Solving Human Resources Issues

Preparation

Source of data

Employee Data

The data contains demographic details, work-related metrics and attrition flag.

- **EmployeeId** - Employee Identifier
- **Attrition** - Did the employee attrition? (0=no, 1=yes)
- **Age** - Age of the employee
- **BusinessTravel** - Travel commitments for the job
- **DailyRate** - Daily salary
- **Department** - Employee Department
- **DistanceFromHome** - Distance from work to home (in km)
- **Education** - 1-Below College, 2-College, 3-Bachelor, 4-Master, 5-Doctor
- **EducationField** - Field of Education
- **EnvironmentSatisfaction** - 1-Low, 2-Medium, 3-High, 4-Very High
- **Gender** - Employee's gender
- **HourlyRate** - Hourly salary
- **JobInvolvement** - 1-Low, 2-Medium, 3-High, 4-Very High
- **JobLevel** - Level of job (1 to 5)
- **JobRole** - Job Roles
- **JobSatisfaction** - 1-Low, 2-Medium, 3-High, 4-Very High
- **MaritalStatus** - Marital Status
- **MonthlyIncome** - Monthly salary
- **MonthlyRate** - Monthly rate
- **NumCompaniesWorked** - Number of companies worked at
- **Over18** - Over 18 years of age?
- **OverTime** - Overtime?
- **PercentSalaryHike** - The percentage increase in salary last year
- **PerformanceRating** - 1-Low, 2-Good, 3-Excellent, 4-Outstanding
- **RelationshipSatisfaction** - 1-Low, 2-Medium, 3-High, 4-Very High
- **StandardHours** - Standard Hours
- **StockOptionLevel** - Stock Option Level
- **TotalWorkingYears** - Total years worked
- **TrainingTimesLastYear** - Number of training attended last year
- **WorkLifeBalance** - 1-Low, 2-Good, 3-Excellent, 4-Outstanding
- **YearsAtCompany** - Years at Company

- **YearsInCurrentRole** - Years in the current role
- **YearsSinceLastPromotion** - Years since the last promotion
- **YearsWithCurrManager** - Years with the current manager

Acknowledgements

IBM Watson Analytics Use Case for HR Retaining Valuable Employees

Import the module and dataset

In [1]:
```
import os
import time
import warnings
# ignore warning
warnings.filterwarnings("ignore")

%load_ext autoreload
%autoreload 2

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

# Set pandas display options to maximize output visibility
pd.set_option("display.max_columns", None)
pd.set_option("display.max_colwidth", None)
pd.set_option("display.width", 0)
pd.set_option("display.expand_frame_repr", False)
```

Data imported into dataframe 'employee_df'

In [2]:
```
# Import dataset
datasource = "dataset/employee_data.csv"
employee_df = pd.read_csv(datasource,encoding='windows-1252')
```

EDA

Data Info

In [3]:
```
print("Dataframe Info:\n")
employee_df.info()
```

```
Dataframe Info:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   EmployeeId                1470 non-null   int64  
 1   Age                       1470 non-null   int64  
 2   Attrition                 1058 non-null   float64
 3   BusinessTravel            1470 non-null   object 
 4   DailyRate                 1470 non-null   int64  
 5   Department                1470 non-null   object 
 6   DistanceFromHome          1470 non-null   int64  
 7   Education                 1470 non-null   int64  
 8   EducationField            1470 non-null   object 
 9   EmployeeCount             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: float64(1), int64(26), object(8)
memory usage: 402.1+ KB
```

In [4]:
```
employee_df.sample(10)
```

Out[4]:

	EmployeeId	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromI
326	327	45	1.0	Travel_Frequently	306	Sales	
566	567	26	0.0	Travel_Frequently	921	Research & Development	
1164	1165	35	NaN	Travel_Rarely	1224	Sales	
156	157	56	NaN	Non-Travel	667	Research & Development	
346	347	26	0.0	Travel_Frequently	496	Research & Development	
1134	1135	44	1.0	Travel_Rarely	621	Research & Development	
456	457	27	0.0	Travel_Frequently	994	Sales	
526	527	36	1.0	Travel_Rarely	660	Research & Development	
1293	1294	43	NaN	Travel_Frequently	1422	Sales	
1230	1231	54	0.0	Travel_Rarely	397	Human Resources	

In [5]:

```
# Displaying the number of unique values in each column
employee_df.nunique().to_frame()
```

Out[5]:

	0
EmployeeId	1470
Age	43
Attrition	2
BusinessTravel	3
DailyRate	886
Department	3
DistanceFromHome	29
Education	5
EducationField	6
EmployeeCount	1
EnvironmentSatisfaction	4
Gender	2
HourlyRate	71
JobInvolvement	4
JobLevel	5
JobRole	9
JobSatisfaction	4
MaritalStatus	3
MonthlyIncome	1349
MonthlyRate	1427
NumCompaniesWorked	10
Over18	1
OverTime	2
PercentSalaryHike	15
PerformanceRating	2
RelationshipSatisfaction	4
StandardHours	1
StockOptionLevel	4
TotalWorkingYears	40
TrainingTimesLastYear	7

	0
WorkLifeBalance	4
YearsAtCompany	37
YearsInCurrentRole	19
YearsSinceLastPromotion	16
YearsWithCurrManager	18

In [6]:
```
# Unique values from the 'Attrition' column
employee_df['Attrition'].unique()
```

Out[6]:
```
array([nan,  1.,  0.])
```

Data format

In [7]:
```
# Listing type data kolom

employee_col_numeric_int = [
    'Age', 'DailyRate', 'DistanceFromHome',
    'EmployeeCount', 'HourlyRate',
    'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
    'PercentSalaryHike', 'StandardHours',
    'TotalWorkingYears', 'TrainingTimesLastYear',
    'YearsAtCompany', 'YearsInCurrentRole',
    'YearsSinceLastPromotion', 'YearsWithCurrManager'
]
employee_col_numeric_int = pd.Index(employee_col_numeric_int)

employee_col_categorical = [
    'BusinessTravel', 'Department', 'Education', 'EducationField',
    'EnvironmentSatisfaction', 'Gender', 'JobInvolvement', 'JobLevel',
    'JobRole', 'JobSatisfaction', 'MaritalStatus', 'Over18', 'OverTime',
    'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel',
    'WorkLifeBalance'
]
employee_col_categorical = pd.Index(employee_col_categorical)

employee_col_object = ['EmployeeId']
employee_col_object = pd.Index(employee_col_object)
```

In [8]:
```
# Convert to integer
employee_df[employee_col_numeric_int] = employee_df[employee_col_numeric_int].astyp

# Conversion to category
employee_df[employee_col_categorical] = employee_df[employee_col_categorical].astyp

# Convert object columns (outside of other categories)
employee_df['EmployeeId'] = employee_df['EmployeeId'].astype('object')

employee_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype   
---  ------                    --------------  -----   
 0   EmployeeId                1470 non-null   object  
 1   Age                       1470 non-null   int64   
 2   Attrition                 1058 non-null   float64 
 3   BusinessTravel            1470 non-null   category
 4   DailyRate                 1470 non-null   int64   
 5   Department                1470 non-null   category
 6   DistanceFromHome          1470 non-null   int64   
 7   Education                 1470 non-null   category
 8   EducationField            1470 non-null   category
 9   EmployeeCount             1470 non-null   int64   
 10  EnvironmentSatisfaction   1470 non-null   category
 11  Gender                    1470 non-null   category
 12  HourlyRate                1470 non-null   int64   
 13  JobInvolvement            1470 non-null   category
 14  JobLevel                  1470 non-null   category
 15  JobRole                   1470 non-null   category
 16  JobSatisfaction           1470 non-null   category
 17  MaritalStatus             1470 non-null   category
 18  MonthlyIncome             1470 non-null   int64   
 19  MonthlyRate               1470 non-null   int64   
 20  NumCompaniesWorked        1470 non-null   int64   
 21  Over18                    1470 non-null   category
 22  OverTime                  1470 non-null   category
 23  PercentSalaryHike         1470 non-null   int64   
 24  PerformanceRating         1470 non-null   category
 25  RelationshipSatisfaction  1470 non-null   category
 26  StandardHours             1470 non-null   int64   
 27  StockOptionLevel          1470 non-null   category
 28  TotalWorkingYears         1470 non-null   int64   
 29  TrainingTimesLastYear     1470 non-null   int64   
 30  WorkLifeBalance           1470 non-null   category
 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: category(17), float64(1), int64(16), object(1)
memory usage: 234.3+ KB
```

Zero-Variance Feature

In [9]:

```
# Removing Zero-Variance Feature
try:
    print("Column with only one unique value:")
    const_cols = [col for col in employee_df.columns if employee_df[col].nunique()
    print(const_cols)

    employee_df = employee_df.drop(columns=const_cols)

    print("Columns with only one unique value have been removed.")
```

```python
except KeyError as e:
    print(f"Column not found when dropping: {e}")
except Exception as e:
    print(f"Terjadi error: {e}")
finally:
    print("Zero-Variance column cleaning is complete.")
```

```
Column with only one unique value:
['EmployeeCount', 'Over18', 'StandardHours']
Columns with only one unique value have been removed.
Zero-Variance column cleaning is complete.
```

Null, NaN

In [10]:

```python
employee_df.isnull().sum().to_frame()
```

Out[10]:

	0
EmployeeId	0
Age	0
Attrition	412
BusinessTravel	0
DailyRate	0
Department	0
DistanceFromHome	0
Education	0
EducationField	0
EnvironmentSatisfaction	0
Gender	0
HourlyRate	0
JobInvolvement	0
JobLevel	0
JobRole	0
JobSatisfaction	0
MaritalStatus	0
MonthlyIncome	0
MonthlyRate	0
NumCompaniesWorked	0
OverTime	0
PercentSalaryHike	0
PerformanceRating	0
RelationshipSatisfaction	0
StockOptionLevel	0
TotalWorkingYears	0
TrainingTimesLastYear	0
WorkLifeBalance	0
YearsAtCompany	0
YearsInCurrentRole	0

	0
YearsSinceLastPromotion	0
YearsWithCurrManager	0

Handling NaN in the Attrition column by dropping NaN values.

In [11]:
```
# Separate rows with NaN values in the 'Attrition' column.
employee_attrition_nan = employee_df[employee_df['Attrition'].isna()]

# Save to CSV file will be used for the prediction file.
employee_attrition_nan.to_csv('saved/employee_attrition_nan.csv', index=False)

# Drop rows with NaN values in the 'Attrition' column to create a clean dataset.
employee_df_clean = employee_df.dropna(subset=['Attrition'])

# Convert the 'Attrition' column to integer type
employee_df_clean['Attrition'] = employee_df_clean['Attrition'].astype(int)

# Show the number of null values after cleaning
print("\nNumber of null values after being cleaned:\n")
employee_df_clean.isnull().sum().to_frame()
```

```
Number of null values after being cleaned:
```

Out[11]:

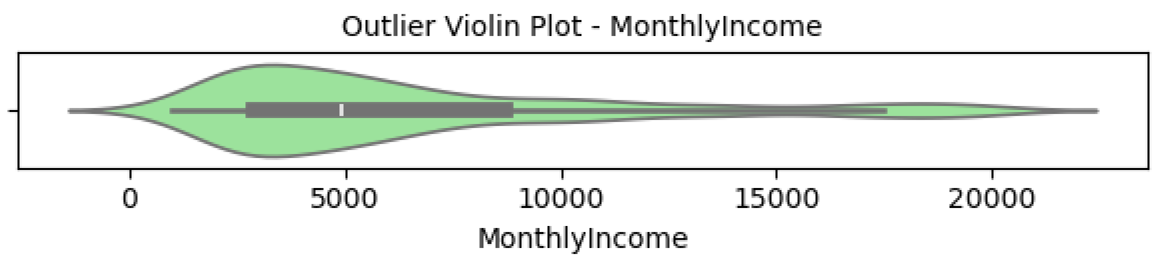
	0
EmployeeId	0
Age	0
Attrition	0
BusinessTravel	0
DailyRate	0
Department	0
DistanceFromHome	0
Education	0
EducationField	0
EnvironmentSatisfaction	0
Gender	0
HourlyRate	0
JobInvolvement	0
JobLevel	0
JobRole	0
JobSatisfaction	0
MaritalStatus	0
MonthlyIncome	0
MonthlyRate	0
NumCompaniesWorked	0
OverTime	0
PercentSalaryHike	0
PerformanceRating	0
RelationshipSatisfaction	0
StockOptionLevel	0
TotalWorkingYears	0
TrainingTimesLastYear	0
WorkLifeBalance	0
YearsAtCompany	0
YearsInCurrentRole	0

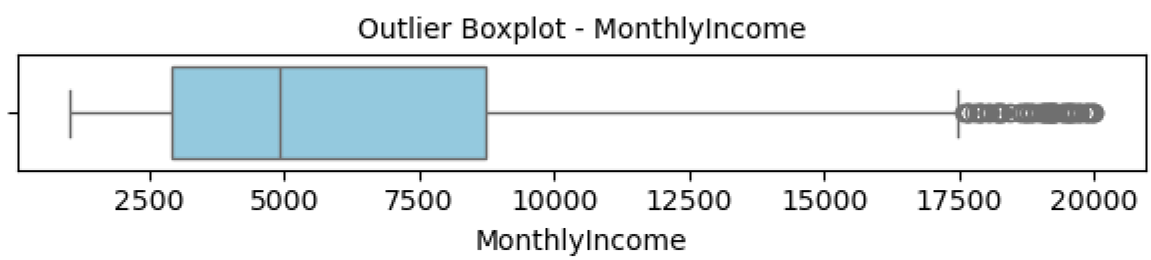
	0
YearsSinceLastPromotion	0
YearsWithCurrManager	0

Outlier

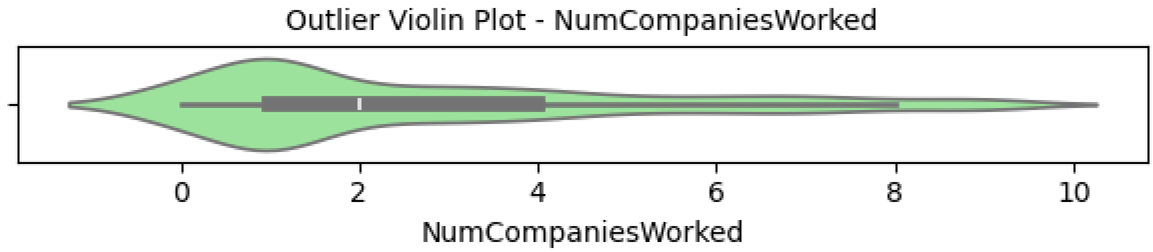
In [12]:
```
# Identify outliers using the Interquartile Range (IQR) method.
from scripts.runPlot_DetectOutliersIQR import plot_detect_outliers_iqr

print("\nOutliers detected using IQR method:\n")
ignore_outlier_cols = ['Attrition']
plot_detect_outliers_iqr(df=employee_df_clean, ignore_cols=ignore_outlier_cols)
```

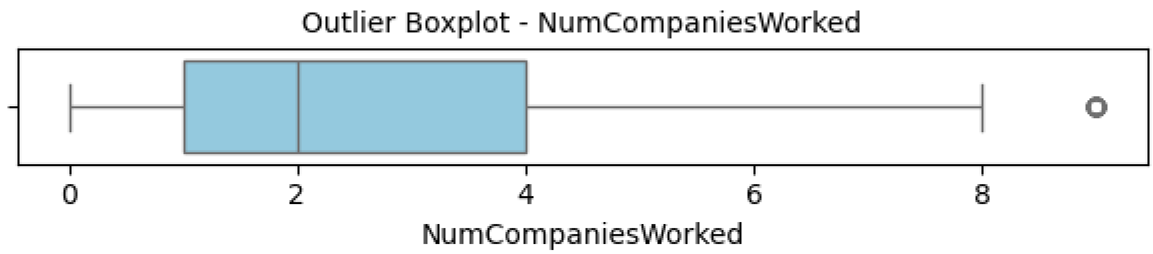
```
Outliers detected using IQR method:

[INFO] No outliers detected in 'Age'.
[INFO] No outliers detected in 'DailyRate'.
[INFO] No outliers detected in 'DistanceFromHome'.
[INFO] No outliers detected in 'HourlyRate'.
[INFO] No outliers detected in 'MonthlyRate'.
[INFO] No outliers detected in 'PercentSalaryHike'.

[OUTLIER] 'MonthlyIncome' — 64 values detected
   Q1 = 2900.25, Median = 4903.50, Q3 = 8736.50
   Lower bound = -5854.12, Upper bound = 17490.88
   Values:
  { 17584: 1x, 17639: 1x, 17650: 1x, 17665: 1x, 17779: 1x, 17856: 1x, 17861: 2x,
  17924: 1x, 18041: 1x, 18061: 1x, 18172: 1x, 18200: 1x, 18213: 1x, 18265: 1x,
  18300: 1x, 18303: 1x, 18430: 1x, 18606: 1x, 18665: 1x, 18711: 1x, 18722: 1x,
  18740: 1x, 18789: 1x, 18824: 1x, 18844: 1x, 18947: 1x, 19033: 1x, 19045: 1x,
  19068: 1x, 19081: 1x, 19094: 1x, 19141: 1x, 19144: 1x, 19187: 1x, 19189: 1x,
  19190: 1x, 19197: 1x, 19202: 1x, 19232: 1x, 19237: 1x, 19246: 1x, 19272: 1x,
  19406: 1x, 19419: 1x, 19436: 1x, 19502: 1x, 19513: 1x, 19517: 1x, 19537: 1x,
  19545: 1x, 19566: 1x, 19626: 1x, 19627: 1x, 19701: 1x, 19717: 1x, 19740: 1x,
  19845: 1x, 19847: 1x, 19859: 1x, 19926: 1x, 19943: 1x, 19973: 1x, 19999: 1x }
```



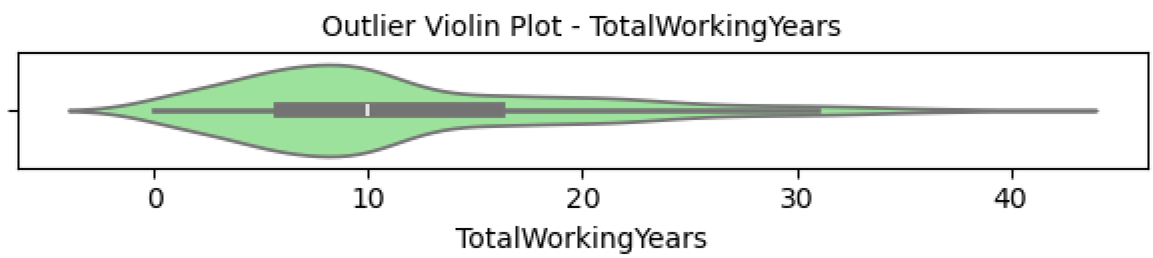


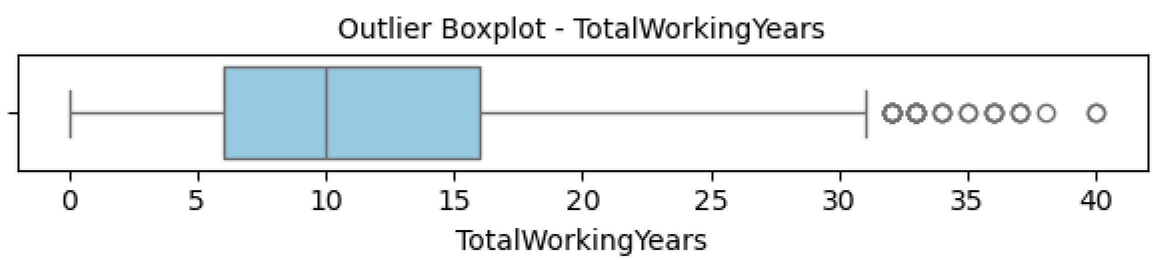
```
[OUTLIER] 'NumCompaniesWorked' — 43 values detected
   Q1 = 1.00, Median = 2.00, Q3 = 4.00
   Lower bound = -3.50, Upper bound = 8.50
   Values:
  { 9: 43x }
```





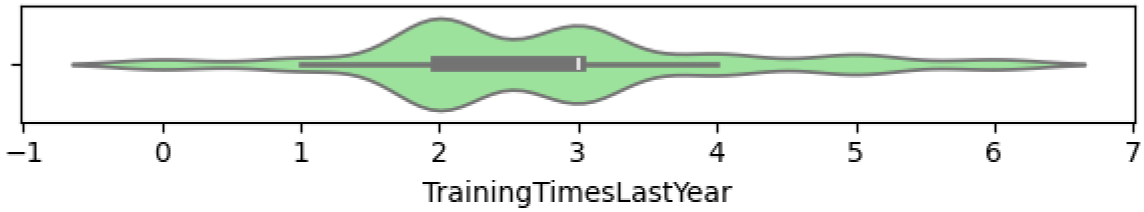

```
[OUTLIER] 'TotalWorkingYears' — 29 values detected
   Q1 = 6.00, Median = 10.00, Q3 = 16.00
   Lower bound = -9.00, Upper bound = 31.00
   Values:
  { 32: 7x, 33: 7x, 34: 3x, 35: 2x, 36: 4x, 37: 3x, 38: 1x, 40: 2x }
```



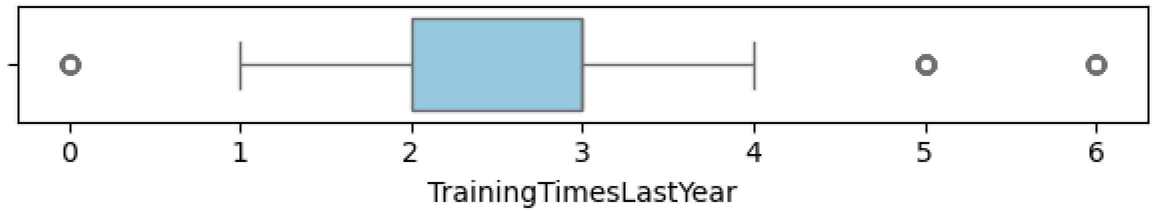



```
[OUTLIER] 'TrainingTimesLastYear' — 174 values detected
   Q1 = 2.00, Median = 3.00, Q3 = 3.00
   Lower bound = 0.50, Upper bound = 4.50
   Values:
  { 0: 43x, 5: 87x, 6: 44x }
```

Outlier Violin Plot - TrainingTimesLastYear

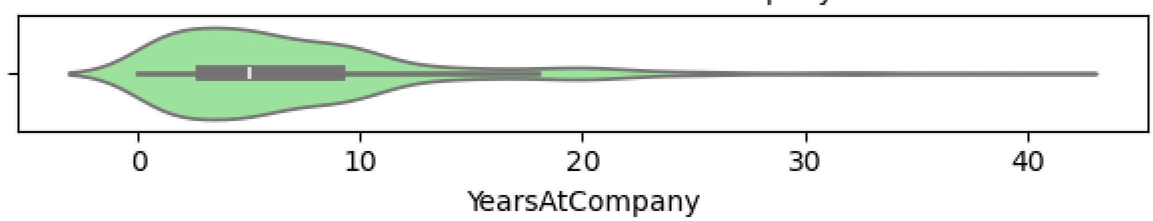


Outlier Boxplot - TrainingTimesLastYear

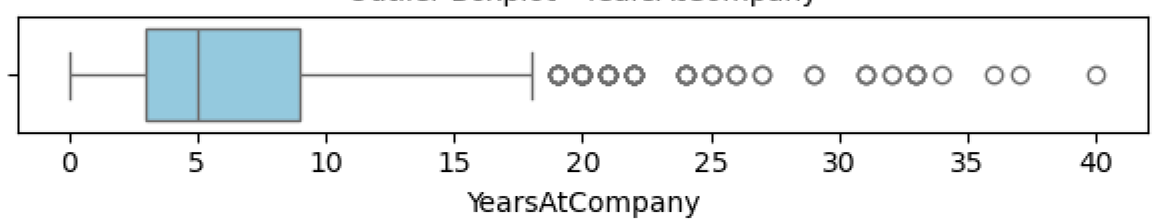



```
[OUTLIER] 'YearsAtCompany' — 79 values detected
   Q1 = 3.00, Median = 5.00, Q3 = 9.00
   Lower bound = -6.00, Upper bound = 18.00
   Values:
  { 19: 9x, 20: 20x, 21: 11x, 22: 9x, 24: 6x, 25: 4x, 26: 3x, 27: 2x, 29: 2x, 31:
  3x, 32: 2x, 33: 4x, 34: 1x, 36: 1x, 37: 1x, 40: 1x }
```

Outlier Violin Plot - YearsAtCompany

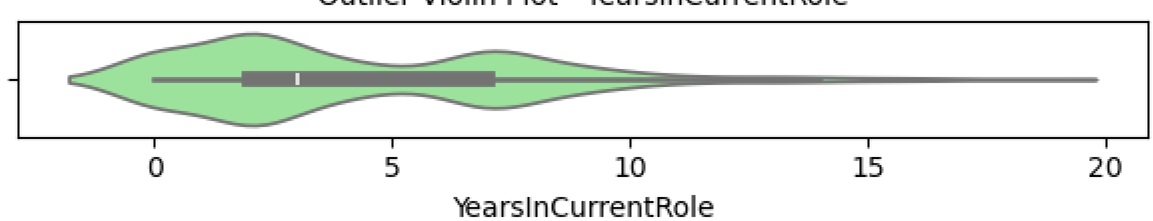


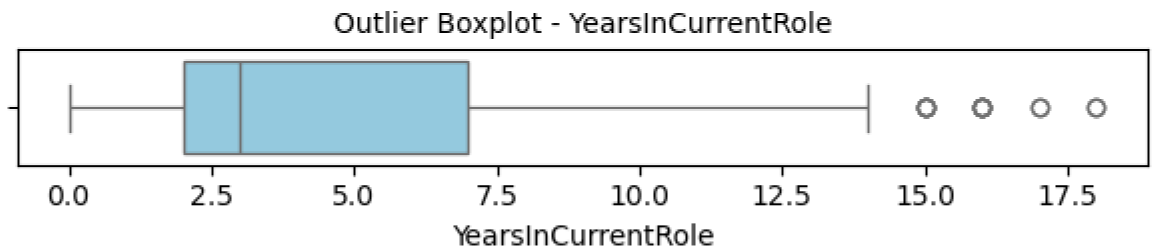
Outlier Boxplot - YearsAtCompany



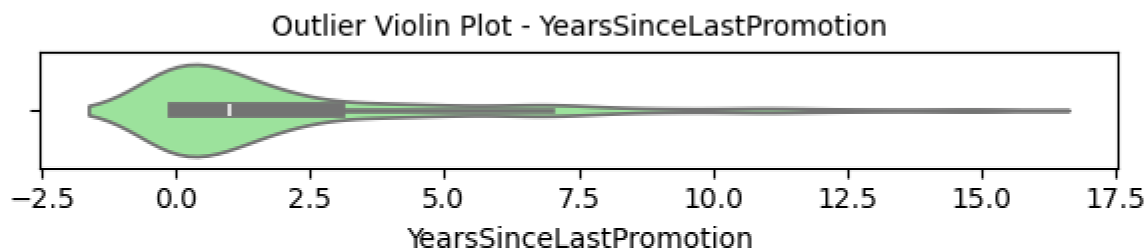

```
[OUTLIER] 'YearsInCurrentRole' — 15 values detected
   Q1 = 2.00, Median = 3.00, Q3 = 7.00
   Lower bound = -5.50, Upper bound = 14.50
   Values:
  { 15: 5x, 16: 6x, 17: 2x, 18: 2x }
```

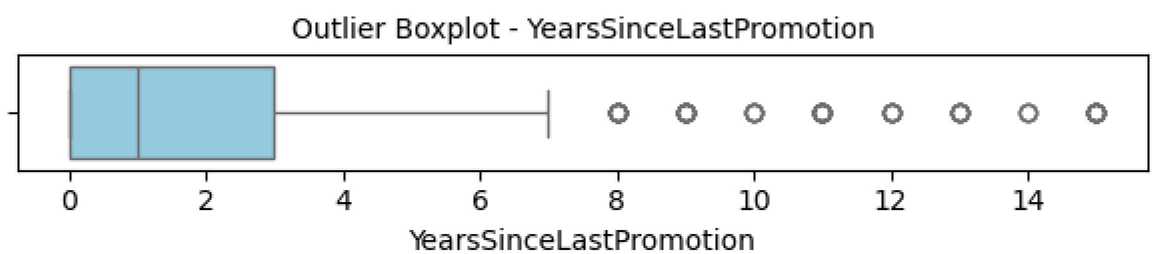
Outlier Violin Plot - YearsInCurrentRole



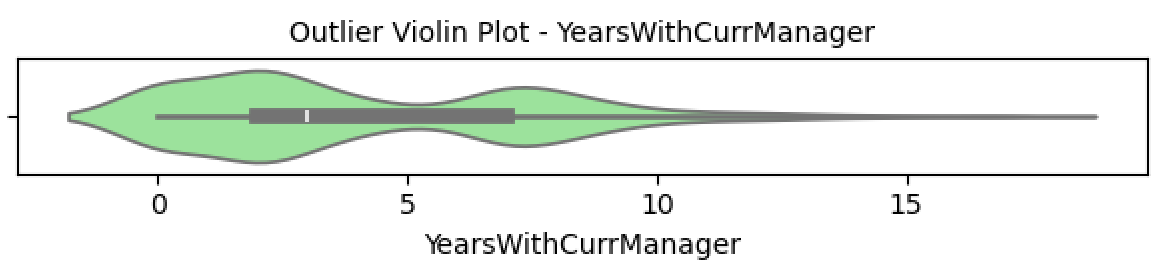


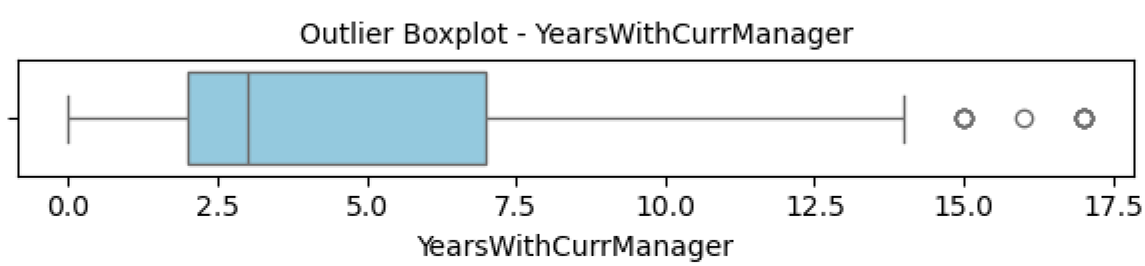

```
[OUTLIER] 'YearsSinceLastPromotion' — 80 values detected
   Q1 = 0.00, Median = 1.00, Q3 = 3.00
   Lower bound = -4.50, Upper bound = 7.50
   Values:
  { 8: 11x, 9: 14x, 10: 5x, 11: 21x, 12: 6x, 13: 7x, 14: 3x, 15: 13x }
```






```
[OUTLIER] 'YearsWithCurrManager' — 12 values detected
   Q1 = 2.00, Median = 3.00, Q3 = 7.00
   Lower bound = -5.50, Upper bound = 14.50
   Values:
  { 15: 5x, 16: 1x, 17: 6x }
```





Creating a new dataframe for data plotting needs

In [13]:

```
employee_plot = employee_df_clean.copy()

# list of numeric columns
numeric_cols = employee_df_clean.select_dtypes(include=['number']).columns.tolist()
numeric_cols = pd.Index(numeric_cols)
print(numeric_cols)

# list of category columns
categorical_cols = employee_df_clean.select_dtypes(include=['object', 'category', '
categorical_cols.remove('EmployeeId')
categorical_cols = pd.Index(categorical_cols)
print(categorical_cols)

label_map = {
    0: 'Active',
    1: 'Resigned'
}

education_map = {
    "1": "1-Below College",
    "2": "2-College",
    "3": "3-Bachelor",
    "4": "4-Master",
    "5": "5-Doctor"
}

env_satisfaction_map = {
    "1": "1-Low",
    "2": "2-Medium",
    "3": "3-High",
    "4": "4-Very High"
}

job_involvement_map = {
    "1": "1-Low",
    "2": "2-Medium",
    "3": "3-High",
    "4": "4-Very High"
}

job_level_map = {
    "1": "1-Entry Level",
    "2": "2-Junior",
    "3": "3-Mid Level",
    "4": "4-Senior",
    "5": "5-Executive/Top Level"
}

job_satisfaction_map = {
    "1": "1-Low",
    "2": "2-Medium",
    "3": "3-High",
    "4": "4-Very High"
}
```

```
performance_rating_map = {
    "1": "1-Low",
    "2": "2-Good",
    "3": "3-Excellent",
    "4": "4-Outstanding"
}

relationship_satisfaction_map = {
    "1": "1-Low",
    "2": "2-Medium",
    "3": "3-High",
    "4": "4-Very High"
}

stock_option_level_map = {
    "0": "0-None",
    "1": "1-Low",
    "2": "2-Medium",
    "3": "3-High"
}

work_life_balance_map = {
    "1": "1-Low",
    "2": "2-Good",
    "3": "3-Excellent",
    "4": "4-Outstanding"
}

employee_plot["Education"] = employee_plot["Education"].astype(str).map(education_m
employee_plot["EnvironmentSatisfaction"] = employee_plot["EnvironmentSatisfaction"]
employee_plot["JobLevel"] = employee_plot["JobLevel"].astype(str).map(job_level_map
employee_plot["JobInvolvement"] = employee_plot["JobInvolvement"].astype(str).map(j
employee_plot["JobSatisfaction"] = employee_plot["JobSatisfaction"].astype(str).map
employee_plot["PerformanceRating"] = employee_plot["PerformanceRating"].astype(str)
employee_plot["RelationshipSatisfaction"] = employee_plot["RelationshipSatisfaction
employee_plot["StockOptionLevel"] = employee_plot["StockOptionLevel"].astype(str).m
employee_plot["WorkLifeBalance"] = employee_plot["WorkLifeBalance"].astype(str).map

employee_plot.info()
```

```
Index(['Age', 'Attrition', 'DailyRate', 'DistanceFromHome', 'HourlyRate',
       'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
       'PercentSalaryHike', 'TotalWorkingYears', 'TrainingTimesLastYear',
       'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
       'YearsWithCurrManager'],
      dtype='object')
Index(['BusinessTravel', 'Department', 'Education', 'EducationField',
       'EnvironmentSatisfaction', 'Gender', 'JobInvolvement', 'JobLevel',
       'JobRole', 'JobSatisfaction', 'MaritalStatus', 'OverTime',
       'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel',
       'WorkLifeBalance'],
      dtype='object')
<class 'pandas.core.frame.DataFrame'>
Index: 1058 entries, 1 to 1469
Data columns (total 32 columns):
 #   Column                    Non-Null Count  Dtype   
---  ------                    --------------  -----   
 0   EmployeeId                1058 non-null   object  
 1   Age                       1058 non-null   int64   
 2   Attrition                 1058 non-null   int32   
 3   BusinessTravel            1058 non-null   category
 4   DailyRate                 1058 non-null   int64   
 5   Department                1058 non-null   category
 6   DistanceFromHome          1058 non-null   int64   
 7   Education                 1058 non-null   object  
 8   EducationField            1058 non-null   category
 9   EnvironmentSatisfaction   1058 non-null   object  
 10  Gender                    1058 non-null   category
 11  HourlyRate                1058 non-null   int64   
 12  JobInvolvement            1058 non-null   object  
 13  JobLevel                  1058 non-null   object  
 14  JobRole                   1058 non-null   category
 15  JobSatisfaction           1058 non-null   object  
 16  MaritalStatus             1058 non-null   category
 17  MonthlyIncome             1058 non-null   int64   
 18  MonthlyRate               1058 non-null   int64   
 19  NumCompaniesWorked        1058 non-null   int64   
 20  OverTime                  1058 non-null   category
 21  PercentSalaryHike         1058 non-null   int64   
 22  PerformanceRating         1058 non-null   object  
 23  RelationshipSatisfaction  1058 non-null   object  
 24  StockOptionLevel          1058 non-null   object  
 25  TotalWorkingYears         1058 non-null   int64   
 26  TrainingTimesLastYear     1058 non-null   int64   
 27  WorkLifeBalance           1058 non-null   object  
 28  YearsAtCompany            1058 non-null   int64   
 29  YearsInCurrentRole        1058 non-null   int64   
 30  YearsSinceLastPromotion   1058 non-null   int64   
 31  YearsWithCurrManager      1058 non-null   int64   
dtypes: category(7), int32(1), int64(14), object(10)
memory usage: 251.5+ KB
```

Descriptive Statistics "employee_df"

In [14]:
```python
# Descriptive statistics
print("\nDescriptive statistics of numeric category columns:")
display(employee_df[numeric_cols].describe())

# Descriptive statistics for the category column
print("\nDescriptive statistics for the category column:")
display(employee_df[categorical_cols].describe(include='all'))
```

```
Descriptive statistics of numeric category columns:
```

	Age	Attrition	DailyRate	DistanceFromHome	HourlyRate	MonthlyInco
count	1470.000000	1058.000000	1470.000000	1470.000000	1470.000000	1470.000(
mean	36.923810	0.169187	802.485714	9.192517	65.891156	6502.931
std	9.135373	0.375094	403.509100	8.106864	20.329428	4707.956
min	18.000000	0.000000	102.000000	1.000000	30.000000	1009.000(
25%	30.000000	0.000000	465.000000	2.000000	48.000000	2911.000(
50%	36.000000	0.000000	802.000000	7.000000	66.000000	4919.000(
75%	43.000000	0.000000	1157.000000	14.000000	83.750000	8379.000(
max	60.000000	1.000000	1499.000000	29.000000	100.000000	19999.000(

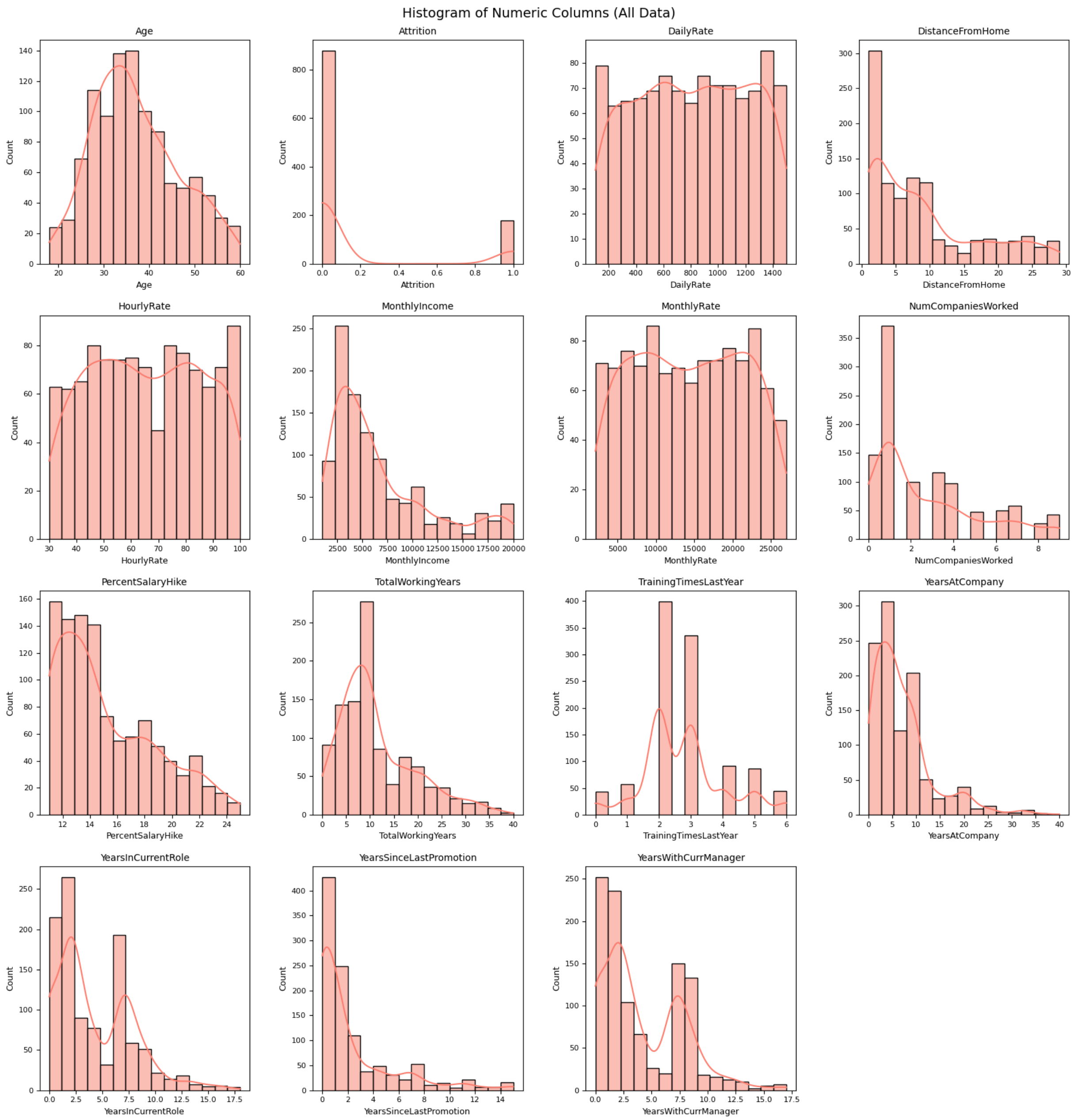
```
Descriptive statistics for the category column:
```

	BusinessTravel	Department	Education	EducationField	EnvironmentSatisfaction	Ge
count	1470	1470	1470	1470	1470	
unique	3	3	5	6	4	
top	Travel_Rarely	Research & Development	3	Life Sciences	3	
freq	1043	961	572	606	453	

Plot data

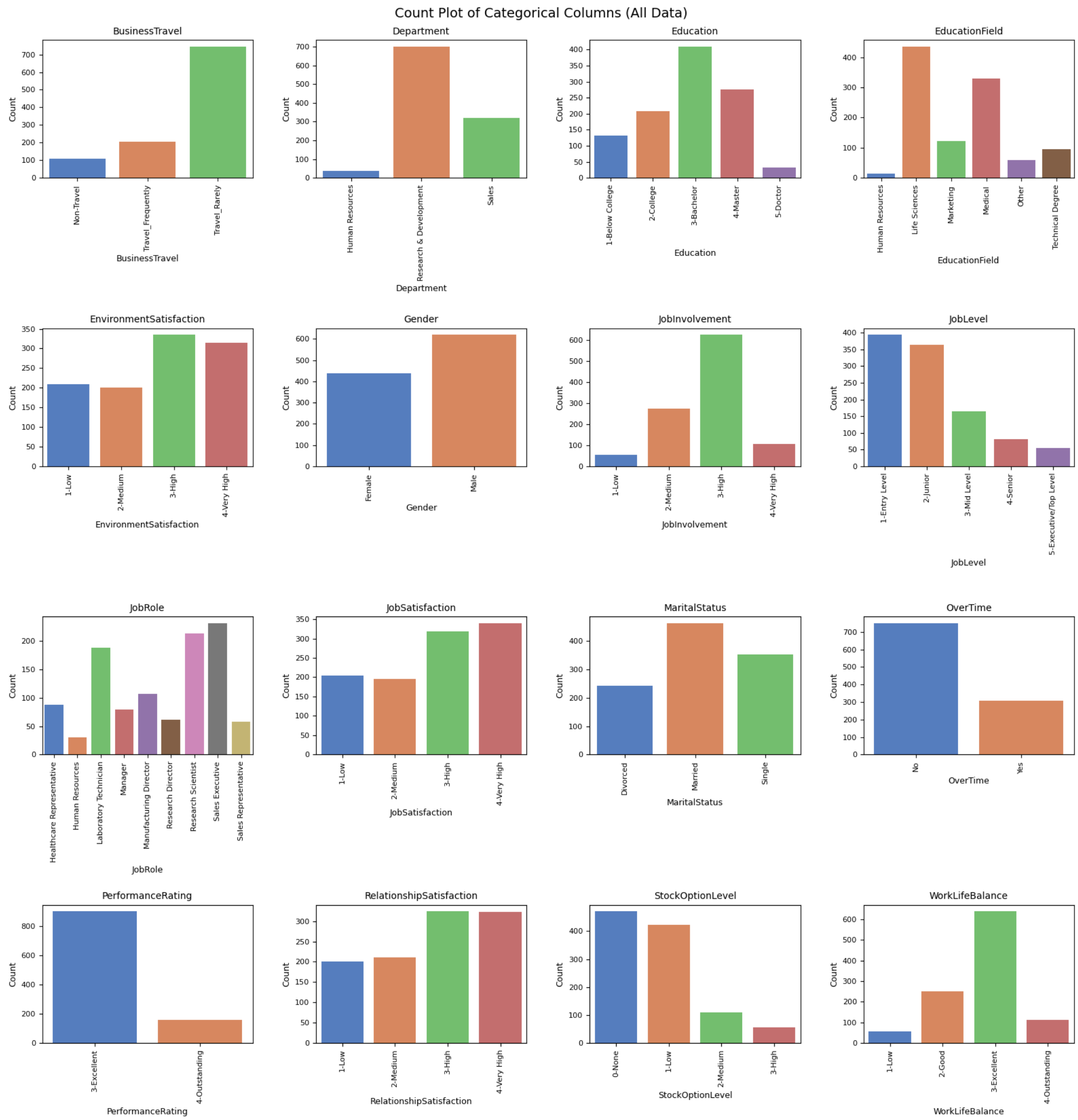
Hitogram Plot

In [15]:
```python
from scripts.runPlot_ObsHistNumeric import plot_obs_histnums
plot_obs_histnums(df=employee_plot, numeric_cols=numeric_cols, n_cols=4, color='sal
```



In [16]:

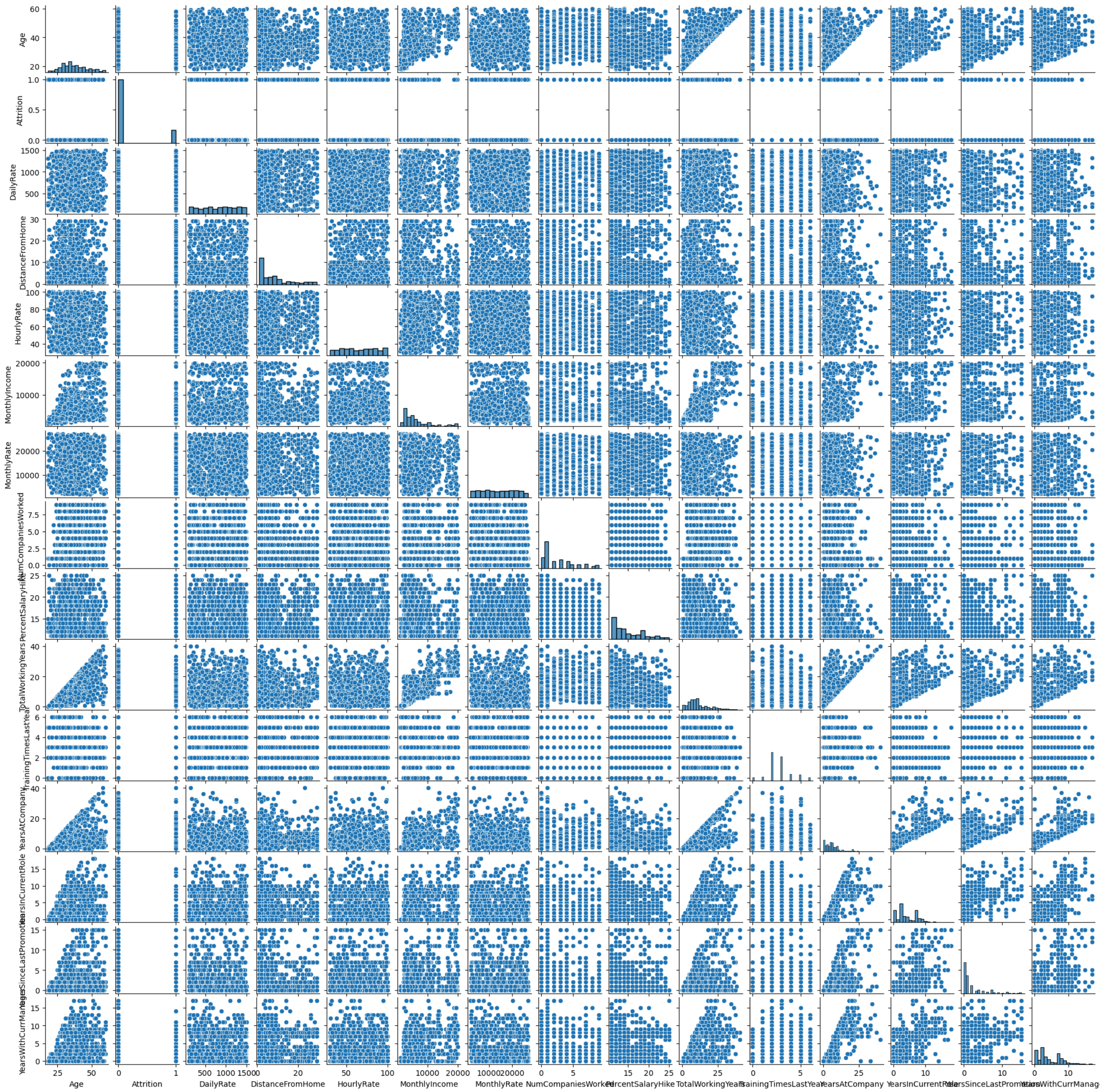
```
from scripts.runPlot_ObsHistCatgs import plot_obs_histcatgs
plot_obs_histcatgs(df=employee_plot, categorical_cols=categorical_cols, n_cols=4)
```



Pair plot

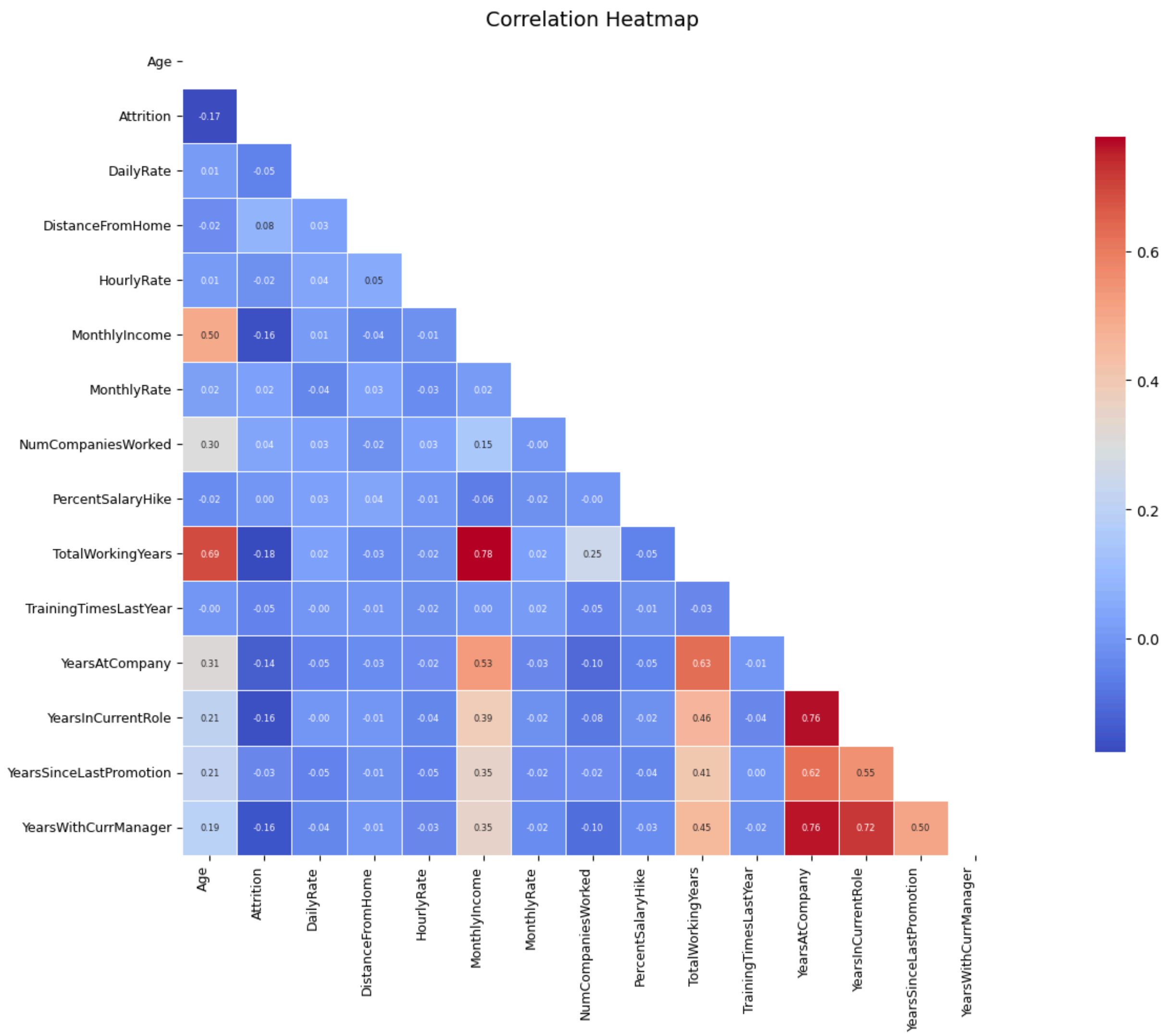
In [17]:

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Creating a scatter matrix (pair plot)
g = sns.pairplot(employee_plot, vars=numeric_cols)
# Ganti ukuran figure secara manual
g.fig.set_size_inches(20, 20)  # (lebar, tinggi)
plt.show()
```



Correlation Headmap

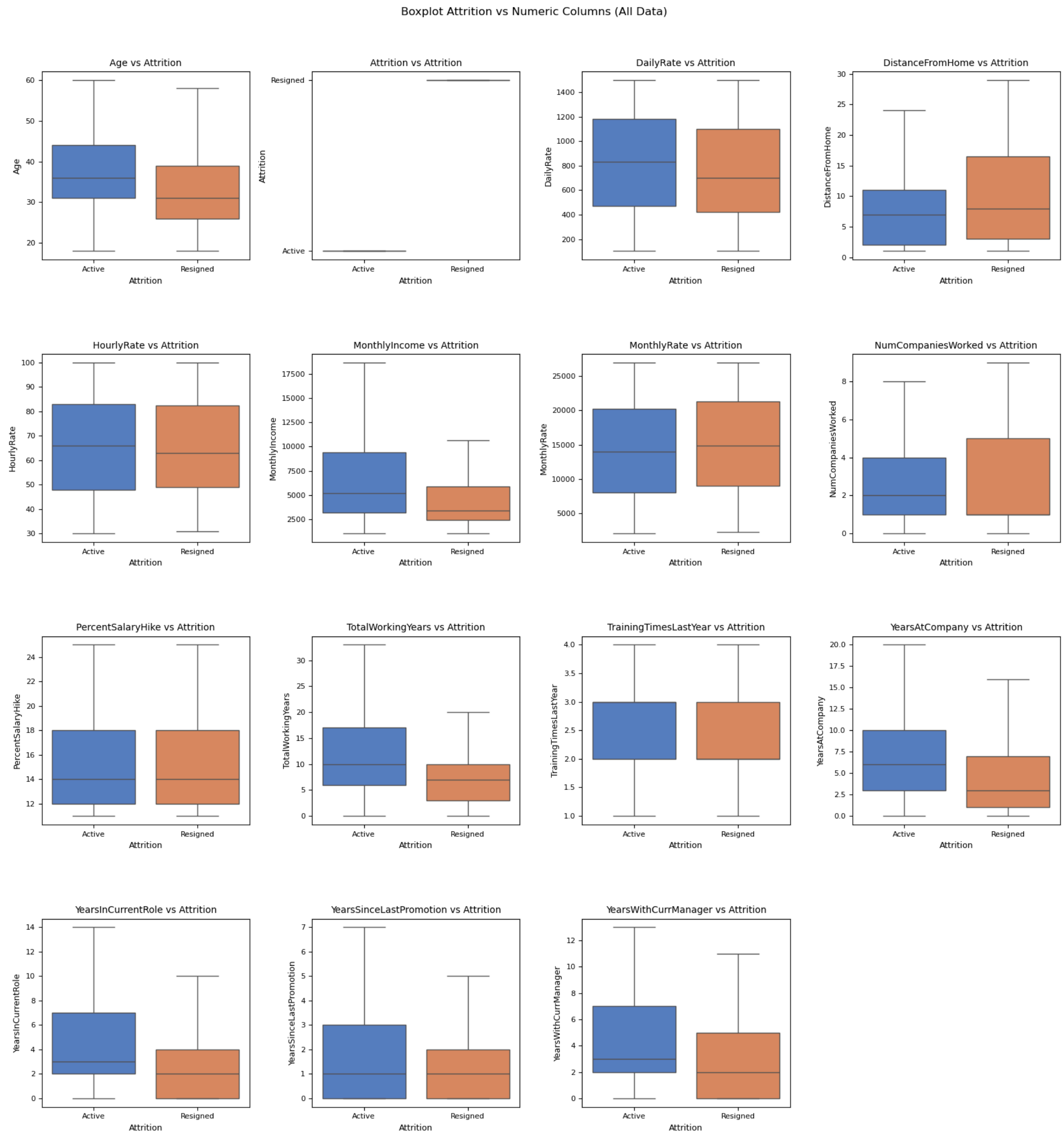
In [18]:
```
from scripts.runPlot_ObsCorrHeatmap import plot_obs_corrheatmap
plot_obs_corrheatmap(employee_plot, columns=numeric_cols,figsize=(18, 10))
```



Box Plot

In [19]:

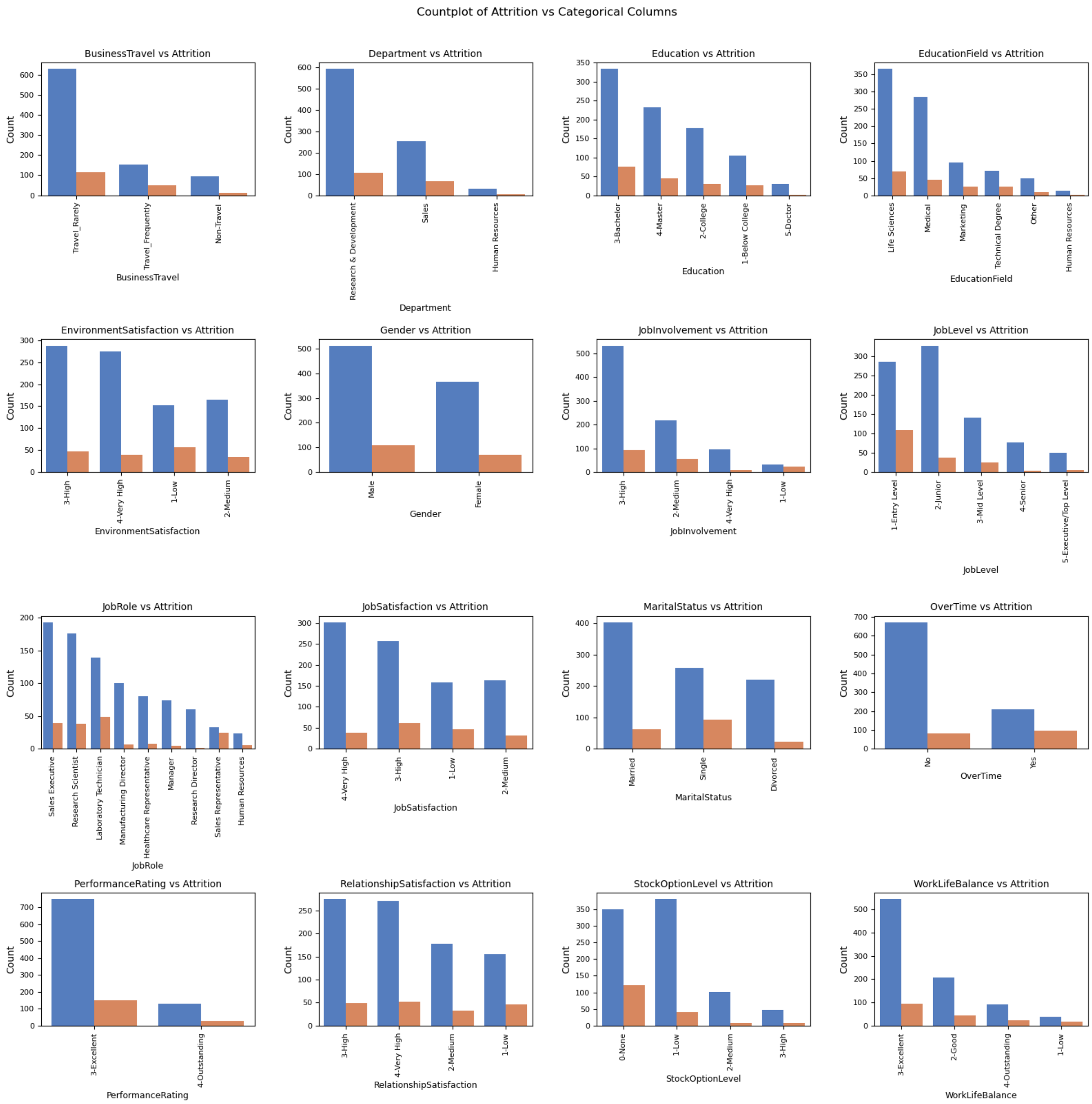
```
from scripts.runPlot_ObsBoxCols import plot_obs_boxcols
plot_obs_boxcols(employee_plot, numeric_cols=numeric_cols, col_target='Attrition',
```



Count Plot

In [20]:

```
from scripts.runPlot_ObsCountCols import plot_obs_countcols
plot_obs_countcols(df=employee_plot, categorical_cols=categorical_cols, col_target=
```



Line plot

In [21]:

```
from scripts.runPlot_ObsLineXYcMetricSubplot import plot_obs_lineXYc_metric_subplot
from scripts.runPlot_ObsHitsXYcMetricSubplot import plot_obs_histXYc_metric_subplot



XcolsList = [col for col in categorical_cols if col != 'Attrition'] + [col for col
XcolsList.sort()
print("columns List to plot:")
print(XcolsList)
```

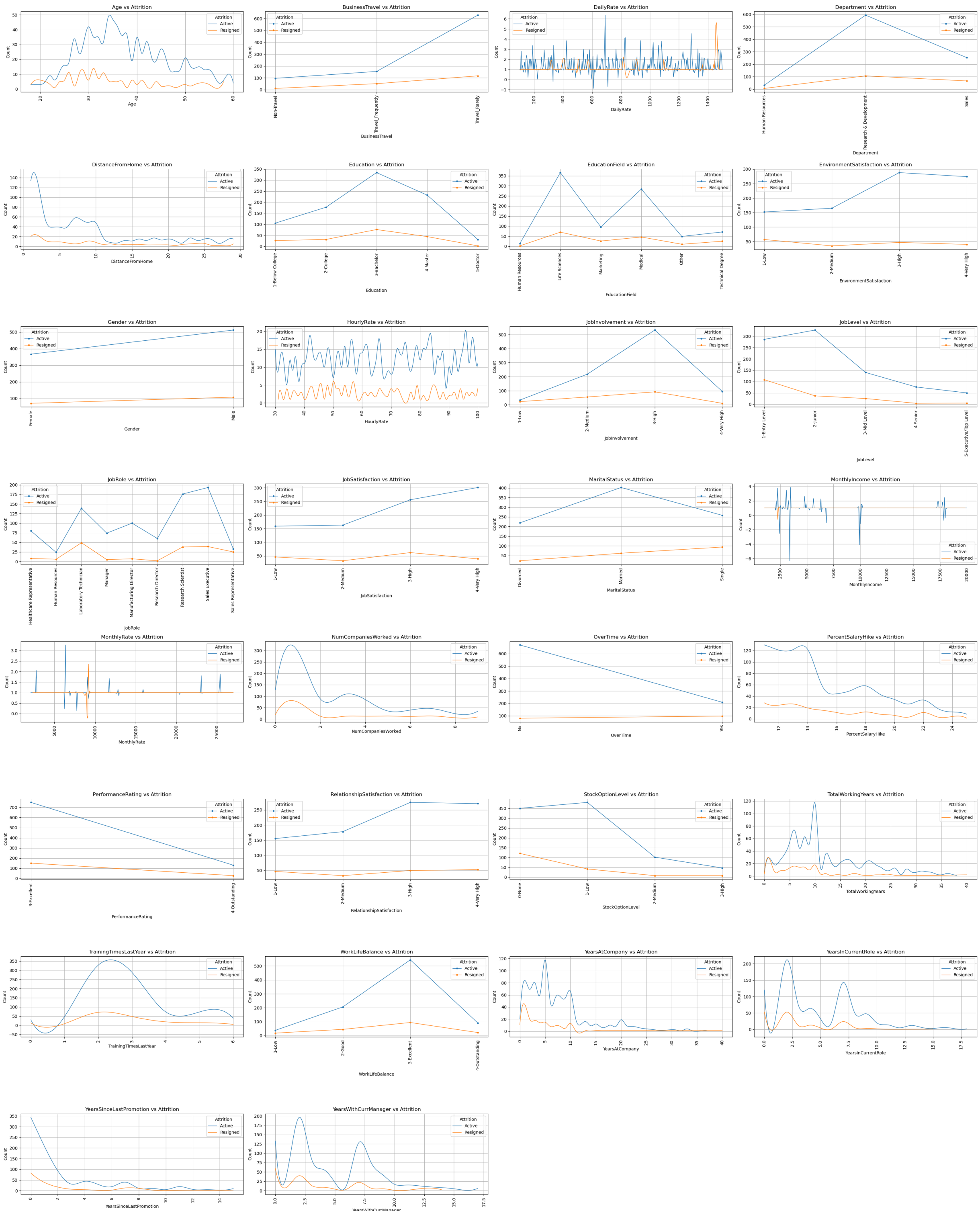
```
columns List to plot:
['Age', 'BusinessTravel', 'DailyRate', 'Department', 'DistanceFromHome', 'Educatio
n', 'EducationField', 'EnvironmentSatisfaction', 'Gender', 'HourlyRate', 'JobInvolve
ment', 'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus', 'MonthlyIncome',
'MonthlyRate', 'NumCompaniesWorked', 'OverTime', 'PercentSalaryHike', 'PerformanceRa
ting', 'RelationshipSatisfaction', 'StockOptionLevel', 'TotalWorkingYears', 'Trainin
gTimesLastYear', 'WorkLifeBalance', 'YearsAtCompany', 'YearsInCurrentRole', 'YearsSi
nceLastPromotion', 'YearsWithCurrManager']
```

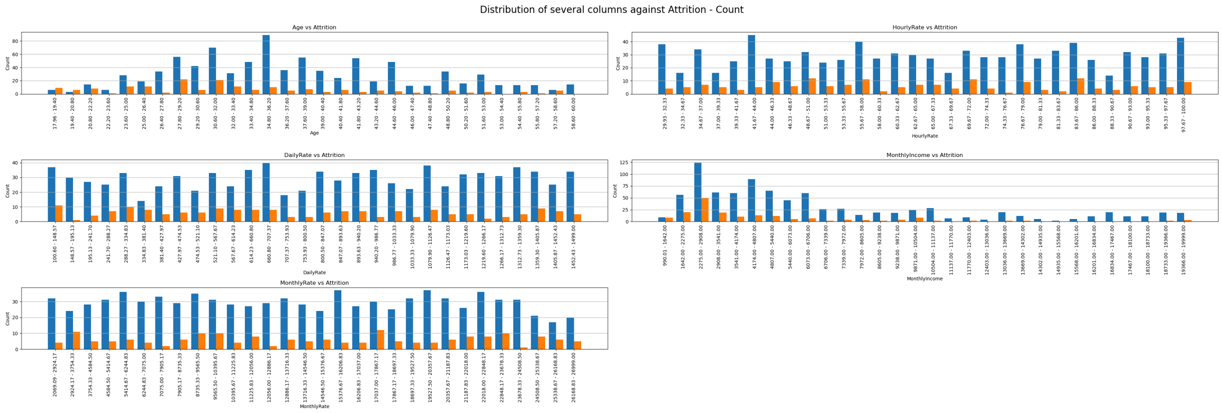
In [22]:

```python
YCol = "Attrition"
Xline_smooth =  True
rotation_label = 90
metric_types = ["count","nimax","rank","catg_contrib"]
continue_cols=["Age", "HourlyRate","DailyRate", "MonthlyIncome","MonthlyRate"]
metric_types_map = {
    "count": "Count Plot",
    "nimax": "niMax Normalization Plot",
    "rank": "Rangking Metric Plot",
    "catg_contrib": "Percentage Contribution Metric Plot"
}


for metric_type in metric_types:
    print(f"------------ Metric Plot: {metric_types_map[metric_type]} ------------"

    plot_obs_lineXYc_metric_subplot(
        employee_plot,
        Xcols=XcolsList,
        lineCol=YCol,
        n_cols=4,
        ignore_cols=None,
        label_map=label_map,
        line_smooth=Xline_smooth,
        rotation_label=rotation_label,
        height_per_plot=5,
        legend_show=True,
        metric_type=metric_type)

    plot_obs_histXYc_metric_subplot(
        employee_plot,
        Xcols=continue_cols,
        barCol=YCol,
        metric_type=metric_type,
        label_map=label_map,
        n_cols=2,
        width_per_plot=18,
        height_per_plot=4,
        rotation_label=rotation_label,
        bins=30,
        legend_show=False)
```

```
------------ Metric Plot: Count Plot ------------
```

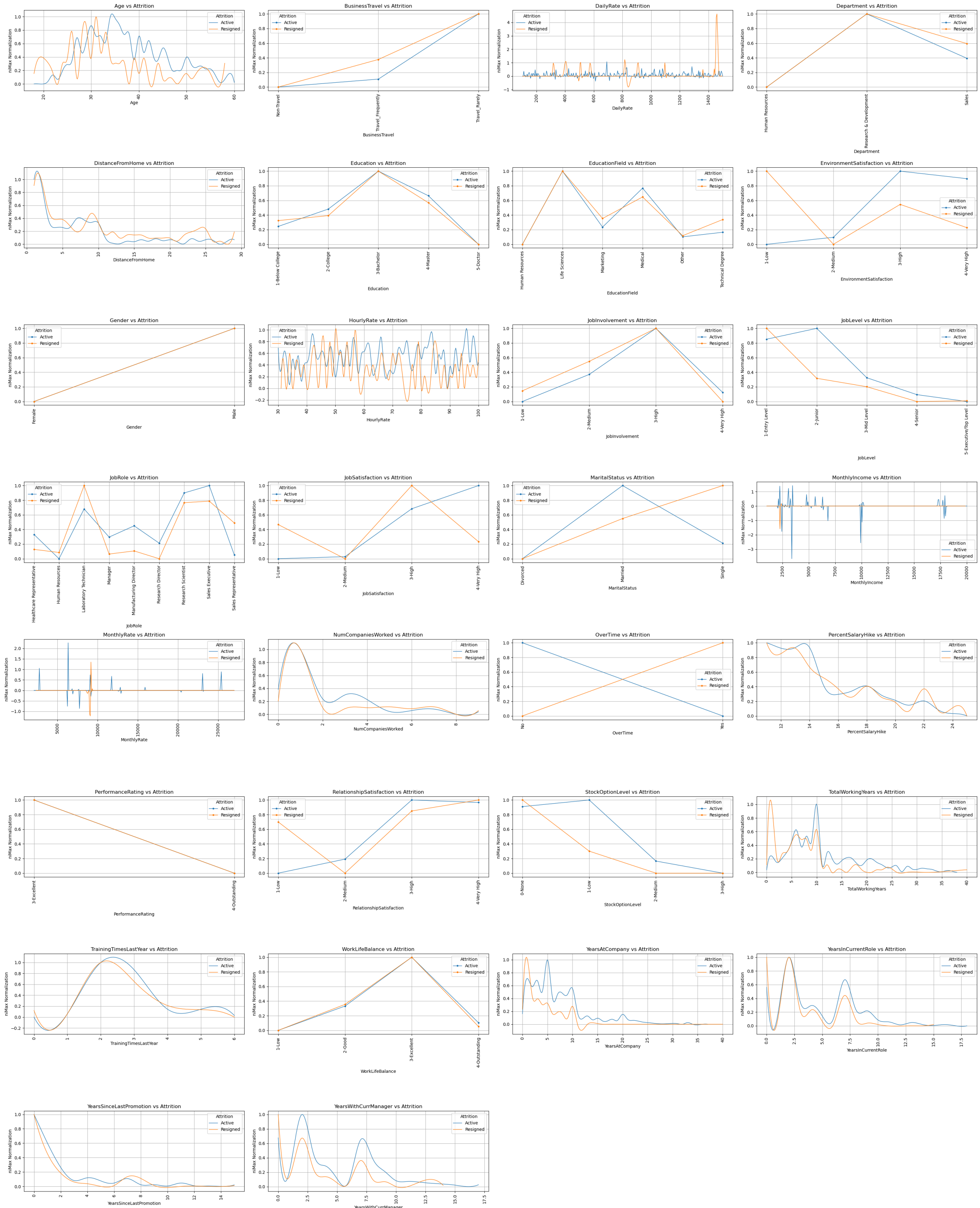
Distribution of several columns against Attrition - Count



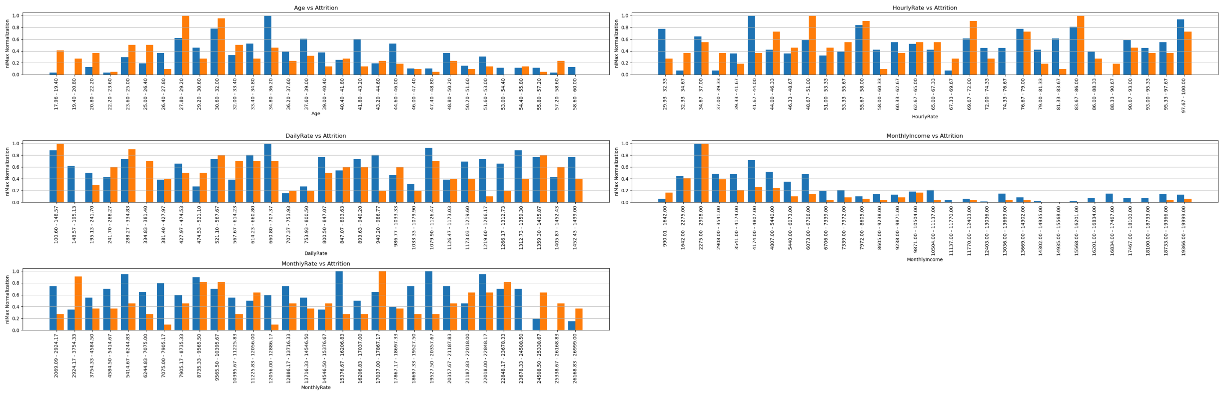


------------ Metric Plot: niMax Normalization Plot ------------

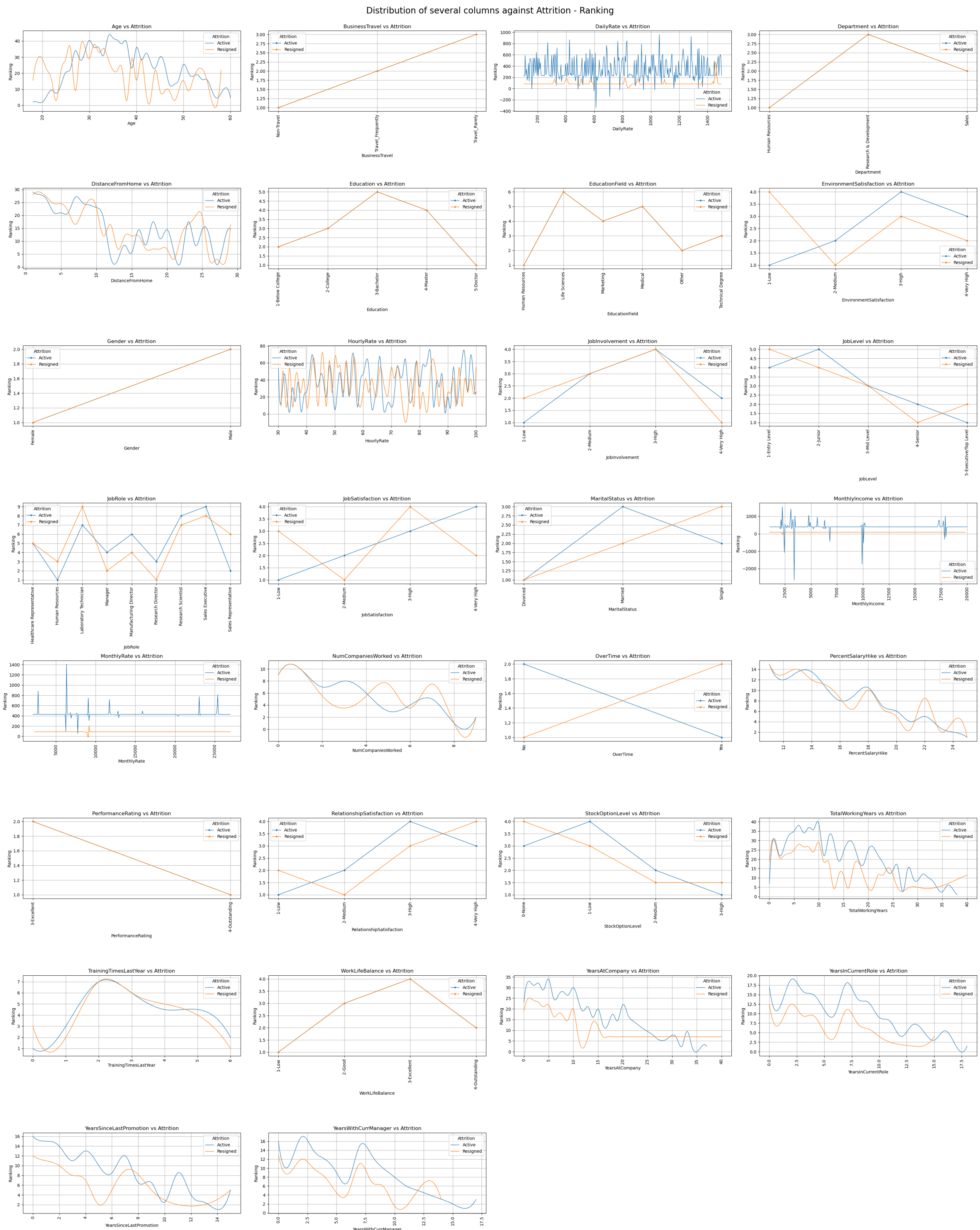
Distribution of several columns against Attrition - niMax Normalization



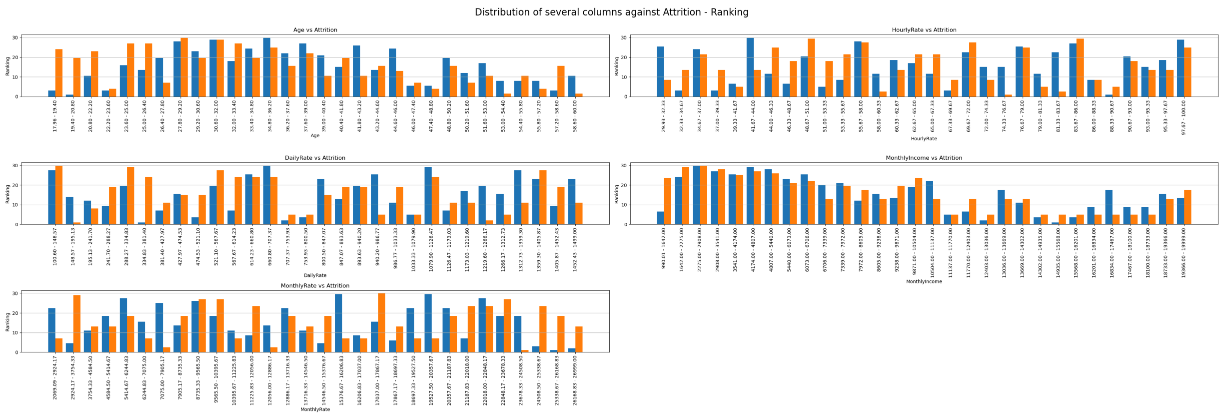
Distribution of several columns against Attrition - niMax Normalization



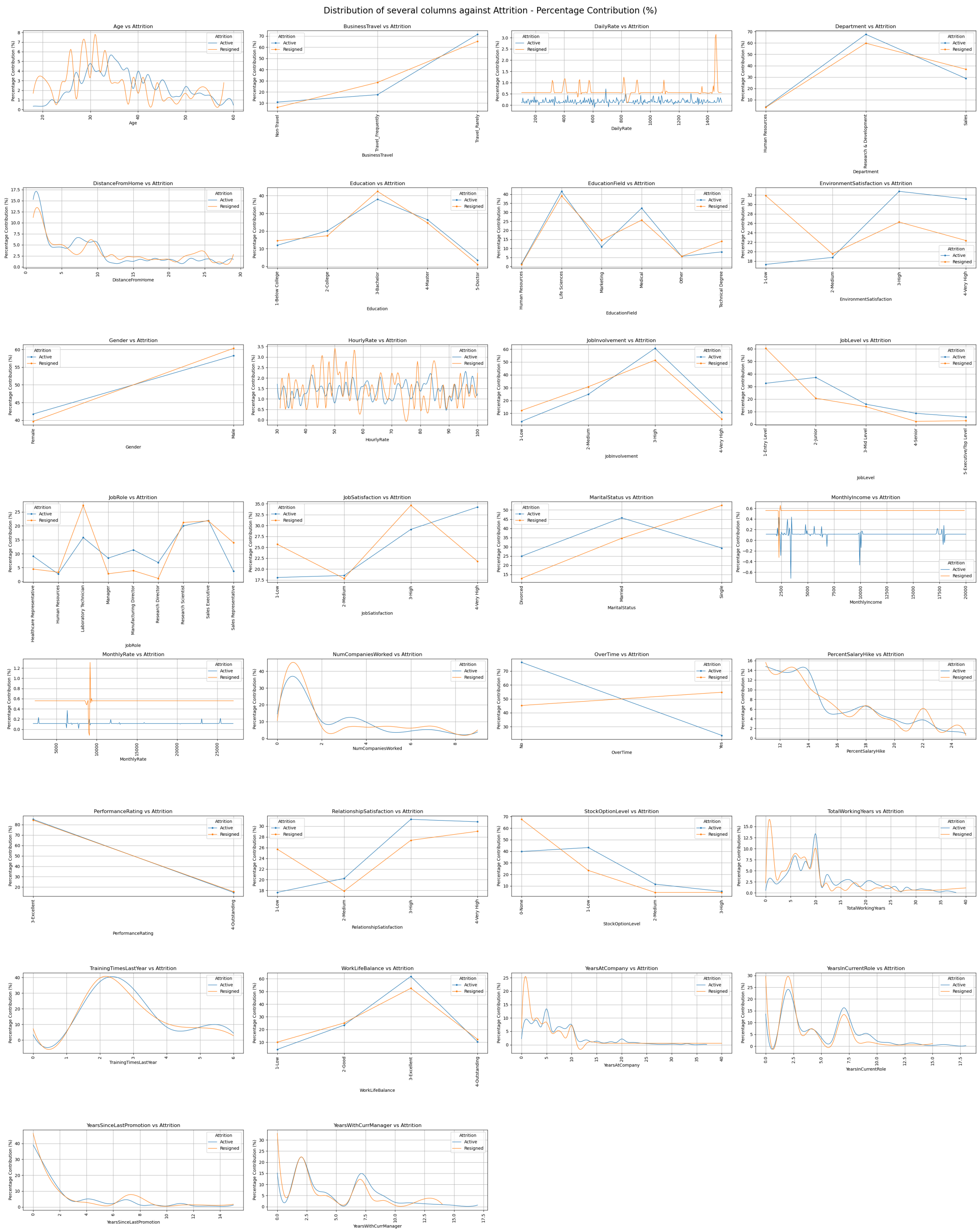
------------ Metric Plot: Rangking Metric Plot ------------

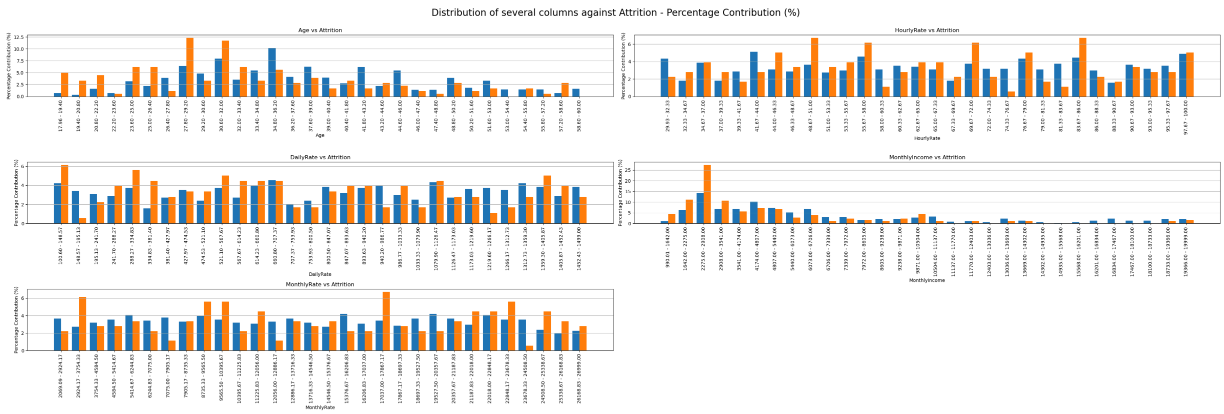


Distribution of several columns against Attrition - Ranking
Age vs Attrition
BusinessTravel vs Attrition
DailyRate vs Attrition
Department vs Attrition
DistanceFromHome vs Attrition
Education vs Attrition
EducationField vs Attrition
EnvironmentSatisfaction vs Attrition
Gender vs Attrition
HourlyRate vs Attrition
JobInvolvement vs Attrition
JobLevel vs Attrition
JobRole vs Attrition
JobSatisfaction vs Attrition
MaritalStatus vs Attrition
MonthlyIncome vs Attrition
MonthlyRate vs Attrition
NumCompaniesWorked vs Attrition
OverTime vs Attrition
PercentSalaryHike vs Attrition
PerformanceRating vs Attrition
RelationshipSatisfaction vs Attrition
StockOptionLevel vs Attrition
TotalWorkingYears vs Attrition
TrainingTimesLastYear vs Attrition
WorkLifeBalance vs Attrition
YearsAtCompany vs Attrition
YearsInCurrentRole vs Attrition
YearsSinceLastPromotion vs Attrition
YearsWithCurrManager vs Attrition
Attrition
Active
Resigned
Ranking



------------ Metric Plot: Percentage Contribution Metric Plot ------------


Distribution of several columns against Attrition - Percentage Contribution (%)
Age vs Attrition
BusinessTravel vs Attrition
DailyRate vs Attrition
Department vs Attrition
DistanceFromHome vs Attrition
Education vs Attrition
EducationField vs Attrition
EnvironmentSatisfaction vs Attrition
Gender vs Attrition
HourlyRate vs Attrition
JobInvolvement vs Attrition
JobLevel vs Attrition
JobRole vs Attrition
JobSatisfaction vs Attrition
MaritalStatus vs Attrition
MonthlyIncome vs Attrition
MonthlyRate vs Attrition
NumCompaniesWorked vs Attrition
OverTime vs Attrition
PercentSalaryHike vs Attrition
PerformanceRating vs Attrition
RelationshipSatisfaction vs Attrition
StockOptionLevel vs Attrition
TotalWorkingYears vs Attrition
TrainingTimesLastYear vs Attrition
WorkLifeBalance vs Attrition
YearsAtCompany vs Attrition
YearsInCurrentRole vs Attrition
YearsSinceLastPromotion vs Attrition
YearsWithCurrManager vs Attrition
Attrition
Active
Resigned
Percentage Contribution (%)



Saving the dataframe to a dataset file

In [ ]:
```
employee_df_clean.to_parquet("saved/employee_cleaned.parquet", index=False)
employee_plot.to_csv("saved/employee_plot.csv", index=False)
```

Modeling

Encoding

In [ ]:
```
from sklearn.preprocessing import LabelEncoder

print(categorical_cols)

employee_df_encode = employee_df_clean.copy()

le = LabelEncoder()
for col in categorical_cols:
    employee_df_encode[col] = le.fit_transform(employee_df_encode[col])
```

```
Index(['BusinessTravel', 'Department', 'Education', 'EducationField',
       'EnvironmentSatisfaction', 'Gender', 'JobInvolvement', 'JobLevel',
       'JobRole', 'JobSatisfaction', 'MaritalStatus', 'OverTime',
       'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel',
       'WorkLifeBalance'],
      dtype='object')
```

Split feature data and target data

In [ ]:
```
X = employee_df_encode.drop(columns=['EmployeeId','Attrition'])
y = employee_df_encode['Attrition']
```

Building a Model

import def function to support the process.

In [ ]:
```
from scripts.runModelSuprt import *

resultRunModel_df = newDf_logModel()
models_path = "__models__/"
```

Running the Model

Logistic Regression

In [ ]:
```
# import script runModel
from scripts.runModel_LogRegression import *

# running several model trainings
#------------------
setModelRun = {
    "IdRun": "LR01",
    "options": {"scheme":20},
    "Params": {"modelParams":{"random_state": 24}}
}

# Simulation model, runModel_LogRegression
setModel_Output, model_output = runModel_LogRegression(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Logistic Regression")


#------------------
setModelRun = {
    "IdRun": "LR02",
    "options": {"scheme":30},
    "Params": {"modelParams":{"random_state": 24}}
}

# Simulation model, runModel_LogRegression
setModel_Output, model_output = runModel_LogRegression(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Logistic Regression")


#------------------
setModelRun = {
    "IdRun": "LR03",
    "options": {"scheme":30,"gridSCV": True},
    "Params": {"modelParams":{"random_state": 24}}
}

# Simulation model, runModel_LogRegression
setModel_Output, model_output = runModel_LogRegression(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Logistic Regression")


#------------------
setModelRun = {
```

```
    "IdRun": "LR04",
    "options": {"scheme":30,"randSCV": True},
    "Params": {"modelParams":{"random_state": 24}}
}

# Simulation model, runModel_LogRegression
setModel_Output, model_output = runModel_LogRegression(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Logistic Regression")
```

	ModelName	IdRun	DateAction	options	Params	Remarks	AccuracyTrair
0	Logistic Regression	LR01	2025-05-08 03:01:02	{'scheme': 20}	{'modelParams': {'random_state': 24}}	NaN	0.836879
1	Logistic Regression	LR02	2025-05-08 03:01:02	{'scheme': 30}	{'modelParams': {'random_state': 24}}	NaN	0.833784
2	Logistic Regression	LR03	2025-05-08 03:01:07	{'scheme': 30, 'gridSCV': True}	{'modelParams': {'random_state': 24, 'n_jobs': 2}, 'BestParamsGridSCV': {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}}	NaN	0.862162
3	Logistic Regression	LR04	2025-05-08 03:01:10	{'scheme': 30, 'randSCV': True}	{'modelParams': {'max_iter': 500, 'random_state': 24}, 'BestParamsRandSCV': {'C': 0.13292918943162169, 'penalty': 'l1', 'solver': 'liblinear'}}	NaN	0.855405

Random Forest

```
In [ ]:
# import script runModel
from scripts.runModel_RandomForest import *

# Running multiple models
#------------------

setModelRun = {
    "IdRun": "RF01",
    "options": {"scheme":20},
    "Params": {"modelParams":{"random_state": 24}}
}

# Model simulation, runModel_RandomForest
setModel_Output, model_output = runModel_RandomForest(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
```

```
showResultDfModel(resultRunModel_df, ModelName = "Random Forest")


#------------------
setModelRun = {
    "IdRun": "RF02",
    "options": {"scheme":30},
    "Params": {"modelParams":{"random_state": 24}}
}

# Model simulation, runModel_RandomForest
setModel_Output, model_output = runModel_RandomForest(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Random Forest")


#------------------
setModelRun = {
    "IdRun": "RF03",
    "options": {"scheme":30,"gridSCV": True},
}

# Model simulation, runModel_RandomForest
setModel_Output, model_output = runModel_RandomForest(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Random Forest")


#------------------
setModelRun = {
    "IdRun": "RF04",
    "options": {"scheme":30,"randSCV": True},
}

# Model simulation, runModel_RandomForest
setModel_Output, model_output = runModel_RandomForest(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Random Forest")
```

	ModelName	IdRun	DateAction	options	Params	Remarks	AccuracyTrain
4	Random Forest	RF01	2025-05-08 03:01:11	{'scheme': 20}	{'modelParams': {'n_jobs': 2, 'random_state': 24}}	NaN	1.000000
5	Random Forest	RF02	2025-05-08 03:01:12	{'scheme': 30}	{'modelParams': {'n_jobs': 2, 'random_state': 24}}	NaN	1.000000
6	Random Forest	RF03	2025-05-08 03:06:12	{'scheme': 30, 'gridSCV': True}	{'modelParams': {'random_state': 42, 'n_jobs': 2}, 'BestParamsGridSCV': {'max_depth': 20, 'max_features': 0.5, 'min_samples_leaf': 1, 'min_samples_split': 15, 'n_estimators': 500}}	NaN	0.935135
7	Random Forest	RF04	2025-05-08 03:06:18	{'scheme': 30, 'randSCV': True}	{'modelParams': {'random_state': 42, 'n_jobs': 2}, 'BestParamsRandSCV': {'n_estimators': 50, 'min_samples_split': 2, 'max_depth': 10}}	NaN	0.993243

Decision Tree

In [ ]:
```
# import script runModel
from scripts.runModel_DecisionTree import *

# Running several training models
#------------------
setModelRun = {
    "IdRun": "DF01",
    "options": {"scheme":20},
}

# Simulation model, runModel_DecisionTree
setModel_Output, model_output = runModel_DecisionTree(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "DecisionTree Classifier")


#------------------
setModelRun = {
    "IdRun": "DF02",
    "options": {"scheme":30},
}
```

```
# Simulation model, runModel_DecisionTree
setModel_Output, model_output = runModel_DecisionTree(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "DecisionTree Classifier")


#------------------
setModelRun = {
    "IdRun": "DF03",
    "options": {"scheme":30,"gridSCV": True},
}

# Simulation model, runModel_DecisionTree
setModel_Output, model_output = runModel_DecisionTree(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "DecisionTree Classifier")


#------------------
setModelRun = {
    "IdRun": "DF04",
    "options": {"scheme":30,"randSCV": True},
}

# Simulation model, runModel_DecisionTree
setModel_Output, model_output = runModel_DecisionTree(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "DecisionTree Classifier")
```

	ModelName	IdRun	DateAction	options	Params	Remarks	AccuracyTrain
0	DecisionTree Classifier	DF01	2025-05-08 03:06:18	{'scheme': 20}	{'modelParams': {'random_state': 42}}	NaN	1.000000
1	DecisionTree Classifier	DF02	2025-05-08 03:06:18	{'scheme': 30}	{'modelParams': {'random_state': 42}}	NaN	1.000000
2	DecisionTree Classifier	DF03	2025-05-08 03:06:23	{'scheme': 30, 'gridSCV': True}	{'modelParams': {'random_state': 42}, 'BestParamsGridSCV': {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2}}	NaN	0.964865
3	DecisionTree Classifier	DF04	2025-05-08 03:06:23	{'scheme': 30, 'randSCV': True}	{'modelParams': {'random_state': 42}, 'BestParamsRandSCV': {'criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 8, 'min_samples_split': 13}}	NaN	0.836486

AdaBoost (Adaptive Boosting)

In [ ]:
```
from scripts.runModel_Adaboost import *


# Running several training models
#------------------
setModelRun = {
    "IdRun": "AD01",
    "options": {"scheme":20},
}

# Model simulation, runModel_Adaboost
setModel_Output, model_output = runModel_Adaboost(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Adaboost Classifier")


#------------------
setModelRun = {
    "IdRun": "AD02",
    "options": {"scheme":30},
}

# Model simulation, runModel_Adaboost
setModel_Output, model_output = runModel_Adaboost(setModelRun, X, y)
```

```
# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Adaboost Classifier")


#------------------
setModelRun = {
    "IdRun": "AD03",
    "options": {"scheme":30,"gridSCV": True},
}

# Model simulation, runModel_Adaboost
setModel_Output, model_output = runModel_Adaboost(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Adaboost Classifier")


#------------------
setModelRun = {
    "IdRun": "AD04",
    "options": {"scheme":30,"randSCV": True},
}

# Model simulation, runModel_Adaboost
setModel_Output, model_output = runModel_Adaboost(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Adaboost Classifier")
```

	ModelName	IdRun	DateAction	options	Params	Remarks	AccuracyTrain
0	Adaboost Classifier	AD01	2025-05-08 03:06:24	{'scheme': 20}	{'modelParams': {'random_state': 42}}	NaN	0.875887
1	Adaboost Classifier	AD02	2025-05-08 03:06:24	{'scheme': 30}	{'modelParams': {'random_state': 42}}	NaN	0.870270
2	Adaboost Classifier	AD03	2025-05-08 03:06:39	{'scheme': 30, 'gridSCV': True}	{'modelParams': {'random_state': 42}, 'BestParamsGridSCV': {'learning_rate': 1.0, 'n_estimators': 50}}	NaN	0.870270
3	Adaboost Classifier	AD04	2025-05-08 03:06:50	{'scheme': 30, 'randSCV': True}	{'modelParams': {'random_state': 42}, 'BestParamsRandSCV': {'learning_rate': 0.7319987722668247, 'n_estimators': 79}}	NaN	0.871622

Gradient Boosting

In [ ]:
```
from scripts.runModel_GradientBoosting import *

# Running several training models
#------------------
setModelRun = {
    "IdRun": "GB01",
    "options": {"scheme":20},
}

# Model simulation, runModel_GradientBoosting
setModel_Output, model_output = runModel_GradientBoosting(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Gradient Boosting")


#------------------
setModelRun = {
    "IdRun": "GB02",
    "options": {"scheme":30},
}

# Model simulation, runModel_GradientBoosting
setModel_Output, model_output = runModel_GradientBoosting(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Gradient Boosting")


#------------------
setModelRun = {
    "IdRun": "GB03",
    "options": {"scheme":30,"gridSCV": True},
}

# Model simulation, runModel_GradientBoosting
setModel_Output, model_output = runModel_GradientBoosting(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Gradient Boosting")


#------------------
setModelRun = {
    "IdRun": "GB04",
    "options": {"scheme":30,"randSCV": True},
}

# Model simulation, runModel_GradientBoosting
setModel_Output, model_output = runModel_GradientBoosting(setModelRun, X, y)

# Set the Log dataframe and save the model
```

```
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Gradient Boosting")
```

	ModelName	IdRun	DateAction	options	Params	Remarks	AccuracyTrai
8	Gradient Boosting	GB01	2025-05-08 03:06:51	{'scheme': 20}	{'modelParams': {'random_state': 42}}	NaN	0.97163
9	Gradient Boosting	GB02	2025-05-08 03:06:51	{'scheme': 30}	{'modelParams': {'random_state': 42}}	NaN	0.97702
10	Gradient Boosting	GB03	2025-05-08 03:08:31	{'scheme': 30, 'gridSCV': True}	{'modelParams': {'random_state': 42}, 'BestParamsGridSCV': {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 50}}	NaN	1.00000
11	Gradient Boosting	GB04	2025-05-08 03:08:46	{'scheme': 30, 'randSCV': True}	{'modelParams': {'random_state': 42}, 'BestParamsRandSCV': {'learning_rate': 0.19727005942368125, 'max_depth': 3, 'n_estimators': 64}}	NaN	0.98513

Support Vector Machines (SVM)

In [ ]:
```
from scripts.runModel_SVM import *

# Running several training models
#------------------
setModelRun = {
    "IdRun": "SV01",
    "options": {"scheme":20},
}

# Model simulation, runModel_SVM
setModel_Output, model_output = runModel_SVM(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Support Vector Machine")


#------------------
setModelRun = {
    "IdRun": "SV02",
    "options": {"scheme":30},
}

# Model simulation, runModel_SVM
setModel_Output, model_output = runModel_SVM(setModelRun, X, y)

# Set the Log dataframe and save the model
```

```
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Support Vector Machine")
```

	ModelName	IdRun	DateAction	options	Params	Remarks	AccuracyTrain	Acc
20	Support Vector Machine	SV01	2025-05-08 03:08:47	{'scheme': 20}	{'modelParams': {'probability': True}}	NaN	0.834515	
21	Support Vector Machine	SV02	2025-05-08 03:08:48	{'scheme': 30}	{'modelParams': {'probability': True}}	NaN	0.832432	

K-Nearest Neighbors (KNN)

In [ ]:

```
from scripts.runModel_KNN import *


# Running several training models
#------------------
setModelRun = {
    "IdRun": "KN01",
    "options": {"scheme":20},
}

# Model simulation, runModel_KNN
setModel_Output, model_output = runModel_KNN(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "K-Nearest Neighbors")


#------------------
setModelRun = {
    "IdRun": "KN02",
    "options": {"scheme":30},
}

# Model simulation, runModel_KNN
setModel_Output, model_output = runModel_KNN(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "K-Nearest Neighbors")


#------------------
setModelRun = {
    "IdRun": "KN03",
    "options": {"scheme":30,"gridSCV": True},
}

# Model simulation, runModel_KNN
setModel_Output, model_output = runModel_KNN(setModelRun, X, y)
```

```
# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "K-Nearest Neighbors")


#------------------
setModelRun = {
    "IdRun": "KN04",
    "options": {"scheme":30,"randSCV": True},
}

# Model simulation, runModel_KNN
setModel_Output, model_output = runModel_KNN(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "K-Nearest Neighbors")
```

	ModelName	IdRun	DateAction	options	Params	Remarks	AccuracyTrai
12	K-Nearest Neighbors	KN01	2025-05-08 03:08:48	{'scheme': 20}	{'modelParams': {}}	NaN	0.85579
13	K-Nearest Neighbors	KN02	2025-05-08 03:08:49	{'scheme': 30}	{'modelParams': {}}	NaN	0.84864
14	K-Nearest Neighbors	KN03	2025-05-08 03:08:50	{'scheme': 30, 'gridSCV': True}	{'modelParams': {}, 'BestParamsGridSCV': {'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'uniform'}}	NaN	0.84189
15	K-Nearest Neighbors	KN04	2025-05-08 03:08:51	{'scheme': 30, 'randSCV': True}	{'modelParams': {}, 'BestParamsRandSCV': {'metric': 'manhattan', 'n_neighbors': 14, 'weights': 'uniform'}}	NaN	0.83243

XGBoost (Extreme Gradient Boosting)

In [ ]:
```
from scripts.runModel_XGBoost import *

# Running several training models
#------------------
setModelRun = {
    "IdRun": "XG01",
    "options": {"scheme":20},
}

# Model simulation, runModel_XGBoost
setModel_Output, model_output = runModel_XGBoost(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
```

```
showResultDfModel(resultRunModel_df, ModelName = "XGBoost")


#------------------
setModelRun = {
    "IdRun": "XG02",
    "options": {"scheme":30},
}

# Model simulation, runModel_XGBoost
setModel_Output, model_output = runModel_XGBoost(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "XGBoost")


#------------------
setModelRun = {
    "IdRun": "XG03",
    "options": {"scheme":30,"gridSCV": True},
}

# Model simulation, runModel_XGBoost
setModel_Output, model_output = runModel_XGBoost(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "XGBoost")


#------------------
setModelRun = {
    "IdRun": "XG04",
    "options": {"scheme":30,"randSCV": True},
}

# Model simulation, runModel_XGBoost
setModel_Output, model_output = runModel_XGBoost(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "XGBoost")
```

	ModelName	IdRun	DateAction	options	Params	Remarks	AccuracyTrai
26	XGBoost	XG01	2025-05-08 03:08:52	{'scheme': 20}	{'modelParams': {'random_state': 42, 'use_label_encoder': False, 'eval_metric': 'mlogloss'}}	NaN	1.00000
27	XGBoost	XG02	2025-05-08 03:08:52	{'scheme': 30}	{'modelParams': {'random_state': 42, 'use_label_encoder': False, 'eval_metric': 'mlogloss'}}	NaN	1.00000
28	XGBoost	XG03	2025-05-08 03:09:52	{'scheme': 30, 'gridSCV': True}	{'modelParams': {'random_state': 42, 'use_label_encoder': False, 'eval_metric': 'mlogloss'}, 'BestParamsGridSCV': {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50, 'subsample': 0.8}}	NaN	0.91486
29	XGBoost	XG04	2025-05-08 03:10:01	{'scheme': 30, 'randSCV': True}	{'modelParams': {'random_state': 42, 'use_label_encoder': False, 'eval_metric': 'mlogloss'}, 'BestParamsRandSCV': {'learning_rate': 0.29156581270472504, 'max_depth': 4, 'n_estimators': 70, 'subsample': 0.8852444528883149}}	NaN	1.00000

Multi-Layer Perceptron (MLP) – Neural Network

In [ ]:
```
from scripts.runModel_MLP import *


# Running several training models
#------------------
setModelRun = {
    "IdRun": "MP01",
    "options": {"scheme":20},
}

# Model simulation, runModel_MLP
setModel_Output, model_output = runModel_MLP(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
```

```
showResultDfModel(resultRunModel_df, ModelName = "Multi-Layer Perceptron")


#------------------
setModelRun = {
    "IdRun": "MP02",
    "options": {"scheme":30},
}

# Model simulation, runModel_MLP
setModel_Output, model_output = runModel_MLP(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Multi-Layer Perceptron")


#------------------
setModelRun = {
    "IdRun": "MP03",
    "options": {"scheme":30,"gridSCV": True},
}

# Model simulation, runModel_MLP
setModel_Output, model_output = runModel_MLP(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Multi-Layer Perceptron")


#------------------
setModelRun = {
    "IdRun": "MP04",
    "options": {"scheme":30,"randSCV": True},
}

# Model simulation, runModel_MLP
setModel_Output, model_output = runModel_MLP(setModelRun, X, y)

# Set the Log dataframe and save the model
resultRunModel_df = updateRunModelDf(df=resultRunModel_df, setx=setModel_Output, mo
showResultDfModel(resultRunModel_df, ModelName = "Multi-Layer Perceptron")
```

	ModelName	IdRun	DateAction	options	Params	Remarks	AccuracyTr
20	Multi-Layer Perceptron	MP01	2025-05-08 03:10:02	{'scheme': 20}	{'modelParams': {'max_iter': 300, 'random_state': 42}}	NaN	0.822
21	Multi-Layer Perceptron	MP02	2025-05-08 03:10:02	{'scheme': 30}	{'modelParams': {'max_iter': 300, 'random_state': 42}}	NaN	0.831
22	Multi-Layer Perceptron	MP03	2025-05-08 03:10:59	{'scheme': 30, 'gridSCV': True}	{'modelParams': {'max_iter': 300, 'random_state': 42}, 'BestParamsGridSCV': {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'adam'}}	NaN	0.832
23	Multi-Layer Perceptron	MP04	2025-05-08 03:11:16	{'scheme': 30, 'randSCV': True}	{'modelParams': {'max_iter': 300, 'random_state': 42}, 'BestParamsRandSCV': {'activation': 'tanh', 'alpha': 0.006086584841970367, 'hidden_layer_sizes': (30, 30, 30), 'solver': 'sgd'}}	NaN	0.832

Evaluation

In [ ]:
```
resultRunModel_df
```

Out[ ]:

	ModelName	IdRun	DateAction	options	Params	Remarks	Accuracy
0	Adaboost Classifier	AD01	2025-05-08 03:06:24	{'scheme': 20}	{'modelParams': {'random_state': 42}}	NaN	0.87
1	Adaboost Classifier	AD02	2025-05-08 03:06:24	{'scheme': 30}	{'modelParams': {'random_state': 42}}	NaN	0.87
2	Adaboost Classifier	AD03	2025-05-08 03:06:39	{'scheme': 30, 'gridSCV': True}	{'modelParams': {'random_state': 42}, 'BestParamsGridSCV': {'learning_rate': 1.0, 'n_estimators': 50}}	NaN	0.87
3	Adaboost Classifier	AD04	2025-05-08 03:06:50	{'scheme': 30, 'randSCV': True}	{'modelParams': {'random_state': 42}, 'BestParamsRandSCV': {'learning_rate': 0.7319987722668247, 'n_estimators': 79}}	NaN	0.87
4	DecisionTree Classifier	DF01	2025-05-08 03:06:18	{'scheme': 20}	{'modelParams': {'random_state': 42}}	NaN	1.00
5	DecisionTree Classifier	DF02	2025-05-08 03:06:18	{'scheme': 30}	{'modelParams': {'random_state': 42}}	NaN	1.00
6	DecisionTree Classifier	DF03	2025-05-08 03:06:23	{'scheme': 30, 'gridSCV': True}	{'modelParams': {'random_state': 42}, 'BestParamsGridSCV': {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2}}	NaN	0.96
7	DecisionTree Classifier	DF04	2025-05-08 03:06:23	{'scheme': 30, 'randSCV': True}	{'modelParams': {'random_state': 42}, 'BestParamsRandSCV': {'criterion': 'entropy', 'max_depth': 2, 'min_samples_leaf': 8, 'min_samples_split': 13}}	NaN	0.83
8	Gradient Boosting	GB01	2025-05-08 03:06:51	{'scheme': 20}	{'modelParams': {'random_state': 42}}	NaN	0.97
9	Gradient Boosting	GB02	2025-05-08 03:06:51	{'scheme': 30}	{'modelParams': {'random_state': 42}}	NaN	0.97
10	Gradient Boosting	GB03	2025-05-08 03:08:31	{'scheme': 30, 'gridSCV': True}	{'modelParams': {'random_state': 42}, 'BestParamsGridSCV': {'learning_rate': 0.5, 'max_depth': 5, 'n_estimators': 50}}	NaN	1.00
11	Gradient Boosting	GB04	2025-05-08 03:08:46	{'scheme': 30,	{'modelParams': {'random_state': 42},	NaN	0.98

	ModelName	IdRun	DateAction	options	Params	Remarks	Accuracy
				'randSCV': True}	'BestParamsRandSCV': {'learning_rate': 0.19727005942368125, 'max_depth': 3, 'n_estimators': 64}}		
12	K-Nearest Neighbors	KN01	2025-05-08 03:08:48	{'scheme': 20}	{'modelParams': {}}	NaN	0.85
13	K-Nearest Neighbors	KN02	2025-05-08 03:08:49	{'scheme': 30}	{'modelParams': {}}	NaN	0.84
14	K-Nearest Neighbors	KN03	2025-05-08 03:08:50	{'scheme': 30, 'gridSCV': True}	{'modelParams': {}, 'BestParamsGridSCV': {'metric': 'manhattan', 'n_neighbors': 9, 'weights': 'uniform'}}	NaN	0.84
15	K-Nearest Neighbors	KN04	2025-05-08 03:08:51	{'scheme': 30, 'randSCV': True}	{'modelParams': {}, 'BestParamsRandSCV': {'metric': 'manhattan', 'n_neighbors': 14, 'weights': 'uniform'}}	NaN	0.83
16	Logistic Regression	LR01	2025-05-08 03:01:02	{'scheme': 20}	{'modelParams': {'random_state': 24}}	NaN	0.83
17	Logistic Regression	LR02	2025-05-08 03:01:02	{'scheme': 30}	{'modelParams': {'random_state': 24}}	NaN	0.83
18	Logistic Regression	LR03	2025-05-08 03:01:07	{'scheme': 30, 'gridSCV': True}	{'modelParams': {'random_state': 24, 'n_jobs': 2}, 'BestParamsGridSCV': {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}}	NaN	0.86
19	Logistic Regression	LR04	2025-05-08 03:01:10	{'scheme': 30, 'randSCV': True}	{'modelParams': {'max_iter': 500, 'random_state': 24}, 'BestParamsRandSCV': {'C': 0.13292918943162169, 'penalty': 'l1', 'solver': 'liblinear'}}	NaN	0.85
20	Multi-Layer Perceptron	MP01	2025-05-08 03:10:02	{'scheme': 20}	{'modelParams': {'max_iter': 300, 'random_state': 42}}	NaN	0.82
21	Multi-Layer Perceptron	MP02	2025-05-08 03:10:02	{'scheme': 30}	{'modelParams': {'max_iter': 300, 'random_state': 42}}	NaN	0.83
22	Multi-Layer Perceptron	MP03	2025-05-08 03:10:59	{'scheme': 30,	{'modelParams': {'max_iter': 300, 'random_state': 42},	NaN	0.83

	ModelName	IdRun	DateAction	options	Params	Remarks	Accuracy
				'gridSCV': True}	'BestParamsGridSCV': {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'solver': 'adam'}}		
23	Multi-Layer Perceptron	MP04	2025-05-08 03:11:16	{'scheme': 30, 'randSCV': True}	{'modelParams': {'max_iter': 300, 'random_state': 42}, 'BestParamsRandSCV': {'activation': 'tanh', 'alpha': 0.006086584841970367, 'hidden_layer_sizes': (30, 30, 30), 'solver': 'sgd'}}	NaN	0.83
24	Random Forest	RF01	2025-05-08 03:01:11	{'scheme': 20}	{'modelParams': {'n_jobs': 2, 'random_state': 24}}	NaN	1.00
25	Random Forest	RF02	2025-05-08 03:01:12	{'scheme': 30}	{'modelParams': {'n_jobs': 2, 'random_state': 24}}	NaN	1.00
26	Random Forest	RF03	2025-05-08 03:06:12	{'scheme': 30, 'gridSCV': True}	{'modelParams': {'random_state': 42, 'n_jobs': 2}, 'BestParamsGridSCV': {'max_depth': 20, 'max_features': 0.5, 'min_samples_leaf': 1, 'min_samples_split': 15, 'n_estimators': 500}}	NaN	0.93
27	Random Forest	RF04	2025-05-08 03:06:18	{'scheme': 30, 'randSCV': True}	{'modelParams': {'random_state': 42, 'n_jobs': 2}, 'BestParamsRandSCV': {'n_estimators': 50, 'min_samples_split': 2, 'max_depth': 10}}	NaN	0.99
28	Support Vector Machine	SV01	2025-05-08 03:08:47	{'scheme': 20}	{'modelParams': {'probability': True}}	NaN	0.83
29	Support Vector Machine	SV02	2025-05-08 03:08:48	{'scheme': 30}	{'modelParams': {'probability': True}}	NaN	0.83
30	XGBoost	XG01	2025-05-08 03:08:52	{'scheme': 20}	{'modelParams': {'random_state': 42, 'use_label_encoder': False, 'eval_metric': 'mlogloss'}}	NaN	1.00

	ModelName	IdRun	DateAction	options	Params	Remarks	Accuracy
31	XGBoost	XG02	2025-05-08 03:08:52	{'scheme': 30}	{'modelParams': {'random_state': 42, 'use_label_encoder': False, 'eval_metric': 'mlogloss'}}	NaN	1.00
32	XGBoost	XG03	2025-05-08 03:09:52	{'scheme': 30, 'gridSCV': True}	{'modelParams': {'random_state': 42, 'use_label_encoder': False, 'eval_metric': 'mlogloss'}, 'BestParamsGridSCV': {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 50, 'subsample': 0.8}}	NaN	0.91
33	XGBoost	XG04	2025-05-08 03:10:01	{'scheme': 30, 'randSCV': True}	{'modelParams': {'random_state': 42, 'use_label_encoder': False, 'eval_metric': 'mlogloss'}, 'BestParamsRandSCV': {'learning_rate': 0.29156581270472504, 'max_depth': 4, 'n_estimators': 70, 'subsample': 0.8852444528883149}}	NaN	1.00

Rank Composite

Metric Ranking Based on General Perception in Machine Learning

In many machine learning scenarios, the importance of evaluation metrics (in general perception) is often ranked as follows:

1. **F1ScoreTest** → The most balanced metric between Precision and Recall, especially useful for imbalanced datasets.
2. **RecallTest** → Important when reducing false negatives is a priority (e.g., in healthcare, fraud detection).
3. **PrecisionTest** → Important when reducing false positives is critical (e.g., in spam detection).
4. **AccuracyTest** → Useful if the dataset is balanced, but can be misleading with imbalanced data.

5. **AccuracyTrain** → Useful to check for overfitting, but not very meaningful as a standalone metric for final model evaluation.

Summary Ranking Based on General Perception:

Rank	Metric	Reason
1	F1ScoreTest	Balances precision and recall, most representative metric overall.
2	RecallTest	Important for maximizing true positives.
3	PrecisionTest	Important for high accuracy in positive predictions.
4	AccuracyTest	Commonly used, but biased with imbalanced datasets.
5	AccuracyTrain	Only useful for detecting overfitting, not for final model assessment.

Metric Weights (0-1):

Metric	Weight (0-1)	Explanation
F1ScoreTest	0.40	Main focus, balances both precision and recall.
RecallTest	0.20	Important to reduce false negatives.
PrecisionTest	0.15	Important to reduce false positives.
AccuracyTest	0.15	Still considered, but not the primary metric.
AccuracyTrain	0.10	Useful for overfitting detection, minor contribution.

In [ ]:
```
def calc_rank_rmodels(df):
    remarks = []

    def apply_penalty(row):
        remark = []
        if row['AccuracyTrain'] == 1:
            row['AccuracyTrain'] -= (row['AccuracyTrain'] - row['F1ScoreTest']) * 0
            remark.append("AccuracyTrain= 1")
        if row['AccuracyTest'] == 1:
            row['AccuracyTest'] -= (row['AccuracyTest'] - row['F1ScoreTest']) * 0.7
            remark.append("AccuracyTest= 1")
        remarks.append("Penalized: " + ", ".join(remark) if remark else "")
        return row

    # Terapkan penalti dan kumpulkan remarks
    df = df.apply(apply_penalty, axis=1)
    df['Remarks'] = remarks

    # Menambahkan ranking untuk setiap metrik
    df['RankAccuracyTrain'] = df['AccuracyTrain'].rank(ascending=False)
    df['RankAccuracyTest'] = df['AccuracyTest'].rank(ascending=False)
    df['RankF1ScoreTest'] = df['F1ScoreTest'].rank(ascending=False)
```

```
    df['RankPrecisionTest'] = df['PrecisionTest'].rank(ascending=False)
    df['RankRecallTest'] = df['RecallTest'].rank(ascending=False)

    # Bobot untuk masing-masing metrik
    weights = {
        'F1ScoreTest': 0.4,
        'RecallTest': 0.2,
        'PrecisionTest': 0.15,
        'AccuracyTest': 0.15,
        'AccuracyTrain': 0.1,
    }

    # Menghitung skor komposit berdasarkan bobot
    df['CompositeScore'] = (
        df['AccuracyTrain'] * weights['AccuracyTrain'] +
        df['AccuracyTest'] * weights['AccuracyTest'] +
        df['F1ScoreTest'] * weights['F1ScoreTest'] +
        df['PrecisionTest'] * weights['PrecisionTest'] +
        df['RecallTest'] * weights['RecallTest']
    )

    # Me-rangking berdasarkan CompositeScore
    df['RankComposite'] = df['CompositeScore'].rank(ascending=False)

    # Mengurutkan berdasarkan RankComposite
    df_sorted = df.sort_values(by='RankComposite')
    df_sorted = df.sort_values('RankComposite').reset_index(drop=True)
    df_sorted.index = df_sorted.index + 1

    return df_sorted[['RankComposite', 'IdRun', 'ModelName', 'Remarks',
                      'AccuracyTrain', 'AccuracyTest',
                      'F1ScoreTest', 'PrecisionTest', 'RecallTest',
                      'options', 'Params']]
```

In [ ]:
```
from scripts.getCalcRank_RModels import get_calc_rank_rmodels
df_model_rankscore = pd.read_csv(f"{models_path}resultRunModel_df.csv", index_col=0
df_model_rankscore = get_calc_rank_rmodels(df_model_rankscore)
df_model_rankscore
```

Out[ ]:

	RankComposite	IdRun	ModelName	Remarks	AccuracyTrain	AccuracyTest	F1Sc
1	1.0	GB04	Gradient Boosting		0.985135	0.871069	0
2	2.0	LR03	Logistic Regression		0.862162	0.880503	0
3	3.0	XG03	XGBoost		0.914865	0.874214	0
4	4.0	XG04	XGBoost	Penalized: AccuracyTrain= 1	0.891938	0.874214	0
5	5.0	AD04	Adaboost Classifier		0.871622	0.874214	0
6	6.0	LR04	Logistic Regression		0.855405	0.874214	0
7	7.5	AD03	Adaboost Classifier		0.870270	0.867925	0
8	7.5	AD02	Adaboost Classifier		0.870270	0.867925	0
9	9.0	GB03	Gradient Boosting	Penalized: AccuracyTrain= 1	0.880668	0.858491	0
10	10.0	GB02	Gradient Boosting		0.977027	0.852201	0

	RankComposite	IdRun	ModelName	Remarks	AccuracyTrain	AccuracyTest	F1Sc
11	11.0	RF02	Random Forest	Penalized: AccuracyTrain= 1	0.865641	0.861635	0
12	12.0	RF03	Random Forest		0.935135	0.855346	0
13	13.0	XG02	XGBoost	Penalized: AccuracyTrain= 1	0.877274	0.855346	0
14	14.0	GB01	Gradient Boosting		0.971631	0.849057	0
15	15.0	DF04	DecisionTree Classifier		0.836486	0.858491	0
16	16.0	AD01	Adaboost Classifier		0.875887	0.853774	0
17	17.0	RF04	Random Forest		0.993243	0.849057	0
18	18.0	RF01	Random Forest	Penalized: AccuracyTrain= 1	0.852208	0.849057	0
19	19.0	XG01	XGBoost	Penalized: AccuracyTrain= 1	0.862265	0.839623	0
20	20.0	DF03	DecisionTree Classifier		0.964865	0.811321	0
21	21.0	LR02	Logistic Regression		0.833784	0.836478	0
22	22.0	KN01	K-Nearest Neighbors		0.855792	0.816038	0

	RankComposite	IdRun	ModelName	Remarks	AccuracyTrain	AccuracyTest	F1Sc
23	23.0	LR01	Logistic Regression		0.836879	0.820755	0
24	24.0	KN02	K-Nearest Neighbors		0.848649	0.814465	0
25	25.0	DF02	DecisionTree Classifier	Penalized: AccuracyTrain= 1	0.837842	0.783019	0
26	26.0	MP01	Multi-Layer Perceptron		0.822695	0.811321	0
27	28.5	SV02	Support Vector Machine		0.832432	0.827044	0
28	28.5	MP04	Multi-Layer Perceptron		0.832432	0.827044	0
29	28.5	KN04	K-Nearest Neighbors		0.832432	0.827044	0
30	28.5	MP03	Multi-Layer Perceptron		0.832432	0.827044	0
31	31.0	KN03	K-Nearest Neighbors		0.841892	0.823899	0
32	32.0	MP02	Multi-Layer Perceptron		0.831081	0.817610	0
33	33.0	SV01	Support Vector Machine		0.834515	0.816038	0
34	34.0	DF01	DecisionTree Classifier	Penalized: AccuracyTrain= 1	0.812591	0.764151	0

Business Dashboard

This business dashboard is created using a platform **Google Looker Studio**.

image

The dashboard can be accessed through the following link: 🔗
lookerstudio.google.com/s/grjGn3AvSms

Prediction (optional)

In [ ]:
```
# ========================Import module and setup========================
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from joblib import load
from prettytable import PrettyTable

# Configure the display of pandas DataFrame to be maximized during debugging
pd.set_option("display.max_columns", None)
pd.set_option("display.max_colwidth", None)
pd.set_option("display.width", 0)
pd.set_option("display.expand_frame_repr", False)

# ========================1. Load Dataset & Model========================
employee_file = 'saved/employee_attrition_nan.csv'
employee_df = pd.read_csv(employee_file)

model_file = "models/model_GB04.pkl"
loaded_model = load(model_file)

# ========================2. Column Data Type========================
employee_col_numeric_int = pd.Index([
    'Age', 'DailyRate', 'DistanceFromHome', 'HourlyRate',
    'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked',
    'PercentSalaryHike', 'TotalWorkingYears', 'TrainingTimesLastYear',
    'YearsAtCompany', 'YearsInCurrentRole',
    'YearsSinceLastPromotion', 'YearsWithCurrManager'
])

employee_col_categorical = pd.Index([
    'BusinessTravel', 'Department', 'Education', 'EducationField',
    'EnvironmentSatisfaction', 'Gender', 'JobInvolvement', 'JobLevel',
    'JobRole', 'JobSatisfaction', 'MaritalStatus', 'OverTime',
    'PerformanceRating', 'RelationshipSatisfaction', 'StockOptionLevel',
    'WorkLifeBalance'
])

employee_col_object = pd.Index(['EmployeeId'])

# ======================== 3. Data Type Conversion ========================
employee_df[employee_col_numeric_int] = employee_df[employee_col_numeric_int].astyp
employee_df[employee_col_categorical] = employee_df[employee_col_categorical].astyp
employee_df['EmployeeId'] = employee_df['EmployeeId'].astype('object')

# ========================4. Encoding Categorical Column========================
```

```
categorical_cols = employee_df.select_dtypes(include=['object', 'category', 'bool']
employee_df_encode = employee_df.copy()

le = LabelEncoder()
for col in categorical_cols:
    employee_df_encode[col] = le.fit_transform(employee_df_encode[col])

# ========================5. Predictions & Results========================
employee_id_list = employee_df_encode['EmployeeId'].tolist()
results = []

for employee_id in employee_id_list:
    row_prediksi = employee_df_encode[employee_df_encode['EmployeeId'] == employee_
    row_prediksi = row_prediksi.drop(columns=['EmployeeId', 'Attrition'])

    prediction = loaded_model.predict(row_prediksi)[0]
    prediction_label = "Active" if prediction == 0 else "Resigned"

    results.append({
        'EmployeeId': employee_id,
        'Prediction': prediction,
        'Status': prediction_label
    })

df_predictions = pd.DataFrame(results)

# Save Results to CSV
output_file = os.path.join('saved', employee_file.replace('.csv', '') + '_predictio
output_file = employee_file.replace('.csv', '') + '_predictions.csv'
df_predictions.to_csv(output_file, index=False)


# ========================= 7. Print Results with PrettyTable ====================
table = PrettyTable()
table.field_names = df_predictions.columns.tolist()

for _, row in df_predictions.iterrows():
    table.add_row(row.tolist())

print(table)
```

EmployeeId	Prediction	Status
0	0	Active
1	0	Active
2	0	Active
3	0	Active
4	0	Active
5	0	Active
6	0	Active
7	0	Active
8	0	Active
9	0	Active
10	0	Active
11	0	Active
12	0	Active
13	0	Active
14	0	Active
15	0	Active
16	0	Active
17	0	Active
18	0	Active
19	0	Active
20	0	Active
21	0	Active
22	0	Active
23	0	Active
24	0	Active
25	0	Active
26	0	Active
27	0	Active
28	0	Active
29	1	Resigned
30	0	Active
31	1	Resigned
32	0	Active
33	0	Active
34	0	Active
35	0	Active
36	0	Active
37	0	Active
38	1	Resigned
39	0	Active
40	0	Active
41	0	Active
42	0	Active
43	0	Active
44	0	Active
45	0	Active
46	0	Active
47	0	Active
48	0	Active
49	0	Active
50	0	Active
51	0	Active
52	0	Active

53	0	Active
54	0	Active
55	0	Active
56	0	Active
57	0	Active
58	0	Active
59	1	Resigned
60	0	Active
61	0	Active
62	0	Active
63	0	Active
64	0	Active
65	0	Active
66	0	Active
67	0	Active
68	0	Active
69	0	Active
70	1	Resigned
71	0	Active
72	0	Active
73	0	Active
74	0	Active
75	0	Active
76	0	Active
77	0	Active
78	0	Active
79	0	Active
80	0	Active
81	0	Active
82	0	Active
83	0	Active
84	0	Active
85	0	Active
86	0	Active
87	0	Active
88	1	Resigned
89	0	Active
90	0	Active
91	0	Active
92	0	Active
93	0	Active
94	0	Active
95	0	Active
96	0	Active
97	0	Active
98	0	Active
99	0	Active
100	0	Active
101	0	Active
102	0	Active
103	0	Active
104	0	Active
105	0	Active
106	0	Active
107	0	Active
108	0	Active

109	0	Active
110	0	Active
111	0	Active
112	0	Active
113	0	Active
114	0	Active
115	0	Active
116	0	Active
117	0	Active
118	0	Active
119	0	Active
120	0	Active
121	1	Resigned
122	0	Active
123	0	Active
124	0	Active
125	0	Active
126	0	Active
127	0	Active
128	0	Active
129	0	Active
130	0	Active
131	0	Active
132	0	Active
133	0	Active
134	1	Resigned
135	0	Active
136	1	Resigned
137	0	Active
138	0	Active
139	0	Active
140	0	Active
141	0	Active
142	0	Active
143	0	Active
144	0	Active
145	0	Active
146	0	Active
147	0	Active
148	1	Resigned
149	0	Active
150	0	Active
151	0	Active
152	0	Active
153	0	Active
154	0	Active
155	0	Active
156	0	Active
157	0	Active
158	0	Active
159	0	Active
160	0	Active
161	0	Active
162	0	Active
163	0	Active
164	0	Active

165	1	Resigned
166	0	Active
167	1	Resigned
168	0	Active
169	0	Active
170	0	Active
171	0	Active
172	0	Active
173	0	Active
174	0	Active
175	0	Active
176	0	Active
177	0	Active
178	0	Active
179	0	Active
180	1	Resigned
181	0	Active
182	0	Active
183	0	Active
184	0	Active
185	0	Active
186	0	Active
187	0	Active
188	0	Active
189	0	Active
190	1	Resigned
191	0	Active
192	0	Active
193	0	Active
194	0	Active
195	0	Active
196	1	Resigned
197	0	Active
198	0	Active
199	0	Active
200	0	Active
201	1	Resigned
202	0	Active
203	0	Active
204	0	Active
205	0	Active
206	0	Active
207	1	Resigned
208	0	Active
209	0	Active
210	0	Active
211	0	Active
212	0	Active
213	0	Active
214	0	Active
215	0	Active
216	0	Active
217	0	Active
218	0	Active
219	1	Resigned
220	0	Active

221	0	Active
222	0	Active
223	0	Active
224	0	Active
225	0	Active
226	0	Active
227	0	Active
228	0	Active
229	1	Resigned
230	0	Active
231	1	Resigned
232	0	Active
233	0	Active
234	1	Resigned
235	0	Active
236	0	Active
237	0	Active
238	0	Active
239	0	Active
240	0	Active
241	0	Active
242	0	Active
243	0	Active
244	0	Active
245	0	Active
246	0	Active
247	0	Active
248	0	Active
249	0	Active
250	1	Resigned
251	0	Active
252	0	Active
253	0	Active
254	0	Active
255	0	Active
256	1	Resigned
257	0	Active
258	0	Active
259	0	Active
260	1	Resigned
261	0	Active
262	0	Active
263	0	Active
264	0	Active
265	0	Active
266	0	Active
267	1	Resigned
268	0	Active
269	0	Active
270	0	Active
271	0	Active
272	0	Active
273	0	Active
274	0	Active
275	0	Active
276	0	Active

277	0	Active
278	0	Active
279	0	Active
280	0	Active
281	0	Active
282	0	Active
283	0	Active
284	0	Active
285	0	Active
286	0	Active
287	0	Active
288	0	Active
289	0	Active
290	0	Active
291	0	Active
292	0	Active
293	0	Active
294	0	Active
295	0	Active
296	0	Active
297	0	Active
298	0	Active
299	0	Active
300	0	Active
301	0	Active
302	0	Active
303	0	Active
304	1	Resigned
305	0	Active
306	0	Active
307	0	Active
308	0	Active
309	1	Resigned
310	0	Active
311	0	Active
312	0	Active
313	0	Active
314	0	Active
315	0	Active
316	0	Active
317	0	Active
318	1	Resigned
319	0	Active
320	0	Active
321	0	Active
322	0	Active
323	0	Active
324	0	Active
325	0	Active
326	0	Active
327	0	Active
328	0	Active
329	0	Active
330	0	Active
331	0	Active
332	1	Resigned

333	0	Active
334	0	Active
335	0	Active
336	0	Active
337	0	Active
338	1	Resigned
339	0	Active
340	0	Active
341	0	Active
342	0	Active
343	1	Resigned
344	0	Active
345	0	Active
346	0	Active
347	0	Active
348	0	Active
349	0	Active
350	0	Active
351	1	Resigned
352	0	Active
353	1	Resigned
354	1	Resigned
355	0	Active
356	0	Active
357	0	Active
358	0	Active
359	0	Active
360	0	Active
361	1	Resigned
362	0	Active
363	0	Active
364	0	Active
365	0	Active
366	0	Active
367	0	Active
368	0	Active
369	0	Active
370	0	Active
371	1	Resigned
372	0	Active
373	0	Active
374	0	Active
375	0	Active
376	0	Active
377	1	Resigned
378	0	Active
379	0	Active
380	0	Active
381	0	Active
382	0	Active
383	1	Resigned
384	0	Active
385	0	Active
386	0	Active
387	0	Active
388	0	Active

389	0	Active
390	0	Active
391	0	Active
392	0	Active
393	0	Active
394	0	Active
395	0	Active
396	0	Active
397	0	Active
398	0	Active
399	0	Active
400	0	Active
401	0	Active
402	0	Active
403	0	Active
404	0	Active
405	0	Active
406	0	Active
407	0	Active
408	0	Active
409	0	Active
410	0	Active
411	0	Active