```
### 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	Pub
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()
```

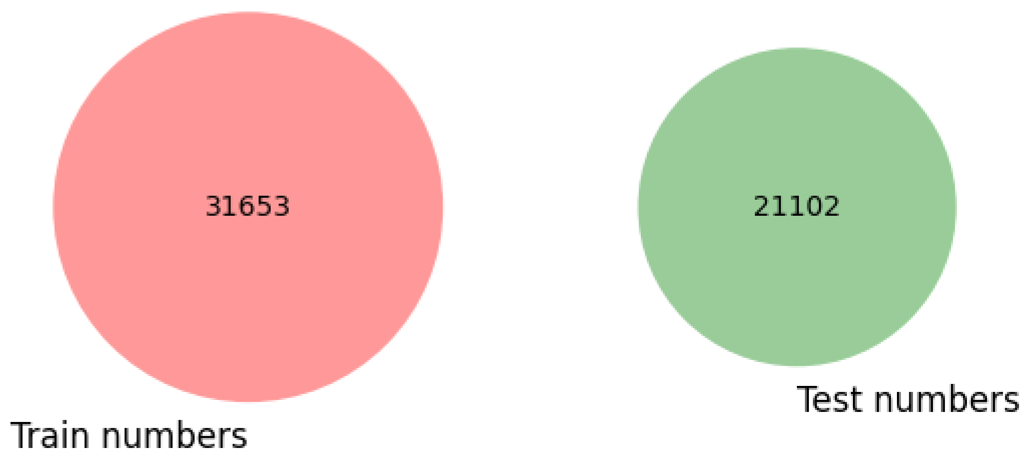
```
BOOK          21707
SOUNDDISC      4149
VIDEOCASS      2751
VIDEODISC      1420
SOUNDCASS      1020
MIXED           347
MUSIC           165
CR               94
Name: MaterialType, dtype: int64
```

```
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 0x786e69cb0b20>
```



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', 'Publicati
```

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)
```

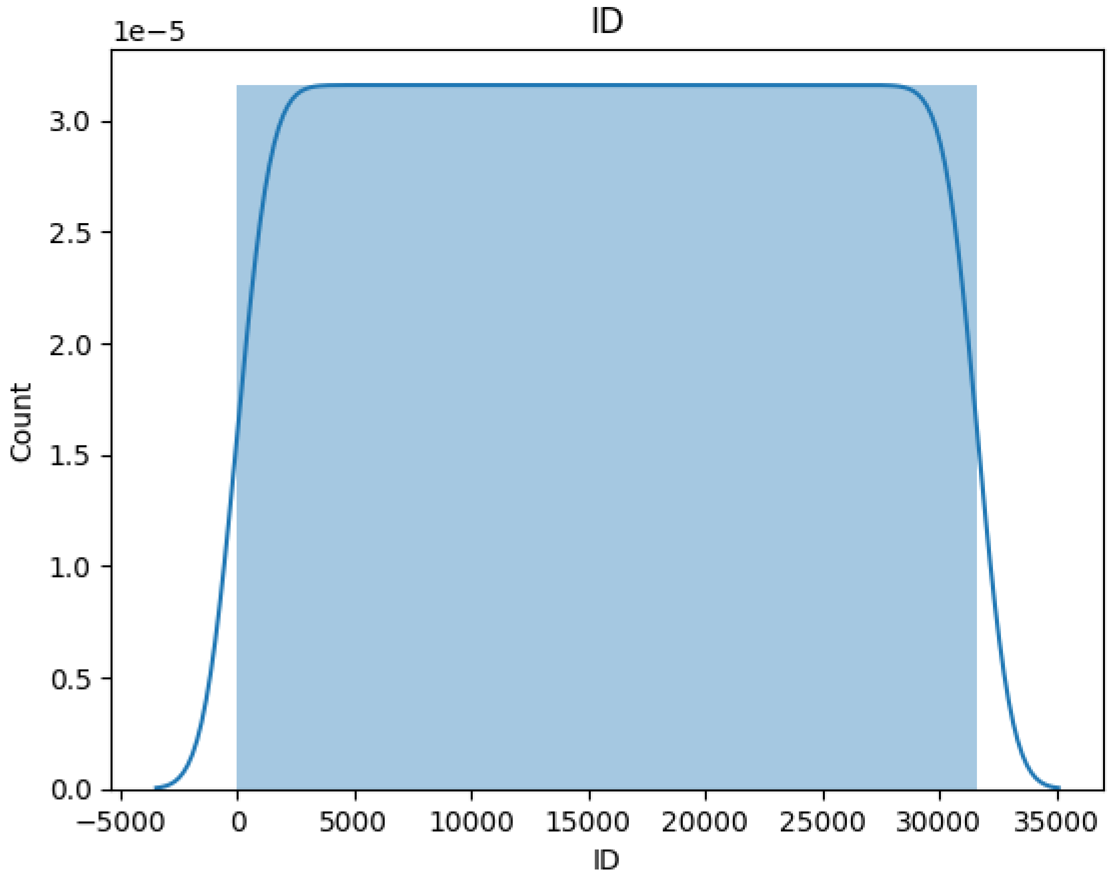
```
{0: 'BOOK', 1: 'CR', 2: 'MIXED', 3: 'MUSIC', 4: 'SOUNDCASS', 5: 'SOUNDDISC', 6: 'VIDEOCASS', 7: 'VIDEODISC'}
```

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

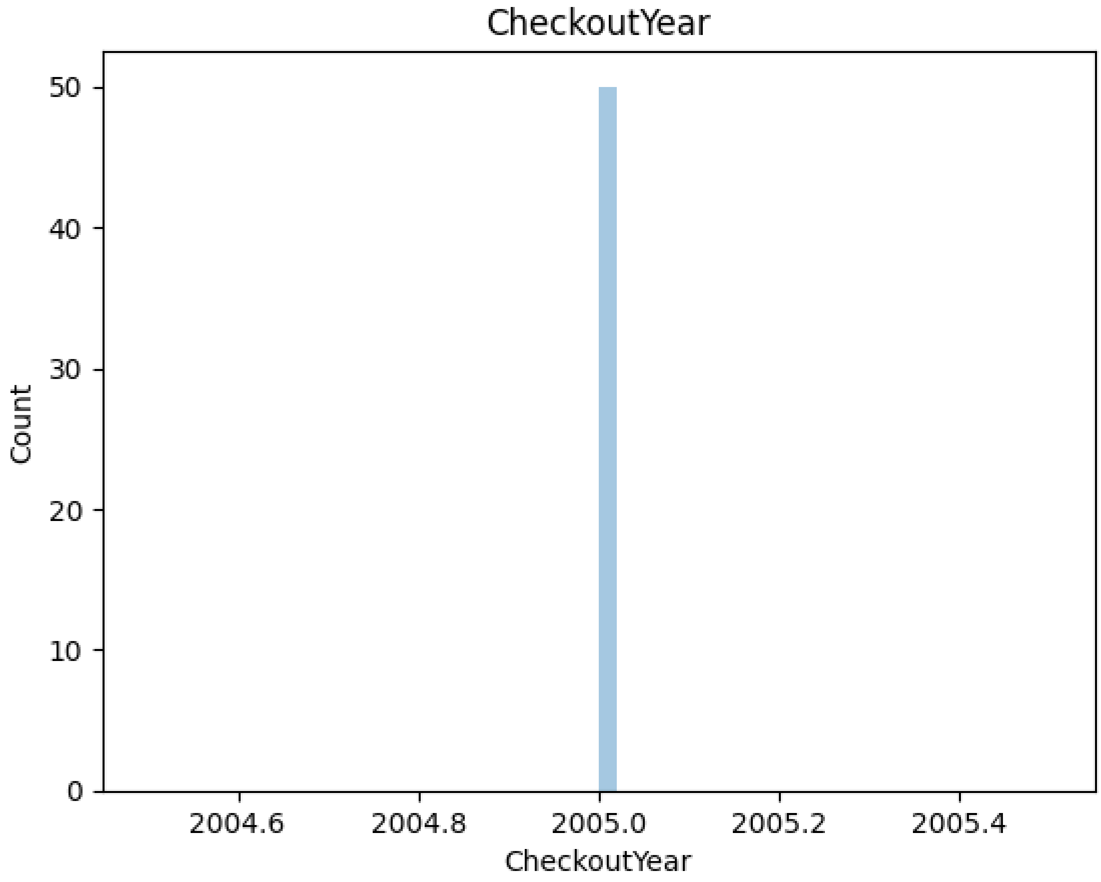
Analysis For `Numerical` Variables

```
#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()
```



```
<Figure size 300x300 with 0 Axes>
```



```
<Figure size 300x300 with 0 Axes>
```

```
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,
									Cinco de	

```
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)
```

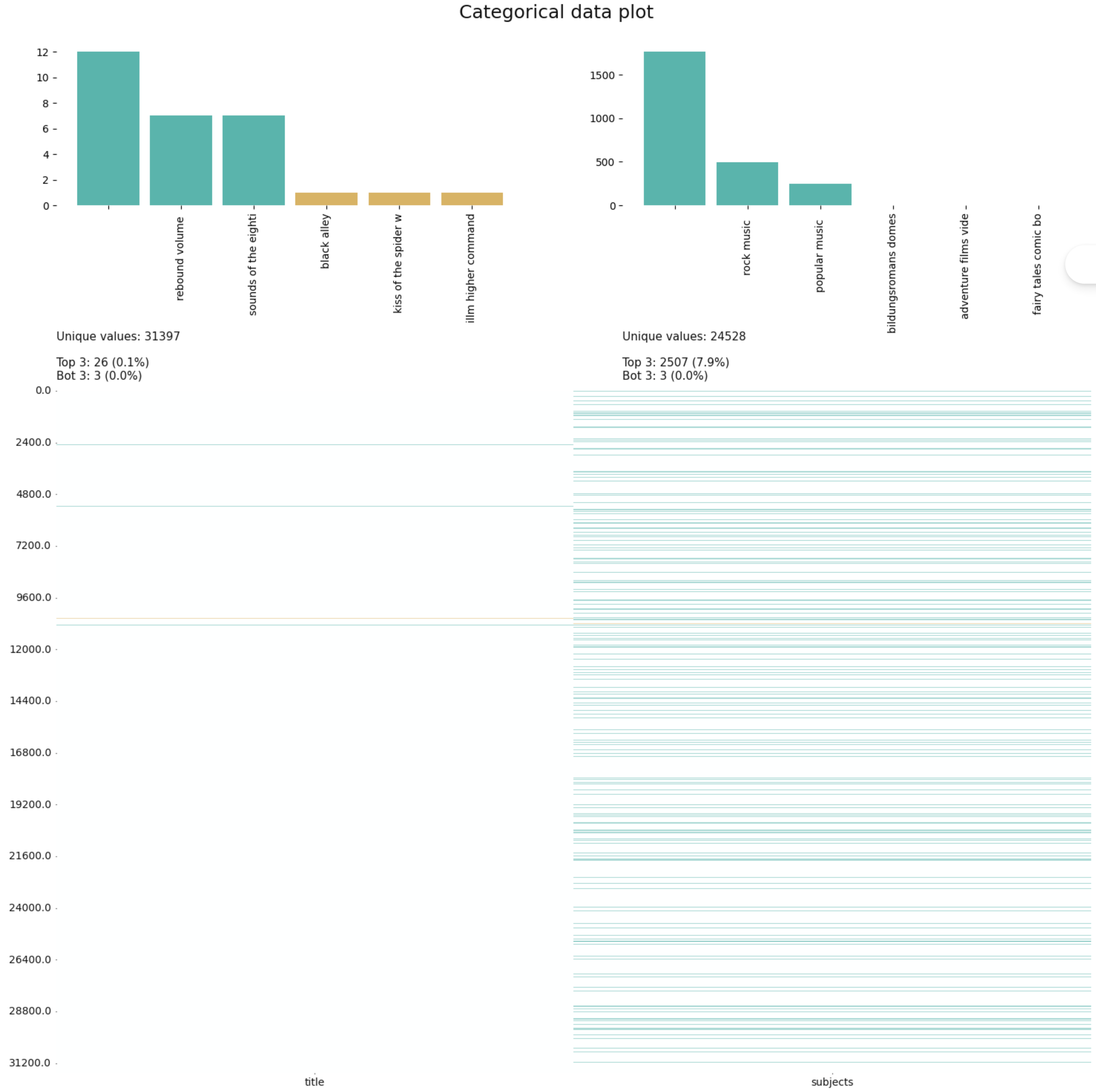
```
train_data.columns
```

```
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)
```

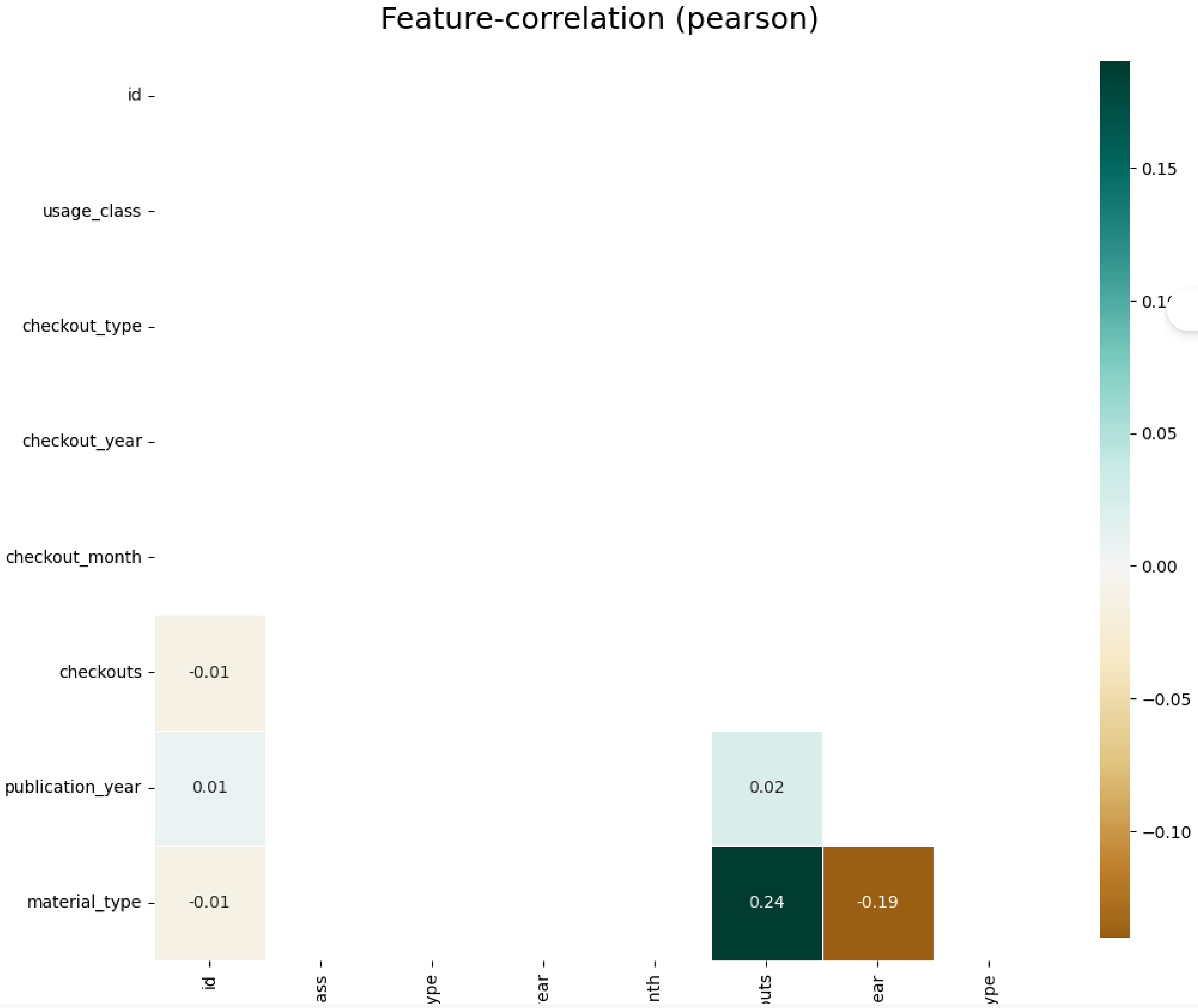


```
klib.corr_mat(train_data)
```

	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	

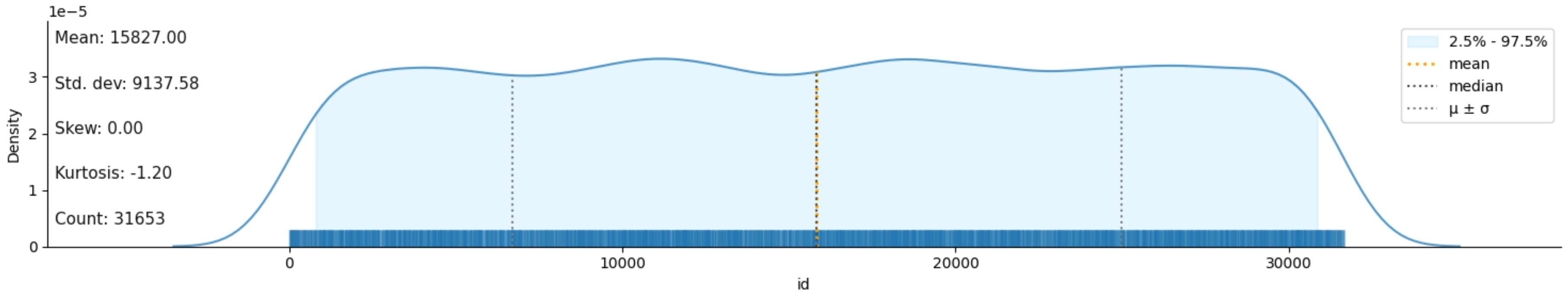
```
klib.corr_plot(train_data)
```

```
<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()
```

```
      'moving',
      'libraries',
      'knighthood',
      'inspector',
      'influences',
      'hong',
      'foods',
      'flight',
      'economics',
      'desert',
      'brain',
      'baltimore',
      'atlantic',
      'aeronautics',
      'walker',
      '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
```

```
     (16, 1800)
```

```
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(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
# Final layer
model.add(Dense(8, activation='sigmoid'))
```

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

```
model.summary()
```

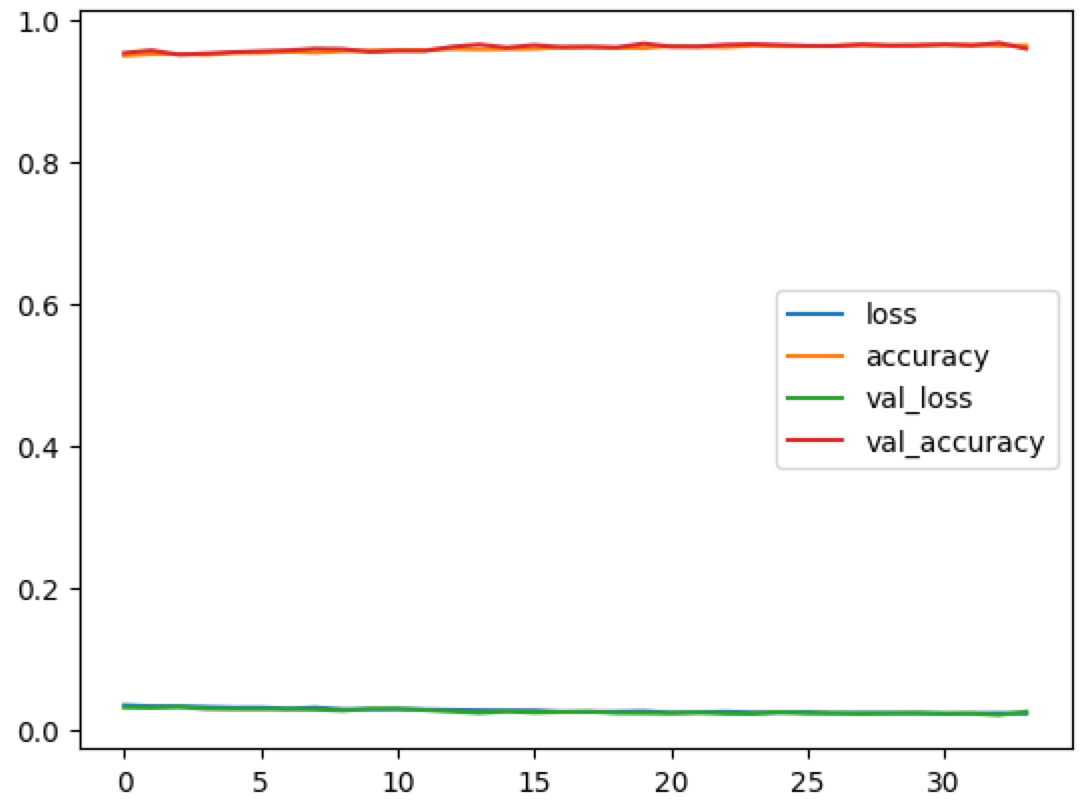
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding (Embedding)       (None, None, 32)          6400032   
                                                                 
 bidirectional (Bidirection  (None, 64)                16640     
 al)                                                             
                                                                 
 dense (Dense)               (None, 128)               8320      
                                                                 
 dense_1 (Dense)             (None, 256)               33024     
                                                                 
 dense_2 (Dense)             (None, 128)               32896     
                                                                 
 dense_3 (Dense)             (None, 8)                 1032      
                                                                 
=================================================================
Total params: 6491944 (24.76 MB)
Trainable params: 6491944 (24.76 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history = model.fit(train, epochs=34, validation_data = val)
```

```
1385/1385 [==============================] - 133s 96ms/step - loss: 0.0323 - accuracy: 0.9538 - val_loss: 0.0301 - v
Epoch 7/34
1385/1385 [==============================] - 136s 98ms/step - loss: 0.0308 - accuracy: 0.9552 - val_loss: 0.0301 - v
Epoch 8/34
1385/1385 [==============================] - 129s 93ms/step - loss: 0.0323 - accuracy: 0.9540 - val_loss: 0.0294 - v
Epoch 9/34
1385/1385 [==============================] - 131s 94ms/step - loss: 0.0298 - accuracy: 0.9559 - val_loss: 0.0281 - v
Epoch 10/34
Epoch 16/34
1385/1385 [==============================] - 129s 93ms/step - loss: 0.0281 - accuracy: 0.9593 - val_loss: 0.0253 - v
Epoch 17/34
1385/1385 [==============================] - 132s 96ms/step - loss: 0.0263 - accuracy: 0.9615 - val_loss: 0.0261 - v
Epoch 18/34
1385/1385 [==============================] - 130s 94ms/step - loss: 0.0263 - accuracy: 0.9623 - val_loss: 0.0267 - v
Epoch 19/34
1385/1385 [==============================] - 128s 93ms/step - loss: 0.0265 - accuracy: 0.9608 - val_loss: 0.0247 - v
Epoch 20/34
1385/1385 [==============================] - 128s 92ms/step - loss: 0.0272 - accuracy: 0.9603 - val_loss: 0.0244 - v
Epoch 21/34
1385/1385 [==============================] - 129s 93ms/step - loss: 0.0251 - accuracy: 0.9634 - val_loss: 0.0242 - v
Epoch 22/34
1385/1385 [==============================] - 128s 93ms/step - loss: 0.0256 - accuracy: 0.9624 - val_loss: 0.0254 - v
Epoch 23/34
1385/1385 [==============================] - 129s 93ms/step - loss: 0.0264 - accuracy: 0.9618 - val_loss: 0.0237 - 
Epoch 24/34
1385/1385 [==============================] - 130s 94ms/step - loss: 0.0253 - accuracy: 0.9640 - val_loss: 0.0234 - v
Epoch 25/34
1385/1385 [==============================] - 129s 93ms/step - loss: 0.0252 - accuracy: 0.9631 - val_loss: 0.0261 - v
Epoch 26/34
1385/1385 [==============================] - 129s 93ms/step - loss: 0.0258 - accuracy: 0.9631 - val_loss: 0.0241 - v
Epoch 27/34
1385/1385 [==============================] - 129s 93ms/step - loss: 0.0243 - accuracy: 0.9633 - val_loss: 0.0244 - v
Epoch 28/34
1385/1385 [==============================] - 130s 94ms/step - loss: 0.0249 - accuracy: 0.9642 - val_loss: 0.0232 - v
Epoch 29/34
1385/1385 [==============================] - 129s 93ms/step - loss: 0.0245 - accuracy: 0.9634 - val_loss: 0.0242 - v
Epoch 30/34
1385/1385 [==============================] - 129s 93ms/step - loss: 0.0244 - accuracy: 0.9642 - val_loss: 0.0247 - v
Epoch 31/34
1385/1385 [==============================] - 130s 94ms/step - loss: 0.0237 - accuracy: 0.9657 - val_loss: 0.0240 - v
Epoch 32/34
1385/1385 [==============================] - 134s 97ms/step - loss: 0.0241 - accuracy: 0.9646 - val_loss: 0.0239 - v
Epoch 33/34
1385/1385 [==============================] - 129s 93ms/step - loss: 0.0244 - accuracy: 0.9638 - val_loss: 0.0219 - v
Epoch 34/34
1385/1385 [==============================] - 130s 94ms/step - loss: 0.0238 - accuracy: 0.9640 - val_loss: 0.0265 - v
```

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

```
<Figure size 1500x1200 with 0 Axes>
```



```
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 [==============================] - 25s 39ms/step
```

```
Y_pred
```

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

```
Y_pred.shape
```

```
##
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.value_counts()
```

```
0    15409
5     2354
6     1778
7      823
4      550
3       87
2       63
1       38
dtype: int64
```

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

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

```
new_Y_pred.value_counts()
```

```
PREDICT
0          15409
5           2354
6           1778
7            823
4            550
3             87
2             63
1             38
dtype: int64
```

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

```
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:  73.02
```

```
new_Y_pred.value_counts()
```

```
PREDICT
0          15409
5           2354
6           1778
7            823
```

```
4          550
3           87
2           63
1           38
dtype: int64
```

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

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

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

```
BOOK         15409
SOUNDDISC     2354
VIDEOCASS     1778
VIDEODISC      823
SOUNDCASS      550
MUSIC           87
MIXED           63
CR              38
Name: PREDICT, dtype: int64
```

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

```
array(['SOUNDDISC', 'BOOK', 'SOUNDCASS', 'VIDEOCASS', 'VIDEODISC',
       'MUSIC', 'MIXED', 'CR'], 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]
```

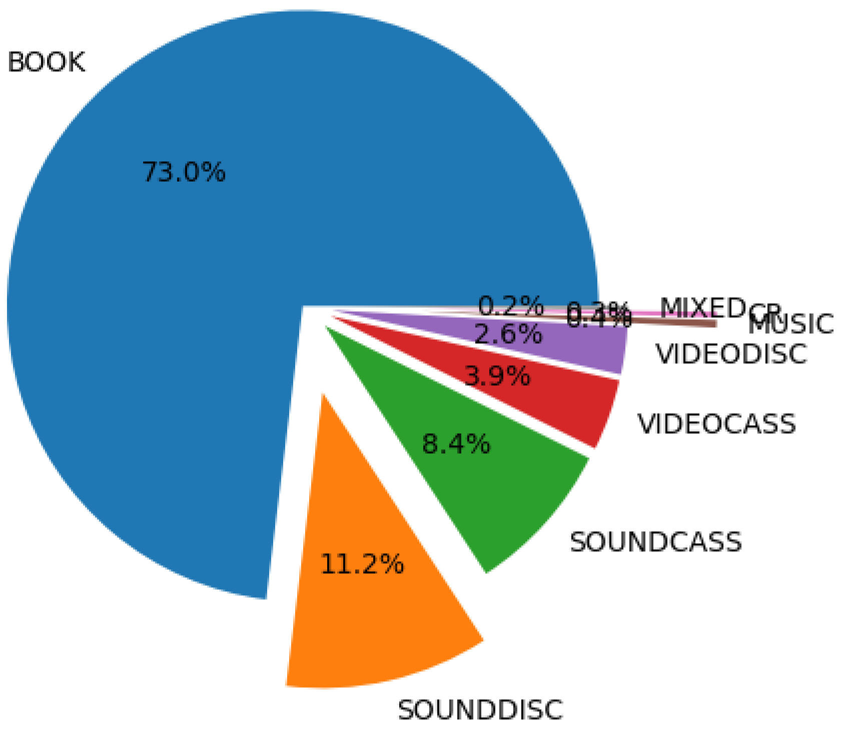
```
all = new_Y_pred["PREDICT"].value_counts()
```

```
plt.figure(figsize = [15,8])
```

```
<Figure size 1500x800 with 0 Axes>
<Figure size 1500x800 with 0 Axes>
```

```
# plotting data on chart
plt.pie(all, labels=keys, explode=explode, autopct='%.1f%%')

# displaying chart
plt.show()
```



```
new_Y_pred.to_csv("Predict.csv")
```

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