EasyVisa Project

Marks: 60

Problem Statement

Context:

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

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

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

Objective:

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

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

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

Data Description

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

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

Please read the instructions carefully before starting the project.

This is a commented Jupyter IPython Notebook file in which all the instructions and tasks to be performed are mentioned.

- Blanks '**___**' are provided in the notebook that needs to be filled with an appropriate code to get the correct result. With every '**___**' blank, there is a comment that briefly describes what needs to be filled in the blank space.
- Identify the task to be performed correctly, and only then proceed to write the required code.
- Fill the code wherever asked by the commented lines like "# write your code here" or "# complete the code". Running incomplete code may throw error.
- Please run the codes in a sequential manner from the beginning to avoid any unnecessary errors.
- Add the results/observations (wherever mentioned) derived from the analysis in the presentation and submit the same.

Importing necessary libraries

In [1]:
```
# this will help in making the Python code more structured automatically
'%load_ext nb_black'

import warnings

warnings.filterwarnings("ignore")

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

# Library to split data
from sklearn.model_selection import train_test_split

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

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


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

from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier

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

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

Importing Dataset

In [2]:
```
visa = pd.read_csv('EasyVisa.csv') ##  Fill the blank to read the data
```

In [3]:
```
# copying data to another variable to avoid any changes to original data
data = visa.copy()
```

Overview of the Dataset

View the first and last 5 rows of the dataset

In [4]:
```
data.head() ##  Complete the code to view top 5 rows of the data
```

Out[4]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_training
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() ##  Complete the code to view last 5 rows of the data
```

Out[5]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_
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	

Understand the shape of the dataset

In [6]:
```
data.shape ##  Complete the code to view dimensions of the data
```

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

Check the data types of the columns for the dataset

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
```

In [8]:

```
# checking for duplicate values
data.duplicated().sum() ##  Complete the code to check duplicate entries
```

Out[8]:

```
0
```

Exploratory Data Analysis

Let's check the statistical summary of the data

In [9]:

```
data.describe() ##  Complete the code to print the statistical summary of
```

Out[9]:

	no_of_employees	yr_of_estab	prevailing_wage
count	25480.000000	25480.000000	25480.000000
mean	5667.043210	1979.409929	74455.814592
std	22877.928848	42.366929	52815.942327
min	-26.000000	1800.000000	2.136700
25%	1022.000000	1976.000000	34015.480000
50%	2109.000000	1997.000000	70308.210000
75%	3504.000000	2005.000000	107735.512500
max	602069.000000	2016.000000	319210.270000

Fixing the negative values in number of employees columns

In [10]:

```
data.loc[data['no_of_employees']<0].shape ## Complete the code to check n
```

Out[10]:

```
(33, 12)
```

In [11]:
```
# taking the absolute values for number of employees
data["no_of_employees"] = abs(data["no_of_employees"]) ## Write the funct
```

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

In [12]:
```
# Making a list of all catrgorical variables
cat_col = list(data.select_dtypes("object").columns)

# Printing number of count of each unique value in each column
for column in cat_col:
    print(data[column].value_counts())
    print("-" * 50)
```

```
EZYV01       1
EZYV16995    1
EZYV16993    1
EZYV16992    1
EZYV16991    1
            ..
EZYV8492     1
EZYV8491     1
EZYV8490     1
EZYV8489     1
EZYV25480    1
Name: case_id, Length: 25480, dtype: int64
--------------------------------------------------
Asia             16861
Europe            3732
North America     3292
South America      852
Africa             551
Oceania            192
Name: continent, dtype: int64
--------------------------------------------------
Bachelor's     10234
Master's        9634
High School     3420
Doctorate       2192
Name: education_of_employee, dtype: int64
--------------------------------------------------
Y    14802
N    10678
Name: has_job_experience, dtype: int64
--------------------------------------------------
N    22525
Y     2955
Name: requires_job_training, dtype: int64
--------------------------------------------------
Northeast    7195
South        7017
West         6586
Midwest      4307
Island        375
Name: region_of_employment, dtype: int64
--------------------------------------------------
Year     22962
Hour      2157
Week       272
Month       89
Name: unit_of_wage, dtype: int64
--------------------------------------------------
Y    22773
N     2707
Name: full_time_position, dtype: int64
--------------------------------------------------
Certified    17018
Denied        8462
Name: case_status, dtype: int64
--------------------------------------------------
```

In [13]:
```
# checking the number of unique values
data["case_id"].nunique() ## Complete the code to check unique values in
```

Out[13]: 25480

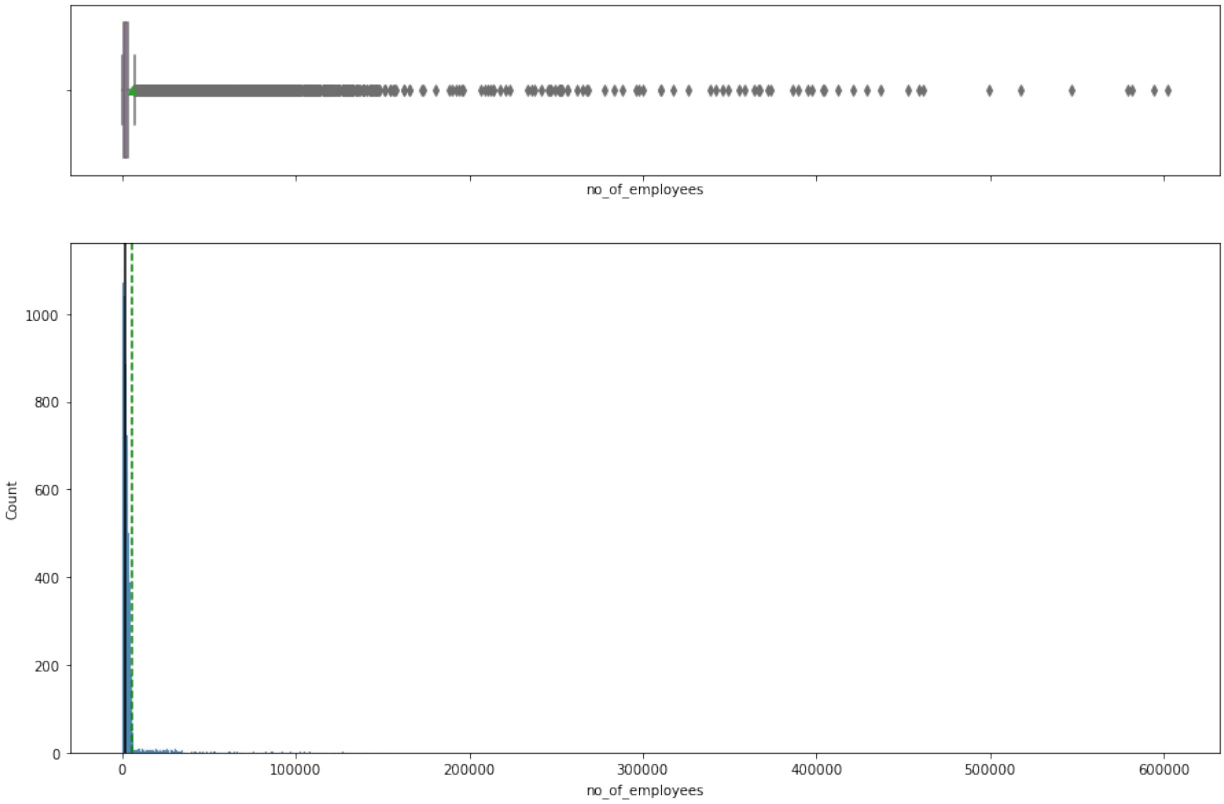
In [14]:
```
data.drop(["case_id"], axis=1, inplace=True) ## Complete the code to drop
```

Univariate Analysis

In [15]:
```
def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=No
    """
    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 va
    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
```

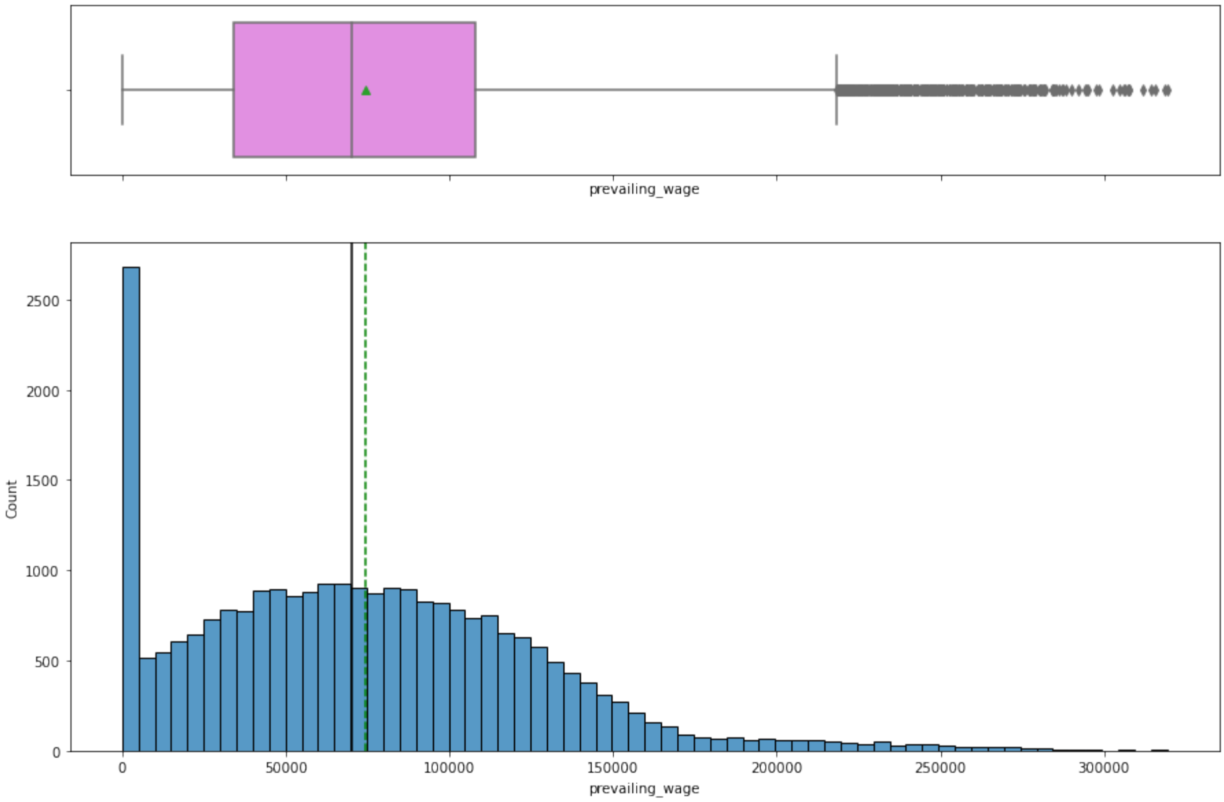
Observations on number of employees

In [16]:
```
histogram_boxplot(data, "no_of_employees")
```



Observations on prevailing wage

In [17]:
```
histogram_boxplot(data, "prevailing_wage") ## Complete the code to create
```



In [18]:
```
# checking the observations which have less than 100 prevailing wage
data.loc[data["prevailing_wage"] < 100] ## Complete the code to find the
```

Out[18]:

	continent	education_of_employee	has_job_experience	requires_job_training	no
338	Asia	Bachelor's	Y	N	
634	Asia	Master's	N	N	
839	Asia	High School	Y	N	
876	South America	Bachelor's	Y	N	
995	Asia	Master's	N	N	
...	...	...	...	...	
25023	Asia	Bachelor's	N	Y	
25258	Asia	Bachelor's	Y	N	
25308	North America	Master's	N	N	
25329	Africa	Bachelor's	N	N	
25461	Asia	Master's	Y	N	

176 rows × 11 columns

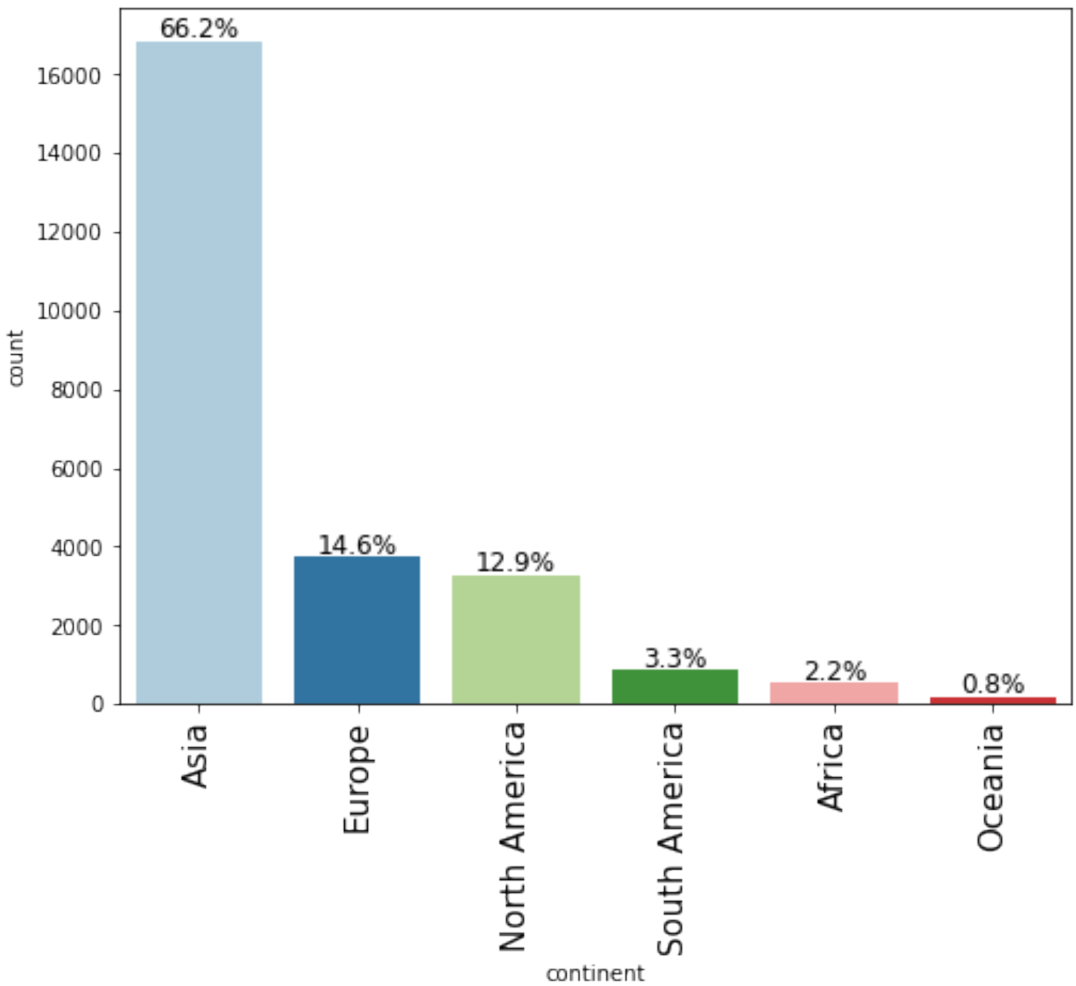
In [19]:
```
data.loc[data["prevailing_wage"] < 100, "unit_of_wage"].value_counts() ##
```

Out[19]:
```
Hour    176
Name: unit_of_wage, dtype: int64
```

In [20]:
```
# 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 Fal
    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
```

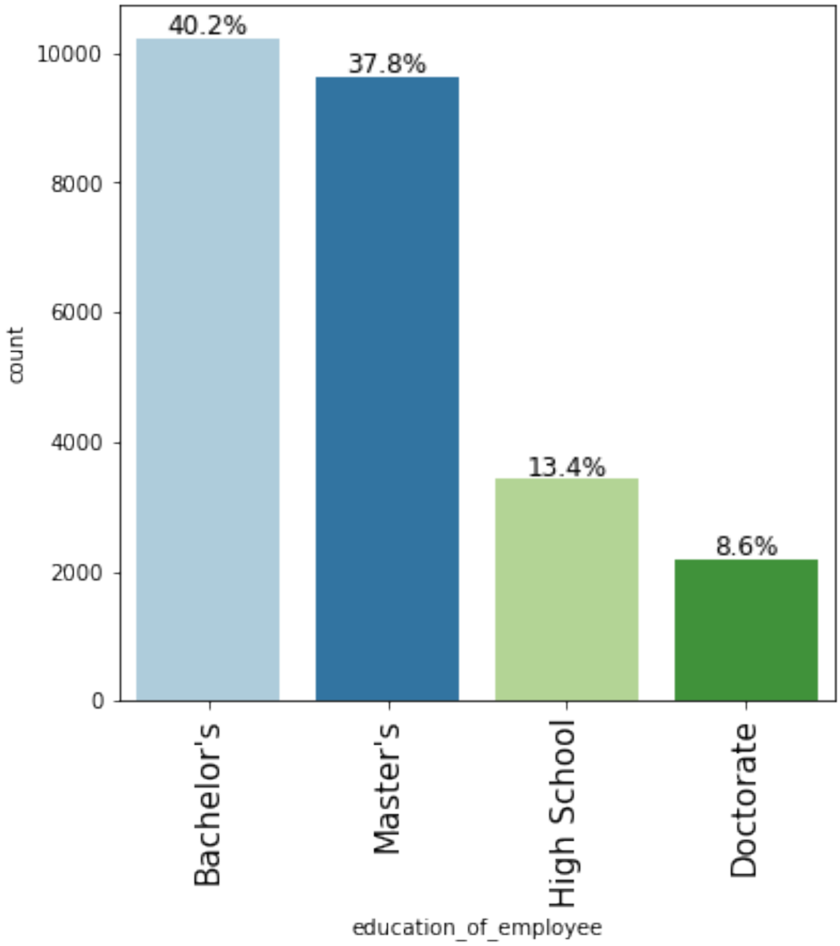
Observations on continent

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



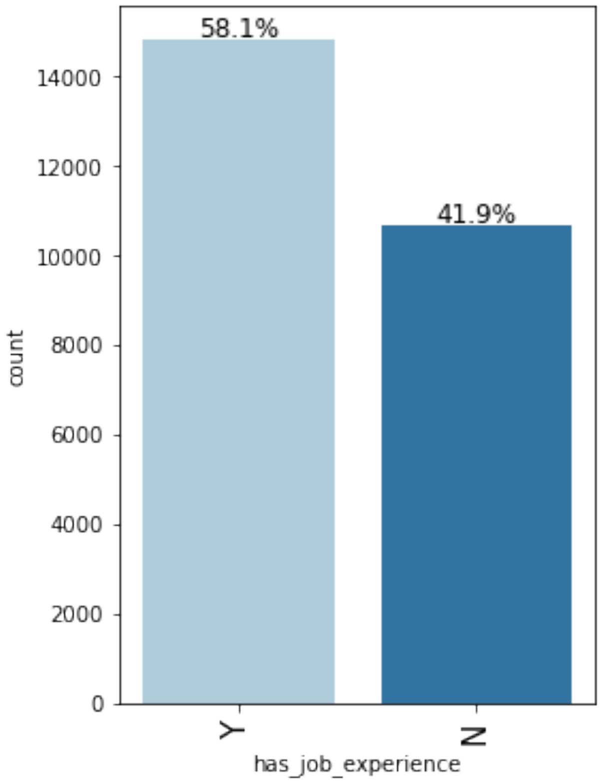
Observations on education of employee

In [22]:
```
labeled_barplot(data, "education_of_employee", perc=True)   ## Complete t
```



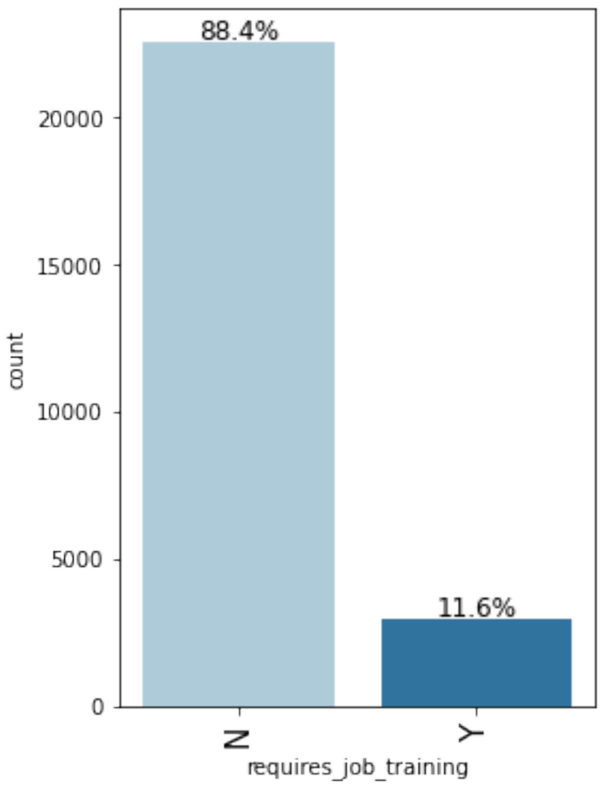
Observations on job experience

In [23]:
```
labeled_barplot(data, "has_job_experience", perc=True) ## Complete the co
```



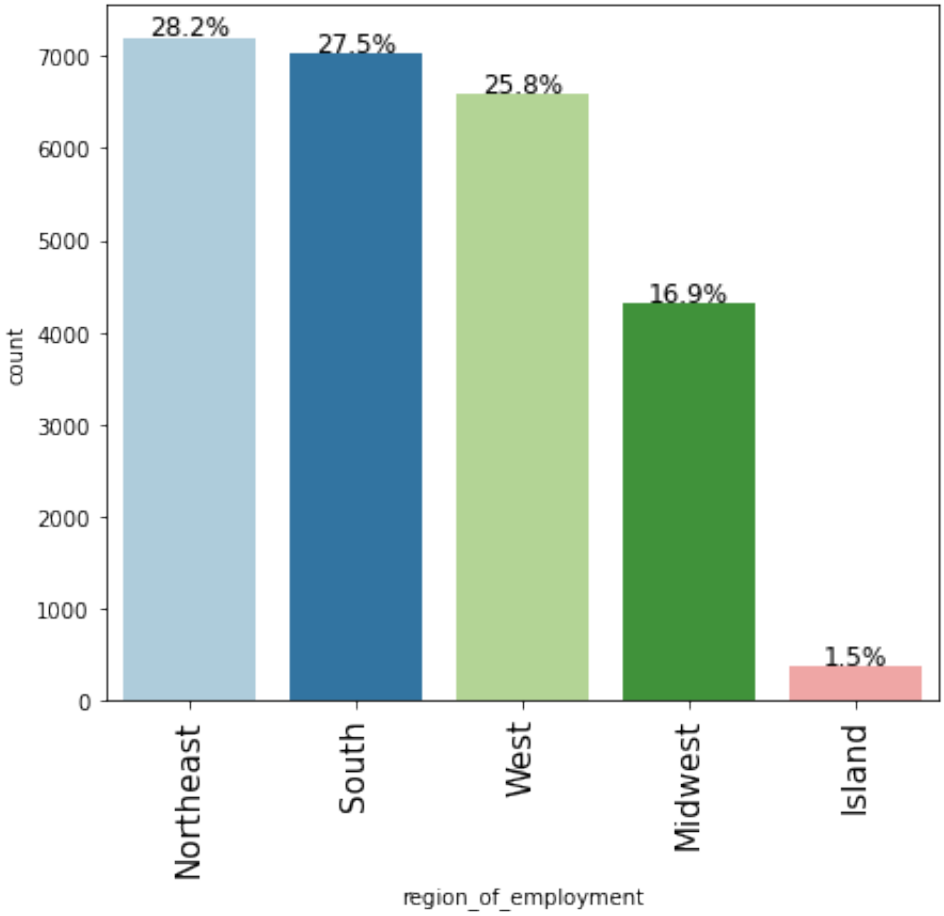
Observations on job training

In [24]:
```
labeled_barplot(data, "requires_job_training", perc=True)  ## Complete th
```



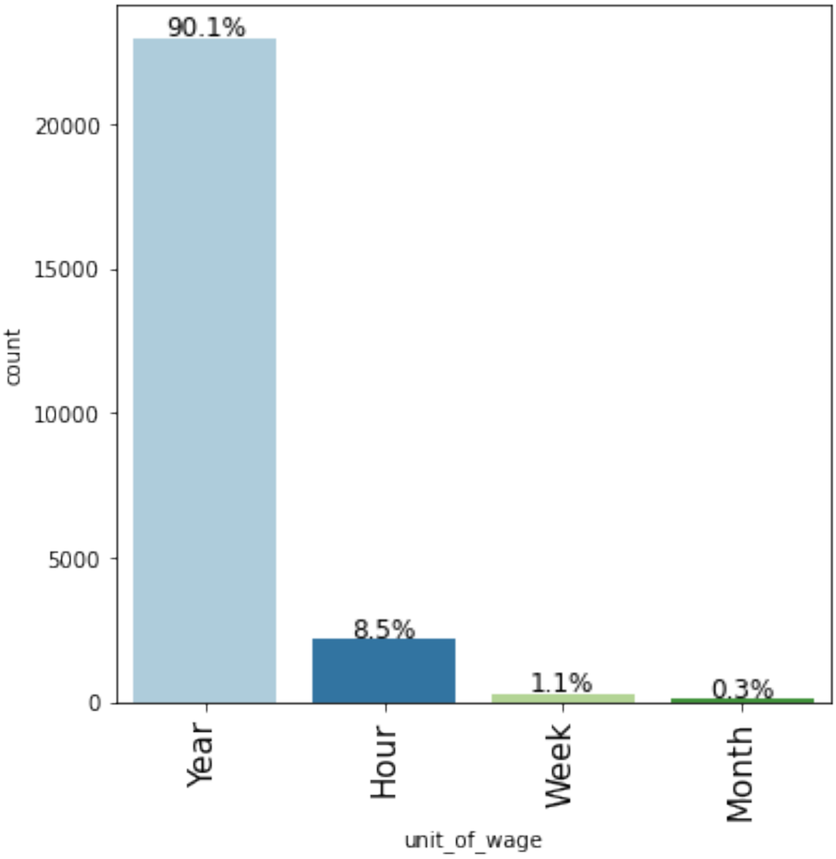
Observations on region of employment

In [25]:
```
labeled_barplot(data, "region_of_employment", perc=True)  ## Complete the
```



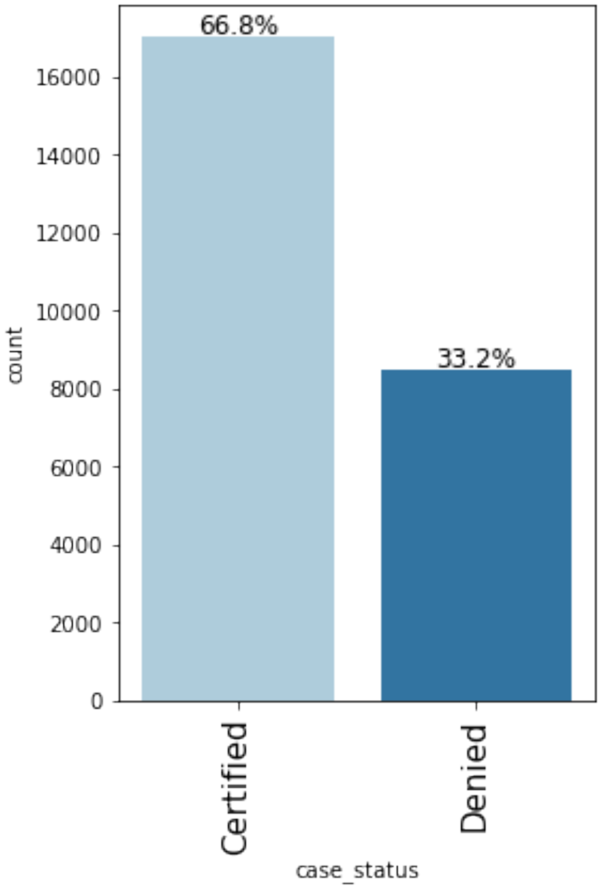
Observations on unit of wage

In [26]:
```
labeled_barplot(data, "unit_of_wage", perc=True)  ## Complete the code to
```



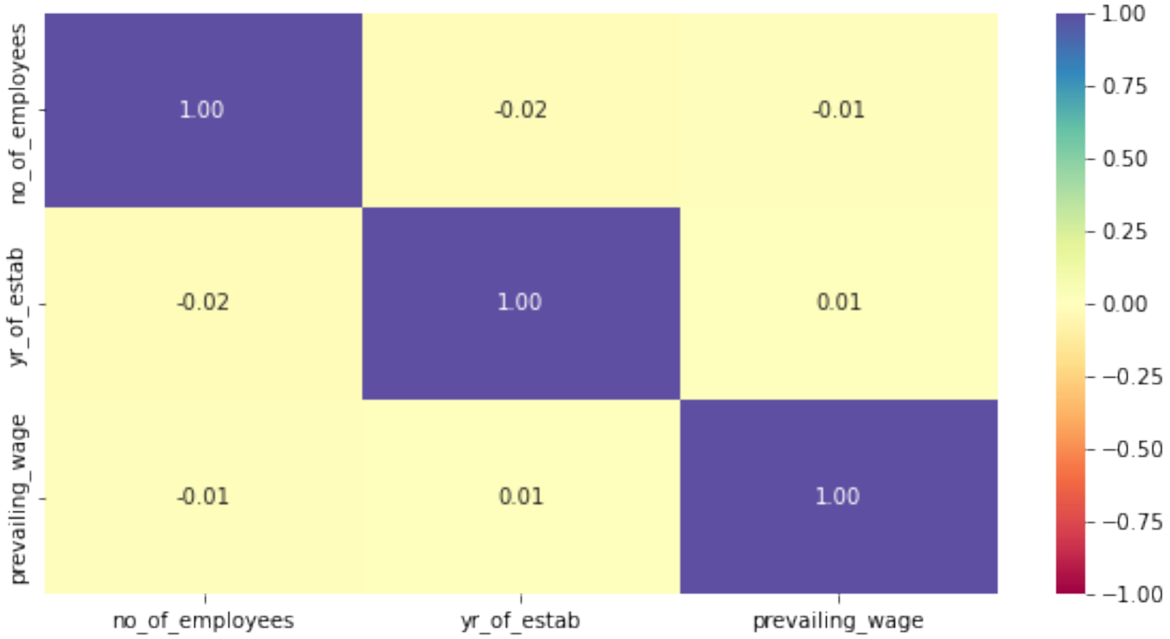
Observations on case status

In [27]:
```
labeled_barplot(data, "case_status", perc=True)  ## Complete the code to
```



Bivariate Analysis

In [28]:
```
cols_list = data.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(10, 5))
sns.heatmap(
    data[cols_list].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap=
) ## Complete the code to find the correlation between the variables
plt.show()
```



Creating functions that will help us with further analysis.

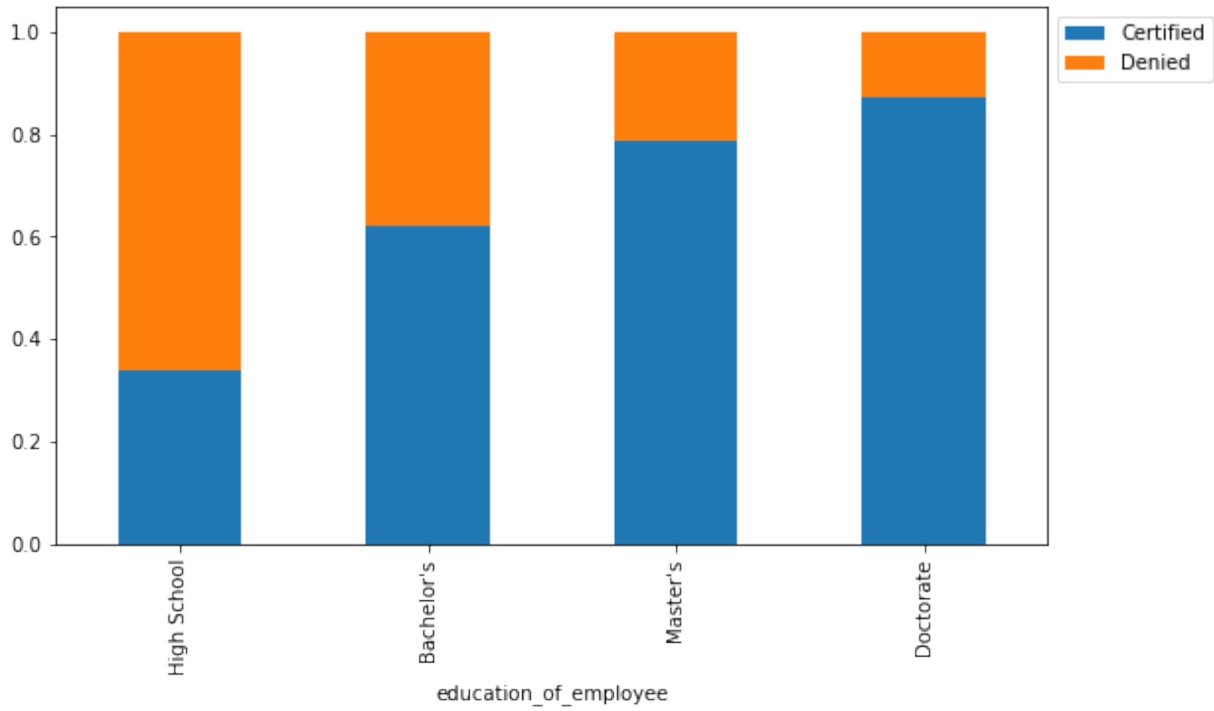
In [29]:
```
### function to plot distributions wrt target


def distribution_plot_wrt_target(data, predictor, target):

    fig, axs = plt.subplots(2, 2, figsize=(12, 10))

    target_uniq = data[target].unique()

    axs[0, 0].set_title("Distribution of target for target=" + str(target_
    sns.histplot(
        data=data[data[target] == target_uniq[0]],
        x=predictor,
        kde=True,
        ax=axs[0, 0],
        color="teal",
        stat="density",
    )

    axs[0, 1].set_title("Distribution of target for target=" + str(target_
    sns.histplot(
        data=data[data[target] == target_uniq[1]],
        x=predictor,
        kde=True,
        ax=axs[0, 1],
        color="orange",
        stat="density",
    )

    axs[1, 0].set_title("Boxplot w.r.t target")
    sns.boxplot(data=data, x=target, y=predictor, ax=axs[1, 0], palette="

    axs[1, 1].set_title("Boxplot (without outliers) w.r.t target")
    sns.boxplot(
        data=data,
        x=target,
        y=predictor,
        ax=axs[1, 1],
        showfliers=False,
        palette="gist_rainbow",
    )

    plt.tight_layout()
    plt.show()
```

In [30]:
```
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_
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").s
        by=sorter, ascending=False
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 5, 5))
    plt.legend(
        loc="lower left", frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

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

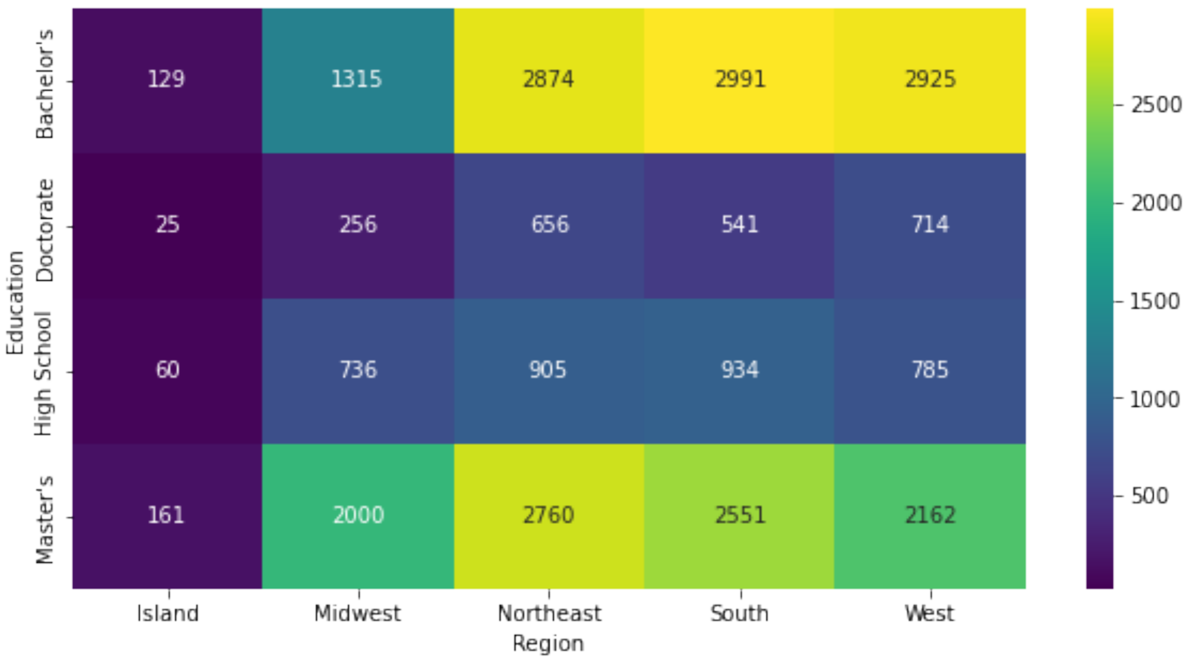
In [31]:
```
stacked_barplot(data, "education_of_employee", "case_status")
```

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



Different regions have different requirements of talent having diverse educational backgrounds. Let's analyze it further

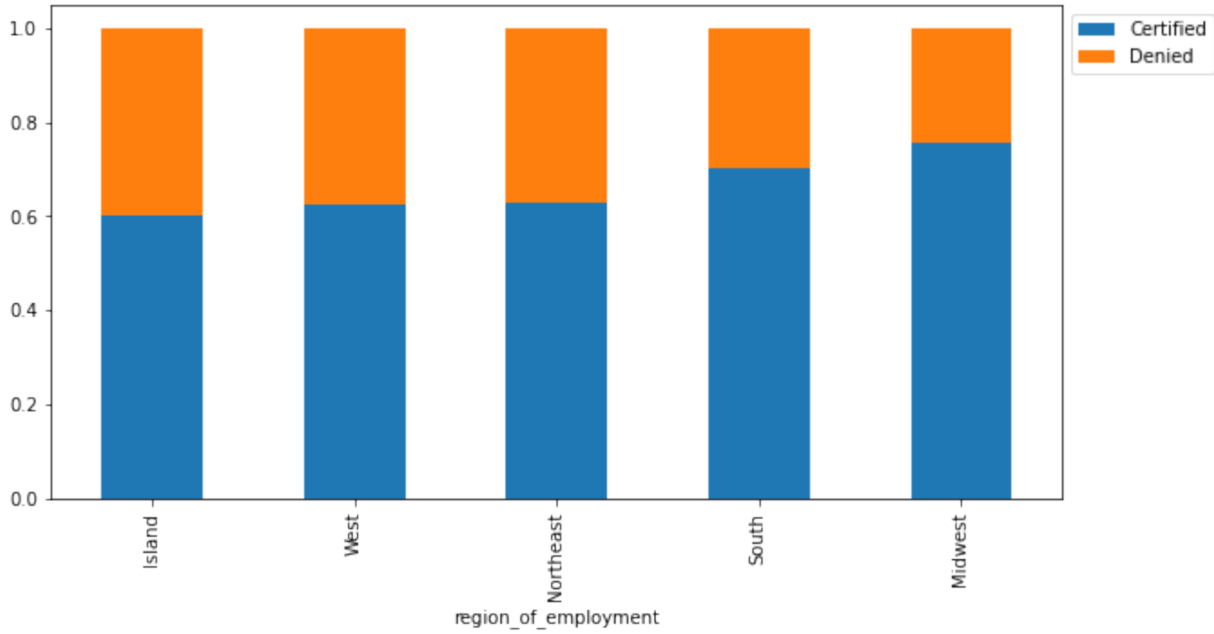
In [32]:
```
plt.figure(figsize=(10, 5))
sns.heatmap(pd.crosstab(data["education_of_employee"], data["region_of_em
    annot=True,
    fmt="g",
    cmap="viridis"
) ## Complete the code to plot heatmap for the crosstab between education

plt.ylabel("Education")
plt.xlabel("Region")
plt.show()
```



Let's have a look at the percentage of visa certifications across each region

In [33]:
```
stacked_barplot(data, "region_of_employment", "case_status") ## Complete
```

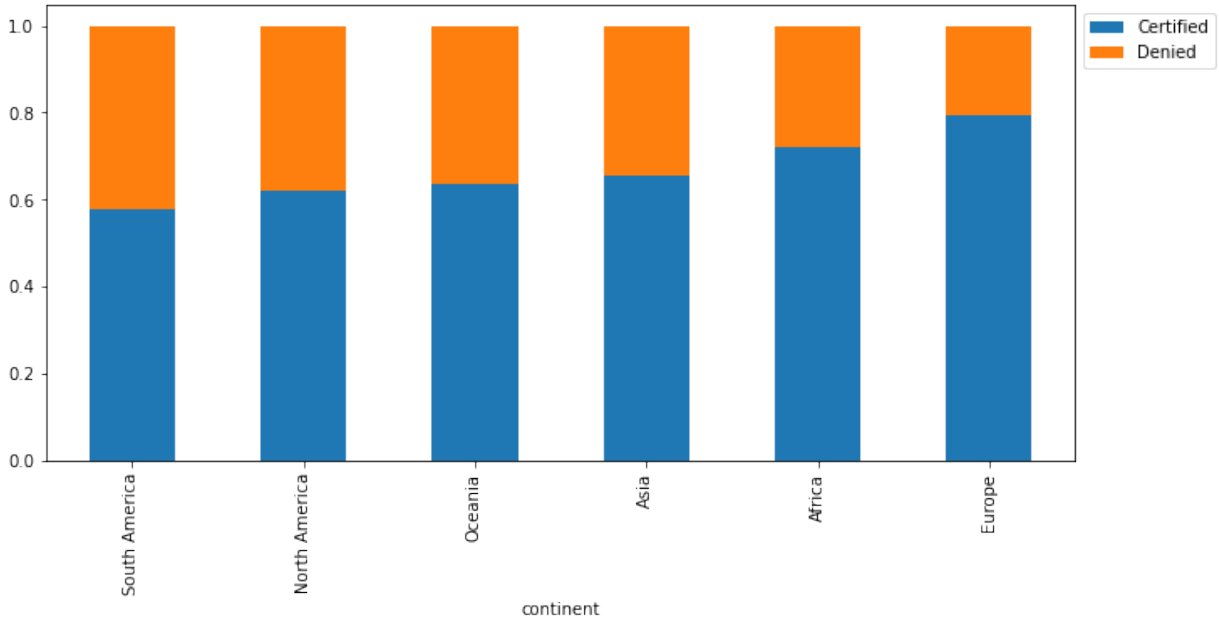
```
case_status           Certified  Denied    All
region_of_employment
All                       17018    8462  25480
Northeast                  4526    2669   7195
West                       4100    2486   6586
South                      4913    2104   7017
Midwest                    3253    1054 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



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

In [34]:
```
stacked_barplot(data,"continent","case_status") ## Complete the code to p
```

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

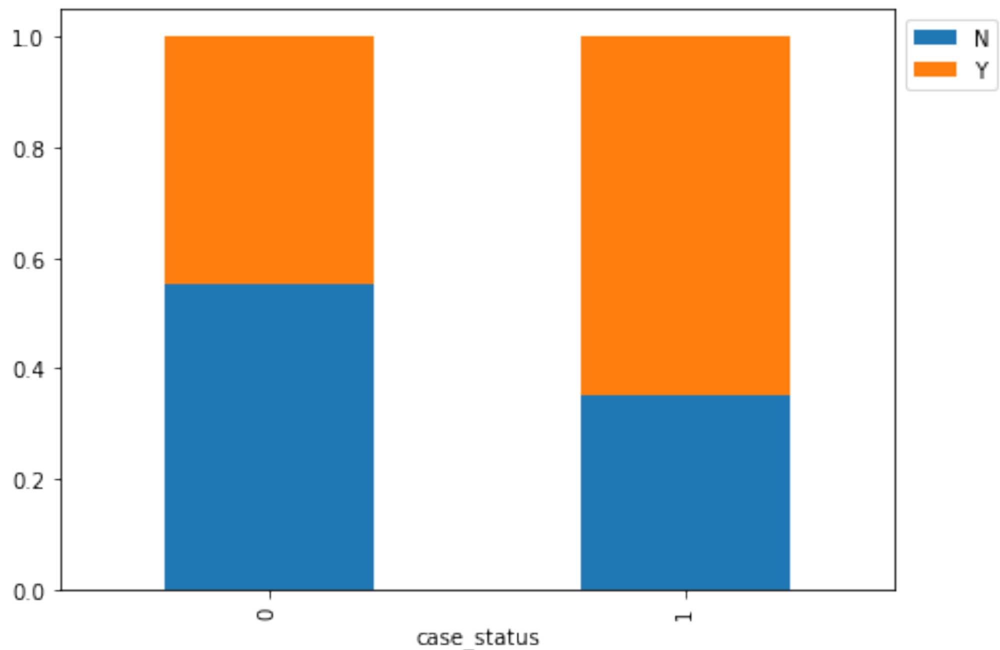


Experienced professionals might look abroad for opportunities to improve their lifestyles and career development. Let's see if having work experience has any influence over visa certification

In [118…

```
stacked_barplot(data,"case_status","has_job_experience") ## Complete the
```

```
has_job_experience      N      Y    All
case_status
All                 10678  14802  25480
1                    5994  11024  17018
0                    4684   3778   8462
------------------------------------------------------------------------
----------------------------------------------
```

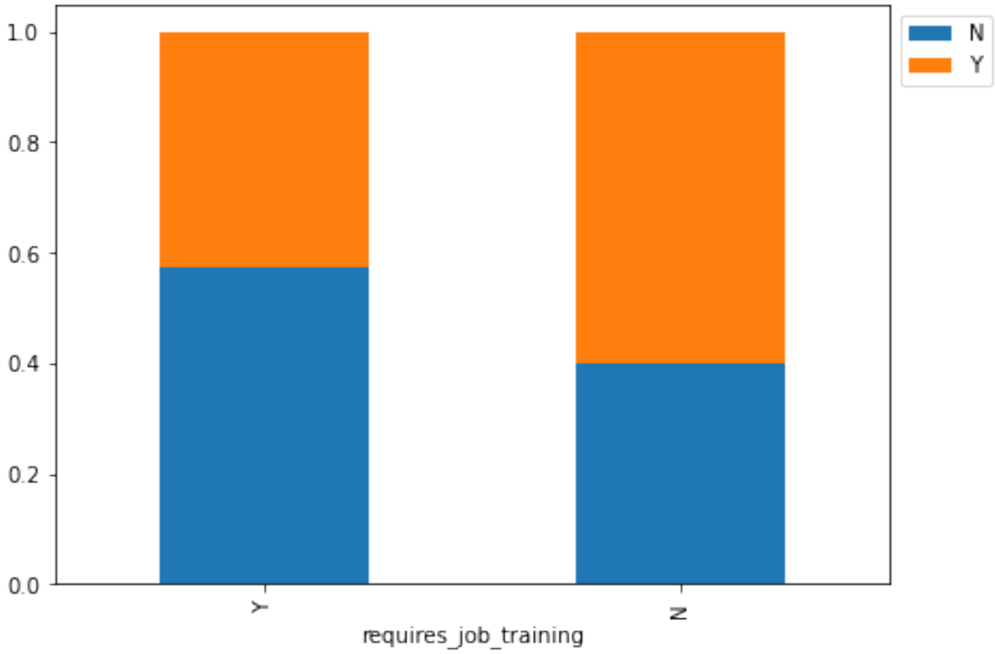


Do the employees who have prior work experience require any job training?

In [117…

```
stacked_barplot(data,"requires_job_training","has_job_experience") ## Com
```

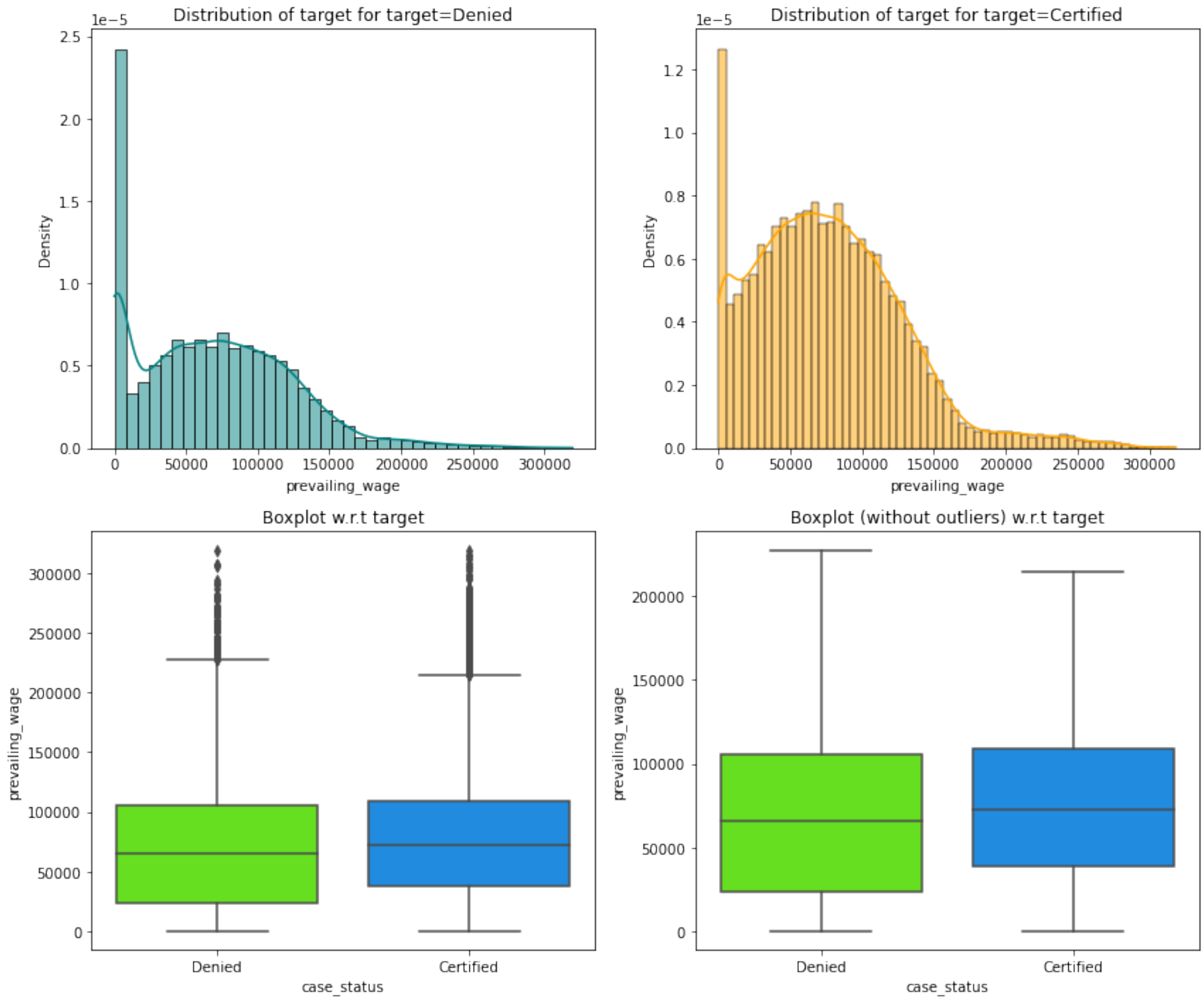
```
has_job_experience          N      Y    All
requires_job_training
All                     10678  14802  25480
N                        8988  13537  22525
Y                        1690   1265   2955
------------------------------------------------------------------------
-----------------------------------------------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



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

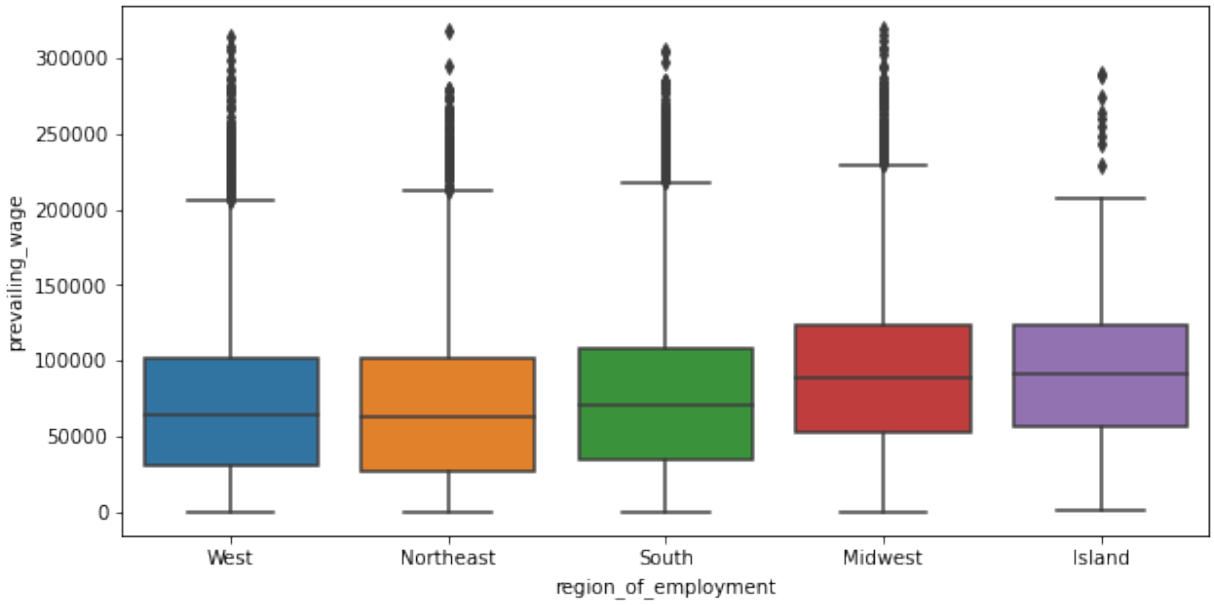
In [37]:

```
distribution_plot_wrt_target(data,'prevailing_wage','case_status') ## Com
```



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

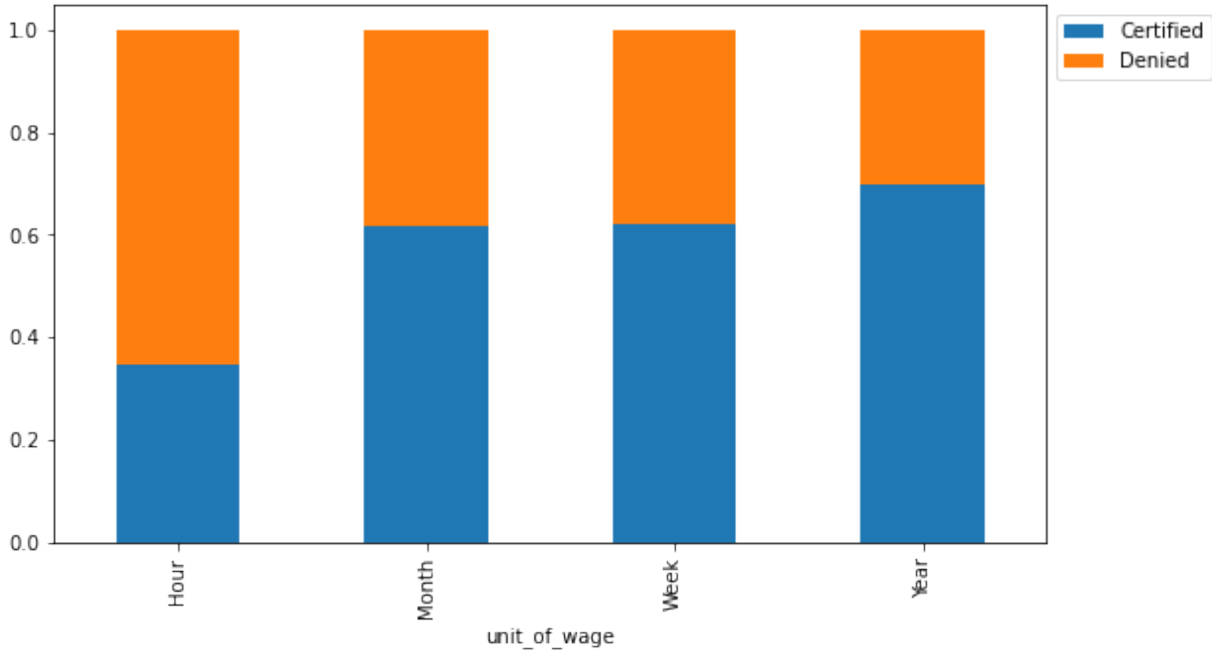
In [38]:
```
plt.figure(figsize=(10, 5))
sns.boxplot(x="region_of_employment",y="prevailing_wage",data=data) ## Co
plt.show()
```



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

In [39]:
```
stacked_barplot(data,"unit_of_wage","case_status") ## Complete the code t
```

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

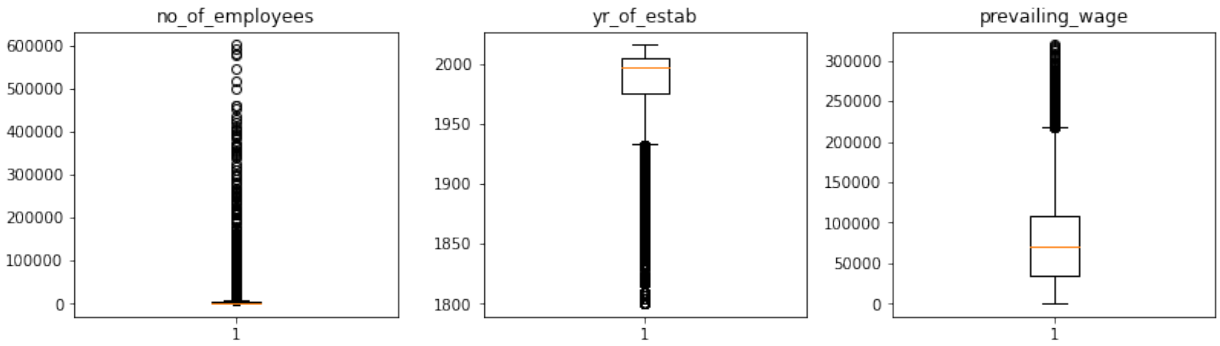


Data Preprocessing

Outlier Check

- Let's check for outliers in the data.

In [40]:
```
# outlier detection using boxplot
numeric_columns = data.select_dtypes(include=np.number).columns.tolist()


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

for i, variable in enumerate(numeric_columns):
    plt.subplot(4, 4, i + 1)
    plt.boxplot(data[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable) ## Complete the code to create boxplots for all t
plt.show()
```



Data Preparation for modeling

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

In [41]:
```
data["case_status"] = data["case_status"].apply(lambda x: 1 if x == "Cert

X = data.drop(["case_status"],axis=1) ## Complete the code to drop case s
Y = data["case_status"]


X = pd.get_dummies(X,drop_first=True)  ## Complete the code to create dum

# Splitting data in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.3, ra
```

In [42]:
```
print("Shape of Training set : ", X_train.shape)
print("Shape of test set : ", X_test.shape)
print("Percentage of classes in training set:")
print(y_train.value_counts(normalize=True))
print("Percentage of classes in test set:")
print(y_test.value_counts(normalize=True))
```

```
Shape of Training set :  (17836, 21)
Shape of test set :  (7644, 21)
Percentage of classes in training set:
1    0.667919
0    0.332081
Name: case_status, dtype: float64
Percentage of classes in test set:
1    0.667844
0    0.332156
Name: case_status, dtype: float64
```

Model evaluation criterion

Model can make wrong predictions as:

1. Model predicts that the visa application will get certified but in reality, the visa application should get denied.
2. Model predicts that the visa application will not get certified but in reality, the visa application should get certified.

Which case is more important?

- Both the cases are important as:

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

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

How to reduce the losses?

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

First, let's create functions to calculate different metrics and confusion matrix so that we don't have to use the same code repeatedly for each model.

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

In [43]:
```
# defining a function to compute different metrics to check performance o


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

    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":
        index=[0],
    )

    return df_perf
```

In [44]:
```
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.flatt
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

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

Decision Tree - Model Building and Hyperparameter Tuning

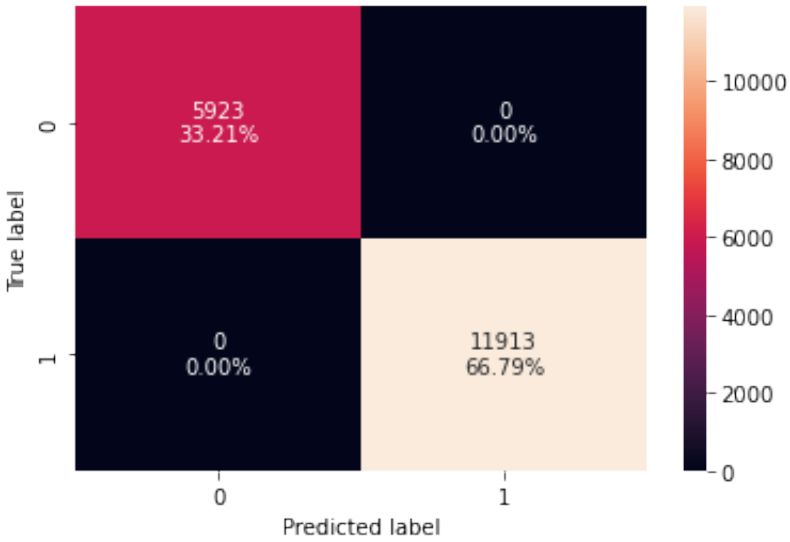
Decision Tree Model

In [45]:
```
model = DecisionTreeClassifier(random_state=1) ## Complete the code to de
model.fit(X_train,y_train) ## Complete the code to fit decision tree clas
```

Out[45]:
```
DecisionTreeClassifier(random_state=1)
```

Checking model performance on training set

In [46]:
```
confusion_matrix_sklearn(model,X_train,y_train) ## Complete the code to c
```



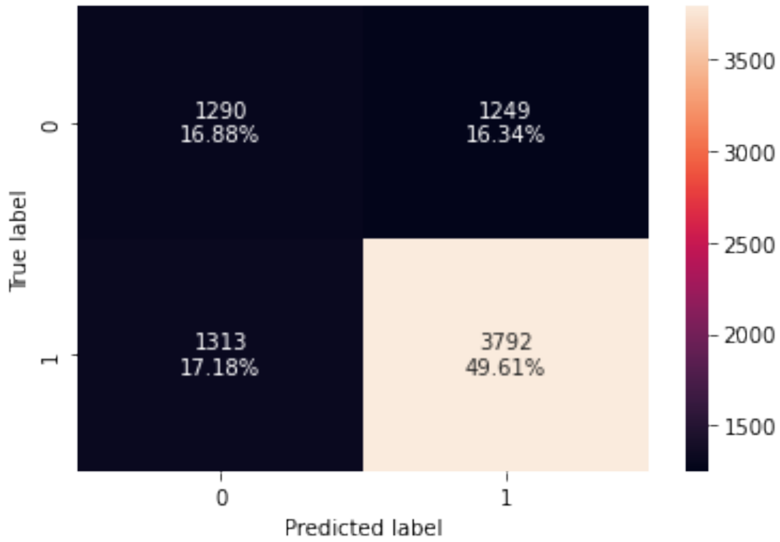
In [47]:
```
decision_tree_perf_train = model_performance_classification_sklearn(model
decision_tree_perf_train
```

Out[47]:

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Checking model performance on test set

In [48]:
```
confusion_matrix_sklearn(model,X_test,y_test) ## Complete the code to cre
```



In [49]:
```
decision_tree_perf_test = model_performance_classification_sklearn(model,
decision_tree_perf_test
```

Out[49]:

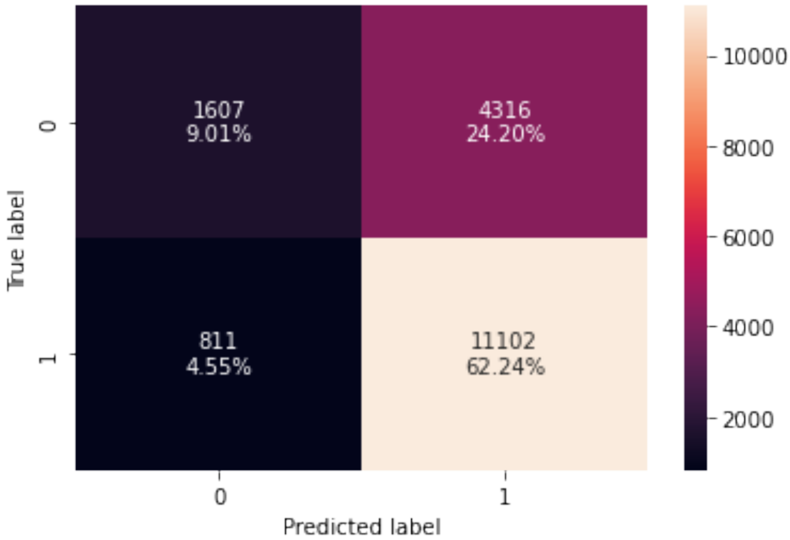
	Accuracy	Recall	Precision	F1
0	0.664835	0.742801	0.752232	0.747487

Hyperparameter Tuning - Decision Tree

In [50]:
```
# Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(class_weight="balanced", random_

# Grid of parameters to choose from
parameters = {
    "max_depth": np.arange(5, 16, 5),
    "min_samples_leaf": [3, 5, 7],
    "max_leaf_nodes": [2, 5],
    "min_impurity_decrease": [0.0001, 0.001],
}

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

# Run the grid search
grid_obj = GridSearchCV(dtree_estimator,parameters,scoring=scorer,n_jobs=

grid_obj = grid_obj.fit(X_train,y_train) ## Complete the code to fit the

# 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[50]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=5, max_leaf_nod
es=2,
                       min_impurity_decrease=0.0001, min_samples_leaf=3,
                       random_state=1)
```

In [51]:
```
confusion_matrix_sklearn(dtree_estimator, X_train, y_train) ## Complete t
```

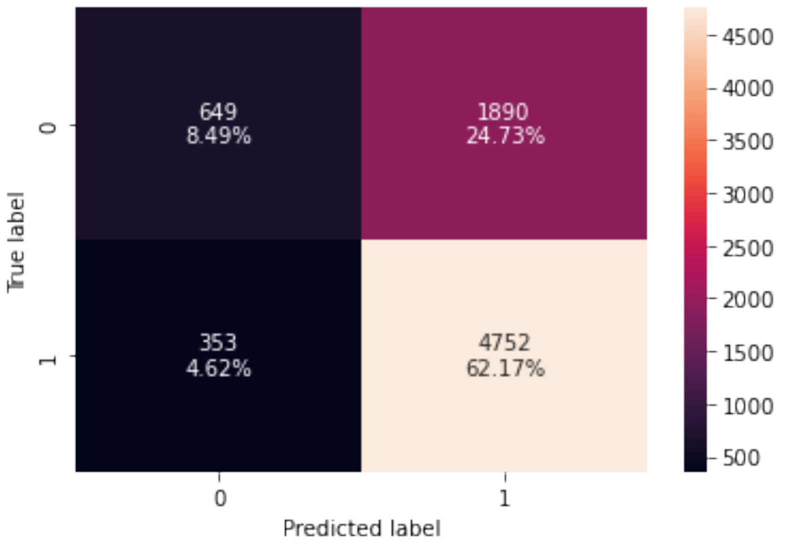


In [52]:
```
dtree_estimator_model_train_perf = model_performance_classification_sklea
dtree_estimator_model_train_perf
```

Out[52]:

	Accuracy	Recall	Precision	F1
0	0.712548	0.931923	0.720067	0.812411

In [53]:
```
confusion_matrix_sklearn(dtree_estimator, X_test, y_test) ## Complete the
```



In [54]:
```
dtree_estimator_model_test_perf = model_performance_classification_sklear
dtree_estimator_model_test_perf
```

Out[54]:

	Accuracy	Recall	Precision	F1
0	0.706567	0.930852	0.715447	0.809058

Bagging - Model Building and Hyperparameter Tuning

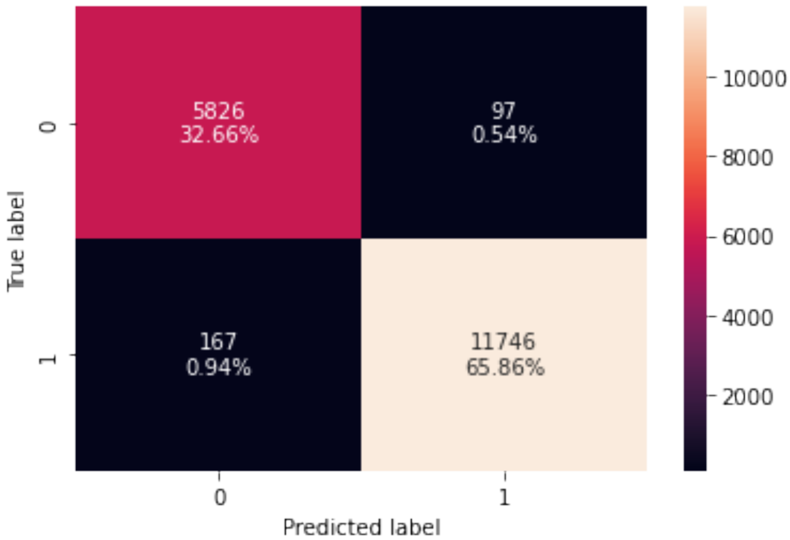
Bagging Classifier

In [55]:
```
bagging_classifier = BaggingClassifier(DecisionTreeClassifier(random_stat
bagging_classifier.fit(X_train,y_train) ## Complete the code to fit baggi
```

Out[55]:
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=1),
                  random_state=1)
```

Checking model performance on training set

In [56]:
```
confusion_matrix_sklearn(bagging_classifier, X_train, y_train) ## Complet
```



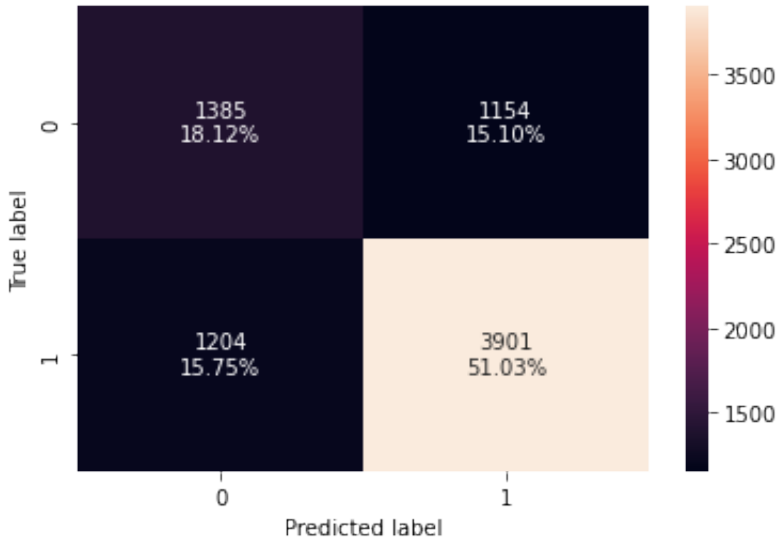
In [57]:
```
bagging_classifier_model_train_perf = model_performance_classification_sk
bagging_classifier_model_train_perf
```

Out[57]:

	Accuracy	Recall	Precision	F1
0	0.985198	0.985982	0.99181	0.988887

Checking model performance on test set

In [58]:
```
confusion_matrix_sklearn(bagging_classifier, X_test, y_test) ## Complete
```



In [59]:
```
bagging_classifier_model_test_perf = model_performance_classification_skl
bagging_classifier_model_test_perf
```

Out[59]:

	Accuracy	Recall	Precision	F1
0	0.691523	0.764153	0.771711	0.767913

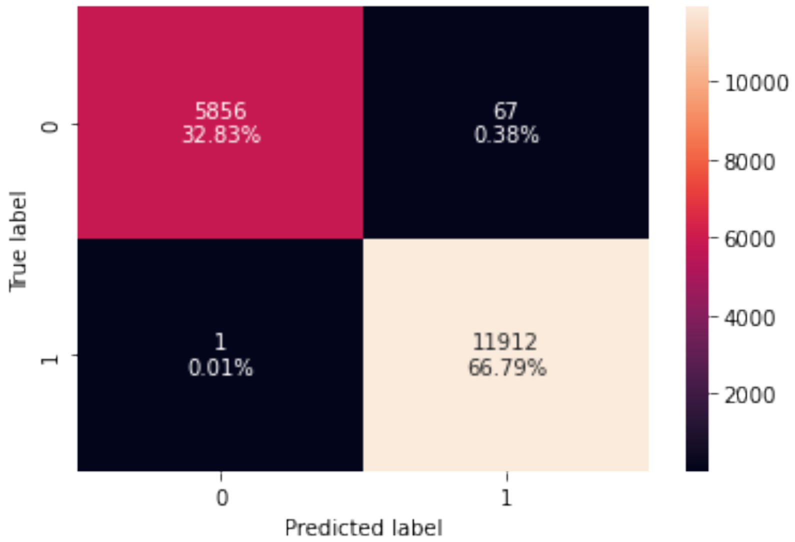
Hyperparameter Tuning - Bagging Classifier

In [60]:
```
# Choose the type of classifier.
bagging_estimator_tuned = BaggingClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    "max_samples": [0.7, 0.9],
    "max_features": [0.7, 0.9],
    "n_estimators": np.arange(90, 111, 10),
}

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

# Run the grid search
grid_obj = GridSearchCV(bagging_estimator_tuned , parameters ,scoring=acc_
grid_obj = grid_obj.fit(X_train,y_train) ## Complete the code to fit the

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

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

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

Checking model performance on training set

In [61]:
```
confusion_matrix_sklearn(bagging_estimator_tuned, X_train, y_train) ## Co
```



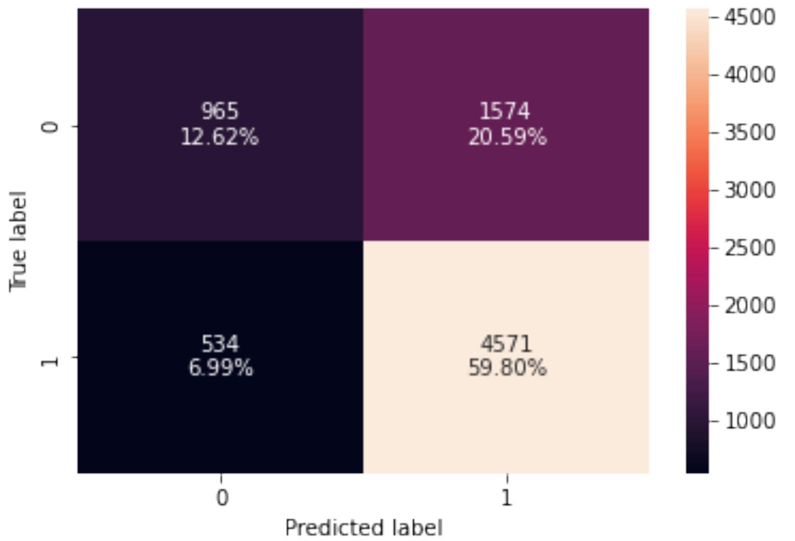
In [62]:
```
bagging_estimator_tuned_model_train_perf = model_performance_classificati
bagging_estimator_tuned_model_train_perf
```

Out[62]:

	Accuracy	Recall	Precision	F1
0	0.996187	0.999916	0.994407	0.997154

Checking model performance on test set

In [63]:
```
confusion_matrix_sklearn(bagging_estimator_tuned, X_test, y_test) ## Comp
```



In [64]:
```
bagging_estimator_tuned_model_test_perf = model_performance_classificatio
bagging_estimator_tuned_model_test_perf
```

Out[64]:

	Accuracy	Recall	Precision	F1
0	0.724228	0.895397	0.743857	0.812622

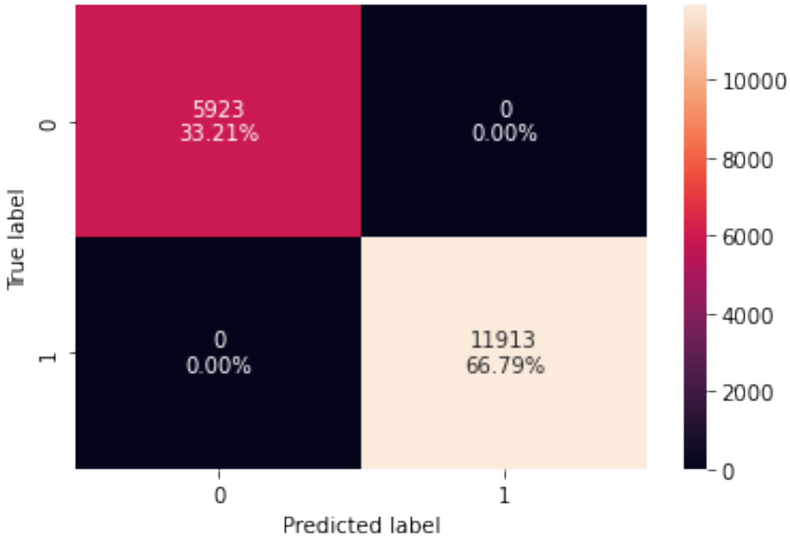
Random Forest

In [65]:
```
# Fitting the model
rf_estimator = RandomForestClassifier(random_state=1, class_weight='balan
rf_estimator.fit(X_train, y_train) ## Complete the code to fit random for
```

Out[65]:
```
RandomForestClassifier(class_weight='balanced', random_state=1)
```

Checking model performance on training set

In [66]:
```
confusion_matrix_sklearn(rf_estimator, X_train, y_train) ## Complete the
```



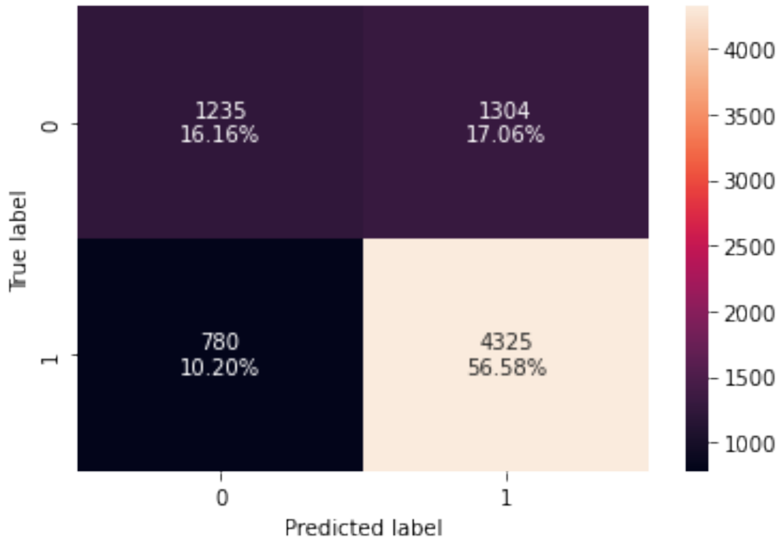
In [67]:
```
# Calculating different metrics
rf_estimator_model_train_perf = model_performance_classification_sklearn(
rf_estimator_model_train_perf
```

Out[67]:

	Accuracy	Recall	Precision	F1
0	1.0	1.0	1.0	1.0

Checking model performance on test set

In [68]:
```
confusion_matrix_sklearn(rf_estimator, X_test, y_test) ## Complete the co
```



In [69]:
```
rf_estimator_model_test_perf = model_performance_classification_sklearn(r
rf_estimator_model_test_perf
```

Out[69]:

	Accuracy	Recall	Precision	F1
0	0.727368	0.847209	0.768343	0.805851

Hyperparameter Tuning - Random Forest

In [70]:
```
# Choose the type of classifier.
rf_tuned = RandomForestClassifier(random_state=1, oob_score=True, bootstr

parameters = {
    "max_depth": list(np.arange(5, 15, 5)),
    "max_features": ["sqrt", "log2"],
    "min_samples_split": [5, 7],
    "n_estimators": np.arange(15, 26, 5),
}

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

# Run the grid search
grid_obj = GridSearchCV(estimator=rf_tuned, param_grid=parameters, scorin
grid_obj = grid_obj.fit(X_train, y_train) ## Complete the code to fit the

# 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)
```

```
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
```

```
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
```

```
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
```

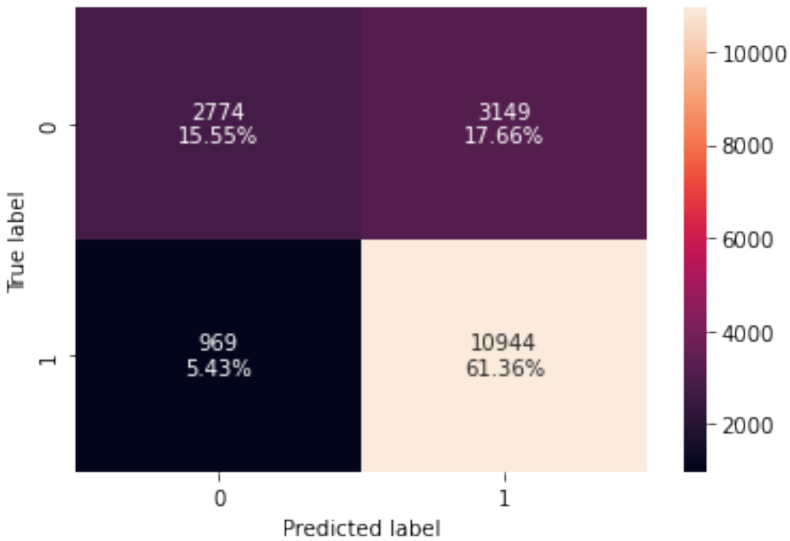
```
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
```

```
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
/Users/Moafdhal/opt/anaconda3/lib/python3.9/site-packages/sklearn/ensembl
e/_forest.py:560: UserWarning: Some inputs do not have OOB scores. This p
robably means too few trees were used to compute any reliable OOB estimat
es.
  warn(
```

Out[70]:
```
RandomForestClassifier(max_depth=10, max_features='sqrt', min_samples_spl
it=7,
                       n_estimators=20, oob_score=True, random_state=1)
```

Checking model performance on training set

In [71]:
```
confusion_matrix_sklearn(rf_tuned, X_train, y_train) ## Complete the code
```



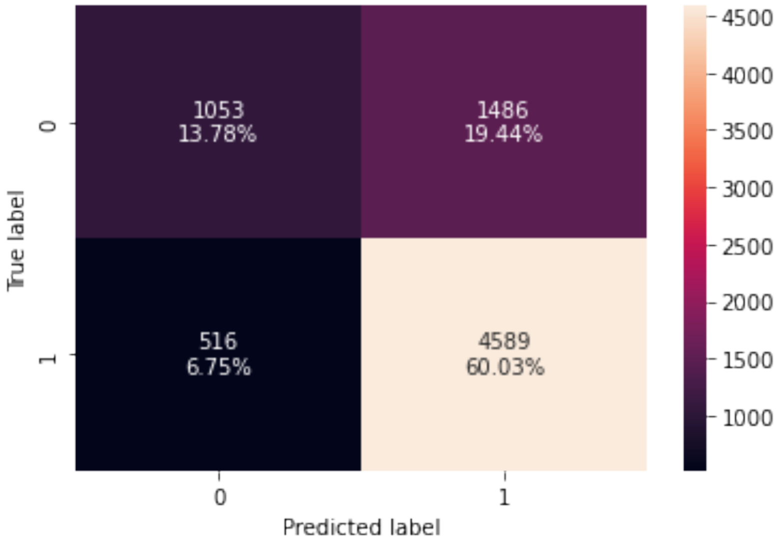
In [72]:
```
rf_tuned_model_train_perf = model_performance_classification_sklearn(rf_t
rf_tuned_model_train_perf
```

Out[72]:

	Accuracy	Recall	Precision	F1
0	0.769119	0.91866	0.776556	0.841652

Checking model performance on test set

In [73]:
```
confusion_matrix_sklearn(rf_tuned, X_test, y_test) ## Complete the code t
```



In [74]:
```
rf_tuned_model_test_perf = model_performance_classification_sklearn(rf_tu
rf_tuned_model_test_perf
```

Out[74]:

	Accuracy	Recall	Precision	F1
0	0.738095	0.898923	0.755391	0.82093

Boosting - Model Building and Hyperparameter Tuning

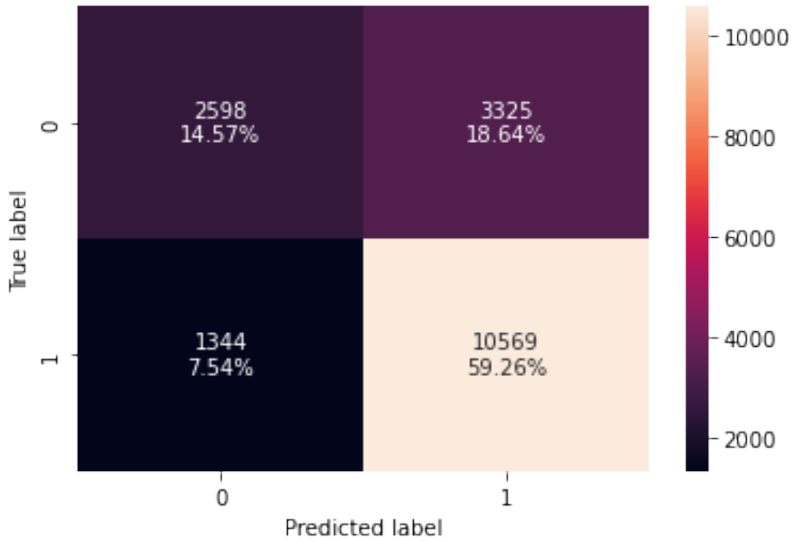
AdaBoost Classifier

In [75]:
```
ab_classifier = AdaBoostClassifier(random_state=1)
ab_classifier.fit(X_train, y_train)
```

Out[75]:
```
AdaBoostClassifier(random_state=1)
```

Checking model performance on training set

In [76]:
```
confusion_matrix_sklearn(ab_classifier, X_train, y_train) ## Complete the
```



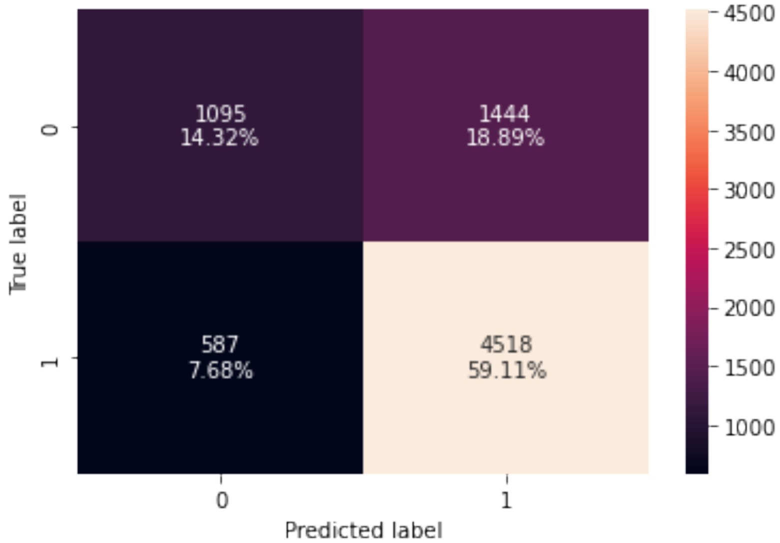
In [77]:
```
ab_classifier_model_train_perf = model_performance_classification_sklearn
ab_classifier_model_train_perf
```

Out[77]:

	Accuracy	Recall	Precision	F1
0	0.738226	0.887182	0.760688	0.81908

Checking model performance on test set

In [78]:
```
confusion_matrix_sklearn(ab_classifier, X_test, y_test) ## Complete the c
```



In [79]:
```
ab_classifier_model_test_perf = model_performance_classification_sklearn(
ab_classifier_model_test_perf
```

Out[79]:

	Accuracy	Recall	Precision	F1
0	0.738226	0.887182	0.760688	0.81908

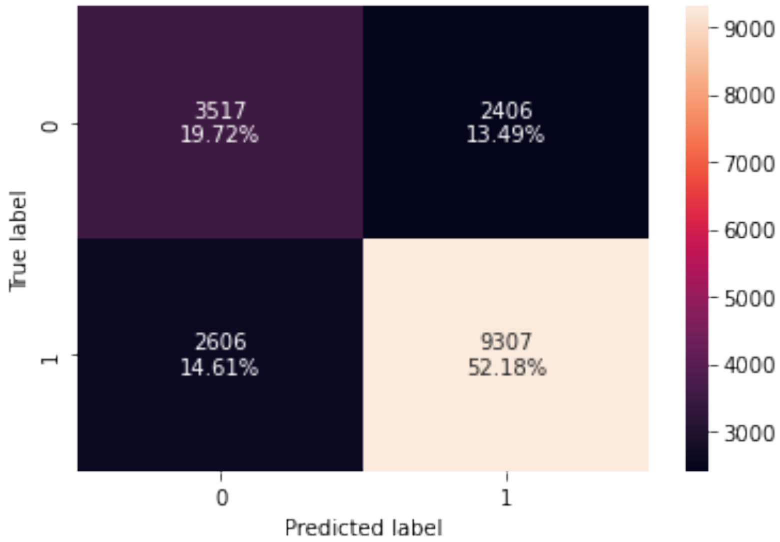
Hyperparameter Tuning - AdaBoost Classifier

In [80]:
```
# Choose the type of classifier.
abc_tuned = AdaBoostClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    # Let's try different max_depth for base_estimator
    "base_estimator": [
        DecisionTreeClassifier(max_depth=1, class_weight="balanced", rand
        DecisionTreeClassifier(max_depth=2, class_weight="balanced", rand
    ],
    "n_estimators": np.arange(80, 101, 10),
    "learning_rate": np.arange(0.1, 0.4, 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, cv=5) ## Complete the code
grid_obj = grid_obj.fit(X_train, y_train) ## Complete the code to fit the

# 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[80]:
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(class_weight='ba
lanced',
                                                         max_depth=1,
                                                         random_state=1),
                   learning_rate=0.1, n_estimators=100, random_state=1)
```

Checking model performance on training set

In [81]:
```
confusion_matrix_sklearn(abc_tuned, X_train, y_train) ## Complete the cod
```



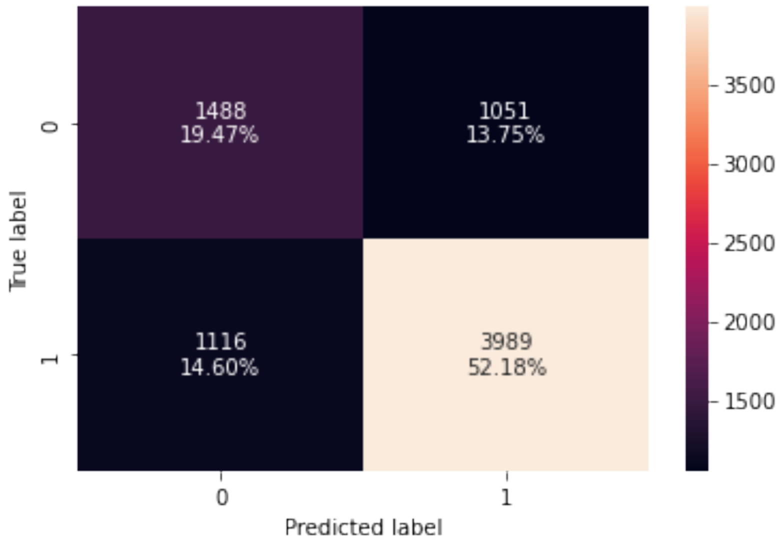
In [82]:
```
abc_tuned_model_train_perf = model_performance_classification_sklearn(abc_
abc_tuned_model_train_perf
```

Out[82]:

	Accuracy	Recall	Precision	F1
0	0.718995	0.781247	0.794587	0.787861

Checking model performance on test set

In [83]:
```
confusion_matrix_sklearn(abc_tuned, X_test, y_test) ## Complete the code
```



In [84]:
```
abc_tuned_model_test_perf = model_performance_classification_sklearn(abc_
abc_tuned_model_test_perf
```

Out[84]:

	Accuracy	Recall	Precision	F1
0	0.71651	0.781391	0.791468	0.786397

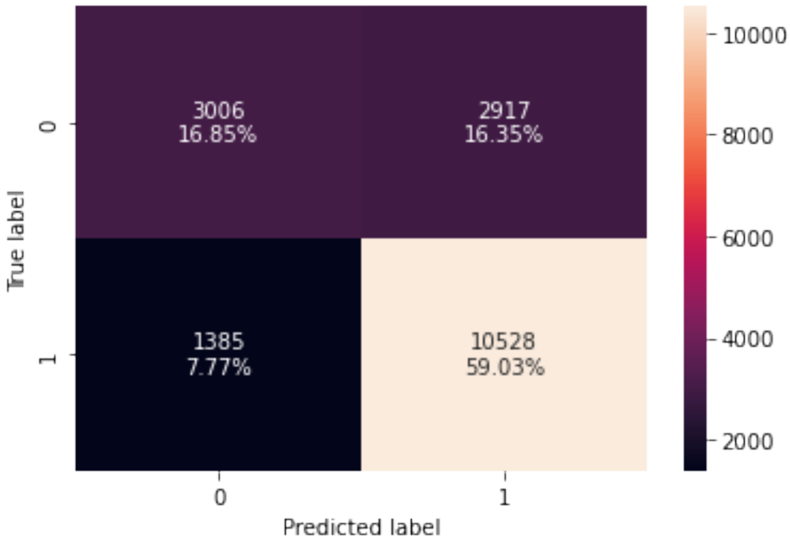
Gradient Boosting Classifier

In [85]:
```
gb_classifier = GradientBoostingClassifier(random_state=1)
gb_classifier.fit(X_train, y_train)
```

Out[85]:
```
GradientBoostingClassifier(random_state=1)
```

Checking model performance on training set

In [86]:
```
confusion_matrix_sklearn(gb_classifier, X_train, y_train) ## Complete the
```



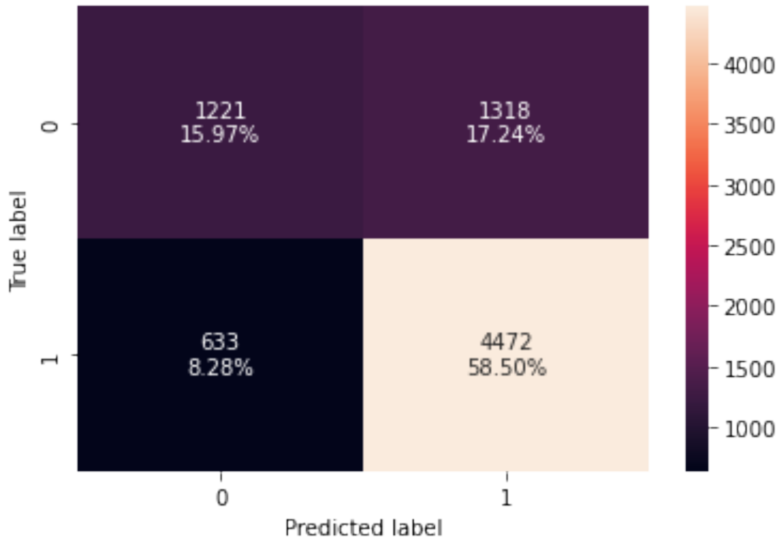
In [87]:
```
gb_classifier_model_train_perf = model_performance_classification_sklearn
gb_classifier_model_train_perf
```

Out[87]:

	Accuracy	Recall	Precision	F1
0	0.758802	0.88374	0.783042	0.830349

Checking model performance on test set

In [88]:
```
confusion_matrix_sklearn(gb_classifier, X_test, y_test) ## Complete the c
```



In [89]:
```
gb_classifier_model_test_perf = model_performance_classification_sklearn(
gb_classifier_model_test_perf
```

Out[89]:

	Accuracy	Recall	Precision	F1
0	0.744767	0.876004	0.772366	0.820927

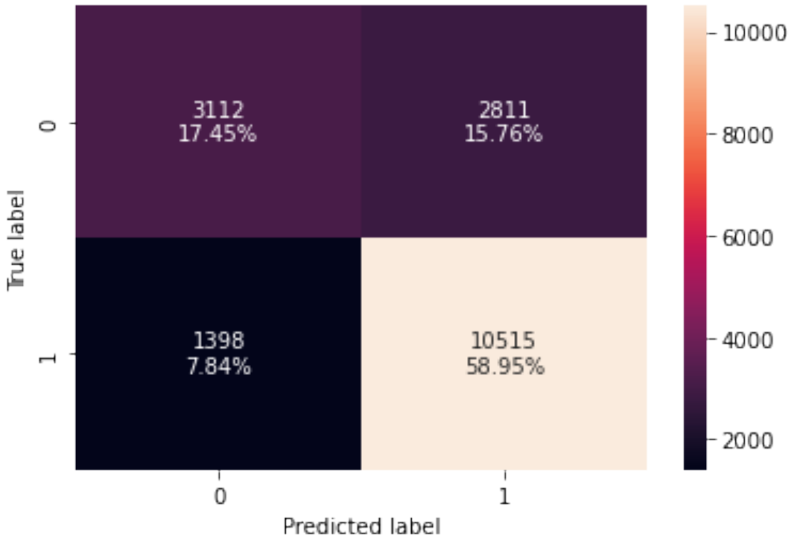
Hyperparameter Tuning - Gradient Boosting Classifier

In [90]:
```
# Choose the type of classifier.
gbc_tuned = GradientBoostingClassifier(
    init=AdaBoostClassifier(random_state=1), random_state=1
)

# Grid of parameters to choose from
parameters = {
    "n_estimators": [200, 250],
    "subsample": [0.9, 1],
    "max_features": [0.8, 0.9],
    "learning_rate": np.arange(0.1, 0.21, 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(gbc_tuned, parameters, scoring=acc_scorer, cv=5)
grid_obj = grid_obj.fit(X_train, y_train)
 ## Complete the code to fit the grid_obj on train data

# 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[90]:
```
GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1),
                           max_features=0.8, n_estimators=200, random_sta
te=1,
                           subsample=1)
```

Checking model performance on training set

In [91]:
```
confusion_matrix_sklearn(gbc_tuned, X_train, y_train) ## Complete the cod
```



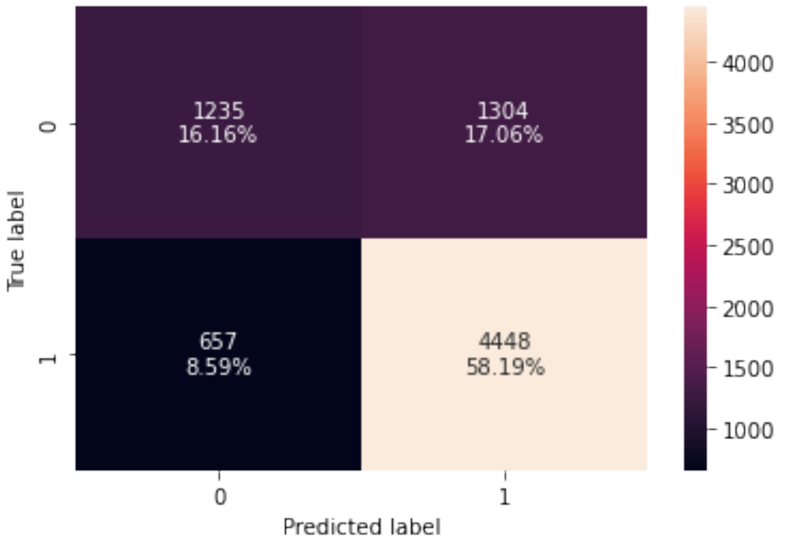
In [92]:
```
gbc_tuned_model_train_perf = model_performance_classification_sklearn(gbc_
gbc_tuned_model_train_perf
```

Out[92]:

	Accuracy	Recall	Precision	F1
0	0.764017	0.882649	0.789059	0.833234

Checking model performance on test set

In [93]:
```
confusion_matrix_sklearn(gbc_tuned, X_test, y_test)## Complete the code t
```



In [94]:
```
gbc_tuned_model_test_perf = model_performance_classification_sklearn(gbc_
gbc_tuned_model_test_perf
```

Out[94]:

	Accuracy	Recall	Precision	F1
0	0.743459	0.871303	0.773296	0.819379

Note - You can choose **not to build** XGBoost if you have any installation issues

In [95]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in /Users/Moafdhal/opt/anaconda3/l
ib/python3.9/site-packages (1.7.5)
Requirement already satisfied: numpy in /Users/Moafdhal/opt/anaconda3/lib
/python3.9/site-packages (from xgboost) (1.21.5)
Requirement already satisfied: scipy in /Users/Moafdhal/opt/anaconda3/lib
/python3.9/site-packages (from xgboost) (1.7.3)
```

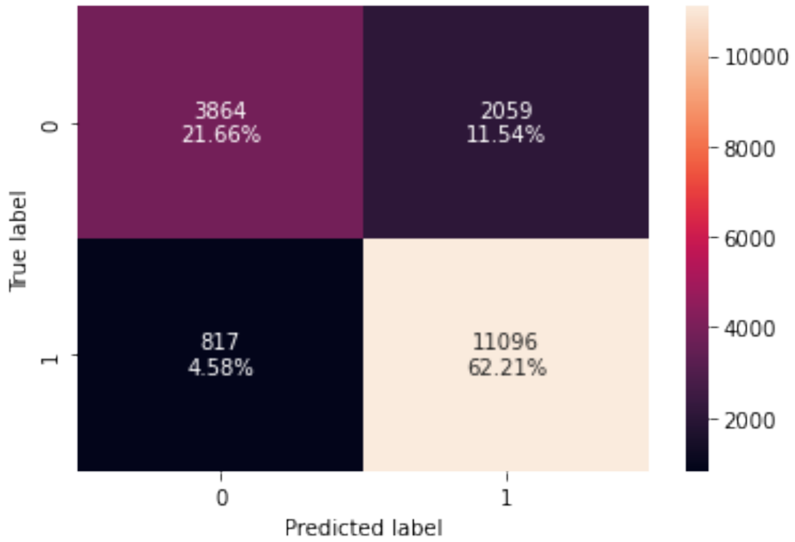
XGBoost Classifier

In [102…
```
xgb_classifier = XGBClassifier(random_state=1, eval_metric="logloss")
xgb_classifier.fit(X_train, y_train)
```

Out[102]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, gpu_id=None, grow_policy=N
one,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, 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, n_estimators=100, n_jobs=None,
              num_parallel_tree=None, predictor=None, random_state=1, ..
.)
```

Checking model performance on training set

In [103…
```
confusion_matrix_sklearn(xgb_classifier, X_train, y_train) ## Complete th
```



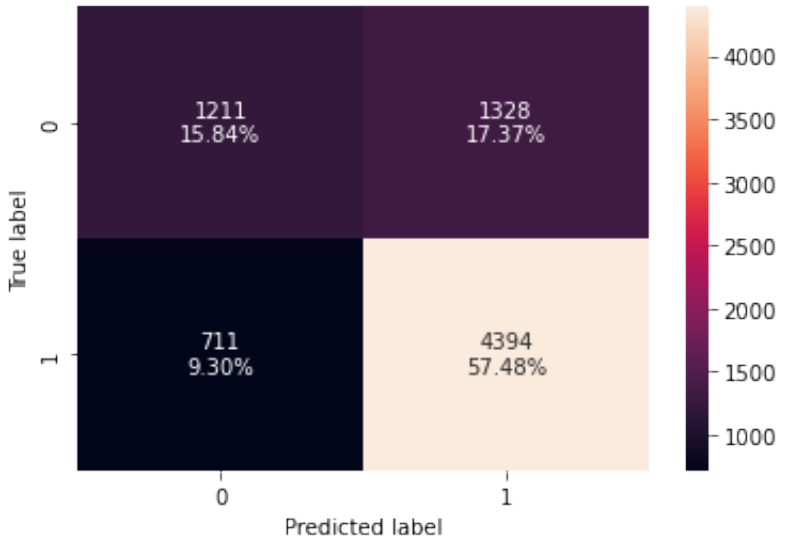
In [104…
```
xgb_classifier_model_train_perf = model_performance_classification_sklear
xgb_classifier_model_train_perf
```

Out[104]:

	Accuracy	Recall	Precision	F1
0	0.838753	0.931419	0.843482	0.885272

Checking model performance on test set

In [105…
```
confusion_matrix_sklearn(xgb_classifier, X_test, y_test) ## Complete the
```



In [106…
```
xgb_classifier_model_test_perf = model_performance_classification_sklearn
xgb_classifier_model_test_perf
```

Out[106]:

	Accuracy	Recall	Precision	F1
0	0.733255	0.860725	0.767913	0.811675

Hyperparameter Tuning - XGBoost Classifier

In [107…

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

# Grid of parameters to choose from
parameters = {
    "n_estimators": np.arange(150, 250, 50),
    "scale_pos_weight": [1, 2],
    "subsample": [0.9, 1],
    "learning_rate": np.arange(0.1, 0.21, 0.1),
    "gamma": [3, 5],
    "colsample_bytree": [0.8, 0.9],
    "colsample_bylevel": [ 0.9, 1],
}

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

# Run the grid search
grid_obj = GridSearchCV(xgb_tuned, parameters, scoring=acc_scorer, cv=5,
grid_obj = grid_obj.fit(X_train, y_train) ## Complete the code to fit the

# 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)
```

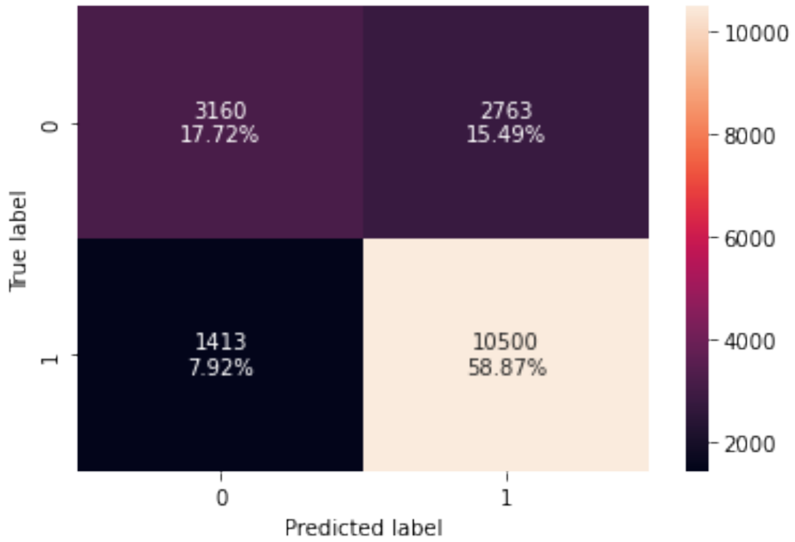
Out[107]:

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=1, colsample_bynode=None, colsample_bytr
ee=0.9,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric='logloss', feature_types=None, gamma=5, gpu_id
=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=N
one,
              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=N
one,
              n_estimators=150, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=1, ...)
```

Checking model performance on training set

In [97]:

```
confusion_matrix_sklearn(xgb_tuned, X_train, y_train) ## Complete the cod
```



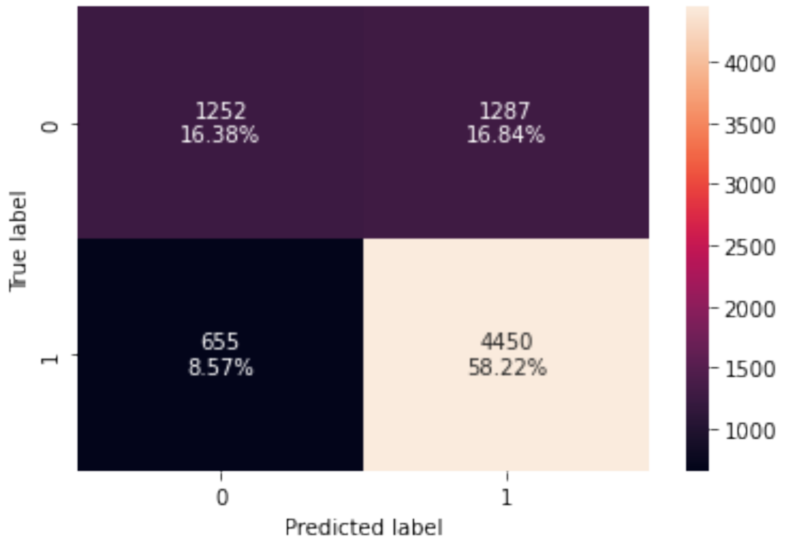
In [98]:
```
xgb_tuned_model_train_perf = model_performance_classification_sklearn(xgb_
xgb_tuned_model_train_perf
```

Out[98]:

	Accuracy	Recall	Precision	F1
0	0.765867	0.88139	0.791676	0.834128

Checking model performance on test set

In [99]:
```
confusion_matrix_sklearn(xgb_tuned, X_test, y_test) ## Complete the code
```



In [100…
```
xgb_tuned_model_test_perf = model_performance_classification_sklearn(xgb_
xgb_tuned_model_test_perf
```

Out[100]:

	Accuracy	Recall	Precision	F1
0	0.745945	0.871694	0.775667	0.820882

Stacking Classifier

In [108…

```
estimators = [
    ("AdaBoost", ab_classifier),
    ("Gradient Boosting", gbc_tuned),
    ("Random Forest", rf_tuned),
]

final_estimator = xgb_tuned
stacking_classifier = StackingClassifier(estimators=estimators, final_est

stacking_classifier.fit(X_train, y_train)## Complete the code to fit Stac
```

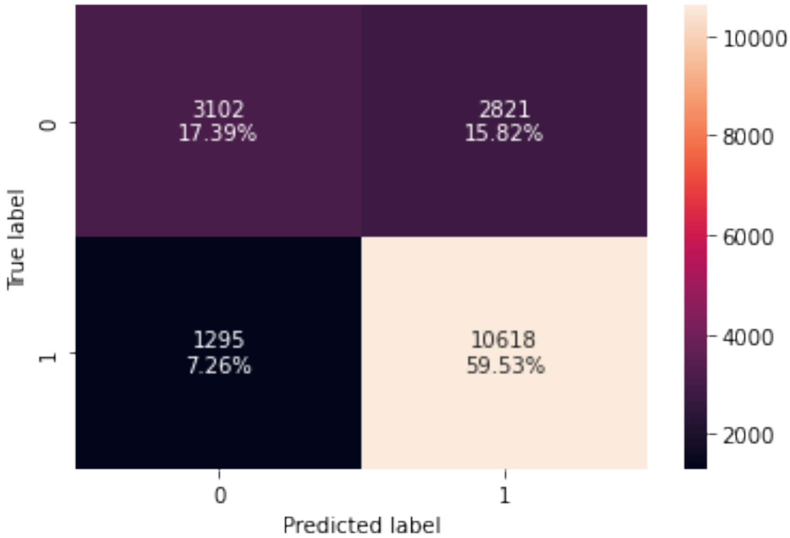
Out[108]:

```
StackingClassifier(estimators=[('AdaBoost', AdaBoostClassifier(random_st
ate=1)),
                               ('Gradient Boosting',
                                GradientBoostingClassifier(init=AdaBoost
Classifier(random_state=1),
                                                           max_features=
0.8,
                                                           n_estimators=
200,
                                                           random_state=
1,
                                                           subsample=1))
,
                               ('Random Forest',
                                RandomForestClassifier(max_depth=10,
                                                       max_features='sqr
t',
                                                       min_samples_split
=7,
                                                       n_estimators=20,
                                                       oob_score=Tru...
                                                 gpu_id=None, grow_polic
y=None,
                                                 importance_type=None,
                                                 interaction_constraints
=None,
                                                 learning_rate=0.1,
                                                 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=No
ne,
                                                 n_estimators=150, n_job
s=None,
                                                 num_parallel_tree=None,
                                                 predictor=None, random_
state=1, ...))
```

Checking model performance on training set

In [109…

```
confusion_matrix_sklearn(stacking_classifier, X_train, y_train) ## Comple
```



In [110…

```
stacking_classifier_model_train_perf = model_performance_classification_s
stacking_classifier_model_train_perf
```

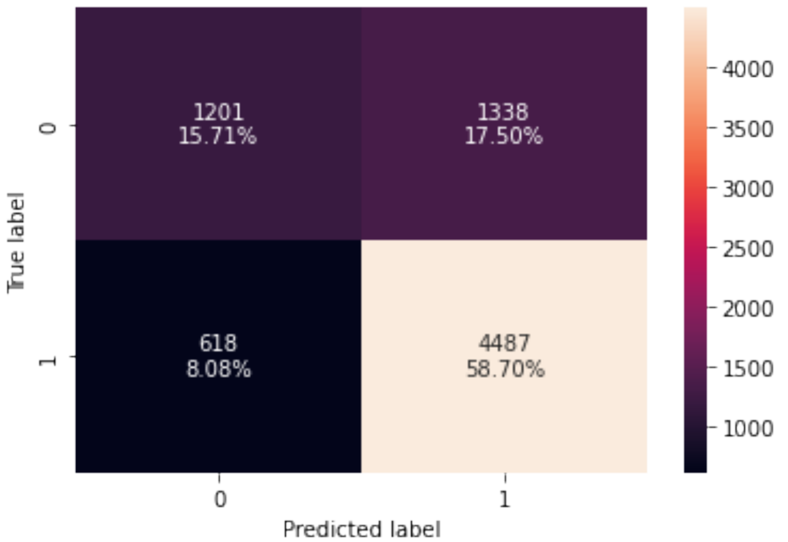
Out[110]:

	Accuracy	Recall	Precision	F1
0	0.769231	0.891295	0.790089	0.837646

Checking model performance on test set

In [111…

```
confusion_matrix_sklearn(stacking_classifier, X_test, y_test) ## Complete
```



In [112…

```
stacking_classifier_model_test_perf = model_performance_classification_sk
stacking_classifier_model_test_perf
```

Out[112]:

	Accuracy	Recall	Precision	F1
0	0.744113	0.878942	0.7703	0.821043

Model Performance Comparison and Final Model Selection

In [113…

```
# training performance comparison

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

```
Training performance comparison:
```

Out[113]:

	Decision Tree	Tuned Decision Tree	Bagging Classifier	Tuned Bagging Classifier	Random Forest	Tuned Random Forest	Adaboost Classifier	Tu Adab Class
Accuracy	1.0	0.712548	0.985198	0.996187	1.0	0.769119	0.738226	0.718
Recall	1.0	0.931923	0.985982	0.999916	1.0	0.918660	0.887182	0.78
Precision	1.0	0.720067	0.991810	0.994407	1.0	0.776556	0.760688	0.79
F1	1.0	0.812411	0.988887	0.997154	1.0	0.841652	0.819080	0.78

In [115…

```
# testing performance comparison

models_test_comp_df = pd.concat(
    [
        decision_tree_perf_test.T,
        dtree_estimator_model_test_perf.T,
        bagging_classifier_model_test_perf.T,
        bagging_estimator_tuned_model_test_perf.T,
        rf_estimator_model_test_perf.T,
        rf_tuned_model_test_perf.T,
        ab_classifier_model_test_perf.T,
        abc_tuned_model_test_perf.T,
        gb_classifier_model_test_perf.T,
        gbc_tuned_model_test_perf.T,
        xgb_classifier_model_test_perf.T,
        xgb_tuned_model_test_perf.T,
        stacking_classifier_model_test_perf.T,
    ],
    axis=1,
)
models_test_comp_df.columns = [     "Decision Tree",     "Tuned Decision Tr
print("Testing performance comparison:")
models_test_comp_df
```

```
Testing performance comparison:
```

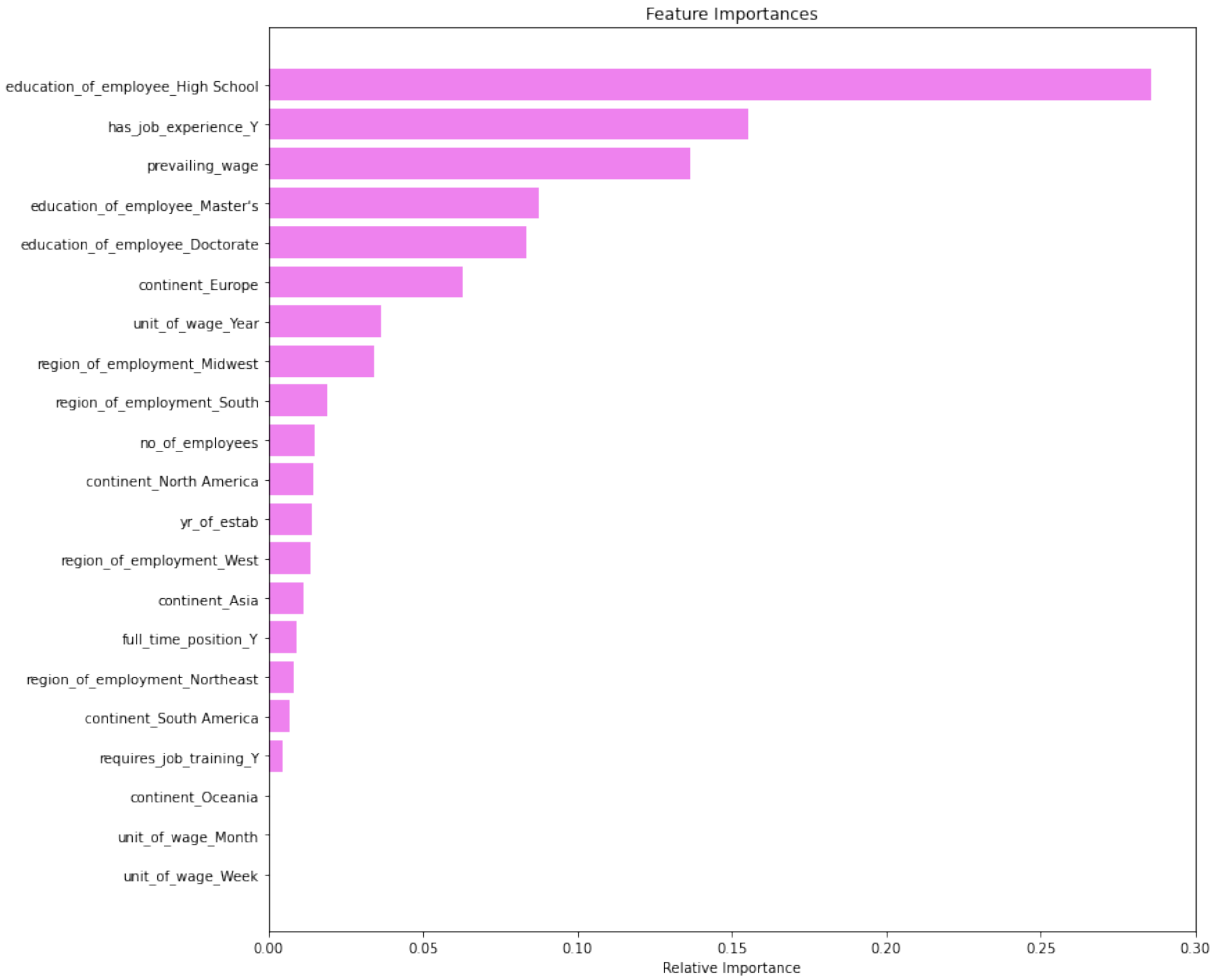
Out[115]:

	Decision Tree	Tuned Decision Tree	Bagging Classifier	Tuned Bagging Classifier	Random Forest	Tuned Random Forest	Adaboost Classifier	Ad Cla
Accuracy	0.664835	0.706567	0.691523	0.724228	0.727368	0.738095	0.738226	0.
Recall	0.742801	0.930852	0.764153	0.895397	0.847209	0.898923	0.887182	0.
Precision	0.752232	0.715447	0.771711	0.743857	0.768343	0.755391	0.760688	0.7
F1	0.747487	0.809058	0.767913	0.812622	0.805851	0.820930	0.819080	0.7

Important features of the final model

In [116…

```
feature_names = X_train.columns
importances = gb_classifier.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
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



Business Insights and Recommendations

-
