Script File for Final Online Assessment (FOA) submission

BACS3013 Data Science

2021/2022 October Examination

It is an open-book e-assessment. You are allowed to refer to any practical notes/use cases/GitHub/Stack Overflow etc. Anyhow, you **MUST NOT** receive any help whatsoever from any other person. If you need any clarification, please directly ask the lecturer(s)-in-charge.

Please insert your details below:

1) Double click the cell below

2) Type your name and student id

3) press CTRL + Enter

Name: Tay Xue Hao

Student ID: 21WMR04790

Programme: RDSY2S1

Step 1: Put the last three digits of your student id as the my_state_number

eg: if your id is 1902589

You should then write

my_state_number = 589

In [1]:
```
# replace 589 with the last three digits of your student id
# and then press CTRL + Enter
my_state_number = 790
```

Step 2: Read the foa_dataset.csv

In [2]:
```
import pandas as pd
data = pd.read_csv("foa_dataset.csv")

data.head()
```

Out[2]:

	names	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend
0	Jessica Stroud	0.38	0.53	2	157	
1	Daryl Fields	0.80	0.86	5	262	
2	Daisy Anderson	0.11	0.88	7	272	
3	Joseph Fernandez	0.72	0.87	5	223	
4	Herbert Moore	0.37	0.52	2	159	

Step 3: Sample the data randomly the data and save the dataframe as myNewData

In [3]:
```
myNewData = data.sample(frac =.90, replace = False, random_state = my_state_numbe
myNewData.head()
```

Out[3]:

	names	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spe
12825	Pamela Robin	0.89	0.92	3	195	
849	Marcus Stilwell	0.54	0.94	6	294	
10275	Cora Riccio	0.72	0.50	4	245	
14929	Susan Lucas	0.10	0.83	7	302	
7073	Mark Martinez	0.55	0.91	3	243	

Step 4: Start the Analytics using myNewData dataframe as the raw data

Note: Your *myNewData* dataframe may be different from other students' *myNewData* dataframe

Business Undestnading

In this project, I aim to find out the probablity of en employee leaving his or her company based on the variables in the dataset. For example, satisfaction level, last evaluation, number of project, average monthly working hours and time spend in company, Working_accident. The goal of this prject is to get an accuracy of higher than 90%.

Data Understanding and EDA

In the Data Understanding phase my goal is to understand the nature the raw data I received. This is a crucial step in CIRSP-DM as it helps me avoid errors during the Data Preparation phase, which is the most important phase of a Data Science project. I will be using libraries such as numpy, pandas, matplotlib, and seaborn to help me visualize, describe and explore the data in the form of beautiful graphs and plots. These plots will also help me determine the data quality of my dataset.

Import Initial Libraries

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Describing the data

In [6]:
```
myNewData.shape
```

Out[6]:
```
(13499, 11)
```

In [7]:
```
myNewData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 13499 entries, 12825 to 11877
Data columns (total 11 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   names                  13499 non-null  object
 1   satisfaction_level     13499 non-null  float64
 2   last_evaluation        13499 non-null  float64
 3   number_project         13499 non-null  int64
 4   average_montly_hours   13499 non-null  int64
 5   time_spend_company     13499 non-null  int64
 6   Work_accident          13499 non-null  int64
 7   promotion_last_5years  13499 non-null  int64
 8   role                   13499 non-null  object
 9   salary                 13499 non-null  object
 10  class                  13499 non-null  int64
dtypes: float64(2), int64(6), object(3)
memory usage: 1.2+ MB
```

Explanation : The dataset contains 13499 rows and 11 columns. It has one columns containing 2 float values, 6 int values, and 3 object columns. Most of the columns are already in numerical format.

In [8]:
```
myNewData.head()
```

Out[8]:

	names	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spe
12825	Pamela Robin	0.89	0.92	3	195	
849	Marcus Stilwell	0.54	0.94	6	294	
10275	Cora Riccio	0.72	0.50	4	245	
14929	Susan Lucas	0.10	0.83	7	302	
7073	Mark Martinez	0.55	0.91	3	243	

In [9]:
```
myNewData.tail()
```

Out[9]:

	names	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_
8615	Jennifer Young	0.19	0.64	5	231	
9216	Guy Channel	0.92	0.63	5	156	
12818	David Brown	0.75	0.59	3	117	
3840	Christopher Preston	0.71	0.56	3	198	
11877	Tomas Jolly	0.33	0.41	2	198	

Explanation : Looking at the first and last 5 rows of the dataset, I can see clearly the nature of my data better, I can also roguhly guess which factors will lead to employees leaving the company such as satisfaction_level, average_monthly_hours, etc.

In [10]:
```
myNewData.nunique()
```

Out[10]:
```
names                    13296
satisfaction_level          92
last_evaluation             65
number_project               6
average_montly_hours       215
time_spend_company           8
Work_accident                2
promotion_last_5years        2
role                        10
salary                       3
class                        2
dtype: int64
```

Explanation : Here I can see the number of unique values in the individual columns. Some are binary variables like work_accident, promotion_last_5years, and class. Some are categorical like number_project, time_spend_company,role, and salary. rest are numerical.

In [11]:
```
myNewData['role'].unique()
```

Out[11]:
```
array(['marketing', 'sales', 'support', 'IT', 'RandD', 'hr', 'technical',
       'product_mng', 'accounting', 'management'], dtype=object)
```

These are the 10 unique roles in the company.

In [12]:
```
myNewData.describe()
```

Out[12]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_compa
count	13499.000000	13499.000000	13499.000000	13499.000000	13499.000(
mean	0.613219	0.716232	3.805245	200.949848	3.500
std	0.248169	0.170917	1.229454	49.810288	1.460
min	0.090000	0.360000	2.000000	96.000000	2.000(
25%	0.440000	0.560000	3.000000	156.000000	3.000(
50%	0.640000	0.720000	4.000000	200.000000	3.000(
75%	0.820000	0.870000	5.000000	245.000000	4.000(
max	1.000000	1.000000	7.000000	310.000000	10.000(

Discussion: The describe function reveals the real landscape of my data. The tale shows some descriptive statistics about the dataset. The columns I want to focus on using this table are the continuous variables such as average_monthly_hours. From the table, average_monthly_hours does not contain any very extreme outliers as the max and min values are close to the 3rd and 1st quartile respectively. However, time_spend_company column's max value is considered a lot higher than its 3rd quartile, but since this column only have 8 unique values, I would consider this a categorical variable and will likely not remove the outliers.

Interpretation: From the descriptive statistics above, I have gained some valuable insights about my data. The columns io instance ratio is very good(13499 columns, 11 rows). There are very few columns with extreme outliers.

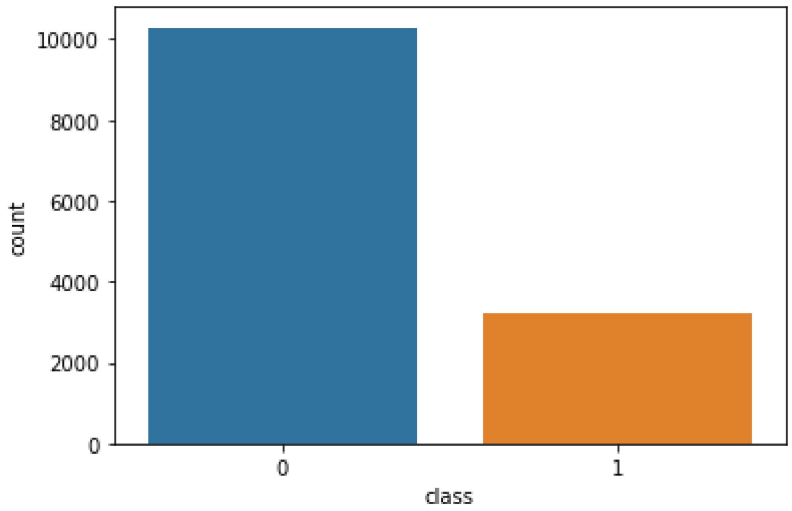
Data Visualization in Graphs and Plots

- Countplot for Class

In [18]:
```
myNewData['class'].value_counts()
```

Out[18]:
```
0    10284
1     3215
Name: class, dtype: int64
```

In [20]:
```
sns.countplot(data=myNewData, x='class')
plt.show()
```



Discussion: There are obviously more instances of Class 0 and Class 1 in the dataset so the classes are not equally distributed. The dataset is a little bit imbalanced and might lead to naive behaviours where the models predicts all 0 but still get a very high accuracy. In the evaluation phase, accuracy should not be the only metrics used to evaluate our models, metrics such as precision, recall, F-measure should be used.

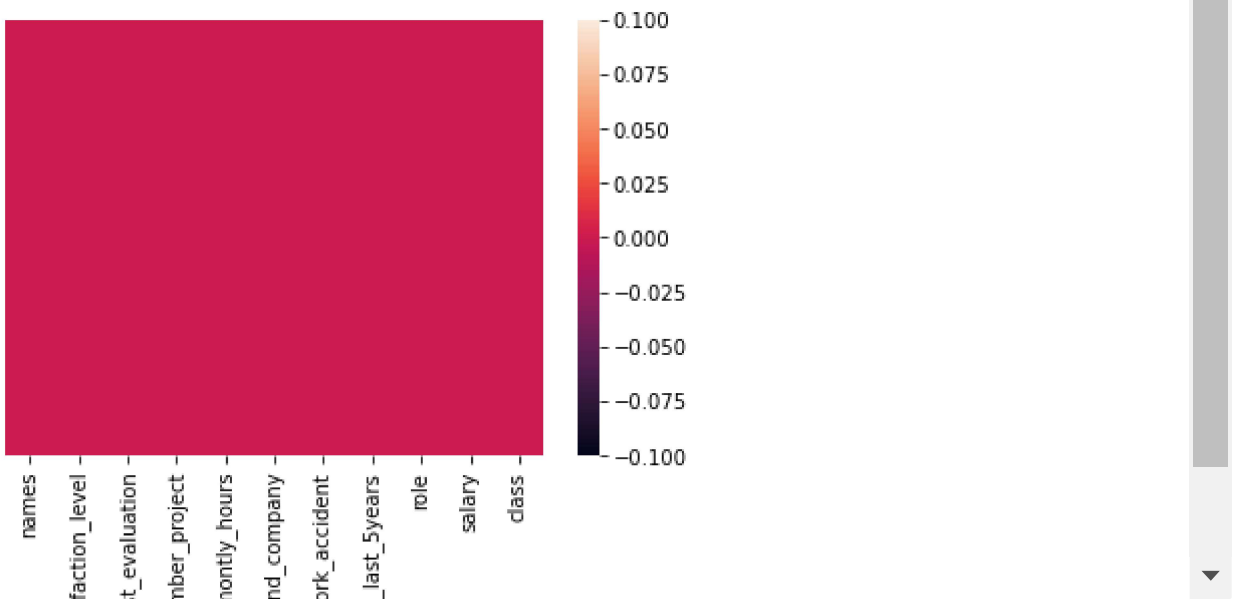
- Check for missing values

In [25]:
```
myNewData.isnull().sum()
```

Out[25]:
```
names                    0
satisfaction_level       0
last_evaluation          0
number_project           0
average_montly_hours     0
time_spend_company       0
Work_accident            0
promotion_last_5years    0
role                     0
salary                   0
class                    0
dtype: int64
```

In [24]:
```
sns.heatmap(myNewData.isnull(), yticklabels=False)
```

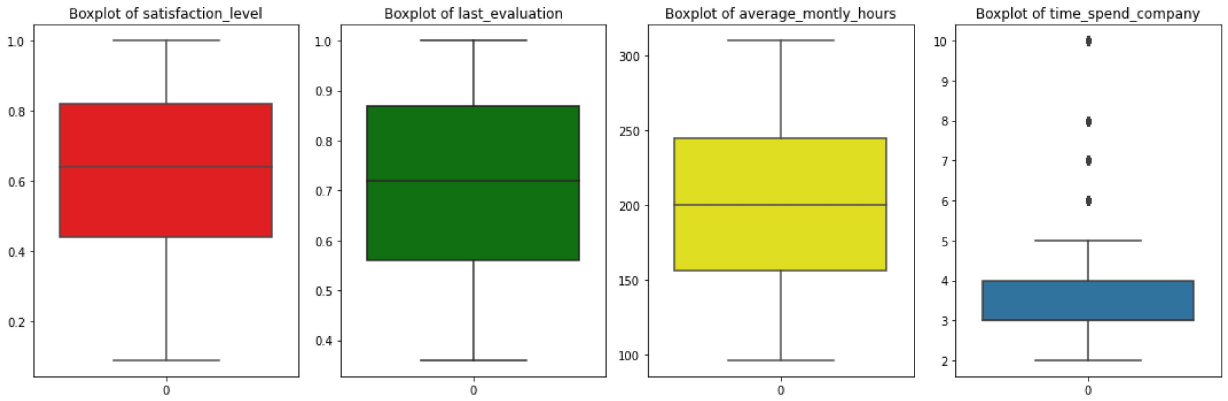
Out[24]:
```
<AxesSubplot:>
```



Explanation: Using seaborn's heatmap, I found out that there are no missing values in my dataset, this means that I do not have to perform handling of missing values in the Data Preparation stage. Horaay!!

- Check for outliers

In [36]:
```
# Using boxplot
plt.figure(figsize=(15,5))

plt.subplot(1,4,1)
sns.boxplot(data=myNewData.satisfaction_level, color = 'Red')
plt.title("Boxplot of satisfaction_level")

plt.subplot(1,4,2)
sns.boxplot(data=myNewData.last_evaluation, color = 'Green')
plt.title("Boxplot of last_evaluation")

plt.subplot(1,4,3)
sns.boxplot(data=myNewData.average_montly_hours, color = 'Yellow')
plt.title("Boxplot of average_montly_hours")

plt.subplot(1,4,4)
sns.boxplot(data=myNewData.time_spend_company)
plt.title("Boxplot of time_spend_company")

plt.tight_layout()
```

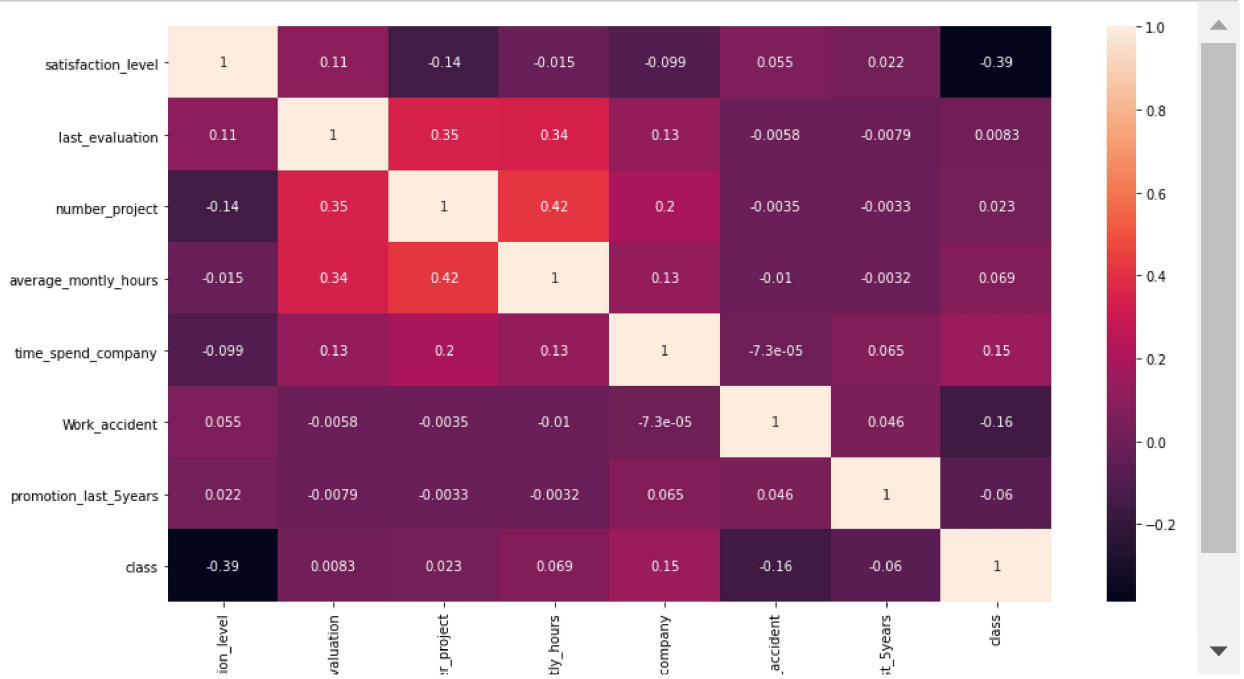


Discussion: From the boxplots above, we can tell that the all the columns have no outliers, I considered time_spend_company a categorical variable so I would have to further explore to decide whether to remove them or not. The other feature columns that I didn't draw the boxplot for are categorical as well.

- Outliers of features by Class

- Correlation Matrix between the varables

In [44]:
```
plt.figure(figsize=(15,8))
sns.heatmap(myNewData.corr(), annot=True)
plt.show()
```

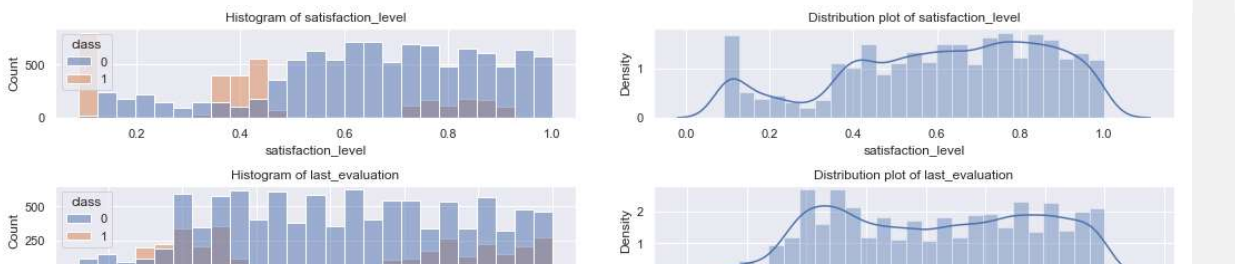


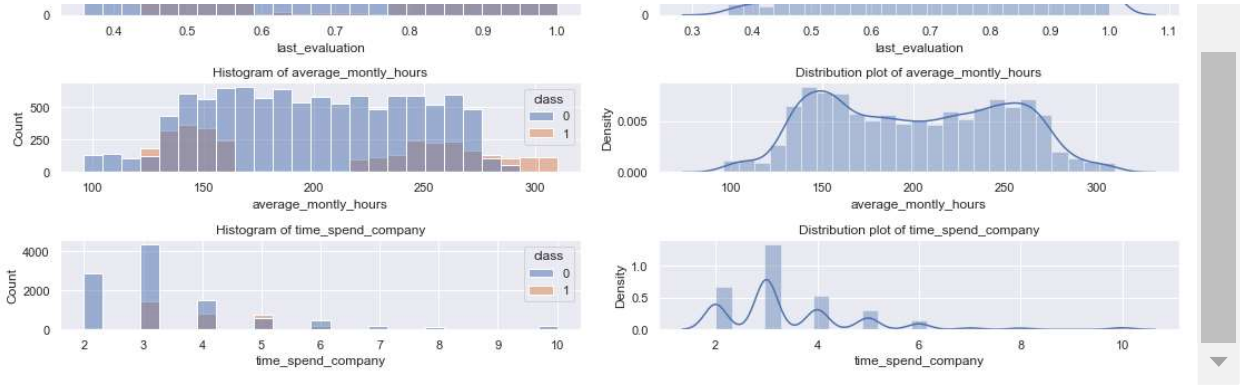
Discussion: This correlation matrix will show the correlations of the variables. The row in the matrix above that I want to focus on is the last row, which is the correlation between the feature columns and the target variable 'class'. From the matrix, it is obvious that satisfaction_level and work_accident have a moderate negative correlation with the 'class' whereas time_spend_company have a positive correlation with Class. Since Class 1 indicates the employee is likely going to leave, we can guess that the longer the employee spends in the company, he/she will likely want to leave. Likewise, if the employee has a high satisfaction level with the company, he or she will likely stay. The other feautures have a weak correlation, but these values can be misleading since the features and target are categorical.

- Histogram and Distribution plot

In [49]:

```
plt.figure(figsize=(15,8))
sns.set_theme()
# Feature_1 histogram
plt.subplot(4,2,1)
sns.histplot(x='satisfaction_level', hue = 'class', data=myNewData, bins = 25, st
plt.title("Histogram of satisfaction_level")

# Feature_1 Distplot
plt.subplot(4,2,2)
sns.distplot(myNewData['satisfaction_level'], bins =25)
plt.title("Distribution plot of satisfaction_level")

# Feature_2 histogram
plt.subplot(4,2,3)
sns.histplot(x='last_evaluation', hue = 'class', data=myNewData, bins = 25, stat
plt.title("Histogram of last_evaluation")

# Feature_2 Distplot
plt.subplot(4,2,4)
sns.distplot(myNewData['last_evaluation'], bins =25)
plt.title("Distribution plot of last_evaluation")

# Feature_3 histogram
plt.subplot(4,2,5)
sns.histplot(x='average_montly_hours', hue = 'class', data=myNewData, bins = 25,
plt.title("Histogram of average_montly_hours")


# Feature_3 Distplot
plt.subplot(4,2,6)
sns.distplot(myNewData['average_montly_hours'], bins =25)
plt.title("Distribution plot of average_montly_hours")

# Feature_4 histogram
plt.subplot(4,2,7)
sns.histplot(x='time_spend_company', hue = 'class', data=myNewData, bins = 25, st
plt.title("Histogram of time_spend_company")

# Feature_4 Distplot
plt.subplot(4,2,8)
sns.distplot(myNewData['time_spend_company'], bins =25)
plt.title("Distribution plot of time_spend_company")


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

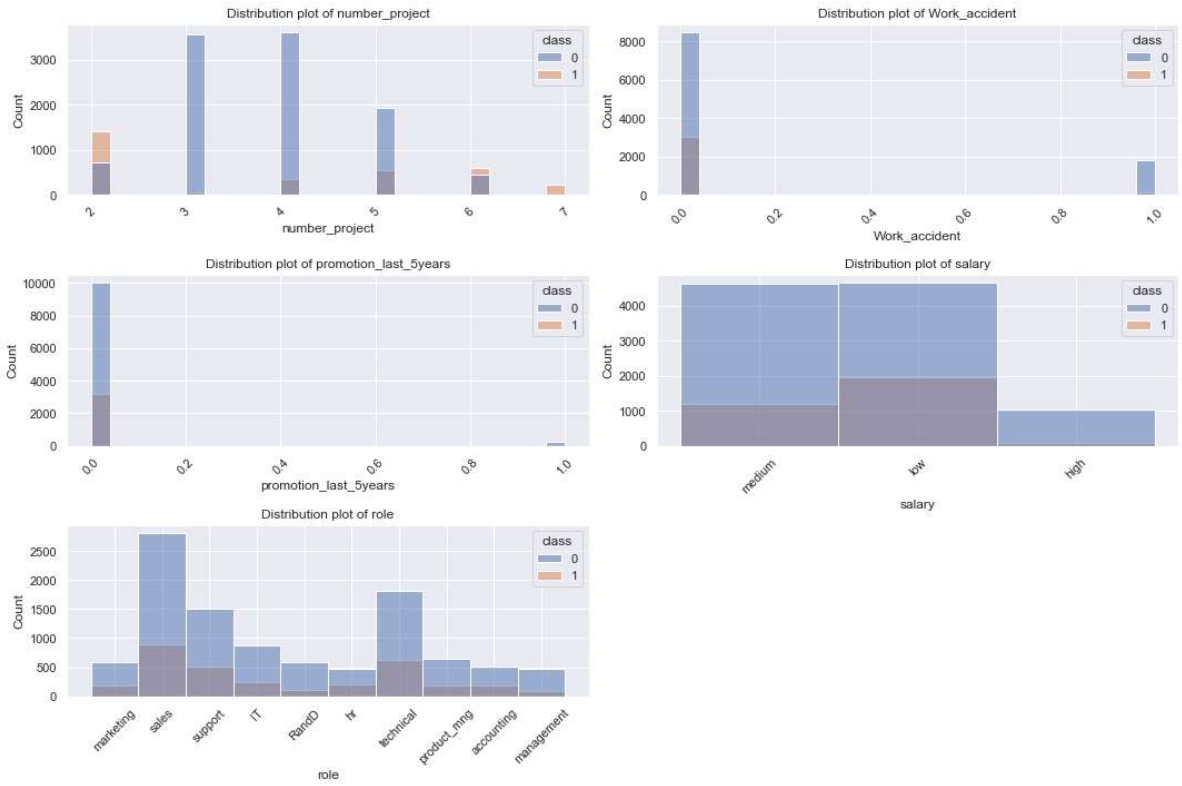




Discussion: First, I want to discuss about the histograms. In the histogram of satisfaction_level, we can see clearly that employees that rated their company with high satisfaction_level will belong to the Class 0 (Stay). In the histogram of last_evaluation, the employees who have around 0.45-0.6 to 0.8-1 are more ikely to leave. In the average_monthly_hours variable, the employees around 150 hours and 220 to 300 are more likely to leave the company. In time_spend_company, he employees who spent 3 to 5 years are more likely to leave. Next, I want to focus on the distribution of the variables.The distributiod of the satisfaction_level is left_skewed, which means normalizing is needed later on. The distribution of the last_evalutaion and average_monthly_hours does not follow a bell-shaped curve and the time_spend_company's distribution is right-skewed.ddd

- Histogram for other categorical variables

In [71]:
```
plt.figure(figsize=(15,10))
sns.set_theme()

plt.subplot(3,2,1)
sns.histplot(x='number_project', hue = 'class', data=myNewData, bins = 25, stat =
plt.title("Distribution plot of number_project")
plt.xticks(rotation=45)

plt.subplot(3,2,2)
sns.histplot(x='Work_accident', hue = 'class', data=myNewData, bins = 25, stat =
plt.title("Distribution plot of Work_accident")
plt.xticks(rotation=45)

plt.subplot(3,2,3)
sns.histplot(x='promotion_last_5years', hue = 'class', data=myNewData, bins = 25,
plt.title("Distribution plot of promotion_last_5years")
plt.xticks(rotation=45)


plt.subplot(3,2,4)
sns.histplot(x='salary', hue = 'class', data=myNewData,  bins = 25, stat = 'count
plt.title("Distribution plot of salary")
plt.xticks(rotation=45)

plt.subplot(3,2,5)
sns.histplot(x='role', hue = 'class', data=myNewData, bins = 25, stat = 'count')
plt.title("Distribution plot of role")


plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Discussion: It is hard to determine the Class using the number_project variable as there are no clear pattern. It is also hard to determine the Class using the role varable as both of the Classes follows a same distribution.

In [74]:
```
cols = ['satisfaction_level','last_evaluation','average_montly_hours', 'time_sper
sns.pairplot(data=myNewData[cols], hue='class', palette = "husl")
plt.show()
```

Summary :

To summarize the Data Understanding phase, we have gained a lot of knowledge and insights

about our data. First, our predictor to sample ratio is good. Second, our data is a little imbalanced with more instance of Class 0 than Class 1. Third, our data have very little outliers that need to be dealt with. Lastly, working_accident and satisfaaction_level have a moderate negative correlation with the class whereas time_spend_company have a positive correlation with class.

Data Preparation

Data Preparation is the most important stage in CRISP-DM. In this stage, I will be performing some data cleaning to fix the data quality issues that I have identified in the Data Preparation phase. If these issues are not fixed, the performance of machine learning models will be affected. The goal of this phase is to make sure the Modeling phase will produce accurate, consistent, and reliable predictions of Classes. I do not need to perform handling of missing values as I have found out in the exploratory data analysis stage that my data does not contain a single missing values.

Feature Selection

Feature selection is done in this phase to reduce the model complexity. This way we can avoid overfitting our models. Training time of models will also reduce and accuracy will increase as well because of less misleading variables in our data.

In [89]:
```
data = myNewData.drop(['names','number_project', 'role'], axis=1)
data.head()
```

Out[89]:

	satisfaction_level	last_evaluation	average_montly_hours	time_spend_company	Work_accide
12825	0.89	0.92	195	2	
849	0.54	0.94	294	3	
10275	0.72	0.50	245	3	
14929	0.10	0.83	302	5	
7073	0.55	0.91	243	4	

Explanation: Using the combination of domain knowledge and statistical evidence, I dropped the names column because it is not useful in predicting the class. The number_project column does not have a distribution that can distinguish the class. Role likely will not affect the employee's desire to stay as well, according to my experience in my internship. The rest of the variables are likely useful in classification. Using domain knowledge, I also can guarantee that salary will affect employee stay.

In [90]:
```
data.shape
```

Out[90]:
```
(13499, 8)
```

Explanation: Now I have 7 input variables and 1 target variable for my model.

Encoding the Salary Variable

In [92]:
```
scale_mapper = {"low":1, "medium":2, "high":3}
data["salary"] = data["salary"].replace(scale_mapper)
data.head()
```

Out[92]:

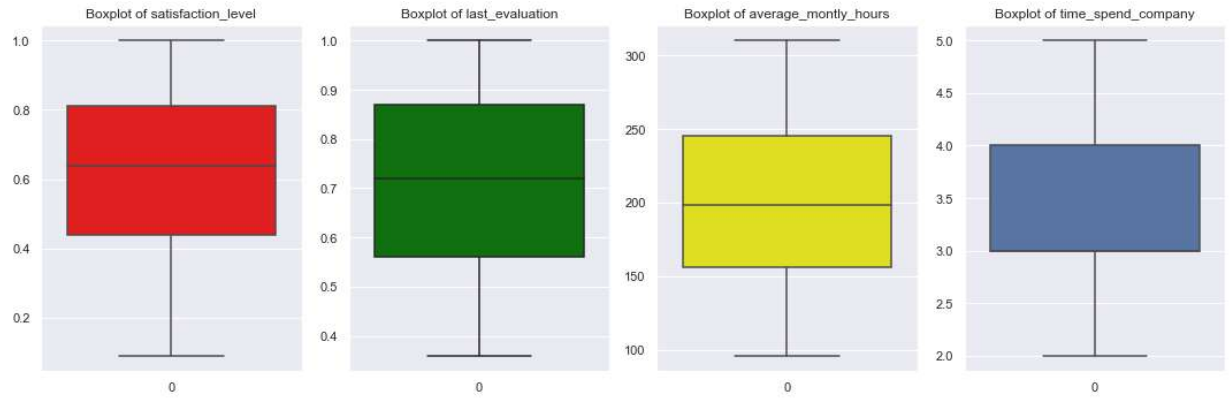
	satisfaction_level	last_evaluation	average_montly_hours	time_spend_company	Work_accide
12825	0.89	0.92	195	2	
849	0.54	0.94	294	3	
10275	0.72	0.50	245	3	
14929	0.10	0.83	302	5	
7073	0.55	0.91	243	4	

Explanation : I encoded the salary variable to low=1, medium=2, high=3 as it is an ordinal variable. This is important as machine learning models only work with numerical data.

Handling Outliers

- Boxplot before removing outleirs

In [125]:
```
plt.figure(figsize=(15,5))

plt.subplot(1,4,1)
sns.boxplot(data=clean_data.satisfaction_level, color = 'Red')
plt.title("Boxplot of satisfaction_level")

plt.subplot(1,4,2)
sns.boxplot(data=clean_data.last_evaluation, color = 'Green')
plt.title("Boxplot of last_evaluation")

plt.subplot(1,4,3)
sns.boxplot(data=clean_data.average_montly_hours, color = 'Yellow')
plt.title("Boxplot of average_montly_hours")

plt.subplot(1,4,4)
sns.boxplot(data=clean_data.time_spend_company)
plt.title("Boxplot of time_spend_company")

plt.tight_layout()
```



In [109]:
```
Q1 = data.time_spend_company.quantile(0.25)
Q3 = data.time_spend_company.quantile(0.75)
IQR = Q3-Q1
LL = Q1 - 1.5*IQR
UL = Q3 + 1.5*IQR

before = ((data.time_spend_company < LL) | (data.time_spend_company > UL)).sum()
print("Number of outliers for column Time_spend_company : ", before)
```

```
Number of outliers for column Time_spend_company :  1165
```

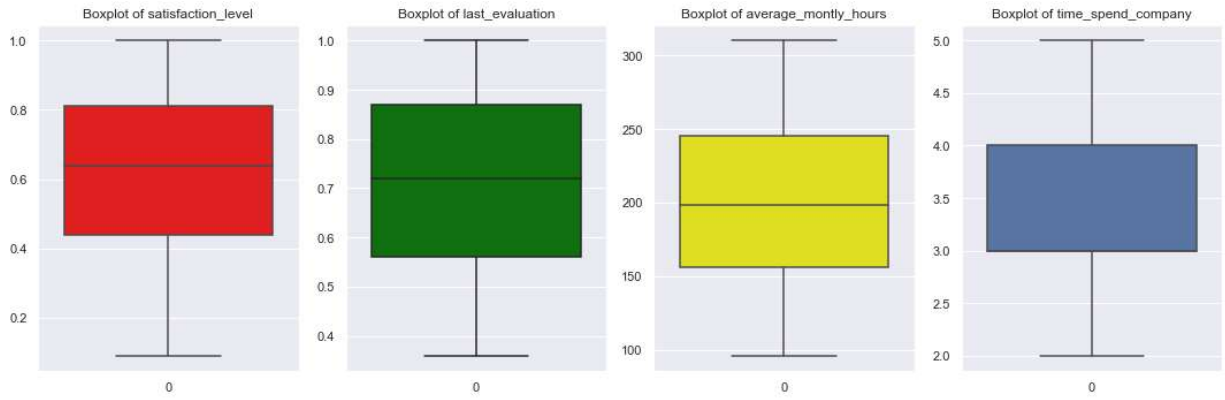
Explanation : Since the number of outliers are not a lot, I can safely remove them as it won't affect the performance of the models.

In [113]:
```
cols = ['time_spend_company']
clean_data = data[~((data[cols] < (LL)) |(data[cols] > (UL))). any(axis=1)]
after = ((clean_data.time_spend_company  < (LL)) | (clean_data.time_spend_company
print("Number of outliers for column Time_spend_company after outlier removal: "
```

```
Number of outliers for column Time_spend_company after outlier removal:  0
```

- Boxplot after removing outleirs

In [124]:
```
plt.figure(figsize=(15,5))

plt.subplot(1,4,1)
sns.boxplot(data=clean_data.satisfaction_level, color = 'Red')
plt.title("Boxplot of satisfaction_level")

plt.subplot(1,4,2)
sns.boxplot(data=clean_data.last_evaluation, color = 'Green')
plt.title("Boxplot of last_evaluation")

plt.subplot(1,4,3)
sns.boxplot(data=clean_data.average_montly_hours, color = 'Yellow')
plt.title("Boxplot of average_montly_hours")

plt.subplot(1,4,4)
sns.boxplot(data=clean_data.time_spend_company)
plt.title("Boxplot of time_spend_company")

plt.tight_layout()
```



-Explanation : The outliers are now removed, HOORAY!

- Descriptive statistics of data after removal of outliers.

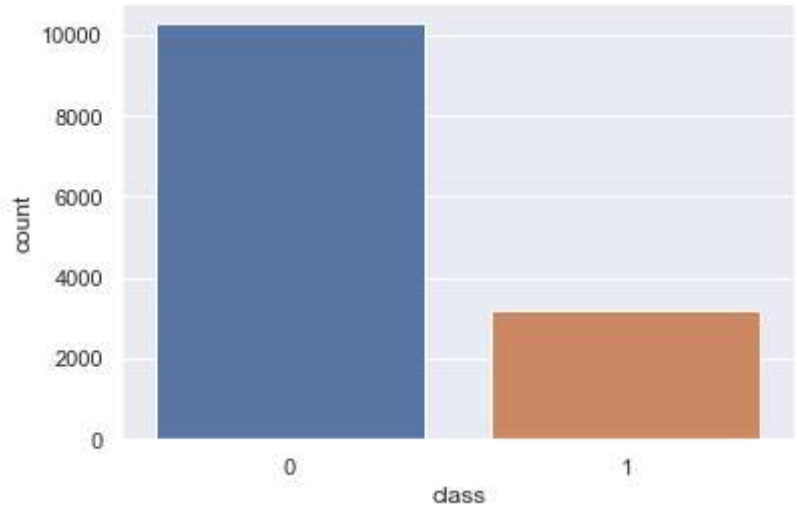
In [114]:
```
clean_data.describe()
```

Out[114]:

	satisfaction_level	last_evaluation	average_montly_hours	time_spend_company	Work_accide
count	12334.000000	12334.000000	12334.000000	12334.000000	12334.0000
mean	0.611853	0.714348	200.458489	3.164829	0.1435
std	0.247681	0.170214	49.769036	0.909418	0.3506
min	0.090000	0.360000	96.000000	2.000000	0.0000
25%	0.440000	0.560000	156.000000	3.000000	0.0000
50%	0.640000	0.720000	198.000000	3.000000	0.0000
75%	0.810000	0.870000	245.000000	4.000000	0.0000
max	1.000000	1.000000	310.000000	5.000000	1.0000

Discussion: After removing the outliers I still have 12334 rows in the data which is still a lot for machine learning models. The changes occur in the max value of the time_spend_company column. The max value was 10 and now it is 5 which is much closer to the 3rd quartile and is within the initial Upper Limit.

In [120]:
```
sns.countplot(data=myNewData, x='class')
plt.show()
```



Explanation: The distribution does not change much after removal which is great!

Splitting the data

Now I will split my data into /outputtarget/independent variables and input/dependent/feature variables(Matrix)

In [129]:
```
X = clean_data.drop(['class'],axis=1)
y = clean_data['class']
X.tail()
```

Out[129]:

	satisfaction_level	last_evaluation	average_montly_hours	time_spend_company	Work_accide
8615	0.19	0.64	231	4	
9216	0.92	0.63	156	3	
12818	0.75	0.59	117	3	
3840	0.71	0.56	198	3	
11877	0.33	0.41	198	4	

In [130]:
```
y.tail()
```

Out[130]:
```
8615     0
9216     0
12818    0
3840     0
11877    0
Name: class, dtype: int64
```

- Now I will split my data into train sets and test sets, where test size is 30% of the data.

In [131]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
print("Size of Train sets X, y: ", X_train.shape, y_train.shape)
print("Size of Test sets X, y: ", X_test.shape, y_test.shape)
```

```
Size of Train sets X, y:  (8633, 7) (8633,)
Size of Test sets X, y:  (3701, 7) (3701,)
```

Discussion: 7 input variables, and 1 output variable

In [132]:
```
print("Number of Class 1 in training set :", np.count_nonzero(y_train))
print("Number of Class 1 in testing set  :", np.count_nonzero(y_test))
print("Number of Class 0 in training set :", y_train.size - np.count_nonzero(y_tr
print("Number of Class 0 in testing set  :", y_test.size - np.count_nonzero(y_tes
```

```
Number of Class 1 in training set : 2089
Number of Class 1 in testing set  : 931
Number of Class 0 in training set : 6544
Number of Class 0 in testing set  : 2770
```

Explanation : I calculated the number of Class 1 and 0 in the training and testing sets to make sure the Classes are not too imbalanced.

Normalizing and Standardizing the data

Now I will rescale my data using Normalizing/MinMaxScaler and Standardizing. This step is very important because the features have very different ranges, where average_monthly_hhours is between 100 to 300. If we do not normalize/standardize, average_monthly_hours will intrinsically influence the output more due to it's large values. So we normalize/standardize to bring all of them to the same range(uniform). Distance based algorithms such as Support Vector Machine, Logistic Regression and K-Nearest Neighbors requires data to be rescaled. Not doing so will affect the distance between points greatly.

Standardizing

In [133]:
```
from sklearn.preprocessing import StandardScaler
standardizer = StandardScaler()
X_train_std = standardizer.fit_transform(X_train)
X_test_std = standardizer.transform(X_test)
X_train_std
```

Out[133]:
```
array([[ 0.19037821, -0.48124737,  1.29942214, ..., -0.40973674,
        -0.13297527, -0.91106686],
       [-1.19433726, -1.42237514, -1.14420647, ..., -0.40973674,
        -0.13297527,  0.6740279 ],
       [ 0.19037821, -1.30473417,  0.12809603, ..., -0.40973674,
        -0.13297527,  0.6740279 ],
       ...,
       [ 0.84200902, -1.30473417, -0.39698119, ..., -0.40973674,
        -0.13297527, -0.91106686],
       [ 1.41218598, -0.06950397,  0.55219686, ..., -0.40973674,
        -0.13297527,  0.6740279 ],
       [ 0.55692054,  0.16577797, -0.27580953, ...,  2.44059147,
        -0.13297527,  0.6740279 ]])
```

Explanation : This technique rescales the feature values in such a way that it have the properties of a standard normal distribution with mean = 0 and standard deviation = 1.

Normalizing

In [134]:
```
from sklearn.preprocessing import MinMaxScaler
normalizer = MinMaxScaler()
X_train_nom = normalizer.fit_transform(X_train)
X_test_nom = normalizer.transform(X_test)
X_train_nom
```

Out[134]:
```
array([[0.62637363, 0.421875  , 0.78504673, ..., 0.        , 0.        ,
        0.        ],
       [0.25274725, 0.171875  , 0.21962617, ..., 0.        , 0.        ,
        0.5       ],
       [0.62637363, 0.203125  , 0.51401869, ..., 0.        , 0.        ,
        0.5       ],
       ...,
       [0.8021978 , 0.203125  , 0.39252336, ..., 0.        , 0.        ,
        0.        ],
       [0.95604396, 0.53125   , 0.61214953, ..., 0.        , 0.        ,
        0.5       ],
       [0.72527473, 0.59375   , 0.42056075, ..., 1.        , 0.        ,
        0.5       ]])
```

Explanation : This technique rescales the feature values into ranges of [0-1] and tranforms the distribution of the data into Gaussian/Bell-shaped.

Summary :

To summarize my Data Preparation phase,I have selected the features to do classification, I have removed the outliers, Split the data into train and test sets (70:30) and I have rescaled the input sets(X_train,X_test). Now my data is clean enough to be fit into the machine learning models. Hooray!

Modeling

This is the most interesting and fun phase of CRISP-DM. In ths phase I will select a few machine learning models that best suit my data to predict the target variable(Class) using the features(feature_1,2,3). The goal of this phase is to train the models and get an accuracy of more than 80% and have no naive behaviour where the model only predicts the majority class(Class 0).

The Machine Learning Algorithms that I will use are as follows :

- K-Nearest Neighbors
- Random Forest Classifier
- Multilayer Neural Network(My favourite!)

Import Machine Learning Libraries

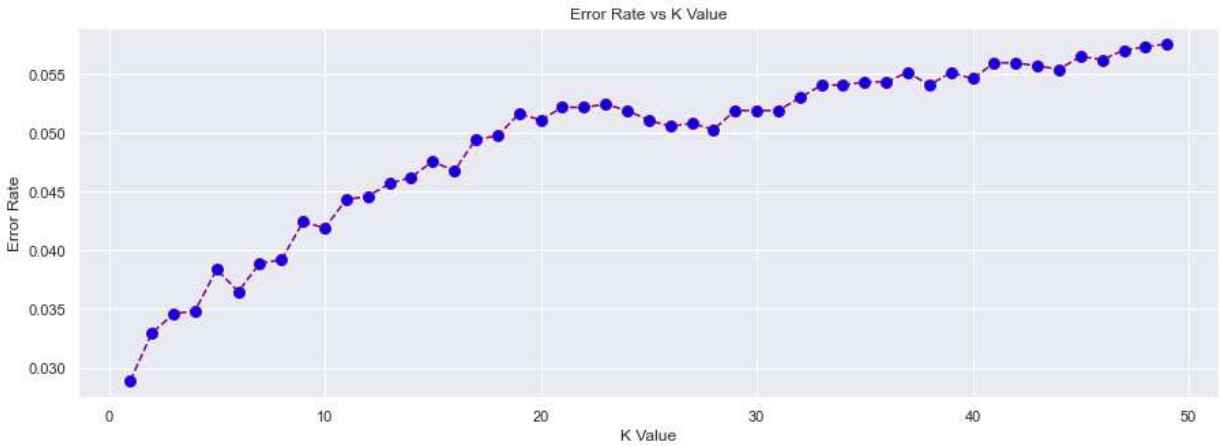
In [136]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

# For deep learning
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from tensorflow.keras.optimizers import Adam

# Dear Miss, tensorflow and keras needs to be installed to run this cell. P.S.
# pip install tensorflow-gpu
# pip install tensorflow
# pip install keras
```

K-Nearest Neighbors

I chose KNN because it is simple to understand,fast to train and have high accuracy. It is also very good for classification problems although it can also deal with regression problems. I will use both normalized data and standardized data in KNN to compare and decide whether normalized or standardized data is better. I will use the better one for the other models.

- Elbow method

In [137]:
```
error_rate = []

for i in range(1,50):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train_nom,y_train)
    predict_error = knn.predict(X_test_nom)
    error_rate.append(np.mean(predict_error != y_test))

plt.figure(figsize=(15,5))
plt.plot(range(1,50),error_rate, color = 'purple',linestyle='dashed', marker='o',
plt.title('Error Rate vs K Value')
plt.xlabel('K Value')
plt.ylabel('Error Rate')
plt.show()
```



Explanation : The figure above is the elbow method which I have used to find the optimal K value that gives me the lowest error rate. The lower the error the better the K value. From the plot, it is obvious that 3 is my optimal K value, luckily for me, it is an odd number and is not too big or too small. If a K value is even, there will be a problem when the classification is 50% Class 0 and 50% Class 1. The model will not know which on to choose from. I do not choose K = 1 or 2 which gives me better result but are meaningless for classification.

- Model building

- Normalized Data

In [139]:
```
# Using Normalized Data
knn_nom = KNeighborsClassifier(n_neighbors=3)
knn_nom.fit(X_train_nom,y_train)
knn_pred_nom = np.around(knn_nom.predict(X_test_nom))
knn_score_nom = accuracy_score(knn_pred_nom, y_test)
knn_metrics_nom = metrics.classification_report(y_test, knn_pred_nom)
print(knn_metrics_nom)
```

```
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      2770
           1       0.93      0.93      0.93       931

    accuracy                           0.97      3701
   macro avg       0.95      0.95      0.95      3701
weighted avg       0.97      0.97      0.97      3701
```

Discussion : Using normalized data, KNN produces a pretty good result with accuracy=0.97, precision=0.98, and recall=0.98 for Class 0. Precision = 0.93 and recall = 0.93 for Class 1.

In [140]:
```
# Using Standardized Data
knn_std = KNeighborsClassifier(n_neighbors=3)
knn_std.fit(X_train_std,y_train)
knn_pred_std = np.around(knn_std.predict(X_test_std))
knn_score_std = accuracy_score(knn_pred_std, y_test)
knn_metrics_std = metrics.classification_report(y_test, knn_pred_std)
print(knn_metrics_std)
```

```
              precision    recall  f1-score   support

           0       0.98      0.97      0.98      2770
           1       0.92      0.94      0.93       931

    accuracy                           0.97      3701
   macro avg       0.95      0.96      0.95      3701
weighted avg       0.97      0.97      0.97      3701
```

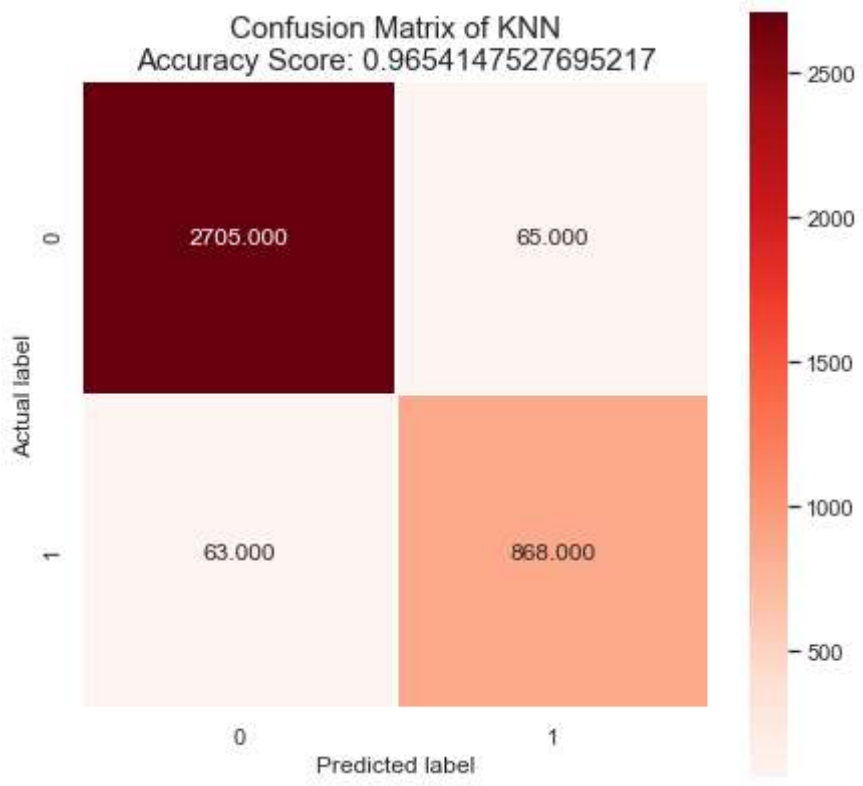
Discussion : Using standardized data, KNN produces a pretty good result with accuracy=0.97, precision=0.98, and recall=0.97 for Class 0. Precision = 0.92 and recall = 0.94 for Class 1.

Decision: According to the F1-score(weighted average of precision and recall) for both Classes, both rescaling methods give the same result. The accuracy are the same as well. However, I decided to use normalized data for the following models as some of my features do not follow Gaussian distribution, normalizing will rescale them into bell-shaped distribution. Therefore, by theory normalizing is the better method for my dataset.

In [141]:
```
knn_precision = metrics.precision_score(y_test,knn_pred_nom)
knn_recall = metrics.recall_score(y_test,knn_pred_nom)
```

- Confusion Matrix(Normalized)

In [142]:
```
knn = knn_nom
knn_cm = metrics.confusion_matrix(y_test, knn_pred_nom)
plt.figure(figsize=(7,7))
sns.heatmap(knn_cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap =
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
accuracy = 'Accuracy Score: {0}'.format(knn_score_nom)
plt.title("Confusion Matrix of KNN\n" + accuracy, size = 15)
plt.show()
```



Interpretation : Even with the imbalanced nature of my data where Class 0 is significantly higher count than Class 1, the model is still able to perform well in predicting Class 1. HOORAY! Let us see how random forest classifier performs.

Random Forest Classifier

- Model Building and Hyper-parameter tuning

Hyperparameter Tuning

In [144]:
```
# Number of trees in random forest
n_estimators = np.arange(10,200,20)
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [2,4]
# Minimum number of samples required to split a node
min_samples_split = [2,5]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1,2]
# Method of selecting samples for training each tree
bootstrap = [True, False]
```

In [145]:
```
param_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap
             }

print(param_grid)
```

```
{'n_estimators': array([ 10,  30,  50,  70,  90, 110, 130, 150, 170, 190]), 'ma
x_features': ['auto', 'sqrt'], 'max_depth': [2, 4], 'min_samples_split': [2,
5], 'min_samples_leaf': [1, 2], 'bootstrap': [True, False]}
```

In [146]:
```
rf_model = RandomForestClassifier( random_state = 32)

rf_Grid = GridSearchCV(estimator = rf_model, param_grid=param_grid, cv=3, verbose
rf_Grid.fit(X_train_nom, y_train)
```

```
Fitting 3 folds for each of 320 candidates, totalling 960 fits
```

Out[146]:
```
GridSearchCV(cv=3, estimator=RandomForestClassifier(random_state=32), n_jobs=4,
             param_grid={'bootstrap': [True, False], 'max_depth': [2, 4],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2],
                         'min_samples_split': [2, 5],
                         'n_estimators': array([ 10,  30,  50,  70,  90, 110, 1
30, 150, 170, 190])},
             verbose=2)
```

In [147]:
```
rf_Grid.best_params_
```

Out[147]:
```
{'bootstrap': True,
 'max_depth': 4,
 'max_features': 'auto',
 'min_samples_leaf': 1,
 'min_samples_split': 5,
 'n_estimators': 10}
```

Explanation : For the model, I performed hyper parameter tuning to boost the performance of the model by finding out the best parameters for the model. The best parameters are {'bootstrap':

model by finding out the best parameters for the model. The best parameters are {'bootstrap': True, 'max_depth': 4, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 10}.

In [148]:
```
print('Train Accuracy : ',rf_Grid.score(X_train_nom,y_train))
print('Test Accuracy  : ',rf_Grid.score(X_test_nom,y_test))
rfGrid_predict = rf_Grid.predict(X_test_nom)
rf_score = accuracy_score(rfGrid_predict, y_test)
print(rf_score)

rf_precision = metrics.precision_score(y_test,rfGrid_predict)
rf_recall = metrics.recall_score(y_test,rfGrid_predict)
```

```
Train Accuracy :  0.973126375535735
Test Accuracy  :  0.973250472845177
0.973250472845177
```

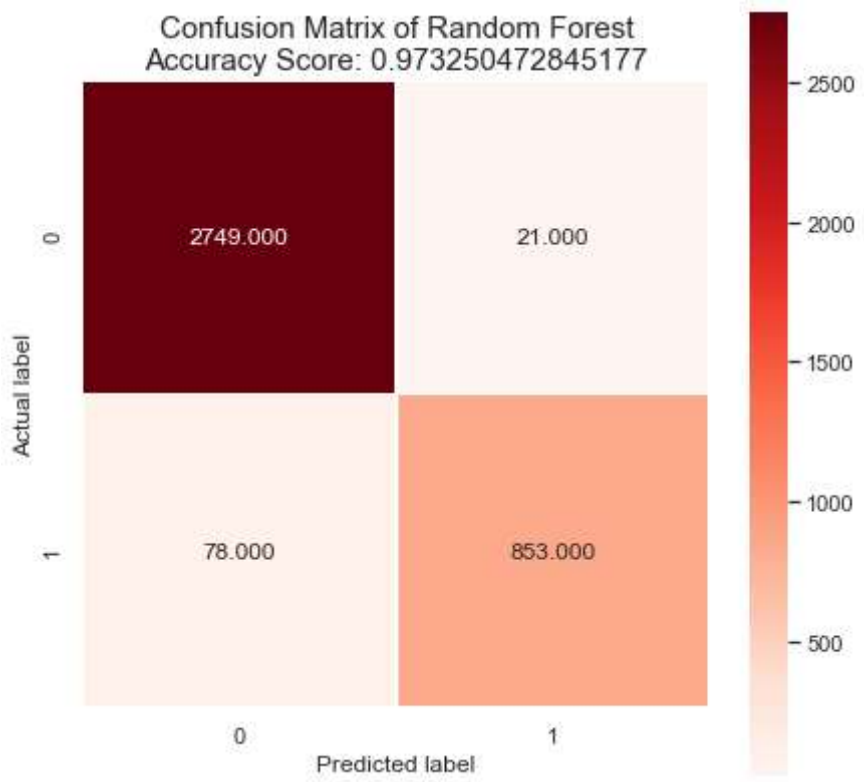
- Classification Report

In [152]:
```
rf_metrics = metrics.classification_report(y_test, rfGrid_predict)
print(rf_metrics)
```

```
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      2770
           1       0.98      0.92      0.95       931

    accuracy                           0.97      3701
   macro avg       0.97      0.95      0.96      3701
weighted avg       0.97      0.97      0.97      3701
```

- Confusion Matrix

In [150]:
```
rf_cm = metrics.confusion_matrix(y_test, rfGrid_predict)
plt.figure(figsize=(7,7))
sns.heatmap(rf_cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'F
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
accuracy = 'Accuracy Score: {0}'.format(rf_score)
plt.title("Confusion Matrix of Random Forest\n" + accuracy, size = 15)

plt.show()
```



Interpretation: Wow! The random forest classifier performed better than KNN with accuracy 0.973

and better precision and recall for both Classes as well. Class 0 : Precision = 0.97 , Recall = 0.99; Class 1 : Precision = 0.98 , Recall = 0.92. HOORAY! Now let us see how Neural Network performs.

Multilayer Neural Network

Specifying the model's architecture

In [154]:
```
NeuralN = Sequential()

NeuralN.add(Dense(62, activation='relu', input_dim=7)) # input_dim is the number
NeuralN.add(Dense(100, activation='relu')) # input_dim is the number of features
NeuralN.add(Dense(1, activation='sigmoid')) # Binary classification so 2 output r
# Multiclass classification use softmax activation

NeuralN.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 62)                496
_________________________________________________________________
dense_1 (Dense)              (None, 100)               6300
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 101
=================================================================
Total params: 6,897
Trainable params: 6,897
Non-trainable params: 0
_________________________________________________________________
```

Explanation : Using Keras's sequential API, I specified my model with one input layer with 3 inputs which are the features. Two hidden layers with 62 neurons and 100 neurons, using 'relu' activation function where the input value will be the output if it is positive, if it is negative, the output will be 0. And the output layer will have one neuron with 'sigmoid' activation function which is an activation function used for binary classification. The out will be a probability value between 0 and 1. I will round them to p >= 0.5 to 1 and p < 0.5 to 0 later on.

In [155]:
```
NeuralN.compile(loss='binary_crossentropy',
                optimizer='adam',
                metrics=['accuracy'])
```

Explanation : Here I used binary crossentropy as my loss function and Adaptive Moment estimation which is a robust version of stochastic gradient descent. The metrics I use is acccuracy, which means the model will try to improve the accuracy.

- Model Training

I will now train my model using 100 epochs and 128 batch size

In [158]:
```
%%time
NeuralN.fit(X_train_nom, y_train, epochs = 100, batch_size = 128) #TnE : batch_si
```

```
Epoch 74/100
68/68 [==============================] - 0s 1ms/step - loss: 0.0580 - accurac
y: 0.9818

Epoch 75/100
68/68 [==============================] - 0s 2ms/step - loss: 0.0581 - accurac
y: 0.9823
Epoch 76/100
68/68 [==============================] - 0s 1ms/step - loss: 0.0586 - accurac
y: 0.9817
Epoch 77/100
68/68 [==============================] - 0s 1ms/step - loss: 0.0602 - accurac
y: 0.9798
Epoch 78/100
68/68 [==============================] - 0s 1ms/step - loss: 0.0586 - accurac
y: 0.9818
Epoch 79/100
68/68 [==============================] - 0s 1ms/step - loss: 0.0598 - accurac
y: 0.9798
Epoch 80/100
68/68 [==============================] - 0s 1ms/step - loss: 0.0581 - accurac
```

- Predicting and comparing results

In [161]:
```
# Prediction
nn_predict = NeuralN.predict(X_test_nom)
# Convert the classification into 0 or 1 using .round()
nn_predict = nn_predict.round()
# Predicted class
nn_predict[:10]
```

Out[161]:
```
array([[0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]], dtype=float32)
```

In [162]:
```
# Actual class
y_test[:10]
```

Out[162]:
```
13359    0
7360     0
14838    1
14773    1
3761     0
8017     0
12872    0
5432     0
2908     0
2209     0
Name: class, dtype: int64
```

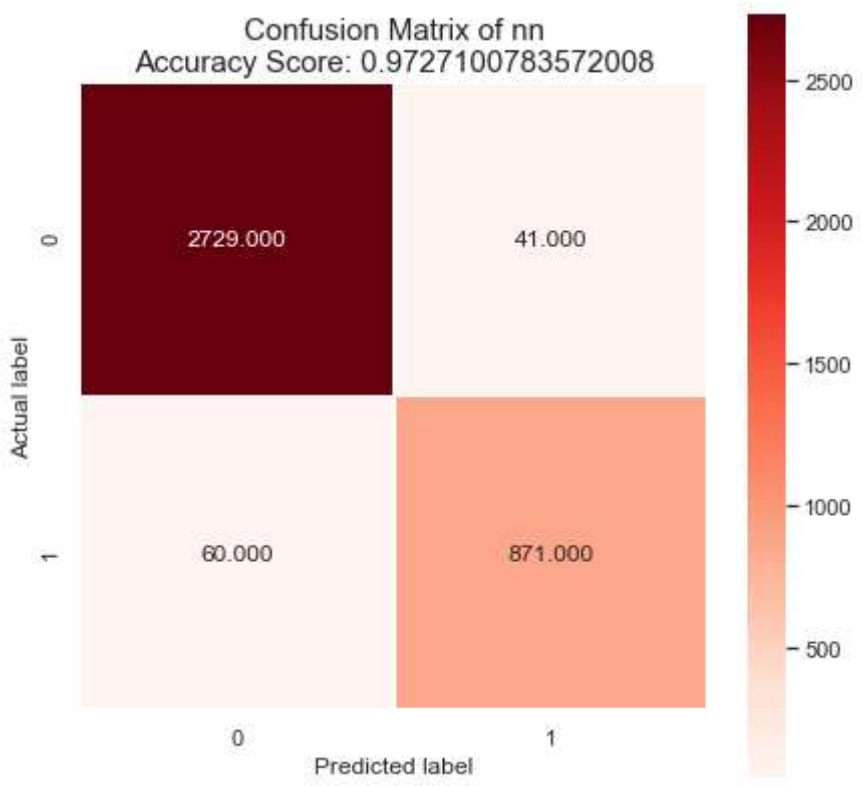
Shocked: The model predicted flawlessly for the first 10 instances!!

In [163]:
```
nn_score = accuracy_score(y_test, nn_predict)
nn_precision = metrics.precision_score(y_test,nn_predict)
nn_recall = metrics.recall_score(y_test,nn_predict)
```

In [165]:
```
nn_metrics = metrics.classification_report(y_test, nn_predict)
print(nn_metrics)
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      2770
           1       0.96      0.94      0.95       931

    accuracy                           0.97      3701
   macro avg       0.97      0.96      0.96      3701
weighted avg       0.97      0.97      0.97      3701
```

In [164]:
```
nn_cm = metrics.confusion_matrix(y_test, nn_predict)
plt.figure(figsize=(7,7))
sns.heatmap(nn_cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'F
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
accuracy = 'Accuracy Score: {0}'.format(nn_score)
plt.title("Confusion Matrix of nn\n" + accuracy, size = 15)
plt.show()
```



Interpretation: The performance of Neural Network and Random Forest are completely the same except a slight difference of 0.001 in accuracy. However, Neural Network are much successful in predicting correctly for Class 1, which is the objective of this project.

predicting correctly for Class 1, which is the objective of this project.

Evaluation

In this stage I am going to evaluate and choose the best model by looking at the confusion matrix and precision and recall for Class 1.

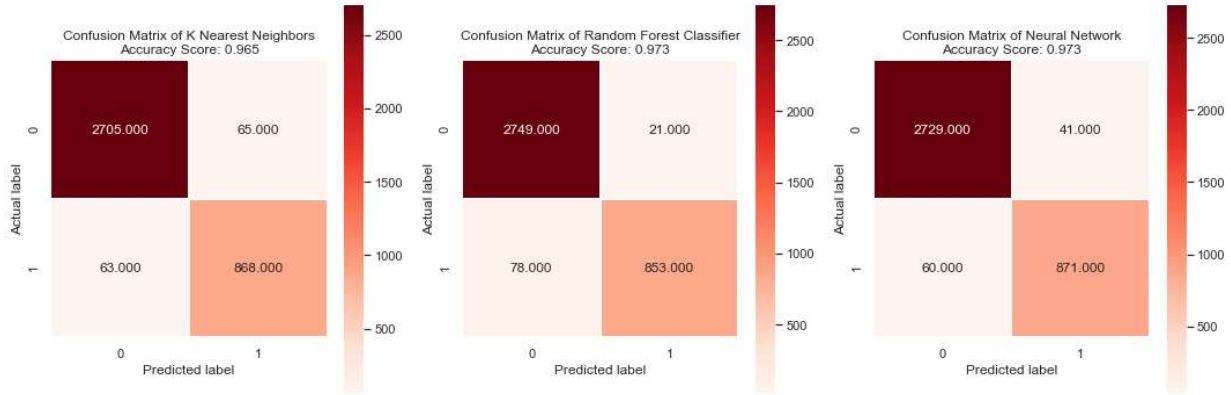
Accuracy, Precision, and Recall

In [166]:
```
apr = {'Metrics':  ['Accuracy', 'Precision', 'Recall'],
        'KNN': [knn_score_nom, knn_precision, knn_recall],
        'Random Forest': [rf_score, rf_precision, rf_recall],
        'Neural Network': [nn_score, nn_precision, nn_recall]
        }
apr_df = pd.DataFrame(apr)
apr_df
```

Out[166]:

	Metrics	KNN	Random Forest	Neural Network
0	Accuracy	0.965415	0.973250	0.972710
1	Precision	0.930332	0.975973	0.955044
2	Recall	0.932331	0.916219	0.935553

In [170]:
```
plt.figure(figsize=(15,5))

# KNN
plt.subplot(1,3,1)
knn_cm = metrics.confusion_matrix(y_test, knn_pred_nom)
sns.heatmap(knn_cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = '
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
accuracy = 'Accuracy Score: {0:.3f}'.format(knn_score_nom)
plt.title("Confusion Matrix of K Nearest Neighbors\n" + accuracy, size = 12)

# Random Forest
plt.subplot(1,3,2)
rf_cm = metrics.confusion_matrix(y_test, rfGrid_predict)
sns.heatmap(rf_cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'R
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
accuracy = 'Accuracy Score: {0:.3f}'.format(rf_score)
plt.title("Confusion Matrix of Random Forest Classifier\n" + accuracy, size = 12)

# Neural network
plt.subplot(1,3,3)
lr_cm = metrics.confusion_matrix(y_test, nn_predict)
sns.heatmap(nn_cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap = 'R
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
accuracy = 'Accuracy Score: {0:.3f}'.format(nn_score)
plt.title("Confusion Matrix of Neural Network\n" + accuracy, size = 12)

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



Final Model Decision :

For the final model I will choose between Random Forest Classifier and Neural Network since KNN only have 0.965 or 96.5% accuracy. The accuracy for Neural Network and Random Forest are the same which is 97.3%. The F1-Score for Random Forest is (0.975973 + 0.916219)/2 = 0.946096. The F1-Score for Neural Network is (0.955044 + 0.935553)/2 = 0.9452985. Both of these values are pretty much equal. However, the Random Forest has a slightly higher F1-score therefore my final model will be **Random Forest Classifier**.

Deployment

This is the last phase of this project. In this phase I will test my deploy my best model for actual classification using user inputs and test the usefulness of my model.

In [195]:
```
final_model = rf_Grid
```

In [196]:
```
def predict (satisfaction_level,last_evaluation,average_montly_hours,time_spend_c
    # Pre-processing : Replace null inputs from user with 0
    if(satisfaction_level==''):
        satisfaction_level=0
    if(last_evaluation==''):
        last_evaluation=0
    if(average_montly_hours==''):
        average_montly_hours=0
    if(time_spend_company==''):
        time_spend_company=0
    if(Work_accident==''):
        Work_accident=0
    if(promotion_last_5years==''):
        promotion_last_5years=0
    if(salary==''):
        salary=2 # Medium
    prediction = final_model.predict([[satisfaction_level,last_evaluation,average
    prediction = prediction.round()
    if prediction == 1:
        result = 'Employee will likely leave.'
    else :
        result = 'Employee will likely stay.'
    return result
```

In [201]:
```
def main():
    input_1 = input ("Enter the value of satisfaction_level: ")
    input_2 = input ("Enter the value of last_evaluation : ")
    input_3 = input ("Enter the value of average_montly_hours : ")
    input_4 = input ("Enter the value of time_spend_company : ")
    input_5 = input ("Enter the value of Work_accident : ")
    input_6 = input ("Enter the value of promotion_last_5years : ")
    input_7 = input ("Enter the value of salary(1 = Low, 2 = Medium, 3 = High) :

    result = predict(input_1,input_2,input_3,input_4,input_5,input_6,input_7)
    print("The predicted Class is : ", result)
    return
main()
```

```
Enter the value of satisfaction_level: 0.8
Enter the value of last_evaluation : 0.86
Enter the value of average_montly_hours : 262
Enter the value of time_spend_company : 6
Enter the value of Work_accident : 0
Enter the value of promotion_last_5years : 0
Enter the value of salary(1 = Low, 2 = Medium, 3 = High) : 2
The predicted Class is :  Employee will likely leave.
```

In [199]:
```
y_pred_test = final_model.predict(X_test_nom)
score = accuracy_score(y_pred_test, y_test)
print(score)
```

```
0.973250472845177
```

Conclusion: The model has an accuracy 0f 97.3% !!!