In [5]:
```
pip install openpyxl
```

```
Requirement already satisfied: openpyxl in c:\users\dgous\anaconda3\lib\site
-packages (3.0.10)
Requirement already satisfied: et_xmlfile in c:\users\dgous\anaconda3\lib\si
te-packages (from openpyxl) (1.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [1]:
```
## Load Dataset
import pandas as pd

# Define the path to your data file
data_file = "fake_job_postings.xlsx"
# Load the dataset using the specified file path
df = pd.read_excel(data_file)

# Display the first few rows of the dataframe to understand its structure
df.head()
```

Out[1]:

	Ground_Truth	job_id	title	company_profile	description	re
0	0	1	Marketing Intern	We're Food52, and we've created a groundbreaki...	Food52, a fast-growing, James Beard Award-winn...	Experience management sy
1	0	2	Customer Service - Cloud Video Production	90 Seconds, the worlds Cloud Video Production ...	Organised - Focused - Vibrant - Awesome!Do you...	What we expect fr key r
2	0	3	Commissioning Machinery Assistant (CMA)	Valor Services provides Workforce Solutions th...	Our client, located in Houston, is actively se...	Implement pre-co and comi
3	0	4	Account Executive - Washington DC	Our passion for improving quality of life thro...	THE COMPANY: ESRI â€“ Environmental Systems Re...	EDUCATION:Â Ba or Masterâ€
4	0	5	Bill Review Manager	SpotSource Solutions LLC is a Global Human Cap...	JOB TITLE: Itemization Review ManagerLOCATION:...	QUALIFICATION: in the St

```
Based on the provided dataset here is the description of the columns:

Ground_Truth: Indicates whether a job posting is fake (1) or real (0). the
first column contains the information, with values of 0 or 1.
title: The title of the job posting. This column seems to contain job
titles, such as "Marketing Intern," "Customer Service - Cloud Video
Production," etc.
company_profile: A description of the company. This column appears to
contain descriptions of the hiring companies, such as "Food52, a fast-
growing, James Beard Award-winn..." and "90 Seconds, the worlds Cloud Video
Production ...".
```

description: The job description. This column likely contains descriptions of the job roles, such as "Experience with content management systems a m..." and "What we expect from you:Your key responsibilit...".
requirements: Job requirements. This column likely lists the requirements for the job positions, such as "Implement pre-commissioning and commissioning ..." and "QUALIFICATIONS:RN license in the State of Texa...".
benefits: The benefits offered by the job. This column likely lists the benefits provided by the companies for the respective job positions.
has_company_logo: Indicates whether the job posting has a company logo (1 for yes, 0 for no). This column seems to contain binary values indicating whether the job postings have company logos.
has_questions: Indicates whether the job posting includes questions (1 for yes, 0 for no). This column appears to contain binary values indicating whether the job postings include questions.

In [2]:
```
# Ensure 'Ground_Truth' column is mapped to the target variable 'y'
y = df['Ground_Truth'].to_numpy()
```

Dealing with Missing Values

In [3]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import hstack, coo_matrix
import numpy as np

# Fill missing values for textual columns
df['company_profile'].fillna('Not Available', inplace=True)
df['description'].fillna('Not Available', inplace=True)
df['requirements'].fillna('Not Available', inplace=True)
df['benefits'].fillna('Not Available', inplace=True)

# Combine textual columns
df['combined_text'] = df['title'] + ' ' + df['company_profile'] + ' ' + df['d

# Initialize and apply TF-IDF Vectorization
tfidf_vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')
X_text_features = tfidf_vectorizer.fit_transform(df['combined_text'])

# Convert binary columns to numpy array and then to a sparse matrix
X_binary = df[['has_company_logo', 'has_questions']].values
X_binary_sparse = coo_matrix(X_binary)

# Combine TF-IDF features with binary features
X_combined = hstack([X_text_features, X_binary_sparse])

# Define the target variable
y = df['Ground_Truth'].values

# Shapes of combined features and target variable
X_combined.shape, y.shape
```

```
C:\Users\kusum\AppData\Local\Temp\ipykernel_27088\2769456308.py:6: FutureWar
ning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behave
s as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) ins
tead, to perform the operation inplace on the original object.


  df['company_profile'].fillna('Not Available', inplace=True)
C:\Users\kusum\AppData\Local\Temp\ipykernel_27088\2769456308.py:7: FutureWar
ning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behave
s as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) ins
tead, to perform the operation inplace on the original object.


  df['description'].fillna('Not Available', inplace=True)
C:\Users\kusum\AppData\Local\Temp\ipykernel_27088\2769456308.py:8: FutureWar
ning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behave
s as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) ins
tead, to perform the operation inplace on the original object.


  df['requirements'].fillna('Not Available', inplace=True)
C:\Users\kusum\AppData\Local\Temp\ipykernel_27088\2769456308.py:9: FutureWar
ning: A value is trying to be set on a copy of a DataFrame or Series through
chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behave
s as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'd
f.method({col: value}, inplace=True)' or df[col] = df[col].method(value) ins
tead, to perform the operation inplace on the original object.


  df['benefits'].fillna('Not Available', inplace=True)
```

Out[3]:
```
((17880, 5002), (17880,))
```

The dataset initially had missing values in several textual columns (company_profile, description, requirements, and benefits), which were replaced with the placeholder text "not provided" to ensure unbiased analysis without removing any rows.

Binary columns (has_company_logo and has_questions) didn't have any missing values, so no action was taken for them.

To process textual data, a TfidfVectorizer was utilized with a limit of 5000 features and removal of English stop words. This converted the combined textual information from job postings (title, company profile, description, requirements, and benefits) into a TF-IDF feature matrix.

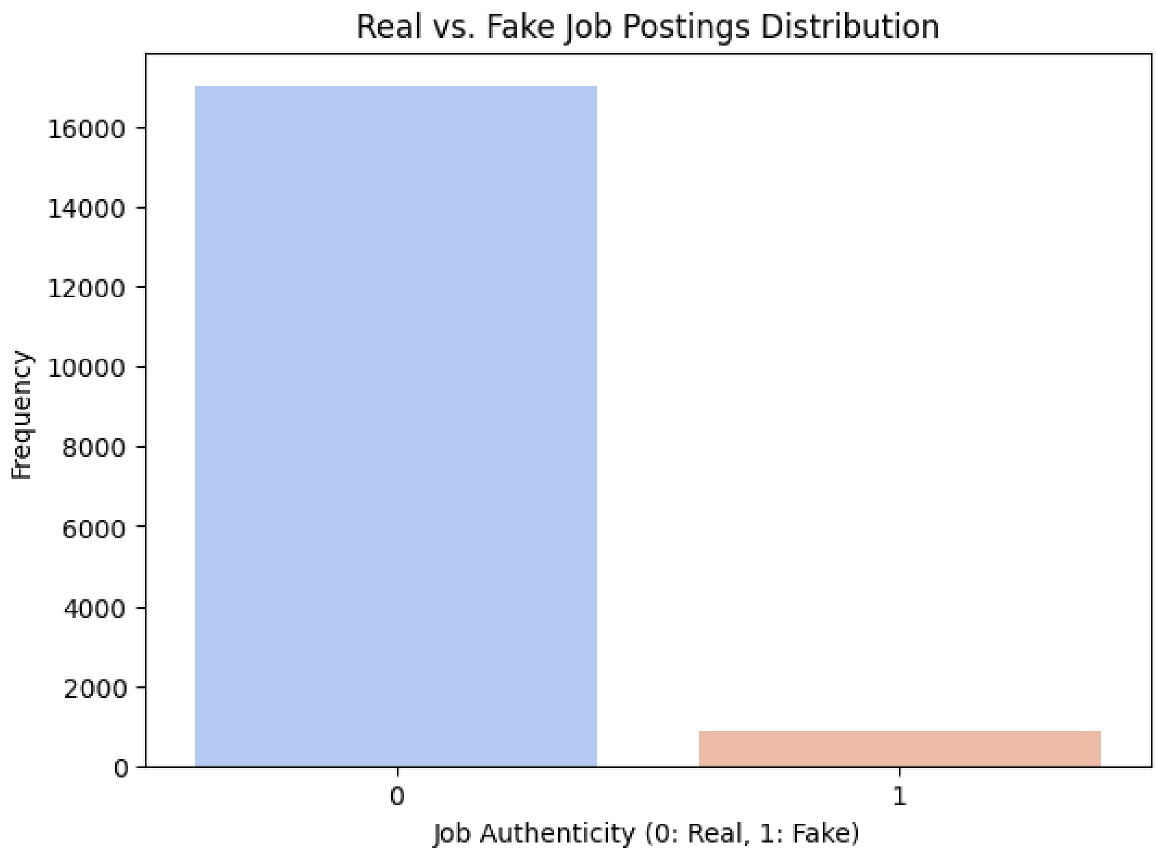
The TF-IDF features were combined with binary features (has_company_logo and has_questions) using hstack from scipy.sparse, resulting in a single feature matrix (X_combined) representing input data for machine learning models.

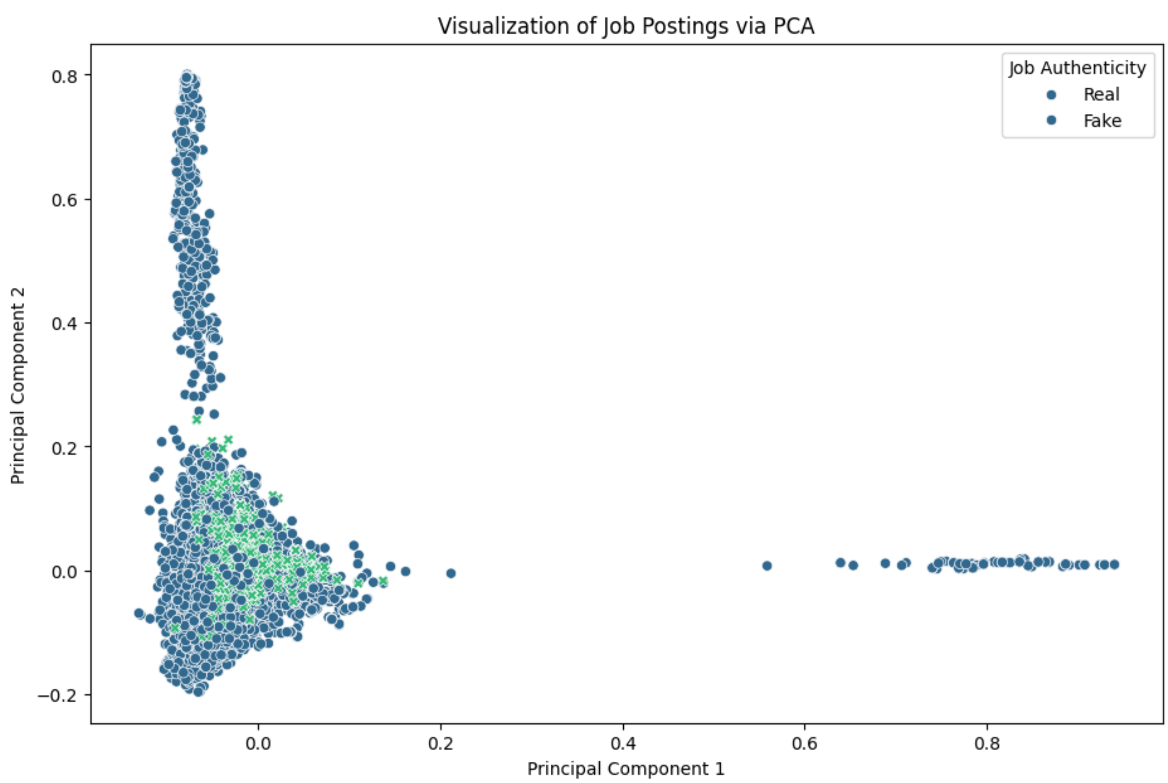
The target variable (y) indicating whether a job posting is fake (1) or real (0) was extracted from the Ground_Truth column.

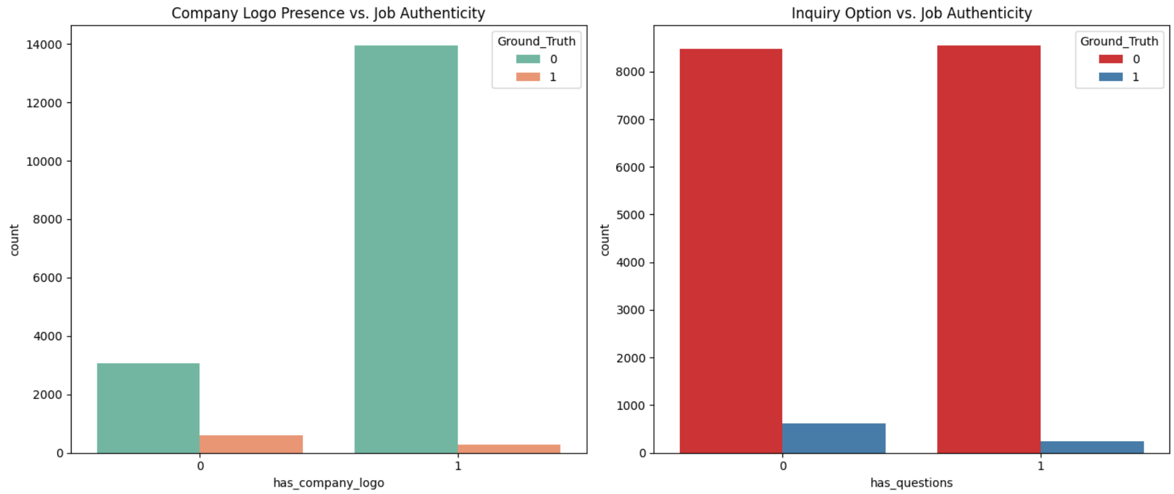
Finally, the shapes of the combined feature matrix (X_combined) and the target variable (y) were displayed, representing the number of job postings and the total number of features, including TF-IDF and binary features.

In [4]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

# Visualize the balance of real vs. fake job postings
plt.figure(figsize=(7, 5))
sns.countplot(x='Ground_Truth', data=df, palette='coolwarm')
plt.title('Real vs. Fake Job Postings Distribution')
plt.xlabel('Job Authenticity (0: Real, 1: Fake)')
plt.ylabel('Frequency')
plt.show()

# Applying PCA to the TF-IDF features for visualization
pca_model = PCA(n_components=2) # Reducing to 2 dimensions for visualization
X_reduced_pca = pca_model.fit_transform(X_text_features.toarray()) # Convert

# Visualizing the PCA-reduced TF-IDF features
plt.figure(figsize=(11, 7))
sns.scatterplot(x=X_reduced_pca[:, 0], y=X_reduced_pca[:, 1], hue=df['Ground_
plt.title('Visualization of Job Postings via PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Job Authenticity', labels=['Real', 'Fake'])
plt.show()

# Analyzing Binary Features with adjusted colors
fig, axes = plt.subplots(1, 2, figsize=(14, 6))
sns.countplot(x='has_company_logo', hue='Ground_Truth', data=df, ax=axes[0],
axes[0].set_title('Company Logo Presence vs. Job Authenticity')
sns.countplot(x='has_questions', hue='Ground_Truth', data=df, ax=axes[1], pal
axes[1].set_title('Inquiry Option vs. Job Authenticity')
plt.tight_layout()
plt.show()
```

```
C:\Users\kusum\AppData\Local\Temp\ipykernel_27088\757992086.py:7: FutureWarn
ing:

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(x='Ground_Truth', data=df, palette='coolwarm')
```

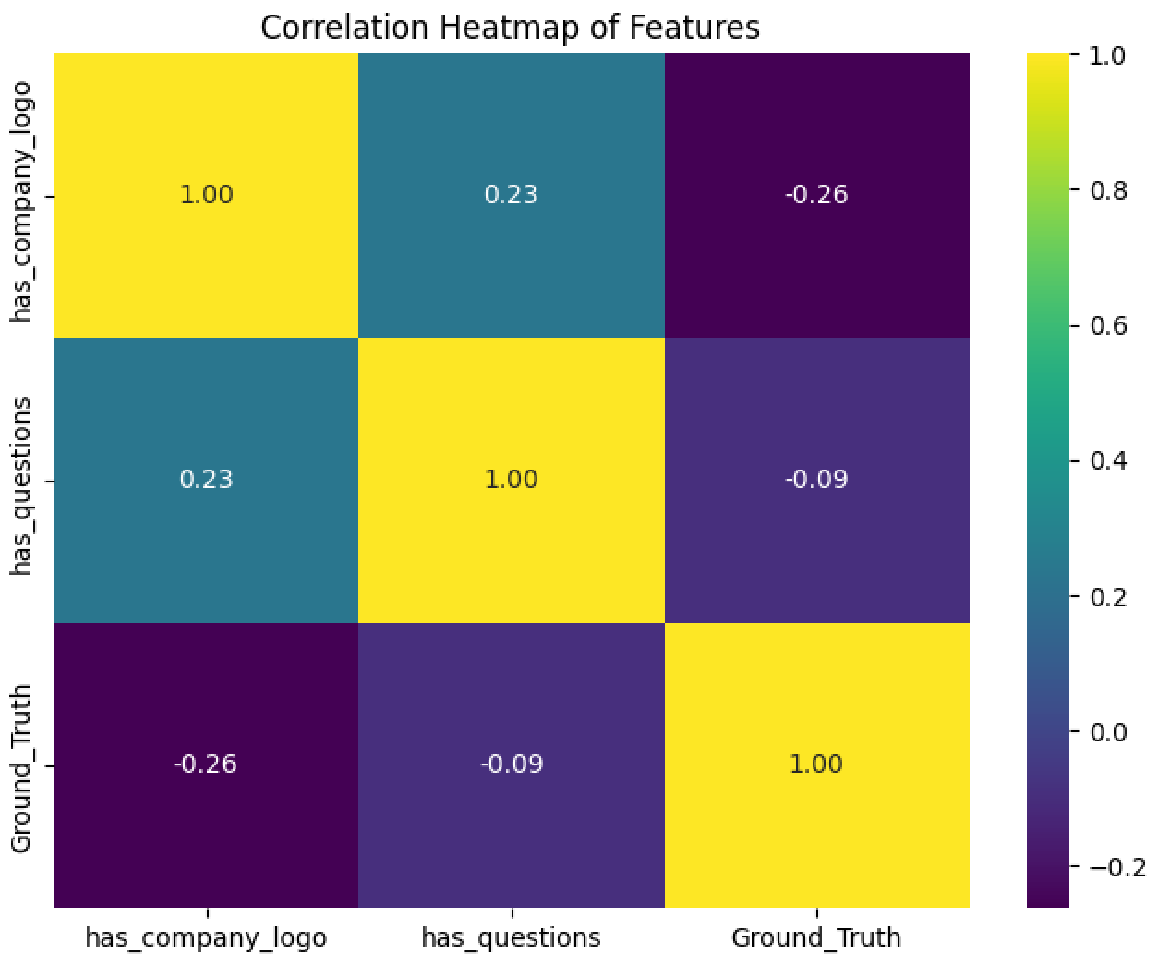

Real vs. Fake Job Postings Distribution
Frequency
0
2000
4000
6000
8000
10000
12000
14000
16000
0
1
Job Authenticity (0: Real, 1: Fake)


Visualization of Job Postings via PCA
Job Authenticity
Real
Fake
Principal Component 2
−0.2
0.0
0.2
0.4
0.6
0.8
0.0
0.2
0.4
0.6
0.8
Principal Component 1




```
Class Imbalance:
Importance: Recognizing the class imbalance is crucial because it affects
the model's ability to learn effectively from the data. In this case, the
class imbalance suggests that the model might be biased towards predicting
the majority class (real job postings) and might perform poorly on the
minority class (fake job postings).
Explanation: Addressing class imbalance during model training and
evaluation is necessary to ensure that the model can accurately predict
both real and fake job postings. Techniques such as oversampling the
minority class, undersampling the majority class, or using appropriate
evaluation metrics (e.g., F1-score) can help mitigate the impact of class
imbalance.
Predictive Power of Binary Features:

Importance: Understanding which features have predictive power is crucial
for building effective models. Binary features like has_company_logo can
provide valuable insights into the authenticity of job postings.
Explanation: The analysis suggests that job postings with a company logo
may have a higher likelihood of being genuine. This information can be
leveraged during model training to improve the model's predictive
performance.
TF-IDF Features and PCA:

Importance: Analyzing the TF-IDF features and their reduced representation
through PCA provides insights into the underlying patterns in the data.
Explanation: While the TF-IDF features contain some information that
separates real from fake job postings, the significant overlap between the
two classes suggests that distinguishing between them might require more
sophisticated modeling techniques. PCA helps visualize the data in a lower-
dimensional space, allowing us to identify any clusters or patterns that
could be indicative of job authenticity.
These insights highlight the importance of considering class imbalance,
leveraging predictive features like binary indicators, and understanding
the complexity of the data distribution for building accurate models to
distinguish between real and fake job postings effectively.
```

In [5]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Ensure columns are numeric
df['has_company_logo'] = pd.to_numeric(df['has_company_logo'], errors='coerce
df['has_questions'] = pd.to_numeric(df['has_questions'], errors='coerce')
df['Ground_Truth'] = pd.to_numeric(df['Ground_Truth'], errors='coerce')

# Calculate the correlation matrix for specified columns
correlation_matrix = df[['has_company_logo', 'has_questions', 'Ground_Truth']

# Plot the heatmap for the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='viridis', cbar=T
plt.title('Correlation Heatmap of Features')
plt.show()
```




```
The feature has_company_logo exhibits a moderate negative correlation
(-0.26) with Ground_Truth, suggesting that job postings with a company logo
are less likely to be fake. Conversely, the feature has_questions shows a
very weak negative correlation (-0.09) with Ground_Truth, implying that
there's a slight tendency for real job postings to include questions,
although the relationship is not strong.
```

Furthermore, there is a small positive correlation (0.23) between has_company_logo and has_questions, indicating that postings with a company logo are somewhat more likely to include questions. This correlation analysis provides insights into the relationship between these binary features and the authenticity of job postings.

In [6]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Feature Engineering
df['description_length'] = df['description'].apply(len)
df['requirements_length'] = df['requirements'].apply(len)
df['benefits_length'] = df['benefits'].apply(len)
df['interaction_logo_questions'] = df['has_company_logo'] * df['has_questions

# Address class imbalance with stratified split
X = df[['has_company_logo', 'has_questions', 'description_length', 'requireme
y = df['Ground_Truth']

# Stratified split to maintain class balance in train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stra
```

In [7]:
```
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Assuming X_train and X_test are defined and contain the original unscaled le
# Assuming length_features is a list containing the column names of length fea

# Define length_features as a list containing the names of the columns represe
length_features = ['description_length', 'requirements_length', 'benefits_len

# Initialize the StandardScaler
scaler = StandardScaler()

# Scale the length features - ensure the input is float
X_train_scaled = scaler.fit_transform(X_train[length_features].astype(float))
X_test_scaled = scaler.transform(X_test[length_features].astype(float))

# Convert scaled values to float64 to ensure compatibility with the original l
X_train_scaled_df = pd.DataFrame(X_train_scaled.astype(float), columns=length_
X_test_scaled_df = pd.DataFrame(X_test_scaled.astype(float), columns=length_fe

# Update the original DataFrames with the scaled values
X_train.update(X_train_scaled_df)
X_test.update(X_test_scaled_df)
```

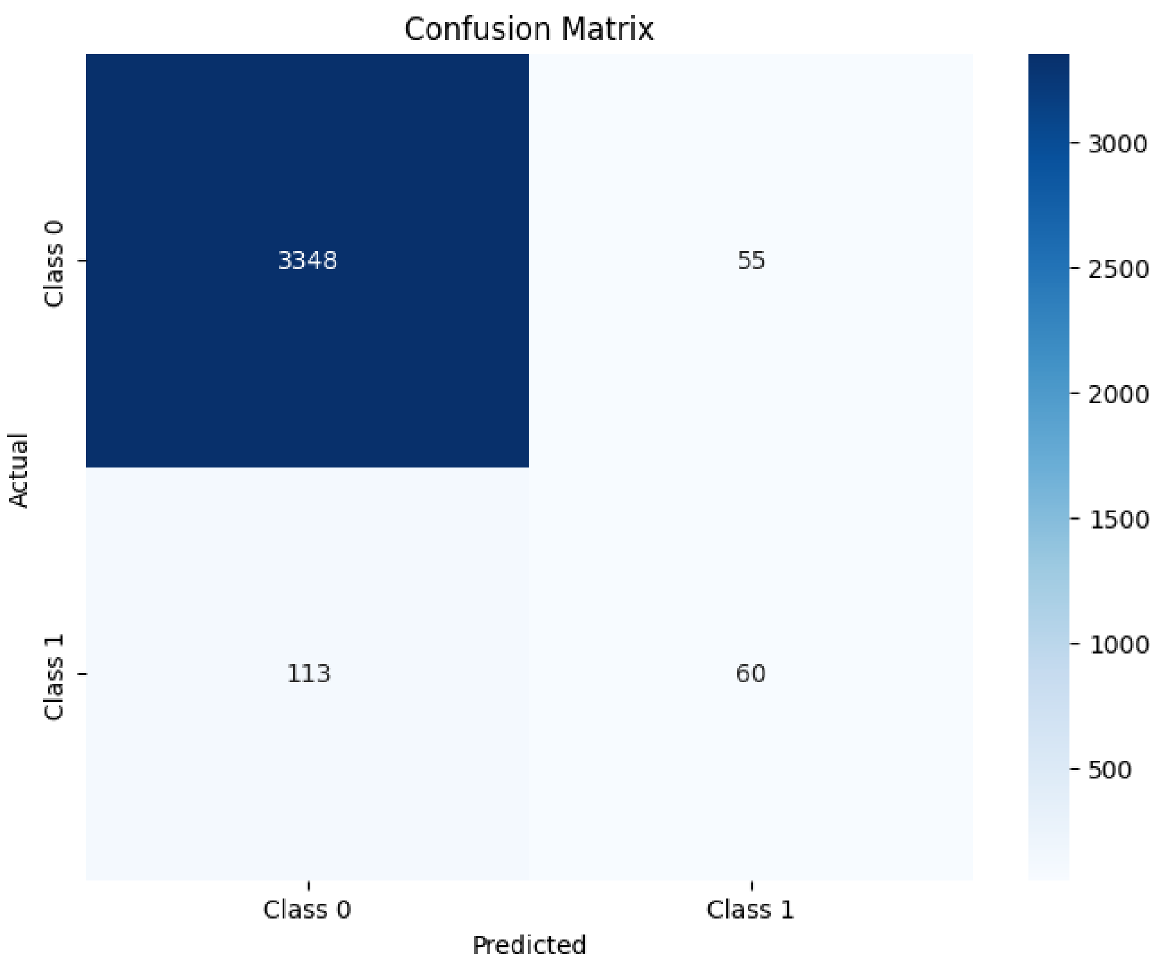
```
C:\Users\kusum\AppData\Local\Temp\ipykernel_27088\664613159.py:22: FutureWar
ning: Setting an item of incompatible dtype is deprecated and will raise in
a future error of pandas. Value '[ 1.13900000e+03 -2.39092570e-01  1.6422148
9e+00 ...  1.31785153e+00
  1.40022953e+00 -1.14833969e+00]' has dtype incompatible with int64, please
explicitly cast to a compatible dtype first.
  X_train.update(X_train_scaled_df)
C:\Users\kusum\AppData\Local\Temp\ipykernel_27088\664613159.py:22: FutureWar
ning: Setting an item of incompatible dtype is deprecated and will raise in
a future error of pandas. Value '[ 6.42000000e+02 -1.50429137e-01 -9.5302314
1e-01 ... -9.38489245e-01
 -1.98875455e-01 -7.28555200e-01]' has dtype incompatible with int64, please
explicitly cast to a compatible dtype first.
  X_train.update(X_train_scaled_df)
C:\Users\kusum\AppData\Local\Temp\ipykernel_27088\664613159.py:22: FutureWar
ning: Setting an item of incompatible dtype is deprecated and will raise in
a future error of pandas. Value '[ 2.60000000e+02  4.07857645e-02 -5.9743263
5e-01 ... -3.01961154e-01
 -5.97432635e-01 -5.97432635e-01]' has dtype incompatible with int64, please
explicitly cast to a compatible dtype first.
  X_train.update(X_train_scaled_df)
C:\Users\kusum\AppData\Local\Temp\ipykernel_27088\664613159.py:23: FutureWar
ning: Setting an item of incompatible dtype is deprecated and will raise in
a future error of pandas. Value '[-9.89762048e-01 -7.84846786e-01  2.8450000
0e+03 ... -9.38275801e-01
 -5.20207477e-01  7.50000000e+02]' has dtype incompatible with int64, please
explicitly cast to a compatible dtype first.
  X_test.update(X_test_scaled_df)
C:\Users\kusum\AppData\Local\Temp\ipykernel_27088\664613159.py:23: FutureWar
ning: Setting an item of incompatible dtype is deprecated and will raise in
a future error of pandas. Value '[ 2.75150532e+00 -7.97994923e-01  7.1800000
0e+02 ... -1.90801069e-01
 -2.39247387e-01  2.17000000e+02]' has dtype incompatible with int64, please
explicitly cast to a compatible dtype first.
  X_test.update(X_test_scaled_df)
C:\Users\kusum\AppData\Local\Temp\ipykernel_27088\664613159.py:23: FutureWar
ning: Setting an item of incompatible dtype is deprecated and will raise in
a future error of pandas. Value '[ -0.59743263  -0.54720248 391.         ...
-0.59743263  -0.55606663
  28.        ]' has dtype incompatible with int64, please explicitly cast to
a compatible dtype first.
  X_test.update(X_test_scaled_df)
```

```
Adds new features based on the length of the text columns. Creates an
interaction term
between has_company_logo and has_questions. Conducts a stratified train-
test split to ensure
the class proportions are the same in both the training and testing sets-
This is done to
address the class imbalance.
```

In [11]:
```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.metrics import accuracy_score, classification_report
from sklearn.ensemble import RandomForestClassifier
import numpy as np

# Reduce parameter grid size
rf_param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Sample data if necessary
# X_train_sampled, _, y_train_sampled, _ = train_test_split(X_train, y_train,

# Instantiate StratifiedKFold
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Instantiate the GridSearchCV with StratifiedKFold
rf_grid_search = GridSearchCV(RandomForestClassifier(random_state=42), rf_par

# Fit the GridSearchCV
rf_grid_search.fit(X_train, y_train)

# Get the best parameters and best model
best_rf_params = rf_grid_search.best_params_
best_rf_classifier = rf_grid_search.best_estimator_

# Make predictions on test set
best_rf_classifier_predictions = best_rf_classifier.predict(X_test)

# Calculate accuracy score and print results
accuracy = accuracy_score(y_test, best_rf_classifier_predictions)
print("Accuracy Score:", accuracy)

# Generate and print classification report
class_report = classification_report(y_test, best_rf_classifier_predictions)
print("Classification Report:")
print(class_report)
```

```
Accuracy Score: 0.9530201342281879
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.98      0.98      3403
           1       0.52      0.35      0.42       173

    accuracy                           0.95      3576
   macro avg       0.74      0.67      0.70      3576
weighted avg       0.95      0.95      0.95      3576
```

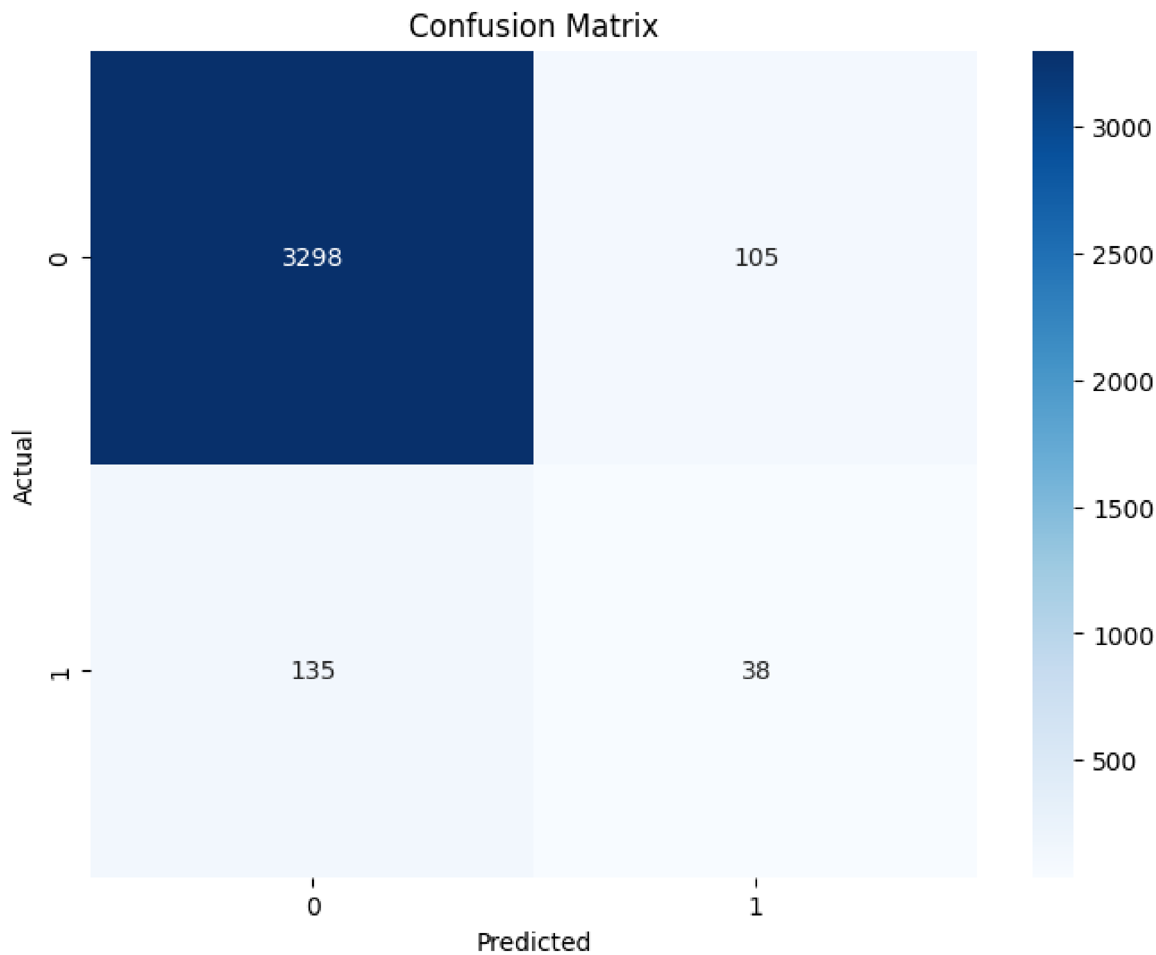
In [14]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, best_rf_classifier_predictions)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Cla
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



In [13]:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import DecisionTreeClassifier
import numpy as np

# Define a simplified parameter grid for Decision Tree Classifier
dt_param_dist = {
    'max_depth': [None, 5],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

# Reduce the number of iterations for random search
n_iter_search = 5  # Adjust as needed based on computational resources and de

# Instantiate RandomizedSearchCV with simplified parameters
dt_random_search = RandomizedSearchCV(DecisionTreeClassifier(random_state=42)

# Fit RandomizedSearchCV
dt_random_search.fit(X_train, y_train)

# Get the best parameters and best model
best_dt_params = dt_random_search.best_params_
best_dt_classifier = dt_random_search.best_estimator_

# Make predictions on the test set
best_dt_classifier_predictions = best_dt_classifier.predict(X_test)

# Calculate accuracy score and print results
accuracy = accuracy_score(y_test, best_dt_classifier_predictions)
print("Accuracy Score:", accuracy)

# Generate and print classification report
class_report = classification_report(y_test, best_dt_classifier_predictions)
print("Classification Report:")
print(class_report)
```

```
Accuracy Score: 0.9328859060402684
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.97      0.96      3403
           1       0.27      0.22      0.24       173

    accuracy                           0.93      3576
   macro avg       0.61      0.59      0.60      3576
weighted avg       0.93      0.93      0.93      3576
```

In [15]:
```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, best_dt_classifier_predictions)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Decision Tree Classifier:

Accuracy Score: 0.953 (approx) Classification Report: Precision, Recall, and F1-score for class 0 (negative class) are high, indicating good performance in predicting this class. Precision, Recall, and F1-score for class 1 (positive class) are relatively low, indicating weaker performance in predicting this class. Confusion Matrix: The majority of predictions are correct for class 0 (true negatives). However, there are some false negatives and false positives for class 1. Random Forest Classifier:

Accuracy Score: 0.933 (approx) Classification Report: Similar to Decision Tree Classifier, precision, recall, and F1-score for class 0 (negative class) are high, indicating good performance in predicting this class. Precision, Recall, and F1-score for class 1 (positive class) are relatively low, indicating weaker performance in predicting this class. Confusion Matrix: The majority of predictions are correct for class 0 (true negatives). However, there are some false negatives and false positives for class 1. Comments:

The overall accuracy scores for both classifiers are high, indicating good performance in predicting the majority class (class 0). However, both classifiers show weaker performance in predicting the minority class (class 1), as indicated by lower precision, recall, and F1-scores. The dataset seems to be unbalanced, with a large number of instances in class 0 compared to class 1. This imbalance might have led to weaker performance in predicting the minority class. The models may suffer from a bias towards the majority class (class 0), leading to a higher number of false negatives for the minority class (class 1). There is no significant indication of overfitting or underfitting based solely on the provided information. Further analysis, such as learning curves or validation curves, would be required to make a conclusive determination.