In [1]:
```
#import required libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


import datetime
import pytz


import datetime as dt
from datetime import timezone

from dateutil.relativedelta import *

import os

from sklearn.model_selection import train_test_split
import statsmodels.api as sm
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

%matplotlib inline

import plotly.graph_objs as go
import plotly.express as px
```

In [2]:
```
#import the dataset into jupyter notebook
visa=pd.read_csv('C:/Users/WANJIKU/Downloads/EasyVisa.csv')
visa.head()
```

Out[2]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees	yr_of_estab	region_of_employmen
0	EZYV01	Asia	High School	N	N	14513	2007	Wes
1	EZYV02	Asia	Master's	Y	N	2412	2002	Northeas
2	EZYV03	Asia	Bachelor's	N	Y	44444	2008	Wes
3	EZYV04	Asia	Bachelor's	N	N	98	1897	Wes
4	EZYV05	Africa	Master's	Y	N	1082	2005	Sout

In [3]:
```
vs = visa.copy()
```

In [4]:
```
vs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25480 entries, 0 to 25479
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   case_id                25480 non-null  object 
 1   continent              25480 non-null  object 
 2   education_of_employee  25480 non-null  object 
 3   has_job_experience     25480 non-null  object 
 4   requires_job_training  25480 non-null  object 
 5   no_of_employees        25480 non-null  int64  
 6   yr_of_estab            25480 non-null  int64  
 7   region_of_employment   25480 non-null  object 
 8   prevailing_wage        25480 non-null  float64
 9   unit_of_wage           25480 non-null  object 
 10  full_time_position     25480 non-null  object 
 11  case_status            25480 non-null  object 
dtypes: float64(1), int64(2), object(9)
memory usage: 2.3+ MB
```

In [5]:
```
vs.describe()
```

Out[5]:

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

In [6]:
```
vs.isnull().sum()
```

Out[6]:
```
case_id                  0
continent                0
education_of_employee    0
has_job_experience       0
requires_job_training    0
no_of_employees          0
yr_of_estab              0
region_of_employment     0
prevailing_wage          0
unit_of_wage             0
full_time_position       0
case_status              0
dtype: int64
```

In [7]:
```
vs.shape
```

Out[7]:
```
(25480, 12)
```

In [8]:
```
vs.describe(include='all').T
```

Out[8]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
case_id	25480	25480	EZYV01	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

In [9]:
```
# there is a minimum number of employees which is not right, shows a negative number

df=vs.loc[vs["no_of_employees"]<0.]
df
```

Out[9]:

	case_id	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees	yr_of_estab	region_of_em
245	EZYV246	Europe	Master's	N	N	-25	1980	
378	EZYV379	Asia	Bachelor's	N	Y	-11	2011	
832	EZYV833	South America	Master's	Y	N	-17	2002	
2918	EZYV2919	Asia	Master's	Y	N	-26	2005	
6439	EZYV6440	Asia	Bachelor's	N	N	-14	2013	
6634	EZYV6635	Asia	Bachelor's	Y	N	-26	1923	
7224	EZYV7225	Europe	Doctorate	N	N	-25	1998	
7281	EZYV7282	Asia	High School	N	N	-14	2000	
7318	EZYV7319	Asia	Bachelor's	Y	Y	-26	2006	
7761	EZYV7762	Asia	Master's	N	N	-11	2009	
9872	EZYV9873	Europe	Master's	Y	N	-26	1996	
11493	EZYV11494	Asia	High School	Y	N	-14	1999	
13471	EZYV13472	North America	Master's	N	N	-17	2003	
14022	EZYV14023	Asia	Bachelor's	N	Y	-11	1946	
14146	EZYV14147	Asia	Bachelor's	N	Y	-26	1954	
14726	EZYV14727	Asia	Master's	N	N	-11	2000	
15600	EZYV15601	Asia	Bachelor's	N	N	-14	2014	
15859	EZYV15860	Asia	High School	N	N	-11	1969	
16157	EZYV16158	Asia	Master's	Y	N	-11	1994	
16883	EZYV16884	North America	Bachelor's	Y	N	-26	1968	
17006	EZYV17007	Asia	Doctorate	Y	N	-11	1984	
17655	EZYV17656	North America	Bachelor's	Y	N	-17	2007	
17844	EZYV17845	Asia	Bachelor's	N	N	-14	2012	
17983	EZYV17984	Asia	Bachelor's	N	N	-26	2004	
20815	EZYV20816	Asia	Bachelor's	N	Y	-17	1990	
20984	EZYV20985	Europe	Doctorate	Y	N	-14	1989	
21255	EZYV21256	North America	High School	N	N	-25	1987	
21760	EZYV21761	Asia	Bachelor's	Y	N	-25	2000	
21944	EZYV21945	Africa	Master's	Y	N	-25	1977	
22084	EZYV22085	North America	Bachelor's	Y	N	-14	1980	
22388	EZYV22389	Asia	Master's	Y	N	-14	1986	
23186	EZYV23187	Asia	Master's	N	Y	-11	2007	
23476	EZYV23477	Europe	Master's	Y	N	-11	2000	

In [10]:
```
vs["no_of_employees"] = np.abs(vs["no_of_employees"]) #to sort the negative number
```

In [11]:
```
vs.describe().T
```

Out[11]:

	count	mean	std	min	25%	50%	75%	max
no_of_employees	25480.0	5667.089207	22877.917453	11.0000	1022.00	2109.00	3504.0000	602069.00
yr_of_estab	25480.0	1979.409929	42.366929	1800.0000	1976.00	1997.00	2005.0000	2016.00
prevailing_wage	25480.0	74455.814592	52815.942327	2.1367	34015.48	70308.21	107735.5125	319210.27

EXPLORATORY DATA ANALYSIS

In [12]:
```
# drop the case_id column

vs.drop(['case_id'], axis=1, inplace=True)
```

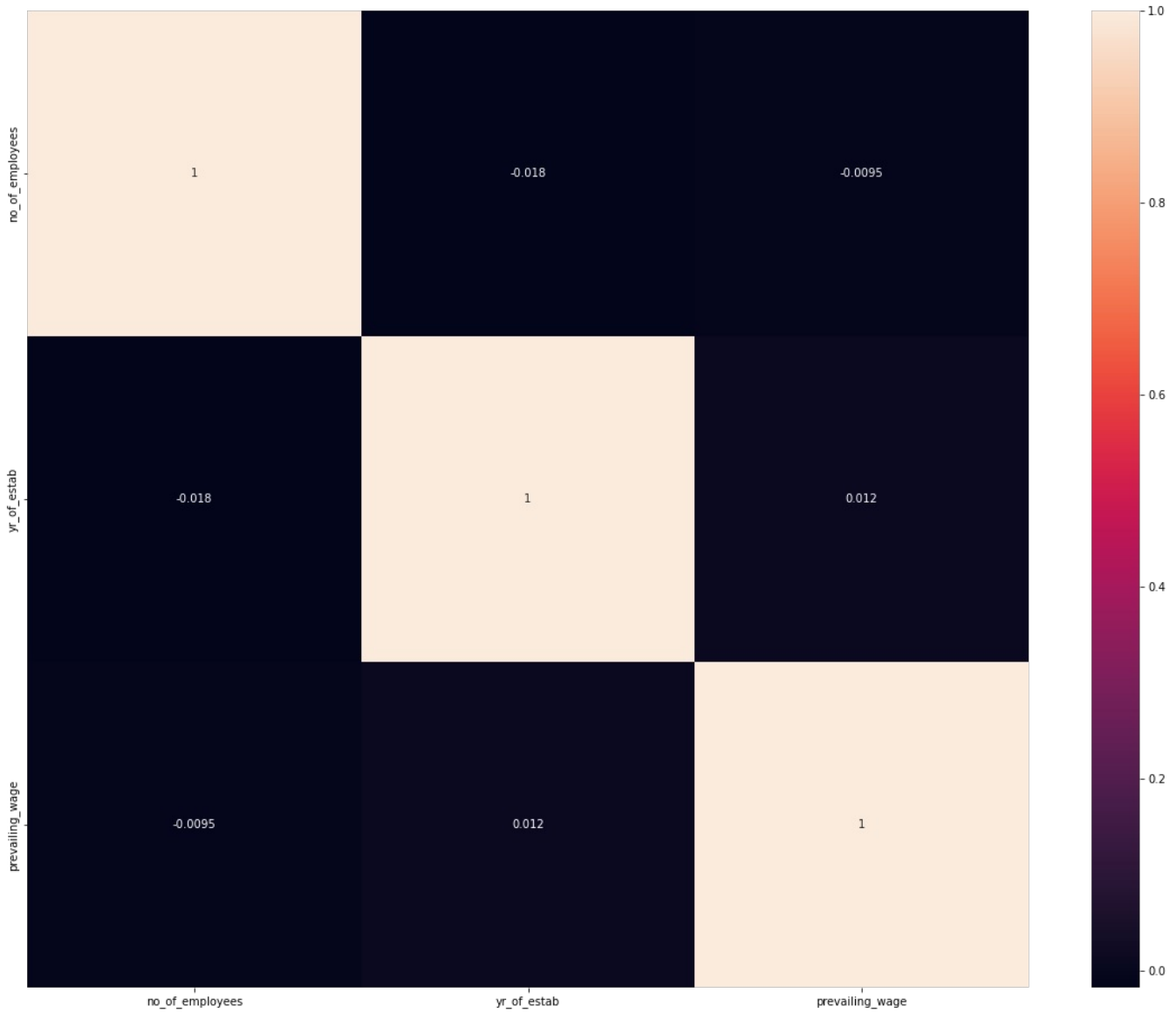
In [13]:
```
vs.head()
```

Out[13]:

	continent	education_of_employee	has_job_experience	requires_job_training	no_of_employees	yr_of_estab	region_of_employment	prevail
0	Asia	High School	N	N	14513	2007	West	
1	Asia	Master's	Y	N	2412	2002	Northeast	8
2	Asia	Bachelor's	N	Y	44444	2008	West	12
3	Asia	Bachelor's	N	N	98	1897	West	8
4	Africa	Master's	Y	N	1082	2005	South	14

In [31]:
```
plt.figure(figsize=(20,16))
sns.heatmap(visa.corr(), fmt='.2g', annot=True)
```

Out[31]:
```
<AxesSubplot:>
```



Leading Questions

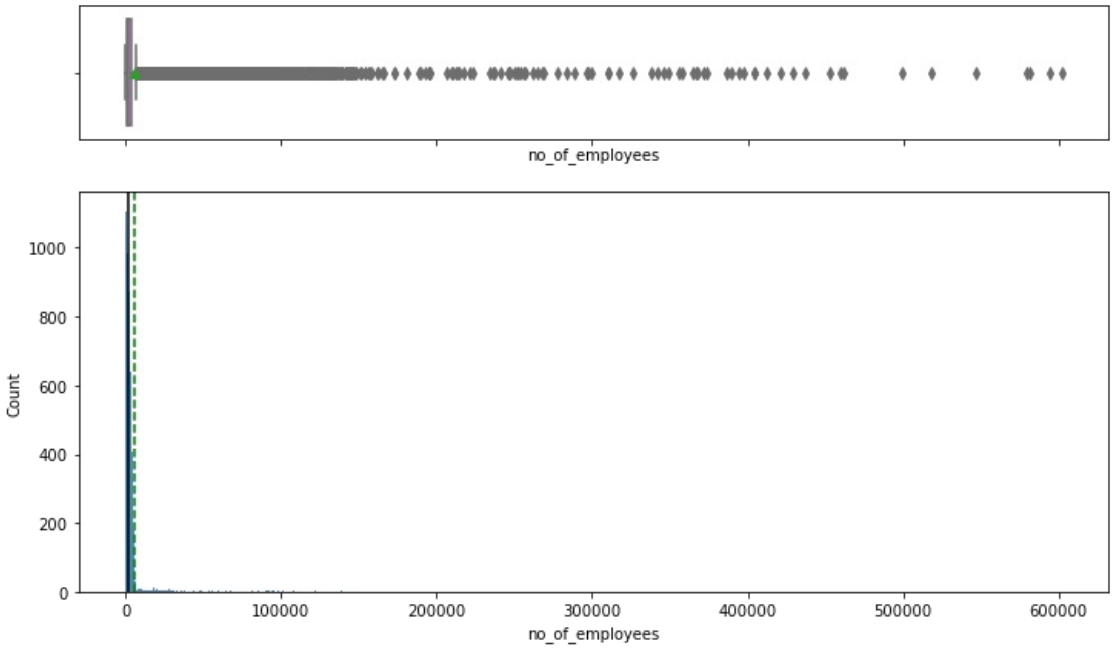
1. Those with higher education may want to travel abroad for a well-paid job. Does education play a role in Visa certification?
2. How does the visa status vary across different continents?
3. Experienced professionals might look abroad for opportunities to improve their lifestyles and career development. Does work experience influence visa status?
4. In the United States, employees are paid at different intervals. Which pay unit is most likely to be certified for a visa?
5. The US government has established a prevailing wage to protect local talent and foreign workers. How does the visa status change with the prevailing wage?

Univariate Analysis

In [14]:
```
def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
```

In [14]:
```
def histogram_boxplot(data, feature, figsize=(12, 7), kde=False, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to show the density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=vs, x=feature, ax=ax_box2, showmeans=True, color="violet"
    )  # boxplot will be created and a star will indicate the mean value of the column
    sns.histplot(
        data=vs, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=vs, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="black", linestyle="-"
    )  # Add median to the histogram
```

In [15]:
```
histogram_boxplot(vs, "no_of_employees")
```

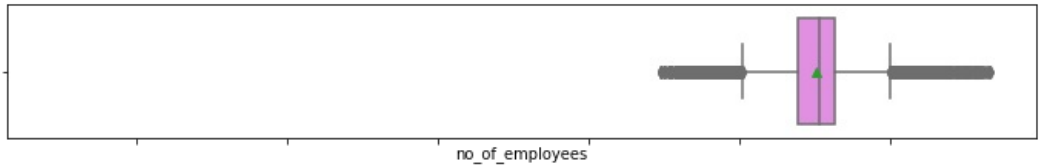


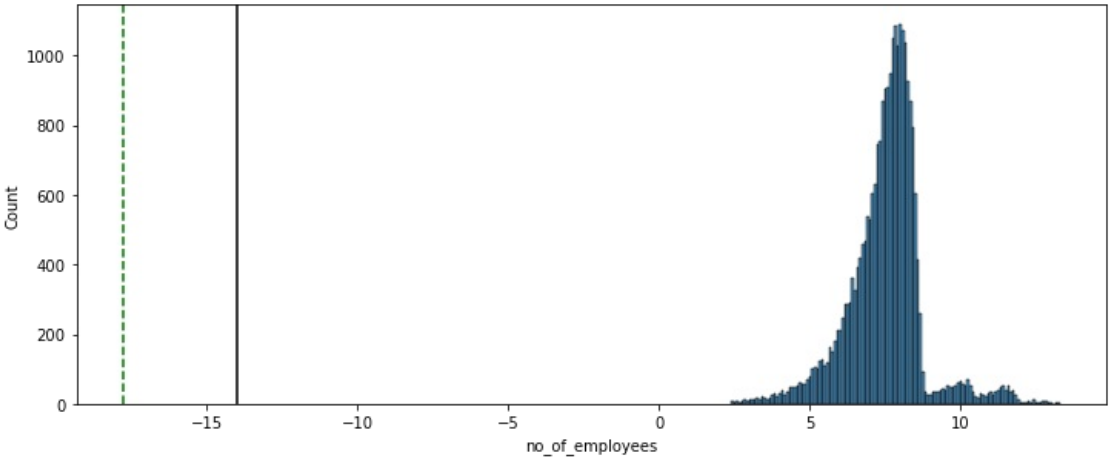
- The data on the number of employees is rightly skewed with a lot of outliers.
- Let's apply the log transform to see if we can make the distribution closer to normal.

No. of Employees

In [16]:
```
vs["no_of_employees"] = np.log(vs["no_of_employees"])
```

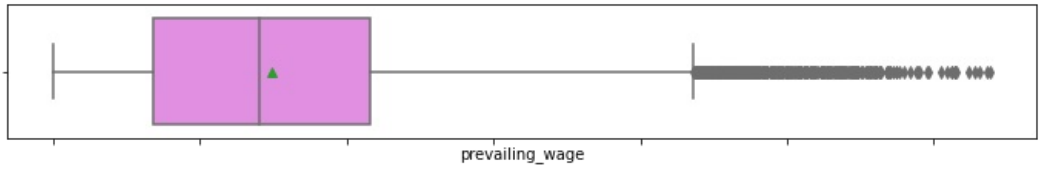
In [17]:
```
histogram_boxplot(df, "no_of_employees")

#the distribution is close to normal but we still have many upper outliers
```

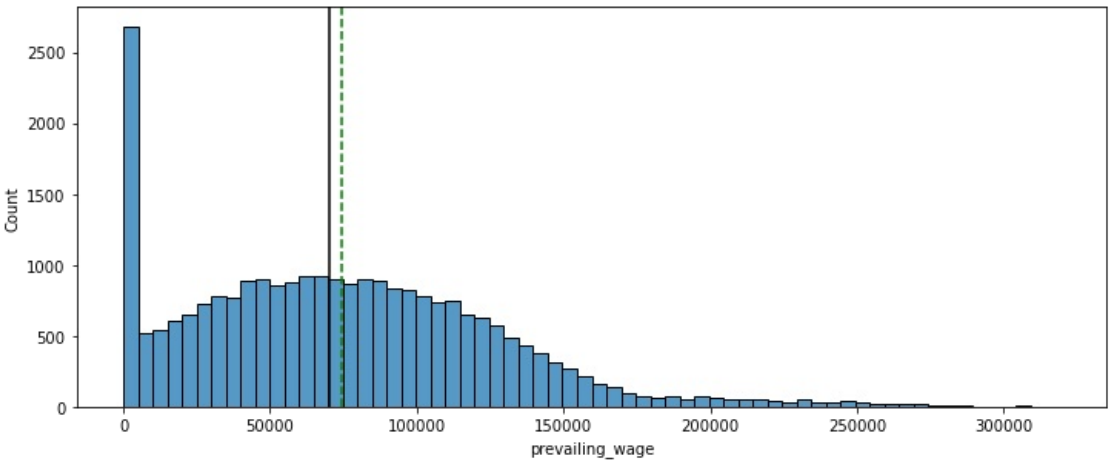




Prevailing Wage

In [18]:
```
histogram_boxplot(vs, "prevailing_wage")

#The data on prevailing wage is normally distributed.
#From the boxplot,there are outliers to the right.
```

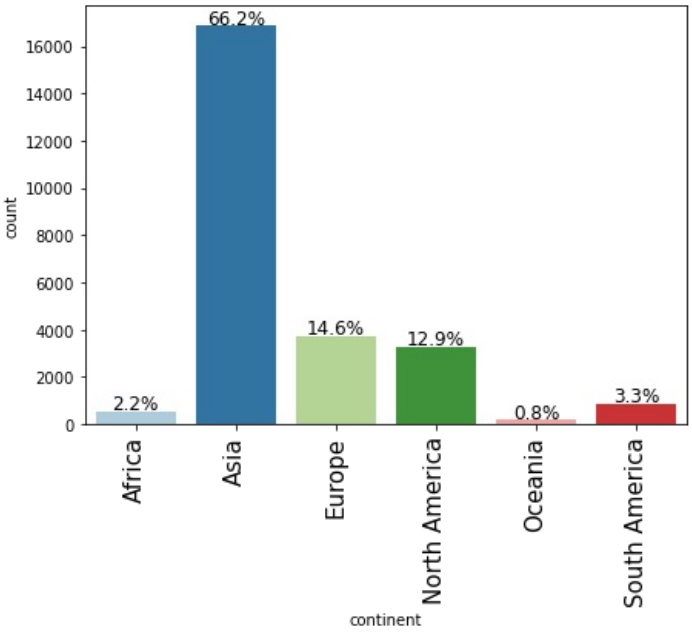




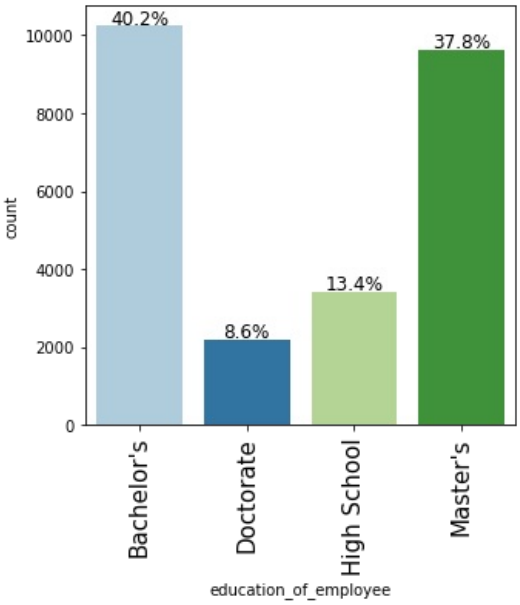
In [19]:
```
# For the categorical features we will use bar plots to analyse

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

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

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 1, 5))

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

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

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

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

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

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

In [20]:
```
labeled_barplot(vs, "continent", perc=True)

#Asia has the largest number of visa applications to the United States at 66.2%
```

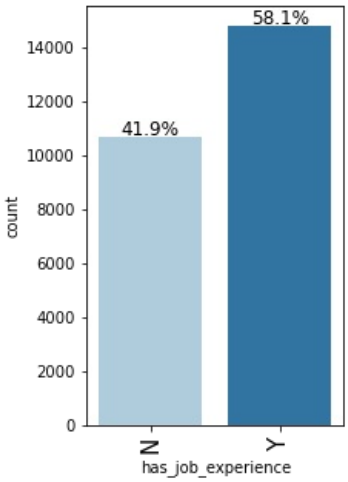


In [21]:
```
labeled_barplot(vs, "education_of_employee", perc=True)

#Bachelor's degree had the most visa applications
```

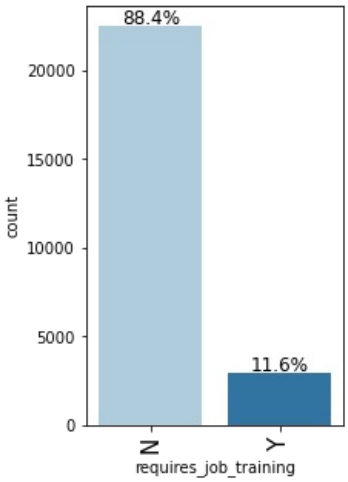


In [22]:
```
labeled_barplot(vs, "has_job_experience", perc=True)
```

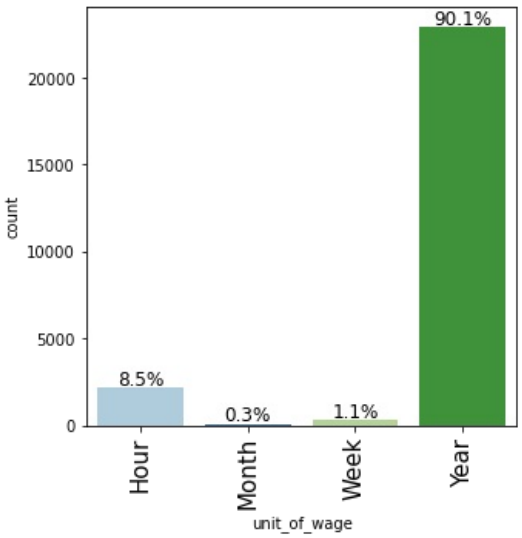
```
#most visa applicants had job experience that is 58.1%
```



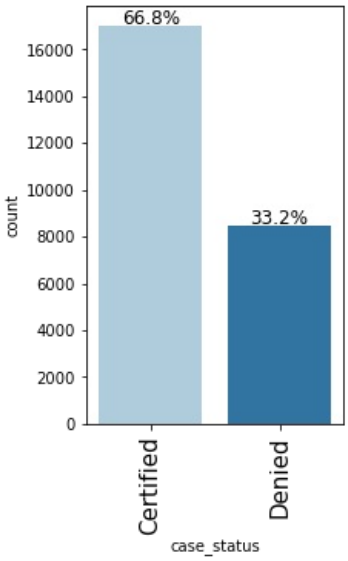
In [23]:
```
labeled_barplot(vs, "requires_job_training", perc=True)

#most of the visa applicants do not require job training
```



In [24]:
```
labeled_barplot(vs, "unit_of_wage", perc=True)

#90% of the applicants have a yearly unit of wage
```



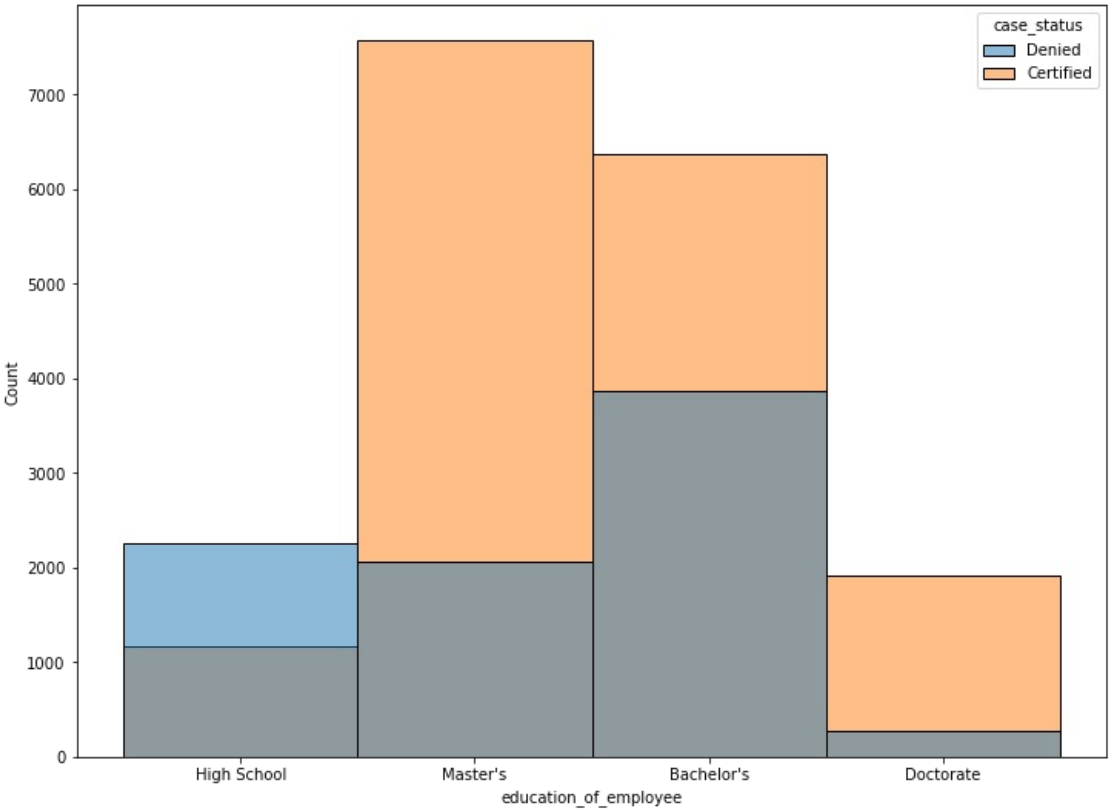
In [26]:
```
labeled_barplot(vs, "case_status", perc=True)

#66.8% of the applicants had their visas certified.
```



Bi-Variate Analysis

Education Versus Case Status

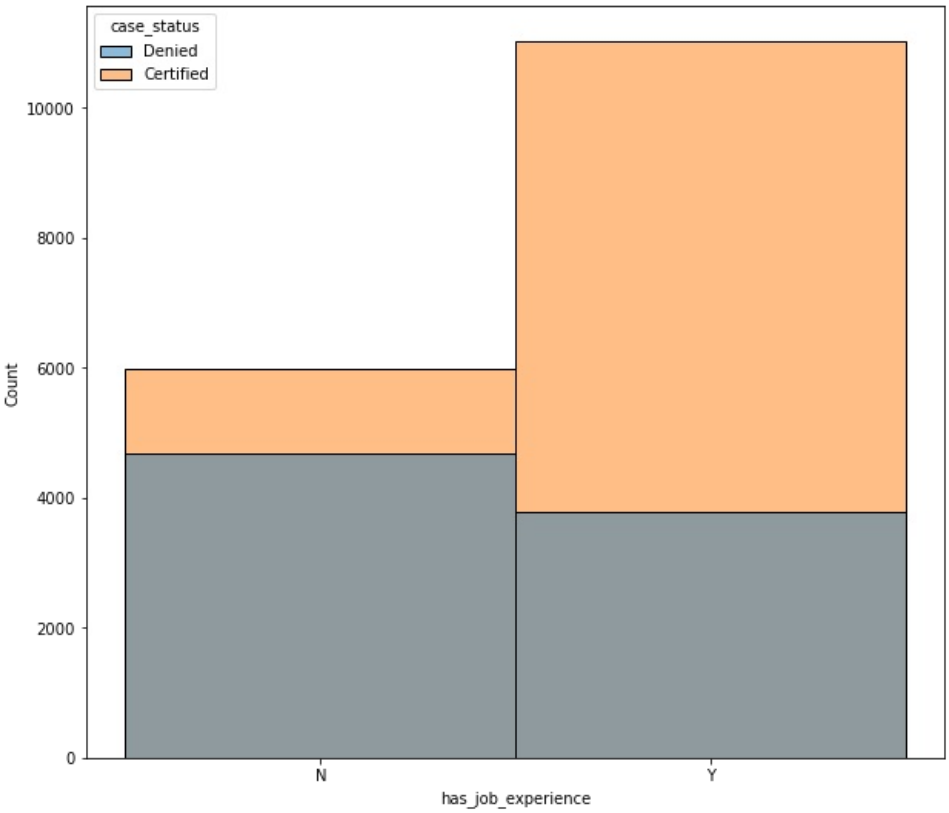
In [27]:
```
plt.figure(figsize=(12, 9))
sns.histplot(data=vs, x="education_of_employee", hue="case_status")
plt.show()
```



- Education plays a role in visa certification and the higher the education the higher the chance of getting certified
- No high school level applicant got their visa certified

Work Experience Versus Case Status

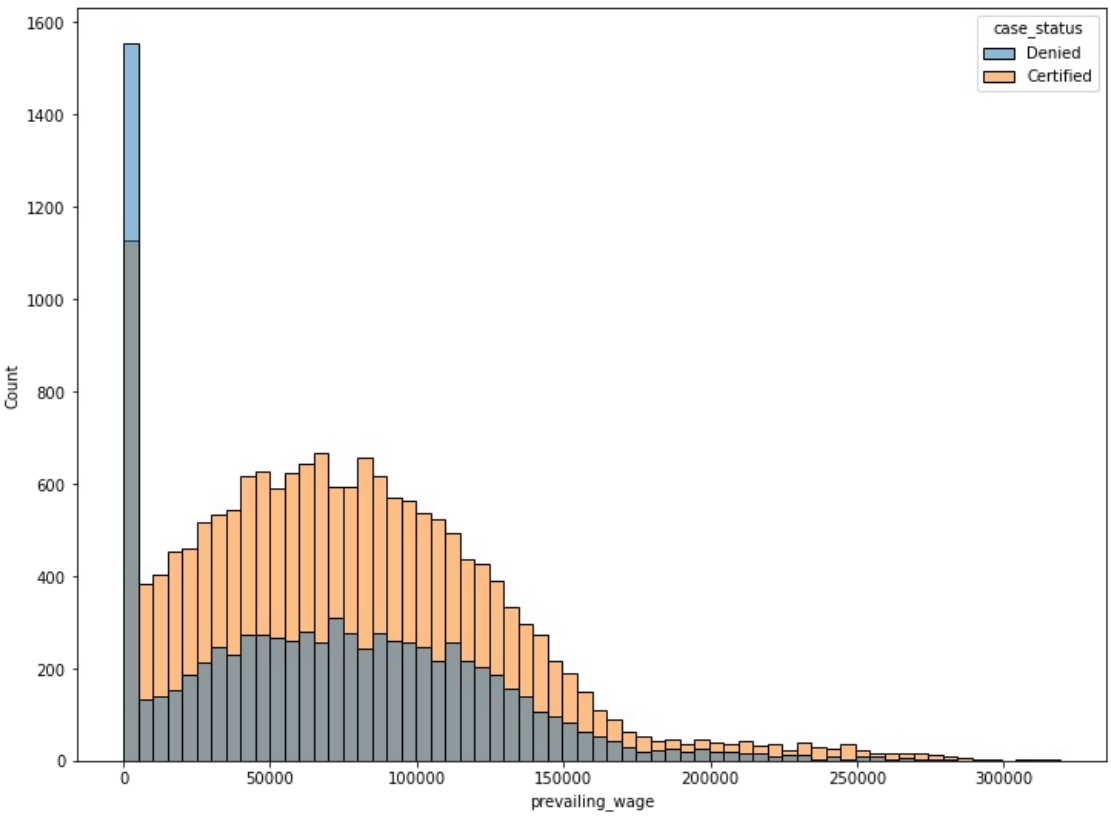
In [28]:
```
plt.figure(figsize=(10, 9))
sns.histplot(data=vs, x="has_job_experience", hue="case_status")
plt.show()
```



- people with job experience had more visa applications certified

Prevailing Wage vs Case Status

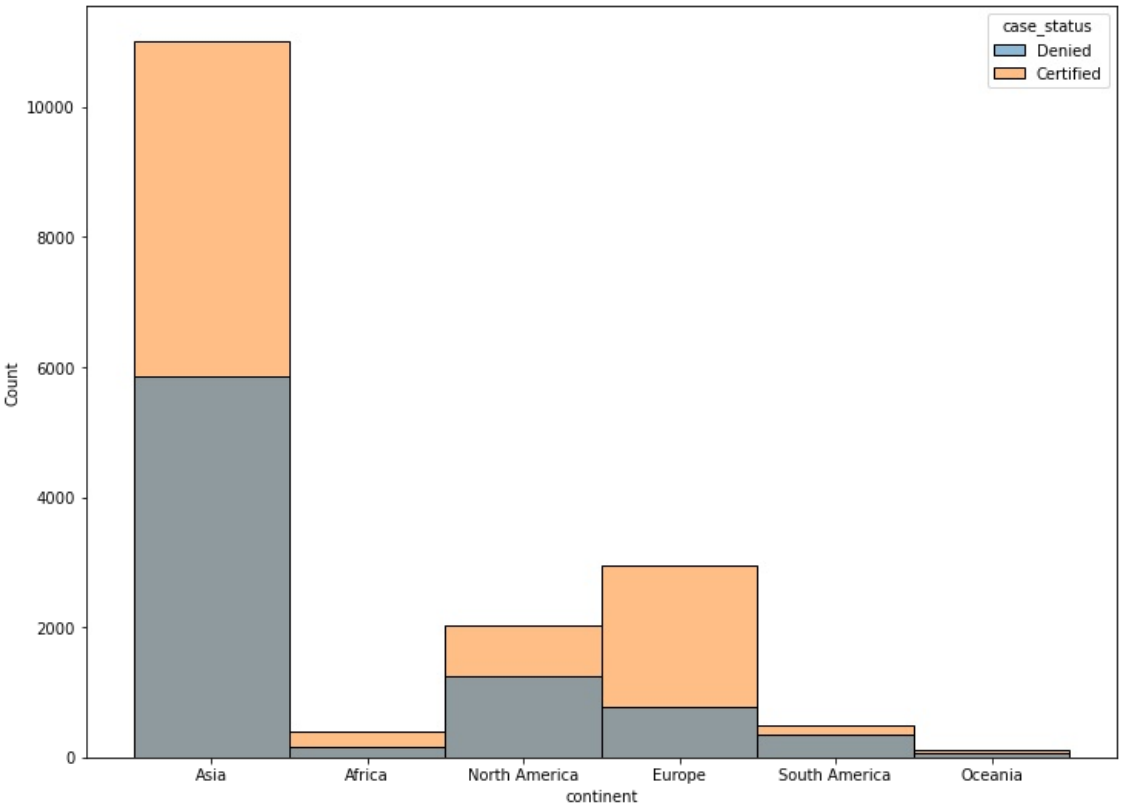
In [29]:
```
plt.figure(figsize=(12, 9))
sns.histplot(data=vs, x="prevailing_wage", hue="case_status")
plt.show()
```



- More than 50% of the visa applications for the prevailing wage of below 150,000 were certified.
- Most of the approval status is between the wage bracket of 40,000 - 120,000

Region Vs Case Status

In [32]:
```
plt.figure(figsize=(12, 9))
sns.histplot(data=vs, x="continent", hue="case_status")
plt.show()
```

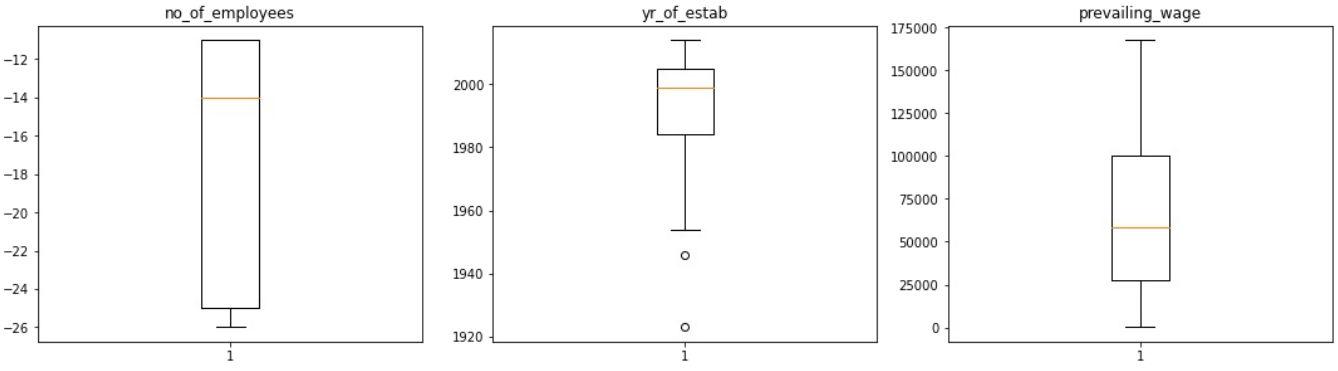


- Asia had the most applications and also the most certified visas, followed by Europe

Data Preprocessing

Detecting and Treating Outliers

In [34]:
```
# outlier detection using boxplot
numeric_columns = vs.select_dtypes(include=np.number).columns.tolist()



plt.figure(figsize=(15, 35))

for i, variable in enumerate(numeric_columns):
    plt.subplot(9, 3, i + 1)
    plt.boxplot(df[variable], whis=1.5)
    plt.tight_layout()
    plt.title(variable)
plt.show()
```



- There are outliers in number of employees and prevailing wage.
- However, we will not treat them as they are proper values.

Data Preparation for Modelling

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

In [35]:
```
#check the number of unique values on object datatypes

vs.select_dtypes(include='object').nunique()
```

Out[35]:
```
continent               6
education_of_employee   4
has_job_experience      2
requires_job_training   2
region_of_employment    5
unit_of_wage            4
full_time_position      2
case_status             2
dtype: int64
```

In [36]:
```
vs[['continent', 'education_of_employee', 'has_job_experience', 'requires_job_training', 'region_of_employment'
```

In [37]:
```
from sklearn import preprocessing
```

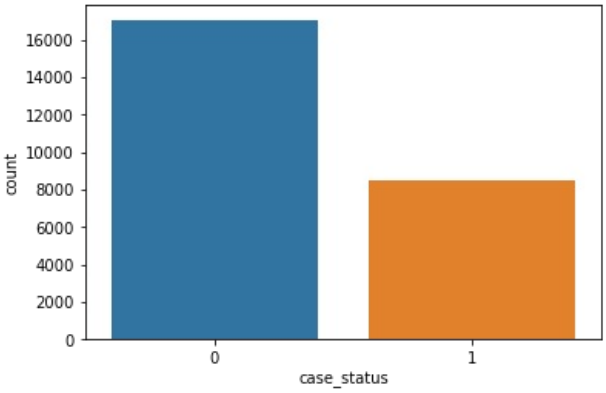
In [38]:
```
for col in vs.select_dtypes(include=['object']).columns:
    label_encoder=preprocessing.LabelEncoder()
    label_encoder.fit(vs[col].unique())
    vs[col]=label_encoder.transform(vs[col])
    print(f"{col}:{vs[col].unique()}")
```

```
continent:[1 0 3 2 5 4]
education_of_employee:[2 3 0 1]
has_job_experience:[0 1]
requires_job_training:[0 1]
region_of_employment:[4 2 3 1 0]
unit_of_wage:[0 3 2 1]
full_time_position:[1 0]
case_status:[1 0]
```

In [39]:
```
sns.countplot(vs['case_status'])

vs['case_status'].value_counts()
```

```
2024-07-18 14:34:04,001 [17224] WARNING  py.warnings:109: [JupyterRequire] C:\Users\WANJIKU\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[39]:
```
0    17018
1     8462
Name: case_status, dtype: int64
```



- Our label is not balanced

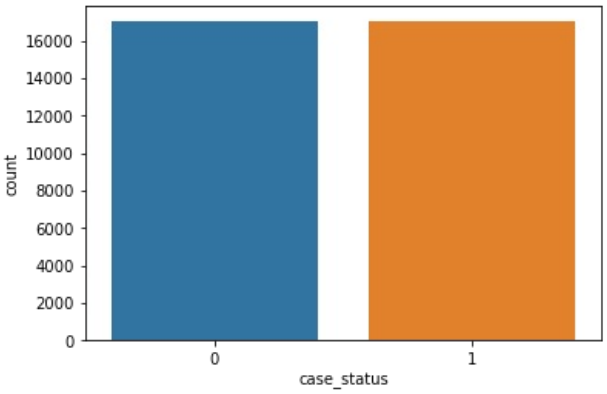
In [41]:
```
#we oversample the minority class to balance the label

from sklearn.utils import resample

vs_majority=vs[(vs['case_status']==0)]
vs_minority=vs[(vs['case_status']==1)]

vs_minority_upsampled=resample(vs_minority,
                                 replace=True,
                                 n_samples=17018,
                                 random_state=0)

vs_upsampled=pd.concat([vs_minority_upsampled, vs_majority])
```

In [42]:
```
sns.countplot(vs_upsampled['case_status'])
```

```
vs_upsampled['case_status'].value_counts()
```

```
2024-07-18 14:39:29,791 [17224] WARNING  py.warnings:109: [JupyterRequire] C:\Users\WANJIKU\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

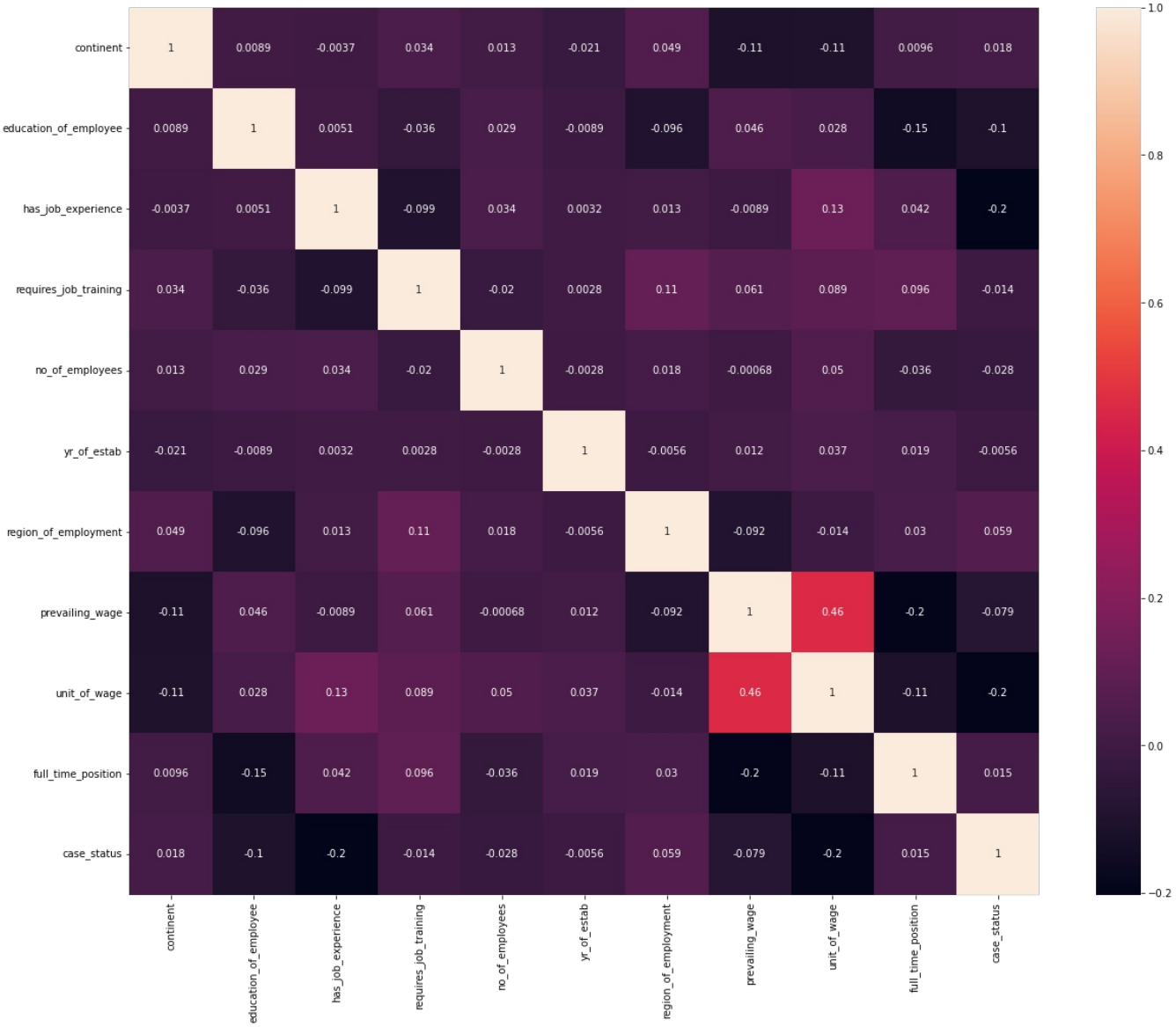
Out[42]:
```
1    17018
0    17018
Name: case_status, dtype: int64
```



- our label is now balanced

In [43]:
```
plt.figure(figsize=(20,16))
sns.heatmap(vs_upsampled.corr(), fmt='.2g', annot=True)
```

Out[43]:
```
<AxesSubplot:>
```



In [44]:
```
X = vs_upsampled.drop('case_status', axis=1)
y = vs_upsampled['case_status']
```

Decision Tree

In [45]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

In [46]:
```
X_train, X_test, y_train, y_test = train_test_split (X,y, test_size=0.2, random_state=0)
```

- Perform Param grid to get the best hyper parameters for the model

In [47]:
```
dtree = DecisionTreeClassifier()

param_grid = { 'max_depth': [ 2,4,6,8],
               'min_samples_split': [2,4,6,8],
               'min_samples_leaf': [1,2,3,4],
               'max_features': ['auto', 'sqrt', 'log2'],
               'random_state': [0,7,42]}

grid_search = GridSearchCV (dtree, param_grid, cv=5)

grid_search.fit(X_train, y_train)

print (grid_search.best_params_)
```

```
{'max_depth': 8, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 8, 'random_state': 42}
```

In [48]:
```
dtree = DecisionTreeClassifier (max_depth=8, min_samples_leaf=2, min_samples_split=8, max_features='auto', rand

dtree.fit(X_train, y_train)
```

Out[48]:
```
DecisionTreeClassifier(max_depth=8, max_features='auto', min_samples_leaf=2,
                       min_samples_split=8, random_state=42)
```

In [49]:
```
y_pred = dtree.predict(X_test)

print("Accuracy Score:", round(accuracy_score(y_test, y_pred)*100,2), "%")
```

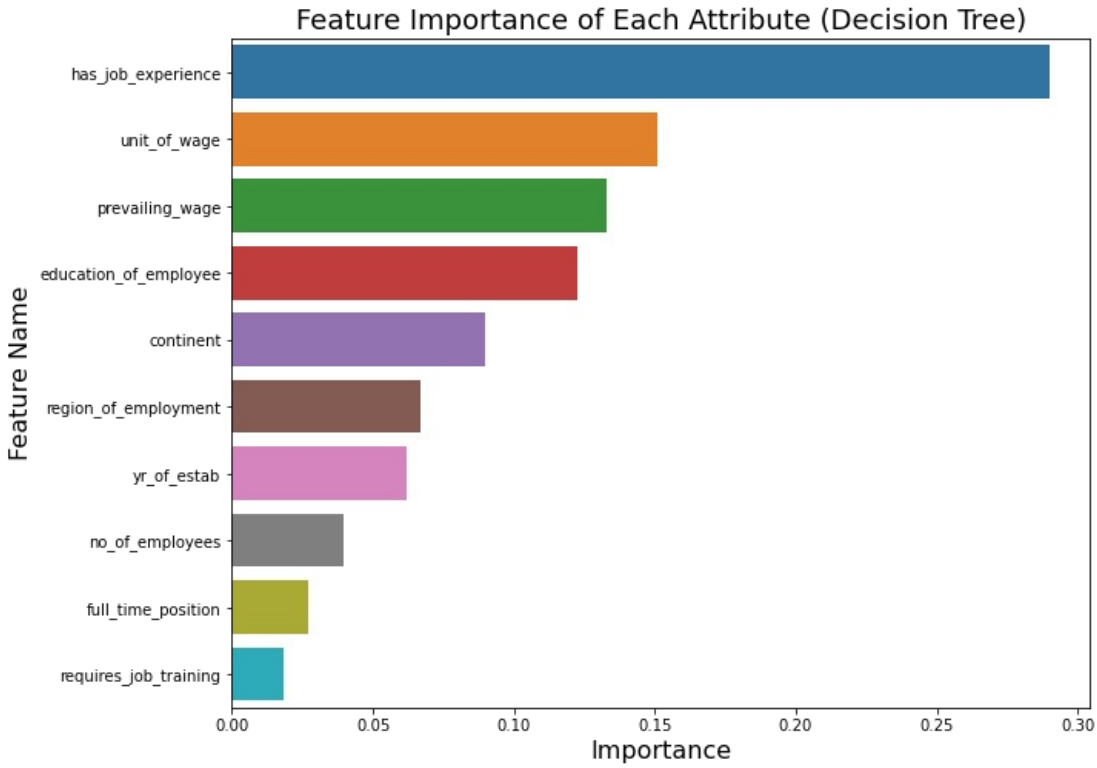
```
Accuracy Score: 63.91 %
```

In [50]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
```

In [51]:
```
print("F-1 Score:", (f1_score(y_test, y_pred, average='micro')))
print("Precision Score:", (precision_score(y_test, y_pred, average='micro')))
print("Recall Score:", (recall_score(y_test, y_pred, average='micro')))
print("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

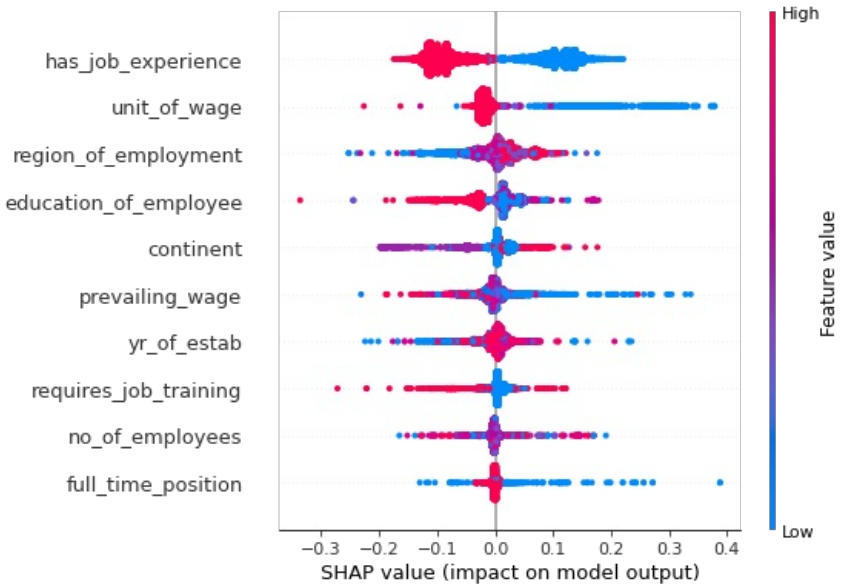
```
F-1 Score: 0.6391010575793185
Precision Score: 0.6391010575793185
Recall Score: 0.6391010575793185
Jaccard Score: 0.4696168375607124
Log Loss: 12.465149987542022
```

In [52]:
```
imp_df = pd.DataFrame({ "Feature Name": X_train.columns,
                        "Importance": dtree.feature_importances_})

fi= imp_df.sort_values(by = "Importance", ascending=False)

plt.figure(figsize=(10,8))
sns.barplot(data=fi, x= 'Importance', y= 'Feature Name')
plt.title('Feature Importance of Each Attribute (Decision Tree)', fontsize=18)
plt.xlabel( 'Importance', fontsize=16)
plt.ylabel( 'Feature Name', fontsize=16)
plt.show()
```

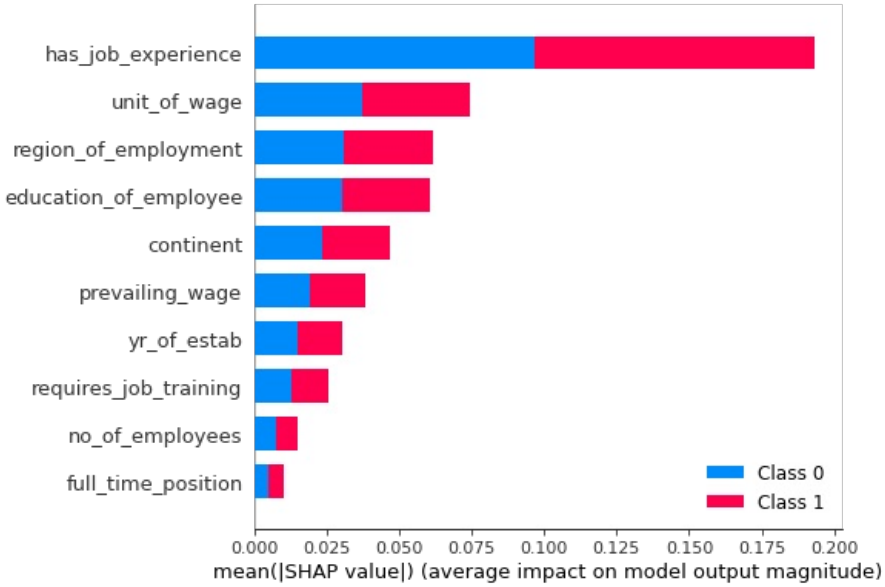


In [53]:
```
import shap
```

In [54]:
```
explainer=shap.TreeExplainer(dtree)

shap_values=explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names=X_test.columns)
```

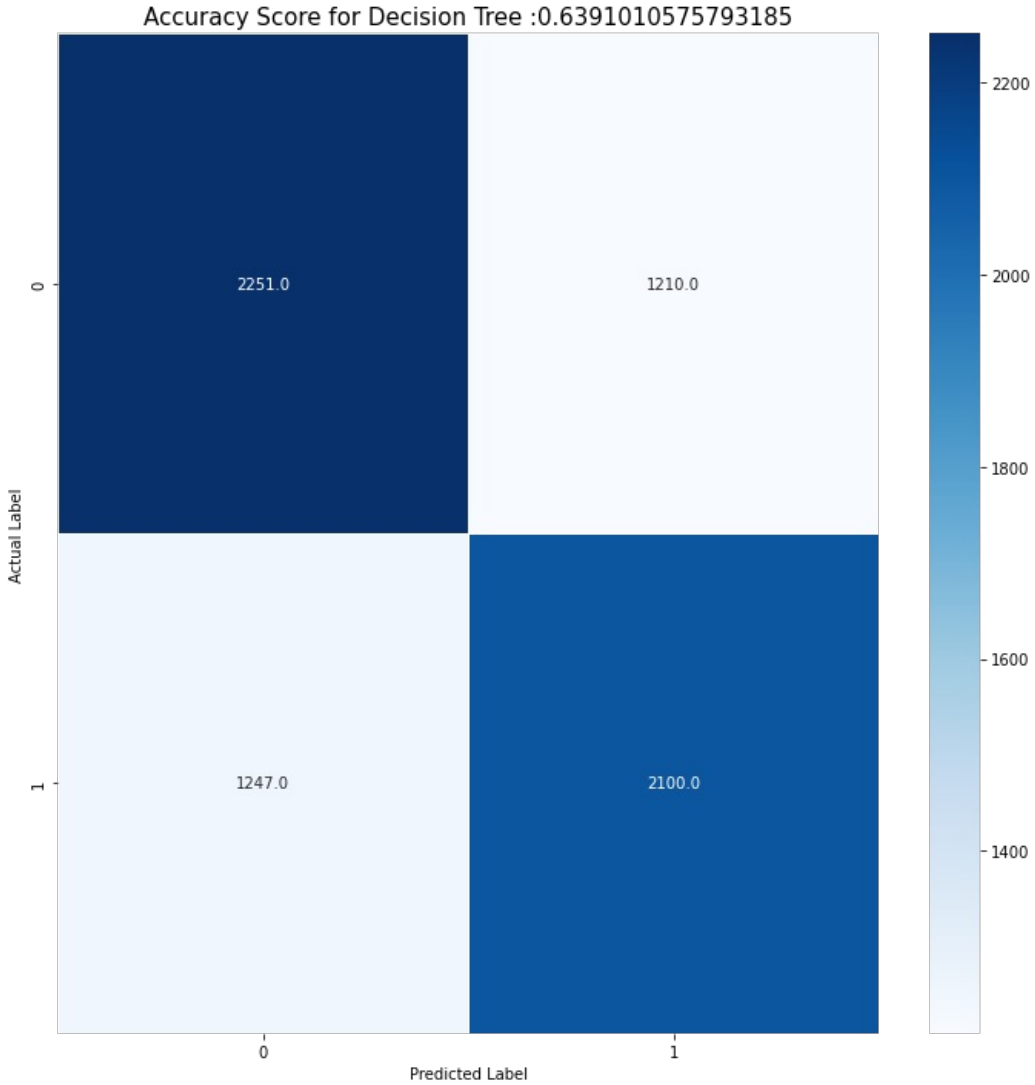


In [55]:
```
shap.summary_plot(shap_values, X_test)
```



In [56]:
```
from sklearn.metrics import confusion_matrix
```

In [57]:
```
cm = confusion_matrix (y_test, y_pred)

plt.figure(figsize=(10,10))
sns.heatmap ( data=cm,linewidths=.5, fmt='.1f', annot=True, cmap='Blues')




plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
all_sample_title = 'Accuracy Score for Decision Tree :{0}'.format (dtree.score(X_test, y_test))
plt.title(all_sample_title, size=15)
plt.tight_layout()
```



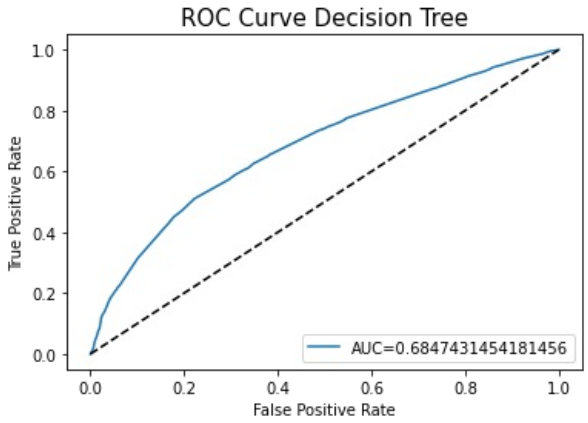
In [58]:
```
from sklearn.metrics import roc_curve, roc_auc_score
```

```
from sklearn import metrics
```

In [59]:
```
y_pred_proba = dtree.predict_proba(X_test)[:][:,1]
vs_actual_predicted=pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual'])])
vs_actual_predicted.index=y_test.index
```

In [60]:
```
fpr,tpr,_= metrics.roc_curve(y_test, y_pred_proba)

auc = metrics.roc_auc_score(y_test, y_pred_proba)

plt.plot(fpr, tpr, label='AUC='+str(auc))
plt.plot(fpr, fpr, linestyle='--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Decision Tree', size=15)
plt.legend(loc=4)
```

Out[60]:
```
<matplotlib.legend.Legend at 0x1e51c15ec70>
```



XGBoost

In [61]:
```
from xgboost import XGBClassifier
```

In [62]:
```
xgbc=XGBClassifier()
```

In [64]:
```
xgbc = XGBClassifier (max_depth=9, min_samples_leaf=10, min_samples_split=2, max_features='sqrt', random_state=

xgbc.fit(X_train, y_train)
```

```
2024-07-18 16:30:59,448 [17224] WARNING  py.warnings:109: [JupyterRequire] [16:30:59] WARNING: C:\buildkite-age
nt\builds\buildkite-windows-cpu-autoscaling-group-i-0cec3277c4d9d0165-1\xgboost\xgboost-ci-windows\src\learner.
cc:742:
Parameters: { "max_features", "min_samples_leaf", "min_samples_split" } are not used.
```

Out[64]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=9, max_features='sqrt',
              max_leaves=None, min_child_weight=None, min_samples_leaf=10,
              min_samples_split=2, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, ...)
```

In [65]:
```
y_pred = xgbc.predict(X_test)

print("Accuracy Score:", round(accuracy_score(y_test, y_pred)*100,2), "%")
```

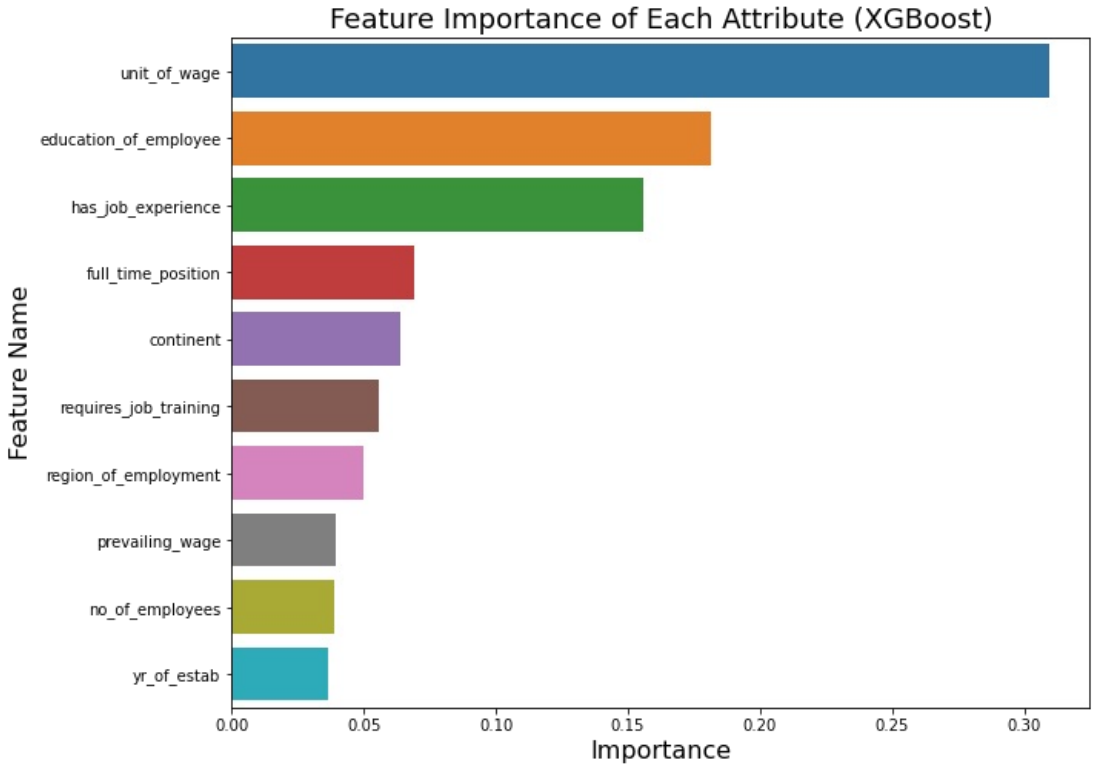
```
Accuracy Score: 81.67 %
```

In [66]:
```
print("F-1 Score:", (f1_score(y_test, y_pred, average='micro')))
print("Precision Score:", (precision_score(y_test, y_pred, average='micro')))
print("Recall Score:", (recall_score(y_test, y_pred, average='micro')))
print("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

```
F-1 Score: 0.8166862514688602
Precision Score: 0.8166862514688602
Recall Score: 0.8166862514688602
Jaccard Score: 0.6901688182720953
Log Loss: 6.331526765274317
```

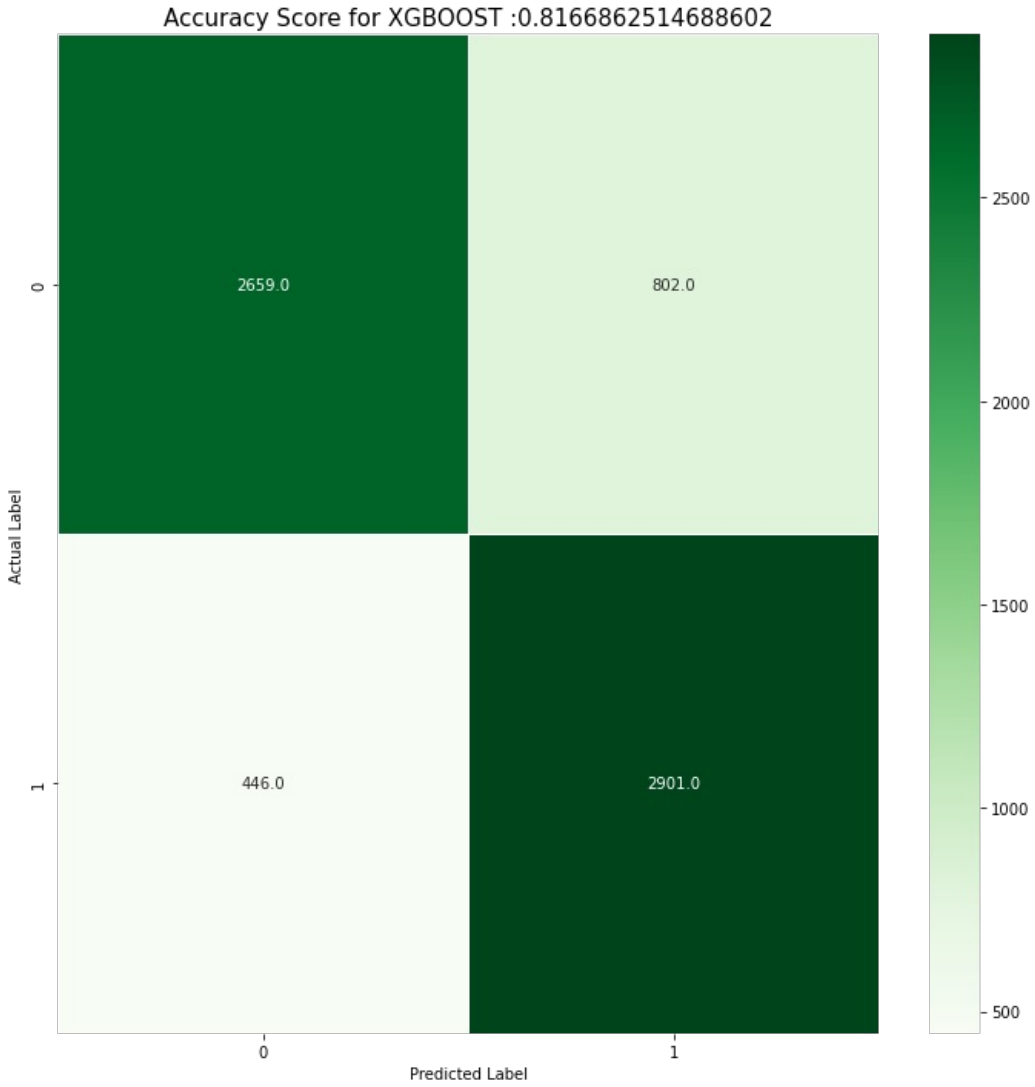
In [67]:
```
imp_df = pd.DataFrame({ "Feature Name": X_train.columns,
                        "Importance": xgbc.feature_importances_})
```

```
fi= imp_df.sort_values(by = "Importance", ascending=False)

plt.figure(figsize=(10,8))
sns.barplot(data=fi, x= 'Importance', y= 'Feature Name')
plt.title('Feature Importance of Each Attribute (XGBoost)', fontsize=18)
plt.xlabel( 'Importance', fontsize=16)
plt.ylabel( 'Feature Name', fontsize=16)
plt.show()
```



In [68]:

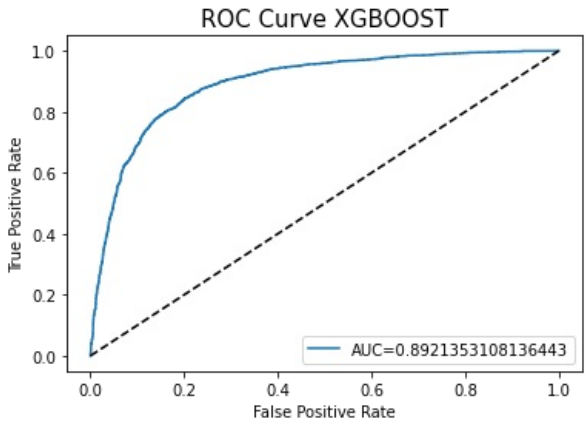
```
cm = confusion_matrix (y_test, y_pred)

plt.figure(figsize=(10,10))
sns.heatmap ( data=cm,linewidths=.5, fmt='.1f', annot=True, cmap='Greens')



plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
all_sample_title = 'Accuracy Score for XGBOOST :{0}'.format (xgbc.score(X_test, y_test))
plt.title(all_sample_title, size=15)
plt.tight_layout()
```



In [69]:
```
y_pred_proba = xgbc.predict_proba(X_test)[:][:,1]
vs_actual_predicted=pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual'])])
vs_actual_predicted.index=y_test.index
```

In [70]:
```
fpr,tpr,_= metrics.roc_curve(y_test, y_pred_proba)

auc = metrics.roc_auc_score(y_test, y_pred_proba)

plt.plot(fpr, tpr, label='AUC='+str(auc))
plt.plot(fpr, fpr, linestyle='--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve XGBOOST', size=15)
plt.legend(loc=4)
```

Out[70]:
```
<matplotlib.legend.Legend at 0x1e523b7be50>
```



Random Forest

In [71]:
```
from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()
```

In [72]:
```
rfc = RandomForestClassifier (max_depth=9, min_samples_leaf=10, min_samples_split=2, max_features='sqrt', rando

rfc.fit(X_train, y_train)
```

Out[72]:
```
RandomForestClassifier(max_depth=9, max_features='sqrt', min_samples_leaf=10,
                       random_state=0)
```

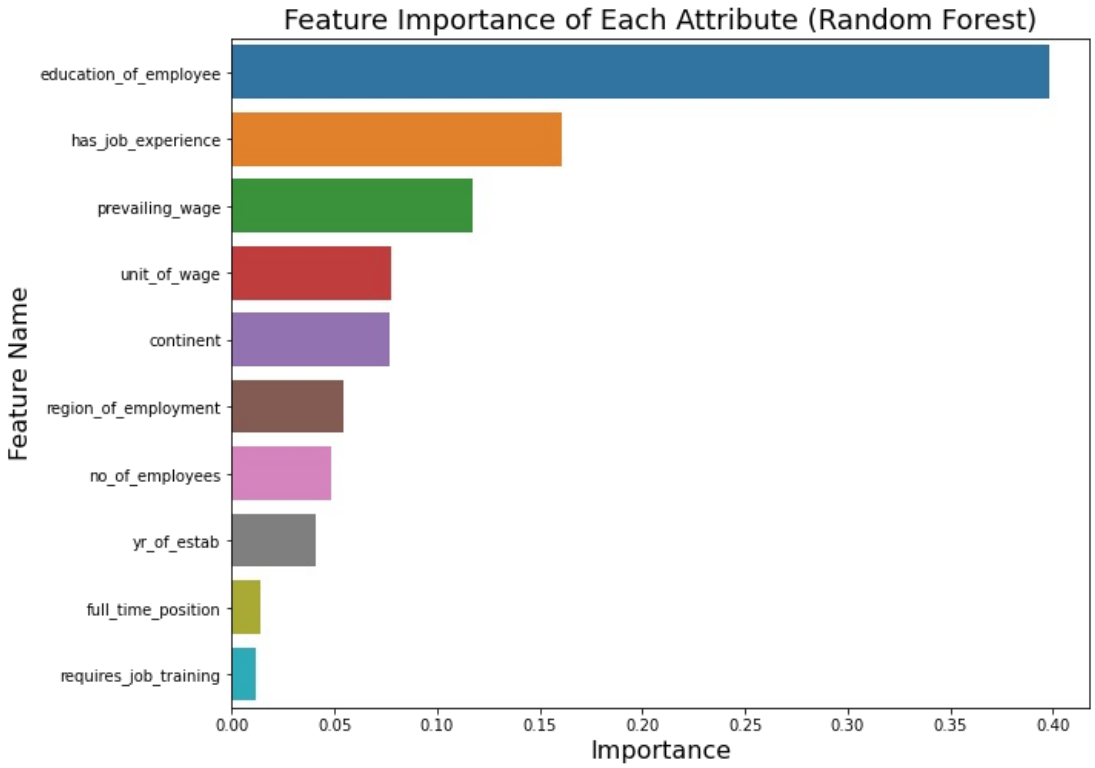
In [73]:
```
y_pred = rfc.predict(X_test)

print("Accuracy Score:", round(accuracy_score(y_test, y_pred)*100,2), "%")
```

```
Accuracy Score: 72.24 %
```

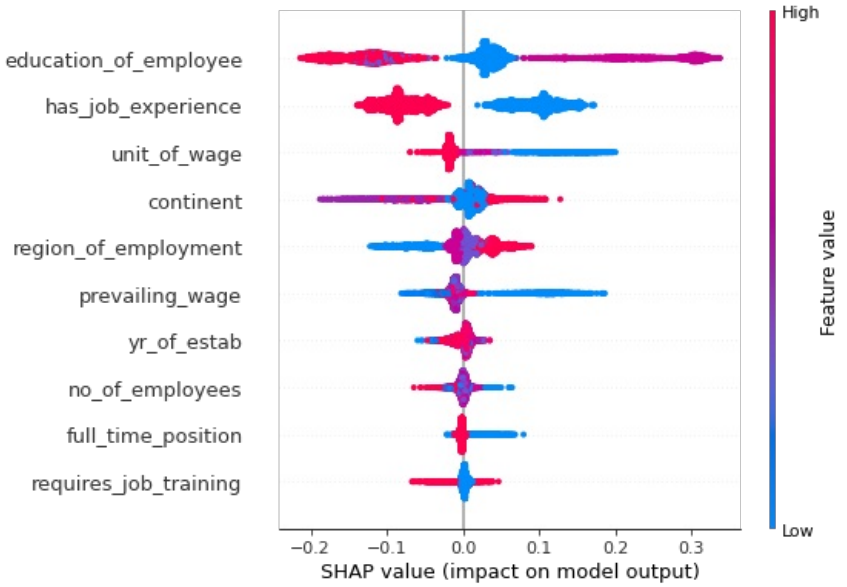
In [74]:
```
print("F-1 Score:", (f1_score(y_test, y_pred, average='micro')))
print("Precision Score:", (precision_score(y_test, y_pred, average='micro')))
print("Recall Score:", (recall_score(y_test, y_pred, average='micro')))
print("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

```
F-1 Score: 0.7223854289071681
Precision Score: 0.7223854289071681
Recall Score: 0.7223854289071681
Jaccard Score: 0.5654173373189239
Log Loss: 9.588571772811658
```

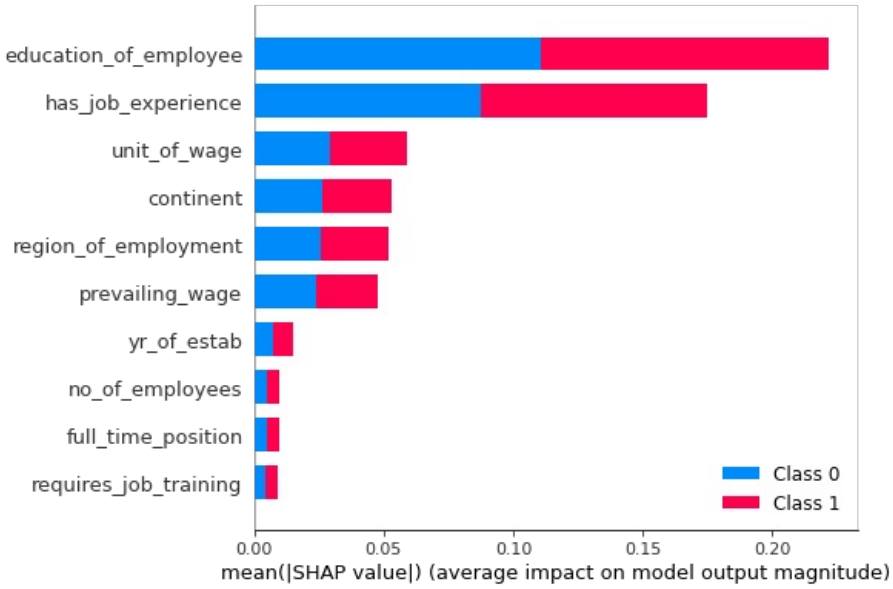
In [75]:
```
imp_df = pd.DataFrame({ "Feature Name": X_train.columns,
                        "Importance": rfc.feature_importances_})

fi= imp_df.sort_values(by = "Importance", ascending=False)

plt.figure(figsize=(10,8))
sns.barplot(data=fi, x= 'Importance', y= 'Feature Name')
plt.title('Feature Importance of Each Attribute (Random Forest)', fontsize=18)
plt.xlabel( 'Importance', fontsize=16)
plt.ylabel( 'Feature Name', fontsize=16)
plt.show()
```



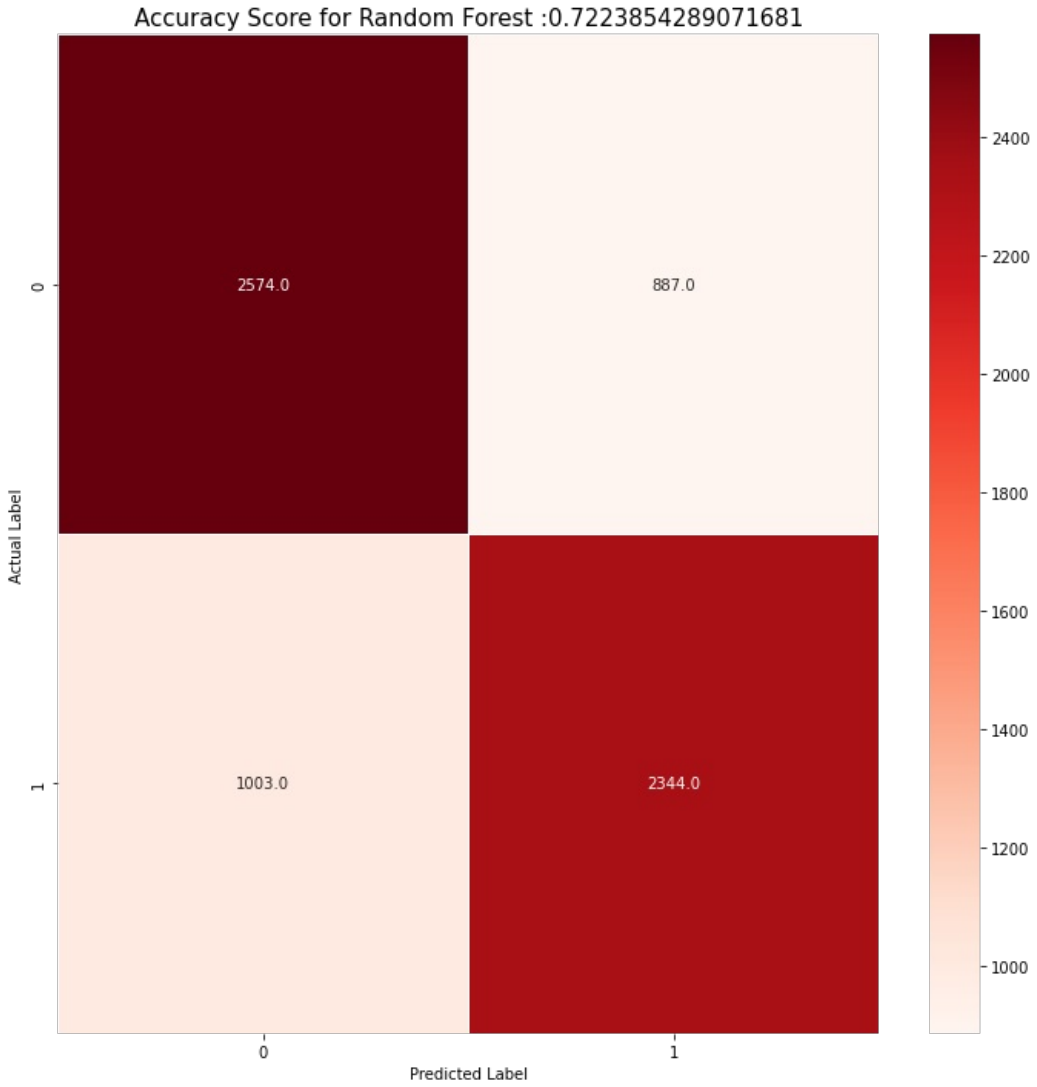
In [76]:
```
explainer=shap.TreeExplainer(rfc)

shap_values=explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names=X_test.columns)
```



In [77]:
```
shap.summary_plot(shap_values, X_test)
```



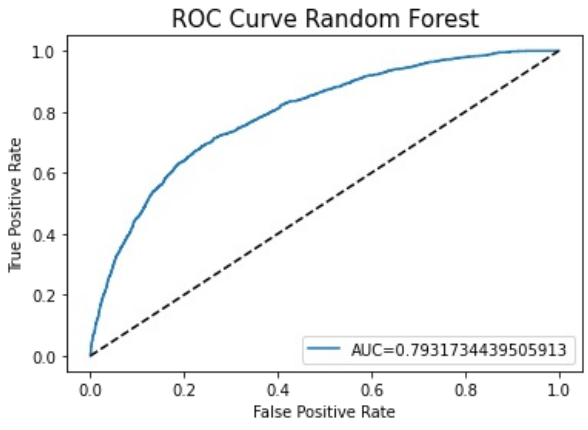
In [78]:
```
cm = confusion_matrix (y_test, y_pred)

plt.figure(figsize=(10,10))
sns.heatmap ( data=cm,linewidths=.5, fmt='.1f', annot=True, cmap='Reds')




plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
all_sample_title = 'Accuracy Score for Random Forest :{0}'.format (rfc.score(X_test, y_test))
plt.title(all_sample_title, size=15)
plt.tight_layout()
```



In [79]:
```
y_pred_proba = rfc.predict_proba(X_test)[:][:,1]
vs_actual_predicted=pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual'])])
vs_actual_predicted.index=y_test.index
```

In [80]:
```
fpr,tpr,_= metrics.roc_curve(y_test, y_pred_proba)

auc = metrics.roc_auc_score(y_test, y_pred_proba)

plt.plot(fpr, tpr, label='AUC='+str(auc))
plt.plot(fpr, fpr, linestyle='--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Random Forest', size=15)
plt.legend(loc=4)
```

Out[80]:
```
<matplotlib.legend.Legend at 0x1e5273a6c40>
```



K Nearest Neighbors

In [81]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [85]:
```
knn = KNeighborsClassifier()
```

```
knn.fit(X_train, y_train)
```

Out[85]:
```
KNeighborsClassifier()
```

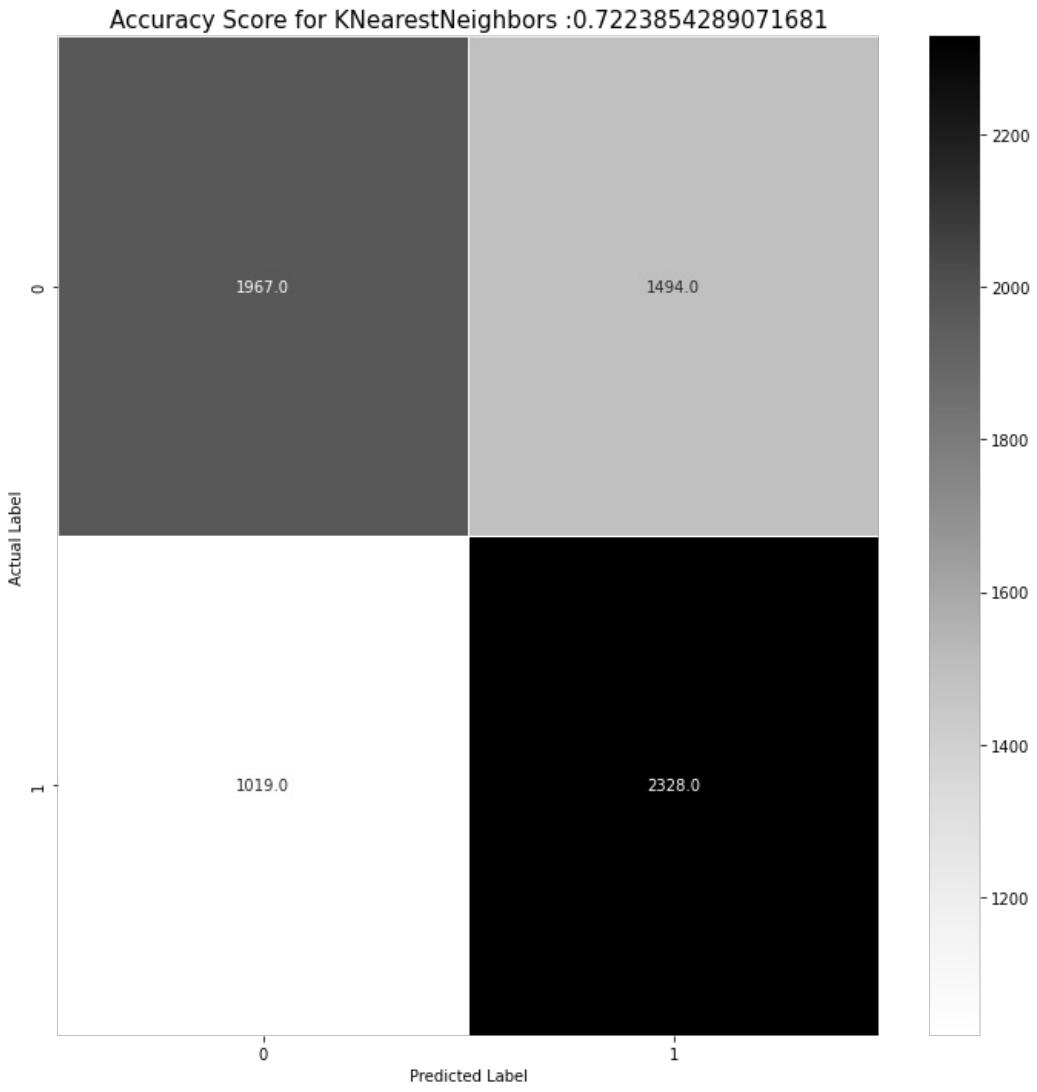
In [86]:
```
y_pred = knn.predict(X_test)

print("Accuracy Score:", round(accuracy_score(y_test, y_pred)*100,2), "%")
```

```
Accuracy Score: 63.09 %
```

In [87]:
```
print("F-1 Score:", (f1_score(y_test, y_pred, average='micro')))
print("Precision Score:", (precision_score(y_test, y_pred, average='micro')))
print("Recall Score:", (recall_score(y_test, y_pred, average='micro')))
print("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

```
F-1 Score: 0.6308754406580493
Precision Score: 0.6308754406580493
Recall Score: 0.6308754406580493
Jaccard Score: 0.46078746915567
Log Loss: 12.749286086805414
```

In [91]:
```
cm = confusion_matrix (y_test, y_pred)

plt.figure(figsize=(10,10))
sns.heatmap ( data=cm,linewidths=.5, fmt='.1f', annot=True, cmap='Greys')




plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
all_sample_title = 'Accuracy Score for KNearestNeighbors :{0}'.format (rfc.score(X_test, y_test))
plt.title(all_sample_title, size=15)
plt.tight_layout()
```

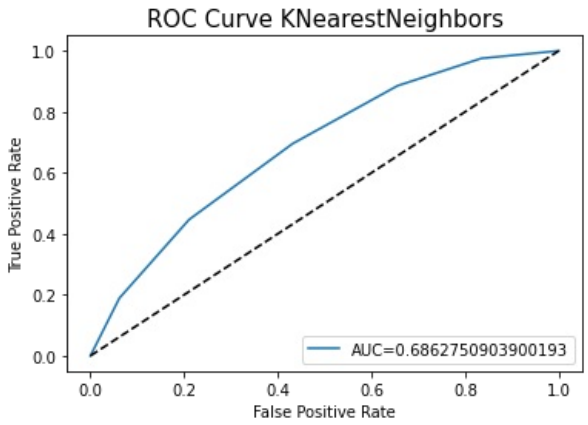


In [92]:
```
y_pred_proba = knn.predict_proba(X_test)[:][:,1]
vs_actual_predicted=pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual'])])
vs_actual_predicted.index=y_test.index
```

In [93]:
```
fpr,tpr,_= metrics.roc_curve(y_test, y_pred_proba)
```

```
auc = metrics.roc_auc_score(y_test, y_pred_proba)

plt.plot(fpr, tpr, label='AUC='+str(auc))
plt.plot(fpr, fpr, linestyle='--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve KNearestNeighbors', size=15)
plt.legend(loc=4)
```

Out[93]:
```
<matplotlib.legend.Legend at 0x1e528a8fe50>
```



Naive Bayes

In [94]:
```
from sklearn.naive_bayes import GaussianNB
```

In [95]:
```
model = GaussianNB()
```

In [96]:
```
model.fit(X_train, y_train)
```

Out[96]:
```
GaussianNB()
```

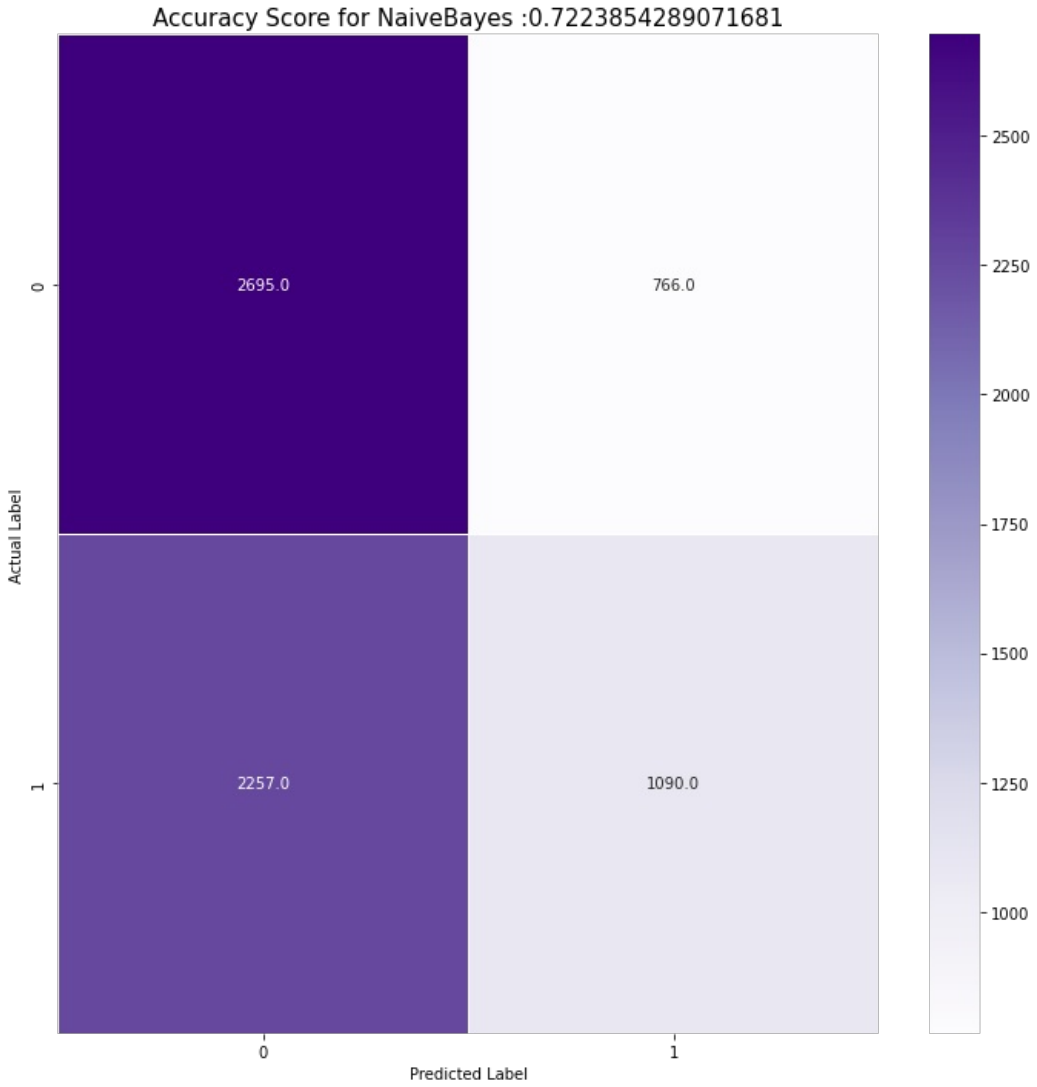
In [97]:
```
y_pred = model.predict(X_test)

print("Accuracy Score:", round(accuracy_score(y_test, y_pred)*100,2), "%")
```

```
Accuracy Score: 55.6 %
```

In [98]:
```
print("F-1 Score:", (f1_score(y_test, y_pred, average='micro')))
print("Precision Score:", (precision_score(y_test, y_pred, average='micro')))
print("Recall Score:", (recall_score(y_test, y_pred, average='micro')))
print("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

```
F-1 Score: 0.5559635722679201
Precision Score: 0.5559635722679201
Recall Score: 0.5559635722679201
Jaccard Score: 0.3850066117383786
Log Loss: 15.33656485508909
```

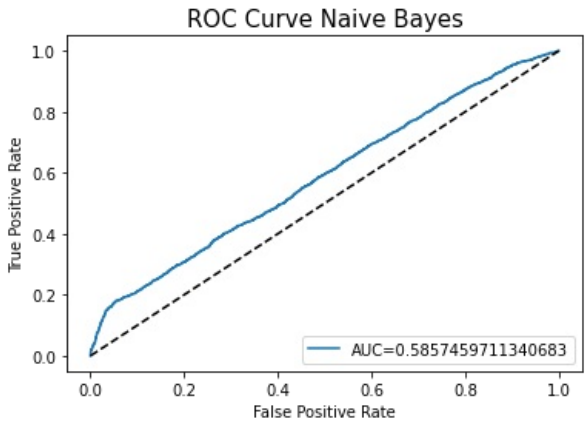
In [99]:
```
cm = confusion_matrix (y_test, y_pred)

plt.figure(figsize=(10,10))
sns.heatmap ( data=cm,linewidths=.5, fmt='.1f', annot=True, cmap='Purples')




plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
all_sample_title = 'Accuracy Score for NaiveBayes :{0}'.format (rfc.score(X_test, y_test))
plt.title(all_sample_title, size=15)
plt.tight_layout()
```



In [100…
```
y_pred_proba = model.predict_proba(X_test)[:][:,1]
vs_actual_predicted=pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual'])])
vs_actual_predicted.index=y_test.index
```

In [101…
```
fpr,tpr,_= metrics.roc_curve(y_test, y_pred_proba)

auc = metrics.roc_auc_score(y_test, y_pred_proba)

plt.plot(fpr, tpr, label='AUC='+str(auc))
plt.plot(fpr, fpr, linestyle='--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Naive Bayes', size=15)
plt.legend(loc=4)
```

Out[101]:
```
<matplotlib.legend.Legend at 0x1e528b3d4c0>
```



Logistic Regression

In [107…
```
from sklearn.linear_model import LogisticRegression
```

In [109…
```
lr = LogisticRegression()
```

In [110…
```
lr.fit(X_train, y_train)
```

Out[110]:
```
LogisticRegression()
```

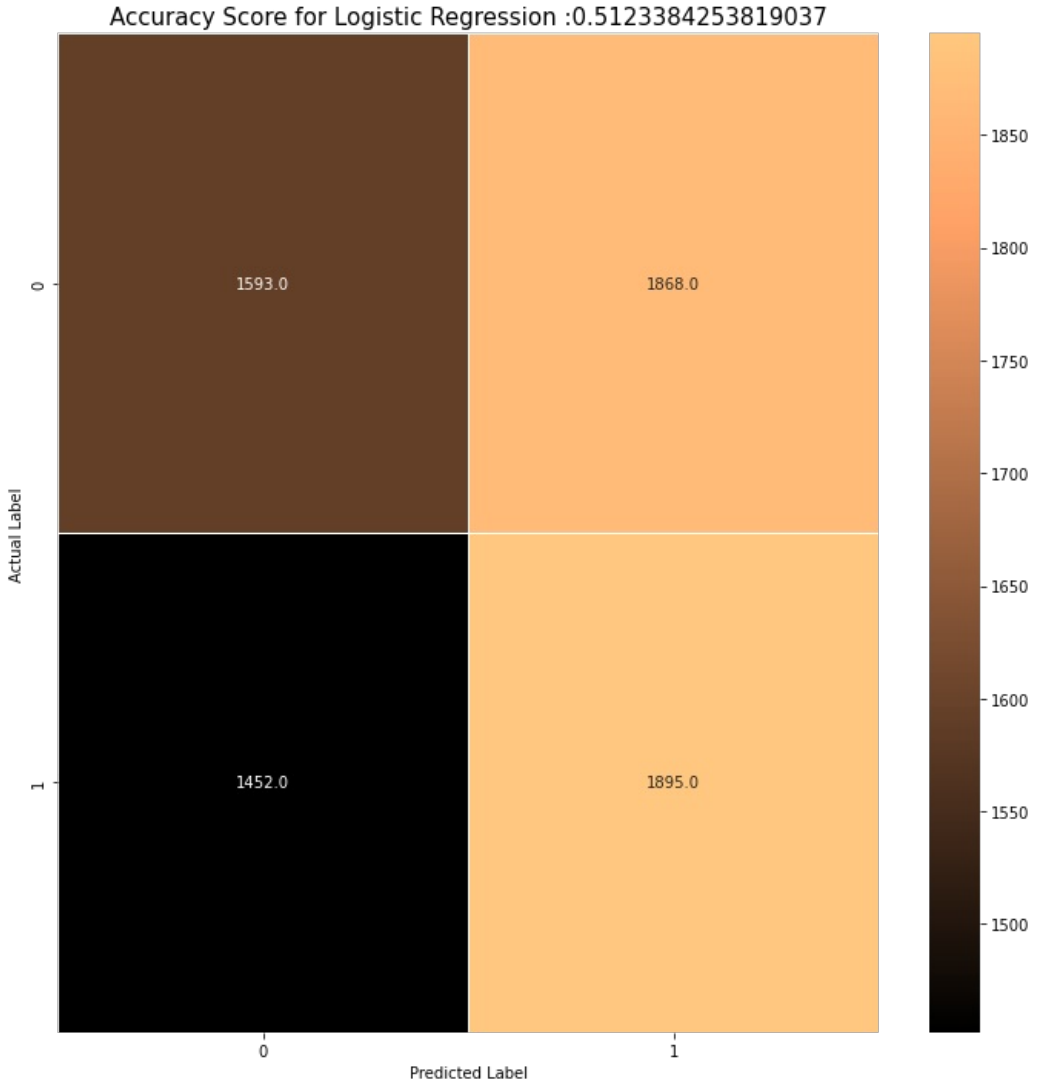
In [111…
```
y_pred = lr.predict(X_test)

print("Accuracy Score:", round(accuracy_score(y_test, y_pred)*100,2), "%")
```

```
Accuracy Score: 51.23 %
```

In [112…
```
print("F-1 Score:", (f1_score(y_test, y_pred, average='micro')))
print("Precision Score:", (precision_score(y_test, y_pred, average='micro')))
print("Recall Score:", (recall_score(y_test, y_pred, average='micro')))
print("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

```
F-1 Score: 0.5123384253819037
Precision Score: 0.5123384253819037
Recall Score: 0.5123384253819037
Jaccard Score: 0.344391785150079
Log Loss: 16.843453478129124
```

In [115…
```
cm = confusion_matrix (y_test, y_pred)

plt.figure(figsize=(10,10))
sns.heatmap ( data=cm,linewidths=.5, fmt='.1f', annot=True, cmap='copper')




plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
all_sample_title = 'Accuracy Score for Logistic Regression :{0}'.format (lr.score(X_test, y_test))
plt.title(all_sample_title, size=15)
plt.tight_layout()
```

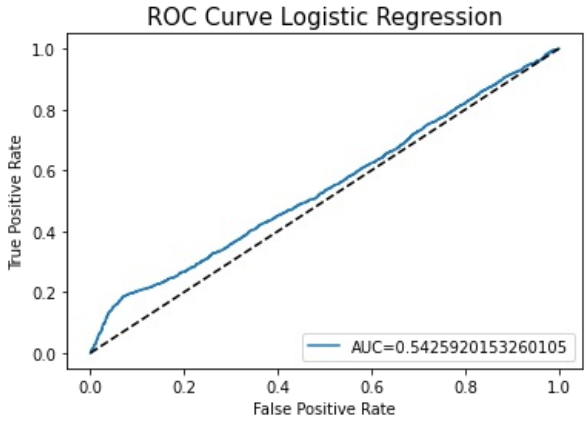


In [116…
```
y_pred_proba = lr.predict_proba(X_test)[:][:,1]
vs_actual_predicted=pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual'])])
vs_actual_predicted.index=y_test.index
```

In [117…
```
fpr,tpr,_= metrics.roc_curve(y_test, y_pred_proba)
```

```
auc = metrics.roc_auc_score(y_test, y_pred_proba)

plt.plot(fpr, tpr, label='AUC='+str(auc))
plt.plot(fpr, fpr, linestyle='--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Logistic Regression', size=15)
plt.legend(loc=4)
```

Out[117]: <matplotlib.legend.Legend at 0x1e529628400>



Support Vector Machine

In [118…
```
from sklearn.svm import SVC
```

In [119…
```
svc= SVC()
svc.fit(X_train, y_train)
```

Out[119]: SVC()

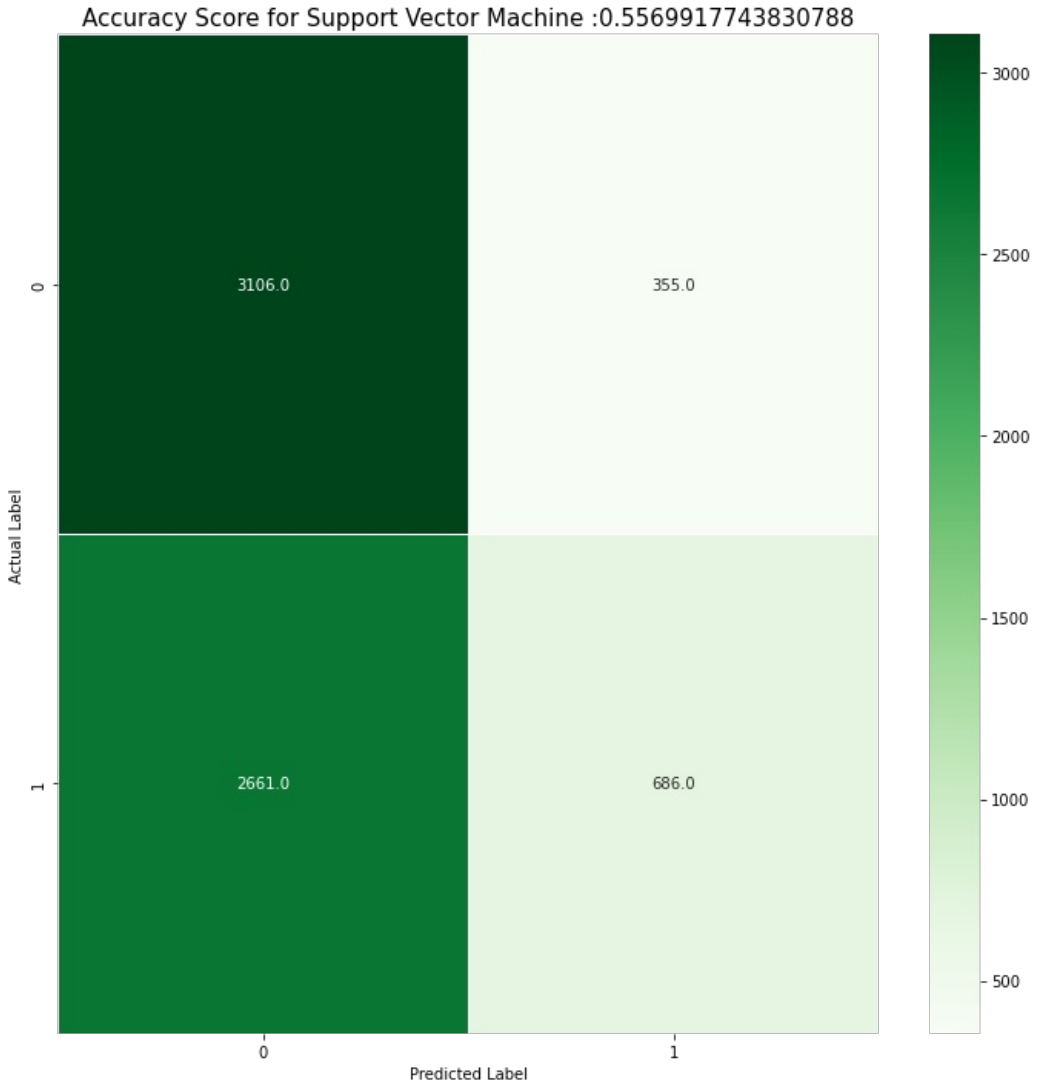
In [120…
```
y_pred = svc.predict(X_test)

print("Accuracy Score:", round(accuracy_score(y_test, y_pred)*100,2), "%")
```

```
Accuracy Score: 55.7 %
```

In [121…
```
print("F-1 Score:", (f1_score(y_test, y_pred, average='micro')))
print("Precision Score:", (precision_score(y_test, y_pred, average='micro')))
print("Recall Score:", (recall_score(y_test, y_pred, average='micro')))
print("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

```
F-1 Score: 0.5569917743830788
Precision Score: 0.5569917743830788
Recall Score: 0.5569917743830788
Jaccard Score: 0.38599348534201955
Log Loss: 15.301003740325843
```

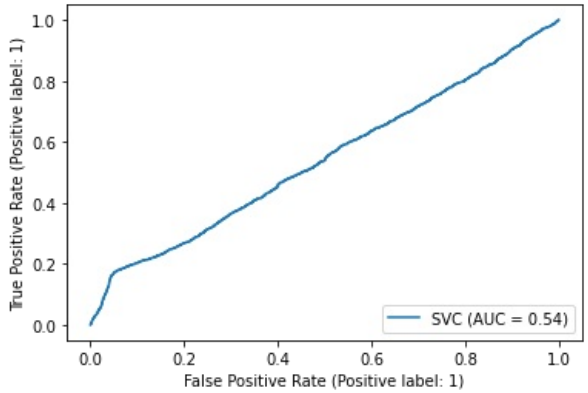
In [122…
```
cm = confusion_matrix (y_test, y_pred)

plt.figure(figsize=(10,10))
sns.heatmap ( data=cm,linewidths=.5, fmt='.1f', annot=True, cmap='Greens')




plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
all_sample_title = 'Accuracy Score for Support Vector Machine :{0}'.format (svc.score(X_test, y_test))
plt.title(all_sample_title, size=15)
plt.tight_layout()
```



In [123…
```
from sklearn.metrics import plot_roc_curve
plot_roc_curve(svc, X_test, y_test)
```

```
2024-07-18 20:41:11,324 [17224] WARNING  py.warnings:109: [JupyterRequire] Function plot_roc_curve is deprecated; Function :func:`plot_roc_curve` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: :meth:`sklearn.metric.RocCurveDisplay.from_predictions` or :meth:`sklearn.metric.RocCurveDisplay.from_estimator`.
```

Out[123]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1e528c54610>
```



In [ ]:

In [ ]:

In [ ]:

In [ ]: