ExtraaLearn Project

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 on a regular basis, one of the issues faced by ExtraaLearn is to identify which of the leads are more likely to convert so that they can allocate resources accordingly. You, as a data scientist at ExtraaLearn, have been provided the leads data to:

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

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 interacted with ExtraaLearn. Values include 'Website', 'Mobile App'
- profile_completed: What percentage of 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 program through email, Representative shared information with lead like brochure of program , etc
    - Phone Activity: Had a Phone Conversation with representative, Had conversation over SMS with representative, etc
    - Website Activity: Interacted on live chat with representative, Updated profile on 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.

In [ ]:
```
# Analyze and build an ML model to help identify which leads are more likely to con
# Find the factors driving the lead conversion process
# Create a profile of the leads which are likely to convert
```

Importing necessary libraries and data

In [ ]:
```
# libraries to help read and manipluate data
import pandas as pd
import numpy as np

# libararies for plotting and visualizing the data
import matplotlib as mpl
from matplotlib import pyplot as plt
import seaborn as sns

# libraries for supervised ML methods of classification
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier

# library for tuning models
from sklearn.model_selection import GridSearchCV

# libaries for colculating score for model validation
import sklearn.metrics as metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

Data Overview

- Observations
- Sanity checks

In [ ]:
```
from google.colab import drive
drive.mount('/content/drive', force_remount = True)
```

```
Mounted at /content/drive
```

In [ ]:
```
original_df = pd.read_csv('/content/drive/MyDrive/classes/Summer 2025/MIT - Data Sc
```

In [ ]:
```
original_df.head()
```

Out[ ]:

	ID	age	current_occupation	first_interaction	profile_completed	website_visits	tim
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 [ ]:
```
original_df.tail()
```

Out[ ]:

	ID	age	current_occupation	first_interaction	profile_completed	website_visits
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

In [ ]:
```
original_df.info()
original_df.isnull().sum()

# there are no null values
```

```
<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
```

Out[ ]:

	0
ID	0
age	0
current_occupation	0
first_interaction	0
profile_completed	0
website_visits	0
time_spent_on_website	0
page_views_per_visit	0
last_activity	0
print_media_type1	0
print_media_type2	0
digital_media	0
educational_channels	0
referral	0
status	0

dtype: int64

In [ ]:
```
original_df.describe() #only describes numerical features
```

Out[ ]:

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

In [ ]:
```
original_df.duplicated().sum()
# there are no duplicated values
```

Out[ ]:
```
np.int64(0)
```

In [ ]:
```
original_df.shape
# there are 4612 rows and 15 columns (features)
```

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

In [ ]:
```
original_df.dtypes
# object (integer and string) features: ID, occupation, first_interaction, profile_
# integer/ float features: age, website_visits, time_spend_on_website, page_views_p

# all of these datatypes make sense for the description of data
# status is a binary classification column (1 for paid, 0 for unpaid)
```

Out[ ]:

	0
ID	object
age	int64
current_occupation	object
first_interaction	object
profile_completed	object
website_visits	int64
time_spent_on_website	int64
page_views_per_visit	float64
last_activity	object
print_media_type1	object
print_media_type2	object
digital_media	object
educational_channels	object
referral	object
status	int64

dtype: object

In [ ]:
```
original_df.nunique()

# Id should have 4612 unique values since all ID #'s are different
# statistics relating to the company website has the seond and third most unique va
# there are 46 unique years of age amongst leads
# all other features have at least 2 unique values
```

Out[ ]:

	0
ID	4612
age	46
current_occupation	3
first_interaction	2
profile_completed	3
website_visits	27
time_spent_on_website	1623
page_views_per_visit	2414
last_activity	3
print_media_type1	2
print_media_type2	2
digital_media	2
educational_channels	2
referral	2
status	2

dtype: int64

Observations:

1. The data is comprised of 15 columns (features) and 4612 non-null values in each column.
2. There is a mix of object, integer and float datatypes. The last six features are binarily classified.
3. The age of leads ranges from 18 - 63 years, with the mean being 46 years.
4. The time spent on website feature has a wide range with a large standard deviation and a minimum of zero minutes. These numbers would probably be more interpretable by filling the zero values with the mean or median of the feature.
5. The mean of the status column is closer to zero than to one, suggesting that most leads are not paying customers yet.

Exploratory Data Analysis (EDA)

- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.

- A few questions have been mentioned below which will help you approach the analysis in the right manner and generate insights from the data.
- A thorough analysis of the data, in addition to the questions mentioned below, should be done.

Questions

1. Leads will have different expectations from the outcome of the course and the current occupation may play a key role in getting them to participate in the program. Find out how current occupation affects lead status.
2. The company's first impression on the customer must have an impact. Do the first channels of interaction have an impact on the lead status?
3. The company uses multiple modes to interact with prospects. Which way of interaction works best?
4. The company gets leads from various channels such as print media, digital media, referrals, etc. Which of these channels have the highest lead conversion rate?
5. People browsing the website or mobile application are generally required to create a profile by sharing their personal data before they can access additional information.Does having more details about a prospect increase the chances of conversion?

In [ ]:
```
# Univariate Analysis
# current_occupation vs. status
# first_interaction vs. status
# last_activity vs. status
# media (3 features) vs. status
# profile_completed vs. status
```

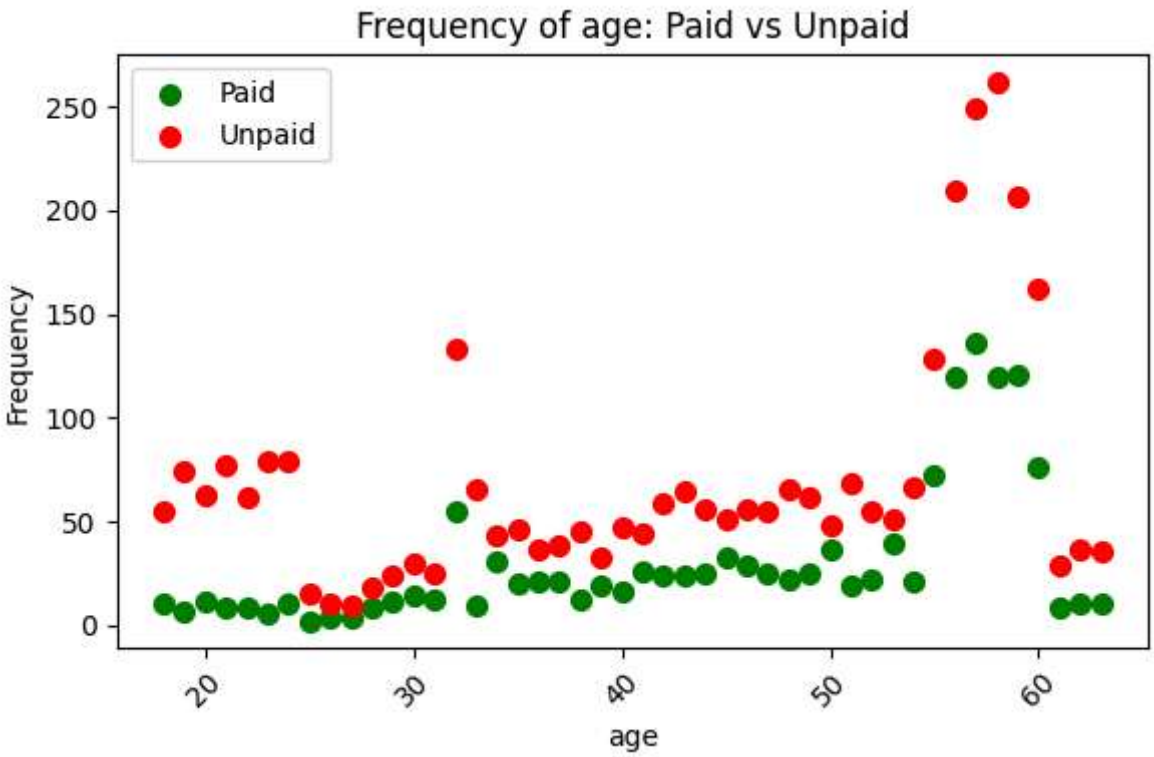
In [ ]:
```
original_df.head()
```

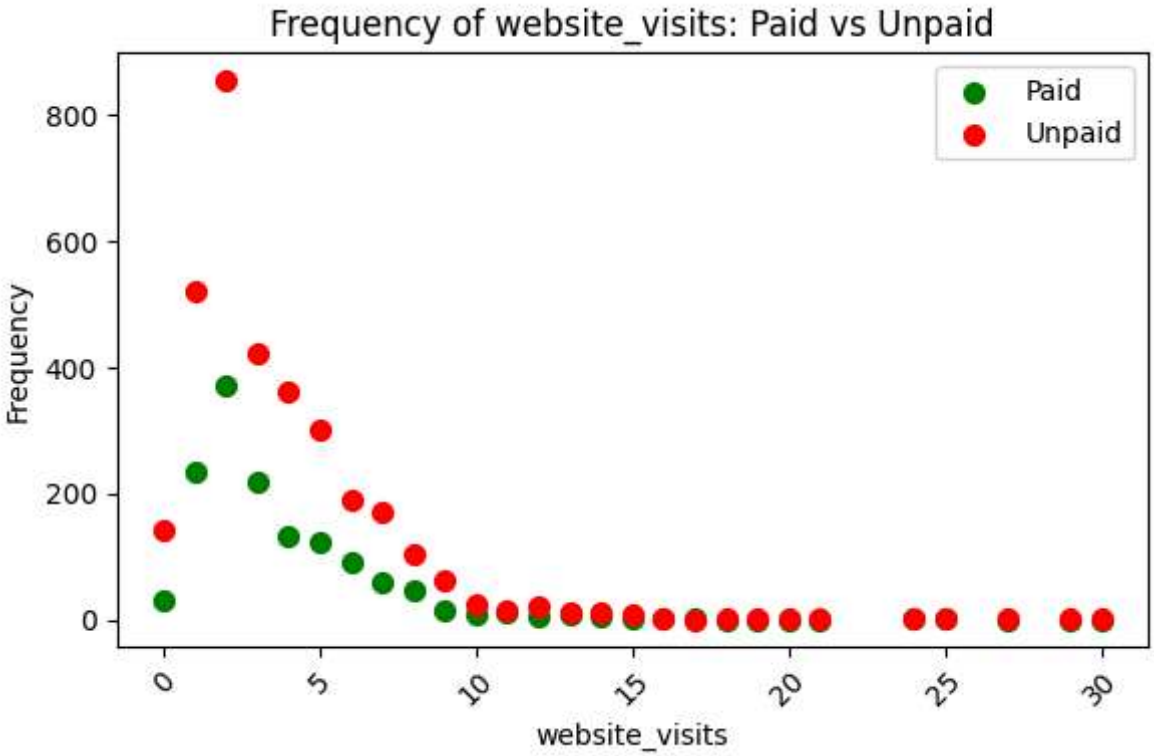
Out[ ]:

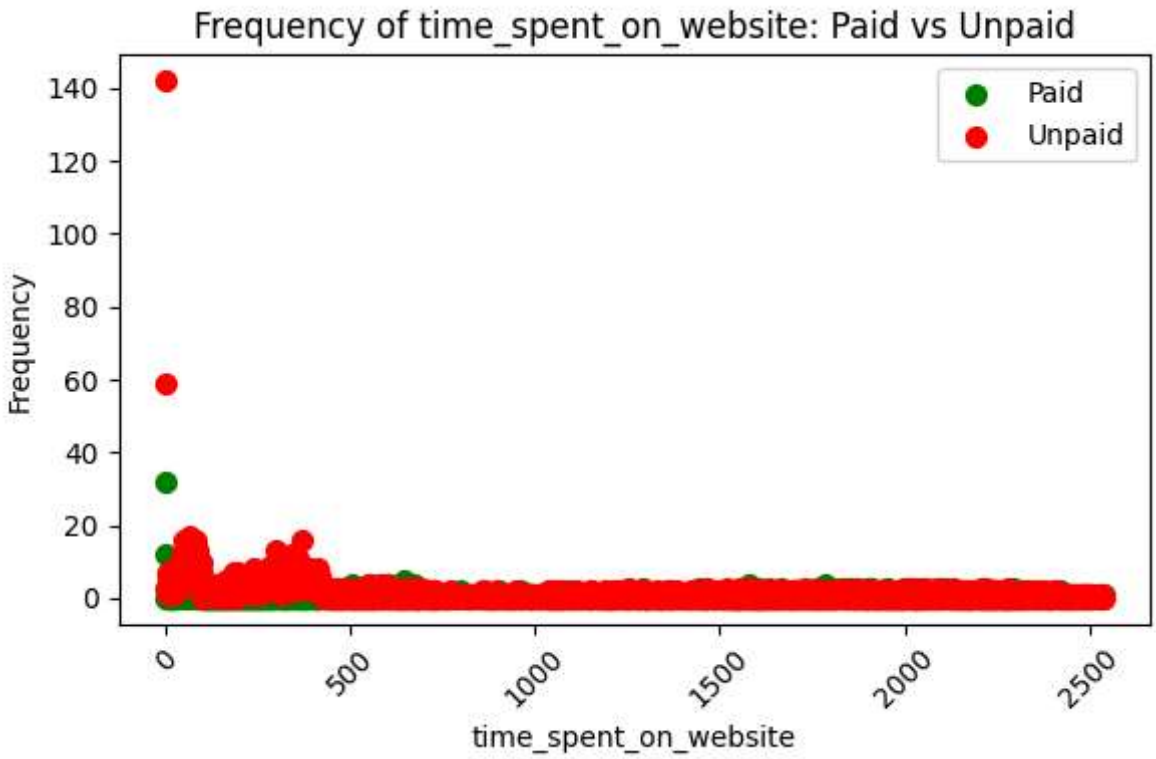
	ID	age	current_occupation	first_interaction	profile_completed	website_visits	tim
0	EXT001	57	Unemployed	Website	High	7	
1	EXT002	56	Professional	Mobile App	Medium	2	
2	EXT003	52	Professional	Website	Medium	3	
3	EXT004	53	Unemployed	Website	High	4	
4	EXT005	23	Student	Website	High	4	

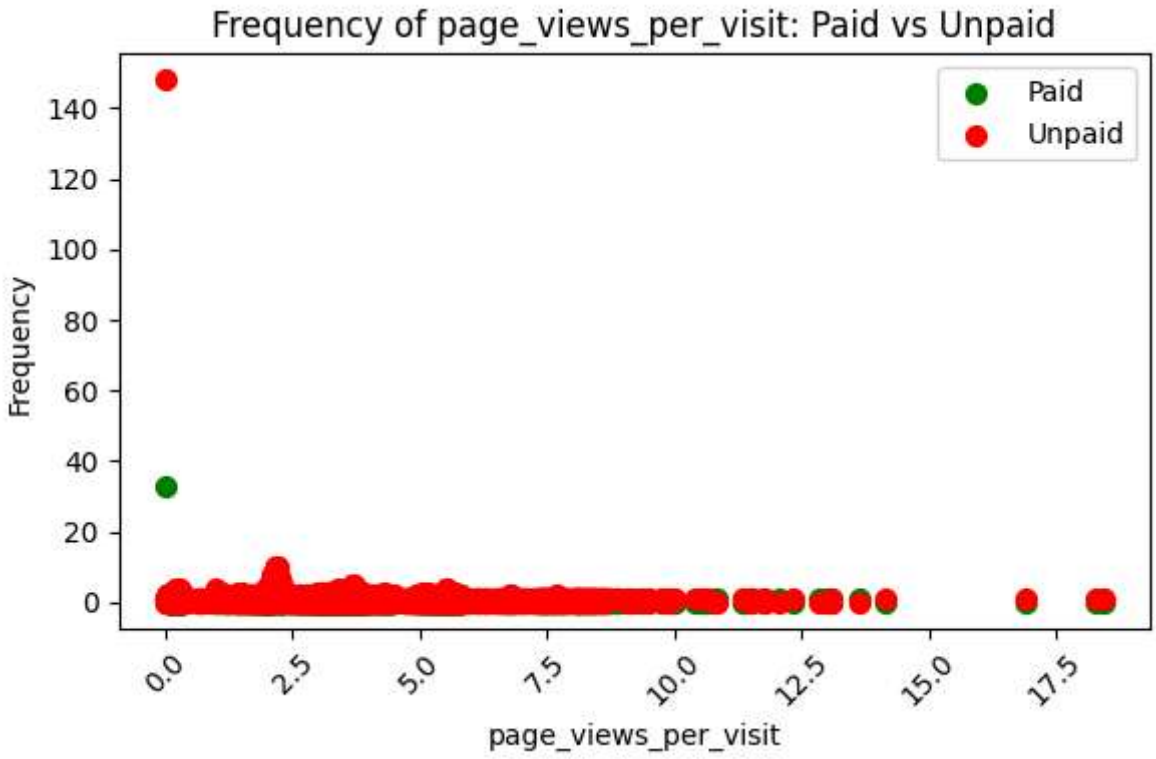
In [ ]:
```
#subsetting the data
original_df = original_df.drop(columns=['ID'])
paid_customer = original_df.loc[original_df['status'] == 1]
unpaid_customer = original_df.loc[original_df['status'] == 0]

# creating categorical vs. numerical vs. binary feature lists
numerical_features = original_df.select_dtypes(include=[np.number]).columns.tolist(
categorical_features = original_df.select_dtypes(exclude=[np.number]).columns.tolis
binary_features = [
    col for col in numerical_features
    if original_df[col].nunique() <=3
] # binary features

# creating a for loop for each feature selected in the numerical features
for feature in numerical_features:
  if feature == 'status':
    continue
  counts_paid = paid_customer[feature].value_counts()
  counts_unpaid = unpaid_customer[feature].value_counts()
  # sort the values of the features into either paid or unpaid based on the status
  all_values = sorted(set(counts_paid.index).union(counts_unpaid.index))
  # convert to lists
  paid_freq = [counts_paid.get(val, 0) for val in all_values]
  unpaid_freq = [counts_unpaid.get(val, 0) for val in all_values]
  # create scatter plots
  plt.figure(figsize=(6,4))
  plt.scatter(all_values, paid_freq, color='green', label='Paid', s=50)
  plt.scatter(all_values, unpaid_freq, color='red', label='Unpaid', s=50)
  plt.title(f'Frequency of {feature}: Paid vs Unpaid')
  plt.xlabel(feature)
  plt.ylabel('Frequency')
  plt.legend()
  plt.xticks(rotation=45)  # rotate x labels if categorical
  plt.tight_layout()
  plt.show()

# creating a for loop for each feature selected in the categorical and binary featu
for feature in categorical_features + binary_features:
  counts_paid = paid_customer[feature].value_counts()
  counts_unpaid = unpaid_customer[feature].value_counts()
  # sort the values of the features into either paid or unpaid based on the status
  all_values = sorted(set(counts_paid.index).union(counts_unpaid.index))
  # convert to lists
  paid_freq = [counts_paid.get(val, 0) for val in all_values]
  unpaid_freq = [counts_unpaid.get(val, 0) for val in all_values]
  # create bar graphs
  x = range(len(all_values))
  plt.figure(figsize=(6,4))
  plt.bar(x, paid_freq, width=0.4, label='Paid', align='center')
  plt.bar([p+0.4 for p in x], unpaid_freq, width=0.4, label='Unpaid', align='center
  plt.xticks([p+0.2 for p in x], all_values, rotation=45)
  plt.title(f'Frequency of {feature}: Paid vs Unpaid')
  plt.ylabel('Frequency')
  plt.legend()
```

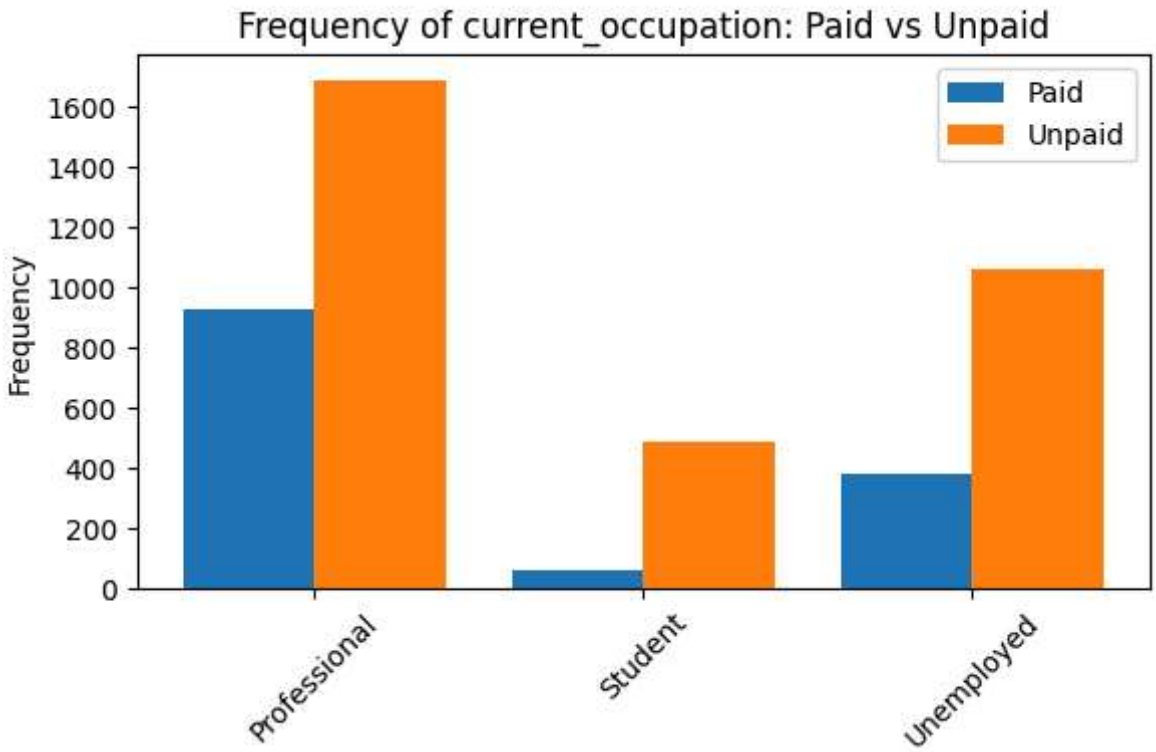
```
plt.tight_layout()
plt.show()
```

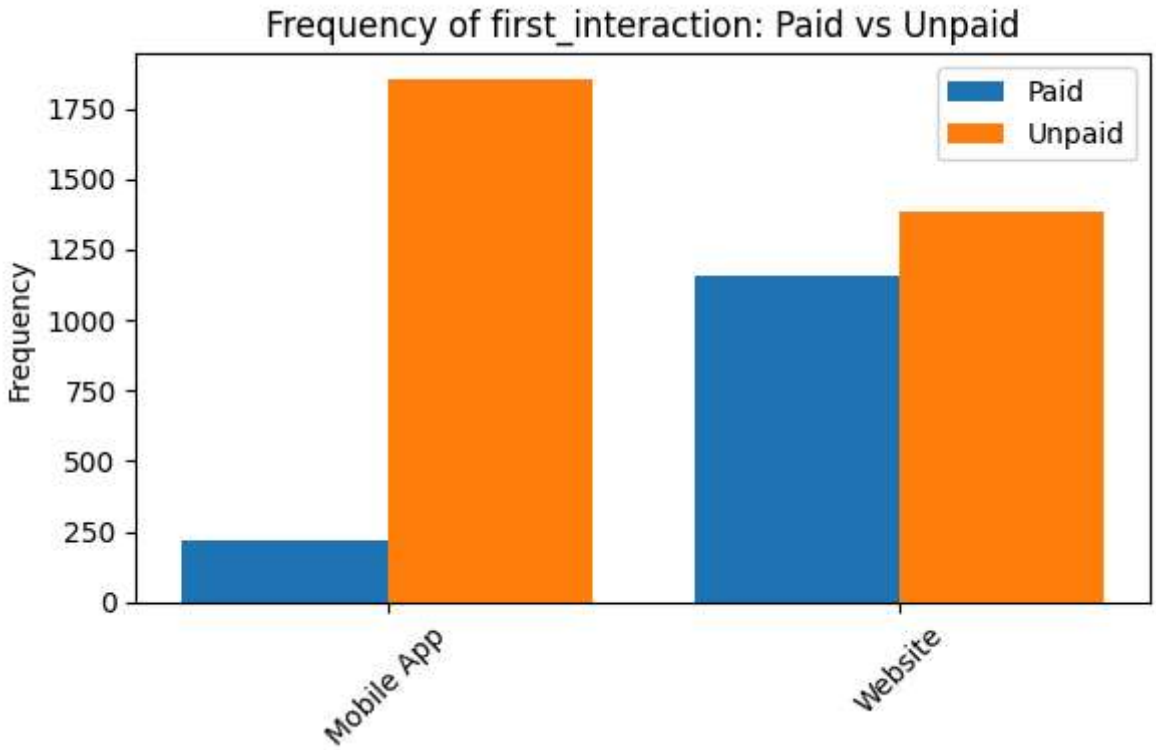


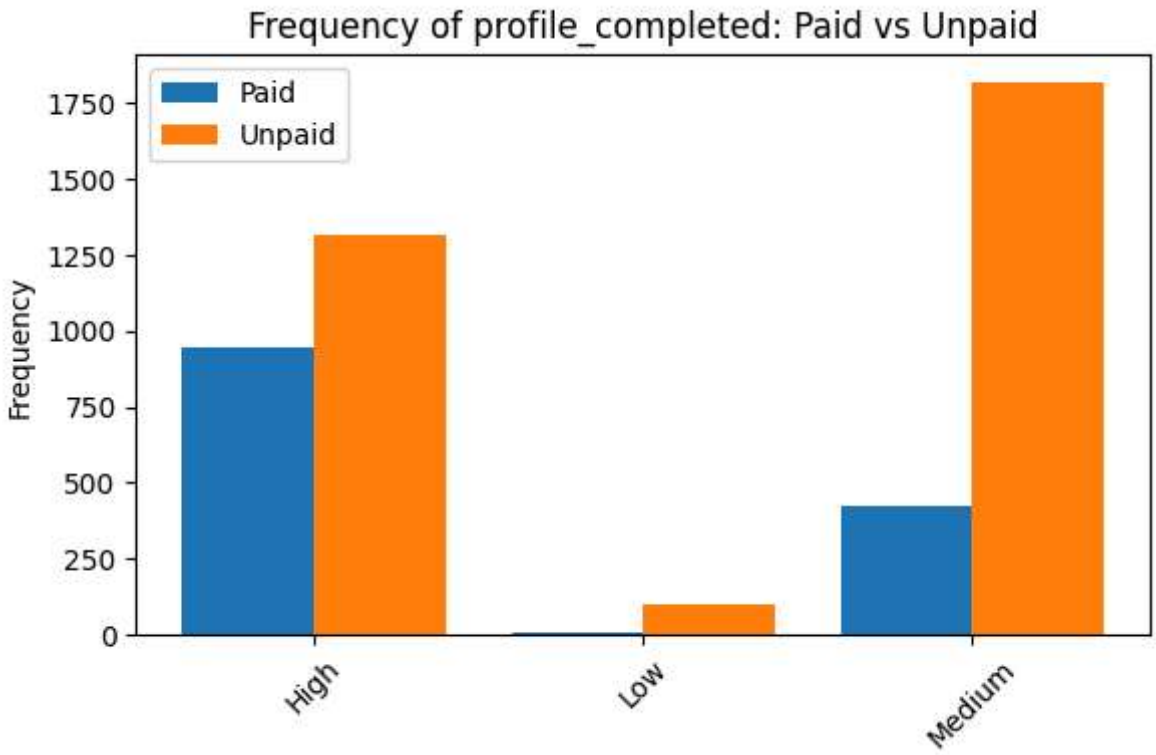


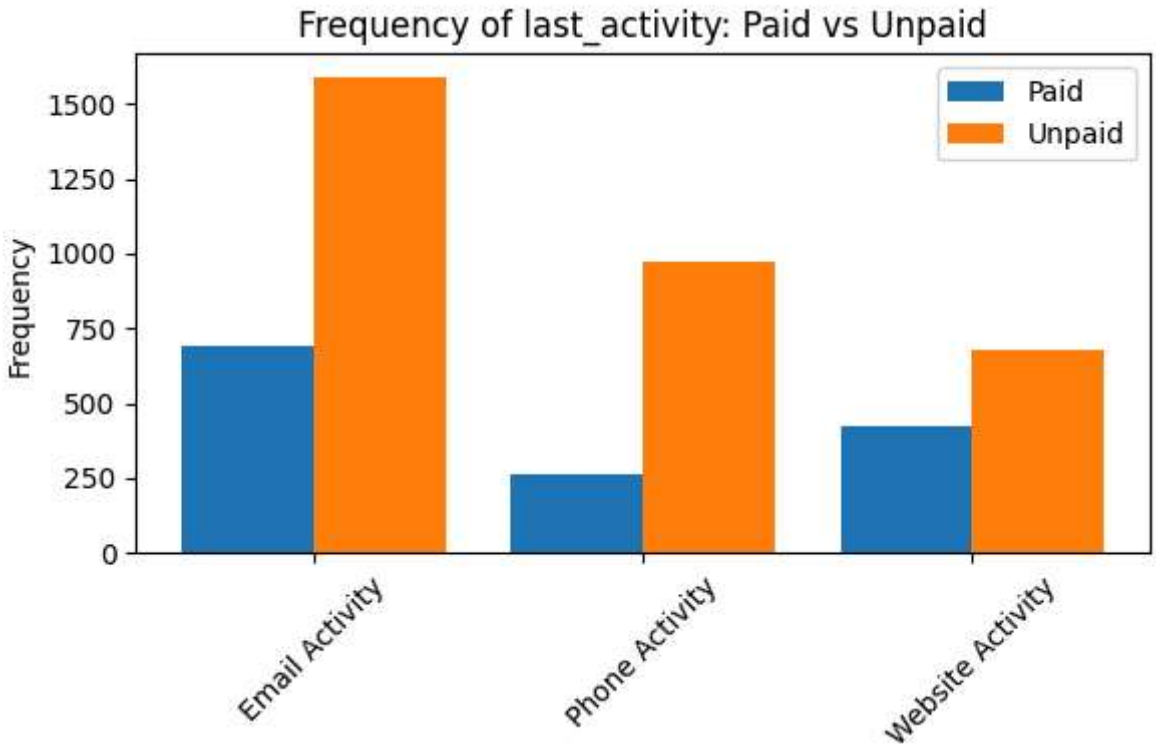

Frequency of time_spent_on_website: Paid vs Unpaid
Paid
Unpaid
Frequency
0
20
40
60
80
100
120
140
0
500
1000
1500
2000
2500
time_spent_on_website

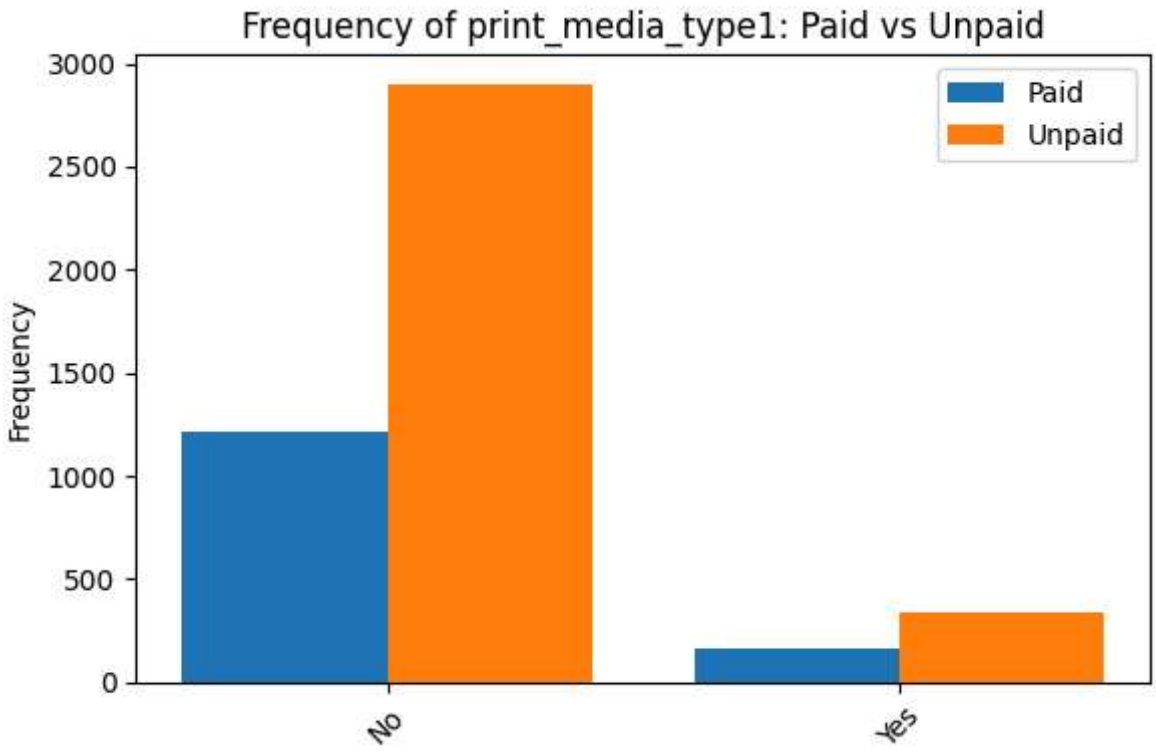

Frequency of page_views_per_visit: Paid vs Unpaid
Paid
Unpaid
Frequency
0
20
40
60
80
100
120
140
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
page_views_per_visit

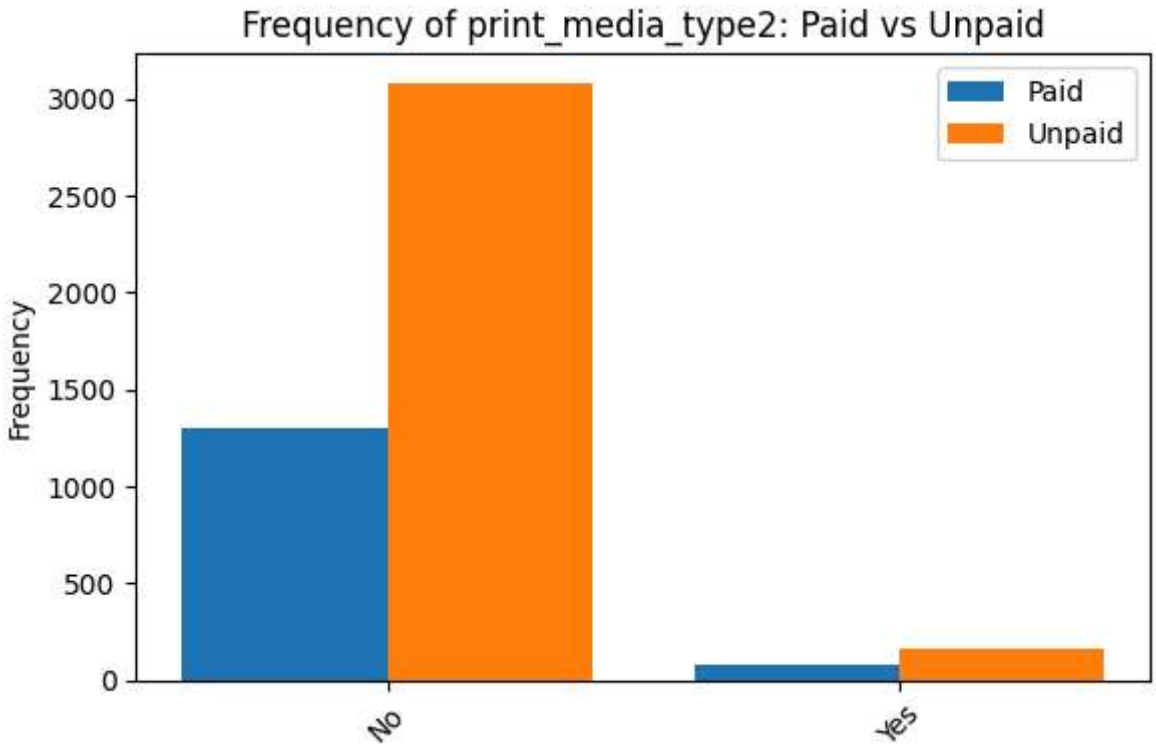

Frequency of current_occupation: Paid vs Unpaid
Paid
Unpaid
Frequency
0
200
400
600
800
1000
1200
1400
1600
Professional
Student
Unemployed

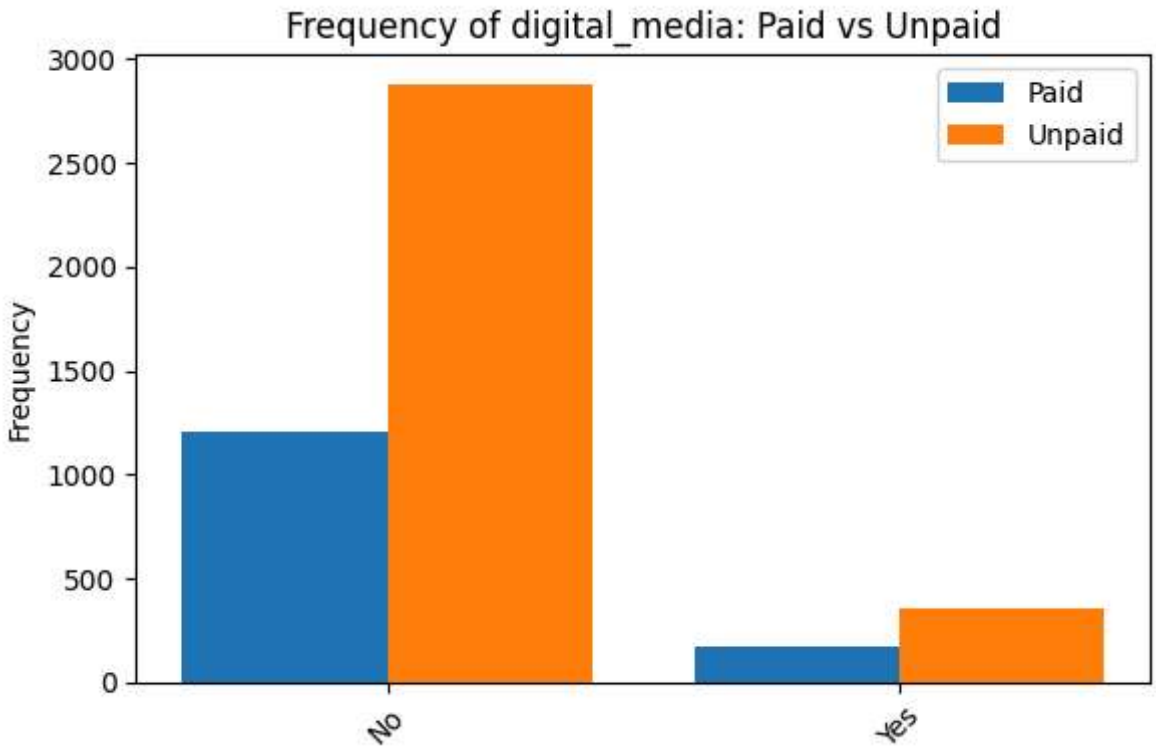

Frequency of first_interaction: Paid vs Unpaid
Paid
Unpaid
Frequency
0
250
500
750
1000
1250
1500
1750
Mobile App
Website

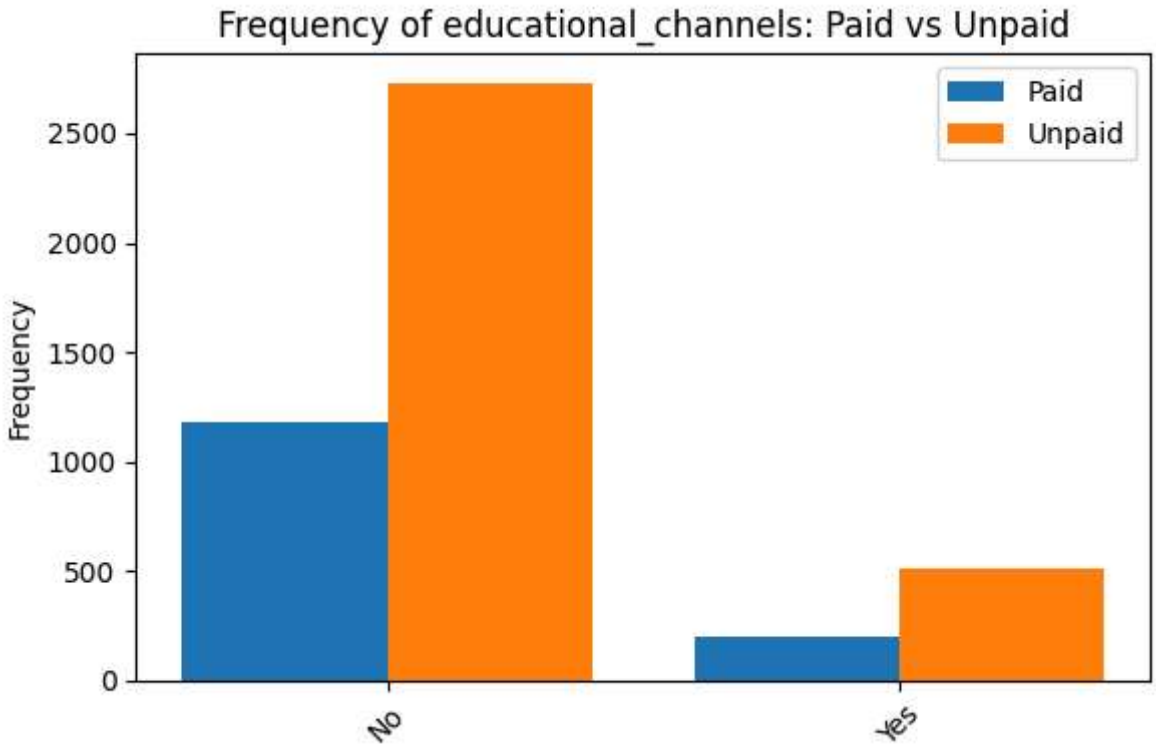

Frequency of profile_completed: Paid vs Unpaid
Paid
Unpaid
Frequency
0
250
500
750
1000
1250
1500
1750
High
Low
Medium

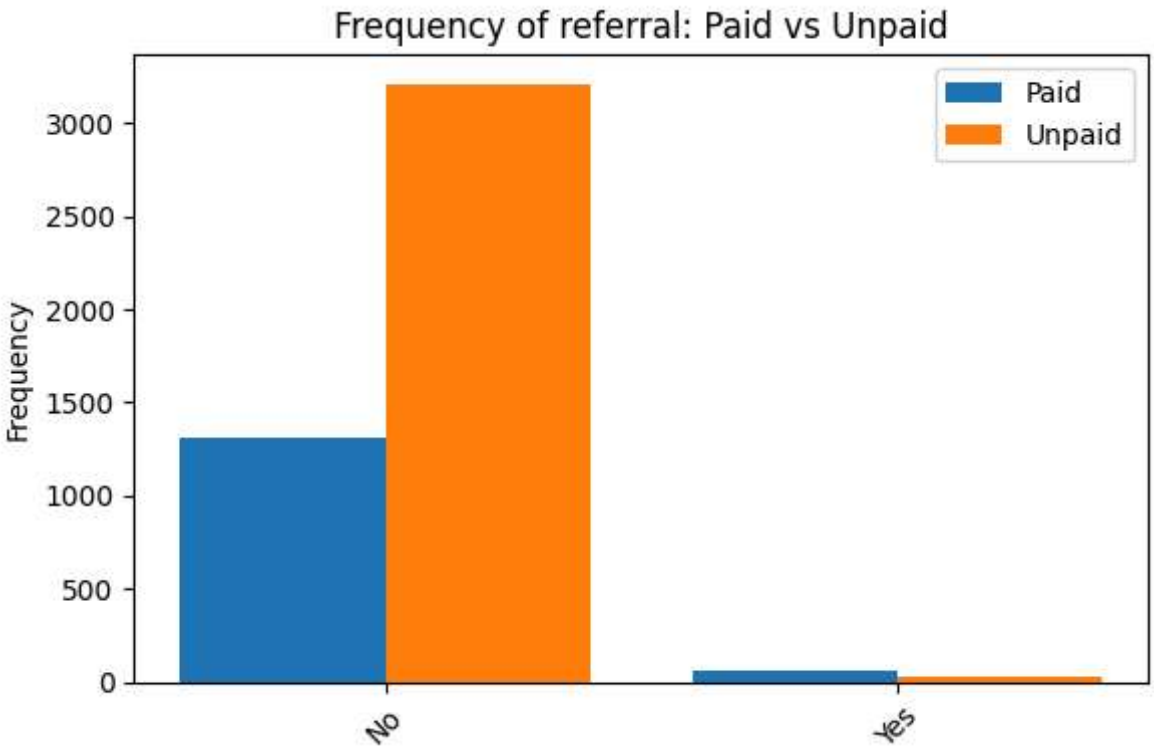

Frequency of last_activity: Paid vs Unpaid
Paid
Unpaid
Frequency
0
250
500
750
1000
1250
1500
Email Activity
Phone Activity
Website Activity

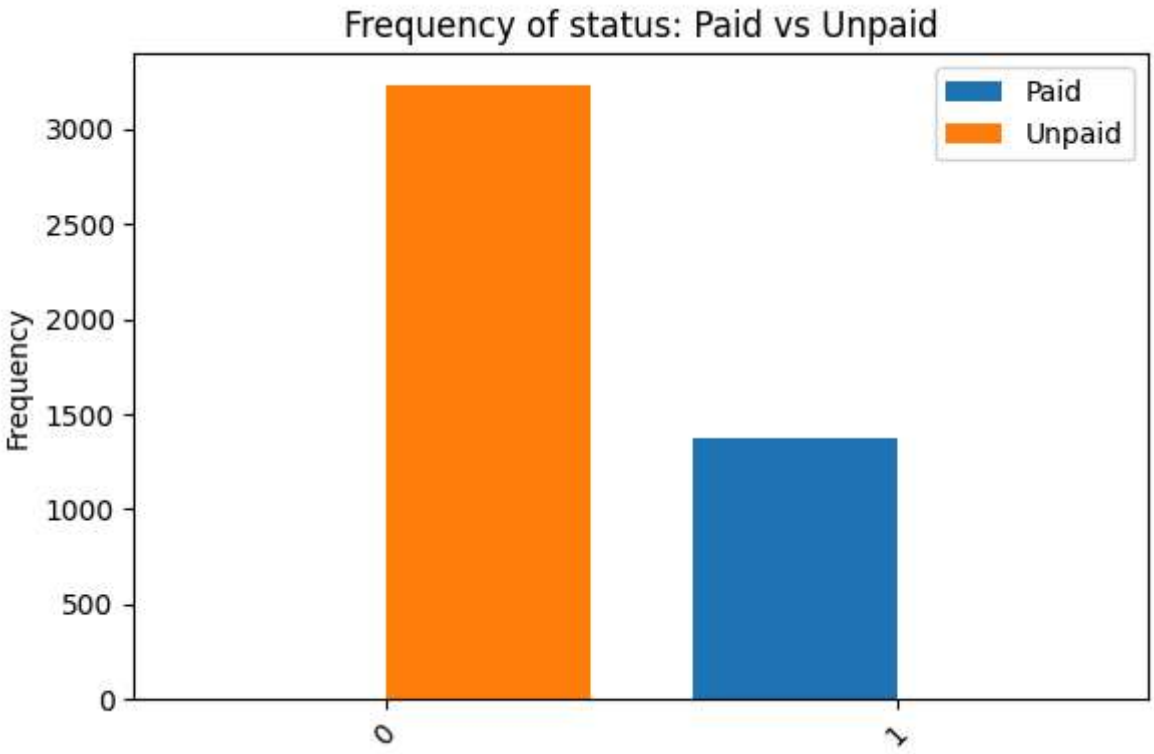

Frequency of print_media_type1: Paid vs Unpaid
Paid
Unpaid
Frequency
3000
2500
2000
1500
1000
500
0
No
Yes


Frequency of print_media_type2: Paid vs Unpaid
Paid
Unpaid
Frequency
3000
2500
2000
1500
1000
500
0
No
Yes


Frequency of digital_media: Paid vs Unpaid
Paid
Unpaid
Frequency
3000
2500
2000
1500
1000
500
0
No
Yes


Frequency of educational_channels: Paid vs Unpaid
Paid
Unpaid
Frequency
2500
2000
1500
1000
500
0
No
Yes



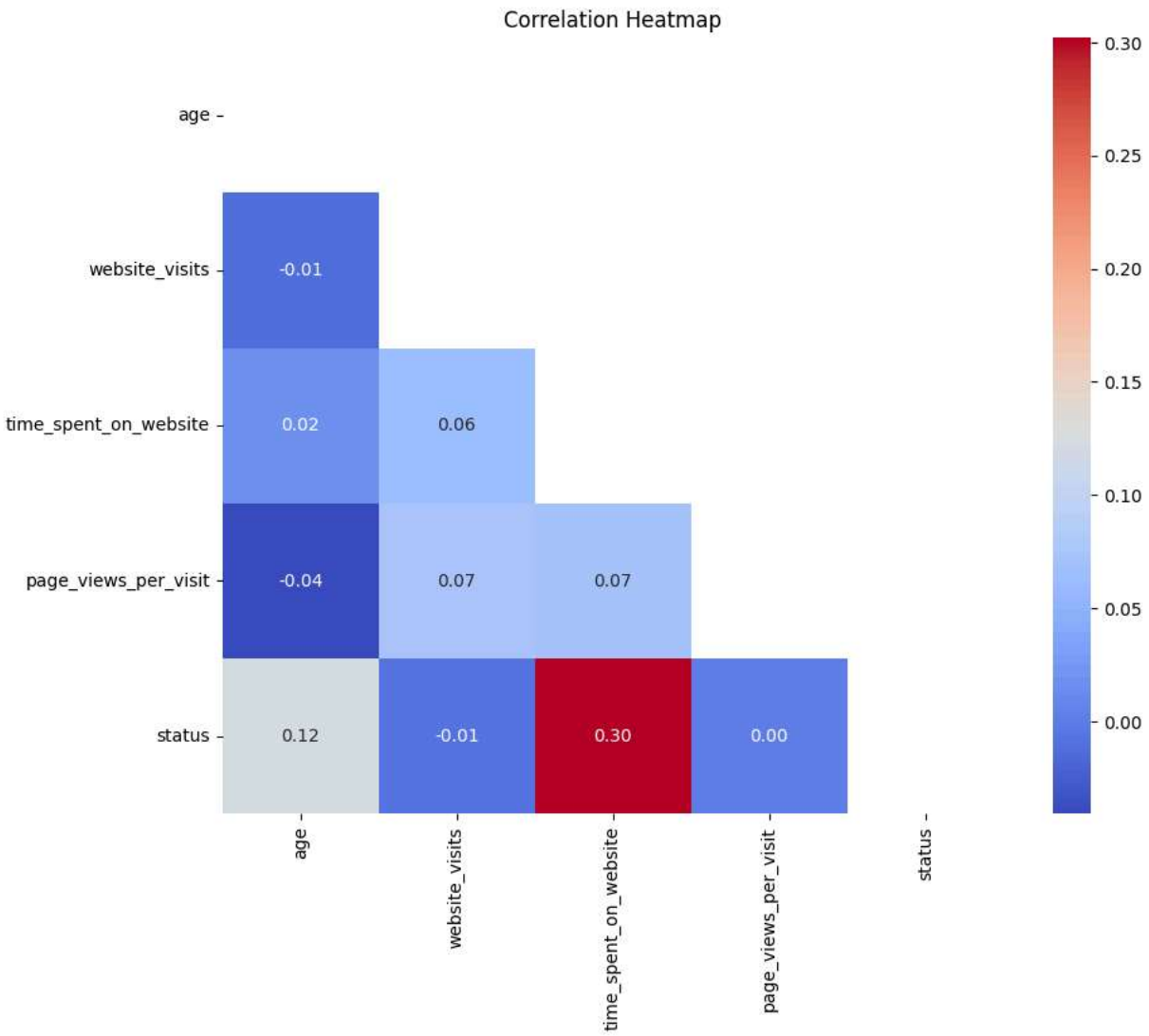


Observations: It is consistant across the board that for every category and graph (besides the "yes" category for refferals), that there are more unpaid leads than paid leads.

1. The age of both paid and unpaid spikes between 50 and 60 years, suggesting that most of the target audience is within the age range.
2. Website visits spike from 0 visits to 10 visits.

3. Frequencies of time spent on web is slightly more frequent between 0 and 500 minutes. There are so many data points on this graph it is hard to read. The number of data points could be reduced by using a sample.
4. Frequencies of page visits does not fluxuate much, but does spike a little at 2.5 pages. There are so many data points on this graph it is hard to read. The number of data points could be reduced by using a sample.
5. Over half of the paid leads have a career change, whist majority of the rest are unemployed, and few are students. This suggests that those in a career are either looking for growth opportunities, or looking to change careers, and both those without employment and students are looking to gain experience for a career.
6. Most of the first interactions are paid leads if the first interaction was the website, whist most of the mobile first interactions are still unpaid. This suggests poor marketing or incorrect targeting efforts on mobile.
7. It seems as though, those who complete more of their profile are more likely to be paid leads, since the "high" completion category has a high conversion rate, whist the "medium" and "low" completion categories have low conversion rates.
8. Website and email activity have more paid leads, while phone has less paid leads but more unpaid leads. This could connect to the issue of the first interaction being mobile.
9. Media types and refferals have very few leads as part of the "yes" collumns, and the ratio of paid leads to unpaid leads for these categories is similar. This suggests that the current media and refferal system does not have a significant impact on increasing the number of paid leads.

In [ ]:
```
correlation_matrix = original_df[numerical_features].corr()
mask = np.triu(np.ones_like(correlation_matrix, dtype=bool))
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", mask=mask)
plt.title("Correlation Heatmap")
plt.show()
```



Observations

1. Correlations with status from highest to lowest: time_spent_on_website, age, wpage_views_per_visit, and website_visits
2. Age has a negative correlation with all website categories

Data Preprocessing

- Missing value treatment (if needed)
- Feature engineering (if needed)
- Outlier detection and treatment (if needed)
- Preparing data for modeling
- Any other preprocessing steps (if needed)

1. There are no missing values.
2. Some features need to be engineered: some continuous variables (EX: age) can be binned, one-hot encoding for low cardinality, and frequency encoding for high

cardinality

3. Use Z-score to find outliers in continuous variable graphs, and remove outliers above or below a certain threashold. NOTE: Decsision trees and random forests are good at handling outliers because they split based on a threshold; however, outliers should still be removed for clarity.
4. StandardScalar is not used for decision trees, so only train_test_split is needed.
5. Decisions trees will have high bias towards the majoity classes, so since there are much more unpaid leads than paid leads, this might have to be remedied by balancing the two, or by using undersampling. (This was taken care of in step 3 when creating the pipeline.)

In [ ]:
```
pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.8.1-py3-none-any.whl.metadata (7.9 kB)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.12/dist-packa
ges (from category_encoders) (2.0.2)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.12/dist-packa
ges (from category_encoders) (2.2.2)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.12/dist-packag
es (from category_encoders) (1.0.1)
Requirement already satisfied: scikit-learn>=1.6.0 in /usr/local/lib/python3.12/dist
-packages (from category_encoders) (1.6.1)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.12/dist-packag
es (from category_encoders) (1.16.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.12/dist-
packages (from category_encoders) (0.14.5)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/d
ist-packages (from pandas>=1.0.5->category_encoders) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packag
es (from pandas>=1.0.5->category_encoders) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-pack
ages (from pandas>=1.0.5->category_encoders) (2025.2)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packa
ges (from scikit-learn>=1.6.0->category_encoders) (1.5.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dis
t-packages (from scikit-learn>=1.6.0->category_encoders) (3.6.0)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.12/dist-pac
kages (from statsmodels>=0.9.0->category_encoders) (25.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages
(from python-dateutil>=2.8.2->pandas>=1.0.5->category_encoders) (1.17.0)
Downloading category_encoders-2.8.1-py3-none-any.whl (85 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 85.7/85.7 kB 2.3 MB/s eta 0:00:00
Installing collected packages: category_encoders
Successfully installed category_encoders-2.8.1
```

In [ ]:
```
# 2. feature engineering
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import category_encoders as ce

# First, split the dataset into features (x) and the target (y).
```

```
# 'status' is what we want to predict (1 = paid lead, 0 = not paid lead),
# so we take it out of the feature set and keep it as y.
x = original_df.drop('status', axis=1)
y = original_df['status']

# Now we want to separate the categorical features into two groups:
# 1. "Low cardinality" → categories with 10 or fewer unique values
#    (like gender, weekdays, yes/no type stuff).
# 2. "High cardinality" → categories with more than 10 unique values
#    (like city names, job titles, product IDs, etc.).
low_card_cat = [col for col in categorical_features if original_df[col].nunique() <
high_card_cat = [col for col in categorical_features if original_df[col].nunique()

# Decision trees can already handle numbers just fine, so no need to encode numeric

# For binary features (like Yes/No, Male/Female), we'll just use an OrdinalEncoder
# to map them to numbers (e.g. Yes=1, No=0).
bin_transformer = OrdinalEncoder()

# For low-cardinality categorical features, OneHotEncoder is best.
# This will create a new column for each category (like dummy variables).
# "handle_unknown='ignore'" means if the model sees a category in test data
# that wasn't in training, it won't throw an error.
low_cat_transformer = OneHotEncoder(handle_unknown='ignore')

# For high-cardinality categorical features, OneHot would blow up the number of col
# Instead, we use TargetEncoder, which replaces each category with
# the average of the target (basically: how likely that category is to be a paid le
high_cat_transformer = ce.TargetEncoder(cols=high_card_cat)

# Now we combine all of these encoders into one preprocessing step
# using ColumnTransformer. This way, different groups of features
# get different encoders automatically.
preprocessor = ColumnTransformer(
    transformers=[
        ('bin', bin_transformer, binary_features),
        ('low_cat', low_cat_transformer, low_card_cat),
        ('high_cat', high_cat_transformer, high_card_cat)
    ]
)

# Finally, we wrap everything in a pipeline.
# Step 1: preprocess the data with the encoders above.
# Step 2: pass the processed data to a Random Forest classifier.
clf = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', RandomForestClassifier(
        n_estimators=200,        # build 200 trees in the forest
        random_state=42,         # keep results reproducible
        class_weight='balanced'  # helps if one class is much smaller than the othe
    ))
])

# The pipeline does not do anything until we fit the training and test data to it.
```

EDA

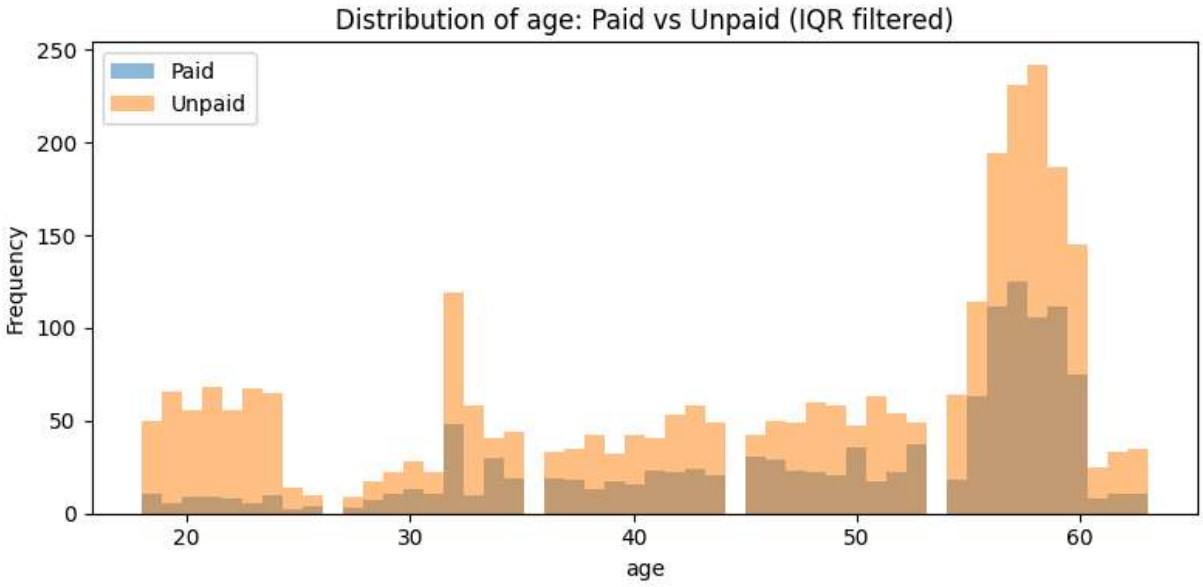
- It is a good idea to explore the data once again after manipulating it.

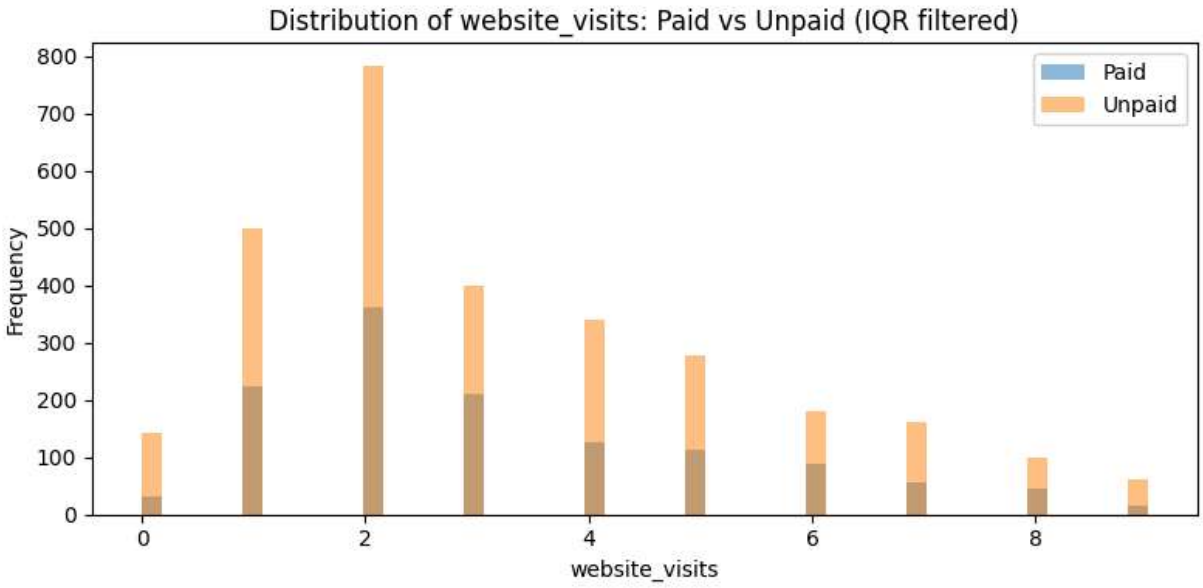
In [ ]:
```
# 3. Outlier Detection and Treatment
# decision trees deal with outliers well because there are thresholds they split at

# Create a helper function that removes outliers from a single column using the IQR
# IQR = Q3 - Q1 (the middle 50% of the data). Anything outside [Q1 - 1.5*IQR, Q3 +
def remove_outliers_iqr(data, column):
    Q1 = data[column].quantile(0.25)   # 25th percentile
    Q3 = data[column].quantile(0.75)   # 75th percentile
    IQR = Q3 - Q1                      # interquartile range
    lower = Q1 - 1.5 * IQR             # lower cutoff
    upper = Q3 + 1.5 * IQR             # upper cutoff
    return data[(data[column] >= lower) & (data[column] <= upper)]  # keep only row


# First try: remove outliers by applying the IQR method across *all* numeric column
df_no_outliers = original_df.copy()
for feature in numerical_features:
    if feature == 'status':
        continue
    # only apply if the column is numeric and has more than one unique value
    if pd.api.types.is_numeric_dtype(df_no_outliers[feature]) and df_no_outliers[fe
        df_no_outliers = remove_outliers_iqr(df_no_outliers, feature)

print(f"Rows before: {len(original_df)}, after global IQR filtering: {len(df_no_out

# If this global filtering deletes too much data (>50% gone, or literally zero rows
# then we won't use it globally. Instead, we'll filter outliers separately for each
if len(df_no_outliers) == 0 or len(df_no_outliers) < 0.5 * len(original_df):
    print("Global IQR filtering removed >50% (or 0) rows. Falling back to per-featu
    use_global_filter = False
else:
    use_global_filter = True


# Helper function to create masks for "paid" vs "unpaid" customers.
# Works whether 'status' is stored as 1/0, Paid/Unpaid, Yes/No, etc. (case insensit
def paid_unpaid_masks(series):
    s = series.astype(str).str.lower()
    paid_mask = s.isin(['1', 'paid', 'true', 't', 'yes', 'y'])
    unpaid_mask = s.isin(['0', 'unpaid', 'false', 'f', 'no', 'n'])
    return paid_mask, unpaid_mask

# Build full versions of the paid/unpaid splits (used if global filter failed).
paid_mask_full, unpaid_mask_full = paid_unpaid_masks(original_df['status'])
paid_customer_full = original_df[paid_mask_full]
unpaid_customer_full = original_df[unpaid_mask_full]

# If the global filter worked, also build paid/unpaid splits from the cleaned datas
if use_global_filter:
    paid_mask_g, unpaid_mask_g = paid_unpaid_masks(df_no_outliers['status'])
```

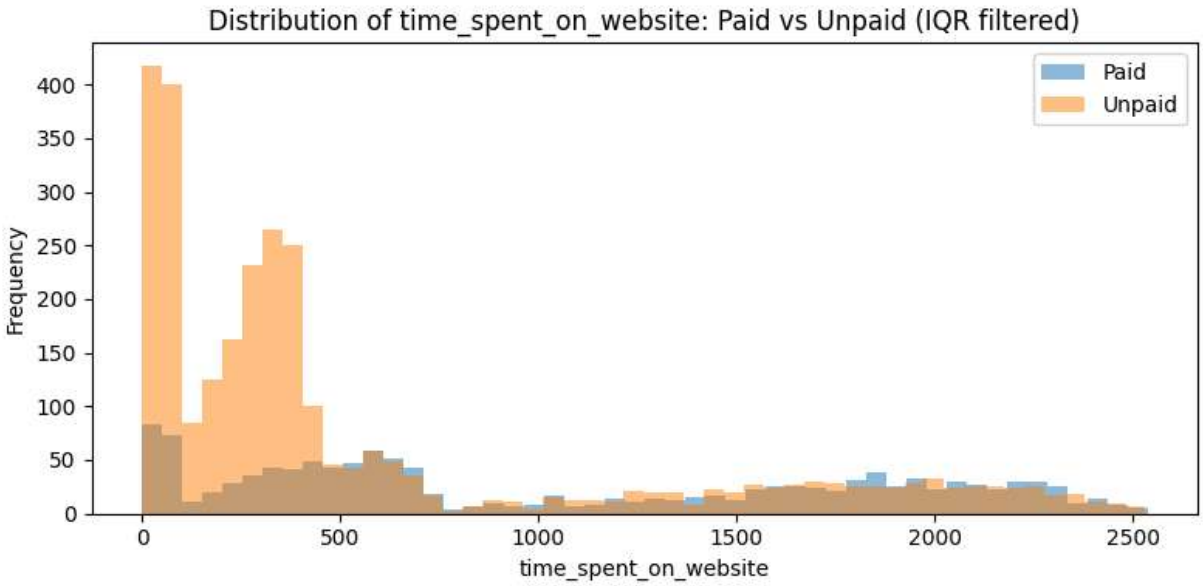
```
    paid_customer = df_no_outliers[paid_mask_g]
    unpaid_customer = df_no_outliers[unpaid_mask_g]
    print(f"After global filter -> Paid: {len(paid_customer)}, Unpaid: {len(unpaid_


# Now let's plot the numeric features to see distributions of Paid vs Unpaid custom
for feature in numerical_features:
    if feature == 'status':
        continue
    plt.figure(figsize=(8,4))

    if use_global_filter:
        # use globally filtered data
        data_paid = paid_customer[feature].dropna()
        data_unpaid = unpaid_customer[feature].dropna()
    else:
        # if global filtering didn't work, apply IQR per feature separately for pai
        df_paid_feat = remove_outliers_iqr(paid_customer_full, feature) if not paid
        df_unpaid_feat = remove_outliers_iqr(unpaid_customer_full, feature) if not
        data_paid = df_paid_feat[feature].dropna()
        data_unpaid = df_unpaid_feat[feature].dropna()

    # if both groups are completely empty after filtering, skip this feature
    if data_paid.empty and data_unpaid.empty:
        print(f"Skipping {feature}: no data after IQR filtering.")
        plt.close()
        continue

    # make histogram overlays (Paid vs Unpaid) for the feature
    plt.hist(data_paid, bins=50, alpha=0.5, label='Paid')
    plt.hist(data_unpaid, bins=50, alpha=0.5, label='Unpaid')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.title(f'Distribution of {feature}: Paid vs Unpaid (IQR filtered)')
    plt.legend()
    plt.tight_layout()
    plt.show()


# Now do the same for categorical/binary features, but use bar charts.
# Outlier removal isn't really relevant here, so we either use the globally filtere
# or the original one depending on what happened earlier.
df_for_cats = df_no_outliers if use_global_filter else original_df

for feature in categorical_features + binary_features:
    # Count frequencies for Paid group
    counts_paid = df_for_cats[df_for_cats['status'].astype(str).str.lower().isin(['
    # Count frequencies for Unpaid group
    counts_unpaid = df_for_cats[df_for_cats['status'].astype(str).str.lower().isin(

    # Combine categories across both groups so the bars line up properly
    all_values = sorted(set(counts_paid.index).union(counts_unpaid.index), key=lamb
    paid_freq = [counts_paid.get(val, 0) for val in all_values]
    unpaid_freq = [counts_unpaid.get(val, 0) for val in all_values]

    # Side-by-side bar chart (Paid vs Unpaid)
```

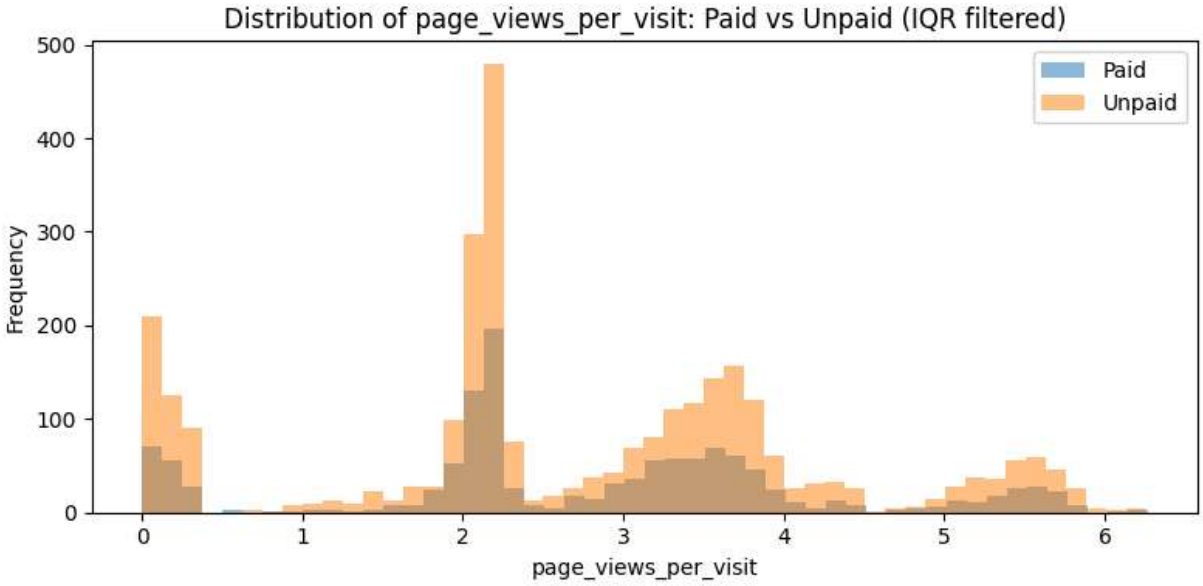
```
    x = np.arange(len(all_values))
    width = 0.35
    plt.figure(figsize=(8,4))
    plt.bar(x - width/2, paid_freq, width, label='Paid', alpha=0.7)
    plt.bar(x + width/2, unpaid_freq, width, label='Unpaid', alpha=0.7)
    plt.xticks(x, all_values, rotation=45)
    plt.title(f'Frequency of {feature}: Paid vs Unpaid')
    plt.ylabel('Frequency')
    plt.legend()
    plt.tight_layout()
    plt.show()
```

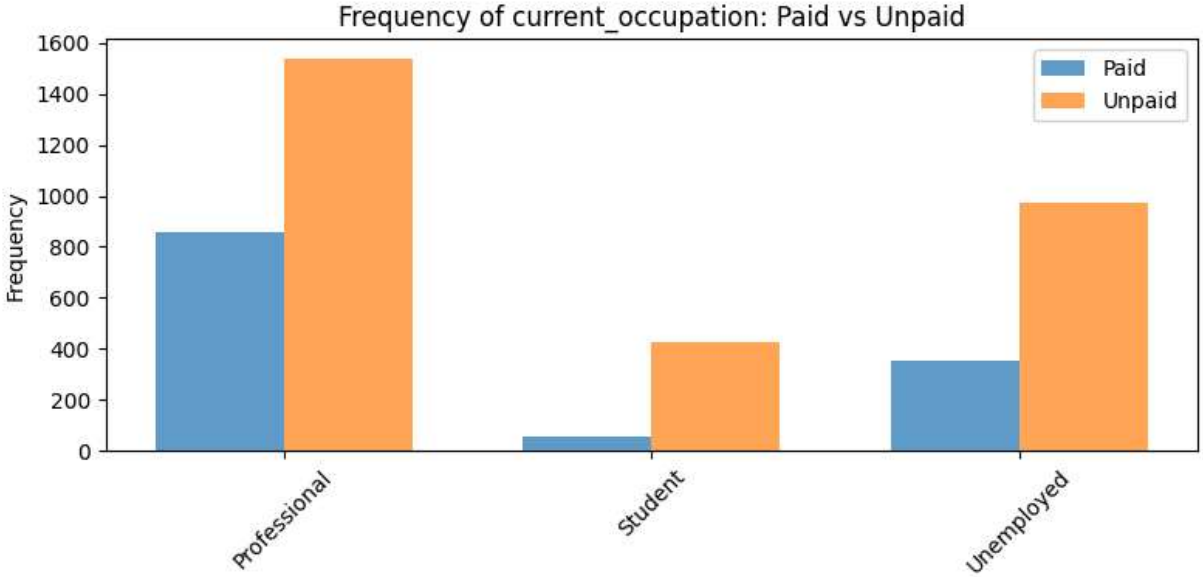
```
Rows before: 4612, after global IQR filtering: 4208
After global filter -> Paid: 1269, Unpaid: 2939
```

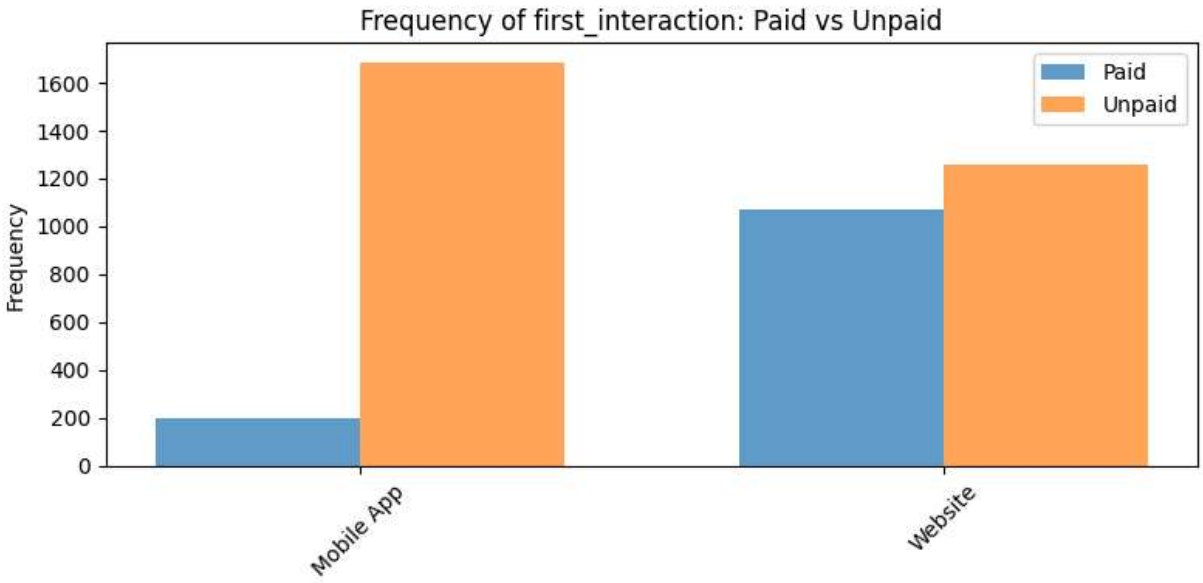


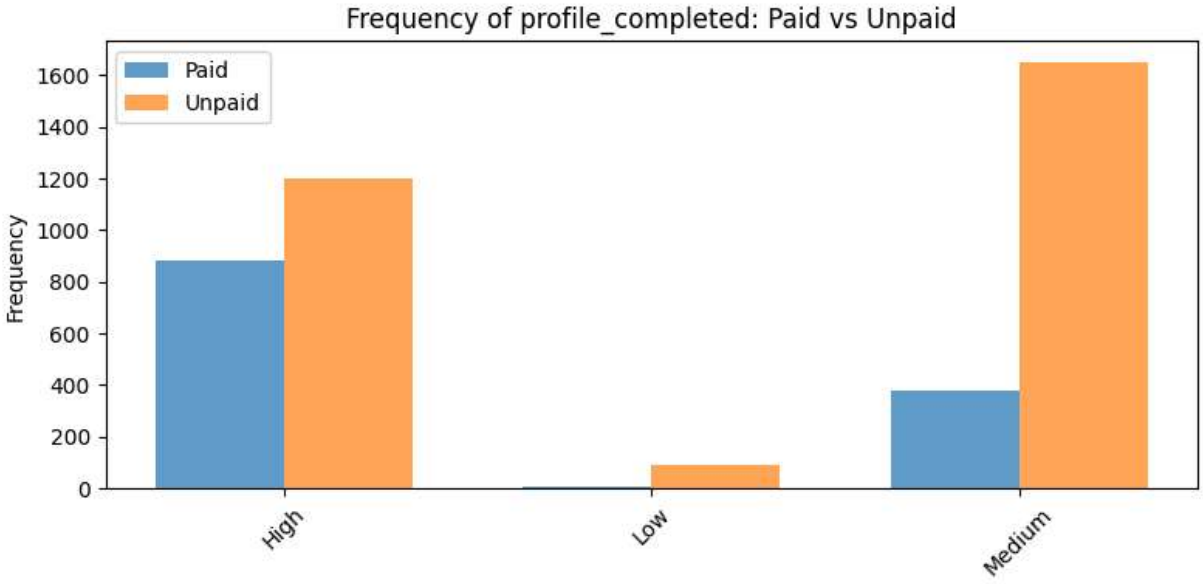


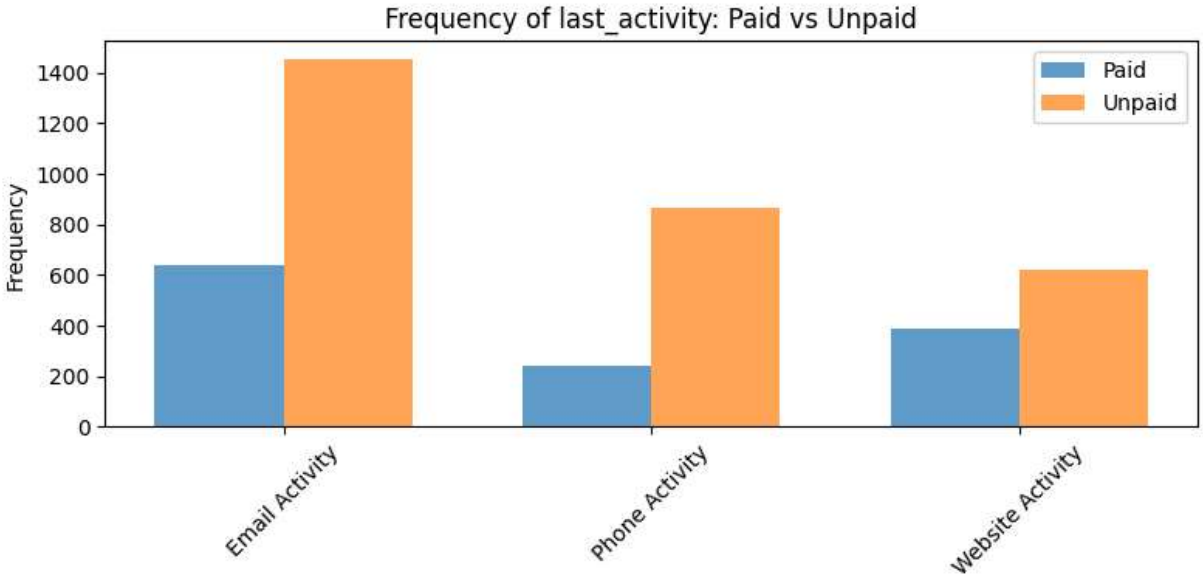

Distribution of time_spent_on_website: Paid vs Unpaid (IQR filtered)
Paid
Unpaid
Frequency
time_spent_on_website
0
50
100
150
200
250
300
350
400
0
500
1000
1500
2000
2500

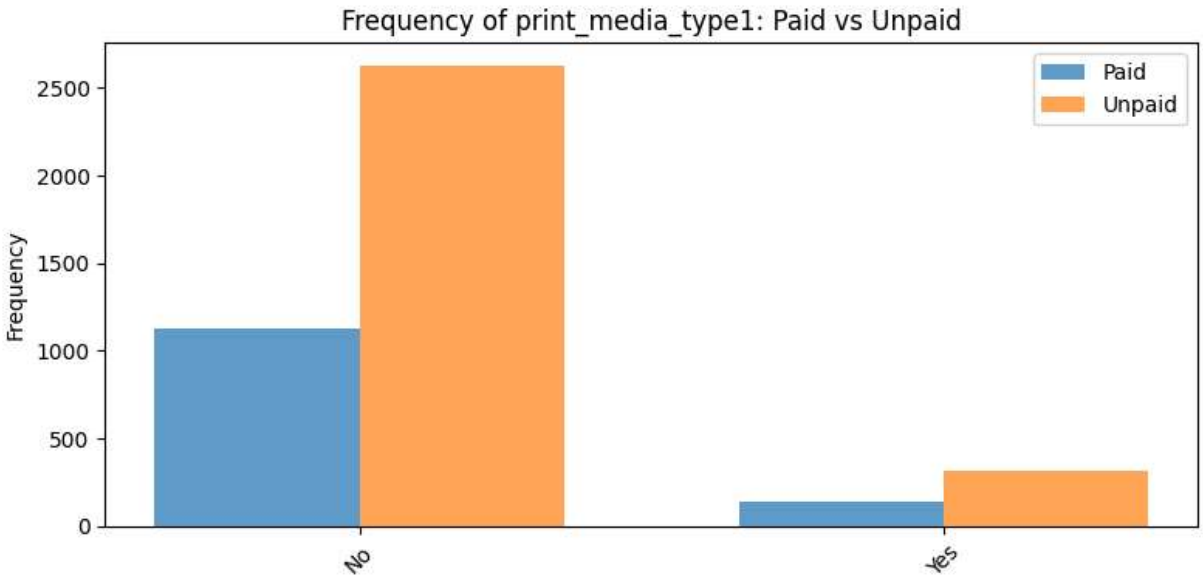

Distribution of page_views_per_visit: Paid vs Unpaid (IQR filtered)
Paid
Unpaid
Frequency
page_views_per_visit
0
100
200
300
400
500
0
1
2
3
4
5
6

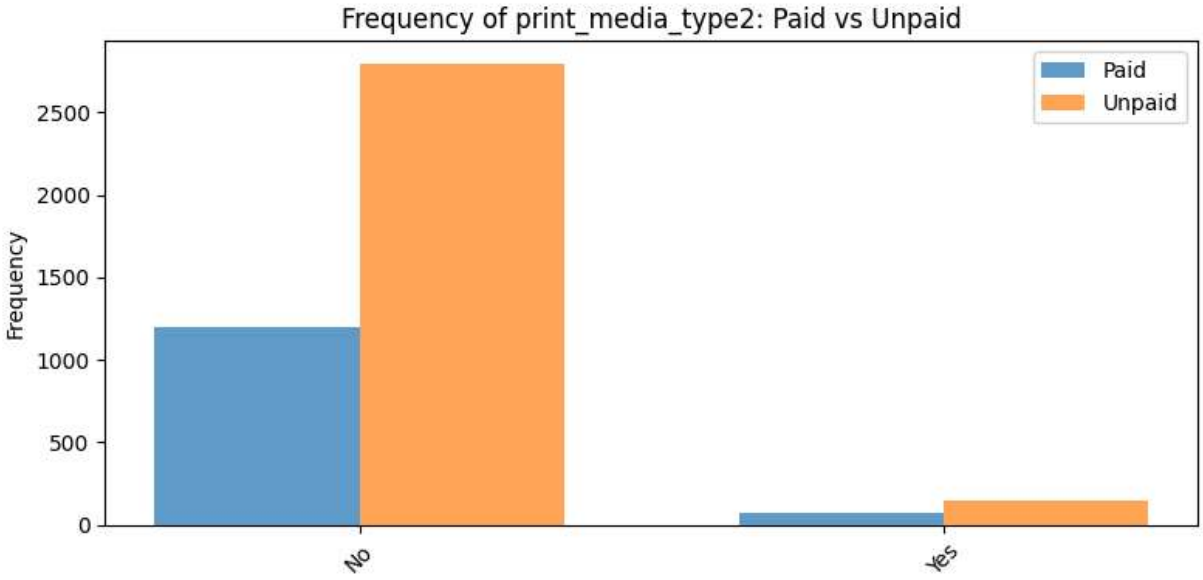

Frequency of current_occupation: Paid vs Unpaid
Paid
Unpaid
Frequency
0
200
400
600
800
1000
1200
1400
1600
Professional
Student
Unemployed

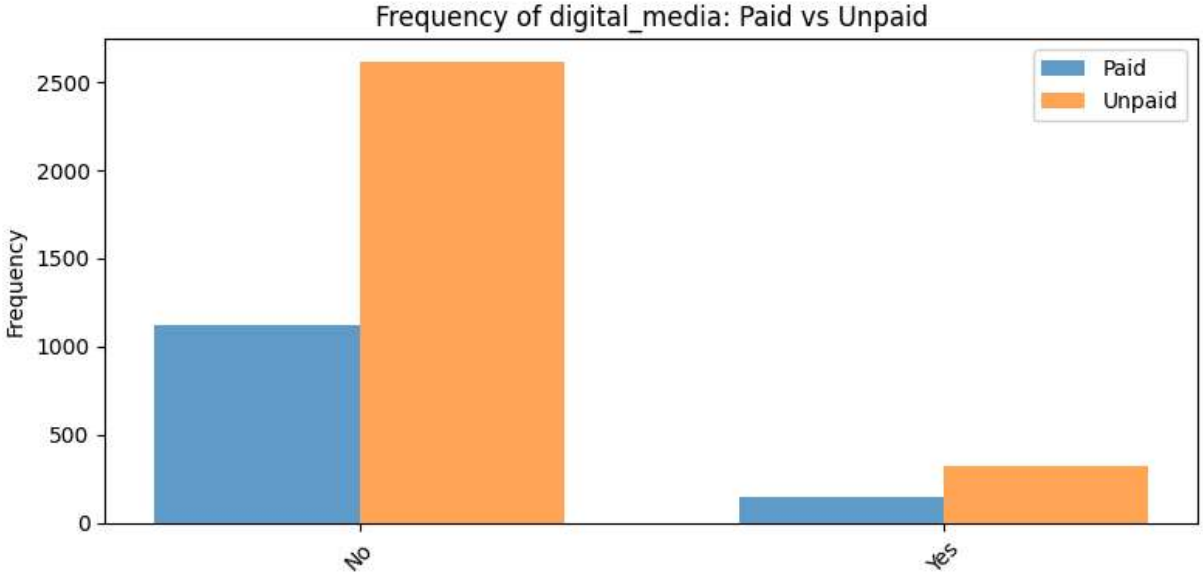

Frequency of first_interaction: Paid vs Unpaid
Paid
Unpaid
Frequency
0
200
400
600
800
1000
1200
1400
1600
Mobile App
Website

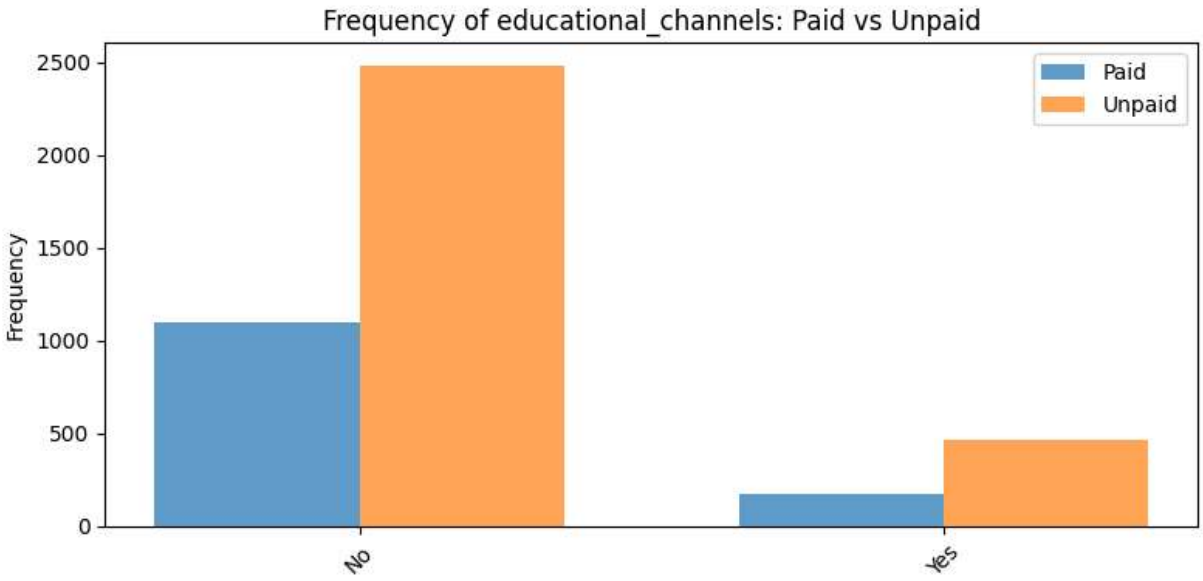

Frequency of profile_completed: Paid vs Unpaid
Paid
Unpaid
Frequency
0
200
400
600
800
1000
1200
1400
1600
High
Low
Medium

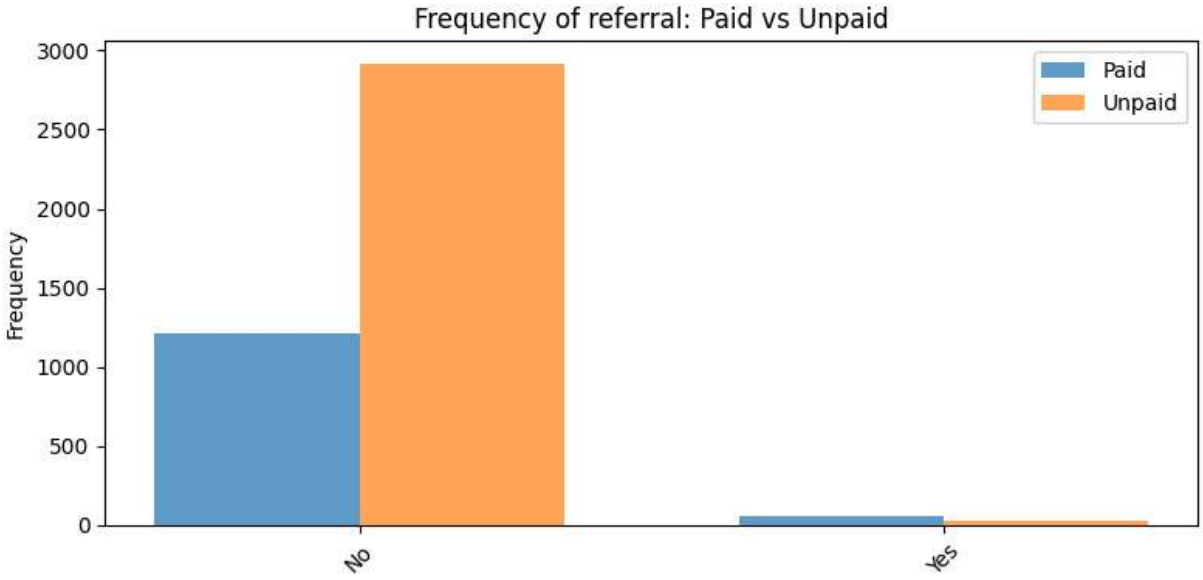

Frequency of last_activity: Paid vs Unpaid
Paid
Unpaid
Frequency
0
200
400
600
800
1000
1200
1400
Email Activity
Phone Activity
Website Activity

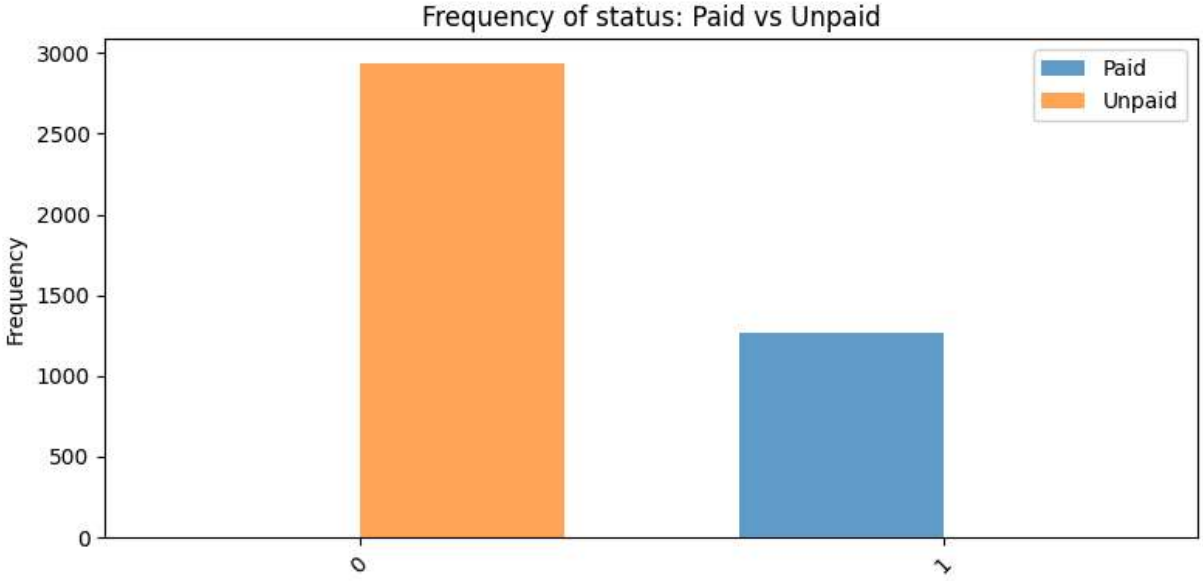

Frequency of print_media_type1: Paid vs Unpaid
Paid
Unpaid
Frequency
2500
2000
1500
1000
500
0
No
Yes


Frequency of print_media_type2: Paid vs Unpaid
Paid
Unpaid
Frequency
2500
2000
1500
1000
500
0
No
Yes


Frequency of digital_media: Paid vs Unpaid
Paid
Unpaid
Frequency
2500
2000
1500
1000
500
0
No
Yes


Frequency of educational_channels: Paid vs Unpaid
Paid
Unpaid
Frequency
2500
2000
1500
1000
500
0
No
Yes


Frequency of referral: Paid vs Unpaid
Paid
Unpaid
Frequency
3000
2500
2000
1500
1000
500
0
No
Yes


Frequency of status: Paid vs Unpaid
Paid
Unpaid
Frequency
3000
2500
2000
1500
1000
500
0
0
1

In [ ]:
```
# 4. Preparing Data for Modeling

X = original_df.drop(["status"], axis=1) # drop the target variable
Y = original_df['status'] # define the target variable

X = pd.get_dummies(X, drop_first=True) # creating dummy variables

# 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 # inputs, output(s)/ thing we want to pred
)
```

Building a Decision Tree model

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?

1. If we predict that a lead will not get converted and the lead would have converted then the company will lose a potential customer.

2. 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 so we care more about maximizing recall, but still don't want a small precision value.

How to reduce the losses?

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

In [ ]:
```
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', 'Co

    plt.ylabel('Actual')

    plt.xlabel('Predicted')
```

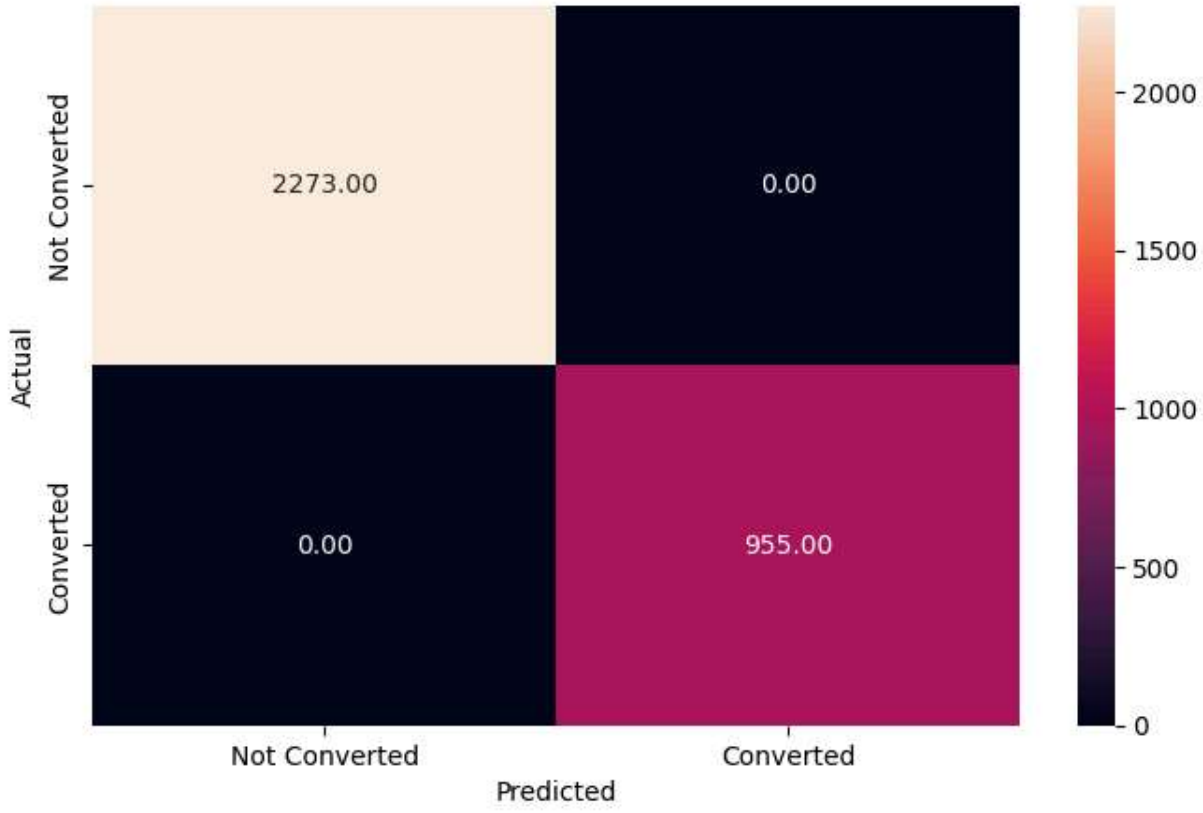
```
    plt.show()
```

In [ ]:
```
# Fit the decision tree classifier on the training data.
d_tree = DecisionTreeClassifier()
d_tree.fit(X_train, y_train)
```

Out[ ]:
```
DecisionTreeClassifier()
```

In [ ]:
```
# Check the performace on the training data (this will later tell us if it is overf
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
```



In [ ]:
```
# Check the performance on the test data.
y_pred_test1 = d_tree.predict(X_test)
metrics_score(y_test, y_pred_test1)
```

	precision	recall	f1-score	support
0	0.86	0.87	0.87	962
1	0.70	0.69	0.70	422
accuracy			0.82	1384
macro avg	0.78	0.78	0.78	1384
weighted avg	0.81	0.82	0.82	1384

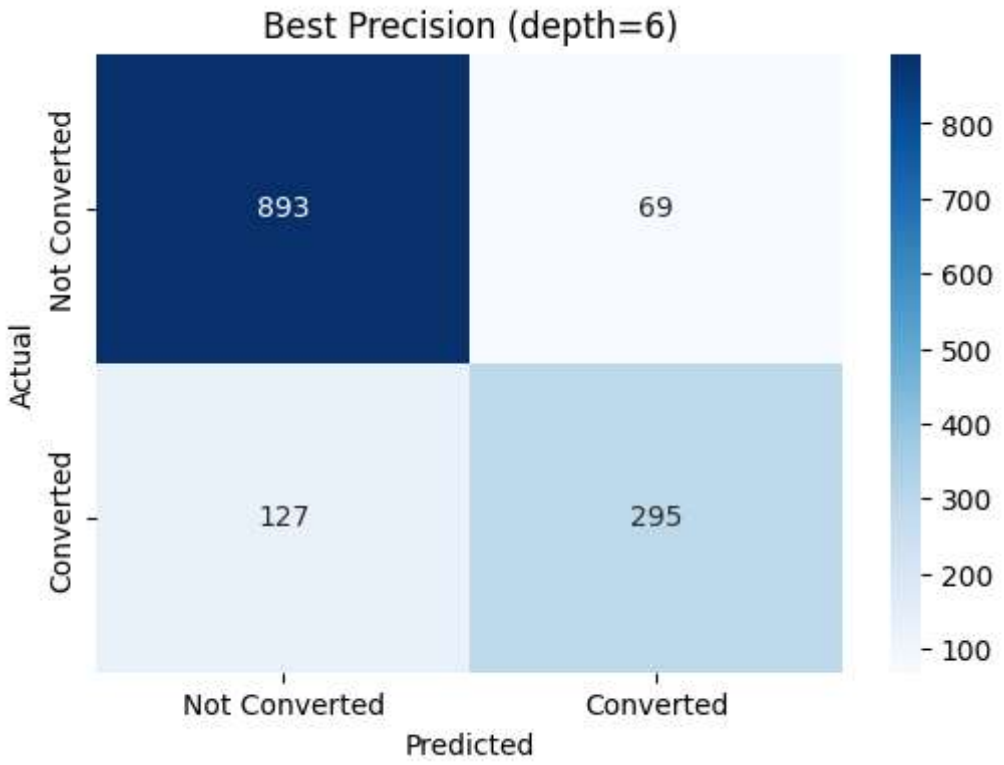
Do we need to prune the tree?

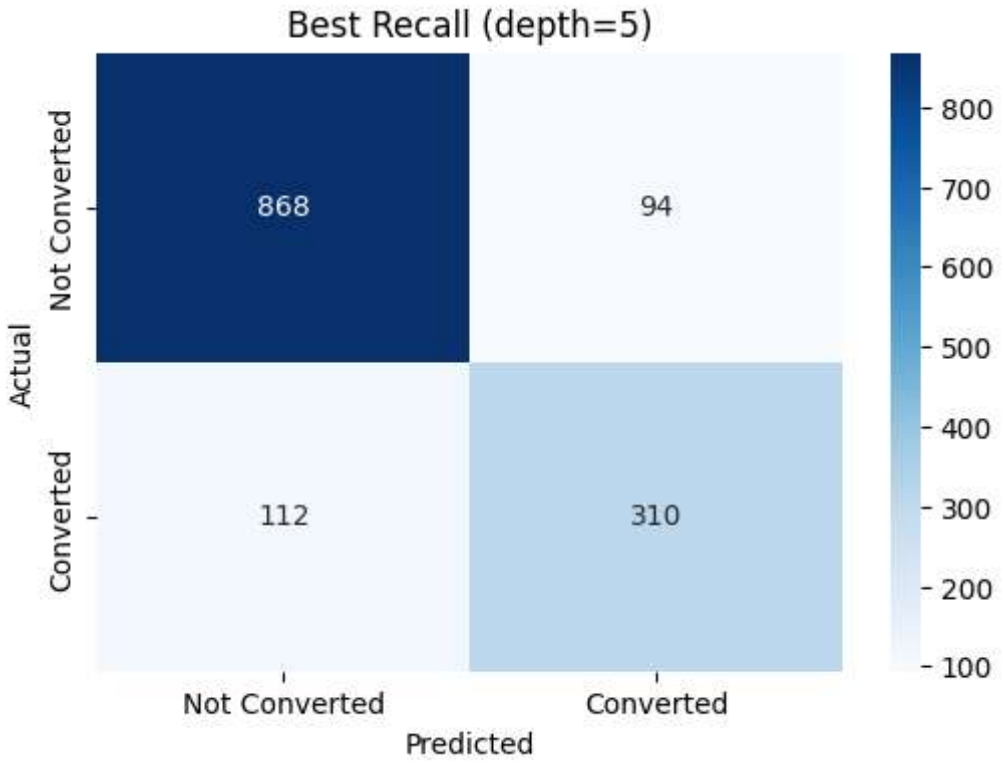
Observations This decision tree is definitely overfitting. Let's use hyperparameter tuning and various tree customizations to find the best dicision tree.

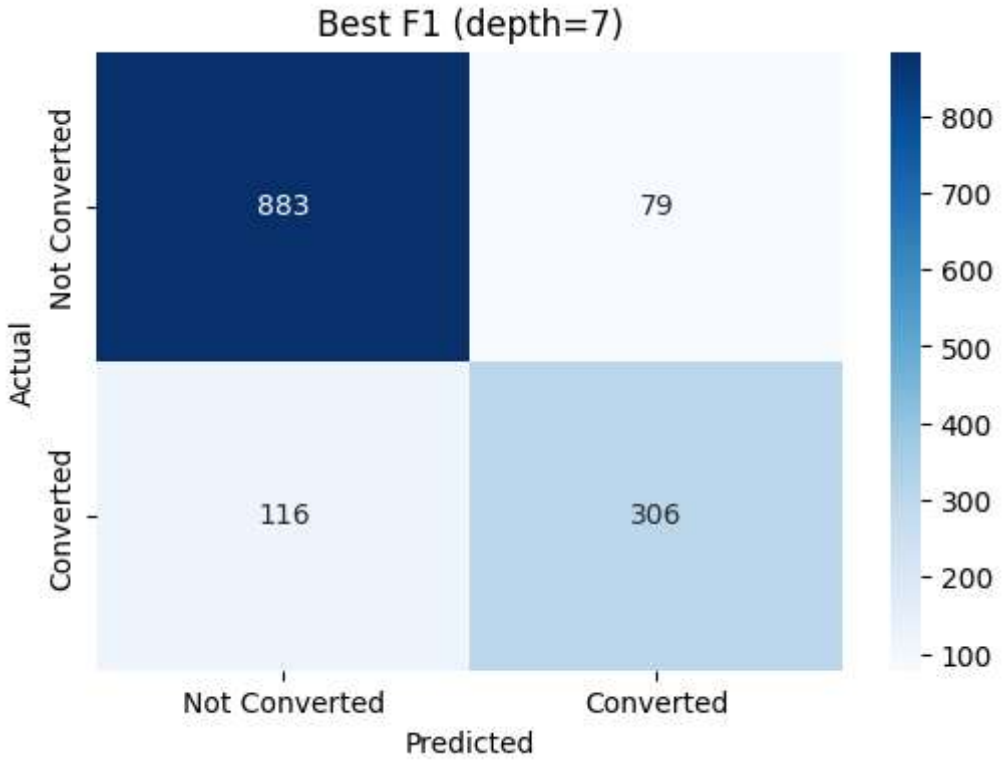
In [ ]:
```
results = []  # we'll keep track of results for each depth here

# try tree depths from 1 through 10
for depth in range(1, 11):
    # build a decision tree with a specific depth
    d_tree = DecisionTreeClassifier(max_depth=depth, random_state=1)
    d_tree.fit(X_train, y_train)  # train it on the training data

    # make predictions on the test set
    test_pred = d_tree.predict(X_test)

    # calculate precision, recall, and F1-score for this depth
    prec = precision_score(y_test, test_pred, pos_label=1)
    rec = recall_score(y_test, test_pred, pos_label=1)
```

```
    f1 = f1_score(y_test, test_pred, pos_label=1)

    # save the metrics, depth, and predictions so we can use them later
    results.append({
        'depth': depth,
        'precision': prec,
        'recall': rec,
        'f1': f1,
        'preds': test_pred
    })

    # print out the scores for quick comparison
    print(f"Depth {depth} -> Precision: {prec:.3f}, Recall: {rec:.3f}, F1: {f1:.3f}

# now that we tested all depths, figure out which one was the best
best_precision = max(results, key=lambda x: x['precision'])  # tree with highest pr
best_recall = max(results, key=lambda x: x['recall'])        # tree with highest re
best_f1 = max(results, key=lambda x: x['f1'])                # tree with best balan

print("\nBest Precision:", best_precision['depth'], "with", best_precision['precisi
print("Best Recall:", best_recall['depth'], "with", best_recall['recall'])
print("Best F1 (balance):", best_f1['depth'], "with", best_f1['f1'])

# loop through each "winner" (precision, recall, f1) and show its confusion matrix
for label, best in [('Precision', best_precision), ('Recall', best_recall), ('F1',
    cm = confusion_matrix(y_test, best['preds'])  # confusion matrix for that model
    plt.figure(figsize=(6,4))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=['Not Converted','Converted'],
                yticklabels=['Not Converted','Converted'])
    plt.title(f"Best {label} (depth={best['depth']})")  # title tells us which metr
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: Und
efinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no predict
ed samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
Depth 1 -> Precision: 0.000, Recall: 0.000, F1: 0.000
Depth 2 -> Precision: 0.719, Recall: 0.571, F1: 0.637
Depth 3 -> Precision: 0.754, Recall: 0.545, F1: 0.633
Depth 4 -> Precision: 0.704, Recall: 0.704, F1: 0.704
Depth 5 -> Precision: 0.767, Recall: 0.735, F1: 0.751
Depth 6 -> Precision: 0.810, Recall: 0.699, F1: 0.751
Depth 7 -> Precision: 0.795, Recall: 0.725, F1: 0.758
Depth 8 -> Precision: 0.791, Recall: 0.709, F1: 0.748
Depth 9 -> Precision: 0.782, Recall: 0.723, F1: 0.751
Depth 10 -> Precision: 0.744, Recall: 0.716, F1: 0.729

Best Precision: 6 with 0.8104395604395604
Best Recall: 5 with 0.7345971563981043
Best F1 (balance): 7 with 0.758364312267658
```


Best Precision (depth=6)
Actual
Not Converted
Converted
893
69
127
295
Not Converted
Converted
Predicted
800
700
600
500
400
300
200
100


Best Recall (depth=5)
Actual
Not Converted
Converted
868
94
112
310
Not Converted
Converted
Predicted
800
700
600
500
400
300
200
100



In [ ]:
```
# Define the grid of parameters to search
param_grid = {
    'max_depth': [2, 4, 6, 8, 10, None],
    'min_samples_split': [2, 5, 10, 20, 25, 30],
    'min_samples_leaf': [1, 2, 4, 10],
    'max_features': [None, 'sqrt', 'log2']
}

# Set up grid search with 5-fold cross validation
grid_search = GridSearchCV(d_tree, param_grid, cv=5, scoring='f1', n_jobs=-1)

# Fit on training data
grid_search.fit(X_train, y_train)

# Best parameters found
print("Best parameters:", grid_search.best_params_)

# Evaluate using the tuned model
best_d_tree = grid_search.best_estimator_
y_pred = best_d_tree.predict(X_test)
metrics_score(y_test, y_pred)

# Hyperparameter tuning allows us to find the combination of parameters from our gr
# If a parameter is the last or first option in the grid there is a possiblity that
```

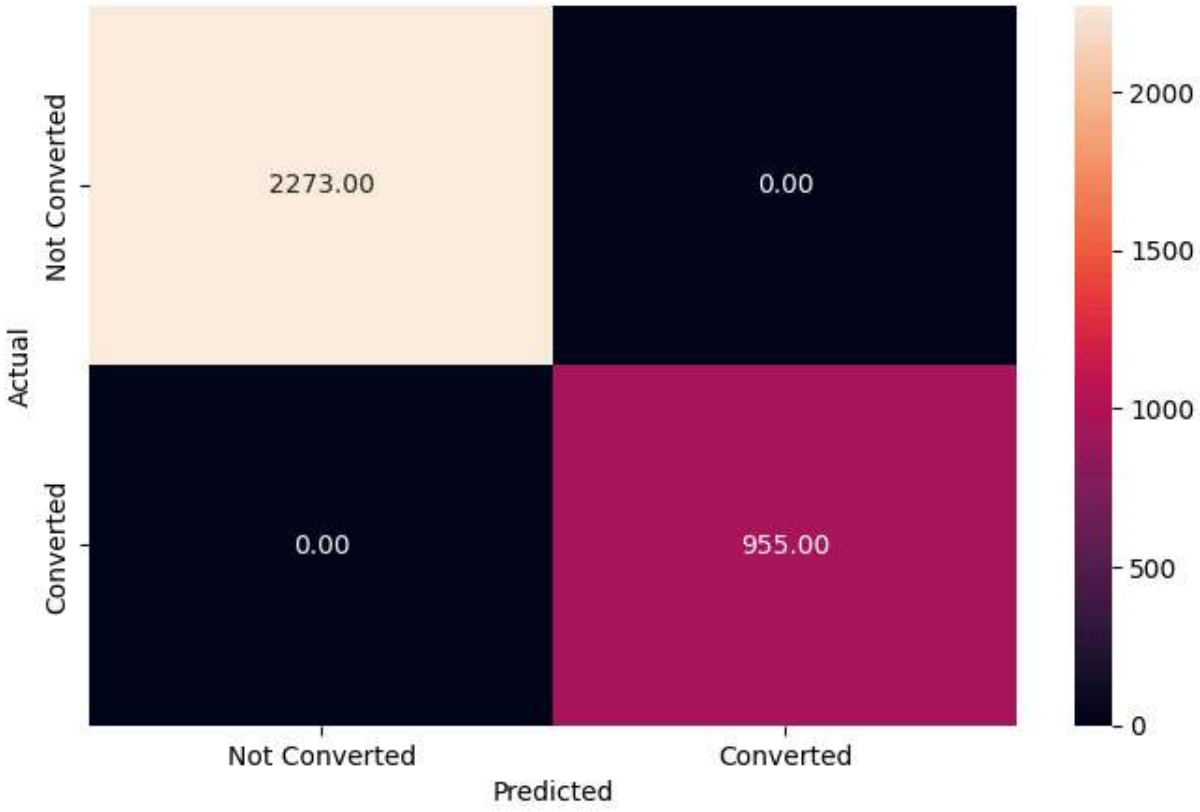
```
Best parameters: {'max_depth': 8, 'max_features': None, 'min_samples_leaf': 1, 'min_
samples_split': 20}
              precision    recall  f1-score   support

           0       0.88      0.92      0.90       962
           1       0.79      0.72      0.75       422

    accuracy                           0.86      1384
   macro avg       0.84      0.82      0.82      1384
weighted avg       0.85      0.86      0.85      1384
```

Building a Random Forest model

In [ ]:
```python
# create a random forest model
rf = RandomForestClassifier(
    n_estimators=200,      # number of trees in the forest
    max_depth=None,        # let trees grow deep (can tune)
    random_state=1,
    n_jobs=-1,             # use all cores for speed
    class_weight='balanced'  # helpful if your data is imbalanced
)

# fit on training data
rf.fit(X_train, y_train)

# make predictions
y_pred_train = rf.predict(X_train)
y_pred_test = rf.predict(X_test)
```

```
# evaluate performance
print("Training performance:")
metrics_score(y_train, y_pred_train)

print("Testing performance:")
metrics_score(y_test, y_pred_test)
```

```
Training performance:
              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
```



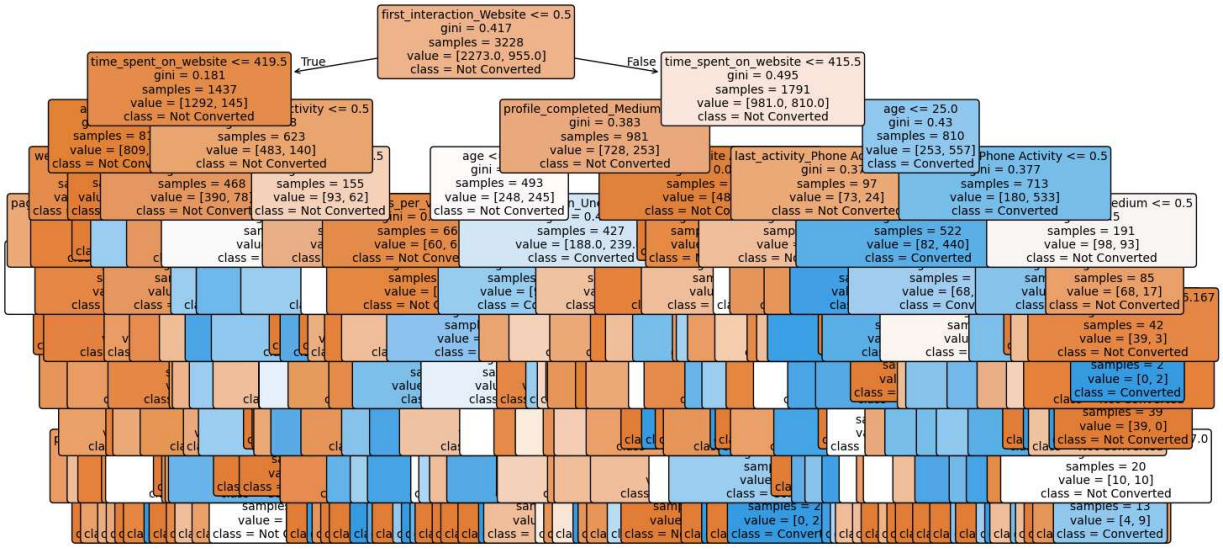

```
Testing performance:
              precision    recall  f1-score   support

           0       0.87      0.93      0.90       962
           1       0.81      0.69      0.75       422

    accuracy                           0.86      1384
   macro avg       0.84      0.81      0.82      1384
weighted avg       0.85      0.86      0.85      1384
```

Do we need to prune the tree?

In [ ]:
```
param_grid = {
    'n_estimators': [100, 200],     # how many trees
    'max_depth': [5, 10, None],      # limit depth
    'min_samples_split': [2, 5],     # minimum samples to split a node
    'min_samples_leaf': [1, 2],       # minimum samples in a leaf
    'max_features': ['sqrt']  # number of features considered at each split
}

rf = RandomForestClassifier(random_state=1, n_jobs=-1)

grid_search = GridSearchCV(
    rf,
    param_grid,
    cv=5,
    scoring='f1',   # optimize for F1 (or recall if you want to minimize false nega
    n_jobs=-1
)

grid_search.fit(X_train, y_train)

print("Best parameters:", grid_search.best_params_)

# evaluate tuned random forest
best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(X_test)
metrics_score(y_test, y_pred)
```

```
Best parameters: {'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'm
in_samples_split': 5, 'n_estimators': 200}
              precision    recall  f1-score   support

           0       0.88      0.93      0.90       962
           1       0.82      0.70      0.76       422

    accuracy                           0.86      1384
   macro avg       0.85      0.82      0.83      1384
weighted avg       0.86      0.86      0.86      1384
```

In [ ]:
```
plt.figure(figsize=(16,8))
plot_tree(
    d_tree,
    feature_names=X_train.columns,
    class_names=['Not Converted','Converted'],
    filled=True,
    rounded=True,
    fontsize=10
)
plt.show()
```

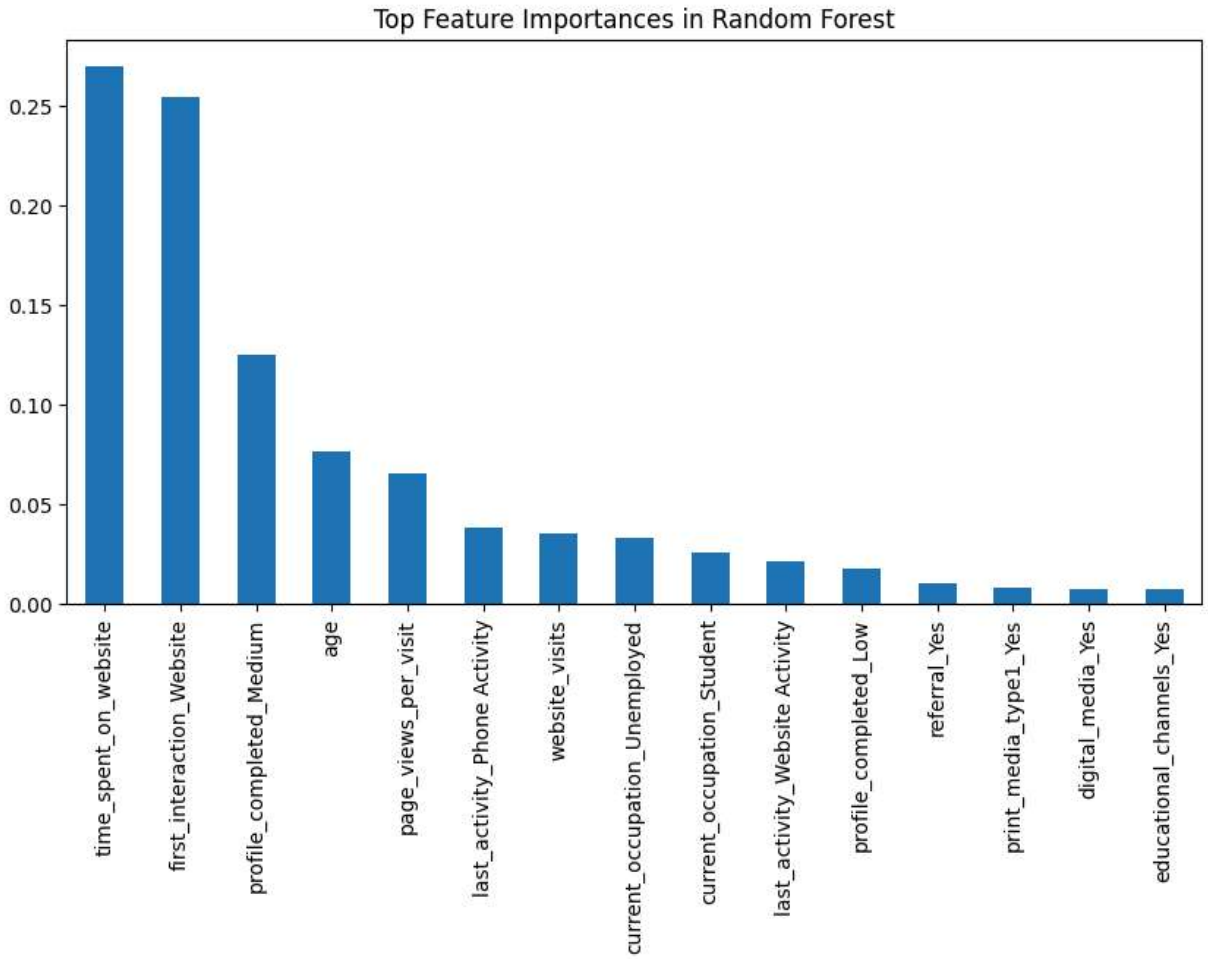


Actionable Insights and Recommendations

In [ ]:
```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[896  66]
 [125 297]]
              precision    recall  f1-score   support

           0       0.88      0.93      0.90       962
           1       0.82      0.70      0.76       422

    accuracy                           0.86      1384
   macro avg       0.85      0.82      0.83      1384
weighted avg       0.86      0.86      0.86      1384
```

In [ ]:
```
importances = best_rf.feature_importances_
feat_importances = pd.Series(importances, index=X_train.columns)

feat_importances.sort_values(ascending=False).head(15).plot(kind='bar', figsize=(10
plt.title("Top Feature Importances in Random Forest")
plt.show()
```



Observations

Factors Driving Lead Conversion:

1. Time spent on website (biggest driver)

- The longer a lead spends exploring, the higher their chance of converting.
- This suggests engaged users are genuinely interested and evaluating offerings in detail.

2. First interaction through website

- Leads who first interacted via the website convert more often compared to other channels.
- The website is likely the strongest acquisition channel.

3. Profile completeness (Medium > Low)

- Leads who provide more information (medium-completed profiles) convert at higher rates.
- This signals stronger intent and trust in the company.

4. Age

- Certain age groups are more likely to convert - probably younger professionals or mid-career learners.
- QUESTION: Why are more people between the age of 50 and 60 interested in the company when they are about to retire?

5. Page views per visit

- Multiple page views during a single visit indicates high curiosity or intent.

6. Last activity through phone or website

- Recent active touchpoints (like phone or browsing) increase conversion likelihood.

7. Occupation (Student, Unemployed)

- Students and unemployed leads appear more conversion-prone, perhaps because they're actively looking for opportunities.
- Professionals are strong prospects because there are more of them interested; however, students and unemployed have a higher lead conversion rate even though there are less of them.

Profile of Leads Likely to Convert:

- High engagement online --> spends a lot of time browsing and views multiple pages.
- First discovered via the website rather than other sources.
- Medium-completed profile --> they cared enough to share info, but not necessarily fully complete.
- 50+ age segment.
- Actively interacting via phone activity or recent website visits.
- Occupation --> professionals, students, and unemployed are all prospects. Perhaps this category needs to be broken down into finer categories for further investigation.