VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Data Mining: Concepts and Techniques

CSE 3054

PROJECT REVIEW REPORT

Submitted to

Prof. Arup Ghosh

Topic - Predicting Attrition of Valuable Employees Using Data Mining Techniques

Slot: B1 + TB1

Team Members:

Mokshda Gangrade - 19BDS0067
Lakshit Kothari - 19BDS0077
Pranjal Gupta - 19BDS0081

Introduction -

Human Resources Management (HRM) has become one of the essential interests of managers and decision-makers in almost all types of businesses to adopt plans for correctly discovering highly qualified employees. Employee attrition refers to the percentage of workers who leave an organization and are replaced by new employees. A high rate of attrition in an organization leads to increased recruitment, hiring, and training costs. Not only it is costly, but qualified and competent replacements are hard to find. In most industries, the top 20% of people produce about 50% of the output.

From here, the interest in the data mining role has been growing. Its objective is the discovery of knowledge from huge amounts of data. In this project, Data Mining techniques were utilized to build a classification model for predicting attrition of valuable employees using a real dataset collected from IBM HR Analytics through a survey of 1470 employees.

We will be taking HR data and using machine learning models to predict what employees will be more likely to leave given some attributes. Such a model would help an organization predict employee attrition and define a strategy to reduce such costly problems.

Objective

The input dataset is an Excel file with information about 1470 employees. For each employee, in addition to whether the employee left or not (attrition), there are attributes/features such as age, employee role, daily rate, job satisfaction, years at the company, years in a current role, etc.
The steps we will go through are:

1. Data preprocessing
2. Data analysis
3. Model training

4. Model validation
5. Model predictions
6. Visualization of results

For the prevention of employee attrition, we will apply some well-known classification methods, that is, Decision tree, Logistic Regression, KNN, Random Forest on the human resource data. For this, we implement a feature selection method on the data and analyze the results to prevent employee attrition.

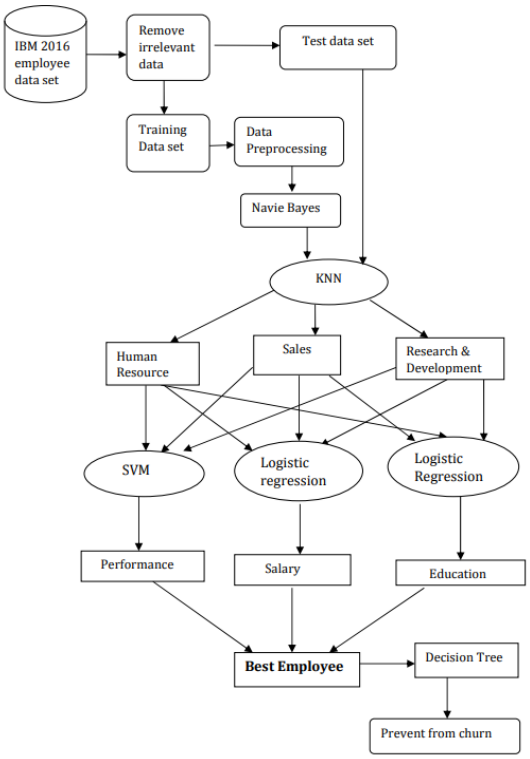
Literature Review

1. Prediction of Employee Attrition using Data Mining -

(IEEE International Conference on System, Computation, Automation & Networking 2018)

The paper analyzed the past and existing employee information to estimate and study the future attritioners and study the reasons for employee turnover. The results of this paper describe that data extraction algorithms can be utilized to construct reliable and accurate predictive methods for employee attrition. The issue of attrition identification is not just to depict attritioners from no attritioners. The paper used tentative data to study and data extraction methods to depict the attrition probability for each one employee and provide them a score to build the retention techniques.

SYSTEM ARCHITECTURE: The below figure shows the architectural diagram representing the overall system framework. In this system architecture consists of employee dataset and different data mining techniques and data preprocessing techniques. Here we maintain two data sets in our database. They are Test datasets and Training datasets, In Training datasets we can use pre-processing technique form given chosen data set in organization. Naive Bayes, KNN, SVM, Logistic Regression and decision algorithms are used to generate a Best employee in an organization. By using this we clean and classify the employee data set into different departments like human resource, sales and research and development.



2. **Early Prediction of Employee Attrition** - ***Ijstr March 2020***
   In the paper, very simple models for predicting employee attrition have been built and compared for accuracy in its predictions. The fundamental general explanation for attrition discovered in all likelihood the effort-reward imbalance. The paper also discovered that various features of work-life balance that may speak to an issue for the HRs and the way in which each of the factors is connected (straightforwardly or in a roundabout way) to factors of work-life balance like - (distance from home, business travel, and work-life balance, etc) showed an indication of the problems that needs to be addressed.

3. **Prediction of Employee Attrition using Data Mining -**
   Gangammagari Mohan Babu, Dr. Mooramreddy Sreedevi
   Two grouping strategies were used to create models for anticipating employee attrition. Fake Neural Network (ANN) model anticipated the employee attrition all the more accurately (85.33%) than Decision Tree (C&R Tree) model (80.89%). The two models, be that as it may, decided a very long time at the organization and staying at work longer than required as the most significant factors impacting employee attrition.

4. **Employee Attrition Prediction using Machine Learning Techniques -**
   ISSN NO: 0886-9367

This study concludes that to reduce attrition industries should create some opportunities for the growth of their employees within the organization by adopting new Innovative Technologies and Effective training programs. Employee satisfaction is essential to any effective employee retention strategy - any good HR manager knows that. Thus, detailed comparisons and analyses of different methodologies have been made pertaining to the employee attrition problem.

5. **Employee Attrition and Retention in a Cut-throat competitive environment in India : A holistic approach** ***- Mrs. Jaya Sharma***
   This research concluded one of the most important reasons for attrition is Compensation because plenty of opportunities are there in the market for experienced, well-qualified employees if they switch over to other companies and they will pay more. There are much push, pull and personal factors are involved and initiating the thought of turnover among employees. Organizations should be alert and frame some necessary strategies to reduce attrition so that they can reduce the expenditure of employees for recruitment, training, and development.

Algorithms Discussed

There were 4 main techniques that we used and compared. The algorithms are:

- **Logistic Regression:** Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval, or ratio-level independent variables.

  On the basis of the categories, Logistic Regression can be classified into three types:

  - **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
  - **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
  - **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

  Based on our data and requirements, we used multinomial logistic regression for this project.

- **Decision Trees:** A decision tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
  In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset.

  It is nothing but a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a decision tree because similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

- **K-nearest neighbors:** K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on the Supervised Learning technique. K-NN algorithm assumes the similarity between the new case/data and available cases and puts the new case into the category that is most similar to the available categories.

  K-NN algorithm, at the training phase, stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well-suited category by using K-NN algorithm. K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for Classification problems.

  K-NN is a non-parametric algorithm, which means it does not make any assumptions on underlying data. It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

- **Random Forest:** Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

  As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying

on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

Proposed Methodology

Our main aim was to predict the attrition of valuable employees, using the dataset provided by IBM. We will be taking HR data and using machine learning models to predict what employees will be more likely to leave given some attributes. Such a model would help an organization predict employee attrition and define a strategy to reduce such costly problems.

The proposed steps in building the model are:

1. **Preprocessing:** Here, we find the unique values by removing duplicates and NA values from the data. We then implement one-hot encoding to get results in the form of binary numbers: 1(Yes) and 0(No). We then drop the columns which only have the same values in all rows to make our dataset more accurate.

2. **Modeling:** In this section, we try 4 machine learning models to find the one that gives us the most accuracy. The models we use here are:

- Logistic Regression
- Decision Tree Classifier
- K-Nearest Neighbor
- Random Forest Classifier

The proposed steps in this are:

a. First, we split our data into training and test-set in a 70:30 ratio.
b. Now we create our 4 models mentioned above.

```
logreg = LogisticRegression()
tree = DecisionTreeClassifier(random_state = 3131)
knn = KNeighborsClassifier()
rf = RandomForestClassifier(random_state = 3131)
```

c. Now we perform cross validation on our models which is a resampling procedure used to evaluate machine learning models on a limited data sample, and generate a mean score, standard score, and recall score.
d. From here, we now evaluate the recall score which means the ratio of the number of true positives divided by the total number of true positives and false negatives.
e. On the basis of the recall score, we choose the model which has the highest recall score.

3. **Handling Imbalance:** Here we implement undersampling and oversampling methods to handle any imbalance in the dataset. First, undersampling will give us a similar table with all the respective scores, and we will compare it with the scores we had gotten before. If in this case, we get better scores, we choose the model with highest recall score and evaluate using an oversampling model. The model with the best recall score overall is now chosen.

4. **HyperParam Tuning:** Here we choose a set of optimal hyperparameters to further enhance our model. We then compare our before and after results and check if our score is increasing or decreasing and continue accordingly.

5. **EDA :** Here we perform Exploratory Data Analysis on the dataset to find out the hidden relationships and dependencies of the variables on one another and also on the target variable - Attrition. We find the correlation matrix to check the overall relationships(+ve, -ve or 0) between all the variables. We use visualizations to find out the deeper relationships between variables and our target variable.

6. **Prediction:** After the best model has been chosen, giving a relatively higher score, we evaluate our test data set by using our selected model. We then get the prediction of all test set employees in a binary format (1-Resign, 0-Stay). After which, we compare the results of our training set accuracy and testing set accuracy.

Result and Conclusion

As we saw in the above methodology, we choose the model for the prediction out of the four based on the highest accuracy after sampling and training it through various processes. First, we perform Cross validation on the models to evaluate the models on limited data size and decide the best model based on the recall score of each.

	method	mean score	std score	recall score
0	Logistic Regression	0.041711	0.044120	0.028169
1	Decision Tree Classifier	0.367736	0.076253	0.267606
2	KNN Classifier	0.168449	0.055310	0.112676
3	Random Forest Classifier	0.108556	0.031235	0.126761

As we can see that the Decision tree has the highest recall score so we choose that

model for this moment, but the score is not good enough for prediction and shows that there is an imbalance in the data, hence we further move on to balancing the data using UnderSampling and OverSampling.

	method	mean score	std score	recall score
0	Logistic Regression UnderSampling	0.674866	0.113426	0.619718
1	Decision Tree Classifier UnderSampling	0.638146	0.080455	0.647887
2	KNN Classifier UnderSampling	0.602317	0.094931	0.619718
3	Random Forest Classifier UnderSampling	0.668093	0.114557	0.619718

	method	mean score	std score	recall score
0	Logistic Regression OverSampling	0.632086	0.118270	0.661972
1	Decision Tree Classifier OverSampling	0.307487	0.059830	0.323944
2	KNN Classifier OverSampling	0.536364	0.063275	0.450704
3	Random Forest Classifier OverSampling	0.186631	0.051612	0.183099

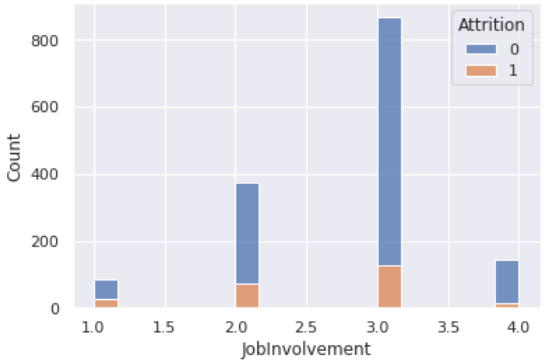
As we can see from all the above 3 methods - Cross-Validation, Under Sampling, and Over Sampling, Logistic Regression is the most stable model in all three. Hence we move on to the HyperParameter Tuning process. After HyperParameter Tuning process, the score is getting higher, it means that tuning process has improved the model for Logistic Regression.

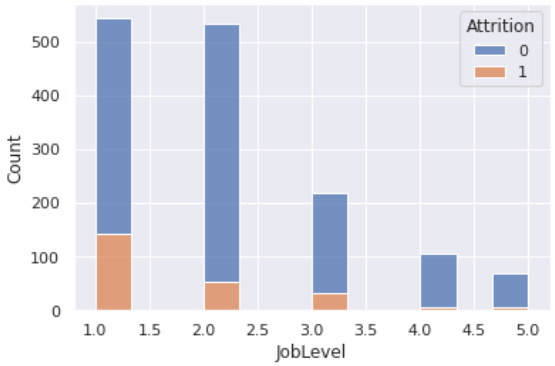
	method	score
0	Logistic Regression OverSampling Before Tuning	0.661972
1	Logistic Regression OverSampling After Tuning	0.690141

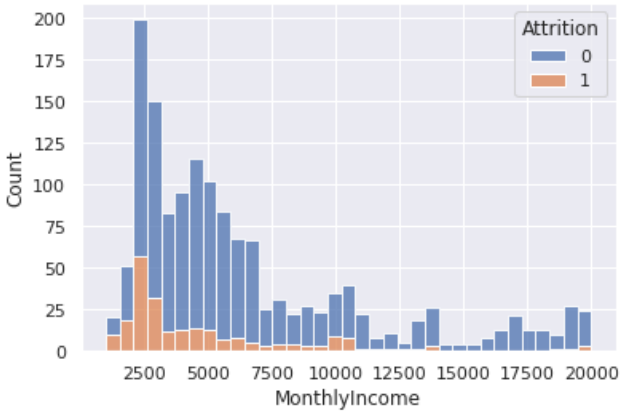
Once we have established Logistic Regression as the best model for our dataset, we can now move on to the next part of the process i.e. **Exploratory Data Analysis** and Prediction. We start by establishing re;ationships between our variables to have better understanding of the data and to find out the hidden relationships and dependencies of the variables on one another and also on the target variable - Attrition.

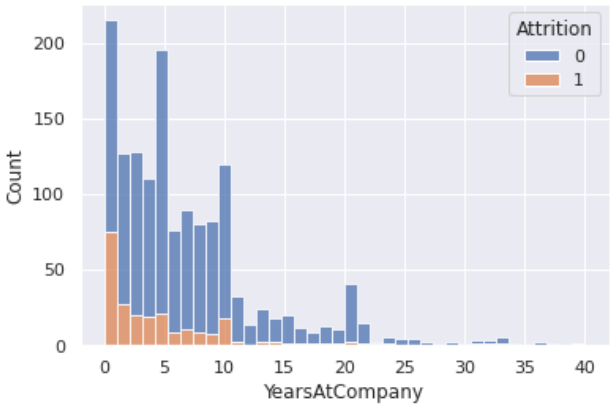
```
Attrition                  1.000000
DailyRate                 -0.056652
DistanceFromHome           0.077924
Education                 -0.031373
EmployeeNumber            -0.010577
EnvironmentSatisfaction   -0.103369
HourlyRate                -0.006846
JobInvolvement            -0.130016
JobLevel                  -0.169105
JobSatisfaction           -0.103481
MonthlyIncome             -0.159840
MonthlyRate                0.015170
NumCompaniesWorked         0.043494
PercentSalaryHike         -0.013478
PerformanceRating          0.002889
RelationshipSatisfaction  -0.045872
StockOptionLevel          -0.137145
TotalWorkingYears         -0.171063
TrainingTimesLastYear     -0.059478
WorkLifeBalance           -0.063939
YearsAtCompany            -0.134392
YearsInCurrentRole        -0.160545
YearsSinceLastPromotion   -0.033019
YearsWithCurrManager      -0.156199
Name: Attrition, dtype: float64
```

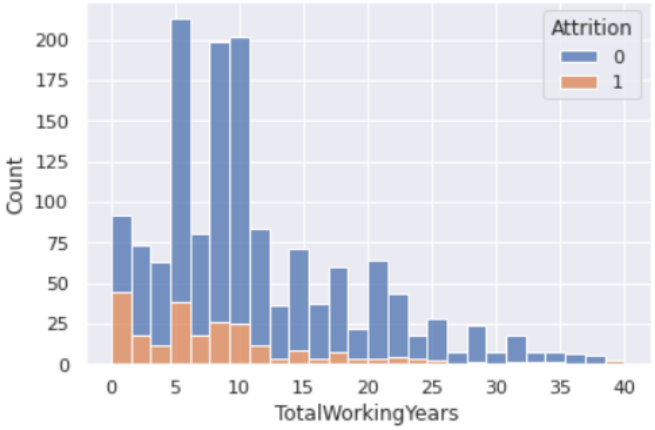
This output represents the correlation of attribute attrition with all the other attributes in the dataset. The highest effects on attrition are observed with the attributes “job involvement”, “job level”, “monthly income”, “total working years” and “years in current role”.











The graphs above show the relation between attrition and the 5 key variables mentioned above. It can be seen that with changing values of variables, the value of attrition is also changing.

Job involvement of employees has an inverse effect on attrition. Higher the job levels, lesser is the employee attrition .

Employees at higher job levels tend to leave the company less often than those at lower levels.

Effect of monthly income on attrition is something we have seen around us as well. People with lower income tend to leave the company to some other companies or other job roles in order to get higher pay.

Increasing time spent in the company leads to lower attrition levels. People who have spent a lot of time in the company tend not to leave the company and continue to work there itself. The total work experience or total working years of an employee also has the same effect has years spent in a particular company. The prime reason

for it is, higher the experience, better the position in the company, hence changing the company is not preferred by many.

Prediction Results -

```
  --
TESTING RESULTS:
================================
CONFUSION MATRIX:
[[360  20]
 [ 38  23]]
ACCURACY SCORE:
0.8685
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.904523    0.534884  0.868481    0.719703      0.853393
recall       0.947368    0.377049  0.868481    0.662209      0.868481
f1-score     0.925450    0.442308  0.868481    0.683879      0.858621
support    380.000000   61.000000  0.868481  441.000000    441.000000
```

From this confusion matrix, it is pretty clear that the model is performing very well. The false-positive and false-negative are 20 and 23 respectively. The accuracy of the model is above 86%.

Appendix

The code of the project is available in the following google collab link :

[Google collab link]

[Dataset]

Code:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from imblearn.under_sampling import RandomUnderSampler
```

```
from imblearn.over_sampling import RandomOverSampler

from sklearn.pipeline import Pipeline
from imblearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler
from sklearn.compose import ColumnTransformer

from sklearn.model_selection import train_test_split, StratifiedKFold, cross_val_score,
RandomizedSearchCV


from sklearn.metrics import recall_score
att = pd.read_csv('WA_Fn-UseC_-HR-Employee-Attrition.csv')
att.head()
att.info()
attFeatures = []
for i in att.columns:
    attFeatures.append([i, att[i].nunique(), att[i].drop_duplicates().values])
pd.DataFrame(attFeatures, columns = ['Features', 'Unique Number', 'Values'])
att['Attrition'] = np.where(att['Attrition'] == 'Yes', 1, 0)
att.drop(columns=['EmployeeCount', 'Over18', 'StandardHours'], inplace=True)
transformer = ColumnTransformer([
    ('one hot', OneHotEncoder(drop = 'first'), ['BusinessTravel', 'Department', 'EducationField',
'Gender',
                                                'JobRole', 'MaritalStatus', 'OverTime']),
], remainder = 'passthrough')
att['Attrition'].value_counts()/att.shape[0]*100
att.head()
X = att.drop('Attrition', axis = 1)
y = att['Attrition']

X.shape
logreg = LogisticRegression()
tree = DecisionTreeClassifier(random_state = 3131)
knn = KNeighborsClassifier()
rf = RandomForestClassifier(random_state = 3131)
logreg_pipe = Pipeline([('transformer', transformer), ('logreg', logreg)])
tree_pipe = Pipeline([('transformer', transformer), ('tree', tree)])
knn_pipe = Pipeline([('transformer', transformer), ('scale', MinMaxScaler()), ('knn', knn)])
rf_pipe = Pipeline([('transformer', transformer), ('rf', rf)])

def model_evaluation(model, metric):
    skfold = StratifiedKFold(n_splits = 5)
    model_cv = cross_val_score(model, X_train, y_train, cv = skfold, scoring = metric)
    return model_cv
```

```
logreg_pipe_cv = model_evaluation(logreg_pipe, 'recall')
tree_pipe_cv = model_evaluation(tree_pipe, 'recall')
knn_pipe_cv = model_evaluation(knn_pipe, 'recall')
rf_pipe_cv = model_evaluation(rf_pipe, 'recall')

for model in [logreg_pipe, tree_pipe, knn_pipe, rf_pipe]:
    model.fit(X_train, y_train)

score_mean = [logreg_pipe_cv.mean(), tree_pipe_cv.mean(), knn_pipe_cv.mean(),
rf_pipe_cv.mean()]
score_std = [logreg_pipe_cv.std(), tree_pipe_cv.std(), knn_pipe_cv.std(), rf_pipe_cv.std()]
score_recall_score = [recall_score(y_test, logreg_pipe.predict(X_test)),
            recall_score(y_test, tree_pipe.predict(X_test)),
            recall_score(y_test, knn_pipe.predict(X_test)),
            recall_score(y_test, rf_pipe.predict(X_test))]
method_name = ['Logistic Regression', 'Decision Tree Classifier', 'KNN Classifier', 'Random
Forest Classifier']
cv_summary = pd.DataFrame({
    'method': method_name,
    'mean score': score_mean,
    'std score': score_std,
    'recall score': score_recall_score
})
cv_summary
rus = RandomUnderSampler(random_state = 3131)
X_under, y_under = rus.fit_resample(X_train, y_train)

logreg_pipe_under = Pipeline([('transformer', transformer), ('rus', rus), ('logreg', logreg)])
tree_pipe_under = Pipeline([('transformer', transformer), ('rus', rus), ('tree', tree)])
knn_pipe_under = Pipeline([('transformer', transformer), ('scale', MinMaxScaler()), ('rus', rus),
('knn', knn)])
rf_pipe_under = Pipeline([('transformer', transformer), ('rus', rus), ('rf', rf)])

def model_evaluation(model, metric):
    skfold = StratifiedKFold(n_splits = 5)
    model_cv = cross_val_score(model, X_train, y_train, cv = skfold, scoring = metric)
    return model_cv
logreg_under_cv = model_evaluation(logreg_pipe_under, 'recall')
tree_under_cv = model_evaluation(tree_pipe_under, 'recall')
knn_under_cv = model_evaluation(knn_pipe_under, 'recall')
rf_under_cv = model_evaluation(rf_pipe_under, 'recall')
```

```
for model in [logreg_pipe_under, tree_pipe_under, knn_pipe_under, rf_pipe_under]:
    model.fit(X_train, y_train)

score_mean = [logreg_under_cv.mean(), tree_under_cv.mean(), knn_under_cv.mean(),
              rf_under_cv.mean()]
score_std = [logreg_under_cv.std(), tree_under_cv.std(), knn_under_cv.std(),
             rf_under_cv.std()]
score_recall_score = [recall_score(y_test, logreg_pipe_under.predict(X_test)),
            recall_score(y_test, tree_pipe_under.predict(X_test)),
            recall_score(y_test, knn_pipe_under.predict(X_test)),
            recall_score(y_test, rf_pipe_under.predict(X_test))]
method_name = ['Logistic Regression UnderSampling', 'Decision Tree Classifier
UnderSampling',
               'KNN Classifier UnderSampling', 'Random Forest Classifier UnderSampling']
under_summary = pd.DataFrame({
    'method': method_name,
    'mean score': score_mean,
    'std score': score_std,
    'recall score': score_recall_score
})
under_summary
ros = RandomOverSampler(random_state = 3131)
X_over, y_over = ros.fit_resample(X_train, y_train)

logreg_pipe_over = Pipeline([('transformer', transformer), ('ros', ros), ('logreg', logreg)])
tree_pipe_over = Pipeline([('transformer', transformer), ('ros', ros), ('tree', tree)])
knn_pipe_over = Pipeline([('transformer', transformer), ('scale', MinMaxScaler()), ('ros', ros),
('knn', knn)])
rf_pipe_over = Pipeline([('transformer', transformer), ('ros', ros), ('rf', rf)])

def model_evaluation(model, metric):
    skfold = StratifiedKFold(n_splits = 5)
    model_cv = cross_val_score(model, X_train, y_train, cv = skfold, scoring = metric)
    return model_cv

logreg_over_cv = model_evaluation(logreg_pipe_over, 'recall')
tree_over_cv = model_evaluation(tree_pipe_over, 'recall')
knn_over_cv = model_evaluation(knn_pipe_over, 'recall')
rf_over_cv = model_evaluation(rf_pipe_over, 'recall')
for model in [logreg_pipe_over, tree_pipe_over, knn_pipe_over, rf_pipe_over]:
    model.fit(X_train, y_train)
```

```
score_mean = [logreg_over_cv.mean(), tree_over_cv.mean(), knn_over_cv.mean(),
              rf_over_cv.mean()]
score_std = [logreg_over_cv.std(), tree_over_cv.std(), knn_over_cv.std(),
             rf_over_cv.std()]
score_recall_score = [recall_score(y_test, logreg_pipe_over.predict(X_test)),
             recall_score(y_test, tree_pipe_over.predict(X_test)),
             recall_score(y_test, knn_pipe_over.predict(X_test)),
             recall_score(y_test, rf_pipe_over.predict(X_test))]
method_name = ['Logistic Regression OverSampling', 'Decision Tree Classifier OverSampling',
               'KNN Classifier OverSampling', 'Random Forest Classifier OverSampling']
over_summary = pd.DataFrame({
    'method': method_name,
    'mean score': score_mean,
    'std score': score_std,
    'recall score': score_recall_score
})
over_summary
estimator = Pipeline([
    ('transformer', transformer),
    ('ros', ros),
    ('model', logreg)
])

hyperparam_space = {
    'model__C': [100, 10, 1, 0.1, 0.01, 0.001],
    'model__solver': ['liblinear', 'newton-cg', 'lbfgs'],
    'model__max_iter': [50, 100, 150, 200],
    'model__random_state': [3131]
}

random = RandomizedSearchCV(
                estimator,
                param_distributions = hyperparam_space,
                cv = StratifiedKFold(n_splits = 5),
                scoring = 'recall',
                n_iter = 10,
                n_jobs = -1)

random.fit(X_train, y_train)

print('best score: ', random.best_score_)
```

```
print('best param: ', random.best_params_)
estimator.fit(X_train, y_train)
y_pred_estimator = estimator.predict(X_test)
recall_estimator = recall_score(y_test, y_pred_estimator)

random.best_estimator_.fit(X_train, y_train)
y_pred_random = random.best_estimator_.predict(X_test)
recall_best_estimator = recall_score(y_test, y_pred_random)

score_list = [recall_estimator, recall_best_estimator]
method_name = ['Logistic Regression OverSampling Before Tuning', 'Logistic Regression
OverSampling After Tuning']
best_summary = pd.DataFrame({
    'method': method_name,
    'score': score_list
})
best_summary
import seaborn as sns
att.head()
att.info()
att.isnull().sum()
att.describe()
for i in att:
  print("column",i," - ",att[i].nunique())
att.head()
att[att.columns[1:]].corr()['Attrition'][:]
sns.set(font_scale=2.4)
plt.figure(figsize=(40, 40))
sns.heatmap(att.corr(), annot=True, cmap="coolwarm", annot_kws={'size': 15})
sns.set(font_scale=1.0)
sns.histplot(data=att, x="Age", hue="Attrition", multiple="stack", bins=35)
#sns.histplot(data=att, x="Age", hue="Attrition", multiple="stack")
#plt.figure(figsize=[15,15])
ax = sns.countplot(data=att, x="BusinessTravel", hue="Attrition")
# to display the count on each bar
for p in ax.patches:
    ax.annotate(format(p.get_height()),
                (p.get_x() + p.get_width() / 2., p.get_height()+20),
                ha = 'center', va = 'center')
plt.show()
sns.histplot(data=att, x="DailyRate", hue="Attrition", multiple="stack", bins=35)
plt.figure(figsize=[7,7])
```

```
ax = sns.countplot(data=att, x="Department", hue="Attrition")
# to display the count on each bar
for p in ax.patches:
    ax.annotate(format(p.get_height()),
                (p.get_x() + p.get_width() / 2., p.get_height()+20),
                ha = 'center', va = 'center')
plt.show()
sns.histplot(data=att, x="DistanceFromHome", hue="Attrition", multiple="stack", bins=25)
sns.histplot(data=att, x="Education", hue="Attrition", multiple="stack")
plt.figure(figsize=[12,7])
ax = sns.countplot(data=att, x="EducationField", hue="Attrition")
# to display the count on each bar
for p in ax.patches:
    ax.annotate(format(p.get_height()),
                (p.get_x() + p.get_width() / 2., p.get_height()+10),
                ha = 'center', va = 'center')
plt.show()
sns.histplot(data=att, x="EnvironmentSatisfaction", hue="Attrition", multiple="stack")
#plt.figure(figsize=[12,7])
sns.set(font_scale=0.8)
ax = sns.countplot(data=att, x="Gender", hue="Attrition")
# to display the count on each bar
for p in ax.patches:
    ax.annotate(format(p.get_height()),
                (p.get_x() + p.get_width() / 2., p.get_height()+10),
                ha = 'center', va = 'center')
plt.show()
sns.set(font_scale=1.0)
sns.histplot(data=att, x="HourlyRate", hue="Attrition", multiple="stack")
sns.histplot(data=att, x="JobInvolvement", hue="Attrition", multiple="stack")
sns.histplot(data=att, x="JobLevel", hue="Attrition", multiple="stack")
sns.histplot(data=att, x="JobSatisfaction", hue="Attrition", multiple="stack")
plt.figure(figsize=[23,5])
ax = sns.countplot(data=att, x="JobRole", hue="Attrition")
# to display the count on each bar
for p in ax.patches:
    ax.annotate(format(p.get_height()),
                (p.get_x() + p.get_width() / 2., p.get_height()+10),
                ha = 'center', va = 'center')
plt.show()
sns.histplot(data=att, x="MonthlyIncome", hue="Attrition", multiple="stack",bins=35)
sns.histplot(data=att, x="NumCompaniesWorked", hue="Attrition", multiple="stack")
```

```
sns.histplot(data=att, x="PercentSalaryHike", hue="Attrition", multiple="stack")
sns.histplot(data=att, x="RelationshipSatisfaction", hue="Attrition", multiple="stack")
sns.histplot(data=att, x="PerformanceRating", hue="Attrition", multiple="stack")
sns.histplot(data=att, x="TotalWorkingYears", hue="Attrition", multiple="stack")
sns.histplot(data=att, x="WorkLifeBalance", hue="Attrition", multiple="stack")
sns.histplot(data=att, x="YearsAtCompany", hue="Attrition", multiple="stack")
sns.histplot(data=att, x="OverTime", hue="Attrition", multiple="stack")
att = pd.get_dummies(att,
prefix=['BusinessTravel','Department','EducationField','Gender','JobRole','MaritalStatus'],columns
=['BusinessTravel','Department','EducationField','Gender','JobRole','MaritalStatus'])
att = pd.get_dummies(att, prefix=['OverTime'], columns=['OverTime'])
att.shape
att.head()
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = att.drop('Attrition', axis=1)
y = att.Attrition

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)
X_std = scaler.transform(X)
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report,
roc_auc_score

def evaluate(model, X_train, X_test, y_train, y_test):
    y_test_pred = model.predict(X_test)
    y_train_pred = model.predict(X_train)


    print("Prediction Result: 1 means Resign, 0 means Stay\nSo for every test set employee, 0/1
represents whether they will stay\leave the
company\n============================================================")

    print(y_test_pred)

    print("\n\nTRAINIG RESULTS: \n===============================")
    clf_report = pd.DataFrame(classification_report(y_train, y_train_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_train, y_train_pred)}")
```

```
    print(f"ACCURACY SCORE:\n{accuracy_score(y_train, y_train_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")

    print("TESTING RESULTS: \n================================")
    clf_report = pd.DataFrame(classification_report(y_test, y_test_pred, output_dict=True))
    print(f"CONFUSION MATRIX:\n{confusion_matrix(y_test, y_test_pred)}")
    print(f"ACCURACY SCORE:\n{accuracy_score(y_test, y_test_pred):.4f}")
    print(f"CLASSIFICATION REPORT:\n{clf_report}")
from sklearn.linear_model import LogisticRegression

lr_clf = LogisticRegression(solver='liblinear', penalty='l1')
lr_clf.fit(X_train_std, y_train)

evaluate(lr_clf, X_train_std, X_test_std, y_train, y_test)
```