HR Analytics Project - Understanding the Attrition in HR

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 objective of the model to increase the effectiveness of their employees and reduce the time and money investing in employees.

HR Analytics:-

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.

Attrition in HR:-

Attrition in human resources refers to the gradual loss of employee's overtime. In general, relatively high attrition is problematic for companies. HR professionals often assume a leadership role in designing company compensation programs, work culture, and motivation systems that help the organization retain top employees.

How does Attrition affect companies? And how does HR Analytics help in analyzing attrition? We will discuss the first question here and for the second question, we will write the code and try to understand the process step by step.

Attrition affecting Companies:-

A major problem in high employee attrition is its cost to an organization. Job postings, hiring processes, paperwork, and new hire training are some of the common expenses of losing employees and replacing them. Additionally, regular employee turnover prohibits your organization from increasing its collective knowledge base and experience over time. This is especially concerning if your business is customer-facing, as customers often prefer to interact with familiar people. Errors and issues are more likely if you constantly have new workers.

Importing the Libraries:-

```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
from matplotlib import pyplot as plt
```

EXPLORATORY DATA ANALYSIS:-

```
In [4]: df.dtypes

Out[4]: Age                         int64
        Attrition                  object
        BusinessTravel             object
        DailyRate                   int64
        Department                 object
        DistanceFromHome            int64
        Education                   int64
        EducationField             object
        EmployeeCount               int64
        EmployeeNumber              int64
        EnvironmentSatisfaction     int64
        Gender                     object
        HourlyRate                  int64
        JobInvolvement              int64
        JobLevel                    int64
        JobRole                    object
        JobSatisfaction             int64
        MaritalStatus              object
        MonthlyIncome               int64
        MonthlyRate                 int64
        NumCompaniesWorked          int64
        Over18                     object
        OverTime                   object
        PercentSalaryHike           int64
        PerformanceRating           int64
        RelationshipSatisfaction    int64
        StandardHours               int64
        StockOptionLevel            int64
        TotalWorkingYears           int64
        TrainingTimesLastYear       int64
        WorkLifeBalance             int64
        YearsAtCompany              int64
        YearsInCurrentRole          int64
        YearsSinceLastPromotion     int64
        YearsWithCurrManager        int64
        dtype: object
```


The features 'Attrition','BusinessTravel','Department','EducationField','Gender','JobRole','MaritalStatus','OverTime',over18 are object type remainiong all are integer type

Numeric variables:-

- Related to personal information: age, distance_from_home, employee_number

- Related to income: hourly_rate, daily_rate, monthly_rate, monthly_income, percent_salary_hike

Related to duration in company: years_at_company, years_in_current_role, years_since_last_promotion, years_with_curr_manager, total_working_years num_companies_worked,standard_hourstraining_times_last_year, employee_count

Categorical variables:-

- Binary variables: attrition(target variable), gender, over18, over_time
- Nominal variables: department, education_field, job_role, marital_status
- Ordinal variables:

Ordinal regarding satisfaction and performance: environment_satisfaction, job_satisfaction, relationship_satisfaction, work_life_balance, job_involvement, performance_rating

➢ Other ordinal: business travel, education, job_level, stock_option_level

```
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
```

dataset has no null values

This dataset has no null values

```
In [6]: df['StandardHours'].unique
```

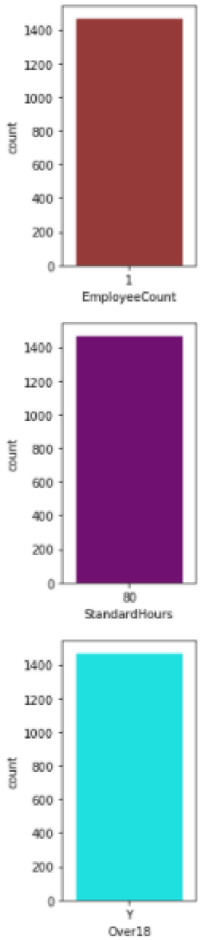
```
Out[6]: <bound method Series.unique of 0       80
1       80
2       80
3       80
4       80
        ..
1465    80
1466    80
1467    80
1468    80
1469    80
Name: StandardHours, Length: 1470, dtype: int64>
```

```
In [7]: df['EmployeeCount'].unique
```

```
Out[7]: <bound method Series.unique of 0       1
1       1
2       1
3       1
4       1
       ..
1465    1
1466    1
1467    1
1468    1
1469    1
Name: EmployeeCount, Length: 1470, dtype: int64>
```

Displaying value count of unique value in each feature. To identify column having single unique value

UNI VARIATE ANALYSIS:-



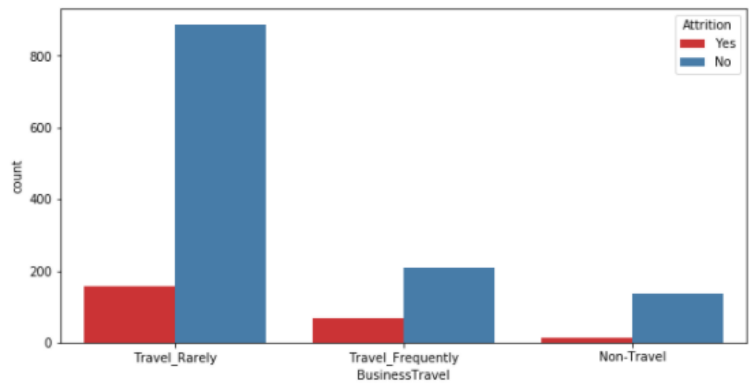
The features standardhours,over18 and employeecount has only one value so it wont create any impact on the target feature Attrition.

The features standard hours, over18 and employee count has only one value so it won’t create any impact on the target feature Attrition.
All the three columns having single value so I'm going to dropping it from the given dataset.

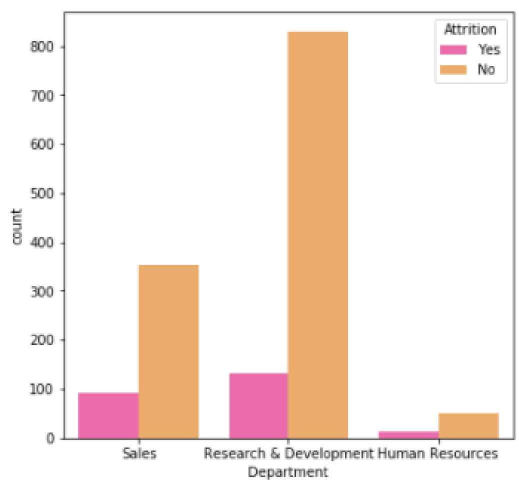
BI VARIATE ANALYSIS (Categorical columns vs Target)

In [12]:
```
plt.figure(figsize=(10,5))
sns.countplot(df.BusinessTravel,palette="Set1",hue=df.Attrition)
```

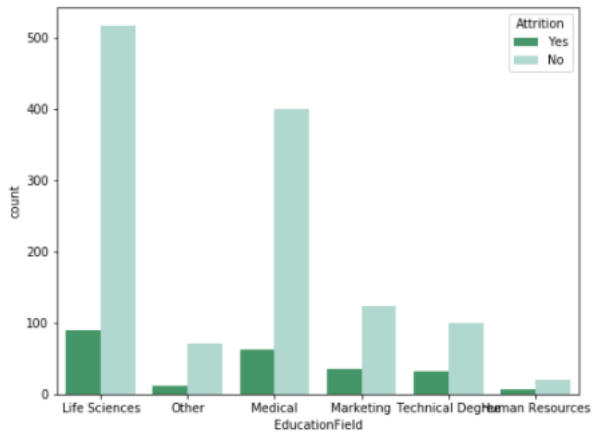
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x20b09313708>



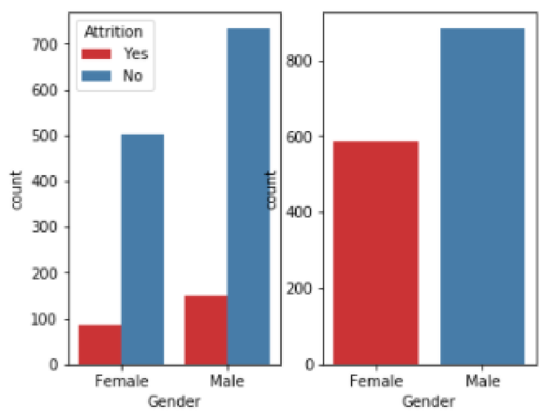
From the above chart it is concluded that employees who travel rarely have high attrition.



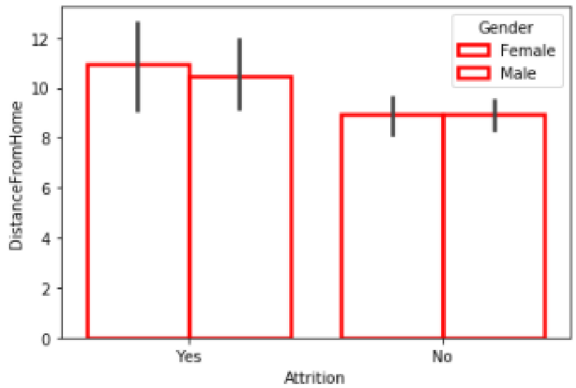
From the above plot it's apparent that comparatively the employees who belongs to Research and Development will like to continue their job.



From the above plot it’s apparent that comparatively the employees who belongs to both lifesciece and medical field are not resigning their job.



Comparing the percentage of attrition out of 588 female only 88 people are quitting.



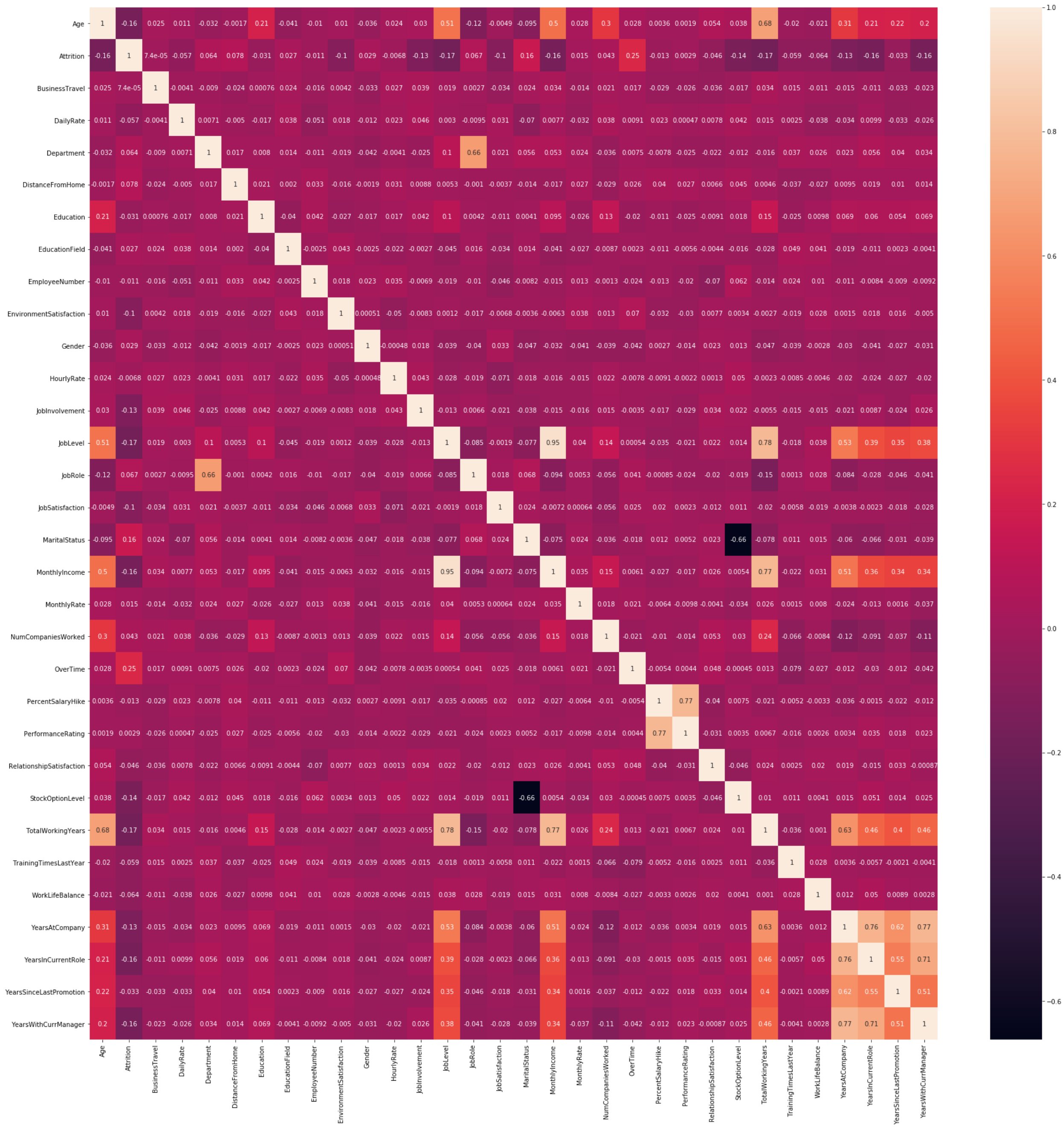
Distance from home is not an important feature to create impact on Attrition feature. Distance from home is not impact on gender.

<u>EDA CONCLUSION:-</u>

Employees who belongs to below category having less attrition on rate

- travel rarely
- who belongs to R&D department
- who belongs to life science and medical field
- female
- working as sales executive and research scientists
- unmarried
- not working over time
- moderate work life balance
- high job involvement
- working in single company
- performance rating:3
- employees who having 0 stocks
- low monthly income.

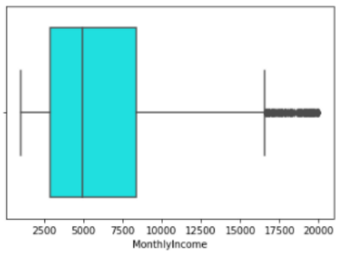
We cannot predict using relationship satisfaction, job satisfaction, Environment satisfaction features



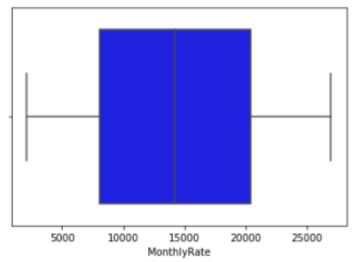
Over time feature is highly correlated with attrition.

Handling outliers in numerical column:-

In [28]:
```
sns.boxplot(df['MonthlyIncome'],color="cyan")
```
Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x20b099b27c8>



In [29]:
```
sns.boxplot(df['MonthlyRate'],color="blue")
```
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x20b0993ddc8>



In [32]:
```
z1 = np.abs(stats.zscore(df_new['MonthlyIncome']))
print(z1)
```
```
[0.10834951 0.29171859 0.93765369 ... 0.07669019 0.23647414 0.44597809]
```

In [33]:
```
df_new['MonthlyIncome'] = df_new.MonthlyIncome[(z1<3)]
df_new.shape
```
Out[33]: (1470, 32)

outliers are removed from numerical data monthly income

Data pre-processing:-

The features standard hours, over18 and employee count has only single value so it won't create any impact on the target feature Attrition.

Employee Number feature is just an identifier and it's not required for modelling either. So I'm dropping these features

The features standardhours,over18 and employeecount has only one value so it wont create any impact on the target feature Attrition.

In [4]:
```
cols=['StandardHours','Over18','EmployeeCount']
df_new=df.drop(cols,axis=1)
```

All the three columns having single value so I'm dropping it from the given dataset

In [4]:
```
df_new.shape
```
Out[4]: (1470, 32)

Encoding all categorical column into numerical column using label encoding technique

Encoding all categorical coloumn into numerical column using label encoding technique

In [36]:
```
data_clean=df_new
col_encod=['Attrition','BusinessTravel','Department','EducationField','Gender','JobRole','MaritalStatus','OverTime']
```

In [37]:
```
from sklearn import preprocessing
for col in col_encod:
    label = preprocessing.LabelEncoder()
    data_clean[col]= label.fit_transform(df_new[col])
```

CHECKING FOR SKEWNESS AND USING METHOD TO REMOVE SKEWNESS:-

In [40]:
```
data_clean.skew()
```

Out[40]:
```
Age                         0.413286
Attrition                   1.844366
BusinessTravel             -1.439006
DailyRate                  -0.003519
Department                  0.172231
DistanceFromHome            0.958118
Education                  -0.289681
EducationField              0.550371
EmployeeNumber              0.016574
EnvironmentSatisfaction    -0.321654
Gender                     -0.408665
HourlyRate                 -0.032311
JobInvolvement             -0.498419
JobLevel                    1.025401
JobRole                    -0.357270
JobSatisfaction            -0.329672
MaritalStatus              -0.152175
MonthlyIncome               1.369817
MonthlyRate                 0.018578
NumCompaniesWorked          1.026471
OverTime                    0.964489
PercentSalaryHike           0.821128
PerformanceRating           1.921883
RelationshipSatisfaction   -0.302828
StockOptionLevel            0.968980
TotalWorkingYears           1.117172
TrainingTimesLastYear       0.553124
WorkLifeBalance            -0.552480
YearsAtCompany              1.764529
YearsInCurrentRole          0.917363
YearsSinceLastPromotion     1.984290
YearsWithCurrManager        0.833451
dtype: float64
```

In [41]:
```
from sklearn.preprocessing import power_transform
x = power_transform(data_clean,method='yeo-johnson')
```

We can check that out imput dataset X is having some skewness so we used yeo-johnson method to remove the skewness from our dataset

HANDLING CLASS IMBALANCE:-

Classification problem where the distribution of examples across the known classes is biased or skewed. To avoid this we are using SMOTE technique

SMOTE synthetic over-sampling works to cause the classifier to build larger decision regions that contain nearby minority class points. This will in turn avoid data loss.

In [42]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x = sc.fit_transform(x)
```

As we have checked earlier our dataset having class imbalancing so we are using standard scaler to balance the dataset before initializing building of our model.

MODELLING:-

It is a binary classification problem so I have modelled using logistic regression and other classification models.

In [45]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn .ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import cross_val_score
from matplotlib import pyplot
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
x=scaled
y=y1
```

In [46]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.25,random_state=1)

models=[LogisticRegression(),RandomForestClassifier(),KNeighborsClassifier(),GradientBoostingClassifier(),DecisionTreeClassifier()]
scorelist=[]
acclist=[]
```

In [47]:

```python
def create_model(model):
    m=model
    m.fit(xtrain,ytrain)
    p=m.predict(xtest)
    score=m.score(xtest,ytest)
    result = confusion_matrix(ytest,p)
    result1 = classification_report(ytest,p)
    result2 = accuracy_score(ytest,p)
    scorelist.append(score)
    acclist.append(result2)


    print(m,"\n")

    print('Accuracy score:',score,"\n")
    print('"Confusion Matrix:\n"',result)
    print('classification_report\n',result1)
    print('Average accuracy_score',result2)
    print('-----------------------------------------------------------------------------------------------
for i in models:
    create_model(i)

print('Maximun accuracy Score is shown by',models[acclist.index(max(acclist))],max(acclist))
```

```
LogisticRegression()

Accuracy score: 0.8586956521739131

"Confusion Matrix:
" [[297   3]
 [ 49  19]]
classification_report
               precision    recall  f1-score   support

           0       0.86      0.99      0.92       300
           1       0.86      0.28      0.42        68

    accuracy                           0.86       368
   macro avg       0.86      0.63      0.67       368
weighted avg       0.86      0.86      0.83       368

Average accuracy_score 0.8586956521739131
----------------------------------------------------------------------
KNeighborsClassifier()

Accuracy score: 0.8315217391304348

"Confusion Matrix:
" [[295   5]
 [ 57  11]]
classification_report
               precision    recall  f1-score   support

           0       0.84      0.98      0.90       300
           1       0.69      0.16      0.26        68

    accuracy                           0.83       368
   macro avg       0.76      0.57      0.58       368
weighted avg       0.81      0.83      0.79       368

Average accuracy_score 0.8315217391304348
----------------------------------------------------------------------
RandomForestClassifier()

Accuracy score: 0.8342391304347826

"Confusion Matrix:
" [[296   4]
 [ 57  11]]
classification_report
               precision    recall  f1-score   support

           0       0.84      0.99      0.91       300
           1       0.73      0.16      0.27        68

    accuracy                           0.83       368
   macro avg       0.79      0.57      0.59       368
weighted avg       0.82      0.83      0.79       368

Average accuracy_score 0.8342391304347826
GradientBoostingClassifier()

Accuracy score: 0.8288043478260869

"Confusion Matrix:
" [[286  14]
 [ 49  19]]
classification_report
               precision    recall  f1-score   support

           0       0.85      0.95      0.90       300
           1       0.58      0.28      0.38        68

    accuracy                           0.83       368
   macro avg       0.71      0.62      0.64       368
weighted avg       0.80      0.83      0.80       368

Average accuracy_score 0.8288043478260869
```

Maximum accuracy Score is shown by LogisticRegression () 0.8586956521739131

Cross Validation:-

In order to avoid over fitting, Cross-validation is used to estimate the skill of a machine learning model on unseen data.

In [49]:
```
score1=[]
```

In [50]:
```
lr=LogisticRegression()
scores=cross_val_score(lr,x,y,cv=5)
score1.append(scores)
scores
```

Out[50]:
```
array([0.87755102, 0.86734694, 0.86054422, 0.8707483 , 0.87755102])
```

In [51]:
```
rf=RandomForestClassifier()
scores=cross_val_score(rf,x,y,cv=5)
score1.append(scores)
scores
```

Out[51]:
```
array([0.8537415 , 0.85714286, 0.86054422, 0.86054422, 0.85714286])
```

In [52]:
```
kn=KNeighborsClassifier()
scores=cross_val_score(kn,x,y,cv=5)
score1.append(scores)
scores
```

Out[52]:
```
array([0.83673469, 0.83673469, 0.8537415 , 0.85034014, 0.84693878])
```

In [53]:
```
gb=GradientBoostingClassifier()
scores=cross_val_score(gb,x,y,cv=5)
score1.append(scores)
scores
```

Out[53]:
```
array([0.84693878, 0.87414966, 0.87414966, 0.85714286, 0.86054422])
```

In [54]:
```
dt=DecisionTreeClassifier()
scores=cross_val_score(dt,x,y,cv=5)
score1.append(scores)
scores
```

Out[54]:
```
array([0.7755102 , 0.78911565, 0.82312925, 0.76870748, 0.77891156])
```

Difference of predicted model and crossvalidation score

In [55]:
```
models=[LogisticRegression(),RandomForestClassifier(),KNeighborsClassifier(),GradientBoostingClassifier(),DecisionTreeClassifier()]
for i in range(0,5):
    print(models[i],"difference is",scorel[i]-acclist[i])
```

```
LogisticRegression() difference is [0.01885537 0.00865129 0.00184857 0.01205265 0.01885537]
RandomForestClassifier() difference is [0.01950237 0.02290373 0.02630509 0.02630509 0.02290373]
KNeighborsClassifier() difference is [0.00521295 0.00521295 0.02221976 0.0188184  0.01541704]
GradientBoostingClassifier() difference is [0.01813443 0.04534531 0.04534531 0.02833851 0.03173987]
DecisionTreeClassifier() difference is [0.02279281 0.03639825 0.07041186 0.01599009 0.02619417]
```

from the observation KNeighborsClassifier() has least difference so I'm selecting KNeighborsClassifier()as best model

From the observation KNeighborsClassifier () has least difference so I'm selecting KNeighborsClassifier () as best model.

Hyper Tuning:

In [56]:
```
from sklearn.model_selection import GridSearchCV,KFold
params = {
    'n_neighbors' : [5,7,9,11,13,15],
               'weights' : ['uniform','distance'],
               'metric' : ['minkowski','euclidean','manhattan'],
                'p':[1,2],'leaf_size':list(range(1,20))

}

gs2 = GridSearchCV(KNeighborsClassifier(), params, verbose = 1, cv=3, n_jobs = -1)
gs2.fit(xtrain, ytrain)
print('Best param:', gs2.best_params_)
```

```
Fitting 3 folds for each of 1368 candidates, totalling 4104 fits
Best param: {'leaf_size': 1, 'metric': 'minkowski', 'n_neighbors': 5, 'p': 2, 'weights': 'uniform'}
```

Best parameters: {'leaf_size': 1, 'metric': 'minkowski', 'n_neighbors': 5, 'p': 1, 'weights': 'distance'}

Modelling using best parameter and best model:-

In [57]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=.25)
model =KNeighborsClassifier(metric='minkowski', n_neighbors=5,weights='distance',p=1,leaf_size=1)
model.fit(x_train,y_train)
model.score(x_test,y_test)
```

Out[57]:
```
0.8505434782608695
```

In [58]:
```
y_pred_1 = model.predict(x_test)
```

In [59]:
```
result = confusion_matrix(y_test, y_pred_1)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred_1)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred_1)
print("Accuracy:",result2)
```

```
Confusion Matrix:
[[306   1]
 [ 54   7]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      1.00      0.92       307
           1       0.88      0.11      0.20        61

    accuracy                           0.85       368
   macro avg       0.86      0.56      0.56       368
weighted avg       0.85      0.85      0.80       368

Accuracy: 0.8505434782608695
```

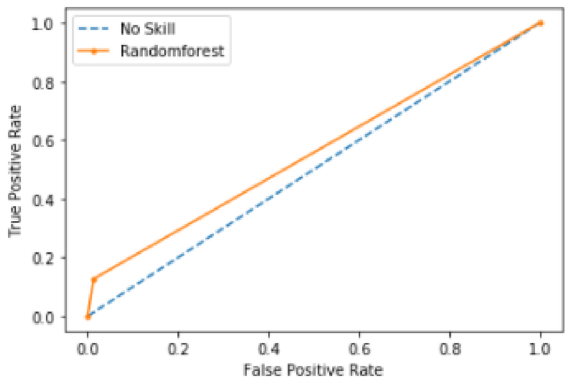
Final model after hyper tuning with accuracy 0.8586956521739131

Best model: KNeighbourClassifier Best param: {'leaf_size': 1, 'metric': 'minkowski', 'n_neighbors': 5, 'p': 1, 'weights': 'distance'}

ROC AUC CURVE:-

```
In [60]:
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.26,random_state=1)
m=KNeighborsClassifier(metric='minkowski', n_neighbors=5,weights='distance',p=1,leaf_size=1)
m.fit(xtrain,ytrain)
p=m.predict(xtest)
ns_probs = [0 for _ in range(len(ytest))]
m_probs = p
ns_auc = roc_auc_score(ytest, ns_probs)
m_auc = roc_auc_score(ytest, m_probs)
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('model: ROC AUC=%.3f' % (m_auc))
ns_fpr, ns_tpr,_= roc_curve(ytest, ns_probs)
m_fpr, m_tpr,_=roc_curve(ytest, m_probs)
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(m_fpr, m_tpr, marker='.', label='Randomforest')
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
pyplot.legend()
pyplot.show()
```

```
No Skill: ROC AUC=0.500
model: ROC AUC=0.557
```



Conclusion:

I have developed a model to predict attrition of an employee with 85.8% accuracy

Saving the model:-

```
In [61]:
from joblib import dump
dump(model, 'model_hr.joblib')

Out[61]: ['model_hr.joblib']

In [62]:
from joblib import load
loaded = load('model_hr.joblib')

In [ ]:
```
