Human Resource Department Case-Study

This case study is to help users understand how to leverage the power of data science to reduce employee turnover and transform human resource department. **Problem:** Hiring and retraining employees are time and resource consuming tasks. Often time, a company may spend 15-20% of the an employees salary to train the new recruit and spend 40% of their working horus on hiring, which does not generate any income.

In [ ]:
```
# imports
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [ ]:
```
# connecting to database
employee_dataframe = pd.read_csv('Human_Resources.csv')
pd.set_option('display.max_columns',None)
employee_dataframe.head(5)
```

Out[ ]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumbe
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life Sciences	1	1
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life Sciences	1	2
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	Other	1	4
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life Sciences	1	5
4	27	No	Travel_Rarely	591	Research & Development	2	1	Medical	1	7

Analyzing Database columns:

Age: This could be a factor as older employees who have families, mortgages may not be reluctant to leave the job, however this depends on other factors provided in this table.

Distance From Home: Further away from the company may be a factor of attrition.

Relationship/Job/Employement Saticfaction: An satisfied employee who also get along with everyone and liked by others may not be inclined to leave a company, such as promotion, years worked at the company, years with current manager etc.

Work Life Balance:Is the employee overworked?

Hourly rate/monthly income/ stock options: To check if the employee is getting paid properly proportionally to their job role and marital status.

In [ ]:
```
employee_dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Age                       1470 non-null   int64 
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64 
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64 
 6   Education                 1470 non-null   int64 
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64 
 9   EmployeeNumber            1470 non-null   int64 
 10  EnvironmentSatisfaction   1470 non-null   int64 
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64 
 13  JobInvolvement            1470 non-null   int64 
 14  JobLevel                  1470 non-null   int64 
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64 
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64 
 19  MonthlyRate               1470 non-null   int64 
 20  NumCompaniesWorked        1470 non-null   int64 
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64 
 24  PerformanceRating         1470 non-null   int64 
 25  RelationshipSatisfaction  1470 non-null   int64 
 26  StandardHours             1470 non-null   int64 
 27  StockOptionLevel          1470 non-null   int64 
 28  TotalWorkingYears         1470 non-null   int64 
 29  TrainingTimesLastYear     1470 non-null   int64 
 30  WorkLifeBalance           1470 non-null   int64 
 31  YearsAtCompany            1470 non-null   int64 
 32  YearsInCurrentRole        1470 non-null   int64 
 33  YearsSinceLastPromotion   1470 non-null   int64 
 34  YearsWithCurrManager      1470 non-null   int64 
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

In [ ]:
```
employee_dataframe.describe()
```

Out[ ]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	EmployeeNumber	EnvironmentSatisfaction	HourlyRate
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	1470.000000	1470.000000	1470.000000
mean	36.923810	802.485714	9.192517	2.912925	1.0	1024.865306	2.721769	65.891156
std	9.135373	403.509100	8.106864	1.024165	0.0	602.024335	1.093082	20.329428
min	18.000000	102.000000	1.000000	1.000000	1.0	1.000000	1.000000	30.000000
25%	30.000000	465.000000	2.000000	2.000000	1.0	491.250000	2.000000	48.000000
50%	36.000000	802.000000	7.000000	3.000000	1.0	1020.500000	3.000000	66.000000
75%	43.000000	1157.000000	14.000000	4.000000	1.0	1555.750000	4.000000	83.750000
max	60.000000	1499.000000	29.000000	5.000000	1.0	2068.000000	4.000000	100.000000

Changing Attritition, Overtime, and Over18 columns from string (Yes/No) to int so it can be easier to visualize

In [ ]:

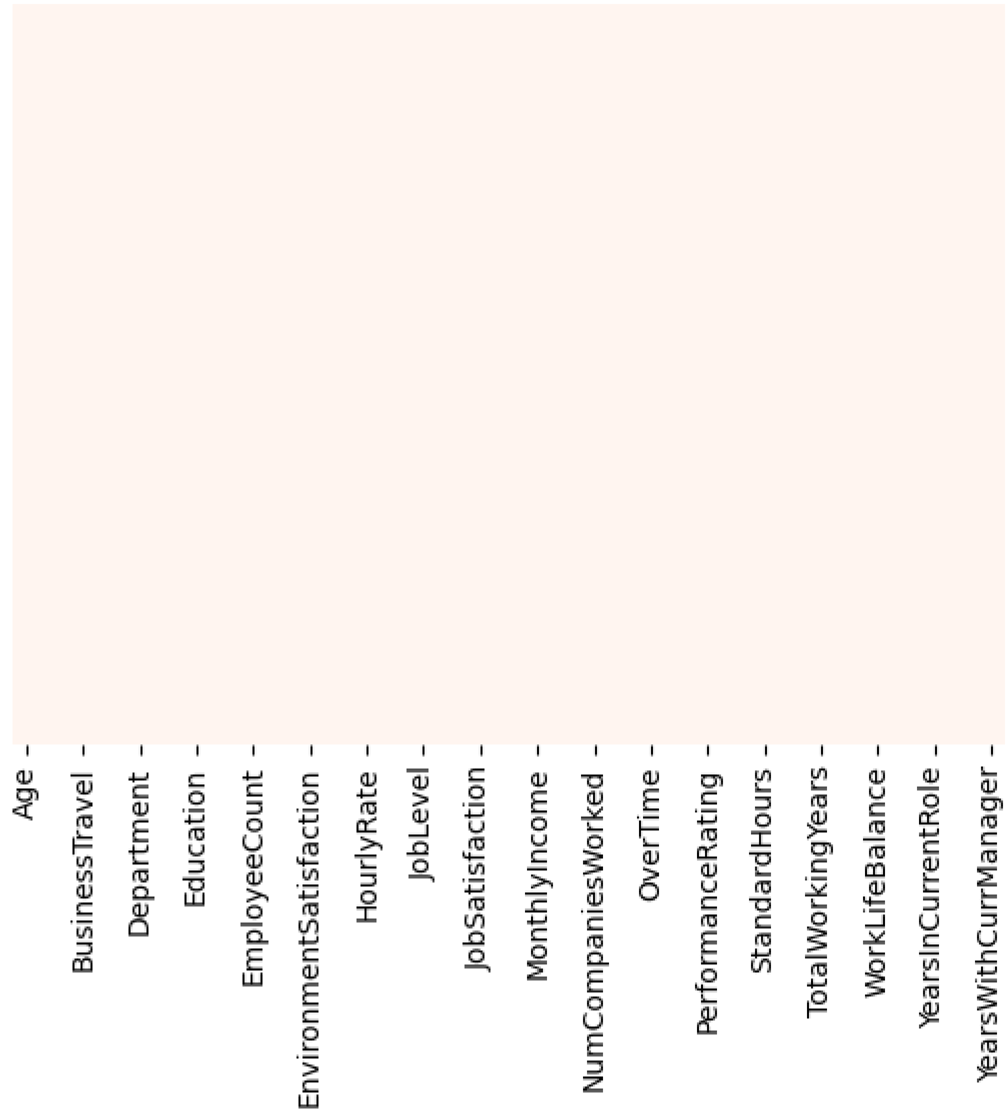
```
# modifying attritition column
employee_dataframe['Attrition'] = employee_dataframe['Attrition'].apply(lambda x:1 if x == "Yes" else 0)
# mofidying Overtime column
employee_dataframe['OverTime'] = employee_dataframe['OverTime'].apply(lambda x:1 if x == "Yes" else 0)
# modifying Over18 column
employee_dataframe['Over18'] = employee_dataframe['Over18'].apply(lambda x:1 if x == "Y" else 0)
```

In [ ]:

```
# Checking for missing data such as null, using Seaborn heat map
sns.heatmap(employee_dataframe.isnull(), yticklabels=False, cbar=False, cmap='Reds')
```

Out[ ]:

```
<AxesSubplot:>
```

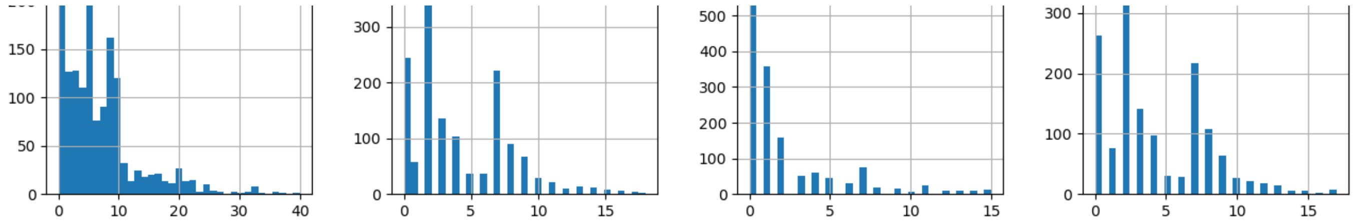


From the heatmap, we can see that there are no null values present in the dataframe. Now we can possibly plot a histogram to visualize each of the attributes.

In [ ]:
```
employee_dataframe.hist(bins=35,
                        figsize=(20,20),
                        )
```

Out[ ]:
```
array([[<AxesSubplot:title={'center':'Age'}>,
        <AxesSubplot:title={'center':'Attrition'}>,
        <AxesSubplot:title={'center':'DailyRate'}>,
        <AxesSubplot:title={'center':'DistanceFromHome'}>,
        <AxesSubplot:title={'center':'Education'}>],
       [<AxesSubplot:title={'center':'EmployeeCount'}>,
        <AxesSubplot:title={'center':'EmployeeNumber'}>,
        <AxesSubplot:title={'center':'EnvironmentSatisfaction'}>,
        <AxesSubplot:title={'center':'HourlyRate'}>,
        <AxesSubplot:title={'center':'JobInvolvement'}>],
       [<AxesSubplot:title={'center':'JobLevel'}>,
        <AxesSubplot:title={'center':'JobSatisfaction'}>,
        <AxesSubplot:title={'center':'MonthlyIncome'}>,
        <AxesSubplot:title={'center':'MonthlyRate'}>,
        <AxesSubplot:title={'center':'NumCompaniesWorked'}>],
       [<AxesSubplot:title={'center':'Over18'}>,
        <AxesSubplot:title={'center':'OverTime'}>,
        <AxesSubplot:title={'center':'PercentSalaryHike'}>,
        <AxesSubplot:title={'center':'PerformanceRating'}>,
        <AxesSubplot:title={'center':'RelationshipSatisfaction'}>],
       [<AxesSubplot:title={'center':'StandardHours'}>,
        <AxesSubplot:title={'center':'StockOptionLevel'}>,
        <AxesSubplot:title={'center':'TotalWorkingYears'}>,
        <AxesSubplot:title={'center':'TrainingTimesLastYear'}>,
        <AxesSubplot:title={'center':'WorkLifeBalance'}>],
       [<AxesSubplot:title={'center':'YearsAtCompany'}>,
        <AxesSubplot:title={'center':'YearsInCurrentRole'}>,
        <AxesSubplot:title={'center':'YearsSinceLastPromotion'}>,
        <AxesSubplot:title={'center':'YearsWithCurrManager'}>,
        <AxesSubplot:>]], dtype=object)
```


Age
Attrition
DailyRate
DistanceFromHome
Education
EmployeeCount
EmployeeNumber
EnvironmentSatisfaction
HourlyRate
JobInvolvement
JobLevel
JobSatisfaction
MonthlyIncome
MonthlyRate
NumCompaniesWorked
Over18
OverTime
PercentSalaryHike
PerformanceRating
RelationshipSatisfaction
StandardHours
StockOptionLevel
TotalWorkingYears
TrainingTimesLastYear
WorkLifeBalance
YearsAtCompany
YearsInCurrentRole
YearsSinceLastPromotion
YearsWithCurrManager



By looking at the chats

Monthly income, it is very tail-heavy as lot of employee gets paid between 0-5000 per month.

Salary hike, it is also very tail-heavy, a very common % increase is around 15% and 20-25% is at a low percentage.

EmployeeCount, EmployeeNumber, StandardHours, and Over18 chat does not contrubute to finding the solution as they all have one single value, so they can be dropped.

MonthlyRate is irelevent as we have hourly rate and monthly income

Once it is completed, we can check the Attrition table to see details about employees who has left the company.

In [ ]:
```
employee_dataframe.drop(['EmployeeCount','EmployeeNumber','StandardHours','Over18','MonthlyRate','DailyRate'], axis=1,
```

In [ ]:
```
# Breaking down the dataframe into two: Employees who have left, and employees who have stayed
left_df = employee_dataframe[employee_dataframe['Attrition'] == 1]
stayed_df = employee_dataframe[employee_dataframe['Attrition'] == 0]
```

In [ ]:
```
# Analyzing data for employees who have left
print('Total employees left: ', len(left_df))
print(len(left_df)/len(employee_dataframe) * 100, '% of the employee has left')
left_df.describe()
```

```
Total employees left:  237
16.122448979591837 % of the employee has left
```

Out[ ]:

	Age	Attrition	DistanceFromHome	Education	EnvironmentSatisfaction	HourlyRate	JobInvolvement	JobLevel	JobSatisfactio
count	237.000000	237.0	237.000000	237.000000	237.000000	237.000000	237.000000	237.000000	237.00000
mean	33.607595	1.0	10.632911	2.839662	2.464135	65.573840	2.518987	1.637131	2.46835
std	9.689350	0.0	8.452525	1.008244	1.169791	20.099958	0.773405	0.940594	1.11805
min	18.000000	1.0	1.000000	1.000000	1.000000	31.000000	1.000000	1.000000	1.00000
25%	28.000000	1.0	3.000000	2.000000	1.000000	50.000000	2.000000	1.000000	1.00000
50%	32.000000	1.0	9.000000	3.000000	3.000000	66.000000	3.000000	1.000000	3.00000
75%	39.000000	1.0	17.000000	4.000000	4.000000	84.000000	3.000000	2.000000	3.00000
max	58.000000	1.0	29.000000	5.000000	4.000000	100.000000	4.000000	5.000000	4.00000

In [ ]:

```
#Analyzing data for employees who have stayed
print('Total employees stayed: ', len(stayed_df))
print(len(stayed_df)/len(employee_dataframe) * 100, '% of the employee has stayed')
stayed_df.describe()
```

```
Total employees stayed:  1233
83.87755102040816 % of the employee has stayed
```

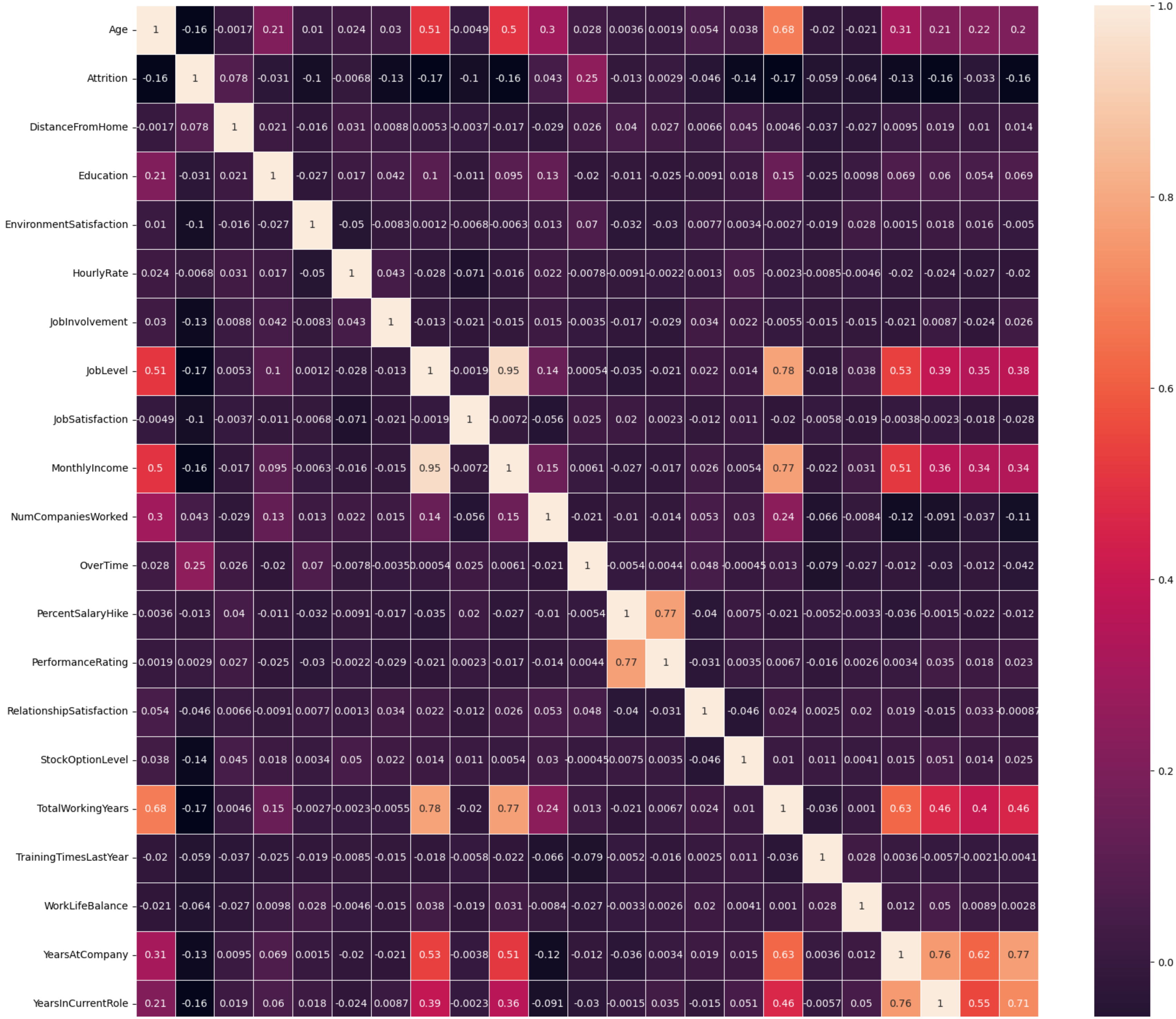
Out[ ]:

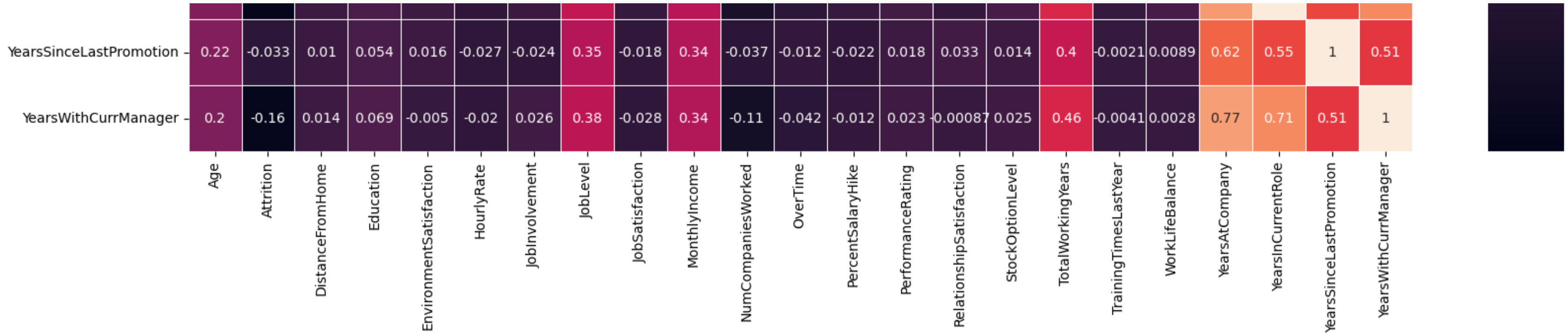
	Age	Attrition	DistanceFromHome	Education	EnvironmentSatisfaction	HourlyRate	JobInvolvement	JobLevel	JobSatisfa
count	1233.000000	1233.0	1233.000000	1233.000000	1233.000000	1233.000000	1233.000000	1233.000000	1233.0
mean	37.561233	0.0	8.915653	2.927007	2.771290	65.952149	2.770479	2.145985	2.7
std	8.888360	0.0	8.012633	1.027002	1.071132	20.380754	0.692050	1.117933	1.0
min	18.000000	0.0	1.000000	1.000000	1.000000	30.000000	1.000000	1.000000	1.0
25%	31.000000	0.0	2.000000	2.000000	2.000000	48.000000	2.000000	1.000000	2.0
50%	36.000000	0.0	7.000000	3.000000	3.000000	66.000000	3.000000	2.000000	3.0
75%	43.000000	0.0	13.000000	4.000000	4.000000	83.000000	3.000000	3.000000	4.0
max	60.000000	0.0	29.000000	5.000000	4.000000	100.000000	4.000000	5.000000	4.0

In [ ]:

```
correlations = employee_dataframe.corr()
fig, ax = plt.subplots(figsize=(20,20))
sns.heatmap(correlations, annot=True, linewidths=0.5)
```

Out[ ]: <AxesSubplot:>


Age 1 -0.16 -0.0017 0.21 0.01 0.024 0.03 0.51 -0.0049 0.5 0.3 0.028 0.0036 0.0019 0.054 0.038 0.68 -0.02 -0.021 0.31 0.21 0.22 0.2
Attrition -0.16 1 0.078 -0.031 -0.1 -0.0068 -0.13 -0.17 -0.1 -0.16 0.043 0.25 -0.013 0.0029 -0.046 -0.14 -0.17 -0.059 -0.064 -0.13 -0.16 -0.033 -0.16
DistanceFromHome -0.0017 0.078 1 0.021 -0.016 0.031 0.0088 0.0053 -0.0037 -0.017 -0.029 0.026 0.04 0.027 0.0066 0.045 0.0046 -0.037 -0.027 0.0095 0.019 0.01 0.014
Education 0.21 -0.031 0.021 1 -0.027 0.017 0.042 0.1 -0.011 0.095 0.13 -0.02 -0.011 -0.025 -0.0091 0.018 0.15 -0.025 0.0098 0.069 0.06 0.054 0.069
EnvironmentSatisfaction 0.01 -0.1 -0.016 -0.027 1 -0.05 -0.0083 0.0012 -0.0068 -0.0063 0.013 0.07 -0.032 -0.03 0.0077 0.0034 -0.0027 -0.019 0.028 0.0015 0.018 0.016 -0.005
HourlyRate 0.024 -0.0068 0.031 0.017 -0.05 1 0.043 -0.028 -0.071 -0.016 0.022 -0.0078 -0.0091 -0.0022 0.0013 0.05 -0.0023 -0.0085 -0.0046 -0.02 -0.024 -0.027 -0.02
JobInvolvement 0.03 -0.13 0.0088 0.042 -0.0083 0.043 1 -0.013 -0.021 -0.015 0.015 -0.0035 -0.017 -0.029 0.034 0.022 -0.0055 -0.015 -0.015 -0.021 0.0087 -0.024 0.026
JobLevel 0.51 -0.17 0.0053 0.1 0.0012 -0.028 -0.013 1 -0.0019 0.95 0.14 0.00054 -0.035 -0.021 0.022 0.014 0.78 -0.018 0.038 0.53 0.39 0.35 0.38
JobSatisfaction -0.0049 -0.1 -0.0037 -0.011 -0.0068 -0.071 -0.021 -0.0019 1 -0.0072 -0.056 0.025 0.02 0.0023 -0.012 0.011 -0.02 -0.0058 -0.019 -0.0038 -0.0023 -0.018 -0.028
MonthlyIncome 0.5 -0.16 -0.017 0.095 -0.0063 -0.016 -0.015 0.95 -0.0072 1 0.15 0.0061 -0.027 -0.017 0.026 0.0054 0.77 -0.022 0.031 0.51 0.36 0.34 0.34
NumCompaniesWorked 0.3 0.043 -0.029 0.13 0.013 0.022 0.015 0.14 -0.056 0.15 1 -0.021 -0.01 -0.014 0.053 0.03 0.24 -0.066 -0.0084 -0.12 -0.091 -0.037 -0.11
OverTime 0.028 0.25 0.026 -0.02 0.07 -0.0078 -0.0035 0.00054 0.025 0.0061 -0.021 1 -0.0054 0.0044 0.048 -0.00045 0.013 -0.079 -0.027 -0.012 -0.03 -0.012 -0.042
PercentSalaryHike 0.0036 -0.013 0.04 -0.011 -0.032 -0.0091 -0.017 -0.035 0.02 -0.027 -0.01 -0.0054 1 0.77 -0.04 0.0075 -0.021 -0.0052 -0.0033 -0.036 -0.0015 -0.022 -0.012
PerformanceRating 0.0019 0.0029 0.027 -0.025 -0.03 -0.0022 -0.029 -0.021 0.0023 -0.017 -0.014 0.0044 0.77 1 -0.031 0.0035 0.0067 -0.016 0.0026 0.0034 0.035 0.018 0.023
RelationshipSatisfaction 0.054 -0.046 0.0066 -0.0091 0.0077 0.0013 0.034 0.022 -0.012 0.026 0.053 0.048 -0.04 -0.031 1 -0.046 0.024 0.0025 0.02 0.019 -0.015 0.033 -0.00087
StockOptionLevel 0.038 -0.14 0.045 0.018 0.0034 0.05 0.022 0.014 0.011 0.0054 0.03 -0.00045 0.0075 0.0035 -0.046 1 0.01 0.011 0.0041 0.015 0.051 0.014 0.025
TotalWorkingYears 0.68 -0.17 0.0046 0.15 -0.0027 -0.0023 -0.0055 0.78 -0.02 0.77 0.24 0.013 -0.021 0.0067 0.024 0.01 1 -0.036 0.001 0.63 0.46 0.4 0.46
TrainingTimesLastYear -0.02 -0.059 -0.037 -0.025 -0.019 -0.0085 -0.015 -0.018 -0.0058 -0.022 -0.066 -0.079 -0.0052 -0.016 0.0025 0.011 -0.036 1 0.028 0.0036 -0.0057 -0.0021 -0.0041
WorkLifeBalance -0.021 -0.064 -0.027 0.0098 0.028 -0.0046 -0.015 0.038 -0.019 0.031 -0.0084 -0.027 -0.0033 0.0026 0.02 0.0041 0.001 0.028 1 0.012 0.05 0.0089 0.0028
YearsAtCompany 0.31 -0.13 0.0095 0.069 0.0015 -0.02 -0.021 0.53 -0.0038 0.51 -0.12 -0.012 -0.036 0.0034 0.019 0.015 0.63 0.0036 0.012 1 0.76 0.62 0.77
YearsInCurrentRole 0.21 -0.16 0.019 0.06 0.018 -0.024 0.0087 0.39 -0.0023 0.36 -0.091 -0.03 -0.0015 0.035 -0.015 0.051 0.46 -0.0057 0.05 0.76 1 0.55 0.71
1.0
0.8
0.6
0.4
0.2
0.0



Upon analyzing both dataframes, we can say that,

Age plays a factor, as the median age is lower for employees who left

Workplace is further from home for those who left

Average Monthly income is lower for those employee who left

Average overtime worked is higher for employees who left

Monthly salary should be proportional to job role and years in current company and role.

We can see that the employees who left were not in the company for long, such as 1-7 years, compared to those who stayed (3-10 years).

The total years of experience is also lower for employees that left

This could mean that the lower average monthly salary may not be a factor as salary increases with years of experience, years worked, and current role.

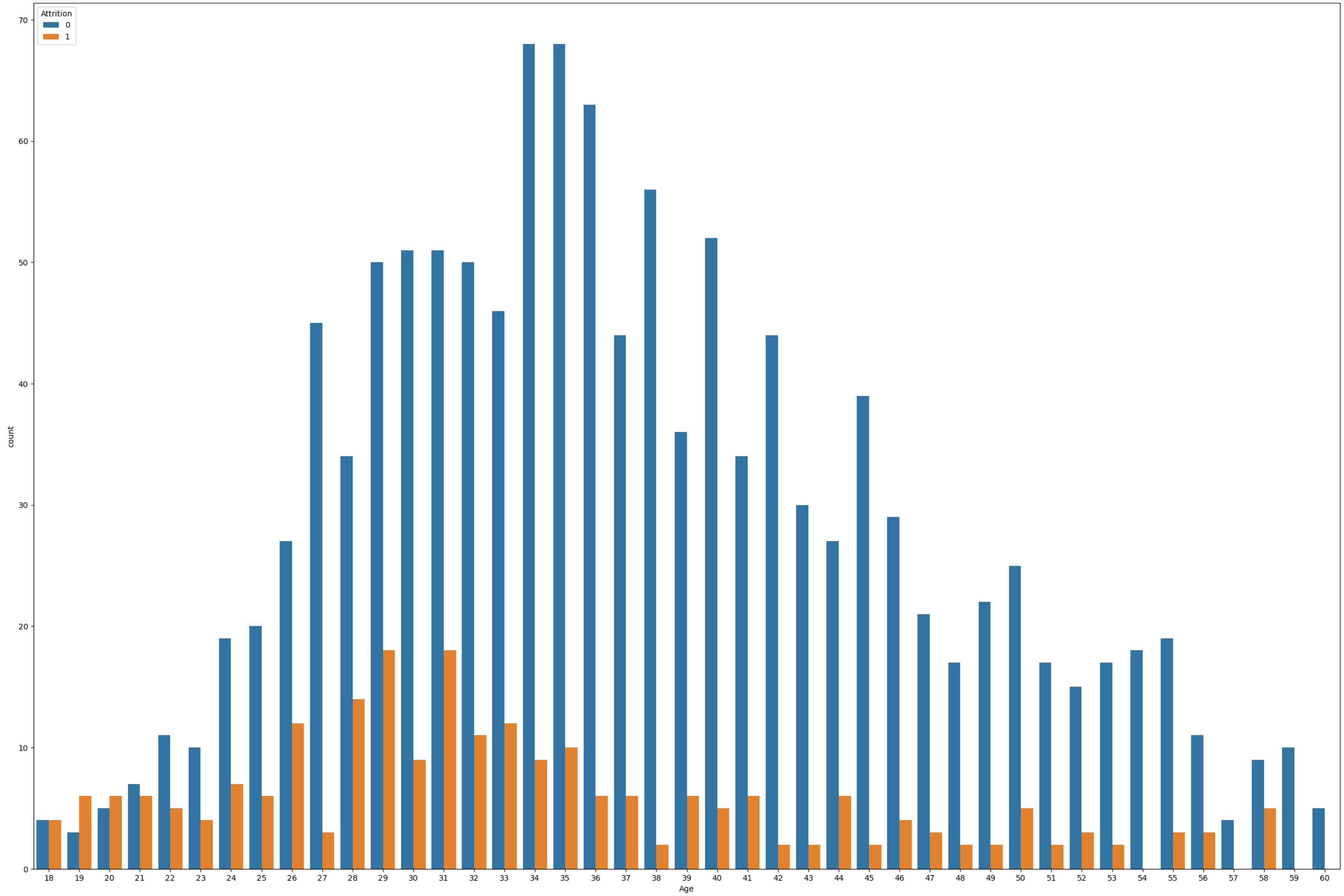
This is confimed by looking at the correlation chart

In [ ]:

```
# Checking for relation between age and attrition
plt.figure(figsize=(30,20))
sns.countplot(x= 'Age', hue= 'Attrition', data=employee_dataframe)
```

Out[ ]:

```
<AxesSubplot:xlabel='Age', ylabel='count'>
```



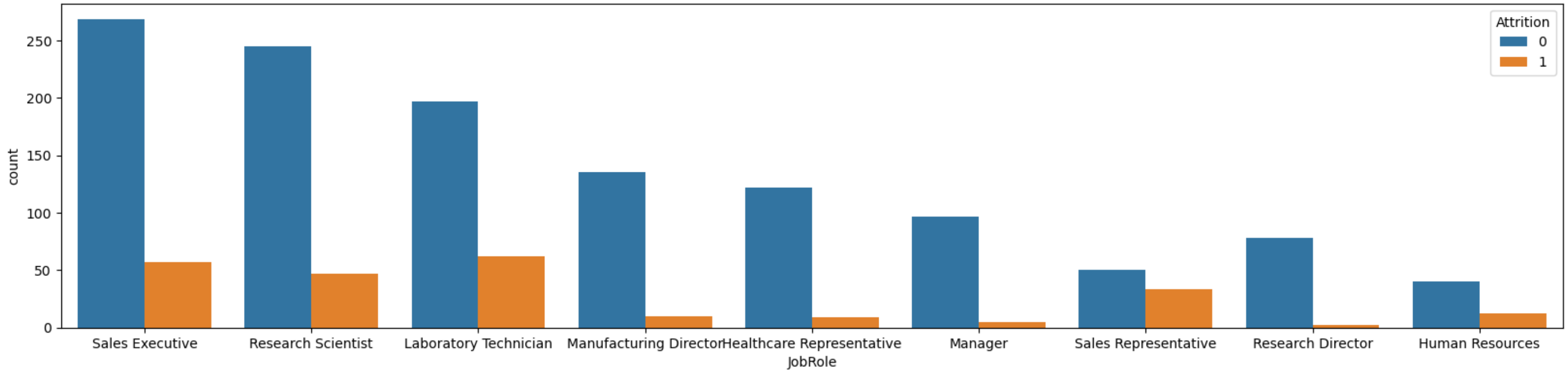
By visualizing the graph, it looks like employees around the age 28-29 and 31 leaves the company the most. It is safe to say that age is a factor.

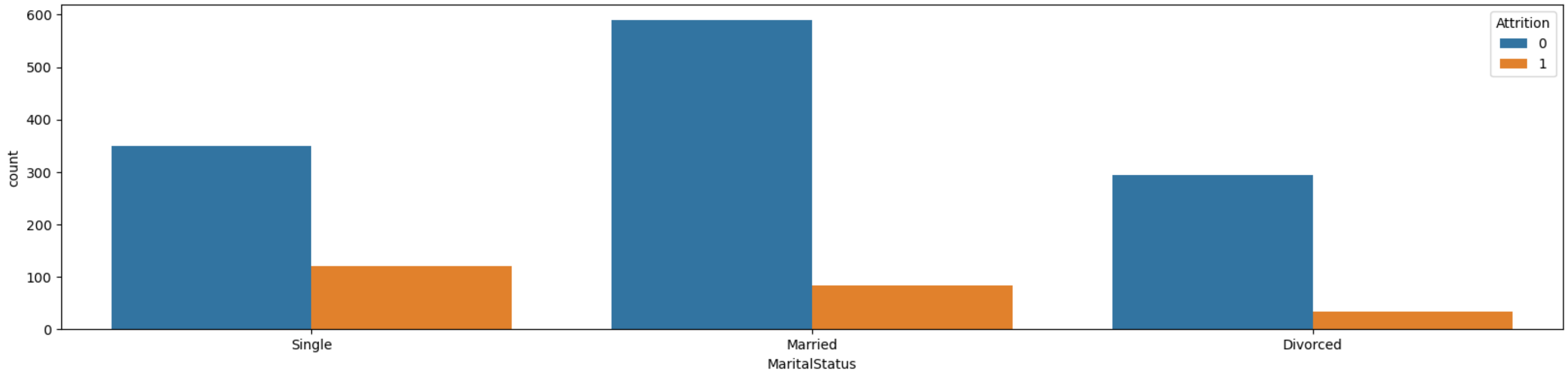
In [ ]:
```
# Exploring the job role, marital status, job involvement and job level.
plt.figure(figsize=(20,20))
plt.subplot(4,1,1)
sns.countplot(x='JobRole', hue='Attrition',data=employee_dataframe)
plt.subplot(4,1,2)
```

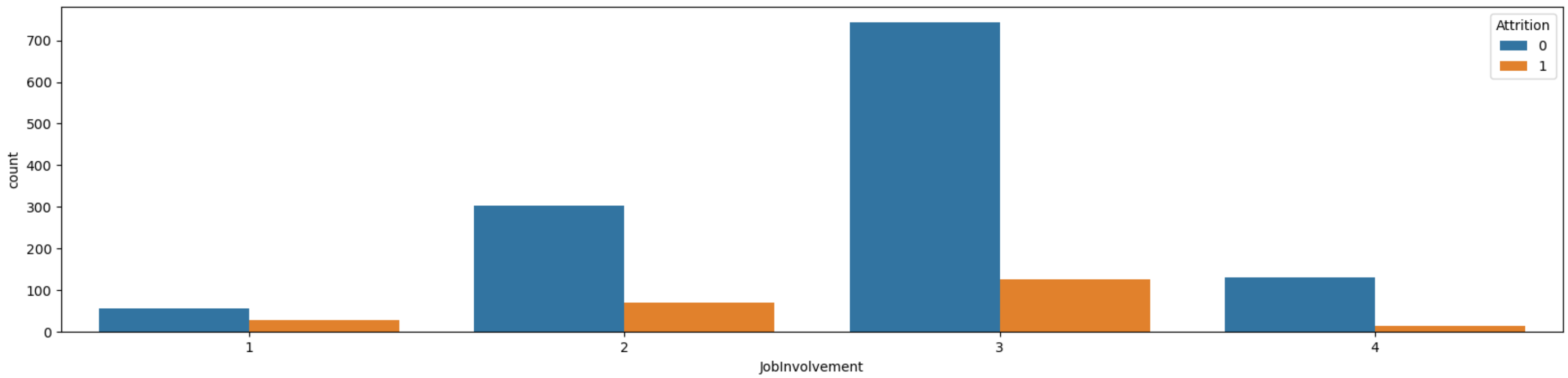
```
sns.countplot(x='MaritalStatus', hue='Attrition',data=employee_dataframe)
plt.subplot(4,1,3)
sns.countplot(x='JobInvolvement', hue='Attrition',data=employee_dataframe)
plt.subplot(4,1,4)
sns.countplot(x='JobLevel', hue='Attrition',data=employee_dataframe)
```

Out[ ]:

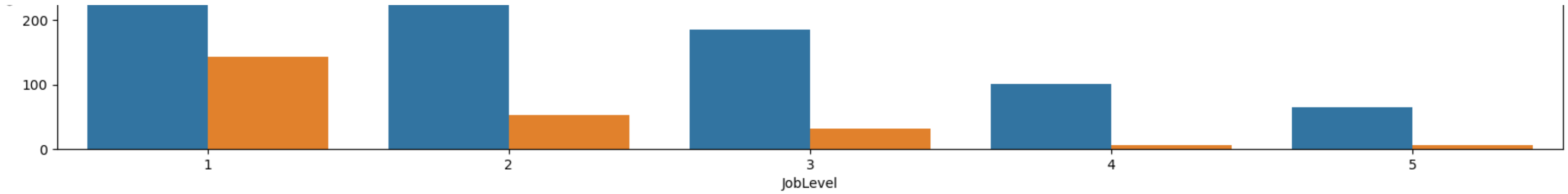
```
<AxesSubplot:xlabel='JobLevel', ylabel='count'>
```


Attrition
0
1
count
JobRole
Sales Executive
Research Scientist
Laboratory Technician
Manufacturing Director
Healthcare Representative
Manager
Sales Representative
Research Director
Human Resources


Attrition
0
1
count
MaritalStatus
Single
Married
Divorced


Attrition
0
1
count
JobInvolvement


Attrition
0
1
count



Upon analyzing the graphs, we can say that,

Proportionaly, sales represnetatives have a high turnover rate

Single employees tent to leave compared to married or divorced

Less experienced emplyees tend to leave

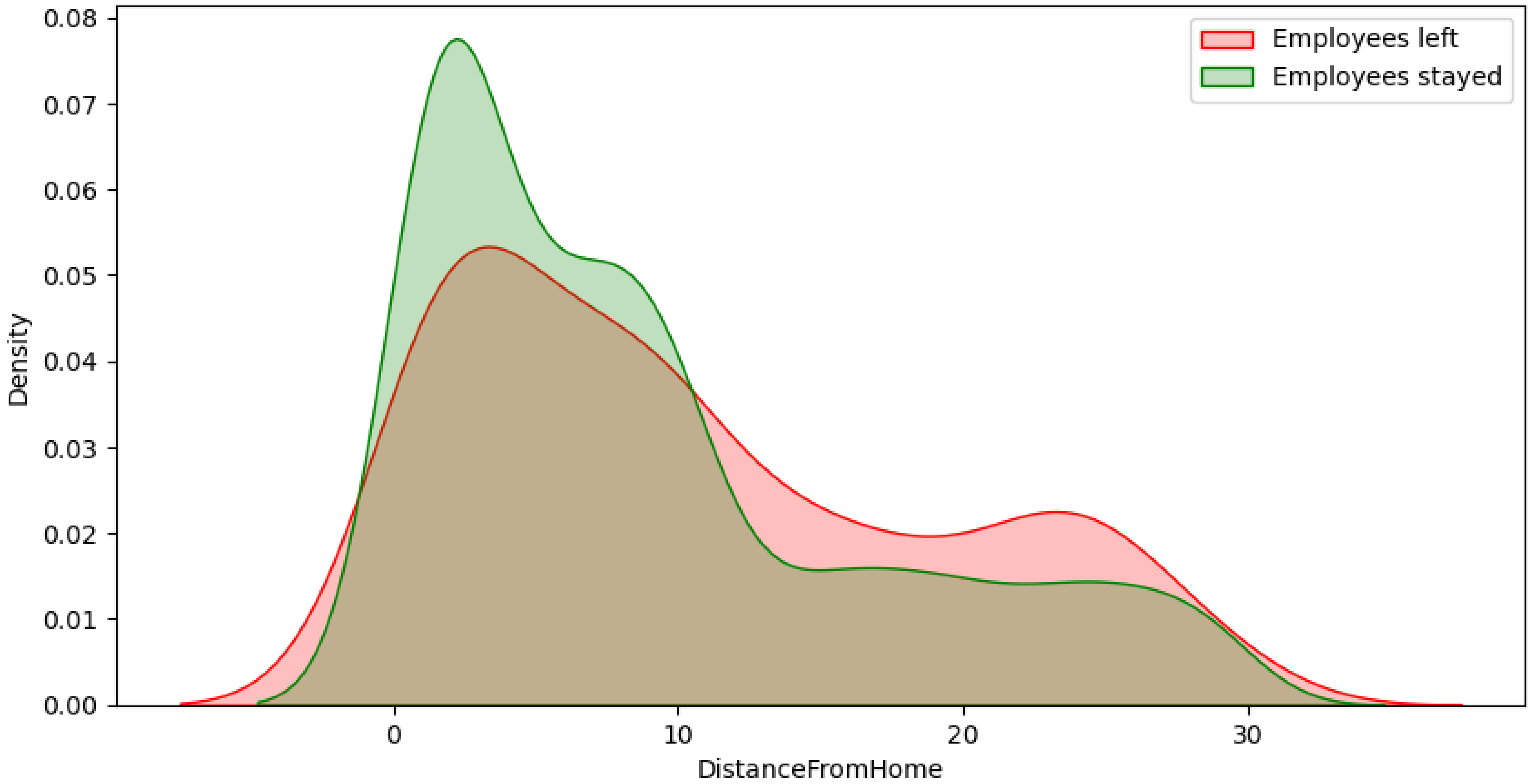
Less involved employees have a high turnover rate

In [ ]:

```
# Analyzing distance from home regarding attrition.
# Developing a KDE (Karnel Density Estimate) which will be used for visualizing the probability density of a continous
plt.figure(figsize=(10,5))
sns.kdeplot(left_df['DistanceFromHome'], shade=True, label ='Employees left', color='red')
sns.kdeplot(stayed_df['DistanceFromHome'], shade=True, label ='Employees stayed', color='green')
plt.legend()
```

Out[ ]:

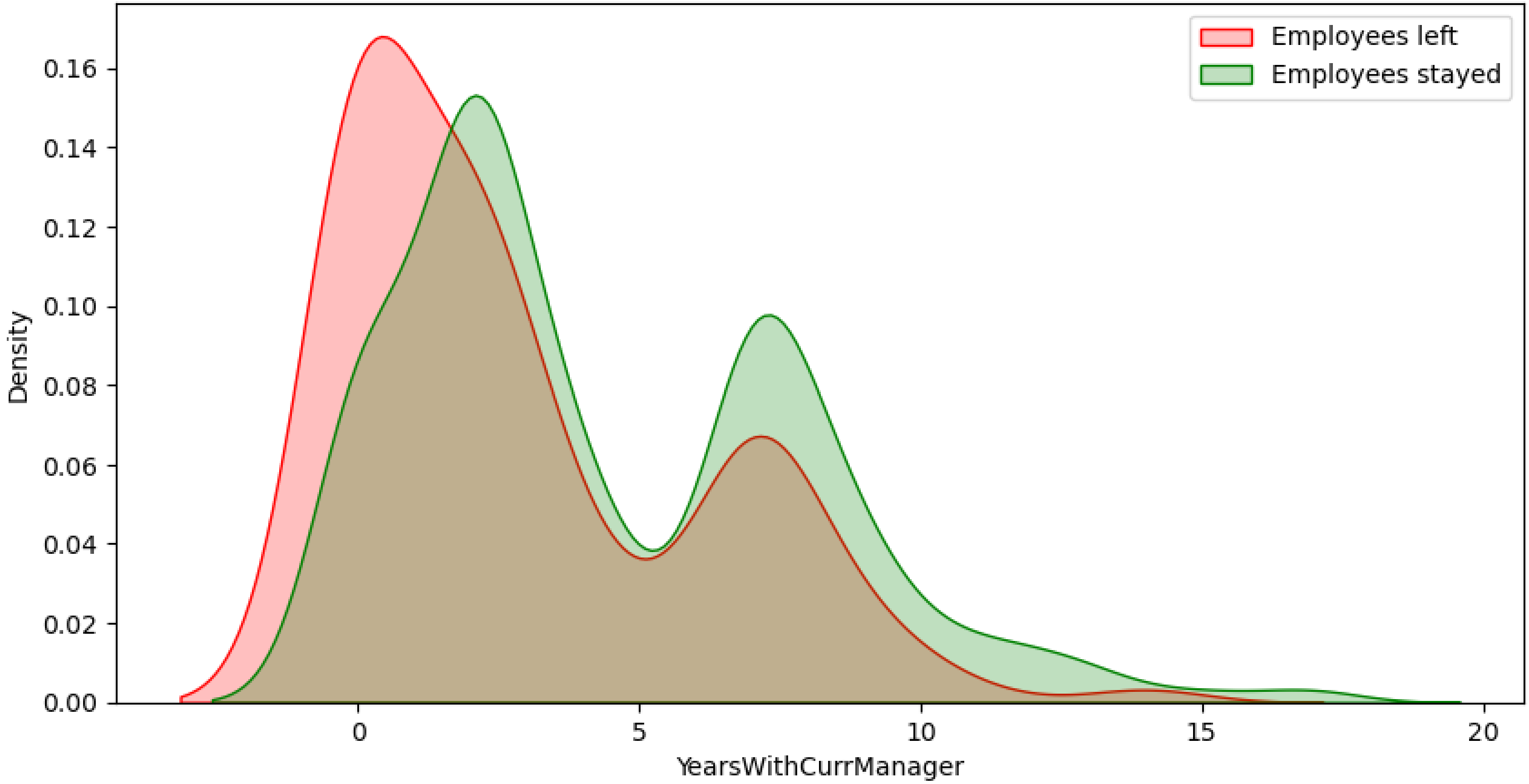
```
<matplotlib.legend.Legend at 0x2562a6766d0>
```



Upon analyzing the above KDE plot, we can visualize that employees that live around 10-20km from their work location tents to leave the job more. However, this is not the only reason as there is not a big difference compared to employees that stayed.

In [ ]:
```
# Analyzing years with current manager with attrition
plt.figure(figsize=(10,5))
sns.kdeplot(left_df['YearsWithCurrManager'], shade=True, color='red', label = 'Employees left')
sns.kdeplot(stayed_df['YearsWithCurrManager'], shade=True, color='green', label = 'Employees stayed')
plt.legend()
```

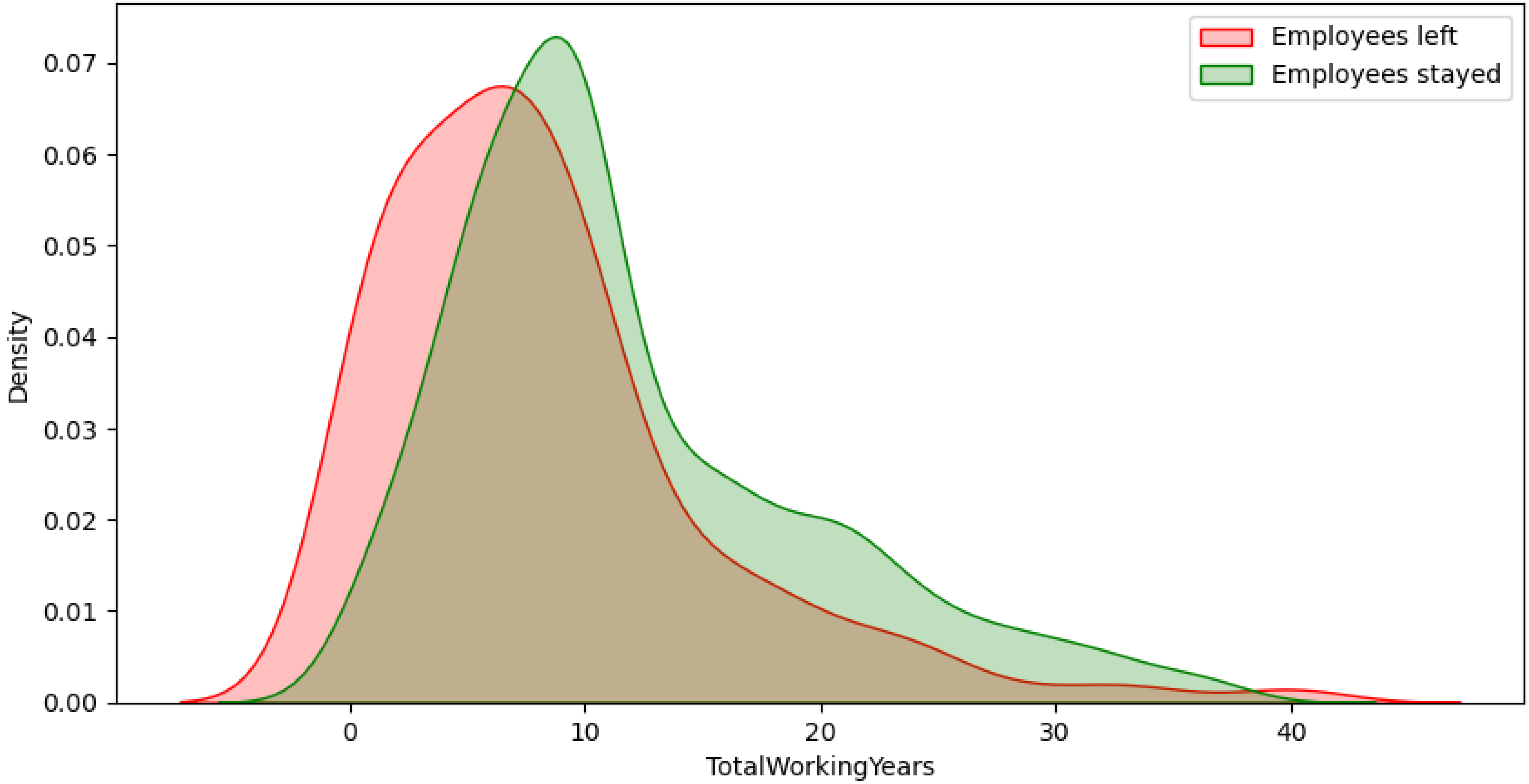
Out[ ]:
```
<matplotlib.legend.Legend at 0x2562b73b3d0>
```



This graph shows that a lot of the employees who has left has been with their current manager less than 1 year. This could possibly mean that the employees may not have liked working with their manager and left early on.

In [ ]:
```
# Analyzing total working years with attrition
plt.figure(figsize=(10,5))
sns.kdeplot(left_df['TotalWorkingYears'], shade=True, color='red', label = 'Employees left')
sns.kdeplot(stayed_df['TotalWorkingYears'], shade=True, color='green', label = 'Employees stayed')
plt.legend()
```

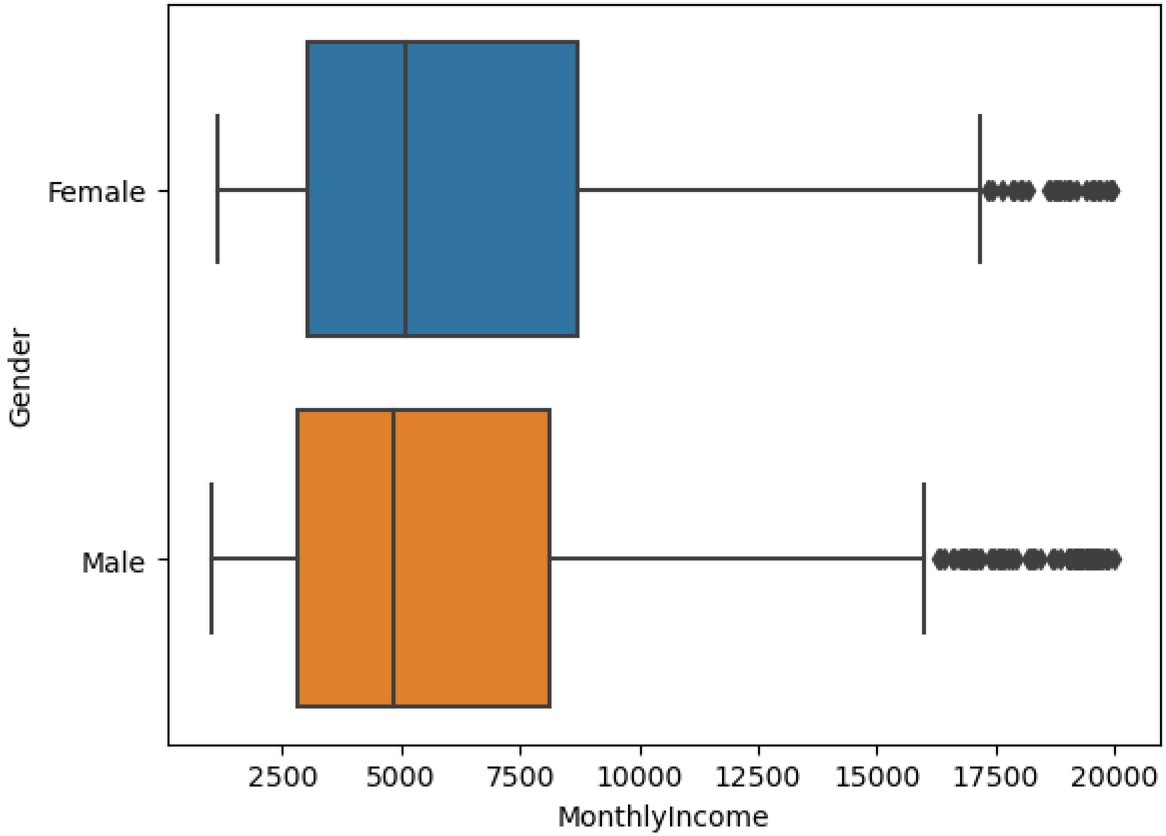
Out[ ]:
```
<matplotlib.legend.Legend at 0x2562b14dd00>
```



Most of the employees that left, left duing 1st 5 years.

In [ ]:
```
#Gender by monthly income
sns.boxplot(x = 'MonthlyIncome', y='Gender', data=employee_dataframe)
```

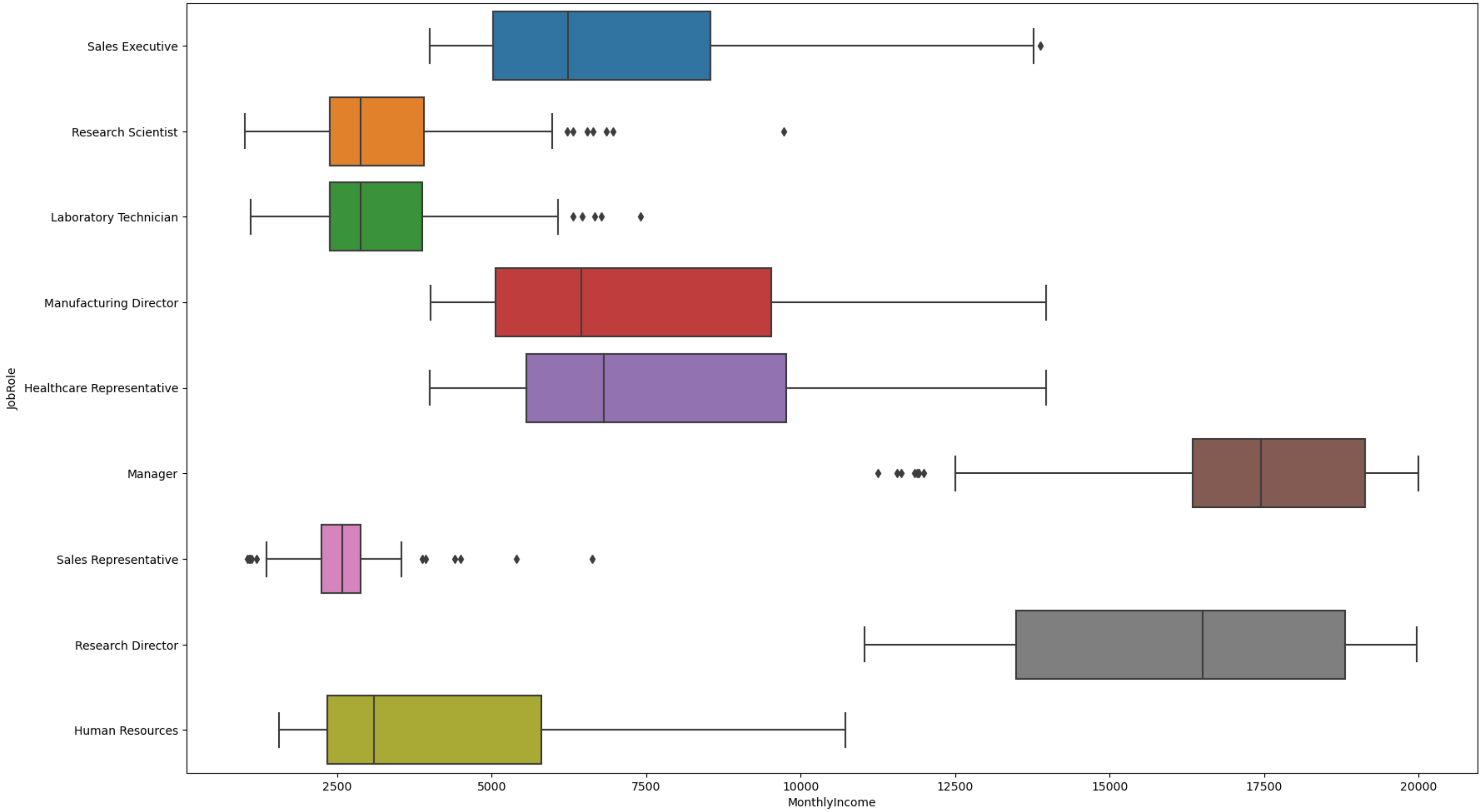
Out[ ]:
```
<AxesSubplot:xlabel='MonthlyIncome', ylabel='Gender'>
```



Average salary seems very similar to each other, so gender discrimination based on salary is not a constrain.

In [ ]:
```
# job role vs monthly income
plt.figure(figsize=(20,12))
sns.boxplot(x = 'MonthlyIncome', y='JobRole', data=employee_dataframe)
```

Out[ ]:
```
<AxesSubplot:xlabel='MonthlyIncome', ylabel='JobRole'>
```



Upon looking at the job role vs monthly income chat, we can say that a reason why sales representative role had the highest turnover rate is due to their monthly income. Their highest and lowest salary is already low compared to other job roles, with a few high outliers.

Data Processing

In [ ]:

```
#Getting all categorical data and changing it to numeric value so it can be used for AI algorithm
categorical = employee_dataframe[['BusinessTravel', 'Department', 'EducationField', 'Gender', 'JobRole', 'MaritalStatus
categorical
```

Out[ ]:

	BusinessTravel	Department	EducationField	Gender	JobRole	MaritalStatus
0	Travel_Rarely	Sales	Life Sciences	Female	Sales Executive	Single
1	Travel_Frequently	Research & Development	Life Sciences	Male	Research Scientist	Married
2	Travel_Rarely	Research & Development	Other	Male	Laboratory Technician	Single
3	Travel_Frequently	Research & Development	Life Sciences	Female	Research Scientist	Married
4	Travel_Rarely	Research & Development	Medical	Male	Laboratory Technician	Married
...	...	...	...	...	...	...
1465	Travel_Frequently	Research & Development	Medical	Male	Laboratory Technician	Married
1466	Travel_Rarely	Research & Development	Medical	Male	Healthcare Representative	Married
1467	Travel_Rarely	Research & Development	Life Sciences	Male	Manufacturing Director	Married
1468	Travel_Frequently	Sales	Medical	Male	Sales Executive	Married
1469	Travel_Rarely	Research & Development	Medical	Male	Laboratory Technician	Married

1470 rows × 6 columns

In [ ]:

```
from sklearn.preprocessing import OneHotEncoder
oneHotEncoder = OneHotEncoder()
categorical = oneHotEncoder.fit_transform(categorical).toarray()
```

In [ ]:

```
#categorical.toarray()
categorical = pd.DataFrame(categorical)
```

In [ ]:

```
categorical
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
1	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
2	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
4	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1465	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
1466	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
1467	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
1468	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0
1469	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0

1470 rows × 26 columns

In [ ]:

```
numerical = employee_dataframe[['Age','DistanceFromHome',       'Education', 'EnvironmentSatisfaction', 'HourlyRate', '
numerical
```

Out[ ]:

	Age	DistanceFromHome	Education	EnvironmentSatisfaction	HourlyRate	JobInvolvement	JobLevel	JobSatisfaction	MonthlyIncome
0	41	1	2	2	94	3	2	4	5993
1	49	8	1	3	61	2	2	2	5130
2	37	2	2	4	92	2	1	3	2090
3	33	3	4	4	56	3	1	3	2909
4	27	2	1	1	40	3	1	2	3468
...	...	...	...	...	...	...	...	...	...
1465	36	23	2	3	41	4	2	4	2571
1466	39	6	1	4	42	2	3	1	9991
1467	27	4	3	2	87	4	2	2	6142
1468	49	2	3	4	63	2	2	2	5390
1469	34	8	3	2	82	4	2	3	4404

1470 rows × 22 columns

In [ ]:

```
#combining both numeric and categoric->numeric data into one
combined_data = pd.concat([categorical,numerical], axis=1)
combined_data
```

Out[ ]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	Age	DistanceFr
0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	41	
1	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	49	
2	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	37	
3	0.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	33	
4	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	27	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
1465	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	36	
1466	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	39	
1467	0.0	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	27	
1468	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	49	
1469	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	34	

1470 rows × 48 columns

In [ ]:

```
#Scaling all the data
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
x = scaler.fit_transform(combined_data)
```

```
c:\Users\yasin\anaconda3\lib\site-packages\sklearn\utils\validation.py:1688: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  warnings.warn(
c:\Users\yasin\anaconda3\lib\site-packages\sklearn\utils\validation.py:1688: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['int', 'str']. An error will be raised in 1.2.
  warnings.warn(
```

In [ ]:

```
x
```

Out[ ]:
```
array([[0.        , 0.        , 1.        , ..., 0.22222222, 0.        ,
        0.29411765],
       [0.        , 1.        , 0.        , ..., 0.38888889, 0.06666667,
        0.41176471],
       [0.        , 0.        , 1.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 1.        , ..., 0.11111111, 0.        ,
        0.17647059],
       [0.        , 1.        , 0.        , ..., 0.33333333, 0.        ,
        0.47058824],
       [0.        , 0.        , 1.        , ..., 0.16666667, 0.06666667,
        0.11764706]])
```

In [ ]:
```
y = employee_dataframe['Attrition']
y
```

Out[ ]:
```
0       1
1       0
2       1
3       0
4       0
       ..
1465    0
1466    0
1467    0
1468    0
1469    0
Name: Attrition, Length: 1470, dtype: int64
```

In [ ]:
```
#Training/Evaluating the data using algorithms
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size= 0.25)
```

Using Logistic Regression Classifier

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
model_LogRegression = LogisticRegression()
model_LogRegression.fit(x_train, y_train, )
```

Out[ ]:
```
LogisticRegression()
```

In [ ]:
```
y_pred_LogRegression = model_LogRegression.predict(x_test)
```

In [ ]:
```
y_pred_LogRegression
# O indicating false that the employee will stay and 1 indicating true that the employee will leave
```

Out[ ]:
```
array([0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0], dtype=int64)
```

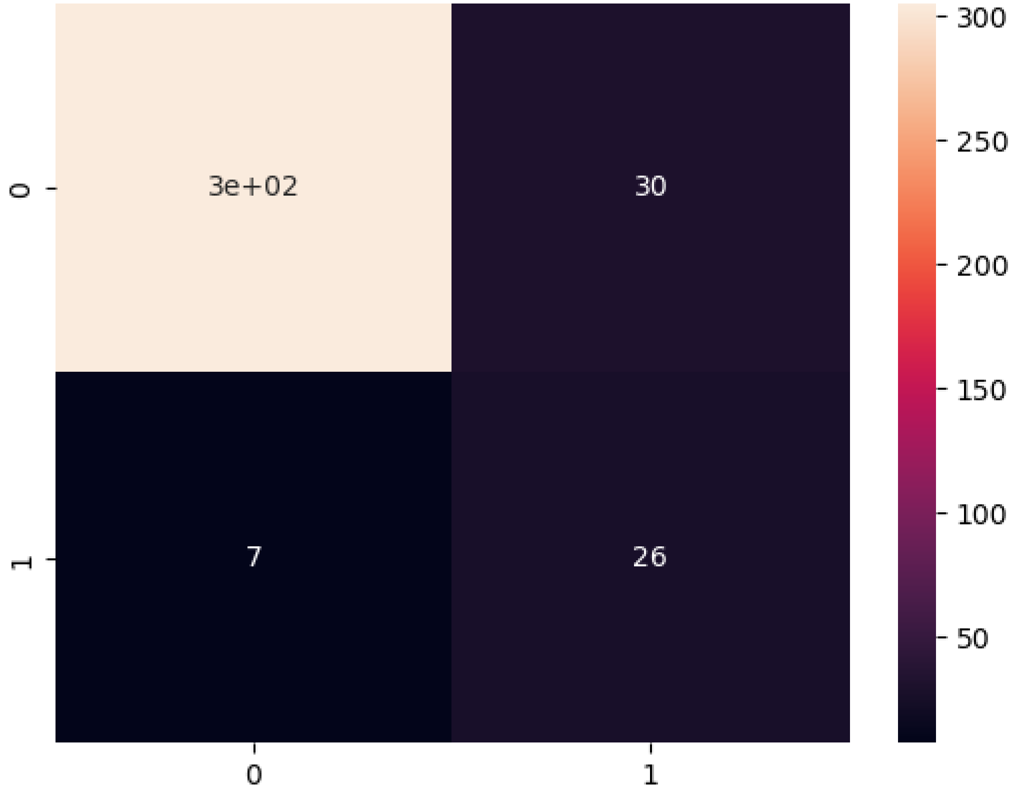
In [ ]:
```
from sklearn.metrics import confusion_matrix, classification_report
```

In [ ]:
```
print ('Accuracy {}%'.format(100* accuracy_score (y_pred_LogRegression, y_test)))
```

```
Accuracy 89.94565217391305%
```

In [ ]:
```
cm_LogRegression = confusion_matrix(y_pred_LogRegression, y_test)
sns.heatmap(cm_LogRegression,annot=True)
```

Out[ ]:
```
<AxesSubplot:>
```



Using Logistic Regression Classifier with an accuracy of 89.6%
First column (index 0) states that the model has provided, 2nd column (index 1) indicates the difference between the real world data vs model's prediction.
First row (0) indicates employees that will stay.
Second row (1) indicates employees that will leave.

Upon looking at the heat-map above, we can see that
model has predicted that around 3000 employees will be staying and it is correct (True positive).
Model has accurately determined that 25 employees will leave the company (True negative).
Model has falsely identified that 32 employees will leave the job but in actual, they will not (False positive/Type 1 error).
Model has falsely identified that 6 employees will not leave the job (False negative/Type 2 error).

Using Random Forest Classifier

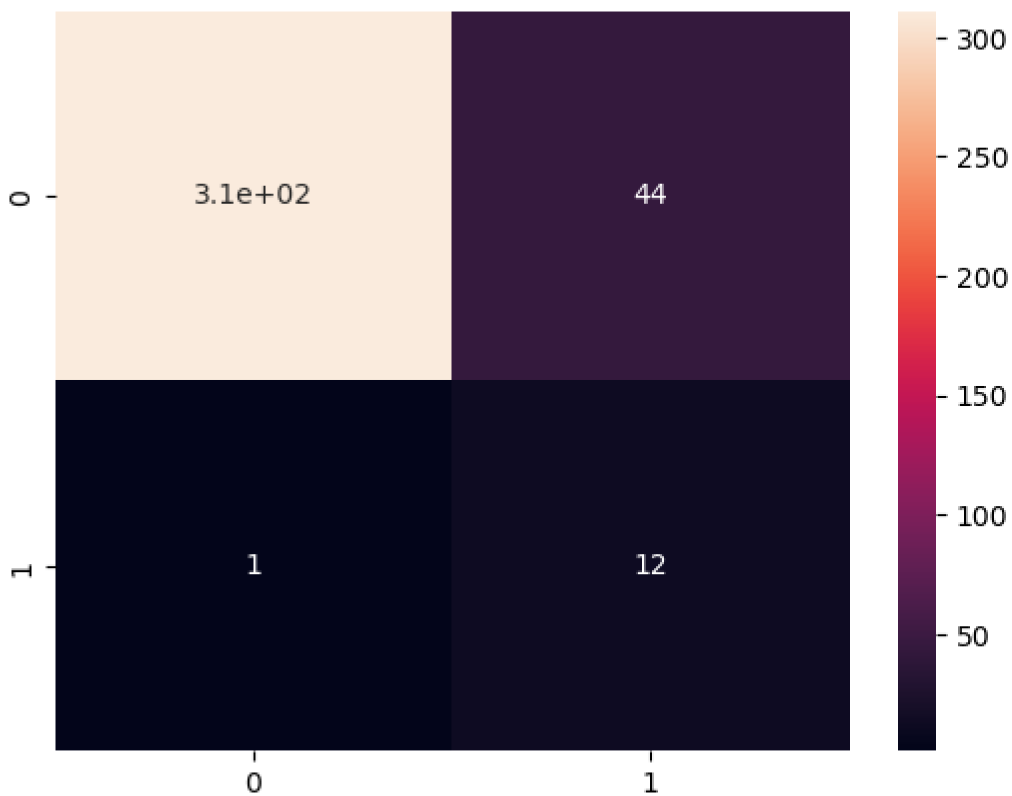
In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
model_randomForest = RandomForestClassifier()
model_randomForest.fit(x_train, y_train)
```

Out[ ]:
```
RandomForestClassifier()
```

In [ ]:
```
y_pred_randomForest = model_randomForest.predict(x_test)
```

In [ ]:
```
cm_RandomForest = confusion_matrix(y_pred_randomForest, y_test)
sns.heatmap(cm_RandomForest,annot=True)
```

Out[ ]:
```
<AxesSubplot:>
```



Using Artificial Neural Network Classifier

In [ ]:
```
import tensorflow as tf
model_tf = tf.keras.models.Sequential()
```

```
model_tf.add(tf.keras.layers.Dense(units = 500, activation='relu', input_shape = (48,)))
model_tf.add(tf.keras.layers.Dense(units=500, activation='relu'))
model_tf.add(tf.keras.layers.Dense(units=500, activation='relu'))
model_tf.add(tf.keras.layers.Dense(units=1, activation='sigmoid'))
```

In [ ]:
```
model_tf.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_4 (Dense)             (None, 500)               24500     
                                                                 
 dense_5 (Dense)             (None, 500)               250500    
                                                                 
 dense_6 (Dense)             (None, 500)               250500    
                                                                 
 dense_7 (Dense)             (None, 1)                 501       
                                                                 
=================================================================
Total params: 526,001
Trainable params: 526,001
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_4 (Dense)             (None, 500)               24500     
                                                                 
 dense_5 (Dense)             (None, 500)               250500    
                                                                 
 dense_6 (Dense)             (None, 500)               250500    
                                                                 
 dense_7 (Dense)             (None, 1)                 501       
                                                                 
=================================================================
Total params: 526,001
Trainable params: 526,001
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:
```
model_tf.compile(optimizer='Adam', loss='binary_crossentropy',metrics=['accuracy'])
```

In [ ]:
```
epochs_hist = model_tf.fit(x_train, y_train, epochs=100, batch_size=48)
```

```
Epoch 1/100
23/23 [==============================] - 1s 8ms/step - loss: 0.4271 - accuracy: 0.8185
Epoch 2/100
23/23 [==============================] - 0s 7ms/step - loss: 0.3648 - accuracy: 0.8485
Epoch 3/100
23/23 [==============================] - 0s 7ms/step - loss: 0.3171 - accuracy: 0.8775
Epoch 4/100
23/23 [==============================] - 0s 8ms/step - loss: 0.2986 - accuracy: 0.8820
Epoch 5/100
23/23 [==============================] - 0s 7ms/step - loss: 0.2545 - accuracy: 0.9020
Epoch 6/100
23/23 [==============================] - 0s 7ms/step - loss: 0.2223 - accuracy: 0.9138
Epoch 7/100
23/23 [==============================] - 0s 7ms/step - loss: 0.1921 - accuracy: 0.9265
Epoch 8/100
23/23 [==============================] - 0s 7ms/step - loss: 0.1638 - accuracy: 0.9446
Epoch 9/100
23/23 [==============================] - 0s 7ms/step - loss: 0.1217 - accuracy: 0.9619
Epoch 10/100
23/23 [==============================] - 0s 6ms/step - loss: 0.1010 - accuracy: 0.9655
Epoch 11/100
23/23 [==============================] - 0s 6ms/step - loss: 0.0809 - accuracy: 0.9728
Epoch 12/100
23/23 [==============================] - 0s 7ms/step - loss: 0.0635 - accuracy: 0.9791
Epoch 13/100
23/23 [==============================] - 0s 5ms/step - loss: 0.0434 - accuracy: 0.9873
Epoch 14/100
23/23 [==============================] - 0s 5ms/step - loss: 0.0415 - accuracy: 0.9855
Epoch 15/100
23/23 [==============================] - 0s 6ms/step - loss: 0.0362 - accuracy: 0.9909
Epoch 16/100
23/23 [==============================] - 0s 6ms/step - loss: 0.0257 - accuracy: 0.9946
Epoch 17/100
23/23 [==============================] - 0s 9ms/step - loss: 0.0207 - accuracy: 0.9936
Epoch 18/100
23/23 [==============================] - 0s 7ms/step - loss: 0.0306 - accuracy: 0.9855
Epoch 19/100
23/23 [==============================] - 0s 8ms/step - loss: 0.0322 - accuracy: 0.9909
Epoch 20/100
23/23 [==============================] - 0s 8ms/step - loss: 0.0204 - accuracy: 0.9918
Epoch 21/100
23/23 [==============================] - 0s 7ms/step - loss: 0.0217 - accuracy: 0.9909
Epoch 22/100
23/23 [==============================] - 0s 6ms/step - loss: 0.0301 - accuracy: 0.9918
Epoch 23/100
```

```
23/23 [==============================] - 0s 7ms/step - loss: 0.0153 - accuracy: 0.9946
Epoch 24/100
23/23 [==============================] - 0s 7ms/step - loss: 0.0058 - accuracy: 1.0000
Epoch 25/100
23/23 [==============================] - 0s 6ms/step - loss: 0.0037 - accuracy: 0.9982
Epoch 26/100
23/23 [==============================] - 0s 6ms/step - loss: 0.0014 - accuracy: 1.0000
Epoch 27/100
23/23 [==============================] - 0s 6ms/step - loss: 8.0462e-04 - accuracy: 1.0000
Epoch 28/100
23/23 [==============================] - 0s 6ms/step - loss: 4.4040e-04 - accuracy: 1.0000
Epoch 29/100
23/23 [==============================] - 0s 6ms/step - loss: 3.5381e-04 - accuracy: 1.0000
Epoch 30/100
23/23 [==============================] - 0s 7ms/step - loss: 2.8868e-04 - accuracy: 1.0000
Epoch 31/100
23/23 [==============================] - 0s 6ms/step - loss: 2.5228e-04 - accuracy: 1.0000
Epoch 32/100
23/23 [==============================] - 0s 6ms/step - loss: 2.1898e-04 - accuracy: 1.0000
Epoch 33/100
23/23 [==============================] - 0s 6ms/step - loss: 1.9434e-04 - accuracy: 1.0000
Epoch 34/100
23/23 [==============================] - 0s 6ms/step - loss: 1.7873e-04 - accuracy: 1.0000
Epoch 35/100
23/23 [==============================] - 0s 6ms/step - loss: 1.6119e-04 - accuracy: 1.0000
Epoch 36/100
23/23 [==============================] - 0s 6ms/step - loss: 1.4476e-04 - accuracy: 1.0000
Epoch 37/100
23/23 [==============================] - 0s 6ms/step - loss: 1.3315e-04 - accuracy: 1.0000
Epoch 38/100
23/23 [==============================] - 0s 6ms/step - loss: 1.2163e-04 - accuracy: 1.0000
Epoch 39/100
23/23 [==============================] - 0s 6ms/step - loss: 1.1321e-04 - accuracy: 1.0000
Epoch 40/100
23/23 [==============================] - 0s 6ms/step - loss: 1.0423e-04 - accuracy: 1.0000
Epoch 41/100
23/23 [==============================] - 0s 7ms/step - loss: 9.7895e-05 - accuracy: 1.0000
Epoch 42/100
23/23 [==============================] - 0s 8ms/step - loss: 9.1520e-05 - accuracy: 1.0000
Epoch 43/100
23/23 [==============================] - 0s 9ms/step - loss: 8.4836e-05 - accuracy: 1.0000
Epoch 44/100
23/23 [==============================] - 0s 7ms/step - loss: 7.8762e-05 - accuracy: 1.0000
Epoch 45/100
23/23 [==============================] - 0s 7ms/step - loss: 7.3821e-05 - accuracy: 1.0000
```

```
Epoch 46/100
23/23 [==============================] - 0s 7ms/step - loss: 6.9117e-05 - accuracy: 1.0000
Epoch 47/100
23/23 [==============================] - 0s 7ms/step - loss: 6.5317e-05 - accuracy: 1.0000
Epoch 48/100
23/23 [==============================] - 0s 8ms/step - loss: 6.1282e-05 - accuracy: 1.0000
Epoch 49/100
23/23 [==============================] - 0s 7ms/step - loss: 5.7416e-05 - accuracy: 1.0000
Epoch 50/100
23/23 [==============================] - 0s 7ms/step - loss: 5.4377e-05 - accuracy: 1.0000
Epoch 51/100
23/23 [==============================] - 0s 7ms/step - loss: 5.1220e-05 - accuracy: 1.0000
Epoch 52/100
23/23 [==============================] - 0s 6ms/step - loss: 4.8342e-05 - accuracy: 1.0000
Epoch 53/100
23/23 [==============================] - 0s 5ms/step - loss: 4.5784e-05 - accuracy: 1.0000
Epoch 54/100
23/23 [==============================] - 0s 6ms/step - loss: 4.3267e-05 - accuracy: 1.0000
Epoch 55/100
23/23 [==============================] - 0s 6ms/step - loss: 4.1141e-05 - accuracy: 1.0000
Epoch 56/100
23/23 [==============================] - 0s 5ms/step - loss: 3.8940e-05 - accuracy: 1.0000
Epoch 57/100
23/23 [==============================] - 0s 5ms/step - loss: 3.6882e-05 - accuracy: 1.0000
Epoch 58/100
23/23 [==============================] - 0s 5ms/step - loss: 3.4975e-05 - accuracy: 1.0000
Epoch 59/100
23/23 [==============================] - 0s 5ms/step - loss: 3.3353e-05 - accuracy: 1.0000
Epoch 60/100
23/23 [==============================] - 0s 5ms/step - loss: 3.1677e-05 - accuracy: 1.0000
Epoch 61/100
23/23 [==============================] - 0s 5ms/step - loss: 3.0329e-05 - accuracy: 1.0000
Epoch 62/100
23/23 [==============================] - 0s 5ms/step - loss: 2.8665e-05 - accuracy: 1.0000
Epoch 63/100
23/23 [==============================] - 0s 8ms/step - loss: 2.7436e-05 - accuracy: 1.0000
Epoch 64/100
23/23 [==============================] - 0s 8ms/step - loss: 2.6150e-05 - accuracy: 1.0000
Epoch 65/100
23/23 [==============================] - 0s 6ms/step - loss: 2.4963e-05 - accuracy: 1.0000
Epoch 66/100
23/23 [==============================] - 0s 5ms/step - loss: 2.3727e-05 - accuracy: 1.0000
Epoch 67/100
23/23 [==============================] - 0s 5ms/step - loss: 2.2681e-05 - accuracy: 1.0000
Epoch 68/100
```

```
23/23 [==============================] - 0s 5ms/step - loss: 2.1728e-05 - accuracy: 1.0000
Epoch 69/100
23/23 [==============================] - 0s 6ms/step - loss: 2.0698e-05 - accuracy: 1.0000
Epoch 70/100
23/23 [==============================] - 0s 7ms/step - loss: 1.9844e-05 - accuracy: 1.0000
Epoch 71/100
23/23 [==============================] - 0s 6ms/step - loss: 1.8980e-05 - accuracy: 1.0000
Epoch 72/100
23/23 [==============================] - 0s 6ms/step - loss: 1.8217e-05 - accuracy: 1.0000
Epoch 73/100
23/23 [==============================] - 0s 6ms/step - loss: 1.7476e-05 - accuracy: 1.0000
Epoch 74/100
23/23 [==============================] - 0s 6ms/step - loss: 1.6618e-05 - accuracy: 1.0000
Epoch 75/100
23/23 [==============================] - 0s 5ms/step - loss: 1.5966e-05 - accuracy: 1.0000
Epoch 76/100
23/23 [==============================] - 0s 6ms/step - loss: 1.5286e-05 - accuracy: 1.0000
Epoch 77/100
23/23 [==============================] - 0s 6ms/step - loss: 1.4717e-05 - accuracy: 1.0000
Epoch 78/100
23/23 [==============================] - 0s 7ms/step - loss: 1.4096e-05 - accuracy: 1.0000
Epoch 79/100
23/23 [==============================] - 0s 5ms/step - loss: 1.3554e-05 - accuracy: 1.0000
Epoch 80/100
23/23 [==============================] - 0s 6ms/step - loss: 1.3009e-05 - accuracy: 1.0000
Epoch 81/100
23/23 [==============================] - 0s 5ms/step - loss: 1.2515e-05 - accuracy: 1.0000
Epoch 82/100
23/23 [==============================] - 0s 6ms/step - loss: 1.2046e-05 - accuracy: 1.0000
Epoch 83/100
23/23 [==============================] - 0s 6ms/step - loss: 1.1602e-05 - accuracy: 1.0000
Epoch 84/100
23/23 [==============================] - 0s 6ms/step - loss: 1.1100e-05 - accuracy: 1.0000
Epoch 85/100
23/23 [==============================] - 0s 8ms/step - loss: 1.0693e-05 - accuracy: 1.0000
Epoch 86/100
23/23 [==============================] - 0s 8ms/step - loss: 1.0242e-05 - accuracy: 1.0000
Epoch 87/100
23/23 [==============================] - 0s 9ms/step - loss: 9.8515e-06 - accuracy: 1.0000
Epoch 88/100
23/23 [==============================] - 0s 7ms/step - loss: 9.4724e-06 - accuracy: 1.0000
Epoch 89/100
23/23 [==============================] - 0s 7ms/step - loss: 9.0959e-06 - accuracy: 1.0000
Epoch 90/100
23/23 [==============================] - 0s 7ms/step - loss: 8.7276e-06 - accuracy: 1.0000
```

```
Epoch 91/100
23/23 [==============================] - 0s 5ms/step - loss: 8.3642e-06 - accuracy: 1.0000
Epoch 92/100
23/23 [==============================] - 0s 6ms/step - loss: 8.0682e-06 - accuracy: 1.0000
Epoch 93/100
23/23 [==============================] - 0s 6ms/step - loss: 7.6635e-06 - accuracy: 1.0000
Epoch 94/100
23/23 [==============================] - 0s 6ms/step - loss: 7.4548e-06 - accuracy: 1.0000
Epoch 95/100
23/23 [==============================] - 0s 5ms/step - loss: 7.1344e-06 - accuracy: 1.0000
Epoch 96/100
23/23 [==============================] - 0s 5ms/step - loss: 6.8104e-06 - accuracy: 1.0000
Epoch 97/100
23/23 [==============================] - 0s 5ms/step - loss: 6.5035e-06 - accuracy: 1.0000
Epoch 98/100
23/23 [==============================] - 0s 7ms/step - loss: 6.2321e-06 - accuracy: 1.0000
Epoch 99/100
23/23 [==============================] - 0s 6ms/step - loss: 5.9631e-06 - accuracy: 1.0000
Epoch 100/100
23/23 [==============================] - 0s 7ms/step - loss: 5.7322e-06 - accuracy: 1.0000
```

In [ ]:

```
y_pred_tf = model_tf.predict(x_test)
y_pred_tf = (y_pred_tf>0.5)
```

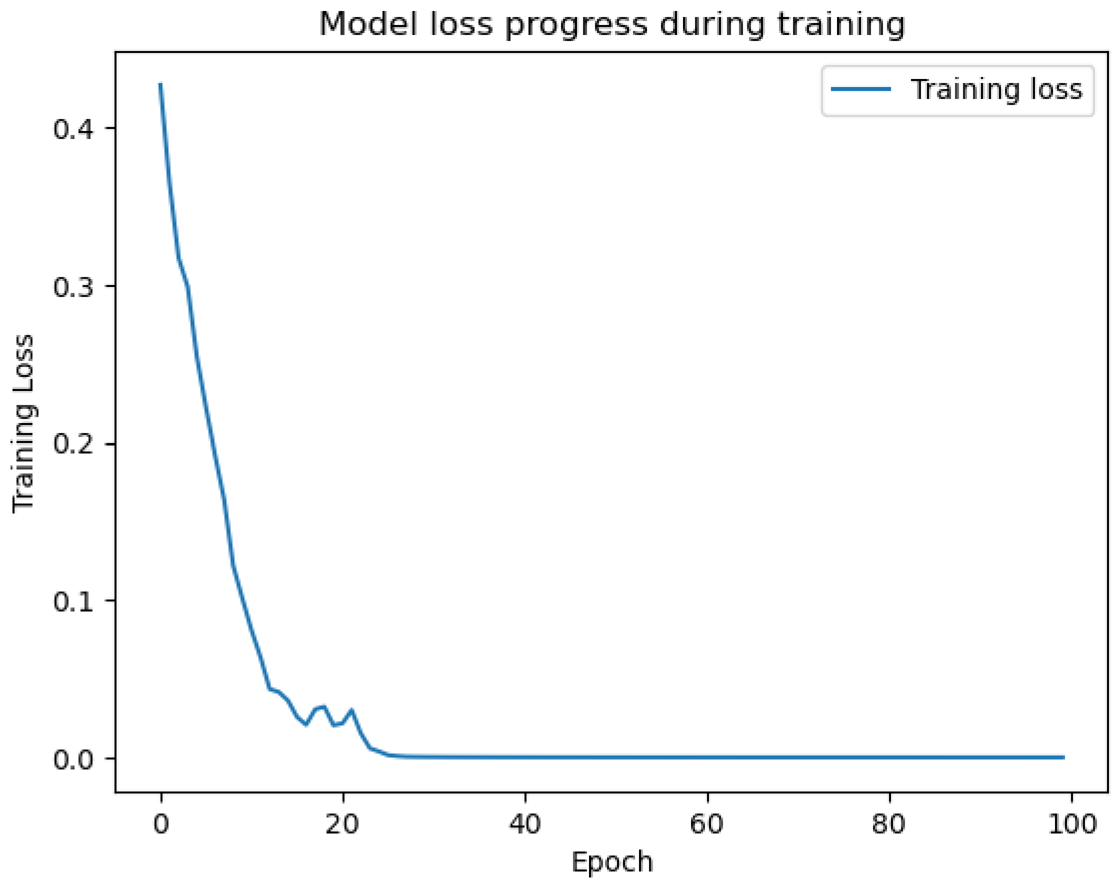
```
12/12 [==============================] - 0s 2ms/step
```

In [ ]:

```
plt.plot(epochs_hist.history['loss'])
plt.title('Model loss progress during training')
plt.xlabel('Epoch')
plt.ylabel('Training Loss')
plt.legend(['Training loss'])
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x2563b431700>
```

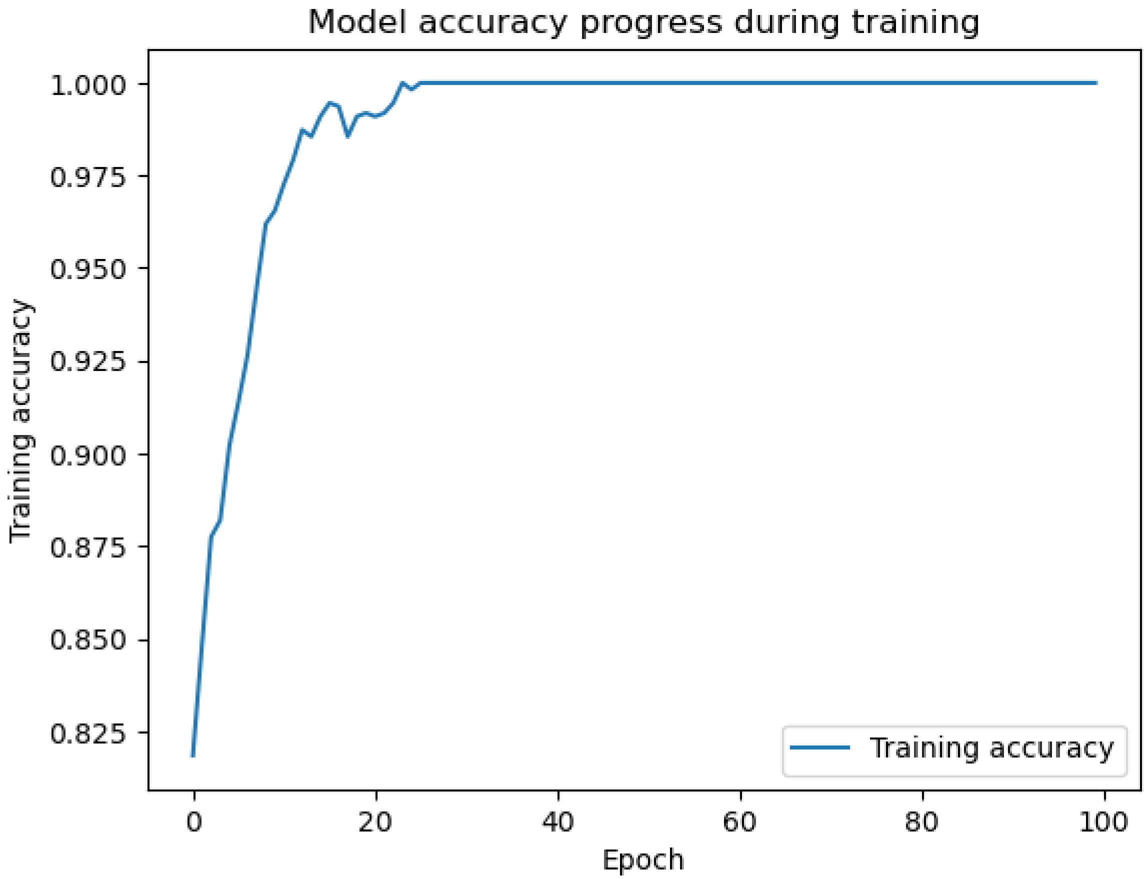


In [ ]:

```
plt.plot(epochs_hist.history['accuracy'])
plt.title('Model accuracy progress during training')
plt.xlabel('Epoch')
plt.ylabel('Training accuracy')
plt.legend(['Training accuracy'])
```

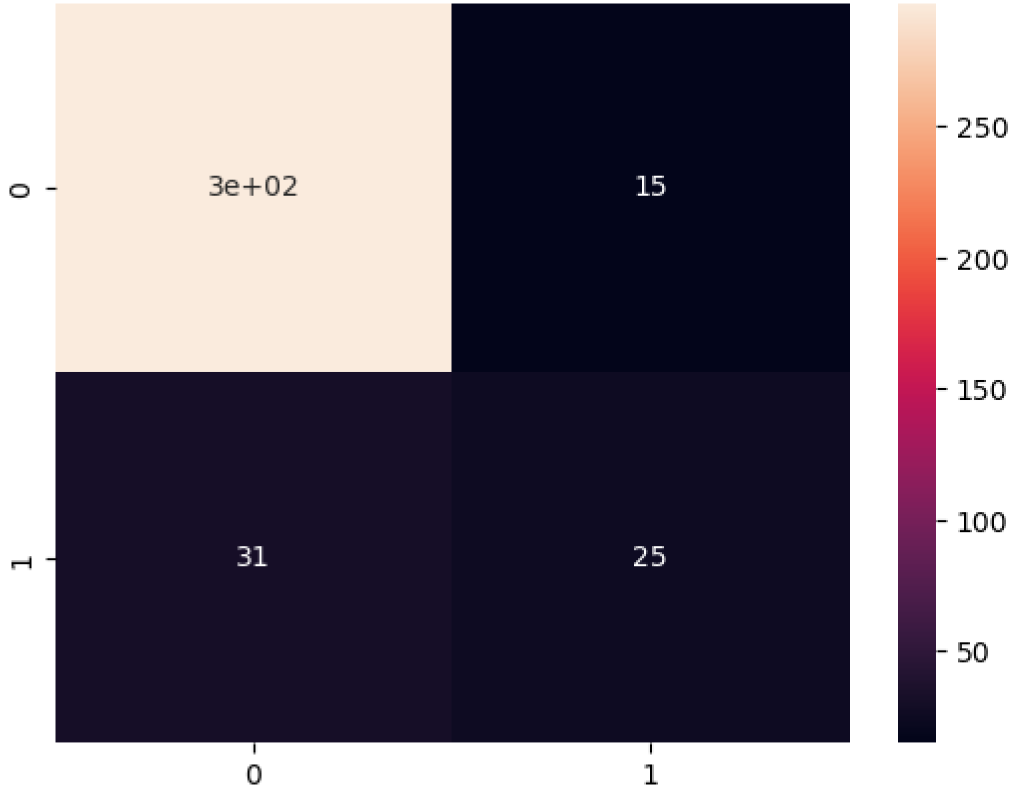
Out[ ]:

```
<matplotlib.legend.Legend at 0x2563b499550>
```



In [ ]:
```
cm_tf = confusion_matrix(y_test, y_pred_tf)
sns.heatmap(cm_tf, annot=True)
```

Out[ ]: <AxesSubplot:>



Viewing reports for each ML models

In [ ]:
```
#Neural Network Classification report
print(classification_report(y_test,y_pred_tf))
```

```
              precision    recall  f1-score   support

           0       0.91      0.95      0.93       312
           1       0.62      0.45      0.52        56

    accuracy                           0.88       368
   macro avg       0.77      0.70      0.72       368
weighted avg       0.86      0.88      0.87       368
```

In [ ]:
```
# Logistic Regression Classification report
print(classification_report(y_test,y_pred_LogRegression))
```

```
              precision    recall  f1-score   support

           0       0.91      0.98      0.94       312
           1       0.79      0.46      0.58        56

    accuracy                           0.90       368
   macro avg       0.85      0.72      0.76       368
weighted avg       0.89      0.90      0.89       368
```

In [ ]:

```
#Random Forest Classification report
print(classification_report(y_test,y_pred_randomForest))
```

```
              precision    recall  f1-score   support

           0       0.88      1.00      0.93       312
           1       0.92      0.21      0.35        56

    accuracy                           0.88       368
   macro avg       0.90      0.61      0.64       368
weighted avg       0.88      0.88      0.84       368
```

Conclusion

After analyzing Logistic Regression, Random Forest, and Neural Netowrk classifier, the precision for employees whom will stay in their respective company is high across all three models. However, the precision for employees who will leave is low and fluctuating often (except for random forest classification, but it has a low recall). This could mean that we will have possibly need more data from employee who has left their company to properly train the model to predict the employees who will leave the company.