TECHNISCHE UNIVERSITÄT DRESDEN

BUSINESS INTELLIGENCE RESEARCH TUD

DRESDEN concept

Data Science: Predictive Analytics

Introduction to Python for Data Science

Prof. Dr. Andreas Hilbert
andreas.hilbert@tu-dresden.de

Carsten Keller
carsten.keller@tu-dresden.de

http://bi.wiwi.tu-dresden.de
01062 Dresden

Telefon +49 351 463-32268
Telefax +49 351 463-32736

Data Science: Predictive Analytics

Python Example - Data Exploration

Prof. Dr. Andreas Hilbert
andreas.hilbert@tu-dresden.de

Carsten Keller
carsten.keller@tu-dresden.de

http://bi.wiwi.tu-dresden.de
01062 Dresden

Telefon +49 351 463-32268
Telefax +49 351 463-32736

Python Example

Introduction

- The case study Human Resource Analytics introduce to data manipulation and cleaning methods using the popular Python **Pandas** Data Science library and present the abstraction of the **DataFrame** as a central structure for data analysis.

- The data set was released from Kaggle Competitions, where programmers from all over the world could attend in Data Science Competitions and win attractive prices.

- By the end of the walkthrough, students will be able to take tabular data, clean and manipulate it.

- The example case also uses some other Applied Data Science methods for Data Exploration, Predictive Analytics and Machine Learning in Python as well as certain measurement and visualization methods.

- As process model we apply the **Cross Industry Standard Process for Data Mining** (CRISP-DM) that describes commonly steps experts use to analyze data.

Python Example

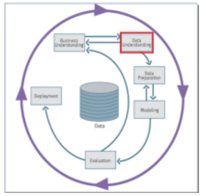
Business Understanding: HR Analytics

- Employee attrition is one of the **biggest challenges** that companies face.

- There are several factors that lead to attrition. While it may not be easy to control all the factors, it may not be worthwhile to look into those factors that seem controllable. Factors such as e.g. average number of hours spend per month by the employees, salary, promotions, Work_accident, number of projects are a few which are easier to manage.

- If we are able to extract cut-off levels for some of the above mentioned factors through our analysis, then we should be able to have a better understanding about the factors that are responsible for employees leaving the company prematurely.

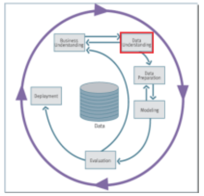
- The analysis seeks answers to the following two questions:
  - Why are our best and most experienced employees leaving prematurely?
  - Which employee will leave next?



- The analysis done in this case based on **Human Resources Analytics** data set obtained from Kaggle, where it was released under CC BY-SA 4.0 License. It serves for exploration of HR analytics data and try to discern which factors matter the most in determining why the personnel leave.
  - **File name**: HR_comma_sep.csv
  - **Number of data sets**: 14.999
  - **Number of variables**: 10 (9 Input-variables, 1 target variable)
  - **Target variable:** satisfaction level → continuoues variable
- Each row corresponds to an employee and the observation itself covers at least 10 years (max. time_spend_company).
- The case starts by exploring the data (univariate and multivariate analysis) then it goes on to explore the questions of interest

Data Understanding

The Data – Attribute Information



	Column description	scales of measure	model role
satisfaction_level	Level of satisfaction (0-1)	interval	output
last_evaluation	The last performance-evaluation of the employee (0-1)	interval	input
number_project	Number of projects completed while at work	ratio scale	input
average_montly_hours	Average monthly hours at workplace	ratio scale	input
time_spend_company	Number of years spent in the company	ratio scale	input
Work_accident	Whether the employee had a workplace accident (1 or 0)	nominal (binary)	input
Left	Whether the employee left the workplace or not (1 or 0)	nominal (binary)	rejected
promotion_last_5years	Whether the employee was promoted in the last five years (1 or 0)	nominal (binary)	Input
sales	Department in which they work	nominal	input
salary	Relative level of salary (low-medium-high)	ordinal	input

- In Python we need to load the basic modules for working with data.

```
#The HR Analytics dateset from kaggle competitions
#Import of  necessary libraries, Modules and classifiers
import numpy as np  #fundamental package for scientific computing with Python
import pandas as pd #package providing fast, flexible, and expressive data structures
import matplotlib.pyplot as plt #for plotting different kinds of diagrams
#commands in cells below the cell that outputs a plot will not affect the plot inline command
#(commentation on the same line causes an error):
%matplotlib inline
import seaborn as sns #visualization library based on matplotlib, for statistical data visualization
```

- We use Pandas‘ read_csv()-method to load the data in a DataFrame-structure.

```
hr_data=pd.read_csv('.\HR_comma_sep.csv',header=0) #read the data from a csv-file; ensure that the  values
#are separated by commas otherwise you need to specify the delimiter explicitly within the load-statement

hr_data_copy=hr_data.copy()  #create a deep copy of the data set for demonstrating how to handle missing values (mv)

hr_data.head() #show the first five entries; attribute in brackets will give the # of printed lines
```

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

- We define functions that we will need during our exploration task

```
###We define small helper functions (Code from E-M-A-D; https://www.kaggle.com/etakla)
#A function for annotating the bars with its total and relative number.
def annotate_bars(bar_plt, bar_plt_var, by=None, x_offset=0, y_offset=0, txt_color="white",
                  fnt_size=12, fnt_weight='bold'):
    if by is None:
        for p in bar_plt.patches:
            bar_plt.annotate(str( int(p.get_height()) ) + "\n" + str(round(
                          (100.0* p.get_height()) /bar_plt_var.count(), 1) )
                              + "%",
                              (p.get_x() + x_offset, p.get_height()-y_offset),
                              color=txt_color, fontsize=fnt_size, fontweight=fnt_weight)
    else:
        grouped = bar_plt_var.groupby(by)
        for p in bar_plt.patches:
            #This part is tricky. The problem is that not each x-tick gets drawn in order,
            #i.e. yes/no of the first group then yes/no of the second group located on the
            #next tick, but rather all the yes on all the x-ticks get drawn first then all
            #the nos next. So we need to know we are using a patch that belongs to which
            #tick (the x-tick) ultimately refers to one of the groups. So, we get the x absolute
            #coordinate, round it to know this patch is closest to which tick (Assuming that it
            #will always belong to its closest tick), then get the group count of that tick and
            #use it as a total to compute the percentage.
            total = grouped.get_group(bar_plot.get_xticks()[int(round(p.get_x()))]).count()
            bar_plt.annotate(str( int(p.get_height()) ) + "\n" + str(round( (100.0*
                                                                  p.get_height()) /total, 1) )+ "%",
                              (p.get_x() + x_offset, p.get_height()-y_offset),
                              color=txt_color, fontsize=fnt_size, fontweight=fnt_weight)
```

Data Understanding

Helper Functions (2/3)


```python
#A function that returns the order of a group_by object according to the average of certain parameter param.
def get_ordered_group_index(df, group_by, param, ascending=False):
    return df.groupby(group_by)[param].mean().sort_values(ascending=ascending).index

#helper function that returns the order of a group_by object according to the average of certain parameter param.
def group_by_2_level_perc(df, level1, level2, level1_index_order = None, level2_index_order = None):
    #http://stackoverflow.com/questions/23377108/pandas-percentage-of-total-with-groupby
    df_by_lvl1_lvl2 = df.groupby([level1, level2]).agg({level1: 'count'})
    df_by_lvl1_lvl2_perc = df_by_lvl1_lvl2.groupby(level=0).apply(lambda x: 100 * x / float(x.sum()))
    #Reorder them in logical ascending order, but first make sure it is not an empty input
    if level1_index_order:
        df_by_lvl1_lvl2_perc = df_by_lvl1_lvl2_perc.reindex_axis(level1_index_order, axis=0, level=0)
    #If a second level order is passed, apply it, else use the default
    if level2_index_order:
        df_by_lvl1_lvl2_perc = df_by_lvl1_lvl2_perc.reindex_axis(level2_index_order, axis=0, level=1)
    return df_by_lvl1_lvl2_perc

#A function that adds some styling to the graphs, like custom ticks for axes, axes labels and a grid
def customise_2lvl_perc_area_graph(p, legend_lst, xtick_label = "", x_label="", y_label=""):
    #If custom ticks are passed, spread them on the axe and write the tick values
    if xtick_label:
        p.set_xticks(range(0,len(xtick_label)))
        p.set_xticklabels(xtick_label)
    #Create y ticks for grid. It will always be a percentage, so it is not customisable
    p.set_yticks(range(0,110,10))
    p.set_yticklabels(['{:3.0f}%'.format(x) for x in range(0,110,10)])
    p.set_yticks(range(0,110,5), minor=True)
```

Data Understanding

Helper Functions (3/3)


```
    #Draw grid and set y limit to be only 100 (By default it had an empty area at the top of the graph)
    p.xaxis.grid('on', which='major', zorder=1, color='gray', linestyle='dashed')
    p.yaxis.grid('on', which='major', zorder=1, color='gray', alpha=0.2)
    p.yaxis.grid('on', which='minor', zorder=1, color='gray', linestyle='dashed', alpha=0.2)
    p.set(ylim=(0,100))

    #Customise legend
    p.legend(labels=legend_lst, frameon=True).get_frame().set_alpha(0.2)

    #Put the axes labels
    if x_label:
        p.set_xlabel(x_label)
    if y_label:
        p.set_ylabel(y_label);
```

- The theme of this dataset is about the employees who left, that is why we plot the distributions for those who left and those who stayed

```
#Univariate Analysis
hr_by_left = hr_data.groupby('left')  #determine the groups on each value of the object's index
employees_left = hr_by_left.get_group(1) #determine and store the group that left
employees_stayed = hr_by_left.get_group(0) #determine and store the group that stayed
```

Data Understanding

Data Description Report

- But before we start we try to get an overview about the observed data set with the help of data description report

DataFrame

show information about the object

```
hr_data.info() #attribut specifications
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
satisfaction_level       14999 non-null float64
last_evaluation          14999 non-null float64
number_project           14999 non-null int64
average_montly_hours     14999 non-null int64
time_spend_company       14999 non-null int64
Work_accident            14999 non-null int64
left                     14999 non-null int64
promotion_last_5years    14999 non-null int64
sales                    14999 non-null object
salary                   14999 non-null object
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
```

double precision float: sign bit, 11 bits exponent, 52 bits mantissa

int64(-9223372036854775808 to 9223372036854775807)

data type object: encapsulation of variables and functions into a single entity (recognize the attribute names)

# of rows

Data Understanding

Feature Description

- We can print some statistics about the attributes (mean, minimal, maximal value; 25%-, 50%- & 75%-percentile) with the describe() method.

features

```
hr_data.describe() #get an overview of the data we'll be working with
```

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	prom
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	1499
mean	0.612834	0.716102	3.803054	201.050337	3.498233	0.144610	0.238083	0.021
std	0.248631	0.171169	1.232592	49.943099	1.460136	0.351719	0.425924	0.144
min	0.090000	0.360000	2.000000	96.000000	2.000000	0.000000	0.000000	0.000
25%	0.440000	0.560000	3.000000	156.000000	3.000000	0.000000	0.000000	0.000
50%	0.640000	0.720000	4.000000	200.000000	3.000000	0.000000	0.000000	0.000
75%	0.820000	0.870000	5.000000	245.000000	4.000000	0.000000	0.000000	0.000
max	1.000000	1.000000	7.000000	310.000000	10.000000	1.000000	1.000000	1.000

# of entries (should be equal for every feature – no missing values)

minimum period of observation

Data Understanding

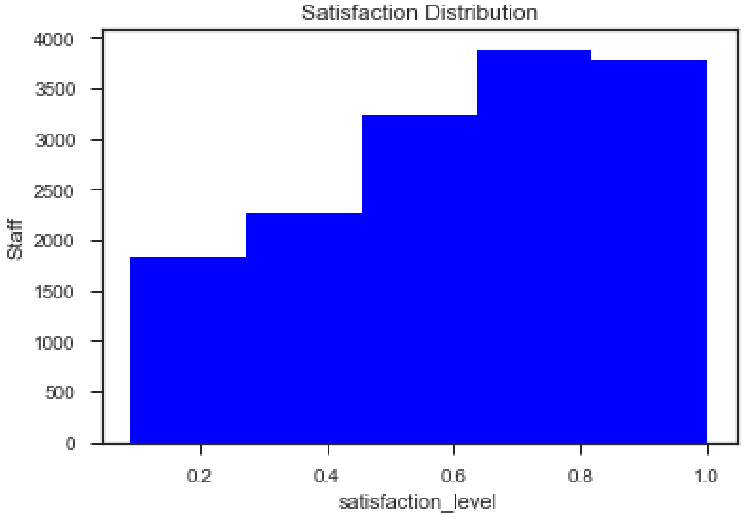
Visualizing Distributions

- To get an overview about the data we can use different types for visualization

```
#To observe the distribution of satisfaction level among the employees we generate metrics
#of skewness and plot the histogram of the satisfaction level.
print ("Skew is:", hr_data.satisfaction_level.skew()) #*.skew() shows tendency (0=no skewness, (-)=left skewed)

plt.hist(hr_data.satisfaction_level, color='blue',bins=5) #plot the histogram
#plt.hist needs argument "data" in form of a 1d numpy array
#we can adress columns as numpy arrays by just adding their name to the data frame (e.g. df.variable_name).
#we choose the parameter color to be blue --> blue histogram and bins=5 to fit the data to 5 pillars.

plt.xlabel('satisfaction_level') #naming the x-axis
plt.ylabel('Staff') #labeling the y-axis
plt.title('Satisfaction Distribution')

plt.show() #display the plot
```

```
Skew is: -0.476360341284
```

Negative value → left skewed



Histograms are used for showing the distribution of measurements

- The join method returns a string, which is the concatenation of the strings in the sequence, separated by commas (every entry appears only one time).
- Doing so we can get an overview about the unique entries in each column.

```
print('Departments: ', ', '.join(hr_no_missing_d['sales'].unique())) #show and join the unique entries in sales
#with the description "departments"
print('Salary levels: ', ', '.join(hr_no_missing_d['salary'].unique())) #show and join the unique entries in salery
#with a global descriptive level
```

```
Departments:  sales, accounting, hr, technical, support, management, IT, product_mng, marketing, RandD
Salary levels:  low, medium, high
```

- Given that the "sales' and "salary" columns are non-numeric we rename the features and dummy code the variables.
- We save them as another DataFrame in case we need to access the string values, such as for a cross-tabulation table.

```
hr_data.rename(columns={'sales':'department'}, inplace=True) #Renaming Columns, note: you do need to specify the
#existing label first followed by the new label to rename it to afterward
hr_data_new = pd.get_dummies(hr_data, ['department', 'salary'] ,drop_first = True) #Whether to get k-1 dummies out
#of k categorical levels by removing the first level. New in version 0.18.0.
hr_data_new.head()
```

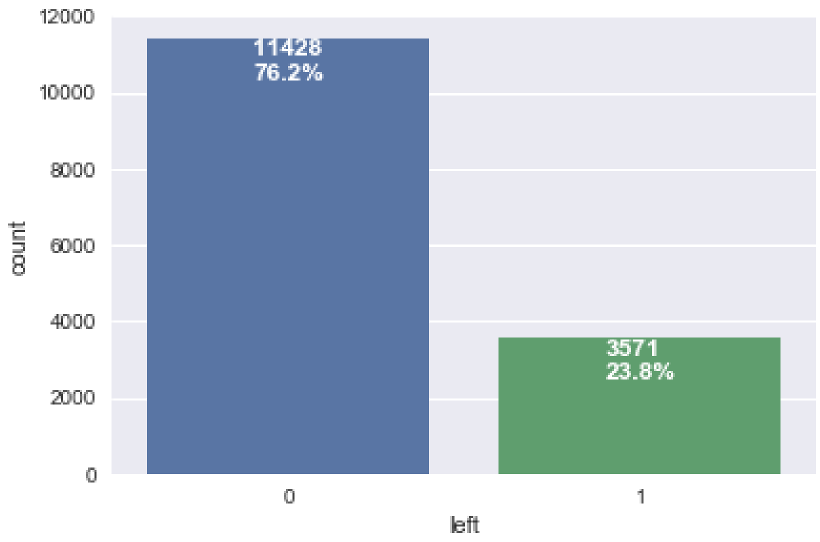
ment	department_marketing	department_product_mng	department_sales	department_support	department_technical	salary_low	salary_medium
	0	0	1	0	0	1	0
	0	0	1	0	0	0	1
	0	0	1	0	0	0	1
	0	0	1	0	0	1	0
	0	0	1	0	0	1	0

"IT" and "high" are the baseline levels for the assigned department and salary level. We create dummy variables as another data table (so called DataFrame) for a cross-tabulation table.

- Pythons matplotlib and seaborn library are bringing some nice plots for different kind of drawings to discover knowledge.
  After having prepared the data we use in the HR-case:
  - **Countplot** can be thought of as a histogram across a categorical, instead of metric, variable
  - **Pairplot** plots pairwise relationships in a data set
  - **Violin plot** plays a similar role as a box and whisker plot. It shows the distribution of quantitative data across several levels of one (or more) categorical variables such that those distributions can be compared. Unlike a **box plot**, in which all of the plot components correspond to actual data points, the violin plot features a kernel density estimation of the underlying distribution. (More information about violin plots are given here: http://seaborn.pydata.org/tutorial/categorical.html?highlight=violinplot)
- Further we use **kdeplot** and **distplot**

- We compare the groups in our company with the small helper functions.

```
#Employees who Left their Jobs in sum
employees_left_plt = sns.countplot(hr_data.left);

for p in employees_left_plt.patches:
    employees_left_plt.annotate(str( int(p.get_height()) ) + "\n" + str(round( (100.0* p.get_height())
                                                                              /hr_data.left.count(), 1) )+ "%",
                            (p.get_x() + 0.3, p.get_height()-1100),
                            color='white', fontsize=12, fontweight='bold')
```



- To see if there are particular departments that tend to have a higher proportion of people leaving, we use a **crosstab** and add a **normalize-parameter** to show the proportion of fluctuation in every department

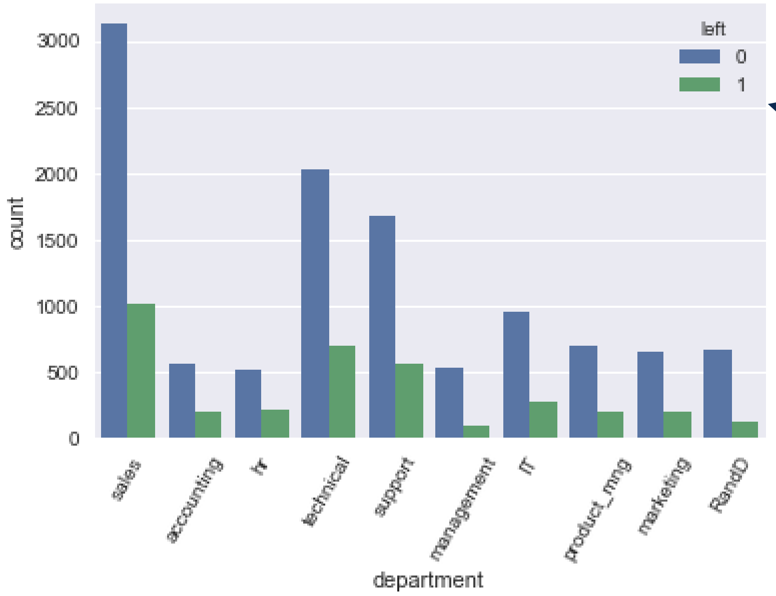
```
#proportion of leaving and staying in the different departments:
dept_table = pd.crosstab(hr_data['department'], hr_data['left'],normalize='index')
#we created a cross tabulation of columns left and department, the normalize parameter is
#dividing all values by the sum of values.
#parameter list for pandas.crosstab can be found in pandas documentation:
#https://pandas.pydata.org/pandas-docs/stable/generated/pandas.crosstab.html
dept_table.index.names = ['Department'] #naming of the index column
dept_table #printing the index column
```

left	0	1
Department		
IT	0.777506	0.222494
RandD	0.846252	0.153748
accounting	0.734029	0.265971
hr	0.709066	0.290934
management	0.855556	0.144444
marketing	0.763403	0.236597
product_mng	0.780488	0.219512
sales	0.755072	0.244928
support	0.751009	0.248991
technical	0.743750	0.256250

R&D and management tend to have lower as well as HR and accounting tend to have higher rates of leaving. The other departments are fairly similar, all between around 22 to 25 percent.

- To visualize he numbers we can apply the countplot for every department

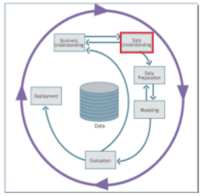
```
g=sns.countplot(x='department', hue='left', data=hr_data) #Show the number of observations
#in each categorical bin of 'department' using bars.
for item in g.get_xticklabels():  #for every entry in x rotate the x-axis for better reading
    item.set_rotation(60)         # No'in brackets = degrees in positive direction
```



Set different color (hue) for the pillar depending on the entries in column 'left'

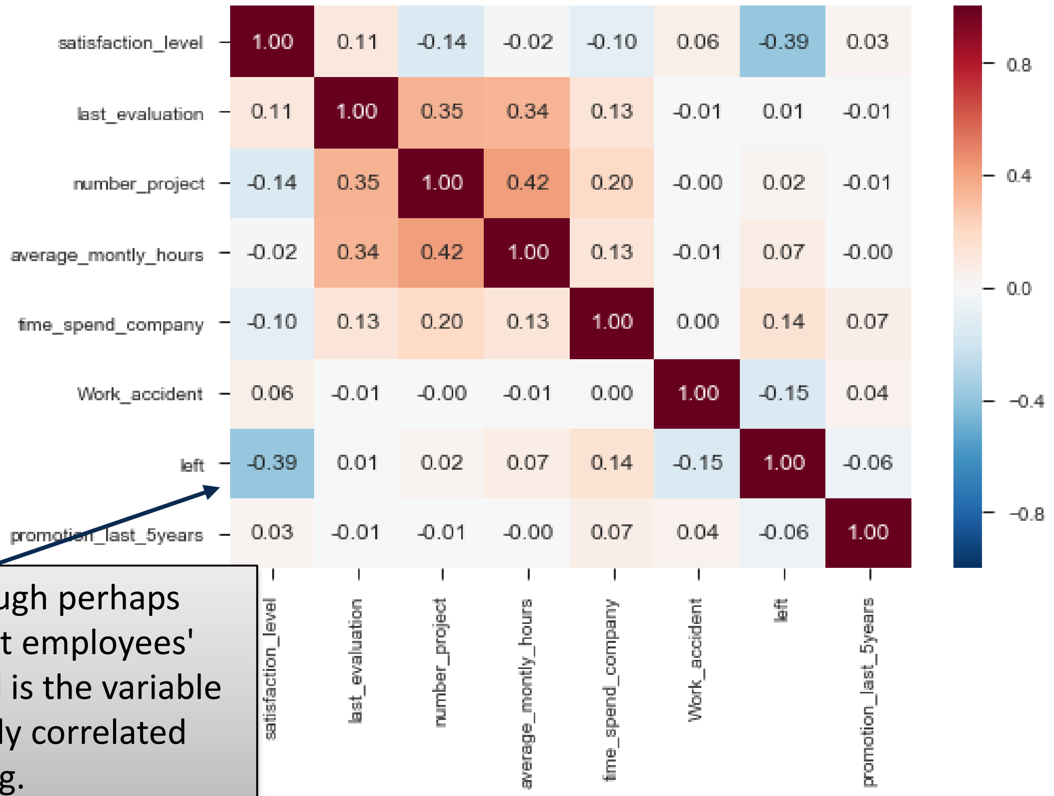
Data Understanding

Feature Correlation - Heatmap



- To recognize the dependencies/correlation between the variables we can use some attractive seaborn-visualization tools

```
# Correlation matrix is used to do some basic visualizations and show any relationships in the data
sns.heatmap(hr_data.corr(), annot=True,fmt='.2f');  #compute pairwise correlation of columns,
#excluding NA/null values, annot=True presents heatmap with values, the format-configuration
#makes it better to read (2 decimal places), <;> is hiding the processing steps
```



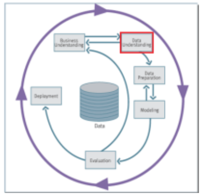
It is notable, though perhaps unsurprising, that employees' **satisfaction level** is the variable that is most highly correlated with them leaving.

Data Understanding

Meaning of the Heatmap

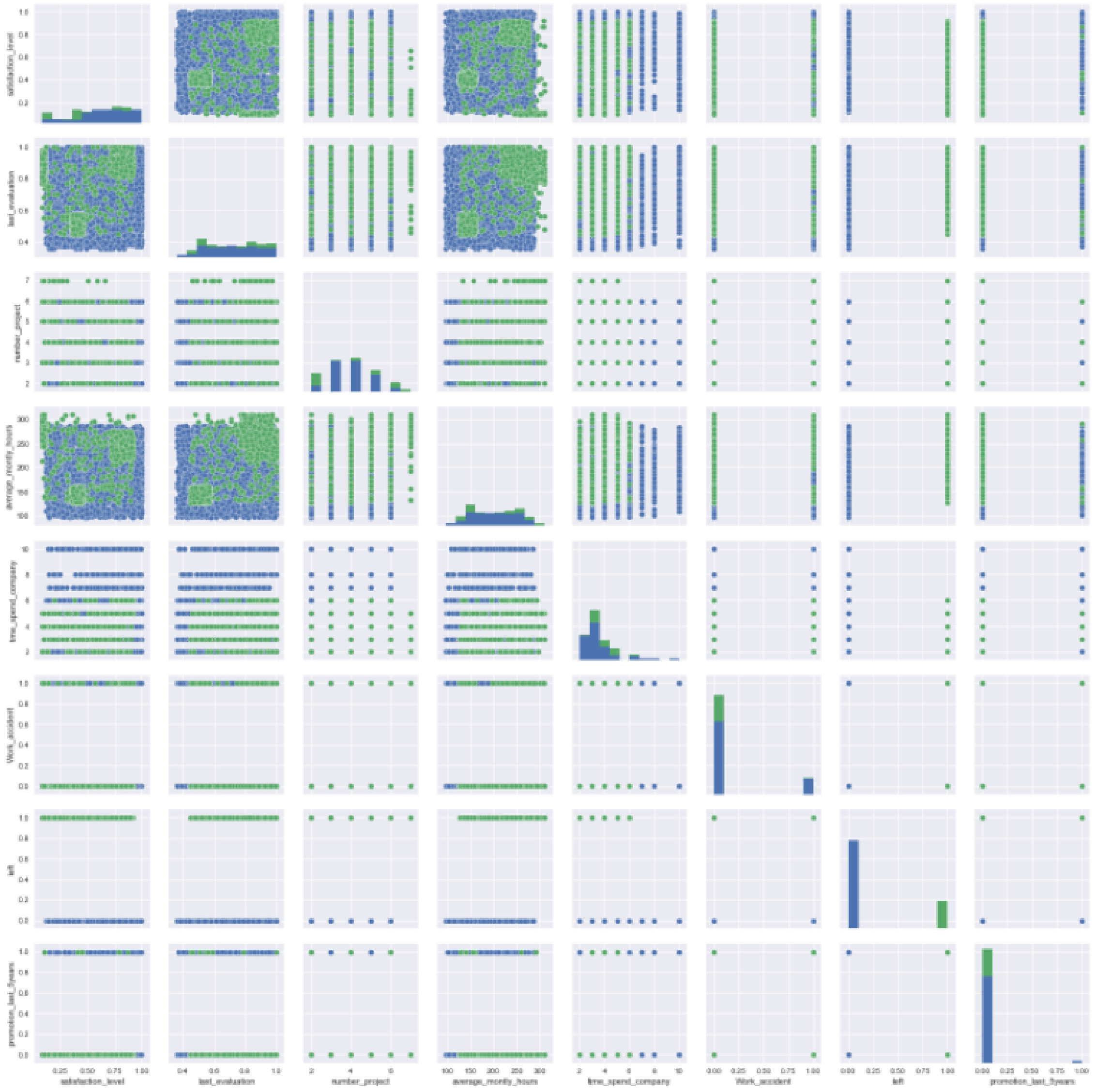
- For the "left" parameter, there is a **moderate correlation** with the satisfaction level and a **negative correlation** with the number of work accidents and the time spent at the company, but the values are too low, it may be noise.
- The other interesting correlation is the red blob around last evaluation, average mont(h)ly hours and the number of projects. They seem to be related in a moderate linear way.
- Because the theme about this dataset is who will leave next, it only makes sense to colour the **scatter plots** by the 'left' parameter:

```
plt.figure(figsize=(10, 10))

sns.pairplot(hr_data,  hue="left");
```



Data Understanding

Meaning of the Heatmap



There are some interesting green patches in the scatters involving the satisfaction level, last evaluation, average monthly hours and time spent at the company.

- It seems that the lowest evaluated employees do not leave.
- Employees who received a promotion within the past 5 years seem to be less likely to leave, their columns are much "blue-er" than those who did not.
- There is a threshold of satisfaction, after which none leaves, in the dataset.
- Employees who remain long enough in the company (over 6 years) are less likely to leave
- The average monthly hours of those who left is higher. Too much work, and maybe too little in return?

Data Understanding

Graphical Analysis

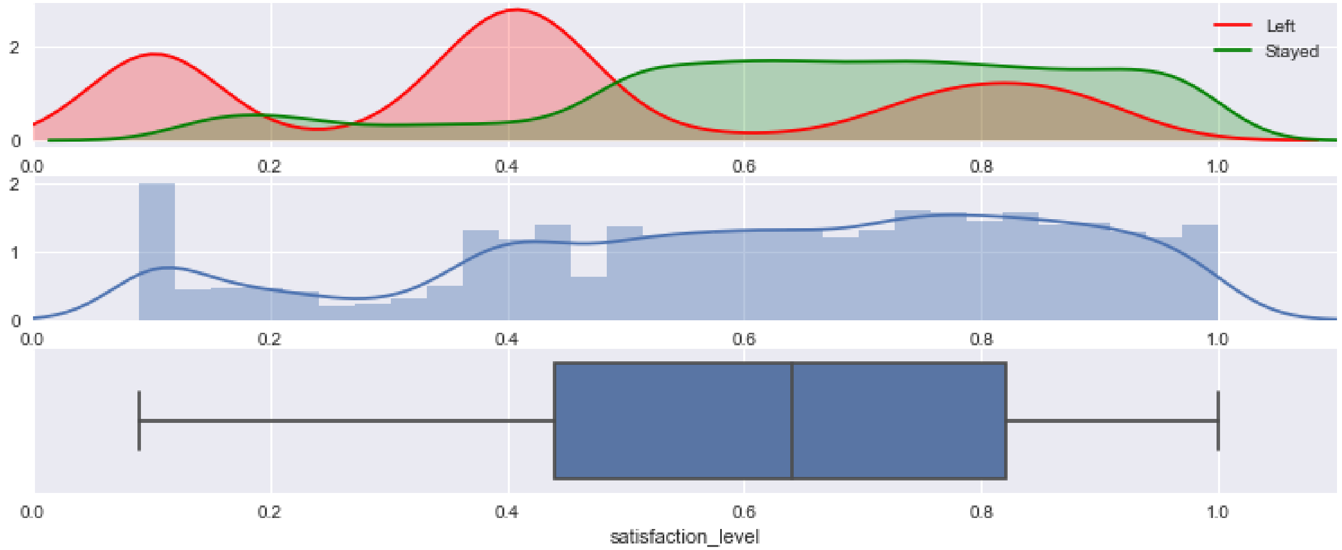
- Since the theme of this dataset is about the employees who left, we plot the distributions for those who left and those who stayed

```
#Univariate Analysis
hr_by_left = hr_data.groupby('left')  #determine the groups on each value of the object’s index
employees_left = hr_by_left.get_group(1) #determine and store the group that left
employees_stayed = hr_by_left.get_group(0) #determine and store the group that stayed
```

Data Understanding

Satisfaction among the employees


```
#distribution of Satisfaction Level
fig, axs = plt.subplots(nrows= 3, figsize=(13, 5))

sns.kdeplot(employees_left.satisfaction_level, ax=axs[0], shade=True, color="r")
kde_plot = sns.kdeplot(employees_stayed.satisfaction_level, ax=axs[0], shade=True, color="g")
kde_plot.legend(labels=['Left', 'Stayed'])

hist_plot = sns.distplot(hr_data.satisfaction_level, ax=axs[1])
box_plot = sns.boxplot(hr_data.satisfaction_level, ax=axs[2])

kde_plot.set(xlim=(0,1.1))
hist_plot.set(xlim=(0,1.1))
box_plot.set(xlim=(0,1.1));
```

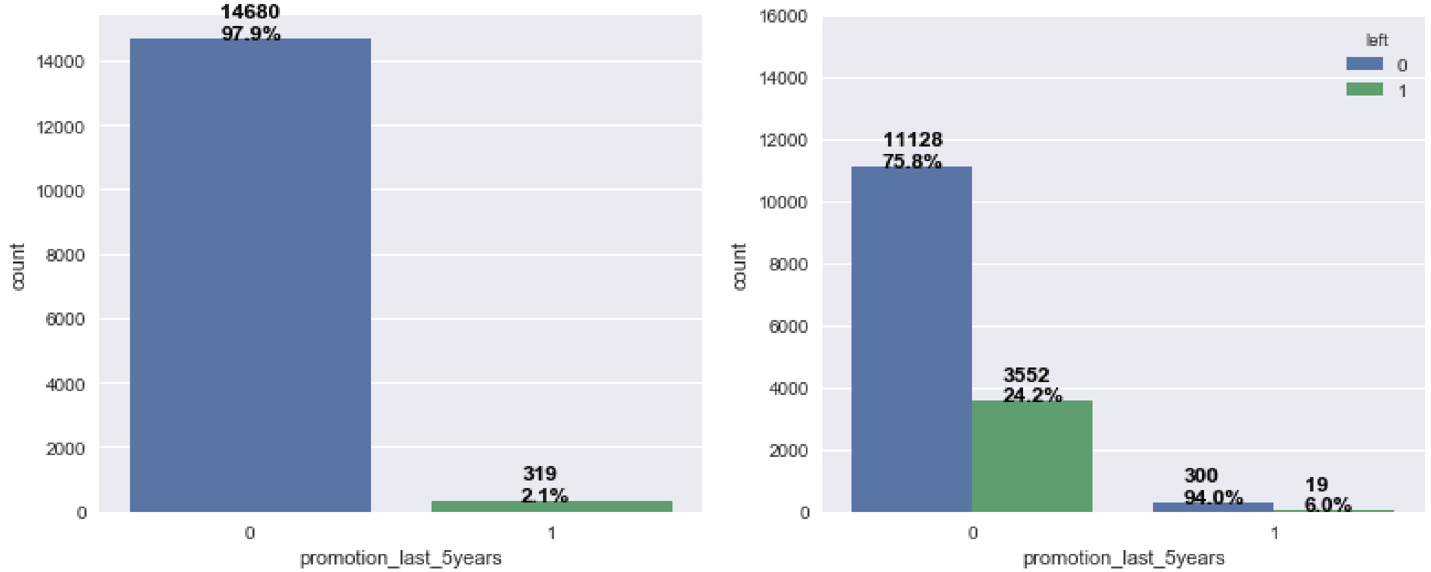


- A "bimodal shape" → one peak for the bitter employees at the lower end of satisfaction and another one well spread from medium to high satisfaction.

Data Understanding

Promotion within the last 5 years


```
#Promotion within the Past 5 Years
fig, axs = plt.subplots(ncols= 2, figsize=(13, 5))

promoted_5years_plt = sns.countplot(hr_data.promotion_last_5years, ax=axs[0]);
annotate_bars(bar_plt=promoted_5years_plt, bar_plt_var=hr_data.promotion_last_5years,
              x_offset=0.3, txt_color="black")

bar_plot = sns.countplot(x=hr_data.promotion_last_5years, hue=hr_data.left, ax=axs[1])
annotate_bars(bar_plt=bar_plot, by=hr_data.promotion_last_5years, bar_plt_var=hr_data.promotion_last_5years,
              x_offset=0.1, txt_color="black")
bar_plot.set(ylim=(0,16000));
```



- A majority did not get promoted. But if they did, they are most likely going to stay.

Data Understanding

Chi-square test for verification


```
#Create groups
employees_by_promotion = hr_data.groupby("promotion_last_5years")
employees_promoted = employees_by_promotion.get_group(1)
employees_not_promoted = employees_by_promotion.get_group(0)

#Get counts
employees_promoted_stayed = employees_promoted.groupby("left").get_group(0).left.count()
employees_promoted_left = employees_promoted.groupby("left").get_group(1).left.count()

employees_not_promoted_stayed = employees_not_promoted.groupby("left").get_group(0).left.count()
employees_not_promoted_left = employees_not_promoted.groupby("left").get_group(1).left.count()

#Create rows that makeup the contingency table
promoted_row = [employees_promoted_stayed, employees_promoted_left,
                employees_promoted_stayed + employees_promoted_left]
not_promoted_row = [employees_not_promoted_stayed, employees_not_promoted_left,
                    employees_not_promoted_stayed + employees_not_promoted_left]
total_row = [employees_promoted_stayed+employees_not_promoted_stayed,
             employees_promoted_left+employees_not_promoted_left,
             hr_data.left.count()]

#Create the contingency table
contingency_table = pd.DataFrame({'Promoted': promoted_row ,
                                  'Not Promoted': not_promoted_row ,
                                  'Total, By Left': total_row},
                                 index = ['Stayed', 'Left', 'Total, by Promotion'],
                                 columns = [ 'Promoted', 'Not Promoted', 'Total, By Left'])

display_html(contingency_table)
```

	Promoted	Not Promoted	Total, By Left
Stayed	300	11128	11428
Left	19	3552	3571
Total, by Promotion	319	14680	14999

Data Understanding

Chi-square test for verification

```
chi_squared, p, degrees_of_freedom, expected_frequency = sp.chi2_contingency( contingency_table )

print("Chi Squared: ", chi_squared)
print("p value: ", p)
print("Degrees of Freedom", degrees_of_freedom)
print("Expected Frequency for The Not Promoted Employees:", expected_frequency[0])
print("Expected Frequency for The Promoted Employees:", expected_frequency[1])
```

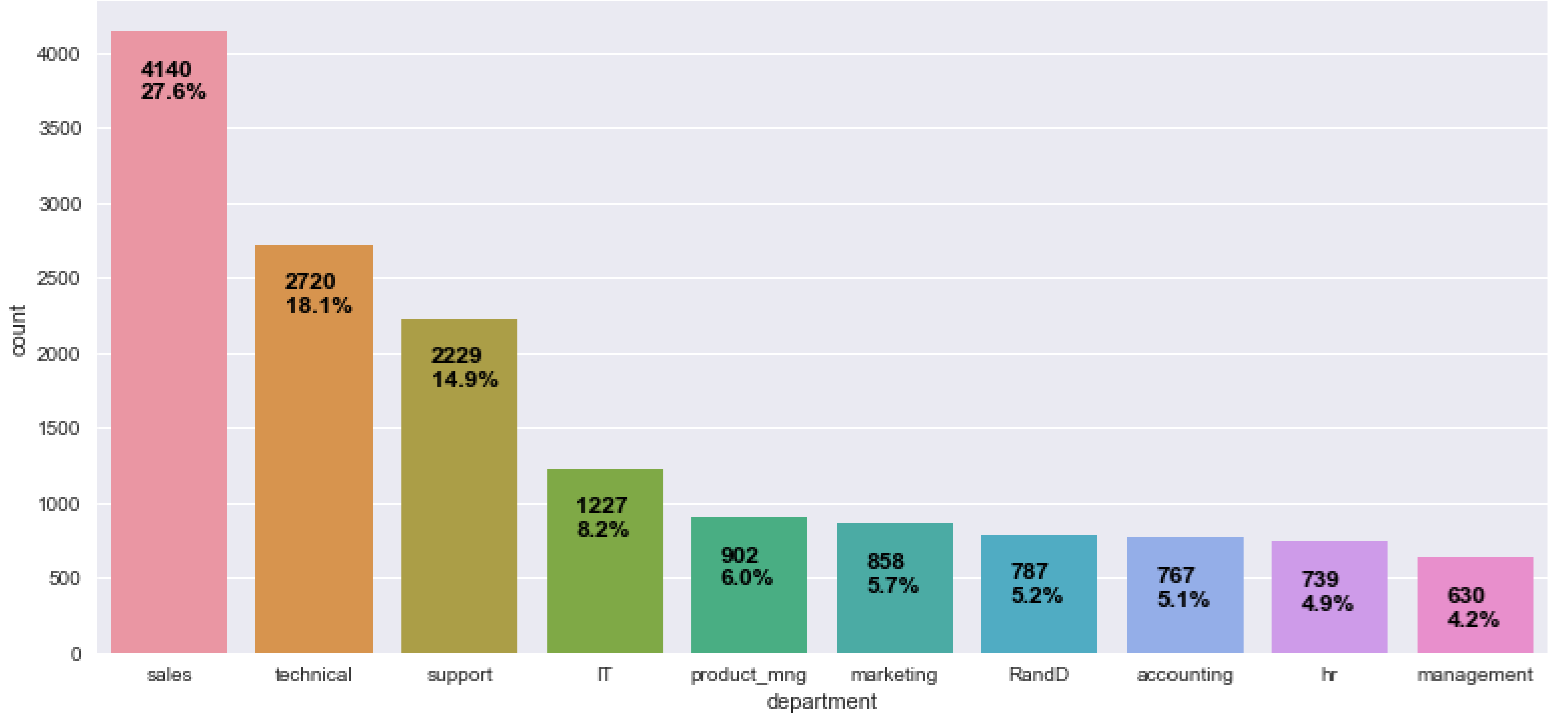
```
Chi Squared:  57.2627339495
p value:  1.08970012479e-11
Degrees of Freedom 4
Expected Frequency for The Not Promoted Employees: [   243.05167011  11184.94832989  11428.        ]
Expected Frequency for The Promoted Employees: [   75.94832989   3495.05167011   3571.        ]
```

- So, this is a significant result. Promotion within the last 5 years is definitely something the company must do for the employees that they want to retain.

Data Understanding

Observe the Department where they worked


```
fig, axs = plt.subplots(figsize=(13, 6))

department_plt = sns.countplot(hr_data.department, order = hr_data.department.value_counts().index);

annotate_bars(bar_plt=department_plt, bar_plt_var=hr_data.department, x_offset=0.2, y_offset=450,
              txt_color="black")
```

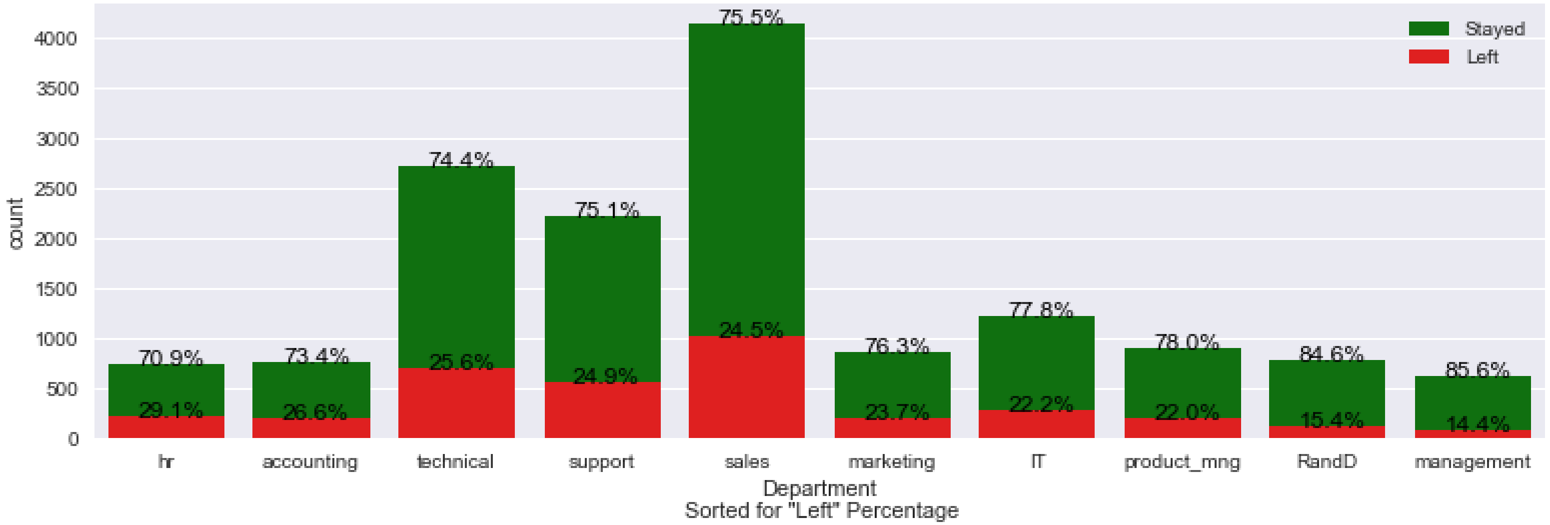


- Sales department has the highest # of employee’s leaving, but let us investigate this fact

- We observe the distribution over the departments

```
#Departments & Who Left; Is there a pattern?
fig, axs = plt.subplots(figsize=(13, 4))

#Order the bars descendingly according to the PERCENTAGE % of those who left in each department
total_employees_by_dept = hr_data.groupby(["department"]).satisfaction_level.count()
left_count_by_dept = hr_data[hr_data["left"] == 1].groupby(["department"]).satisfaction_level.count()
percentages_left_by_dept = (left_count_by_dept / total_employees_by_dept).sort_values(ascending=False)
axe_name_order = percentages_left_by_dept.index

department_plt = sns.countplot(hr_data.department, order = axe_name_order, color='g');
sns.countplot(employees_left.department, order = axe_name_order, color='r');

department_plt.legend(labels=['Stayed', 'Left'])
department_plt.set(xlabel='Department\n Sorted for "Left" Percentage')

#Annotate the percentages of those who stayed. It was more straightforward to loop for each
#category (left, stayed) than doing all the work in one loop.
#The zip creates an output that is equal to the shortest parameter, so we do not need to adjust the
#patches length, since the loop will stop after finishing the columns of those who stayed
for p, current_column in zip(department_plt.patches, axe_name_order):
    current_column_total = hr_data[hr_data['department'] == current_column].department.count()
    stayed_count = p.get_height() - employees_left[employees_left['department'] == current_column].department.count()
    department_plt.annotate(str(round( (100.0* stayed_count) /current_column_total, 1) )+ "%",
                            (p.get_x() + 0.2, p.get_height()-10),
                            color='black', fontsize=12)
```

```python
#In this loop, we want to use the patches located on the second half of patches list, which are the
#bars for those who left.
for p, current_column in zip(department_plt.patches[int(len(department_plt.patches)/2):], axe_name_order):
    current_column_total = hr_data[hr_data['department'] == current_column].department.count()
    left_count = p.get_height()
    department_plt.annotate(str(round( (100.0* left_count) /current_column_total, 1) )+ "%",
                            (p.get_x() + 0.2, p.get_height()-10),
                            color='black', fontsize=12)
```



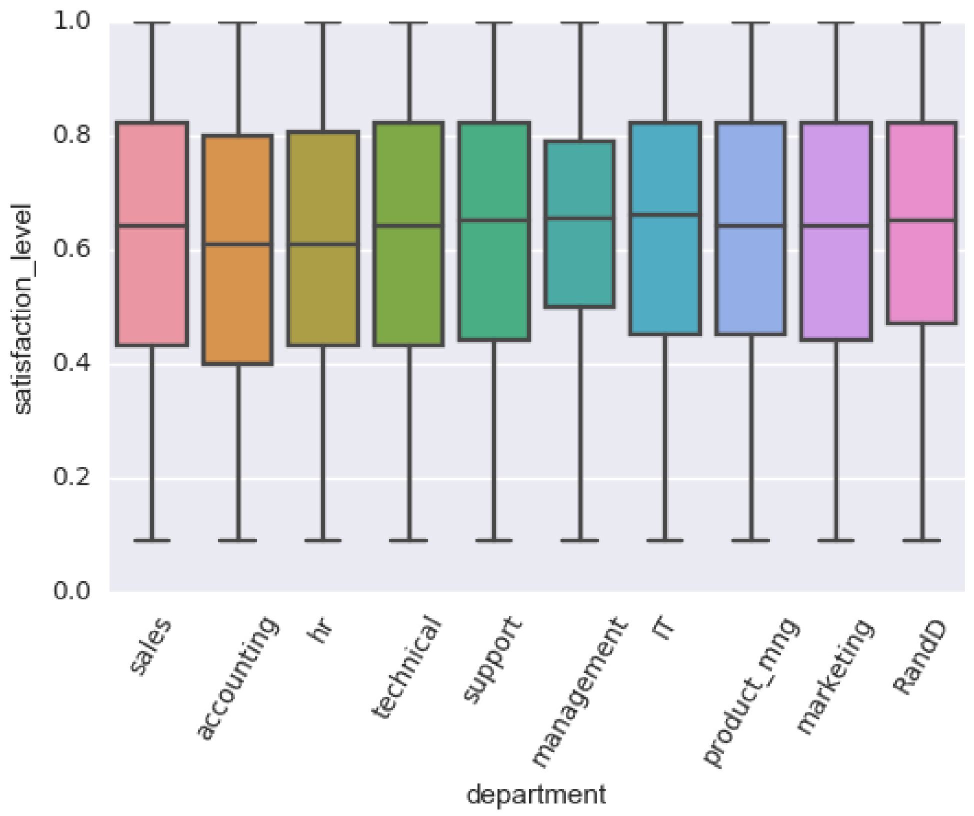
- Leaving rate in HR is a little high, R&D and management are low.

Data Understanding

Distribution of satisfaction in departments


```
g=sns.boxplot(x='department', y='satisfaction_level', data=hr_data) #Draw a box plot to show
#distributions (satisfaction level) with respect to categories (departments). A box plot (or
#box-and-whisker plot) shows the distribution of quantitative data in a way that facilitates
#comparisons between variables or across levels of a categorical variable. The box shows the
#quartiles of the dataset while the whiskers extend to show the rest of the distribution,
#except for points that are determined to be #"outliers" using a method that is a function
#of the inter-quartile range.
for item in g.get_xticklabels():  #rotate the x-axis for better reading experience
    item.set_rotation(60) # No'in brackets = degrees in positive direction
```

HR and accounting, the departments that have the highest rates of leaving, have slightly lower median satisfaction levels than the rest of the departments.



Maximum 1,5×IQR (depends on the data)

Median (= 50%-Quantil)

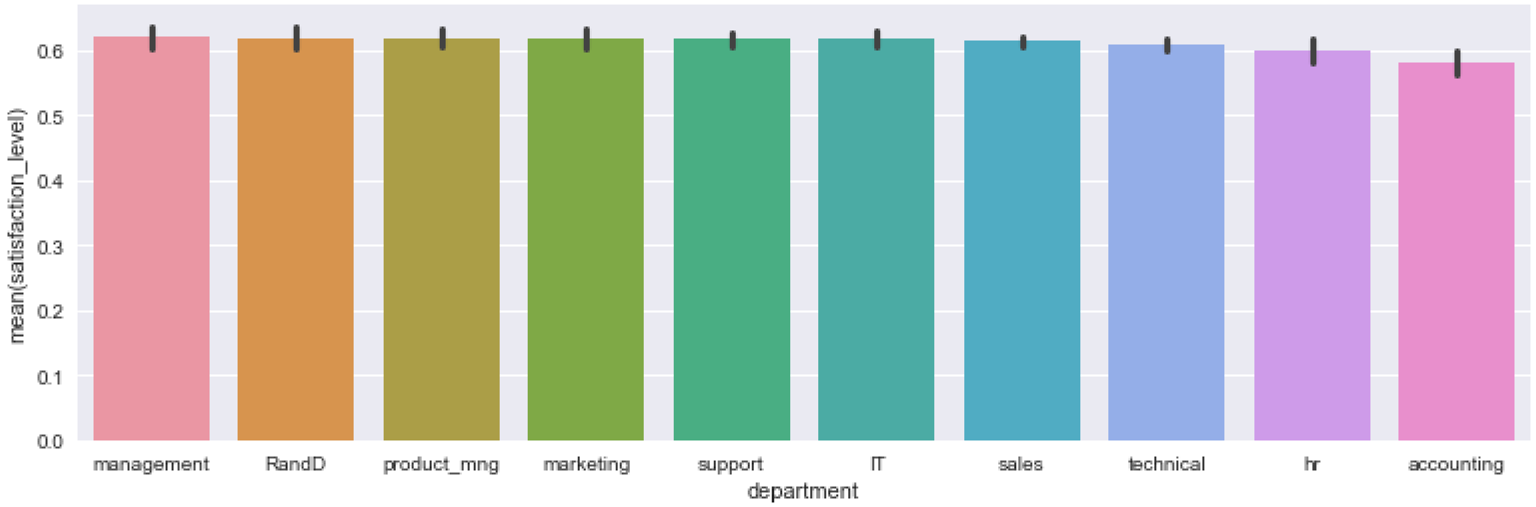
25%-Quantil (to both sides)

Minimum (as Max.) usually 1,5xIQR

Data Understanding

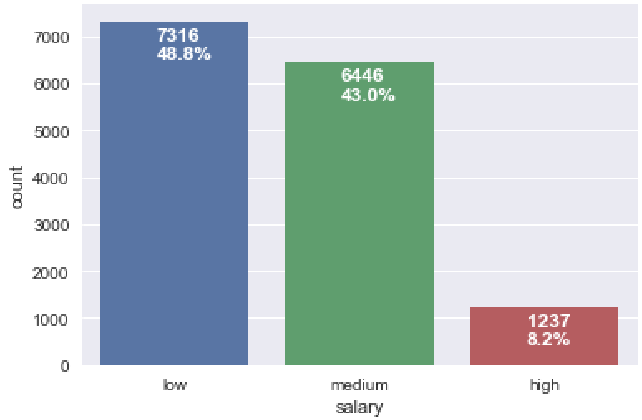
Distribution of satisfaction in departments


```
fig, axs = plt.subplots(figsize=(13, 4))

bar_plot = sns.barplot(x=hr_data.department, y=hr_data.satisfaction_level,
                       order=get_ordered_group_index(hr_data, 'department', 'satisfaction_level'))
```

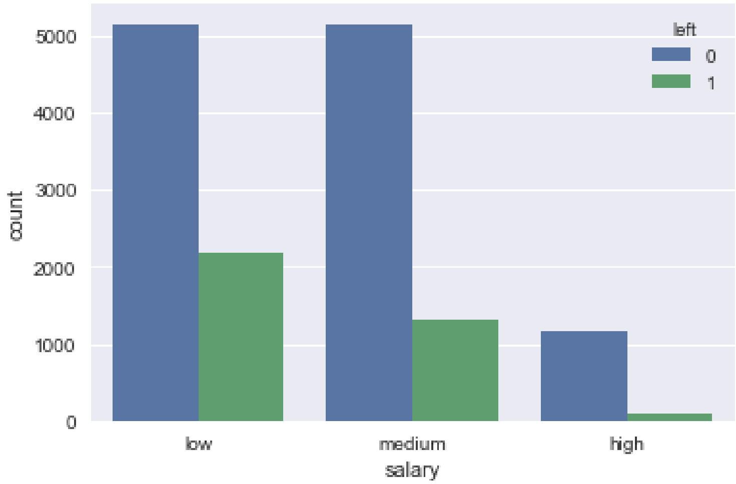


- We check the salary among our staff.

```
department_plt = sns.countplot(hr_data.salary, order = hr_data.salary.value_counts().index);

for p in department_plt.patches:
    department_plt.annotate(str( int(p.get_height()) ) + "\n" + str(round(
                (100.0* p.get_height()) /hr_data.salary.count(), 1) )+ "%",
                                (p.get_x() + 0.3, p.get_height()-800),
                                color='white', fontsize=12, fontweight='bold')
```




```
g=sns.countplot(x='salary', hue='left', data=hr_data) #shows the number of
#observations in each categorical bin using bars;
#here: the dependecy of staying from the catagories of salary
```

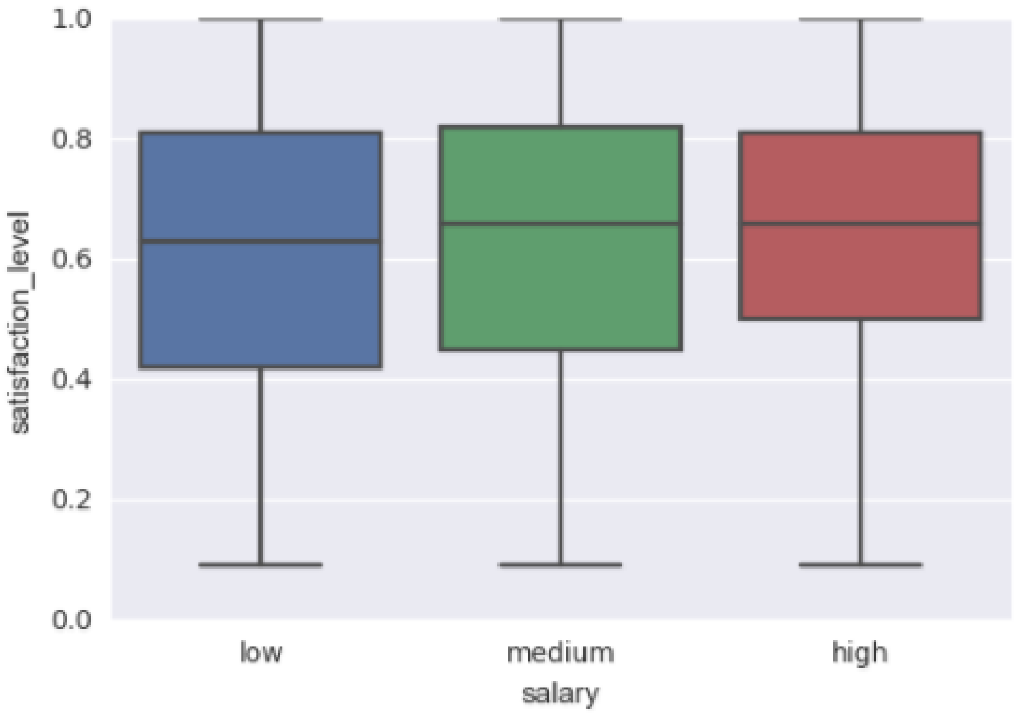


- Around half the employees had a low salary. The high salary employees made up about 8%

- We check the spread of **satisfaction level** between the different **salary** ranges.

```
sns.boxplot(x='salary', y='satisfaction_level', data=hr_data) #satisfaction level with respect to salary level
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2b72d195a90>
```

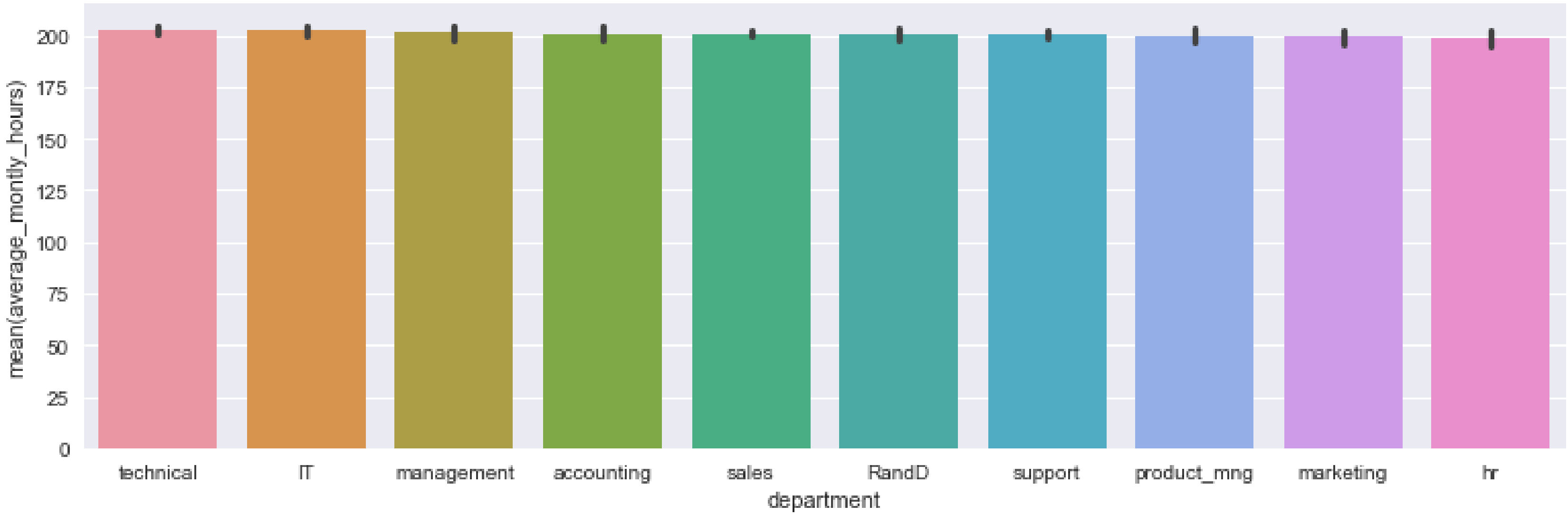


low salary has the lowest median satisfaction and the highest spread.

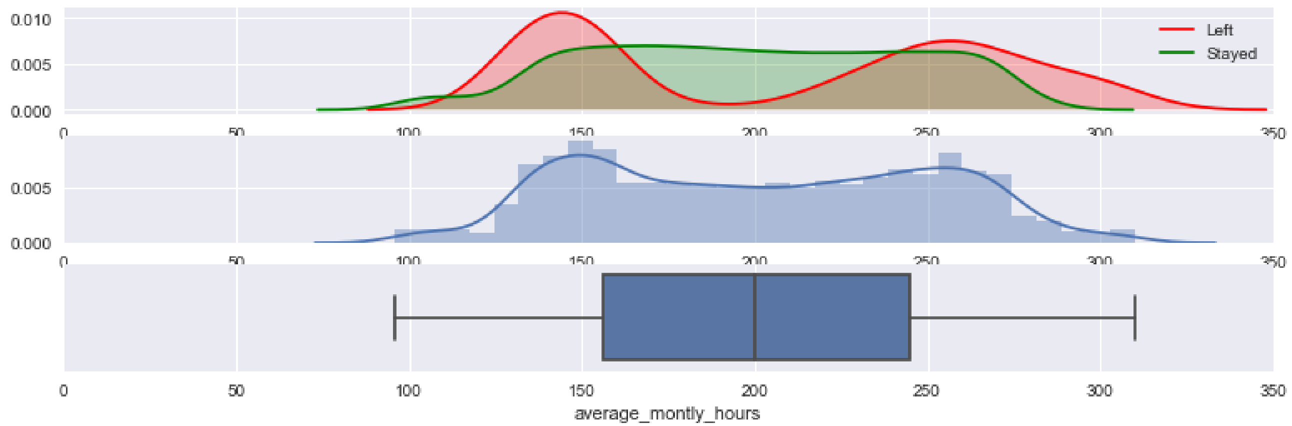
Data Understanding

Influence of the work load


```
#How Hard does Each Department Work?
fig, axs = plt.subplots(figsize=(13, 4))

bar_plot = sns.barplot(x=hr_data.department, y=hr_data.average_montly_hours,
                       order=get_ordered_group_index(hr_data, 'department', 'average_montly_hours') )
```



```
fig, axs = plt.subplots(nrows=3, figsize=(13, 4))

sns.kdeplot(employees_left.average_montly_hours, ax=axs[0], shade=True, color="r") #tool in seaborn for
#examining univariate and bivariate distributions
kde_plot = sns.kdeplot(employees_stayed.average_montly_hours, ax=axs[0], shade=True, color="g")
kde_plot.legend(labels=['Left', 'Stayed'])

hist_plot = sns.distplot(hr_data.average_montly_hours, ax=axs[1])#plot a univariate distribution
box_plot = sns.boxplot(hr_data.average_montly_hours, ax=axs[2]) #Draw a box plot to show distributions

kde_plot.set(xlim=(0,350)) #set or query x-axis limits
hist_plot.set(xlim=(0,350))
box_plot.set(xlim=(0,350));
```

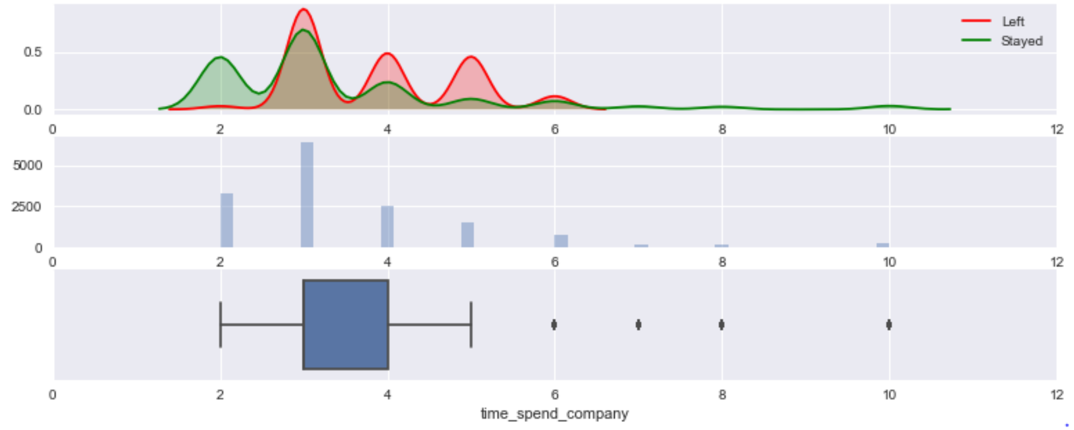


- Another bimodal shape, one at around 150 hours a month and the other is a little over 250 hours a month. Some very high values, 300 (?!) hours a month → employee never takes any days off
- They work 10 hours a day. If they take one day off, it would be 11.5 hours of work, and two days weekend would mean they work for 13 hours a day.

Data Understanding

Number of Years Working for the Company


```
#Number of Years Working for the Company
fig, axs = plt.subplots(nrows= 3, figsize=(13, 5))

sns.kdeplot(employees_left.time_spend_company, ax=axs[0], shade=True, color="r")
kde_plot = sns.kdeplot(employees_stayed.time_spend_company, ax=axs[0], shade=True, color="g")
kde_plot.legend(labels=['Left', 'Stayed'])

hist_plot = sns.distplot(hr_data.time_spend_company, ax=axs[1], kde=False)
box_plot = sns.boxplot(hr_data.time_spend_company, ax=axs[2])

kde_plot.set(xlim=(0,12))
hist_plot.set(xlim=(0,12))
box_plot.set(xlim=(0,12));
```



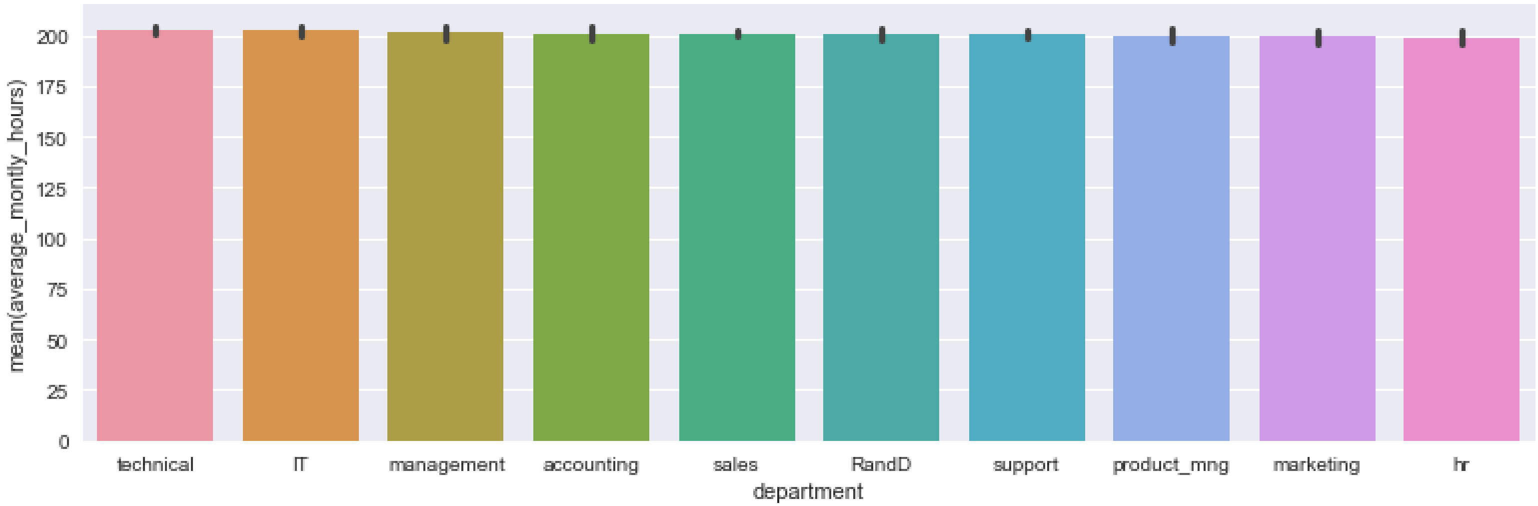
- Interestingly, none of the surveyed employees worked for less than two years. That raises some concerns about the randomness of picking the subjects, there was a certain bias for those who stayed longer.

- Going back to the dataset's welcome page, it was clearly stated that this is a simulated dataset, so maybe that is a reason.

Data Understanding

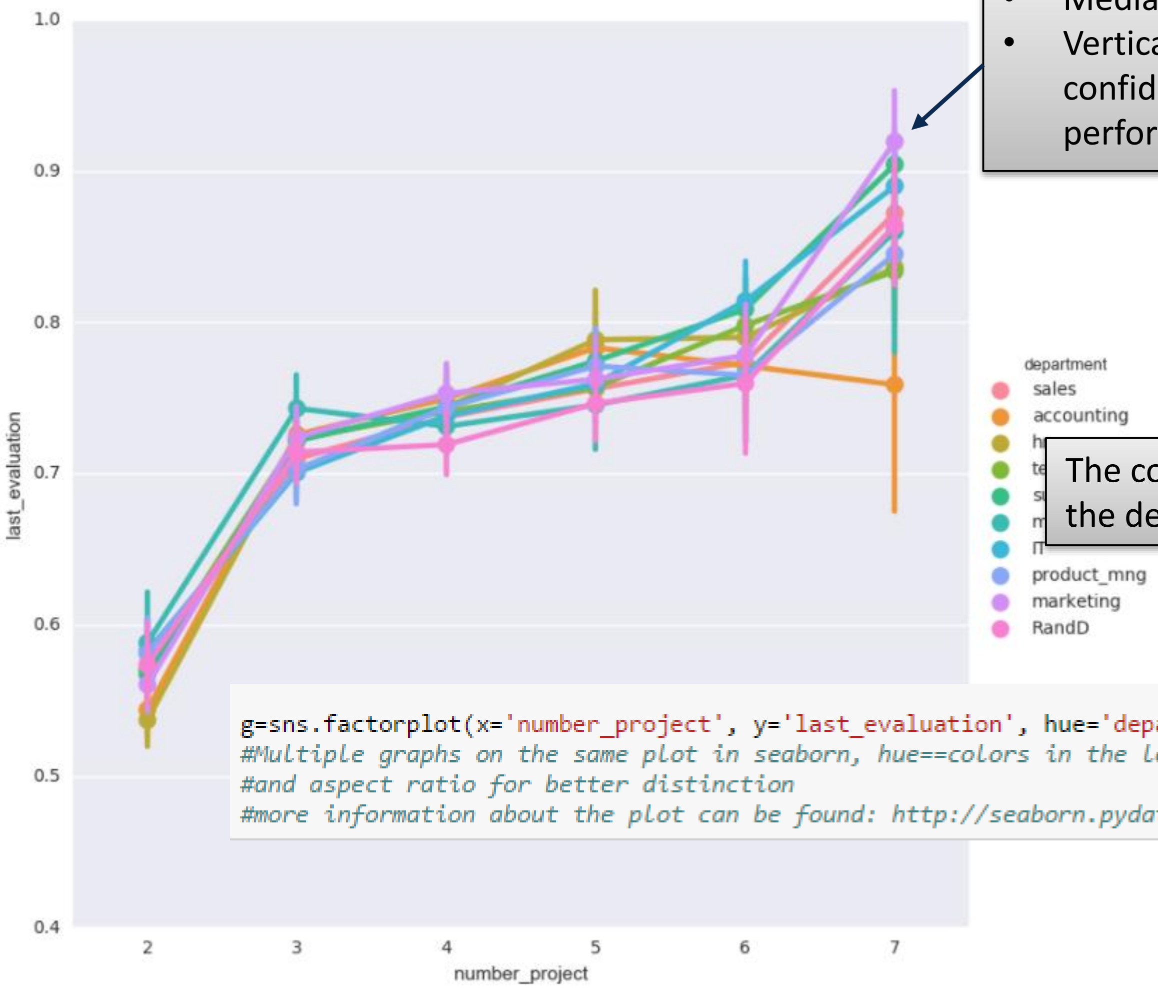
Work intensity of each department


```
#How Hard does Each Department Work?
fig, axs = plt.subplots(figsize=(13, 4))

bar_plot = sns.barplot(x=hr_data.department, y=hr_data.average_montly_hours,
                       order=get_ordered_group_index(hr_data, 'department', 'average_montly_hours') )
```



- Seems that they all work equally hard, around 10 hours a day on average (Assuming two days off each week)

- Something that may impact employee perception in the company is the number of projects they are assigned



- Median (= 50%-Quantil)
- Vertical line is showing the confidence of the last performance testing

The colors show the departments

Height (in inches) of each facet

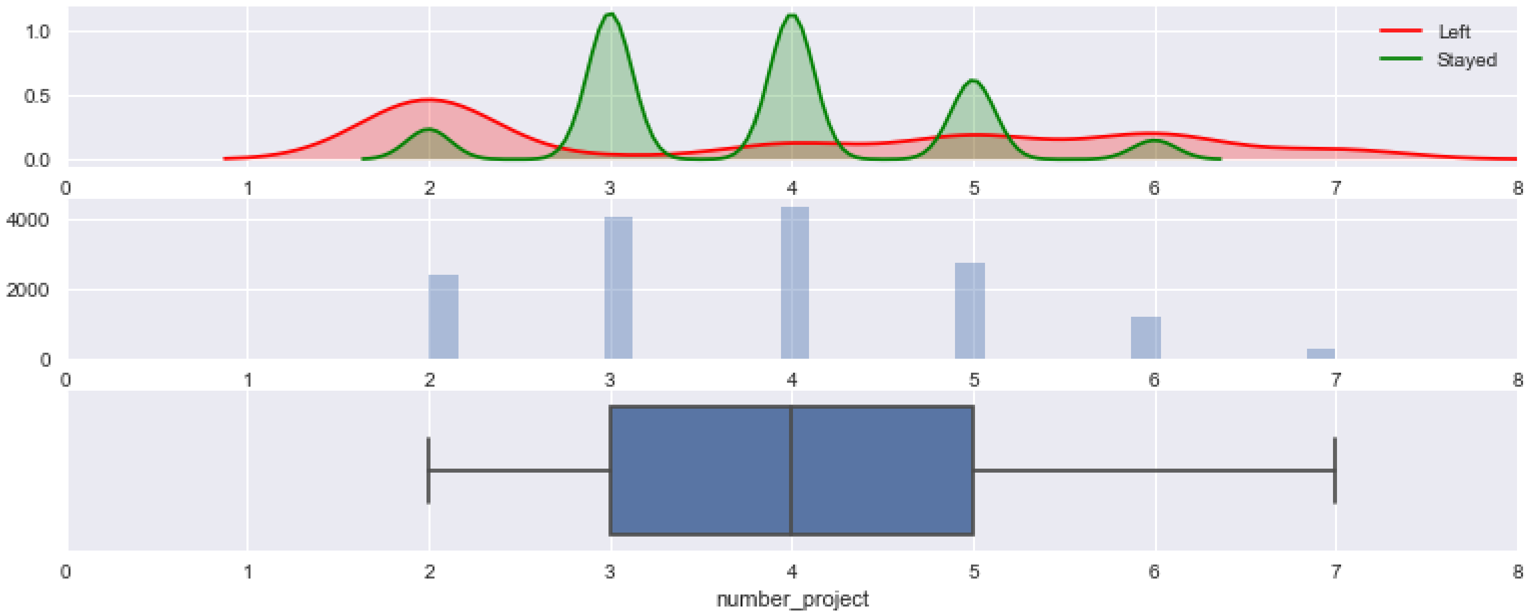
datasource

width of each facet in inches.

```
g=sns.factorplot(x='number_project', y='last_evaluation', hue='department', data=hr_data,size=8, aspect=1)
#Multiple graphs on the same plot in seaborn, hue==colors in the legend, size (is the extension of x-axis)
#and aspect ratio for better distinction
#more information about the plot can be found: http://seaborn.pydata.org/generated/seaborn.factorplot.html
```

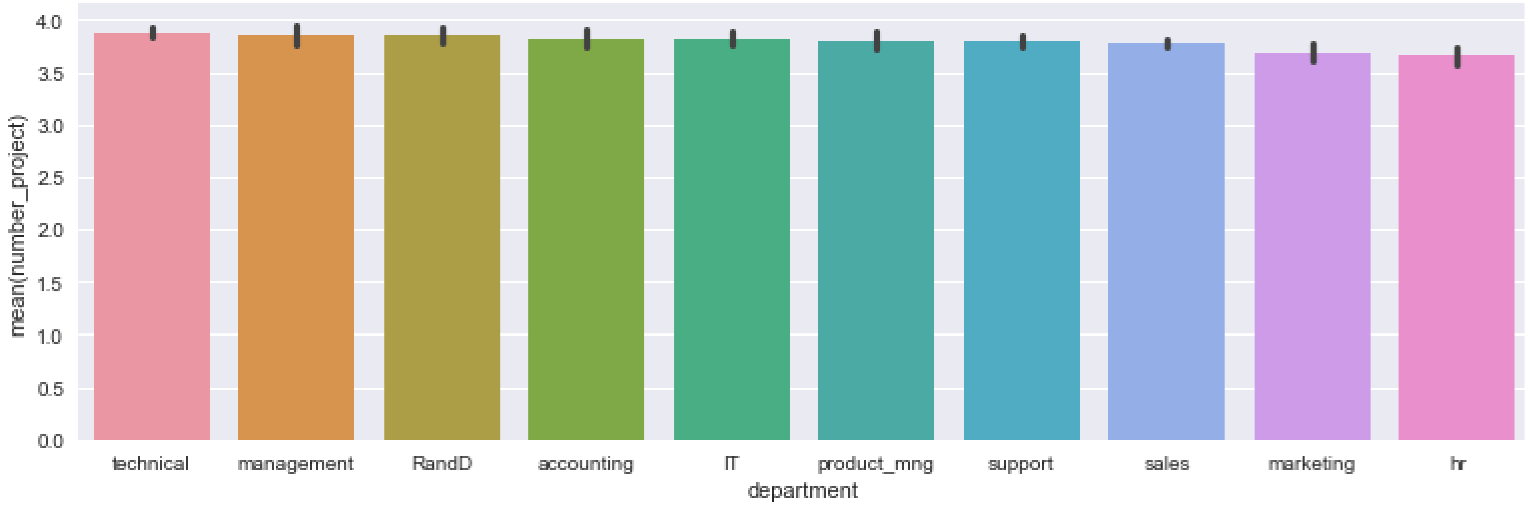
Data Understanding
Number of Projects


```
fig, axs = plt.subplots(nrows= 3, figsize=(13, 5))

sns.kdeplot(employees_left.number_project, ax=axs[0], shade=True, color="r")
kde_plot = sns.kdeplot(employees_stayed.number_project, ax=axs[0], shade=True, color="g")
kde_plot.legend(labels=['Left', 'Stayed'])

hist_plot = sns.distplot(hr_data.number_project, ax=axs[1], kde=False)
box_plot = sns.boxplot(hr_data.number_project, ax=axs[2])

kde_plot.set(xlim=(0,8))
hist_plot.set(xlim=(0,8))
box_plot.set(xlim=(0,8));
```



Data Understanding
Number of years working for the company


```
#How Many Projects are Assigned on Average per Employee?
fig, axs = plt.subplots(figsize=(13, 4))

bar_plot = sns.barplot(x=hr_data.department, y=hr_data.number_project,
                       order=get_ordered_group_index(hr_data,'department', 'number_project'))
```

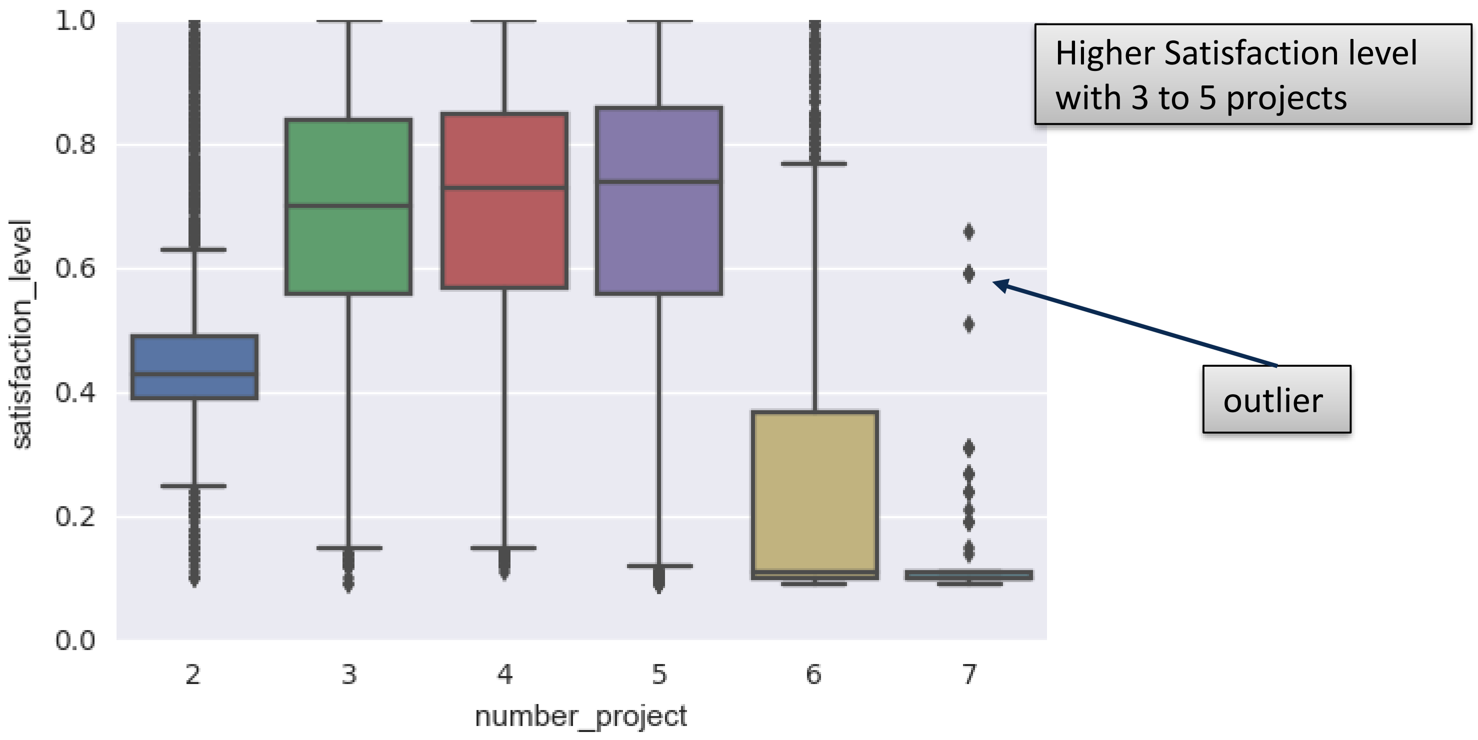


Data Understanding
Graphical Analysis

- It seems very clear that evaluation scores are affected by the number of projects assigned to the employee. There is a peculiar trend in accounting - they have a lower last_evaluation score than the other departments at 7 projects.

```
sns.boxplot(x='number_project', y='satisfaction_level', data=hr_data_new)
```

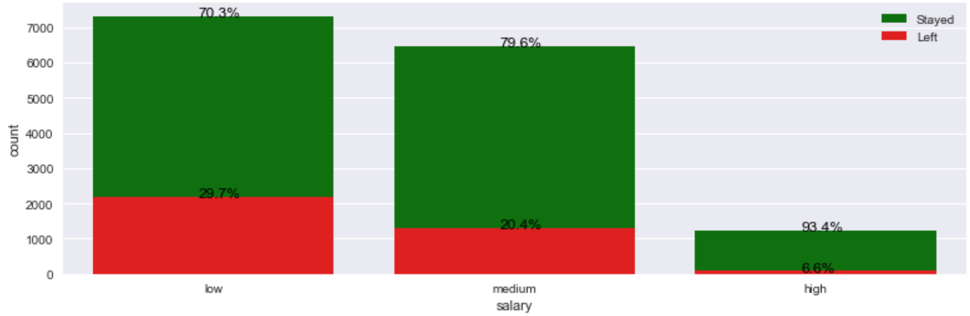
```
<matplotlib.axes._subplots.AxesSubplot at 0x2b72d758898>
```



General overview about salaries

- We compare the dependency between salaries and employee‘s motivation to leave?
- Is there a reliance to their timein company?

```
fig, axs = plt.subplots(figsize=(13, 4))

axe_name_order = hr_data.salary.value_counts().index

salary_plt = sns.countplot(hr_data.salary, order = axe_name_order, color='g');
sns.countplot(employees_left.salary, order = axe_name_order, color='r');

salary_plt.legend(labels=['Stayed', 'Left'])

#Annotate the percentages of those who stayed. It was more straightforward to loop for each
#category (left, stayed) than doing all the work in one loop. The zip creates an output that
#is equal to the shortest parameter, so we do not need to adjust the patches length, since
#the loop will stop after finishing the columns of those who stayed
for p, current_column in zip(salary_plt.patches, axe_name_order):
    current_column_total = hr_data[hr_data['salary'] == current_column].salary.count()
    stayed_count = p.get_height() - employees_left[employees_left['salary'] == current_column].salary.count()
    salary_plt.annotate(str(round( (100.0* stayed_count) /current_column_total, 1) )+ "%",
                        (p.get_x() + 0.35, p.get_height()-10),
                        color='black', fontsize=12)

#In this loop, we want to use the patches located on the second half of patches list, which are the
#bars for those who left.
for p, current_column in zip(salary_plt.patches[int(len(salary_plt.patches)/2):], axe_name_order):
    current_column_total = hr_data[hr_data['salary'] == current_column].salary.count()
    left_count = p.get_height()
    salary_plt.annotate(str(round( (100.0* left_count) /current_column_total, 1) )+ "%",
                        (p.get_x() + 0.35, p.get_height()-10),
                        color='black', fontsize=12)
```



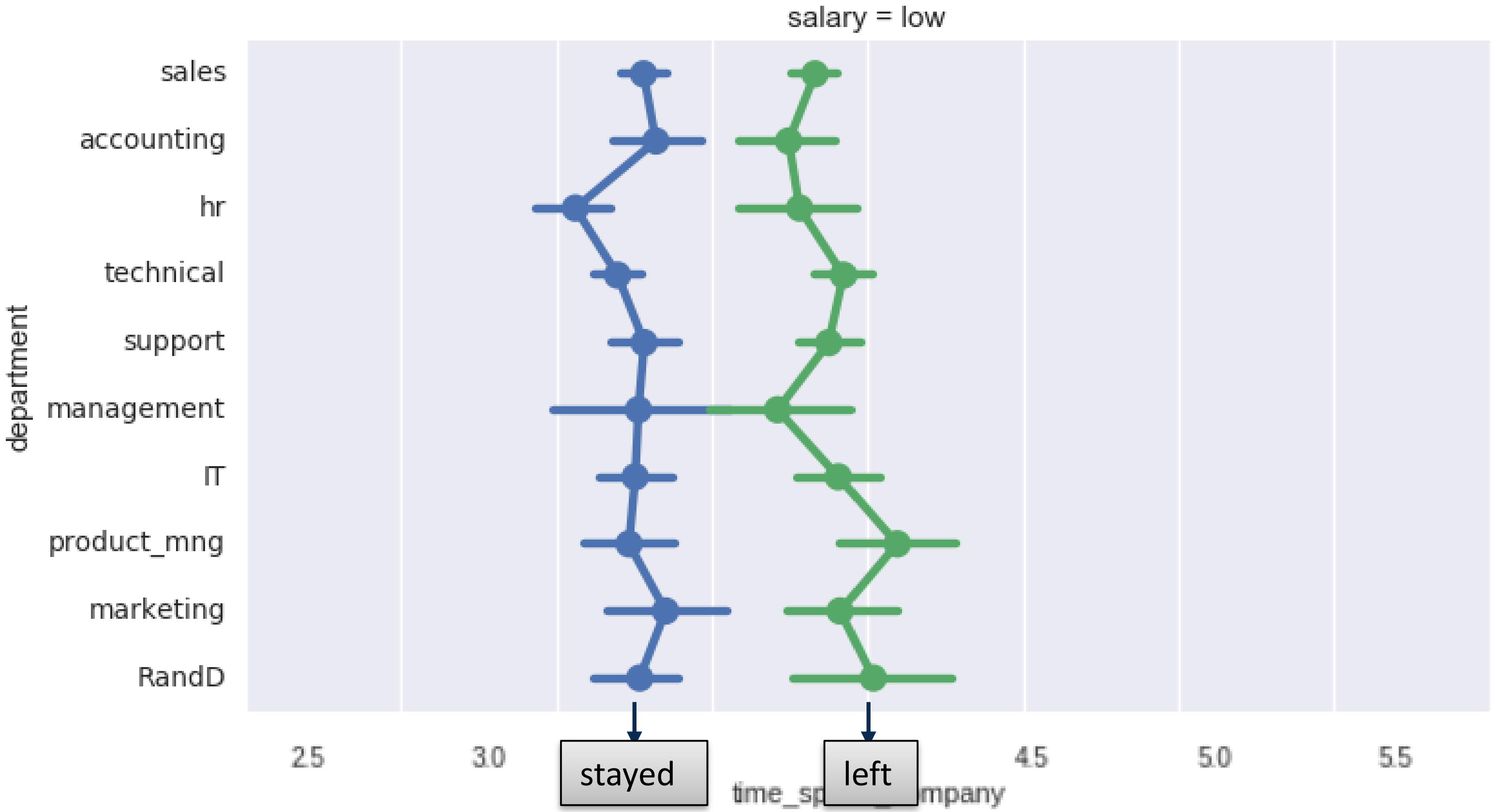
This makes sense, people with high salaries are less motivated to leave.

Data Understanding

Graphical Analysis (1/3)

- We take a look at time spent at the company and the effect of that on leaving; it is also influenced by the salary level (people tend to stay longer when it is high)

```
timeplot = sns.factorplot(x='time_spend_company', hue='left', y='department', row='salary', data=hr_data, aspect=2)
```

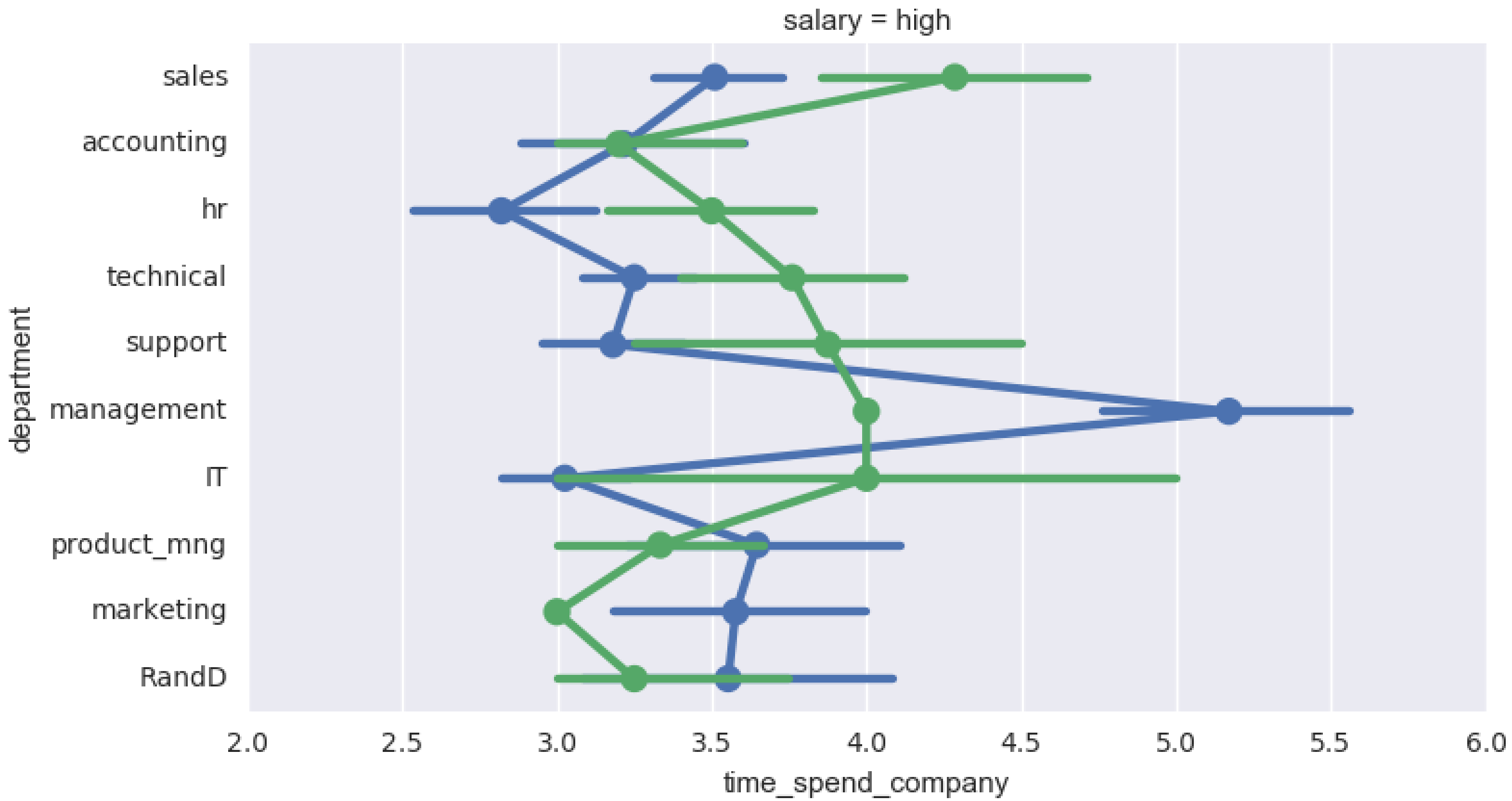


Data Understanding

Graphical Analysis (2/3)

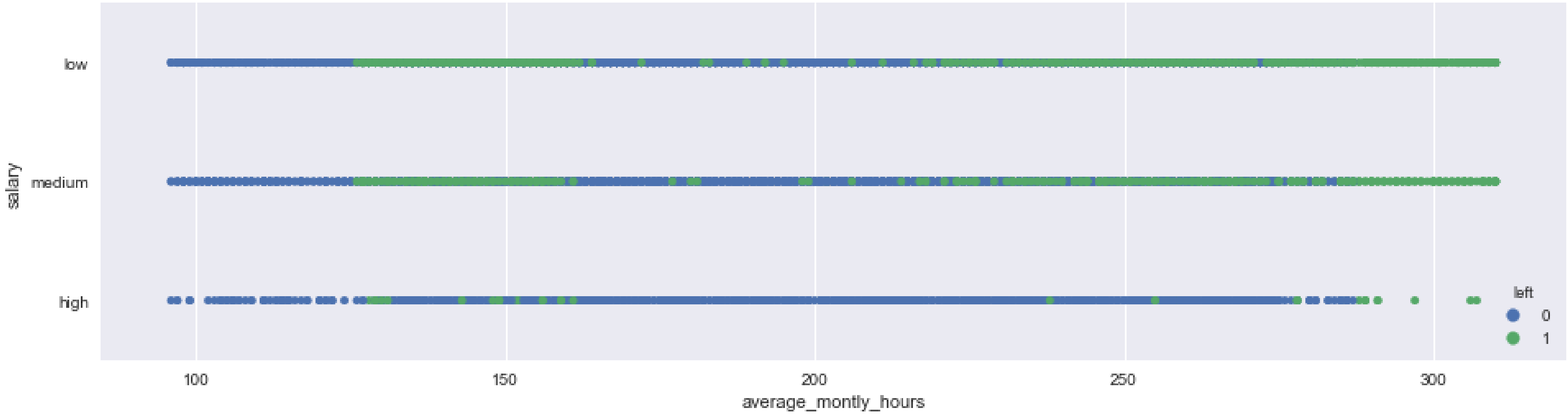
Data Understanding

Graphical Analysis (3/3)



- Those that leave tend to have spent more time at the company. For those with high salaries, leaving depends on the department. At a high salary level, time spent doesn't vary in accounting for those that left versus those that haven't but it varies pretty wildly for the support and IT departments

```
fig, axs = plt.subplots(figsize=(16, 4))

sns.stripplot(y = 'salary', x='average_montly_hours', hue='left', data=hr_data);
```

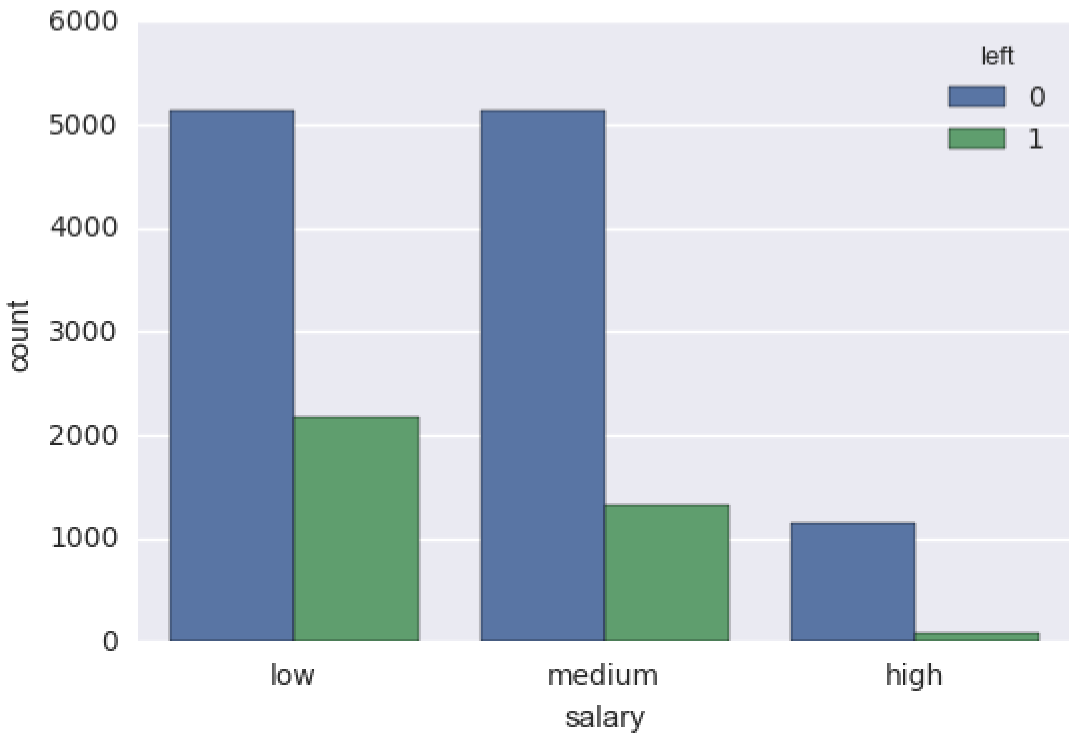


- Now it is time to have a closer look to the clusters seen in the cross-plot. For low and medium salaries, too much work or just below the standard 40 hours\week work have a high concentration of leaving. For the high salaries, the rate of leaving the company is low anyway, so although that the green dots seem to be at around the same values, they are just too little to compel attention.

- Salary is likely to have a high impact on leaving. In fact, it is likely that both R&D and management, the two departments with lower leaving rates, have high salaries.

```
sns.countplot(x='salary', hue='left', data=hr_data) #shows the number of observations in
#each categorical bin using bars; here: the dependecy of staying from the catagory of salaries
```

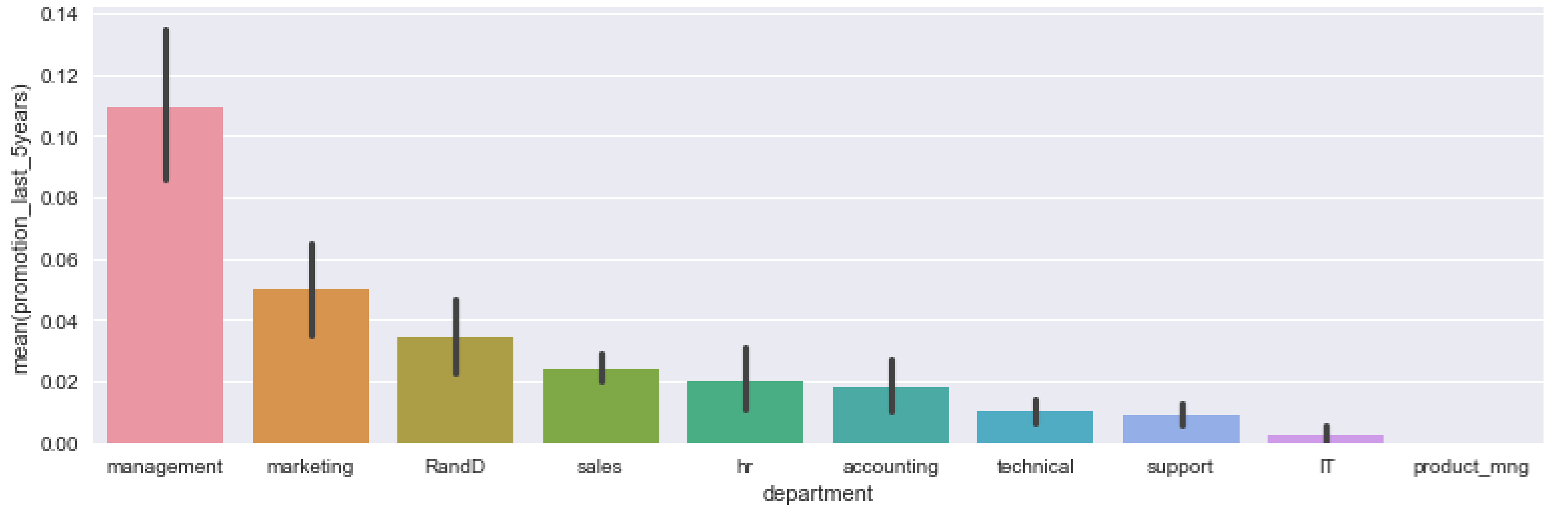
```
<matplotlib.axes._subplots.AxesSubplot at 0x2b7304e7080>
```



Data Understanding

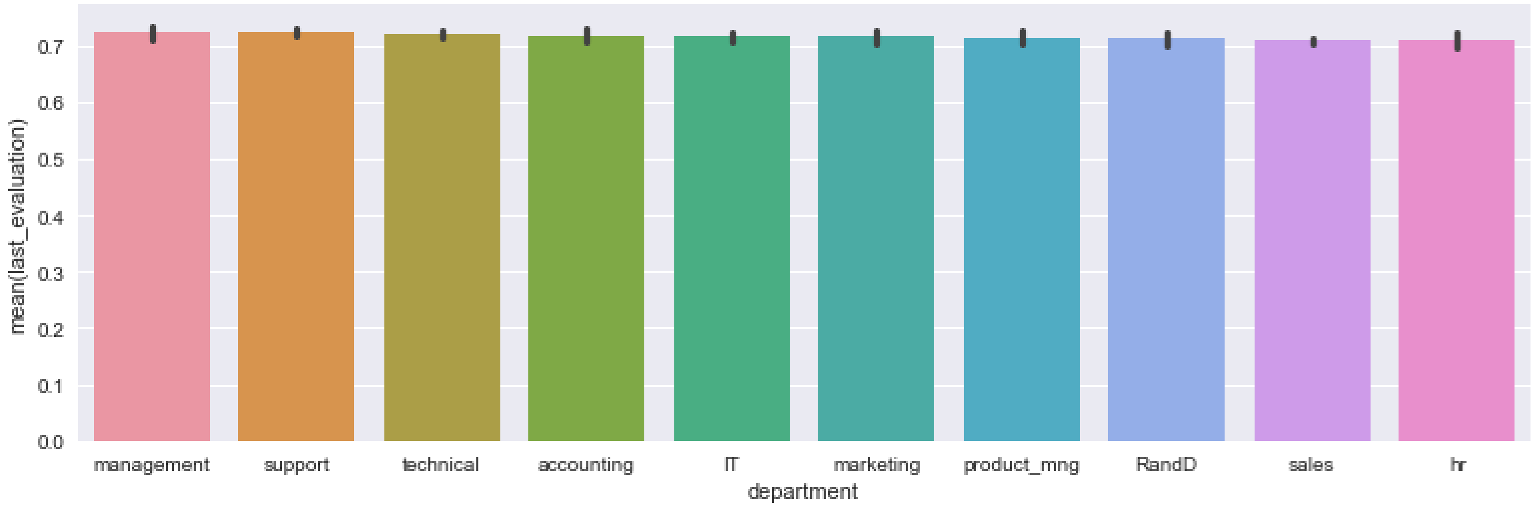
Graphical Analysis of promotions


```python
#Promotions in the departments
fig, axs = plt.subplots(figsize=(13, 4))

bar_plot = sns.barplot(x=hr_data.department, y=hr_data.promotion_last_5years,
                       order=get_ordered_group_index(hr_data, 'department', 'promotion_last_5years'))
```



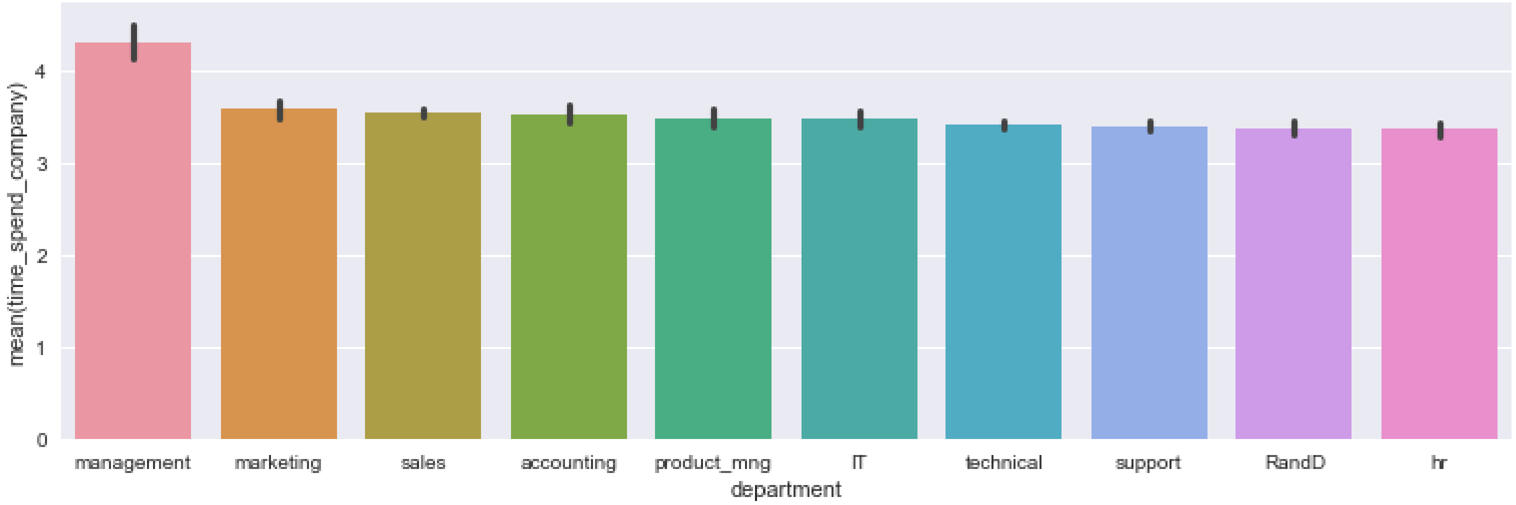
- Is the Management Department Really Outperforming the Rest?

```
#Evaluation of the Management Department
fig, axs = plt.subplots(figsize=(13, 4))

bar_plot = sns.barplot(x=hr_data.department, y=hr_data.last_evaluation,
                       order=get_ordered_group_index(hr_data,'department', 'last_evaluation'))
```



- Did they Stayed Longer?

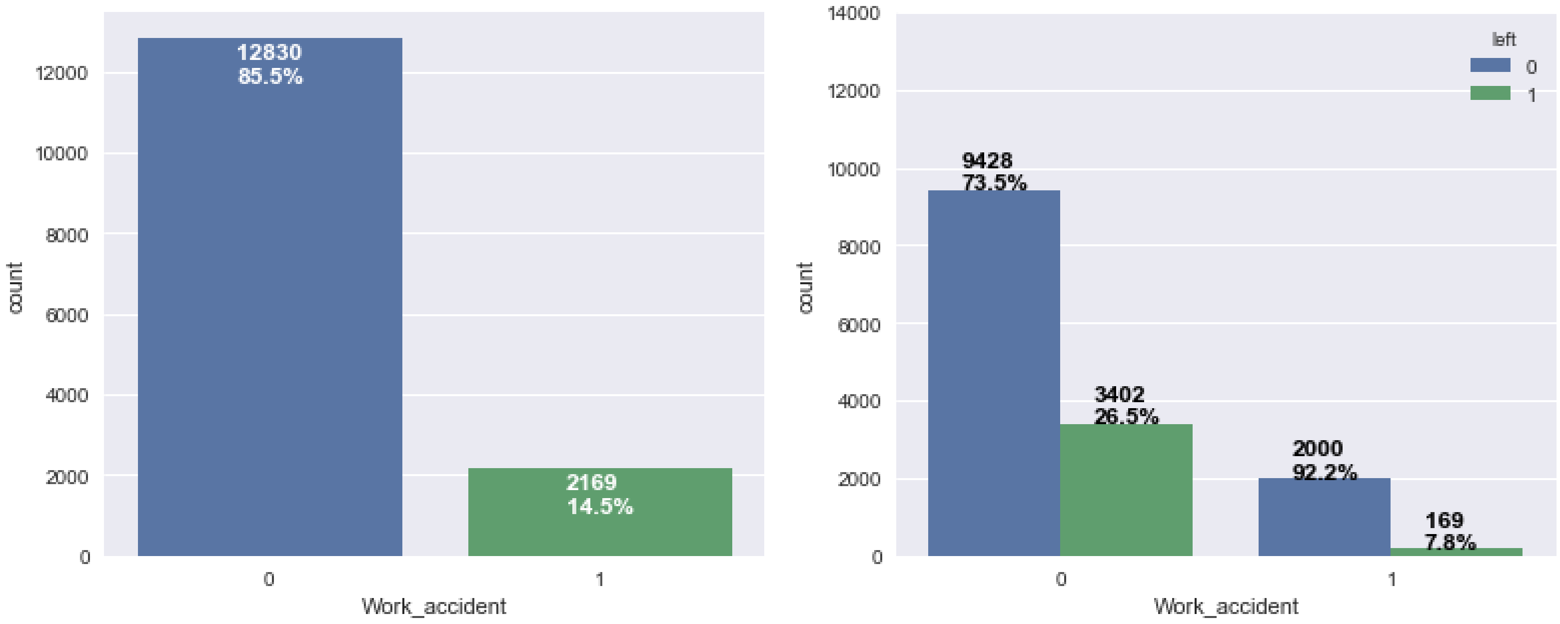
```
fig, axs = plt.subplots(figsize=(13, 4))

bar_plot = sns.barplot(x=hr_data.department, y=hr_data.time_spend_company,
                       order=get_ordered_group_index(hr_data, 'department', 'time_spend_company'))
```



Data Understanding

Employees who Left their Jobs

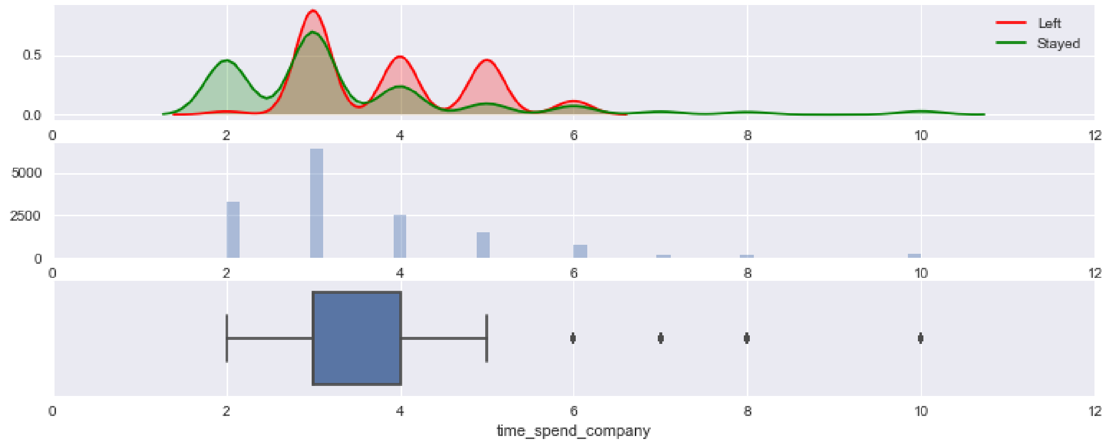

```
fig, axs = plt.subplots(ncols= 2, figsize=(13, 5)) #set the number and size of the diagram spaces

work_accidents_plt = sns.countplot(hr_data.Work_accident, ax=axs[0]);
annotate_bars(bar_plt=work_accidents_plt, bar_plt_var=hr_data.Work_accident,
              x_offset=0.3, y_offset=1100)

bar_plot = sns.countplot(x=hr_data.Work_accident, hue=hr_data.left, ax=axs[1])
annotate_bars(bar_plt=bar_plot, by=hr_data.Work_accident, bar_plt_var=hr_data.Work_accident,
              x_offset=0.1, txt_color="black")
bar_plot.set(ylim=(0,14000));
```



Data Understanding

Number of Years Working for the Company


```
fig, axs = plt.subplots(nrows= 3, figsize=(13, 5))

sns.kdeplot(employees_left.time_spend_company, ax=axs[0], shade=True, color="r")
kde_plot = sns.kdeplot(employees_stayed.time_spend_company, ax=axs[0], shade=True, color="g")
kde_plot.legend(labels=['Left', 'Stayed'])

hist_plot = sns.distplot(hr_data.time_spend_company, ax=axs[1], kde=False)
box_plot = sns.boxplot(hr_data.time_spend_company, ax=axs[2])

kde_plot.set(xlim=(0,12))
hist_plot.set(xlim=(0,12))
box_plot.set(xlim=(0,12));
```



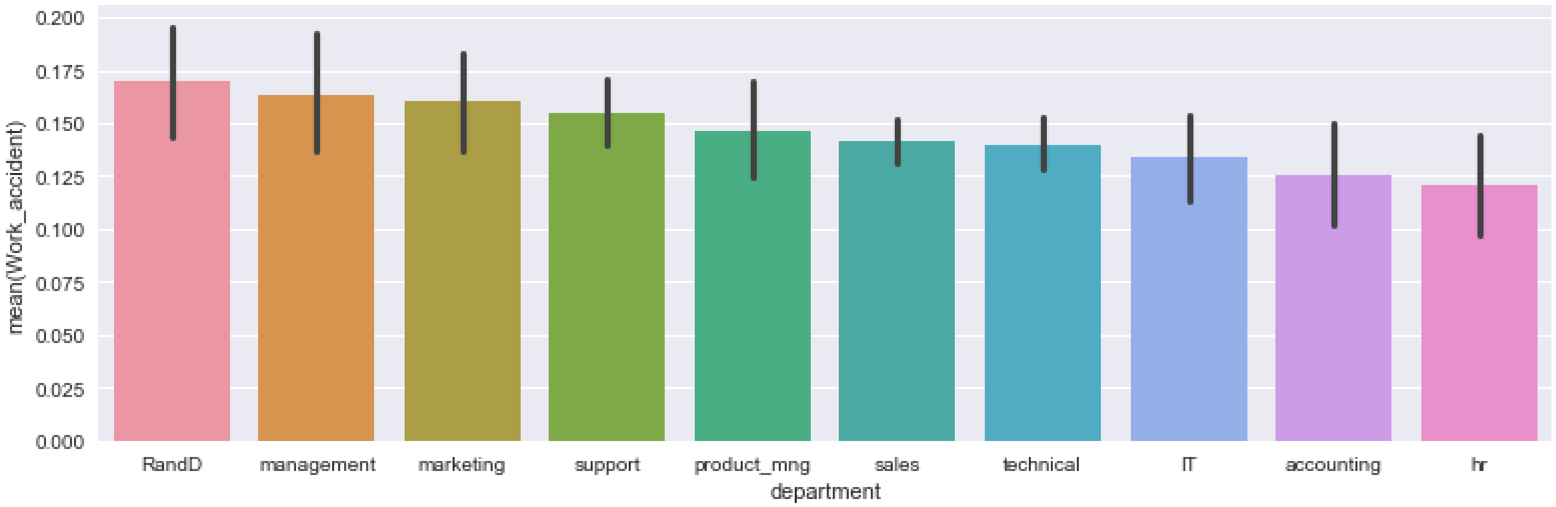
- Interestingly, none of the surveyed employees worked for less than two years. That raises some concerns about the randomness of picking the subjects, there was a certain bias for those who stayed longer. We have to take that into account, to start with. Going back to the dataset's welcome page, it was clearly stated that this is a simulated dataset, so maybe this is the reason.

Data Understanding

Observation of accidants

- Employees Who Suffered Work Related Accidents

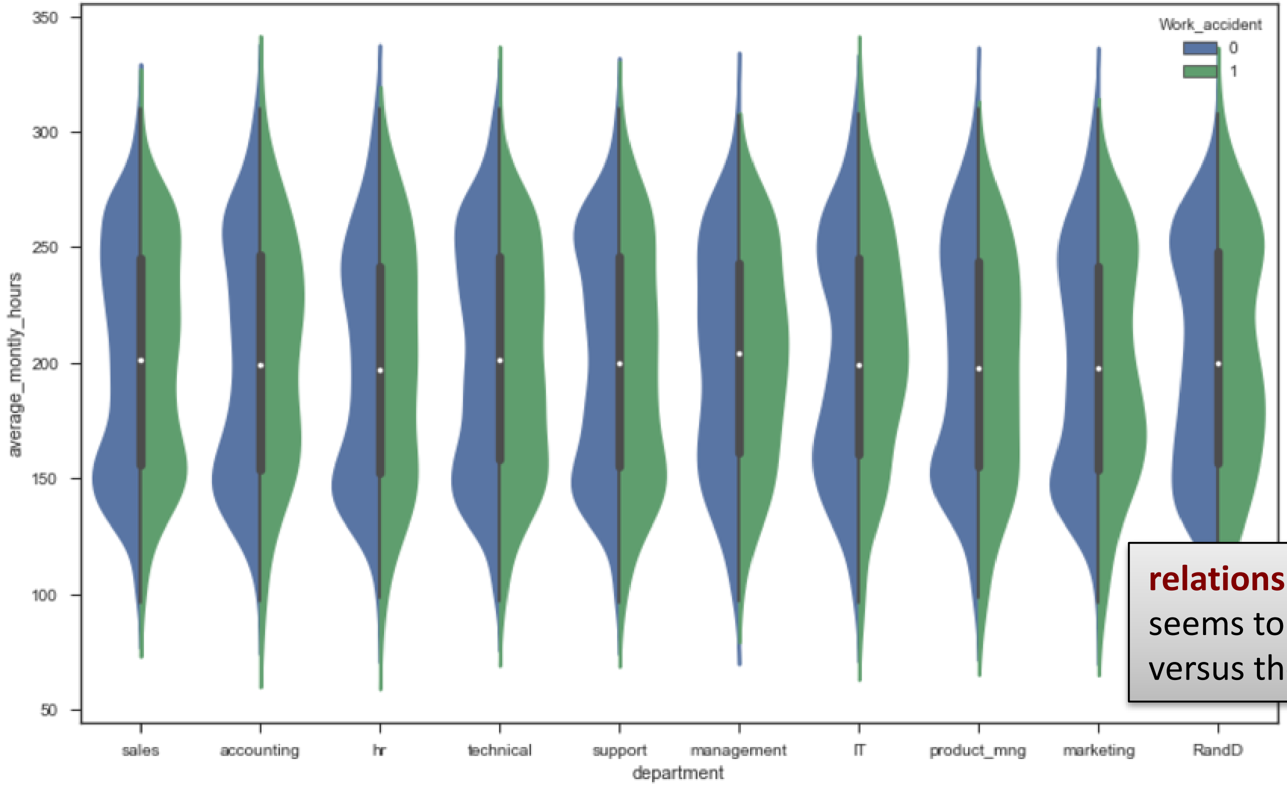
```
fig, axs = plt.subplots(figsize=(13, 4))

bar_plot = sns.barplot(x=hr_data.department, y=hr_data.Work_accident,
                       order=get_ordered_group_index(hr_data, 'department', 'Work_accident'))
```



Data Understanding

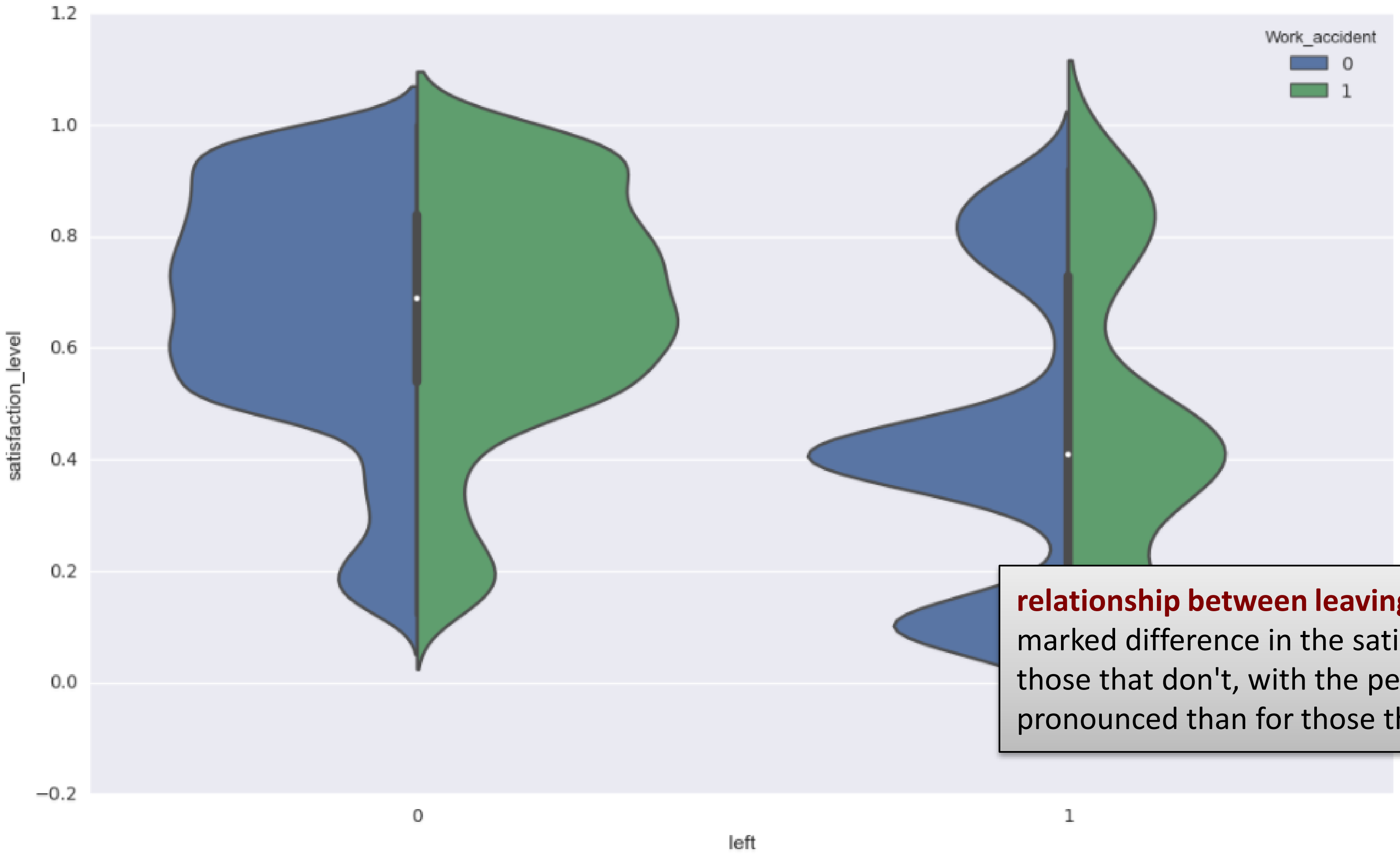
Graphical Analysis

```
accidentplot = plt.figure(figsize=(10,6)) #figure module provides the top-level Artist, the Figure,
#which contains all the plot elements, figsize-->w,h tuple in inches
#more information about the figure module can be found at: https://matplotlib.org/api/figure_api.html
accidentplotax = accidentplot.add_axes([0,0,1,1]) #Add an axes at position rect [left, bottom, width, height]
#where all quantities are in fractions of figure width and height.
accidentplotax = sns.violinplot(x='department', y='average_montly_hours',
                                hue='Work_accident', split=True, data = hr_data, jitter = 0.47)
#A violin plot plays a similar role as a box and whisker plot. It shows the distribution of quantitative
#data across several levels of one (or more) categorical variables such that those distributions can be compared.
#more information about parameters and using violinplots can be found in documntation:
#https://seaborn.pydata.org/generated/seaborn.violinplot.html
```



relationship between leaving, work accidents, and satisfaction level
seems to be bimodally distributed more often for those without work accidents versus those with (across the departments)

```
satisaccident = plt.figure(figsize=(10,6))   #relationship between leaving, work accidents, and satisfaction level.
#Let's check a similar plot to see the relationship between leaving, work accidents, and satisfaction level.
satisaccidentax = satisaccident.add_axes([0,0,1,1])
satisaccidentax = sns.violinplot(x='left', hue='Work_accident', y='satisfaction_level', split=True, data=hr_data)
```



relationship between leaving, work accidents, and satisfaction level
marked difference in the satisfaction level spreads of those that leave versus those that don't, with the peaks for those that left being slightly more pronounced than for those that have not had workplace accidents

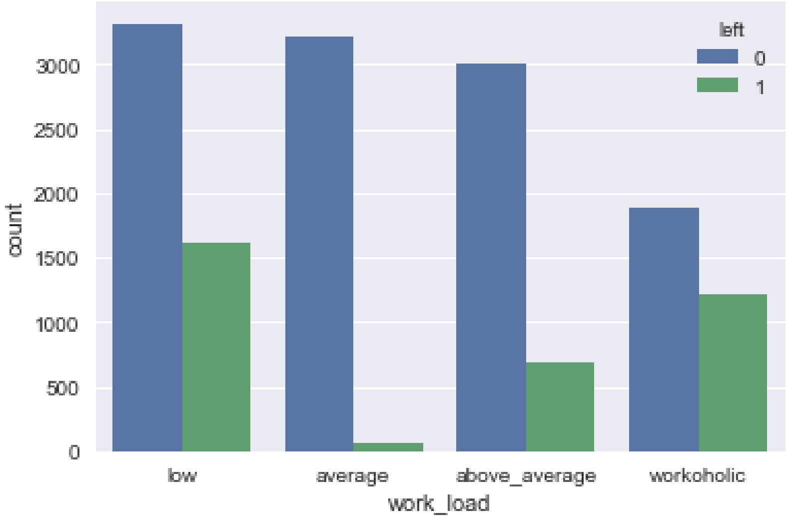
- If we divide working hours into four categories:

1	Those who work a regular 8 hours a day or less (< 168 a month, assuming that a calendar month have on average 21 days of work)
2	Those who work between 8 and 10 hours a day (168 < average_montly_hours < 210 a month)
3	Those who work between 10 and 12 hours a day (210 < average_montly_hours < 252 a month)
4	Those who work over 12 hours a day.

- and see how the salary goes with this effort:

```
#A function to bin the average monthly hours into the categories described above
def work_load_cat(avg_mnthly_hrs):
    work_load = "unknown"
    if avg_mnthly_hrs < 168:
        work_load = "low"
    elif (avg_mnthly_hrs >= 168) & (avg_mnthly_hrs < 210):
        work_load = "average"
    elif (avg_mnthly_hrs >= 210) & (avg_mnthly_hrs < 252):
        work_load = "above_average"
    elif avg_mnthly_hrs >= 252:
        work_load = "workoholic"

    return work_load
```

```
hr_data['work_load'] = hr_data.average_montly_hours.apply(work_load_cat)

sns.countplot(x='work_load', hue='left', data=hr_data, order = ['low', 'average', 'above_average', 'workoholic']);
```

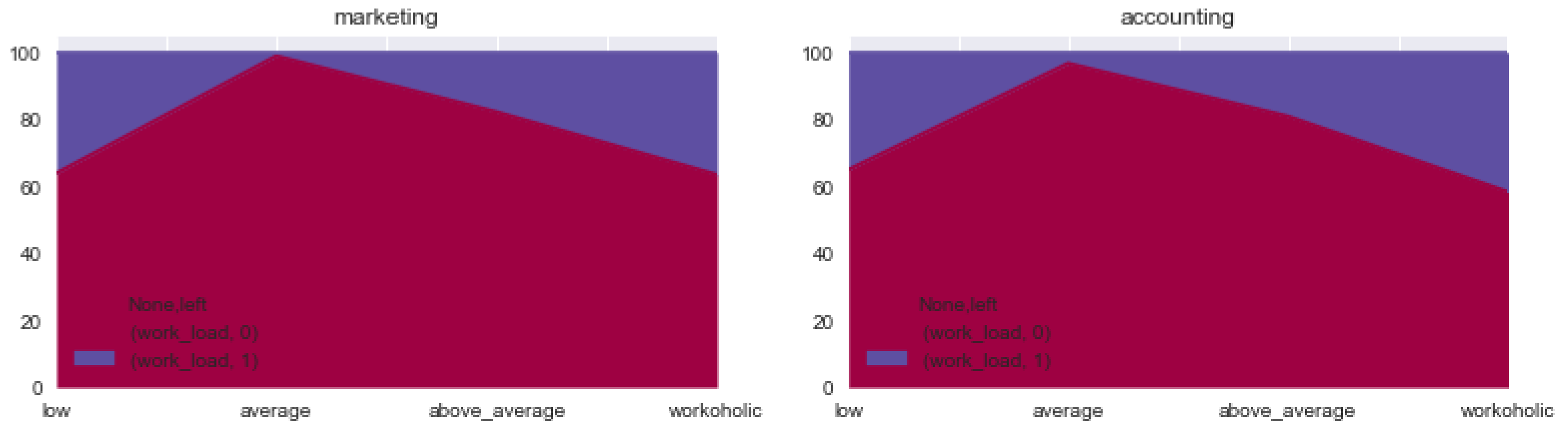


- The average zone (8~10 hours a day at work) have a very low record of leaving.

Data Understanding

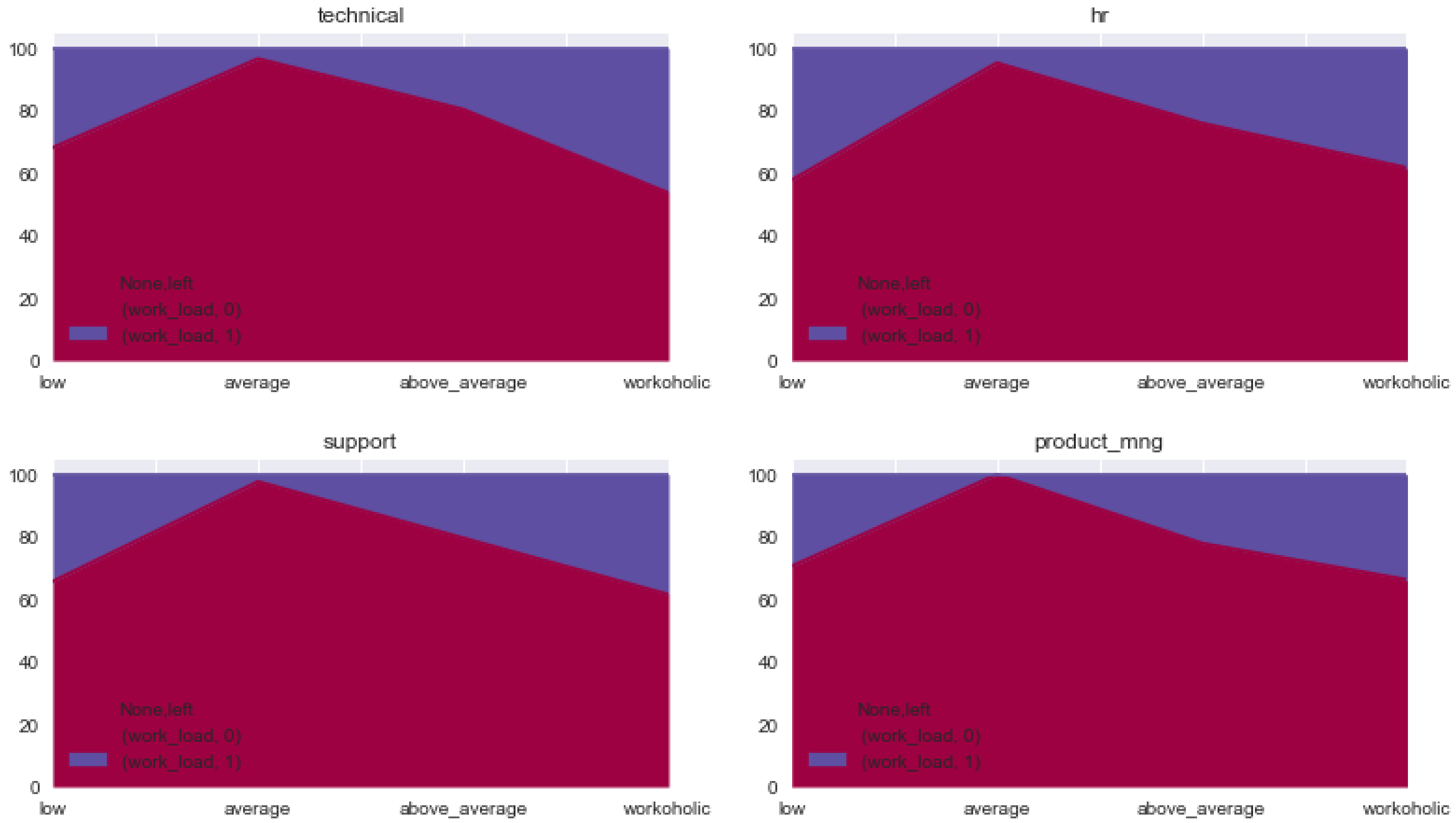
Graphical Analysis of workload (1/3)


```
#Normalised stacked
departments = list(set(hr_data.department.values))
number_of_departments = len(departments)

fig, axs = plt.subplots(nrows= int(number_of_departments/2), ncols=2, figsize=(13, 20))

for i in range(number_of_departments):
    current_dep = departments[i]

    ratio_df = 100*hr_data[hr_data.department == current_dep].groupby(['work_load', 'left']).agg(
        {'work_load': 'count'})/hr_data[hr_data.department == current_dep].groupby(['work_load']).agg(
        {'work_load': 'count'})
    ratio_df = ratio_df.reindex_axis(["low", "average", "above_average", "workoholic"], axis=0, level=0)
    #plot the department
    ratio_df.unstack().plot(kind='area',stacked=True, colormap= 'Spectral', ax=axs[int(i/2),i%2])
    axs[int(i/2),i%2].set_title(current_dep)
    axs[int(i/2),i%2].set_xlabel("")

axs[int(i/2),i%2].set_xlabel("work_load")
plt.subplots_adjust(hspace=0.3);
```

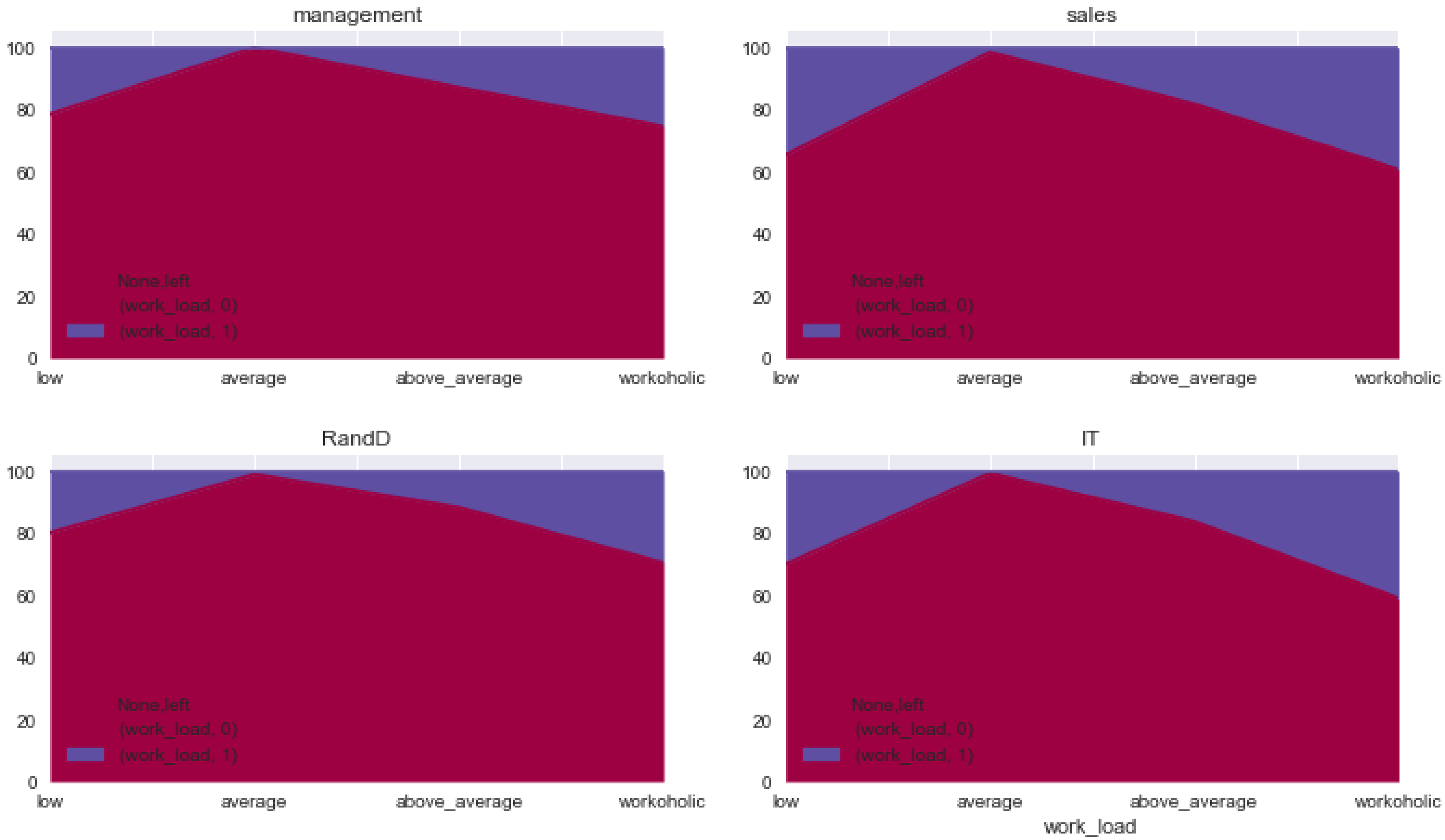


Data Understanding

Graphical Analysis of workload (2/3)



Data Understanding
Graphical Analysis of workload (3/3)



Data Understanding

Graphical Analysis of last_evaluation


```python
#Understanding how the Company Evaluates its Employees
#A function to bin last evaluation into one of 5 categories
def last_evaluation_cat(last_evaluation):
    evaluation = "unknown"
    if last_evaluation < 0.45:
        evaluation = "very_low"
    elif (last_evaluation >= 0.45) & (last_evaluation < 0.55):
        evaluation = "mediocre"
    elif (last_evaluation >= 0.55) & (last_evaluation < 0.8):
        evaluation = "average"
    elif (last_evaluation >= 0.8) & (last_evaluation < 0.9):
        evaluation = "very_good"
    elif last_evaluation >= 0.9:
        evaluation = "excellent"

    return evaluation
```

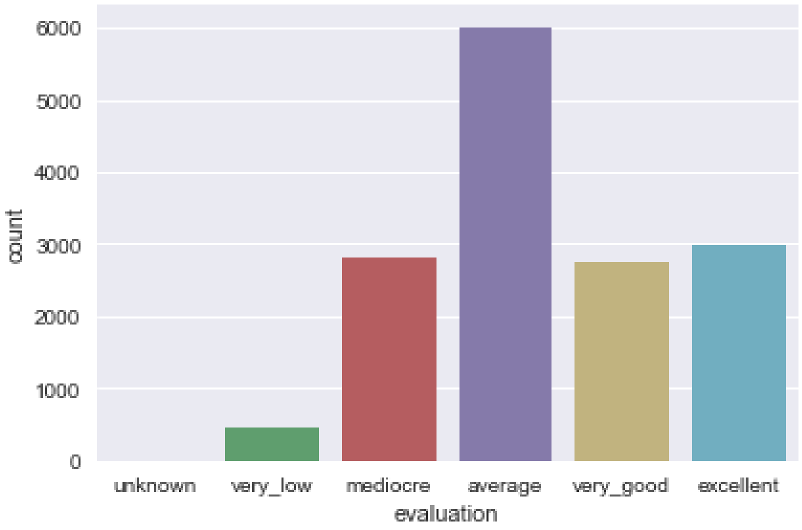
```python
hr_data['evaluation'] = hr_data.last_evaluation.apply(last_evaluation_cat)
```

Categorising the employees evaluation would yield better visualisations

Data Understanding

Evaluation Categories Across the Company

```
sns.countplot(x='evaluation',  data=hr_data, order = ["unknown", 'very_low', 'mediocre',
                                                     'average', 'very_good', 'excellent']);
```



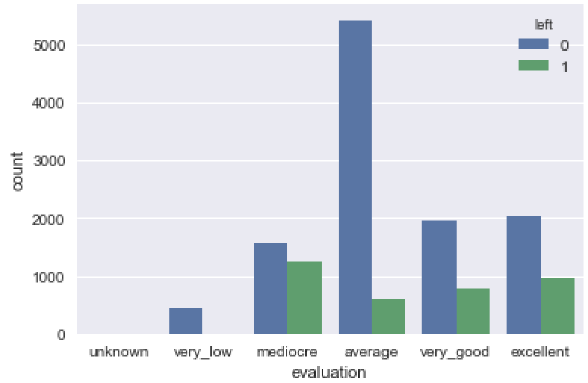
- Average is definitely the dominant, with mediocre, very good and excellent having close counts. Can there be a pattern if we visualise them in terms of who left?

Data Understanding

Graphical Analysis of workload (3/3)

```
#A function to bin the average monthly hours into the categories described above
def work_load_cat(avg_mnthly_hrs):
    work_load = "unknown"
    if avg_mnthly_hrs < 168:
        work_load = "low"
    elif (avg_mnthly_hrs >= 168) & (avg_mnthly_hrs < 210):
        work_load = "average"
    elif (avg_mnthly_hrs >= 210) & (avg_mnthly_hrs < 252):
        work_load = "above_average"
    elif avg_mnthly_hrs >= 252:
        work_load = "workoholic"

    return work_load
```

```
sns.countplot(x='evaluation',  hue = 'left', data=hr_data,
             order = ["unknown", 'very_low', 'mediocre', 'average', 'very_good', 'excellent']);
```



Findings:

1- The very low performance did not leave the company

2- Mediocre has a leaving rate almost as high as those who stayed

3- The average here is similar to that of monthly-time-spent, they have the lowest leaving rate.

4- Very good and excellent emplyees remain sort of similar.

Data Understanding

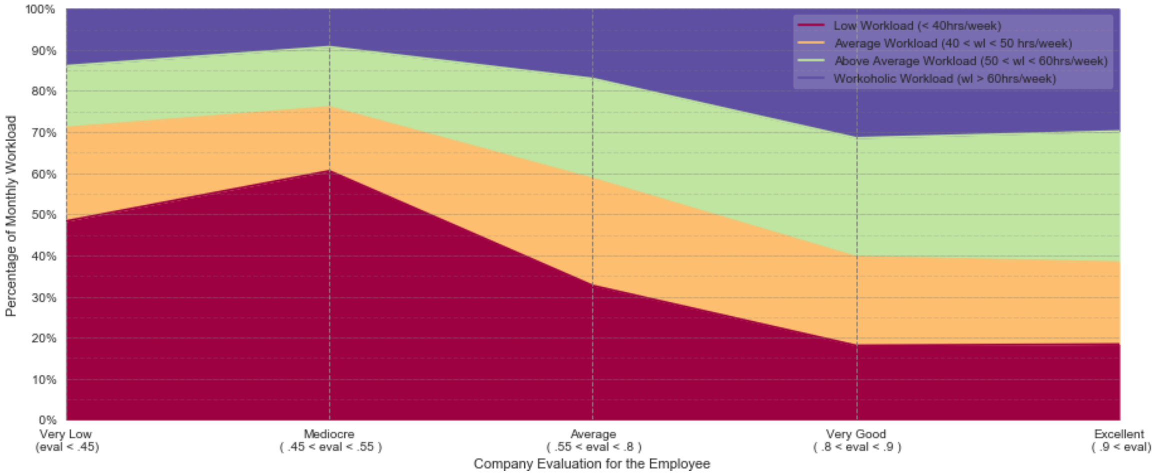
Further exploration


```
evaluation_index_order = ["unknown", 'very_low', 'mediocre', 'average', 'very_good', 'excellent']
evaluation_xticks = ['Very Low\n (eval < .45)', 'Mediocre\n ( .45 < eval < .55 )',
                     'Average\n ( .55 < eval < .8 )', 'Very Good\n ( .8 < eval < .9 )',
                     'Excellent\n ( .9 < eval)']
evaluation_x_label = "Company Evaluation for the Employee"
```

```
#A function to bin the average monthly hours into the categories described above
def work_load_cat(avg_mnthly_hrs):
    work_load = "unknown"
    if avg_mnthly_hrs < 168:
        work_load = "low"
    elif (avg_mnthly_hrs >= 168) & (avg_mnthly_hrs < 210):
        work_load = "average"
    elif (avg_mnthly_hrs >= 210) & (avg_mnthly_hrs < 252):
        work_load = "above_average"
    elif avg_mnthly_hrs >= 252:
        work_load = "workoholic"

    return work_load
```

Data Understanding
Performance in Terms of Working Hours


```
#Categories Makeup in Terms of Working Hours
employees_by_eval_and_workload = group_by_2_level_perc(hr_data,
                                                       'evaluation', 'work_load',
                                                       evaluation_index_order,
                                                       ['low','average','above_average', 'workoholic'])#Index Order

workload_legend = ['Low Workload (< 40hrs/week)', 'Average Workload (40 < wl < 50 hrs/week)',
                   'Above Average Workload (50 < wl < 60hrs/week)',
                   'Workoholic Workload (wl > 60hrs/week)']

#Plot the Graph
p=employees_by_eval_and_workload.unstack().plot(kind='area',stacked=True, colormap= 'Spectral',
                                                figsize=(15, 6), zorder=0)

customise_2lvl_perc_area_graph(p, workload_legend,
                               xtick_label=evaluation_xticks, x_label=evaluation_x_label,
                               y_label="Percentage of Monthly Workload")
```

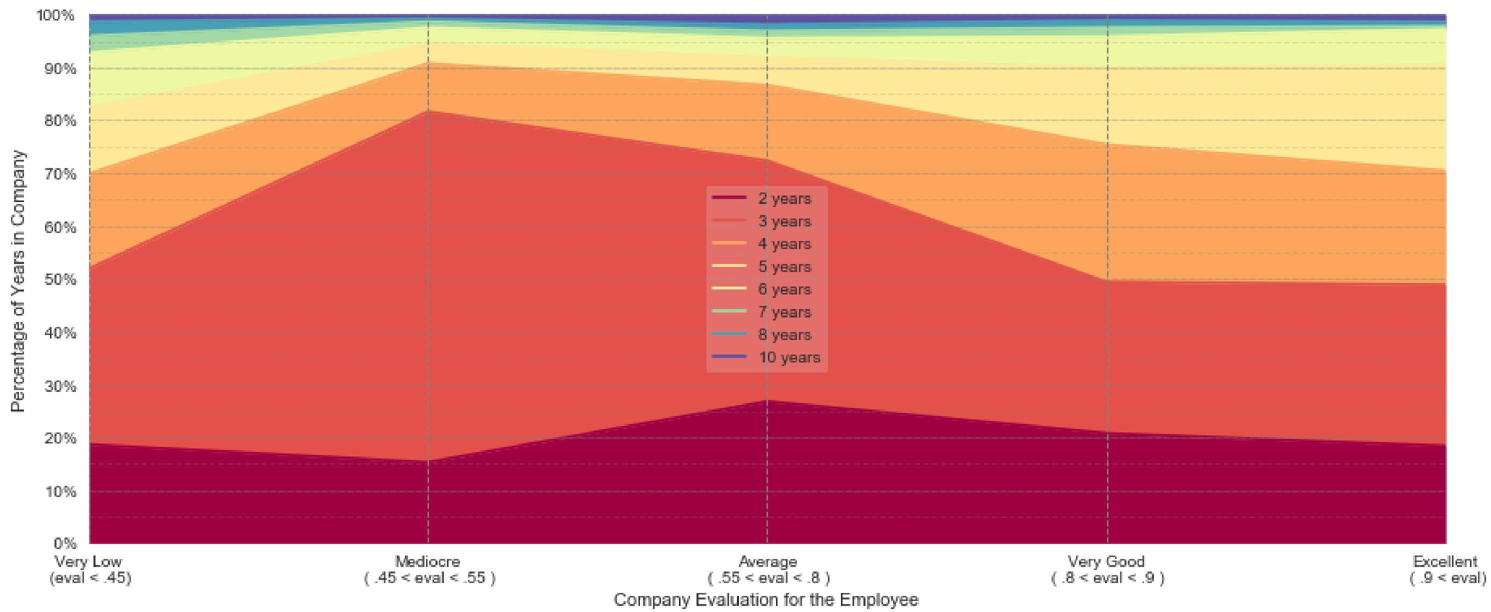


- It surprises, but it is not an impossibility to work less hours and still get excellent ratings, hopefully it is because they work smarter.

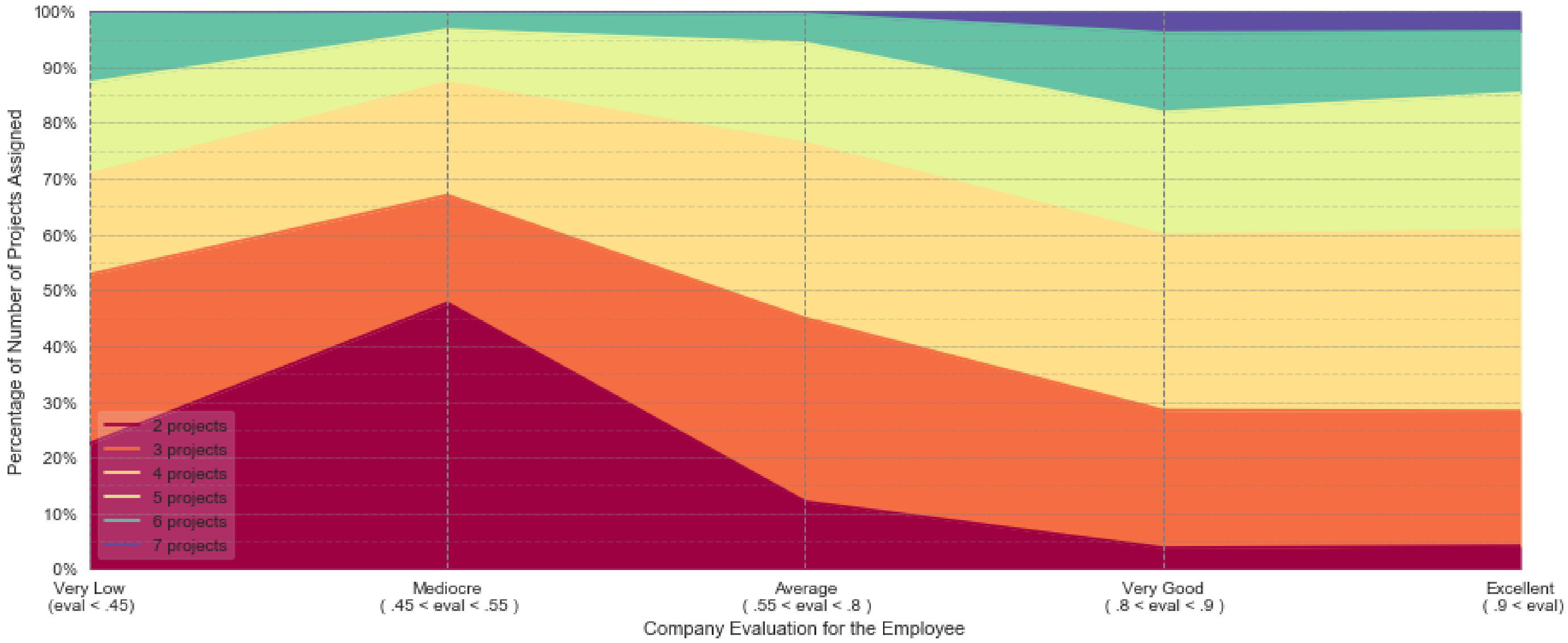
Data Understanding

Number of Years with the Company


```
#Number of Years with the Company
employees_by_eval_and_time_in_company_perc = group_by_2_level_perc(hr_data,
                                                                   'evaluation',
                                                                   'time_spend_company',
                                                                   evaluation_index_order)

#Plot the Graph
p=employees_by_eval_and_time_in_company_perc.unstack().plot(kind='area',
                                                            stacked=True,
                                                            colormap= 'Spectral',
                                                            figsize=(15, 6), zorder=0)

time_spent_legend = [str(x) + " years" for x in range(2,9)] + ['10 years']

customise_2lvl_perc_area_graph(p, time_spent_legend, xtick_label=evaluation_xticks,
                               x_label=evaluation_x_label, #Company Evaluation Graph
                               y_label="Percentage of Years in Company")
```



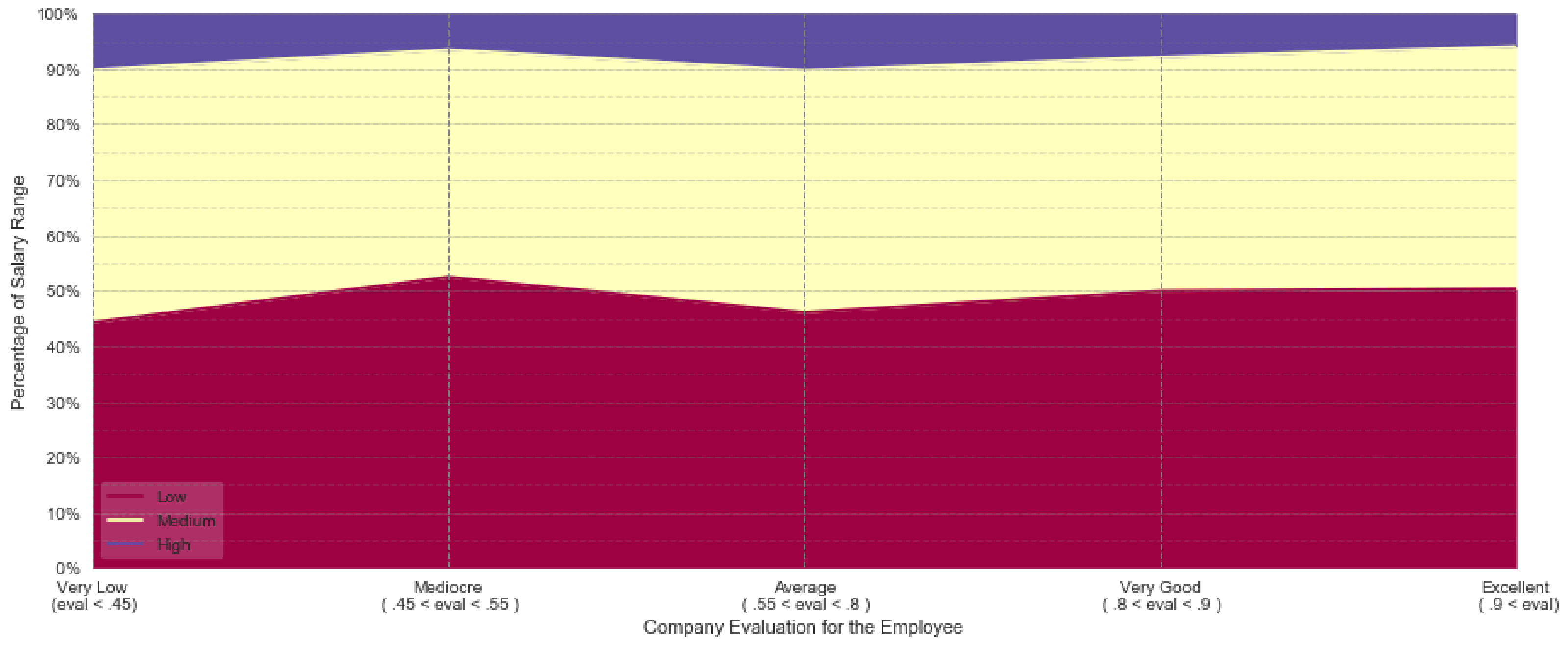
```
employees_by_eval_and_time_in_company_perc = group_by_2_level_perc(hr_data,
                                                                   'evaluation', 'number_project',
                                                                   evaluation_index_order)

#Plot the Graph
p=employees_by_eval_and_time_in_company_perc.unstack().plot(kind='area',stacked=True, colormap= 'Spectral',
                                                            figsize=(15, 6), zorder=0)

num_projects_legend = [str(x) + " projects" for x in range(2,8)]

customise_2lvl_perc_area_graph(p, num_projects_legend,
                               xtick_label=evaluation_xticks, x_label=evaluation_x_label,
                               y_label="Percentage of Number of Projects Assigned")
```



Data Understanding

Graphical Analysis of Salary


```
#salary
employees_by_eval_and_salary_perc = group_by_2_level_perc(hr_data,
                                                          'evaluation', 'salary',
                                                          evaluation_index_order,
                                                          ['low', 'medium', 'high'])

#Plot the Graph
p=employees_by_eval_and_salary_perc.unstack().plot(kind='area',stacked=True, colormap= 'Spectral',
                                                   figsize=(15, 6), zorder=0)

num_projects_legend = ['Low', 'Medium', 'High']

customise_2lvl_perc_area_graph(p, num_projects_legend,
                               xtick_label=evaluation_xticks, x_label=evaluation_x_label,
                               y_label="Percentage of Salary Range")
```



```
#satisfaction level
#Create a satisfaction categories
#Arbitrary boundaries:
# < 4.5 low
# 4.5 < < 7.5 medium
# 7.5 < high
def rank_satisfaction(employee):
    level = "unknown"
    if employee.satisfaction_level < 0.45:
        level='low'
    elif employee.satisfaction_level < 0.75:
        level = 'medium'
    else:
        level = 'high'
    return level
```

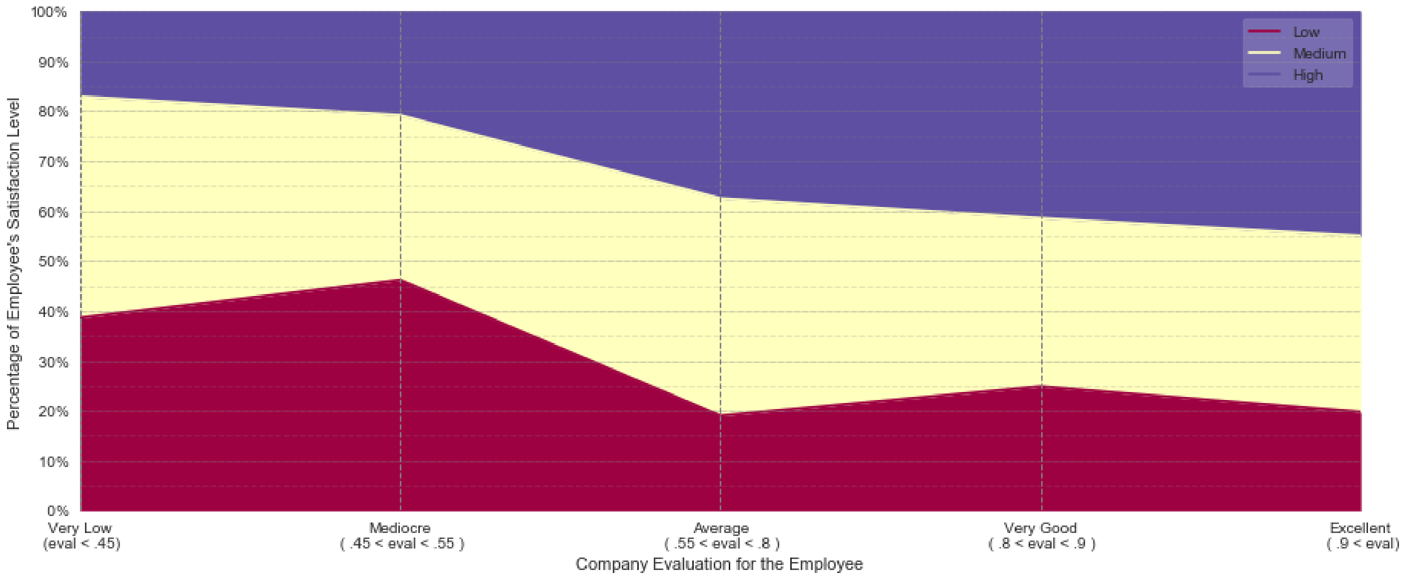
```
hr_data['satisfaction'] = hr_data.apply(rank_satisfaction, axis=1)
```

- With the Definition of limitations for employee satisfaction we can visualize them and reinforce it

Data Understanding

Graphical Analysis of satisfaction level


```
employees_by_eval_and_satisfaction_perc = group_by_2_level_perc(hr_data,
                                                                 'evaluation',
                                                                 'satisfaction',
                                                                 evaluation_index_order,
                                                                 ['low', 'medium', 'high'])

#Plot the Graph
p=employees_by_eval_and_satisfaction_perc.unstack().plot(kind='area',stacked=True,
                                                          colormap= 'Spectral', figsize=(15, 6),
                                                          zorder=0)


satisfaction_lvl_legend = ['Low', 'Medium', 'High']

customise_2lvl_perc_area_graph(p, satisfaction_lvl_legend,
                               xtick_label=evaluation_xticks, x_label=evaluation_x_label,
                               y_label="Percentage of Employee's Satisfaction Level")
```



Data Preparation

Handling missing values (mv) 1/3

- For further analytics e.g. making predictions with the help of a Support Vector Machine we need to survey if the data are “good”
- check if data contains missing values:

```
if(not hr_data.isnull().values.any()):  #Checking for NaN-values
    print('QC (Y): Dataset does not contain missing values')
else:
    print('QC (N): Dataset contains missing values')
```

```
QC (Y): Dataset does not contain missing values
```

- The original HR Analytics data set fortunately has no missing values. For demonstrating how to face this issue let us randomly create some mv‘s and see strategies

```
#for showing how to handle missing values(mv), we randomly create some NaN-values inside the dataset
import random
ix = [(row, col) for row in range(hr_data_copy.shape[0]) for col in range(hr_data_copy.shape[1])]
for row, col in random.sample(ix, int(round(.1*len(ix)))):   #for a 10% random sample in the sequenz ix-array
       hr_data_copy.iloc[row, col] = np.nan                  #entries replaced by NaN-values
```

```
hr_data_copy.head() #showing the first five entries again with NaN-entries
```

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	left	promotion_l
0	0.38	0.53	2.0	157.0	3.0	0.0	1.0	0.0
1	0.80	0.86	5.0	NaN	6.0	0.0	1.0	0.0
2	0.11	0.88	7.0	272.0	4.0	0.0	1.0	0.0
3	0.72	0.87	5.0	223.0	5.0	0.0	1.0	0.0
4	0.37	0.52	NaN	159.0	3.0	0.0	1.0	0.0

```
hr_data_copy.describe() #just to show the mv-effect, now less entries, because of mv
```

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident	l
count	13481.000000	13510.000000	13534.000000	13541.000000	13422.000000	13509.000000	13488.0000

Data Preparation

Handling missing values (mv) 3/3


```
hr_no_missing_f=hr_data_copy.fillna(hr_data_copy.iloc[:10].median())
hr_no_missing_f.describe()
```

	satisfaction_level	last_evaluation	number_project	average_montly_hours	ti
count	14999.000000	14999.000000	14999.000000	14999.000000	1

```
#3.way
hr_no_missing_d = hr_data_copy.dropna()
```

```
hr_no_missing_d.describe()
```

	satisfaction_level	last_evaluation	nu
count	5244.000000	5244.000000	52

- There are different possibilities to manage missing values (displayed as NaN-entries in python): **ignore**, **fill** the gaps or **reject** non-conforming entries
- **Filling** NaN-values is possible with varying imputing strategies e.g. mean, median, most_frequent
- **Dropping** missing values

The exploration notebook can be found:
https://github.com/BIRatTUDD/HR-AnalyticsDS/blob/master/HRA_EDA.ipynb