MSBA7021 GROUP PROJECT

Employee Attrition Analytics:

Turnover Rate Management with Optimal Cost

Cathy Yuan, Judy Wu, Yuri Zhang

Table of Contents

1. Introduction

Employee Turnover Management is becoming an important topic in Human Resource Management (HRM), which refers to how frequently employees leave a business in a certain period. For most cases, the occupations of the leavers need to be filled with new employees, which causes extra hiring cost and potential loss in business.

According to Cameron Keng (2014), the unsatisfied salary level is still the major reason of quitting the job since the average raise employees can expect is 3% every year, while the average raise they will receive for leaving is between 10% to 20%. Therefore, to keep the turnover rate in a fair level, one of the most effective ways is to raise the salary of the potential leavers.

To achieve this goal, the first step is to precisely detect the potential leavers. As it is a binary classification problem (whether quit or not), we apply logistics regression and random forest classification to predict the potential leavers with different thresholds (0.1 - 0.9). The model with highest f1 score and most reasonable importance features will be chosen and used in optimization.

After detecting the potential leavers, our target is to minimize the HRM cost of maintaining an acceptable turnover rate. In our scenario analysis, we compare the fixed cost of hiring a new employee with the cost of raising 10%,15% and 20% annual base salary of the leavers and choose the smaller one to minimize the cost. We formulate different optimization models to fulfill different demand of the company: (1) global minimize with no constraint on turnover rate; (2) keep overall turnover rate $\leq$10%; (3) keep middle level employees' turnover rate $\leq$15%.

From the scenario analysis of all optimization models, the conclusions are as follows. (1) If raising 10% annual base salary, the optimal solutions always achieve the minimum total cost of $865,000 and the lowest overall turnover rate of 10%. (2) When the level of annual base salary raises increases, both the total cost and the overall turnover rate increases; (3) The best optimal solution under 10% salary raise scenario reduces total cost by 37.94% compared to the benchmark of all hiring new employees, and reduces total cost by 46.61% compared to the benchmark of all raising 10% base salary.

2. Literature Review

2.1. Cost of Hiring a New Employee and Keeping a Leaving Employee

The cost of hiring a new employee is expensive, which accounts for HR Manager 30 hours effort, 1-month job ad on job boards, basic background check for job candidates, IT equipment, training courses, employee support with 20 hours effort and new employee signing bonus (Elloitt, 2022). The detailed cost of an example is as follows.

Cost Breakdown Components	Cost
HR Manager 30 hours effort	$7,700
1-month job ad on Monster	$249
Basic background check	$50
IT equipment	$600
Training courses	$400
Employee support 20 hours effort	$6,000
New employee signing bonus	$2,000
	$16,999

Besides, to prevent an employee from leaving the current business, the company should raise the base salary that will be given by new business, at 10% - 20% (Keng, 2014).

The current study makes use of the findings on previous study and further apply machine learning and operation research method to solve this HR analytics problem.

3. Data Analytics and Prediction

3.1 Dataset

Here is a sample data of an imaginary organization from Kaggle. It can be a tech analytics startup. It contains 1000 records of its current employees and covers messages including academic background, hierarchy level, fixed salary with whether leaving the company or not as the predictors.

While the occupational group with the highest turnover rate in the UK is only 31% (Verlinden, 2019), the turnover rate of this company reaches 34.6%, which illustrates extremely poor HRM, and more effort is needed to reduce the turnover rate with minimal cost.

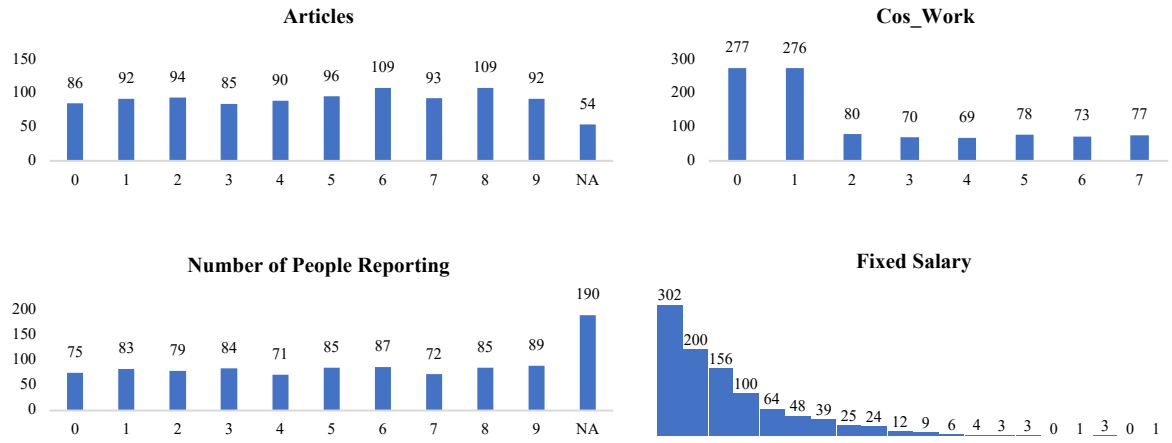
3.2 Exploratory Data Analytics (EDA) and Data Pre-processing

3.2.1 Exploratory Data Analytics (EDA)

(1) Univariate Analysis

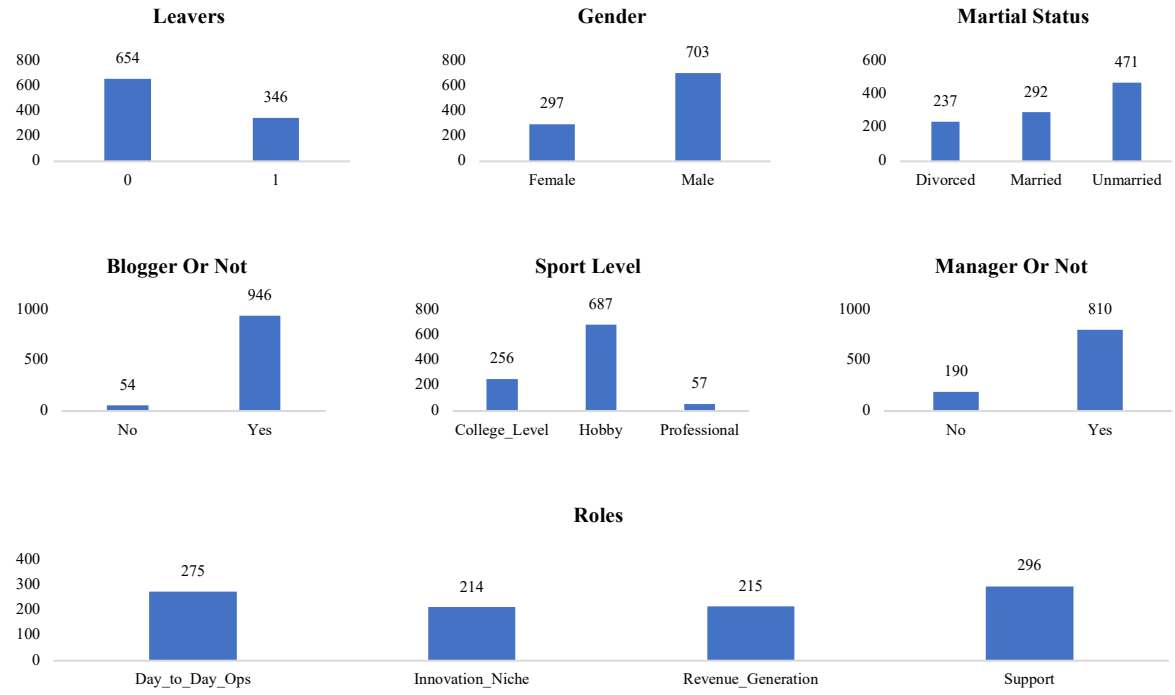
We plot the distribution of each variable.

(a) Numerical Variables

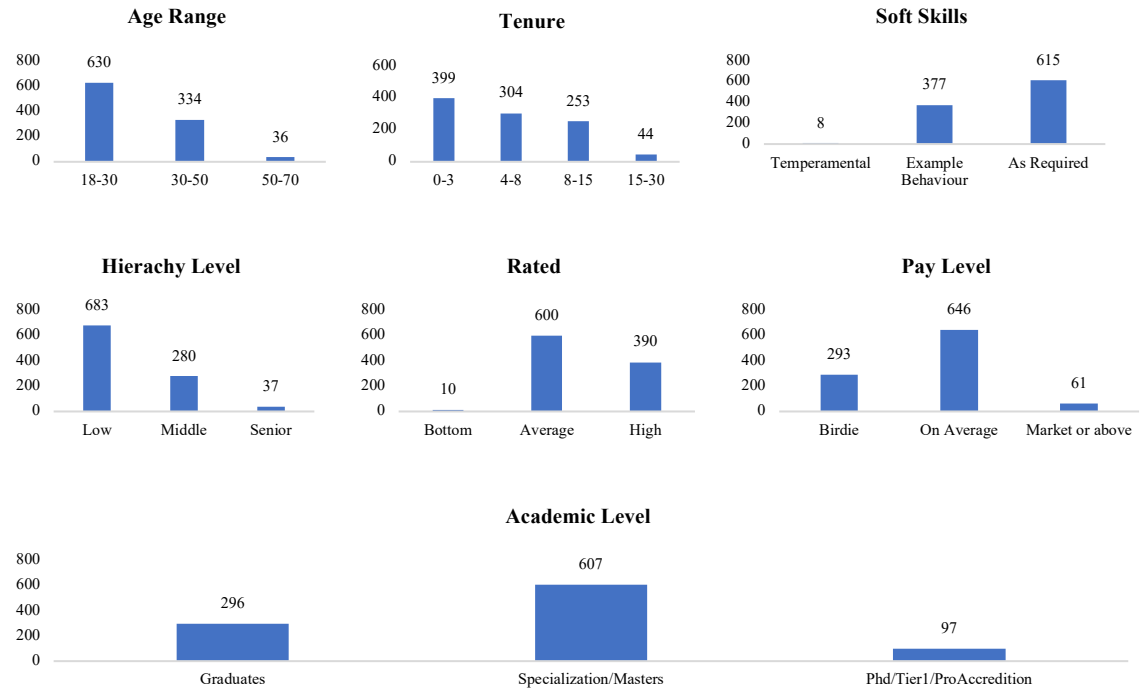


First, there are some null values for the variables, Articles and Number of People Reporting. We further check the relevant variables, Blogger Or Not and Manager Or Not. It is found that employees that are not bloggers have do not write any articles and that are not manager do not have people reporting to them. Hence, we decide to fill these variables with 0. Besides, the variable, Fixed Salary, is skewed and therefore we plan to take logarithm of this variable.

(b) Nominal Variables



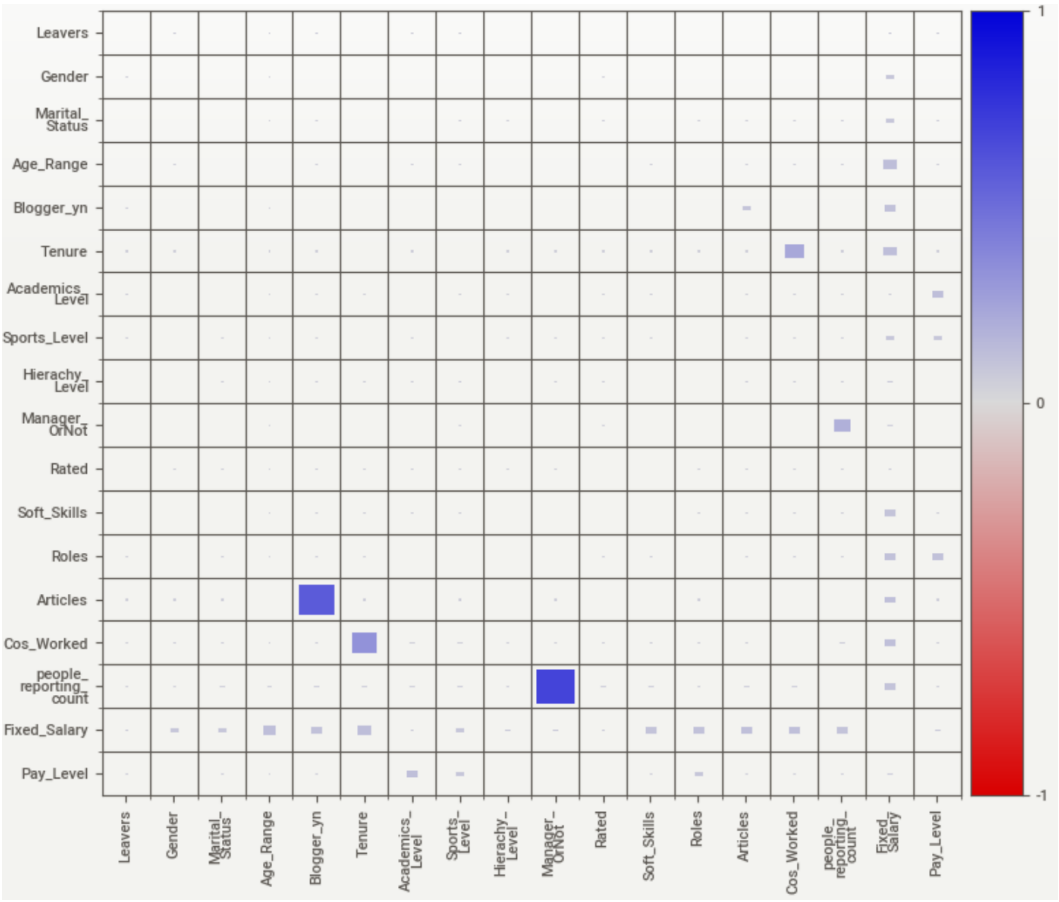
(c) Ordinal Variables



For nominal and ordinal variables, there is not serious issue with these variables.

(2) Association Analysis

We further apply association analysis among all variables. For numerical variables, we calculate Pearson Correlation. For categorical variables, we calculate Cramer's V. We plot pairwise association plot.



We found that there are some associations between "Articles" and "Blogger_yn" (0.56) and between "Manager_OrNot" and "people_reporting_count" (0.68). Since the association is not large and there are not many inputs in this dataset, we do not delete the correlated variables.

3.2.2 Data Pre-processing

(1) Fill the Null Values

(a) For input "people_reporting_count", when the employee is not a manager ("Manager_OrNot" =No), there is no one to report to them so "people_reporting_count"= 0. Therefore, we fill the null values with 0.

(b) "Articles": When the employee is not a blogger ("Blogger_yn"='No'), they will write 0 articles. Therefore, we fill the null values with 0.

(2) Rescaling: Transform the Salary to log(Salary)

As the variable "Fixed_Salary" is skewed that its effect on the prediction indicator may be overestimated. Thus, we take logarithm and transform this variable to "log(Fixed_Salary)".

(3) Label Encoding for the Ordinal Categorical Variables

We apply label encoding on the following ordinal categorical variables so as to guarantee that their ordinal features will be captured by the model.

(a) Age_Range: {'18-30':0,'30-50':1,'50-70':2}

(b) Tenure: {'0-3':0,'4-8':1,'8-15':2,'15-30':3}

(c) Academics_Level:{'Graduates':0,'Specialization/Masters':1,'Phd/Tier1/ProAccredition':2}

(d) Hierachy_Level: {'Low':0,'Middle':1,'Senior':2};

(e) Rated: {'Bottom':0,'Average':1,'High':2}

(f) Soft_Skills: {'Temperamental':0,'As Required':1,'Example Behaviour':2}

(g) Pay_Level: {'Birdie':0,'On Average':1,'Market or above':2}

(4) One-hot Encoding to Encode the Nominal Categorical Variables

For the nominal categorical variables, to avoid the order effect which do not exist, we use one-hot encoding on these variables.

(5) Split the Data into Training and Testing Set

We split the data into 70% training and 30% testing set with stratified shuffle to guarantee they have the same proportion for each level of output. We also set fixed random state at 123 to guarantee this result can be reproduced.

3.3 Prediction Model

Since we want to figure out the potential leavers accurately, which is the true positive (TP), we not only focus on the accuracy, but also focus on the F1 score since we want to find a balance between precision and recall.

$$\text{accuracy} = \frac{TP+TN}{TP+FN+TN+FP}$$

$$\text{precision} = \frac{TP}{TP+FP}$$

$$\text{recall} = \frac{TP}{TP+FN}$$

$$\text{F1 score} = \frac{2 \times precision \times recall}{precision + recall}$$

3.3.1 Prediction Model: Logistics Regression

We first apply logistics regression to the training data. However, there is some perfect collinearity problem here so we delete the following variables: "Gender_Male", "Roles_Support", "Marital_Status_Unmarried", "Sports_Level_Professional", "Manager_OrNot_No" and "Blogger_yn_No".

For this logistics regression model, the top 3 importance features are as follows.

Features	**Blogger_yn_Yes**	**Roles_Day_to_Day_Ops**	**Sports_Level_Hobby**
| Coef |	0.731815	0.436177	0.397005

We further set different threshold (0.1-0.9) to adjust the prediction of the testing set and check the corresponding accuracy/f1 score.

	Decision Boundary_0.7	**Decision Boundary_0.6**	**Decision Boundary_0.5**	**Decision Boundary_0.4**
Accuracy	0.653333	0.646667	0.636667	0.600000
Precision	0.000000	0.250000	0.380952	0.402439
Recall	0.000000	0.009615	0.076923	0.317308
F1 score	0.000000	0.018519	0.128000	0.354839

Although we obtained a better accuracy with higher threshold, the corresponding F1 score is low, showing that our model has poor performance with both poor precision and recall. Thus, further improvement may be needed.

3.3.2 Prediction Model: Random Forest Classification

Next, we train the random forest model. Here we use grid search CV to tune the parameters and select accuracy as the scoring criteria (using f1 score directly can cause severe overfitting issue here).

The Optimal tunning parameters are shown as below with model score = 0.699.

n_estimators	max_depth	min_samples_split	criterion	bootstrap
10	5	2	entropy	True

For the random forest classification model, the top 3 importance features are:

	Fixed_Salary	Academics_Level	people_reporting_count
Feature importance	0.239940	0.093853	0.056199

Compared with the top 3 important features of logistics regression, the random forest model gives out more reasonable split node so it will be our priority if they have close accuracy and f1 score.

Similarly, we further set different threshold (0.1-0.9) to adjust the prediction of the testing set and check the corresponding accuracy/f1 score.

	Decision Boundary_0.7	Decision Boundary_0.6	Decision Boundary_0.5	Decision Boundary_0.4
accuracy	0.653333	0.650000	0.636667	0.600000
precision	0.000000	0.333333	0.272727	0.378788
recall	0.000000	0.009615	0.028846	0.240385
f1score	0.000000	0.018692	0.052174	0.294118

Disappointedly, similar issue happens, that is, high accuracy is accompanied with low f1 score showing poor model performance.

Therefore, some enhanced method should be conducted in the later study.

3.3.3 Prediction Model: Over-sample the minority class with logistic regression

To solve the above problem, we made some adjustment to the original dataset. As the original dataset only contains 34.6% of leavers, the model does not have enough data to gives

out poor performance. Thus, we apply oversampling for the minority class to 654 records and the new dataset have equal records of leavers = 1 and leavers = 0.

Again, we split the data into 70% training and 30% testing set with stratified shuffle. Now we train the logistics model again with the new training set.

For the oversampling logistics regression model, the top 3 importance features are follows.

Features	**Blogger_yn_Yes**	**Sports_Level_Hobby**	**Pay_Level**
| Coef |	0.639465	0.518185	0.450151

We further set different threshold (0.1-0.9) to adjust the prediction of the testing set and check the corresponding accuracy/f1 score.

	Decision Boundary_0.6	**Decision Boundary_0.5**	**Decision Boundary_0.4**	**Decision Boundary_0.3**
accuracy	0.539440	0.539440	**0.544529**	0.506361
precision	0.622951	0.537688	0.525680	0.502604
recall	0.193878	0.545918	0.887755	0.984694
f1score	0.295720	0.541772	**0.660342**	0.665517

Although optimal accuracy is only 0.545, we reach a relatively good f1 score here. Thus, for the oversampled logistics regression, the optimal threshold is 0.4.

3.3.4 Prediction Model: Over-sample the minority class with Random Forest

We continue to use the new oversampled training set to train the RF model. Here we use grid search CV to tune the parameters and select accuracy as the scoring criteria (using f1 may cause severe overfitting issue and poor performance in testing set).

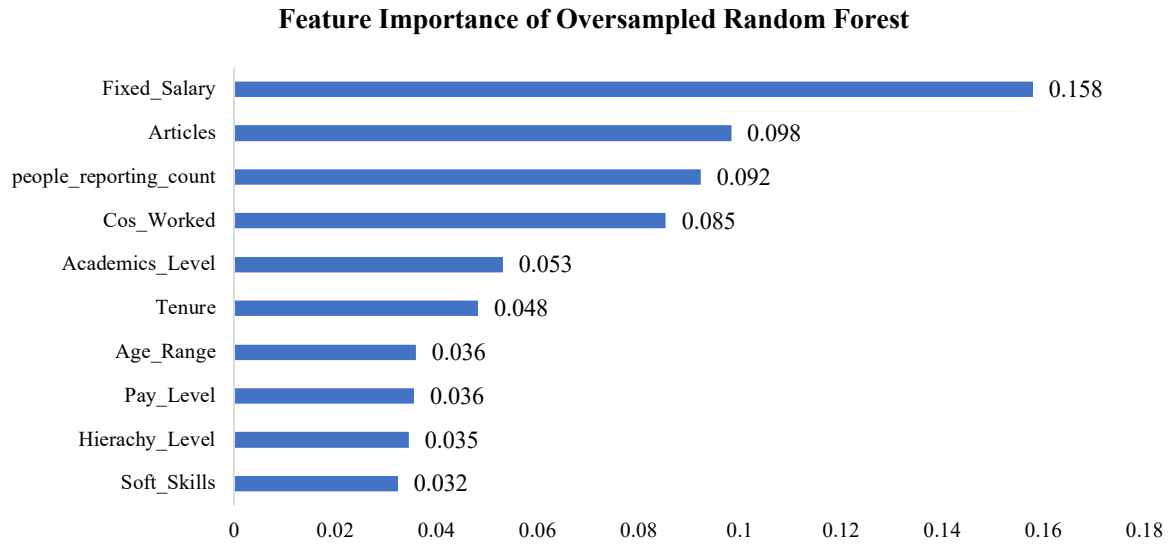
The Optimal tunning parameters are shown as below with model score = 0.992.

n_estimators	**max_depth**	**min_samples_split**	**criterion**	**bootstrap**
1000	10	2	gini	False

For the random forest classification model, the top 3 importance features are:

Features	**Fixed_Salary**	**Articles**	**people_reporting_count**
importance	0.158	0.098	0.092

The most important feature is still the fixed salary and this is why we hold a strong believe that make adjustment to salary can reduce the turnover rate significantly.



We further set different threshold (0.1-0.9) to adjust the prediction of the testing set and check the corresponding accuracy/f1 score.

	Decision_Boundary_0.6	Decision_Boundary_0.5	Decision_Boundary_0.4
accuracy	0.834606	**0.832061**	0.692112
precision	0.933775	**0.831633**	0.632509
recall	0.719388	**0.831633**	0.913265
f1score	0.812680	**0.831633**	0.747390

Compared with all the above models, this oversampling random forest model gives the highest accuracy with best corresponding f1 score. Here, we choose the decision boundary = 0.5 for the optimal prediction performance on the testing set, and further use this random forest model in the optimization formulation.

4. Optimization Question Formulation

Our objective is to minimize the total cost of retaining this company's current workforce while reducing the employee turnover rate. The total cost comes from two aspects, one is the cost of raising fixed annual base salary to prevent employees from leaving, the other is the cost of hiring new employees to fill the vacancy. We solve the optimization problem on the testing

set with 300 employees and use the best oversampling random forest model with decision boundary of 0.5 to predict the leavers on the testing set.

Out of the 300 employees, there're 82 predicted leavers and the overall employee turnover rate is 27.3%, the turnover rates for employees of different level are shown in the table below.

	Low	**Middle**	**High**	**Overall**
Turnover Rate	27.05%	30.12%	10.0%	27.3%

The assumptions are as follows:

1. The total cost of hiring a new employee is a fixed one-time fee of $16999.
2. The cost of keeping a leaving employee is proportional to the annual base salary, normally ranging from 10% to 20%. If the company raises the annual base salary for leaving employees by a certain proportion, then they will choose to stay at their current position, otherwise, they will choose to leave. Here we analyze under three scenarios, raise 10%, 15% and 20% of the annual fixed salary, and solve the minimization problem respectively.
3. For Optimization Model 1, our objective is to minimize the total cost with no constraint on employee turnover rate
4. For Optimization Model 2, our objective is to minimize the total cost with constraint that the overall employee turnover rate does not exceed 10%.
5. For Optimization Model 3, our objective is to minimize the total cost with constraint that the middle level employee turnover rate does not exceed 15%.

4.1 Model Formulation

We define the variables as follows.

rc_i: base pay raise cost

hc_i: hiring cost

N: number of predicted leavers

M: number of predicted middle level leavers

x_{1i}: binary decision variable, whether to raise base pay or not for employee *i*,

x_{2i}: binary decision variable, whether to hire new employee *i*

The corresponding objective function is as follows.

$$\min_{x_1, x_2} \sum_{i=1}^{N} rc_i\, x_{1i} + hc_i x_{2i}$$

Model 1 constraint:

$$x_{1i} + x_{2i} = 1,\ \forall i \in N$$

$$x_{1i}, x_{2i} \in \{0,1\},\ \forall i \in N$$

Model 2 additional constraint:

$$\frac{1}{N}\sum_{i=1}^{N} x_{2i} \leq 0.1,\ \forall i \in N$$

Model 3 additional constraint

$$\tfrac{1}{M}\textstyle\sum_{i=1}^{M} x_{2i} \leq 0.15,\ \forall i \in M$$

4.2 Benchmark

We set two benchmarks with no total cost optimization to compare the performance on cost reduction of our optimization model.

(1) Benchmark 1: Assume the company doesn't raise base salary pay and all hire new employees to fill the vacancy of all 82 leavers, then total cost (hiring cost) will be $1,393,918.

(2) Benchmark 2: Assume the company decides to give all leaving employees annual base salary pay raise, then the total cost under the three scenarios 10%, 15% and 20% are $1,620,076, $2,430,114, $3,240,152 respectively.

	Benchmark 1	**Benchmark 2 10%**	**Benchmark 2 15%**	**Benchmark 2 20%**
Total cost	$1,393,918	$1,620,076	$2,430,114	$3,240,152

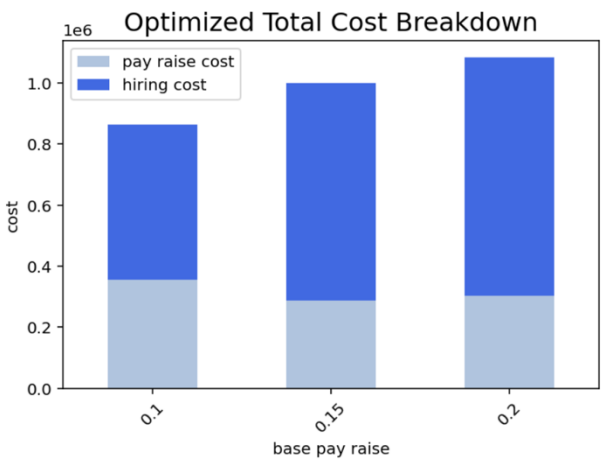
4.3 Scenario Analysis

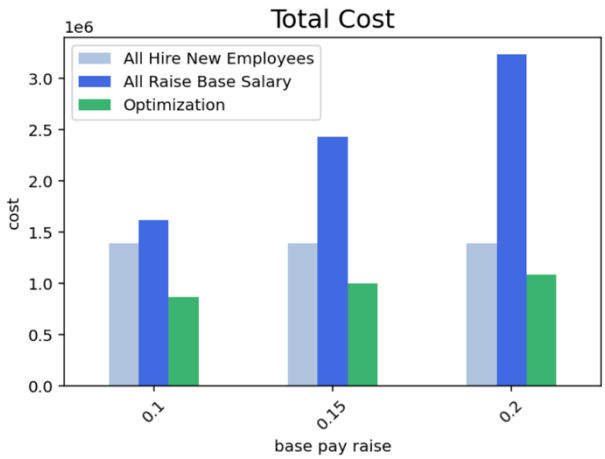
4.3.1 Optimization Model 1: Global Minimum Cost

The optimal solutions and the corresponding employee turnover rate are shown in the table below, the average turnover rate across the three scenarios is 13.11%.

scenario	**Number of stayers**	**Number of leavers**	**Turnover rate**
10%	52	30	10.00%

15%	40	42	14.00%
20%	36	46	15.33%





The optimal total cost breakdown and cost reduction compared with benchmark are shown in the table below. Across the three scenarios, the average total cost is $983,578, the average cost reduction compared to benchmark 1 is 29.44%, the average cost reduction compared to benchmark 2 is 57.32%.

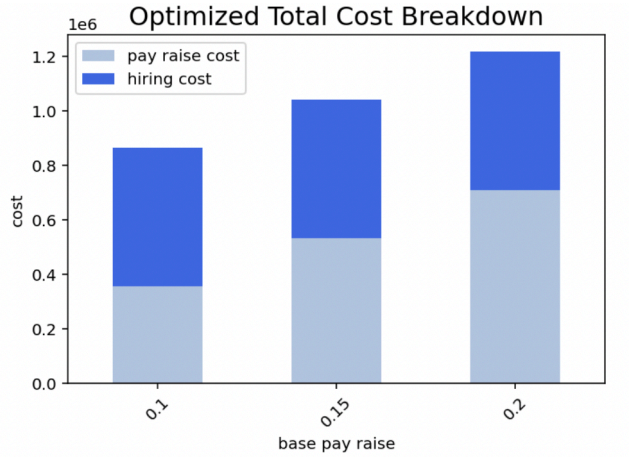
scenario	**Pay raise cost**	**Hiring cost**	**Total cost**	**Cost reduction (Benchmark 1)**	**Cost reduction (Benchmark 2)**
10%	$355,030	$509,970	$865,000	37.94%	46.61%
15%	$286,524	$713,958	$1,000,482	28.23%	58.83%
20%	$303,299	$781,954	$1,085,253	22.14%	66.51%

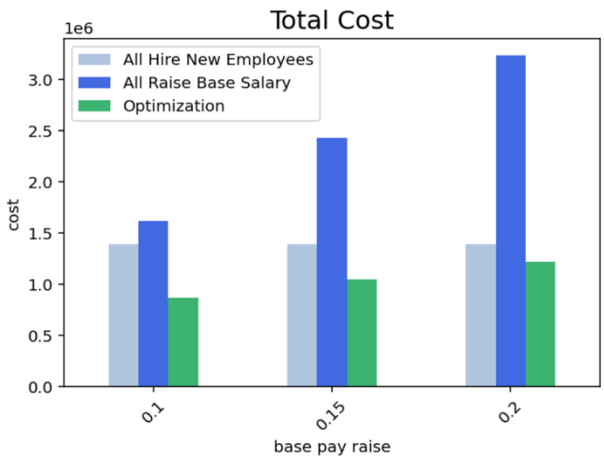
In summary, under Optimization Model 1, the 10% annual base pay raise scenario achieves the minimum total cost of $865,000, the highest cost reduction compared to benchmark 1 and lowest overall turnover rate of 10%. Additionally, the 20% annual base pay raise scenario achieves the highest cost reduction compared to benchmark 2.

4.3.2 Optimization Model 2: Overall Turnover Rate $\leq$ 10%

The optimal solutions and the corresponding employee turnover rate are shown in the table below, because of the added constraint on the overall turnover rate, the turnover rates under optimal solutions are all 10% across the three scenarios.

Scenario	Number of stayers	Number of leavers	Turnover rate
10%	52	30	10%
15%	52	30	10%
20%	52	30	10%





The optimal total cost breakdown and cost reduction compared with benchmark are shown in the table below. Across the three scenarios, the average total cost is $1,042,514, the average cost reduction compared to benchmark 1 is 25.21%, the average cost reduction compared to benchmark 2 is 55.35%.

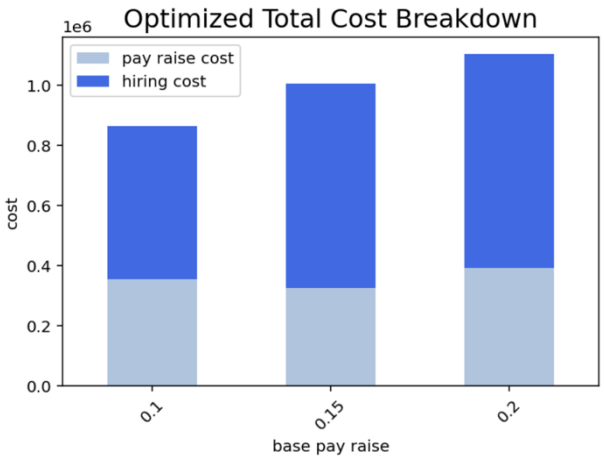
Scenario	Pay raise cost	Hiring cost	Total cost	Cost reduction (Benchmark 1)	Cost reduction (Benchmark 2)
10%	$355,030	$509,970	$865,000	37.94%	46.61%
15%	$532,544	$509,970	$1,042,514	25.21%	57.10%
20%	$710,059	$509,970	$1,220,029	12.47%	62.35%

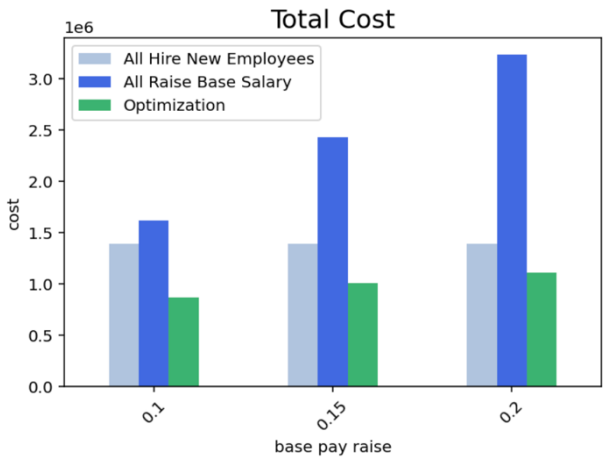
In conclusion, under Optimization Model 2, the 10% annual base pay raise scenario achieves the minimum total cost of $865,000 and the highest cost reduction compared to benchmark 1. Besides, the 20% annual base pay raise scenario achieves the highest cost reduction compared to benchmark 2.

4.3.3 Optimization Model 3: Middle Level Turnover Rate $\leq$ 15%

The optimal solutions and the corresponding employee turnover rate are shown in the table below, across the three scenarios, the average turnover rate is 12.44%, the average middle level turnover rate is 12.45%.

Scenario	Number of stayers	Number of leavers	Turnover rate	Middle Level Turnover rate
10%	52	30	10.00%	8.43%
15%	42	40	13.33%	14.46%
20%	40	42	14.00%	14.46%





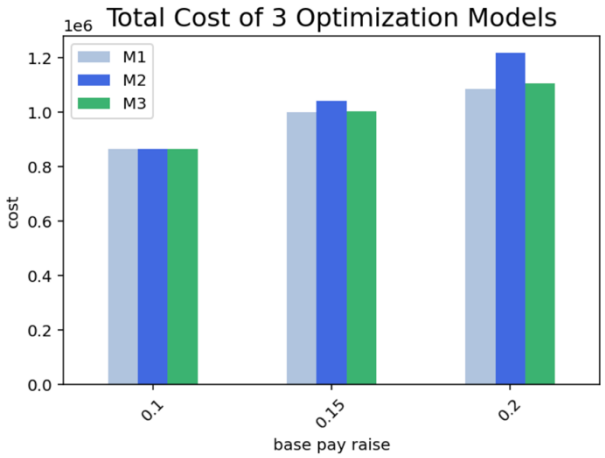
The optimal total cost breakdown and cost reduction compared with benchmark are shown in the table below. Across the three scenarios, the average total cost is $991,972, the average cost reduction compared to benchmark 1 is 28.83%, the average cost reduction compared to benchmark 2 is 57.04%.

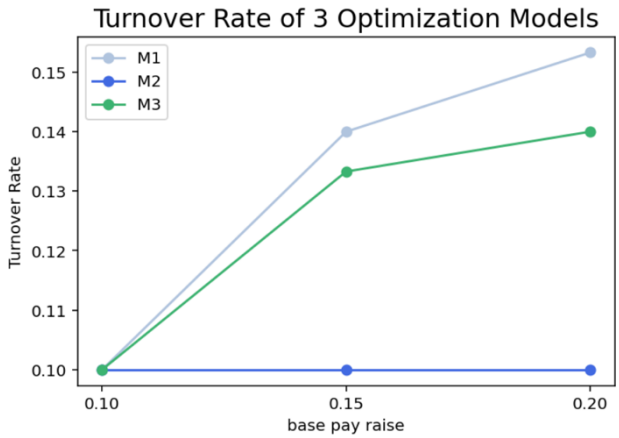
Scenario	Pay raise cost	Hiring cost	Total cost	Cost reduction (Benchmark 1)	Cost reduction (Benchmark 2)
10%	$355,030	$509,970	$865,000	37.94%	46.61%
15%	$325,005	$679,960	$1,004,965	27.9%	58.65%
20%	$391,994	$713,958	$1,105,952	20.66%	65.87%

Hence, under Optimization Model 3, the 10% annual base pay raise scenario achieves the minimum total cost of $865,000, the highest cost reduction compared to benchmark 1 and lowest overall turnover rate of 10%. In addition, the 20% annual base pay raise scenario achieves the highest cost reduction compared to benchmark 2.

5. Analysis

Overall, across the 3 Optimization Models, the 10% annual base pay raise scenario always achieves the minimum total cost of $865,000 and the lowest employment turnover rate of 10% regardless of the additional constraints added. As shown from the figures below, for the 3 scenarios, as the level of pay raise increases, the total employment cost, and the overall turnover rate both increases under each Optimization Model. Furthermore, increasing the proportion of annual base pay raise has no effect on reducing the turnover rate, while it increases the turnover rate. Since our objective is to minimize the total employment cost, including both the base pay raise cost and the hiring cost, and the average marginal cost for raising annual fixed salary is higher than the fixed marginal cost for hiring new employee, the total cost will surely increase as the level of base pay raise increases. Therefore, it is optimal to keep the pay raise level at 10% while achieving the minimal cost and low turnover rate.





Scenario	Average Marginal Cost for Raising Salary	Marginal Cost for Hiring New
10%	$19,757	$16,999
15%	$29,636	$16,999
20%	$39,514	$16,999

6. Conclusion

In present study, we provide service for a company, which has a poor turnover rate (34.6%), to help them reduce the turnover rate with minimal cost. First, we need to develop a leaver detection program by training the data with ordinary logistics regression, ordinary random forest classification, oversampling logistics regression and oversampling random forest classification. With 0.832 accuracy and 0.832 f1 score, oversampling random forest classification model gives out the best performance and is selected for latter optimization.

Based on different requirements on the turnover rate, we formulated 3 different optimization models and conducted the scenario analysis with 10%, 15% and 20% annual base pay raise. After carefully analyzing the optimal solutions, there are some conclusions as follows.

(1) 10% annual base pay raise always achieves the minimum total cost at $865,000 and the lowest employment turnover rate of 10% under the 3 optimization models.
(2) Increasing the proportion of annual base pay raise has negative effect on reducing the turnover rate, which increases the turnover rate and the total cost.

Hence, based on current dataset and hiring cost assumptions, we suggest the company should offer a 10% annual base pay raise for the detected potential leavers. With this pay raise, the company can retain the turnover rate at 10% with the optimal cost.

7. Limitations and Further Studies

Optimization model assumption

In our optimization model, we made a few assumptions based on the detailed literature review which is only valid in ideal situation. Violating any of the ideal assumptions can lead to different optimal solutions and further studies should be conducted in the future.

(1) The assumption that raising annual base pay by 10% to 20% will 100% prevent employees from leaving may not be valid. In fact, previous study has pointed out more factors can influence employees' resigning decision like high work stress, poor co-workers and family factors (Verlinden, 2019). That is, merely raising the salary do not guarantee to keep leavers by 100%.

(2) The hiring cost may be varied for different situations instead of remaining at a fixed value. In real world, the hiring cost of different position in different periods may vary quite a lot and more information is needed to enhance the model.

(3) In our case, we only analyze within a one-year time horizon and compare the fixed one-time cost of hiring new employees with the cost of raising annual fixed salary. However, this change is not a one-time issue and in reality, dynamic optimization may be needed within a longer time horizon.

Model constraints

To further enhance the model, more realistic constraints should be added in practice if more detailed information of the company can be disclosed.

(1) Some sophisticated requirements across employee turnover rate across different employee level, department, etc.

(2) The employment pays raise budget on different employee tenure, department, etc.

8. Appendix 1: Reference List

Elliott, J. (2022). The True Cost of Hiring an Employee in 2022. Retrieved from https://toggl.com/blog/cost-of-hiring-an-employee#totalcost

Keng, C. (2014). Employees Who Stay in Companies Longer Than Two Years Get Paid 50% Less. Retrieved from https://www.forbes.com/sites/cameronkeng/2014/06/22/employees-that-stay-in-companies-longer-than-2-years-get-paid-50-less/?sh=63b29418e07f

Muralidharan, M. (2019). HR data - for ML/DL/optimization practice. Retrieved from https://www.kaggle.com/datasets/stitch/hr-data-for-mldloptimization-practice

Muralidharan, M. (2020). HR data for practicing ML/DL. Retrieved from https://medium.com/analytics-vidhya/hr-data-for-practicing-ml-dl-5fba688bebc1

Verlinden, N. (2019). What does high turnover mean? Turnover rates, jobs, and causes. Retrieved from https://www.aihr.com/blog/high-turnover-meaning-rates/

9. Appendix 2: Code

```
#!/usr/bin/env python
```

```
# coding: utf-8

# ## 1. Import the data and Feature Engineering

# In[1]:


import pandas as pd
#import sweetviz as sv
#import pandas_profiling
import numpy as np
import matplotlib.pyplot as plt
import IPython
get_ipython().run_line_magic('matplotlib', 'inline')
import matplotlib_inline
# improve inline plot resolution with retina format
matplotlib_inline.backend_inline.set_matplotlib_formats('retina', 'jpg')
pd.set_option('display.max_columns', None)
import pyomo.environ as pe
solver = pe.SolverFactory('gurobi', solver_io = "python")


# In[2]:


df1 = pd.read_csv(r'hr_data_comprehensive.csv')
df1.head(20)


# In[3]:


list(df1)


# In[4]:


df1=df1[[ 'Gender',
 'Marital_Status',
 'Age_Range',
 'Blogger_yn',
 'Tenure',
 'Academics_Level',
 'Sports_Level',
 'Hierachy_Level',
```

```
 'Manager_OrNot',
 'Rated',
 'Soft_Skills',
 'Roles',
 'Articles',
 'Cos_Worked',
 'people_reporting_count',
 'Fixed_Salary',
 'Pay_Level',
 'Leavers']]


# In[5]:


df1


# In[6]:


df1['Tenure'].unique()

# ## 2. Data pre-processing
# ### (1) fill the NA values

# In[7]:


df1['Tenure'].unique()


# In[8]:


df1[df1['Manager_OrNot']=='No']['people_reporting_count'].value_counts()


# In[9]:


df1[df1['Manager_OrNot']=='Yes']['people_reporting_count'].unique()


# In[10]:
```

```
df1[df1['Manager_OrNot']=='No']['people_reporting_count']


# In[11]:


df1['people_reporting_count']=df1['people_reporting_count'].fillna(0)


# In[12]:


df1.loc[(df1.Blogger_yn=='No') & (df1['Articles'].isnull()),'Articles'] = 0
df1['Articles'].unique()


# In[13]:


df1


# In[14]:


df1.isnull().any()


# In[15]:


len(df1)


# In[16]:


df_raw = df1.copy() # copy of original dataset for optimization part


# ### (2) Rescaling: transform the Fixed_Salary to log(Fixed_Salary)

# In[17]:


import math

for i in range(len(df1)):
```

```
    df1['Fixed_Salary'][i]=math.log(df1['Fixed_Salary'][i])
df1
```

```
# ### (3)  Label encoding for the Ordinal Categorical variables
```

```
# In[18]:
```

```
df1['Age_Range'].unique()
```

```
# In[19]:
```

```
age_mappings = {'18-30':0,'30-50':1,'50-70':2}
df1['Age_Range'] = df1['Age_Range'].map(age_mappings)
```

```
# In[20]:
```

```
df1['Tenure'].unique()
```

```
# In[21]:
```

```
tenure_mappings = {'0-3':0,'4-8':1,'8-15':2,'15-30':3}
df1['Tenure'] = df1['Tenure'].map(tenure_mappings)
```

```
# In[22]:
```

```
df1['Academics_Level'].unique()
```

```
# In[23]:
```

```
aca_mappings =
{'Graduates':0,'Specialization/Masters':1,'Phd/Tier1/ProAccredition':2}
df1['Academics_Level'] = df1['Academics_Level'].map(aca_mappings)
```

```
# In[24]:
```

```
df1['Hierachy_Level'].unique()
```

```
# In[25]:
```

```
hier_mappings = {'Low':0,'Middle':1,'Senior':2}
df1['Hierachy_Level'] = df1['Hierachy_Level'].map(hier_mappings)
```

```
# In[26]:
```

```
df1['Rated'].unique()
```

```
# In[27]:
```

```
rate_mappings = {'Bottom':0,'Average':1,'High':2}
df1['Rated'] = df1['Rated'].map(rate_mappings)
```

```
# In[28]:
```

```
df1['Soft_Skills'].unique()
```

```
# In[29]:
```

```
hier_mappings = {'Temperamental':0,'As Required':1,'Example Behaviour':2}
df1['Soft_Skills'] = df1['Soft_Skills'].map(hier_mappings)
```

```
# In[30]:
```

```
df1['Pay_Level'].unique()
```

```
# In[31]:
```

```
hier_mappings = {'Birdie':0,'On Average':1,'Market or above':2}
```

```
df1['Pay_Level'] = df1['Pay_Level'].map(hier_mappings)


# ### (4) One-hot encoding to encode the nominal categorical variables

# In[32]:


one_hot_encoded_training_predictors = pd.get_dummies(df1)


# In[33]:


one_hot_encoded_training_predictors


# ## 3. Split the data into training and testing set

# In[34]:


#pip install sklearn


# In[81]:


x = one_hot_encoded_training_predictors.drop(['Leavers'],axis=1)
y = one_hot_encoded_training_predictors['Leavers']

from sklearn.model_selection import StratifiedShuffleSplit
sss=StratifiedShuffleSplit(n_splits=5,test_size=0.3,random_state=123)

for train_index, test_index in sss.split(df1, df1["Leavers"]):
    X_train = x.iloc[train_index]
    y_train = y.iloc[train_index]
    X_test = x.iloc[test_index]
    y_test = y.iloc[test_index]


# In[36]:


y_train.value_counts()[1]/700
```

```
# In[37]:


y_test.value_counts()[1]/300


# ## 4.1 Train the model: logistics regression

# In[46]:


from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state=123,max_iter=10000).fit(X_train, y_train)


# In[47]:


clf.classes_


# In[48]:


clf.intercept_


# In[49]:


clf.coef_


# In[42]:


logistic_coef = pd.DataFrame(clf.coef_,columns=list(X_train))
logistic_coef.T


# * We have perfect colinearity problem.
# * Delete `Gender_Male`,`Marital_Status_Unmarried`,`Blogger_yn_No`,
`Manager_OrNot_No` and `Sports_Level_Professional`,`Roles_Support`.

# In[43]:
```

```
X_train_1 = X_train.drop(['Gender_Male', 'Marital_Status_Unmarried',
'Blogger_yn_No',
'Manager_OrNot_No','Sports_Level_Professional','Roles_Support'],axis=1)
X_test_1 = X_test.drop(['Gender_Male', 'Marital_Status_Unmarried',
'Blogger_yn_No',
'Manager_OrNot_No','Sports_Level_Professional','Roles_Support'],axis=1)
```

```
# In[44]:
```

```
clf1 = LogisticRegression(random_state=123,max_iter=10000).fit(X_train_1, y_train)
```

```
# In[45]:
```

```
logistic_coef1 = pd.DataFrame(clf1.coef_,columns=list(X_train_1))
logistic_coef1.T.sort_values(0,axis=0,ascending=False).head(10)
```

```
# In[46]:
```

```
import numpy as np
from sklearn import metrics
def get_metrics_threshold(model,x,y,theta):
    ypred = np.where(model.predict_proba(x)[:,1]>=theta,1,0)

    # SVM dont have prabability prediction, so no auc_score
    try:
        pscore = model.predict_proba(x)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y, pscore)
        auc_score = metrics.auc(fpr, tpr)
    except:
        auc_score = "NaN"

    accuracy = metrics.accuracy_score(y, ypred)
    precision = metrics.precision_score(y, ypred,zero_division=0)
    precision_0 = metrics.precision_score(y, ypred,pos_label=0,zero_division=0)
    recall = metrics.recall_score(y, ypred)
    recall_0 = metrics.recall_score(y, ypred,pos_label=0)
    f1score = metrics.f1_score(y, ypred)
    f1score_0 = metrics.f1_score(y, ypred,pos_label=0)
    accuracy_insample = model.score(X_train_1,y_train)
```

```
    return
[accuracy,auc_score,precision,recall,f1score,precision_0,recall_0,f1score_0,accura
cy_insample]

result_logit =
pd.DataFrame([],index=['accuracy','auc_score','precision','recall','f1score','prec
ision_0','recall_0','f1score_0','accuracy_insample'])
result_logit['Decision_Boundary_0.9'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.9)
result_logit['Decision_Boundary_0.8'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.8)
result_logit['Decision_Boundary_0.7'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.7)
result_logit['Decision_Boundary_0.6'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.6)
result_logit['Decision_Boundary_0.5'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.5)
result_logit['Decision_Boundary_0.4'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.4)
result_logit['Decision_Boundary_0.3'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.3)
result_logit['Decision_Boundary_0.2'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.2)
result_logit['Decision_Boundary_0.1'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.1)


# In[47]:


result_logit


# ## 4.2 Train the model: random forest

# In[48]:


from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier


estimator= RandomForestClassifier()
param_grid = {
    'n_estimators': [10,100,1000],
    'max_depth': [5, 8, 10],
```

```
    'min_samples_split':[2,5,8],
    'bootstrap':[True,False],
    'criterion': ['gini','entropy']
}

model = GridSearchCV(RandomForestClassifier(random_state=123), param_grid,
cv=5,scoring='accuracy')
model.fit(X_train, y_train.ravel())
model.score(X_train,y_train)


# In[49]:


model.best_estimator_.get_params()


# In[50]:


model.predict_proba(X_train)


# In[51]:


model.score(X_train,y_train)


# In[52]:


model.score(X_test,y_test)


# In[53]:


def get_metrics_threshold(model,x,y,theta):
    ypred = np.where(model.predict_proba(x)[:,1]>=theta,1,0)

    try:
        pscore = model.predict_proba(x)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y, pscore)
        auc_score = metrics.auc(fpr, tpr)
    except:
        auc_score = "NaN"
```

```
    accuracy = metrics.accuracy_score(y, ypred)
    precision = metrics.precision_score(y, ypred)
    precision_0 = metrics.precision_score(y, ypred,pos_label=0)
    recall = metrics.recall_score(y, ypred)
    recall_0 = metrics.recall_score(y, ypred,pos_label=0)
    f1score = metrics.f1_score(y, ypred)
    f1score_0 = metrics.f1_score(y, ypred,pos_label=0)
    accuracy_insample = model.score(X_train,y_train)
    return
[accuracy,auc_score,precision,recall,f1score,precision_0,recall_0,f1score_0,accura
cy_insample]

result_rf =
pd.DataFrame([],index=['accuracy','auc_score','precision','recall','f1score','prec
ision_0','recall_0','f1score_0','accuracy_insample'])
result_rf['Decision_Boundary_0.9'] =
get_metrics_threshold(model,X_test,y_test,0.9)
result_rf['Decision_Boundary_0.8'] =
get_metrics_threshold(model,X_test,y_test,0.8)
result_rf['Decision_Boundary_0.7'] =
get_metrics_threshold(model,X_test,y_test,0.7)
result_rf['Decision_Boundary_0.6'] =
get_metrics_threshold(model,X_test,y_test,0.6)
result_rf['Decision_Boundary_0.5'] =
get_metrics_threshold(model,X_test,y_test,0.5)
result_rf['Decision_Boundary_0.4'] =
get_metrics_threshold(model,X_test,y_test,0.4)
result_rf['Decision_Boundary_0.3'] =
get_metrics_threshold(model,X_test,y_test,0.3)
result_rf['Decision_Boundary_0.2'] =
get_metrics_threshold(model,X_test,y_test,0.2)
result_rf['Decision_Boundary_0.1'] =
get_metrics_threshold(model,X_test,y_test,0.1)


# In[54]:


result_rf


# In[55]:
```

```
def get_accuracy_train_rf(threshold):
    y_train_new = []
    new = model.predict_proba(X_train)
    for i in range(len(new)):
        if new[i][0]>=threshold:
            y_train_new.append(1)
        else:
            y_train_new.append(0)

    y_train_old = list(y_train)

    accuracy_train = 0
    for i in range(700):
        if y_train_new[i] == y_train_old[i]:
            accuracy_train = accuracy_train+1
        else:
            accuracy_train = accuracy_train
    return accuracy_train/700, threshold


# In[56]:


def get_accuracy_test_rf(threshold):
    y_test_new = []
    new = model.predict_proba(X_test)
    for i in range(len(new)):
        if new[i][0]>=threshold:
            y_test_new.append(1)
        else:
            y_test_new.append(0)

    y_test_old = list(y_test)

    accuracy_test = 0
    for i in range(300):
        if y_test_new[i] == y_test_old[i]:
            accuracy_test = accuracy_test+1
        else:
            accuracy_test = accuracy_test
    return accuracy_test/300, threshold


# In[57]:
```

```
thr_list=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]

for threshold in thr_list:
    print('Decision boundary: ',threshold,'Training Accuracy:
',get_accuracy_train_rf(threshold)[0],'Testing Accuracy:
',get_accuracy_test_rf(threshold)[0])


# In[58]:


feature_names = [i for i in x]
importances = model.best_estimator_.feature_importances_
feature_importances = pd.DataFrame({"features" : feature_names,
 "importances" : importances})
feature_importances.sort_values('importances',axis=0,ascending=False).head(10)


# In[59]:


import numpy as np
import matplotlib.pyplot as plt

y=list(feature_importances.sort_values('importances',axis=0,ascending=False).head(
10)['importances'])
feature_names_new=list(feature_importances.sort_values('importances',axis=0,ascend
ing=False).head(10)['features'])

fig, ax = plt.subplots()
b = ax.barh(range(len(feature_names_new)), y, color='#6699CC')
ax.set_yticks(range(len(feature_names_new)))
ax.set_yticklabels(feature_names_new)


# - Although the prediction accuracy of logistics regression and the random forest
model is the same, random forest provide us with more interpretable split
nodes(important features).
#
# - Thus, we use the random forest model to generate the y_pred value in the
optimization part.

# ## 4.3 Over-sample the minority class with logistic regression

# In[259]:
```

```
from sklearn.utils import resample
# df1 is a data frame with `Leavers` as the target column with classes 0 and 1.
# There are more instances of class 0 than class 1 in the data frame df.

# Separate majority and minority classes
df_majority =
one_hot_encoded_training_predictors.loc[one_hot_encoded_training_predictors.Leaver
s == 0].copy()
df_minority =
one_hot_encoded_training_predictors.loc[one_hot_encoded_training_predictors.Leaver
s == 1].copy()

# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,  # sample with replacement
                                 n_samples=654,  # to match majority class
                                 random_state=123)  # reproducible results

# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_majority, df_minority_upsampled])

# Display new class counts
print(df_upsampled.Leavers.value_counts())


# In[260]:


x1 = df_upsampled.drop(['Leavers'],axis=1)
y1 = df_upsampled['Leavers']

from sklearn.model_selection import StratifiedShuffleSplit
sss=StratifiedShuffleSplit(n_splits=5,test_size=0.3,random_state=123)

for train_index, test_index in sss.split(df_upsampled, df_upsampled["Leavers"]):
    X_train = x1.iloc[train_index]
    y_train = y1.iloc[train_index]
    X_test = x1.iloc[test_index]
    y_test = y1.iloc[test_index]


# In[261]:


y_train.value_counts()[1]/700
```

```
# In[106]:


y_test.value_counts()[1]/300


# In[54]:


X_train_1 = X_train.drop(['Gender_Male', 'Marital_Status_Unmarried',
'Blogger_yn_No',
'Manager_OrNot_No','Sports_Level_Professional','Roles_Support'],axis=1)
X_test_1 = X_test.drop(['Gender_Male', 'Marital_Status_Unmarried',
'Blogger_yn_No',
'Manager_OrNot_No','Sports_Level_Professional','Roles_Support'],axis=1)


# In[55]:


clf1 = LogisticRegression(random_state=123,max_iter=10000).fit(X_train_1, y_train)


# In[56]:


logistic_coef1 = pd.DataFrame(clf1.coef_,columns=list(X_train_1))
logistic_coef1.T.sort_values(0,axis=0,ascending=False).head(10)


# In[60]:


from sklearn import metrics
def get_metrics_threshold(model,x,y,theta):
    ypred = np.where(model.predict_proba(x)[:,1]>=theta,1,0)

    try:
        pscore = model.predict_proba(x)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y, pscore)
        auc_score = metrics.auc(fpr, tpr)
    except:
        auc_score = "NaN"

    accuracy = metrics.accuracy_score(y, ypred)
```

```
    precision = metrics.precision_score(y, ypred)
    precision_0 = metrics.precision_score(y, ypred,pos_label=0)
    recall = metrics.recall_score(y, ypred)
    recall_0 = metrics.recall_score(y, ypred,pos_label=0)
    f1score = metrics.f1_score(y, ypred)
    f1score_0 = metrics.f1_score(y, ypred,pos_label=0)
    accuracy_insample = model.score(X_train_1,y_train)
    return
[accuracy,auc_score,precision,recall,f1score,precision_0,recall_0,f1score_0,accura
cy_insample]

result_logit_upsample =
pd.DataFrame([],index=['accuracy','auc_score','precision','recall','f1score','prec
ision_0','recall_0','f1score_0','accuracy_insample'])
result_logit_upsample['Decision_Boundary_0.9'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.9)
result_logit_upsample['Decision_Boundary_0.8'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.8)
result_logit_upsample['Decision_Boundary_0.7'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.7)
result_logit_upsample['Decision_Boundary_0.6'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.6)
result_logit_upsample['Decision_Boundary_0.5'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.5)
result_logit_upsample['Decision_Boundary_0.4'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.4)
result_logit_upsample['Decision_Boundary_0.3'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.3)
result_logit_upsample['Decision_Boundary_0.2'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.2)
result_logit_upsample['Decision_Boundary_0.1'] =
get_metrics_threshold(clf1,X_test_1,y_test,0.1)


# In[61]:


result_logit_upsample


# ## 4.4 Over-sample the minority class with random forest

# In[107]:
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier


# In[108]:


estimator= RandomForestClassifier()
param_grid = {
    'n_estimators': [10,100,1000],
    'max_depth': [5, 8, 10],
    'min_samples_split':[2,5,8],
    'bootstrap':[True,False],
    'criterion': ['gini','entropy']
}

model = GridSearchCV(RandomForestClassifier(random_state=123), param_grid,
cv=5,scoring='accuracy')
model.fit(X_train, y_train.ravel())
model.score(X_train,y_train)


# In[109]:


model.best_estimator_.get_params()


# In[69]:


def get_metrics_threshold(model,x,y,theta):
    ypred = np.where(model.predict_proba(x)[:,1]>=theta,1,0)

    try:
        pscore = model.predict_proba(x)[:,1]
        fpr, tpr, thresholds = metrics.roc_curve(y, pscore)
        auc_score = metrics.auc(fpr, tpr)
    except:
        auc_score = "NaN"

    accuracy = metrics.accuracy_score(y, ypred)
    precision = metrics.precision_score(y, ypred)
    precision_0 = metrics.precision_score(y, ypred,pos_label=0)
    recall = metrics.recall_score(y, ypred)
    recall_0 = metrics.recall_score(y, ypred,pos_label=0)
```

```
    f1score = metrics.f1_score(y, ypred)
    f1score_0 = metrics.f1_score(y, ypred,pos_label=0)
    accuracy_insample = model.score(X_train,y_train)
    return
[accuracy,auc_score,precision,recall,f1score,precision_0,recall_0,f1score_0,accura
cy_insample]

result_rf_upsample =
pd.DataFrame([],index=['accuracy','auc_score','precision','recall','f1score','prec
ision_0','recall_0','f1score_0','accuracy_insample'])
result_rf_upsample['Decision_Boundary_0.9'] =
get_metrics_threshold(model,X_test,y_test,0.9)
result_rf_upsample['Decision_Boundary_0.8'] =
get_metrics_threshold(model,X_test,y_test,0.8)
result_rf_upsample['Decision_Boundary_0.7'] =
get_metrics_threshold(model,X_test,y_test,0.7)
result_rf_upsample['Decision_Boundary_0.6'] =
get_metrics_threshold(model,X_test,y_test,0.6)
result_rf_upsample['Decision_Boundary_0.5'] =
get_metrics_threshold(model,X_test,y_test,0.5)
result_rf_upsample['Decision_Boundary_0.4'] =
get_metrics_threshold(model,X_test,y_test,0.4)
result_rf_upsample['Decision_Boundary_0.3'] =
get_metrics_threshold(model,X_test,y_test,0.3)
result_rf_upsample['Decision_Boundary_0.2'] =
get_metrics_threshold(model,X_test,y_test,0.2)
result_rf_upsample['Decision_Boundary_0.1'] =
get_metrics_threshold(model,X_test,y_test,0.1)


# In[72]:


result_rf


# In[73]:


result_rf_upsample


# In[70]:


#output the final y_predict
```

```
y_pred = np.where(model.predict_proba(X_test)[:,1]>=0.5,1,0)


# In[71]:


pd.DataFrame(y_pred).value_counts()


# In[72]:


from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)


# In[73]:


model.best_estimator_.get_params()


# In[ ]:


feature_names = [i for i in x]
importances = model.best_estimator_.feature_importances_
feature_importances = pd.DataFrame({"features" : feature_names,
 "importances" : importances})
sorted_feature_importances =
feature_importances.sort_values('importances',axis=0,ascending=False)
sorted_feature_importances.to_csv(r'rf_feature_importance.csv')
sorted_feature_importances.head(10)


# In[ ]:


import numpy as np
import matplotlib.pyplot as plt

y=list(feature_importances.sort_values('importances',axis=0,ascending=False).head(
10)['importances'])
feature_names_new=list(feature_importances.sort_values('importances',axis=0,ascend
ing=False).head(10)['features'])

fig, ax = plt.subplots()
```

```
b = ax.barh(range(len(feature_names_new)), y, color='#6699CC')
ax.set_yticks(range(len(feature_names_new)))
ax.set_yticklabels(feature_names_new)


# ## 5. Optimization
# Our objective is to minimize total cost of retaining this company's current workforce. The total cost comes from two aspects, one is the cost of raising fixed annual base salary to prevent employees from leaving, the other is cost of hiring new employees to fill the vacancy.
#
# Assumptions:
# 1. The total cost of hiring a new employee is $16999. (see the example cost breakdown for more details)
# 2. The cost of keeping an leaving employee is propotional to his/her annual base salary, normally ranging from 10% to 20%. Here we will analyze under three common scenairos: 10%, 15% and 20% and solve the minimization problem.
# 3. **Variant1**: We consider adding a hypothetical constraint, assuming the company aims to maintain an employee turnover rate that's no more than 10% while minimizing the total employment cost.
# 4. **Variant2**: We consider adding a hypothetical constraint, assuming the company aims to maintain a Middle Level employee turnover rate that's no more than 15% while minimizing the total employment cost.
#
# ### 5.1 Model Formulation
# Define base pay raise cost as $rc_i$, hiring cost as $hc_i$, number of predicted leavers is $N$, number of predicted Middle level leavers is $M$, binary decision variable $x_{1i}$ indicates whether to raise base pay or not for employee $i$, binary decision variable $x_{2i}$ indicates whether to hire new employee
#
# Objective function
#
# $$\min _{x_1,x_2} \sum_{i=1}^n rc_i x_{1i} + hc_i x_{2i}$$
# - $\sum_{i=1}^n rc_i x_{1i}$: The total cost for raising fixed salary to keep employees
# - $\sum_{i=1}^n hc_i x_{2i}$: The total cost for hiring new employees
#
# Constrants
# $$x_{1i} + x_{2i} = 1 \quad \forall i \in N$$
# $$x_{1i}, x_{2i}\in \{0,1\} \quad \forall i \in N$$
#
# Variant1 Constrant:
# $$\frac{1}{N} \sum_{i=1}^n x_{2i} \le 0.1 \quad \forall i \in N$$
#
# Variant2 Constrant:
```

```
# $$\frac{1}{M} \sum_{i=1}^m x_{2i} \le 0.15 \quad \forall i \in M$$

# In[262]:


# best RF model
model = RandomForestClassifier(bootstrap=False,
                               criterion='gini',
                               max_depth=10,
                               min_samples_split=2,
                               n_estimators=1000,
                               random_state=123,
                               verbose=0)
model.fit(X_train, y_train.ravel())
model.score(X_train,y_train)


# In[263]:


# original train test split
x = one_hot_encoded_training_predictors.drop(['Leavers'],axis=1)
y = one_hot_encoded_training_predictors['Leavers']

from sklearn.model_selection import StratifiedShuffleSplit
sss=StratifiedShuffleSplit(n_splits=5,test_size=0.3,random_state=123)

for train_index, test_index in sss.split(df1, df1["Leavers"]):
    X_train = x.iloc[train_index]
    y_train = y.iloc[train_index]
    X_test = x.iloc[test_index]
    y_test = y.iloc[test_index]


# In[264]:


# predict original test set
ypred = np.where(model.predict_proba(X_test)[:,1]>=0.5,1,0)


# In[267]:


sum(ypred) # number of predicted leavers
```

```
# In[268]:


indexs = X_test.index.values # get indexes of test set rows


# In[269]:


df_test = df_raw.iloc[indexs]
df_test['Leavers_preds'] = ypred


# In[270]:


# get fixed salary of all leavers in test set
salary = df_test[df_test['Leavers_preds'] == 1]["Fixed_Salary"].values
n = len(salary)


# In[271]:


df_test.head()


# In[272]:


df_test_leavers = df_test[df_test['Leavers_preds'] == 1]
df_test_leavers.reset_index(inplace = True)


# In[273]:


# turnover by level before optimization
for hl in df_test["Hierachy_Level"].unique():
    print("{0} Level Turnover:  {1:0.2f}%".format(hl,
100*sum(df_test_leavers["Hierachy_Level"] == hl)/sum(df_test["Hierachy_Level"] ==
hl)))


# In[274]:
```

```
# after optimization without additional constraint,
# the turnover rate for Middle will be reduced to 20% below
# the turnover rate for Middle will be reduced to about 15% below


# In[275]:


sum(df_test["Hierachy_Level"] == "Middle") # 83 middle
sum(df_test["Hierachy_Level"] == "Low") # 207 Low


# In[276]:


m_index = df_test_leavers[df_test_leavers["Hierachy_Level"] ==
"Middle"].index.values # get indexes of middle level leavers
l_index = df_test_leavers[df_test_leavers["Hierachy_Level"] == "Low"].index.values
# get indexes of low level leavers


# ### 5.2 Benchmark

# In[277]:


# All hire new employee
hc = 16999
hiring_cost = hc*n
hiring_cost


# In[278]:


# All raise base salary
raise_prop = np.array([0.1, 0.15, 0.2]) # raise = 10%, 15%, 20%
raise_cost = [round(p*sum(salary),2) for p in raise_prop]
raise_cost


# ### 5.3 Scenario Analysis
# #### Global Minimum

# In[307]:
```

```
total_cost = np.zeros(3)
total_rc = np.zeros(3)
total_hc = np.zeros(3)
n_stayer = np.zeros(3)
n_leaver = np.zeros(3)
for j in range(len(raise_prop)):
    model = pe.ConcreteModel()
    model.x1 = pe.Var(range(n), domain=pe.Binary)# indicate raise pay or not
    model.x2 = pe.Var(range(n), domain=pe.Binary)# indicate hire new or not
    model.constraint = pe.Constraint(range(n), rule = lambda m, i: model.x1[i] +
model.x2[i] == 1)
    model.cost = pe.Objective(
        expr=sum(raise_prop[j]*salary[i]*model.x1[i]+ hc*model.x2[i] for i in
range(n)))

    results = solver.solve(model)
    total_cost[j] = model.cost()
    n_stayer[j] = sum([model.x1[i]() for i in range(n)])
    n_leaver[j] = sum([model.x2[i]() for i in range(n)])
    total_rc[j] = sum([raise_prop[j]*salary[i]*model.x1[i]() for i in range(n)])
    total_hc[j] = sum([hc*model.x2[i]() for i in range(n)])


# In[280]:


# sum([model.x2[i]() for i in m_index])
# sum([model.x2[i]() for i in l_index])


# In[317]:


# optimal decision variables
for i in range(3):
    print("Under {} pay raise, {} existing employees get pay raise, {} new
employees hired".format(raise_prop[i], round(n_stayer[i]), round(n_leaver[i])))


# In[308]:


df = pd.DataFrame({'pay raise proportion': raise_prop,
                   'pay raise cost': np.round(total_rc),
                   'hiring cost': np.round(total_hc)})
df
```

```
# In[282]:


df.set_index('pay raise proportion').plot(kind='bar', stacked=True)
plt.title('Optimized Total Cost Breakdown', fontsize=16)
plt.xlabel('base pay raise')
plt.ylabel('cost')
plt.legend(loc = "upper left")
plt.xticks(rotation=45)


# In[289]:


df2 = pd.DataFrame({
    "pay raise proportion": raise_prop,
    "All Hire New Employees": [hiring_cost, hiring_cost, hiring_cost],
    "All Raise Base Salary": raise_cost,
    "Optimization": np.round(total_cost,2)
    })
df2


# In[284]:


df2.set_index('pay raise proportion').plot(kind='bar')
plt.title('Total Cost', fontsize=16)
plt.xlabel('base pay raise')
plt.ylabel('cost')
plt.xticks(rotation=45)


# In[285]:


# cost reduction compared to benchmark all hire new employee
print('average cost reduction compared to benchmark all hire new employee')
print('{0:0.2f}%\n'.format(100*np.mean([round((hiring_cost -
total_cost[i])/hiring_cost,4) for i in range(3)])))

for i in range(len(raise_prop)):
```

```
    print('Base Pay Raise Level: {0} , Cost Reduction:
{1:0.2f}%'.format(raise_prop[i], 100*round((hiring_cost -
total_cost[i])/hiring_cost,4)))


# In[286]:


# cost reduction compared to benchmark all raise base pay
print('average cost reduction compared to benchmark all raise base pay')
print('{0:0.2f}%\n'.format(100*np.mean([round((raise_cost[i] -
total_cost[i])/raise_cost[i],4) for i in range(3)])))

for i in range(len(raise_prop)):
    print('Base Pay Raise Level: {0} , Cost Reduction:
{1:0.2f}%'.format(raise_prop[i], 100*round((raise_cost[i] -
total_cost[i])/raise_cost[i],4)))


# In[287]:


# employee turnover rate
print('average employee turnover rate:
{0:0.2f}%\n'.format(100*np.mean([round(n_leaver[i]/len(X_test),4) for i in
range(3)])))

for i in range(len(raise_prop)):
    print('Base Pay Raise Level: {0} , employee turnover rate:
{1:0.2f}%'.format(raise_prop[i], 100*round(n_leaver[i]/len(X_test),4)))


# In[290]:


# employee turnover rate
df3 = pd.DataFrame({
    "pay raise proportion": raise_prop,
    "employee turnover": [round(n_leaver[i]/len(X_test),4) for i in range(3)]
    })
df3


# #### Variant1: Overall Employee Turnover Rate <= 10%

# In[318]:
```

```
total_cost1 = np.zeros(3)
total_rc1 = np.zeros(3)
total_hc1 = np.zeros(3)
n_stayer1 = np.zeros(3)
n_leaver1 = np.zeros(3)
for j in range(len(raise_prop)):
    model = pe.ConcreteModel()
    model.x1 = pe.Var(range(n), domain=pe.Binary)# indicate raise pay or not
    model.x2 = pe.Var(range(n), domain=pe.Binary)# indicate hire new or not
    model.constraint = pe.Constraint(range(n), rule = lambda m, i: model.x1[i] +
model.x2[i] == 1)
    model.constraint2 = pe.Constraint(expr = sum(model.x2[i] for i in range(n)) <=
30)

    model.cost = pe.Objective(
        expr=sum(raise_prop[j]*salary[i]*model.x1[i]+ hc*model.x2[i] for i in
range(n)))

    results = solver.solve(model)
    total_cost1[j] = model.cost()
    n_stayer1[j] = sum([model.x1[i]() for i in range(n)])
    n_leaver1[j] = sum([model.x2[i]() for i in range(n)])
    total_rc1[j] = sum([raise_prop[j]*salary[i]*model.x1[i]() for i in range(n)])
    total_hc1[j] = sum([hc*model.x2[i]() for i in range(n)])


# In[319]:


# optimal decision variables
for i in range(3):
    print("Under {} pay raise, {} existing employees get pay raise, {} new
employees hired".format(raise_prop[i], round(n_stayer1[i]), round(n_leaver1[i])))


# In[292]:


df_v1 = pd.DataFrame({'pay raise proportion': raise_prop,
                  'pay raise cost': np.round(total_rc1),
                  'hiring cost': np.round(total_hc1)})
df_v1
```

```
# In[293]:


df_v1.set_index('pay raise proportion').plot(kind='bar', stacked=True)
plt.title('Optimized Total Cost Breakdown', fontsize=16)
plt.xlabel('base pay raise')
plt.ylabel('cost')
plt.legend(loc = "upper left")
plt.xticks(rotation=45)


# In[294]:


df2_v1 = pd.DataFrame({
    "pay raise proportion": raise_prop,
    "All Hire New Employees": [hiring_cost, hiring_cost, hiring_cost],
    "All Raise Base Salary": raise_cost,
    "Optimization": np.round(total_cost1,2)
    })
df2_v1


# In[295]:


df2_v1.set_index('pay raise proportion').plot(kind='bar')
plt.title('Total Cost', fontsize=16)
plt.xlabel('base pay raise')
plt.ylabel('cost')
plt.xticks(rotation=45)


# In[296]:


# cost reduction compared to benchmark all hire new employee
print('average cost reduction compared to benchmark all hire new employee')
print('{0:0.2f}%\n'.format(100*np.mean([round((hiring_cost -
total_cost1[i])/hiring_cost,4) for i in range(3)])))

for i in range(len(raise_prop)):
    print('Base Pay Raise Level: {0} , Cost Reduction:
{1:0.2f}%'.format(raise_prop[i], 100*round((hiring_cost -
total_cost1[i])/hiring_cost,4)))
```

```
# In[297]:


# cost reduction compared to benchmark all raise base pay
print('average cost reduction compared to benchmark all raise base pay')
print('{0:0.2f}%\n'.format(100*np.mean([round((raise_cost[i] -
total_cost1[i])/raise_cost[i],4) for i in range(3)])))

for i in range(len(raise_prop)):
    print('Base Pay Raise Level: {0} , Cost Reduction:
{1:0.2f}%'.format(raise_prop[i], 100*round((raise_cost[i] -
total_cost1[i])/raise_cost[i],4)))


# #### Variant2: Middle Level Employee Turnover Rate <= 15%

# In[321]:


total_cost2 = np.zeros(3)
total_rc2 = np.zeros(3)
total_hc2 = np.zeros(3)
n_stayer2 = np.zeros(3)
n_leaver2 = np.zeros(3)
for j in range(len(raise_prop)):
    model = pe.ConcreteModel()
    model.x1 = pe.Var(range(n), domain=pe.Binary)# indicate raise pay or not
    model.x2 = pe.Var(range(n), domain=pe.Binary)# indicate hire new or not
    model.constraint = pe.Constraint(range(n), rule = lambda m, i: model.x1[i] +
model.x2[i] == 1)
    model.constraint2 = pe.Constraint(expr = sum(model.x2[i] for i in m_index) <=
12)

    model.cost = pe.Objective(
        expr=sum(raise_prop[j]*salary[i]*model.x1[i]+ hc*model.x2[i] for i in
range(n)))

    results = solver.solve(model)
    total_cost2[j] = model.cost()
    n_stayer2[j] = sum([model.x1[i]() for i in range(n)])
    n_leaver2[j] = sum([model.x2[i]() for i in range(n)])
    total_rc2[j] = sum([raise_prop[j]*salary[i]*model.x1[i]() for i in range(n)])
    total_hc2[j] = sum([hc*model.x2[i]() for i in range(n)])
```

```
# In[320]:


# optimal decision variables
for i in range(3):
    print("Under {} pay raise, {} existing employees get pay raise, {} new
employees hired".format(raise_prop[i], round(n_stayer2[i]), round(n_leaver2[i])))


# In[299]:


df_v2 = pd.DataFrame({'pay raise proportion': raise_prop,
                  'pay raise cost': np.round(total_rc2),
                  'hiring cost': np.round(total_hc2)})
df_v2


# In[300]:


df_v2.set_index('pay raise proportion').plot(kind='bar', stacked=True)
plt.title('Optimized Total Cost Breakdown', fontsize=16)
plt.xlabel('base pay raise')
plt.ylabel('cost')
plt.legend(loc = "upper left")
plt.xticks(rotation=45)


# In[301]:


df2_v2 = pd.DataFrame({
    "pay raise proportion": raise_prop,
    "All Hire New Employees": [hiring_cost, hiring_cost, hiring_cost],
    "All Raise Base Salary": raise_cost,
    "Optimization": np.round(total_cost2,2)
    })
df2_v2


# In[302]:


df2_v2.set_index('pay raise proportion').plot(kind='bar')
plt.title('Total Cost', fontsize=16)
```

```
plt.xlabel('base pay raise')
plt.ylabel('cost')
plt.xticks(rotation=45)


# In[303]:


# cost reduction compared to benchmark all hire new employee
print('average cost reduction compared to benchmark all hire new employee')
print('{0:0.2f}%\n'.format(100*np.mean([round((hiring_cost -
total_cost2[i])/hiring_cost,4) for i in range(3)])))

for i in range(len(raise_prop)):
    print('Base Pay Raise Level: {0} , Cost Reduction:
{1:0.2f}%'.format(raise_prop[i], 100*round((hiring_cost -
total_cost2[i])/hiring_cost,4)))


# In[304]:


# cost reduction compared to benchmark all raise base pay
print('average cost reduction compared to benchmark all raise base pay')
print('{0:0.2f}%\n'.format(100*np.mean([round((raise_cost[i] -
total_cost2[i])/raise_cost[i],4) for i in range(3)])))

for i in range(len(raise_prop)):
    print('Base Pay Raise Level: {0} , Cost Reduction:
{1:0.2f}%'.format(raise_prop[i], 100*round((raise_cost[i] -
total_cost2[i])/raise_cost[i],4)))
```