In [31]:

```
# The goal of this project is if we can properly predict a high popularity s
# our target variable. We will be trying two different classifier models Log
# and see which one is best.
from google.cloud import storage
import io
from io import StringIO, BytesIO
import pandas as pd
import gzip
import numpy as np
import gcsfs
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, classification_report
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import f1_score, precision_recall_fscore_support
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.inspection import permutation_importance
from sklearn.inspection import PartialDependenceDisplay
import matplotlib.pyplot as plt
fs = gcsfs.GCSFileSystem()
print(fs.ls('edosa_spotify_project/cleaned'))
```

```
['edosa_spotify_project/cleaned/null', 'edosa_spotify_project/cleaned/spotif
y_clean.csv']
```

In [2]:

```
gcfs_path = 'gs://edosa_spotify_project/cleaned/spotify_clean.csv'
spotify_df = pd.read_csv(gcfs_path)
```

In [3]:

```
spotify_df.head()
```

Out[3]:

	song_name	artists	daily_rank	daily_movement	weekly_movement	country	sr
0	Die With A Smile	Lady Gaga, Bruno Mars	1	49	49	Unknown	
1	APT.	ROSÉ, Bruno Mars	2	1	0	Unknown	
2	luther (with sza)	Kendrick Lamar, SZA	3	-1	0	Unknown	
3	BIRDS OF A FEATHER	Billie Eilish	4	0	1	Unknown	
4	Abracadabra	Lady Gaga	5	45	45	Unknown	

5 rows × 24 columns

In [4]:
```
print(spotify_df['is_explicit'].value_counts(dropna=False))
```

```
is_explicit
False    1206510
True      589544
Name: count, dtype: int64
```

In [5]:
```
# lets covert the booleans into integers
spotify_df['is_explicit_flag'] = spotify_df['is_explicit'].astype(int)
print(spotify_df['is_explicit_flag'].value_counts(dropna=False))
spotify_df.head(5)
```

```
is_explicit_flag
0    1206510
1     589544
Name: count, dtype: int64
```

Out[5]:

	song_name	artists	daily_rank	daily_movement	weekly_movement	country	sr
0	Die With A Smile	Lady Gaga, Bruno Mars	1	49	49	Unknown	
1	APT.	ROSÉ, Bruno Mars	2	1	0	Unknown	
2	luther (with sza)	Kendrick Lamar, SZA	3	-1	0	Unknown	
3	BIRDS OF A FEATHER	Billie Eilish	4	0	1	Unknown	
4	Abracadabra	Lady Gaga	5	45	45	Unknown	

5 rows × 25 columns

In [6]:
```
spotify_df.drop(columns =['is_explicit'], inplace = True)
```

In [7]:
```
spotify_df.rename(columns={'is_explicit_flag': 'is_explicit'}, inplace=True)
spotify_df.head(5)
```

Out[7]:

	song_name	artists	daily_rank	daily_movement	weekly_movement	country	sr
0	Die With A Smile	Lady Gaga, Bruno Mars	1	49	49	Unknown	
1	APT.	ROSÉ, Bruno Mars	2	1	0	Unknown	
2	luther (with sza)	Kendrick Lamar, SZA	3	-1	0	Unknown	
3	BIRDS OF A FEATHER	Billie Eilish	4	0	1	Unknown	
4	Abracadabra	Lady Gaga	5	45	45	Unknown	

5 rows × 24 columns

In [ ]:

In [8]:

```python
#lets drop some more irrelevant shit
spotify_df.drop(columns=['daily_movement','weekly_movement'], inplace=True)
```

In [9]:

```python
spotify_df.drop(columns = ['album_release_date'], inplace = True)
```

In [10]:

```python
spotify_df['popularity'].value_counts()
```

Out[10]:

```
popularity
87     64326
88     62992
86     59436
89     58845
84     55362
       ...  
5        119
10       119
9        117
2         99
4         95
Name: count, Length: 101, dtype: int64
```

In [11]:

```python
# lets drop alot of stuff
spotify_df.drop(columns=['daily_rank','song_name','artists','country','snaps
```

In [12]:

```python
#Lets check the popularity score range
print("Min popularity:", spotify_df['popularity'].min())
print("Max popularity:", spotify_df['popularity'].max())
print(spotify_df['popularity'].describe())
```

```
Min popularity: 0
Max popularity: 100
count    1.796054e+06
mean     7.591953e+01
std      1.581922e+01
min      0.000000e+00
25%      6.500000e+01
50%      8.000000e+01
75%      8.800000e+01
max      1.000000e+02
Name: popularity, dtype: float64
```

In [13]:

```
# Now lets bin the popularity score into a binary format
threshold = 61

spotify_df['popularity'] = (spotify_df['popularity'] >= threshold).astype(ir
```

In [14]:

```
spotify_df
```

Out[14]:

	popularity	duration_ms	danceability	energy	key	loudness	mode	speec
0	0	251667	0.519	0.601	6	-7.727	0	
1	1	169917	0.777	0.783	0	-4.477	0	
2	1	177598	0.707	0.575	2	-7.546	1	
3	1	210373	0.747	0.507	2	-10.171	1	
4	0	223398	0.679	0.906	10	-3.443	0	
...	...	...	...	...	...	...	...	
1796049	1	310490	0.483	0.408	5	-9.243	0	
1796050	1	173253	0.773	0.635	10	-5.060	1	
1796051	1	184791	0.573	0.422	10	-7.621	0	
1796052	1	179560	0.633	0.454	9	-8.016	0	
1796053	1	132359	0.638	0.717	8	-5.804	1	

1796054 rows × 15 columns

In [15]:

```
y= spotify_df['popularity']
X = spotify_df.drop(columns=['popularity'])
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [16]:

```
#logisistic regression
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)
```

```
clf = LogisticRegression(max_iter=1000)
clf.fit(X_train, y_train)
y_proba = clf.predict_proba(X_test)[:, 1]
y_pred  = clf.predict(X_test)
print("Test AUC:", roc_auc_score(y_test, y_proba))
print("\nClassification Report:\n", classification_report(y_test, y_pred, di
```

```
Test AUC: 0.6521549017311478

Classification Report:
               precision    recall  f1-score   support

           0      0.302     0.005     0.011     63826
           1      0.823     0.997     0.902    295385

    accuracy                          0.821    359211
   macro avg      0.562     0.501     0.456    359211
weighted avg      0.730     0.821     0.743    359211
```

In [17]:

```
#balanced logistic regression
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)


clf = LogisticRegression(
    class_weight='balanced',
    max_iter=1000,
    solver='lbfgs'
)
clf.fit(X_train, y_train)

y_proba = clf.predict_proba(X_test)[:,1]
y_pred  = clf.predict(X_test)

print("AUC (balanced):", roc_auc_score(y_test, y_proba))
print("\nClassification Report (balanced):\n", classification_report(y_test,
```

```
AUC (balanced): 0.6545164436672691

Classification Report (balanced):
               precision    recall  f1-score   support

           0      0.259     0.614     0.365     63826
           1      0.882     0.621     0.729    295385

    accuracy                          0.620    359211
   macro avg      0.571     0.618     0.547    359211
weighted avg      0.771     0.620     0.664    359211
```

In [18]:

```
y_proba = clf.predict_proba(X_test)[:, 1]
```

```
best_thresh = 0.5
best_f1     = 0

for thresh in np.linspace(0.1, 0.9, 17):
    y_pred_thresh = (y_proba >= thresh).astype(int)
    f1 = f1_score(y_test, y_pred_thresh)
    print(f"Threshold {thresh:.2f} → F1 = {f1:.3f}")
    if f1 > best_f1:
        best_f1     = f1
        best_thresh = thresh

print(f"\nOptimal threshold for max F1: {best_thresh:.2f} (F1 = {best_f1:.3f
```

```
Threshold 0.10 → F1 = 0.902
Threshold 0.15 → F1 = 0.901
Threshold 0.20 → F1 = 0.901
Threshold 0.25 → F1 = 0.895
Threshold 0.30 → F1 = 0.886
Threshold 0.35 → F1 = 0.874
Threshold 0.40 → F1 = 0.853
Threshold 0.45 → F1 = 0.801
Threshold 0.50 → F1 = 0.729
Threshold 0.55 → F1 = 0.619
Threshold 0.60 → F1 = 0.461
Threshold 0.65 → F1 = 0.318
Threshold 0.70 → F1 = 0.193
Threshold 0.75 → F1 = 0.087
Threshold 0.80 → F1 = 0.018
Threshold 0.85 → F1 = 0.013
Threshold 0.90 → F1 = 0.001

Optimal threshold for max F1: 0.10 (F1 = 0.902)
```

In [19]:

```
#random forest
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42,
    stratify=y
)

rf_clf = RandomForestClassifier(
    n_estimators=100,
    class_weight='balanced',
    random_state=42,
    n_jobs=-1
)
rf_clf.fit(X_train, y_train)

y_pred  = rf_clf.predict(X_test)
y_proba = rf_clf.predict_proba(X_test)[:,1]

print("Test AUC:", roc_auc_score(y_test, y_proba))
print("\nClassification Report:\n", classification_report(y_test, y_pred, di

importances = pd.Series(rf_clf.feature_importances_, index=X.columns) \
                  .sort_values(ascending=False)
print("\nTop 10 Feature Importances:\n", importances.head(10))
```

```
Test AUC: 0.9822063544546652

Classification Report:
               precision    recall  f1-score   support

           0      0.756     0.926     0.832     63826
           1      0.983     0.935     0.959    295385

    accuracy                          0.934    359211
   macro avg      0.869     0.931     0.895    359211
weighted avg      0.943     0.934     0.936    359211


Top 10 Feature Importances:
 loudness            0.113056
speechiness         0.106430
duration_ms         0.096763
energy              0.095099
tempo               0.095084
danceability        0.091057
acousticness        0.087240
valence             0.085894
liveness            0.081455
instrumentalness    0.052300
dtype: float64
```

In [20]:

```
#cross validator
scores_5 = cross_val_score(
    rf_clf, X, y,
    cv=5,
    scoring='roc_auc',
    n_jobs=-1
)
print("5-fold AUC scores:", scores_5)
print("Mean 5-fold AUC:", scores_5.mean())
```

```
5-fold AUC scores: [0.87228341 0.9271378  0.89784449 0.89230724 0.91042276]
Mean 5-fold AUC: 0.8999991414733433
```

In [21]:

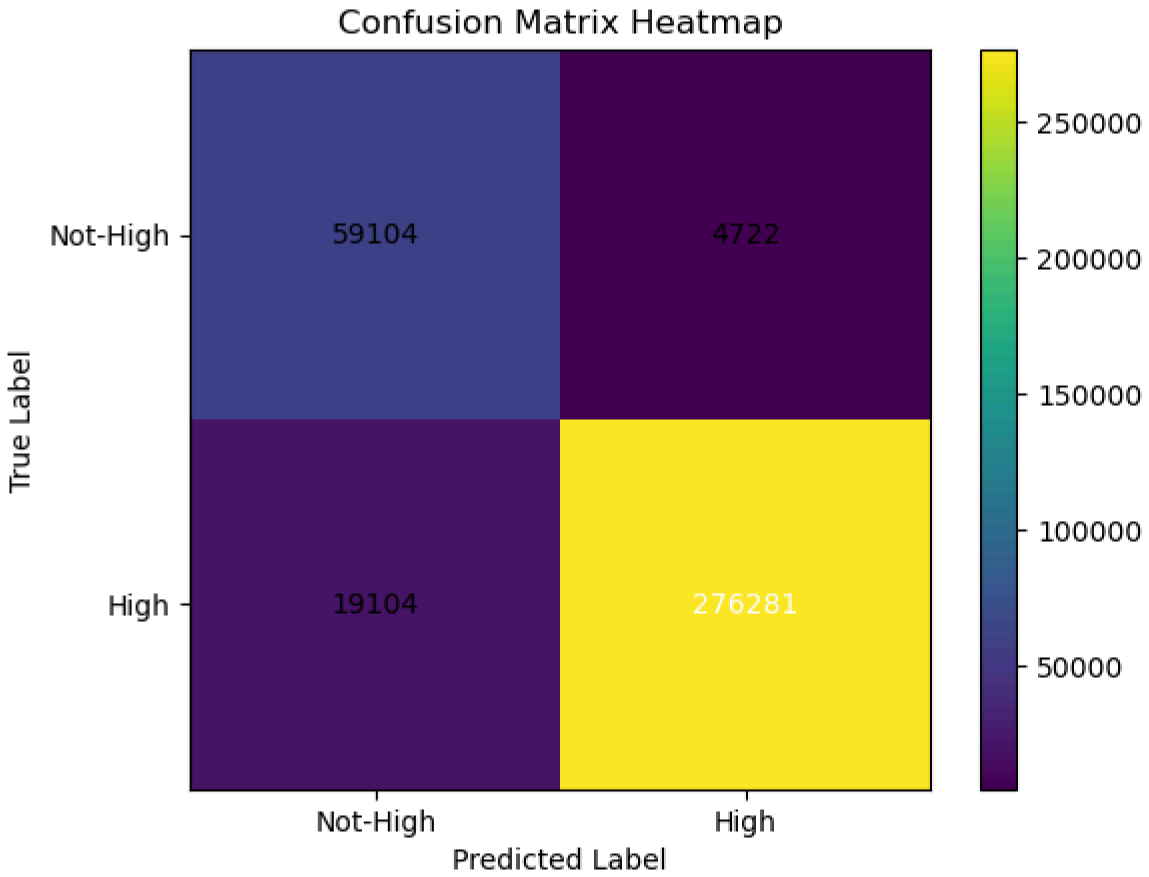
```
#confusion matrix
y_pred = rf_clf.predict(X_test)
cm = confusion_matrix(y_test, y_pred, labels=[0,1])
cm_df = pd.DataFrame(
    cm,
    index=['Actual 0 (Not-High)', 'Actual 1 (High)'],
    columns=['Pred 0', 'Pred 1']
)

print(cm_df)
```

```
                     Pred 0  Pred 1
Actual 0 (Not-High)   59104    4722
Actual 1 (High)       19104  276281
```

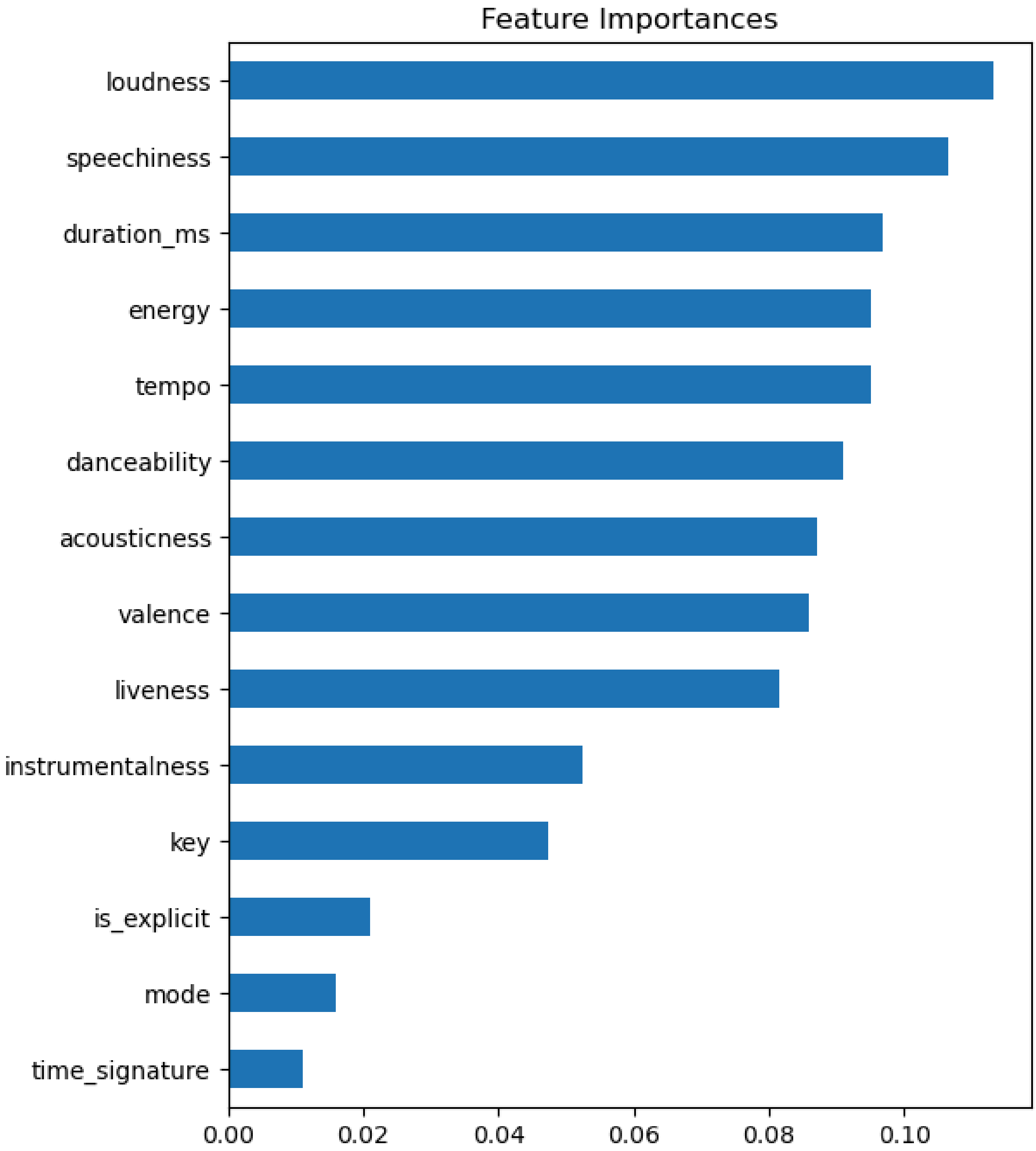
In [22]:
```
accuracy = rf_clf.score(X_test, y_test)
print("Test set accuracy:", accuracy)
```

```
Test set accuracy: 0.9336712962576313
```

In [23]:
```
#Visualizations heatmap of confusion matrix
fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest')
labels = ['Not-High', 'High']
ax.set_xticks(np.arange(len(labels)))
ax.set_yticks(np.arange(len(labels)))
ax.set_xticklabels(labels)
ax.set_yticklabels(labels)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix Heatmap')
thresh = cm.max() / 2
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        color = 'white' if cm[i, j] > thresh else 'black'
        ax.text(j, i, f"{cm[i, j]}", ha='center', va='center', color=color)

plt.colorbar(im, ax=ax)
plt.show()
```



In [24]:
```
importances.sort_values().plot.barh(figsize=(6,8))
plt.title("Feature Importances")
plt.show()
```

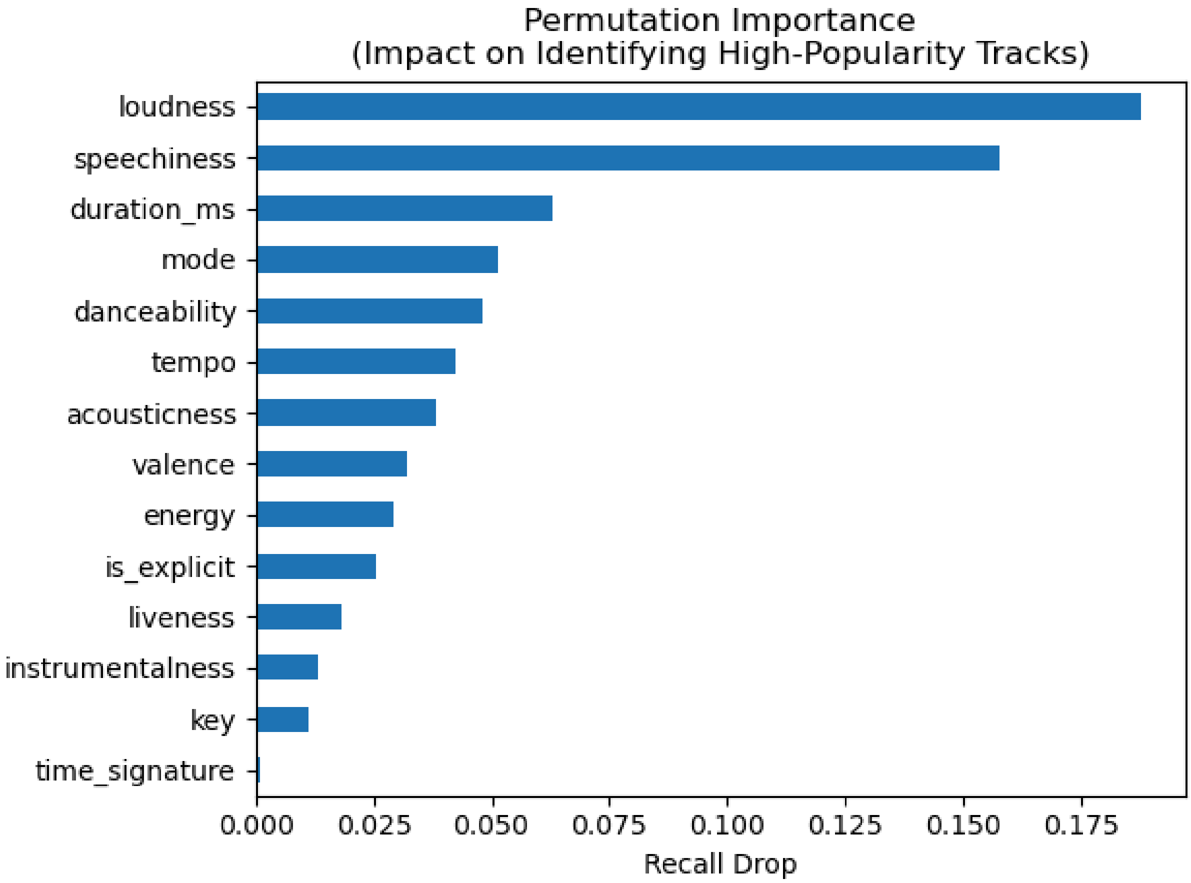


In [25]:

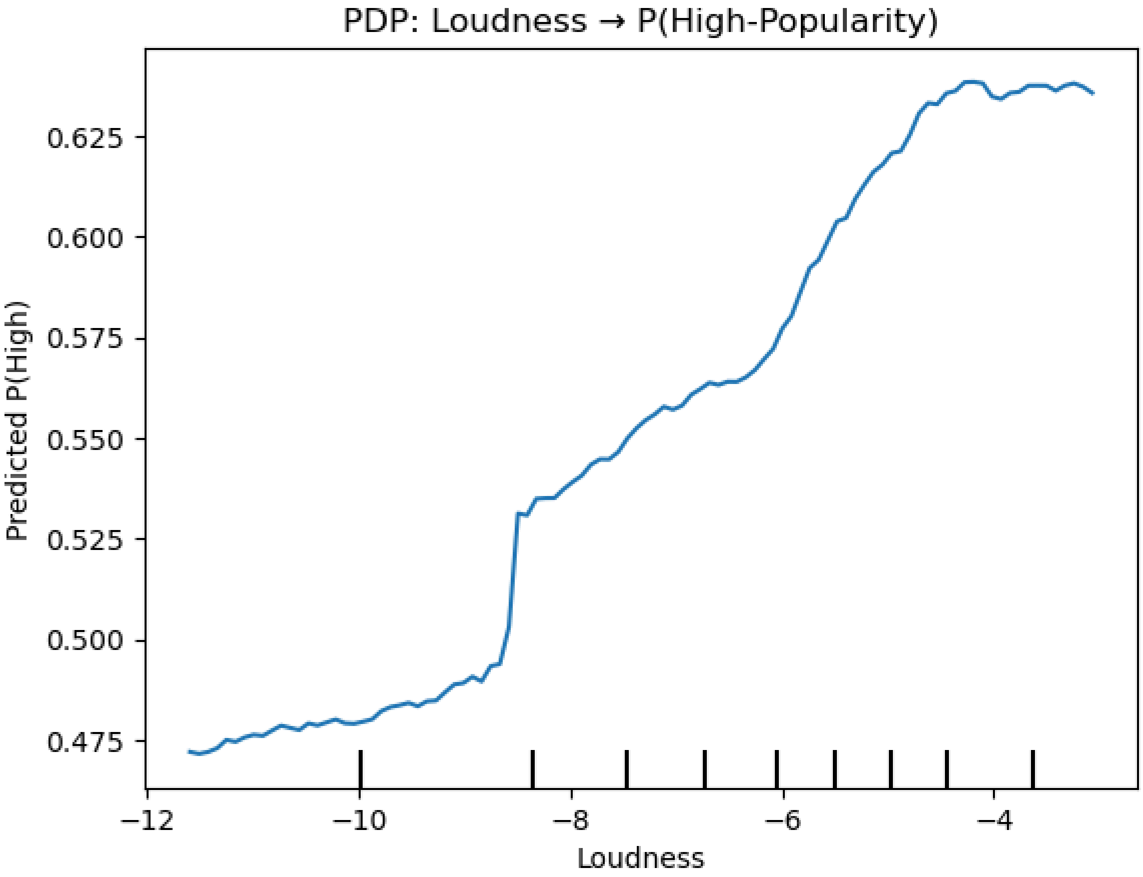
```
perm_prec = permutation_importance(
    rf_clf, X_test, y_test,
    scoring='recall',
    n_repeats=5,
    random_state=42,
    n_jobs=1
)
feat_imp = pd.Series(perm_prec.importances_mean, index=X.columns) \
            .sort_values(ascending=False)

print("Top features for identifying High-popularity (by recall drop):")
print(feat_imp.head(10))
```

```
Top features for identifying High-popularity (by recall drop):
loudness        0.187788
speechiness     0.157779
duration_ms     0.062901
mode            0.051500
danceability    0.048116
tempo           0.042381
acousticness    0.038238
valence         0.032172
energy          0.029393
is_explicit     0.025665
dtype: float64
```

In [26]:
```
fig, ax = plt.subplots()
feat_imp.sort_values().plot.barh(ax=ax)
ax.set_xlabel('Recall Drop')
ax.set_title('Permutation Importance\n(Impact on Identifying High-Popularity
plt.tight_layout()
plt.show()
```

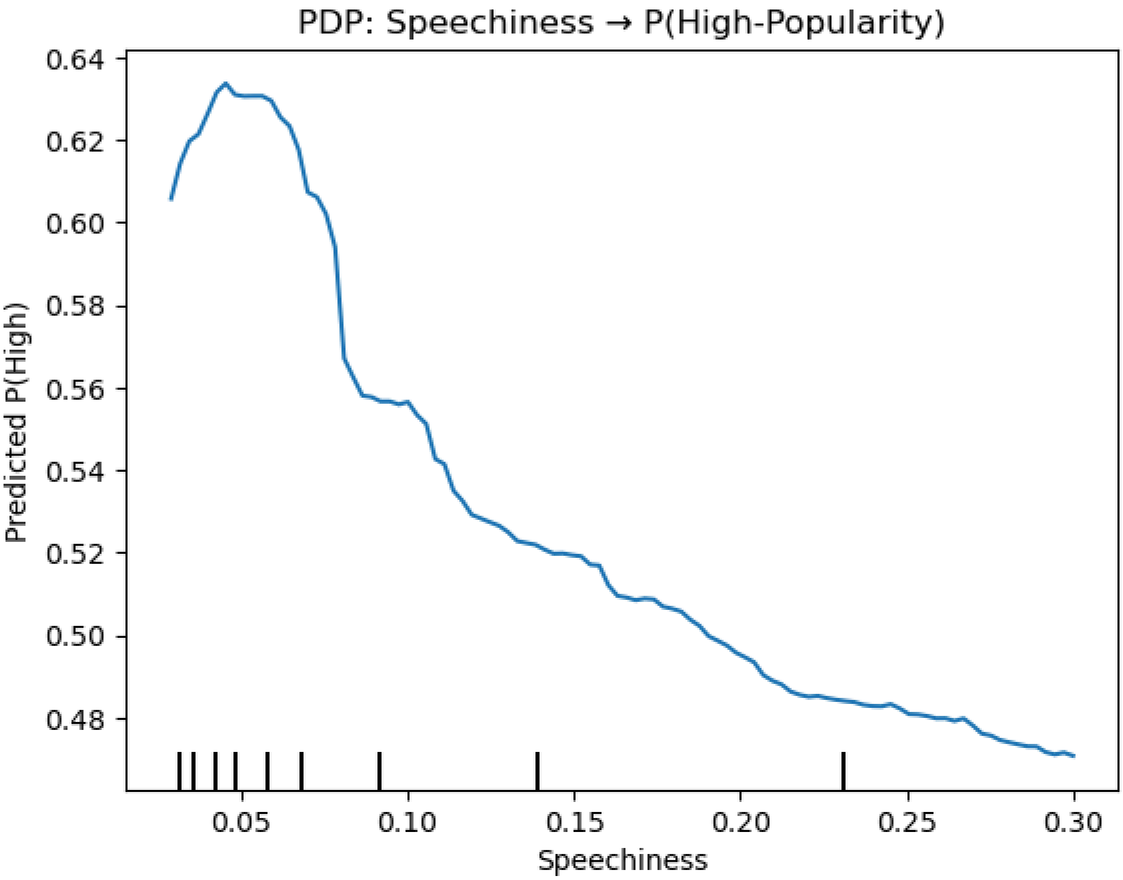


In [27]:
```
#Partial Dependence for Loudness
PartialDependenceDisplay.from_estimator(
    rf_clf,
    X_test,
    ['loudness'],
    kind='average'
)
plt.title('PDP: Loudness → P(High-Popularity)')
plt.xlabel('Loudness')
plt.ylabel('Predicted P(High)')
plt.show()
```



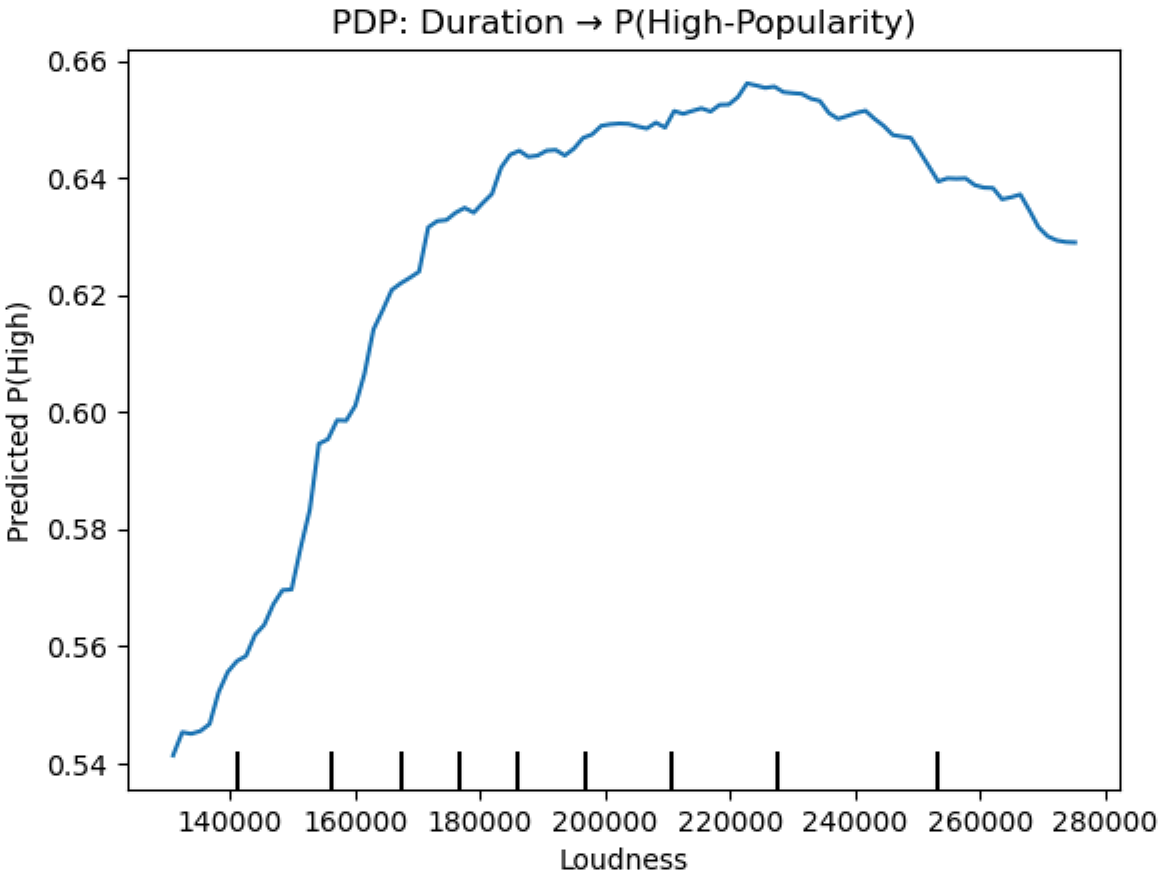
In [28]:

```
# Partial Dependence for Speechiness
PartialDependenceDisplay.from_estimator(
    rf_clf,
    X_test,
    ['speechiness'],
    kind='average'
)
plt.title('PDP: Speechiness → P(High−Popularity)')
plt.xlabel('Speechiness')
plt.ylabel('Predicted P(High)')
plt.show()
```



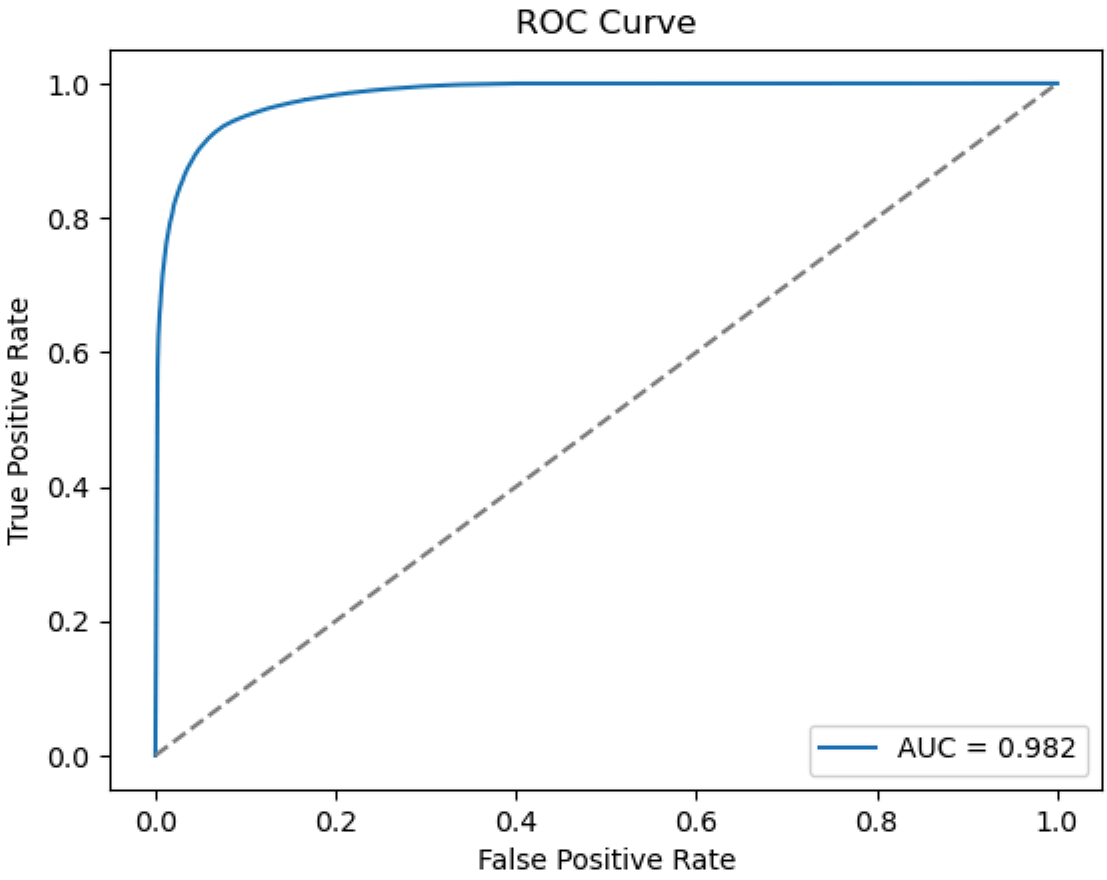
In [29]:

```
#Partial Dependence for Duration
PartialDependenceDisplay.from_estimator(
    rf_clf,
    X_test,
    ['duration_ms'],
    kind='average'
)
plt.title('PDP: Duration → P(High–Popularity)')
plt.xlabel('Loudness')
plt.ylabel('Predicted P(High)')
plt.show()
```



In [32]:

```
#Plot the AUC curve
fpr, tpr, _ = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f"AUC = {roc_auc:.3f}")
plt.plot([0,1], [0,1], "--", color="gray")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend(loc="lower right")
plt.show()
```


ROC Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.982

```
In [ ]:
```

```
In [ ]:
```