Problem Statement

Business Context

Business communities in the United States are facing high demand for human resources, but one of the constant challenges is identifying and attracting the right talent, which is perhaps the most important element in remaining competitive. Companies in the United States look for hard-working, talented, and qualified individuals both locally as well as abroad.

The Immigration and Nationality Act (INA) of the US permits foreign workers to come to the United States to work on either a temporary or permanent basis. The act also protects US workers against adverse impacts on their wages or working conditions by ensuring US employers' compliance with statutory requirements when they hire foreign workers to fill workforce shortages. The immigration programs are administered by the Office of Foreign Labor Certification (OFLC).

OFLC processes job certification applications for employers seeking to bring foreign workers into the United States and grants certifications in those cases where employers can demonstrate that there are not sufficient US workers available to perform the work at wages that meet or exceed the wage paid for the occupation in the area of intended employment.

Objective

In FY 2016, the OFLC processed 775,979 employer applications for 1,699,957 positions for temporary and permanent labor certifications. This was a nine percent increase in the overall number of processed applications from the previous year. The process of reviewing every case is becoming a tedious task as the number of applicants is increasing every year.

The increasing number of applicants every year calls for a Machine Learning based solution that can help in shortlisting the candidates having higher chances of VISA approval. OFLC has hired the firm EasyVisa for data-driven solutions. You as a data scientist at EasyVisa have to analyze the data provided and, with the help of a classification model:

- Facilitate the process of visa approvals.
- Recommend a suitable profile for the applicants for whom the visa should be certified or denied based on the drivers that significantly influence the case status.

Data Description

The data contains the different attributes of employee and the employer. The detailed data dictionary is given below.

- case_id: ID of each visa application
- continent: Information of continent the employee
- education_of_employee: Information of education of the employee
- has_job_experience: Does the employee has any job experience? Y= Yes; N = No
- requires_job_training: Does the employee require any job training? Y = Yes; N = No
- no_of_employees: Number of employees in the employer's company
- yr_of_estab: Year in which the employer's company was established
- region_of_employment: Information of foreign worker's intended region of employment in the US.
- prevailing_wage: Average wage paid to similarly employed workers in a specific occupation in the area of intended employment. The purpose of the prevailing wage is to ensure that the foreign worker is not underpaid compared to other workers offering the same or similar service in the same area of employment.
- unit_of_wage: Unit of prevailing wage. Values include Hourly, Weekly, Monthly, and Yearly.
- full_time_position: Is the position of work full-time? Y = Full Time Position; N = Part Time Position
- case_status: Flag indicating if the Visa was certified or denied

Installing and Importing the necessary libraries

In [101]:

```
# Installing the libraries with the specified version.
!pip install numpy==1.25.2 pandas==1.5.3 scikit-learn==1.5.2 matplotlib==3.7.1 seaborn==0.13.1 xgboost==2.0.3 -q
--user
```

Note: *After running the above cell, kindly restart the notebook kernel and run all cells sequentially from the below.*

In [102]:

```
# To help with reading and manipulation of data
import numpy as np
import pandas as pd

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

# To split the data
from sklearn.model_selection import train_test_split

# To impute missing values
from sklearn.impute import SimpleImputer

# To build a Random forest classifier
from sklearn.ensemble import RandomForestClassifier

# To tune a model
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

# To get different performance metrics
import sklearn.metrics as metrics
from sklearn.metrics import (
    classification_report,
    confusion_matrix,
    recall_score,
    accuracy_score,
    precision_score,
    f1_score,
)
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score

# Libraries different ensemble classifiers
from sklearn.ensemble import (
    BaggingClassifier,
    RandomForestClassifier,
    AdaBoostClassifier,
    GradientBoostingClassifier
)


from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier

# To undersample and oversample the data
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

# To suppress warnings
import warnings
warnings.filterwarnings("ignore")

# To suppress warnings
import warnings

warnings.filterwarnings("ignore")
```

Import Dataset

In [103]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/
drive", force_remount=True).
```

In [104]:

```
path = '/content/drive/My Drive/python/EasyVisa.csv'
df_orig = pd.read_csv(path)
df = df_orig.copy()
```

Overview of the Dataset

View the first and last 5 rows of the dataset

In [105]:

```
df.head()
```

Out[105]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees	yr_of_estab	region_of_empl
0	EZYV01	Asia	High School	N	N	14513	2007	
1	EZYV02	Asia	Master's	Y	N	2412	2002	N
2	EZYV03	Asia	Bachelor's	N	Y	44444	2008	
3	EZYV04	Asia	Bachelor's	N	N	98	1897	
4	EZYV05	Africa	Master's	Y	N	1082	2005	

In [106]:

```
df.tail()
```

Out[106]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees	yr_of_estab	region_
25475	EZYV25476	Asia	Bachelor's	Y	Y	2601	2008	
25476	EZYV25477	Asia	High School	Y	N	3274	2006	
25477	EZYV25478	Asia	Master's	Y	N	1121	1910	
25478	EZYV25479	Asia	Master's	Y	Y	1918	1887	
25479	EZYV25480	Asia	Bachelor's	Y	N	3195	1960	

Understand the shape of the dataset

In [107]:

```
df.shape
```

Out[107]:

```
(25480, 12)
```

Check the data types of the columns for the dataset

In [108]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25480 entries, 0 to 25479
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   case_id                25480 non-null  object 
 1   continent              25480 non-null  object 
 2   education_of_employee  25480 non-null  object 
 3   has_job_experience     25480 non-null  object 
 4   requires_job_training  25480 non-null  object 
 5   no_of_employees        25480 non-null  int64  
 6   yr_of_estab            25480 non-null  int64  
 7   region_of_employment   25480 non-null  object 
 8   prevailing_wage        25480 non-null  float64
 9   unit_of_wage           25480 non-null  object 
 10  full_time_position     25480 non-null  object 
 11  case_status            25480 non-null  object 
dtypes: float64(1), int64(2), object(9)
memory usage: 2.3+ MB
```

Exploratory Data Analysis (EDA)

Let's check the statistical summary of the data

In [109]:

```
summary = df.describe(include="all");
print(summary)
```

```
          case_id continent education_of_employee has_job_experience  \
count       25480     25480                 25480               25480
unique      25480         6                     4                   2
top     EZYV25480      Asia           Bachelor's                   Y
freq            1     16861                 10234               14802
mean          NaN       NaN                   NaN                 NaN
std           NaN       NaN                   NaN                 NaN
min           NaN       NaN                   NaN                 NaN
25%           NaN       NaN                   NaN                 NaN
50%           NaN       NaN                   NaN                 NaN
75%           NaN       NaN                   NaN                 NaN
max           NaN       NaN                   NaN                 NaN

       requires_job_training  no_of_employees   yr_of_estab  \
count                  25480     25480.000000  25480.000000
unique                     2              NaN           NaN
top                        N              NaN           NaN
freq                   22525              NaN           NaN
mean                     NaN      5667.043210   1979.409929
std                      NaN     22877.928848     42.366929
min                      NaN       -26.000000   1800.000000
25%                      NaN      1022.000000   1976.000000
50%                      NaN      2109.000000   1997.000000
75%                      NaN      3504.000000   2005.000000
max                      NaN    602069.000000   2016.000000

       region_of_employment  prevailing_wage unit_of_wage full_time_position  \
count                 25480     25480.000000        25480               25480
unique                    5              NaN            4                   2
top               Northeast              NaN         Year                   Y
freq                   7195              NaN        22962               22773
mean                    NaN     74455.814592          NaN                 NaN
std                     NaN     52815.942327          NaN                 NaN
min                     NaN         2.136700          NaN                 NaN
25%                     NaN     34015.480000          NaN                 NaN
50%                     NaN     70308.210000          NaN                 NaN
75%                     NaN    107735.512500          NaN                 NaN
max                     NaN    319210.270000          NaN                 NaN

       case_status
count        25480
unique           2
top      Certified
freq         17018
mean           NaN
std            NaN
min            NaN
25%            NaN
50%            NaN
75%            NaN
max            NaN
```

Fixing the negative values in number of employees columns

In [110]:

```
negative_values = df['no_of_employees'] < 0
negative_rows = df[negative_values]
df['no_of_employees'] = df['no_of_employees'].apply(lambda x: max(x, 0))
print(df['no_of_employees'] < 0)
```

```
0        False
1        False
2        False
3        False
4        False
         ...
25475    False
25476    False
25477    False
25478    False
25479    False
Name: no_of_employees, Length: 25480, dtype: bool
```

Let's check the count of each unique category in each of the categorical variables

In [111]:

```
# Count unique categories for each categorical variable
category_counts = {col: df[col].value_counts() for col in df.select_dtypes(include='object').columns}

# Display the counts
for col, counts in category_counts.items():
    print(f"Counts for {col}:\n{counts}\n")
```

```
Counts for case_id:
case_id
EZYV25480    1
EZYV01       1
EZYV02       1
EZYV03       1
EZYV04       1
            ..
EZYV13       1
EZYV12       1
EZYV11       1
EZYV10       1
EZYV09       1
Name: count, Length: 25480, dtype: int64

Counts for continent:
continent
Asia             16861
Europe            3732
North America     3292
South America      852
Africa             551
Oceania            192
Name: count, dtype: int64

Counts for education_of_employee:
education_of_employee
Bachelor's     10234
Master's        9634
High School     3420
Doctorate       2192
Name: count, dtype: int64

Counts for has_job_experience:
has_job_experience
Y    14802
N    10678
Name: count, dtype: int64

Counts for requires_job_training:
requires_job_training
N    22525
Y     2955
Name: count, dtype: int64

Counts for region_of_employment:
region_of_employment
Northeast    7195
South        7017
West         6586
Midwest      4307
Island        375
Name: count, dtype: int64

Counts for unit_of_wage:
unit_of_wage
Year     22962
Hour      2157
Week       272
Month       89
Name: count, dtype: int64

Counts for full_time_position:
full_time_position
Y    22773
N     2707
Name: count, dtype: int64

Counts for case_status:
case_status
Certified    17018
Denied        8462
Name: count, dtype: int64
```

Univariate Analysis

In [112]:

```
def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (15,10))
    kde: whether to show the 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 triangle will indicate the mean value of the column
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins
    ) 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
```

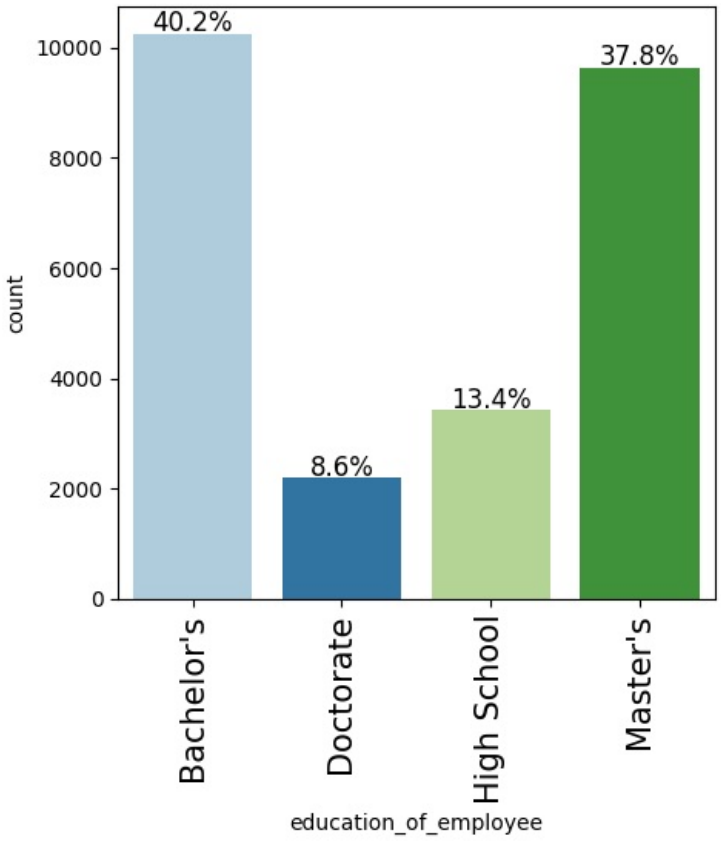
In [113]:

```
# 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 levels)
    """

    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 education of employee

In [114]:

```
# histogram_boxplot(df, 'education_of_employee', figsize=(15, 10), kde=False, bins=None)
labeled_barplot(df, 'education_of_employee', perc=True, n=None)
```

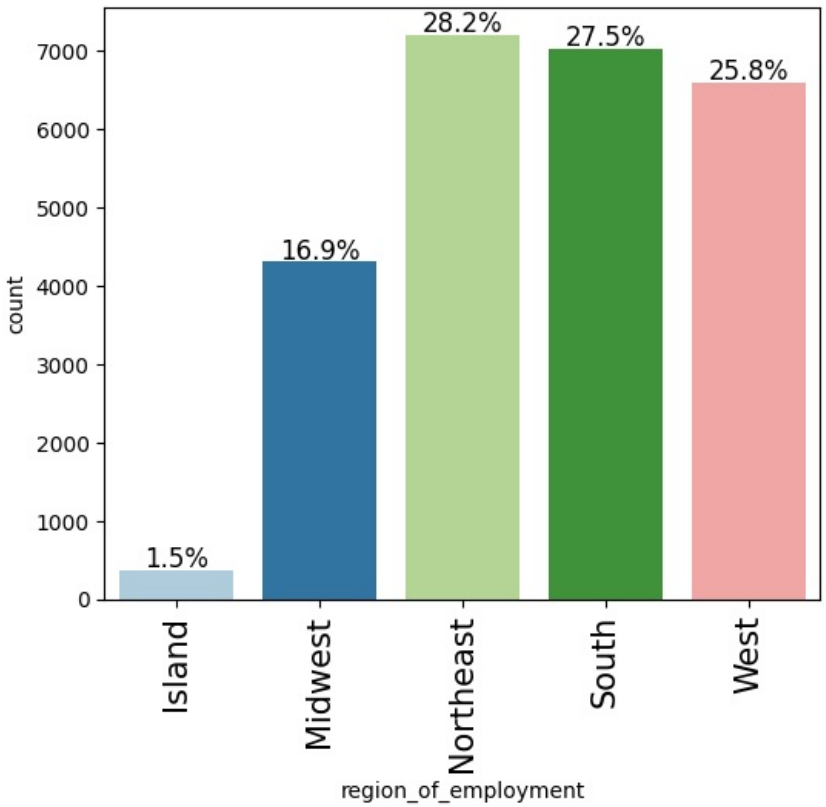


Obsevation: The vast majority of the employees have a bachelors degree (40.16%), followed by Masters (31.81%), the high school and doctorate contribyte to 13.4 and 8.6% respectively.

Observations on region of employment

In [115]:

```
labeled_barplot(df, 'region_of_employment', perc=True, n=None)
```

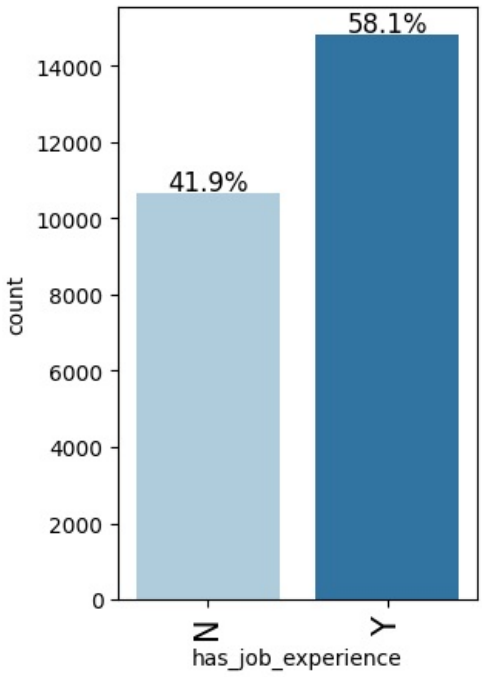


Observations: The region of employment is even for the most. 28.2% from the north east and 27.5 % from the South. 25.8% from the west

Observations on job experience

In [116]:

```
labeled_barplot(df, 'has_job_experience', perc=True, n=None)
```

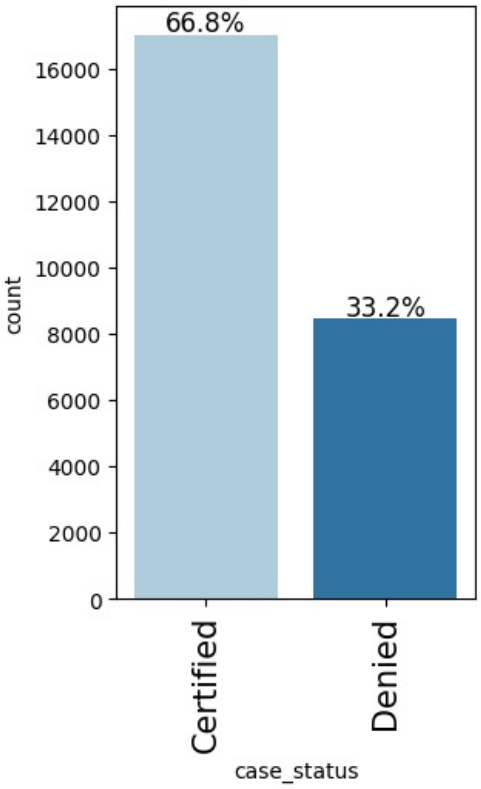


Observation: 58.1 % have experience whereas 41.9% do NOT have experience.

Observations on case status

In [117]:

```
labeled_barplot(df, 'case_status', perc=True, n=None)
```



Observation: 66.8% have been approved, the rest were not approved.

Bivariate Analysis

Creating functions that will help us with further analysis.

In [118]:

```
### 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[0]))
    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[1]))
    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_rainbow")

    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 [119]:

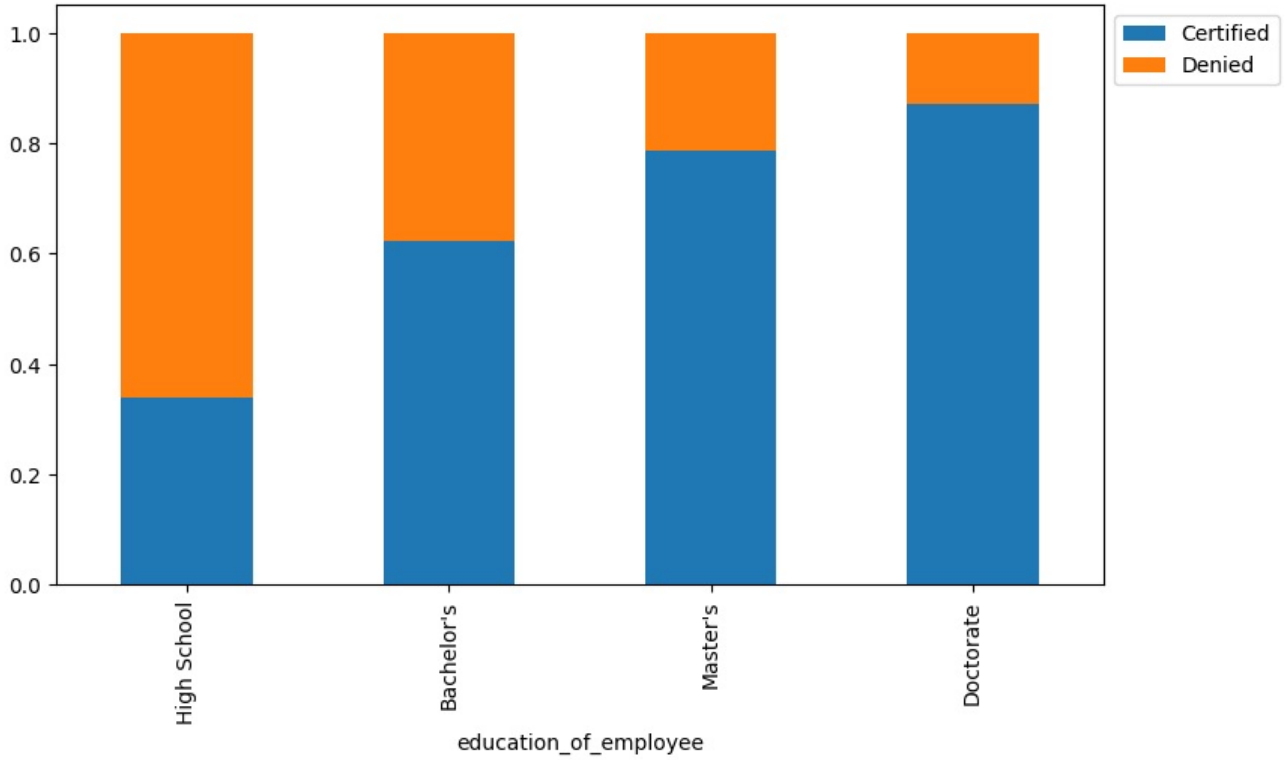
```
def stacked_barplot(data, predictor, target):
    """
    Print the category counts and plot a stacked bar chart

    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_values(
        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()
```

Does higher education increase the chances of visa certification for well-paid jobs abroad?

In [120]:

```
stacked_barplot(df, 'education_of_employee', 'case_status')
```

```
case_status            Certified  Denied    All
education_of_employee
All                        17018    8462  25480
Bachelor's                  6367    3867  10234
High School                 1164    2256   3420
Master's                    7575    2059   9634
Doctorate                   1912     280   2192
------------------------------------------------------------------------------------------------------------------------
--------------------
```

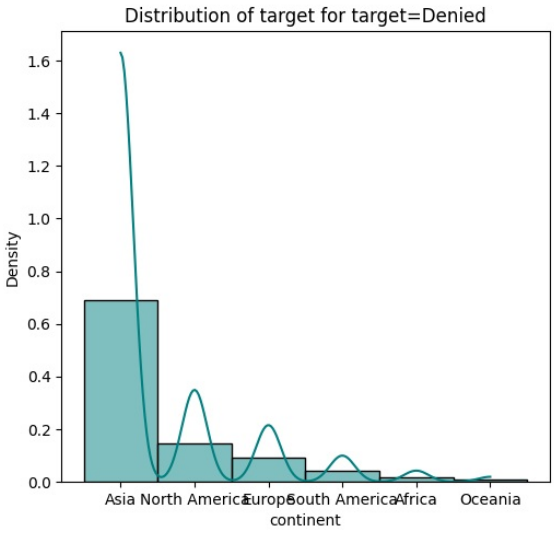


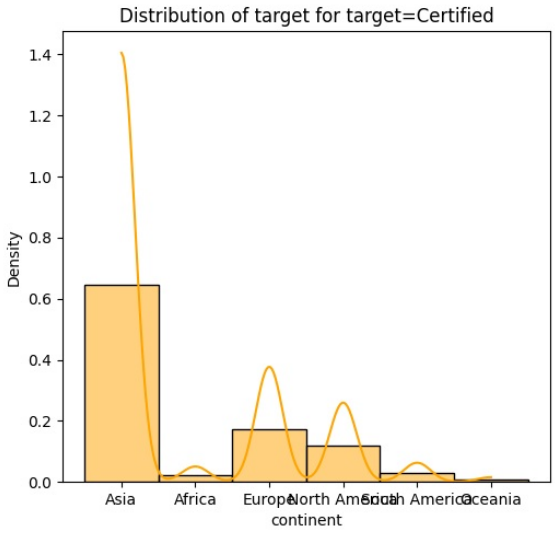
Observation: Yes, the higher the education better are the changes of acceptance of visa. As one can see above, the doctorate's (indicated by blue) have higher acceptance rates, followed by Masters, then bachelors and then high school.

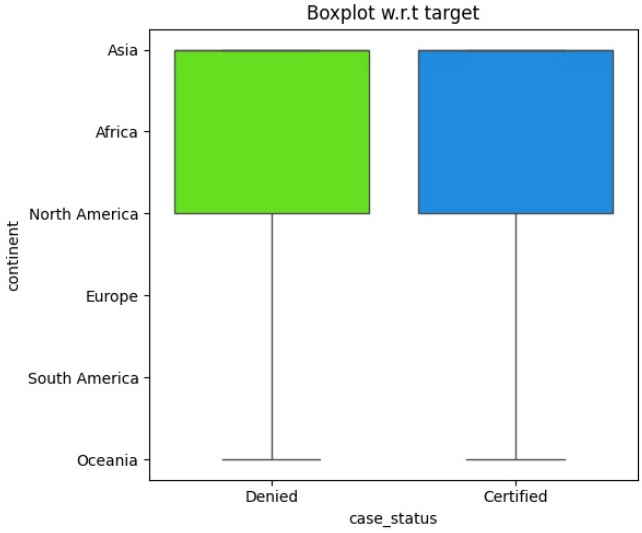
How does visa status vary across different continents?

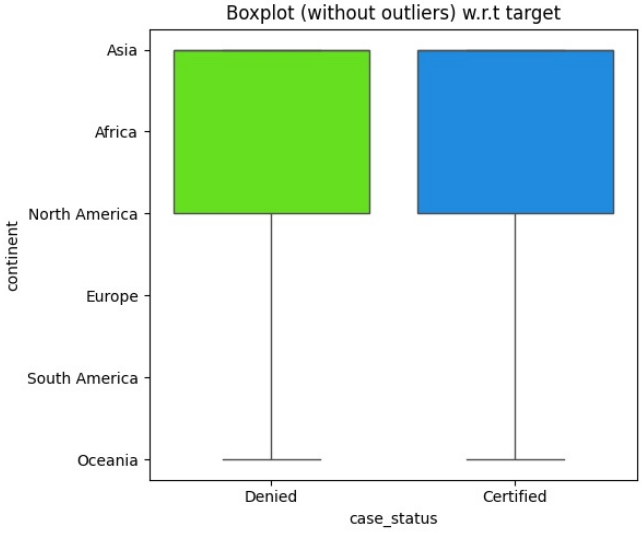
In [121]:

```
distribution_plot_wrt_target(df, 'continent', 'case_status')
```





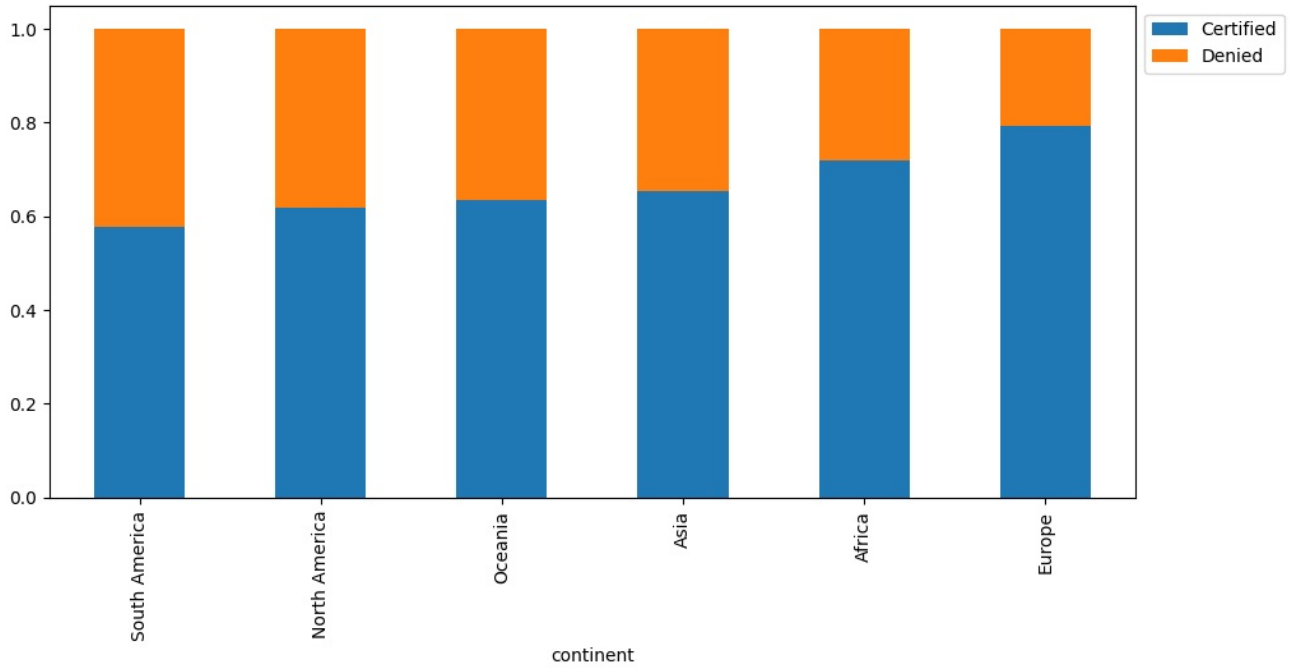




In [122]:

```
stacked_barplot(df, 'continent', 'case_status')
```

```
case_status    Certified  Denied    All
continent
All                17018    8462  25480
Asia               11012    5849  16861
North America       2037    1255   3292
Europe              2957     775   3732
South America        493     359    852
Africa               397     154    551
Oceania              122      70    192
-------------------------------------------------------------------------------------------------------
--------------------
```



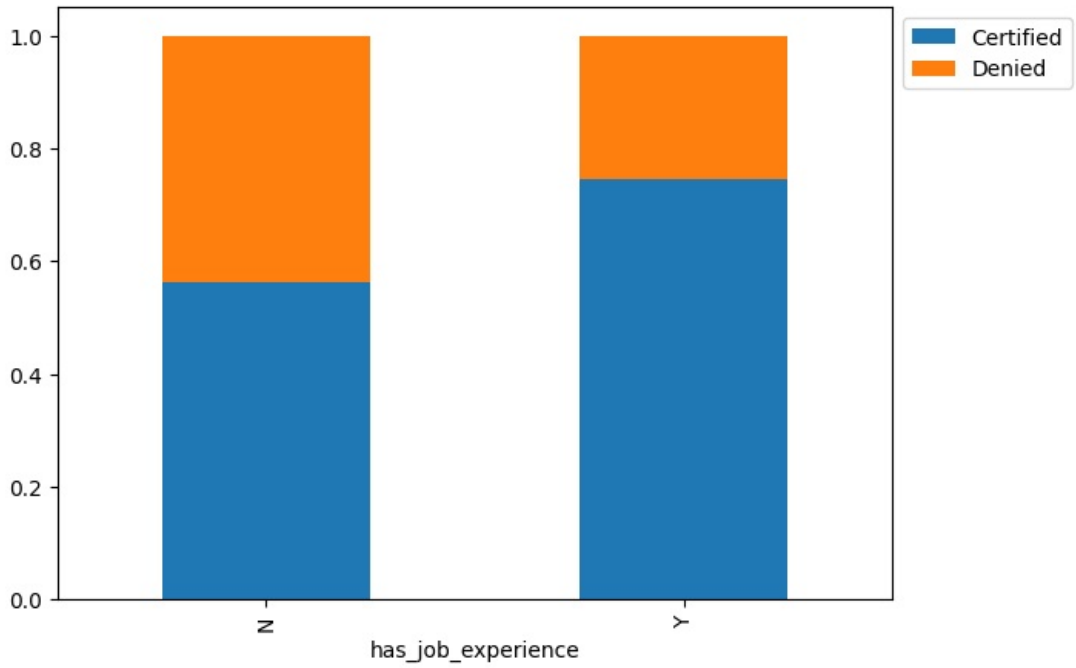
Observation: The % of approved applications was highest in Europe, followed by Africe, Asia, Ocenia and the lowest % is from South America.

Does having prior work experience influence the chances of visa certification for career opportunities abroad?

In [123]:

```
stacked_barplot(df, 'has_job_experience', 'case_status')
```

```
case_status         Certified  Denied    All
has_job_experience
All                     17018    8462  25480
N                        5994    4684  10678
Y                       11024    3778  14802
------------------------------------------------------------------------------------------------------------------------
--------------------
```

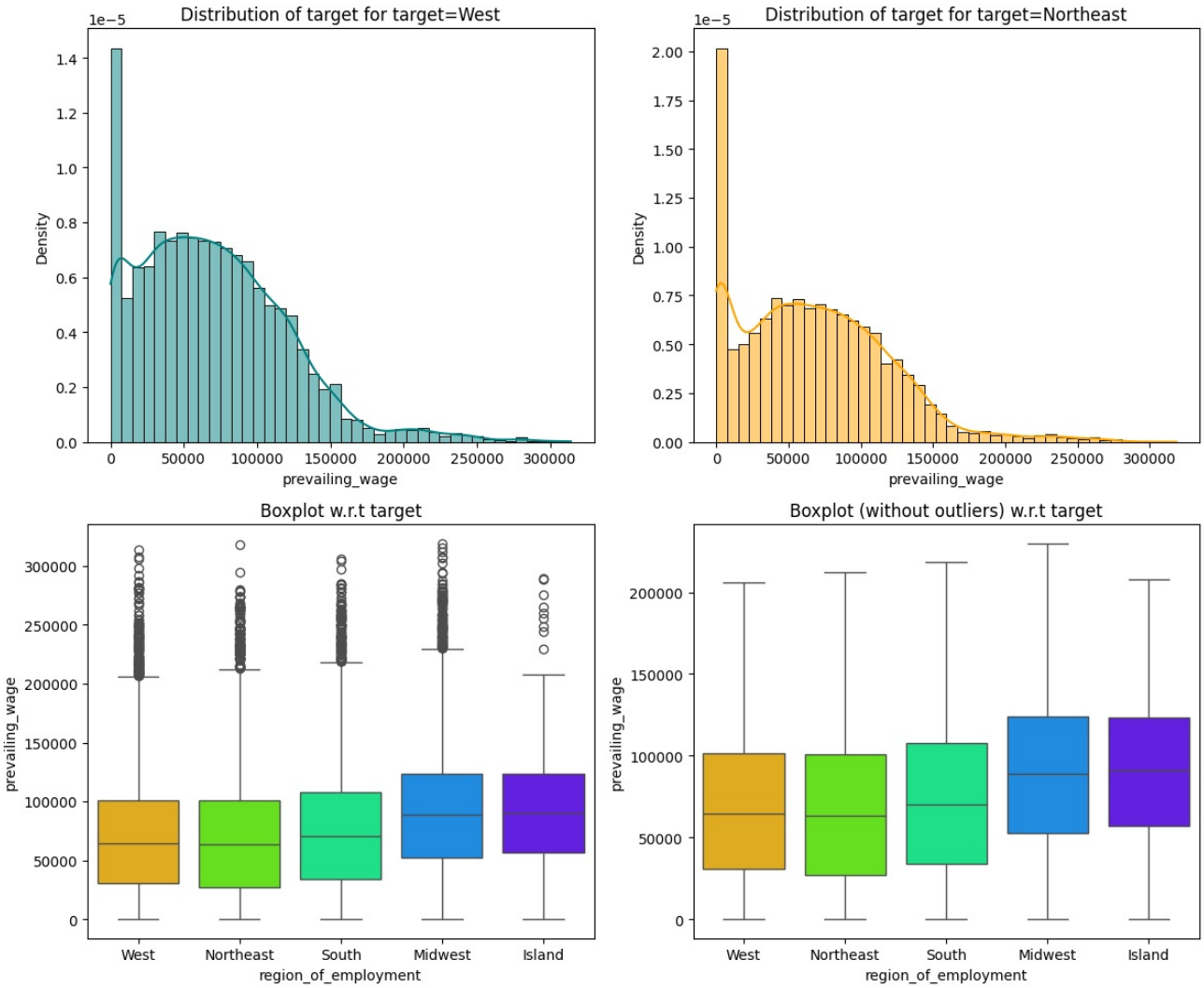


Observation: Yes, have prior job experience increases the chances of getting an approval.

Is the prevailing wage consistent across all regions of the US?

In [124]:

```
distribution_plot_wrt_target(df, 'prevailing_wage', 'region_of_employment')
```

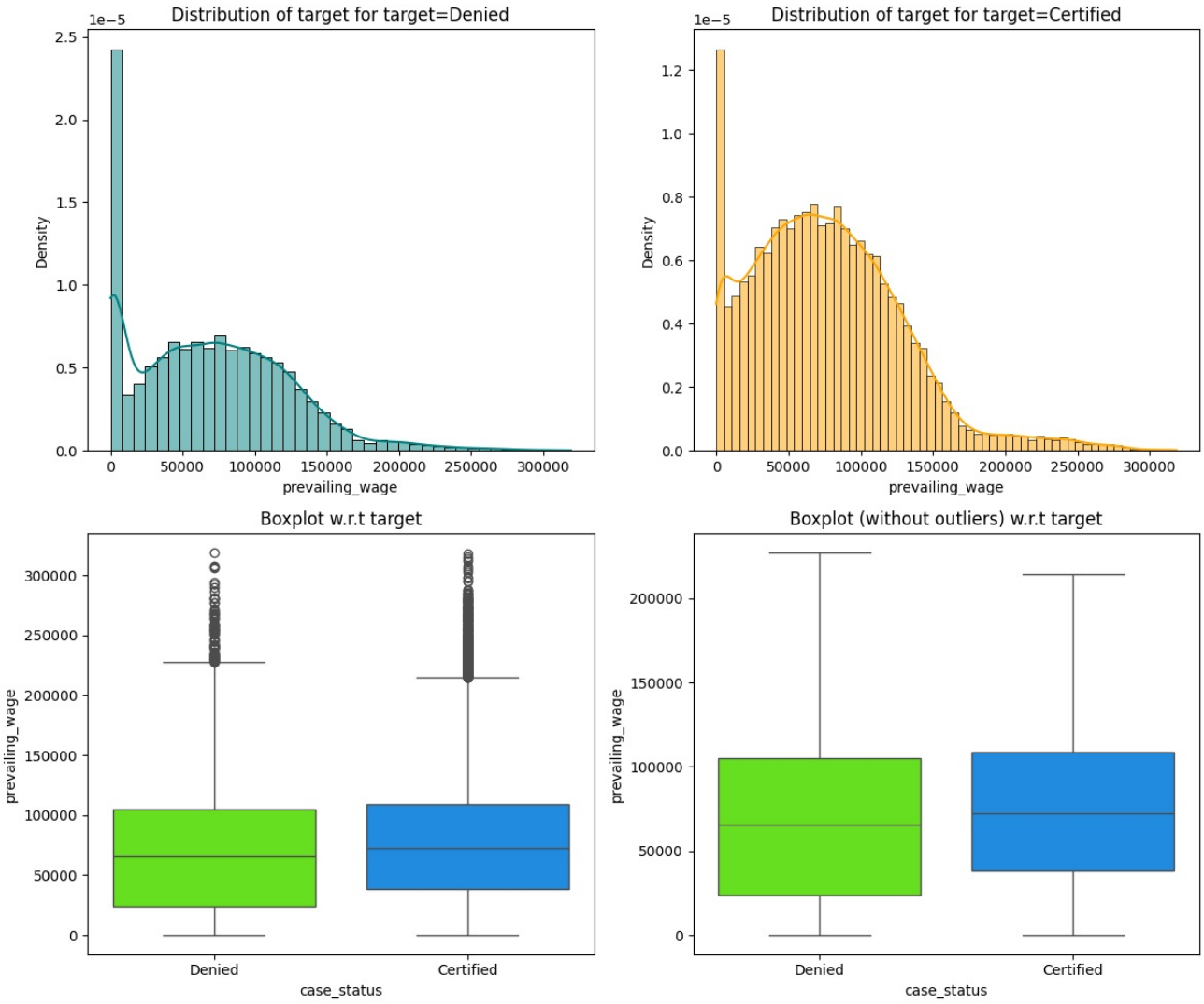


Observation: The average prevailing wage in Midewest and Island are higher. So its not the same.

Does visa status vary with changes in the prevailing wage set to protect both local talent and foreign workers?

In [125]:

```
distribution_plot_wrt_target(df, 'prevailing_wage', 'case_status')
```



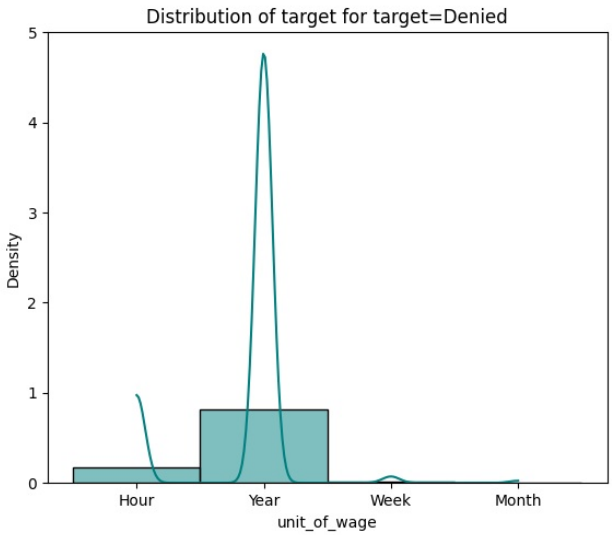
Observation: The prevailaing wage does NOT seem to be a big factor between certified and denied.

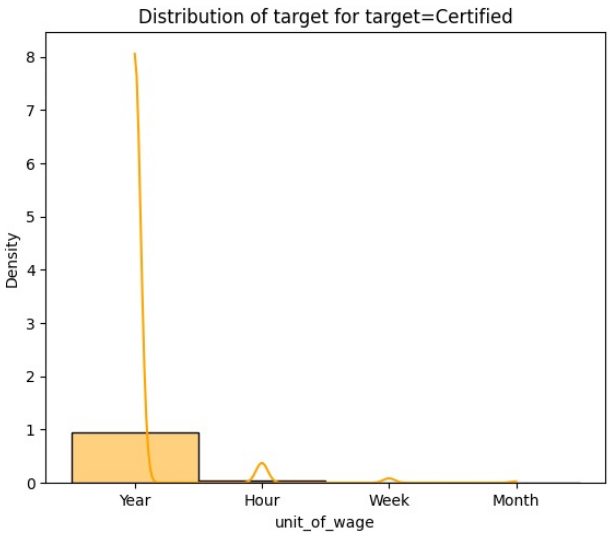
Although the average prevailing wage is slightly higher for Certified cases. Its not very far apart to mention of a distinction, visually

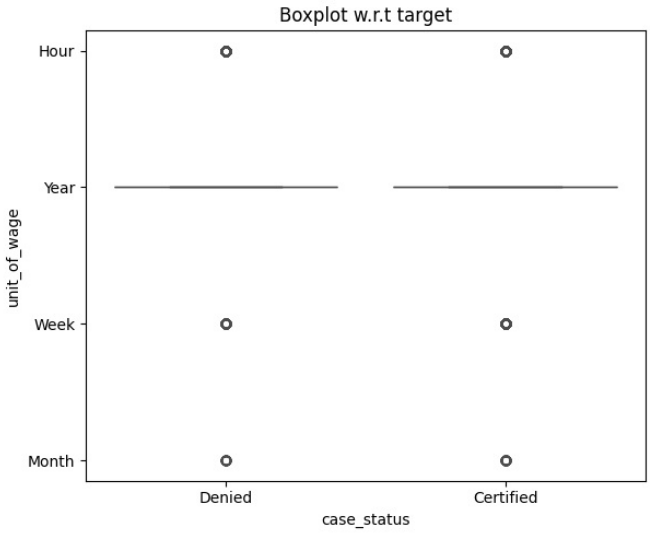
Does the unit of prevailing wage (Hourly, Weekly, etc.) have any impact on the likelihood of visa application certification?

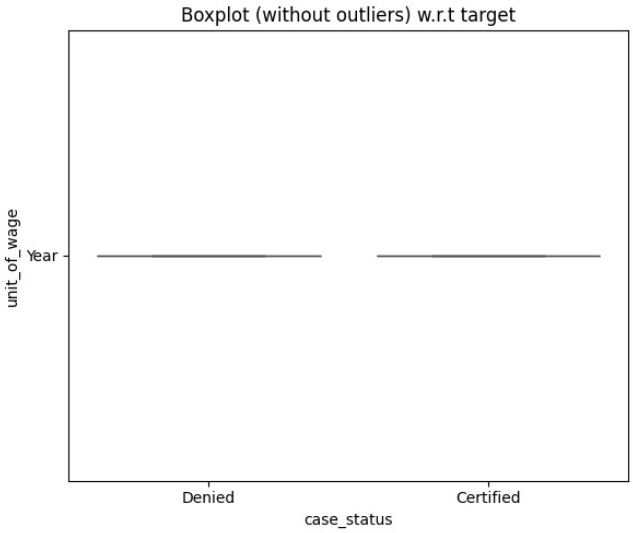
In [126]:

```
distribution_plot_wrt_target(df, 'unit_of_wage', 'case_status')

stacked_barplot(df, 'unit_of_wage', 'case_status')
```

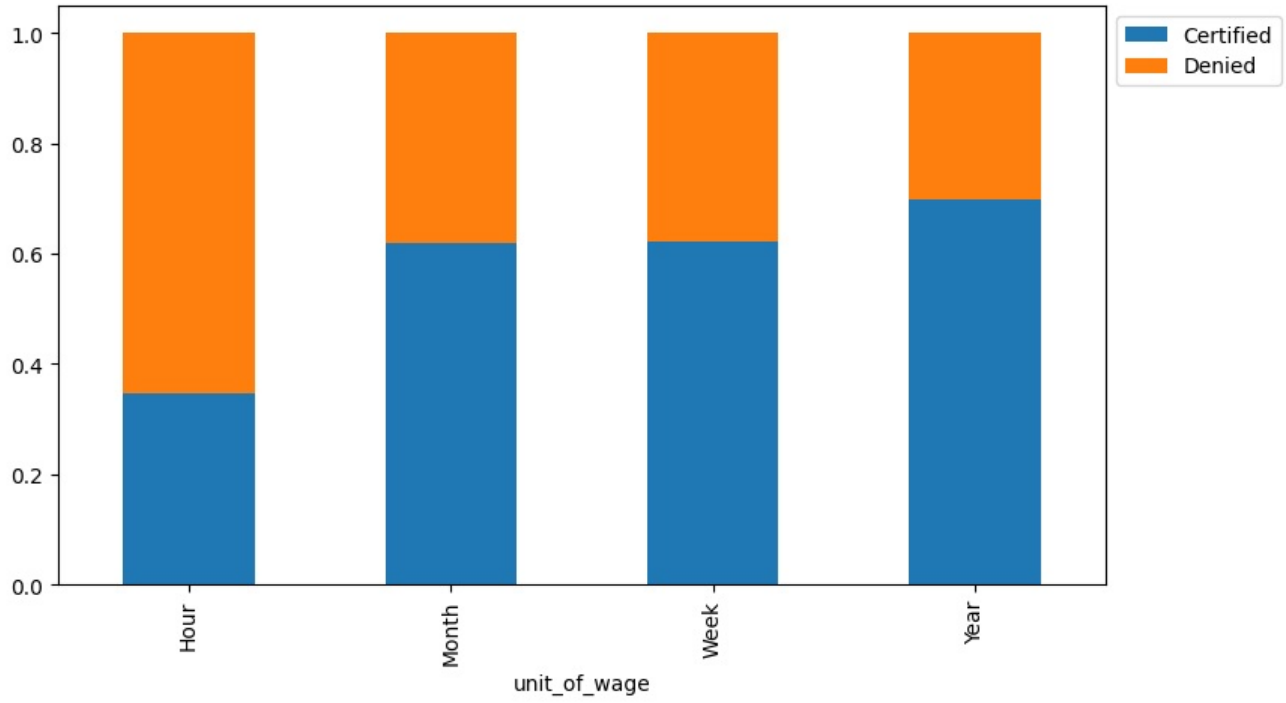








case_status	Certified	Denied	All
unit_of_wage			
All	17018	8462	25480
Year	16047	6915	22962
Hour	747	1410	2157
Week	169	103	272
Month	55	34	89



Observation: yes, the unit of wage seems to a factor in the case status. The amount of denial for yearly wagers is lesser.

Data Pre-processing

Outlier Check

In [127]:

```
# Calculate Q1, Q3, and IQR for prevailing wage
Q1 = df['prevailing_wage'].quantile(0.25)
Q3 = df['prevailing_wage'].quantile(0.75)
IQR = Q3 - Q1

# Define outlier thresholds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers_wage = df[(df['prevailing_wage'] < lower_bound) | (df['prevailing_wage'] > upper_bound)]
print("outlier counts for wage" , outliers_wage.value_counts())
print(outliers_wage)
```

```
outlier counts for wage case_id    continent      education_of_employee  has_job_experience  require
s_job_training  no_of_employees  yr_of_estab  region_of_employment  prevailing_wage  unit_of_wage  f
ull_time_position  case_status
EZYV9898   Asia           High School            Y                    N                    18
2001         West                   272311.21        Year          Y                   Denied
1
EZYV10151  Asia           Bachelor's             Y                    N                    3525
1970         Island                 289878.68        Year          Y                   Denied
1
EZYV10181  Asia           Doctorate              N                    N                    2705
1996         South                  227441.23        Year          N                   Certified
1
EZYV10331  North America  Master's               Y                    N                    2535
1975         Northeast              245197.29        Year          N                   Certified
1
EZYV1039   Asia           Master's               Y                    N                    538
2011         Midwest                222628.84        Year          Y                   Certified
1

..
EZYV1083   Asia           Doctorate              Y                    N                    6024
1989         Northeast              274019.43        Year          Y                   Certified
1
EZYV10821  Africa         Bachelor's             N                    N                    2651
2006         West                   268713.50        Year          Y                   Certified
1
EZYV10811  Asia           Bachelor's             Y                    N                    92780
2012         West                   250163.10        Year          Y                   Certified
1
EZYV10706  Asia           High School            N                    Y                    1856
2003         West                   228715.49        Year          Y                   Denied
1
EZYV1069   Africa         Master's               N                    N                    1486
1992         Northeast              233641.72        Year          Y                   Denied
1
Name: count, Length: 427, dtype: int64
         case_id      continent education_of_employee has_job_experience  \
14        EZYV15           Asia              Master's                   Y
34        EZYV35           Asia              Master's                   N
130      EZYV131  South America           High School                   N
216      EZYV217           Asia              Master's                   Y
221      EZYV222  North America             Doctorate                   Y
...          ...            ...                   ...                 ...
25191  EZYV25192           Asia              Master's                   N
25195  EZYV25196  North America              Master's                   Y
25468  EZYV25469           Asia             Bachelor's                  N
25469  EZYV25470  North America              Master's                   Y
25476  EZYV25477           Asia           High School                   Y

      requires_job_training  no_of_employees  yr_of_estab  \
14                        Y            15756         2006
34                        N             1809         2010
130                       N             2554         2005
216                       N             1515         2001
221                       Y             2518         2010
...                     ...              ...          ...
25191                     N             4983         2005
25195                     N               47         2001
25468                     N              373         2005
25469                     N             2261         1997
```

```
25476                    N            3274        2006

      region_of_employment  prevailing_wage unit_of_wage full_time_position  \
14                   South        220081.73         Year                  Y
34                   South        225569.73         Year                  N
130                Midwest        247393.01         Year                  Y
216                Midwest        269321.68         Year                  N
221                  South        219529.62         Year                  Y
...                    ...              ...          ...                ...
25191              Midwest        280482.51         Year                  Y
25195                South        234308.77         Year                  N
25468              Midwest        272715.74         Year                  N
25469            Northeast        273772.47         Year                  N
25476            Northeast        279174.79         Year                  Y

      case_status
14      Certified
34      Certified
130     Certified
216     Certified
221     Certified
...           ...
25191      Denied
25195   Certified
25468   Certified
25469   Certified
25476   Certified

[427 rows x 12 columns]
```

Data Preparation for modeling

In [128]:

```
# Lets prepare the data.
df["case_status"] = df["case_status"].apply(lambda x: 1 if x == "Certified" else 0)
X = df.drop(["case_id"], inplace=True, axis=1)
X = df.drop(["case_status"], axis=1)

y = df["case_status"]

# Apply one-hot encoding to categorical features
categorical_cols = X.select_dtypes(include='object').columns
X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)
print(X.shape)
```

```
(25480, 21)
```

In [129]:

```
# Splitting data into training, validation and test set:

# first we split data into 2 parts, say temporary and test
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# then we split the temporary set into train and validation
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp)

print(X_train.shape, X_val.shape, X_test.shape)
```

```
(17836, 21) (3822, 21) (3822, 21)
```

Model Building

Model Evaluation Criterion

- Choose the primary metric to evaluate the model on
- Elaborate on the rationale behind choosing the metric

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_sklearn` function will be used to check the model performance of models.
- The `confusion_matrix_sklearn` function will be used to plot the confusion matrix.

In [130]:

```
# defining a function to compute different metrics to check performance of a classification model built using skl
earn


def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred = model.predict(predictors)

    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred)  # to compute Recall
    precision = precision_score(target, pred)  # to compute Precision
    f1 = f1_score(target, pred)  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {"Accuracy": acc, "Recall": recall, "Precision": precision, "F1": f1,},
        index=[0],
    )

    return df_perf
```

In [131]:

```
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    y_pred = model.predict(predictors)
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

Defining scorer to be used for cross-validation and hyperparameter tuning

Let us choose the **F1 score** since that is a balance between precision and recall. We dont want people being denied visa(when they truly deserve) and also we dont want them approved when they are not supposed to be. (This might be a security issue in the USA)

In [132]:

```
scorer = metrics.make_scorer(metrics.f1_score)
```

We are now done with pre-processing and evaluation criterion, so let's start building the model.

Model building with Original data

In [133]:

```
models = []  # Empty list to store all the models
results = []  # Empty list to store all model's CV scores
names = []  # Empty list to store name of the models

# Appending models into the list
models.append(("Bagging", BaggingClassifier(random_state=42)))
models.append(("Random forest", RandomForestClassifier(random_state=42)))
models.append(("GBM", GradientBoostingClassifier(random_state=42)))
models.append(("Adaboost", AdaBoostClassifier(random_state=42)))
models.append(("dtree", DecisionTreeClassifier(random_state=42)))
```

In [134]:

```
print(X_train.shape, X_val.shape, X_test.shape)
print(y_train.shape, y_val.shape, y_test.shape)
```

```
(17836, 21) (3822, 21) (3822, 21)
(17836,) (3822,) (3822,)
```

In [135]:

```
# Lets cross validate
print("\n" "Cross-Validation performance on training dataset for all the 5 models:" "\n")

  #keeping the n_splits low since the running is taking a long time and also crashing the runtime engine sometimes.
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for name, model in models:
    cv_result = cross_val_score(
        estimator=model, X=X_train, y=y_train, scoring = scorer,cv=kfold
    )
    results.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean()))
    model.fit(X_train,y_train)
    perf = model_performance_classification_sklearn(model, X_val, y_val)
    conf = confusion_matrix_sklearn(model, X_val, y_val)
    print("\n" "Performance: Various Parameters" "\n")
    print(perf)
    print("\n" "Performance: Confusion Matrix" "\n")
    print(conf)
    print("\n")
```

Cross-Validation performance on training dataset for all the 5 models:

Bagging: 0.776902593316487

Performance: Various Parameters

	Accuracy	Recall	Precision	F1
0	0.694139	0.779083	0.766769	0.772877

Performance: Confusion Matrix

None

Random forest: 0.8062327459069424

Performance: Various Parameters

	Accuracy	Recall	Precision	F1
0	0.72449	0.840971	0.768432	0.803067

Performance: Confusion Matrix

None

GBM: 0.824619206135037

Performance: Various Parameters

	Accuracy	Recall	Precision	F1
0	0.744113	0.871524	0.773913	0.819823

Performance: Confusion Matrix

None

Adaboost: 0.8200334908816742

Performance: Various Parameters

	Accuracy	Recall	Precision	F1
0	0.731031	0.8868	0.753913	0.814975

Performance: Confusion Matrix

None

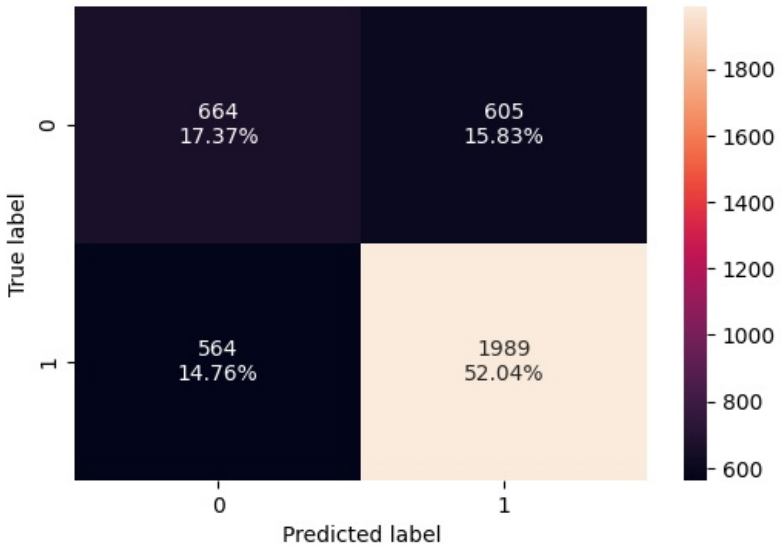
dtree: 0.7431976564222731

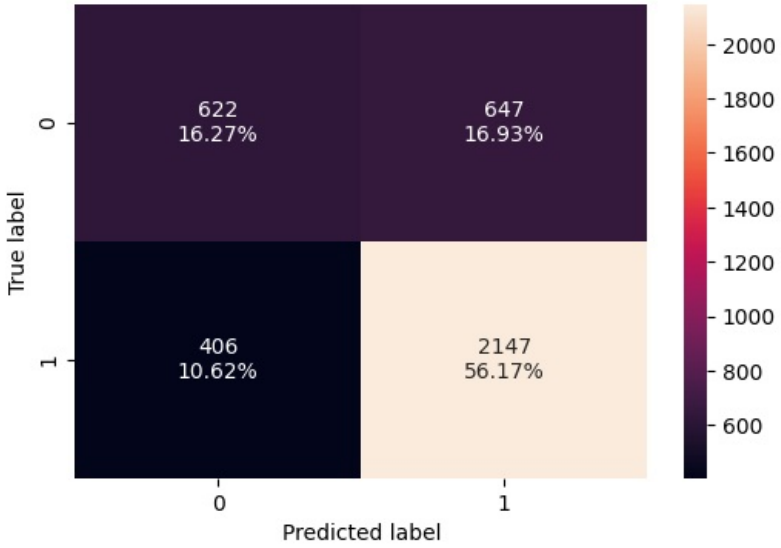
Performance: Various Parameters

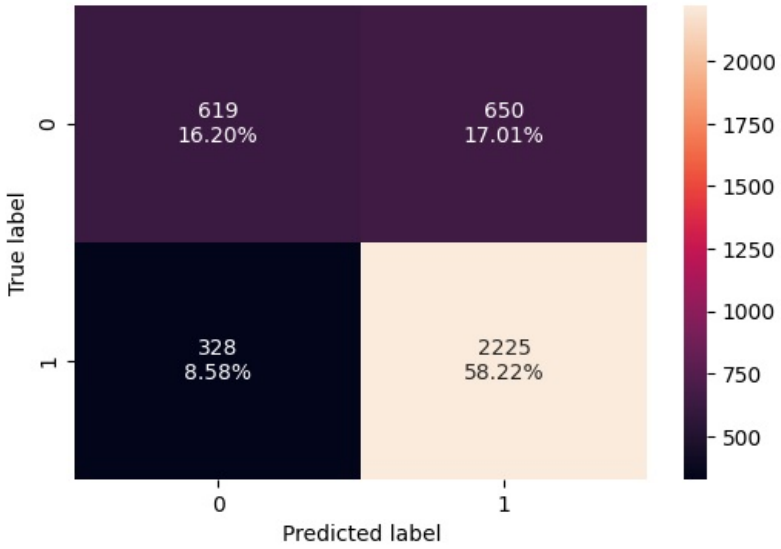
	Accuracy	Recall	Precision	F1
0	0.657509	0.747356	0.741835	0.744585

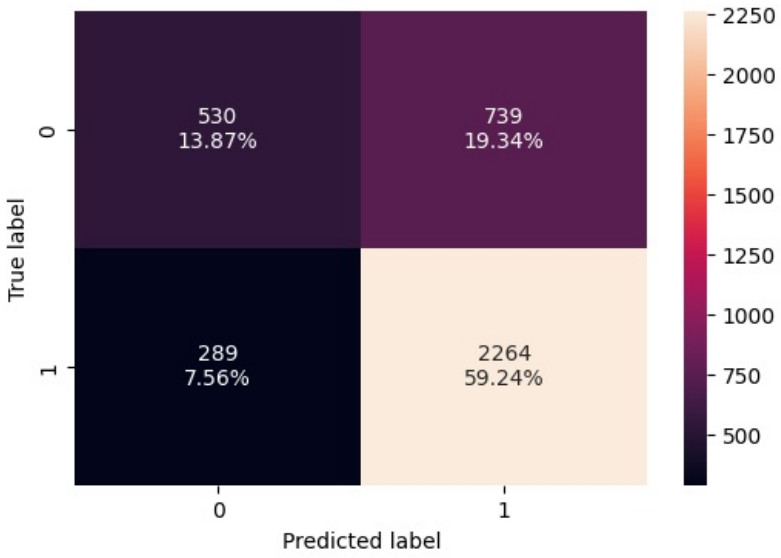
Performance: Confusion Matrix

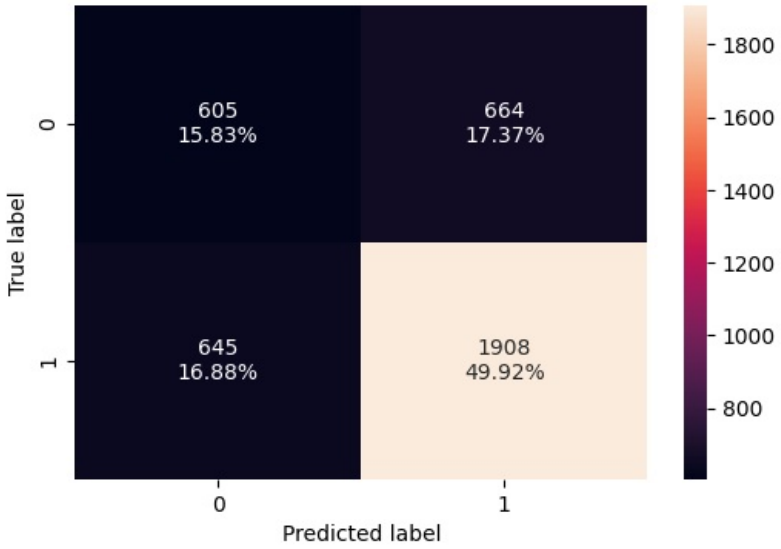
None


664
17.37%
605
15.83%
564
14.76%
1989
52.04%
True label
Predicted label
0
1
1800
1600
1400
1200
1000
800
600


622
16.27%
647
16.93%
406
10.62%
2147
56.17%
True label
Predicted label
0
1
2000
1800
1600
1400
1200
1000
800
600

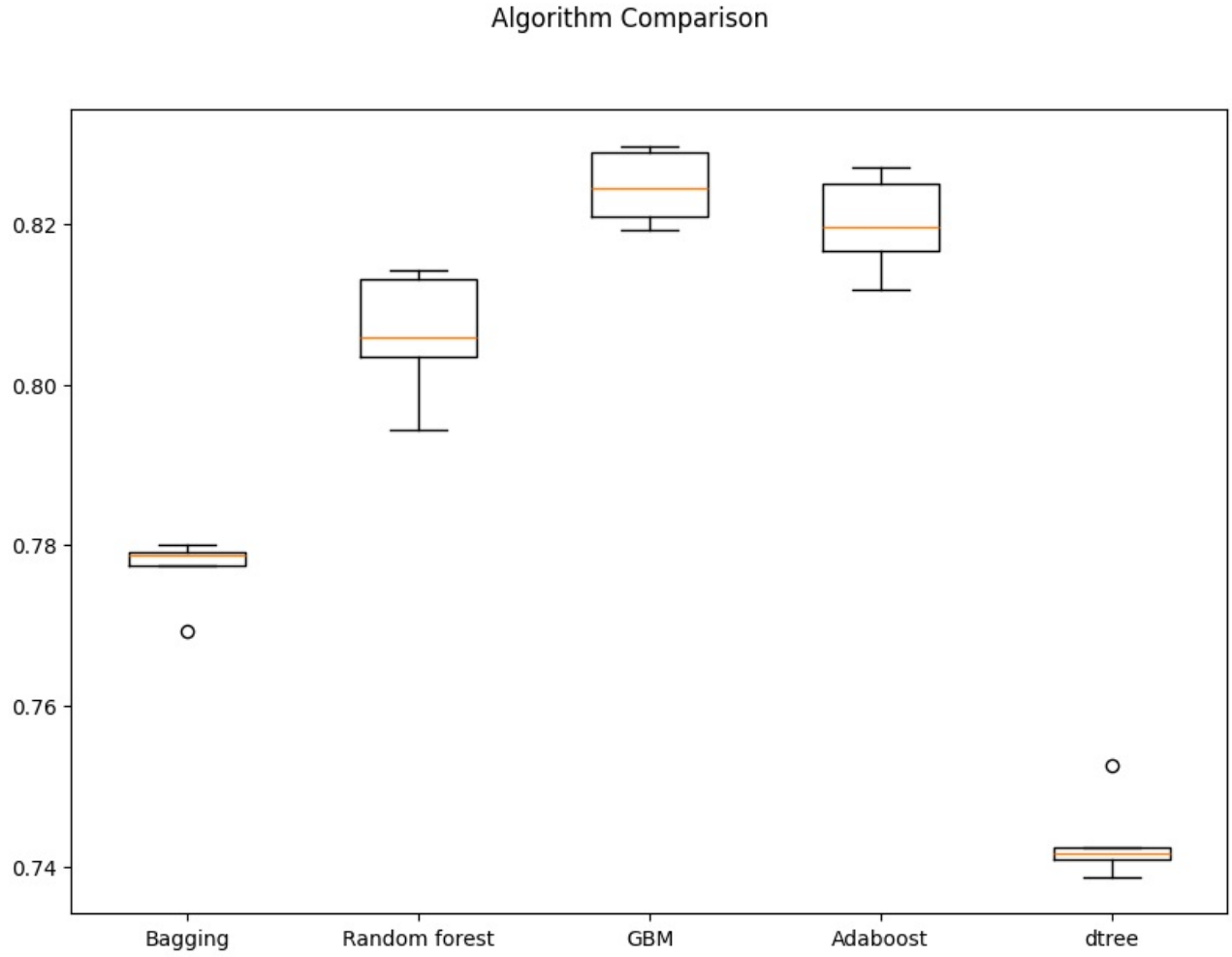

619
16.20%
650
17.01%
328
8.58%
2225
58.22%
True label
Predicted label
0
1
2000
1750
1500
1250
1000
750
500


530
13.87%
739
19.34%
289
7.56%
2264
59.24%
True label
Predicted label
0
1
2250
2000
1750
1500
1250
1000
750
500


605
15.83%
664
17.37%
645
16.88%
1908
49.92%
True label
Predicted label
0
1
1800
1600
1400
1200
1000
800

In [136]:

```
#Plotting boxplots for CV scores of all models defined above
fig = plt.figure(figsize=(10, 7))

fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results)
ax.set_xticklabels(names)

plt.show()
```



Model Building with Oversampled data

Lets check for class imbalance.

In [137]:

```
# Checking class balance for whole data, train set, validation set, and test set

print("Target value ratio in y")
print(y.value_counts(1))
print("*" * 80)
print("Target value ratio in y_train")
print(y_train.value_counts(1))
print("*" * 80)
print("Target value ratio in y_val")
print(y_val.value_counts(1))
print("*" * 80)
print("Target value ratio in y_test")
print(y_test.value_counts(1))
print("*" * 80)
```

```
Target value ratio in y
case_status
1    0.667896
0    0.332104
Name: proportion, dtype: float64
********************************************************************************
Target value ratio in y_train
case_status
1    0.667919
0    0.332081
Name: proportion, dtype: float64
********************************************************************************
Target value ratio in y_val
case_status
1    0.667975
0    0.332025
Name: proportion, dtype: float64
********************************************************************************
Target value ratio in y_test
case_status
1    0.667713
0    0.332287
Name: proportion, dtype: float64
********************************************************************************
```

In [138]:

```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train == 0)))

# Synthetic Minority Over Sampling Technique
sm = SMOTE(sampling_strategy=1, k_neighbors=5, random_state=42)
X_train_over, y_train_over = sm.fit_resample(X_train, y_train)


print("After OverSampling, counts of label '1': {}".format(sum(y_train_over == 1)))
print("After OverSampling, counts of label '0': {} \n".format(sum(y_train_over == 0)))


print("After OverSampling, the shape of train_X: {}".format(X_train_over.shape))
print("After OverSampling, the shape of train_y: {} \n".format(y_train_over.shape))
```

```
Before OverSampling, counts of label '1': 11913
Before OverSampling, counts of label '0': 5923 

After OverSampling, counts of label '1': 11913
After OverSampling, counts of label '0': 11913 

After OverSampling, the shape of train_X: (23826, 21)
After OverSampling, the shape of train_y: (23826,) 
```

In [139]:

```
models = []  # Empty list to store all the models
results = []  # Empty list to store all model's CV scores
names = []  # Empty list to store name of the models

# Appending models into the list
models.append(("Bagging", BaggingClassifier(random_state=42)))
models.append(("Random forest", RandomForestClassifier(random_state=42)))
models.append(("GBM", GradientBoostingClassifier(random_state=42)))
models.append(("Adaboost", AdaBoostClassifier(random_state=42)))
models.append(("dtree", DecisionTreeClassifier(random_state=42)))

# loop through all models to get the mean cross validated score
print("\n" "Cross-Validation performance on training dataset using oversampled data:" "\n")
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)

for name, model in models:
    cv_result = cross_val_score(
        estimator=model, X=X_train_over, y=y_train_over,scoring = scorer, cv=kfold
    )
    results.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean()))
    model.fit(X_train_over,y_train_over)
    perf = model_performance_classification_sklearn(model, X_val, y_val)
    conf = confusion_matrix_sklearn(model, X_val, y_val)
    print("\n" "Performance: Various Parameters" "\n")
    print(perf)
    print("\n" "Performance: Confusion Matrix" "\n")
    print(conf)
    print("\n")
```

Cross-Validation performance on training dataset using oversampled data:

Bagging: 0.7615542807808515

Performance: Various Parameters

	Accuracy	Recall	Precision	F1
0	0.693093	0.762241	0.774682	0.768411

Performance: Confusion Matrix

None

Random forest: 0.7927067506689416

Performance: Various Parameters

	Accuracy	Recall	Precision	F1
0	0.719257	0.817861	0.774481	0.79558

Performance: Confusion Matrix

None

GBM: 0.8050749813829021

Performance: Various Parameters

	Accuracy	Recall	Precision	F1
0	0.743328	0.854289	0.78172	0.816395

Performance: Confusion Matrix

None

Adaboost: 0.7908741049483738

Performance: Various Parameters

	Accuracy	Recall	Precision	F1
0	0.731816	0.855464	0.769014	0.809939

Performance: Confusion Matrix

None

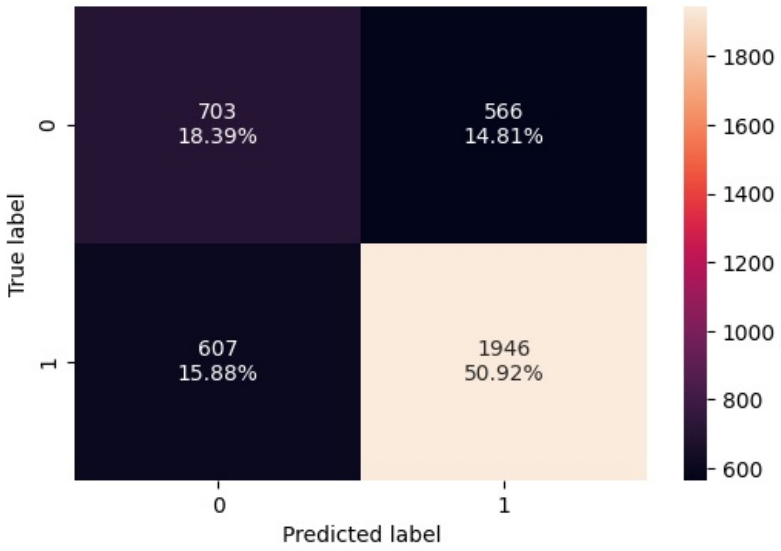
dtree: 0.7301740046260312

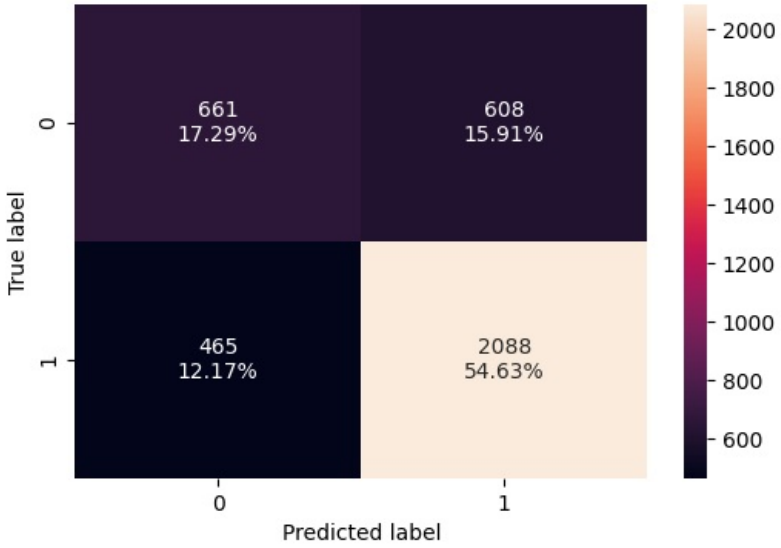
Performance: Various Parameters

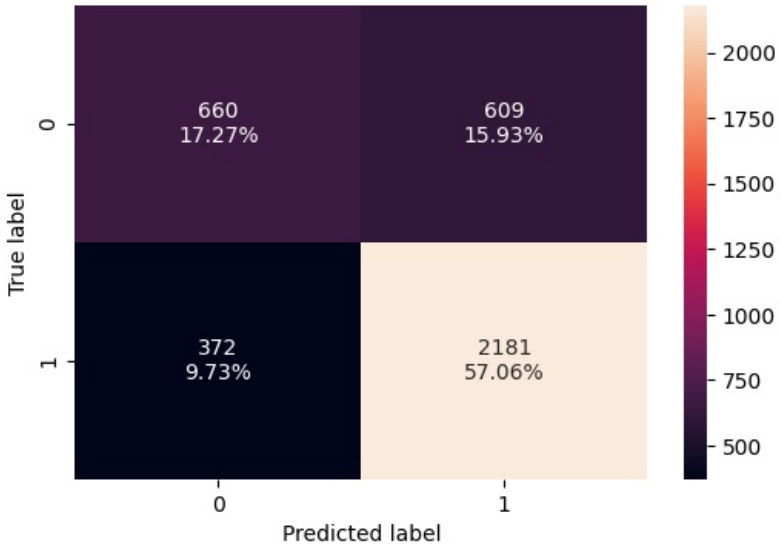
	Accuracy	Recall	Precision	F1
0	0.650183	0.732863	0.740697	0.736759

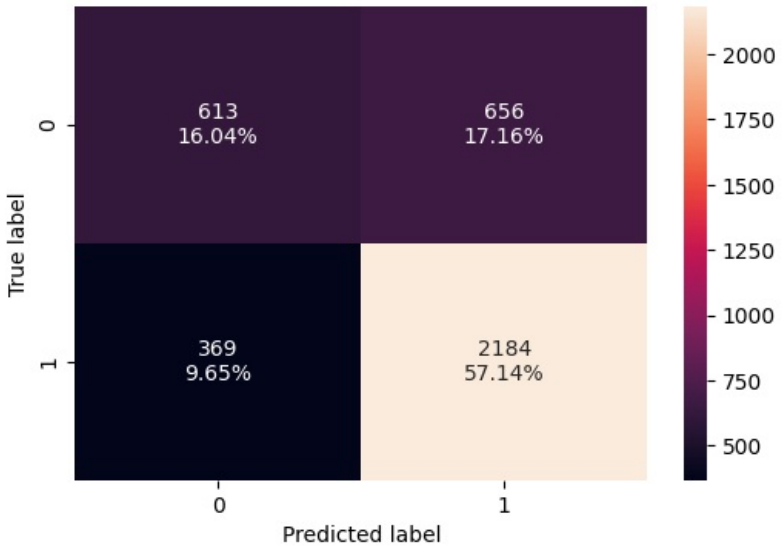
Performance: Confusion Matrix

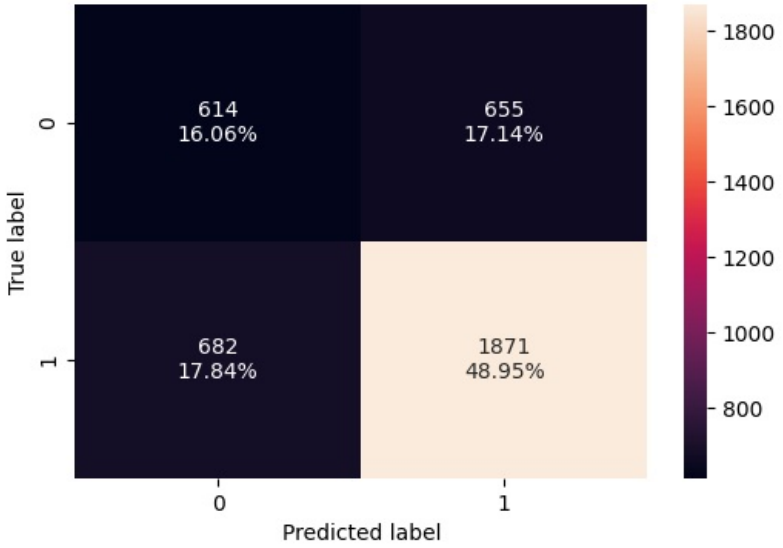
None


703
18.39%
566
14.81%
607
15.88%
1946
50.92%
True label
Predicted label
0
1
1800
1600
1400
1200
1000
800
600


661
17.29%
608
15.91%
465
12.17%
2088
54.63%
True label
Predicted label
0
1
2000
1800
1600
1400
1200
1000
800
600

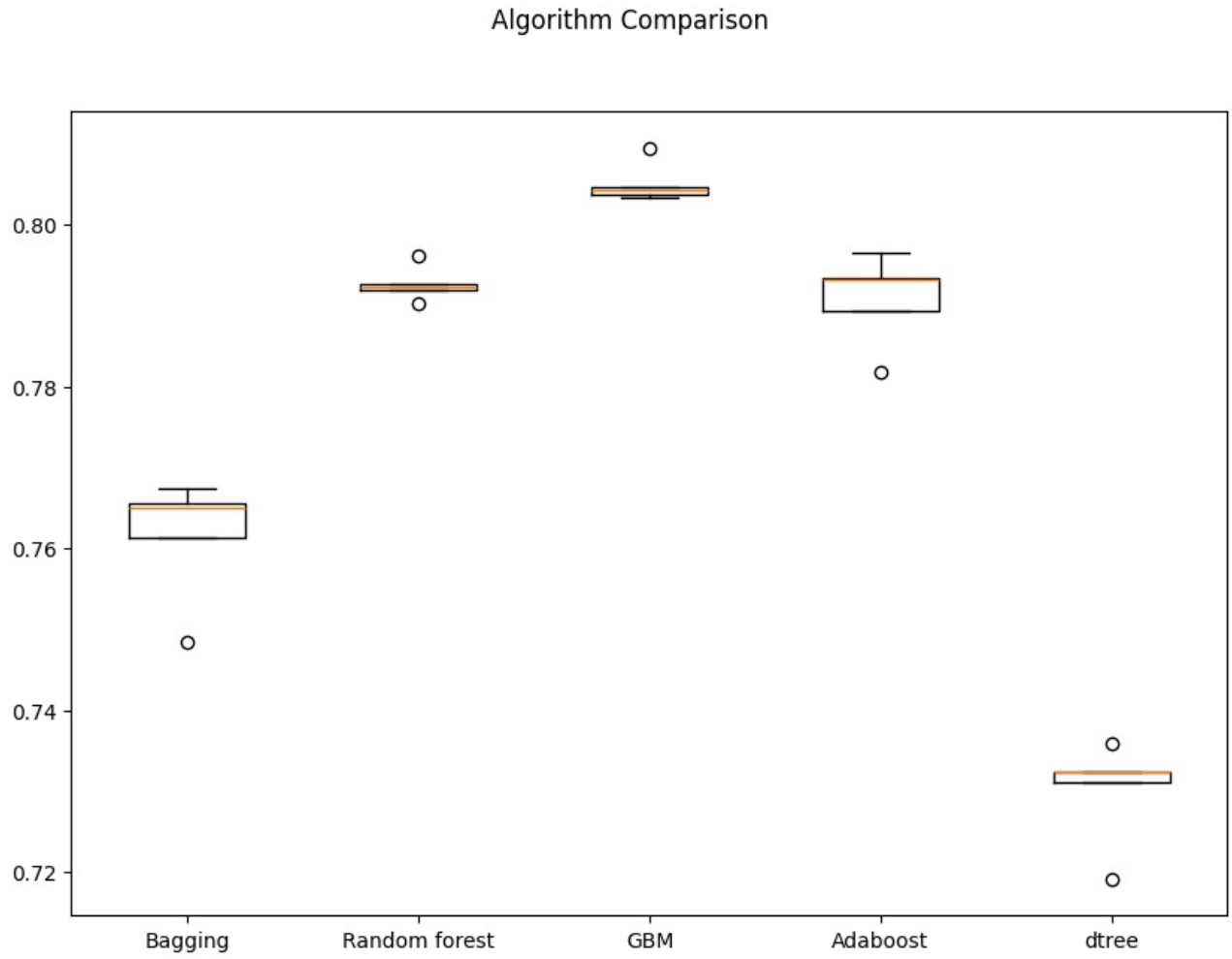

660
17.27%
609
15.93%
372
9.73%
2181
57.06%
True label
Predicted label
0
1
2000
1750
1500
1250
1000
750
500


613
16.04%
656
17.16%
369
9.65%
2184
57.14%
True label
Predicted label
0
1
2000
1750
1500
1250
1000
750
500


614
16.06%
655
17.14%
682
17.84%
1871
48.95%
True label
Predicted label
0
1
1800
1600
1400
1200
1000
800

In [140]:

```
# Plotting boxplots for CV scores of all models defined above
fig = plt.figure(figsize=(10, 7))

fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results)
ax.set_xticklabels(names)

plt.show()
```



Model Building with Undersampled data

In [141]:

```
rus = RandomUnderSampler(random_state=42, sampling_strategy=1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)


print("Before UnderSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before UnderSampling, counts of label '0': {} \n".format(sum(y_train == 0)))


print("After UnderSampling, counts of label '1': {}".format(sum(y_train_un == 1)))
print("After UnderSampling, counts of label '0': {} \n".format(sum(y_train_un == 0)))


print("After UnderSampling, the shape of train_X: {}".format(X_train_un.shape))
print("After UnderSampling, the shape of train_y: {} \n".format(y_train_un.shape))
```

```
Before UnderSampling, counts of label '1': 11913
Before UnderSampling, counts of label '0': 5923

After UnderSampling, counts of label '1': 5923
After UnderSampling, counts of label '0': 5923

After UnderSampling, the shape of train_X: (11846, 21)
After UnderSampling, the shape of train_y: (11846,)
```

In [142]:

```
models = []  # Empty list to store all the models
results = []  # Empty list to store all model's CV scores
names = []  # Empty list to store name of the models

# Appending models into the list
models.append(("Bagging", BaggingClassifier(random_state=42)))
models.append(("Random forest", RandomForestClassifier(random_state=42)))
models.append(("GBM", GradientBoostingClassifier(random_state=42)))
models.append(("Adaboost", AdaBoostClassifier(random_state=42)))
models.append(("dtree", DecisionTreeClassifier(random_state=42)))


# loop through all models to get the mean cross validated score
print("\n" "Cross-Validation performance on training dataset using undersampled data:" "\n")
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for name, model in models:
    cv_result = cross_val_score(estimator=model, X=X_train_un, y=y_train_un,scoring = scorer, cv=kfold)
    results.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean()))
    model.fit(X_train_un,y_train_un)
    perf = model_performance_classification_sklearn(model, X_val, y_val)
    conf = confusion_matrix_sklearn(model, X_val, y_val)
    print("\n" "Performance: Various Parameters" "\n")
    print(perf)
    print("\n" "Performance: Confusion Matrix" "\n")
    print(conf)
    print("\n")
```

Cross-Validation performance on training dataset using undersampled data:

Bagging: 0.648227026875054

Performance: Various Parameters

	Accuracy	Recall	Precision	F1
0	0.644689	0.609087	0.81201	0.696061

Performance: Confusion Matrix

None

Random forest: 0.6883663664092916

Performance: Various Parameters

	Accuracy	Recall	Precision	F1
0	0.68001	0.681943	0.809015	0.740064

Performance: Confusion Matrix

None

GBM: 0.7131226753112041

Performance: Various Parameters

	Accuracy	Recall	Precision	F1
0	0.70853	0.732863	0.812419	0.770593

Performance: Confusion Matrix

None

Adaboost: 0.7048541754838573

Performance: Various Parameters

	Accuracy	Recall	Precision	F1
0	0.694924	0.724246	0.800087	0.76028

Performance: Confusion Matrix

None

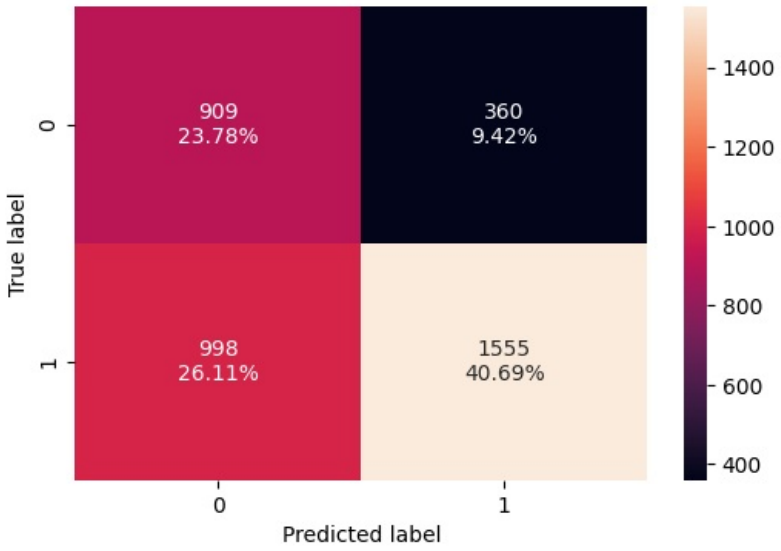
dtree: 0.6213718649289948

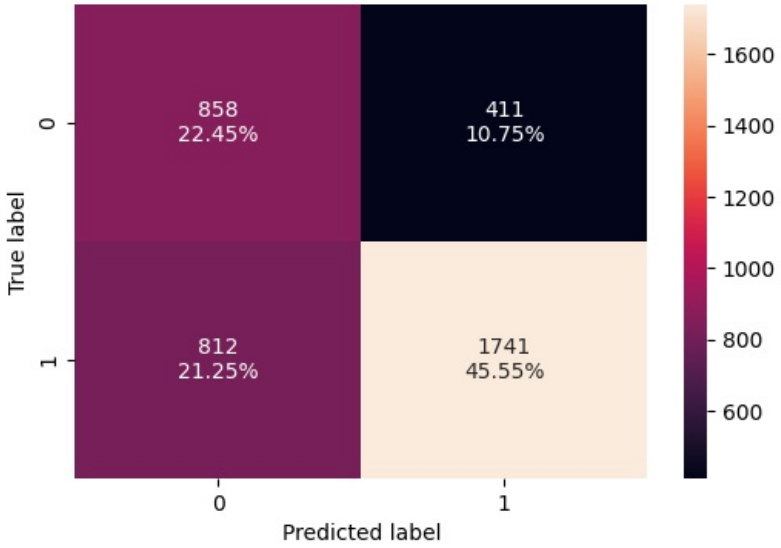
Performance: Various Parameters

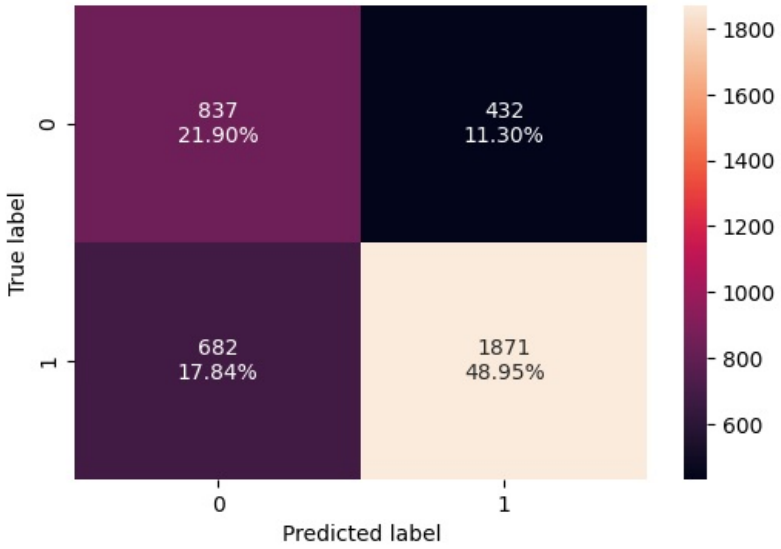
	Accuracy	Recall	Precision	F1
0	0.623234	0.636898	0.760168	0.693095

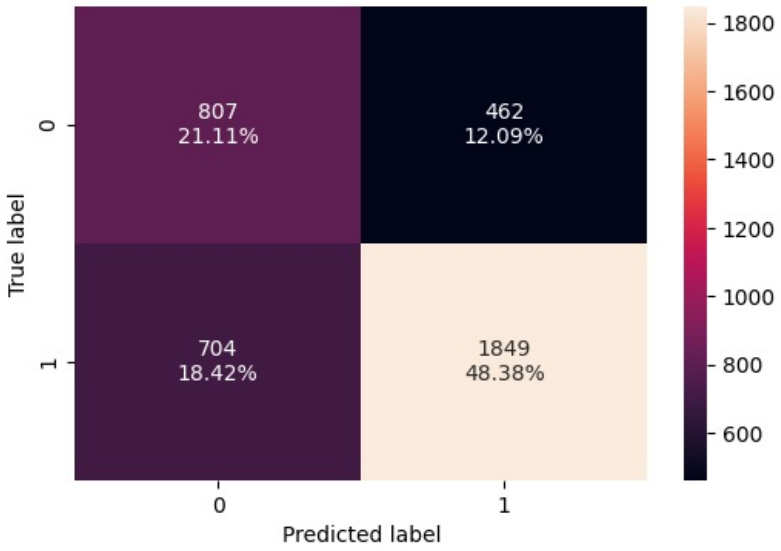
Performance: Confusion Matrix

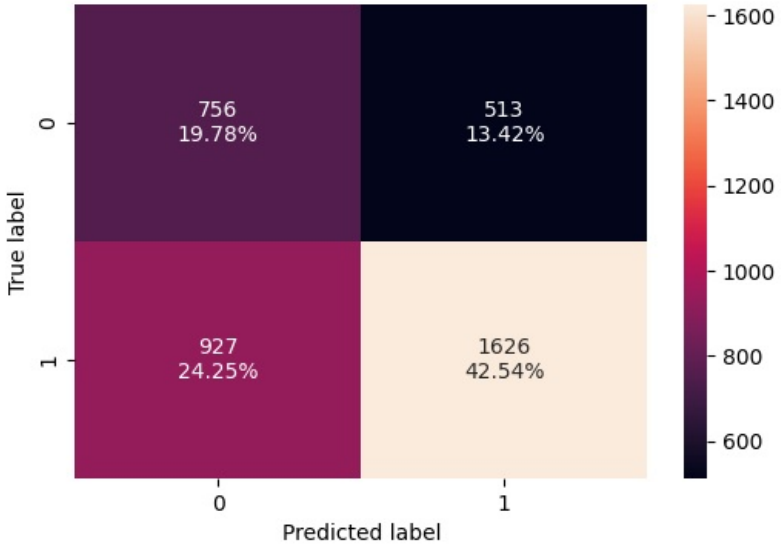
None


909
23.78%
360
9.42%
998
26.11%
1555
40.69%
True label
Predicted label
0
1
1400
1200
1000
800
600
400


858
22.45%
411
10.75%
812
21.25%
1741
45.55%
True label
Predicted label
0
1
1600
1400
1200
1000
800
600


837
21.90%
432
11.30%
682
17.84%
1871
48.95%
True label
Predicted label
0
1
1800
1600
1400
1200
1000
800
600





Hyperparameter Tuning

The 3 models that will be used for Hyperparameter tuning

RandomForest - with oversampling
GBM - with oversampling
AdaBoost - with oversampling.

Best practices for hyperparameter tuning in AdaBoost:

`n_estimators` :

- Start with a specific number (50 is used in general) and increase in steps: 50, 75, 85, 100
- Use fewer estimators (e.g., 50 to 100) if using complex base learners (like deeper decision trees)
- Use more estimators (e.g., 100 to 150) when learning rate is low (e.g., 0.1 or lower)
- Avoid very high values unless performance keeps improving on validation

`learning_rate` :

- Common values to try: 1.0, 0.5, 0.1, 0.01
- Use 1.0 for faster training, suitable for fewer estimators
- Use 0.1 or 0.01 when using more estimators to improve generalization
- Avoid very small values (< 0.01) unless you plan to use many estimators (e.g., >500) and have sufficient data

In [143]:

```
%%time

# defining model
model = AdaBoostClassifier(random_state=42)

# Parameter grid to pass in RandomSearchCV
param_grid = {
    "n_estimators": [50, 75, 85, 100],
    "learning_rate": [0.01, 0.1, 0.5, 1],
    "estimator": [DecisionTreeClassifier(max_depth=1, random_state=42), DecisionTreeClassifier(max_depth=2, rando
m_state=42), DecisionTreeClassifier(max_depth=3, random_state=42),
    ]
}

#Calling RandomizedSearchCV
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

randomized_cv = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_grid,
    n_iter=7,
    scoring=scorer,
    cv=kfold,
    random_state=42
)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_over,y_train_over)

# Print the best combination of parameters
adaboost_bestparams = randomized_cv.best_params_
adaboost_bestmodel = randomized_cv.best_estimator_
print("Best parameters: ", adaboost_bestparams)
print("Best model: ", adaboost_bestmodel)

adaboost_train = model_performance_classification_sklearn(adaboost_bestmodel, X_train_over, y_train_over)
adaboost_val = model_performance_classification_sklearn(adaboost_bestmodel, X_val, y_val)

print("\n" "Train Performance:" "\n")
print(adaboost_train)
print("\n" "Validation Performance:" "\n")
print(adaboost_val)
print("\n")
```

```
Best parameters:  {'n_estimators': 100, 'learning_rate': 0.5, 'estimator': DecisionTreeClassifier(ma
x_depth=3, random_state=42)}
Best model:  AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=3,
                                                 random_state=42),
                   learning_rate=0.5, n_estimators=100, random_state=42)

Train Performance:

   Accuracy    Recall  Precision        F1
0  0.791908  0.856879   0.758339  0.804603

Validation Performance:

   Accuracy    Recall  Precision        F1
0  0.735217  0.852722   0.773907  0.811405


CPU times: user 1min 3s, sys: 78.1 ms, total: 1min 3s
Wall time: 1min 3s
```

Best practices for hyperparameter tuning in Random Forest:

`n_estimators` :

- Start with a specific number (50 is used in general) and increase in steps: 50, 75, 100, 125
- Higher values generally improve performance but increase training time
- Use 100-150 for large datasets or when variance is high

`min_samples_leaf` :

- Try values like: 1, 2, 4, 5, 10
- Higher values reduce model complexity and help prevent overfitting
- Use 1–2 for low-bias models, higher (like 5 or 10) for more regularized models
- Works well in noisy datasets to smooth predictions

`max_features` :

- Try values: `"sqrt"` (default for classification), `"log2"` , `None` , or float values (e.g., `0.3` , `0.5`)
- `"sqrt"` balances between diversity and performance for classification tasks
- Lower values (e.g., `0.3`) increase tree diversity, reducing overfitting
- Higher values (closer to `1.0`) may capture more interactions but risk overfitting

`max_samples` (for bootstrap sampling):

- Try float values between `0.5` to `1.0` or fixed integers
- Use `0.6–0.9` to introduce randomness and reduce overfitting
- Smaller values increase diversity between trees, improving generalization

In [144]:

```
%%time

# defining model
model = RandomForestClassifier(random_state=42)

# Parameter grid to pass in RandomSearchCV
param_grid = {
    "n_estimators": [50, 75, 100, 125], ## Complete the code to set the number of estimators.
    "min_samples_leaf": [2, 4, 5], ## Complete the code to set the minimum number of samples in the leaf node.
    "max_features": [0.3, 0.5], ## Complete the code to set the maximum number of features.
    "max_samples": [0.6, 0.9], ## Complete the code to set the maximum number of samples.
}


#Calling RandomizedSearchCV
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
randomized_cv = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_grid,
    n_iter=7,
    scoring=scorer,
    cv=kfold,
    random_state=42
)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_over, y_train_over)
randomForest_bestparams = randomized_cv.best_params_
randomForest_bestmodel = randomized_cv.best_estimator_
print("Best parameters: ", randomForest_bestparams)
print("Best model: ", randomForest_bestmodel)

# perf = model_performance_classification_sklearn(randomForest_bestmodel, X_val, y_val)
# conf = confusion_matrix_sklearn(randomForest_bestmodel, X_val, y_val)
randomForest_trained = model_performance_classification_sklearn(randomForest_bestmodel, X_train_over, y_train_ove
r)
randomForest_val = model_performance_classification_sklearn(randomForest_bestmodel, X_val, y_val)
print("\n" "Performance:" "\n")
print("\n" "Train Performance:" "\n")
print(randomForest_trained)
print("\n" "Validation Performance:" "\n")
print(randomForest_val)
print("\n")
```

```
Best parameters:  {'n_estimators': 50, 'min_samples_leaf': 2, 'max_samples': 0.9, 'max_features': 0.
3}
Best model:  RandomForestClassifier(max_features=0.3, max_samples=0.9, min_samples_leaf=2,
                       n_estimators=50, random_state=42)

Performance:


Train Performance:

   Accuracy    Recall  Precision        F1
0   0.92781  0.953916   0.906582  0.929647

Validation Performance:

   Accuracy    Recall  Precision        F1
0  0.732601  0.835879   0.779686  0.806805


CPU times: user 1min 28s, sys: 91.7 ms, total: 1min 28s
Wall time: 1min 28s
```

Best practices for hyperparameter tuning in Gradient Boosting:

`n_estimators` :

- Start with 100 (default) and increase: 100, 200, 300, 500
- Typically, higher values lead to better performance, but they also increase training time
- Use 200–500 for larger datasets or complex problems
- Monitor validation performance to avoid overfitting, as too many estimators can degrade generalization

`learning_rate` :

- Common values to try: 0.1, 0.05, 0.01, 0.005
- Use lower values (e.g., 0.01 or 0.005) if you are using many estimators (e.g., > 200)
- Higher learning rates (e.g., 0.1) can be used with fewer estimators for faster convergence
- Always balance the learning rate with `n_estimators` to prevent overfitting or underfitting

`subsample` :

- Common values: 0.7, 0.8, 0.9, 1.0
- Use a value between `0.7` and `0.9` for improved generalization by introducing randomness
- `1.0` uses the full dataset for each boosting round, potentially leading to overfitting
- Reducing `subsample` can help reduce overfitting, especially in smaller datasets

`max_features` :

- Common values: `"sqrt"` , `"log2"` , or float (e.g., `0.3` , `0.5`)
- `"sqrt"` (default) works well for classification tasks
- Lower values (e.g., `0.3`) help reduce overfitting by limiting the number of features considered at each split

In [145]:

```
%%time

# defining model
model = GradientBoostingClassifier(random_state=1)

param_grid={
    "n_estimators": [50,100,150, 200, 250],
    "learning_rate": [0.1, 0.05, 0.01, 0.005],
    "subsample":[0.7, 0.8, 0.9, 1.0],
    "max_features":[0.3, 0.5]
}

#Calling RandomizedSearchCV
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
randomized_cv = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_grid,
    n_iter=7,
    scoring=scorer,
    cv=kfold,
    random_state=1
)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_over, y_train_over)
gbBoost_bestparams = randomized_cv.best_params_
gbBoost_bestmodel = randomized_cv.best_estimator_
print("Best GD Boost parameters: ", gbBoost_bestparams)
print("Best GD Boost model: ", gbBoost_bestmodel)

gbm_trained = model_performance_classification_sklearn(gbBoost_bestmodel, X_train_over, y_train_over)
gbm_val = model_performance_classification_sklearn(gbBoost_bestmodel, X_val, y_val)
print("\n" "Performance: on trained data" "\n")
print(gbm_trained)
print("\n" "Performance: on validated data" "\n")
print(gbm_val)
print("\n")
```

```
Best GD Boost parameters:  {'subsample': 1.0, 'n_estimators': 250, 'max_features': 0.3, 'learning_ra
te': 0.1}
Best GD Boost model:  GradientBoostingClassifier(max_features=0.3, n_estimators=250, random_state=1)

Performance: on trained data

   Accuracy    Recall  Precision        F1
0  0.802233  0.863007   0.769478  0.813563

Performance: on validated data

   Accuracy    Recall  Precision        F1
0   0.74202  0.857031   0.778925  0.816113


CPU times: user 51.3 s, sys: 69.3 ms, total: 51.3 s
Wall time: 51.4 s
```

Model Performance Summary and Final Model Selection

We will go with the GBM model since they gave the best F1 scores.

The details have been provided in the table(s) below

In [146]:

```
# training performance comparison
models_train_comp_df = pd.concat(
    [
        gbm_trained.T,
        adaboost_train.T,
        randomForest_trained.T
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "GBM (oversampled)",
    "AdaBoost (oversampled)",
    "Random forest (oversampled)",
]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[146]:

	GBM (oversampled)	AdaBoost (oversampled)	Random forest (oversampled)
Accuracy	0.802233	0.791908	0.927810
Recall	0.863007	0.856879	0.953916
Precision	0.769478	0.758339	0.906582
F1	0.813563	0.804603	0.929647

In [147]:

```
# validation performance comparison

models_val_comp_df = pd.concat(
    [
        gbm_val.T,
        adaboost_val.T,
        randomForest_val.T,
    ],
    axis=1,
)
models_val_comp_df.columns = [
    "GBM (oversampled)",
    "AdaBoost (oversampled)",
    "Random forest (oversampled)",
]
print("Validation performance comparison:")
models_val_comp_df
```

```
Validation performance comparison:
```

Out[147]:

	GBM (oversampled)	AdaBoost (oversampled)	Random forest (oversampled)
Accuracy	0.742020	0.735217	0.732601
Recall	0.857031	0.852722	0.835879
Precision	0.778925	0.773907	0.779686
F1	0.816113	0.811405	0.806805

As we can see that the gradient boosting with oversampling gives consistent results in training and validation.
The F1 scores are 0.813 and 0.816.
So we will go with that.

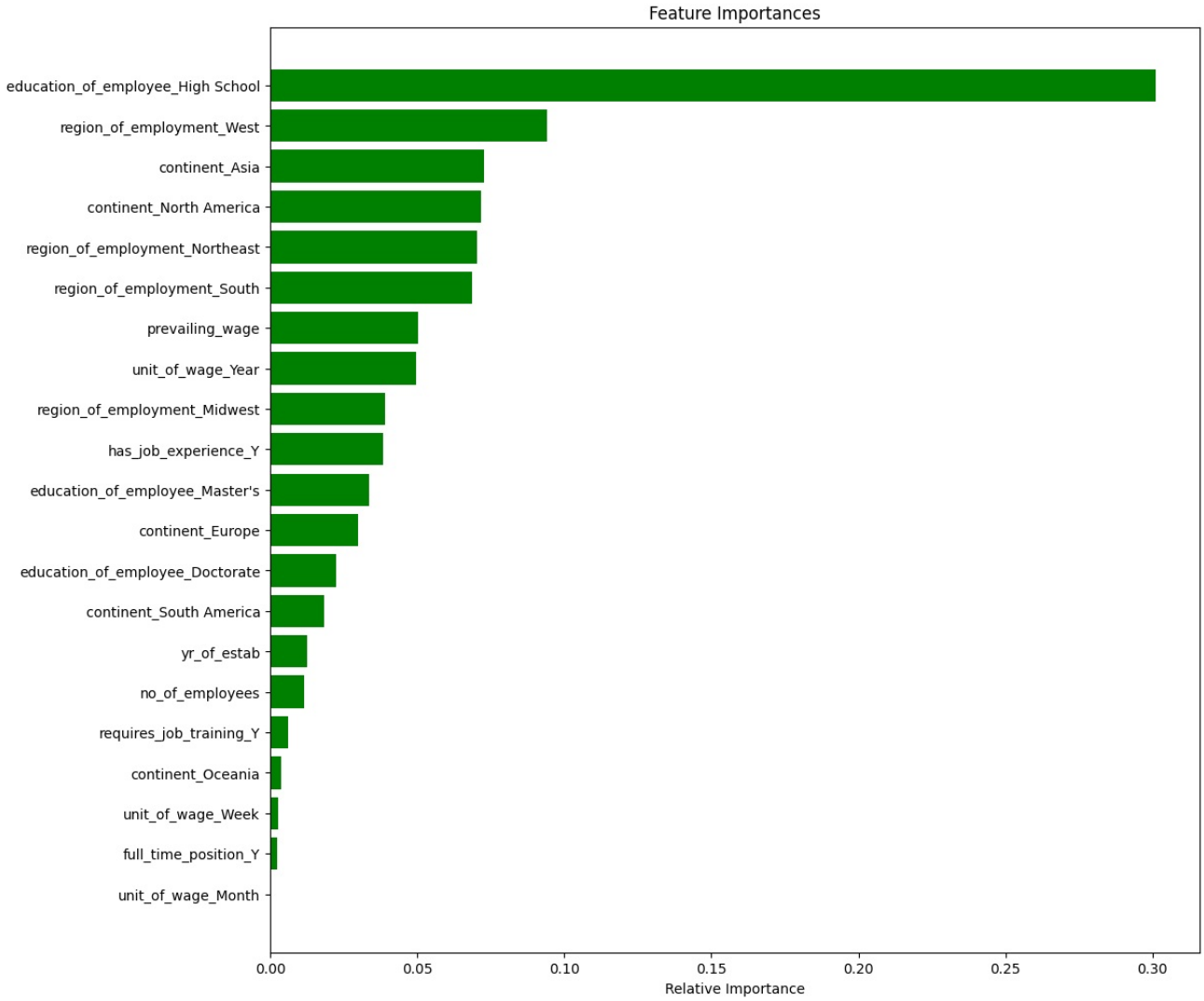
In [148]:

```
final_model = gbBoost_bestmodel
```

In [149]:

```
production_model = model_performance_classification_sklearn(final_model, X_test, y_test)
print(production_model)
print("\n")

feature_names = X_train.columns
importances = final_model.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(12, 12))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="green", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```

```
   Accuracy    Recall  Precision        F1
0   0.73888  0.852273   0.777897  0.813388
```



Actionable Insights and Recommendations

Power Ahead

The gradient boosting model gives the best F1 scores.

- The F1 scores are pretty consistent across training, validation and test which shows very clearly that we have not overfit.

In order to increase the likeliness of a visa certification (approval)

1. Have a PhD or a masters degree
2. Being from Europe or Africa incrases the chances.
3. Having prior work experience also increases the changes
4. Having yearly wage also increases the chances.

We can conclude that we need to let the visa aspirants know that they need to study hard and get at least a bachelors and have work experience. This would surely increase their chances of getting a visa approved.