# Step 5: Performance Evaluation

- Load Test Data
    - Load the original dataset
    - Apply the same preprocessing pipeline used in training.ipynb
    - Extract the 20% test set (the same as used during training)
    - Load Trained Models & Pipeline
- Load the saved classification model
- Load the saved regression model
- Load the preprocessing pipeline

**Evaluate Classification Model**

- Generate predictions on the test set. Compute:
    - Accuracy
    - Precision, Recall, F1-score
    - Confusion Matrix
    - ROC-AUC Curve

**Evaluate Regression Model**

- Generate predictions on the test set. Compute:
    - $R^2$ (Coefficient of Determination)
    - MAE (Mean Absolute Error)
    - MSE (Mean Squared Error)
    - RMSE (Root Mean Squared Error)

`Task 6. After completing all steps above, provide the following:`

- Compare models and justify which one is better for each task.
- At least one visualizations per classification tasks (e.g., confusion matrix, ROC curve, precision-recall curves).

In [1]:
```
import pandas as pd
import joblib
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_e
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from utility import DataCleaner

df = pd.read_csv('Spotify_Song_Attributes.csv')
```

In [2]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10080 entries, 0 to 10079
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   trackName         10080 non-null  object 
 1   artistName        10080 non-null  object 
 2   msPlayed          10080 non-null  int64  
 3   genre             8580 non-null   object 
 4   danceability      9530 non-null   float64
 5   energy            9530 non-null   float64
 6   key               9530 non-null   float64
 7   loudness          9530 non-null   float64
 8   mode              9530 non-null   float64
 9   speechiness       9530 non-null   float64
 10  acousticness      9530 non-null   float64
 11  instrumentalness  9530 non-null   float64
 12  liveness          9530 non-null   float64
 13  valence           9530 non-null   float64
 14  tempo             9530 non-null   float64
 15  type              9530 non-null   object 
 16  id                9530 non-null   object 
 17  uri               9530 non-null   object 
 18  track_href        9530 non-null   object 
 19  analysis_url      9530 non-null   object 
 20  duration_ms       9530 non-null   float64
 21  time_signature    9530 non-null   float64
dtypes: float64(13), int64(1), object(8)
memory usage: 1.7+ MB
```

In [3]:
```
data_cleaning_pipeline=joblib.load('cleaning_pipeline.pkl')
```

In [4]:
```
df = data_cleaning_pipeline.transform(df)
```

```
/Users/chaotzuchieh/Documents/GitHub/project-1-TzuChieh_Chao/venv/lib/pyth
on3.12/site-packages/sklearn/pipeline.py:62: FutureWarning: This Pipeline
instance is not fitted yet. Call 'fit' with appropriate arguments before u
sing other methods such as transform, predict, etc. This will raise an err
or in 1.8 instead of the current warning.
  warnings.warn(
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4261 entries, 1 to 5039
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   trackName         4261 non-null   object 
 1   artistName        4261 non-null   object 
 2   msPlayed          4261 non-null   int64  
 3   genre             4261 non-null   object 
 4   danceability      4261 non-null   float64
 5   energy            4261 non-null   float64
 6   key               4261 non-null   float64
 7   loudness          4261 non-null   float64
 8   mode              4261 non-null   float64
 9   speechiness       4261 non-null   float64
 10  acousticness      4261 non-null   float64
 11  instrumentalness  4261 non-null   float64
 12  liveness          4261 non-null   float64
 13  valence           4261 non-null   float64
 14  tempo             4261 non-null   float64
 15  type              4261 non-null   object 
 16  id                4261 non-null   object 
 17  uri               4261 non-null   object 
 18  track_href        4261 non-null   object 
 19  analysis_url      4261 non-null   object 
 20  duration_ms       4261 non-null   float64
 21  time_signature    4261 non-null   float64
dtypes: float64(13), int64(1), object(8)
memory usage: 765.6+ KB
```

In [6]:
```python
df.head()
```

Out[6]:

| | trackName | artistName | msPlayed | genre | danceability | energy | k |
|---|---|---|---|---|---|---|---|
| 1 | "In The Hall Of The Mountain King" from Peer G... | London Symphony Orchestra | 1806234 | british orchestra | 0.475 | 0.130 | 7 |
| 2 | #BrooklynBloodPop! | SyKo | 145610 | glitchcore | 0.691 | 0.814 | 1 |
| 3 | $10 | Good Morning | 25058 | experimental pop | 0.624 | 0.596 | 4 |
| 4 | (I Just) Died In Your Arms | Cutting Crew | 5504949 | album rock | 0.625 | 0.726 | 1 |
| 5 | (L)only Child | salem ilese | 2237969 | alt z | 0.645 | 0.611 | 8 |

5 rows × 22 columns

In [7]:
```python
classification_preprocessing_pipeline= joblib.load('cls_preprocessor.pkl'
regression_preprocessing_pipeline= joblib.load('reg_preprocessor.pkl')
```

In [8]:
```python
logreg_model = joblib.load('best_logreg_model_danceability.pkl')
rf_classifer = joblib.load('best_rf_model_danceability.pkl')
```

```
linear_regression_model = joblib.load('linear_regression_model.pkl')
ridge_linear_model = joblib.load('ridge_model.pkl')
lasso_linear_model = joblib.load('lasso_model.pkl')
GridSearch_decision_tree = joblib.load('best_GridSearch_decision_tree_mod
RandomizedSearch_decision_tree = joblib.load('best_RandomizedSearch_decis
```

# Classification

In [9]:
```
cls_x_test = pd.read_csv('df_cls_x_test.csv')
cls_y_test = pd.read_csv('df_cls_y_test.csv')
```

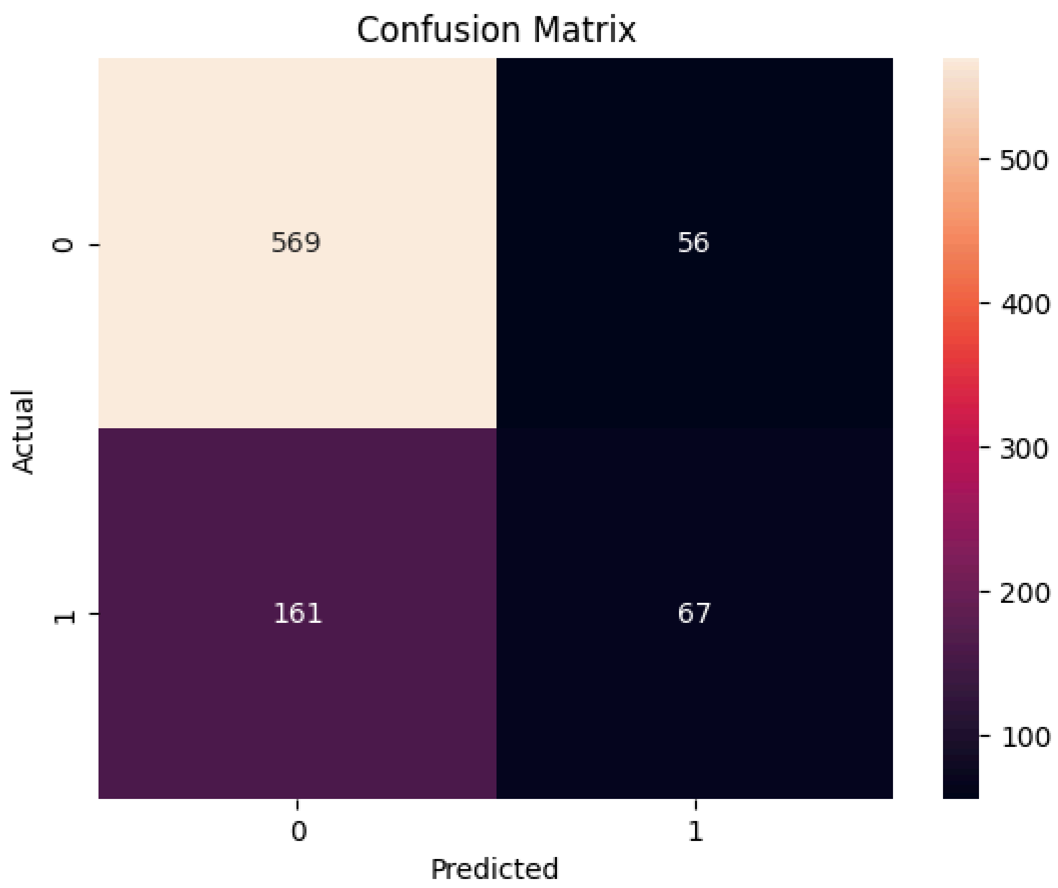
## Logistic regression

In [10]:
```
y1_pred_log_reg = logreg_model.predict(cls_x_test)
y1_pred_log_reg_proba = logreg_model.predict_proba(cls_x_test)


print('Logistic Regression')
print('Accuracy:', accuracy_score(cls_y_test, y1_pred_log_reg))
print('Precision:', precision_recall_fscore_support(cls_y_test, y1_pred_l
print('Recall:', precision_recall_fscore_support(cls_y_test, y1_pred_log_
print('F1 Score:', precision_recall_fscore_support(cls_y_test, y1_pred_lo
print('Confusion Matrix:', confusion_matrix(cls_y_test, y1_pred_log_reg))


cm = confusion_matrix(cls_y_test, y1_pred_log_reg)
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Logistic Regression
Accuracy: 0.7456037514654161
Precision: 0.5447154471544715
Recall: 0.29385964912280704
F1 Score: 0.3817663817663818
Confusion Matrix: [[569  56]
 [161  67]]
```



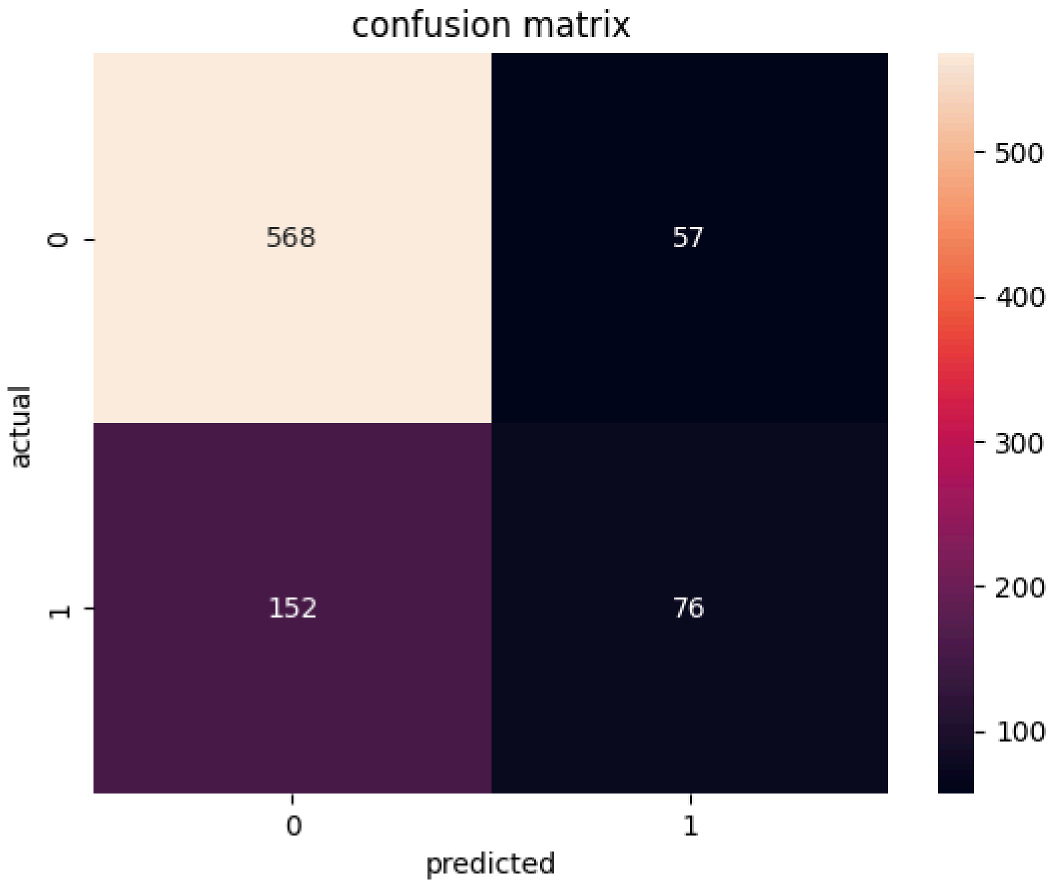

## Random Forest Classifier

In [11]:
```
y1_pred_rand_for=rf_classifer.predict(cls_x_test)
y1_pred_rand_for_proba=rf_classifer.predict_proba(cls_x_test)[:,1]

print('Random Forest')
print('Accuracy:',accuracy_score(cls_y_test,y1_pred_rand_for))
print('Precision:',precision_recall_fscore_support(cls_y_test,y1_pred_ran
print('Recall:',precision_recall_fscore_support(cls_y_test,y1_pred_rand_f
print('F1 Score:',precision_recall_fscore_support(cls_y_test,y1_pred_rand
print('Confusion Matrix:',confusion_matrix(cls_y_test,y1_pred_rand_for))
cm=confusion_matrix(cls_y_test,y1_pred_rand_for)
sns.heatmap(cm,annot=True,fmt='d')
plt.xlabel('predicted')
plt.ylabel('actual')
plt.title('confusion matrix')
```

```
Random Forest
Accuracy: 0.7549824150058617
Precision: 0.5714285714285714
Recall: 0.3333333333333333
F1 Score: 0.42105263157894735
Confusion Matrix: [[568  57]
 [152  76]]
```

Out[11]: `Text(0.5, 1.0, 'confusion matrix')`




---

# Regression

In [12]:
```
reg_x_test = pd.read_csv('df_reg_x_test.csv')
reg_y_test = pd.read_csv('df_reg_y_test.csv')
```

## Linear Regression

In [13]:
```
y2_pred_lig_reg=linear_regression_model.predict(reg_x_test)
print('Linear Regression')
print('R2 Score:',r2_score(reg_y_test,y2_pred_lig_reg))
print('Mean Absolute Error:',mean_absolute_error(reg_y_test,y2_pred_lig_r
print('Mean Squared Error:',mean_squared_error(reg_y_test,y2_pred_lig_reg
print('Root Mean Squared Error:',np.sqrt(mean_squared_error(reg_y_test,y2
```

```
Linear Regression
R2 Score: 0.3602056987416682
Mean Absolute Error: 0.15229598307314432
Mean Squared Error: 0.03626068026938245
Root Mean Squared Error: 0.190422373342479
```

## Linear Regression(Ridge)

In [14]:
```
y2_pred_lig_reg=ridge_linear_model.predict(reg_x_test)
print('Linear Regression(Ridge)')
print('R2 Score:',r2_score(reg_y_test,y2_pred_lig_reg))
```

```
print('Mean Absolute Error:',mean_absolute_error(reg_y_test,y2_pred_lig_r
print('Mean Squared Error:',mean_squared_error(reg_y_test,y2_pred_lig_reg
print('Root Mean Squared Error:',np.sqrt(mean_squared_error(reg_y_test,y2
```

```
Linear Regression(Ridge)
R2 Score: 0.35670687196360706
Mean Absolute Error: 0.15419020882149784
Mean Squared Error: 0.03645897812678397
Root Mean Squared Error: 0.19094234241462518
```

# Linear Regression(Lasso)

In [15]:
```
y2_pred_lig_reg=lasso_linear_model.predict(reg_x_test)
print('Linear Regression(Lasso)')
print('R2 Score:',r2_score(reg_y_test,y2_pred_lig_reg))
print('Mean Absolute Error:',mean_absolute_error(reg_y_test,y2_pred_lig_r
print('Mean Squared Error:',mean_squared_error(reg_y_test,y2_pred_lig_reg
print('Root Mean Squared Error:',np.sqrt(mean_squared_error(reg_y_test,y2
```

```
Linear Regression(Lasso)
R2 Score: 0.35649835874578195
Mean Absolute Error: 0.15423930116020515
Mean Squared Error: 0.03647079572364069
Root Mean Squared Error: 0.19097328536641112
```

# DecisionTreeRegressor(GridSearchCV)

In [16]:
```
y2_pred_decision_tree=GridSearch_decision_tree.predict(reg_x_test)
print('Decision Tree')
print('R2 Score:',r2_score(reg_y_test,y2_pred_decision_tree))
print('Mean Absolute Error:',mean_absolute_error(reg_y_test,y2_pred_decis
print('Mean Squared Error:',mean_squared_error(reg_y_test,y2_pred_decisio
print('Root Mean Squared Error:',np.sqrt(mean_squared_error(reg_y_test,y2
```

```
Decision Tree
R2 Score: 0.2950428441908042
Mean Absolute Error: 0.16042207583291546
Mean Squared Error: 0.03995381950126051
Root Mean Squared Error: 0.19988451541142577
```

# DecisionTreeRegressor(RandomizedSearchCV)

In [17]:
```
y2_pred_decision_tree=RandomizedSearch_decision_tree.predict(reg_x_test)
print('Decision Tree')
print('R2 Score:',r2_score(reg_y_test,y2_pred_decision_tree))
print('Mean Absolute Error:',mean_absolute_error(reg_y_test,y2_pred_decis
print('Mean Squared Error:',mean_squared_error(reg_y_test,y2_pred_decisio
print('Root Mean Squared Error:',np.sqrt(mean_squared_error(reg_y_test,y2
```

```
Decision Tree
R2 Score: 0.2509975738287197
Mean Absolute Error: 0.16550395583307478
Mean Squared Error: 0.04245010848482428
Root Mean Squared Error: 0.20603424104945342
```

# Analysis of Models

## Classification

**The results indicated that the Random Forest classifier outperformed the Logistic Regression model.**

## Regression

**I implemented several machine learning regression models aimed at predicting valence to determine which model exhibited superior performance. However, it is noteworthy that none of the models yielded satisfactory results, with linear regression emerging as the most effective option.**