Python

Certification Project

Submitted by
Arjun Kumar Reddy Geeda

edureka!

Industry: Human Resource management

Challenge –

AB Consultants is a company that outsources its employees as Consultants to top various IT firms. They have been in the Industry for a long time. Their business had been increasing quite well over past, however in recent times there has been a slowdown in terms of growth because their best and most experienced employees have started leaving the Company. The VP of the firm is not very happy with the company's best and most experienced employees leaving prematurely. The VP of the firm has employed you to find out insights in the Company's Employee Data and find out an answer as to know why the best and most experienced employees are leaving.

Solution –
The VP has laid out the requirements to you. As a Data Analyst of the Company now it's your charter to do the required analysis and find out patterns as to why the best employees are leaving so early.
Using Python, you derive at a forecast model to predict which employees could be leaving the company, as well as a probability as to why our best and most experienced employees are leaving prematurely. This will help to plan our next steps to avoid the churn out. You decide to create a script that will contain the following:

• A visualization and distribution (of all the employee relative fields)

• Forecast using different Machine Learning models and see the outcome

• Compare different Machine Learning models and cross validate them

• Find out why best and most experienced employees are leaving prematurely

• Give a Final Prediction Model (the best one) to Forecast

Contents

1. Visualization and Distribution of All the Employee Related Fields

1.1. Distribution of Dataset

To create Visualizations and distributions we need to import some libraries like Numpy, Pandas, Seaborn, Matplotlib. Numpy library is used to read the file which contains the data. Pandas are used to apply numerical computations on the data. Seaborn and Matplotlib are used to create the visualizations for the data.

```
import numpy as np #to read the file
import pandas as pd #for numerical computations
import seaborn as sns
import matplotlib.pyplot as plt
```

1.1.1. Read the Dataset

To read the dataset we use the panda library. We use **pd.read_csv** command to import the data.

```
dataset = pd.read_csv('G:\\python\\python project\\HR_Data.csv')
```

We get the shape of the dataset by the **dataset.shape** command.

```
In [3]: #shape of Dataset
        dataset.shape

Out[3]: (14999, 10)
```

We can print the first 5 rows of the dataset by using **dataset.head()** command.

```
In [4]: #prints first 5 rows
        dataset.head()

Out[4]:    satisfaction_level  last_evaluation  number_project  average_montly_hours  \
        0                0.38             0.53               2                   157
        1                0.80             0.86               5                   262
        2                0.11             0.88               7                   272
        3                0.72             0.87               5                   223
        4                0.37             0.52               2                   159

           time_spend_company  Work_accident  left  promotion_last_5years department  \
        0                   3              0     1                      0      sales
        1                   6              0     1                      0      sales
        2                   4              0     1                      0      sales
        3                   5              0     1                      0      sales
        4                   3              0     1                      0      sales

           salary
        0     low
        1  medium
        2  medium
        3     low
        4     low
```

1.1.2. Renaming the Variables in the Dataset.

We can rename the variables in the dataset by using **dataset.rename()**.

```
In [5]: #Renaming of dataset
        dataset=dataset.rename(columns={'department':'Department'})
        dataset=dataset.rename(columns={'average_montly_hours':'Average_Monthly_Hours'})
        dataset=dataset.rename(columns={'satisfaction_level':'Satisfaction_Level'})
        dataset=dataset.rename(columns={'last_evaluation':'Last_Evaluation'})
        dataset=dataset.rename(columns={'time_spend_company':'Time_Spent_in_Company'})
        dataset=dataset.rename(columns={'Work_accident':'Work_Accident'})
        dataset=dataset.rename(columns={'left':'Left'})
        dataset=dataset.rename(columns={'promotion_last_5years':'Promotion_Last_5Years'})
        dataset=dataset.rename(columns={'salary':'Salary'})
        dataset=dataset.rename(columns={'number_project':'Number_Project'})
```

1.1.3. Describing the dataset

We can get the mean, median, standard deviation, minimum value, maximum value and Quartiles. We can get the summary of the dataset by using **dataset.describe()**.

```
In [6]: #gives Summary of the Data
        dataset.describe()
```

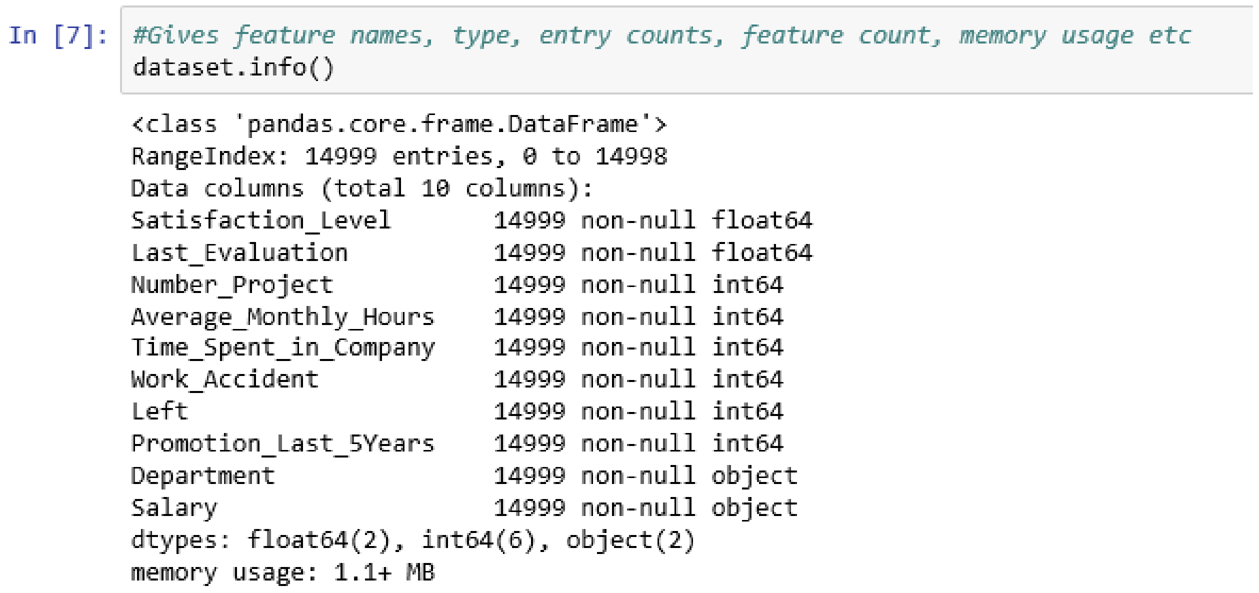
```
Out[6]:        Satisfaction_Level  Last_Evaluation  Number_Project  \
        count        14999.000000     14999.000000    14999.000000
        mean             0.612834         0.716102        3.803054
        std              0.248631         0.171169        1.232592
        min              0.090000         0.360000        2.000000
        25%              0.440000         0.560000        3.000000
        50%              0.640000         0.720000        4.000000
        75%              0.820000         0.870000        5.000000
        max              1.000000         1.000000        7.000000

               Average_Monthly_Hours  Time_Spent_in_Company  Work_Accident  \
        count           14999.000000           14999.000000   14999.000000
        mean              201.050337               3.498233       0.144610
        std                49.943099               1.460136       0.351719
        min                96.000000               2.000000       0.000000
        25%               156.000000               3.000000       0.000000
        50%               200.000000               3.000000       0.000000
        75%               245.000000               4.000000       0.000000
        max               310.000000              10.000000       1.000000

                 Left  Promotion_Last_5Years
  count  14999.000000           14999.000000
  mean       0.238083               0.021268
  std        0.425924               0.144281
  min        0.000000               0.000000
  25%        0.000000               0.000000
  50%        0.000000               0.000000
  75%        0.000000               0.000000
  max        1.000000               1.000000
```

1.1.4. Getting the information of the dataset.

We can get the feature names type of variables, number of values for each variable, memory usage of each variable by using **dataset.info()** command.

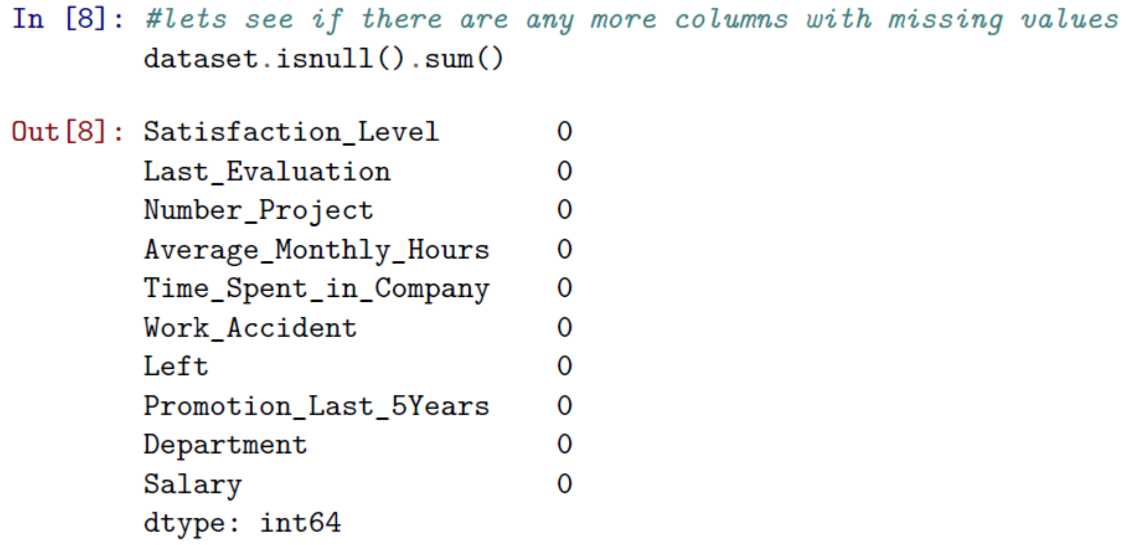


```
In [7]: #Gives feature names, type, entry counts, feature count, memory usage etc
        dataset.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 14999 entries, 0 to 14998
        Data columns (total 10 columns):
        Satisfaction_Level       14999 non-null float64
        Last_Evaluation          14999 non-null float64
        Number_Project           14999 non-null int64
        Average_Monthly_Hours    14999 non-null int64
        Time_Spent_in_Company    14999 non-null int64
        Work_Accident            14999 non-null int64
        Left                     14999 non-null int64
        Promotion_Last_5Years    14999 non-null int64
        Department               14999 non-null object
        Salary                   14999 non-null object
        dtypes: float64(2), int64(6), object(2)
        memory usage: 1.1+ MB
```

1.1.5. Checking Null Values

We need to check if there are null values because it provides inappropriate results in calculation and visualization of dataset.



```
In [8]: #lets see if there are any more columns with missing values
        dataset.isnull().sum()

Out[8]: Satisfaction_Level       0
        Last_Evaluation          0
        Number_Project           0
        Average_Monthly_Hours    0
        Time_Spent_in_Company    0
        Work_Accident            0
        Left                     0
        Promotion_Last_5Years    0
        Department               0
        Salary                   0
        dtype: int64
```

1.2. Visualization of Dataset

1.2.1. Correlation Plots

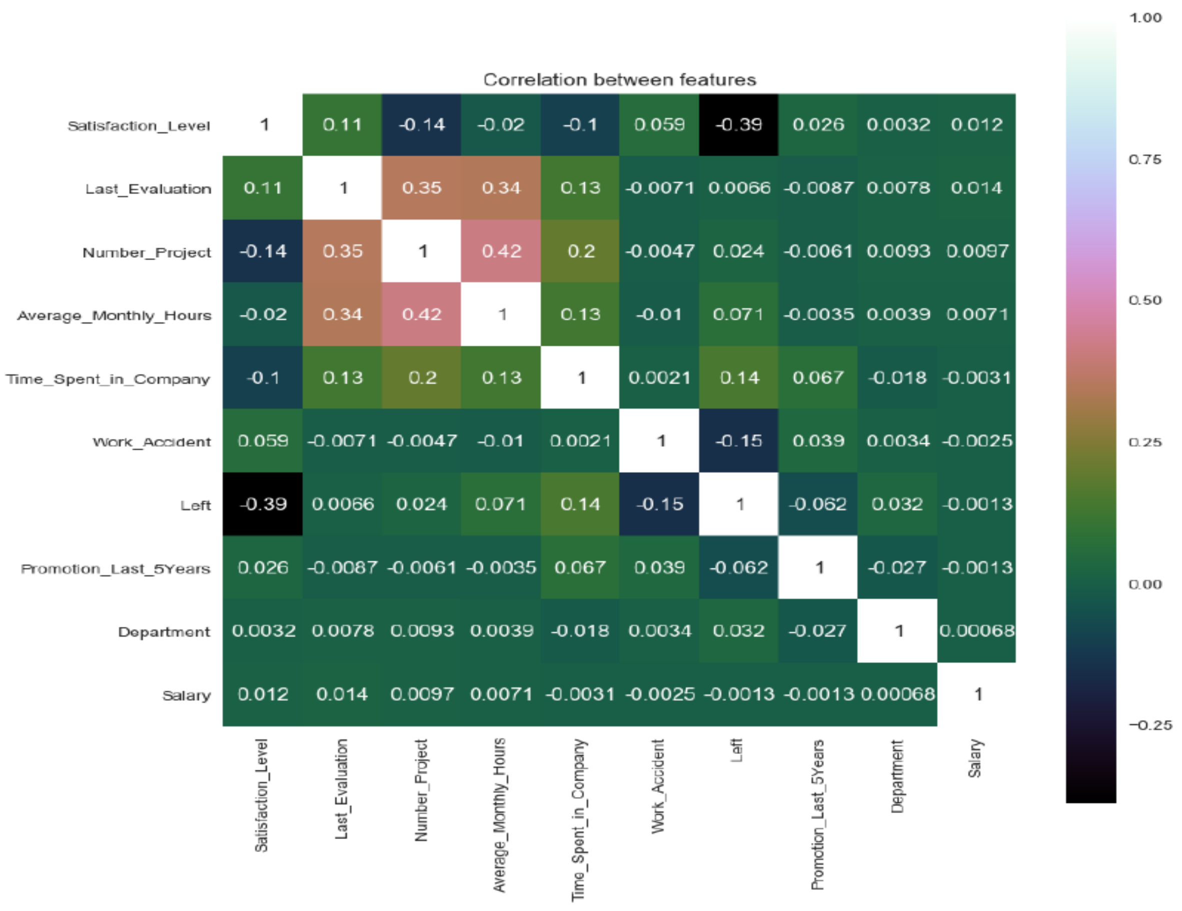
Definition of correlation:
Correlation is a statistical measure that indicates the extent to which two or more variables fluctuate together. A positive correlation indicates the extent to which those variables increase or decrease in parallel; a negative correlation indicates the extent to which one variable increases as the other decreases.

Correlation values for the given data can be found out by using command **dataset.corr().** We can plot the correlation using **sns.heatmap()** command. For using **sns.heatmap()** we need the library of '**seaborn**'.

Commands for correlation plot:

```
sns.set(font_scale=1)
corr=dataset.corr()
plt.figure(figsize=(10, 10))
sns.heatmap(corr, vmax=1, square=True,annot=True,cmap='cubehelix')
plt.title('Correlation between features')
```

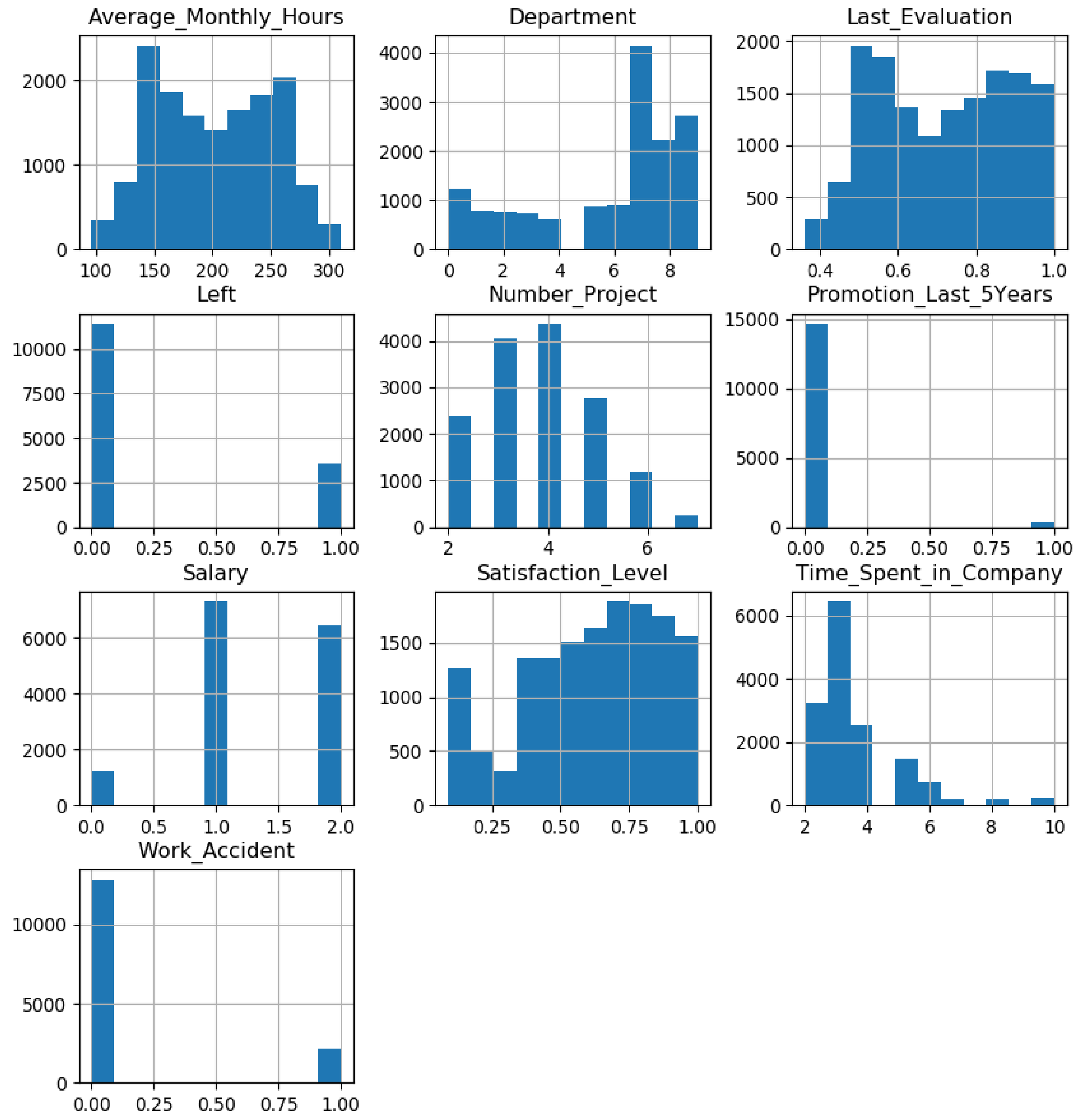
Correlation plot:



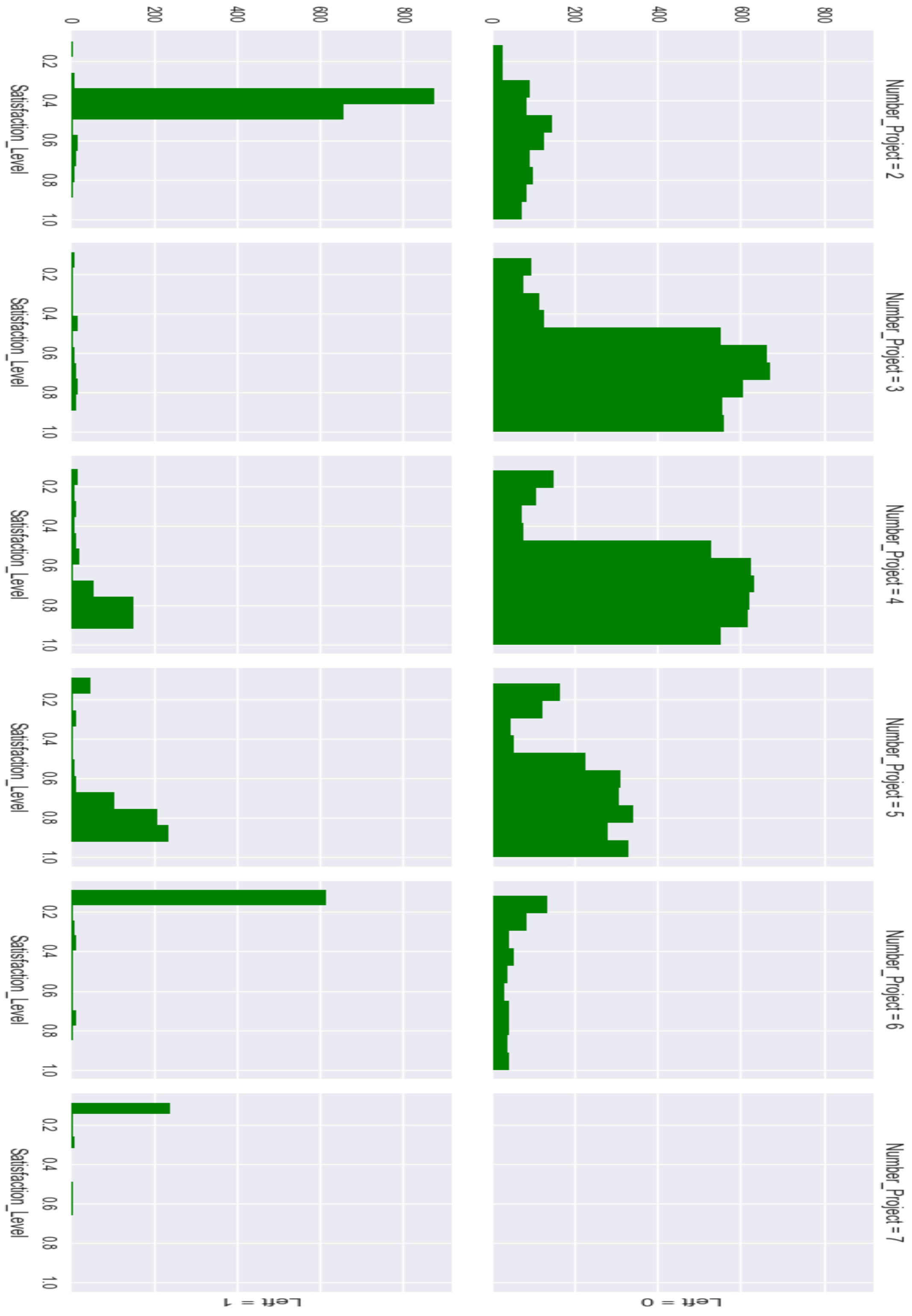
1.2.2. Data Visualization for all the data

```
plt.style.use(style = 'default')
dataset.hist(bins=11,figsize=(10,10),grid=True)
plt.show()
```

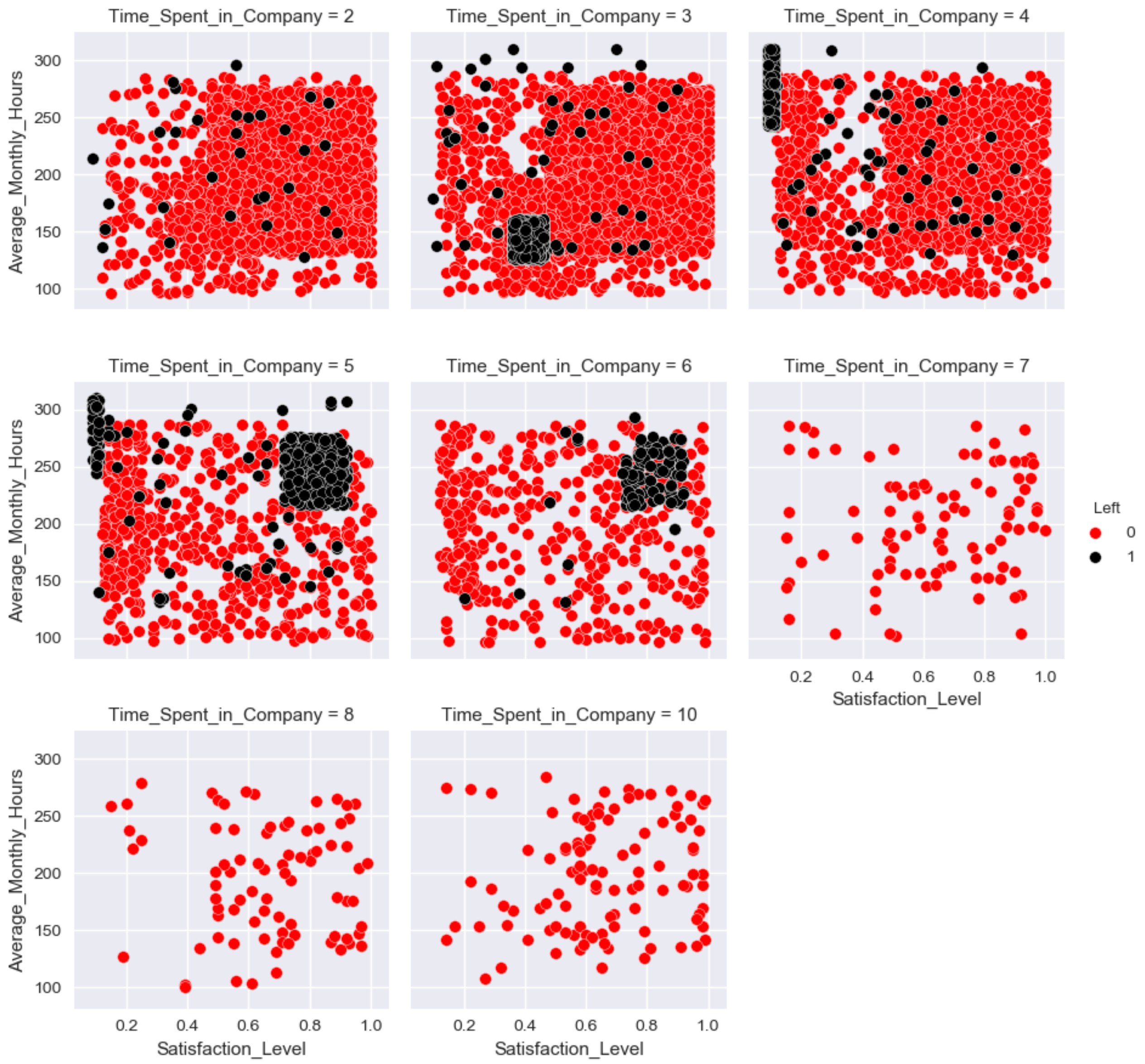
```
sns.set(font_scale=1)
g = sns.FacetGrid(dataset, col="Number_Project", row="Left", margin_titles=True)
g.map(plt.hist, "Satisfaction_Level",color="green")
```



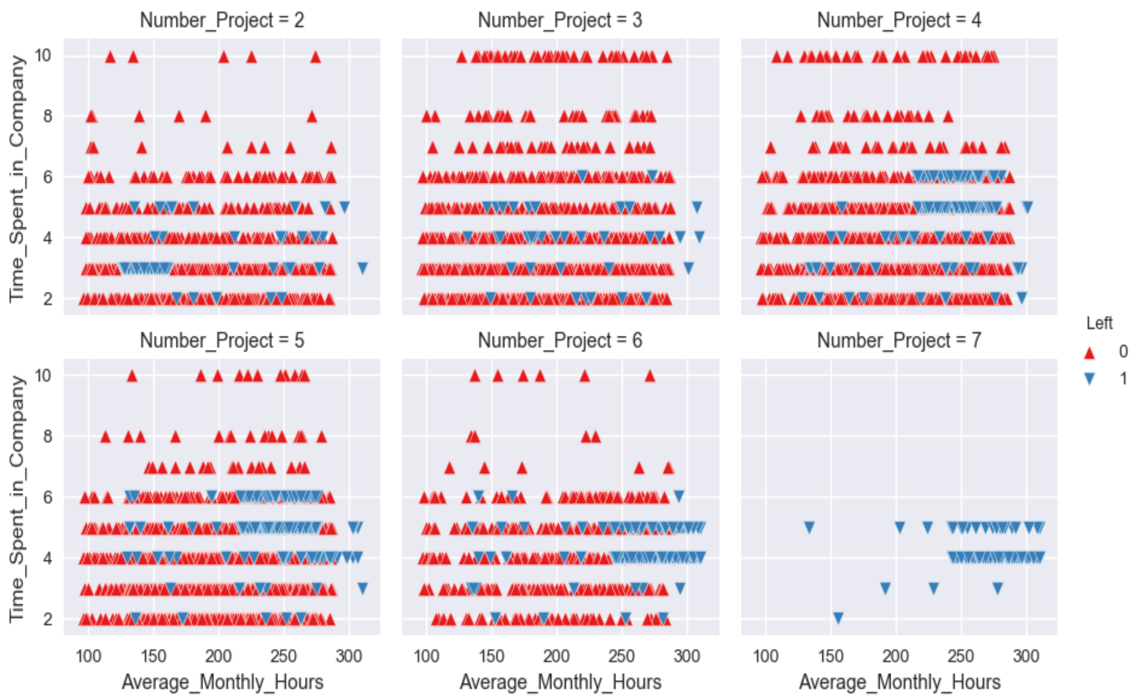
```
sns.set(font_scale=1)
g = sns.FacetGrid(dataset, col="Number_Project", row="Left", margin_titles=True)
g.map(plt.hist, "Satisfaction_Level",color="green")
```



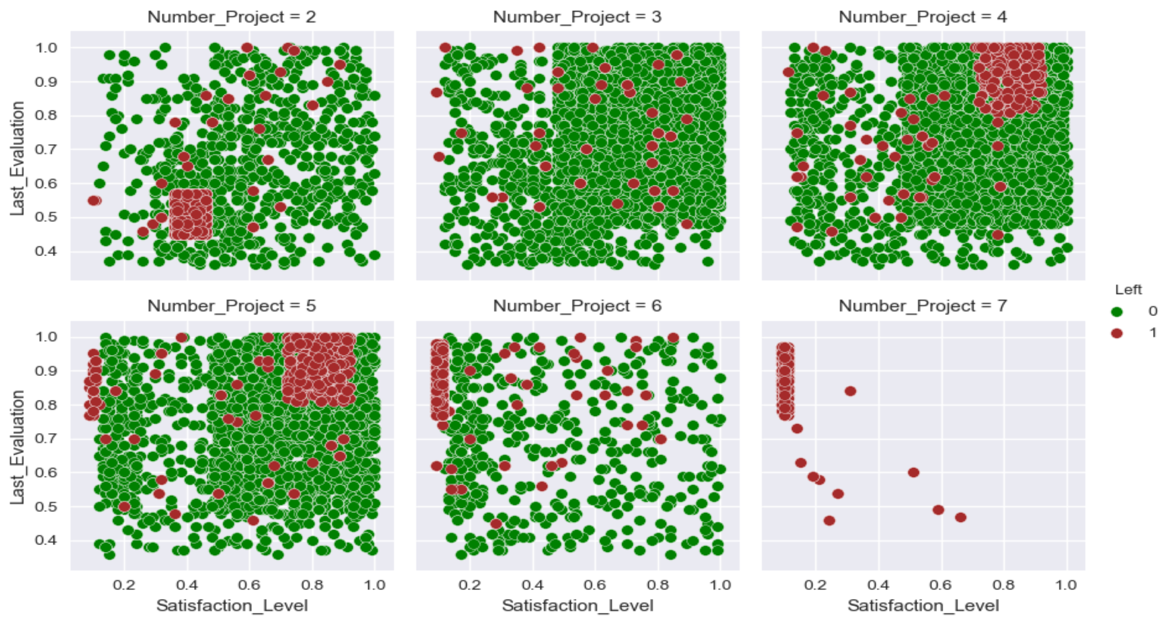
```
g = sns.FacetGrid(dataset, hue="Left", col="Time_Spent_in_Company",
                  margin_titles=True,
                  palette={1:"black", 0:"red"}, col_wrap=2)
g=g.map(plt.scatter, "Satisfaction_Level", "Average_Monthly_Hours",
        edgecolor="w").add_legend()
```



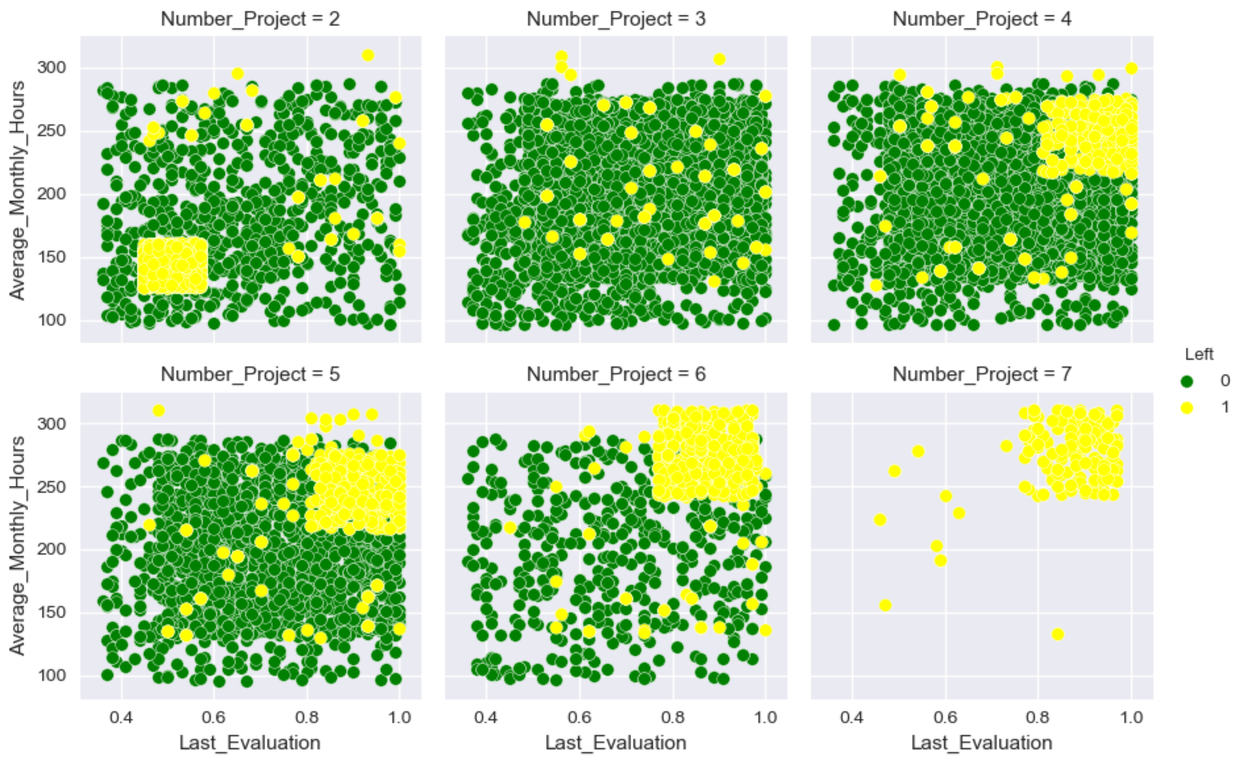
```
g = sns.FacetGrid(dataset, hue="Left", col="Number_Project", margin_titles=True,
                  palette="Set1", hue_kws=dict(marker=["^", "v"]), col_wrap=3)
g.map(plt.scatter, "Average_Monthly_Hours", "Time_Spent_in_Company",
      edgecolor="w").add_legend()
plt.subplots_adjust(top=0.8)
```



```
g = sns.FacetGrid(dataset, hue="Left", col="Number_Project", margin_titles=True,
                  palette={1:"brown", 0:"green"}, col_wrap=3)
g=g.map(plt.scatter, "Satisfaction_Level", "Last_Evaluation",
        edgecolor="w").add_legend()
```



```
g = sns.FacetGrid(dataset, hue="Left", col="Number_Project",
                  margin_titles=True,
                  palette={1:"yellow", 0:"green"}, col_wrap=3)
g=g.map(plt.scatter, "Last_Evaluation", "Average_Monthly_Hours",
        edgecolor="w").add_legend()
```



2. Forecast Using Different Machine Learning Models

Machine learning is defined as "A computer program which is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P* if its performance at tasks in *T*, as measured by *P*, improves with experience *E*.
Machine learning is categorized into two types.

1. Classification
2. Clustering

<u>Classification:</u> It is one of the type of supervised learning problem. It is the problem of identifying to which set of categories a new observation belongs. The goal of classification is to find boundaries that best separate different categories of data. These decision boundaries then allow you to differentiate between classes of data and classify any new value.
<u>Example:</u> Assigning a given email “spam” or “non-spam” classes.

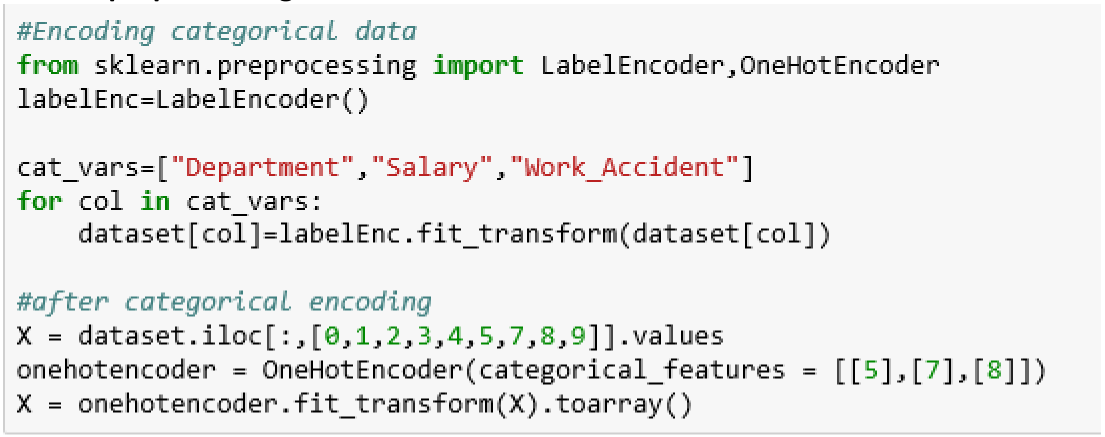
<u>Clustering:</u> It is one of the type of unsupervised learning problem. It means groping of objects based on the information found in the data, describing the objects or their relationship. The goal is that objects in one group will be similar to one other and different from objects in another group. It deals with finding a structure in a collection of unlabeled data. The goal of clustering is to determine the intrinsic grouping in a set of unlabeled data. Organizing data into clusters shows internal structure of the data.
<u>Example:</u> Finding groups of customers with similar behavior.

Our problem comes under Classification problem. The classification problem is categorized into six types

1. Decision Tree Modelling
2. Random Forest
3. Naïve Bayes
4. Support vector Machines
5. Logistic Regression
6. KNN Algorithm

2.1. Building a classification model

To implement a machine learning routine first we need to clean the data of Categorical features by converting them into factors using **onehotencoder**. In the given data categorical features are department, work accident and salary. For using that we need to import that from library of **sklearn.preprocessing.**

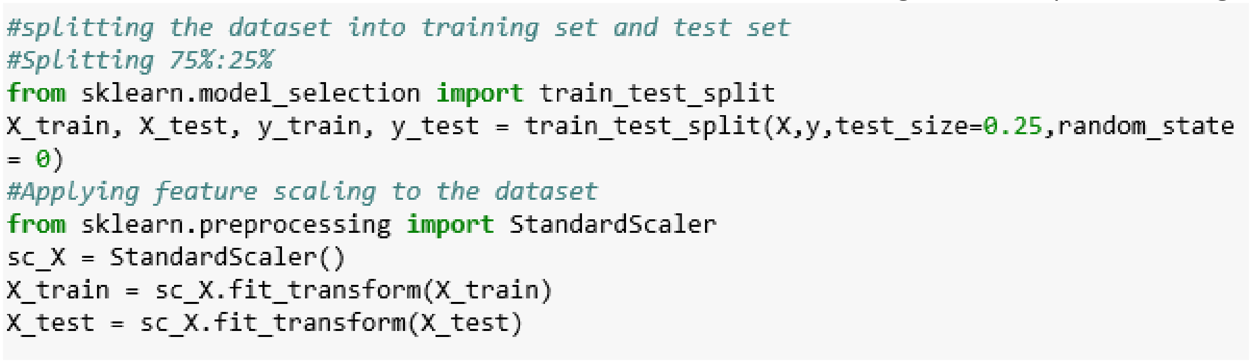


```
#Encoding categorical data
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
labelEnc=LabelEncoder()

cat_vars=["Department","Salary","Work_Accident"]
for col in cat_vars:
    dataset[col]=labelEnc.fit_transform(dataset[col])

#after categorical encoding
X = dataset.iloc[:,[0,1,2,3,4,5,7,8,9]].values
onehotencoder = OneHotEncoder(categorical_features = [[5],[7],[8]])
X = onehotencoder.fit_transform(X).toarray()
```

The next step is dividing the data for training and testing. The data is split using the **train_test_split** command. This command should be imported from **sklearn.model_selection** library. This command splits the data depending on the ratio we give. The data is split based on the vector of data label i.e. left. The 75% of data is divided for training and other part is testing.



```
#splitting the dataset into training set and test set
#Splitting 75%:25%
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.25,random_state
= 0)
#Applying feature scaling to the dataset
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.fit_transform(X_test)
```

For finding the most accurate model we need few terms to decide which gives better result.
A few important terms:
Confusion matrix: In a confusion matrix each column of the matrix represents the instances in a predicted class while each row represents the instances in an actual class (or vice versa). It takes predicted values and the factor against which the model is tested as the arguments.
Accuracy Score: Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. Therefore, you have to look at other parameters to evaluate the performance of your model.

Accuracy score= $\frac{TP+TN}{TP+FP+FN+TN}$

True Positives (TP) - These are the correctly predicted positive values which means that the value of actual class is yes. and the value of predicted class is also yes. E.g. if actual class value indicates that this passenger survived and predicted class tells you the same thing.
True Negatives (TN) - These are the correctly predicted negative values which means that the value of actual class is no. and value of predicted class is also no. E.g. if actual class says this passenger did not survive and predicted class tells you the same thing.
False Positives (FP) – When actual class is no and predicted class is yes. E.g. if actual class says this passenger did not survive but predicted class tells you that this passenger will survive.
False Negatives (FN) – When actual class is yes but predicted class in no. E.g. if actual class value indicates that this passenger survived and predicted class tells you that passenger will die.
False positives and false negatives, these values occur when your actual class contradicts with the predicted class.

2.1.1. Decision Tree

Decision tree learning uses a decision tree (as a predictive model) to go from observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modelling approaches used in statistics, data mining and machine_learning. Tree models where the target variable can take a discrete set of values are called **classification trees**; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels.
To use Decision tree, we need to load the library of **sklearn.tree** and from that library import **DecisionTreeClassifier.**

2.1.1.1. Syntax for Decision Tree Model

```
#Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier as dt
classifier = dt(criterion='entropy',random_state=0)
classifier.fit(X_train,y_train)
#Predicting the results
y_pred=classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
print"Accuracy Score and Confusion Matrix for Decision Tree Classifier"
print"Accuracy Score for Decision Tree Classifier:",accuracy_score(y_test,y_pred)
print"Confusion Matrix for Decision Tree Classifier\n",cm
```

The library of ‘DecisionTreeClassifier’ loads the machine learning algorithm in to the environment. Accuracy score requires from y_test and y_pred values. Confusion matrix requires from y_test and y_pred values.

2.1.1.2. Output for Decision Tree Classification

```
Accuracy Score and Confusion Matrix for Decision Tree Classifier
Accuracy Score for Decision Tree Classifier: 0.9584
Confusion Matrix for Decision Tree Classifier
[[2837   44]
 [ 112  757]]
```

2.1.2. Random Forest

A random forest is a meta estimator that fits number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size, but the samples are drawn with replacement if *bootstrap=True* (default). To use Random Forest, we need to load the library of **sklearn.ensemble** import **RandomForestClassifier**.

2.1.2.1. Syntax for Random Forest classification

```
#random forest classification
from sklearn.ensemble import RandomForestClassifier as rfc
#for 5 Trees in forest
classifier=rfc(n_estimators=10,criterion='entropy',random_state=0)
classifier.fit(X_train,y_train)
#Predicting the results
y_pred=classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
print"Accuracy Score and Confusion Matrix for Random Forest Classifier"
print"Accuracy Score for Random Forest Classifier:",accuracy_score(y_test,y_pred)
print"Confusion Matrix for Random Forest Classifier\n",cm
```
```

The library of "**RandomForestClassifier**" loads the machine learning algorithm in to the environment. Accuracy score requires from y_test and y_pred values. Confusion matrix requires from y_test and y_pred values.

#### 2.1.2.2. Output for Random Forest Classification

```
Accuracy Score and Confusion Matrix for Random Forest Classifier
Accuracy Score for Random Forest Classifier: 0.9586666666666667
Confusion Matrix for Random Forest Classifier
[[2873   8]
 [147 722]]
```
```

2.1.3. Naive Bayes

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one-dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality. To use Naïve Bayes, we need to load the library of **sklearn.naivebayes** import **GaussianNB**.

2.1.3.1. Syntax for Naïve Bayes classification

```
#Naive bayes
from sklearn.naive_bayes import GaussianNB
classifier=GaussianNB()
classifier.fit(X_train,y_train)
#Predicting the test results
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
print"Accuracy Score and Confusion Matrix for Naive Bayes"
print"Accuracy Score for Naive Bayes:",accuracy_score(y_test,y_pred)
print"Confusion Matrix for Naive Bayes\n",cm
```

The library of "**GaussianNB**" loads the machine learning algorithm in to the environment. Accuracy score requires from y_test and y_pred values. Confusion matrix requires from y_test and y_pred values.

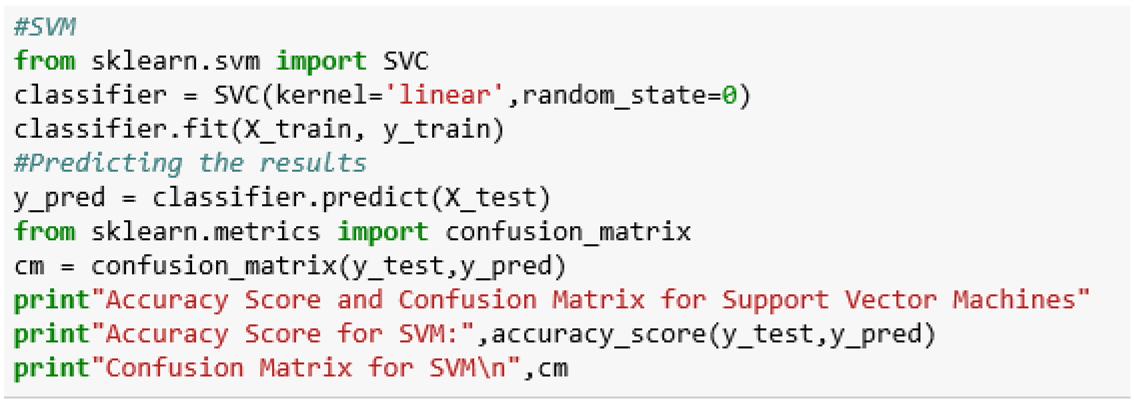
2.1.3.1. Output for Naïve Bayes classification

```
Accuracy Score and Confusion Matrix for Naive Bayes
Accuracy Score for Naive Bayes: 0.8037333333333333
Confusion Matrix for Naive Bayes
[[2360  521]
 [ 215  654]]
```

2.1.4. Support Vector Machines

Support Vector Machines (SVM) is a data classification method that separates data using hyperplanes. The concept of SVM is very intuitive and easily understandable. If we have labeled data, SVM can be used to generate multiple separating hyperplanes such that the data space is divided into segments and each segment contains only one kind of data. SVM technique is generally useful for data which has non-regularity which means, data whose distribution is unknown. To use SVM, we need to load the library of **sklearn.svm** import **SVC**.

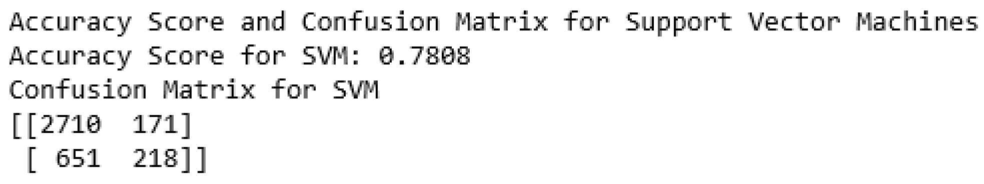
2.1.4.1. Syntax for Support Vector Machine



```
#SVM
from sklearn.svm import SVC
classifier = SVC(kernel='linear',random_state=0)
classifier.fit(X_train, y_train)
#Predicting the results
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
print"Accuracy Score and Confusion Matrix for Support Vector Machines"
print"Accuracy Score for SVM:",accuracy_score(y_test,y_pred)
print"Confusion Matrix for SVM\n",cm
```

The library of "**SVC**" loads the machine learning algorithm in to the environment. Accuracy score requires from y_test and y_pred values. Confusion matrix requires from y_test and y_pred values.

2.1.4.2. Output for Support Vector Machines model



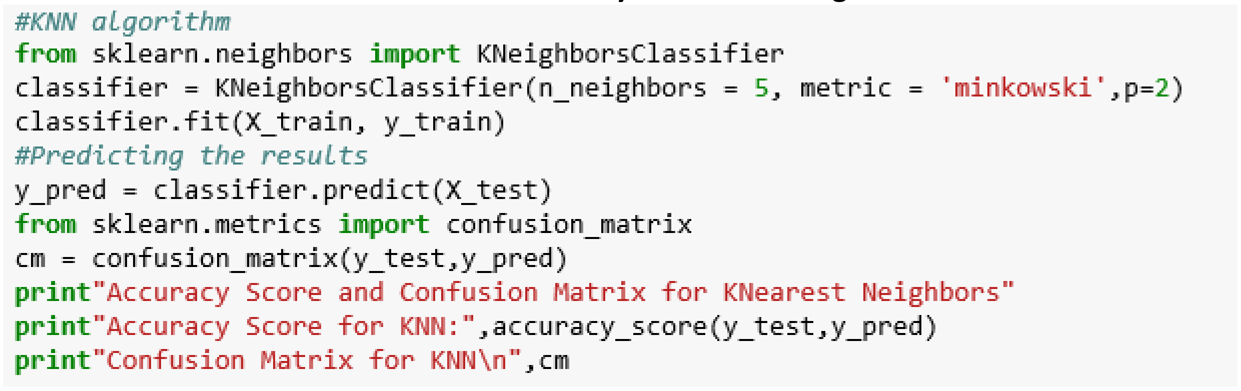
```
Accuracy Score and Confusion Matrix for Support Vector Machines
Accuracy Score for SVM: 0.7808
Confusion Matrix for SVM
[[2710  171]
 [ 651  218]]
```

2.1.5. KNN algorithm

KNN is a non-parametric lazy learning algorithm. That is a pretty concise statement. When you say a technique is non-parametric, it means that it does not make any assumptions on the underlying data distribution. This is pretty useful, as in the real world, most of the practical data does not obey the typical theoretical assumptions made (eg gaussian mixtures, linearly separable etc). Non-parametric algorithms like KNN come to the rescue here.

It is also a lazy algorithm. What this means is that it does not use the training data points to do any *generalization*. In other words, there is no explicit training phase or it is very minimal. This means the training phase is pretty fast. Lack of generalization means that KNN keeps all the training data. More exactly, all the training data is needed during the testing phase. (Well this is an exaggeration, but not far from truth). This is in contrast to other techniques like SVM where you can discard all non-support vectors without any problem. Most of the lazy algorithms – especially KNN – makes decision based on the entire training data set (in the best case a subset of them). To use **KNN algorithm**, we need to load the library of **sklearn.neighbors** import **KNeighborsClassifier**.

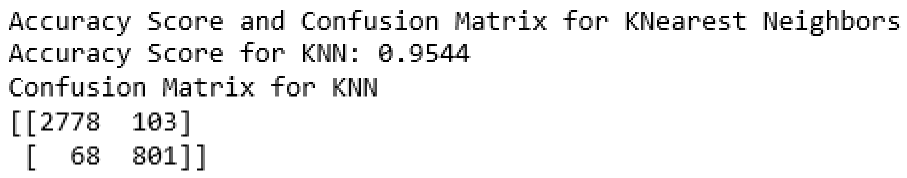
2.1.5.1. Syntax for KNN algorithm



```
#KNN algorithm
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski',p=2)
classifier.fit(X_train, y_train)
#Predicting the results
y_pred = classifier.predict(X_test)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
print"Accuracy Score and Confusion Matrix for KNearest Neighbors"
print"Accuracy Score for KNN:",accuracy_score(y_test,y_pred)
print"Confusion Matrix for KNN\n",cm
```

The library of "**KNeighborsClassifier**" loads the machine learning algorithm in to the environment. Accuracy score requires from y_test and y_pred values. Confusion matrix requires from y_test and y_pred values.

2.1.5.2. Output for KNN algorithm model



```
Accuracy Score and Confusion Matrix for KNearest Neighbors
Accuracy Score for KNN: 0.9544
Confusion Matrix for KNN
[[2778  103]
 [  68  801]]
```

2.1.6. Logistic regression

Logistic regression is a statistical method for analyzing a dataset in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes).
In logistic regression, the dependent variable is binary or dichotomous, i.e. it only contains data coded as 1 (TRUE, success, pregnant, etc.) or 0 (FALSE, failure, non-pregnant, etc.).
The goal of logistic regression is to find the best fitting (yet biologically reasonable) model to describe the relationship between the dichotomous characteristic of interest and a set of independent variables. Logistic regression generates the coefficients (and its standard errors and significance levels) of a formula to predict a *logit transformation* of the probability of presence of the characteristic of interest:
$logit(p) = b_0 + b_1X_1 + b_2X_2 + b_3X_3 + \ldots + b_kX_k$ where p is the probability of presence of the characteristic of interest. The logit transformation is defined as the logged odds:

$$odds = \frac{p}{1-p} = \frac{probability\ of\ presence\ of\ characteristic}{probability\ of\ absence\ of\ characteristic}$$

and

$$logit(p) = \ln\left(\frac{p}{1-p}\right)$$

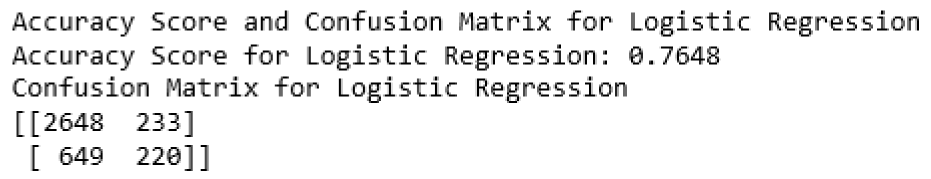
To use **KNN algorithm**, we need to load the library of **sklearn.linearmodel** import **LogisticRegression**.

2.1.6.1. Syntax for Logistic Regression model

```
#logistic regression model
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train,y_train)
#Predicting the test result
y_pred = classifier.predict(X_test)
#making confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
print"Accuracy Score and Confusion Matrix for Logistic Regression"
print"Accuracy Score for Logistic Regression:",accuracy_score(y_test, y_pred)
print"Confusion Matrix for Logistic Regression\n",cm
```

The library of "**LogisticRegression**" loads the machine learning algorithm in to the environment. Accuracy score requires from y_test and y_pred values. Confusion matrix requires from y_test and y_pred values.

2.1.6.1. Output for Logistic Regression model



```
Accuracy Score and Confusion Matrix for Logistic Regression
Accuracy Score for Logistic Regression: 0.7648
Confusion Matrix for Logistic Regression
[[2648  233]
 [ 649  220]]
```

3. Comparing Different Machine Learning Models and Cross Validating them

Cross-validation is a technique to evaluate predictive models by partitioning the original sample into a training set to train the model, and a test set to evaluate it. In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k-1 subsamples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged to produce a single estimation. The advantage of this method is that all observations are used for both training and validation, and each observation is used for validation exactly once.

For classification problems, one typically uses stratified k-fold cross-validation, in which the folds are selected so that each fold contains roughly the same proportions of class labels. In repeated cross-validation, the cross-validation procedure is repeated n times, yielding n random partitions of the original sample. The n results are again averaged to produce a single estimation.

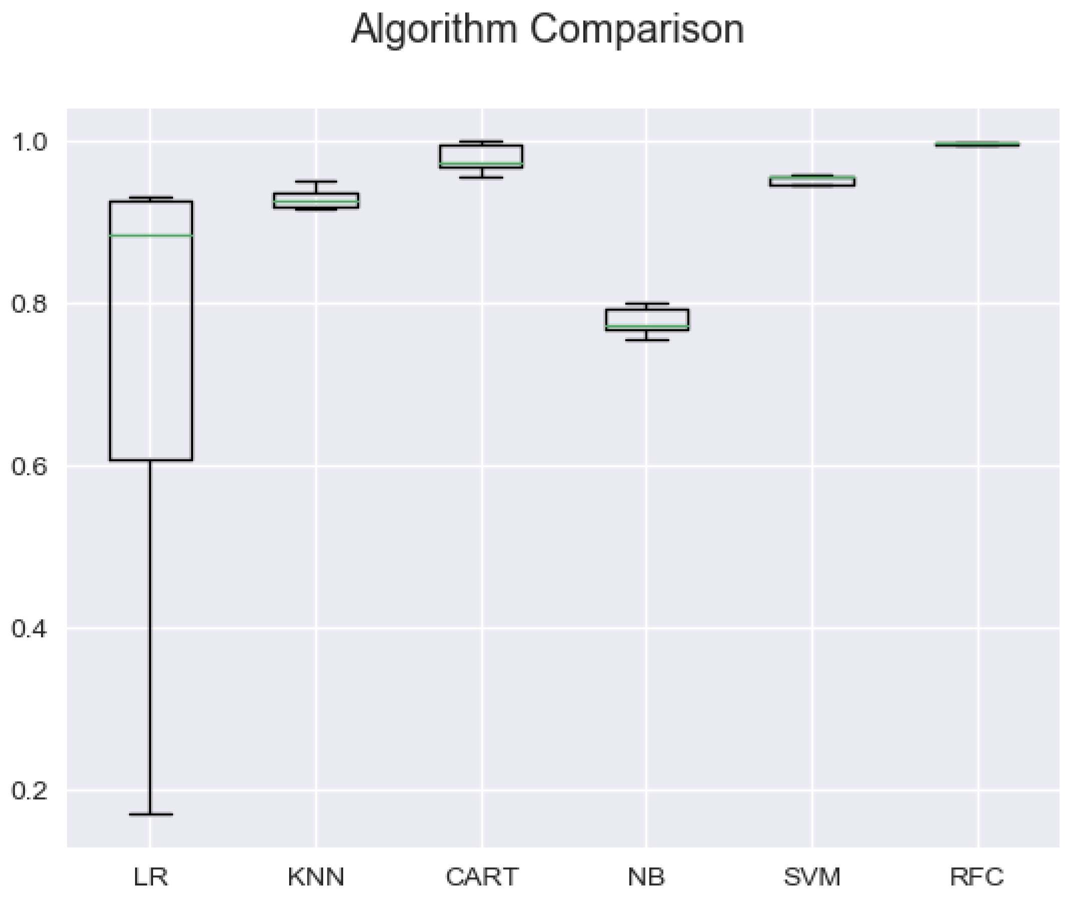
3.1. Syntax for Cross Validation

```python
#Comparing different Machine Learning models and cross validating them
print("#Comparing different Machine Learning models and cross validating them.\n"
# Compare Algorithms
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
# load dataset
#dataset = pandas.read_csv('HR_Data.csv')
array = dataset.values
X = dataset.iloc[:,[0,1,2,3,4,5,7,8,9]].values
y = dataset.iloc[:,6].values
# prepare configuration for cross validation test harness
seed = 7
# prepare models
models = []
models.append(('LR', LogisticRegression()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('RFC', RandomForestClassifier()))
# evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=seed)
    cv_results = model_selection.cross_val_score(model, X, y, cv=kfold, scoring=s
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```

3.2. Output for Cross Validation

```
#Comparing different Machine Learning models and cross validating them.

LR: 0.748236 (0.240326)
KNN: 0.925063 (0.021160)
CART: 0.979400 (0.015039)
NB: 0.774386 (0.043260)
SVM: 0.945730 (0.028706)
RFC: 0.991267 (0.013950)
```



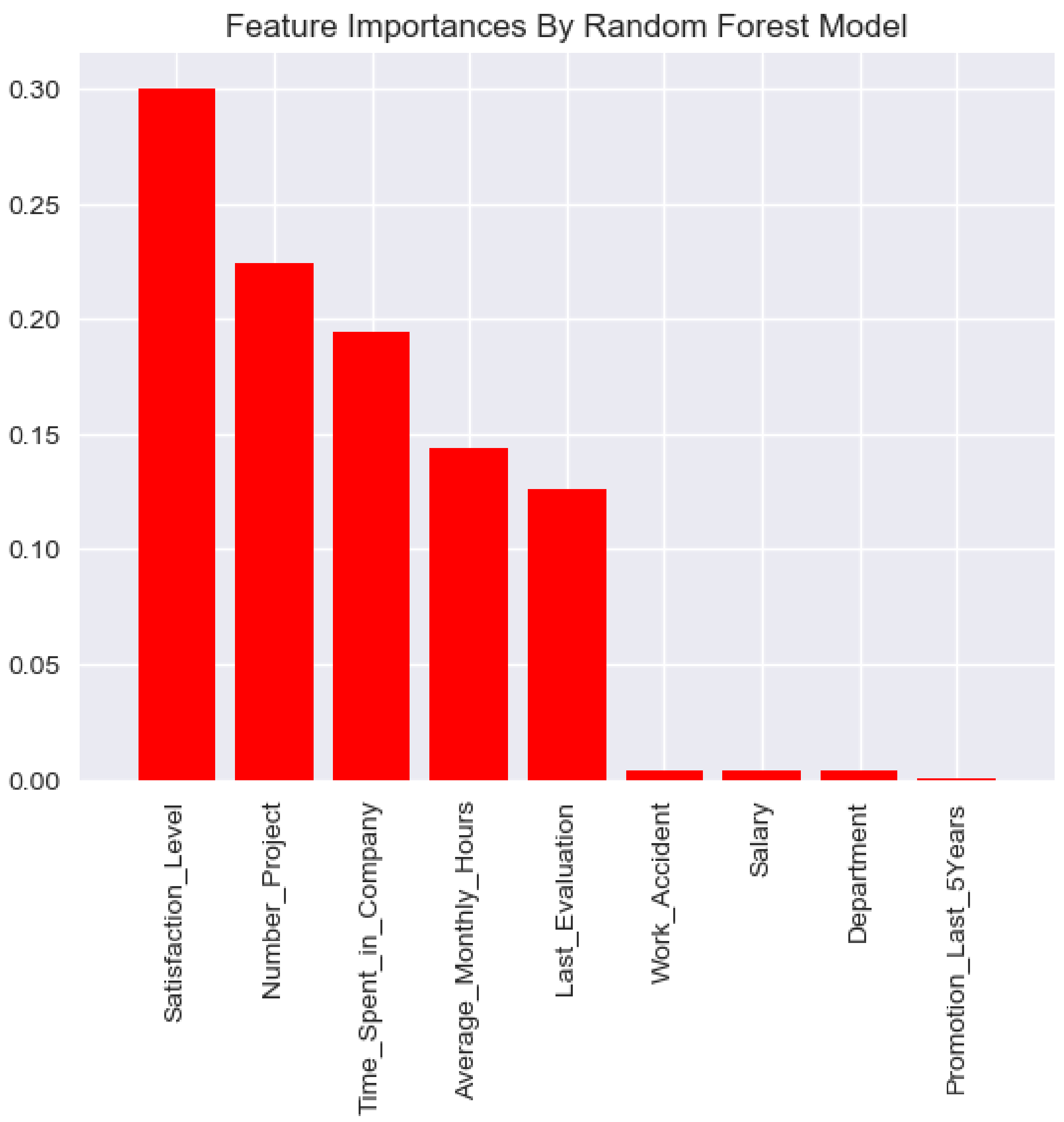
4. Why employees are leaving

We can find why employees are leaving by finding out which of features are having so much importance. We can find feature importance using random forest classifier.

4.1. Syntax for finding Feature importance

```python
from sklearn.ensemble import RandomForestClassifier
predictors = ["Satisfaction_Level", "Last_Evaluation", "Number_Project",
              "Average_Monthly_Hours","Time_Spent_in_Company",
              "Work_Accident", "Promotion_Last_5Years", "Department","Salary"]
rf = RandomForestClassifier(random_state=1, n_estimators=50, max_depth=9,
                            min_samples_split=6, min_samples_leaf=4)
rf.fit(dataset[predictors],dataset["Left"])
importances=rf.feature_importances_
std = np.std([rf.feature_importances_ for tree in rf.estimators_],axis=0)
indices = np.argsort(importances)[::-1]
sorted_important_features=[]
for i in indices:
    sorted_important_features.append(predictors[i])
plt.figure()
plt.title("Feature Importances By Random Forest Model")
plt.bar(range(np.size(predictors)), importances[indices],color="r",
        yerr=std[indices], align="center")
plt.xticks(range(np.size(predictors)), sorted_important_features,
           rotation='vertical')
plt.xlim([-1, np.size(predictors)])
plt.show()
```

4.2. Output for finding feature importance



From the graph we can find that satisfaction level, number of projects, time spent in company, average monthly hours, last evaluation have highest importance. From this we can infer that as the employees who are working on number of projects and they have spent more time in the company and they are working for more than the average monthly hours have left the company.

5. Result

Of all the models Random Forest has the highest accuracy. From the cross validation results we can say that random forest classifier has highest cross-validation mean. From this we can infer that Random Forest Classifier is the best model for the given data.