In [1]:
```
# Import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
from scipy.stats import shapiro, probplot
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_s
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.stats.stattools import durbin_watson
```

In [2]:
```
# Import dataset
df_original = pd.read_csv('spotify_data.csv')
df_original = df_original.drop(columns=['Unnamed: 0', 'artist_name', 'track_nar
df_original = df_original.dropna()
df = df_original.sample(n=1000, random_state=42)
df['popularity'] = np.log1p(df['popularity'])

df.head()
```

Out[2]:

	popularity	year	genre	danceability	energy	key	loudness	mode	speechiness
882616	3.663562	2006	alt-rock	0.725	0.553	6	-6.319	0	0.0340
621408	2.484907	2023	swedish	0.277	0.164	9	-16.743	0	0.0373
927704	3.663562	2007	alt-rock	0.486	0.927	2	-4.845	0	0.0428
439351	2.944439	2020	dubstep	0.411	0.442	1	-12.745	0	0.0270
266036	2.944439	2017	dancehall	0.748	0.660	10	-4.648	0	0.2710

In [3]:
```
# Define features and target variable
X = df[['danceability', 'instrumentalness', 'valence', 'duration_ms']]
y = df['popularity']

# Split data into 70% train, 15% validation, and 15% test
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, randor
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, 

# Standardize the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

# Add constant to features, fit the model with statsmodels, and print summary
X_copy = X.copy()
y_copy = y.copy()
X_copy = sm.add_constant(X_copy)
model = sm.OLS(y_copy, X_copy).fit()
print(model.summary())
```

```
# Function to evaluate models
def evaluate_model(model, X_val, y_val):
    y_val_pred = model.predict(X_val)
    mse_val = mean_squared_error(y_val, y_val_pred)
    r2_val = r2_score(y_val, y_val_pred)
    mae_val = mean_absolute_error(y_val, y_val_pred)
    rmse_val = np.sqrt(mse_val)
    return mse_val, r2_val, mae_val, rmse_val
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:             popularity   R-squared:                       0.068
Model:                            OLS   Adj. R-squared:                  0.065
Method:                 Least Squares   F-statistic:                     18.25
Date:                Sat, 03 Aug 2024   Prob (F-statistic):           1.77e-14
Time:                        22:56:51   Log-Likelihood:                -1581.9
No. Observations:                1000   AIC:                             3174.
Df Residuals:                     995   BIC:                             3198.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
======
                       coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------
const                2.9280      0.146     20.119      0.000       2.642
3.214
danceability         0.8877      0.243      3.647      0.000       0.410
1.365
instrumentalness    -0.5850      0.107     -5.458      0.000      -0.795
-0.375
valence             -0.9645      0.170     -5.681      0.000      -1.298
-0.631
duration_ms      -1.343e-06   3.11e-07     -4.320      0.000   -1.95e-06   -7.
33e-07
==============================================================================
Omnibus:                      102.740   Durbin-Watson:                   1.985
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              134.567
Skew:                          -0.895   Prob(JB):                     6.01e-30
Kurtosis:                       2.850   Cond. No.                     2.08e+06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correct
ly specified.
[2] The condition number is large, 2.08e+06. This might indicate that there ar
e
strong multicollinearity or other numerical problems.
```

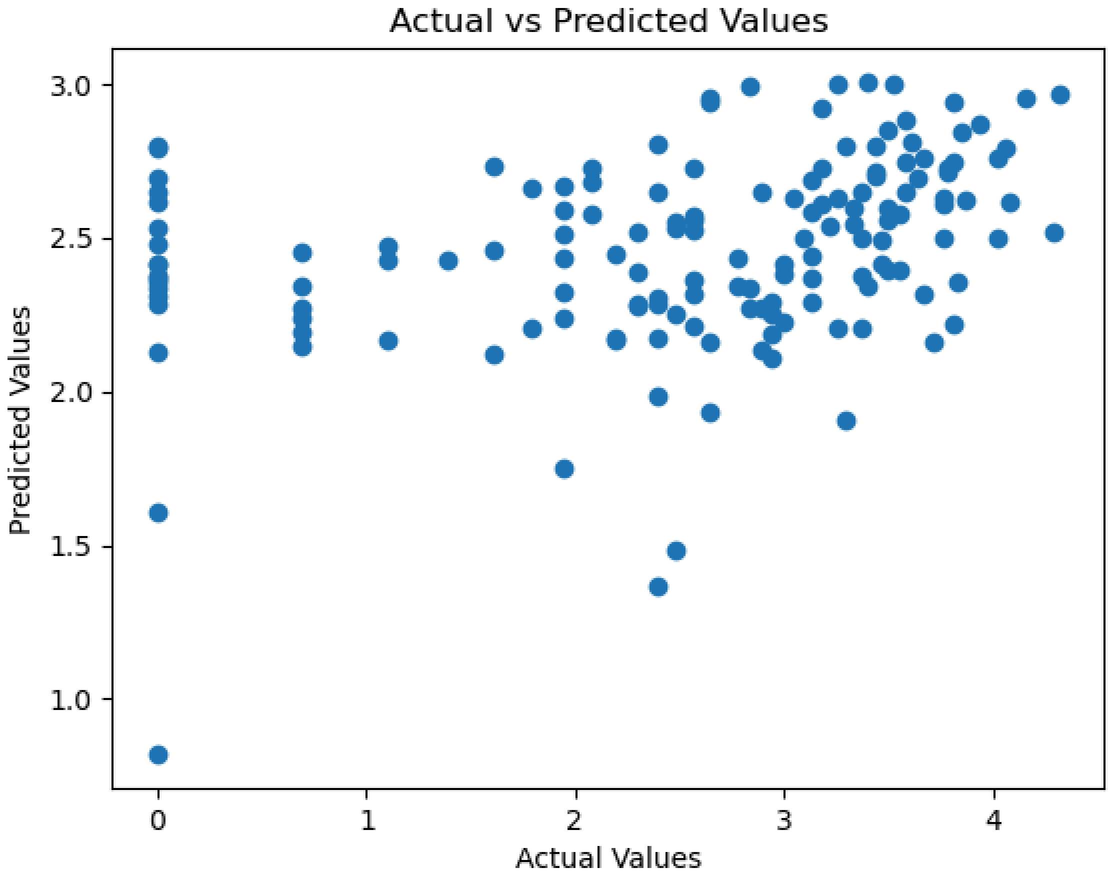
In [4]:

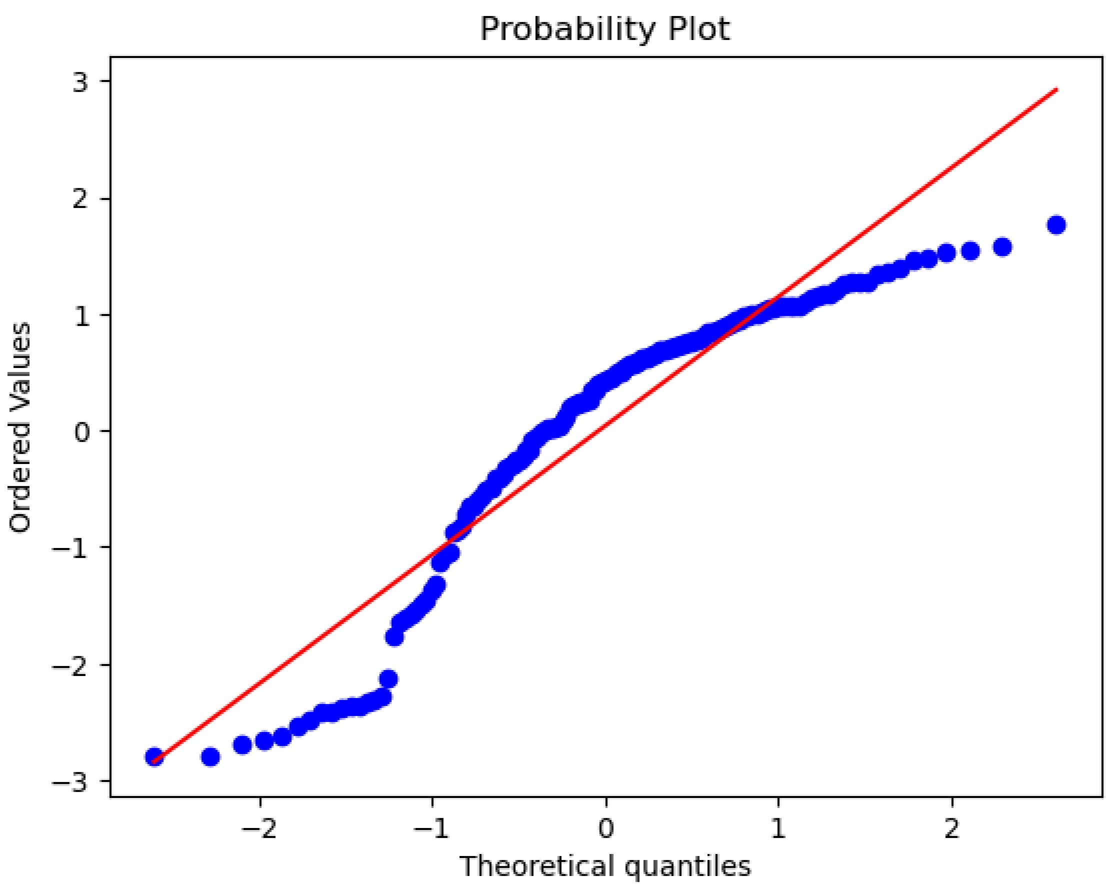
```
# Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
mse_val_lr, r2_val_lr, mae_val_lr, rmse_val_lr = evaluate_model(lr_model, X_va

# Ridge Regression
ridge = Ridge()
param_grid = {'alpha': np.logspace(-3, 3, 10)}
ridge_cv = GridSearchCV(ridge, param_grid, cv=5, scoring='neg_mean_squared_err
ridge_cv.fit(X_train, y_train)
```

```
mse_val_ridge, r2_val_ridge, mae_val_ridge, rmse_val_ridge = evaluate_model(ri

# Lasso Regression
lasso = Lasso()
param_grid = {'alpha': np.logspace(-3, 3, 10)}
lasso_cv = GridSearchCV(lasso, param_grid, cv=5, scoring='neg_mean_squared_err
lasso_cv.fit(X_train, y_train)
mse_val_lasso, r2_val_lasso, mae_val_lasso, rmse_val_lasso = evaluate_model(la

# ElasticNet Regression
elastic_net = ElasticNet()
param_grid = {'alpha': np.logspace(-3, 3, 10), 'l1_ratio': np.linspace(0.1, 1,
elastic_net_cv = GridSearchCV(elastic_net, param_grid, cv=5, scoring='neg_mean_
elastic_net_cv.fit(X_train, y_train)
mse_val_elastic_net, r2_val_elastic_net, mae_val_elastic_net, rmse_val_elastic_
```

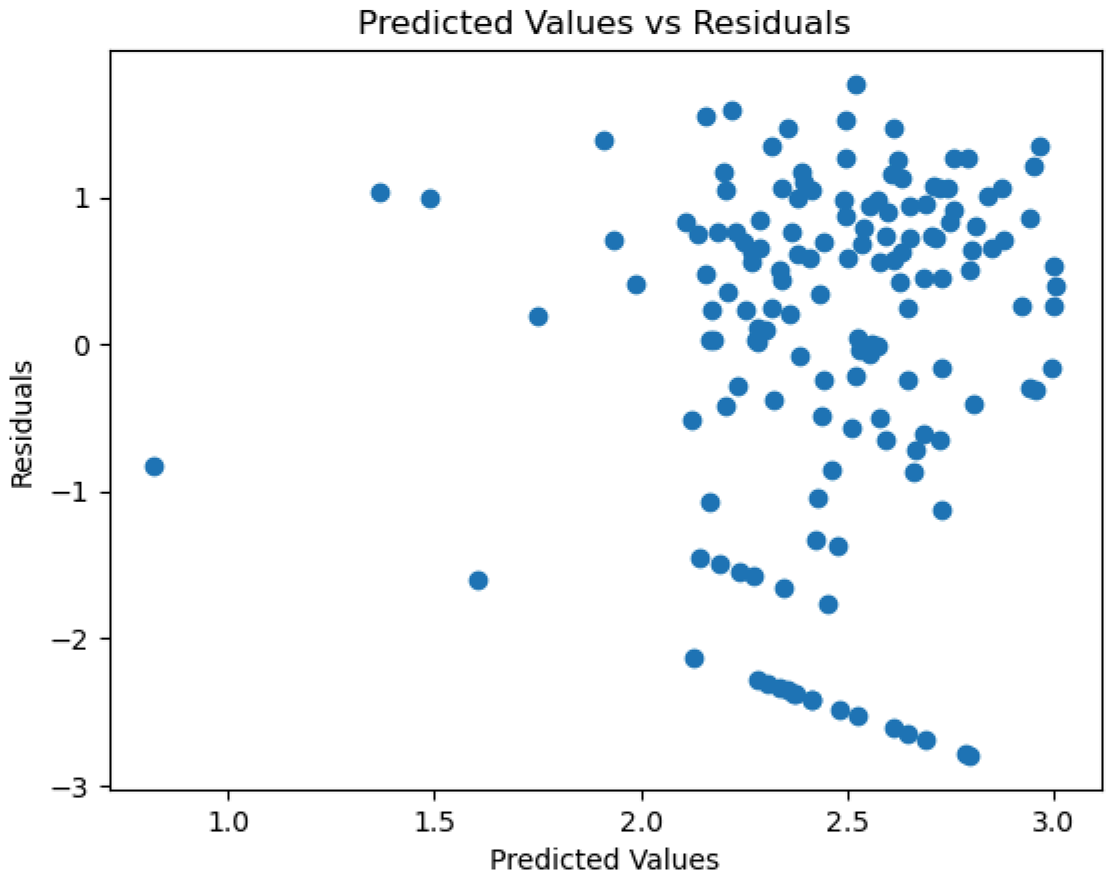
In [5]:

```
# Assumption Checking for Linear Regression
# Linearity
plt.scatter(y_val, lr_model.predict(X_val))
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Actual vs Predicted Values')
plt.show()

# Normality of Residuals
residuals = y_val - lr_model.predict(X_val)
probplot(residuals, dist="norm", plot=plt)
plt.show()

# Shapiro-Wilk test
shapiro_test = shapiro(residuals)
print(f"Shapiro-Wilk test: {shapiro_test}")

# Homoscedasticity
plt.scatter(lr_model.predict(X_val), residuals)
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Predicted Values vs Residuals')
plt.show()

# Multicollinearity
vif_data = pd.DataFrame()
vif_data["feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X_train, i) for i in range(len(X.
print(vif_data)

# Autocorrelation
dw_test = durbin_watson(residuals)
print(f"Durbin-Watson test: {dw_test}")
```






```
Shapiro-Wilk test: ShapiroResult(statistic=0.8877507448196411, pvalue=2.847743
1524009944e-09)
```




```
              feature       VIF
0        danceability  1.384544
1  instrumentalness  1.107301
2             valence  1.491721
3         duration_ms  1.086401
Durbin-Watson test: 2.1424983451625663
```

In [6]:
```
# Create interaction terms
interactions = PolynomialFeatures(degree=2, include_bias=False, interaction_on
X_train_interactions = interactions.fit_transform(X_train)
X_val_interactions = interactions.transform(X_val)
X_test_interactions = interactions.transform(X_test)

# Fit linear regression model with interaction terms, then evaluate on validat
lr_interactions = LinearRegression()
lr_interactions.fit(X_train_interactions, y_train)
mse_val_lri, r2_val_lri, mae_val_lri, rmse_val_lri = evaluate_model(lr_interac
print(f"Linear Regression with Interactions Validation MSE: {mse_val_lri}, R^2

# Evaluate on test data
y_test_pred_interactions = lr_interactions.predict(X_test_interactions)
mse_test_interactions = mean_squared_error(y_test, y_test_pred_interactions)
r2_test_interactions = r2_score(y_test, y_test_pred_interactions)
mae_test_interactions = mean_absolute_error(y_test, y_test_pred_interactions)
rmse_test_interactions = np.sqrt(mse_test_interactions)
print(f"Interaction Terms Test MSE: {mse_test_interactions}, R^2: {r2_test_int
```

```
Linear Regression with Interactions Validation MSE: 1.355373367876404, R^2: 0.
09013118704420131, MAE: 0.9405954110921296, RMSE: 1.1642050368712566
Interaction Terms Test MSE: 1.3540706358420913, R^2: 0.05091557484689291, MAE:
0.9101173310681738, RMSE: 1.163645408121431
```

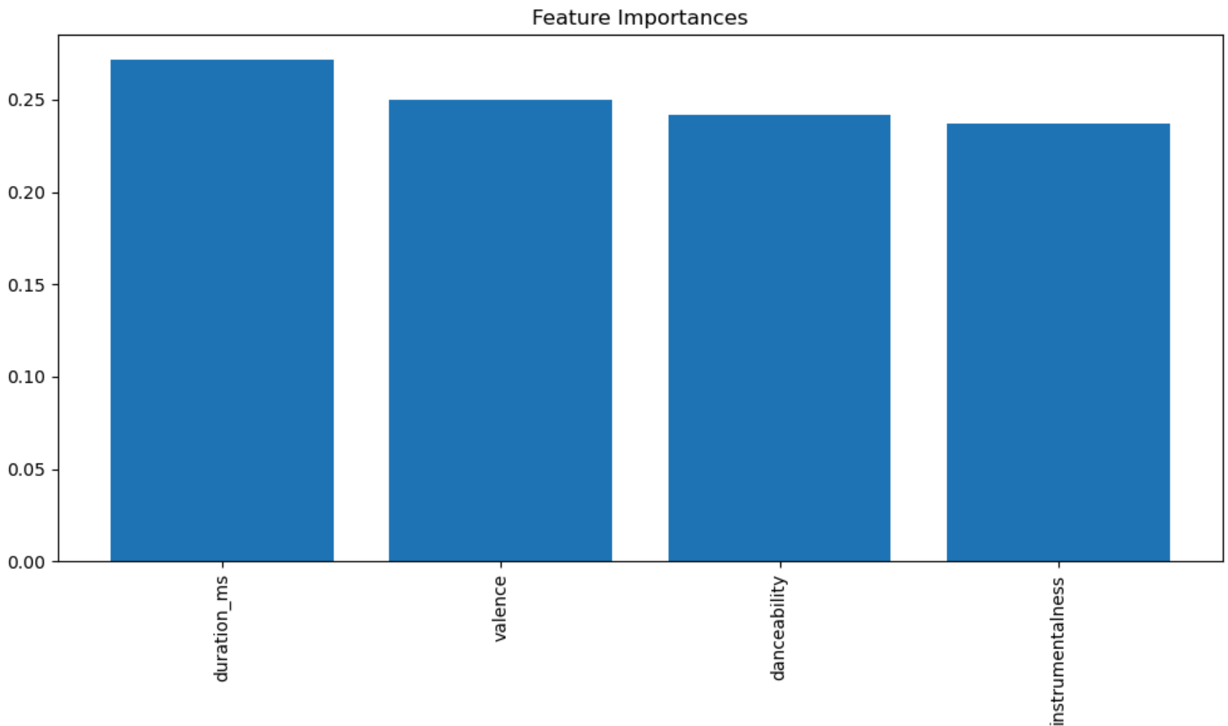
In [7]:
```
# Decision Tree Regressor
# Define parameters
param_grid = {
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2']}

# Initialize the model, GridSearchCV, and fit it
dt_regressor = DecisionTreeRegressor(random_state=42)
grid_search_dt = GridSearchCV(estimator=dt_regressor, param_grid=param_grid, cv
grid_search_dt.fit(X_train, y_train)

# Best parameters and score
print("Best parameters:", grid_search_dt.best_params_)
print("Best score (negative MSE):", grid_search_dt.best_score_)

# Evaluate on validation data
best_dt_regressor = grid_search_dt.best_estimator_
mse_val_dt, r2_val_dt, mae_val_dt, rmse_val_dt = evaluate_model(best_dt_regress
print(f"Tuned Decision Tree Regressor Validation MSE: {mse_val_dt}, R^2: {r2_va
```

```
Best parameters: {'max_depth': 10, 'max_features': None, 'min_samples_leaf':
4, 'min_samples_split': 2}
Best score (negative MSE): -1.9263691992555525
Tuned Decision Tree Regressor Validation MSE: 1.9851332151161853, R^2: -0.3326
2969806383214, MAE: 1.096318620086914, RMSE: 1.4089475558430786
```

In [8]:
```
# Random Forest Regressor
# Define the the model
rf_regressor = RandomForestRegressor(oob_score=True, random_state=42)

# Try different values for n_estimators, max_depth, and max_features
n_estimators_options = [50, 100, 200]
max_depth_options = [None, 10, 20, 30, 40]
max_features_options = [None, 'sqrt', 'log2']

# Initialize variables for parameters and OOB score
best_n_estimators = 0
best_max_depth = None
best_max_features = ''
best_oob_score = 0

# Iterate through all combinations of hyperparameters
for n_estimators in n_estimators_options:
    for max_depth in max_depth_options:
        for max_features in max_features_options:
            rf_regressor.set_params(n_estimators=n_estimators, max_depth=max_de
            rf_regressor.fit(X_train, y_train)

            # Update best parameters based on OOB score
            if rf_regressor.oob_score_ > best_oob_score:
                best_oob_score = rf_regressor.oob_score_
                best_n_estimators = n_estimators
                best_max_depth = max_depth
                best_max_features = max_features

print(f'Optimal n_estimators: {best_n_estimators}')
print(f'Optimal max_depth: {best_max_depth}')
print(f'Optimal max_features: {best_max_features}')
```

```
print(f'Best OOB score: {best_oob_score}')

# Train the best model
best_rf_regressor = RandomForestRegressor(n_estimators=best_n_estimators, max_(
best_rf_regressor.fit(X_train, y_train)

# Plot feature importances to see what has the greatest affect
importances = best_rf_regressor.feature_importances_
indices = np.argsort(importances)[::-1]
features = X.columns

plt.figure(figsize=(10, 6))
plt.title('Feature Importances')
plt.bar(range(X.shape[1]), importances[indices], align='center')
plt.xticks(range(X.shape[1]), [features[i] for i in indices], rotation=90)
plt.tight_layout()
plt.show()
```

```
Optimal n_estimators: 200
Optimal max_depth: 10
Optimal max_features: sqrt
Best OOB score: 0.0340203588565573
```



In [9]:
```
# Model selection
models = {
    "Linear Regression": lr_model,
    "Ridge Regression": ridge_cv,
    "Lasso Regression": lasso_cv,
    "ElasticNet Regression": elastic_net_cv,
    "Decision Tree Regressor": best_dt_regressor,
    "Random Forest Regressor": best_rf_regressor
}

best_model_name = None
best_mse_val = float('inf')

print(f"Linear Regression with Interactions Validation MSE: {mse_val_lri}, R^2
```

```
# Loop through each model to find the best predictor
for model_name, model in models.items():
    mse_val, r2_val, mae_val, rmse_val = evaluate_model(model, X_val, y_val)
    print(f"{model_name} Validation MSE: {mse_val}, R^2: {r2_val}, MAE: {mae_va
    if mse_val < best_mse_val:
        best_model_name = model_name
        best_mse_val = mse_val
        best_model = model


# Test the final models and find the predictive metrics
print()
if mse_val_lri > best_mse_val:   # Checking if Linear Regressions with interac
    print(f"Best model: Linear Regression with Interactions with Validation MSE
    y_test_pred_interactions = lr_interactions.predict(X_test_interactions)
    mse_test_interactions = mean_squared_error(y_test, y_test_pred_interactions
    r2_test_interactions = r2_score(y_test, y_test_pred_interactions)
    mae_test_interactions = mean_absolute_error(y_test, y_test_pred_interactio
    rmse_test_interactions = np.sqrt(mse_test_interactions)
    print(f"Linear Regression with Interactions Test MSE: {mse_test_interactio
else:
    print(f"Best model: {best_model_name} with Validation MSE: {best_mse_val}"
    y_test_pred = best_model.predict(X_test)
    mse_test = mean_squared_error(y_test, y_test_pred)
    r2_test = r2_score(y_test, y_test_pred)
    mae_test = mean_absolute_error(y_test, y_test_pred)
    rmse_test = np.sqrt(mse_test)
    print(f"{best_model_name} Test MSE: {mse_test}, R^2: {r2_test}, MAE: {mae_
```

```
Linear Regression with Interactions Validation MSE: 1.355373367876404, R^2: 0.
09013118704420131, MAE: 0.9405954110921296, RMSE: 1.1642050368712566
Linear Regression Validation MSE: 1.33522691896367, R^2: 0.10365559735943741,
MAE: 0.9334440087668509, RMSE: 1.1555201940960054
Ridge Regression Validation MSE: 1.3364473368783023, R^2: 0.1028363248064903,
MAE: 0.9334750289827433, RMSE: 1.1560481550862414
Lasso Regression Validation MSE: 1.3357138729771805, R^2: 0.10332870273339723,
MAE: 0.9334764706613574, RMSE: 1.1557308825921286
ElasticNet Regression Validation MSE: 1.3380441715686644, R^2: 0.1017643618192
8572, MAE: 0.933595199141141, RMSE: 1.1567385925820337
Decision Tree Regressor Validation MSE: 1.9851332151161853, R^2: -0.3326296980
6383214, MAE: 1.096318620086914, RMSE: 1.4089475558430786
Random Forest Regressor Validation MSE: 1.365089221468639, R^2: 0.083608886706
64202, MAE: 0.9399954630794586, RMSE: 1.168370327194524

Best model: Linear Regression with Interactions with Validation MSE: 1.3553733
67876404
Linear Regression with Interactions Test MSE: 1.3540706358420913, R^2: 0.05091
557484689291, MAE: 0.9101173310681738, RMSE: 1.163645408121431
```

Dataset Description

For my final project, I analyzed a dataset from Spotify to understand how different features influence a song's popularity. The dataset included features such as danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, tempo, and duration.

Assumption Checking and Preprocessing

Before building the models, I checked the assumptions of linear regression:

- Linearity: I verified the linear relationship between predictors and the response variable using scatter plots. Some predictors, like valence and danceability, showed a clear linear relationship with popularity, while others did not.
- Homoscedasticity: Assessed the constant variance of residuals through residual plots. The plots revealed heteroscedasticity, indicating that the variance of the residuals was not constant across all levels of the independent variables.
- Multicollinearity: Checked for multicollinearity using Variance Inflation Factor (VIF) values. High VIF values for some features suggested multicollinearity, indicating that these predictors were highly correlated with each other.
- Normality of Residuals: Used Q-Q plots to ensure residuals follow a normal distribution. The Q-Q plots showed deviations from normality, particularly in the tails, indicating that the residuals were not perfectly normally distributed.

Normality was somewhat addressed/improved by doing log transformation on the response variable

Model Building and Evaluation

Despite some problems with linear regression assumptions, I built multiple regression models, including Linear Regression, Linear Regression with Interactions, Ridge Regression, Lasso Regression, ElasticNet Regression, Decision Tree Regressor, and Random Forest Regressor. The data was split into training, validation, and test sets to evaluate the models' performance. Linear Regression with Interations was used to explore if the relationship between the predictors and the dependent variable changes when the predictors interact with each other. This allows us to see if there are more complex relationships within the data, which might improve the model’s performance on predicting outcome.

Statistically Significant Features

In the linear regression model, the statistically significant features affecting song popularity were:

- Danceability ($p = 0.001$)
- Instrumentalness ($p = 0.000$)
- Valence ($p = 0.000$)
- Duration ($p = 0.000$)

Best Model

The model that performed best was the Linear Regression with Interactions, which had the lowest validation MSE of 1.355. Despite having a slightly lower R^2 value compared to the simpler linear regression model, it demonstrated the most effectiveness in terms of MAE and RMSE. This tells us that accounting for interactions between features improved the model's predictive performance.

Performing Metrics

The Linear Regression with Interactions model achieved the following metrics on the test set:

- Test MSE: 1.354
- Test R^2: 0.051
- Test MAE: 0.910
- Test RMSE: 1.164

These metrics suggest that while the model captures some variance in song popularity, there is room for improvement. Additional features or more complex models could further enhance predictive performance.

Non-Parametric Models

I also explored non-parametric methods such as Decision Tree Regressor and Random Forest Regressor. However, these models did not perform as well as the linear regression models. For example, the Random Forest Regressor had a validation MSE of 1.365, which was higher than the Linear Regression with Interactions model.

Conclusion

Overall, the Linear Regression with Interactions model was the best performer, finding some of the complexity in the data. The features danceability, instrumentalness, valence, and duration give the best insights into which aspects of a song contribute to its popularity. However, the model's performance metrics suggest that there are likely additional factors influencing popularity that were not in this analysis. Further research with more features or advanced modeling techniques could help find more predictive results.

Addressing Assumption Issues

The problems with linear regression assumptions, such as heteroscedasticity, multicollinearity, and non-normality of residuals, show that the linear model may not fully capture the data's effects on popularity. The issues suggest that more powerful models or feature transformations could be explored to fit the data and improve predictive performance.