Employee Turnover

January 3, 2025

[10]:
```
import pandas as pd
import numpy as np
import scipy.stats
import seaborn as sns
from sklearn.metrics.pairwise import cosine_similarity
import matplotlib.pyplot as plt
import statistics
import operator
from sklearn.metrics import silhouette_score
from imblearn.pipeline import Pipeline
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression  # Logistic Regression
from sklearn.ensemble import RandomForestClassifier  # Random Forest
from sklearn.ensemble import GradientBoostingClassifier  # Gradient Boosting
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import classification_report
```

0.0.1 1.

0.0.2 Data Quality Check

[11]:
```
df = pd.read_csv('HR_comma_sep.csv')
df.head()
```

[11]:
```
   satisfaction_level  last_evaluation  number_project  average_montly_hours  \
0                0.38             0.53               2                   157
1                0.80             0.86               5                   262
2                0.11             0.88               7                   272
3                0.72             0.87               5                   223
4                0.37             0.52               2                   159

   time_spend_company  Work_accident  left  promotion_last_5years  sales  \
```

```
0                   3              0     1                       0   sales
1                   6              0     1                       0   sales
2                   4              0     1                       0   sales
3                   5              0     1                       0   sales
4                   3              0     1                       0   sales

   salary
0     low
1  medium
2  medium
3     low
4     low
```

[12]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   satisfaction_level     14999 non-null  float64
 1   last_evaluation        14999 non-null  float64
 2   number_project         14999 non-null  int64
 3   average_montly_hours   14999 non-null  int64
 4   time_spend_company     14999 non-null  int64
 5   Work_accident          14999 non-null  int64
 6   left                   14999 non-null  int64
 7   promotion_last_5years  14999 non-null  int64
 8   sales                  14999 non-null  object
 9   salary                 14999 non-null  object
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
```

Check for missing values

[13]:
```
df.isnull().sum()
```

[13]:
```
satisfaction_level       0
last_evaluation          0
number_project           0
average_montly_hours     0
time_spend_company       0
Work_accident            0
left                     0
promotion_last_5years    0
sales                    0
salary                   0
dtype: int64
```

Drop duplicated records

[14]:
```
df.duplicated().sum()
df.shape
```

[14]: (14999, 10)

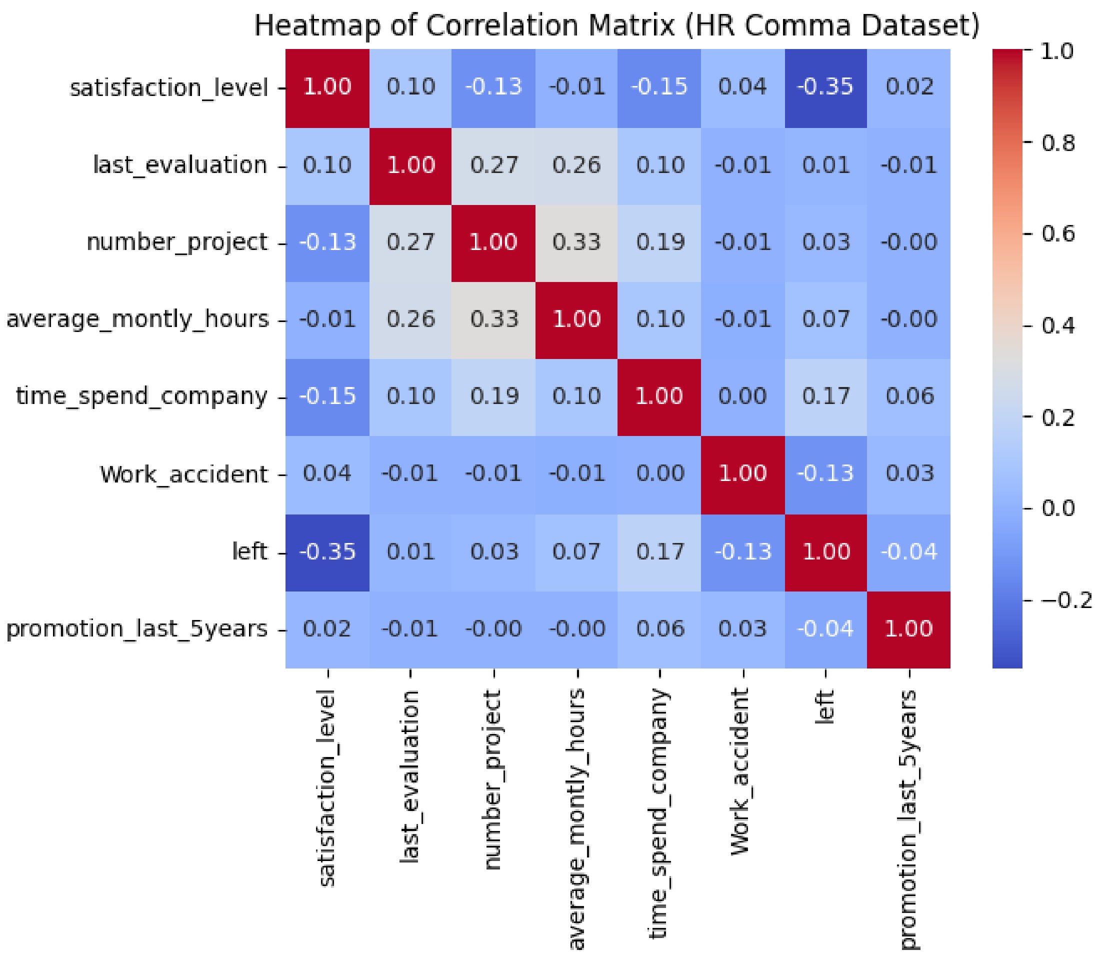
[15]:
```
#drop duplicate records
df = df.drop_duplicates()
df.shape
```

[15]: (11991, 10)

0.0.3 2.

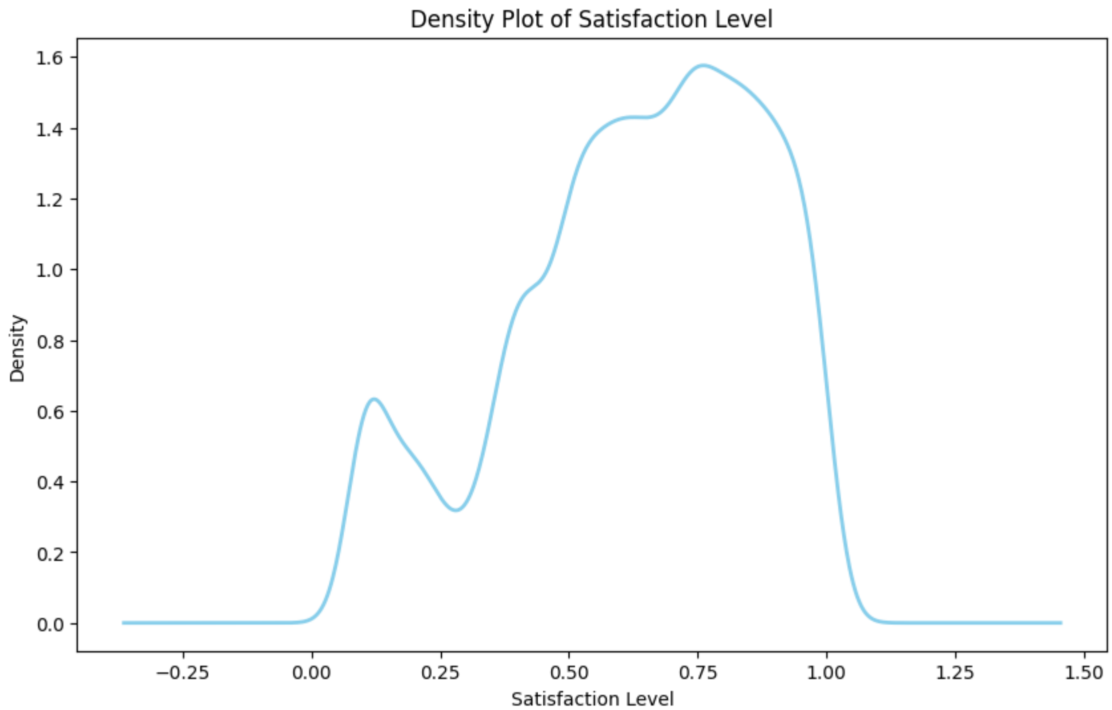
Heatmap of correlation between all numerical features in the data

[16]:
```
# heatmap of correlation of all numeric features.
correlation_matrix = df.select_dtypes(include='number').corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Heatmap of Correlation Matrix (HR Comma Dataset)')
plt.show()
```



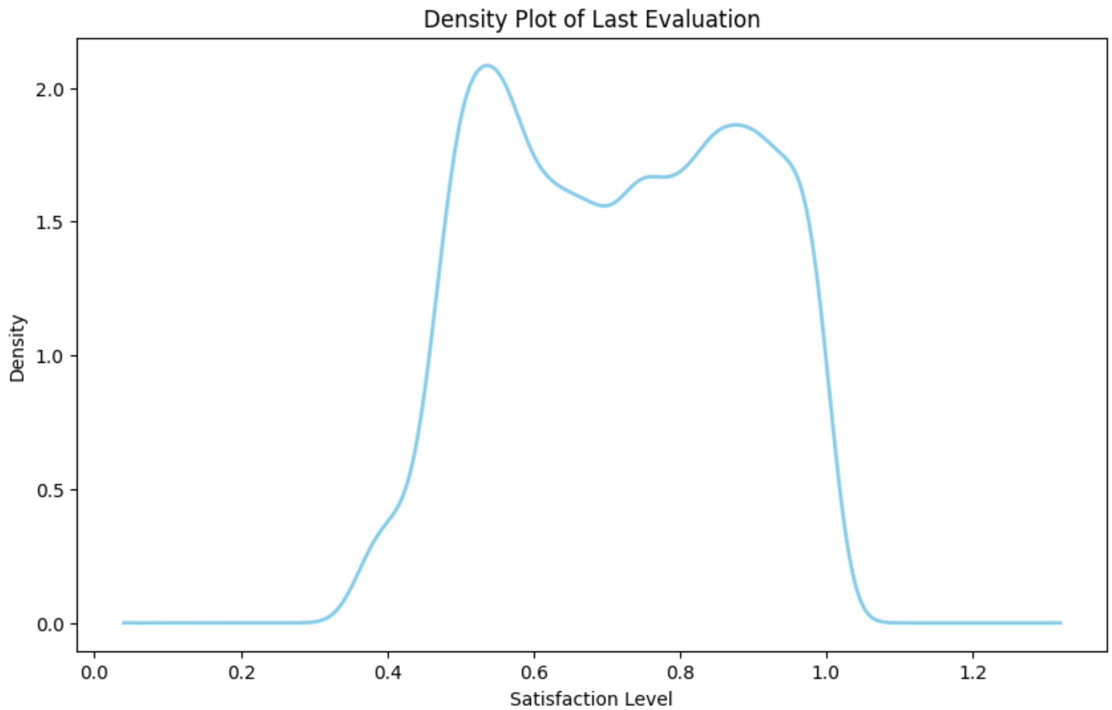
Employee Satisfaction Distribution Plot

[17]:
```
# density plot best shows the relationships between what we are looking for
plt.figure(figsize=(10, 6))
df['satisfaction_level'].plot(kind='density', color='skyblue', linewidth=2)
plt.title('Density Plot of Satisfaction Level')
plt.xlabel('Satisfaction Level')
plt.ylabel('Density')
plt.show()
```



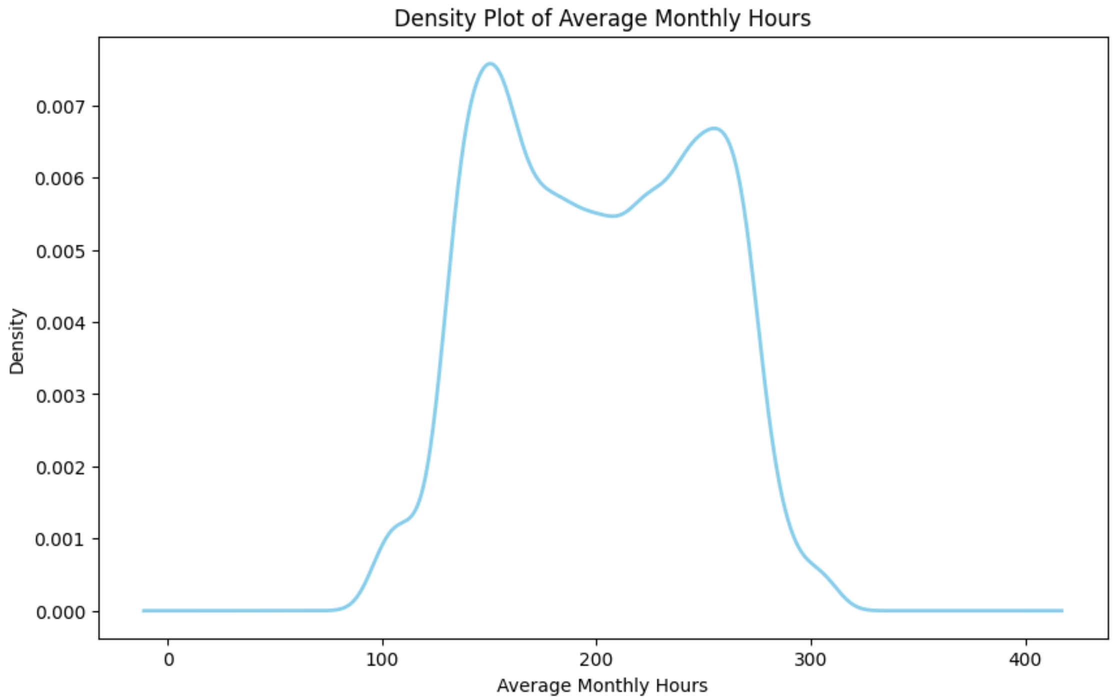
Employee Evaluation Distribution Plot

[18]:
```
plt.figure(figsize=(10, 6))
df['last_evaluation'].plot(kind='density', color='skyblue', linewidth=2)
plt.title('Density Plot of Last Evaluation')
plt.xlabel('Satisfaction Level')
plt.ylabel('Density')
plt.show()
```



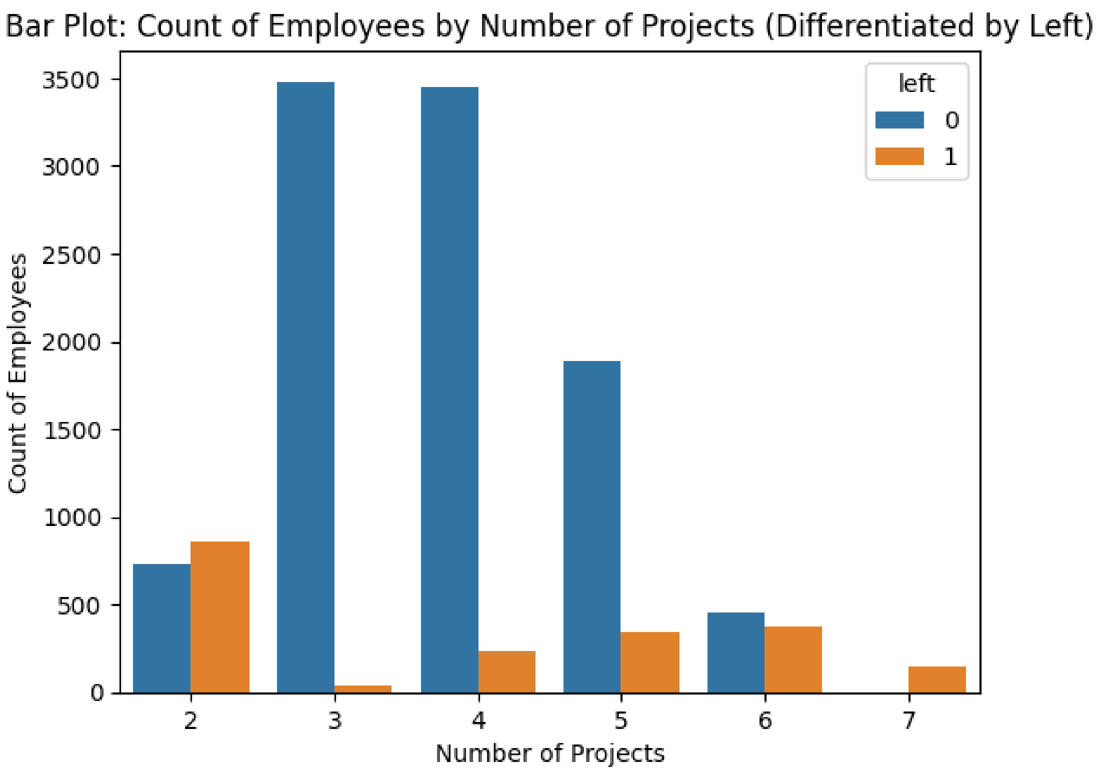
Employee Average Monthly Hours Distribution Plot

[19]:
```
plt.figure(figsize=(10, 6))
df['average_montly_hours'].plot(kind='density', color='skyblue', linewidth=2)
plt.title('Density Plot of Average Monthly Hours')
plt.xlabel('Average Monthly Hours')
plt.ylabel('Density')
plt.show()
```



Bar plot of the employee project count of both employees who left and stayed in the organization

[20]:
```
# Bar Plot with count of employees for each number_project differentiated by␣
↪left
sns.countplot(x='number_project', data=df, hue='left')
plt.title('Bar Plot: Count of Employees by Number of Projects (Differentiated␣
↪by Left)')
plt.ylabel('Count of Employees')
plt.xlabel('Number of Projects')
plt.show()
```



Underwork and Overwork Cause Turnover: Employees with 2 projects (underworked) or 5+ projects (overworked) are more likely to leave. Ensuring an optimal workload of 3-4 projects can reduce turnover.

0.0.4 3.

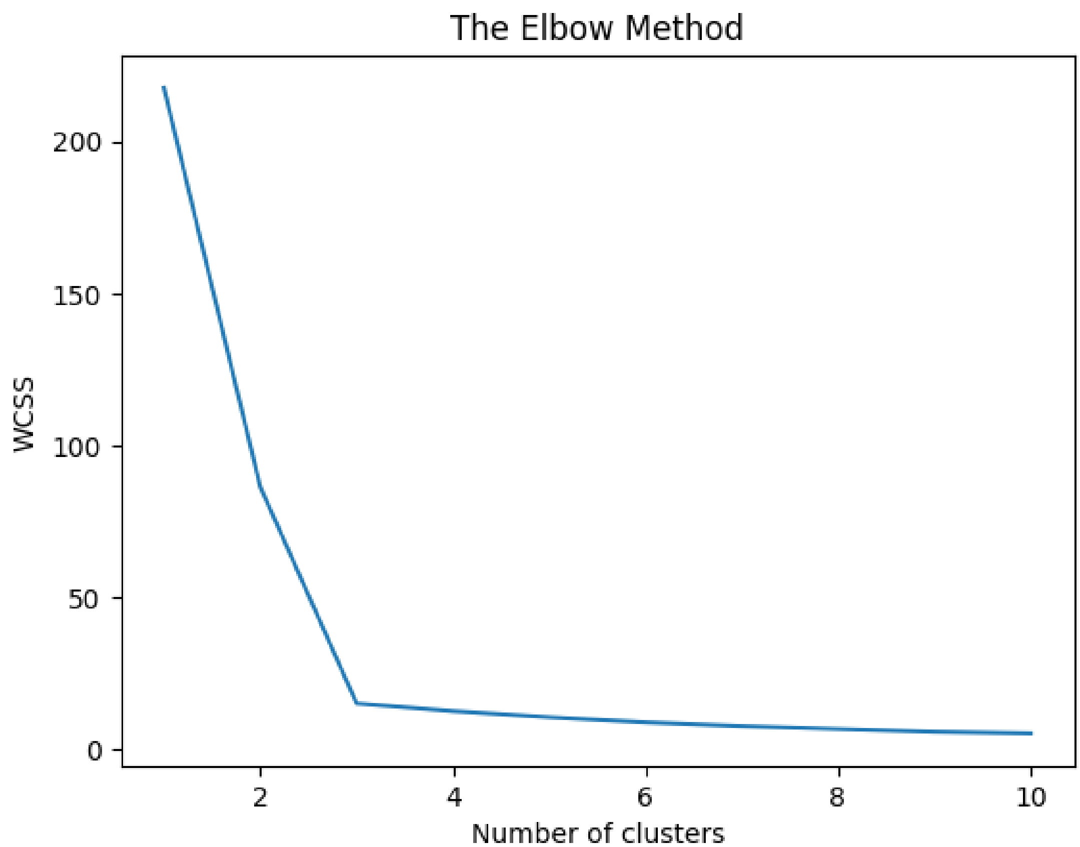
0.0.5 Perform clustering of employees who left based on their satisfaction and evaluation

Finding optimal K for K-means clustering of emplpyees who left the company based on their satisfaction and evaluation

[21]:
```
satisfaction_evaluation_left = df[df['left'] == 1][['satisfaction_level',␣
↪'last_evaluation']]
```

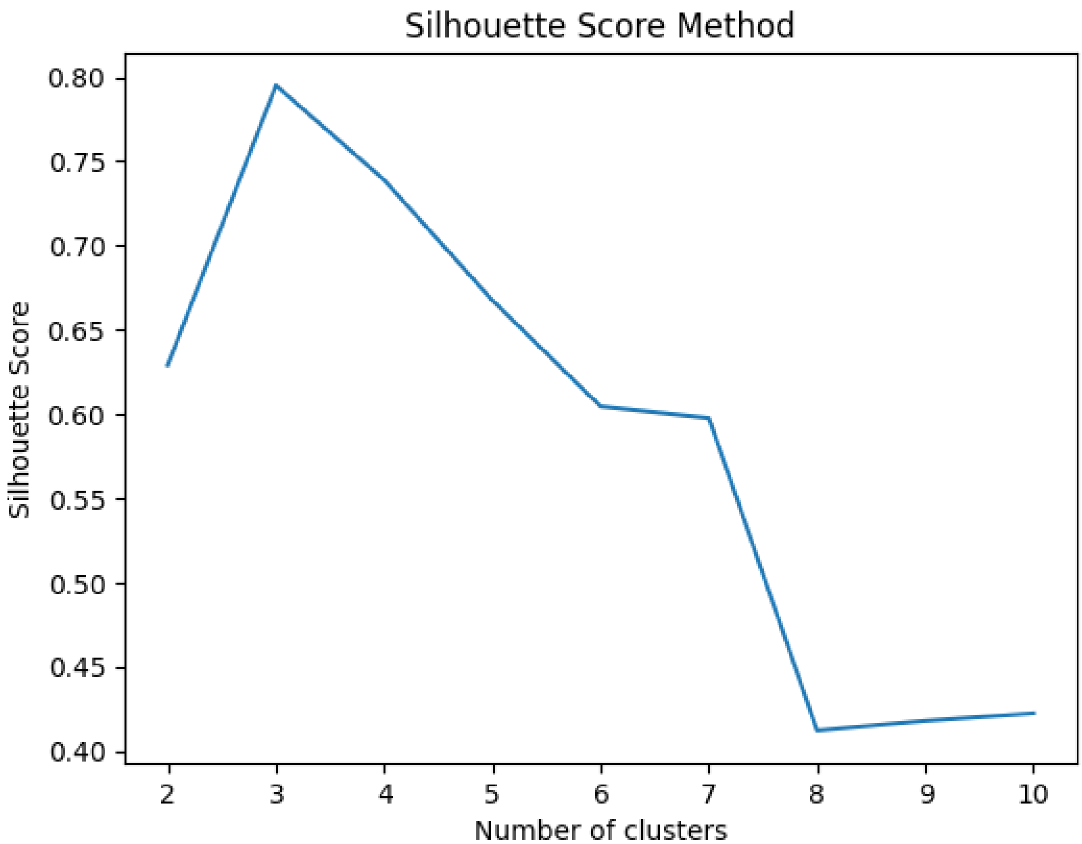
[22]:
```
wcss = []
for i in range(1, 11):
    model = KMeans(n_clusters = i, n_init=10, init = 'k-means++', random_state␣
↪= 42)
    model.fit(satisfaction_evaluation_left)
    wcss.append(model.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('The Elbow Method')
```

```
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



[23]:
```
#since using wcss score was tricky, we can use silhoutte score to find the␣
↪optimal number of clusters :
# Silhouette Score method
silhouette_scores = []
for i in range(2, 11):
    model = KMeans(n_clusters=i, n_init=10,  init='k-means++', random_state=42)
    model.fit(satisfaction_evaluation_left)
    score = silhouette_score(satisfaction_evaluation_left, model.labels_)
    silhouette_scores.append(score)

plt.plot(range(2, 11), silhouette_scores)
plt.title('Silhouette Score Method')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.show()
```

```
# we see that three is the optimal number of clusters, so we go ahead and make␣
↪three clusters
```



[24]:
```
optimal_clusters = range(2, 11)[silhouette_scores.index(max(silhouette_scores))]
optimal_clusters
```

[24]: 3

0.0.6 Optimal number of K = 3

[25]:
```
model = KMeans(n_clusters=optimal_clusters, n_init=10, init='k-means++',␣
↪random_state=42)
y_kmeans = model.fit_predict(satisfaction_evaluation_left)
```

[26]:
```
# Convert the DataFrame to a NumPy array for correct indexing
X = satisfaction_evaluation_left.values

# Visualize the clusters
plt.scatter(
```

```
    X[y_kmeans == 0, 0], X[y_kmeans == 0, 1],
    s=100, c='red', label='Cluster 1'
)
plt.scatter(
    X[y_kmeans == 1, 0], X[y_kmeans == 1, 1],
    s=100, c='blue', label='Cluster 2'
)
plt.scatter(
    X[y_kmeans == 2, 0], X[y_kmeans == 2, 1],
    s=100, c='green', label='Cluster 3'
)

# Add centroids
plt.scatter(
    model.cluster_centers_[:, 0], model.cluster_centers_[:, 1],
    s=300, c='yellow', marker='x', label='Centroids'
)

# Customize the plot
plt.title('Clusters of Employees Who Left')
plt.xlabel('Satisfaction Level')
plt.ylabel('Last Evaluation')
plt.legend()
plt.show()
```

0.0.7 Clusters:

Cluster 1 (Blue - High Satisfaction, High Evaluation): Employees in this group are highly satisfied and performed well. Possible Reason for Leaving: Lack of growth opportunities or external offers despite strong performance.

Cluster 2 (Red - Low Satisfaction, Medium Evaluation): These employees have low satisfaction and moderate evaluations. Possible Reason for Leaving: Disengagement, dissatisfaction with work environment, or misaligned roles.

Cluster 3 (Green - Low Satisfaction, High Evaluation): Employees in this group have high evaluations but very low satisfaction. Possible Reason for Leaving: Burnout or lack of recognition for their efforts despite strong performance.

0.0.8 4

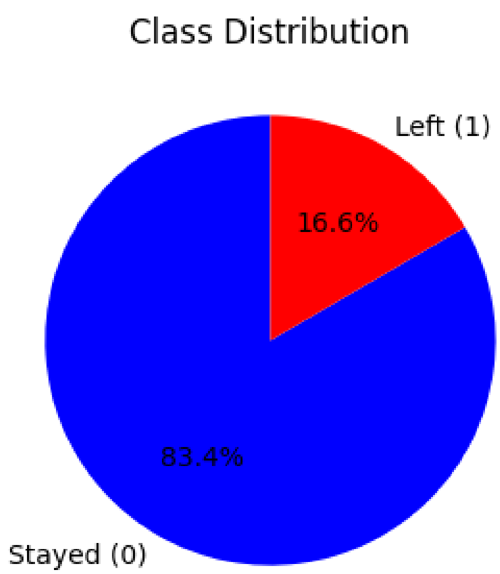
Convert categorical variables to numerical variables

[27]:
```
# 4 imbalanced data we have over here
class_distribution = df['left'].value_counts()
print(class_distribution)
```

```
left
0    10000
1     1991
Name: count, dtype: int64
```

[28]:

```
# first we have to get the categorical data :
# Plot the distribution of the 'Class' column as a pie chart
plt.subplot(1, 2, 2)
class_distribution.plot(kind='pie', labels=['Stayed (0)', 'Left (1)'],␣
 ↪colors=['blue', 'red'], autopct='%1.1f%%', startangle=90)
plt.title('Class Distribution')
plt.ylabel('')

# Show the plots
plt.tight_layout()
plt.show()
```



[29]:

```
# Separate features and target
X = df.drop('left', axis=1)   # Features
y = df['left']                # Target variable

# Identify categorical and numerical features
categorical_features = X.select_dtypes(include=['object'])  # Categorical␣
 ↪columns
numerical_features = X.select_dtypes(include=['number'])    # Numerical columns
```

```
# Apply one-hot encoding to categorical features
categorical_encoded = pd.get_dummies(categorical_features, drop_first=True).
 ↪astype(int)

# Combine numerical and encoded categorical features
X_processed = pd.concat([numerical_features, categorical_encoded], axis=1)
X_processed
```

[29]:
```
       satisfaction_level  last_evaluation  number_project  \
0                    0.38             0.53               2
1                    0.80             0.86               5
2                    0.11             0.88               7
3                    0.72             0.87               5
4                    0.37             0.52               2
…                      …                …              …
11995                0.90             0.55               3
11996                0.74             0.95               5
11997                0.85             0.54               3
11998                0.33             0.65               3
11999                0.50             0.73               4

       average_montly_hours  time_spend_company  Work_accident  \
0                       157                   3              0
1                       262                   6              0
2                       272                   4              0
3                       223                   5              0
4                       159                   3              0
…                     …                   …              …
11995                   259                  10              1
11996                   266                  10              0
11997                   185                  10              0
11998                   172                  10              0
11999                   180                   3              0

       promotion_last_5years  sales_RandD  sales_accounting  sales_hr  \
0                          0            0                 0         0
1                          0            0                 0         0
2                          0            0                 0         0
3                          0            0                 0         0
4                          0            0                 0         0
…                        …          …                 …       …
11995                      1            0                 0         0
11996                      1            0                 0         0
11997                      1            0                 0         0
11998                      1            0                 0         0
11999                      0            0                 0         0
```

```
       sales_management  sales_marketing  sales_product_mng  sales_sales  \
0                     0                0                  0            1
1                     0                0                  0            1
2                     0                0                  0            1
3                     0                0                  0            1
4                     0                0                  0            1
…                   …              …                …          …
11995                 1                0                  0            0
11996                 1                0                  0            0
11997                 1                0                  0            0
11998                 0                1                  0            0
11999                 0                0                  0            0

       sales_support  sales_technical  salary_low  salary_medium
0                  0                0           1              0
1                  0                0           0              1
2                  0                0           0              1
3                  0                0           1              0
4                  0                0           1              0
…                …              …         …            …
11995              0                0           0              0
11996              0                0           0              0
11997              0                0           0              0
11998              0                0           0              0
11999              0                0           1              0

[11991 rows x 18 columns]
```

[90]:

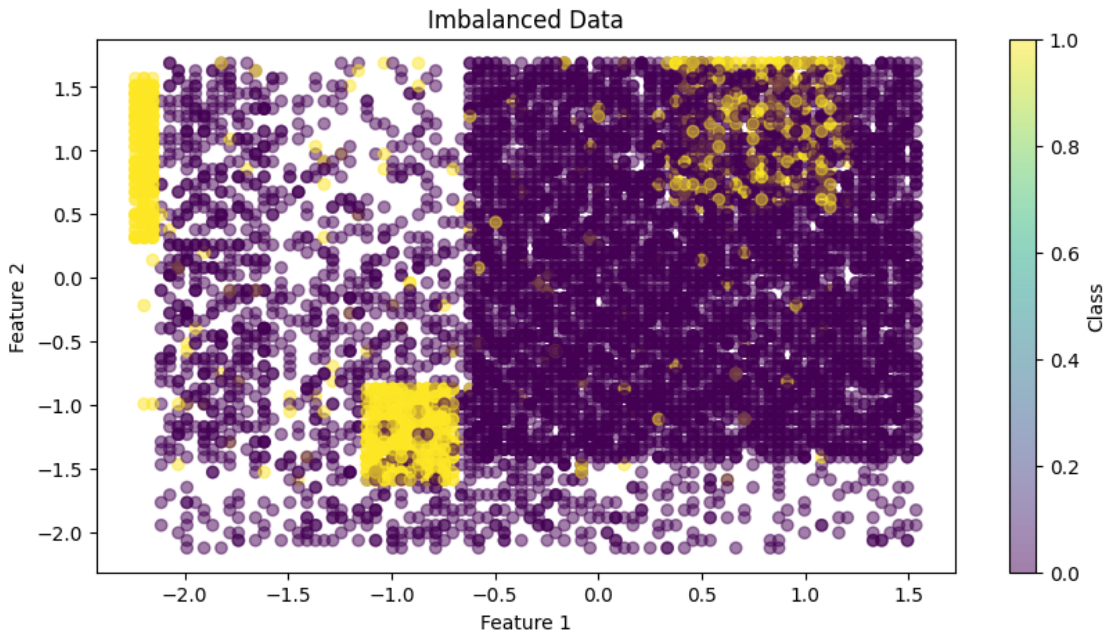
```
# Perform stratified split
X_train, X_test, y_train, y_test = train_test_split(
    X_processed, y,
    test_size=0.2,
    stratify=y,
    random_state=123
)

# Scale the numerical data
scaler = StandardScaler()
X_train_sc = scaler.fit_transform(X_train)
X_test_sc = scaler.transform(X_test)
```

[91]:

```
# Scatter plot for the imbalanced data
plt.figure(figsize=(10, 5))
plt.scatter(X_train_sc[:, 0], X_train_sc[:, 1], c=y_train, alpha=0.5,␣
 ↪cmap='viridis', marker='o')
plt.title('Imbalanced Data')
plt.xlabel('Feature 1')
```

```python
plt.ylabel('Feature 2')
plt.colorbar(label='Class')
```

[91]: <matplotlib.colorbar.Colorbar at 0x31b697c70>

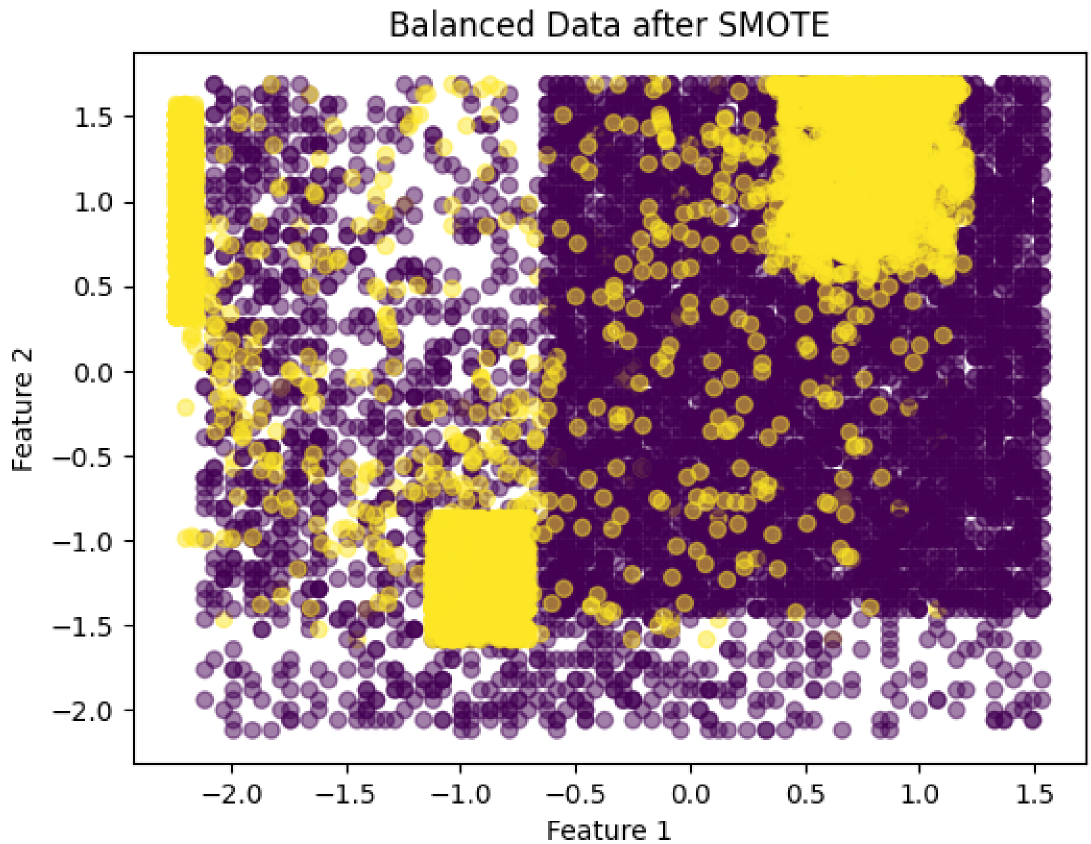


[92]:
```python
# now I have them combined , so I can proceed with the rest of the tasks
```

[93]:
```python
# Apply SMOTE
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train_sc, y_train)
```

[94]:
```python
plt.scatter(X_train_smote[:, 0], X_train_smote[:, 1], c=y_train_smote, alpha=0.
↪5, cmap="viridis", marker='o')
plt.title('Balanced Data after SMOTE')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
```

[94]: Text(0, 0.5, 'Feature 2')



0.0.9 5

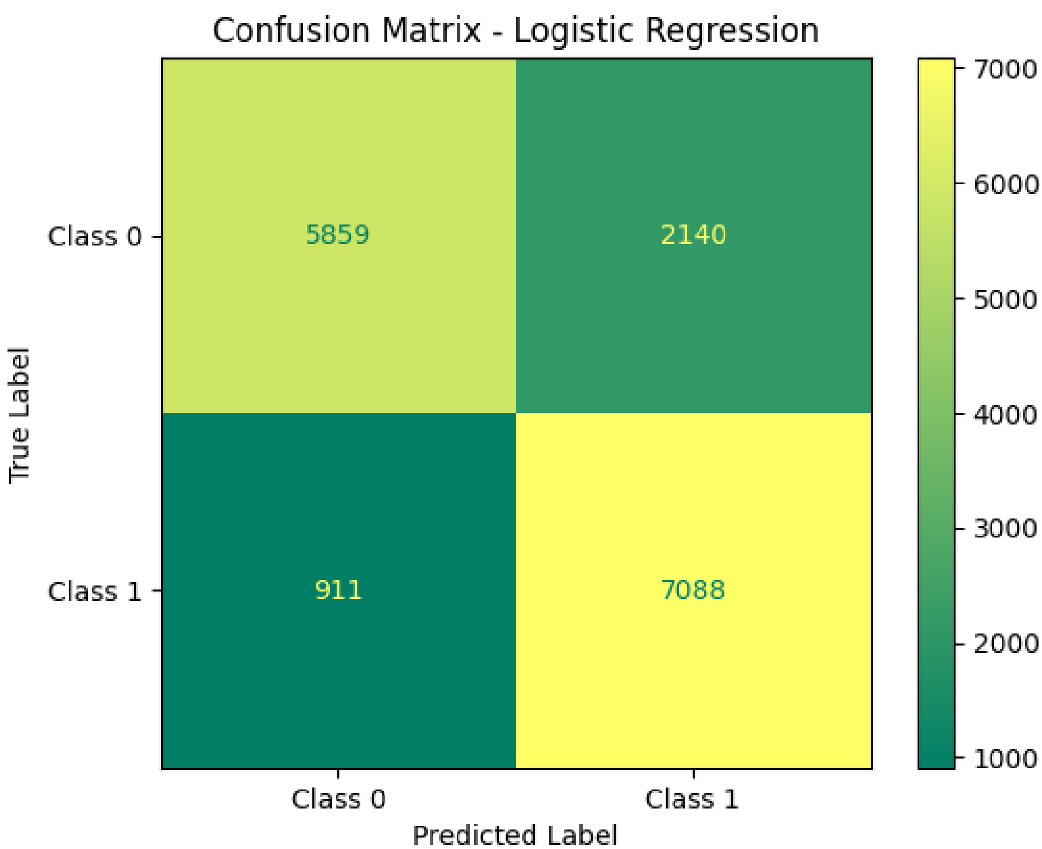
Perform 5-fold cross-validation model training and evaluate performance

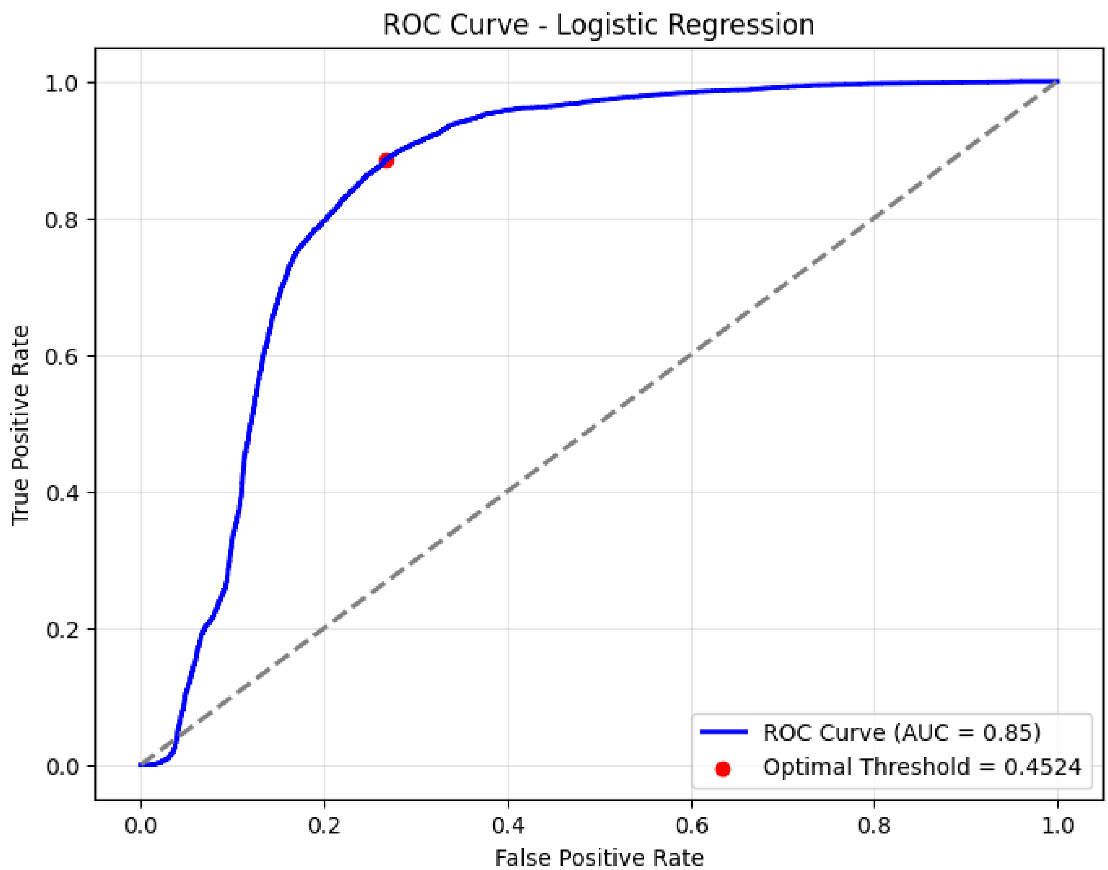
0.0.10 Logistic Regression Model

[109]:
```
# Initialize the Logistic Regression model
logistic_model = LogisticRegression(random_state=123)

# Perform 5-fold cross-validation and predict probabilities for the positive␣
↪class
y_prob_cv = cross_val_predict(logistic_model, X_train_smote, y_train_smote,␣
↪cv=5, method='predict_proba')[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_train_smote, y_prob_cv)

# Compute AUC
roc_auc = auc(fpr, tpr)
```

```
# Compute Youden's J statistic and the optimal threshold
youden_j = tpr - fpr
optimal_threshold_index = np.argmax(youden_j)
optimal_threshold = thresholds[optimal_threshold_index]

# Print classification report
y_pred_cv = (y_prob_cv >= optimal_threshold).astype(int)  # Use optimal threshold for predictions
print(f"Logistic Regression Optimal Threshold: {optimal_threshold:.4f}")
print("Logistic Regression Classification Report:")
print(classification_report(y_train_smote, y_pred_cv))

# Generate the classification report as a dictionary
report = classification_report(y_train_smote, y_pred_cv, output_dict=True)

# Convert the dictionary into a pandas DataFrame
report_df = pd.DataFrame(report).transpose()

# Filter for classes 0 and 1 (employees on the left class)
filtered_report = report_df.loc[["0", "1"], ["precision", "recall", "f1-score"]]

# Compute the confusion matrix
cm = confusion_matrix(y_train_smote, y_pred_cv)

# Display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Class 0', 'Class 1'])
disp.plot(cmap="summer", values_format="d")  # Using 'summer' colormap as requested
plt.title("Confusion Matrix - Logistic Regression")
plt.ylabel("True Label")
plt.xlabel("Predicted Label")
plt.show()

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.scatter(fpr[optimal_threshold_index], tpr[optimal_threshold_index], color='red', label=f'Optimal Threshold = {optimal_threshold:.4f}')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--', lw=2)  # Diagonal line for random guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression')
plt.legend(loc='lower right')
plt.grid(alpha=0.3)
plt.show()
```

```
Logistic Regression Optimal Threshold: 0.4524
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.73      0.79      7999
           1       0.77      0.89      0.82      7999

    accuracy                           0.81     15998
   macro avg       0.82      0.81      0.81     15998
weighted avg       0.82      0.81      0.81     15998
```



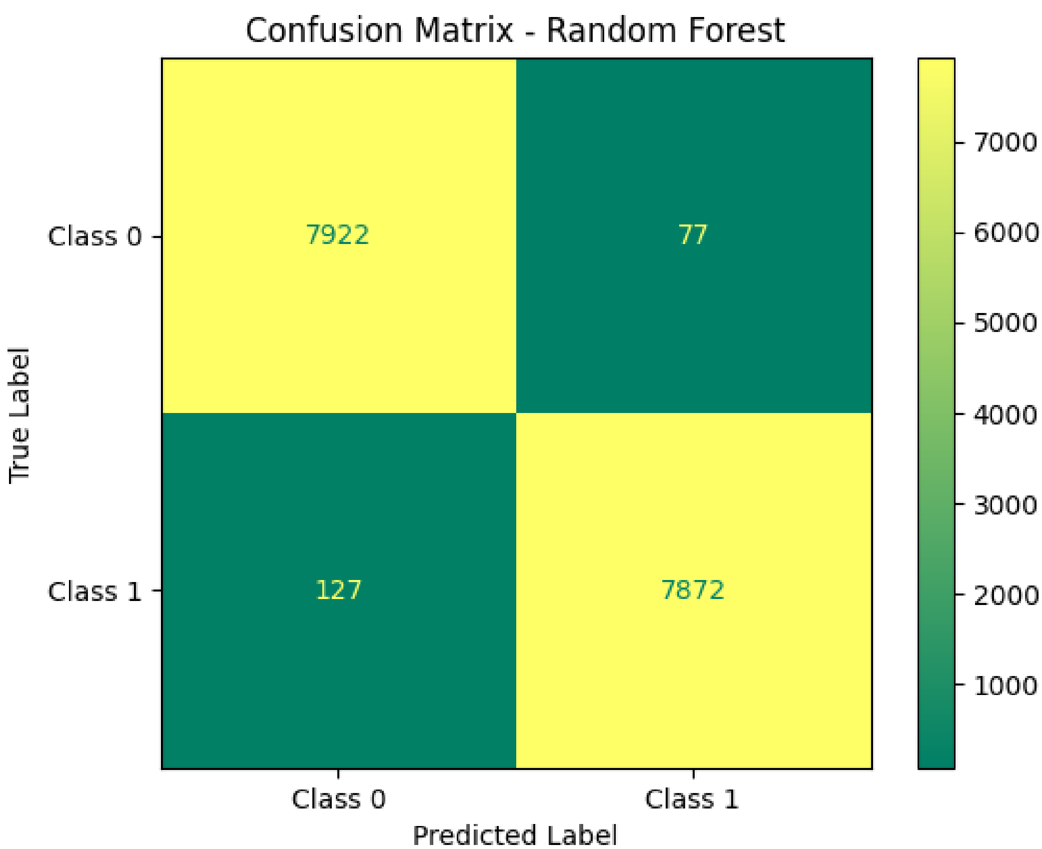


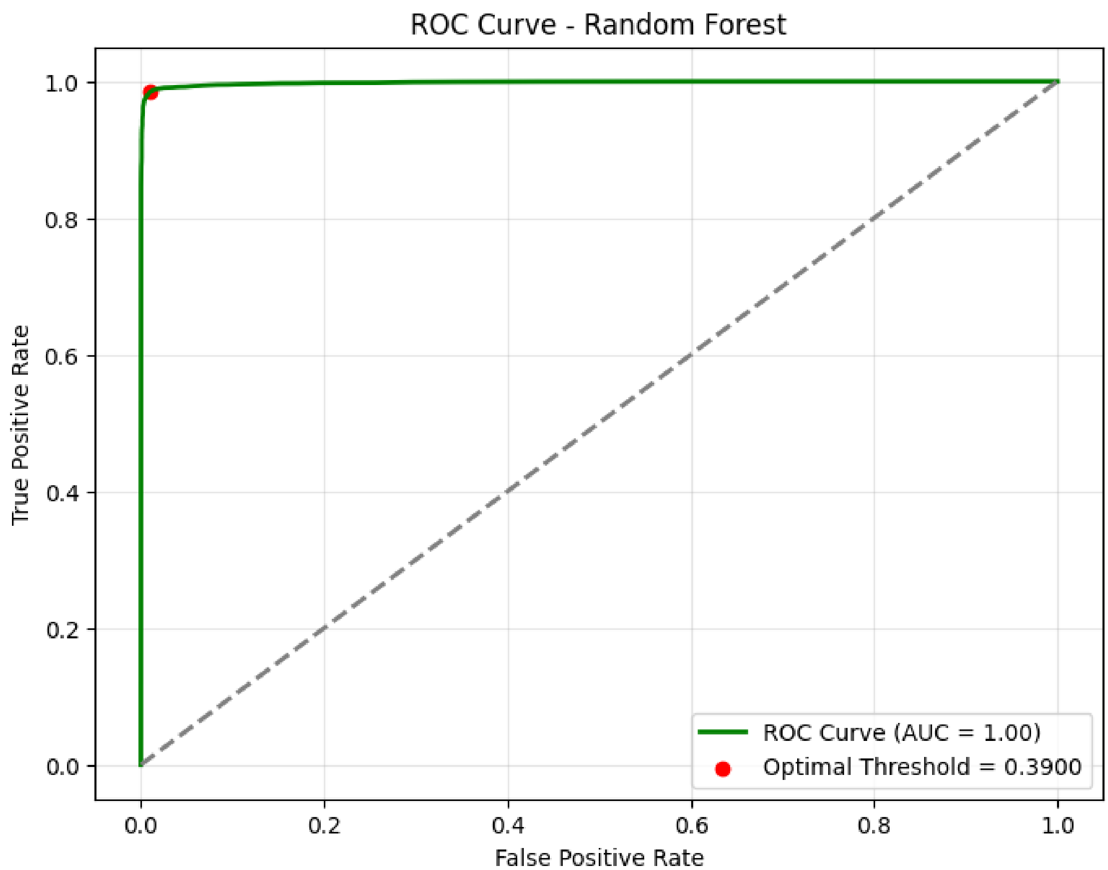
0.0.11 Random Forest Classifier Model

[122]:
```
# Initialize the Random Forest model
rf_model = RandomForestClassifier(random_state=123)

# Perform 5-fold cross-validation and predict probabilities for the positive␣
↪class
y_prob_cv_rf = cross_val_predict(rf_model, X_train_smote, y_train_smote, cv=5,␣
↪method='predict_proba')[:, 1]

# Compute ROC curve
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_train_smote, y_prob_cv_rf)

# Compute AUC
roc_auc_rf = auc(fpr_rf, tpr_rf)

# Compute Youden's J statistic and the optimal threshold
youden_j_rf = tpr_rf - fpr_rf
optimal_threshold_index_rf = np.argmax(youden_j_rf)
```

```
optimal_threshold_rf = thresholds_rf[optimal_threshold_index_rf]

# Generate predictions based on the optimal threshold
y_pred_cv_rf = (y_prob_cv_rf >= optimal_threshold_rf).astype(int)

# Print classification report
print(f"Random Forest Optimal Threshold: {optimal_threshold_rf:.4f}")
print("Random Forest Classification Report:")
print(classification_report(y_train_smote, y_pred_cv_rf))

# Generate the classification report as a dictionary
report_rf = classification_report(y_train_smote, y_pred_cv_rf, output_dict=True)

# Convert the dictionary into a pandas DataFrame
report_df_rf = pd.DataFrame(report_rf).transpose()

# Filter for classes 0 and 1
filtered_report_rf = report_df_rf.loc[["0", "1"], ["precision", "recall",␣
↪"f1-score"]]

# Compute the confusion matrix
cm_rf = confusion_matrix(y_train_smote, y_pred_cv_rf)

# Display the confusion matrix
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=['Class␣
↪0', 'Class 1'])
disp_rf.plot(cmap="summer", values_format="d")  # Using 'summer' colormap
plt.title("Confusion Matrix - Random Forest")
plt.ylabel("True Label")
plt.xlabel("Predicted Label")
plt.show()

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='green', lw=2, label=f'ROC Curve (AUC =␣
↪{roc_auc_rf:.2f})')
plt.scatter(fpr_rf[optimal_threshold_index_rf],␣
↪tpr_rf[optimal_threshold_index_rf], color='red', label=f'Optimal Threshold =␣
↪{optimal_threshold_rf:.4f}')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--', lw=2)  # Diagonal line␣
↪for random guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend(loc='lower right')
plt.grid(alpha=0.3)
plt.show()
```

```
Random Forest Optimal Threshold: 0.3900
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      7999
           1       0.99      0.98      0.99      7999

    accuracy                           0.99     15998
   macro avg       0.99      0.99      0.99     15998
weighted avg       0.99      0.99      0.99     15998
```





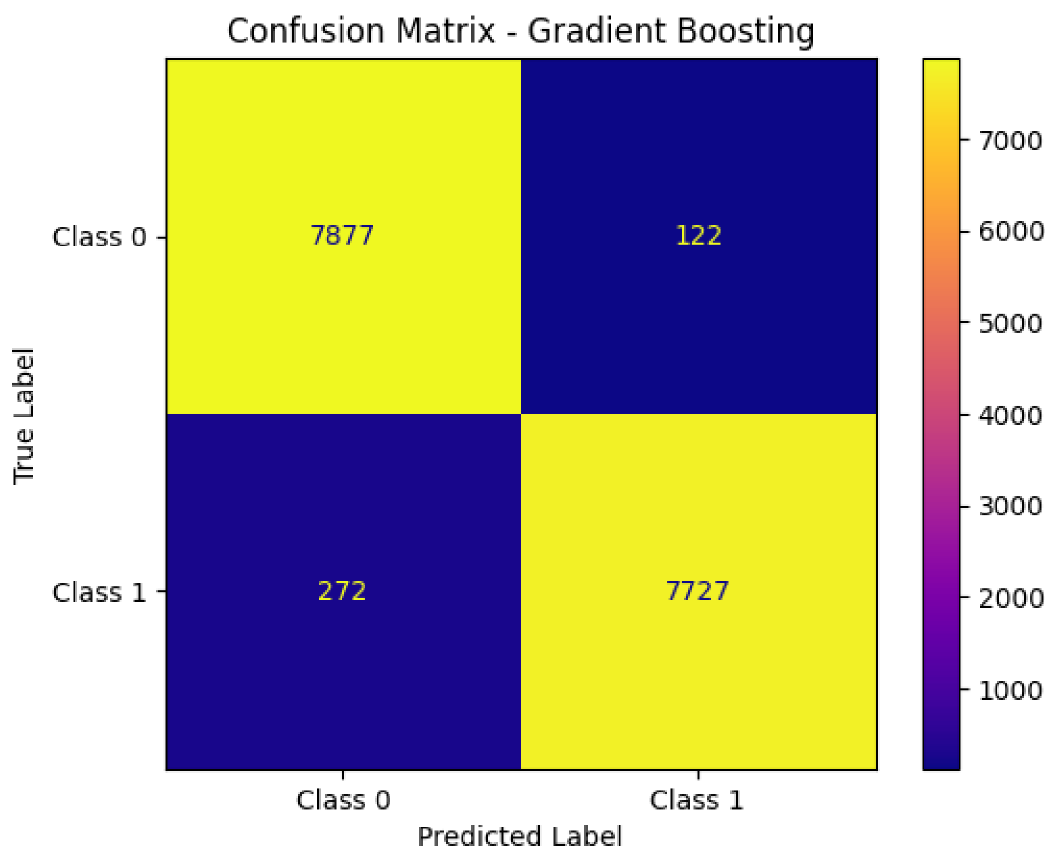
0.0.12 Gradient Boosting Classifier Model

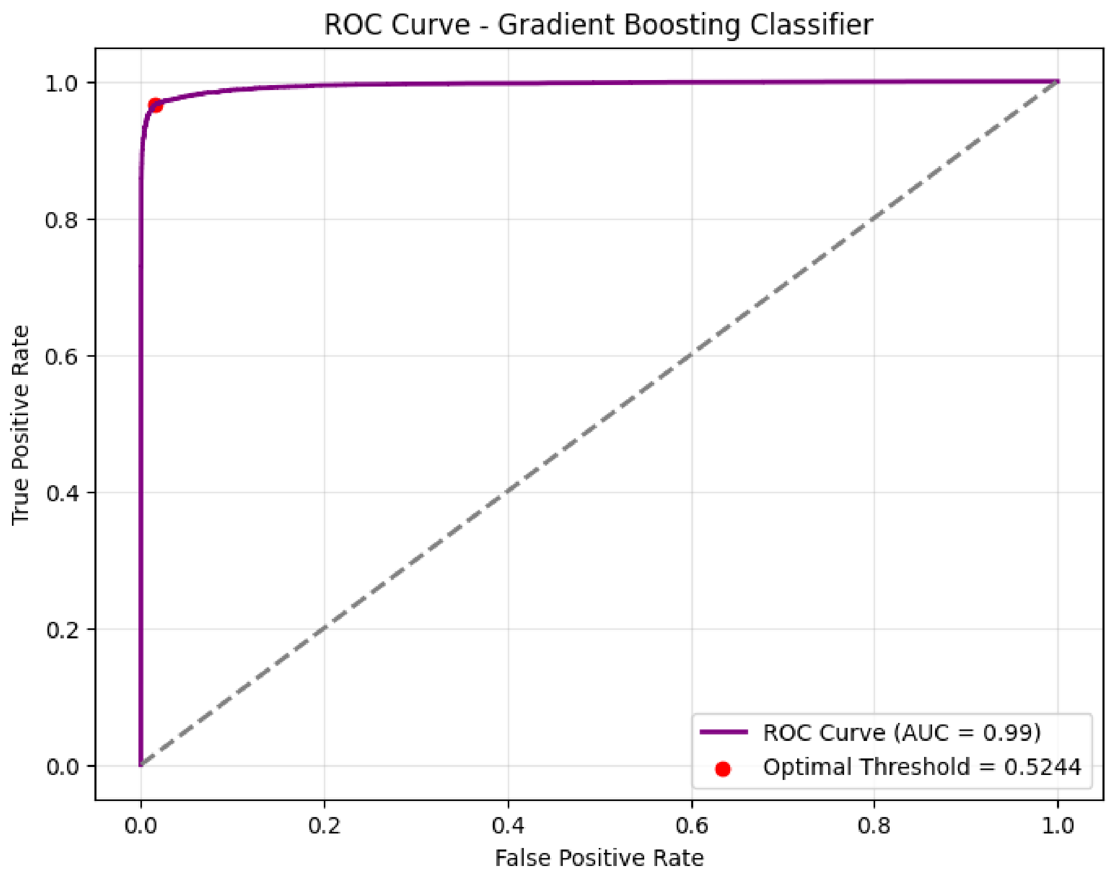
[123]:
```python
# Initialize the Gradient Boosting model
gb_model = GradientBoostingClassifier(random_state=123)

# Perform 5-fold cross-validation and predict probabilities for the positive␣
↪class
y_prob_cv_gb = cross_val_predict(gb_model, X_train_smote, y_train_smote, cv=5,␣
↪method='predict_proba')[:, 1]

# Compute ROC curve
fpr_gb, tpr_gb, thresholds_gb = roc_curve(y_train_smote, y_prob_cv_gb)

# Compute AUC
roc_auc_gb = auc(fpr_gb, tpr_gb)

# Compute Youden's J statistic and the optimal threshold
youden_j_gb = tpr_gb - fpr_gb
optimal_threshold_index_gb = np.argmax(youden_j_gb)
```

```
optimal_threshold_gb = thresholds_gb[optimal_threshold_index_gb]

# Generate predictions based on the optimal threshold
y_pred_cv_gb = (y_prob_cv_gb >= optimal_threshold_gb).astype(int)

# Print classification report
print(f"Gradient Boosting Optimal Threshold: {optimal_threshold_gb:.4f}")
print("Gradient Boosting Classification Report:")
print(classification_report(y_train_smote, y_pred_cv_gb))

# Generate the classification report as a dictionary
report_gb = classification_report(y_train_smote, y_pred_cv_gb, output_dict=True)

# Convert the dictionary into a pandas DataFrame
report_df_gb = pd.DataFrame(report_gb).transpose()

# Filter for classes 0 and 1
filtered_report_gb = report_df_gb.loc[["0", "1"], ["precision", "recall",␣
↪"f1-score"]]

# Compute the confusion matrix
cm_gb = confusion_matrix(y_train_smote, y_pred_cv_gb)

# Display the confusion matrix
disp_gb = ConfusionMatrixDisplay(confusion_matrix=cm_gb, display_labels=['Class␣
↪0', 'Class 1'])
disp_gb.plot(cmap="plasma", values_format="d")  # Using 'plasma' colormap for a␣
↪vibrant look
plt.title("Confusion Matrix - Gradient Boosting")
plt.ylabel("True Label")
plt.xlabel("Predicted Label")
plt.show()

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_gb, tpr_gb, color='purple', lw=2, label=f'ROC Curve (AUC =␣
↪{roc_auc_gb:.2f})')
plt.scatter(fpr_gb[optimal_threshold_index_gb],␣
↪tpr_gb[optimal_threshold_index_gb], color='red', label=f'Optimal Threshold =␣
↪{optimal_threshold_gb:.4f}')
plt.plot([0, 1], [0, 1], color='grey', linestyle='--', lw=2)  # Diagonal line␣
↪for random guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Gradient Boosting Classifier')
plt.legend(loc='lower right')
```

```
plt.grid(alpha=0.3)
plt.show()
```

```
Gradient Boosting Optimal Threshold: 0.5244
Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98      7999
           1       0.98      0.97      0.98      7999

    accuracy                           0.98     15998
   macro avg       0.98      0.98      0.98     15998
weighted avg       0.98      0.98      0.98     15998
```





Best Model: Gradient Boosting.

Offers high AUC (0.99) with balanced Precision and Recall, making it reliable and less prone to overfitting.

Metric to Optimize: Recall (to capture as many employees who left as possible).

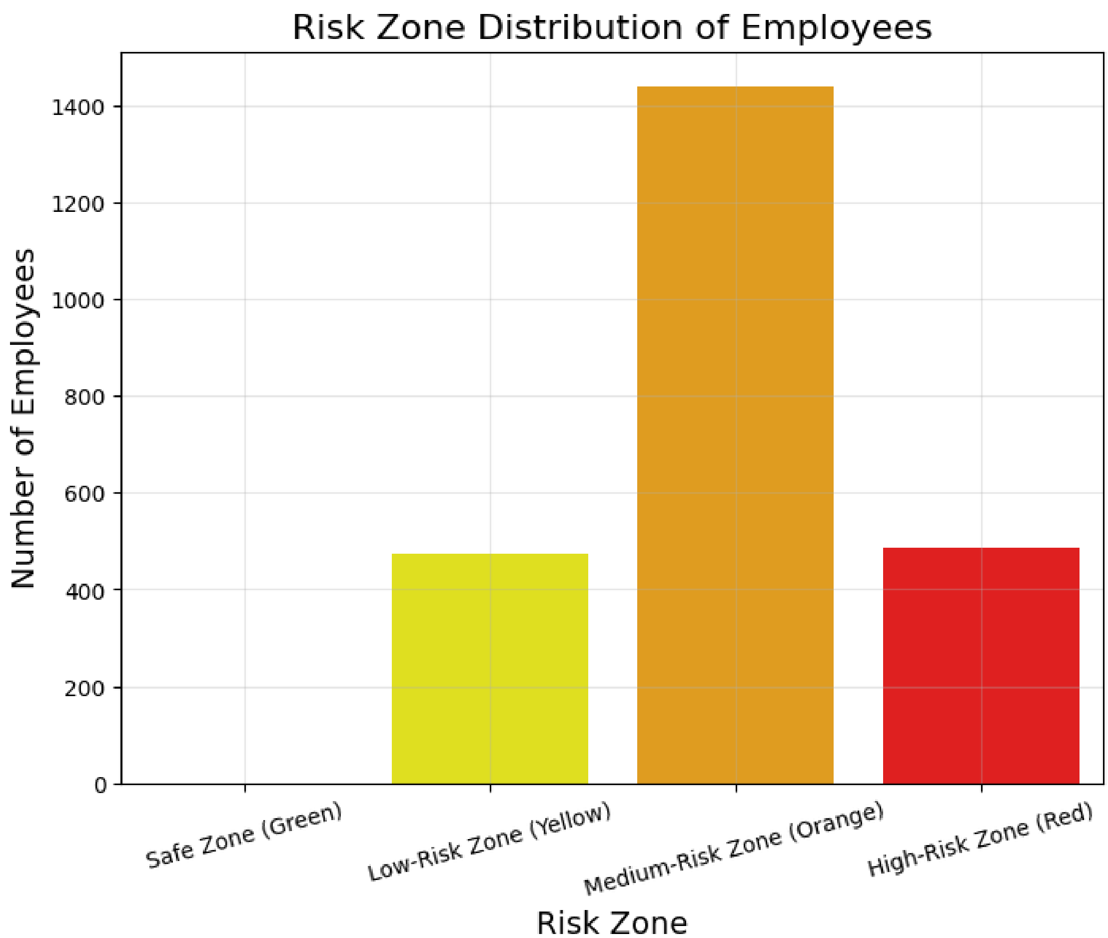
[114]:
```
# Fit the Gradient Boosting model on the entire training data
gb_model.fit(X_train_smote, y_train_smote)

# Predict probabilities for the test data
y_prob_test = gb_model.predict_proba(X_test)[:, 1]

# Add the probabilities to a DataFrame for easy categorization
test_results = pd.DataFrame({
    'EmployeeID': X_test.index,
    'Turnover_Probability': y_prob_test
})
```

```python
# Define a function to categorize employees into risk zones
def categorize_risk(prob):
    if prob < 0.2:
        return 'Safe Zone (Green)'
    elif 0.2 <= prob < 0.6:
        return 'Low-Risk Zone (Yellow)'
    elif 0.6 <= prob < 0.9:
        return 'Medium-Risk Zone (Orange)'
    else:
        return 'High-Risk Zone (Red)'

# Categorize employees into risk zones
test_results['Risk_Zone'] = test_results['Turnover_Probability'].
 ↪apply(categorize_risk)

# Ensure Risk_Zone column has a specific order
risk_zone_order = ['Safe Zone (Green)', 'Low-Risk Zone (Yellow)', 'Medium-Risk␣
 ↪Zone (Orange)', 'High-Risk Zone (Red)']
test_results['Risk_Zone'] = pd.Categorical(test_results['Risk_Zone'],␣
 ↪categories=risk_zone_order, ordered=True)

# Display the first few rows of the categorized results
print(test_results.head())


plt.figure(figsize=(8, 6))
sns.countplot(
    x='Risk_Zone',
    data=test_results,
    palette={
        'Safe Zone (Green)': 'green',
        'Low-Risk Zone (Yellow)': 'yellow',
        'Medium-Risk Zone (Orange)': 'orange',
        'High-Risk Zone (Red)': 'red'
    }
)
plt.title('Risk Zone Distribution of Employees', fontsize=16)
plt.xlabel('Risk Zone', fontsize=14)
plt.ylabel('Number of Employees', fontsize=14)
plt.xticks(rotation=15)
plt.grid(alpha=0.3)
plt.show()
```

```
   EmployeeID  Turnover_Probability                  Risk_Zone
0        8578              0.802393  Medium-Risk Zone (Orange)
1        5756              0.674578  Medium-Risk Zone (Orange)
2        3994              0.974625       High-Risk Zone (Red)
```

```
3        1784              0.417309     Low-Risk Zone (Yellow)
4       10508              0.846878  Medium-Risk Zone (Orange)
```

```
/opt/anaconda3/envs/simplienv/lib/python3.9/site-packages/sklearn/base.py:486:
UserWarning: X has feature names, but GradientBoostingClassifier was fitted
without feature names
  warnings.warn(
/var/folders/y3/g8cf6rzj7wz97zptmgzfqffc0000gn/T/ipykernel_14183/3539641596.py:3
7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(
```



0.0.13 Retention Strategies:

1. Safe Zone (Green) (<20%) Maintain satisfaction with rewards, recognition, and growth opportunities. Regular check-ins to address potential concerns.

2. Low-Risk Zone (Yellow) (20%-60%) Boost engagement with involvement in decisions and occasional perks. Address concerns through surveys or one-on-ones.

3. Medium-Risk Zone (Orange) (60%-90%) Provide growth paths, better compensation, and training. Act on employee feedback to resolve dissatisfaction.

4. High-Risk Zone (Red) (>90%) Address critical issues (pay, workload) immediately. Create personalized plans and fix root causes quickly.

[ ]:
```

```