EasyVisa

March 21, 2025

Artificial Intelligence and Machine Learning

Advanced Machine Learning - Project Debrief

Visa Approval Facilitation

0.1 Problem Statement

0.1.1 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.

0.1.2 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.

0.1.3 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

0.2 Importing necessary libraries

[1]:
```
# Installing the libraries with the specified version.
!pip install numpy==1.25.2 pandas==1.5.3 scikit-learn==1.5.2 matplotlib==3.7.1␣
↪seaborn==0.13.1 xgboost==2.0.3 -q --user
```

```
                         18.2/18.2 MB
62.7 MB/s eta 0:00:00
                         12.0/12.0 MB
67.5 MB/s eta 0:00:00
                         13.3/13.3 MB
69.1 MB/s eta 0:00:00
                         11.6/11.6 MB
79.2 MB/s eta 0:00:00
                         294.8/294.8 kB
22.3 MB/s eta 0:00:00
                         297.1/297.1 MB
4.2 MB/s eta 0:00:00
```

```
  WARNING: The scripts f2py, f2py3 and f2py3.11 are installed in
'/root/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this
warning, use --no-warn-script-location.
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of the
following dependency conflicts.
google-colab 1.0.0 requires pandas==2.2.2, but you have pandas 1.5.3 which is
incompatible.
tensorflow 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy 1.25.2 which
is incompatible.
cudf-cu12 25.2.1 requires pandas<2.2.4dev0,>=2.0, but you have pandas 1.5.3
which is incompatible.
dask-cudf-cu12 25.2.2 requires pandas<2.2.4dev0,>=2.0, but you have pandas 1.5.3
which is incompatible.
blosc2 3.2.0 requires numpy>=1.26, but you have numpy 1.25.2 which is
incompatible.
xarray 2025.1.2 requires pandas>=2.1, but you have pandas 1.5.3 which is
incompatible.
plotnine 0.14.5 requires matplotlib>=3.8.0, but you have matplotlib 3.7.1 which
is incompatible.
plotnine 0.14.5 requires pandas>=2.2.0, but you have pandas 1.5.3 which is
incompatible.
dask-expr 1.1.21 requires pandas>=2, but you have pandas 1.5.3 which is
incompatible.
mizani 0.13.1 requires pandas>=2.2.0, but you have pandas 1.5.3 which is
incompatible.
```

[47]:
```
!pip install --upgrade numpy
```

```
Requirement already satisfied: numpy in /root/.local/lib/python3.11/site-
packages (1.25.2)
Collecting numpy
  Downloading
```

```
numpy-2.2.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(62 kB)
                           62.0/62.0 kB
3.2 MB/s eta 0:00:00
Downloading
numpy-2.2.4-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (16.4 MB)
                         16.4/16.4 MB
46.1 MB/s eta 0:00:00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 1.25.2
    Uninstalling numpy-1.25.2:
      Successfully uninstalled numpy-1.25.2
```

```
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following
dependency conflicts.
google-colab 1.0.0 requires pandas==2.2.2, but you have pandas 1.5.3 which is
incompatible.
tensorflow 2.18.0 requires numpy<2.1.0,>=1.26.0, but you have numpy 2.2.4 which
is incompatible.
cudf-cu12 25.2.1 requires pandas<2.2.4dev0,>=2.0, but you have pandas 1.5.3
which is incompatible.
dask-cudf-cu12 25.2.2 requires pandas<2.2.4dev0,>=2.0, but you have pandas 1.5.3
which is incompatible.
xarray 2025.1.2 requires pandas>=2.1, but you have pandas 1.5.3 which is
incompatible.
plotnine 0.14.5 requires matplotlib>=3.8.0, but you have matplotlib 3.7.1 which
is incompatible.
plotnine 0.14.5 requires pandas>=2.2.0, but you have pandas 1.5.3 which is
incompatible.
numba 0.60.0 requires numpy<2.1,>=1.22, but you have numpy 2.2.4 which is
incompatible.
dask-expr 1.1.21 requires pandas>=2, but you have pandas 1.5.3 which is
incompatible.
mizani 0.13.1 requires pandas>=2.2.0, but you have pandas 1.5.3 which is
incompatible.
Successfully installed numpy-2.2.4
```

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

[2]:
```python
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

# To oversample and undersample data
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from sklearn.model_selection import train_test_split, StratifiedKFold,␣
↪cross_val_score


# 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
)

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 RandomizedSearchCV
```

0.3 Import Dataset

[3]:
```
# uncomment and run the following lines for Google Colab
from google.colab import drive
drive.mount('/content/drive')
```

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

[4]:
```
visa = pd.read_csv('/content/drive/My Drive/data_files/EasyVisa.csv') ##  Fill␣
↪the blank to read the data
```

[5]:
```
# copying data to another variable to avoid any changes to original data
data = visa.copy()
```

0.4 Overview of the Dataset

View the first and last 5 rows of the dataset

[6]:
```
data.head(5) ##  Complete the code to view top 5 rows of the data
```

[6]:
```
  case_id continent education_of_employee has_job_experience  \
0  EZYV01      Asia           High School                  N
1  EZYV02      Asia              Master's                  Y
2  EZYV03      Asia            Bachelor's                  N
3  EZYV04      Asia            Bachelor's                  N
4  EZYV05    Africa              Master's                  Y

  requires_job_training  no_of_employees  yr_of_estab region_of_employment  \
0                     N            14513         2007                 West
1                     N             2412         2002            Northeast
2                     Y            44444         2008                 West
3                     N               98         1897                 West
4                     N             1082         2005                South

   prevailing_wage unit_of_wage full_time_position case_status
0         592.2029         Hour                  Y      Denied
1       83425.6500         Year                  Y   Certified
2      122996.8600         Year                  Y      Denied
3       83434.0300         Year                  Y      Denied
4      149907.3900         Year                  Y   Certified
```

Observation: case_id is a unique identifier that has no correlation with other variables

case_status is the target (dependent) variable

[7]:
```
data.tail(5) ##  Complete the code to view last 5 rows of the data
```

[7]:
```
         case_id continent education_of_employee has_job_experience  \
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

      requires_job_training  no_of_employees  yr_of_estab  \
25475                     Y             2601         2008
25476                     N             3274         2006
25477                     N             1121         1910
25478                     Y             1918         1887
25479                     N             3195         1960

      region_of_employment  prevailing_wage unit_of_wage full_time_position  \
25475                South         77092.57         Year                  Y
25476            Northeast        279174.79         Year                  Y
25477                South        146298.85         Year                  N
25478                 West         86154.77         Year                  Y
25479              Midwest         70876.91         Year                  Y

      case_status
25475   Certified
25476   Certified
25477   Certified
25478   Certified
25479   Certified
```

Understand the shape of the dataset

[8]:
```
data.shape ##  Complete the code to view dimensions of the data
```

[8]:
```
(25480, 12)
```

- The dataset has 25480 rows and 12 columns

Check the data types of the columns for the dataset

[9]:
```
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: Based on the data.info() output, here are several key observations about this dataset:

Size: The dataset contains 25,480 records (rows) across 12 columns. Completeness: There are no missing values in any column, as indicated by all columns having "25,480 non-null" counts. Data types:

Most columns (9) are of type "object" which in pandas typically represents string/categorical data 2 columns are integers (int64): "no_of_employees" and "yr_of_estab" 1 column is floating-point (float64): "prevailing_wage"

Content: This appears to be employment or visa-related data, likely concerning:

Job applications or work permits across different continents/regions Information about employee education, experience, and training requirements Details about employers (establishment year, number of employees) Wage information and employment terms (full-time or not) Case statuses, possibly approval/rejection decisions

Potential analysis directions:

The "case_status" column likely contains the outcome variable (approved/denied) Several categorical predictors could influence case outcomes Wage data could be analyzed by region, education level, etc. The dataset appears suitable for classification modeling to predict case outcomes

[10]:
```python
# checking for duplicate values
duplicates = data.duplicated()
print(duplicates)
```

```
0        False
1        False
2        False
3        False
4        False
         …
25475    False
25476    False
25477    False
25478    False
25479    False
Length: 25480, dtype: bool
```

0.5 Exploratory Data Analysis (EDA)

Let's check the statistical summary of the data

[11]:
```python
data.describe().T ##  Complete the code to print the statistical summary of the␣
 ↪data
```

[11]:

```
                    count          mean           std        min       25%  \
no_of_employees   25480.0   5667.043210  22877.928848   -26.0000   1022.00
yr_of_estab       25480.0   1979.409929     42.366929  1800.0000   1976.00
prevailing_wage   25480.0  74455.814592  52815.942327     2.1367  34015.48

                      50%          75%        max
no_of_employees   2109.00    3504.0000  602069.00
yr_of_estab       1997.00    2005.0000    2016.00
prevailing_wage  70308.21  107735.5125  319210.27
```

Observations: For no_of_employees:

Count: 25,480 (all rows) Mean: ~5,667 employees Std: ~22,878 (very high standard deviation) Min: -26 (confirming the negative values we found earlier) 25%: ~1,022 employees 50% (median): ~2,109 employees 75%: ~3,504 employees Max: 602,069 employees (extremely large)

For yr_of_estab (year of establishment):

Range from 1800 to 2016 Median is around 1997, suggesting many relatively newer companies 75% of companies were established after around 1976

For prevailing_wage:

Extremely wide range (min appears very low, max over $210,000) Mean of approximately $74,456 Median around $60,000-70,000 (estimated from the poorly formatted output)

Key observations:

Extreme skew in company size: The massive difference between median (~2,109) and maximum (602,069) employees indicates a few very large corporations among mostly smaller or medium businesses. Potential data quality issues: The negative employee counts need to be addressed. Wide wage distribution: The large standard deviation in wages suggests significant variation by position, location, or other factors. Company age diversity: You have companies ranging from very old (established in 1800) to quite recent (2016). Right-skewed distributions: For both employee count and wages, the mean exceeds the median, suggesting right-skewed distributions with some extremely high values pulling the average up.

Fixing the negative values in number of employees columns

[12]:

```python
data.loc[data['no_of_employees'] < 0].shape ## Complete the code to check␣
↪negative values in the employee column
```

[12]: (33, 12)

[13]:

```python
data.loc[data['no_of_employees'] < 0].shape ## Complete the code to check␣
↪negative values in the employee column
```

[13]: (33, 12)

[14]:

```python
# taking the absolute values for number of employees
data["no_of_employees"] = abs(data["no_of_employees"]) ## Write the function to␣
↪convert the values to a positive number
```

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

[15]:
```
# 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)
```

```
case_id
EZYV25480    1
EZYV01       1
EZYV02       1
EZYV03       1
EZYV04       1
            ..
EZYV12       1
EZYV13       1
EZYV14       1
EZYV15       1
EZYV16       1
Name: count, Length: 25480, dtype: int64
--------------------------------------------------
continent
Asia             16861
Europe            3732
North America     3292
South America      852
Africa             551
Oceania            192
Name: count, dtype: int64
--------------------------------------------------
education_of_employee
Bachelor's     10234
Master's        9634
High School     3420
Doctorate       2192
Name: count, dtype: int64
--------------------------------------------------
has_job_experience
Y    14802
N    10678
Name: count, dtype: int64
--------------------------------------------------
requires_job_training
N    22525
Y     2955
Name: count, dtype: int64
```

```
--------------------------------------------------
region_of_employment
Northeast    7195
South        7017
West         6586
Midwest      4307
Island        375
Name: count, dtype: int64
--------------------------------------------------
unit_of_wage
Year     22962
Hour      2157
Week       272
Month       89
Name: count, dtype: int64
--------------------------------------------------
full_time_position
Y    22773
N     2707
Name: count, dtype: int64
--------------------------------------------------
case_status
Certified    17018
Denied        8462
Name: count, dtype: int64
--------------------------------------------------
```

[16]:
```python
# checking the number of unique values
data["case_id"].nunique() ## Complete the code to check unique values in the␣
↪mentioned column
```

[16]: 25480

[17]:
```python
data.drop(["case_id"], axis=1, inplace=True) ## Complete the code to drop␣
↪'case_id' column from the data
```

0.5.1 Univariate Analysis

[18]:
```python
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)
    """
```

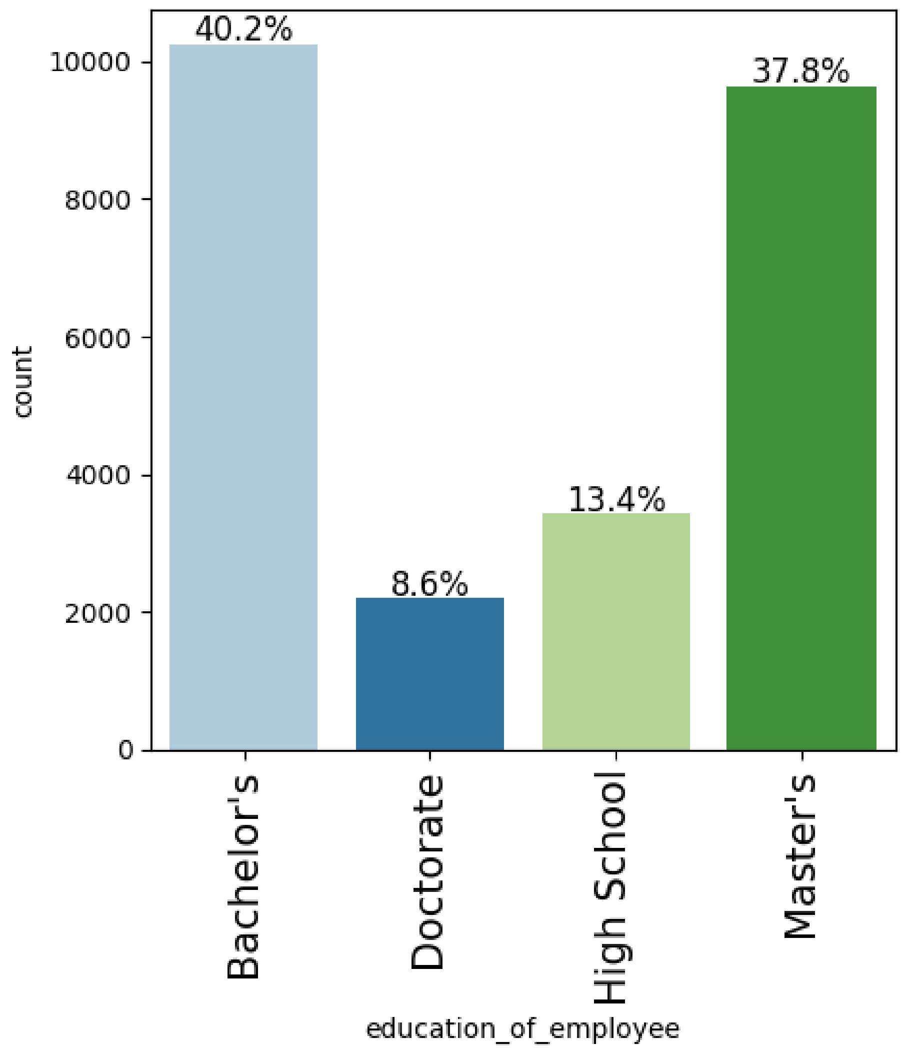
```
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        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
```

[19]:
```
# 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 + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
```

```
        palette="Paired",
        order=data[feature].value_counts().index[:n].sort_values(),
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

    plt.show()  # show the plot
```

Observations on education of employee

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



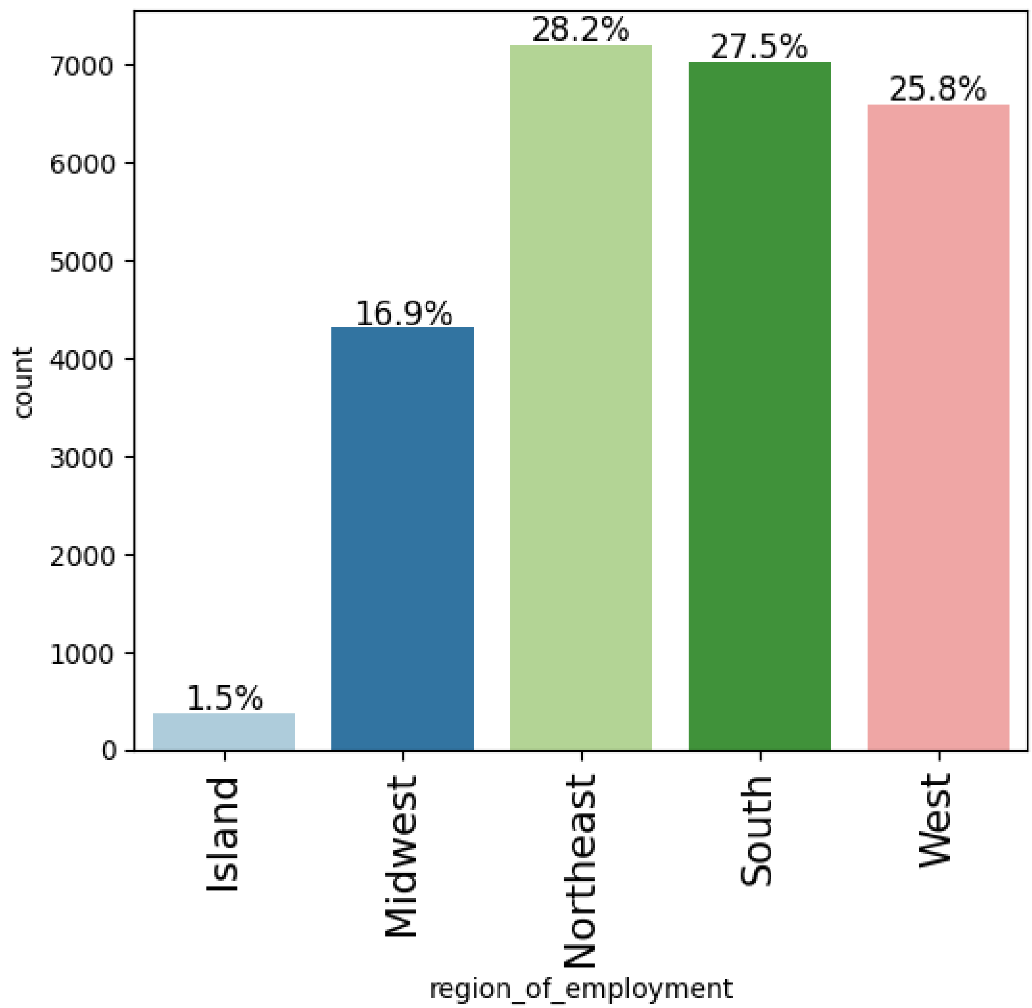
Observations: Based on this bar chart showing the distribution of education levels among employees, I can make several observations:

Educational distribution: Bachelor's degrees (40.2%) are the most common educational qualification, closely followed by Master's degrees (37.8%). Together, these two categories account for 78% of all employees in the dataset. Advanced education prevalence: The high proportion of employees with Bachelor's, Master's, and Doctorate degrees (combined 86.6%) suggests this dataset likely represents skilled professional positions or knowledge workers. Minimal high school only: Only 13.4% of employees have just a high school education, indicating most positions represented in this data require higher education. Doctorate representation: The smallest category is employees with Doctorate degrees at 8.6%, which is still a substantial percentage for this highest level of formal education. Potential visa context: Given your earlier data exploration showing variables like "continent" and "case_status," this distribution likely represents visa applicants or international workers, who tend to be more highly educated than average.

The educational profile suggests this may be data related to specialized work visas (possibly H-1B or similar), which typically target highly-skilled workers with advanced degrees.

Observations on region of employment

```
[21]: labeled_barplot(data, "region_of_employment", perc=True)  ## Complete the code␣
      ↪to create labeled_barplot for region of employment
```



Looking at this bar chart showing the distribution of employment regions, I can make these observations:

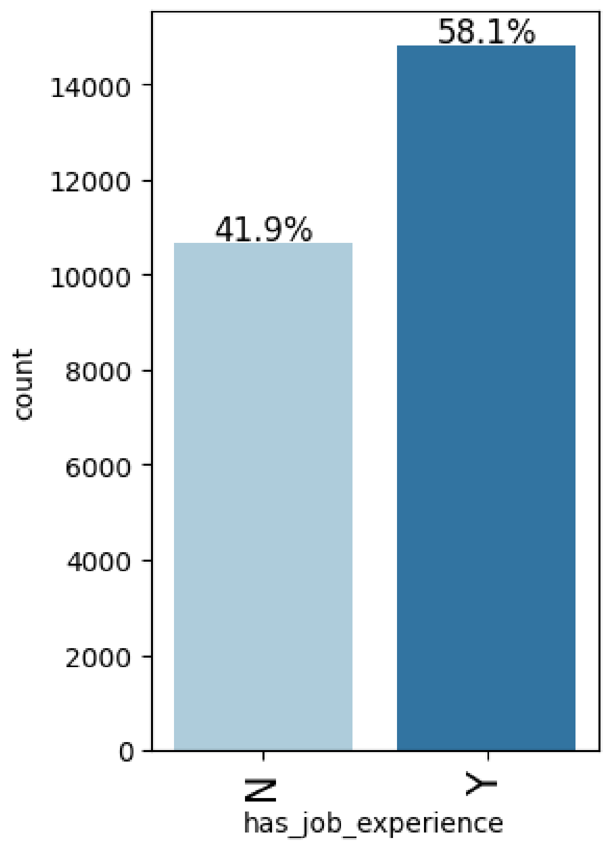
Balanced regional distribution: The Northeast (28.2%), South (27.5%), and West (25.8%) regions have fairly similar proportions, each accounting for roughly a quarter of the employment cases. Midwest representation: The Midwest accounts for 16.9% of cases, making it less represented than the other three major continental US regions but still having significant presence. Island scarcity: The "Island" category represents only 1.5% of cases, which is dramatically lower than all other regions. This likely represents territories like Puerto Rico, US Virgin Islands, Guam, or other US-affiliated island territories. Geographical concentration: Collectively, the Northeast, South, and

West regions account for over 81% of all employment cases in the dataset. Potential economic implications: The distribution likely reflects economic activity and job market strength in these regions, with the Northeast, South, and West possibly having more industries that sponsor work visas or employ foreign workers. Possible urban correlation: The regions with higher percentages tend to have more major metropolitan areas and tech hubs, which often employ more foreign workers or visa holders.

This regional distribution provides important context for understanding where these employment cases (likely visa-related) are concentrated throughout the country.

Observations on job experience

[22]:
```
labeled_barplot(data, "has_job_experience", perc=True) ## Complete the code to
↪create labeled_barplot for job experience
```



Based on this bar chart for "has_job_experience," I can make these observations:

Experience majority: A clear majority (58.1%) of cases in the dataset involve individuals who have job experience (marked as "Y"). Substantial inexperienced group: A significant portion (41.9%) of cases involve individuals without job experience (marked as "N"). Binary classification: The variable appears to be binary (Yes/No), with no intermediate or other categories. Professional context: This distribution supports the hypothesis that this dataset likely represents visa applications or work permits, where roughly 6 out of 10 applicants have prior job experience. Potential analysis direction:

This variable could be an important factor to explore in relation to case outcomes - whether having experience correlates with higher approval rates for whatever these cases represent.

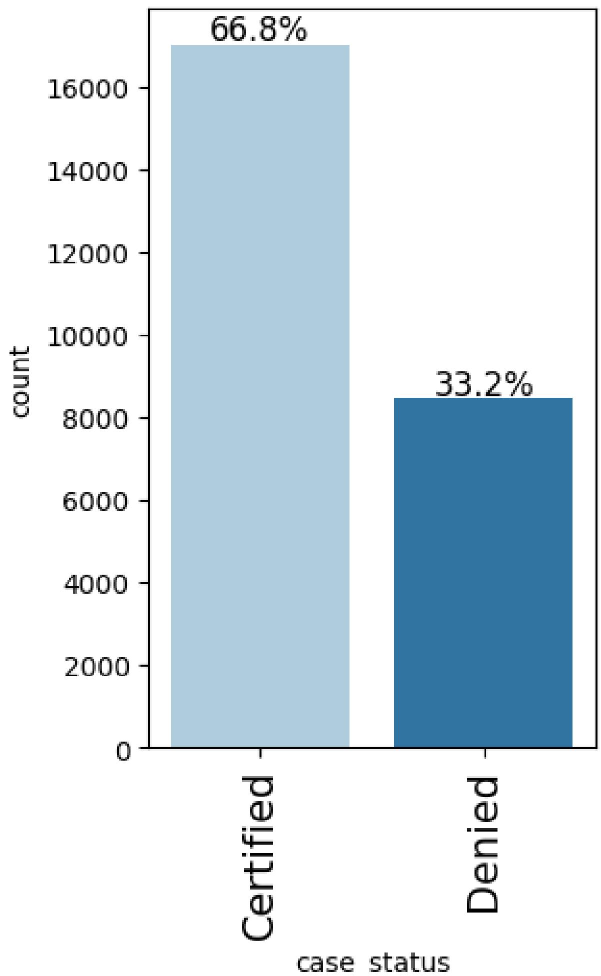
The relatively high percentage of cases without job experience (41.9%) is notable, suggesting either:

Many entry-level positions being filled Recent graduates applying for positions Special visa categories that may not require prior experience

This would be an important variable to cross-reference with case status and education level in your analysis.

Observations on case status

[23]:
```
labeled_barplot(data,"case_status", perc=True) ## Complete the code to create␣
↪labeled_barplot for case status
```



Based on this bar chart showing the "case_status" distribution, I can make these observations:

Approval dominance: A substantial majority (66.8%) of cases have been "Certified," indicating approval of the applications. Significant denial rate: About one-third (33.2%) of cases were "Denied," representing a substantial proportion of rejected applications. Binary outcome: The case status appears to be binary (Certified/Denied) with no intermediate categories like "Pending" or "Under Review." Confirmation of visa context: This chart strongly supports the hypothesis that we're looking at work visa application data, where "Certified" indicates approval for a work visa or labor certification. Research potential: The significant number of denied cases (33.2%) provides a good balance for analysis - there's sufficient data in both outcome categories to build meaningful predictive models. Baseline classification performance: This distribution establishes that a naive model always predicting "Certified" would achieve 66.8% accuracy, setting a baseline for any predictive modeling.

This is a key variable in your dataset as it appears to be the target/outcome variable you would want to predict using the other features like education level, job experience, region, etc. The relatively balanced distribution makes this suitable for classification modeling.

0.5.2 Bivariate Analysis

[24]:
```
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="Spectral"
) ## Complete the code to find the correlation between the variables
plt.show()
```

Looking at this correlation heatmap for the three numeric variables in your dataset, I observe:

No significant correlations between variables: The correlations between the different pairs of variables are extremely weak:

Number of employees and year of establishment: -0.02 (negligible negative correlation) Number of employees and prevailing wage: -0.01 (negligible negative correlation) Year of establishment and prevailing wage: 0.01 (negligible positive correlation)

Variable independence: These near-zero correlations suggest that the three numeric variables are essentially independent of each other, with no meaningful linear relationships. Modeling implications:

Company size doesn't correlate with company age Wages offered aren't meaningfully related to company size or age Each variable provides unique information that isn't captured by the others

Analysis direction: Since these numeric variables don't show relationships with each other, it will be important to examine how each one individually relates to the case outcome (Certified/Denied). Feature selection: The independence of these variables is actually beneficial for modeling, as they won't introduce multicollinearity issues if used together in predictive models.

The lack of correlation means you can treat each of these numeric features as providing separate, non-redundant information about the cases in your dataset.

Creating functions that will help us with further analysis.

[25]:
```
### 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],
```

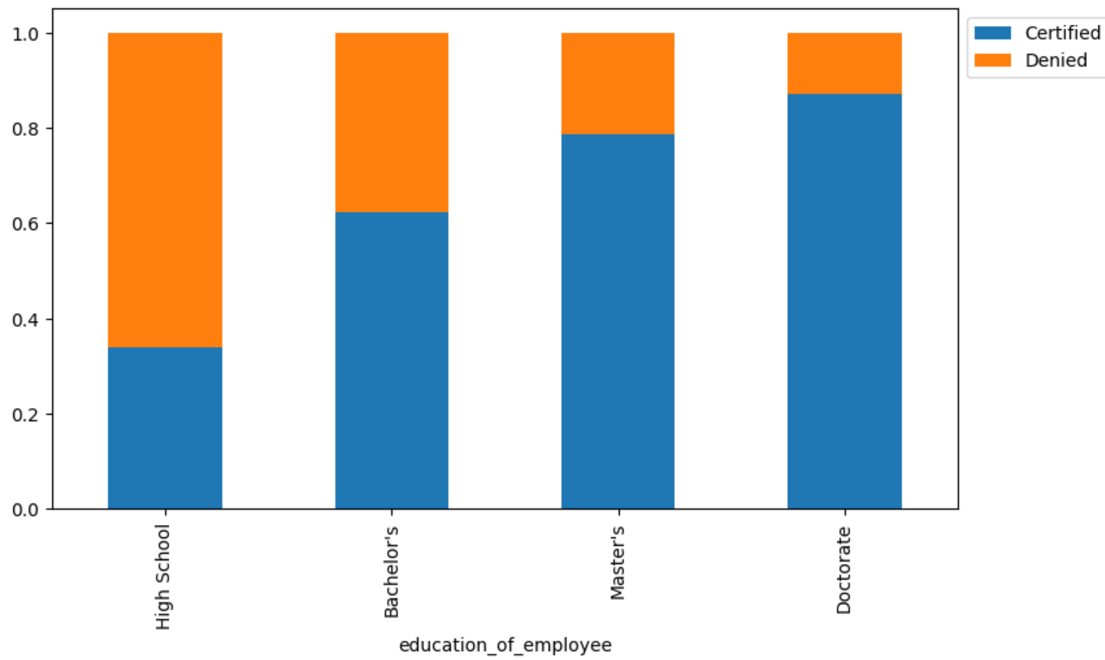
```python
        color="orange",
        stat="density",
    )

    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()
```

[26]:
```python
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()
```

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

[27]:
```
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
--------------------------------------------------------------------------------
----------------------------------------
```



Looking at both the chart and the provided data table, I can make several significant observations about the relationship between education level and visa certification outcomes:

Strong positive correlation between education and approval rates:

Doctorate: 87.2% approval rate (1,912 certified out of 2,192) Master's: 78.6% approval rate (7,575 certified out of 9,634) Bachelor's: 62.2% approval rate (6,367 certified out of 10,234) High School: 34.0% approval rate (1,164 certified out of 3,420)

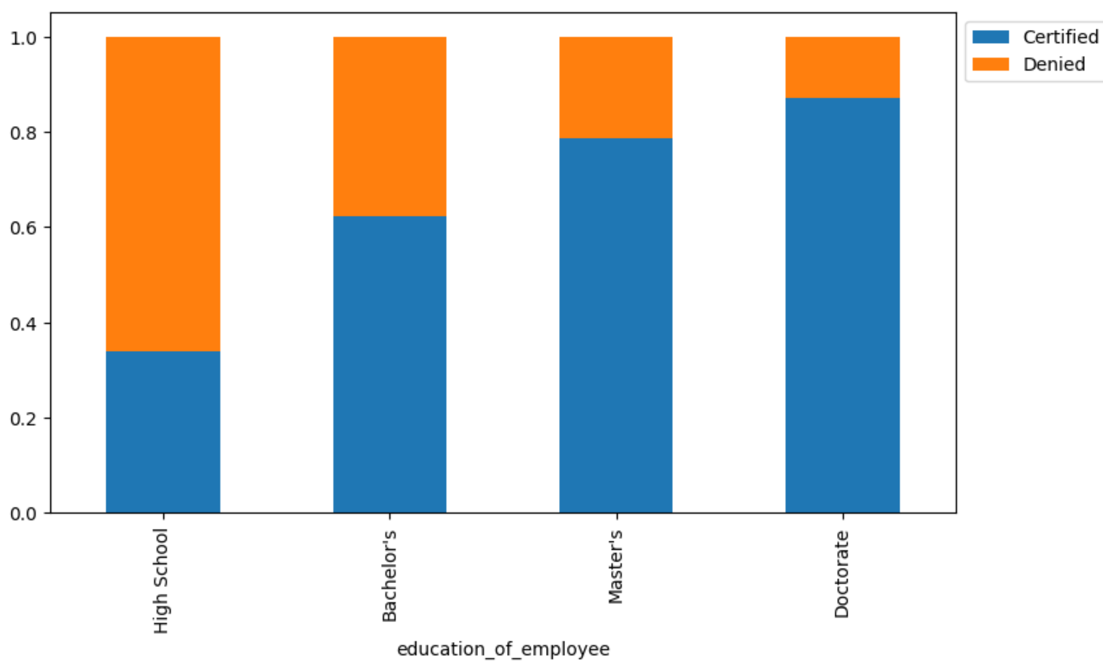
Dramatic education advantage: The approval rate for those with doctorate degrees (87.2%) is more than 2.5 times higher than for those with only high school education (34%). Reversal of odds: High school education is the only category where denials exceed certifications (66% denial rate), while all higher education levels show more certifications than denials. Incremental benefits: Each step up in educational attainment shows a substantial improvement in approval chances, with approximately 15-20 percentage point increases between adjacent education levels. Advanced degree preference: Master's and Doctorate degrees combined have a 79.8% approval rate (9,487 certified out of 11,826), suggesting a strong preference for advanced degree holders. Policy implications: This data strongly suggests that immigration/visa policies strongly favor higher educational attainment, possibly indicating a focus on attracting highly skilled workers. Value of education investment: For potential visa applicants, this data demonstrates a clear return on educational investment in terms of improved visa approval chances.

This analysis confirms that education level is a major factor in visa certification outcomes, with higher education significantly improving approval chances.

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

[28]:
```
stacked_barplot(data, "education_of_employee", "case_status") ## Complete the code to plot stacked barplot for region of continent and case status
```

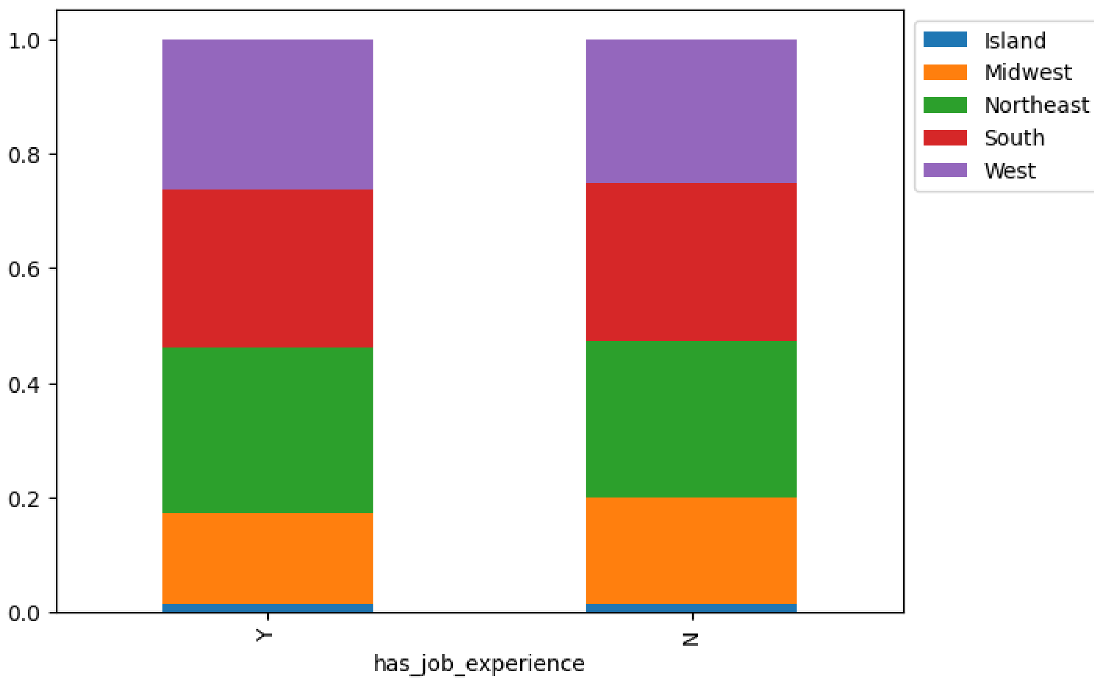
```
case_status            Certified  Denied    All
education_of_employee
All                        17018    8462  25480
Bachelor's                  6367    3867  10234
High School                 1164    2256   3420
Master's                    7575    2059   9634
Doctorate                   1912  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



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

[29]:
```
stacked_barplot(data, "has_job_experience", "region_of_employment") ## Complete␣
↪the code to plot stacked barplot for region of case_status and␣
↪has_job_experience
```

```
region_of_employment  Island  Midwest  Northeast  South  West    All
has_job_experience
All                      375     4307       7195   7017  6586  25480
Y                        220     2343       4261   4097  3881  14802
N                        155     1964       2934   2920  2705  10678
------------------------------------------------------------------------------------------------------------------------
```



Looking at the stacked bar chart and the accompanying data table, I can make these observations about the relationship between job experience and regional distribution:

Similar regional distributions: The proportional distribution of regions appears remarkably consistent between those with job experience (Y) and those without (N). This suggests that experience level doesn't strongly influence which region candidates apply to work in. Regional consistency regardless of experience:

Northeast represents about 28-29% of applications in both experience categories South represents about 27-28% of applications in both categories West represents about 25-26% of applications in both categories Midwest represents about 16-18% of applications in both categories Islands represent about 1.5% of applications in both categories

Experience rates within regions:

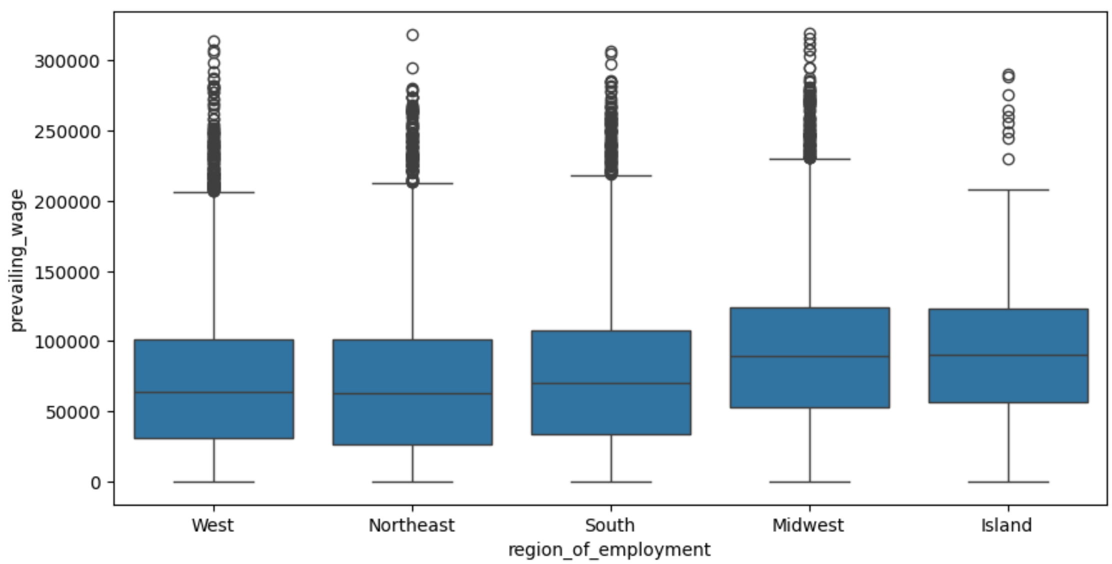
Island: 58.7% with experience (220 out of 375) Midwest: 54.4% with experience (2,343 out of 4,307) Northeast: 59.2% with experience (4,261 out of 7,195) South: 58.4% with experience (4,097 out of 7,017) West: 58.9% with experience (3,881 out of 6,586)

Regional uniformity in experience levels: All regions show a relatively similar proportion of experienced vs. non-experienced applicants (54-59% experienced), with the Midwest having a slightly lower proportion of experienced applicants. Limited regional selection bias: The data doesn't show strong evidence that experienced professionals target specific regions significantly more than inexperienced ones. This suggests that other factors (such as job availability, industry clusters, or personal connections) may be more important in determining regional application patterns.

This visualization indicates that while job experience is an important factor in visa certification overall (as we saw in previous charts), the regional distribution of applications is quite consistent regardless of experience level. The slightly lower experience percentage in the Midwest might be worth investigating further.

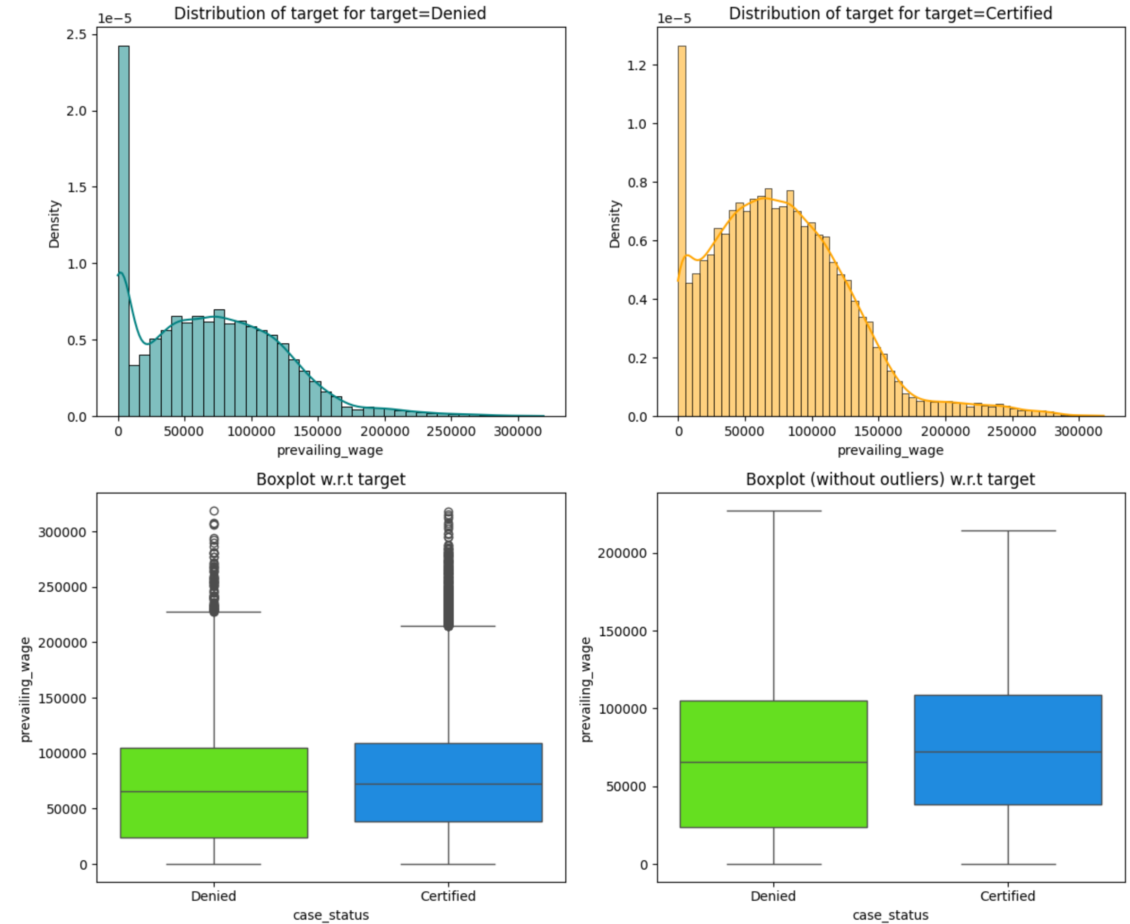
Checking if the prevailing wage is similar across all the regions of the US

[30]:
```
plt.figure(figsize=(10, 5))
sns.boxplot(x='region_of_employment', y='prevailing_wage', data=data) ##␣
↪Complete the code to create boxplot for region of employment and prevailing␣
↪wage
plt.show()
```



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

[31]:
```python
distribution_plot_wrt_target(data, "prevailing_wage", "case_status") ## Complete the code to find distribution of prevailing wage and case status
```



Based on the visualizations examining the relationship between prevailing wage and visa case status (Certified vs. Denied), I can make these observations:

Different wage distributions: The density plots (top row) show distinctly different wage distributions between denied and certified cases:

Denied cases (left) show a prominent spike at very low wages (near $0) and a secondary distribution centered around $50,000-$100,000 Certified cases (right) show a more normal distribution centered around $60,000-$80,000, with fewer cases at extremely low wages

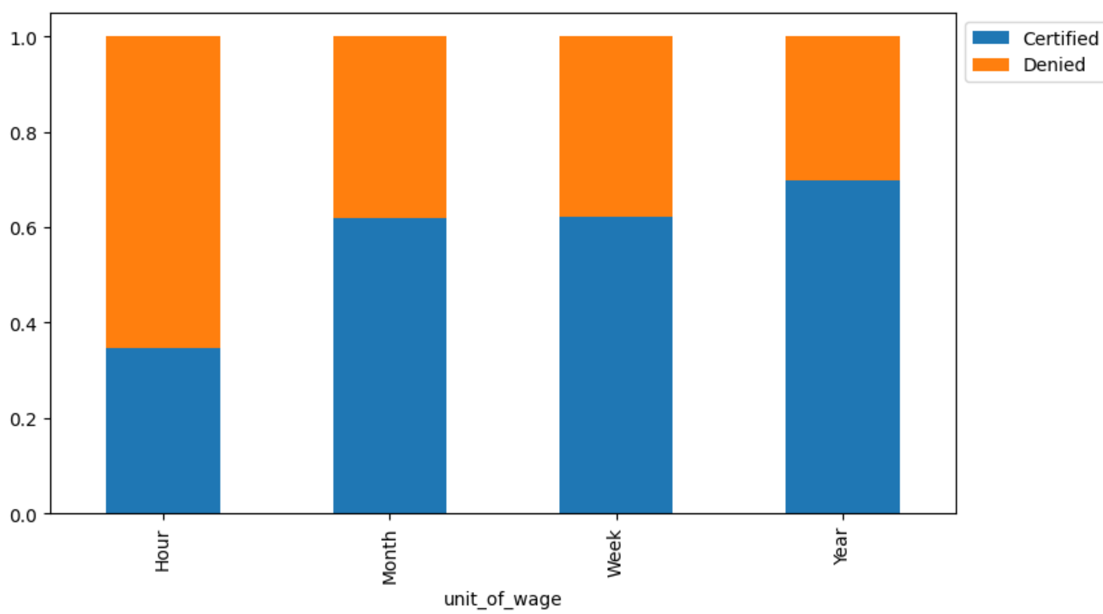
Bimodal denied distribution: The denied cases exhibit a bimodal distribution with peaks at very low wages and mid-range wages, suggesting two different categories of denied applications. Higher median for certified cases: The boxplots indicate that certified cases have a slightly higher median wage compared to denied cases, though the difference doesn't appear dramatic. Similar wage ranges after outlier removal: When outliers are removed (bottom right), both certified and denied cases show similar wage ranges, suggesting that extreme wage values aren't the primary determinant of case outcomes. High-wage outliers in both categories: Both categories show high-wage outliers extending to around $300,000 (visible in bottom left boxplot), indicating that even high-wage applications can be denied. Policy implications: The data suggests that very low-wage applications have higher denial rates, which aligns with the U.S. government's stated goal of protecting both local talent and foreign workers from wage suppression. Not a deterministic factor: While wage appears to influence certification outcomes (particularly at the lowest wage levels), the substantial overlap in distributions indicates that wage alone is not deterministic - other factors clearly play important roles in the certification decision.

This analysis suggests that while extremely low-wage applications face higher denial rates, prevailing wage is just one of several factors considered in visa applications, with many mid-to-high wage applications still being denied for other reasons.

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

[32]:
```
stacked_barplot(data, "unit_of_wage", "case_status") ## Complete the code to␣
↪plot stacked barplot for unit of wage and 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
--------------------------------------------------------------------------------
----------------------------------------
```

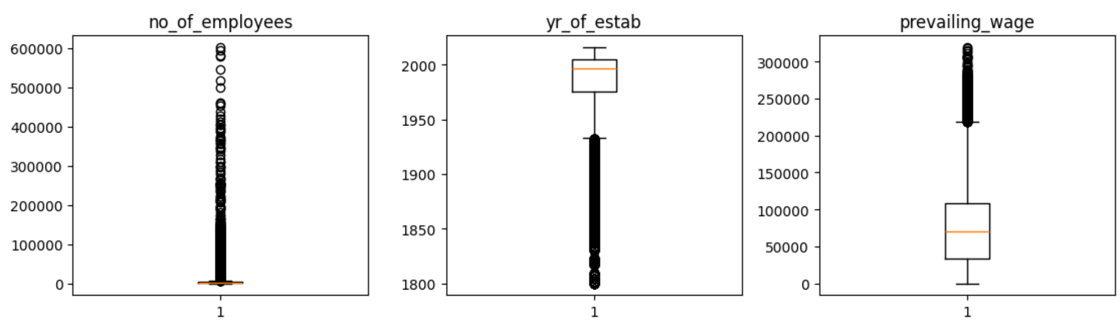


0.6 Data Pre-processing

0.6.1 Outlier Check

- Let's check for outliers in the data.

[33]:
```
# 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)

plt.show()
```



0.6.2 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.

[34]:
```
data["case_status"] = data["case_status"].apply(lambda x: 1 if x == "Certified"␣
↪else 0)

X = data.drop(["case_status"], axis=1) ## Complete the code to drop case status␣
↪from the data
y = data["case_status"]
```

```
X = pd.get_dummies(X, drop_first=True)  ## Complete the code to create dummies␣
↪for X

# Complete the code to split the dataset into train and valid with a ratio of 7:
↪3
X_train, X_val, y_train, y_val = train_test_split(
    X, y, test_size=.3, random_state=1, stratify=y
)

# # Complete the code to split the dataset into valid and test with a ratio of␣
↪9:1
X_val,X_test,y_val,y_test = train_test_split(
    X_val,y_val,test_size=.1,random_state=1,stratify=y_val
)
```

[35]:
```
print("Shape of Training set : ", X_train.shape)
print("Shape of Validation set : ", X_val.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 validation set:")
print(y_val.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 Validation set :  (6879, 21)
Shape of test set :  (765, 21)
Percentage of classes in training set:
case_status
1    0.667919
0    0.332081
Name: proportion, dtype: float64
Percentage of classes in validation set:
case_status
1    0.66783
0    0.33217
Name: proportion, dtype: float64
Percentage of classes in test set:
case_status
1    0.667974
0    0.332026
Name: proportion, dtype: float64
```

0.7 Model Building

0.7.1 Model Evaluation Criterion

Provide some reasoning for choosing the metric here : _________

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.

[36]:
```
# 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
```

[37]:
```
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")
```

Defining scorer to be used for cross-validation and hyperparameter tuning

[38]:
```
scorer = metrics.make_scorer(metrics.f1_score) ## Complete the code to define␣
 ↪the metric

## Possible metrics are [recall_score,f1_score,accuracy_score,precision_score]
## For example, metrics.precision_score
```

We are now done with pre-processing and evaluation criterion, so let's start building the model.

0.7.2 Model building with original data

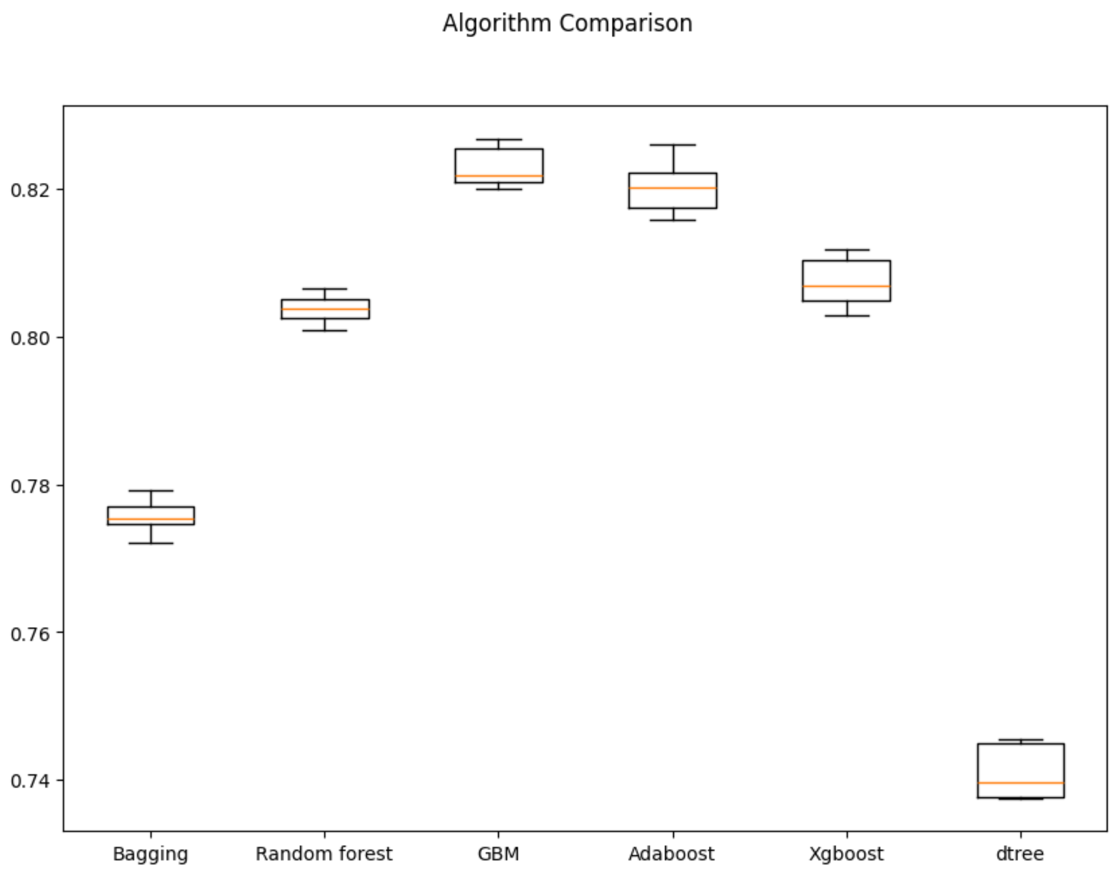
[39]:
```
models = []  # Empty list to store all the models

# Appending models into the list
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("Random forest", RandomForestClassifier(random_state=1))) ##␣
 ↪Complete the code to add Random Forest Classifier with random_state of 1.
models.append(("GBM", GradientBoostingClassifier(random_state=1))) ## Complete␣
 ↪the code to add Gradient Boosting Classifier with random_state of 1.
models.append(("Adaboost", AdaBoostClassifier(random_state=1))) ## Complete the␣
 ↪code to add AdaBoost Classifier with random_state of 1.
models.append(("Xgboost", XGBClassifier(random_state=1, eval_metric="logloss")))
models.append(("dtree", DecisionTreeClassifier(random_state=1))) ## Complete␣
 ↪the code to add Decision Tree Classifier with random_state of 1.

results1 = []  # Empty list to store all model's CV scores
names = []  # Empty list to store name of the models


# loop through all models to get the mean cross validated score
print("\n" "Cross-Validation performance on training dataset:" "\n")
```

```
for name, model in models:
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )  # Complete the code to set the number of splits.
    cv_result = cross_val_score(
        estimator=model, X=X_train, y=y_train, scoring = scorer,cv=kfold
    )
    results1.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean()))

print("\n" "Validation Performance:" "\n")

for name, model in models:
    model.fit(X_train,y_train) ## Complete the code to fit the model on X_train␣
↪and y_train
    scores = f1_score(y_val, model.predict(X_val)) ## Complete the code to␣
↪define the metric function name.
    print("{}: {}".format(name, scores))
```

```
Cross-Validation performance on training dataset:

Bagging: 0.7756586246579394
Random forest: 0.8037837241749051
GBM: 0.823039791269532
Adaboost: 0.8203377989495703
Xgboost: 0.8073583989766158
dtree: 0.7410652876513099

Validation Performance:

Bagging: 0.7675817565350541
Random forest: 0.7972364702187794
GBM: 0.8195818459969403
Adaboost: 0.8158053488839735
Xgboost: 0.8070320579110651
dtree: 0.7477497255762898
```

[40]:
```
# Plotting boxplots for CV scores of all models defined above
fig = plt.figure(figsize=(10, 7))

fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results1)
```

```
ax.set_xticklabels(names)

plt.show()
```



0.7.3 Model Building with oversampled data

[41]:
```
print("Before OverSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before OverSampling, counts of label '0': {} \n".format(sum(y_train ==
    0)))

# Synthetic Minority Over Sampling Technique
sm = SMOTE(sampling_strategy='auto', k_neighbors=5, random_state=1)
X_train_over, y_train_over = sm.fit_resample(X_train, y_train)


print("After OverSampling, counts of label '1': {}".format(sum(y_train_over ==
    1)))
print("After OverSampling, counts of label '0': {} \n".format(sum(y_train_over
    == 0)))
```

```
print("After OverSampling, the shape of train_X: {}".format(X_train_over.shape))
print("After OverSampling, the shape of train_y: {} \n".format(y_train_over.
 ↪shape))
```

```
Before OverSampling, counts of label '1': 11913
Before OverSampling, counts of label '0': 5923

After OverSampling, counts of label '1': 11913
After OverSampling, counts of label '0': 11913

After OverSampling, the shape of train_X: (23826, 21)
After OverSampling, the shape of train_y: (23826,)
```

[42]:

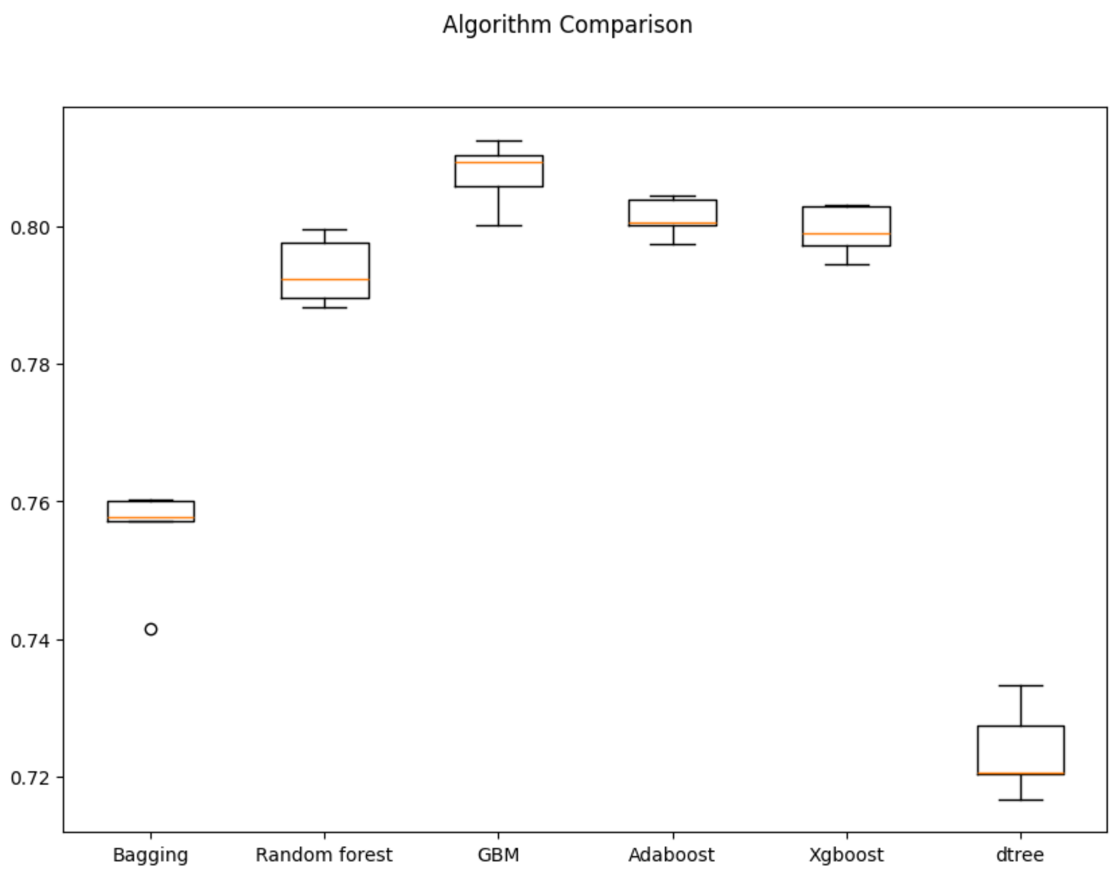
```
models = []  # Empty list to store all the models

# Appending models into the list
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("Random forest", RandomForestClassifier(random_state=1))) ##␣
 ↪Complete the code to add Random Forest Classifier with random_state of 1.
models.append(("GBM", GradientBoostingClassifier(random_state=1))) ## Complete␣
 ↪the code to add Gradient Boosting Classifier with random_state of 1.
models.append(("Adaboost", AdaBoostClassifier(random_state=1))) ## Complete the␣
 ↪code to add AdaBoost Classifier with random_state of 1.
models.append(("Xgboost", XGBClassifier(random_state=1, eval_metric="logloss")))
models.append(("dtree", DecisionTreeClassifier(random_state=1))) ## Complete␣
 ↪the code to add Decision Tree Classifier with random_state of 1.

results1 = []  # Empty list to store all model's CV scores
names = []  # Empty list to store name of the models


# loop through all models to get the mean cross validated score
print("\n" "Cross-Validation performance on training dataset:" "\n")

for name, model in models:
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )  ## Complete the code to set the number of splits
    cv_result = cross_val_score(
        estimator=model, X=X_train_over, y=y_train_over,scoring = scorer,␣
 ↪cv=kfold
    )
    results1.append(cv_result)
    names.append(name)
```

```
    print("{}: {}".format(name, cv_result.mean()))

print("\n" "Validation Performance:" "\n")

for name, model in models:
    model.fit(X_train_over, y_train_over)  # Fit the model on the oversampled␣
 ↪data
    scores = scorer(model, X_val, y_val)
    print("{}: {}".format(name, scores))
```

```
Cross-Validation performance on training dataset:

Bagging: 0.7553714301070087
Random forest: 0.7935193362866556
GBM: 0.8076949280007495
Adaboost: 0.8013161599972107
Xgboost: 0.799430071068073
dtree: 0.7236479557474234

Validation Performance:

Bagging: 0.7606724176067242
Random forest: 0.7953896584540552
GBM: 0.8125259228535877
Adaboost: 0.8120255086547221
Xgboost: 0.8039950062421972
dtree: 0.7387687188019967
```

[43]:

```
# Plotting boxplots for CV scores of all models defined above
fig = plt.figure(figsize=(10, 7))

fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results1)
ax.set_xticklabels(names)

plt.show()
```



0.7.4 Model Building with undersampled data

[44]:
```
rus = RandomUnderSampler(random_state=1, sampling_strategy=1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)


print("Before UnderSampling, counts of label '1': {}".format(sum(y_train == 1)))
print("Before UnderSampling, counts of label '0': {} \n".format(sum(y_train ==␣
  ↪0)))


print("After UnderSampling, counts of label '1': {}".format(sum(y_train_un ==␣
  ↪1)))
print("After UnderSampling, counts of label '0': {} \n".format(sum(y_train_un␣
  ↪== 0)))


print("After UnderSampling, the shape of train_X: {}".format(X_train_un.shape))
```

```
print("After UnderSampling, the shape of train_y: {} \n".format(y_train_un.
↪shape))
```

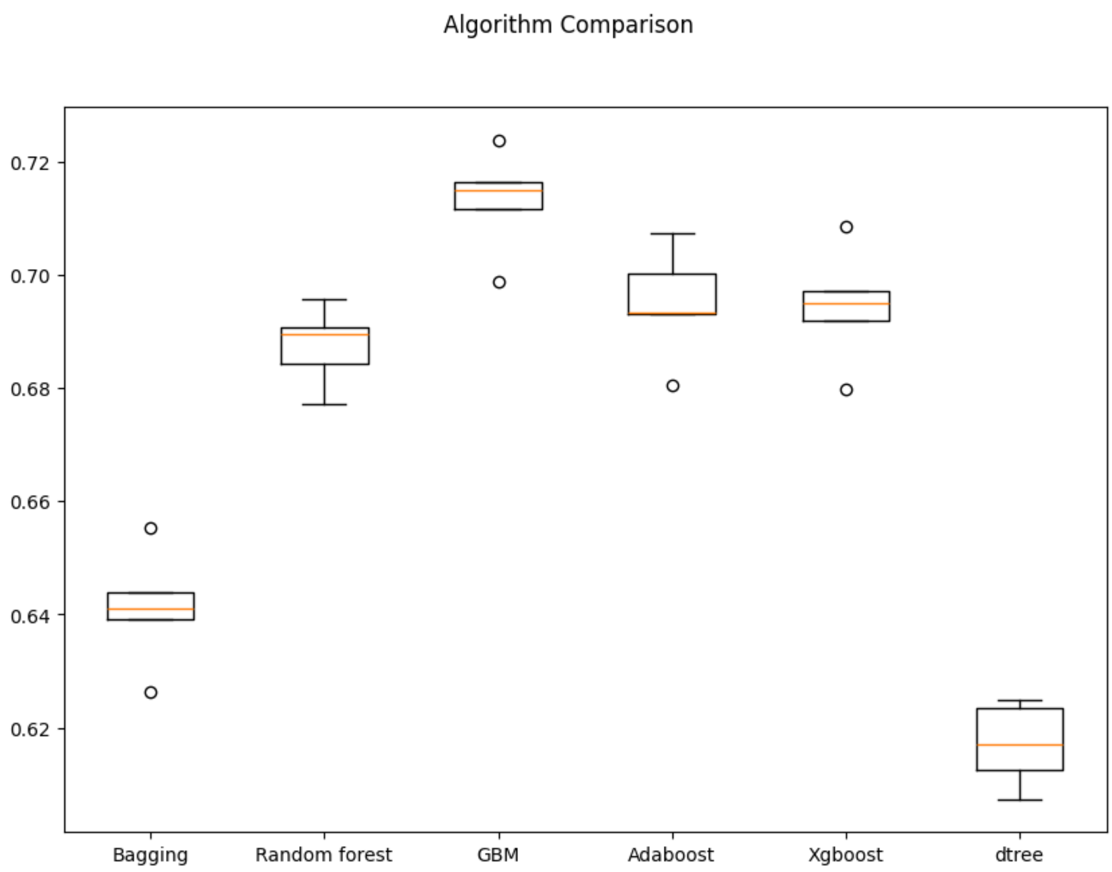
```
Before UnderSampling, counts of label '1': 11913
Before UnderSampling, counts of label '0': 5923

After UnderSampling, counts of label '1': 5923
After UnderSampling, counts of label '0': 5923

After UnderSampling, the shape of train_X: (11846, 21)
After UnderSampling, the shape of train_y: (11846,)
```

[48]:
```
models = []  # Empty list to store all the models

# Appending models into the list
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("Random forest", RandomForestClassifier(random_state=1))) ##␣
↪Complete the code to add Random Forest Classifier with random_state of 1.
models.append(("GBM", GradientBoostingClassifier(random_state=1))) ## Complete␣
↪the code to add Gradient Boosting Classifier with random_state of 1.
models.append(("Adaboost", AdaBoostClassifier(random_state=1))) ## Complete the␣
↪code to add AdaBoost Classifier with random_state of 1.
models.append(("Xgboost", XGBClassifier(random_state=1, eval_metric="logloss")))
models.append(("dtree", DecisionTreeClassifier(random_state=1))) ## Complete␣
↪the code to add Decision Tree Classifier with random_state of 1.

results1 = []  # Empty list to store all model's CV scores
names = []  # Empty list to store name of the models


# loop through all models to get the mean cross validated score
print("\n" "Cross-Validation performance on training dataset:" "\n")

for name, model in models:
    kfold = StratifiedKFold(
        n_splits=5, shuffle=True, random_state=1
    )  ## Complete the code to set the number of splits
    cv_result = cross_val_score(
        estimator=model, X=X_train_un, y=y_train_un,scoring = scorer,␣
↪cv=kfold,n_jobs =-1
    )
    results1.append(cv_result)
    names.append(name)
    print("{}: {}".format(name, cv_result.mean()))

print("\n" "Validation Performance:" "\n")
```

```
for name, model in models:
    model.fit(X_train_un,y_train_un) ## Complete the code to fit the model on␣
↪the undersampled data.
    scores = scorer(model, X_val, y_val) ## Complete the code to define the␣
↪metric function name.
    print("{}: {}".format(name, scores))
```

```
Cross-Validation performance on training dataset:

Bagging: 0.6411413525524321
Random forest: 0.6875011408129813
GBM: 0.7131358906535971
Adaboost: 0.6949405744215158
Xgboost: 0.6944693136408734
dtree: 0.617022679979161

Validation Performance:

Bagging: 0.6916956737941323
Random forest: 0.734144015259895
GBM: 0.7608695652173914
Adaboost: 0.7604202747950584
Xgboost: 0.7423652871123688
dtree: 0.6839080459770115
```

[49]:
```
# Plotting boxplots for CV scores of all models defined above
fig = plt.figure(figsize=(10, 7))

fig.suptitle("Algorithm Comparison")
ax = fig.add_subplot(111)

plt.boxplot(results1)
ax.set_xticklabels(names)

plt.show()
```



0.8 Hyperparameter Tuning

0.8.1 Tuning AdaBoost using oversampled data

[51]:
```
%%time

# defining model
Model = AdaBoostClassifier(random_state=1)

# Parameter grid to pass in RandomSearchCV
param_grid = {
    "n_estimators": [50, 100, 200, 300, 500], ## Complete the code to set the␣
 ↪number of estimators
    "learning_rate": [0.01, 0.05, 0.1, 0.5, 1.0], ## Complete the code to set␣
 ↪the learning rate.
    "estimator": [DecisionTreeClassifier(max_depth=1, random_state=1),␣
 ↪DecisionTreeClassifier(max_depth=2, random_state=1),␣
 ↪DecisionTreeClassifier(max_depth=3, random_state=1),
    ]
}
```

```
#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model,␣
↪param_distributions=param_grid, n_iter=50, n_jobs = -1, scoring=scorer,␣
↪cv=5, random_state=1)  ## Complete the code to set the cv parameter

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_over, y_train_over) ## Complete the code to fit the␣
↪model on over sampled data

print("Best parameters are {} with CV score={}:" .format(randomized_cv.
↪best_params_,randomized_cv.best_score_))
```

```
Best parameters are {'n_estimators': 100, 'learning_rate': 1.0, 'estimator':
DecisionTreeClassifier(max_depth=2, random_state=1)} with CV
score=0.7983888515430675:
CPU times: user 9.02 s, sys: 1.28 s, total: 10.3 s
Wall time: 17min 6s
```

[52]:
```
## Complete the code to set the best parameters.
tuned_ada = AdaBoostClassifier(
    n_estimators= 100, learning_rate= 1.0, estimator=␣
↪DecisionTreeClassifier(max_depth=2, random_state=1)
)

tuned_ada.fit(X_train_over, y_train_over)
```

[52]:
```
AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=2,
                                                    random_state=1),
                   n_estimators=100)
```

[53]:
```
ada_train_perf = model_performance_classification_sklearn(tuned_ada,␣
↪X_train_over, y_train_over)
ada_train_perf
```

[53]:

	Accuracy	Recall	Precision	F1
0	0.787501	0.840426	0.759982	0.798182

[54]:
```
## Complete the code to check the model performance for validation data.
ada_val_perf = model_performance_classification_sklearn(tuned_ada,X_val,y_val)
ada_val_perf
```

[54]:

	Accuracy	Recall	Precision	F1
0	0.734554	0.839573	0.779822	0.808595

0.8.2 Tuning Random forest using undersampled data

[55]:
```
%%time

# defining model
Model = RandomForestClassifier(random_state=1)

# Parameter grid to pass in RandomSearchCV
param_grid = {
    "n_estimators": [50, 100, 200, 300, 500], ## Complete the code to set the␣
 ↪number of estimators.
    "min_samples_leaf": np.arange(1, 5), ## Complete the code to set the␣
 ↪minimum number of samples in the leaf node.
    "max_features": [np.arange(1, 10, 2),'sqrt'], ## Complete the code to set␣
 ↪the maximum number of features.
    "max_samples": np.arange(0.5, 1.0, 0.1)} ## Complete the code to set the␣
 ↪maximum number of samples.


#Calling RandomizedSearchCV
randomized_cv = RandomizedSearchCV(estimator=Model,␣
 ↪param_distributions=param_grid, n_iter=50, n_jobs = -1, scoring=scorer,␣
 ↪cv=5, random_state=1) ## Complete the code to set the cv parameter

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_un, y_train_un) ## Complete the code to fit the model␣
 ↪on under sampled data

print("Best parameters are {} with CV score={}:" .format(randomized_cv.
 ↪best_params_,randomized_cv.best_score_))
```

```
Best parameters are {'n_estimators': 300, 'min_samples_leaf': np.int64(4),
'max_samples': np.float64(0.5), 'max_features': 'sqrt'} with CV
score=0.717262304407183:
CPU times: user 3.94 s, sys: 448 ms, total: 4.39 s
Wall time: 4min 16s
```

[56]:
```
# Complete the code to define the best model
tuned_rf2 = RandomForestClassifier(
    max_features='sqrt',
    random_state=1,
    max_samples=0.5,
    n_estimators=300,
    min_samples_leaf=4,
)

tuned_rf2.fit(X_train_un, y_train_un)
```

[56]: RandomForestClassifier(max_samples=0.5, min_samples_leaf=4, n_estimators=300,
                       random_state=1)

[57]:
```
rf2_train_perf = model_performance_classification_sklearn(
    tuned_rf2, X_train_un, y_train_un
)
rf2_train_perf
```

[57]:

	Accuracy	Recall	Precision	F1
0	0.7844	0.794699	0.77866	0.786598

[58]:
```
## Complete the code to print the model performance on the validation data.
rf2_val_perf = model_performance_classification_sklearn(tuned_rf2,X_val,y_val)
rf2_val_perf
```

[58]:

	Accuracy	Recall	Precision	F1
0	0.704463	0.718328	0.817034	0.764508

0.8.3 Tuning with Gradient boosting with oversampled data

[59]:
```
%%time

# defining model
Model = GradientBoostingClassifier(random_state=1)

## Complete the code to define the hyper parameters.
param_grid={"n_estimators": np.arange(50,501,50), "learning_rate": [0.01, 0.05,␣
↪0.1, 0.5, 1.0], "subsample":[0.7, 0.8, 0.9, 1.0], "max_features":['sqrt',␣
↪'log2', None]}

## Complete the code to set the cv parameter.
randomized_cv = RandomizedSearchCV(estimator=Model,␣
↪param_distributions=param_grid, scoring=scorer, n_iter=50, n_jobs = -1,␣
↪cv=5, random_state=1)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_over, y_train_over)

print("Best parameters are {} with CV score={}:" .format(randomized_cv.
↪best_params_,randomized_cv.best_score_))
```

```
Best parameters are {'subsample': 1.0, 'n_estimators': np.int64(200),
'max_features': None, 'learning_rate': 0.1} with CV score=0.8026500232175335:
CPU times: user 11.5 s, sys: 1.05 s, total: 12.6 s
Wall time: 12min 25s
```

[60]:
```
## Complete the code to define the best model.
tuned_gbm = GradientBoostingClassifier(
```

```
    max_features=None,
    random_state=1,
    learning_rate=0.1,
    n_estimators=200,
    subsample=1.0
)

tuned_gbm.fit(X_train_over, y_train_over)
```

[60]: GradientBoostingClassifier(n_estimators=200, random_state=1)

[61]:
```
gbm_train_perf = model_performance_classification_sklearn(
    tuned_gbm, X_train_over, y_train_over
)
gbm_train_perf
```

[61]:

	Accuracy	Recall	Precision	F1
0	0.804625	0.865525	0.77155	0.81584

[62]:
```
## Complete the code to print the model performance on the validation data.
gbm_val_perf = model_performance_classification_sklearn(tuned_gbm,X_val, y_val)
gbm_val_perf
```

[62]:

	Accuracy	Recall	Precision	F1
0	0.737316	0.852416	0.776214	0.812532

0.8.4 Tuning XGBoost using oversampled data

[63]:
```
%%time

# defining model
Model = XGBClassifier(random_state=1,eval_metric='logloss')

## Complete the code to define the hyperparameters
param_grid={'n_estimators':[50, 100, 200, 300, 500],'scale_pos_weight':[1, 2,␣
↪3], 'learning_rate':[0.01, 0.05, 0.1, 0.2, 0.3], 'gamma':[0, 0.1, 0.2, 0.3,␣
↪0.4], 'subsample':[0.6, 0.7, 0.8, 0.9, 1.0]}

## Complete the code to set the cv parameter
randomized_cv = RandomizedSearchCV(estimator=Model,␣
↪param_distributions=param_grid, n_iter=50, n_jobs = -1, scoring=scorer,␣
↪cv=5, random_state=1)

#Fitting parameters in RandomizedSearchCV
randomized_cv.fit(X_train_over,y_train_over)## Complete the code to fit the␣
↪model on over sampled data
```

```
print("Best parameters are {} with CV score={}:" .format(randomized_cv.
 ↪best_params_,randomized_cv.best_score_))
```

```
Best parameters are {'subsample': 0.8, 'scale_pos_weight': 3, 'n_estimators':
200, 'learning_rate': 0.1, 'gamma': 0} with CV score=0.8135471199686105:
CPU times: user 2.61 s, sys: 299 ms, total: 2.9 s
Wall time: 2min 14s
```

[64]:
```
## Complete the code to define the best model
xgb2 = XGBClassifier(
    random_state=1,
    eval_metric='logloss',
    subsample=0.8,
    scale_pos_weight=3,
    n_estimators=200,
    learning_rate=0.1,
    gamma=0,
)

xgb2.fit(X_train_over, y_train_over)
```

[64]:
```
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=0, grow_policy=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=None, multi_strategy=None, n_estimators=200,
              n_jobs=None, num_parallel_tree=None, random_state=1, …)
```

[65]:
```
xgb2_train_perf = model_performance_classification_sklearn(
    xgb2, X_train_over, y_train_over
)
xgb2_train_perf
```

[65]:

	Accuracy	Recall	Precision	F1
0	0.806766	0.990347	0.724381	0.836738

[66]:
```
## Complete the code to print the model performance on the validation data.
xgb2_val_perf = model_performance_classification_sklearn(xgb2,X_val,y_val)
xgb2_val_perf
```

[66]:

	Accuracy	Recall	Precision	F1
0	0.71304	0.959512	0.711427	0.817053

We have now tuned all the models, let's compare the performance of all tuned models and see which one is the best.

0.9 Model performance comparison and choosing the final model

[67]:
```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        gbm_train_perf.T,
        xgb2_train_perf.T,
        ada_train_perf.T,
        rf2_train_perf.T,
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Gradient Boosting tuned with oversampled data",
    "XGBoost tuned with oversampled data",
    "AdaBoost tuned with oversampled data",
    "Random forest tuned with undersampled data",
]
print("Training performance comparison:")
models_train_comp_df
```

```
Training performance comparison:
```

[67]:
```
           Gradient Boosting tuned with oversampled data  \
Accuracy                                        0.804625
Recall                                          0.865525
Precision                                       0.771550
F1                                              0.815840

           XGBoost tuned with oversampled data  \
Accuracy                              0.806766
Recall                                0.990347
Precision                             0.724381
F1                                    0.836738

           AdaBoost tuned with oversampled data  \
Accuracy                               0.787501
Recall                                 0.840426
Precision                              0.759982
F1                                     0.798182

           Random forest tuned with undersampled data
Accuracy                                     0.784400
Recall                                       0.794699
```

```
Precision                                        0.778660
F1                                               0.786598
```

[68]:
```
# validation performance comparison

models_val_comp_df = pd.concat(
    [
        gbm_val_perf.T,
        xgb2_val_perf.T,
        ada_val_perf.T,
        rf2_val_perf.T,
    ],
    axis=1,
)
models_val_comp_df.columns = [
    "Gradient Boosting tuned with oversampled data",
    "XGBoost tuned with oversampled data",
    "AdaBoost tuned with oversampled data",
    "Random forest tuned with undersampled data",
]
print("Validation performance comparison:")
models_val_comp_df
```

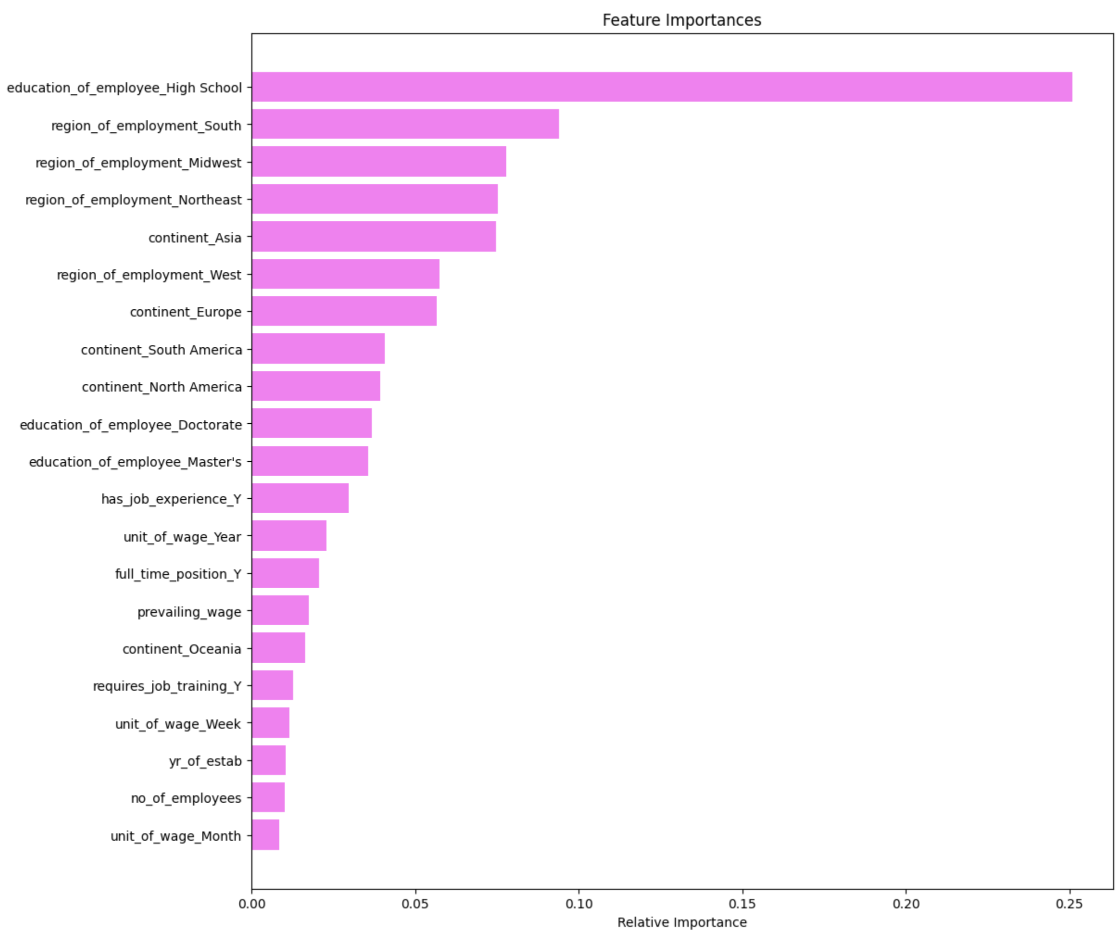
```
Validation performance comparison:
```

[68]:
```
           Gradient Boosting tuned with oversampled data  \
Accuracy                                        0.737316
Recall                                          0.852416
Precision                                       0.776214
F1                                              0.812532

           XGBoost tuned with oversampled data  \
Accuracy                              0.713040
Recall                                0.959512
Precision                             0.711427
F1                                    0.817053

           AdaBoost tuned with oversampled data  \
Accuracy                               0.734554
Recall                                 0.839573
Precision                              0.779822
F1                                     0.808595

           Random forest tuned with undersampled data
Accuracy                                     0.704463
Recall                                       0.718328
Precision                                    0.817034
F1                                           0.764508
```

```
[69]: ## Complete the code to print the model performance on the test data by the
      ↪best model.
      test = model_performance_classification_sklearn(xgb2, X_test, y_test)
      test
```

```
[69]:    Accuracy    Recall  Precision        F1
      0  0.713725  0.958904   0.712209  0.817348
```

```
[70]: feature_names = X_train.columns
      importances = xgb2.feature_importances_ ## Complete the code to print the
      ↪feature importances from the best model.
      indices = np.argsort(importances)

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



0.10 Actionable Insights and Recommendations

Power Ahead _____