TeamA_Final_Report

December 11, 2019

MSIN0143 Programming for Business Analytics

Analysis on Millennials in the Workforce

Group Project of Team A

Outline

****

1. Introduction
   1.1 Business Context

Employee attrition can be costly for a company. For instance, a company needs to spend time and resources on recruiting and training new talent. Skill gaps between new employees and experienced employees may reduce productivity, thus affecting overall profits. A high turnover rate is also especially concerning in customer facing roles as customers often prefer to interact with the same people, rather than with new individuals each time (Incentius.com, 2019).

The increasing number of Millennials in today's workplace may increase the overall attrition rate as they are harder to retain than previous generations (Bannon et al., 2011). According to KPMG (2017), Millennials prefer flexibility and are more entrepreneurial than Gen X, indicating that they are more difficult to retain. Apart from the age gap, there are also many other factors that may affect employee attrition rate such as gender, education and so on. Hence, many researches have been conducted to find factors that can affect employee attrition rate in an organization.

This project, therefore, aims to help businesses to mitigate their employee attrition rate based on both descriptive and predictive analysis of an employee attrition dataset. To better meet the trends in the future workplace, this project will put emphasis on Millennials.

1.2 Getting Data

Where and Why did we get this dataset

We get the data from the kaggle: https://www.kaggle.com/pavansubhasht/ibm-hr-analytics-attrition-dataset.

The dataset contains data for employees aged 18 - 60 years old and is comprised of over 30 attributes (e.g. employees income, age,job role, the distance they have to travel to work, etc.) which we will explore in detail below.

Our analysis is based on the assumption that employees leave the company during one year and no one joins the company during that same year.

Import libraries

[1]:
```
conda install -c conda-forge xgboost #Install xgboost package in our environment
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
  current version: 4.7.12
  latest version: 4.8.0

Please update conda by running

    $ conda update -n base -c defaults conda



# All requested packages already installed.


Note: you may need to restart the kernel to use updated packages.
```

[2]:
```
# Python libraries
import pandas as pd #Data Analysis for tabular data
import numpy as np # Scientific computing and data manipulation
import scipy.stats as stats # Statistical package

import matplotlib.pyplot as plt # Visualizations
from matplotlib import rcParams  #Customizing Matplotlib with style sheets and␣
 ↪rcParams
import seaborn as sns #Advances Visualizations based on matplotlib.pyplot

import xgboost as xgb #Gradient Boosting Framework for Machine Learning

import warnings #Warning control

from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

#from sklearn.metrics import precision_score, roc_auc_score, recall_score,␣
↪confusion_matrix, roc_curve, precision_recall_curve, accuracy_score


#Setting general properties in this document
pd.set_option('display.max_columns', 50) #show all the columns when we call our␣
↪dataset
warnings.filterwarnings('ignore') #do not display warning messages
```

Import dataset

First we import our dataset:

```
[3]: raw_data = pd.read_csv('/project/HR-Employee-Attrition.csv') # pandas's␣
↪read_csv() function to read .csv files
```

Now we are going to get the size of our imported dataset.

```
[4]: raw_data.shape
```

```
[4]: (1470, 35)
```

We can see that it has 1470 observations (no. of employees) and 35 attributes. We extract these 35 attributes below.

```
[5]: raw_data.columns
```

```
[5]: Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
        'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
        'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
        'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
        'MaritalStatus', 'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
        'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
        'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
        'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
        'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
        'YearsWithCurrManager'],
       dtype='object')
```

As we have many attributes, we believe that examining each one will not be time-efficient. For this reason we create lists that keep the name of the columns with the same data type (numerical, categorical). These lists will be used further in our report.

```
[6]: numerical_columns = ['Age', 'DailyRate', 'DistanceFromHome', 'EmployeeCount',
 'HourlyRate', 'MonthlyIncome', 'MonthlyRate',
```

```
 'NumCompaniesWorked','PercentSalaryHike',  'StandardHours',
 'TotalWorkingYears', 'TrainingTimesLastYear','YearsAtCompany',
 'YearsInCurrentRole', 'YearsSinceLastPromotion',
 'YearsWithCurrManager']

categorical_columns = ['Attrition', 'BusinessTravel', 'Department', 'Education',
'EducationField', 'EnvironmentSatisfaction',
'Gender', 'JobInvolvement', 'JobLevel', 'JobRole',
'JobSatisfaction', 'MaritalStatus','Over18',
'OverTime', 'PerformanceRating', 'RelationshipSatisfaction',
'StockOptionLevel','WorkLifeBalance' ]
```

2. Exploring the dataset
   2.1 Get general metrics for each attribute / Identify outliers

[7]:
```
df=raw_data.copy() #We make a copy so we can always refer to it without risking␣
↪to change our original dataset by mistake.
```

Below we see if we have any missing values:

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
Age                         1470 non-null int64
Attrition                   1470 non-null object
BusinessTravel              1470 non-null object
DailyRate                   1470 non-null int64
Department                  1470 non-null object
DistanceFromHome            1470 non-null int64
Education                   1470 non-null int64
EducationField              1470 non-null object
EmployeeCount               1470 non-null int64
EmployeeNumber              1470 non-null int64
EnvironmentSatisfaction     1470 non-null int64
Gender                      1470 non-null object
HourlyRate                  1470 non-null int64
JobInvolvement              1470 non-null int64
JobLevel                    1470 non-null int64
JobRole                     1470 non-null object
JobSatisfaction             1470 non-null int64
MaritalStatus               1470 non-null object
MonthlyIncome               1470 non-null int64
MonthlyRate                 1470 non-null int64
NumCompaniesWorked          1470 non-null int64
Over18                      1470 non-null object
OverTime                    1470 non-null object
```

```
PercentSalaryHike           1470 non-null int64
PerformanceRating           1470 non-null int64
RelationshipSatisfaction    1470 non-null int64
StandardHours               1470 non-null int64
StockOptionLevel            1470 non-null int64
TotalWorkingYears           1470 non-null int64
TrainingTimesLastYear       1470 non-null int64
WorkLifeBalance             1470 non-null int64
YearsAtCompany              1470 non-null int64
YearsInCurrentRole          1470 non-null int64
YearsSinceLastPromotion     1470 non-null int64
YearsWithCurrManager        1470 non-null int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

As the number of non-null values is the same for every column, we conclude that we do not have any missing values.

Next, we want to check our data for any outliers. We do this by looking at the data points in the numerical columns (as defined in Import dataset section). We could do this as well by generating boxplots for each variable but this seems to be rather inconvenient due to the number of numerical variables. Therefore we decided to go for the approach using a mathematical function. If the z-score value is greater than or less than 3 or -3 respectively, this data point will be identified as outliers (Medium, 2019).

[9]:
```
zscores = pd.DataFrame(stats.zscore(df.loc[:, numerical_columns])>3) #returns␣
↪True if there is an outlier in the subsequent cell.
zscores.columns = df.loc[:, numerical_columns].columns #getting the columns'␣
↪names
zscores
```

[9]:
```
        Age  DailyRate  DistanceFromHome  EmployeeCount  HourlyRate  \
0     False      False             False          False       False
1     False      False             False          False       False
2     False      False             False          False       False
3     False      False             False          False       False
4     False      False             False          False       False
…       …          …                 …              …           …
1465  False      False             False          False       False
1466  False      False             False          False       False
1467  False      False             False          False       False
1468  False      False             False          False       False
1469  False      False             False          False       False

      MonthlyIncome  MonthlyRate  NumCompaniesWorked  PercentSalaryHike  \
0             False        False               False              False
1             False        False               False              False
2             False        False               False              False
```

```
3             False     False              False              False
4             False     False              False              False
...           ...       ...                ...                ...
1465          False     False              False              False
1466          False     False              False              False
1467          False     False              False              False
1468          False     False              False              False
1469          False     False              False              False

      StandardHours  TotalWorkingYears  TrainingTimesLastYear  YearsAtCompany  \
0             False              False                  False           False
1             False              False                  False           False
2             False              False                  False           False
3             False              False                  False           False
4             False              False                  False           False
...             ...                ...                    ...             ...
1465          False              False                  False           False
1466          False              False                  False           False
1467          False              False                  False           False
1468          False              False                  False           False
1469          False              False                  False           False

      YearsInCurrentRole  YearsSinceLastPromotion  YearsWithCurrManager
0                  False                    False                 False
1                  False                    False                 False
2                  False                    False                 False
3                  False                    False                 False
4                  False                    False                 False
...                  ...                      ...                   ...
1465               False                    False                 False
1466               False                    False                 False
1467               False                    False                 False
1468               False                    False                 False
1469               False                    False                 False

[1470 rows x 16 columns]
```

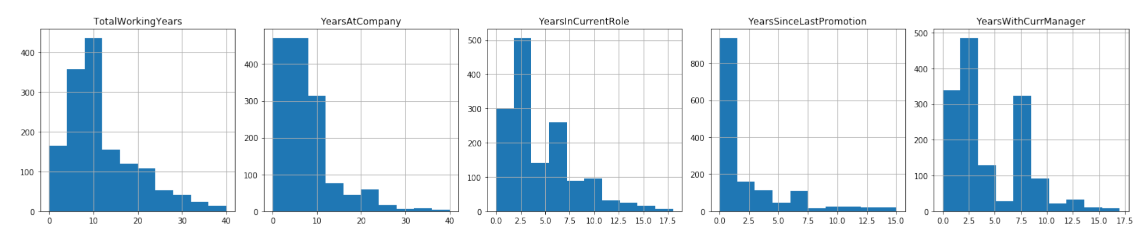
[10]:
```
zscores.eq(True).any() #As we do not want to go through all cells manually and
↪look for any True values, we use the next function to detect if there are
↪any True values.
```

[10]:
```
Age                         False
DailyRate                   False
DistanceFromHome            False
EmployeeCount               False
HourlyRate                  False
MonthlyIncome               False
```

```
MonthlyRate                 False
NumCompaniesWorked          False
PercentSalaryHike           False
StandardHours               False
TotalWorkingYears            True
TrainingTimesLastYear       False
YearsAtCompany               True
YearsInCurrentRole           True
YearsSinceLastPromotion      True
YearsWithCurrManager         True
dtype: bool
```

Indeed we can see that there are 5 columns which contain True values, i.e. they indicate that there are outliers in these columns. We look at the histograms and the minimum and the maximum values of these variables to make sense of whether the indicated outliers are actual or just statistical outliers.

[11]:
```
plt.rcParams['figure.figsize'] = [20, 4]
fig, axes = plt.subplots(nrows=1, ncols=5)

df.loc[:, numerical_columns].loc[:,'TotalWorkingYears'].hist(ax=axes[0]).title.
↪set_text('TotalWorkingYears')
df.loc[:, numerical_columns].loc[:,'YearsAtCompany'].hist(ax=axes[1]).title.
↪set_text('YearsAtCompany')
df.loc[:, numerical_columns].loc[:,'YearsInCurrentRole'].hist(ax=axes[2]).title.
↪set_text('YearsInCurrentRole')
df.loc[:, numerical_columns].loc[:,'YearsSinceLastPromotion'].hist(ax=axes[3]).
↪title.set_text('YearsSinceLastPromotion')
df.loc[:, numerical_columns].loc[:,'YearsWithCurrManager'].hist(ax=axes[4]).
↪title.set_text('YearsWithCurrManager')

plt.tight_layout()
```



[12]:
```
df.loc[:, ['TotalWorkingYears','YearsAtCompany','YearsInCurrentRole',␣
↪'YearsSinceLastPromotion', 'YearsWithCurrManager']].describe().
↪loc[['min','max']]
```

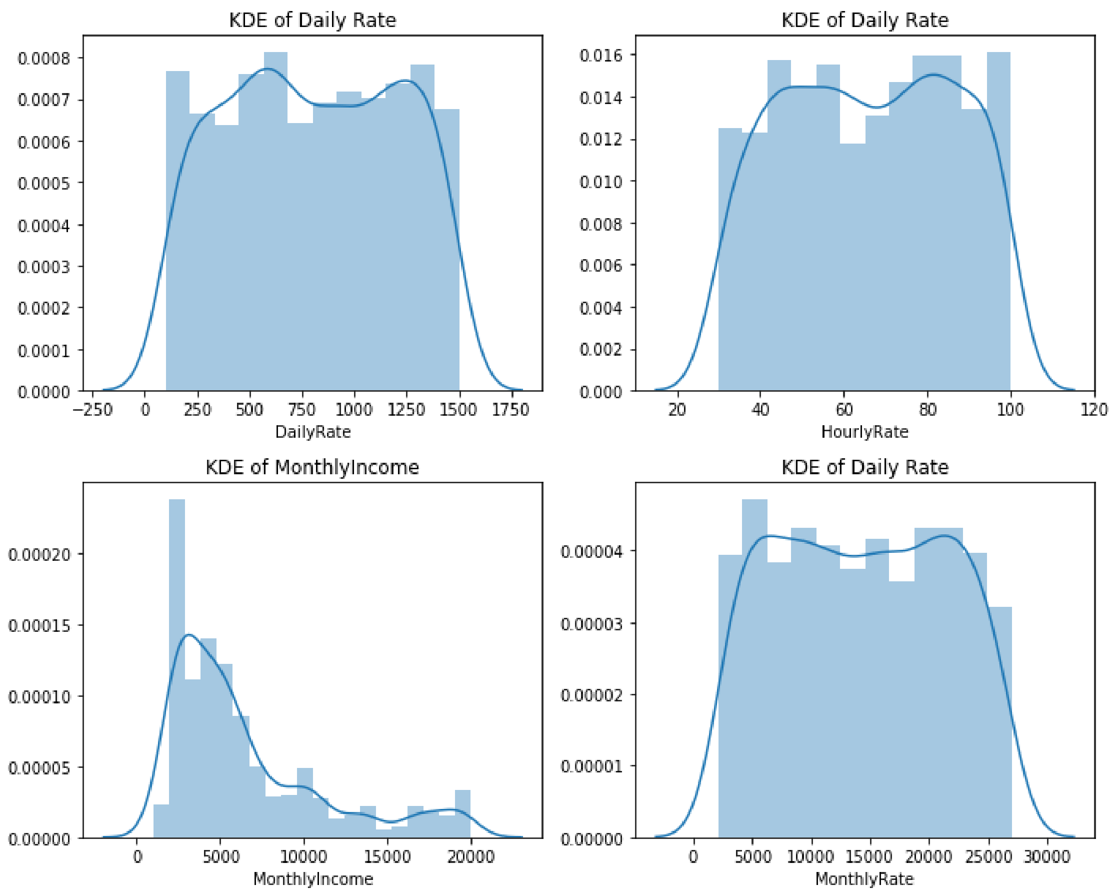
[12]:
```
     TotalWorkingYears  YearsAtCompany  YearsInCurrentRole  \
min                0.0             0.0                 0.0
max               40.0            40.0                18.0

     YearsSinceLastPromotion  YearsWithCurrManager
min                      0.0                   0.0
max                     15.0                  17.0
```

The minimum and maximum values are reasonable for all five columns. Therefore, we can see that we do not have any outliers.

2.2 Distribution check

Before we enter deeper into our analysis, we need to find out the distribution of some numerical attributes (especially attributes regarding income as they are often believed to be skewed across population).

[13]:
```
f, axes = plt.subplots(2, 2, figsize=(10,8))
sns.distplot(df['DailyRate'], ax=axes[0,0]).set_title('KDE of Daily Rate') ␣
↪#seaborn package is used to create a histogram combined with Kernel Density␣
↪Estimation (KDE)
sns.distplot(df['HourlyRate'], ax=axes[0,1]).set_title('KDE of Daily Rate')␣
↪#Neither pandas nor matplotlib can create these two graphs combined in a␣
↪single visualization.
sns.distplot(df['MonthlyIncome'], ax=axes[1,0]).set_title('KDE of␣
↪MonthlyIncome')
sns.distplot(df['MonthlyRate'], ax=axes[1,1]).set_title('KDE of Daily Rate')
plt.tight_layout();
```



From the above plots we realize that the MonthlyIncome is positively skewed. This can be confirmed also by calculating its skewness score:

[14]:
```
df['MonthlyIncome'].skew()
#We will take log transformations if some distributions are skewed to decrease␣
↪the skewness of these variables, which will be beneficial for further␣
↪analysis.
```

[14]:
```
1.3698166808390662
```

In skewed data, the tail region may act as an outlier for the statistical model and we know that outliers adversely affect the model's performance especially in regression-based models. A log transformation can help to fit a very skewed distribution into a Gaussian one. Therefore, the log transformation could be applied to "MonthlyIncome", however, as we are not going to use any linear regression model in this analysis, we omit this transformation.

3. Data Cleaning - Data Manipulation

For aiding our analysis, we performed major cleaning steps and manipulations.

The "EmployeeNumber" is the id of the employee. As every employee has a different employee number we replace the index with the "EmployeeNumber" as this will be more meaningful.

[15]:
```python
df = df.set_index('EmployeeNumber')
```

We change the following variables to categorical ones:

[16]:
```python
print(categorical_columns) #increase
```

```
['Attrition', 'BusinessTravel', 'Department', 'Education', 'EducationField',
'EnvironmentSatisfaction', 'Gender', 'JobInvolvement', 'JobLevel', 'JobRole',
'JobSatisfaction', 'MaritalStatus', 'Over18', 'OverTime', 'PerformanceRating',
'RelationshipSatisfaction', 'StockOptionLevel', 'WorkLifeBalance']
```

Changing the attribute type can potentially increase the execution time and simplifies further our analysis.

[17]:
```python
#It may not reduce the Python runtime significantly in terms of this dataset,
#however, variable type transformation will significantly reduce runtime if
#other datasets are merged with the current dataset and if complex computations
#are required in future analysis.

for col in categorical_columns:
    df[col] = df[col].astype('category')
```

To check if we have correctly assigned the categorical columns we used the .info( ) method:

[18]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1470 entries, 1 to 2068
Data columns (total 34 columns):
Age                         1470 non-null int64
Attrition                   1470 non-null category
BusinessTravel              1470 non-null category
DailyRate                   1470 non-null int64
Department                  1470 non-null category
DistanceFromHome            1470 non-null int64
Education                   1470 non-null category
EducationField              1470 non-null category
EmployeeCount               1470 non-null int64
EnvironmentSatisfaction     1470 non-null category
Gender                      1470 non-null category
HourlyRate                  1470 non-null int64
JobInvolvement              1470 non-null category
JobLevel                    1470 non-null category
JobRole                     1470 non-null category
JobSatisfaction             1470 non-null category
MaritalStatus               1470 non-null category
MonthlyIncome               1470 non-null int64
```

```
MonthlyRate                  1470 non-null int64
NumCompaniesWorked           1470 non-null int64
Over18                       1470 non-null category
OverTime                     1470 non-null category
PercentSalaryHike            1470 non-null int64
PerformanceRating            1470 non-null category
RelationshipSatisfaction     1470 non-null category
StandardHours                1470 non-null int64
StockOptionLevel             1470 non-null category
TotalWorkingYears            1470 non-null int64
TrainingTimesLastYear        1470 non-null int64
WorkLifeBalance              1470 non-null category
YearsAtCompany               1470 non-null int64
YearsInCurrentRole           1470 non-null int64
YearsSinceLastPromotion      1470 non-null int64
YearsWithCurrManager         1470 non-null int64
dtypes: category(18), int64(16)
memory usage: 223.9 KB
```

Since there are some variables whose values are identical across the employees:
- Over18: As all employees are over 18 years old, it will always return "Y" - EmployeeCount: Each employee is counted once, hence it will always return 1
- StandardHours: Since the standard hours (bi-weekly) are 80 for all employees, it will always return 80

We need to drop these columns as they are redundant.

[19]:
```
df.Over18.value_counts()
```

[19]:
```
Y    1470
Name: Over18, dtype: int64
```

[20]:
```
df.EmployeeCount.value_counts()
```

[20]:
```
1    1470
Name: EmployeeCount, dtype: int64
```

[21]:
```
df.StandardHours.value_counts()
```

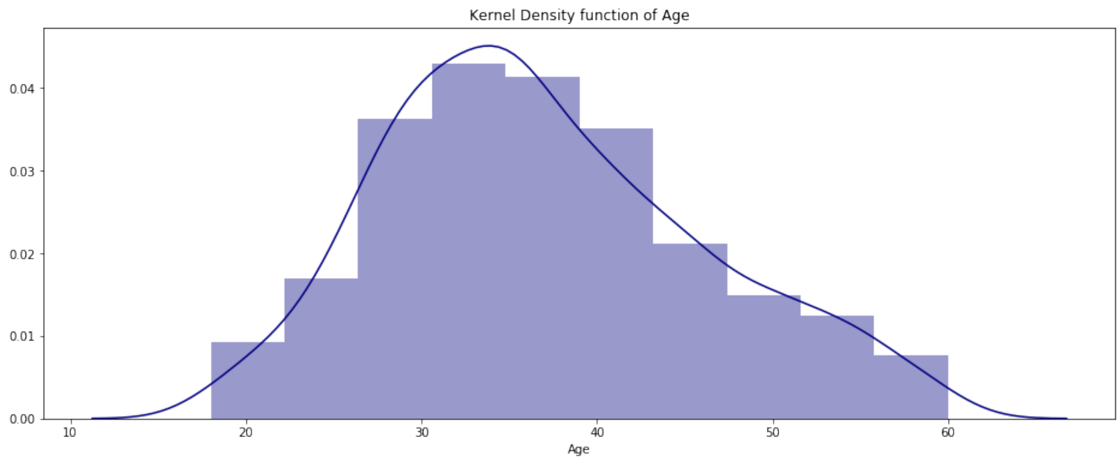
[21]:
```
80    1470
Name: StandardHours, dtype: int64
```

[22]:
```
df.drop(columns=['Over18', 'EmployeeCount', 'StandardHours'], inplace=True)
```

4. Descriptive Analysis
   4.1 Define Age Groups

As our analysis will focus on the Millennials who work in the company, we firstly explore the age attribute:

```
[23]: plt.figure(figsize=(16, 6))
sns.distplot(df['Age'], bins=10, color="navy").set_title('Kernel Density␣
↪function of Age');
```



From the Kernel Density Function, we see that most employees in the company are between 30 and 40 years old. Now we are going to check the attrition rate at different ages.

```
[24]: target_map = {'Yes':1, 'No':0} #we assign '1' to employees who leave the␣
↪company and '0' to employees who stay in the company.
df["Attrition_numerical"] = df["Attrition"].apply(lambda x: target_map[x]) # we␣
↪use the pandas apply method to numerically encode our attrition target␣
↪variable
df = df.drop('Attrition', axis=1)
```

For each age we want to have a glance at the total number of employees who have left the company but also their subsequent proportion.

```
[25]: att_age = pd.crosstab(df.Age, df.Attrition_numerical).apply(lambda r: r/r.
↪sum(), axis=1) #calcualate the percentage
att_age.columns = ['No Attrition Percentage', 'Attrition Percentage']

att_age_count = df.groupby(['Age', "Attrition_numerical"])['Age'].count().
↪transpose() #count number of employees who left the company or not and then␣
↪transpose the table
att_age_count = att_age_count.unstack()
att_age_count.columns = ['No Attrition-Number of Employees', 'Attrition-Number␣
↪of Employees']

att_age = att_age.reset_index().merge(att_age_count, on='Age', how='left').
↪set_index('Age') #we merge these two tables
att_age.iloc[list(range(2,45,5))] #a snapshot of the table
```

[25]:
```
     No Attrition Percentage  Attrition Percentage  \
Age
20                  0.454545              0.545455
25                  0.769231              0.230769
30                  0.850000              0.150000
35                  0.871795              0.128205
40                  0.912281              0.087719
45                  0.951220              0.048780
50                  0.833333              0.166667
55                  0.863636              0.136364
60                  1.000000              0.000000

     No Attrition-Number of Employees  Attrition-Number of Employees
Age
20                                5.0                            6.0
25                               20.0                            6.0
30                               51.0                            9.0
35                               68.0                           10.0
40                               52.0                            5.0
45                               39.0                            2.0
50                               25.0                            5.0
55                               19.0                            3.0
60                                5.0                            NaN
```

We create four bins which correspond to the quantiles of the age observations. The first quantile actually will correspond to the Millennials.

[26]:
```
df.Age.describe()
```

[26]:
```
count    1470.000000
mean       36.923810
std         9.135373
min        18.000000
25%        30.000000
50%        36.000000
75%        43.000000
max        60.000000
Name: Age, dtype: float64
```

[27]:
```
df['Age-quantiles'] = pd.qcut(df['Age'], q=[0, .25, .50, .75, 1],␣
↪labels=['18-30', '31-36', '37-43', '44-60'])
```

Now we replicate the previous table considering the age groups we have defined.

[28]:
```
att_age_bins = pd.crosstab(df['Age-quantiles'], df.Attrition_numerical).
↪apply(lambda r: r/r.sum(), axis=1) #calcualate the percentage
att_age_bins.columns = ['No Attrition Percentage', 'Attrition Percentage']
```

```
att_age_count_bins = df.groupby(['Age-quantiles',␣
 ↪"Attrition_numerical"])['Age-quantiles'].count().transpose().unstack()␣
 ↪#count number of employees who left the company or not and then transpose␣
 ↪the table
att_age_count_bins.columns = ['No Attrition-Number of Employees',␣
 ↪'Attrition-Number of Employees']

att_age_bins = att_age_bins.reset_index().merge(att_age_count_bins,␣
 ↪on='Age-quantiles', how='left').set_index('Age-quantiles') #we merge these␣
 ↪two tables
att_age_bins #a snapshot of the table
```

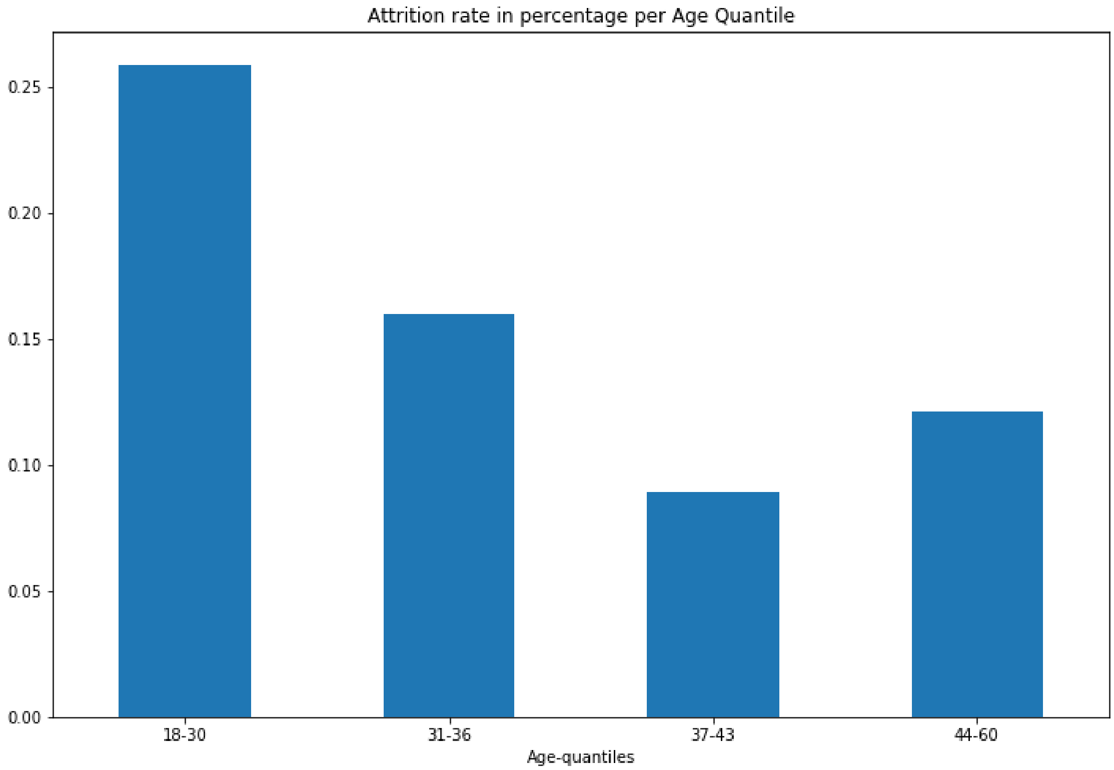
[28]:

```
              No Attrition Percentage  Attrition Percentage  \
Age-quantiles
18-30                        0.740933              0.259067
31-36                        0.839806              0.160194
37-43                        0.910769              0.089231
44-60                        0.878963              0.121037

              No Attrition-Number of Employees  Attrition-Number of Employees
Age-quantiles
18-30                                      286                            100
31-36                                      346                             66
37-43                                      296                             29
44-60                                      305                             42
```

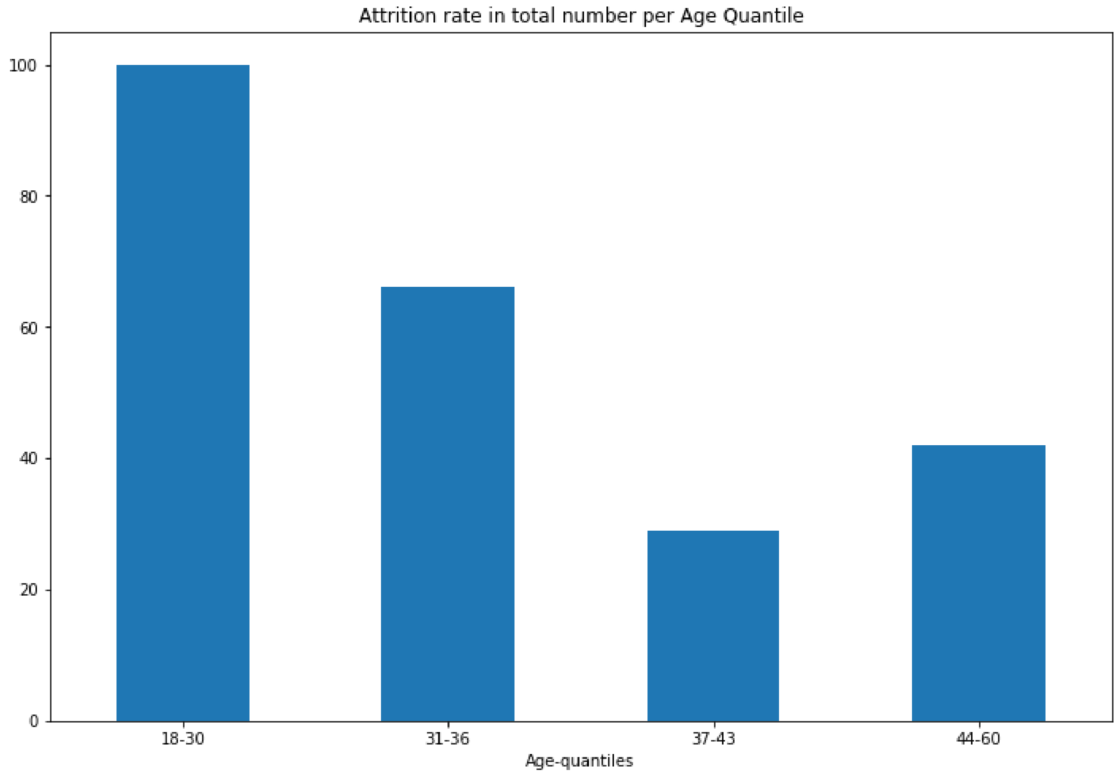
Below, we visualize the percentage and absolute value of attrition for each age group.

[29]:

```
plt.figure(figsize=(12, 8))
att_age_bins['Attrition Percentage'].plot.bar(rot=0, title='Attrition rate in␣
 ↪percentage per Age Quantile'); #attrition percentage across different␣
 ↪quantiles
```



[30]:
```
plt.figure(figsize=(12, 8))
att_age_count_bins['Attrition-Number of Employees'].plot.bar(rot=0,␣
 ↪title='Attrition rate in total number per Age Quantile');
```

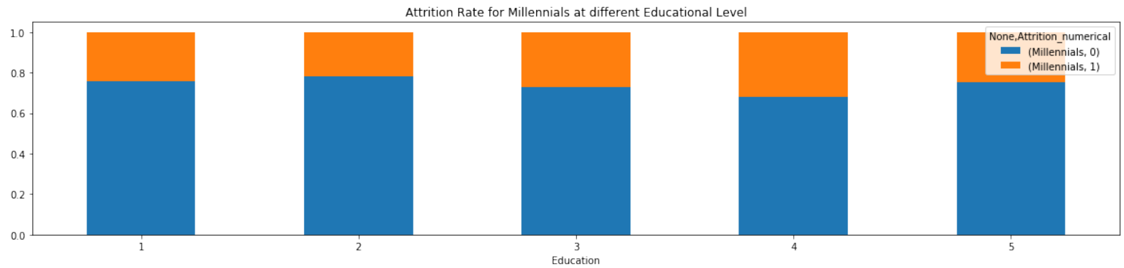



```
[31]: df['Millennials'] = 0 #We define employees who are aged between 18 and 30 to be␣
      ↪Millennials and assign them value of 1.
      df.loc[df['Age-quantiles']=='18-30', 'Millennials'] = 1
```

4.2 Education of Millennials

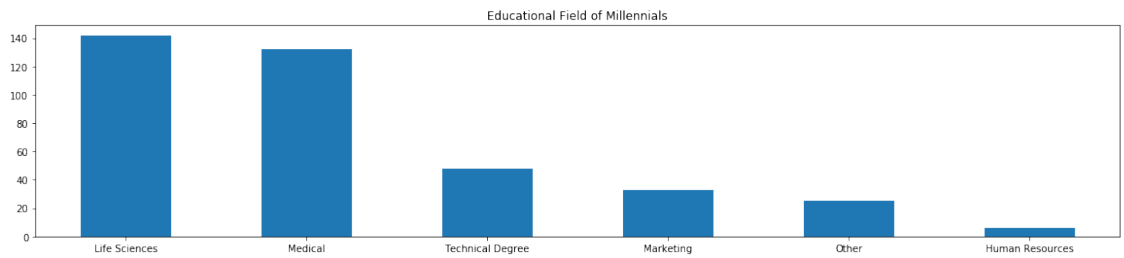
We now look at whether we can find a difference in the attrition rate depending on the education level. This is best to be seen when all education levels are stereteched to 100% and then the attrition rates are compared.

```
[32]: df_educ = df.set_index(['Education', 'Attrition_numerical'])

      tps = df_educ.pivot_table( values=['Millennials'],
                              index='Education',
                              columns='Attrition_numerical',
                              aggfunc='sum')

      tps = tps.div(tps.sum(1), axis=0)
      tps.plot(kind='bar', stacked=True,  rot=0, title='Attrition Rate for␣
      ↪Millennials at different Educational Level');
```



In Education level 4, the attrition rate seems to be relatively higher compared to the other groups, however we could not detect an education group where the attrition rate is way larger than in any other group. Also we want to investigate the field of education from Millennials as shown in the graph below.

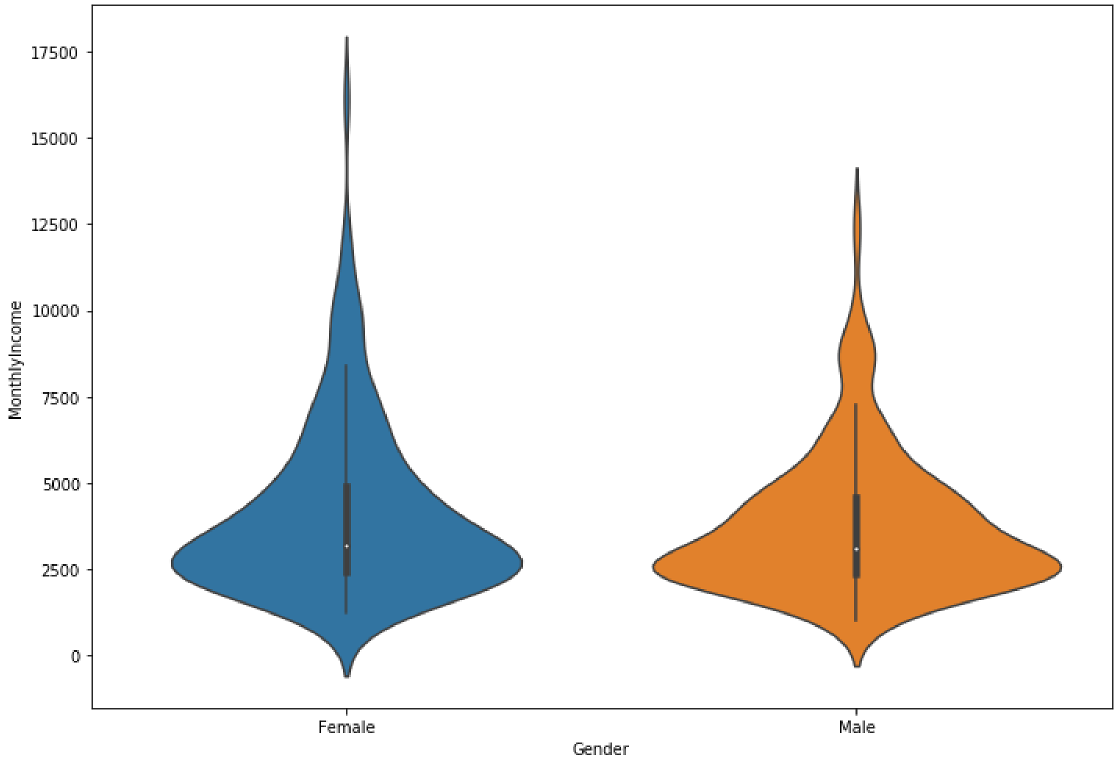
[33]:
```
df.loc[df['Millennials']==1].EducationField.value_counts().plot('bar', rot=0,␣
↪title='Educational Field of Millennials');
```



4.2 The Gender Pay Gap of Millennials

We use a violin plot to describe the relationship between Monthly Income and Gender among Millennials.

[34]:
```
# A violin plot is a method of plotting numeric data which includes a marker␣
↪for the median of the data;
# a box or marker indicating the interquartile range and the distribution of␣
↪the data.
fig, ax = plt.subplots()
fig.set_size_inches(11.7, 8.27)
fig.set_size_inches(11.7, 8.27)
sns.violinplot(x="Gender", y="MonthlyIncome",data=df.loc[df['Millennials']==1]);
```

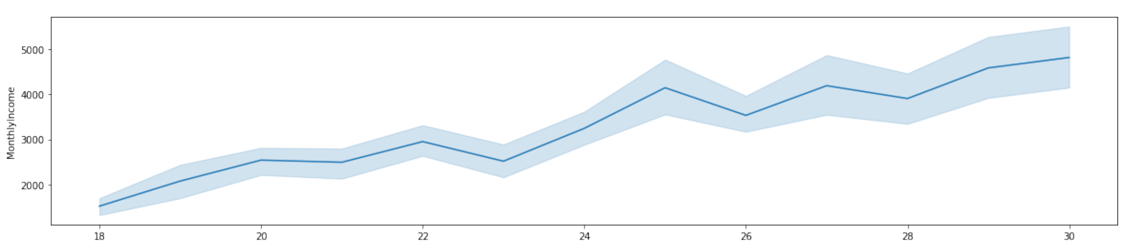


From this plot chart, the distribution of both female and male in the Millennials group are almost the same. They have similar mean values and the same most common monthly income (2500). However, the interquartile range of female is wider than the male's one (from 25% to 75%). Compared to females, the distribution of male concentrates in the bottom, which means male have lower monthly incomes in general. Moreover, the highest income value of female is higher than male.

Therefore, the monthly income situation of females and males are similar in general while there are some females with very high monthly incomes (we can regard them as outliers).

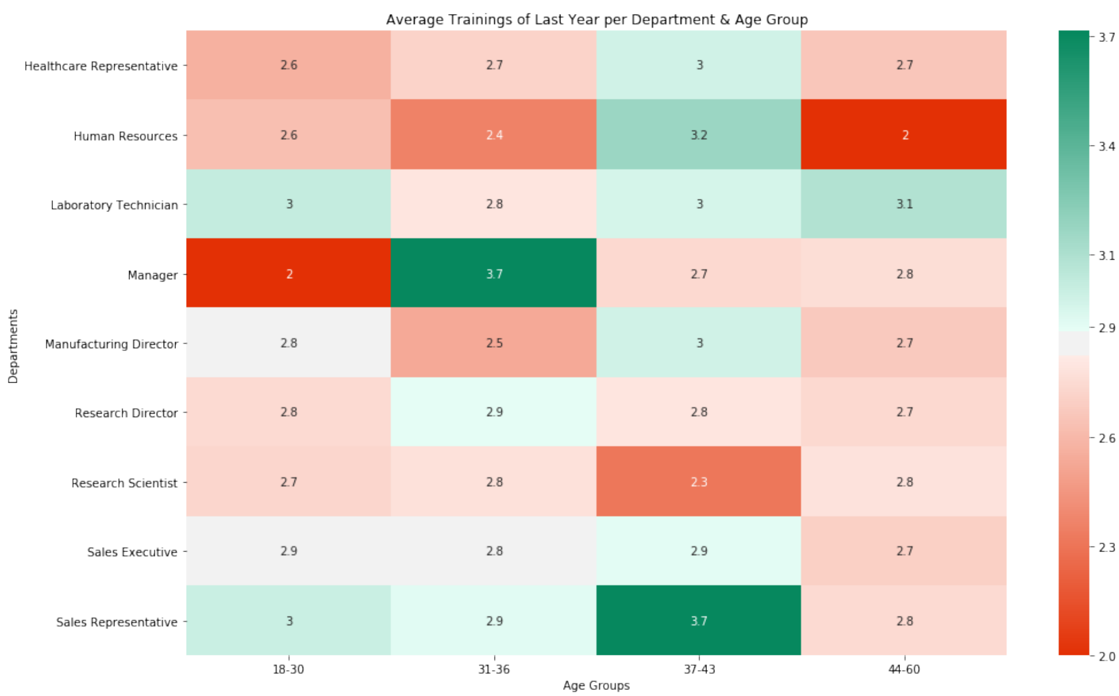
Drawing a line gragh of Millennials' monthly income at different ages, we can see that there is a continuous increasing trend with small fluctuations.

[35]:
```
ax = sns.lineplot(x="Age", y="MonthlyIncome", data=df.loc[df['Millennials']==1])
```



We then created a heat map that includes training days and job role.

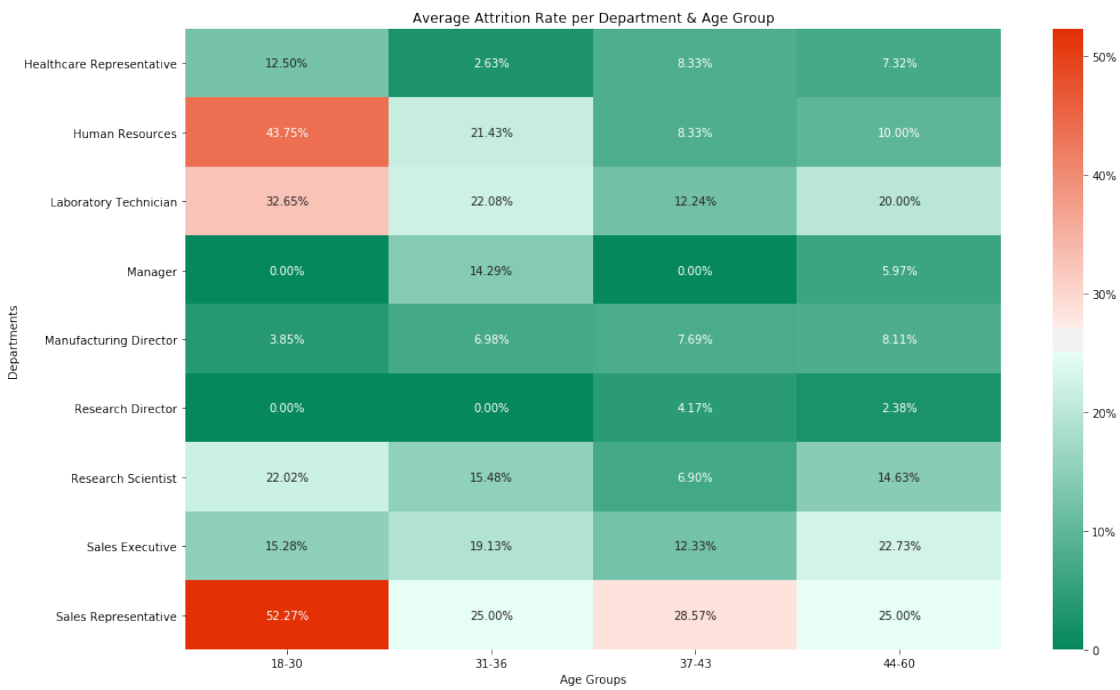
[36]:
```
heat = pd.pivot_table(df.reset_index(), values='TrainingTimesLastYear',␣
 ↪aggfunc=np.mean,index="JobRole", columns='Age-quantiles')
cmap = sns.diverging_palette(15, 150, s=99, l=50, n=10, as_cmap=True)
fig, ax = plt.subplots(figsize=(16,10))
ax = sns.heatmap(heat, cmap=cmap, annot=True)
bottom, top = ax.get_ylim() #fix bug "matplotlib/seaborn: first and last row␣
 ↪cut in half of heatmap plot" ref: https://stackoverflow.com/a/58165593
ax.set_ylim(bottom + 0.5, top - 0.5)
ax.set_title('Average Trainings of Last Year per Department & Age Group')
ax.set(xlabel='Age Groups', ylabel='Departments');
ax = ax.collections[0].colorbar
ax.set_ticks([2.0, 2.3, 2.6, 2.9, 3.1 , 3.4, 3.7])
#pd.pivot_table(df, values = 'Value', index=['Country','Year'], columns =␣
 ↪'Indicator').reset_index()
```



Then we created a heatmap which combines job roles, age groups and attrition rate. We found out that Millennials who are sales representatives have an exceptionally high attrition rate, followed by Millennials who are members of the Human Resources. For non-millennial age groups, there is no significant difference in attrition rate across different job roles.

[37]:
```
df['Attrition_numerical'] = df['Attrition_numerical'].astype(int)
heat = pd.pivot_table(df.reset_index(), values='Attrition_numerical',␣
 ↪aggfunc=np.mean,index="JobRole", columns='Age-quantiles')
```

```
cmap = sns.diverging_palette(150, 15, s=99, l=50, n=10, as_cmap=True)
fig, ax = plt.subplots(figsize=(16,10))
ax = sns.heatmap(heat, cmap=cmap, annot=True, fmt=".2%")
bottom, top = ax.get_ylim() #fix bug "matplotlib/seaborn: first and last row␣
↪cut in half of heatmap plot" ref: https://stackoverflow.com/a/58165593
ax.set_ylim(bottom + 0.5, top - 0.5)
ax.set_title('Average Attrition Rate per Department & Age Group')
ax.set(xlabel='Age Groups', ylabel='Departments')
ax = ax.collections[0].colorbar
ax.set_ticks([0, .1, .2, .3, .4, .5])
ax.set_ticklabels(['0', '10%', '20%', '30%', '40%', '50%']);

df['Attrition_numerical'] = df['Attrition_numerical'].astype('category')
```



By comparing the two heatmaps, we found out that Millennials who take HR roles receive less training, and they have a relatively high attrition rate.

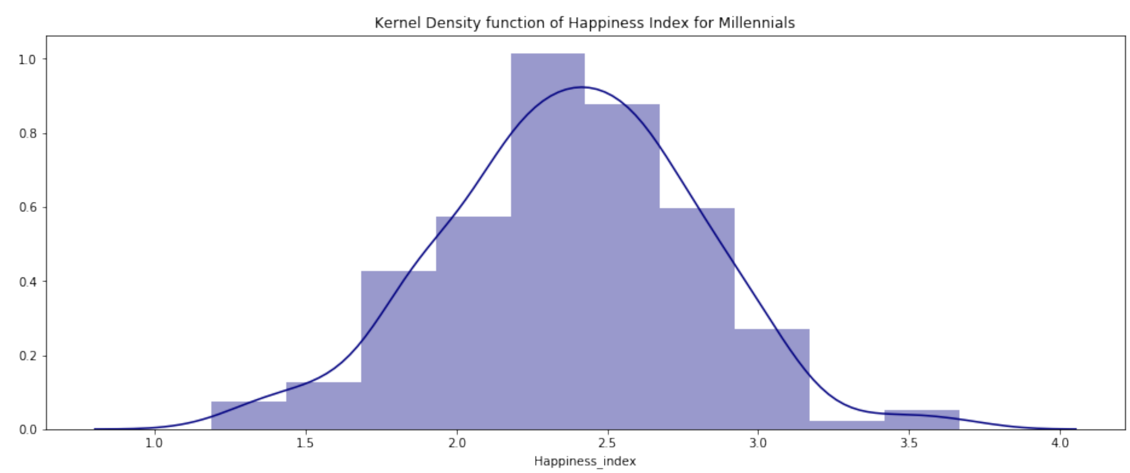
5. Featuring Engineering
   In this section we are going to create four new features based on our existing ones.

[38]:
```
#We create a ratio which keeps the Job Satisfaction Level divided by the␣
↪Distance From Home for every employee
df['Commute_JobSatisfaction_Ratio'] = df['JobSatisfaction'].astype(int) /␣
↪df['DistanceFromHome'].astype(int)
```

```
#We create a Job Fullfillment attribute which get the average from Job Satisfaction and Job Involvement
df['JobFulfillment'] = (df['JobSatisfaction'].astype(int) + df['JobInvolvement'].astype(int)) / 2

#We create a Loyalty Index attribute which is a ratio of Number of Companies Worked divided by the Total Working Years
df['Loyalty_index'] = df['NumCompaniesWorked'].astype(int) / df['TotalWorkingYears'].astype(int)

#We create a Happiness Index which summarizes factors that could affect employee's satisfaction
df['Happiness_index'] = (df['RelationshipSatisfaction'].astype(int)  + df['EnvironmentSatisfaction'].astype(int) + df['JobSatisfaction'].astype(int)
                                  + df['JobInvolvement'].astype(int) + df['WorkLifeBalance'].astype(int) + df['Commute_JobSatisfaction_Ratio'])/6
```

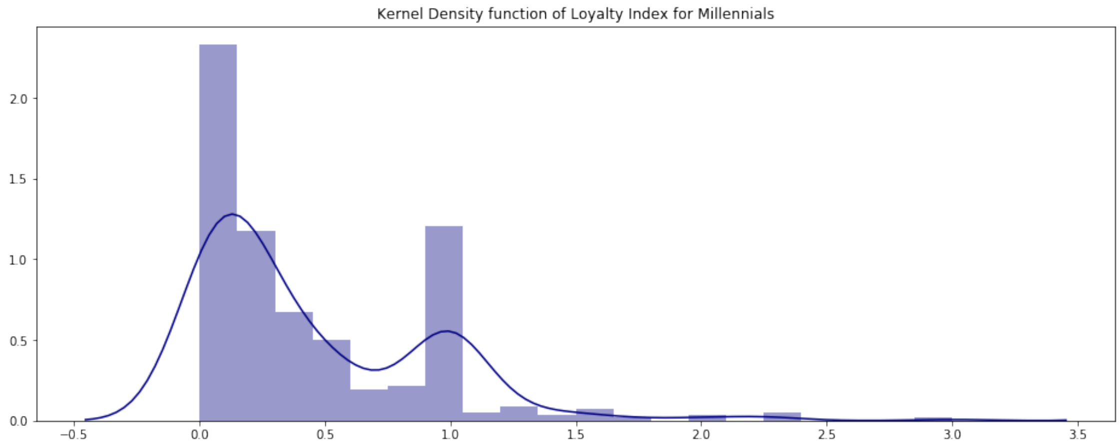
Now we draw two graphs - for the distribution of both Happiness Index and Loyalty Index.

[39]:
```
plt.figure(figsize=(16, 6))
sns.distplot(df.loc[df['Millennials']==1 , 'Happiness_index'], bins=10, color="navy").set_title('Kernel Density function of Happiness Index for Millennials');
```



[40]:
```
plt.figure(figsize=(16, 6))

loyalty_index = df[np.isfinite(df['Loyalty_index']) ]
loyalty_index = loyalty_index.loc[loyalty_index['Millennials']==1, ['Loyalty_index']]
loyalty_index
```

```
sns.distplot(loyalty_index, bins=20, color="navy").set_title('Kernel Density␣
↪function of Loyalty Index for Millennials');
```



6. Predictive Analysis
   Now we demonstrate how we can predict Millennial employee's turnover

[41]:
```
dfp = df.copy() #copy of the original DF because we are going to preprocess the␣
↪features for the prediction
```

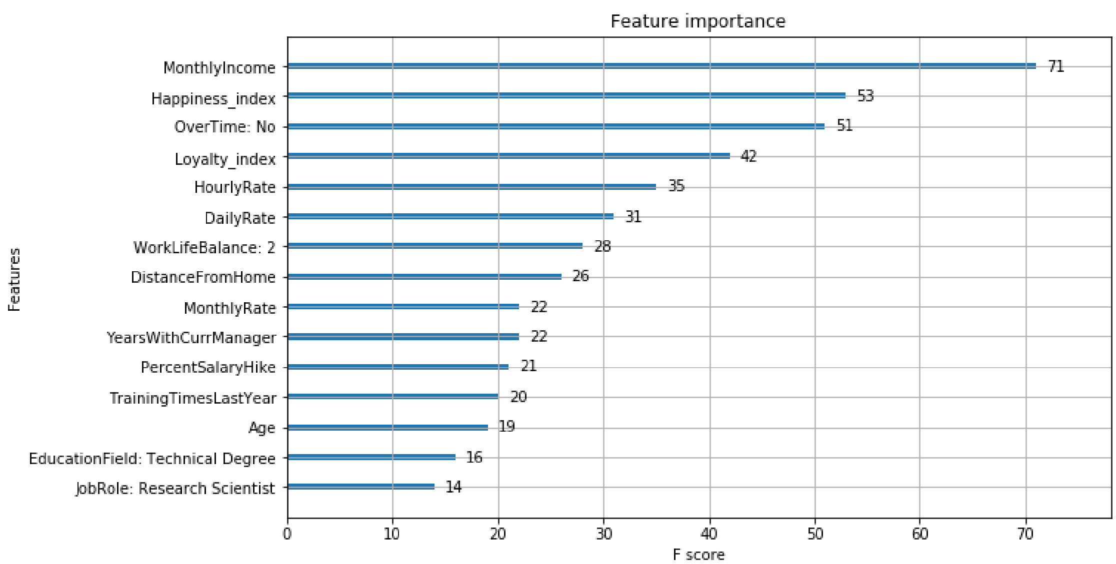
First we preprocess the data.

[42]:
```
specs_to_dummies = list(dfp.select_dtypes(['object','category']).columns)␣
↪#categorical variables into dummies
specs_to_dummies.remove('Attrition_numerical') #we exclude the response from␣
↪converting

for item in specs_to_dummies:
    dummies = pd.get_dummies(df[item], prefix_sep=': ', prefix=item)
    dfp = pd.concat([dfp, dummies], sort=False, axis=1)

dfp = dfp.drop(specs_to_dummies, axis=1).copy()
dfp = dfp.loc[dfp.Millennials==1]
#dfp.head()
```

Then we split the data into training and testing subsets and we create a predictive model.

[43]:
```
X_train, X_test, y_train, y_test = train_test_split(dfp.
↪drop("Attrition_numerical", axis=1), dfp["Attrition_numerical"], test_size=0.
↪30, random_state=42)

#Unbalanced dataset 286 remained in the company and 100 left the company.
```

```
#We need to adjust this in our training algorithm. ref: https://xgboost.
↪readthedocs.io/en/latest/parameter.html

scale_pos_weight = dfp.loc[(dfp['Attrition_numerical']==0) &␣
↪dfp['Millennials']==1].shape[0] / dfp.loc[(dfp['Attrition_numerical']==0) &␣
↪dfp['Millennials']==1].shape[0]

model = xgb.XGBClassifier(objective="reg:logistic",␣
↪scale_pos_weight=scale_pos_weight, scoring='recall')
#we use scoring='recall' as we believe is the most important metric for our␣
↪business problem

model.fit(X_train, y_train)
fig, ax = plt.subplots(figsize=(10,6))
xgb.plot_importance(model, max_num_features = 15, ax=ax);
```

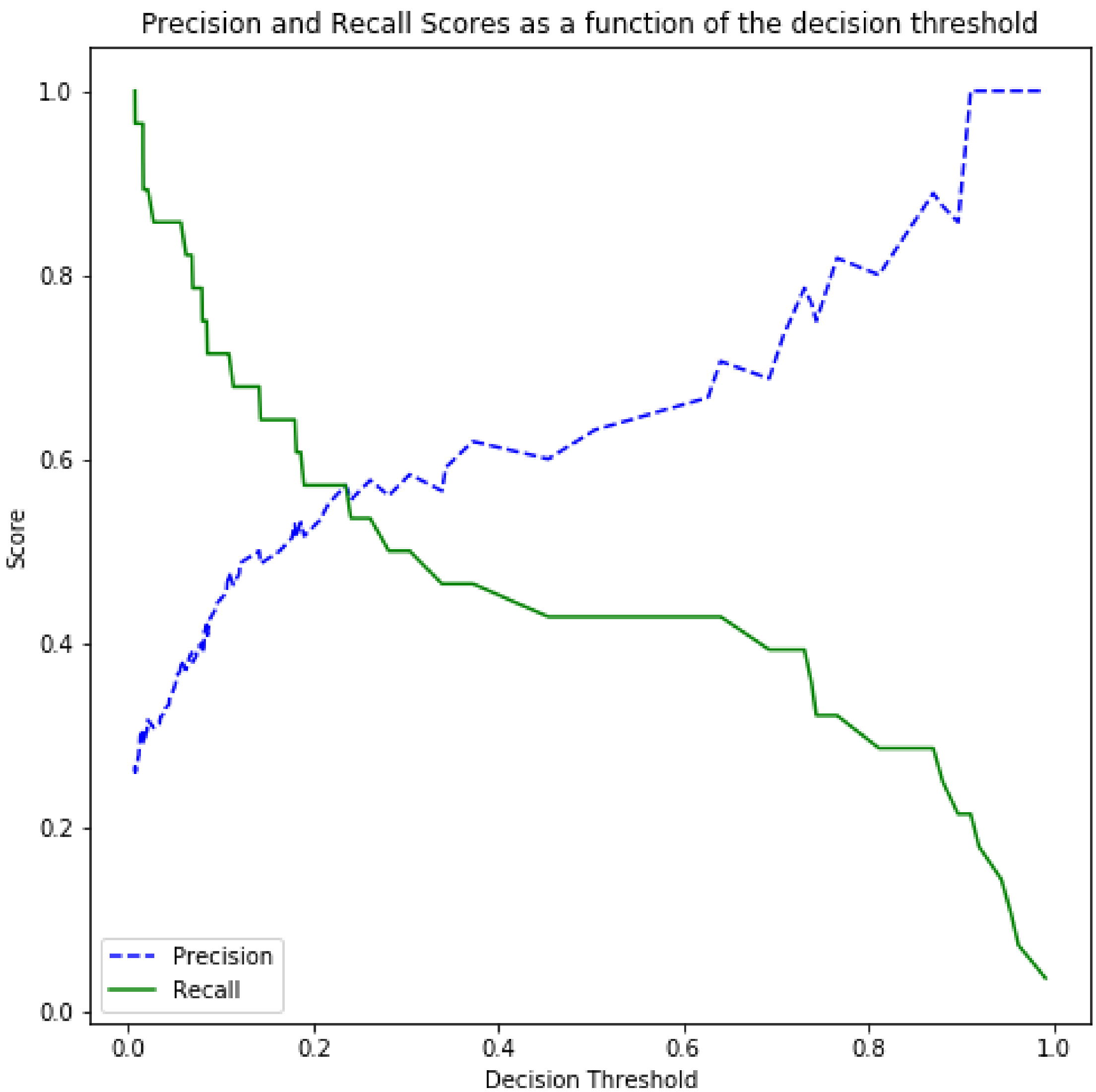


In the produced plot we get a ranking of the features that have been to be the most important for predicting attrition.

We use the Recall metric which focuses on predicting well the employees who will leave the company (although it might predict erroneously that some people will leave the company while indeed they will not). Below we demonstrate how a different threshold, how easily we predict that someone will leave the company, actually affects the Recall:

[44]:
```
y_pred_prob = model.predict_proba(X_test)[:,1]

p, r, thresholds = precision_recall_curve(y_test,y_pred_prob)
```

```
#https://towardsdatascience.com/
↪fine-tuning-a-classifier-in-scikit-learn-66e048c21e65
def plot_precision_recall_vs_threshold(precisions, recalls, thresholds):
    """
    Modified from:
    Hands-On Machine learning with Scikit-Learn
    and TensorFlow; p.89
    """
    plt.figure(figsize=(8, 8))
    plt.title("Precision and Recall Scores as a function of the decision␣
↪threshold")
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    plt.ylabel("Score")
    plt.xlabel("Decision Threshold")
    plt.legend(loc='best')

plot_precision_recall_vs_threshold(p, r, thresholds) #the plot demonstrates for␣
↪different theshold how precision and recall varies
#threshold actually is the cut-off point for classifying an employee if they␣
↪will churn or not.
```



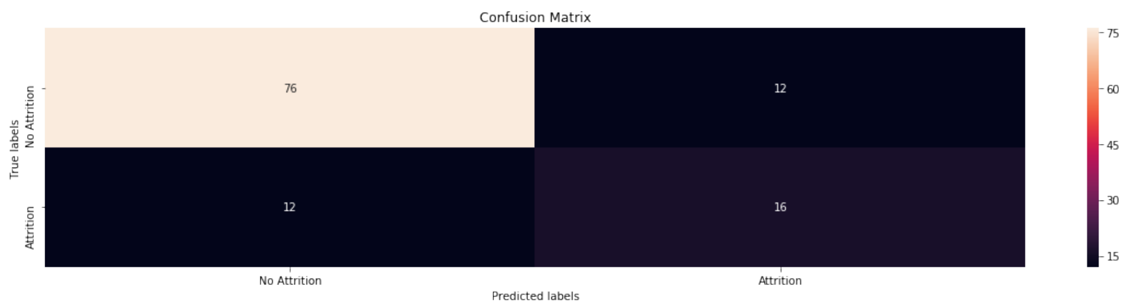
[45]:
```python
#for our final prediction model, we adjust the threshold to 0.22 to increase␣
↪the recall which is our major objective
#This means that if an employee has a probability to leave company above 0.22,␣
↪then it will be classified as attrition
y_pred_class = (model.predict_proba(X_test)[:,1] >= 0.22).astype(bool)

print(classification_report(y_test,y_pred_class))
```

```
              precision    recall  f1-score   support

           0       0.86      0.86      0.86        88
           1       0.57      0.57      0.57        28
```

```
    accuracy                           0.79       116
   macro avg       0.72      0.72      0.72       116
weighted avg       0.79      0.79      0.79       116
```

The confusion matrix below displays the accuracy of the prediction (i.e. How many employees we predicted to leave (in total 12+16=28) and how many left indeed(16) as well as how many we predicted to remain in the company (76+12=88) and how many remained indeed(76).

[46]:
```python
cm = confusion_matrix(y_test, y_pred_class)

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax, fmt='g'); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['No Attrition', 'Attrition']); ax.yaxis.set_ticklabels(['No Attrition', 'Attrition'])
bottom, top = ax.get_ylim() #fix bug "matplotlib/seaborn: first and last row cut in half of heatmap plot" ref: https://stackoverflow.com/a/58165593
ax.set_ylim(bottom + 0.5, top - 0.5);
```



7. Conclusion and business actions
   As our analysis is based on Millennials exclusively, we would like to give out recommendations on how to potentially lower the attrition rates for the subsequent age group. We firstly would like to draw attention to the fact that there are different attrition rates across different departments. We therefore suggest the company to take a closer look into the HR department and the Sales Representatives as they show the highest attrition rates (52% and 43%). As in general we assume that attrition rates could be lowered with higher number of training days we also investigated this variable for each age group and department. We cannot draw a causal inference from our model, however based on our results we suggest increasing the number of training days for Millennials in the HR department.

With our model we identified the most important features with the highest predictive abilitz on attrition rate, and we found that MonthlyIncome, Happiness_index, Overtime:no and Loyalty_index had the highest impact on an employees' churn. We therefore suggest increasing the income for

Millennials as well as making sure they are satisfied with their workplace and they do not work too much. We have also found out that employees who historically have had a lower loyalty to their workplace are more likely to churn in the future as well – this should be taken into account when selecting candidates for a newly opened position.

- Limitations

As we were particularly interested in the attrition of workforce aged 18-30 years old, we had to base our model on this group to obtain interesting insights. Unfortunately, the dataset is mainly comprised of people outside of this age range therefore we had to work with a rather small number of observations for our predictive model. It is also worth mentioning that we are working with a fictional dataset. The group of the Millennials is therefore based on the assumption that this dataset is from this year (2019).

- Suggestions for improvements

Firstly, we could focus more on feature engineering and conduct more analysis regarding Millennial's' attrition rate. For example, we could find more interactions among different attributes. Secondly, we could compare different predictive algorithms for hyperparameter tuning.

Appendix

JIRA

Although this is the first time that we use JIRA as our project management tool, we get used to this software quickly and we use it as the primary tool when we have a meeting. We firstly assign tickets to different group members so that every group member is fully aware of his or her duty. The Pie Chart Report of Assignee provides an intuitive overview of the task allocation. Then we use JIRA to monitor the progress of each group member. More importantly, we can have an overview of the progress of the whole project from the Cumulative Flow Diagram. Overall, JIRA makes a great contribution to the completion of our project.

References

Bannon, S., Ford, K. and Meltzer, L. (2011). Understanding Millennials in the Workplace. The CPA Journal, [online] 81(11), pp.61-65. Available at: https://search.proquest.com/docview/922065831/abstract/20271DCF78D54B2EPQ/1?accountid=14511.

Incentius.com. (2019). Incentius - Innovative cloud-enabled business intelligent platforms and solutions developed using secure and scalable technologies for your enterprise.. [online] Available at: https://incentius.com/blog-posts/analyzing-attrition-performance/ [Accessed 5 Nov. 2019].

KPMG (2017). Meet the Millennials. [online] KPMG, pp.7-20. Available at: https://home.kpmg/content/dam/kpmg/uk/pdf/2017/04/Meet-the-Millennials-Secured.pdf.

Medium. (2019). Ways to Detect and Remove the Outliers. [online] Available at: https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608dba [Accessed 11 Dec. 2019].