**IBM HR Analytics Employee Attrition & Performance using KNN**

Attrition is a problem that impacts all businesses, irrespective of geography, industry and size of the company. It is a major problem to an organization, and predicting turnover is at the forefront of the needs of Human Resources (HR) in many organizations. Organizations face huge costs resulting from employee turnover. With advances in machine learning and data science, it’s possible to predict the employee attrition and we will predict using KNN (k-nearest neighbours) algorithm.   
**Dataset:**  
The dataset that is published by the Human Resource department of IBM is made available at Kaggle.   
[dataset](https://www.kaggle.com/pavansubhasht/ibm-hr-analytics-attrition-dataset)  
**Code: Implementation of KNN algorithm for classification.**

* Python3

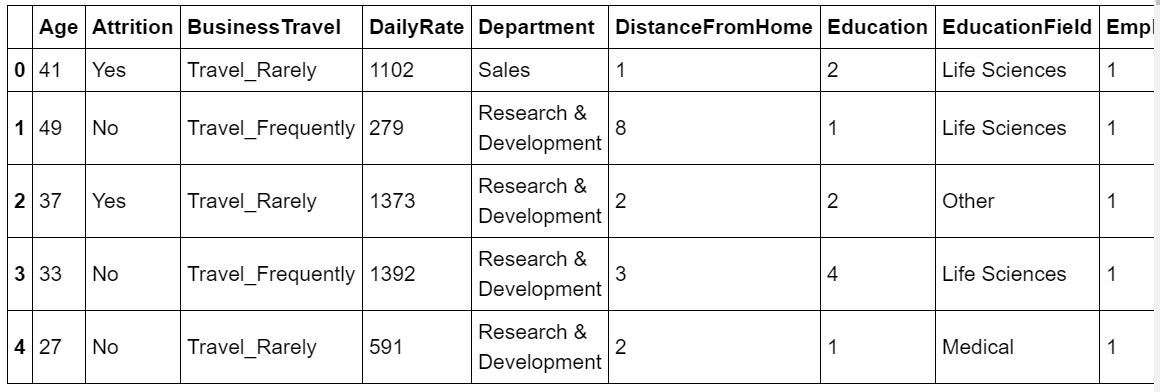
|  |
| --- |
| # performing linear algebra  import numpy as np    # data processing  import pandas as pd    # visualisation  import matplotlib.pyplot as plt  import seaborn as sns % matplotlib inline |

**Code: Importing the dataset**

* Python3

|  |
| --- |
| dataset = pd.read\_csv("WA\_Fn-UseC\_-HR-Employee-Attrition.csv")  print (dataset.head) |

**Output :**



**Code: Information about the dataset**

* Python3

|  |
| --- |
| df.info() |

**Output :**

RangeIndex: 1470 entries, 0 to 1469

Data columns (total 35 columns):

Age 1470 non-null int64

Attrition 1470 non-null object

BusinessTravel 1470 non-null object

DailyRate 1470 non-null int64

Department 1470 non-null object

DistanceFromHome 1470 non-null int64

Education 1470 non-null int64

EducationField 1470 non-null object

EmployeeCount 1470 non-null int64

EmployeeNumber 1470 non-null int64

EnvironmentSatisfaction 1470 non-null int64

Gender 1470 non-null object

HourlyRate 1470 non-null int64

JobInvolvement 1470 non-null int64

JobLevel 1470 non-null int64

JobRole 1470 non-null object

JobSatisfaction 1470 non-null int64

MaritalStatus 1470 non-null object

MonthlyIncome 1470 non-null int64

MonthlyRate 1470 non-null int64

NumCompaniesWorked 1470 non-null int64

Over18 1470 non-null object

OverTime 1470 non-null object

PercentSalaryHike 1470 non-null int64

PerformanceRating 1470 non-null int64

RelationshipSatisfaction 1470 non-null int64

StandardHours 1470 non-null int64

StockOptionLevel 1470 non-null int64

TotalWorkingYears 1470 non-null int64

TrainingTimesLastYear 1470 non-null int64

WorkLifeBalance 1470 non-null int64

YearsAtCompany 1470 non-null int64

YearsInCurrentRole 1470 non-null int64

YearsSinceLastPromotion 1470 non-null int64

YearsWithCurrManager 1470 non-null int64

dtypes: int64(26), object(9)

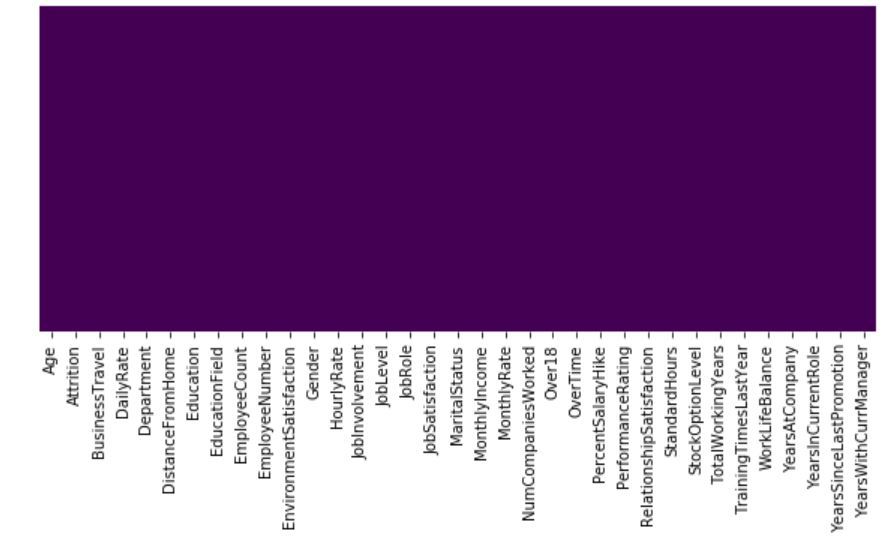
memory usage: 402.0+ KB

**Code: Visualizing the data**

* Python3

|  |
| --- |
| # heatmap to check the missing value  plt.figure(figsize =(10, 4))  sns.heatmap(dataset.isnull(), yticklabels = False, cbar = False, cmap ='viridis') |

**Output:**

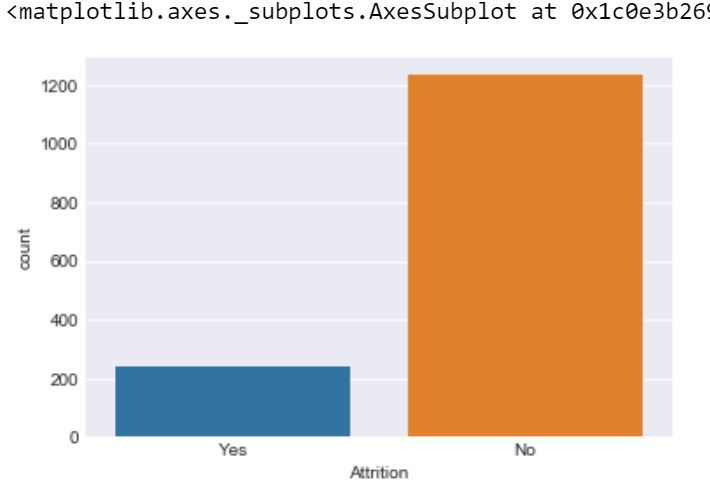


So, we can see that there are no missing values in the dataset.   
This is a Binary Classification Problem, so the Distribution of instances among the 2 classes, is visualized below:

* Python3

|  |
| --- |
| sns.set\_style('darkgrid')  sns.countplot(x ='Attrition', data = dataset) |

**Output:**

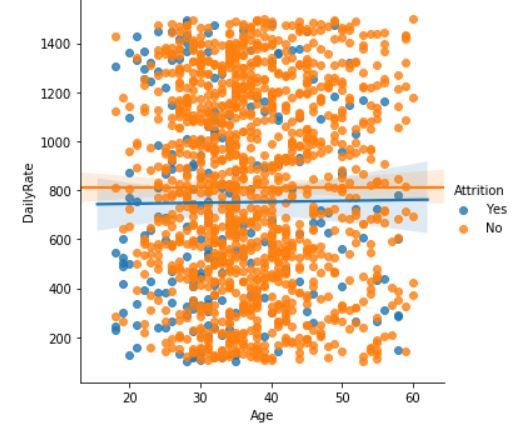


**Code:**

* Python3

|  |
| --- |
| sns.lmplot(x = 'Age', y = 'DailyRate', hue = 'Attrition', data = dataset) |

**Output:**

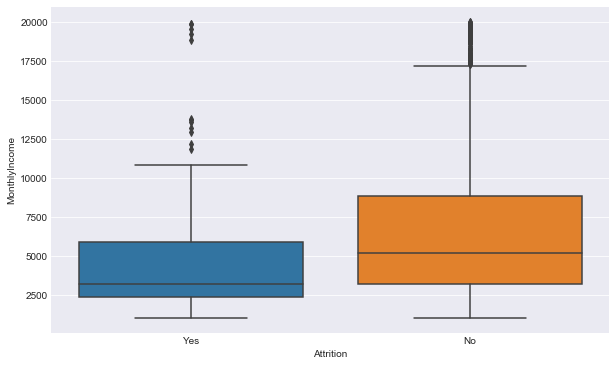


**Code :**

* Python3

|  |
| --- |
| plt.figure(figsize =(10, 6))  sns.boxplot(y ='MonthlyIncome', x ='Attrition', data = dataset) |

**Output:**



**Preprocessing the data**   
In the dataset there are 4 irrelevant columns, i.e:EmployeeCount, EmployeeNumber, Over18 and StandardHour. So, we have to remove these for more accuracy.   
**Code:**

* Python3

|  |
| --- |
| dataset.drop('EmployeeCount', axis = 1, inplace = True)  dataset.drop('StandardHours', axis = 1, inplace = True)  dataset.drop('EmployeeNumber', axis = 1, inplace = True)  dataset.drop('Over18', axis = 1, inplace = True)  print(dataset.shape) |

**Output:**

(1470, 31)

So, we have removed the irrelevant column.  
**Code: Input and Output data**

* Python3

|  |
| --- |
| y = dataset.iloc[:, 1]  X = dataset  X.drop('Attrition', axis = 1, inplace = True) |

**Code: Label Encoding**

* Python3

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder  lb = LabelEncoder()  y = lb.fit\_transform(y) |

In the dataset there are 7 categorical data, so we have to change them to int data, i.e we hava to create 7 dummy variable for better accuracy.  
**Code: Dummy variable creation**

* Python3

|  |
| --- |
| dum\_BusinessTravel = pd.get\_dummies(dataset['BusinessTravel'],                                      prefix ='BusinessTravel')  dum\_Department = pd.get\_dummies(dataset['Department'],                                  prefix ='Department')  dum\_EducationField = pd.get\_dummies(dataset['EducationField'],                                      prefix ='EducationField')  dum\_Gender = pd.get\_dummies(dataset['Gender'],                              prefix ='Gender', drop\_first = True)  dum\_JobRole = pd.get\_dummies(dataset['JobRole'],                               prefix ='JobRole')  dum\_MaritalStatus = pd.get\_dummies(dataset['MaritalStatus'],                                     prefix ='MaritalStatus')  dum\_OverTime = pd.get\_dummies(dataset['OverTime'],                                prefix ='OverTime', drop\_first = True)  # Adding these dummy variable to input X  X = pd.concat([x, dum\_BusinessTravel, dum\_Department,                 dum\_EducationField, dum\_Gender, dum\_JobRole,                 dum\_MaritalStatus, dum\_OverTime], axis = 1)  # Removing the categorical data  X.drop(['BusinessTravel', 'Department', 'EducationField',          'Gender', 'JobRole', 'MaritalStatus', 'OverTime'],          axis = 1, inplace = True)    print(X.shape)  print(y.shape) |

**Output:**

(1470, 49)

(1470, )

**Code: Splitting data to training and testing**

* Python3

|  |
| --- |
| 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 = 40) |

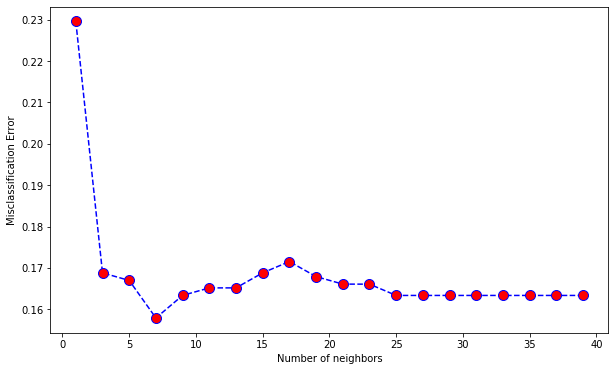
So, the preprocessing is done, now we have to apply KNN to the dataset.   
**Model Execution code: Using KNeighborsClassifier for finding the best number of neighbour with the help of misclassification error.**

* Python3

|  |
| --- |
| from sklearn.neighbors import KNeighborsClassifier  neighbors = []  cv\_scores = []    from sklearn.model\_selection import cross\_val\_score  # perform 10 fold cross validation  for k in range(1, 40, 2):      neighbors.append(k)      knn = KNeighborsClassifier(n\_neighbors = k)      scores = cross\_val\_score(          knn, X\_train, y\_train, cv = 10, scoring = 'accuracy')      cv\_scores.append(scores.mean())  error\_rate = [1-x for x in cv\_scores]    # determining the best k  optimal\_k = neighbors[error\_rate.index(min(error\_rate))]  print('The optimal number of neighbors is % d ' % optimal\_k)    # plot misclassification error versus k  plt.figure(figsize = (10, 6))  plt.plot(range(1, 40, 2), error\_rate, color ='blue', linestyle ='dashed', marker ='o',           markerfacecolor ='red', markersize = 10)  plt.xlabel('Number of neighbors')  plt.ylabel('Misclassification Error')  plt.show() |

**Output:**

The optimal number of neighbors is 7



**Code: Prediction Score**

* Python3

|  |
| --- |
| from sklearn.model\_selection import cross\_val\_predict, cross\_val\_score  from sklearn.metrics import accuracy\_score, classification\_report  from sklearn.metrics import confusion\_matrix    def print\_score(clf, X\_train, y\_train, X\_test, y\_test, train = True):      if train:          print("Train Result:")          print("------------")          print("Classification Report: \n {}\n".format(classification\_report(                  y\_train, clf.predict(X\_train))))          print("Confusion Matrix: \n {}\n".format(confusion\_matrix(                  y\_train, clf.predict(X\_train))))            res = cross\_val\_score(clf, X\_train, y\_train,                                cv = 10, scoring ='accuracy')          print("Average Accuracy: \t {0:.4f}".format(np.mean(res)))          print("Accuracy SD: \t\t {0:.4f}".format(np.std(res)))          print("accuracy score: {0:.4f}\n".format(accuracy\_score(                  y\_train, clf.predict(X\_train))))          print("----------------------------------------------------------")        elif train == False:          print("Test Result:")          print("-----------")          print("Classification Report: \n {}\n".format(                  classification\_report(y\_test, clf.predict(X\_test))))          print("Confusion Matrix: \n {}\n".format(                  confusion\_matrix(y\_test, clf.predict(X\_test))))          print("accuracy score: {0:.4f}\n".format(                  accuracy\_score(y\_test, clf.predict(X\_test))))          print("-----------------------------------------------------------")    knn = KNeighborsClassifier(n\_neighbors = 7)  knn.fit(X\_train, y\_train)  print\_score(knn, X\_train, y\_train, X\_test, y\_test, train = True)  print\_score(knn, X\_train, y\_train, X\_test, y\_test, train = False) |

**Output:**

Train Result:

------------

Classification Report:

precision recall f1-score support

0 0.86 0.99 0.92 922

1 0.83 0.19 0.32 180

accuracy 0.86 1102

macro avg 0.85 0.59 0.62 1102

weighted avg 0.86 0.86 0.82 1102

Confusion Matrix:

[[915 7]

[145 35]]

Average Accuracy: 0.8421

Accuracy SD: 0.0148

accuracy score: 0.8621

-----------------------------------------------------------

Test Result:

-----------

Classification Report:

precision recall f1-score support

0 0.84 0.96 0.90 311

1 0.14 0.04 0.06 57

accuracy 0.82 368

macro avg 0.49 0.50 0.48 368

weighted avg 0.74 0.82 0.77 368

Confusion Matrix:

[[299 12]

[ 55 2]]

accuracy score: 0.8179