ExtraaLearn Project : Connor Rubenstein 9/28/23

Marks: 60

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.

In the present scenario due to the Covid-19, 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

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 regularly, 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

Data Description

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

Data Dictionary

- 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', '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: How many times has a lead 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 ExtraaLearn.
    - Email Activity: Seeking for details about the program through email, Representative shared information with a lead like a brochure of program, etc
    - Phone Activity: Had a Phone Conversation with a representative, Had 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 ExtraaLearn in the Newspaper.
- print_media_type2: Flag indicating whether the lead had seen the ad of ExtraaLearn in the Magazine.
- digital_media: Flag indicating whether the lead had seen the ad of ExtraaLearn on the digital platforms.
- educational_channels: Flag indicating whether the lead had heard about ExtraaLearn in the education channels like online forums, discussion threads, educational websites, etc.
- referral: Flag indicating whether the lead had heard about ExtraaLearn through reference.
- status: Flag indicating whether the lead was converted to a paid customer or not.

Please read the instructions carefully before starting the project.

This is a commented Jupyter IPython Notebook file in which all the instructions and tasks to be performed are mentioned. Read along carefully to complete the project.

- Blanks '**___**' are provided in the notebook that needs to be filled with an appropriate code to get the correct result. Please replace the blank with the right code snippet. With every '**___**' blank, there is a comment that briefly describes what needs to be filled in the blank space.
- Identify the task to be performed correctly, and only then proceed to write the required code.
- Fill the code wherever asked by the commented lines like "# Fill in the blank" or "# Complete the code". Running incomplete code may throw an error.
- Remove the blank and state your observations in detail wherever the mark down says 'Write your observations here:**_**'
- Please run the codes in a sequential manner from the beginning to avoid any unnecessary errors.
- You can the results/observations derived from the analysis here and use them to create your final report.

In [1]:
```
import warnings

warnings.filterwarnings("ignore")
from statsmodels.tools.sm_exceptions import ConvergenceWarning

warnings.simplefilter("ignore", ConvergenceWarning)

# Libraries to help with reading and manipulating data

import pandas as pd
import numpy as np

# Library to split data
from sklearn.model_selection import train_test_split

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)
# Sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)
# setting the precision of floating numbers to 5 decimal points
pd.set_option("display.float_format", lambda x: "%.5f" % x)

# To build model for prediction
import statsmodels.stats.api as sms
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from statsmodels.tools.tools import add_constant
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn import metrics
```

```
from sklearn.ensemble import RandomForestClassifier

# To tune different models
from sklearn.model_selection import GridSearchCV


# To get diferent metric scores
from sklearn.metrics import (
    f1_score,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
    classification_report,
    precision_recall_curve,
    roc_curve,
    make_scorer,
)
```

Import Dataset

In [2]:
```
learn = pd.read_csv("ExtraaLearn.csv") ##  Complete the code to read the data
```

In [3]:
```
# copying data to another variable to avoid any changes to original data
data = learn.copy()
```

View the first and last 5 rows of the dataset

In [4]:
```
data.head()##  Complete the code to view top 5 rows of the data
```

Out[4]:

	ID	age	current_occupation	first_interaction	profile_completed	website_visits	time_s
0	EXT001	57	Unemployed	Website	High	7	
1	EXT002	56	Professional	Mobile App	Medium	2	
2	EXT003	52	Professional	Website	Medium	3	
3	EXT004	53	Unemployed	Website	High	4	
4	EXT005	23	Student	Website	High	4	

In [5]:
```
data.tail() ##  Complete the code to view last 5 rows of the data
```

Out[5]:

	ID	age	current_occupation	first_interaction	profile_completed	website_visits	ti
4607	EXT4608	35	Unemployed	Mobile App	Medium	15	
4608	EXT4609	55	Professional	Mobile App	Medium	8	
4609	EXT4610	58	Professional	Website	High	2	
4610	EXT4611	57	Professional	Mobile App	Medium	1	
4611	EXT4612	55	Professional	Website	Medium	4	

Understand the shape of the dataset

In [6]:
```
data.shape ## Complete the code to get the shape of data
```

Out[6]:
```
(4612, 15)
```

Check the data types of the columns for the dataset

In [7]:
```
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
```

In [8]:
```
# checking for duplicate values
data.duplicated().sum()## Complete the code to check duplicate entries in the c
```

Out[8]:
```
0
```

Exploratory Data Analysis

Let's check the statistical summary of the data.

In [9]:
```
data.describe() ##  Complete the code to print the statistical summary of the c
```

Out[9]:

	age	website_visits	time_spent_on_website	page_views_per_visit	status
count	4612.00000	4612.00000	4612.00000	4612.00000	4612.00000
mean	46.20121	3.56678	724.01127	3.02613	0.29857
std	13.16145	2.82913	743.82868	1.96812	0.45768
min	18.00000	0.00000	0.00000	0.00000	0.00000
25%	36.00000	2.00000	148.75000	2.07775	0.00000
50%	51.00000	3.00000	376.00000	2.79200	0.00000
75%	57.00000	5.00000	1336.75000	3.75625	1.00000
max	63.00000	30.00000	2537.00000	18.43400	1.00000

In [10]:
```
# Making a list of all catrgorical variables
cat_col = list(data.select_dtypes("object").columns)

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

```
EXT001     1
EXT2884    1
EXT3080    1
EXT3079    1
EXT3078    1
          ..
EXT1537    1
EXT1536    1
EXT1535    1
EXT1534    1
EXT4612    1
Name: ID, Length: 4612, dtype: int64
--------------------------------------------------
Professional    2616
Unemployed      1441
Student          555
Name: current_occupation, dtype: int64
--------------------------------------------------
Website       2542
Mobile App    2070
Name: first_interaction, dtype: int64
--------------------------------------------------
High      2264
Medium    2241
Low        107
Name: profile_completed, dtype: int64
--------------------------------------------------
Email Activity      2278
Phone Activity      1234
Website Activity    1100
Name: last_activity, dtype: int64
--------------------------------------------------
No     4115
Yes     497
Name: print_media_type1, dtype: int64
--------------------------------------------------
No     4379
Yes     233
Name: print_media_type2, dtype: int64
--------------------------------------------------
No     4085
Yes     527
Name: digital_media, dtype: int64
--------------------------------------------------
No     3907
Yes     705
Name: educational_channels, dtype: int64
--------------------------------------------------
No     4519
Yes      93
Name: referral, dtype: int64
--------------------------------------------------
```

In [11]:
```
# checking the number of unique values
data["ID"].nunique() # Complete the code to check the number of unique values
```

Out[11]:
```
4612
```

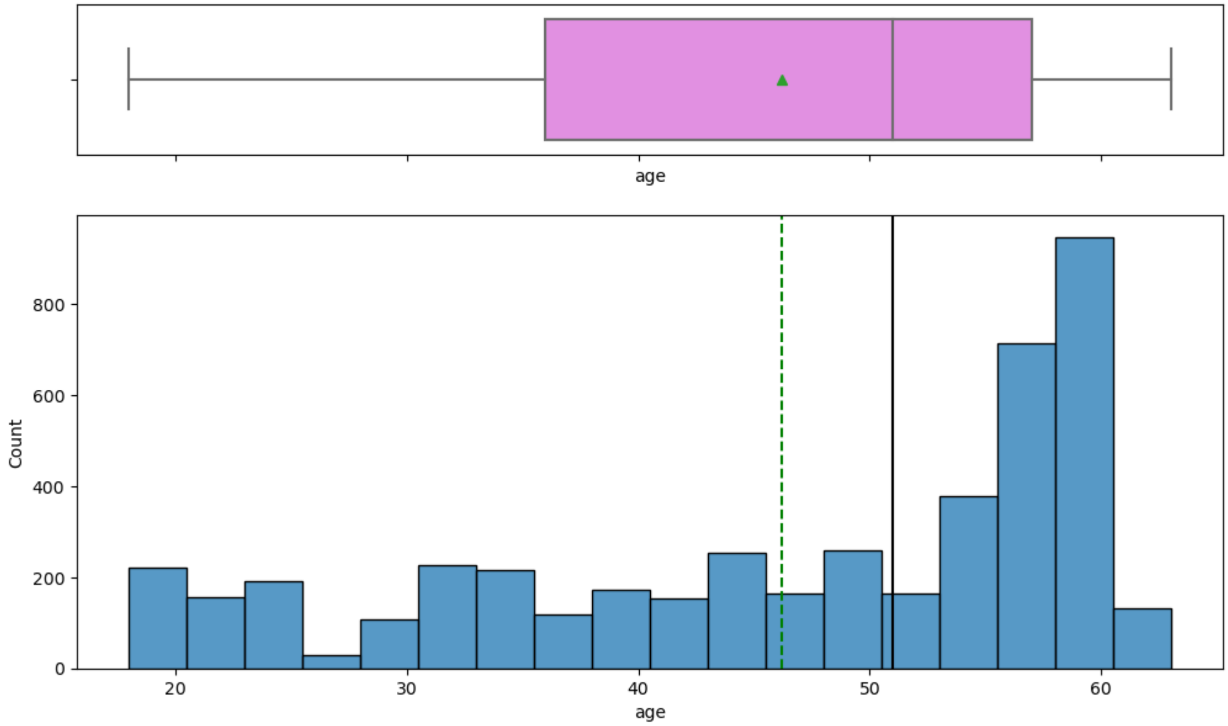
In [12]:
```
data.drop(["ID"], axis=1, inplace=True) # Complete the code to drop "ID" colum
```

Univariate Analysis

In [13]:
```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

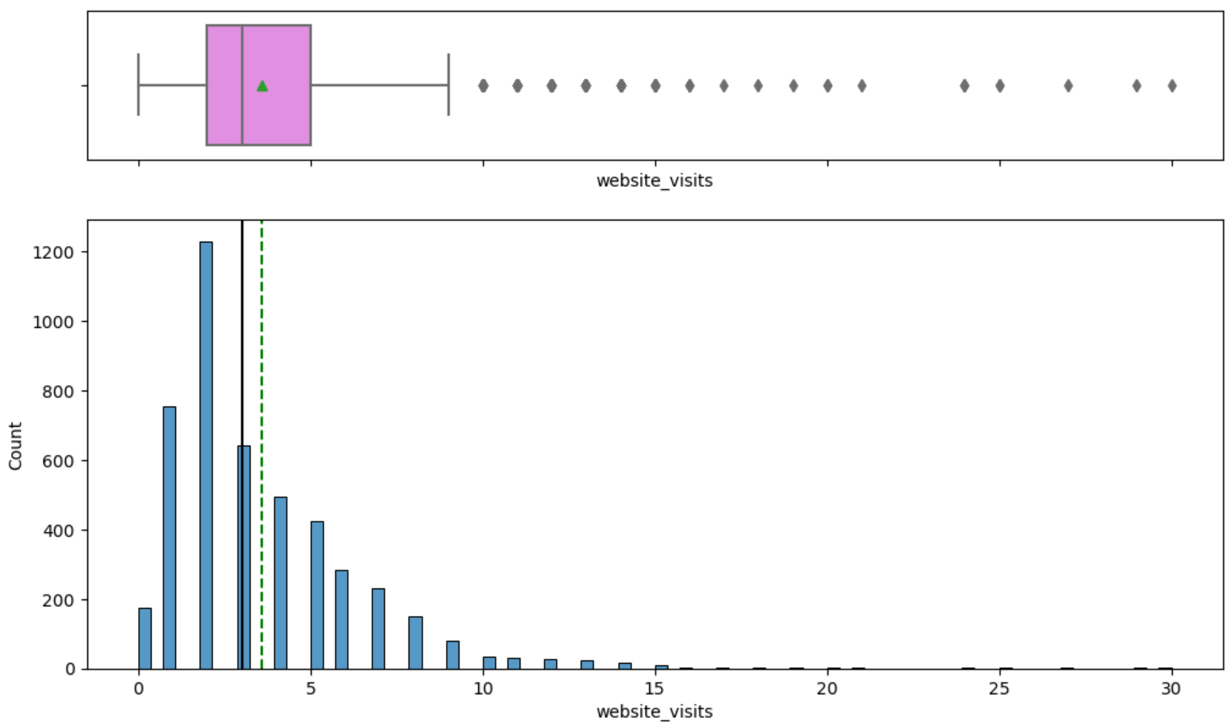
Observations on age

In [14]:
```
histogram_boxplot(data, "age")
```



Observations on website_visits

In [15]:
```
histogram_boxplot(data,"website_visits") # Complete the code to plot a histogra
```

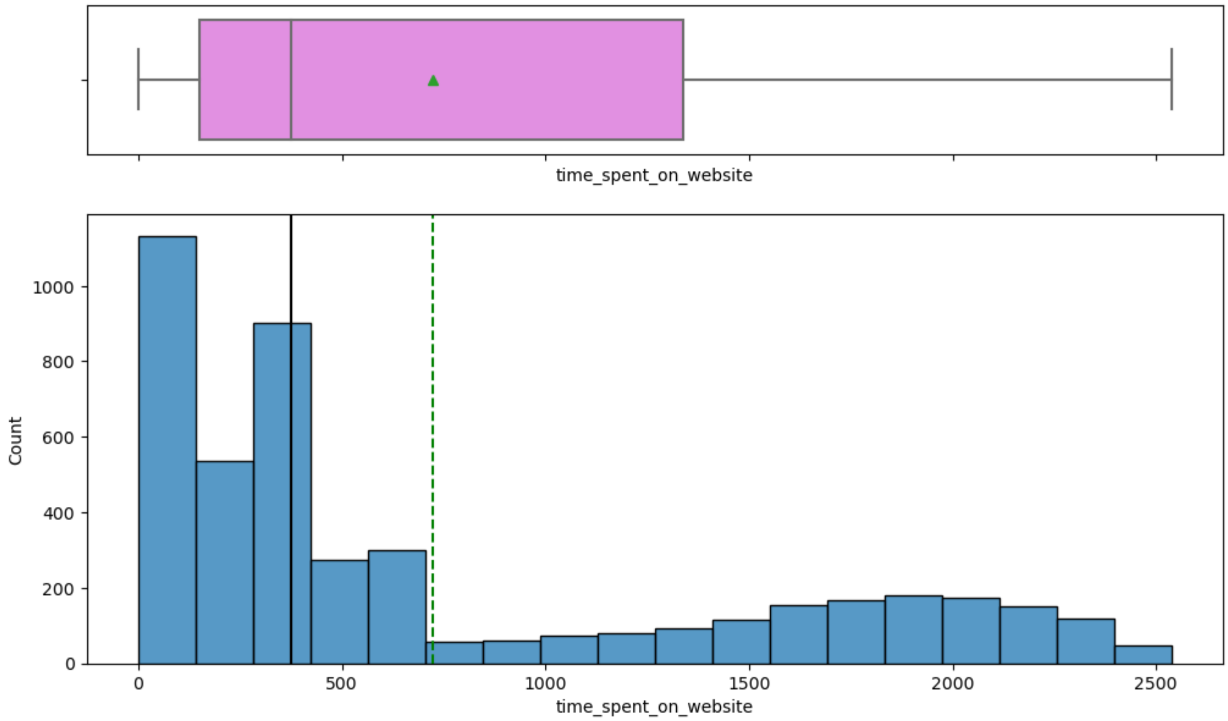


In [16]:
```
# To check how many leads have not visited web-site
data[data["website_visits"] == 0].shape
```

Out[16]:
```
(174, 14)
```

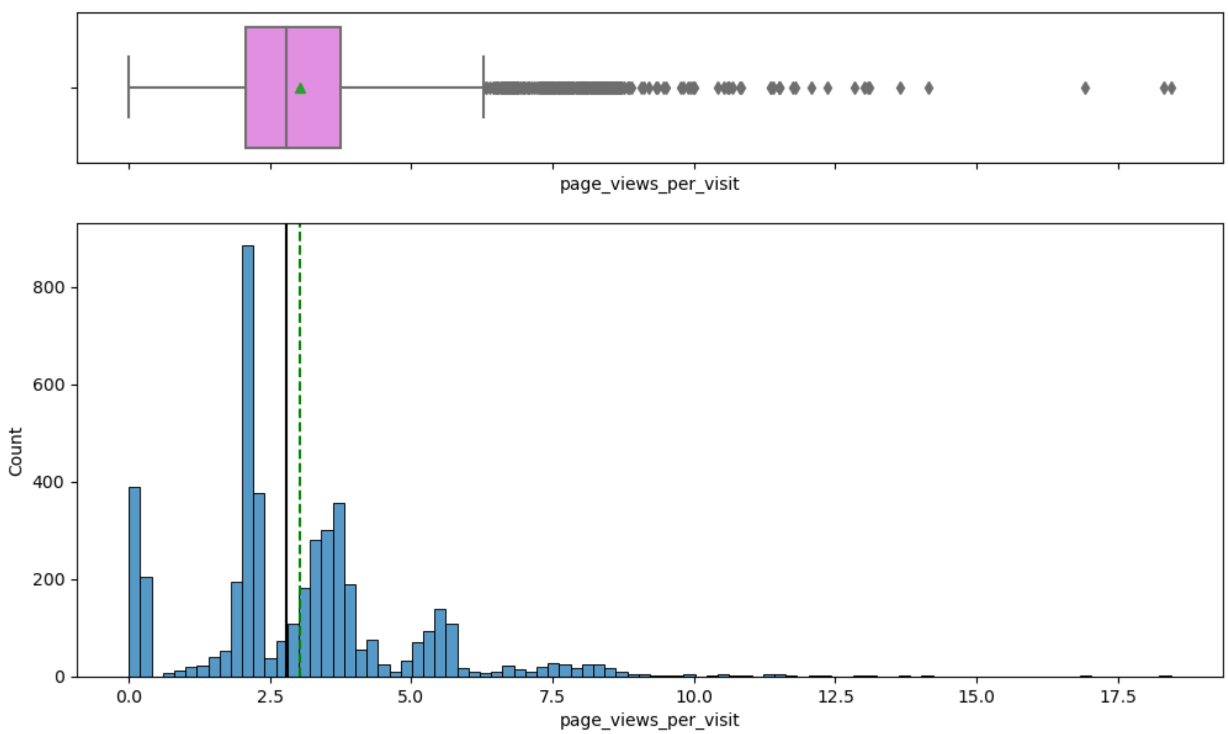
Observations on number of time_spent_on_website

In [17]:
```
histogram_boxplot(data,"time_spent_on_website") # Complete the code to plot a h
```



Observations on number of page_views_per_visit

In [18]:
```
histogram_boxplot(data,"page_views_per_visit") # Complete the code to plot a hi
```

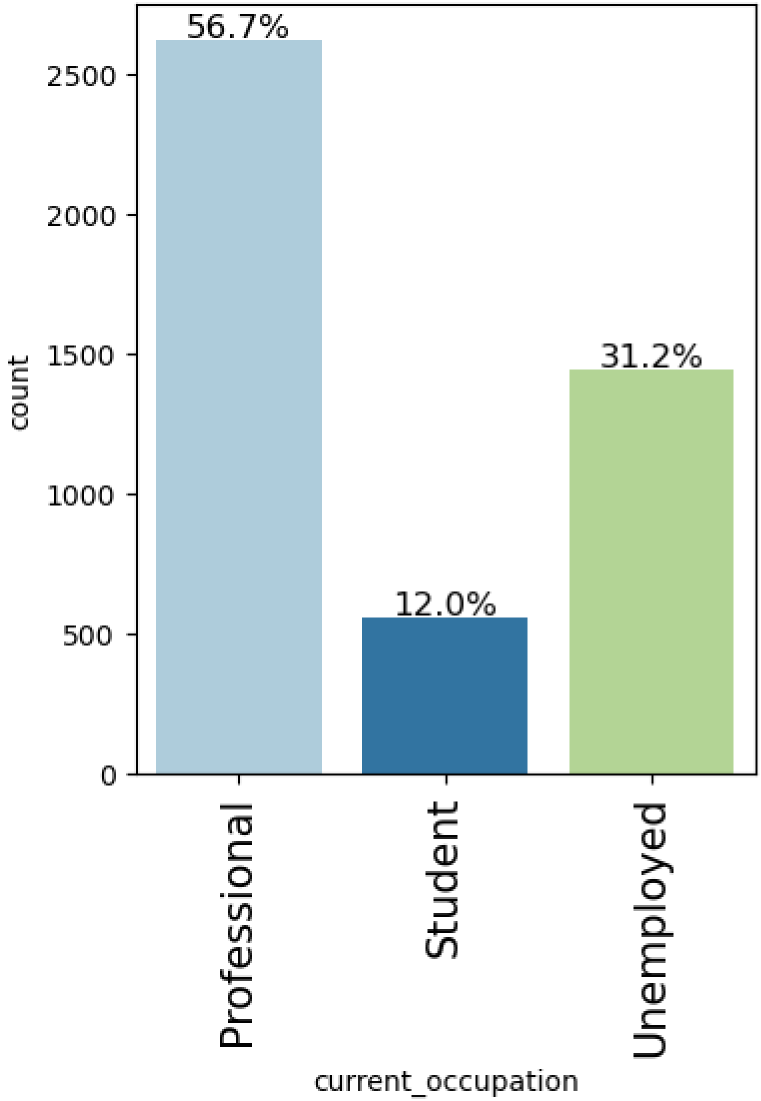


In [19]:
```
# function to create labeled barplots
```

```
def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all l
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

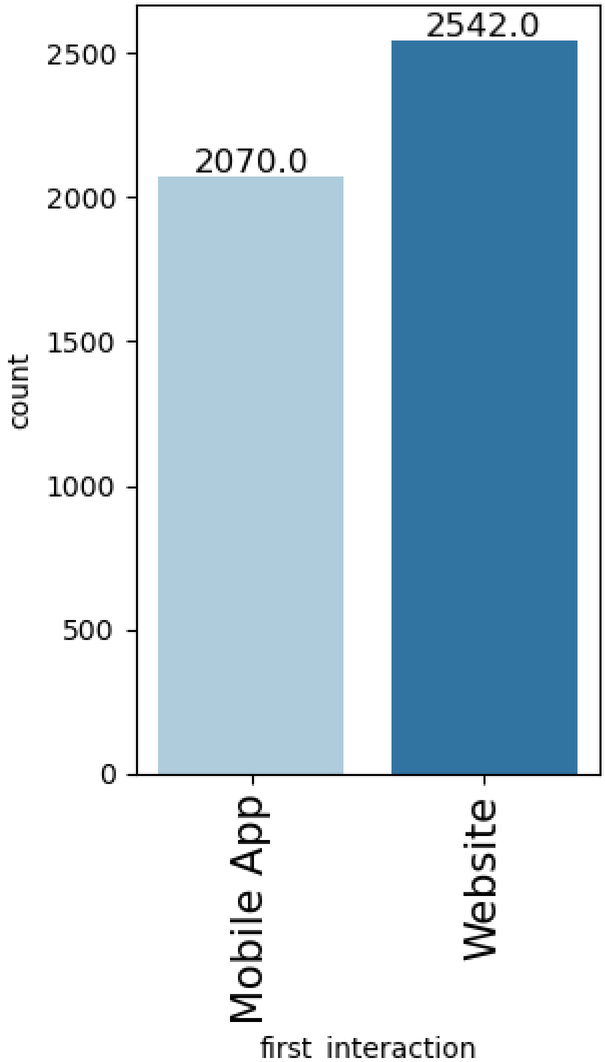
Observations on current_occupation

In [20]:
```
labeled_barplot(data, "current_occupation", perc=True)
```



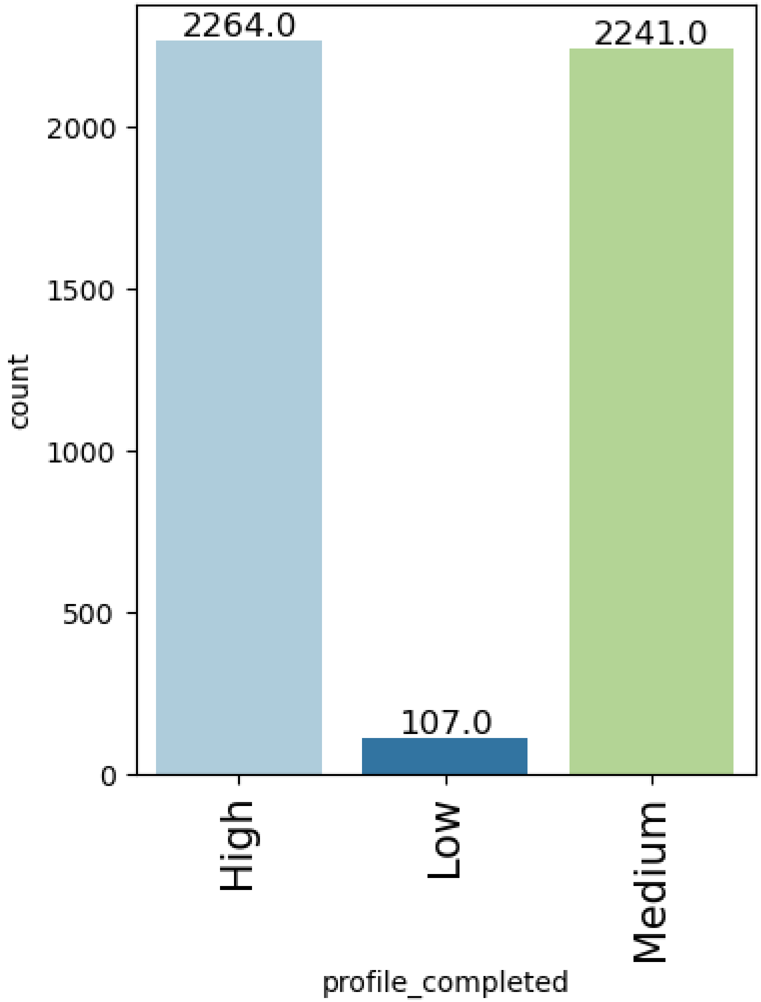
Observations on number of first_interaction

In [21]:
```
labeled_barplot(data,"first_interaction") # Complete the code to plot labeled_b
```



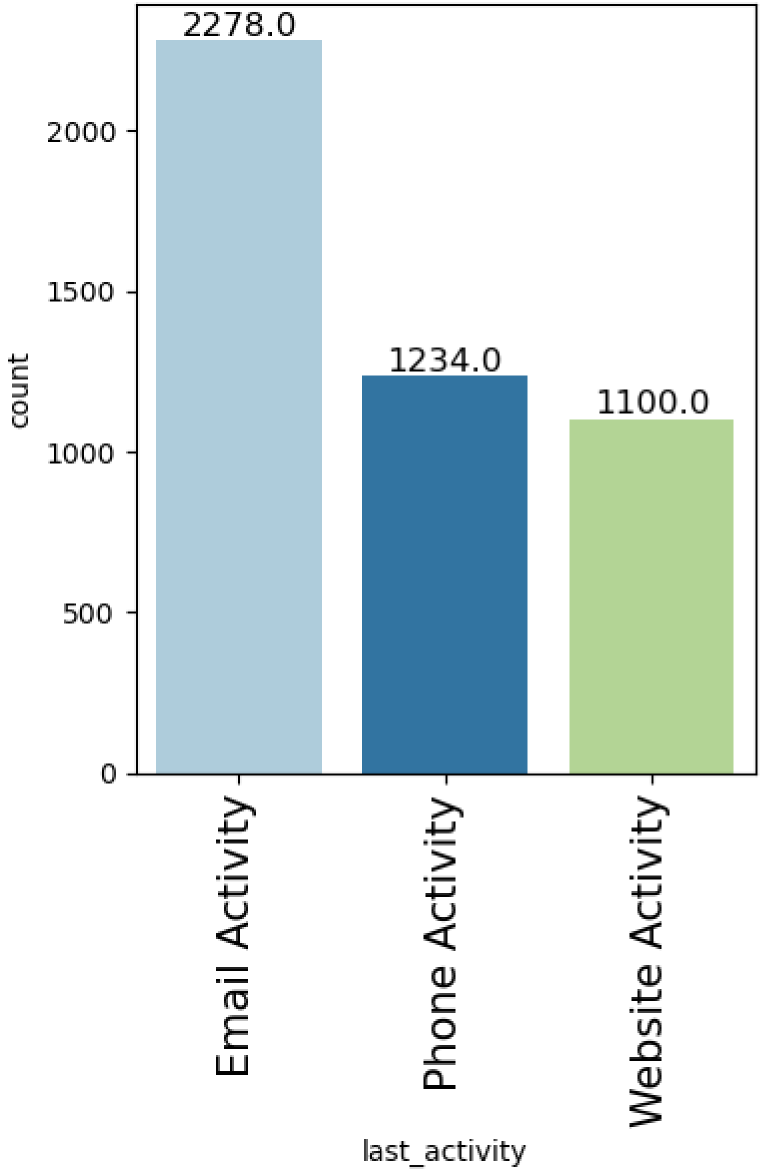
Observations on profile_completed

In [22]:
```
labeled_barplot(data,"profile_completed") # Complete the code to plot labeled_b
```



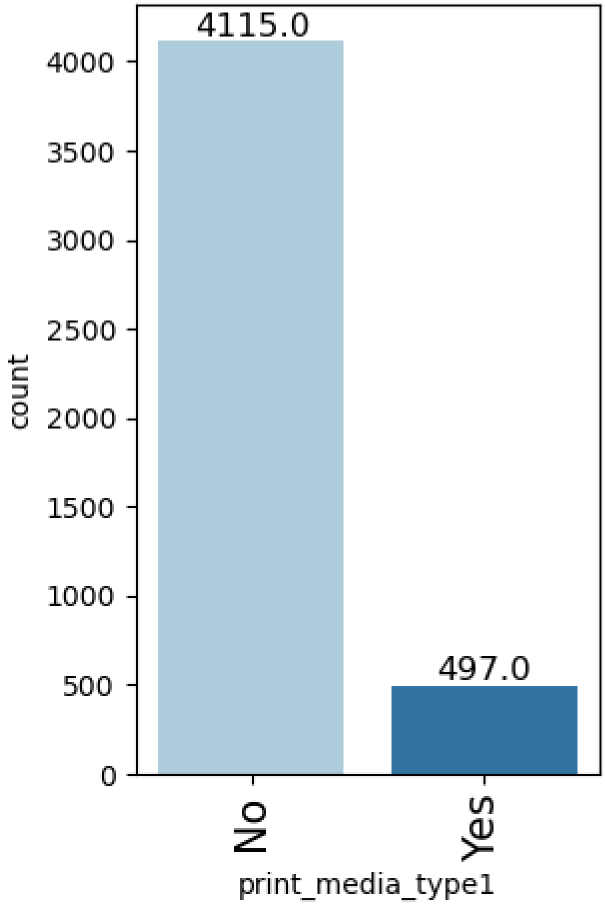
Observations on last_activity

In [23]:
```
labeled_barplot(data,"last_activity") # Complete the code to plot labeled_barpl
```



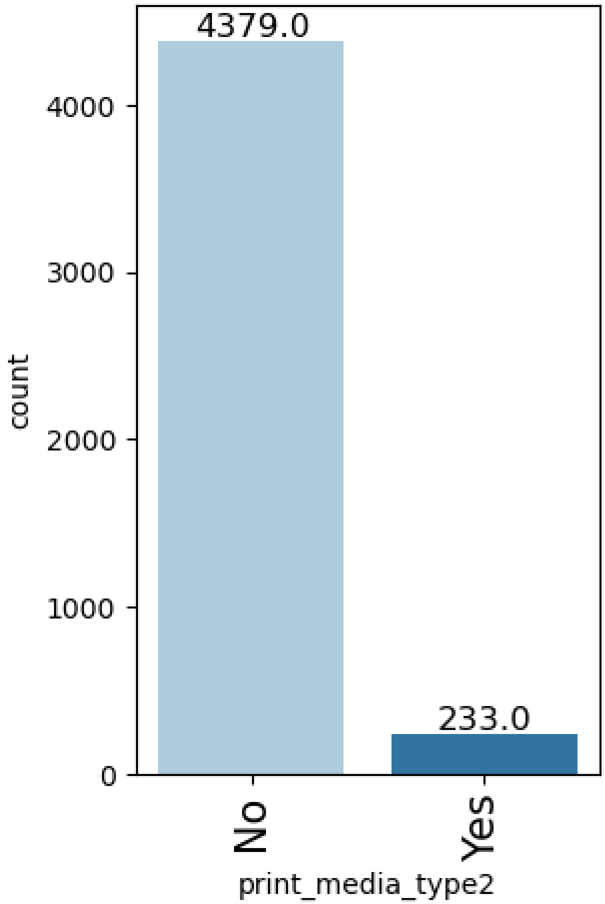
Observations on print_media_type1

In [24]:
```
labeled_barplot(data,"print_media_type1") # Complete the code to plot labeled_b
```



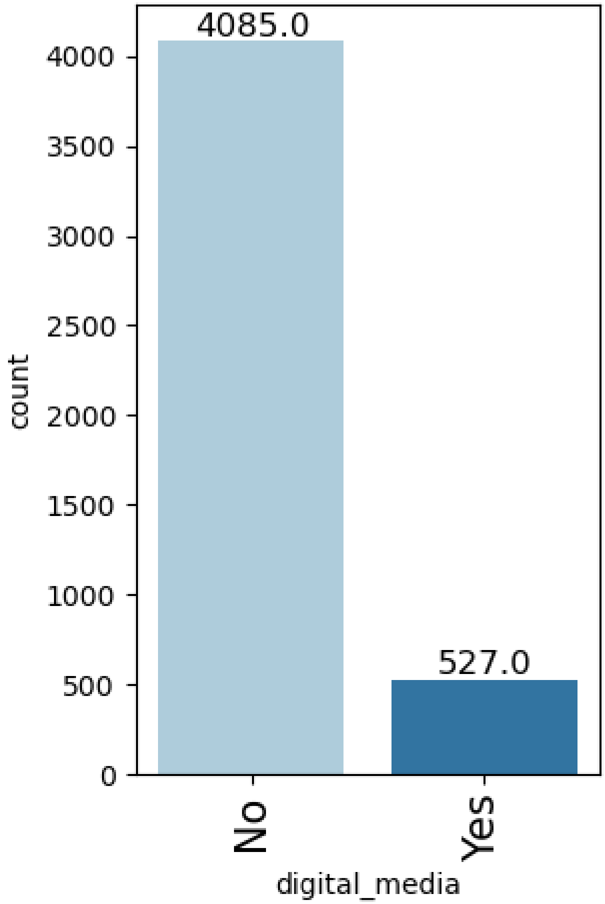
Observations on print_media_type2

In [25]:
```
labeled_barplot(data,"print_media_type2") # Complete the code to plot labeled_b
```



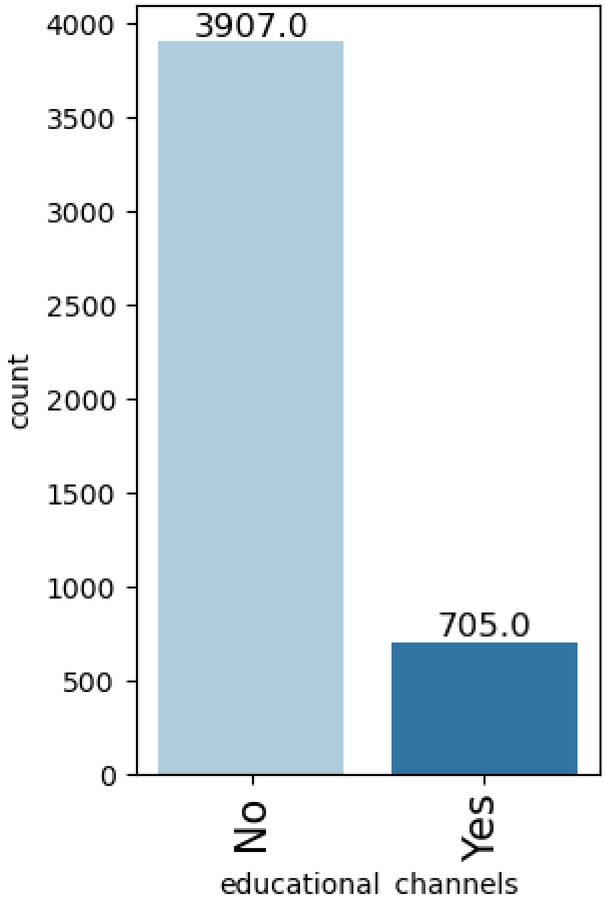
Observations on digital_media

In [26]:
```
labeled_barplot(data,"digital_media") # Complete the code to plot labeled_barpl
```



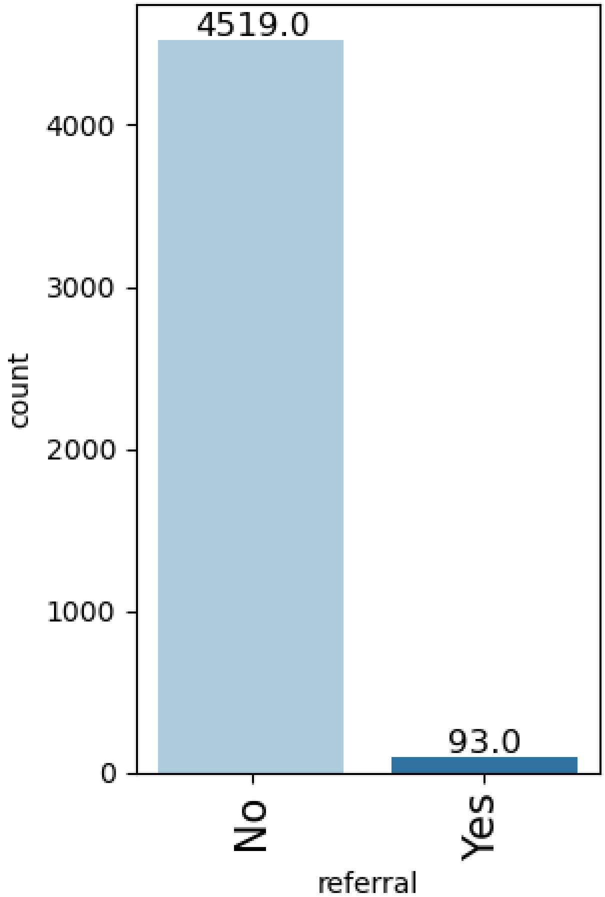
Observations on educational_channels

In [27]:
```
labeled_barplot(data,"educational_channels") # Complete the code to plot labele
```



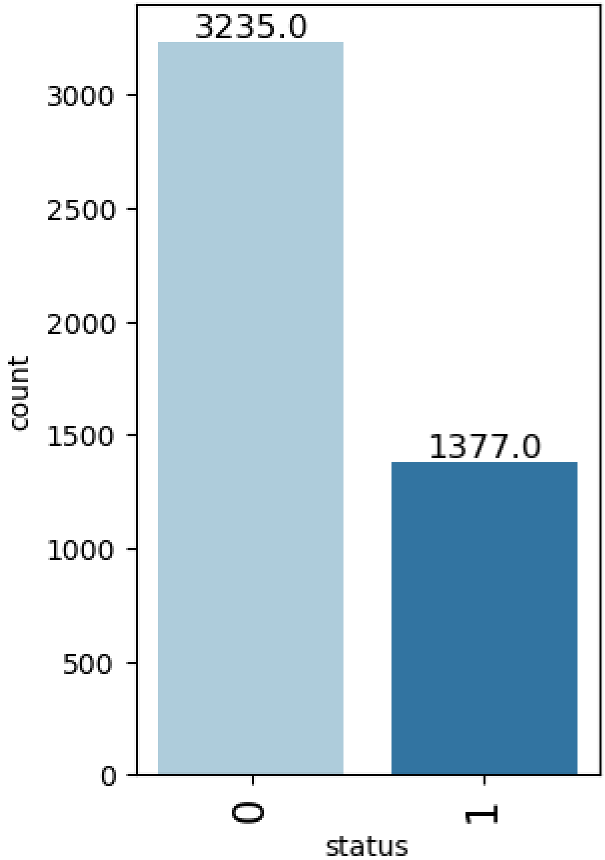
Observations on referral

In [28]:
```
labeled_barplot(data,"referral") # Complete the code to plot labeled_barplot fo
```



Observations on status

In [29]:
```
labeled_barplot(data,"status") # Complete the code to plot labeled_barplot for
```



Observations from Univariate Analysis: Current occupation is nearly twice the size of unemployed 2616 vs 1441 (56.7% vs 31.2% respectively). Most first interactions are from the website. High to medium profile completed roughly the same amount. Educational channels had the most yes's. The average age was in late 40's (approximately 46), median age was in early 50's (approximately 51).

Bivariate Analysis

In [30]:
```
cols_list = data.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(12, 7))
sns.heatmap(
    data[cols_list].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spect
)
plt.show()
```

Creating functions that will help us with further analysis.

In [31]:
```
### function to plot distributions wrt target


def distribution_plot_wrt_target(data, predictor, target):

    fig, axs = plt.subplots(2, 2, figsize=(12, 10))

    target_uniq = data[target].unique()

    axs[0, 0].set_title("Distribution of target for target=" + str(target_uniq[
    sns.histplot(
        data=data[data[target] == target_uniq[0]],
        x=predictor,
        kde=True,
        ax=axs[0, 0],
        color="teal",
        stat="density",
    )

    axs[0, 1].set_title("Distribution of target for target=" + str(target_uniq[
    sns.histplot(
        data=data[data[target] == target_uniq[1]],
        x=predictor,
        kde=True,
        ax=axs[0, 1],
        color="orange",
        stat="density",
    )

    axs[1, 0].set_title("Boxplot w.r.t target")
```

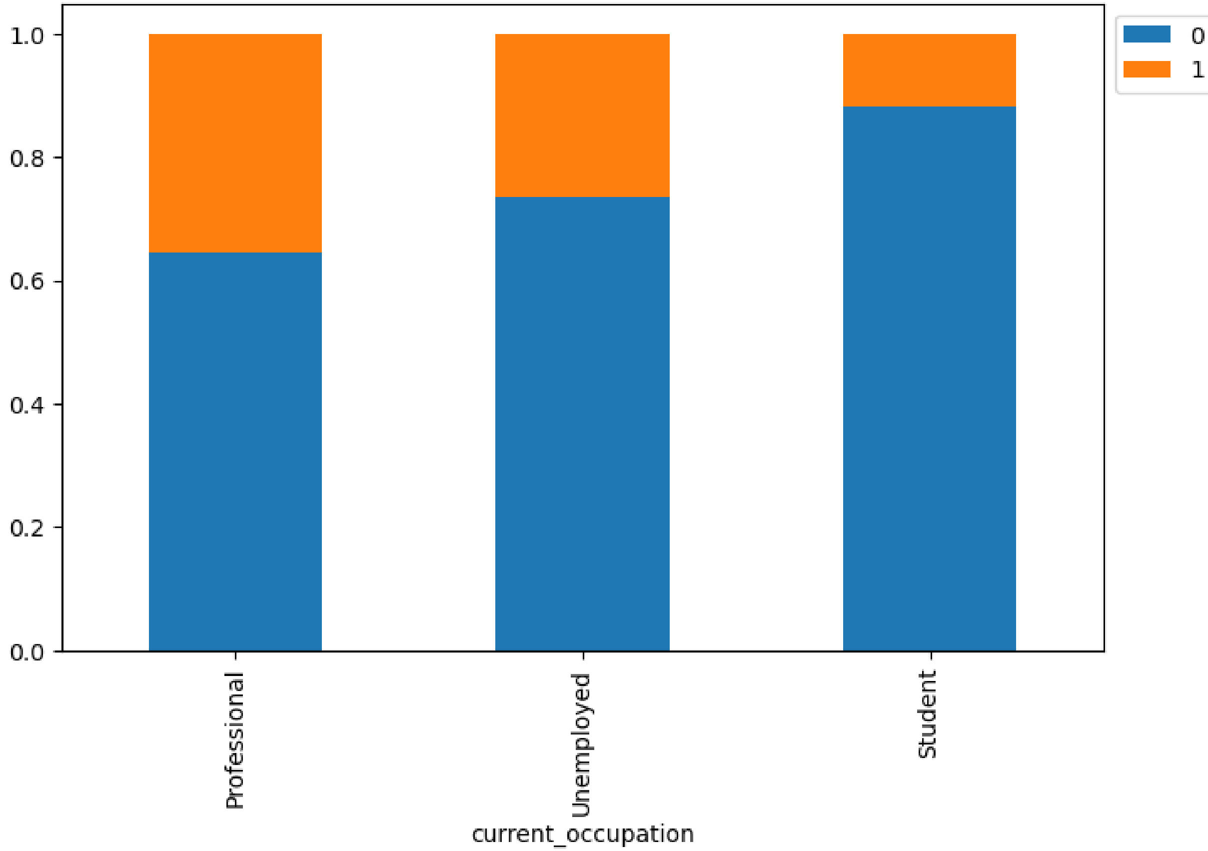
```
    sns.boxplot(data=data, x=target, y=predictor, ax=axs[1, 0], palette="gist_r

    axs[1, 1].set_title("Boxplot (without outliers) w.r.t target")
    sns.boxplot(
        data=data,
        x=target,
        y=predictor,
        ax=axs[1, 1],
        showfliers=False,
        palette="gist_rainbow",
    )

    plt.tight_layout()
    plt.show()
```

In [32]:
```
def stacked_barplot(data, predictor, target):
    """
    Print the category counts and plot a stacked bar chart

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_values
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_va
        by=sorter, ascending=False
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 5, 5))
    plt.legend(
        loc="lower left", frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

Leads will have different expectations from the outcome of the course and the current occupation may play a key role for them to take the program. Let's analyze it

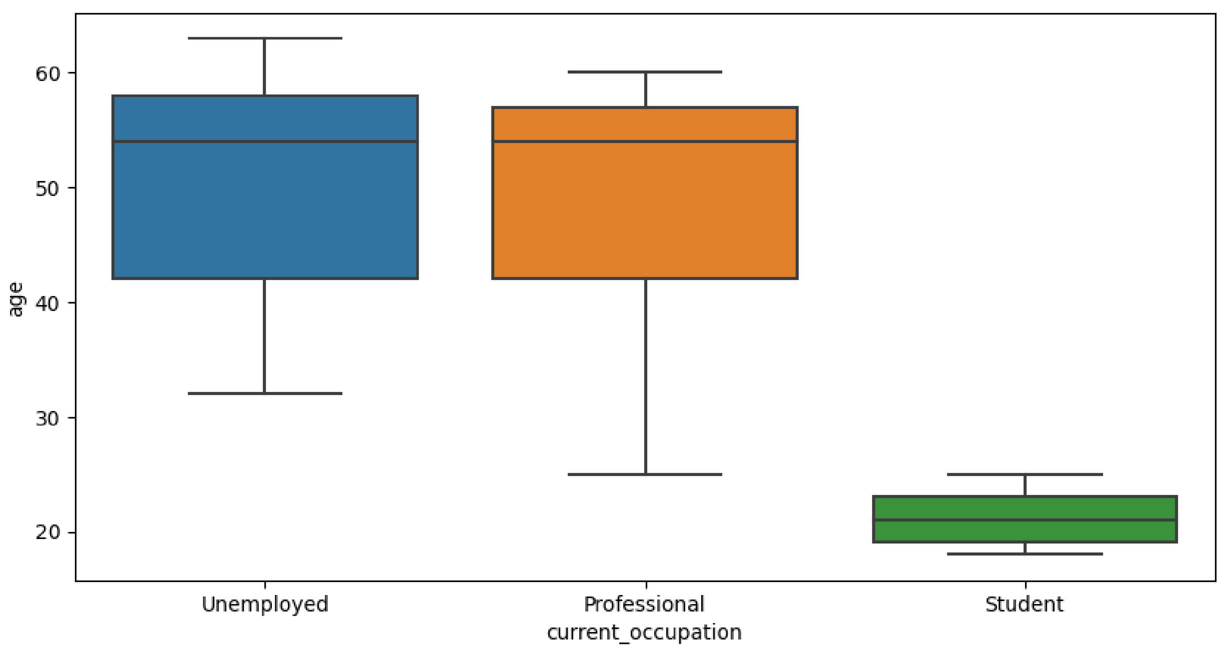
In [33]:
```
stacked_barplot(data, "current_occupation", "status")
```

```
status                 0     1   All
current_occupation
All                 3235  1377  4612
Professional        1687   929  2616
Unemployed          1058   383  1441
Student              490    65   555
------------------------------------------------------------------------------
------------------------------------------
```



Age can be a good factor to differentiate between such leads

In [34]:
```
plt.figure(figsize=(10, 5))
sns.boxplot(x=data["current_occupation"],y=data["age"])
plt.show()
```



In [35]:
```
data.groupby(["current_occupation"])["age"].describe()
```

Out[35]:

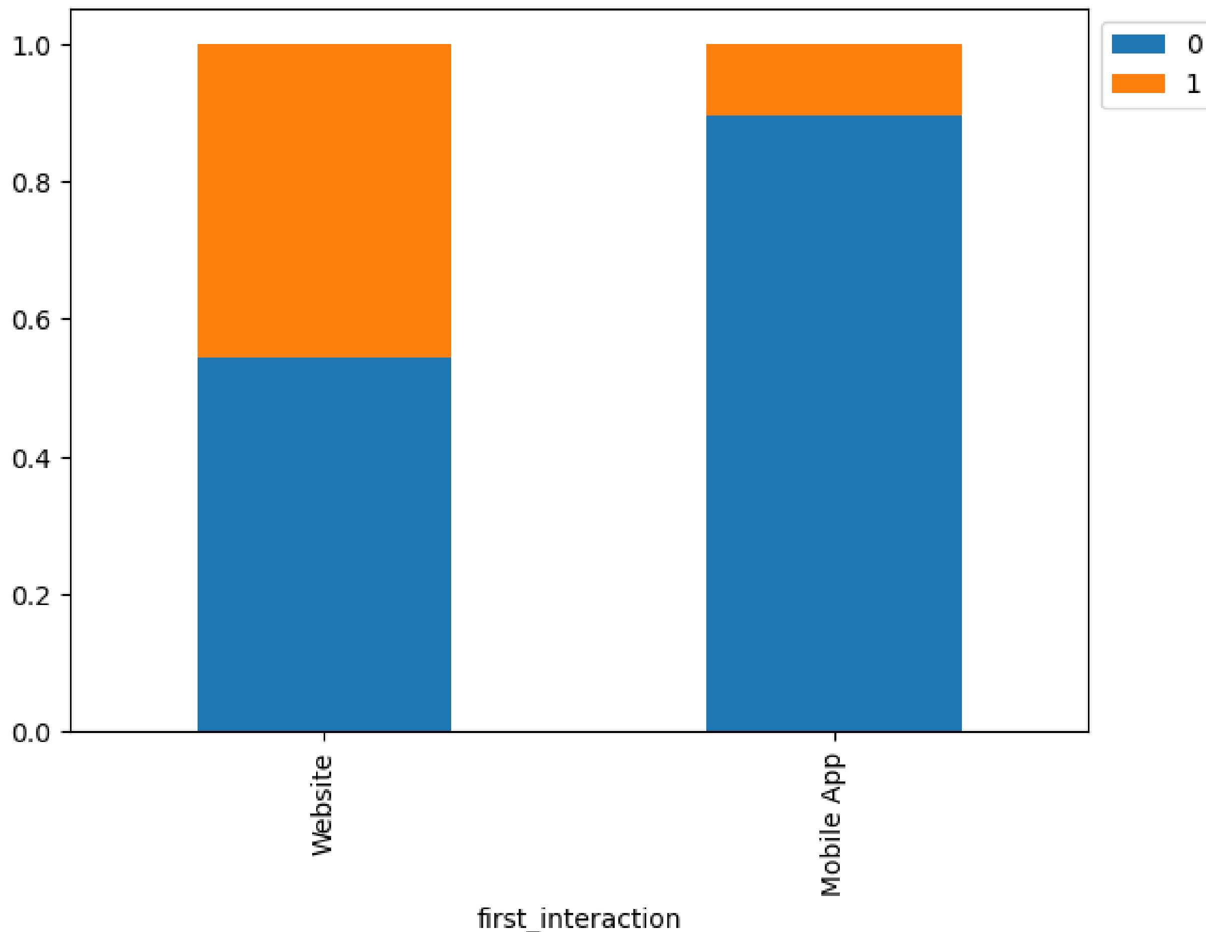
current_occupation	count	mean	std	min	25%	50%	75%
Professional	2616.00000	49.34748	9.89074	25.00000	42.00000	54.00000	57.00000
Student	555.00000	21.14414	2.00111	18.00000	19.00000	21.00000	23.00000
Unemployed	1441.00000	50.14018	9.99950	32.00000	42.00000	54.00000	58.00000

The company's first interaction with leads should be compelling and persuasive. Let's see if the channels of the first interaction have an impact on the conversion of leads

In [36]:

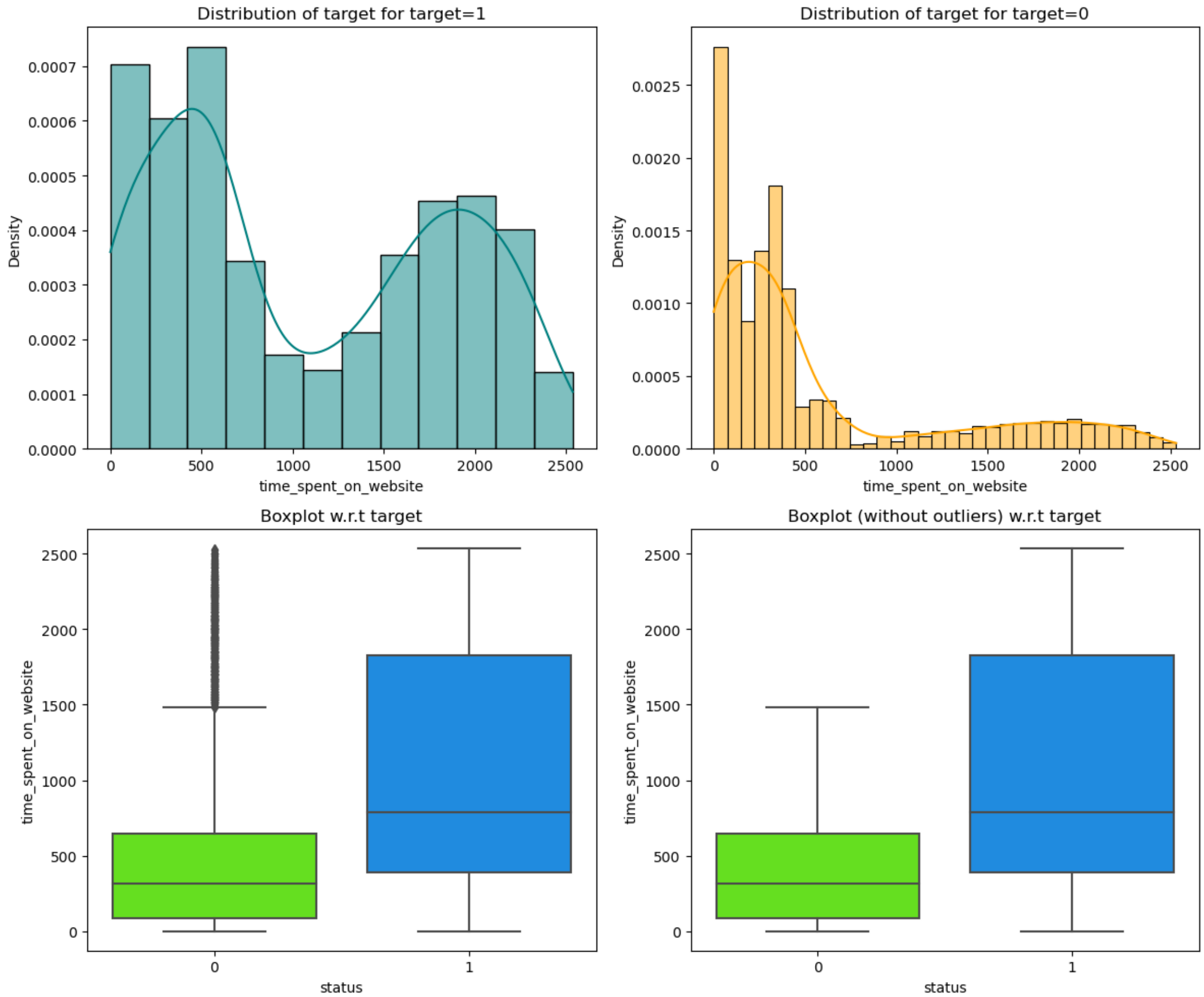
```
stacked_barplot(data,"first_interaction","status") # Complete the code to plot
```

```
status                 0     1   All
first_interaction
All                 3235  1377  4612
Website             1383  1159  2542
Mobile App          1852   218  2070
------------------------------------------------------------------------------------------------------------------------
```



In [37]:

```
distribution_plot_wrt_target(data, "time_spent_on_website", "status")
```

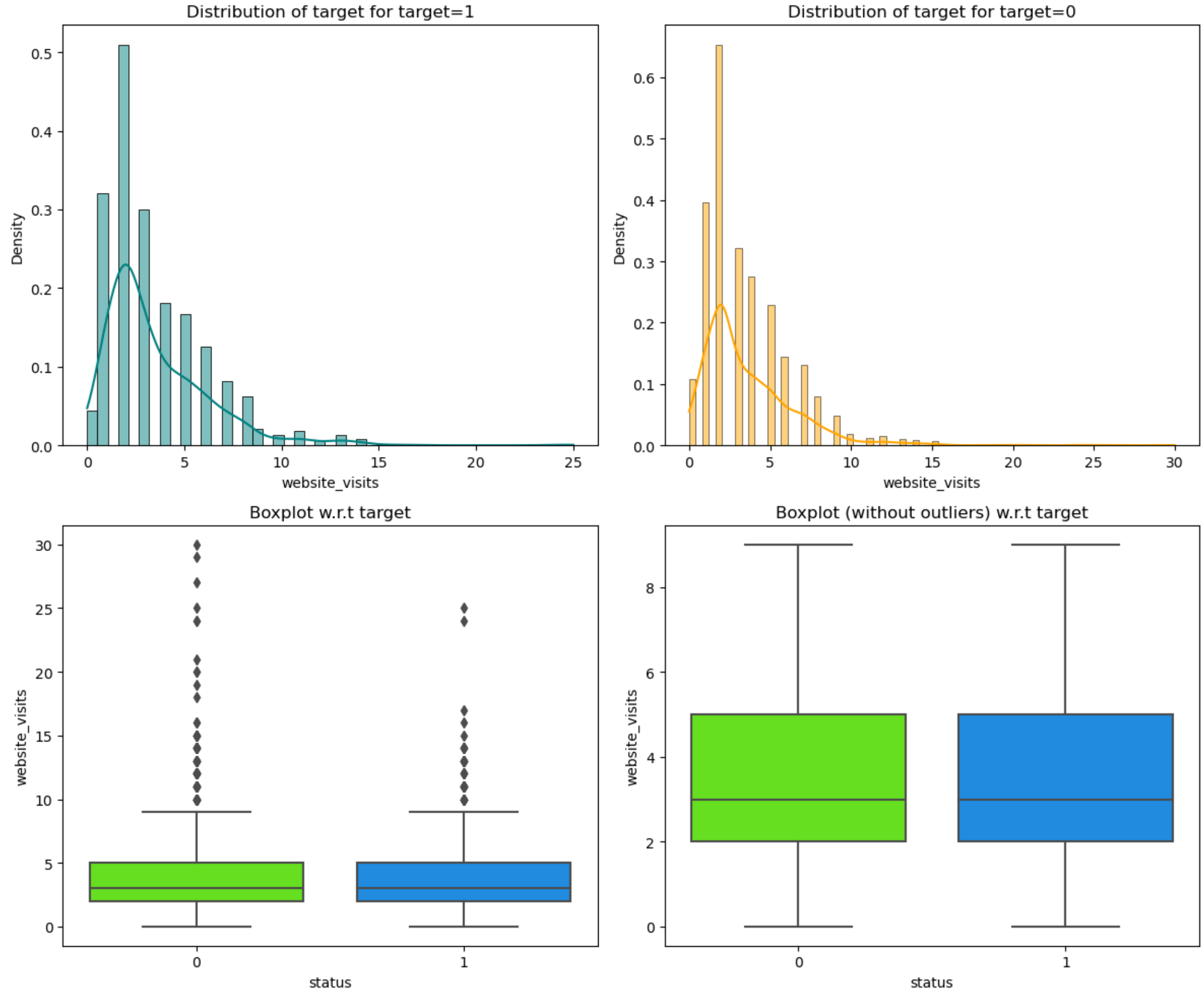


In [38]:
```
# checking the median value
data.groupby(["status"])["time_spent_on_website"].median()
```

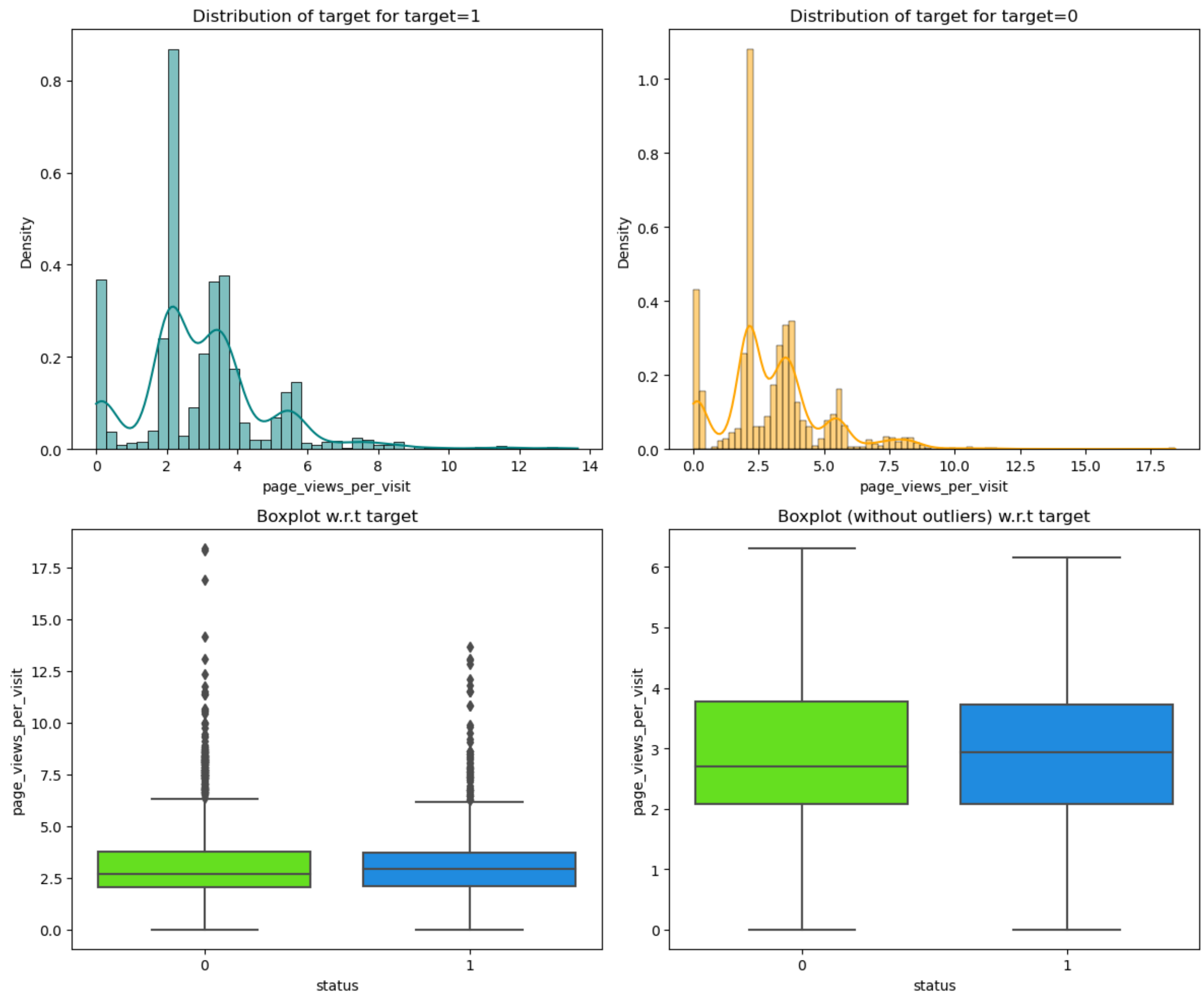
Out[38]:
```
status
0    317.00000
1    789.00000
Name: time_spent_on_website, dtype: float64
```

Let's do a similar analysis for time spent on website and page views per visit.

In [39]:
```
distribution_plot_wrt_target(data,"website_visits","status") # Complete the coc
```



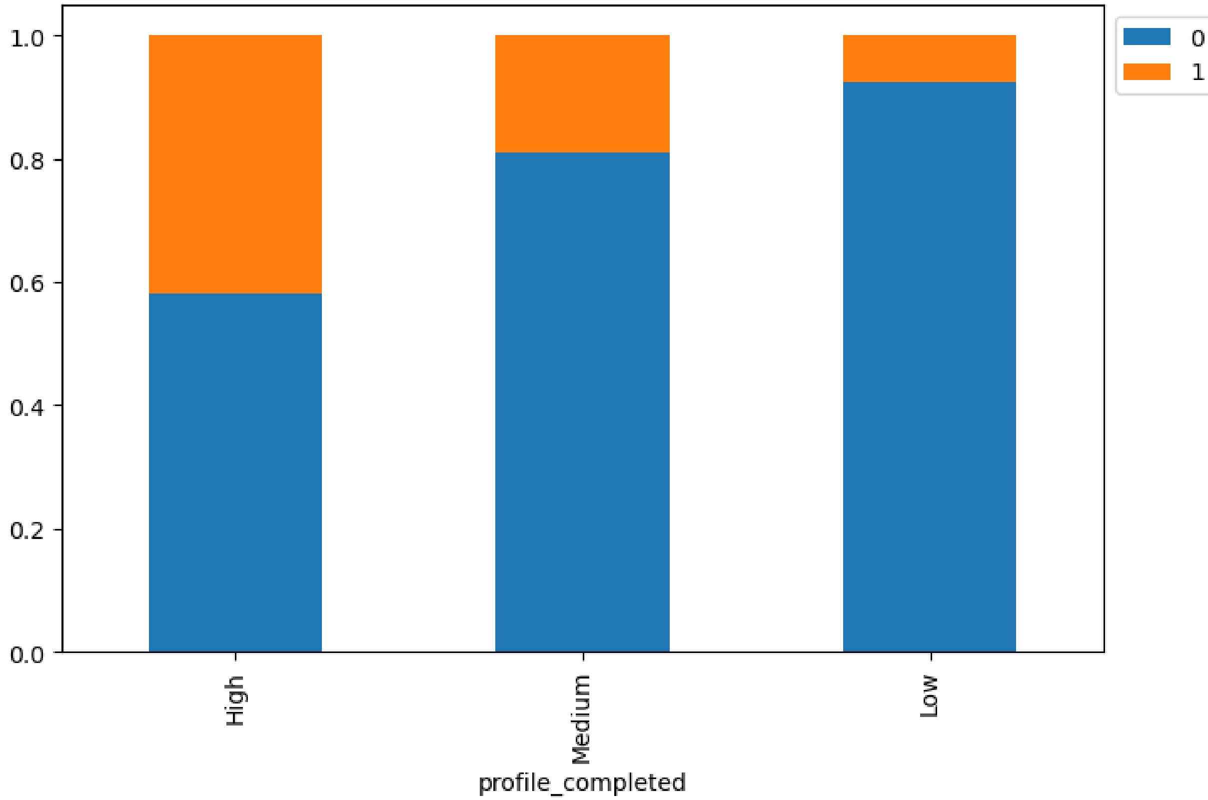
In [40]:
```python
distribution_plot_wrt_target(data,"page_views_per_visit","status") # Complete t
```



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

In [41]:
```
stacked_barplot(data,"profile_completed","status")  # Complete the code to plot
```

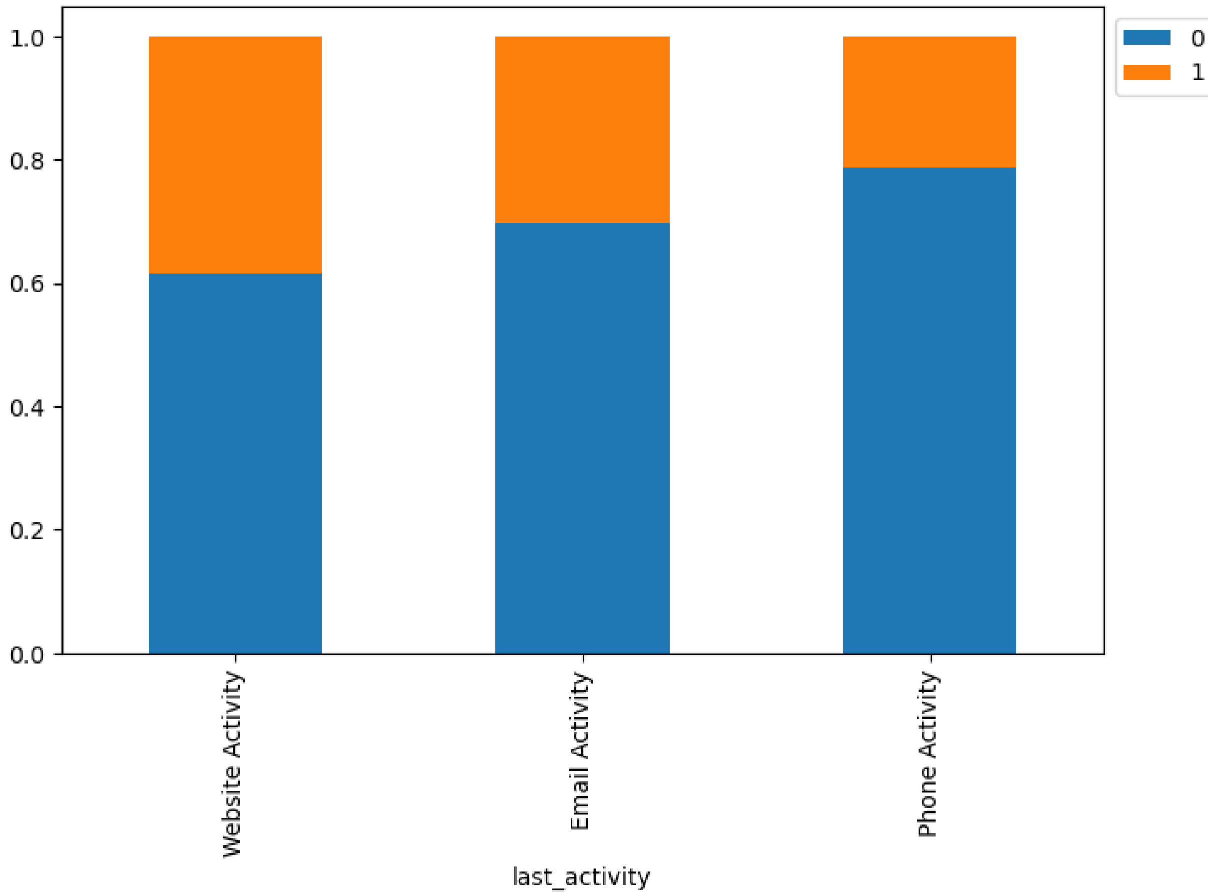
```
status                 0     1   All
profile_completed
All                 3235  1377  4612
High                1318   946  2264
Medium              1818   423  2241
Low                   99     8   107
------------------------------------------------------------------------------
------------------------------------------
```



After a lead shares their information by creating a profile, there may be interactions between the lead and the company to proceed with the process of enrollment. Let's see how the last activity impacts lead conversion status

In [42]:
```
stacked_barplot(data,"last_activity","status") # Complete the code to plot stac
```

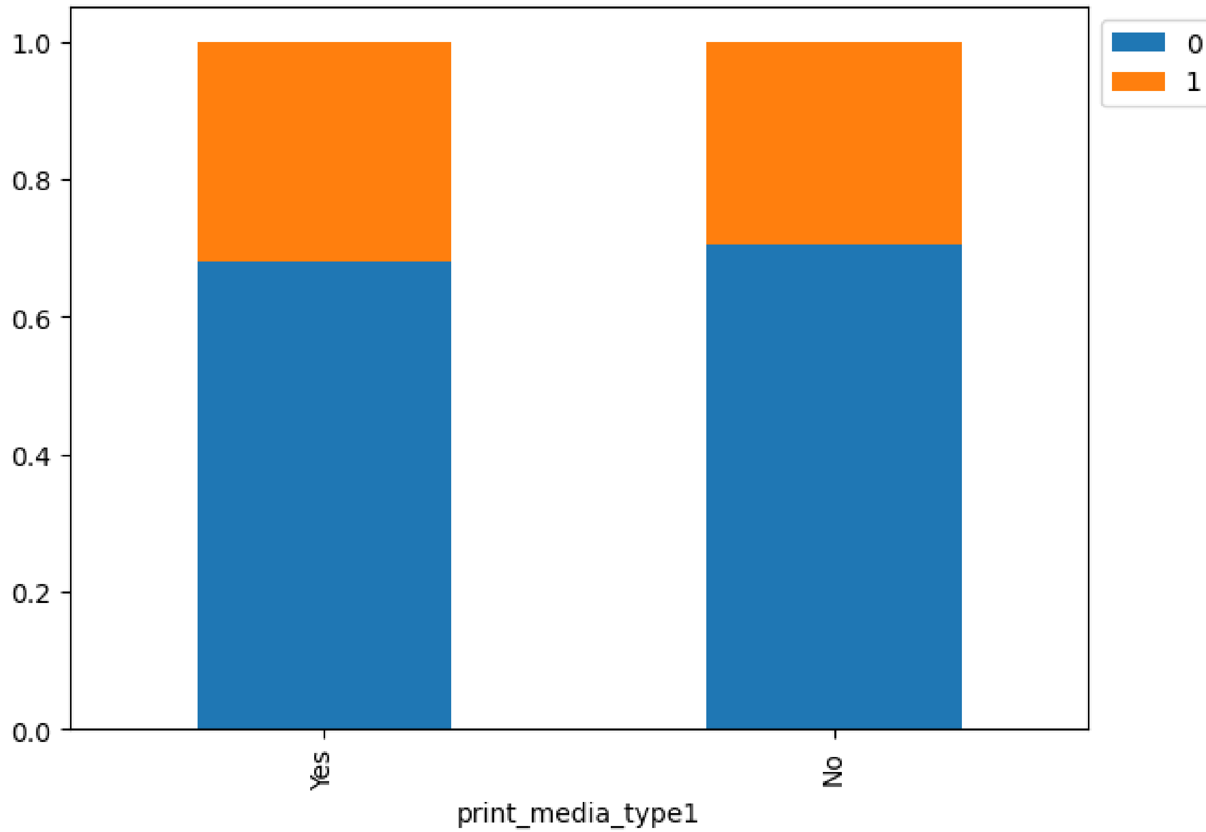
```
status                 0     1   All
last_activity
All                 3235  1377  4612
Email Activity      1587   691  2278
Website Activity     677   423  1100
Phone Activity       971   263  1234
------------------------------------------------------------------------------
------------------------------------------
```



Let's see how advertisement and referrals impact the lead status

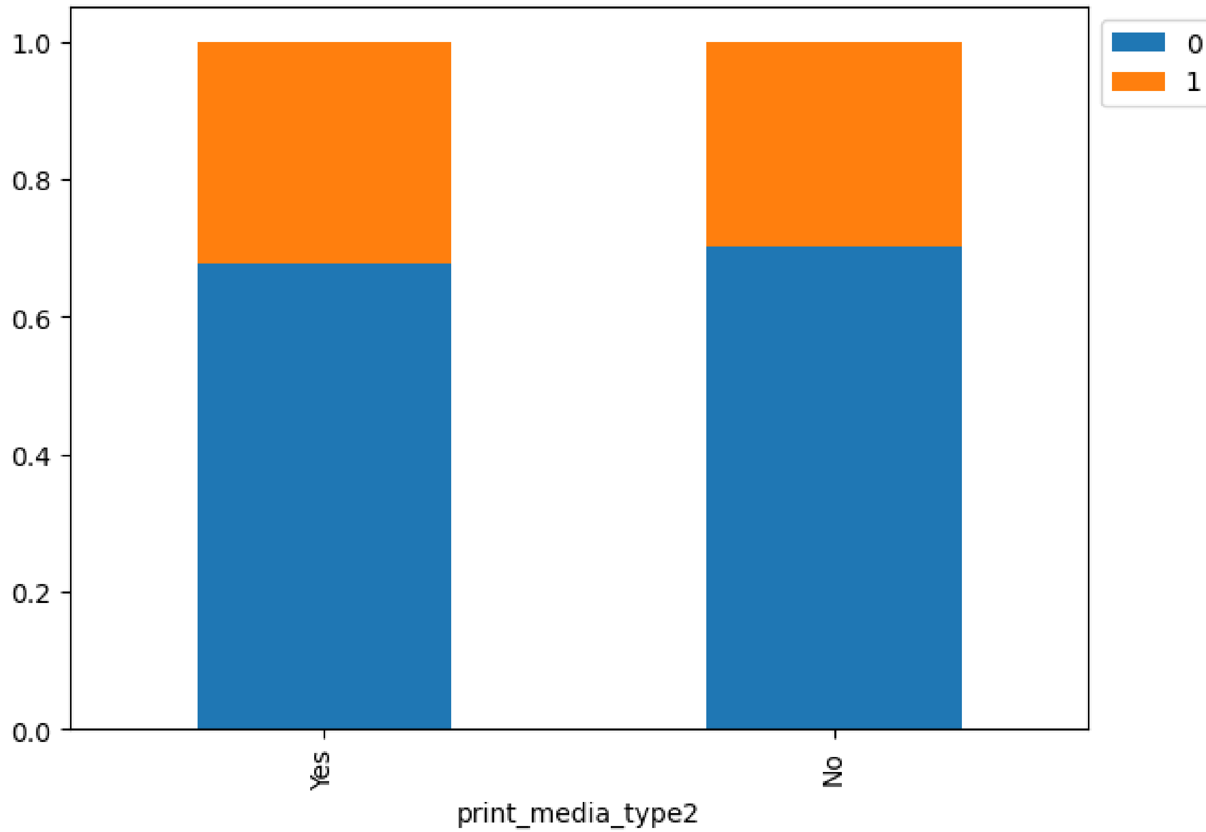
In [43]:
```
stacked_barplot(data,"print_media_type1","status") # Complete the code to plot
```

```
status                0     1   All
print_media_type1
All                3235  1377  4612
No                 2897  1218  4115
Yes                 338   159   497
------------------------------------------------------------------------------------------------------------------------
```



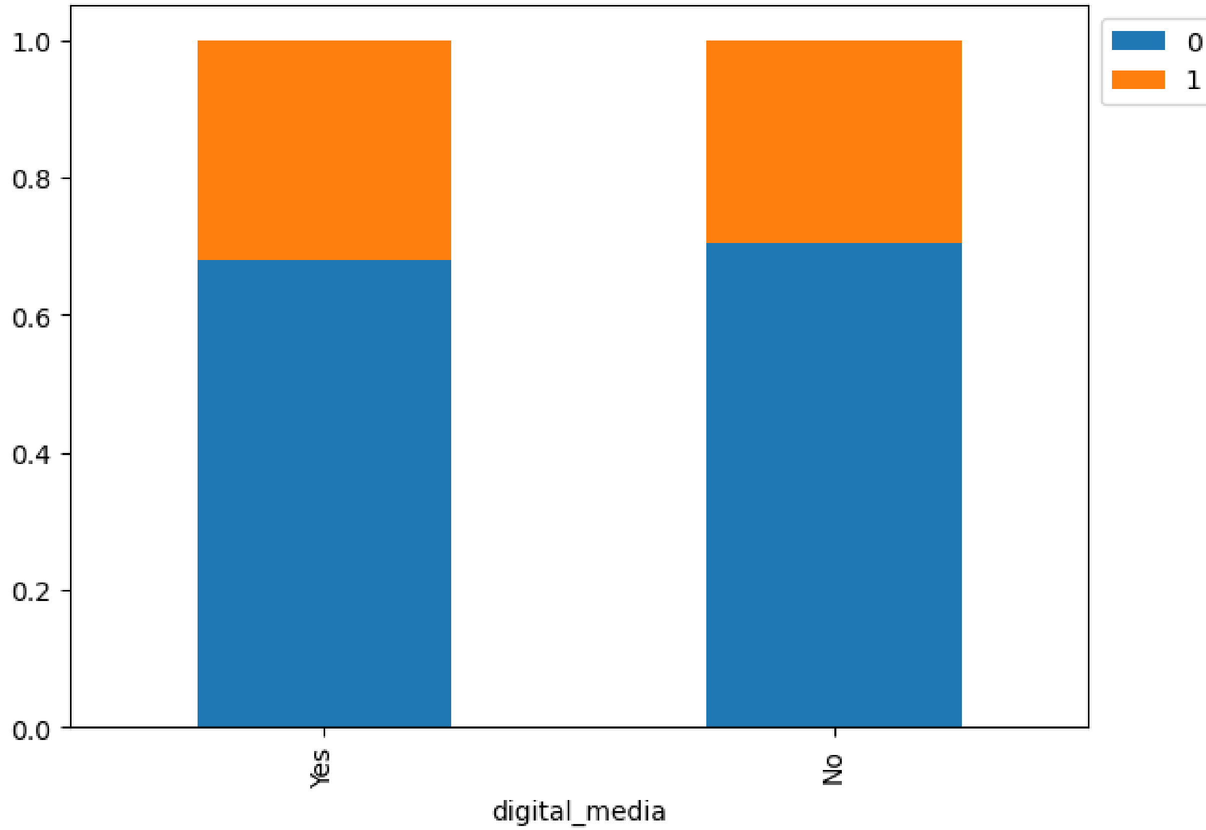
In [44]:
```
stacked_barplot(data,"print_media_type2","status") # Complete the code to plot
```

```
status                 0     1   All
print_media_type2
All                 3235  1377  4612
No                  3077  1302  4379
Yes                  158    75   233
------------------------------------------------------------------------------
------------------------------------------
```



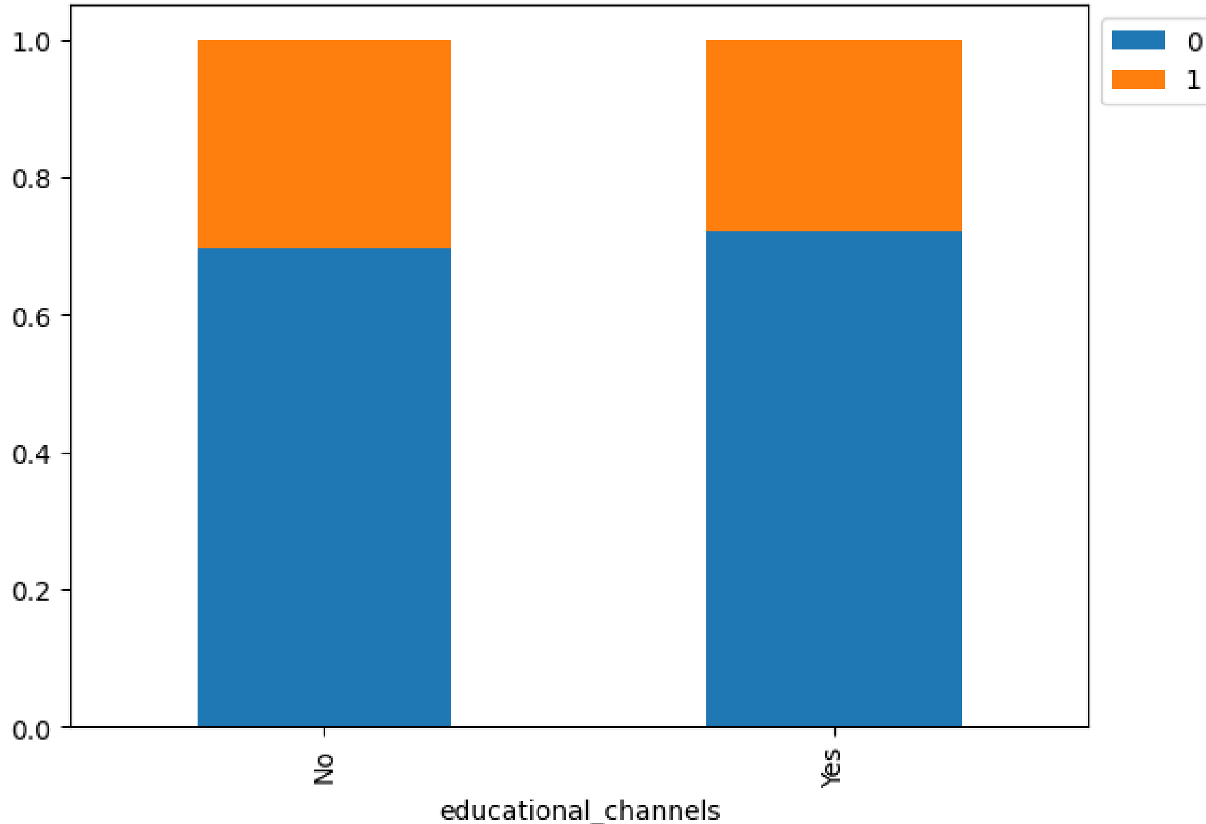
In [45]:
```
stacked_barplot(data,"digital_media","status") # Complete the code to plot stac
```

```
status             0     1   All
digital_media
All             3235  1377  4612
No              2876  1209  4085
Yes              359   168   527
------------------------------------------------------------------------------
------------------------------------------
```



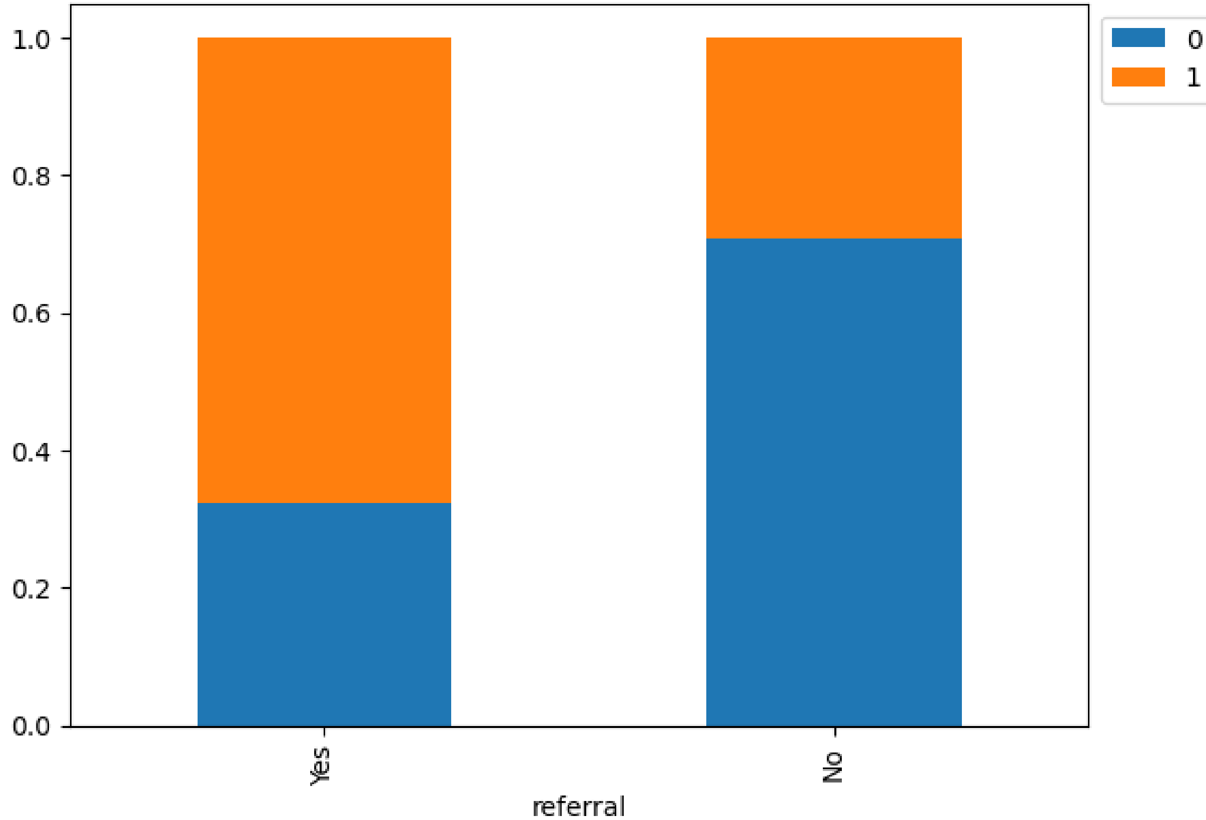
In [46]:
```
stacked_barplot(data,"educational_channels","status") # Complete the code to pl
```

```
status                   0     1   All
educational_channels
All                   3235  1377  4612
No                    2727  1180  3907
Yes                    508   197   705
------------------------------------------------------------------------------
------------------------------------------
```



In [47]:
```
stacked_barplot(data,"referral","status") # Complete the code to plot stacked_b
```

```
status       0     1   All
referral
All       3235  1377  4612
No        3205  1314  4519
Yes         30    63    93
------------------------------------------------------------------------------
------------------------------------------
```

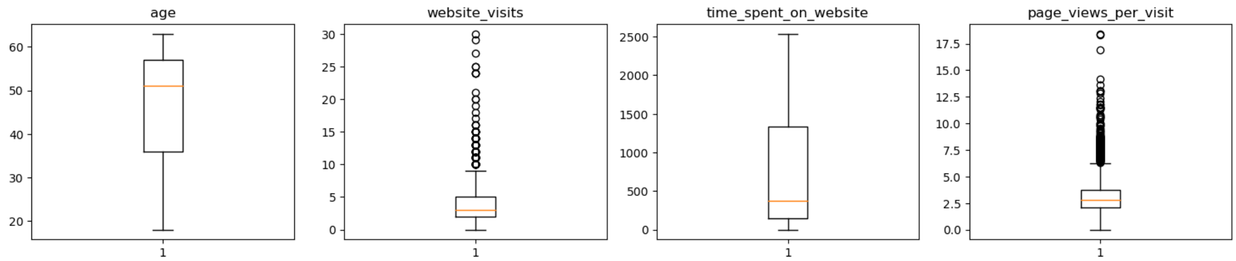


Observations from Bivariate Analysis: For current occupation, Professional had the most leads converted to a paid customer. Average age for unemployed (50.14) and professional (49.34) for current occupation was approximately the same (mid 50's). Leads that interacted with the website first had a greater number of leads converted to paid customers versus leads first interacting with the mobile app. Profiles completed that were categorized as high, had a larger number of leads converter to paid customers. Website activity had more leads converted to paid customers compared to leads with email activity and phone activity.

Outlier Check

- Let's check for outliers in the data.

In [48]:
```
# outlier detection using boxplot
numeric_columns = data.select_dtypes(include=np.number).columns.tolist()
# dropping release_year as it is a temporal variable
numeric_columns.remove("status")

plt.figure(figsize=(15, 12))

for i, variable in enumerate(numeric_columns):
    plt.subplot(4, 4, i + 1)
    plt.boxplot(data[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)

plt.show()
```



Observations: Majority of leads visited less than 5 websites, spent an average of 400 units of time on the website, and viewed on average approximately 3 pages per visit. Website visits and page views per visit had a large number of outliers.

Data Preparation for modeling

- We want to predict which lead is more likely to be converted.
- Before we proceed to build a model, we'll have to encode categorical features.
- We'll split the data into train and test to be able to evaluate the model that we build on the train data.

In [49]:
```
X = data.drop(["status"], axis=1)
Y = data.status

X = pd.get_dummies(X, drop_first=True) # Complete the code to get dummies for X

# Splitting the data in 70:30 ratio for train to test data
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.30, random_state=1
)
```

In [50]:
```
print("Shape of Training set : ", X_train.shape)
print("Shape of test set : ", X_test.shape)
print("Percentage of classes in training set:")
print(y_train.value_counts(normalize=True))
print("Percentage of classes in test set:")
print(y_test.value_counts(normalize=True))
```

```
Shape of Training set :  (3228, 16)
Shape of test set :  (1384, 16)
Percentage of classes in training set:
0    0.70415
1    0.29585
Name: status, dtype: float64
Percentage of classes in test set:
0    0.69509
1    0.30491
Name: status, dtype: float64
```

Building Classification Models

Model evaluation criterion

Model can make wrong predictions as:

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

How to reduce the losses?

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

First, let's create functions to calculate different metrics and confusion matrix so that we don't have to use the same code repeatedly for each model.

- The model_performance_classification_statsmodels function will be used to check the model performance of models.
- The confusion_matrix_statsmodels function will be used to plot the confusion matrix.

In [51]:
```
# Function to print the classification report and get confusion matrix in a pro

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',

    plt.ylabel('Actual')

    plt.xlabel('Predicted')

    plt.show()
```

Decision Tree

Building Decision Tree Model

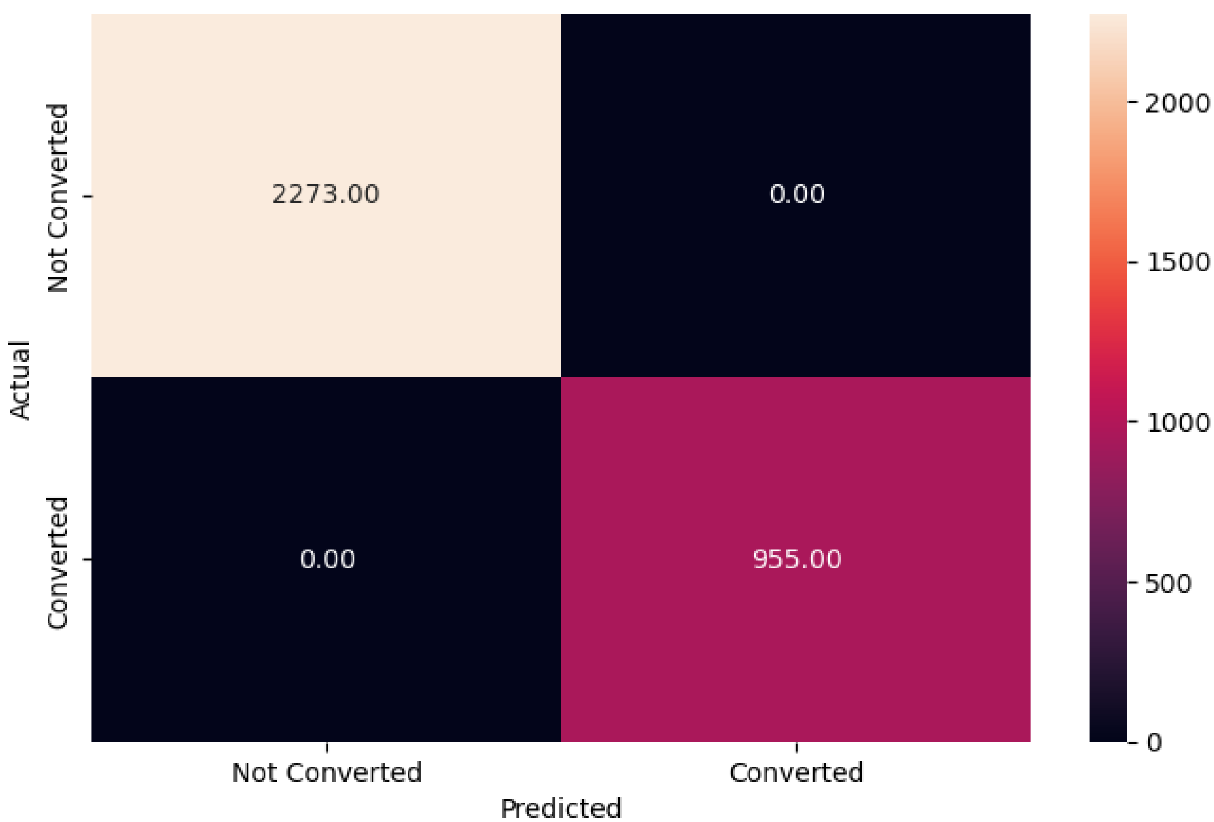
In [52]:
```
# Fitting the decision tree classifier on the training data
d_tree =  DecisionTreeClassifier(random_state = 1)

d_tree.fit(X_train, y_train)
```

Out[52]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(random_state=1)
```

Checking model performance on training set

In [53]:
```
# Checking performance on the training data
y_pred_train1 = d_tree.predict(X_train)

metrics_score(y_train, y_pred_train1)

#______________**Observations:_________**
```

```
              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
```

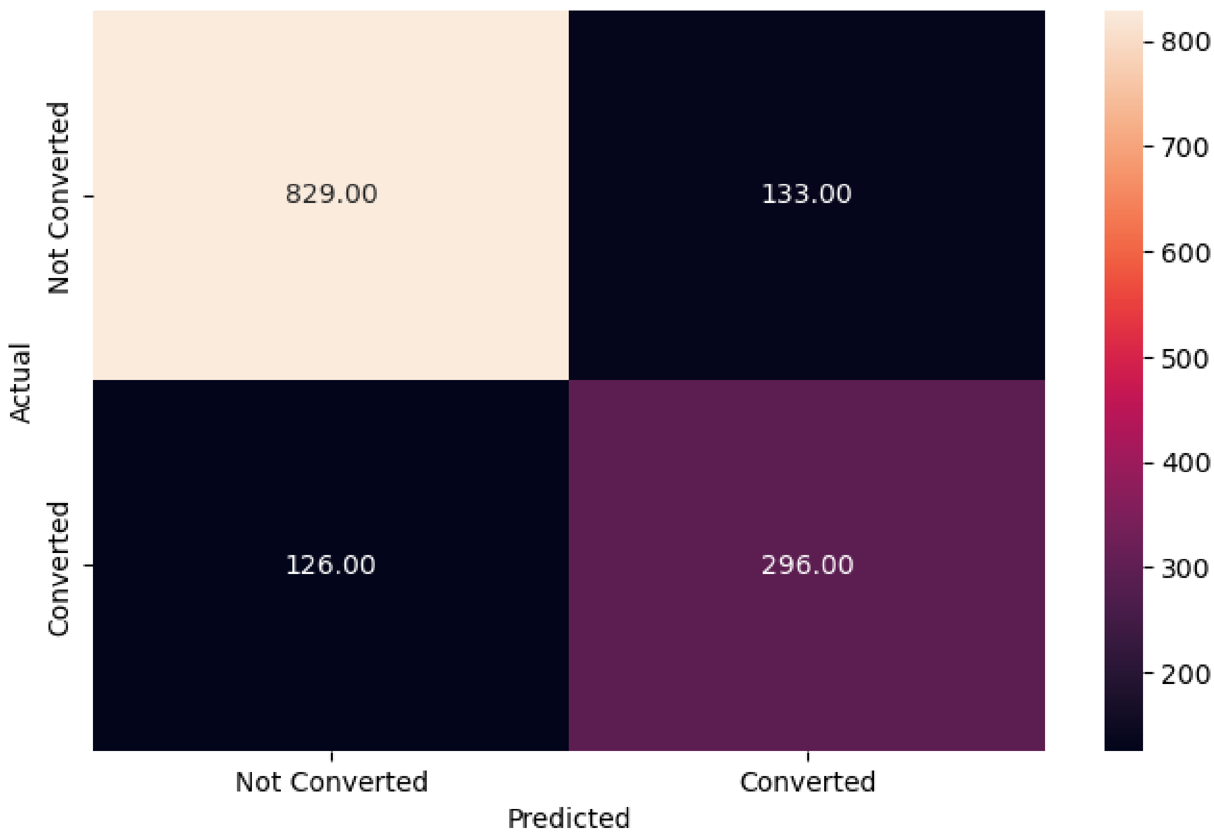


Observations: Predicted Not Converted and was actual converted was 0. Predict Converted and was actual Not Converted was 0. Percision was 1.00 and accuracy was 1.00 f1-score was 1.00.

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

In [54]:
```
# 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       0.87      0.86      0.86       962
           1       0.69      0.70      0.70       422

    accuracy                           0.81      1384
   macro avg       0.78      0.78      0.78      1384
weighted avg       0.81      0.81      0.81      1384
```



Observations: Percision, recall, and f1-score have been reduced for 0, 0.87, 0.86, and 0.86 respectively. And percision, recall and f1-score for 1 is 0.69, 0.70, and 0.70 respectively.

Let's try hyperparameter tuning using GridSearchCV to find the optimal max_depth to reduce overfitting of the model. We can tune some other hyperparameters as well.

Decision Tree - Hyperparameter Tuning

We will 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 [55]:
```
# Choose the type of classifier
d_tree_tuned = DecisionTreeClassifier(random_state = 7, class_weight = {0: 0.3,

# 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 cla
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)
```

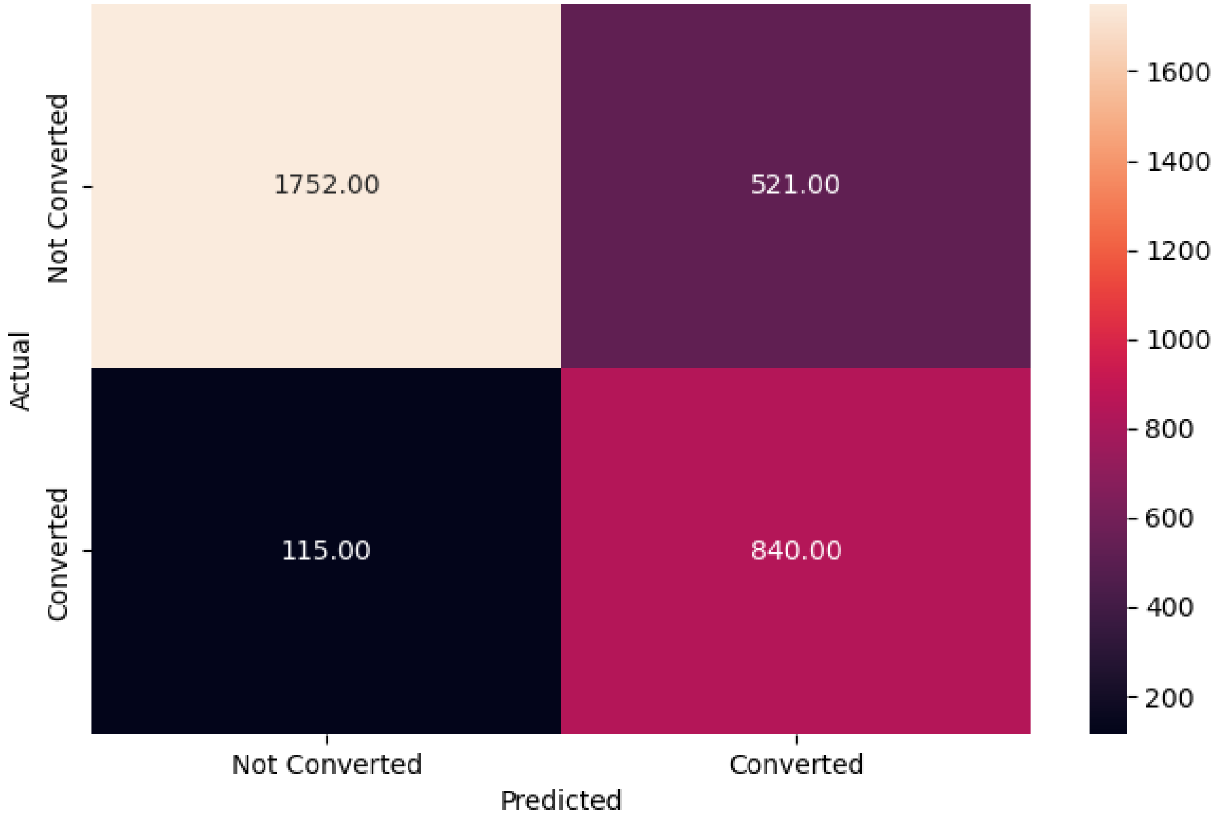
Out[55]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(class_weight={0: 0.3, 1: 0.7}, criterion='entro
py',
                       max_depth=3, min_samples_leaf=5, random_state=
7)
```

We have tuned the model and fit the tuned model on the training data. Now, **let's check the model performance on the training and testing data.**

Checking model performance on train and test set

In [56]:
```
# 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
```

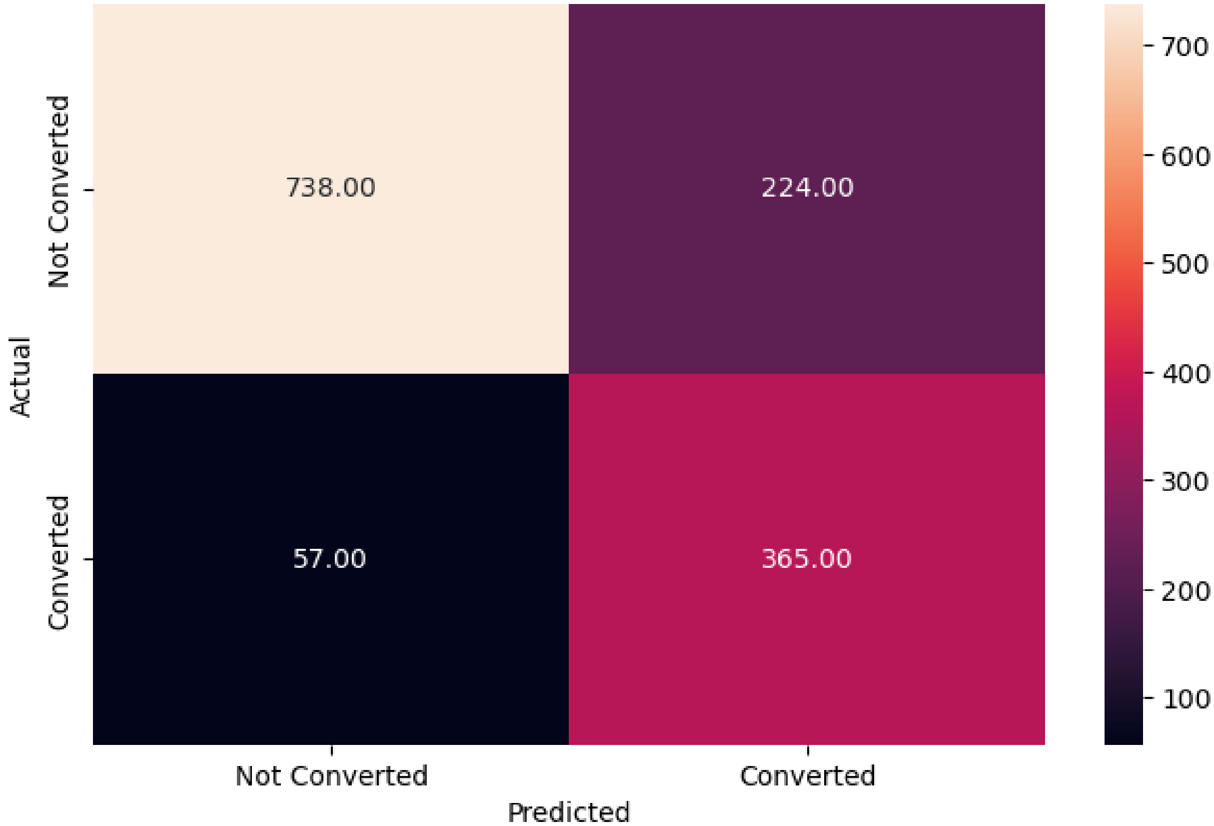


Observations: We are reducing overfitting. Pericision for 0 is 0.94 and for 1 is 0.62. Accuarcy is 0.80.

Let's check the model performance on the testing data

In [57]:
```
# 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
```

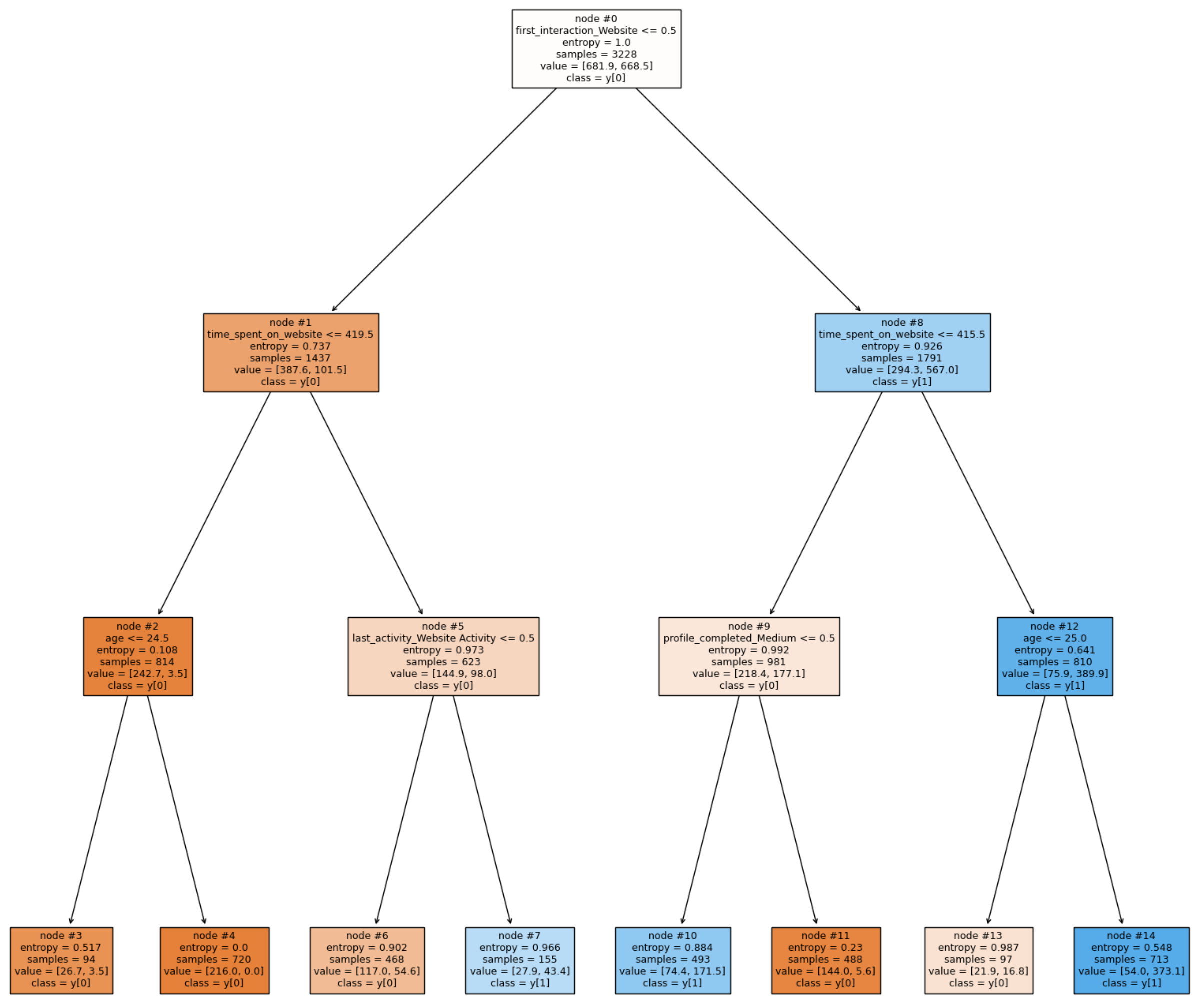


Observations: In this model, our percision has slightly decreased and is now performing more generalized on the training data and the testing data.

Visualizing the Decision Tree

Let's visualize the tuned decision tree and observe the decision rules:

In [58]:
```
features = list(X.columns)

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

tree.plot_tree(d_tree_tuned, feature_names = features, filled = True, fontsize

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 first split in the decision tree is at first_interaction_website, implying it the the most important facotr on whether or not a lead is converted into a customer. If the lead spends less than 415.5 units of time on the webiste, they are more likely to be converted into a customer. If the lead is less than or equal to age 25, then they are more likely to represent converted leads. If the leads' last acitivity website activity is less than 0.5 they are more likely not to convert. If a lead has a profile completed categorized as medium, they are more likely not to convert.

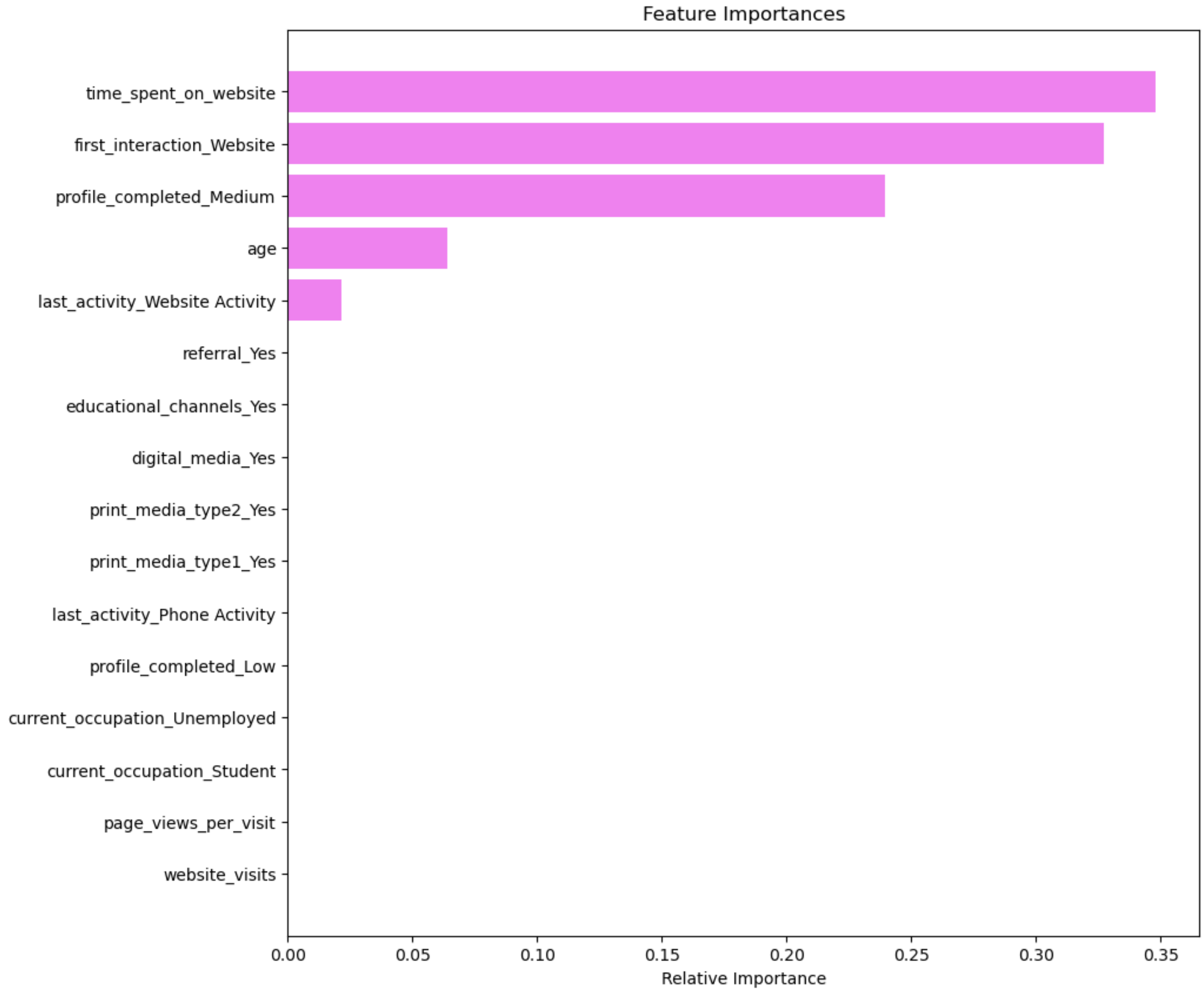
Let's look at the feature importance of the tuned decision tree model

In [59]:
```
# Importance of features in the tree building

print (pd.DataFrame(d_tree_tuned.feature_importances_, columns = ["Imp"], index
```

```
                                      Imp
time_spent_on_website           0.34814
first_interaction_Website       0.32718
profile_completed_Medium        0.23927
age                             0.06389
last_activity_Website Activity  0.02151
website_visits                  0.00000
page_views_per_visit            0.00000
current_occupation_Student      0.00000
current_occupation_Unemployed   0.00000
profile_completed_Low           0.00000
last_activity_Phone Activity    0.00000
print_media_type1_Yes           0.00000
print_media_type2_Yes           0.00000
digital_media_Yes               0.00000
educational_channels_Yes        0.00000
referral_Yes                    0.00000
```

In [60]:

```
# 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 = '

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**.

Now, let's build another model - **a random forest classifier.**

Random Forest Classifier

Building Random Forest Model

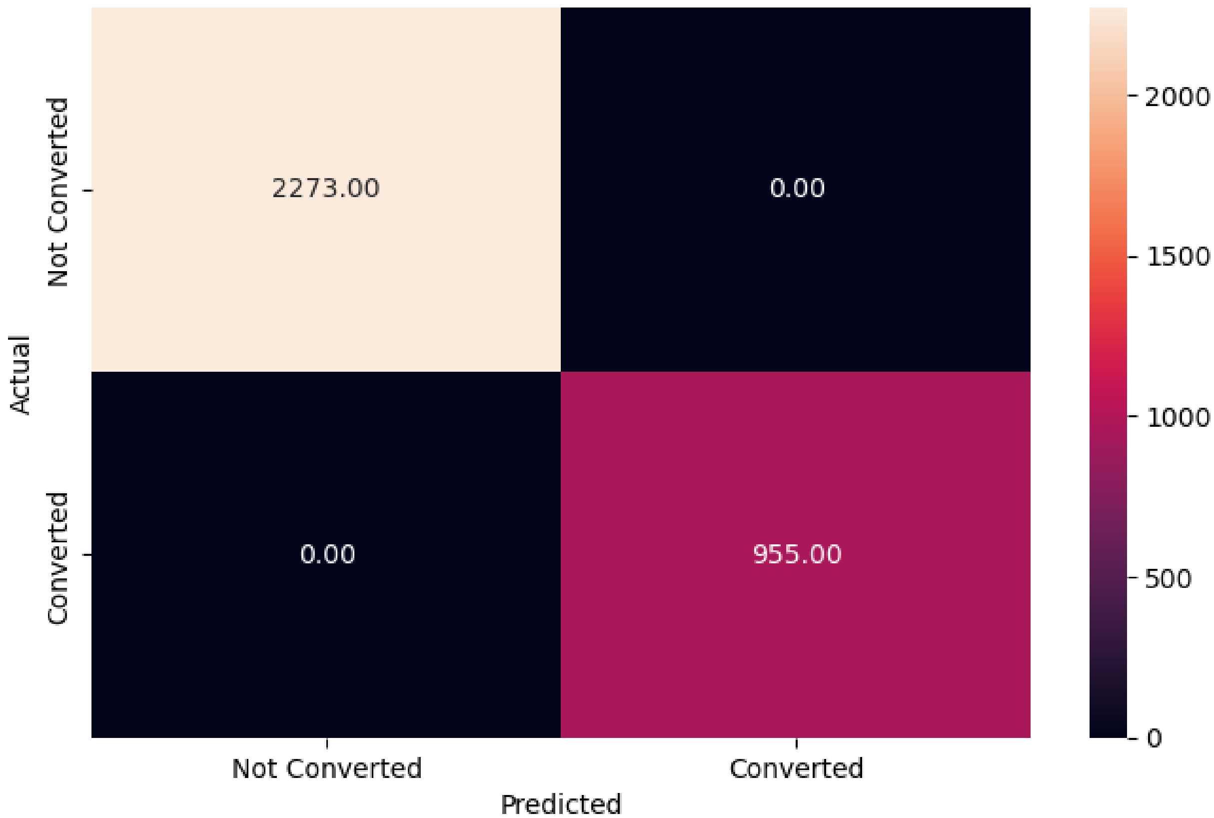
In [67]:
```
# Fitting the random forest tree classifier on the training data
rf_estimator = RandomForestClassifier(random_state = 7, criterion = "entropy")

rf_estimator.fit(X_train,y_train)
```

Out[67]:
```
▾ RandomForestClassifier
RandomForestClassifier(criterion='entropy', random_state=7)
```

Let's check the performance of the model on the training data

In [68]:
```
# 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
```

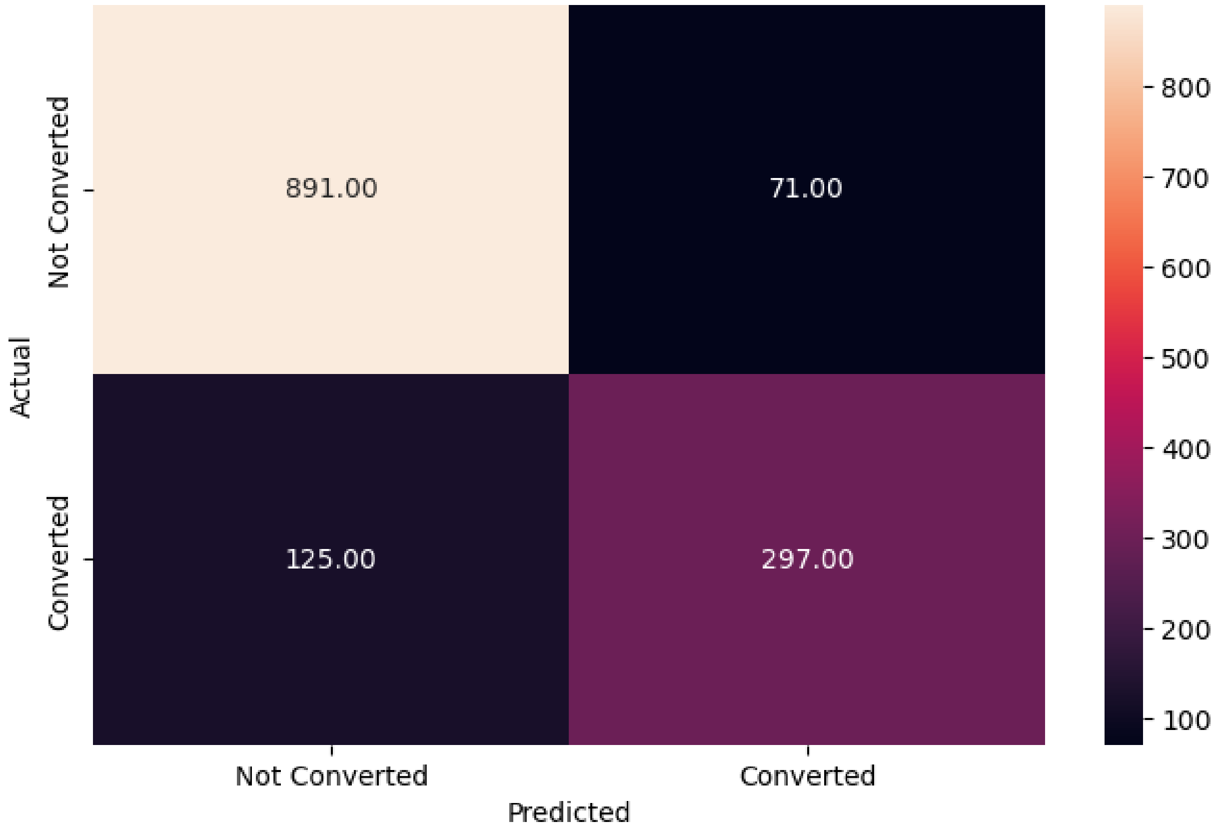


Observations: Clearly looking at the values percision and accuracy as 1.00 this model is overfitted. We can also view this in the graph showing predicted not converted to actual converted as 0.00 and predicted converted to actual not converted as 0.00.

Let's check the performance on the testing data

In [69]:
```
# 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: On the tesitng data we get percision for 0 as 0.88 and for 1 as 0.81. F1-score is .90 for 0 and 0.75 for 1. Accuracy is 0.86.

Let's see if we can get a better model by tuning the random forest classifier

Random Forest Classifier - Hyperparameter Tuning

Let's try **tuning some of the important hyperparameters of the Random Forest Classifier**.

We will **not** tune the `criterion` hyperparameter as we know from hyperparameter tuning for decision trees that `entropy` is a better splitting criterion for this data.

In [70]:
```
# Choose the type of classifier
rf_estimator_tuned = RandomForestClassifier(criterion = "entropy", random_state

# 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": ["balanced",{0: 0.3, 1: 0.7}]
             }

# Type of scoring used to compare parameter combinations - recall score for cla
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 =

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)
```

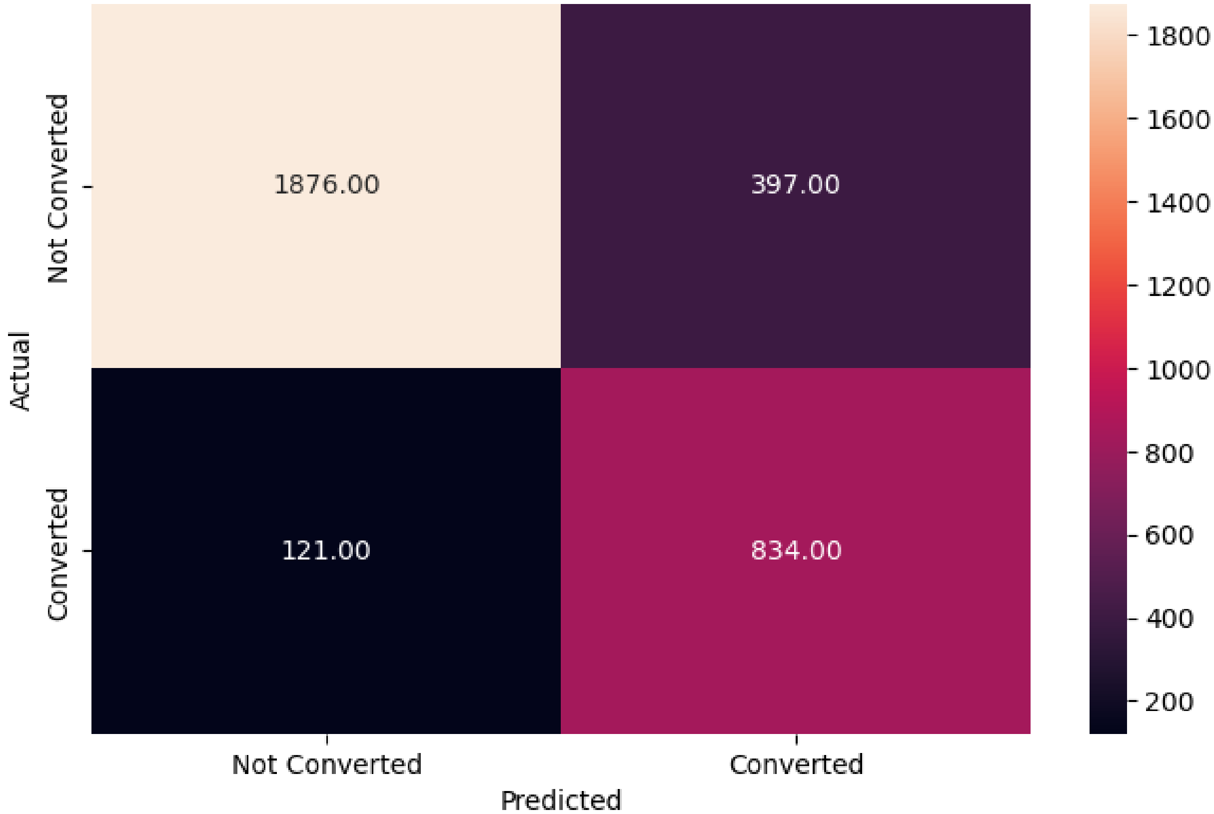
Out[70]:

```
▾ RandomForestClassifier
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_s
tate=7)
```

In [71]:

```
# 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       0.94      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
```

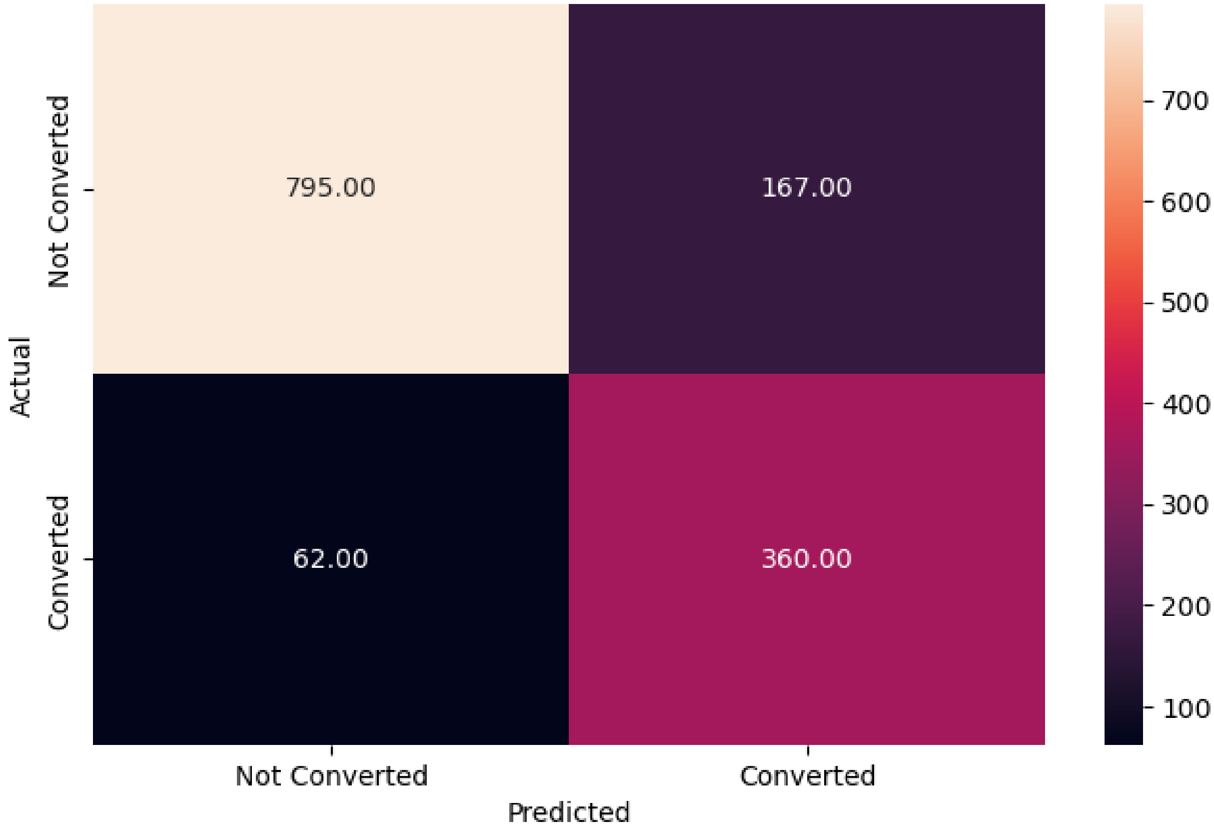


Observations: Percision has increased. Accuracy has slighly decreased.

Let's check the model performance on the test data

In [72]:
```
# Checking performance on the test data
y_pred_test4 = rf_estimator_tuned.predict(X_test)

metrics_score(y_test, y_pred_test4)
```

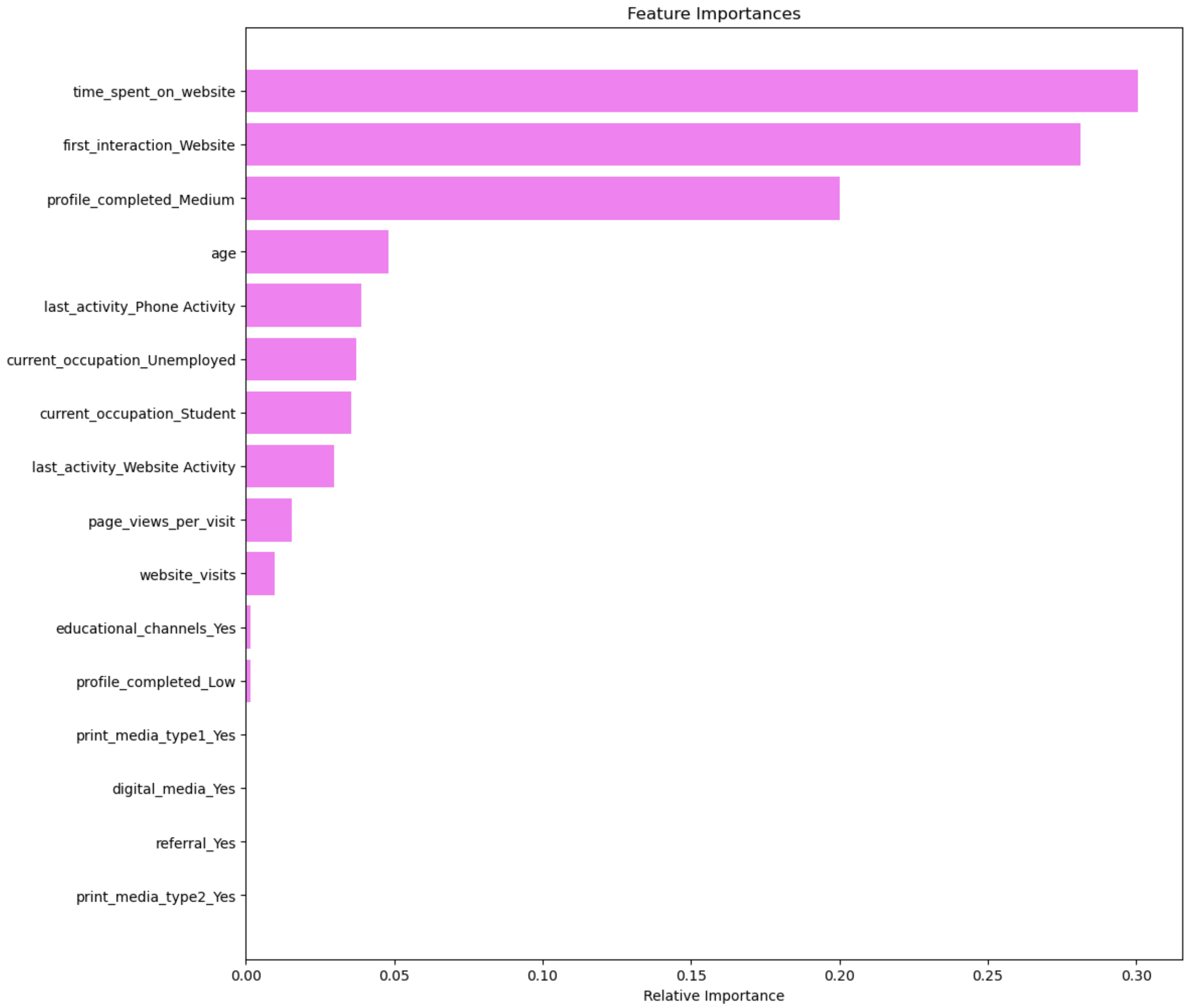
```
              precision    recall  f1-score   support

           0       0.93      0.83      0.87       962
           1       0.68      0.85      0.76       422

    accuracy                           0.83      1384
   macro avg       0.81      0.84      0.82      1384
weighted avg       0.85      0.83      0.84      1384
```



Observations: Percision and accuracy has slightly decreased. F1-score is the same.

One of the drawbacks of ensemble models is that we 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, let's just check the feature importance of the model.

In [73]:
```
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 = '

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

plt.xlabel('Relative Importance')

plt.show()
```



Observations: In comparison to our decision tree model, time_spent_on_website, first_interaction_Website, profile_completed_Medium, age, and last_activity_Phone Activity are the top five features that determine whether or not a lead is converted into a customer. Interestingly enough, our random forest showed the importance of other features like: current_occupation_Unemployed, current_occupation_Student, last_activity_Website Activity, page_views_per_visit, website_visits, educational_channels_Yes, and profile_completed_Low.

Conclusion and Recommendations

Conclusions:

We have built tree-based models that can predict which leads are likely to be converted in to a customer. We know the important features to target for a higher conversion rate. Our random forest model has the highest f1-score of 87% on the test data and a macro avg of 82%. The most important features are time_spent_on_website, first_interaction_Website, profile_completed_Medium, age, and last_activity_Phone Activity. ExtraaLearn will be able to focus company resources towards features that have a higher likelihood of converting a lead to a customer.

Business Recommendations

1. The time spent on website plays a key factor on whether or not a lead converts to a customer. ExtraaLearn can add more information, either with infographics or interactive slides to increase the time a lead stays on their site.
2. Another important feature is if the lead contacts ExtraaLearn's website first. ExtraaLearn can devote more resources to online marketing to get more web based interactions. This is an important feature to convert a lead into a customer.
3. If leads completed the profile and were categorized as medium, it lead to a higher likelihood to convert to a customer. ExtraaLearn could review which portions of leads' profiles are completed versus left incomplete and then compare those portions to customer profiles. They can target the missing gaps or redesign the profile form to reduce incompleteness.
4. Age was another feature that played a key role in conversion to customer. Although the average age of leads was in the 50's, in our Decision Tree we clearly see if the age of the lead was less than 25, it had a higher likelihood of converting to a customer. In combination with data collected about current_occupation, ExtraaLearn can focus more on students as they tend to be in the age demographic that have a higher conversion percentage to customer.
5. The previous recommendation is a good segway into the last feature of importance, last_activity_Phone Activity. The last conversation ExtraaLearn had with a lead via phone conversatoin had a higher likelihood of turning a lead into a customer. ExtraaLearn can hire more phone representatives to follow up with leads thus leading to a higher conversion to customers.