Employee Turnover Prediction project is to construct an accurate predictive model that anticipates employee attrition. By analyzing historical employee data, encompassing job satisfaction, salary, work environment, and performance metrics, the model aims to identify employees at risk of leaving the organization. The objective is to provide actionable insights to Human Resources for implementing targeted retention strategies. This involves data preprocessing, feature engineering, and leveraging machine learning to build a robust predictive model, ultimately aiding organizations in reducing turnover rates and enhancing workplace productivity.

Import Libraries

In [39]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [40]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neural_network import MLPClassifier
```

Undertanding the data

In [41]:
```
df=pd.read_csv("Employee turnover prediction.csv")
```

In [42]:
```
df.head()
```

Out[42]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company	Wc
0	0.38	0.53	2	157	3	
1	0.80	0.86	5	262	6	
2	0.11	0.88	7	272	4	
3	0.72	0.87	5	223	5	
4	0.37	0.52	2	159	3	

In [43]:
```
df.tail()
```

Out[43]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company
14994	0.40	0.57	2	151	3
14995	0.37	0.48	2	160	3
14996	0.37	0.53	2	143	3
14997	0.11	0.96	6	280	4
14998	0.37	0.52	2	158	3

In [44]:
```
print(df.shape)
```

```
(14999, 10)
```

In [45]:
```
print(df.columns.values)
```

```
['satisfaction_level' 'last_evaluation' 'number_project'
 'average_montly_hours' 'time_spend_company' 'Work_accident' 'left'
 'promotion_last_5years' 'Department' 'salary']
```

In [46]:
```
df.describe()
```

Out[46]:

	satisfaction_level	last_evaluation	number_project	average_montly_hours	time_spend_company
count	14999.000000	14999.000000	14999.000000	14999.000000	14999.000000
mean	0.612834	0.716102	3.803054	201.050337	3.498233
std	0.248631	0.171169	1.232592	49.943099	1.460136
min	0.090000	0.360000	2.000000	96.000000	2.000000
25%	0.440000	0.560000	3.000000	156.000000	3.000000
50%	0.640000	0.720000	4.000000	200.000000	3.000000
75%	0.820000	0.870000	5.000000	245.000000	4.000000
max	1.000000	1.000000	7.000000	310.000000	10.000000

In [47]:
```
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   Department             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 data

In [48]:
```
df.isnull().any()
```

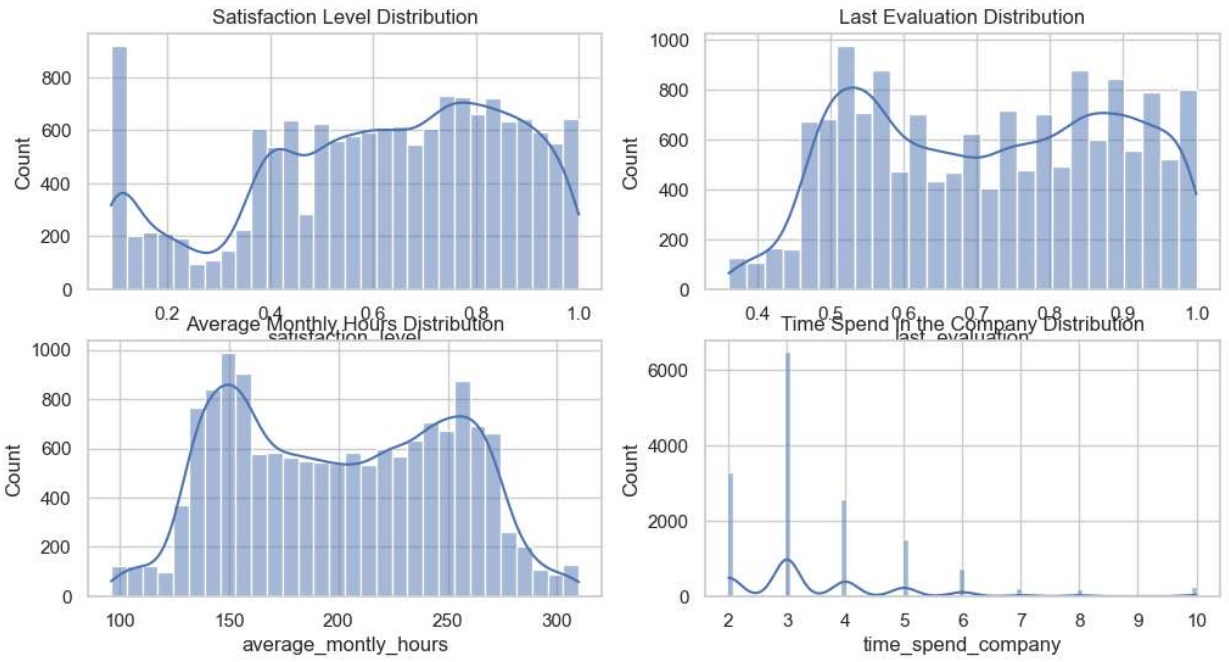
Out[48]:
```
satisfaction_level       False
last_evaluation          False
number_project           False
average_montly_hours     False
time_spend_company       False
Work_accident            False
left                     False
promotion_last_5years    False
Department               False
salary                   False
dtype: bool
```

Data Visualization:

Distributions of Numerical Features:

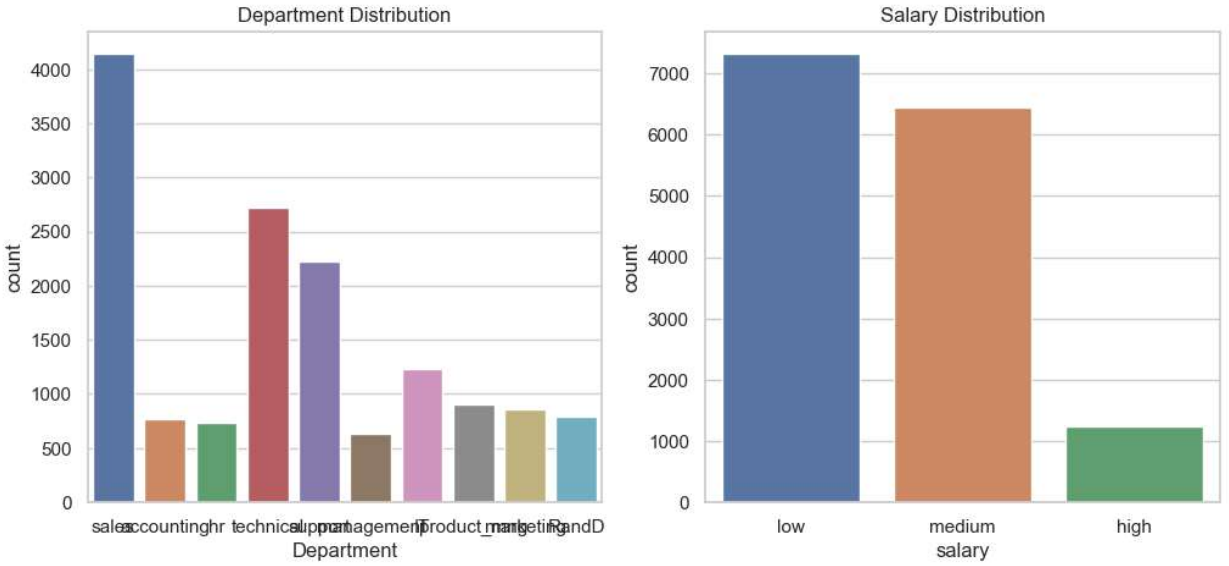
In [49]:
```
sns.set(style="whitegrid")
plt.figure(figsize=(12, 6))

plt.subplot(2, 2, 1)
sns.histplot(df['satisfaction_level'], kde=True)
plt.title('Satisfaction Level Distribution')

plt.subplot(2, 2, 2)
sns.histplot(df['last_evaluation'], kde=True)
plt.title('Last Evaluation Distribution')

plt.subplot(2, 2, 3)
sns.histplot(df['average_montly_hours'], kde=True)
plt.title('Average Monthly Hours Distribution')

plt.subplot(2, 2, 4)
sns.histplot(df['time_spend_company'], kde=True)
plt.title('Time Spend in the Company Distribution')
```

```
plt.show()
```



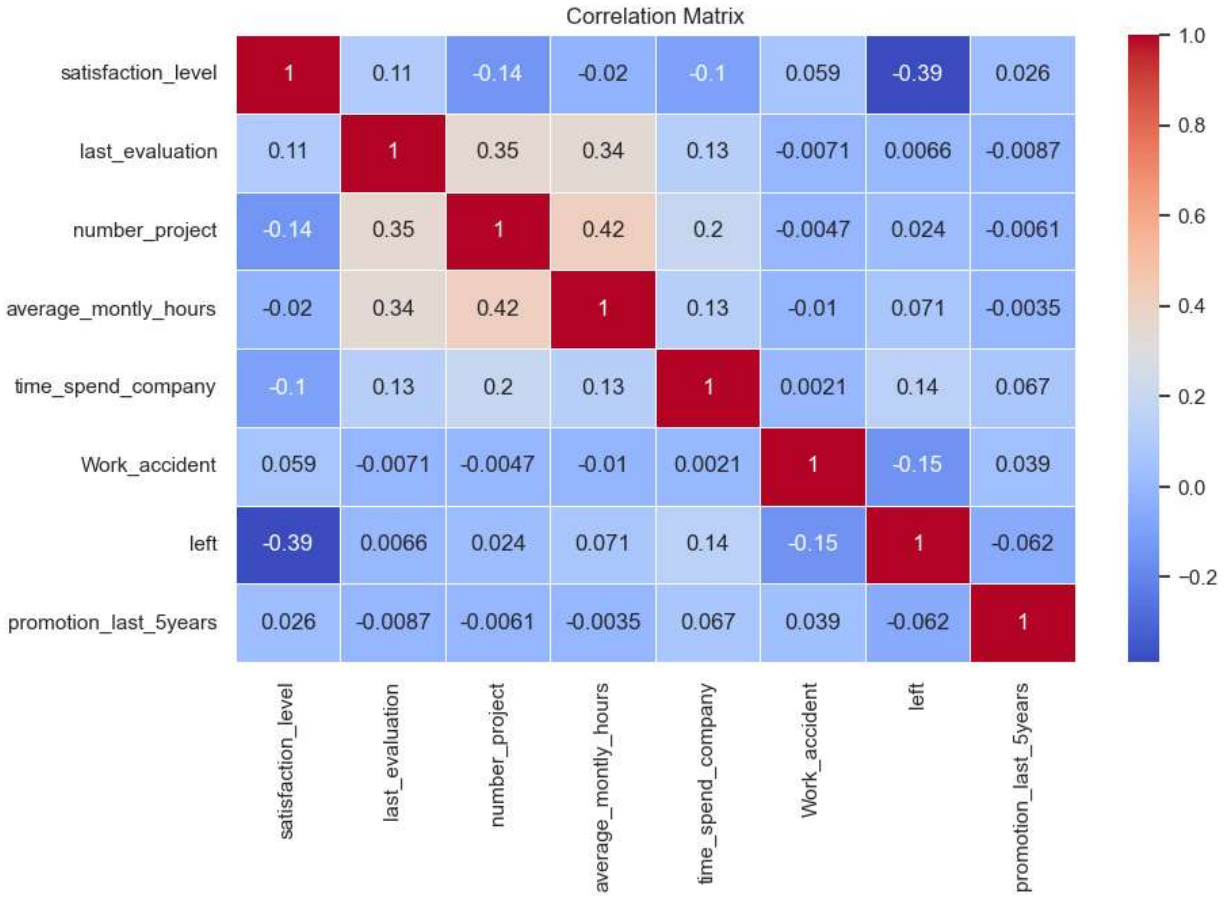
Categorical Features

In [50]:
```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
sns.countplot(data=df, x='Department')
plt.title('Department Distribution')

plt.subplot(1, 2, 2)
sns.countplot(data=df, x='salary')
plt.title('Salary Distribution')

plt.show()
```

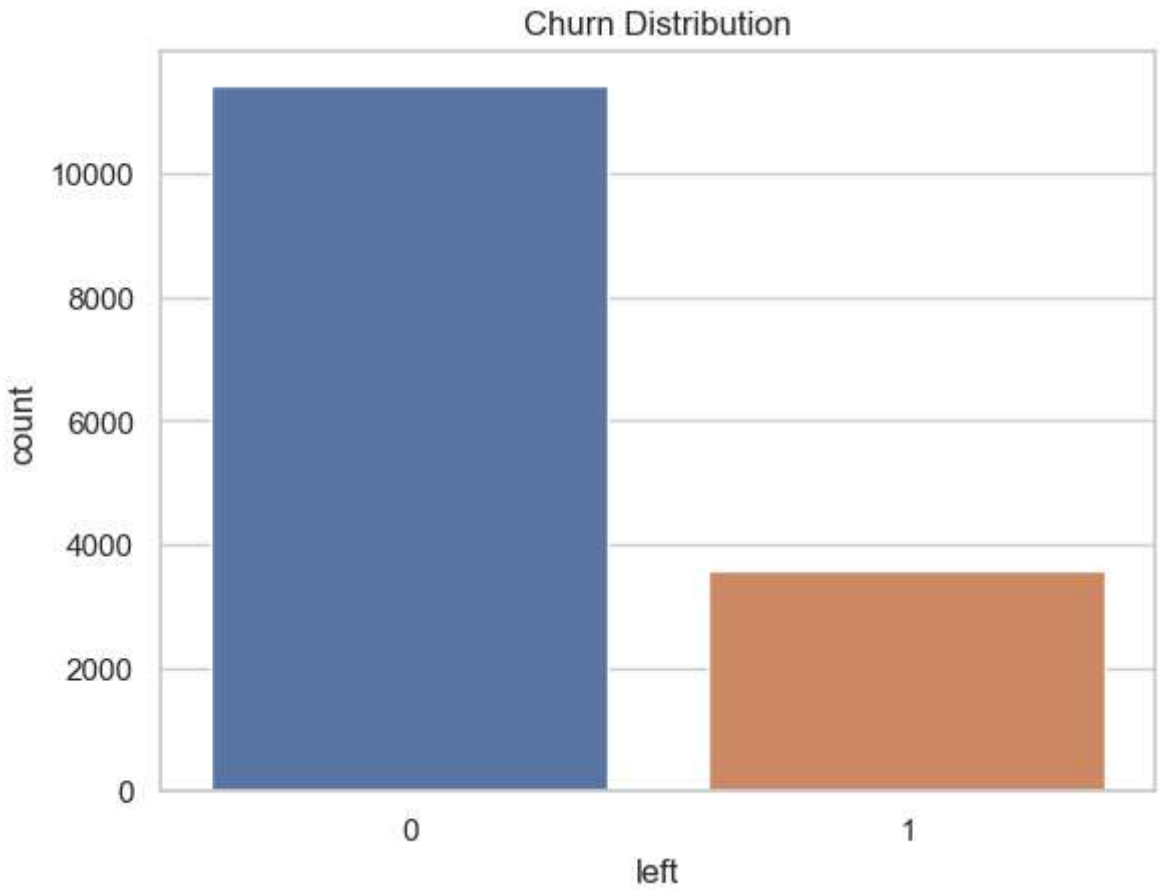


In [51]:
```
# Exclude non-numeric columns before calculating the correlation
numeric_df = df.select_dtypes(include=[float, int])
```

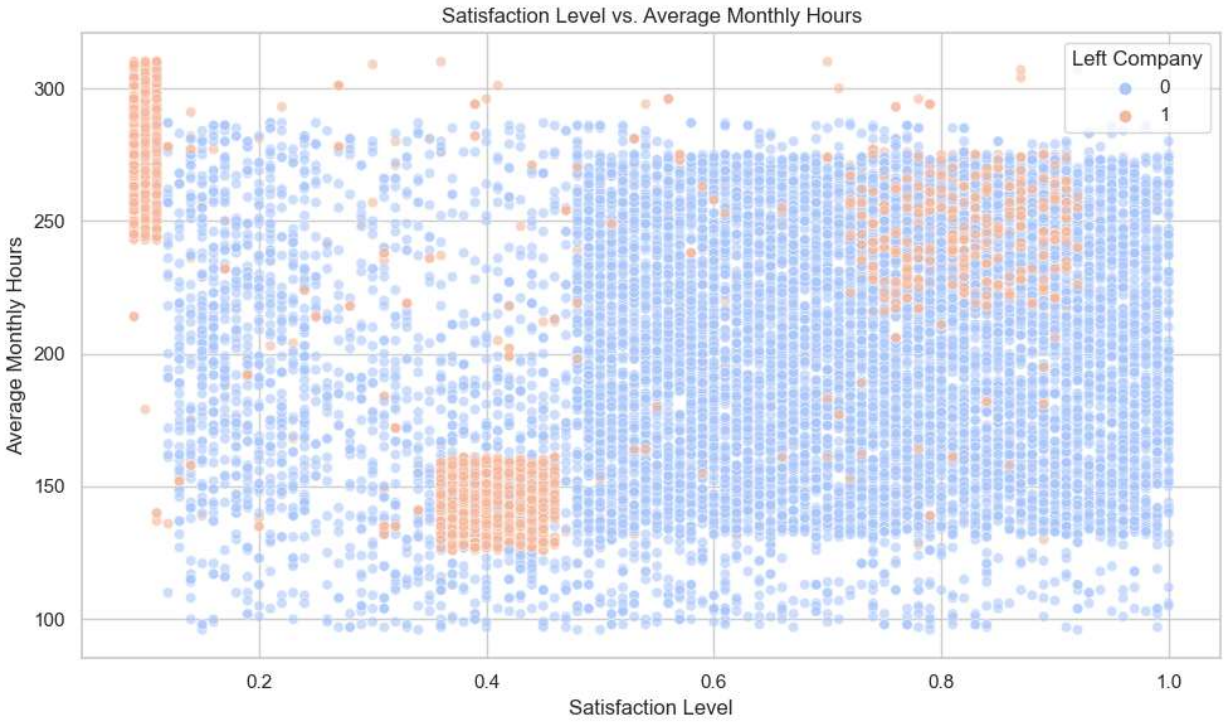
```
correlation_matrix = numeric_df.corr()

plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



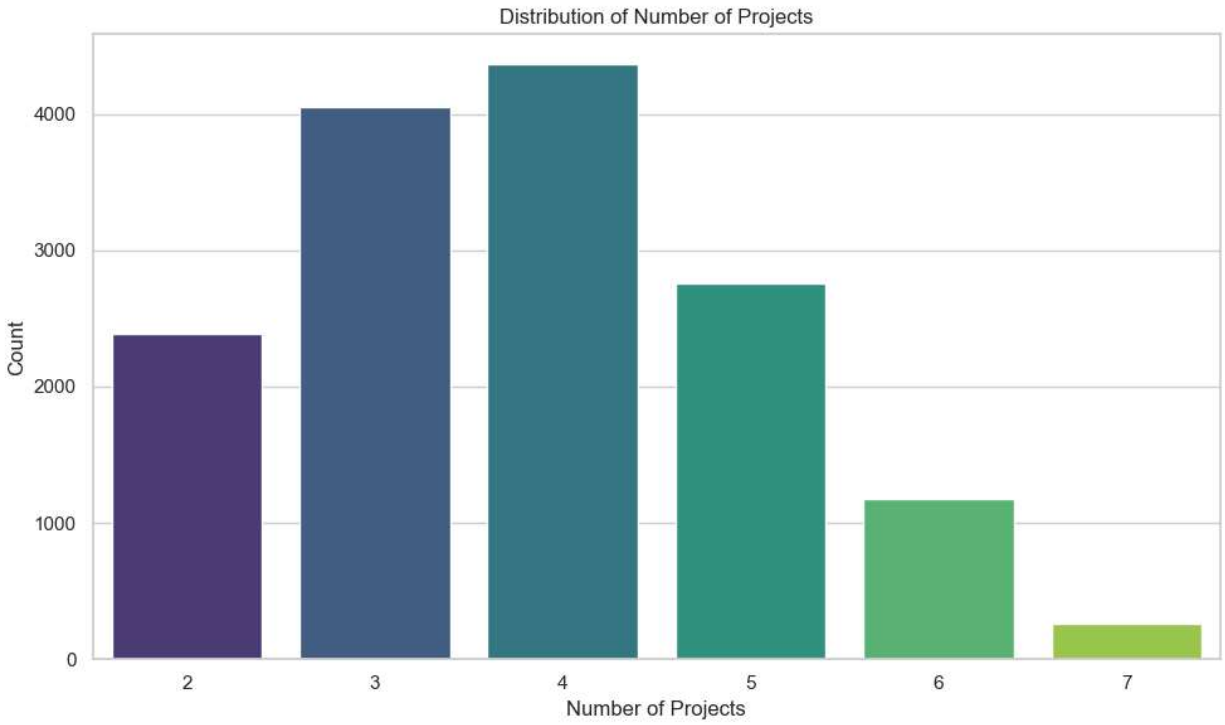
In [52]:
```
#Churn Distribution
sns.countplot(data=df, x='left')
plt.title('Churn Distribution')
plt.show()
```



In [53]:
```
# Scatterplot to visualize the relationship between satisfaction level and ave
plt.figure(figsize=(10, 6))
sns.scatterplot(x='satisfaction_level', y='average_montly_hours', data=df, hue=
plt.title('Satisfaction Level vs. Average Monthly Hours')
plt.xlabel('Satisfaction Level')
plt.ylabel('Average Monthly Hours')
plt.legend(title='Left Company')
plt.tight_layout()
plt.show()
```



In [54]:
```
# Visualize the distribution of the number of projects
f, ax = plt.subplots(figsize=(10, 6))
sns.countplot(x='number_project', data=df, palette='viridis', ax=ax)
ax.set_title('Distribution of Number of Projects')
ax.set_xlabel('Number of Projects')
ax.set_ylabel('Count')
plt.tight_layout()
plt.show()
```



Split the data

In [55]:
```
# Split the data into features (X) and the target variable (y)
X = df.drop('left', axis=1)
y = df['left']
```

Data Preprocessing

In [56]:
```
# Encode categorical variables (e.g., 'Department' and 'salary') using one-hot
X = pd.get_dummies(X, columns=['Department', 'salary'], drop_first=True)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rando
```

Model Selection and Training

Random Forest

In [57]:
```
# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
rf_classifier.fit(X_train, y_train)
```

Out[57]:
```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [58]:
```
# Predict on the testing set
y_pred = rf_classifier.predict(X_test)

# Calculate the accuracy
accuracy_rf = accuracy_score(y_test, y_pred)
```

In [59]:
```
# Generate a classification report
report = classification_report(y_test, y_pred)

# Display the accuracy and the classification report
print(f'Accuracy: {accuracy_rf}')
print(report)
```

```
Accuracy: 0.9871111111111112
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      3428
           1       0.99      0.96      0.97      1072

    accuracy                           0.99      4500
   macro avg       0.99      0.98      0.98      4500
weighted avg       0.99      0.99      0.99      4500
```

In [60]:
```
# Generate the classification report as a dictionary
classification_dict = classification_report(y_test, y_pred, output_dict=True)
```

```
# Convert the dictionary to a DataFrame for visualization
classification_df = pd.DataFrame(classification_dict).T

# Create a heatmap for visualization
plt.figure(figsize=(8, 6))
sns.heatmap(classification_df.iloc[:-1, :-3], annot=True, cmap='Blues', cbar=Fa
plt.title('Classification Report Heatmap')
plt.show()
```

Support Vector Machines

In [61]:
```
# Initialize the Support Vector Machine classifier
svm_classifier = SVC(kernel='linear', random_state=42)

# Train the classifier
svm_classifier.fit(X_train, y_train)
```

Out[61]:
```
SVC(kernel='linear', random_state=42)
```

In [62]:
```
# Predict on the testing set
y_pred_svm = svm_classifier.predict(X_test)

# Calculate the accuracy
accuracy_svm = accuracy_score(y_test, y_pred_svm)
```

In [63]:
```
# Generate a classification report
report_svm = classification_report(y_test, y_pred_svm)

# Display the accuracy and the classification report
print(accuracy_svm),
print(report_svm)
```

```
0.7717777777777778
              precision    recall  f1-score   support

           0       0.80      0.94      0.86      3428
           1       0.55      0.24      0.34      1072

    accuracy                           0.77      4500
   macro avg       0.67      0.59      0.60      4500
weighted avg       0.74      0.77      0.74      4500
```

Logistic Regression

In [64]:
```
# Initialize the Logistic Regression classifier
logreg_classifier = LogisticRegression(random_state=42)

# Train the classifier
logreg_classifier.fit(X_train, y_train)
```

```
D:\Anaconda\Lib\site-packages\sklearn\linear_model\_logistic.py:460: Convergen
ceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regress
ion
  n_iter_i = _check_optimize_result(
```

Out[64]:
```
LogisticRegression(random_state=42)
```

In [65]:
```
# Predict on the testing set
y_pred_logreg = logreg_classifier.predict(X_test)
```

In [66]:
```
# Calculate the accuracy
accuracy_logreg = accuracy_score(y_test, y_pred_logreg)
```

In [67]:
```
# Generate a classification report
report_logreg = classification_report(y_test, y_pred_logreg)
```

In [68]:
```
# Display the accuracy and the classification report
print(accuracy_logreg)
print(report_logreg)
```

```
0.7895555555555556
              precision    recall  f1-score   support

           0       0.82      0.93      0.87      3428
           1       0.60      0.35      0.44      1072

    accuracy                           0.79      4500
   macro avg       0.71      0.64      0.66      4500
weighted avg       0.77      0.79      0.77      4500
```

Gradient Boosting

In [69]:
```
# Initialize the Gradient Boosting Classifier
gb_model = GradientBoostingClassifier(random_state=42)

# Train the model
gb_model.fit(X_train, y_train)
```

Out[69]:
```
GradientBoostingClassifier(random_state=42)
```

In [70]:
```
# Predict on the test set
y_pred_gb = gb_model.predict(X_test)
```

In [71]:
```
# Evaluate the model
accuracy_gb = accuracy_score(y_test, y_pred_gb)
class_report_gb = classification_report(y_test, y_pred_gb)
```

In [72]:
```
# Output the accuracy and the classification report for both models
print(accuracy_gb)
print(class_report_gb)
```

```
0.9724444444444444
              precision    recall  f1-score   support

           0       0.98      0.99      0.98      3428
           1       0.96      0.92      0.94      1072

    accuracy                           0.97      4500
   macro avg       0.97      0.95      0.96      4500
weighted avg       0.97      0.97      0.97      4500
```

Multi-Layer Perceptron (MLP) Classifie

In [73]:
```
# Initialize the Multi-Layer Perceptron (MLP) Classifier
mlp_model = MLPClassifier(random_state=42)

# Train the model
mlp_model.fit(X_train, y_train)
```

```
D:\Anaconda\Lib\site-packages\sklearn\neural_network\_multilayer_perceptron.p
y:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reac
hed and the optimization hasn't converged yet.
  warnings.warn(
```

Out[73]:

```
MLPClassifier(random_state=42)
```

In [74]:

```
# Predict on the test set
y_pred_mlp = mlp_model.predict(X_test)
```

In [75]:

```
# Evaluate the model
accuracy_mlp = accuracy_score(y_test, y_pred_mlp)
class_report_mlp = classification_report(y_test, y_pred_mlp)

# Output the accuracy and the classification report for the MLP model
print(accuracy_mlp)
print(class_report_mlp)
```

```
0.938
              precision    recall  f1-score   support

           0       0.96      0.96      0.96      3428
           1       0.87      0.87      0.87      1072

    accuracy                           0.94      4500
   macro avg       0.91      0.91      0.91      4500
weighted avg       0.94      0.94      0.94      4500
```

In [76]:

```
accuracy_df = pd.DataFrame({'Model': ['Random Forest','Support Vector Machines
accuracy_df
```

Out[76]:

	Model	Accuracy
0	Random Forest	0.987111
1	Support Vector Machines	0.771778
2	Logistic Regression	0.789556
3	Gradient Boosting	0.972444
4	MLP	0.938000

In [ ]: