Human Resources Attrition Prediction: Feature Engineering and Modeling

This project demonstrates a structured workflow for predicting employee attrition using HR analytics data. The analysis involves exploratory data analysis (EDA), feature engineering, logistic regression modeling, and performance evaluation.

Tools and Technologies

- **Language**: Python 3.8+
- **IDE/Environment**: Jupyter Notebook or VS Code
- **Core Libraries**:
  - `pandas`, `numpy` – data manipulation
  - `matplotlib`, `seaborn` – visualization
  - `scikit-learn` – modeling, metrics, preprocessing
  - `os`, `datetime`, `warnings` – utilities

Workflow Overview

1. Data Overview & Preprocessing

- **Dataset**: Human Resources employee data (`hr_data.csv`)
- **Tasks**:
  - Data type inspection, summary statistics
  - Missing values and duplicate check
  - One-hot encoding of categorical variables

2. Exploratory Data Analysis (EDA)

- Distribution analysis of:
  - Target variable: `Attrition`
  - Numerical features: e.g., `MonthlyIncome`, `YearsAtCompany`
  - Categorical features: `Department`, `JobRole`, `Overtime`
- Bar plots of feature distribution segmented by attrition outcome

3. Baseline Model

- **Model**: Logistic Regression (using `sklearn.linear_model`)
- **Evaluation**: Accuracy, Precision, Recall, F1-score, ROC-AUC
- **Utilities**:
  - Train-test split (stratified)
  - ROC curve and confusion matrix visualization

4. Feature Engineering

- **Interaction features**:
  - `PromotionStagnationRatio = YearsSinceLastPromotion / (YearsAtCompany + 1)`
  - `OverworkedAndUnhappy` indicator based on domain knowledge
- **Standard scaling** for variables like `MonthlyIncome`
- **Model comparison** after introducing engineered features

5. Feature Selection

- Extract logistic regression coefficients
- Identify top 15 predictors by absolute weight
- Retrain model using selected features for simplicity and interpretability

6. New Feature Experimentation

- Added `CommuteCostIndex = DistanceFromHome / MonthlyIncome`
- Found meaningful lift in attrition prediction, especially in high commute cost segments

---

1. Load libraries and read the data

1.1. Load libraries

In [ ]:
```python
import os
from datetime import datetime
import warnings

# Data manipulation
import pandas as pd
import numpy as np

# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import ConfusionMatrixDisplay

# Machine learning
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (
    classification_report,
    roc_auc_score,
    roc_curve,
    confusion_matrix,
    precision_score,
    recall_score,
    accuracy_score
)

# Optional: Plotly (only if interactive plots are needed later)
# import plotly.offline as py
# import plotly.graph_objs as go
# import plotly.tools as tls
# import plotly.figure_factory as ff

# Suppress warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```python
os.makedirs("output", exist_ok=True)
os.makedirs("data/processed", exist_ok=True)
```

1.2. Dataset Overview

In [ ]:
```python
df = pd.read_csv('hr_data.csv')
```

In [ ]:
```python
def summarize_dataset(df: pd.DataFrame) -> pd.DataFrame:
    numeric_cols = df.select_dtypes(include=np.number).columns.tolist()
    variable_summary = pd.DataFrame({
        "Variable": df.columns,
        "Non-null Count": df.notnull().sum().values,
        "Data Type": df.dtypes.astype(str).values,
    })
    variable_summary["Min"] = variable_summary["Variable"].apply(
        lambda col: df[col].min() if col in numeric_cols else "N/A"
    )
    variable_summary["Max"] = variable_summary["Variable"].apply(
        lambda col: df[col].max() if col in numeric_cols else "N/A"
    )
    os.makedirs("output/1.2", exist_ok=True)
    variable_summary.to_csv("output/1.2/variable_summary.csv", index=False)
    return variable_summary
```

In [ ]:
```python
summary_table = summarize_dataset(df)
```

In [ ]:
```python
display(summary_table)
```

	Variable	Non-null Count	Data Type	Min	Max
0	Age	10000	int64	18	64
1	Gender	10000	object	N/A	N/A
2	Department	10000	object	N/A	N/A
3	JobRole	10000	object	N/A	N/A
4	EducationLevel	10000	int64	1	5
5	YearsAtCompany	10000	int64	0	19
6	YearsSinceLastPromotion	10000	int64	0	9
7	JobSatisfaction	10000	int64	1	10
8	WorkLifeBalance	10000	int64	1	5
9	MonthlyIncome	10000	float64	2000.0	17437.85
10	Overtime	10000	object	N/A	N/A
11	DistanceFromHome	10000	int64	1	49
12	Attrition	10000	int64	0	1
13	PerformanceRating	10000	float64	1.0	4.44

1.3. Data Quality Check

In [ ]:
```
def data_quality_checks(df: pd.DataFrame) -> pd.DataFrame:
    missing_values = df.isnull().sum().sum()
    duplicates = df.duplicated().sum()
    result = pd.DataFrame({
        "Check": ["Missing Values", "Duplicated Rows"],
        "Result": [missing_values, duplicates]
    })
    os.makedirs("output/1.3", exist_ok=True)
    result.to_csv("output/1.3/data_quality_checks.csv", index=False)
    return result
```

In [ ]:
```
quality_report = data_quality_checks(df)
display(quality_report)
```

	Check	Result
0	Missing Values	0
1	Duplicated Rows	0

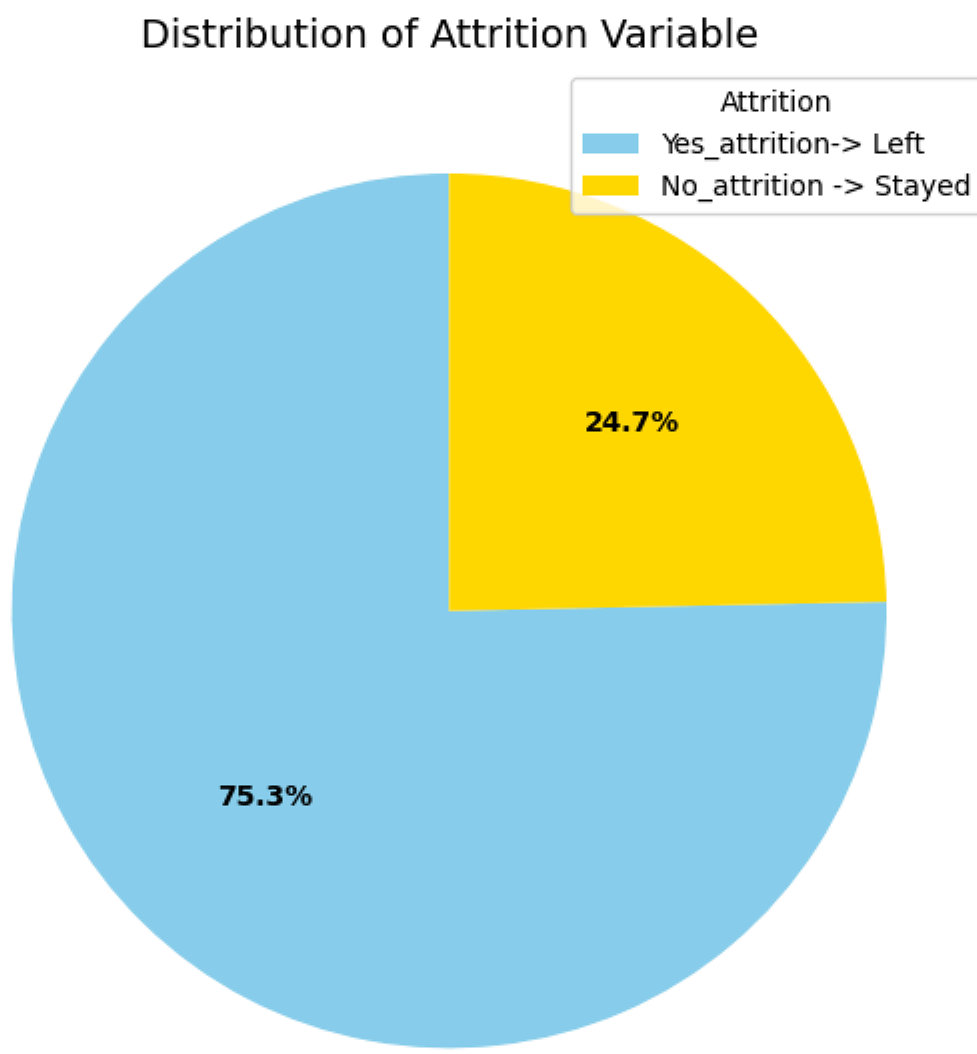
2. Exploratory Data Analysis (EDA)

2.1. Target distribution (number and %)

In [ ]:
```
# Define numeric and categorical variables (excluding target)
numeric_vars = df.select_dtypes(include=np.number).drop(columns=["Attrition", "PerformanceRating"]).columns.tolist()
categorical_vars = df.select_dtypes(include="object").columns.tolist()
```

In [ ]:
```
def plot_target_distribution(df: pd.DataFrame, target_col: str = "Attrition") -> None:
    """
    Plot a pie chart of the target variable with labeled legend and colors.
    """
    counts = df[target_col].value_counts()
    labels = counts.index.map({0: "No_attrition -> Stayed", 1: "Yes_attrition-> Left"})
    colors = ["skyblue", "gold"]

    plt.figure(figsize=(6, 6))
    wedges, texts, autotexts = plt.pie(
        counts, labels=None, autopct='%1.1f%%', startangle=90, colors=colors, textprops={'fontsize': 12}
    )

    # Add custom legend
    plt.legend(wedges, labels, title="Attrition", loc="best")
    plt.setp(autotexts, size=10, weight="bold")
    plt.title("Distribution of Attrition Variable", fontsize=14)
    plt.tight_layout()
    os.makedirs("output/2.1_target_distribution", exist_ok=True)
    plt.savefig("output/2.1_target_distribution/attrition_pie_chart_labeled.png")
    plt.show()
```

In [ ]:
```
plot_target_distribution(df)
```



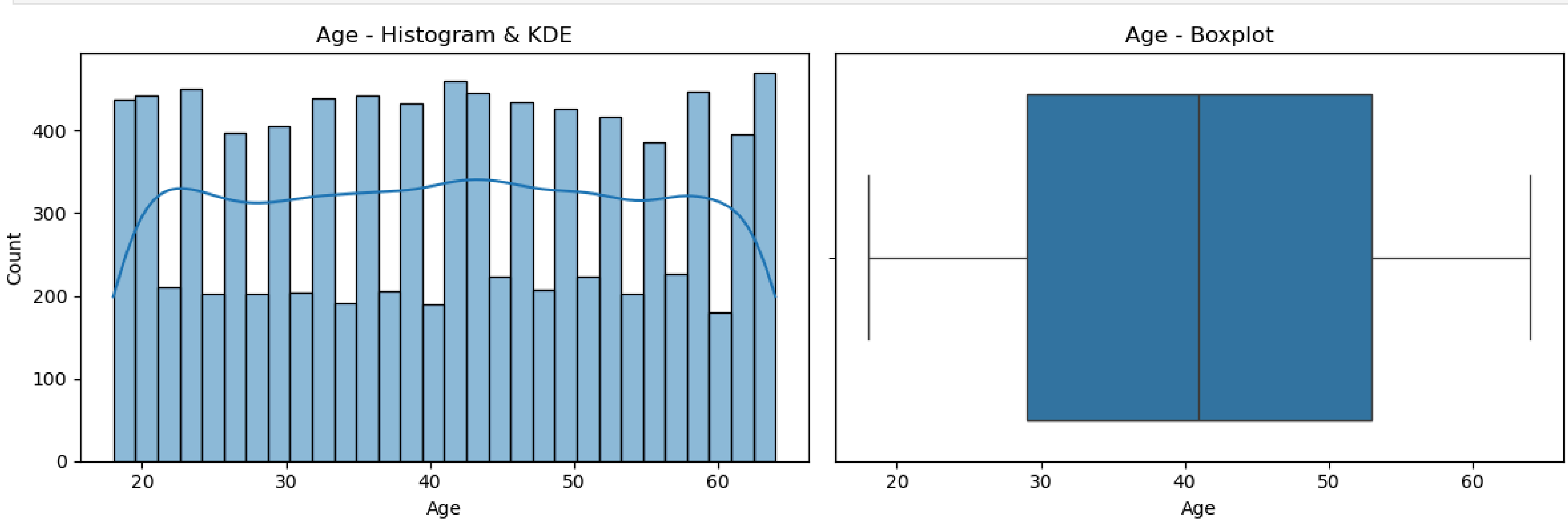
The dataset is moderately imbalanced, with 24.7% of employees having left the company (Yes_attrition) and 75.3% staying (No_attrition).

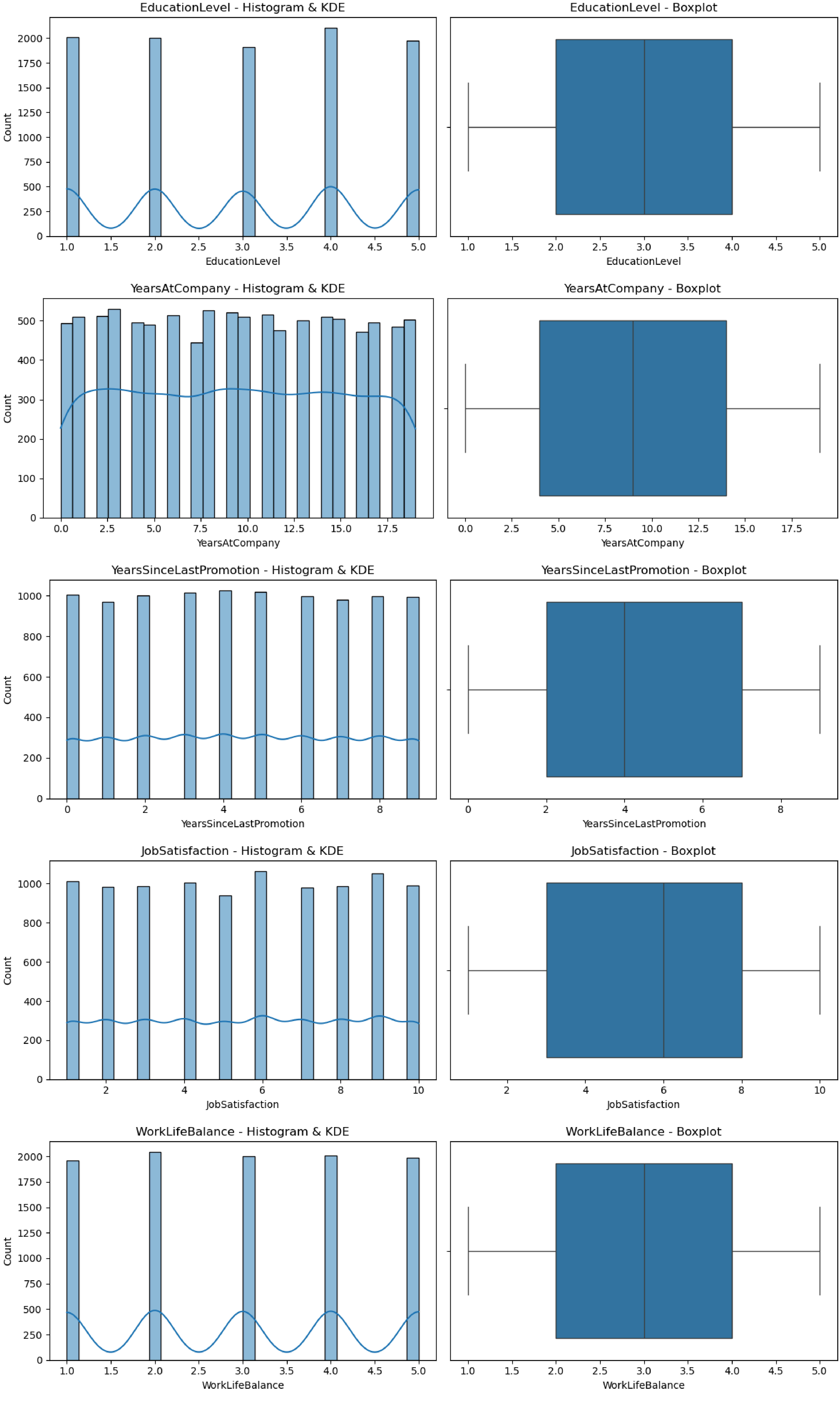
This class imbalance suggests the need for models that emphasize recall and AUC when predicting attrition cases

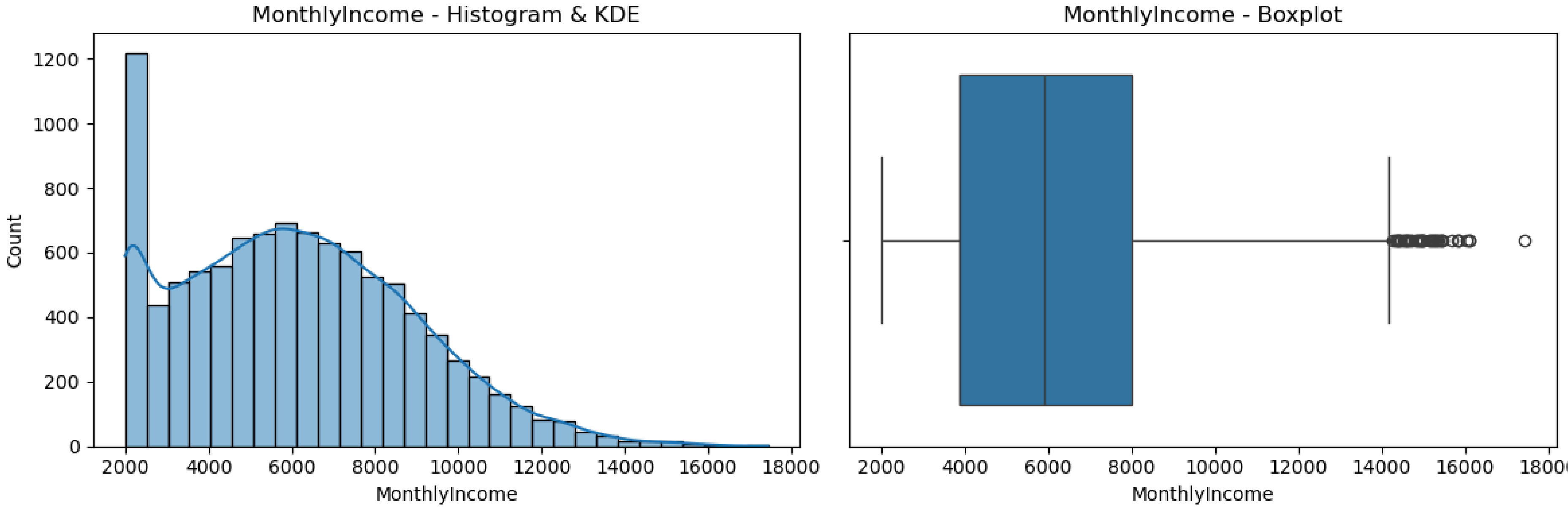
2.2. Numerical variables distribution

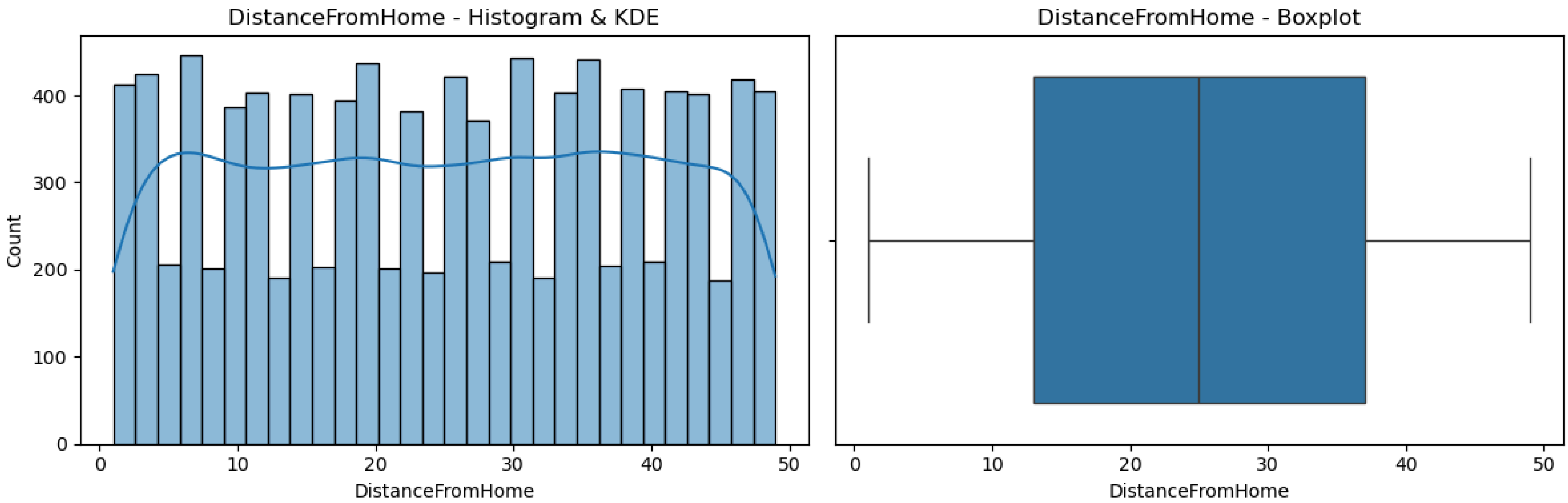
In [ ]:
```
def plot_numeric_distributions(df: pd.DataFrame, num_vars: list, bins: dict = None) -> None:
    """
    Plot histogram, KDE, and boxplot for each numeric variable.
    """
    for var in num_vars:
        plt.figure(figsize=(12, 4))

        # Histogram + KDE
        plt.subplot(1, 2, 1)
        sns.histplot(df[var], kde=True, bins=bins.get(var, 30) if bins else 30)
        plt.title(f"{var} - Histogram & KDE")

        # Boxplot
        plt.subplot(1, 2, 2)
        sns.boxplot(x=df[var])
        plt.title(f"{var} - Boxplot")

        plt.tight_layout()
        os.makedirs("output/2.2_numerical_dis", exist_ok=True)
        plt.savefig(f"output/2.2_numerical_dis/{var}_distribution.png")
        plt.show()
```

In [ ]:
```
# Apply
plot_numeric_distributions(df, numeric_vars)
```




EducationLevel - Histogram & KDE
Count
EducationLevel
EducationLevel - Boxplot
EducationLevel
YearsAtCompany - Histogram & KDE
Count
YearsAtCompany
YearsAtCompany - Boxplot
YearsAtCompany
YearsSinceLastPromotion - Histogram & KDE
Count
YearsSinceLastPromotion
YearsSinceLastPromotion - Boxplot
YearsSinceLastPromotion
JobSatisfaction - Histogram & KDE
Count
JobSatisfaction
JobSatisfaction - Boxplot
JobSatisfaction
WorkLifeBalance - Histogram & KDE
Count
WorkLifeBalance
WorkLifeBalance - Boxplot
WorkLifeBalance



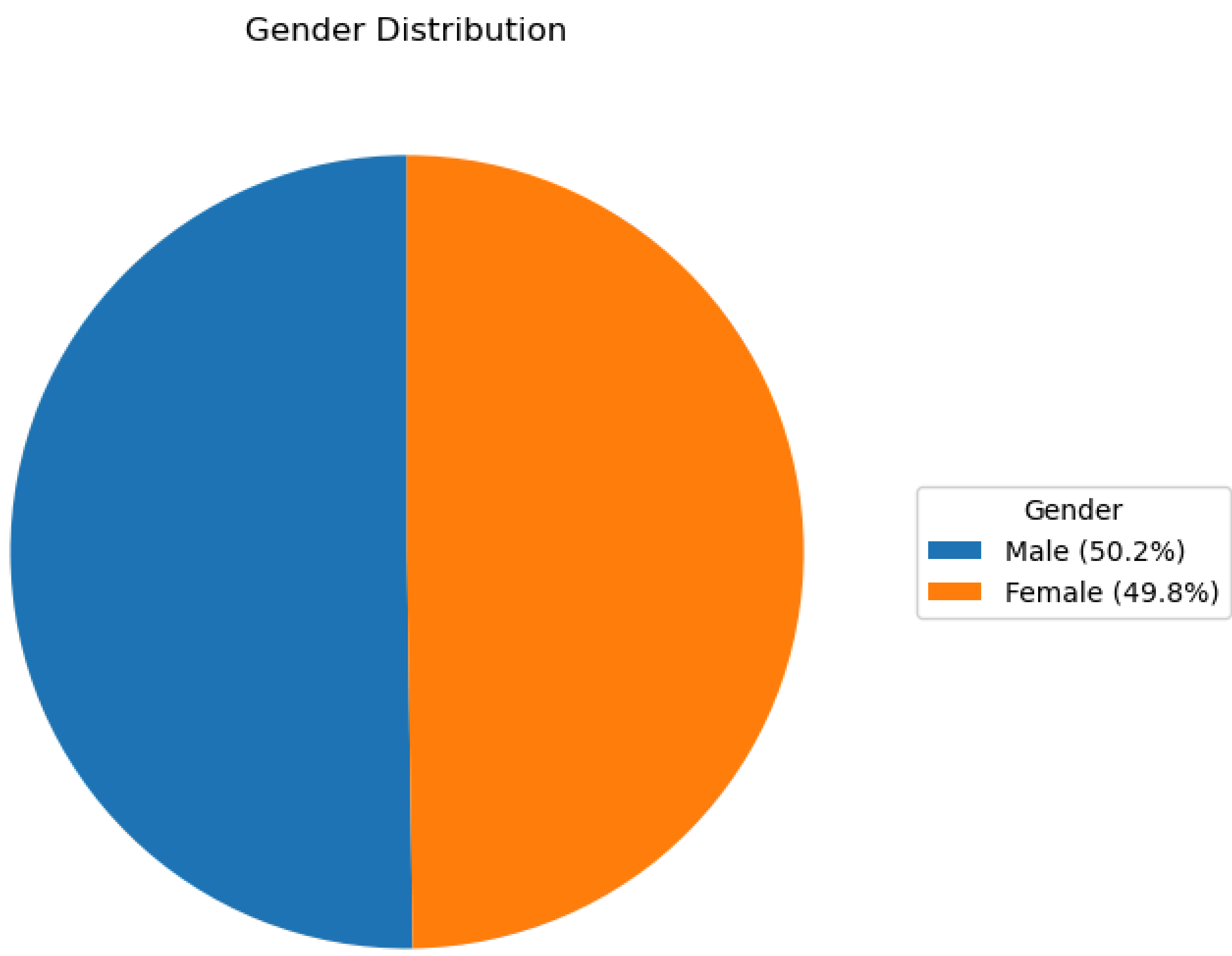


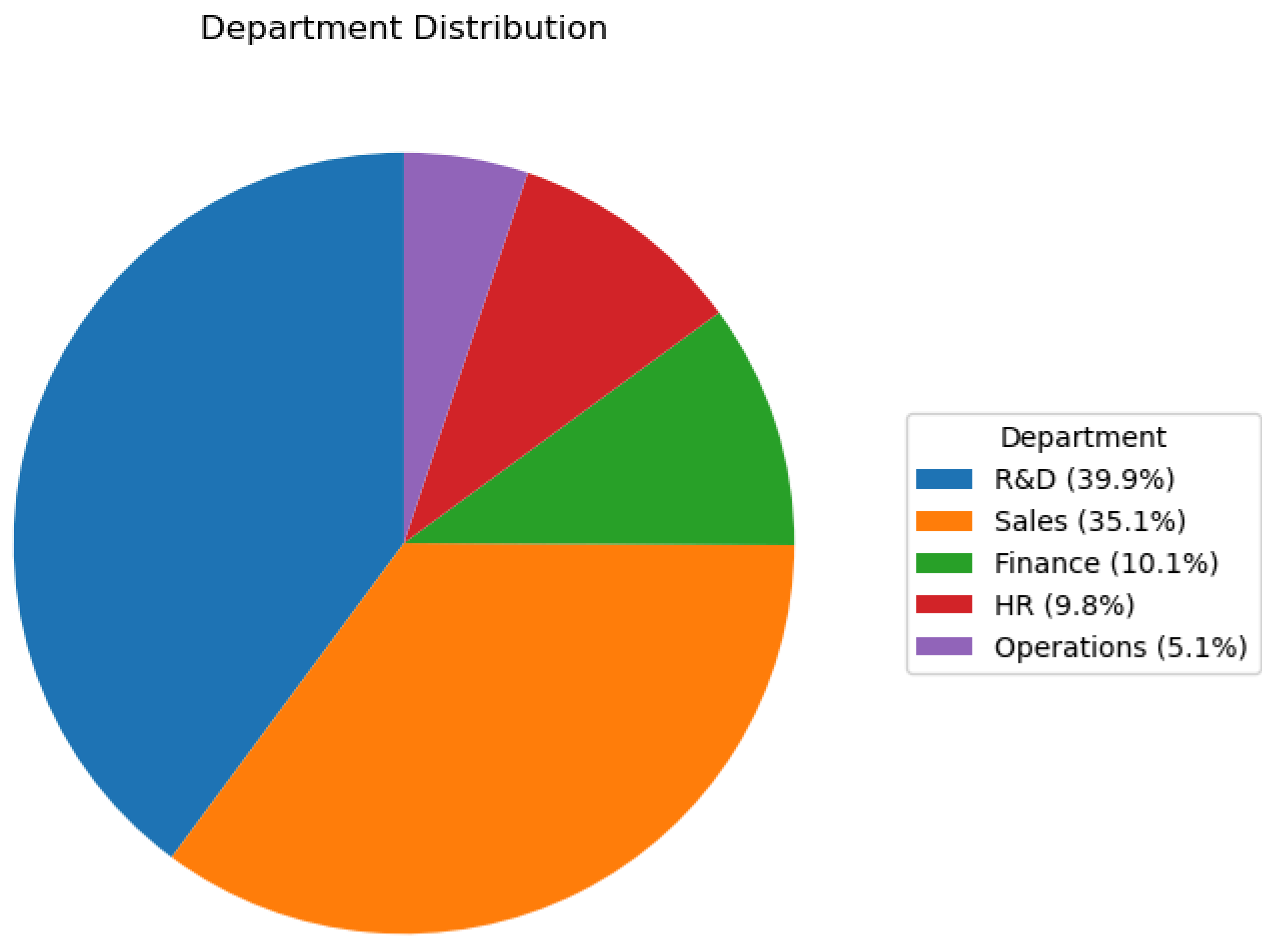
- **MonthlyIncome**: right-skewed, outliers present; median ~6500; common range 2000–8000; consider log/bins
- **Age**: uniform 18–65; no outliers; may bin
- **EducationLevel**: ordinal; uniform 1–5;
- **YearsAtCompany**: uniform 0–19; median ~9; use in tenure-based features
- **YearsSinceLastPromotion**: 0–9; median ~4; no outliers; combine with tenure
- **JobSatisfaction**, **WorkLifeBalance**: ordinal; good for burnout features
- **DistanceFromHome**: uniform 1–48; use for commute cost feature (e.g., Distance/Income)

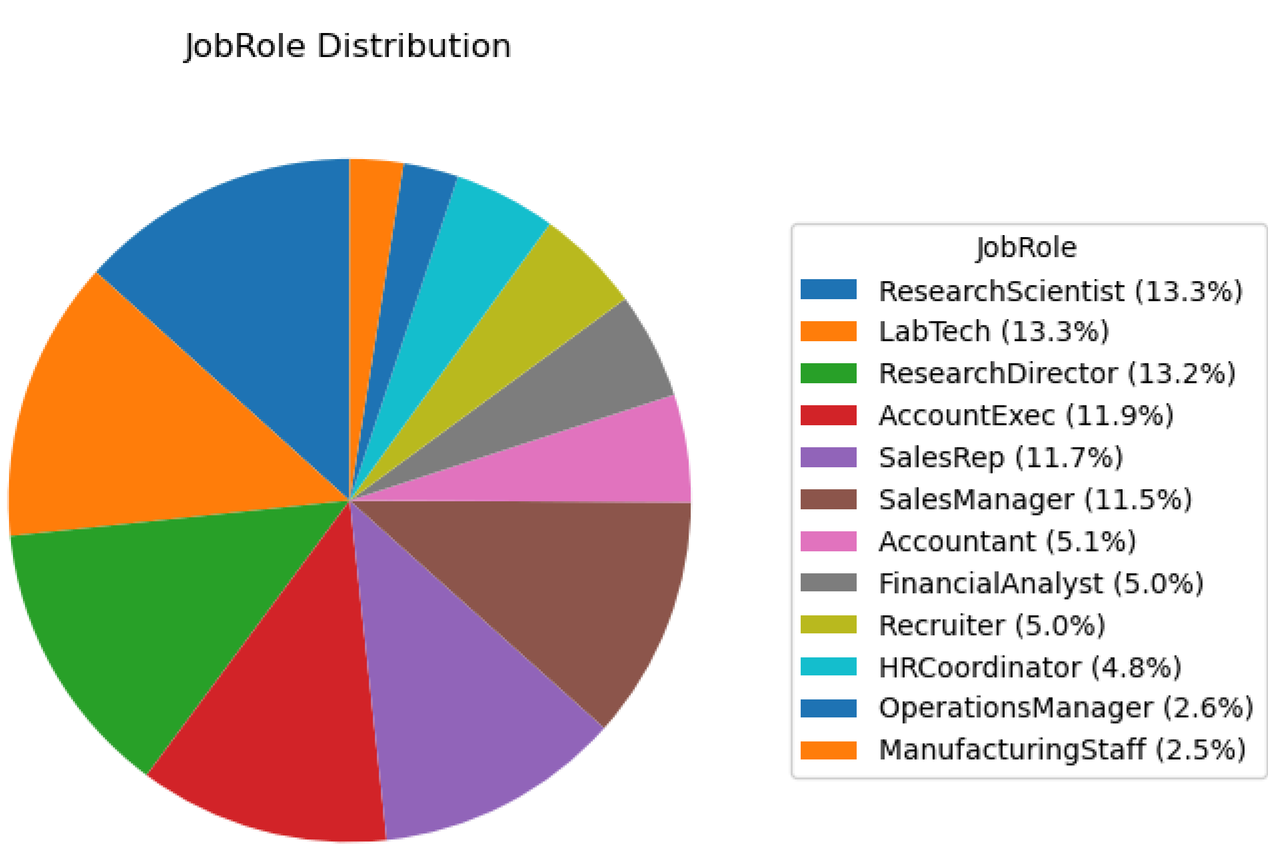
2.3. Categorical variables distribution

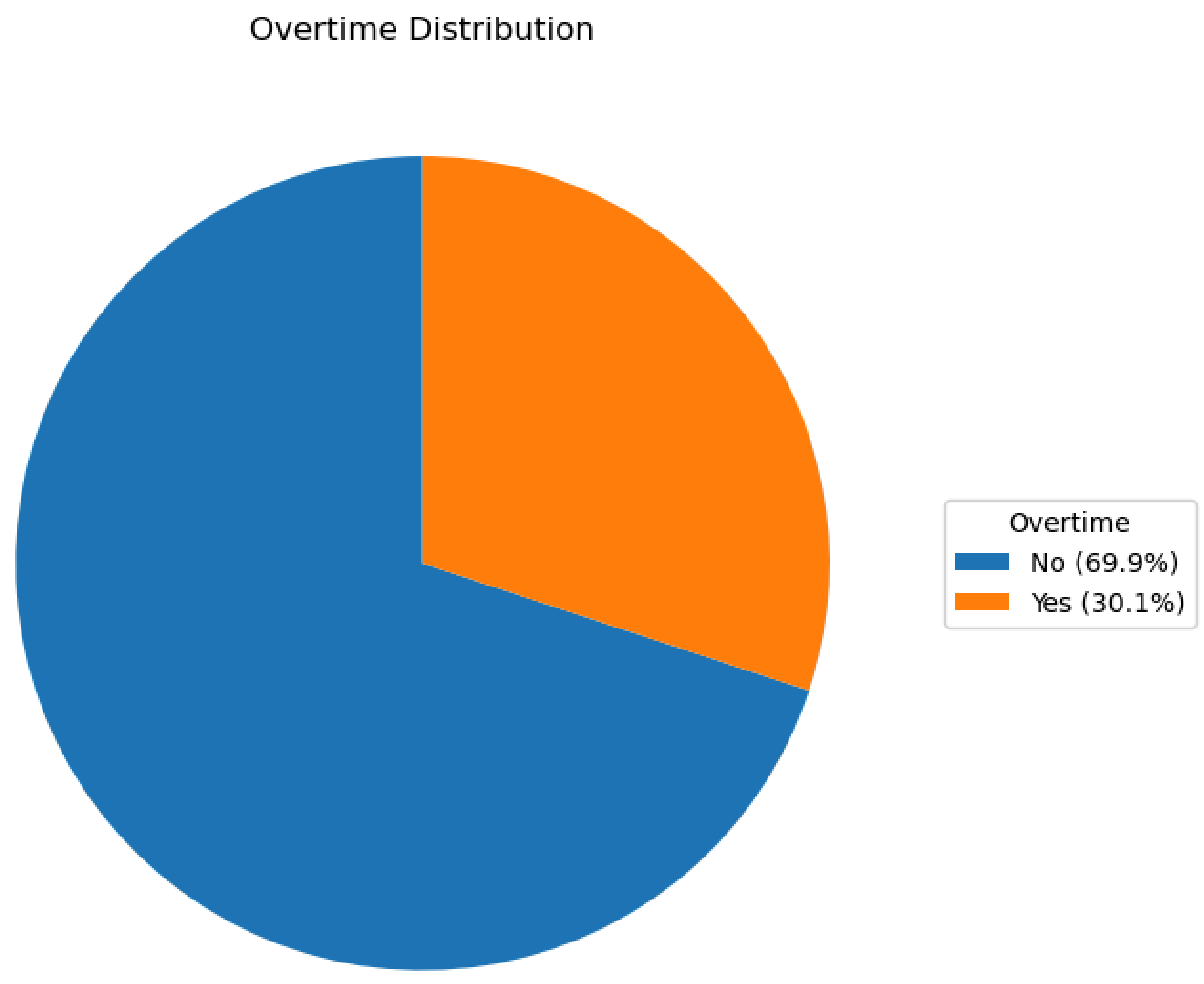
In [ ]:
```python
def plot_categorical_distributions(df: pd.DataFrame, cat_vars: list) -> None:
    """
    Plot full pie charts for categorical variable distributions.
    Labels removed from slices; a legend is added to the right.
    """

    os.makedirs("output/2.3_categorical_dist", exist_ok=True)
    for var in cat_vars:
        counts = df[var].value_counts()
        total = counts.sum()
        labels = [f"{k} ({v/total:.1%})" for k, v in zip(counts.index, counts.values)]

        fig, ax = plt.subplots(figsize=(7, 7))
        wedges, _ = ax.pie(
            counts,
            labels=None,  # no cluttered labels
            startangle=90,
            textprops={"fontsize": 9}
        )

        ax.legend(wedges, labels, title=var, loc="center left", bbox_to_anchor=(1, 0.5))
        ax.set_title(f"{var} Distribution", fontsize=12)
        plt.tight_layout()
        plt.savefig(f"output/2.3_categorical_dist/{var}_pie_chart.png")
        plt.show()
```

In [ ]:
```python
# Apply
plot_categorical_distributions(df, categorical_vars)
```


Gender Distribution
Gender
Male (50.2%)
Female (49.8%)


Department Distribution
Department
R&D (39.9%)
Sales (35.1%)
Finance (10.1%)
HR (9.8%)
Operations (5.1%)


JobRole Distribution
JobRole
ResearchScientist (13.3%)
LabTech (13.3%)
ResearchDirector (13.2%)
AccountExec (11.9%)
SalesRep (11.7%)
SalesManager (11.5%)
Accountant (5.1%)
FinancialAnalyst (5.0%)
Recruiter (5.0%)
HRCoordinator (4.8%)
OperationsManager (2.6%)
ManufacturingStaff (2.5%)



- Gender distribution: No gender bias
    - but need coding into numerical values
- Department Distribution: Dominated by R&D (39.9%) and Sales (35.1%); Smaller groups: Finance, HR, Operations (~25% combined)
    - Consider one-hot encoding or frequency encoding
- JobRole Distribution:
    - Wide spread with 10 distinct roles;
    - Top 4 roles: ResearchScientist, LabTech, ResearchDirector, AccountExec

(each around 11–13%); Bottom roles: Manager and HRCoord (~2–5%)

- Overtime Distribution: No Overtime: 69.9%, Yes: 30.1%

Note:

- R&D: research and development

2.4. Features distribution and barplot (hue = Attrition)

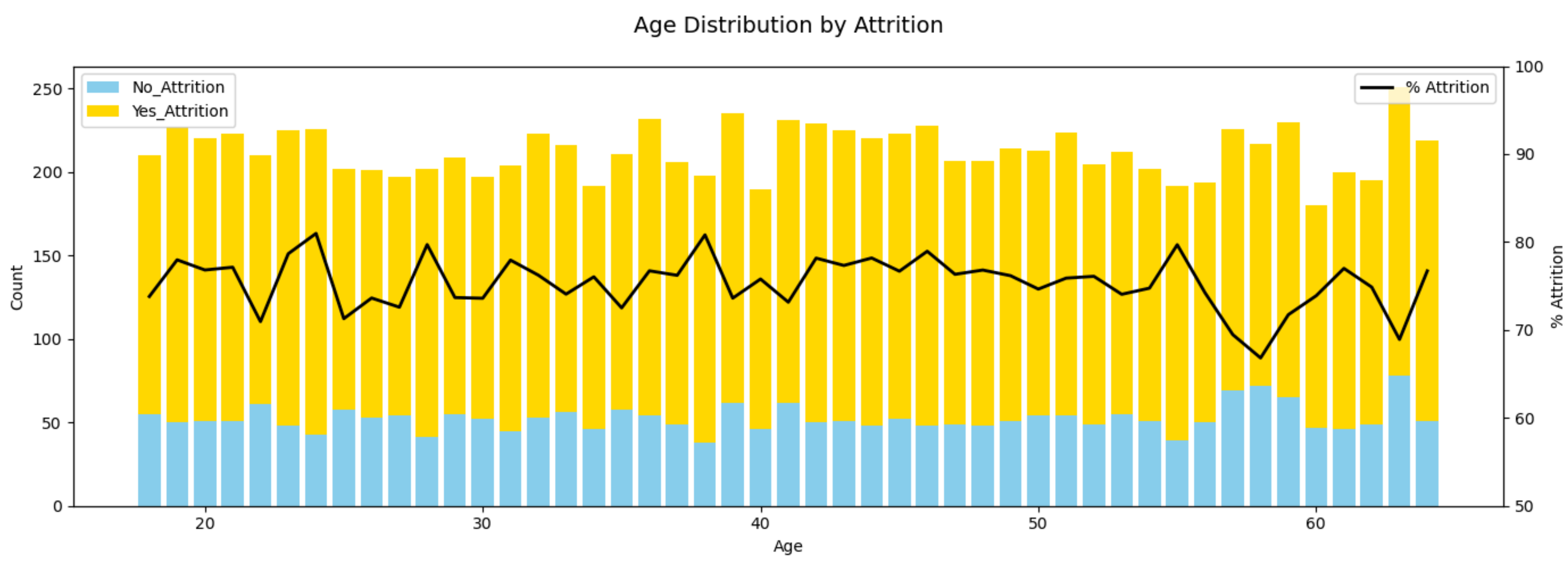
In [ ]:
```
def plot_bar_distributions_by_attrition(
    df: pd.DataFrame,
    num_vars: list,
    target: str = 'Attrition',
    out_dir: str = "output/2.4_numerical_with_attribution",
    y_lim: tuple = (50, 98),
    x_rotation: int = 45
) -> None:
    """
    Plot stacked bar distributions by Attrition with attrition rate line.

    Args:
        df: Input DataFrame
        num_vars: List of numerical feature names
        target: Target variable name (default: 'Attrition')
        out_dir: Output directory for plots
        y_lim: Tuple to fix y-axis limits for attrition rate (right axis)
        x_rotation: Rotation angle for x-axis tick labels
    """
    os.makedirs(out_dir, exist_ok=True)

    for var in num_vars:
        grouped = df.groupby([var, target]).size().unstack(fill_value=0)
        grouped['Total'] = grouped.sum(axis=1)
        grouped['AttritionRate'] = grouped[1] / grouped['Total'] * 100
        grouped = grouped.reset_index().rename(columns={0: 'No_Attrition', 1: 'Yes_Attrition'})
        grouped = grouped.sort_values(by=var)

        fig, ax1 = plt.subplots(figsize=(14, 5))
        ax1.bar(grouped[var], grouped['No_Attrition'], color='skyblue', label='No_Attrition')
        ax1.bar(grouped[var], grouped['Yes_Attrition'], bottom=grouped['No_Attrition'], color='gold', label='Yes_Attrit
        ax1.set_xlabel(var)
        ax1.set_ylabel("Count")
        ax1.tick_params(axis='x', rotation=x_rotation)
```

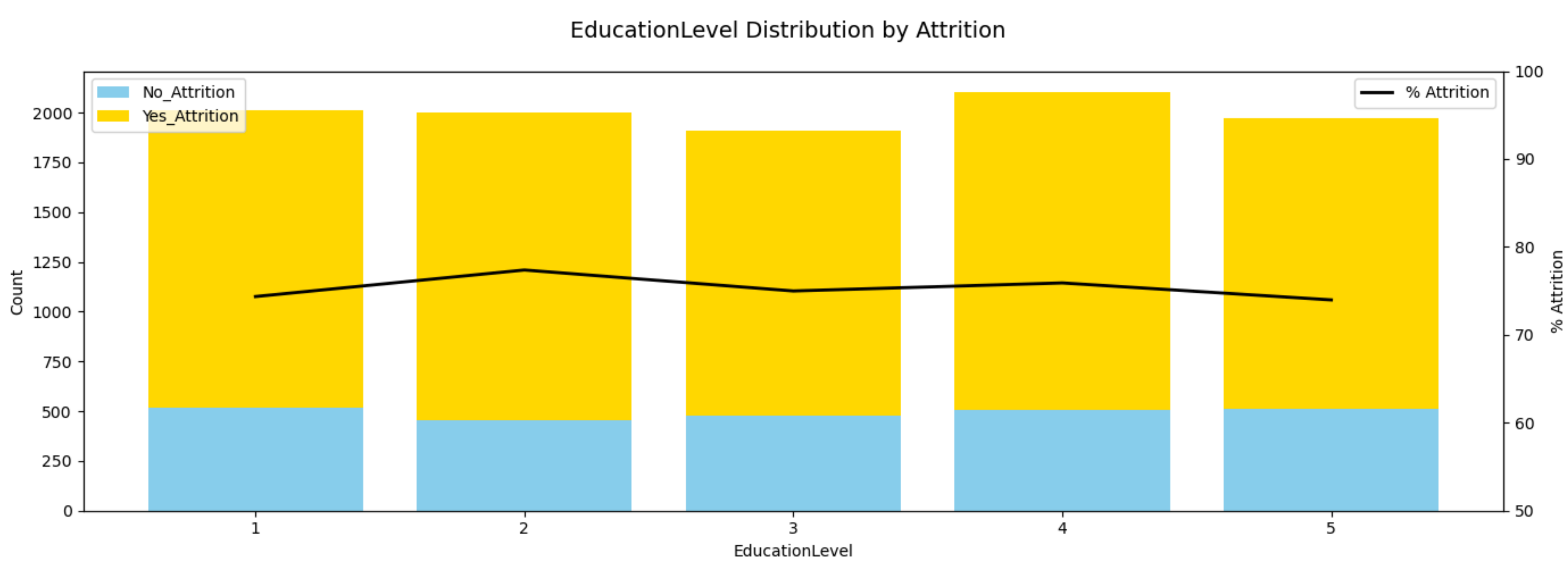
```python
    ax2 = ax1.twinx()
    ax2.plot(grouped[var], grouped['AttritionRate'], color='black', linewidth=2, label='% Attrition')
    ax2.set_ylabel('% Attrition')
    if y_lim:
        ax2.set_ylim(y_lim)

    fig.suptitle(f"{var} Distribution by Attrition", fontsize=14)
    ax1.legend(loc="upper left")
    ax2.legend(loc="upper right")

    plt.tight_layout()
    plt.savefig(os.path.join(out_dir, f"{var}_distribution_plot_stacked.png"))
    plt.show()
```

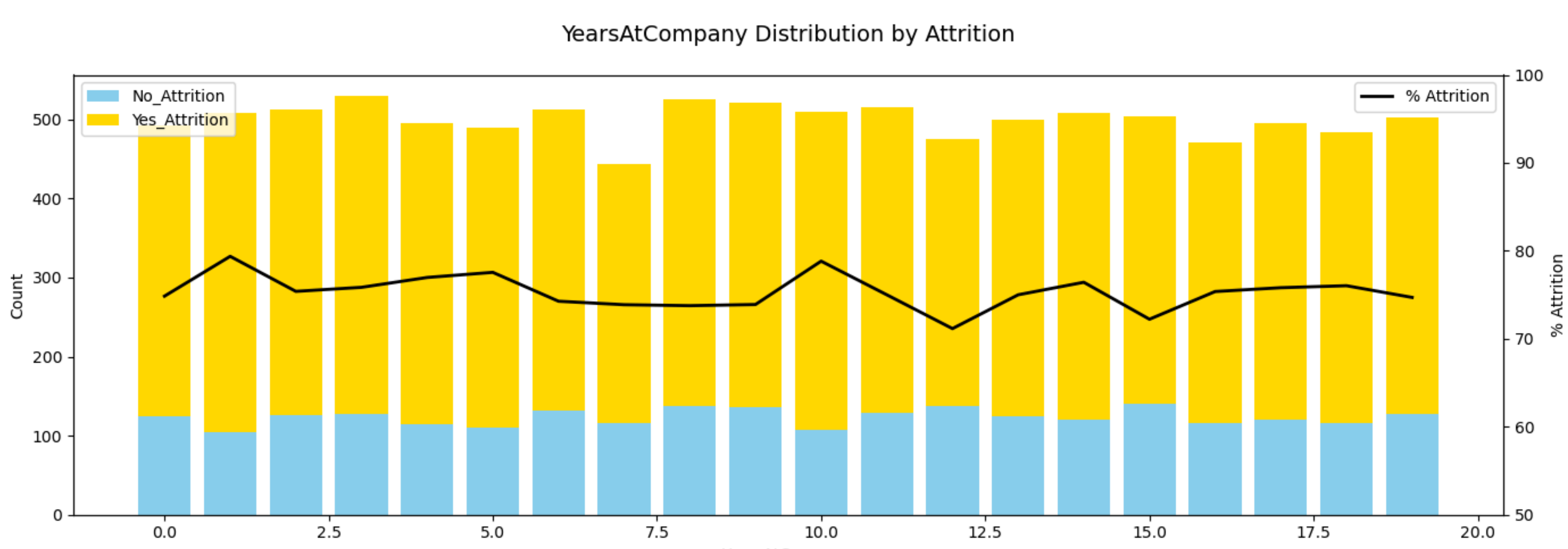
In [ ]:
```python
df = pd.read_csv('hr_data.csv')
```

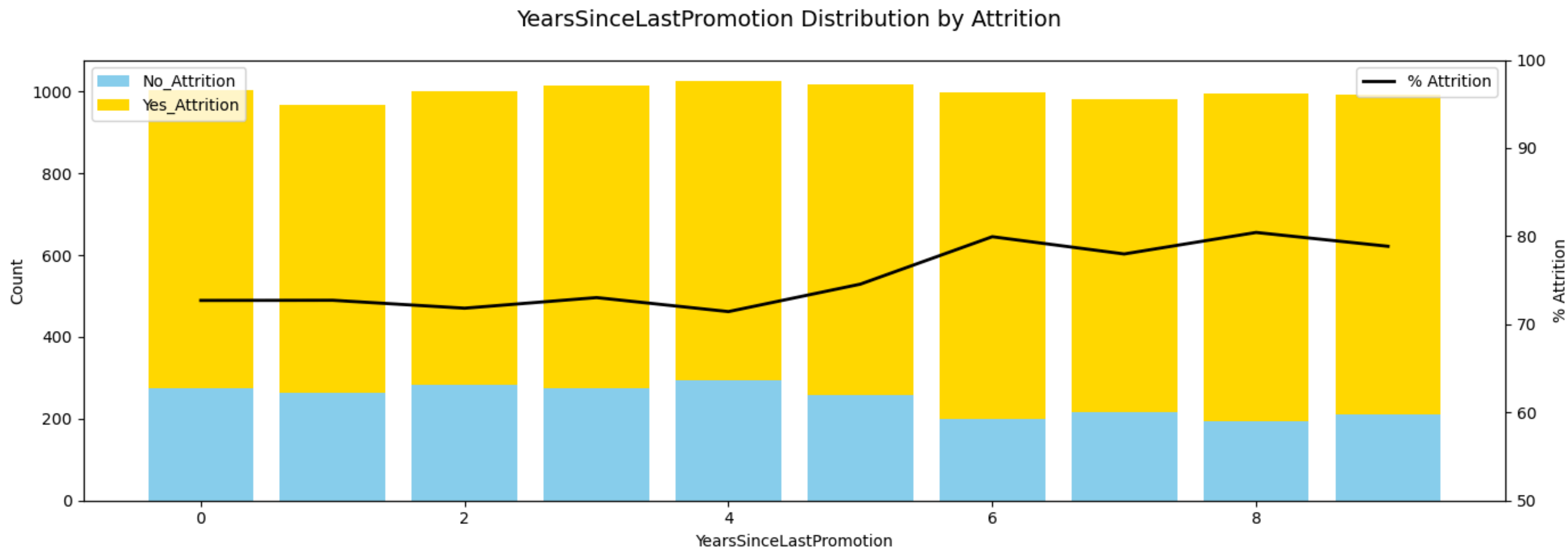
In [ ]:
```python
# Define numeric and categorical variables (excluding target)
numeric_vars_2 = df.select_dtypes(include=np.number).drop(columns=["Attrition", "PerformanceRating", "MonthlyIncome"]).
categorical_vars = df.select_dtypes(include="object").columns.tolist()
```

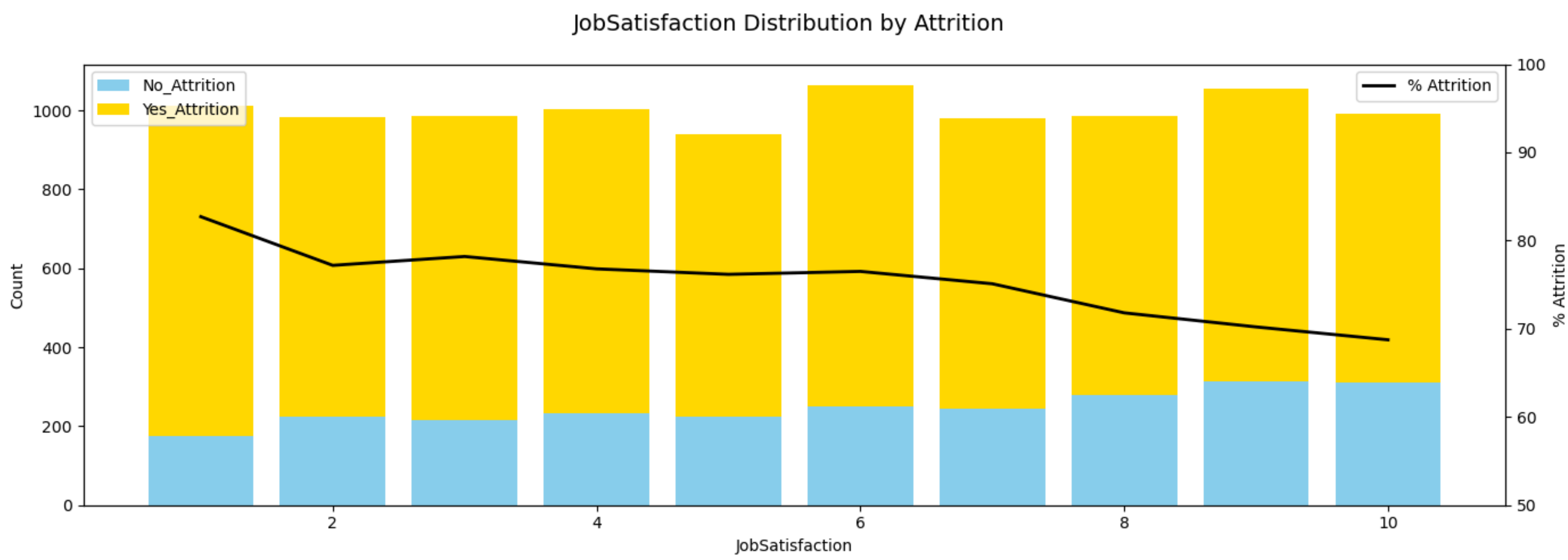
In [ ]:
```python
# Apply
plot_bar_distributions_by_attrition(
    df=df,
    num_vars = numeric_vars_2,
    out_dir="output/2.4_numerical_with_attribution",
    y_lim=(50, 100),
    x_rotation=0  # cleaner for integers or binned vars
)
```

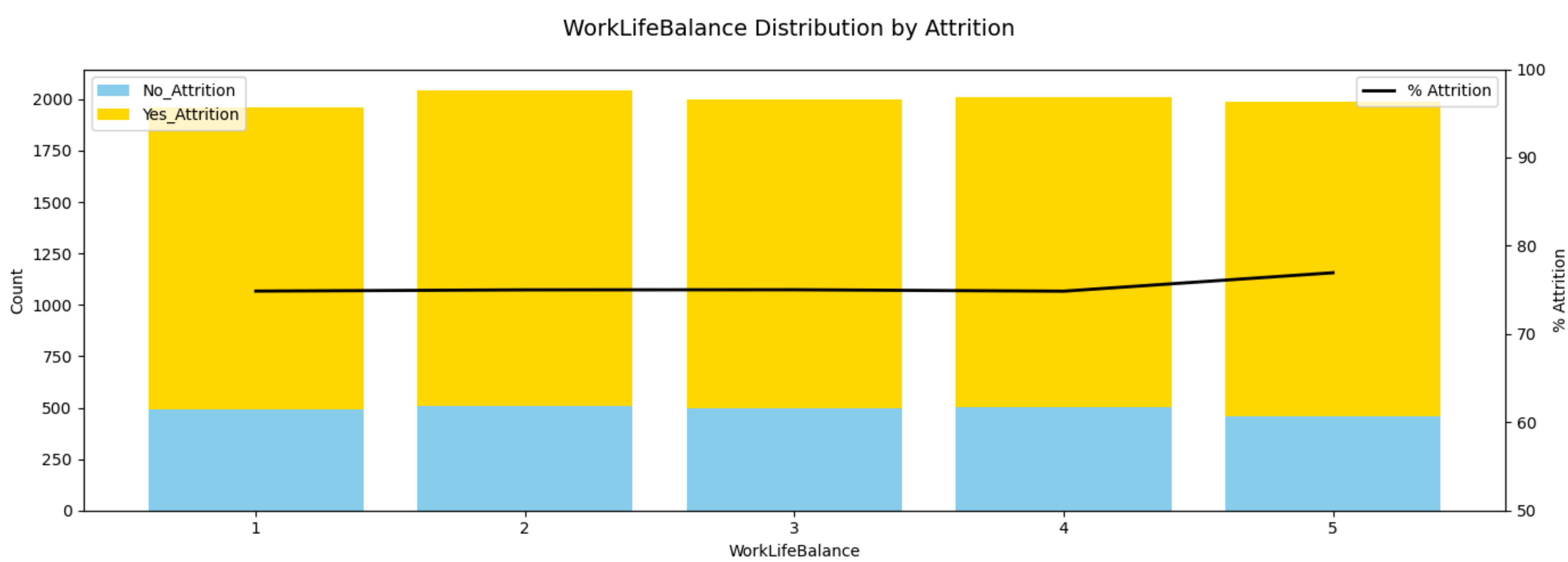


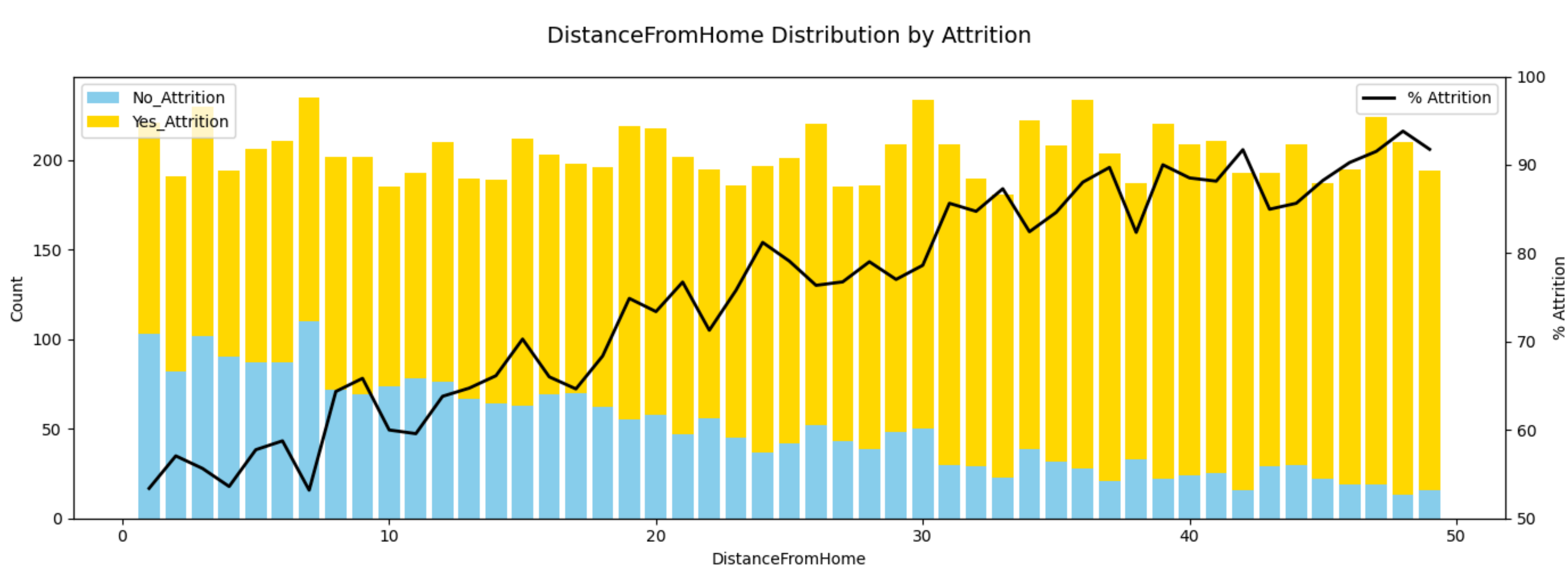










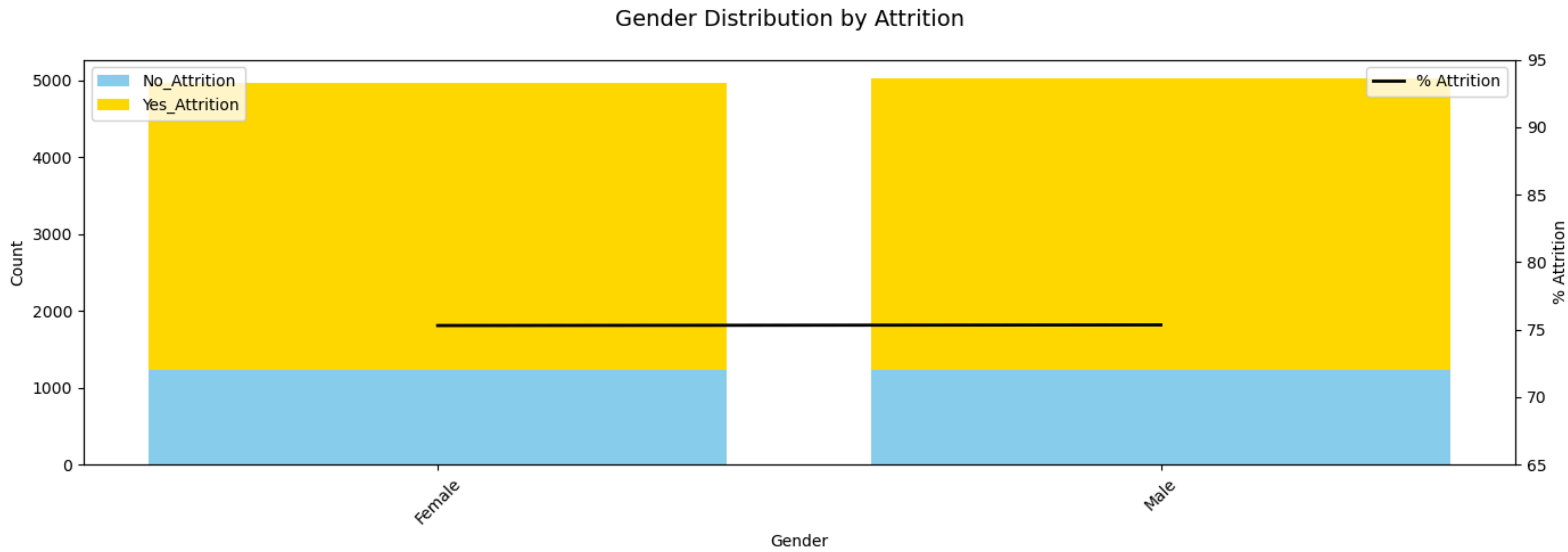


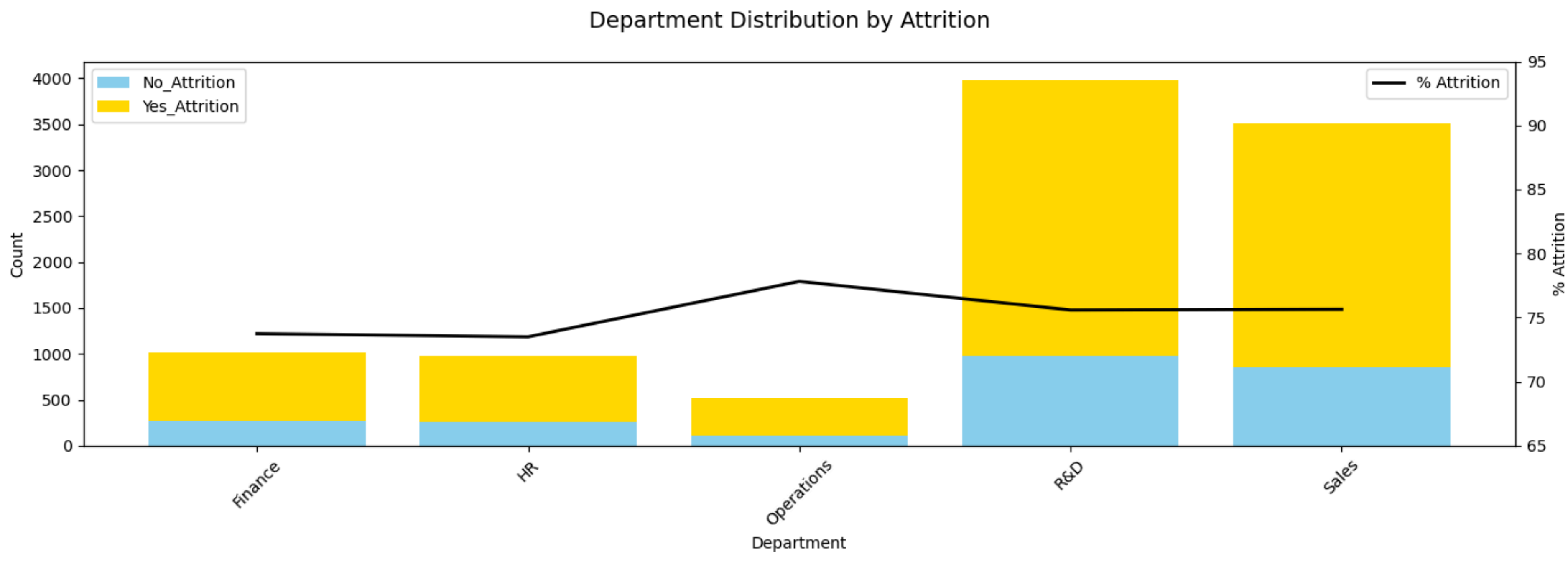
Variable	Insight Summary
Age	Attrition is relatively flat, slightly higher in younger groups.
EducationLevel	Rate is stable around 75%, not strongly related to education level.
YearsAtCompany	Early leavers (<3 yrs) spike; attrition dips slightly after ~12 years.
YearsSinceLastPromotion	Longer time since promotion (5+ yrs) correlates with higher attrition.
JobSatisfaction	Clear negative correlation: higher satisfaction → lower attrition; Key values:3
WorkLifeBalance	Overall flat; small drop in attrition at level 4.

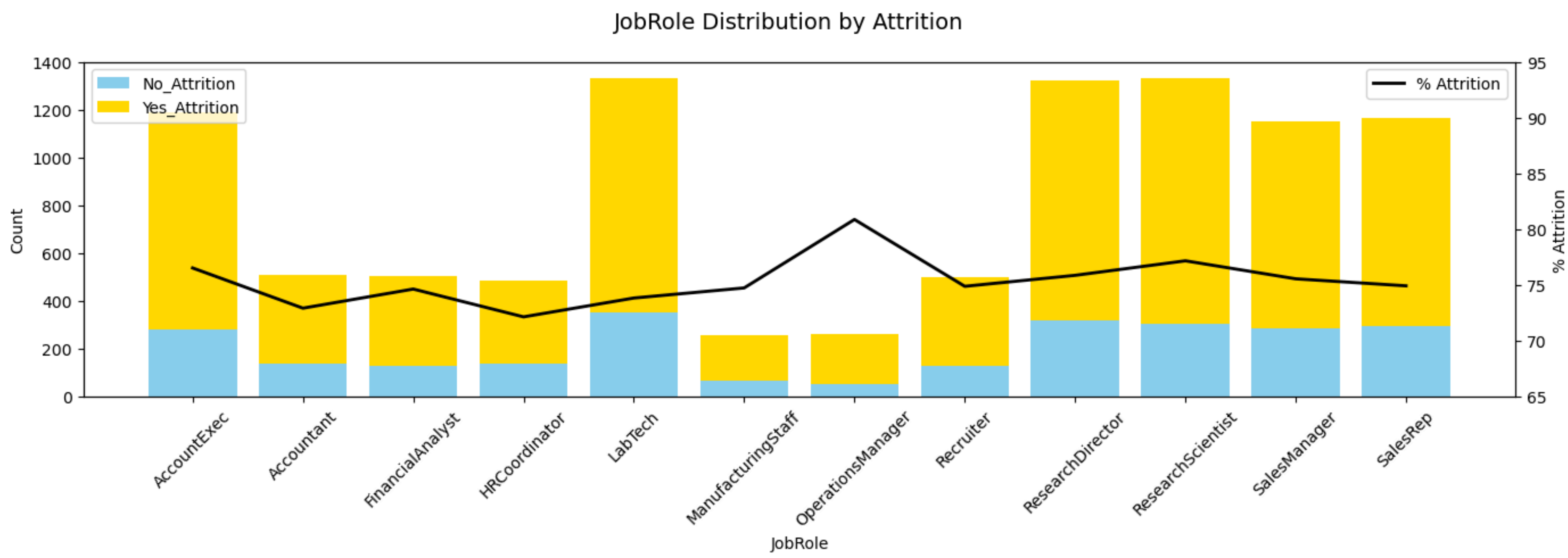
Variable	Insight Summary
DistanceFromHome	Strong upward trend: longer distance → higher attrition risk.
MonthlyIncome	Not interpretable without binning. Need transformation for future use.

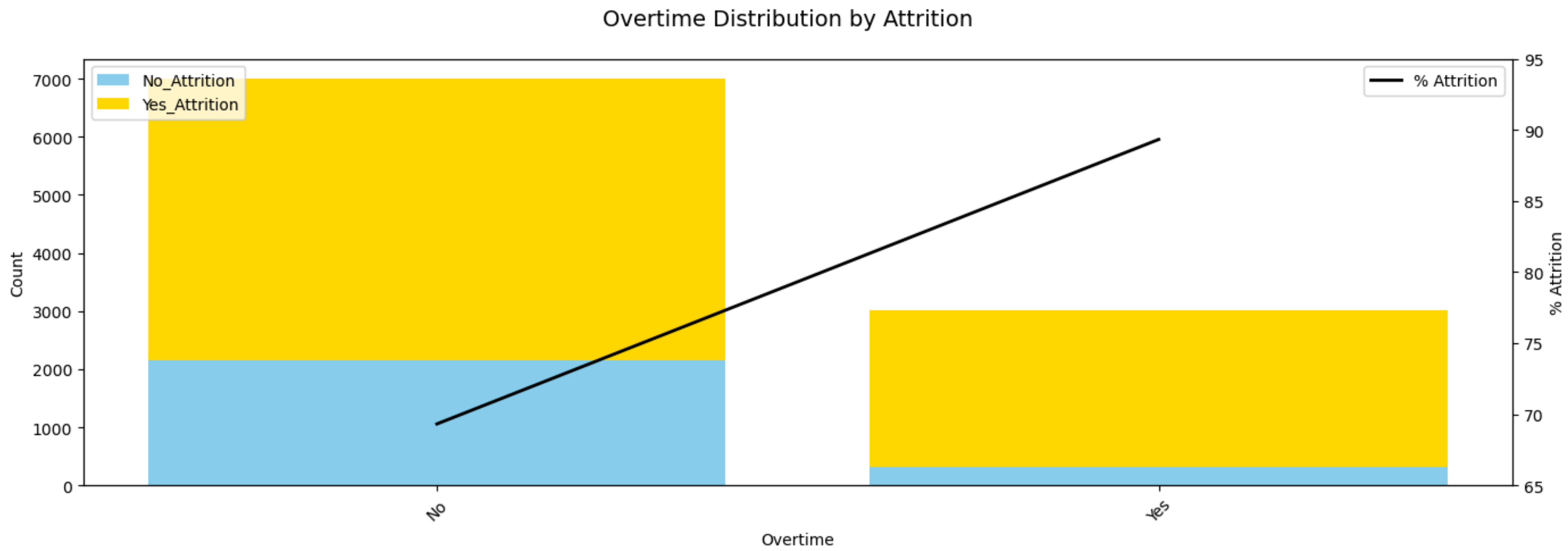
2.5. Bar plot for other varaibles (hue = Attrition)

In [ ]:
```
# Apply
plot_bar_distributions_by_attrition(
    df=df,
    num_vars = categorical_vars,
    out_dir="output/2.5_categorical_with_attribution",
    y_lim=(65, 95),
    x_rotation=45  # cleaner for integers or binned vars
)
```









Variable	Summary Insight
Gender	Super slight difference in attrition rate between male and female.
Department	Slight variation; Operations has marginally higher attrition.
JobRole	Roles like Operations Manager & Manufacturing Staff have higher attrition.
Overtime	Strong signal: overtime workers show very high attrition (~88%).

3. Baseline Model

3.1. Define Functions

Timer Utility

In [ ]:
```
from datetime import datetime

def timer(start_time=None):
    if not start_time:
        return datetime.now()
    else:
        elapsed = datetime.now() - start_time
        h, rem = divmod(elapsed.total_seconds(), 3600)
        m, s = divmod(rem, 60)
        print(f"\nTime taken: {int(h)}h {int(m)}m {round(s, 2)}s")
```

Define Features and Target

In [ ]:
```
def define_X_y(df: pd.DataFrame, drop_cols: list, target: str = "Attrition"):
    """
    Drop unused columns and split into X, y.
    """
    X = df.drop(columns=drop_cols)
    y = df[target]
    return X, y
```

Stratified Train-Test Split

In [ ]:
```
def get_train_test_split(X, y, test_size=0.2, stratify=True, random_state=42):
    """
    Perform stratified train-test split.
    """
    stratify_param = y if stratify else None
    return train_test_split(X, y, test_size=test_size, stratify=stratify_param, random_state=random_state)
```

Model Training & Evaluation – Logistic Regression

In [ ]:
```
def train_logistic_model(X_train, X_test, y_train, y_test, model_name="baseline", out_dir="output/models") -> tuple:

    result_path = os.path.join(out_dir, model_name)
    os.makedirs(result_path, exist_ok=True)

    print(f"Training model: {model_name}")
    start = timer()

    model = LogisticRegression(max_iter=1000, solver='liblinear')
    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1]
    timer(start)

    # --- Save predictions
    pred_df = pd.DataFrame({
        "y_true": y_test,
        "y_pred": y_pred,
```

```
        "y_prob": y_prob
    })
    pred_df.to_csv(os.path.join(result_path, "predictions.csv"), index=False)

    # --- Save classification report
    report = classification_report(y_test, y_pred, output_dict=True)
    report_df = pd.DataFrame(report).transpose()
    report_df.to_csv(os.path.join(result_path, "classification_report.csv"))
    print(report_df)

    # --- Save ROC curve
    fpr, tpr, _ = roc_curve(y_test, y_prob)
    auc_score = roc_auc_score(y_test, y_prob)

    plt.figure(figsize=(6, 5))
    plt.plot(fpr, tpr, label=f"AUC = {auc_score:.2f}")
    plt.plot([0, 1], [0, 1], linestyle="--", color="gray")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(f"ROC Curve - {model_name}")
    plt.legend()
    plt.tight_layout()
    plt.savefig(os.path.join(result_path, "roc_curve.png"))
    plt.show()

    # --- Save confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot(cmap="Blues", values_format="d")
    plt.title(f"Confusion Matrix - {model_name}")
    plt.tight_layout()
    plt.savefig(os.path.join(result_path, "confusion_matrix.png"))
    plt.show()

    return model, report_df, auc_score
```

3.2. Basic Data Preprocessing

In [ ]:
```
def preprocess_data(df: pd.DataFrame) -> pd.DataFrame:
    """
    - One-hot encode all object (categorical string) variables, including Gender, Department, JobRole, etc.
    - Drop first dummy to avoid multicollinearity
    """
    # Strip spaces and lowercase for consistency (optional but useful)
    df = df.applymap(lambda x: x.strip().lower() if isinstance(x, str) else x)

    # Identify object-type columns (categorical strings)
    cat_vars = df.select_dtypes(include='object').columns.tolist()

    # One-hot encode
    df = pd.get_dummies(df, columns=cat_vars, drop_first=True)

    return df
```

In [ ]:
```
df = pd.read_csv('hr_data.csv')
```

In [ ]:
```
# Preprocess df
df = preprocess_data(df)
display(df)
```

	Age	EducationLevel	YearsAtCompany	YearsSinceLastPromotion	JobSatisfaction	WorkLifeBalance	MonthlyIncome	DistanceFrom
0	27	2	10	2	4	1	5363.93	
1	31	4	0	7	2	3	7864.56	
2	18	2	7	1	7	4	4171.64	
3	61	5	18	2	5	3	8517.27	
4	49	3	3	3	2	5	2000.00	
...	...	...	...	...	...	...	...	
9995	32	5	11	9	4	1	7221.75	
9996	43	3	10	4	1	2	7423.44	
9997	32	5	19	4	9	2	6545.84	
9998	40	4	19	8	10	5	11162.27	
9999	39	1	17	6	7	2	6710.05	

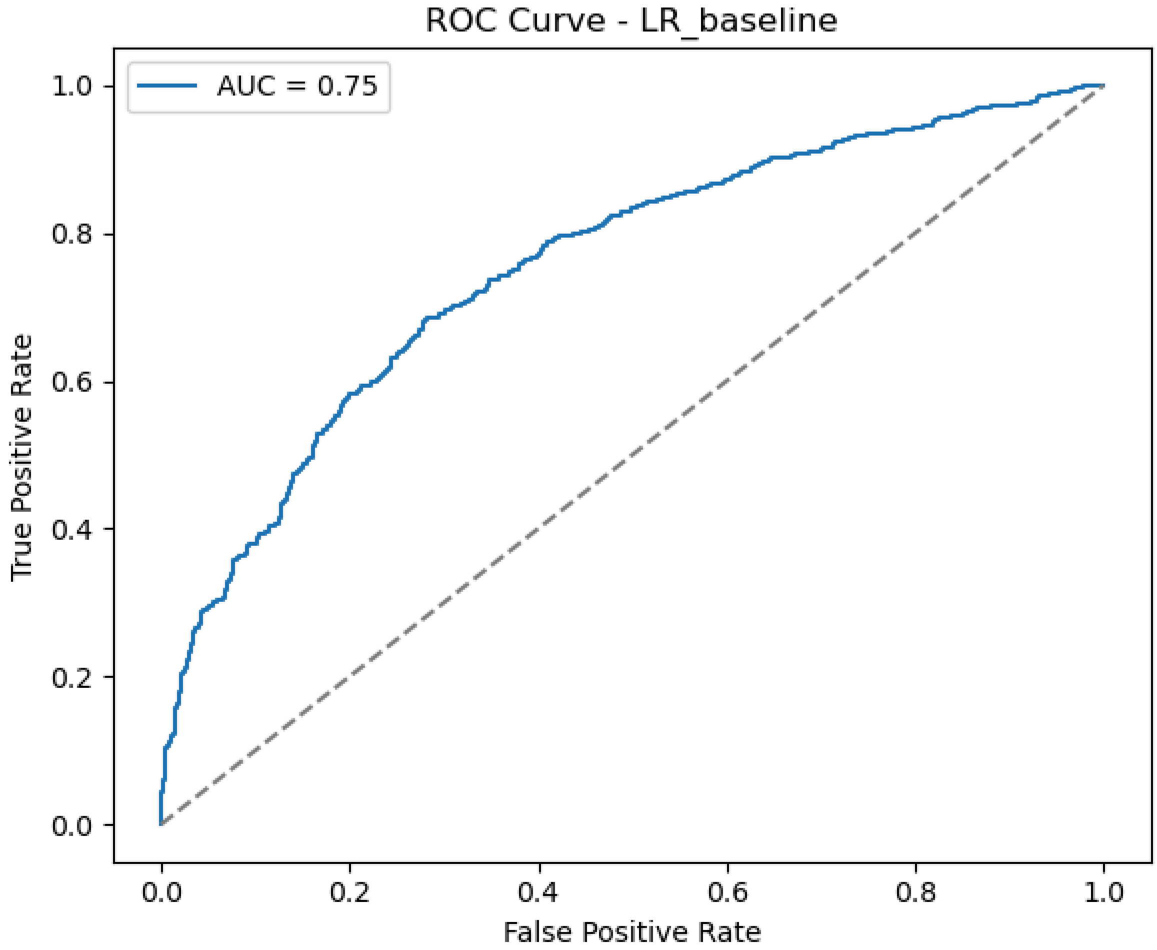
10000 rows × 27 columns

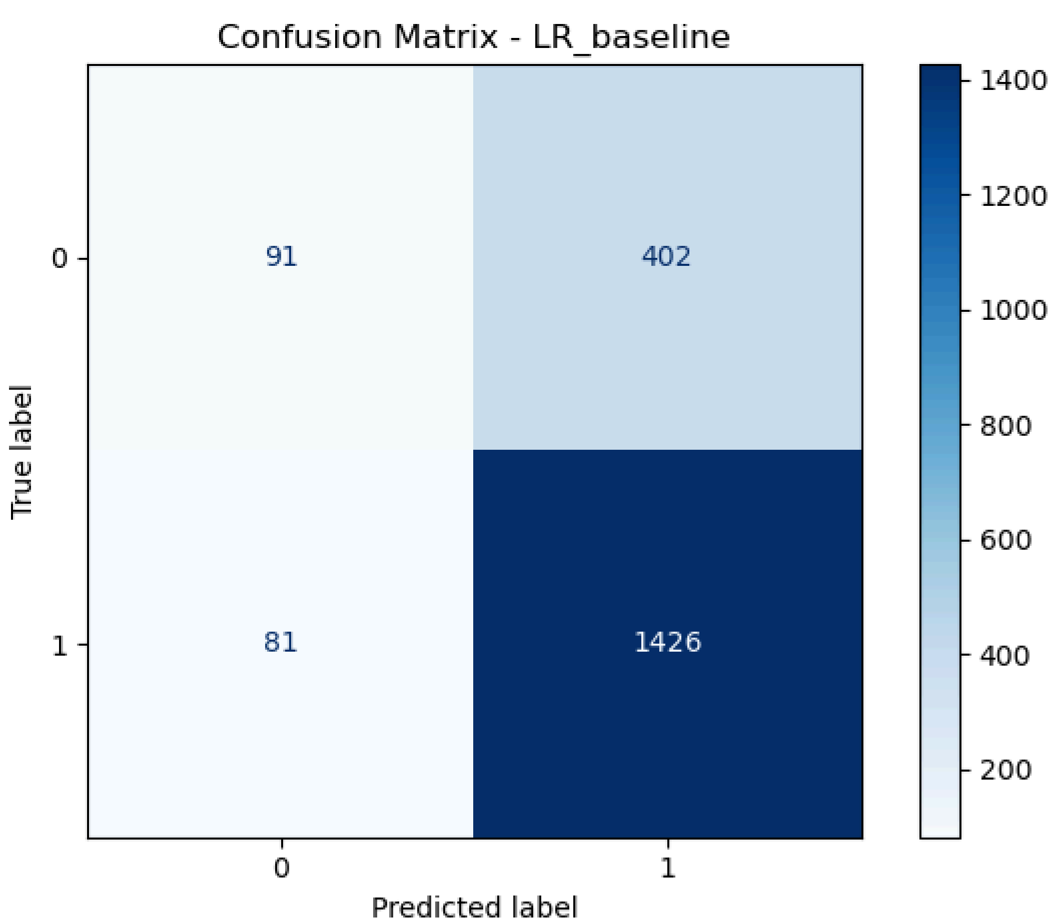
3.3. Baseline Model Training

In [ ]:
```
# Define X, y and scale
drop_cols = ['Attrition']
X, y = define_X_y(df, drop_cols)
```

```python
# Train-test split
X_train, X_test, y_train, y_test = get_train_test_split(X, y)

# Train and evaluate
model, report_df, auc = train_logistic_model(
    X_train, X_test, y_train, y_test,
    model_name="LR_baseline",
    out_dir = "output/3_baseline_model"
)
```

```
Training model: LR_baseline

Time taken: 0h 0m 0.03s
              precision    recall  f1-score    support
0              0.529070  0.184584  0.273684   493.0000
1              0.780088  0.946251  0.855172  1507.0000
accuracy       0.758500  0.758500  0.758500     0.7585
macro avg      0.654579  0.565418  0.564428  2000.0000
weighted avg   0.718212  0.758500  0.711836  2000.0000
```





4. Feature Engineering

1. Scale numeric features (e.g., Monthly Income) if they vary widely
2. Consider interaction terms (e.g., ratio of Years Since Last Promotion to Years at Company)
3. Retrain and note changes in performance

In [ ]:
```python
from sklearn.preprocessing import StandardScaler

def feature_engineering_interaction_and_scaling(df: pd.DataFrame) -> pd.DataFrame:
    """
    Apply scaling to MonthlyIncome and create interaction feature:
    PromotionStagnationRatio = YearsSinceLastPromotion / (YearsAtCompany + 1)
    """
```

```
    df = df.copy()

    # Scale MonthlyIncome
    if 'MonthlyIncome' in df.columns:
        scaler = StandardScaler()
        df['MonthlyIncome_Scaled'] = scaler.fit_transform(df[['MonthlyIncome']])

    # Add interaction feature
    if 'YearsSinceLastPromotion' in df.columns and 'YearsAtCompany' in df.columns:
        df['PromotionStagnationRatio'] = df['YearsSinceLastPromotion'] / (df['YearsAtCompany'] + 1)

    return df
```

In [ ]:
```
# Apply
df = feature_engineering_interaction_and_scaling(df)
df
```

Out[ ]:

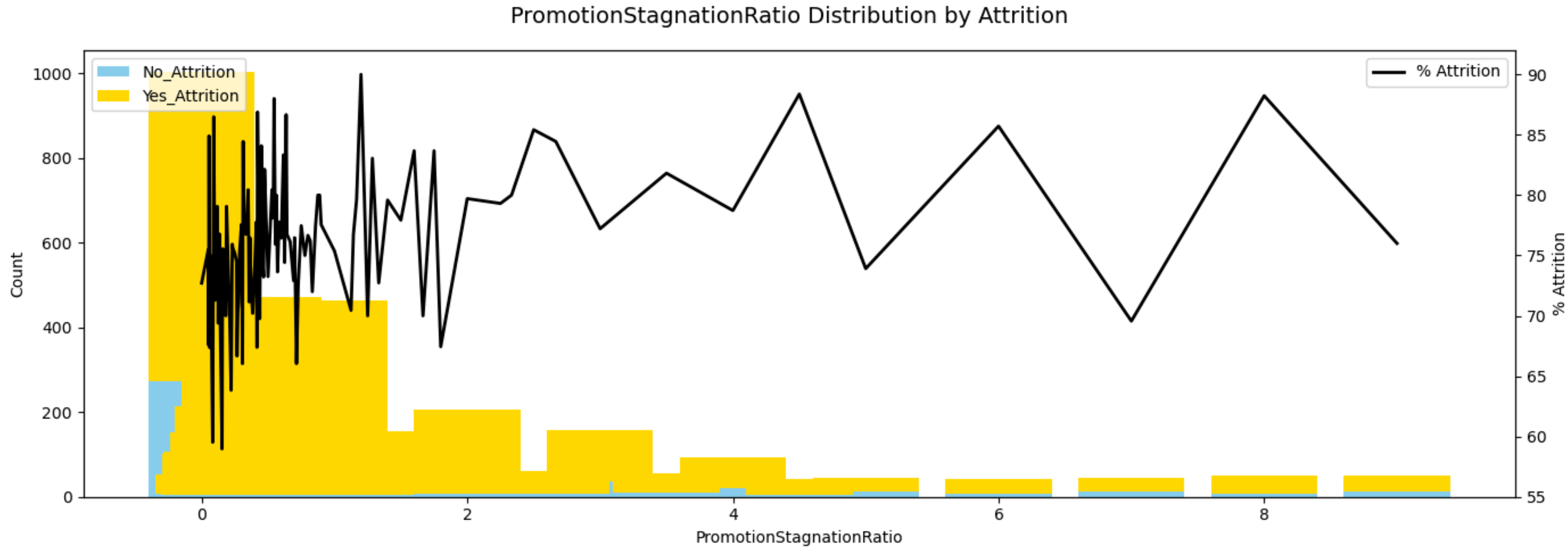
	Age	EducationLevel	YearsAtCompany	YearsSinceLastPromotion	JobSatisfaction	WorkLifeBalance	MonthlyIncome	DistanceFro
0	27	2	10	2	4	1	5363.93	
1	31	4	0	7	2	3	7864.56	
2	18	2	7	1	7	4	4171.64	
3	61	5	18	2	5	3	8517.27	
4	49	3	3	3	2	5	2000.00	
...	...	...	...	...	...	...	...	
9995	32	5	11	9	4	1	7221.75	
9996	43	3	10	4	1	2	7423.44	
9997	32	5	19	4	9	2	6545.84	
9998	40	4	19	8	10	5	11162.27	
9999	39	1	17	6	7	2	6710.05	

10000 rows × 29 columns

Explore the pattern of new feature : `PromotionStagnationRatio`

- It reflects how long an employee has been “stuck” without a promotion, relative to how long they’ve been with the company

In [ ]:
```
# Visualization
plot_bar_distributions_by_attrition(
    df=df,
    num_vars = ['PromotionStagnationRatio'],
    out_dir="output/4_feature_engineering",
    y_lim=(55, 92),
    x_rotation=0  # cleaner for integers or binned vars
)
```



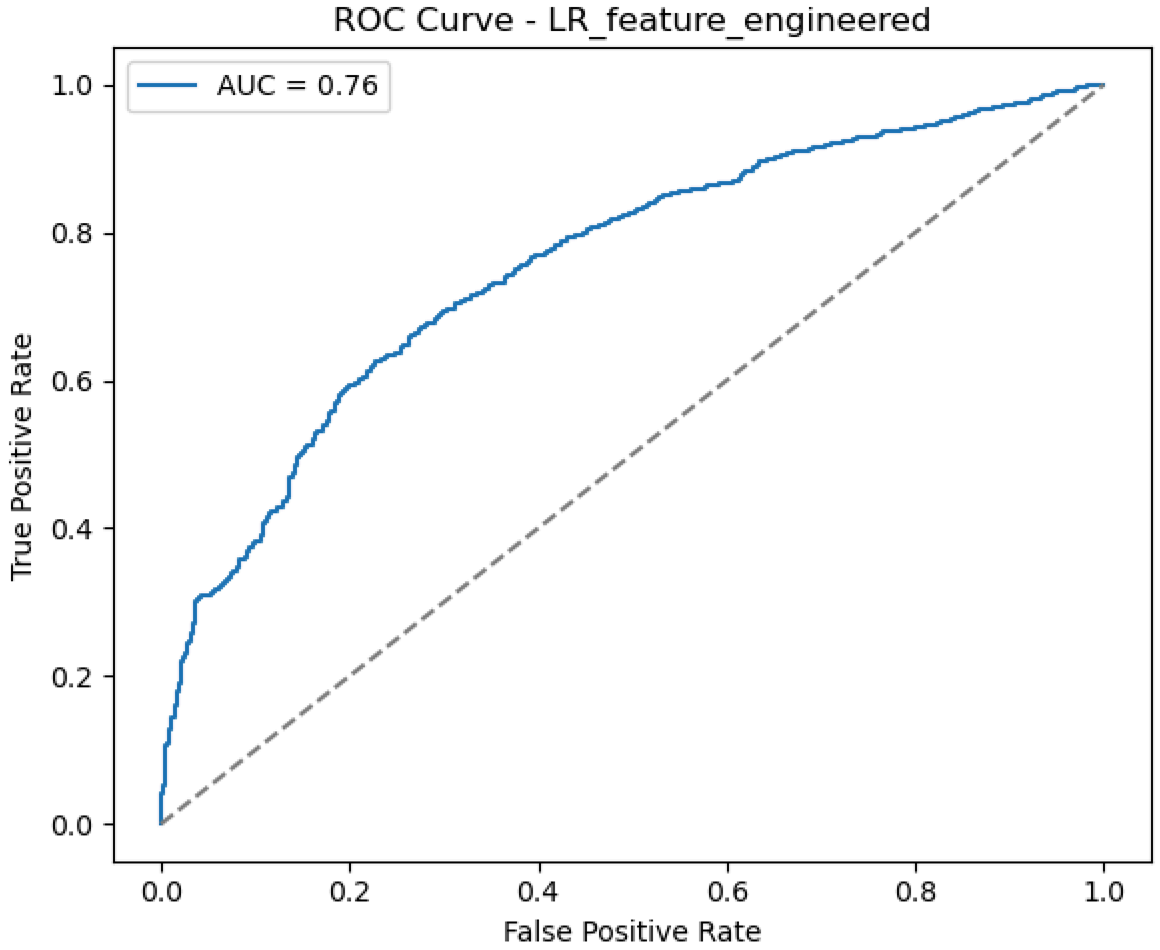
1. Skewed distribution: most values are clustered on the left side (close to 0–1)
2. May need binning

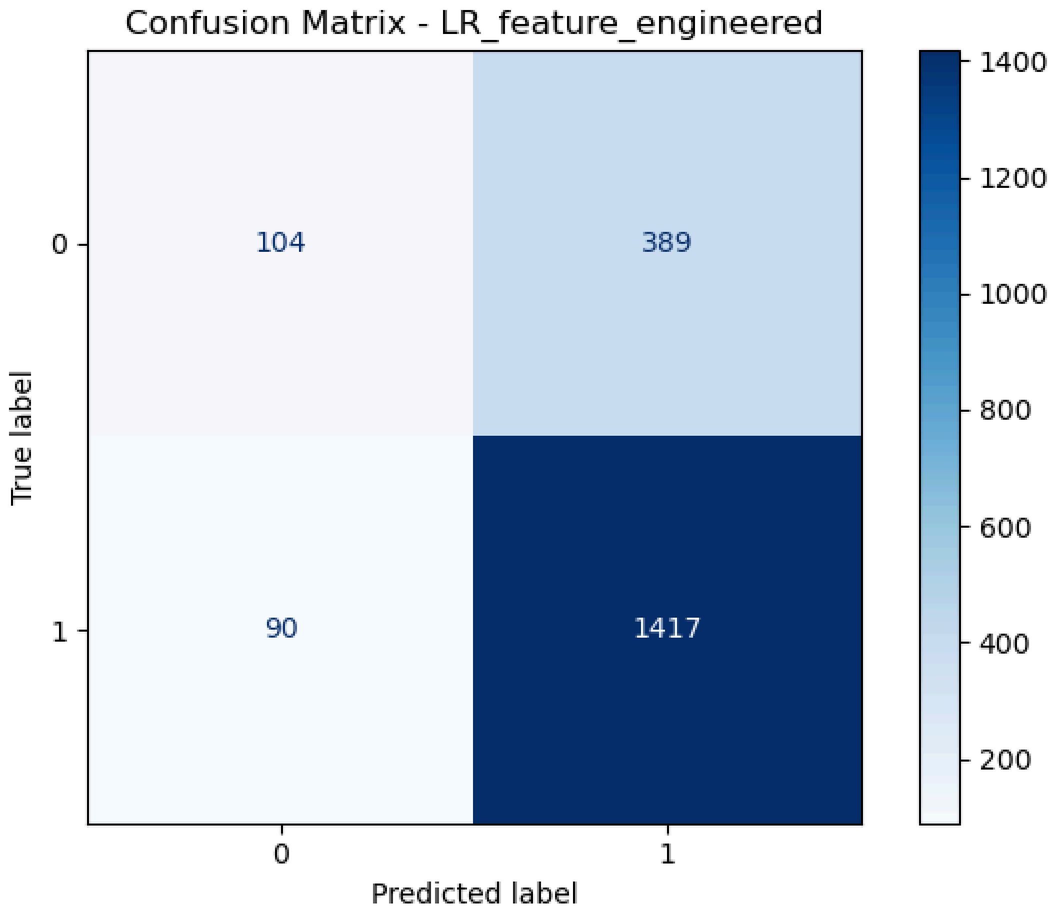
Retrain model

In [ ]:
```
# --- Feature Engineering
df = feature_engineering_interaction_and_scaling(df)

# --- Define X, y and scale numeric features
drop_cols = ['Attrition']
X, y = define_X_y(df, drop_cols)

# --- Train-test split
X_train, X_test, y_train, y_test = get_train_test_split(X, y)

# --- Train & Evaluate with New Features
```

```
model, report_df, auc = train_logistic_model(
    X_train, X_test, y_train, y_test,
    model_name="LR_feature_engineered",
    out_dir="output/4_model_feature_engineer"
)
```

```
Training model: LR_feature_engineered

Time taken: 0h 0m 0.03s
              precision    recall  f1-score    support
0              0.536082  0.210953  0.302766   493.0000
1              0.784607  0.940279  0.855418  1507.0000
accuracy       0.760500  0.760500  0.760500     0.7605
macro avg      0.660345  0.575616  0.579092  2000.0000
weighted avg   0.723346  0.760500  0.719189  2000.0000
```





Compare results

In [ ]:

```
import pandas as pd
import os

def extract_metrics_with_auc(report_path: str, pred_path: str, model_name: str) -> pd.DataFrame:
    """
    Extract accuracy, precision, recall, F1-score and AUC from saved classification report and predictions.
    """
    # Load classification report
    report_df = pd.read_csv(report_path, index_col=0)

    # Load predictions
    preds = pd.read_csv(pred_path)

    from sklearn.metrics import roc_auc_score
    auc_score = roc_auc_score(preds["y_true"], preds["y_prob"])

    metrics = {
        "Model": model_name,
        "Accuracy": report_df.loc["accuracy", "f1-score"],
```

```
        "Precision": report_df.loc["1", "precision"],
        "Recall": report_df.loc["1", "recall"],
        "F1-Score": report_df.loc["1", "f1-score"],
        "ROC-AUC": auc_score
    }
    return pd.DataFrame([metrics])
```

In [ ]:
```
# Define paths
base_dir = "output"

model_configs = [
    ("3_baseline_model/LR_baseline", "LR_Baseline"),
    ("4_model_feature_engineer/LR_feature_engineered", "LR_FeatureEngineered"),
]

# Extract all metrics
all_summaries = []
for subdir, name in model_configs:
    report_path = os.path.join(base_dir, subdir, "classification_report.csv")
    pred_path = os.path.join(base_dir, subdir, "predictions.csv")
    summary = extract_metrics_with_auc(report_path, pred_path, model_name=name)
    all_summaries.append(summary)

# Combine and export
summary_df = pd.concat(all_summaries, ignore_index=True)
display(summary_df)
```

	Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
0	LR_Baseline	0.7585	0.780088	0.946251	0.855172	0.754451
1	LR_FeatureEngineered	0.7605	0.784607	0.940279	0.855418	0.755782

5. Domain Knowledge Feature

1. Constructs a new feature called `OverworkedAndUnhappy` and set the threshold based on previous EDA insights
   - Frequently working overtime
   - Low job satisfaction
   - Poor work-life balance
2. Visualizatuon
3. Rerun the model and see if this domain-driven feature boosts accuracy or recall for attrition.

Construct a new feature

In [ ]:
```
def add_domain_knowledge_features(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()

    df["OverworkedAndUnhappy"] = (
        (df["Overtime_yes"] == 1) &
        (df["JobSatisfaction"] <= 3) &
        (df["WorkLifeBalance"] <= 4)
    ).astype(int)

    return df
```

In [ ]:
```
# Apply
df = add_domain_knowledge_features(df)
df
```

Out[ ]:

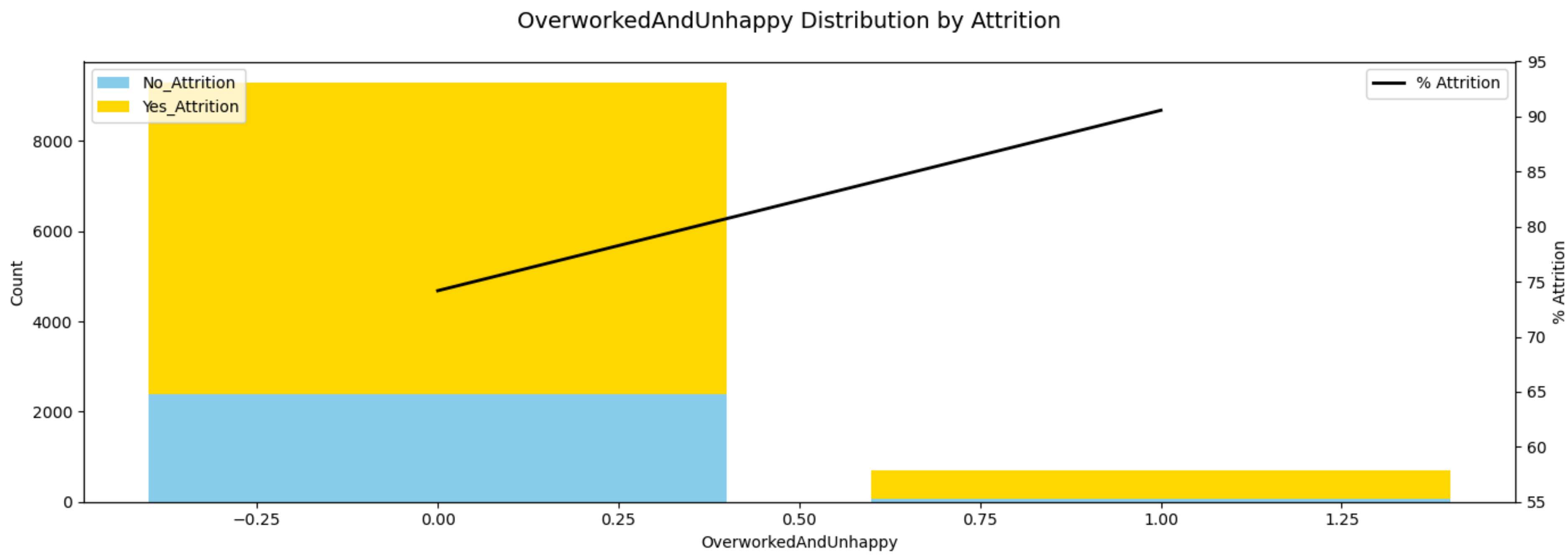
	Age	EducationLevel	YearsAtCompany	YearsSinceLastPromotion	JobSatisfaction	WorkLifeBalance	MonthlyIncome	DistanceFro
0	27	2	10	2	4	1	5363.93	
1	31	4	0	7	2	3	7864.56	
2	18	2	7	1	7	4	4171.64	
3	61	5	18	2	5	3	8517.27	
4	49	3	3	3	2	5	2000.00	
...	...	...	...	...	...	...	...	
9995	32	5	11	9	4	1	7221.75	
9996	43	3	10	4	1	2	7423.44	
9997	32	5	19	4	9	2	6545.84	
9998	40	4	19	8	10	5	11162.27	
9999	39	1	17	6	7	2	6710.05	

10000 rows × 30 columns

EDA

In [ ]:
```
plot_bar_distributions_by_attrition(
    df=df,
```

```
    num_vars=["OverworkedAndUnhappy"],
    out_dir="output/5_domain_features",
    y_lim=(55, 95),
    x_rotation=0
)
```



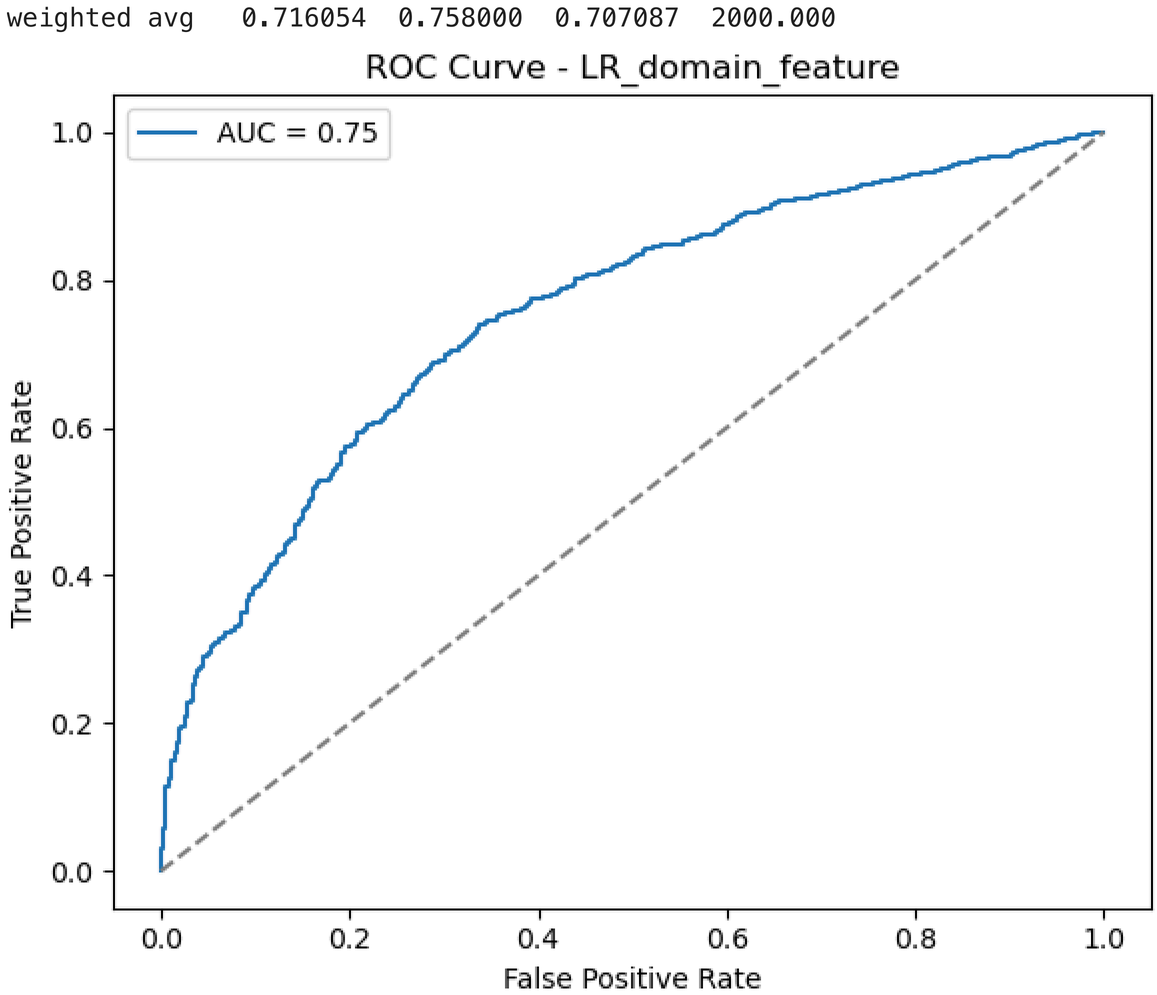
This OverworkedAndUnhappy indicator is highly possibly predictive

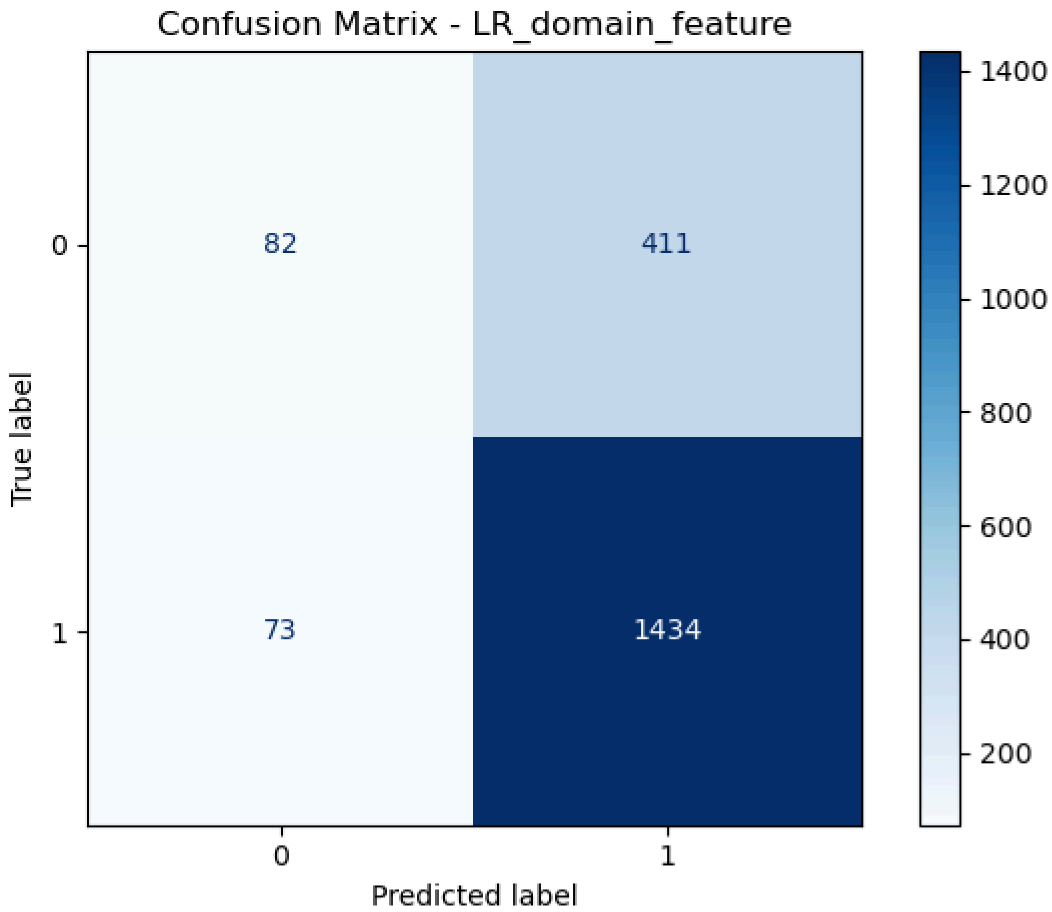
- Most employees fall into the “0” group (not overworked/unhappy), In group 0, attrition is lower (~70%)
- In group 1, attrition shoots up to over 90%, which shows that people flagged as overworked and unhappy are far more likely to quit

Retrain model

In [ ]:
```
# --- Feature Engineering
df = add_domain_knowledge_features(df)

# --- Define X, y and scale numeric features
drop_cols = ['Attrition']
X, y = define_X_y(df, drop_cols)

# --- Train-test split
X_train, X_test, y_train, y_test = get_train_test_split(X, y)

# --- Train & Evaluate with New Features
model, report_df, auc = train_logistic_model(
    X_train, X_test, y_train, y_test,
    model_name="LR_domain_feature",
    out_dir="output/5_model_domain_feature"
)
```

```
Training model: LR_domain_feature

Time taken: 0h 0m 0.03s
              precision    recall  f1-score   support
0              0.529032  0.166329  0.253086   493.000
1              0.777236  0.951559  0.855609  1507.000
accuracy       0.758000  0.758000  0.758000     0.758
macro avg      0.653134  0.558944  0.554348  2000.000
weighted avg   0.716054  0.758000  0.707087  2000.000
```





Compare results

```
In [ ]: # Define paths
base_dir = "output"

model_configs = [
    ("3_baseline_model/LR_baseline", "LR_Baseline"),
    ("4_model_feature_engineer/LR_feature_engineered", "LR_FeatureEngineered"),
    ("5_model_domain_feature/LR_domain_feature", "LR_DomainFeature")
]

# Extract all metrics
all_summaries = []
for subdir, name in model_configs:
    report_path = os.path.join(base_dir, subdir, "classification_report.csv")
    pred_path = os.path.join(base_dir, subdir, "predictions.csv")
    summary = extract_metrics_with_auc(report_path, pred_path, model_name=name)
    all_summaries.append(summary)

# Combine and export
summary_df = pd.concat(all_summaries, ignore_index=True)
display(summary_df)
```

	Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
0	LR_Baseline	0.7585	0.780088	0.946251	0.855172	0.754451
1	LR_FeatureEngineered	0.7605	0.784607	0.940279	0.855418	0.755782
2	LR_DomainFeature	0.7580	0.777236	0.951559	0.855609	0.754215

6. Feature Selection

1. Determine which features matter most
    - Visualize feature importance
2. Remove irrelevant or redundant features and retrain.
3. Compare with the model that used all features.

Get Feature Coefficients from Logistic Model

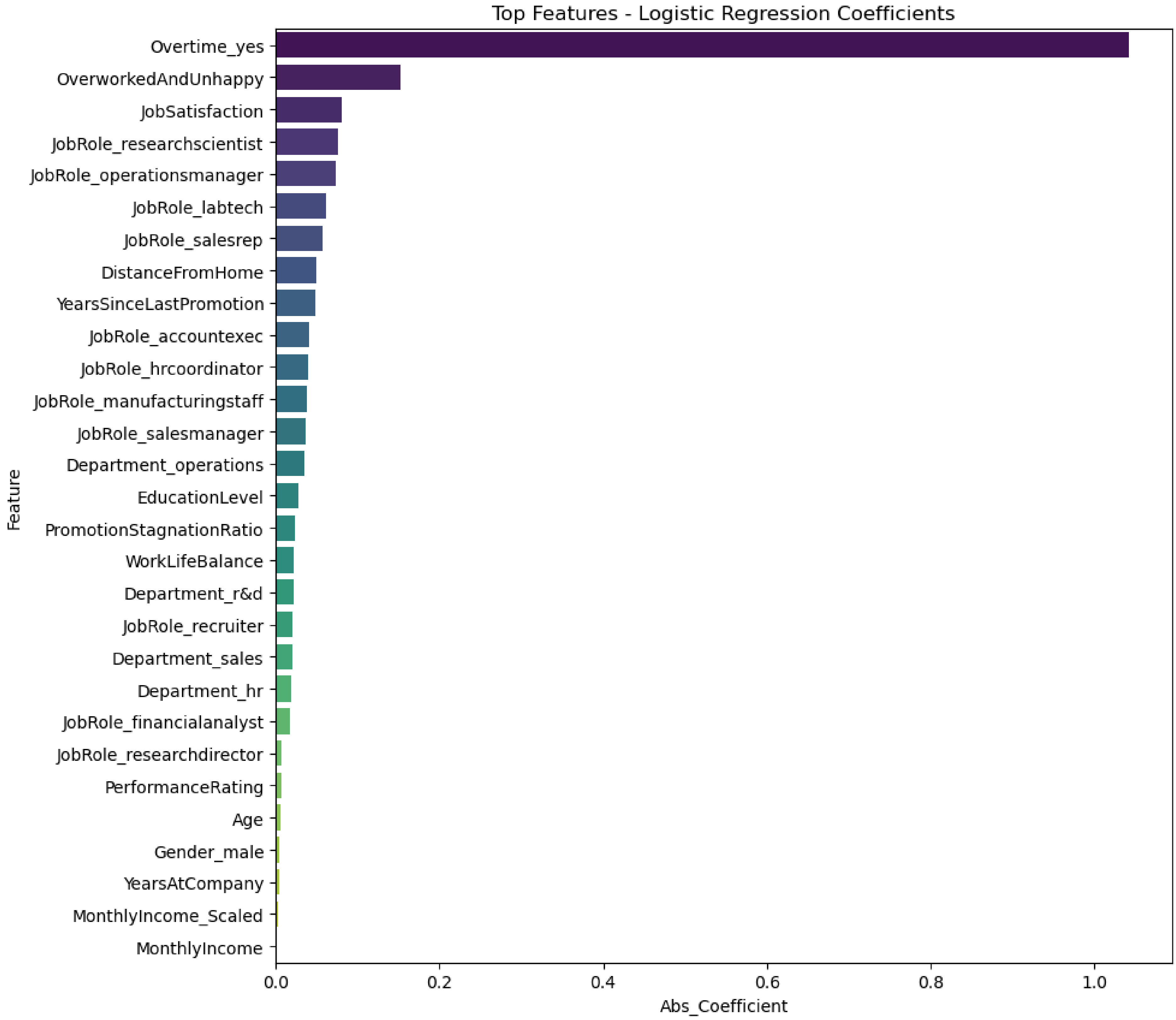
```
In [ ]: def get_logistic_coefficients(model, feature_names) -> pd.DataFrame:
    """
    Return absolute coefficients of logistic regression as feature importance.
    """
    coef_df = pd.DataFrame({
        "Feature": feature_names,
        "Coefficient": model.coef_[0]
    })
    coef_df["Abs_Coefficient"] = coef_df["Coefficient"].abs()
    return coef_df.sort_values("Abs_Coefficient", ascending=False)
```

Define feature importance plot

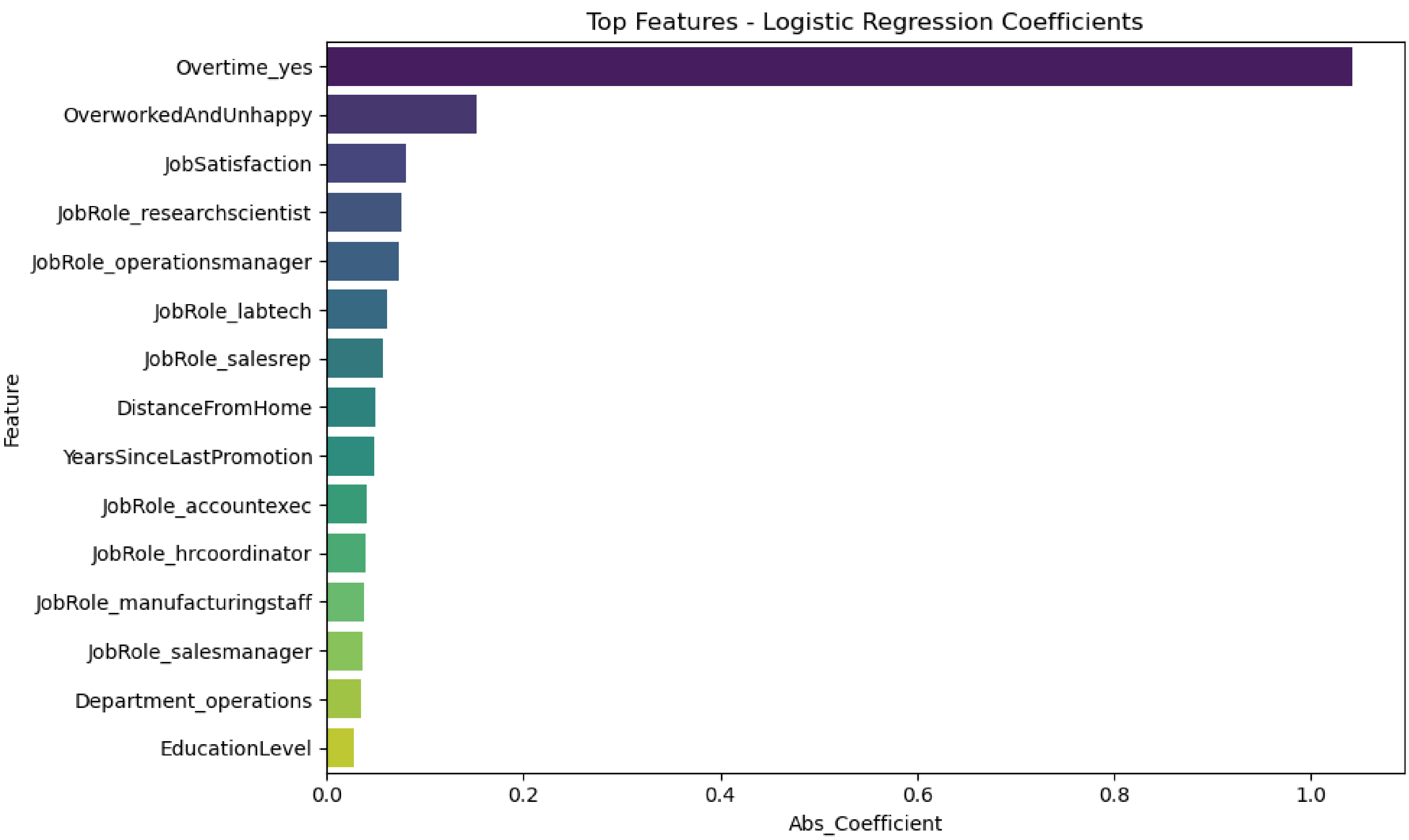
```
In [ ]: def plot_logistic_features(coef_df: pd.DataFrame, out_path: str = "output/6_feature_selection/logistic_features_rank.pr
    """
    Barplot of top-N logistic regression features by absolute coefficient value.
    """
    os.makedirs(os.path.dirname(out_path), exist_ok=True)
    coef_df_sorted = coef_df.sort_values("Abs_Coefficient", ascending=False)
```

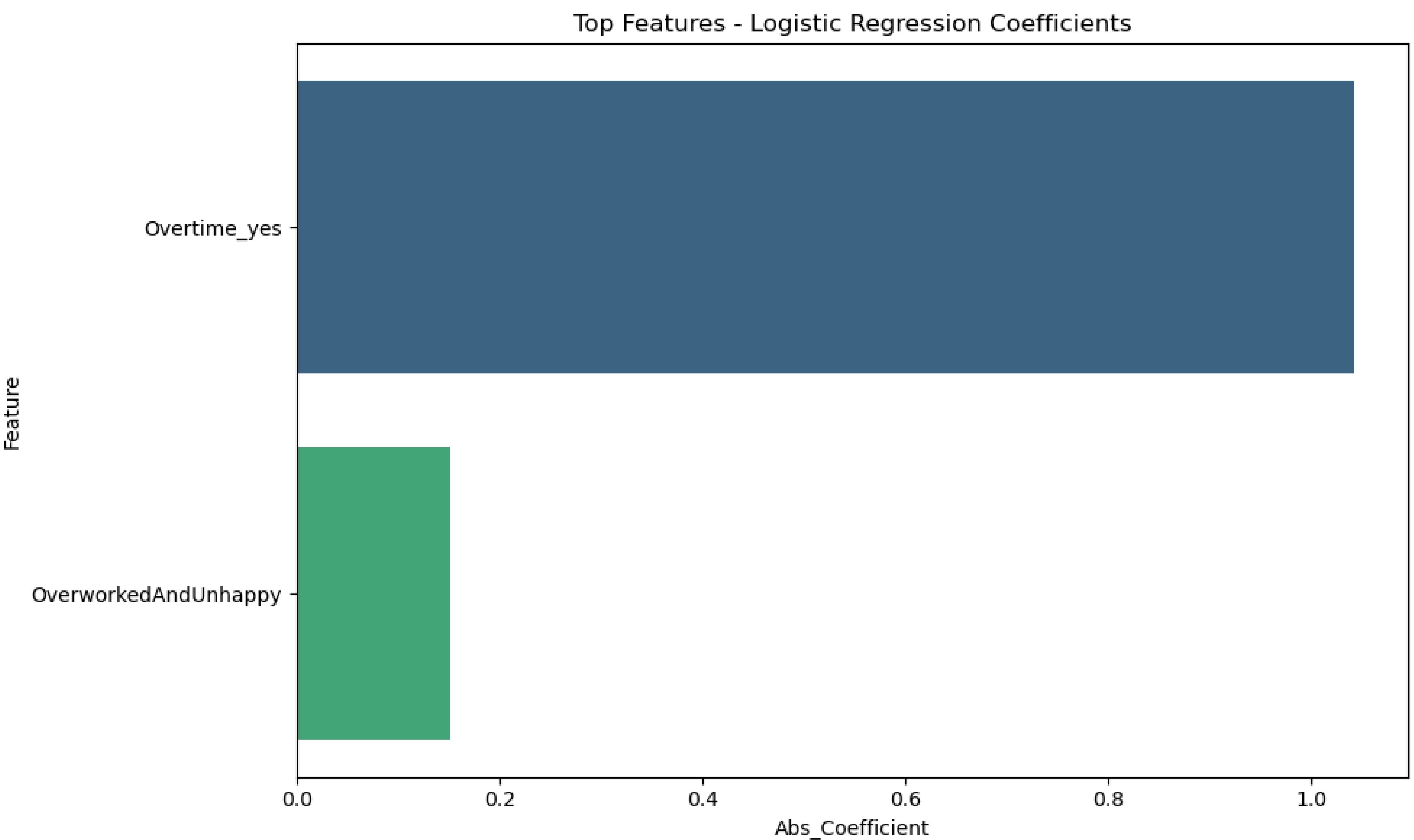
```python
    plt.figure(figsize=(10, max(6, 0.3 * len(coef_df_sorted))))  # dynamic height
    sns.barplot(data=coef_df_sorted, y="Feature", x="Abs_Coefficient", palette="viridis")
    plt.title("Top Features - Logistic Regression Coefficients")
    plt.tight_layout()
    plt.savefig(out_path)
    plt.show()
```

In [ ]:
```python
# Apply coefficient extraction using the actual model object
coef_df = get_logistic_coefficients(model, X.columns)

# Plot all coefficients
plot_logistic_features(coef_df, out_path="output/6_feature_selection/logistic_all_coefficients.png")
```



In [ ]:
```python
# Sort full coefficient DataFrame
coef_df_sorted = coef_df.sort_values("Abs_Coefficient", ascending=False)

# --- Plot top 15 features
top_15_df = coef_df_sorted.head(15)
plot_logistic_features(top_15_df, out_path="output/6_feature_selection/logistic_top_15.png")

# --- Plot features with abs(coef) > 0.1
threshold_df = coef_df_sorted[coef_df_sorted["Abs_Coefficient"] > 0.1]
plot_logistic_features(threshold_df, out_path="output/6_feature_selection/logistic_0.1.png")
```

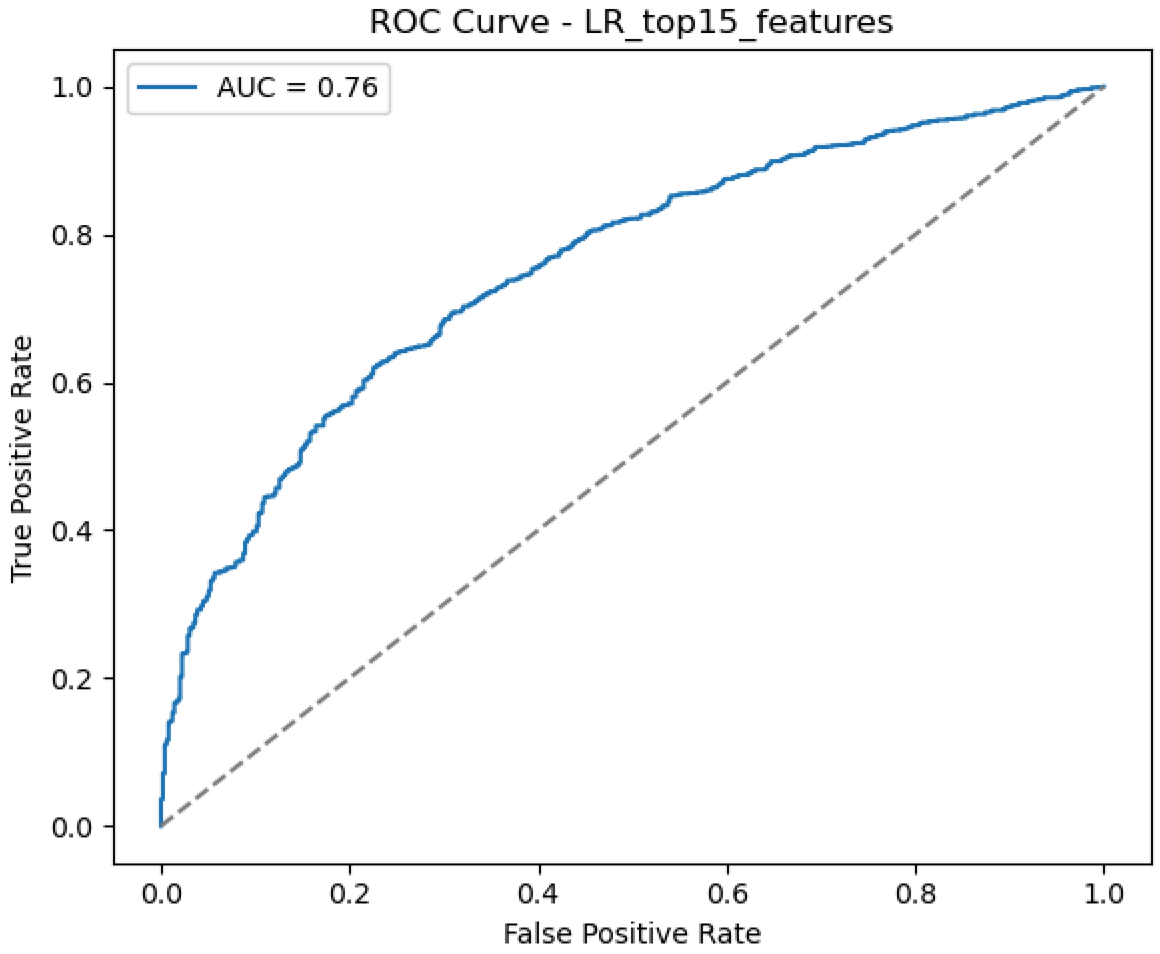


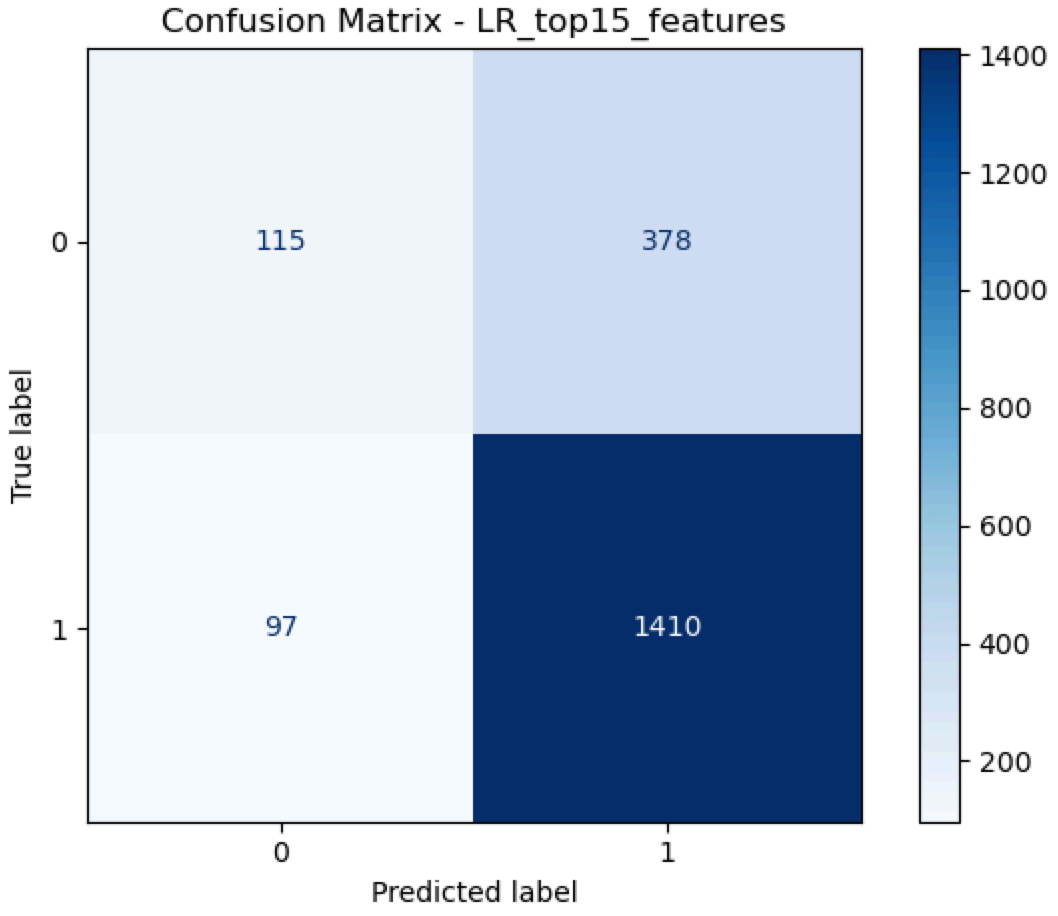


Drop Low-Importance Features and Retrain

In [ ]:
```
# --- Select Top 15 Features
top_features = coef_df.sort_values("Abs_Coefficient", ascending=False).head(15)["Feature"].tolist()

# --- Subset X using only Top 15
X_selected = X[top_features]
y = df["Attrition"]

# --- Train-Test Split
X_train, X_test, y_train, y_test = get_train_test_split(X_selected, y)

# --- Retrain Logistic Model with Selected Features
model, report_df, auc_score = train_logistic_model(
    X_train, X_test, y_train, y_test,
    model_name="LR_top15_features",
    out_dir="output/6_feature_selection/LR_top15_features"
)
```

```
Training model: LR_top15_features

Time taken: 0h 0m 0.01s
              precision    recall  f1-score    support
0              0.542453  0.233266  0.326241   493.0000
1              0.788591  0.935634  0.855842  1507.0000
accuracy       0.762500  0.762500  0.762500     0.7625
macro avg      0.665522  0.584450  0.591042  2000.0000
weighted avg   0.727918  0.762500  0.725296  2000.0000
```





7. Performance Comparison

Compare results

In [ ]:
```
# Define paths
base_dir = "output"

model_configs = [
    ("3_baseline_model/LR_baseline", "LR_Baseline"),
    ("4_model_feature_engineer/LR_feature_engineered", "LR_FeatureEngineered"),
    ("5_model_domain_feature/LR_domain_feature", "LR_DomainFeature"),
    ("6_feature_selection/LR_top15_features/LR_top15_features", "LR_Top15Features")
]


# Extract all metrics
all_summaries = []
for subdir, name in model_configs:
    report_path = os.path.join(base_dir, subdir, "classification_report.csv")
    pred_path = os.path.join(base_dir, subdir, "predictions.csv")
    summary = extract_metrics_with_auc(report_path, pred_path, model_name=name)
    all_summaries.append(summary)

# Combine and export
```

```python
summary_df = pd.concat(all_summaries, ignore_index=True)
summary_df.to_csv("output/model_comparison_summary_4models.csv", index=False)
display(summary_df)
```

	Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
0	LR_Baseline	0.7585	0.780088	0.946251	0.855172	0.754451
1	LR_FeatureEngineered	0.7605	0.784607	0.940279	0.855418	0.755782
2	LR_DomainFeature	0.7580	0.777236	0.951559	0.855609	0.754215
3	LR_Top15Features	0.7625	0.788591	0.935634	0.855842	0.755534

1. Feature engineering yields small but meaningful gains
2. Using domain knowledge doesn't degrade performance and can help with interpretability
3. Feature selection (top 15) offers comparable or better performance with fewer variables, improving model simplicity and deployment

8. New Feature Experimentation

1. Add another creative feature based on potential HR insights: Distance-to-Income Ratio
    A. CommuteCostIndex = DistanceFromHome / MonthlyIncome
    B. Rationale: Employees commuting far for low pay may be more dissatisfied and likely to leave.
2. EDA
3. Train a new model and evaluate performance changes

Construct a new feature

In [ ]:
```python
def add_new_feature(df: pd.DataFrame) -> pd.DataFrame:
    df = df.copy()
    # Avoid division by 0
    df["CommuteCostIndex"] = df["DistanceFromHome"] / (df["MonthlyIncome"] + 1)
    return df
```

In [ ]:
```python
# Apply
df = add_new_feature(df)
df
```

Out[ ]:

	Age	EducationLevel	YearsAtCompany	YearsSinceLastPromotion	JobSatisfaction	WorkLifeBalance	MonthlyIncome	DistanceFro
0	27	2	10	2	4	1	5363.93	
1	31	4	0	7	2	3	7864.56	
2	18	2	7	1	7	4	4171.64	
3	61	5	18	2	5	3	8517.27	
4	49	3	3	3	2	5	2000.00	
...	...	...	...	...	...	...	...	
9995	32	5	11	9	4	1	7221.75	
9996	43	3	10	4	1	2	7423.44	
9997	32	5	19	4	9	2	6545.84	
9998	40	4	19	8	10	5	11162.27	
9999	39	1	17	6	7	2	6710.05	

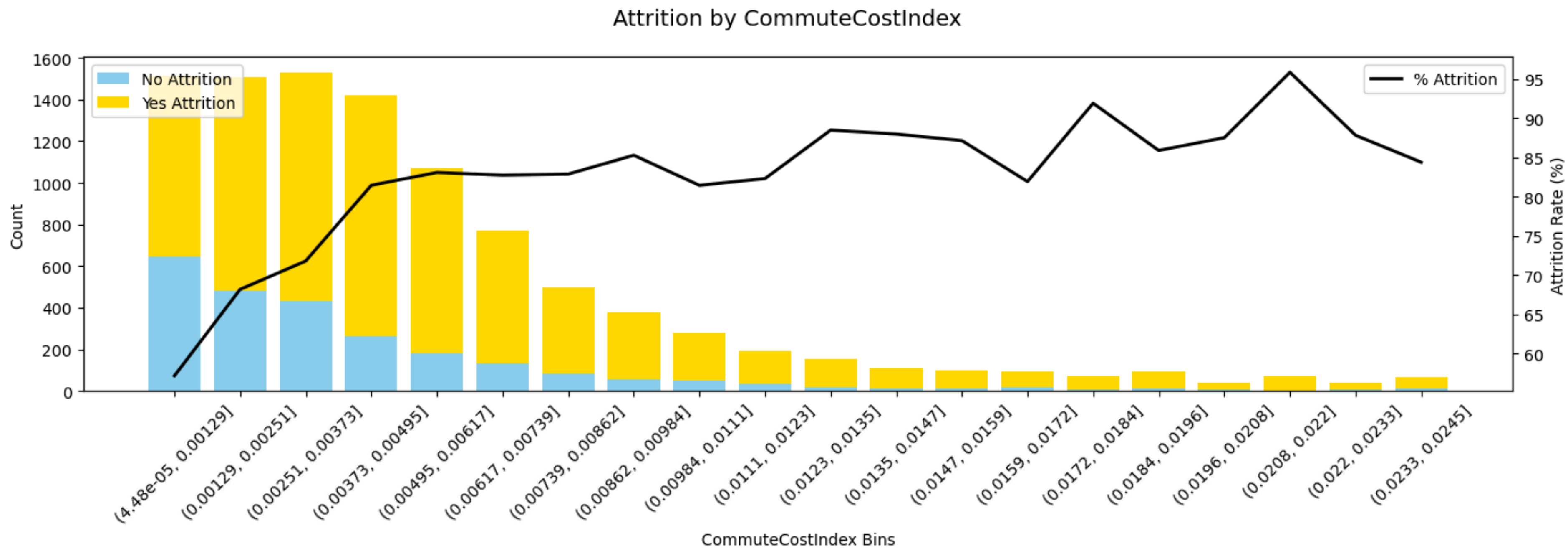
10000 rows × 31 columns

EDA

In [ ]:
```python
def plot_commute_cost_by_attrition(df: pd.DataFrame, bins: int = 20, out_dir: str = "output/8_model_commute_cost") -> N
    """
    Bin CommuteCostIndex and show distribution by Attrition
    """
    df = df.copy()
    df["CommuteBin"] = pd.cut(df["CommuteCostIndex"], bins=bins)

    grouped = df.groupby(["CommuteBin", "Attrition"]).size().unstack(fill_value=0)
    grouped["Total"] = grouped.sum(axis=1)
    grouped["AttritionRate"] = grouped[1] / grouped["Total"] * 100
    grouped = grouped.reset_index()

    # Plot
    fig, ax1 = plt.subplots(figsize=(14, 5))
    ax1.bar(grouped["CommuteBin"].astype(str), grouped[0], color="skyblue", label="No Attrition")
    ax1.bar(grouped["CommuteBin"].astype(str), grouped[1], bottom=grouped[0], color="gold", label="Yes Attrition")
    ax1.set_ylabel("Count")
    ax1.set_xlabel("CommuteCostIndex Bins")
    ax1.tick_params(axis="x", rotation=45)

    ax2 = ax1.twinx()
    ax2.plot(grouped["CommuteBin"].astype(str), grouped["AttritionRate"], color="black", linewidth=2, label="% Attritic
```

```
    ax2.set_ylabel("Attrition Rate (%)")

    fig.suptitle("Attrition by CommuteCostIndex", fontsize=14)
    ax1.legend(loc="upper left")
    ax2.legend(loc="upper right")

    os.makedirs(out_dir, exist_ok=True)
    plt.tight_layout()
    plt.savefig(f"{out_dir}/CommuteCostIndex_by_attrition.png")
    plt.show()
```

In [ ]:
```
plot_commute_cost_by_attrition(df)
```



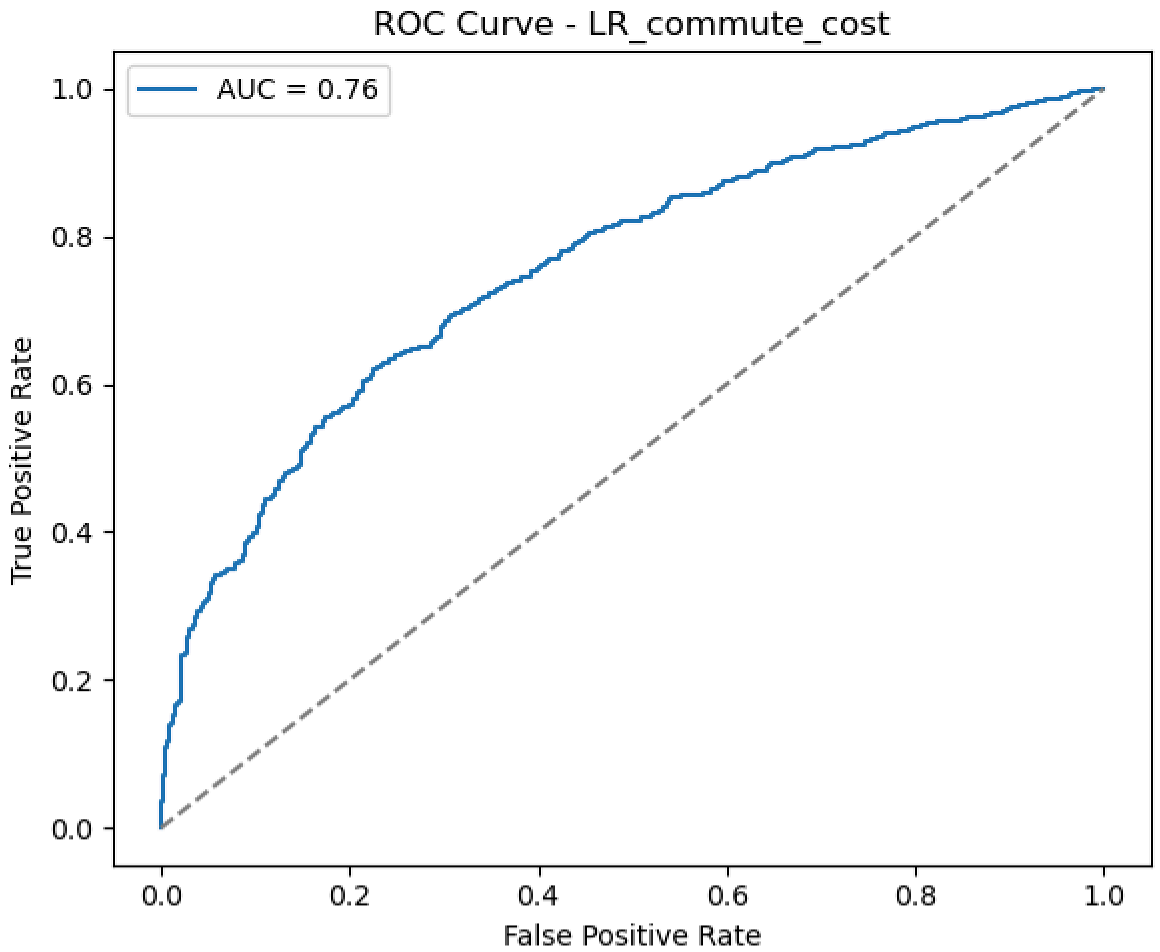
Employees commuting far for relatively low pay may be more dissatisfied and more likely to leave

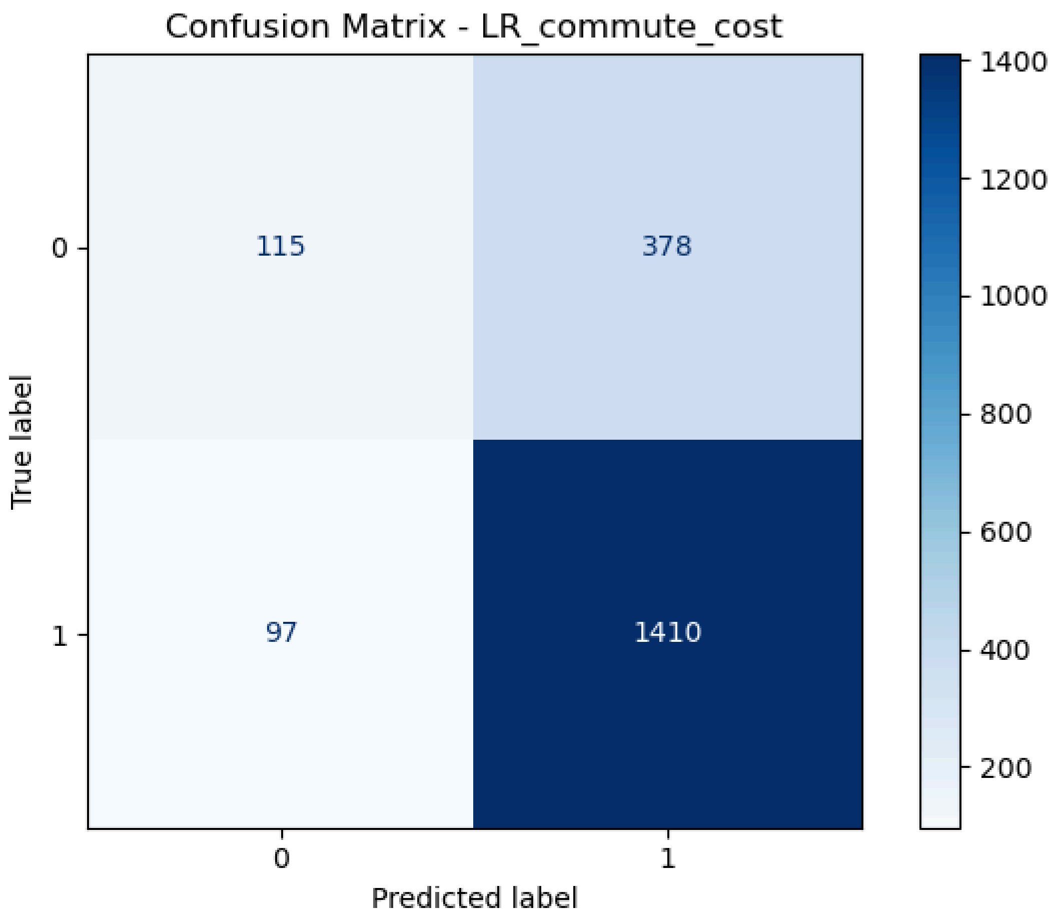
- Extremely high values are rare (small count), but attrition spikes again to 95%+

Retrain model

In [ ]:
```
# --- Step 1: Add new feature
df = add_new_feature(df)
top_features_plus = top_features + ["CommuteCostIndex"]

# --- Step 2: Define X, y with the new feature
drop_cols = ["Attrition"]
X_full, y = define_X_y(df, drop_cols)
X = X_full[top_features_plus]  # Select only the desired columns


# --- Step 3: Train-test split
X_train, X_test, y_train, y_test = get_train_test_split(X, y)

# --- Step 4: Train & evaluate
model, report_df, auc_score = train_logistic_model(
    X_train, X_test, y_train, y_test,
    model_name="LR_commute_cost",
    out_dir="output/8_model_commute_cost"
)
```

```
Training model: LR_commute_cost

Time taken: 0h 0m 0.02s
              precision    recall  f1-score    support
0              0.542453  0.233266  0.326241   493.0000
1              0.788591  0.935634  0.855842  1507.0000
accuracy       0.762500  0.762500  0.762500     0.7625
macro avg      0.665522  0.584450  0.591042  2000.0000
weighted avg   0.727918  0.762500  0.725296  2000.0000
```





Compare results

In [ ]:
```python
# --- Define paths
base_dir = "output"

model_configs = [
    ("3_baseline_model/LR_baseline", "LR_Baseline"),
    ("4_model_feature_engineer/LR_feature_engineered", "LR_FeatureEngineered"),
    ("5_model_domain_feature/LR_domain_feature", "LR_DomainFeature"),
    ("6_feature_selection/LR_top15_features/LR_top15_features", "LR_Top15Features"),
    ("8_model_commute_cost/LR_commute_cost", "LR_CommuteCost")
]

# --- Load & compare all models
all_summaries = []
for subdir, name in model_configs:
    report_path = os.path.join(base_dir, subdir, "classification_report.csv")
    pred_path = os.path.join(base_dir, subdir, "predictions.csv")
    summary = extract_metrics_with_auc(report_path, pred_path, model_name=name)
    all_summaries.append(summary)

summary_df = pd.concat(all_summaries, ignore_index=True)
summary_df.to_csv("output/model_comparison_5models.csv", index=False)
display(summary_df)
```

	Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC
0	LR_Baseline	0.7585	0.780088	0.946251	0.855172	0.754451
1	LR_FeatureEngineered	0.7605	0.784607	0.940279	0.855418	0.755782
2	LR_DomainFeature	0.7580	0.777236	0.951559	0.855609	0.754215
3	LR_Top15Features	0.7625	0.788591	0.935634	0.855842	0.755534
4	LR_CommuteCost	0.7625	0.788591	0.935634	0.855842	0.755522