In [1]:
```
# Here I import all packages I will need
import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from statsmodels.tools.tools import add_constant
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifie
from xgboost import XGBClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, accur
from sklearn.model_selection import GridSearchCV
from scipy.stats import uniform
from sklearn import linear_model, datasets
from sklearn.model_selection import RandomizedSearchCV
```

In [2]:
```
# Here I am loading in my csv file into a dataframe
df = pd.read_csv('/Users/conne/Downloads/EasyVisa.csv')
```

In [3]:
```
# Here I creata a copy in order to preserve my original data
data = df.copy()
```

In [4]:
```
data.head()
```

Out[4]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_training	no
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 [5]:
```
data.tail()
```

Out[5]:

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

We see we have 12 different variables but only 11 of them will relevent as case_id has no statistical value.

In [6]:
```
data.shape
```

Out[6]:
```
(25480, 12)
```

In [7]:
```
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
```

As we can see we have no missing values but we do have 9 columns that are objects that will need to be converted to categorical variables.

In [8]:
```
cols = data.select_dtypes(['object'])
cols.columns
```

Out[8]:
```
Index(['case_id', 'continent', 'education_of_employee', 'has_job_experien
ce',
       'requires_job_training', 'region_of_employment', 'unit_of_wage',
       'full_time_position', 'case_status'],
      dtype='object')
```

This is a list of all colummns we'll need to convert

In [9]:
```
data.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	
no_of_employees	25480.0	5667.043210	22877.928848	-26.0000	1022.00	2109.00	35
yr_of_estab	25480.0	1979.409929	42.366929	1800.0000	1976.00	1997.00	20
prevailing_wage	25480.0	74455.814592	52815.942327	2.1367	34015.48	70308.21	1077

no_of_employees may have a mean that is skewed due to some large outliers. These outliers shouldn't be removed though as they are valid data points. The yr_of_estab ranges between 1800 and 2016 with an average of 1979. prevailing_wage has has an unusally small min when compared to the other numbers in the 5 number summary. This min should be examined futher.

In [10]:

```
print(df.value_counts(subset='continent'))
print(df.value_counts(subset='education_of_employee'))
print(df.value_counts(subset='has_job_experience'))
print(df.value_counts(subset='requires_job_training'))
print(df.value_counts(subset='no_of_employees'))
print(df.value_counts(subset='yr_of_estab'))
print(df.value_counts(subset='region_of_employment'))
print(df.value_counts(subset='prevailing_wage'))
print(df.value_counts(subset='unit_of_wage'))
print(df.value_counts(subset='full_time_position'))
print(df.value_counts(subset='case_status'))
```

```
continent
Asia             16861
Europe            3732
North America     3292
South America      852
Africa             551
Oceania            192
dtype: int64
education_of_employee
Bachelor's     10234
Master's        9634
High School     3420
Doctorate       2192
dtype: int64
has_job_experience
Y    14802
N    10678
dtype: int64
requires_job_training
N    22525
Y     2955
dtype: int64
no_of_employees
183       18
854       16
724       16
766       15
1476      15
          ..
6129       1
6130       1
6137       1
6138       1
602069     1
Length: 7105, dtype: int64
yr_of_estab
1998    1134
2005    1051
2001    1017
2007     994
1999     870
        ...
1807       6
1822       4
1846       4
1810       3
1824       2
Length: 199, dtype: int64
region_of_employment
Northeast    7195
South        7017
West         6586
Midwest      4307
Island        375
dtype: int64
prevailing_wage
60948.15     2
```

```
88664.77     2
82560.28     2
105.96       2
87751.88     2
            ..
46738.47     1
46727.57     1
46725.85     1
46719.75     1
319210.27    1
Length: 25454, dtype: int64
unit_of_wage
Year     22962
Hour      2157
Week       272
Month       89
dtype: int64
full_time_position
Y    22773
N     2707
dtype: int64
case_status
Certified    17018
Denied        8462
dtype: int64
```

We can see a majority of positions are full time. Most people were payed a salary wage. The Workers are fairly evenly spread except for the Island region. 1999 was the year with the highest number of businesses were established with 1134 busnesses being established that year.

In [11]:

```python
for feature in data.columns:
    if data[feature].dtype == 'object':
        data[feature] = pd.Categorical(data[feature])
data.head(10)
```

Out[11]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_training	no
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	
5	EZYV06	Asia	Master's	Y	N	
6	EZYV07	Asia	Bachelor's	N	N	
7	EZYV08	North America	Bachelor's	Y	N	
8	EZYV09	Asia	Bachelor's	N	N	
9	EZYV10	Europe	Doctorate	Y	N	

Here I convert all objects in categories.

In [12]:
```
def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=Non
    """
    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 val
    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
```

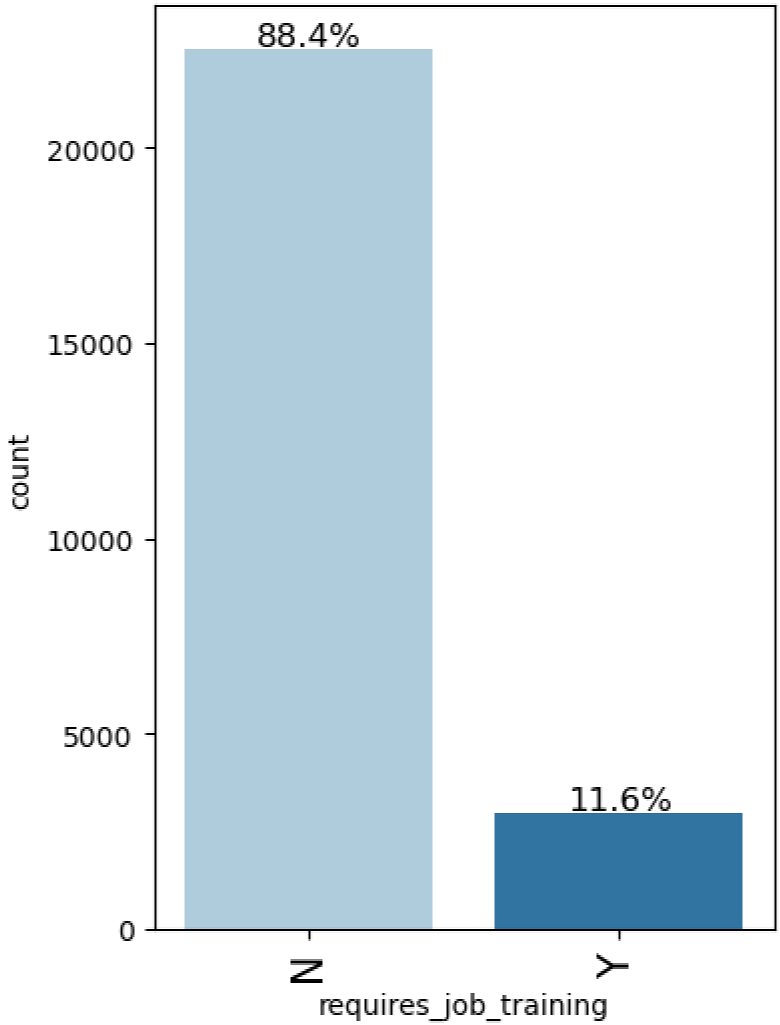
In [13]:

```
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 Fals
    n: displays the top n category levels (default is None, i.e., display
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 2, 6))
    else:
        plt.figure(figsize=(n + 2, 6))

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

    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 [14]:

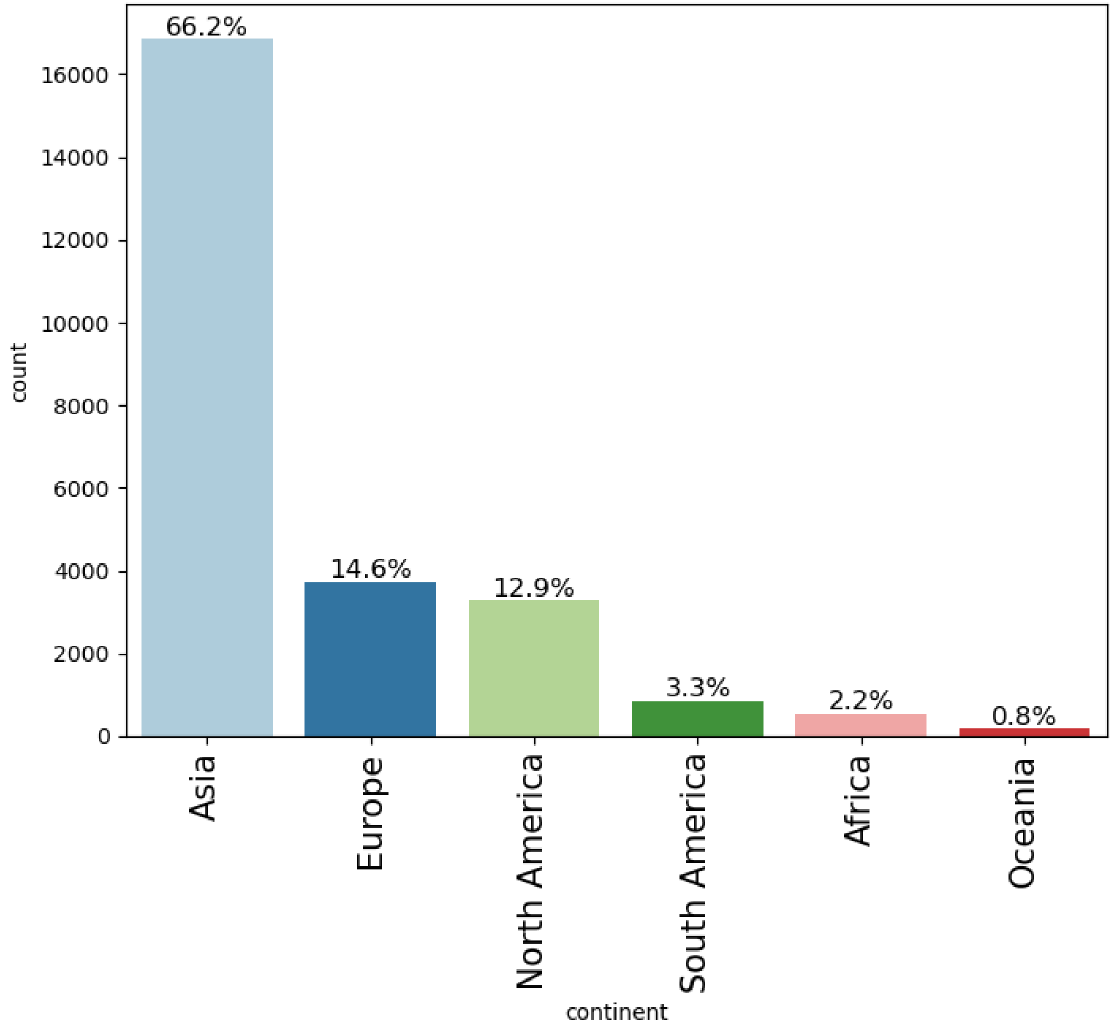
```
def stacked_barplot(data, predictor, target):
    """
    Print the category counts and plot a stacked bar chart

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

In [15]:
```
labeled_barplot(data, "requires_job_training", perc=True)
```



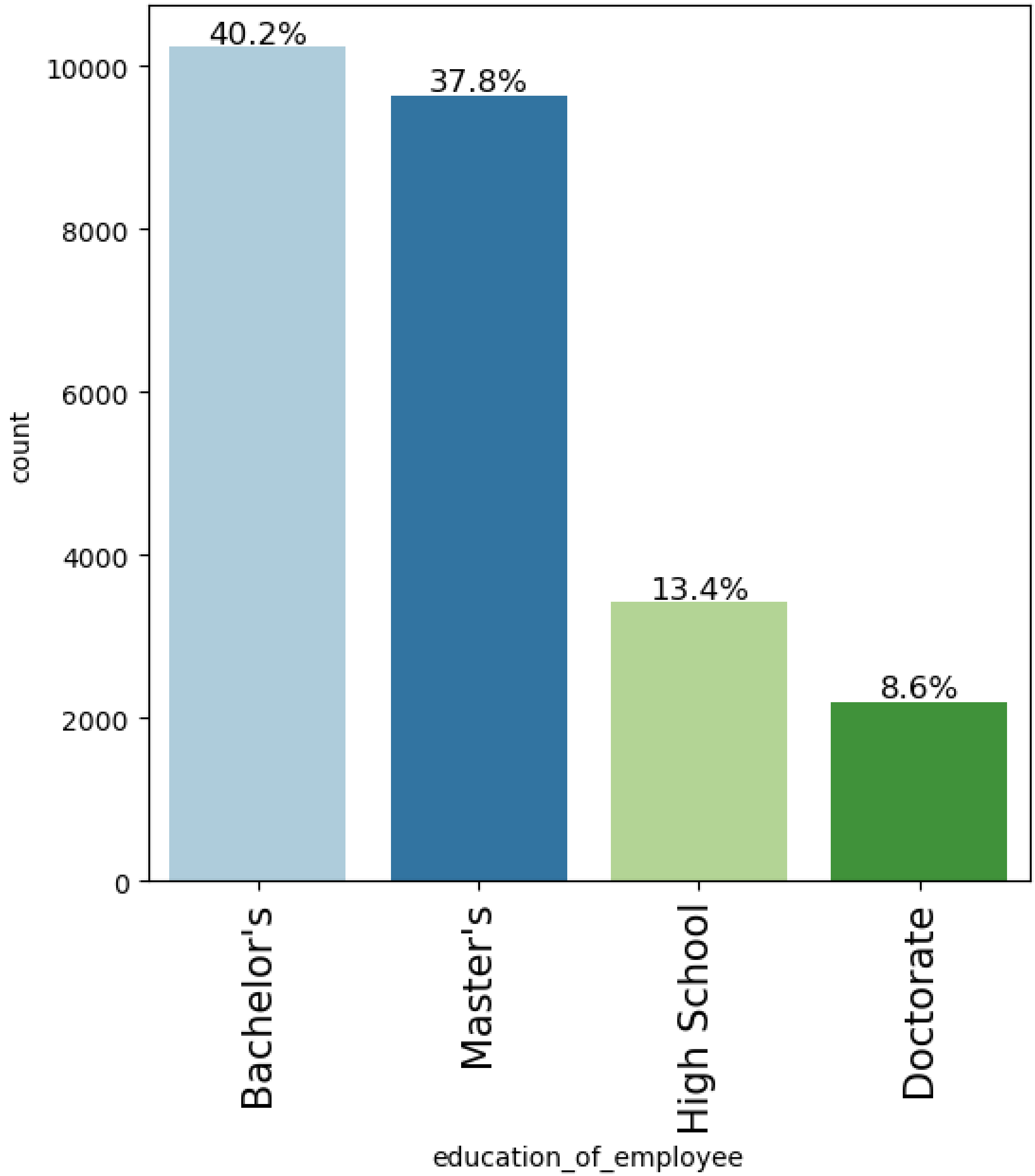
It seems as though most jobs don't require job training.

In [16]:
```
labeled_barplot(data, "continent", perc=True)
```



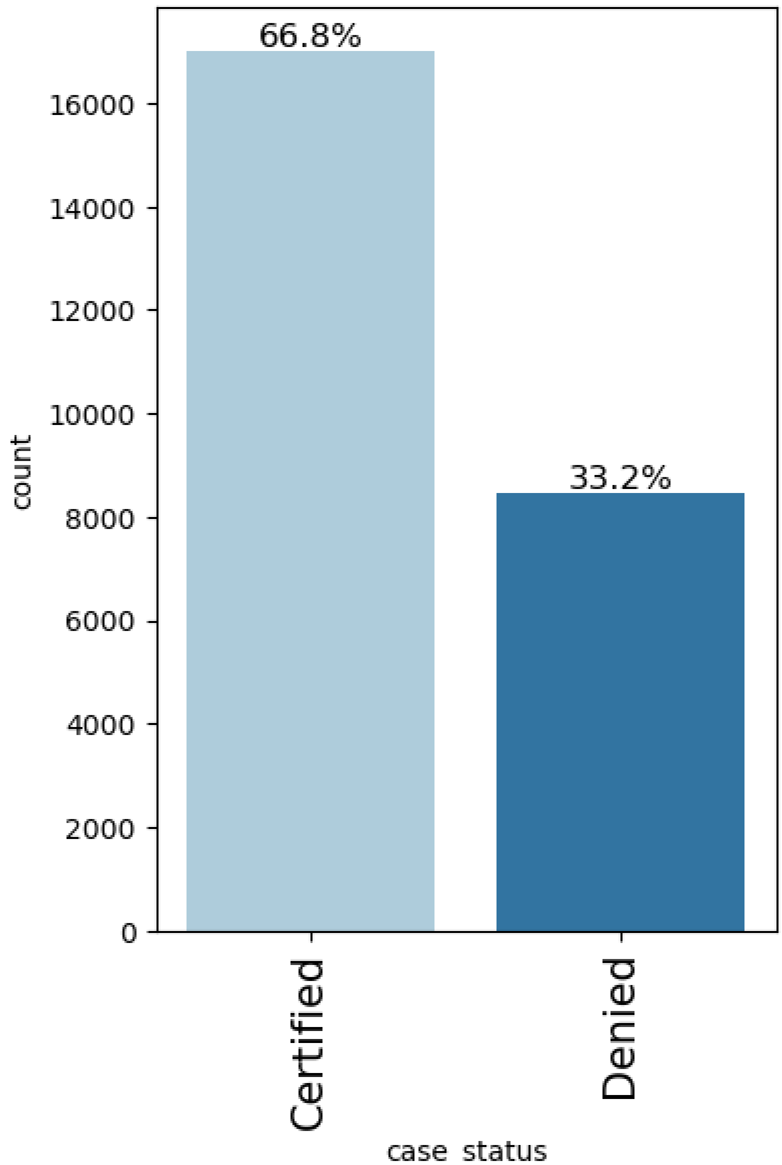
Most workers came from asia with asians making up 66.2% of visas submitted

In [17]:
```
labeled_barplot(data, "education_of_employee", perc=True)
```



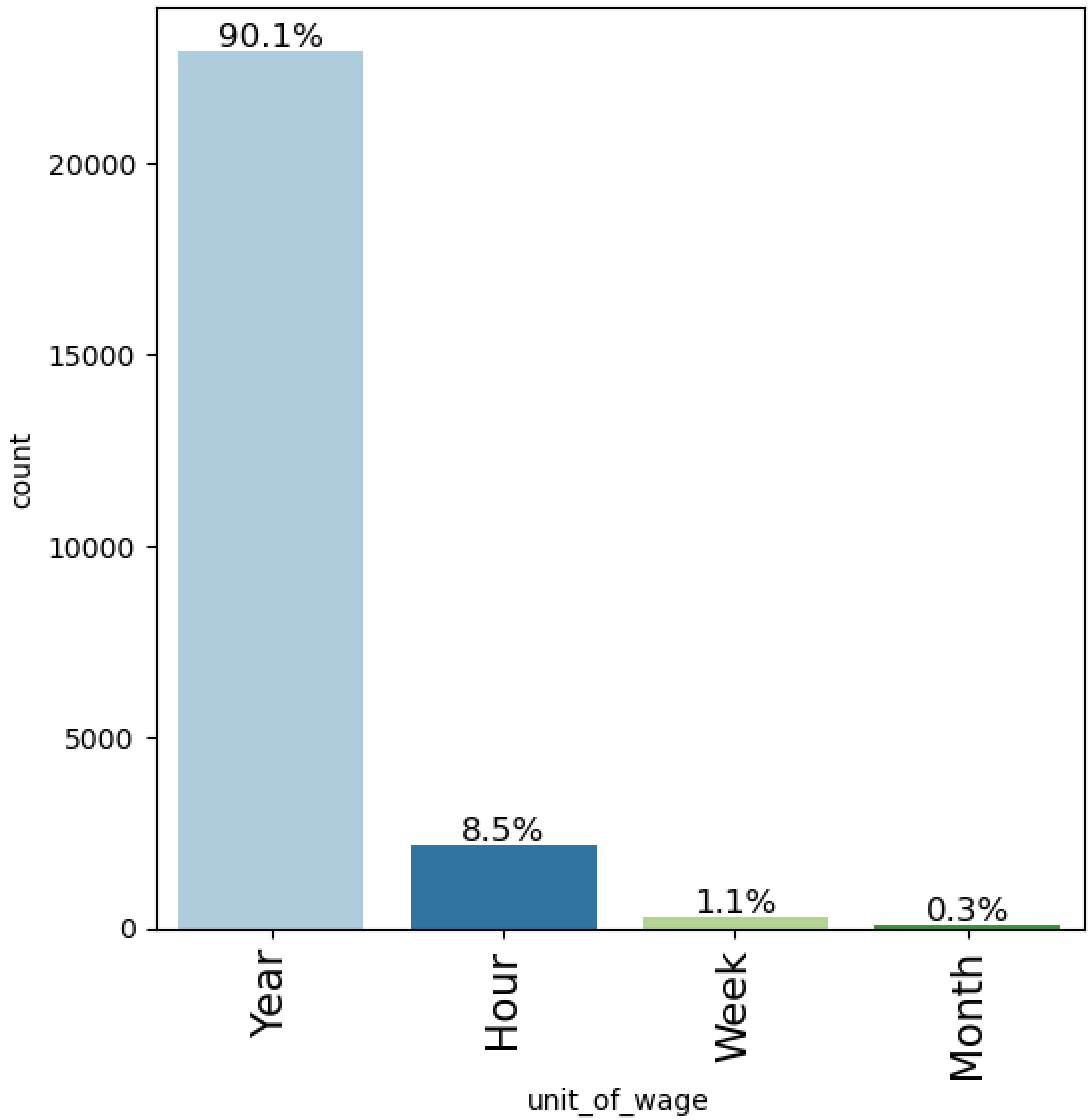
It appears only 13.4% of applicants weren't college educated.

In [18]:
```
labeled_barplot(data, "case_status", perc=True)
```



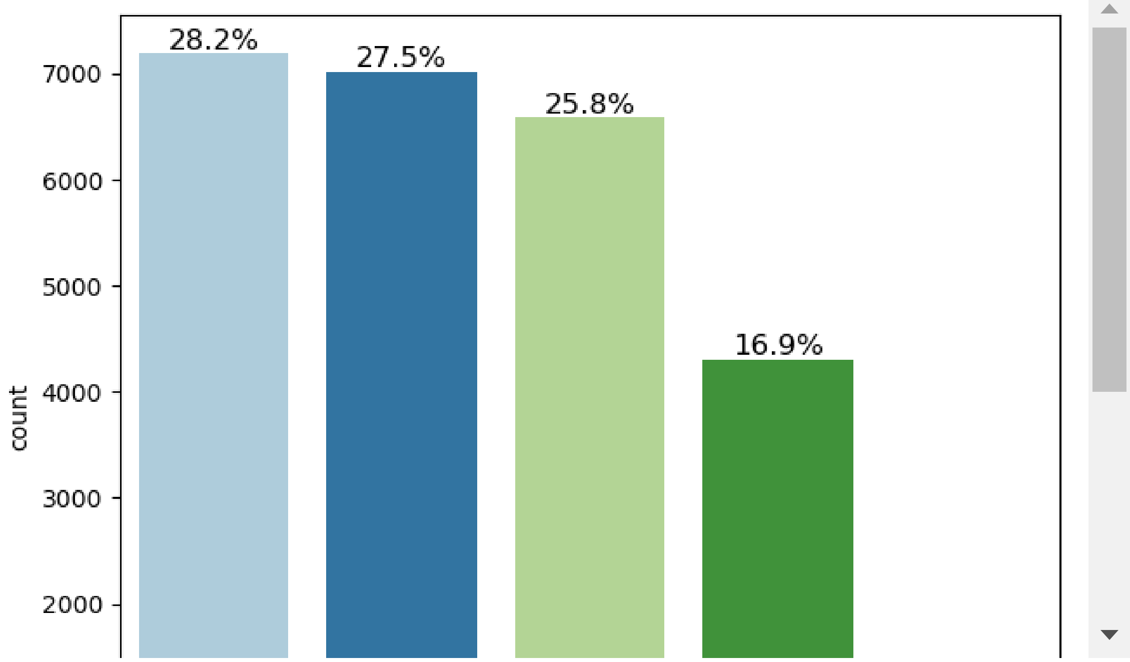
It seems around 2/3 of people were certified.

In [19]:
```
labeled_barplot(data, "unit_of_wage", perc=True)
```



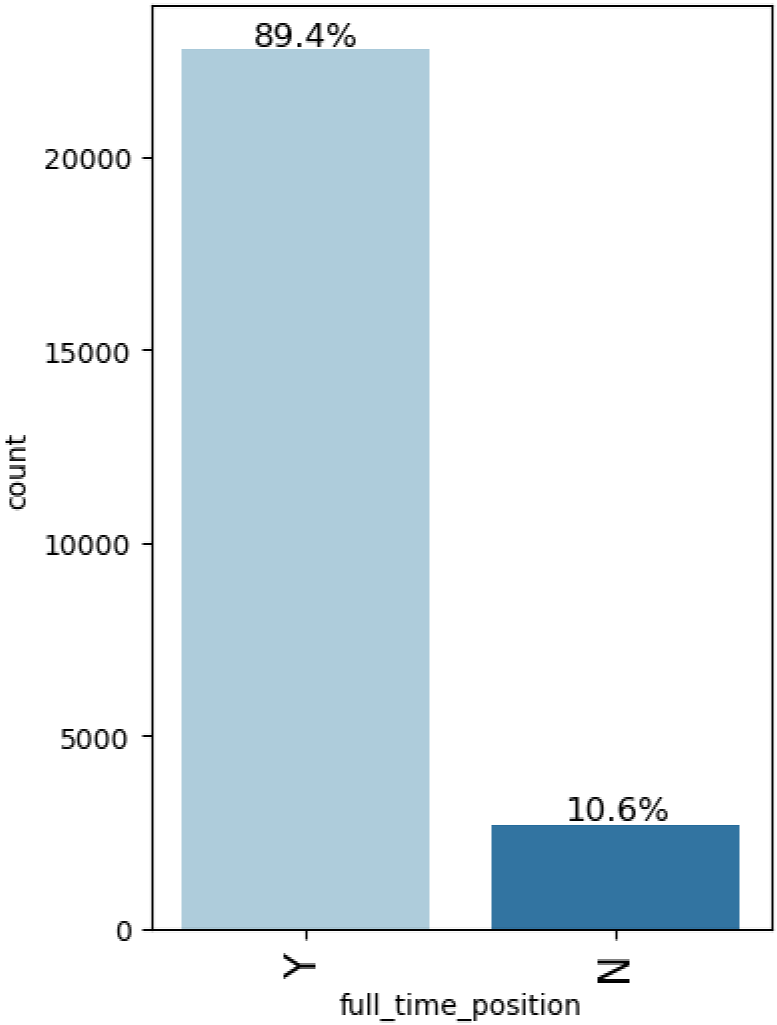
Salary was by far the most popular unit of wage as it makes up 90.1%

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



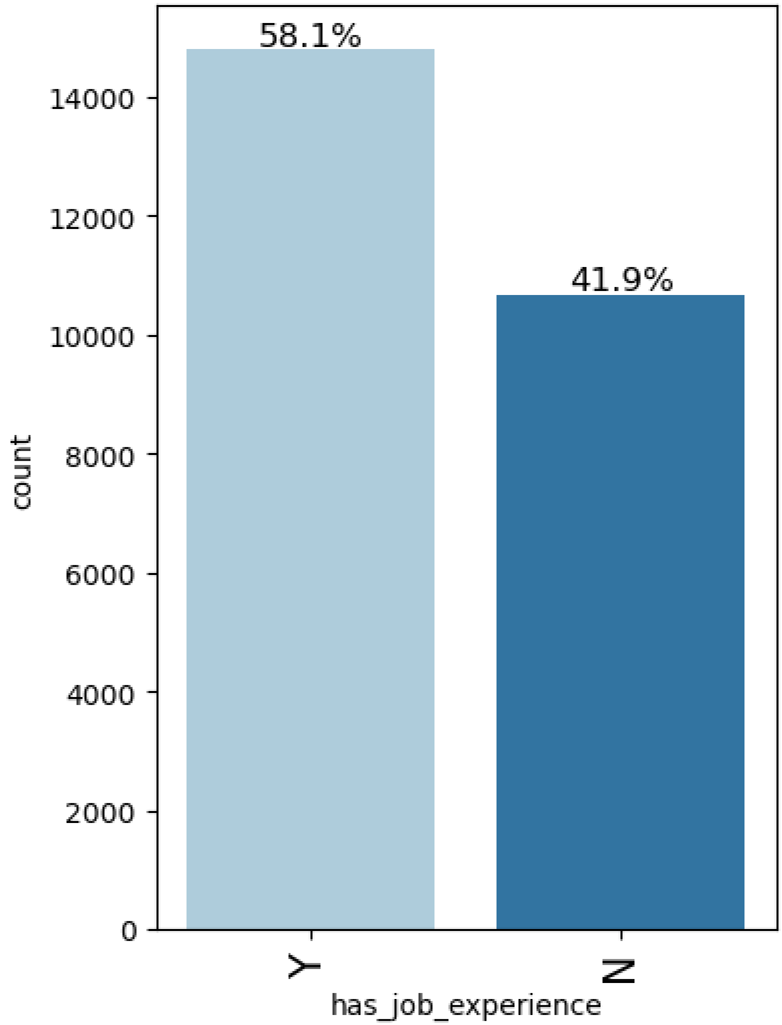
The Northeast and South seem to have the highest rates of visa employment with the West in a close third.

In [21]:
```
labeled_barplot(data, "full_time_position", perc=True)
```



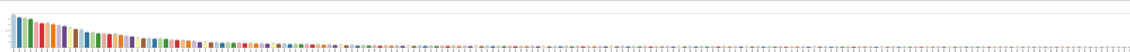
Most visa jobs are full time positions/

In [22]:
```
labeled_barplot(data, "has_job_experience", perc=True)
```

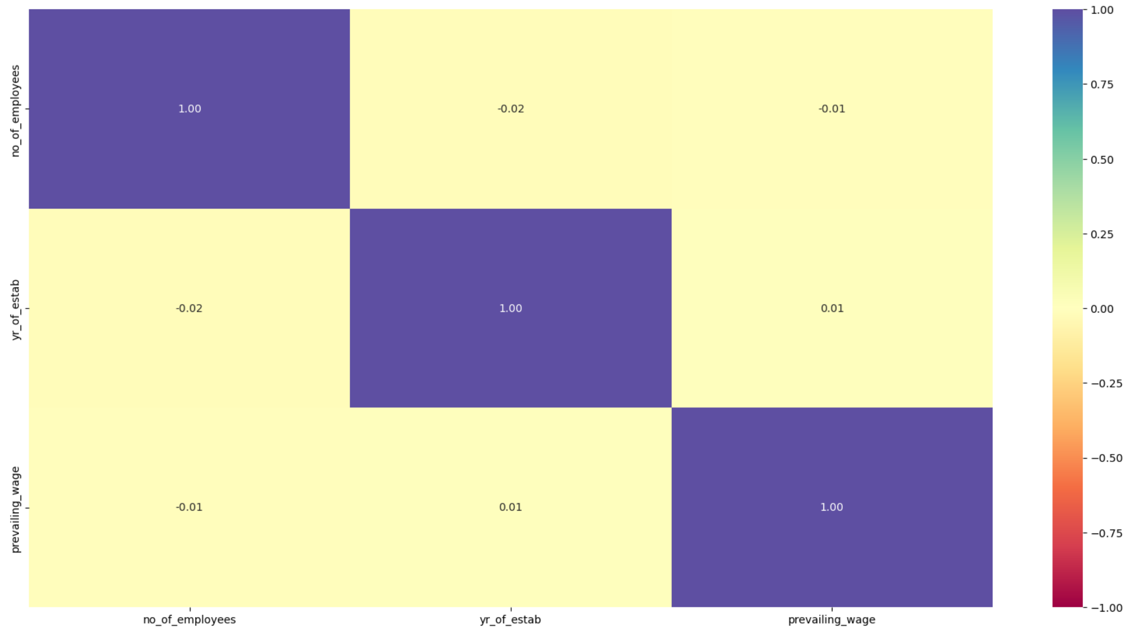


Just over half of visa applicants had job experience.

In [23]:
```
labeled_barplot(data, "yr_of_estab", perc=True)
```



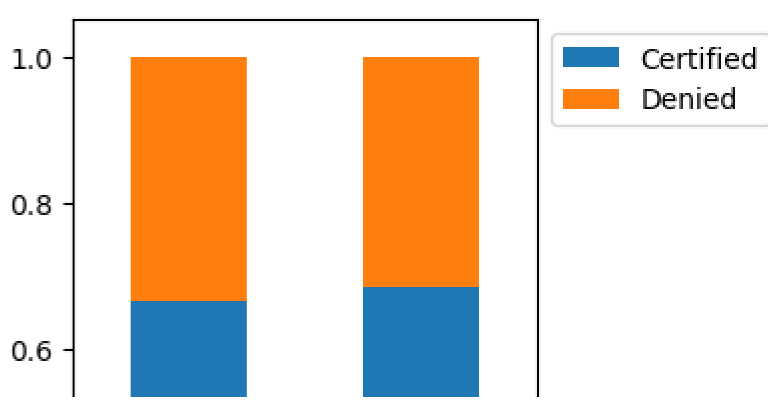
In [24]:
```
plt.figure(figsize=(20,10))
sns.heatmap(data.corr(),annot=True,vmin=-1,vmax=1,fmt='.2f',cmap="Spectral
plt.show()
```



We found no correlation

In [25]:
```
stacked_barplot(data, "full_time_position", "case_status")
```

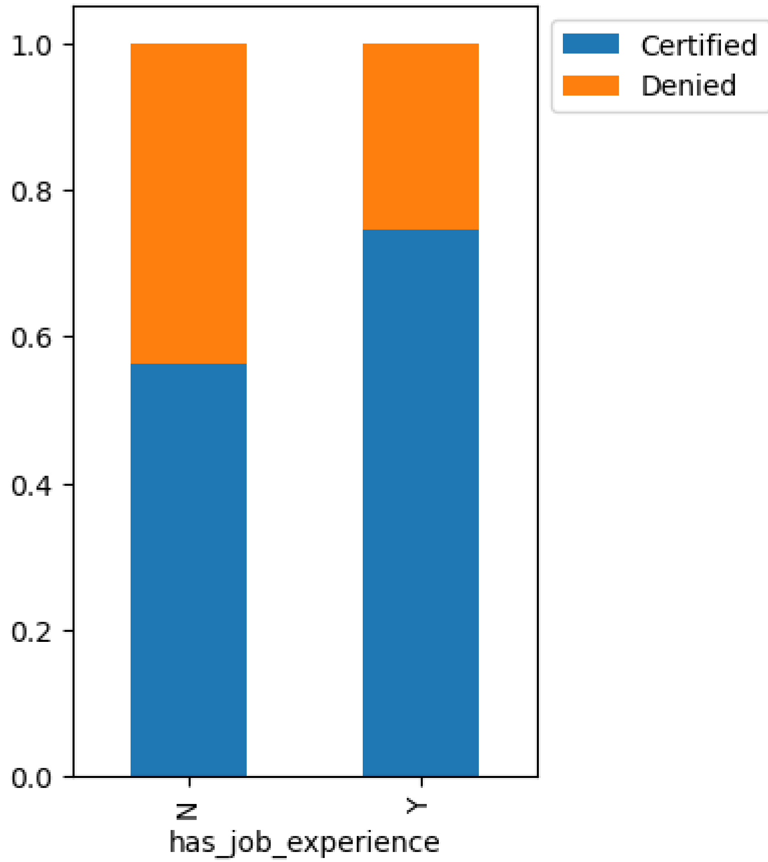
```
case_status          Certified  Denied    All
full_time_position
All                      17018    8462  25480
Y                        15163    7610  22773
N                         1855     852   2707
----------------------------------------------------------------------
--------------------------------------------------
```



Full time positions don't seem to really effect case status.

In [26]:
```
stacked_barplot(data, "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
------------------------------------------------------------------------
------------------------------------------------
```

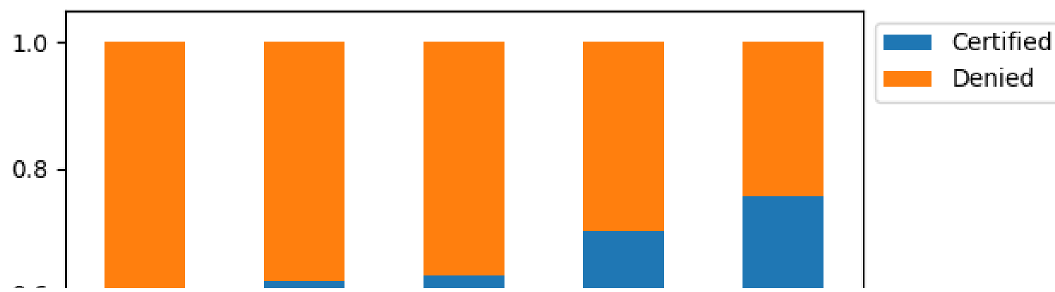


Question 3. Yes, if you have previous job expirence you stand a much better chance at getting a visa.

Job experience seems to positivly affect case status as having job experience increased the rate in which a visa was certified.

In [27]:
```
stacked_barplot(data, "region_of_employment", "case_status")
```

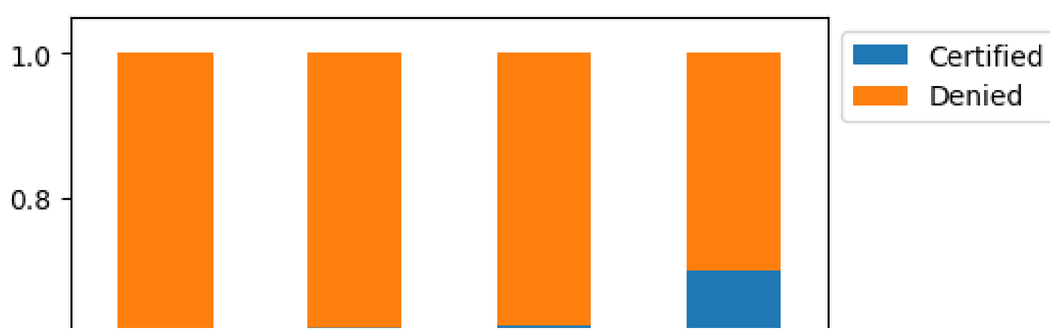
```
case_status           Certified  Denied    All
region_of_employment
All                       17018    8462  25480
Northeast                  4526    2669   7195
West                       4100    2486   6586
South                      4913    2104   7017
Midwest                    3253    1054 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



The Midwest has a slightly lower rate of denial thus meaning it will have minor importance in predicting case status.

In [28]:
```
stacked_barplot(data, "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
----------------------------------------------------------------------
--------------------------------------------------
```

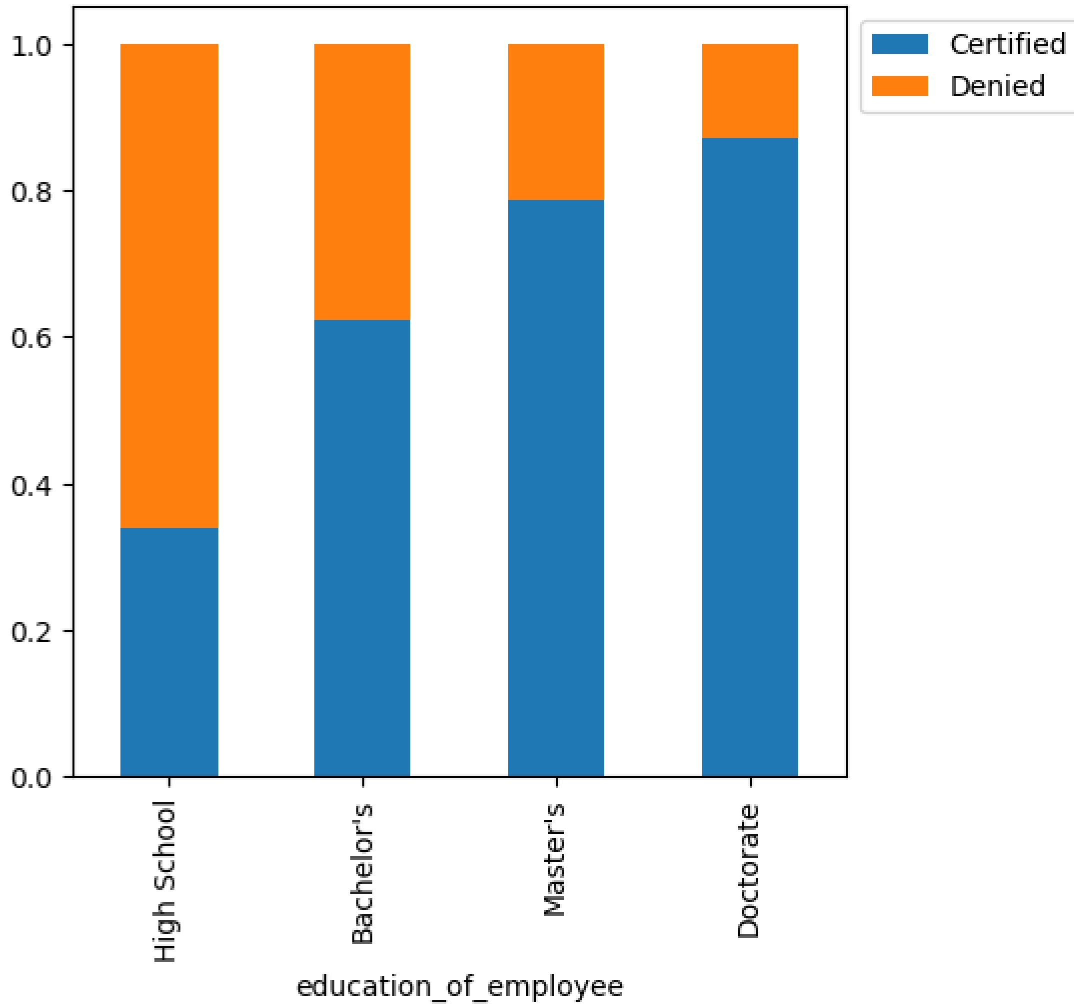


Question 4. Year or salary is most likely to be certified for a visa.

Salary has a much lower rate of denial thus meaning unit of wage will have importance in predicting case status.

In [29]:
```
stacked_barplot(data, "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
------------------------------------------------------------------------
------------------------------------------------
```

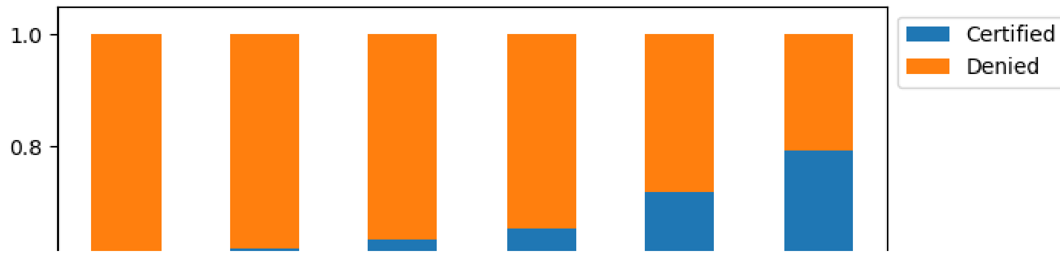


Question 1. Yes, education plays a large role in increasing the rate of certification.

Rate of denial drops greatly as the level of education increases meaning education of employee will be a valuable variable when trying to predict visa status.

In [30]:
```
stacked_barplot(data, "continent", "case_status")
```

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

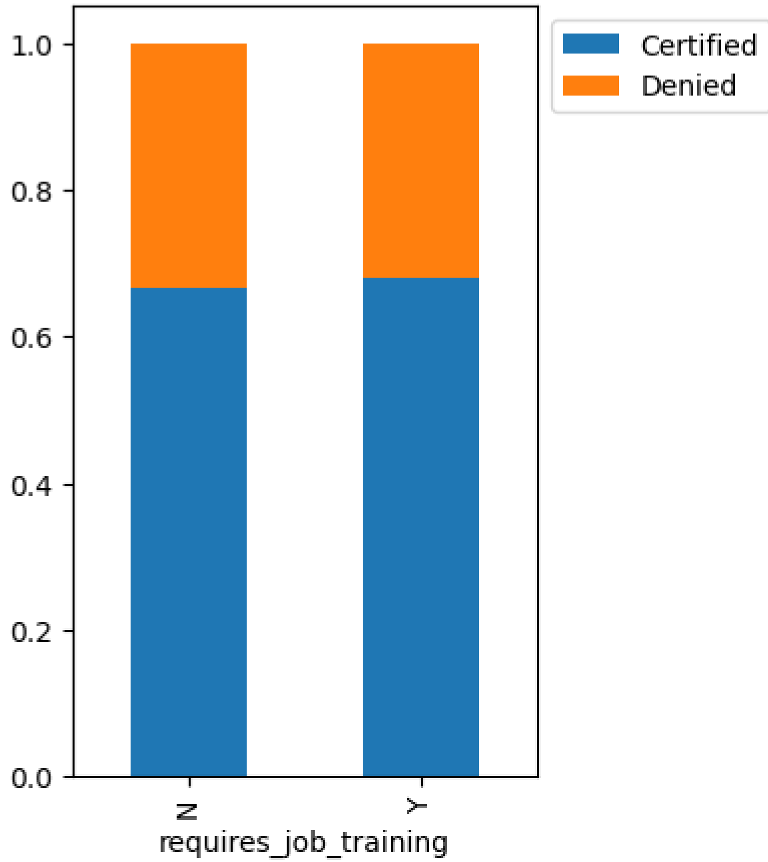


Question 2. visa status is a bit higher in Europe but fairly consistant throughout.

Europe has a slightly lower rate of denial thus meaning it will have minor importance in predicting case status.

In [31]:
```
stacked_barplot(data, "requires_job_training", "case_status")
```

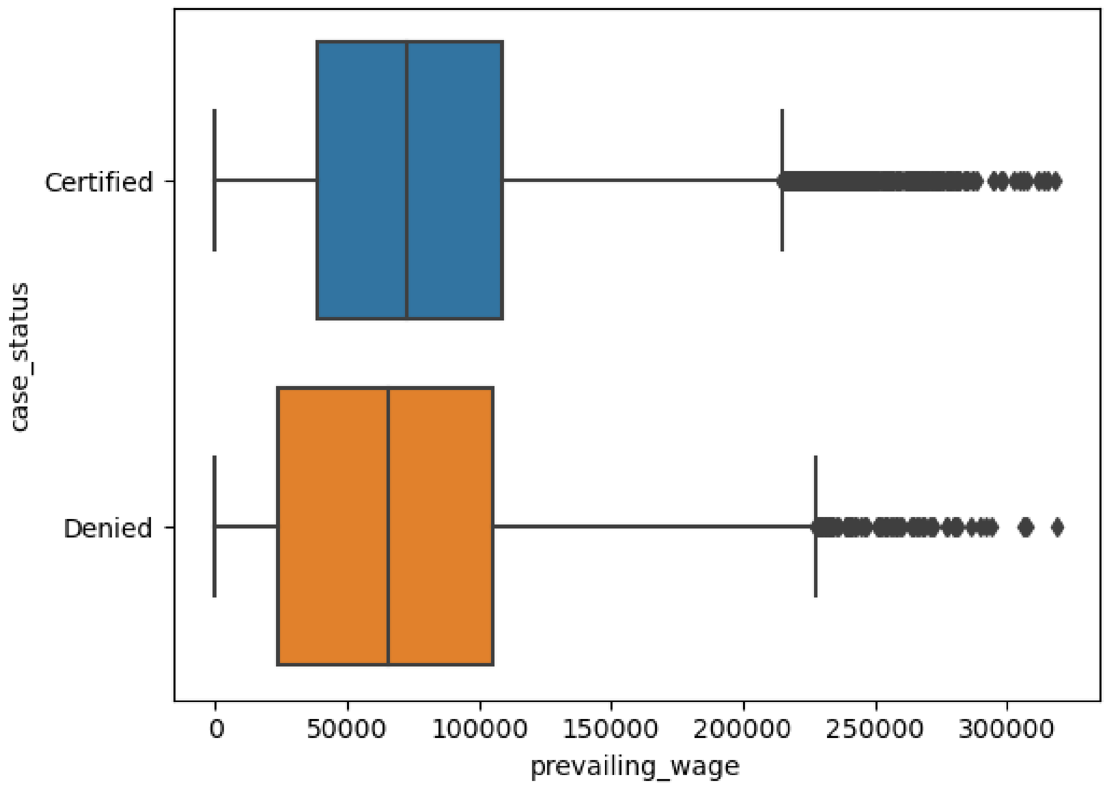
```
case_status            Certified  Denied    All
requires_job_training
All                        17018    8462  25480
N                          15012    7513  22525
Y                           2006     949   2955
------------------------------------------------------------------------
------------------------------------------------
```



There is no change in the rate of denial when looking at requires_job_training which means it will have less importance in predicting case status.

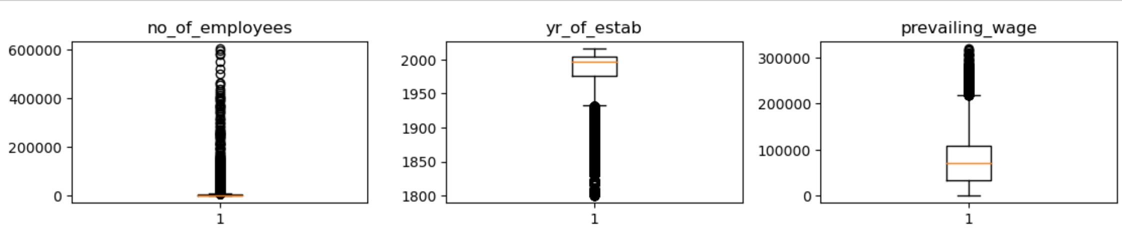
In [32]:
```
sns.boxplot(data, x='prevailing_wage', y="case_status")
```

Out[32]:
```
<Axes: xlabel='prevailing_wage', ylabel='case_status'>
```



Question 5. The prevailing wage doesn't seem to have much of an impact on case status.

In [33]:
```
numeric_columns = data.select_dtypes(include=np.number).columns.tolist()


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

for i, variable in enumerate(numeric_columns):
    plt.subplot(6, 4, i + 1)
    plt.boxplot(data[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)

plt.show()
```



We can see we have quite a few outliers but they are valid data point and will help us when predicting case status.

In [34]:
```
data_half = data.sample(frac=0.40)
```

In [35]:
```
data_half.shape
```

Out[35]:
```
(10192, 12)
```

In [36]:
```
data_half.drop('case_id', axis=1, inplace=True)
cols = ["case_status"]
data_half[cols] = data_half[cols].replace("Certified", 1)
data_half[cols] = data_half[cols].replace("Denied",0)
```

In [37]:
```
X = data_half.drop(["case_status"], axis=1)
y = data_half["case_status"]
X = add_constant(X)
X = pd.get_dummies(X, drop_first=True)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.30, random_state=1, stratify=y)
```

In [38]:
```
y.value_counts(1)
```

Out[38]:
```
1    0.669937
0    0.330063
Name: case_status, dtype: float64
```

In [39]:
```
y_test.value_counts(1)
```

Out[39]:
```
1    0.670046
0    0.329954
Name: case_status, dtype: float64
```

In [40]:
```
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.flatte
            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")
```

In [41]:
```
def get_metrics_score(model,flag=True):
    '''
    model : classifier to predict values of X

    '''
    score_list=[]

    pred_train = model.predict(X_train,y_train)
    pred_test = model.predict(X_test,y_test)

    train_acc = model.score(X_train,y_train)
    test_acc = model.score(X_test,y_test)

    train_recall = metrics.recall_score(y_train,pred_train, pos_label=1)
    test_recall = metrics.recall_score(y_test,pred_test)

    train_precision = metrics.precision_score(y_train,pred_train)
    test_precision = metrics.precision_score(y_test,pred_test)

    score_list.extend((train_acc,test_acc,train_recall,test_recall,train_p

    if flag == True:
        print("Accuracy on training set : ",model.score(X_train,y_train))
        print("Accuracy on test set : ",model.score(X_test,y_test))
        print("Recall on training set : ",metrics.recall_score(y_train,pre
        print("Recall on test set : ",metrics.recall_score(y_test,pred_tes
        print("Precision on training set : ",metrics.precision_score(y_tra
        print("Precision on test set : ",metrics.precision_score(y_test,pr

    return score_list
```

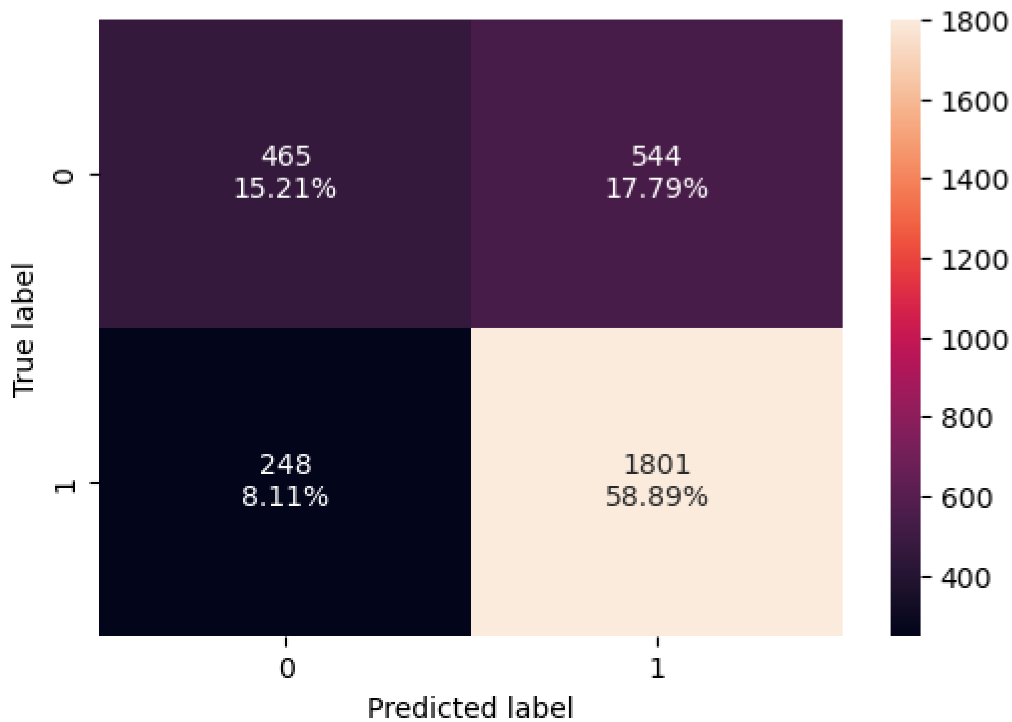
In [42]:

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

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

    pred = model.predict(predictors)

    acc = accuracy_score(target, pred)
    recall = recall_score(target, pred,pos_label=1)
    precision = precision_score(target, pred,pos_label=1)
    f1 = f1_score(target, pred,pos_label=1)

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

    return df_perf
```

In [43]:

```
ab_classifier = AdaBoostClassifier(random_state=1)
ab_classifier.fit(X_train,y_train)

ab_classifier_model_train_perf=model_performance_classification_sklearn(ab_
print(ab_classifier_model_train_perf)
ab_classifier_model_test_perf=model_performance_classification_sklearn(ab_
print(ab_classifier_model_test_perf)

confusion_matrix_sklearn(ab_classifier,X_test,y_test)
```

```
   Accuracy    Recall  Precision        F1
0  0.737454  0.881147   0.763416  0.818067
   Accuracy    Recall  Precision        F1
0  0.741007  0.878965   0.768017  0.819754
```



In [44]:

```
gbc = GradientBoostingClassifier(random_state=1)
gbc.fit(X_train,y_train)
```

Out[44]: GradientBoostingClassifier(random_state=1)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

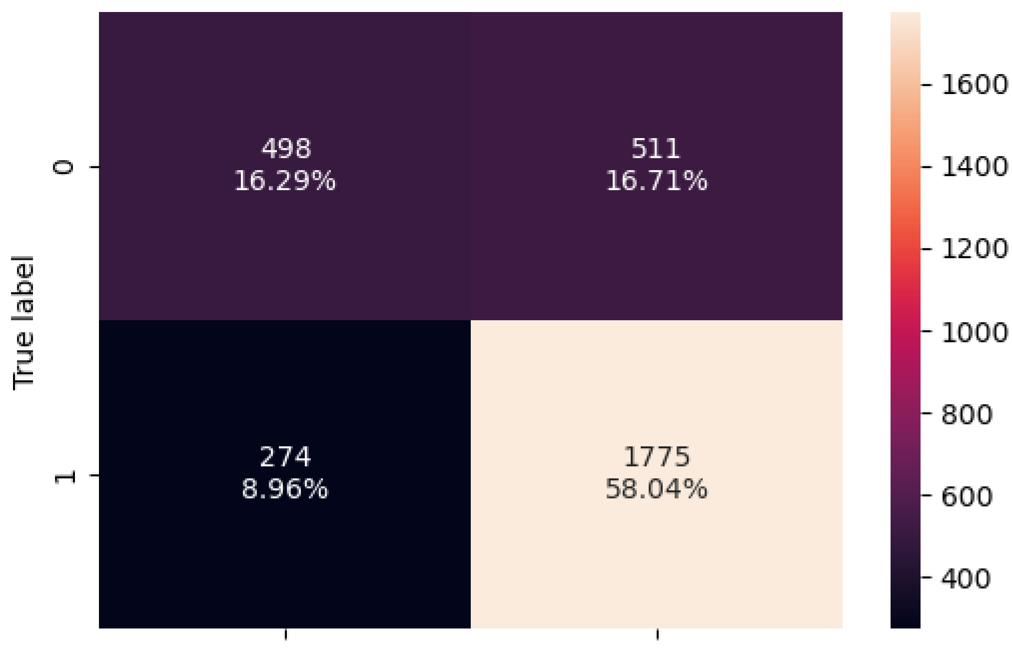
In [45]:

```
gb_classifier_model_train_perf=model_performance_classification_sklearn(gb
print("Training performance:\n",gb_classifier_model_train_perf)
gb_classifier_model_test_perf=model_performance_classification_sklearn(gbc
print("Testing performance:\n",gb_classifier_model_test_perf)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.762265  0.884076   0.787218  0.832841
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.743296  0.866276   0.776465  0.818916
```

In [46]:

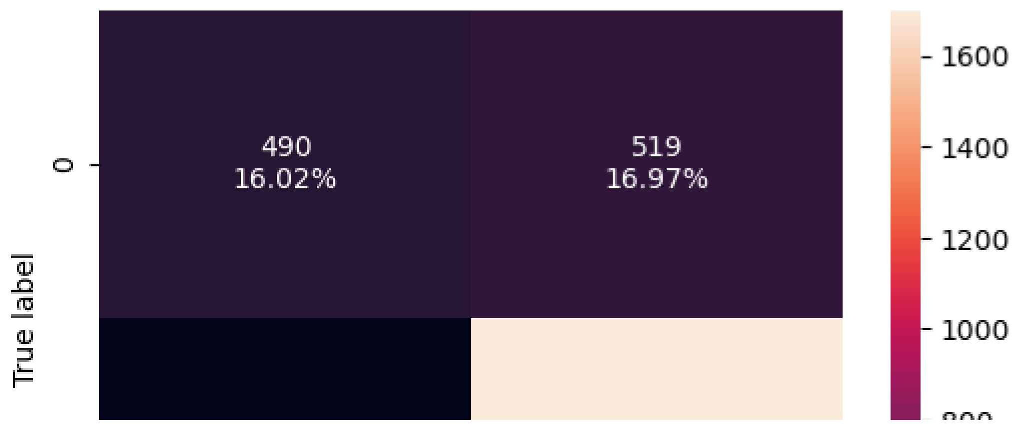
```
confusion_matrix_sklearn(gbc,X_test,y_test)
```



We can see that the gradient boosting training and testing models was fairly accurate and robust but it might still be improved through tuning the model.

In [47]:
```
xgb_classifier = XGBClassifier(random_state=1, eval_metric='logloss')
xgb_classifier.fit(X_train,y_train)

xgb_classifier_model_train_perf=model_performance_classification_sklearn(x
print("Training performance:\n",xgb_classifier_model_train_perf)
xgb_classifier_model_test_perf=model_performance_classification_sklearn(xg
print("Testing performance:\n",xgb_classifier_model_test_perf)

confusion_matrix_sklearn(xgb_classifier,X_test,y_test)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.908747  0.966311   0.904072  0.934156
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.716481  0.830161   0.766216  0.796908
```

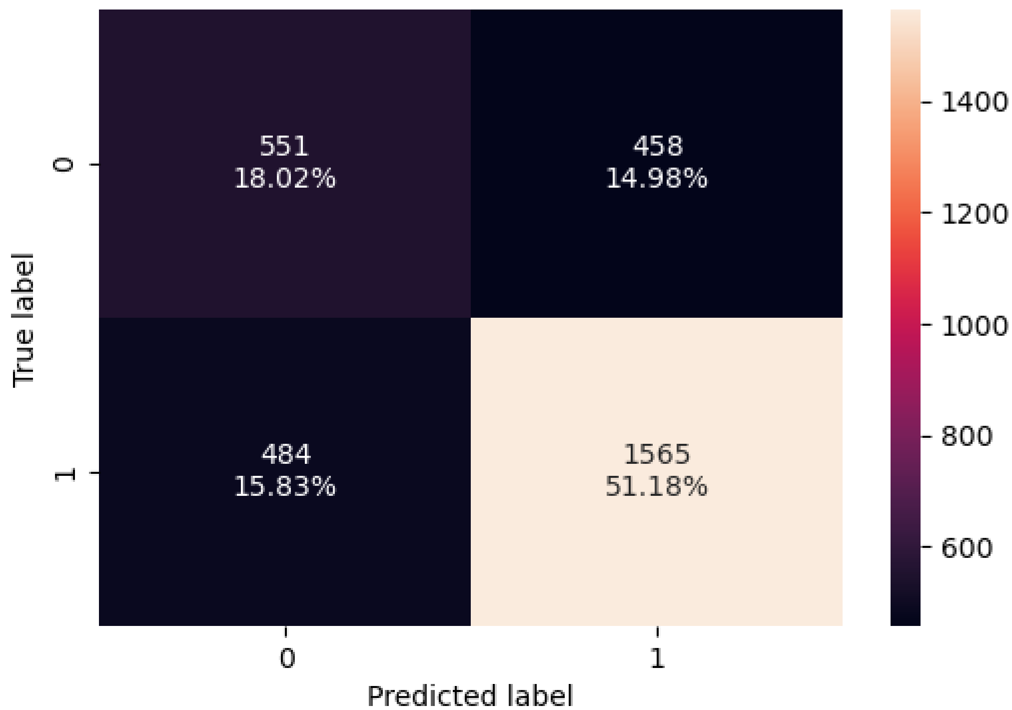


We can see that the XGB test was fairly accurate and robust but it might still be improved through tuning the model. We can see that the XGB train suffered from overfitting but it might be improved through tuning the model.

In [48]:

```
bagging_classifier = BaggingClassifier(random_state=1)
bagging_classifier.fit(X_train,y_train)

bagging_classifier_model_train_perf=model_performance_classification_sklea
print(bagging_classifier_model_train_perf)
bagging_classifier_model_test_perf=model_performance_classification_sklear
print(bagging_classifier_model_test_perf)

confusion_matrix_sklearn(bagging_classifier,X_test,y_test)
```

```
   Accuracy    Recall  Precision        F1
0  0.983459  0.984934   0.990322  0.987621
   Accuracy    Recall  Precision        F1
0  0.691956  0.763787   0.773604  0.768664
```



In [49]:

```
bagging_estimator_tuned = BaggingClassifier(random_state=1)

parameters = {'max_samples': [0.7,0.8,0.9,1],
              'max_features': [0.7,0.8,0.9,1],
              'n_estimators' : [10,20,30,40,50],
             }

scorer = metrics.make_scorer(metrics.f1_score)

grid_obj = RandomizedSearchCV(bagging_estimator_tuned, parameters, scoring
grid_obj = grid_obj.fit(X_train, y_train)

bagging_estimator_tuned = grid_obj.best_estimator_

bagging_estimator_tuned.fit(X_train, y_train)
```

Out[49]:

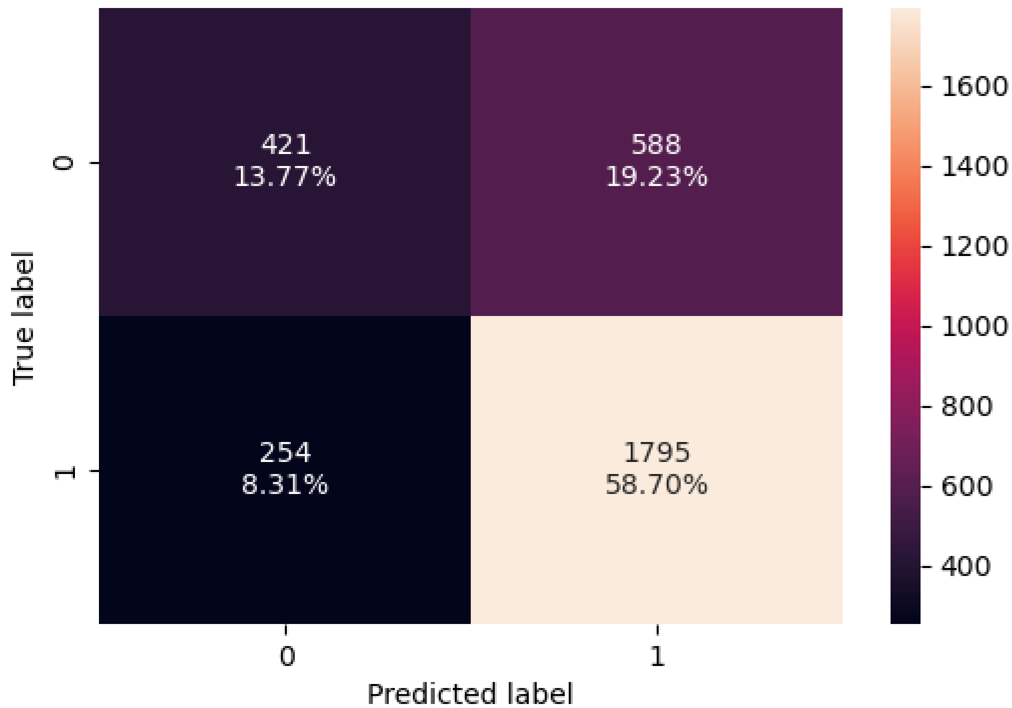
```
BaggingClassifier(max_features=0.7, max_samples=0.7, n_estimators=40,
                  random_state=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [50]:

```
bagging_estimator_tuned_model_train_perf=model_performance_classification_
print(bagging_estimator_tuned_model_train_perf)
bagging_estimator_tuned_model_test_perf=model_performance_classification_s
print(bagging_estimator_tuned_model_test_perf)

confusion_matrix_sklearn(bagging_estimator_tuned,X_test,y_test)
```

```
   Accuracy    Recall  Precision        F1
0  0.985001  0.997908   0.980066  0.988906
   Accuracy    Recall  Precision        F1
0  0.724657  0.876037   0.753252  0.810018
```



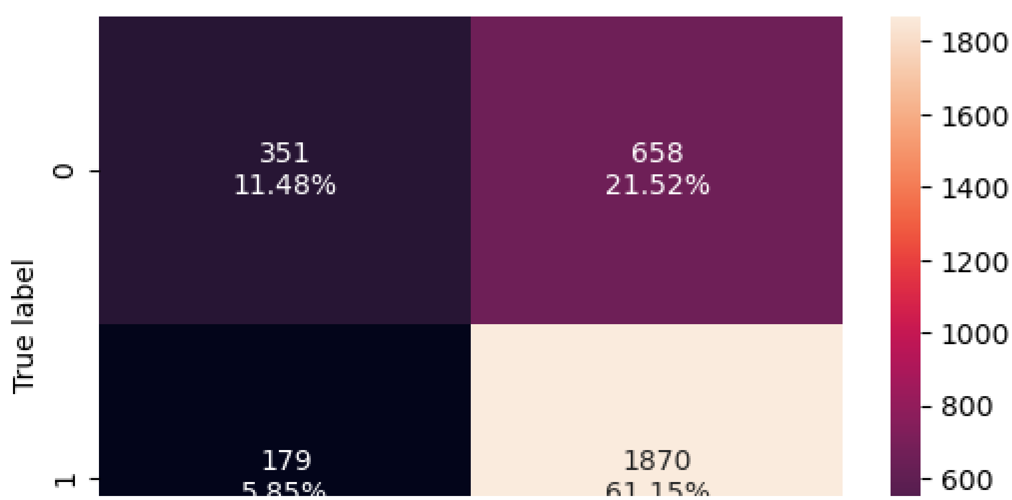
In [51]:
```
abc_tuned = AdaBoostClassifier(random_state=1)

parameters = {
    "base_estimator":[DecisionTreeClassifier(max_depth=1),DecisionTreeClas
                      DecisionTreeClassifier(max_depth=3)],
    "n_estimators": np.arange(10,110,10),
    "learning_rate":np.arange(0.1,2,0.1)
}

scorer = metrics.make_scorer(metrics.f1_score)

grid_obj = RandomizedSearchCV(abc_tuned, parameters, scoring=scorer,cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

abc_tuned = grid_obj.best_estimator_

abc_tuned.fit(X_train, y_train)
```

Out[51]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=2),
                   learning_rate=0.1, n_estimators=20, random_state=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [52]:
```
abc_tuned_model_train_perf=model_performance_classification_sklearn(abc_tu
print(abc_tuned_model_train_perf)
abc_tuned_model_test_perf=model_performance_classification_sklearn(abc_tun
print(abc_tuned_model_test_perf)

confusion_matrix_sklearn(abc_tuned,X_test,y_test)
```

```
   Accuracy    Recall  Precision        F1
0  0.730726  0.914417   0.742945  0.819811
   Accuracy   Recall  Precision        F1
0  0.726292  0.91264   0.739715  0.817129
```



The test abc model proformed well and is fairly robust We saw slight improvements with tuning

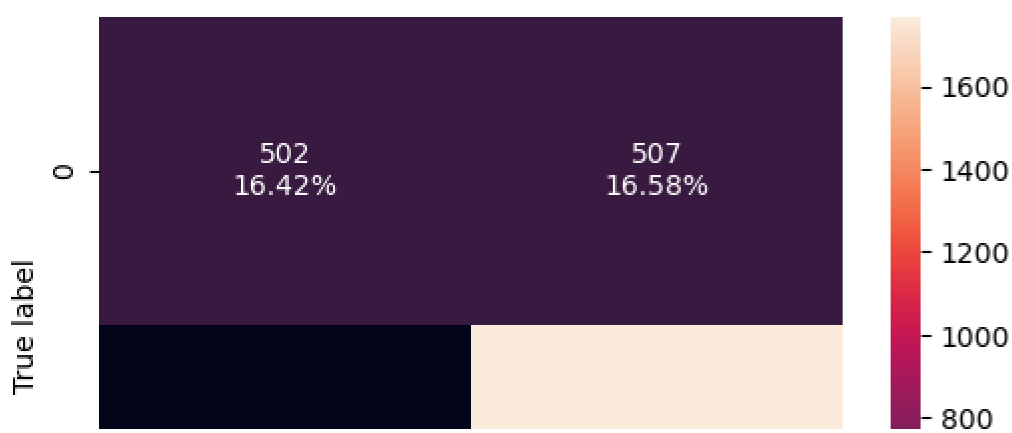
In [54]:
```
gbc_tuned = GradientBoostingClassifier(init=AdaBoostClassifier(random_stat

parameters = {
    "n_estimators": [100,150,200,250],
    "subsample":[0.8,0.9,1],
    "max_features":[0.7,0.8,0.9,1]
}

scorer = metrics.make_scorer(metrics.f1_score)

grid_obj = RandomizedSearchCV(gbc_tuned, parameters, scoring=scorer,cv=5)
grid_obj = grid_obj.fit(X_train, y_train)

gbc_tuned = grid_obj.best_estimator_

gbc_tuned.fit(X_train, y_train)
```

Out[54]:
```
GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                           max_features=0.8, random_state=1, subsample=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [55]:
```
gbc_tuned_model_train_perf=model_performance_classification_sklearn(gbc_tu
print("Training performance:\n",gbc_tuned_model_train_perf)
gbc_tuned_model_test_perf=model_performance_classification_sklearn(gbc_tun
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  0.762826  0.882821   0.788451  0.832971
Testing performance:
    Accuracy   Recall  Precision        F1
0  0.742315  0.86286   0.777143  0.817761
```



In [56]:

```
xgb_tuned = XGBClassifier(random_state=1,eval_metric='logloss')

# Grid of parameters to choose from
## add from
parameters = {
    "n_estimators": np.arange(10,100,20),
    "scale_pos_weight":[0,1,2,5],
    "subsample":[0.5,0.7,0.9,1],
    "learning_rate":[0.01,0.1,0.2,0.05],
    "gamma":[0,1,3],
    "colsample_bytree":[0.5,0.7,0.9,1],
    "colsample_bylevel":[0.5,0.7,0.9,1]
}

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

# Run the grid search
grid_obj = RandomizedSearchCV(xgb_tuned, parameters,scoring=acc_scorer,cv=
grid_obj = grid_obj.fit(X_train, y_train)

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

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

Out[56]:

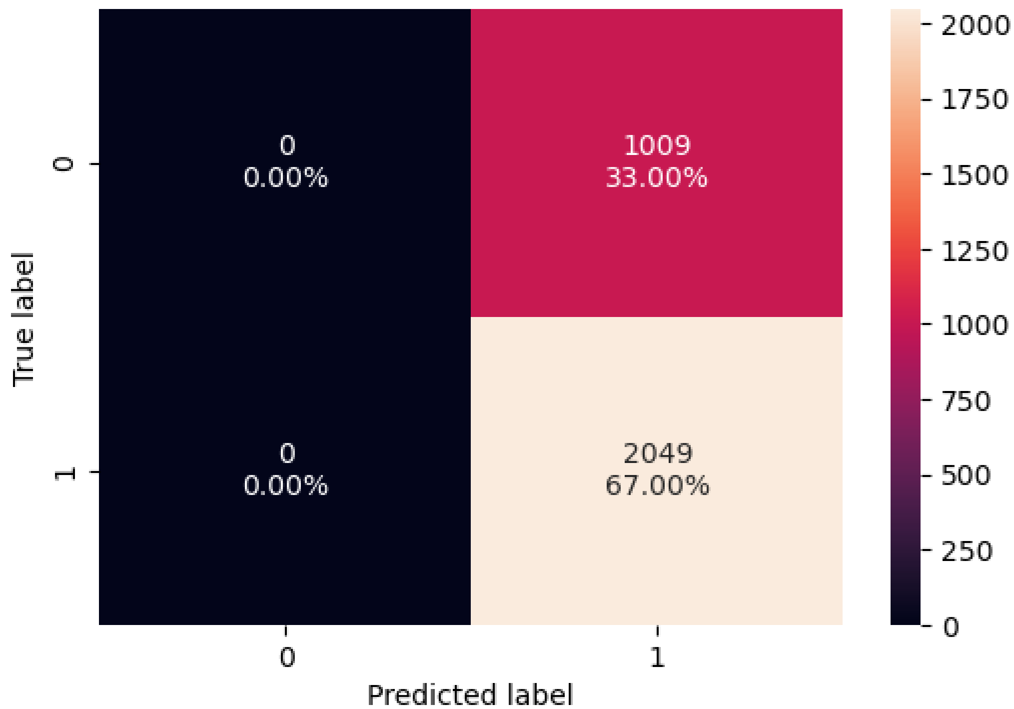
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=0.7, colsample_bynode=None,
              colsample_bytree=0.7, device=None, early_stopping_rounds=No
ne,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=1, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.01, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=Non
e,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimator
s=10,
              n_jobs=None, num_parallel_tree=None, random_state=1, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [58]:

```
xgb_tuned_model_train_perf=model_performance_classification_sklearn(xgb_tu
print("Training performance:\n",xgb_tuned_model_train_perf)
xgb_tuned_model_test_perf=model_performance_classification_sklearn(xgb_tun
print("Testing performance:\n",xgb_tuned_model_test_perf)

confusion_matrix_sklearn(xgb_tuned,X_test,y_test)
```

```
Training performance:
    Accuracy  Recall  Precision        F1
0  0.669891     1.0   0.669891  0.802317
Testing performance:
    Accuracy  Recall  Precision        F1
0  0.670046     1.0   0.670046  0.802428
```



In [63]:

```
models_test_comp_df = pd.concat(
    [
     bagging_classifier_model_test_perf.T,bagging_estimator_tuned_model_te
     abc_tuned_model_test_perf.T,gb_classifier_model_test_perf.T,gbc_tuned
    xgb_tuned_model_test_perf.T],
    axis=1,
)
models_test_comp_df.columns = [
    "Bagging Classifier",
    "Bagging Estimator Tuned",
    "Adaboost Classifier",
    "Adabosst Classifier Tuned",
    "Gradient Boost Classifier",
    "Gradient Boost Classifier Tuned",
    "XGBoost Classifier",
    "XGBoost Classifier Tuned"]
print("Testing performance comparison:")
models_test_comp_df
```

```
Testing performance comparison:
```

Out[63]:

	Bagging Classifier	Bagging Estimator Tuned	Adaboost Classifier	Adabosst Classifier Tuned	Gradient Boost Classifier	Gradient Boost Classifier Tuned	XGBoost Classifier	XGBc Class Tu
Accuracy	0.691956	0.724657	0.741007	0.726292	0.743296	0.742315	0.716481	0.670
Recall	0.763787	0.876037	0.878965	0.912640	0.866276	0.862860	0.830161	1.000
Precision	0.773604	0.753252	0.768017	0.739715	0.776465	0.777143	0.766216	0.670
F1	0.768664	0.810018	0.819754	0.817129	0.818916	0.817761	0.796908	0.802

It seems bagging tuned preformed the best overall I would recommend that the business look at prior job experience and education when evaluating visas.