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

```
# Installing the libraries with the specified version.
!pip install numpy==1.25.2 pandas==1.5.3 scikit-learn==1.2.2 matplotlib==3.7.1 seaborn==0.13.1 xgboost==2.0.3
```

```
Requirement already satisfied: numpy==1.25.2 in /usr/local/lib/python3.11/dist-packages (1.25.2)
Requirement already satisfied: pandas==1.5.3 in /usr/local/lib/python3.11/dist-packages (1.5.3)
Requirement already satisfied: scikit-learn==1.2.2 in /usr/local/lib/python3.11/dist-packages (1.2.2)
Requirement already satisfied: matplotlib==3.7.1 in /usr/local/lib/python3.11/dist-packages (3.7.1)
Requirement already satisfied: seaborn==0.13.1 in /usr/local/lib/python3.11/dist-packages (0.13.1)
Requirement already satisfied: xgboost==2.0.3 in /usr/local/lib/python3.11/dist-packages (2.0.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.11/dist-packages (from pandas==1.5.3) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas==1.5.3) (2025.1)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.11/dist-packages (from scikit-learn==1.2.2) (1.13.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.11/dist-packages (from scikit-learn==1.2.2) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn==1.2.2) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib==3.7.1) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib==3.7.1) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib==3.7.1) (4.56.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib==3.7.1) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib==3.7.1) (24.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib==3.7.1) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib==3.7.1) (3.2.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.1->pandas==1.5.3) (1.17.0)
```

Note: *After running the above cell, kindly restart the notebook kernel and run all cells sequentially from the start again.*

```
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

from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import StackingClassifier
```

Loading the dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
# original data
df = pd.read_csv('/content/drive/My Drive/Colab Notebooks/5 - Ensemble Techniques/Final Project/EasyVisa.csv')

# copying data to another variable to avoid any changes to original data
data = df.copy()
```

Data Overview

- Observations
- Sanity checks

Loading the dataset

```
data.head()
```

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

Next steps: Generate code with data View recommended plots New interactive sheet

```
data.tail()
```

	case_id	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees	yr_of_estab	region_of_employment	prevailing_wage	unit_of
25475	EZYV25476	Asia	Bachelor's	Y	Y	2601	2008	South	77092.57	
25476	EZYV25477	Asia	High School	Y	N	3274	2006	Northeast	279174.79	
25477	EZYV25478	Asia	Master's	Y	N	1121	1910	South	146298.85	
25478	EZYV25479	Asia	Master's	Y	Y	1918	1887	West	86154.77	
25479	EZYV25480	Asia	Bachelor's	Y	N	3195	1960	Midwest	70876.91	

Shape of the dataset

```
data.shape
```

```
(25480, 12)
```

Observations - There are 25,480 rows and 12 columns in the dataset

Info regarding column datatypes

```
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
```

Observations - There are 3 numerical (2 int64 & 1 float64) and 8 object type columns in the dataset

Statistics summary for the numerical columns

```
data.describe()
```

	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

Checking missing values

```
data.isnull().sum()
```

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

dtype: int64

Observations - There is no missing values in the data

Check for duplicates in the dataset

```
print("There are",data.duplicated().sum(),"duplicated rows")
```

```
There are 0 duplicated rows
```

Dropping the columns with all unique values

```
data.case_id.nunique()
```

```
25480
```

Observations - the case_id column contains only unique values, so we can drop it

```
#drop the case_id column
data = data.drop(["case_id"], axis=1)
```

```
#get info after dropping the case_id column
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25480 entries, 0 to 25479
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype
```

```
 ---  ------                 --------------  -----
 0   continent              25480 non-null  object
 1   education_of_employee  25480 non-null  object
 2   has_job_experience     25480 non-null  object
 3   requires_job_training  25480 non-null  object
 4   no_of_employees        25480 non-null  int64
 5   yr_of_estab            25480 non-null  int64
 6   region_of_employment   25480 non-null  object
 7   prevailing_wage        25480 non-null  float64
 8   unit_of_wage           25480 non-null  object
 9   full_time_position     25480 non-null  object
 10  case_status            25480 non-null  object
dtypes: float64(1), int64(2), object(8)
memory usage: 2.1+ MB
```

Observations - There are 3 numerical (2 int64 & 1 float64) and 7 object type columns in the dataset

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.

Statistical summary of the data

```
data.describe().T
```

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

Fixing negative values for the no_of_employees column

```
data.loc[data['no_of_employees'] < 0].shape
```

```
(33, 11)
```

```
data["no_of_employees"] = abs(data["no_of_employees"])
```

Print categorical values

```
categories = list(data.select_dtypes("object").columns)

for col in categories:
    print(data[col].value_counts())
    print("-" * 100)
```

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

Univariate Analysis

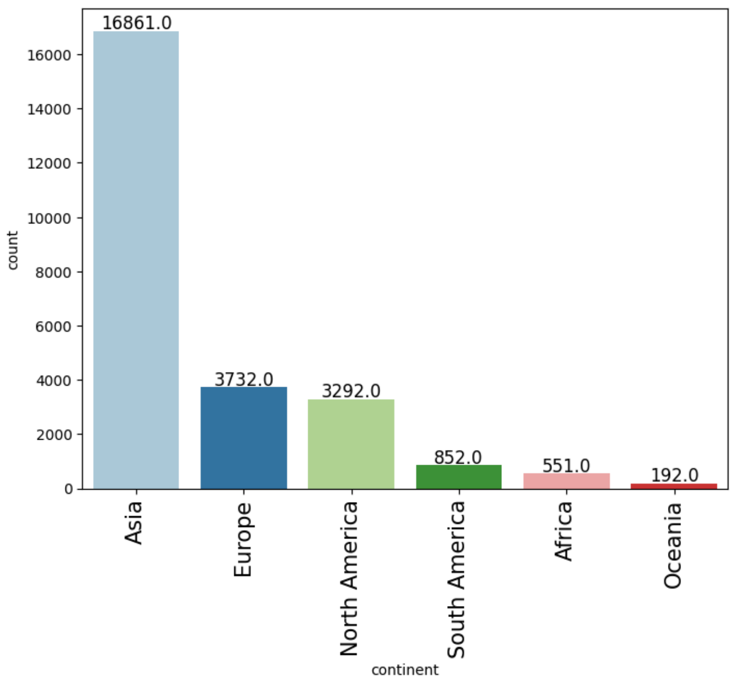
Support functions

```
# function to create histogram boxplot
def histogram_boxplot(data, feature, figsize=(15, 10), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (15,10))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a triangle will indicate the mean value of the column
    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


# 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


# function to plot distributions wrt target
def distribution_plot_wrt_target(data, predictor, target):

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

    target_uniq = data[target].unique()

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

    axs[0, 1].set_title("Distribution of target for target=" + str(target_uniq[1]))
    sns.histplot(
```

```
        data=data[data[target] == target_uniq[1]],
        x=predictor,
        kde=True,
        ax=axs[0, 1],
        color="orange",
        stat="density",
    )

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

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

    plt.tight_layout()
    plt.show()


# function to plot stacked barplot
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 + 5, 5))
    plt.legend(
        loc="lower left", frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

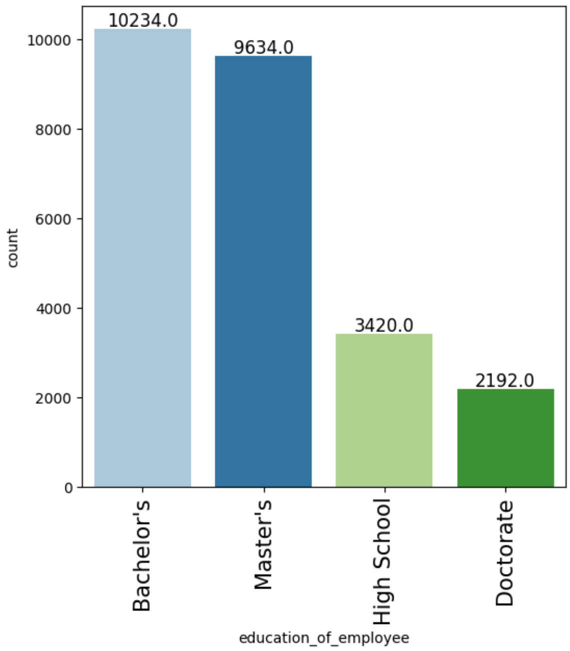
continent

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



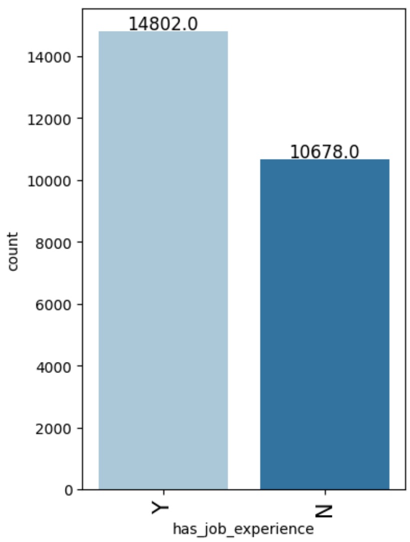
education_of_employee

```
labeled_barplot(data, "education_of_employee")
```



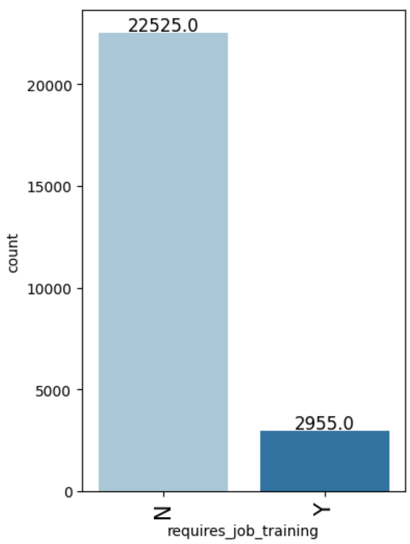
has_job_experience

```
labeled_barplot(data, "has_job_experience")
```



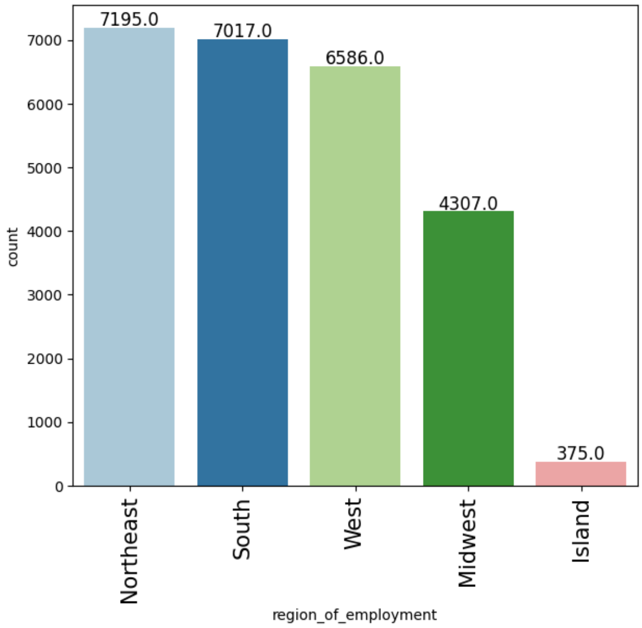
requires_job_training

```
labeled_barplot(data, "requires_job_training")
```



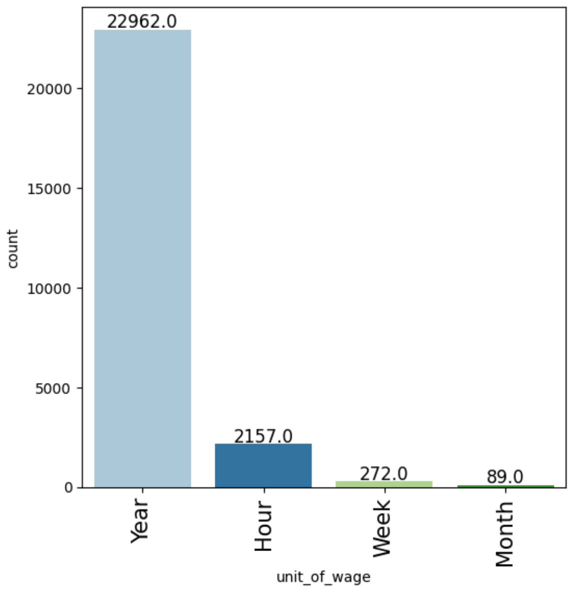
region_of_employment

```
labeled_barplot(data, "region_of_employment")
```



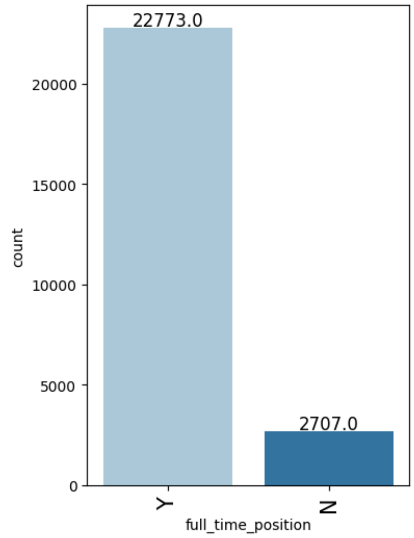
unit_of_wage

```
labeled_barplot(data, "unit_of_wage")
```



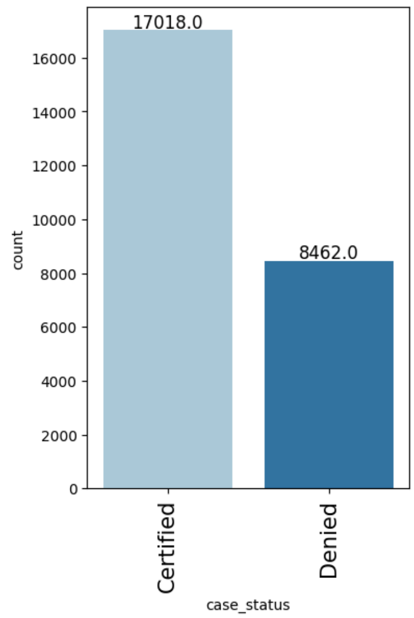
full_time_position

```
labeled_barplot(data, "full_time_position")
```



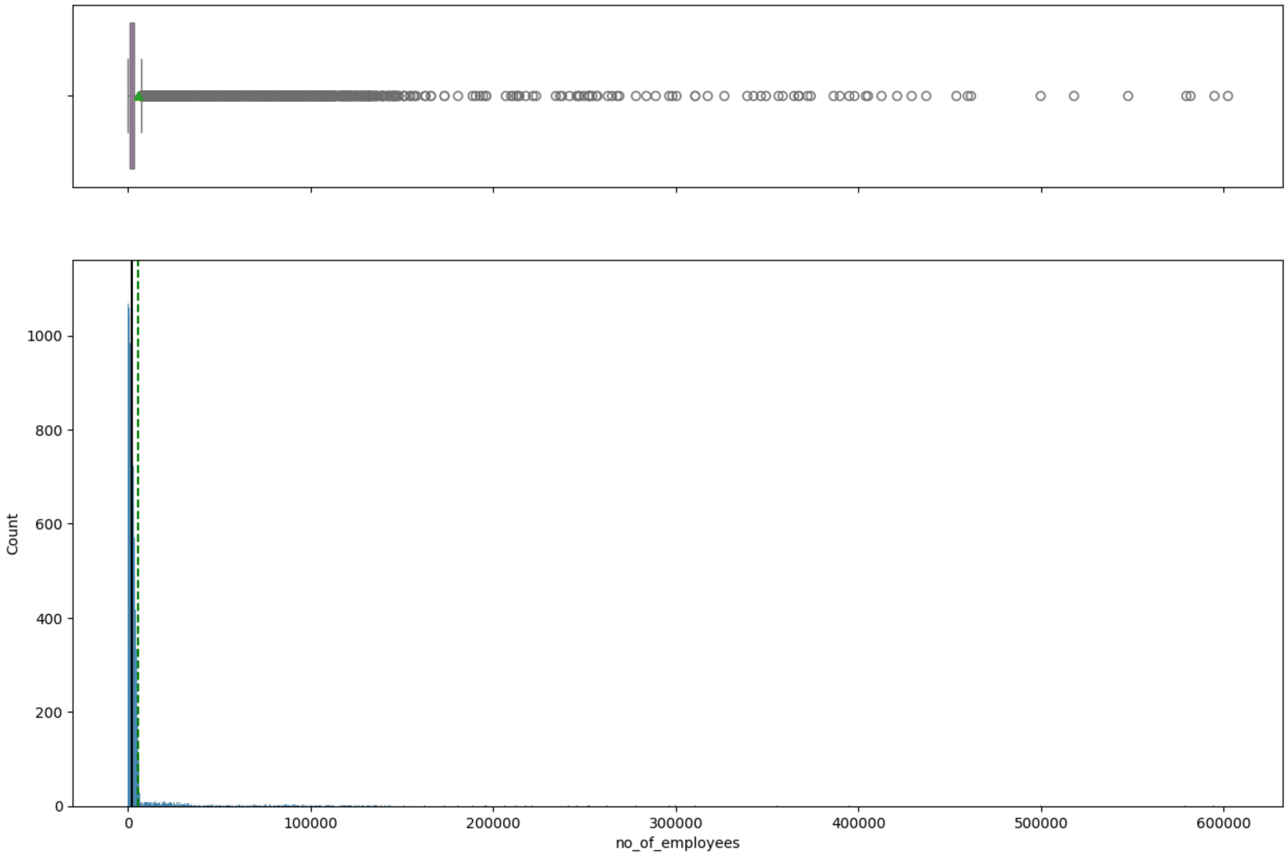
case_status

```
labeled_barplot(data, "case_status")
```



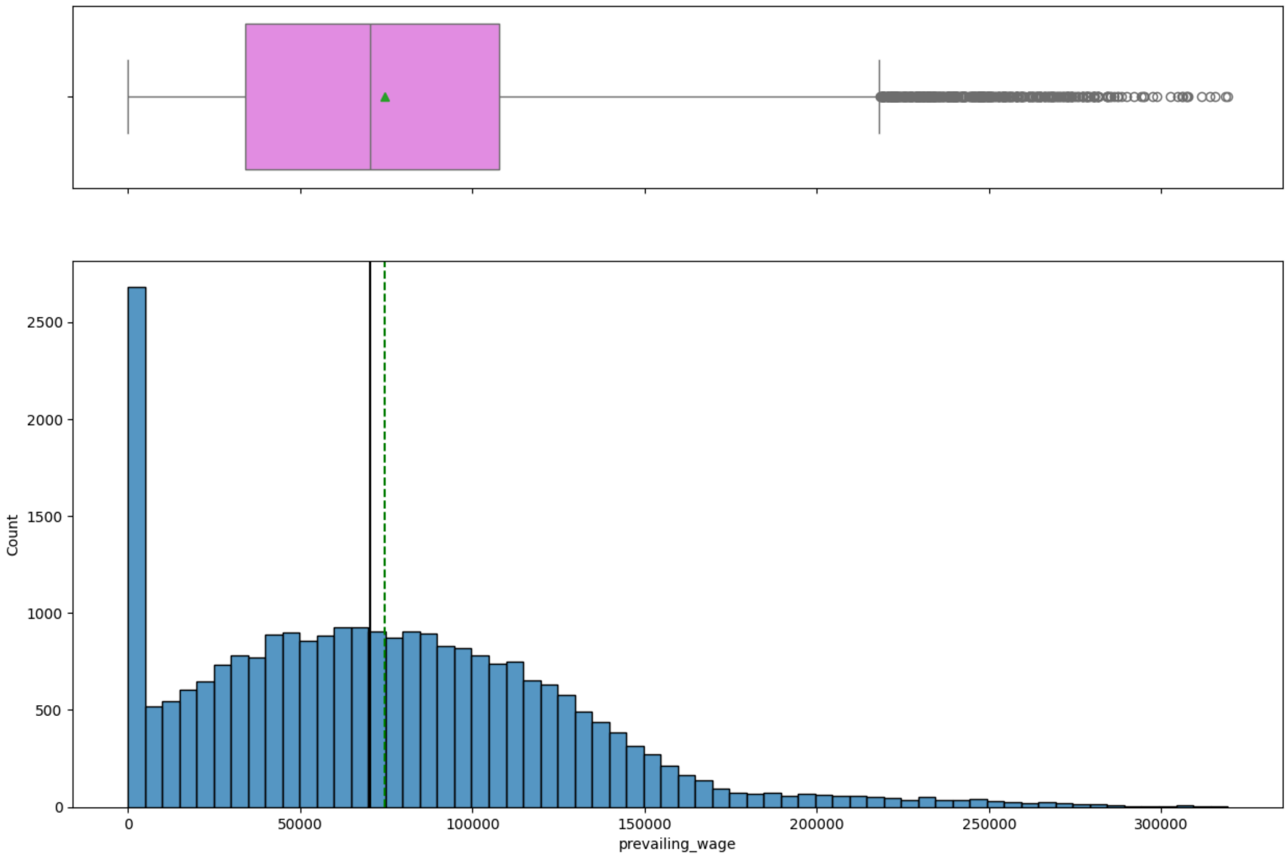
no_of_employees

```
histogram_boxplot(data=data, feature="no_of_employees")
```



prevailing_wage

```
histogram_boxplot(data, "prevailing_wage")
```



Check underpaid workers - prevailing_wage

```
data.loc[data['prevailing_wage'] < 100]
```

	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees	yr_of_estab	region_of_employment	prevailing_wage	unit_of_wage	full_
338	Asia	Bachelor's	Y	N	2114	2012	Northeast	15.7716	Hour	
634	Asia	Master's	N	N	834	1977	Northeast	3.3188	Hour	
839	Asia	High School	Y	N	4537	1999	West	61.1329	Hour	
876	South America	Bachelor's	Y	N	731	2004	Northeast	82.0029	Hour	
995	Asia	Master's	N	N	302	2000	South	47.4872	Hour	
...	...	...	...	...	...	...	...	...	...	
25023	Asia	Bachelor's	N	Y	3200	1994	South	94.1546	Hour	
25258	Asia	Bachelor's	Y	N	3659	1997	South	79.1099	Hour	
25308	North America	Master's	N	N	82953	1977	Northeast	42.7705	Hour	
25329	Africa	Bachelor's	N	N	2172	1993	Northeast	32.9286	Hour	

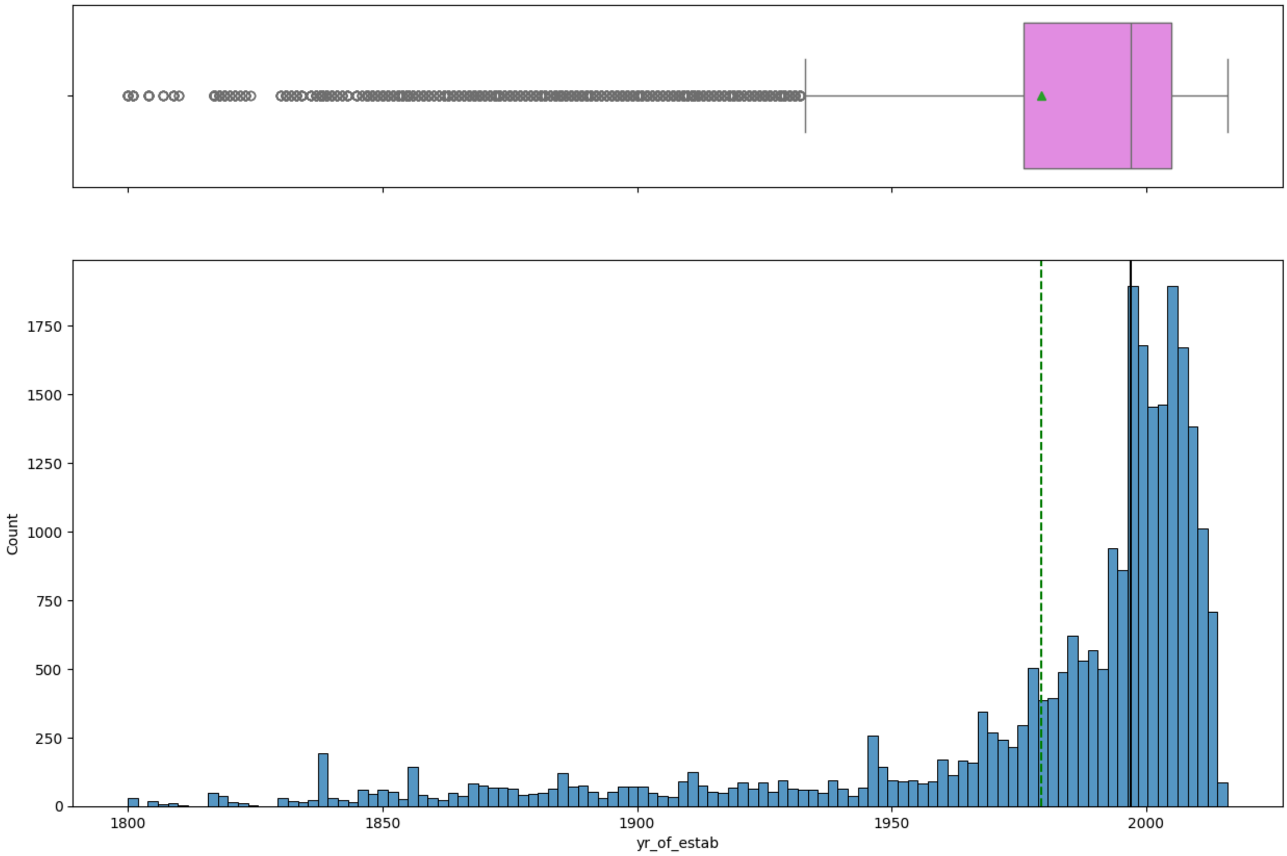
```
data.loc[data["prevailing_wage"] < 100, "unit_of_wage"].value_counts()
```

	unit_of_wage
Hour	176

dtype: int64

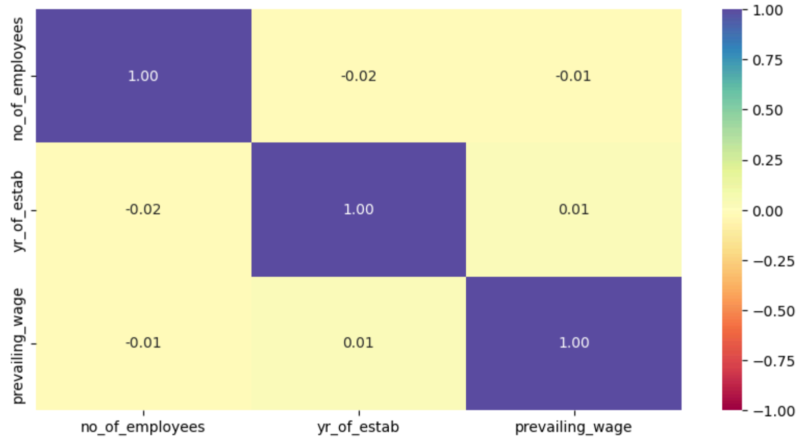
yr_of_estab

```
histogram_boxplot(data, "yr_of_estab")
```



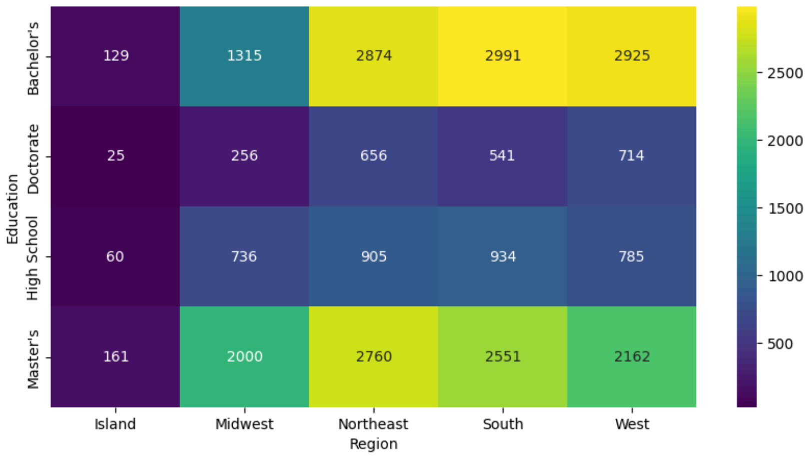
Bivariate Analysis

```
# Select numerical columns
cols_list = data.select_dtypes(include=np.number).columns.tolist()

# Plot heatmap
plt.figure(figsize=(10, 5))
sns.heatmap(data[cols_list].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral")
plt.show()
```



education_of_employee VS region_of_employment

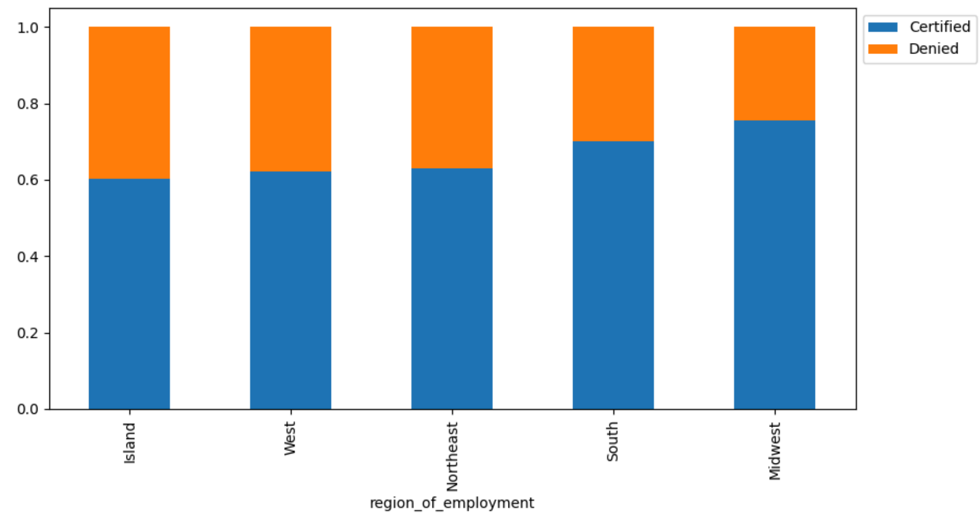
```
plt.figure(figsize=(10, 5))
sns.heatmap(pd.crosstab(data['education_of_employee'], data['region_of_employment']),annot=True,fmt="g",cmap="viridis")

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



region_of_employment VS case_status

```
stacked_barplot(data, 'region_of_employment', 'case_status')
```

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



continent VS case_status

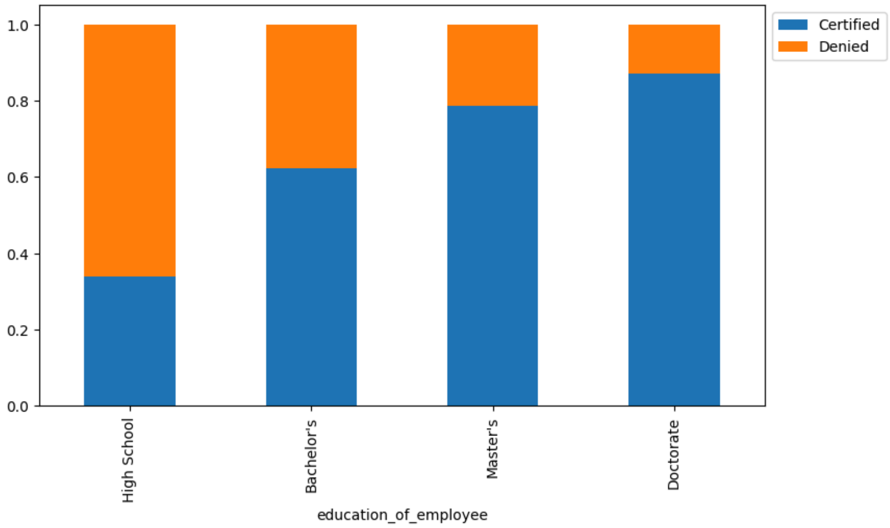
has_job_experience VS case_status

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?

```
stacked_barplot(data, "education_of_employee", "case_status")
```

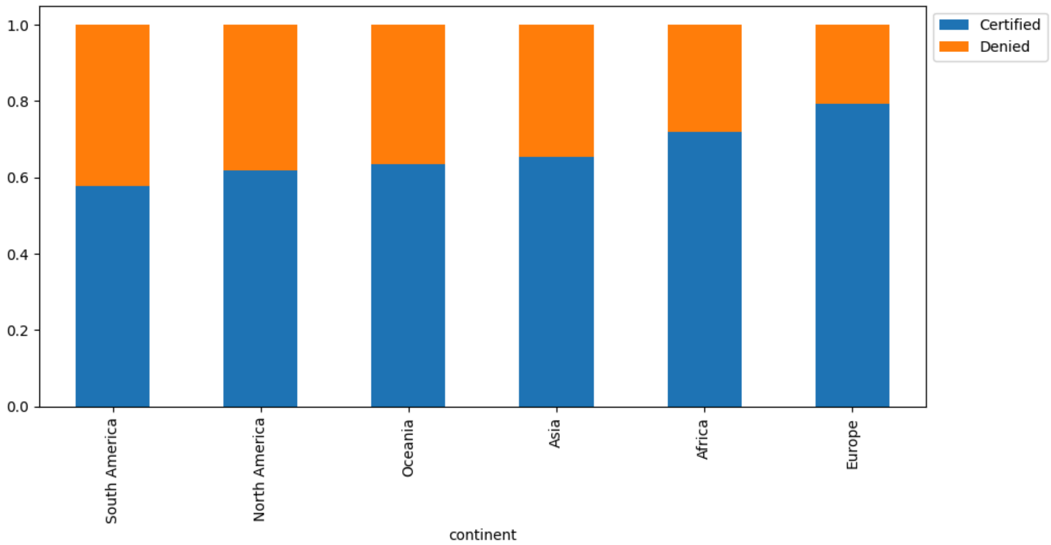
```
case_status           Certified  Denied    All
education_of_employee
All                       17018    8462  25480
Bachelor's                 6367    3867  10234
High School                1164    2256   3420
Master's                   7575    2059   9634
Doctorate                  1912     280   2192
------------------------------------------------------------------------------------------------------------------------
```



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

```
stacked_barplot(data, 'continent', 'case_status')
```

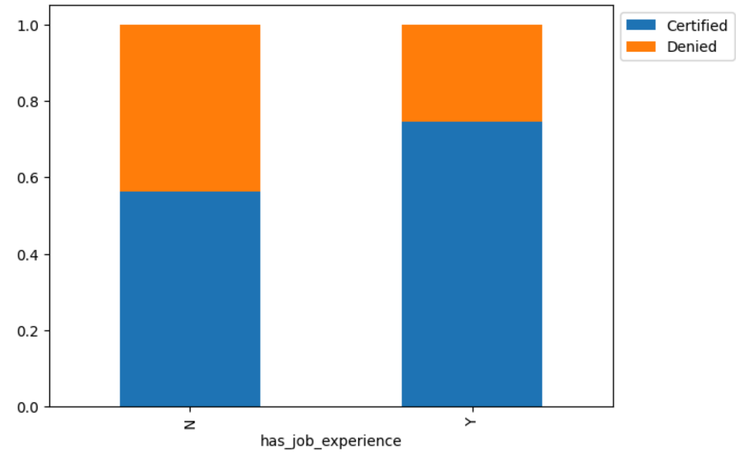
```
case_status    Certified  Denied    All
continent
All                17018    8462  25480
Asia               11012    5849  16861
North America       2037    1255   3292
Europe              2957     775   3732
South America        493     359    852
Africa               397     154    551
Oceania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



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

```
stacked_barplot(data, 'has_job_experience', 'case_status')
```

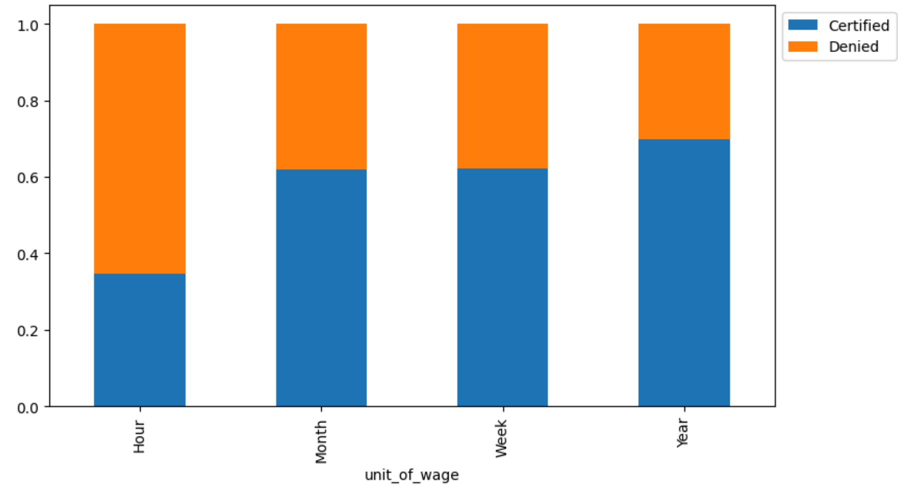
```
case_status        Certified  Denied    All
has_job_experience
All                    17018    8462  25480
N                       5994    4684  10678
Y                      11024    3778  14802
------------------------------------------------------------------------------------------------------------------------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



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

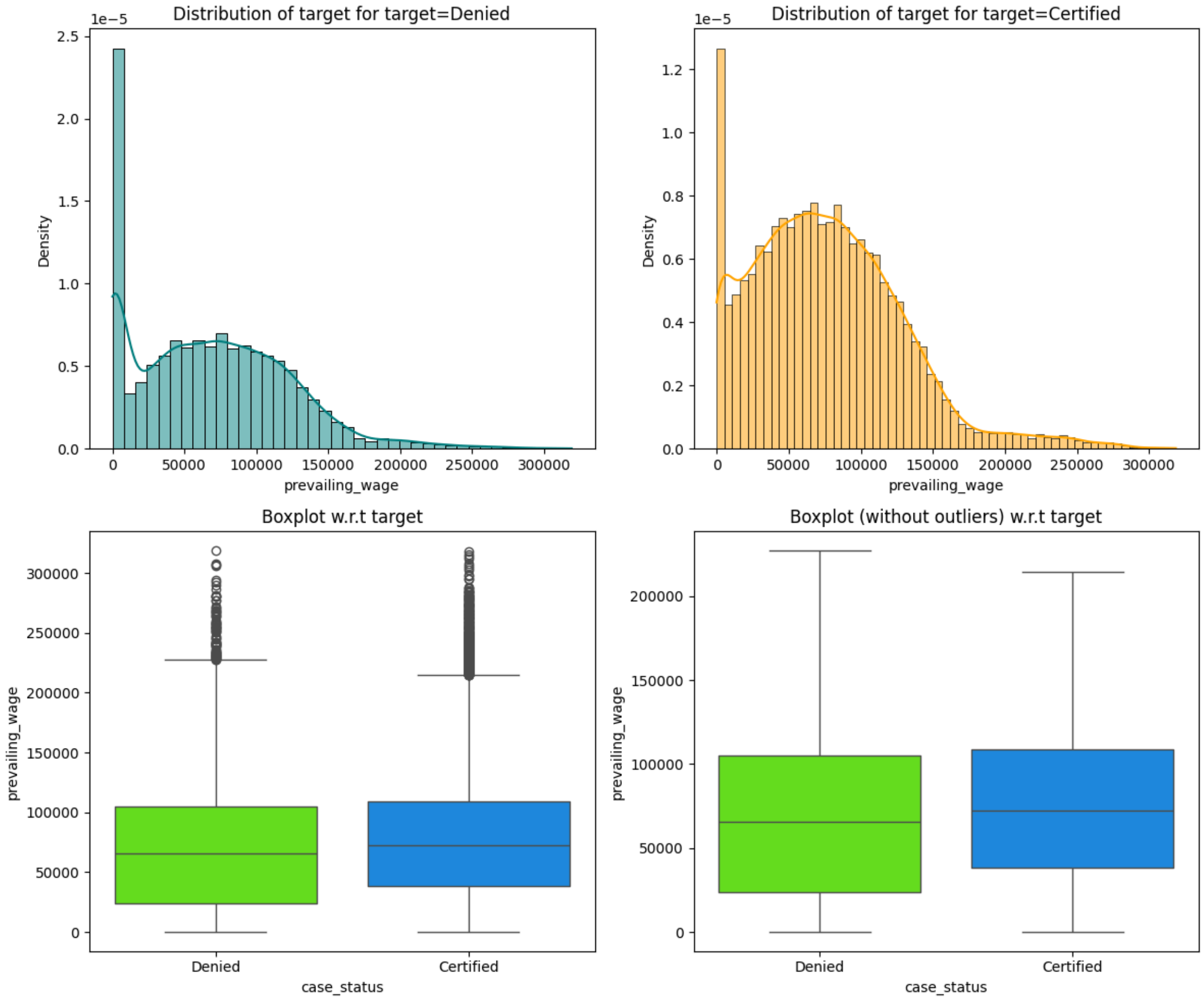
```
stacked_barplot(data, 'unit_of_wage', 'case_status')
```

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



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

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

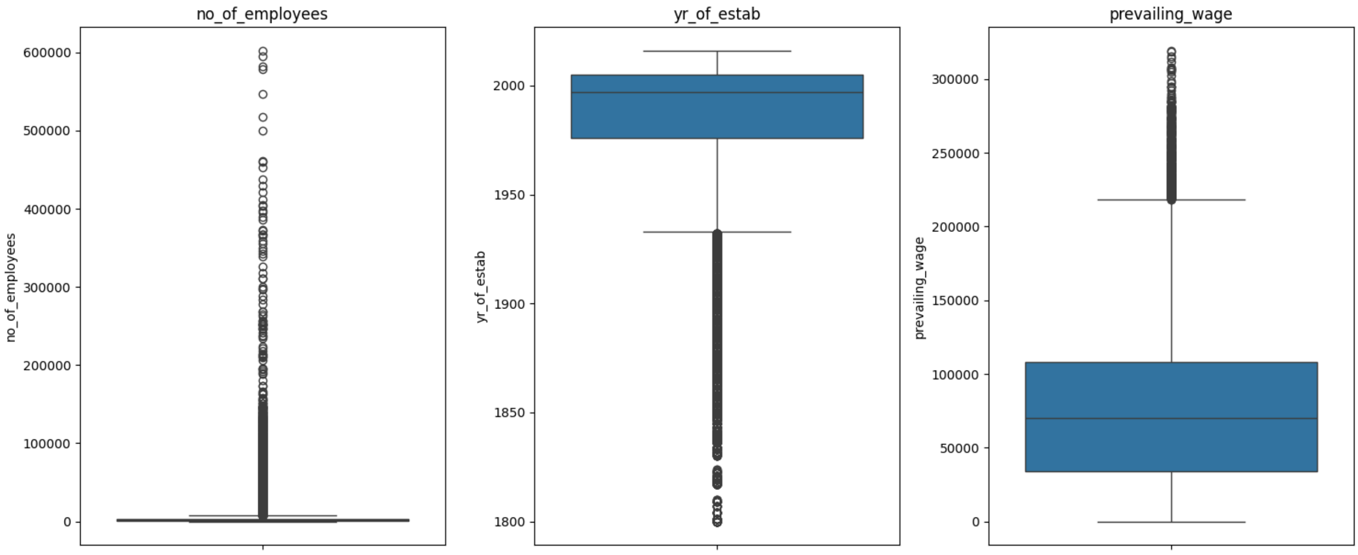


Data Preprocessing

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

Check outliers

```
numeric_columns = data.select_dtypes(include=np.number).columns.tolist()

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

for i, variable in enumerate(numeric_columns):
    plt.subplot(len(numeric_columns) // 3 + 1, 3, i + 1)
    sns.boxplot(y=data[variable])
    plt.title(variable)

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



Data Preparation for modeling

```
# Convert 'case_status' to binary
data["case_status"] = data["case_status"].apply(lambda x: 1 if x == "Certified" else 0)

# Drop 'case_status' from the data to create feature set X
X = data.drop(columns=["case_status"])

# Create dummy variables for X
X = pd.get_dummies(X)

# Define the target variable Y
Y = data["case_status"]

# Splitting data into train and test sets in the ratio 70:30
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=1, stratify=Y)

# Example print statements to confirm shapes of the splits
print(f"X_train shape: {X_train.shape}")
print(f"X_test shape: {X_test.shape}")
print(f"y_train shape: {y_train.shape}")
print(f"y_test shape: {y_test.shape}")
```

```
X_train shape: (17836, 28)
X_test shape: (7644, 28)
y_train shape: (17836,)
y_test shape: (7644,)
```

EDA

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

[ ] ↳ 5 cells hidden

Building bagging and boosting models

Support functions

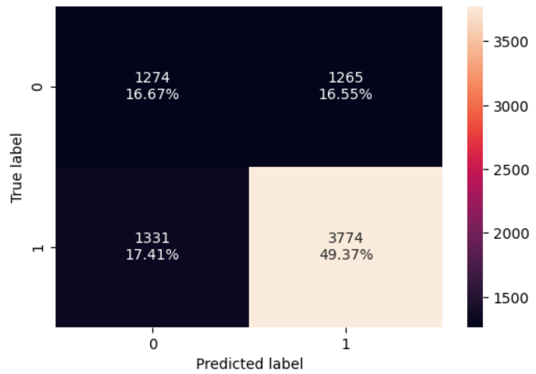
```
# defining a function to compute different metrics to check performance of a classification model built using sklearn
def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

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

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

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

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

    return df_perf

def confusion_matrix_sklearn(model, predictors, target):
    """
```

```
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    y_pred = model.predict(predictors)
    cm = confusion_matrix(target, y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

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

Decision Tree

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

#Calculating different metrics
decision_tree_perf_train=model_performance_classification_sklearn(d_tree,X_train,y_train)
print("Training performance:\n",decision_tree_perf_train)
decision_tree_perf_test=model_performance_classification_sklearn(d_tree,X_test,y_test)
print("Testing performance:\n",decision_tree_perf_test)
#Creating confusion matrix
confusion_matrix_sklearn(d_tree, X_test, y_test)
```

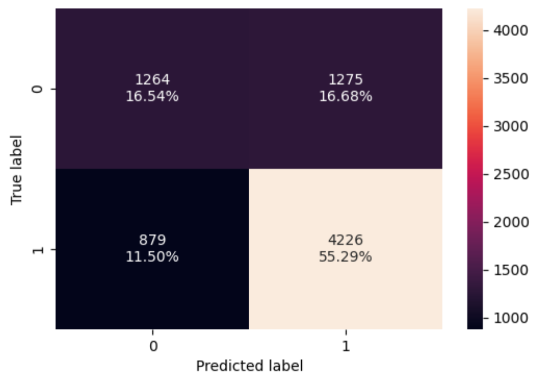
```
Training performance:
    Accuracy  Recall  Precision   F1
0       1.0     1.0        1.0  1.0
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.660387  0.739275   0.748958  0.744085
```



Random Forest

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

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

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

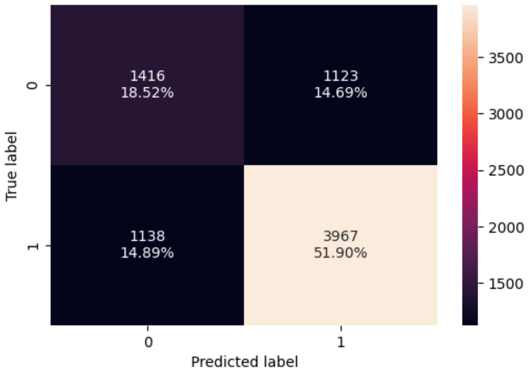
```
Training performance:
    Accuracy    Recall  Precision        F1
0  0.999944  0.999916        1.0  0.999958
Testing performance:
    Accuracy    Recall  Precision        F1
0   0.71821  0.827816   0.768224  0.796907
```



Bagging Classifier

```
#Fitting the model
bagging_classifier = BaggingClassifier(random_state=1)
bagging_classifier.fit(X_train,y_train)

#Calculating different metrics
bagging_classifier_model_train_perf=model_performance_classification_sklearn(bagging_classifier,X_train,y_train)
print("Training performance:\n",bagging_classifier_model_train_perf)
bagging_classifier_model_test_perf=model_performance_classification_sklearn(bagging_classifier,X_test,y_test)
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  0.983797  0.984639   0.991044  0.987831
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.704212  0.777081   0.779371  0.778225
```



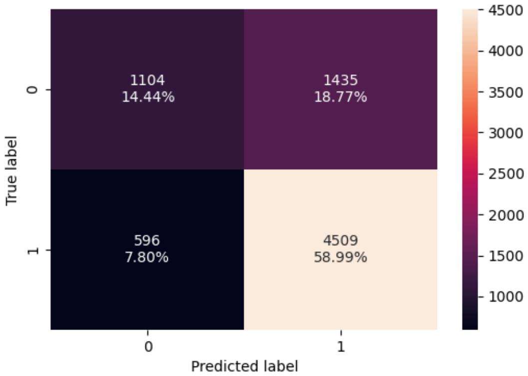
AdaBoost Classifier

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

```
AdaBoostClassifier
AdaBoostClassifier(random_state=1)
```

```
#Calculating different metrics
ab_classifier_model_train_perf=model_performance_classification_sklearn(ab_classifier,X_train,y_train)
print("Training performance:\n",ab_classifier_model_train_perf)
ab_classifier_model_test_perf=model_performance_classification_sklearn(ab_classifier,X_test,y_test)
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  0.738058  0.886259   0.760937  0.81883
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.734301  0.883252    0.75858  0.816182
```



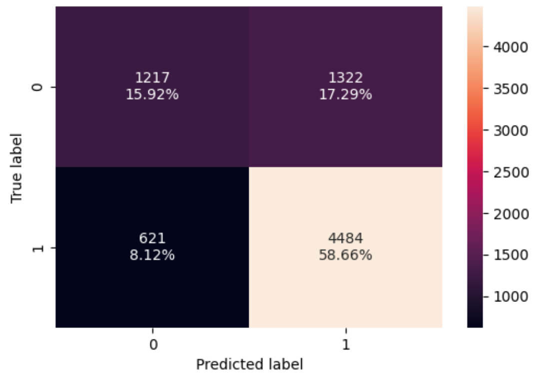
Gradient Boosting Classifier

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

```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=1)
```

```
#Calculating different metrics
gb_classifier_model_train_perf=model_performance_classification_sklearn(gb_classifier,X_train,y_train)
print("Training performance:\n",gb_classifier_model_train_perf)
gb_classifier_model_test_perf=model_performance_classification_sklearn(gb_classifier,X_test,y_test)
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  0.757849  0.883657   0.782095  0.82978
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.745814  0.878355   0.772305  0.821923
```



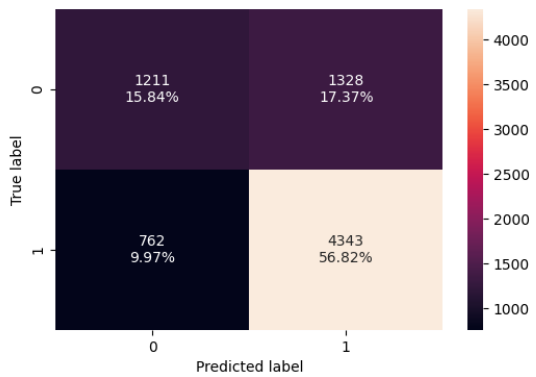
XGBoost Classifier

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

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              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=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, random_state=1, ...)
```

```
#Calculating different metrics
xgb_classifier_model_train_perf=model_performance_classification_sklearn(xgb_classifier,X_train,y_train)
print("Training performance:\n",xgb_classifier_model_train_perf)
xgb_classifier_model_test_perf=model_performance_classification_sklearn(xgb_classifier,X_test,y_test)
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  0.840884  0.930664     0.8464  0.886534
Testing performance:
    Accuracy    Recall  Precision       F1
0  0.726583  0.850735   0.765826  0.80605
```



Note

1. Sample parameter grids have been provided to do necessary hyperparameter tuning. These sample grids are expected to provide a balance between model performance improvement and execution time. One can extend/reduce the parameter grid based on execution time and system configuration.
    - Please note that if the parameter grid is extended to improve the model performance further, the execution time will increase

- For Gradient Boosting:

```
param_grid = {
    "init": [AdaBoostClassifier(random_state=1),DecisionTreeClassifier(random_state=1)],
    "n_estimators": np.arange(50,110,25),
    "learning_rate": [0.01,0.1,0.05],
    "subsample":[0.7,0.9],
    "max_features":[0.5,0.7,1],
}
```

- For Adaboost:

```
param_grid = {
    "n_estimators": np.arange(50,110,25),
    "learning_rate": [0.01,0.1,0.05],
    "base_estimator": [
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=1),
    ],
}
```

- For Bagging Classifier:

```
param_grid = {
    'max_samples': [0.8,0.9,1],
    'max_features': [0.7,0.8,0.9],
    'n_estimators' : [30,50,70],
}
```

- For Random Forest:

```
param_grid = {
    "n_estimators": [50,110,25],
    "min_samples_leaf": np.arange(1, 4),
    "max_features": [np.arange(0.3, 0.6, 0.1),'sqrt'],
    "max_samples": np.arange(0.4, 0.7, 0.1)
}
```

- For Decision Trees:

```
param_grid = {
    'max_depth': np.arange(2,6),
    'min_samples_leaf': [1, 4, 7],
    'max_leaf_nodes' : [10, 15],
    'min_impurity_decrease': [0.0001,0.001]
}
```

- For XGBoost:

```
param_grid={'n_estimators':np.arange(50,110,25),
            'scale_pos_weight':[1,2,5],
            'learning_rate':[0.01,0.1,0.05],
            'gamma':[1,3],
            'subsample':[0.7,0.9]
}
```

Will tuning the hyperparameters improve the model performance?

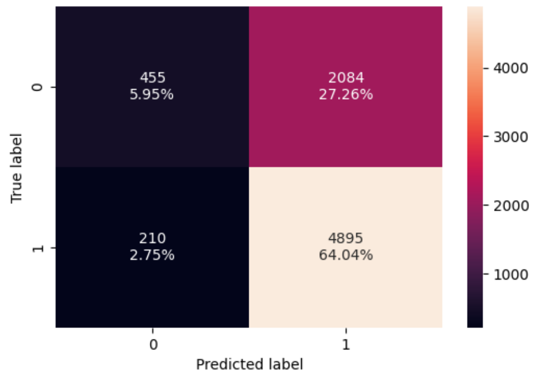
Tuning Decision Tree

```
#Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(class_weight={0:0.35,1:0.65},random_state=1)

# Grid of parameters as provided above
parameters = {
    'max_depth': np.arange(2,6),
    'min_samples_leaf': [1, 4, 7],
    'max_leaf_nodes' : [10, 15],
    'min_impurity_decrease': [0.0001,0.001]
}

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

```
DecisionTreeClassifier

DecisionTreeClassifier(class_weight={0: 0.35, 1: 0.65}, max_depth=2,
                       max_leaf_nodes=10, min_impurity_decrease=0.0001,
                       random_state=1)
```

```
#Calculating different metrics
dtree_estimator_model_train_perf=model_performance_classification_sklearn(dtree_estimator,X_train,y_train)
print("Training performance:\n",dtree_estimator_model_train_perf)
dtree_estimator_model_test_perf=model_performance_classification_sklearn(dtree_estimator,X_test,y_test)
print("Testing performance:\n",dtree_estimator_model_test_perf)
#Creating confusion matrix
confusion_matrix_sklearn(dtree_estimator, X_test, y_test)
```

```
Training performance:
   Accuracy    Recall  Precision        F1
0  0.706773  0.960883   0.706125  0.814038
Testing performance:
   Accuracy    Recall  Precision        F1
0  0.699895  0.958864    0.70139  0.810162
```



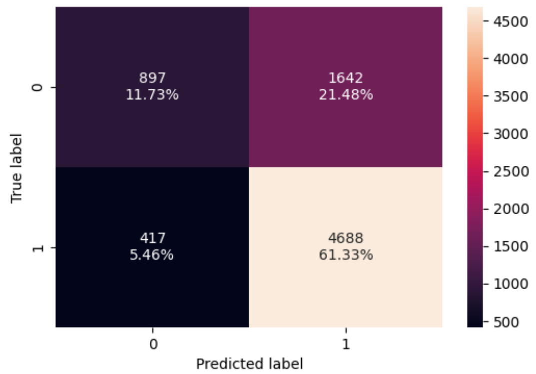
Tunning Random Forest

```
# Choose the type of classifier.
rf_tuned = RandomForestClassifier(class_weight={0:0.35,1:0.65},random_state=1)

parameters = {
    "n_estimators": [50,110,25],
    "min_samples_leaf": np.arange(1, 4),
    "max_features": [np.arange(0.3, 0.6, 0.1),'sqrt'],
    "max_samples": np.arange(0.4, 0.7, 0.1)
}


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

```
                    RandomForestClassifier
RandomForestClassifier(class_weight={0: 0.35, 1: 0.65}, max_samples=0.4,
                       min_samples_leaf=3, n_estimators=25, random_state=1)
```

```
#Calculating different metrics
rf_tuned_model_train_perf=model_performance_classification_sklearn(rf_tuned,X_train,y_train)
print("Training performance:\n",rf_tuned_model_train_perf)
rf_tuned_model_test_perf=model_performance_classification_sklearn(rf_tuned,X_test,y_test)
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  0.783135  0.954168   0.773844  0.854597
Testing performance:
   Accuracy    Recall  Precision        F1
0  0.730638  0.918315     0.7406  0.819939
```



Tuning Bagging Classifier

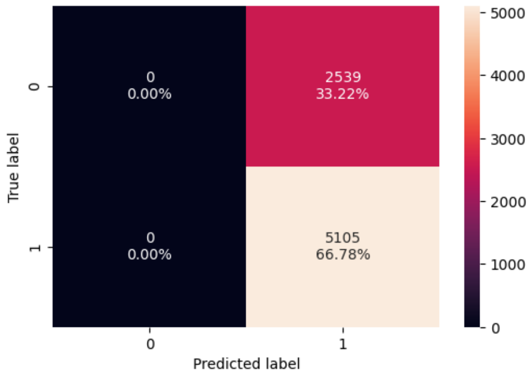
```
# Choose the type of classifier.
bagging_estimator_tuned = BaggingClassifier(random_state=1)

# Grid of parameters as provided above
parameters = {
    'max_samples': [0.8,0.9,1],
    'max_features': [0.7,0.8,0.9],
```

```
    'n_estimators' : [30,50,70],
}

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

# Run the grid search
grid_obj = GridSearchCV(bagging_estimator_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
bagging_estimator_tuned = grid_obj.best_estimator_

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

```
BaggingClassifier
BaggingClassifier(max_features=0.7, max_samples=1, n_estimators=30,
                  random_state=1)
```

```
#Calculating different metrics
bagging_estimator_tuned_model_train_perf=model_performance_classification_sklearn(bagging_estimator_tuned,X_train,y_train)
print("Training performance:\n",bagging_estimator_tuned_model_train_perf)
bagging_estimator_tuned_model_test_perf=model_performance_classification_sklearn(bagging_estimator_tuned,X_test,y_test)
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  0.667919     1.0   0.667919  0.800901
Testing performance:
    Accuracy  Recall  Precision        F1
0  0.667844     1.0   0.667844  0.800847
```



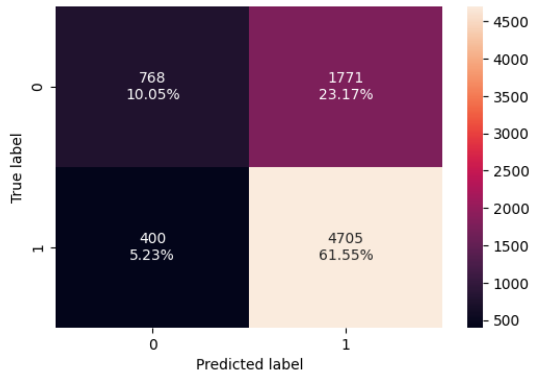
Tuning AdaBoost Classifier

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

# Grid of parameters as provided above
parameters = {
    "n_estimators": np.arange(50,110,25),
    "learning_rate": [0.01,0.1,0.05],
    "base_estimator": [
        DecisionTreeClassifier(max_depth=2, random_state=1),
        DecisionTreeClassifier(max_depth=3, random_state=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=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)
```

```
AdaBoostClassifier
base_estimator: DecisionTreeClassifier
    DecisionTreeClassifier
```

```
#Calculating different metrics
abc_tuned_model_train_perf=model_performance_classification_sklearn(bagging_estimator_tuned,X_train,y_train)
print("Training performance:\n",abc_tuned_model_train_perf)
abc_tuned_model_test_perf=model_performance_classification_sklearn(bagging_estimator_tuned,X_test,y_test)
print("Testing performance:\n",abc_tuned_model_test_perf)

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

```
Training performance:
    Accuracy  Recall  Precision        F1
0  0.667919     1.0   0.667919  0.800901
Testing performance:
    Accuracy  Recall  Precision        F1
0  0.667844     1.0   0.667844  0.800847
```



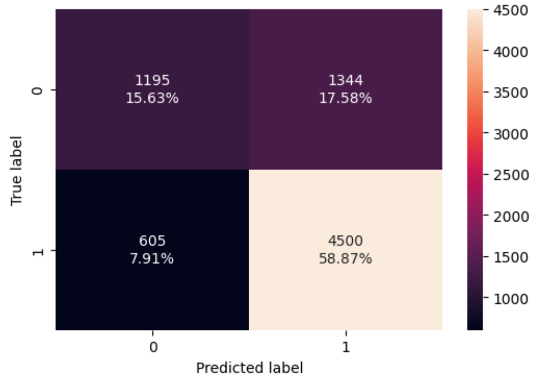
Tuning Gradient Boosting Classifier

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

# Grid of parameters as provided above
parameters = {
    "init": [AdaBoostClassifier(random_state=1),DecisionTreeClassifier(random_state=1)],
    "n_estimators": np.arange(50,110,25),
    "learning_rate": [0.01,0.1,0.05],
    "subsample":[0.7,0.9],
    "max_features":[0.5,0.7,1],
}

# 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)

# 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)
```

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

```
#Calculating different metrics
gbc_tuned_model_train_perf=model_performance_classification_sklearn(gbc_tuned,X_train,y_train)
print("Training performance:\n",gbc_tuned_model_train_perf)
gbc_tuned_model_test_perf=model_performance_classification_sklearn(gbc_tuned,X_test,y_test)
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  0.753756  0.885671   0.776894  0.827724
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.745029  0.881489   0.770021  0.821993
```



Tuning XGBoost Classifier

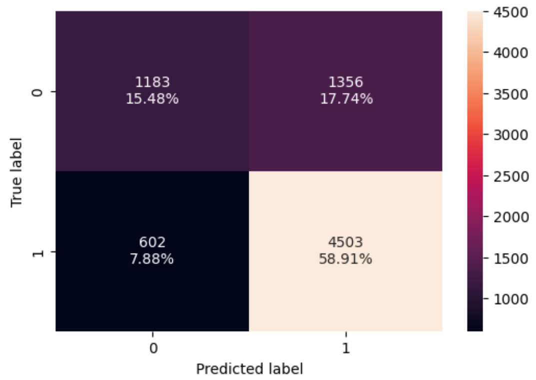
```
# Choose the type of classifier.
xgb_tuned = XGBClassifier(random_state=1, eval_metric="logloss")
```

```
# Grid of parameters as provided above
parameters = {'n_estimators':np.arange(50,110,25),
            'scale_pos_weight':[1,2,5],
            'learning_rate':[0.01,0.1,0.05],
            'gamma':[1,3],
            'subsample':[0.7,0.9]
}

# 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)

# 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)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=3, 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, multi_strategy=None, n_estimators=50,
              n_jobs=None, num_parallel_tree=None, random_state=1, ...)
```

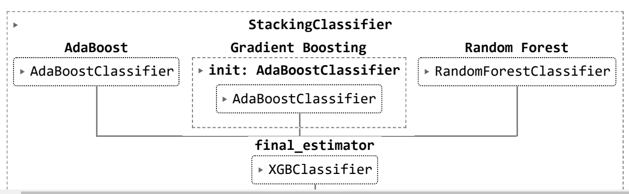
```
#Calculating different metrics
xgb_tuned_model_train_perf=model_performance_classification_sklearn(xgb_tuned,X_train,y_train)
print("Training performance:\n",xgb_tuned_model_train_perf)
xgb_tuned_model_test_perf=model_performance_classification_sklearn(xgb_tuned,X_test,y_test)
print("Testing performance:\n",xgb_tuned_model_test_perf)

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

```
Training performance:
    Accuracy   Recall  Precision       F1
0  0.759475  0.89247   0.779415  0.83212
Testing performance:
    Accuracy    Recall  Precision        F1
0  0.743851  0.882076   0.768561  0.821416
```



Stacking Classifier

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

final_estimator = XGBClassifier(random_state=1)

stacking_classifier = StackingClassifier(
    estimators=estimators,
    final_estimator=final_estimator,
    cv=5
)
stacking_classifier.fit(X_train, y_train)
```




```
stacking_classifier_model_train_perf = model_performance_classification_sklearn(stacking_classifier, X_train, y_train)
print("Training performance \n",stacking_classifier_model_train_perf)
```

```
Training performance
   Accuracy    Recall  Precision        F1
0  0.759699  0.872996   0.789494  0.829148
```

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

```
Testing performance
   Accuracy    Recall  Precision        F1
0  0.738749  0.859941   0.773977  0.814698
```

Model Performance Comparison and Conclusions

```
# 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:

	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
Accuracy	1.0	0.706773	0.983797	0.667919	0.999944	0.783135	0.738058	0.667919	0.757849	0.753756	0.840884	0.759475	0.759699
Recall	1.0	0.960883	0.984639	1.000000	0.999916	0.954168	0.886259	1.000000	0.883657	0.885671	0.930664	0.892470	0.872996
Precision	1.0	0.706125	0.991044	0.667919	1.000000	0.773844	0.760937	0.667919	0.782095	0.776894	0.846400	0.779415	0.789494