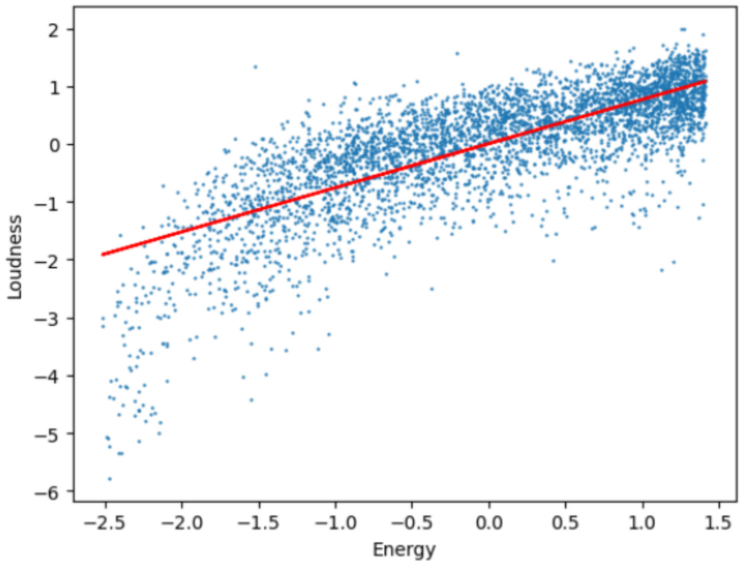
Out[ ]: 0.5928105584823308

In [ ]:
```
plt.plot(X_test, lin_with_regularization.predict(X_test), color = 'r')
plt.scatter(X_test, y_test, s = 0.5)
plt.xlabel("Energy")
plt.ylabel("Loudness")
plt.show()
```



In [ ]:
```
train_MSE = mean_squared_error(y_train, lin_with_regularization.predict(X_train))
test_MSE = mean_squared_error(y_test, lin_with_regularization.predict(X_test))
print('Train MSE: ', train_MSE)
print('Test MSE: ', test_MSE)
```

```
Train MSE:  0.40385130693463783
Test MSE:  0.43182741760107024
```

IV. Logistic Regression Analysis

How was logistic regression analysis applied in your project? What did you learn about your data set from this analysis and were you able to use this analysis for feature importance? Was regularization needed?

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.metrics import accuracy_score
from tabulate import tabulate
```

```python
rando = 1
df_logistic = df.copy()

# Split data into training and testing
X_train, X_test, y_train, y_test = train_test_split(
    df_logistic.drop(['track_genre'], axis=1),
    df_logistic.track_genre, test_size=0.2, random_state=rando)
```

We take a subset of the whole data set to be able to build the logistic regression model and visualize the results efficiently. We also split the data into training and testing sets, with a split of 80/20.

In [ ]:

```python
# Build the logistic regression model
pipeline = make_pipeline(
    StandardScaler(),
    LogisticRegression(penalty='l1', solver='saga', max_iter=10000)
)
pipeline.fit(X_train, y_train)

# Determine the model's accuracy
accuracy = pipeline.score(X_test, y_test)
print(f"Model accuracy: {accuracy:.2f}")
```

```
Model accuracy: 0.55
```

In [ ]:

```python
from sklearn.model_selection import cross_val_score
cv_score_logistic = cross_val_score(pipeline, X_test, y_test, cv=5, scoring='accuracy')
cv_score_logistic
```

Out[ ]:

```
array([0.53441802, 0.5112782 , 0.53258145, 0.52756892, 0.54761905])
```

In [ ]:

```python
np.mean(cv_score_logistic)
```

Out[ ]:

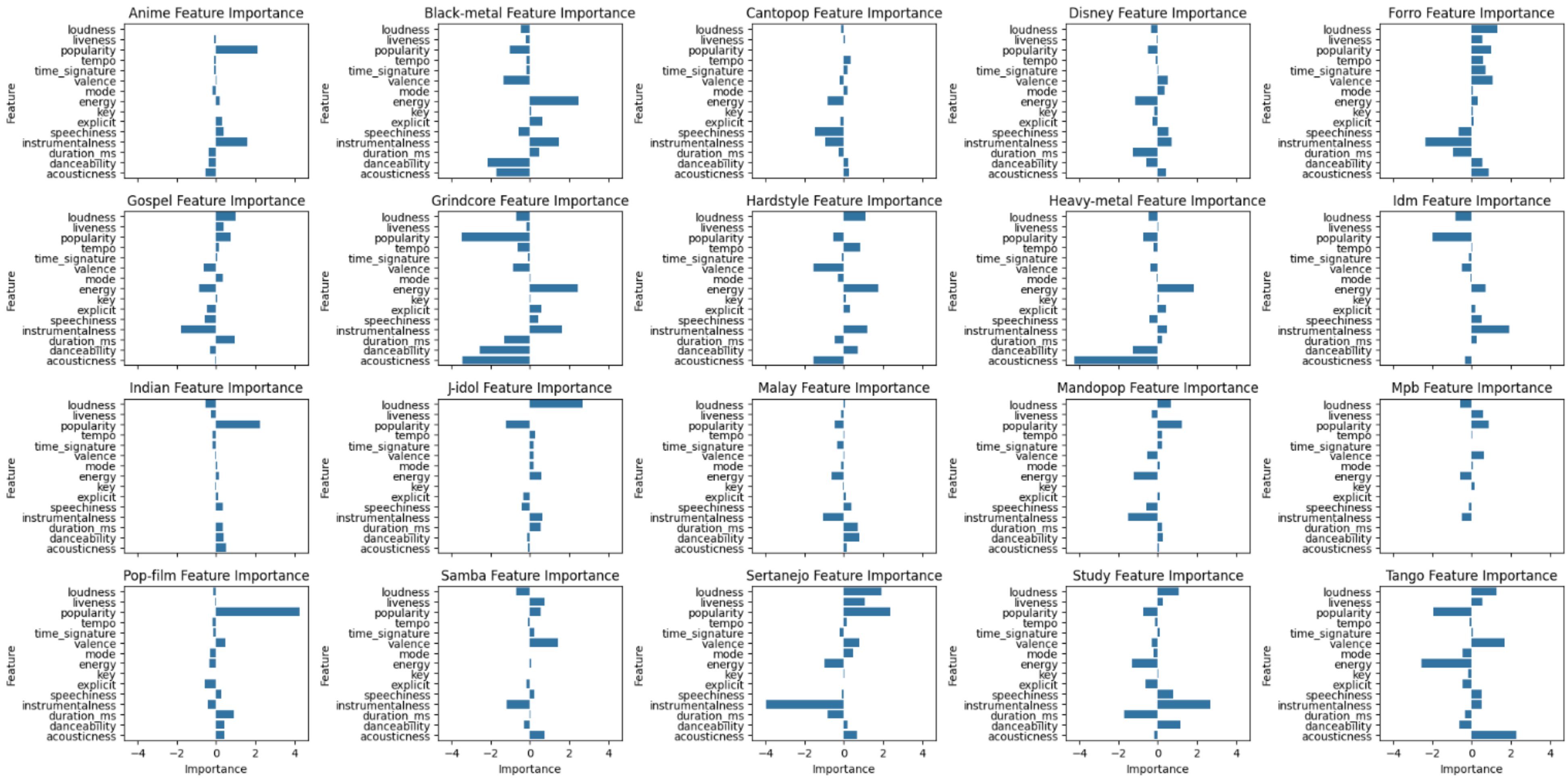
```
0.5306931283151558
```

We can get an idea of how each feature is used in the model. The following bar plots are separated by genre, and show the feature importance in each.

In [ ]:

```python
# Determine the importance of each of the features
model = pipeline.named_steps['logisticregression']

feature_importance = pd.DataFrame(model.coef_, columns=X_train.columns, index=genres)
feature_importance['Genre'] = model.classes_

feature_importance_melted = feature_importance.melt(id_vars='Genre', var_name='Feature', value_name='Importance')

average_importance = feature_importance_melted.groupby('Feature')['Importance'].mean().sort_values(ascending=False).index
feature_importance_melted['Feature'] = pd.Categorical(feature_importance_melted['Feature'], categories=average_importance, ordered=True)

# Build bar plots for each of the genres, showing the importance of each feature
fig, axes = plt.subplots(4, 5, figsize=(20, 10), sharex=True)

for i, genre in enumerate(genres):
    ax = axes[i // 5, i % 5]
    sns.barplot(x='Importance', y='Feature', data=feature_importance_melted[feature_importance_melted['Genre'] == genre], ax=ax)
    ax.set_title(f'{genre.capitalize()} Feature Importance')
```

```python
plt.tight_layout()
plt.show()
```

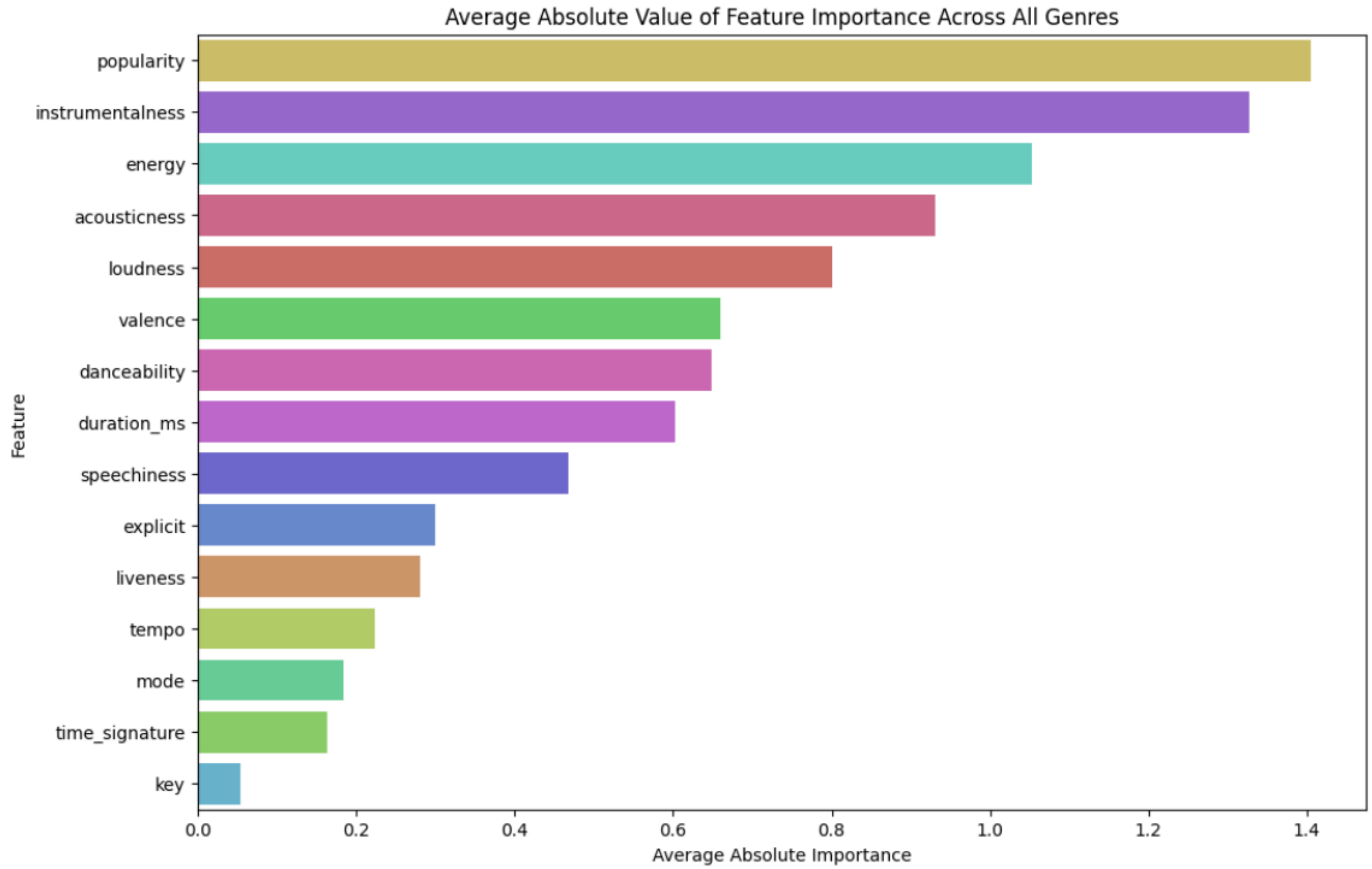


We can see that there is a clear difference in the importance of features like popularity, instrumetalness, and energy versus the less important features like mode, time signature, and key.

```python
In [ ]: # Print out the importance values for each of the features by genre
print(tabulate(feature_importance.round(4), headers='keys', tablefmt='pretty', numalign='center', stralign='center'))

# Calculate the average absolute value of importance for each feature
```

```
average_abs_importance = feature_importance_melted.groupby('Feature', observed=False)['Importance'].apply(lambda x: np.mean(np.abs(x)))

# Create a DataFrame for plotting
average_abs_importance_df = average_abs_importance.reset_index().sort_values(by='Importance', ascending=False)

# Plot the average absolute value of importance
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=average_abs_importance_df, hue='Feature', palette='hls', order=average_abs_importance_df.sort_values('Importance', ascending=False).Feature)
plt.title('Average Absolute Value of Feature Importance Across All Genres')
plt.xlabel('Average Absolute Importance')
plt.ylabel('Feature')
plt.show()
```

	popularity	duration_ms	explicit	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	time_signature	Genre
anime	2.1316	-0.3808	0.3165	-0.3596	0.1665	0.0	0.0001	-0.1936	0.391	-0.5373	1.617	-0.088	0.0087	-0.103	-0.0874	anime
black-metal	-1.0447	0.4631	0.6202	-2.1651	2.5019	0.0479	-0.4789	-0.0264	-0.5885	-1.7283	1.4762	-0.2153	-1.3642	-0.1691	-0.1915	black-metal
cantopop	0.0007	-0.2555	-0.1831	0.2412	-0.836	-0.016	-0.127	0.192	-1.4654	0.2759	-0.9579	0.0744	-0.2097	0.3544	0.1888	cantopop
disney	-0.5169	-1.2507	-0.2651	-0.564	-1.1548	-0.1894	-0.3387	0.3371	0.5554	0.438	0.7108	-0.0611	0.5155	-0.1033	0.0156	disney
forro	0.983	-0.9405	0.1038	0.5427	0.3142	0.0471	1.32	0.0628	-0.6794	0.8743	-2.3476	0.5531	1.057	0.5858	0.6936	forro
gospel	0.7435	0.9416	-0.4717	-0.2983	-0.8794	0.0542	0.9947	0.3437	-0.5776	-0.0434	-1.8046	0.3817	-0.6405	0.1565	0.0826	gospel
grindcore	-3.5029	-1.2876	0.5992	-2.577	2.4482	0.0338	-0.6968	0.0344	0.4163	-3.4497	1.6563	-0.1926	-0.8745	-0.6288	-0.1171	grindcore
hardstyle	-0.5284	-0.444	0.2981	0.7089	1.7588	0.0973	1.13	-0.3026	-0.0007	-1.5636	1.1769	0.0	-1.55	0.8337	-0.0828	hardstyle
heavy-metal	-0.7325	0.2398	0.4251	-1.2829	1.8308	0.0657	-0.459	-0.0675	-0.4323	-4.2579	0.4739	0.0159	-0.3957	-0.224	-0.0271	heavy-metal
idm	-2.0062	0.2517	0.1924	0.0	0.7154	-0.0157	-0.8198	-0.0485	0.4971	-0.3473	1.901	-0.0172	-0.5192	0.0325	-0.1375	idm
indian	2.2401	0.356	0.1056	0.3691	0.1462	-0.0624	-0.5279	0.0671	0.3553	0.5241	0.0	-0.2534	-0.0694	-0.1687	-0.1605	indian
j-idol	-1.2269	0.53	-0.3255	-0.1292	0.5764	0.0	2.6981	0.2018	-0.4145	-0.0801	0.6424	-0.0304	0.1831	0.2526	0.2016	j-idol
malay	-0.4787	0.7064	0.1012	0.8023	-0.6332	-0.0556	0.0538	-0.1519	0.3707	0.1254	-1.0551	-0.1508	0.007	0.0055	-0.3317	malay
mandopop	1.2276	0.2394	0.1098	0.2696	-1.2424	-0.0069	0.6748	0.1086	-0.5715	0.0674	-1.5242	-0.2964	-0.5245	0.2129	0.2403	mandopop
mpb	0.8549	-0.0	-0.0003	0.0	-0.5648	0.1303	-0.5722	0.0816	-0.1225	-0.0001	-0.4882	0.5911	0.6414	0.0151	0.001	mpb
pop-film	4.2766	0.9233	-0.5702	0.4121	-0.3538	-0.0144	-0.1334	-0.2809	0.2664	0.415	-0.4023	-0.0553	0.4495	-0.1597	-0.1266	pop-film
samba	0.5434	0.0075	-0.1894	-0.2865	0.0831	-0.0101	-0.7104	-0.0122	0.233	0.7557	-1.1707	0.7376	1.4322	-0.0903	0.2164	samba
sertanejo	2.3734	-0.8286	-0.0132	0.1969	-0.9746	0.0345	1.925	0.4757	-0.0877	0.6597	-3.9668	1.0533	0.7711	0.1465	-0.2121	sertanejo
study	-0.7327	-1.6919	-0.615	1.1455	-1.3264	0.0178	1.0567	-0.2225	0.7994	-0.1942	2.6698	0.2668	-0.3075	-0.1483	0.1071	study
tango	-1.9592	-0.3235	-0.4797	-0.6334	-2.5664	-0.171	1.2928	-0.4566	0.5234	2.2749	0.5191	0.5607	1.6812	-0.0825	0.0458	tango



V. KNN, Decision Trees, and Random Forest

KNN

We can fit a KNN model to the Spotify data by encoding the categorical response variable and using split data into 80/20 training and testing.