In [87]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder

logs_df = pd.read_csv(r"C:\Users\junai\OneDrive - Middlesex University\Applied Data
scores_df = pd.read_csv(r"C:\Users\junai\OneDrive - Middlesex University\Applied Da
unseen_df = pd.read_csv(r"C:\Users\junai\OneDrive - Middlesex University\Applied Da
```

In [88]:
```
logs_df.head(5)
```

Out[88]:

	StudentId	Time	Type	Action
0	72af	28/05/23, 10:51	User report	Grade user report viewed
1	72af	28/05/23, 10:51	System	Course viewed
2	c426	27/05/23, 15:53	System	Course viewed
3	0326	26/05/23, 22:22	System	Course viewed
4	8b7a	26/05/23, 21:52	System	Course viewed

In [89]:
```
scores_df.head(5)
```

Out[89]:

	StudentId	Grade
0	c426	2nd
1	8de3	2nd
2	d969	2nd
3	6d29	1st
4	1dd9	1st

In [90]:
```
unseen_df.head(5)
```

Out[90]:

	StudentId
0	aca3
1	4f2c
2	295e
3	d1d7
4	6cd6

In [153…
```
# Define a mapping for grade categories to numerical values
grade_mapping = {'1st': 1, '2nd': 2, '3rd': 3, 'Fail': 0}
```

In [155…
```
# Map the 'Grade' column to numerical values
scores_df['Grade'] = scores_df['Grade'].map(grade_mapping)
```

In [156…
```
merged_df = pd.merge(logs_df, scores_df, on='StudentId', how='inner')
```

In [157…
```
# Convert 'Time' column to datetime format
merged_df['Time'] = pd.to_datetime(merged_df['Time'], errors='coerce')
```

```
C:\Users\junai\AppData\Local\Temp\ipykernel_23080\3727546246.py:2: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify a format.
  merged_df['Time'] = pd.to_datetime(merged_df['Time'], errors='coerce')
```

In [158…
```
# Drop rows with invalid datetime values
merged_df = merged_df.dropna(subset=['Time'])
```

In [159…
```
merged_df.head(5)
```

Out[159]:

	StudentId	Time	Type	Action	Grade
0	72af	2023-05-28 10:51:00	User report	Grade user report viewed	1
1	72af	2023-05-28 10:51:00	System	Course viewed	1
2	72af	2023-05-26 09:58:00	User report	Grade user report viewed	1
3	72af	2023-05-26 09:58:00	System	Course viewed	1
4	72af	2023-05-22 16:15:00	User report	Grade user report viewed	1

In [160…
```
# Check the column names in the merged DataFrame
print(merged_df.columns)
```

```
Index(['StudentId', 'Time', 'Type', 'Action', 'Grade'], dtype='object')
```

In [161…
```
type_counts = merged_df.groupby(['StudentId', 'Type']).size().unstack(fill_value=0)
```

In [162…
```
type_counts.head(5)
```

Out[162]:

Type	StudentId	Type_Assignment	Type_File	Type_File submissions	Type_Folder	Type_Forum	Type_Kaltura Video Resource
0	0126	0	1	0	14	0	0
1	0139	0	0	0	51	0	1
2	020c	0	32	0	136	57	105
3	026e	0	10	0	69	6	37
4	0326	12	59	2	244	23	66

In [163…
```
action_counts = merged_df.groupby(['StudentId', 'Action']).size().unstack(fill_valu
```

In [164…
```
action_counts.head(5)
```

Out[164]:

	Action	StudentId	Action_A file has been uploaded.	Action_A submission has been submitted.	Action_Add Submission	Action_Calendar event created	Action_Calendar event deleted	Action_C a comp up
0		0126	0	0	2	0	0	
1		0139	0	0	2	0	0	
2		020c	0	0	2	0	0	
3		026e	0	0	4	0	0	
4		0326	1	1	6	0	0	

5 rows × 48 columns

In [165…
```
merged_df['DayOfWeek'] = merged_df['Time'].dt.day_name()
merged_df['HourOfDay'] = merged_df['Time'].dt.hour
```

In [166…
```
merged_df.head(5)
```

Out[166]:

	StudentId	Time	Type	Action	Grade	DayOfWeek	HourOfDay
0	72af	2023-05-28 10:51:00	User report	Grade user report viewed	1	Sunday	10
1	72af	2023-05-28 10:51:00	System	Course viewed	1	Sunday	10
2	72af	2023-05-26 09:58:00	User report	Grade user report viewed	1	Friday	9
3	72af	2023-05-26 09:58:00	System	Course viewed	1	Friday	9
4	72af	2023-05-22 16:15:00	User report	Grade user report viewed	1	Monday	16

In [167…
```
merged_df['TimeBetweenActions'] = merged_df.groupby('StudentId')['Time'].diff().fil
```

In [168…
```
merged_df['TimeBetweenActions'].head(5)
```

Out[168]:
```
0                    0
1      0 days 00:00:00
2    -3 days +23:07:00
3      0 days 00:00:00
4    -4 days +06:17:00
Name: TimeBetweenActions, dtype: object
```

In [ ]:
```
```

In [169…
```
# Count of actions
action_counts = merged_df.groupby('StudentId')['Action'].count().reset_index(name='
```

In [170…
```
# Create a column for TotalActions
merged_df['TotalActions'] = 1  # Each row represents an action
```

In [171…
```
# Convert 'Time' to numeric format representing time duration in seconds
merged_df['Time'] = (merged_df['Time'] - merged_df['Time'].min()).dt.total_seconds(
```

In [172…
```
merged_df['ActionFrequency'] = merged_df.groupby('StudentId')['TotalActions'].rolli
```

In [173…
```
merged_df['LastActionType'] = merged_df.groupby('StudentId')['Type'].shift(1)
```

In [174…
```
merged_df['SessionDuration'] = merged_df.groupby(['StudentId', 'Type'])['Time'].dif
```

In [175…
```
merged_df.head(5)
```

Out[175]:

	StudentId	Time	Type	Action	Grade	DayOfWeek	HourOfDay	TimeBetweenActions
0	72af	43468680.0	User report	Grade user report viewed	1	Sunday	10	0
1	72af	43468680.0	System	Course viewed	1	Sunday	10	0 days 00:00:00
2	72af	43292700.0	User report	Grade user report viewed	1	Friday	9	-3 days +23:07:00
3	72af	43292700.0	System	Course viewed	1	Friday	9	0 days 00:00:00
4	72af	42969720.0	User report	Grade user report viewed	1	Monday	16	-4 days +06:17:00

In [ ]:

In [176…
```
# Convert categorical features to numerical using Label Encoding
le = LabelEncoder()
merged_df['Type_encoded'] = le.fit_transform(merged_df['Type'])

merged_df['Action_encoded'] = le.fit_transform(merged_df['Action'])
```

In [ ]:

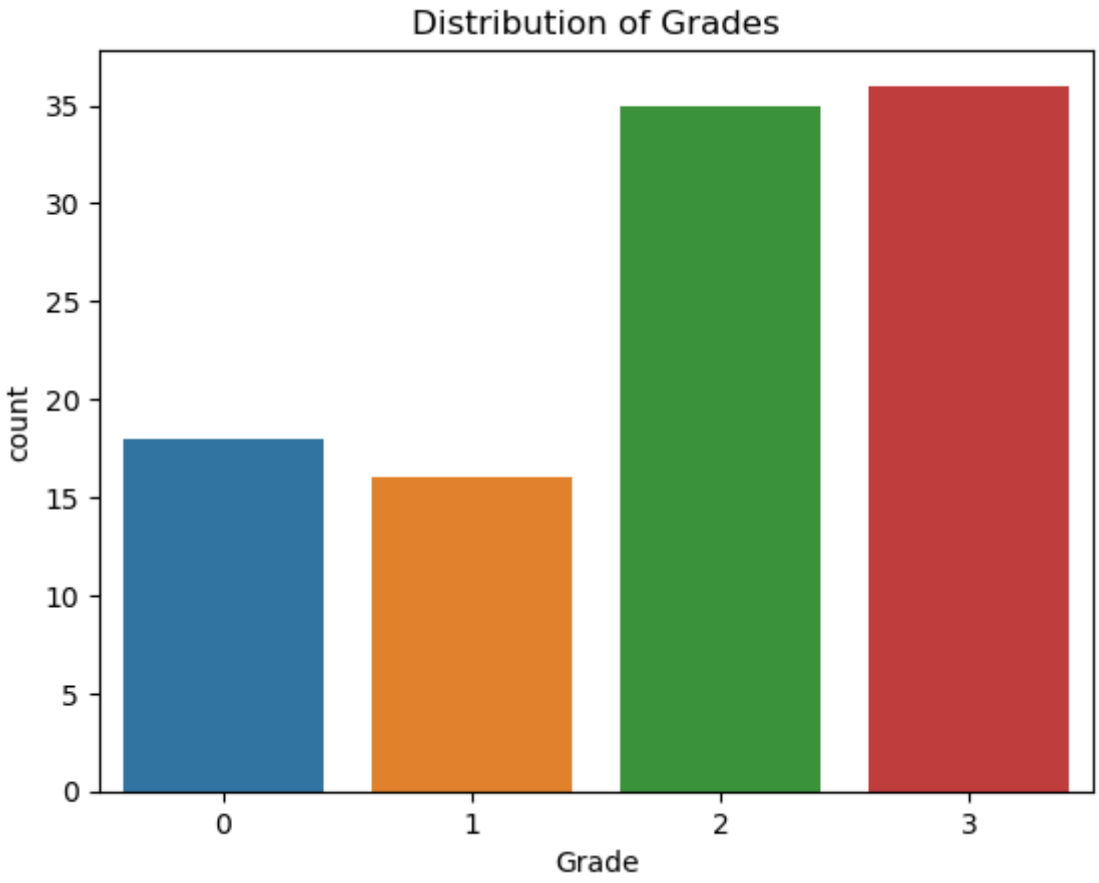
In [ ]:

In [177…
```
# Aggregate features by student
features = merged_df.groupby('StudentId').agg({
    'TotalActions': 'sum',
    'Time': 'mean',  # You might want to engineer more features based on 'Time'
    'Type_encoded': 'mean',  # Mean encoding for categorical features
    'Action_encoded': 'mean',
    'Grade': 'first'  # Assuming 'Grade' is a constant for each student
}).reset_index()
```

In [178…
```
# Check the column names in the features DataFrame
print(features.columns)
```

```
Index(['StudentId', 'TotalActions', 'Time', 'Type_encoded', 'Action_encoded',
       'Grade'],
      dtype='object')
```

In [179…
```
# Check the distribution of classes
sns.countplot(x='Grade', data=features)
plt.title('Distribution of Grades')
plt.show()
```



In [180…
```
# Split the dataset into training and test sets
X = features.drop(['StudentId', 'Grade'], axis=1)
y = features['Grade']
```

In [181…
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [234…
```
# Choose a machine learning model (Gradient Boosting)
model = LogisticRegression(max_iter=1000, random_state=42)
```

In [235…
```
# Train the model
model.fit(X_train, y_train)
```

Out[235]:
```
LogisticRegression
LogisticRegression(max_iter=1000, random_state=42)
```

In [236…
```
# Test the model
y_pred = model.predict(X_test)
```

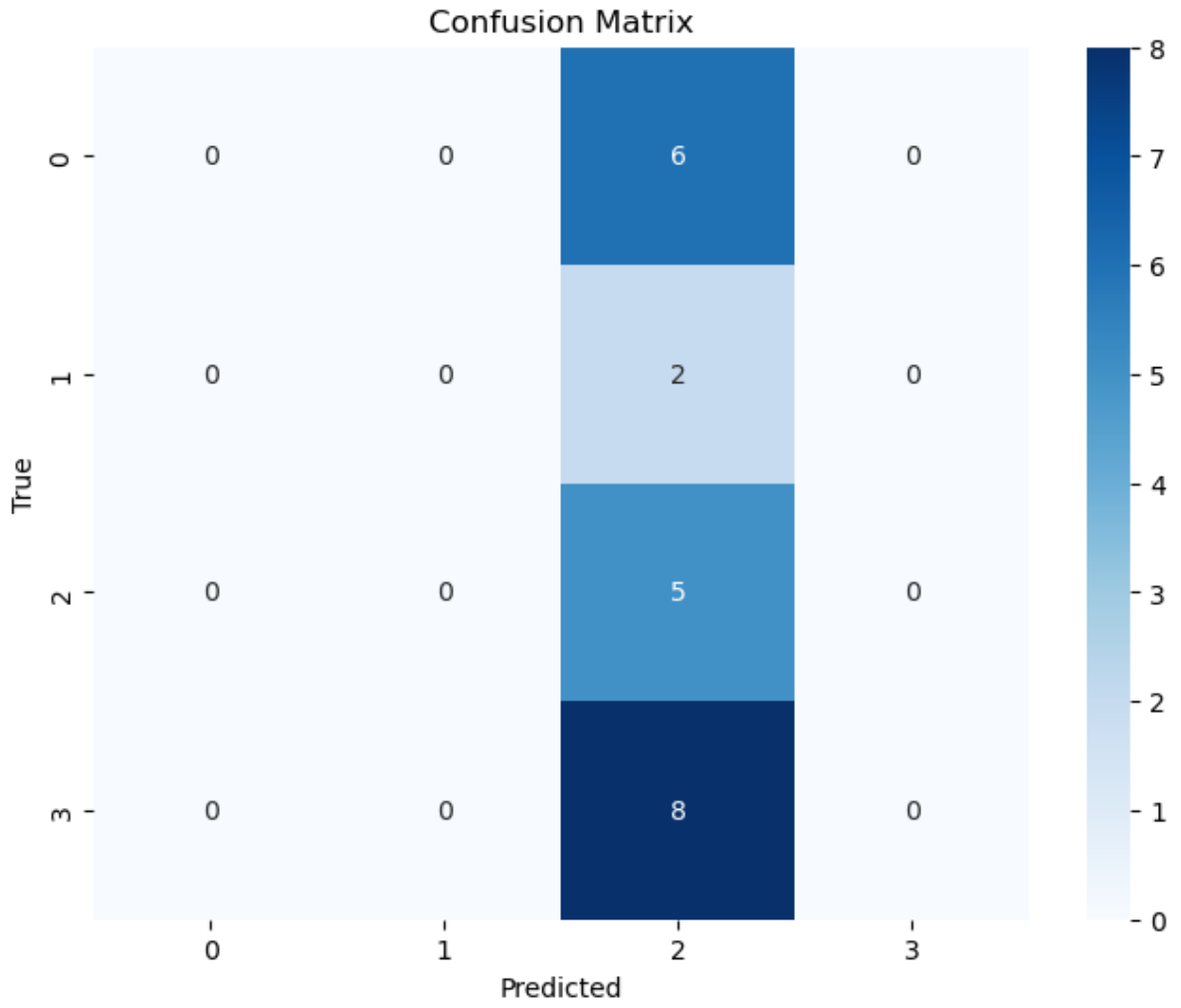
In [237…
```
# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.23809523809523808
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         6
           1       0.00      0.00      0.00         2
           2       0.24      1.00      0.38         5
           3       0.00      0.00      0.00         8

    accuracy                           0.24        21
   macro avg       0.06      0.25      0.10        21
weighted avg       0.06      0.24      0.09        21
```

```
C:\Users\junai\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:146
9: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\junai\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:146
9: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\junai\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:146
9: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to control
this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [238…

```
# Visualize confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=model.classes_, ytic
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



In [239…
```
# Print the confusion matrix
print(f"\n{name} Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
SVM Confusion Matrix:
[[0 0 6 0]
 [0 0 2 0]
 [0 0 5 0]
 [0 0 8 0]]
```

In [261…
```
# Create a DataFrame with 'StudentId' from the unseen dataset
unseen_features = pd.DataFrame({'StudentId': unseen_df['StudentId']})
```

In [262…
```
# Extract features for the 'StudentId' values in the unseen dataset
unseen_features = unseen_features.merge(features, on='StudentId', how='left')
```

In [263…
```
# If a student in the unseen dataset wasn't in the training data, assign them defau
unseen_features = unseen_features.fillna(0)  # You may need to adjust this based or
```

In [264…
```
# Make predictions on the unseen dataset
unseen_predictions = model.predict(unseen_features.drop(['StudentId', 'Grade'], axi
```

In [265…
```
# Create a DataFrame with 'StudentId' and predicted grades
unseen_result_df = pd.DataFrame({'StudentId': unseen_df['StudentId'], 'PredictedGra
```

In [266…
```
# Reverse the mapping to get original grade categories
unseen_result_df['PredictedGrade'] = unseen_result_df['PredictedGrade'].map({v: k f
```

In [267…
```
# Print the result
print(unseen_result_df)
```

```
  StudentId PredictedGrade
0      aca3            3rd
1      4f2c            3rd
2      295e            3rd
3      d1d7            3rd
4      6cd6            3rd
5      c0a8            3rd
6      2e3f            3rd
7      cad7            3rd
8      ade7            3rd
9      05cf            3rd
```

In [206…
```
num_features = ['StudentId', 'Grade']
cat_features = ['Action', 'Type']
```

In [207…
```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
# Numerical pipeline
num_pipeline = Pipeline([
 ('imputer', SimpleImputer(strategy='median')),
 ('scaler', StandardScaler())
])
# Categorical pipeline
cat_pipeline = Pipeline([
 ('imputer', SimpleImputer(strategy='most_frequent')),
 ('encoder', OneHotEncoder())
])

# Full pipeline
full_pipeline = ColumnTransformer([
 ('num', num_pipeline, num_features),
 ('cat', cat_pipeline, cat_features)
])
```

In [226…
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
# Prepare multiple classifiers for comparison
classifiers = [
 ('Random Forest', RandomForestClassifier(n_estimators=100)),
 ('Logistic Regression', LogisticRegression()),
 ('SVM', SVC())
]
```

In [227…
```
#Initialize an empty dictionary to store the results
results = {}
```

In [228…
```
# Loop through each classifier to perform k-fold cross-validation
for name, clf in classifiers:
 # Create a pipeline with the classifier
 full_pipeline_with_classifier = Pipeline([
 ('preprocessing', full_pipeline),
 ('classifier', clf)
 ])
```

In [229…
```
# Cross-validation
cross_val_scores = cross_val_score(model, X, y,cv=5)
```

In [ ]:

In [230…
```
# Calculate the mean score across all folds
mean_score = cross_val_scores.mean()
```

In [231…
```
#Store the results
results[name] = mean_score
```

In [232…
```
# Print the results
print(f"Cross-Validation Results: {results}")
```

```
Cross-Validation Results: {'SVM': 0.3333333333333333}
```

In [ ]:

In [ ]:

In [ ]:

In [215…
```
# Assuming `model` is your trained model
baseline_prediction = model.classes_[np.argmax(np.bincount(y))]
unseen_df['PredictedGrade'] = baseline_prediction
print(unseen_df[['StudentId', 'PredictedGrade']])
```

```
  StudentId  PredictedGrade
0      aca3               3
1      4f2c               3
2      295e               3
3      d1d7               3
4      6cd6               3
5      c0a8               3
6      2e3f               3
7      cad7               3
8      ade7               3
9      05cf               3
```

In [ ]:

In [ ]: