HR - People Analytics Project

Solving HR Analytics and gathering information from the data.

The data is from Kaggle:

User: *PAVANSUBHASH*

Title: *IBM HR Analytics Employee Attrition & Performance*

Link: https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset

The purpose of the project is to practice my analytical skills with a real HR database, using my knowledge in ***Python, Excel, and Power BI***. Machine learning techniques will be applied where possible.

I will analyze the data to obtain valuable insights that allow actions to be taken related to HR. Also, an attempt will be made to create a comparative of the metrics, by creating data to simulate the passing years.

I will work with the file ***'WA_Fn-UseC_-HR-Employee-Attrition'***, downloaded from the Kaggle database.

Hypotheses will be raised, which must be confirmed or rejected by the data. The necessary dashboards will then be created to visualize the results of the hypotheses raised. Finally, the conclusions reached will be detailed.

1. Import libraries

In [ ]:
```
import pandas as pd
import numpy as np

# This is to ignore warnings. Was a recommendation from my friend Manuel Angel Rodr
import warnings
warnings.filterwarnings('ignore')
```

2. Importing our data file

It's time to import our file ***'WA_Fn-UseC_-HR-Employee-Attrition'*** to work with it. This is a .csv file. First, I will explore the data, clean it, and remove those values that are not necessary to our analysis

In [ ]:
```
df_rawdata = pd.read_csv('WA_Fn_UseC_HR_Employee_Attrition.csv')
df_rawdata.head(5)
```

Out[ ]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educa
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life
4	27	No	Travel_Rarely	591	Research & Development	2	1	

5 rows × 35 columns

In [ ]:
```
df_rawdata.columns
```

Out[ ]:
```
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')
```

In [ ]:
```
df_rawdata.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
```

3 Data Cleaning

In order to do data cleaning, I will remove the columns that I will not use for our analysis." However, prior to doing so, I will create a copy of the database "df_rawdata," allowing me to refer to the original database if necessary.

3.1 Copy the database

In [ ]:
```
# The copy will be called df_padb from dataframe peopleanalyticsdatabase
df_padb = df_rawdata.copy()
df_padb.head()
```

Out[ ]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	Educa
0	41	Yes	Travel_Rarely	1102	Sales	1	2	Life
1	49	No	Travel_Frequently	279	Research & Development	8	1	Life
2	37	Yes	Travel_Rarely	1373	Research & Development	2	2	
3	33	No	Travel_Frequently	1392	Research & Development	3	4	Life
4	27	No	Travel_Rarely	591	Research & Development	2	1	

5 rows × 35 columns

In [ ]:
```
# Checking for missing values
missing_values = df_padb.isnull().sum()
print('Number of missing values: ', missing_values)
```

```
Number of missing values:  Age                         0
Attrition                   0
BusinessTravel              0
DailyRate                   0
Department                  0
DistanceFromHome            0
Education                   0
EducationField              0
EmployeeCount               0
EmployeeNumber              0
EnvironmentSatisfaction     0
Gender                      0
HourlyRate                  0
JobInvolvement              0
JobLevel                    0
JobRole                     0
JobSatisfaction             0
MaritalStatus               0
MonthlyIncome               0
MonthlyRate                 0
NumCompaniesWorked          0
Over18                      0
OverTime                    0
PercentSalaryHike           0
PerformanceRating           0
RelationshipSatisfaction    0
StandardHours               0
StockOptionLevel            0
TotalWorkingYears           0
TrainingTimesLastYear       0
WorkLifeBalance             0
YearsAtCompany              0
YearsInCurrentRole          0
YearsSinceLastPromotion     0
YearsWithCurrManager        0
dtype: int64
```

In [ ]:
```
# Deleting columns that won't be used in the analysis
df_padb.drop(['BusinessTravel', 'DailyRate', 'EmployeeNumber', 'MaritalStatus', 'Nu
df_padb.columns
```

Out[ ]:
```
Index(['Age', 'Attrition', 'Department', 'DistanceFromHome', 'Education',
       'EducationField', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement', 'JobLevel', 'JobRole', 'JobSatisfaction',
       'MonthlyIncome', 'MonthlyRate', 'OverTime', 'TotalWorkingYears',
       'TrainingTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany',
       'YearsInCurrentRole', 'YearsSinceLastPromotion'],
      dtype='object')
```

3.2 Creating an ID Column

Using a two letter code for the 'Department' and a four random number combination, I will make an ID for every employee.

In [ ]:
```
# First let's take a look at how many departments we have in the database
departments = df_padb['Department'].value_counts()
print(departments)
```

```
Research & Development    961
Sales                     446
Human Resources            63
Name: Department, dtype: int64
```

To generate the two-letter code from the *'Department'* column, I will write a function. I will then use the four random integers to form an array. In order to deploy the values into the *'ID'* column later, I will merge the two values and save them in a variable.

In [ ]:
```
# Function to create the code values for the departments
def departments_code(departments):
    if 'Research & Development' in departments:
        return 'RD'
    elif 'Sales' in departments:
        return 'SL'
    else:
        return 'HR'

# Now is time to create our random numbers and add them to the TempID column
random_number = np.random.randint(1000, 9999, size=len(df_padb))

# Join the two values together
new_data = df_padb['Department'].apply(departments_code) + pd.Series(random_number)

# Inserting the new ID column with the values created lately
df_padb.insert(0, 'ID', new_data, True)

df_padb.head()
```

Out[ ]:

	ID	Age	Attrition	Department	DistanceFromHome	Education	EducationField	Environm
0	SL8842	41	Yes	Sales	1	2	Life Sciences	
1	RD9746	49	No	Research & Development	8	1	Life Sciences	
2	RD4085	37	Yes	Research & Development	2	2	Other	
3	RD8254	33	No	Research & Development	3	4	Life Sciences	
4	RD6169	27	No	Research & Development	2	1	Medical	

5 rows × 23 columns

In [ ]:
```
df_padb['ID'].dtype
```

Out[ ]:
```
dtype('O')
```

3.3 Replacing the code numbers from columns '*Education*', '*EnvironmentSatisfaction*', '*JobInvolvement*', '*JobSatisfaction*', '*PerformanceRating*', '*RelationshipSatisfaction*', *and* '*WorkLifeBalance*'

To improve the understanding of the data and make working with it easier, I'll change the code numbers in those columns. I'll convert the values in this task from integer to string format, then replace the values with their references.

In [ ]:
```
# Education column
education_ref = {
    1: 'Below College',
    2: 'College',
    3: 'Bachelor',
    4: 'Master',
    5: 'Doctor'
}

df_padb['Education'] = df_padb['Education'].map(education_ref)

# 'EnvironmentSatisfaction' column
environment_satisfaction_ref = {
    1: 'Low',
    2: 'Medium',
    3: 'High',
    4: 'Very High'
}

df_padb['EnvironmentSatisfaction'] = df_padb['EnvironmentSatisfaction'].map(enviror

# 'JobInvolvement' column
job_involvement_ref = {
    1: 'Low',
    2: 'Medium',
    3: 'High',
    4: 'Very High'
}

df_padb['JobInvolvement'] = df_padb['JobInvolvement'].map(job_involvement_ref)
```

```
# 'JobSatisfaction' column
job_satisfaction_ref = {
    1: 'Low',
    2: 'Medium',
    3: 'High',
    4: 'Very High'
}

df_padb['JobSatisfaction'] = df_padb['JobSatisfaction'].map(job_satisfaction_ref)

# 'WorkLifeBalance' column
wlb_ref = {
    1: 'Bad',
    2: 'Good',
    3: 'Better',
    4: 'Best'
}

df_padb['WorkLifeBalance'] = df_padb['WorkLifeBalance'].map(wlb_ref)

df_padb.head()
```

Out[ ]:

	ID	Age	Attrition	Department	DistanceFromHome	Education	EducationField	Environm
0	SL8842	41	Yes	Sales	1	College	Life Sciences	
1	RD9746	49	No	Research & Development	8	Below College	Life Sciences	
2	RD4085	37	Yes	Research & Development	2	College	Other	
3	RD8254	33	No	Research & Development	3	Master	Life Sciences	
4	RD6169	27	No	Research & Development	2	Below College	Medical	

5 rows × 23 columns

3.4 Export our data into an Excel spreadsheet

Now that the data has been cleansed and the values have been replaced with their references, it is time to export the database to an Excel workbook. I will also produce a.csv file just in case.

In [ ]:

```
# Exporting the data to an Excel spreadsheet
df_padb.to_excel('imb_analytics_2021.xlsx', sheet_name='hr_analytics_2021', index=F

# Exporting the data to a CSV file
df_padb.to_csv('ibm_hranalytics_2021.csv', index=False)
```

4 Working with the ISO 30414-2018

After the data has been cleansed, it's time to utilize the ISO and begin analyzing the information to provide HR insights.

4.1 DIVERSITY

To find out the composition and level of diversification of the firm, I will work using the information provided in the ISO.

4.1.a Creating the column with the age range

Before starting to work with the data, I'm going to create an age range to simplify the work and establish a better understanding of the different generations that make up our company

In [ ]:
```
# Function to create the age range
def age_range(age):
    if age >= 18 and age <= 27:
        return '18 to 27'
    elif age >= 28 and age <= 37:
        return '28 to 37'
    elif age >= 38 and age <= 47:
        return '38 to 47'
    elif age >= 48 and age <= 57:
        return '48 to 57'
    else:
        return 'more than 58'

# New data
new_age_data = df_padb['Age'].apply(lambda x: pd.Series(age_range(x)))
# print(new_age_data)

# Inserting the new column into the dataframe
df_padb.insert(loc=df_padb.columns.get_loc('Age')+1, column='AgeRange', value=new_a
df_padb.head()
```

Out[ ]:

	ID	Age	AgeRange	Attrition	Department	DistanceFromHome	Education	EducationField
0	SL8842	41	38 to 47	Yes	Sales	1	College	Life Sciences
1	RD9746	49	48 to 57	No	Research & Development	8	Below College	Life Sciences
2	RD4085	37	28 to 37	Yes	Research & Development	2	College	Othe
3	RD8254	33	28 to 37	No	Research & Development	3	Master	Life Sciences
4	RD6169	27	18 to 27	No	Research & Development	2	Below College	Medica

5 rows × 24 columns

4.1.b Analysing the data and creating the plots

With the age range column in place, let's review the data and identify the factors required to create the charts that illustrate the impact of diversity on our organization. I'm going to create a narrative for every one of these: *study field, study level, gender, and age range*. We will be able to see the variety of worker diversity inside our company as a result.

Gaining an understanding of the diversity of our business will help us create a more diverse workplace and set policies for future hiring.

I'll use matplotlib for the plots.

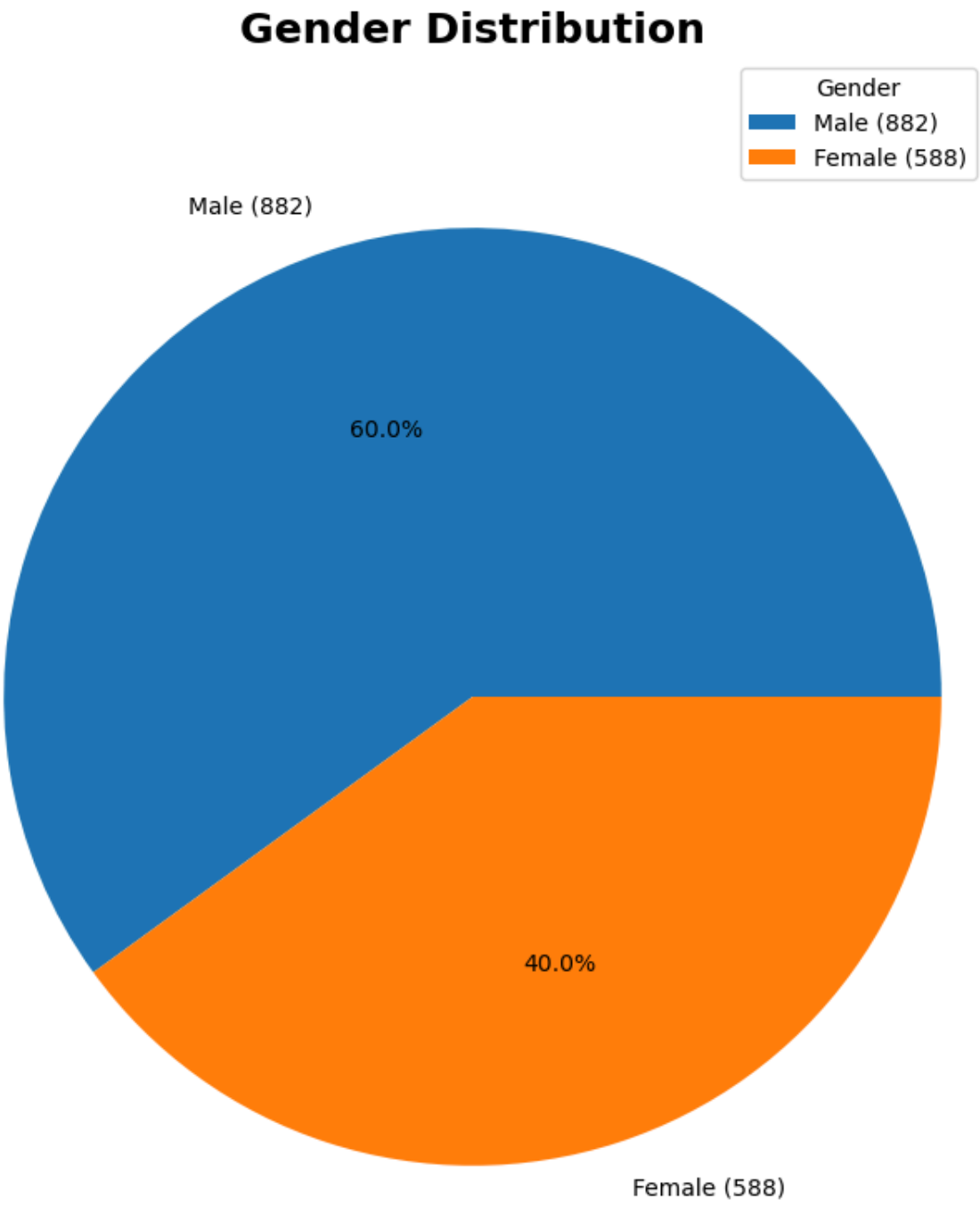
In [ ]:
```
import matplotlib.pyplot as plt
```

4.1.b.1 Gender

Let's now examine the gender distribution by looking at the *'Gender' column*. The gender distribution gives us an overview of how gender is spread throughout our business, which will be helpful for future recruitment efforts.

In [ ]:
```
# Counting the gender values and storage them into a variable
gender_counts = df_padb['Gender'].value_counts()
print(gender_counts)

# Total and percentage variables to use on our charts
total_count = gender_counts.sum()
gender_percentage = (gender_counts / total_count) * 100

# labels = gender_counts.index
# Using a function to create the labels
labels = [f'{gender} ({count})' for gender, count in zip(gender_counts.index, gende

# Chart size
fig, ax = plt.subplots(figsize=(8, 10))

# Chart generation
plt.pie(gender_counts, labels=labels, autopct='%1.1f%%')
plt.legend(title='Gender')
ax.set_title('Gender Distribution', fontsize=18, fontweight='bold')
plt.axis('equal')
plt.show()
```

```
Male      882
Female    588
Name: Gender, dtype: int64
```



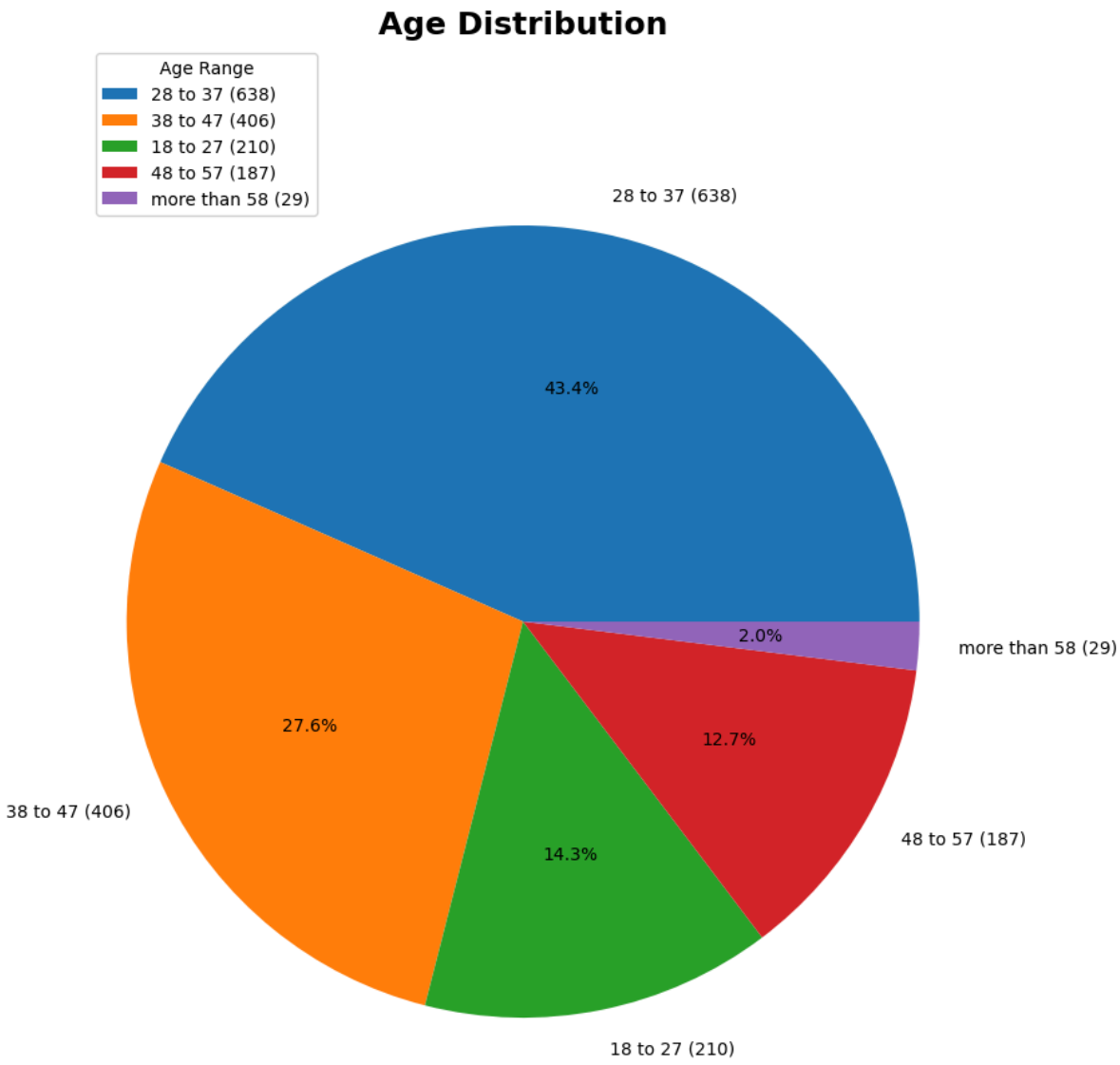
4.1.b.2 Age Distribution

Continuing with our analysis, we will now examine how our company is composed by ages. To do this, we will use the values in the *column 'AgeRange'*. We will be able to know the composition of our company according to the age ranges. This analysis is useful to know if we have several employees near retirement.

In [ ]:
```
age_range_count = df_padb['AgeRange'].value_counts()
print(age_range_count)

total_age_count = age_range_count.sum()
age_percentage = (age_range_count / total_age_count) * 100

labels = [f'{age} ({count})' for age, count in zip(age_range_count.index, age_range
```

```
fig, ax = plt.subplots(figsize=(9, 12))


plt.pie(age_range_count, labels=labels, autopct='%1.1f%%')
plt.legend(title='Age Range', loc= 'upper left')
ax.set_title('Age Distribution', fontsize=18, fontweight='bold')
plt.axis('equal')
plt.show()
```

```
28 to 37        638
38 to 47        406
18 to 27        210
48 to 57        187
more than 58     29
Name: AgeRange, dtype: int64
```



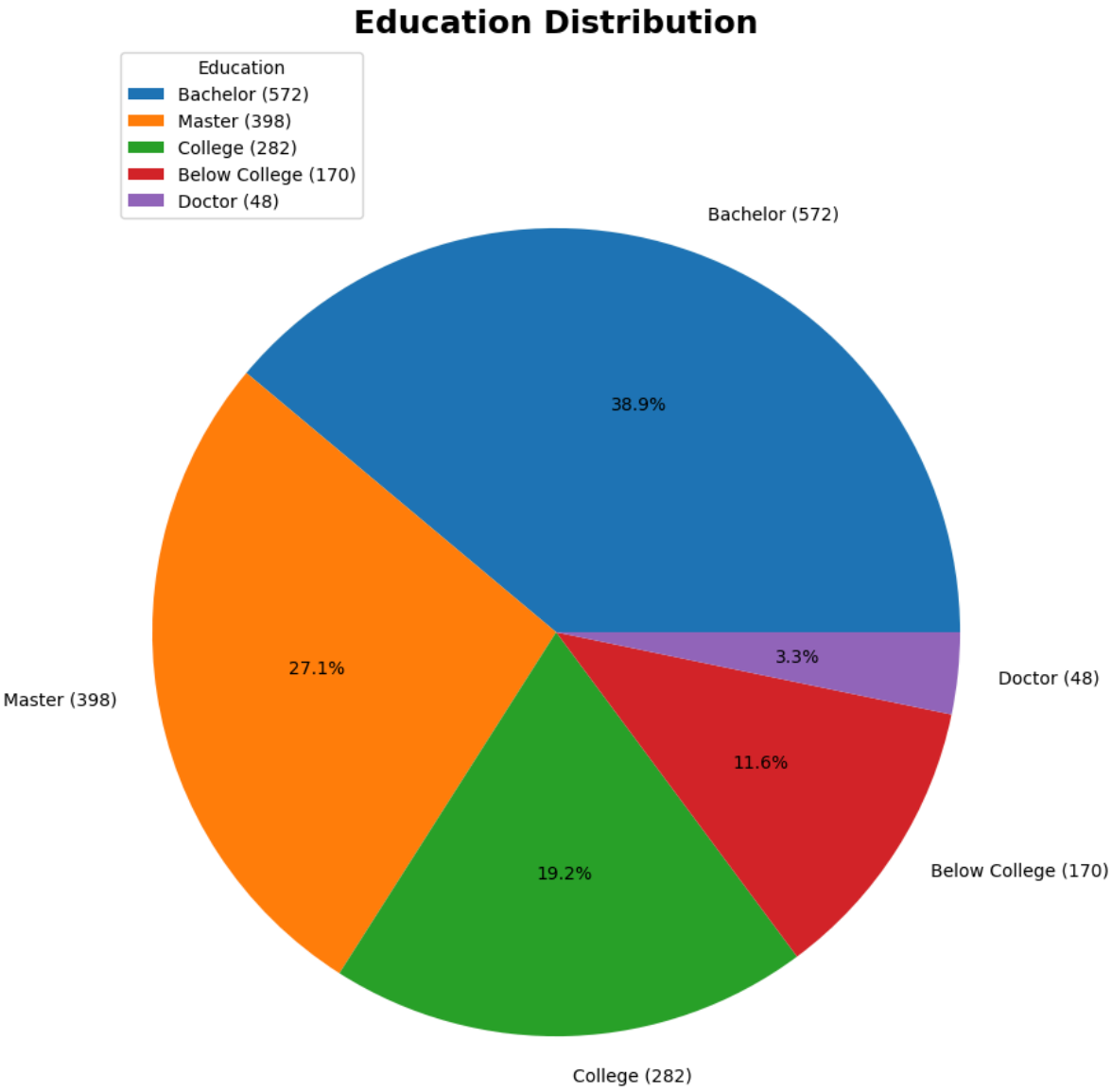
4.1.b.3 More Distributions

What level of training and education do our employees have? I would like to know if workers need assistance in improving their academic achievement. For this task, the columns "*Education*" and "*EducationField*" will be used by me.

4.1.b.3.a Education Distribution

In [ ]:
```
education_count = df_padb['Education'].value_counts()
print(education_count)

total_education_count = education_count.sum()
education_percentage = (education_count / total_education_count) * 100

labels = [f'{education} ({count})' for education, count in zip(education_count.inde

fig, ax = plt.subplots(figsize=(9, 12))


plt.pie(education_count, labels=labels, autopct='%1.1f%%')
plt.legend(title='Education', loc= 'upper left')
ax.set_title('Education Distribution', fontsize=18, fontweight='bold')
plt.axis('equal')
plt.show()
```

```
Bachelor         572
Master           398
College          282
Below College    170
Doctor            48
Name: Education, dtype: int64
```

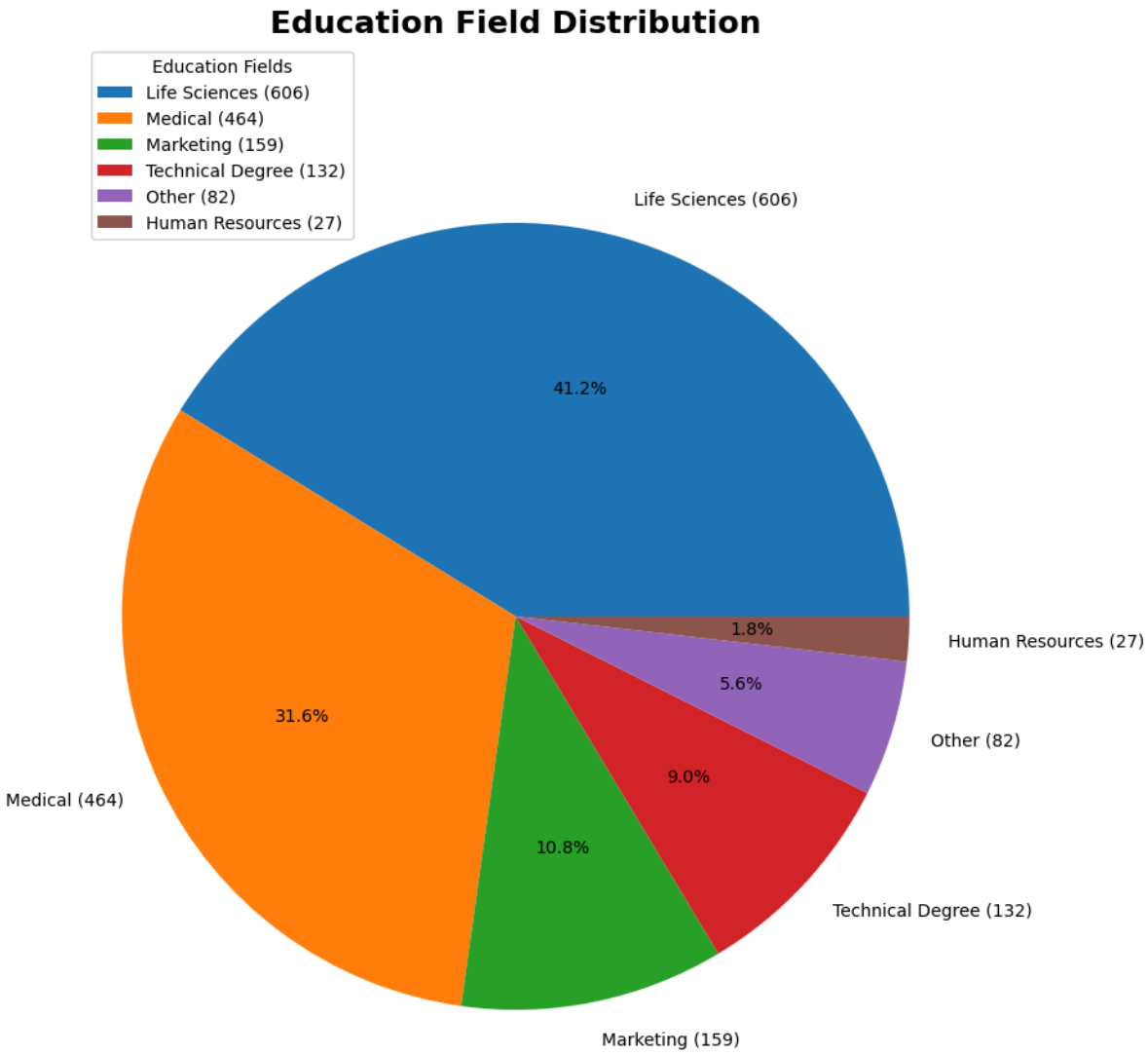


4.1.b.3.b Field Education Distribution

The objective of the subsequent analysis is to determine how people are distributed according to the different educational qualifications they have.

In [ ]:
```
educationfield_count = df_padb['EducationField'].value_counts()
print(educationfield_count)

total_educationfield_count = educationfield_count.sum()
educationfield_percentage = (educationfield_count / total_educationfield_count) * 1

labels = [f'{education} ({count})' for education, count in zip(educationfield_count

fig, ax = plt.subplots(figsize=(9, 12))


plt.pie(educationfield_count, labels=labels, autopct='%1.1f%%')
plt.legend(title='Education Fields', loc= 'upper left')
ax.set_title('Education Field Distribution', fontsize=18, fontweight='bold')
plt.axis('equal')
plt.show()
```

```
Life Sciences       606
Medical             464
Marketing           159
Technical Degree    132
Other                82
Human Resources      27
Name: EducationField, dtype: int64
```



4.1.c Appendix

I compute certain numbers in this appendix for use in upcoming analyses. Recall that in the future, we want to *compare and understand how diversity evolves throughout time*.

In [ ]:
```
# Calculate the employees average age
age_info = df_padb['Age'].describe()
print(age_info)

age_sum = df_padb['Age'].mean().round(0)
print("\nThe average Age is:")
print(age_sum)
```

```
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

The average Age is:
37.0
```

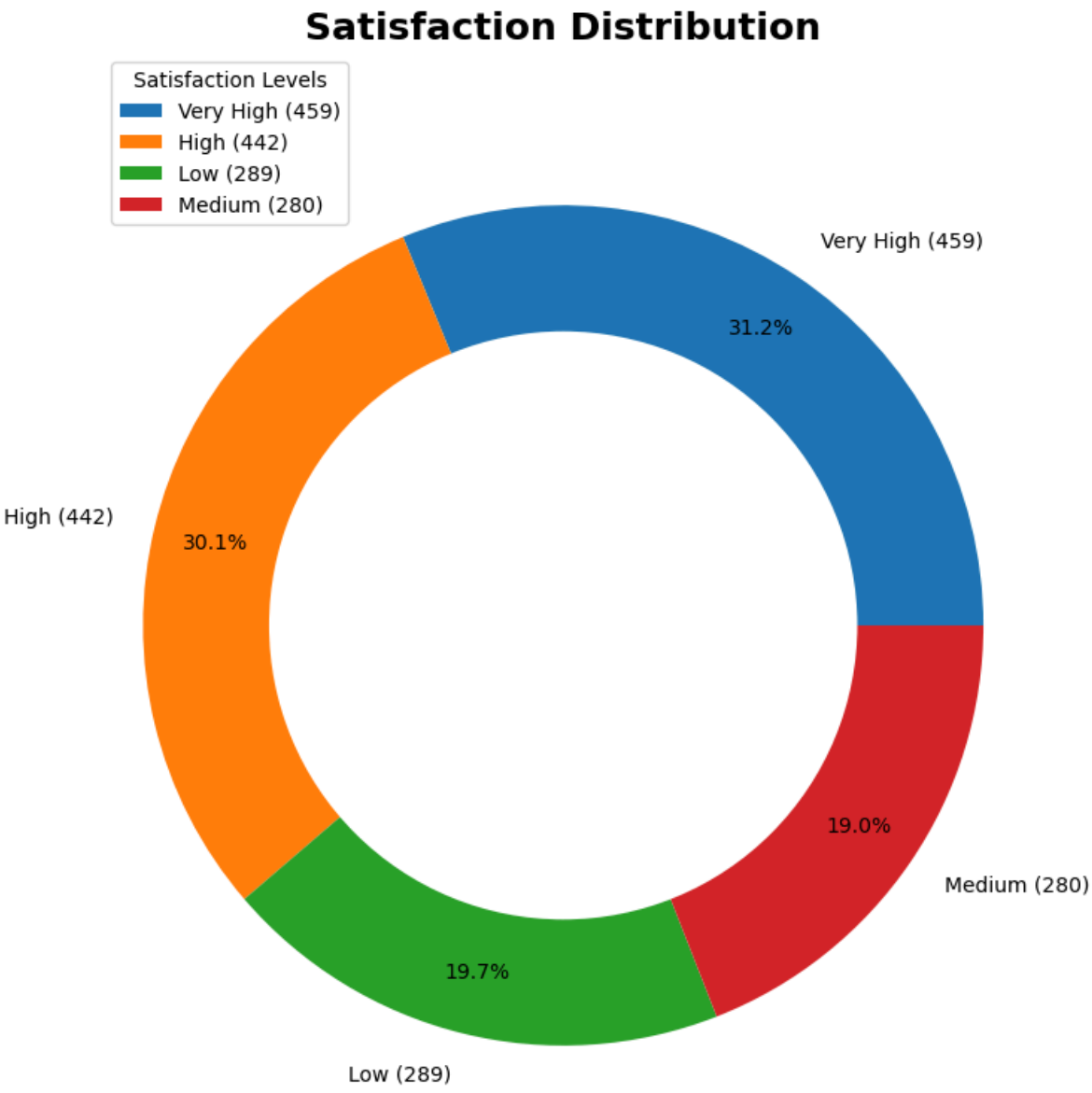
4.2 JOB SATISFACTION

Let's examine employee's satisfaction levels across the company's different roles. Which role has the greatest or lowest degree of contentment? With this metric, we can identify those roles where *satisfaction is **low***, discover what is causing the low levels, and design or create strategies to improve it.

I'll start by tallying the distribution of the satisfaction categories.

In [ ]:
```
# Count the satisfaction categories.
job_satisfaction_count = df_padb['JobSatisfaction'].value_counts()
print(job_satisfaction_count)

job_satisfaction_total = df_padb['JobSatisfaction'].sum()

labels = [f'{jobsatisfaction} ({count})' for jobsatisfaction, count in zip(job_sati

fig, ax = plt.subplots(figsize=(8, 10))

plt.pie(job_satisfaction_count, labels=labels, autopct='%1.1f%%', pctdistance=0.85)
plt.legend(title='Satisfaction Levels', loc= 'upper left')
ax.set_title('Satisfaction Distribution', fontsize=18, fontweight='bold')
plt.axis('equal')
# draw circle
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
fig = plt.gcf()

# Adding Circle in Pie chart
fig.gca().add_artist(centre_circle)
plt.show()
```

```
Very High    459
High         442
Low          289
Medium       280
Name: JobSatisfaction, dtype: int64
```



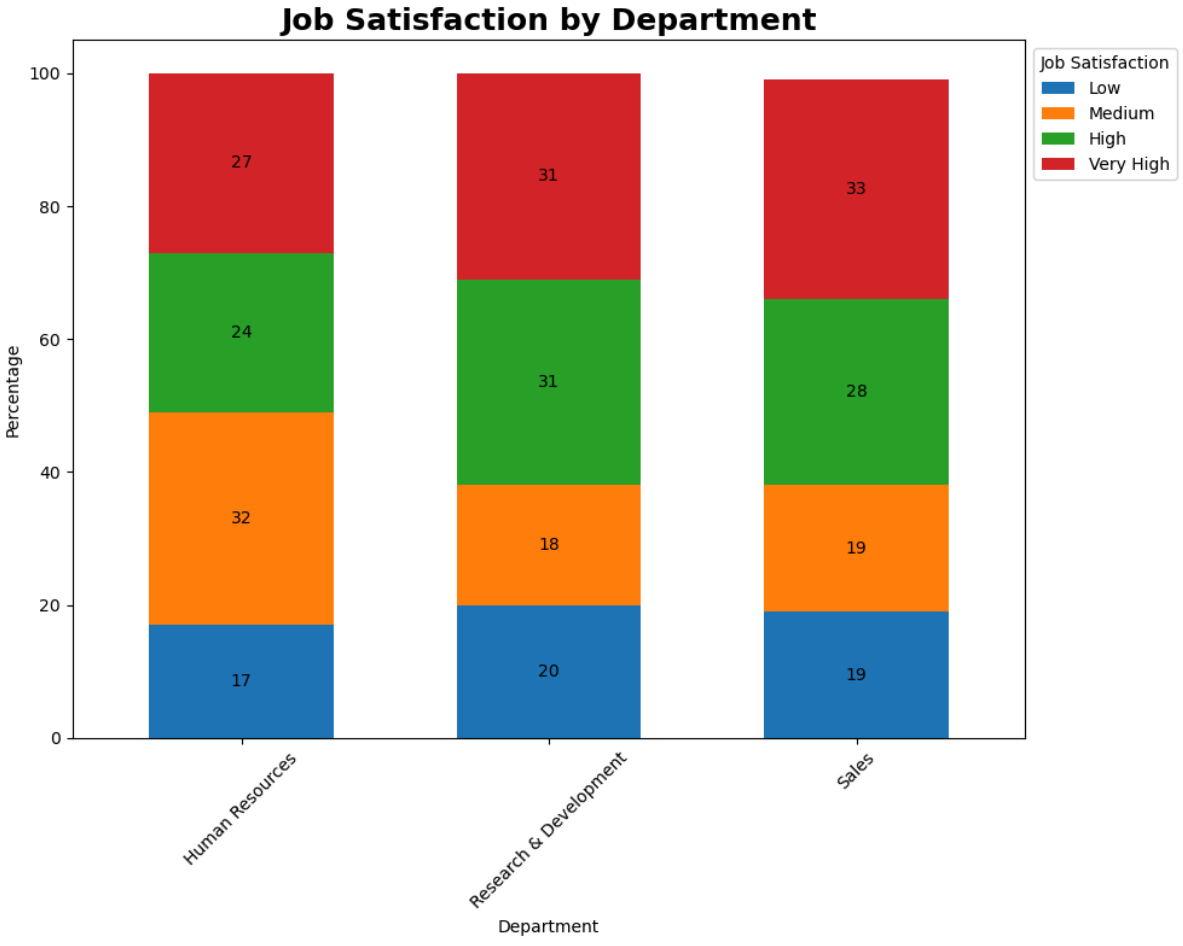
The data shows that ***19,7%*** of the employees have Low satisfaction with their job. I will identify which of the roles are not so satisfied with their job

In [ ]:
```
# With a pivot table, I will identify the responses for each department about their
pivot = pd.pivot_table(df_padb[['Department', 'JobSatisfaction']], index='Departmer
print("Pivot Table with Counts:")
print(pivot)

# Let's calculate the percentage of satisfaction for each department and their cate
pivot_percentage = pivot.div(pivot.sum(axis=1), axis=0).round(2) * 100

# Define the desired order of values
desired_order = ['Low', 'Medium', 'High', 'Very High']

# Reindex the DataFrame to specify the desired order
pivot_percentage_ordered = pivot_percentage.reindex(desired_order, axis=1)

print("\nPivot Table with Percentages:")
print(pivot_percentage)
```

```
Pivot Table with Counts:
JobSatisfaction         High  Low  Medium  Very High
Department
Human Resources           15   11      20         17
Research & Development   300  192     174        295
Sales                    127   86      86        147

Pivot Table with Percentages:
JobSatisfaction         High   Low  Medium  Very High
Department
Human Resources         24.0  17.0    32.0       27.0
Research & Development  31.0  20.0    18.0       31.0
Sales                   28.0  19.0    19.0       33.0
```

Let's create a chart to visualize our data.

In [ ]:
```python
# Creating the chart to visualize our data
ax = pivot_percentage_ordered.plot(kind='bar', stacked=True, figsize=(10, 8), width

# Adding the values to the bars
for container in ax.containers:
    ax.bar_label(container, label_type='center', fontsize=10)

plt.title('Job Satisfaction by Department', fontsize=18, fontweight='bold')
plt.xlabel('Department')
plt.ylabel('Percentage')
plt.xticks(rotation=45)
plt.legend(title='Job Satisfaction', bbox_to_anchor=(1, 1), loc='upper left')
plt.tight_layout()
plt.show()
```



Previously, looking at the overall percentage among departments, we found that 19.7% of all employees have a low motivation with their work.

In order to have a more detailed report, I proceeded to break down by department in order to know more deeply the dissatisfaction with the work. We have to keep in mind that out of *1470 employees, 63 belong to the HRD department, 961 to the Research and Development department and 446 to Sales*. The percentages of dissatisfaction between the three departments were very similar. *RRHH* presented a **17%** about 11 employees, *R&D* a total of **20%** about 192 employees and *Sales* total of **19%** about 86 employees.

We can indicate that in the *R&D department* we have the largest number of employees who are dissatisfied or low motivated with their work. It is recommended to investigate what is causing this and find some solutions.

I would like to make an assessment: *in RRHH there are 17% of employees with low motivation and 32% with medium motivation*. It is recommended to follow up to see if these indices are rising because we would have most half of the department with a low motivation with their tasks. While the other two departments showed high rates of average and high satisfaction with their work.

4.2.a Appendix

In order to further my investigation, I choose to find out how many of them are **"Very High"** invested in their work and what proportion of them have **"Low"** motivation. I will identify the employees who exhibit **"Low"** motivation by utilizing the '*JobInvolvement*' column. Those with **"High"** participation and **"Low"** satisfaction will also be examined by me.

In [ ]:
```
# Create a copy of the columns to work better with them
job_db = df_padb[['JobInvolvement', 'JobSatisfaction']].copy()

# Filtering the data
jobFilteredData_VHL = job_db[(job_db['JobInvolvement'] == 'Very High') & (job_db['J
jobFilteredData_HL = job_db[(job_db['JobInvolvement'] == 'High') & (job_db['JobSati

# Counting the filtered data.
countVH_Low = jobFilteredData_VHL.shape[0]
countH_Low = jobFilteredData_HL.shape[0]
print("The total number of employees with Very High Job Involvement and Low Job Sat
print("The total number of employees with High Job Involvement and Low Job Satisfac

# Percentage they represent
percentageVH_Low = countVH_Low / len(df_padb) * 100
percentageH_Low = countH_Low / len(df_padb) * 100
print('The employees with Very High Job Involvement and Low Job Satisfaction repres
print('The employees with High Job Involvement and Low Job Satisfaction represents
```

```
The total number of employees with Very High Job Involvement and Low Job Satisfact
ion is:  34
The total number of employees with High Job Involvement and Low Job Satisfaction i
s:  166
The employees with Very High Job Involvement and Low Job Satisfaction represents a
percentage of:  2.31
The employees with High Job Involvement and Low Job Satisfaction represents a perc
entage of:  11.29
```

4.3 CAREER PROGRESSION

Is there decent career growth at the company? Through data analysis, my goal is to ascertain whether the organization provides its employees with a decent opportunity for professional

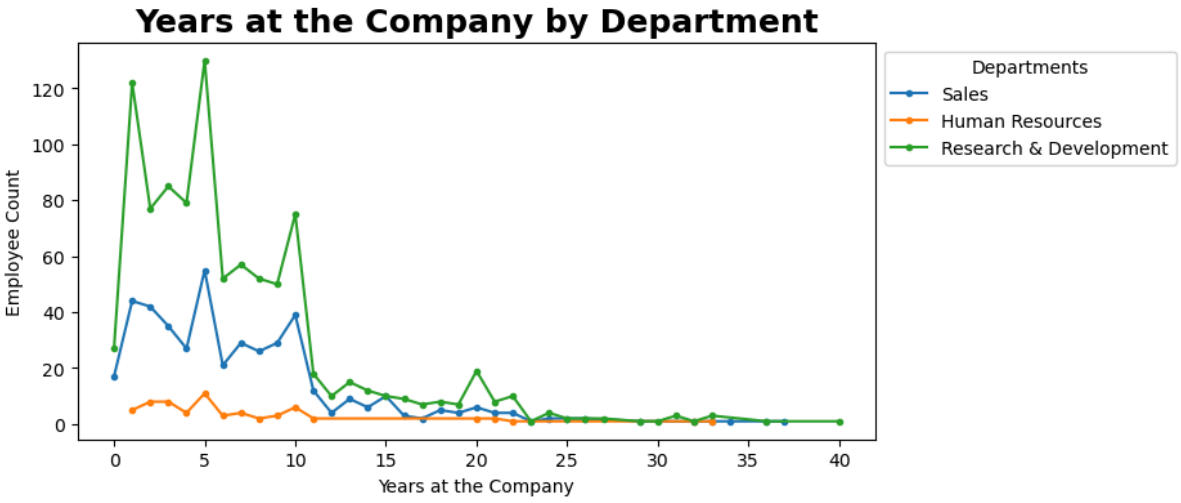
advancement. Can this be related to low work satisfaction?. I will use for the analysis the data from the column 'YearsAtCompany'.

In [ ]:
```
df_total_records = len(df_padb)
print(df_total_records)
```

```
1470
```

In [ ]:
```
departments_count = df_padb['Department'].value_counts()
print('By department there are the following number of employees: ', "\n", departme
# Preparing the data
sales_count = df_padb[df_padb['Department'] == 'Sales'].groupby('YearsAtCompany').s
rrhh_count = df_padb[df_padb['Department'] == 'Human Resources'].groupby('YearsAtCo
rd_count = df_padb[df_padb['Department'] == 'Research & Development'].groupby('Year

# Creating the line plot
plt.figure(figsize=(8, 4))

# Ploting the lines
plt.plot(sales_count.index, sales_count.values, label='Sales', marker='o', ms = 3)
plt.plot(rrhh_count.index, rrhh_count.values, label='Human Resources', marker='o',
plt.plot(rd_count.index, rd_count.values, label='Research & Development', marker='o

# Adding Labels
plt.xlabel('Years at the Company')
plt.ylabel('Employee Count')
plt.title('Years at the Company by Department', fontsize=18, fontweight='bold')
plt.legend(title='Departments', bbox_to_anchor=(1, 1), loc='upper left')
plt.show()
```

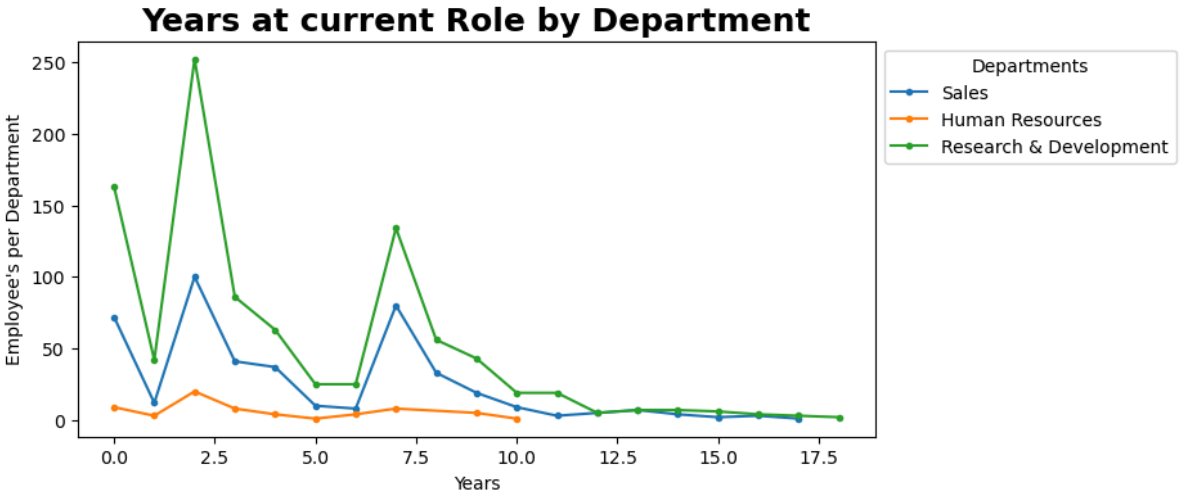
```
By department there are the following number of employees:
 Research & Development    961
Sales                      446
Human Resources             63
Name: Department, dtype: int64
```



Let's examine how many years the employee's stay in the same role in their departments.

In [ ]:
```
# Preparing the data
sales_count_cr = df_padb[df_padb['Department'] == 'Sales'].groupby('YearsInCurrentR
rrhh_count_cr = df_padb[df_padb['Department'] == 'Human Resources'].groupby('YearsI
rd_count_cr = df_padb[df_padb['Department'] == 'Research & Development'].groupby('Y

# Creating the line plot
plt.figure(figsize=(8, 4))

# Ploting the lines
```

```
plt.plot(sales_count_cr.index, sales_count_cr.values, label='Sales', marker='o', ms
plt.plot(rrhh_count_cr.index, rrhh_count_cr.values, label='Human Resources', marker
plt.plot(rd_count_cr.index, rd_count_cr.values, label='Research & Development', mar

# Adding Labels
plt.xlabel('Years')
plt.ylabel("Employee's per Department")
plt.title('Years at current Role by Department', fontsize=18, fontweight='bold')
plt.legend(title='Departments', bbox_to_anchor=(1, 1), loc='upper left')
plt.show()
```



In [ ]:
```
average_years = df_padb['YearsAtCompany'].mean().round()
average_years_at_role = df_padb['YearsInCurrentRole'].mean().round()
average_years_promotion = df_padb['YearsSinceLastPromotion'].mean().round()
max_years_in_role = df_padb['YearsInCurrentRole'].max()

print(
    "Average years at the Company: ", average_years, "\n",
    "Average years at a Role: ", average_years_at_role, "\n",
    "Average Years since last Promotion: ", average_years_promotion, "\n",
    "Max Years in a Role: ", max_years_in_role)
# print(max_years_in_role)
```

```
Average years at the Company:  7.0
 Average years at a Role:  4.0
 Average Years since last Promotion:  2.0
 Max Years in a Role:  18
```

Most employees are *between 0 and 10 years working for the company*. And we can see that it decreases significantly after ten years. So I tried to figure out *how long the average employee stays in the company* and found out that it's 7 years, with *4 years on average working in the same position*. Then I decided to investigate the company's promotion system and found out that our company has an average of two years to grant promotions.

A first promotion can be seen *around 2 years* after joining the company, and then a second promotion can be seen *5 years* later. Then there may be a change of business, or the person may continue to work in the position until his retirement.

For more information, the maximum number of years in a single role was examined. It was found that the *longest duration in a position is **18 years***.

4.4 COMPENSATION ANALYSIS

It's time to analyze *whether there is a significant disparity between* ***employee salaries*** *and* ***educational attainment***. '*MonthlyIncome*' values will be used, and the values will be distributed according to the values in the '*Education*' column.

In [ ]:
```
# To facilitate our work, I will create a table with the required columns 'MonthlyI
df_monthedu = df_padb[['MonthlyIncome', 'Education']].copy()
df_monthedu.head()
```

Out[ ]:

	MonthlyIncome	Education
0	5993	College
1	5130	Below College
2	2090	College
3	2909	Master
4	3468	Below College

It's time to work with our new data frame. Let us calculate the median for every category of education.

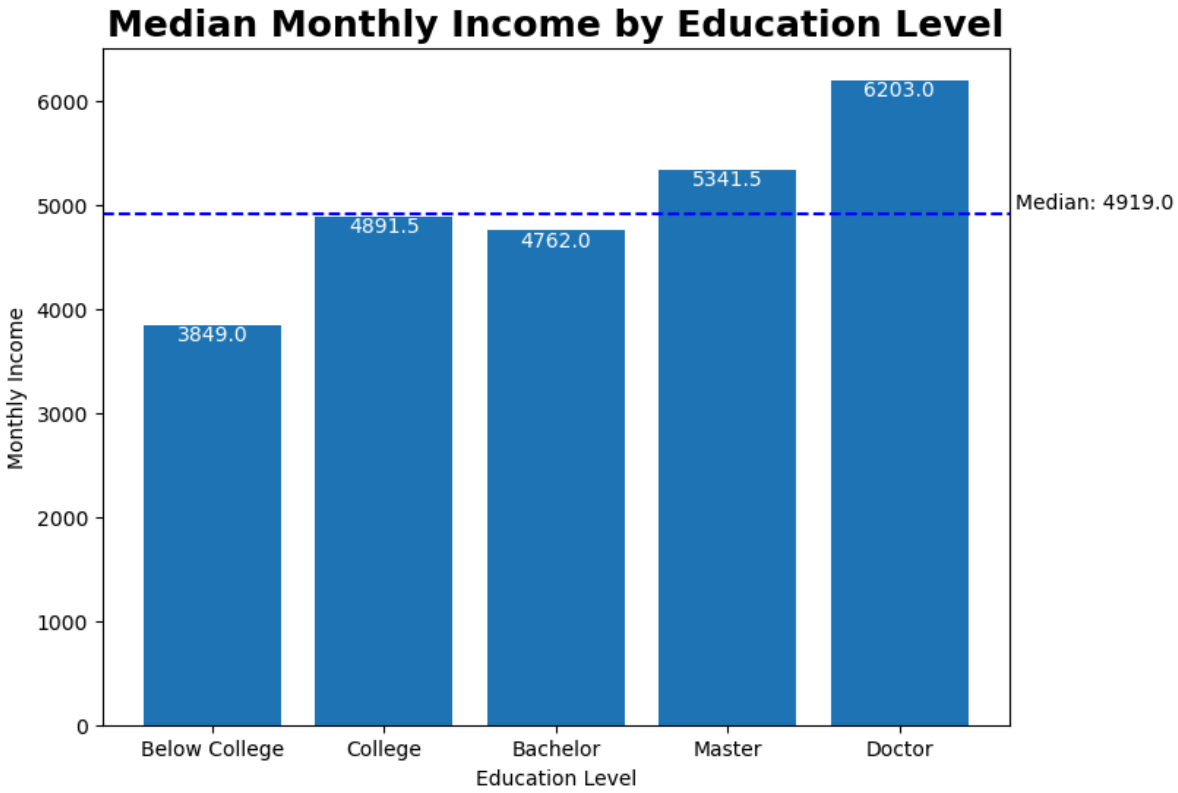
In [ ]:
```
# First, let's calculate the median from the 'MonthlyIncome' column.
monthlyIncome_median = df_monthedu['MonthlyIncome'].median()

# Second, let's calculate the median for each category of education
below_college_median = df_monthedu[df_monthedu['Education'] == 'Below College'].gro
college_median = df_monthedu[df_monthedu['Education'] == 'College'].groupby('Educat
bachelor_median= df_monthedu[df_monthedu['Education'] == 'Bachelor'].groupby('Educa
master_median = df_monthedu[df_monthedu['Education'] == 'Master'].groupby('Educatio
doctor_median = df_monthedu[df_monthedu['Education'] == 'Doctor'].groupby('Educatio
print(
    "The median for Below College Education is: ", below_college_median.values, "\r
    "The median for College Education is: ", college_median.values, "\n",
    "The median for Bachelor Education is: ", bachelor_median.values, "\n",
    "The median for Master Education is: ", master_median.values, "\n",
    "The median for Doctor Education is: ", doctor_median.values, "\n",
    'The median for the "MonthlyIncome" column is: ', monthlyIncome_median )
```

```
The median for Below College Education is:  [[3849.]]
 The median for College Education is:  [[4891.5]]
 The median for Bachelor Education is:  [[4762.]]
 The median for Master Education is:  [[5341.5]]
 The median for Doctor Education is:  [[6203.]]
 The median for the "MonthlyIncome" column is:  4919.0
```

To see our data, let's make some charts. I will use a ***Bar chart*** to compare the monthly income median of each education level.

In [ ]:
```
# Let's create the Bar chart
plt.figure(figsize = (8, 6))

# Adding the variables
bar_data = [below_college_median['MonthlyIncome'].values[0],
            college_median['MonthlyIncome'].values[0],
            bachelor_median['MonthlyIncome'].values[0],
            master_median['MonthlyIncome'].values[0],
            doctor_median['MonthlyIncome'].values[0]]
education_cat = ['Below College', 'College', 'Bachelor', 'Master', 'Doctor']
```

```
# Adding the labels
for i, value in enumerate(bar_data):
    plt.text(i, value, str(value), ha='center', va='top', color='white')

# Add the average line
plt.axhline(monthlyIncome_median, color='blue', linestyle='--', label='Media')

# Add the value of the average to the line
plt.text(len(education_cat) + 0.6, monthlyIncome_median, f'Median: {monthlyIncome_m

# Joining the data to create the chart
plt.bar(education_cat, bar_data)
plt.xlabel('Education Level')
plt.ylabel('Monthly Income')
plt.title('Median Monthly Income by Education Level', fontsize=18, fontweight='bold
plt.show()
```



Employees with a *Bachelor's degree* have less income than those with a *College degree*, according to the findings when viewed using the median. But keep in mind that we have more employees with *Bachelor's degrees* than the *college* does, so I'll check to see if the mean exhibits the same abnormality.

In [ ]:

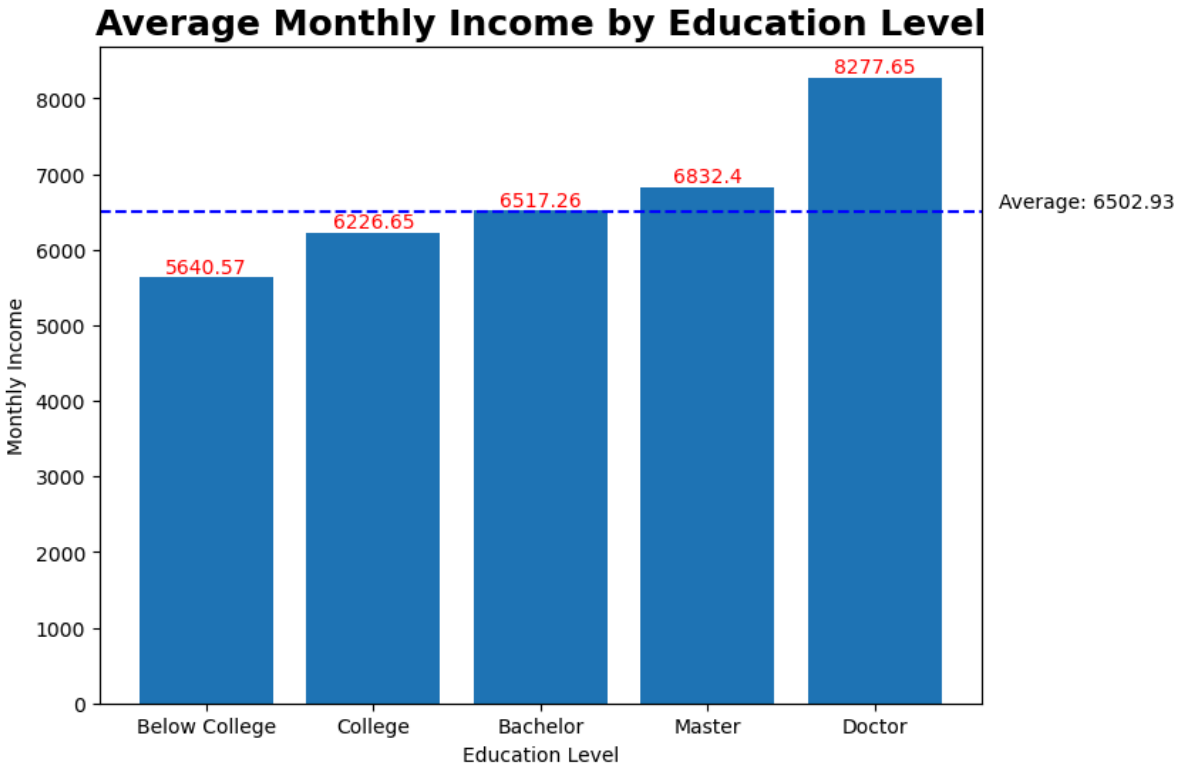
```
# First, let's calculate the mean from the 'MonthlyIncome' column.
monthlyIncome_mean = df_monthedu['MonthlyIncome'].mean().round(2)

# Second, let's calculate the mean for each category of education
below_college_mean = df_monthedu[df_monthedu['Education'] == 'Below College'].group
college_mean = df_monthedu[df_monthedu['Education'] == 'College'].groupby('Educatio
bachelor_mean= df_monthedu[df_monthedu['Education'] == 'Bachelor'].groupby('Educati
master_mean = df_monthedu[df_monthedu['Education'] == 'Master'].groupby('Education'
doctor_mean = df_monthedu[df_monthedu['Education'] == 'Doctor'].groupby('Education'
print(
    "The mean for Below College Education is: ", below_college_mean.values, "\n",
    "The mean for College Education is: ", college_mean.values, "\n",
    "The mean for Bachelor Education is: ", bachelor_mean.values, "\n",
```

```
    "The mean for Master Education is: ", master_mean.values, "\n",
    "The mean for Doctor Education is: ", doctor_mean.values, "\n",
    'The mean for the "MonthlyIncome" column is: ', monthlyIncome_mean )
```

```
The mean for Below College Education is:  [[5640.57]]
 The mean for College Education is:  [[6226.65]]
 The mean for Bachelor Education is:  [[6517.26]]
 The mean for Master Education is:  [[6832.4]]
 The mean for Doctor Education is:  [[8277.65]]
 The mean for the "MonthlyIncome" column is:  6502.93
```

I can establish that employees with a *Bachelor's* degree have more income than those with a *College* degree by looking at the mean. By the time, there is no problem with the salaries, but it's crucial to note that certain *Bacherlor* employees are not getting paid enough. The mean for the *Bachelor's degree* is **6517,26** against the **6226,65** from the *College degree*.

In [ ]:
```
# Let's create the Bar chart
plt.figure(figsize = (8, 6))

# Adding the variables
bar_mean_data = [below_college_mean['MonthlyIncome'].values[0],
            college_mean['MonthlyIncome'].values[0],
            bachelor_mean['MonthlyIncome'].values[0],
            master_mean['MonthlyIncome'].values[0],
            doctor_mean['MonthlyIncome'].values[0]]
education_cat = ['Below College', 'College', 'Bachelor', 'Master', 'Doctor']

# Adding the labels
for i, value in enumerate(bar_mean_data):
    plt.text(i, value, str(value), ha='center', va='bottom', color='red')

# Add the average line
plt.axhline(monthlyIncome_mean, color='blue', linestyle='--', label='Media')

# Add the value of the average to the line
plt.text(len(education_cat) + 0.8, monthlyIncome_mean, f'Average: {monthlyIncome_me

# Joining the data to create the chart
plt.bar(education_cat, bar_mean_data)
plt.xlabel('Education Level')
plt.ylabel('Monthly Income')
plt.title('Average Monthly Income by Education Level', fontsize=18, fontweight='bol
plt.show()
```



4.4 TRAINING AND DEVELOPMENT

Continuing with our descriptive analysis, I will explore the years the company has committed to providing training to their employees in each department. I will make an effort to identify potential areas to improve the training and which departments could benefit more.

In [ ]:
```
# To facilitate our work, I will create a table with the required columns 'Monthly1
df_timesTraining = df_padb[['Department', 'TrainingTimesLastYear']].copy()
df_timesTraining.head()
```

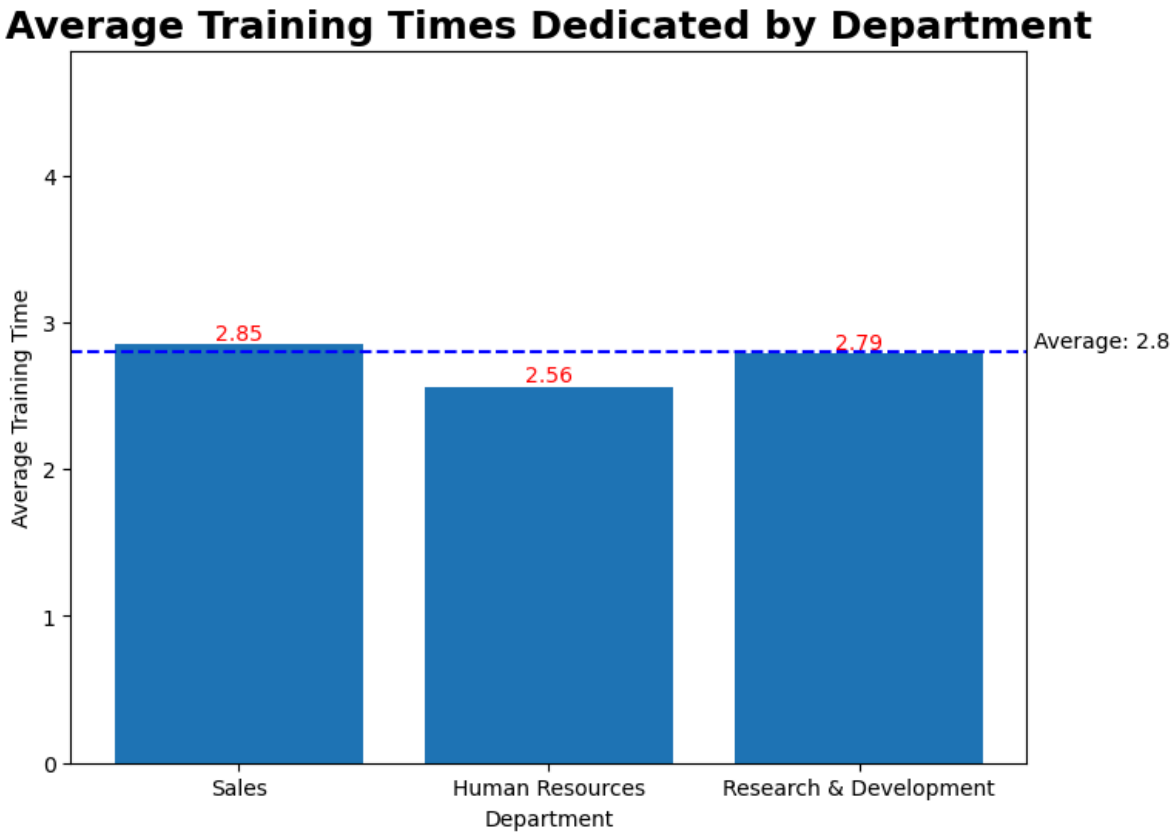
Out[ ]:

	Department	TrainingTimesLastYear
0	Sales	0
1	Research & Development	3
2	Research & Development	3
3	Research & Development	3
4	Research & Development	3

In [ ]:
```
timesTraining_average = df_timesTraining['TrainingTimesLastYear'].mean().round(2)

# Second, let's calculate the average times commited in training for each departmer
sales_training_average = df_timesTraining[df_timesTraining['Department'] == 'Sales'
rrhh_training_average = df_timesTraining[df_timesTraining['Department'] == 'Human F
rd_training_average= df_timesTraining[df_timesTraining['Department'] == 'Research &

print(
    "The average training time dedicated in Sales is: ", sales_training_average.val
    "The average training time dedicated in Human Resources is: ", rrhh_training_av
    "The average training time dedicated in Research & Development is: ", rd_traini
    'The average times for the "TrainingTimeLastYear" column is: ', timesTraining_a
```

```
The average training time dedicated in Sales is:  [[2.85]]
 The average training time dedicated in Human Resources is:  [[2.56]]
 The average training time dedicated in Research & Development is:  [[2.79]]
 The average times for the "TrainingTimeLastYear" column is:  2.8
```

In [ ]:
```python
# Let's create the Bar chart
plt.figure(figsize = (8, 6))

# Adding the variables
bar_training_data = [sales_training_average['TrainingTimesLastYear'].values[0],
            rrhh_training_average['TrainingTimesLastYear'].values[0],
            rd_training_average['TrainingTimesLastYear'].values[0]
        ]
department_cat = ['Sales', 'Human Resources', 'Research & Development']

# Adding the labels
for i, value in enumerate(bar_training_data):
    plt.text(i, value, str(value), ha='center', va='bottom', color='red')

# Add the average line
plt.axhline(timesTraining_average, color='blue', linestyle='--', label='Average Tra

# Add the value of the average to the line
plt.text(len(department_cat), timesTraining_average, f'Average: {timesTraining_aver

# Adjust y-axis limits to ensure the line is visible
plt.ylim(0, max(bar_training_data) + 2)

# Joining the data to create the chart
plt.bar(department_cat, bar_training_data)
plt.xlabel('Department')
plt.ylabel('Average Training Time')
plt.title('Average Training Times Dedicated by Department', fontsize=18, fontweight
plt.show()
```



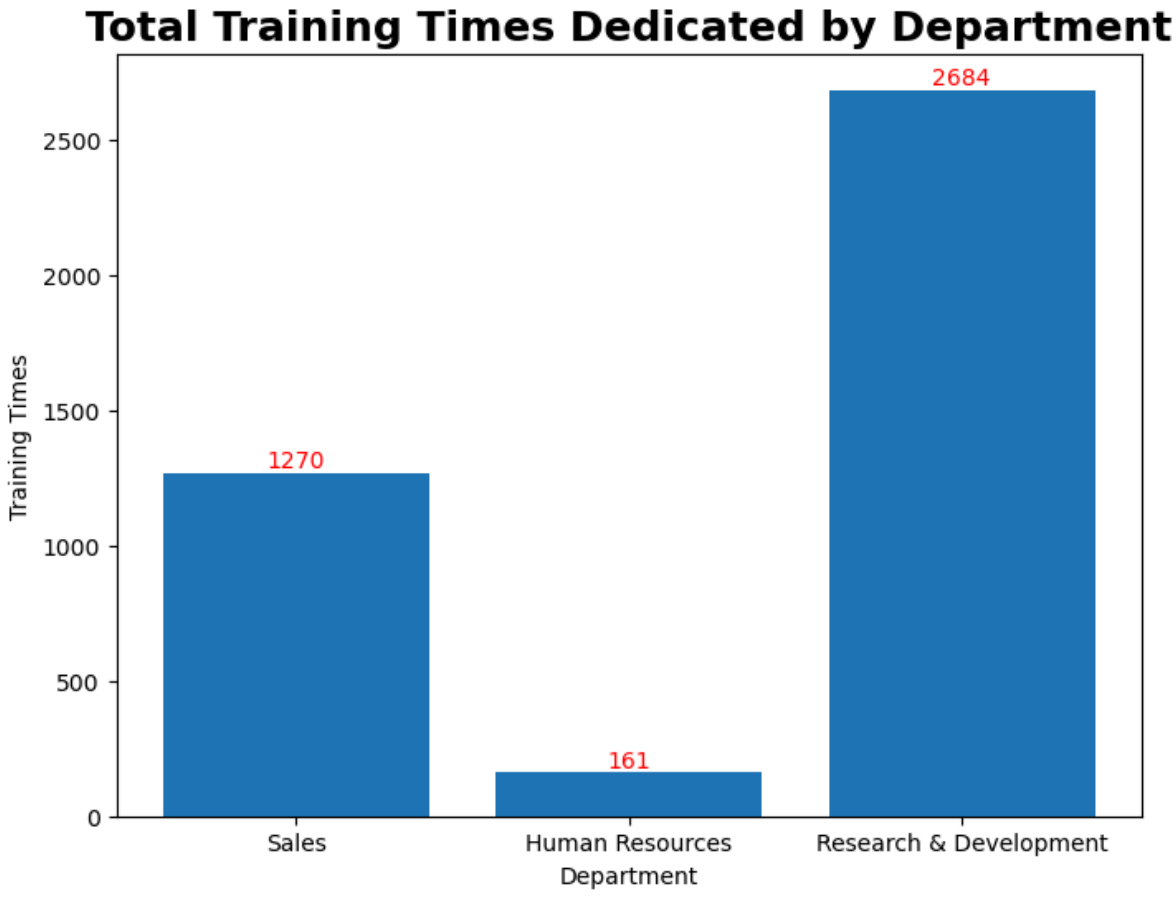
In [ ]:
```python
# Second, let's calculate the average years commited in training for each departmen
sales_training_sum = df_timesTraining[df_timesTraining['Department'] == 'Sales'].gr
```

```
rrhh_training_sum = df_timesTraining[df_timesTraining['Department'] == 'Human Resou
rd_training_sum= df_timesTraining[df_timesTraining['Department'] == 'Research & Dev

print(
    "The total training time dedicated in Sales is: ", sales_training_sum.values, '
    "The total training time dedicated in Human Resources is: ", rrhh_training_sum.
    "The total training time dedicated in Research & Development is: ", rd_training
    'The total years for the "TrainingTimeLastYear" column is: ', timesTraining_ave
```

```
The total training time dedicated in Sales is:  [[1270]]
 The total training time dedicated in Human Resources is:  [[161]]
 The total training time dedicated in Research & Development is:  [[2684]]
 The total years for the "TrainingTimeLastYear" column is:  2.8
```

In [ ]:

```
# Let's create the Bar chart
plt.figure(figsize = (8, 6))

# Adding the variables
bar_training_data = [sales_training_sum['TrainingTimesLastYear'].values[0],
            rrhh_training_sum['TrainingTimesLastYear'].values[0],
            rd_training_sum['TrainingTimesLastYear'].values[0]
        ]
department_cat = ['Sales', 'Human Resources', 'Research & Development']

# Adding the labels
for i, value in enumerate(bar_training_data):
    plt.text(i, value, str(value), ha='center', va='bottom', color='red')

# Joining the data to create the chart
plt.bar(department_cat, bar_training_data)
plt.xlabel('Department')
plt.ylabel('Training Times')
plt.title('Total Training Times Dedicated by Department', fontsize=18, fontweight='
plt.show()
```



4.5 WORK-LIFE BALANCE AND OVERTIME ANALYSIS

These days, evaluating the ***workers' work-life balance*** is among the most crucial evaluations. Let's investigate if they are working too much and whether they are able to make up for their job hours with their free time. Recall that most *workers value a healthy work-life balance*, particularly in light of the growing popularity of remote employment.

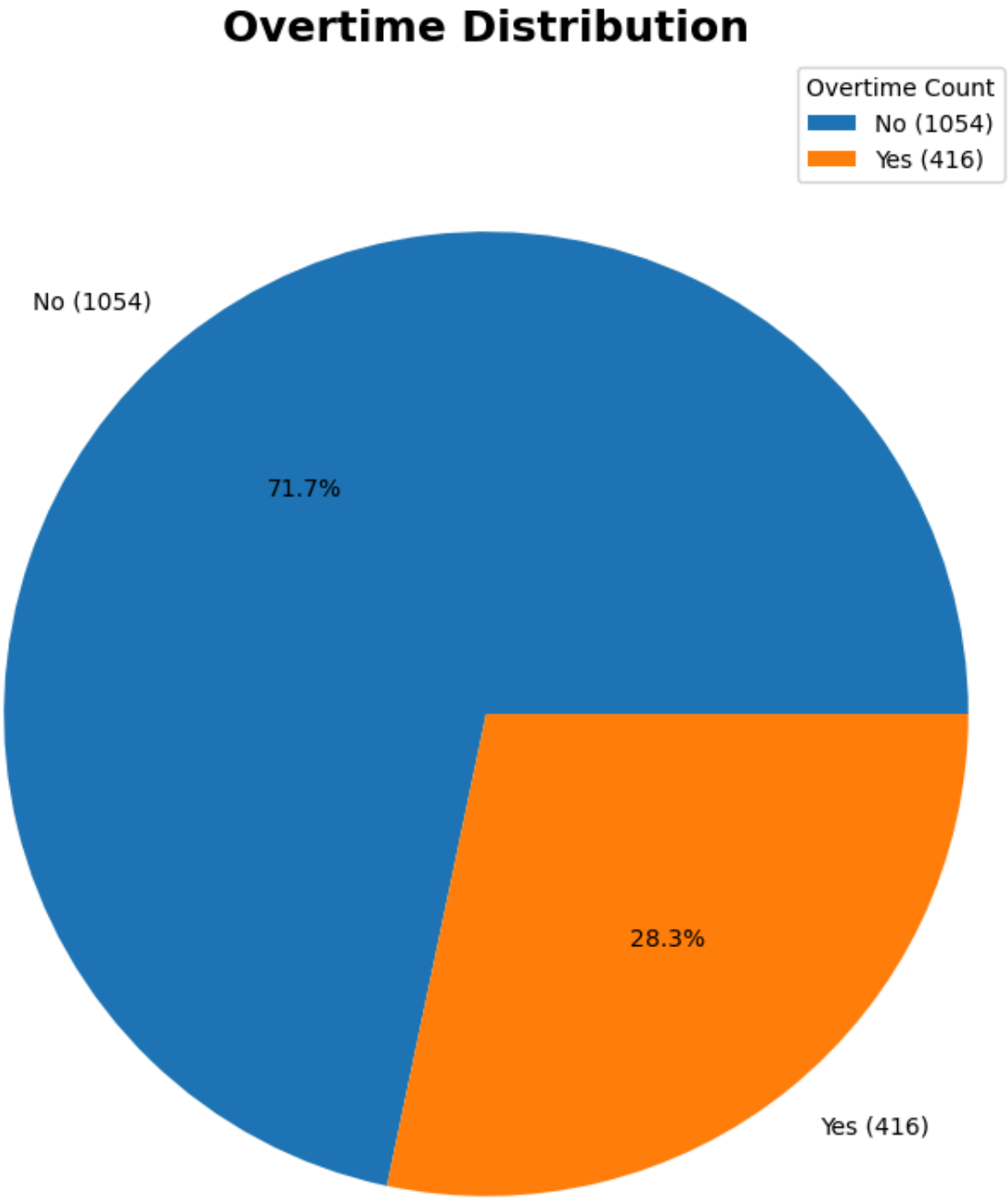
In [ ]:
```
df_wlb = df_padb[['OverTime', 'JobSatisfaction', 'WorkLifeBalance', 'DistanceFromHo
df_wlb.head()
```

Out[ ]:

	OverTime	JobSatisfaction	WorkLifeBalance	DistanceFromHome
0	Yes	Very High	Bad	1
1	No	Medium	Better	8
2	Yes	High	Better	2
3	Yes	High	Better	3
4	No	Medium	Better	2

In [ ]:
```
# Let's analyze the data from the OverTime column
overtime_count = df_wlb['OverTime'].value_counts()
print(overtime_count)

# Total and percentage variables to use on our charts
total_overtime_count = overtime_count.sum()
overtime_percentage = (overtime_count / total_count) * 100

# labels = gender_counts.index
# Using a function to create the labels
labels = [f'{overtime} ({count})' for overtime, count in zip(overtime_count.index,

# Chart size
fig, ax = plt.subplots(figsize=(8, 10))

# Chart generation
plt.pie(overtime_count, labels=labels, autopct='%1.1f%%')
plt.legend(title='Overtime Count')
ax.set_title('Overtime Distribution', fontsize=18, fontweight='bold')
plt.axis('equal')
plt.show()
```

```
No     1054
Yes     416
Name: OverTime, dtype: int64
```

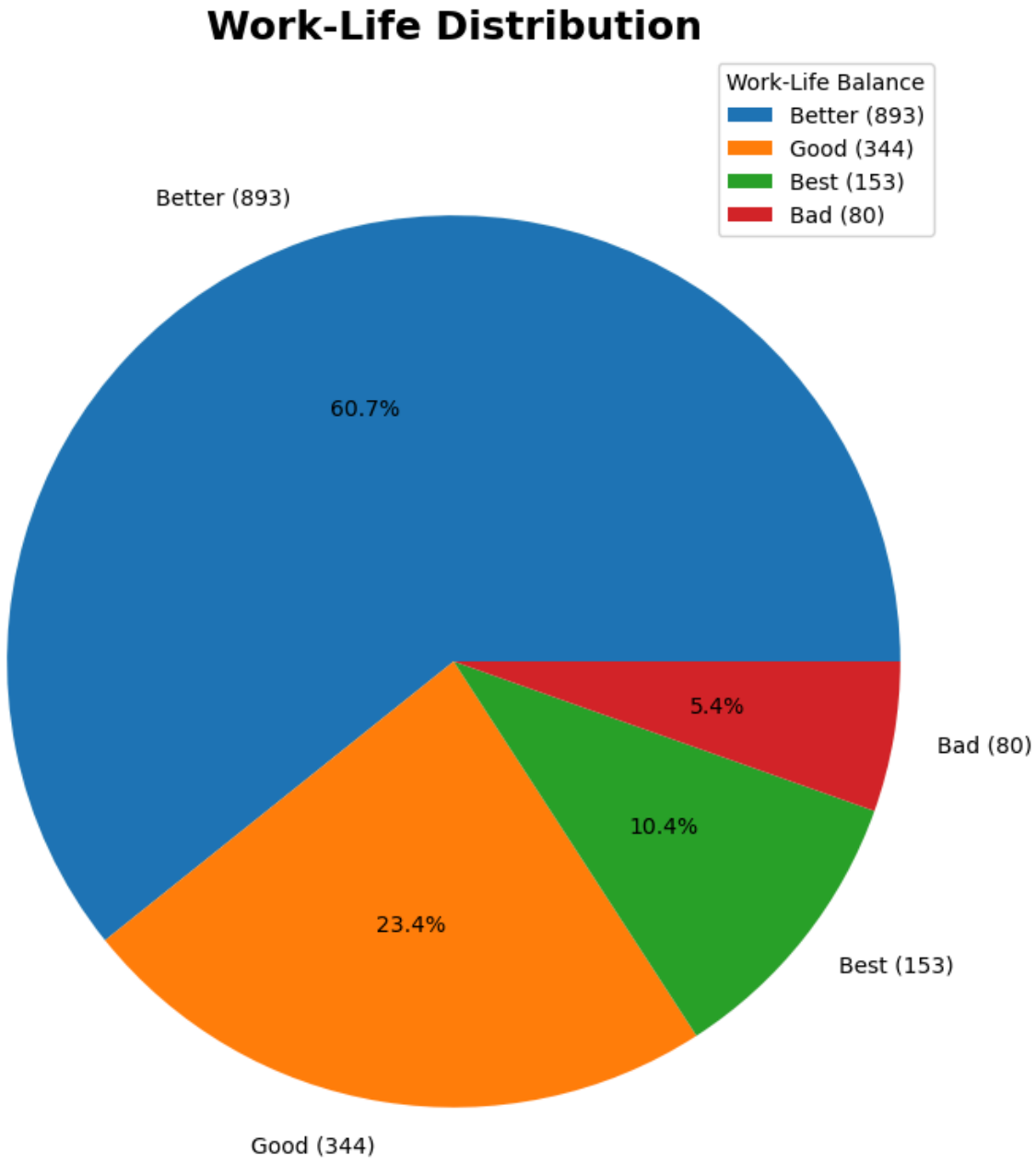


A review of the '*OverTime*' column data reveals that just **28.3%** of our staff members work overtime. It is not possible to ascertain how an employee's work-life balance is impacted by overtime. In order to obtain a useful result, I thus choose to look at the numbers in the "*WorkLifeBalance*" column.

In [ ]:
```
# Let's analyze the data from the OverTime column
wlbalance_count = df_wlb['WorkLifeBalance'].value_counts()
print(wlbalance_count)

# Total and percentage variables to use on our charts
total_wlbalance_count = wlbalance_count.sum()
wlbalance_percentage = (wlbalance_count / total_count) * 100
```

```
# labels = gender_counts.index
# Using a function to create the labels
labels = [f'{wlbalance} ({count})' for wlbalance, count in zip(wlbalance_count.inde

# Chart size
fig, ax = plt.subplots(figsize=(8, 10))

# Chart generation
plt.pie(wlbalance_count, labels=labels, autopct='%1.1f%%')
plt.legend(title='Work-Life Balance')
ax.set_title('Work-Life Distribution', fontsize=18, fontweight='bold')
plt.axis('equal')
plt.show()
```

```
Better    893
Good      344
Best      153
Bad        80
Name: WorkLifeBalance, dtype: int64
```



More pertinent information about our employees' appreciation of work-life balance was found in the 'WorkLifeBalance' column. According to my research, **5.4%** of workers think that there is not a good work-life balance. I'm now attempting to determine whether there is another factor influencing them. Let's narrow down the data to just see those whose *"Job Satisfaction"* and *"WorkLifeBalance"* are both **"Low"**. Also I will explore if they work *'OverTime'*

In [ ]:
```
# Filtering the data
wlb_JsWlb_low = df_wlb[(df_wlb['WorkLifeBalance'] == 'Bad') & (df_wlb['JobSatisfact
wlb_JsWlb_good = df_wlb[(df_wlb['WorkLifeBalance'] == 'Good') & (df_wlb['JobSatisfa

# Counting the filtered data.
countWlbJs_low = wlb_JsWlb_low.shape[0]
countWlbJs_good = wlb_JsWlb_good.shape[0]
print("The total number of employees with Low Job Satisfaction, Bad Work-Life Balar
print("The total number of employees with Good Job Satisfaction, Bad Work-Life Bala
```

```
The total number of employees with Low Job Satisfaction, Bad Work-Life Balance and
Over time Yes, is:  2
The total number of employees with Good Job Satisfaction, Bad Work-Life Balance an
d Over time Yes is:  17
```

In [ ]:
```
df_wlb['DistanceFromHome'].describe()
```

Out[ ]:
```
count    1470.000000
mean        9.192517
std         8.106864
min         1.000000
25%         2.000000
50%         7.000000
75%        14.000000
max        29.000000
Name: DistanceFromHome, dtype: float64
```

I discovered after analyzing the data that employees are satisfied with the work-life balance the organization provides. *Emphasizing and attempting to implement a policy to lower the **28.3%** of workers over time is crucial*. Additionally, I discovered that very few workers truly struggle with work-life balance, thus attending this employee disconfirmation is advised.

4.6 ATTRITION ANALYSIS

Now it's time to attend to one of the most important metrics for our company: ***the attrition rate of the company during 2021***. Knowing *the rate of turnover* in our organization will help us *know if we are losing underperforming employees or the good ones*. Maybe a high rate of turnover means that we are losing low-performing employees; this metric will help us retain good talent.

In [ ]:
```
# Creating a data frame to make it easier to work with the data
df_attrition = df_padb[['Attrition', 'EnvironmentSatisfaction', 'JobInvolvement', '
df_attrition.head()
```

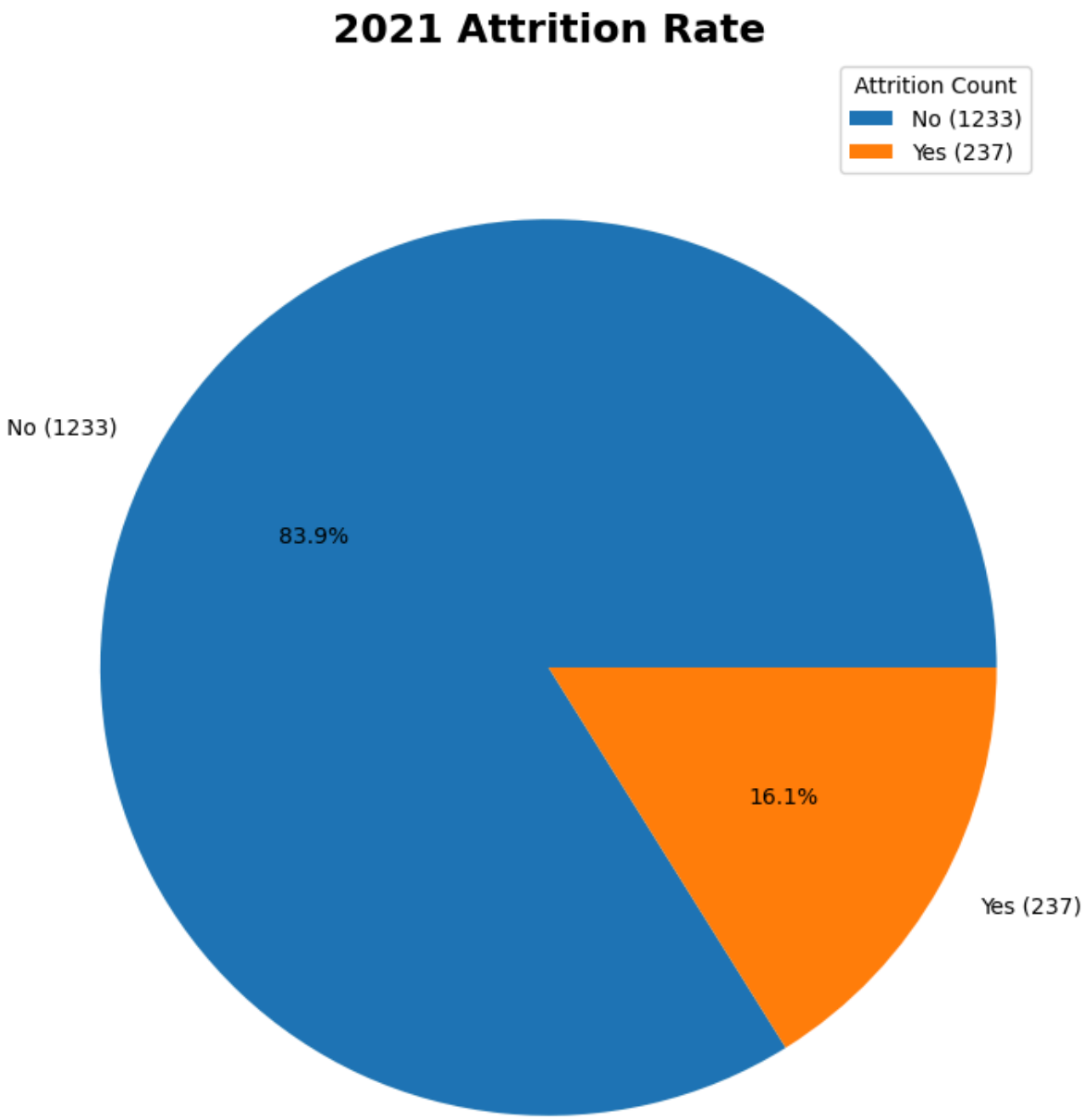
Out[ ]:

	Attrition	EnvironmentSatisfaction	JobInvolvement	JobSatisfaction	MonthlyIncome	OverTime
0	Yes	Medium	High	Very High	5993	Yes
1	No	High	Medium	Medium	5130	No
2	Yes	Very High	Medium	High	2090	Yes
3	No	Very High	High	High	2909	Yes
4	No	Low	High	Medium	3468	No

With the new data frame, now we can calculate the attrition rate of our company

In [ ]:
```
# Let's analyze the data from the OverTime column
attrition_count = df_attrition['Attrition'].value_counts()
print('The attrition count is: ', '\n', attrition_count)

# Total and percentage variables to use on our charts
total_attrition_count = attrition_count.sum()
attrition_percentage = (attrition_count / total_count) * 100
print('The attrition percentage is: ', '\n', round(attrition_percentage, 1))

# labels = gender_counts.index
# Using a function to create the labels
labels = [f'{attrition} ({count})' for attrition, count in zip(attrition_count.inde

# Chart size
fig, ax = plt.subplots(figsize=(8, 10))

# Chart generation
plt.pie(attrition_count, labels=labels, autopct='%1.1f%%')
plt.legend(title='Attrition Count')
ax.set_title('2021 Attrition Rate', fontsize=18, fontweight='bold')
plt.axis('equal')
plt.show()
```

```
The attrition count is:
 No     1233
Yes     237
Name: Attrition, dtype: int64
The attrition percentage is:
 No     83.9
Yes    16.1
Name: Attrition, dtype: float64
```



With a **16.1% attrition rate in 2021**, *237 employees departed* the organization in total. Let's attempt to ascertain whether they were driven by any particular reason or whether a change was all they desired.

In [ ]:
```
# Filtering the data
attrition_worstvalues = df_attrition[(df_attrition['EnvironmentSatisfaction'] == 'L
attrition_mediumvalues = df_attrition[(df_attrition['EnvironmentSatisfaction'] == '

# Counting the filtered data.
count_attrition_wv = attrition_worstvalues.shape[0]
count_attrition_mv = attrition_mediumvalues.shape[0]
print("The number of employees who leave the company with Job Satisfaction and Envi
print("The number of employees who leave the company, with Job Satisfaction 'Low' a
```

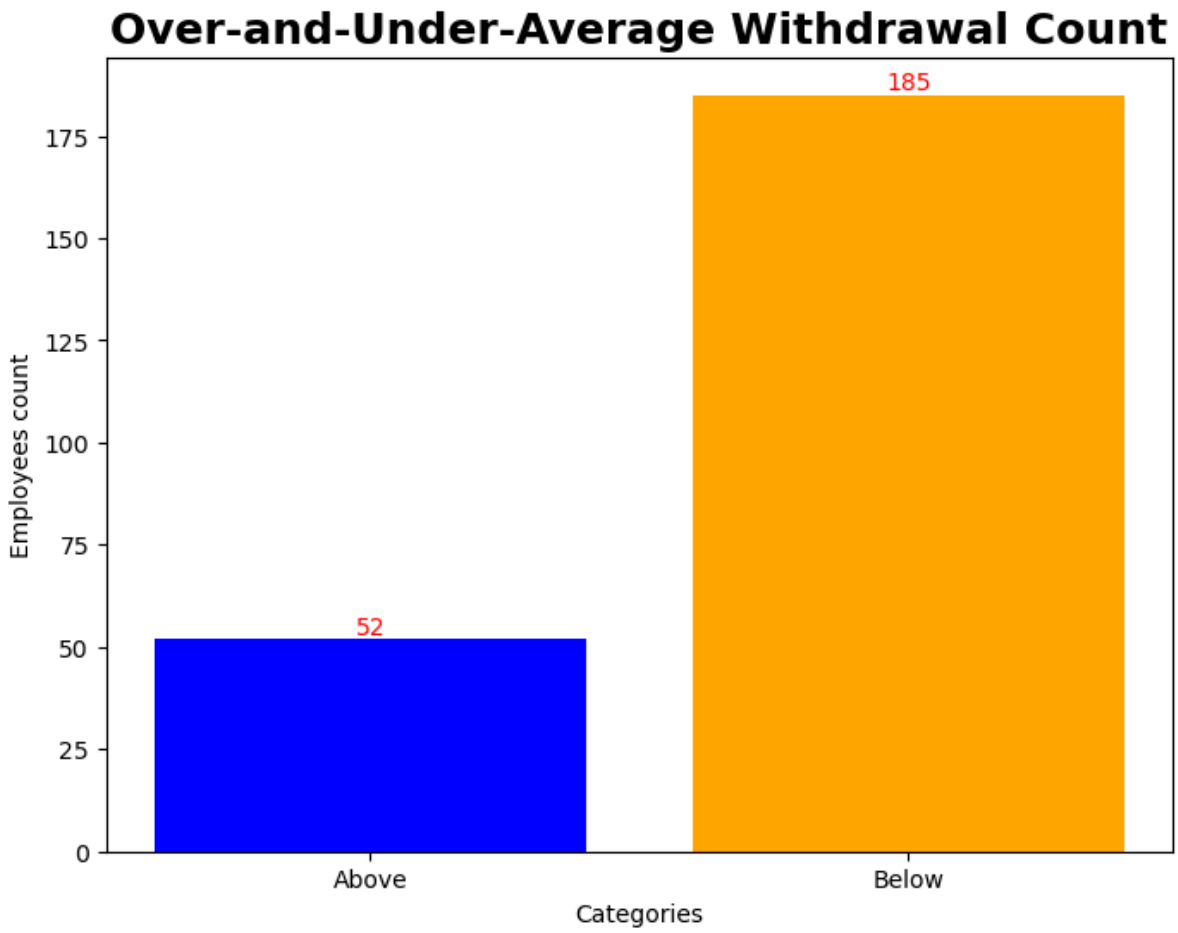
```
The number of employees who leave the company with Job Satisfaction and Environmen
t Satisfaction 'Low', and working Over time is:  15
The number of employees who leave the company, with Job Satisfaction 'Low' and Env
ironment Satisfaction 'Medium', and working Over time is:  16
```

Let's find out if the employee's choice to quit the firm was influenced by their ***monthly income***.

In [ ]:
```
# Calculating the monthly income mean
attrition_mincome_mean = df_attrition['MonthlyIncome'].mean().round(2)
print(attrition_mincome_mean)

# Filtering the attrition data by those who are under or over the mean monthly incc
attrition_over_mean_mi = df_attrition[(df_attrition['MonthlyIncome'] > attrition_mi
attrition_under_mean_mi = df_attrition[(df_attrition['MonthlyIncome'] < attrition_m

count_attrition_over_mean = attrition_over_mean_mi.shape[0]
count_attrition_under_mean = attrition_under_mean_mi.shape[0]

print('The number of employees who left and were over the mean is: ', count_attriti
print('The number of employees who left and were under the mean is: ', count_attrit
```

```
6502.93
The number of employees who left and were over the mean is:  52
The number of employees who left and were under the mean is:  185
```

In [ ]:
```
# Let's create the Bar chart
plt.figure(figsize = (8, 6))

# Adding the variables
attrition_data = [count_attrition_over_mean, count_attrition_under_mean]
attrition_cat = ['Above', 'Below']

# Adding the labels
for i, value in enumerate(attrition_data):
    plt.text(i, value, str(value), ha='center', va='bottom', color='red')

# Joining the data to create the chart
plt.bar(attrition_cat, attrition_data, color=['blue', 'orange'])
plt.xlabel('Categories')
plt.ylabel('Employees count')
plt.title('Over-and-Under-Average Withdrawal Count', fontsize=18, fontweight='bold'
plt.show()
```



After the analysis the data shows us that 52 of the 237 employees who left the firm had monthly incomes that were above average, while 185 of the employees who left had their incomes below average.

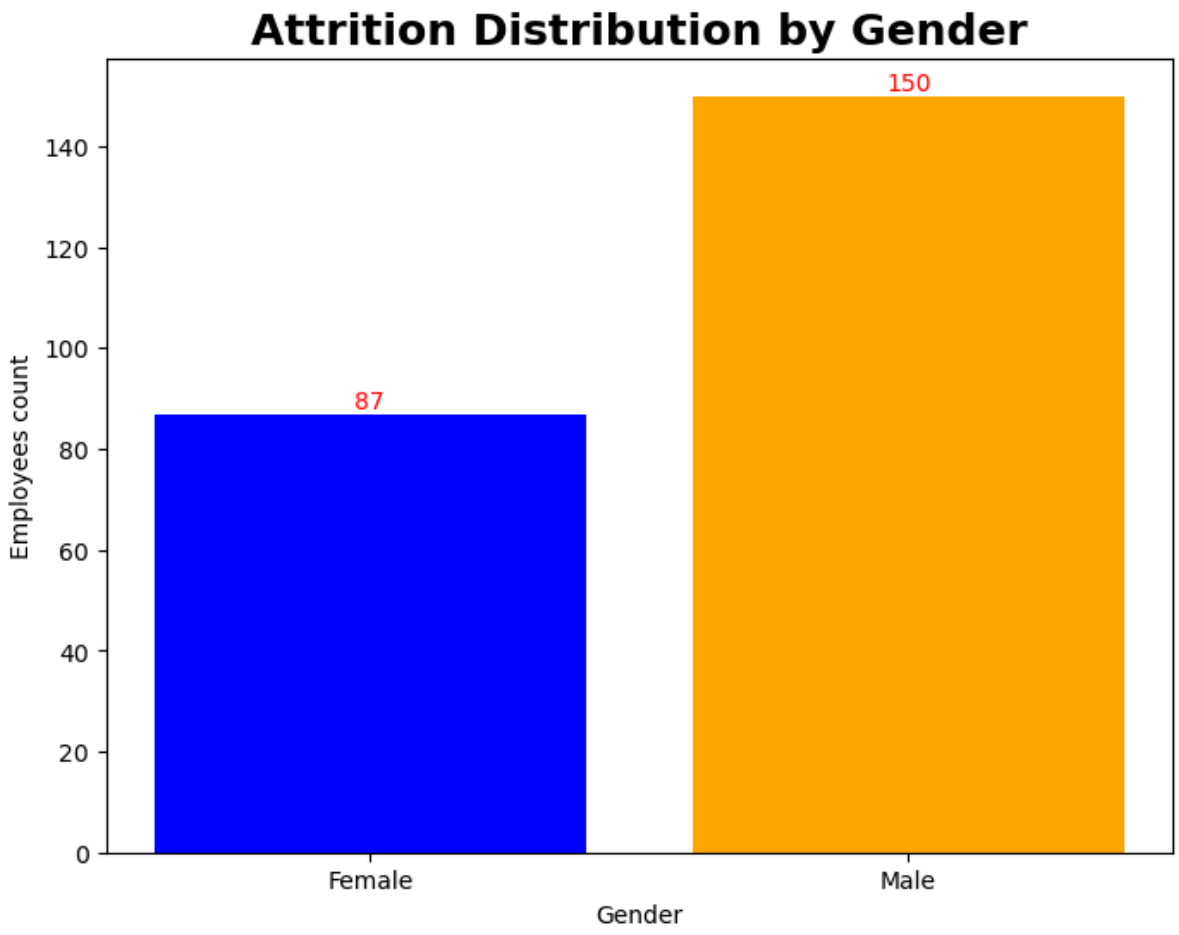
Let's examine the distribution of attrition by ***Gender*** and ***Age Range***.

In [ ]:
```
# Let's start examining by the gender
attrition_female = df_attrition[(df_attrition['Gender'] == 'Female') & (df_attritic
attrition_male = df_attrition[(df_attrition['Gender'] == 'Male') & (df_attrition['A

# Counting the distribution of attrition by gender
count_attrition_female = attrition_female.shape[0]
count_attrition_male = attrition_male.shape[0]

print('Number of females that had resigned from the company: ', '\n', count_attriti
print('Number of males that had resigned from the company: ', '\n', count_attritior
```

```
Number of females that had resigned from the company:
 87
Number of males that had resigned from the company:
 150
```

In [ ]:
```
# Let's create the Bar chart
plt.figure(figsize = (8, 6))

# Adding the variables
gender_attrition_data = [count_attrition_female, count_attrition_male]
attrition_gender_cat = ['Female', 'Male']

# Adding the labels
for i, value in enumerate(gender_attrition_data):
```

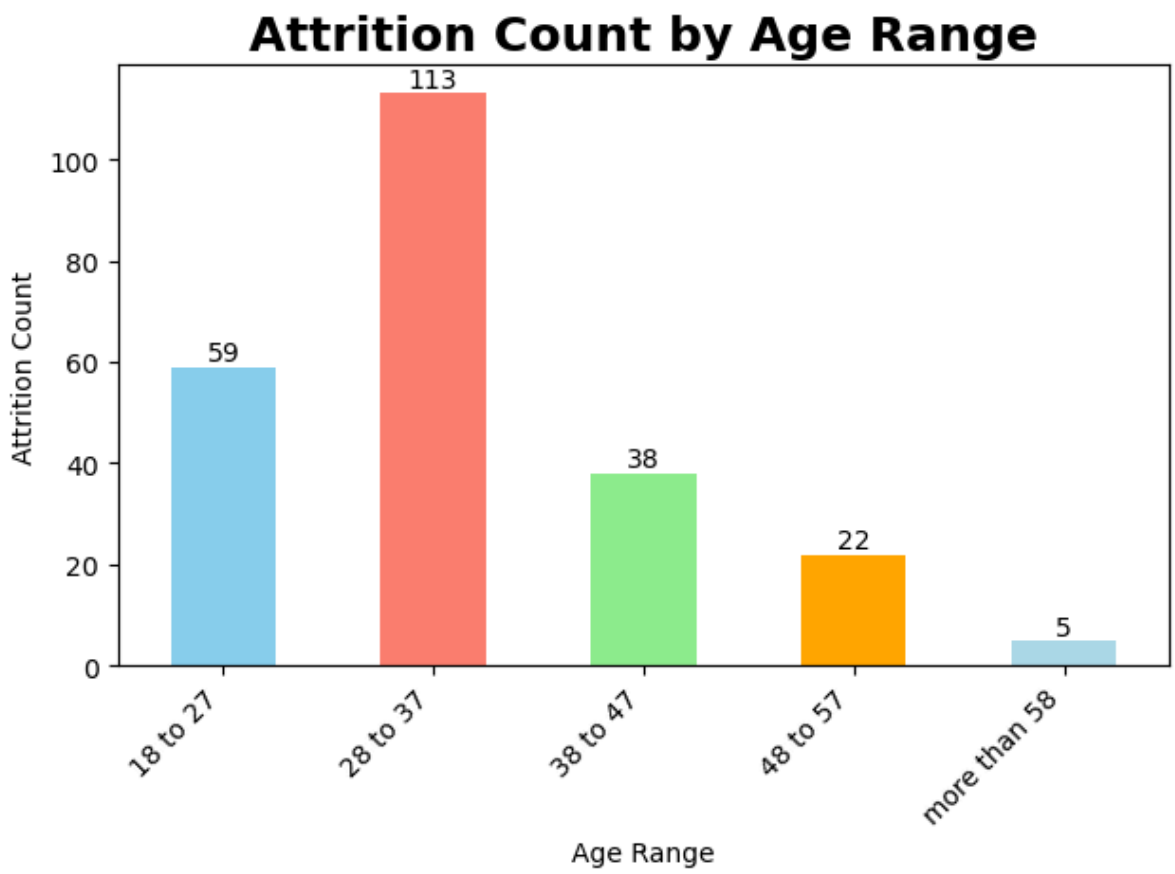
```
    plt.text(i, value, str(value), ha='center', va='bottom', color='red')

# Joining the data to create the chart
plt.bar(attrition_gender_cat, gender_attrition_data, color=['blue', 'orange'])
plt.xlabel('Gender')
plt.ylabel('Employees count')
plt.title('Attrition Distribution by Gender', fontsize=18, fontweight='bold')
plt.show()
```



In [ ]:

```
# Filtering the Attrition by the value 'Yes'
age_range_attrition_filter = df_attrition[(df_attrition['Attrition'] == 'Yes')]

colors = ['skyblue', 'salmon', 'lightgreen', 'orange', 'lightblue']

# Group the values by the 'AgeRange'
attrition_by_age = age_range_attrition_filter.groupby('AgeRange').size()

# Create bar chart
ax = attrition_by_age.plot(kind='bar', color=colors)

# Explicitly specify x-axis tick locations and labels
ax.set_xticks(range(len(attrition_by_age)))
ax.set_xticklabels(attrition_by_age.index, rotation=45, ha='right')

# Add values to the bars
for i, v in enumerate(attrition_by_age):
    ax.text(i, v + 0.1, str(v), ha='center', va='bottom')

# Add labels and title
plt.xlabel('Age Range')
plt.ylabel('Attrition Count')
plt.title('Attrition Count by Age Range', fontsize=18, fontweight='bold')

# Show plot
```

```
plt.tight_layout()
plt.show()
```

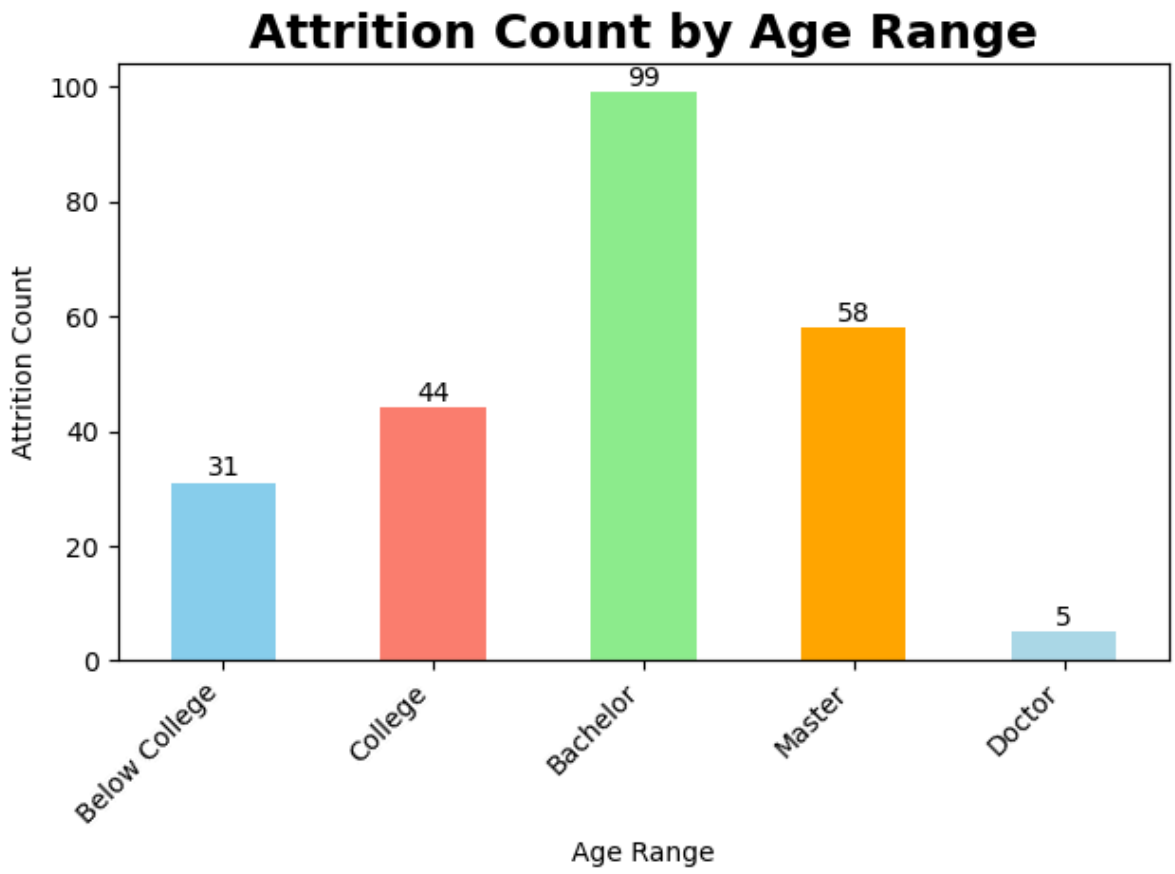


When I break down the resignations by *age range*, I find that the age range of *18 to 47 years old* is where the firm is losing the most employees, with **210 departing**. The age range of 28 to 37 years old is where the company *records the highest number of resignations*.

From what I see with the age range, let's investigate if, from those resignations, the company is ***losing good talent***. I will analyze attrition based on the employee's education level.

In [ ]:
```
# Filtering the Attrition by the value 'Yes'
education_attrition_filter = df_attrition[(df_attrition['Attrition'] == 'Yes')]

colors = ['skyblue', 'salmon', 'lightgreen', 'orange', 'lightblue']

# Group the values by the 'AgeRange'
attrition_by_education = education_attrition_filter.groupby('Education').size()

# Define the desired order of education levels
desired_order = ['Below College', 'College', 'Bachelor', 'Master', 'Doctor']

# Reindex the Series according to the desired order
attrition_by_education = attrition_by_education.reindex(desired_order)

# Create bar chart
ax = attrition_by_education.plot(kind='bar', color=colors)

# Explicitly specify x-axis tick locations and labels
ax.set_xticks(range(len(attrition_by_age)))
ax.set_xticklabels(attrition_by_education.index, rotation=45, ha='right')

# Add values to the bars
for i, v in enumerate(attrition_by_education):
    ax.text(i, v + 0.1, str(v), ha='center', va='bottom')
```

```
# Add labels and title
plt.xlabel('Age Range')
plt.ylabel('Attrition Count')
plt.title('Attrition Count by Age Range', fontsize=18, fontweight='bold')

# Show plot
plt.tight_layout()
plt.show()
```



When I break down the resignations by educational level, I find that employees with degrees of "*Bachelor*" and "*Master*", followed by "*College*", are the most likely to quit, registering a *total of **201** resignations*. This information raises the possibility that *the organization is losing talent as a result of the resignations*.

4.7 MACHINE LEARNING MODELS

Let's attempt to create some *Machine Learning Models* using the company's data.

Remember that for this kind of analysis, we need to use ***Numpy***.

4.7.a Correlation between Salary and Experience - Linear Regression

I'll use the data from the columns "*MonthlyIncome*" and "*TotalWorkingYears*" for this study. The **Linear Regression Model** will be my tool. *I would want to know if the pay is commensurate with years of experience*.

I will use two types of calculations, one using Numpy and the other from StatsModels, which include information such as coefficients, standard errors, t-values, p-values, and R-squared,

which are crucial for interpreting the results of the linear regression model. To visualize the data, I will use a scatter plot

In [ ]:
```
# Creating our data frame
df_salary = df_padb[['MonthlyIncome', 'TotalWorkingYears']].copy()
df_salary.describe()
```

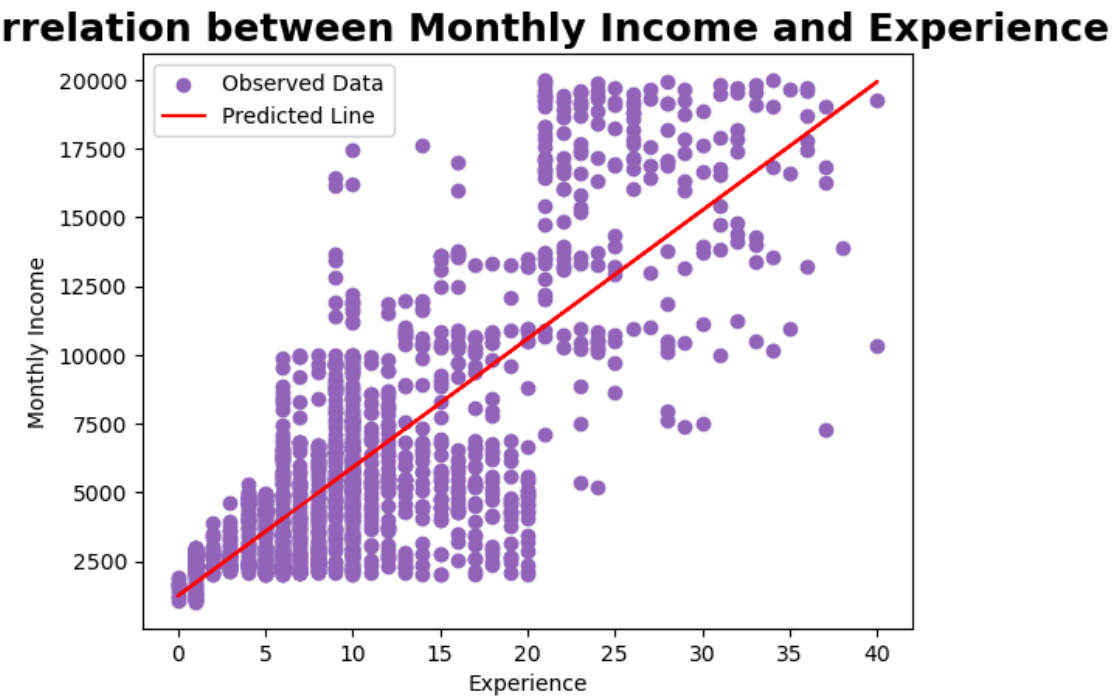
Out[ ]:

	MonthlyIncome	TotalWorkingYears
count	1470.000000	1470.000000
mean	6502.931293	11.279592
std	4707.956783	7.780782
min	1009.000000	0.000000
25%	2911.000000	6.000000
50%	4919.000000	10.000000
75%	8379.000000	15.000000
max	19999.000000	40.000000

4.7.a.1 Linear Regression - Numpy

In [ ]:
```
# Defining the data for the x-axis and y-axis
x = df_salary['TotalWorkingYears'];
y = df_salary['MonthlyIncome'];

# Creating the plot
plt.scatter(x = x, y = y, color='#9467bd')

#obtain m (slope) and b(intercept) of linear regression line
m, b = np.polyfit(x, y, 1)
lreg = np.corrcoef(x, y)
# Ploting the linear regression  line
plt.plot(x, m*x+b, color='red')

# Adding labels and title
plt.xlabel('Experience')
plt.ylabel('Monthly Income')
plt.title('Correlation between Monthly Income and Experience', fontsize=18, fontwei
plt.legend(['Observed Data', 'Predicted Line'])

# Printing the linear regression value
print(lreg)
```

```
[[1.         0.77289325]
 [0.77289325 1.        ]]
```



Our *correlation coefficient* is approximately **0.77** indicates a relatively strong positive linear relationship between *TotalWorkingYears* and *MonthlyIncome*.

Some salaries need to be fixed to be closer to the regression line because the company experiences a better correlation between salary and experience. The company will have attractive salary options.

Note that we are not accounting for the company's provision of any further financial incentives in this research.

I am going to import the statmodels library to work with the other Linear Regression formula.

4.7.a.2 Linear Regression - Statsmodels

I am going to import the statmodels library to work with the other Linear Regression formula.

In [ ]:
```
import statsmodels.formula.api as smf
model = smf.ols('MonthlyIncome ~ TotalWorkingYears', data = df_salary).fit()
model.summary()
```

Out[ ]:

OLS Regression Results

Dep. Variable:	MonthlyIncome	**R-squared:**	0.597
Model:	OLS	**Adj. R-squared:**	0.597
Method:	Least Squares	**F-statistic:**	2178.
Date:	Sat, 30 Mar 2024	**Prob (F-statistic):**	2.73e-292
Time:	00:02:04	**Log-Likelihood:**	-13848.
No. Observations:	1470	**AIC:**	2.770e+04
Df Residuals:	1468	**BIC:**	2.771e+04
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	1227.9353	137.299	8.944	0.000	958.612	1497.259
TotalWorkingYears	467.6584	10.021	46.669	0.000	448.002	487.315

Omnibus:	47.473	**Durbin-Watson:**	1.993
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	79.304
Skew:	0.269	**Prob(JB):**	6.02e-18
Kurtosis:	4.003	**Cond. No.**	24.2

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Now I have the model that is performing a *linear regression analysis* to explore the relationship between the *TotalWorkingYears predictor variable and the MonthlyIncome*.

Let's use the model to create some predictions.

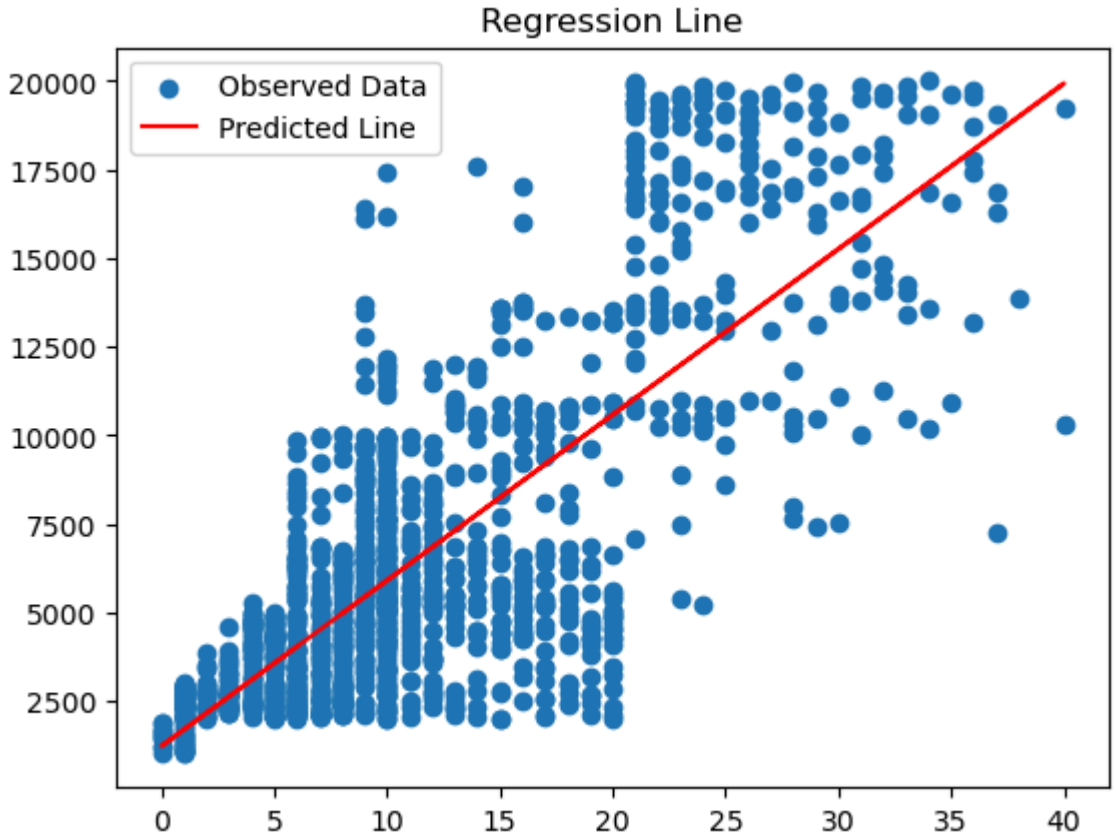
In [ ]:
```
# Using our model to create predictions
# Let's predict our salaries for each experience years and store them into a variab
pred1 = model.predict(pd.DataFrame(df_salary['TotalWorkingYears']))
print(pred1)
```

```
0       4969.202583
1       5904.519406
2       4501.544171
3       4969.202583
4       4033.885759
           ...
1465    9178.128289
1466    5436.860994
1467    4033.885759
1468    9178.128289
1469    4033.885759
Length: 1470, dtype: float64
```

With the fitted linear regression model (model), we predict salaries based on years of experience. The values are now stored in a variable called ***pred1***.

Let's examine the model's appearance in a scatter plot.

In [ ]:
```
# Regression Line
# X and Y were defined at the beginning, first the scatter then the line with the p
plt.scatter(x, y)
plt.plot(x, pred1, 'r')
# Let's add a legend to our plot
plt.legend(['Observed Data', 'Predicted Line'])
plt.title('Regression Line')
plt.show()
```



The outcome of the prediction model is identical to the one we produced with Numpy.

Let's examine *the error calculation*.

In [ ]:
```
# Error Calculation
res1 = y - pred1
res_sqr1 = res1 * res1
mse1 = np.mean(res_sqr1)
rmse1 = np.sqrt(mse1)
print(rmse1)
```

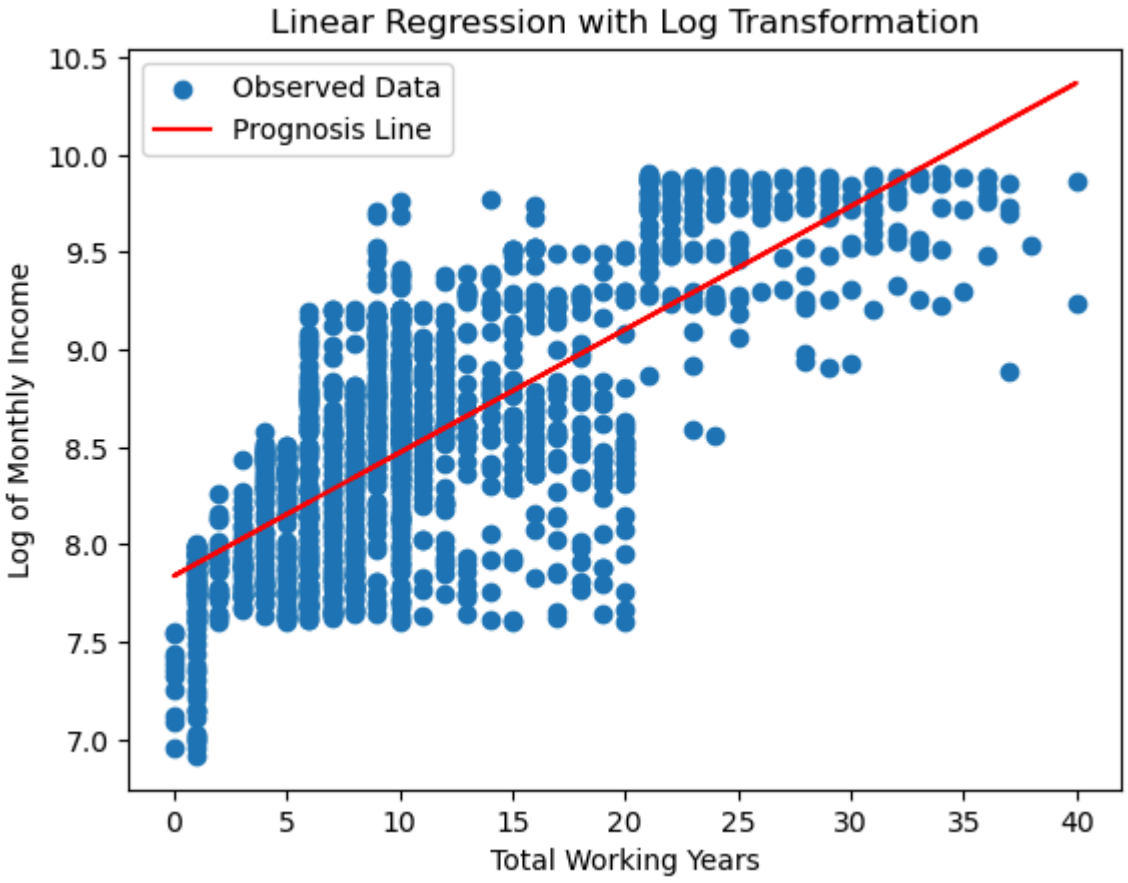
```
2986.3521316844103
```

As a gauge of the linear regression model's prediction ability, I computed the root mean squared error (RMSE). The model computes the residuals, squares them to get the mean squared error (MSE), and then takes the square root to get the RMSE. A lower RMSE number indicates greater performance, and it gives information about how well the model matches the observed data. With a value of 2986.35, this is the average magnitude of the errors between predicted values and actual values. The model is performing well in terms of prediction.

4.7.a.3 Linear Regression - Log Transformation

Let's convert our data using logarithms.

In [ ]:
```
# Transformed data
# Log Transformation
plt.scatter(x = x, y = np.log(y))
np.corrcoef(x, np.log(y))
model2 = smf.ols('np.log(MonthlyIncome) ~ TotalWorkingYears', data=df_salary).fit()
# Adding the linear regression line
plt.plot(x, model2.predict(df_salary), color='red')

# Labels and title
plt.xlabel('Total Working Years')
plt.ylabel('Log of Monthly Income')
plt.title('Linear Regression with Log Transformation')
plt.legend(['Observed Data', 'Prognosis Line'])

# Show plot
plt.show()
```



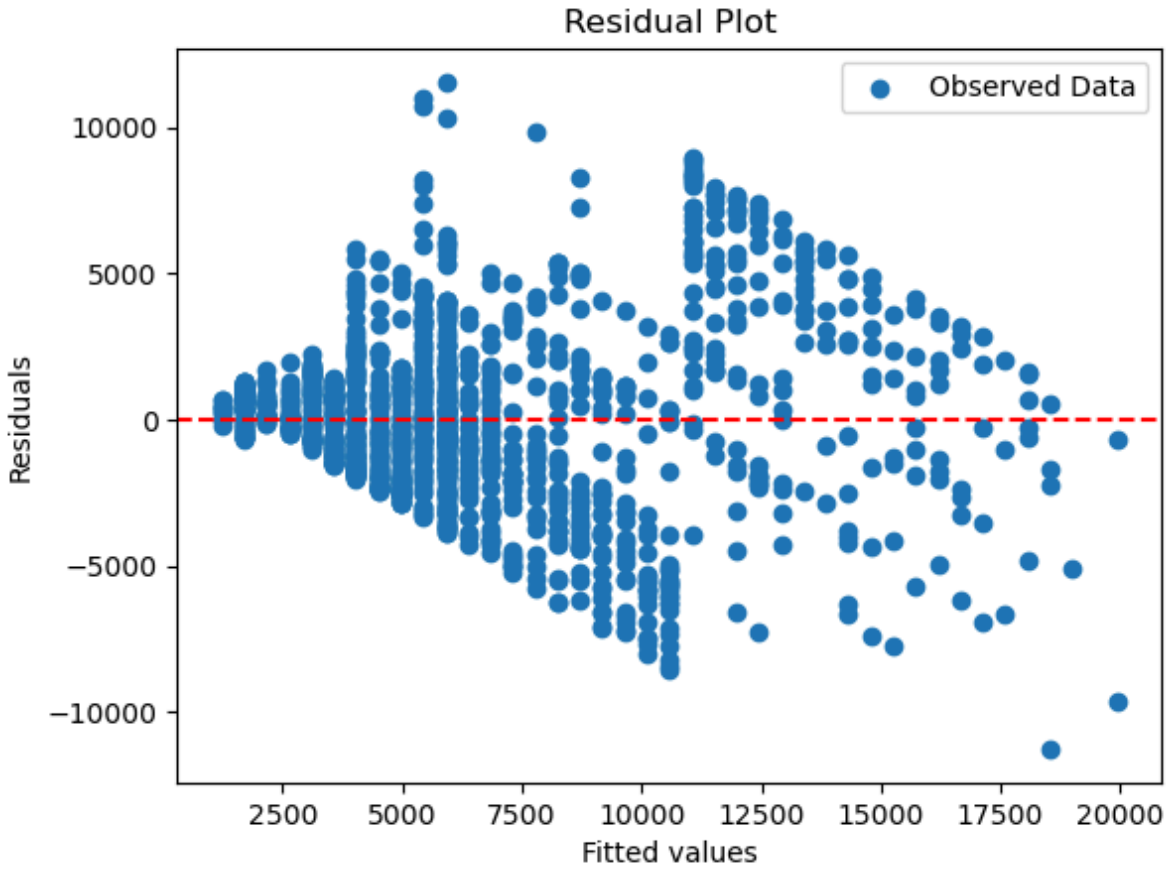
After giving Salary a log transformation, we can see how Experience and Salary relate to one another. The correlation coefficient between the two variables is now available, and a linear regression model is provided to further examine the connection between Experience and the salary logarithm.

It's recommended to compare the original data to see if the relationship is better represented on a logarithmic scale. We can interpret the coefficients of the linear regression model to understand the relationship between 'TotalWorkingYears' and the expected value of 'MonthlyIncome' on the log scale.

4.7.a.4 Linear Regression - Comparation

Using a residual analysis, I will contrast the log-transformed data with the original data.

In [ ]:
```
# Calculate residuals
residuals = model.resid

# Visualize residuals
plt.scatter(model.fittedvalues, residuals)
plt.xlabel('Fitted values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.legend(['Observed Data', 'Prognosis Line'])
plt.axhline(y=0, color='red', linestyle='--')  # Add horizontal line at y=0
plt.show()
```



The plot shows us the *residual values* against the *predicted values*.

In [ ]:
```
# Original model summary
print(model.summary())

# Log-transformed model summary (assuming you've already fit model2)
print(model2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:          MonthlyIncome   R-squared:                       0.597
Model:                            OLS   Adj. R-squared:                  0.597
Method:                 Least Squares   F-statistic:                     2178.
Date:                Sat, 30 Mar 2024   Prob (F-statistic):          2.73e-292
Time:                        00:02:05   Log-Likelihood:                -13848.
No. Observations:                1470   AIC:                         2.770e+04
Df Residuals:                    1468   BIC:                         2.771e+04
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept          1227.9353    137.299      8.944      0.000     958.612    1497.259
TotalWorkingYears   467.6584     10.021     46.669      0.000     448.002     487.315
==============================================================================
Omnibus:                       47.473   Durbin-Watson:                   1.993
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               79.304
Skew:                           0.269   Prob(JB):                     6.02e-18
Kurtosis:                       4.003   Cond. No.                         24.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
                            OLS Regression Results
==============================================================================
Dep. Variable:     np.log(MonthlyIncome)   R-squared:                       0.549
Model:                               OLS   Adj. R-squared:                  0.548
Method:                    Least Squares   F-statistic:                     1784.
Date:                   Sat, 30 Mar 2024   Prob (F-statistic):          9.04e-256
Time:                           00:02:05   Log-Likelihood:                -899.93
No. Observations:                   1470   AIC:                             1804.
Df Residuals:                       1468   BIC:                             1814.
Df Model:                              1
Covariance Type:               nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept             7.8391      0.021    382.037      0.000       7.799       7.879
TotalWorkingYears     0.0632      0.001     42.232      0.000       0.060       0.066
==============================================================================
Omnibus:                       10.743   Durbin-Watson:                   1.981
Prob(Omnibus):                  0.005   Jarque-Bera (JB):               10.779
Skew:                          -0.195   Prob(JB):                      0.00456
Kurtosis:                       2.845   Cond. No.                         24.2
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

There is more to study about the models, at the time, it is beyond my knowledge, but ***I am interested in learning more about them***.

4.7.b Attrition Prediction - Survival Model with Kaplan-Meier method

Having ascertained our *company's attrition rate*, let's investigate the possibility of developing a machine learning model that anticipates resignation.

For this analysis I will use the values from the columns *'Attrition', 'YearsAtCompany' and 'AgeRange'*.

In [ ]:
```
# Create a data frame with the data we need
df_km_attrition = df_padb[['YearsAtCompany', 'Attrition', 'AgeRange']].copy()
df_km_attrition.head()
```

Out[ ]:

	YearsAtCompany	Attrition	AgeRange
0	6	Yes	38 to 47
1	10	No	48 to 57
2	0	Yes	28 to 37
3	8	No	28 to 37
4	2	No	18 to 27

To work with our model, I will encode the values from the 'Attrition' and 'AgeRange' columns.

In [ ]:
```
# Attrition column
# Encoding the data from Attrition
attrition_ref = {
    "Yes": 1,
    "No": 0
}
# Encoding the values
df_km_attrition['Attrition'] = df_km_attrition['Attrition'].map(attrition_ref)

# AgeRange Column
# Encoding the data from AgeRange
encoded_age_range = pd.get_dummies(df_km_attrition['AgeRange'], prefix='AgeRange')
# Now I concatenate the new encoded values on a new data frame
df_km_attencoded = pd.concat([df_km_attrition, encoded_age_range], axis=1)
df_km_attencoded.drop('AgeRange', axis='columns' , inplace=True)
df_km_attencoded.head()
```

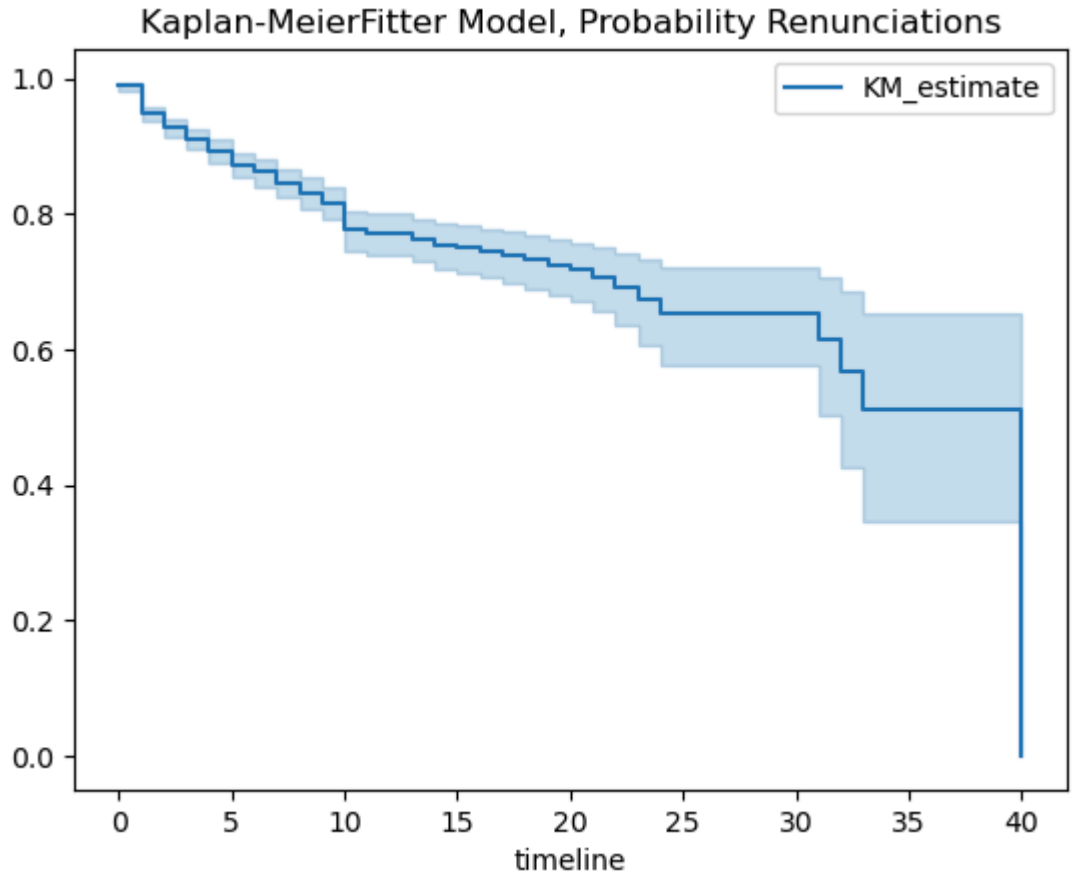
Out[ ]:

	YearsAtCompany	Attrition	AgeRange_18 to 27	AgeRange_28 to 37	AgeRange_38 to 47	AgeRange_48 to 57	AgeRang
0	6	1	0	0	1	0	
1	10	0	0	0	0	1	
2	0	1	0	1	0	0	
3	8	0	0	1	0	0	
4	2	0	1	0	0	0	

Now is the time to compute the survival curves with the Kaplan-MeierFitter.

In [ ]:
```
# Importing the Kaplan-MeierFitter package for the survival analysis
from lifelines import KaplanMeierFitter

# First we need to inicialize the Kaplan-MeierFitter model and we storage into a va
kmf = KaplanMeierFitter()

# Let's fit the data into the kmf function
kmf.fit(df_km_attencoded['YearsAtCompany'], event_observed = df_km_attencoded['Attr
# Checking the status of our curve
kmf.plot(title="Kaplan-MeierFitter Model, Probability Renunciations")
```

Out[ ]:
```
<Axes: title={'center': 'Kaplan-MeierFitter Model, Probability Renunciations'}, xl
abel='timeline'>
```



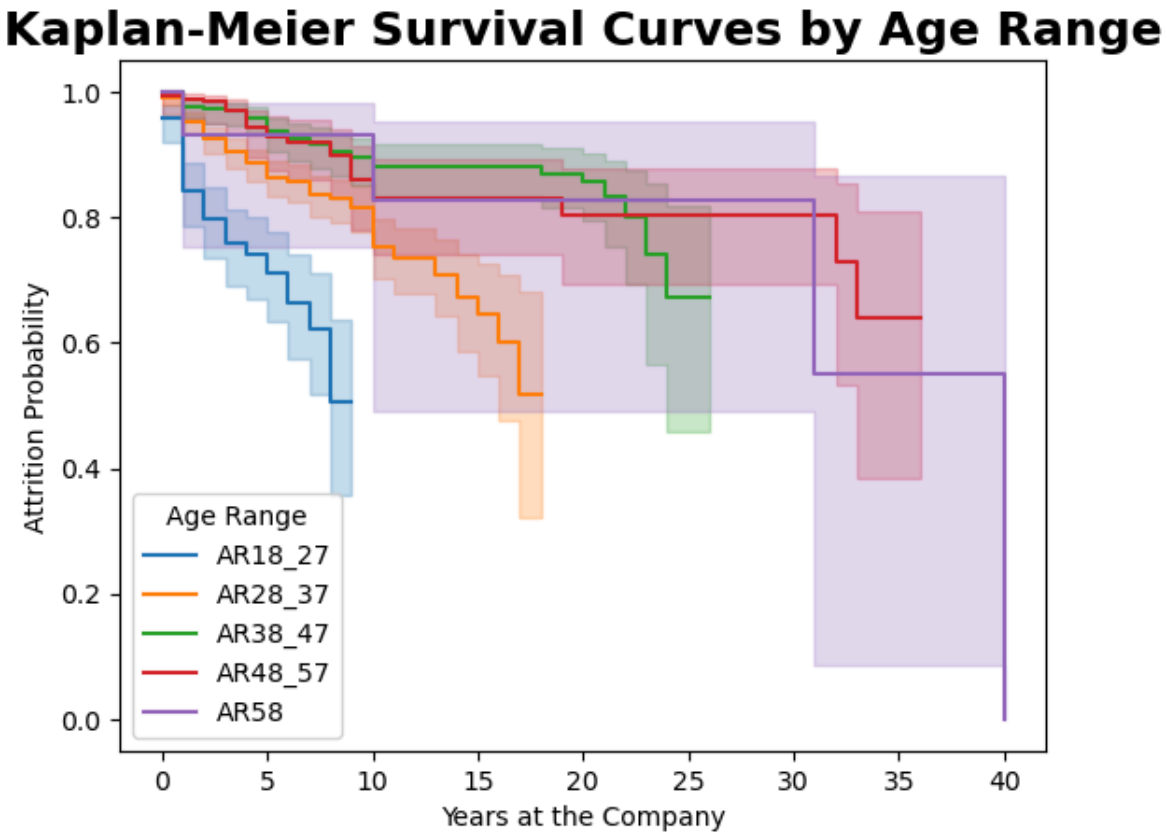
First impressions of the narrative lead us to believe that the first period of quitting the job can happen during the *first* and *second* years of employment. We need to consider where the plot takes big leaps. *The second period of resignations* is around **10 years** at the company. After that, a period of stability came, until the **25 years** at the company when it starts the retirement period.

To have a better idea, let's include the 'AgeRange' data.

In [ ]:
```
# Rename the age columns
df_km_attencoded.rename(columns={"AgeRange_18 to 27": "AR18_27", "AgeRange_28 to 37
df_km_attencoded.columns
```

Out[ ]:
```
Index(['YearsAtCompany', 'Attrition', 'AR18_27', 'AR28_37', 'AR38_47',
       'AR48_57', 'AR58'],
      dtype='object')
```

In [ ]:
```
# Plotting survival curves for each age range group
for age_range in ['AR18_27', 'AR28_37', 'AR38_47', 'AR48_57', 'AR58']:
    # Filter data for each age range group
    years_at_work = df_km_attencoded.loc[df_km_attencoded[age_range] == 1, 'YearsAt
    attrition = df_km_attencoded.loc[df_km_attencoded[age_range] == 1, 'Attrition']

    # Fit the Kaplan-Meier model for the current age range group
    kmf.fit(years_at_work, event_observed=attrition, label=age_range)

    # Plot the survival curve for the current age range group
    kmf.plot()

# Add labels and title
plt.xlabel('Years at the Company')
plt.ylabel('Attrition Probability')
plt.legend(title='Age Range')
plt.title('Kaplan-Meier Survival Curves by Age Range', fontsize=18, fontweight='bol

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



With the addition of the '*AgeRange*' values, we now have 5 lines. Each of them shows different information. For the range of **18 to 27**, we can appreciate that the probability of leaving the company is at the *first year at the company*, and then *at about nine years. In the range of **28 to 37**, the probability of leaving the company is also at the first year, then ten years later, and the last one comes when they are 16 or 17 years working fot he company. The ranges **38 to 47** and **48 to 57** show more stability at the company. For the last range, **more than 58**, we have three important breaks: the first year, then at ten years, and the last one at 31 years.*

4.8 Conclusion and Observations

After conducting an exploratory analysis of the database 'WA_Fn-UseC_-HR-Employee-Attrition', I was able to establish that the organization has a "good balance" between the levels of education, the gender and the age of the employees. From this point on, the organization can decide on the diversity policies it deems necessary.

*As for the levels of satisfaction, they are at **acceptable levels**, but it is recommended to pay attention to the percentage of discomfort, and to take steps to reduce the percent and avoid problems in the future, especially in jobs that are heavily engaged with their work.*

Career plans may need adjustments, because, as can be seen from the redundancies, the company is losing employees in lower age ranks. It could not be measured is the entry of new employees to have a better picture of the entry and exit of employees.

The salaries showed to be consistent with the levels of the educational level of employees.

The hours devoted to the Training are equal for all departments and are within normal ranges. Bear in mind that different departments have different numbers of employees and that all have the same average hours devoted to training.

The company had a good work-family balance for its employees, and there were no significant abnormalities in the work-family balance. It is recommended to continue with what is being done.

*With regard to **the resignations**, it was found that most of them could be related to salaries below average. In addition, people from **18 to 47** showed the highest mobility. If we add the level of education to the analysis, we can say that the company lost talent. However, it was not possible to verify whether the company made revenue to compensate for them.*

After the descriptive analysis, the next step will be to recreate the time to make a comparison of the metrics.