```
### 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
```

```
Downloading ericsson-ml-challenge-materialtype-prediction.zip to /content
  0% 0.00/2.85M [00:00<?, ?B/s]
100% 2.85M/2.85M [00:00<00:00, 68.7MB/s]
```

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

```
Archive:  /content/ericsson-ml-challenge-materialtype-prediction.zip
  inflating: sample_submission.csv
  inflating: test_file.csv
  inflating: train_file.csv
```

```
###! 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	PublicationYear	MaterialType
0	1	Physical	Horizon	2005	4	1	Tidal wave	NaN	Tsunamis, Tsunamis Juvenile literature	NaN	NaN	BOOK
1	2	Physical	Horizon	2005	4	1	London holiday / Richard Peck.	Peck, Richard, 1934-	NaN	Viking,	1998.	BOOK
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,	c2002.	BOOK

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

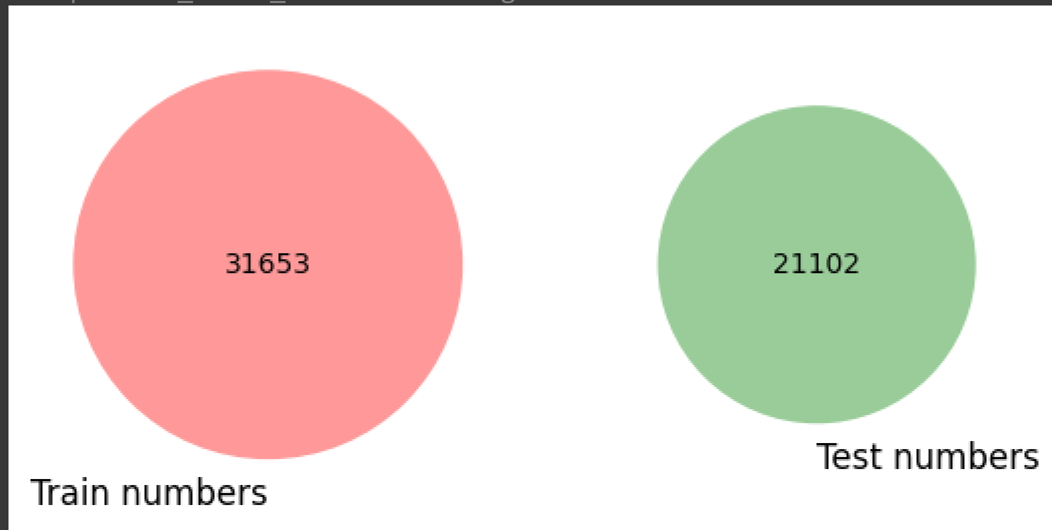
```
BOOK          21707
SOUNDDISC      4149
VIDEOCASS      2751
VIDEODISC      1420
SOUNDCASS      1020
MIXED           347
MUSIC           165
CR               94
Name: MaterialType, dtype: int64
```

```
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 0x7e3f73d54dc0>
```



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', 'PublicationYear', 'MaterialType']
```

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)
```

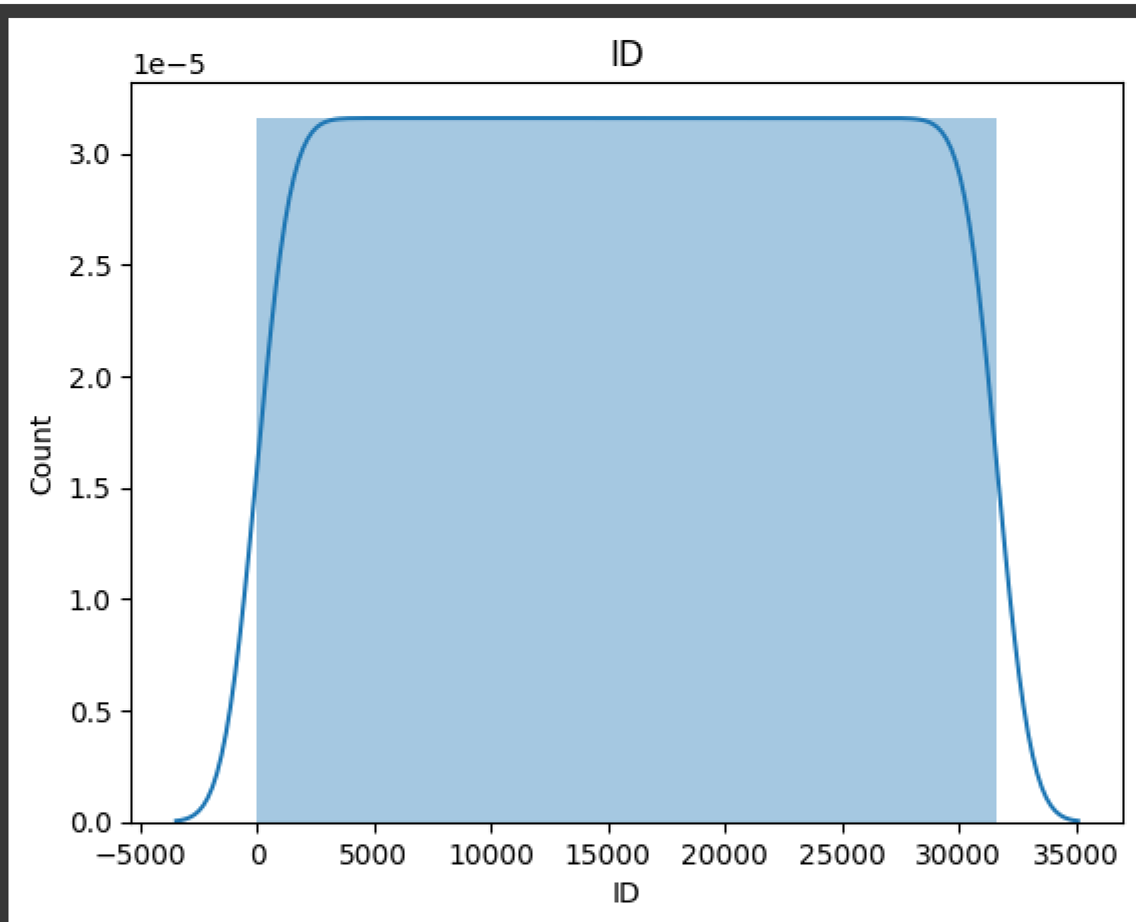
```
{0: 'BOOK', 1: 'CR', 2: 'MIXED', 3: 'MUSIC', 4: 'SOUNDCASS', 5: 'SOUNDDISC', 6: 'VIDEOCASS', 7: 'VIDEODISC'}
```

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

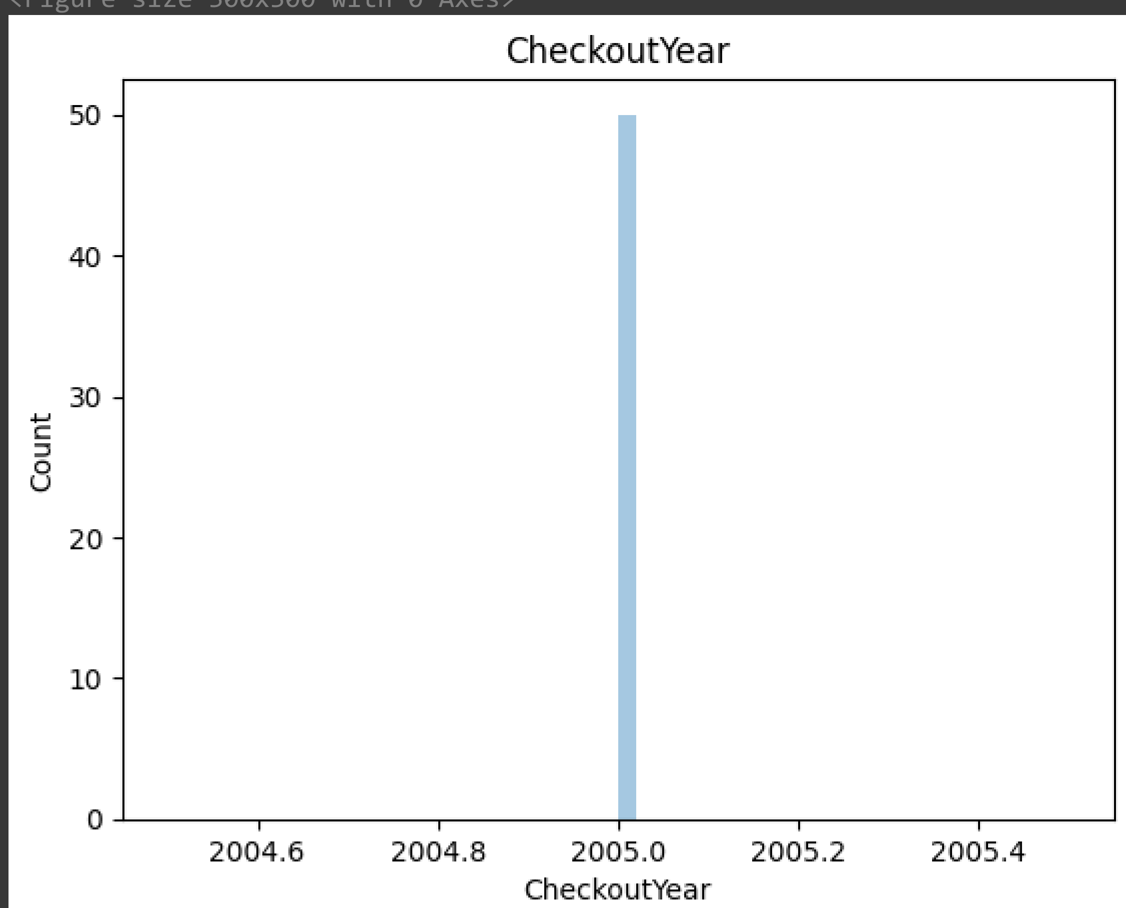
Analysis For `Numerical` Variables

```
#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()
```

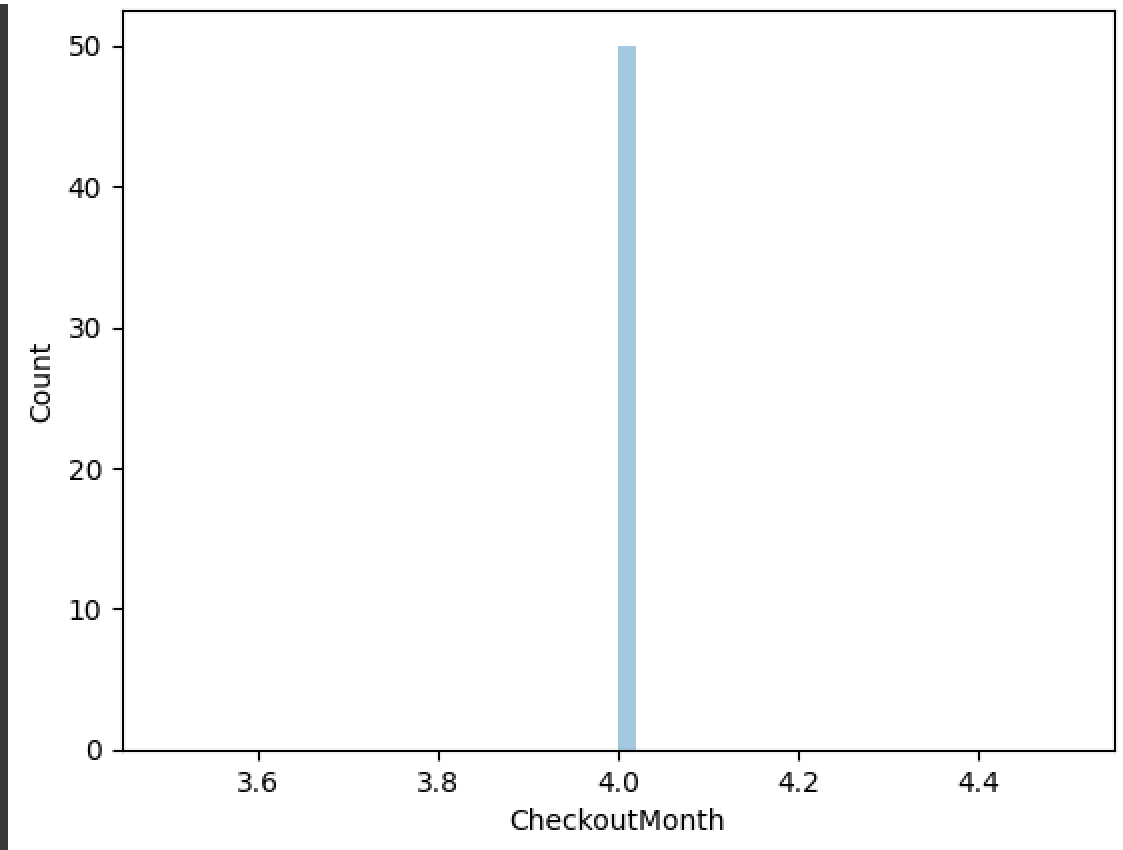



```
<Figure size 300x300 with 0 Axes>
```

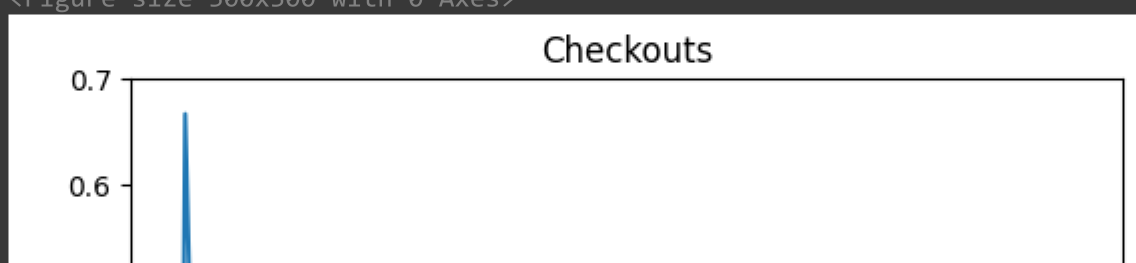



```
<Figure size 300x300 with 0 Axes>
```

CheckoutMonth




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




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

	ID	UsageClass	CheckoutType	CheckoutYear	CheckoutMonth	Checkouts	Title	Creator	Subjects	Publisher	PublicationYear	MaterialType
0	1	Physical	Horizon	2005	4	1	Tidal wave	0	Tsunamis, Tsunamis Juvenile literature	0	0	0
1	2	Physical	Horizon	2005	4	1	London holiday / Richard Peck.	Peck, Richard, 1934-	0	Viking,	1998.	0
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,	c2002.	0

```
###! 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	publication_year	material_type
0	1	Physical	Horizon	2005	4	1	Tidal wave	0	Tsunamis, Tsunamis Juvenile literature	0	0	0

```
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	publication_year	material_type
0	1	0	0	2005	4	1	Tidal wave	0	Tsunamis, Tsunamis Juvenile literature	0	0	0
1	2	0	0	2005	4	1	London holiday / Richard Peck.	Peck, Richard, 1934-	0	Viking,	1998.	0
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,	c2002.	0

```
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	publication_year	material_type
0	1	0	0	2005	4	1	Tidal wave	0	Tsunamis, Tsunamis Juvenile literature	0	0	0
1	2	0	0	2005	4	1	London holiday / Richard Peck.	Peck, Richard, 1934-	0	Viking,	25	0

```
###!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)
```

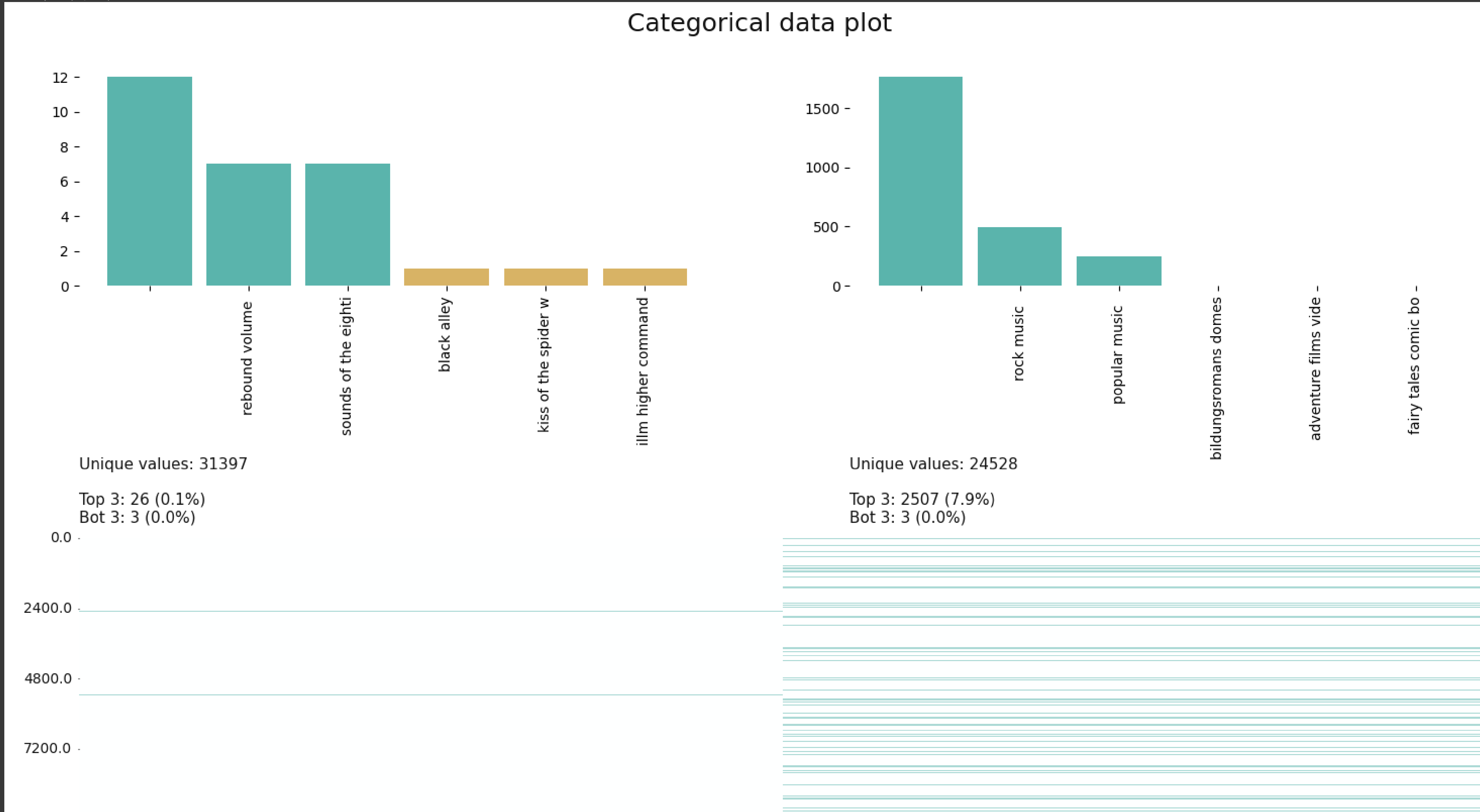
```
train_data.columns
```

```
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)
```

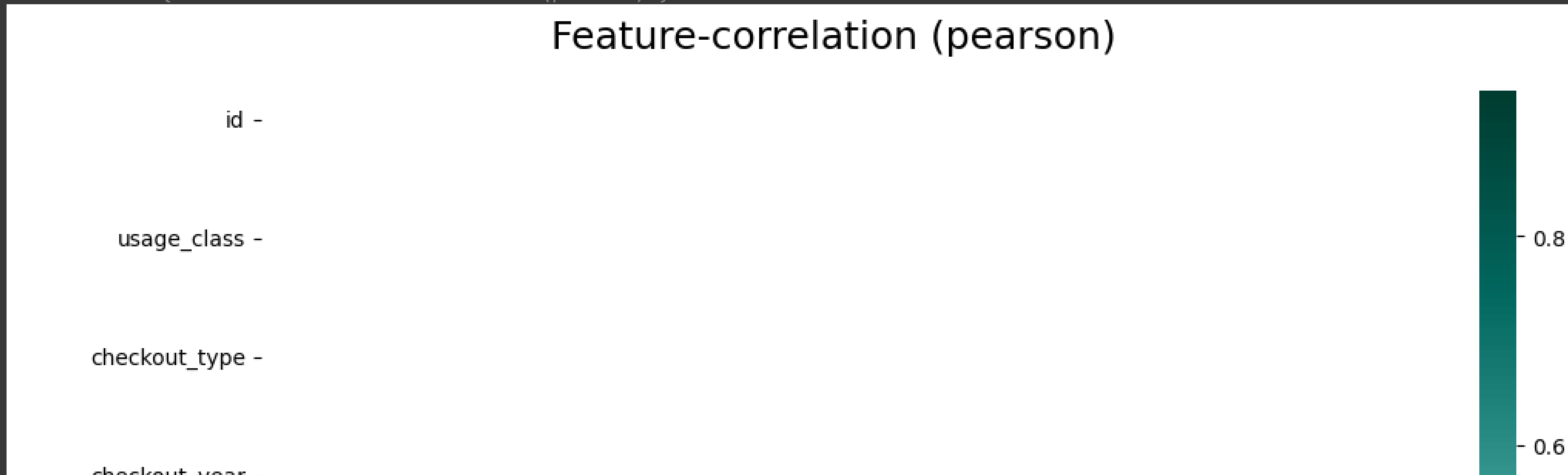



```
klib.corr_mat(train_data)
```

	id	usage_class	checkout_type	checkout_year	checkout_month	checkouts	publication_year	material_type	subjects_len	word_count_subjects
id	1.00	-	-	-	-	-0.01	0.01	-0.01	0.00	0.00
usage_class	-	-	-	-	-	-	-	-	-	-
checkout_type	-	-	-	-	-	-	-	-	-	-
checkout_year	-	-	-	-	-	-	-	-	-	-
checkout_month	-	-	-	-	-	-	-	-	-	-
checkouts	-0.01	-	-	-	-	1.00	0.02	0.24	0.09	0.10
publication_year	0.01	-	-	-	-	0.02	1.00	-0.19	0.02	0.01
material_type	-0.01	-	-	-	-	0.24	-0.19	1.00	-0.02	0.01

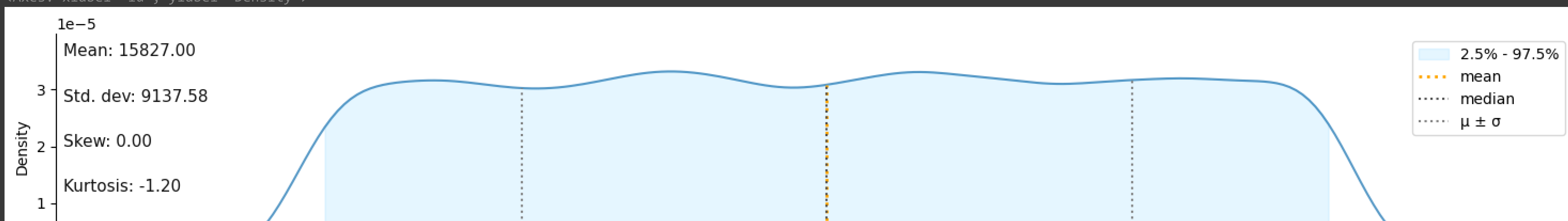
```
klib.corr_plot(train_data)
```

```
<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 dataset.
<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
```

```
tfidf = TfidfVectorizer(max_features = 500,
                              ngram_range = (1,3),
                              stop_words = "english")
train_subjects = tfidf.fit_transform(train_data["subjects"].tolist())
```

```
test_subjects = tfidf.fit_transform(test_data["subjects"].tolist())
```

```
train_title = tfidf.fit_transform(train_data["title"].tolist())
```

```
test_title = tfidf.fit_transform(test_data["title"].tolist())
```

```
train_data.dtypes
```

```
id                   int16
usage_class           int8
checkout_type         int8
checkout_year        int16
checkout_month        int8
checkouts             int8
title               object
subjects            object
publication_year     int64
material_type         int8
dtype: object
```

```
train_data['subjects_len'] = train_data['subjects'].astype(str).apply(len)
train_data['word_count_subjects'] = train_data['subjects'].apply(lambda x: len(str(x).split()))
```

```
test_data['subjects_len'] = test_data['subjects'].astype(str).apply(len)
test_data['word_count_subjects'] = test_data['subjects'].apply(lambda x: len(str(x).split()))
```

```
train_data.columns
```

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

```
import scipy
```

```
X_train = scipy.sparse.hstack((train_subjects, train_title,
                         train_data[["id", "usage_class", "checkout_type", "checkout_year", "checkout_month",
       "checkouts", "subjects_len", "word_count_subjects"]].to_numpy())).tocsr()
```

```
X_test = scipy.sparse.hstack((test_subjects, test_title,
                         test_data[["id", "usage_class", "checkout_type", "checkout_year", "checkout_month",
       "checkouts", "subjects_len", "word_count_subjects"]].to_numpy())).tocsr()
```

```
y_train = train_data['material_type']
y_test = test_data['material_type']
```

```
X_train = pd.DataFrame(X_train.todense())
```

```
X_test = pd.DataFrame(X_test.todense())
```

```
print(X_train.shape, X_test.shape)
```

```
(31653, 1008) (21102, 1008)
```

```
y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)
```

```
print(y_train.shape, y_test.shape)
```

```
(31653, 1) (21102, 1)
```

XGBClassifier

```
###! pip install xgboost
```

```
from xgboost import XGBClassifier
```

```
xgb = XGBClassifier()
xgb.fit(X_train,y_train)
xgbpred = xgb.predict(X_test)
print(metrics.accuracy_score(y_test,xgbpred))
```

```
0.8847976495118947
```

```
from sklearn.metrics import accuracy_score, recall_score, classification_report, cohen_kappa_score
from sklearn import metrics
```

```
print('Baseline: Accuracy: ', round(accuracy_score(y_test, xgbpred)*100, 2))
print('\n Classification Report:\n', classification_report(y_test,xgbpred))
```

```
Baseline: Accuracy:  88.48
```

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

           0       1.00      0.88      0.94     21102
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0
           5       0.00      0.00      0.00         0
           6       0.00      0.00      0.00         0
           7       0.00      0.00      0.00         0

    accuracy                           0.88     21102
   macro avg       0.14      0.13      0.13     21102
weighted avg       1.00      0.88      0.94     21102
```

```
from sklearn.metrics import accuracy_score as acc_score
```

```
from sklearn.metrics import accuracy_score as accuracy
accuracy = make_scorer(accuracy, greater_is_better=False)
```

```
import time
```

XGBClassifier with `Bayesian Optimization`

```
def xgb_cl_bo(min_child_weight, gamma, subsample, colsample_bytree, max_depth):
    params_xgb = {}
    params_xgb['min_child_weight'] = min_child_weight
    params_xgb['gamma'] = gamma
    params_xgb['subsample'] = subsample
    params_xgb['colsample_bytree'] = int(colsample_bytree)
    params_xgb['max_depth'] = int(max_depth)
    scores = cross_val_score(XGBClassifier(random_state=123, **params_xgb),
                             X_train, y_train, scoring=accuracy, cv=4).mean()
    score = scores.mean()
    score = -score
    return score
# Run Bayesian Optimization
start = time.time()
params_xgb ={
    'min_child_weight':(1, 10),
    'gamma':(0.5, 10),
    'subsample':(0.6, 1.0),
    'colsample_bytree':(0.6, 1.0),
    'max_depth': (3, 10)
}
```

```
xgb_bo = BayesianOptimization(xgb_cl_bo, params_xgb)
xgb_bo.maximize(init_points=12, n_iter=4)
print('It takes %s minutes' % ((time.time() - start)/60))
```

```
|   iter    |  target   | colsam... |   gamma   | max_depth | min_ch... | subsample |
-------------------------------------------------------------------------------------
| 1         | 0.7158    | 0.6293    | 9.017     | 6.815     | 8.96      | 0.9135    |
| 2         | 0.7168    | 0.7273    | 9.475     | 9.777     | 1.724     | 0.9527    |
| 3         | 0.7172    | 0.9785    | 6.255     | 8.997     | 1.545     | 0.6176    |
| 4         | 0.7156    | 0.7794    | 5.176     | 3.523     | 8.818     | 0.6412    |
| 5         | 0.716     | 0.876     | 9.844     | 8.295     | 2.638     | 0.745     |
| 6         | 0.7155    | 0.7789    | 8.788     | 5.659     | 5.765     | 0.6004    |
| 7         | 0.7161    | 0.6412    | 7.975     | 5.588     | 2.092     | 0.7848    |
| 8         | 0.7182    | 0.7557    | 6.66      | 8.121     | 9.383     | 0.8889    |
| 9         | 0.7196    | 0.915     | 4.882     | 8.633     | 1.914     | 0.879     |
| 10        | 0.7194    | 0.879     | 4.248     | 8.435     | 3.525     | 0.7445    |
| 11        | 0.7188    | 0.874     | 5.917     | 9.242     | 2.438     | 0.926     |
| 12        | 0.7161    | 0.6753    | 8.648     | 4.995     | 2.547     | 0.8598    |
| 13        | 0.7191    | 0.6       | 3.677     | 10.0      | 2.361     | 1.0       |
| 14        | 0.8785    | 1.0       | 2.728     | 7.442     | 1.543     | 1.0       |
| 15        | 0.8785    | 1.0       | 2.105     | 6.947     | 1.219     | 1.0       |
| 16        | 0.8759    | 1.0       | 1.283     | 7.673     | 2.211     | 1.0       |
=====================================================================================
It takes 37.373470834891 minutes
```

```
params_xgb = xgb_bo.max['params']
params_xgb['max_depth'] = round(params_xgb['max_depth'])
params_xgb['min_child_weight'] = round(params_xgb['min_child_weight'])
params_xgb['gamma'] = round(params_xgb['gamma'])
params_xgb['colsample_bytree'] = round(params_xgb['colsample_bytree'])
params_xgb['subsample'] = round(params_xgb['subsample'])
params_xgb
```

```
{'colsample_bytree': 1,
 'gamma': 3,
 'max_depth': 7,
 'min_child_weight': 2,
 'subsample': 1}
```

```
xgb_hyp2 =  XGBClassifier(**params_xgb, random_state=123)
```

```
xgb_hyp2.fit(X_train, y_train)
```

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None, colsample_bytree=1,
```

```
# Predict the validation data
pred_xgb2 = xgb_hyp2.predict(X_test)

# Compute the accuracy
print('Accuracy: ' + str(accuracy_score(y_test, pred_xgb2)))
```

```
Accuracy: 0.8857928158468391
```

```
pred_xgb2 = pd.DataFrame(pred_xgb2)
```

```
pred_xgb2.rename(columns = {0 : "MaterialType_Pred"}, inplace=True)
```

```
print('Baseline: Accuracy: ', round(accuracy_score(y_test, pred_xgb2)*100, 2))
print('\n Classification Report:\n', classification_report(y_test,pred_xgb2))
```

```
Baseline: Accuracy:  88.58

 Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.89      0.94     21102
           2       0.00      0.00      0.00         0
           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0
           5       0.00      0.00      0.00         0
           6       0.00      0.00      0.00         0
           7       0.00      0.00      0.00         0

    accuracy                           0.89     21102
   macro avg       0.14      0.13      0.13     21102
weighted avg       1.00      0.89      0.94     21102
```

OPTUNA - Random Forest Classifier

```
###! pip install optuna
```

```
import optuna
import sklearn
```

```
param_grid_optuna = {
    "bootstrap": [True, False],
    "max_depth": [10, 20, 30, 40, 50, None],
    "max_features": ['auto', 'sqrt'],
    "min_samples_leaf": [1, 2, 4],
    "min_samples_split": [2, 5, 10, 15],
    "n_estimators": [200, 400, 600]
}
```

```
from sklearn.model_selection import cross_val_score
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
def objective(trial):
    bootstrap = trial.suggest_categorical('bootstrap',[True, False])
    max_depth = trial.suggest_int('max_depth', 10, 100)
    max_features = trial.suggest_categorical('max_features', ['auto', 'sqrt'])
    min_samples_leaf = trial.suggest_int('min_samples_leaf', 1, 4)
    min_samples_split = trial.suggest_int('min_samples_split', 2, 6)
    n_estimators =  trial.suggest_int('n_estimators', 200, 300)

    clsr = RandomForestClassifier(bootstrap = bootstrap,
                             max_depth = max_depth, max_features = max_features,min_samples_leaf = min_samples_leaf,
                             min_samples_split = min_samples_split,n_estimators = n_estimators)


    #regr.fit(X_train, y_train)
    #y_pred = regr.predict(X_val)
    #return r2_score(y_val, y_pred)

    score = cross_val_score(clsr, X_train, y_train, cv=5, n_jobs=-1)
    meanvalue = score.mean()

    return meanvalue
```

```
#Execute optuna and set hyperparameters
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=4)
```

```
[I 2023-12-21 15:54:14,244] A new study created in memory with name: no-name-429c7ef0-ca99-42af-ad92-7b87466aea7b
[I 2023-12-21 15:58:59,229] Trial 0 finished with value: 0.8758411517538389 and parameters: {'bootstrap': True, 'max_depth': 56, 'max_features': 'sqrt', 'min_samples_leaf': 3, 'min_samples_spli
[I 2023-12-21 16:03:37,166] Trial 1 finished with value: 0.8716400080548509 and parameters: {'bootstrap': True, 'max_depth': 64, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_spli
[I 2023-12-21 16:11:56,530] Trial 2 finished with value: 0.8163235644561491 and parameters: {'bootstrap': False, 'max_depth': 93, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_spl
[I 2023-12-21 16:15:02,532] Trial 3 finished with value: 0.8788739627944743 and parameters: {'bootstrap': True, 'max_depth': 38, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_spli
```

```
#Create an instance with tuned hyperparameters
optimised_rf_optuna = RandomForestClassifier(bootstrap = study.best_params['bootstrap'],
                                     max_depth = study.best_params['max_depth'], max_features = study.best_params['max_features'],
                                     min_samples_leaf = study.best_params['min_samples_leaf'],
                                     min_samples_split = study.best_params['min_samples_split'],
                                     n_estimators = study.best_params['n_estimators'])
#learn
optimised_rf_optuna.fit(X_train ,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=38, max_features='auto', min_samples_split=3,
                       n_estimators=247)
```

```
trial = study.best_trial
print('Accuracy: {}'.format(trial.value))
```

```
Accuracy: 0.8788739627944743
```

```
y_pred_optuna = optimised_rf_optuna.predict(X_test)
```

```
print('Baseline: Accuracy: ', round(accuracy_score(y_test, y_pred_optuna)*100, 2))
print('\n Classification Report:\n', classification_report(y_test,y_pred_optuna))
```

```
Baseline: Accuracy:  89.75

 Classification Report:
               precision    recall  f1-score   support

           0       1.00      0.90      0.95     21102
           2       0.00      0.00      0.00         0
           5       0.00      0.00      0.00         0
           6       0.00      0.00      0.00         0
           7       0.00      0.00      0.00         0

    accuracy                           0.90     21102
   macro avg       0.20      0.18      0.19     21102
weighted avg       1.00      0.90      0.95     21102
```

```
y_pred_optuna = pd.DataFrame(y_pred_optuna)
y_pred_optuna.rename(columns = {0: "MaterialType_Pred2"}, inplace=True)
```

```
y_pred_optuna.value_counts()
```

```
MaterialType_Pred2
0                     18940
6                      1306
7                       496
5                       338
```

```
2                          22
dtype: int64
```

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

```
pred_xgb2.value_counts()
```

```
MaterialType_Pred
0                    18692
6                     1351
7                      578
5                      311
2                      135
4                       32
3                        3
dtype: int64
```

```
pred_xgb2["MaterialType_Pred"] = pred_xgb2["MaterialType_Pred"].map(class_names)
```

```
y_pred_optuna["MaterialType_Pred2"] = y_pred_optuna["MaterialType_Pred2"].map(class_names)
```

```
pred_xgb2.value_counts()
```

```
MaterialType_Pred
BOOK                 18692
VIDEOCASS             1351
VIDEODISC              578
SOUNDDISC              311
MIXED                  135
SOUNDCASS               32
MUSIC                    3
dtype: int64
```

```
test_data["MaterialType_Pred"] = pred_xgb2
```

```
test_data2 = test_data[["id", "MaterialType_Pred"]]
```

```
test_data2.to_csv("test_data2.csv")
```

```
submission = pd.read_csv("/content/sample_submission.csv")
```

```
print(submission.shape, test_data2.shape)
```

```
(6, 2) (21102, 2)
```

```
y_pred_optuna.value_counts()
```

```
MaterialType_Pred2
BOOK            18940
VIDEOCASS        1306
VIDEODISC         496
SOUNDDISC         338
MIXED              22
dtype: int64
```

```
test_data["MaterialType_Pred2"] = y_pred_optuna
```

```
test_data3 = test_data[["id", "MaterialType_Pred2"]]
```

```
test_data3.to_csv("test_data3.csv")
```

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