Human Resources Analytics - Milestone Report

Human Resources Analytics - Milestone Report

"Yeah, they all said that to me...", *Bob replied as we were at Starbucks sipping on our dark roast coffee. Bob is a friend of mine and was the owner of a multi-million dollar company, that's right, "m-i-l-l-i-o-n". He used to tell me stories about how his company's productivity and growth has sky rocketed from the previous years and everything has been going great. But recently, he's been noticing some decline within his company. In a five month period, he lost one-fifth of his employees. At least a dozen of them throughout each department made phone calls and even left sticky notes on their tables informing him about their leave. Nobody knew what was happening. In that year, he was contemplating about filing for bankruptcy. Fast-forward seven months later, he's having a conversation with his co-founder of the company. The conversation ends with, **"I quit..."**

That is the last thing anybody wants to hear from their employees. In a sense, it's the employees who make the company. It's the employees who do the work. It's the employees who shape the company's culture. Long-term success, a healthy work environment, and high employee retention are all signs of a successful company. But when a company experiences a high rate of employee turnover, then something is going wrong. This can lead the company to huge monetary losses by these innovative and valuable employees.

Companies that maintain a healthy organization and culture are always a good sign of future prosperity. Recognizing and understanding what factors that were associated with employee turnover will allow companies and individuals to limit this from happening and may even increase employee productivity and growth. These predictive insights give managers the opportunity to take corrective steps to build and preserve their successful business.

"You don't build a business. You build people, and people build the business." - Zig Ziglar

---

About This Kernel

Feel free to use this kernel as a reference as a template for your analysis :)

For those that are in or interested in **Human Resources** and would like a detailed guide on how to approach an **employee retention** problem through a **data science** point of view, feel free to check this notebook out..

I will be covering my analysis and approach through different process flows in the data science pipeline, which includes statistical inference and exploratory data analysis. The main goal is to understand the reasoning behind employee turnover and to come up with a model to classify an employee's risk of attrition. A recommendation for a retention plan was created, which incorporates some best practices for employee retention at different risk levels of attrition.

Hopefully the kernel added some new insights/perspectives to the data science community! If there are any suggestions/changes you would like to see in the Kernel please let me know :). Appreciate every ounce of help!

This notebook will always be a work in progress. Please leave any comments about further improvements to the notebook! Any feedback or constructive criticism is greatly appreciated!. Thank you guys!

UPDATE: R Version

R Users:

Thanks to Ragul, he has created a similar kernel but using R. Check it out if you are an R user!

https://www.kaggle.com/ragulram/hr-analytics-exploration-and-modelling-with-r

Business Problem

Bob's multi-million dollar company is about to go bankrupt and he wants to know why his employees are leaving.

Client

Bob the Boss

Objective

The company wants to understand what factors contributed most to employee turnover and to create a model that can predict if a certain employee will leave the company or not. The goal is to create or improve different retention strategies on targeted employees. Overall, the implementation of this model will allow management to create better decision-making actions.

OSEMN Pipeline

I'll be following a typical data science pipeline, which is call "OSEMN" (pronounced awesome).

1. **O**btaining the data is the first approach in solving the problem.

2. **S**crubbing or cleaning the data is the next step. This includes data imputation of missing or invalid data and fixing column names.
3. **E**xploring the data will follow right after and allow further insight of what our dataset contains. Looking for any outliers or weird data. Understanding the relationship each explanatory variable has with the response variable resides here and we can do this with a correlation matrix.
4. **M**odeling the data will give us our predictive power on whether an employee will leave.
5. I**N**terpreting the data is last. With all the results and analysis of the data, what conclusion is made? What factors contributed most to employee turnover? What relationship of variables were found?

Note: *The data was found from the "Human Resources Analytics" dataset provided by Kaggle's website.*
https://www.kaggle.com/ludobenistant/hr-analytics

Note: THIS DATASET IS **SIMULATED**.

Part 1: Obtaining the Data

In [1]:

```
# Import the neccessary modules for data manipulation and visual representation
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as matplot
import seaborn as sns
%matplotlib inline
```

In [2]:

```
#Read the analytics csv file and store our dataset into a dataframe called "df"
df = pd.DataFrame.from_csv('../input/HR_comma_sep.csv', index_col=None)
```

Part 2: Scrubbing the Data

Typically, cleaning the data requires a lot of work and can be a very tedious procedure. This dataset from Kaggle is super clean and contains no missing values. But still, I will have to examine the dataset to make sure that everything else is readable and that the observation values match the feature names appropriately.

In [3]:

```
# Check to see if there are any missing values in our data set

df.isnull().any()
```

Out[3]:

```
satisfaction_level       False
last_evaluation          False
number_project           False
average_montly_hours     False
time_spend_company       False
Work_accident            False
left                     False
promotion_last_5years    False
sales                    False
salary                   False
dtype: bool
```

In [4]:

 # Get a quick overview of what we are dealing with in our dataset

```
df.head()
```

Out[4]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident
0	0.38	0.53	2	157	3	0
1	0.80	0.86	5	262	6	0
2	0.11	0.88	7	272	4	0
3	0.72	0.87	5	223	5	0
4	0.37	0.52	2	159	3	0

In [5]:

 # Renaming certain columns for better readability

```
df = df.rename(columns={'satisfaction_level': 'satisfaction',
                        'last_evaluation': 'evaluation',
```

```
'number_project': 'projectCount',

'average_montly_hours': 'averageMonthlyHours',

'time_spend_company': 'yearsAtCompany',

'Work_accident': 'workAccident',

'promotion_last_5years': 'promotion',

'sales' : 'department',

'left' : 'turnover'

})
```

In [6]:

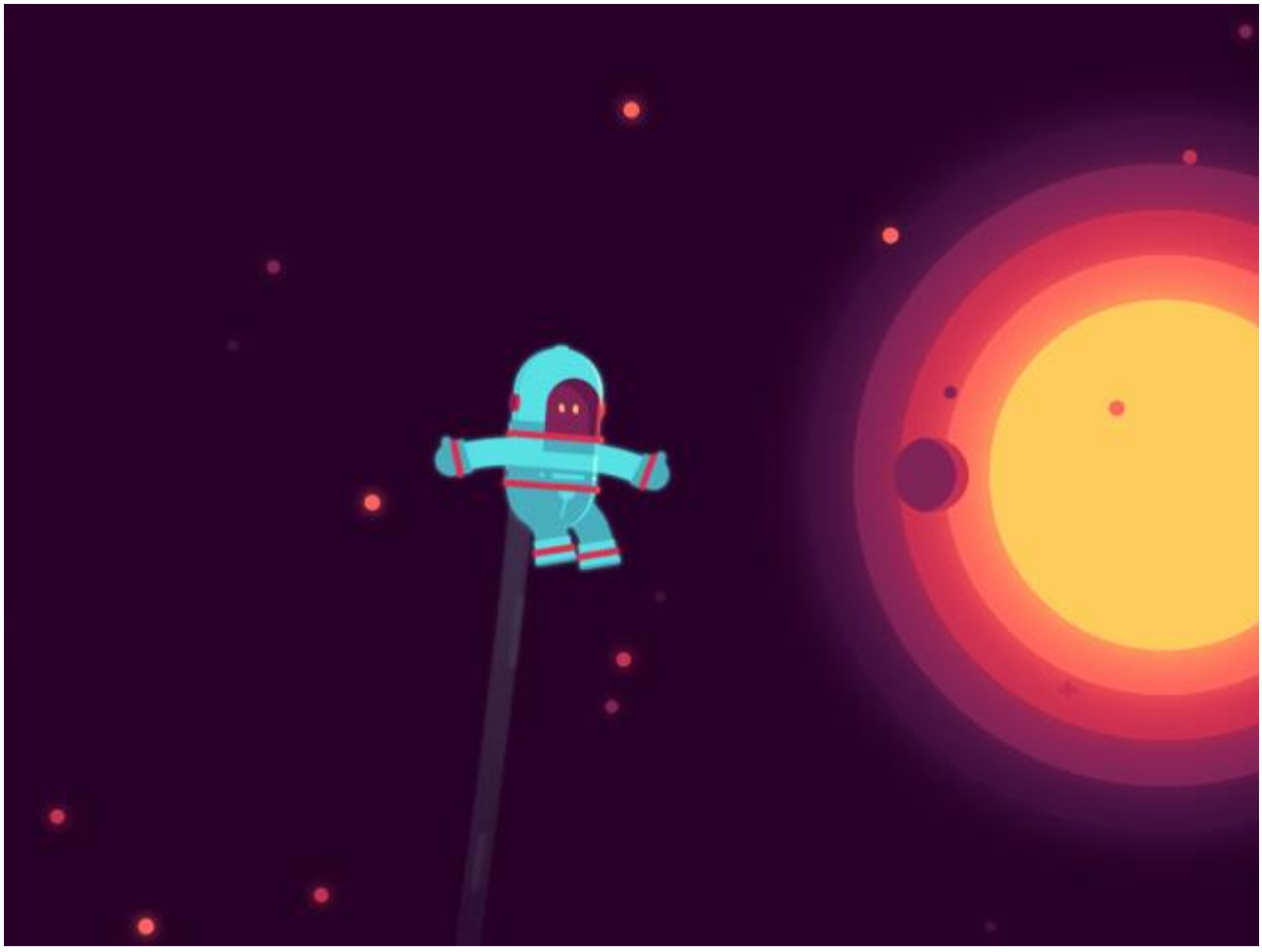
```
# Move the reponse variable "turnover" to the front of the table

front = df['turnover']

df.drop(labels=['turnover'], axis=1,inplace = True)

df.insert(0, 'turnover', front)

df.head()
```

Out[6]:

	turnover	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany	workAccident	promotio
0	1	0.38	0.53	2	157	3	0	0
1	1	0.80	0.86	5	262	6	0	0

2	1	0.11	0.88	7	272	4	0	0
3	1	0.72	0.87	5	223	5	0	0
4	1	0.37	0.52	2	159	3	0	0

Part 3: Exploring the Data



3a. Statistical Overview

The dataset has:

- About 15,000 employee observations and 10 features
- The company had a turnover rate of about 24%
- Mean satisfaction of employees is 0.61

In [7]:

```
# The dataset contains 10 columns and 14999 observations

df.shape
```

Out[7]:

(14999, 10)

In [8]:

```
# Check the type of our features.

df.dtypes
```

Out[8]:

```
turnover                int64
satisfaction          float64
evaluation            float64
projectCount            int64
averageMonthlyHours     int64
yearsAtCompany          int64
workAccident            int64
promotion               int64
```

department object

salary object

dtype: object

In [9]:

```
# Looks like about 76% of employees stayed and 24% of employees left.

# NOTE: When performing cross validation, its important to maintain this turnover ratio

turnover_rate = df.turnover.value_counts() / len(df)

turnover_rate
```

Out[9]:

0 0.761917

1 0.238083

Name: turnover, dtype: float64

In [10]:

```
# Display the statistical overview of the employees

df.describe()
```

Out[10]:

	turnover	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany	workAccident	promotion
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000

mean	0.238083	0.612834	0.716102	3.803054	201.050337	3.498233	0.144610	0.021268
std	0.425924	0.248631	0.171169	1.232592	49.943099	1.460136	0.351719	0.144281
min	0.000000	0.090000	0.360000	2.000000	96.000000	2.000000	0.000000	0.000000
25%	0.000000	0.440000	0.560000	3.000000	156.000000	3.000000	0.000000	0.000000
50%	0.000000	0.640000	0.720000	4.000000	200.000000	3.000000	0.000000	0.000000
75%	0.000000	0.820000	0.870000	5.000000	245.000000	4.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	7.000000	310.000000	10.000000	1.000000	1.000000

In [11]:

```
# Overview of summary (Turnover V.S. Non-turnover)

turnover_Summary = df.groupby('turnover')

turnover_Summary.mean()
```

Out[11]:

	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany	workAccident	promotion
turnover							
0	0.666810	0.715473	3.786664	199.060203	3.380032	0.175009	0.026251
1	0.440098	0.718113	3.855503	207.419210	3.876505	0.047326	0.005321

3b. Correlation Matrix & Heatmap

Moderate Positively Correlated Features:

- projectCount vs evaluation: 0.349333
- projectCount vs averageMonthlyHours: 0.417211
- averageMonthlyHours vs evaluation: 0.339742

Moderate Negatively Correlated Feature:

- satisfaction vs turnover: -0.388375

Stop and Think:

- What features affect our target variable the most (turnover)?
- What features have strong correlations with each other?
- Can we do a more in depth examination of these features?

Summary:

From the heatmap, there is a **positive(+)** correlation between projectCount, averageMonthlyHours, and evaluation. Which could mean that the employees who spent more hours and did more projects were evaluated highly.

For the **negative(-)** relationships, turnover and satisfaction are highly correlated. I'm assuming that people tend to leave a company more when they are less satisfied.

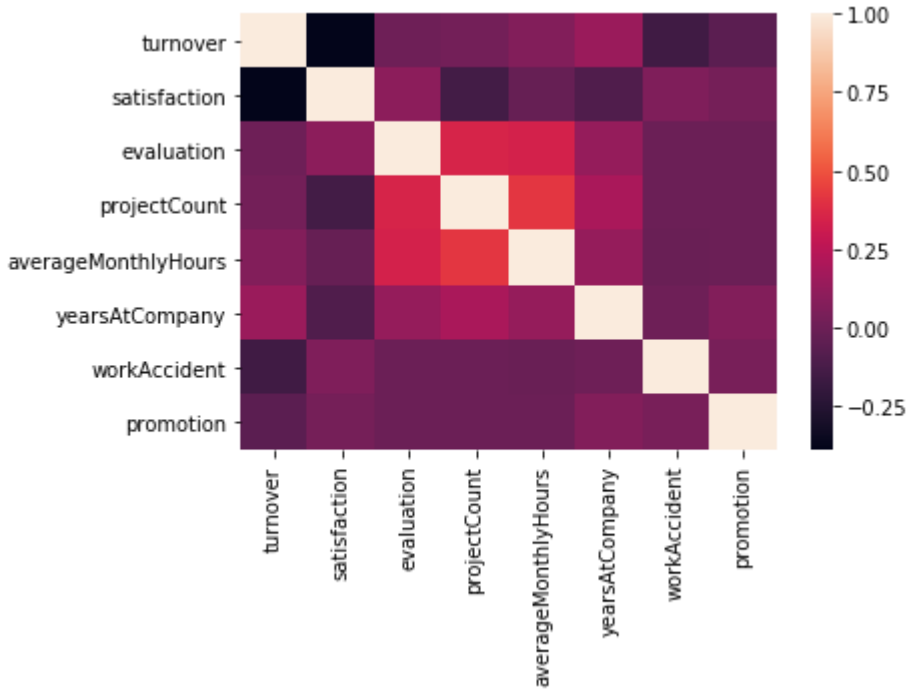
In [12]:

```
#Correlation Matrix
corr = df.corr()
corr = (corr)
sns.heatmap(corr,
        xticklabels=corr.columns.values,
        yticklabels=corr.columns.values)


corr
```

Out[12]:

	turnover	satisfaction	evaluation	projectCount	averageMonthlyHours	yearsAtCompany	workAccident	promotion
turnover	1.000000	-0.388375	0.006567	0.023787	0.071287	0.144822	-0.154622	-0.061788

satisfaction	-0.388375	1.000000	0.105021	-0.142970	-0.020048	-0.100866	0.058697	0.025605
evaluation	0.006567	0.105021	1.000000	0.349333	0.339742	0.131591	-0.007104	-0.008684
projectCount	0.023787	-0.142970	0.349333	1.000000	0.417211	0.196786	-0.004741	-0.006064
averageMonthlyHours	0.071287	-0.020048	0.339742	0.417211	1.000000	0.127755	-0.010143	-0.003544
yearsAtCompany	0.144822	-0.100866	0.131591	0.196786	0.127755	1.000000	0.002120	0.067433
workAccident	-0.154622	0.058697	-0.007104	-0.004741	-0.010143	0.002120	1.000000	0.039245
promotion	-0.061788	0.025605	-0.008684	-0.006064	-0.003544	0.067433	0.039245	1.000000



3b2. Statistical Test for Correlation

One-Sample T-Test (Measuring Satisfaction Level)

A one-sample t-test checks whether a sample mean differs from the population mean. Since satisfaction has the highest correlation with our dependent variable turnover, let's test to see whether the average satisfaction level of employees that had a turnover differs from the those that had no turnover.

Hypothesis Testing: Is there significant difference in the **means of satisfaction level** between employees who had a turnover and temployees who had no turnover?

- **Null Hypothesis:** *(H0: pTS = pES)* The null hypothesis would be that there is **no** difference in satisfaction level between employees who did turnover and those who did not..
- **Alternate Hypothesis:** *(HA: pTS != pES)* The alternative hypothesis would be that there **is** a difference in satisfaction level between employees who did turnover and those who did not..

In [13]:

```
# Let's compare the means of our employee turnover satisfaction against the employee population satisfaction

#emp_population = df['satisfaction'].mean()

emp_population = df['satisfaction'][df['turnover'] == 0].mean()

emp_turnover_satisfaction = df[df['turnover']==1]['satisfaction'].mean()


print( 'The mean satisfaction for the employee population with no turnover is: ' + str(emp_population))

print( 'The mean satisfaction for employees that had a turnover is: ' + str(emp_turnover_satisfaction) )
```

```
The mean satisfaction for the employee population with no turnover is: 0.666809590479516

The mean satisfaction for employees that had a turnover is: 0.44009801176140917
```

Conducting the T-Test

Let's conduct a t-test at **95% confidence level** and see if it correctly rejects the null hypothesis that the sample comes from the same distribution as the employee population. To conduct a one sample t-test, we can use the **stats.ttest_1samp()** function:

In [14]:

```
import scipy.stats as stats

stats.ttest_1samp(a=  df[df['turnover']==1]['satisfaction'], # Sample of Employee satisfaction who had a Turnover

                  popmean = emp_population)  # Employee Who Had No Turnover satisfaction mean
```

Out[14]:

```
Ttest_1sampResult(statistic=-51.3303486754725, pvalue=0.0)
```

T-Test Result

The test result shows the **test statistic "t" is equal to -51.33**. This test statistic tells us how much the sample mean deviates from the null hypothesis. If the t-statistic lies **outside** the quantiles of the t-distribution corresponding to our confidence level and degrees of freedom, we reject the null hypothesis. We can check the quantiles with **stats.t.ppf()**:

T-Test Quantile

If the t-statistic value we calculated above **(-51.33)** is outside the quantiles, then we can reject the null hypothesis

In [15]:

```
degree_freedom = len(df[df['turnover']==1])


LQ = stats.t.ppf(0.025,degree_freedom)  # Left Quartile


RQ = stats.t.ppf(0.975,degree_freedom)  # Right Quartile


print ('The t-distribution left quartile range is: ' + str(LQ))

print ('The t-distribution right quartile range is: ' + str(RQ))
```

```
The t-distribution left quartile range is: -1.9606285216

The t-distribution right quartile range is: 1.9606285216
```

One-Sample T-Test Summary

T-Test = -51.33 | P-Value = 0.000_ | Reject Null Hypothesis

Question: How come the P-Value is literally 0.0? Can anybody answer this?

Reject the null hypothesis because:

- T-Test score is outside the quantiles
- P-value is lower than confidence level of 5%

Based on the statistical analysis of a one sample t-test, there seems to be some significant difference between the mean satisfaction of employees who had a turnover and the entire employee population. The super low P-value of **0.00_** at a 5% confidence level is a good indicator to **reject the null hypothesis**.

But this does not neccessarily mean that there is practical significance. We would have to conduct more experiments or maybe collect more data about the employees in order to come up with a more accurate finding.

3c. Distribution Plots (Satisfaction - Evaluation - AverageMonthlyHours)

Summary: Let's examine the distribution on some of the employee's features. Here's what I found:

- **Satisfaction** - There is a huge spike for employees with low satisfaction and high satisfaction.
- **Evaluation** - There is a bimodal distrubtion of employees for low evaluations (less than 0.6) and high evaluations (more than 0.8)
- **AverageMonthlyHours** - There is another bimodal distribution of employees with lower and higher average monthly hours (less than 150 hours & more than 250 hours)
- The evaluation and average monthly hour graphs both share a similar distribution.
- Employees with lower average monthly hours were evaluated less and vice versa.
- If you look back at the correlation matrix, the high correlation between evaluation and averageMonthlyHours does support this finding.

Stop and Think:

- Is there a reason for the high spike in low satisfaction of employees?
- Could employees be grouped in a way with these features?
- Is there a correlation between evaluation and averageMonthlyHours?

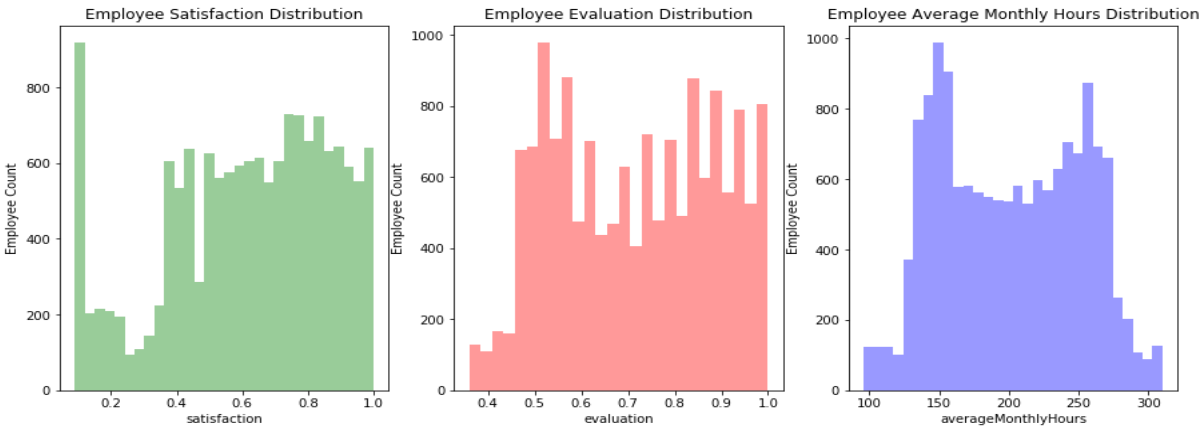
In [16]:

```
# Set up the matplotlib figure

f, axes = plt.subplots(ncols=3, figsize=(15, 6))


# Graph Employee Satisfaction

sns.distplot(df.satisfaction, kde=False, color="g", ax=axes[0]).set_title('Employee Satisfaction Distribution')

axes[0].set_ylabel('Employee Count')


# Graph Employee Evaluation

sns.distplot(df.evaluation, kde=False, color="r", ax=axes[1]).set_title('Employee Evaluation Distribution')
```

```
axes[1].set_ylabel('Employee Count')


# Graph Employee Average Monthly Hours

sns.distplot(df.averageMonthlyHours, kde=False, color="b", ax=axes[2]).set_title('Employee Average Monthly Hours Distribution')

axes[2].set_ylabel('Employee Count')
```

Out[16]:

```
Text(0,0.5,'Employee Count')
```



3d. Salary V.S. Turnover

Summary: This is not unusual. Here's what I found:

- Majority of employees who left either had **low** or **medium** salary.
- Barely any employees left with **high** salary
- Employees with low to average salaries tend to leave the company.

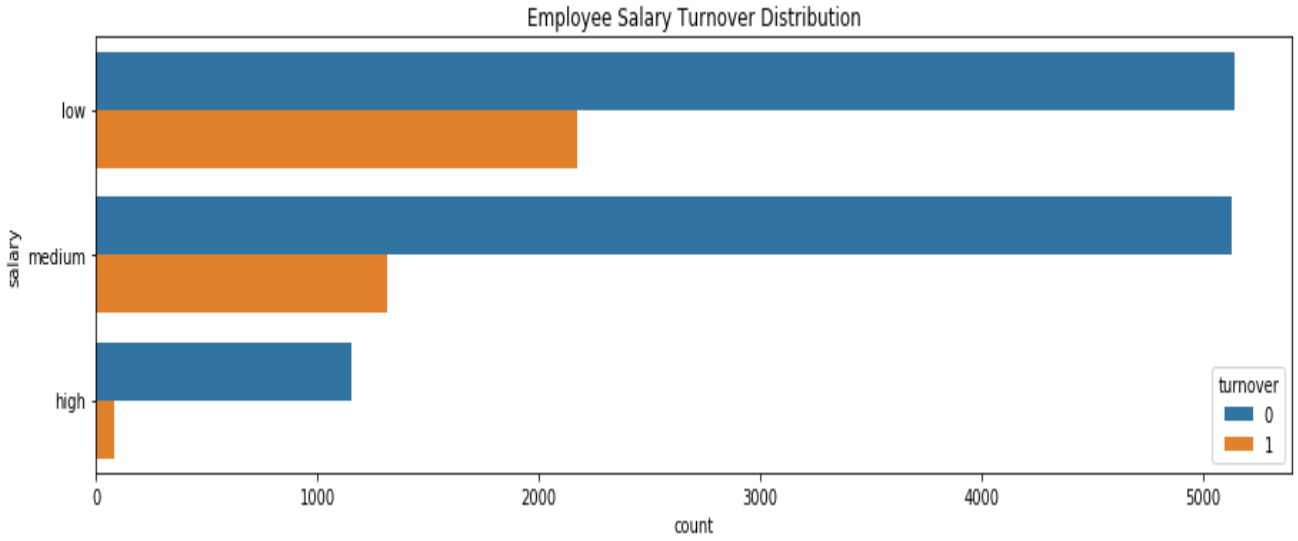
Stop and Think:

- What is the work environment like for low, medium, and high salaries?

- What made employees with high salaries to leave?

In [17]:

```
f, ax = plt.subplots(figsize=(15, 4))

sns.countplot(y="salary", hue='turnover', data=df).set_title('Employee Salary Turnover Distribution');
```



3e. Department V.S. Turnover

Summary: Let's see more information about the departments. Here's what I found:

- The **sales, technical, and support department** were the top 3 departments to have employee turnover
- The management department had the smallest amount of turnover

Stop and Think:

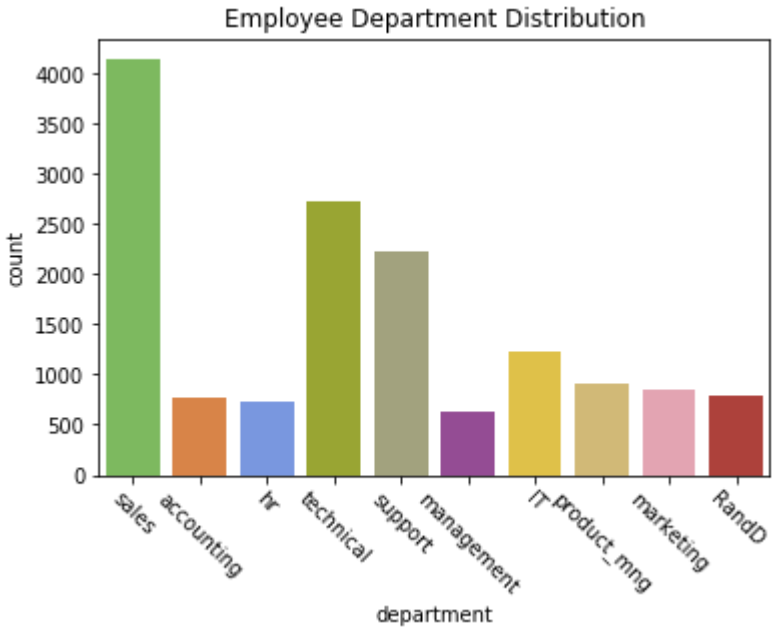
- If we had more information on each department, can we pinpoint a more direct cause for employee turnover?

In [18]:

```
# Employee distri

# Types of colors

color_types = ['#78C850','#F08030','#6890F0','#A8B820','#A8A878','#A040A0','#F8D030',

        '#E0C068','#EE99AC','#C03028','#F85888','#B8A038','#705898','#98D8D8','#7038F8']


# Count Plot (a.k.a. Bar Plot)

sns.countplot(x='department', data=df, palette=color_types).set_title('Employee Department Distribution');


# Rotate x-labels

plt.xticks(rotation=-45)
```

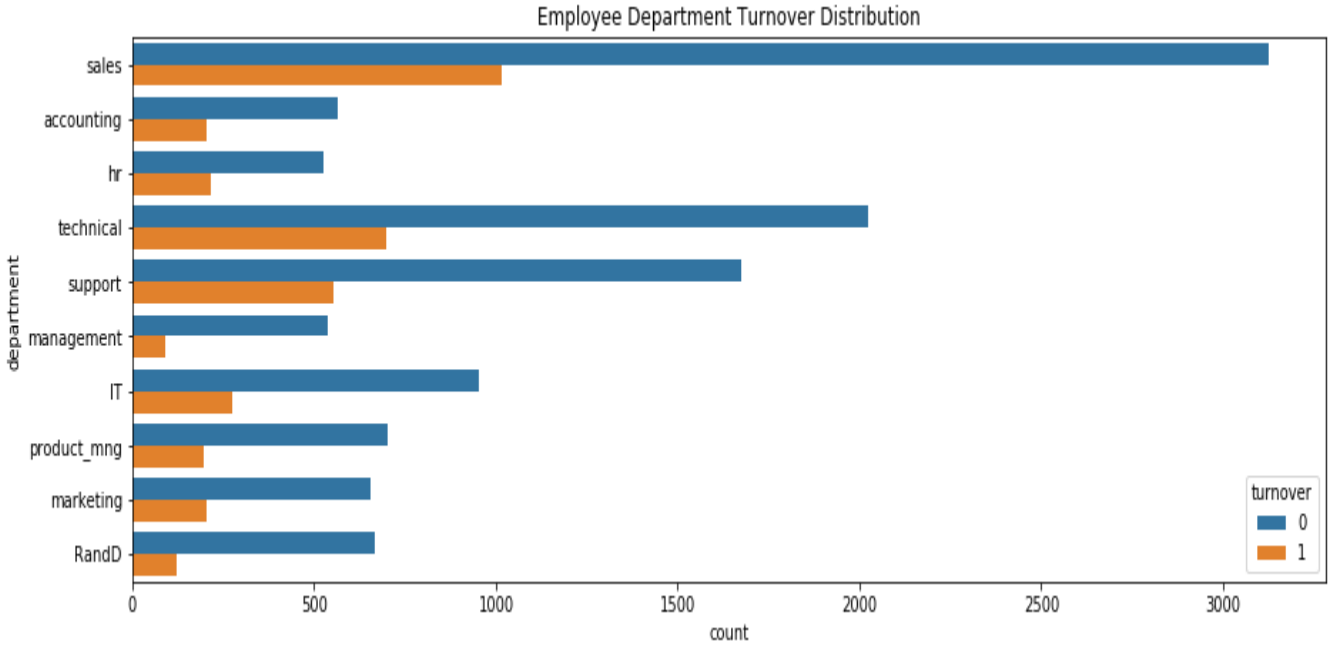
Out[18]:

(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), <a list of 10 Text xticklabel objects>)



In [19]:

```
f, ax = plt.subplots(figsize=(15, 5))

sns.countplot(y="department", hue='turnover', data=df).set_title('Employee Department Turnover Distribution');
```



3f. Turnover V.S. ProjectCount

Summary: This graph is quite interesting as well. Here's what I found:

- More than half of the employees with **2,6, and 7** projects left the company
- Majority of the employees who did not leave the company had **3,4, and 5** projects
- All of the employees with **7** projects left the company
- There is an increase in employee turnover rate as project count increases

Stop and Think:

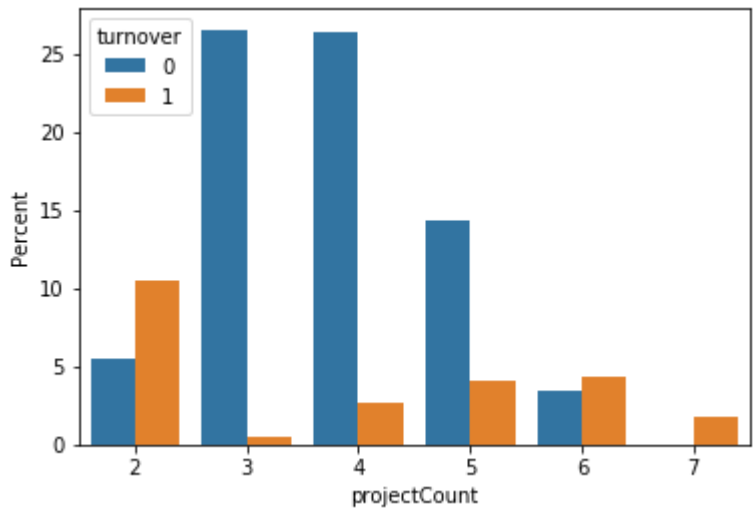
- Why are employees leaving at the lower/higher spectrum of project counts?
- Does this means that employees with project counts 2 or less are not worked hard enough or are not highly valued, thus leaving the company?
- Do employees with 6+ projects are getting overworked, thus leaving the company?

In [20]:

```
ax = sns.barplot(x="projectCount", y="projectCount", hue="turnover", data=df, estimator=lambda x: len(x) / len(df) * 100)

ax.set(ylabel="Percent")
```

Out[20]:

```
[Text(0,0.5,'Percent')]
```



3g. Turnover V.S. Evaluation

Summary:

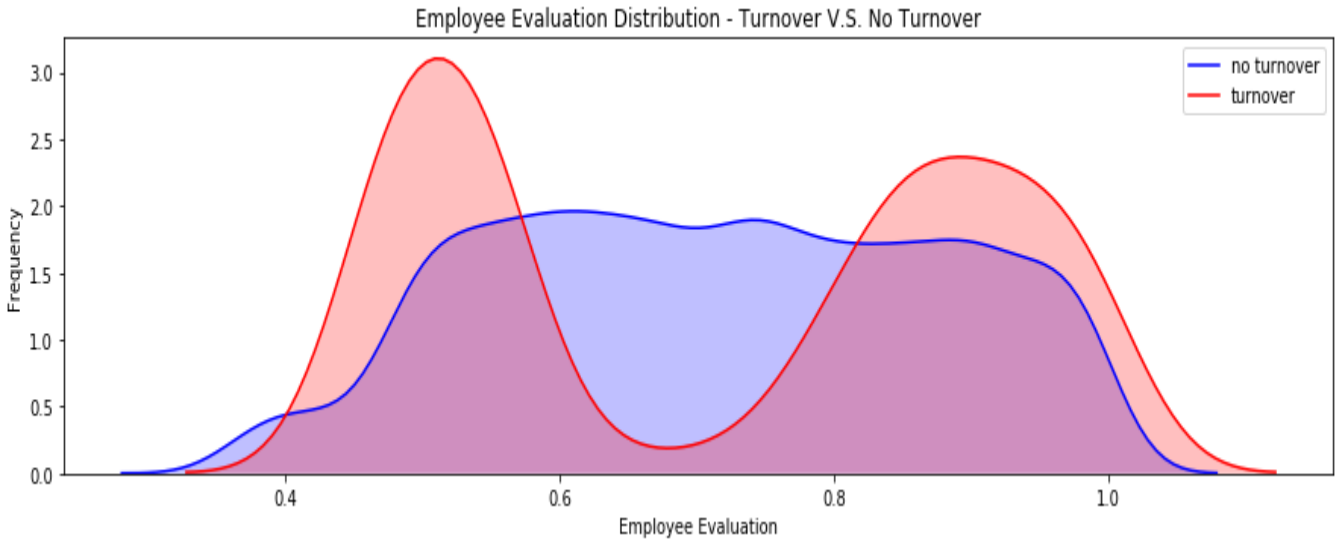
- There is a biomodal distribution for those that had a turnover.
- Employees with **low** performance tend to leave the company more
- Employees with **high** performance tend to leave the company more
- The **sweet spot** for employees that stayed is within **0.6-0.8** evaluation

In [21]:

```
# Kernel Density Plot

fig = plt.figure(figsize=(15,4),)

ax=sns.kdeplot(df.loc[(df['turnover'] == 0),'evaluation'] , color='b',shade=True,label='no turnover')

ax=sns.kdeplot(df.loc[(df['turnover'] == 1),'evaluation'] , color='r',shade=True, label='turnover')

ax.set(xlabel='Employee Evaluation', ylabel='Frequency')

plt.title('Employee Evaluation Distribution - Turnover V.S. No Turnover')
```

Out[21]:

```
Text(0.5,1,'Employee Evaluation Distribution - Turnover V.S. No Turnover')
```



3h. Turnover V.S. AverageMonthlyHours

Summary:

- Another bi-modal distribution for employees that turnovered
- Employees who had less hours of work **(~150hours or less)** left the company more
- Employees who had too many hours of work **(~250 or more)** left the company
- Employees who left generally were **underworked** or **overworked**.

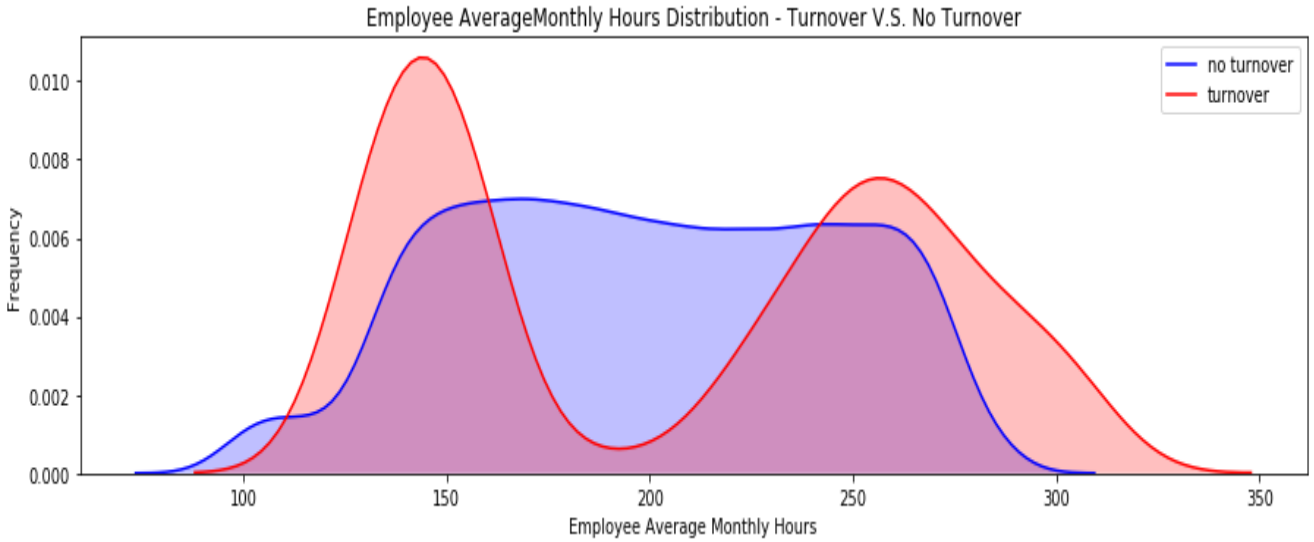
In [22]:

#KDEPlot: Kernel Density Estimate Plot

```
fig = plt.figure(figsize=(15,4))

ax=sns.kdeplot(df.loc[(df['turnover'] == 0),'averageMonthlyHours'] , color='b',shade=True, label='no turnover')

ax=sns.kdeplot(df.loc[(df['turnover'] == 1),'averageMonthlyHours'] , color='r',shade=True, label='turnover')

ax.set(xlabel='Employee Average Monthly Hours', ylabel='Frequency')

plt.title('Employee AverageMonthly Hours Distribution - Turnover V.S. No Turnover')
```

Out[22]:

Text(0.5,1,'Employee AverageMonthly Hours Distribution - Turnover V.S. No Turnover')



3i. Turnover V.S. Satisfaction

Summary:

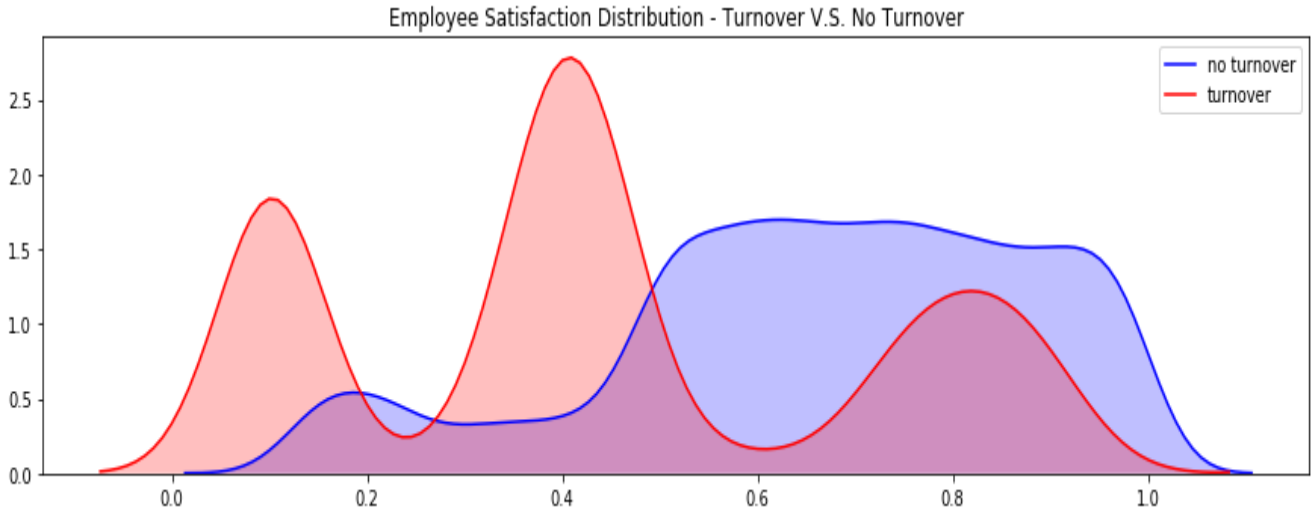
- There is a **tri-modal** distribution for employees that turnovered
- Employees who had really low satisfaction levels **(0.2 or less)** left the company more
- Employees who had low satisfaction levels **(0.3~0.5)** left the company more
- Employees who had really high satisfaction levels **(0.7 or more)** left the company more

In [23]:

```
#KDEPlot: Kernel Density Estimate Plot

fig = plt.figure(figsize=(15,4))

ax=sns.kdeplot(df.loc[(df['turnover'] == 0),'satisfaction'] , color='b',shade=True, label='no turnover')

ax=sns.kdeplot(df.loc[(df['turnover'] == 1),'satisfaction'] , color='r',shade=True, label='turnover')

plt.title('Employee Satisfaction Distribution - Turnover V.S. No Turnover')
```

Out[23]:

Text(0.5,1,'Employee Satisfaction Distribution - Turnover V.S. No Turnover')



3j. ProjectCount VS AverageMonthlyHours

Summary:

- As project count increased, so did average monthly hours
- Something weird about the boxplot graph is the difference in averageMonthlyHours between people who had a turnver and did not.
- Looks like employees who **did not** have a turnover had **consistent** averageMonthlyHours, despite the increase in projects
- In contrast, employees who **did** have a turnover had an increase in averageMonthlyHours with the increase in projects

Stop and Think:

- What could be the meaning for this?
- **Why is it that employees who left worked more hours than employees who didn't, even with the same project count?**

In [24]:

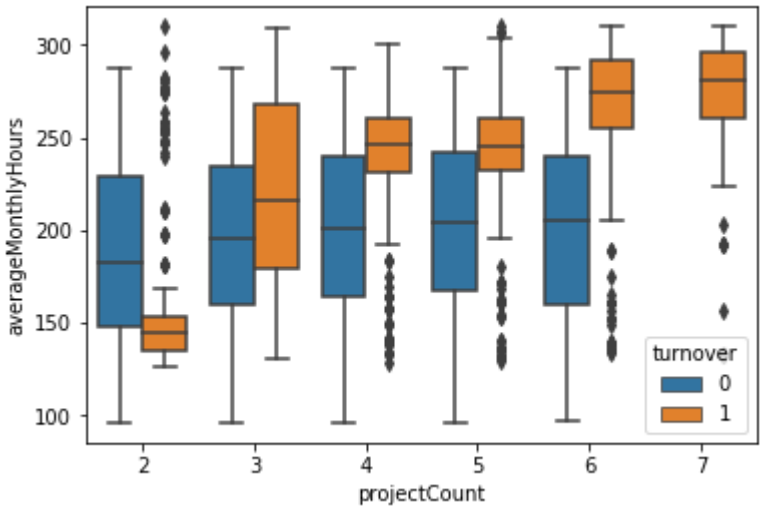
#ProjectCount VS AverageMonthlyHours [BOXPLOT]

#Looks like the average employees who stayed worked about 200hours/month. Those that had a turnover worked about 250hours/month and 150hours/month

```
import seaborn as sns

sns.boxplot(x="projectCount", y="averageMonthlyHours", hue="turnover", data=df)
```

Out[24]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f941e037cc0>



3k. ProjectCount VS Evaluation

Summary: This graph looks very similar to the graph above. What I find strange with this graph is with the turnover group. There is an increase in evaluation for employees who did more projects within the turnover group. But, again for the non-turnover group, employees here had a consistent evaluation score despite the increase in project counts.

Questions to think about:

- **Why is it that employees who left, had on average, a higher evaluation than employees who did not leave, even with an increase in project count?**

- Shouldn't employees with lower evaluations tend to leave the company more?

In [25]:

#ProjectCount VS Evaluation

#Looks like employees who did not leave the company had an average evaluation of around 70% even with different projectCounts

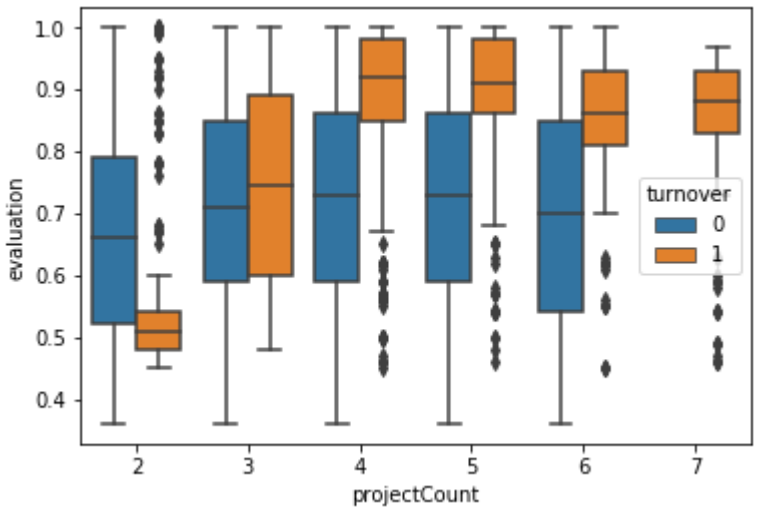
#There is a huge skew in employees who had a turnover though. It drastically changes after 3 projectCounts.

#Employees that had two projects and a horrible evaluation left. Employees with more than 3 projects and super high evaluations left

```
import seaborn as sns

sns.boxplot(x="projectCount", y="evaluation", hue="turnover", data=df)
```

Out[25]:

<matplotlib.axes._subplots.AxesSubplot at 0x7f941db4cef0>



3l. Satisfaction VS Evaluation

Summary: This is by far the most compelling graph. This is what I found:

- There are **3** distinct clusters for employees who left the company

Cluster 1 (Hard-working and Sad Employee): Satisfaction was below 0.2 and evaluations were greater than 0.75. Which could be a good indication that employees who left the company were good workers but felt horrible at their job.

- **Question:** What could be the reason for feeling so horrible when you are highly evaluated? Could it be working too hard? Could this cluster mean employees who are "overworked"?

Cluster 2 (Bad and Sad Employee): Satisfaction between about 0.35~0.45 and evaluations below ~0.58. This could be seen as employees who were badly evaluated and felt bad at work.

- **Question:** Could this cluster mean employees who "under-performed"?

Cluster 3 (Hard-working and Happy Employee): Satisfaction between 0.7~1.0 and evaluations were greater than 0.8. Which could mean that employees in this cluster were "ideal". They loved their work and were evaluated highly for their performance.

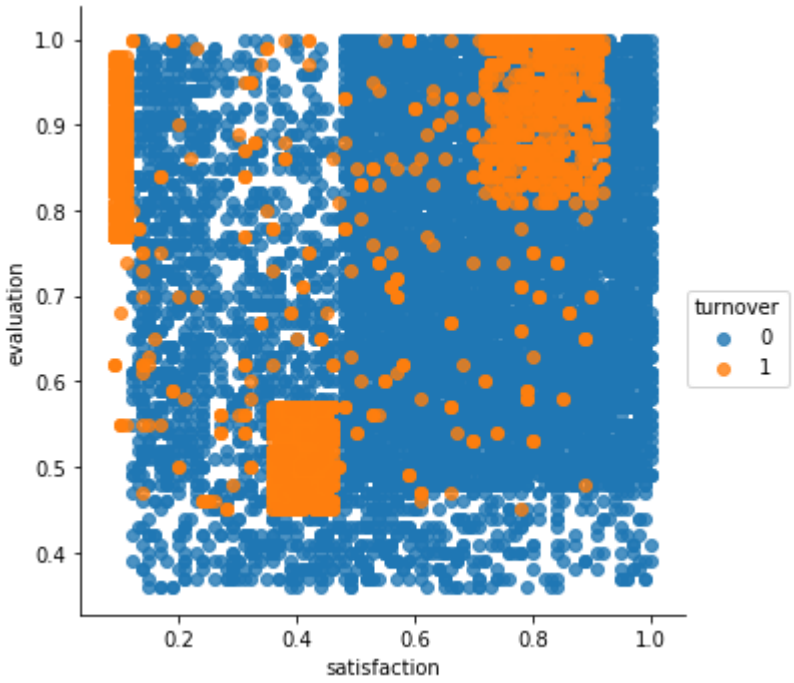
- **Question:** Could this cluser mean that employees left because they found another job opportunity?

In [26]:

```
sns.lmplot(x='satisfaction', y='evaluation', data=df,
           fit_reg=False, # No regression line
           hue='turnover')   # Color by evolution stage
```

Out[26]:

<seaborn.axisgrid.FacetGrid at 0x7f941e32f5f8>



3m. Turnover V.S. YearsAtCompany

Summary: Let's see if theres a point where employees start leaving the company. Here's what I found:

- More than half of the employees with **4 and 5** years left the company
- Employees with **5** years should **highly** be looked into

Stop and Think:

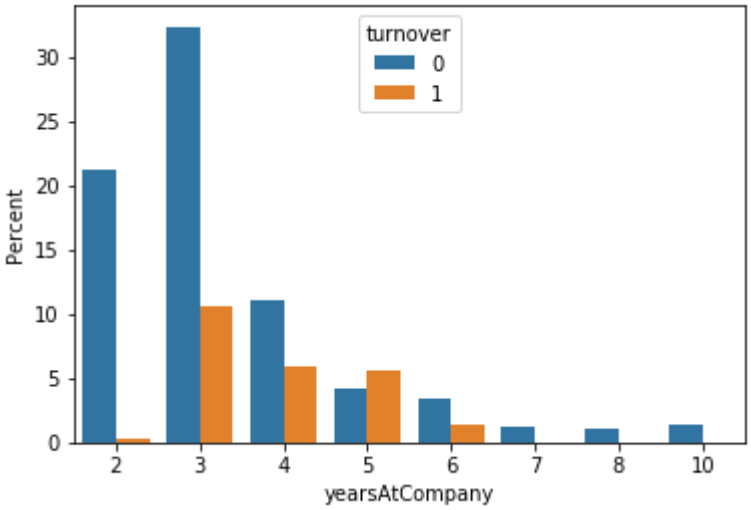
- Why are employees leaving mostly at the **3-5** year range?
- Who are these employees that left?
- Are these employees part-time or contractors?

In [27]:

```
ax = sns.barplot(x="yearsAtCompany", y="yearsAtCompany", hue="turnover", data=df, estimator=lambda x: len(x) / len(df) * 100)

ax.set(ylabel="Percent")
```

Out[27]:

[Text(0,0.5,'Percent')]



3n. K-Means Clustering of Employee Turnover

Cluster 1 (Blue): Hard-working and Sad Employees

Cluster 2 (Red): Bad and Sad Employee

Cluster 3 (Green): Hard-working and Happy Employee

Clustering PROBLEM:

- How do we know that there are "3" clusters?

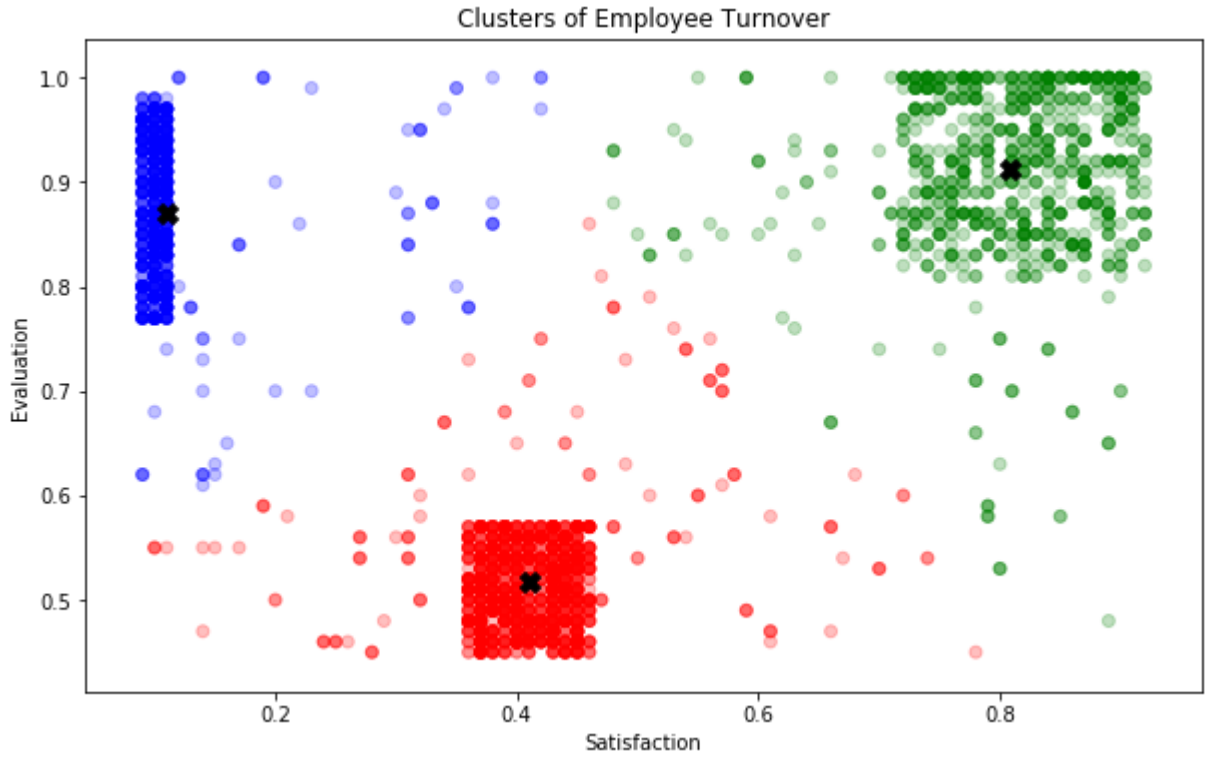
- We would need expert domain knowledge to classify the right amount of clusters

- Hidden uknown structures could be present

In [28]:

```
# Import KMeans Model

from sklearn.cluster import KMeans
```

```
# Graph and create 3 clusters of Employee Turnover

kmeans = KMeans(n_clusters=3,random_state=2)

kmeans.fit(df[df.turnover==1][["satisfaction","evaluation"]])


kmeans_colors = ['green' if c == 0 else 'blue' if c == 2 else 'red' for c in kmeans.labels_]


fig = plt.figure(figsize=(10, 6))

plt.scatter(x="satisfaction",y="evaluation", data=df[df.turnover==1],

        alpha=0.25,color = kmeans_colors)

plt.xlabel("Satisfaction")

plt.ylabel("Evaluation")

plt.scatter(x=kmeans.cluster_centers_[:,0],y=kmeans.cluster_centers_[:,1],color="black",marker="X",s=100)

plt.title("Clusters of Employee Turnover")

plt.show()
```



Feature Importance

Summary:

By using a decision tree classifier, it could rank the features used for the prediction. The top three features were employee satisfaction, yearsAtCompany, and evaluation. This is helpful in creating our model for logistic regression because it'll be more interpretable to understand what goes into our model when we utilize less features.

Top 3 Features:

1. Satisfaction
2. YearsAtCompany
3. Evaluation

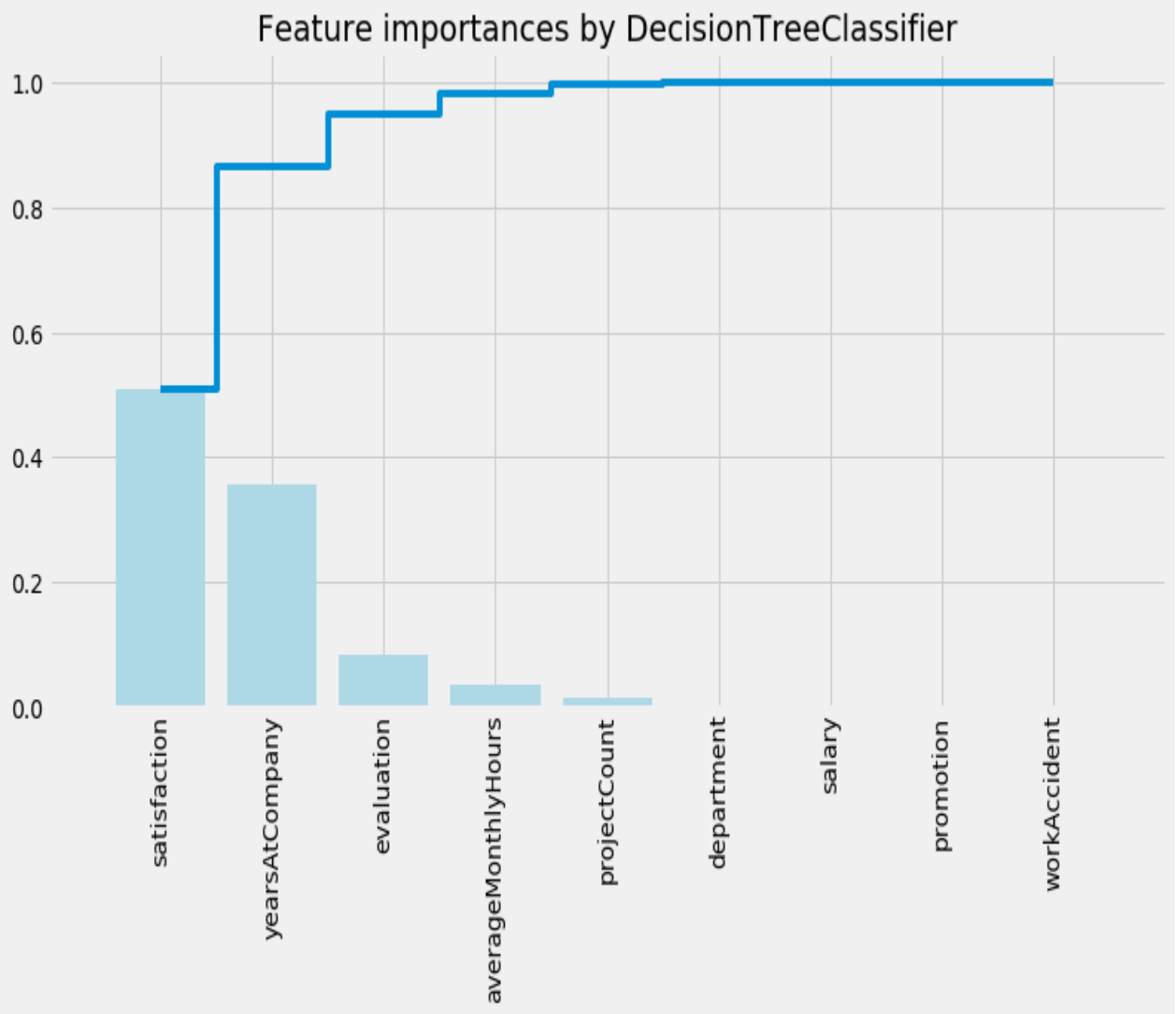
In [29]:

```
from sklearn import tree

from sklearn.tree import DecisionTreeClassifier
```

```
from sklearn.model_selection import train_test_split

plt.style.use('fivethirtyeight')

plt.rcParams['figure.figsize'] = (12,6)


# Renaming certain columns for better readability

df = df.rename(columns={'satisfaction_level': 'satisfaction',

                        'last_evaluation': 'evaluation',

                        'number_project': 'projectCount',

                        'average_montly_hours': 'averageMonthlyHours',

                        'time_spend_company': 'yearsAtCompany',

                        'Work_accident': 'workAccident',

                        'promotion_last_5years': 'promotion',

                        'sales' : 'department',

                        'left' : 'turnover'

                        })


# Convert these variables into categorical variables

df["department"] = df["department"].astype('category').cat.codes

df["salary"] = df["salary"].astype('category').cat.codes


# Create train and test splits

target_name = 'turnover'

X = df.drop('turnover', axis=1)
```

```
y=df[target_name]


X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.15, random_state=123, stratify=y)


dtree = tree.DecisionTreeClassifier(
    #max_depth=3,
    class_weight="balanced",
    min_weight_fraction_leaf=0.01
    )
dtree = dtree.fit(X_train,y_train)


## plot the importances ##
importances = dtree.feature_importances_
feat_names = df.drop(['turnover'],axis=1).columns



indices = np.argsort(importances)[::-1]
plt.figure(figsize=(12,6))
plt.title("Feature importances by DecisionTreeClassifier")
plt.bar(range(len(indices)), importances[indices], color='lightblue',  align="center")
plt.step(range(len(indices)), np.cumsum(importances[indices]), where='mid', label='Cumulative')
plt.xticks(range(len(indices)), feat_names[indices], rotation='vertical',fontsize=14)
plt.xlim([-1, len(indices)])
```

plt.show()



4a. Modeling the Data: Logistic Regression Analysis

NOTE: This will be an in-depth analysis of using logistic regression as a classifier. I do go over other types of models in the other section below this. **This is more of a use-case example of what can be done and explained to management in a company.**

Logistic Regression commonly deals with the issue of how likely an observation is to belong to each group. This model is commonly used to predict the likelihood of an event occurring. In contrast to linear

regression, the output of logistic regression is transformed with a logit function. This makes the output either 0 or 1. This is a useful model to take advantage of for this problem because we are interested in predicting whether an employee will leave (0) or stay (1).

Another reason for why logistic regression is the preferred model of choice is because of its interpretability. Logistic regression predicts the outcome of the response variable (turnover) through a set of other explanatory variables, also called predictors. In context of this domain, the value of our response variable is categorized into two forms: 0 (zero) or 1 (one). The value of 0 (zero) represents the probability of an employee not leaving the company and the value of 1 (one) represents the probability of an employee leaving the company.

Logistic Regression models the probability of 'success' as:

The equation above shows the relationship between, the dependent variable (success), denoted as (θ) and independent variables or predictor of event, denoted as xi. Where α is the constant of the equation and, β is the coefficient of the predictor variables

In [30]:

```
# Import the neccessary modules for data manipulation and visual representation
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as matplot
import seaborn as sns
%matplotlib inline
#Read the analytics csv file and store our dataset into a dataframe called "df"
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score,
confusion_matrix, precision_recall_curve

from sklearn.preprocessing import RobustScaler

df = pd.DataFrame.from_csv('../input/HR_comma_sep.csv', index_col=None)


# Renaming certain columns for better readability

df = df.rename(columns={'satisfaction_level': 'satisfaction',

                        'last_evaluation': 'evaluation',

                        'number_project': 'projectCount',

                        'average_montly_hours': 'averageMonthlyHours',

                        'time_spend_company': 'yearsAtCompany',

                        'Work_accident': 'workAccident',

                        'promotion_last_5years': 'promotion',

                        'sales' : 'department',

                        'left' : 'turnover'

                        })


# Convert these variables into categorical variables

df["department"] = df["department"].astype('category').cat.codes

df["salary"] = df["salary"].astype('category').cat.codes



# Move the reponse variable "turnover" to the front of the table

front = df['turnover']

df.drop(labels=['turnover'], axis=1,inplace = True)
```

```
df.insert(0, 'turnover', front)


# Create an intercept term for the logistic regression equation

df['int'] = 1

indep_var = ['satisfaction', 'evaluation', 'yearsAtCompany', 'int', 'turnover']

df = df[indep_var]


# Create train and test splits

target_name = 'turnover'

X = df.drop('turnover', axis=1)


y=df[target_name]

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.15, random_state=123, stratify=y)


X_train.head()

#
```

Out[30]:

	satisfaction	evaluation	yearsAtCompany	int
9003	0.59	1.00	3	1

5697	0.81	0.98	2	1
10691	1.00	0.93	2	1
1884	0.87	0.91	5	1
13572	0.87	0.48	3	1

Using Logistic Regression Coefficients

With the elimination of the other variables, I'll be using the three most important features to create our model: Satisfaction, Evaluation, and YearsAtCompany.

Following overall equation was developed:

Employee Turnover Score = Satisfaction***(-3.769022)*** + *Evaluation*(**0.207596**) + YearsAtCompany*(**0.170145**) + **0.181896**

In [31]:

```
import statsmodels.api as sm

iv = ['satisfaction','evaluation','yearsAtCompany', 'int']

logReg = sm.Logit(y_train, X_train[iv])

answer = logReg.fit()


answer.summary

answer.params
```

/opt/conda/lib/python3.6/site-packages/statsmodels/compat/pandas.py:56: FutureWarning: The pandas.core.datetools module is deprecated and will be removed in a future version. Please use the pandas.tseries module instead.

```
from pandas.core import datetools
```

Optimization terminated successfully.

Current function value: 0.467233

Iterations 6

Out[31]:

satisfaction -3.769022

evaluation 0.207596

yearsAtCompany 0.170145

int 0.181896

dtype: float64

Explanation of Coefficients

Employee Turnover Score = Satisfaction***(-3.769022)*** + *Evaluation*(**0.207596**) + YearsAtCompany*(**0.170145**) + **0.181896**

The values above are the coefficient assigned to each independent variable. The **constant** 0.181896 represents the effect of all uncontrollable variables.

In [32]:

 # Create function to compute coefficients

```
coef = answer.params

def y (coef, Satisfaction, Evaluation, YearsAtCompany) :
    return coef[3] + coef[0]*Satisfaction + coef[1]*Evaluation + coef[2]*YearsAtCompany


import numpy as np


# An Employee with 0.7 Satisfaction and 0.8 Evaluation and worked 3 years has a 14% chance of turnover

y1 = y(coef, 0.7, 0.8, 3)

p = np.exp(y1) / (1+np.exp(y1))

p
```

Out[32]:

0.14431462559738251

Intepretation of Score

If you were to use these employee values into the equation:

- **Satisfaction**: 0.7
- **Evaluation**: 0.8
- **YearsAtCompany**: 3

You would get:

Employee Turnover Score = (**0.7**)*(-3.769022)* + (***0.8***)(0.207596) + (**3**)*(0.170145)* + *0.181896 = 0.14431 = **14%***

Result: This employee would have a **14%** chance of leaving the company. This information can then be used to form our retention plan.

Retention Plan Using Logistic Regression

Reference: http://rupeshkhare.com/wp-content/uploads/2013/12/Employee-Attrition-Risk-Assessment-using-Logistic-Regression-Analysis.pdf

With the logistic regression model, we can now use our scores and evaluate the employees through different scoring metrics. Each zone is explain here:

1. **Safe Zone (Green)** – Employees within this zone are considered safe.
2. **Low Risk Zone (Yellow)** – Employees within this zone are too be taken into consideration of potential turnover. This is more of a long-term track.
3. **Medium Risk Zone (Orange)** – Employees within this zone are at risk of turnover. Action should be taken and monitored accordingly.
4. **High Risk Zone (Red)** – Employees within this zone are considered to have the highest chance of turnover. Action should be taken immediately.

So with our example above, the employee with a **14%** turnover score will be in the **safe zone**.

4b. Using Other Models

NOTE: I'll be using four other models in this section to measure the accuracy of the different models

The best model performance out of the four (Decision Tree Model, AdaBoost Model, Logistic Regression Model, Random Forest Model) was **Random Forest**!

Remember: Machines can predict the future, as long as the future doesn't look too different from the past.

Note: Base Rate

- A **Base Rate Model** is a simple model or heuristic used as reference point for comparing how well a model is performing. A baseline helps model developers quantify the minimal, expected performance on a particular problem. In this dataset, the majority class that will be predicted will be **0's**, which are employees who did not leave the company.
- If you recall back to *Part 3: Exploring the Data*, 24% of the dataset contained 1's (employee who left the company) and the remaining 76% contained 0's (employee who did not leave the company). The Base Rate Model would simply predict every 0's and ignore all the 1's.
- **Example**: The base rate accuracy for this data set, when classifying everything as 0's, would be 76% because 76% of the dataset are labeled as 0's (employees not leaving the company).

Note: Evaluating the Model

Precision and Recall / Class Imbalance

This dataset is an example of a class imbalance problem because of the skewed distribution of employees who did and did not leave. More skewed the class means that accuracy breaks down.

In this case, evaluating our model's algorithm based on **accuracy** is the **wrong** thing to measure. We would have to know the different errors that we care about and correct decisions. Accuracy alone does not measure an important concept that needs to be taken into consideration in this type of evaluation: **False Positive** and **False Negative** errors.

False Positives (Type I Error): You predict that the employee will leave, but do not

False Negatives (Type II Error): You predict that the employee will not leave, but does leave

In this problem, what type of errors do we care about more? False Positives or False Negatives?

Note: Different Ways to Evaluate Classification Models

---

1. **Predictive Accuracy:** How many does it get right?
2. **Speed:** How fast does it take for the model to deploy?
3. **Scalability:** Can the model handle large datasets?

4. **Robustness:** How well does the model handle outliers/missing values?
5. **Interpretability:** Is the model easy to understand?

In [33]:

```
from sklearn.linear_model import LogisticRegression

from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, classification_report, precision_score, recall_score,
confusion_matrix, precision_recall_curve

from sklearn.preprocessing import RobustScaler
```

In [34]:

```
# Create base rate model

def base_rate_model(X) :

    y = np.zeros(X.shape[0])

    return y
```

In [35]:

```
# Create train and test splits

target_name = 'turnover'

X = df.drop('turnover', axis=1)

#robust_scaler = RobustScaler()

#X = robust_scaler.fit_transform(X)
```

```
y=df[target_name]

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.15, random_state=123, stratify=y)
```

In [36]:

```
# Check accuracy of base rate model

y_base_rate = base_rate_model(X_test)

from sklearn.metrics import accuracy_score

print ("Base rate accuracy is %2.2f" % accuracy_score(y_test, y_base_rate))
```

```
Base rate accuracy is 0.76
```

In [37]:

```
# Check accuracy of Logistic Model

from sklearn.linear_model import LogisticRegression

model = LogisticRegression(penalty='l2', C=1)


model.fit(X_train, y_train)

print ("Logistic accuracy is %2.2f" % accuracy_score(y_test, model.predict(X_test)))
```

```
Logistic accuracy is 0.77
```

In [38]:

```
# Using 10 fold Cross-Validation to train our Logistic Regression Model
```

```
from sklearn import model_selection

from sklearn.linear_model import LogisticRegression

kfold = model_selection.KFold(n_splits=10, random_state=7)

modelCV = LogisticRegression(class_weight = "balanced")

scoring = 'roc_auc'

results = model_selection.cross_val_score(modelCV, X_train, y_train, cv=kfold, scoring=scoring)

print("AUC: %.3f (%.3f)" % (results.mean(), results.std()))
```

```
AUC: 0.793 (0.014)
```

Logistic Regression V.S. Random Forest V.S. Decision Tree V.S. AdaBoost Model

---

In [39]:

linkcode

```
# Compare the Logistic Regression Model V.S. Base Rate Model V.S. Random Forest Model

from sklearn.metrics import roc_auc_score

from sklearn.metrics import classification_report

from sklearn.ensemble import RandomForestClassifier


from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

from sklearn.linear_model import LogisticRegression
```

```
from sklearn.ensemble import ExtraTreesClassifier

from sklearn.ensemble import BaggingClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.ensemble import VotingClassifier



print ("---Base Model---")

base_roc_auc = roc_auc_score(y_test, base_rate_model(X_test))

print ("Base Rate AUC = %2.2f" % base_roc_auc)

print(classification_report(y_test, base_rate_model(X_test)))


# NOTE: By adding in "class_weight = balanced", the Logistic Auc increased by about 10%! This adjusts the threshold value

logis = LogisticRegression(class_weight = "balanced")

logis.fit(X_train, y_train)

print ("\n\n ---Logistic Model---")

logit_roc_auc = roc_auc_score(y_test, logis.predict(X_test))

print ("Logistic AUC = %2.2f" % logit_roc_auc)

print(classification_report(y_test, logis.predict(X_test)))


# Decision Tree Model

dtree = tree.DecisionTreeClassifier(

    #max_depth=3,

    class_weight="balanced",
```

```
    min_weight_fraction_leaf=0.01
    )
dtree = dtree.fit(X_train,y_train)
print ("\n\n ---Decision Tree Model---")
dt_roc_auc = roc_auc_score(y_test, dtree.predict(X_test))
print ("Decision Tree AUC = %2.2f" % dt_roc_auc)
print(classification_report(y_test, dtree.predict(X_test)))


# Random Forest Model
rf = RandomForestClassifier(
    n_estimators=1000,
    max_depth=None,
    min_samples_split=10,
    class_weight="balanced"
    #min_weight_fraction_leaf=0.02
    )
rf.fit(X_train, y_train)
print ("\n\n ---Random Forest Model---")
rf_roc_auc = roc_auc_score(y_test, rf.predict(X_test))
print ("Random Forest AUC = %2.2f" % rf_roc_auc)
print(classification_report(y_test, rf.predict(X_test)))



# Ada Boost
```

```
ada = AdaBoostClassifier(n_estimators=400, learning_rate=0.1)

ada.fit(X_train,y_train)

print ("\n\n ---AdaBoost Model---")

ada_roc_auc = roc_auc_score(y_test, ada.predict(X_test))

print ("AdaBoost AUC = %2.2f" % ada_roc_auc)

print(classification_report(y_test, ada.predict(X_test)))
```

```
---Base Model---

Base Rate AUC = 0.50

             precision    recall  f1-score   support

          0       0.76      1.00      0.86      1714
          1       0.00      0.00      0.00       536

avg / total       0.58      0.76      0.66      2250
```

```
 ---Logistic Model---

Logistic AUC = 0.74

             precision    recall  f1-score   support

          0       0.90      0.76      0.82      1714
          1       0.48      0.73      0.58       536
```

avg / total 0.80 0.75 0.76 2250

---Decision Tree Model---

Decision Tree AUC = 0.94

	precision	recall	f1-score	support
0	0.97	0.96	0.97	1714
1	0.87	0.91	0.89	536
avg / total	0.95	0.95	0.95	2250

/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.

'precision', 'predicted', average, warn_for)

---Random Forest Model---

Random Forest AUC = 0.97

	precision	recall	f1-score	support
0	0.99	0.98	0.99	1714

1 0.95 0.96 0.95 536

avg / total 0.98 0.98 0.98 2250

---AdaBoost Model---

AdaBoost AUC = 0.90

precision recall f1-score support

0 0.95 0.97 0.96 1714

1 0.90 0.82 0.86 536

avg / total 0.93 0.94 0.93 2250

ROC Graph

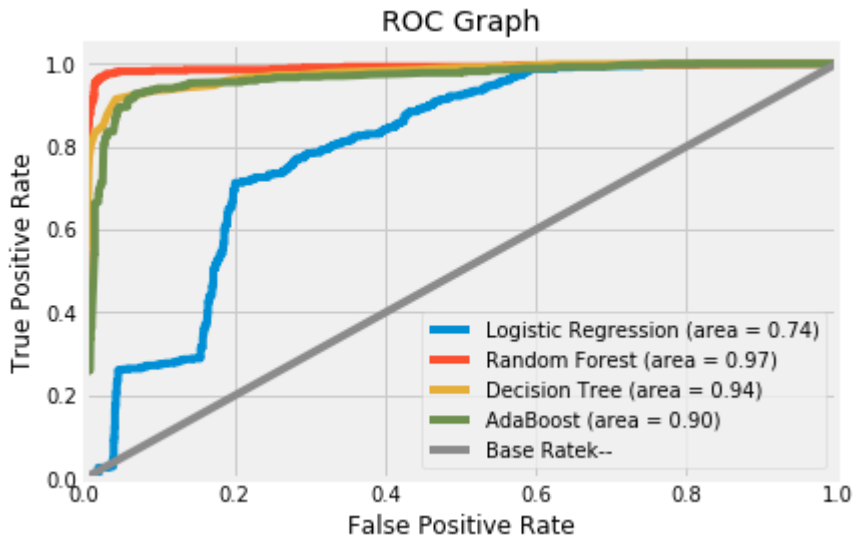
In [40]:

```
# Create ROC Graph

from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, logis.predict_proba(X_test)[:,1])

rf_fpr, rf_tpr, rf_thresholds = roc_curve(y_test, rf.predict_proba(X_test)[:,1])

dt_fpr, dt_tpr, dt_thresholds = roc_curve(y_test, dtree.predict_proba(X_test)[:,1])
```

```
ada_fpr, ada_tpr, ada_thresholds = roc_curve(y_test, ada.predict_proba(X_test)[:,1])


plt.figure()


# Plot Logistic Regression ROC

plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)


# Plot Random Forest ROC

plt.plot(rf_fpr, rf_tpr, label='Random Forest (area = %0.2f)' % rf_roc_auc)


# Plot Decision Tree ROC

plt.plot(dt_fpr, dt_tpr, label='Decision Tree (area = %0.2f)' % dt_roc_auc)


# Plot AdaBoost ROC

plt.plot(ada_fpr, ada_tpr, label='AdaBoost (area = %0.2f)' % ada_roc_auc)


# Plot Base Rate ROC

plt.plot([0,1], [0,1],label='Base Rate' 'k--')


plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Graph')
```

plt.legend(loc="lower right")

plt.show()



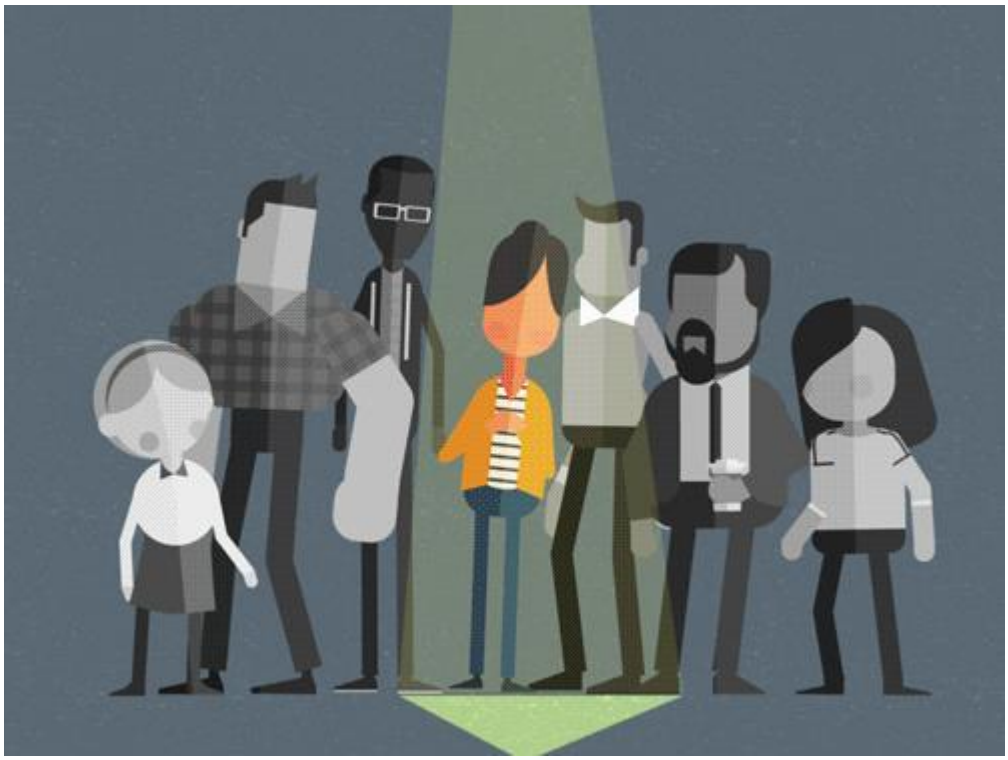
5. Interpreting the Data

Summary: With all of this information, this is what Bob should know about his company and why his employees probably left:

1. Employees generally left when they are **underworked** (less than 150hr/month or 6hr/day)
2. Employees generally left when they are **overworked** (more than 250hr/month or 10hr/day)
3. Employees with either **really high or low evaluations** should be taken into consideration for high turnover rate
4. Employees with **low to medium salaries** are the bulk of employee turnover
5. Employees that had **2,6, or 7 project count** was at risk of leaving the company
6. Employee **satisfaction** is the highest indicator for employee turnover.
7. Employee that had **4 and 5 yearsAtCompany** should be taken into consideration for high turnover rate
8. Employee **satisfaction**, **yearsAtCompany**, and **evaluation** were the three biggest factors in determining turnover.

"You don't build a business. You build people, and people build the business." - Zig Ziglar

---



Potential Solution

Binary Classification: Turnover V.S. Non Turnover

Instance Scoring: Likelihood of employee responding to an offer/incentive to save them from leaving.

Need for Application: Save employees from leaving

In our employee retention problem, rather than simply predicting whether an employee will leave the company within a certain time frame, we would much rather have an estimate of the probability that he/she will leave the company. We would rank employees by their probability of leaving, then allocate a limited incentive budget to the highest probability instances.

Consider employee turnover domain where an employee is given treatment by Human Resources because they think the employee will leave the company within a month, but the employee actually does not. This

is a false positive. This mistake could be expensive, inconvenient, and time consuming for both the Human Resources and employee, but is a good investment for relational growth.

Compare this with the opposite error, where Human Resources does not give treatment/incentives to the employees and they do leave. This is a false negative. This type of error is more detrimental because the company lost an employee, which could lead to great setbacks and more money to rehire. Depending on these errors, different costs are weighed based on the type of employee being treated. For example, if it's a high-salary employee then would we need a costlier form of treatment? What if it's a low-salary employee? The cost for each error is different and should be weighed accordingly.

Solution 1:

- We can rank employees by their probability of leaving, then allocate a limited incentive budget to the highest probability instances.
- OR, we can allocate our incentive budget to the instances with the highest expected loss, for which we'll need the probability of turnover.

Solution 2: Develop learning programs for managers. Then use analytics to gauge their performance and measure progress. Some advice:

- Be a good coach
- Empower the team and do not micromanage
- Express interest for team member success
- Have clear vision / strategy for team
- Help team with career development

Google Docs Report

---

https://docs.google.com/document/d/1E1oBewdQuX0f_LW26vKV_jcyUCNZqlivICss-ORZFtw/edit?usp=sharing

What Now?

This problem is about people decision. When modeling the data, we should not be using this predictive metric as a solution decider. But, we can use this to arm people with much better relevant information for better decision making.

We would have to conduct more experiments or collect more data about the employees in order to come up with a more accurate finding. I would recommend to gather more variables from the database that could have more impact on determining employee turnover and satisfaction such as their distance from home, gender, age, and etc.

Reverse Engineer the Problem

After trying to understand what caused employees to leave in the first place, we can form another problem to solve by asking ourselves

1. **"What features caused employees stay?**
2. **"What features contributed to employee retention?** ** There are endless problems to solve!

What would you do?

Reddit Commentor (DSPublic): I worked in HR for a couple of years and here's a few questions I have: People that have HIGH salary and not been promoted, did they leave? If so, could it be a signal that we're not developing people or providing enough opportunities?

How would you define a 'high performer' without using their last evaluation rating? Evaluations tend to be inconsistently applied across departments and highly dependent on your relationship with the person doing that evaluation. Can we do an Evaluation Vs. Departments (see if there are actual differences)? Once defined, did these high performers leave? If so, why? Are we not providing opportunities or recognizing these high performers? Is it a lack of salary?

To add some additional context, 24% turnover rate is high in general but do we know what industry this is from? If the industry norm is 50%, this company is doing great! I see you've done Turnover by dept which is great. If only we have more info and classify these turnovers.

We have voluntary and involuntary turnovers as well. Also, who are these employees - is it part timers, contract workers that turn over? We don't worry about those, they're suppose to go. I'd like to see Turnover vs. Years of service. In real life, we found a cluster / turning point where people 'turn sour' after about 5 years at the company. Can we see satisfaction vs. years at company?

Recommended Websites:

Statiscal Concepts: https://www.youtube.com/user/BCFoltz/playlists

Common Machine Learning Algorithms: https://www.linkedin.com/pulse/machine-learning-whats-inside-box-randy-lao/

Basics of Machine Learning: https://www.linkedin.com/pulse/machine-learning-fresh-bloods-randy-lao/

Data Science Pipeline (OSEMN): https://www.linkedin.com/pulse/life-data-science-osemn-randy-lao/

Edits:

To Do's:

1. Define "high performers". It's ambiguous and is normally determined by relationships. Could be inconsistent. To verify, do a **Evaluation V.S. Department**.
2. Create Expected Value Model. Cost and Benefits. Understand the cost of targeting and cost of employee leaving. Known as Cost Matrix.
3. Create a tableu dashboard for relevant/important information and highlight

Edit 1: Added Hypothesis testing for employee turnover satisfaction and entire employee satisfaction 8/29/2017

Edit 2: Added Turnover VS Satisfaction graph 9/14/2017

Edit 3: Added pd.get_dummies for 'salary' and 'department' features. This increased the AUC score by 2% (76%-78%) 9/23/2017

Edit 4: Added Random Forest Model and updated the ROC Curve. Added Base Rate Model explanation. Added AdaBoost Model. Added Decision Tree Model 9/27/2017

Edit 5: Added decision tree classifier feature importance. Added visualization for decision tree. 9/30/2017

Edit 6: Added more information about precision/recall and class imbalance solutions. Updated potential solution section and included a new section: evaluating model. 10/1/2017

Edit 7: Added an in-depth interpretation of logistic regression model. Using this for a more interpretable classifier. Showing how the coefficents are computed and how each variable is presented in the algoirthm. Added a retention plan as a metric to evaluate our model. 10/11/2017