IBM_HR_Analytics_Employee_Attrition_Performance

The dataset is about employee attrition. This analysis can discover if any particular factors or patterns that lead to attrition. If so, employers can take certain precausion to prevent attrition which in employer of view, employee attrition is a loss to company, in both monetary and non-monetary.

Import packages

```
##Importing the packages
#Data processing packages
import numpy as np
import pandas as pd

#Visualization packages
import matplotlib.pyplot as plt
import seaborn as sns

#Machine Learning packages
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix

#Suppress warnings
import warnings
warnings.filterwarnings('ignore')
```

Import data

```
#Import Employee Attrition data
data=pd.read_csv('/content/WA_Fn-UseC_-HR-Employee-Attrition.csv')
data.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education
0	41	Yes	Travel_Rarely	1102	Sales	1	2
1	49	No	Travel_Frequently	279	Research & Development	8	1
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2
3	33	No	Travel_Frequently	1392	Research & Development	3	4
4	27	No	Travel_Rarely	591	Research & Development	2	1

5 rows × 35 columns

Check and remediate if there are any null values

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64
 4   Department                1470 non-null   object
```

```
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64
 9   EmployeeNumber            1470 non-null   int64
 10  EnvironmentSatisfaction   1470 non-null   int64
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64
 13  JobInvolvement            1470 non-null   int64
 14  JobLevel                  1470 non-null   int64
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64
 19  MonthlyRate               1470 non-null   int64
 20  NumCompaniesWorked        1470 non-null   int64
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64
 24  PerformanceRating         1470 non-null   int64
 25  RelationshipSatisfaction  1470 non-null   int64
 26  StandardHours             1470 non-null   int64
 27  StockOptionLevel          1470 non-null   int64
 28  TotalWorkingYears         1470 non-null   int64
 29  TrainingTimesLastYear     1470 non-null   int64
 30  WorkLifeBalance           1470 non-null   int64
 31  YearsAtCompany            1470 non-null   int64
 32  YearsInCurrentRole        1470 non-null   int64
 33  YearsSinceLastPromotion   1470 non-null   int64
 34  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

COMMENT: Above output shows that there are No Null values.

Check and remove if there are any fields which does not add value

```
data['Over18'].value_counts()
```

```
Over18
Y    1470
Name: count, dtype: int64
```

COMMENT: From the above output ALL the employees are above 18, so this field does not add any value.

```
data.describe()
```

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNu
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.00
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.86
std	9.135373	403.509100	8.106864	1.024165	0.0	602.02
min	18.000000	102.000000	1.000000	1.000000	1.0	1.00
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.25
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.50
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.75
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.00

8 rows × 26 columns

COMMENT: Standard deviation(std) for the fields "EmployeeCount" and ."StandardHours" are ZERO. Hence these fields does not add value, hence they can be removed.

```
#These fields does not add value, hence removed
data = data.drop(['EmployeeCount','Over18'], axis = 1)
```

Perform datatype conversion or translation wherever required

"Attrition" field has values **Yes/No**, however for machin learning algorithms we need numeric values. Hence translating **Yes/No** to binary **1/0**

```
#A lambda function is a small anonymous function.
#A lambda function can take any number of arguments, but can only have one expression.
data['Attrition']=data['Attrition'].apply(lambda x : 1 if x=='Yes' else 0)
```

Convert Categorical values to Numeric Values

```
data.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education
0	41	1	Travel_Rarely	1102	Sales	1	2
1	49	0	Travel_Frequently	279	Research & Development	8	1
2	37	1	Travel_Rarely	1373	Research & Development	2	2
3	33	0	Travel_Frequently	1392	Research & Development	3	4
4	27	0	Travel_Rarely	591	Research & Development	2	1

5 rows × 33 columns

```
#This function is used to convert Categorical values to Numerical values
data=pd.get_dummies(data)
```

```
data.head()
```

	Age	Attrition	DailyRate	DistanceFromHome	Education	EmployeeNumber	EnvironmentSa
0	41	1	1102	1	2	1	
1	49	0	279	8	1	2	
2	37	1	1373	2	2	4	
3	33	0	1392	3	4	5	
4	27	0	591	2	1	7	

5 rows × 54 columns

COMMENT: It can be seen from the difference in the output of **data.head()** before and after the coversion that now **ALL the fields have numerical values.**

Separating the Feature and Target Matrices

```
#Separating Feature and Target matrices
X = data.drop(['Attrition'], axis=1)
y=data['Attrition']
```

Scaling the data values to standardize the range of independent variables

```
#Feature scaling is a method used to standardize the range of independent variables or features of data.
#Since the range of values of raw data varies widely, in some machine learning algorithms, objective functions will not work properly withou
from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
X = scale.fit_transform(X)
```

Split the data into Training set and Testing set

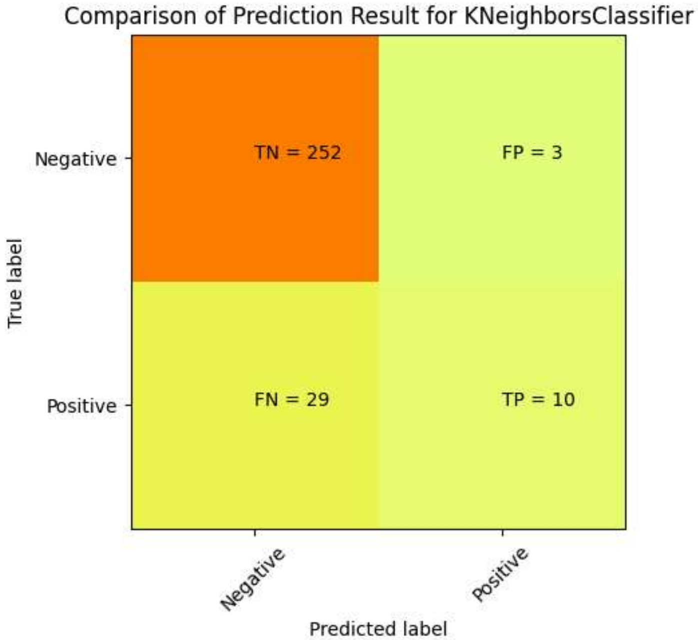
```
# Split the data into Training set and Testing set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size =0.2,random_state=42)
```

Function definition

```
#Function to Train and Test Machine Learning Model
def train_test_ml_model(X_train,y_train,X_test,Model):
    model.fit(X_train,y_train) #Train the Model
    y_pred = model.predict(X_test) #Use the Model for prediction

    # Test the Model
    from sklearn.metrics import confusion_matrix
    cm = confusion_matrix(y_test,y_pred)
    accuracy = round(100*np.trace(cm)/np.sum(cm),1)

    #Plot/Display the results
    cm_plot(cm,Model)
    print('Accuracy of the Model' ,Model, str(accuracy)+'%')


#Function to plot Confusion Matrix
def cm_plot(cm,Model):
    plt.clf()
    plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Wistia)
    classNames = ['Negative','Positive']
    plt.title('Comparison of Prediction Result for '+ Model)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    tick_marks = np.arange(len(classNames))
    plt.xticks(tick_marks, classNames, rotation=45)
    plt.yticks(tick_marks, classNames)
    s = [['TN','FP'], ['FN', 'TP']]
    for i in range(2):
        for j in range(2):
            plt.text(j,i, str(s[i][j])+" = "+str(cm[i][j]))
    plt.show()
```

PERFORM PREDICTIONS USING K NEIGHBORS CLASSIFIER

```
from sklearn.neighbors import KNeighborsClassifier  #Import packages related to Model
Model = "KNeighborsClassifier"
model=KNeighborsClassifier()

train_test_ml_model(X_train,y_train,X_test,Model)
```




```
Accuracy of the Model KNeighborsClassifier 89.1%
```

```
# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled = pd.DataFrame(X_scaled, columns=X.columns)

# Train the model
model = RandomForestClassifier()
model.fit(X_scaled, y)

# Make a new prediction
new_data = pd.DataFrame({
    'Age': [30],
    'BusinessTravel': ['Travel_Frequently'],
    'DailyRate': [300],
    'Department': ['Sales'],
    'DistanceFromHome': [10],
    'Education': [3],
    'EducationField': ['Marketing'],
    'EnvironmentSatisfaction': [2],
    'Gender': ['Male'],
    'HourlyRate': [50],
    'JobInvolvement': [3],
    'JobLevel': [2],
    'JobRole': ['Sales Executive'],
    'JobSatisfaction': [4],
    'MaritalStatus': ['Single'],
    'MonthlyIncome': [5000],
    'MonthlyRate': [15],
    'NumCompaniesWorked': [2],
    'OverTimePercent': [10],
    'PerformanceRating': [3],
    'RelationshipSatisfaction': [3],
    'SalaryHike': [15],
    'StandardHours': [80],
    'StockOptionLevel': [1],
    'TotalWorkingYears': [5],
    'TrainingTimesLastYear': [2],
    'WorkLifeBalance': [3],
    'YearsAtCompany': [3],
    'YearsInCurrentRole': [2],
    'YearsSinceLastPromotion': [1],
    'YearsWithCurrManager': [2]
})

# Preprocess the new data
new_data = pd.get_dummies(new_data)
new_data = new_data.reindex(columns=X_scaled.columns, fill_value=0)
new_data_scaled = scaler.transform(new_data)
```

```
# Make the prediction
prediction = model.predict(new_data_scaled)

# Print the prediction
print(f"Predicted Attrition: {'Yes' if prediction[0] == 1 else 'No'}")
```