Project Name - **HR Analytics Project**

Name - Aman Mulla.

Batch - DS2307

Project Summary -

The proposed project centers on HR Analytics' role in addressing employee attrition and enhancing organizational effectiveness. Employee turnover imposes substantial costs on companies, including expenses related to hiring, training, and a decline in cumulative knowledge. Furthermore, high attrition hampers the establishment of a stable, experienced workforce, potentially impacting customer relations and operational efficiency. HR Analytics offers a strategic approach by leveraging data insights to identify attrition patterns, understand contributing factors, and implement proactive measures for retention. By analyzing employee data, such as performance metrics, engagement levels, and workplace dynamics, HR Analytics facilitates informed decision-making to optimize retention strategies. The project aims to explore these dynamics, illustrating how data-driven insights from HR Analytics can mitigate attrition's adverse impacts while fostering a more robust, engaged workforce.

HR Analytics:

Human resource analytics (HR analytics) is an area in the field of analytics that refers to applying analytic processes to the human resource department of an organization in the hope of improving employee performance and therefore getting a better return on investment. HR analytics does not just deal with gathering data on employee efficiency. Instead, it aims to provide insight into each process by gathering data and then using it to make relevant decisions about how to improve these processes.

Attrition in HR:

Attrition in human resources refers to the gradual loss of employees overtime. In general, relatively high attrition is problematic for companies. HR professionals often assume a leadership role in designing company compensation programs, work culture, and motivation systems that help the organization retain top employees.

We have below columns in our dataset,

Sure, here are brief descriptions of each input feature:

1. **Age**: Employee's age.
2. **Attrition**: Indicates whether the employee has left the company ('Yes' or 'No').
3. **BusinessTravel**: Frequency of travel for work ('Travel_Rarely', 'Travel_Frequently', 'Non-Travel').
4. **DailyRate**: Employee's daily rate of pay.
5. **Department**: Department in the company where the employee works ('Sales', 'Research & Development', 'Human Resources').
6. **DistanceFromHome**: Distance between employee's home and workplace.
7. **Education**: Employee's level of education (from 1 to 5).
8. **EducationField**: Field of education the employee studied.
9. **EmployeeCount**: Number of employees, typically constant (might not provide useful information).
10. **EmployeeNumber**: Unique identifier for each employee.
11. **EnvironmentSatisfaction**: Employee's satisfaction with the work environment (from 1 to 4).
12. **Gender**: Employee's gender ('Male' or 'Female').
13. **HourlyRate**: Employee's hourly rate of pay.
14. **JobInvolvement**: Level of job involvement (from 1 to 4).
15. **JobLevel**: Employee's job level (from 1 to 5).
16. **JobRole**: Role or position in the company.
17. **JobSatisfaction**: Employee's satisfaction with their job (from 1 to 4).
18. **MaritalStatus**: Employee's marital status ('Single', 'Married', 'Divorced').
19. **MonthlyIncome**: Employee's monthly income.
20. **MonthlyRate**: Employee's monthly rate of pay.
21. **NumCompaniesWorked**: Number of companies the employee has worked for.
22. **Over18**: Indicates if the employee is over 18 years old ('Yes').
23. **OverTime**: Indicates if the employee works overtime ('Yes' or 'No').
24. **PercentSalaryHike**: Percentage of the employee's salary increase.
25. **PerformanceRating**: Employee's performance rating (from 3 to 4).
26. **RelationshipSatisfaction**: Employee's satisfaction with work relationships (from 1 to 4).
27. **StandardHours**: Standard number of working hours (might not provide useful information).
28. **StockOptionLevel**: Level of stock options the employee has (from 0 to 3).
29. **TotalWorkingYears**: Total number of years the employee has worked.
30. **TrainingTimesLastYear**: Number of training times attended by the employee last year.

31. **WorkLifeBalance**: Employee's satisfaction with work-life balance (from 1 to 4).
32. **YearsAtCompany**: Number of years the employee has worked at the company.
33. **YearsInCurrentRole**: Number of years the employee has been in the current role.
34. **YearsSinceLastPromotion**: Number of years since the employee's last promotion.
35. **YearsWithCurrManager**: Number of years the employee has been with the current manager.

Problem Statement

High employee attrition rates pose significant challenges for organizations, resulting in increased operational costs, a loss of cumulative expertise, and potential disruptions in customer service. This project aims to investigate the impact of attrition on companies and explore how HR Analytics can effectively analyze and address this issue. The objective is to understand the factors contributing to employee turnover, identify patterns or trends within the data, and develop proactive strategies to minimize attrition. By leveraging data-driven insights, this study seeks to offer actionable recommendations for HR professionals and organizational leaders to optimize retention efforts and enhance overall workforce stability and performance.

Knowing data and variable in dataset

```
# Importing Necessary Libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

```
HR_data = pd.read_csv('/content/drive/MyDrive/DataSets/WA_Fn-UseC_-HR-Employee-Attrition.csv')
```

```
HR_data.head()
```

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

```
HR_data.columns
```

```
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')
```

```
# Will Check for shape of dataset

HR_data.shape
```

```
(1470, 35)
```

We have total 1470 rows and 35 column in our detaset.

Dataset Information

```
HR_data.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
```

From .info(), we can observe that there were varaibles with datatype of float,object and int only.

```
# Will check for description of dataset

HR_data.describe()
```

	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

From .describe() we can get count, mean, minimum value, maximum values and quirtile value for each numerical column.

```
HR_data.isnull().sum()
```

```
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
```

```
sns.heatmap(HR_data.isnull())
```

```
<Axes: >
```

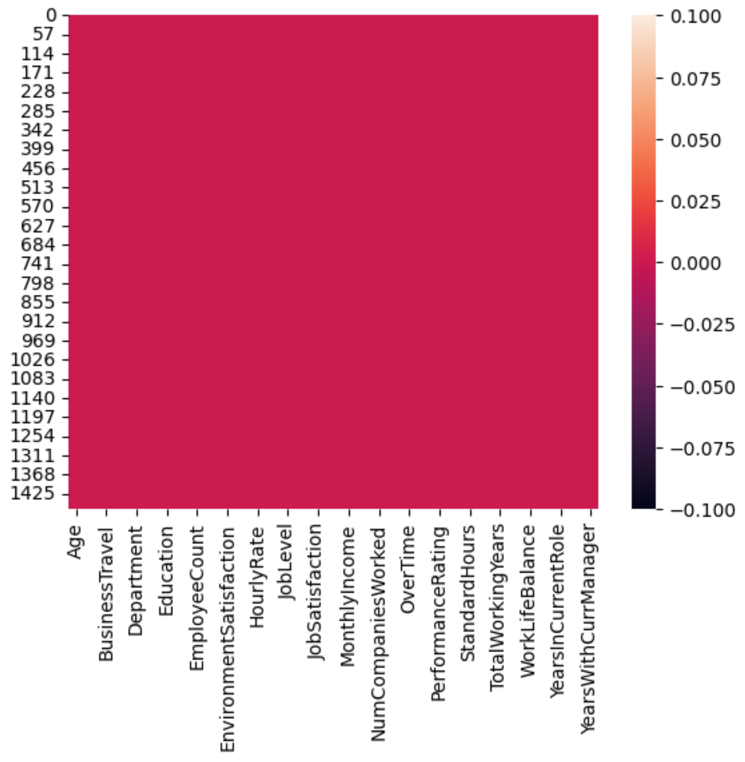
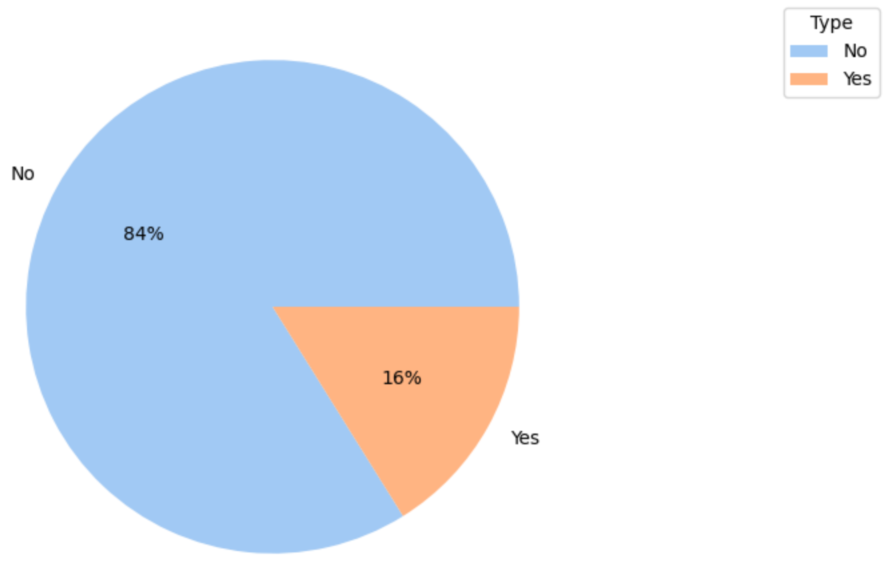


Chart - 1

Attrition Distribution

```
attrition_distribution = HR_data['Attrition'].value_counts()

attrition_distribution

labels = attrition_distribution.index

plt.figure(figsize=(8,6))
sns.set_palette('pastel')
plt.pie(attrition_distribution,labels=attrition_distribution.index,autopct='%.0f%%')
plt.legend(labels, title="Type", loc="upper right", bbox_to_anchor=(1, 0, 0.5, 1))

plt.show()
```



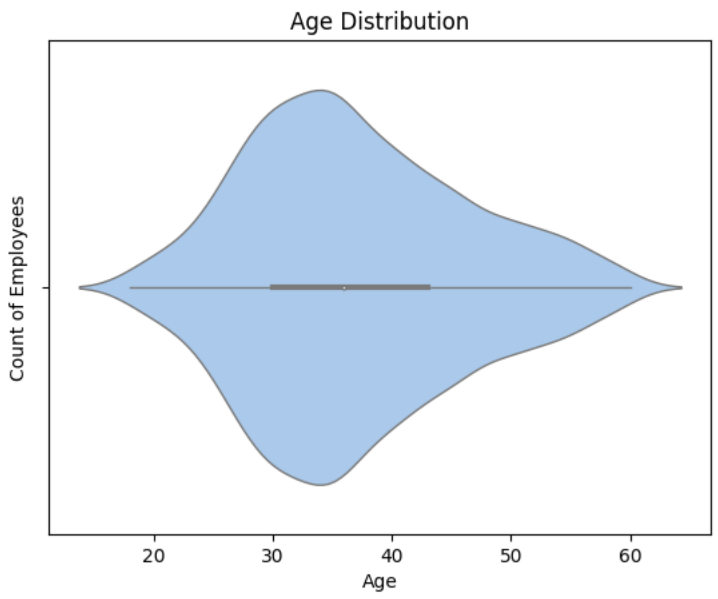
Insights from above chart:

- Displays the distribution of attrition types using a pie chart.
- The legend showing the labels (types of attrition - likely 'Yes' and 'No').
- This visualization include the proportion or percentage of employees who have experienced attrition ('Yes') compared to those who have not ('No'). This help in understanding the attrition rate within the dataset, providing a quick view of how many employees have left the company versus those who have stayed.

Chart - 2

Age Distribution

```
sns.violinplot(x=HR_data["Age"],linewidth=1, linecolor="k")

plt.xlabel('Age')
plt.ylabel('Count of Employees')
plt.title('Age Distribution')
plt.show()
```



Insights from above plot:

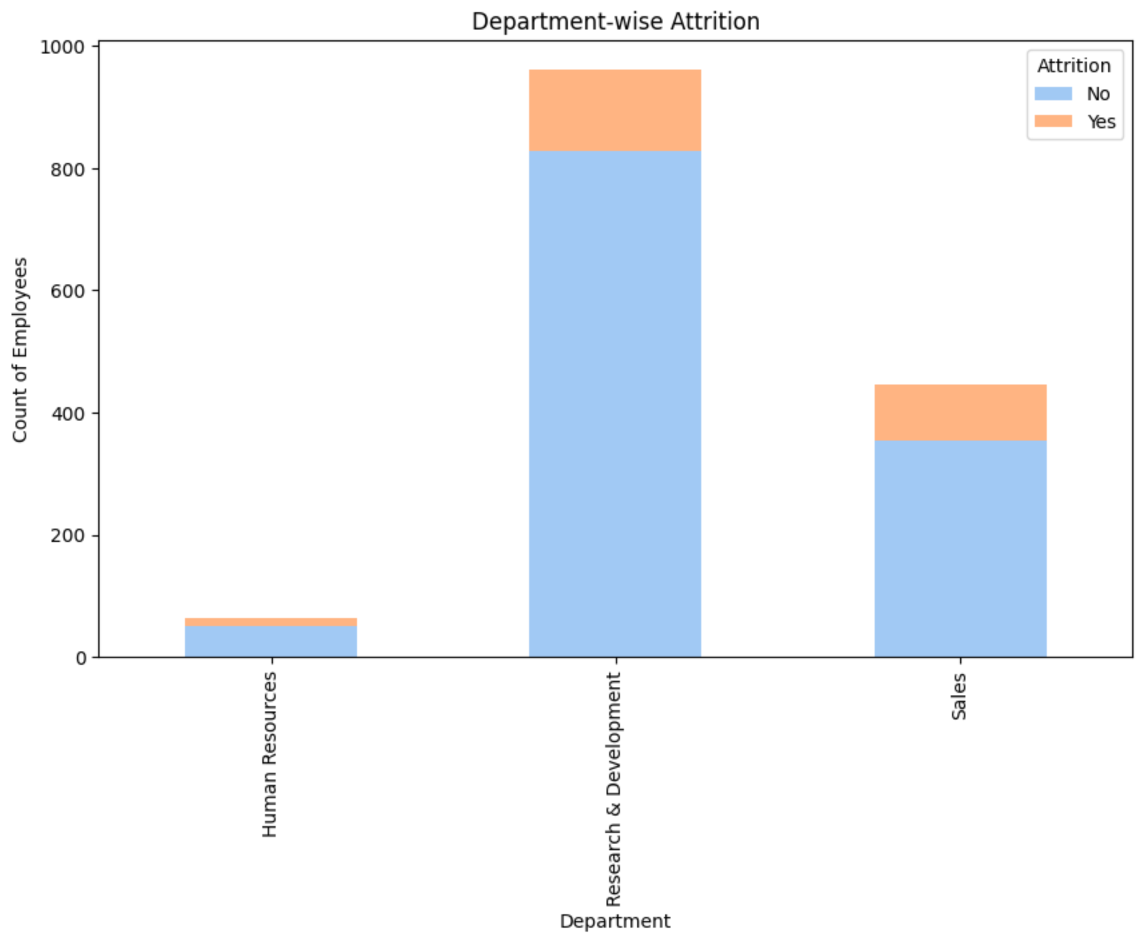
- The plot will show a distribution of ages among employees in your dataset. The spread and concentration of ages can be visualized through the shape of the violins.
- The width of the plot at different points indicates the density of employees in specific age groups. A wider section suggests more employees falling within that age range.
- A symmetric plot implies a more uniform distribution, while skewness can indicate an uneven distribution toward older or younger ages.

Chart - 3

Department-wise Attrition

```
department_attrition_count = HR_data.groupby(['Department', 'Attrition']).size().unstack()

department_attrition_count.plot(kind='bar', stacked=True, figsize=(10, 6))
plt.xlabel('Department')
plt.ylabel('Count of Employees')
plt.title('Department-wise Attrition')

plt.legend(title='Attrition')
```

```
<matplotlib.legend.Legend at 0x7f34830d97b0>
```



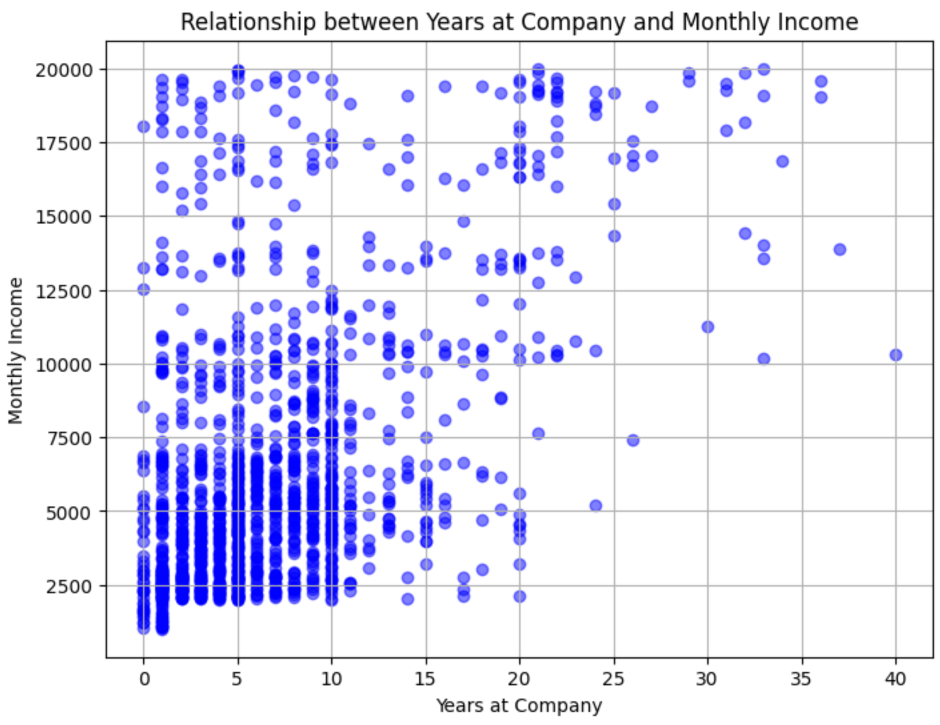
Insights from above plot:

- Each department will have segments representing both "Attrition" and "No Attrition" categories.
- By comparing the length of the bars for each department,gauge the overall size of the workforce in each department. Larger bars indicate a higher count of employees in that specific department.
- Within each department bar, the stacked segments will show the proportion of employees who have experienced attrition versus those who have not.

Chart - 4

Years at Company vs. Salary

```
# Extracting relevant columns
years_at_company = HR_data['YearsAtCompany']
monthly_income = HR_data['MonthlyIncome']

# Creating the scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(years_at_company, monthly_income, color='blue', alpha=0.5)

# Adding labels and title
plt.xlabel('Years at Company')
plt.ylabel('Monthly Income')
plt.title('Relationship between Years at Company and Monthly Income')

# Show plot
plt.grid(True)
plt.show()
```



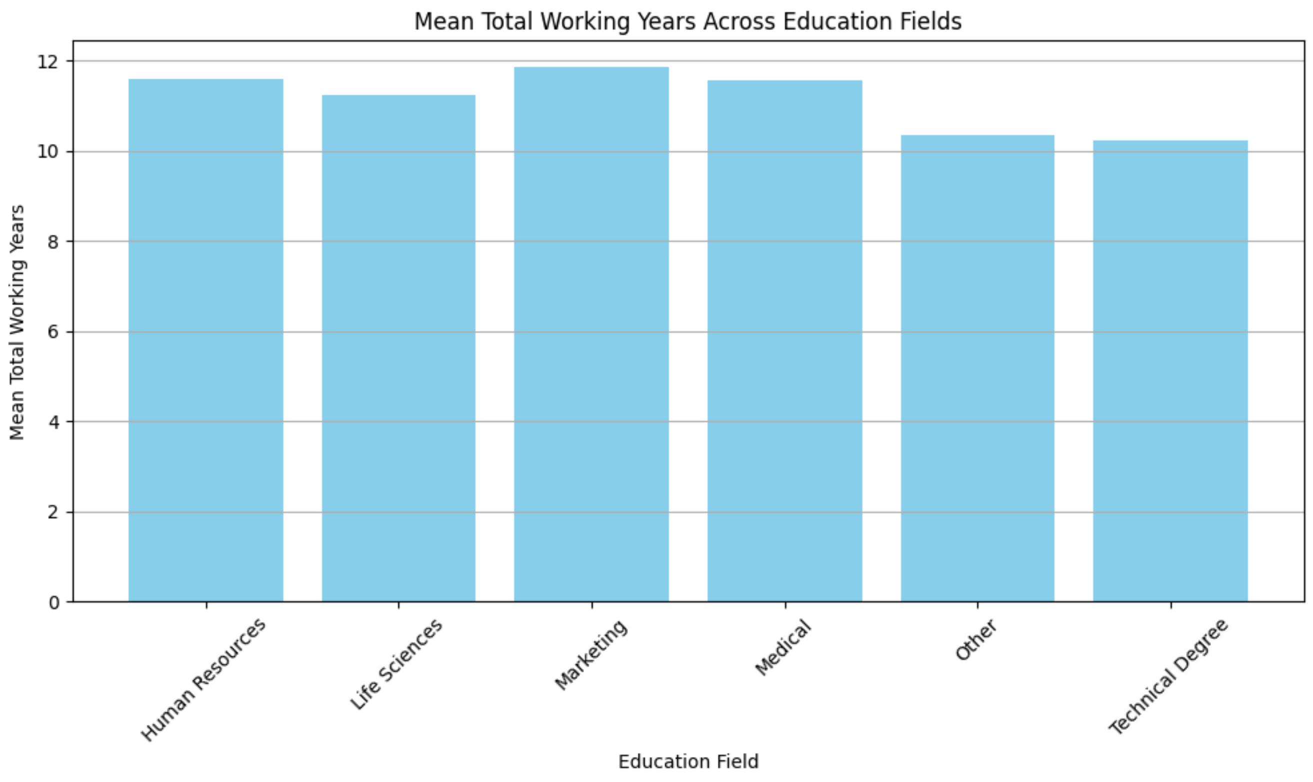
Insights from above plot:

- Upon observing the scatter plot, you may notice that there isn't a clear, direct linear relationship between "Years at Company" and "Monthly Income."
- There might be clusters or patterns visible in the data points. For example, you might notice groups of employees who have similar monthly incomes despite varying years of experience at the company. This could indicate the presence of categorical factors like job roles, departments, or education levels that impact income more significantly.
- The absence of a noticeable trend in the scatter plot might indicate that the relationship between these two variables is more complex or influenced by additional factors.

Chart - 5

Education Field and Total Working Years

```
education_working_years = HR_data.groupby('EducationField')['TotalWorkingYears'].mean().reset_index()

plt.figure(figsize=(10, 6))
plt.bar(education_working_years['EducationField'], education_working_years['TotalWorkingYears'], color='skyblue')
plt.xlabel('Education Field')
plt.ylabel('Mean Total Working Years')
plt.title('Mean Total Working Years Across Education Fields')

# Rotating x-axis labels for better readability if needed
plt.xticks(rotation=45)

plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



Insights from above graph:

- The visualization helps compare the mean total working years across different education fields.
- It allows for easy identification of which education fields tend to have longer or shorter average working experience.

Chart - 6

Pair Plot

```
sns.pairplot(HR_data)

plt.show()
```

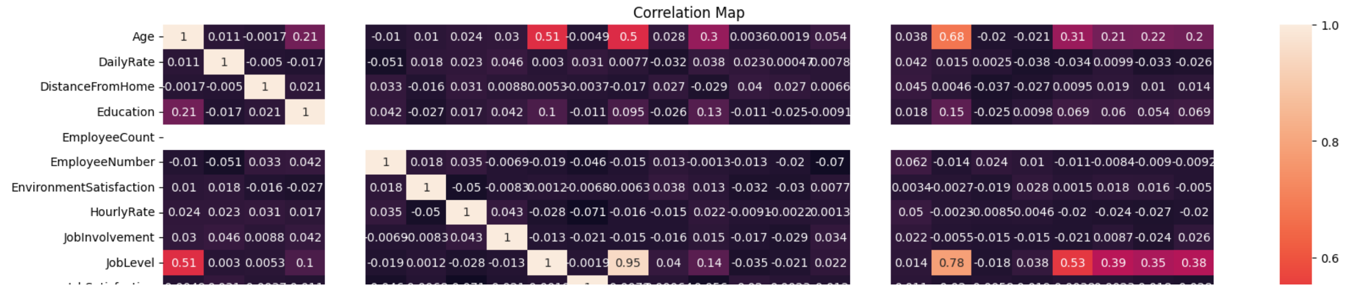
```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-15-3facd4133232> in <cell line: 1>()
----> 1 sns.pairplot(HR_data)
      2 
      3 plt.show()

                                  14 frames
/usr/local/lib/python3.10/dist-packages/matplotlib/cbook/__init__.py in <listcomp>(.0)
    832         """Clean dead weak references from the dictionary."""
    833         mapping = self._mapping
--> 834         to_drop = [key for key in mapping if key() is None]
    835         for key in to_drop:
    836             val = mapping.pop(key)

KeyboardInterrupt: 

 SEARCH STACK OVERFLOW

Error in callback <function _draw_all_if_interactive at 0x7f34c55fdfc0> (for post_execute):
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py in _draw_all_if_interactive()
    118 def _draw_all_if_interactive():
    119     if matplotlib.is_interactive():
--> 120         draw_all()
    121 
    122 

                                  18 frames
```

Chart - 12

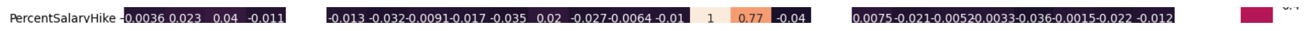
Heatmap

```
correlation_data = HR_data

correlation_matrix = correlation_data.corr()

plt.figure(figsize=(20,10))

sns.heatmap(correlation_matrix,annot=True)
plt.title('Correlation Map')
plt.show()
```

```
<ipython-input-16-ff073efd817b>:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future ver
  correlation_matrix = correlation_data.corr()
```



- Several that columns have high correlation: Age, JobLevel, MonthlyIncome, NumCompaniesWorked, TotalWorkingYears, YearsAtCompany, YearsInCurrentRole, YearsSinceLastPromotion, YearsWithCurrManager



To get VIF, will first define x and y varibales for our ML Model

```
HR_data.head()
```

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

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

HR_data_1=HR_data.copy()

le = LabelEncoder()

for i in HR_data_1.columns:
    if HR_data_1[i].dtype == 'object':
        HR_data_1[i] = le.fit_transform(HR_data_1[i])


HR_data_1.head()
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Education	EducationField	EmployeeCount	EmployeeNumber
0	41	1	2	1102	2	1	2	1	1	1
1	49	0	1	279	1	8	1	1	1	2
2	37	1	2	1373	1	2	2	4	1	4
3	33	0	1	1392	1	3	4	1	1	5
4	27	0	2	591	1	2	1	3	1	7

To get VIF, will first define x and y varibales for our ML Model

```
x = HR_data_1.drop(columns=['Attrition'])
y = HR_data_1['Attrition']

from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor

scalar = StandardScaler()
x_scaled=scalar.fit_transform(x)

# VIF

vif = pd.DataFrame()

vif['vif']=[variance_inflation_factor(x_scaled,i) for i in range(x_scaled.shape[1])]

vif['features'] = x.columns

vif
```

```
  return 1 - self.ssr/self.uncentered_tss
          vif                  features
 0   2.054226                       Age
 1   1.016808            BusinessTravel
 2   1.026401                 DailyRate
 3   1.942165                Department
 4   1.018096          DistanceFromHome
 5   1.065295                 Education
 6   1.016240            EducationField
 7        NaN             EmployeeCount
 8   1.022699            EmployeeNumber
 9   1.018022   EnvironmentSatisfaction
10   1.020038                    Gender
11   1.021984                HourlyRate
12   1.020836            JobInvolvement
13  11.825039                  JobLevel
14   1.894366                   JobRole
15   1.022894           JobSatisfaction
16   1.844040             MaritalStatus
17  11.055038             MonthlyIncome
18   1.015803               MonthlyRate
19   1.261958        NumCompaniesWorked
20        NaN                    Over18
21   1.028856                  OverTime
22   2.521583         PercentSalaryHike
23   2.519973         PerformanceRating
24   1.025574  RelationshipSatisfaction
25        NaN             StandardHours
26   1.828706          StockOptionLevel
27   4.824457         TotalWorkingYears
28   1.024312     TrainingTimesLastYear
29   1.018572           WorkLifeBalance
30   4.601995            YearsAtCompany
31   2.728269        YearsInCurrentRole
32   1.678897   YearsSinceLastPromotion
33   2.782981      YearsWithCurrManager
```

ML Model Implementation

ML Model - 1

Using all Variables for ML Model-1

```
# Importing Necessary Libraries

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score,confusion_matrix,roc_curve,roc_auc_score
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
```

```
HR_data_1.columns
```

```
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')
```

```
# For ML Model 1 Using all variables from dataset, putting selected featurs.

x = HR_data_1[['Age','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']]

y = HR_data_1['Attrition']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=348)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting Logistic Regression model to dataset

logistic_reg = LogisticRegression()

logistic_reg.fit(x_train,y_train)

# Make predictions on the test set
y_pred = logistic_reg.predict(x_test)

# Evaluate the model using various metrics
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy For ML Model 1:", accuracy)
print("Confusion Matrix For ML Model 1:\n", confusion)
print("Classification Report for ML Model 1:\n", classification_report_str)

# Initialize MinMaxScaler
# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the data using the scaler
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Define the hyperparameter grid
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Inverse of regularization strength
    'penalty': ['l1', 'l2'],  # Regularization penalty
    'solver': ['liblinear', 'lbfgs'],  # Solver for optimization
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=logistic_reg, param_grid=param_grid, scoring='accuracy', cv=5)

# Fit the grid search to your training data
grid_search.fit(x_train_scaled, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Get the best model
best_model = grid_search.best_estimator_

# Make predictions on the test set using the best model
y_pred = best_model.predict(x_test_scaled)

# Evaluate the model with the best hyperparameters
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Print the best hyperparameters, best model, and evaluation metrics
print("Best Hyperparameters:", best_params)
print("Best Model:", best_model)
print("Accuracy For ML Model 1(with Hyperparameter Tuning):", accuracy)
print("Confusion Matrix For ML Model 1(with Hyperparameter Tuning):\n", confusion)
print("Classification Report for ML Model 1(with Hyperparameter Tuning):\n", classification_report_str)
```

```
Classification Report for ML Model 1:
              precision    recall  f1-score   support

           0       0.85      0.98      0.91       297
           1       0.79      0.27      0.40        71

    accuracy                           0.85       368
   macro avg       0.82      0.63      0.66       368
weighted avg       0.84      0.85      0.81       368

Best Hyperparameters: {'C': 1, 'penalty': 'l1', 'solver': 'liblinear'}
Best Model: LogisticRegression(C=1, penalty='l1', solver='liblinear')
Accuracy For ML Model 1(with Hyperparameter Tuning): 0.845108695652174
Confusion Matrix For ML Model 1(with Hyperparameter Tuning):
 [[293   4]
 [ 53  18]]
Classification Report for ML Model 1(with Hyperparameter Tuning):
              precision    recall  f1-score   support

           0       0.85      0.99      0.91       297
           1       0.82      0.25      0.39        71

    accuracy                           0.85       368
   macro avg       0.83      0.62      0.65       368
weighted avg       0.84      0.85      0.81       368

/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (stat
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning:
30 fits failed out of a total of 120.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.
```

```
        estimator.fit(X_train, y_train, **fit_params)
      File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1162, in fit
        solver = _check_solver(self.solver, self.penalty, self.dual)
      File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 54, in _check_solver
        raise ValueError(
    ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

      warnings.warn(some_fits_failed_message, FitFailedWarning)
    /usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: UserWarning: One or more of the test scores are n
     0.84936652 0.84936652 0.85299877        nan 0.85844097 0.8566269
     0.87748252        nan 0.87658577 0.87295763 0.8747717         nan
     0.87386672 0.87386672 0.87386261        nan 0.8747717  0.8747717 ]
      warnings.warn(
```

Insights from ML Model 1:

- Before Tuning:

  Accuracy: 0.8451

  Classification Report Highlights:

  Precision for label 0: 0.85

  Precision for label 1: 0.79

  Recall for label 0: 0.98

  Recall for label 1: 0.27

  F1-score for label 0: 0.91

  F1-score for label 1: 0.40

- After Tuning:

  Accuracy: 0.8451 (same as before tuning)

  Classification Report Highlights:

  Precision for label 0: 0.85

  Precision for label 1: 0.82

  Recall for label 0: 0.99

  Recall for label 1: 0.25

  F1-score for label 0: 0.91

  F1-score for label 1: 0.39

The accuracy of the model remained the same, indicating that the tuning did not significantly impact overall correct predictions.Precision Improved for label 1 after tuning, which means that among the predicted positives, a higher proportion were actually true positives.

ML Model - 2

Using selected Variables for ML Model-2 (Neglecting verible whose VIF is more than 5)

```
# For ML Model 1 Using all variables from dataset, putting selected featurs.

x = HR_data_1[['Age','BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement','JobRole', 'JobSatisfaction',
       'MaritalStatus','MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager']]

y = HR_data_1['Attrition']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=348)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
```

```
x_test = scaler.fit_transform(x_test)

# Fitting Logistic Regression model to dataset

logistic_reg = LogisticRegression()

logistic_reg.fit(x_train,y_train)

# Make predictions on the test set
y_pred = logistic_reg.predict(x_test)

# Evaluate the model using various metrics
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy For ML Model 1:", accuracy)
print("Confusion Matrix For ML Model 1:\n", confusion)
print("Classification Report for ML Model 1:\n", classification_report_str)

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the data using the scaler
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Define the hyperparameter grid
param_grid = {
    'C': [0.001, 0.01, 0.1, 1, 10, 100],  # Inverse of regularization strength
    'penalty': ['l1', 'l2'],  # Regularization penalty
    'solver': ['liblinear', 'lbfgs'],  # Solver for optimization
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=logistic_reg, param_grid=param_grid, scoring='accuracy', cv=5)

# Fit the grid search to your training data
grid_search.fit(x_train_scaled, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Get the best model
best_model = grid_search.best_estimator_

# Make predictions on the test set using the best model
y_pred = best_model.predict(x_test_scaled)

# Evaluate the model with the best hyperparameters
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Print the best hyperparameters, best model, and evaluation metrics
print("Best Hyperparameters:", best_params)
print("Best Model:", best_model)
print("Accuracy For ML Model 1(with Hyperparameter Tuning):", accuracy)
print("Confusion Matrix For ML Model 1(with Hyperparameter Tuning):\n", confusion)
print("Classification Report for ML Model 1(with Hyperparameter Tuning):\n", classification_report_str)
```

```
Accuracy For ML Model 1: 0.842391304347826
Confusion Matrix For ML Model 1:
 [[290   7]
 [ 51  20]]
Classification Report for ML Model 1:
               precision    recall  f1-score   support

           0       0.85      0.98      0.91       297
           1       0.74      0.28      0.41        71

    accuracy                           0.84       368
   macro avg       0.80      0.63      0.66       368
weighted avg       0.83      0.84      0.81       368

Best Hyperparameters: {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}
Best Model: LogisticRegression(C=1, solver='liblinear')
Accuracy For ML Model 1(with Hyperparameter Tuning): 0.842391304347826
Confusion Matrix For ML Model 1(with Hyperparameter Tuning):
 [[290   7]
 [ 51  20]]
Classification Report for ML Model 1(with Hyperparameter Tuning):
               precision    recall  f1-score   support

           0       0.85      0.98      0.91       297
           1       0.74      0.28      0.41        71

    accuracy                           0.84       368
   macro avg       0.80      0.63      0.66       368
weighted avg       0.83      0.84      0.81       368

/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py:378: FitFailedWarning: 
30 fits failed out of a total of 120.
The score on these train-test partitions for these parameters will be set to nan.
If these failures are not expected, you can try to debug them by setting error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
30 fits failed with the following error:
Traceback (most recent call last):
  File "/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 1162, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
  File "/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py", line 54, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/usr/local/lib/python3.10/dist-packages/sklearn/model_selection/_search.py:952: UserWarning: One or more of the test scores are non
 0.84936652 0.84936652 0.85299877        nan 0.85844097 0.85572193
 0.86569724        nan 0.86751131 0.86751131 0.86661045        nan
 0.86569724 0.86478815 0.86661045        nan 0.86661045 0.86661045]
  warnings.warn(
```

Insights from ML Model 2

- Before Tuning:

  Accuracy: 0.842 (84.2%)

- After Tuning:

  Accuracy: 0.842 (84.2%)

The performance metrics (accuracy, confusion matrix, and classification report) remain unchanged before and after hyperparameter tuning.

ML Model - 3

Decision Tree Model

```
# Importing decision tree classifier

from sklearn.tree import DecisionTreeClassifier

# For ML Model 3, using selected variable from previous model which gives more accuracy.

x = HR_data_1[['Age','BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement','JobRole', 'JobSatisfaction',
       'MaritalStatus','MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager']]

y = HR_data_1['Attrition']


# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=348)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting Logistic Regression model to dataset

decision_tree = DecisionTreeClassifier()
decision_tree.fit(x_train, y_train)

# Make predictions on the test set
y_pred = decision_tree.predict(x_test)

# Evaluate the model using various metrics
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy For Decision Tree Model:", accuracy)
print("Confusion Matrix Decision Tree Model:\n", confusion)
print("Classification Report Decision Tree Model:\n", classification_report_str)

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the data using the scaler
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Define the hyperparameter grid

param_grid = {
    'criterion': ['gini','entropy'],
    'max_depth': range(10,15),
    'min_samples_leaf': range(2,6),
    'min_samples_split': range(3,8),
    'max_leaf_nodes': range(5,10)
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=decision_tree, param_grid=param_grid, scoring='accuracy', cv=5)

# Fit the grid search to your training data
grid_search.fit(x_train_scaled, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Get the best model
best_model = grid_search.best_estimator_

# Make predictions on the test set using the best model
```

```
y_pred = best_model.predict(x_test_scaled)

# Evaluate the model with the best hyperparameters
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Print the best hyperparameters, best model, and evaluation metrics
print("Best Hyperparameters:", best_params)
print("Best Model:", best_model)
print("Accuracy For Decision Tree Model (with Hyperparameter Tuning):", accuracy)
print("Confusion Matrix For Decision Tree Model (with Hyperparameter Tuning):\n", confusion)
print("Classification Report for Decision Tree Model (with Hyperparameter Tuning):\n", classification_report_str)
```

```
Accuracy For Decision Tree Model: 0.7608695652173914
Confusion Matrix Decision Tree Model:
 [[251  46]
 [ 42  29]]
Classification Report Decision Tree Model:
              precision    recall  f1-score   support

           0       0.86      0.85      0.85       297
           1       0.39      0.41      0.40        71

    accuracy                           0.76       368
   macro avg       0.62      0.63      0.62       368
weighted avg       0.77      0.76      0.76       368

Best Hyperparameters: {'criterion': 'gini', 'max_depth': 10, 'max_leaf_nodes': 6, 'min_samples_leaf': 2, 'min_samples_split': 3}
Best Model: DecisionTreeClassifier(max_depth=10, max_leaf_nodes=6, min_samples_leaf=2,
                       min_samples_split=3)
Accuracy For Decision Tree Model (with Hyperparameter Tuning): 0.8260869565217391
Confusion Matrix For Decision Tree Model (with Hyperparameter Tuning):
 [[289   8]
 [ 56  15]]
Classification Report for Decision Tree Model (with Hyperparameter Tuning):
              precision    recall  f1-score   support

           0       0.84      0.97      0.90       297
           1       0.65      0.21      0.32        71

    accuracy                           0.83       368
   macro avg       0.74      0.59      0.61       368
weighted avg       0.80      0.83      0.79       368
```

Insights from Decision Tree Model-

- Before Tuning:

  Accuracy: 76.08%

  Classification Report:

  Precision for class 0: 86%

  Recall for class 0: 85%

  F1-score for class 0: 85%

  Precision for class 1: 39%

  Recall for class 1: 41%

  F1-score for class 1: 40%

- After Tuning:

  Accuracy: 82.61%

  Classification Report:

  Precision for class 0: 84%

  Recall for class 0: 97%

  F1-score for class 0: 90%

  Precision for class 1: 65%

  Recall for class 1: 21%

  F1-score for class 1: 32%

The model's accuracy increased from 76.08% to 82.61% after hyperparameter tuning. This indicates a better overall prediction capability.The model after tuning performs better in terms of overall accuracy and correctly identifying class 0 instances (higher precision, recall, and F1-

score).

ML Model - 4

RandomForestClassifier

```
# Importing decision tree classifier

from sklearn.tree import DecisionTreeClassifier

# For ML Model RandomForestClassifier, using selected variable from previous model which gives more accuracy.

x = HR_data_1[['Age','BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement','JobRole', 'JobSatisfaction',
       'MaritalStatus','MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager']]

y = HR_data_1['Attrition']

# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=348)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting RandomForest model, first will import from sklearn package

from sklearn.ensemble import RandomForestClassifier

random_for = RandomForestClassifier()

random_for.fit(x_train,y_train)

# Make predictions on the test set
y_pred = random_for.predict(x_test)

# Evaluate the model using various metrics
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy For RandomForestClassifier:", accuracy)
print("Confusion Matrix RandomForestClassifier:\n", confusion)
print("Classification Report RandomForestClassifier:\n", classification_report_str)

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the data using the scaler
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Define the hyperparameter grid

param_grid = {
    'criterion': ['gini'],
    'max_depth': range(10,12),
    'min_samples_leaf': range(2,4),
    'min_samples_split': range(3,8),
    'max_leaf_nodes': range(5,10)
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=random_for, param_grid=param_grid, scoring='accuracy', cv=5)

# Fit the grid search to your training data
grid_search.fit(x_train_scaled, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Get the best model
best_model = grid_search.best_estimator_
```

```
# Make predictions on the test set using the best model
y_pred = best_model.predict(x_test_scaled)

# Evaluate the model with the best hyperparameters
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Print the best hyperparameters, best model, and evaluation metrics
print("Best Hyperparameters:", best_params)
print("Best Model:", best_model)
print("Accuracy For RandomForestClassifier (with Hyperparameter Tuning):", accuracy)
print("Confusion Matrix For RandomForestClassifier (with Hyperparameter Tuning):\n", confusion)
print("Classification Report for RandomForestClassifier (with Hyperparameter Tuning):\n", classification_report_str)
```

```
Accuracy For RandomForestClassifier: 0.8315217391304348
Confusion Matrix RandomForestClassifier:
 [[295   2]
 [ 60  11]]
Classification Report RandomForestClassifier:
               precision    recall  f1-score   support

           0       0.83      0.99      0.90       297
           1       0.85      0.15      0.26        71

    accuracy                           0.83       368
   macro avg       0.84      0.57      0.58       368
weighted avg       0.83      0.83      0.78       368

Best Hyperparameters: {'criterion': 'gini', 'max_depth': 11, 'max_leaf_nodes': 9, 'min_samples_leaf': 3, 'min_samples_split': 6}
Best Model: RandomForestClassifier(max_depth=11, max_leaf_nodes=9, min_samples_leaf=3,
                       min_samples_split=6)
Accuracy For RandomForestClassifier (with Hyperparameter Tuning): 0.8097826086956522
Confusion Matrix For RandomForestClassifier (with Hyperparameter Tuning):
 [[297   0]
 [ 70   1]]
Classification Report for RandomForestClassifier (with Hyperparameter Tuning):
               precision    recall  f1-score   support

           0       0.81      1.00      0.89       297
           1       1.00      0.01      0.03        71

    accuracy                           0.81       368
   macro avg       0.90      0.51      0.46       368
weighted avg       0.85      0.81      0.73       368
```

Insights from ML Model 4-

- Before Tuning:

  Accuracy: 83.15%

- After Tuning:

  Accuracy: 80.98%

Accuracy: After tuning, the accuracy decreased slightly from 83.15% to 80.98%.

Confusion Matrix: Before tuning, the model misclassified 62 instances. After tuning, it misclassified 70 instances of class 1.

Precision, Recall, F1-score: For class 0 (majority class), precision and recall remained fairly similar after tuning. For class 1, precision was perfect (1.00) after tuning but recall dropped significantly from 0.15 to almost 0.01, indicating the model's inability to correctly identify instances of class 1.

Before Tuning: The model had a higher overall accuracy but struggled with recall and precision for the minority class (class 1). After Tuning: Although the accuracy decreased slightly, there was an improvement in precision for class 1 but at the cost of significantly lower recall.

ML Model - 5

KNeighborsClassifier

```
# Importing decision tree classifier

from sklearn.tree import DecisionTreeClassifier

# For ML Model 7,Using all KNeighborsClassifier

x = HR_data_1[['Age','BusinessTravel', 'DailyRate', 'Department',
       'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
       'EmployeeNumber', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate',
       'JobInvolvement','JobRole', 'JobSatisfaction',
       'MaritalStatus','MonthlyRate', 'NumCompaniesWorked',
       'Over18', 'OverTime', 'PercentSalaryHike', 'PerformanceRating',
       'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
       'TrainingTimesLastYear', 'WorkLifeBalance',
       'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager']]

y = HR_data_1['Attrition']


# splitting data into train and test set.

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=348)

# Transforming data standardization

scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.fit_transform(x_test)

# Fitting KNeighborsClassifier model, first will import from sklearn package

from sklearn.neighbors import KNeighborsClassifier

kNN = KNeighborsClassifier()

kNN.fit(x_train,y_train)

# Make predictions on the test set
y_pred = kNN.predict(x_test)

# Evaluate the model using various metrics
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
classification_report_str = classification_report(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy For KNeighborsClassifier:", accuracy)
print("Confusion Matrix for KNeighborsClassifier:\n", confusion)
print("Classification Report for KNeighborsClassifier:\n", classification_report_str)

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the data using the scaler
```