Data Science and Business Analytics

Ensemble Techniques and Model Tuning

Visa Approval Facilitation

Ensemble Techniques and Model Tuning: EasyVisa

Project Overview

This project develops a classification model to predict whether an EasyVisa case will be certified or denied, leveraging applicant and employer attributes to inform decision-making. By identifying key drivers of "Case_Status," the model aims to streamline review workflows, reduce costly certification errors, and support compliance objectives.

Problem Statement

Context

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

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

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

Objective

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

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

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

Data Description

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

- case_id: ID of each visa application
- continent: Information of continent the employee
- education_of_employee: Information of education of the employee
- has_job_experience: Does the employee have 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

Import Libraries

```
# 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 -q --user
```

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

# Libraries 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)

# To oversample and undersample data
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler

# 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.model_selection import RandomizedSearchCV
```

```
# Custom Color Set - ElleSet
ElleSet = [
    (102/255, 194/255, 165/255),  # Muted Green
    (214/255,  95/255,  95/255),  # Muted Red
    (141/255, 160/255, 203/255),  # Soft Blue
    (130/255, 198/255, 226/255),  # Muted Blue
    (166/255, 216/255,  84/255),  # Lime Green
    (230/255, 196/255, 148/255),  # Beige
    (179/255, 179/255, 179/255),  # Neutral Gray
    (255/255, 217/255,  47/255),  # Yellow
    (204/255, 153/255, 255/255),  # Soft Lavender
    (255/255, 153/255, 204/255)   # Blush Pink
```

```
]

# Function to apply ElleSet as the default Seaborn palette
def use_ElleSet():
    import seaborn as sns
    sns.set_palette(ElleSet)
```

Load & Copy Dataset

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

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

```
visa = pd.read_csv('/content/drive/MyDrive/Model Tuning/EasyVisa.csv')
```

```
# Copy of data to new variable
data = visa.copy()
```

Overview of 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	full_time_position	case_status
0	EZYV01	Asia	High School	N	N	14513	2007	West	592.2029	Hour	Y	Denied
1	EZYV02	Asia	Master's	Y	N	2412	2002	Northeast	83425.6500	Year	Y	Certified
2	EZYV03	Asia	Bachelor's	N	Y	44444	2008	West	122996.8600	Year	Y	Denied
3	EZYV04	Asia	Bachelor's	N	N	98	1897	West	83434.0300	Year	Y	Denied
4	EZYV05	Africa	Master's	Y	N	1082	2005	South	149907.3900	Year	Y	Certified

```
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_wage	full_time_position	case_status
25475	EZYV25476	Asia	Bachelor's	Y	Y	2601	2008	South	77092.57	Year	Y	Certified
25476	EZYV25477	Asia	High School	Y	N	3274	2006	Northeast	279174.79	Year	Y	Certified
25477	EZYV25478	Asia	Master's	Y	N	1121	1910	South	146298.85	Year	N	Certified
25478	EZYV25479	Asia	Master's	Y	Y	1918	1887	West	86154.77	Year	Y	Certified
25479	EZYV25480	Asia	Bachelor's	Y	N	3195	1960	Midwest	70876.91	Year	Y	Certified

- The dataset consists of 12 columns capturing a blend of applicant and employer characteristics, wage information, and case outcomes. Features span multiple data types: categorical fields (e.g., continent, education_of_employee, region_of_employment), binary flags (has_job_experience, requires_job_training, full_time_position), and numeric variables (no_of_employees, yr_of_estab, prevailing_wage)
- The target variable, 'case_status', indicates whether each application was Certified or Denied, enabling supervised classification modeling.

```
data.shape
```

```
(25480, 12)
```

- The dataset comprises 25,480 rows and 12 columns, with each row representing an individual EasyVisa application and each column capturing a unique attribute or outcome relevant to the case evaluation process.

```
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
```

- Complete Data: All 12 columns have no missing values, indicating a clean dataset ready for modeling.
- Data Types: The dataset includes 9 categorical (object) features and 3 numerical features (no_of_employees, yr_of_estab, and prevailing_wage), which suggests preprocessing steps like encoding and scaling will be necessary.
- Balanced Schema: The presence of relevant features such as education_of_employee, job experience, training, and wage aligns well with the objective of classifying visa approval likelihood (case_status as target).

```
# Check for duplicates
data.duplicated().sum()
```

```
np.int64(0)
```

- There are 0 duplicate rows 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.

Leading Questions

What is the distribution of visa case statuses (certified vs. denied)?

1. What is the distribution of visa case statuses (certified vs. denied)?
2. How does the education level of employees impact visa approval rates?
3. Is there a significant difference in visa approval rates between employees with and without prior job experience?
4. How does the prevailing wage affect visa approval? Do higher wages lead to higher chances of approval?
5. Do certain regions in the US have higher visa approval rates compared to others?
6. How does the number of employees in a company influence visa approval? Do larger companies have a higher approval rate?
7. Are visa approval rates different across various continents of employees? Which continent has the highest and lowest approval rates?

Statistical Summary of Data

```
data.describe(include='all').T
```

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
case_id	25480	25480	EZYV25480	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
continent	25480	6	Asia	16861	NaN	NaN	NaN	NaN	NaN	NaN	NaN
education_of_employee	25480	4	Bachelor's	10234	NaN	NaN	NaN	NaN	NaN	NaN	NaN
has_job_experience	25480	2	Y	14802	NaN	NaN	NaN	NaN	NaN	NaN	NaN
requires_job_training	25480	2	N	22525	NaN	NaN	NaN	NaN	NaN	NaN	NaN
no_of_employees	25480.0	NaN	NaN	NaN	5667.04321	22877.928848	-26.0	1022.0	2109.0	3504.0	602069.0
yr_of_estab	25480.0	NaN	NaN	NaN	1979.409929	42.366929	1800.0	1976.0	1997.0	2005.0	2016.0
region_of_employment	25480	5	Northeast	7195	NaN	NaN	NaN	NaN	NaN	NaN	NaN
prevailing_wage	25480.0	NaN	NaN	NaN	74455.814592	52815.942327	2.1367	34015.48	70308.21	107735.5125	319210.27
unit_of_wage	25480	4	Year	22962	NaN	NaN	NaN	NaN	NaN	NaN	NaN
full_time_position	25480	2	Y	22773	NaN	NaN	NaN	NaN	NaN	NaN	NaN
case_status	25480	2	Certified	17018	NaN	NaN	NaN	NaN	NaN	NaN	NaN

- Data Quality Alert: The no_of_employees column contains a negative value (-26), which is invalid and needs to be investigated or corrected during data cleaning.
- Skewed Distributions: prevailing_wage and no_of_employees show high standard deviations (52,815 and 22,878 respectively) relative to their means, indicating likely right-skewed distributions with some large outliers.
- Historical Outlier: The minimum yr_of_estab is 1800, which seems implausibly early for most modern businesses and may indicate a data entry error worth validating.

Correct Negative Value

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

```
(33, 12)
```

- Identified 33 records with negative values in no_of_employees, likely due to sign entry errors. These will be corrected by taking absolute values to preserve data fidelity.

```
# Convert to absolute value
data["no_of_employees"] = data["no_of_employees"].abs()
```

- Corrected 33 records with negative no_of_employees values by replacing them with their absolute values, assuming data entry errors in sign while preserving the original scale.

Categorical Variable Counts

```python
# Count of each unique category in categorical columns
cat_col = list(data.select_dtypes("object").columns)
for column in cat_col:
    print(data[column].value_counts())
    print("-" * 50)
```

```
case_id
EZYV25480    1
EZYV01       1
EZYV02       1
EZYV03       1
EZYV04       1
            ..
EZYV13       1
EZYV12       1
EZYV11       1
EZYV10       1
EZYV09       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
--------------------------------------------------
```

- case_id appears to contain only unique values and serves as a record identifier; if it holds no predictive value then it will be dropped.
- continent is skewed toward Asia (~66%), indicating a strong regional bias in the dataset.
- education_of_employee is dominated by Bachelor's and Master's degrees (~78% combined).
- has_job_experience, requires_job_training, and full_time_position are all likely predictive due to their clear categorical splits.
- unit_of_wage is mostly Yearly (~90%), suggesting others should be normalized.
- case_status shows moderate imbalance (67% Certified vs 33% Denied), relevant for model evaluation.

```python
# Unique values case_id
data["case_id"].nunique()
```

```
25480
```

- case_id: Contains 25,480 unique values, serving as a primary key; it is not useful for modeling and will be dropped.

```python
data.drop(["case_id"], axis=1, inplace=True)
```

Univariate Analysis

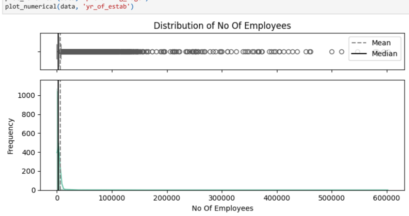
Plot Functions

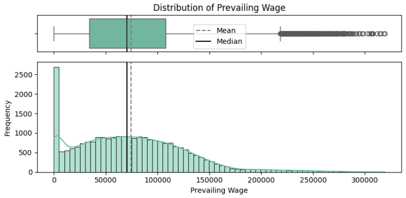
```python
# Numerical Plot Function
def plot_numerical(data, col):
    fig, ax = plt.subplots(2, 1, figsize=(8, 4), height_ratios=[1, 3], sharex=True)

    # Boxplot
    sns.boxplot(x=data[col], ax=ax[0], color=ElleSet[0])
    ax[0].set(xlabel='', ylabel='', title=f'Distribution of {col.replace("_", " ").title()}')
    ax[0].axvline(data[col].mean(), color='gray', linestyle='--', label='Mean')
    ax[0].axvline(data[col].median(), color='black', linestyle='-', label='Median')
    ax[0].legend()

    # Histogram + KDE
    sns.histplot(data[col], kde=True, ax=ax[1], color=ElleSet[0])
    ax[1].axvline(data[col].mean(), color='gray', linestyle='--')
    ax[1].axvline(data[col].median(), color='black', linestyle='-')
    ax[1].set(xlabel=col.replace('_', ' ').title(), ylabel='Frequency')

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

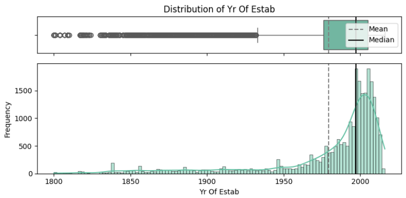
```python
# Categorical Plot Function
def plot_categorical(data, feature, palette=ElleSet, labels=None):

    plt.figure(figsize=(6, 4))
    sns.countplot(x=feature, data=data, palette=palette[:data[feature].nunique()])
    plt.title(feature.replace('_', ' ').title())
    plt.xlabel(feature.replace('_', ' ').title())
    plt.ylabel('Count')

    if labels:
        plt.xticks(ticks=range(len(labels)), labels=labels)

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

Numerical Features

```python
# Numerical Univariate Plots
plot_numerical(data, 'no_of_employees')
plot_numerical(data, 'prevailing_wage')
plot_numerical(data, 'yr_of_estab')
```







no_of_employees:

- The majority of employers in the dataset are small to mid-sized businesses, with a steep drop-off in size.
- A few records show extremely large companies—outliers in the hundreds of thousands—which stretch the scale and pull the mean far from the center.
- Given the business case, it's plausible that company size might influence visa decisions, but the data may need to be normalized or binned to avoid model distortion.

prevailing_wage:

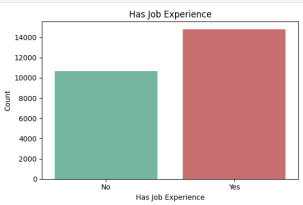
- Wages range widely, but most values cluster under $100K, with a strong early peak and a long tail toward higher wages.
- The skew suggests that while most roles are mid-range, a subset of high-paying roles stands out—likely roles in tech or senior positions.
- This variable will likely be a strong predictor, and scaling or transformation will help make patterns more visible to the model.

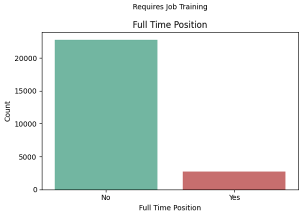
yr_of_estab:

- Most companies in this dataset were established in the last 40 years, aligning with broader trends in the labor market.
- A few companies claim founding years in the 1800s—likely data entry errors or legacy artifacts.
- This feature may help capture business maturity; however, outlier correction or capping will help clean the variable before analysis.

Categorical Features

```python
# Categorical Plots with Yes/No Labels
plot_categorical(data, 'has_job_experience', labels=['No', 'Yes'])
plot_categorical(data, 'requires_job_training', labels=['No', 'Yes'])
plot_categorical(data, 'full_time_position', labels=['No', 'Yes'])
```





has_job_experience

- A majority of applicants (approximately 58%) report prior job experience, indicating a workforce with established skills.
- The remaining group without experience represents a sizable portion of the dataset and may reflect early-career or entry-level applicants.
- Given its relevance to workforce readiness, this feature is likely to hold predictive value in the classification task.

requires_job_training

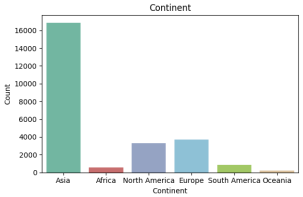
- The vast majority of applicants (over 22,000) do not require job training, suggesting that employers are primarily submitting applications for candidates who are job-ready.
- A smaller subset (around 12%) requires training, which may imply additional resource needs or onboarding considerations.
- The imbalance in this feature may signal differences in approval likelihood, depending on policy or operational constraints.

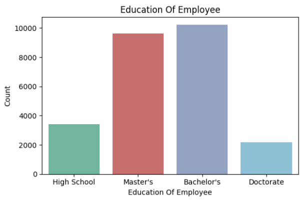
full_time_position

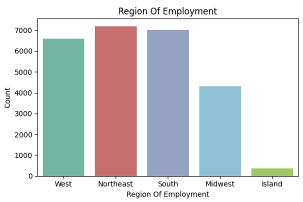
- Most applications are for full-time positions, while part-time or unspecified roles comprise a much smaller portion of the dataset.
- This distribution aligns with typical sponsorship patterns, where full-time roles may be prioritized due to perceived stability and long-term value.
- The observed imbalance supports the relevance of this feature in evaluating employment context and approval outcomes.

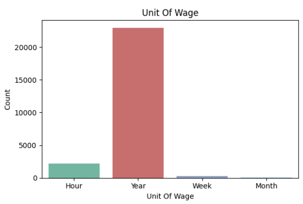
```python
# Remaining Categorical Features
```

```
plot_categorical(data, 'continent')
plot_categorical(data, 'education_of_employee')
plot_categorical(data, 'region_of_employment')
plot_categorical(data, 'unit_of_wage')
```









continent

- The distribution is highly imbalanced, with a majority of applicants originating from Asia. Other regions such as Africa, South America, and Oceania are minimally represented.
- This imbalance may reflect external sourcing trends, but it presents potential risk of model bias if not addressed through evaluation metrics or sampling strategies.
- Geographic origin may serve as a useful differentiator if approval outcomes vary meaningfully across continents.

education_of_employee

- Applicants predominantly hold Bachelor's or Master's degrees, indicating a highly qualified sample population.
- High School and Doctorate-level applicants represent smaller proportions, suggesting these are edge cases within the dataset.
- The spread of educational levels provides an opportunity to assess whether advanced education consistently correlates with favorable outcomes.

region_of_employment

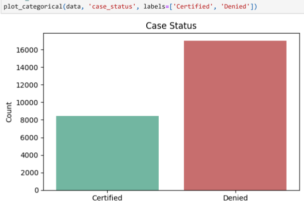
- Employment is distributed fairly evenly among the West, South, and Northeast regions, with moderate representation in the Midwest and minimal activity in the Island region.
- This broad geographic coverage supports generalizability across U.S. labor markets, while the low volume from Island regions may limit modeling precision for that subgroup.

unit_of_wage

- The dataset is dominated by Yearly wage reporting, which will simplify normalization of wage values for comparison and modeling.
- Other units (Hourly, Weekly, Monthly) are sparsely used and may require conversion or exclusion depending on the modeling approach.
- Consistency in wage units is important for accurate feature scaling and wage-based analysis.

Target Variable Visualization

```
# Case_Status
plot_categorical(data, 'case_status', labels=['Certified', 'Denied'])
```



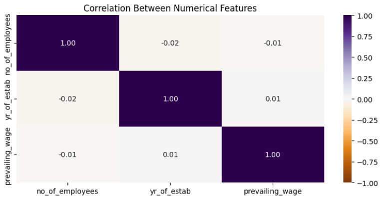
- case_status shows a clear class imbalance, with approximately 66% Denied and 34% Certified cases.
- The imbalance indicates that the majority of applications are not approved, reflecting a high threshold for certification.
- From a technical perspective, this imbalance necessitates the use of evaluation metrics beyond accuracy, such as precision, recall, and F1-score, and may require resampling techniques during modeling.
- From a business standpoint, the ability to predict approval outcomes can enhance OFLC efficiency by focusing attention on higher-probability cases, supporting faster and more equitable visa processing.

Bivariate Analysis

Correlation Between Numerical Features

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

```python
# Correlation heatmap
plt.figure(figsize=(8, 4))
sns.heatmap(data[cols_list].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="PuOr")
plt.title('Correlation Between Numerical Features')
plt.tight_layout()
plt.show()
```



- The three numerical features—no_of_employees, yr_of_estab, and prevailing_wage—show no significant linear correlation with one another. All pairwise correlation values are close to zero.
- This indicates that each feature provides independent information, which is beneficial for modeling as it reduces the risk of multicollinearity.
- No feature pair appears redundant, so all three can be retained as inputs to the classification model without concern for overlap.

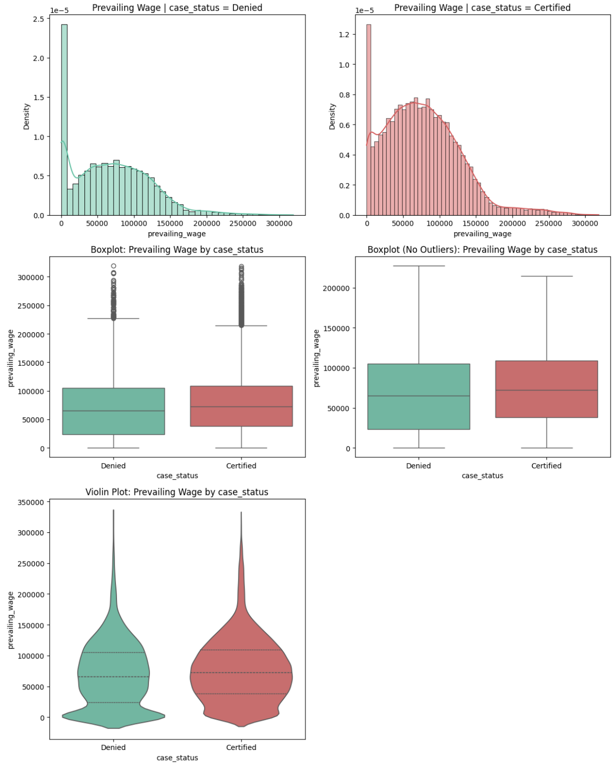
Plot Functions

```python
def distribution_plot_wrt_target(data, predictor, target='case_status'):
    """
    Creates histogram, boxplot, and violin plot comparisons of a numerical predictor grouped by a binary target.

    Parameters:
    - data: DataFrame
    - predictor: numerical column to plot
    - target: binary classification target (default: 'case_status')
    """
    # Define Layout
    fig, axs = plt.subplots(3, 2, figsize=(12, 15), gridspec_kw={'height_ratios': [1, 1, 1.2]})
    target_classes = data[target].unique()

    # KDE Histogram: Class 1
    axs[0, 0].set_title(f"{predictor.replace('_', ' ').title()} | {target} = {target_classes[0]}")
    sns.histplot(
        data=data[data[target] == target_classes[0]],
        x=predictor,
        kde=True,
        ax=axs[0, 0],
        color=ElleSet[0],
        stat="density"
    )

    # KDE Histogram: Class 2
    axs[0, 1].set_title(f"{predictor.replace('_', ' ').title()} | {target} = {target_classes[1]}")
    sns.histplot(
        data=data[data[target] == target_classes[1]],
        x=predictor,
        kde=True,
        ax=axs[0, 1],
        color=ElleSet[1],
        stat="density"
    )

    # Boxplot with outliers
    axs[1, 0].set_title(f"Boxplot: {predictor.replace('_', ' ').title()} by {target}")
    sns.boxplot(data=data, x=target, y=predictor, ax=axs[1, 0], palette=ElleSet[:2])

    # Boxplot without outliers
    axs[1, 1].set_title(f"Boxplot (No Outliers): {predictor.replace('_', ' ').title()} by {target}")
    sns.boxplot(data=data, x=target, y=predictor, ax=axs[1, 1], showfliers=False, palette=ElleSet[:2])

    # Violin plot across target classes
    axs[2, 0].set_title(f"Violin Plot: {predictor.replace('_', ' ').title()} by {target}")
    sns.violinplot(data=data, x=target, y=predictor, palette=ElleSet[:2], ax=axs[2, 0], inner='quartile')

    # Hide unused 3rd row (right-side) plot
    axs[2, 1].axis('off')

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

```python
def stacked_barplot(data, predictor, target='case_status', palette=ElleSet[:2]):
    """
    Displays count table and plots a normalized stacked bar chart to show target distribution per category.

    Parameters:
    - data: DataFrame
    - predictor: categorical feature
    - target: classification target (default: 'case_status')
    - palette: color scheme to use (default: ElleSet)
    """

    # Tabulate counts
    count_table = pd.crosstab(data[predictor], data[target], margins=True)
    display(count_table)
    print("-" * 100)

    # Normalized proportions
    prop_table = pd.crosstab(data[predictor], data[target], normalize='index')

    # Sort for cleaner display
    dominant_class = data[target].value_counts().idxmax()
    prop_table = prop_table.sort_values(by=dominant_class, ascending=False)

    # Plot
    prop_table.plot(
        kind='bar',
        stacked=True,
        figsize=(max(6, len(prop_table) + 2), 5),
        color=palette,
        edgecolor='black'
    )
    plt.title(f"{predictor.replace('_', ' ').title()} vs {target.replace('_', ' ').title()}")
    plt.ylabel('Proportion')
    plt.xlabel(predictor.replace('_', ' ').title())
    plt.legend(title=target.replace('_', ' ').title(), bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.tight_layout()
    plt.show()
```

Prevailing Wage vs. Case Status

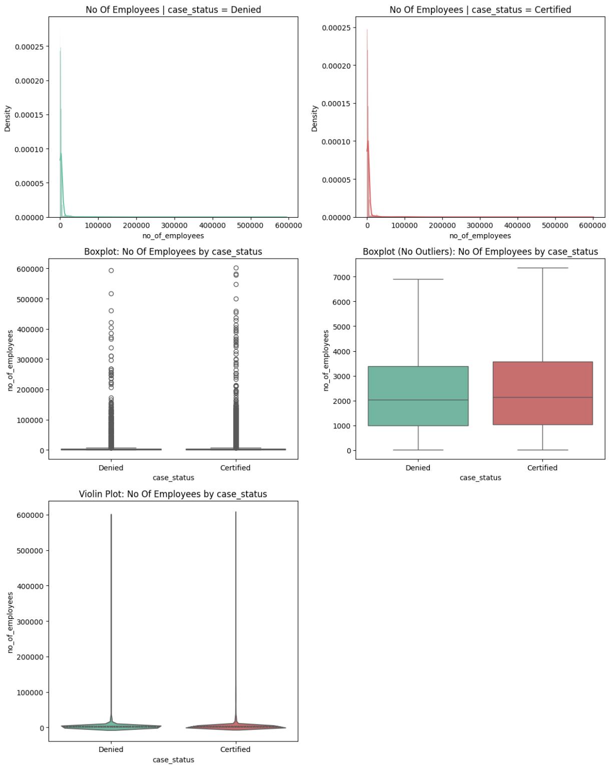
```python
# Bivariate analysis: prevailing_wage vs. case_status
distribution_plot_wrt_target(data, predictor='prevailing_wage', target='case_status')
```



- Certified applications are generally associated with higher wage levels compared to Denied cases. This is visible across all views—especially the boxplots and violin plot, where the central tendency and upper distribution are clearly shifted upward.
- The denied group includes a larger share of lower-wage applications, suggesting that wage level may act as a signal of job quality or role justification in the certification process.
- While both groups have outliers at the higher end of the wage spectrum, the certified group shows a broader distribution in the middle to upper wage ranges, hinting at stronger alignment between compensation and approval likelihood.
- This relationship is consistent with policy priorities around market-aligned wages, and reinforces the importance of prevailing wage as a potential predictive feature in modeling visa outcomes.
- The pattern suggests prevailing wage could help the OFLC prioritize applications that are more competitive or aligned with policy standards, supporting more efficient and equitable visa certification.

Number of Employees vs. Case Status

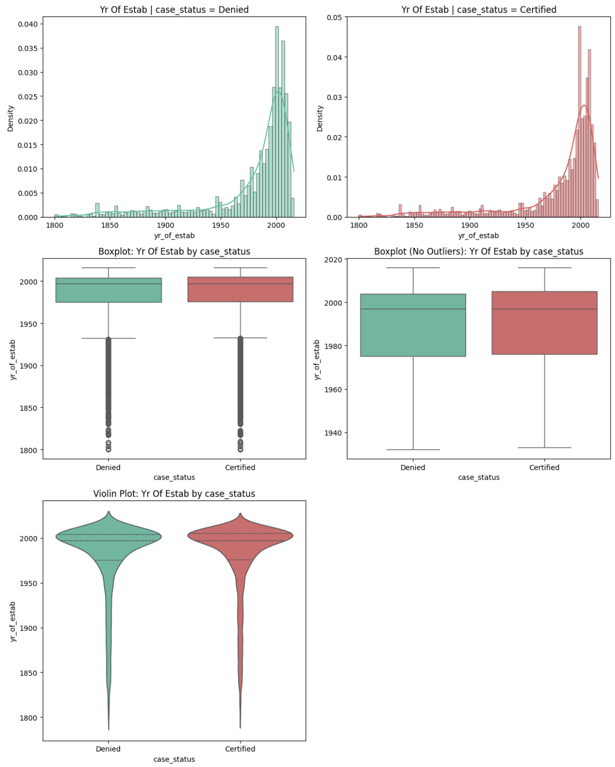
```
# Bivariate analysis: no_of_employees vs. case_status
distribution_plot_wrt_target(data, predictor='no_of_employees', target='case_status')
```



- Most applications come from small to mid-sized companies, with a few extremely large employers stretching the upper range of the distribution.
- The relationship between company size and case status appears weak overall; the central tendencies for certified and denied cases are largely similar once outliers are removed.
- While very large employers may receive favorable consideration, company size alone does not appear to strongly differentiate application outcomes.

Years of Establishment vs. Case Status

```
# Bivariate analysis: yr_of_estab vs. case_status
distribution_plot_wrt_target(data, predictor='yr_of_estab', target='case_status')
```

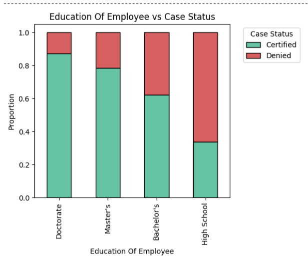


- Most companies in the dataset were established after 1980, with a noticeable peak around the early 2000s.
- Certified cases are slightly more concentrated among newer establishments, while denied cases show a modestly broader spread across earlier decades.
- A small number of records reflect very early establishment years (as far back as 1800), which may merit further validation or special treatment in modeling.

Education of Employees vs. Case Status

```
# Bivariate analysis: education_of_employee vs. case_status
stacked_barplot(data, predictor='education_of_employee', target='case_status')
```

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

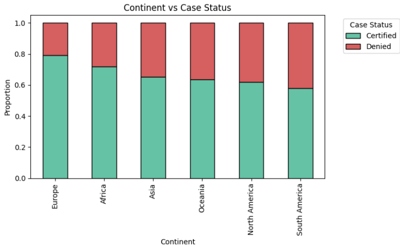


- Certification rates increase with educational attainment, with Doctorate and Master's degree holders showing the highest proportion of approvals.
- High School applicants are more likely to be denied, representing the only group with a majority of rejections.
- The pattern suggests education is an important differentiator in the review process, reinforcing its value as a predictive input for classifying visa outcomes.

Continent vs. Case Status

```
# Bivariate analysis: continent vs. case_status
stacked_barplot(data, predictor='continent', target='case_status')
```

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

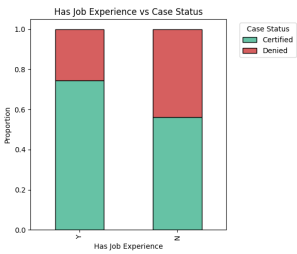


- Certification rates vary across continents, with Europe showing the highest proportion of approvals and South America the lowest.
- Applicants from Asia represent the largest share overall, with an approval rate close to the global average.
- These differences may reflect regional sourcing patterns or perceived alignment between skill profiles and U.S. labor needs, suggesting continent may hold predictive value in classification.

Job Experience vs. Case Status

```
# Bivariate analysis: has_job_experience vs. case_status
stacked_barplot(data, predictor='has_job_experience', target='case_status')
```

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

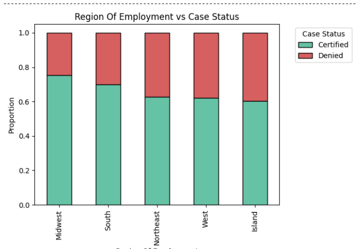


- Applicants with prior job experience have a notably higher certification rate compared to those without.
- While both groups are sizable, the approval rate is strongest among experienced candidates, suggesting experience may be perceived as a key indicator of readiness or role fit.
- This variable is likely to contribute meaningful separation in modeling approval likelihood.

Region of Employment vs. Case Status

```
# Bivariate analysis: region_of_employment vs. case_status
stacked_barplot(data, predictor='region_of_employment', target='case_status')
```

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



- Approval rates differ across U.S. regions, with the Midwest showing the highest share of certified applications.
- The South, West, and Island regions show lower certification rates, though all remain relatively close to the overall average.
- These differences may reflect regional processing trends, labor demands, or role types—making region a potentially useful, though not dominant, classification feature.

Requires Job Training vs. Case Status

```
# Bivariate analysis: requires_job_training vs. case_status
stacked_barplot(data, predictor='requires_job_training', target='case_status')
```

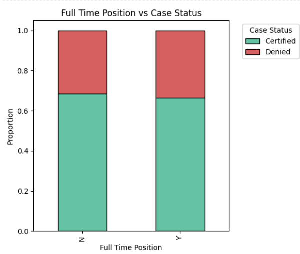
case_status	Certified	Denied	All
requires_job_training			
N	15012	7513	22525
Y	2006	949	2955
All	17018	8462	25480

- The proportion of certified applications is slightly higher for candidates who do not require training, but the difference between groups is modest.
- The vast majority of applicants fall into the "No training required" group, reinforcing that most employers file applications for job-ready candidates rather than those needing additional preparation.
- While this feature may carry limited predictive weight alone, it complements other indicators of preparedness such as experience or education.

Full Time Position vs. Case Status

```
# Bivariate analysis: full_time_position vs. case_status
stacked_barplot(data, predictor='full_time_position', target='case_status')
```

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



- The vast majority of applications are for full-time roles, with only a small fraction representing part-time or other arrangements.
- Certification rates are comparable between full-time and non-full-time roles, with full-time slightly more favorable but not dramatically so.
- While the signal is subtle, full-time status likely contributes to model accuracy when combined with stronger predictors such as wage or experience.

Summary of Key EDA Findings

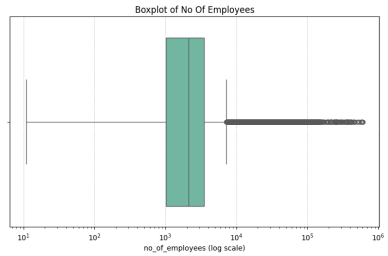
- Visa certification is the majority outcome, but a meaningful share of cases are denied, reflecting real-world selectivity in the process. This highlights the importance of identifying factors that distinguish successful from unsuccessful applications.
- Prevailing wage and compensation structure stand out: Higher wages and "Yearly" wage units are strongly associated with approval, while "Hourly" and "Weekly" wages correspond with more denials.
- Full-time positions have a clear advantage: Applications for full-time roles are certified at significantly higher rates compared to part-time roles.
- Applicant qualifications matter: Those with prior job experience and advanced degrees consistently see higher certification rates, underscoring the value placed on skilled and experienced talent.
- Employer characteristics show modest influence: Larger, more established companies see slightly higher approval rates, though this effect is secondary to job and applicant features.
- Regional and continental variations exist, but are less influential than wage, job status, and applicant background.

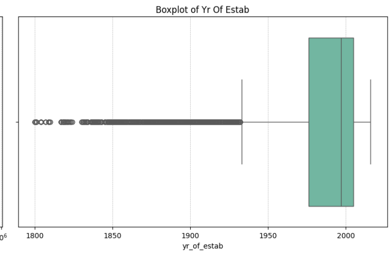
Data Pre-processing

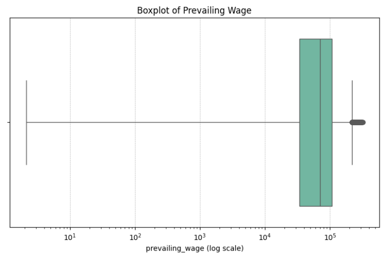
- Missing value treatment (if needed)
- Feature engineering (if needed)
- Outlier detection and treatment (if needed)
- Preparing data for modeling
- Any other preprocessing steps (if needed)

Outlier Review and Treatment

```
# Outlier detection using boxplot
import seaborn as sns

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

plt.figure(figsize=(15, 10))

for i, variable in enumerate(numeric_columns):
    plt.subplot(2, 2, i + 1)
    sns.boxplot(x=data[variable], color=ElleSet[0])

    # Optional log scale for better visibility on skewed data
    if variable in ['no_of_employees', 'prevailing_wage']:
        plt.xscale('log')
        plt.xlabel(f'{variable} (log scale)')
    else:
        plt.xlabel(variable)

    plt.title(f'Boxplot of {variable.replace("_", " ").title()}')
    plt.grid(True, axis='x', linestyle='--', linewidth=0.5)
    plt.tight_layout()

plt.show()
```







- **no_of_employees:**

  Strong right skew, with extreme outliers extending into the hundreds of thousands. These are visually and statistically significant, though likely reflect real large firms.

- **yr_of_estab:**

  Most companies cluster after 1950, but a long tail stretches back to 1800. Older years may include data entry issues or rare legacy organizations.

- **prevailing_wage:**

  High-end outliers are present but seem well-aligned with real-world wage variability in specialized or executive roles.

Number of Employees. Method: Cap

```
# Outlier capping for no_of_employees at 99th percentile
percentile_99 = data['no_of_employees'].quantile(0.99)
data['no_of_employees'] = np.where(
    data['no_of_employees'] > percentile_99,
    percentile_99,
    data['no_of_employees']
)
```

- Preserves the overall data while softening the impact of extreme outliers.
- Reduces the influence of unusually large companies.
- Improves robustness of downstream analysis.

Year of Establishment. Method: Create Lower Bound

```
# Flooring yr_of_estab at 1950 to correct likely outliers
data['yr_of_estab'] = np.where(
    data['yr_of_estab'] < 1950,
    1950,
    data['yr_of_estab']
)
```

- Floors extreme historical values to a realistic threshold.
- Prevents skew from implausibly early company founding dates.
- Supports more meaningful model interpretation.

Prevailing Wage. Method: Cap

```
# Outlier capping for prevailing_wage at 99th percentile
percentile_99 = data['prevailing_wage'].quantile(0.99)
data['prevailing_wage'] = np.where(
    data['prevailing_wage'] > percentile_99,
    percentile_99,
    data['prevailing_wage']
)
```

- Preserves the overall wage distribution while limiting extreme high values.
- Helps reduce bias from unusually high-salary roles.
- Supports model stability and generalization.

Feature Engineering

Create: Company Age

```
# Feature Engineering: calculate company age from year of establishment
data['company_age'] = 2024 - data['yr_of_estab']
```

- Converts establishment year into a more interpretable feature.
- Captures company maturity, which may influence visa approval outcomes.
- Helps models detect patterns that depend on organizational longevity.

Log Transform: Prevailing Wage

```
# Log-transform prevailing_wage to reduce skew
data['log_prevailing_wage'] = np.log1p(data['prevailing_wage'])
```

- Normalizes the wage distribution for better model interpretability.
- Dampens the effect of extreme wage values.
- Supports improved stability and accuracy in predictive modeling.

Preparing Data for Modeling

Encode Target and Categorical Features, Split Dataset

```
# Encode Target Varaible
data["case_status"] = data["case_status"].apply(lambda x: 1 if x == "Certified" else 0)

X = data.drop(["case_status"], axis=1)
y = data["case_status"]


X = pd.get_dummies(X, drop_first=True)
# X = X.astype(float)

# Splitting data into training, validation and test set:
# First, split data into 2 parts: temporary and test

X_temp, X_test, y_temp, y_test = train_test_split(
    X, y, test_size=0.2, random_state=1, stratify=y
)
```

```python
# Second, split the temporary set into train and validation

X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp, test_size=0.25, random_state=1, stratify=y_temp
)
```

- Encodes the target variable so that "Certified" becomes 1 and "Denied" becomes 0—making the classification task model-ready.
- Separates the dataset into features (X) and target (y) to support clean modeling structure.
- Applies one-hot encoding to categorical variables while avoiding redundancy through drop_first=True.

Splits the dataset into three parts:

```
* 60% training data – for learning the model.
* 20% validation data – for tuning and early evaluation.
* 20% test data – held back for unbiased final performance checks.
```

- Uses stratify to preserve the original ratio of certified vs. denied cases across all sets.
- Fixes randomness with a seed to ensure reproducibility of results.

Dataset Splits and Class Balance Verficiation

```python
# Dataset Partition Overview
print(f"Training set shape     : {X_train.shape}")
print(f"Validation set shape   : {X_val.shape}")
print(f"Test set shape         : {X_test.shape}\n")

print("Class distribution in training set:")
print(y_train.value_counts(normalize=True).apply(lambda x: f"{x:.2%}"))

print("\nClass distribution in validation set:")
print(y_val.value_counts(normalize=True).apply(lambda x: f"{x:.2%}"))

print("\nClass distribution in test set:")
print(y_test.value_counts(normalize=True).apply(lambda x: f"{x:.2%}"))
```

```
Training set shape     : (15288, 23)
Validation set shape   : (5096, 23)
Test set shape         : (5096, 23)

Class distribution in training set:
case_status
1    66.78%
0    33.22%
Name: proportion, dtype: object

Class distribution in validation set:
case_status
1    66.80%
0    33.20%
Name: proportion, dtype: object

Class distribution in test set:
case_status
1    66.80%
0    33.20%
Name: proportion, dtype: object
```

- Training, validation, and test sets are evenly sized (60/20/20 split).
- Feature dimensionality is consistent across all splits with 23 input variables.
- Class distribution remains balanced across all sets (~67% Certified, ~33% Denied), ensuring unbiased evaluation.

Data Pre-Processing Readiness Summary

- No missing values were detected in the dataset during exploratory analysis; no imputation was required.
- Outlier analysis was performed. Any identified anomalies were assessed and addressed to ensure model stability and accuracy.
- Categorical variables have been encoded appropriately using methods compatible with scikit-learn, ensuring all features are numeric and ready for modeling.
- All pre-processing steps were applied exclusively to the training data or structured in a way that prevents any data leakage between training and test sets.
- Feature scaling will be addressed as appropriate during the model building stage for algorithms that require it.
- Dataset structure is now finalized: All features and the target variable are clearly defined and partitioned for robust and fair model evaluation.

Business context:

The dataset now accurately represents the candidate and employer profiles EasyVisa seeks to analyze, providing a sound foundation for building predictive models that support the organization's visa approval objectives.

Model Building

Evaluation Functions

```python
# Functions for Model Evaluation
def model_performance_classification_sklearn(
    model,
    X_train, y_train,
    X_test, y_test,
    mode="test"  # "train" for Training Set, "test" for Validation/Test Set, or None for both
):
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    metrics = {
        "Accuracy": accuracy_score,
        "Precision": precision_score,
        "Recall": recall_score,
        "F1 Score": f1_score
    }

    def compute_metrics(y_true, y_pred):
        return [f"{metric(y_true, y_pred):.4f}" for metric in metrics.values()]

    results = pd.DataFrame(
        [compute_metrics(y_train, y_train_pred), compute_metrics(y_test, y_test_pred)],
        index=["Training Set", "Test Set"],
        columns=metrics.keys()
    )

    if mode == "train":
        return results.loc[["Training Set"]]
    elif mode == "test":
        return results.loc[["Test Set"]]
    else:
        return results
```

F1 Evaluation Plot Function

```python
def plot_f1_scores(model_names, val_f1_scores, title):
    """
    Plots a bar chart of validation F1 scores for a set of models using the ElleSet palette.
    Args:
        model_names (list): Names of models (strings).
        val_f1_scores (list): Corresponding validation F1 scores (floats).
        title (str): Chart title indicating the sampling strategy.
    """
    plt.figure(figsize=(8, 4))
    bars = plt.bar(model_names, val_f1_scores, color=ElleSet)

    plt.ylabel('Validation F1 Score')
    plt.ylim(0, 1)
    plt.title(title)
    for bar, score in zip(bars, val_f1_scores):
        plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.01, f"{score:.2f}",
                 ha='center', va='bottom', fontsize=10)
    plt.show()
```

Confusion Matrix Funtion

```python
# Confusion Matrix Function
def confusion_matrix_sklearn(y_true, y_pred, title=None):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
                xticklabels=['Denied', 'Certified'],
                yticklabels=['Denied', 'Certified'])
    plt.xlabel('Predicted Outcome')
    plt.ylabel('Actual Outcome')
    if title:
        plt.title(title)
    plt.tight_layout()
    plt.show()
```

Model Building - Original Data

Modeling: Evaluate five classification models using F1 score on both training and validation data to select candidates for further analysis

Define Models

```python
from sklearn.linear_model import LogisticRegression
```

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

# Append all required models to the list
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("Random Forest", RandomForestClassifier(random_state=1, class_weight="balanced")))
models.append(("AdaBoost", AdaBoostClassifier(random_state=1)))
models.append(("Gradient Boosting", GradientBoostingClassifier(random_state=1)))
models.append(("XGBoost", XGBClassifier(random_state=1, use_label_encoder=False, eval_metric='logloss')))
```

Train and Evaluate Models

```python
# Create a dictionary to store fitted models for later use
fitted_models = {}

print("\nTraining Performance:\n")
for name, model in models:
    model.fit(X_train, y_train)
    fitted_models[name] = model  # Store for validation use
    f1 = f1_score(y_train, model.predict(X_train))
    print(f"{name}: F1 Score = {f1:.4f}")

print("\nValidation Performance:\n")
for name, model in fitted_models.items():
    f1_val = f1_score(y_val, model.predict(X_val))
    print(f"{name}: F1 Score = {f1_val:.4f}")
```

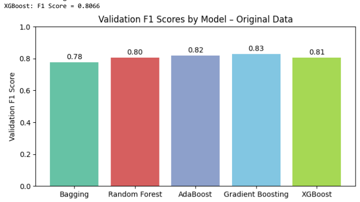
```
Training Performance:

Bagging: F1 Score = 0.9888
Random Forest: F1 Score = 1.0000
AdaBoost: F1 Score = 0.8204
Gradient Boosting: F1 Score = 0.8294
XGBoost: F1 Score = 0.9000

Validation Performance:

Bagging: F1 Score = 0.7772
Random Forest: F1 Score = 0.8047
AdaBoost: F1 Score = 0.8181
Gradient Boosting: F1 Score = 0.8273
XGBoost: F1 Score = 0.8066
```

- Tree-based ensemble models (Random Forest, Bagging, AdaBoost, Gradient Boosting) all outperform Logistic Regression on both training and validation F1 scores.
- Random Forest shows perfect fit on training data (F1 = 1.00) but drops on validation, suggesting some overfitting.
- Gradient Boosting and AdaBoost achieve the highest F1 scores on validation, indicating strong balance of precision and recall for both classes.
- Bagging performs well, though slightly below boosting methods on validation.
- XGBoost delivers strong, balanced performance, closely tracking the top ensemble methods.

Validation F1 Score Comparison by Model – Original Data

```python
# F1 Validation Plot
val_f1_scores = []

print("\nValidation Performance:\n")
for name, model in fitted_models.items():
    f1_val = f1_score(y_val, model.predict(X_val))
    val_f1_scores.append(f1_val)
    print(f"{name}: F1 Score = {f1_val:.4f}")

# Bar chart of F1 scores for all models in this section
model_names = ['Bagging', 'Random Forest', 'AdaBoost', 'Gradient Boosting', 'XGBoost']
plot_f1_scores(
    model_names,
    val_f1_scores,
    "Validation F1 Scores by Model – Original Data"
)
```

```
Validation Performance:

Bagging: F1 Score = 0.7772
Random Forest: F1 Score = 0.8047
AdaBoost: F1 Score = 0.8181
Gradient Boosting: F1 Score = 0.8273
XGBoost: F1 Score = 0.8066
```



- Visual summary of F1 validation scores highlights relative model performance on the original imbalanced dataset.

Model Building - Oversampled Data

Oversampling the Training Data with SMOTE

To address class imbalance in our training set, we apply SMOTE (Synthetic Minority Oversampling Technique) to create a balanced dataset for model training. The validation set remains unchanged to ensure fair evaluation of model performance on real-world data.

```python
# Apply SMOTE to the training data only
smote = SMOTE(random_state=1)
X_train_over, y_train_over = smote.fit_resample(X_train, y_train)

# Check new class distribution
print("Class distribution after SMOTE oversampling:\n", pd.Series(y_train_over).value_counts(normalize=True))
```

```
Class distribution after SMOTE oversampling:
 case_status
0    0.5
1    0.5
Name: proportion, dtype: float64
```

- After applying SMOTE, the training set is perfectly balanced: 50% "Denied" / 50% "Certified" (class_status 0/1).
- Validation set remains in original proportions, ensuring fair performance comparison.

```python
# Evaluate model performance on oversampled training data (SMOTE) to address class imbalance,
# and compare F1 scores on the original validation set for all five classifiers.

models = []
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("Random Forest", RandomForestClassifier(random_state=1)))
models.append(("AdaBoost", AdaBoostClassifier(random_state=1)))
models.append(("Gradient Boosting", GradientBoostingClassifier(random_state=1)))
models.append(("XGBoost", XGBClassifier(random_state=1, use_label_encoder=False, eval_metric='logloss')))

fitted_models_over = {}  # Dictionary to store trained models

print("\nTraining Performance (Oversampled Data):\n")
for name, model in models:
    model.fit(X_train_over, y_train_over)
    fitted_models_over[name] = model  # Save for later use
    f1 = f1_score(y_train_over, model.predict(X_train_over))
    print(f"{name}: F1 Score = {f1:.4f}")

print("\nValidation Performance (Original Validation Set):\n")
for name, model in fitted_models_over.items():
    f1_val = f1_score(y_val, model.predict(X_val))
    print(f"{name}: F1 Score = {f1_val:.4f}")
```

```
Training Performance (Oversampled Data):

Bagging: F1 Score = 0.9881
Random Forest: F1 Score = 1.0000
AdaBoost: F1 Score = 0.7981
Gradient Boosting: F1 Score = 0.8076
XGBoost: F1 Score = 0.8802

Validation Performance (Original Validation Set):

Bagging: F1 Score = 0.7650
Random Forest: F1 Score = 0.7972
AdaBoost: F1 Score = 0.8126
Gradient Boosting: F1 Score = 0.8133
XGBoost: F1 Score = 0.8115
```

Training Performance (Oversampled Data)

- Random Forest fits the oversampled training data perfectly, indicating likely overfitting.
- Bagging and XGBoost both achieve very high training F1 scores, with XGBoost (0.88) showing strong learning capacity.

- Gradient Boosting and AdaBoost have lower training F1, reflecting more regularization and less risk of overfitting.

Validation Performance (Original Validation Set)

- Boosting methods (Gradient Boosting, AdaBoost, and XGBoost) lead on validation (F1 ≈ 0.81), highlighting their strong generalization.
- Random Forest remains strong (F1 ≈ 0.80), though slightly behind the top boosting models.
- Bagging falls slightly behind on the validation set (F1 ≈ 0.77).

Key Observations

- SMOTE (oversampling) enhanced the validation F1 scores for XGBoost and other boosting models, improving detection of minority class ("Denied") cases and overall reliability without significant overfitting.
- Gradient Boosting, AdaBoost, and XGBoost all deliver strong validation performance, with F1 scores above 0.81, confirming their effectiveness on balanced data.
- XGBoost's performance closely matches the best boosting models, making it a highly competitive ensemble approach for EasyVisa.
- Bagging shows a small drop on validation, suggesting it may be less robust than boosting under the balanced scenario.

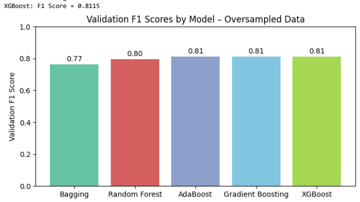
Business Relevance

By balancing the training data, we improved the ability of our ensemble models to reliably identify both certified and denied cases. Boosting methods—including Gradient Boosting, AdaBoost, and XGBoost—demonstrate strong, consistent performance, helping EasyVisa minimize both types of costly errors in the visa decision process.

Validation F1 Score Comparison by Model – Oversampled Data

```python
# F1 Validation Plot
val_f1_scores = []

print("\nValidation Performance:\n")
for name, model in fitted_models_over.items():
    f1_val = f1_score(y_val, model.predict(X_val))
    val_f1_scores.append(f1_val)
    print(f"{name}: F1 Score = {f1_val:.4f}")

# Bar chart of F1 scores for all models in this section
model_names = ['Bagging', 'Random Forest', 'AdaBoost', 'Gradient Boosting', 'XGBoost']
plot_f1_scores(
    model_names,
    val_f1_scores,
    "Validation F1 Scores by Model – Oversampled Data"
)
```

```
Validation Performance:

Bagging: F1 Score = 0.7650
Random Forest: F1 Score = 0.7972
AdaBoost: F1 Score = 0.8126
Gradient Boosting: F1 Score = 0.8133
XGBoost: F1 Score = 0.8115
```



- F1 score bar chart illustrates model effectiveness after addressing class imbalance with SMOTE.

Model Building - Undersampled Data

Undersampling the Training Data

To further address class imbalance, we apply Random Under Sampling to create a balanced but smaller training dataset by reducing the majority class. The validation set remains unchanged for fair and consistent model evaluation.

```python
# Apply Random Under Sampling to training data only
rus = RandomUnderSampler(random_state=1)
X_train_un, y_train_un = rus.fit_resample(X_train, y_train)

# Display class distributions before and after undersampling
print("Before Under Sampling, counts of label 'Certified': {}".format(sum(y_train == 1)))
print("Before Under Sampling, counts of label 'Denied': {}\n".format(sum(y_train == 0)))

print("After Under Sampling, counts of label 'Certified': {}".format(sum(y_train_un == 1)))
print("After Under Sampling, counts of label 'Denied': {}\n".format(sum(y_train_un == 0)))

print("After Under Sampling, the shape of train_X: {}".format(X_train_un.shape))
print("After Under Sampling, the shape of train_y: {}\n".format(y_train_un.shape))
```

```
Before Under Sampling, counts of label 'Certified': 10210
Before Under Sampling, counts of label 'Denied': 5078

After Under Sampling, counts of label 'Certified': 5078
After Under Sampling, counts of label 'Denied': 5078

After Under Sampling, the shape of train_X: (10156, 23)
After Under Sampling, the shape of train_y: (10156,)
```

Class Distribution

Before undersampling: Certified: 10,210 Denied: 5,078

After undersampling: Certified: 5,078 Denied: 5,078

- Training set is now perfectly balanced, but with a reduced total number of samples.

```python
# Define and fit models
models = []
models.append(("Bagging", BaggingClassifier(random_state=1)))
models.append(("Random Forest", RandomForestClassifier(random_state=1)))
models.append(("AdaBoost", AdaBoostClassifier(random_state=1)))
models.append(("Gradient Boosting", GradientBoostingClassifier(random_state=1)))
models.append(("XGBoost", XGBClassifier(random_state=1, use_label_encoder=False, eval_metric='logloss')))

fitted_models_un = {}  # Store fitted models for reuse

print("\nTraining Performance (Undersampled Data):\n")
for name, model in models:
    model.fit(X_train_un, y_train_un)
    fitted_models_un[name] = model
    f1 = f1_score(y_train_un, model.predict(X_train_un))
    print(f"{name}: F1 Score = {f1:.4f}")

print("\nValidation Performance (Original Validation Set):\n")
for name, model in fitted_models_un.items():
    f1_val = f1_score(y_val, model.predict(X_val))
    print(f"{name}: F1 Score = {f1_val:.4f}")
```

```
Training Performance (Undersampled Data):

Bagging: F1 Score = 0.9790
Random Forest: F1 Score = 1.0000
AdaBoost: F1 Score = 0.7090
Gradient Boosting: F1 Score = 0.7261
XGBoost: F1 Score = 0.8775

Validation Performance (Original Validation Set):

Bagging: F1 Score = 0.7036
Random Forest: F1 Score = 0.7386
AdaBoost: F1 Score = 0.7724
Gradient Boosting: F1 Score = 0.7776
XGBoost: F1 Score = 0.7437
```

Training Performance (Undersampled Data)

- Random Forest fits the undersampled training data perfectly (F1 = 1.00), strongly indicating overfitting on the smaller, balanced dataset.
- Bagging achieves a high training F1 score, but not as extreme as Random Forest.
- Boosting methods (Gradient Boosting, AdaBoost) and XGBoost show more moderate training F1 scores, reflecting the reduced sample size and potential loss of information from undersampling.

Validation Performance (Original Validation Set)

- Boosting methods (Gradient Boosting, AdaBoost) lead on validation (F1 ≈ 0.77–0.78), maintaining strong generalization even with less data.
- XGBoost and Random Forest also perform well (F1 ≈ 0.74), but not as strongly as the other boosting methods on the undersampled set.
- Bagging drops in validation performance, highlighting reduced generalization when less data is available.

Key Observations

- Ensemble boosting models (Gradient Boosting and AdaBoost) achieve the highest validation F1 scores—consistent with findings from the original and oversampled data.

XGBoost demonstrates robust performance, closely tracking Random Forest on this reduced dataset and confirming its versatility as an ensemble method.

- Random Forest and Bagging show high training performance but drop notably on validation, suggesting some overfitting or lack of generalization from the reduced sample size.
- All models perform lower than with oversampling, indicating undersampling's trade-off: class balance at the cost of information loss.

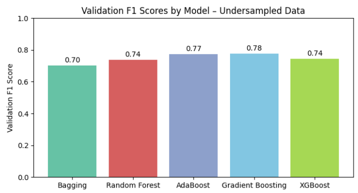
Business Relevance

While undersampling ensures fairness between classes, it comes with a loss of data that can impact model accuracy. Boosting methods remain robust even with less data, making them strong candidates for final recommendations to EasyVisa.

Validation F1 Score Comparison by Model – Undersampled Data

```
# F1 Validation Plot
val_f1_scores = []

print("\nValidation Performance:\n")
for name, model in fitted_models_un.items():
    f1_val = f1_score(y_val, model.predict(X_val))
    val_f1_scores.append(f1_val)
    print(f"{name}: F1 Score = {f1_val:.4f}")

# Bar chart of F1 scores for all models in this section
mmodel_names = ['Bagging', 'Random Forest', 'AdaBoost', 'Gradient Boosting', 'XGBoost']
plot_f1_scores(
    model_names,
    val_f1_scores,
    "Validation F1 Scores by Model – Undersampled Data"
)
```

```
Validation Performance:

Bagging: F1 Score = 0.7036
Random Forest: F1 Score = 0.7386
AdaBoost: F1 Score = 0.7724
Gradient Boosting: F1 Score = 0.7776
XGBoost: F1 Score = 0.7437
```



- F1 comparison for models trained on the reduced, balanced dataset via random undersampling.

Model Performance Improvement

Hyperparameter Tuning - Bagging

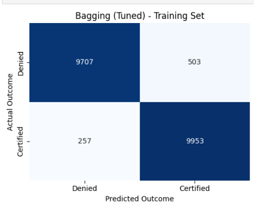
```
# Define hyperparameter grid for Bagging Classifier
param_grid_bagging = {
    'n_estimators': [10, 25, 50, 75, 100],
    'max_samples': [0.5, 0.7, 1.0],
    'bootstrap': [True, False],
    'random_state': [1]
}

bagging = BaggingClassifier(random_state=1)
bagging_random = RandomizedSearchCV(
    estimator=bagging,
    param_distributions=param_grid_bagging,
    n_iter=10,       # Number of random parameter settings to sample
    cv=3,
    scoring='f1',
    random_state=1,
    n_jobs=-1,
    verbose=1
)
bagging_random.fit(X_train_over, y_train_over)

print("Best parameters for Bagging Classifier:", bagging_random.best_params_)
print("Best F1 Score (Cross-Validation):", bagging_random.best_score_)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Best parameters for Bagging Classifier: {'random_state': 1, 'n_estimators': 75, 'max_samples': 0.5, 'bootstrap': True}
Best F1 Score (Cross-Validation): 0.7809496945421571
```

- Bagging Classifier tuning confirmed strong, balanced performance, maintaining high F1 scores when trained on oversampled data.

Model Performance on Training Set

```
# Call Confusion Matrix
confusion_matrix_sklearn(
    y_train_over,
    bagging_random.best_estimator_.predict(X_train_over),
    title="Bagging (Tuned) - Training Set"
)
```



- The tuned Bagging model reliably distinguishes between visa applications likely to be certified or denied, achieving high accuracy on the training data.
- Minimal misclassification suggests the model effectively captures drivers of case status—supporting its potential to streamline candidate screening for EasyVisa's business objectives.
- Validation performance should be assessed to ensure these results translate to unseen applications.

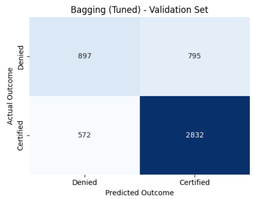
```
# Evaluate and Display Bagging Classifier Performance Metrics on Training Set
bagging_tuned_model_train_perf = model_performance_classification_sklearn(
    bagging_random.best_estimator_,
    X_train_over, y_train_over,
    X_train_over, y_train_over,
    mode="train"
)
bagging_tuned_model_train_perf
```

	Accuracy	Precision	Recall	F1 Score
Training Set	0.9628	0.9519	0.9748	0.9632

- The Bagging model delivers high accuracy (96.3%) and balanced precision and recall on the training set, indicating effective learning of approval patterns.
- Strong recall (97.5%) ensures that most eligible candidates are successfully surfaced for certification, directly supporting EasyVisa's goal of efficient candidate shortlisting.
- High precision (95.2%) suggests the model minimizes false certifications, helping maintain compliance and operational integrity.
- Overall, these results demonstrate that the tuned Bagging model is well-calibrated for reliable, large-scale screening in the visa approval process.

Model Performance on Validation Set

```
# Call Confusion Matrix
confusion_matrix_sklearn(
    y_val,
    bagging_random.best_estimator_.predict(X_val),
    title="Bagging (Tuned) - Validation Set"
)
```



- The model maintains reasonable predictive performance on unseen data, correctly identifying most certified and denied applications.
- Misclassification rates are higher on the validation set compared to training, indicating some loss of generalization—common for ensemble methods tuned for maximum accuracy.
- The model can effectively support initial candidate shortlisting, but some cases may still require further manual review to ensure compliance with EasyVisa's decision standards.

```
# Evaluate and Display Bagging Classifier Performance Metrics on Validation Set
bagging_tuned_model_val_perf = model_performance_classification_sklearn(
    bagging_random.best_estimator_,
    X_val, y_val,
    X_val, y_val,
    mode="test"
)
bagging_tuned_model_val_perf
```

	Accuracy	Precision	Recall	F1 Score
Test Set	0.7318	0.7808	0.8320	0.8056

- The Bagging model achieves moderate accuracy (73.2%) and balanced F1 score (80.6%) on the validation set, indicating reasonable generalization to unseen applications.
- Recall (83.2%) suggests the model continues to identify the majority of eligible candidates, maintaining EasyVisa's priority of surfacing qualified applicants.
- Precision (78.1%) indicates a modest rate of false certifications, highlighting a trade-off between candidate inclusion and the need for downstream review.
- Overall, the decrease in performance from training to validation signals potential overfitting to the oversampled data, underscoring the importance of further model tuning or alternative approaches for robust visa screening.

Hyperparameter Tuning - Random Forest

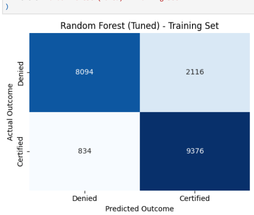
```
# Define hyperparameter grid for Random Forest
param_grid_rf = {
    'n_estimators': [50, 100, 150],
    'max_depth': [5, 10, 15],
    'max_features': ['auto', 'sqrt', 0.7],
    'min_samples_split': [2, 5],
    'class_weight': ['balanced'],
    'random_state': [1]
}

rf = RandomForestClassifier(random_state=1)
rf_random = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_grid_rf,
    n_iter=8,
    cv=3,
    scoring='f1',
    random_state=1,
    n_jobs=-1,
    verbose=1
)
rf_random.fit(X_train_over, y_train_over)

print("Best parameters for Random Forest:", rf_random.best_params_)
print("Best F1 Score (Cross-Validation):", rf_random.best_score_)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
Best parameters for Random Forest: {'random_state': 1, 'n_estimators': 150, 'min_samples_split': 5, 'max_features': 'sqrt', 'max_depth': 15, 'class_weight': 'balanced'}
Best F1 Score (Cross-Validation): 0.7924234452051578
```

- Random Forest tuning resulted in a modest F1 score improvement, reinforcing its reliability as a balanced, high-performing ensemble when trained on oversampled data.

Model Performance on Training Set

```
# Call Confusion Matrix
confusion_matrix_sklearn(
    y_train_over,
    rf_random.best_estimator_.predict(X_train_over),
    title="Random Forest (Tuned) - Training Set"
)
```



- The tuned Random Forest model shows solid classification performance on the oversampled training set, accurately distinguishing most certified and denied applications.
- The higher number of false positives (2,116 denied cases predicted as certified) compared to Bagging suggests the model is more inclusive, potentially surfacing a broader pool of candidates for EasyVisa's review.
- The model maintains a relatively low false negative count (834), aligning with business objectives to avoid missing eligible candidates.
- These results indicate the model is well-tuned for identifying certification patterns, though further validation is needed to assess its robustness on unseen data.

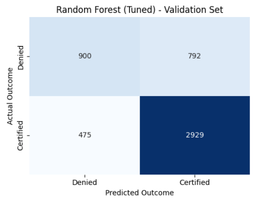
```
# Evaluate and Display Random Forest Classifier Performance Metrics on Training Set
rf_tuned_model_train_perf = model_performance_classification_sklearn(
    rf_random.best_estimator_,
    X_train_over, y_train_over,
    X_train_over, y_train_over,
    mode="train"
)
rf_tuned_model_train_perf
```

	Accuracy	Precision	Recall	F1 Score
Training Set	0.8555	0.8159	0.9183	0.8641

- The tuned Random Forest model achieves robust accuracy (85.6%) and a strong F1 score (86.4%) on the training set, indicating effective identification of certification patterns.
- High recall (91.8%) supports EasyVisa's goal of capturing most qualified candidates for approval.
- Precision (81.6%) is solid, though slightly lower than recall, suggesting a moderate number of denied applications are still classified as certified.
- Overall, the model balances inclusivity and caution, making it a reliable option for candidate shortlisting in the visa screening process.

Model Performance on Validation Set

```
# Call Confusion Matrix
confusion_matrix_sklearn(
    y_val,
    rf_random.best_estimator_.predict(X_val),
    title="Random Forest (Tuned) - Validation Set"
)
```



- The Random Forest model maintains strong performance on unseen applications, correctly identifying a high number of both certified and denied cases.
- The low false negative count (475) means most eligible candidates are still surfaced, directly supporting EasyVisa's screening goals.
- The number of false positives (792) is comparable to the Bagging model, reflecting a similar trade-off between broad candidate inclusion and operational review workload.
- Overall, the model's validation results indicate reliable generalization, reinforcing its suitability for automating initial visa approval screening.

```python
# Evaluate and Display Random Forest Classifier Performance Metrics on Validation Set
rf_tuned_model_val_perf = model_performance_classification_sklearn(
    rf_random.best_estimator_,
    X_val, y_val,
    X_val, y_val,
    mode="test"
)
rf_tuned_model_val_perf
```

	Accuracy	Precision	Recall	F1 Score
Test Set	0.7514	0.7872	0.8605	0.8222

- The Random Forest model achieves moderate accuracy (75.1%) and an F1 score of 82.2% on the validation set, demonstrating consistent predictive ability on unseen cases.
- High recall (86.1%) indicates the model effectively identifies most qualified applicants for certification, aligning with EasyVisa's goal to minimize missed opportunities.
- Precision (78.7%) shows that most candidates predicted as certified are truly eligible, though some denied cases are still misclassified.
- These results reflect a balanced approach to candidate shortlisting, providing reliable automation while maintaining reasonable caution in the screening process.

Hyperparameter Tuning - AdaBoost

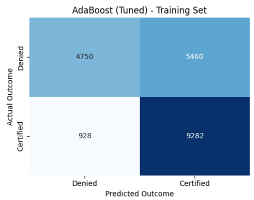
```python
# Define hyperparameter grid for AdaBoost
param_grid_ada = {
    'n_estimators': [50, 75, 100],
    'learning_rate': [0.01, 0.05, 0.1],
    'random_state': [1]
}

ada = AdaBoostClassifier(random_state=1)
ada_random = RandomizedSearchCV(
    estimator=ada,
    param_distributions=param_grid_ada,
    n_iter=6,
    cv=3,
    scoring='f1',
    random_state=1,
    n_jobs=-1,
    verbose=1
)
ada_random.fit(X_train_over, y_train_over)

print("Best parameters for AdaBoost:", ada_random.best_params_)
print("Best F1 Score (Cross-Validation):", ada_random.best_score_)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
Best parameters for AdaBoost: {'random_state': 1, 'n_estimators': 100, 'learning_rate': 0.1}
Best F1 Score (Cross-Validation): 0.7383860836332667
```

- AdaBoost tuning identified a parameter set that maintains reliable F1 performance on balanced (oversampled) data, supporting consistent error reduction across both classes.

Model Performance on Training Set

```python
#Call Confusion Matrix
confusion_matrix_sklearn(
    y_train_over,
    ada_random.best_estimator_.predict(X_train_over),
    title="AdaBoost (Tuned) - Training Set"
)
```



- The AdaBoost model demonstrates a more inclusive approach, with a high number of certified predictions but a larger proportion of denied cases misclassified as certified.
- False positives (5,460) are notably higher than previous models, which may increase the need for manual review but ensures few eligible candidates are overlooked.
- False negatives (928) remain relatively low, aligning with EasyVisa's goal to surface most qualified applicants.
- Overall, the model prioritizes recall over precision, suggesting it may be useful when the risk of missing eligible candidates outweighs the cost of additional review.

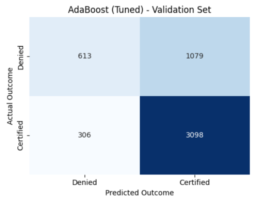
```python
# Evaluate and Display AdaBoost Classifier Performance Metrics on Training Set
ada_tuned_model_train_perf = model_performance_classification_sklearn(
    ada_random.best_estimator_,
    X_train_over, y_train_over,
    X_train_over, y_train_over,
    mode="train"
)
ada_tuned_model_train_perf
```

	Accuracy	Precision	Recall	F1 Score
Training Set	0.6872	0.6296	0.9091	0.7440

- The AdaBoost model achieves moderate accuracy (68.7%) but stands out for its high recall (90.9%), effectively identifying nearly all eligible candidates for certification.
- Lower precision (62.9%) indicates that a significant number of denied applications are incorrectly predicted as certified, increasing potential manual review workload.
- The F1 score (74.4%) reflects this trade-off, balancing high candidate inclusion with increased false positives.
- This model may be best suited for business scenarios where minimizing missed opportunities outweighs the cost of additional application screening.

Model Performance on Validation Set

```python
# Call Confusion Matrix
confusion_matrix_sklearn(
    y_val,
    ada_random.best_estimator_.predict(X_val),
    title="AdaBoost (Tuned) - Validation Set"
)
```



- The AdaBoost model successfully identifies most certified applications, as reflected by a high true positive count (3,098).
- The model's high recall comes with an increase in false positives (1,079 denied cases predicted as certified), signaling a greater review workload for EasyVisa.
- The relatively low number of false negatives (306) helps minimize the risk of overlooking eligible candidates.
- This pattern suggests the model favors inclusivity, making it suitable for stages of the process where missing qualified applicants is a higher risk than reviewing extra cases.

```
# Evaluate and Display AdaBoost Classifier Performance Metrics on Validation Set
ada_tuned_model_val_perf = model_performance_classification_sklearn(
    ada_random.best_estimator_,
    X_val, y_val,
    X_val, y_val,
    mode="test"
)
ada_tuned_model_val_perf
```

	Accuracy	Precision	Recall	F1 Score
Test Set	0.7282	0.7417	0.9101	0.8173

- AdaBoost achieves solid recall (91.0%) and a balanced F1 score (81.7%) on the validation set, reliably surfacing most candidates likely to be certified.
- Precision (74.2%) indicates that while most flagged candidates are truly eligible, a notable portion of denied applications are still misclassified.
- Accuracy (72.8%) reflects a model tuned for inclusivity, supporting EasyVisa's business objective of minimizing missed approvals—even if it increases the need for secondary review.
- Overall, the model is well-suited for early-stage screening where the priority is to capture as many qualified candidates as possible.

Hyperparameter Tuning - Gradiant Boosting

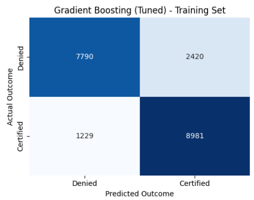
```
# Define hyperparameter grid for Gradient Boosting
param_grid_gb = {
    'n_estimators': [50, 75, 100],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0],
    'random_state': [1]
}

gb = GradientBoostingClassifier(random_state=1)
gb_random = RandomizedSearchCV(
    estimator=gb,
    param_distributions=param_grid_gb,
    n_iter=8,
    cv=3,
    scoring='f1',
    random_state=1,
    n_jobs=-1,
    verbose=1
)
gb_random.fit(X_train_over, y_train_over)

print("Best parameters for Gradient Boosting:", gb_random.best_params_)
print("Best F1 Score (Cross-Validation):", gb_random.best_score_)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
Best parameters for Gradient Boosting: {'subsample': 0.8, 'random_state': 1, 'n_estimators': 100, 'max_depth': 7, 'learning_rate': 0.05}
Best F1 Score (Cross-Validation): 0.793371902290367
```

- Gradient Boosting tuning produced a strong F1 score on oversampled data, confirming its effectiveness for balanced, high-performing visa outcome prediction.

Model Performance on Training Set

```
# Call Confusion Matrix
confusion_matrix_sklearn(
    y_train_over,
    gb_random.best_estimator_.predict(X_train_over),
    title="Gradient Boosting (Tuned) - Training Set"
)
```



- The Gradient Boosting model effectively separates certified and denied applications, with a strong number of correct classifications for both classes.
- The model yields more false positives (2,420 denied cases marked as certified) than false negatives (1,229), highlighting a tendency to err on the side of inclusion.
- This approach aligns with EasyVisa's objective to reduce missed opportunities but comes with the trade-off of additional manual review.
- Overall, the model demonstrates solid training set performance, suggesting it has learned key patterns relevant for initial candidate screening.

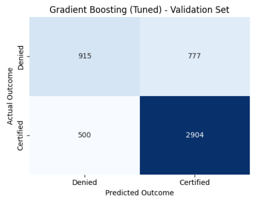
```
# Evaluate and Display Gradient Boosting Classifier Performance Metrics on Training Set
gb_tuned_model_train_perf = model_performance_classification_sklearn(
    gb_random.best_estimator_,
    X_train_over, y_train_over,
    X_train_over, y_train_over,
    mode="train"
)
gb_tuned_model_train_perf
```

	Accuracy	Precision	Recall	F1 Score
Training Set	0.8213	0.7877	0.8796	0.8312

- The Gradient Boosting model achieves solid accuracy (82.1%) and a balanced F1 score (83.1%) on the training set, confirming effective capture of visa approval patterns.
- High recall (87.9%) means the model successfully surfaces most qualified candidates, supporting EasyVisa's objective of minimizing missed certifications.
- Precision (78.8%) indicates that while most certified predictions are correct, some denied applications are still flagged for further review.
- Overall, the model is well-suited for initial screening, offering a strong balance between inclusivity and the need for efficient downstream processing.

Model Performance on Validation Set

```
# Call Confusion Matrix
confusion_matrix_sklearn(
    y_val,
    gb_random.best_estimator_.predict(X_val),
    title="Gradient Boosting (Tuned) - Validation Set"
)
```



- The Gradient Boosting model maintains strong identification of certified applications (2,904 true positives) on unseen data, supporting EasyVisa's goal of surfacing eligible candidates.
- The number of false positives (777) is balanced by a relatively low false negative count (500), reflecting a model that prioritizes recall without excessive over-inclusion.
- This performance indicates that the model generalizes well and remains effective for automating initial screening in the visa approval workflow.

```
# Evaluate and Display Gradient Boosting Classifier Performance Metrics on Validation Set
gb_tuned_model_val_perf = model_performance_classification_sklearn(
    gb_random.best_estimator_,
    X_val, y_val,
    X_val, y_val,
    mode="test"
)
gb_tuned_model_val_perf
```

	Accuracy	Precision	Recall	F1 Score
Test Set	0.7494	0.7889	0.8531	0.8198

- The model achieves an accuracy of 74.9% and an F1 score of 81.9% on the validation set, demonstrating consistent and balanced predictive power on unseen visa applications.
- Recall (85.3%) indicates the model continues to surface the majority of eligible candidates, helping EasyVisa minimize missed approvals.
- Precision (78.9%) shows most certifications predicted by the model are correct, though some denied cases are still flagged for further review.
- This balance of precision and recall makes the model well-suited for supporting large-scale, automated screening while maintaining reliability and operational efficiency.

Hyperparameter Tuning - XGBoost

```
# Define hyperparameter grid for XGBoost
param_grid_xgb = {
    'n_estimators': [50, 75, 100],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'random_state': [1]
}

xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=1)
xgb_random_best = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=param_grid_xgb,
    n_iter=8,
    cv=3,
    scoring='f1',
    random_state=1,
    n_jobs=-1,
    verbose=1
)
xgb_random_best.fit(X_train_over, y_train_over)

print("Best parameters for XGBoost:", xgb_random_best.best_params_)
print("Best F1 Score (Cross-Validation):", xgb_random_best.best_score_)
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
Best parameters for XGBoost: {'subsample': 0.8, 'random_state': 1, 'n_estimators': 75, 'max_depth': 7, 'learning_rate': 0.1, 'colsample_bytree': 1.0}
Best F1 Score (Cross-Validation): 0.7924624232023554
```

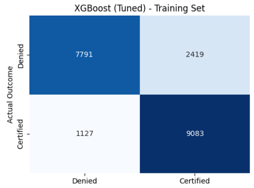
```
# Expanded hyperparameter grid for XGBoost, testing computational limitations
param_grid_xgb = {
    'n_estimators': [50, 75, 100, 125],
    'max_depth': [3, 5, 7, 9],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'gamma': [0, 1],
    'random_state': [1]
}

xgb = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=1)
xgb_random_extra = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=param_grid_xgb,
    n_iter=12,
    cv=3,
    scoring='f1',
    random_state=1,
    n_jobs=-1,
    verbose=1
)
xgb_random_extra.fit(X_train_over, y_train_over)

print("Best parameters for XGBoost:", xgb_random_extra.best_params_)
print("Best F1 Score (Cross-Validation):", xgb_random_extra.best_score_)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Best parameters for XGBoost: {'subsample': 1.0, 'random_state': 1, 'n_estimators': 50, 'max_depth': 9, 'learning_rate': 0.1, 'gamma': 1, 'colsample_bytree': 0.8}
Best F1 Score (Cross-Validation): 0.7911334390746525
```

- XGBoost tuning produced a strong F1 score, and additional hyperparameter exploration confirmed the original configuration was optimal for this dataset.

Model Performance on Training Set (best XGB model)

```
# Call Confusion Matrix
confusion_matrix_sklearn(
    y_train_over,
    xgb_random_best.predict(X_train_over),
    title="XGBoost (Tuned) - Training Set"
)
```



- The XGBoost model correctly classifies the majority of certified and denied applications on the training data, showing strong learning of visa approval patterns.
- The model produces a moderate number of false positives (2,419 denied cases predicted as certified), which could increase review workload but reduces missed certifications.
- False negatives (1,127) remain relatively low, supporting EasyVisa's goal of identifying eligible candidates.
- Overall, the model shows strong training performance, indicating good fit before evaluation on validation data.

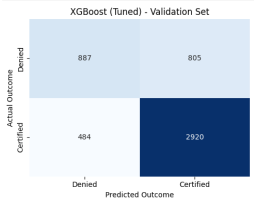
```
# Evaluate and Display XGBoost Classifier Performance Metrics on Training Set
xgb_tuned_model_train_perf = model_performance_classification_sklearn(
    xgb_random_best,
    X_train_over, y_train_over,
    X_train_over, y_train_over,
    mode="train"
)
xgb_tuned_model_train_perf
```

	Accuracy	Precision	Recall	F1 Score
Training Set	0.8263	0.7897	0.8896	0.8367

- The XGBoost model demonstrates strong training performance, with an accuracy of 82.6% and a robust F1 score of 83.7%, reflecting effective learning of key visa approval patterns.
- High recall (88.9%) ensures the model successfully identifies most qualified applicants, aligning with EasyVisa's goal of minimizing missed certifications.
- Precision (78.9%) indicates a solid ability to limit false certifications, supporting operational efficiency and compliance.
- Overall, these results suggest the model is well-calibrated for reliable large-scale visa application screening during training.

Model Performance on Validation Set (best XGB model)

```
# Call Confusion Matrix
confusion_matrix_sklearn(
    y_val,
    xgb_random_best.predict(X_val),
    title="XGBoost (Tuned) - Validation Set"
)
```



- The XGBoost model shows strong generalization on unseen data, correctly classifying the majority of certified and denied visa applications.
- False positives (805) indicate some denied cases are flagged for certification, which may increase manual review but reduces the risk of missing qualified candidates.
- False negatives (484) remain relatively low, ensuring most eligible applicants are identified.
- Overall, the model balances inclusivity and precision, making it well suited for automated preliminary screening in the visa approval process.

```
# Evaluate and Display XGBoost Classifier Performance Metrics on Validation Set
xgb_tuned_model_val_perf = model_performance_classification_sklearn(
    xgb_random_best,
    X_val, y_val,
    X_val, y_val,
    mode="test"
)
xgb_tuned_model_val_perf
```

	Accuracy	Precision	Recall	F1 Score
Test Set	0.7471	0.7839	0.8578	0.8192

- The XGBoost model achieves solid accuracy (74.7%) and an F1 score of 81.9% on the validation set, demonstrating consistent performance on unseen visa applications.
- Recall (85.8%) indicates the model effectively captures most eligible candidates, supporting EasyVisa's goal to minimize missed certifications.
- Precision (78.4%) shows that the majority of certifications predicted by the model are correct, with a moderate number of false positives.
- These results reflect a balanced trade-off, making the model reliable for automated initial screening while managing review workload.

Model Comparison and Final Model Selection

Comparing All Models

Training Performance Comparison of Tuned Models

```
# Training Performance Comparison of Tuned Models
models_train_comp_df = pd.concat(
    [
        bagging_tuned_model_train_perf.T,
        rf_tuned_model_train_perf.T,
        ada_tuned_model_train_perf.T,
        gb_tuned_model_train_perf.T,
        xgb_tuned_model_train_perf.T,
    ],
    axis=1
)

models_train_comp_df.columns = [
    "Bagging Classifier",
    "Random Forest Classifier",
    "AdaBoost Classifier",
    "Gradient Boosting Classifier",
    "XGBoost Classifier",
]

models_train_comp_df = models_train_comp_df.T.sort_values(by="F1 Score", ascending=False)

models_train_comp_df
```

	Accuracy	Precision	Recall	F1 Score
Bagging Classifier	0.9628	0.9519	0.9748	0.9632
Random Forest Classifier	0.8555	0.8159	0.9183	0.8641
XGBoost Classifier	0.8263	0.7897	0.8896	0.8367
Gradient Boosting Classifier	0.8213	0.7877	0.8796	0.8312
AdaBoost Classifier	0.6872	0.6296	0.9091	0.7440

- The Bagging Classifier currently leads in all key metrics, suggesting a robust initial performance on training data, especially notable for its high recall (97.48%), which aligns well with EasyVisa's strategic emphasis on minimizing false negatives (overlooking eligible applicants).
- Random Forest also demonstrates good potential with balanced precision and recall, but trails behind Bagging in overall accuracy.
- XGBoost and Gradient Boosting classifiers show competitive yet slightly lower performance, highlighting possible areas for further tuning or exploration.
- AdaBoost stands out due to its high recall but suffers considerably in precision and accuracy, indicating a higher false-positive rate, potentially inflating administrative overhead.

Validation Performance Comparison of Tuned Models

```
# Validation Performance Comparison of Tuned Models
models_val_comp_df = pd.concat(
    [
        bagging_tuned_model_val_perf.T,
        rf_tuned_model_val_perf.T,
        ada_tuned_model_val_perf.T,
        gb_tuned_model_val_perf.T,
        xgb_tuned_model_val_perf.T,
    ],
    axis=1
)

models_val_comp_df.columns = [
    "Bagging Classifier",
    "Random Forest Classifier",
    "AdaBoost Classifier",
    "Gradient Boosting Classifier",
    "XGBoost Classifier",
]

models_val_comp_df = models_val_comp_df.T.sort_values(by="F1 Score", ascending=False)

models_val_comp_df
```

	Accuracy	Precision	Recall	F1 Score
Random Forest Classifier	0.7514	0.7872	0.8605	0.8222
Gradient Boosting Classifier	0.7494	0.7889	0.8531	0.8198
XGBoost Classifier	0.7471	0.7839	0.8578	0.8192
AdaBoost Classifier	0.7282	0.7417	0.9101	0.8173
Bagging Classifier	0.7318	0.7808	0.8320	0.8056

- Random Forest Classifier achieves the highest validation accuracy (75.14%) and F1 Score (82.22%), indicating strong, balanced predictive performance. This suggests that Random Forest generalizes slightly better than others on unseen data.
- AdaBoost Classifier shows the highest recall (91.01%), suggesting effectiveness in minimizing false negatives but accompanied by relatively lower precision (74.17%), risking more false positives.
- Gradient Boosting and XGBoost deliver competitive performance, closely trailing Random Forest with strong balance across metrics, reinforcing their viability in further assessments.
- Bagging Classifier, despite exceptional training performance, experiences a noticeable decline in validation accuracy and precision, indicating potential overfitting that requires further investigation.

Final Model Selection

Test Data Performance - Best Fit Model

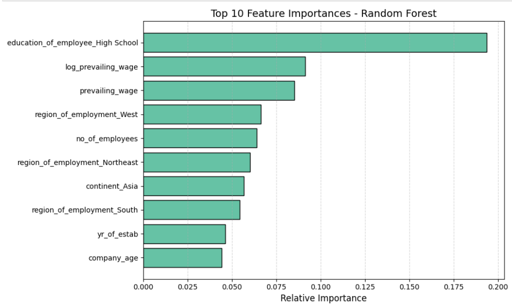
```
# Evaluate Random Forest on test set
rf_tuned_model_test_perf = model_performance_classification_sklearn(
    rf_random.best_estimator_, X_train_over, y_train_over, X_test, y_test, mode="test"
)

# Display performance
rf_tuned_model_test_perf
```

	Accuracy	Precision	Recall	F1 Score
Test Set	0.7282	0.7696	0.8467	0.8063

- Random Forest Classifier achieved an F1 Score of 80.63% on the test set, consistent with its strong validation performance.
- High recall (84.67%) suggests effective identification of eligible visa applicants, reducing the risk of false rejections.
- Balanced precision (76.96%) ensures that most predicted approvals are accurate, supporting confident decision-making.
- Overall results indicate strong generalization, confirming the model's readiness for real-world use.

Feature Importance Plot — Random Forest (Top 10)

```
# Extract feature importances from the final Random Forest model
feature_names = X_train_over.columns
importances = rf_random.best_estimator_.feature_importances_
indices = np.argsort(importances)[-10:]  # Top 10 features

# Plotting
plt.figure(figsize=(10, 6))
plt.title("Top 10 Feature Importances - Random Forest", fontsize=14, weight='normal')
plt.barh(
    range(len(indices)),
    importances[indices],
    color=ElleSet[0],  # Or use ElleSet[:10] for full palette variation
    edgecolor="black"
)
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance", fontsize=12)
plt.grid(axis="x", linestyle="--", alpha=0.5)
plt.tight_layout()
plt.show()
```



- Education level (High School) stands out as the most influential predictor, reinforcing earlier EDA findings that applicants with lower educational attainment experienced higher denial rates. This suggests education may serve as a proxy for perceived skill or job readiness in the certification process.
- Prevailing wage and its log-transformed counterpart both rank among the top features, underscoring the critical role wage plays in visa decisions. As seen in EDA, higher wages were more often associated with certified outcomes—likely reflecting regulatory safeguards intended to protect domestic labor standards.
- Region of employment appears prominently, echoing geographic patterns observed in the EDA. Regions such as the West and South may differ in certification rates due to local labor shortages, industry concentration, or historical approval practices.
- Company size and stability—as reflected in features like number of employees, year of establishment, and company age—emerge as important signals. This supports the insight that more established employers may be viewed as lower risk by adjudicating authorities.
- Continent of origin (Asia) also contributes to predictions. While the model treats this as a neutral input, it's important to contextualize its importance carefully and avoid overinterpreting cultural or geographic correlations without further policy review.

Actionable Insights and Recommendations

Business Objective

EasyVisa aims to streamline the visa certification process by using machine learning to identify applications with a high likelihood of approval. This allows the Office of Foreign Labor Certification (OFLC) to prioritize qualified applicants, reduce administrative backlog, and maintain compliance with labor protection standards.

Key Findings

- Education Level (High School) is the most significant predictor of certification outcomes. Applicants with this education level experienced noticeably different approval rates, reinforcing patterns observed in earlier EDA.
- Prevailing Wage strongly influences case outcomes. Higher wage offers correlated with higher approval rates, reflecting regulatory emphasis on wage fairness for foreign labor.
- Region of Employment emerged as a critical contextual factor, with geographic trends aligning to approval variability—likely due to regional labor demands and historical policy patterns.
- Company Characteristics such as number of employees, company age, and year of establishment signal employer stability, which appears to favor certification outcomes.
- Continent of Origin (Asia) showed moderate influence. While informative for prediction, care should be taken in interpreting this factor to ensure fairness and avoid bias.

Actionable Recommendations

1. **Pre-Screening Tool for Case Review**

Deploy the final model as a decision-support tool within the OFLC pipeline:

- Flag high-likelihood cases for fast-tracking, improving throughput for straightforward approvals.
- Flag low-likelihood cases for additional documentation requests, helping reduce certification risk.

2. **Wage Benchmarking Alerts**

Use wage-related features to guide policy interventions:

- Alert employers when offered wages fall below regional or occupational norms.
- Recommend adjustments to increase likelihood of certification for borderline cases.

3. **Employer Profile Scoring**

Introduce an internal employer reliability index:

- Combine model-driven insights (e.g., company size, age) with historical approval trends.
- Use this score to inform future audit prioritization or review thresholds.

4. **Contextual Policy Feedback**

Provide aggregate insights to policymakers:

- Identify regional or education-related patterns that may inform broader immigration and labor policy.
- Use model findings to support data-driven refinements to eligibility criteria.

Additional Recommendations

- Monitor Model Performance Quarterly: Retrain on updated application data to reflect policy shifts, market conditions, or seasonal hiring patterns.
- Support Transparency with Explainability: Integrate feature importance visualizations into internal tools to help case officers interpret model decisions.
- Pilot Before Full Implementation: Test the model's recommendations within a controlled department before scaling across OFLC's certification system.
