In [1]:
```
# import libraries
import pandas as pd
import numpy as np
```

In [76]:
```
# Load data
df= pd.read_csv('hr_dataset.csv')
df.head()
```

Out[76]:

	Employee_ID	Age	Department	Satisfaction_Level	Last_Evaluation	Projects	Average_Mon
0	896999	NaN	Finance	0.42	0.08	NaN	
1	331148	NaN	Hr	0.91	0.73	NaN	
2	559437	36.0	Operations	0.93	0.82	7.0	
3	883201	41.0	Finance	0.03	0.53	7.0	
4	562242	NaN	Finance	0.66	0.72	3.0	

In [77]:
```
print(df.head(40))
```

	Employee_ID	Age	Department	Satisfaction_Level	Last_Evaluation	\
0	896999	NaN	Finance	0.42	0.08	
1	331148	NaN	Hr	0.91	0.73	
2	559437	36.0	Operations	0.93	0.82	
3	883201	41.0	Finance	0.03	0.53	
4	562242	NaN	Finance	0.66	0.72	
5	538510	NaN	Sales	0.85	0.76	
6	585585	36.0	NaN	0.65	0.39	
7	689574	NaN	Hr	0.89	0.80	
8	394433	58.0	Hr	0.33	0.92	
9	314638	NaN	IT	0.57	0.42	
10	9767	NaN	Hr	0.62	0.26	
11	747950	NaN	Finance	0.71	0.09	
12	102408	NaN	finanCe	0.27	0.12	
13	726713	NaN	NaN	0.31	0.00	
14	40517	43.0	NaN	0.13	0.68	
15	420135	NaN	NaN	0.65	0.21	
16	913501	NaN	Sales	0.28	0.80	
17	783720	NaN	Operations	0.44	0.88	
18	268678	NaN	NaN	0.09	0.56	
19	556615	NaN	Sales	0.13	0.02	
20	574096	NaN	Hr	0.09	0.75	
21	873303	48.0	Hr	0.95	0.70	
22	19284	23.0	finanCe	0.03	0.20	
23	323130	34.0	NaN	0.42	0.99	
24	189340	NaN	NaN	0.67	0.29	
25	776669	NaN	Hr	0.82	0.96	
26	634699	NaN	Hr	0.01	0.08	
27	110188	NaN	NaN	0.06	0.98	
28	196980	NaN	finanCe	0.17	0.60	
29	35597	NaN	Finance	0.76	0.05	
30	23450	NaN	Operations	0.20	0.29	
31	606051	31.0	NaN	0.12	0.00	
32	101643	33.0	Finance	0.29	0.08	
33	922195	57.0	Operations	0.20	0.53	
34	857149	44.0	finanCe	0.21	0.14	
35	827733	27.0	Hr	0.38	0.75	
36	857918	NaN	Finance	0.72	0.83	
37	68193	NaN	Hr	0.94	0.73	
38	94198	39.0	finanCe	0.01	0.69	
39	281086	NaN	Hr	0.42	0.96	

	Projects	Average_Monthly_Hours	Years_at_Company	Left
0	NaN	999	NaN	1.0
1	NaN	180	7.0	1.0
2	7.0	999	4.0	1.0
3	7.0	297	NaN	1.0
4	3.0	186	5.0	NaN
5	4.0	999	NaN	1.0
6	NaN	999	5.0	NaN
7	NaN	227	NaN	1.0
8	3.0	205	7.0	NaN
9	NaN	227	NaN	1.0
10	NaN	999	1.0	1.0
11	NaN	268	7.0	NaN
12	NaN	999	NaN	NaN
13	5.0	999	8.0	1.0
14	NaN	999	8.0	NaN
15	2.0	193	8.0	1.0
16	2.0	210	1.0	1.0

17	5.0	999	NaN	0.0
18	6.0	999	NaN	1.0
19	NaN	999	7.0	NaN
20	NaN	292	4.0	NaN
21	NaN	260	2.0	NaN
22	NaN	276	NaN	0.0
23	NaN	209	5.0	NaN
24	7.0	246	2.0	NaN
25	NaN	164	NaN	0.0
26	3.0	249	4.0	0.0
27	3.0	157	NaN	0.0
28	5.0	999	NaN	1.0
29	5.0	999	NaN	NaN
30	NaN	999	7.0	1.0
31	NaN	999	5.0	NaN
32	NaN	240	NaN	0.0
33	NaN	999	NaN	NaN
34	NaN	209	6.0	NaN
35	4.0	999	NaN	0.0
36	7.0	999	7.0	0.0
37	NaN	222	NaN	0.0
38	2.0	254	NaN	1.0
39	NaN	999	3.0	0.0

DATA CLEANING

In [78]:

```
df.shape
```

Out[78]: (30100, 9)

In [79]:

```
# check for null values
df.isnull().sum()
```

Out[79]:

```
Employee_ID                 0
Age                     15075
Department               3912
Satisfaction_Level          0
Last_Evaluation             0
Projects                15052
Average_Monthly_Hours       0
Years_at_Company        15056
Left                    10005
dtype: int64
```

In [81]:

```
# calaculate the percentage of missing values for each column
missing_counts = df.isnull().sum()
missing_percent = (missing_counts / len(df)) * 100
missing_percent = missing_percent.round(1)
missing_percent
```

Out[81]:
```
Employee_ID              0.0
Age                     50.1
Department              13.0
Satisfaction_Level       0.0
Last_Evaluation          0.0
Projects                50.0
Average_Monthly_Hours    0.0
Years_at_Company        50.0
Left                    33.2
dtype: float64
```

In [82]:
```
# handle missing values
# Normalize text  in colum Departments, remove spaces and ensure the case is lower
df['Department']=df['Department'].str.lower().str.strip()
df['Department']= df['Department'].fillna('unknown')
```

In [83]:
```
df.columns
```

Out[83]:
```
Index(['Employee_ID', 'Age', 'Department', 'Satisfaction_Level',
       'Last_Evaluation', 'Projects', 'Average_Monthly_Hours',
       'Years_at_Company', 'Left'],
      dtype='object')
```

In [84]:
```
# fill missing values in numerical columns with meadian
numerical_cols= ['Age','Projects','Years_at_Company', 'Left']
for col in numerical_cols:
    median_value = df[col].median()
    df[col]= df[col].fillna(median_value)
```

In [85]:
```
df.isnull().sum()
```

Out[85]:
```
Employee_ID              0
Age                      0
Department               0
Satisfaction_Level       0
Last_Evaluation          0
Projects                 0
Average_Monthly_Hours    0
Years_at_Company         0
Left                     0
dtype: int64
```

1. For Department, I chose to fill missing values with "unknown" rather than dropping rows.

   This preserves all employees in the dataset (~13% had missing departments) and allows analysis of how employees with unknown departments compare to others.

   Dropping rows would have removed valuable data and reduced the dataset unnecessarily.

2. For numerical columns (Age, Projects, YearsAtCompany, Left), I fill missing values with the median.

This is because the median is not affected by outliers, and it maintains the central tendency of the data without introducing bias from extreme values.

In [86]:
```
df.describe()
```

Out[86]:

	Employee_ID	Age	Satisfaction_Level	Last_Evaluation	Projects	Average_
count	30100.000000	30100.000000	30100.000000	30100.000000	30100.000000	
mean	501459.486844	41.021528	0.500229	0.500954	4.751595	
std	289871.790946	7.925980	0.287933	0.290158	1.232629	
min	1005.000000	22.000000	0.000000	0.000000	2.000000	
25%	250675.750000	41.000000	0.250000	0.250000	5.000000	
50%	500246.500000	41.000000	0.500000	0.500000	5.000000	
75%	754074.000000	41.000000	0.750000	0.760000	5.000000	
max	999999.000000	60.000000	1.000000	1.000000	7.000000	

- Average_Monthly_Hours has unrealistic values (e.g., 999 hours), which exceed the maximum possible monthly hours (~730). These could be placeholders for missing or invalid data and would distort analysis and model performance.

  Fix: Cap values above 400 hours at 200 hours, a realistic monthly workload baseline (≈45 hours/week × 4.3 weeks/month). This ensures the values within humanly possible limits without introducing artificial averages, preserving data integrity.

- Age (22–60 years): No outliers detected; all values fall within a realistic working-age range.
- Satisfaction_Level & Last_Evaluation (0.0–1.0): Values fall within expected normalized limits.
- Projects (2–7 projects): Range reflects a reasonable employee workload.
- Years_at_Company (1–10 years): Values fall within a typical tenure range.

In [87]:
```
df.loc[df['Average_Monthly_Hours'] > 300, 'Average_Monthly_Hours']=200
```

In [88]:
```
print("\nOutliers capped using human workload baseline. ")
```

```
Outliers capped using human workload baseline.
```

In [89]:
```
df.duplicated().value_counts()
```

Out[89]:
```
False    30000
True       100
Name: count, dtype: int64
```

In [90]:
```
#Remove Exact Duplicate Rows
df = df.drop_duplicates()
print("\nDuplicate rows removed .")
```

```
Duplicate rows removed .
```

In [91]:
```
#Ensure Unique Employee_IDs
# Remove duplicate Employee_IDs (keep first valid record)
df = df.drop_duplicates(subset=["Employee_ID"], keep="first")
```

In [92]:
```
df.duplicated().value_counts()
```

Out[92]:
```
False    29524
Name: count, dtype: int64
```

In [93]:
```
df.duplicated().sum()
```

Out[93]:
```
np.int64(0)
```

In [94]:
```
df.duplicated(subset= ['Employee_ID']).sum()
```

Out[94]:
```
np.int64(0)
```

In [95]:
```
df.columns
```

Out[95]:
```
Index(['Employee_ID', 'Age', 'Department', 'Satisfaction_Level',
       'Last_Evaluation', 'Projects', 'Average_Monthly_Hours',
       'Years_at_Company', 'Left'],
      dtype='object')
```

In [96]:
```
df= df.rename(columns={
    'Employee_ID': 'EmployeeID',
    'Age': 'Age',
    'Department': 'Department',
    'Satisfaction_Level': 'SatisfactionScore',
    'Last_Evaluation': 'LastEvaluationScore',
    'Projects': 'NumProjects',
    'Average_Monthly_Hours': 'AvgMonthlyHours',
    'Years_at_Company': 'YearsAtCompany',
    'Left': 'Attrition'}
)
```

In [97]:
```
df.dtypes
```

Out[97]:
```
EmployeeID               int64
Age                    float64
Department              object
SatisfactionScore      float64
LastEvaluationScore    float64
NumProjects            float64
AvgMonthlyHours          int64
YearsAtCompany         float64
Attrition              float64
dtype: object
```

After checking the data types:

- Convert 'Age', 'NumProjects', and 'YearsAtCompany' from float to integer because these values are whole numbers.
- Convert 'Attrition' to integer to clearly represent the binary target.
- Convert 'Department' to categorical to reflect discrete groups and improve memory efficiency.

In [98]:
```
df["Age"] = df["Age"].astype("Int64")
df["NumProjects"] = df["NumProjects"].astype("Int64")
df["YearsAtCompany"] = df["YearsAtCompany"].astype("Int64")
df["Attrition"] = df["Attrition"].astype("Int64")
```

In [99]:
```
df["Department"] = df["Department"].astype("category")
```

In [101…
```
print("\n---DATA QUALITY REPORT  ---")
print("Dataset shape:", df.shape)
print("Employee_ID unique?", df["EmployeeID"].is_unique)
print("\nRemaining missing values:\n", df.isnull().sum())
print("\nDepartment distribution:\n", df["Department"].value_counts())
```

```
---DATA QUALITY REPORT  ---
Dataset shape: (29524, 9)
Employee_ID unique? True

Remaining missing values:
 EmployeeID             0
Age                    0
Department             0
SatisfactionScore      0
LastEvaluationScore    0
NumProjects            0
AvgMonthlyHours        0
YearsAtCompany         0
Attrition              0
dtype: int64

Department distribution:
 Department
finance       7314
hr            7269
unknown       3828
sales         3790
operations    3720
it            3603
Name: count, dtype: int64
```

In [103…

```
## Save the cleaened data as csv
df.to_csv("hr_cleaned_dataset.csv", index=False)
print("\nCleaned HR dataset saved.")
```

```
Cleaned HR dataset saved.
```

In [ ]: