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 [1]:
```
# 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 [2]:
```
# Libraries to help with reading and manipulating data
import pandas as pd
import numpy as np

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

# To tune model, get different metric scores, and split data
from sklearn.metrics import (
    f1_score,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
    ConfusionMatrixDisplay,
)
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_v
al_score

# To be used for data scaling and one hot encoding
from sklearn.preprocessing import StandardScaler, MinMaxScaler, OneHotEncoder

# To impute missing values
from sklearn.impute import SimpleImputer
from sklearn import metrics

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

# To do hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV

# To define maximum number of columns to be displayed in a dataframe
pd.set_option("display.max_columns", None)

# To supress scientific notations for a dataframe
pd.set_option("display.float_format", lambda x: "%.3f" % x)

# To help with model building
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import (
    AdaBoostClassifier,
    GradientBoostingClassifier,
    RandomForestClassifier,
    BaggingClassifier,
)
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression

# To suppress scientific notations
pd.set_option("display.float_format", lambda x: "%.3f" % x)

# To supress warnings
```

```
import warnings

warnings.filterwarnings("ignore")
```

Import Dataset

In [3]:
```
from google.colab import drive
drive.mount('/content/drive')
file = r'/content/drive/MyDrive/AI Class/Projects/Project 3/EasyVisa.csv'
EasyVisaCampgn = pd.read_csv(file)
```

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

Overview of the Dataset

View the first and last 5 rows of the dataset

In [4]:
```
data = EasyVisaCampgn.copy()
```

In [5]:
```
data.head()
```

Out[5]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_training	no_of_e
0	EZYV01	Asia	High School	N	N	
1	EZYV02	Asia	Master's	Y	N	
2	EZYV03	Asia	Bachelor's	N	Y	
3	EZYV04	Asia	Bachelor's	N	N	
4	EZYV05	Africa	Master's	Y	N	

In [6]:
```
data.tail()
```

Out[6]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_training
25475	EZYV25476	Asia	Bachelor's	Y	Y
25476	EZYV25477	Asia	High School	Y	N
25477	EZYV25478	Asia	Master's	Y	N
25478	EZYV25479	Asia	Master's	Y	Y
25479	EZYV25480	Asia	Bachelor's	Y	N

Understand the shape of the dataset

In [7]:
```
data.shape
```

Out[7]:
```
(25480, 12)
```

Check the data types of the columns for the dataset

In [8]:
```
data.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
```

In [9]:
```
object_cols = data.select_dtypes(include='object').columns

for col in object_cols:
    data[col] = data[col].astype('category')

data.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  category
 1   continent              25480 non-null  category
 2   education_of_employee  25480 non-null  category
 3   has_job_experience     25480 non-null  category
 4   requires_job_training  25480 non-null  category
 5   no_of_employees        25480 non-null  int64   
 6   yr_of_estab            25480 non-null  int64   
 7   region_of_employment   25480 non-null  category
 8   prevailing_wage        25480 non-null  float64 
 9   unit_of_wage           25480 non-null  category
 10  full_time_position     25480 non-null  category
 11  case_status            25480 non-null  category
dtypes: category(9), float64(1), int64(2)
memory usage: 2.0 MB
```

In [10]:
```
data.duplicated().sum()
```

Out[10]:
```
np.int64(0)
```

In [11]:
```
data.isna().sum()
```

Out[11]:

	0
case_id	0
continent	0
education_of_employee	0
has_job_experience	0
requires_job_training	0
no_of_employees	0
yr_of_estab	0
region_of_employment	0
prevailing_wage	0
unit_of_wage	0
full_time_position	0
case_status	0

dtype: int64

In [12]:
```
data.isnull().sum()
```

Out[12]:

	0
case_id	0
continent	0
education_of_employee	0
has_job_experience	0
requires_job_training	0
no_of_employees	0
yr_of_estab	0
region_of_employment	0
prevailing_wage	0
unit_of_wage	0
full_time_position	0
case_status	0

dtype: int64

In [13]:
```
data.nunique()
```

Out[13]:

	0
case_id	25480
continent	6
education_of_employee	4
has_job_experience	2
requires_job_training	2
no_of_employees	7105
yr_of_estab	199
region_of_employment	5
prevailing_wage	25454
unit_of_wage	4
full_time_position	2
case_status	2

dtype: int64

In [14]:
```
round(data.isnull().sum() / data.isnull().count() * 100, 2)
```

Out[14]:

	0
case_id	0.000
continent	0.000
education_of_employee	0.000
has_job_experience	0.000
requires_job_training	0.000
no_of_employees	0.000
yr_of_estab	0.000
region_of_employment	0.000
prevailing_wage	0.000
unit_of_wage	0.000
full_time_position	0.000
case_status	0.000

dtype: float64

Exploratory Data Analysis (EDA)

Let's check the statistical summary of the data

In [15]:
```
data.describe().T
```

Out[15]:

	count	mean	std	min	25%	50%	75%
no_of_employees	25480.000	5667.043	22877.929	-26.000	1022.000	2109.000	3504.000
yr_of_estab	25480.000	1979.410	42.367	1800.000	1976.000	1997.000	2005.000
prevailing_wage	25480.000	74455.815	52815.942	2.137	34015.480	70308.210	107735.513

In [16]:
```
data.describe(include=['category']).T
```

Out[16]:

	count	unique	top	freq
case_id	25480	25480	EZYV9999	1
continent	25480	6	Asia	16861
education_of_employee	25480	4	Bachelor's	10234
has_job_experience	25480	2	Y	14802
requires_job_training	25480	2	N	22525
region_of_employment	25480	5	Northeast	7195
unit_of_wage	25480	4	Year	22962
full_time_position	25480	2	Y	22773
case_status	25480	2	Certified	17018

Fixing the negative values in number of employees columns

In [17]:
```
negative_rows = data[data['no_of_employees'] < 0]
print(negative_rows)

#printing all rows with employees < 0
```

```
         case_id      continent education_of_employee has_job_experience  \
245      EZYV246         Europe              Master's                  N   
378      EZYV379           Asia            Bachelor's                  N   
832      EZYV833  South America              Master's                  Y   
2918    EZYV2919           Asia              Master's                  Y   
6439    EZYV6440           Asia            Bachelor's                  N   
6634    EZYV6635           Asia            Bachelor's                  Y   
7224    EZYV7225         Europe             Doctorate                  N   
7281    EZYV7282           Asia           High School                  N   
7318    EZYV7319           Asia            Bachelor's                  Y   
7761    EZYV7762           Asia              Master's                  N   
9872    EZYV9873         Europe              Master's                  Y   
11493  EZYV11494           Asia           High School                  Y   
13471  EZYV13472  North America              Master's                  N   
14022  EZYV14023           Asia            Bachelor's                  N   
14146  EZYV14147           Asia            Bachelor's                  N   
14726  EZYV14727           Asia              Master's                  N   
15600  EZYV15601           Asia            Bachelor's                  N   
15859  EZYV15860           Asia           High School                  N   
16157  EZYV16158           Asia              Master's                  Y   
16883  EZYV16884  North America            Bachelor's                  Y   
17006  EZYV17007           Asia             Doctorate                  Y   
17655  EZYV17656  North America            Bachelor's                  Y   
17844  EZYV17845           Asia            Bachelor's                  N   
17983  EZYV17984           Asia            Bachelor's                  N   
20815  EZYV20816           Asia            Bachelor's                  N   
20984  EZYV20985         Europe             Doctorate                  Y   
21255  EZYV21256  North America           High School                  N   
21760  EZYV21761           Asia            Bachelor's                  Y   
21944  EZYV21945         Africa              Master's                  Y   
22084  EZYV22085  North America            Bachelor's                  Y   
22388  EZYV22389           Asia              Master's                  Y   
23186  EZYV23187           Asia              Master's                  N   
23476  EZYV23477         Europe              Master's                  Y   

      requires_job_training  no_of_employees  yr_of_estab  \
245                       N              -25         1980   
378                       Y              -11         2011   
832                       N              -17         2002   
2918                      N              -26         2005   
6439                      N              -14         2013   
6634                      N              -26         1923   
7224                      N              -25         1998   
7281                      N              -14         2000   
7318                      Y              -26         2006   
7761                      N              -11         2009   
9872                      N              -26         1996   
11493                     N              -14         1999   
13471                     N              -17         2003   
14022                     Y              -11         1946   
14146                     Y              -26         1954   
14726                     N              -11         2000   
15600                     N              -14         2014   
15859                     N              -11         1969   
16157                     N              -11         1994   
16883                     N              -26         1968   
17006                     N              -11         1984   
```

```
17655                    N              -17         2007
17844                    N              -14         2012
17983                    N              -26         2004
20815                    Y              -17         1990
20984                    N              -14         1989
21255                    N              -25         1987
21760                    N              -25         2000
21944                    N              -25         1977
22084                    N              -14         1980
22388                    N              -14         1986
23186                    Y              -11         2007
23476                    N              -11         2000

      region_of_employment  prevailing_wage unit_of_wage full_time_position
\
245              Northeast        39452.990         Year                  Y
378              Northeast        32506.140         Year                  Y
832                  South       129701.940         Year                  Y
2918               Midwest       112799.460         Year                  Y
6439                 South          103.970         Hour                  Y
6634                  West         5247.320         Year                  Y
7224               Midwest       141435.950         Year                  Y
7281               Midwest        58488.500         Year                  Y
7318                 South       115005.610         Year                  Y
7761               Midwest        38457.510         Year                  Y
9872                 South        37397.050         Year                  Y
11493                South        27599.350         Year                  Y
13471            Northeast          257.241         Hour                  Y
14022            Northeast       108403.560         Year                  Y
14146                 West        81982.270         Year                  Y
14726              Midwest       167851.800         Year                  Y
15600                South        24641.610         Year                  Y
15859                South        44640.600         Year                  Y
16157                South        62681.250         Year                  Y
16883            Northeast          168.156         Hour                  Y
17006                 West        25753.510         Year                  Y
17655            Northeast       129753.180         Year                  Y
17844                 West        29325.850         Year                  Y
17983                South        84359.980         Year                  Y
20815                 West        91897.570         Year                  Y
20984              Midwest        37012.800         Year                  Y
21255                South        99405.470         Year                  N
21760                 West       100463.580         Year                  Y
21944              Midwest        79150.510         Year                  Y
22084                 West          691.061         Hour                  Y
22388                South        17893.110         Year                  Y
23186              Midwest       120195.350         Year                  Y
23476                 West        95072.750         Year                  Y

      case_status
245     Certified
378        Denied
832     Certified
2918    Certified
6439       Denied
6634       Denied
7224    Certified
```

```
7281        Denied
7318     Certified
7761     Certified
9872     Certified
11493       Denied
13471       Denied
14022    Certified
14146    Certified
14726    Certified
15600       Denied
15859       Denied
16157    Certified
16883       Denied
17006       Denied
17655       Denied
17844       Denied
17983       Denied
20815    Certified
20984    Certified
21255       Denied
21760    Certified
21944    Certified
22084       Denied
22388    Certified
23186    Certified
23476       Denied
```

In [18]:

```
data['no_of_employees'] = data['no_of_employees'].abs()
updated_negative_rows = data['no_of_employees']
print(updated_negative_rows)
#once the negative vals are found, get the absolute value of them converting t
hem to positive vals
```

```
0        14513
1         2412
2        44444
3           98
4         1082
         ...
25475     2601
25476     3274
25477     1121
25478     1918
25479     3195
Name: no_of_employees, Length: 25480, dtype: int64
```

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

In [19]:
```
cat_col = data.select_dtypes(include='category').columns

for col in cat_col:
    print(f"Value distribution for '{col}':")
    print(data[col].value_counts(normalize=True))
    print("-" * 50)

#'continent': The majority of the data is from Asia (66.2%), with Europe and N
orth America having smaller proportions.
#'education_of_employee': The dataset is fairly balanced between Bachelor's (4
0.2%) and Master's (37.8%) degrees, with fewer employees holding a Doctorate
(8.6%).
#'has_job_experience': Most individuals have job experience (58.1%), with 41.
9% lacking experience.
#'requires_job_training': The majority (88.4%) do not require job training, wh
ile only 11.6% do.
#'region_of_employment': Employment is fairly distributed between the Northeas
t (28.2%), South (27.5%), and West (25.8%), with a small proportion in the Mid
west and Island regions.
#'unit_of_wage': Most employees are paid annually (90.1%), with a small fracti
on paid hourly or weekly.
#'full_time_position': The majority of employees hold full time positions (89.
4%).
#'case_status': The majority of cases are certified (66.8%), with a smaller pr
oportion being denied (33.2%).
```

```
Value distribution for 'case_id':
case_id
EZYV9999    0.000
EZYV01      0.000
EZYV02      0.000
EZYV03      0.000
EZYV04      0.000
             ...
EZYV10000   0.000
EZYV1000    0.000
EZYV100     0.000
EZYV10      0.000
EZYV09      0.000
Name: proportion, Length: 25480, dtype: float64
--------------------------------------------------
Value distribution for 'continent':
continent
Asia            0.662
Europe          0.146
North America   0.129
South America   0.033
Africa          0.022
Oceania         0.008
Name: proportion, dtype: float64
--------------------------------------------------
Value distribution for 'education_of_employee':
education_of_employee
Bachelor's    0.402
Master's      0.378
High School   0.134
Doctorate     0.086
Name: proportion, dtype: float64
--------------------------------------------------
Value distribution for 'has_job_experience':
has_job_experience
Y   0.581
N   0.419
Name: proportion, dtype: float64
--------------------------------------------------
Value distribution for 'requires_job_training':
requires_job_training
N   0.884
Y   0.116
Name: proportion, dtype: float64
--------------------------------------------------
Value distribution for 'region_of_employment':
region_of_employment
Northeast   0.282
South       0.275
West        0.258
Midwest     0.169
Island      0.015
Name: proportion, dtype: float64
--------------------------------------------------
Value distribution for 'unit_of_wage':
unit_of_wage
Year    0.901
```

```
Hour     0.085
Week     0.011
Month    0.003
Name: proportion, dtype: float64
--------------------------------------------------
Value distribution for 'full_time_position':
full_time_position
Y    0.894
N    0.106
Name: proportion, dtype: float64
--------------------------------------------------
Value distribution for 'case_status':
case_status
Certified    0.668
Denied       0.332
Name: proportion, dtype: float64
--------------------------------------------------
```

In [20]:
```
data2 = data.copy()
```

In [21]:
```
data2.drop(['case_id'],axis=1,inplace=True)
#no need for this col, doesnt add value when running models
```

In [22]:
```
data2['continent'].unique()
```

Out[22]:
```
['Asia', 'Africa', 'North America', 'Europe', 'South America', 'Oceania']
Categories (6, object): ['Africa', 'Asia', 'Europe', 'North America', 'Oceani
a', 'South America']
```

In [23]:
```
data2['region_of_employment'].unique()
```

Out[23]:
```
['West', 'Northeast', 'South', 'Midwest', 'Island']
Categories (5, object): ['Island', 'Midwest', 'Northeast', 'South', 'West']
```

In [24]:
```
data2.head()
```

Out[24]:

	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees
0	Asia	High School	N	N	14513
1	Asia	Master's	Y	N	2412
2	Asia	Bachelor's	N	Y	44444
3	Asia	Bachelor's	N	N	98
4	Africa	Master's	Y	N	1082

In [25]:
```
data2['education_of_employee'].unique() #'prevailing_wage', 'yr_of_estab', 'no
_of_employees', 'has_job_experience', 'requires_job_training', 'full_time_posi
tion', 'case_status', unit_of_wage
```

Out[25]:
```
['High School', 'Master's', 'Bachelor's', 'Doctorate']
Categories (4, object): ['Bachelor's', 'Doctorate', 'High School', 'Maste
r's']
```

In [26]:
```
data2['unit_of_wage'].unique()
```

Out[26]:
```
['Hour', 'Year', 'Week', 'Month']
Categories (4, object): ['Hour', 'Month', 'Week', 'Year']
```

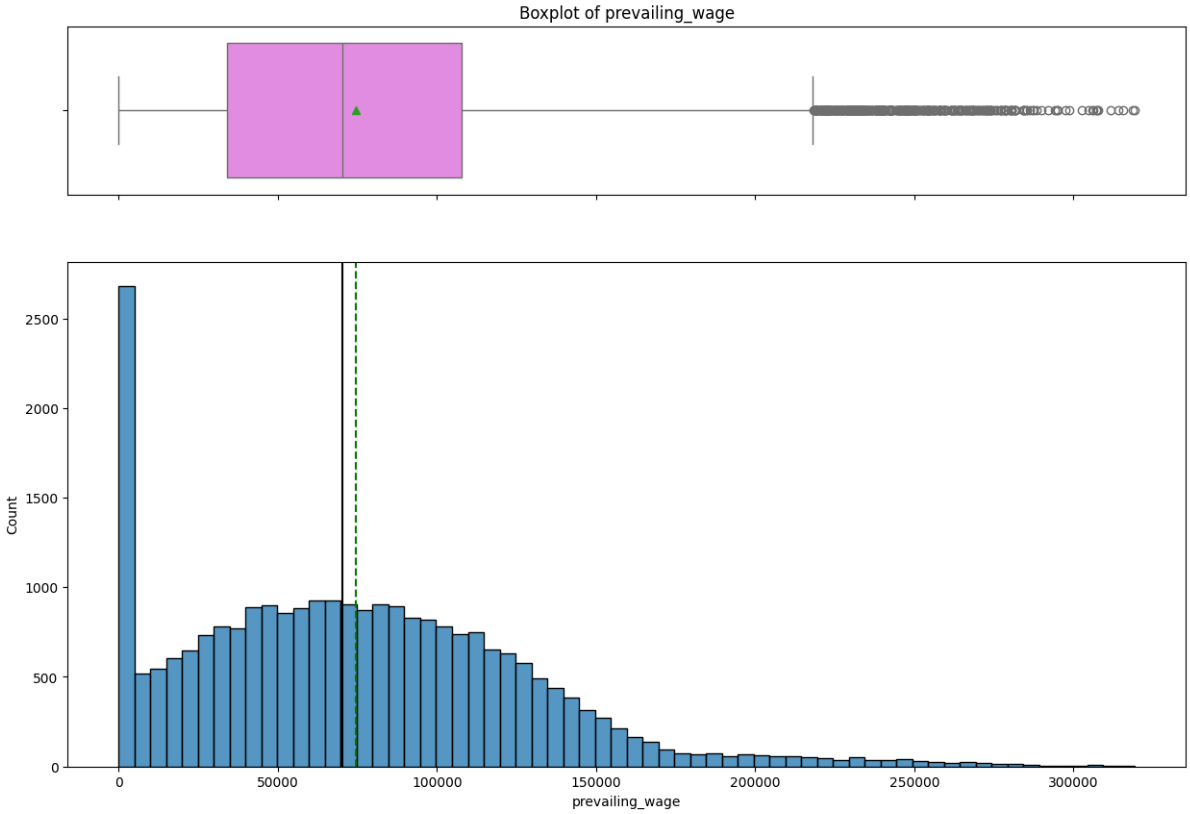
In [27]:
```
data2['education_of_employee'].replace({
    'High School': 1,
    "Bachelor's": 2,
    "Master's": 3,
    'Doctorate': 4
}, inplace=True)


data2['has_job_experience'].replace({'Y': 1, 'N': 0}, inplace=True)
data2['requires_job_training'].replace({'Y': 1, 'N': 0}, inplace=True)
data2['full_time_position'].replace({'Y': 1, 'N': 0}, inplace=True)
data2['case_status'].replace({'Certified': 1, 'Denied': 0}, inplace=True)


region_mapping = {
    'West': 0,
    'Northeast': 1,
    'South': 2,
    'Midwest': 3,
    'Island': 4
}
data2['region_of_employment'] = data2['region_of_employment'].replace(region_m
apping)


continent_mapping = {
    'North America': 0,
    'Europe': 1,
    'South America': 2,
    'Africa': 3,
    'Oceania': 4,
    'Asia': 5
}
data2['continent'] = data2['continent'].replace(continent_mapping)


wage_mapping = {
    'Hour': 1,
    'Month': 2,
    'Week': 3,
    'Year': 4
}
data2['unit_of_wage'] = data2['unit_of_wage'].replace(wage_mapping)

#converting all columns with string vals to one hot encoding for better consum
ption when running the models
```

Univariate Analysis

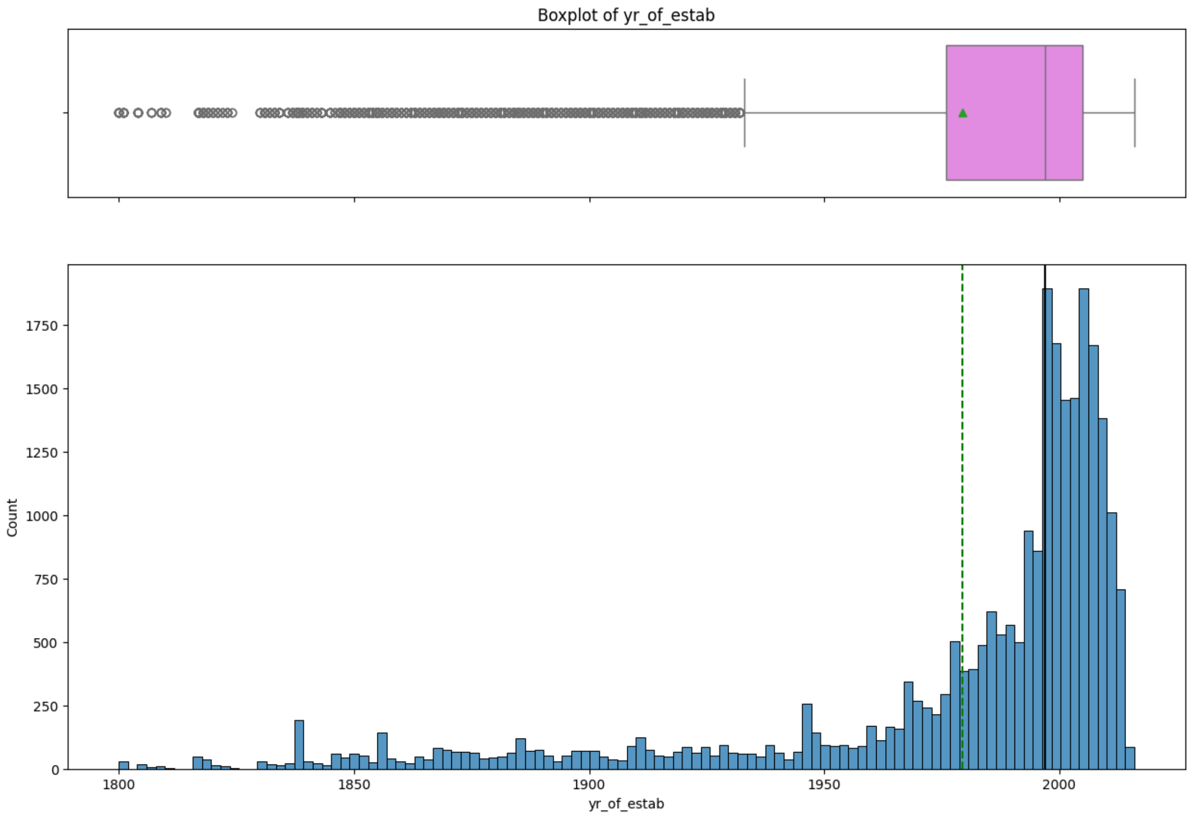
In [28]:
```
def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=None):
    """
    Boxplot and histogram combined

    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 o
f the column
    ax_box2.set_title(f'Boxplot of {feature}')
    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 [29]:
```
# 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(),
    )
    ax.set_title(f'Boxplot of {feature}')

    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
```

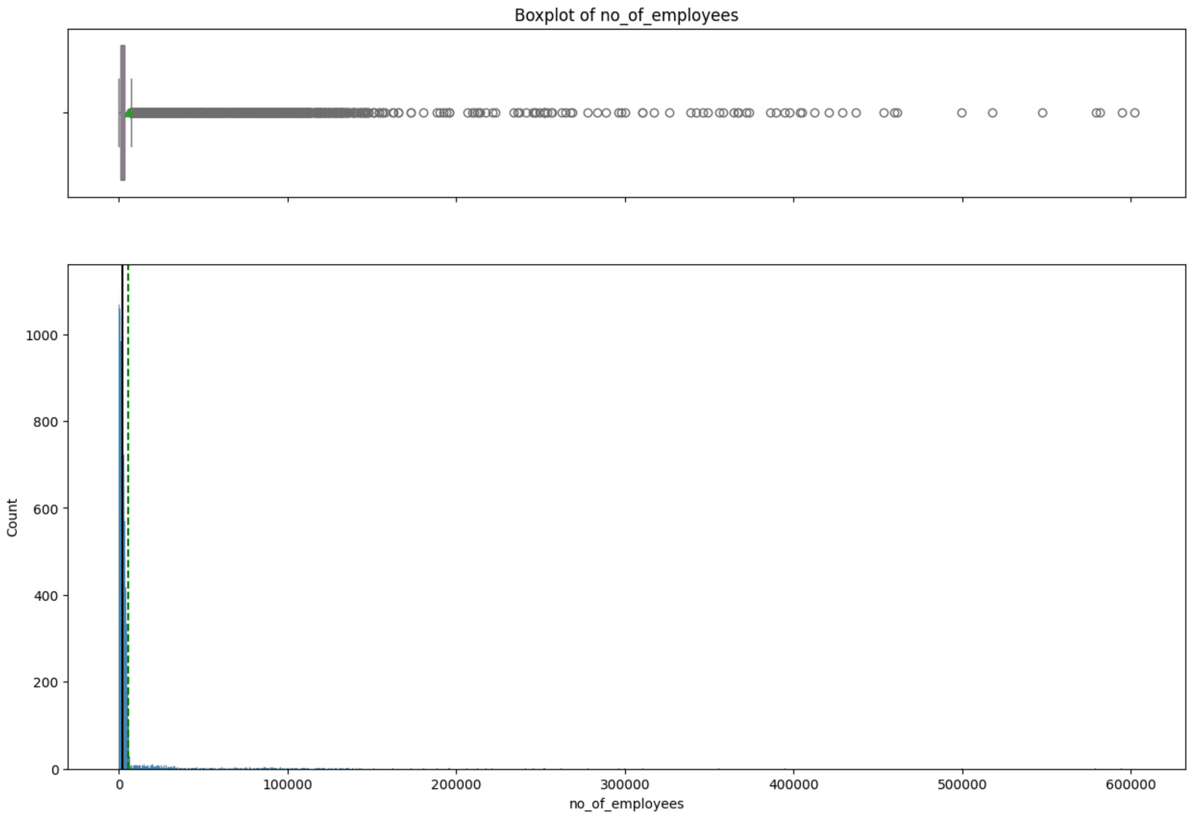
In [30]:
```
histogram_boxplot(data, 'prevailing_wage')
#majorty of people in the ds dont have a prevailing wage since they are new gr
ads, but there are a lot of outliers that are making a ton of money
#avg wage is around 70k or so
```



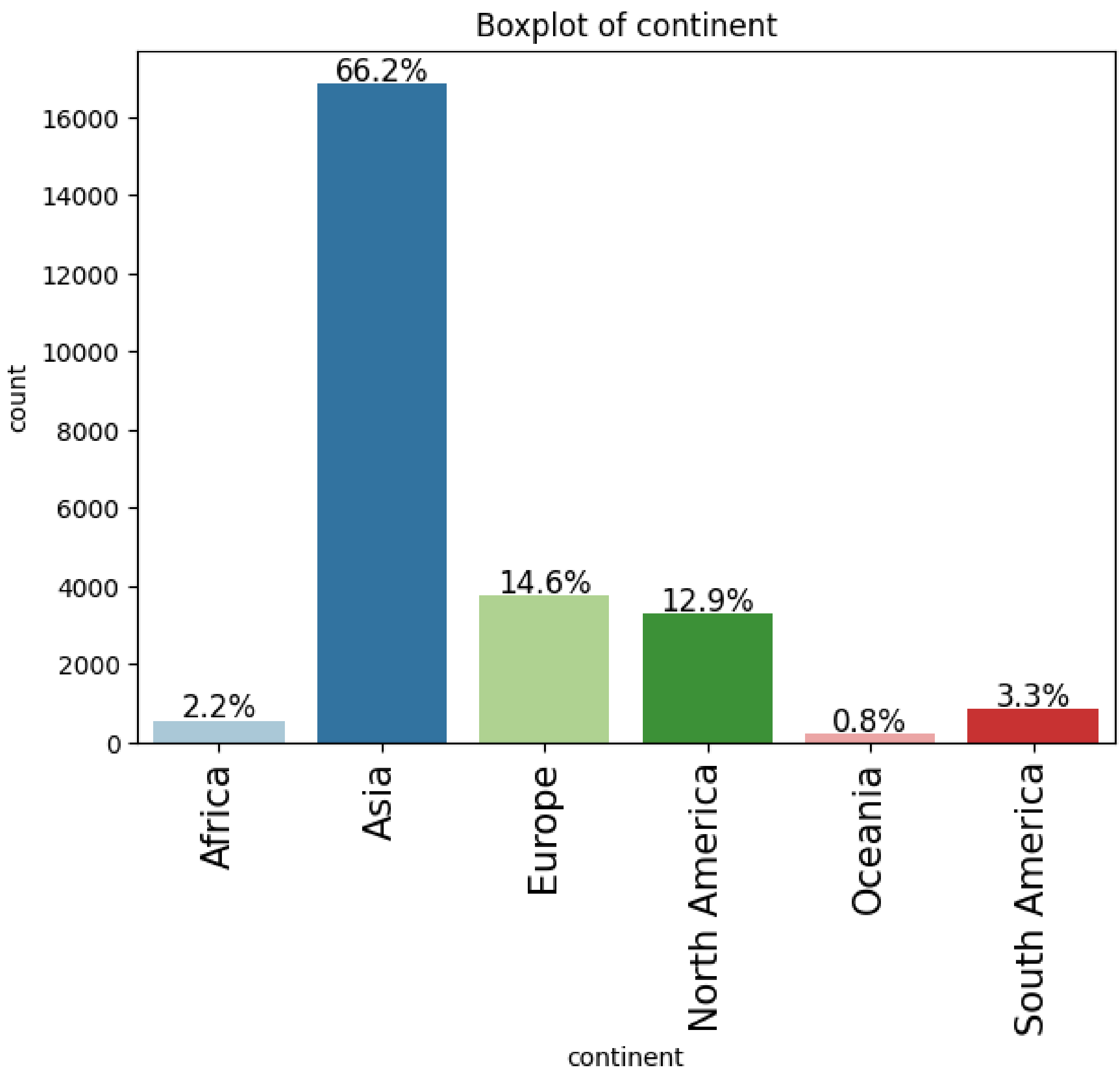
In [31]:
```
histogram_boxplot(data, 'yr_of_estab')
#majority of companies were established around 2000, dot com boom, there are a
ton of outliers but since this is yr the math might not be correct
```



In [32]:
```
histogram_boxplot(data, 'no_of_employees')
#a lot of companies are maybe only a few emps, but others are well established so they have more
```

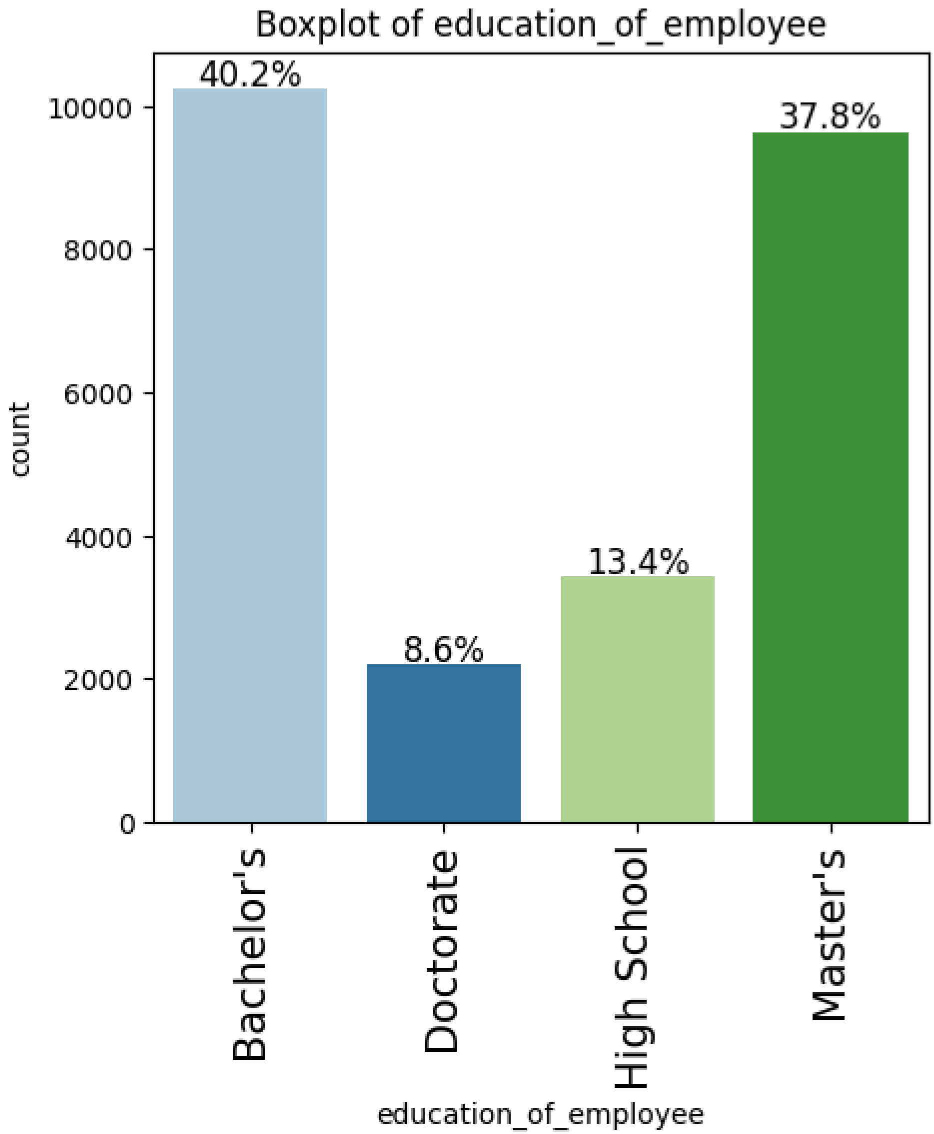


In [33]:
```
labeled_barplot(data, 'continent', perc = True)
#nearly 70% of workers are located in asia
```



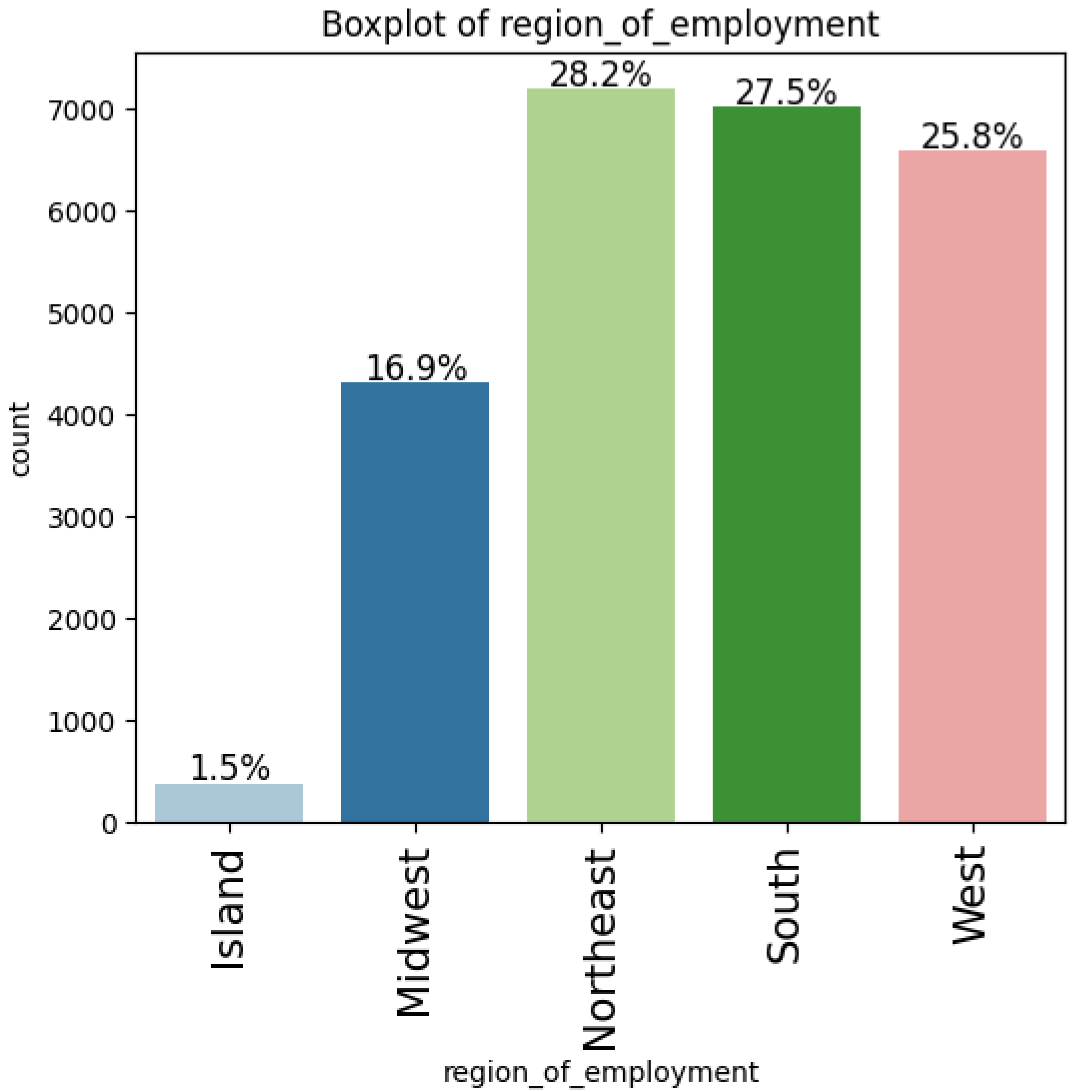
Observations on education of employee

In [34]:
```
labeled_barplot(data, 'education_of_employee', perc = True)
#clear split between new grads and BS and veterans and masters holders that ar
e applying for a case
```



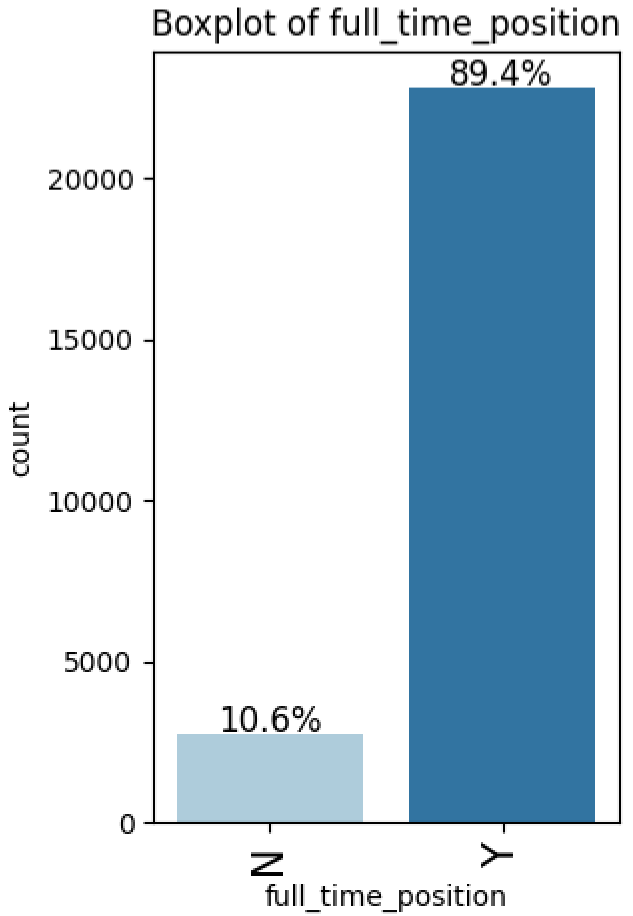
Observations on region of employment

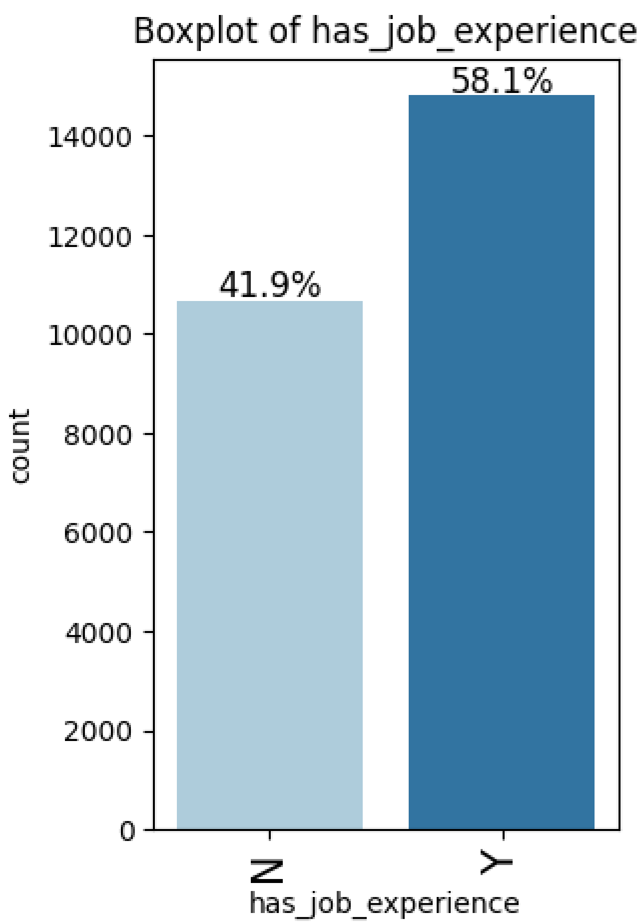
In [35]:
```
labeled_barplot(data, 'region_of_employment', perc = True)
#this map is a bit confusing since we dont know for sure which part of the wor
ld this is talking about
```

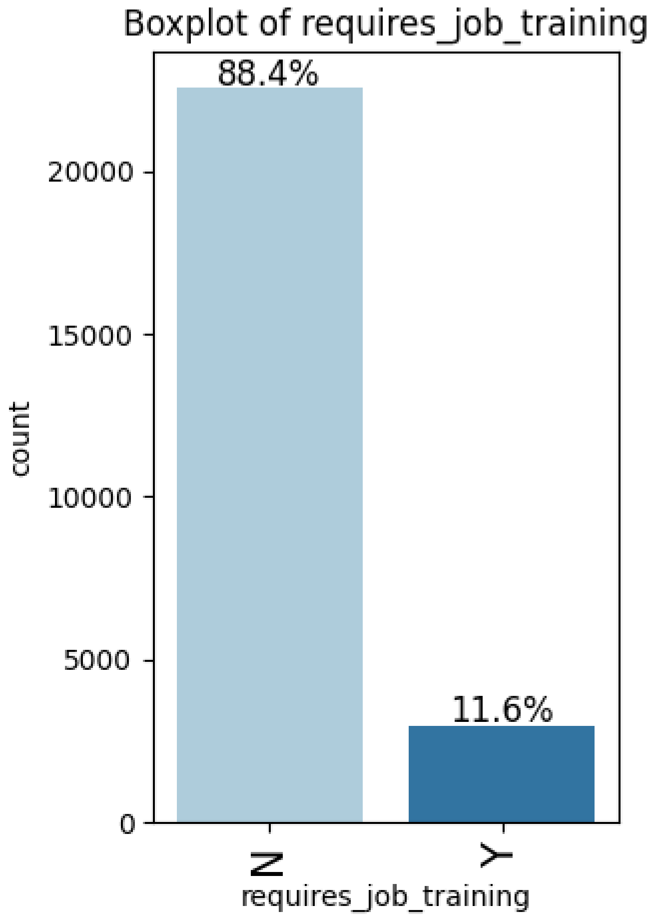


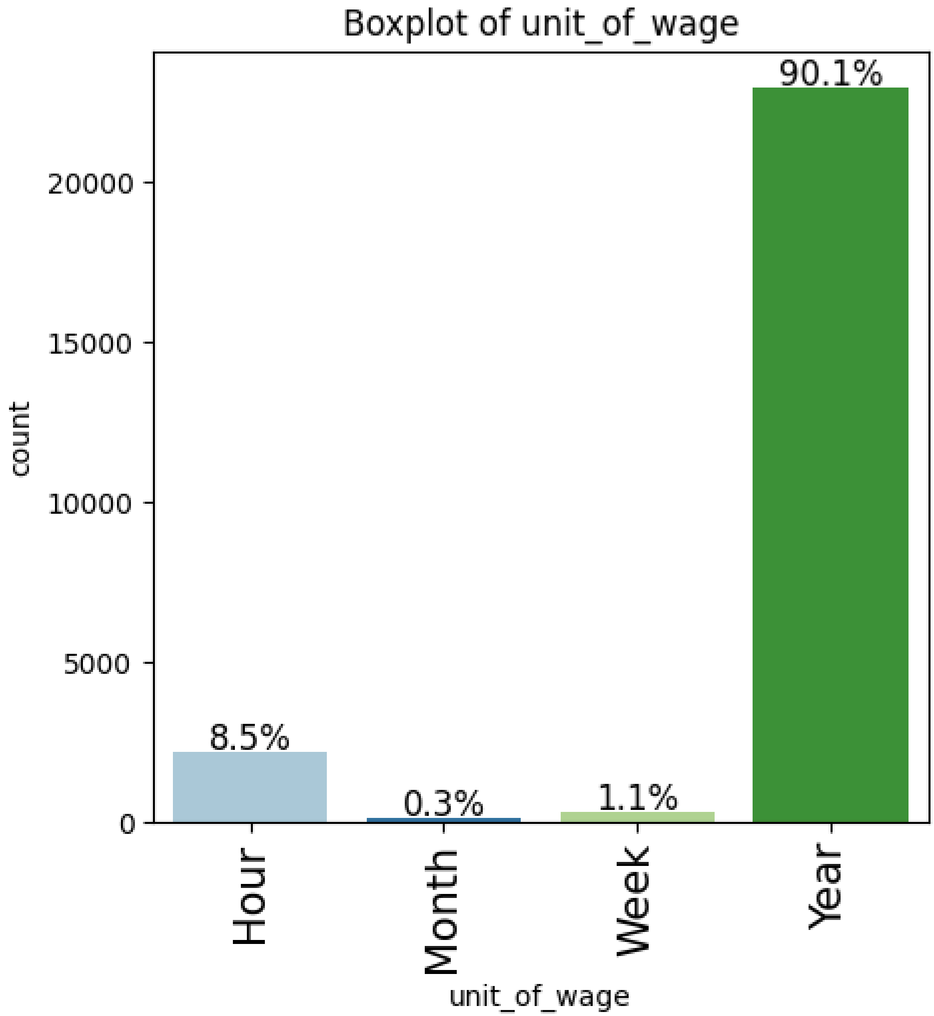
Observations on job experience

In [36]:
```
labeled_barplot(data, 'full_time_position', perc = True)
labeled_barplot(data, 'has_job_experience', perc = True)
labeled_barplot(data, 'requires_job_training', perc = True)
labeled_barplot(data, 'unit_of_wage', perc = True)

# 90% of workers have a full time job which is good for getting a case
# a little more than half of the ds has work exp, which means theyll get a cas
e
# 90% of workers dont need training which means that they have prior exp
# 90% of workers have jobs, which means they get a salary, but near 10% are po
ssibly contractors
```


Boxplot of full_time_position
89.4%
10.6%
count
20000
15000
10000
5000
0
N
Y
full_time_position

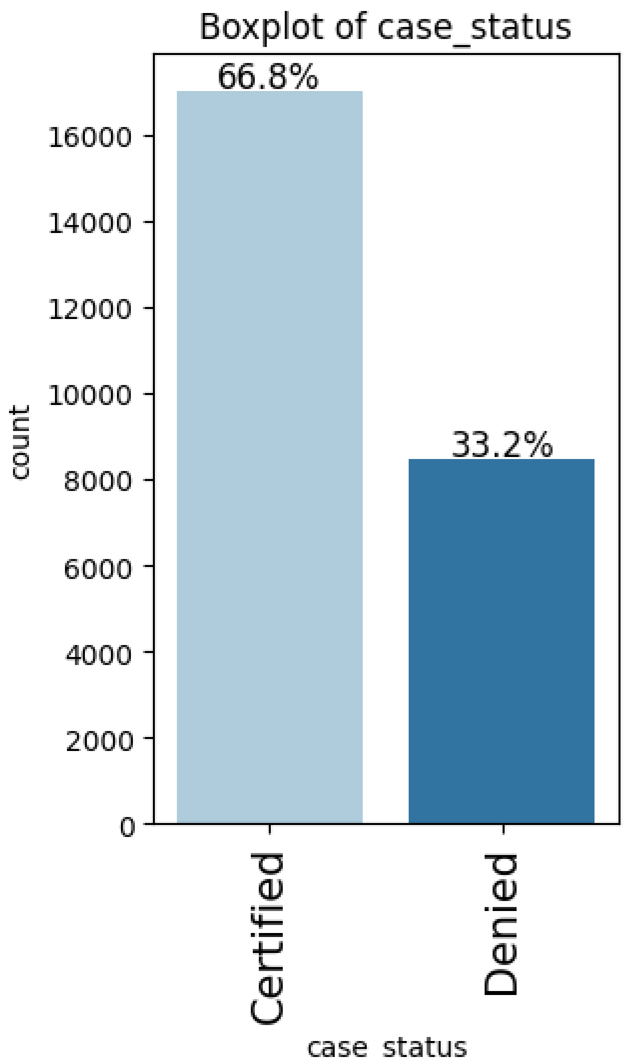

Boxplot of has_job_experience
58.1%
41.9%
count
14000
12000
10000
8000
6000
4000
2000
0
N
Y
has_job_experience


Boxplot of requires_job_training
88.4%
11.6%
count
20000
15000
10000
5000
0
N
Y
requires_job_training



Observations on case status

In [37]:
```
labeled_barplot(data, 'case_status', perc = True)
# 2/3 of workers are getting their case certified
```



Bivariate Analysis

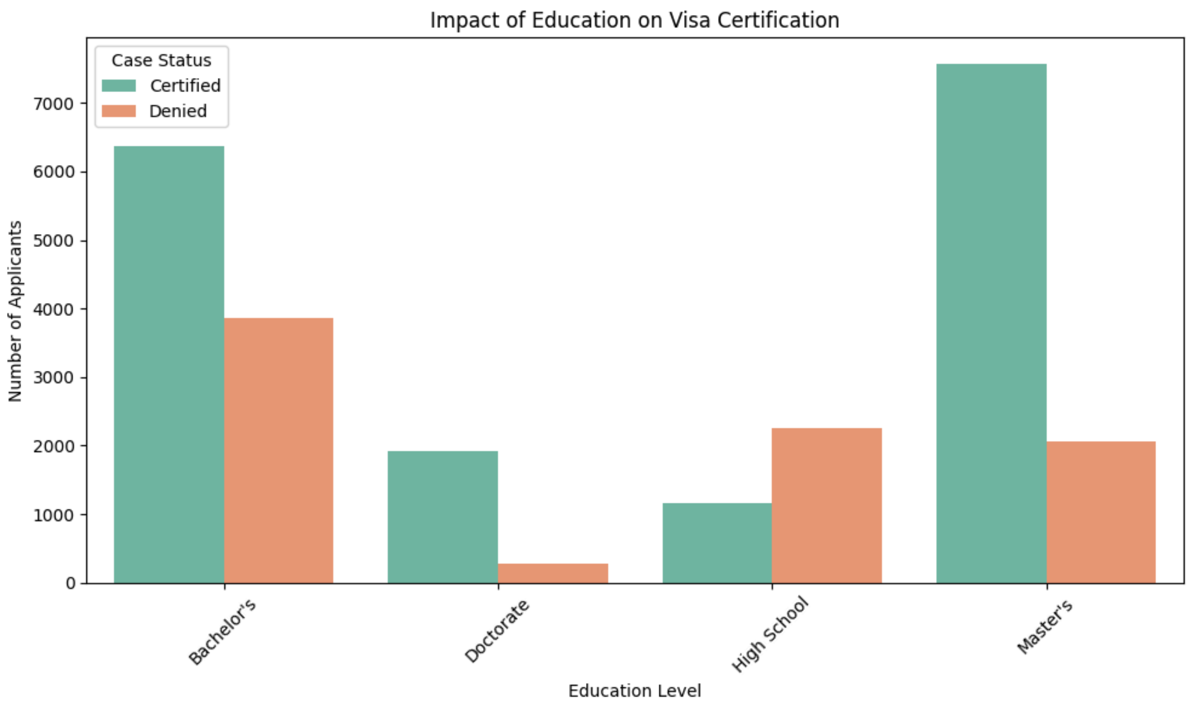
Creating functions that will help us with further analysis.

In [38]:
```
### 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 [39]:
```
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_value
s(
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_v
alues(
        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()
```

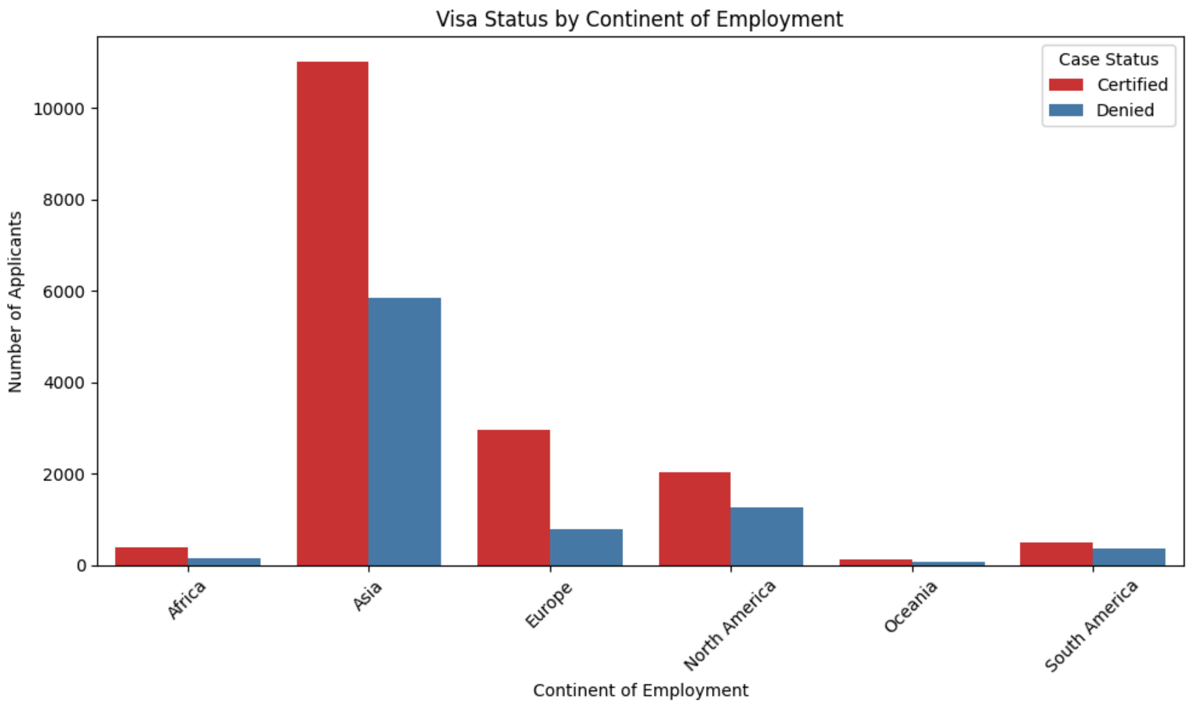
Those with higher education may want to travel abroad for a well-paid job. Let's find out if education has any impact on visa certification

In [40]:
```
plt.figure(figsize=(10, 6))
sns.countplot(data=data, x='education_of_employee', hue='case_status', palette
='Set2')
plt.title('Impact of Education on Visa Certification')
plt.xlabel('Education Level')
plt.ylabel('Number of Applicants')
plt.legend(title='Case Status')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# a majority of workers that have either a beginner education or a seasoned ed
ucation are getting their cases certified, HS workers probably wont get it
# good portion of phd folks are getting certified
```



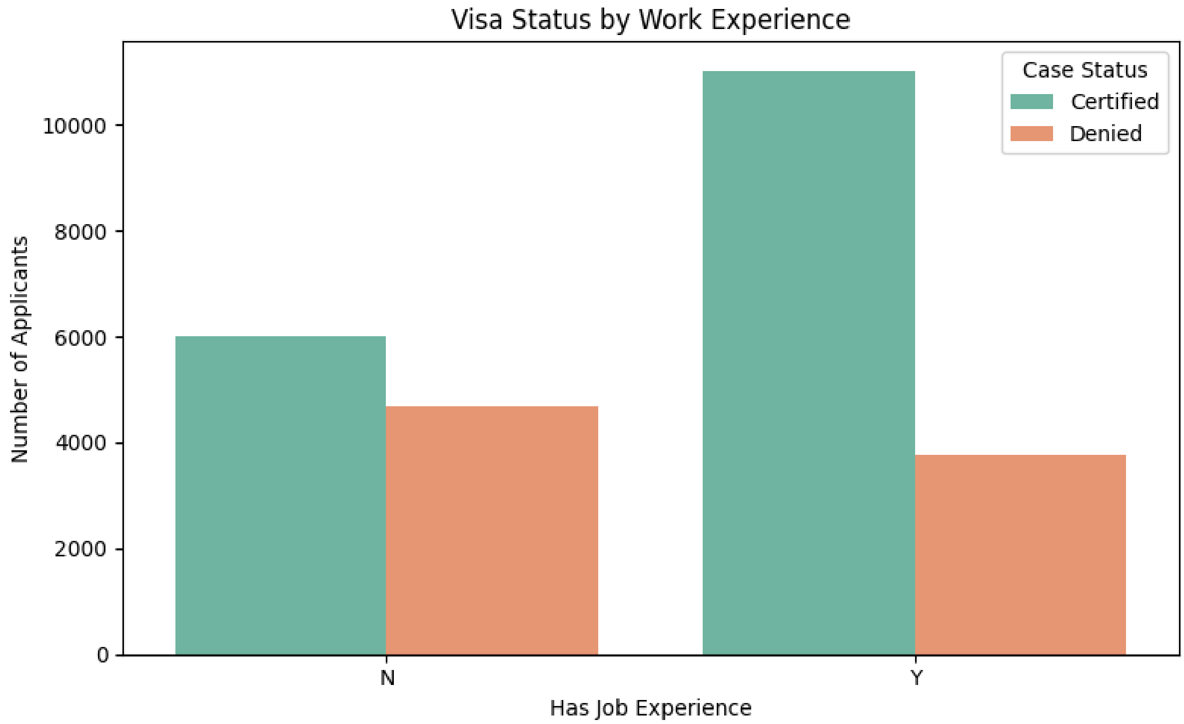
Lets' similarly check for the continents and find out how the visa status vary across different continents.

In [41]:
```
plt.figure(figsize=(10, 6))
sns.countplot(data=data, x='continent', hue='case_status', palette='Set1')
plt.title('Visa Status by Continent of Employment')
plt.xlabel('Continent of Employment')
plt.ylabel('Number of Applicants')
plt.legend(title='Case Status')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# mainly only folks in asia are getting certified for cases, this seems that t
hey are qualified and better than the rest of the continents
```



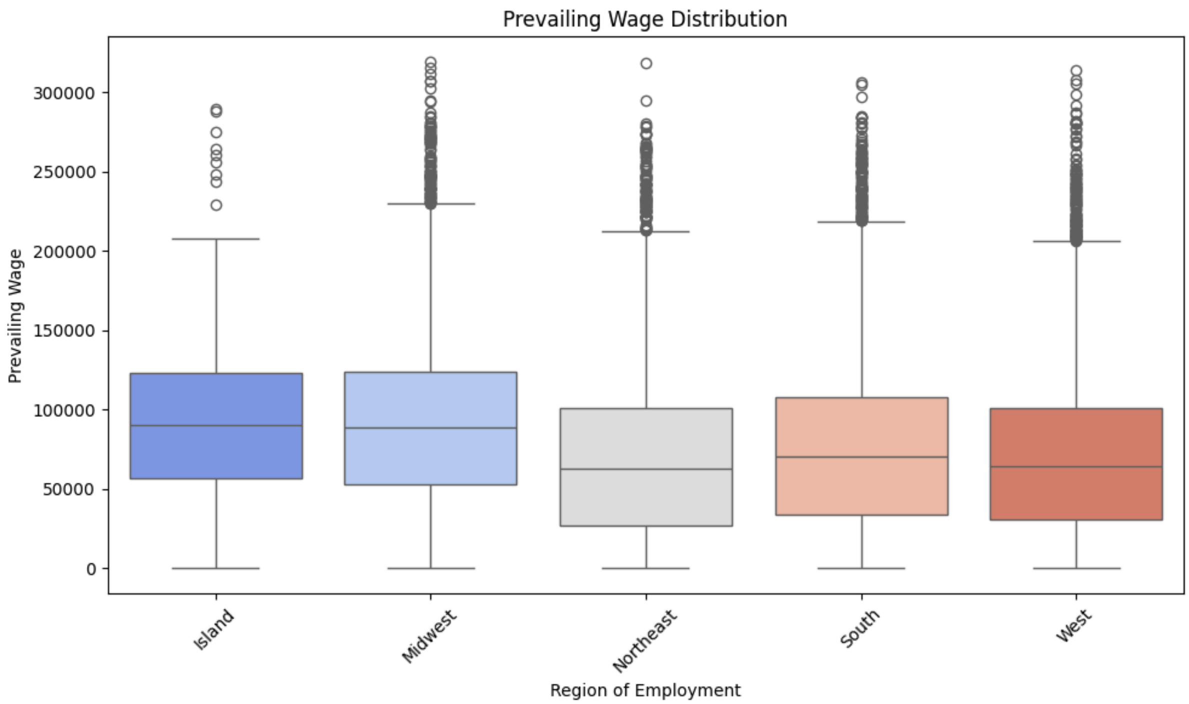
Experienced professionals might look abroad for opportunities to improve their lifestyles and career development. Let's see if having work experience has any influence over visa certification

In [42]:
```
plt.figure(figsize=(8, 5))
sns.countplot(data=data, x='has_job_experience', hue='case_status', palette='Set2')
plt.title('Visa Status by Work Experience')
plt.xlabel('Has Job Experience')
plt.ylabel('Number of Applicants')
plt.legend(title='Case Status')
plt.tight_layout()
plt.show()

# if workers have job exp then they are more likely to get a visa, which makes sense, we need workers
```



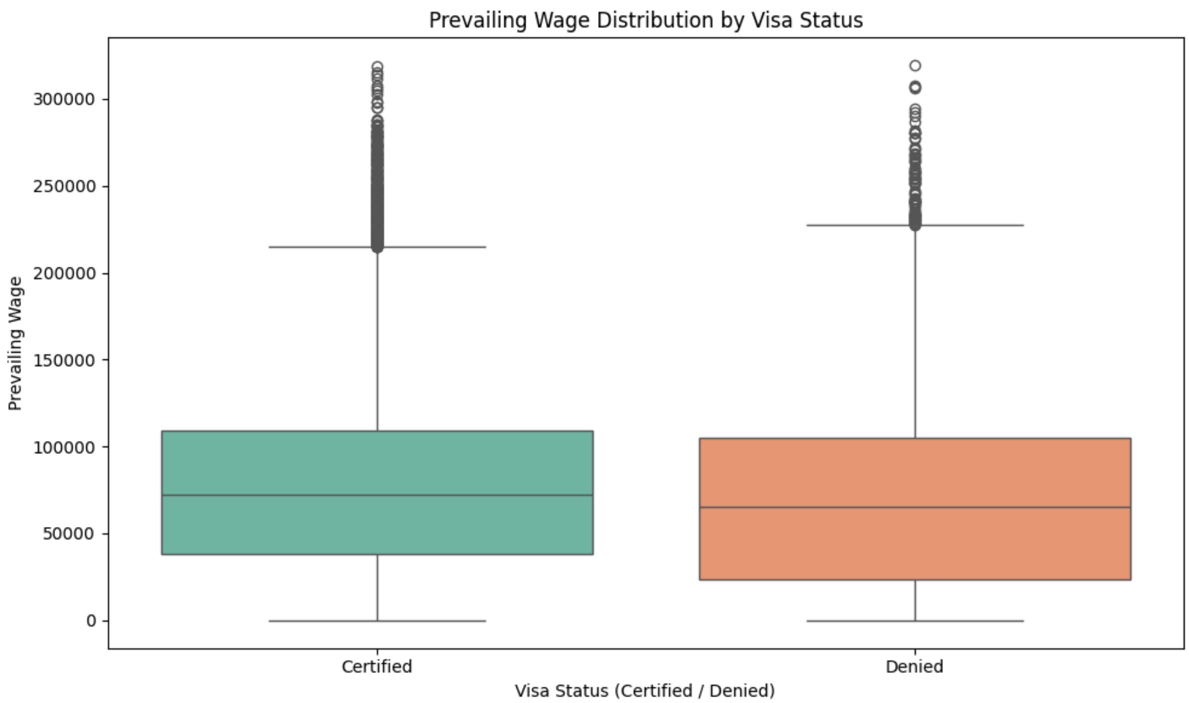
Checking if the prevailing wage is similar across all the regions of the US

In [43]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='region_of_employment', y='prevailing_wage', palette
='coolwarm')
plt.title('Prevailing Wage Distribution')
plt.xlabel('Region of Employment')
plt.ylabel('Prevailing Wage')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

# the midwest region has a lot of outliers in the higher end of wages, but nam
ing scheme is a bit tough to understand which part of world exactly
```



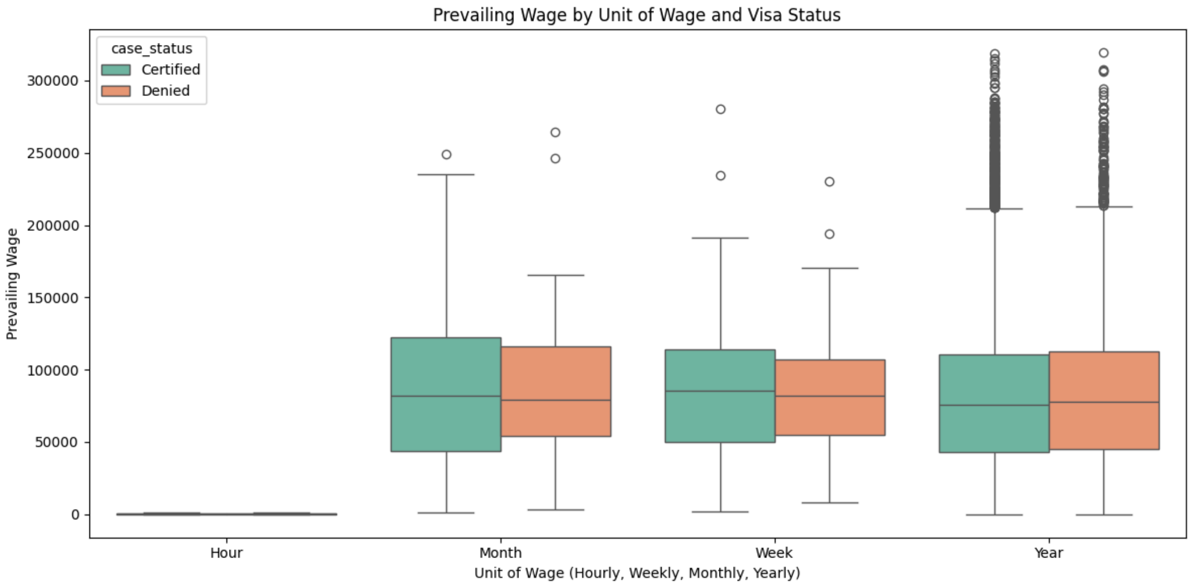
The US government has established a prevailing wage to protect local talent and foreign workers. Let's analyze the data and see if the visa status changes with the prevailing wage

In [44]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(data=data, x='case_status', y='prevailing_wage', palette='Set2')
plt.title('Prevailing Wage Distribution by Visa Status')
plt.xlabel('Visa Status (Certified / Denied)')
plt.ylabel('Prevailing Wage')
plt.tight_layout()
plt.show()

# there is a higher baseline for wages for people that have visas and they hav
e more outliers, 50% of them make slightly more than the denied folks
```

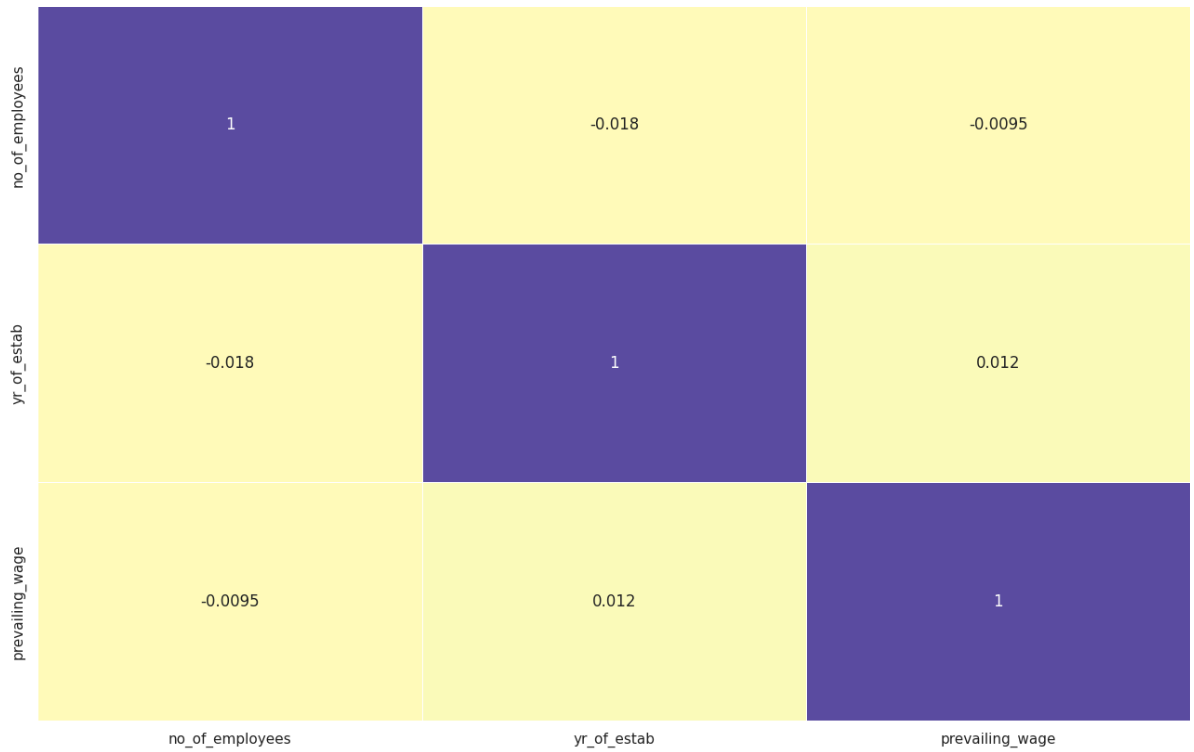


The prevailing wage has different units (Hourly, Weekly, etc). Let's find out if it has any impact on visa applications getting certified.

In [45]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(data=data, x='unit_of_wage', y='prevailing_wage', hue='case_status', palette='Set2')
plt.title('Prevailing Wage by Unit of Wage and Visa Status')
plt.xlabel('Unit of Wage (Hourly, Weekly, Monthly, Yearly)')
plt.ylabel('Prevailing Wage')
plt.tight_layout()
plt.show()

# hourly wages dont exist basically, yearly wages is split essentially since half are certified and other half is denied
```



In [46]:
```
sns.set(rc={'figure.figsize': (16, 10)})

sns.heatmap(
    data.corr(numeric_only=True),
    annot=True,
    linewidths=0.5,
    center=0,
    cbar=False,
    cmap="Spectral"
)
plt.show()
```

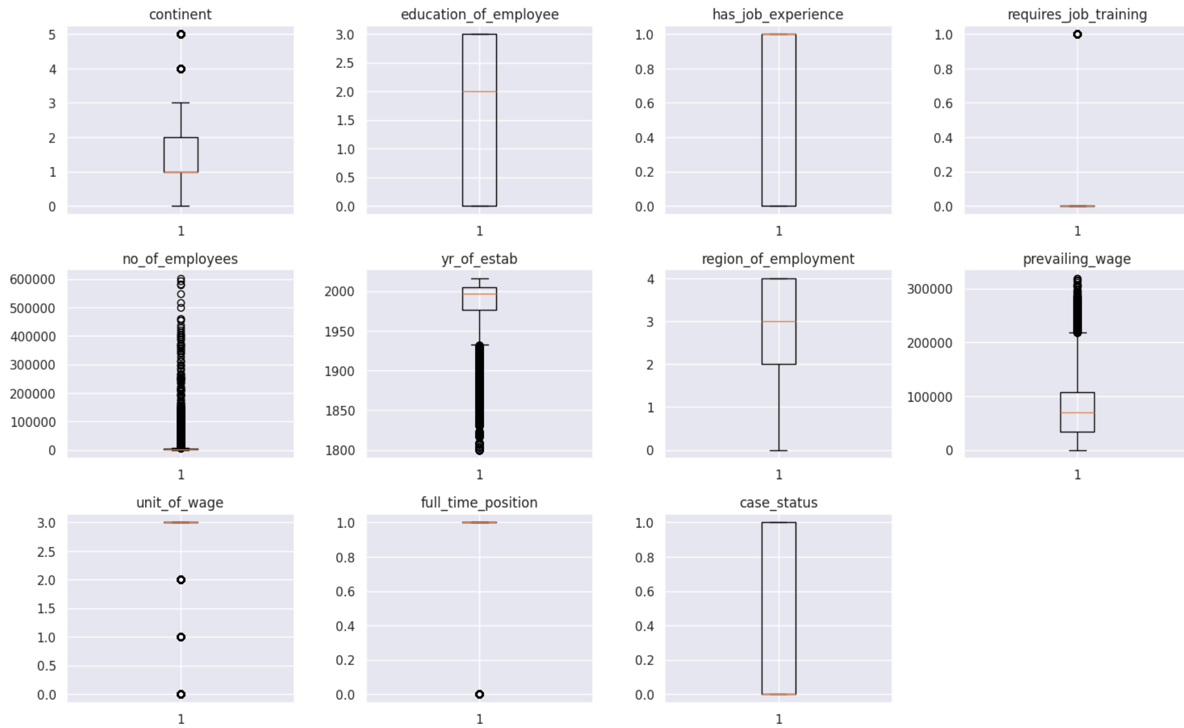


Data Pre-processing

Outlier Check

In [47]:
```
data_encoded = data2.copy()

for col in data_encoded.select_dtypes(include='category').columns:
    data_encoded[col] = data_encoded[col].cat.codes

numeric_columns = data_encoded.select_dtypes(include=np.number).columns.tolist
()

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

for i, variable in enumerate(numeric_columns):
    plt.subplot(4, 4, i + 1)
```

```
    plt.boxplot(data_encoded[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)

plt.show()
```



Data Preparation for modeling

In [48]:
```
X = data2.drop(["case_status"], axis=1)
y = data2["case_status"]

# dropping the target variable
```

In [49]:
```
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=0.3, random_state=1, stratify=y
)

# then we split the temporary set into train and validation

X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.25, random_state=1, stratify=y_temp
)
print(X_train.shape, X_val.shape, X_test.shape)

# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, ra
ndom_state=1, shuffle=True)
```

```
(13377, 10) (4459, 10) (7644, 10)
```

In [50]:
```
X_train.shape, X_val.shape
```

Out[50]:
```
((13377, 10), (4459, 10))
```

In [51]:
```
import pandas as pd
from sklearn.impute import SimpleImputer

# Get list of categorical and numerical columns
cat_cols = list(X_train.select_dtypes(include='category').columns)
num_cols = list(X_train.select_dtypes(include=['int', 'float']).columns)

# Impute categorical columns
cat_imputer = SimpleImputer(strategy='most_frequent')
X_train[cat_cols] = cat_imputer.fit_transform(X_train[cat_cols])
X_val[cat_cols] = cat_imputer.transform(X_val[cat_cols])
X_test[cat_cols] = cat_imputer.transform(X_test[cat_cols])

# Impute numerical columns
num_imputer = SimpleImputer(strategy='mean')
X_train[num_cols] = num_imputer.fit_transform(X_train[num_cols])
X_val[num_cols] = num_imputer.transform(X_val[num_cols])
X_test[num_cols] = num_imputer.transform(X_test[num_cols])
data2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25480 entries, 0 to 25479
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype   
---  ------                 --------------  -----   
 0   continent              25480 non-null  category
 1   education_of_employee  25480 non-null  category
 2   has_job_experience     25480 non-null  category
 3   requires_job_training  25480 non-null  category
 4   no_of_employees        25480 non-null  int64   
 5   yr_of_estab            25480 non-null  int64   
 6   region_of_employment   25480 non-null  category
 7   prevailing_wage        25480 non-null  float64 
 8   unit_of_wage           25480 non-null  category
 9   full_time_position     25480 non-null  category
 10  case_status            25480 non-null  category
dtypes: category(8), float64(1), int64(2)
memory usage: 797.7 KB
```

In [52]:
```
# Checking that no column has missing values in train, validation or test sets
print(X_train.isna().sum())
print("-" * 30)
print(X_val.isna().sum())
print("-" * 30)
print(X_test.isna().sum())
```

```
continent                0
education_of_employee    0
has_job_experience       0
requires_job_training    0
no_of_employees          0
yr_of_estab              0
region_of_employment     0
prevailing_wage          0
unit_of_wage             0
full_time_position       0
dtype: int64
------------------------------
continent                0
education_of_employee    0
has_job_experience       0
requires_job_training    0
no_of_employees          0
yr_of_estab              0
region_of_employment     0
prevailing_wage          0
unit_of_wage             0
full_time_position       0
dtype: int64
------------------------------
continent                0
education_of_employee    0
has_job_experience       0
requires_job_training    0
no_of_employees          0
yr_of_estab              0
region_of_employment     0
prevailing_wage          0
unit_of_wage             0
full_time_position       0
dtype: int64
```

In [53]:
```
data2.isnull().sum()
```

Out[53]:

	0
continent	0
education_of_employee	0
has_job_experience	0
requires_job_training	0
no_of_employees	0
yr_of_estab	0
region_of_employment	0
prevailing_wage	0
unit_of_wage	0
full_time_position	0
case_status	0

dtype: int64

Model Building

Model Evaluation Criterion

In [54]:
```
# defining a function to compute different metrics to check performance of a c
lassification model built using sklearn


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

    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": f
1,},
        index=[0],
    )

    return df_perf
```

In [55]:
```
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

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

Model building with original data

In [56]:
```
models = []

models.append(("Bagging", BaggingClassifier(estimator=DecisionTreeClassifier(r
andom_state=1, class_weight='balanced'), random_state=1)))
models.append(("Random forest", RandomForestClassifier(random_state=1, class_w
eight='balanced')))
models.append(("GBM", GradientBoostingClassifier(random_state=1)))
models.append(("Adaboost", AdaBoostClassifier(random_state=1)))
models.append(("dtree", DecisionTreeClassifier(random_state=1, class_weight='b
alanced')))

print("\nTraining Performance:\n")
for name, model in models:
    model.fit(X_train, y_train)
    scores = recall_score(y_train, model.predict(X_train))
    print("{}: {}".format(name, scores))

print("\nValidation Performance:\n")
for name, model in models:
    model.fit(X_train, y_train)
    scores_val = recall_score(y_val, model.predict(X_val))
    print("{}: {}".format(name, scores_val))

# adding the 5 models and running the training and validation tests on it, thi
s is right out of the gate before anything is done to it
```

```
Training Performance:

Bagging: 0.9863458310016788
Random forest: 1.0
GBM: 0.8846110800223839
Adaboost: 0.8878567431449357
dtree: 1.0

Validation Performance:

Bagging: 0.7991940899932841
Random forest: 0.8707186030893217
GBM: 0.8935527199462726
Adaboost: 0.8989254533243788
dtree: 0.7635997313633311
```

In [57]:
```
print("\nTraining and Validation Performance Difference:\n")

for name, model in models:
    model.fit(X_train, y_train)
    scores_train = recall_score(y_train, model.predict(X_train))
    scores_val = recall_score(y_val, model.predict(X_val))
    difference1 = scores_train - scores_val
    print("{}: Training Score: {:.4f}, Validation Score: {:.4f}, Difference: 
{:.4f}".format(name, scores_train, scores_val, difference1))

# getting the diff between the 2 tests to see how they stack up against each o
ther and also to see room for improvement.
```

```
Training and Validation Performance Difference:

Bagging: Training Score: 0.9863, Validation Score: 0.7992, Difference: 0.1872
Random forest: Training Score: 1.0000, Validation Score: 0.8707, Difference: 
0.1293
GBM: Training Score: 0.8846, Validation Score: 0.8936, Difference: -0.0089
Adaboost: Training Score: 0.8879, Validation Score: 0.8989, Difference: -0.01
11
dtree: Training Score: 1.0000, Validation Score: 0.7636, Difference: 0.2364
```

Model Building with oversampled data

In [58]:
```
print("Before Oversampling, counts of label 'Yes': {}".format(sum(y_train ==
1)))
print("Before Oversampling, counts of label 'No': {} \n".format(sum(y_train ==
0)))

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


print("After Oversampling, counts of label 'Yes': {}".format(sum(y_train_over
== 1)))
print("After Oversampling, counts of label 'No': {} \n".format(sum(y_train_ove
r == 0)))


print("After Oversampling, the shape of train_X: {}".format(X_train_over.shap
e))
print("After Oversampling, the shape of train_y: {} \n".format(y_train_over.sh
ape))

# running SMOTE on the dataset for over/under sampling to see if the ds change
s at all, only the over changes for NO after running it which means some folks
were on the border
```

```
Before Oversampling, counts of label 'Yes': 8935
Before Oversampling, counts of label 'No': 4442

After Oversampling, counts of label 'Yes': 8935
After Oversampling, counts of label 'No': 8935

After Oversampling, the shape of train_X: (17870, 10)
After Oversampling, the shape of train_y: (17870,)
```

In [59]:
```
X_train_over.isnull().sum()
```

Out[59]:

	0
continent	0
education_of_employee	0
has_job_experience	0
requires_job_training	0
no_of_employees	0
yr_of_estab	0
region_of_employment	0
prevailing_wage	0
unit_of_wage	0
full_time_position	0

dtype: int64

In [60]:
```
models = []  # Empty list to store all the models

# Appending models into the list
models.append(("Bagging", BaggingClassifier(estimator=DecisionTreeClassifier(random_state=1, class_weight='balanced'), random_state=1)))
models.append(("Random forest", RandomForestClassifier(random_state=1, class_weight='balanced')))
models.append(("GBM", GradientBoostingClassifier(random_state=1)))
models.append(("Adaboost", AdaBoostClassifier(random_state=1)))
models.append(("dtree", DecisionTreeClassifier(random_state=1, class_weight='balanced')))

print("\n" "Training Performance:" "\n")
for name, model in models:
    model.fit(X_train_over, y_train_over)
    scores = recall_score(y_train_over, model.predict(X_train_over))
    print("{}: {}".format(name, scores))

print("\n" "Validation Performance:" "\n")

for name, model in models:
    model.fit(X_train_over, y_train_over)
    scores = recall_score(y_val, model.predict(X_val))
    print("{}: {}".format(name, scores))

# running the validation tests on the OG data to see how the data reacts
# GBM and Ada models did well and increased their numbers which means they generalized well
```

```
Training Performance:

Bagging: 0.9748181309457191
Random forest: 1.0
GBM: 0.7898153329602686
Adaboost: 0.7838836038052602
dtree: 1.0

Validation Performance:

Bagging: 0.7374076561450638
Random forest: 0.7971793149764943
GBM: 0.7978509066487576
Adaboost: 0.7931497649429147
dtree: 0.708865010073875
```

In [61]:
```
print("\nTraining and Validation Performance Difference:\n")

for name, model in models:
    model.fit(X_train_over, y_train_over)
    scores_train = recall_score(y_train_over, model.predict(X_train_over))
    scores_val = recall_score(y_val, model.predict(X_val))
    difference2 = scores_train - scores_val
    print("{}: Training Score: {:.4f}, Validation Score: {:.4f}, Difference: {:.4f}".format(name, scores_train, scores_val, difference2))
```

```
Training and Validation Performance Difference:

Bagging: Training Score: 0.9748, Validation Score: 0.7374, Difference: 0.2374
Random forest: Training Score: 1.0000, Validation Score: 0.7972, Difference: 0.2028
GBM: Training Score: 0.7898, Validation Score: 0.7979, Difference: -0.0080
Adaboost: Training Score: 0.7839, Validation Score: 0.7931, Difference: -0.0093
dtree: Training Score: 1.0000, Validation Score: 0.7089, Difference: 0.2911
```

Model Building with undersampled data

In [62]:
```
rus = RandomUnderSampler(random_state=1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)
```

In [63]:
```
print("Before Under Sampling, counts of label 'Yes': {}".format(sum(y_train == 1)))
print("Before Under Sampling, counts of label 'No': {} \n".format(sum(y_train == 0)))

print("After Under Sampling, counts of label 'Yes': {}".format(sum(y_train_un == 1)))
print("After Under Sampling, counts of label 'No': {} \n".format(sum(y_train_un == 0)))

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

```
Before Under Sampling, counts of label 'Yes': 8935
Before Under Sampling, counts of label 'No': 4442

After Under Sampling, counts of label 'Yes': 4442
After Under Sampling, counts of label 'No': 4442

After Under Sampling, the shape of train_X: (8884, 10)
After Under Sampling, the shape of train_y: (8884,)
```

In [64]:
```
models = []  # Empty list to store all the models

# Appending models into the list
models.append(("Bagging", BaggingClassifier(estimator=DecisionTreeClassifier(r
andom_state=1, class_weight='balanced'), random_state=1)))
models.append(("Random forest", RandomForestClassifier(random_state=1, class_w
eight='balanced')))
models.append(("GBM", GradientBoostingClassifier(random_state=1)))
models.append(("Adaboost", AdaBoostClassifier(random_state=1)))
models.append(("dtree", DecisionTreeClassifier(random_state=1, class_weight='b
alanced')))


print("\n" "Training Performance:" "\n")
for name, model in models:
    model.fit(X_train_un, y_train_un)
    scores = recall_score(y_train_un, model.predict(X_train_un))
    print("{}: {}".format(name, scores))

print("\n" "Validation Performance:" "\n")

for name, model in models:
    model.fit(X_train_un, y_train_un)
    scores = recall_score(y_val, model.predict(X_val))
    print("{}: {}".format(name, scores))

# The high training performance scores could indicate overfitting, where the m
odel is too closely aligned with the training data and might not generalize we
ll
# Models like GBM and Random Forest show relatively good validation performanc
e, which suggests they generalize better than others
```

```
Training Performance:

Bagging: 0.9669067987393066
Random forest: 1.0
GBM: 0.7478613237280505
Adaboost: 0.7014858171994597
dtree: 1.0

Validation Performance:

Bagging: 0.6229012760241773
Random forest: 0.7081934184016119
GBM: 0.7437877770315648
Adaboost: 0.6984553391537945
dtree: 0.6437206178643384
```

In [65]:
```
print("\nTraining and Validation Performance Difference:\n")

for name, model in models:
    model.fit(X_train_un, y_train_un)
    scores_train = recall_score(y_train_un, model.predict(X_train_un))
    scores_val = recall_score(y_val, model.predict(X_val))
    difference3 = scores_train - scores_val
    print("{}: Training Score: {:.4f}, Validation Score: {:.4f}, Difference:
{:.4f}".format(name, scores_train, scores_val, difference3))
```

```
Training and Validation Performance Difference:

Bagging: Training Score: 0.9669, Validation Score: 0.6229, Difference: 0.3440
Random forest: Training Score: 1.0000, Validation Score: 0.7082, Difference:
0.2918
GBM: Training Score: 0.7479, Validation Score: 0.7438, Difference: 0.0041
Adaboost: Training Score: 0.7015, Validation Score: 0.6985, Difference: 0.003
0
dtree: Training Score: 1.0000, Validation Score: 0.6437, Difference: 0.3563
```

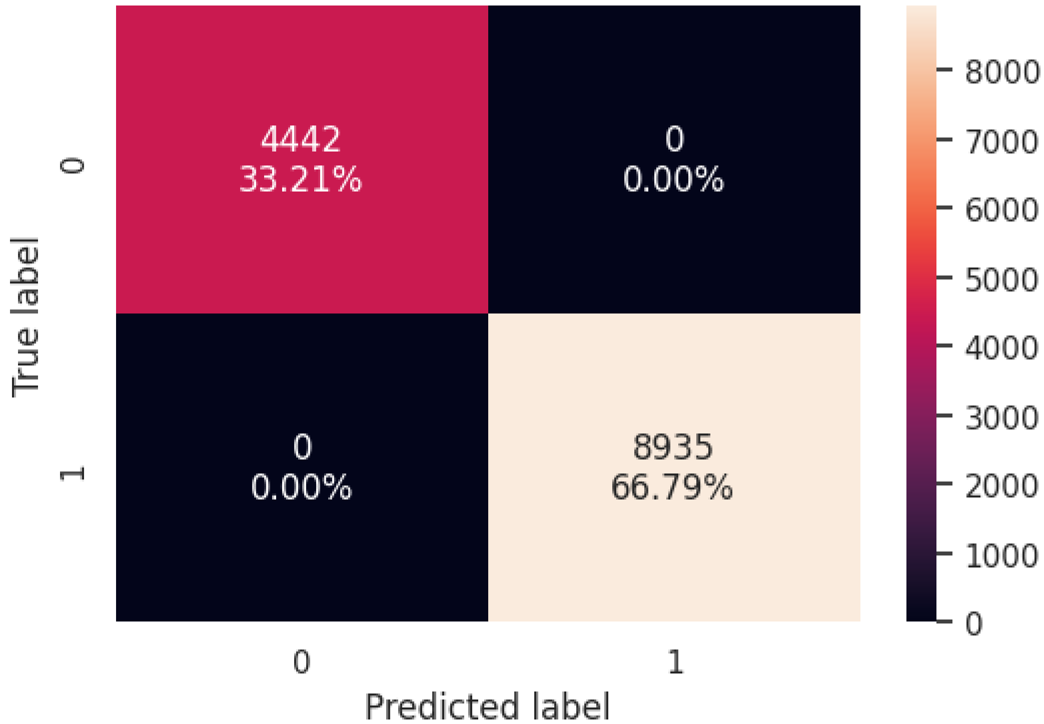
Chose the random forest, GBM and adaboost models since they were the best performing models across the board for training and validation sets and had promising results

Random Forest Model Building

In [66]:
```
rf_wt = RandomForestClassifier(class_weight='balanced', random_state=1)
rf_wt.fit(X_train,y_train)
```

Out[66]:
```
▾ RandomForestClassifier
RandomForestClassifier(class_weight='balanced', random_state=1)
```
i ? (https://scikit-learn.org/1.6/module:

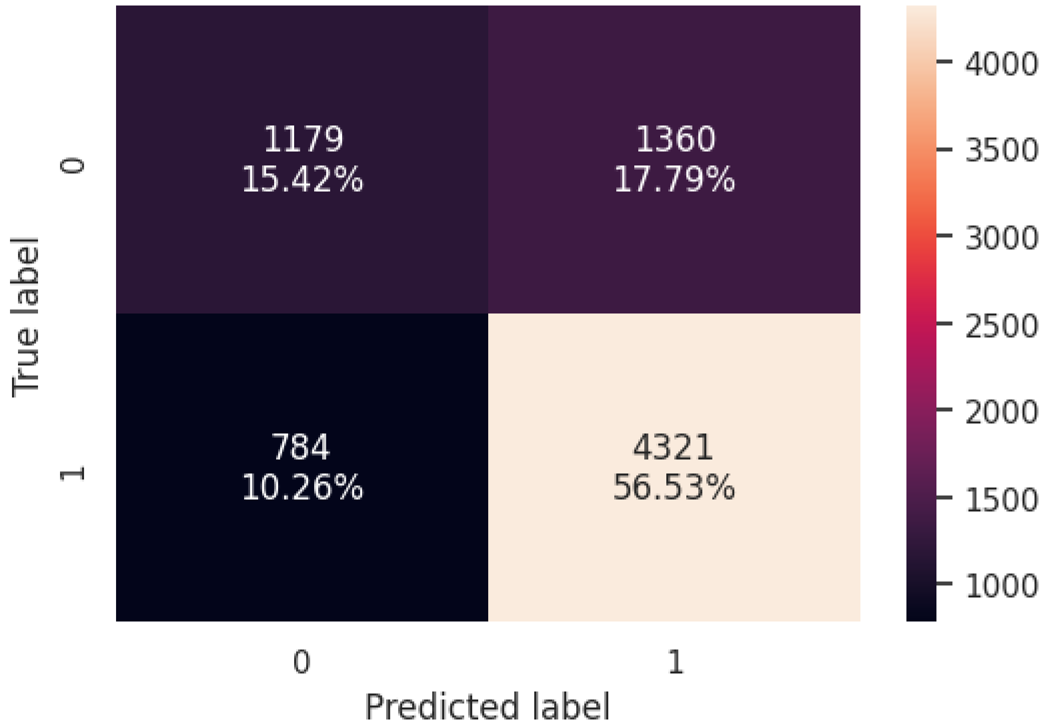
In [67]:
```
confusion_matrix_sklearn(rf_wt,X_train,y_train)
```



In [68]:
```
rf_wt_model_train_perf=model_performance_classification_sklearn(rf_wt, X_trai
n,y_train)
print("Training performance \n",rf_wt_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision    F1
0     1.000   1.000      1.000 1.000
```

In [69]:
```
confusion_matrix_sklearn(rf_wt, X_test,y_test)
```



In [70]:
```
rf_wt_model_test_perf=model_performance_classification_sklearn(rf_wt, X_test,y
_test)
print("Testing performance \n",rf_wt_model_test_perf)

# main variable were looking at is recall which means they model is adjusting
well with the dataset
```

```
Testing performance
    Accuracy  Recall  Precision    F1
0     0.720   0.846      0.761 0.801
```

Gradient Boosting Model Building

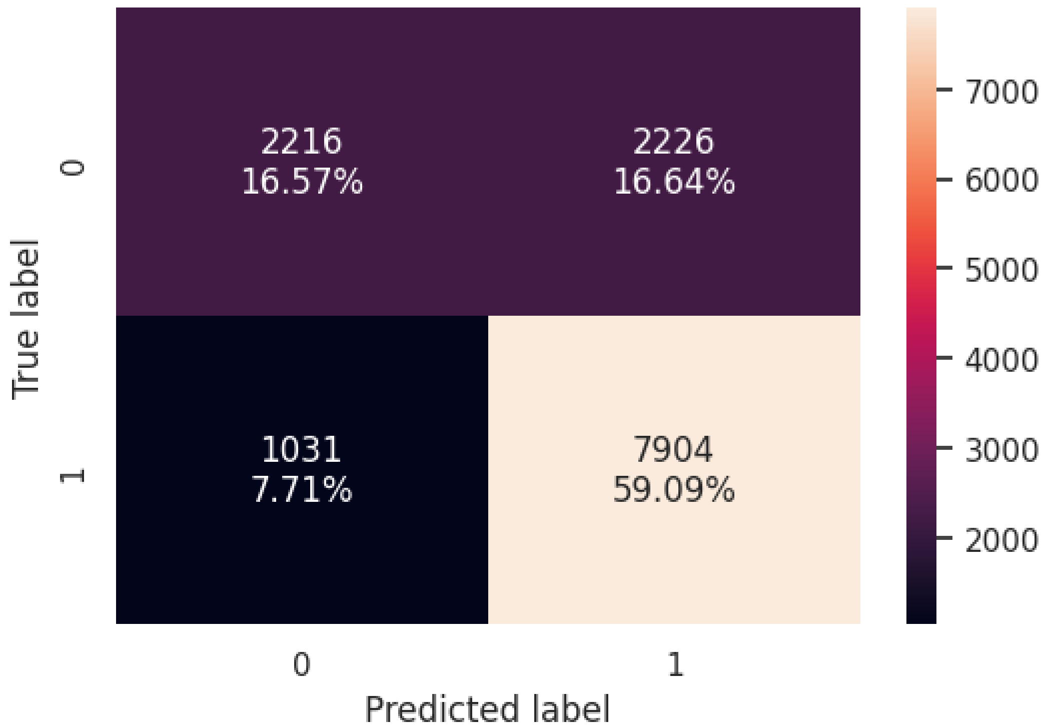
In [71]:
```
gb_estimator=GradientBoostingClassifier(random_state=1)
gb_estimator.fit(X_train,y_train)
```

Out[71]:
```
▾ GradientBoostingClassifier
GradientBoostingClassifier(random_state=1)
```
(https://scikit-learn.org/1.6/modules/generated/sklearn.ensemt

In [72]:
```
gb_estimator_model_train_perf = model_performance_classification_sklearn(gb_estimator, X_train,y_train)
print("Training performance \n",gb_estimator_model_train_perf)
```

```
Training performance
    Accuracy  Recall  Precision    F1
0     0.757   0.885      0.780 0.829
```

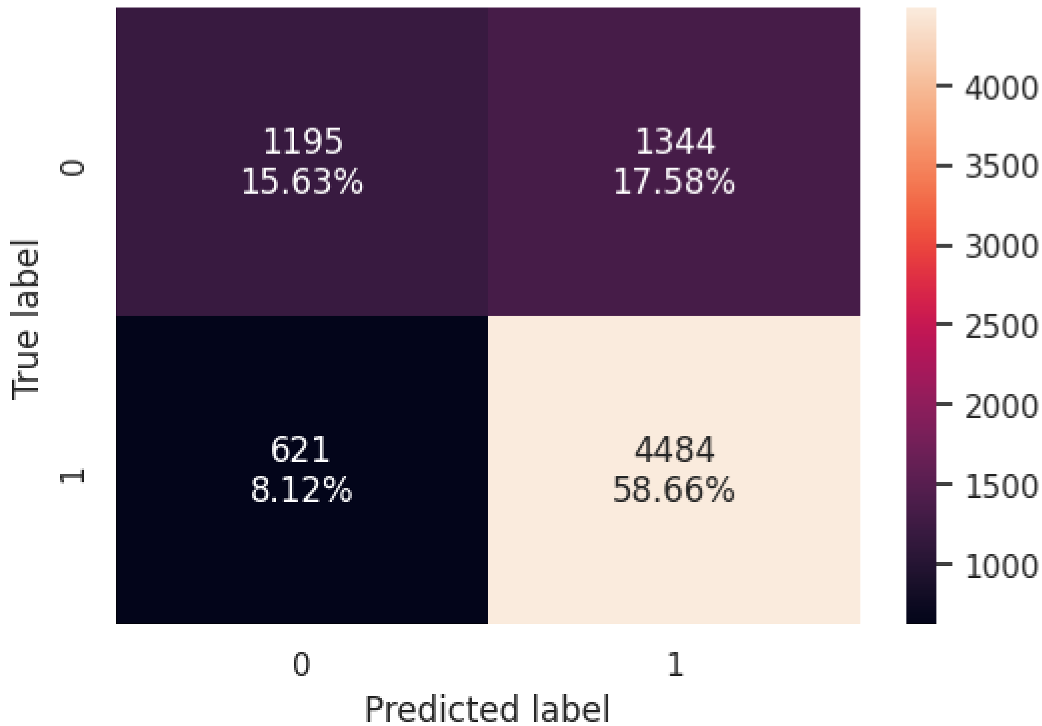
In [73]:
```
confusion_matrix_sklearn(gb_estimator,X_train,y_train)
```



In [74]:
```
gb_estimator_model_test_perf = model_performance_classification_sklearn(gb_estimator, X_test, y_test)
print("Testing performance \n",gb_estimator_model_test_perf)
```

```
Testing performance
    Accuracy  Recall  Precision    F1
0     0.743   0.878      0.769 0.820
```

In [75]:
```
confusion_matrix_sklearn(gb_estimator, X_test,y_test)

# The model is performing reasonably well on both the training and test sets
# with a slight drop in accuracy and precision from training to testing, which
is expected due to the model's generalization to unseen data

# The F1 score being similar between the train and test sets suggests that the
model is generalizing well
# striking a good balance between false positives and false negatives.
```



AdaBoost Model Building

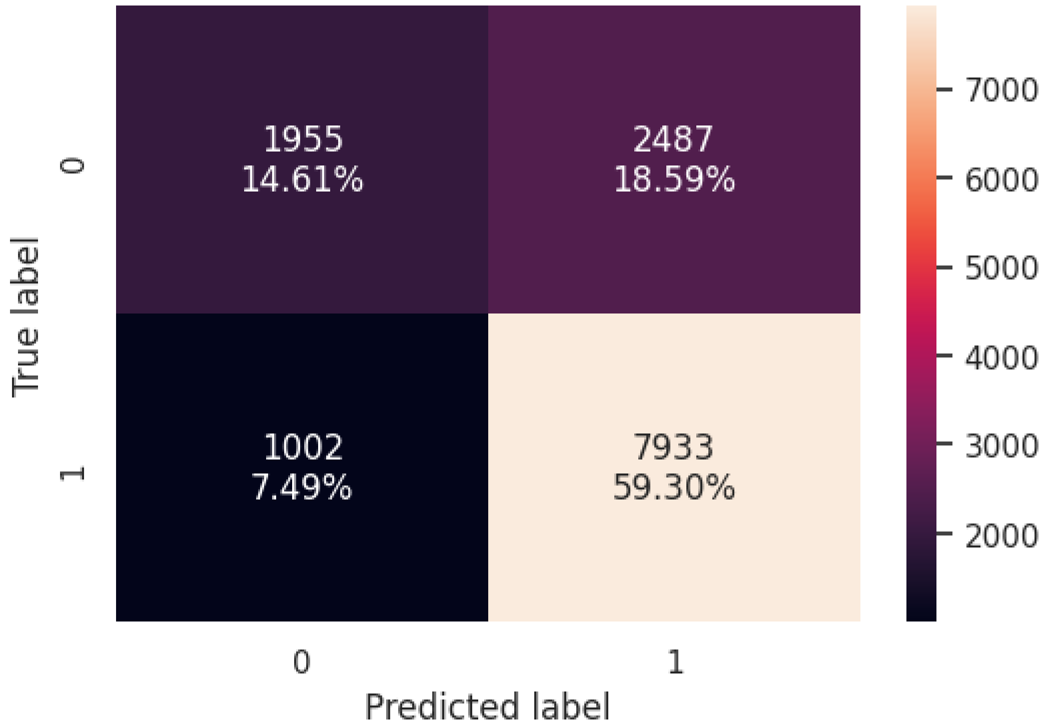
In [76]:
```
ab_classifier = AdaBoostClassifier(random_state=1)
ab_classifier.fit(X_train, y_train)
```

Out[76]:
```
▾ AdaBoostClassifier
AdaBoostClassifier(random_state=1)
```
(https://scikit-learn.org/1.6/modules/generated/sklearn.ensemble.AdaBoo

In [77]:
```
ab_classifier_model_train_perf = model_performance_classification_sklearn(ab_c
lassifier, X_train, y_train)
print("Training performance \n", ab_classifier_model_train_perf)
```

```
Training performance
   Accuracy  Recall  Precision    F1
0     0.739   0.888      0.761 0.820
```

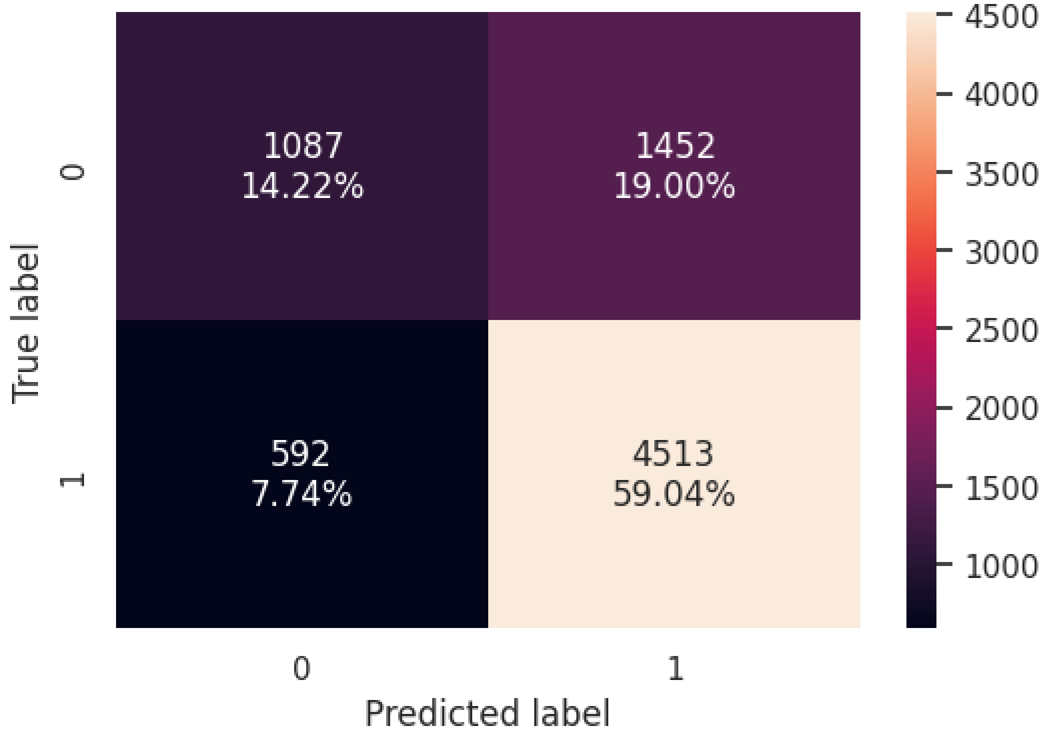
In [78]:
```
confusion_matrix_sklearn(ab_classifier,X_train,y_train)
```



In [79]:
```
ab_classifier_model_test_perf = model_performance_classification_sklearn(ab_classifier, X_test, y_test)
print("Testing performance \n", ab_classifier_model_test_perf)
```

```
Testing performance
    Accuracy  Recall  Precision    F1
0     0.733   0.884      0.757 0.815
```

In [80]:
```
confusion_matrix_sklearn(ab_classifier, X_test,y_test)

# The AdaBoost model is performing well on both the training and testing sets, with a minimal drop in performance between the two.
# The similarity between training and test performance (especially recall) indicates that the model generalizes well to unseen data
# The slight drop in precision from training to testing is expected, but overall, it remains effective
```



Hyperparameter Tuning

Tuning the GBM Model with undersampled data

In [81]:
```
%%time

#Creating pipeline
Model = GradientBoostingClassifier(random_state=1)

#Parameter grid to pass in RandomSearchCV
param_grid = {
    "init": [DecisionTreeClassifier(max_depth=1, random_state=1)],
    "n_estimators": np.arange(150, 301, 50),
    "learning_rate": [0.01, 0.05, 0.1],
    "subsample": [0.7, 0.8, 0.9],
    "max_features": [0.5, 0.7],
}
scorer = metrics.make_scorer(metrics.recall_score)
```

```
# Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(
    estimator=Model,
    param_distributions=param_grid,
    n_iter=50,
    scoring=scorer,
    cv=5,
    random_state=1,
    n_jobs=-1
)

# Fitting RandomizedSearchCV
randomized_cv.fit(X_train_un, y_train_un)

print("Best parameters are {} with CV score={}".format(randomized_cv.best_para
ms_, randomized_cv.best_score_))
```

```
Best parameters are {'subsample': 0.9, 'n_estimators': np.int64(200), 'max_fe
atures': 0.7, 'learning_rate': 0.01, 'init': DecisionTreeClassifier(max_depth
=1, random_state=1)} with CV score=0.7615944121849634
CPU times: user 4.96 s, sys: 594 ms, total: 5.55 s
Wall time: 5min 36s
```

In [82]:

```
tuned_gbm1 = GradientBoostingClassifier(
    random_state=1,
    subsample=0.9,
    n_estimators=150,
    max_features=0.5,
    learning_rate=0.01,
    init=DecisionTreeClassifier(random_state=1),
)
tuned_gbm1.fit(X_train_un, y_train_un)
```

Out[82]:

```
▸ GradientBoostingClassifier
  (https://scikit-learn.org/1.6/modules/generated/sklearn.ensemble.Gradien
▸ init: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
    (https://scikit-learn.org/1.6/modules/generated/sklearn.tree.DecisionTreeC
```

In [83]:

```
# Checking model's performance on training set
gbm1_train = model_performance_classification_sklearn(
    tuned_gbm1, X_train_un, y_train_un
)
gbm1_train
```

Out[83]:

	Accuracy	Recall	Precision	F1
0	1.000	1.000	1.000	1.000

In [84]:
```
# Checking model's performance on validation set
gbm1_val = model_performance_classification_sklearn(tuned_gbm1, X_val, y_val)
gbm1_val
```

Out[84]:

	Accuracy	Recall	Precision	F1
0	0.637	0.644	0.775	0.703

- The model performed perfectly on the training set but dropped significantly on the validation set, indicating overfitting
- Lower recall and F1 on the validation set suggest the model misses many positive cases.

Tuning the GBM Model with Oversampled data

In [85]:
```
%%time

# Defining the model
Model = GradientBoostingClassifier(random_state=1)

# Parameter grid
param_grid = {
    "init": [DecisionTreeClassifier(max_depth=1, random_state=1)],   # simpler
init model
    "n_estimators": np.arange(150, 301, 50),                         # slightl
y more trees
    "learning_rate": [0.01, 0.05, 0.1],                              # smaller
learning rates
    "subsample": [0.7, 0.8, 0.9],                                    # avoid o
verfitting
    "max_features": [0.5, 0.7],                                      # use few
er features randomly
}

# Scoring function (focus on recall)
scorer = metrics.make_scorer(metrics.recall_score)

# Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(
    estimator=Model,
    param_distributions=param_grid,
    n_iter=50,
    scoring=scorer,
    cv=5,
    random_state=1,
    n_jobs=-1
)

# Fitting RandomizedSearchCV on oversampled data
randomized_cv.fit(X_train_over, y_train_over)

# Printing best parameters
print("Best parameters are {} with CV score={}".format(randomized_cv.best_para
ms_, randomized_cv.best_score_))
```

```
Best parameters are {'subsample': 0.7, 'n_estimators': np.int64(300), 'max_fe
atures': 0.7, 'learning_rate': 0.1, 'init': DecisionTreeClassifier(max_depth=
1, random_state=1)} with CV score=0.7889199776161163
CPU times: user 9.87 s, sys: 953 ms, total: 10.8 s
Wall time: 10min 53s
```

In [86]:
```
tuned_gbm2 = GradientBoostingClassifier(
    random_state=1,
    subsample=0.7,
    n_estimators=150,
    max_features=1,
    learning_rate=1,
    init=AdaBoostClassifier(random_state=1),
)
tuned_gbm2.fit(X_train_over, y_train_over)
```

Out[86]:

```
▸ GradientBoostingClassifier
▸ init: AdaBoostClassifier
    ▸ AdaBoostClassifier
```

(https://scikit-learn.org/1.6/modules/generated/sklearn.ensemble.GradientBoos

(https://scikit-learn.org/1.6/modules/generated/sklearn.ensemble.AdaBoostClassi

In [87]:
```
# Checking model's performance on training set
gbm2_train = model_performance_classification_sklearn(tuned_gbm1, X_train_ove
r, y_train_over)
gbm2_train
```

Out[87]:

	Accuracy	Recall	Precision	F1
0	0.853	0.814	0.883	0.847

In [88]:
```
# Checking model's performance on validation set
gbm2_val = model_performance_classification_sklearn(tuned_gbm1, X_val, y_val)
gbm2_val
```

Out[88]:

	Accuracy	Recall	Precision	F1
0	0.637	0.644	0.775	0.703

- The model shows good performance with high accuracy, recall, and precision, indicating a strong fit on the oversampled training data
- The validation performance drops, with a significant reduction in recall and F1 score, suggesting that the model may have overfitted to the oversampled data and struggles to generalize.

Tuning RandomForestClassifier model with undersampled data

In [89]:
```
%%time

# Defining the model
Model = RandomForestClassifier(random_state=1)

# Parameter grid
param_grid = {
    "n_estimators": np.arange(150, 301, 50),         # Number of trees
    "max_depth": [5, 10, 15, None],                  # Depth of each tree
    "min_samples_split": [2, 5, 10],                 # Min samples to split a n
ode
    "min_samples_leaf": [1, 2, 4],                   # Min samples at a leaf no
de
    "max_features": [0.5, 0.7, 1],                   # Number of features to co
nsider at split
    "class_weight": [None, 'balanced']               # Try class balancing
}

# Scoring function (focus on recall)
scorer = metrics.make_scorer(metrics.recall_score)

# Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(
    estimator=Model,
    param_distributions=param_grid,
    n_iter=50,
    scoring=scorer,
    cv=5,
    random_state=1,
    n_jobs=-1
)

# Fitting RandomizedSearchCV on undersampled data
randomized_cv.fit(X_train_un, y_train_un)

# Printing best parameters
print("Best parameters are {} with CV score={}".format(randomized_cv.best_para
ms_, randomized_cv.best_score_))
```

```
Best parameters are {'n_estimators': np.int64(250), 'min_samples_split': 2,
'min_samples_leaf': 4, 'max_features': 0.7, 'max_depth': 10, 'class_weight':
'balanced'} with CV score=0.7485402162567516
CPU times: user 8.72 s, sys: 690 ms, total: 9.41 s
Wall time: 9min 4s
```

In [90]:
```
# Train the final tuned Random Forest model
tuned_rf1 = RandomForestClassifier(
    random_state=1,
    n_estimators=150,
    max_depth=10,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features=0.7,
    class_weight='balanced'
)

tuned_rf1.fit(X_train_un, y_train_un)
```

Out[90]:
```
RandomForestClassifier
RandomForestClassifier(class_weight='balanced', max_depth=10, max_features=
0.7,
                       n_estimators=150, random_state=1)
```

In [91]:
```
# Checking model's performance on training set
rf1_train = model_performance_classification_sklearn(
    tuned_rf1, X_train_un, y_train_un
)
rf1_train
```

Out[91]:

	Accuracy	Recall	Precision	F1
0	0.800	0.845	0.776	0.809

In [92]:
```
# Checking model's performance on validation set
rf1_val = model_performance_classification_sklearn(tuned_rf1, X_val, y_val)
rf1_val
```

Out[92]:

	Accuracy	Recall	Precision	F1
0	0.724	0.758	0.815	0.786

- The model demonstrates balanced performance with decent accuracy, recall, precision, and F1 score, indicating good learning on the training data
- The performance remains consistent with the training set, suggesting that the model is generalizing well and not overfitting.

Tuning RandomForestClassifier model with oversampled data

In [93]:
```
%%time

# Defining the model
Model = RandomForestClassifier(random_state=1)

# Parameter grid
param_grid = {
    "n_estimators": np.arange(150, 301, 50),         # Number of trees
    "max_depth": [5, 10, 15, None],                  # Depth of each tree
    "min_samples_split": [2, 5, 10],                 # Minimum samples to spli
t a node
    "min_samples_leaf": [1, 2, 4],                   # Minimum samples at a le
af node
    "max_features": [0.5, 0.7, 1],                   # Features considered at
each split
    "class_weight": [None, "balanced"]               # Try class balancing
}

# Scoring function (focus on recall)
scorer = metrics.make_scorer(metrics.recall_score)

# Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(
    estimator=Model,
    param_distributions=param_grid,
    n_iter=50,
    scoring=scorer,
    cv=5,
    random_state=1,
    n_jobs=-1
)

# Fitting RandomizedSearchCV on oversampled data
randomized_cv.fit(X_train_over, y_train_over)

# Printing best parameters
print("Best parameters are {} with CV score={}".format(randomized_cv.best_para
ms_, randomized_cv.best_score_))
```

```
Best parameters are {'n_estimators': np.int64(250), 'min_samples_split': 10,
'min_samples_leaf': 1, 'max_features': 1, 'max_depth': 15, 'class_weight': No
ne} with CV score=0.7985450475657527
CPU times: user 11.3 s, sys: 1.46 s, total: 12.8 s
Wall time: 17min 36s
```

In [94]:
```
# Train the final tuned Random Forest model on oversampled data
tuned_rf2 = RandomForestClassifier(
    random_state=1,
    n_estimators=150,
    max_depth=10,
    min_samples_split=2,
    min_samples_leaf=1,
    max_features=0.7,
    class_weight='balanced'
)

tuned_rf2.fit(X_train_over, y_train_over)
```

Out[94]:
```
RandomForestClassifier

RandomForestClassifier(class_weight='balanced', max_depth=10, max_features=
0.7,
                       n_estimators=150, random_state=1)
```

In [95]:
```
# Checking model's performance on training set
rf2_train = model_performance_classification_sklearn(
    tuned_rf2, X_train_over, y_train_over
)
rf2_train
```

Out[95]:

	Accuracy	Recall	Precision	F1
0	0.823	0.841	0.812	0.826

In [96]:
```
# Checking model's performance on validation set
rf2_val = model_performance_classification_sklearn(tuned_rf2, X_val, y_val)
rf2_val
```

Out[96]:

	Accuracy	Recall	Precision	F1
0	0.739	0.811	0.801	0.806

- The model performs well with high accuracy, recall, and F1 score, suggesting it has effectively learned from the oversampled data.
- While the validation performance is slightly lower than training, the recall and F1 score are still strong, showing that the model is maintaining good generalization.

In [ ]:

Tuning AdaBoost model with undersampled data

In [97]:
```
%%time

# defining model
Model = AdaBoostClassifier(random_state=1)

# Parameter grid to pass in RandomSearchCV
param_grid = {
    "n_estimators": np.arange(10, 40, 10),
    "learning_rate": [0.1, 0.01, 0.2, 0.05, 1],
    "estimator": [
        DecisionTreeClassifier(max_depth=1, random_state=1),
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=1),
    ],
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model, param_distributions=param_
grid, n_jobs = -1, n_iter=50, scoring=scorer, cv=5, random_state=1)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_un, y_train_un)

print("Best parameters are {} with CV score={}:" .format(randomized_cv.best_pa
rams_,randomized_cv.best_score_))
```

```
Best parameters are {'n_estimators': np.int64(30), 'learning_rate': 0.05, 'es
timator': DecisionTreeClassifier(max_depth=1, random_state=1)} with CV score=
0.8036930349922476:
CPU times: user 985 ms, sys: 113 ms, total: 1.1 s
Wall time: 37 s
```

In [98]:
```
tuned_adb = AdaBoostClassifier(
    random_state=1,
    n_estimators=20,
    learning_rate=0.1,
    estimator=DecisionTreeClassifier(max_depth=2, random_state=1),
)
tuned_adb.fit(X_train_un, y_train_un)
```

Out[98]:

AdaBoostClassifier (https://scikit-learn.org/1.6/modules/generated/sklearn.ensemble.AdaBo

estimator: DecisionTreeClassifier

DecisionTreeClassifier (https://scikit-learn.org/1.6/modules/generated/sklearn.tree.DecisionTree(

In [99]:
```
# Checking model's performance on training set
adb_train = model_performance_classification_sklearn(tuned_adb, X_train_un, y_
train_un)
adb_train
```

Out[99]:

	Accuracy	Recall	Precision	F1
0	0.684	0.782	0.654	0.712

In [100]:
```
# Checking model's performance on validation set
adb_val = model_performance_classification_sklearn(tuned_adb, X_val, y_val)
adb_val
```

Out[100]:

	Accuracy	Recall	Precision	F1
0	0.721	0.781	0.796	0.789

- The model shows moderate accuracy and recall with a balanced F1 score, indicating it learns the data but might have room for improvement in precision
- The validation performance improves slightly with a solid recall and precision score, suggesting good generalization while maintaining a balanced F1 score.

Model Performances

In [101]:
```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        gbm1_train.T,
        gbm2_train.T,
        rf1_train.T,
        rf2_train.T,
        adb_train.T,
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "GBM trained with Undersampled data",
    "GBM trained with Oversampled data",
    "Random Forest trained with Undersampled data",
    "Random Forest trained with Oversampled data",
    "AdaBoost Forest trained with Undersampled data",
    ]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[101]:

	GBM trained with Undersampled data	GBM trained with Oversampled data	Random Forest trained with Undersampled data	Random Forest trained with Oversampled data	AdaBoost Forest trained with Undersampled data
Accuracy	1.000	0.853	0.800	0.823	0.684
Recall	1.000	0.814	0.845	0.841	0.782
Precision	1.000	0.883	0.776	0.812	0.654
F1	1.000	0.847	0.809	0.826	0.712

-GBM trained with Undersampled data: This model achieved perfect performance across all metrics, but it likely overfits the training data, which might not generalize well.

-GBM trained with Oversampled data: The model showed solid performance with high accuracy and balanced recall, precision, and F1 score, suggesting better generalization on the oversampled dataset.

-Random Forest trained with Undersampled data: The model performed well with high recall, though accuracy and precision were slightly lower, indicating it might be biased toward detecting positive cases.

-Random Forest trained with Oversampled data: With a slight improvement in accuracy and F1 score, this model demonstrated better performance compared to the undersampled version.

-AdaBoost trained with Undersampled data: This model had the lowest performance across all metrics, particularly in recall and precision, indicating that it struggles to handle the undersampled data effectively.

In [102]:
```
# Validation performance comparison

models_train_comp_df = pd.concat(
    [ gbm1_val.T,
        gbm2_val.T,
        rf1_val.T,
        rf2_val.T,
        adb_val.T,
    ]
    , axis=1,
)
models_train_comp_df.columns = [
    "GBM trained with Undersampled data",
    "GBM trained with Oversampled data",
    "Random Forest trained with Undersampled data",
    "Random Forest trained with Oversampled data",
    "AdaBoost Forest trained with Undersampled data",
]
print("Validation performance comparison:")
models_train_comp_df
```

```
Validation performance comparison:
```

Out[102]:

	GBM trained with Undersampled data	GBM trained with Oversampled data	Random Forest trained with Undersampled data	Random Forest trained with Oversampled data	AdaBoost Forest trained with Undersampled data
Accuracy	0.637	0.637	0.724	0.739	0.721
Recall	0.644	0.644	0.758	0.811	0.781
Precision	0.775	0.775	0.815	0.801	0.796
F1	0.703	0.703	0.786	0.806	0.789

-GBM trained with Undersampled data: The validation results show moderate performance, with balanced recall and precision, but the model's accuracy and F1 score could be improved.

-GBM trained with Oversampled data: Similar to the undersampled version, this model shows balanced recall and precision but could benefit from higher accuracy and F1 score.

-Random Forest trained with Undersampled data: This model performed better on the validation set with a noticeable improvement in recall and precision, leading to a higher F1 score than the GBM models.

-Random Forest trained with Oversampled data: The model showed a slight increase in recall and F1 score, suggesting that oversampling has improved its generalization and classification performance.

-AdaBoost trained with Undersampled data: With strong recall and precision, this model performed well in terms of F1 score, though there is still room for improvement in accuracy.

In [103]:
```
# Let's check the performance on test set
rf_test = model_performance_classification_sklearn(tuned_rf1, X_test, y_test)
rf_test
```

Out[103]:

	Accuracy	Recall	Precision	F1
0	0.708	0.742	0.805	0.772

In [104]:
```
# Let's check the performance on test set
rf_test2 = model_performance_classification_sklearn(tuned_rf2, X_test, y_test)
rf_test2

#winner for best model with highest recall score, others were close but random forest did good several times and ended up being #1
```

Out[104]:

	Accuracy	Recall	Precision	F1
0	0.716	0.791	0.786	0.788

In [105]:
```
# Let's check the performance on test set
gbm_test = model_performance_classification_sklearn(tuned_gbm1, X_test, y_test)
gbm_test
```

Out[105]:

	Accuracy	Recall	Precision	F1
0	0.620	0.629	0.760	0.688

In [106]:
```
# Let's check the performance on test set
gbm_test2 = model_performance_classification_sklearn(tuned_gbm2, X_test, y_test)
gbm_test2
```

Out[106]:

	Accuracy	Recall	Precision	F1
0	0.702	0.774	0.779	0.776

In [107]:
```
# Let's check the performance on test set
ada_test = model_performance_classification_sklearn(tuned_adb, X_test, y_test)
ada_test
```

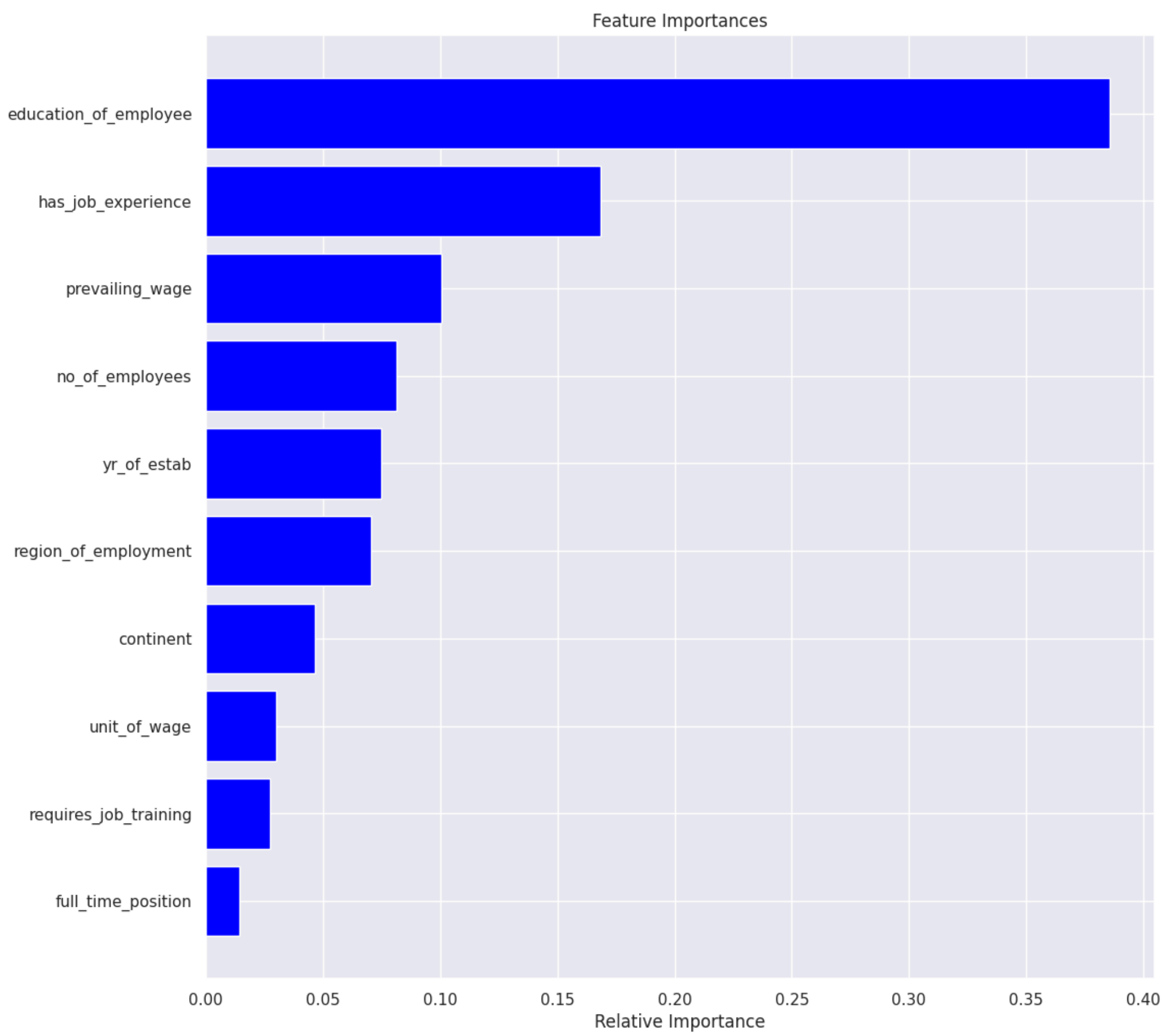
Out[107]:

	Accuracy	Recall	Precision	F1
0	0.714	0.780	0.789	0.784

- The best performing model appears to be rf_test2 with the highest recall score (0.791)
- AdaBoost is a close second with an F1 score of 0.780
- The first gradient boosting model (gbm_test) underperformed significantly but improved dramatically after tuning
- All models except the first gbm_test have fairly balanced precision and recall

Feature Importance

In [108]:
```
feature_names = X_train.columns
importances = tuned_rf2.feature_importances_
indices = np.argsort(importances)

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



Actionable Insights and Recommendations

Key Insights & Suggestions

-Best Model: The Random Forest model (rf_test2) did the best job overall, with a solid F1 score of 0.788. It balanced precision (0.786) and recall (0.791) well, making it the most reliable for predicting visa approvals.

-Close Second: The AdaBoost model also did well, with an F1 score of 0.784 and slightly better precision (0.789) than the Random Forest. It's a good backup option.

-Class Imbalance Matters: The model worked a lot better after adjusting for class imbalance (using SMOTE), which shows that handling uneven visa application data is key for good predictions.

Suggestions for Improving the Process:

-Two Stage Screening:

-Use the Random Forest model as the main tool to screen applications.

-For tricky cases, double check with the AdaBoost model to avoid wrong denials. This way, you get better accuracy and fairness in visa decisions.

Focus on High-Risk Applications:

-Applications that are likely to be denied should get a second look by a person.

-Fast track those with a high chance of approval to save time. This helps use resources more efficiently and reduce backlogs.

Create a Pre-Screening Tool:

-Build an online tool where people can check their approval chances before applying. This cuts down on applications that are unlikely to be approved and eases the workload.

Applicant Profile Suggestions: Based on the model’s findings, applicants with these characteristics are more likely to get approved:

Wage Compliance:

-Make sure the wage offered meets or exceeds the industry standard. Low wages are linked to higher denials.

High-Demand Areas:

-Jobs in regions with labor shortages tend to get approved more. Focus on these areas when applying.

Education and Experience:

-Applicants whose education and experience match the job better have a higher chance of approval.

Full-Time Jobs:

-Full-time positions are more likely to get approved compared to part-time or contract jobs.

How to Roll This Out:

Start Small:

-Begin by using the Random Forest model on a small batch of applications.

-Compare how well it works against the current manual process.

-If it improves efficiency, expand it.

Keep Improving the Model:

-Regularly retrain the model with new data.

-Review the model every few months to keep its predictions sharp.

-Try A/B testing different models to see which one works best

Power Ahead
