```
### Ericsson_ML_Challenge_MaterialType_Prediction
```

```
###!mkdir ~/.kaggle
```

```
###!cp /kaggle.json ~/.kaggle/
```

```
###!chmod 600 ~/.kaggle/kaggle.json
```

```
###! pip install kaggle
```

```
###!pip install keras-tuner
```

```
###!kaggle datasets download -d saranyashalya/ericsson-ml-challenge-materialtype-prediction
```

```
###! unzip /content/ericsson-ml-challenge-materialtype-prediction.zip
```

```
####! pip install tensorflow
```

```
###! pip install bayesian-optimization
```

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn import preprocessing
import matplotlib.pyplot as plt
tf.random.set_seed(123)
np.random.seed(123)
```

```
##!pip install bayesian-optimization
```

```
# Import packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from keras.models import Sequential
from keras.layers import Dense, BatchNormalization, Dropout
#from tensorflow.keras.optimizers import Adam, SGD, RMSprop, Adadelta, Adagrad, Adamax, Nadam, Ftrl
from keras.callbacks import EarlyStopping, ModelCheckpoint
###from keras.wrappers.scikit_learn import KerasClassifier
from math import floor
from sklearn.metrics import make_scorer, accuracy_score
from bayes_opt import BayesianOptimization
from sklearn.model_selection import StratifiedKFold
from keras.layers import LeakyReLU
LeakyReLU = LeakyReLU(alpha=0.1)
import warnings
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns", None)
```

```
# Import packages
# Basic packages

import pickle
from math import floor
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import MinMaxScaler

# Evaluation and bayesian optimization
from sklearn.metrics import make_scorer, mean_absolute_error
from sklearn.metrics import mean_squared_error as MSE
from hyperopt import hp, fmin, tpe
from sklearn.model_selection import GridSearchCV, StratifiedKFold

import warnings
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns", None)
```

```
train_data = pd.read_csv("/content/train_file.csv")
```

```
test_data = pd.read_csv("/content/test_file.csv")
```

```
print(train_data.shape, test_data.shape)
```

```
(31653, 12) (21102, 11)
```

```
train_data.columns
```

```
Index(['ID', 'UsageClass', 'CheckoutType', 'CheckoutYear', 'CheckoutMonth',
       'Checkouts', 'Title', 'Creator', 'Subjects', 'Publisher',
       'PublicationYear', 'MaterialType'],
      dtype='object')
```

```
train_data.head(3)
```

	ID	UsageClass	CheckoutType	CheckoutYear	CheckoutMonth	Checkouts	Title	Creator	Subjects	Publisher	Publicat
0	1	Physical	Horizon	2005	4	1	Tidal wave	NaN	Tsunamis, Tsunamis Juvenile literature	NaN	
1	2	Physical	Horizon	2005	4	1	London holiday / Richard Peck.	Peck, Richard, 1934-	NaN	Viking,	
2	3	Physical	Horizon	2005	4	3	Cinco de Mayo : celebrating Hispanic pride / C...	Gnojewski, Carol	Cinco de Mayo Mexican holiday History Juvenile...	Enslow Publishers,	

```
train_data["MaterialType"].value_counts()
```

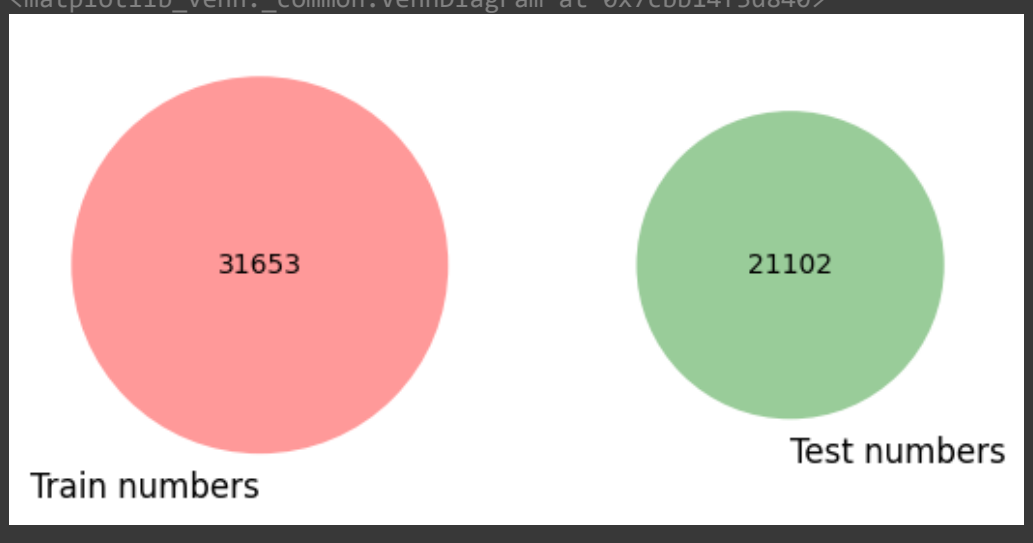
```
BOOK         21707
SOUNDDISC     4149
VIDEOCASS     2751
VIDEODISC     1420
SOUNDCASS     1020
MIXED          347
MUSIC          165
CR              94
Name: MaterialType, dtype: int64
```

```
test_data["MaterialType"] = 0
```

```
from matplotlib_venn import venn2, venn2_circles, venn2_unweighted
from matplotlib_venn import venn3, venn3_circles
```

```
set_numbers_train = set(train_data[['ID']].drop_duplicates().sort_values(by = 'ID')['ID'].tolist())
set_numbers_test = set(test_data[['ID']].drop_duplicates().sort_values(by = 'ID')['ID'].tolist())
venn2((set_numbers_train, set_numbers_test), set_labels = ('Train numbers', 'Test numbers'))
```

```
<matplotlib_venn._common.VennDiagram at 0x7cbb14f3d840>
```



The above data explains the `size` of train and test data.

```
train_data.columns
```

```
Index(['ID', 'UsageClass', 'CheckoutType', 'CheckoutYear', 'CheckoutMonth',
       'Checkouts', 'Title', 'Creator', 'Subjects', 'Publisher',
       'PublicationYear', 'MaterialType'],
      dtype='object')
```

```
num_var = [feature for feature in train_data.columns if train_data[feature].dtypes != 'O']
discrete_var = [feature for feature in num_var if len(train_data[feature].unique()) <= 25]
cont_var = [feature for feature in num_var if feature not in discrete_var]
categ_var = [feature for feature in train_data.columns if feature not in num_var]
```

```
print("The Numerical Variables are :", num_var)
print("The Discreate Variables are :", discrete_var)
print("The Continuous Variables are :", cont_var)
print("The Categorical Variables are :", categ_var)
```

```
The Numerical Variables are : ['ID', 'CheckoutYear', 'CheckoutMonth', 'Checkouts']
The Discreate Variables are : ['CheckoutYear', 'CheckoutMonth']
The Continuous Variables are : ['ID', 'Checkouts']
The Categorical Variables are : ['UsageClass', 'CheckoutType', 'Title', 'Creator', 'Subjects', 'Publisher', 'Publication
```

CHECKING `NULL VALUES` OR NOT

```
train_data.isnull().sum()
```

```
ID                    0
UsageClass            0
CheckoutType          0
CheckoutYear          0
CheckoutMonth         0
Checkouts             0
Title                 0
Creator           23137
Subjects           1763
Publisher         21916
PublicationYear   21931
MaterialType          0
dtype: int64
```

```
train_data = train_data.fillna(0)
```

```
train_data.isnull().sum()
```

```
ID                0
UsageClass        0
CheckoutType      0
CheckoutYear      0
CheckoutMonth     0
Checkouts         0
Title             0
Creator           0
Subjects          0
Publisher         0
PublicationYear   0
MaterialType      0
dtype: int64
```

```
test_data = test_data.fillna(0)
```

CATEGORICAL VARIABLES

```
train_data.columns
```

```
Index(['ID', 'UsageClass', 'CheckoutType', 'CheckoutYear', 'CheckoutMonth',
       'Checkouts', 'Title', 'Creator', 'Subjects', 'Publisher',
       'PublicationYear', 'MaterialType'],
      dtype='object')
```

```
train_data["MaterialType"].value_counts()
```

```
BOOK         21707
SOUNDDISC     4149
VIDEOCASS     2751
VIDEODISC     1420
SOUNDCASS     1020
MIXED          347
MUSIC          165
CR              94
Name: MaterialType, dtype: int64
```

(0: 'BOOK', 1: 'CR', 2: 'MIXED', 3: 'MUSIC', 4: 'SOUNDCASS', 5: 'SOUNDDISC', 6: 'VIDEOCASS', 7: 'VIDEODISC'}

```
c = train_data["MaterialType"].astype('category')
d = dict(enumerate(c.cat.categories))
print(d)
```

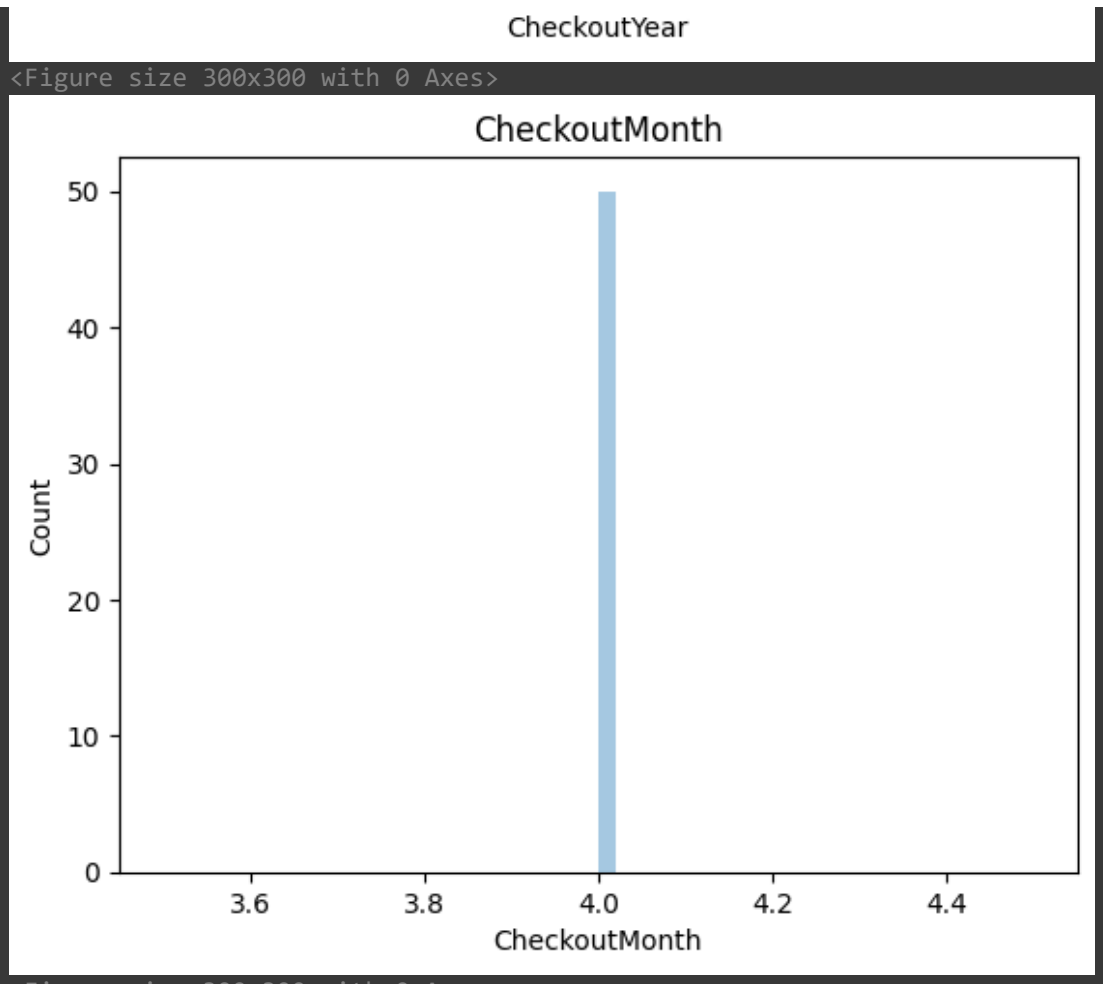
```
{0: 'BOOK', 1: 'CR', 2: 'MIXED', 3: 'MUSIC', 4: 'SOUNDCASS', 5: 'SOUNDDISC', 6: 'VIDEOCASS', 7: 'VIDEODISC'}
```

```
train_data['MaterialType'] = train_data.MaterialType.astype('category').cat.codes
```

Analysis For `Numerical` Variables

```
#import seaborn as sns
#import matplotlib.pyplot as plt

for feature in num_var:
  data=train_data.copy()
  sns.distplot(train_data[feature])
  plt.xlabel(feature)
  plt.ylabel("Count")
  plt.title(feature)
  plt.figure(figsize=(3,3))
  plt.show()
```




```
train_data.head(3)
```

	ID	UsageClass	CheckoutType	CheckoutYear	CheckoutMonth	Checkouts	Title	Creator	Subjects	Publisher	Pub
0	1	Physical	Horizon	2005	4	1	Tidal wave	0	Tsunamis, Tsunamis Juvenile literature	0	
1	2	Physical	Horizon	2005	4	1	London holiday / Richard Peck.	Peck, Richard, 1934-	0	Viking,	
2	3	Physical	Horizon	2005	4	3	Cinco de Mayo : celebrating Hispanic pride / C...	Gnojewski, Carol	Cinco de Mayo Mexican holiday History Juvenile...	Enslow Publishers,	

```
####! pip install klib
```

Using KLIB Library

```
import klib
```

```
train_data = klib.clean_column_names(train_data)
```

```
test_data = klib.clean_column_names(test_data)
```

```
train_data = klib.convert_datatypes(train_data)
test_data = klib.convert_datatypes(test_data)
```

```
train_data = klib.mv_col_handling(train_data)
test_data = klib.mv_col_handling(test_data)
```

```
train_data.dtypes
```

```
id                   int16
usage_class       category
checkout_type     category
checkout_year        int16
checkout_month        int8
checkouts             int8
title               string
creator             object
subjects            object
publisher           object
publication_year  category
material_type         int8
dtype: object
```

Data Conversion

```
train_data.head(2)
```

	id	usage_class	checkout_type	checkout_year	checkout_month	checkouts	title	creator	subjects	publisher	publ
0	1	Physical	Horizon	2005	4	1	Tidal wave	0	Tsunamis, Tsunamis Juvenile literature	0	
1	2	Physical	Horizon	2005	4	1	London holiday / Richard Peck.	Peck, Richard, 1934-	0	Viking,	

```
train_data["usage_class"] = train_data["usage_class"].astype('category').cat.codes
test_data["usage_class"] = test_data["usage_class"].astype('category').cat.codes
train_data["checkout_type"] = train_data["checkout_type"].astype('category').cat.codes
test_data["checkout_type"] = test_data["checkout_type"].astype('category').cat.codes
```

```
train_data.dtypes
```

```
id                   int16
usage_class           int8
checkout_type         int8
checkout_year        int16
checkout_month        int8
checkouts             int8
title               string
creator             object
subjects            object
publisher           object
publication_year  category
material_type         int8
dtype: object
```

```
train_data.head(3)
```

	id	usage_class	checkout_type	checkout_year	checkout_month	checkouts	title	creator	subjects	publisher
0	1	0	0	2005	4	1	Tidal wave	0	Tsunamis, Tsunamis Juvenile literature	0
1	2	0	0	2005	4	1	London holiday / Richard Peck.	Peck, Richard, 1934-	0	Viking,
2	3	0	0	2005	4	3	Cinco de Mayo : celebrating Hispanic pride / C...	Gnojewski, Carol	Cinco de Mayo Mexican holiday History Juvenile...	Enslow Publishers,

```
import re
def publication_year(s):
    '''
    '''
    k = re.findall(r'\d{4}', s)
    if len(k) != 0:
        try:
            k = sorted(k)[0]
            k = (pd.to_datetime('now').year - pd.to_datetime(k, format = '%Y').year)
        except:
            k = pd.period_range(start = k, end = '1678', freq = 'Y')
            k = len(k)
            k = (pd.to_datetime('now').year - pd.to_datetime('1678', format = '%Y').year + k)
    else:
        k = 0
    return k###train_data['publication_year'][:100]
```

```
train_data["publication_year"] = train_data["publication_year"].astype(str)
```

```
train_data["publication_year"] = train_data["publication_year"].apply(publication_year)
```

```
test_data["publication_year"] = test_data["publication_year"].astype(str)
test_data["publication_year"] = test_data["publication_year"].apply(publication_year)
```

```
train_data.head(2)
```

	id	usage_class	checkout_type	checkout_year	checkout_month	checkouts	title	creator	subjects	publisher	publ
0	1	0	0	2005	4	1	Tidal wave	0	Tsunamis, Tsunamis Juvenile literature	0	
1	2	0	0	2005	4	1	London holiday / Richard Peck.	Peck, Richard, 1934-	0	Viking,	

```
###!pip install textblob
```

```
####! pip install pickle
```

```
import pandas, numpy, string, textblob
import pickle
from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, svm, decomposition, ensemble
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
import xgboost
from keras import layers, models, optimizers
from keras.preprocessing import text, sequence
import matplotlib.pyplot as plt
```

```
import string
import re
def clean_text(text):
    '''Make text lowercase, remove text in square brackets,remove links,remove punctuation
    and remove words containing numbers.'''
    text = text.lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('<.*?>+', '', text)
    text = re.sub('[%s]' % re.escape(string.punctuation), '', text)
    text = re.sub('\n', '', text)
    text = re.sub('\d+', '', text)
    text = re.sub('\w*\d\w*', '', text)
    return text
```

```
train_data.columns
```

```
Index(['id', 'usage_class', 'checkout_type', 'checkout_year', 'checkout_month',
       'checkouts', 'title', 'creator', 'subjects', 'publisher',
       'publication_year', 'material_type'],
      dtype='object')
```

```
test_data.columns
```

```
Index(['id', 'usage_class', 'checkout_type', 'checkout_year', 'checkout_month',
       'checkouts', 'title', 'creator', 'subjects', 'publisher',
```

```
       'publication_year', 'material_type'],
      dtype='object')
```

```
train_data.isnull().sum()
```

```
id                 0
usage_class        0
checkout_type      0
checkout_year      0
checkout_month     0
checkouts          0
title              0
creator            0
subjects           0
publisher          0
publication_year   0
material_type      0
dtype: int64
```

```
train_data["title"] = train_data["title"].astype('string')
train_data["creator"] = train_data["creator"].astype('string')
train_data["subjects"] = train_data["subjects"].astype('string')
train_data["publisher"] = train_data["publisher"].astype('string')
```

```
test_data["title"] = test_data["title"].astype('string')
#test_data["creator"] = test_data["creator"].astype('string')
test_data["subjects"] = test_data["subjects"].astype('string')
#test_data["publisher"] = test_data["publisher"].astype('string')
```

```
train_data["title"] = train_data["title"].apply(lambda x: clean_text(x))
train_data["creator"] = train_data["creator"].apply(lambda x: clean_text(x))
train_data["subjects"] = train_data["subjects"].apply(lambda x: clean_text(x))
train_data["publisher"] = train_data["publisher"].apply(lambda x: clean_text(x))
```

```
test_data["title"] = test_data["title"].apply(lambda x: clean_text(x))
test_data["subjects"] = test_data["subjects"].apply(lambda x: clean_text(x))
```

```
train_data.drop(columns ="creator", inplace=True)
train_data.drop(columns = "publisher", inplace=True)
```

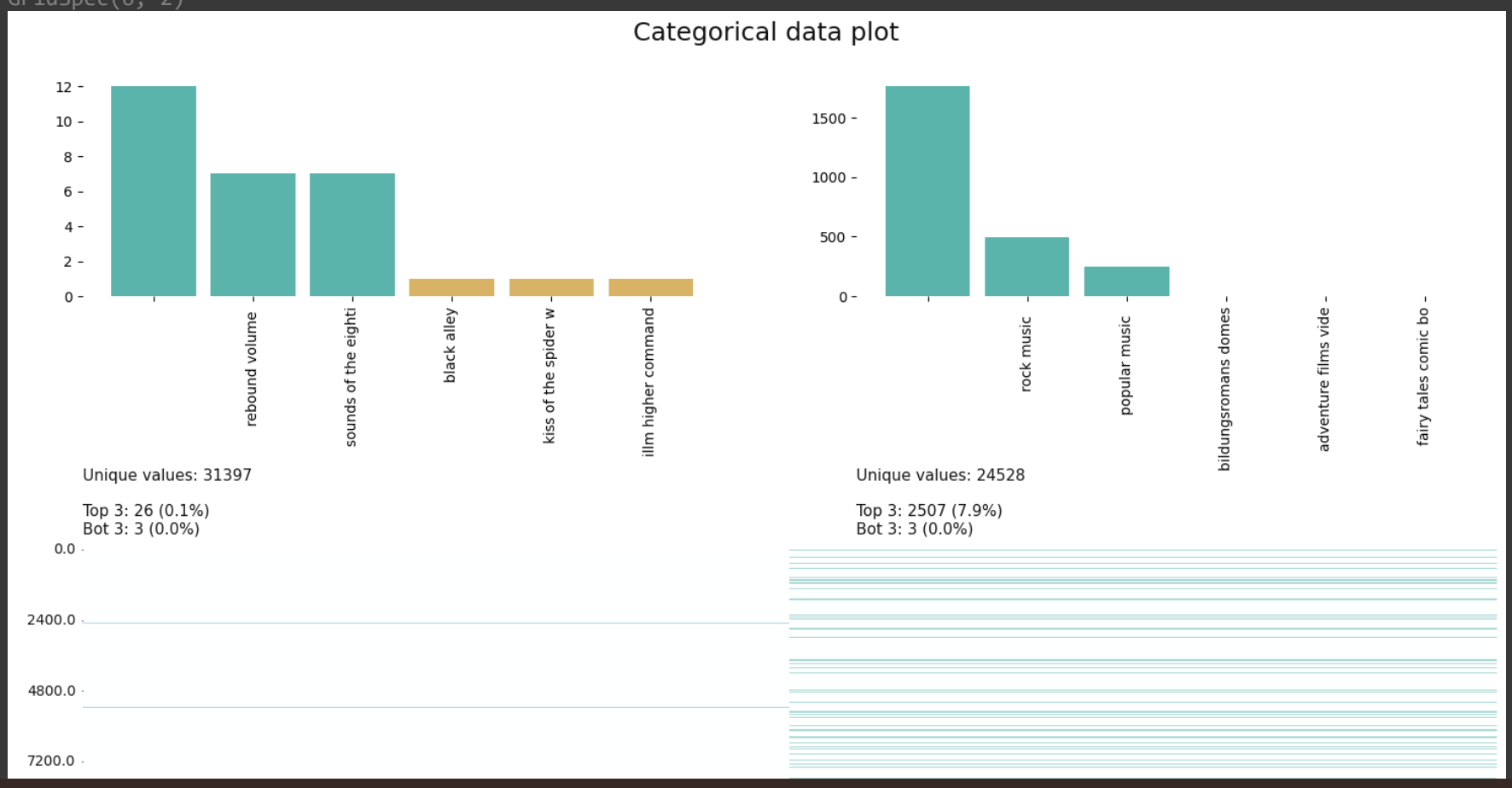
```
train_data.columns
```

```
Index(['id', 'usage_class', 'checkout_type', 'checkout_year', 'checkout_month',
       'checkouts', 'title', 'subjects', 'publication_year', 'material_type'],
      dtype='object')
```

Data Visualization

```
klib.cat_plot(train_data)
```

```
GridSpec(6, 2)
```

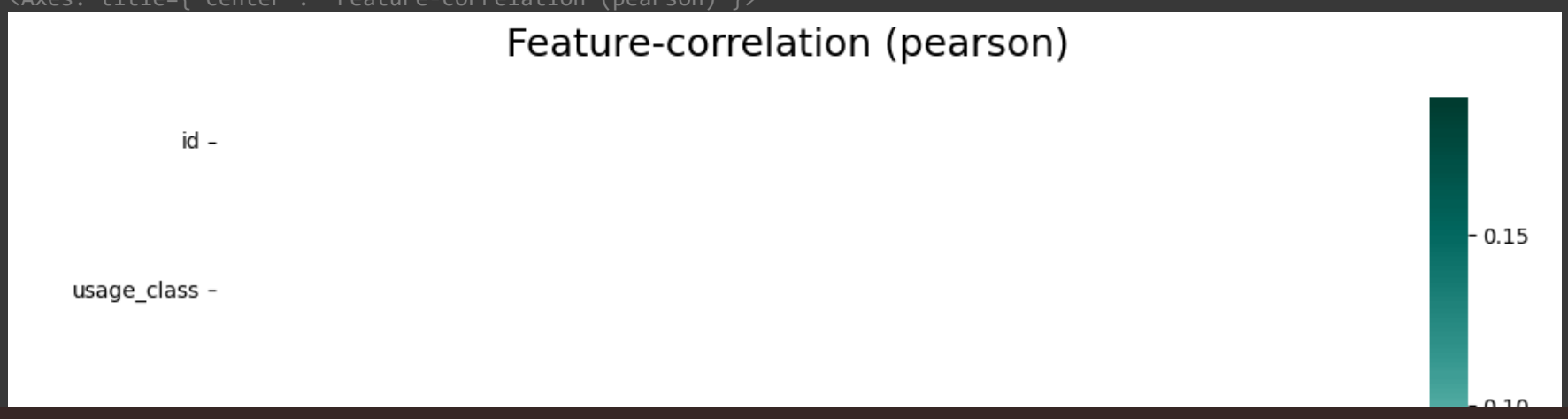



```
klib.corr_mat(train_data)
```

	id	usage_class	checkout_type	checkout_year	checkout_month	checkouts	publication_year	material_
id	1.00	-	-	-	-	-0.01	0.01	
usage_class	-	-	-	-	-	-	-	
checkout_type	-	-	-	-	-	-	-	
checkout_year	-	-	-	-	-	-	-	
checkout_month	-	-	-	-	-	-	-	
checkouts	-0.01	-	-	-	-	1.00	0.02	
publication_year	0.01	-	-	-	-	0.02	1.00	
material_type	-0.01	-	-	-	-	0.24	-0.19	

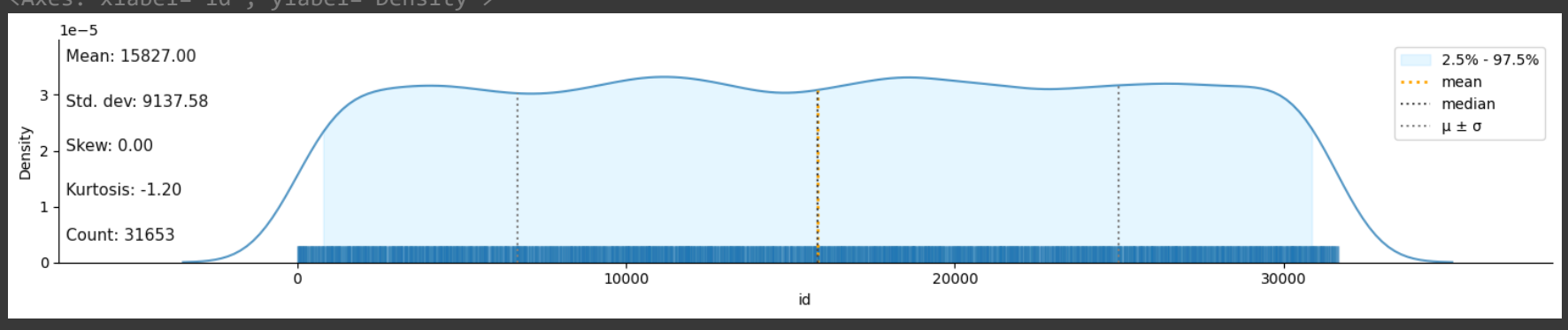
```
klib.corr_plot(train_data)
```

```
<Axes: title={'center': 'Feature-correlation (pearson)'}>
```




```
klib.dist_plot(train_data)
```

```
Large dataset detected, using 10000 random samples for the plots. Summary statistics are still based on the entire dat
<Axes: xlabel='id', ylabel='Density'>
```




```
klib.missingval_plot(train_data)
```

```
No missing values found in the dataset.
```

```
test_data.columns
```

```
Index(['id', 'usage_class', 'checkout_type', 'checkout_year', 'checkout_month',
       'checkouts', 'title', 'creator', 'subjects', 'publisher',
       'publication_year', 'material_type'],
      dtype='object')
```

Vectorization Of the Column Using `TF-IDF Vectorizer`

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
MAX_WORDS = 200000
from tensorflow.keras.layers import TextVectorization
```

```
vectorizer = TextVectorization(max_tokens = MAX_WORDS,
output_sequence_length = 1800,
output_mode = 'int')
```

```
train_data.columns
```

```
Index(['id', 'usage_class', 'checkout_type', 'checkout_year', 'checkout_month',
       'checkouts', 'title', 'subjects', 'publication_year', 'material_type'],
      dtype='object')
```

```
vectorizer.adapt(train_data["subjects"].values)
```

```
vectorizer.get_vocabulary()
```

```
 'organized',
 'nevada',
 'network',
 'maryland',
 'lesbians',
 'jane',
 'inventors',
 'herbs',
 'healing',
 'forecasting',
 'cultural',
 'cross',
 'criminal',
 'composers',
 'anime',
 'aging',
 'aerial',
 'zen',
 'statesmen',
 'star',
 'quality',
 'plains',
 'moon',
 'meditation',
 'martin',
 'mammals',
 'lost',
 'inventions',
 'fishing',
 'eastern',
 'butterflies',
 'best',
 'answers',
 'acting',
 'x',
 'whales',
 'treasure',
 'texts',
 'snakes',
 'safety',
 'research',
 'quilts',
 ...]
```

```
TrainvectorizedText = vectorizer(train_data["subjects"].values)
```

```
vectorizer.adapt(test_data["subjects"].values)
```

```
TestvectorizedText = vectorizer(test_data["subjects"].values)
```

```
import scipy
```

```
train_data.columns
```

```
Index(['id', 'usage_class', 'checkout_type', 'checkout_year', 'checkout_month',
       'checkouts', 'title', 'subjects', 'publication_year', 'material_type'],
      dtype='object')
```

```
Y_train = pd.get_dummies(train_data.material_type)
Y_test = pd.get_dummies(test_data.material_type)
```

```
import tensorflow as tf
```

```
#MCSHBAP - map, cache, suffle, batch, prefetch
dataset = tf.data.Dataset.from_tensor_slices((TrainvectorizedText, Y_train))
dataset = dataset.cache()
dataset = dataset.shuffle(160000)
dataset = dataset.batch(16)
dataset = dataset.prefetch(8) #helps bottleneck
```

```
dataset.as_numpy_iterator().next()
```

```
(array([[  26,    2,  331, ...,    0,    0,    0],
        [ 202, 2153,  178, ...,    0,    0,    0],
        [5063,   82,    3, ...,    0,    0,    0],
        ...,
        [3164, 1383,    2, ...,    0,    0,    0],
```

```
            [   0,    0,    0, ...,    0,    0,    0],
            [  19,   15, 3038, ...,    0,    0,    0]]),
      array([[1, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 1, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0],
            [1, 0, 0, 0, 0, 0, 0, 0]], dtype=uint8))
```

```
batch_x, batch_y = dataset.as_numpy_iterator().next()
```

```
batch_x
batch_x.shape
batch_y.shape
```

```
(16, 8)
```

```
len(train_data)
```

```
31653
```

```
train = dataset.take(int(len(dataset)*.7))
val = dataset.skip(int(len(dataset)*.7)).take(int(len(dataset)*.2))
test = dataset.skip(int(len(dataset)*.9)).take(int(len(dataset)*.1))
```

```
len(train)
```

```
1385
```

```
train_generator = train.as_numpy_iterator()
```

```
train_generator.next()
```

```
(array([[ 7323,  4821,    18, ...,     0,     0,     0],
        [    8,     7,    10, ...,     0,     0,     0],
        [   99,     0,     0, ...,     0,     0,     0],
        ...,
        [ 1820,  1600,   264, ...,     0,     0,     0],
        [10517,   473,    45, ...,     0,     0,     0],
        [11367,  4158,     0, ...,     0,     0,     0]]),
 array([[1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 1, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 1],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 1, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [1, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 1, 0]], dtype=uint8))
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Bidirectional, Embedding, Dense
```

```
model = Sequential()
# Creating the embedding layer
model.add(Embedding(MAX_WORDS+1, 32))
# Bidirectional LSTM Layer
model.add(Bidirectional(LSTM(32, activation='tanh')))
# Feature extractor Fully connected layers
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(128, activation='relu'))
# Final layer
model.add(Dense(8, activation='sigmoid'))
```

```
model.compile(loss='BinaryCrossentropy', optimizer='Adam', metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, None, 32)          6400032

 bidirectional_1 (Bidirecti  (None, 64)                16640
 onal)

 dense_4 (Dense)             (None, 128)               8320

 dropout_2 (Dropout)         (None, 128)               0

 dense_5 (Dense)             (None, 256)               33024

 dropout_3 (Dropout)         (None, 256)               0

 dense_6 (Dense)             (None, 128)               32896

 dense_7 (Dense)             (None, 8)                 1032

=================================================================
Total params: 6491944 (24.76 MB)
Trainable params: 6491944 (24.76 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
import tensorflow as tf
```

```
es_callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3)
```

```
model.fit(train, epochs=41, validation_data = val, callbacks=[es_callback])
```

```
Epoch 1/41
1385/1385 [==============================] - 158s 111ms/step - loss: 0.1375 - accuracy: 0.8242 - val_loss: 0.1047 - val_a
Epoch 2/41
1385/1385 [==============================] - 141s 102ms/step - loss: 0.1016 - accuracy: 0.8644 - val_loss: 0.0900 - val_a
Epoch 3/41
1385/1385 [==============================] - 142s 103ms/step - loss: 0.0894 - accuracy: 0.8751 - val_loss: 0.0815 - val_a
Epoch 4/41
1385/1385 [==============================] - 143s 103ms/step - loss: 0.0835 - accuracy: 0.8796 - val_loss: 0.0699 - val_a
Epoch 5/41
1385/1385 [==============================] - 139s 100ms/step - loss: 0.0759 - accuracy: 0.8905 - val_loss: 0.0647 - val_a
Epoch 6/41
1385/1385 [==============================] - 138s 100ms/step - loss: 0.0706 - accuracy: 0.8963 - val_loss: 0.0611 - val_a
Epoch 7/41
1385/1385 [==============================] - 139s 100ms/step - loss: 0.0659 - accuracy: 0.9039 - val_loss: 0.0577 - val_a
Epoch 8/41
1385/1385 [==============================] - 137s 99ms/step - loss: 0.0620 - accuracy: 0.9109 - val_loss: 0.0522 - val_ac
Epoch 9/41
1385/1385 [==============================] - 139s 101ms/step - loss: 0.0589 - accuracy: 0.9157 - val_loss: 0.0490 - val_a
Epoch 10/41
1385/1385 [==============================] - 138s 99ms/step - loss: 0.0562 - accuracy: 0.9192 - val_loss: 0.0484 - val_ac
Epoch 11/41
1385/1385 [==============================] - 137s 99ms/step - loss: 0.0526 - accuracy: 0.9250 - val_loss: 0.0508 - val_ac
Epoch 12/41
1385/1385 [==============================] - 139s 101ms/step - loss: 0.0506 - accuracy: 0.9294 - val_loss: 0.0457 - val_a
Epoch 13/41
1385/1385 [==============================] - 137s 99ms/step - loss: 0.0500 - accuracy: 0.9290 - val_loss: 0.0460 - val_ac
Epoch 14/41
1385/1385 [==============================] - 139s 100ms/step - loss: 0.0471 - accuracy: 0.9332 - val_loss: 0.0452 - val_a
Epoch 15/41
1385/1385 [==============================] - 138s 100ms/step - loss: 0.0465 - accuracy: 0.9347 - val_loss: 0.0423 - val_a
Epoch 16/41
1385/1385 [==============================] - 138s 99ms/step - loss: 0.0458 - accuracy: 0.9352 - val_loss: 0.0405 - val_ac
Epoch 17/41
```

```
1385/1385 [==============================] - 139s 101ms/step - loss: 0.0429 - accuracy: 0.9387 - val_loss: 0.0376 - val_a
Epoch 18/41
1385/1385 [==============================] - 145s 105ms/step - loss: 0.0433 - accuracy: 0.9381 - val_loss: 0.0383 - val_a
Epoch 19/41
1385/1385 [==============================] - 138s 99ms/step - loss: 0.0422 - accuracy: 0.9411 - val_loss: 0.0375 - val_ac
Epoch 20/41
1385/1385 [==============================] - 138s 99ms/step - loss: 0.0407 - accuracy: 0.9417 - val_loss: 0.0361 - val_ac
Epoch 21/41
1385/1385 [==============================] - 144s 104ms/step - loss: 0.0397 - accuracy: 0.9447 - val_loss: 0.0355 - val_a
Epoch 22/41
1385/1385 [==============================] - 141s 102ms/step - loss: 0.0386 - accuracy: 0.9446 - val_loss: 0.0377 - val_a
Epoch 23/41
1385/1385 [==============================] - 139s 101ms/step - loss: 0.0385 - accuracy: 0.9454 - val_loss: 0.0361 - val_a
Epoch 24/41
1385/1385 [==============================] - 137s 99ms/step - loss: 0.0377 - accuracy: 0.9468 - val_loss: 0.0347 - val_ac
Epoch 25/41
1385/1385 [==============================] - 140s 101ms/step - loss: 0.0375 - accuracy: 0.9478 - val_loss: 0.0313 - val_a
Epoch 26/41
1385/1385 [==============================] - 138s 100ms/step - loss: 0.0356 - accuracy: 0.9499 - val_loss: 0.0338 - val_a
Epoch 27/41
1385/1385 [==============================] - 140s 101ms/step - loss: 0.0356 - accuracy: 0.9497 - val_loss: 0.0324 - val_a
Epoch 28/41
1385/1385 [==============================] - 139s 100ms/step - loss: 0.0354 - accuracy: 0.9498 - val_loss: 0.0330 - val_a
<keras.src.callbacks.History at 0x7cbbb68767a0>
```

```
#from matplotlib import pyplot as plt
#plt.figure(figsize=(15,12))
#pd.DataFrame(history.history).plot()
#plt.show()
```

```
from tensorflow.keras.metrics import Precision, Recall, CategoricalAccuracy
```

```
model.evaluate(TrainvectorizedText)
```

```
990/990 [==============================] - 2s 2ms/step - loss: 0.0000e+00 - accuracy: 0.0000e+00
[0.0, 0.0]
```

```
Y_pred = (model.predict(TestvectorizedText) > 0.5).astype(int)
```

```
660/660 [==============================] - 30s 45ms/step
```

```
Y_pred
Y_pred = pd.DataFrame(Y_pred)
```

```
Y_pred.shape
```

```
(21102, 8)
```

```
##
output = {
    0: "BOOK",
    1: "CR",
    2: "MIXED" ,
    3: "MUSIC" ,
    4: "SOUNDCASS" ,
    5: "SOUNDDISC" ,
    6: "VIDEOCASS" ,
    7: "VIDEODISC"
}
```

```
Y_pred = pd.DataFrame(Y_pred)
```

```
from sklearn.preprocessing import OneHotEncoder
enc = OneHotEncoder(handle_unknown='ignore')
```

```
new_Y_pred = Y_pred.idxmax(axis=1)
```

```
new_Y_test = Y_test.idxmax(axis=1)
```

```
new_Y_pred = pd.DataFrame(new_Y_pred)
```

```
new_Y_test = pd.DataFrame(new_Y_test)
```

```
new_Y_pred = new_Y_pred.rename(columns ={0: "PREDICT"} )
```

```
new_Y_test = new_Y_test.rename(columns ={0: "PREDICT"} )
```

```
print('Baseline: Accuracy: ', round(accuracy_score(new_Y_test, new_Y_pred)*100, 2))
```

```
Baseline: Accuracy:  77.54
```

```
new_Y_pred["PREDICT"] = new_Y_pred["PREDICT"].map(output)
new_Y_test["PREDICT"] = new_Y_test["PREDICT"].map(output)
```

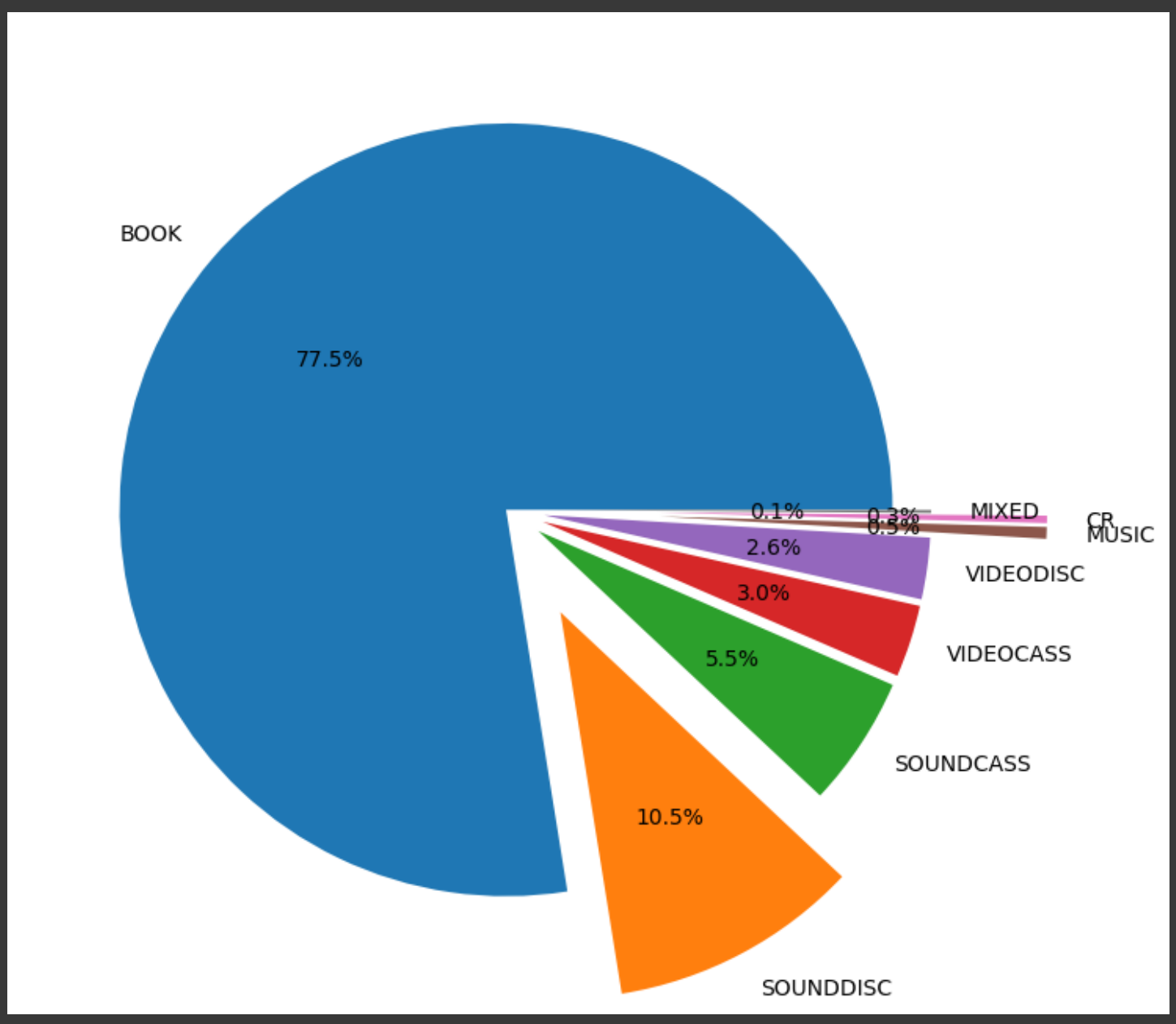
```
new_Y_pred["PREDICT"].unique()
```

```
array(['BOOK', 'VIDEOCASS', 'SOUNDDISC', 'VIDEODISC', 'CR', 'SOUNDCASS',
       'MIXED', 'MUSIC'], dtype=object)
```

```
keys = ['BOOK', 'SOUNDDISC', 'SOUNDCASS', 'VIDEOCASS', 'VIDEODISC',
'MUSIC', 'CR', 'MIXED']
explode = [0, 0.3, 0.1, 0.1, 0.1, 0.4, 0.4, 0.1]
```

```
all = new_Y_pred["PREDICT"].value_counts()
```

```
plt.figure(figsize = [15,8])
# plotting data on chart
plt.pie(all, labels=keys, explode=explode, autopct='%.1f%%')
# displaying chart
plt.show()
```




```
Start coding or generate with AI.
```