HR Analytics Project- Understanding the Attrition in HR

AUTHOR

SAYAN KUMAR BHUNIA

INTRODUCTION

➢ Problem Statement :

Every year a lot of companies hire a number of employees. The companies invest time and money in training those employees, not just this but there are training programs within the companies for their existing employees as well. The aim of these programs is to increase the effectiveness of their employees. But where HR Analytics fit in this? and is it just about improving the performance of employees?

➢ Conceptual Background of the Domain Problem

Human resource analytics (HR analytics) is an area in the field of analytics that refers to applying analytic processes to the human resource department of an organization in the hope of improving employee performance and therefore getting a better return on investment. HR analytics does not just deal with gathering data on employee efficiency. Instead, it aims to provide insight into each process by gathering data and then using it to make relevant decisions about how to improve these processes.

Analytical Problem Framing

➢ Mathematical/ Analytical Modelling of the Problem

We shall build a supervised classification model to predict attrition.

Now, when we talk about building a supervised classifier catering to certain use-case, for example, classifying risk of loan default, following three things come into our minds:

- **Data** appropriate to the business requirement or use-case we are trying to solve
- A **classification model** which we think (or, rather assess) to be the best for our solution.
- **Optimize** the chosen model to ensure best performance.

➢ Data Sources and their formats

I am using CSV (comma-separated values) format file which is having 1470 rows × 35 columns

```
In [2]: df=pd.read_csv(r"C:\Users\sayan\OneDrive\Desktop\WA_Fn-UseC_-HR-Employee-Attrition.csv")
        df
```

Out[2]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1
...	...	...	...	...	...	...	...	...	...
1465	36	No	Travel_Frequently	884	Research & Development	23	2	Medical	1
1466	39	No	Travel_Rarely	613	Research & Development	6	1	Medical	1
1467	27	No	Travel_Rarely	155	Research & Development	4	3	Life Sciences	1
1468	49	No	Travel_Frequently	1023	Sales	2	3	Medical	1
1469	34	No	Travel_Rarely	628	Research & Development	8	3	Medical	1

1470 rows × 35 columns

```
In [3]: df.head()
```

➢ Data Pre-processing and EDA

Following steps have been performed on the data.

- **checking missing values**-
  - If there is any missing value present in your data set then for a better and correct accuracy you have to impute it.
  - If missing data present in object type column, then you have to take most frequent value for your missing data.
  - If missing data present in int or float type column then use mean/median for missing value.

In the following case no missing value present:

```
In [5]: df.isnull().sum()

Out[5]: Age                         0
        Attrition                   0
        BusinessTravel              0
        DailyRate                   0
        Department                  0
        DistanceFromHome            0
        Education                   0
        EducationField              0
        EmployeeCount               0
        EmployeeNumber              0
        EnvironmentSatisfaction     0
        Gender                      0
        HourlyRate                  0
        JobInvolvement              0
        JobLevel                    0
        JobRole                     0
        JobSatisfaction             0
        MaritalStatus               0
        MonthlyIncome               0
        MonthlyRate                 0
        NumCompaniesWorked          0
        Over18                      0
        OverTime                    0
        PercentSalaryHike           0
        PerformanceRating           0
        RelationshipSatisfaction    0
        StandardHours               0
        StockOptionLevel            0
        TotalWorkingYears           0
        TrainingTimesLastYear       0
        WorkLifeBalance             0
        YearsAtCompany              0
        YearsInCurrentRole          0
        YearsSinceLastPromotion     0
        YearsWithCurrManager        0
        dtype: int64
```

- **Encoding categorical variables** -as we can see there are 9 object data type columns present so we will encode it into (int) format.
  - Apply **Label Encoding**, if number of categories in a categorical variable is equal to 2.
  - Apply **One-Hot Encoding**, if number of categories in a categorical variable is greater than 2.

In the following case Label Encoding is used.

In [6]:
```
df.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)
```

In [7]:
```
import sklearn
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
```

In [8]:
```
,'BusinessTravel','Department','EducationField','Gender','JobRole','MaritalStatus','Over18','OverTime']

ransform(df[i].astype(str))
```

Out[8]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	E
0	41	1	2	1102	2	1	2	1	1	
1	49	0	1	279	1	8	1	1	1	
2	37	1	2	1373	1	2	2	4	1	
3	33	0	1	1392	1	3	4	1	1	
4	27	0	2	591	1	2	1	3	1	
...	...	...	...	...	...	...	...	...	...	
1465	36	0	1	884	1	23	2	3	1	
1466	39	0	2	613	1	6	1	3	1	
1467	27	0	2	155	1	4	3	1	1	
1468	49	0	1	1023	2	2	3	3	1	
1469	34	0	2	628	1	8	3	3	1	

1470 rows × 35 columns

- **Feature scaling-** Feature Scaling ensures that all features will get equal importance in supervised classifier models. Standard scaler was used to scale all features in the data.

```
In [50]: from sklearn.preprocessing import StandardScaler

In [51]: sc=StandardScaler()
         x=sc.fit_transform(x)
         x

Out[51]: array([[ 0.5366811 ,  0.51686825,  0.73432467, ...,  0.04090327,
                 -0.99236588,  0.36826963],
                [ 1.44211114, -0.43742516, -1.30776866, ...,  0.96589518,
                  0.0648491 ,  0.9821999 ],
                [ 0.08396607,  0.51686825,  1.40675151, ..., -1.19241928,
                 -0.99236588, -1.16655603],
                ...,
                [-1.04782149,  0.51686825, -1.61544736, ..., -0.575758  ,
                 -0.99236588, -0.24566063],
                [ 1.44211114, -0.43742516,  0.53830356, ...,  0.65756454,
                 -0.99236588,  1.28916503],
                [-0.2555702 ,  0.51686825, -0.44180199, ..., -0.26742737,
                  0.0648491 , -0.55262577]])
```

- **Reducing dimension of the data-** Sklearn's pca can be used to apply principal component analysis on the data. This helped in finding the vectors of maximal variance in the data.

- **Outliers detection-** In simple words, an outlier is an observation that diverges from an overall pattern on a sample.

  In the following case box plot is used to detect outliers.

```
[17]: df.plot(kind='box',subplots=True,layout=(10,20))

[17]: Age                              AxesSubplot(0.125,0.816017;0.032563x0.0639831)
      Attrition                     AxesSubplot(0.164076,0.816017;0.032563x0.0639831)
      BusinessTravel                AxesSubplot(0.203151,0.816017;0.032563x0.0639831)
      DailyRate                     AxesSubplot(0.242227,0.816017;0.032563x0.0639831)
      Department                    AxesSubplot(0.281303,0.816017;0.032563x0.0639831)
      DistanceFromHome              AxesSubplot(0.320378,0.816017;0.032563x0.0639831)
      Education                     AxesSubplot(0.359454,0.816017;0.032563x0.0639831)
      EducationField                AxesSubplot(0.398529,0.816017;0.032563x0.0639831)
      EmployeeCount                 AxesSubplot(0.437605,0.816017;0.032563x0.0639831)
      EmployeeNumber                AxesSubplot(0.476681,0.816017;0.032563x0.0639831)
      EnvironmentSatisfaction       AxesSubplot(0.515756,0.816017;0.032563x0.0639831)
      Gender                        AxesSubplot(0.554832,0.816017;0.032563x0.0639831)
      HourlyRate                    AxesSubplot(0.593908,0.816017;0.032563x0.0639831)
      JobInvolvement                AxesSubplot(0.632983,0.816017;0.032563x0.0639831)
      JobLevel                      AxesSubplot(0.672059,0.816017;0.032563x0.0639831)
      JobRole                       AxesSubplot(0.711134,0.816017;0.032563x0.0639831)
      JobSatisfaction                AxesSubplot(0.75021,0.816017;0.032563x0.0639831)
      MaritalStatus                 AxesSubplot(0.789286,0.816017;0.032563x0.0639831)
      MonthlyIncome                 AxesSubplot(0.828361,0.816017;0.032563x0.0639831)
      MonthlyRate                   AxesSubplot(0.867437,0.816017;0.032563x0.0639831)
      NumCompaniesWorked               AxesSubplot(0.125,0.739237;0.032563x0.0639831)
      Over18                        AxesSubplot(0.164076,0.739237;0.032563x0.0639831)
      OverTime                      AxesSubplot(0.203151,0.739237;0.032563x0.0639831)
      PercentSalaryHike             AxesSubplot(0.242227,0.739237;0.032563x0.0639831)
      PerformanceRating             AxesSubplot(0.281303,0.739237;0.032563x0.0639831)
      RelationshipSatisfaction      AxesSubplot(0.320378,0.739237;0.032563x0.0639831)
      StandardHours                 AxesSubplot(0.359454,0.739237;0.032563x0.0639831)
      StockOptionLevel              AxesSubplot(0.398529,0.739237;0.032563x0.0639831)
      TotalWorkingYears             AxesSubplot(0.437605,0.739237;0.032563x0.0639831)
      TrainingTimesLastYear         AxesSubplot(0.476681,0.739237;0.032563x0.0639831)
      WorkLifeBalance               AxesSubplot(0.515756,0.739237;0.032563x0.0639831)
      YearsAtCompany                AxesSubplot(0.554832,0.739237;0.032563x0.0639831)
      YearsInCurrentRole            AxesSubplot(0.593908,0.739237;0.032563x0.0639831)
      YearsSinceLastPromotion       AxesSubplot(0.632983,0.739237;0.032563x0.0639831)
      YearsWithCurrManager          AxesSubplot(0.672059,0.739237;0.032563x0.0639831)
      dtype: object
```

There are many types of outlier detection techniques such as Z-Score or Extreme Value Analysis, Probabilistic and Statistical Modelling, Information Theory Models, Standard Deviation etc.

➢ Outliers Removal

In our dataset, we observed variations in the relation between values of some attributes.

So that these types of rows are dropped from the dataset.

In [27]:
```
threshold=3
print(np.where(z>3))
```

```
(array([  28,   45,   62,   62,   63,   64,   85,   98,   98,  110,  123,
        123,  123,  126,  126,  126,  153,  178,  187,  187,  190,  190,
        218,  231,  231,  237,  237,  270,  270,  281,  326,  386,  386,
        401,  411,  425,  425,  427,  445,  466,  473,  477,  535,  561,
        561,  584,  592,  595,  595,  595,  616,  624,  635,  653,  653,
        677,  686,  701,  716,  746,  749,  752,  799,  838,  861,  861,
        875,  875,  894,  914,  914,  918,  922,  926,  926,  937,  956,
        962,  976,  976, 1008, 1024, 1043, 1078, 1078, 1086, 1086, 1093,
       1111, 1116, 1116, 1135, 1138, 1138, 1156, 1184, 1221, 1223, 1242,
       1295, 1301, 1301, 1303, 1327, 1331, 1348, 1351, 1401, 1414, 1430],
```

[25]:
```
from scipy.stats import zscore
z=np.abs(zscore(df))
z
```

:[25]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	Gender	Hou
0	0.446350	2.280906	0.590048	0.742527	1.401512	1.010909	0.891688	0.937414	1.224745	1
1	1.322365	0.438422	0.913194	1.297775	0.493817	0.147150	1.868426	0.937414	0.816497	0
2	0.008343	2.280906	0.590048	1.414363	0.493817	0.887515	0.891688	1.316673	0.816497	1
3	0.429664	0.438422	0.913194	1.461466	0.493817	0.764121	1.061787	0.937414	1.224745	0
4	1.086676	0.438422	0.590048	0.524295	0.493817	0.887515	1.868426	0.565311	0.816497	1
...	...	...	...	...	...	...	...	...	...	
1465	0.101159	0.438422	0.913194	0.202082	0.493817	1.703764	0.891688	0.565311	0.816497	1
1466	0.227347	0.438422	0.590048	0.469754	0.493817	0.393938	1.868426	0.565311	0.816497	1
1467	1.086676	0.438422	0.590048	1.605183	0.493817	0.640727	0.085049	0.937414	0.816497	1
1468	1.322365	0.438422	0.913194	0.546677	1.401512	0.887515	0.085049	0.565311	0.816497	0

In [28]:
```
df_new=df[(z<3).all(axis=1)]
df_new
```

➢ Software Requirements and library Used

```
[1]: import pandas
     import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
     import warnings
     warnings.filterwarnings('ignore')
```

```
10]: import sklearn
     from sklearn.preprocessing import LabelEncoder,OneHotEncoder
```

- **NumPy**
  NumPy is a popular Python library for multi-dimensional array and matrix processing because it can be used to perform a great variety of mathematical operations. Its capability to handle linear algebra, Fourier transform, and more, makes NumPy ideal for machine learning and artificial intelligence (AI) projects, allowing users to manipulate the matrix to easily improve machine learning performance. NumPy is faster and easier to use than most other Python libraries.

- **Scikit-learn**
  Scikit-learn is a very popular machine learning library that is built on NumPy and SciPy. It supports most of the classic supervised and unsupervised learning algorithms, and it can also be used for data mining, modelling, and analysis.

- **Seaborn**
  Seaborn is another open-source Python library, one that is based on Matplotlib (which focuses on plotting and data visualization) but features Pandas' data structures. Seaborn is often used in ML projects because it can generate plots of learning data. Of all the Python libraries, it produces the most aesthetically pleasing graphs and plots, making it an effective choice if you'll also use it for marketing and data analysis.

- **Pandas**

  Pandas is another Python library that is built on top of NumPy, responsible for preparing high-level data sets for machine learning and training. It relies on two types of data structures, one-dimensional (series) and two-dimensional (Data Frame). This allows Pandas to be applicable in a variety of industries including finance, engineering, and statistics. Unlike the slow-moving animals themselves, the Pandas library is quick, compliant, and flexible.

➢ Class imbalance problem

The first challenge we hit upon exploring the data, is class imbalanced problem. Imbalance data will lead to a bad accuracy of a model. To achieve better accuracy, we'll balance the data by using Smote Over Sampling Method.

```
In [52]: from imblearn import under_sampling, over_sampling

In [53]: from imblearn.over_sampling import SMOTE

In [54]: smt=SMOTE()
         dfx,dfy=smt.fit_resample(x,y)

In [55]: dfy.value_counts()

Out[55]: 1    1158
         0    1158
         Name: Attrition, dtype: int64
```

Model/s Development and Evaluation

➢ Run and evaluate selected models

Let's select our classification model for this project:

- Logistic Regression
- KNeighborsClassifier
- SVC
- DecisionTreeClassifier

➢ Testing of Identified Approaches (Algorithms)

I have used train_test_ split.I selected 70% data for traing purpose and 30% data for testing purpose.

```
In [56]: import sklearn
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.svm import SVC
         from sklearn.linear_model import LogisticRegression
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
         from sklearn.model_selection import train_test_split
```

```
In [57]: x_train,x_test,y_train,y_test=train_test_split(dfx,dfy,test_size=.30,random_state=45)
```

Then I build the required model for training and testing purpose.

```
In [62]: lg=LogisticRegression()
         knn=KNeighborsClassifier()
         dtc=DecisionTreeClassifier()
         svc=SVC()
         model=[lg,knn,dtc,svc]
         for m in model:
             m.fit(x_train,y_train)
             predm=m.predict(x_test)
             print('accuracy_score of',m ,'is:')
             print(accuracy_score(y_test,predm))
             print(confusion_matrix(y_test,predm))
             print(classification_report(y_test,predm))
             print('\n')
```

➢ Key Metrics for success in solving problem under consideration

Selection of a model requires evaluation and evaluation requires a good metric. This is indeed important. If we optimize a model based on incorrect metric, then, our model might not be suitable for the business goals.

We have a number of metrics, for example, accuracy, recall, precision, F1 score, area under receiver operating characteristic curve, to choose from.

Svc model is giving high accuracy around 90%.I will select svc model for prediction.

```
accuracy_score of SVC() is:
0.902158273381295
[[315  40]
 [ 28 312]]
              precision    recall  f1-score   support

           0       0.92      0.89      0.90       355
           1       0.89      0.92      0.90       340

    accuracy                           0.90       695
   macro avg       0.90      0.90      0.90       695
weighted avg       0.90      0.90      0.90       695
```

➢ Visualizations

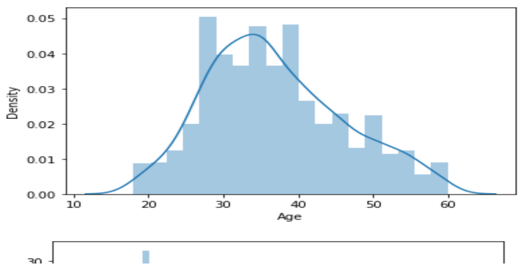
For better understanding of outliers, I have used boxplot.

```
[17]: df.plot(kind='box',subplots=True,layout=(10,20))
```

```
t[17]: Age                         AxesSubplot(0.125,0.816017;0.032563x0.0639831)
       Attrition                AxesSubplot(0.164076,0.816017;0.032563x0.0639831)
       BusinessTravel           AxesSubplot(0.203151,0.816017;0.032563x0.0639831)
       DailyRate                AxesSubplot(0.242227,0.816017;0.032563x0.0639831)
RelationshipSatisfaction    AxesSubplot(0.320378,0.739237;0.032563x0.0639831)
StandardHours               AxesSubplot(0.359454,0.739237;0.032563x0.0639831)
StockOptionLevel            AxesSubplot(0.398529,0.739237;0.032563x0.0639831)
TotalWorkingYears           AxesSubplot(0.437605,0.739237;0.032563x0.0639831)
TrainingTimesLastYear       AxesSubplot(0.476681,0.739237;0.032563x0.0639831)
WorkLifeBalance             AxesSubplot(0.515756,0.739237;0.032563x0.0639831)
YearsAtCompany              AxesSubplot(0.554832,0.739237;0.032563x0.0639831)
YearsInCurrentRole          AxesSubplot(0.593908,0.739237;0.032563x0.0639831)
YearsSinceLastPromotion     AxesSubplot(0.632983,0.739237;0.032563x0.0639831)
YearsWithCurrManager        AxesSubplot(0.672059,0.739237;0.032563x0.0639831)
dtype: object
```

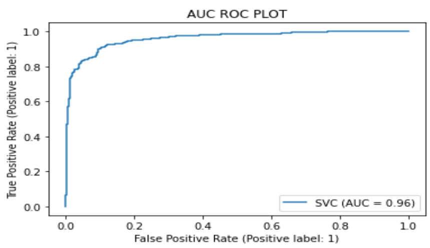
For better understanding of skewness, I have used d istribution plot.

```
In [18]: for i in df.columns:
             plt.figure()
             sns.distplot(df[i])
```



AUC AND ROC curve

```
In [295]: from sklearn.metrics import plot_roc_curve
          plot_roc_curve(gcv.best_estimator_,x_test,y_test)
          plt.title("AUC ROC PLOT")
          plt.show()
```



CONCLUSION

Key aspects of building successful classifier are:

- Selecting correct data according to the purpose or problem statement.
- Proper processing and understanding of the data
- Selecting the model and optimizing the model.

In this project I have dealt with outliers and using z score I removed those outliers for better accuracy.

I have used label encoder for encoding object data type into int datatype as machine doesn’t understand object type data.

I have performed Smote operation as data was imbalanced.

I have used three classification model and found support vector classifier Classifier to be the best fit. After hypermeter turning AUC score is 96% which is pretty good for selecting a model.