EasyVisa Project

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 your firm EasyVisa for data-driven solutions. You as a data scientist 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 the 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

Importing necessary libraries and data

In [1]:
```
# suppress all warnings
import warnings
warnings.filterwarnings("ignore")

#import libraries needed for data manipulation
import pandas as pd
import numpy as np

#import libraries needed for data visualization

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# using statsmodels to build our model
import statsmodels.stats.api as sms
import statsmodels.api as sm

# unlimited number of displayed columns, limit of 100 for displayed rows
pd.set_option("display.max_columns", None)
pd.set_option("display.max_rows", 100)

# split the data into random train and test subsets
from sklearn.model_selection import train_test_split

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

from sklearn.tree import DecisionTreeClassifier
```

```python
# Libraries to get different metric scores
from sklearn import metrics
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
)

# To tune different models
from sklearn.model_selection import GridSearchCV
```

Data Overview

- Observations
- Sanity checks

In [2]:

```python
#import dataset named 'EasyVisa.csv'

visa = pd.read_csv('EasyVisa.csv')

# read first five rows of the dataset

visa.head()
```

Out[2]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees	yr_of_estab	region_of_employment	prevailing_wage
0	EZYV01	Asia	High School	N	N	14513	2007	West	592.2029
1	EZYV02	Asia	Master's	Y	N	2412	2002	Northeast	83425.6500
2	EZYV03	Asia	Bachelor's	N	Y	44444	2008	West	122996.8600
3	EZYV04	Asia	Bachelor's	N	N	98	1897	West	83434.0300
4	EZYV05	Africa	Master's	Y	N	1082	2005	South	149907.3900

In [3]:

```python
visa.shape
```

Out[3]:

```
(25480, 12)
```

In [4]:

```python
visa.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 [5]:

```
visa.isnull().sum()
```

Out[5]:

```
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 [6]:

```
visa.duplicated().sum()
```

Out[6]: 0

In [7]:

```
visa['case_id'].value_counts().shape
```

Out[7]: (25480,)

Observations

- There are 25,480 rows and 12 columns.
- `no_of_employees` , `yr_of_estab` , and `prevailing_wage` are numeric type, while the rest are object in nature.
    - `case_id` is just an identifier for each hotel guest.
- There are no missing or duplicated values.

In [8]:
```
# create a copy of the data so that the original dataset is not changed.

df = visa.copy()
```

In [9]:
```
# drop Case ID variable, since it is just an identifier

df.drop(columns=['case_id'], inplace=True)
```

Exploratory Data Analysis (EDA)

- EDA is an important part of any project involving data.
- It is important to investigate and understand the data better before building a model with it.
- A few questions have been mentioned below which will help you approach the analysis in the right manner and generate insights from the data.
- A thorough analysis of the data, in addition to the questions mentioned below, should be done.

Leading Questions: *Done within Bivariate analysis section*

1. Those with higher education may want to travel abroad for a well-paid job. Does education play a role in Visa certification?

2. How does the visa status vary across different continents?

3. Experienced professionals might look abroad for opportunities to improve their lifestyles and career development. Does work experience influence visa status?

4. In the United States, employees are paid at different intervals. Which pay unit is most likely to be certified for a visa?

5. The US government has established a prevailing wage to protect local talent and foreign workers. How does the visa status change with the prevailing wage?

In [10]:
```
df.describe().T
```

Out[10]:

	count	mean	std	min	25%	50%	75%	max
no_of_employees	25480.0	5667.043210	22877.928848	-26.0000	1022.00	2109.00	3504.0000	602069.00
yr_of_estab	25480.0	1979.409929	42.366929	1800.0000	1976.00	1997.00	2005.0000	2016.00
prevailing_wage	25480.0	74455.814592	52815.942327	2.1367	34015.48	70308.21	107735.5125	319210.27

In [11]:
```
# let's view a sample of the data (random_state set to 1 to validate data every time)

df.sample(n=10, random_state=1)
```

Out[11]:

	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees	yr_of_estab	region_of_employment	prevailing_wage	uni

	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees	yr_of_estab	region_of_employment	prevailing_wage	uni
17639	Asia	Bachelor's	Y	N	567	1992	Midwest	26842.9100	
23951	Oceania	Bachelor's	N	N	619	1938	Midwest	66419.9800	
8625	Asia	Master's	N	N	2635	2005	South	887.2921	
20206	Asia	Bachelor's	Y	Y	3184	1986	Northeast	49435.8000	
7471	Europe	Bachelor's	Y	N	4681	1928	West	49865.1900	
3433	Asia	Bachelor's	Y	N	222	1989	South	813.7261	
24440	Europe	High School	N	Y	3278	1994	South	204948.3900	
12104	Asia	Master's	Y	N	1359	1997	West	202237.0400	
15656	Asia	Bachelor's	N	N	2081	2003	West	111713.0200	
23110	North America	Bachelor's	Y	N	854	1998	Northeast	444.8257	

Observations:

- The number of employees ranges from -26 to 602,069. We will take the absolute value of this column to convert all to positive values.
- The year established ranges from 1800 to 2016, with over half of the companies established after 1997.
- Prevailing wage ranges from 2 to 319,2190 dollars. Between minimum and 25th percentile is a large difference, suggests many outliers on the lower end of the range.
- `has_job_experience`, `requires_job_training`, and `full_time_position` are all Yes/No variables.
- `case_status` tells us whether the Visa certification has been Certified or Denied.

In [12]:

```
# taking the absolute values for number of employees

df["no_of_employees"] = df["no_of_employees"].abs()
```

In [13]:

```
# looking at unique value counts of all categorical variables
c = list(df.select_dtypes("object").columns)

for column in c:
    print(df[column].value_counts())
    print("-" * 50)
```

```
Asia             16861
Europe            3732
North America     3292
South America      852
Africa             551
Oceania            192
Name: continent, dtype: int64
--------------------------------------------------
Bachelor's     10234
```

```
Master's        9634
High School     3420
Doctorate       2192
Name: education_of_employee, dtype: int64
--------------------------------------------------
Y    14802
N    10678
Name: has_job_experience, dtype: int64
--------------------------------------------------
N    22525
Y     2955
Name: requires_job_training, dtype: int64
--------------------------------------------------
Northeast    7195
South        7017
West         6586
Midwest      4307
Island        375
Name: region_of_employment, dtype: int64
--------------------------------------------------
Year     22962
Hour      2157
Week       272
Month       89
Name: unit_of_wage, dtype: int64
--------------------------------------------------
Y    22773
N     2707
Name: full_time_position, dtype: int64
--------------------------------------------------
Certified    17018
Denied        8462
Name: case_status, dtype: int64
--------------------------------------------------
```

In [14]:

```
# define a function to plot a boxplot and a histogram along the same scale

def histbox(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined
    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (box, hist) = plt.subplots(
        nrows=2,                                           # Number of rows of the subplot grid = 2
                                                               # boxplot first then histogram created below
        sharex=True,                                       # x-axis same among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},       # boxplot 1/3 height of histogram
        figsize=figsize,                                   # figsize defined above as (12, 7)
    )
    # defining boxplot inside function, so when using it say histbox(df, 'cost'), df: data and cost: feature
```

```
    sns.boxplot(
        data=data, x=feature, ax=box, showmeans=True, color="chocolate"
    )  # showmeans makes mean val on boxplot have star, ax =
    sns.histplot(
        data=data, x=feature, kde=kde, ax=hist, bins=bins, color = "darkgreen"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=hist, color = "darkgreen"
    )  # For histogram if there are bins in potential graph

    # add vertical line in histogram for mean and median
    hist.axvline(
        data[feature].mean(), color="purple", linestyle="--"
    )  # Add mean to the histogram
    hist.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [15]:

```
# define a function to create labeled barplots

def bar(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
```

In [16]:

```
# function to plot distributions with respect to target


def dist_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()
```

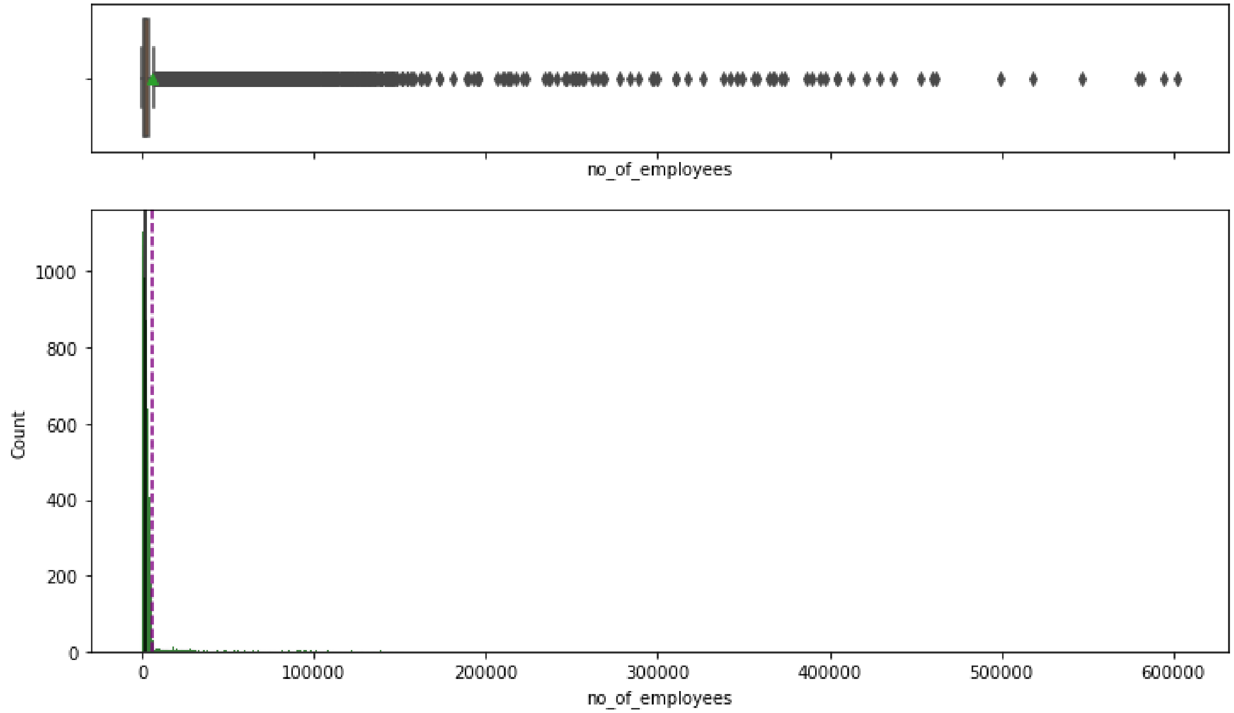
In [17]:

```
# function to plot stacked barplot

def stack(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, 6))
    plt.legend(
        loc="lower left", frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

Univariate Analysis

In [18]:

```
histbox(df, "no_of_employees")
```

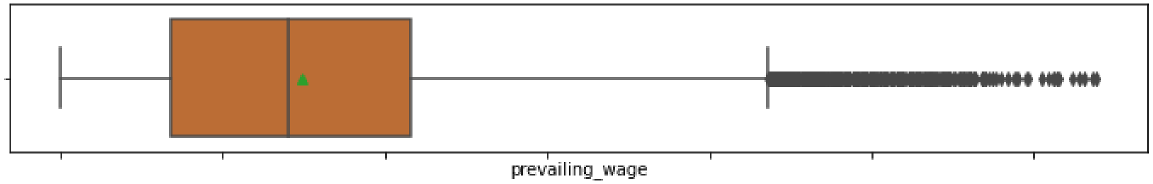


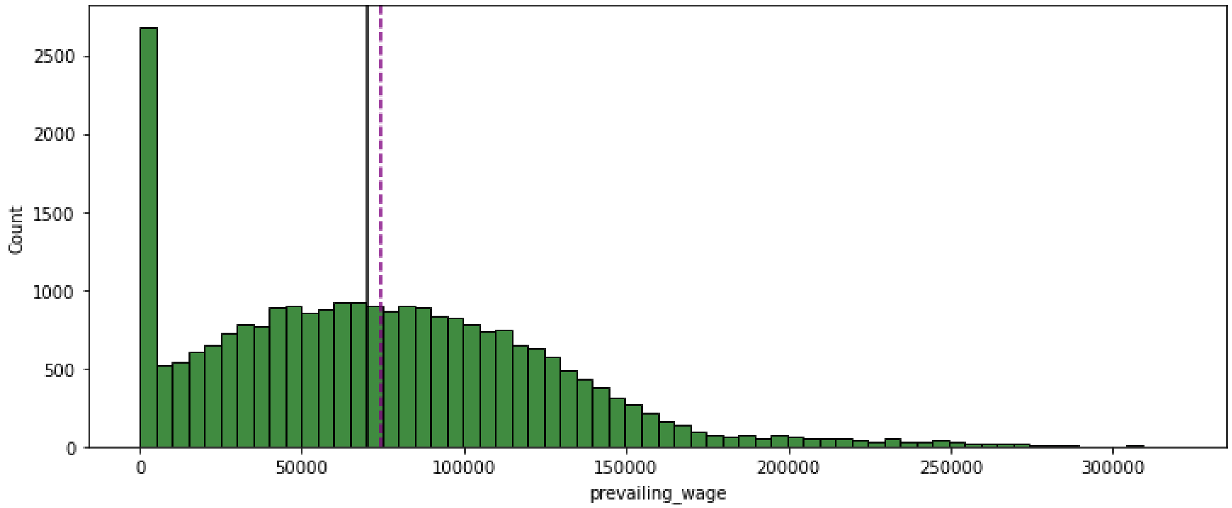
Observations

- The number of employees is highly concentrated on the first quartile of the data, with many outliers.

In [19]:

```
histbox(df, 'prevailing_wage')
```





In [20]:

```
#large number of prevailing wage less than 100 dollars, see what some of it looks like:

low = df[df["prevailing_wage"] < 100]

low.head()
```

Out[20]:

	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees	yr_of_estab	region_of_employment	prevailing_wage	unit_c
338	Asia	Bachelor's	Y	N	2114	2012	Northeast	15.7716	
634	Asia	Master's	N	N	834	1977	Northeast	3.3188	
839	Asia	High School	Y	N	4537	1999	West	61.1329	
876	South America	Bachelor's	Y	N	731	2004	Northeast	82.0029	
995	Asia	Master's	N	N	302	2000	South	47.4872	

In [21]:

```
low.groupby("unit_of_wage")['prevailing_wage'].count()
```

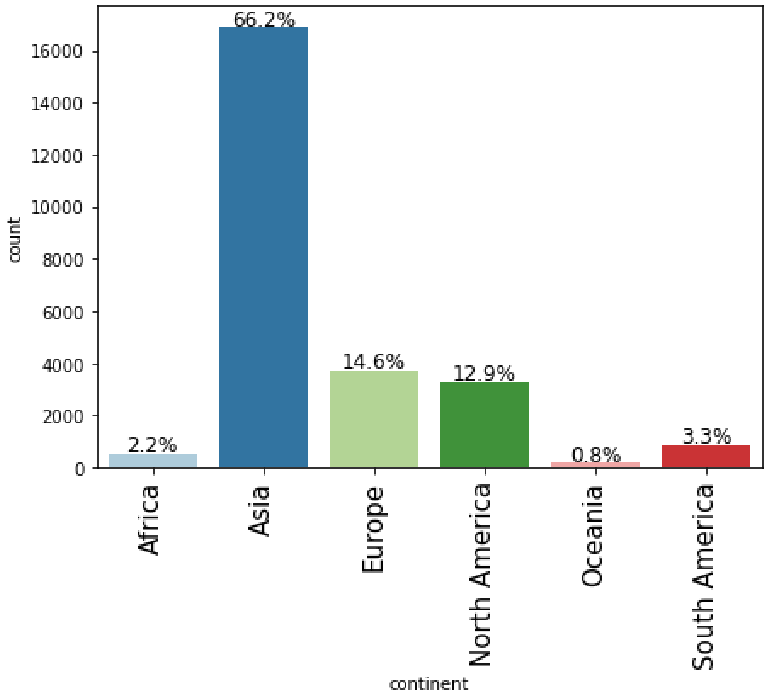
Out[21]:

```
unit_of_wage
Hour    176
Name: prevailing_wage, dtype: int64
```

Observations

- There is a concentrated number of employees with a prevailing wage (average paid to similarly employed workers in a specific intended job) in the lower range.
    - Only those paid on an hourly basis are in the less than 100 dollars bracket.
- The rest of the distribution is relatively normal, with a mean around 74,000.

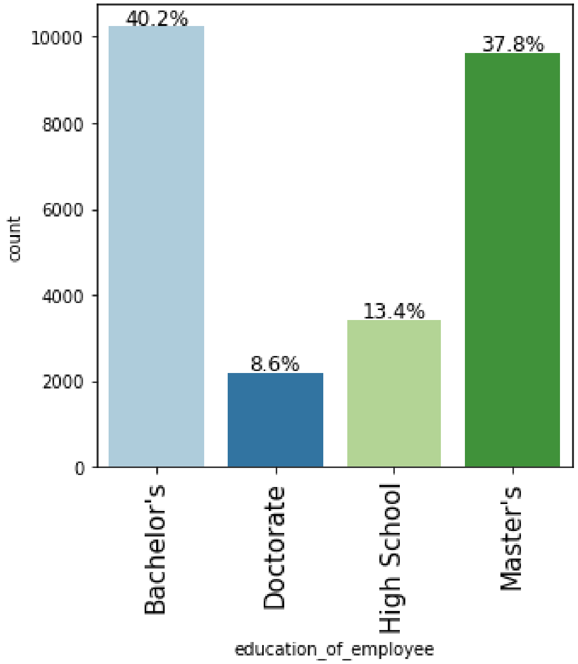
In [22]:
```
bar(df, "continent", perc=True)
```



Observations

- The majority of employees in the data are from Asia, over 66%.
- Lowest proportion of employees is in Oceania.

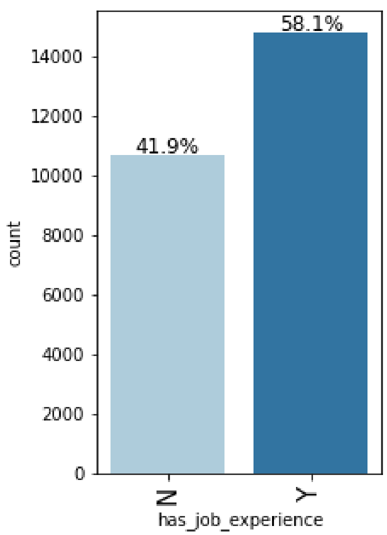
In [23]:
```
bar(df, 'education_of_employee', perc=True)
```



Observations

- 40% of employees are at a Bachelors level, closely followed by Masters.

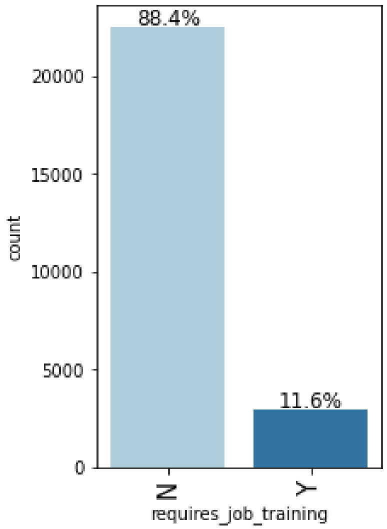
In [24]:
```
bar(df, 'has_job_experience', perc=True)
```



Observations

- About 58% of employees have job experience.

In [25]:
```
bar(df, 'requires_job_training', perc=True)
```



Observations

- Over 88% of positions require job training.

In [26]:
```
df['yr_of_estab'].describe()
```

Out[26]:
```
count    25480.000000
mean      1979.409929
std         42.366929
min       1800.000000
25%       1976.000000
50%       1997.000000
75%       2005.000000
max       2016.000000
Name: yr_of_estab, dtype: float64
```

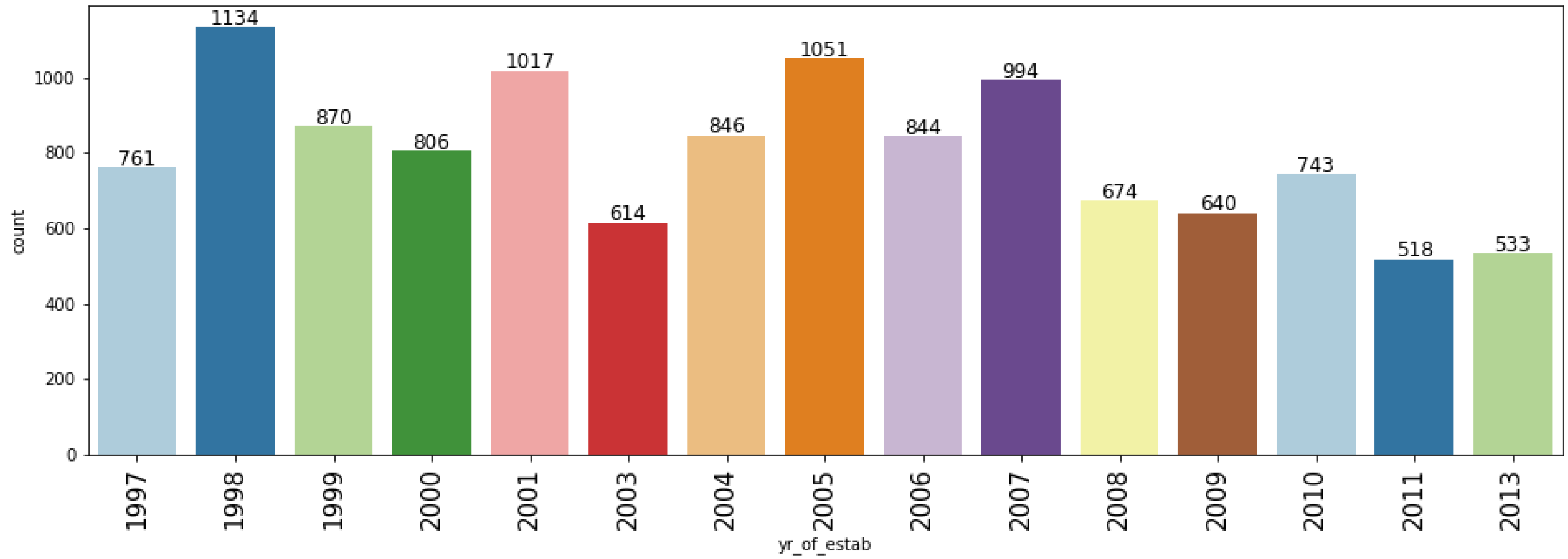
In [27]:
```
df['yr_of_estab'].value_counts()[:10]
```

Out[27]:
```
1998    1134
2005    1051
2001    1017
2007     994
1999     870
2004     846
```

```
2006      844
2000      806
1997      761
2010      743
Name: yr_of_estab, dtype: int64
```

In [28]:

```
bar(df, 'yr_of_estab', n=15)
```

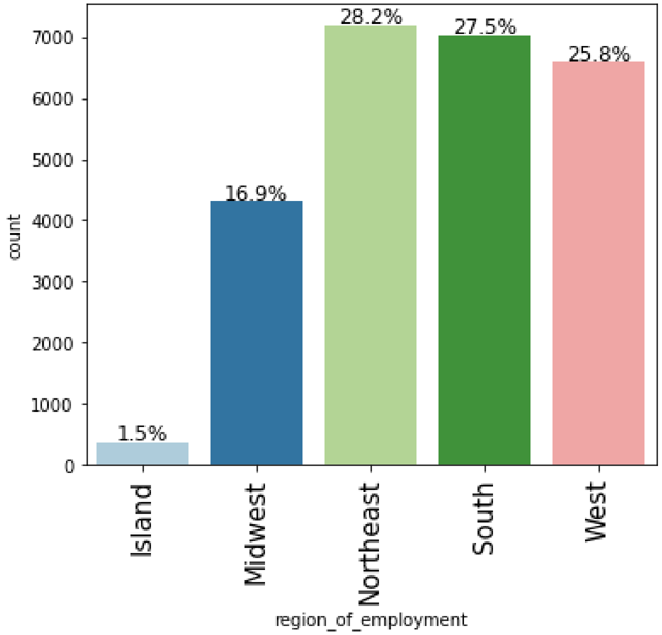


Observations

- The dataset ranges from 1800 to 2016.
- The majority of the employees' companies are established from 1997-2010, most common being 1998.

In [29]:

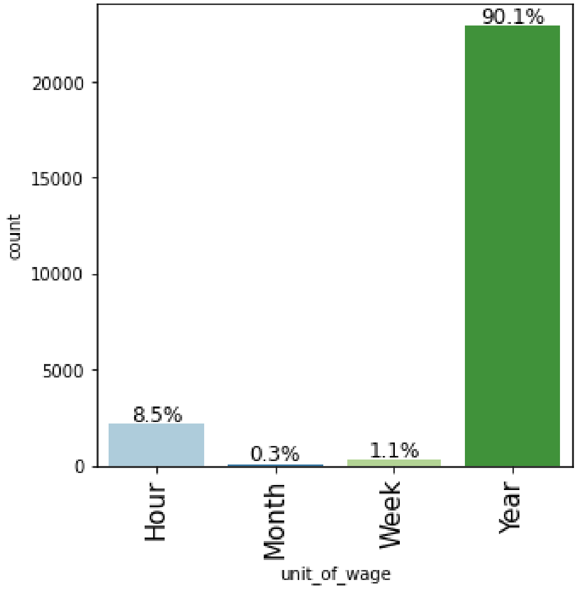
```
bar(df, 'region_of_employment', perc=True)
```



Observations

- The region of employment is about equal across the Northeast, South, and West.
- Island is lowest, with 1.5%.

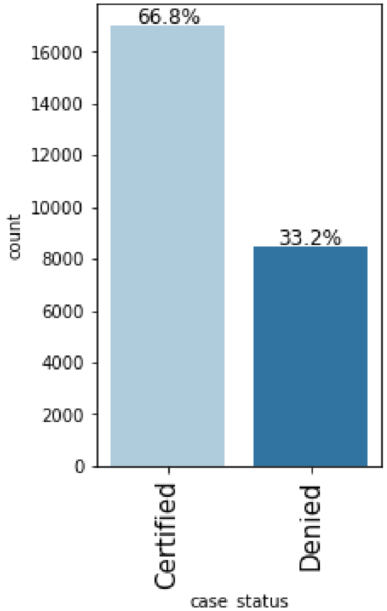
In [30]:
```
bar(df, 'unit_of_wage', perc=True)
```



Observations

- Over 90% of employees are paid on a yearly unit, i.e. a yearly salary (ex. 70,000).

In [31]:
```
bar(df, 'case_status', perc=True)
```

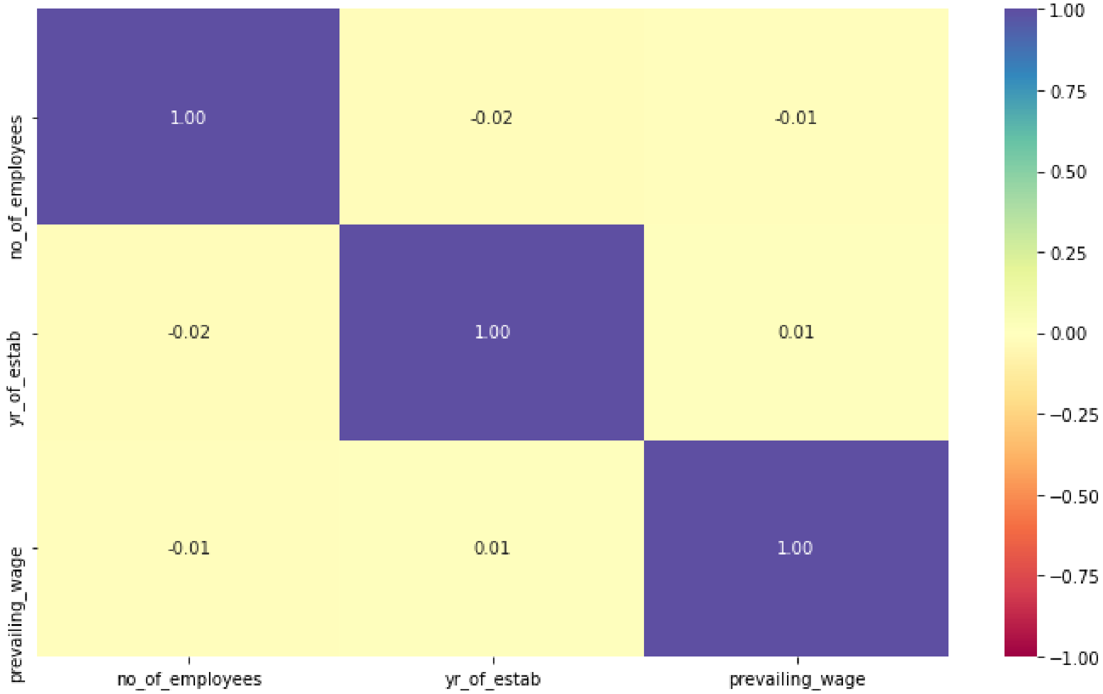


Observations

- About two-thirds of employees applying for Visas are certified.

Bivariate Analysis

In [32]:
```
corr_cols = df.select_dtypes(include=np.number).columns.tolist()

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



Observations

- None of the numeric variables are moderately or highly correlated.

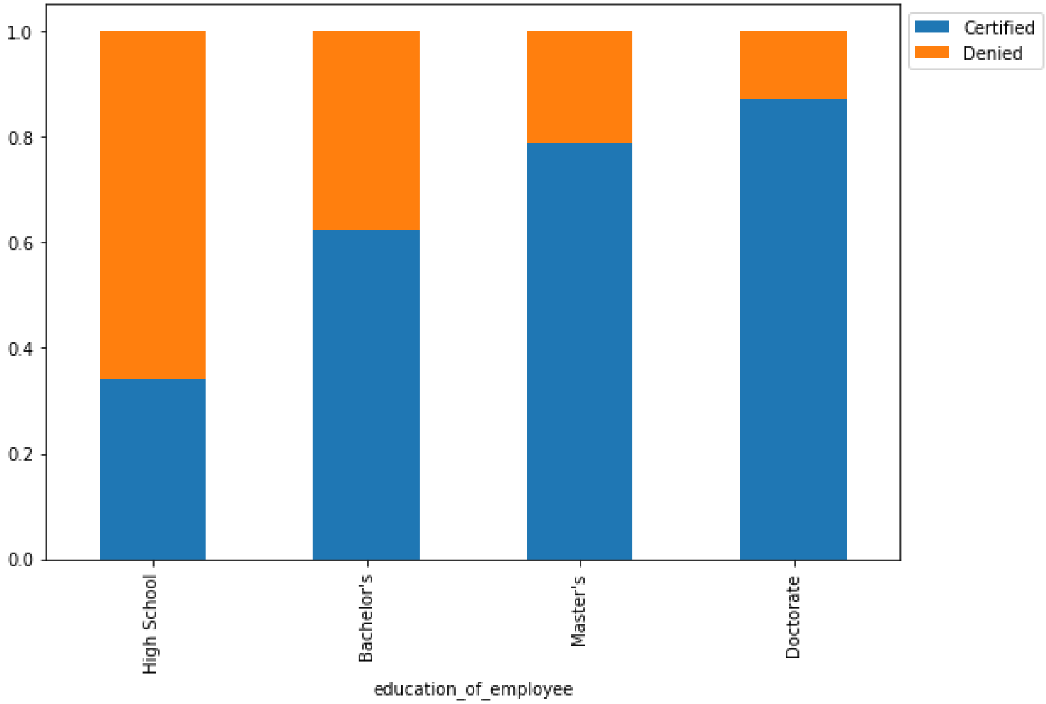
Question 1: Those with higher education may want to travel abroad for a well-paid job. Does education play a role in Visa certification?

In [33]:
```
# compare market segment to bookings

df.groupby("education_of_employee")["case_status"].value_counts()
```

Out[33]:
```
education_of_employee  case_status
Bachelor's             Certified      6367
                       Denied         3867
Doctorate              Certified      1912
                       Denied          280
High School            Denied         2256
                       Certified      1164
Master's               Certified      7575
                       Denied         2059
Name: case_status, dtype: int64
```

In [34]:
```
stack(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
------------------------------------------------------------------------------------------------------------------------
```



Observations

- The higher education level an employee has, the larger the ratio of certification approval is.
- Those with high school level education are ~66% denied certification versus ~13% for those with doctorates.
- Overall, around a third of employees are denied certification.

Question 2: How does the visa status vary across different continents?

In [35]:
```
pd.crosstab(df['case_status'], df['continent'])
```

Out[35]:

continent	Africa	Asia	Europe	North America	Oceania	South America
case_status						
Certified	397	11012	2957	2037	122	493
Denied	154	5849	775	1255	70	359

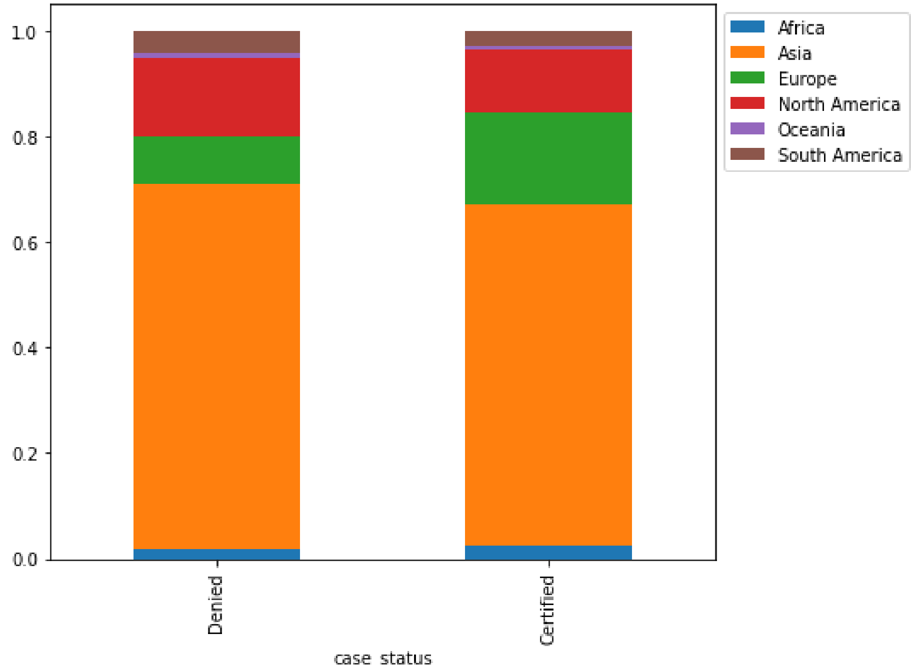
In [36]:
```
pd.crosstab(df['case_status'], df['region_of_employment'])
```

Out[36]:

region_of_employment	Island	Midwest	Northeast	South	West
case_status					
Certified	226	3253	4526	4913	4100
Denied	149	1054	2669	2104	2486

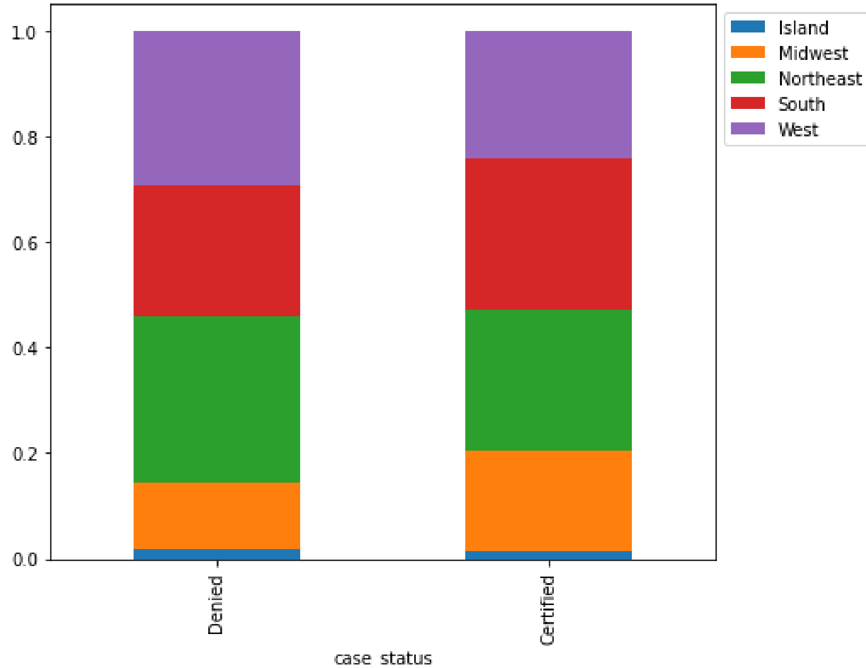
In [37]:
```
stack(df, 'case_status', 'continent')
```

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

continent      All
case_status
All          25480
Certified    17018
Denied        8462
------------------------------------------------------------------------------------------------------------------------
```



In [38]:
```
stack(df,'case_status', 'region_of_employment' )
```

```
region_of_employment  Island  Midwest  Northeast  South  West    All
case_status
All                      375     4307       7195   7017  6586  25480
Certified                226     3253       4526   4913  4100  17018
Denied                   149     1054       2669   2104  2486   8462
------------------------------------------------------------------------------------------------------------------------
```



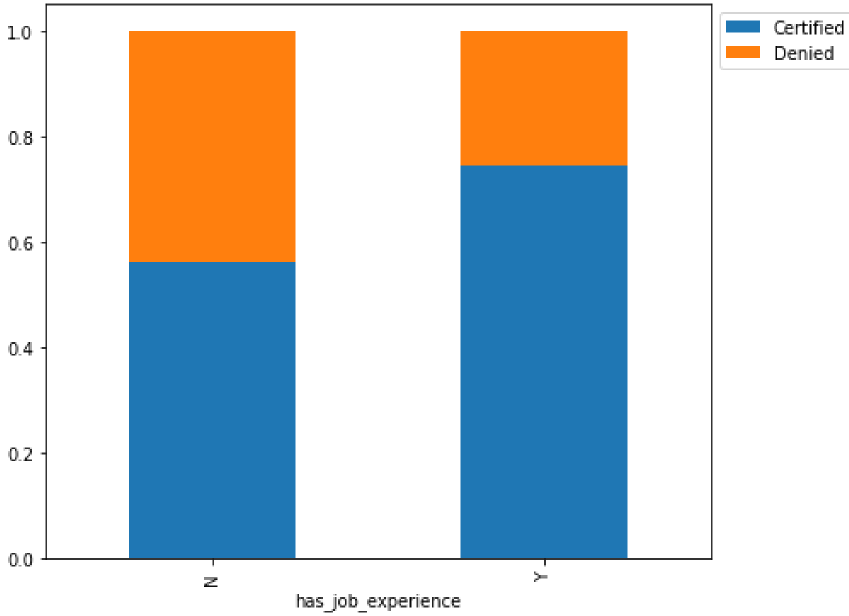
Observations

- The majority of employees are from Asia, where over 65% are certified.
- Across continents, the majority (over half) of employees seem to be certified, South America is a more even split between certified/denied.
- Across regions, besides islands, the split between certified/denied is balanced.
- The Midwest has the highest proportion of certified employees, followed by the South, then Northeast.

Question 3: Experienced professionals might look abroad for opportunities to improve their lifestyles and career development. Does work experience influence visa status?

In [39]:
```
stack(df, 'has_job_experience', 'case_status')
```

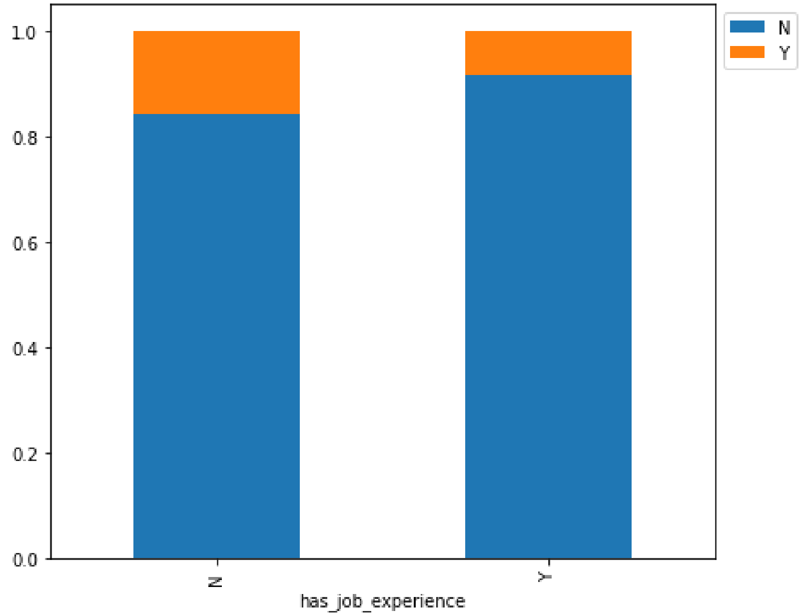
```
case_status         Certified  Denied    All
has_job_experience
All                     17018    8462  25480
N                        5994    4684  10678
Y                       11024    3778  14802
------------------------------------------------------------------------------------------------------------------------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



In [40]:

```
# job experience relationship to requiring job training

stack(df, 'has_job_experience', 'requires_job_training')
```

```
requires_job_training      N     Y    All
has_job_experience
All                    22525  2955  25480
N                       8988  1690  10678
Y                      13537  1265  14802
------------------------------------------------------------------------------------------------------------------------
```



In [41]:
```
pd.crosstab(df['has_job_experience'], df['requires_job_training'])
```

Out[41]:

requires_job_training	N	Y
has_job_experience		
N	8988	1690
Y	13537	1265

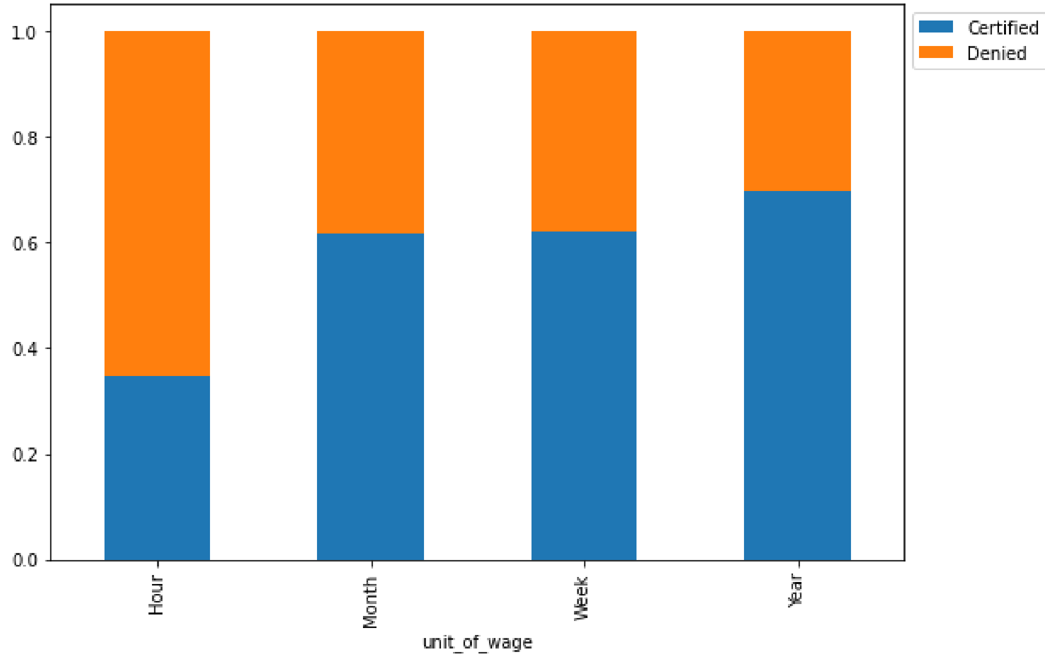
Observations

- For those with job experience, around 75% are certified, versus 56% of those without job experience.
- Unsurprisingly, those with job experience are in the majority for not requiring job training.

Question 4: In the United States, employees are paid at different intervals. Which pay unit is most likely to be certified for a visa?

In [42]:
```
stack(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
------------------------------------------------------------------------------------------------------------------------
```

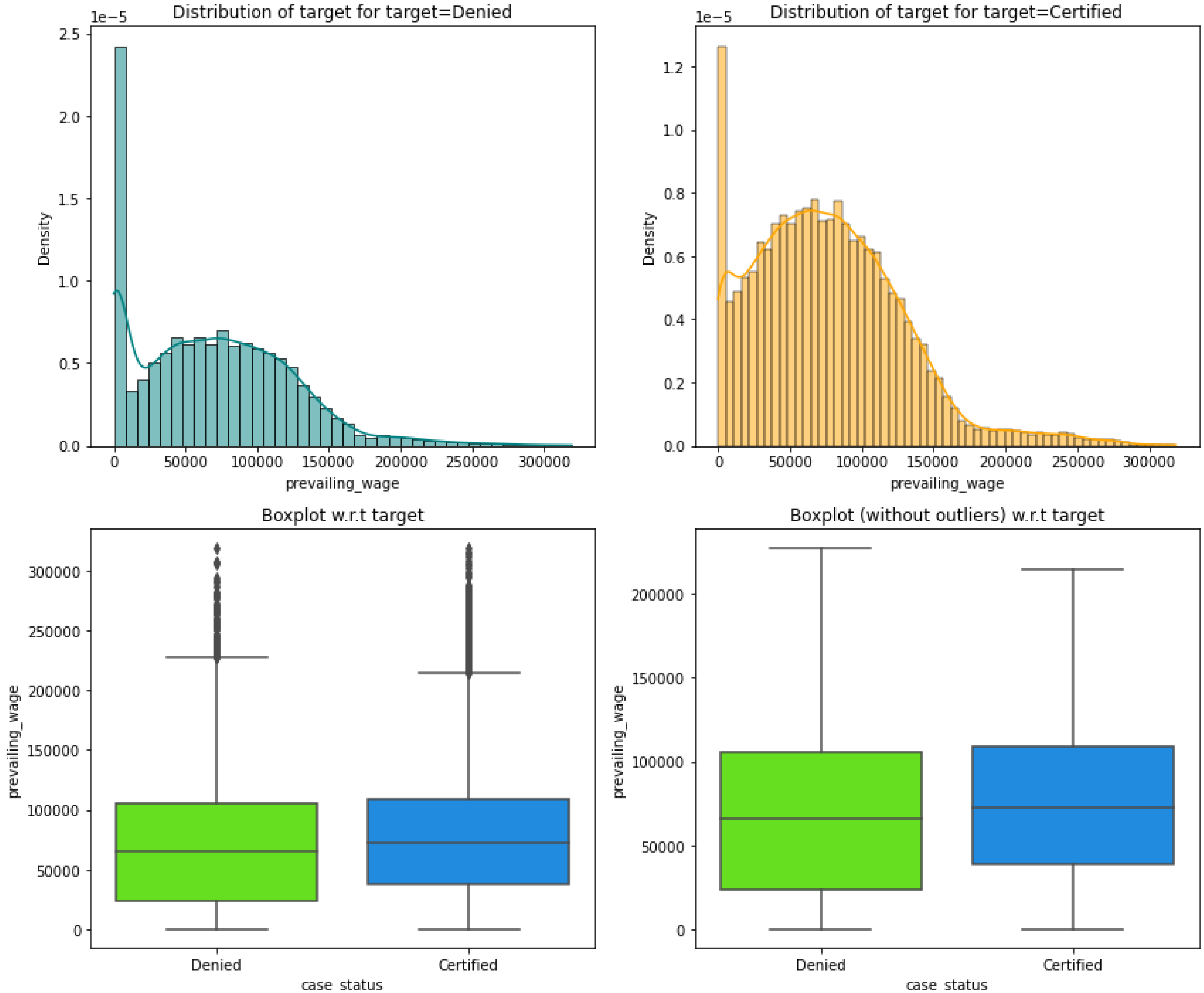


Observations

- Those paid on a yearly basis have the highest rate of certification, and hourly basis the lowest.

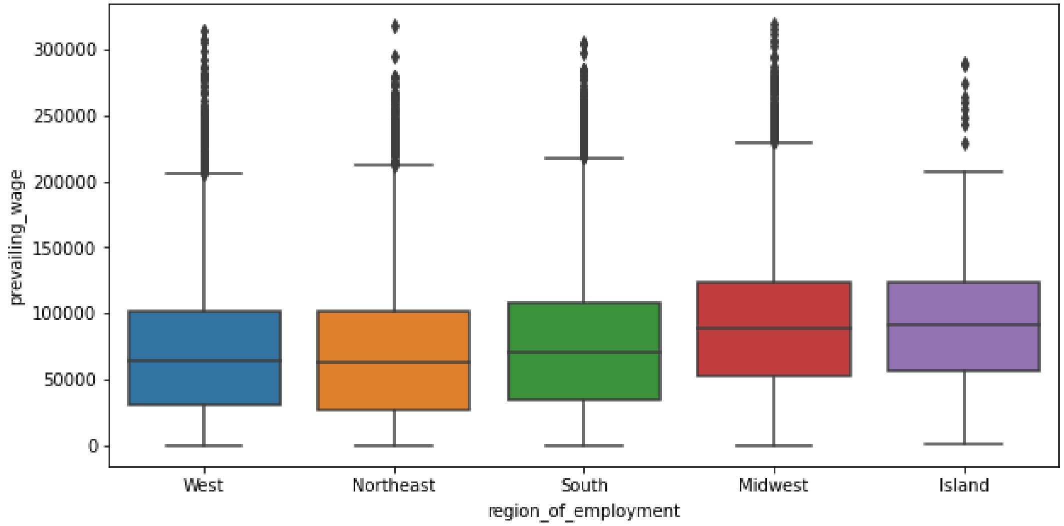
Question 5: The US government has established a prevailing wage to protect local talent and foreign workers. How does the visa status change with the prevailing wage?

In [43]:
```
dist_target(df, 'prevailing_wage', 'case_status')
```



In [44]:

```
# region of employment vs prevailing wage

plt.figure(figsize=(10, 5))
sns.boxplot(data=df, x="region_of_employment", y="prevailing_wage")
plt.show()
```



Observations

- Firstly, we can see from the bar graphs that there are many outliers on the lower range for prevailing wage.
- The prevailing wage for those denied is significantly lower than those certified.
- The Midwest has the highest proportion of certified employees, followed by the South, then Northeast. Prevailing wage for each region also follows this pattern, lowering each time the proportion of certified employees lowers.

Data Preprocessing

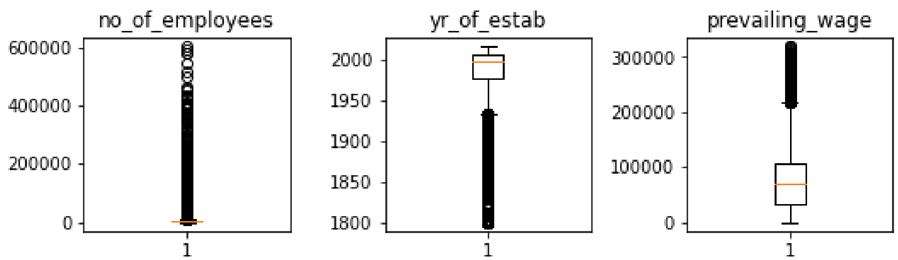
- Missing value treatment (not needed, checked above none missing or duplicates)
- Outlier detection and treatment (if needed)
- Preparing data for modeling
- Any other preprocessing steps (if needed)

Outlier Detection and Treatment

In [45]:

```
# outlier detection using boxplot

num_cols = df.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(10, 8))

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

```
plt.show()
```



In [46]:
```
df['yr_of_estab'].describe()
```

Out[46]:
```
count    25480.000000
mean      1979.409929
std         42.366929
min       1800.000000
25%       1976.000000
50%       1997.000000
75%       2005.000000
max       2016.000000
Name: yr_of_estab, dtype: float64
```

In [47]:
```
# many companies established on the lower outlier range year, let's see what some of it looks like

old = df[df["yr_of_estab"] < 1975]

old.head()
```

Out[47]:

	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees	yr_of_estab	region_of_employment	prevailing_wage	unit_of_
3	Asia	Bachelor's	N	N	98	1897	West	83434.0300	
7	North America	Bachelor's	Y	N	3035	1924	West	418.2298	
12	Asia	Bachelor's	Y	N	123876	1963	Northeast	28663.0500	
19	Asia	Doctorate	N	N	843	1972	Midwest	79948.1200	
22	Asia	Master's	Y	N	2878	1968	West	45642.3900	

Observations:

- There are quite a few outliers in the data, notably in year established and number of employees.
- However, since they are proper values and reflect the distribution of employees, we will not treat them.

Building bagging and boosting models

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

In [48]:
```
X = df.drop('case_status', axis=1)
y = df["case_status"].apply(lambda x: 1 if x == "Certified" else 0)
```

In [49]:
```
# create dummy variables

X = pd.get_dummies(X, columns=X.select_dtypes(include=["object", "category"]).columns.tolist(), drop_first=True)

X.head()
```

Out[49]:

	no_of_employees	yr_of_estab	prevailing_wage	continent_Asia	continent_Europe	continent_North America	continent_Oceania	continent_South America	education_of_employe
0	14513	2007	592.2029	1	0	0	0	0	
1	2412	2002	83425.6500	1	0	0	0	0	
2	44444	2008	122996.8600	1	0	0	0	0	
3	98	1897	83434.0300	1	0	0	0	0	
4	1082	2005	149907.3900	0	0	0	0	0	

In [50]:
```
# splitting the data in 70:30 ratio for train to test data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

In [51]:
```
print("Shape of Training set : ", X_train.shape)
print("Shape of test set : ", X_test.shape)
print("Percentage of classes in training set:")
print(y_train.value_counts(normalize=True))
print("Percentage of classes in test set:")
print(y_test.value_counts(normalize=True))
```

```
Shape of Training set :  (17836, 21)
Shape of test set :  (7644, 21)
Percentage of classes in training set:
1    0.663602
0    0.336398
Name: case_status, dtype: float64
Percentage of classes in test set:
1    0.677917
0    0.322083
Name: case_status, dtype: float64
```

Model evaluation criterion

Model can make wrong predictions as:

1. Model predicts that the visa application will get certified but in reality, the visa application should get denied. (false negative, type 2 error)
2. Model predicts that the visa application will not get certified but in reality, the visa application should get certified. (false positive, type 1 error)

Which case is more important?

- Both the cases are important as:

- If a visa is certified when it had to be denied a wrong employee will get the job position while US citizens will miss the opportunity to work on that position.

- If a visa is denied when it had to be certified the U.S. will lose a suitable human resource that can contribute to the economy.

How to reduce the losses?

- `F1 Score` can be used a the metric for evaluation of the model, greater the F1 score higher are the chances of minimizing False Negatives and False Positives.
- We will use balanced class weights so that model focuses equally on both classes.

Let's define a function to provide metric scores on the train and test set and a function to show confusion matrix so that we do not have to use the same code repetitively while evaluating models.

In [52]:

```
# defining a function to compute different metrics to check performance of a classification model built using sklearn
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,
        },
```

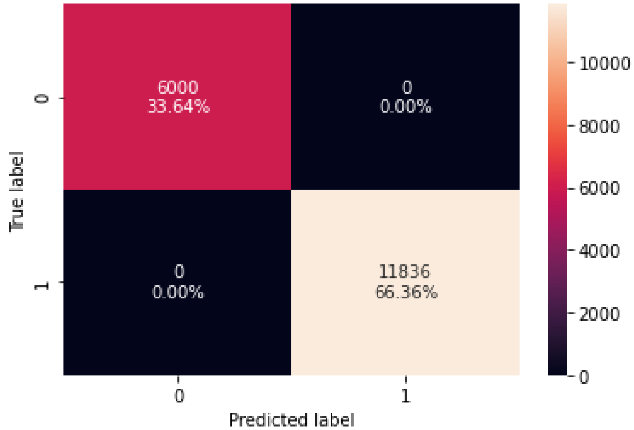
```
        index=[0],
    )

    return df_perf

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")
```

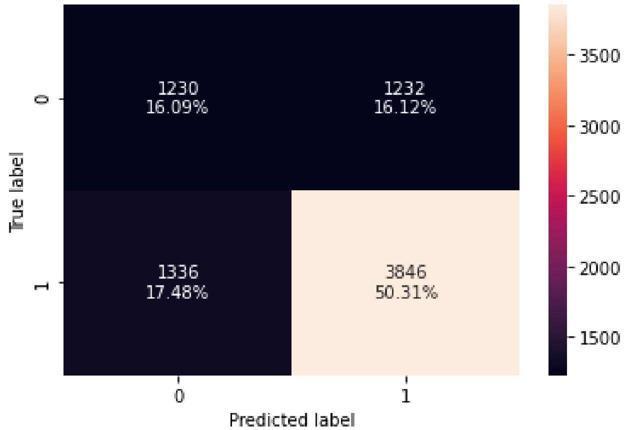
Decision Tree Model

In [53]:

```
#Fitting the model
d_tree = DecisionTreeClassifier(random_state=1)
d_tree.fit(X_train,y_train)

#Calculating different metrics
d_tree_model_train_perf=model_performance_classification_sklearn(d_tree,X_train,y_train)
print("Training performance:\n",d_tree_model_train_perf)

#Creating confusion matrix
confusion_matrix_sklearn(d_tree,X_train,y_train)

d_tree_model_test_perf=model_performance_classification_sklearn(d_tree,X_test,y_test)
print("Testing performance:\n",d_tree_model_test_perf)

confusion_matrix_sklearn(d_tree,X_test,y_test)
```

```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
Testing performance:
    Accuracy    Recall  Precision        F1
0   0.66405  0.742184   0.757385  0.749708
```





- The decision tree is overfitting the training data.
- Let's try hyperparameter tuning and see if the model performance improves.

Hyperparameter Tuning (Decision Tree)

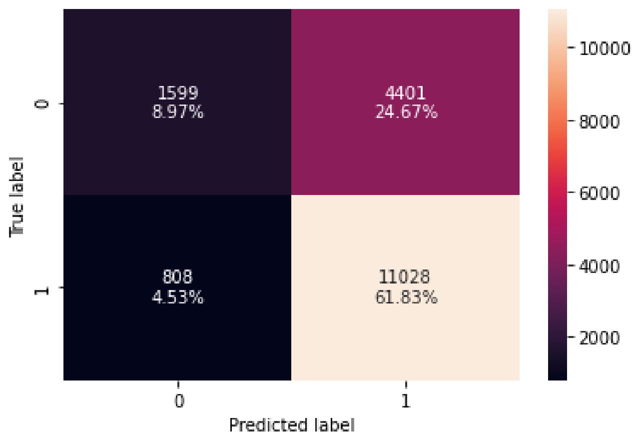
In [54]:
```
# Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(class_weight="balanced", random_state=1)

# Grid of parameters to choose from
parameters = {
    "max_depth": np.arange(10, 30, 5),
    "min_samples_leaf": [3, 5, 7],
    "max_leaf_nodes": [2, 3, 5],
    "min_impurity_decrease": [0.0001, 0.001],
}

# Type of scoring used to compare parameter combinations
```

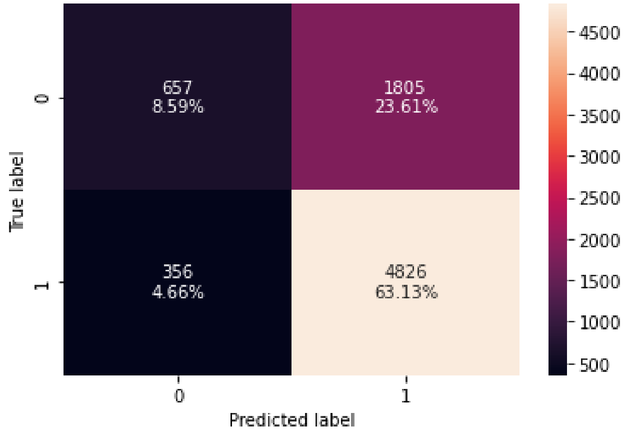
```
scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(dtree_estimator, parameters, scoring=scorer,n_jobs=-1)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
dtree_estimator = grid_obj.best_estimator_

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

Out[54]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=10, max_leaf_nodes=2,
                       min_impurity_decrease=0.0001, min_samples_leaf=3,
                       random_state=1)
```

In [55]:
```
#Calculating different metrics
dtree_estimator_model_train_perf=model_performance_classification_sklearn(d_tree,X_train,y_train)
print("Training performance:\n",dtree_estimator_model_train_perf)

#Creating confusion matrix
confusion_matrix_sklearn(dtree_estimator,X_train,y_train)

dtree_estimator_model_test_perf=model_performance_classification_sklearn(d_tree,X_test,y_test)
print("Testing performance:\n",dtree_estimator_model_test_perf)

#Creating confusion matrix
confusion_matrix_sklearn(dtree_estimator,X_test,y_test)
```

```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
Testing performance:
    Accuracy    Recall  Precision        F1
0   0.66405  0.742184   0.757385  0.749708
```





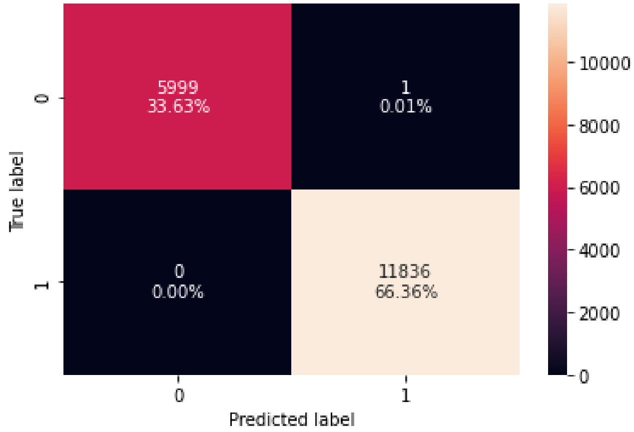
- The overfitting has reduced but the test f1-score has also decreased.
- Let's try some other models.

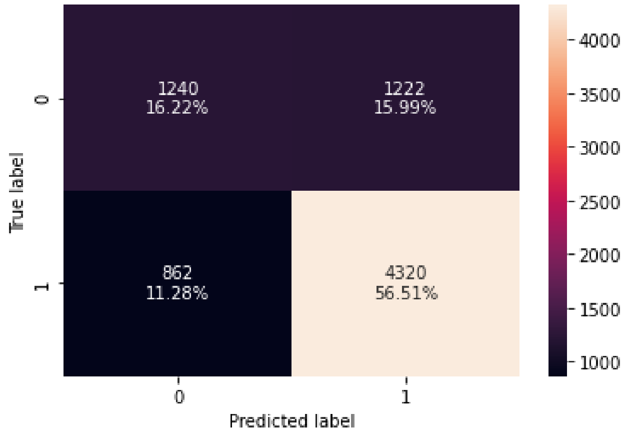
Random Forest Classifier

In [56]:

```
#Fitting the model
rf_estimator = RandomForestClassifier(random_state=1)
rf_estimator.fit(X_train,y_train)

#Calculating different metrics
rf_estimator_model_train_perf=model_performance_classification_sklearn(rf_estimator,X_train,y_train)
print("Training performance:\n",rf_estimator_model_train_perf)

#Creating confusion matrix
confusion_matrix_sklearn(rf_estimator,X_train,y_train)

rf_estimator_model_test_perf=model_performance_classification_sklearn(rf_estimator,X_test,y_test)
print("Testing performance:\n",rf_estimator_model_test_perf)

confusion_matrix_sklearn(rf_estimator,X_test,y_test)
```

```
Training performance:
    Accuracy  Recall  Precision        F1
0  0.999944     1.0   0.999916  0.999958
Testing performance:
    Accuracy    Recall  Precision       F1
0  0.727368  0.833655   0.779502  0.80567
```





- Random forest is giving higher test accuracy, recall, precision, and f1 compared to decision trees.
- Still overfitting the data, let's try hyperparameter tuning and see if the model performance improves.

Hyperparameter Tuning (Random Forest)

In [57]:

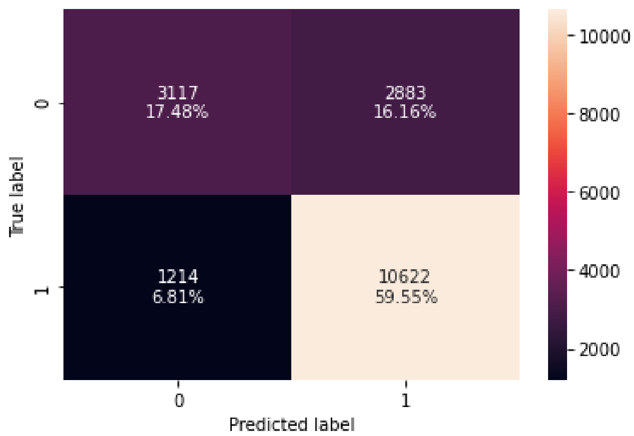
```
# Choose the type of classifier.
rf_tuned = RandomForestClassifier(random_state=1, oob_score=True, bootstrap=True)

parameters = {
    "max_depth": list(np.arange(5, 15, 5)),
    "max_features": ["sqrt", "log2"],
    "min_samples_split": [3, 5, 7],
    "n_estimators": np.arange(10, 40, 10),
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.f1_score)
```

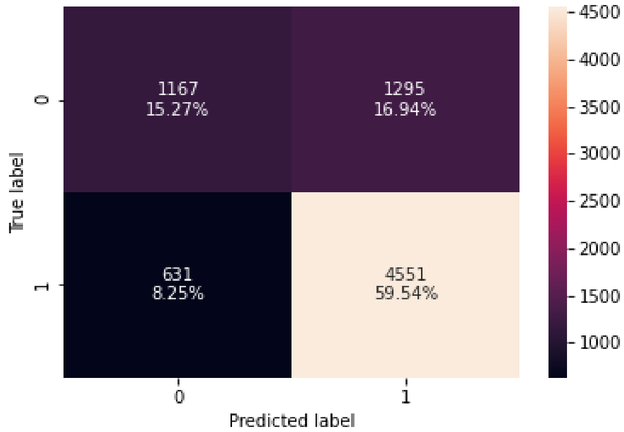
```
# Run the grid search
grid_obj = GridSearchCV(rf_tuned, parameters, scoring=scorer, cv=5,n_jobs=-1)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
rf_tuned = grid_obj.best_estimator_

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

Out[57]:
```
RandomForestClassifier(max_depth=10, max_features='sqrt', min_samples_split=5,
                       n_estimators=30, oob_score=True, random_state=1)
```

In [58]:
```
#Calculating different metrics
rf_tuned_model_train_perf=model_performance_classification_sklearn(rf_tuned,X_train,y_train)
print("Training performance:\n",rf_tuned_model_train_perf)
confusion_matrix_sklearn(rf_tuned,X_train,y_train)

rf_tuned_model_test_perf=model_performance_classification_sklearn(rf_tuned,X_test,y_test)
print("Testing performance:\n",rf_tuned_model_test_perf)
confusion_matrix_sklearn(rf_tuned,X_test,y_test)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.770296  0.897432   0.786524  0.838325
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.748038  0.878232   0.778481  0.825354
```



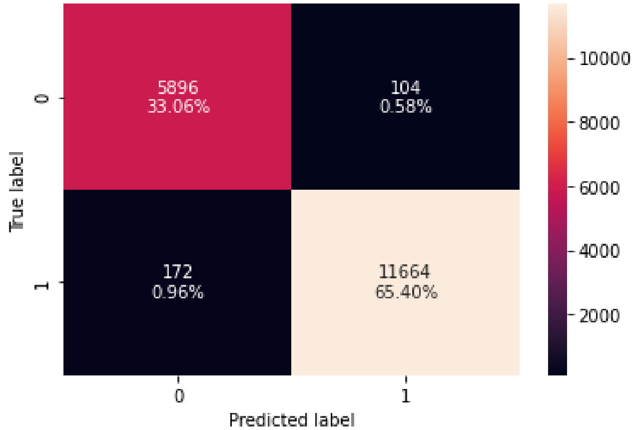


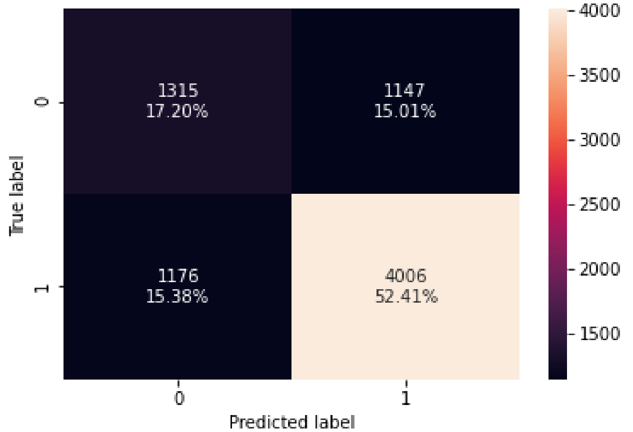
- Model performance has improved and overfitting has reduced.
- Test recall and f1 have increased.

Bagging Classifier

In [59]:
```
#Fitting the model
bagging_classifier = BaggingClassifier(random_state=1)
bagging_classifier.fit(X_train,y_train)

#Calculating different metrics
bagging_classifier_model_train_perf=model_performance_classification_sklearn(bagging_classifier,X_train,y_train)
print(bagging_classifier_model_train_perf)

#Creating confusion matrix
confusion_matrix_sklearn(bagging_classifier,X_train,y_train)

bagging_classifier_model_test_perf=model_performance_classification_sklearn(bagging_classifier,X_test,y_test)
print(bagging_classifier_model_test_perf)
confusion_matrix_sklearn(bagging_classifier,X_test,y_test)
```

```
   Accuracy    Recall  Precision        F1
0  0.984526  0.985468   0.991162  0.988307
   Accuracy    Recall  Precision       F1
0  0.696102  0.773061   0.777411  0.77523
```





- Let's try hyperparameter tuning and see if the model performance improves.

Hyperparameter Tuning (Bagging Classifier)

In [60]:

```
# Choose the type of classifier.
bagging_estimator_tuned = BaggingClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    "max_features": [0.7, 0.8, 0.9],
    "n_estimators": [90,100]
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
```

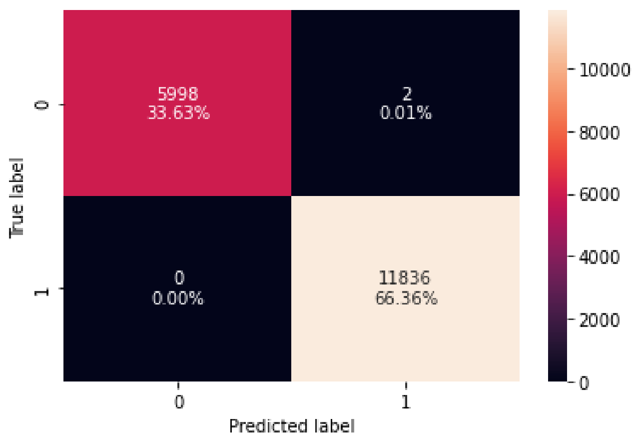
```
grid_obj = GridSearchCV(bagging_estimator_tuned, parameters, scoring=scorer,cv=5,n_jobs=-1)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
bagging_estimator_tuned = grid_obj.best_estimator_

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

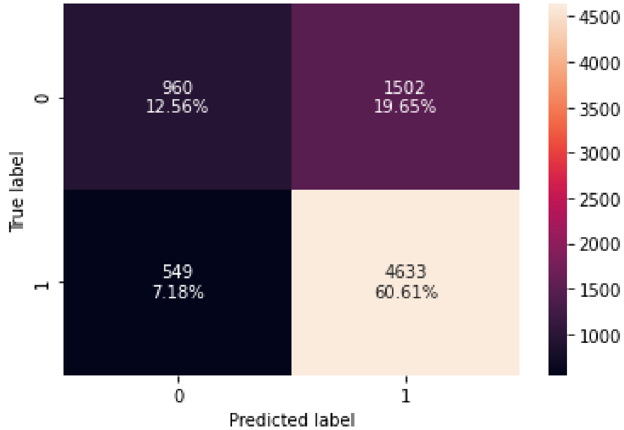
Out[60]: BaggingClassifier(max_features=0.7, n_estimators=100, random_state=1)

In [61]:

```
#Calculating different metrics
bagging_estimator_tuned_model_train_perf=model_performance_classification_sklearn(bagging_estimator_tuned,X_train,y_train)
print(bagging_estimator_tuned_model_train_perf)

#Creating confusion matrix
confusion_matrix_sklearn(bagging_estimator_tuned,X_train,y_train)

bagging_estimator_tuned_model_test_perf=model_performance_classification_sklearn(bagging_estimator_tuned,X_test,y_test)
print(bagging_estimator_tuned_model_test_perf)
confusion_matrix_sklearn(bagging_estimator_tuned,X_test,y_test)
```

```
   Accuracy  Recall  Precision        F1
0  0.999888     1.0   0.999831  0.999916
   Accuracy    Recall  Precision        F1
0  0.731685  0.894056   0.755175  0.818768
```





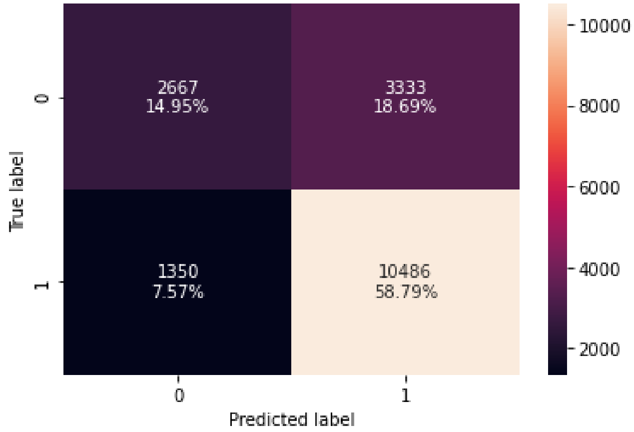
- The test accuracy, recall, precision, and f1 score are all higher.

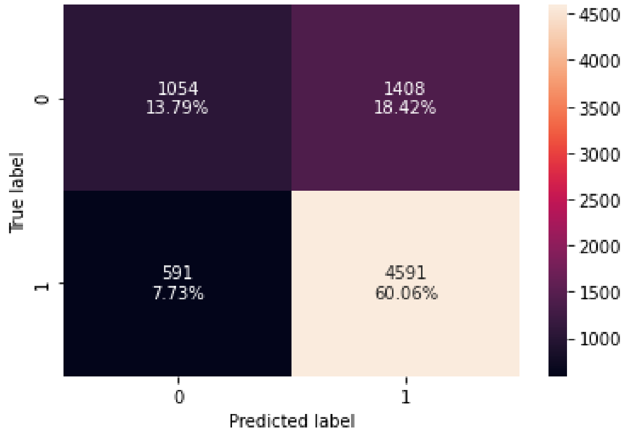
AdaBoost Classifier

In [62]:

```
#Fitting the model
ab_classifier = AdaBoostClassifier(random_state=1)
ab_classifier.fit(X_train,y_train)

#Calculating different metrics
ab_classifier_model_train_perf=model_performance_classification_sklearn(ab_classifier,X_train,y_train)
print(ab_classifier_model_train_perf)

#Creating confusion matrix
confusion_matrix_sklearn(ab_classifier,X_train,y_train)

ab_classifier_model_test_perf=model_performance_classification_sklearn(ab_classifier,X_test,y_test)
print(ab_classifier_model_test_perf)
confusion_matrix_sklearn(ab_classifier,X_test,y_test)
```

```
   Accuracy    Recall  Precision        F1
0  0.737441  0.885941    0.75881  0.817462
   Accuracy    Recall  Precision        F1
0  0.738488  0.885951   0.765294  0.821215
```





- Adaboost is giving more generalized performance than previous models.

Hyperparameter Tuning (Adaboost)

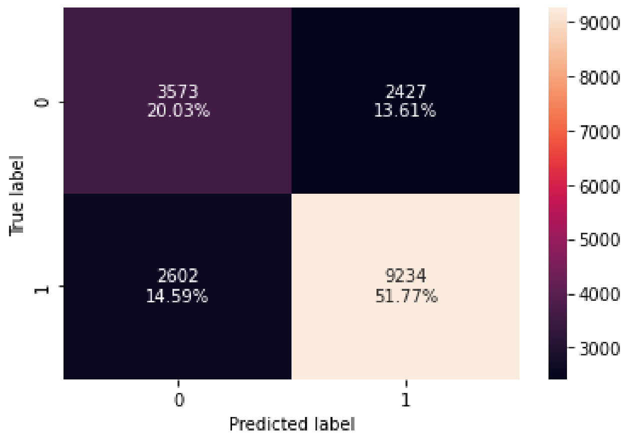
In [63]:
```
# Choose the type of classifier.
abc_tuned = AdaBoostClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    # Let's try different max_depth for base_estimator
    "base_estimator": [
        DecisionTreeClassifier(max_depth=1, class_weight="balanced", random_state=1),
        DecisionTreeClassifier(max_depth=2, class_weight="balanced", random_state=1),
        DecisionTreeClassifier(max_depth=3, class_weight="balanced", random_state=1),
    ],
    "n_estimators": np.arange(60, 100, 10),
    "learning_rate": np.arange(0.1, 0.4, 0.1),
}
```

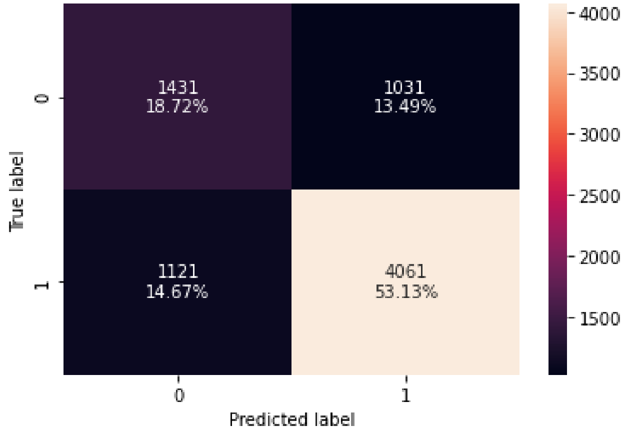
```python
# Type of scoring used to compare parameter  combinations
acc_scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(abc_tuned, parameters, scoring=scorer,cv=5,n_jobs=-1)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
abc_tuned = grid_obj.best_estimator_

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

Out[63]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(class_weight='balanced',
                                                         max_depth=1,
                                                         random_state=1),
                   learning_rate=0.1, n_estimators=90, random_state=1)
```

In [64]:
```python
#Calculating different metrics
abc_tuned_model_train_perf=model_performance_classification_sklearn(abc_tuned,X_train,y_train)
print(abc_tuned_model_train_perf)

#Creating confusion matrix
confusion_matrix_sklearn(abc_tuned,X_train,y_train)

abc_tuned_model_test_perf=model_performance_classification_sklearn(abc_tuned,X_test,y_test)
print(abc_tuned_model_test_perf)
confusion_matrix_sklearn(abc_tuned,X_test,y_test)
```

```
   Accuracy    Recall  Precision        F1
0  0.718042  0.780162    0.79187  0.785973
   Accuracy    Recall  Precision        F1
0  0.718472  0.783674   0.797526  0.790539
```



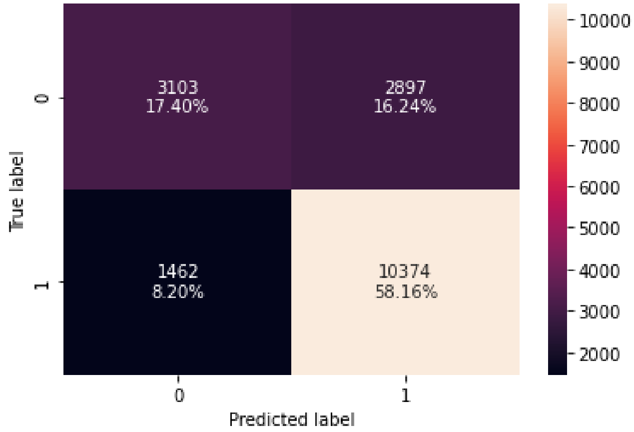


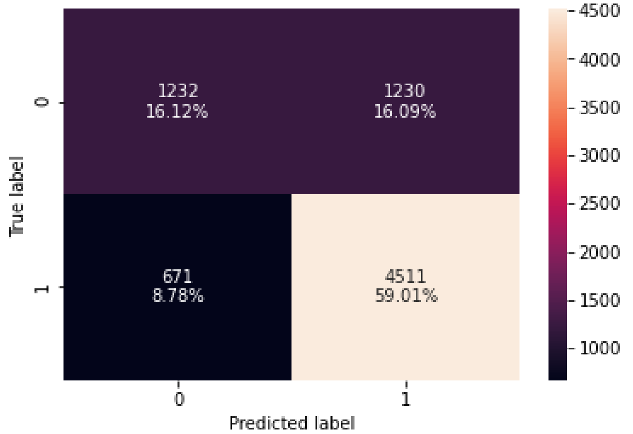
- Surprisingly, the model performance has decreased after hyperparameter tuning.

Gradient Boosting Classifier

In [65]:
```
#Fitting the model
gb_classifier = GradientBoostingClassifier(random_state=1)
gb_classifier.fit(X_train,y_train)

#Calculating different metrics
gb_classifier_model_train_perf=model_performance_classification_sklearn(gb_classifier,X_train,y_train)
print("Training performance:\n",gb_classifier_model_train_perf)

#Creating confusion matrix
confusion_matrix_sklearn(gb_classifier,X_train,y_train)

gb_classifier_model_test_perf=model_performance_classification_sklearn(gb_classifier,X_test,y_test)
print("Testing performance:\n",gb_classifier_model_test_perf)
confusion_matrix_sklearn(gb_classifier,X_test,y_test)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.755607  0.876479   0.781704  0.826383
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.751308  0.870513   0.785752  0.825964
```





- The gradient booster is overfitting the training data.

Hyperparameter Tuning (Gradient Boosting)

In [66]:

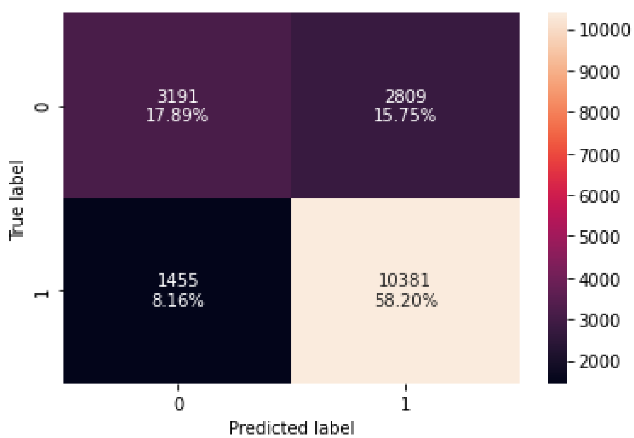
```
# Choose the type of classifier.
gbc_tuned = GradientBoostingClassifier(
    init=AdaBoostClassifier(random_state=1), random_state=1
)

# Grid of parameters to choose from
parameters = {
    "n_estimators": [200, 250],
    "subsample": [0.8, 0.9],
    "max_features": [0.8, 0.9],
    "learning_rate": [0.1, 0.2],
}
```

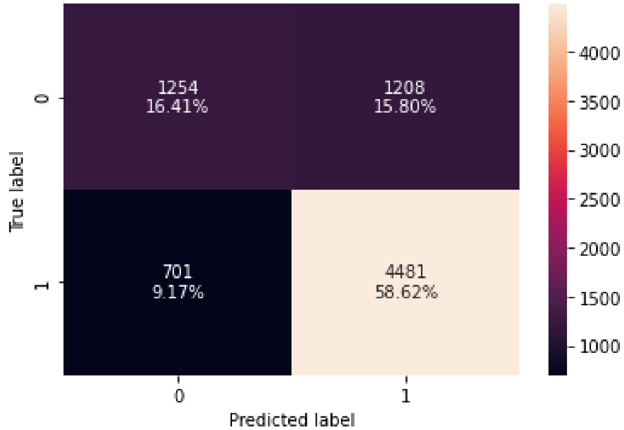
```
# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(gbc_tuned, parameters, scoring=scorer,cv=5,n_jobs=-1)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
gbc_tuned = grid_obj.best_estimator_

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

Out[66]:
```
GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                           max_features=0.9, n_estimators=200, random_state=1,
                           subsample=0.9)
```

In [67]:
```
#Calculating different metrics
gbc_tuned_model_train_perf=model_performance_classification_sklearn(gbc_tuned,X_train,y_train)
print("Training performance:\n",gbc_tuned_model_train_perf)

#Creating confusion matrix
confusion_matrix_sklearn(gbc_tuned,X_train,y_train)

gbc_tuned_model_test_perf=model_performance_classification_sklearn(gbc_tuned,X_test,y_test)
print("Testing performance:\n",gbc_tuned_model_test_perf)
confusion_matrix_sklearn(gbc_tuned,X_test,y_test)
```

```
Training performance:
    Accuracy   Recall  Precision        F1
0  0.760933  0.87707   0.787036  0.829617
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.750262  0.864724    0.78766  0.824395
```





- The overfitting has reduced slightly but there is not much difference in the model performance.

Stacking Classifier

In [68]:
```
estimators = [('Random Forest',rf_tuned), ('AdaBoost',abc_tuned), ('Decision Tree',dtree_estimator)]

final_estimator = gbc_tuned # gradient boosting

stacking_classifier= StackingClassifier(estimators=estimators,final_estimator=final_estimator)

stacking_classifier.fit(X_train,y_train)
```

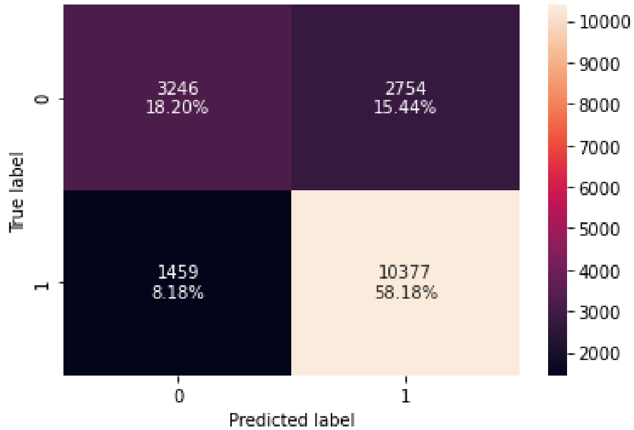
Out[68]:
```
StackingClassifier(estimators=[('Random Forest',
                                 RandomForestClassifier(max_depth=10,
                                                        max_features='sqrt',
                                                        min_samples_split=5,
                                                        n_estimators=30,
                                                        oob_score=True,
                                                        random_state=1)),
                                ('AdaBoost',
                                 AdaBoostClassifier(base_estimator=DecisionTreeClassifier(class_weight='balanced',
                                                                                          max_depth=1,
                                                                                          random_state=1),
                                                    learning_rate=0.1,
                                                    n_estimators=90,
                                                    random_state=1)),
                                ('Decision Tree',
                                 DecisionTreeClassifier(class_weight='balanced',
                                                        max_depth=10,
                                                        max_leaf_nodes=2,
                                                        min_impurity_decrease=0.0001,
                                                        min_samples_leaf=3,
                                                        random_state=1))],
                   final_estimator=GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                                                              max_features=0.9,
```

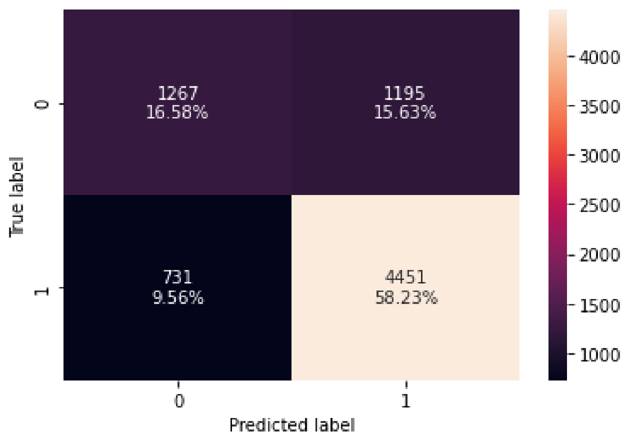
```
                                   n_estimators=200,
                                   random_state=1,
                                   subsample=0.9))
```

In [69]:

```
#Calculating different metrics
stacking_classifier_model_train_perf=model_performance_classification_sklearn(stacking_classifier,X_train,y_train)
print("Training performance:\n",stacking_classifier_model_train_perf)

#Creating confusion matrix
confusion_matrix_sklearn(stacking_classifier,X_train,y_train)

stacking_classifier_model_test_perf=model_performance_classification_sklearn(stacking_classifier,X_test,y_test)
print("Testing performance:\n",stacking_classifier_model_test_perf)
confusion_matrix_sklearn(stacking_classifier,X_test,y_test)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.763792  0.876732   0.790267  0.831257
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.748038  0.858935   0.788346  0.822128
```





- The stacking classifier is giving a similar performance to the gradient boosting.
- The confusion matrix shows that the model is better at identifying certified visas (coded 1 earlier)

Model Performance Comparison

In [70]:
```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        dtree_estimator_model_train_perf.T,
        dtree_estimator_model_train_perf.T,
        bagging_classifier_model_train_perf.T,
        bagging_estimator_tuned_model_train_perf.T,
        rf_estimator_model_train_perf.T,
        rf_tuned_model_train_perf.T,
        ab_classifier_model_train_perf.T,
        abc_tuned_model_train_perf.T,
        gb_classifier_model_train_perf.T,
        gbc_tuned_model_train_perf.T,
        stacking_classifier_model_train_perf.T,
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Decision Tree",
    "Tuned Decision Tree",
    "Bagging Classifier",
    "Tuned Bagging Classifier",
    "Random Forest",
    "Tuned Random Forest",
    "Adaboost Classifier",
    "Tuned Adaboost Classifier",
    "Gradient Boost Classifier",
    "Tuned Gradient Boost Classifier",
    "Stacking Classifier",
]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[70]:

	Decision Tree	Tuned Decision Tree	Bagging Classifier	Tuned Bagging Classifier	Random Forest	Tuned Random Forest	Adaboost Classifier	Tuned Adaboost Classifier	Gradient Boost Classifier	Tuned Gradient Boost Classifier	Stacking Classifier
Accuracy	1.0	1.0	0.984526	0.999888	0.999944	0.770296	0.737441	0.718042	0.755607	0.760933	0.763792
Recall	1.0	1.0	0.985468	1.000000	1.000000	0.897432	0.885941	0.780162	0.876479	0.877070	0.876732
Precision	1.0	1.0	0.991162	0.999831	0.999916	0.786524	0.758810	0.791870	0.781704	0.787036	0.790267

	Decision Tree	Tuned Decision Tree	Bagging Classifier	Tuned Bagging Classifier	Random Forest	Tuned Random Forest	Adaboost Classifier	Tuned Adaboost Classifier	Gradient Boost Classifier	Tuned Gradient Boost Classifier	Stacking Classifier
F1	1.0	1.0	0.988307	0.999916	0.999958	0.838325	0.817462	0.785973	0.826383	0.829617	0.831257

In [71]:

```
# testing performance comparison

models_test_comp_df = pd.concat(
    [
        dtree_estimator_model_test_perf.T,
        dtree_estimator_model_test_perf.T,
        bagging_classifier_model_test_perf.T,
        bagging_estimator_tuned_model_test_perf.T,
        rf_estimator_model_test_perf.T,
        rf_tuned_model_test_perf.T,
        ab_classifier_model_test_perf.T,
        abc_tuned_model_test_perf.T,
        gb_classifier_model_test_perf.T,
        gbc_tuned_model_test_perf.T,
        stacking_classifier_model_test_perf.T,
    ],
    axis=1,
)
models_test_comp_df.columns = [
    "Decision Tree",
    "Tuned Decision Tree",
    "Bagging Classifier",
    "Tuned Bagging Classifier",
    "Random Forest",
    "Tuned Random Forest",
    "Adaboost Classifier",
    "Tuned Adaboost Classifier",
    "Gradient Boost Classifier",
    "Tuned Gradient Boost Classifier",
    "Stacking Classifier",
]
print("Testing performance comparison:")
models_test_comp_df
```

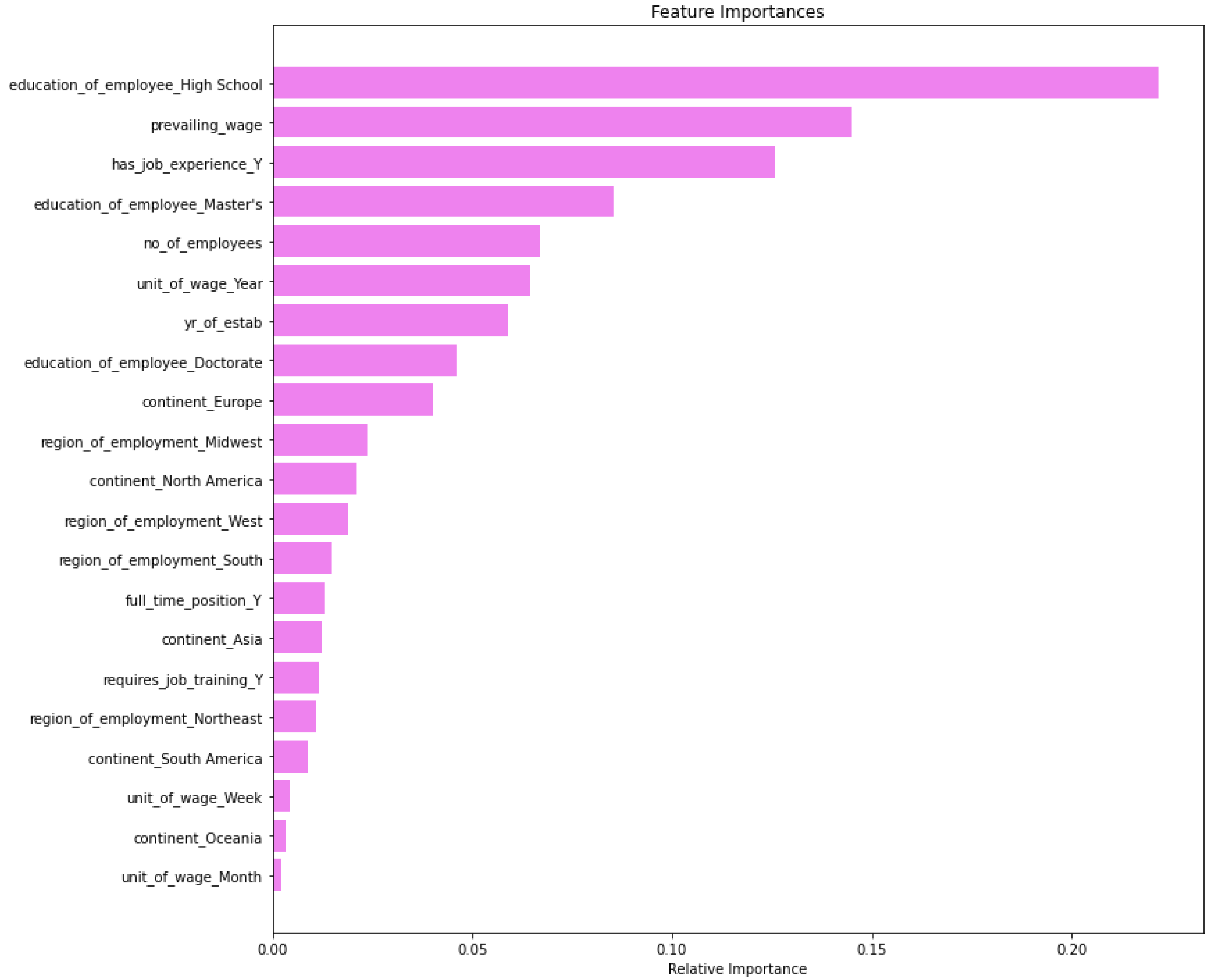
```
Testing performance comparison:
```

Out[71]:

	Decision Tree	Tuned Decision Tree	Bagging Classifier	Tuned Bagging Classifier	Random Forest	Tuned Random Forest	Adaboost Classifier	Tuned Adaboost Classifier	Gradient Boost Classifier	Tuned Gradient Boost Classifier	Stacking Classifier
Accuracy	0.664050	0.664050	0.696102	0.731685	0.727368	0.748038	0.738488	0.718472	0.751308	0.750262	0.748038
Recall	0.742184	0.742184	0.773061	0.894056	0.833655	0.878232	0.885951	0.783674	0.870513	0.864724	0.858935
Precision	0.757385	0.757385	0.777411	0.755175	0.779502	0.778481	0.765294	0.797526	0.785752	0.787660	0.788346
F1	0.749708	0.749708	0.775230	0.818768	0.805670	0.825354	0.821215	0.790539	0.825964	0.824395	0.822128

In [72]:

```
### Important features of the final model

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

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



- A high school education of an employee is the most important feature in determining visa certification followed by prevailing wage, and whether they do have job experience.

Actionable Insights and Recommendations

- Based on our analysis, we can say that employees that receive cerfications versus those denied have the following features:
  - a high school diploma
  - a high prevailing wage (might correspond to higher paying jobs in general, ex. engineer)
  - existing past job experience
  - medium to large sized company/# of employees
  - a Master's degree
  - paid in a yearly salary
  - a Doctorate
  - from the following continents in order of importance: Europe, North America, Asia, South America, Ocenia
    - Surprising, as most employees in the dataset are from Asia
  - from the following regions in order of importance: Midwest, West, South, Northeast
    - Also surprising, majority regions in dataset are Northeast, Midwest, and South
- EasyVisa, in order to create a suitable profile for visa applicants that should be approved, should produce a profile that focuses on the above features that significantly influence case status.
  - Consider education level on their applications, especially if they have pursued higher education.
  - Make note of their previous work experiences, salary, and company recognition as well.
  - Since they were the two most inflated continents for confirmed certification, if the applicant is from Europe or North America, or from the regions of the Midwest or South, make a note to look more closely.

Considering the above features will help streamline the OFLC's screening process for visa certifications.