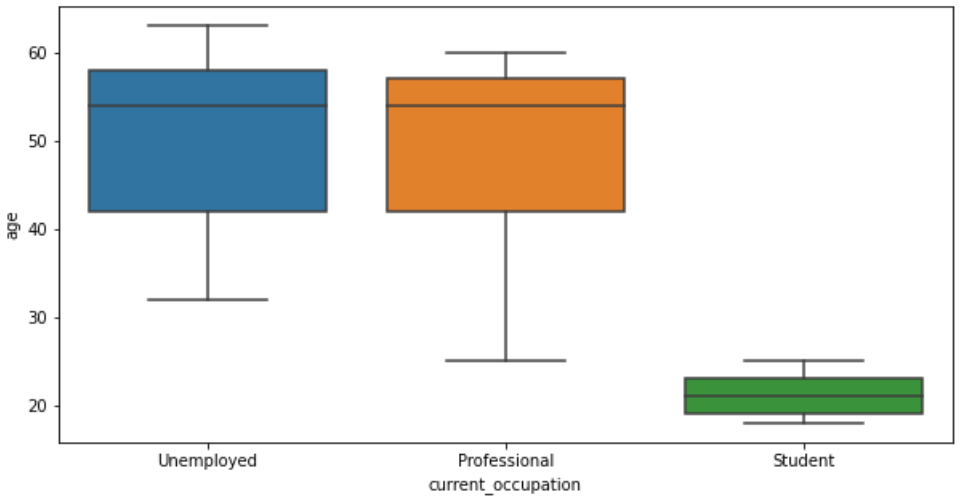
Observations:

- The plot shows that working professional leads are more likely to opt for a course offered by the organization and the students are least likely to be converted.
- This shows that the currently offered programs are more oriented toward working professionals or unemployed personnel. The programs might be suitable for the working professionals who might want to transition to a new role or take up more responsibility in their current role. And also focused on skills that are in high demand making it more suitable for working professionals or currently unemployed leads.

Age can also be a good factor to differentiate between such leads.

In [124]:

```
plt.figure(figsize = (10, 5))

sns.boxplot(data["current_occupation"], data["age"])

plt.show()
```



In [125]:

```
data.groupby(["current_occupation"])["age"].describe()
```

Out[125]:

	count	mean	std	min	25%	50%	75%	max
current_occupation								
Professional	2616.0	49.347477	9.890744	25.0	42.0	54.0	57.0	60.0
Student	555.0	21.144144	2.001114	18.0	19.0	21.0	23.0	25.0
Unemployed	1441.0	50.140180	9.999503	32.0	42.0	54.0	58.0	63.0

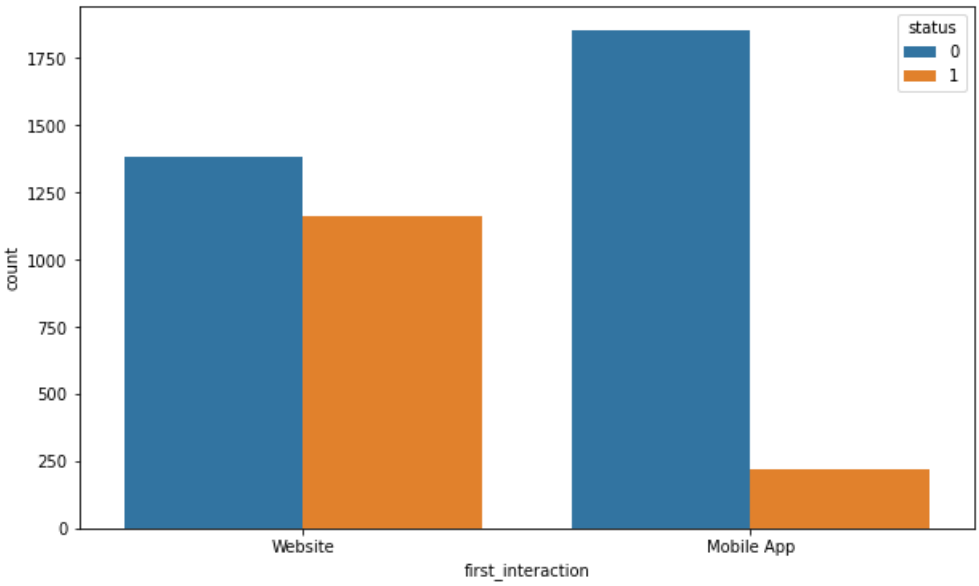
Observations:

- The range of age for students is 18 to 25 years.
- The range of age for professionals is 25 to 60 years.
- The range of age for unemployed leads is 32 to 63 years.
- The average age of working professionals and unemployed leads is almost 50 years.

Check if the channels of the first interaction have an impact on the conversion of leads.

In [126]:

```
plt.figure(figsize = (10, 6))

sns.countplot(x = 'first_interaction', hue = 'status', data = data)

plt.show()
```



Observations:

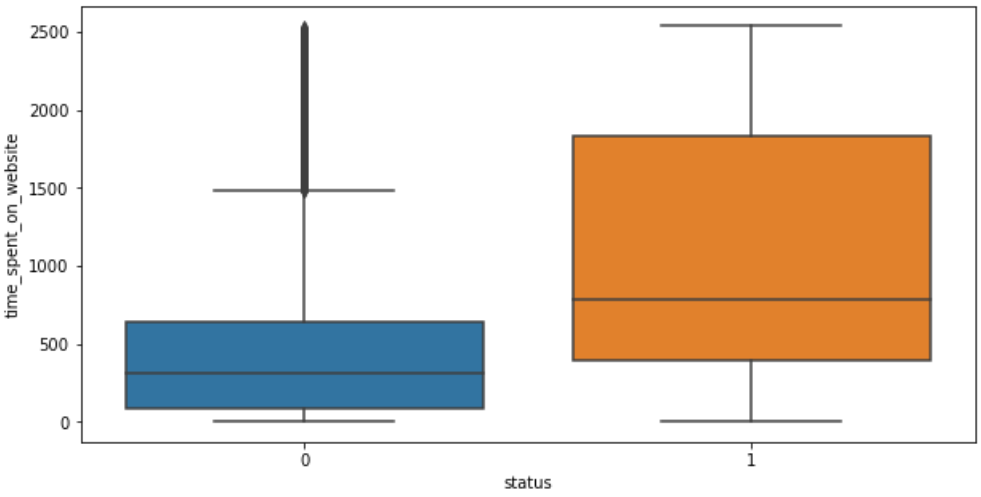
- The website seems to be doing a good job as compared to mobile app as there is a huge difference in the number of conversions of the leads who first interacted with the company through website and those who interacted through mobile application.
- Majority of the leads who interacted through websites were converted to paid customers, while only a small number of leads, who interacted through mobile app, converted.

We observed earlier that some leads spend more time on websites than others. Analyze if spending more time on websites results in conversion.

- **a boxplot for variables 'status' and 'time_spent_on_website' using sns.boxplot() function**

In [127]:

```
plt.figure(figsize = (10, 5))

sns.boxplot(data['status'], data['time_spent_on_website'])

plt.show()
```



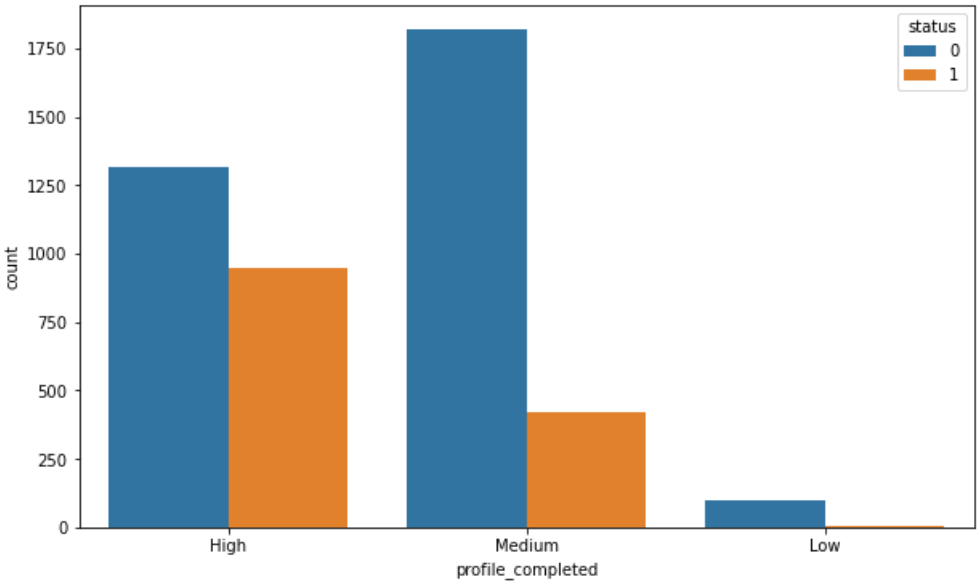
**Observations:

-- Most leads that were converted to paid customers spent more time on the website..

People browsing the website or the mobile app are generally required to create a profile by sharing their details before they can access more information. Check if the profile completion level has an impact on lead coversion

In [128]:

```
plt.figure(figsize = (10, 6))

sns.countplot(x = 'profile_completed', hue = 'status', data = data)

plt.show()
```



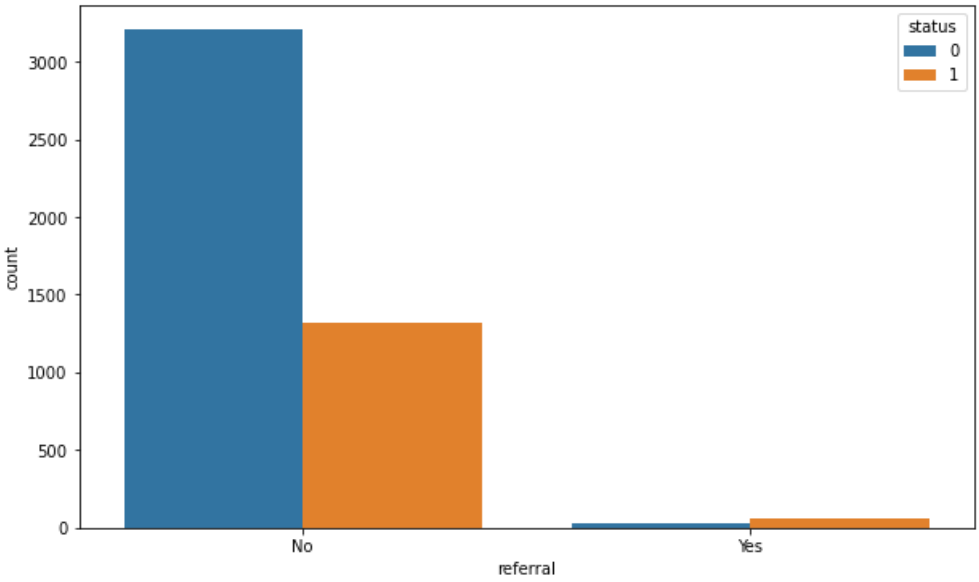
Observations:

- The leads whose profile completion level is high converted more in comparison to other levels of profile completion.
- The medium and low levels of profile completion saw comparatively very less conversions.
- The high level of profile completion might indicate a lead's intent to pursue the course which results in high conversion.

Referrals from a converted lead can be a good source of income with a very low cost of advertisement. Check how referrals impact lead conversion status.

In [129]:

```
plt.figure(figsize = (10, 6))

sns.countplot(x = 'referral', hue = 'status', data = data)

plt.show()
```



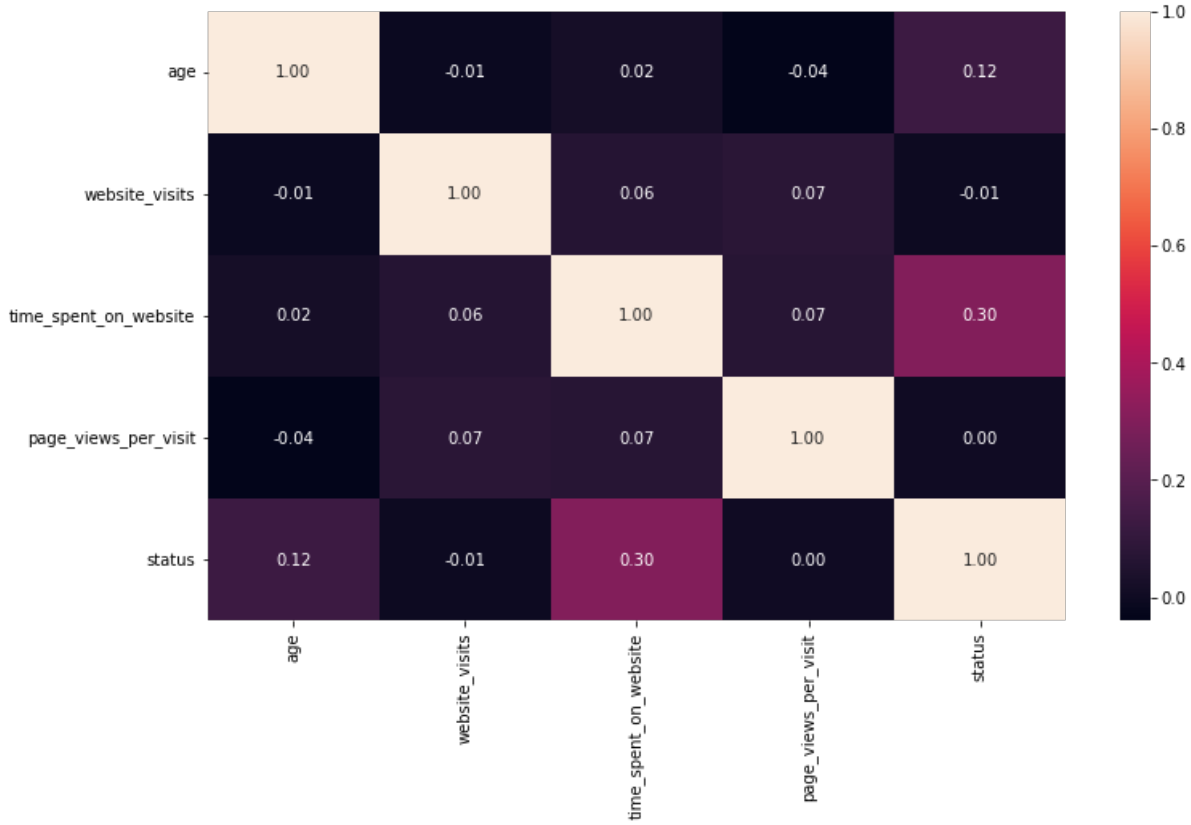
Observations:

- There are a very less number of referrals but the conversion is high.
- Company should try to get more leads through referrals by promoting rewards for existing customer base when they refer someone.

Check to see the pairwise correlations between all the numerical variables.

In [130]:

```
plt.figure(figsize = (12, 7))

sns.heatmap(data.corr(), annot = True, fmt = ".2f")

plt.show()
```



Observations:

- There's a weak positive correlation between status and time spent on the website. This implies that a person spending more time on the website is more likely to be converted.
- There's no correlation between any independent variable.

Data preparation for modeling

- Predict which lead is more likely to be converted.
- Before building a model, check to encode categorical features.
- Split the data into train and test sets to be able to evaluate the model that we build on the train data.

In [131]:

```
# Separating the target variable and other variables
X = data.drop(columns = 'status')

Y = data['status']
```

In [132]:

```
# Creating dummy variables, drop_first=True is used to avoid redundant variables
X = pd.get_dummies(X, drop_first = True)
```

In [133]:

```
# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.30, random_state = 1)
```

Checking the shape of the train and test data

In [134]:

```
print("Shape of the training set: ", X_train.shape)

print("Shape of the test set: ", X_test.shape)

print("Percentage of classes in the training set:")

print(y_train.value_counts(normalize = True))

print("Percentage of classes in the test set:")

print(y_test.value_counts(normalize = True))
```

```
Shape of the training set:  (3228, 16)
Shape of the test set:  (1384, 16)
Percentage of classes in the training set:
0    0.704151
1    0.295849
Name: status, dtype: float64
Percentage of classes in the test set:
0    0.695087
1    0.304913
Name: status, dtype: float64
```

Building Classification Models

Before training the model, choose the appropriate model evaluation criterion as per the problem at hand.

Model evaluation criterion

Model can make wrong predictions as:

1. Predicting a lead will not be converted to a paid customer but, in reality, the lead would have converted to a paid customer.
2. Predicting a lead will be converted to a paid customer but, in reality, the lead would have not converted to a paid customer.

Which case is more important?

- If we predict that a lead will not get converted and the lead would have converted then the company will lose a potential customer.
- If we predict that a lead will get converted and the lead doesn't get converted the company might lose resources by nurturing false-positive cases.

Losing a potential customer is a greater loss for the organization.

How to reduce the losses?

- Company would want Recall to be maximized. The greater the Recall score, higher the chances of minimizing False Negatives.

create a function to calculate and print the classification report and confusion matrix so that I don't have to rewrite the same code repeatedly for each model.

In [135]:

```
# Function to print the classification report and get confusion matrix in a proper format

def metrics_score(actual, predicted):
    print(classification_report(actual, predicted))

    cm = confusion_matrix(actual, predicted)

    plt.figure(figsize = (8, 5))

    sns.heatmap(cm, annot = True,  fmt = '.2f', xticklabels = ['Not Converted', 'Converted'], yticklabels = ['Not Converted', 'Converted'])

    plt.ylabel('Actual')

    plt.xlabel('Predicted')

    plt.show()
```

Decision Tree

- **Fit the decision tree classifier on the training data (use random_state=7)**
- **Check the performance on both training and testing datasets (use metrics_score function)**

In [136]:

```
# Fitting the decision tree classifier on the training data
d_tree =  DecisionTreeClassifier(class_weight = {0: 0.3, 1: 0.7}, random_state = 7)

d_tree.fit(X_train, y_train)
```

Out[136]:

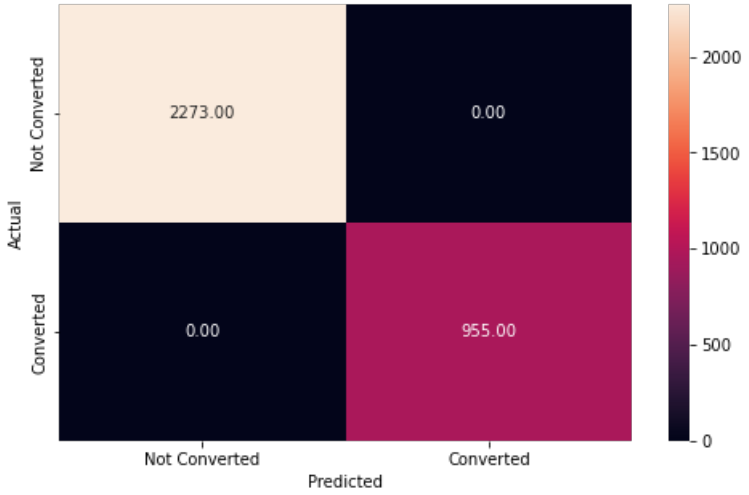
```
DecisionTreeClassifier(class_weight={0: 0.3, 1: 0.7}, random_state=7)
```

check the performance on the training data

In [137]:

```
# Checking performance on the training data
y_pred_train1 = d_tree.predict(X_train)

metrics_score(y_train, y_pred_train1)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2273
           1       1.00      1.00      1.00       955

    accuracy                           1.00      3228
   macro avg       1.00      1.00      1.00      3228
weighted avg       1.00      1.00      1.00      3228
```



**Observations:

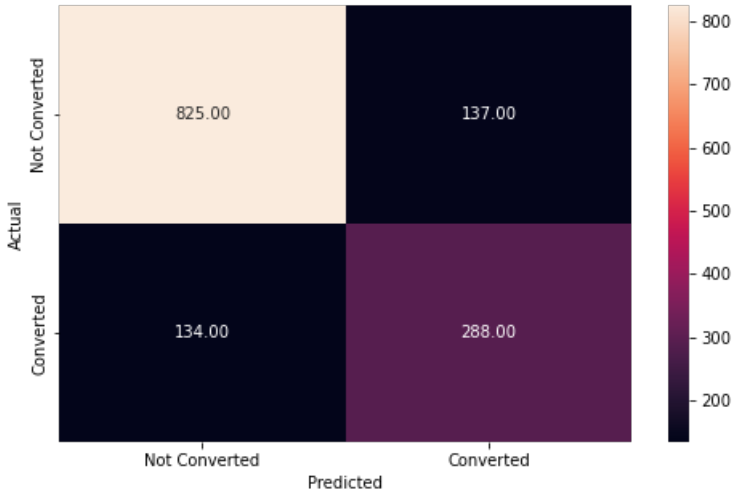
--The Decision tree is giving a 100% score for all metrics on the training dataset which means the model is overfitting(low bias, high variance)

check the performance on test data to see if the model is overfitting.

In [138]:

```
# Checking performance on the testing data
y_pred_test1 = d_tree.predict(X_test)

metrics_score(y_test, y_pred_test1)
```

```
              precision    recall  f1-score   support

           0       0.86      0.86      0.86       962
           1       0.68      0.68      0.68       422

    accuracy                           0.80      1384
   macro avg       0.77      0.77      0.77      1384
weighted avg       0.80      0.80      0.80      1384
```



**Observations:

--The Decision Tree works well on the training data but not so well on the test data.

--The recall on the test data with 86% does better in predicting leads was not converted to paid customers. The recall is not so good in predicting leads that will be converted to paid customers (68%) i.e. 32% chance it will not predict leads that will be converted to paid customers; this needs to be improved.

Try hyperparameter tuning using GridSearchCV to find the optimal max_depth to reduce overfitting of the model, then tune some other hyperparameters as well.

Decision Tree - Hyperparameter Tuning

Use the class_weight hyperparameter with the value equal to {0: 0.3, 1: 0.7} which is approximately the opposite of the imbalance in the original data.

This would tell the model that 1 is the important class here.

In [139]:

```
# Choose the type of classifier
d_tree_tuned = DecisionTreeClassifier(random_state = 7, class_weight = {0: 0.3, 1: 0.7})

# Grid of parameters to choose from
parameters = {'max_depth': np.arange(2, 10),
              'criterion': ['gini', 'entropy'],
              'min_samples_leaf': [5, 10, 20, 25]
             }

# Type of scoring used to compare parameter combinations - recall score for class 1
scorer = metrics.make_scorer(recall_score, pos_label = 1)

# Run the grid search
grid_obj = GridSearchCV(d_tree_tuned, parameters, scoring = scorer, cv = 5)

grid_obj = grid_obj.fit(X_train, y_train)

# Set the classifier to the best combination of parameters
d_tree_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data
d_tree_tuned.fit(X_train, y_train)
```

Out[139]:

```
DecisionTreeClassifier(class_weight={0: 0.3, 1: 0.7}, criterion='entropy',
                       max_depth=3, min_samples_leaf=5, random_state=7)
```

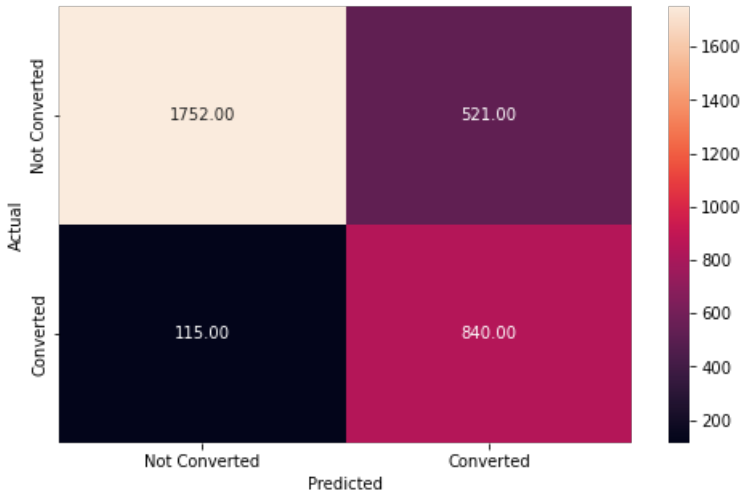
After tuning, check the model performance on the training and testing data.

- **Check the performance on both training and testing datasets**
- **Compare the results with the results from the decision tree model with default parameters**

In [140]:

```
# Checking performance on the training data
y_pred_train2 = d_tree_tuned.predict(X_train)
metrics_score(y_train, y_pred_train2)
```

```
              precision    recall  f1-score   support

           0       0.94      0.77      0.85      2273
           1       0.62      0.88      0.73       955

    accuracy                           0.80      3228
   macro avg       0.78      0.83      0.79      3228
weighted avg       0.84      0.80      0.81      3228
```



**Observations:

-- Decision tree with default parametes is more fitted compared to the model after hyperparameter tuning; i.e. the hyperparamter tuning model is not overfitting and now we have reduced over overfitting.

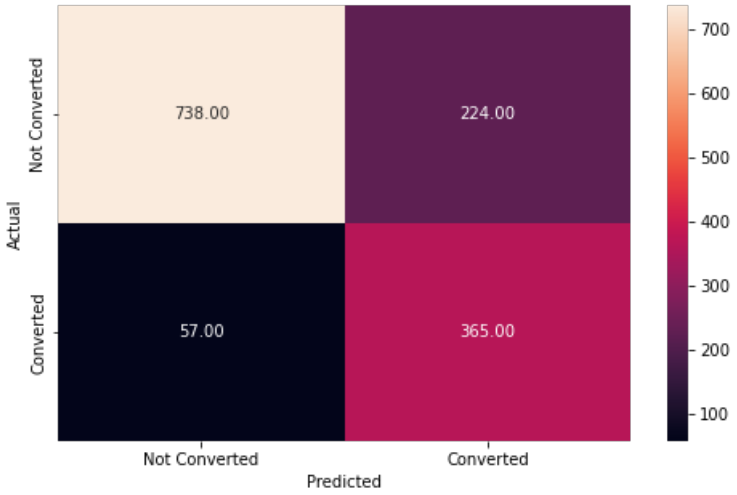
Check the model performance on the testing data

In [141]:

```
# Checking performance on the testing data
```

```
y_pred_test2 = d_tree_tuned.predict(X_test)

metrics_score(y_test, y_pred_test2)
```

```
              precision    recall  f1-score   support

           0       0.93      0.77      0.84       962
           1       0.62      0.86      0.72       422

    accuracy                           0.80      1384
   macro avg       0.77      0.82      0.78      1384
weighted avg       0.83      0.80      0.80      1384
```



**Observations:

-- training and test results using hyperparameter tuning are giving similar values..

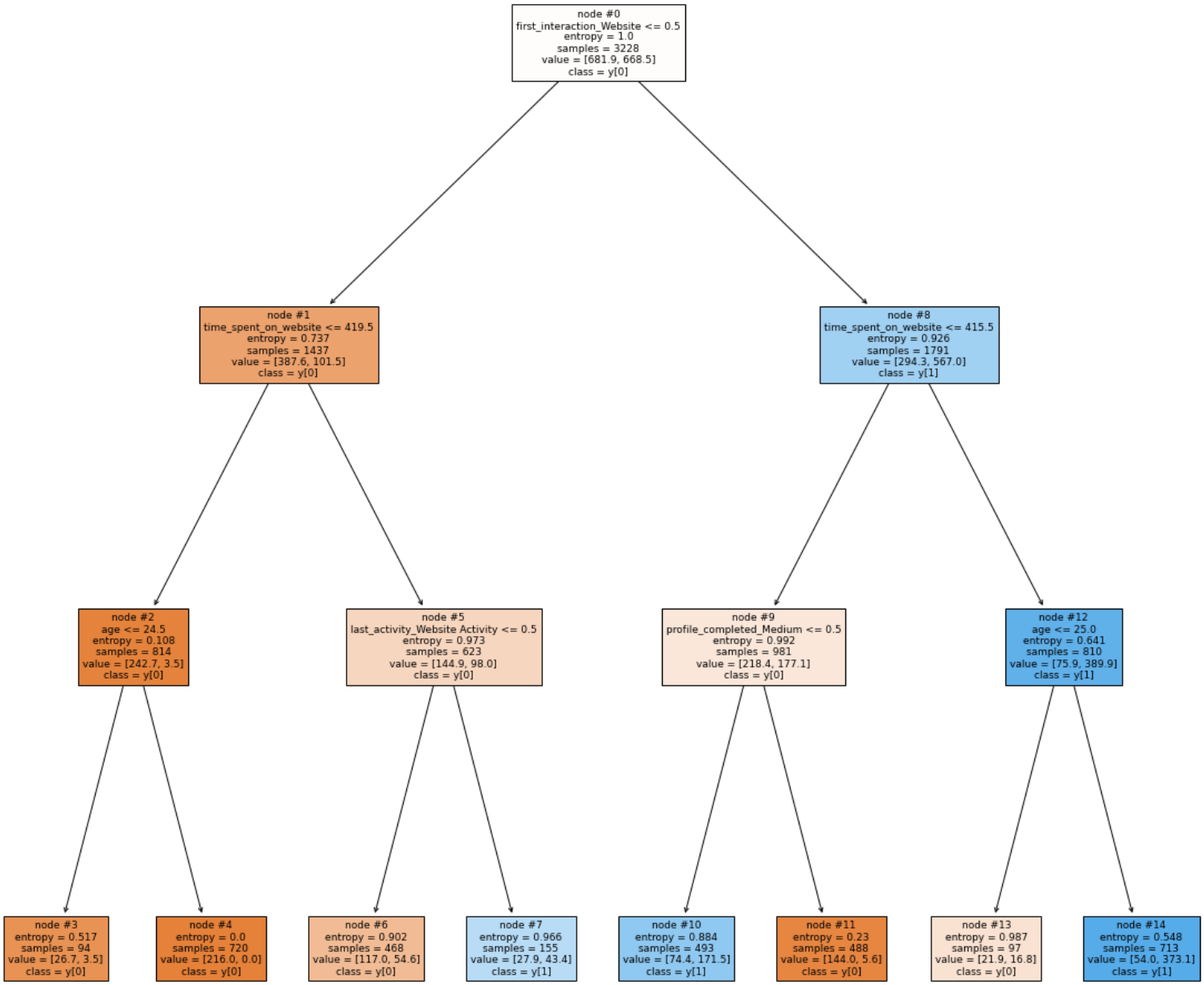
-- Though the recall for predicting the leads that won't be converted to customers decreased, The recall in the test data using hyperparameter tuning gave better results for leads that would be converted to customers compared to Decision tree test model using default parameters. The avg recall increased using hyperparametrs compared to decision tree test.

Visualize the tuned decision tree and observe the decision rules:

Observations from the below visualization of the tuned decision tree

In [142]:

```
features = list(X.columns)

plt.figure(figsize = (20, 20))

tree.plot_tree(d_tree_tuned, feature_names = features, filled = True, fontsize = 9, node_ids = True, class_names = True)

plt.show()
```



Note: Blue leaves represent the converted leads, i.e., **y[1]**, while the orange leaves represent the not converted leads, i.e., **y[0]**. Also, the more the number of observations in a leaf, the darker its color gets.

**Observations:

-- the split is based on First_interaction_with_website. It is one of the most important factors when considering whether the lead will be converted to a paid customer or not, this could be because leads had easier access to the website than mobile apps.

-- leads who spend more time on the website and there ages are above 25yrs are more likely to be converted to paid customers

-- Even leads whose first interaction is based on other form (mobile app), spend less time on website and their age are above 24.5 yrs still have a high chance of not converting to paid customers..

--leads whose first interaction is with website, spend time of less than 415.5 on website and don't have upto medium completed profiles are less likely to be converted to paid customers.

Look at the feature importance of the tuned decision tree model

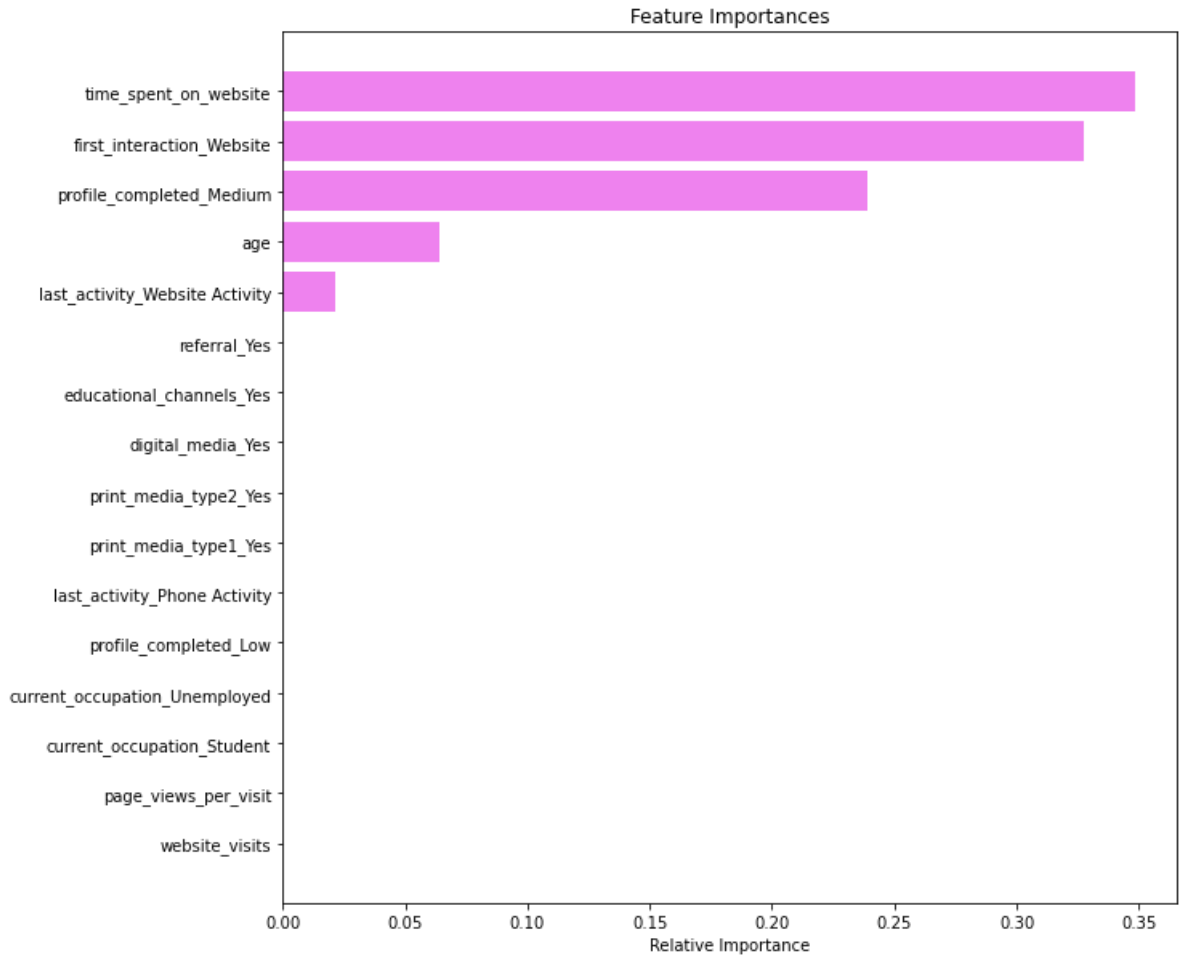
In [143]:

```
# Importance of features in the tree building

print (pd.DataFrame(d_tree_tuned.feature_importances_, columns = ["Imp"], index = X_train.columns).sort_values(by = 'Imp', ascending = False))
```

	Imp
time_spent_on_website	0.348142
first_interaction_Website	0.327181
profile_completed_Medium	0.239274
age	0.063893
last_activity_Website Activity	0.021511
website_visits	0.000000
page_views_per_visit	0.000000
current_occupation_Student	0.000000
current_occupation_Unemployed	0.000000
profile_completed_Low	0.000000
last_activity_Phone Activity	0.000000
print_media_type1_Yes	0.000000
print_media_type2_Yes	0.000000
digital_media_Yes	0.000000
educational_channels_Yes	0.000000
referral_Yes	0.000000

In [144]:

```
# Plotting the feature importance
importances = d_tree_tuned.feature_importances_

indices = np.argsort(importances)

plt.figure(figsize = (10, 10))

plt.title('Feature Importances')

plt.barh(range(len(indices)), importances[indices], color = 'violet', align = 'center')

plt.yticks(range(len(indices)), [features[i] for i in indices])

plt.xlabel('Relative Importance')

plt.show()
```



Observations:

- **Time spent on the website and first_interaction_website are the most important features followed by profile_completed, age, and last_activity**.
- **The rest of the variables have no impact in this model, while deciding whether a lead will be converted or not** .

Random Forest Classifier

- **Fit the random forest classifier on the training data (use random_state=7)**
- **Check the performance on both training and testing data (use metrics_score function)**

In [145]:

```
# Fitting the random forest tree classifier on the training data
rf_estimator = RandomForestClassifier(criterion= 'entropy', random_state = 7)

rf_estimator.fit(X_train, y_train)
```

Out[145]:

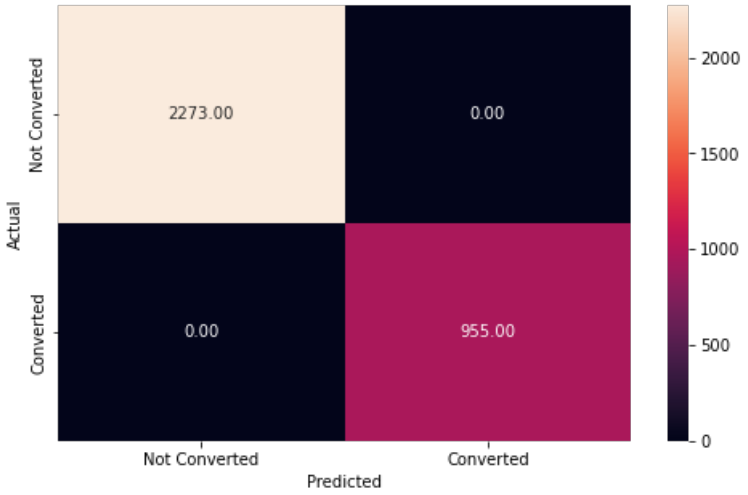
```
RandomForestClassifier(criterion='entropy', random_state=7)
```

Check the performance of the model on the training data

In [146]:

```
# Checking performance on the training data
y_pred_train3 = rf_estimator.predict(X_train)

metrics_score(y_train, y_pred_train3)
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2273
           1       1.00      1.00      1.00       955

    accuracy                           1.00      3228
   macro avg       1.00      1.00      1.00      3228
weighted avg       1.00      1.00      1.00      3228
```



**Observations:

-- Random Forest model on train data shows a 100% accuracy which means the model is over fitting on train data

Check the performance on the testing data

In [147]:

```
# Checking performance on the testing data
y_pred_test3 = rf_estimator.predict(X_test)

metrics_score(y_test, y_pred_test3)
```

```
              precision    recall  f1-score   support

           0       0.88      0.93      0.90       962
           1       0.81      0.70      0.75       422

    accuracy                           0.86      1384
   macro avg       0.84      0.81      0.83      1384
weighted avg       0.86      0.86      0.86      1384
```

**Observations:

--Precision, recall, accuracy are all reduced for the test data and it is no more overfitted. The recall does better predicting leads that won't convert to paid customer than leads that would convert to paid customer.

Check to see if one can get a better model by tuning the random forest classifier

Random Forest Classifier - Hyperparameter Tuning

Try **tuning some of the important hyperparameters of the Random Forest Classifier** .

Don't tune the criterion hyperparameter as we know from hyperparameter tuning for decision trees that entropy is a better splitting criterion for this data.

In [148]:

```
# Choose the type of classifier
rf_estimator_tuned = RandomForestClassifier(criterion = "entropy", random_state = 7)

# Grid of parameters to choose from
parameters = {"n_estimators": [100, 110, 120],
    "max_depth": [5, 6, 7],
    "max_features": [0.8, 0.9, 1]
          }

# Type of scoring used to compare parameter combinations - recall score for class 1
scorer = metrics.make_scorer(recall_score, pos_label = 1)

# Run the grid search
grid_obj = GridSearchCV(rf_estimator_tuned, parameters, scoring = scorer, cv = 5)

grid_obj = grid_obj.fit(X_train, y_train)

# Set the classifier to the best combination of parameters
rf_estimator_tuned = grid_obj.best_estimator_
```

In [149]:

```
# Fitting the best algorithm to the training data
rf_estimator_tuned.fit(X_train, y_train)
```

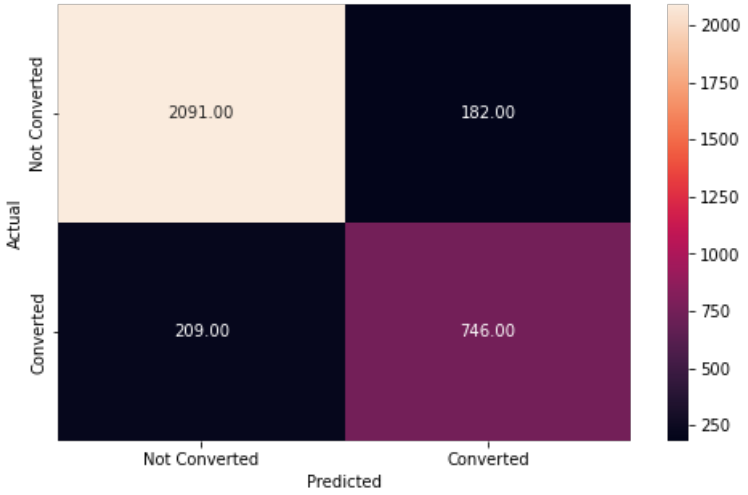
Out[149]:

```
RandomForestClassifier(criterion='entropy', max_depth=6, max_features=0.8,
                       n_estimators=110, random_state=7)
```

In [150]:

```
# Checking performance on the training data
y_pred_train4 = rf_estimator_tuned.predict(X_train)

metrics_score(y_train, y_pred_train4)
```

```
              precision    recall  f1-score   support

           0       0.91      0.92      0.91      2273
           1       0.80      0.78      0.79       955

    accuracy                           0.88      3228
   macro avg       0.86      0.85      0.85      3228
weighted avg       0.88      0.88      0.88      3228
```



Observations:

- After hyperparameter tuning, the model is performing poorly on the train data as well.
- Try adding some other hyperparameters and/or changing values of some hyperparameters to tune the model and see if we can get better performance.

Note: GridSearchCV can take a long time to run depending on the number of hyperparameters and the number of values tried for each hyperparameter. **Therefore, reduced the number of values passed to each hyperparameter.**

- **Tune the random forest classifier using GridSearchCV**
- **Check the performance on both training and testing datasets**
- **Compare the results with the results from the random forest model with default parameters and write your observations**

In [151]:

```
# Choose the type of classifier
rf_estimator_tuned = RandomForestClassifier(criterion = "entropy", random_state = 7)

# Grid of parameters to choose from
parameters = {"n_estimators": [110, 120],
    "max_depth": [6, 7],
    "min_samples_leaf": [20, 25],
    "max_features": [0.8, 0.9],
    "max_samples": [0.9, 1],
    "class_weight": [{0: 0.7, 1: 0.3}, "balanced", {0: 0.4, 1: 0.1}]
          }

# Type of scoring used to compare parameter combinations - recall score for class 1
scorer = metrics.make_scorer(recall_score, pos_label = 1)

# Run the grid search on the training data using scorer=scorer and cv=5
grid_obj = GridSearchCV(rf_estimator_tuned, parameters, scoring = scorer, cv = 5)

grid_obj = grid_obj.fit(X_train, y_train)

# Save the best estimator to variable rf_estimator_tuned
rf_estimator_tuned = grid_obj.best_estimator_

#Fit the best estimator to the training data
rf_estimator_tuned.fit(X_train, y_train)
```

Out[151]:

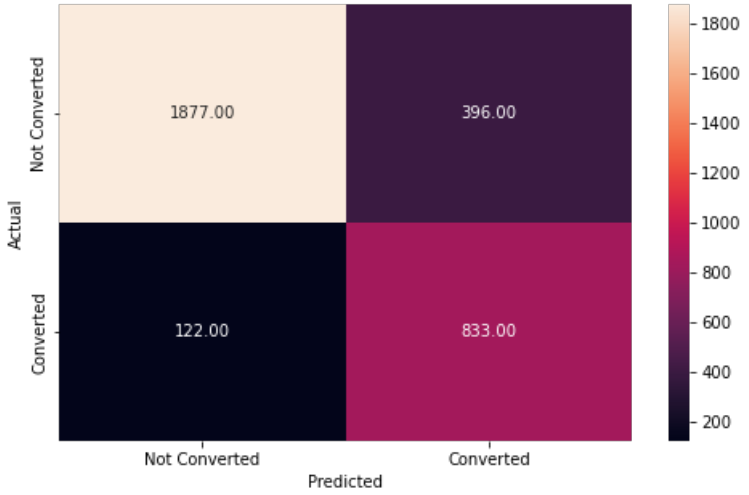
```
RandomForestClassifier(class_weight='balanced', criterion='entropy',
                       max_depth=6, max_features=0.8, max_samples=0.9,
                       min_samples_leaf=25, n_estimators=120, random_state=7)
```

Check the performance of the tuned model

In [152]:

```
# Checking performance on the training data
y_pred_train5 = rf_estimator_tuned.predict(X_train)

metrics_score(y_train, y_pred_train5)
```

	precision	recall	f1-score	support
0	0.94	0.83	0.88	2273
1	0.68	0.87	0.76	955
accuracy			0.84	3228
macro avg	0.81	0.85	0.82	3228
weighted avg	0.86	0.84	0.84	3228



**Observations:

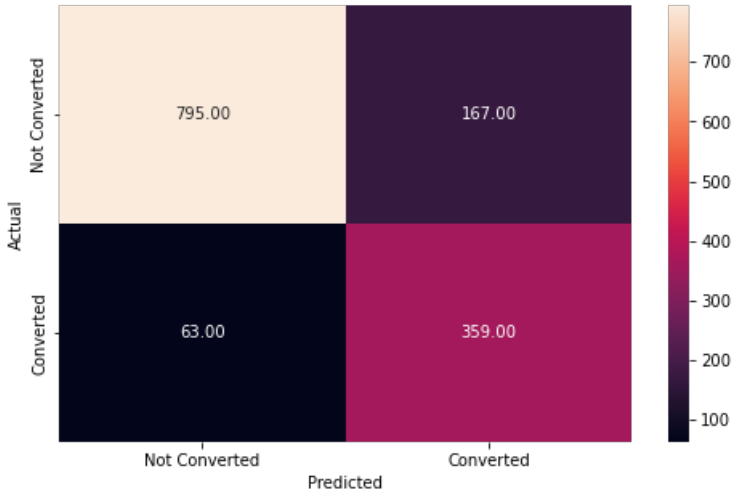
--This model using GridSearchCV doesn't yield a good performance on the training data compared to the random forest model with default parameters.

check the model performance on the test data

In [153]:

```
# Checking performance on the test data
y_pred_test5 = rf_estimator_tuned.predict(X_test)

metrics_score(y_test, y_pred_test5)
```

	precision	recall	f1-score	support
0	0.93	0.83	0.87	962
1	0.68	0.85	0.76	422
accuracy			0.83	1384
macro avg	0.80	0.84	0.82	1384
weighted avg	0.85	0.83	0.84	1384



**Observations:

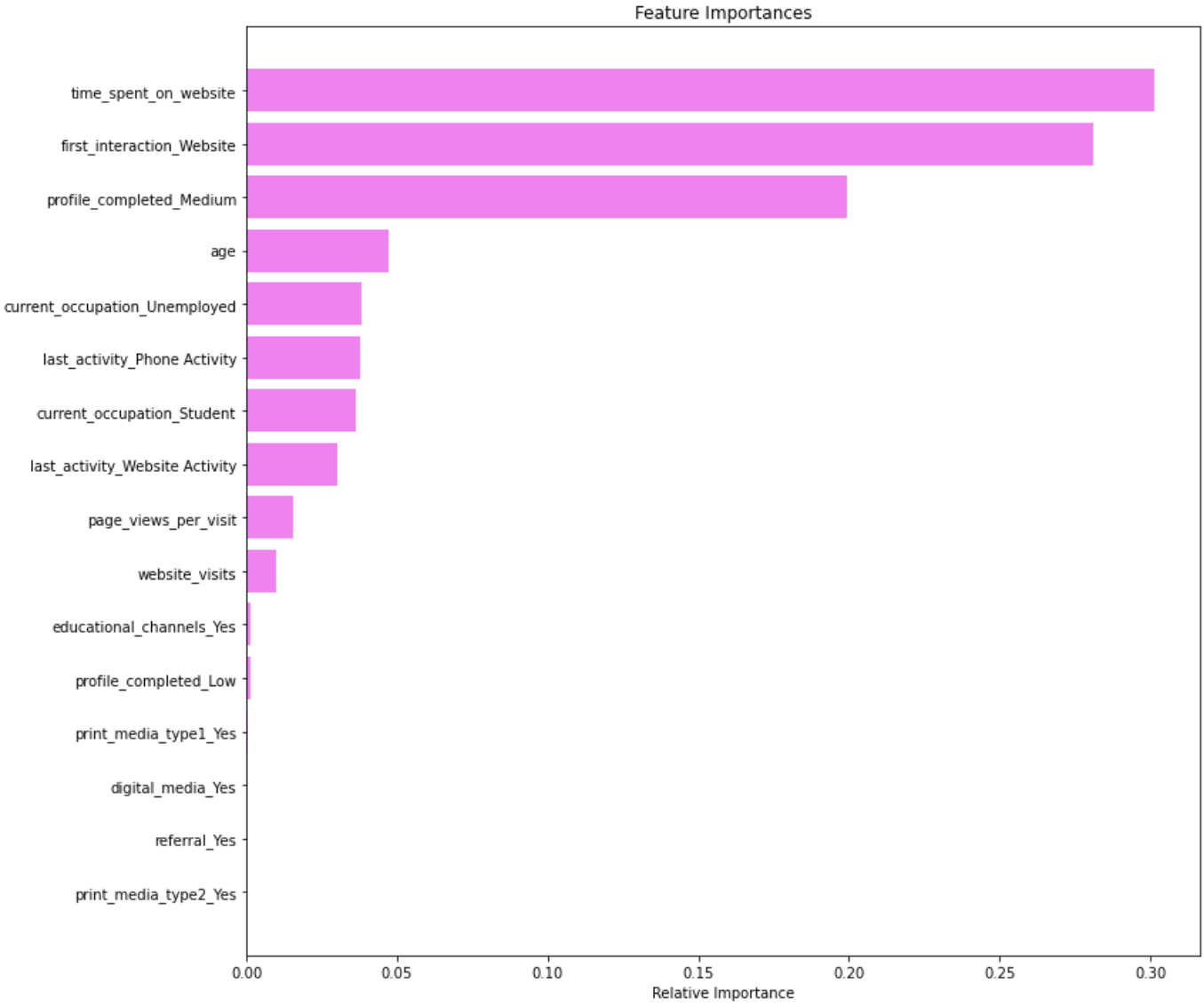
--Comparing Random Forest with default parameters with GridSearchCv on test dat, GridSearchCv on test data has a lower accuracy, the recall is higher, for leads converted to paid customers..

One of the drawbacks of ensemble models is that you lose the ability to obtain an interpretation of the model. We cannot observe the decision rules for random forests the way we did for decision trees, so check the feature importance of the model.

In [154]:

```
importances = rf_estimator_tuned.feature_importances_

indices = np.argsort(importances)

feature_names = list(X.columns)
```

```
plt.figure(figsize = (12, 12))

plt.title('Feature Importances')

plt.barh(range(len(indices)), importances[indices], color = 'violet', align = 'center')

plt.yticks(range(len(indices)), [feature_names[i] for i in indices])

plt.xlabel('Relative Importance')

plt.show()
```



Observations:

- Similar to the decision tree model, **time spent on website, first_interaction_website, profile_completed, and age are the top four features** that help distinguish between not converted and converted leads.
- Unlike the decision tree, **the random forest gives some importance to other variables like occupation, page_views_per_visit, as well.** This implies that the random forest is giving importance to more factors in comparison to the decision tree.

Conclusions:

--The model created by GridSearchCv gives a better average Recall performance compared to default parameters of Random Forest model.

--Edtech Industry can use this model to determine leads that will be converted to paid customers or not

--time_spent_on_website, first_interaction_website, profile_completed are the most important factors Edtech Industry needs to pay attention to drive better conversion.

****Business Recommendations**

--Edtech Industries should allocate more resources on leads whose first interaction are through the website; new strategies should be put in place to improve first interaction by mobile as there seems to be low conversion of leads using this medium.

--leads who spent more time on the website are likely to be those who are converting to paid customers; so what Edtech Industries can do is make the website interface more user friendly, with lesser time during sign up.

-- Edtech Industries should find ways to target more professionals and students; resources spent on leads of unemployed should be totally reduced because they are less likely to have the money to convert to a paid customer.. Or unemployed leads could be given a discount just as a form of encouragement.

--Edtech Industries should find out the reasons why leads abandon profiles during signup/filling even when they have completed more than half of the profile, they seem to less reluctant to be converted to paid customers.

--Finally the company can explore other means of interacting with leads aside websites or mobile app; carry out surveys/questionaires to find out the challenges or attractions the leads encountered before they decided to become paid customers.

In [ ]: