**MINISTRY OF EDUCATION AND SCIENCE OF RUSSIA**
**SAINT PETERSBURG STATE**
**ELECTRICAL ENGINEERING UNIVERSITY**
**"LETI" NAMED AFTER V.I. ULYANOV (LENIN)**
**Department of Computer Engineering**

**REPORT**
**for laboratory work #4**
**in the discipline "Machine Learning"**
**Topic: Regression**

Student group 0308 ____________________ Dashkin D.Sh.

Teacher ____________________ A. Alhasan

Saint Petersburg
2024

**Purpose of the work.**

Serves to acquire and consolidate skills in data preprocessing and application of machine learning methods to solve regression problems.

**Exercise.**

During the laboratory work the following steps must be completed:

1. Data preprocessing
    a. Visualization of significant features (scatterplots, box plots, histograms)
    b. Data cleaning (removing gaps, normalization, removing duplicates)
2. Model training and parameter selection (where applicable):
    a. Linear regression
    b. LASSO
    c. Ridge regression
3. Evaluation of models
    a. Output of metrics
    b. Plotting graphs

**Completing the work.**

1. **Preparing data for training.**

We load data and process it

```python
# Загружаем данные и обрабатываем их
ds = pd.read_csv('Spotify_Youtube.csv')
url_cols = ['Url_spotify', 'Uri', 'Url_youtube', 'Title', 'Description', 'Unnamed: 0']
ds.drop(url_cols, axis=1, inplace=True)

ds = ds.drop_duplicates()

ds.head(10)
```

```python
ds.dropna(inplace=True)
ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 19549 entries, 0 to 20717
Data columns (total 22 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Artist            19549 non-null  object
 1   Track             19549 non-null  object
 2   Album             19549 non-null  object
 3   Album_type        19549 non-null  object
 4   Danceability      19549 non-null  float64
 5   Energy            19549 non-null  float64
 6   Key               19549 non-null  float64
 7   Loudness          19549 non-null  float64
 8   Speechiness       19549 non-null  float64
 9   Acousticness      19549 non-null  float64
 10  Instrumentalness  19549 non-null  float64
 11  Liveness          19549 non-null  float64
 12  Valence           19549 non-null  float64
 13  Tempo             19549 non-null  float64
 14  Duration_ms       19549 non-null  float64
 15  Channel           19549 non-null  object
 16  Views             19549 non-null  float64
 17  Likes             19549 non-null  float64
 18  Comments          19549 non-null  float64
 19  Licensed          19549 non-null  object
 20  official_video    19549 non-null  object
 21  Stream            19549 non-null  float64
dtypes: float64(15), object(7)
memory usage: 3.4+ MB
```

Figure 1. Loading and processing

```
columns=['Artist','Track', 'Album', 'Album_type', 'Key', 'Channel', 'Comments','Licensed', 'official_video']
ds.drop(columns, axis=1, inplace=True)
ds.describe()
```

| | Danceability | Energy | Loudness | Speechiness | Acousticness | Instrumentalness | Liveness | Valence | Tempo | Duration_ms | Views | Likes | Stream |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 1.954900e+04 | 1.954900e+04 | 1.954900e+04 | 1.954900e+04 |
| **mean** | 0.621059 | 0.635170 | -7.633179 | 0.095392 | 0.289106 | 0.055292 | 0.191226 | 0.528950 | 120.605702 | 2.246281e+05 | 9.545626e+07 | 6.700487e+05 | 1.371101e+08 |
| **std** | 0.165489 | 0.213555 | 4.618839 | 0.106243 | 0.285908 | 0.192519 | 0.165197 | 0.245228 | 29.619340 | 1.269126e+05 | 2.775744e+08 | 1.805054e+06 | 2.463589e+08 |
| **min** | 0.000000 | 0.000020 | -46.251000 | 0.000000 | 0.000001 | 0.000000 | 0.014500 | 0.000000 | 0.000000 | 3.098500e+04 | 2.600000e+01 | 0.000000e+00 | 6.574000e+03 |
| **25%** | 0.519000 | 0.508000 | -8.772000 | 0.035700 | 0.044400 | 0.000000 | 0.094000 | 0.339000 | 96.990000 | 1.802400e+05 | 1.911528e+06 | 2.238000e+04 | 1.781089e+07 |
| **50%** | 0.639000 | 0.666000 | -6.516000 | 0.050700 | 0.190000 | 0.000002 | 0.125000 | 0.536000 | 119.964000 | 2.132530e+05 | 1.491440e+07 | 1.279090e+05 | 4.979139e+07 |
| **75%** | 0.742000 | 0.797000 | -4.929000 | 0.104000 | 0.470000 | 0.000433 | 0.234000 | 0.725000 | 139.951000 | 2.519200e+05 | 7.152989e+07 | 5.266400e+05 | 1.390828e+08 |
| **max** | 0.975000 | 1.000000 | 0.920000 | 0.964000 | 0.996000 | 1.000000 | 1.000000 | 0.993000 | 243.372000 | 4.676058e+06 | 8.079649e+09 | 5.078865e+07 | 3.386520e+09 |

Figure 2. Data after processing.

## 2. Data visualization

Now we will plot some scatter plots, histograms and a box plot for the Danceability parameter.

```
sns.scatterplot(data = ds, x = "Views", y = "Likes")
```

```
<Axes: xlabel='Views', ylabel='Likes'>
```

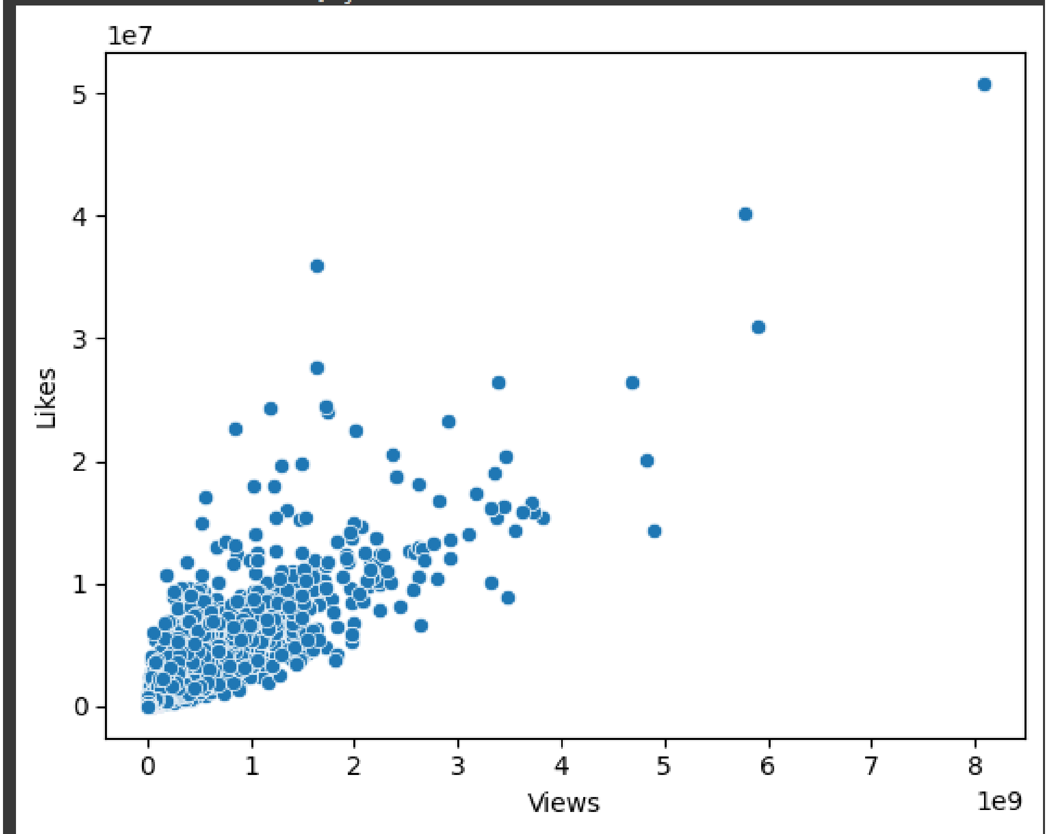


Figure 3. Scatterplot for Likes from Views.

```
sns.scatterplot(data = ds, x ="Loudness", y = "Energy")
```

```
<Axes: xlabel='Loudness', ylabel='Energy'>
```

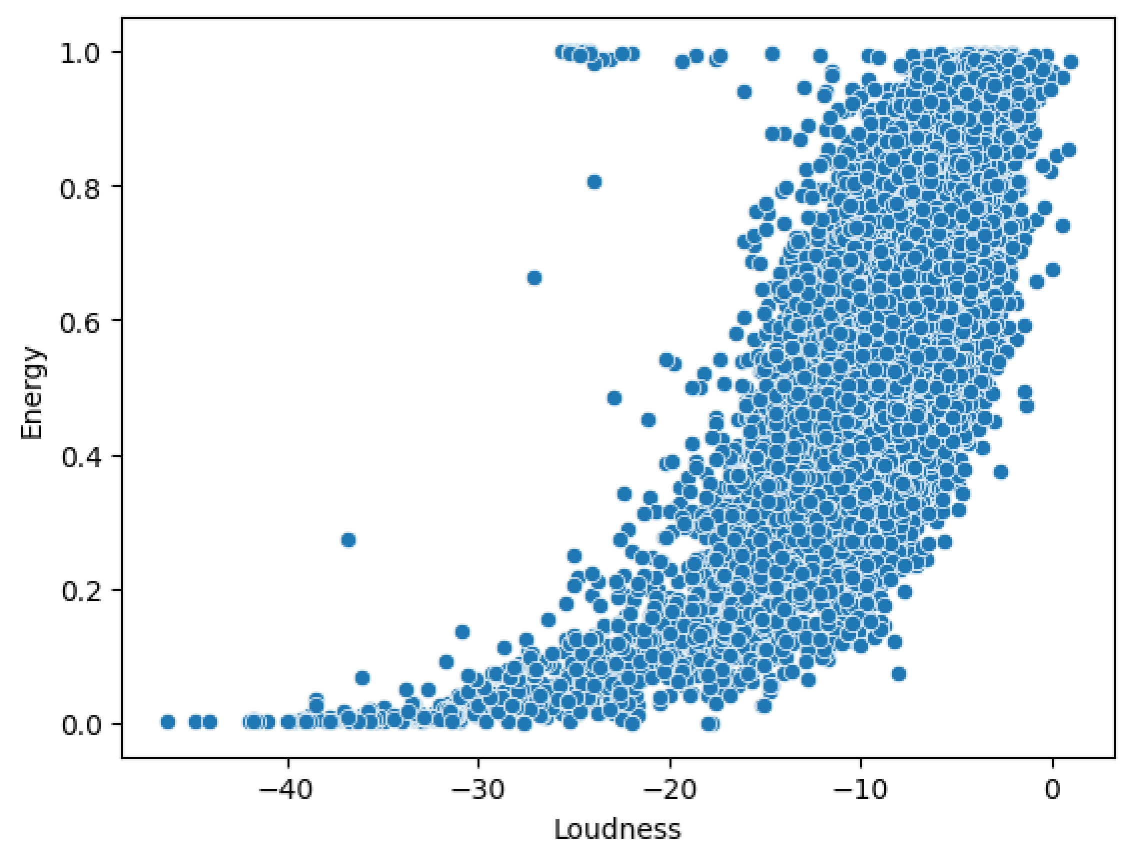


Figure 4. Scatter plot for Energy vs. Loudness

```
sns.scatterplot(data = ds, x = "Acousticness", y = "Loudness")
```

```
<Axes: xlabel='Acousticness', ylabel='Loudness'>
```

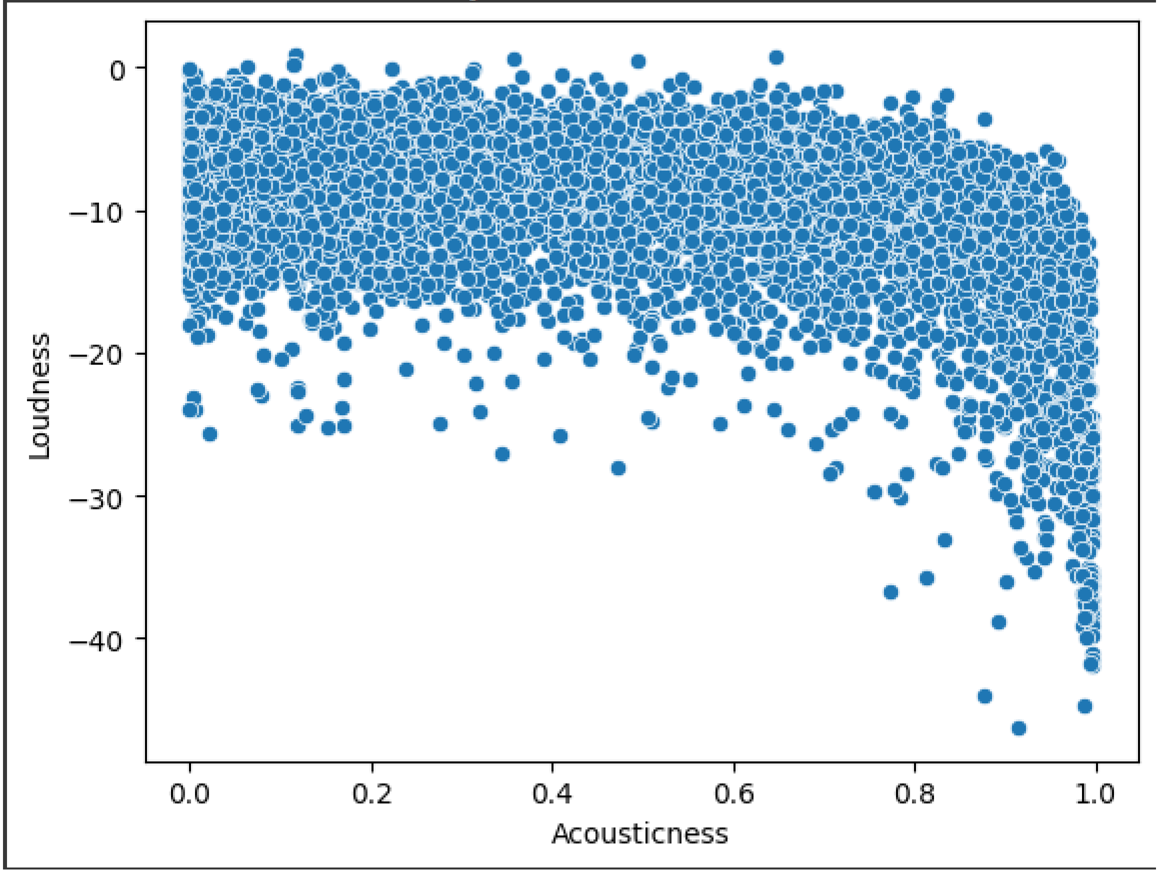


Figure 5. Scatter plot for loudness versus acoustics

```python
sns.histplot(data = ds, x = "Energy", bins = 100)
```

```
<Axes: xlabel='Energy', ylabel='Count'>
```

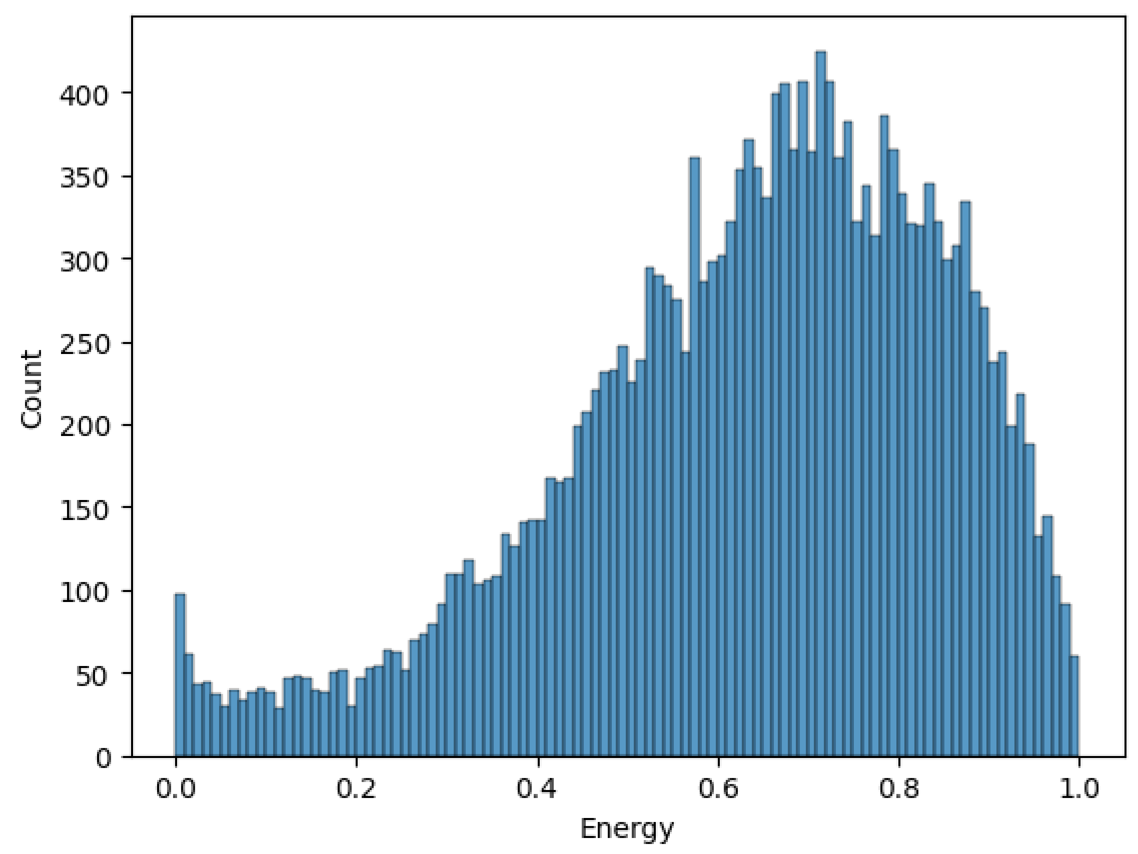


Figure 6. Histogram for Energy.

```
sns.histplot(data = ds, x = "Danceability", bins = 100)
```

```
<Axes: xlabel='Danceability', ylabel='Count'>
```

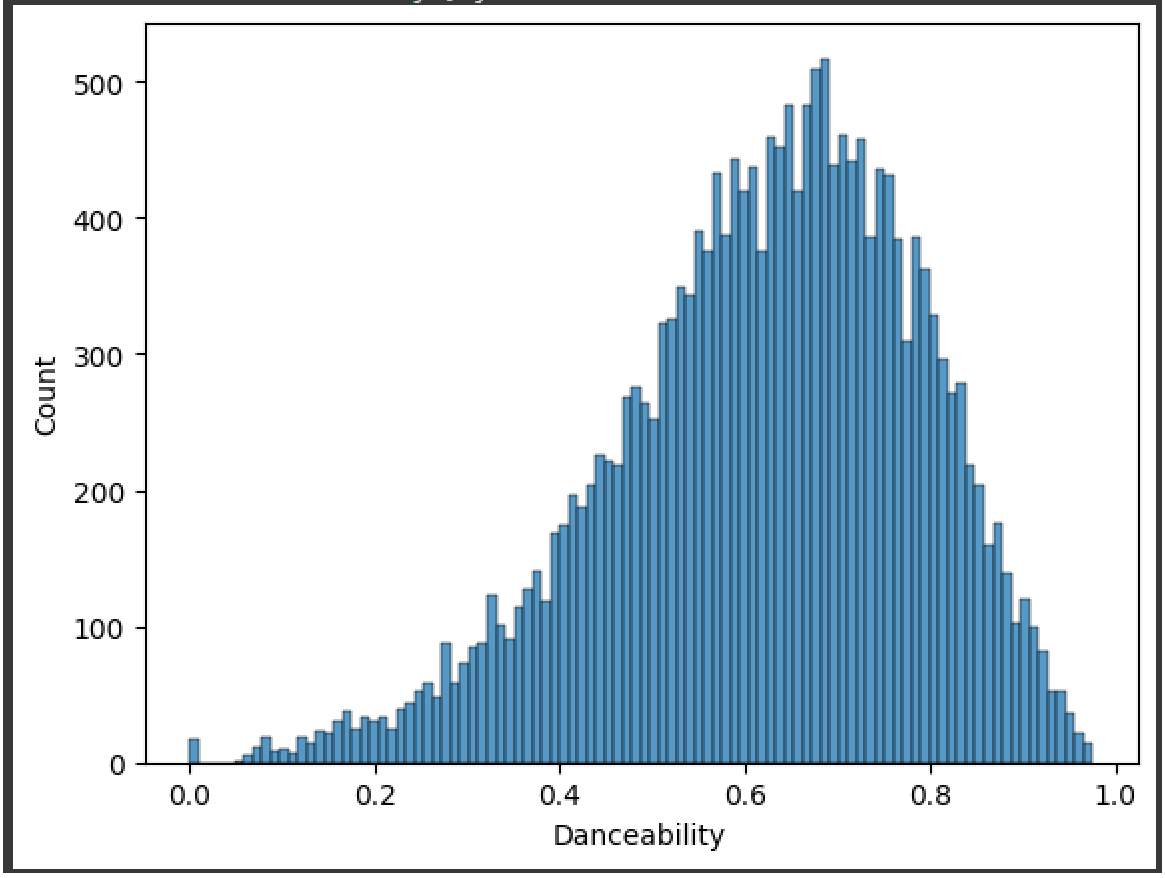


Figure 7. Histogram for Danceability.

```
sns.histplot(data = ds, x = "Loudness", bins = 50)
```

```
<Axes: xlabel='Loudness', ylabel='Count'>
```

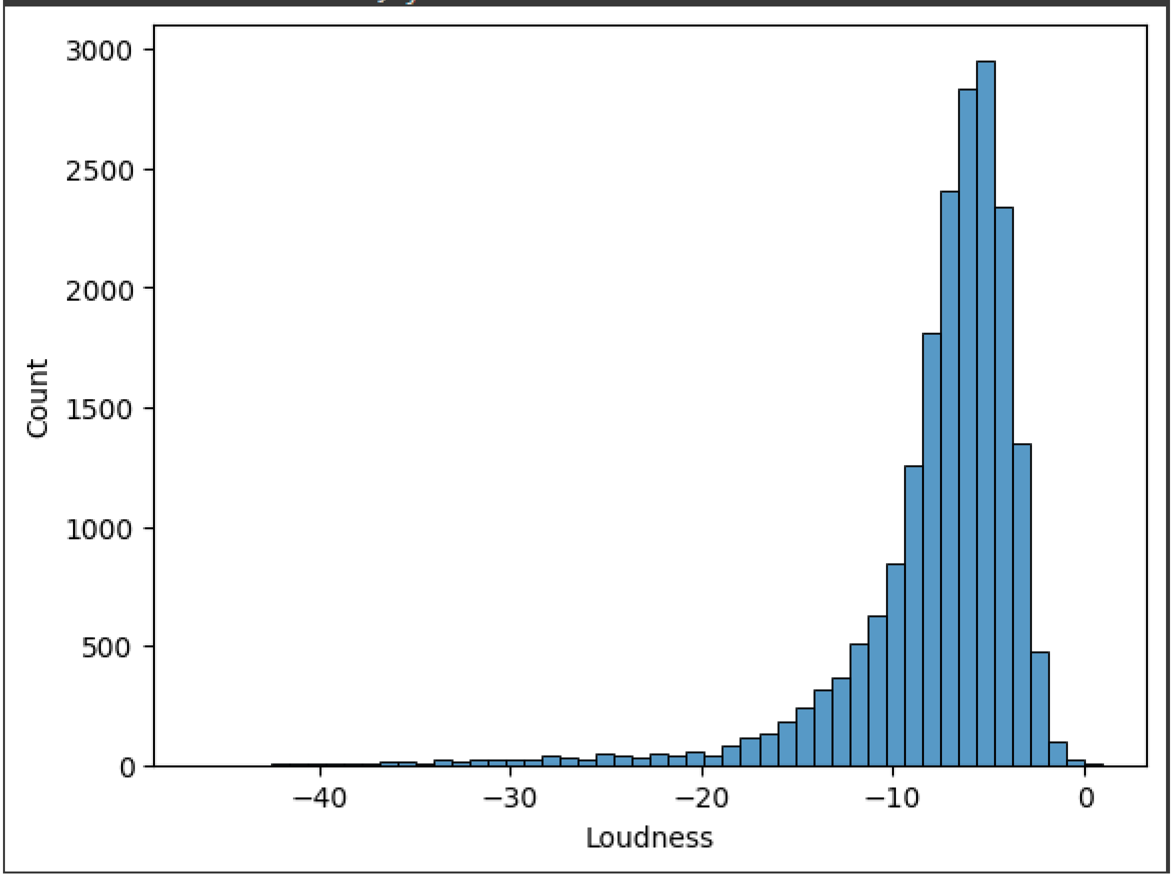


Figure 8. Histogram for Loudness

```
sns.histplot(data = ds, x = "Tempo", bins = 30)
```

```
<Axes: xlabel='Tempo', ylabel='Count'>
```

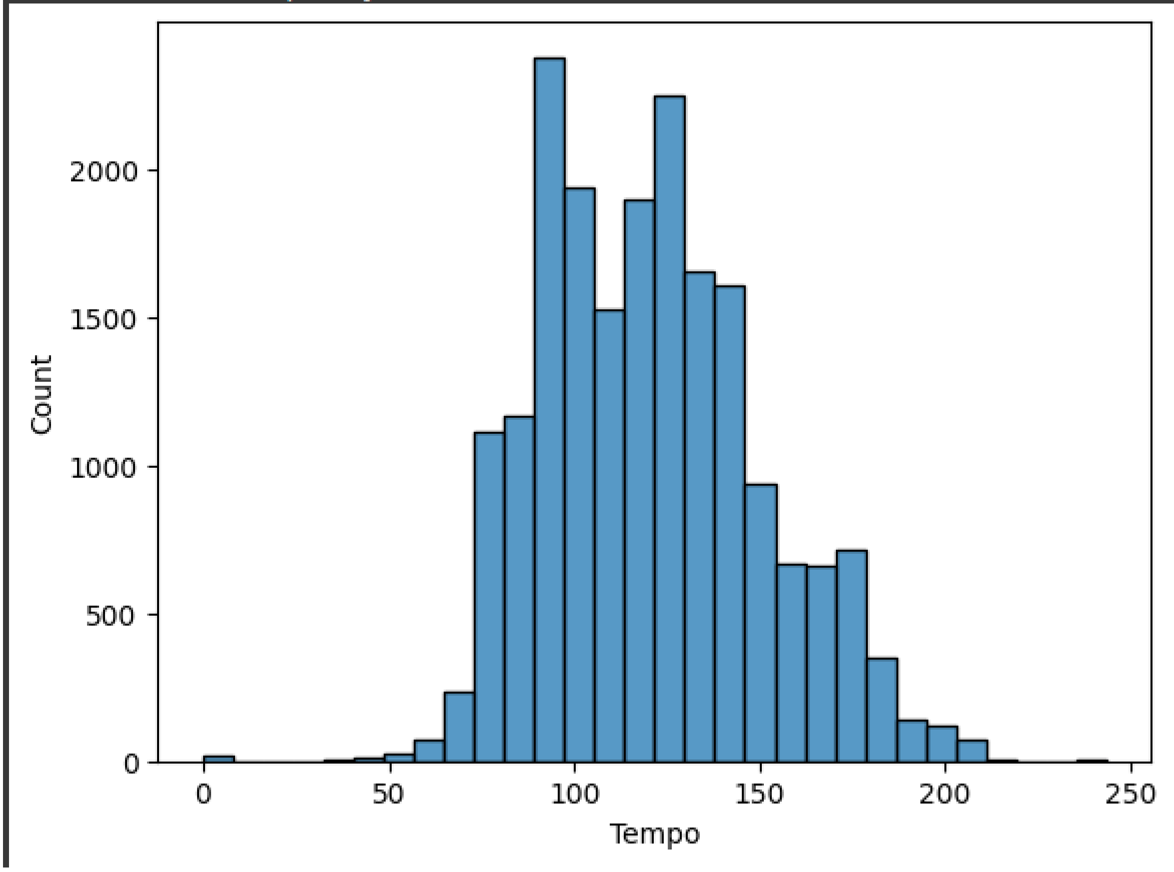


Figure 9. Histogram for Tempo

```
sns.boxplot (data = ds, x = "Danceability")
```

```
<Axes: xlabel='Danceability'>
```

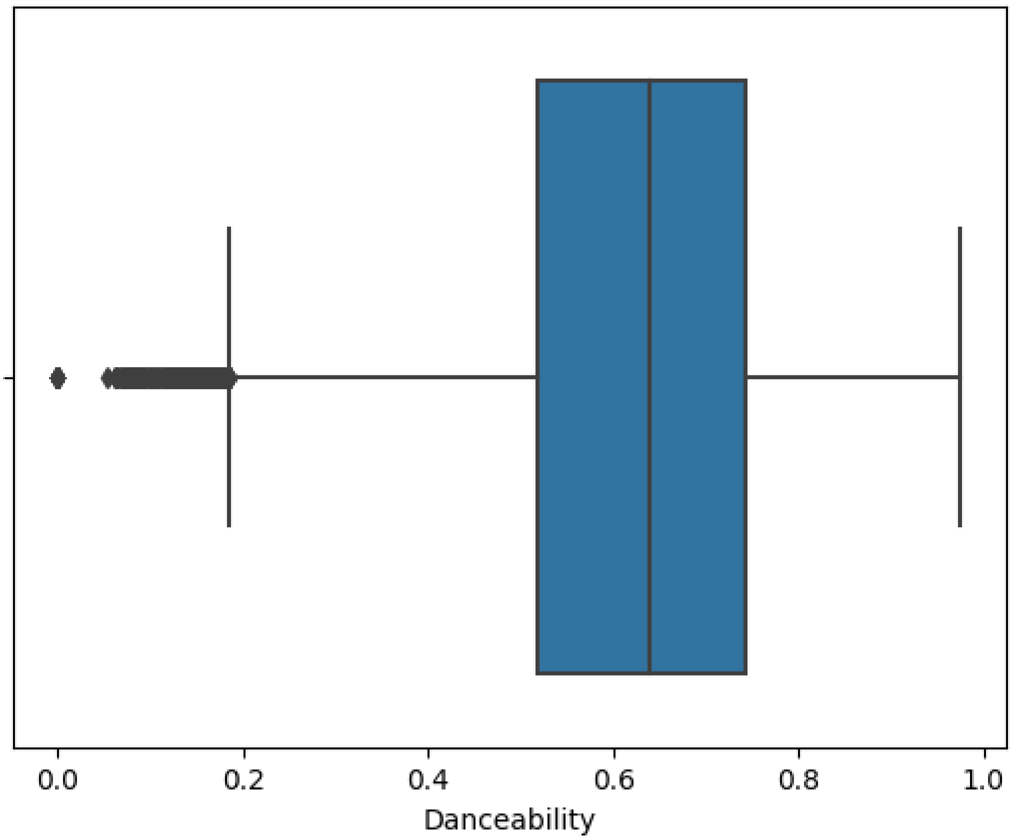


Figure 10. Box with whiskers for Danceability

## 3. Normalization

Next, the data was normalized using the MinMaxScaler() method.

НОРМАЛИЗАЦИЯ

```
ML_DF = ds[['Danceability', 'Energy', 'Loudness', 'Speechiness', 'Acousticness', 'Instrumentalness', 'Liveness', 'Valence',
            'Tempo', 'Duration_ms', 'Views', 'Likes', 'Stream']]


#Нормализация
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaled_ds = scaler.fit_transform(ML_DF)
scaled_ds =pd.DataFrame(scaled_ds,columns=['Danceability', 'Energy', 'Loudness', 'Speechiness', 'Acousticness', 'Instrumentalness', 'Liveness', 'Valence',
            'Tempo', 'Duration_ms', 'Views', 'Likes', 'Stream'])
```

```
scaled_ds.describe()
```

| | Danceability | Energy | Loudness | Speechiness | Acousticness | Instrumentalness | Liveness | Valence | Tempo | Duration_ms | Views | Likes | Stream |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 | 19549.000000 |
| mean | 0.636984 | 0.635163 | 0.818677 | 0.098955 | 0.290266 | 0.055292 | 0.179327 | 0.532679 | 0.495561 | 0.041688 | 0.011814 | 0.013193 | 0.040485 |
| std | 0.169732 | 0.213559 | 0.097917 | 0.110211 | 0.287057 | 0.192519 | 0.167628 | 0.246957 | 0.121704 | 0.027322 | 0.034355 | 0.035541 | 0.072747 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 25% | 0.532308 | 0.507990 | 0.794535 | 0.037033 | 0.044577 | 0.000000 | 0.080670 | 0.341390 | 0.398526 | 0.032132 | 0.000237 | 0.000441 | 0.005257 |
| 50% | 0.655385 | 0.665993 | 0.842361 | 0.052593 | 0.190762 | 0.000002 | 0.112126 | 0.539778 | 0.492924 | 0.039239 | 0.001846 | 0.002518 | 0.014701 |
| 75% | 0.761026 | 0.796996 | 0.876004 | 0.107884 | 0.471887 | 0.000433 | 0.222730 | 0.730111 | 0.575050 | 0.047563 | 0.008853 | 0.010369 | 0.041068 |
| max | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |

Figure 11. Normalization

And I again output histograms for some parameters

```
plt.hist(scaled_ds["Danceability"], bins=120, range=(0.0,1))
plt.title("Danceability")
```

```
Text(0.5, 1.0, 'Danceability')
```

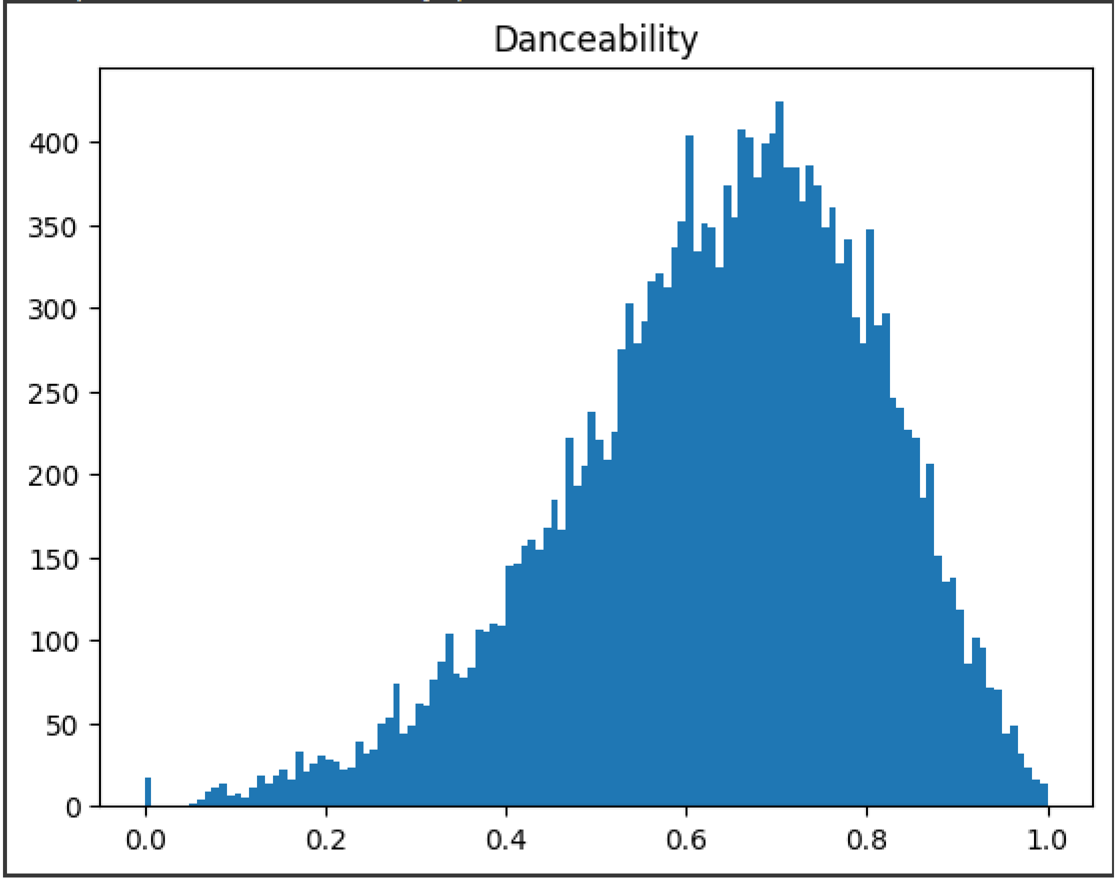


Figure 12. Histogram for Danceability

```
sns.histplot(data = scaled_ds, x = "Energy", bins = 100)
```

```
<Axes: xlabel='Energy', ylabel='Count'>
```

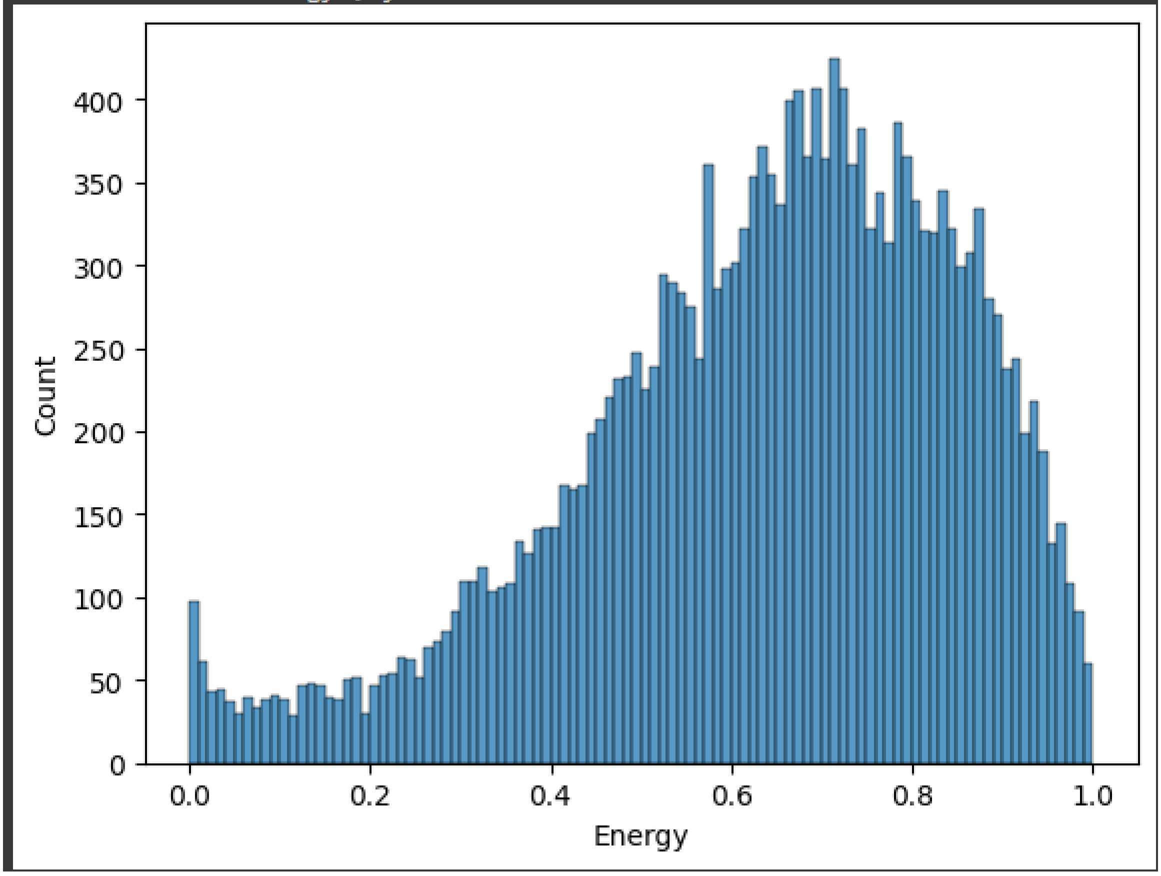


Figure 13. Histogram for Energy

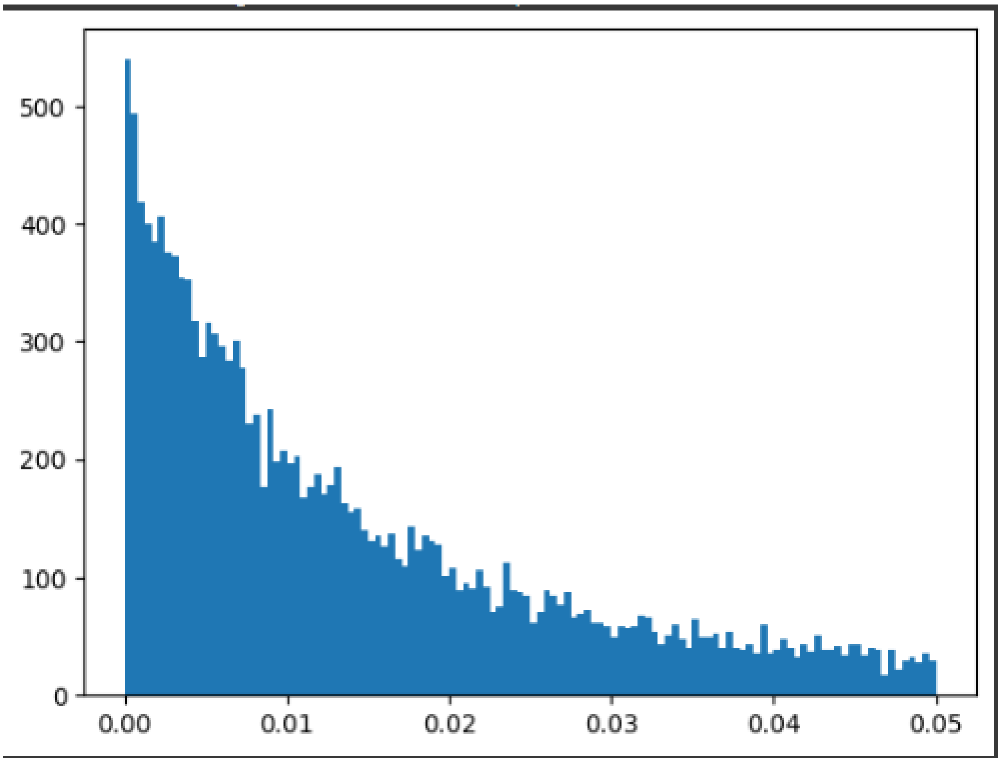


Figure 14. Histogram for Streams

## 4. Linear regression

I will predict the Streams parameter based on other parameters. Using the train_test_split method, I split the data into two categories (train = 90% of the dataset).

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.1, random_state = 42)
```

Figure 15. Train_test_split.

Next we take the linear regression method and train and predict it on the test sample. We get the coefficients.

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

# обучение на тренировочной выборке
model.fit(X_train, Y_train)

# прогноз по тестовой выборке
Y_pred = model.predict(X_train)

#выведем полученные коэффициенты для наших признаков
coef = pd.DataFrame([X_train.columns, model.coef_]).T
coef = coef.rename (columns = {0:"Attribute", 1 : "Coefficient"})
print(coef)
```

```
            Attribute Coefficient
0        Danceability   -0.003264
1              Energy   -0.028489
2            Loudness    0.041365
3         Speechiness   -0.017496
4        Acousticness   -0.020265
5    Instrumentalness   -0.010056
6            Liveness   -0.007554
7             Valence   -0.007816
8               Tempo   -0.002945
9         Duration_ms   -0.058661
10              Views    0.133381
11              Likes    1.199788
```

Figure 16. Test sample coefficients

The result of our model was:

```
#выведем результаты
train_score_lr = model.score(X_train, Y_train)
#test_score_lr = model.score(X_test, Y_test)

print("The train score for lr model is {}".format(train_score_lr))
#print("The test score for lr model is {}".format(test_score_lr))
```

```
The train score for lr model is 0.43802499036369946
```

Figure 17. Result of the linear regression model

Visualization of Predictions

```
plt.scatter(Y_train, Y_pred)
plt.xlabel("Streams")
plt.ylabel("Predicted Streams")
plt.title("Streams vs Predicted Streams")
plt.show()
```

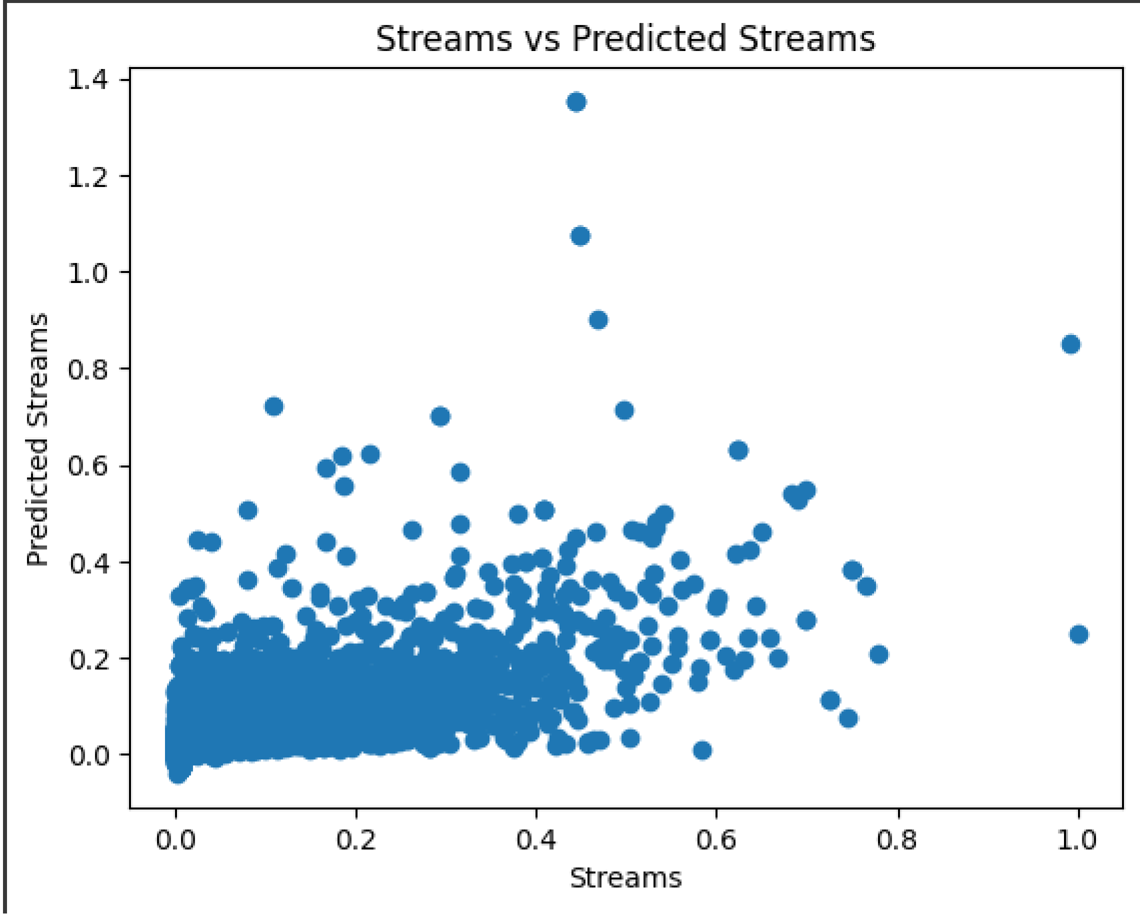


Figure 18. Predicted and actual values

Another important part of the work is checking the residuals. The residual is the difference between the actual price and the predicted price. A good model should always have only random errors, i.e. it fits the data well if the differences between the observations and the predicted values are small and unbiased. Unbiased in this context means that the predicted values are not systematically too high or too low anywhere in the observation space. If the model is biased, the results cannot be trusted.

```
#проверка остатков
plt.scatter(Y_pred, abs(Y_pred-Y_train))
plt.title("Predicted vs residuals")
plt.xlabel("Predicted")
plt.ylabel("Residuals")
plt.show()
```

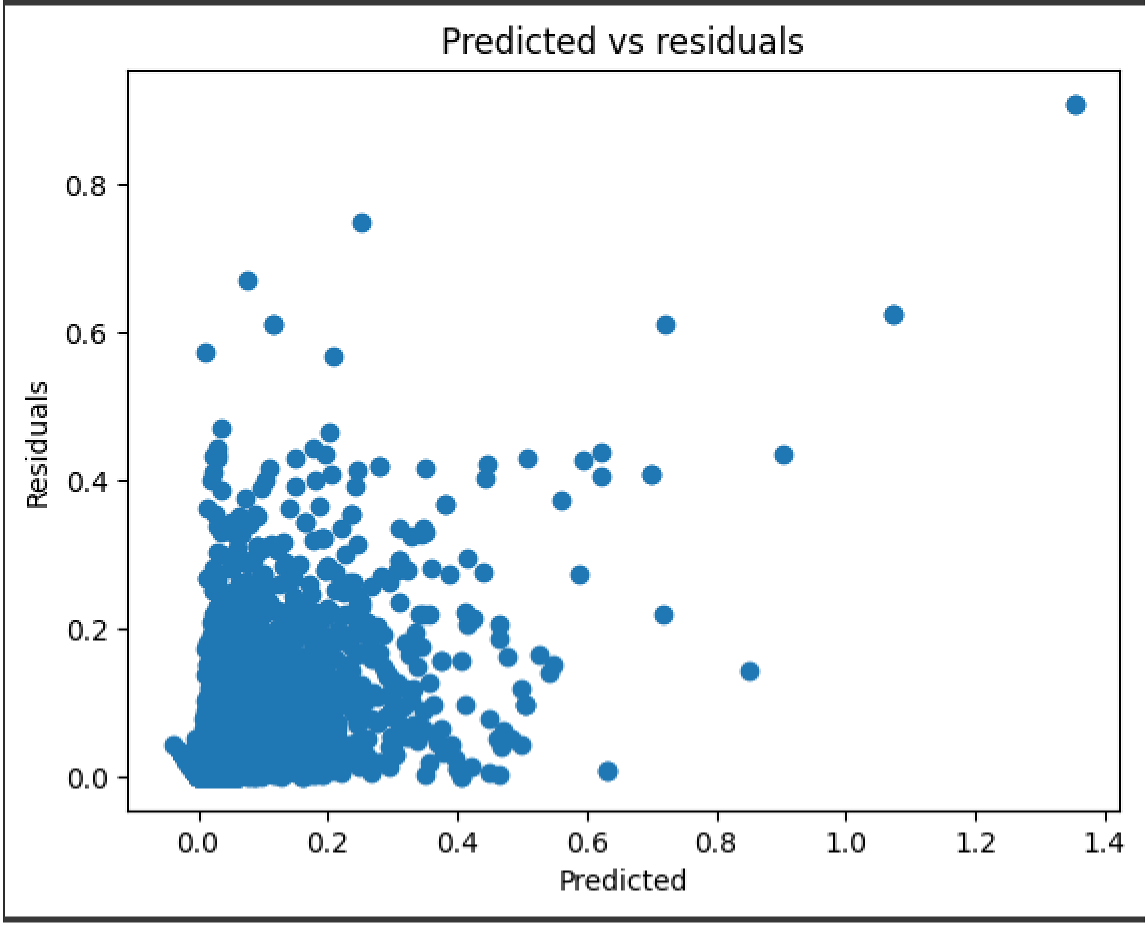


Figure 19. Remains

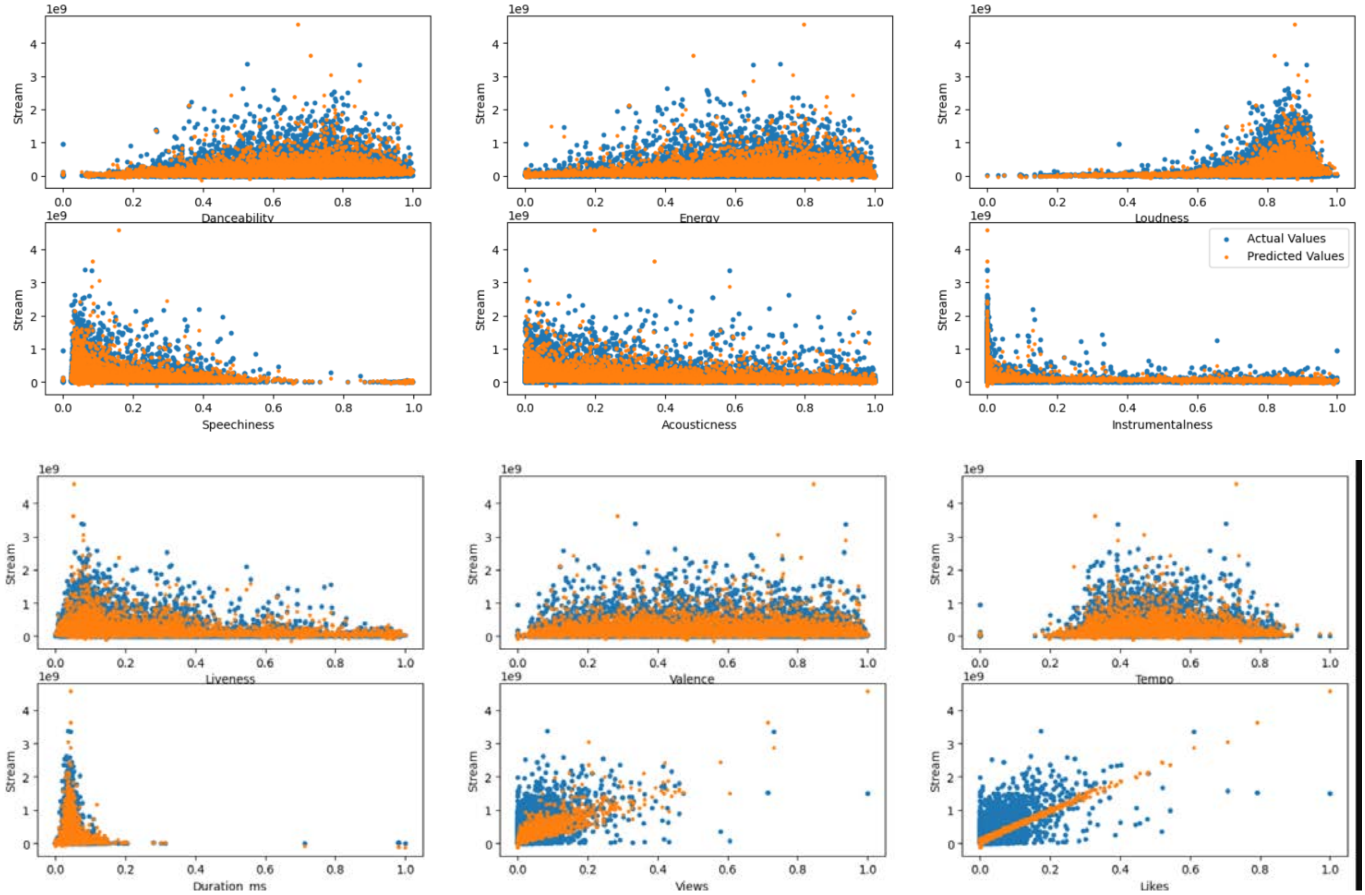


Figure 20. Predicted (orange) and actual values (blue)

There is no clear dependence, so we throw in test values and get metrics:

```
Y_test_pred = model.predict(X_test)
```

```
#оценка модели
R2_score = metrics.r2_score(Y_test, Y_test_pred)
print("LINEAR REGRESSION METRICS")
print("R^2: ", R2_score)
print("MAE: ", metrics.mean_absolute_error(Y_test, Y_test_pred))
print("MSE: ", metrics.mean_squared_error(Y_test, Y_test_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y_test, Y_test_pred)))
```

```
LINEAR REGRESSION METRICS
R^2:  0.45255311009444277
MAE:  0.030927330555887526
MSE:  0.002777614981253085
RMSE:  0.05270308322340435
```

```
#визуализация
plt.scatter(Y_test, Y_test_pred)
plt.xlabel("Streams")
plt.ylabel("Predicted Streams")
plt.title("Streams vs Predicted Streams")
plt.show()
```

Figure 21. Training the model on test values and metrics.

We also display the visualization and the remains (Fig. 22-23). We see that there is no dependence, which means the model has a right to exist.

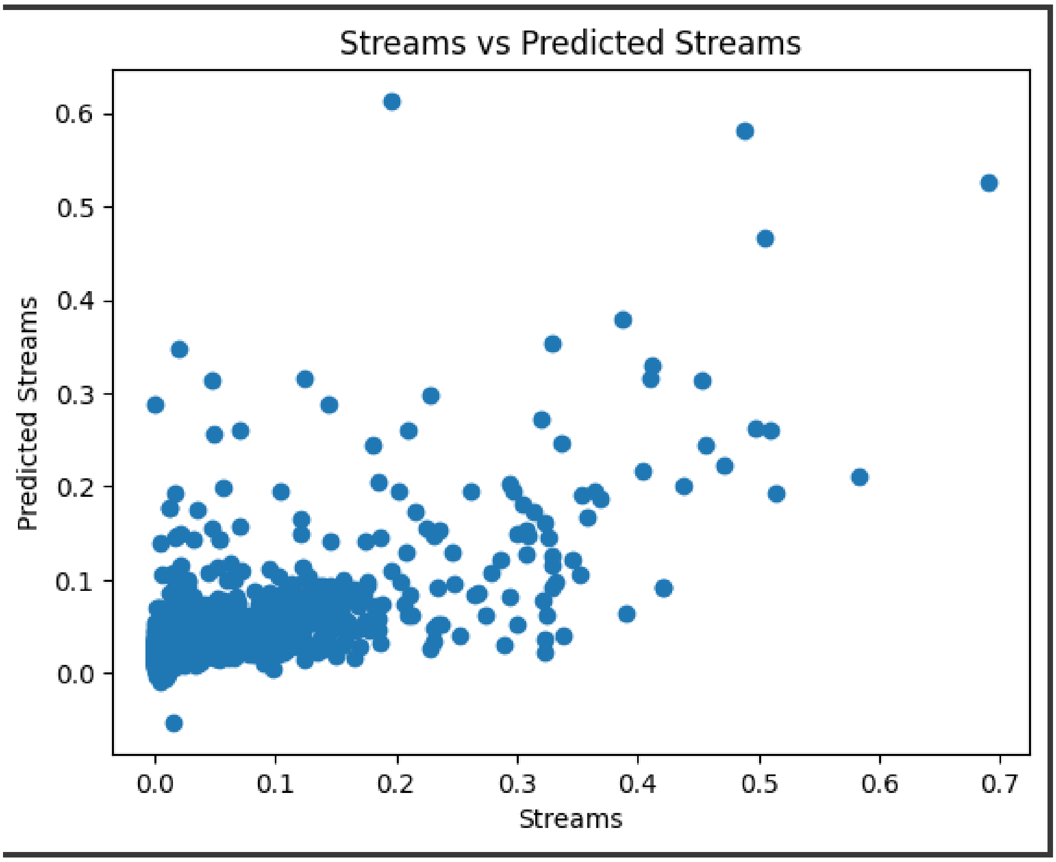


Figure 22. Visualization

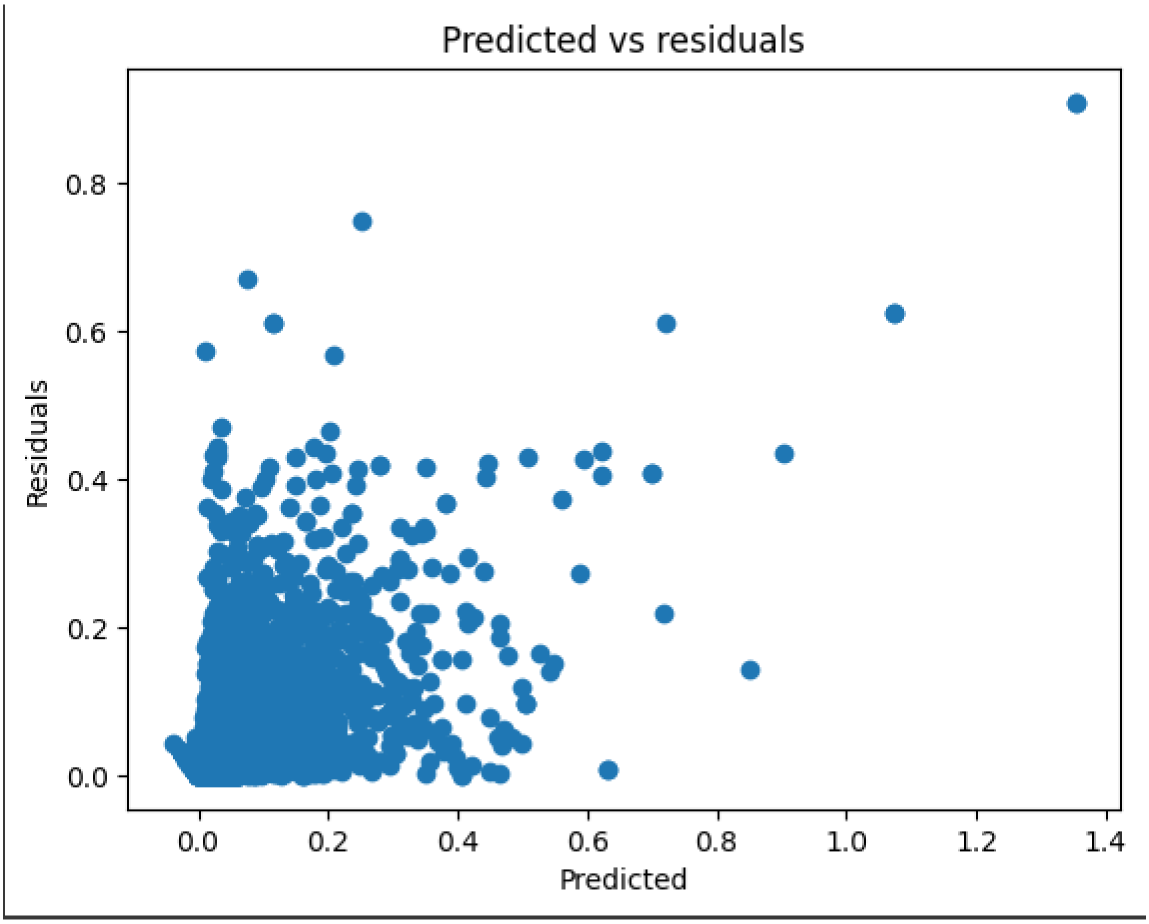


Figure 23. Remains

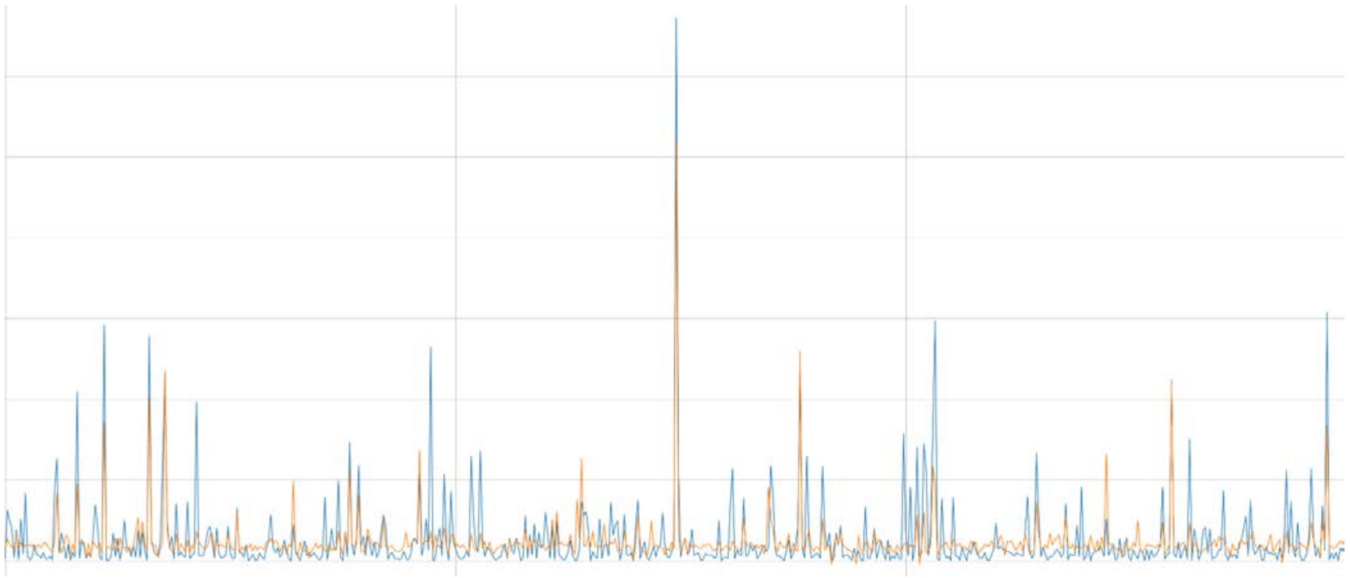

Figure 24. Actual (blue) and predicted (orange) values

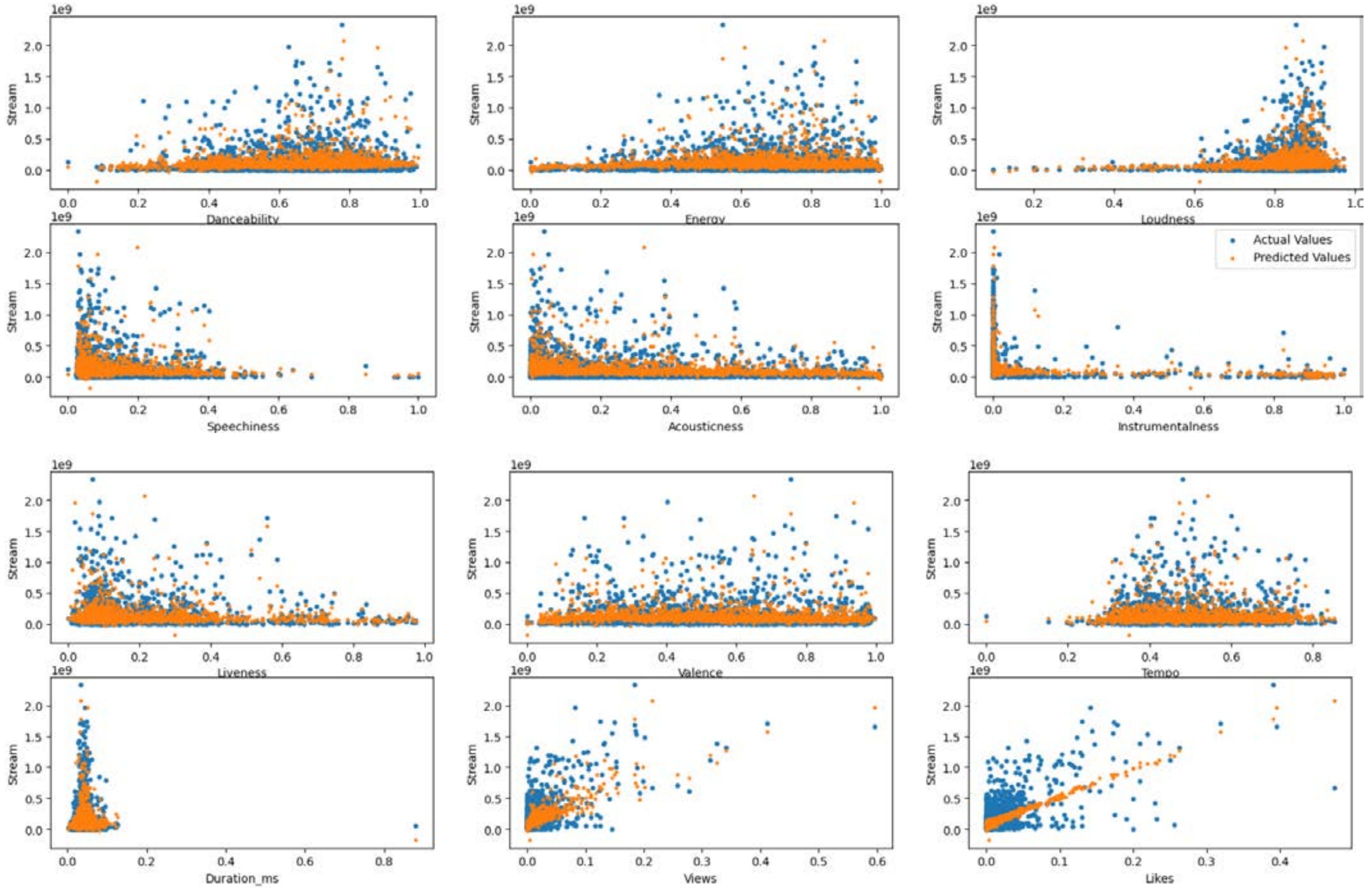


Figure 25. Actual (blue) and predicted (orange) values

## 5. LASSO

Regression method**lasso (LASSO, Least Absolute Shrinkage and Selection Operator) Penalizes the L1 norm of the vector "Try to achieve the best performance, but if some coefficients are useless, then discard them"**

We will go through different regularization coefficients and create a LASSO model for each one. The best coefficient has the least squared error.

```
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
lambda1_values = [0.000001, 0.0001, 0.001, 0.005, 0.01, 0.05,  0.1, 0.2, 0.3, 0.4, 0.5, 1, 2, 5, 10, 100]# список коэффициента регуляризации 1
for lambda_val in lambda1_values:
    # для каждого коэффициента регуляризации создаем модель Lasso
    lasso_reg = Lasso(lambda_val)
    # обучение на тренировочной выборке
    lasso_reg.fit(X_train, Y_train)
    # прогноз
    Y_pred__ = lasso_reg.predict(X_test)
    # средняя квадратичная ошибка
    mse_lasso = mean_squared_error(Y_pred__, Y_test)
    print(("Lasso MSE with Lambda={} is {}").format(lambda_val, mse_lasso))
```

Fig.26. LASSO method

The smallest error is at a coefficient equal to 0.000001

```
Lasso MSE with Lambda=1e-06 is 0.0027777247881661904
Lasso MSE with Lambda=0.0001 is 0.0028189634598784325
Lasso MSE with Lambda=0.001 is 0.003662658593756887
Lasso MSE with Lambda=0.005 is 0.005073825414868122
Lasso MSE with Lambda=0.01 is 0.005073825414868122
Lasso MSE with Lambda=0.05 is 0.005073825414868122
Lasso MSE with Lambda=0.1 is 0.005073825414868122
Lasso MSE with Lambda=0.2 is 0.005073825414868122
Lasso MSE with Lambda=0.3 is 0.005073825414868122
Lasso MSE with Lambda=0.4 is 0.005073825414868122
Lasso MSE with Lambda=0.5 is 0.005073825414868122
Lasso MSE with Lambda=1 is 0.005073825414868122
Lasso MSE with Lambda=2 is 0.005073825414868122
Lasso MSE with Lambda=5 is 0.005073825414868122
Lasso MSE with Lambda=10 is 0.005073825414868122
Lasso MSE with Lambda=100 is 0.005073825414868122
```

Figure 27. Errors for different coefficients

We will train the model on training data and predict on test data. We will derive the coefficients.

```
lasso_reg = Lasso(0.000001)
Y_pred_LAS = lasso_reg.fit(X_train, Y_train)
Y_pred_LAS = lasso_reg.predict(X_test)
```

```
#выведем полученные коэффициенты для наших признаков
lasso_coef = pd.DataFrame([X_train.columns, lasso_reg.coef_]).T
lasso_coef = lasso_coef.rename (columns = {0:"Attribute", 1 : "Coefficient"})
print(lasso_coef)
```

```
           Attribute Coefficient
0       Danceability   -0.003137
1             Energy   -0.028336
2           Loudness    0.041018
3        Speechiness    -0.01744
4       Acousticness    -0.02022
5   Instrumentalness   -0.010063
6           Liveness   -0.007526
7            Valence   -0.007837
8              Tempo    -0.00285
9        Duration_ms    -0.05727
10             Views    0.133214
11             Likes    1.199176
```

Figure 28. LASSO at L1 = 0.000001.

Let's look at the visualization and the remains for the presence of dependencies (Fig. 29-32).

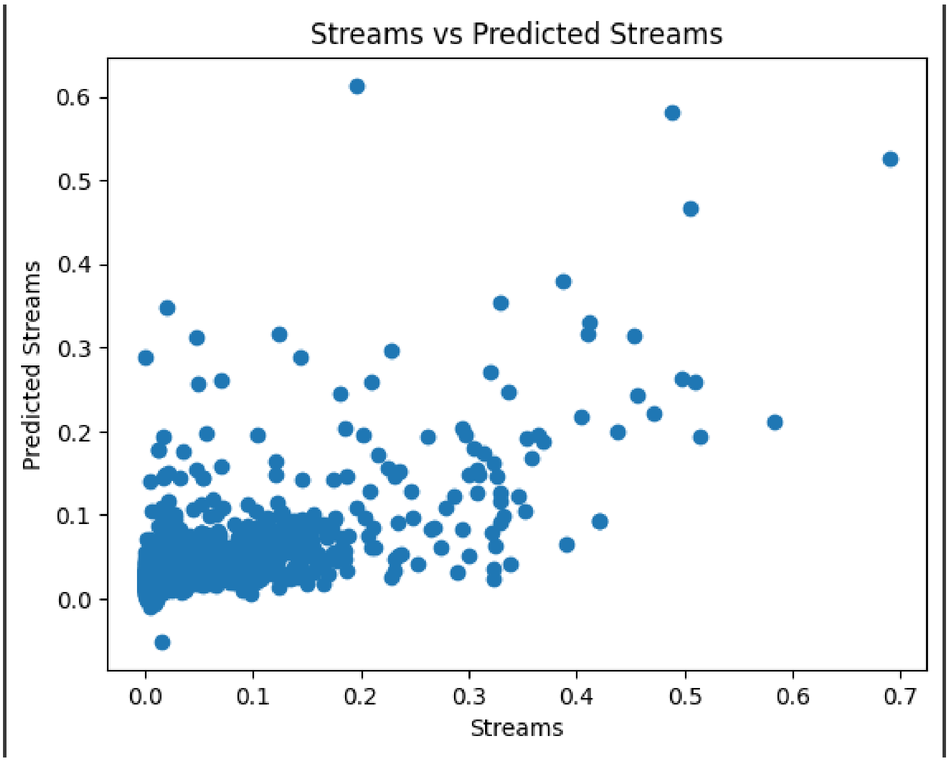


Figure 29. Visualization

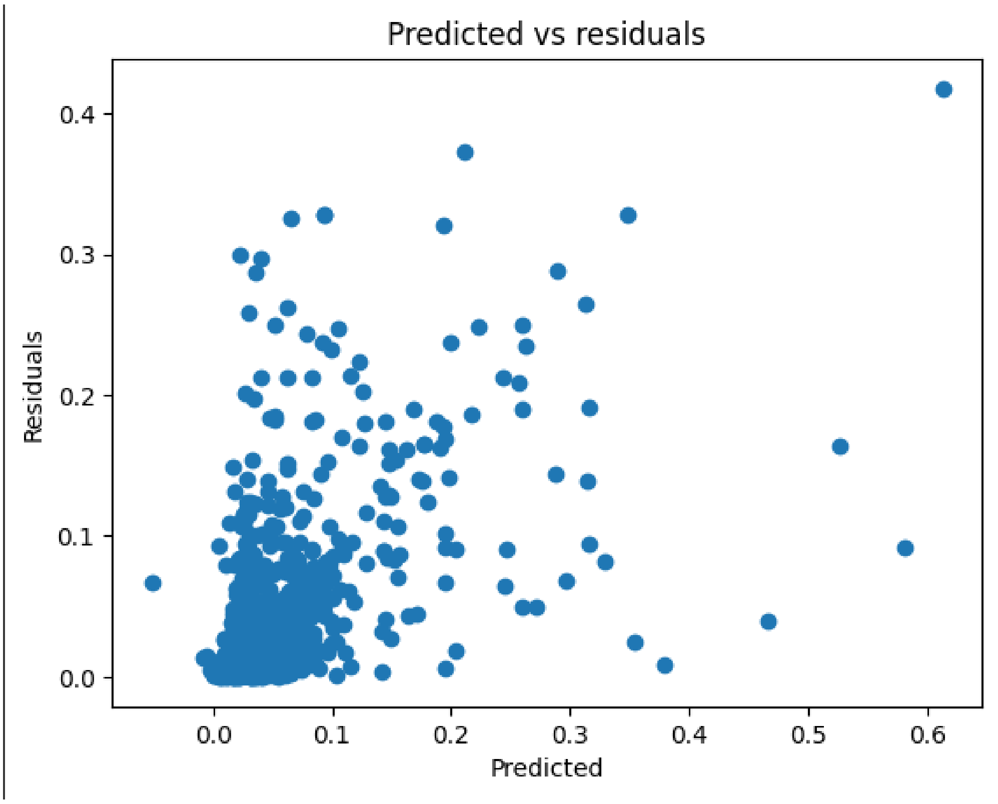


Figure 30. Remains

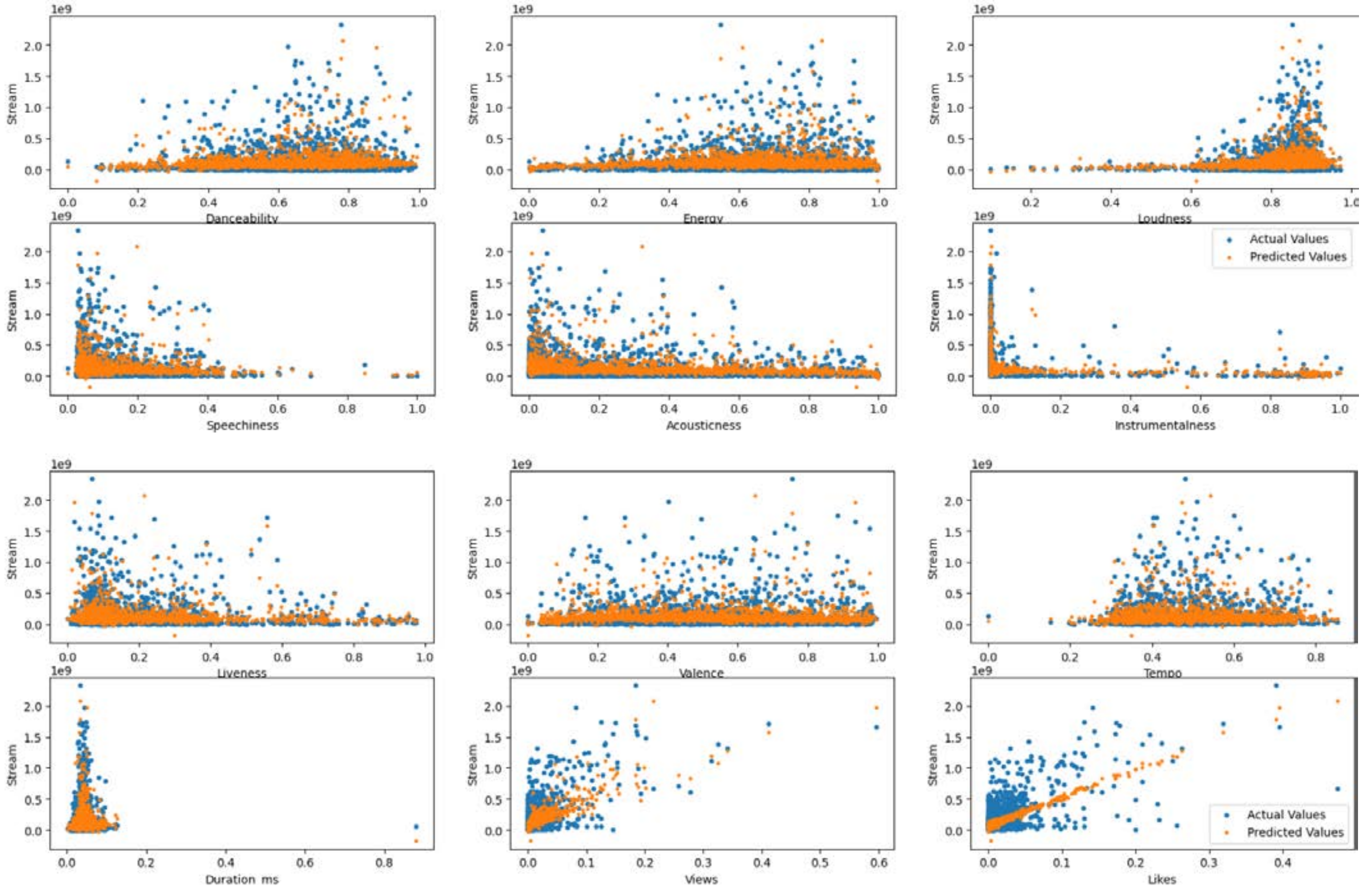


Figure 31. Predicted (orange) and actual (blue) values

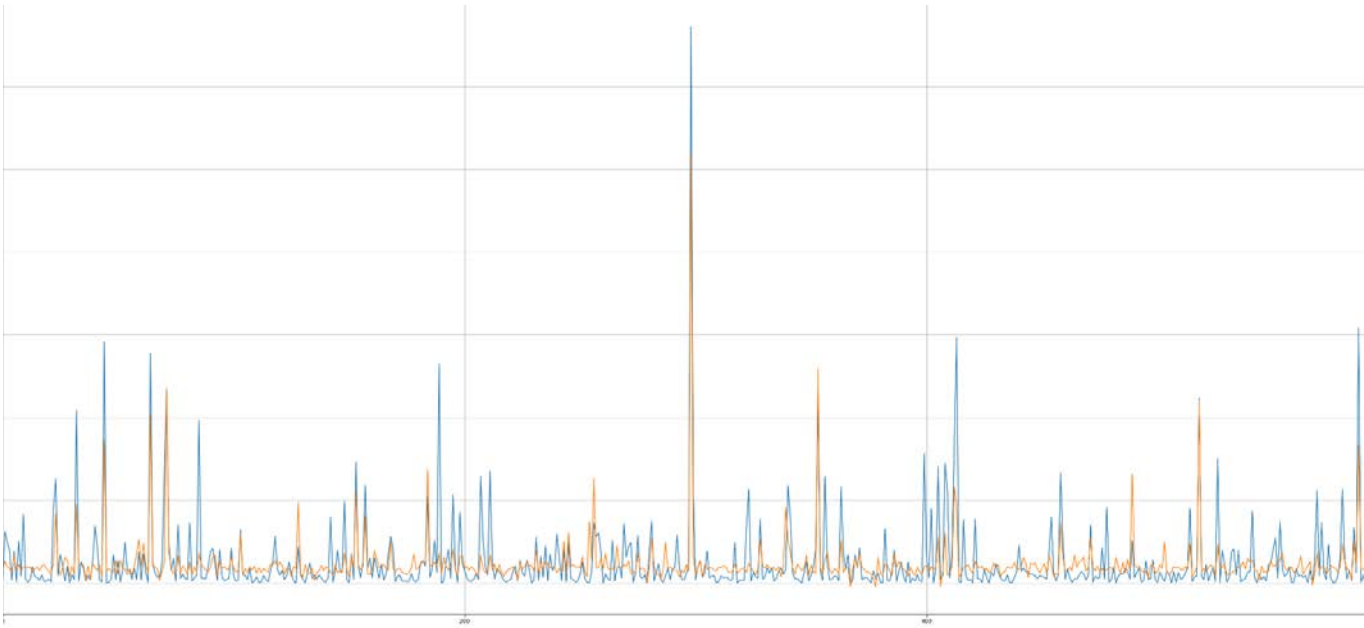

Figure 32. Predicted (orange) and actual (blue) values

No patterns were found, so we derive metrics

```
#оценка модели
R2_score = metrics.r2_score(Y_test, Y_pred_LAS)
print("LASSO REGRESSION METRICS")
print("R^2: ", R2_score)
print("MAE: ", metrics.mean_absolute_error(Y_test, Y_pred_LAS))
print("MSE: ", metrics.mean_squared_error(Y_test, Y_pred_LAS))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y_test, Y_pred_LAS)))
```

```
LASSO REGRESSION METRICS
R^2:  0.45253146798296395
MAE:  0.030928614327273844
MSE:  0.0027777247881661904
RMSE:  0.05270412496348071
```

Figure 33. Metrics

## 6. RIDGE

A model that penalizes the L2 norm of a vector. "Try to achieve the best performance, but none of the coefficients should reach an extreme value."

We will iterate over various regularization coefficients and create a RIDGE model for each one. The best coefficient has the least squared error.

```python
from sklearn.linear_model import Ridge

lambda2_values = [0.000001, 0.0001, 0.001, 0.005, 0.01, 0.05,  0.1, 0.2, 0.3, 0.4, 0.5, 1, 2, 2.13, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 10, 100]# список коэффициента регуляризации 2


for lambda_val in lambda2_values:
    # для каждого коэффициента регуляризации создаем модель Ridge
    ridge_reg = Ridge(lambda_val)

    ridge_reg.fit(X_train, Y_train)
    y_pred = ridge_reg.predict(X_test)
    mse_ridge = mean_squared_error(y_pred, Y_test)
    print(("Lasso MSE with Lambda={} is {}").format(lambda_val, mse_ridge))
```

Fig.34. RIDGE method

The smallest error is at a coefficient equal to 2

```
Lasso MSE with Lambda=1e-06 is 0.002777614924211195
Lasso MSE with Lambda=0.0001 is 0.0027776092775756920
Lasso MSE with Lambda=0.001 is 0.0027775579841161175
Lasso MSE with Lambda=0.005 is 0.002777330873953267
Lasso MSE with Lambda=0.01 is 0.002777048951487078
Lasso MSE with Lambda=0.05 is 0.002774870057547550 5
Lasso MSE with Lambda=0.1 is 0.002772327886122870 6
Lasso MSE with Lambda=0.2 is 0.002767789482871226 6
Lasso MSE with Lambda=0.3 is 0.002763889803585516
Lasso MSE with Lambda=0.4 is 0.002760538760259591 3
Lasso MSE with Lambda=0.5 is 0.002757661966146236
Lasso MSE with Lambda=1 is 0.0027485408473949184
Lasso MSE with Lambda=2 is 0.0027451206766844235
Lasso MSE with Lambda=2.13 is 0.002745502405200136
Lasso MSE with Lambda=2.4 is 0.002746694336356767
Lasso MSE with Lambda=2.5 is 0.002747256401602717
Lasso MSE with Lambda=2.6 is 0.0027478766512970348
Lasso MSE with Lambda=2.7 is 0.0027485512106303514
Lasso MSE with Lambda=2.8 is 0.0027492765607544266
Lasso MSE with Lambda=2.9 is 0.002750049498685065
Lasso MSE with Lambda=10 is 0.0028535860945361497
Lasso MSE with Lambda=100 is 0.0039058524046915203
```

Figure 35. Errors for different coefficients

We will train the model on training data and predict on test data. We will derive the coefficients.

```
ridge_reg = Ridge(2)
Y_pred_Ridge = ridge_reg.fit(X_train, Y_train)
Y_pred_Ridge = ridge_reg.predict(X_test)
#выведем полученные коэффициенты для наших признаков
ridge_coef = pd.DataFrame([X_train.columns, ridge_reg.coef_]).T
ridge_coef = ridge_coef.rename (columns = {0:"Attribute", 1 : "Coefficient"})
print(ridge_coef)
```

```
           Attribute Coefficient
0       Danceability   -0.001284
1             Energy   -0.028754
2           Loudness    0.043755
3        Speechiness   -0.016297
4       Acousticness   -0.020573
5   Instrumentalness   -0.010116
6           Liveness   -0.007966
7            Valence   -0.009355
8              Tempo   -0.002518
9        Duration_ms   -0.053926
10             Views    0.349093
11             Likes    0.929212
```

Figure 36. RIDGE at L2 = 2.

Let's look at the visualization and the remains for the presence of dependencies (Fig. 32-33).

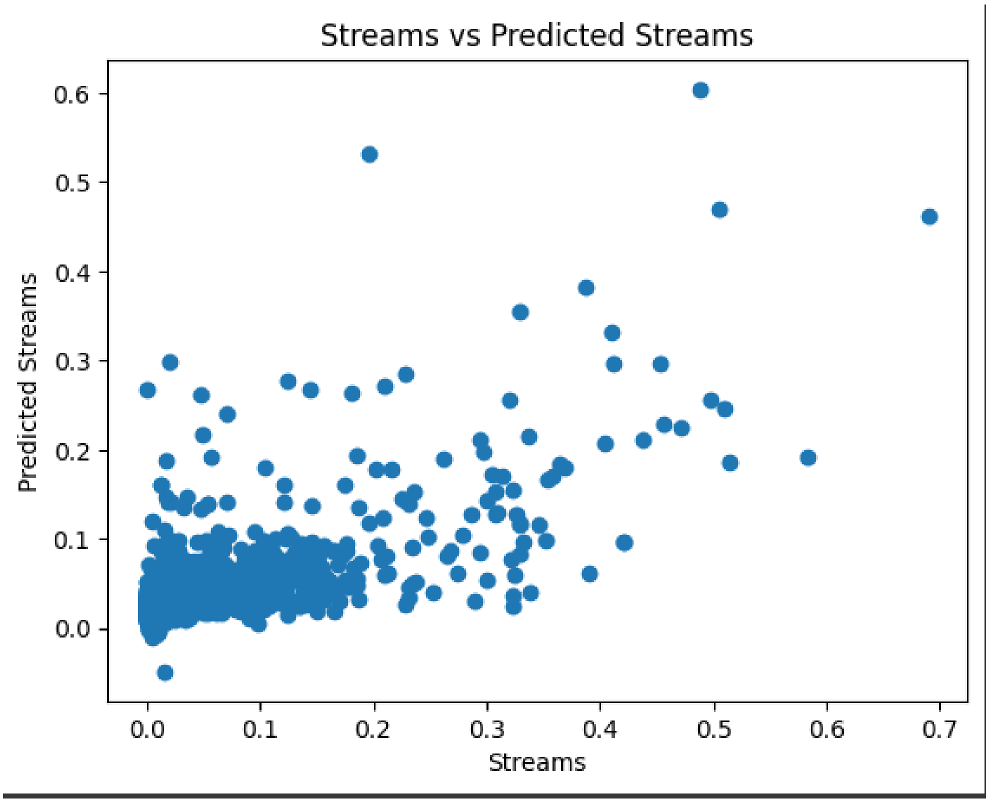


Figure 37. Visualization

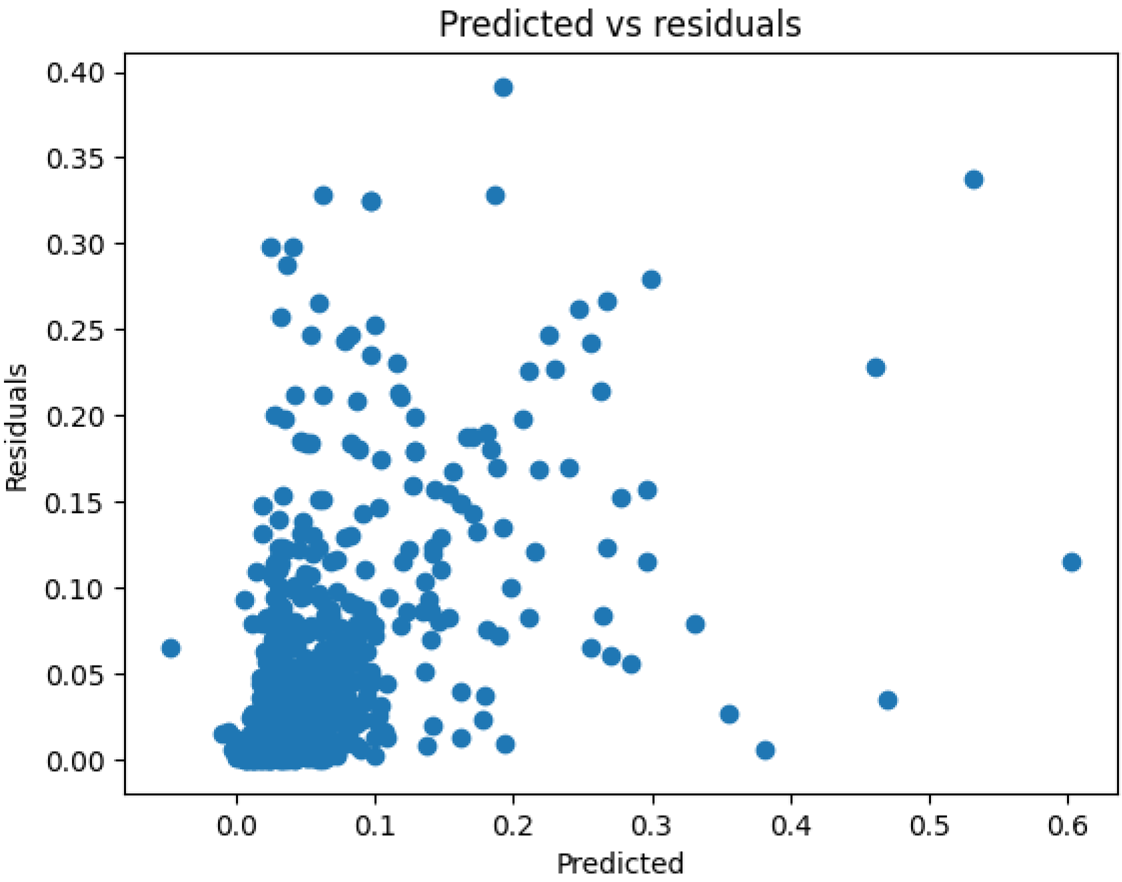


Figure 38. Remains

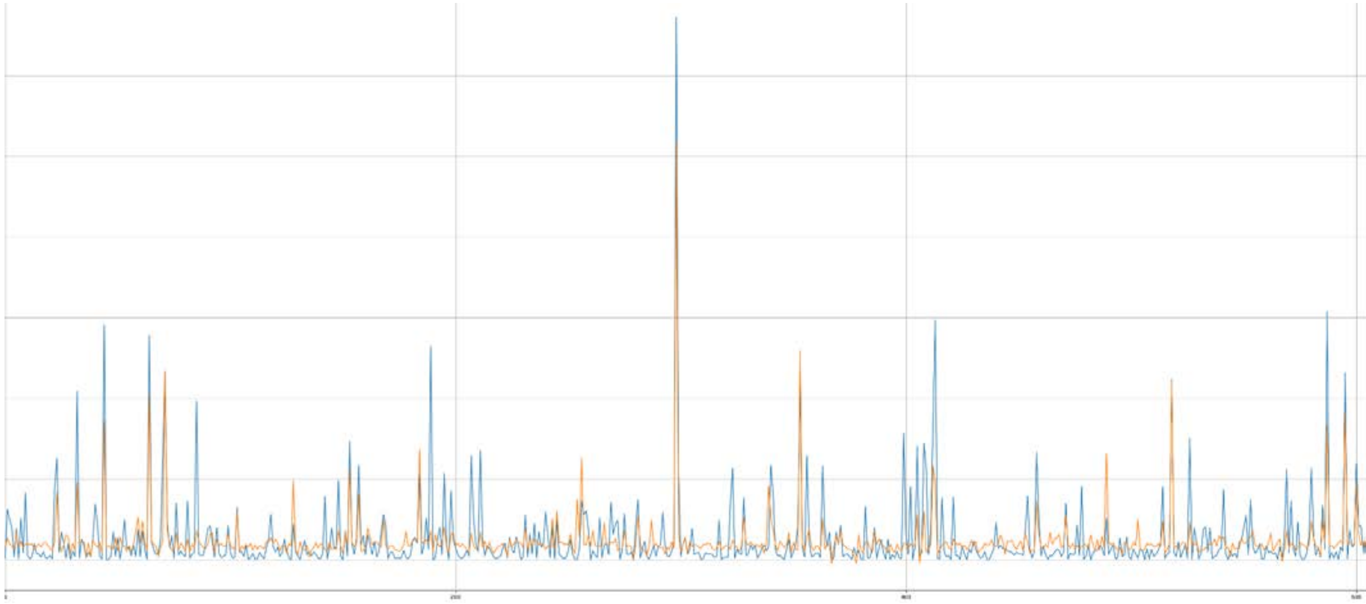

Figure 39. Predicted (orange) and actual (blue) values

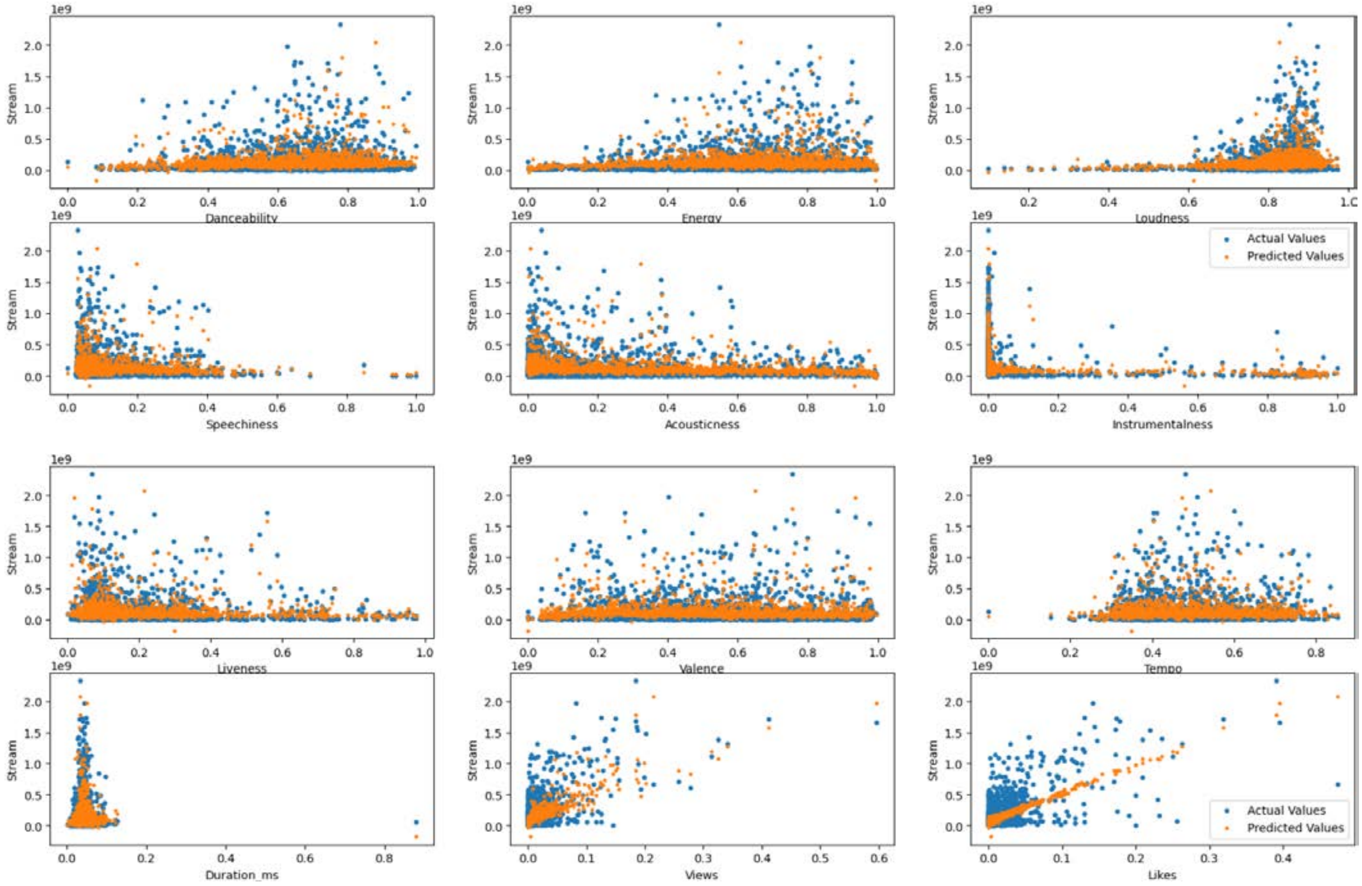


Figure 40. Predicted (orange) and actual (blue) values

No patterns were found, so we derive metrics

```
#оценка модели
R2_score = metrics.r2_score(Y_test, Y_pred_Ridge)
print("RIDGE REGRESSION METRICS")
print("R^2: ", R2_score)
print("MAE: ", metrics.mean_absolute_error(Y_test, Y_pred_Ridge))
print("MSE: ", metrics.mean_squared_error(Y_test, Y_pred_Ridge))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y_test, Y_pred_Ridge)))
```

```
RIDGE REGRESSION METRICS
R^2:  0.4589574915856933
MAE:  0.031171846688285026
MSE:  0.0027451206766844235
RMSE:  0.052393899231536714
```

Figure 41. Metrics

## 6. Metrics

```
#вывод всех оценок моделей
R2_score = metrics.r2_score(Y_test, Y_test_pred)
print("LINEAR REGRESSION METRICS")
print("R^2: ", R2_score)
print("MAE: ", metrics.mean_absolute_error(Y_test, Y_test_pred))
print("MSE: ", metrics.mean_squared_error(Y_test, Y_test_pred))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y_test, Y_test_pred)))

R2_score = metrics.r2_score(Y_test, Y_pred_LAS)
print("LASSO REGRESSION METRICS")
print("R^2: ", R2_score)
print("MAE: ", metrics.mean_absolute_error(Y_test, Y_pred_LAS))
print("MSE: ", metrics.mean_squared_error(Y_test, Y_pred_LAS))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y_test, Y_pred_LAS)))

R2_score = metrics.r2_score(Y_test, Y_pred_Ridge)
print("RIDGE REGRESSION METRICS")
print("R^2: ", R2_score)
print("MAE: ", metrics.mean_absolute_error(Y_test, Y_pred_Ridge))
print("MSE: ", metrics.mean_squared_error(Y_test, Y_pred_Ridge))
print("RMSE: ", np.sqrt(metrics.mean_squared_error(Y_test, Y_pred_Ridge)))
```

```
LINEAR REGRESSION METRICS
R^2:  0.45255311009444277
MAE:  0.030927330555887526
MSE:  0.002777614981253085
RMSE:  0.05270308322340435
LASSO REGRESSION METRICS
R^2:  0.45253146798296395
MAE:  0.030928614327273844
MSE:  0.0027777247881661904
RMSE:  0.05270412496348071
RIDGE REGRESSION METRICS
R^2:  0.4589574915856933
MAE:  0.031171846688285026
MSE:  0.0027451206766844235
RMSE:  0.052393899231536714
```

Figure 42. All metrics

## 7. Selection of parameters

Let's try to find the "best" parameters for RIDGE and LASSO models.

```
from sklearn.linear_model import RidgeCV

#Lasso Cross validation
ridge_cv = RidgeCV(alphas = [0.000001, 0.0001, 0.001, 0.005, 0.01, 0.05,  0.1, 0.2, 0.3, 0.4, 0.5, 1, 2, 3, 100]).fit(X_train, Y_train)
y_pred_ridgeCV = ridge_cv.predict(X_test)
#выведем результаты
reg_score_lr = ridge_cv.score(X_test, Y_test)
print("The test score for ridge model is {}".format(reg_score_lr))
```

```
The test score for ridge model is 0.45448964145941706
```

```
from sklearn.linear_model import LassoCV

#Lasso Cross validation
lasso_cv = LassoCV(alphas = [0.0000000000000000001, 0.0000000000001, 0.00000000001, 0.000000001, 0.0000001, 0.00001, 0.000001, 0.0001, 0.001, 0.0015, 0.0002, 0.003, 0.004, 0.005, 0.01, 0.02, 0
y_pred_lassoCV = lasso_cv.predict(X_test)
#выведем результаты
reg_score_lasso = lasso_cv.score(X_test, Y_test)
print("The test score for ridge model is {}".format(reg_score_lasso))
```

```
The test score for ridge model is 0.45448964145941706
```

Figure 36. Parameter enumeration for RIDGE and LASSO

As a result, the values of the RIDGE model were slightly increased, but LASSO did not succeed in selecting better parameters.

**Conclusion:** I acquired and consolidated the skills of data preprocessing and applying machine learning methods to solve regression problems.

[Link to code]