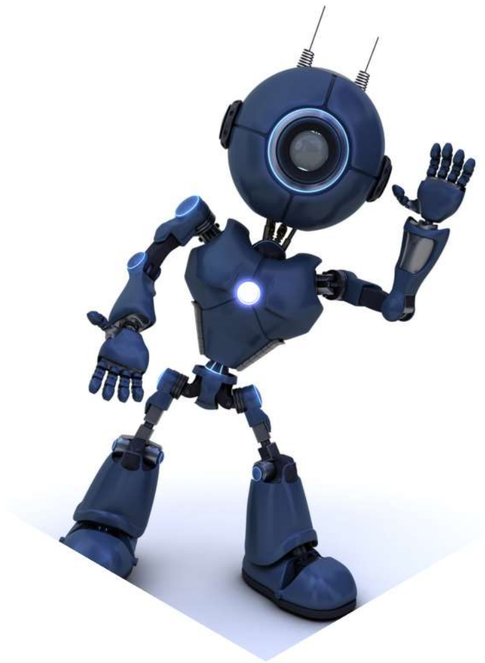
April 26, 2025 @ 09:00 A.M.

IBM HR Analytics for Employee Attrition and Performance Prediction

OIKANTIK BASU
Data Science Intern at Unified Mentor



INTRODUCTION TO HR ANALYTICS

For HR Analytics

HR Analytics, also known as People Analytics, Talent Analytics, or Workforce Analytics, is the strategic process of collecting, analyzing, and leveraging Human Resource (HR) data to enhance organizational performance. By transforming routine HR data into actionable insights, this approach aligns workforce metrics with business objectives, offering clear, data-driven evidence of how HR initiatives impact overall organizational goals and strategies. Through HR Analytics, companies can make informed decisions to optimize talent management, improve employee engagement, and drive sustainable growth.

Objective

The primary objective of this project is to analyze and predict Employee Attrition by leveraging data-driven methodologies. Through comprehensive Exploratory Data Analysis (EDA), statistical evaluation, and machine learning models, this analysis aims to:

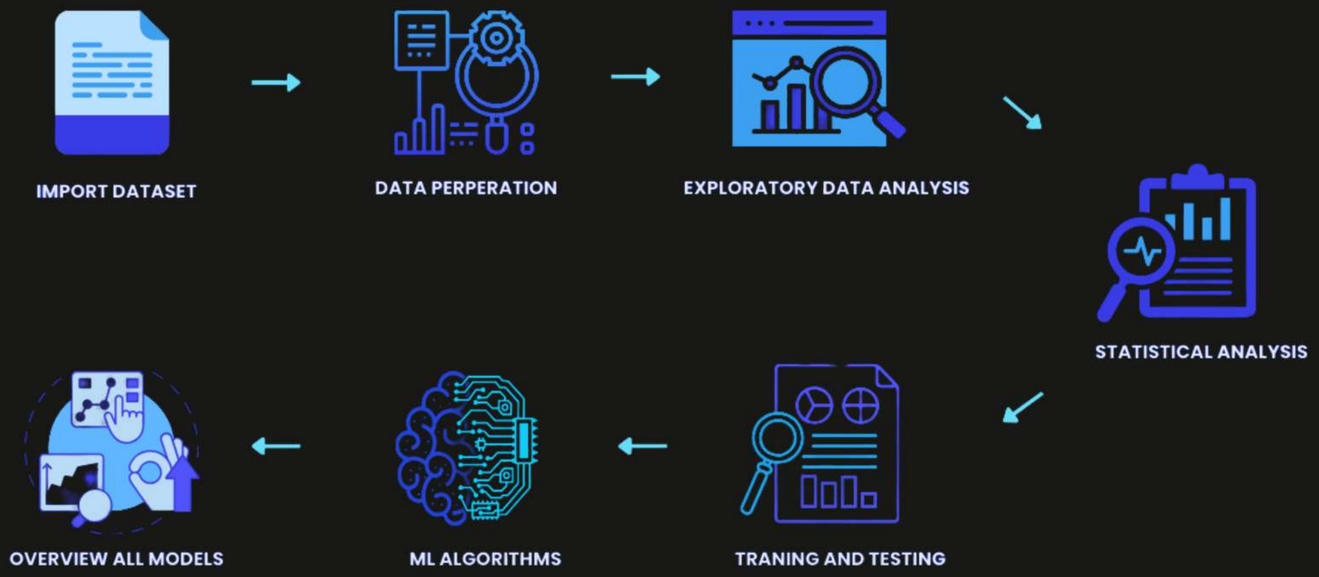
- Identify key factors influencing employee turnover.
- Quantify the likelihood of attrition across the workforce.
- Detect employees at high risk of leaving the organization.
- Provide actionable insights to support strategic HR decision-making and retention initiatives.

MOTIVATION

This project is driven by the potential of leveraging data analytics to proactively address one of the most critical challenges in Human Resource management — employee attrition. High turnover impacts organizational performance, increases operational costs, and affects workplace morale.
By harnessing insights from Exploratory Data Analysis, Statistical Techniques, and Machine Learning

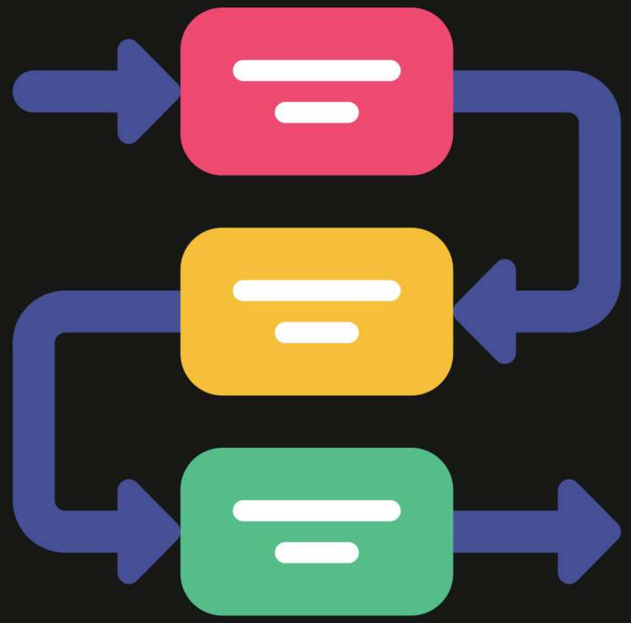
- Enhance employee satisfaction through informed HR strategies.
- Reduce attrition-related costs by identifying and mitigating key risk factors.
- Strengthen organizational performance with data-backed decision-making.
- Foster a stable and positive workplace culture by anticipating workforce challenges before they escalate.

System Architecture Diagram



A structured pipeline transforming raw HR data through Data preperation, EDA, statistical analysis, and machine learning to predict employee attrition and identify key risk factors with optimal model performance.

Methodology



- **Data Loading & Exploration:** Imported the IBM HR Analytics Attrition dataset and explored its structure using pandas.
- **Data Cleaning & Preparation:** Handled missing values, mapped target variables, and applied one-hot encoding for categorical features.
- **Exploratory Data Analysis (EDA):** Utilized Matplotlib and Seaborn for visualizing key patterns and trends related to employee attrition.
- **Statistical Analysis:** Performed ANOVA for numerical features and Chi-Square tests for categorical feature significance.
- **Data Splitting:** Divided the dataset into training and testing sets using scikit-learn's train_test_split().
- **Machine Learning Implementation:** Applied Logistic Regression, SVM, Random Forest, XGBoost, CatBoost, AdaBoost, and LightGBM classifiers.
- **Model Evaluation:** Assessed models using accuracy scores, confusion matrices, and ROC curves.
- **Performance Comparison:** Visualized and compared model performance using hvPlot to identify the best predictive model.

DATASET OVERVIEW

This analysis utilizes a hypothetical dataset developed by IBM data scientists, consisting of 1,470 records and 35 features. The dataset includes a mix of numerical and categorical variables capturing employee demographics, job roles, satisfaction levels, performance metrics, and work-life attributes. Each entry is labeled with the employee's attrition status (whether they stayed or left the company), enabling both statistical analysis and predictive modeling.

Key features include:

Age, Attrition, BusinessTravel, DailyRate, Department, DistanceFromHome, Education, EducationField, EmployeeCount, EmployeeNumber, EnvironmentSatisfaction, Gender, HourlyRate, JobInvolvement, JobLevel, JobRole, JobSatisfaction, MaritalStatus, MonthlyIncome, MonthlyRate, NumCompaniesWorked, Over18, OverTime, PercentSalaryHike, PerformanceRating, RelationshipSatisfaction, StandardHours, StockOptionLevel, TotalWorkingYears, TrainingTimesLastYear, WorkLifeBalance, YearsAtCompany, YearsInCurrentRole, YearsSinceLastPromotion, YearsWithCurrManager. — providing a comprehensive view of factors influencing employee turnover.

DATASET IMPLEMENTATION

The IBM HR Analytics Attrition dataset was imported using pandas (pd.read_csv()), containing 1,470 records and 35 features. Initial data exploration was performed with head() and tail() to review the dataset structure and verify successful loading.

```
# Load the dataset
employee_data = '../WA_Fn-UseC_-HR-Employee-Attrition.csv'
df = pd.read_csv(employee_data)

# Display the first 5 rows
df.head()
```

✔ 0.0s

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

5 rows × 35 columns

```
df.tail()
```

✔ 0.0s

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educatio
1465	36	No	Travel_Frequently	884	Research & Development	23	
1466	39	No	Travel_Rarely	613	Research & Development	6	
1467	27	No	Travel_Rarely	155	Research & Development	4	
1468	49	No	Travel_Frequently	1023	Sales	2	
1469	34	No	Travel_Rarely	628	Research & Development	8	

5 rows × 35 columns

Dataset Information Summary

The dataset comprises 1,470 entries across 35 features, including 26 numerical and 9 categorical variables. No missing values were detected, ensuring a clean dataset for analysis and modeling.

```
# Generating basic information about the dataset
df.info()
```

✔ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Age                       1470 non-null   int64
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64
 6   Education                 1470 non-null   int64
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64
 9   EmployeeNumber            1470 non-null   int64
 10  EnvironmentSatisfaction   1470 non-null   int64
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64
 13  JobInvolvement            1470 non-null   int64
 14  JobLevel                  1470 non-null   int64
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64
 19  MonthlyRate               1470 non-null   int64
...
 33  YearsSinceLastPromotion   1470 non-null   int64
 34  YearsWithCurrManager      1470 non-null   int64
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Attribute Summary:

- Total Entries: 1470
- Total Features: 35

Data Types:

- 26 Numerical (int64)
- 9 Categorical (object)

No missing values detected so far (all columns show 1470 non-null).

Data Cleaning: Removing Irrelevant Columns

To optimize analysis, constant-value and non-predictive columns were removed, reducing the dataset from 35 to 31 features. This step ensures a cleaner dataset, eliminating noise for better EDA and modeling performance.

Remove columns that don't add value for analysis or modeling.

Columns to Drop:

- Column----------Reason
- EmployeeCount----------Constant value (always 1)
- Over18----------Constant value ('Y' for all entries)
- StandardHours----------Constant value (always 80)
- EmployeeNumber----Just a unique ID, no predictive importance

Dropping these will clean up the dataset and avoid noise in further steps like EDA or Modeling.

```
# Dropping unnecessary columns
df.drop(['EmployeeCount', 'Over18', 'StandardHours', 'EmployeeNumber'], axis=1, inplace=True)

# Verifying the shape after dropping
df.shape
```

✔ 0.0s

```
(1470, 31)
```

We have successfully dropped unnecessary columns.

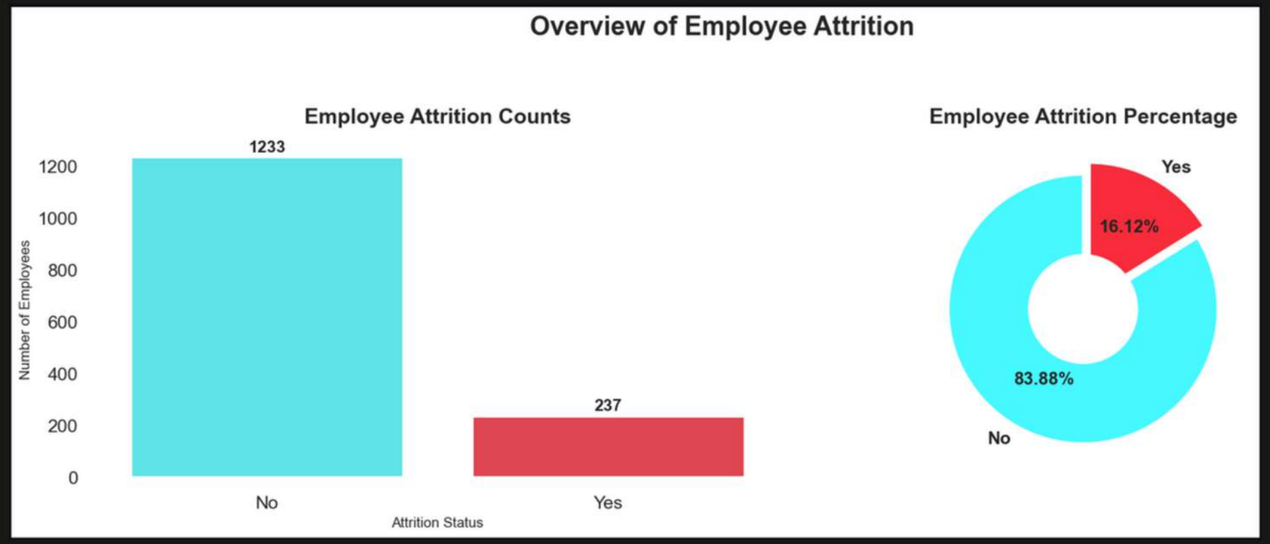
- The dataset now contains 31 columns (down from 35).
- All remaining features are relevant for analysis.

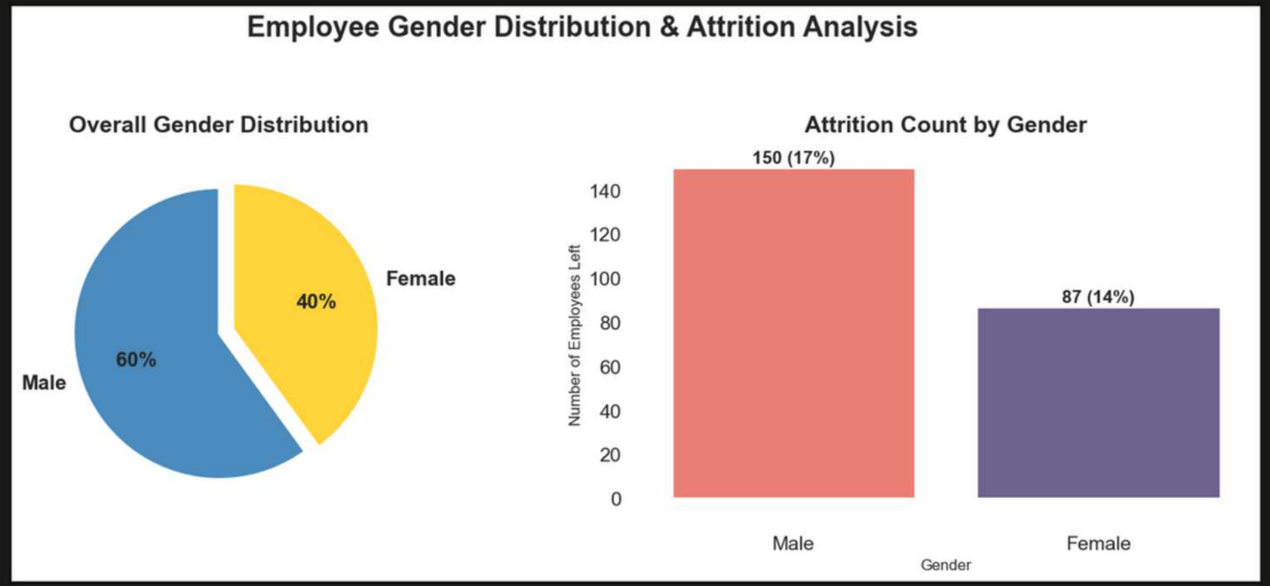
Predictive Insights & Talent Retention through HR Analytics

By leveraging HR analytics, organizations can identify key factors driving employee attrition—such as job satisfaction, compensation, work-life balance, and overtime—while also recognizing high-performing employees through performance metrics like productivity, quality, and customer satisfaction. These data-driven insights enable targeted strategies to reduce turnover, retain top talent, and enhance overall organizational performance.

Employee Attrition Insights

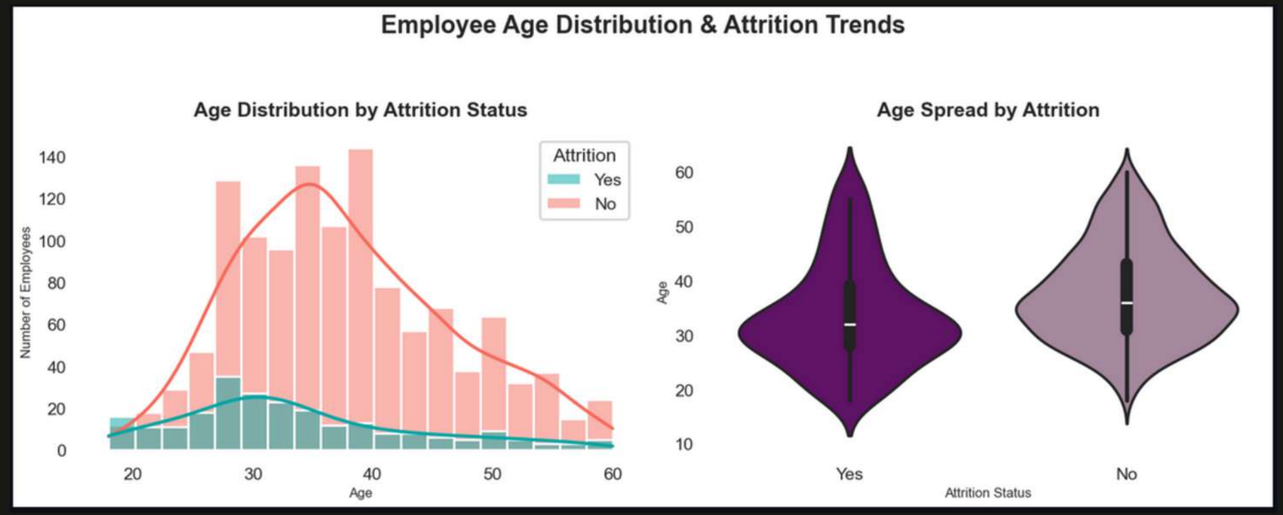
- The organization’s attrition rate is 16.12%, exceeding the typical benchmark of 4%-6%, indicating a critical concern.
- Male employees constitute a higher workforce percentage (60%) and exhibit a higher attrition rate (17%) compared to females (14%).
- Targeted retention strategies are needed, especially focusing on reducing male employee turnover to stabilize workforce dynamics.

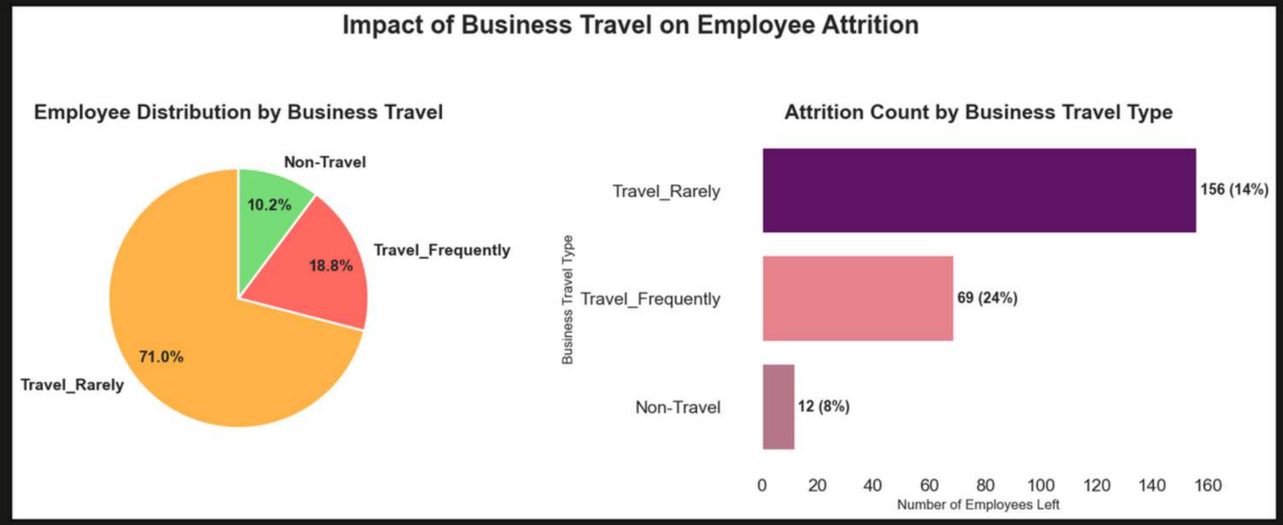




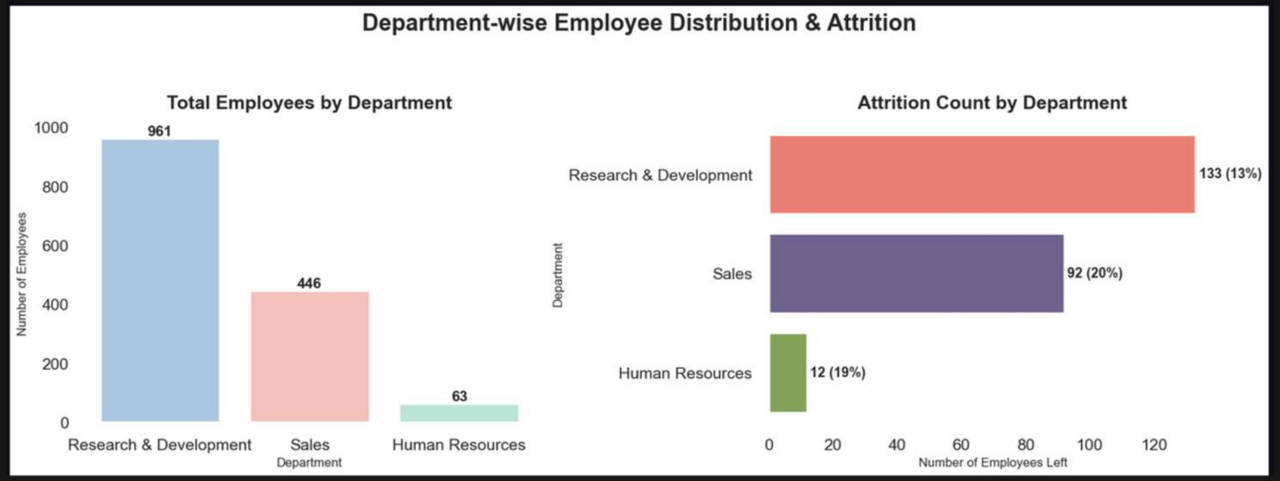
Attrition Insights: Age & Business Travel Impact

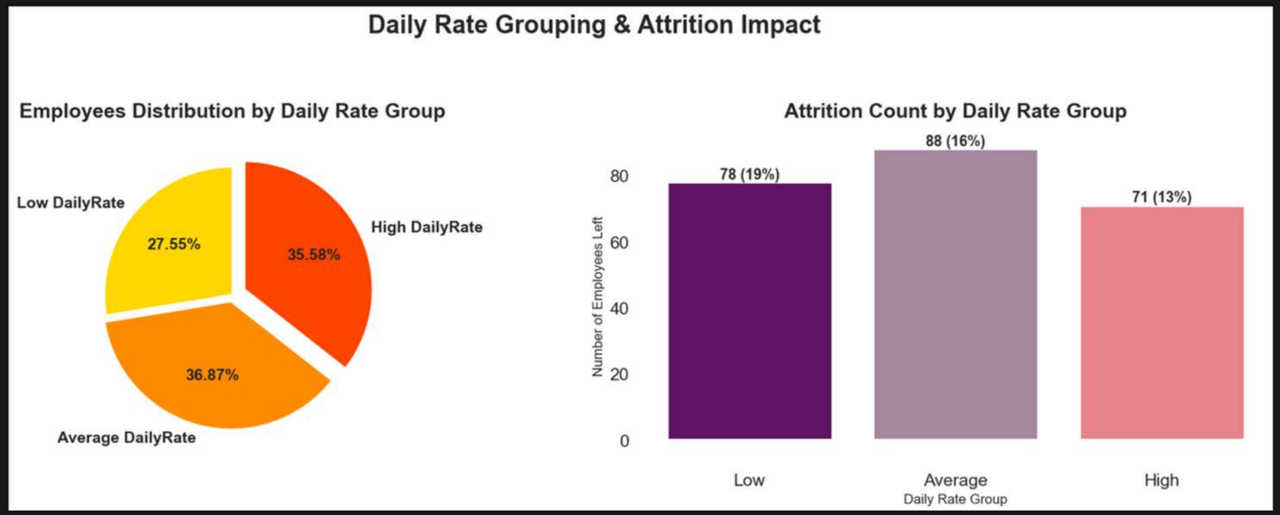
- Younger Employees (25-35) show the highest attrition rates.
- Attrition decreases with age, indicating early-career professionals are more likely to leave.
- Retention strategies should focus on supporting younger workforce segments.
- Employees who travel frequently face a 24% attrition rate, the highest among all travel categories.
- Those who travel rarely also show notable attrition, while non-travelers have the lowest attrition at 8%.
- Organizations should address travel-related stress through flexible policies and employee support programs.





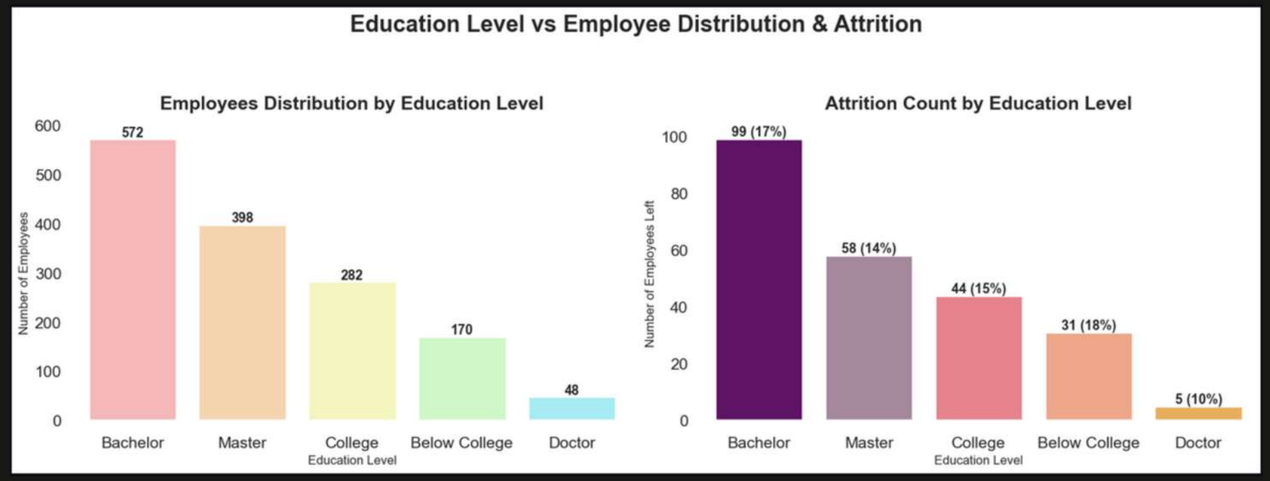
Department & Compensation Impact on Employee Attrition

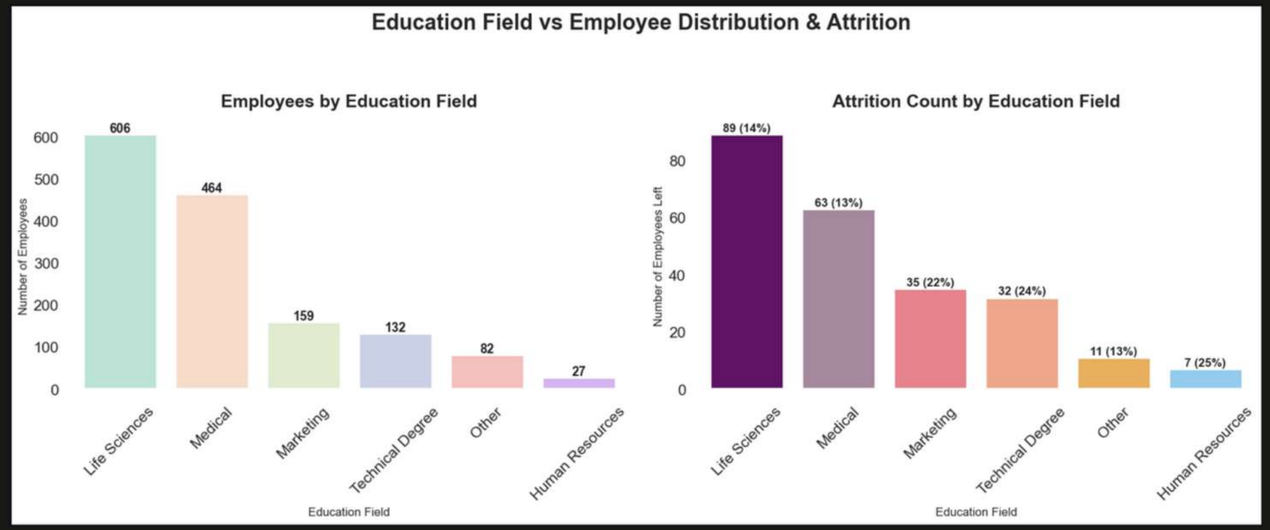




- The highest attrition rates are observed in the **Sales (20%)** and **Human Resources (19%)** departments, indicating a need for targeted retention efforts in these areas.
- Although Research & Development has the largest workforce, its attrition rate is comparatively lower at **13%**.
- Employees with a low daily rate experience the highest attrition **(19%)**, highlighting a clear correlation between lower compensation and turnover.
- Even those with average compensation face notable attrition **(16%)**, while employees with higher daily rates show reduced attrition **(13%)**.
- These insights emphasize the importance of addressing both **department-specific challenges** and **compensation structures** to improve employee retention.

Impact of Education Level & Field on Employee Attrition

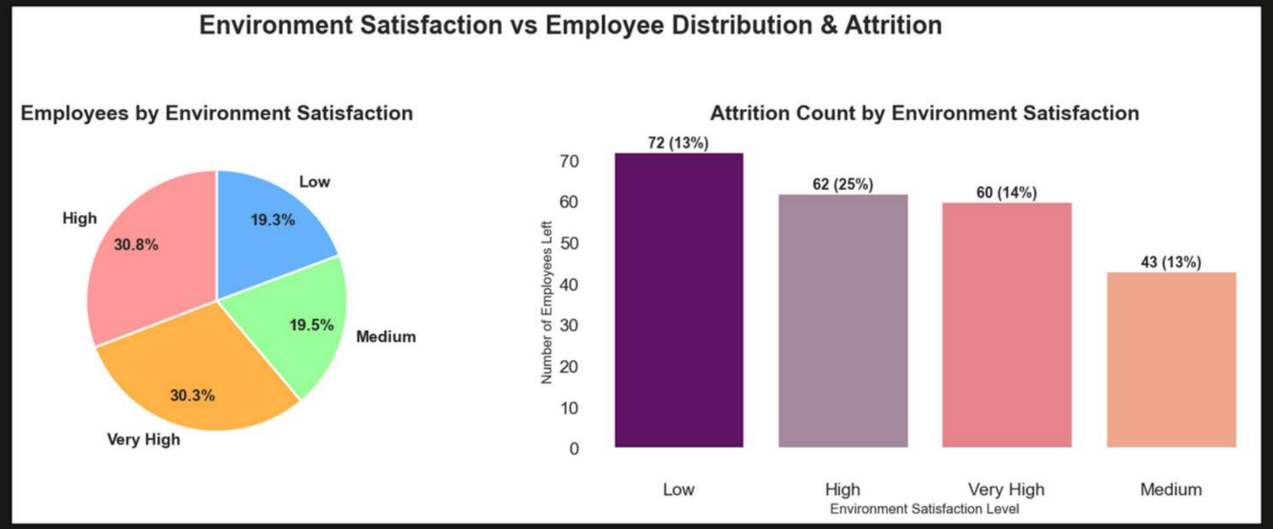


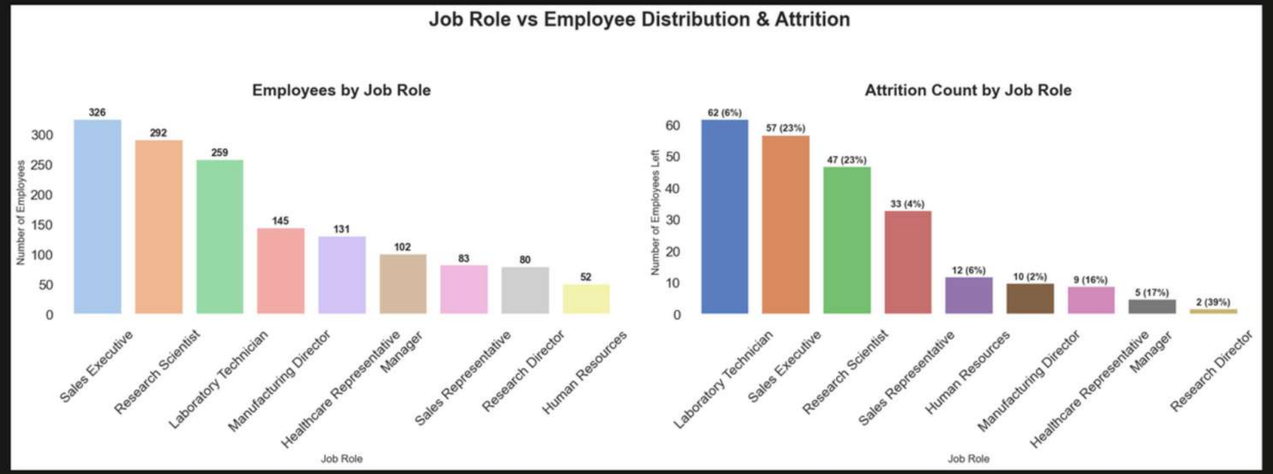


- Employees with Below College education show the highest attrition rate (18%), followed by those with a Bachelor's degree (17%).
- The lowest attrition is observed among employees with a Doctorate (10%), indicating higher education may correlate with better retention.
- In terms of Education Field, employees from Human Resources (25%), Technical Degree (24%), and Marketing (22%) experience the highest attrition rates.
- Fields like Life Sciences and Medical have larger workforces but maintain relatively lower attrition rates (13-14%).

Impact of Environment Satisfaction & Job Role on Attrition

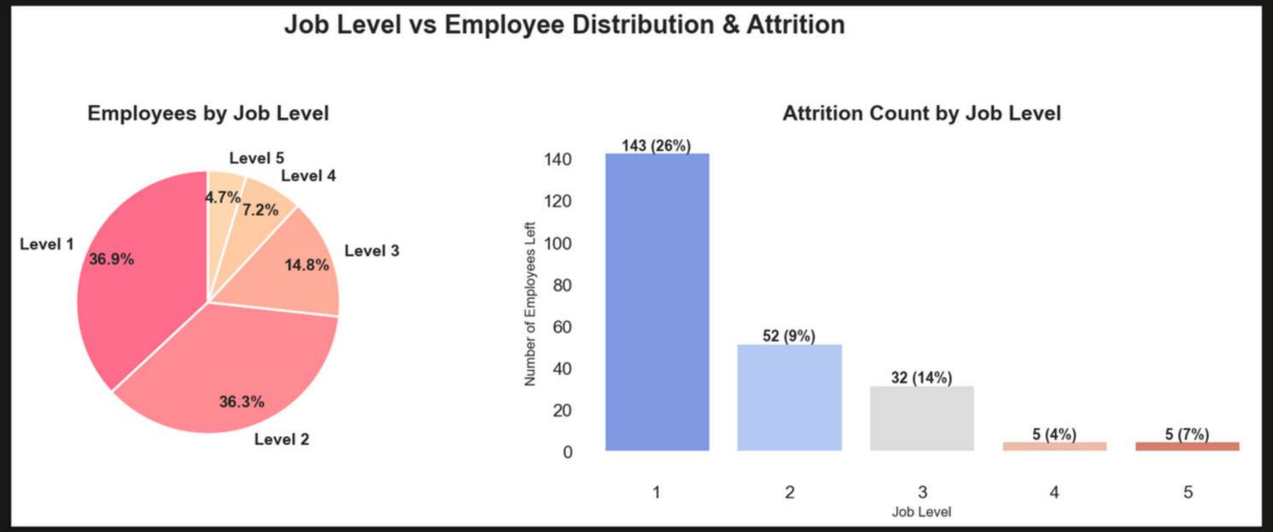
- Employees with High Environment Satisfaction surprisingly show the highest attrition rate (25%), suggesting other factors may outweigh satisfaction alone.
- Those with Low and Medium satisfaction levels have lower attrition rates (13-14%), indicating that satisfaction does not always directly correlate with retention.
- In terms of Job Roles, the highest attrition rates are observed among:
- Research Directors (39%)
- Sales Executives and Laboratory Technicians (both 23%)
- Managers and Healthcare Representatives also show notable attrition (16-17%).
- Roles like Manufacturing Directors and Sales Representatives experience significantly lower attrition rates (2-4%).

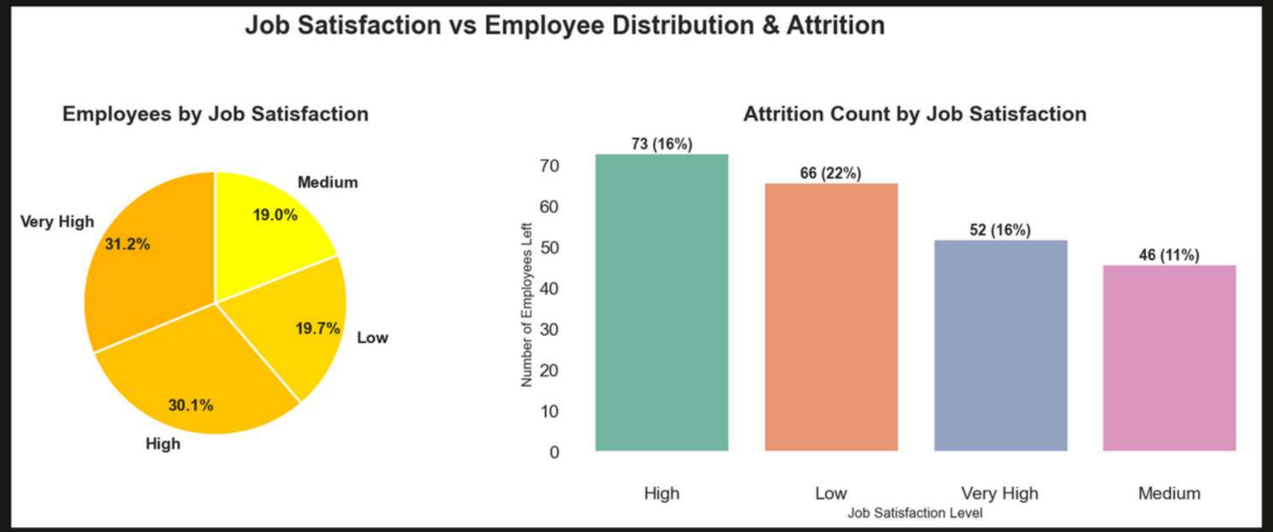




Influence of Job Level & Job Satisfaction on Attrition

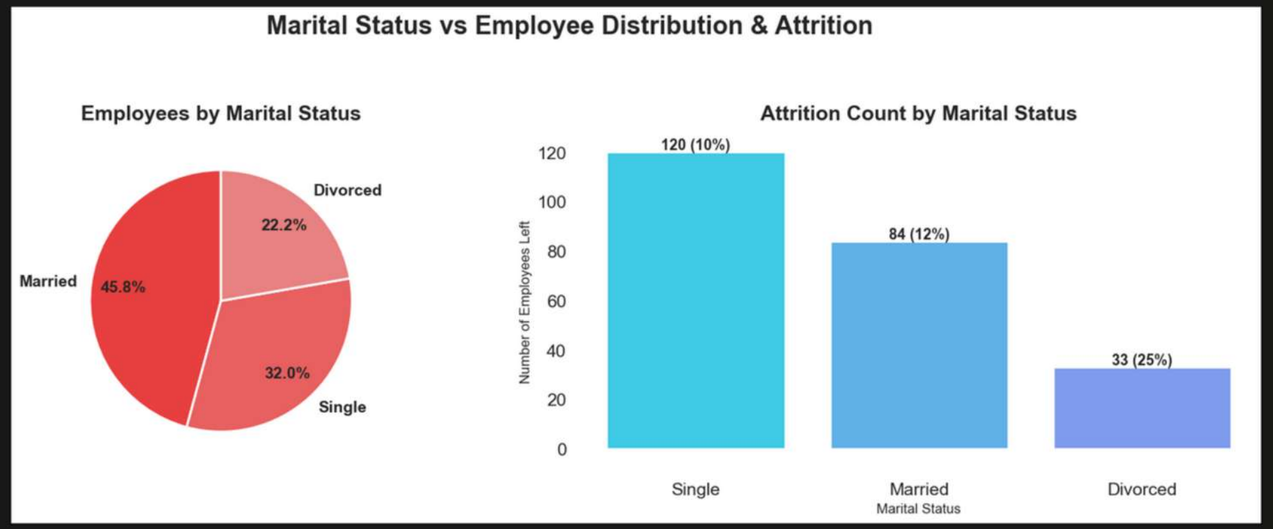
- Entry-level employees (Job Level 1) experience the highest attrition rate at 26%, indicating early-career employees are more likely to leave.
- Attrition significantly decreases with higher job levels, dropping to 4-7% for senior positions (Levels 4 and 5).
- Employees with Low Job Satisfaction show a notably high attrition rate of 22%.
- Interestingly, even employees reporting High and Very High satisfaction face attrition rates of 16%, while those with Medium satisfaction show the lowest attrition at 11%.

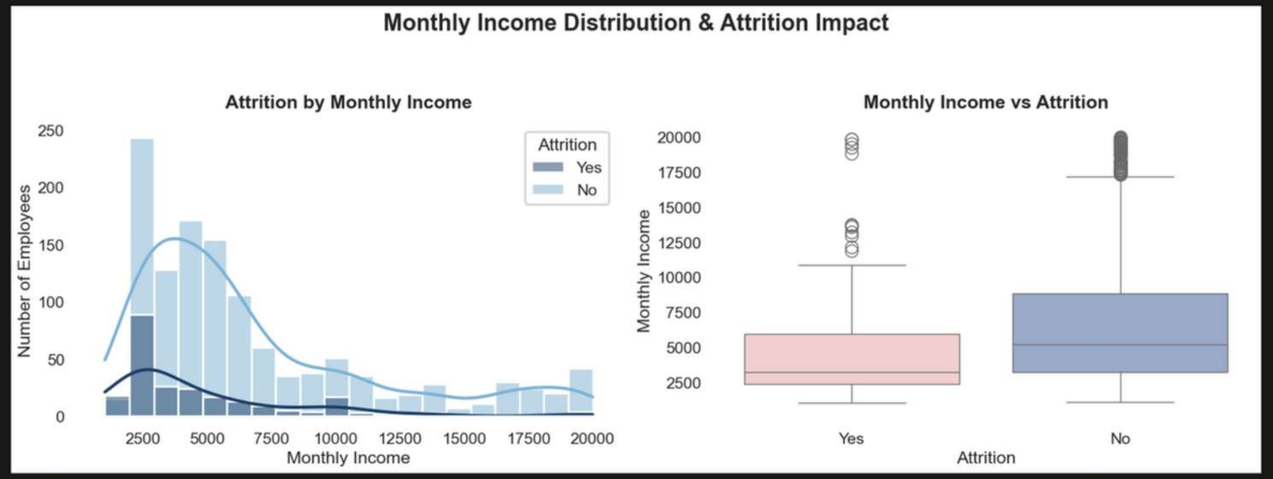




Impact of Marital Status & Monthly Income on Attrition

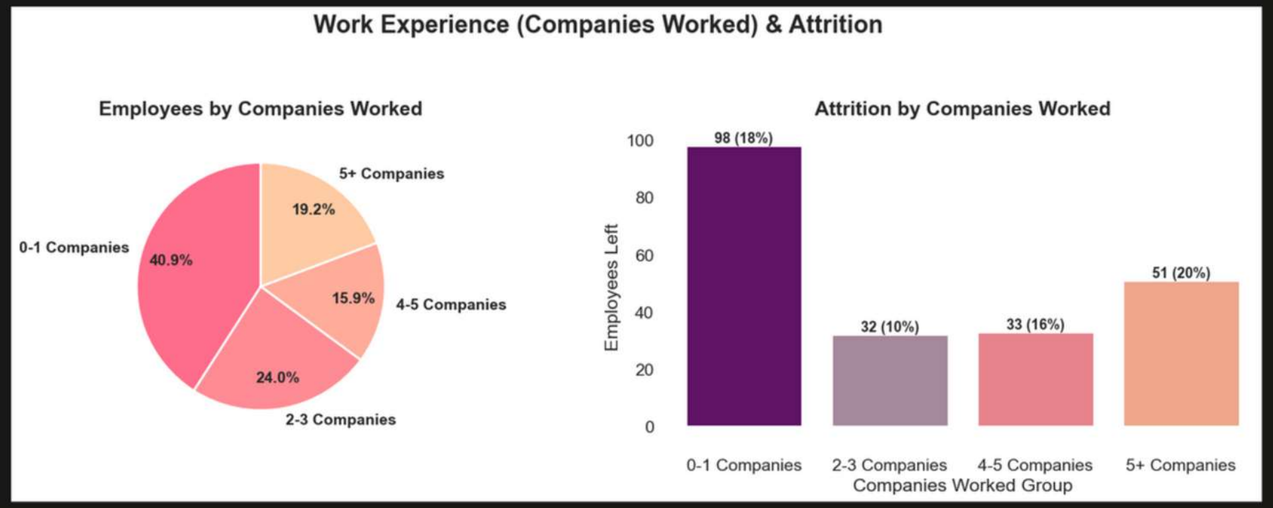
- Divorced employees show the highest attrition rate at 25%, followed by Single (10%) and Married employees (12%), indicating personal circumstances may influence turnover risk.
- Employees with lower monthly incomes are more prone to attrition, as seen in the higher concentration of leavers in lower salary ranges.
- Higher income levels correlate with reduced attrition, suggesting that competitive compensation plays a key role in employee retention.

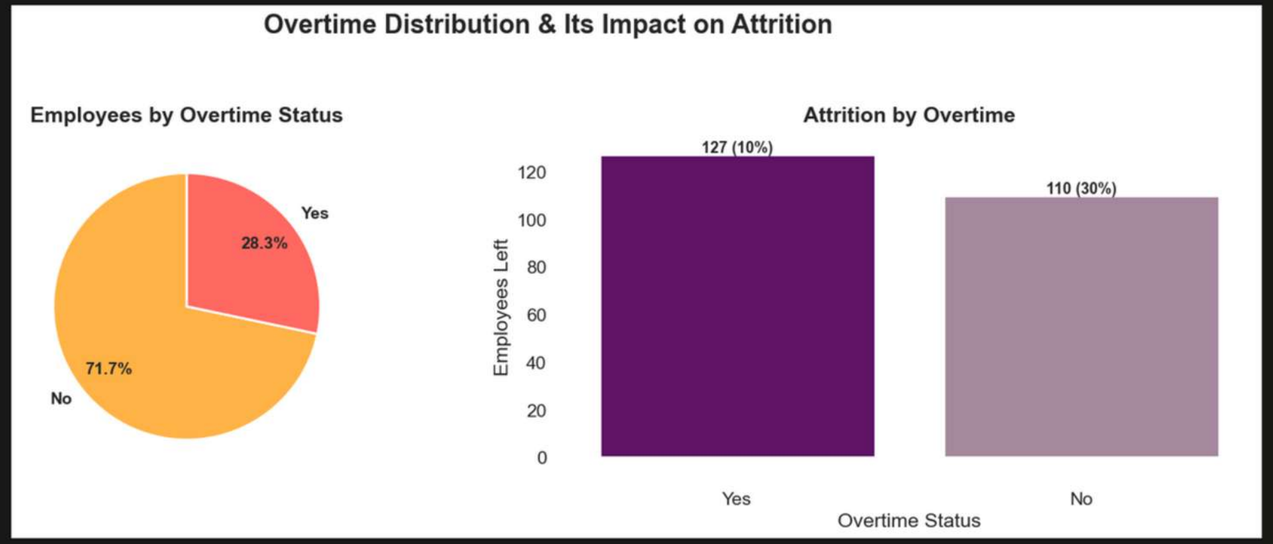




Impact of Work Experience & Overtime on Attrition

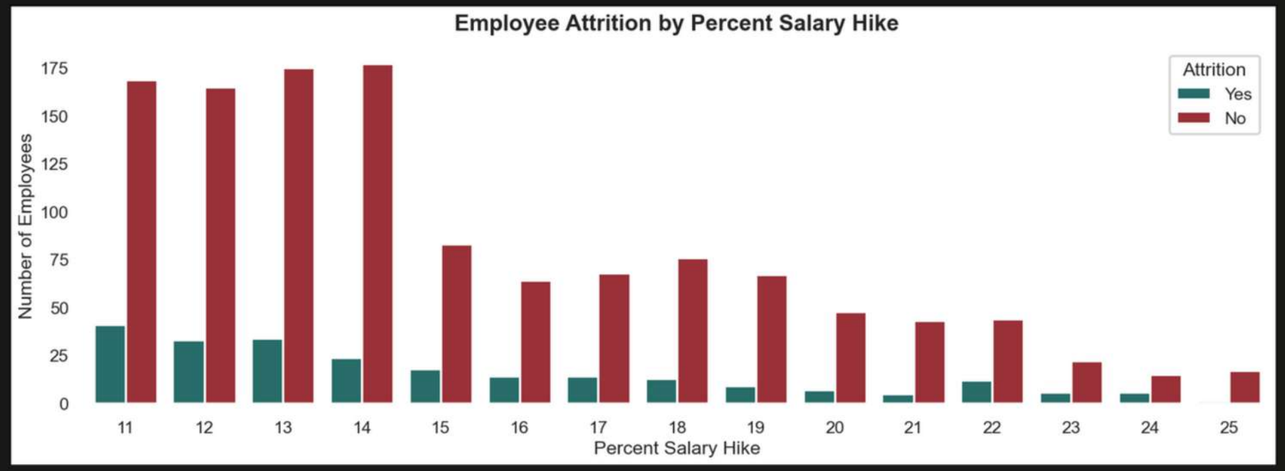
- Employees with experience in 5+ companies face the highest attrition rate (20%), indicating a trend of frequent job changes.
- Interestingly, even employees with 0-1 companies show a high attrition rate (18%), suggesting early exits or lack of long-term engagement.
- Those with moderate experience (2-3 companies) have the lowest attrition (10%), reflecting greater stability.
- Overtime has a significant impact on attrition—employees working overtime exhibit a 30% attrition rate, compared to just 10% for those who don't.
- Excessive workload and poor work-life balance appear to be key drivers of turnover among overtime workers.

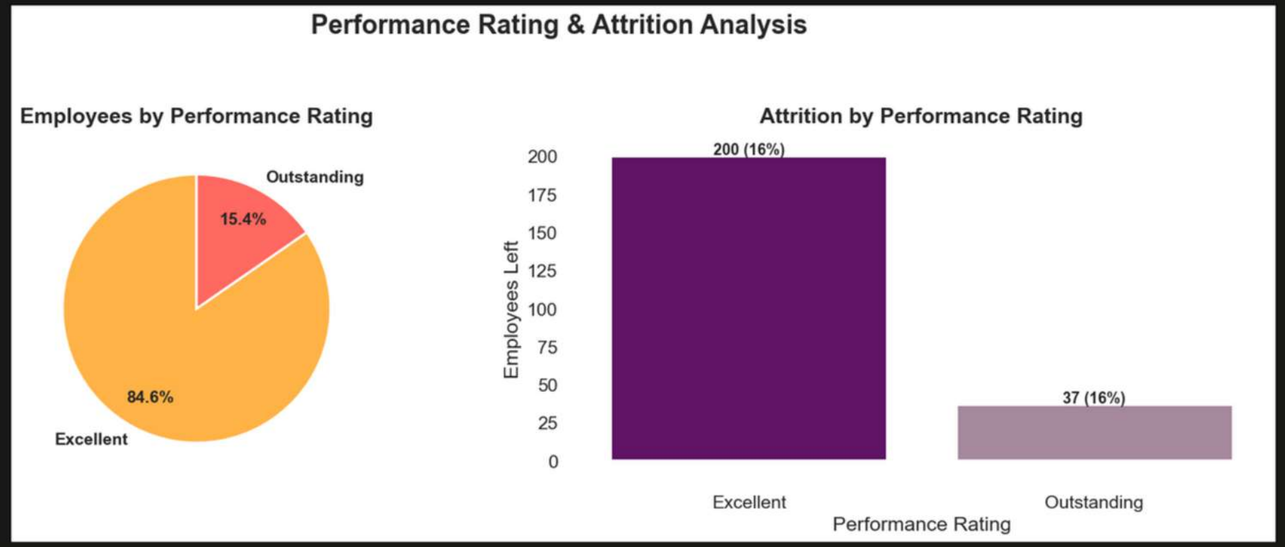




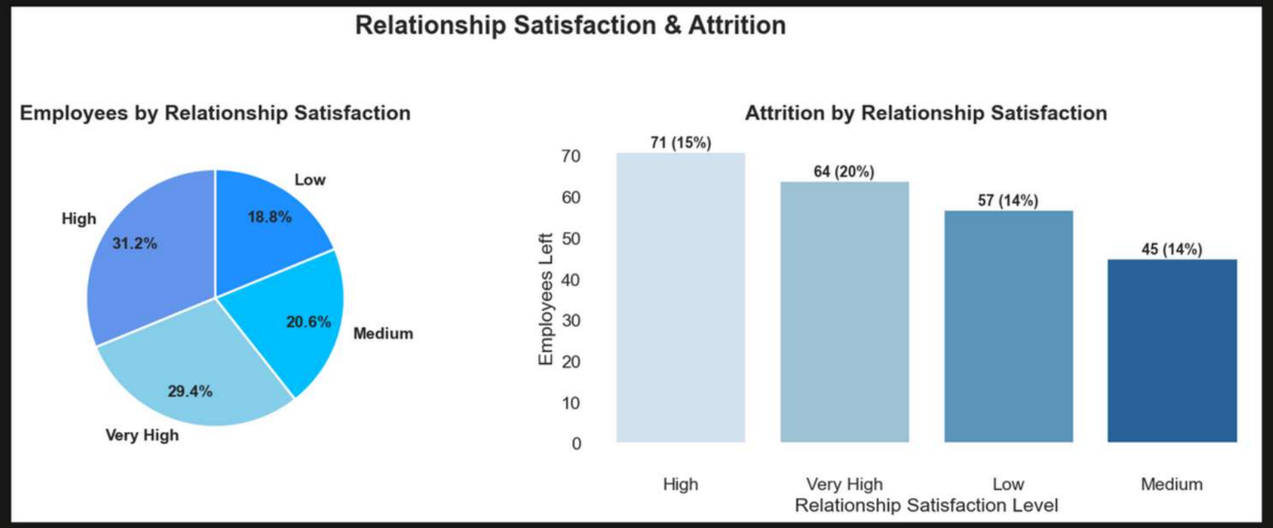
Impact of Salary Hike & Performance Rating on Attrition

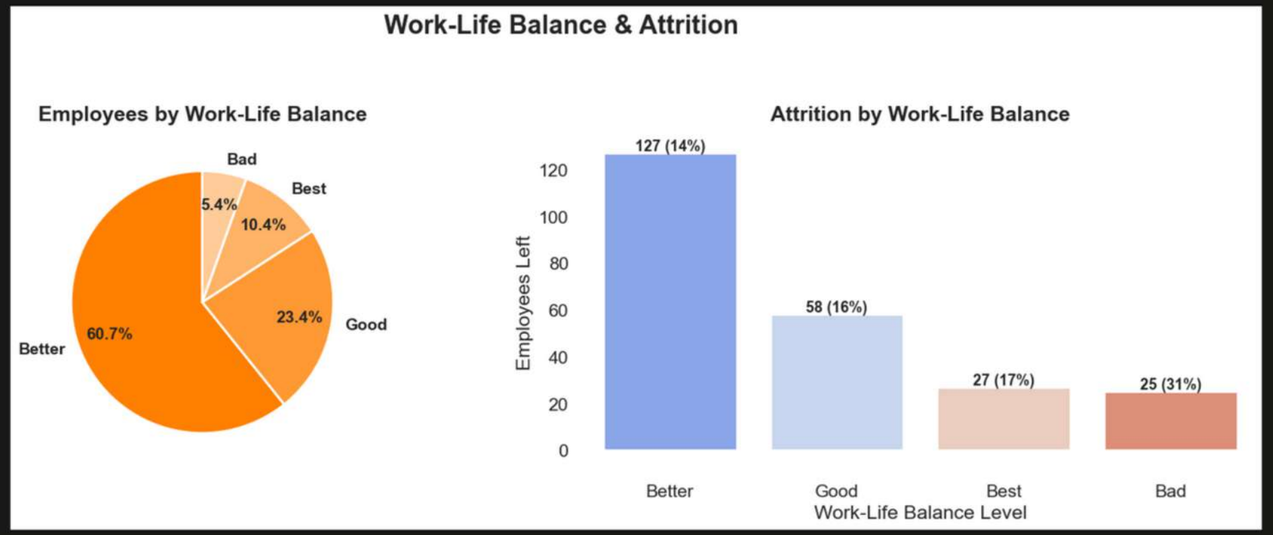
- Employees receiving lower salary hikes (11-14%) show higher attrition rates, indicating dissatisfaction with minimal compensation growth.
- As the percent salary hike increases, attrition decreases, highlighting the importance of competitive and motivating salary increments in retaining talent.
- Attrition remains consistent (16%) across both Excellent and Outstanding performance ratings, suggesting that high-performing employees are equally at risk of leaving if not adequately engaged or rewarded.





Impact of Relationship Satisfaction & Work-Life Balance on Attrition

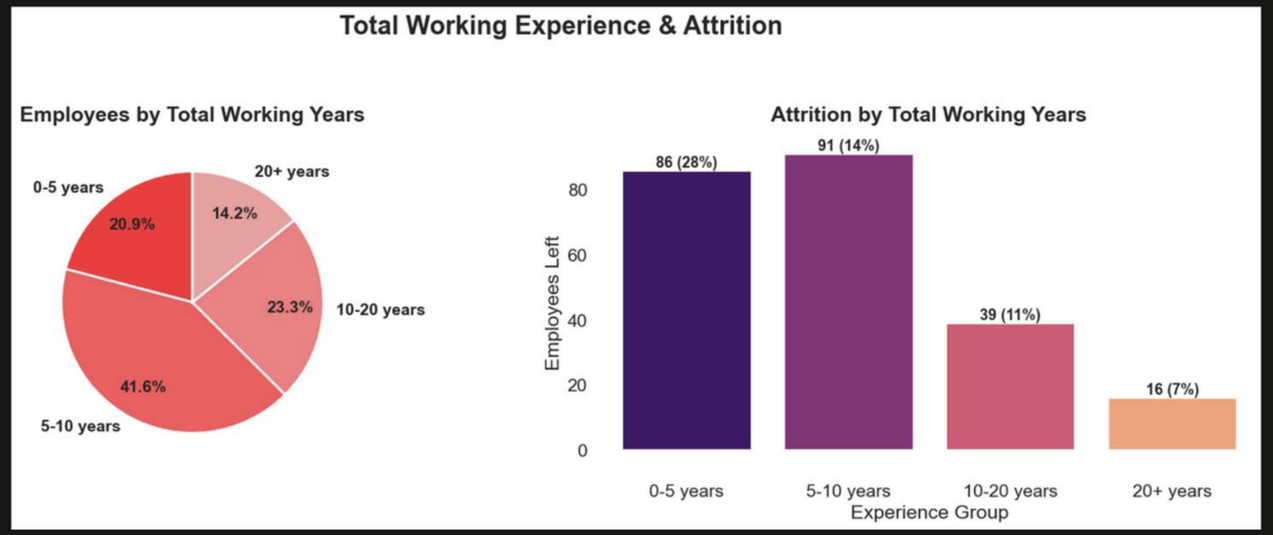


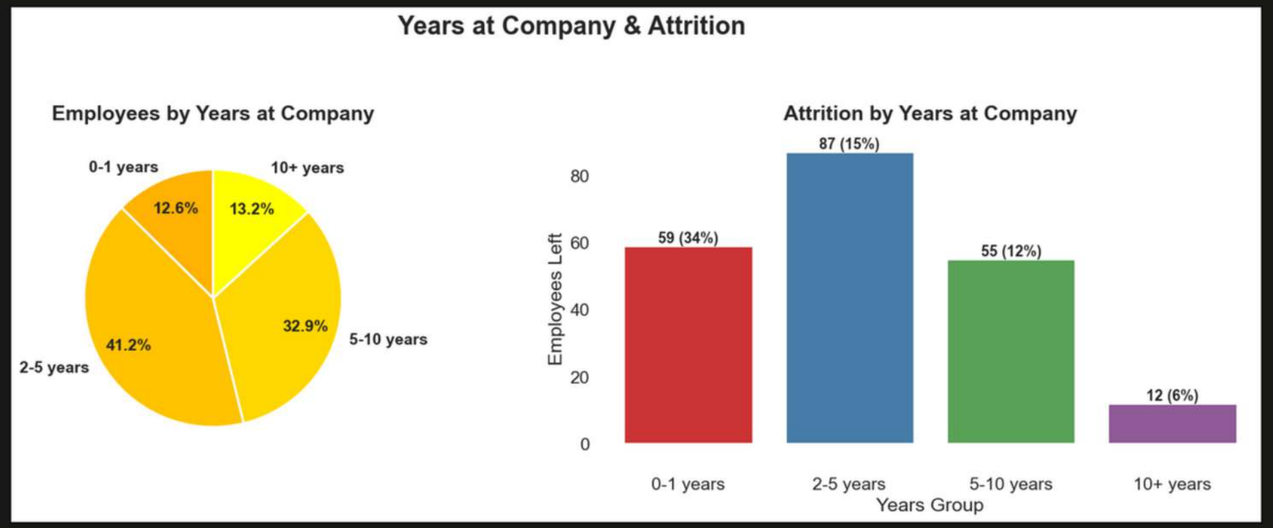


- Employees with Very High Relationship Satisfaction still show a notable attrition rate of 20%, indicating that good workplace relationships alone may not prevent turnover.
- Across all satisfaction levels, attrition ranges between 14% to 20%, suggesting relationship satisfaction has a moderate influence on retention.
- Poor Work-Life Balance strongly correlates with higher attrition—employees rating their balance as Bad face a 31% attrition rate.
- Even those with a Best balance experience 17% attrition, while employees reporting Better or Good balance show slightly lower rates (14-16%).

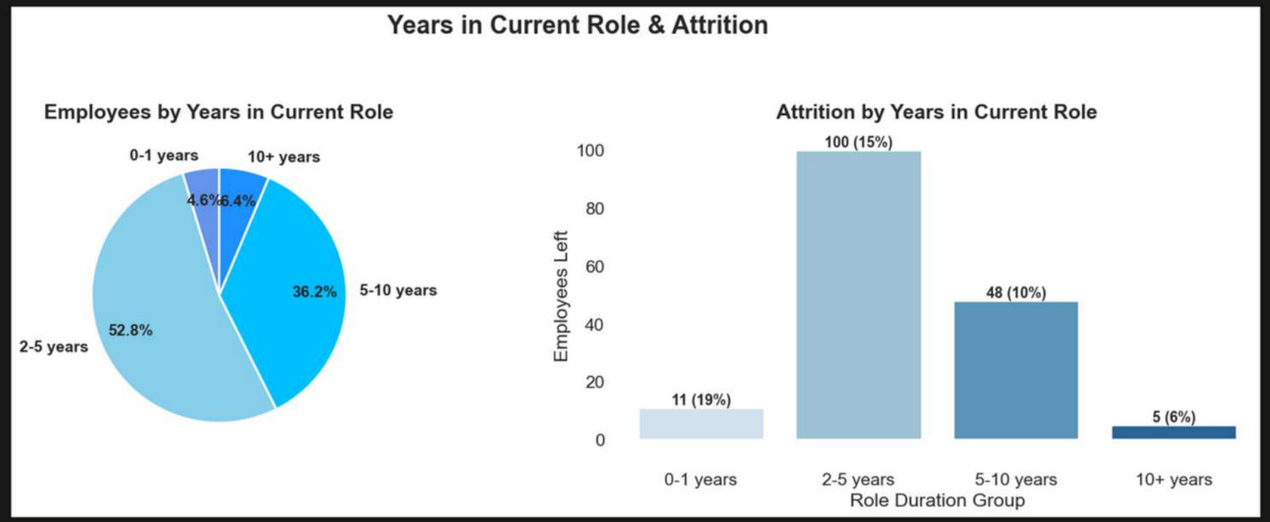
Impact of Total Experience & Company Tenure on Attrition

- Employees with 0-5 years of total working experience face the highest attrition rate (28%), indicating early-career professionals are more likely to switch jobs.
- Attrition steadily decreases with experience, dropping to just 7% for those with 20+ years of total work experience.
- Similarly, employees with 0-1 years at the company show a critical attrition rate of 34%, highlighting challenges in initial retention.
- Attrition reduces significantly for employees who stay beyond 5 years, with those at 10+ years showing only 6% attrition.



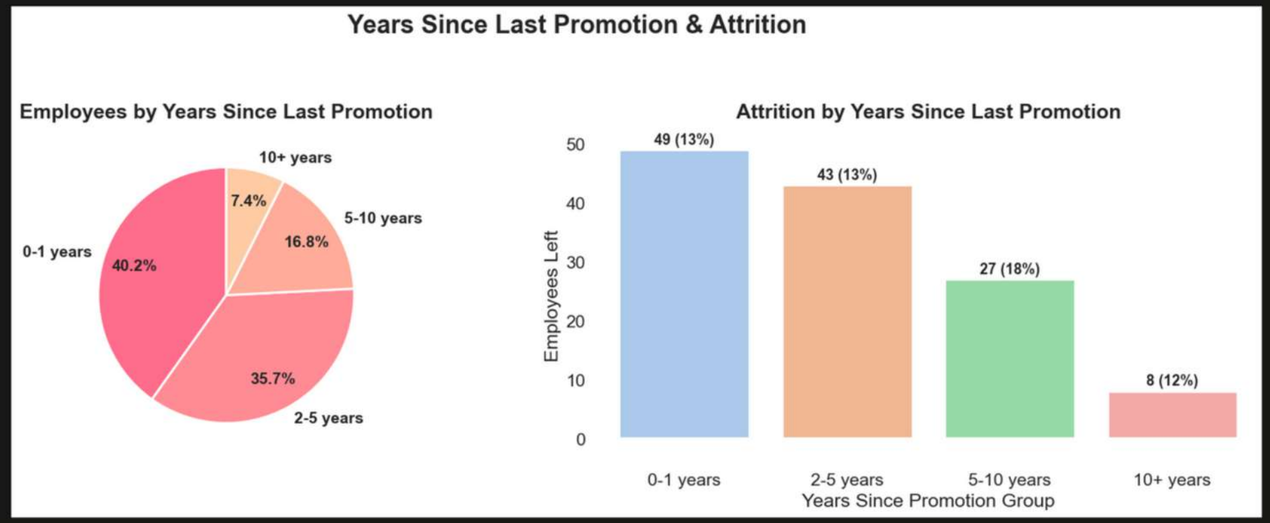


Role Duration, Promotions & Attrition Insights



Years in Current Role:

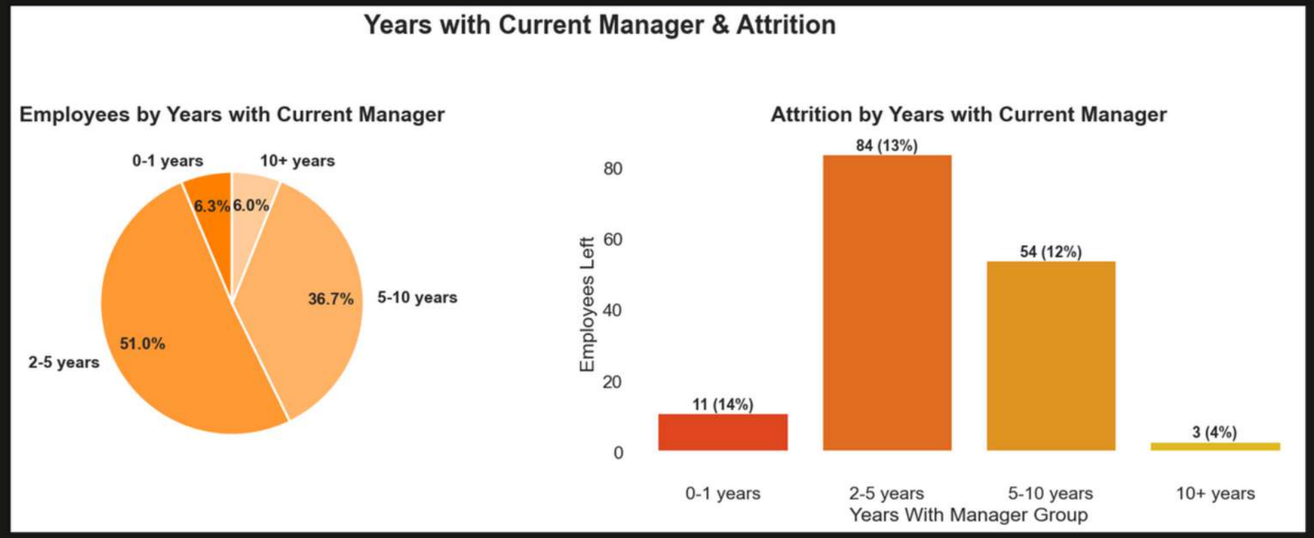
- Employees with 2-5 years in their current role face the highest attrition (15%).
- Attrition significantly drops for those staying 5+ years, indicating stability increases with tenure in role.
- Focus on engagement strategies for mid-tenure employees to improve retention.



Years Since Last Promotion:

- Employees who haven't been promoted for 2-5 years or 5-10 years show elevated attrition rates (13%-18%).
- Lack of career progression appears to drive turnover.
- Implementing regular growth opportunities and clear promotion paths can reduce attrition linked to stagnation.

Manager Tenure & Attrition Insights

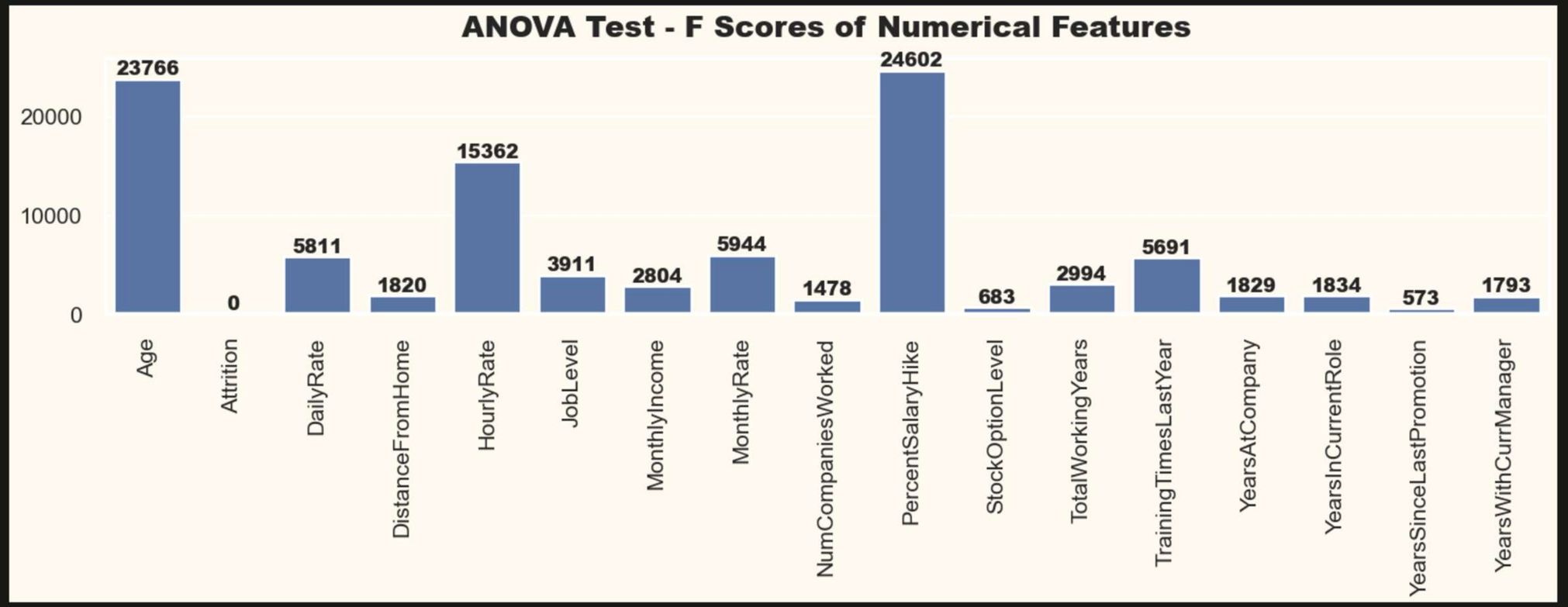


- Employees with 2-5 years under the same manager experience the highest attrition rate (13%), suggesting mid-term managerial relationships may lead to disengagement or stagnation.
- Attrition significantly decreases for employees managed for 5+ years, indicating that longer manager-employee relationships foster stability and retention.
- Employees with 0-1 year under a manager show moderate attrition, possibly due to adaptation challenges in early stages.

Recommendation:

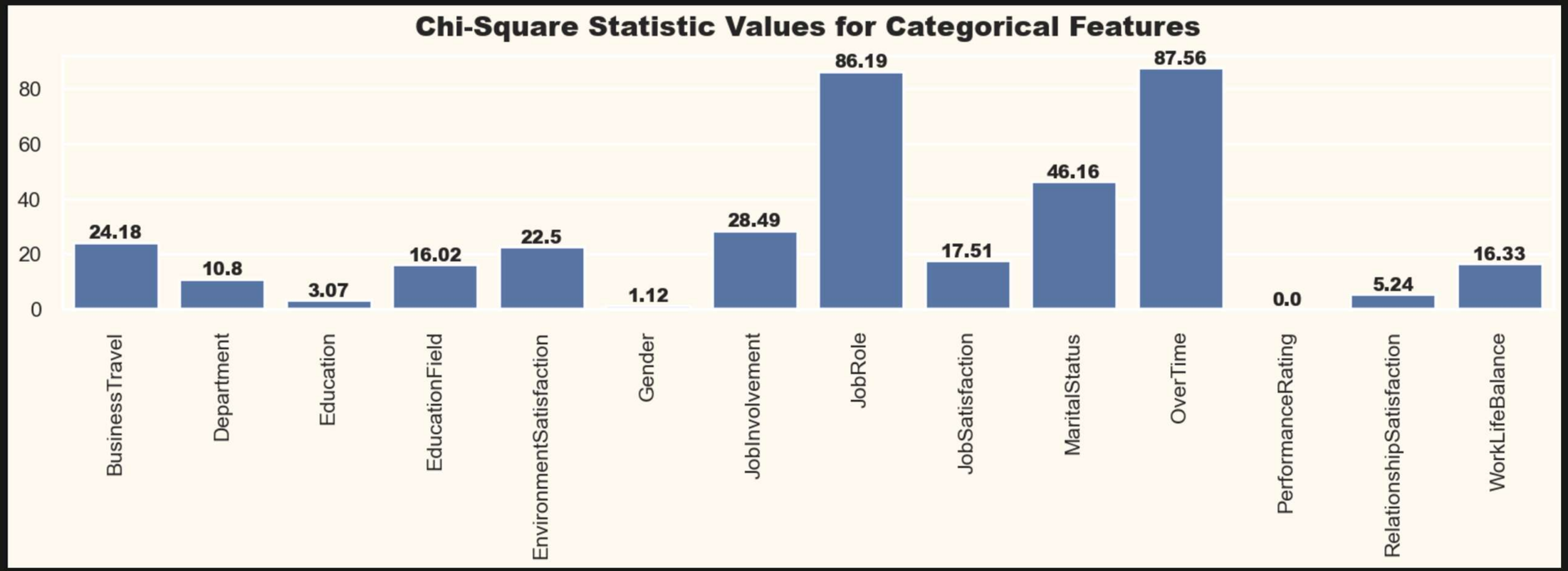
- Encourage regular feedback sessions and career development discussions, especially for employees in the 2-5 year range with their manager.
- Strengthen leadership training to improve long-term manager-employee engagement.

Key Numerical Drivers of Employee Attrition (ANOVA Test)



- Percent Salary Hike and Age are the top numerical factors influencing attrition.
- Hourly Rate and Daily Rate also show significant impact.
- Variables like Years Since Last Promotion and Stock Option Level have minimal influence.

Key Categorical Drivers of Employee Attrition (Chi-Square Test)



- **OverTime** and **Job Role** are the most significant categorical factors impacting attrition.
- **Marital Status** and **Job Involvement** also show strong influence.
- Factors like **Gender**, **Education**, and **Performance Rating** have minimal impact.

Feature Importance Analysis

Numerical Features (ANOVA Test)

	Features	F_Score	P_value
0	PercentSalaryHike	2.460251e+04	0.0000000000000000000
1	Age	2.376693e+04	0.0000000000000000000
2	HourlyRate	1.536212e+04	0.0000000000000000000
3	MonthlyRate	5.944089e+03	0.0000000000000000000
4	DailyRate	5.811797e+03	0.0000000000000000000
5	TrainingTimesLastYear	5.691402e+03	0.0000000000000000000
6	JobLevel	3.911332e+03	0.0000000000000000000
7	TotalWorkingYears	2.994906e+03	0.0000000000000000000
8	MonthlyIncome	2.804460e+03	0.0000000000000000000
9	YearsInCurrentRole	1.834262e+03	0.0000000000000000000
10	YearsAtCompany	1.829443e+03	0.0000000000000000000
11	DistanceFromHome	1.820615e+03	0.0000000000000000000
12	YearsWithCurrManager	1.793291e+03	0.0000000000000000000
13	NumCompaniesWorked	1.478189e+03	0.0000000000000000000
14	StockOptionLevel	6.830696e+02	0.0000000000000000000
15	YearsSinceLastPromotion	5.738964e+02	0.0000000000000000000
16	Attrition	-1.469261e-30	nan

Categorical Features (Chi-Square Test)

	Features	Chi2_Statistic	P_value
0	OverTime	87.564294	0.00000000000000000001
1	JobRole	86.190254	0.0000000000000275248
2	MaritalStatus	46.163677	0.0000000009455511060
3	JobInvolvement	28.492021	0.00000286318063671342
4	BusinessTravel	24.182414	0.00000560861447644993
5	EnvironmentSatisfaction	22.503881	0.00005123468906289433
6	JobSatisfaction	17.505077	0.00055630045103875563
7	WorkLifeBalance	16.325097	0.00097256988453488225
8	EducationField	16.024674	0.00677398013902521298
9	Department	10.796007	0.00452560657447963373
10	RelationshipSatisfaction	5.241068	0.15497244371052626422
11	Education	3.073961	0.54552533765659494414
12	Gender	1.116967	0.29057244902890855265
13	PerformanceRating	0.000155	0.99007454659345761616

Top Numerical Drivers:

- **Percent Salary Hike, Age, Hourly Rate, and Monthly Rate show the highest F-Scores with strong significance (p-value = 0).**

Top Categorical Drivers:

- **Overtime, Job Role, and Marital Status are critical factors based on Chi-Square values with highly significant p-values.**

Data Preprocessing & Feature Correlation Analysis

Label Encoding & One-Hot Encoding:

- Converted categorical variables into numerical formats to make them suitable for machine learning algorithms.

Data Cleaning:

- Removed duplicate features and redundant data to ensure model efficiency.

Feature Selection:

- Selected relevant features based on correlation thresholds to focus on impactful variables.

Train-Test Split & Scaling:

- Standardized the data and split it for training and evaluation.

Correlation Heatmap:

- Visualized relationships between features to identify strong positive or negative correlations, ensuring reduced multicollinearity and better model performance.

```
# Label Encoding 'Attrition'
label = LabelEncoder()
data["Attrition"] = label.fit_transform(data["Attrition"])

# One-Hot Encoding
dummy_col = [col for col in data.drop('Attrition', axis=1).columns if data[col].nunique() < 20]
data = pd.get_dummies(data, columns=dummy_col, drop_first=True, dtype='uint8')

# Remove duplicate features and rows
data = data.T.drop_duplicates().T
data.drop_duplicates(inplace=True)

# Feature Selection
feature_correlation = data.drop('Attrition', axis=1).corrwith(data.Attrition).sort_values()
model_col = feature_correlation[np.abs(feature_correlation) > 0.02].index

X = data[model_col]
y = data.Attrition

# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Scaling
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)

# Correlation Heatmap
plt.figure(figsize=(12,8))
sns.heatmap(data[model_col].corr(), annot=False, cmap='coolwarm')
plt.title("Feature Correlation Heatmap")
plt.show()
```

Train-Test Split & Class Distribution Check

Feature & Target Definition:

- Separated independent variables (features) and the target variable (Attrition).

Stratified Splitting:

- Maintained the original class distribution (Attrition vs. Non-Attrition) across both training and test sets to avoid bias.

Feature Scaling:

- Applied standardization to normalize feature values, enhancing model performance and convergence.

Class Balance Verification:

Checked class distribution to confirm consistent attrition rates:

- Train Set: 83.87% Staying | 16.13% Leaving
- Test Set: 83.90% Staying | 16.10% Leaving

```python
# Define Features and Target
X = data.drop('Attrition', axis=1)
y = data['Attrition']

# Perform Stratified Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y)

# Standardize Features
scaler = StandardScaler()
X_train_std = scaler.fit_transform(X_train)
X_test_std = scaler.transform(X_test)

# Check Class Distribution Function
def display_class_distribution(y_train, y_test):
    print("======= CLASS DISTRIBUTION =======")
    for dataset, label in zip([y_train, y_test], ['TRAIN', 'TEST']):
        total = dataset.shape[0]
        staying = dataset.value_counts()[0]
        leaving = dataset.value_counts()[1]
        print(f"\n----- {label} SET -----")
        print(f"Staying Rate : {(staying / total) * 100:.2f}%")
        print(f"Leaving Rate : {(leaving / total) * 100:.2f}%")

# Display class balance
display_class_distribution(y_train, y_test)
```

Logistic Regression Model

```python
# Initialize and Train Logistic Regression
lr_clf = LogisticRegression(solver='liblinear', penalty='l1', random_state=42)
lr_clf.fit(X_train_std, y_train)

# Evaluate Model
evaluate(lr_clf, X_train_std, X_test_std, y_train, y_test)


# Use predict_proba for probability estimates
y_scores = lr_clf.predict_proba(X_test_std)[:, 1]

# Precision-Recall Curve
precisions, recalls, thresholds = precision_recall_curve(y_test, y_scores)

plt.figure(figsize=(16, 12))

# Plot Precision vs Recall vs Threshold
plt.subplot(2, 2, 1)
plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
plt.xlabel("Threshold")
plt.ylabel("Score")
plt.title("Precision & Recall vs Threshold")
plt.legend()
plt.grid(True)

# PR Curve
plt.subplot(2, 2, 2)
plt.plot(recalls, precisions, marker='.')
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.grid(True)
```

```python
# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_scores)
plt.subplot(2, 2, 3)
plt.plot(fpr, tpr, linewidth=2, label=f"AUC = {roc_auc_score(y_test, y_scores):.3f}")
plt.plot([0, 1], [0, 1], "k--")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

# Store ROC-AUC Scores

scores_dict = {
    'Logistic Regression': {
        'Train': roc_auc_score(y_train, lr_clf.predict_proba(X_train_std)[:,1]),
        'Test': roc_auc_score(y_test, y_scores),
    }
}

# Display Scores
print(f"ROC-AUC (Train) : {scores_dict['Logistic Regression']['Train']:.4f}")
print(f"ROC-AUC (Test)  : {scores_dict['Logistic Regression']['Test']:.4f}")
```

Logistic Regression Evaluation

A Logistic Regression model was trained with L1 regularization.

- Model performance was evaluated using:
- Precision-Recall Curve

ROC Curve

Confusion Matrix

Classification Report

The model achieved:

- Training Accuracy: 92.7%
- Testing Accuracy: 86.4%

ROC-AUC scores were calculated for both training and test sets to assess model robustness.

```
TRAINIG RESULTS:
==============================
CONFUSION MATRIX:
[[847  16]
 [ 59 107]]
ACCURACY SCORE:
0.9271
CLASSIFICATION REPORT:
                    0           1  accuracy    macro avg  weighted avg
precision    0.934879    0.869919  0.927114     0.902399      0.924399
recall       0.981460    0.644578  0.927114     0.813019      0.927114
f1-score     0.957603    0.740484  0.927114     0.849044      0.922577
support    863.000000  166.000000  0.927114  1029.000000   1029.000000
TESTING RESULTS:
==============================
CONFUSION MATRIX:
[[351  19]
 [ 41  30]]
ACCURACY SCORE:
0.8639
CLASSIFICATION REPORT:
                    0           1  accuracy    macro avg  weighted avg
precision    0.895408    0.612245  0.863946     0.753827      0.849820
recall       0.948649    0.422535  0.863946     0.685592      0.863946
f1-score     0.921260    0.500000  0.863946     0.710630      0.853438
support    370.000000   71.000000  0.863946   441.000000    441.000000
```

Random Forest Classifier Model

```python
# Initial Training (Before Hyperparameter Tuning)
rf_clf = RandomForestClassifier(n_estimators=100, bootstrap=False, random_state=42)
rf_clf.fit(X_train, y_train)

print("Initial Random Forest Evaluation (Default Params)")
evaluate(rf_clf, X_train, X_test, y_train, y_test)


# Hyperparameter Tuning with GridSearchCV


param_grid = {
    'n_estimators': [100, 300],
    'max_features': ['auto', 'sqrt'],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'bootstrap': [True, False]
}

grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    scoring='roc_auc',
    cv=3,
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)

print(f"Best Parameters: {grid_search.best_params_}")

# Train with Best Parameters

best_rf = grid_search.best_estimator_
best_rf.fit(X_train, y_train)

print("\n✨ Random Forest Evaluation (Tuned Model)")
evaluate(best_rf, X_train, X_test, y_train, y_test)
```

```python
# Precision-Recall & ROC Curves

y_scores = best_rf.predict_proba(X_test)[:, 1]
precisions, recalls, thresholds = precision_recall_curve(y_test, y_scores)
fpr, tpr, _ = roc_curve(y_test, y_scores)

plt.figure(figsize=(16, 12))

# Precision vs Threshold
plt.subplot(2, 2, 1)
plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
plt.title("Precision & Recall vs Threshold")
plt.xlabel("Threshold")
plt.legend()
plt.grid(True)

# PR Curve
plt.subplot(2, 2, 2)
plt.plot(recalls, precisions, marker='.')
plt.title("Precision-Recall Curve")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.grid(True)

# ROC Curve
plt.subplot(2, 2, 3)
plt.plot(fpr, tpr, label=f"AUC = {roc_auc_score(y_test, y_scores):.3f}")
plt.plot([0,1],[0,1],'k--')
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

# Store ROC-AUC Scores

scores_dict['Random Forest'] = {
    'Train': roc_auc_score(y_train, best_rf.predict_proba(X_train)[:,1]),
    'Test': roc_auc_score(y_test, y_scores),
}

print(f"Train ROC-AUC: {scores_dict['Random Forest']['Train']:.4f}")
print(f"Test  ROC-AUC: {scores_dict['Random Forest']['Test']:.4f}")
```

```python
# Feature Importance Plot

def feature_imp(df, model):
    """
    Returns a DataFrame of feature importances sorted in descending order.

    Parameters:
    df : DataFrame
        The feature set used for training.
    model : Trained model with feature_importances_ attribute.

    Returns:
    DataFrame with features and their importance scores.
    """
    fi = pd.DataFrame()
    fi["feature"] = df.columns
    fi["importance"] = model.feature_importances_
    return fi.sort_values(by="importance", ascending=False)

feature_importance_df = feature_imp(X, best_rf).head(30)
feature_importance_df.set_index('feature', inplace=True)

feature_importance_df.plot(kind='barh', figsize=(10, 8))
plt.title('Top 30 Feature Importances - Random Forest')
plt.gca().invert_yaxis()  # Highest importance on top
plt.grid(True)
plt.show()
```

Random Forest Classifier Evaluation

Model evaluation included:

- Precision-Recall & ROC Curves
- Confusion Matrix
- Classification Report
- Feature Importance Plot

The tuned model achieved:

- Training Accuracy: 100% (Default), improved after tuning
- Testing Accuracy: 83.2% (Default), improved after tuning

```
Initial Random Forest Evaluation (Default Params)
TRAINIG RESULTS:
==============================
CONFUSION MATRIX:
[[863   0]
 [  0 166]]
ACCURACY SCORE:
1.0000
CLASSIFICATION REPORT:
              0      1  accuracy  macro avg  weighted avg
precision   1.0    1.0       1.0        1.0           1.0
recall      1.0    1.0       1.0        1.0           1.0
f1-score    1.0    1.0       1.0        1.0           1.0
support   863.0  166.0       1.0     1029.0        1029.0
TESTING RESULTS:
==============================
CONFUSION MATRIX:
[[359  11]
 [ 63   8]]
ACCURACY SCORE:
0.8322
CLASSIFICATION REPORT:
                   0          1  accuracy   macro avg  weighted avg
precision   0.850711   0.421053    0.8322    0.635882      0.781537
recall      0.970270   0.112676    0.8322    0.541473      0.832200
...
precision   0.848131   0.461538  0.836735    0.654835      0.785890
recall      0.981081   0.084507  0.836735    0.532794      0.836735
f1-score    0.909774   0.142857  0.836735    0.526316      0.786302
support   370.000000  71.000000  0.836735  441.000000    441.000000
```

Support Vector Machine Model

```python
# Initial Training with Linear Kernel
svm_clf = SVC(kernel='linear', probability=True, random_state=42)
svm_clf.fit(X_train_std, y_train)

print("Initial SVM Evaluation (Linear Kernel)")
evaluate(svm_clf, X_train_std, X_test_std, y_train, y_test)


# Hyperparameter Tuning using GridSearchCV
param_grid = [
    {'C': [1, 10, 100], 'kernel': ['linear']},
    {'C': [1, 10, 100], 'gamma': [0.001, 0.0001], 'kernel': ['rbf']}
]

grid_search = GridSearchCV(
    SVC(probability=True, random_state=42),
    param_grid=param_grid,
    scoring='roc_auc',
    cv=3,
    refit=True,
    verbose=1
)

grid_search.fit(X_train_std, y_train)
print(f"Best Parameters: {grid_search.best_params_}")

# 3️⃣ Train Final SVM Model with Best Params
best_svm = grid_search.best_estimator_
best_svm.fit(X_train_std, y_train)

print("\nSVM Evaluation (Tuned Model)")
evaluate(best_svm, X_train_std, X_test_std, y_train, y_test)


# Precision-Recall & ROC Curves
# For SVM, use decision_function or predict_proba (if enabled)
y_scores = best_svm.decision_function(X_test_std)

precisions, recalls, thresholds = precision_recall_curve(y_test, y_scores)
fpr, tpr, _ = roc_curve(y_test, y_scores)

plt.figure(figsize=(16, 12))
```

```python
# Precision vs Threshold
plt.subplot(2, 2, 1)
plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
plt.title("Precision & Recall vs Threshold")
plt.xlabel("Threshold")
plt.legend()
plt.grid(True)

# PR Curve
plt.subplot(2, 2, 2)
plt.plot(recalls, precisions, marker='.')
plt.title("Precision-Recall Curve")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.grid(True)

# ROC Curve
plt.subplot(2, 2, 3)
plt.plot(fpr, tpr, label=f"AUC = {roc_auc_score(y_test, y_scores):.3f}")
plt.plot([0,1],[0,1],'k--')
plt.title("ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()


# Store ROC-AUC Scores
scores_dict['Support Vector Machine'] = {
    'Train': roc_auc_score(y_train, best_svm.decision_function(X_train_std)),
    'Test': roc_auc_score(y_test, y_scores),
}

print(f"Train ROC-AUC: {scores_dict['Support Vector Machine']['Train']:.4f}")
print(f"Test  ROC-AUC: {scores_dict['Support Vector Machine']['Test']:.4f}")
```

Support Vector Machine Evaluation

Model performance was evaluated using:

- Precision-Recall Curve
- ROC Curve
- Confusion Matrix
- Classification Report

The model achieved:

- Training Accuracy: 94.6%
- Testing Accuracy: 86.6%

```
Initial SVM Evaluation (Linear Kernel)
TRAINIG RESULTS:
==============================
CONFUSION MATRIX:
[[855   8]
 [ 48 118]]
ACCURACY SCORE:
0.9456
CLASSIFICATION REPORT:
                    0           1  accuracy    macro avg  weighted avg
precision    0.946844    0.936508  0.945578     0.941676      0.945176
recall       0.990730    0.710843  0.945578     0.850787      0.945578
f1-score     0.968290    0.808219  0.945578     0.888255      0.942467
support    863.000000  166.000000  0.945578  1029.000000   1029.000000
TESTING RESULTS:
==============================
CONFUSION MATRIX:
[[347  23]
 [ 44  27]]
ACCURACY SCORE:
0.8481
CLASSIFICATION REPORT:
                    0           1  accuracy   macro avg  weighted avg
precision    0.887468    0.540000  0.848073    0.713734      0.831526
recall       0.937838    0.380282  0.848073    0.659060      0.848073
...
precision    0.880196    0.687500  0.866213    0.783848      0.849172
recall       0.972973    0.309859  0.866213    0.641416      0.866213
f1-score     0.924262    0.427184  0.866213    0.675723      0.844234
support    370.000000   71.000000  0.866213  441.000000    441.000000
```

XGBoost Model

```python
# Initialize and Train XGBoost
xgb_clf = XGBClassifier(eval_metric='logloss', use_label_encoder=False, random_state=42)
xgb_clf.fit(X_train, y_train)

print("XGBoost Evaluation")
evaluate(xgb_clf, X_train, X_test, y_train, y_test)

# Precision-Recall & ROC Curves
# Use predict_proba for probability-based metrics
y_scores = xgb_clf.predict_proba(X_test)[:, 1]

precisions, recalls, thresholds = precision_recall_curve(y_test, y_scores)
fpr, tpr, _ = roc_curve(y_test, y_scores)

plt.figure(figsize=(16, 12))

# Precision vs Threshold
plt.subplot(2, 2, 1)
plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
plt.title("Precision & Recall vs Threshold (XGBoost)")
plt.xlabel("Threshold")
plt.legend()
plt.grid(True)

# PR Curve
plt.subplot(2, 2, 2)
plt.plot(recalls, precisions, marker='.')
plt.title("Precision-Recall Curve (XGBoost)")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.grid(True)

# ROC Curve
plt.subplot(2, 2, 3)
plt.plot(fpr, tpr, label=f"AUC = {roc_auc_score(y_test, y_scores):.3f}")
plt.plot([0,1],[0,1],'k--')
plt.title("ROC Curve (XGBoost)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)

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

```python
# Store ROC-AUC Scores
scores_dict['XGBoost'] = {
    'Train': roc_auc_score(y_train, xgb_clf.predict_proba(X_train)[:,1]),
    'Test': roc_auc_score(y_test, y_scores),
}

print(f"Train ROC-AUC: {scores_dict['XGBoost']['Train']:.4f}")
print(f"Test  ROC-AUC: {scores_dict['XGBoost']['Test']:.4f}")


# Feature Importance Plot
feature_importance_df = feature_imp(X, xgb_clf).head(30)
feature_importance_df.set_index('feature', inplace=True)

feature_importance_df.plot(kind='barh', figsize=(10, 8))
plt.title('Top 30 Feature Importances - XGBoost')
plt.gca().invert_yaxis()
plt.grid(True)
plt.show()
```

XGBoost Evaluation

Model performance was evaluated using:

- Precision-Recall Curve
- ROC Curve
- Confusion Matrix
- Classification Report

The model achieved:

- Training Accuracy: 94.6%
- Testing Accuracy: 86.6%

```
XGBoost Evaluation
TRAINIG RESULTS:
==============================
CONFUSION MATRIX:
[[863   0]
 [  0 166]]
ACCURACY SCORE:
1.0000
CLASSIFICATION REPORT:
              0      1  accuracy  macro avg  weighted avg
precision   1.0    1.0       1.0        1.0           1.0
recall      1.0    1.0       1.0        1.0           1.0
f1-score    1.0    1.0       1.0        1.0           1.0
support   863.0  166.0       1.0     1029.0        1029.0
TESTING RESULTS:
==============================
CONFUSION MATRIX:
[[357  13]
 [ 53  18]]
ACCURACY SCORE:
0.8503
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.870732   0.580645   0.85034    0.725688      0.824028
recall       0.964865   0.253521   0.85034    0.609193      0.850340
f1-score     0.915385   0.352941   0.85034    0.634163      0.824832
support    370.000000  71.000000   0.85034  441.000000    441.000000
```

LightGBM Classification Model

```
# Initialize and Train LightGBM
lgb_clf = LGBMClassifier(scale_pos_weight=(len(y_train) - sum(y_train)) / sum(y_train), verbose=-1)
lgb_clf.fit(X_train, y_train)

print("LightGBM Evaluation")
evaluate(lgb_clf, X_train, X_test, y_train, y_test)

# Precision-Recall & ROC Curves
# Use predict_proba for probability outputs
y_scores = lgb_clf.predict_proba(X_test)[:, 1]

precisions, recalls, thresholds = precision_recall_curve(y_test, y_scores)
fpr, tpr, _ = roc_curve(y_test, y_scores)

plt.figure(figsize=(16, 12))

# Precision vs Threshold
plt.subplot(2, 2, 1)
plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
plt.title("Precision & Recall vs Threshold (LightGBM)")
plt.xlabel("Threshold")
plt.legend()
plt.grid(True)

# PR Curve
plt.subplot(2, 2, 2)
plt.plot(recalls, precisions, marker='.')
plt.title("Precision-Recall Curve (LightGBM)")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.grid(True)
```

```
# ROC Curve
plt.subplot(2, 2, 3)
plt.plot(fpr, tpr, label=f"AUC = {roc_auc_score(y_test, y_scores):.3f}")
plt.plot([0,1],[0,1],'k--')
plt.title("ROC Curve (LightGBM)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()


# Store ROC-AUC Scores
scores_dict['LightGBM'] = {
    'Train': roc_auc_score(y_train, lgb_clf.predict_proba(X_train)[:,1]),
    'Test': roc_auc_score(y_test, y_scores),
}

print(f"Train ROC-AUC: {scores_dict['LightGBM']['Train']:.4f}")
print(f"Test  ROC-AUC: {scores_dict['LightGBM']['Test']:.4f}")
```

LightGBM Evaluation

Model performance was evaluated through:

- Precision-Recall Curve
- ROC Curve
- Confusion Matrix
- Classification Report

The model achieved:

- Training Accuracy: 100.0%
- Testing Accuracy: 84.4%

```
LightGBM Evaluation
TRAINIG RESULTS:
==============================
CONFUSION MATRIX:
[[863   0]
 [  0 166]]
ACCURACY SCORE:
1.0000
CLASSIFICATION REPORT:
                0      1  accuracy  macro avg  weighted avg
precision     1.0    1.0       1.0        1.0           1.0
recall        1.0    1.0       1.0        1.0           1.0
f1-score      1.0    1.0       1.0        1.0           1.0
support     863.0  166.0       1.0     1029.0        1029.0
TESTING RESULTS:
==============================
CONFUSION MATRIX:
[[350  20]
 [ 49  22]]
ACCURACY SCORE:
0.8435
CLASSIFICATION REPORT:
                   0          1  accuracy   macro avg  weighted avg
precision   0.877193   0.523810  0.843537    0.700501      0.820299
recall      0.945946   0.309859  0.843537    0.627903      0.843537
f1-score    0.910273   0.389381  0.843537    0.649827      0.826411
support   370.000000  71.000000  0.843537  441.000000    441.000000
```

CatBoost Classification Model

```python
# Initialize and Train CatBoost
cb_clf = CatBoostClassifier(verbose=0, random_state=42)
cb_clf.fit(X_train, y_train)

print("CatBoost Evaluation")
evaluate(cb_clf, X_train, X_test, y_train, y_test)

# Precision-Recall & ROC Curves
# Use predict_proba for proper probability scores
y_scores = cb_clf.predict_proba(X_test)[:, 1]

precisions, recalls, thresholds = precision_recall_curve(y_test, y_scores)
fpr, tpr, _ = roc_curve(y_test, y_scores)

plt.figure(figsize=(16, 12))

# Precision vs Threshold
plt.subplot(2, 2, 1)
plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
plt.title("Precision & Recall vs Threshold (CatBoost)")
plt.xlabel("Threshold")
plt.legend()
plt.grid(True)

# PR Curve
plt.subplot(2, 2, 2)
plt.plot(recalls, precisions, marker='.')
plt.title("Precision-Recall Curve (CatBoost)")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.grid(True)
```

```python
# ROC Curve
plt.subplot(2, 2, 3)
plt.plot(fpr, tpr, label=f"AUC = {roc_auc_score(y_test, y_scores):.3f}")
plt.plot([0,1],[0,1],'k--')
plt.title("ROC Curve (CatBoost)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

# Store ROC-AUC Scores
scores_dict['CatBoost'] = {
    'Train': roc_auc_score(y_train, cb_clf.predict_proba(X_train)[:,1]),
    'Test': roc_auc_score(y_test, y_scores),
}

print(f"Train ROC-AUC: {scores_dict['CatBoost']['Train']:.4f}")
print(f"Test  ROC-AUC: {scores_dict['CatBoost']['Test']:.4f}")
```

CatBoost Evaluation

Model performance was evaluated using:

- Precision-Recall Curve
- ROC Curve
- Confusion Matrix
- Classification Report

The model achieved:

- Training Accuracy: 98.35%
- Testing Accuracy: 85.49%

```
CatBoost Evaluation
TRAINIG RESULTS:
==============================
CONFUSION MATRIX:
[[863   0]
 [ 17 149]]
ACCURACY SCORE:
0.9835
CLASSIFICATION REPORT:
                    0           1  accuracy    macro avg  weighted avg
precision    0.980682    1.000000  0.983479     0.990341      0.983798
recall       1.000000    0.897590  0.983479     0.948795      0.983479
f1-score     0.990247    0.946032  0.983479     0.968139      0.983114
support    863.000000  166.000000  0.983479  1029.000000   1029.000000
TESTING RESULTS:
==============================
CONFUSION MATRIX:
[[363   7]
 [ 57  14]]
ACCURACY SCORE:
0.8549
CLASSIFICATION REPORT:
                    0          1  accuracy   macro avg  weighted avg
precision    0.864286   0.666667  0.854875    0.765476      0.832469
recall       0.981081   0.197183  0.854875    0.589132      0.854875
f1-score     0.918987   0.304348  0.854875    0.611668      0.820032
support    370.000000  71.000000  0.854875  441.000000    441.000000
```

AdaBoost Classification Model

```
# Train AdaBoost Model
ab_clf = AdaBoostClassifier(random_state=42)
ab_clf.fit(X_train, y_train)

print("AdaBoost Evaluation")
evaluate(ab_clf, X_train, X_test, y_train, y_test)

#Precision-Recall & ROC Curves
# Use predict_proba for probabilistic scores
y_scores = ab_clf.predict_proba(X_test)[:, 1]

precisions, recalls, thresholds = precision_recall_curve(y_test, y_scores)
fpr, tpr, _ = roc_curve(y_test, y_scores)

plt.figure(figsize=(16, 12))

# Precision vs Threshold
plt.subplot(2, 2, 1)
plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
plt.title("Precision & Recall vs Threshold (AdaBoost)")
plt.xlabel("Threshold")
plt.legend()
plt.grid(True)

# PR Curve
plt.subplot(2, 2, 2)
plt.plot(recalls, precisions, marker='.')
plt.title("Precision-Recall Curve (AdaBoost)")
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.grid(True)
```

```
# ROC Curve
plt.subplot(2, 2, 3)
plt.plot(fpr, tpr, label=f"AUC = {roc_auc_score(y_test, y_scores):.3f}")
plt.plot([0,1],[0,1],'k--')
plt.title("ROC Curve (AdaBoost)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.show()

#Store ROC-AUC Scores
scores_dict['AdaBoost'] = {
    'Train': roc_auc_score(y_train, ab_clf.predict_proba(X_train)[:,1]),
    'Test': roc_auc_score(y_test, y_scores),
}

print(f"Train ROC-AUC: {scores_dict['AdaBoost']['Train']:.4f}")
print(f"Test  ROC-AUC: {scores_dict['AdaBoost']['Test']:.4f}")
```

AdaBoost Evaluation

Model performance was evaluated using:

- Precision-Recall Curve
- ROC Curve
- Confusion Matrix
- Classification Report

The model achieved:

- Training Accuracy: 89.2%
- Testing Accuracy: 82.9%

```
AdaBoost Evaluation
TRAINIG RESULTS:
==============================
CONFUSION MATRIX:
[[851  12]
 [ 99  67]]
ACCURACY SCORE:
0.8921
CLASSIFICATION REPORT:
                   0           1  accuracy    macro avg  weighted avg
precision   0.895789    0.848101  0.892128     0.871945      0.888096
recall      0.986095    0.403614  0.892128     0.694855      0.892128
f1-score    0.938776    0.546939  0.892128     0.742857      0.875564
support   863.000000  166.000000  0.892128  1029.000000   1029.000000
TESTING RESULTS:
==============================
CONFUSION MATRIX:
[[350  20]
 [ 55  16]]
ACCURACY SCORE:
0.8299
CLASSIFICATION REPORT:
                   0          1  accuracy   macro avg  weighted avg
precision   0.864198   0.444444  0.829932    0.654321      0.796618
recall      0.945946   0.225352  0.829932    0.585649      0.829932
f1-score    0.903226   0.299065  0.829932    0.601146      0.805957
support   370.000000  71.000000  0.829932  441.000000    441.000000
```

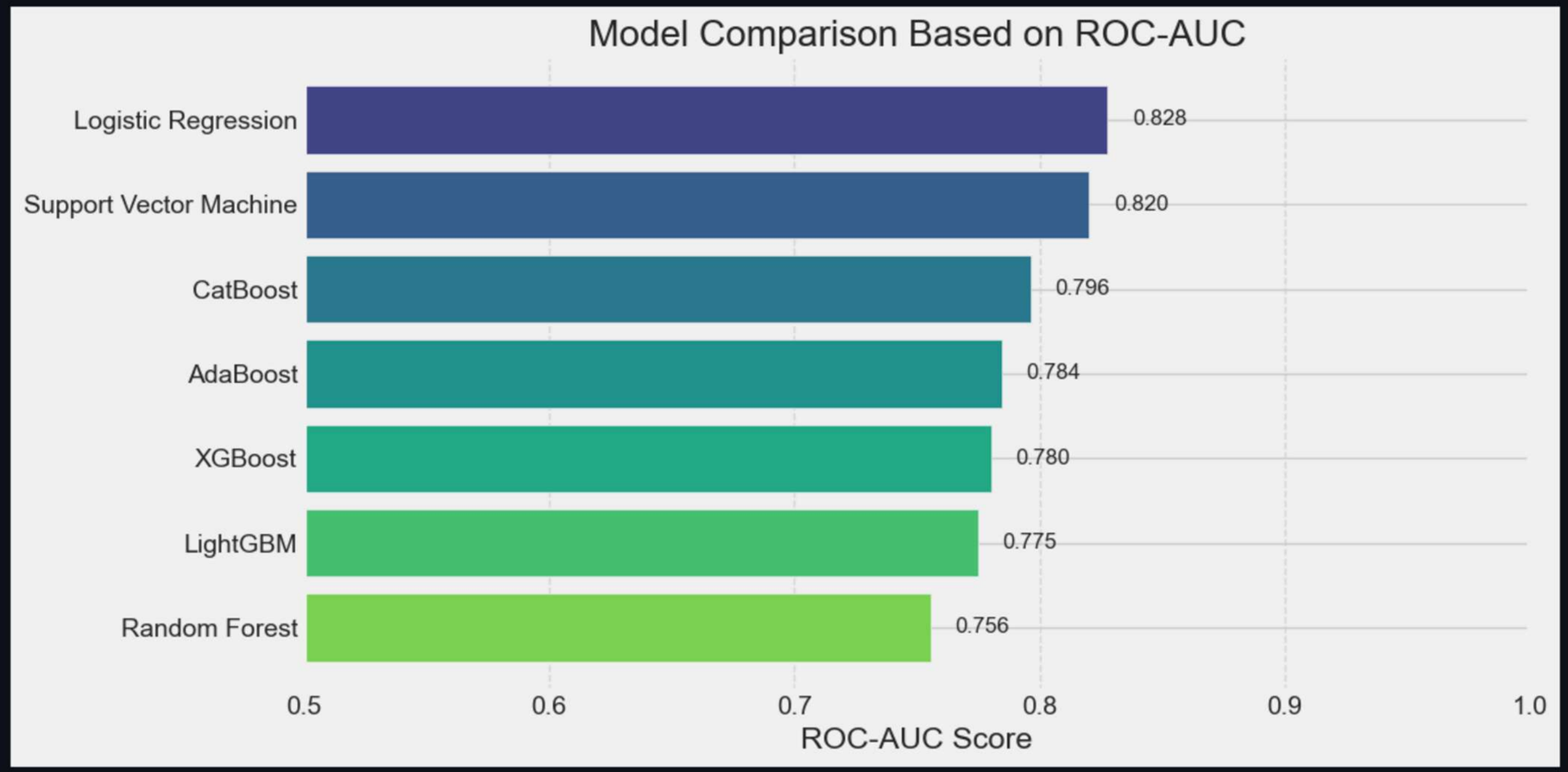
Overall Model Comparison

```python
# Define models and their respective test datasets
ml_models = {
    'Random Forest': (rf_clf, X_test),
    'XGBoost': (xgb_clf, X_test),
    'Logistic Regression': (lr_clf, X_test_std),
    'Support Vector Machine': (svm_clf, X_test_std),
    'LightGBM': (lgb_clf, X_test),
    'CatBoost': (cb_clf, X_test),
    'AdaBoost': (ab_clf, X_test)
}

# Collect AUC Scores
auc_scores = {}

for model_name, (model, X_tst) in ml_models.items():
    if hasattr(model, "predict_proba"):
        y_pred_prob = model.predict_proba(X_tst)[:, 1]
    elif hasattr(model, "decision_function"):
        y_pred_prob = model.decision_function(X_tst)
    else:
        y_pred_prob = model.predict(X_tst)

    auc = roc_auc_score(y_test, y_pred_prob)
    auc_scores[model_name] = auc
    print(f"{model_name:<25}  ROC-AUC Score: {auc:.3f}")


# Bar Plot of AUC Scores
sorted_auc = dict(sorted(auc_scores.items(), key=lambda item: item[1], reverse=True))

plt.figure(figsize=(12,6))
bars = plt.barh(list(sorted_auc.keys()), list(sorted_auc.values()), color=plt.cm.viridis(np.linspace(0.2, 0.8, len(sorted_auc))))

plt.xlabel("ROC-AUC Score")
plt.title("Model Comparison Based on ROC-AUC")
plt.xlim(0.5, 1.0)
plt.grid(axis='x', linestyle='--', alpha=0.7)

# Annotate Bars
for bar in bars:
    plt.text(bar.get_width() + 0.01, bar.get_y() + bar.get_height()/2, f"{bar.get_width():.3f}", va='center', fontsize=12)

plt.gca().invert_yaxis()  # Highest on top
plt.tight_layout()
plt.show()
```

```python
# Combined ROC Curve Plot
plt.figure(figsize=(12,8))
colors = get_cmap('tab10')

for idx, (model_name, (model, X_tst)) in enumerate(ml_models.items()):
    if hasattr(model, "predict_proba"):
        y_pred_prob = model.predict_proba(X_tst)[:, 1]
    elif hasattr(model, "decision_function"):
        y_pred_prob = model.decision_function(X_tst)
    else:
        y_pred_prob = model.predict(X_tst)

    fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
    auc_score = auc_scores[model_name]
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {auc_score:.3f})', color=colors(idx))

plt.plot([0, 1], [0, 1], 'k--', label='Random Guessing')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves Comparison Across Models')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

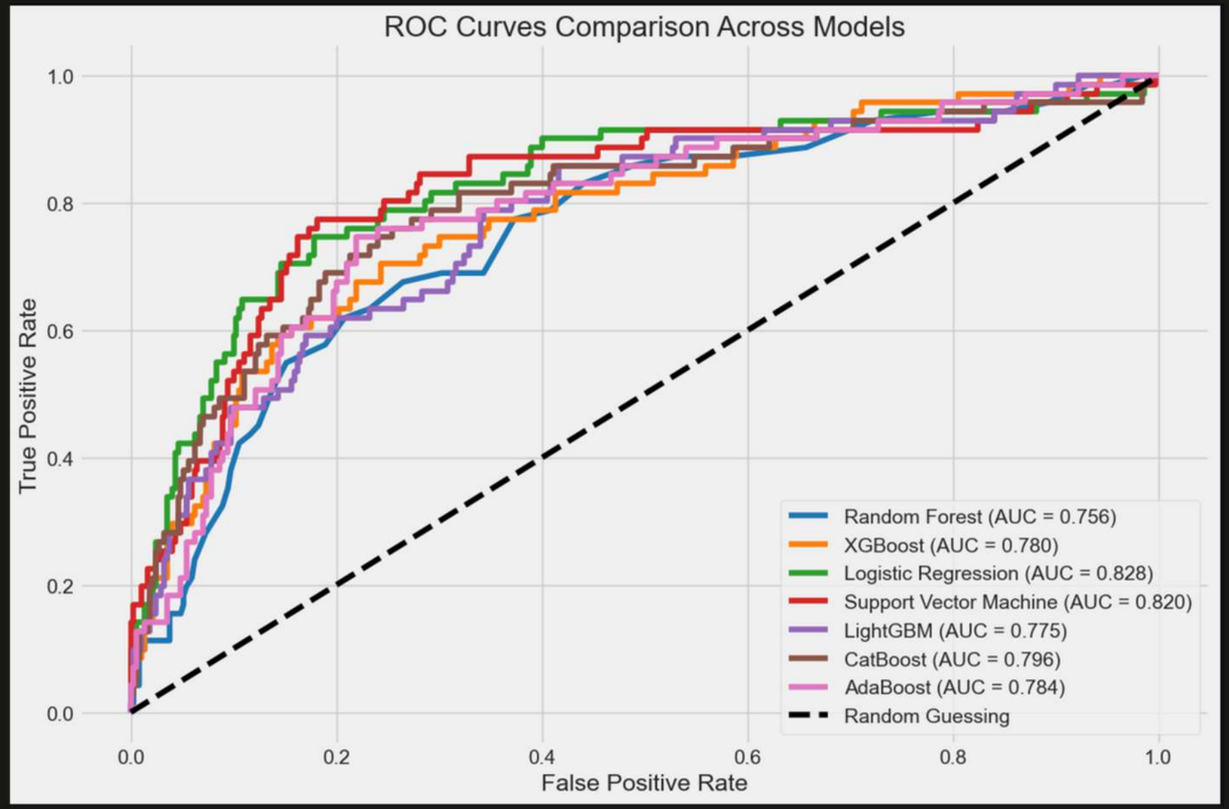
ROC–AUC SCORE

```
Random Forest             ROC-AUC Score: 0.756
XGBoost                   ROC-AUC Score: 0.780
Logistic Regression       ROC-AUC Score: 0.828
Support Vector Machine    ROC-AUC Score: 0.820
LightGBM                  ROC-AUC Score: 0.775
CatBoost                  ROC-AUC Score: 0.796
AdaBoost                  ROC-AUC Score: 0.784
```



ROC Curves Comparision

- Logistic Regression and SVM outperformed other models in terms of ROC-AUC.
- Tree-based ensemble methods like CatBoost and XGBoost showed strong but slightly overfitted performance.
- Random Forest lagged due to overfitting despite perfect training accuracy.



- Conclusion:

Logistic Regression achieved the best balance between accuracy and robustness, as indicated by the highest ROC-AUC score of 0.828.

Conclusion

In conclusion, this project provided an end-to-end analysis of the IBM HR Analytics Employee Attrition dataset, covering every stage from exploratory data analysis (EDA) to advanced machine learning model evaluation. Through detailed EDA, we identified key factors influencing employee attrition, such as job satisfaction, work-life balance, total working years, and salary hikes.

We implemented a range of machine learning algorithms—including Logistic Regression, Random Forest, SVM, XGBoost, LightGBM, CatBoost, and AdaBoost—to predict attrition. Each model was rigorously evaluated using metrics like Accuracy, Precision-Recall curves, ROC curves, Confusion Matrices, and ROC-AUC scores.

Among all models, Logistic Regression and Support Vector Machine demonstrated the best balance between performance and generalization, achieving the highest ROC-AUC scores. Tree-based models, while powerful, showed tendencies toward overfitting, emphasizing the importance of hyperparameter tuning and model selection based on the business context. This project highlights how data-driven approaches and machine learning can empower HR departments to proactively address attrition risks. By leveraging predictive insights, organizations can develop targeted retention strategies, ultimately reducing turnover and enhancing workforce stability.

Overall, this analysis demonstrates the practical application of machine learning in solving real-world HR challenges, reinforcing the value of data science in strategic decision-making.


Thank You