Regression analysis to predict employee satisfaction at UBS - Karl Merisalu

Background: One way of measuring employee satisfaction at UBS is conducting employee surveys. Employee surveys are run on a regular interval and as part of this process every employee is asked to rate their attitude towards ca 30 topics, including how satisfied they are with their job. These topics range from very specific ones, such as: performance management, happines with senior management, how happy employees are with ongoing innovation initiatives and etc., to somewhat broader topics like: employee engagement/job satisfaction.

Problem situation: Some of these areas are relatively easy to interpret (eg cost awareness), but some are harder (employee engagement) as there may be several and different factors affecting each category. Employee engagement is sometimes considered as an overall metric and a proxy for job satisfaction. It combines many other factors and as such often receives the most attention by senior management. As a result, a lot of the initiatives in the company are targeted to improve employee engagement. However, since it is difficult to interpret what exactly causes higher or lower employee engagement levels in different counties or teams, it is hard to know which initiatives will work and which won't.

Task: In order to shed more light onto which factors affect employee engagement / job satisfaction UBS, I will counduct a multivariate linear regression analysis on our internal employee survey data. I will be able to access some of this data because I work with employee surveys directly together with our HR department who conduct the survey. My role in this process is to contribute to building a presentation for senior management, which includes a summary analysis and recommendations for improvements.

This is analysis is important, because the results will help me to propose actions to be taken to improve employee engagement in the future. Not only will I be able to say which areas I should focus on, but I would also be able to take into account cost considerations in order to propose most cost effective measures.

More specifically, the analysis will help me to identify which factors are sigfinicant in affecting employee engagement and job satisfaction, and how strongly each factor affects them. I will be using Python ecosystem for this analysis because it is good in dealing with large quantities of data and is also relatively easily replicable for new datasets once the analytics code has been developed.

Limitations: Full analysis will be provided using a similar publicly available dataset from Kaggle to simulate the analysis in my company and show the steps I would go through as if I would be conducting this analysis at UBS. However, since our company internal data is confidential and cannot be revealed to 3rd parties, part of the analysis, which will use internal data, will be edited to cover sensitive details.

Summary of actions: To summarize, this is a regression project, where I'm going to predict employee satisfaction rating based on a number of features. Due to the confidential nature of internal data at my employer, I will be first using a similar public dataset to showcase a full proxy analysis and steps taken. Then I will run an edited (and shorter) analysis on limited UBS data that I have access to. This project will be useful for my role and my employer, because replicating it at UBS with our data will help us to identify which areas the company should focus on for the greatest impact of increasing employee engagement.

Following is a step by step guide on how I'm going to accomplish it:

1) Import required libraries *(enables me to use pre-developed complex functions on the data)*
2) Import the data *(proxy data)*
3) Data exploration *(to understand what I'm dealing and if there are any shortcomings/flaws)*
4) Cleaning the dataset *(preparing data for further analysis)*
5) Employee satisfaction considerations *(deeper look into the dependent/main variable data)*
6) Data preparation for regression analysis *(making data understandable for machine learning algorithms)*
7) Multivariate linear regression *(building the regression model)*
7.1) Multivariate regression model in STATSMODEL *(alternative way to build the regression model)*
8) Multivariate regression with internal data *(example on limited data)*
9) Summary Conclusion *(findings and concluding the analysis)*

1) Importing libraries

First I import required libraries which will help us to do data manipulation and run regression analysis algorithms later on

In [1]:
```
# importing required libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import seaborn as sns
import statsmodels.formula.api as smf

# Suppressing depreciation warnings
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

2) Importing the dataset

Data: For this analysis I use data from Kaggle.com containing 15k employee profiles together with their job satisfaction level. This will be the basis for building the regression model to determine which factors and how they affect job satisfaction levels. The dataset consists of numerical and categorical data as do most employee surveys.

In order to replicate this analysis the data used can be downloaded from: https://www.kaggle.com/jacksonchou/hr-data-for-analytics (https://www.kaggle.com/jacksonchou/hr-data-for-analytics).

I'm using this public dataset to be compliant with UBS data sharing policies, because our internal data is confidential. Given that this HR dataset is relatively similar to our employee survey data (100pt rating scale for dependent variable, including categorical and numberical variables), it is as close as it gets to what I have internally in the company.

First, I import the dataset, then take an initial look at it:

In [2]:
```
# importing dataset
dataset = pd.read_csv('HR_comma_sep.csv')

# making sure dataset is a dataframe
dataset = pd.DataFrame(dataset)

# looking at the header of the dataset
dataset.head()
```

Out[2]:

	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

Comment: as you can see from above, the dataset contains several variables on each employee (each row equals to 1 employee). At this point I'm interested in all variables, so I will not drop any columns.

At UBS I would keep the following variables: 1) answers to employee survey questions and 2) employee location (I would probably also keep employee rank). Other employee profile variables like for example employee phone number, email would not be relevant for this analysis and as such would be dropped.

3) Data exploration

Let's take an initial look at the dataset to see if I have any missing valuables.

In [3]:
```
dataset.info()
```

```
<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
```

Comment: I can see from above that I have 14 999 datapoints for each 9 variable. This is good news, as it suggests that I don't have missing values in the dataset. I have decimal data (variables 0-1), integers (variables 2-7) and categorical data (variables 8-9)

When putting it into context at UBS, our employee survey is filled in by 50k+ employees and employee engagement rating at employee surveys would be equivalent to "employee job satisfaction" rate in our public dataset. Coincidentally, employee engagement is measured on a 100pt scale (as is job satisfaction in our current dataset) and similarly we also have employee profile data available. At UBS I'd want to find out how different characteristics (as mentioned above: performance management, happines with senior management, how happy employees are with ongoing innovation initiatives, work location, rank and etc.) affect employee engagement rating.

In the case of UBS data, there are likely some questions that employees have not answered in the survey. This would create missing values in the dataset. The code above would help to identify the amount of these missing values.

To drop rows with missing values I could use the following code:
dataset = dataset.dropna()
dataset.head()

Next, let's take a closer look at the value distribution of the dataset

In [4]:
```
dataset.describe()
```

Out[4]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_acci
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.00
mean	0.612834	0.716102	3.803054	201.050337	3.498233	0.14
std	0.248631	0.171169	1.232592	49.943099	1.460136	0.35
min	0.090000	0.360000	2.000000	96.000000	2.000000	0.00
25%	0.440000	0.560000	3.000000	156.000000	3.000000	0.00
50%	0.640000	0.720000	4.000000	200.000000	3.000000	0.00
75%	0.820000	0.870000	5.000000	245.000000	4.000000	0.00
max	1.000000	1.000000	7.000000	310.000000	10.000000	1.00

Comment: I can see that satisfaction level ratings are between 0.09 and 1 points (0 meaning 0% satisfied and 1 meaning 100% satisfied), but 50% of responses are between a narrow range in the middle from 44% to 82% satisfied.

Interestingly work accident, left (employees who left) and promotion last 5 years seems to have very few "hits" with predominantly showing 0 or none as a response.

For context, industry employee survey engagement results are generally in the range 50-100 points, which is aligned with our current results

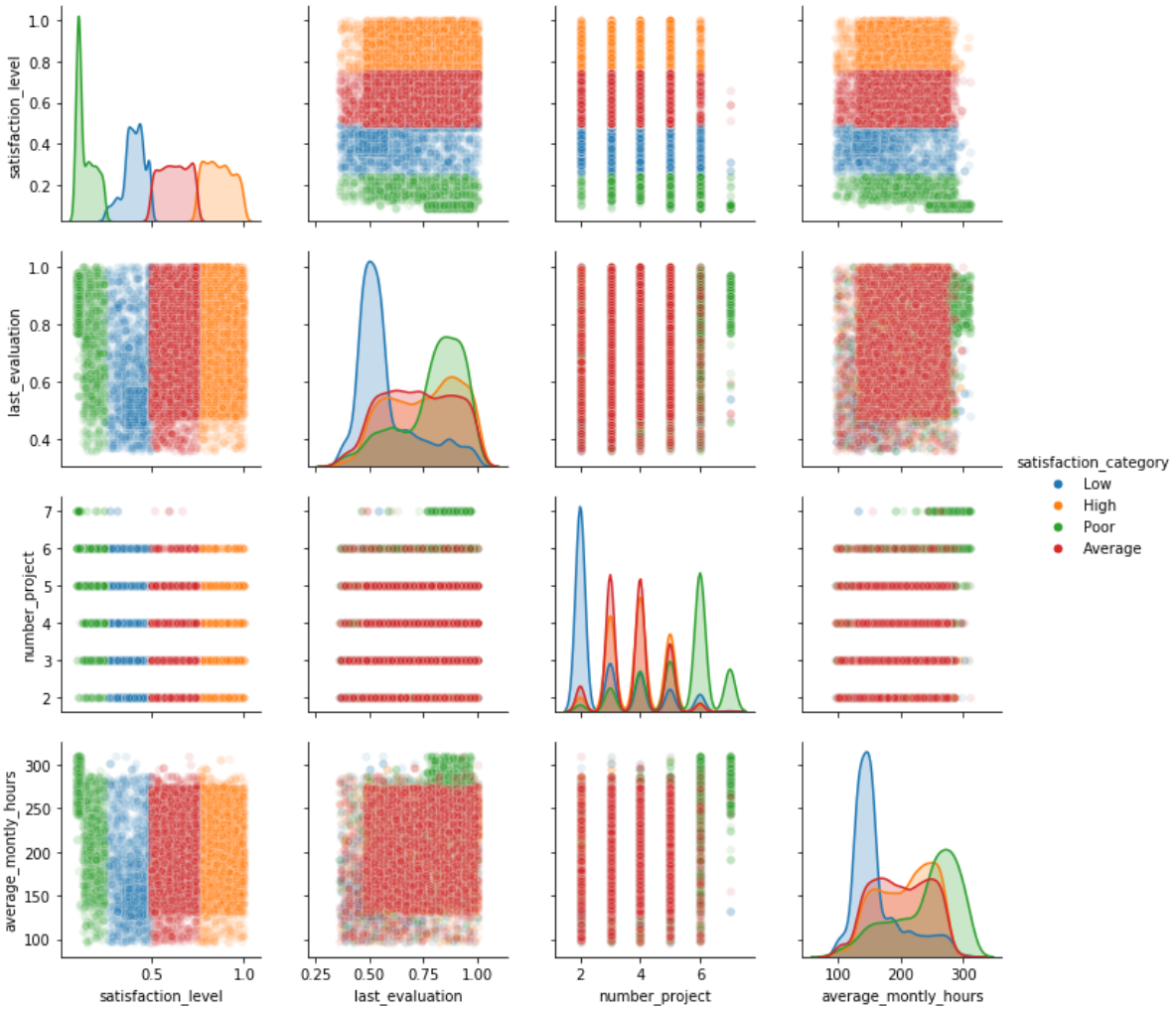
Next, I'm checking if I can already identify some patterns (data mining) in the data by using a quick seaborn pairplot algorithm, which visualises all pairs of selected variables on charts. In order to have more meaningful results I will only inlcude non-categorical variables and I will create a new column by categorising satiscation levels into 4 categories (quadrants) according to the rating:
1) <0.25
2) 0.25-0.5
3) 0.5-0.75
4) >0.75

In [5]:
```
# Creating a new satisfaction category column

def cat(dataset):
    if dataset['satisfaction_level'] < 0.25:
        return 'Poor'
    elif 0.25 <= dataset['satisfaction_level'] < 0.5:
        return 'Low'
    elif 0.5 <= dataset['satisfaction_level'] < 0.75:
        return 'Average'
    else:
        return 'High'

dataset['satisfaction_category'] = dataset.apply(cat, axis=1)

dataset.head()
```

Out[5]:

	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 [6]:
```
# Plotting the categories against all non-categorical variables for exploration/data mining
dataset_test = dataset[['satisfaction_level','last_evaluation', 'number_project','average_m
ontly_hours', 'satisfaction_category']]

import seaborn as sns
g = sns.pairplot(dataset_test, hue = "satisfaction_category", plot_kws={'alpha': 0.1})

for lh in g._legend.legendHandles:
    lh.set_alpha(1)
    lh._sizes = [50]
```



Comment: Above charts provide me some very interesting initial insights:
1) Low and poorly satisfied employees tend to be working on very little number of projects or on very many
2) There seems to be a significant cluster of employees, who in the last evaluation rated their satisfaction rather highly, but now are categorised as poorly satisfied (noted in green)

Predominantly, people seem to have rated their satisfaction level above 0.5, so it may be beneficial to reclassify category thresholds in further analysis for additional insights. To do this, I'm going to conduct a quick clustering analysis next.

Clustering Analysis

Clustering analysis is a type of unsupervised machine learning method, which enables us to identify groups data points with similar features.

I'm going to use k-means clustering algorithm, which (shortly) works like this:
1) I select a predetermined number of centroids (definition: average position of all the points of an object) to be randomly placed on a chart among all datapoints. Let's call these centroids "k" or "k-points".
2) All datapoints are then associated with the "k-point" that they're closest distance to.
3) The algorithm calculates a mean location for datapoints associated with their closest "k-point". "K-point" location is then updated with this mean value.
4) We go back to step #2, and repeat the process of adjusting "k-point" for a several times until it reaches its equilibrium location.
5) We have now identified centers of clusters, which are "k-points" and clusters themselves, which are the closest points surrounding these "k-points", respectively.

To start with, I import required libraries and begin modelling with with 4 clusters. I'll start with 2 dimensional clustering and will then be adding a 3rd dimension in order to complement pairplot charts above. I'll use non-categorical ('satisfaction_level', 'last_evaluation', 'average_montly_hours') datapoints to have a more meaninful analysis.

In [7]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import scale
```

Next, I create a new dataset with only required columns:

In [8]:
```
# Creating a new dataset with only required columns
dataset_cluster = dataset[['satisfaction_level', 'last_evaluation', 'average_montly_hours'
]]
```

I run the kmeans function with 4 clusters and store the result in our variable, 'model'. I also fit the model to data:

In [9]:
```
# I run the kmeans function and store the result in our variable, 'model'
model = KMeans(n_clusters=4)

# Now I fit our kmeans model to our data.
model.fit(scale(dataset_cluster))
```

Out[9]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

Now, I apply labels to each cluster

In [10]:
```
# Next we apply labels to each cluster for better tracking and creation of summary statistics
dataset_cluster['cluster'] = model.labels_
dataset_cluster.head()
```

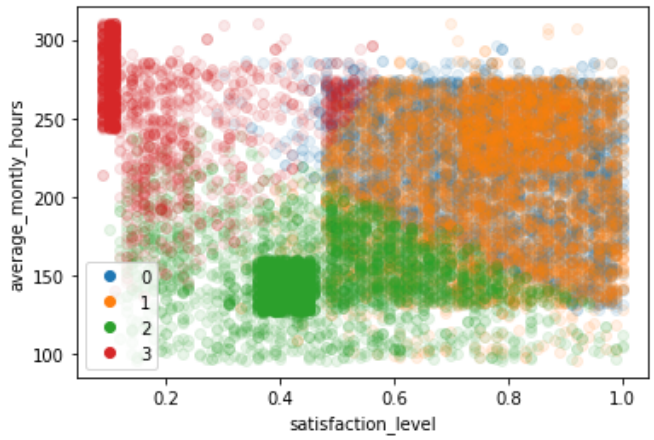
```
\\ubsprod.msad.ubs.net\UserData\MERISALK\Home\Miniconda3_64bit\envs\pyt367\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

Out[10]:

	satisfaction_level	last_evaluation	average_montly_hours	cluster
0	0.38	0.53	157	2
1	0.80	0.86	262	1
2	0.11	0.88	272	3
3	0.72	0.87	223	1
4	0.37	0.52	159	2

Next, I group the dataset by clusters and plot clusters on the chart in 2 dimensons

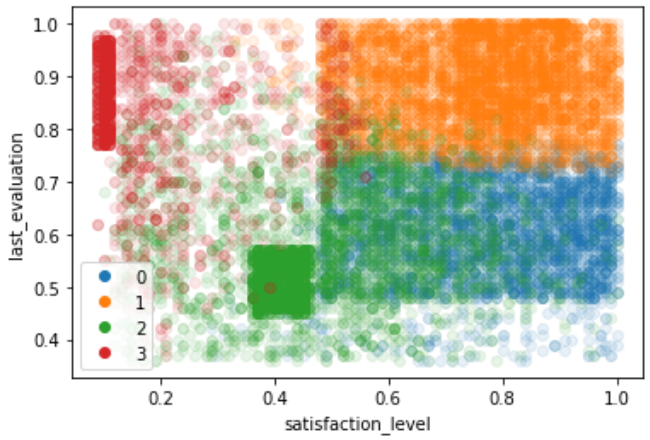
In [11]:
```
groups = dataset_cluster.groupby('cluster')

# Plotting the clusters!
fig, ax = plt.subplots()
for name, group in groups:
    ax.plot(group.satisfaction_level, group.average_montly_hours, marker='o', linestyle='', label=name, alpha= 0.1)
    # Add axis labels appropriately
plt.xlabel('satisfaction_level')
plt.ylabel('average_montly_hours')
leg = ax.legend()
for lh in leg.legendHandles:
    lh._legmarker.set_alpha(1)

plt.show()
```



Comment: interesting! There seems to be at least 2 very strong clusters (cluster 3 - red and cluster 2 - green). However, the clusters are formed taking into account 3 features, so see how it will look like on a 3D chart.

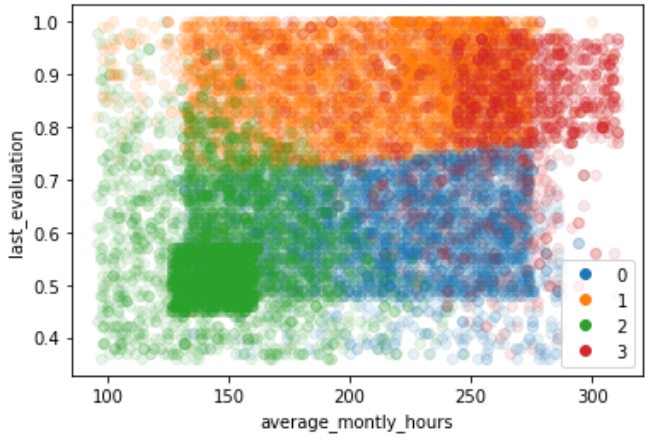
But first, I'm going to plot 2 remaining possible 2D charts (between these 3 variables) for completeness:

In [12]:
```
# Plotting the clusters #2!
fig, ax = plt.subplots()
for name, group in groups:
    ax.plot(group.satisfaction_level, group.last_evaluation, marker='o', linestyle='', label=name, alpha= 0.1)
    # Add axis labels appropriately
plt.xlabel('satisfaction_level')
plt.ylabel('last_evaluation')
leg = ax.legend()
for lh in leg.legendHandles:
    lh._legmarker.set_alpha(1)

plt.show()
```



Angle 2: red and green clusters remain intact

In [13]:
```
# Plotting the clusters #2!
fig, ax = plt.subplots()
for name, group in groups:
    ax.plot(group.average_montly_hours, group.last_evaluation, marker='o', linestyle='', label=name, alpha= 0.1)
    # Add axis labels appropriately
plt.xlabel('average_montly_hours')
plt.ylabel('last_evaluation')
leg = ax.legend()
for lh in leg.legendHandles:
    lh._legmarker.set_alpha(1)

plt.show()
```



Angle 3: We can still identify red and green clusters. Perhaps red looks a bit more faded from this angle...

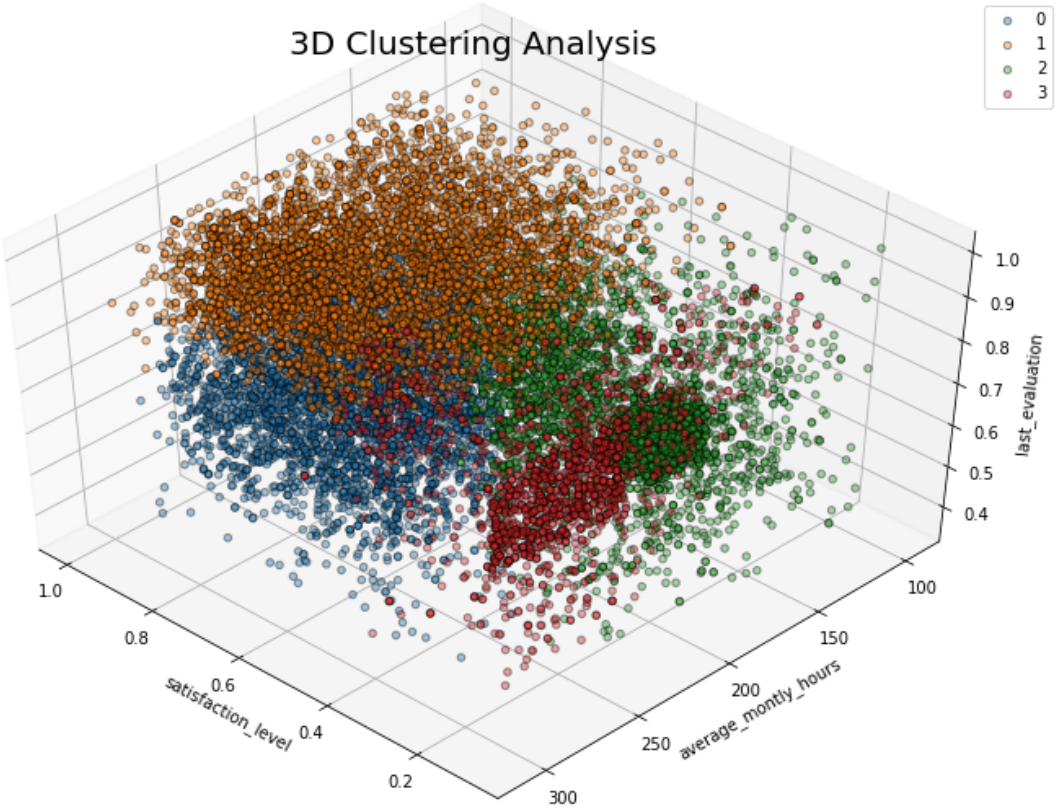
Before moving to 3D plotting, I also take a quick look at how the dataset looks like

In [14]:
```
print(dataset_cluster)
```

```
       satisfaction_level  last_evaluation  average_montly_hours  cluster
0                    0.38             0.53                   157        2
1                    0.80             0.86                   262        1
2                    0.11             0.88                   272        3
3                    0.72             0.87                   223        1
4                    0.37             0.52                   159        2
...                   ...              ...                   ...      ...
14994                0.40             0.57                   151        2
14995                0.37             0.48                   160        2
14996                0.37             0.53                   143        2
14997                0.11             0.96                   280        3
14998                0.37             0.52                   158        2

[14999 rows x 4 columns]
```

Before plotting a 3D chart, I'm importing new required libraries. Then I adjust plot parameters and plot:

In [15]:
```
# importing required libraries

from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets

# Plotting the clusters!
fig = plt.figure(figsize=(10, 7))
ax = Axes3D(fig, elev=48, azim=134)

for name, group in groups:
    ax.scatter(group.satisfaction_level, group.average_montly_hours, group.last_evaluation,
edgecolor = 'k', label=name, alpha= 0.4)

# Add axis labels appropriately

ax.set_xlabel('satisfaction_level')
ax.set_ylabel('average_montly_hours')
ax.set_zlabel('last_evaluation')
ax.set_title('3D Clustering Analysis', fontsize=20)


ax.legend(fontsize = 10)

plt.show()
```



Comment: I can see from above that cluster number 3 (red) represents people who previously had a rather high evaluation, but now for some reason rate their job satisfaction close to 0. These people are also doing a lot of average monthly hours.

Cluster number 2 (green) also seem to also have experienced a decline in job satisfaction level from last evaluation, but their average monthly hours seem to be in the middle of the pack.

This clustering analysis gives a quick initial look into the data. Let's continue with leaning the dataset for further analysis.

4) Cleaning the dataset

As I identified previously, I do not have missing values in the dataset. So there is no need to drop rows.

In the context of UBS, however, this point in the analysis is important as it is unlikely that I can always have that clean datasets. Should it be necessary to clean the dataset at my employer I will use the following code as already mentioned above: 1) dataset = dataset.dropna() 2) dataset.describe()

I don't need to drop variables, but what I can do is to change the satisfaction_level and last_evaluation data into a 100pt system for an easier read. I can do this by multiplying the 2 columns by 100:

In [16]:
```
dataset["satisfaction_level"] = 100 * dataset["satisfaction_level"]
dataset["last_evaluation"] = 100 * dataset["last_evaluation"]
dataset.head()
```

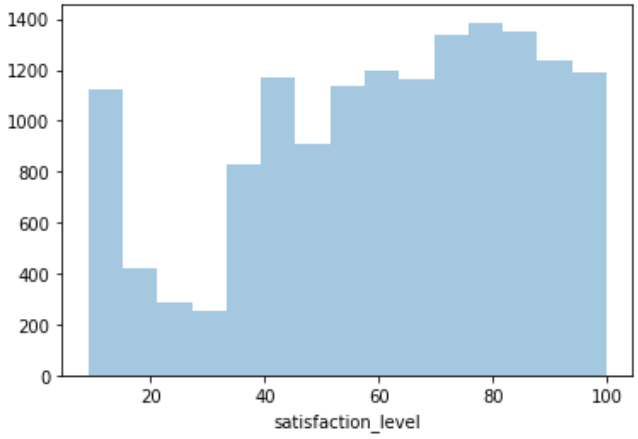
Out[16]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident
0	38.0	53.0	2	157	3	0
1	80.0	86.0	5	262	6	0
2	11.0	88.0	7	272	4	0
3	72.0	87.0	5	223	5	0
4	37.0	52.0	2	159	3	0

5) Employee satisfaction considerations

To get a more detailed view how points are distributed across the whole dataset I should plot values on a histogram:

In [17]:
```
sns.distplot(dataset['satisfaction_level'], kde=False, bins= 15)
plt.show()
```



Comment: It looks like I have a more complex probability density function / histogram than just a normal distribution (bell shape), which seems to be skewed towards high satisfaction and low satisfaction with high frequency. I can not tell from this that job satisfaction ratings are statistically independent and indentically distributed, but I will carry on. In some cases this analysis would show if I have extreme outliers, which I would then consider removing as they would distort the results disproportionately, but in this case I don't seem to have extreme outliers.

But I wonder is the overall satisfaction distribution represented similarly at department level? Let's have a quick look:

In [18]:
```
sns.pairplot(dataset, vars=["satisfaction_level"], hue='sales')
plt.show()
```

Comment: satisfaction levels look quite similar across departments, however, this chart (despite being convenient and quick) is clearly too small. Let's create satisfaction datasets manually and plot them on a larger chart next to get a better insight.

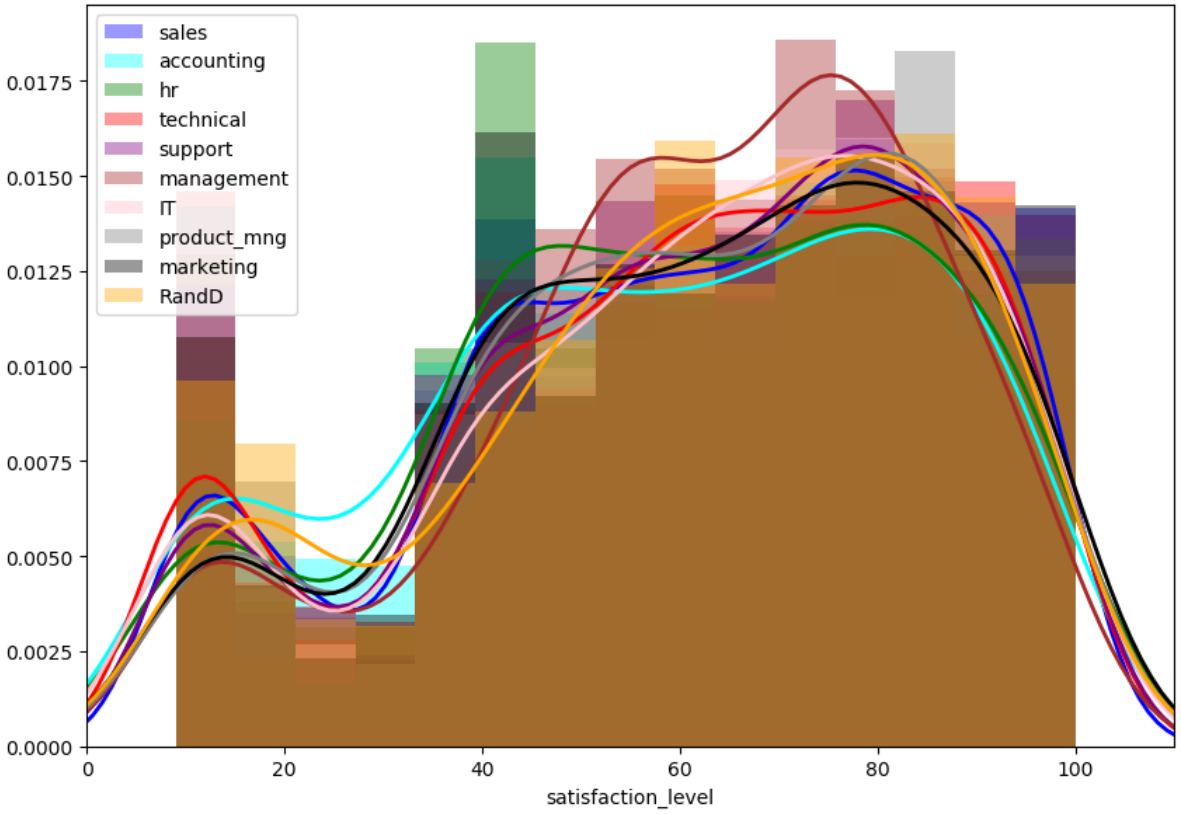
Slicing dataset by department next and assigning them to new variables x1-x10:

In [19]:
```
x1 = dataset[dataset['sales']=='sales']
x2 = dataset[dataset['sales']=='accounting']
x3 = dataset[dataset['sales']=='hr']
x4 = dataset[dataset['sales']=='technical']
x5 = dataset[dataset['sales']=='support']
x6 = dataset[dataset['sales']=='management']
x7 = dataset[dataset['sales']=='IT']
x8 = dataset[dataset['sales']=='product_mng']
x9 = dataset[dataset['sales']=='marketing']
x10 = dataset[dataset['sales']=='RandD']
```

In order for the following plot to be comparable to the overall satisfactions above, I keep the number of coulmns at 15, but I also keep the density plot on *(not like kde=False above)*. I do it, as you will see next, to make it easier to compare 10 departments being plotted on top of each other.

kwargs below enable us to modify all 10 plots below at once:
linewidth - to modify the thickness of the density line
alpha - to modify the transparency of the columns
bins - to modify number of columns

In [20]:
```
kwargs = dict(hist_kws={'alpha':.4}, kde_kws={'linewidth':2}, bins=15)

plt.figure(figsize=(10,7), dpi= 100)
sns.distplot(x1['satisfaction_level'], color="blue", label="sales", **kwargs)
sns.distplot(x2['satisfaction_level'], color="cyan", label="accounting", **kwargs)
sns.distplot(x3['satisfaction_level'], color="green", label="hr", **kwargs)
sns.distplot(x4['satisfaction_level'], color="red", label="technical", **kwargs)
sns.distplot(x5['satisfaction_level'], color="purple", label="support", **kwargs)
sns.distplot(x6['satisfaction_level'], color="brown", label="management", **kwargs)
sns.distplot(x7['satisfaction_level'], color="pink", label="IT", **kwargs)
sns.distplot(x8['satisfaction_level'], color="gray", label="product_mng", **kwargs)
sns.distplot(x9['satisfaction_level'], color="black", label="marketing", **kwargs)
sns.distplot(x10['satisfaction_level'], color="orange", label="RandD", **kwargs)
plt.xlim(0,110) # Limiting the x-axis
plt.legend();
```



Comment: I can see that departments have roughly similar views on satisfaction levels. Perhaps I can say that management department has more of higher ratings and accounting department has more of lower ratings than other departments.

6) Data preparation for regression analysis

Getting dummy variables for countries: I need to convert our categorical sales and salary variables to dummy variables in order to make them machine understandable (dummy variables are variables that equal 1 when the condition is present and equal 0 when the condition is not present). I don't need to adjust other variables, because it is understanable the same way for each wine.

As an example, I already have dummy variables in the dataset: work_accident and left. If these = 1, it means the condition was present for a certain employee.

After getting dummy variables I display the new dataset below

The same step would apply to my employee survey dataset at UBS

In [21]:
```
# creating a new dataset with dummy variables
dum_dataset = pd.get_dummies(dataset)
dum_dataset.head()
```

Out[21]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_accident
0	38.0	53.0	2	157	3	0
1	80.0	86.0	5	262	6	0
2	11.0	88.0	7	272	4	0
3	72.0	87.0	5	223	5	0
4	37.0	52.0	2	159	3	0

5 rows × 25 columns

Comment: Let's take a look how the new dataset looks like next.

In [22]:
```
dum_dataset.describe()
```

Out[22]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Work_acci
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000	14999.00
mean	61.283352	71.610174	3.803054	201.050337	3.498233	0.14
std	24.863065	17.116911	1.232592	49.943099	1.460136	0.35
min	9.000000	36.000000	2.000000	96.000000	2.000000	0.00
25%	44.000000	56.000000	3.000000	156.000000	3.000000	0.00
50%	64.000000	72.000000	4.000000	200.000000	3.000000	0.00
75%	82.000000	87.000000	5.000000	245.000000	4.000000	0.00
max	100.000000	100.000000	7.000000	310.000000	10.000000	1.00

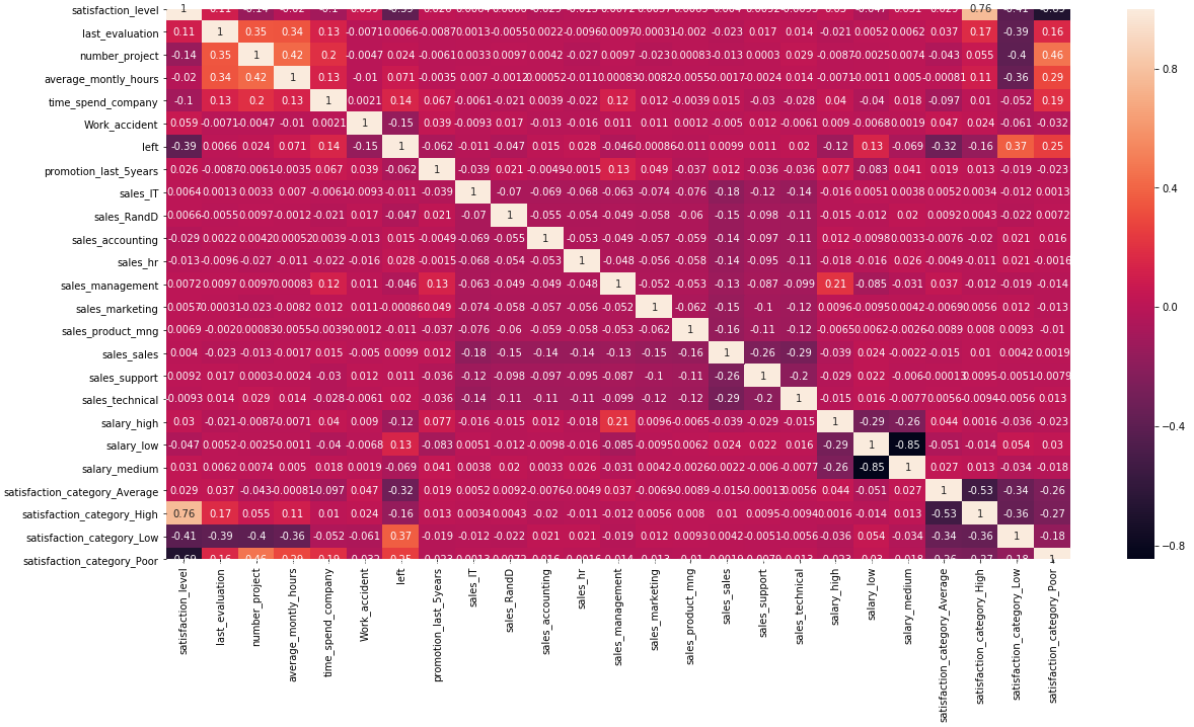
8 rows × 25 columns

Comment: I have gone from 10 variables/columns to many more variables/columns as a result of creating dummy variables.

Correlation analysis: initial analysis to get a quick overview of correlation between all variables in the dataset follows next

In [23]:
```
# as the dataset is huge, calculating correlation matrix first
cor_matrix = dum_dataset.corr()
```

In [24]:
```
# plotting the matrix separately
plt.figure(figsize=(20, 10))
sns.heatmap(cor_matrix, annot=True)
plt.show()
```



Comment: as expected I see a strong correlation between satisfaction_level and left (probably because employees who have left were dissatisfied with sometihng). There doesn't seem to be too strong correlation between satisfaction and employee department variables based on the above though!

With employee survey results dataset at UBS we have ca 30 numerical variables such as satisfaction_level in the this dataset, so the above picture should be much more interesting! Additionally, a lot of the employee "mood" could depend on country specific factors, so I'd also expect see more correlation between country and engagement rating at UBS.

7) Multivariate linear regression

Let's get started with the regression analysis below. First I check that I have a complete dataset and that data types are as I expect them to be

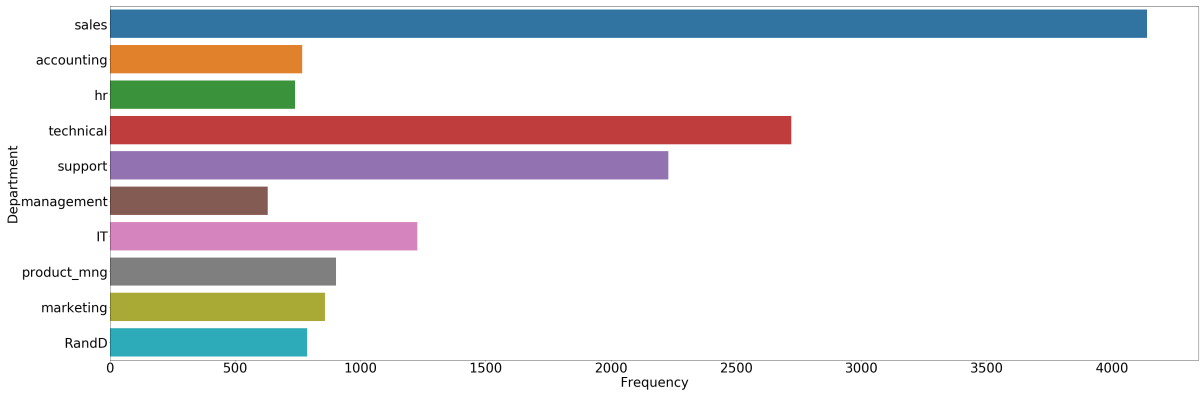
In [25]:
```
dum_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 25 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_IT                         14999 non-null uint8
sales_RandD                      14999 non-null uint8
sales_accounting                 14999 non-null uint8
sales_hr                         14999 non-null uint8
sales_management                 14999 non-null uint8
sales_marketing                  14999 non-null uint8
sales_product_mng                14999 non-null uint8
sales_sales                      14999 non-null uint8
sales_support                    14999 non-null uint8
sales_technical                  14999 non-null uint8
salary_high                      14999 non-null uint8
salary_low                       14999 non-null uint8
salary_medium                    14999 non-null uint8
satisfaction_category_Average    14999 non-null uint8
satisfaction_category_High       14999 non-null uint8
satisfaction_category_Low        14999 non-null uint8
satisfaction_category_Poor       14999 non-null uint8
dtypes: float64(2), int64(6), uint8(17)
memory usage: 1.2 MB
```

Comment: I now have 21 variables and I seem to have no missing values. I do suspect though, that departments are not equally represented...

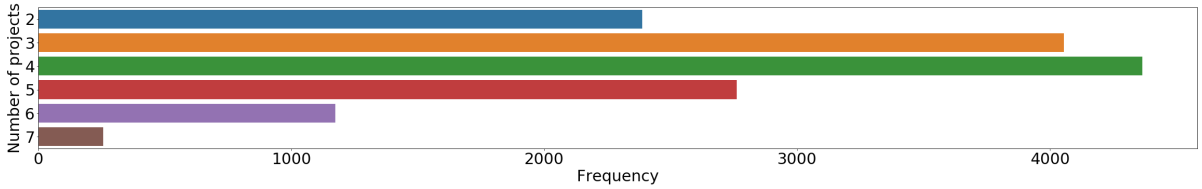
Next, I take a quick look at the histogram on how many times are different departments represented. Here I wrote some code to make it look nicer:

In [26]:
```
chart2 = sns.countplot(y='sales', data=dataset)
chart2.figure.set_size_inches(60,20)
chart2.set_xlabel("Frequency", fontsize=40)
chart2.set_ylabel("Department", fontsize=40)
chart2.tick_params(labelsize = 40)
plt.show()
```



Comment: departments are not exactly equally represented, but the good news is that I have sufficient data for each of the department, so I don't need to necessarily drop any of this data. How would a similar plot look like for the number of projects employees are working on? This is how:

In [27]:
```
chart2 = sns.countplot(y='number_project', data=dataset)
chart2.figure.set_size_inches(40,5)
chart2.set_xlabel("Frequency", fontsize=30)
chart2.set_ylabel("Number of projects", fontsize=30)
chart2.tick_params(labelsize = 30)
plt.show()
```



Comment: Again, good that I have sufficient reults for each category. There seem to be some people with 7(!) projects which may be a bit much, but let's keep going. Let's quickly also do a similar check for Work_accident, left, promotion_last_5years and salary categories

In [28]:
```
print("left: \n", dataset.left.value_counts(),"\n",
      "==========\n"
     "Work_accident: \n", dataset.Work_accident.value_counts(),"\n",
      "==========\n"
     "promotion_last_5years: \n", dataset.promotion_last_5years.value_counts(),"\n",
      "==========\n"
     "salary: \n", dataset.salary.value_counts())
```

```
left:
 0    11428
1     3571
Name: left, dtype: int64
 ==========
Work_accident:
 0    12830
1     2169
Name: Work_accident, dtype: int64
 ==========
promotion_last_5years:
 0    14680
1      319
Name: promotion_last_5years, dtype: int64
 ==========
salary:
 low       7316
medium    6446
high      1237
Name: salary, dtype: int64
```

Comment: I can see from above that salary variable is the most evenly distributed, and promotion_last_5years is the least even with only 319 people being promoted, compared to 14680 not promoted. Still, I have at least several hundred instances for each outcome on these variables, which means I can use it in our analysis.

Defining regression variables: Now I define which variables I want to take into account in our regression model. Based on the analysis above, to start with, I will focus on all variables as they have a reasonably significant representation in the dataset.

*If I had variables with only "tens" of instances, I would most likely drop them, as it would be unreliable to draw any conclusions on these - this means, that there would be a relatively high probability of any identified relationship being due to random chance.

satisfaction_level will be our 'y' (dependent) variable that I will predict by using X variables

all other variables in the dataset will be defined as 'X' (independent) variables of the model.

After defining y and X variables I randomise our dataset and then split it to train & test portions (75% / 25% respectively)

Putting this into context, I would go through the same steps with the dataste at my company. Employee survey questions and some dummy variables (rank, country etc.) would be defined as independent variables and employee engagement rating would be the dependent variable that I'm trying to predict.

Randomising and splitting to train and test portions would be done the same way in both datasets

In [29]:
```
# creating subsets for multivariate model

X = dum_dataset[["last_evaluation",
                 "number_project",
                 "average_montly_hours",
                 "time_spend_company",
                 "Work_accident",
                 "left",
                 "promotion_last_5years",
                 "sales_IT",
                 "sales_RandD",
                 "sales_accounting",
                 "sales_hr",
                 "sales_management",
                 "sales_marketing",
                 "sales_product_mng",
                 "sales_sales",
                 "sales_support",
                 "sales_technical",
                 "salary_high",
                 "salary_low",
                 "salary_medium"]]
y = dum_dataset[['satisfaction_level']]
X_train, X_test, y_train, y_test=train_test_split(X, y,
                                                  test_size = 0.25,
                                                  random_state = 123)
```

Quick look below at the dimensions of the train/test datasets to test splitting results will reveal I have successfully split the data

In [30]:
```
print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)
```

```
(11249, 20) (11249, 1)
(3750, 20) (3750, 1)
```

Creating the linear regression model below and training it with our training dataset

In [31]:
```
# Creating linear regression object
regression_model = linear_model.LinearRegression()

# Training the model using the training sets
regression_model = regression_model.fit(X_train, y_train)
```

Initial model results: After having created and trained the model, let's take a quick look at how well does the model explain y results by using X variables. This metric is called R-squared:

In [32]:
```
# The R-squared
print('The R-squared is:', regression_model.score(X_train, y_train))
```

```
The R-squared is: 0.1997029094859163
```

R-squared of 19.97% can be considered very low, and is not sufficient for a well fitted model. This means that our model is able to explain only 19.97% of the employee satisfaction_level.

Given that I'm predicting satisfaction_level only based on a very small number of variables, this is to be expected. Somewhat surprising perhaps is that departments and salary levels don't explain satisfaction_level more than that...

Next, let's take a quick look at coefficients and the intercept of our regression model

In [33]:
```
# The coefficients
print('Coefficients: \n', regression_model.coef_)
```

```
Coefficients:
 [[ 2.45269169e-01 -4.04539212e+00  1.77063668e-02 -4.97710359e-01
  -1.99737144e-01 -2.25764848e+01  5.57997650e-01  9.52493119e-01
  -1.15726079e+00 -1.89460135e+00 -9.13729956e-01  4.24902017e-01
  -9.03977541e-02  7.27804975e-01  8.15682549e-01  3.65128802e-01
   7.69978384e-01 -4.64751163e-01  6.98199355e-02  3.94931227e-01]]
```

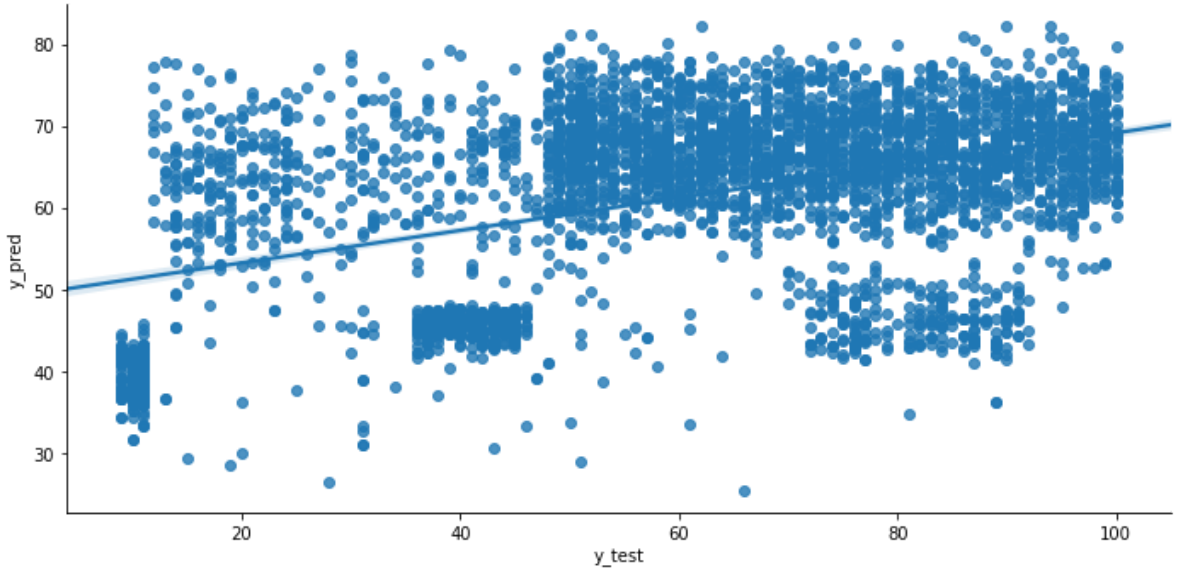
In [34]:
```
# The intercept
print('Intercept: \n', regression_model.intercept_)
```

```
Intercept:
 [62.26700139]
```

Plotting results: first let's get predicted y results from our model by using the x_test dataset and then plot them on the chart along with the y_test (real results) to compare accuracy.

In [35]:
```
y_pred = regression_model.predict(X_test)
y_pred = pd.DataFrame(y_pred)
```

In [36]:
```
y_test = y_test['satisfaction_level'].reset_index()
y_obs_pred = pd.concat([y_test['satisfaction_level'], y_pred], axis=1)
y_obs_pred.columns = ['y_test', 'y_pred']
y_obs_pred['diff']= y_obs_pred['y_test'] - y_obs_pred['y_pred']
r = sns.lmplot(x='y_test', y = 'y_pred', data=y_obs_pred, height=4.9, aspect=2)
```



Comment: for an accuracte model y_test ~ y_pred above, but I already know that I'm are able to explain only 19.97% of the results.

7.1) Multivariate regression model in STATSMODEL

Running the same model by using statsmodel libraries.

In [37]:
```
# Running the same regression in statsmodel

Train = pd.concat([X_train, y_train], axis=1)
Formula = 'satisfaction_level ~ last_evaluation + number_project + average_montly_hours + t
ime_spend_company + Work_accident + left + promotion_last_5years + sales_IT + sales_RandD +
sales_accounting + sales_hr + sales_management + sales_marketing + sales_product_mng + sale
s_sales + sales_support + sales_technical + salary_high + salary_low + salary_medium'
regression_model = smf.ols(Formula, data = Train).fit()
print(regression_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     satisfaction_level   R-squared:                       0.200
Model:                            OLS   Adj. R-squared:                  0.198
Method:                 Least Squares   F-statistic:                     155.7
Date:                Tue, 30 Jun 2020   Prob (F-statistic):               0.00
Time:                        16:36:56   Log-Likelihood:                -50870.
No. Observations:               11249   AIC:                         1.018e+05
Df Residuals:                   11230   BIC:                         1.019e+05
Df Model:                          18
Covariance Type:            nonrobust
==============================================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
Intercept                43.4421      0.813     53.464      0.000      41.849      45.035
last_evaluation           0.2453      0.013     18.227      0.000       0.219       0.272
number_project           -4.0454      0.195    -20.775      0.000      -4.427      -3.664
average_montly_hours      0.0177      0.005      3.712      0.000       0.008       0.027
time_spend_company       -0.4977      0.148     -3.363      0.001      -0.788      -0.208
Work_accident            -0.1997      0.606     -0.330      0.742      -1.388       0.988
left                    -22.5765      0.516    -43.762      0.000     -23.588     -21.565
promotion_last_5years     0.5580      1.479      0.377      0.706      -2.341       3.457
sales_IT                  5.2967      0.714      7.417      0.000       3.897       6.696
sales_RandD               3.1869      0.876      3.640      0.000       1.471       4.903
sales_accounting          2.4496      0.889      2.757      0.006       0.708       4.192
sales_hr                  3.4305      0.892      3.844      0.000       1.681       5.180
sales_management          4.7691      0.971      4.909      0.000       2.865       6.673
sales_marketing           4.2538      0.829      5.129      0.000       2.628       5.879
sales_product_mng         5.0720      0.813      6.240      0.000       3.479       6.665
sales_sales               5.1599      0.439     11.755      0.000       4.299       6.020
sales_support             4.7093      0.553      8.522      0.000       3.626       5.793
sales_technical           5.1142      0.514      9.945      0.000       4.106       6.122
salary_high              14.0159      0.631     22.212      0.000      12.779      15.253
salary_low               14.5505      0.413     35.263      0.000      13.742      15.359
salary_medium            14.8756      0.418     35.577      0.000      14.056      15.695
==============================================================================
Omnibus:                      245.829   Durbin-Watson:                   2.011
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              162.469
Skew:                          -0.170   Prob(JB):                     5.25e-36
Kurtosis:                       2.520   Cond. No.                     2.90e+18
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 6.43e-29. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Comment: I can see that Work_accident and promotion_last_5years have high p-values above (meaning their significance is lower), so I will remove these variables and train the model again to see if it improves the results.

I would do the same steps with any variables with high p-values in our internal dataset at my employer

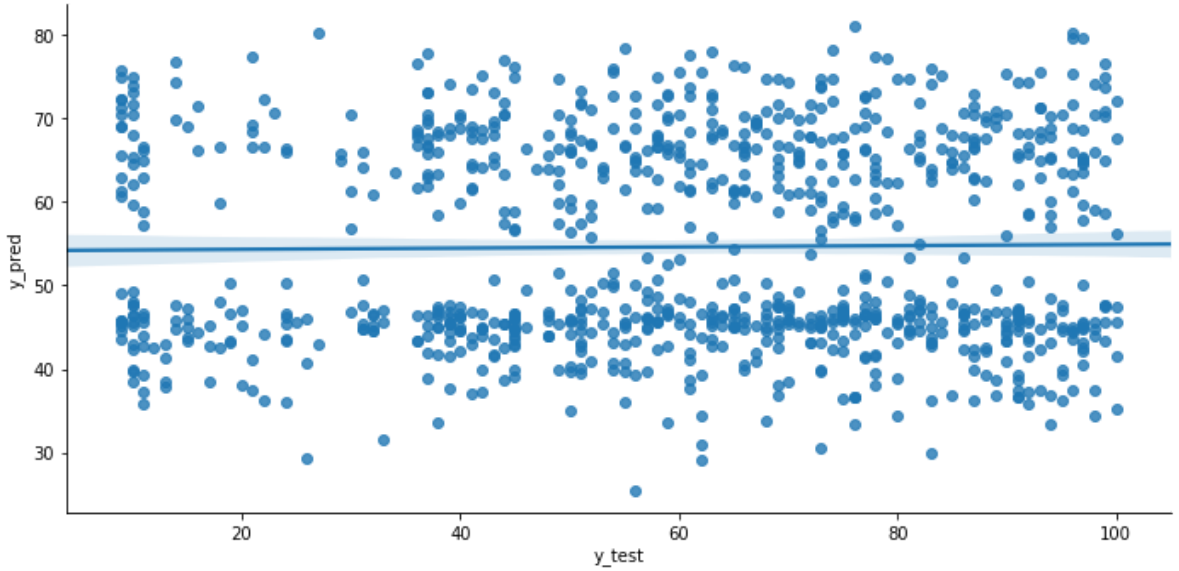
In [38]:
```
# Removing Work_accident and promotion_last_5years due to high p-values

Train = pd.concat([X_train, y_train], axis=1)
Formula = 'satisfaction_level ~ last_evaluation + number_project + average_montly_hours + time_spend_company + left + sales_IT + sales_RandD + sales_accounting + sales_hr + sales_management + sales_marketing + sales_product_mng + sales_sales + sales_support + sales_technical + salary_high + salary_low + salary_medium'
regression_model = smf.ols(Formula, data = Train).fit()
print(regression_model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     satisfaction_level   R-squared:                       0.200
Model:                            OLS   Adj. R-squared:                  0.199
Method:                 Least Squares   F-statistic:                     175.2
Date:                Tue, 30 Jun 2020   Prob (F-statistic):               0.00
Time:                        16:36:56   Log-Likelihood:                -50870.
No. Observations:               11249   AIC:                         1.018e+05
Df Residuals:                   11232   BIC:                         1.019e+05
Df Model:                          16
Covariance Type:            nonrobust
==============================================================================
                             coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept                43.4310      0.809     53.673      0.000      41.845      45.017
last_evaluation           0.2452      0.013     18.225      0.000       0.219       0.272
number_project           -4.0455      0.195    -20.783      0.000      -4.427      -3.664
average_montly_hours      0.0177      0.005      3.712      0.000       0.008       0.027
time_spend_company       -0.4957      0.148     -3.359      0.001      -0.785      -0.206
left                    -22.5589      0.509    -44.346      0.000     -23.556     -21.562
sales_IT                  5.2860      0.713      7.410      0.000       3.888       6.684
sales_RandD               3.1931      0.875      3.649      0.000       1.478       4.908
sales_accounting          2.4481      0.889      2.755      0.006       0.706       4.190
sales_hr                  3.4309      0.892      3.845      0.000       1.682       5.180
sales_management          4.8010      0.967      4.964      0.000       2.905       6.697
sales_marketing           4.2611      0.829      5.142      0.000       2.637       5.885
sales_product_mng         5.0540      0.812      6.226      0.000       3.463       6.645
sales_sales               5.1572      0.439     11.751      0.000       4.297       6.017
sales_support             4.6966      0.552      8.509      0.000       3.615       5.778
sales_technical           5.1030      0.514      9.935      0.000       4.096       6.110
salary_high              14.0205      0.630     22.251      0.000      12.785      15.256
salary_low               14.5376      0.412     35.322      0.000      13.731      15.344
salary_medium            14.8729      0.417     35.647      0.000      14.055      15.691
==============================================================================
Omnibus:                      246.127   Durbin-Watson:                   2.011
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              162.745
Skew:                          -0.171   Prob(JB):                     4.57e-36
Kurtosis:                       2.520   Cond. No.                     1.70e+18
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 1.88e-28. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

Comment: after removing 2 least significant variables, the results have barely changed (R-squared still 20%). Finally, I use this (new) model to make predictions and I plot the results similarly to what I did previously against real values

In [39]:
```
y_pred = regression_model.predict(X_test)
y_pred = pd.DataFrame(y_pred)
```

In [40]:
```
# Comparing model prediction to actual numbers in a scatterplot.

y_test = y_test['satisfaction_level'].reset_index()
y_obs_pred = pd.concat([y_test['satisfaction_level'], y_pred], axis=1)
y_obs_pred.columns = ['y_test', 'y_pred']
y_obs_pred['diff']= y_obs_pred['y_test'] - y_obs_pred['y_pred']
r = sns.lmplot(x='y_test', y = 'y_pred', data=y_obs_pred, height=4.9, aspect=2)
```



Comment: As seen also from the plot above, the regression fit line doesn't predict our results very accurately.

8) Multivariate regression with internal data

Finally, I will run the same analysis as above with limited/aggregated internal data that is avaiable to me. Because this data is so sensitive, the following analysis serves as a suggestion to our HR team, who can then run the analysis with full data to uncover new insights.

Below analysis is run on already cleaned aggregated country results data. For more explanations on each step, please refer to section 7 of this notebook:

In [41]:
```
# importing dataset
survey = pd.read_csv('Survey data.csv')

# making sure dataset is a dataframe
survey = pd.DataFrame(survey)

# Looking at the header of the dataset. Next step is commented to preserve confidentiality
survey.head()
```

Out[41]:

	n								
0	278	80	63	71	62	51	61	67	72
1	534	74	61	70	58	41	56	60	70
2	14	64	50	56	57	39	35	67	61
3	492	76	52	57	58	62	56	63	73
4	4343	79	59	74	64	63	61	67	78

5 rows × 34 columns

Comment: the dataset looks as expected. Next, checking for missing values and data types

In [42]:
```
# getting the list of variables and verifying there are no missing values. Next step commen
ted to preserve confidentiality
survey.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17 entries, 0 to 16
Data columns (total 34 columns):
n                               17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
                                17 non-null int64
GER                             17 non-null int64
IRL                             17 non-null int64
ITA                             17 non-null int64
UK                              17 non-null int64
ESP                             17 non-null int64
JEY                             17 non-null int64
MCO                             17 non-null int64
AUT                             17 non-null int64
NDL                             17 non-null int64
SWE                             17 non-null int64
ISR                             17 non-null int64
FRA                             17 non-null int64
UAE                             17 non-null int64
RUS                             17 non-null int64
POL                             17 non-null int64
ZAF                             17 non-null int64
dtypes: int64(34)
memory usage: 4.6 KB
```

Next: creating subsets for multivariate model. Keeping only survey question variables as country data is limited

In [43]:
```
# creating subsets for multivariate model. Keeping only survey question variables as countr
y data is limited

X = survey[["feature1",









                ]]

y = survey[['Employee engagement']]
X_train, X_test, y_train, y_test=train_test_split(X, y,
                                                  test_size = 0.5,
                                                  random_state = 123)


# checking the sizes of train and test datasets

print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)
```

```
(8, 14) (8, 1)
(9, 14) (9, 1)
```

Survey results are now divided into groups of 8 and 9, train and test sets, respectively.

Next, I fit the data to a linear regression model and print R-squared result

In [44]:
```
# Creating linear regression object
regression_model2 = linear_model.LinearRegression()

# Training the model using the training sets
regression_model2 = regression_model2.fit(X_train, y_train)

# The R-squared
print('The R-squared is:', regression_model2.score(X_train, y_train))
```

```
The R-squared is: 1.0
```

Comment: R-squared 1 is almost certainly overfitting the situation due to limited data and as such is too high. When running with full data, we can expect R-squared to drop.

Let's check out the intercept and coefficient values next:

In [45]:
```
# The intercept
print('Intercept: \n', regression_model2.intercept_)

# The coefficients
print('Coefficients: \n', regression_model2.coef_)
```

```
Intercept:
 [21.07476249]
Coefficients:
 [[ 0.30091463 -0.00591583  0.4518057   0.27826581 -0.19470011 -0.1303452
   0.02633942  0.20037486 -0.29775592 -0.02642489 -0.15015778  0.02747386
   0.33539094  0.0147239 ]]
```

Comment: After a quick look, we can tell that [xxx] is the factor impacting employee engagement most positively.

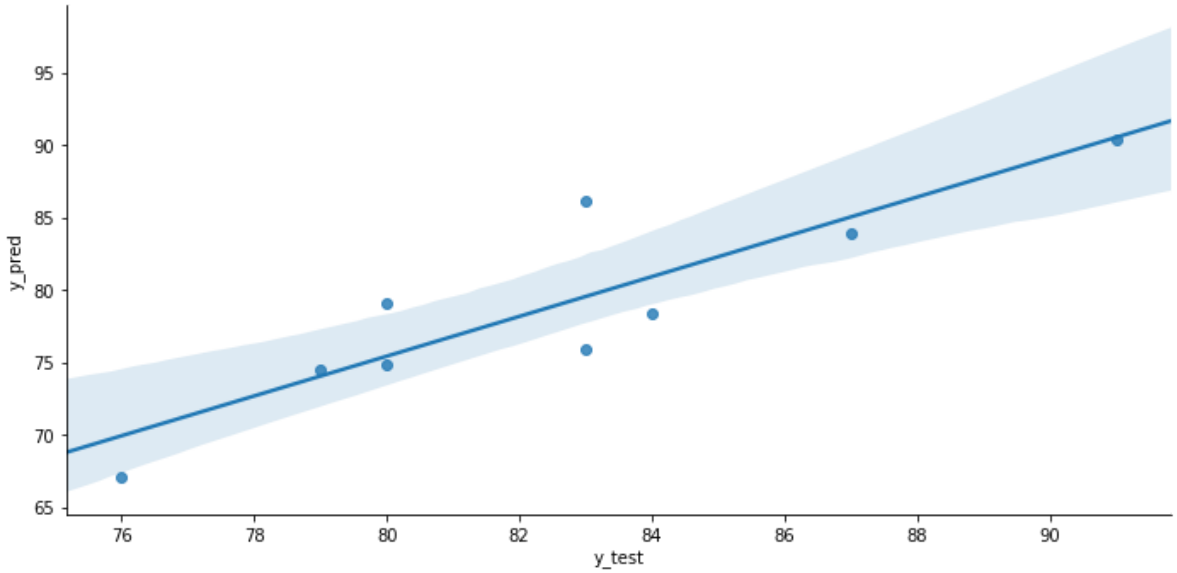
Everything else constant, 1 point increase in **[xxx]** rating would lead to 0.4518 points increase in employee engagement rating.

At the same time, increase in **[xxx]** rating seems to be most detrimental to employee engagement ratings. 1 point increase in [xxx] rating seems to result in -0.2978 points change in employee engagement score.

Next, I make predictions using the model and plot it on a chart against real survey values to visualise how accurately our model predicts

In [46]:
```
y_pred = regression_model2.predict(X_test)
y_pred = pd.DataFrame(y_pred)
```

In [47]:
```
y_test = y_test['Employee engagement'].reset_index()
y_obs_pred = pd.concat([y_test['Employee engagement'], y_pred], axis=1)
y_obs_pred.columns = ['y_test', 'y_pred']
y_obs_pred['diff']= y_obs_pred['y_test'] - y_obs_pred['y_pred']
r = sns.lmplot(x='y_test', y = 'y_pred', data=y_obs_pred, height=4.9, aspect=2)
```



In [50]:
```
from sklearn import metrics
print('Mean Squared Error:', metrics.mean_squared_error(y_obs_pred['y_test'], y_obs_pred['y
_pred']))
```

```
Mean Squared Error: 25.118839118864408
```

Comment / conclusion: in this limited data example, I can see that the regression model fits the data fairly well despite being a linear regression model and having limited data - this is a good start, but far from perfect/realistic.

Technically speaking, I've also calculated Mean Squared Error (MSE) above to show the accuracy of our predictions. Roughly speaking, MSE shows the average of a set of prediction errors. The smaller the MSE the better/more accurate results I have.

As a next step I need to reach out to HR and ask them to run the analysis on complete data.

9) Summary Conclusion

Summary findings (external data):
1) Given available data I have built a model analysing how employee profile and survey responses affect employee satisfaction levels. Our model explains 20% (R^2) of employee satisfaction results.

2) From the results it can be concluded that there is a negative correlation between employee satisfaction levels and the time spent in the company, number of projects the employee is working on and also employee having left the company. On average, empployees who have left the company had rated their satisfaction levels 22.56 points lower, everything else constant.

3) Given available data, I could also model the effect of employee department and salary band to employee satisfaction level. Everything else constant, employees in IT and sales departments seem to rate their satisfaction levels highest. Employees in the Accounting department seem to give lowest ratings on their satisfaction levels.

4) Employees with the following profile/survey responses are most likely to rate their satisfaction levels the highest:
- positive last evaluation
- working on small number of projects
- with high average monthly hours
- new to the company (little time spent in the company)
- have not left the company
- working in IT department
- having a high salary range

5) Employees with the following profile/survey responses are most likely to rate their satisfaction levels the lowest:
- negative last evaluation
- working on large number of projects
- with low average monthly hours
- been with the company for a long time
- have left left the company
- working in Accounting department
- having a medium salary range

Real life implications:
Based on our analysis results I could suggest the company's senior management to consider the following:

1) Conduct an employee survey (could be done for free online) and introduce more questions to ask employees at least the following questions: how long they're seeing themselves working with the company and perhaps ask them also to identify what's the cause of biggest dissatisfaction for them at work. Conduct the survey anonymously to get true responses and treat these the responses as an early warning of potential turnover. Take action on areas of biggest dissatisfactions.

2) Bring the heads of accounting (worst performaing) and management (best performing) departments together to exchange "best practice" and discuss how each keep up their employees morale and job satisfaction.

3) Take a look into how many projects each employee works on. As I uncovered in our analysis earlier, there are several people working on 7 projects and quite a few on 2 projects only. Working on too many projects at the same time seems to affect job satisfaction levels negatively. If possible, reallocating/rebalancing number of projects between employees may be a quick an cost effective way to improve satisfaction levels.

Suggestions for further analysis:
This analysis is meant to show the process of regression analysis. It is limited by available data and could be definitely improved in accuracy by including additional data on factors affecting employee satisfaction. One way to do that would be asking employees for additional data via surveys e.g. how happy are employees with senior management, are they proud to work at their employer, how do they view the company compared to their competitors etc.

As the R^2 is very low, this model shouldn't be used for performing predictions. However, all the features studied are statistically significant and they must be taken in account when trying to predict satisfaction level ratings.

As a suggestion for further analysis, it may be intersting to use Random Forest Regression analysis to take on this task. This would enable us to take into account also combinations of relationships. Whereas linear regression simply assigns a positive or negative factor to each variable, Random Forest regression analysis would be able to uncover combinations when for example 2 negative factors together would actually affect the satisfaction level positively.

Example: Time spent in company and number of projects (both negative individually), could turn out to be positive if taken together. This could be because working on many projects early in the role may be overwhelming, but people with significant experience and a lot of ongoing projects may be doing it because they've become good at it and as a result love their jobs.

***Implied conclusion in the case of UBS,** should I have similar results on our full employee survey dataset, this model would have too little explanatory power to be able to use it to improve employee engagement. However, if on UBS data I for example discover that innovation rating has a statistically significant and a strong relationship to explainging employee engagement, I could introduce more (or less) innovation related initiatives in order to improve our employee satisfaction in the company.*

At the same should I discover that country variables are statistically significant in explaining employee engagement rating at UBS, it may be worth taking a closer look into these countries to understand what are they doing differently to other countries and perhaps be able to share best practice with other lower performing countries to improve their results.

In [ ]: