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 [177…

```
# Library to suppress warnings or deprecation notes
import warnings
warnings.filterwarnings('ignore')

# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd

# Libraries to help with data visualization
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

# Library to split data
from sklearn.model_selection import train_test_split

# Libraries to import decision tree classifier and different ensemble classifiers
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
#To install xgboost library use - !pip install xgboost
from xgboost import XGBClassifier
from sklearn import tree
from sklearn.ensemble import StackingClassifier
from sklearn.tree import DecisionTreeClassifier

# Libtune to tune model, get different metric scores
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, p
from sklearn.model_selection import GridSearchCV
```

In [23]:
```
#Loading dataset
data=pd.read_csv("EasyVisa.csv")
```

Data Overview

- Observations
- Sanity checks

In [24]:
```
data.shape
```

Out[24]:
```
(25480, 12)
```

- There are 25480 rows and 12 columns in the dataset.

In [25]:
```
data.head()
```

Out[25]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employ
0	EZYV01	Asia	High School	N	N	145
1	EZYV02	Asia	Master's	Y	N	24
2	EZYV03	Asia	Bachelor's	N	Y	444
3	EZYV04	Asia	Bachelor's	N	N	
4	EZYV05	Africa	Master's	Y	N	10

In [26]:
```
data.tail()
```

Out[26]:

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

In [27]:
```
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
```

- There are two integers, one float and nine object data types.
- The object data types will be converted to category.
- case_id is a unique identifier for each row and can be dropped from the dataset

In [30]:
```
data.describe(include='all').T
```

Out[30]:

	count	unique	top	freq	mean	std	min	25
case_id	25480	25480	EZYV01	1	NaN	NaN	NaN	Na
continent	25480	6	Asia	16861	NaN	NaN	NaN	Na
education_of_employee	25480	4	Bachelor's	10234	NaN	NaN	NaN	Na
has_job_experience	25480	2	Y	14802	NaN	NaN	NaN	Na
requires_job_training	25480	2	N	22525	NaN	NaN	NaN	Na
no_of_employees	25480.0	NaN	NaN	NaN	5667.04321	22877.928848	-26.0	1022
yr_of_estab	25480.0	NaN	NaN	NaN	1979.409929	42.366929	1800.0	1976
region_of_employment	25480	5	Northeast	7195	NaN	NaN	NaN	Na
prevailing_wage	25480.0	NaN	NaN	NaN	74455.814592	52815.942327	2.1367	34015.
unit_of_wage	25480	4	Year	22962	NaN	NaN	NaN	Na
full_time_position	25480	2	Y	22773	NaN	NaN	NaN	Na
case_status	25480	2	Certified	17018	NaN	NaN	NaN	Na

- There are 25480 unique entries.
- There are six different continents with Asia being the most popular
- There are 4 distinct education levels, with most entries being for Bacherlor's

- The 50th percentile for employee strength is 2109 and 75th percentile is 3504. There are quite a few outliers wherein we have employers that have a significantly higher number of employees.
- There appear to be few outliers in the salary range as well. The median salary is 70308 whereas maz salary is over 300K.

In [29]:
```
data.isnull().sum()
```

Out[29]:
```
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
```

- There are no null values in the dataset. All columns have values.

In [21]:
```
data[data.duplicated()]
```

Out[21]:

case_id	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employe

There are no duplicate rows in the data.

In [31]:
```
# convert all columns with dtype object into category
for i in data.columns[data.dtypes=='object']:
    data[i] = data[i].astype('category')
```

In [32]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25480 entries, 0 to 25479
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype   
---  ------                 --------------  -----   
 0   case_id                25480 non-null  category
 1   continent              25480 non-null  category
 2   education_of_employee  25480 non-null  category
 3   has_job_experience     25480 non-null  category
 4   requires_job_training  25480 non-null  category
 5   no_of_employees        25480 non-null  int64   
 6   yr_of_estab            25480 non-null  int64   
 7   region_of_employment   25480 non-null  category
 8   prevailing_wage        25480 non-null  float64 
 9   unit_of_wage           25480 non-null  category
 10  full_time_position     25480 non-null  category
 11  case_status            25480 non-null  category
dtypes: category(9), float64(1), int64(2)
memory usage: 2.0 MB
```

- All the object data types have been converted to category data type. The memory usage has reduced to 2MB from 2.3 MB.

In [33]:
```python
data.describe(include='all').T
```

Out[33]:

	count	unique	top	freq	mean	std	min	25
case_id	25480	25480	EZYV01	1	NaN	NaN	NaN	Na
continent	25480	6	Asia	16861	NaN	NaN	NaN	Na
education_of_employee	25480	4	Bachelor's	10234	NaN	NaN	NaN	Na
has_job_experience	25480	2	Y	14802	NaN	NaN	NaN	Na
requires_job_training	25480	2	N	22525	NaN	NaN	NaN	Na
no_of_employees	25480.0	NaN	NaN	NaN	5667.04321	22877.928848	-26.0	1022
yr_of_estab	25480.0	NaN	NaN	NaN	1979.409929	42.366929	1800.0	1976
region_of_employment	25480	5	Northeast	7195	NaN	NaN	NaN	Na
prevailing_wage	25480.0	NaN	NaN	NaN	74455.814592	52815.942327	2.1367	34015.
unit_of_wage	25480	4	Year	22962	NaN	NaN	NaN	Na
full_time_position	25480	2	Y	22773	NaN	NaN	NaN	Na
case_status	25480	2	Certified	17018	NaN	NaN	NaN	Na

In [34]:
```python
# dropping case_id as it is just a unique identifier for each row
data.drop(['case_id'],axis=1,inplace=True)
```

In [38]:
```python
data[data.duplicated()].count()
```

Out[38]:
```
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
```

- There are no duplicates in the data set even after dropping case_id.

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.

Univariate analysis

In [39]:
```
# function to plot a boxplot and a histogram along the same scale.


def histogram_boxplot(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, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a triangle will indicate the mean value of the co
    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
```

```python
    )  # 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 [40]:

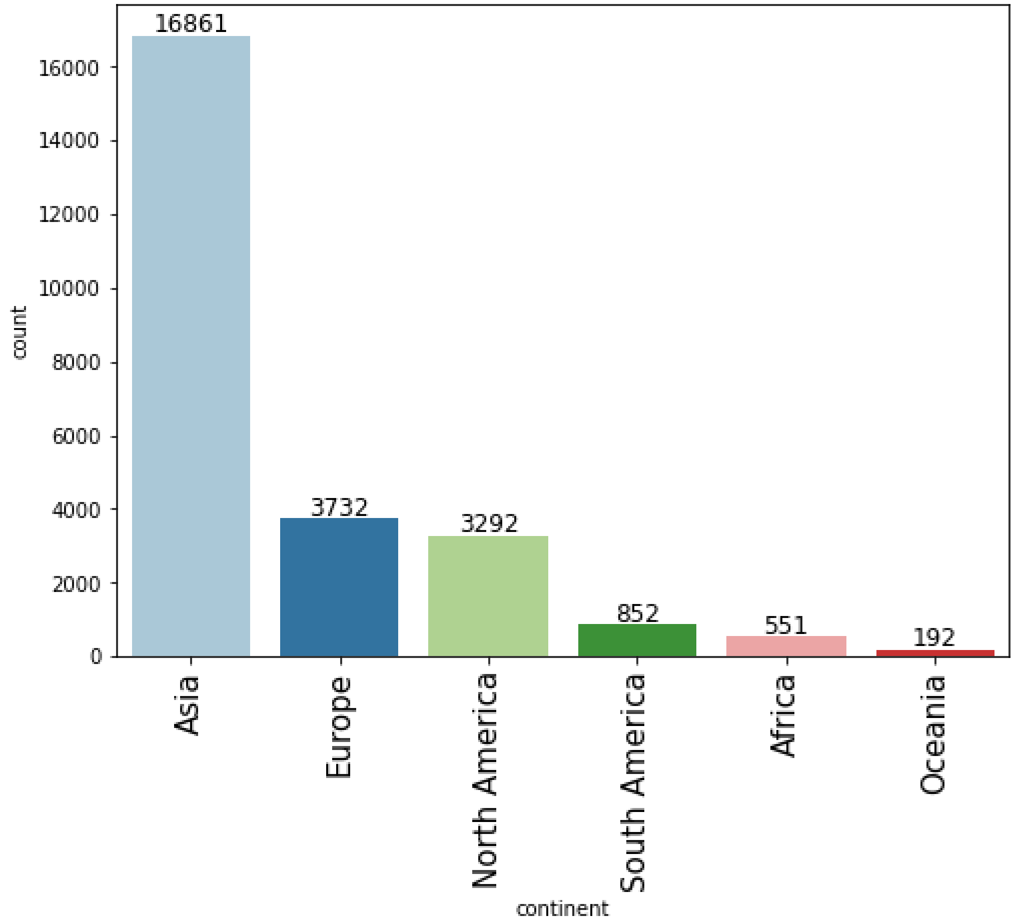
```python
# function to create labeled barplots


def labeled_barplot(data, feature, perc=False, n=None):
    """
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 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 [41]:

```python
# function to plot stacked bar chart

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

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_values(
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_values(
        by=sorter, ascending=False
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 1, 5), cmap='viridis')
    plt.legend(
        loc="lower left",
        frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

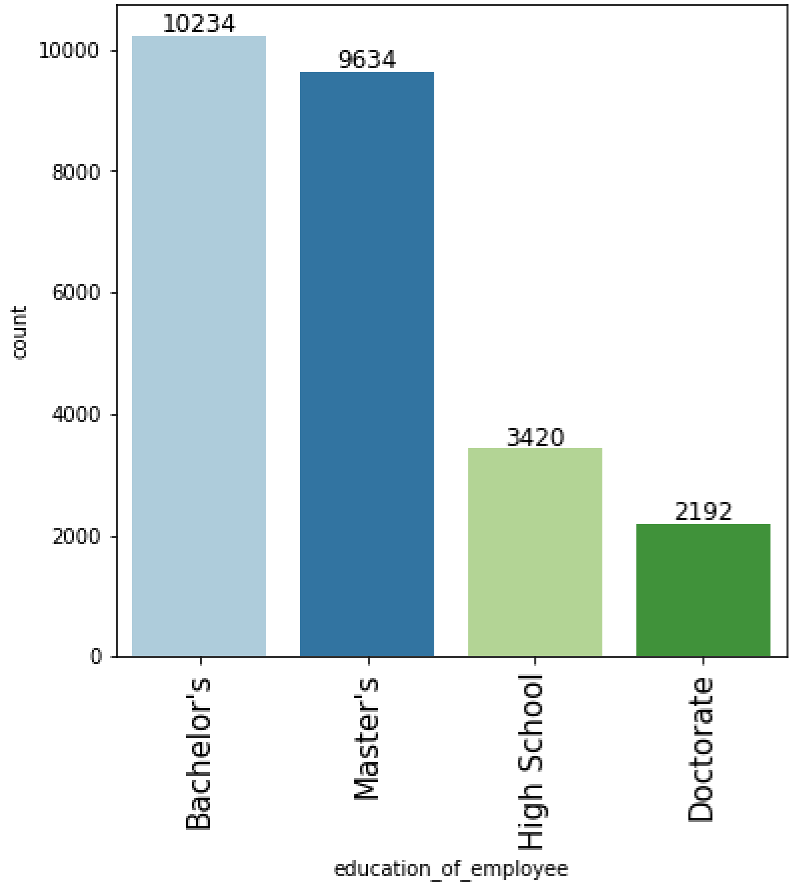
In [44]:

```python
labeled_barplot(data,"continent")
```



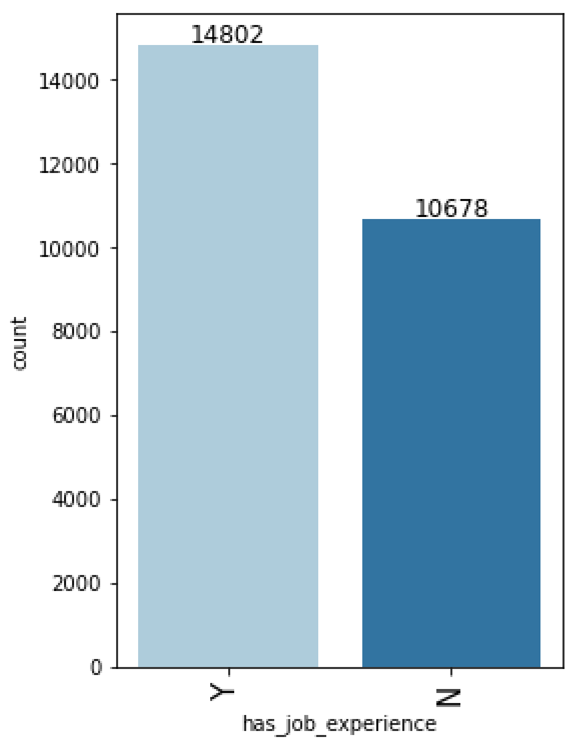
- There are six different continents listed in the dataset.
- Asia has the maximum entries at 16861 followed by Europe, North America, South America, Africa and Oceania at 192.

In [45]:
```
labeled_barplot(data,"education_of_employee")
```



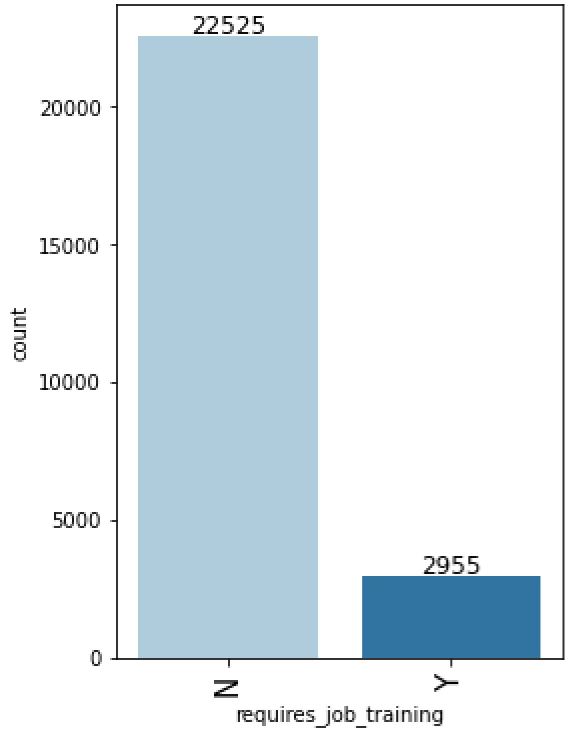
* The data has most people with atleast a Bachelor's degree, followed by Master's, High schoolers and PhDs.

In [48]:
```
labeled_barplot(data,"has_job_experience")
```



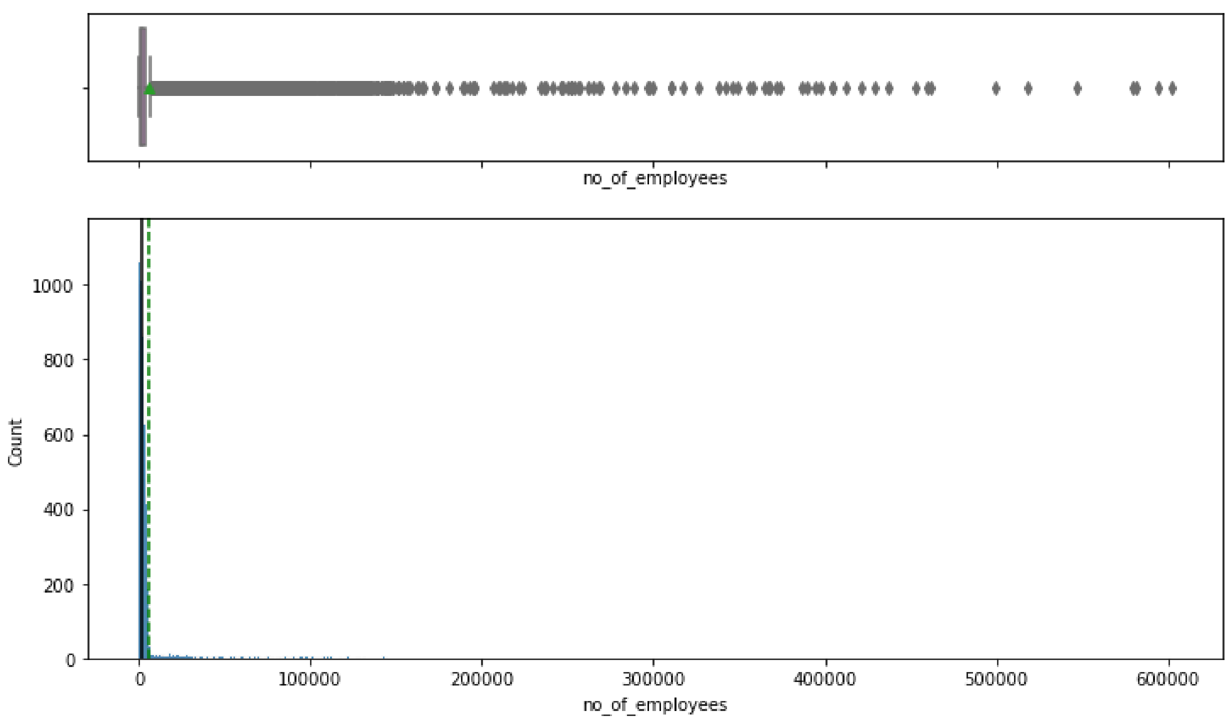
- The dataset has 14802 people that have job experience as opposed to 10678 who do not have a job experience.

In [49]:
```
labeled_barplot(data,"requires_job_training")
```



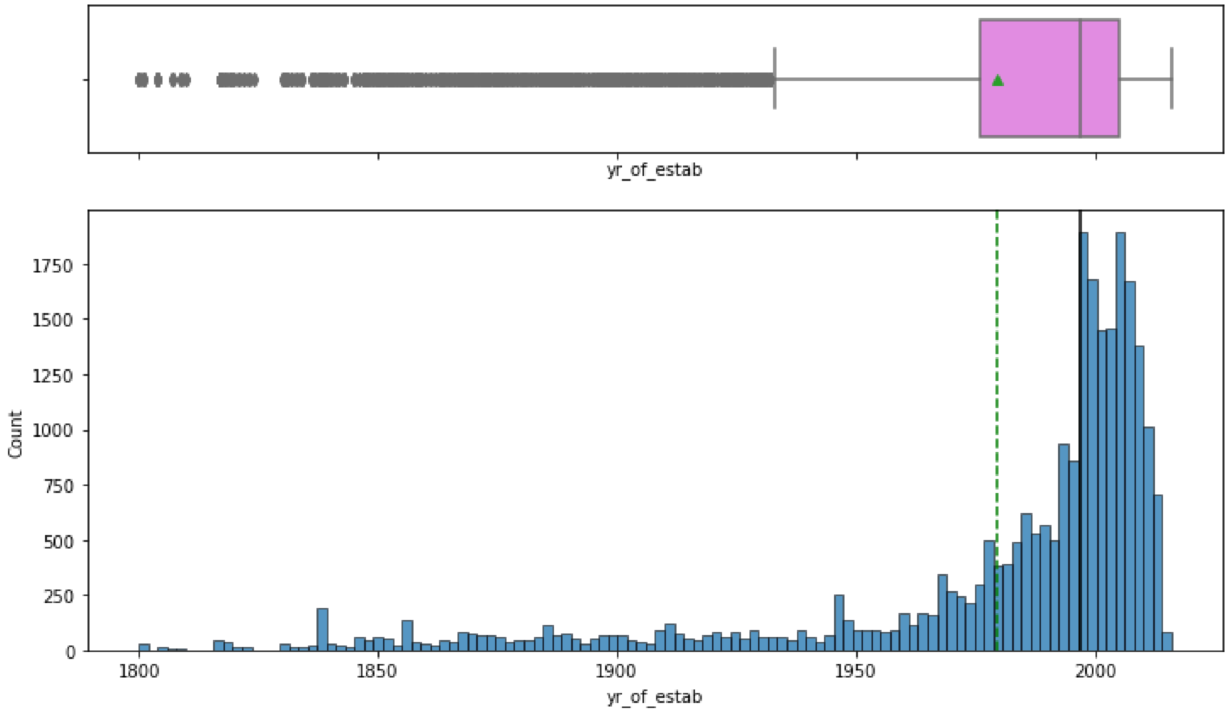
- Per the dataset, 22525 people do not require a job training as opposed to 2955 requiring it.

In [51]:
```
histogram_boxplot(data,"no_of_employees")
```



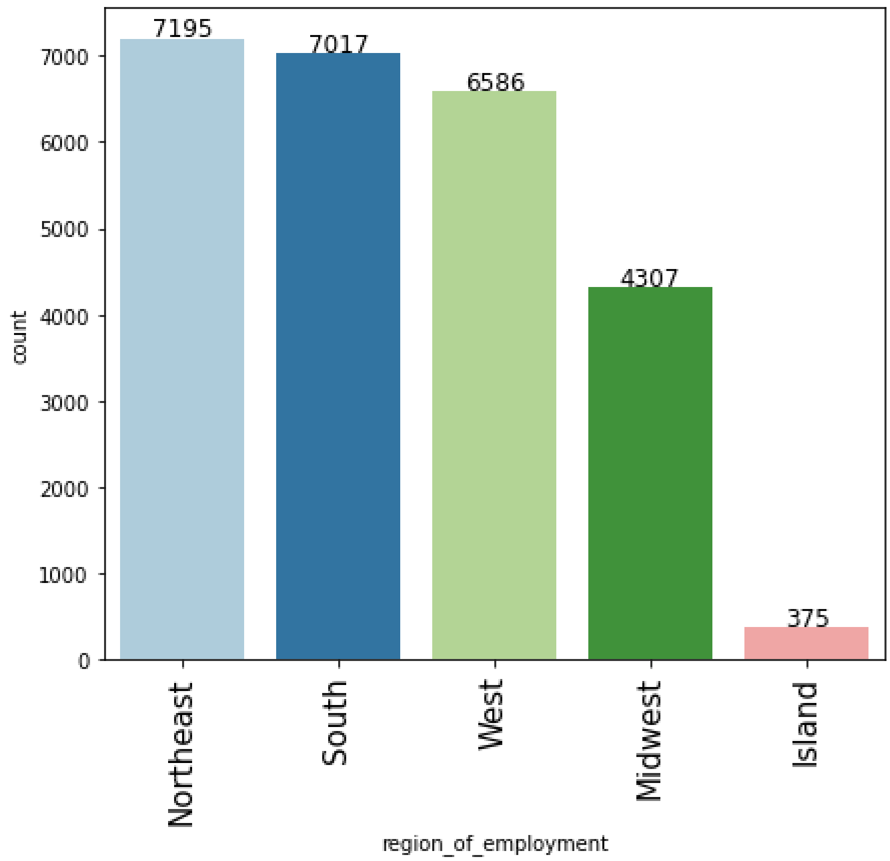
- Number of employees in a firm has a huge number of outliers. The data is heavily right skewed.

In [52]:
```
histogram_boxplot(data,'yr_of_estab')
```



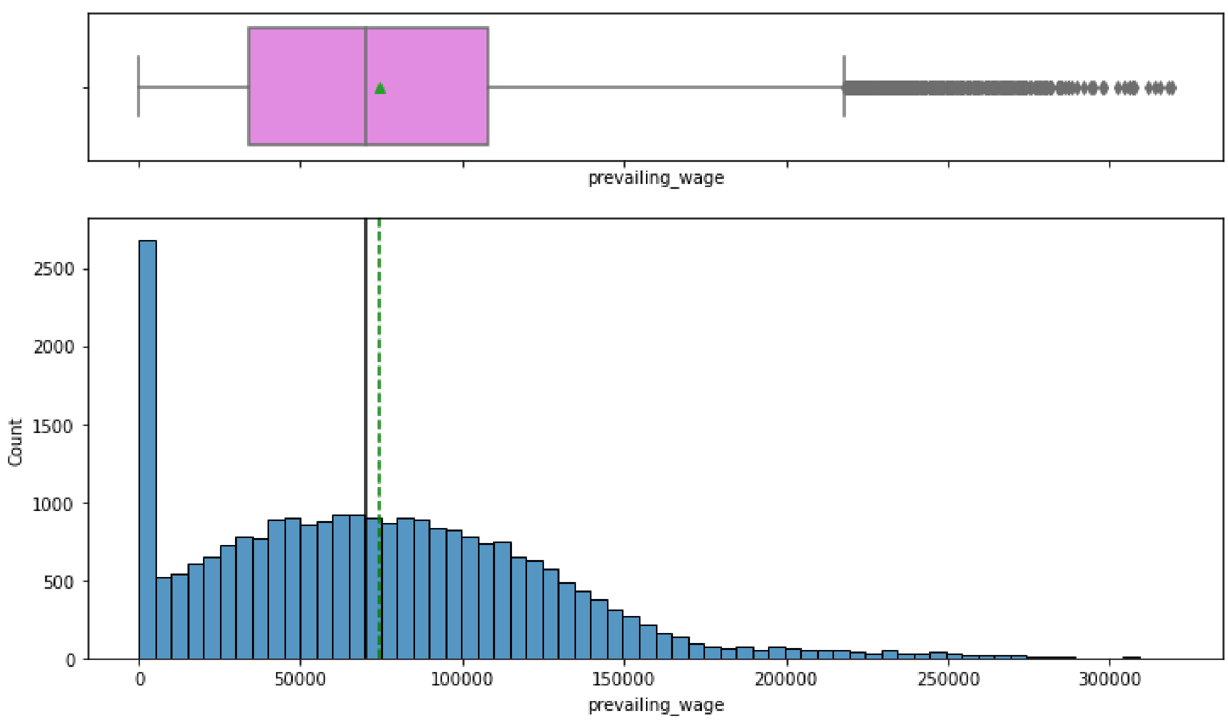
- This data is heavily left skewed. It appears that maximum organizations have been recently established. Very few organizations are old.

In [54]:
```
labeled_barplot(data,"region_of_employment")
```



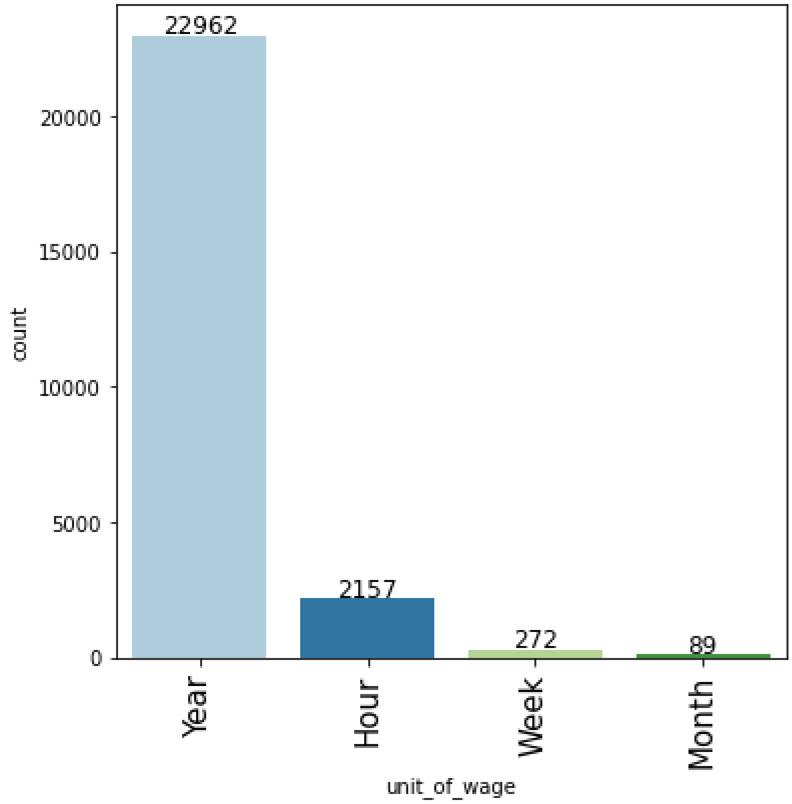
- Northeast region has the maximum number of entries, followed by South, West, Midwest and Island.

In [56]:
```
histogram_boxplot(data,'prevailing_wage')
```



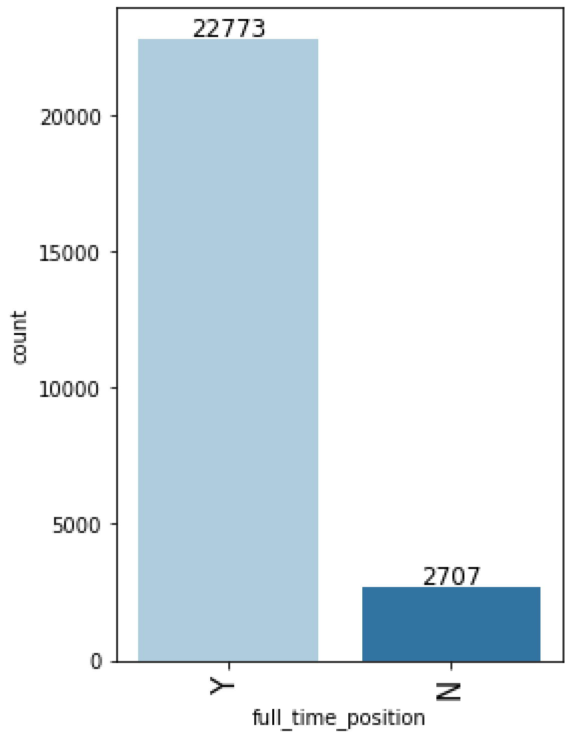
- This data appears to be right skewed. There are quite a few outliers.

In [57]:
```
labeled_barplot(data,"unit_of_wage")
```



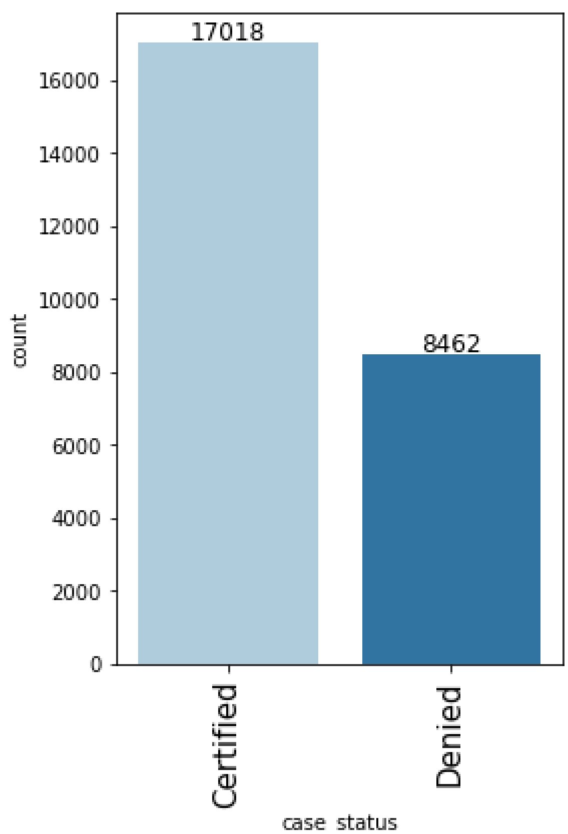
- Yearly compensation is the most popular mode of payment followed by Hourly, Weekly and Monthly mode.

In [59]:
```
labeled_barplot(data,"full_time_position")
```



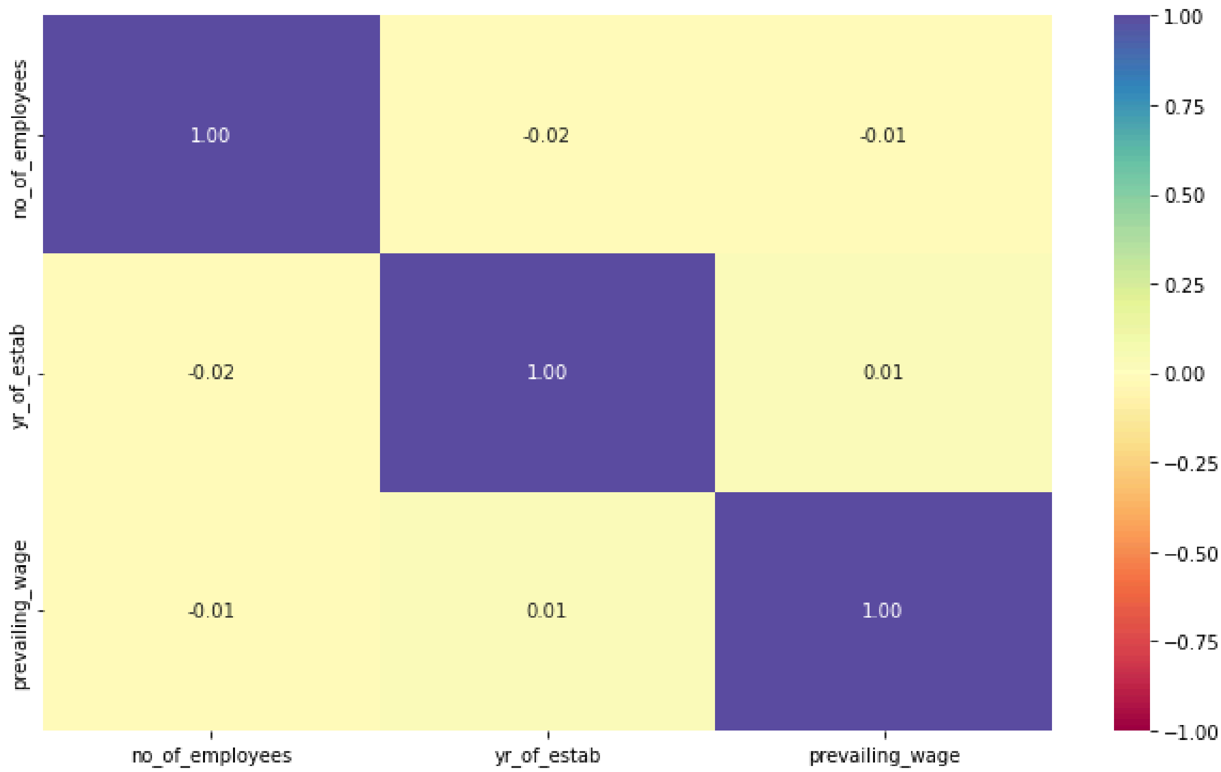
- The dataset has 22773 entries for full time positions as opposed to 2707 for part time positions.

In [60]:
```
labeled_barplot(data,"case_status")
```



- As per the dataset, visa was approved for 17018 and denied for 8462 people.

In [83]:
```
num_cols = data.select_dtypes(include=np.number).columns.tolist()

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

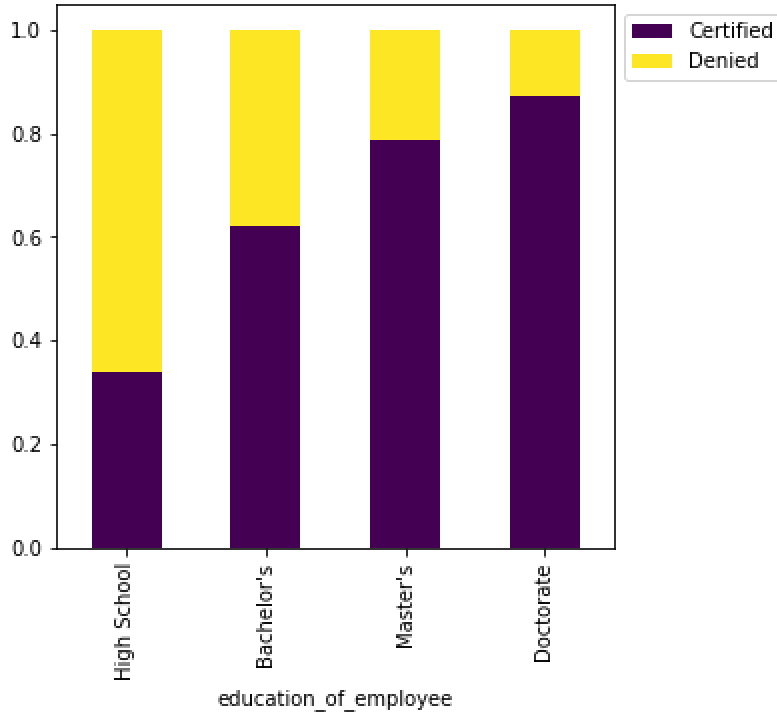


Leading Questions:

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

In [86]:
```
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
------------------------------------------------------------------------------------------------------------------------
```

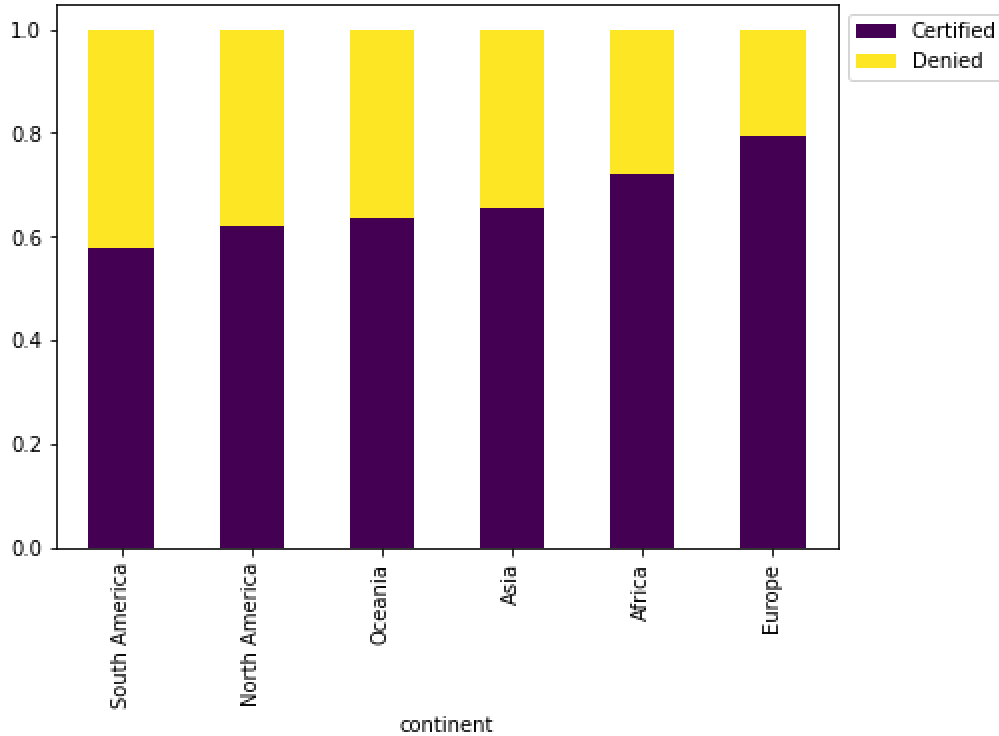


- Education does play a very important role in certifying visa. As we can see, as the level of education increases, the probability of getting a visa also increases.

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

In [87]:
```
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
------------------------------------------------------------------------------------------------------------------------
```



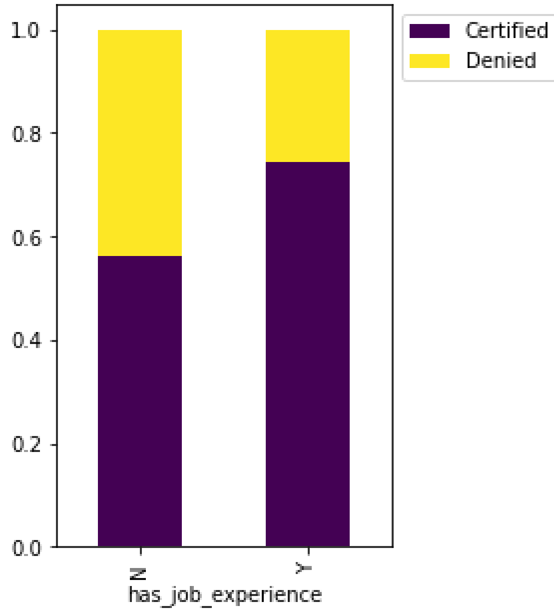
- It would not be very fair to say that visa approval depends on the continent of applicant due to the significant different in numbers across continents. But if we look at the percentages, it appears that there's a difference in approvals. Europe has a higher probability of approval than Africa and others. South America has the lowest percentage of approvals.

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

In [88]:

```
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
------------------------------------------------------------------------------------------------------------------------
```

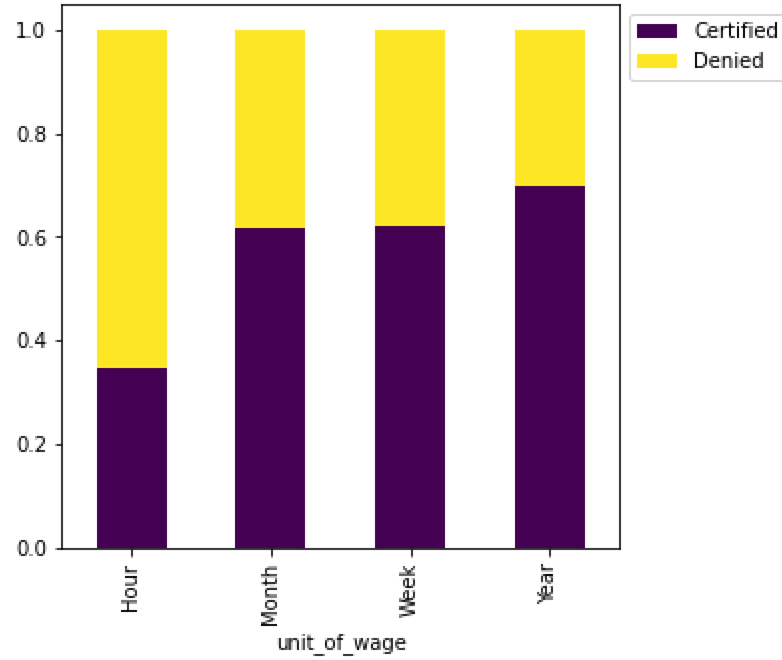


- Having job experience certainly increases the odds of visa approval. There's a high probability of getting visa if a person has job experience.

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

In [89]:
```
stacked_barplot(data, "unit_of_wage", "case_status")
```

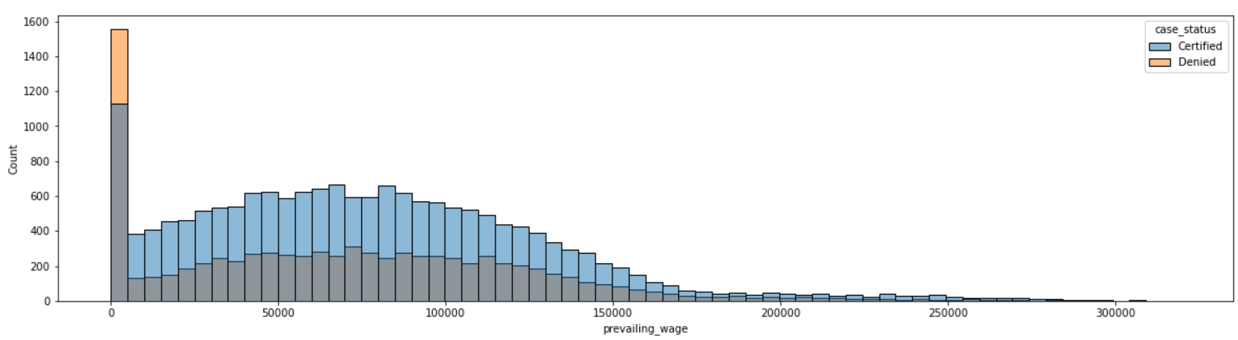
```
case_status   Certified  Denied    All
unit_of_wage
All               17018    8462  25480
Year              16047    6915  22962
Hour                747    1410   2157
Week                169     103    272
Month                55      34     89
-----------------------------------------------------------------------------------------------------------------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



- There's a high probability of getting visa approved if the employer offers YEARLY pay as opposed to HOURLY pay. Monthly and weekly payment modes have a fair chance of approval.

1. 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 [92]:
```
plt.figure(figsize=(20,5))
sns.histplot(data, x="prevailing_wage", hue="case_status");
```



- There are outliers in the data. There appears to be an even possibility within each prevailing wage, some slightly higher than others. But there is no significant variance except where wage appears to be 0 or close to 0. This needs to be investigated.

Data Preprocessing

- Missing value treatment (if needed)
- Feature engineering

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

In [101…

```
data.head()
```

Out[101]:

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

In [109…

```
data[data.no_of_employees<=0]
```

Out[109]:

	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees
245	Europe	Master's	N	N	-25
378	Asia	Bachelor's	N	Y	-11
832	South America	Master's	Y	N	-17
2918	Asia	Master's	Y	N	-26
6439	Asia	Bachelor's	N	N	-14
6634	Asia	Bachelor's	Y	N	-26
7224	Europe	Doctorate	N	N	-25
7281	Asia	High School	N	N	-14
7318	Asia	Bachelor's	Y	Y	-26
7761	Asia	Master's	N	N	-11
9872	Europe	Master's	Y	N	-26
11493	Asia	High School	Y	N	-14
13471	North America	Master's	N	N	-17
14022	Asia	Bachelor's	N	Y	-11
14146	Asia	Bachelor's	N	Y	-26
14726	Asia	Master's	N	N	-11
15600	Asia	Bachelor's	N	N	-14
15859	Asia	High School	N	N	-11
16157	Asia	Master's	Y	N	-11
16883	North America	Bachelor's	Y	N	-26
17006	Asia	Doctorate	Y	N	-11
17655	North America	Bachelor's	Y	N	-17
17844	Asia	Bachelor's	N	N	-14
17983	Asia	Bachelor's	N	N	-26
20815	Asia	Bachelor's	N	Y	-17
20984	Europe	Doctorate	Y	N	-14
21255	North America	High School	N	N	-25
21760	Asia	Bachelor's	Y	N	-25
21944	Africa	Master's	Y	N	-25
22084	North America	Bachelor's	Y	N	-14

	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees
22388	Asia	Master's	Y	N	-14
23186	Asia	Master's	N	Y	-11
23476	Europe	Master's	Y	N	-11

In [110…
```
data[data.no_of_employees<=0].shape
```
Out[110]:
```
(33, 11)
```

- These rows could either be deleted or the negative employee numbers be updated to positive. Updating them to positive(multiplying by -1) could alter the data that was intended for these rows. It would be a better decision to delete these 33 rows.

In [112…
```
#dropping the 33 rows with negative employees
data = data[data.no_of_employees>0]
data.shape
```
Out[112]:
```
(25447, 11)
```

In [113…
```
data["case_status"] = data["case_status"].apply(lambda x: 0 if x == "Denied" else 1)
```

- This would replace "Denied" with 0 and "Certified" with 1.

In [118…
```
#Separating features and the target column
X = data.drop("case_status", axis=1)
y = data["case_status"]
```

In [119…
```
# Creating dummy variables for all categorical variables
X = pd.get_dummies(X, drop_first=True)
```

In [122…
```
X.shape
```
Out[122]:
```
(25447, 21)
```

- The number of columns has increased to 21 after adding dummy variables.

In [123…
```
# Splitting data into training and test set:
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=
print(X_train.shape, X_test.shape)
```
```
(17812, 21) (7635, 21)
```

In [124…
```
y.value_counts(1)
```
Out[124]:
```
1    0.668094
0    0.331906
Name: case_status, dtype: float64
```

In [127…
```
y_train.value_counts(1)
```

Out[127]:
```
1    0.668089
0    0.331911
Name: case_status, dtype: float64
```

In [128…
```
y_test.value_counts(1)
```

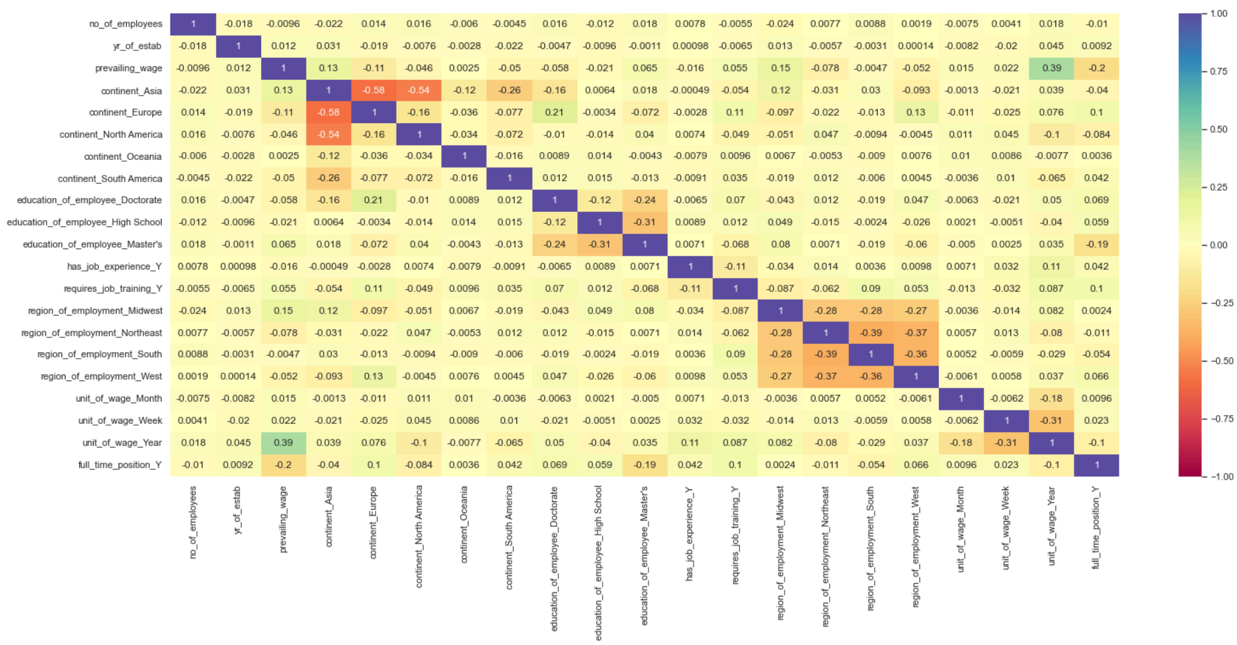
Out[128]:
```
1    0.668107
0    0.331893
Name: case_status, dtype: float64
```

- We have split the dataset into train and test.

EDA

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

In [132…
```
plt.figure(figsize=(25,10))
sns.heatmap(X.corr(),annot=True,vmin=-1,vmax=1,cmap="Spectral")
plt.show()
```



- We have deleted 33 rows that had negative number of employess and added dummy variables for all categorical variables. EDA here is different than what was observed before.

Building bagging and boosting models

Model evaluation criterion

The model can make wrong predictions as:

1. Predicting a person doesn't get visa when all required criterion are met.
2. Predicting a person does get visa when all required criterion are not met.

Which case is more important?

1. Predicting a person doesn't get visa when all required criterion are not met.
2. Predicting a person does get visa when all required criterion are met.

Which metric to optimize?

- We would want higher F1 score so that we can predict both the cases correctly.

Let's define a function to provide recall 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 [133…

```
# defining a function to compute different metrics to check performance of a classific
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

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

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

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

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

    return df_perf
```

In [134…

```
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")
```

In [144…

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

    '''
    # defining an empty list to store train and test results
    score_list=[]

    pred_train = model.predict(X_train)
    pred_test = model.predict(X_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)
    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)

    train_f1 = metrics.f1_score(y_train,pred_train)
    test_f1 = metrics.f1_score(y_test,pred_test)

    score_list.extend((train_acc,test_acc,train_recall,test_recall,train_precision,tes

    # If the flag is set to True then only the following print statements will be disp
    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,pred_train))
        print("Recall on test set : ",metrics.recall_score(y_test,pred_test))
        print("Precision on training set : ",metrics.precision_score(y_train,pred_trai
        print("Precision on test set : ",metrics.precision_score(y_test,pred_test))
        print("F1 on training set : ",metrics.f1_score(y_train,pred_train))
        print("F1 on test set : ",metrics.f1_score(y_test,pred_test))

    return score_list # returning the list with train and test scores
```

In [142…

```
## Function to create confusion matrix
def make_confusion_matrix(model,y_actual,labels=[1, 0]):
    '''
    model : classifier to predict values of X
    y_actual : ground truth

    '''
    y_predict = model.predict(X_test)
    cm=metrics.confusion_matrix( y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(cm, index = [i for i in ["Actual - No","Actual - Yes"]],
                  columns = [i for i in ['Predicted - No','Predicted - Yes']])
    group_counts = ["{0:0.0f}".format(value) for value in
```

```
                cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in
                         cm.flatten()/np.sum(cm)]
    labels = [f"{v1}\n{v2}" for v1, v2 in
              zip(group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    plt.figure(figsize = (10,7))
    sns.heatmap(df_cm, annot=labels,fmt='')
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```

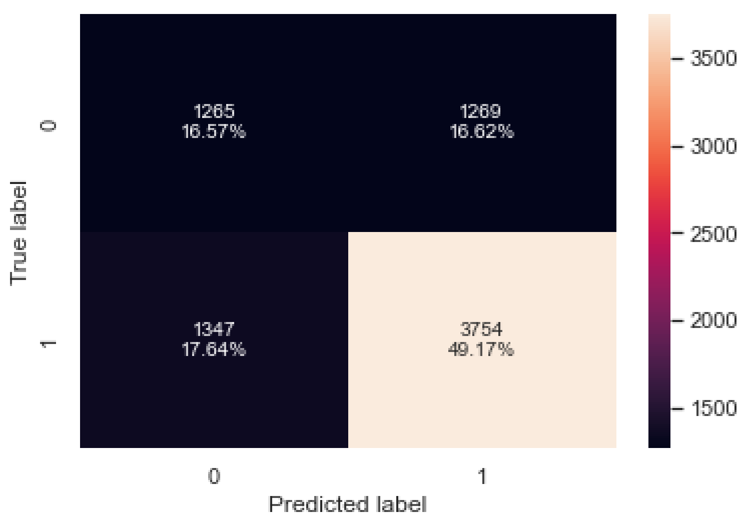
Model Building and Hyperparameter Tuning

Decision Tree Model

In [182…

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

#Calculating different metrics
dtree_model_train_perf=model_performance_classification_sklearn(d_tree,X_train,y_trair
print("Training performance:\n",dtree_model_train_perf)
dtree_model_test_perf=model_performance_classification_sklearn(d_tree,X_test,y_test)
print("Testing performance:\n",dtree_model_test_perf)
#Creating confusion matrix
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  0.657367  0.735934   0.747362  0.741604
```



- The decision tree is overfitting the training data as there is a significant difference between training and test scores for all the metrics.

Hyperparameter Tuning

In [158…

```python
#Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(class_weight={0:0.66,1:0.33},random_state=1)

# Grid of parameters to choose from
parameters = {'max_depth': np.arange(2,30),
              'min_samples_leaf': [1, 2, 5, 7, 10],
              'max_leaf_nodes' : [2, 3, 5, 10,15],
              'min_impurity_decrease': [0.0001,0.001,0.01,0.1]
             }

# 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)
```

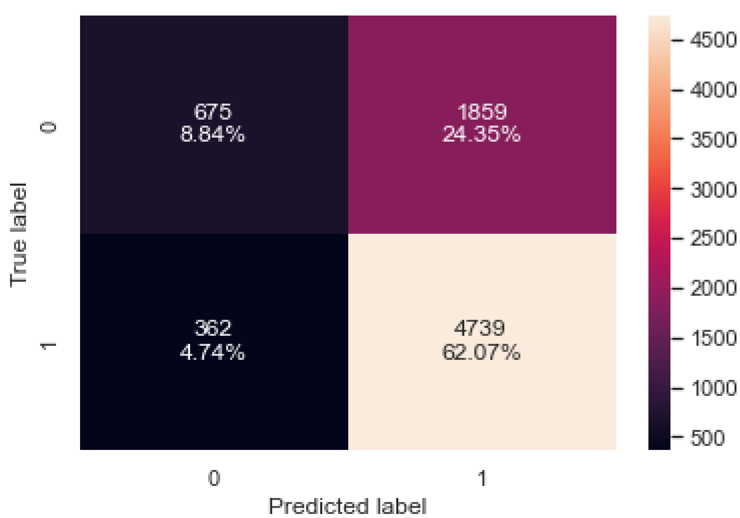
Out[158]:

```
DecisionTreeClassifier(class_weight={0: 0.66, 1: 0.33}, max_depth=2,
                       max_leaf_nodes=2, min_impurity_decrease=0.0001,
                       random_state=1)
```

In [159…

```python
#Calculating different metrics
dtree_estimator_model_train_perf=model_performance_classification_sklearn(dtree_estima
print("Training performance:\n",dtree_estimator_model_train_perf)
dtree_estimator_model_test_perf=model_performance_classification_sklearn(dtree_estimat
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  0.711599  0.932605   0.719108  0.812059
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.709103  0.929034   0.718248  0.810155
```



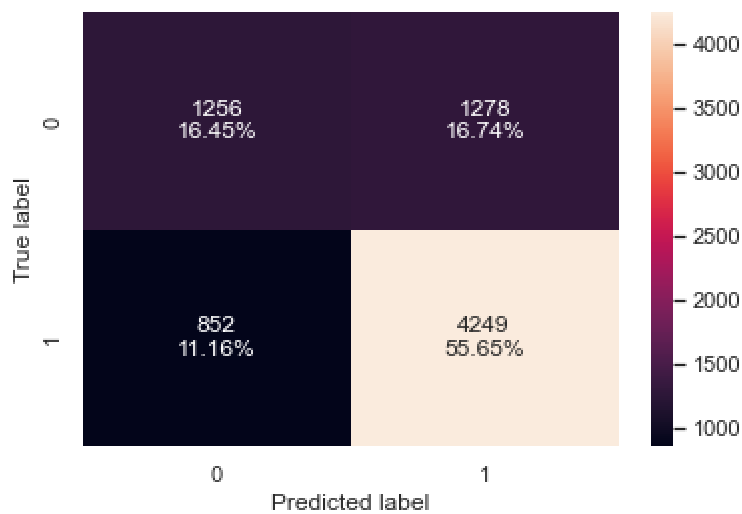
- The overall fitting has reduced and F1 score has increased.

Random Forest Model

In [137…

```
#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_
print("Training performance:\n",rf_estimator_model_train_perf)
rf_estimator_model_test_perf=model_performance_classification_sklearn(rf_estimator,X_t
print("Testing performance:\n",rf_estimator_model_test_perf)

#Creating confusion matrix
confusion_matrix_sklearn(rf_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  0.721022  0.832974   0.768771  0.799586
```



- Random forest is overfitting the training data as there is still difference between training and test scores for all the metrics.
- F1 score has increased

Hyperparameter Tuning

In [160…

```
# Choose the type of classifier.
rf_tuned = RandomForestClassifier(class_weight={0:0.668,1:0.332},random_state=1,oob_sc

parameters = {
                'max_depth': list(np.arange(5,30,5)) + [None],
                'max_features': ['sqrt',None],
                'min_samples_leaf': np.arange(1,15,5),
```

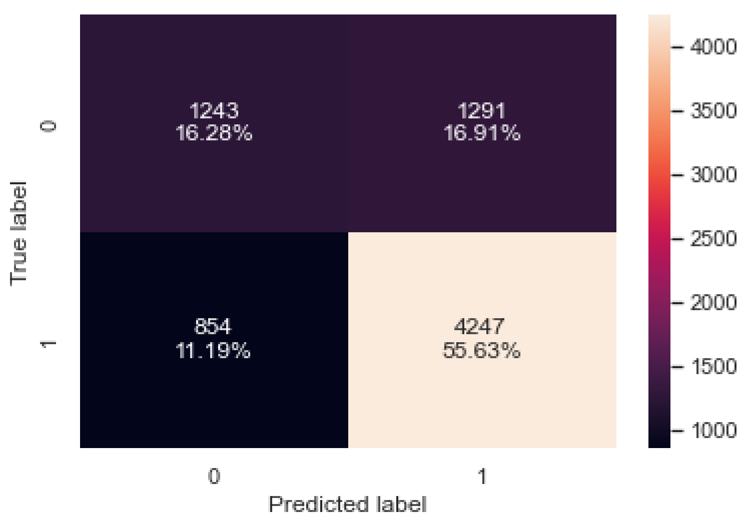
```
                'min_samples_split': np.arange(2, 20, 5),
                'n_estimators': np.arange(10,110,10)}


# Type of scoring used to compare parameter combinations
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[160]:
```
RandomForestClassifier(class_weight={0: 0.668, 1: 0.332}, max_depth=25,
                       max_features='sqrt', oob_score=True, random_state=1)
```

In [161…

```
#Calculating different metrics
rf_tuned_model_train_perf=model_performance_classification_sklearn(rf_tuned,X_train,y_
print("Training performance:\n",rf_tuned_model_train_perf)
rf_tuned_model_test_perf=model_performance_classification_sklearn(rf_tuned,X_test,y_te
print("Testing performance:\n",rf_tuned_model_test_perf)

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

```
Training performance:
    Accuracy    Recall  Precision       F1
0  0.997754  0.998571   0.998068  0.99832
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.719057  0.832582   0.766883  0.798383
```



Hyperparameter tuning has decreased the overfit and increased F1 score

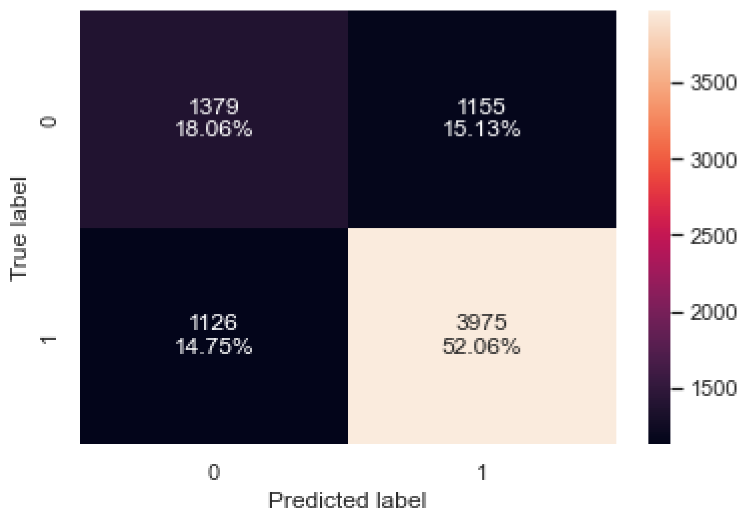
Bagging Classifier Model

In [138…

```
#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_c
print("Training performance:\n",bagging_classifier_model_train_perf)
bagging_classifier_model_test_perf=model_performance_classification_sklearn(bagging_cl
print("Testing performance:\n",bagging_classifier_model_test_perf)
#Creating confusion matrix
confusion_matrix_sklearn(bagging_classifier, X_test, y_test)
```

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.984673  0.985882    0.99113  0.988499
Testing performance:
    Accuracy    Recall  Precision       F1
0  0.701244  0.779259   0.774854  0.77705
```



- Bagging classifier has a lower F1 score than random forest.

Hyperparameter Tuning

In [162…

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

# Grid of parameters to choose from
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],
             }

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

# Run the grid search
grid_obj = GridSearchCV(bagging_estimator_tuned, parameters, scoring=scorer,cv=5)
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_
```

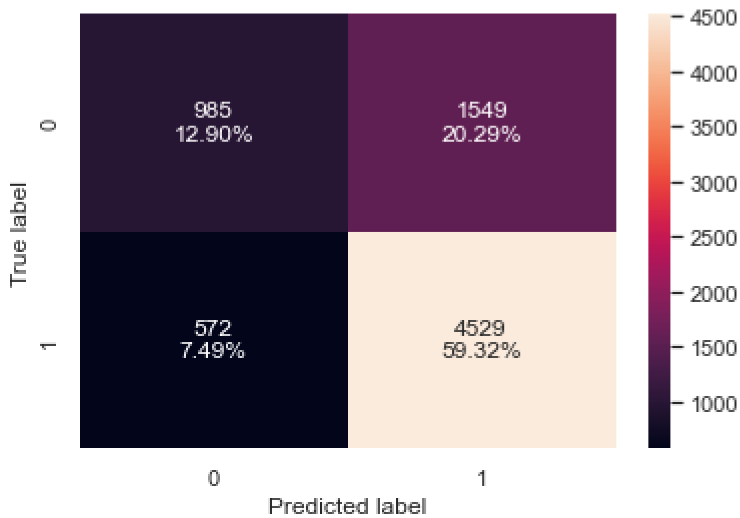
```
# Fit the best algorithm to the data.
bagging_estimator_tuned.fit(X_train, y_train)
```

Out[162]:
```
BaggingClassifier(max_features=0.7, max_samples=0.7, n_estimators=50,
                  random_state=1)
```

In [163…

```
#Calculating different metrics
bagging_estimator_tuned_model_train_perf=model_performance_classification_sklearn(bagg
print("Training performance:\n",bagging_estimator_tuned_model_train_perf)
bagging_estimator_tuned_model_test_perf=model_performance_classification_sklearn(baggi
print("Testing performance:\n",bagging_estimator_tuned_model_test_perf)

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

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.989894  0.999412   0.985662  0.992489
Testing performance:
    Accuracy    Recall  Precision        F1
0    0.7222  0.887865   0.745146  0.810269
```



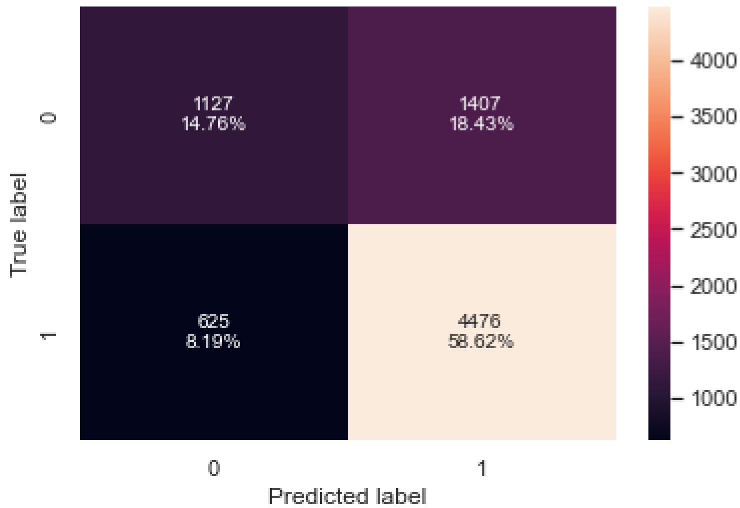
- Hyperparameter tuning of Bagging Classifier has reduced overfitting on training and increased F1 score.

AdaBoost Classifier

In [186…

```
#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,
print("Training performance:\n", ab_classifier_model_train_perf)
ab_classifier_model_test_perf=model_performance_classification_sklearn(ab_classifier,X
print("Testing performance:\n", ab_classifier_model_test_perf)

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

```
Training performance:
    Accuracy   Recall  Precision        F1
0  0.740568  0.89084   0.761402  0.821051
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.733857  0.877475   0.760836  0.815004
```



- The F1 score has increased

Hyperparameter Tuning

In [168…

```
# Choose the type of classifier.
abc_tuned = AdaBoostClassifier(random_state=1)

# Grid of parameters to choose from
## add from article
parameters = {
    #Let's try different max_depth for base_estimator
    "base_estimator":[DecisionTreeClassifier(max_depth=1, random_state=1),
                      DecisionTreeClassifier(max_depth=2, random_state=1),
                      DecisionTreeClassifier(max_depth=3, random_state=1)],
    "n_estimators": np.arange(10,110,10),
    "learning_rate":np.arange(0.1,2,0.1)
}

# 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=acc_scorer,cv=5)
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)
```

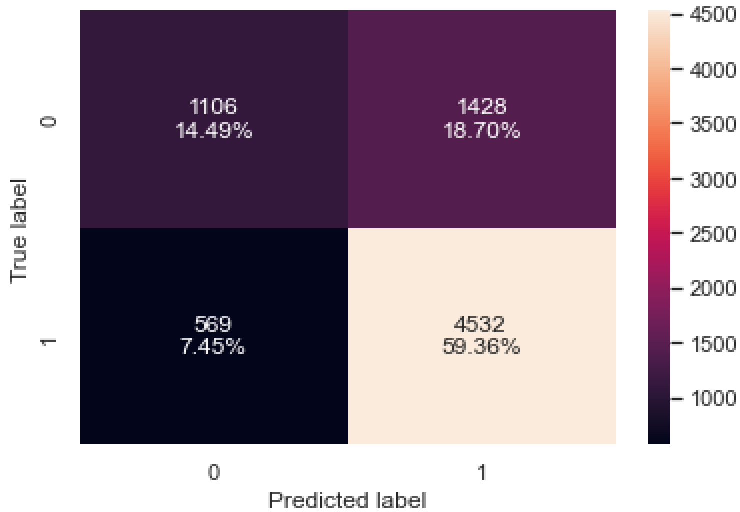
Out[168]:

```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3,
                                                         random_state=1),
                   learning_rate=0.4, n_estimators=10, random_state=1)
```

In [171…

```
#Calculating different metrics
abc_tuned_model_train_perf=model_performance_classification_sklearn(abc_tuned,X_train,
print(abc_tuned_model_train_perf)
abc_tuned_model_test_perf=model_performance_classification_sklearn(abc_tuned,X_test,y_
print(abc_tuned_model_test_perf)

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

```
   Accuracy   Recall  Precision        F1
0  0.745509  0.89916   0.762488  0.825203
   Accuracy    Recall  Precision        F1
0  0.738441  0.888453   0.760403  0.819456
```



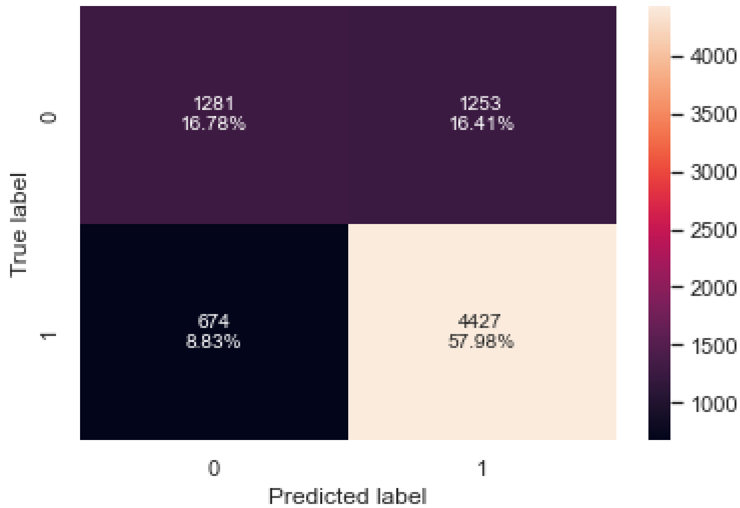
- There is not a very significant change upon hypertuning the bagging classifier model.

Gradient Boosting Classifier

In [190…

```
#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,
print("Training performance:\n",gb_classifier_model_train_perf)
gb_classifier_model_test_perf=model_performance_classification_sklearn(gb_classifier,X
print("Testing performance:\n",gb_classifier_model_test_perf)

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

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.757242  0.880504   0.783109  0.828956
Testing performance:
    Accuracy    Recall  Precision       F1
0   0.74761  0.867869   0.779401  0.82126
```



- We're observing a further improved F1 score.

Hyperparameter Tuning

In [172…

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

# Grid of parameters to choose from
parameters = {
    "n_estimators": [100,175,250],
    "subsample":[0.8,1],
    "max_features":[0.8,0.9,1]
}

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

# Run the grid search
grid_obj = GridSearchCV(gbc_tuned, parameters, scoring=scorer,cv=3)
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)
```

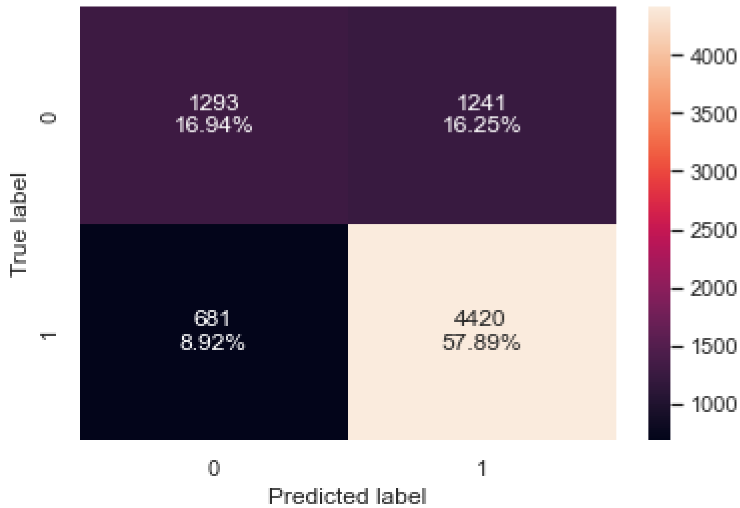
Out[172]:

```
GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                           max_features=0.9, random_state=1, subsample=0.8)
```

In [173…

```
#Calculating different metrics
gbc_tuned_model_train_perf=model_performance_classification_sklearn(gbc_tuned,X_train,
print("Training performance:\n",gbc_tuned_model_train_perf)
gbc_tuned_model_test_perf=model_performance_classification_sklearn(gbc_tuned,X_test,y_
print("Testing performance:\n",gbc_tuned_model_test_perf)

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

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.757804  0.879496   0.784205  0.829121
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.748265  0.866497   0.780781  0.821409
```



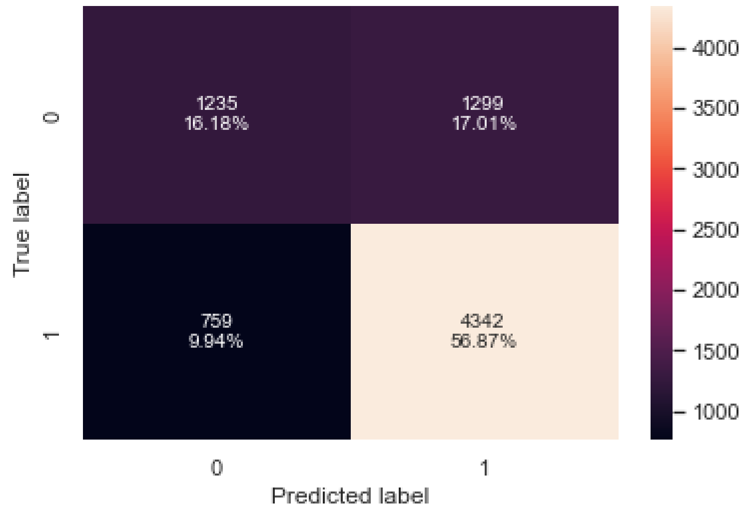
- The F1 score has increased but there isn't a very significant change.

XGBoost Classifier

In [189…

```
#Fitting the model
xgb_classifier = XGBClassifier(random_state=1, eval_metric='logloss')
xgb_classifier.fit(X_train,y_train)

#Calculating different metrics
xgb_classifier_model_train_perf=model_performance_classification_sklearn(xgb_classifie
print("Training performance:\n",xgb_classifier_model_train_perf)
xgb_classifier_model_test_perf=model_performance_classification_sklearn(xgb_classifier
print("Testing performance:\n",xgb_classifier_model_test_perf)

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

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.832922  0.928151   0.838903  0.881273
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.730452  0.851206   0.769722  0.808416
```



- We see a slightly lower F1(not significantly lower) than Gradient boosting.

Hyperparameter Tuning

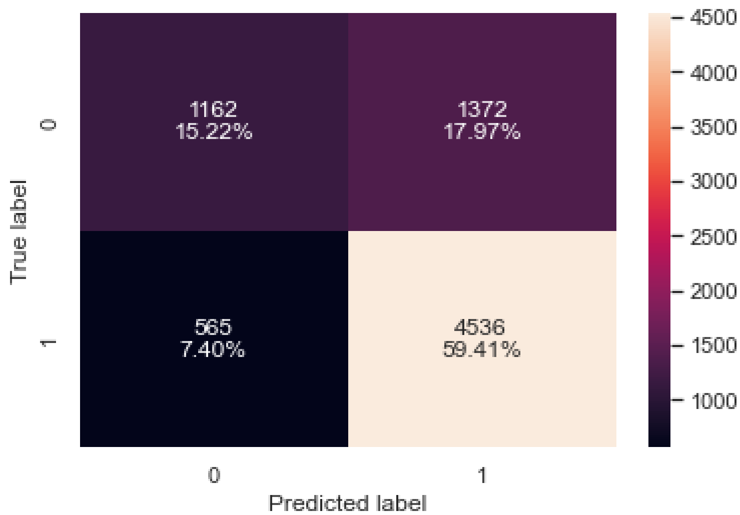
In [174…

```
# Choose the type of classifier.
xgb_tuned = XGBClassifier(random_state=1, eval_metric='logloss')

# Grid of parameters to choose from
parameters = {
    "n_estimators": [10,30,50],
    "scale_pos_weight":[1,2,5],
    "subsample":[0.7,0.9,1],
    "learning_rate":[0.05, 0.1,0.2],
    "colsample_bytree":[0.7,0.9,1],
    "colsample_bylevel":[0.5,0.7,1]
}

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

# Run the grid search
grid_obj = GridSearchCV(xgb_tuned, parameters,scoring=scorer,cv=3)
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[174]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=0.7, colsample_bynode=None,
              colsample_bytree=0.7, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, gpu_id=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.05, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, n_estimators=30, n_jobs=None,
              num_parallel_tree=None, predictor=None, random_state=1, ...)
```

In [175…
```
#Calculating different metrics
xgb_tuned_model_train_perf=model_performance_classification_sklearn(xgb_tuned,X_train,
print("Training performance:\n",xgb_tuned_model_train_perf)
xgb_tuned_model_test_perf=model_performance_classification_sklearn(xgb_tuned,X_test,y_
print("Testing performance:\n",xgb_tuned_model_test_perf)

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

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.756288  0.903109   0.771224  0.831972
Testing performance:
    Accuracy    Recall  Precision        F1
0    0.7463  0.889237   0.767773  0.824053
```



- The F1 score has slightly increased and is the highest amongst all the models.

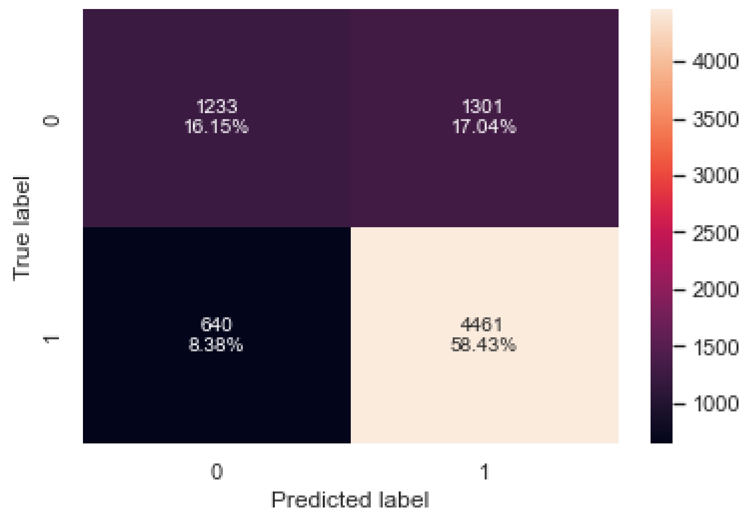
Stacking Model

In [178…
```
estimators = [('Random Forest',rf_tuned), ('Gradient Boosting',gbc_tuned), ('Decision

final_estimator = xgb_tuned

stacking_classifier= StackingClassifier(estimators=estimators,final_estimator=final_es

stacking_classifier.fit(X_train,y_train)
```

Out[178]:
```
StackingClassifier(estimators=[('Random Forest',
                                RandomForestClassifier(class_weight={0: 0.668,
                                                                     1: 0.332},
                                                       max_depth=25,
                                                       max_features='sqrt',
                                                       oob_score=True,
                                                       random_state=1)),
                               ('Gradient Boosting',
                                GradientBoostingClassifier(init=AdaBoostClassifier(ra
ndom_state=1),
                                                           max_features=0.9,
                                                           random_state=1,
                                                           subsample=0.8)),
                               ('Decision Tree',
                                DecisionTreeClassifier(class_weig...
                                                 gpu_id=None, grow_policy=None,
                                                 importance_type=None,
                                                 interaction_constraints=None,
                                                 learning_rate=0.05,
                                                 max_bin=None,
                                                 max_cat_threshold=None,
                                                 max_cat_to_onehot=None,
                                                 max_delta_step=None,
                                                 max_depth=None,
                                                 max_leaves=None,
                                                 min_child_weight=None,
                                                 missing=nan,
                                                 monotone_constraints=None,
                                                 n_estimators=30, n_jobs=None,
                                                 num_parallel_tree=None,
                                                 predictor=None, random_state=1,
...))
```

In [179…
```python
#Calculating different metrics
stacking_classifier_model_train_perf=model_performance_classification_sklearn(stacking
print("Training performance:\n",stacking_classifier_model_train_perf)
stacking_classifier_model_test_perf=model_performance_classification_sklearn(stacking_
print("Testing performance:\n",stacking_classifier_model_test_perf)

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

```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.802942  0.928908   0.805803  0.862987
Testing performance:
    Accuracy    Recall  Precision       F1
0  0.745776  0.874534    0.77421  0.82132
```



- The F1 score has increased but there isn't a very significant change.

Model Performance Comparison and Conclusions

In [192…

```
# training performance comparison

models_train_comp_df = pd.concat(
    [dtree_model_train_perf.T,dtree_estimator_model_train_perf.T,rf_estimator_model_tr
     bagging_classifier_model_train_perf.T,bagging_estimator_tuned_model_train_perf.T,
     abc_tuned_model_train_perf.T,gb_classifier_model_train_perf.T,gbc_tuned_model_tra
    xgb_tuned_model_train_perf.T,stacking_classifier_model_train_perf.T],
    axis=1,
)
models_train_comp_df.columns = [
    "Decision Tree",
    "Decision Tree Tuned",
    "Random Forest",
    "Random Forest Tuned",
    "Bagging Classifier",
    "Bagging Estimator Tuned",
    "Adaboost Classifier",
    "Adabosst Classifier Tuned",
    "Gradient Boost Classifier",
    "Gradient Boost Classifier Tuned",
    "XGBoost Classifier",
    "XGBoost Classifier Tuned",
    "Stacking Classifier"]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

Out[192]:

	Decision Tree	Decision Tree Tuned	Random Forest	Random Forest Tuned	Bagging Classifier	Bagging Estimator Tuned	Adaboost Classifier	Adabosst Classifier Tuned	Gradient Boost Classifier
Accuracy	1.0	0.711599	1.0	0.997754	0.984673	0.989894	0.740568	0.745509	0.757242
Recall	1.0	0.932605	1.0	0.998571	0.985882	0.999412	0.890840	0.899160	0.880504
Precision	1.0	0.719108	1.0	0.998068	0.991130	0.985662	0.761402	0.762488	0.783109
F1	1.0	0.812059	1.0	0.998320	0.988499	0.992489	0.821051	0.825203	0.828956

In [193…

```python
# testing performance comparison

models_test_comp_df = pd.concat(
    [dtree_model_test_perf.T,dtree_estimator_model_test_perf.T,rf_estimator_model_test
     bagging_classifier_model_test_perf.T,bagging_estimator_tuned_model_test_perf.T,ab
     abc_tuned_model_test_perf.T,gb_classifier_model_test_perf.T,gbc_tuned_model_test_
    xgb_tuned_model_test_perf.T,stacking_classifier_model_test_perf.T],
    axis=1,
)
models_test_comp_df.columns = [
    "Decision Tree",
    "Decision Tree Tuned",
    "Random Forest",
    "Random Forest Tuned",
    "Bagging Classifier",
    "Bagging Estimator Tuned",
    "Adaboost Classifier",
    "Adabosst Classifier Tuned",
    "Gradient Boost Classifier",
    "Gradient Boost Classifier Tuned",
    "XGBoost Classifier",
    "XGBoost Classifier Tuned",
    "Stacking Classifier"]
print("Testing performance comparison:")
models_test_comp_df
```

Testing performance comparison:

Out[193]:

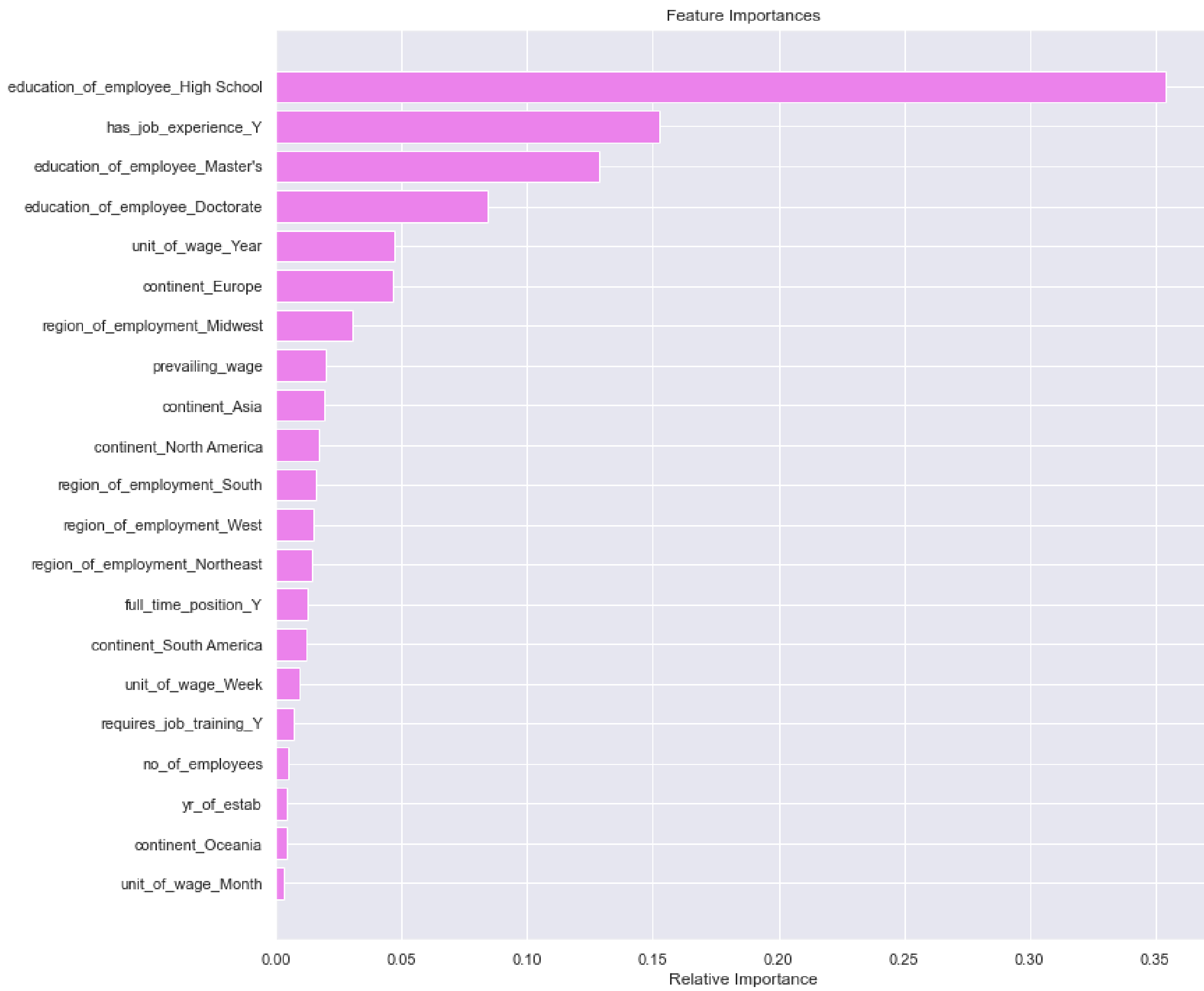
	Decision Tree	Decision Tree Tuned	Random Forest	Random Forest Tuned	Bagging Classifier	Bagging Estimator Tuned	Adaboost Classifier	Adabosst Classifier Tuned	Gradient Boost Classifier
Accuracy	0.657367	0.709103	0.721022	0.719057	0.701244	0.722200	0.733857	0.738441	0.747610
Recall	0.735934	0.929034	0.832974	0.832582	0.779259	0.887865	0.877475	0.888453	0.867869
Precision	0.747362	0.718248	0.768771	0.766883	0.774854	0.745146	0.760836	0.760403	0.779401
F1	0.741604	0.810155	0.799586	0.798383	0.777050	0.810269	0.815004	0.819456	0.821260

- Most models have a nearly same F1 score in training and test, except the ones that did overfit the data.
- XG Boost tuned has a slightly higher F1 than all others.

Feature Importance of Tuned XGB

In [194…

```
feature_names = X_train.columns
importances = xgb_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()
```



Actionable Insights and Recommendations

- The most important parameters in certifying visa appear to be education of the associate, their job experience and their wage.
  - Education : An uptick in higher education certainly increases the odds of visa approval as compared to being a highschool diploma holder.
  - Job Experience : Previously held job experience certainly increases the odds of visa approval over no prior job experience.

    - Yearly offered pay in accordance to prevailing wage certainly gives a confidence to the government that there is no abuse of foreign labor

- In order to process applications that have a higher chance of approval, OFLC can maybe prioritize applications that meet the above mentioned parameters. There would then be minimal necessary checks needed to certify or deny the visa
- A previous history of the organization filing for the visa could be an important factor, as in have they abused the visa policies before or have always complied.
- With the Tuned XGB model we've received an F1 score of 82% but with additional data/parameters a higher F1 could've been achieved that would've ensured the predition of cases correctly ensuring that those who deserve visa get it.