به نام خدا

گزارش پروژه

درس داده‌کاوی

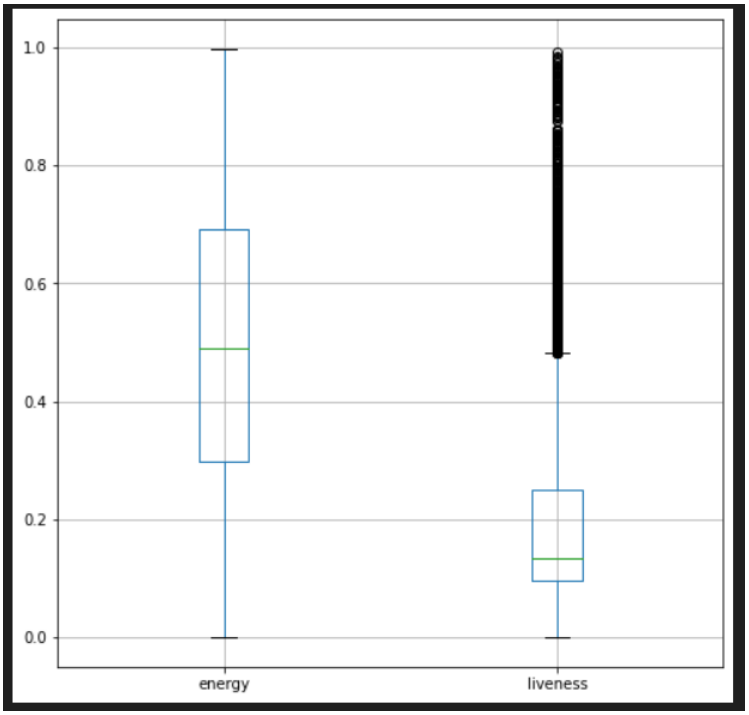
نگارنده: زهرا رحیمی

استاد درس: دکتر ناظرفرد

بهار 1402

Box plot **انرژی و** liveness **را باهم مقایسه کنید**

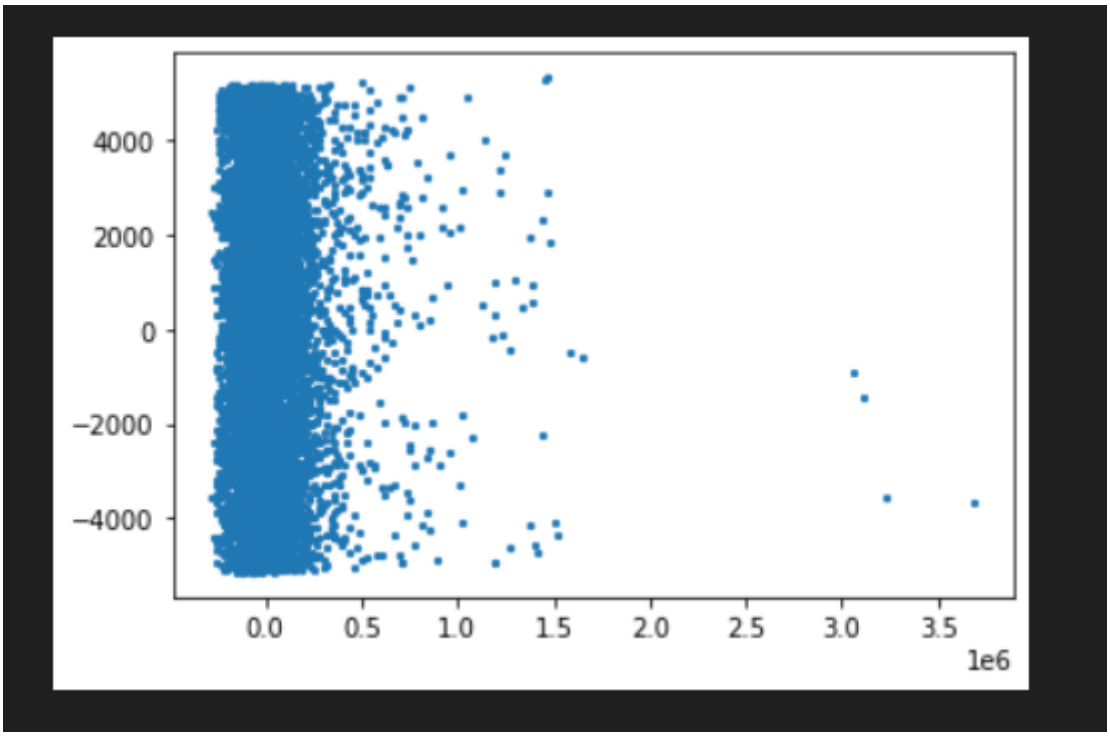
از کران های بالا و پایین متوجه می‌شویم که هر دو مقادیر 0 تا 1 ر ا فقط شامل می‌شوند. اما ویژگی انرژی فراوانی بیشتری در مقادیر وسط یعنی بازه 0.3 تا 0.7 با میانه 0.5 دارد. اما مقادیر liveness بیشتر در همان ابتدا بازه یعنی 0.1 تا 0.3 فراوانی بیشتری دارند. همچنین برای liveness مقادیر بیشتر از 0.5 داده پرت محسوب می‌شود.



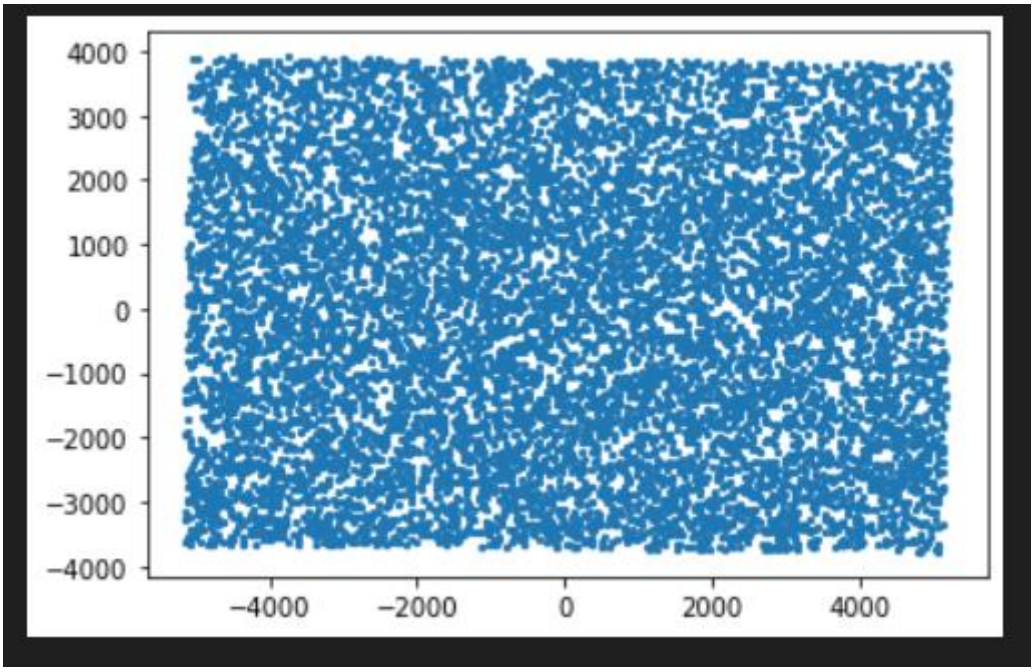
برای بقیه فیچرها نیز می توان با همین خط کد نمودار box plot شان را کشید.

نمایش فیچرهای دیتاست

اگر داده‌های عددی را normalize نکنیم به این نمودار دست پیدا خواهیم کرد:



اما اگر آنها را normalize کنیم نتیجه به صورت دیگری است، همانطور که مشخص است توزیع داده به صورت یکنواختی انجام شده است:



انتخاب فیچر

در ابتدا حتی ویژگی‌هایی که شاید در محبوبیت تاثیر زیادی نداشتند شامل:

```
regression_features = [
    # Your desired features columns
    'track_id','duration_ms', 'explicit', 'track_name',
      'artist_name',
      'album_release_year',
      'danceability', 'energy', 'key', 'loudness', 'mode',
      'speechiness', 'acousticness', 'instrumentalness', 'liveness',
      'valence', 'tempo', 'time_signature'
]
```

هم انتخاب کردم و نتیجه به این صورت شد:

```
#autograde
#Checking the accuracy of Linear Regression
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

✓ 0.0s

```
Mean Absolute Error: 0.7123365641524408
Mean Squared Error: 7.633834822532622
Root Mean Squared Error: 2.7629395256741724
```

اما از آنجا که می دانیم اضافه کردن هر فیچر برابر با بار محاسباتی بیشتر با کم و زیاد کردن فیچرهای مختلف در نهایت به این فیچرها رسیدیم:

```
regression_features = [
    # Your desired features columns
    'duration_ms',
    'explicit',
    'track_name',
    'artist_name',
    'album_release_year',
    'danceability',
    'loudness',
    'mode',
    'instrumentalness',
    'valence',
    'tempo',
]
```

که نتیجه f1 score آن برابر 8.6 شد:

```
Mean Absolute Error: 1.3352875997584992
Mean Squared Error: 8.611120981564527
Root Mean Squared Error: 2.9344711587549344
```

برای دسته بندی نیز فیچرهای‌مان همان فیچرهای رگرسیون است بجز artist_name زیرا مدل قرار است همین را پیش‌بینی کند:

```
'duration_ms',
'album_name',
'explicit',
'track_name',
'album_release_year',
'danceability',
'loudness',
'mode',
'instrumentalness',
'valence',
'tempo',
'is_sonnati'
```

انتخاب مدل برای classification

از بین الگوریتم‌های متفاوتی که برای classification وجود دارد نظیر Logistic Regression, Decision Tree, Random Forest, SVM, KNN, Naïve Bayes کاربردهای مختلف آنها را بررسی و به این نتیجه رسیدیم که برای دسته‌بندی ژانر موسیقی مدل Random Forest احتمالا نتایج بهتری به ما نشان دهد. نتایج آن را در زیر می‌بینید:

```
{'acc': 0.9420062695924765,
 'confusion': array([[2011,   84],
       [ 101,  994]], dtype=int64),
 'f1': 0.9148642429820525,
 'precision': 0.922077922077922,
 'recall': 0.9077625570776255}
```

از جمله دلایلی که random forest عملکرد خوبی داشته است می تواند این باشد که طبقه‌بندی ژانر موسیقی (در اینجا سنتی) مستلزم یادگیری روابط غیر خطی است، که random forest به خوبی با استفاده از درخت های تصمیم می تواند روابط

موجود در داده های موسیقی را مدل سازی کرده و نتیجه را بهبود بخشد. علاوه بر این random forest قابلیت تعمیم خوبی دارد و با سوگیری کمی نسبت به داده های دیده شده، می تواند داده های آزمایش را به خوبی طبقه بندی کند.