MUSIC GENRE CLASSIFICATION

SHAI For AI | شاي للذكاء الاصطناعي

TABLE OF CONTENT

Exploring dataset

The first thing to know about the dataset is that it contains two types of columns.

The first type consists of numerical columns, which include:

- Popularity
- Danceability
- Energy
- Key
- Loudness
- Mode
- Speechiness
- Acousticness
- Instrumentalness
- Liveness
- Valence
- Tempo
- Duration (in minutes/milliseconds)
- Time signature
- Class

On the other hand, the second type of columns is categorical and includes:

- Artist name
- Track name

Furthermore, the dataset includes a target variable called 'Class' which consists of 11 types:

- Rock
- Indie
- Alt
- Pop
- Metal
- HipHop
- Alt_Music
- Blues
- Acoustic/Folk
- Instrumental
- Country
- Bollywood

Column description:

	Id	Popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_in min/ms	time_signature	Class
count	14396.000000	14063.000000	14396.000000	14396.000000	12787.000000	14396.000000	14396.000000	14396.000000	14396.000000	10855.000000	14396.000000	14396.000000	14396.000000	1.439600e+04	14396.000000	14396.000000
mean	7198.500000	44.525208	0.543105	0.662422	5.953781	-7.900852	0.640247	0.080181	0.246746	0.178129	0.195782	0.486379	122.695372	2.000942e+05	3.924354	6.695679
std	4155.911573	17.418940	0.165517	0.235967	3.200013	4.057362	0.479944	0.085157	0.310922	0.304266	0.159258	0.239476	29.538490	1.116891e+05	0.359520	3.206170
min	1.000000	1.000000	0.059600	0.001210	1.000000	-39.952000	0.000000	0.022500	0.000000	0.000001	0.011900	0.021500	30.557000	5.016500e-01	1.000000	0.000000
25%	3599.750000	33.000000	0.432000	0.508000	3.000000	-9.538000	0.000000	0.034800	0.004280	0.000088	0.097275	0.299000	99.799000	1.654458e+05	4.000000	5.000000
50%	7198.500000	44.000000	0.545000	0.699000	6.000000	-7.013500	1.000000	0.047100	0.081450	0.003920	0.129000	0.480500	120.060000	2.089410e+05	4.000000	8.000000
75%	10797.250000	56.000000	0.658000	0.861000	9.000000	-5.162000	1.000000	0.083100	0.432250	0.201000	0.256000	0.672000	141.988250	2.522470e+05	4.000000	10.000000
max	14396.000000	100.000000	0.989000	1.000000	11.000000	1.342000	1.000000	0.955000	0.996000	0.996000	0.992000	0.986000	217.416000	1.477187e+06	5.000000	10.000000

The dataset doesn't contain duplicate rows:

```
[ ] train.duplicated().sum()

    0
```

Info about dataset:

It shows the different data types we will deal with, also it shows that it contains null values.

```
[ ]  train.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 14396 entries, 0 to 14395
     Data columns (total 18 columns):
      #   Column              Non-Null Count  Dtype
     ---  ------              --------------  -----
      0   Id                  14396 non-null  int64
      1   Artist Name         14396 non-null  object
      2   Track Name          14396 non-null  object
      3   Popularity          14063 non-null  float64
      4   danceability        14396 non-null  float64
      5   energy              14396 non-null  float64
      6   key                 12787 non-null  float64
      7   loudness            14396 non-null  float64
      8   mode                14396 non-null  int64
      9   speechiness         14396 non-null  float64
      10  acousticness        14396 non-null  float64
      11  instrumentalness    10855 non-null  float64
      12  liveness            14396 non-null  float64
      13  valence             14396 non-null  float64
      14  tempo               14396 non-null  float64
      15  duration_in min/ms  14396 non-null  float64
      16  time_signature      14396 non-null  int64
      17  Class               14396 non-null  int64
     dtypes: float64(12), int64(4), object(2)
     memory usage: 2.0+ MB
```

Exploring dataset

Null values exist in there columns “popularity, key and instrumentalness”

```
[ ] train.isna().sum()

Id                    0
Artist Name           0
Track Name            0
Popularity          333
danceability          0
energy                0
key                1609
loudness              0
mode                  0
speechiness           0
acousticness          0
instrumentalness   3541
liveness              0
valence               0
tempo                 0
duration_in min/ms    0
time_signature        0
Class                 0
dtype: int64
```

The distribution of classes in dataset:

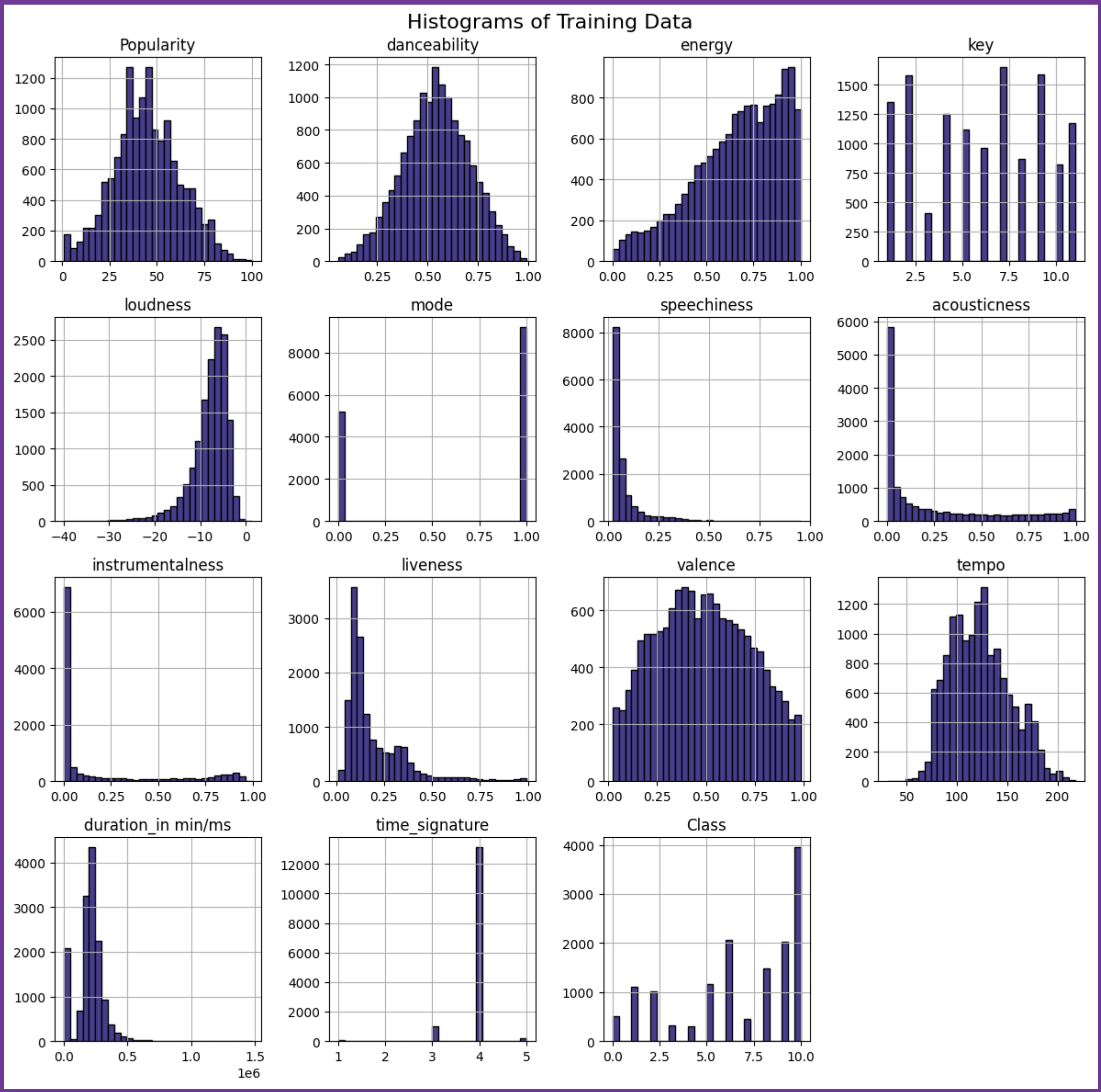
```
[ ] train['Class'].value_counts()

10    3959
6     2069
9     2019
8     1483
5     1157
1     1098
2     1018
0      500
7      461
3      322
4      310
Name: Class, dtype: int64
```

Visualization

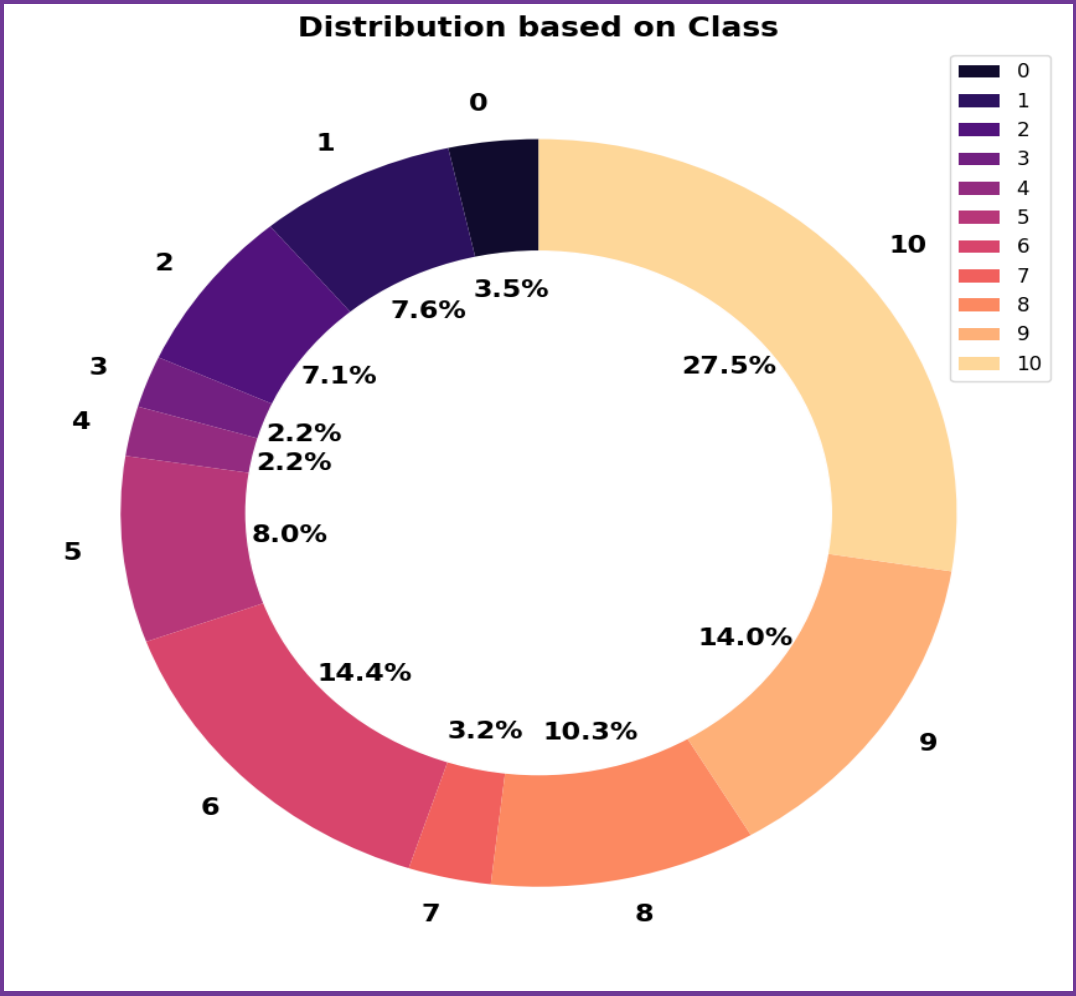
Explore the data further through visualizations and plots:

Columns Histogram:



Visualization

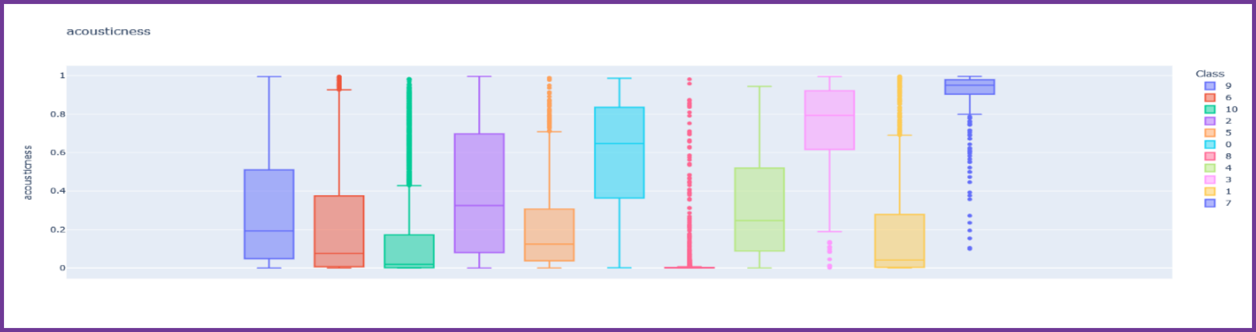
Class column distribution:

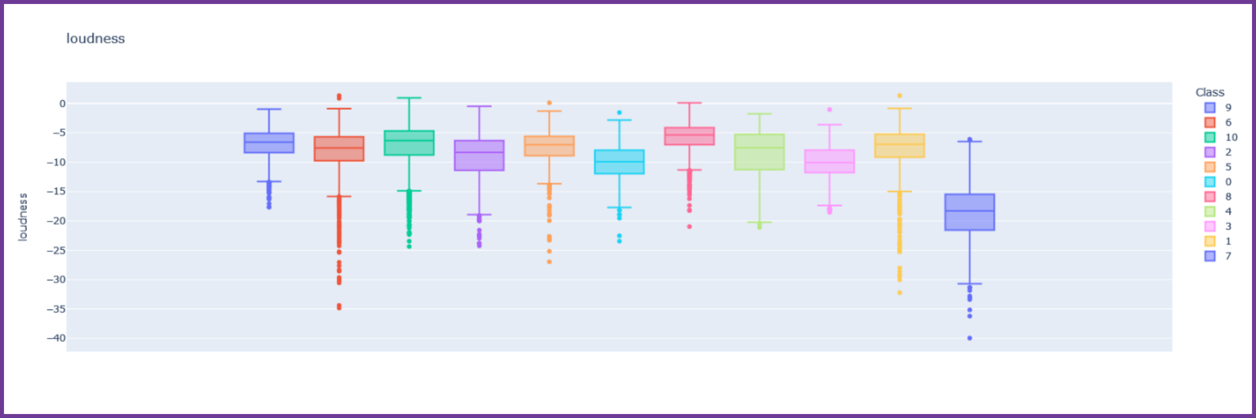


Visualization

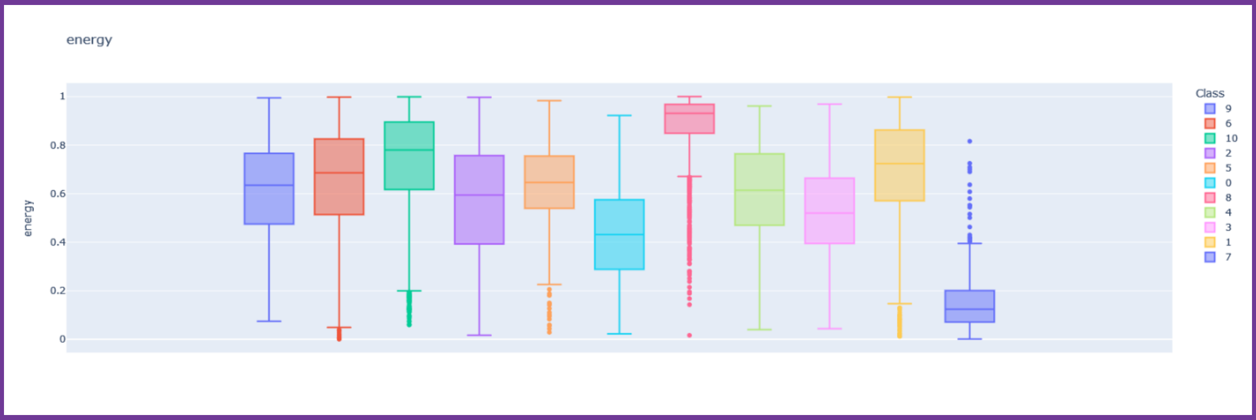
Box plot:

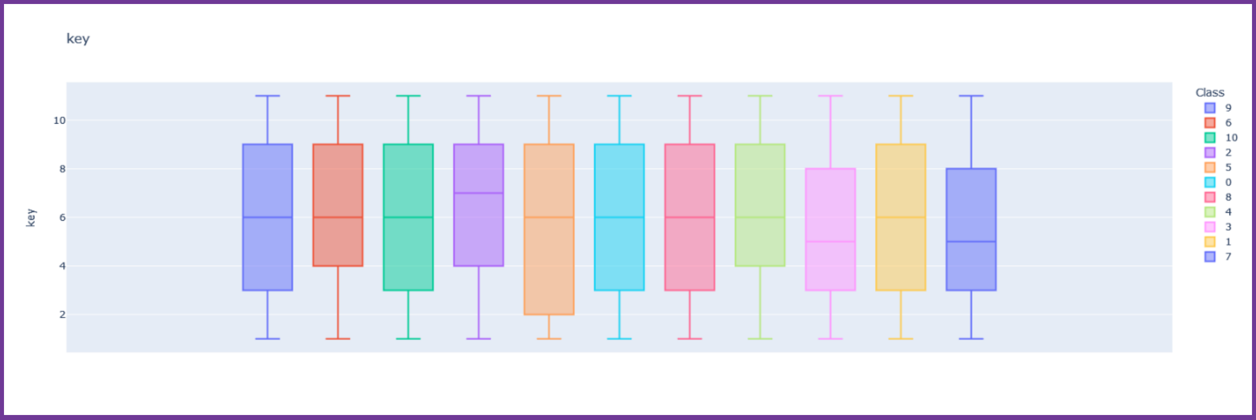
Used a code to create box plots for each numerical column in the training dataset, with the "Class" variable determining the color of the boxes and the column name as the title

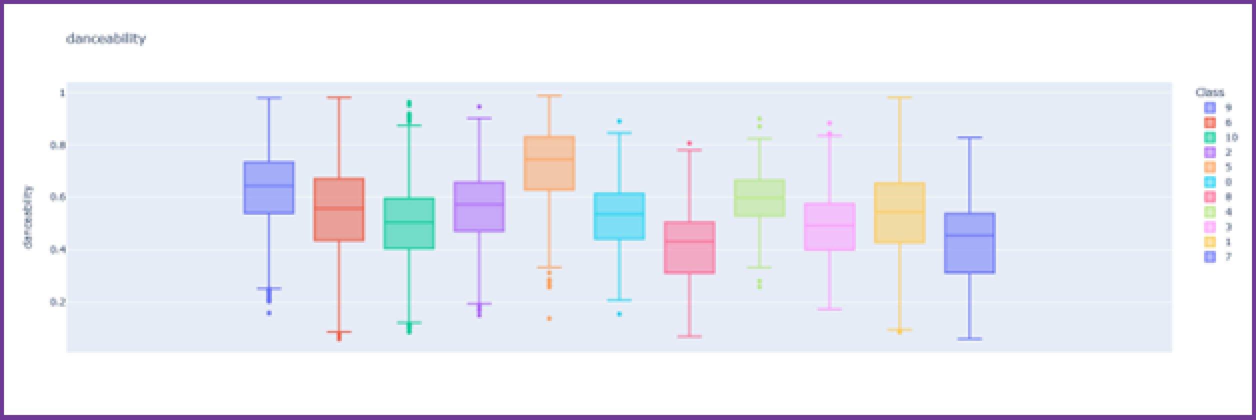




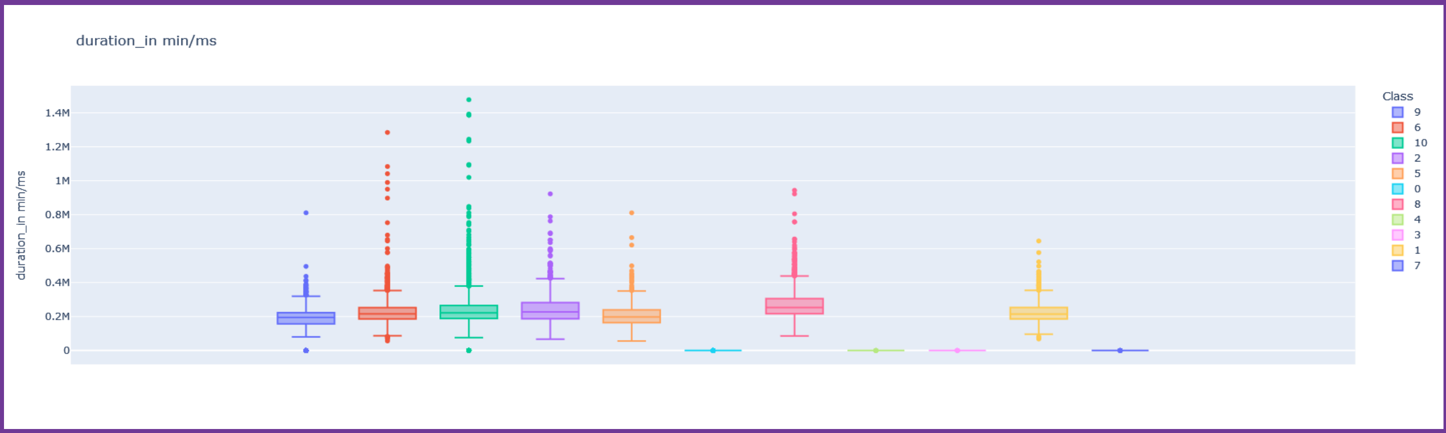
Box plot:

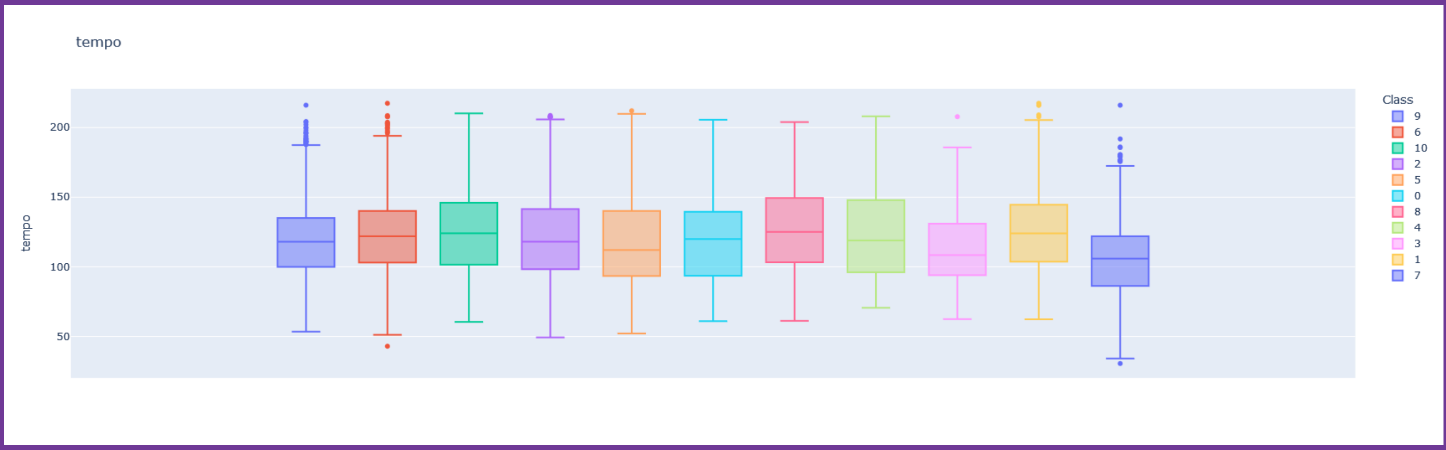

energy
energy
1
0.8
0.6
0.4
0.2
0
Class
9
6
10
2
5
0
8
4
3
1
7

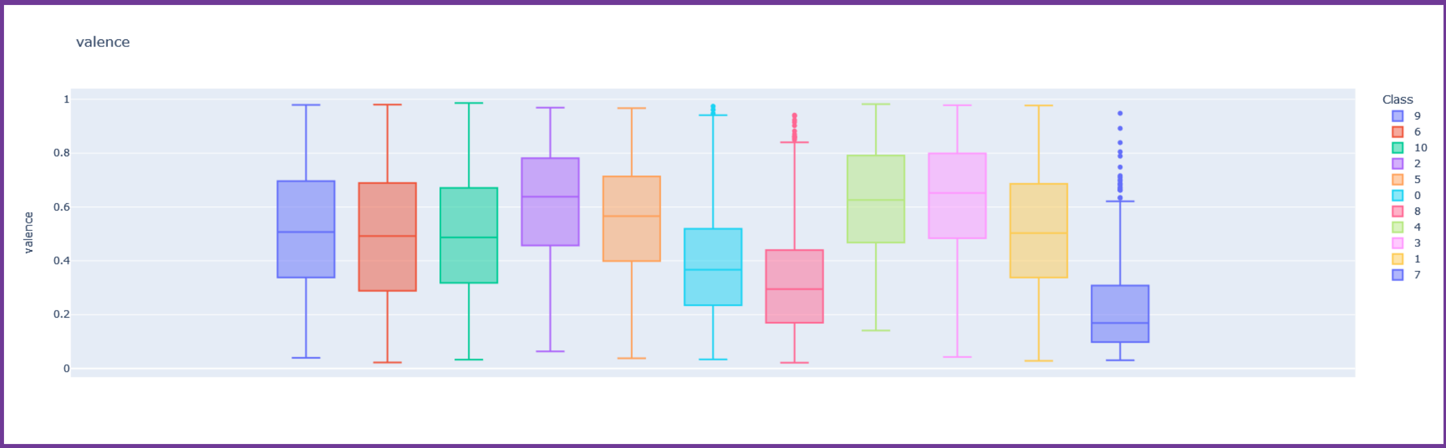

key
key
10
8
6
4
2
Class
9
6
10
2
5
0
8
4
3
1
7


danceability
1
0.8
0.6
0.4
0.2
Class

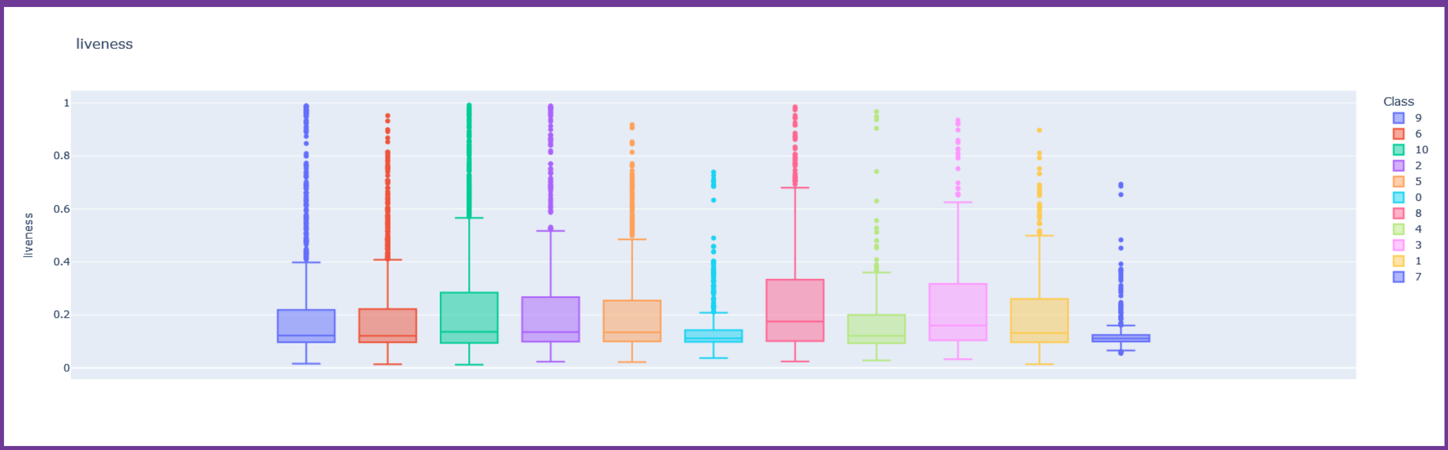
Box plot:


duration_in min/ms
duration_in min/ms
0
0.2M
0.4M
0.6M
0.8M
1M
1.2M
1.4M
Class
9
6
10
2
5
0
8
4
3
1
7

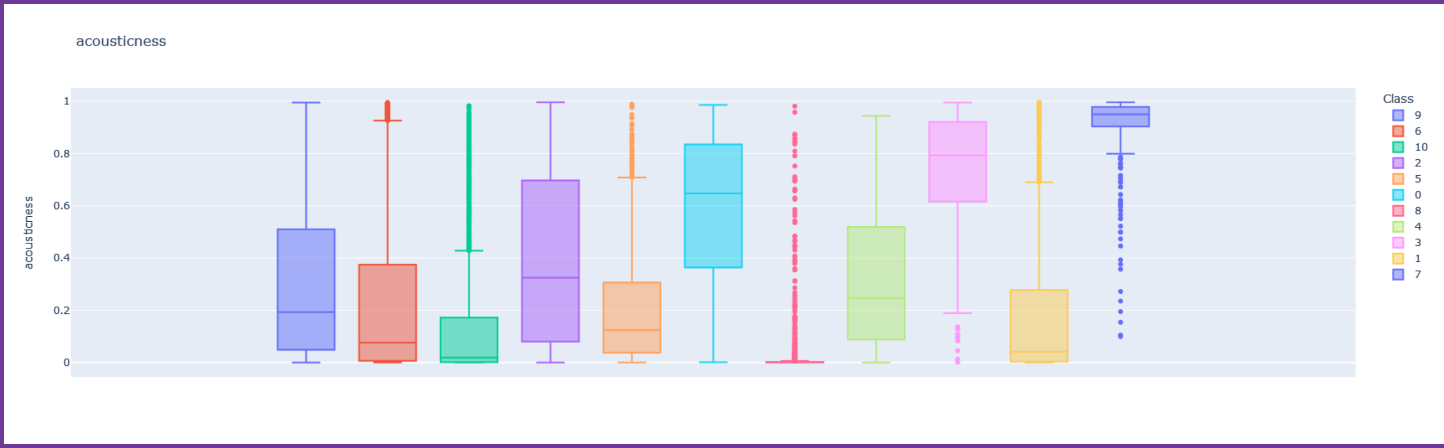

tempo
tempo
50
100
150
200
Class
9
6
10
2
5
0
8
4
3
1
7


valence
valence
0
0.2
0.4
0.6
0.8
1
Class
9
6
10
2
5
0
8
4
3
1
7

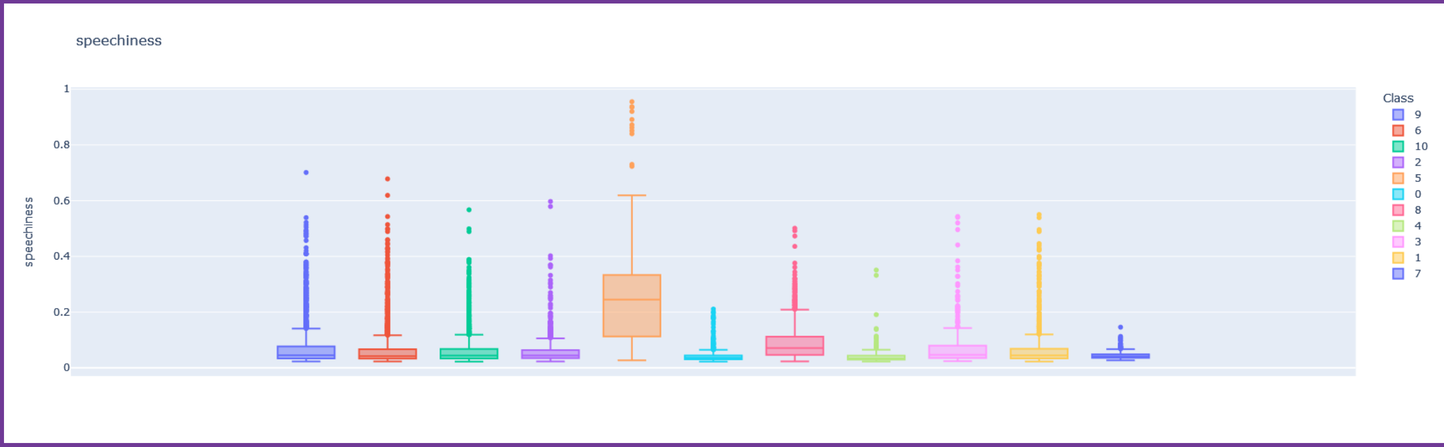
Box plot:



liveness
liveness
0
0.2
0.4
0.6
0.8
1
Class
9
6
10
2
5
0
8
4
3
1
7



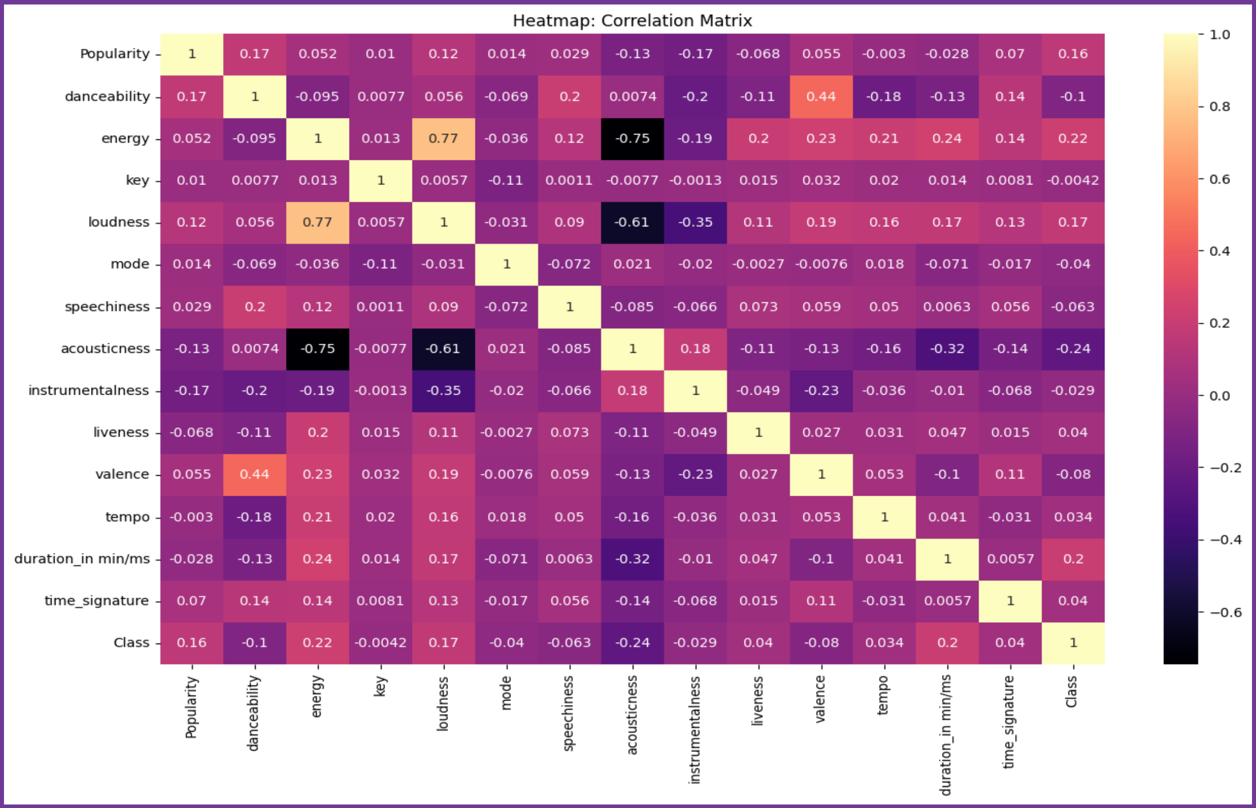
acousticness
acousticness
0
0.2
0.4
0.6
0.8
1
Class
9
6
10
2
5
0
8
4
3
1
7



speechiness
speechiness
0
0.2
0.4
0.6
0.8
1
Class
9
6
10
2
5
0
8
4
3
1
7

Heatmap plot:

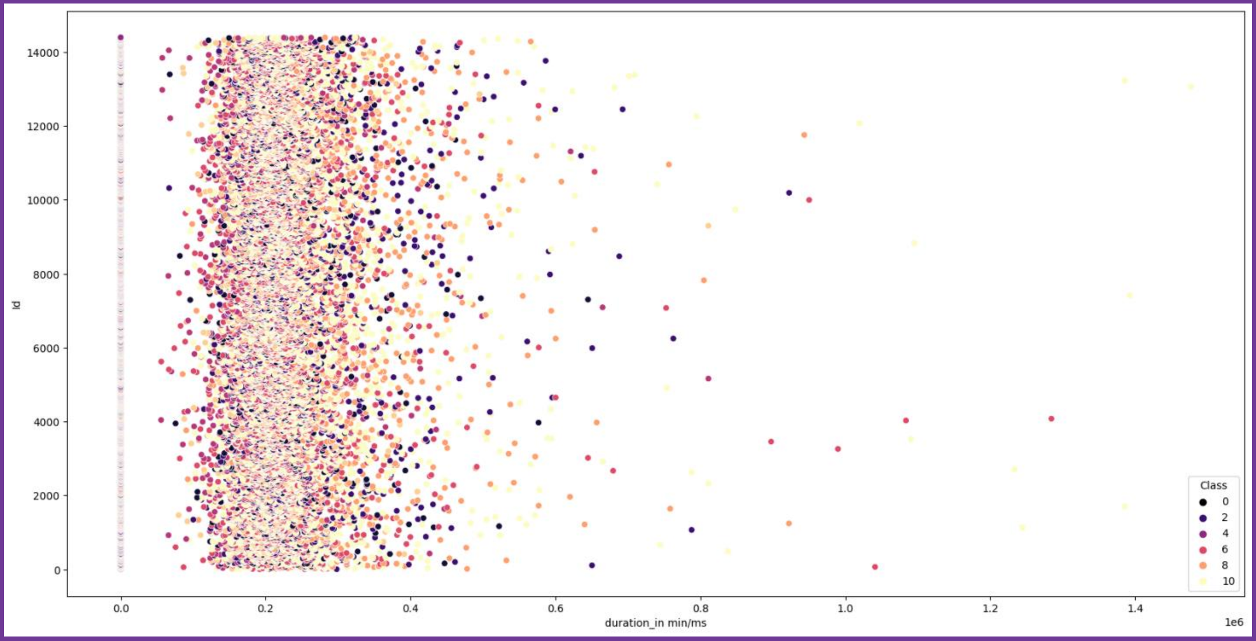
Preview the correlation matrix between variables in the dataset providing a visual representation of the relationships between variables.



Visualization

Duration scatter plot:

Use scatterplot to visualize the relationship between the 'duration_in min/ms' variable and the 'Id' variable in the dataset, with different data points colored by the 'Class' variable.



Preprocessing

Performing data cleaning and applying various transformations to the data.

During this step, various types of features were created to enhance model accuracy. Throughout the process of building and refining the model, multiple approaches were employed, and the results were compared. The iterative nature of the process involved trying different methods until the final update of the code, which yielded the best output among all the models. I will now explain some of the approaches that were used and provide an analysis of why they either failed or succeeded.

Initially, a preview of the data was utilized for data cleaning and feature engineering purposes across all versions of the code. Ultimately, the approach that worked best for me was implemented in the final version:

Data cleaning:

For these first few codes I dropped the categorical data but then decided to use them with catboost classifies since it gave a better score:

```
train_set.drop(columns=["Id", "Track Name", "Artist Name"],
inplace=True)
```

For removing null values:

First time: tried to use mean value for to fill the place of missing values:

```
imputer = SimpleImputer(strategy="mean")

X_train_imputed = imputer.fit_transform(X_train)

X_test_imputed = imputer.transform(X_test)
```

Second try done for each column separately used the mean with “popularity”, mode with “key” and, zero for “instrumentalness” and it was the most effective:

```
train["Popularity"].fillna(train["Popularity"].mean(), inplace=True)

train["key"].fillna(train["key"].mode()[0], inplace=True)

train["instrumentalness"].fillna(0, inplace=True)
```

For feature engineering:

tried to create new features but at the end it gave a bad effect on score so decided not to use them:

```
X_train_fe["popularity_energy_ratio"] = X_train_fe["Popularity"] / X_train_fe["energy"]

X_train_fe["danceability_tempo_product"] = X_train_fe["danceability"] * X_train_fe["tempo"]

X_train['loudness_energy_ratio'] = X_train['loudness'] / X_train['energy']
```

For scaling data:

used standard scaler but for the last version of coded didn’t use it since it gave a better score keeping categorical data the same without transforming them into numerical and I can’t use scaler on any other type of data except numbers:

```
scaler = StandardScaler()
```

```
X_train_scaled = scaler.fit_transform(X_train_imputed)
```

Splitting data :

```
X = train_data.drop('Class' ,axis = 1)

y = train_data['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.30, random_state=42)
```

For the last version of code used the split method with RandomForestClassifier

but for catboost didn’t use it but used the training data and target variables:

```
X_train= train_data.drop('Class' ,axis = 1)  X_train represents
the training data

y_train = train_data['Class']  y_train represents the target
variable
```

Modeling

First, we initialize classifiers with specific settings and hyperparameters. We have RandomForestClassifier, LGBMClassifier, CatBoostClassifier, and ExtraTreesClassifier. These classifiers will be trained to make predictions.

Next, we use a technique called grid search to find the best combination of hyperparameters for each classifier. We define different values for the number of estimators and the maximum depth of the trees in each classifier and evaluate their performance using cross-validation. The goal is to find the combination that maximizes the f1_macro score, which is a metric used to assess the models' overall performance.

After the hyperparameter tuning, we have the best estimators for each classifier. Now, we proceed to train a stacking classifier. This classifier combines the predictions of the individual classifiers (Random Forest, LightGBM, CatBoost, and Extra Trees) to make a final prediction. We use RandomForestClassifier as the final estimator to make the ultimate prediction.

We train the stacking classifier using the feature-engineered training data (X_train_fe and y_train).

Finally, we evaluate the performance of each model on the test data (X_test_fe). We print a classification report for each model, which includes metrics such as precision, recall, and F1 score for each class. This helps us understand how well each model is performing and compare their results:

```
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')

param_grid_rf = {

    'n_estimators': [100, 150, 200],

    'max_depth': [None, 10, 20, 30]

}

grid_rf = GridSearchCV(rf_clf, param_grid_rf, cv=3, scoring='f1_macro')

grid_rf.fit(X_train_fe, y_train)

rf_clf = grid_rf.best_estimator_

lgbm_clf = LGBMClassifier(random_state=42)

param_grid_lgbm = {

    'n_estimators': [100, 150, 200],

    'max_depth': [None, 10, 20, 30]

}

grid_lgbm = GridSearchCV(lgbm_clf, param_grid_lgbm, cv=3, scoring='f1_macro')

grid_lgbm.fit(X_train_fe, y_train)

lgbm_clf = grid_lgbm.best_estimator_

cat_clf = CatBoostClassifier(random_state=42, verbose=0)

cat_clf.fit(X_train_fe, y_train)  # Fit the CatBoost classifier

extra_trees_clf = ExtraTreesClassifier(random_state=42)

param_grid_extra_trees = {

    'n_estimators': [100, 150, 200],

    'max_depth': [None, 10, 20, 30]

}

grid_extra_trees = GridSearchCV(extra_trees_clf, param_grid_extra_trees, cv=3, scoring='f1_macro')

grid_extra_trees.fit(X_train_fe, y_train)

extra_trees_clf = grid_extra_trees.best_estimator_
```

```
# Train the stacking classifier
stacking_clf = StackingClassifier(
    estimators=[
        ('rf', rf_clf),
        ('lgbm', lgbm_clf),
        ('cat', cat_clf),
        ('extra_trees', extra_trees_clf)
    ],
    final_estimator=RandomForestClassifier(random_state=42),
    stack_method='predict_proba'  # Use predict_proba for meta-features
)

# Train the stacking classifier
stacking_clf.fit(X_train_fe, y_train)

# Print classification report for each model
classifiers = {
    'Random Forest': rf_clf,
    'LightGBM': lgbm_clf,
    'CatBoost': cat_clf,
    'Extra Trees': extra_trees_clf,
    'Stacking': stacking_clf
}
```

Classification Report for Stacking

	precision	recall	f1-score	support
0	0.73	0.79	0.76	160
1	0.49	0.24	0.32	315
2	0.57	0.42	0.48	327
3	0.85	0.71	0.77	100
4	0.66	0.63	0.64	105
5	0.73	0.75	0.74	361
6	0.46	0.42	0.44	610
7	0.92	0.94	0.93	125
8	0.62	0.58	0.60	435
9	0.55	0.55	0.55	595
10	0.52	0.66	0.58	1186
accuracy			0.57	4319
macro avg	0.64	0.61	0.62	4319
weighted avg	0.57	0.57	0.57	4319

Classification Report for CatBoost

	precision	recall	f1-score	support
0	0.71	0.76	0.73	160
1	0.10	0.03	0.05	315
2	0.57	0.44	0.49	327
3	0.72	0.71	0.71	100
4	0.70	0.61	0.65	105
5	0.72	0.73	0.72	361
6	0.37	0.31	0.34	610
7	0.91	0.93	0.92	125
8	0.61	0.53	0.57	435
9	0.50	0.54	0.52	595
10	0.47	0.62	0.53	1186
accuracy			0.52	4319
macro avg	0.58	0.56	0.57	4319
weighted avg	0.51	0.52	0.51	4319

Classification Report for Random Forest

	precision	recall	f1-score	support
0	0.69	0.81	0.74	160
1	0.06	0.03	0.04	315
2	0.55	0.38	0.45	327
3	0.82	0.68	0.74	100
4	0.66	0.62	0.64	105
5	0.70	0.70	0.70	361
6	0.35	0.28	0.31	610
7	0.92	0.94	0.93	125
8	0.63	0.51	0.56	435
9	0.47	0.53	0.50	595
10	0.46	0.61	0.52	1186
accuracy			0.51	4319
macro avg	0.57	0.55	0.56	4319
weighted avg	0.49	0.51	0.49	4319

LGBM Classifier:

	precision	recall	f1-score	support
0	0.72	0.75	0.73	175
1	0.06	0.03	0.04	339
2	0.57	0.42	0.48	363
3	0.82	0.69	0.75	108
4	0.68	0.64	0.66	118
5	0.71	0.70	0.71	397
6	0.40	0.32	0.36	689
7	0.93	0.91	0.92	137
8	0.59	0.54	0.56	484
9	0.49	0.52	0.50	646
10	0.46	0.61	0.52	1295
accuracy			0.52	4751
macro avg	0.58	0.56	0.57	4751
weighted avg	0.51	0.52	0.51	4751

Extra Trees Classifier:

	precision	recall	f1-score	support
0	0.69	0.74	0.71	175
1	0.03	0.01	0.02	339
2	0.58	0.27	0.37	363
3	0.80	0.71	0.75	108
4	0.70	0.54	0.61	118
5	0.71	0.71	0.71	397
6	0.35	0.26	0.29	689
7	0.88	0.93	0.91	137
8	0.60	0.51	0.56	484
9	0.47	0.55	0.51	646
10	0.44	0.64	0.52	1295
accuracy			0.50	4751
macro avg	0.57	0.53	0.54	4751
weighted avg	0.49	0.50	0.48	4751

After initial dissatisfaction with the results, several modifications and alternative approaches were attempted. Ultimately, it was concluded that the most satisfactory outcome was achieved by employing the CatBoost classifier. Although ensemble methods, such as stacking and voting, yielded promising results, they were incompatible with the categorical data utilized. To address this, the categorical data was transformed into numerical values. However, this transformation adversely affected the scores obtained from the stacking method and rendered the voting method ineffective.

The highest result reached using stacking:

```
# Initialize classifiers with hyperparameter tuning

rf_clf = RandomForestClassifier(n_estimators=200, max_depth=30, random_state=42,
class_weight='balanced')

lgbm_clf = LGBMClassifier(n_estimators=200, max_depth=30, random_state=42)

cat_clf = CatBoostClassifier(random_state=42, verbose=0)

extra_trees_clf = ExtraTreesClassifier(n_estimators=200, max_depth=30, random_state=42)


# Stacking Classifier with RandomForest, LGBM, CatBoost, and ExtraTrees as base models

# and RandomForest as the meta-classifier

stacking_clf = StackingClassifier(

    estimators=[

        ('rf', rf_clf),

        ('lgbm', lgbm_clf),

        ('cat', cat_clf),

        ('extra_trees', extra_trees_clf)

    ],

    final_estimator=RandomForestClassifier(random_state=42),

    stack_method='predict_proba'  # Use predict_proba for meta-features
```

```
)

# Train the stacking classifier

stacking_clf.fit(X_train, y_train)

# Make predictions on the test set

y_pred = stacking_clf.predict(X_test)

# Calculate F1 score

f1_macro = f1_score(y_test, y_pred, average='macro')

print("F1 score (macro):", f1_macro)

# Generate classification report

class_report = classification_report(y_test, y_pred)

print("Classification Report:\n", class_report)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.73      0.81      0.77       160
           1       0.47      0.26      0.33       315
           2       0.55      0.40      0.46       327
           3       0.87      0.76      0.81       100
           4       0.67      0.63      0.65       105
           5       0.73      0.73      0.73       361
           6       0.46      0.42      0.44       610
           7       0.94      0.94      0.94       125
           8       0.65      0.59      0.62       435
           9       0.53      0.53      0.53       595
          10       0.53      0.67      0.59      1186

    accuracy                           0.58      4319
   macro avg       0.65      0.61      0.63      4319
weighted avg       0.58      0.58      0.57      4319
```

As for the of using catboost classifier for the last version of code which gave the highest score:

```
X_train= train_data.drop('Class' ,axis = 1)

y_train = train_data['Class']


model = CatBoostClassifier(loss_function='MultiClass', verbose=False)

model.fit(X_train, y_train, cat_features=cat_cols)
```

Test Classification Report:

	precision	recall	f1-score	support
0	0.85	0.88	0.87	104
1	0.44	0.21	0.28	204
2	0.72	0.62	0.67	235
3	0.90	0.93	0.91	68
4	0.85	0.81	0.83	68
5	0.79	0.81	0.79	236
6	0.54	0.51	0.53	403
7	0.98	0.94	0.96	87
8	0.74	0.74	0.74	292
9	0.70	0.67	0.68	406
10	0.62	0.76	0.68	777
accuracy			0.68	2880
macro avg	0.74	0.72	0.72	2880
weighted avg	0.67	0.68	0.67	2880

Result:

As the result shows the difference between the different types of classifiers and methods that were applied in conclusion the best score that was reached was by catboots classifier.

Also tried to imbalance the data but using:

```
oversampler = RandomOverSampler(random_state=42)

X_train_resampled, y_train_resampled =
oversampler.fit_resample(X_train, y_train)
```

```
smote = SMOTE(random_state=42)

X_resampled, y_resampled = smote.fit_resample(X, y)
```

Both methods gave an almost perfect score for training but bad result using the test dataset

That's it for the final project for the Shai for ai course in data science.

Team members:

- ❖ Yasmin Hammad
- ❖ Yousef Al Farani
- ❖ Mahmoud Abdelqader
- ❖ Malak Kanana