KKBOX's Music Recommendation Challange

In [ ]:

```
# Importing basic libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import missingno as msno
import time
import gc
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, VotingClassifier, StackingC
lassifier
from sklearn.model_selection import RandomizedSearchCV
from sklearn.feature_selection import SelectKBest, chi2, f_classif
from sklearn.decomposition import PCA
import xgboost
import lightgbm as lgb
from sklearn import tree
import seaborn as sns
import matplotlib.pyplot as plt
import time
# Deep learning libraries
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, Input, Model
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from sklearn.metrics import roc_auc_score
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.p
y:19: FutureWarning: pandas.util.testing is deprecated. Use the fun
ctions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

1. Apply Models

- After EDA and FE we will try different machine learning algorithms on our data points.

Sampling

- As we have very large data size which cannot fit into the RAM during execution of algorithms.
- So we will take sample from the dataset and apply all algorithms on top of those samples.

In [ ]:

```
train_all = pd.read_csv('/content/drive/My Drive/CS-1/train_all_new.csv')
val_all = pd.read_csv('/content/drive/My Drive/CS-1/val_all_new.csv')
test_all = pd.read_csv('/content/drive/My Drive/CS-1/test_all_new.csv')
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshel
l.py:2718: DtypeWarning: Columns (11) have mixed types.Specify dtyp
e option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

In [ ]:

```
print(train_all.shape, val_all.shape, test_all.shape)
```

```
(5901934, 45) (1475484, 45) (2556790, 45)
```

- As we can see that we have nearlly about 6M data points into the train data and around 1.5M data points in val dataset and 2.5M data pointss in test dataset.
- We will take 1.5M data points from train data and 0.7M data points from val data.
- As our data is in the chronological order so we will take last points from train data and starting oints from val data.
- We will take test data as it is because we have to predict true lables fro them to submit the score into kaggle and we need not to perform hyper parameter tuning on them.

In [ ]:

```
train_all.isnull().any()
```

Out[ ]:

```
msno                     False
song_id                  False
source_system_tab        False
source_screen_name       False
source_type              False
target                   False
city                     False
bd                       False
gender                   False
registered_via           False
song_length              False
language                 False
name                     False
membership_days          False
registration_year        False
registration_month       False
registration_day         False
expiration_year          False
expiration_month         False
expiration_day           False
first_genre_id           False
second_genre_id          False
third_genre_id           False
genre_ids_count          False
is_featured              False
artist_count             False
first_artist_name         True
lyricist_count           False
first_lyricist            True
composer_count           False
first_composer            True
song_lang_boolean        False
song_size_boolean        False
country_code             False
registration_code        False
song_year                False
isrc_missing             False
member_song_count        False
artist_song_count        False
composer_song_count      False
lyricist_song_count      False
genre_song_count         False
lang_song_count          False
song_member_count        False
age_song_count           False
dtype: bool
```

In [ ]:

```
train_all.drop(['first_lyricist', 'first_composer'], axis=1, inplace=True)
val_all.drop(['first_lyricist', 'first_composer'], axis=1, inplace=True)
test_all.drop(['first_lyricist', 'first_composer'], axis=1, inplace=True)
```

In [ ]:

```
def fill_missing_values(data):
  '''Function to fill missing values'''
  data['first_artist_name'].fillna('no_artist_name', inplace=True)
  #data['first_lyricist'].fillna('no_lyricist', inplace=True)
  #data['first_composer'].fillna('no_composer', inplace=True)
  return data

train_all = fill_missing_values(train_all)
val_all = fill_missing_values(val_all)
test_all = fill_missing_values(test_all)
```

In [ ]:

```
# convert language values into string
train_all['language'] = train_all['language'].apply(lambda x: str(x))
val_all['language'] = val_all['language'].apply(lambda x: str(x))
test_all['language'] = test_all['language'].apply(lambda x: str(x))
```

In [ ]:

```
tr_data = train_all[5751934:] # 1.5M data points
#tr_data = train_all[3401934:] # 2.5M data points
val_data = val_all[:70000]

print('Sample size for train data is : ', tr_data.shape, ' and val data is : ',
val_data.shape)
```

```
Sample size for train data is :  (150000, 43)  and val data is :
(70000, 43)
```

In [ ]:

```
tr_data.to_csv('/content/drive/My Drive/CS-1/tr_data.csv', index=False)
val_data.to_csv('/content/drive/My Drive/CS-1/val_data.csv', index=False)
```

In [ ]:

```
del train_all, val_all
```

In [ ]:

```
gc.collect() # explicitely free memory
```

Out[ ]:

```
162
```

Pre-Processing

- We will use label encoding to convert categorical features.
- We will use standardization to scale numeric features.
- We will use only train data to fit to avoid data leakage.
- Some features are having NaN values which we will fill with '0' values.
- There are some values for features which are present in train data points byt not in val and test data points.

In [ ]:

```
test = len(set(test_all['msno']).difference(set(tr_data['msno'])))
val = len(set(val_data['msno']).difference(set(tr_data['msno'])))

print('Users from test : {0} {1}% and from val : {2} {3}% which are not present
 in train data'.format(test, (test/test_all.shape[0] * 100), val, (val/val_data.
shape[0] * 100)))
```

```
Users from test : 15215 0.5950821146828641% and from val : 2009 2.8
7% which are not present in train data
```

In [ ]:

```
tr_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 5751934 to 5901933
Data columns (total 44 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   msno                 150000 non-null  object 
 1   song_id              150000 non-null  object 
 2   source_system_tab    150000 non-null  object 
 3   source_screen_name   150000 non-null  object 
 4   source_type          150000 non-null  object 
 5   target               150000 non-null  int64  
 6   city                 150000 non-null  int64  
 7   bd                   150000 non-null  float64
 8   gender               150000 non-null  object 
 9   registered_via       150000 non-null  int64  
 10  song_length          150000 non-null  float64
 11  language             150000 non-null  object 
 12  name                 150000 non-null  object 
 13  membership_days      150000 non-null  int64  
 14  registration_year    150000 non-null  int64  
 15  registration_month   150000 non-null  int64  
 16  registration_day     150000 non-null  int64  
 17  expiration_year      150000 non-null  int64  
 18  expiration_month     150000 non-null  int64  
 19  expiration_day       150000 non-null  int64  
 20  first_genre_id       150000 non-null  float64
 21  second_genre_id      150000 non-null  float64
 22  third_genre_id       150000 non-null  float64
 23  genre_ids_count      150000 non-null  float64
 24  is_featured          150000 non-null  int64  
 25  artist_count         150000 non-null  int64  
 26  first_artist_name    149998 non-null  object 
 27  lyricist_count       150000 non-null  int64  
 28  first_lyricist       150000 non-null  object 
 29  composer_count       150000 non-null  int64  
 30  first_composer       149994 non-null  object 
 31  song_lang_boolean    150000 non-null  int64  
 32  song_size_boolean    150000 non-null  int64  
 33  country_code         150000 non-null  object 
 34  registration_code    150000 non-null  object 
 35  song_year            150000 non-null  object 
 36  isrc_missing         150000 non-null  float64
 37  member_song_count    150000 non-null  int64  
 38  artist_song_count    150000 non-null  int64  
 39  composer_song_count  150000 non-null  int64  
 40  lyricist_song_count  150000 non-null  int64  
 41  genre_song_count     150000 non-null  int64  
 42  lang_song_count      150000 non-null  int64  
 43  song_member_count    150000 non-null  int64  
dtypes: float64(7), int64(23), object(14)
memory usage: 50.4+ MB
```

Standardization

- Standardization will transform numeric values such that mean = 0 and std_dev = 1.

In [ ]:

```
numeric_features = ['bd', 'song_length', 'membership_days', 'genre_ids_count',
'artist_count', 'lyricist_count', \
                    'composer_count', 'member_song_count', 'artist_song_count',
'composer_song_count', 'lyricist_song_count', \
                    'genre_song_count', 'lang_song_count', 'song_member_count',
'age_song_count']
```

In [ ]:

```
# transform numeric values
pd.set_option('mode.chained_assignment', None)
for feature in numeric_features:
  scaler = StandardScaler()
  tr_data[feature] = scaler.fit_transform(tr_data[feature].values.reshape(-1,1))
  val_data[feature] = scaler.transform(val_data[feature].values.reshape(-1,1))
  test_all[feature] = scaler.transform(test_all[feature].values.reshape(-1,1))
```

Label-Encoding

In [ ]:

```
cat_features = ['msno', 'song_id', 'source_system_tab', 'source_screen_name', 's
ource_type', 'city', 'gender',\
                'registered_via', 'name', 'registration_year', 'registration_mon
th', 'registration_day',\
                'expiration_year', 'expiration_month', 'expiration_day', 'first_
genre_id', 'second_genre_id',\
                'third_genre_id', 'first_artist_name', 'country_code',\
                'registration_code','song_year', 'language']
```

- As LabelEncoder can not handle the unseen categories during transformation.
- So we will use train + val + test categories for LabelEncoder()

In [ ]:

```
# transform categorical values
pd.set_option('mode.chained_assignment', None)
for feature in cat_features:
  le = LabelEncoder()
  print(feature)
  combined = tr_data[feature].append(val_data[feature])
  combined = set(combined.append(test_all[feature]))
  combined = np.array(list(combined))
  le = le.fit(combined)
  tr_data[feature] = le.transform(tr_data[feature].values.reshape(-1,1))
  val_data[feature] = le.transform(val_data[feature].values.reshape(-1,1))
  test_all[feature] = le.transform(test_all[feature].values.reshape(-1,1))
```

```
msno
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/preprocessing/_labe
l.py:268: DataConversionWarning: A column-vector y was passed when
a 1d array was expected. Please change the shape of y to (n_sample
s, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
song_id
source_system_tab
source_screen_name
source_type
city
gender
registered_via
name
registration_year
registration_month
registration_day
expiration_year
expiration_month
expiration_day
first_genre_id
second_genre_id
third_genre_id
first_artist_name
country_code
registration_code
song_year
language
```

In [ ]:

```
x_tr = tr_data.drop(['target'], axis=1)
y_tr = tr_data['target']

x_val = val_data.drop(['target'], axis=1)
y_val = val_data['target']

x_te = test_all.drop(['id'], axis=1)
ids = test_all['id'].values
```

1. Logistic Regression

- We will use SGDclassifier from sklearn with log loss as it uses SGD instead of GD and converge faster.
- We will use GridSearchCV for hyper-parameter tuning.

Hyper parameter tuning using GridSearchCV

In [ ]:

```
# Hyper parameter tuning using GridearchCV for LR
start = time.time()
parameters = {
                'penalty':['l2', 'l1'],
                'alpha':[10 ** x for x in range(-4, 2)]
            }
clf = SGDClassifier(loss='log', n_jobs=-1, random_state=23, class_weight='balanc
ed' )
model = GridSearchCV(clf, parameters, scoring = 'roc_auc', n_jobs=-1, verbose=2,
cv=3)
model.fit(x_tr, y_tr)

print(model.best_estimator_)
print('train AUC = ',model.score(x_tr, y_tr))
print('val AUC = ',model.score(x_val, y_val))
print('Time taken for hyper parameter tuning is : ', (time.time() -  start))
print('Done!')
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:  8.0min finis
hed

SGDClassifier(alpha=0.1, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_inte
rcept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='log', m
ax_iter=1000,
              n_iter_no_change=5, n_jobs=-1, penalty='l1', power_t=
0.5,
              random_state=23, shuffle=True, tol=0.001, validation_
fraction=0.1,
              verbose=0, warm_start=False)
train AUC =  0.5348631550190844
val AUC =  0.541259700278176
Time taken for hyper parameter tuning is :  489.2709345817566
Done!
```

In [ ]:

```
# train LR with best parameters
lr = SGDClassifier(alpha=0.1, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='log', max_iter=1000,
              n_iter_no_change=5, n_jobs=-1, penalty='l1', power_t=0.5,
              random_state=23, shuffle=True, tol=0.001, validation_fraction=0.1,
              verbose=0, warm_start=False)
lr.fit(x_tr, y_tr)
```

Out[ ]:

```
SGDClassifier(alpha=0.1, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_inte
rcept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='log', m
ax_iter=1000,
              n_iter_no_change=5, n_jobs=-1, penalty='l1', power_t=
0.5,
              random_state=23, shuffle=True, tol=0.001, validation_
fraction=0.1,
              verbose=0, warm_start=False)
```

Feature Importance

In [ ]:

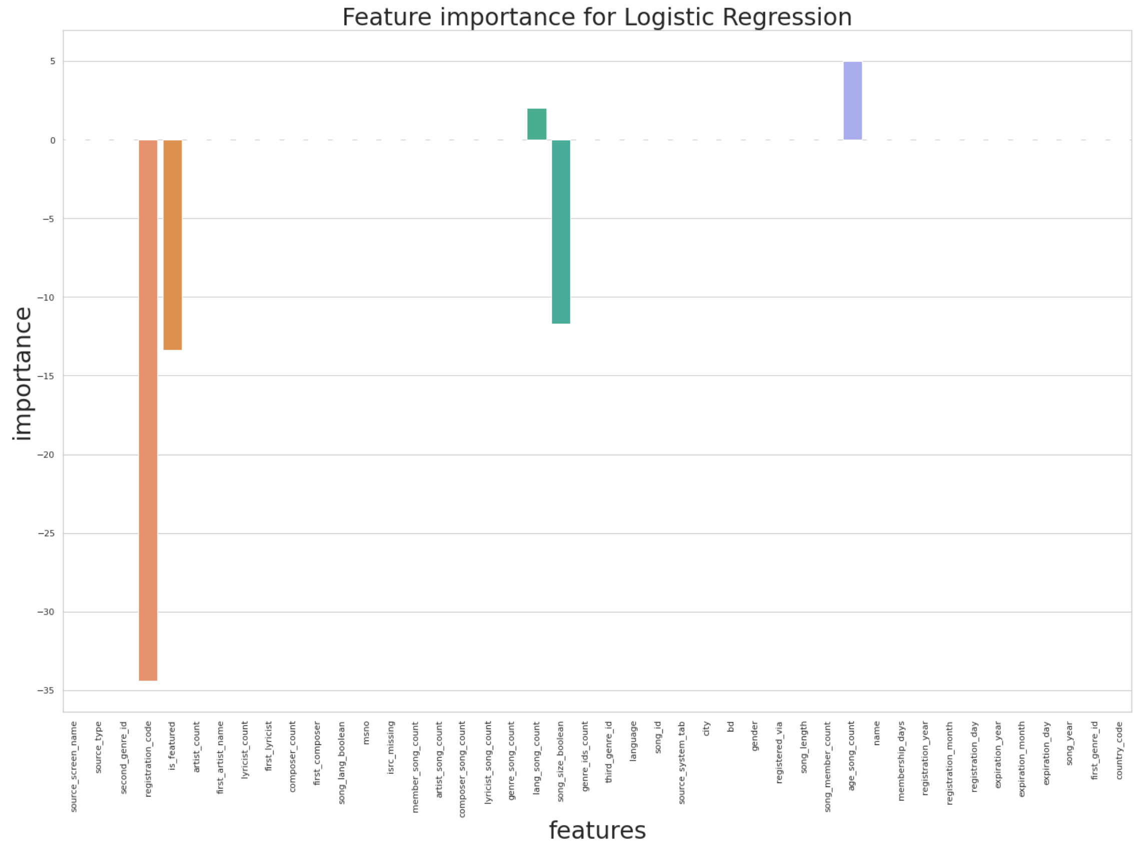
```
# create dataframe for features and it importance
sorted_indices = lr.coef_[0].argsort()
features = x_tr.columns[sorted_indices]
lr_fea_imp = pd.DataFrame({
    'features' : features,
    'importance' : lr.coef_[0]
})
```

In [ ]:

```
# plot feature importance for LR
def plot_feature_importance(data, features, importance, model):
  '''Function to plot feature imporatance'''
  plt.figure(figsize=(20,15))
  sns.set(font_scale=2)
  sns.set(style="whitegrid")
  plt.xlabel('features',fontsize=30)
  plt.ylabel('importance',fontsize=30)
  plt.xticks(rotation='90')
  ax = sns.barplot(x=features, y=importance, data=data)
  plt.title('Feature importance for {model}'.format(model=model),fontsize=30)
  plt.tight_layout()
```

In [ ]:

```
plot_feature_importance(lr_fea_imp, 'features', 'importance', 'Logistic Regressi
on')
```



- We can see from the above plot that, features like registraion_code, msno, song_lang_boolean have negative feature importance.
- Fatures like genre_song_aount and name have higher feature importance.

In [ ]:

```
# Apply model on test data and save predictions
def predict_and_save(clf, x_te, model):
  '''Function to apply model on test data and save the results'''
  file_name = 'submission_' + model +'.csv.gz'
  print('Making predictions')
  p_test = clf.predict(x_te)
  print ('Saving predictions')
  subm = pd.DataFrame()
  subm['id'] = ids
  subm['target'] = p_test
  subm.to_csv(file_name, compression = 'gzip', index=False, float_format = '%.5f
')

  print('Done!')
```

In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(lr, x_te, 'lr')
```

```
Making predictions
Saving predictions
Done!
```

2. SVM

- We will use SGDClassifier from sklearn with 'hinge' loss and use GridSearchCV for hyperparameter tuning.

In [ ]:

```
start = time.time()
parameters = {
                'penalty':['l2', 'l1'],
                'alpha':[10 ** x for x in range(-4, 2)]
            }
clf = SGDClassifier(loss='hinge', n_jobs=-1, random_state=23, class_weight='bala
nced' )
model = GridSearchCV(clf, parameters, scoring = 'roc_auc', n_jobs=-1, verbose=2,
cv=3)
model.fit(x_tr, y_tr)

print(model.best_estimator_)
print('train AUC = ',model.score(x_tr, y_tr))
print('val AUC = ',model.score(x_val, y_val))
print('Time taken for hyper parameter tuning is : ', (time.time() -  start))
print('Done!')
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  36 out of  36 | elapsed:  7.9min finis
hed

SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_inte
rcept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=-1, penalty
='l1',
              power_t=0.5, random_state=23, shuffle=True, tol=0.00
1,
              validation_fraction=0.1, verbose=0, warm_start=False)
train AUC =  0.5290734822292754
val AUC =  0.5253347624820917
Time taken for hyper parameter tuning is :  515.5763082504272
Done!
```

In [ ]:

```
# train model with best parameters
svm = SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=-1, penalty='l1',
              power_t=0.5, random_state=23, shuffle=True, tol=0.001,
              validation_fraction=0.1, verbose=0, warm_start=False)
svm.fit(x_tr, y_tr)
```

Out[ ]:

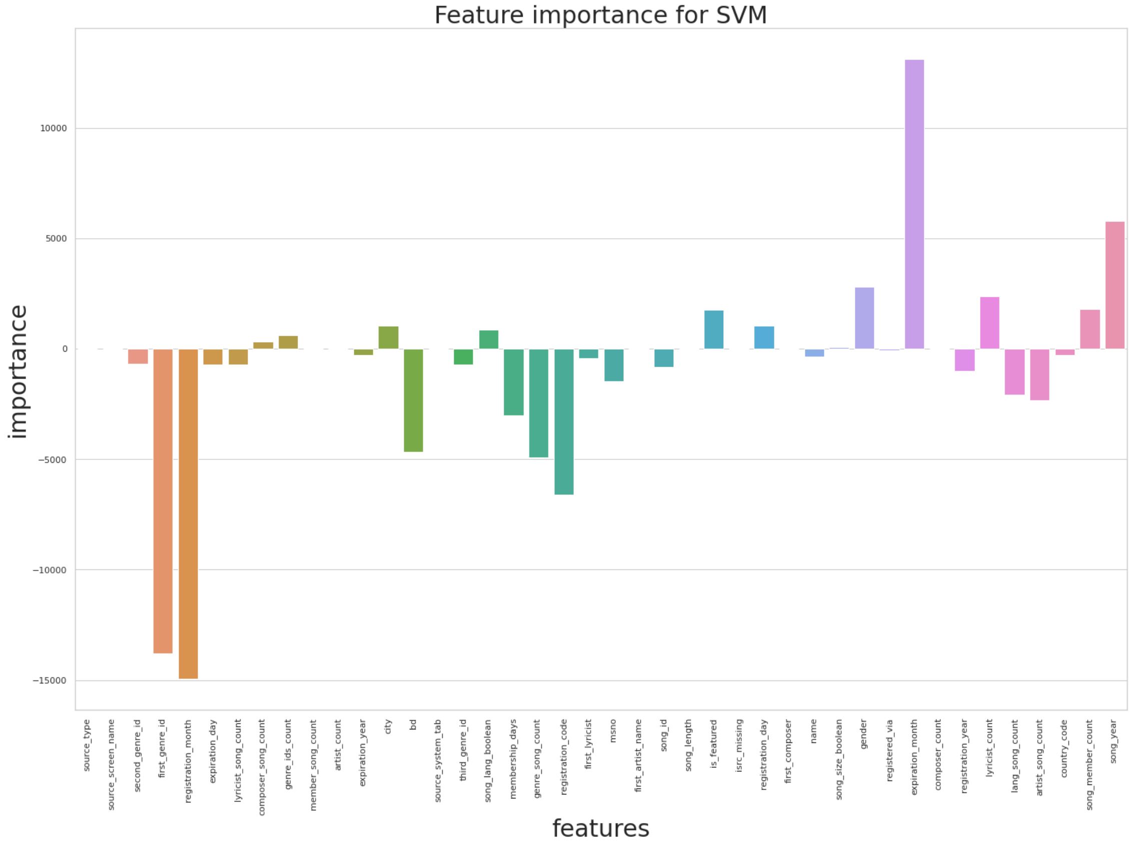
```
SGDClassifier(alpha=0.0001, average=False, class_weight='balanced',
              early_stopping=False, epsilon=0.1, eta0=0.0, fit_inte
rcept=True,
              l1_ratio=0.15, learning_rate='optimal', loss='hinge',
              max_iter=1000, n_iter_no_change=5, n_jobs=-1, penalty
='l1',
              power_t=0.5, random_state=23, shuffle=True, tol=0.00
1,
              validation_fraction=0.1, verbose=0, warm_start=False)
```

In [ ]:

```
# create dataframe for features and it importance
sorted_indices = svm.coef_[0].argsort()
features = x_tr.columns[sorted_indices]
svm_fea_imp = pd.DataFrame({
    'features' : features,
    'importance' : svm.coef_[0]
})
```

In [ ]:

```
# plot feature importance for SVM
plot_feature_importance(svm_fea_imp, 'features', 'importance', 'SVM')
```



- From the above plot we can conclude that, registraino_via has highest feature importance compare to all other features and registraion_month has lower ferature importance.

In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(svm, x_te, 'svm')
```

```
Making predictions
Saving predictions
Done!
```

3. Decision Tree

- We will use DecisionTreeClassifier from sklearn

In [ ]:

```
# Hyper parameter tuning using GridearchCV
start = time.time()
parameters = {
                'max_depth':[3, 5, 8, 10, 15, 50],
                'min_samples_split':[5, 10, 100, 500, 1000],
                'max_leaf_nodes': list(range(2, 100))
            }
clf = DecisionTreeClassifier(random_state=23, class_weight='balanced' )
model = GridSearchCV(clf, parameters, scoring = 'roc_auc', n_jobs=-1, verbose=2,
cv=3)
model.fit(x_tr, y_tr)

print(model.best_estimator_)
print('train AUC = ',model.score(x_tr, y_tr))
print('val AUC = ',model.score(x_val, y_val))
print('Time taken for hyper parameter tuning is : ', (time.time() -  start))
print('Done!')
```

```
Fitting 3 folds for each of 2940 candidates, totalling 8820 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   14.3s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  1.2min
[Parallel(n_jobs=-1)]: Done 361 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done 644 tasks      | elapsed:  5.0min
[Parallel(n_jobs=-1)]: Done 1009 tasks      | elapsed:  7.8min
[Parallel(n_jobs=-1)]: Done 1454 tasks      | elapsed: 11.2min
[Parallel(n_jobs=-1)]: Done 1981 tasks      | elapsed: 16.9min
[Parallel(n_jobs=-1)]: Done 2588 tasks      | elapsed: 24.2min
[Parallel(n_jobs=-1)]: Done 3277 tasks      | elapsed: 32.3min
[Parallel(n_jobs=-1)]: Done 4046 tasks      | elapsed: 44.4min
[Parallel(n_jobs=-1)]: Done 4897 tasks      | elapsed: 56.4min
[Parallel(n_jobs=-1)]: Done 5828 tasks      | elapsed: 72.4min
[Parallel(n_jobs=-1)]: Done 6841 tasks      | elapsed: 86.8min
[Parallel(n_jobs=-1)]: Done 7934 tasks      | elapsed: 103.2min
[Parallel(n_jobs=-1)]: Done 8820 out of 8820 | elapsed: 118.6min fi
nished
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', crit
erion='gini',
                       max_depth=15, max_features=None, max_leaf_no
des=81,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, presort='depre
cated',
                       random_state=23, splitter='best')
train AUC =  0.6699633384389215
val AUC =  0.6298246852683993
Time taken for hyper parameter tuning is :  7116.2408702373505
Done!
```

In [ ]:

```
# train model with best parameters
dt = DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='g
ini',
                       max_depth=15, max_features=None, max_leaf_nodes=81,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=23, splitter='best')
dt.fit(x_tr, y_tr)
```

Out[ ]:

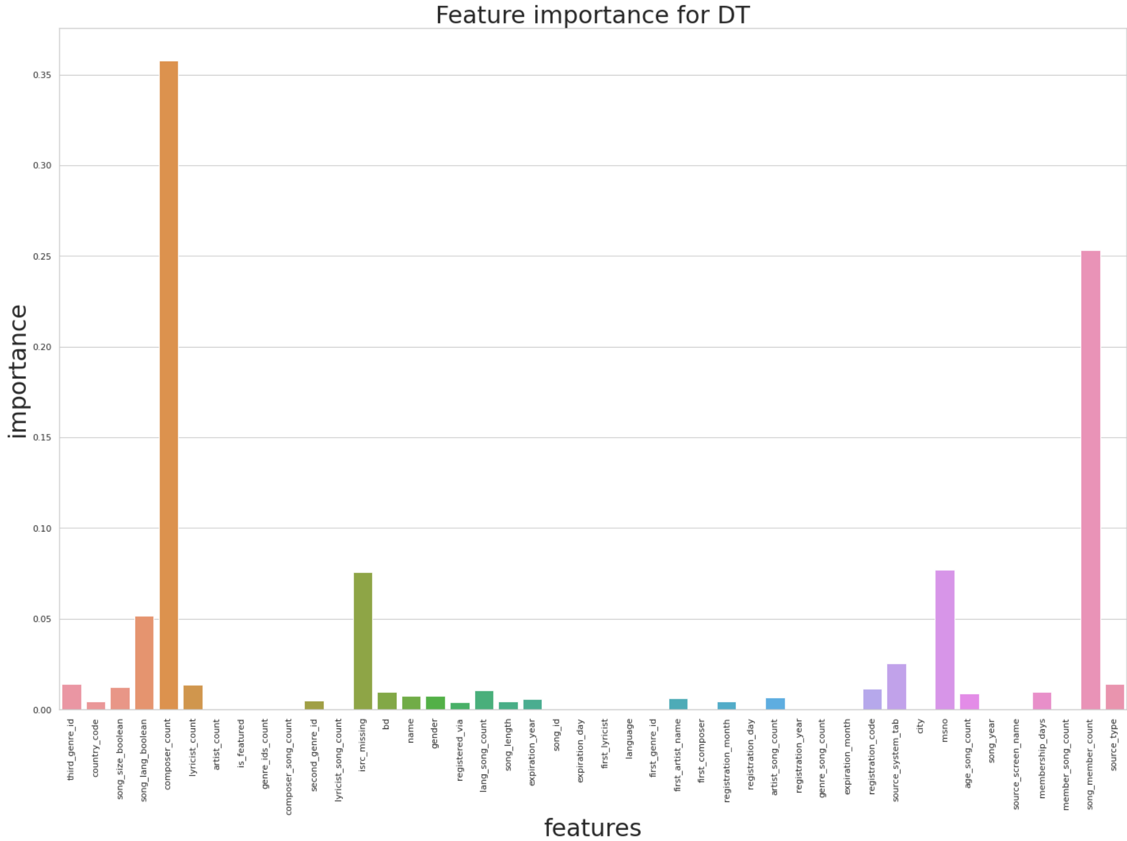
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', crit
erion='gini',
                       max_depth=15, max_features=None, max_leaf_no
des=81,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, presort='depre
cated',
                       random_state=23, splitter='best')
```

In [ ]:

```
# create dataframe for features and it importance
sorted_indices = dt.feature_importances_.argsort()
features = x_tr.columns[sorted_indices]
dt_fea_imp = pd.DataFrame({
    'features' : features,
    'importance' : dt.feature_importances_
})
```

In [ ]:

```
# plot feature importance for DT
plot_feature_importance(dt_fea_imp, 'features', 'importance', 'DT')
```



- From the feature importance plot we can say that, lyrics_count has maximum importance.
- Let's take the features with positive importance like composer_count, song_lang_boolean, isrc_missing, source_system_tab, msno, song_member_count, source_type, third_genre_id.
- Let's try other model using the above mentioned features only.

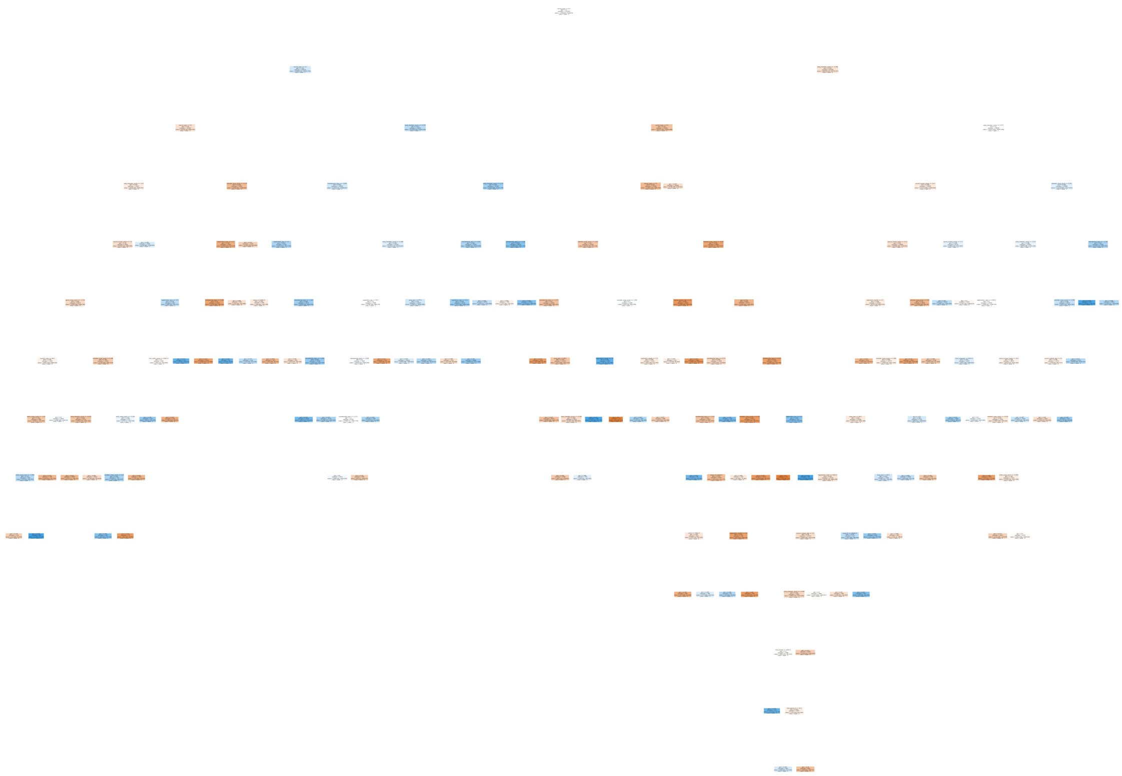
In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(dt, x_te, 'dt')
```

```
Making predictions
Saving predictions
Done!
```

In [ ]:

```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (20,15), dpi=300)
tree.plot_tree(dt,
               feature_names = x_tr.columns,
               class_names=['label - 0', 'label - 1'],
               filled = True);
fig.savefig('tree.png')
```



4. Random Forest

- We will apply random forest on our data sets.
- As hyper parameters tuning will take time for random forest, we will do one by one with list of parameters.

In [ ]:

```
# Hyper parameter tuning using GridearchCV on n_estimators
start = time.time()
parameters = {
                'n_estimators':[10, 50, 100, 200, 300, 500,1000]

            }
clf = RandomForestClassifier(random_state=23, class_weight='balanced', n_jobs=-1
)
model = GridSearchCV(clf, parameters, scoring = 'roc_auc', verbose=2, cv=3)
model.fit(x_tr, y_tr)

print(model.best_estimator_)
print('train AUC = ',model.score(x_tr, y_tr))
print('val AUC = ',model.score(x_val, y_val))
print('Time taken for hyper parameter tuning is : ', (time.time() -  start))
print('Done!')
```

```
Fitting 3 folds for each of 7 candidates, totalling 21 fits
[CV] n_estimators=10
.............................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concur
rent workers.

[CV] .................................. n_estimators=10, total=
4.3s
[CV] n_estimators=10
.............................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    4.3s remain
ing:    0.0s
```

```
[CV] .................................. n_estimators=10, total=
3.3s
[CV] n_estimators=10
.................................................
[CV] .................................. n_estimators=10, total=
3.2s
[CV] n_estimators=50
.................................................
[CV] .................................. n_estimators=50, total=  1
4.6s
[CV] n_estimators=50
.................................................
[CV] .................................. n_estimators=50, total=  1
4.7s
[CV] n_estimators=50
.................................................
[CV] .................................. n_estimators=50, total=  1
4.8s
[CV] n_estimators=100
................................................
[CV] ................................. n_estimators=100, total=  2
9.0s
[CV] n_estimators=100
................................................
[CV] ................................. n_estimators=100, total=  2
9.2s
[CV] n_estimators=100
................................................
[CV] ................................. n_estimators=100, total=  2
9.2s
[CV] n_estimators=200
................................................
[CV] ................................. n_estimators=200, total= 1.1
min
[CV] n_estimators=200
................................................
[CV] ................................. n_estimators=200, total=  5
8.2s
[CV] n_estimators=200
................................................
[CV] ................................. n_estimators=200, total=  5
7.9s
[CV] n_estimators=300
................................................
[CV] ................................. n_estimators=300, total= 1.4
min
[CV] n_estimators=300
................................................
[CV] ................................. n_estimators=300, total= 1.4
min
[CV] n_estimators=300
................................................
[CV] ................................. n_estimators=300, total= 1.4
min
[CV] n_estimators=500
................................................
[CV] ................................. n_estimators=500, total= 2.4
min
[CV] n_estimators=500
................................................
[CV] ................................. n_estimators=500, total= 2.4
```

```
min
[CV] n_estimators=500 
.............................................
[CV] ............................... n_estimators=500, total= 2.4
min
[CV] n_estimators=1000 
............................................
[CV] .............................. n_estimators=1000, total= 4.8
min
[CV] n_estimators=1000 
............................................
[CV] .............................. n_estimators=1000, total= 4.9
min
[CV] n_estimators=1000 
............................................
[CV] .............................. n_estimators=1000, total= 4.9
min

[Parallel(n_jobs=1)]: Done  21 out of  21 | elapsed: 31.6min finish
ed

RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight
='balanced',
                       criterion='gini', max_depth=None, max_featur
es='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1
000,
                       n_jobs=-1, oob_score=False, random_state=23,
verbose=0,
                       warm_start=False)
train AUC =  1.0
val AUC =  0.6584052518351906
Time taken for hyper parameter tuning is :  2375.6078023910522
Done!
```

- We can see here our train auc is 1.0 and val auc is 0.65 which means our model is overfitted.

In [ ]:

```
# Hyper parameter tuning using GridearchCV on max_depth
start = time.time()
parameters = {
                'max_depth': [4, 8, 10, 15, 25]

            }
clf = RandomForestClassifier(n_estimators=1000,random_state=23, class_weight='ba
lanced', n_jobs=-1)
model = GridSearchCV(clf, parameters, scoring = 'roc_auc', verbose=2, cv=3)
model.fit(x_tr, y_tr)

print(model.best_estimator_)
print('train AUC = ',model.score(x_tr, y_tr))
print('val AUC = ',model.score(x_val, y_val))
print('Time taken for hyper parameter tuning is : ', (time.time() -  start))
print('Done!')
```

```
Fitting 3 folds for each of 5 candidates, totalling 15 fits
[CV] max_depth=4
.....................................................

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concur
rent workers.

[CV] ...................................... max_depth=4, total= 1.2
min
[CV] max_depth=4
.....................................................

[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  1.2min remain
ing:    0.0s
```

```
[CV] ...................................... max_depth=4, total= 1.1
min
[CV] max_depth=4
.....................................................
[CV] ...................................... max_depth=4, total= 1.1
min
[CV] max_depth=8
.....................................................
[CV] ...................................... max_depth=8, total= 2.0
min
[CV] max_depth=8
.....................................................
[CV] ...................................... max_depth=8, total= 2.0
min
[CV] max_depth=8
.....................................................
[CV] ...................................... max_depth=8, total= 2.0
min
[CV] max_depth=10
....................................................
[CV] ..................................... max_depth=10, total= 2.4
min
[CV] max_depth=10
....................................................
[CV] ..................................... max_depth=10, total= 2.5
min
[CV] max_depth=10
....................................................
[CV] ..................................... max_depth=10, total= 2.4
min
[CV] max_depth=15
....................................................
[CV] ..................................... max_depth=15, total= 3.4
min
[CV] max_depth=15
....................................................
[CV] ..................................... max_depth=15, total= 3.5
min
[CV] max_depth=15
....................................................
[CV] ..................................... max_depth=15, total= 3.5
min
[CV] max_depth=25
....................................................
[CV] ..................................... max_depth=25, total= 4.7
min
[CV] max_depth=25
....................................................
[CV] ..................................... max_depth=25, total= 4.7
min
[CV] max_depth=25
....................................................
[CV] ..................................... max_depth=25, total= 4.7
min

[Parallel(n_jobs=1)]: Done  15 out of  15 | elapsed: 41.1min finish
ed
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight
='balanced',
                       criterion='gini', max_depth=25, max_features
='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1
000,
                       n_jobs=-1, oob_score=False, random_state=23,
verbose=0,
                       warm_start=False)
train AUC =  0.9999947771685423
val AUC =  0.6576674894915711
Time taken for hyper parameter tuning is :  2936.1337826251984
Done!
```

- Here difference between train and val auc is very huge which tells us about overfitting.

In [ ]:

```
# Hyper parameter tuning using GridearchCV on min_samples_split
start = time.time()
parameters = {
                'min_samples_split':[3, 5, 10],

            }
clf = RandomForestClassifier(n_estimators=1000, max_depth=25, random_state=23, c
lass_weight='balanced', n_jobs=-1)
model = GridSearchCV(clf, parameters, scoring = 'roc_auc', verbose=2, cv=3)
model.fit(x_tr, y_tr)

print(model.best_estimator_)
print('train AUC = ',model.score(x_tr, y_tr))
print('val AUC = ',model.score(x_val, y_val))
print('Time taken for hyper parameter tuning is : ', (time.time() -  start))
print('Done!')
```

```
Fitting 3 folds for each of 3 candidates, totalling 9 fits
[CV] min_samples_split=3
.............................................
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concur
rent workers.
[CV] .............................. min_samples_split=3, total= 4.7
min
[CV] min_samples_split=3
.............................................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:  4.7min remain
ing:    0.0s
[CV] .............................. min_samples_split=3, total= 4.6
min
[CV] min_samples_split=3
.............................................
[CV] .............................. min_samples_split=3, total= 4.7
min
[CV] min_samples_split=5
.............................................
[CV] .............................. min_samples_split=5, total= 4.6
min
[CV] min_samples_split=5
.............................................
[CV] .............................. min_samples_split=5, total= 4.7
min
[CV] min_samples_split=5
.............................................
[CV] .............................. min_samples_split=5, total= 4.7
min
[CV] min_samples_split=10
............................................
[CV] ............................. min_samples_split=10, total= 4.5
min
[CV] min_samples_split=10
............................................
[CV] ............................. min_samples_split=10, total= 4.5
min
[CV] min_samples_split=10
............................................
[CV] ............................. min_samples_split=10, total= 4.5
min
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed: 41.4min finish
ed
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight
='balanced',
                       criterion='gini', max_depth=25, max_features
='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=10,
                       min_weight_fraction_leaf=0.0, n_estimators=1
000,
                       n_jobs=-1, oob_score=False, random_state=23,
verbose=0,
                       warm_start=False)
train AUC =  0.9956691642313812
val AUC =  0.6576797612990944
Time taken for hyper parameter tuning is :  2919.7276499271393
Done!
```

- Again we can see overfitting of our model.
- Lets try first above parametrs and check the performance.

In [ ]:

```
# train model with best parameters
rf = RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight='balance
d',
                       criterion='gini', max_depth=25, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=10,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       n_jobs=-1, oob_score=False, random_state=23, verbose=0,
                       warm_start=False)
rf.fit(x_tr, y_tr)
```

Out[ ]:

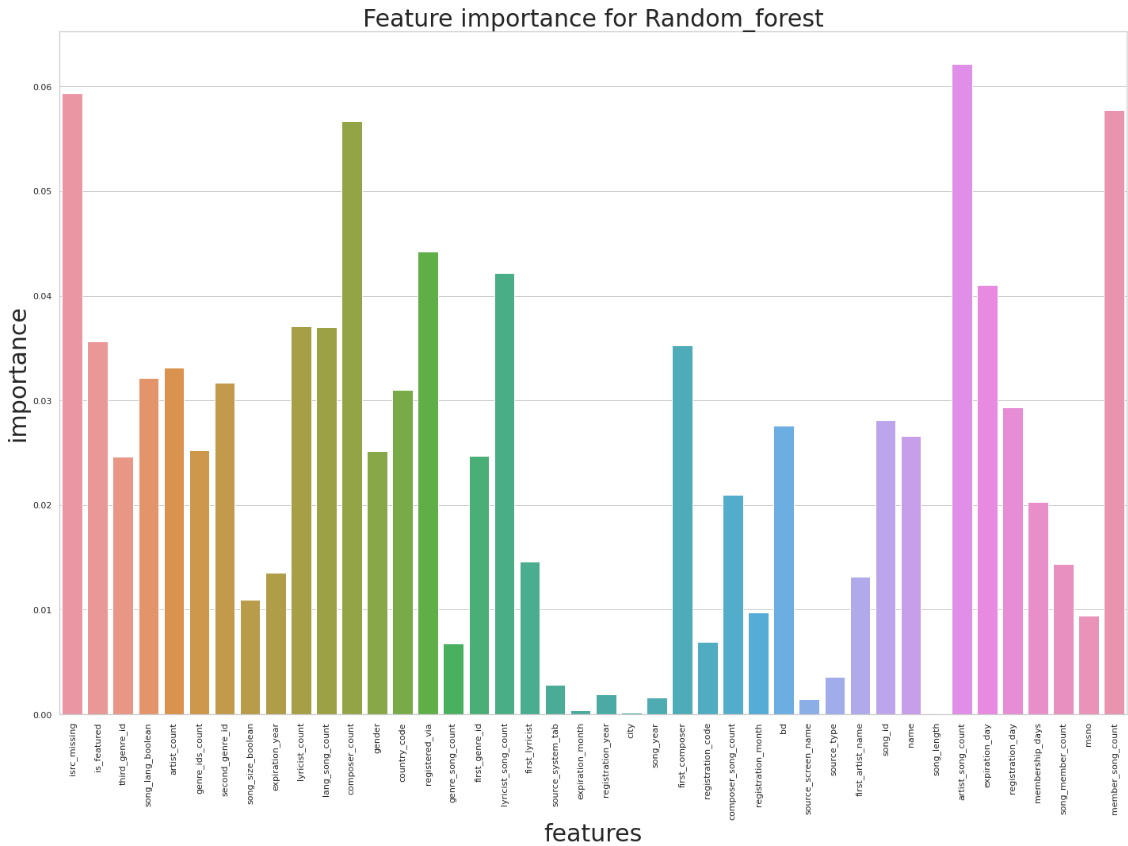
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight
='balanced',
                       criterion='gini', max_depth=25, max_features
='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=10,
                       min_weight_fraction_leaf=0.0, n_estimators=1
000,
                       n_jobs=-1, oob_score=False, random_state=23,
verbose=0,
                       warm_start=False)
```

In [ ]:

```
# create dataframe for features and it importance
sorted_indices = rf.feature_importances_.argsort()
features = x_tr.columns[sorted_indices]
rf_fea_imp = pd.DataFrame({
    'features' : features,
    'importance' : rf.feature_importances_
})
```

In [ ]:

```
# plot feature importance for DT
plot_feature_importance(rf_fea_imp, 'features', 'importance', 'Random_forest')
```



- We can see from the above plot that all features have positive importance.

In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(rf, x_te, 'rf')
```

```
Making predictions
Saving predictions
Done!
```

5. XgBoost

- We will apply XgBoost for GBDT.

In [ ]:

```
start = time.time()
parameters={
              'learning_rate' : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30],
              'max_depth': [3, 4, 5, 6, 8, 10, 12, 15],
              'min_child_weight' : [1, 3, 5, 7],
              'gamma' : [0.0, 0.1, 0.2 , 0.3, 0.4],
              'colsample_bytree': [0.3, 0.4, 0.5 , 0.7]
          }

clf = xgboost.XGBClassifier(objective='binary:logistic', random_state=2)
model = RandomizedSearchCV(clf, parameters, scoring = 'roc_auc', n_jobs=-1, verb
ose=2, cv=3)
model.fit(x_tr, y_tr)

print(model.best_estimator_)
print('train AUC = ',model.score(x_tr, y_tr))
print('val AUC = ',model.score(x_val, y_val))
print('Time taken for hyper parameter tuning is : ', (time.time() -  start))
print('Done!')
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent
workers.
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed:  9.3min finis
hed

XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=
1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0.3,
              learning_rate=0.05, max_delta_step=0, max_depth=15,
              min_child_weight=7, missing=None, n_estimators=100, n
_jobs=1,
              nthread=None, objective='binary:logistic', random_sta
te=2,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=N
one,
              silent=None, subsample=1, verbosity=1)
train AUC =  0.9600632533373371
val AUC =  0.662671619401928
Time taken for hyper parameter tuning is :  654.7759299278259
Done!
```

- As our model is overfitting so we have to be more careful during parameter tuning.

In [ ]:

```
xgb = xgboost.XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=
1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0.3,
              learning_rate=0.05, max_delta_step=0, max_depth=15,
              min_child_weight=7, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=2,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
xgb.fit(x_tr, y_tr)
```

Out[ ]:

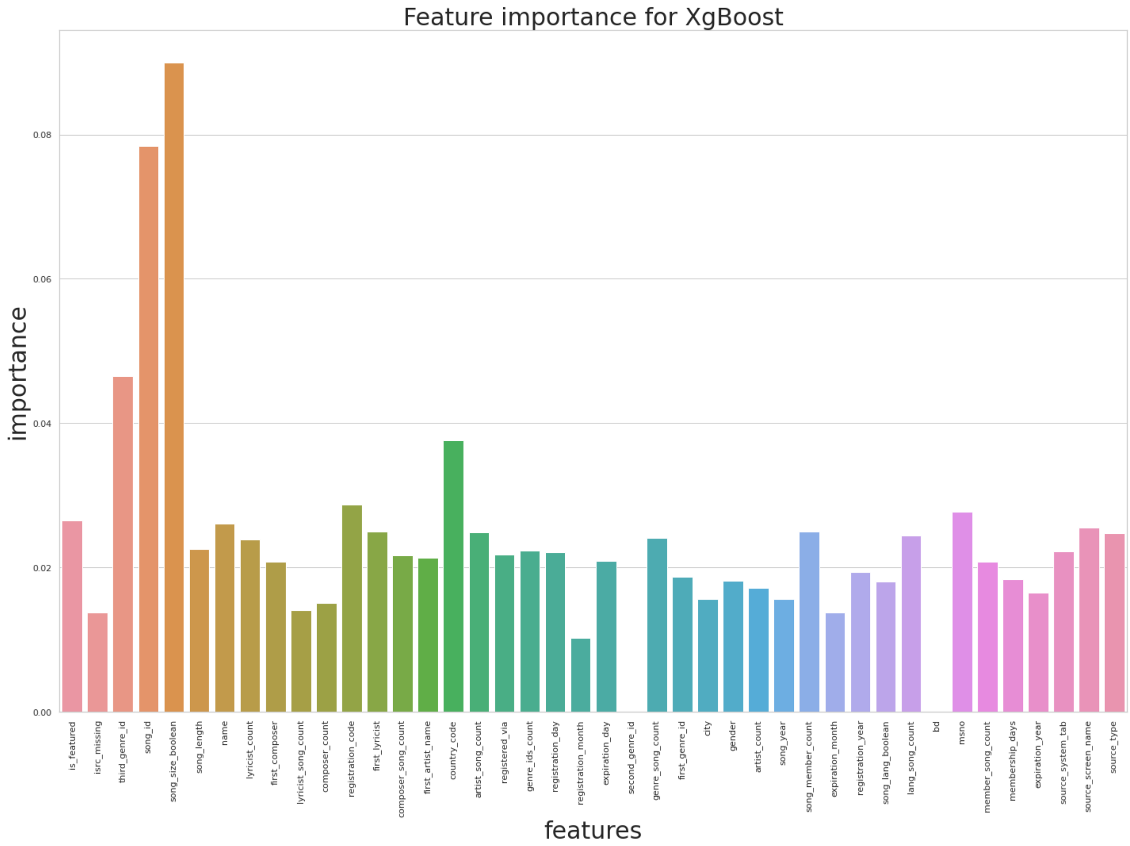
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=
1,
              colsample_bynode=1, colsample_bytree=0.5, gamma=0.3,
              learning_rate=0.05, max_delta_step=0, max_depth=15,
              min_child_weight=7, missing=None, n_estimators=100, n
_jobs=1,
              nthread=None, objective='binary:logistic', random_sta
te=2,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=N
one,
              silent=None, subsample=1, verbosity=1)
```

In [ ]:

```
# create dataframe for features and it importance
sorted_indices = xgb.feature_importances_.argsort()
features = x_tr.columns[sorted_indices]
xgb_fea_imp = pd.DataFrame({
    'features' : features,
    'importance' : xgb.feature_importances_
})
```

In [ ]:

```
# plot feature importance for DT
plot_feature_importance(xgb_fea_imp, 'features', 'importance', 'XgBoost')
```



- We can see that all features have positive importance.

In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(xgb, x_te, 'xgb')
```

```
Making predictions
Saving predictions
Done!
```

6. LightGBM

- Lets try lightGBM on our data points.

In [ ]:

```
params = {
        'objective': 'binary',
        'metric': 'binary_logloss',
        'boosting': 'gbdt',
        'learning_rate': 0.3 ,
        'verbose': 0,
        'num_leaves': 108,
        'bagging_fraction': 0.95,
        'bagging_freq': 1,
        'bagging_seed': 1,
        'feature_fraction': 0.9,
        'feature_fraction_seed': 1,
        'max_bin': 256,
        'max_depth': 10,
        'num_rounds': 400,
        'metric' : 'auc'
    }
```

In [ ]:

```
tr_final = lgb.Dataset(x_tr, y_tr)
val_final = lgb.Dataset(x_val, y_val)
```

In [ ]:

```
%time model_f1 = lgb.train(params, train_set=tr_final,  valid_sets=val_final, ve
rbose_eval=5)
```

```
/usr/local/lib/python3.6/dist-packages/lightgbm/engine.py:118: User
Warning: Found `num_rounds` in params. Will use it instead of argum
ent
  warnings.warn("Found `{}` in params. Will use it instead of argum
ent".format(alias))
```

```
[5]	valid_0's auc: 0.642236
[10]	valid_0's auc: 0.641663
[15]	valid_0's auc: 0.644865
[20]	valid_0's auc: 0.644785
[25]	valid_0's auc: 0.645383
[30]	valid_0's auc: 0.64256
[35]	valid_0's auc: 0.64102
[40]	valid_0's auc: 0.640037
[45]	valid_0's auc: 0.638787
[50]	valid_0's auc: 0.641199
[55]	valid_0's auc: 0.641181
[60]	valid_0's auc: 0.64099
[65]	valid_0's auc: 0.641217
[70]	valid_0's auc: 0.640791
[75]	valid_0's auc: 0.640626
[80]	valid_0's auc: 0.641487
[85]	valid_0's auc: 0.640911
[90]	valid_0's auc: 0.640651
[95]	valid_0's auc: 0.639755
[100]	valid_0's auc: 0.638531
[105]	valid_0's auc: 0.63871
[110]	valid_0's auc: 0.637891
[115]	valid_0's auc: 0.63814
[120]	valid_0's auc: 0.638356
[125]	valid_0's auc: 0.637261
[130]	valid_0's auc: 0.637838
[135]	valid_0's auc: 0.636793
[140]	valid_0's auc: 0.636234
[145]	valid_0's auc: 0.63532
[150]	valid_0's auc: 0.635333
[155]	valid_0's auc: 0.63501
[160]	valid_0's auc: 0.635277
[165]	valid_0's auc: 0.63524
[170]	valid_0's auc: 0.635297
[175]	valid_0's auc: 0.63461
[180]	valid_0's auc: 0.634174
[185]	valid_0's auc: 0.633746
[190]	valid_0's auc: 0.634136
[195]	valid_0's auc: 0.634111
[200]	valid_0's auc: 0.634393
[205]	valid_0's auc: 0.633509
[210]	valid_0's auc: 0.633133
[215]	valid_0's auc: 0.633004
[220]	valid_0's auc: 0.633179
[225]	valid_0's auc: 0.633364
[230]	valid_0's auc: 0.633661
[235]	valid_0's auc: 0.633889
[240]	valid_0's auc: 0.634692
[245]	valid_0's auc: 0.634294
[250]	valid_0's auc: 0.634093
[255]	valid_0's auc: 0.633851
[260]	valid_0's auc: 0.633366
[265]	valid_0's auc: 0.63315
[270]	valid_0's auc: 0.63293
[275]	valid_0's auc: 0.632124
[280]	valid_0's auc: 0.632388
[285]	valid_0's auc: 0.632391
[290]	valid_0's auc: 0.632617
[295]	valid_0's auc: 0.632686
[300]	valid_0's auc: 0.632706
[305]	valid_0's auc: 0.632559
```

```
[310]	valid_0's auc: 0.632591
[315]	valid_0's auc: 0.632686
[320]	valid_0's auc: 0.632865
[325]	valid_0's auc: 0.632925
[330]	valid_0's auc: 0.633159
[335]	valid_0's auc: 0.632684
[340]	valid_0's auc: 0.632691
[345]	valid_0's auc: 0.633159
[350]	valid_0's auc: 0.632803
[355]	valid_0's auc: 0.632521
[360]	valid_0's auc: 0.632576
[365]	valid_0's auc: 0.63243
[370]	valid_0's auc: 0.632812
[375]	valid_0's auc: 0.632423
[380]	valid_0's auc: 0.632974
[385]	valid_0's auc: 0.633319
[390]	valid_0's auc: 0.633255
[395]	valid_0's auc: 0.633354
[400]	valid_0's auc: 0.633277
CPU times: user 37.1 s, sys: 483 ms, total: 37.6 s
Wall time: 19.8 s
```

In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(model_f1, x_te, 'lgb')
```

```
Making predictions
Saving predictions
Done!
```

In [ ]:

```
import joblib
# save model
joblib.dump(model_f1, '/content/drive/My Drive/CS-1/Data/lgb.pkl')
```

Out[ ]:

```
['/content/drive/My Drive/CS-1/Data/lgb.pkl']
```

8. Voting Classifier

- We will use voting classifier from sklearn with DT, RF and XgBoost.

In [ ]:

```
# Lets use voting classifier
voting_hard = VotingClassifier(estimators=[('dt', dt), ('xgb', xgb), ('rf', rf
)], voting='hard', n_jobs=-1)
voting_hard.fit(x_tr, y_tr)
```

Out[ ]:

```
VotingClassifier(estimators=[('dt',
                              DecisionTreeClassifier(ccp_alpha=0.0,
                                                     class_weight
='balanced',
                                                     criterion='gin
i',
                                                     max_depth=15,
                                                     max_features=N
one,
                                                     max_leaf_nodes
=81,
                                                     min_impurity_d
ecrease=0.0,
                                                     min_impurity_s
plit=None,
                                                     min_samples_le
af=1,
                                                     min_samples_sp
lit=5,
                                                     min_weight_fra
ction_leaf=0.0,
                                                     presort='depre
cated',
                                                     random_state=2
3,
                                                     splitter='bes
t')),
                             ('xgb',
                              XGBClass...
                                                     criterion='gin
i',
                                                     max_depth=25,
                                                     max_features
='auto',
                                                     max_leaf_nodes
=None,
                                                     max_samples=No
ne,
                                                     min_impurity_d
ecrease=0.0,
                                                     min_impurity_s
plit=None,
                                                     min_samples_le
af=1,
                                                     min_samples_sp
lit=10,
                                                     min_weight_fra
ction_leaf=0.0,
                                                     n_estimators=1
000,
                                                     n_jobs=-1, oob
_score=False,
                                                     random_state=2
3, verbose=0,
                                                     warm_start=Fal
se))],
                 flatten_transform=True, n_jobs=-1, voting='hard',
                 weights=None)
```

In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(voting_hard, x_te, 'Voting_hard_Classifier')
```

```
Making predictions
Saving predictions
Done!
```

In [ ]:

```
estimators = [
              ('dt', DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balance
d', criterion='gini',
                       max_depth=15, max_features=None, max_leaf_nodes=81,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=23, splitter='best')),

              ('rf', RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class
_weight='balanced',
                       criterion='gini', max_depth=25, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=10,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       oob_score=False, random_state=23, verbose=0,
                       warm_start=False)),

              ('xgb', xgboost.XGBClassifier(base_score=0.5, booster='gbtree', co
lsample_bylevel=1,
                      colsample_bynode=1, colsample_bytree=0.5, gamma=0.3,
                      learning_rate=0.05, max_delta_step=0, max_depth=15,
                      min_child_weight=7, missing=None, n_estimators=100,
                      nthread=None, objective='binary:logistic', random_state=2,
                      reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                      silent=None, subsample=1, verbosity=1))
]

voting_soft = VotingClassifier(estimators=estimators, voting='soft')
voting_soft.fit(x_tr, y_tr)
```

In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(voting_soft, x_te, 'Voting_Soft_Classifier')
```

9. Stacking Classifier

In [ ]:

```
estimators = [
              ('dt', DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balance
d', criterion='gini',
                       max_depth=15, max_features=None, max_leaf_nodes=81,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=23, splitter='best')),

              ('rf', RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class
_weight='balanced',
                       criterion='gini', max_depth=25, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=10,
                       min_weight_fraction_leaf=0.0, n_estimators=1000,
                       oob_score=False, random_state=23, verbose=0,
                       warm_start=False)),

              ('xgb', xgboost.XGBClassifier(base_score=0.5, booster='gbtree', co
lsample_bylevel=1,
                      colsample_bynode=1, colsample_bytree=0.5, gamma=0.3,
                      learning_rate=0.05, max_delta_step=0, max_depth=15,
                      min_child_weight=7, missing=None, n_estimators=100,
                      nthread=None, objective='binary:logistic', random_state=2,
                      reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
                      silent=None, subsample=1, verbosity=1))
]
```

In [ ]:

```
stacking = StackingClassifier(estimators=estimators, verbose=1)
stacking.fit(x_tr, y_tr)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concur
rent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.5s finish
ed
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concur
rent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed: 35.1min finish
ed
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concur
rent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:  5.8min finish
ed
```

Out[ ]:

```
StackingClassifier(cv=None,
                   estimators=[('dt',
                                DecisionTreeClassifier(ccp_alpha=0.
0,
                                                       class_weight
='balanced',
                                                       criterion='g
ini',
                                                       max_depth=1
5,
                                                       max_features
=None,
                                                       max_leaf_nod
es=81,
                                                       min_impurity
_decrease=0.0,
                                                       min_impurity
_split=None,
                                                       min_samples_
leaf=1,
                                                       min_samples_
split=5,
                                                       min_weight_f
raction_leaf=0.0,
                                                       presort='dep
recated',
                                                       random_state
=23,
                                                       splitter='be
st')),
                               ('rf'...
                                             colsample_bytree=0.5,
gamma=0.3,
                                             learning_rate=0.05,
                                             max_delta_step=0, max
_depth=15,
                                             min_child_weight=7, m
issing=None,
                                             n_estimators=100, n_j
obs=1,
                                             nthread=None,
                                             objective='binary:log
istic',
                                             random_state=2, reg_a
lpha=0,
                                             reg_lambda=1, scale_p
os_weight=1,
                                             seed=None, silent=Non
e,
                                             subsample=1, verbosit
y=1))],
                   final_estimator=None, n_jobs=None, passthrough=F
alse,
                   stack_method='auto', verbose=1)
```

In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(stacking, x_te, 'stacking')
```

```
Making predictions
Saving predictions
Done!
```

2. Feature Extraction and Selection

1. Feature Selection using DT

- From feature importance (using our decision tree model) we have selected features which are having higher importance.

In [ ]:

```
x_tr_new = x_tr[['composer_count', 'song_lang_boolean', 'isrc_missing', 'source_
system_tab', 'msno', 'song_member_count', 'source_type', 'third_genre_id']]
x_val_new = x_val[['composer_count', 'song_lang_boolean', 'isrc_missing', 'sourc
e_system_tab', 'msno', 'song_member_count', 'source_type', 'third_genre_id']]
x_te_new = x_te[['composer_count', 'song_lang_boolean', 'isrc_missing', 'source_
system_tab', 'msno', 'song_member_count', 'source_type', 'third_genre_id']]
```

1. DT

In [ ]:

```
dt = DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='g
ini',
                       max_depth=15, max_features=None, max_leaf_nodes=81,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=23, splitter='best')
dt.fit(x_tr_new, y_tr)
```

Out[ ]:

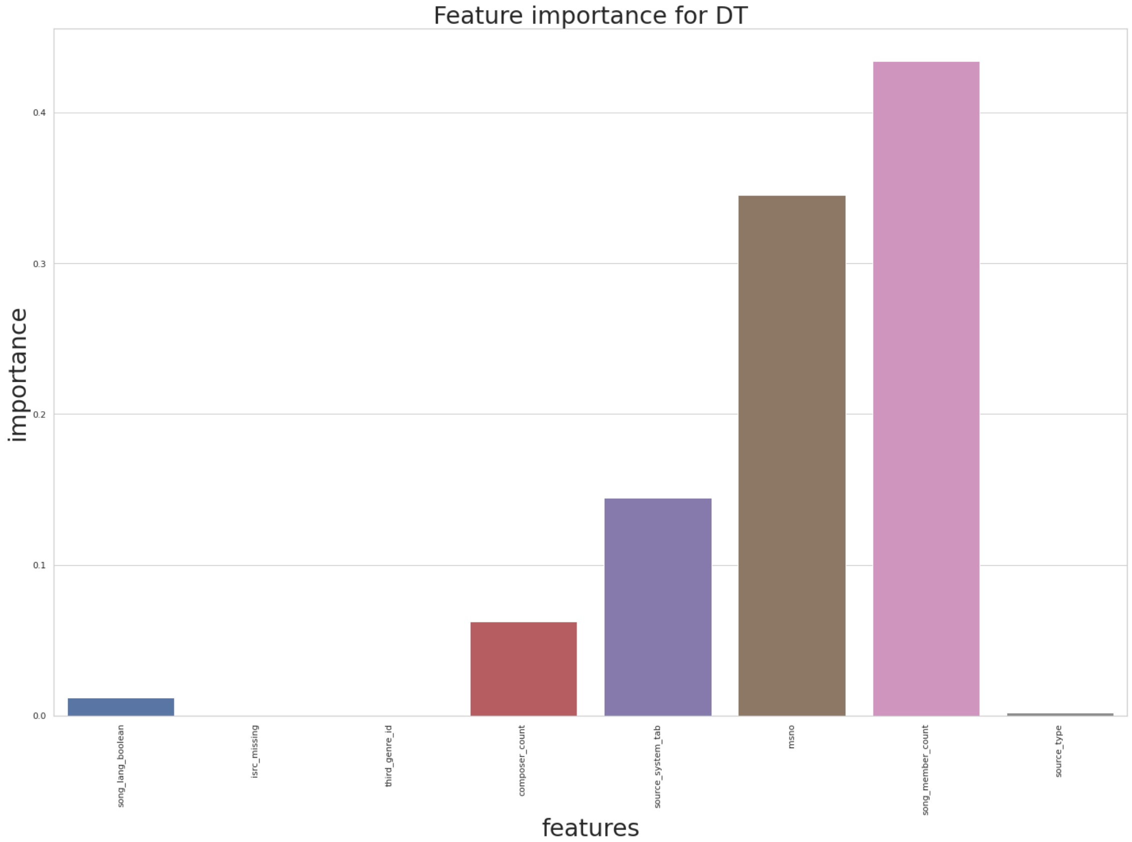
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', crit
erion='gini',
                       max_depth=15, max_features=None, max_leaf_no
des=81,
                       min_impurity_decrease=0.0, min_impurity_spli
t=None,
                       min_samples_leaf=1, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, presort='depre
cated',
                       random_state=23, splitter='best')
```

In [ ]:

```
# create dataframe for features and it importance
sorted_indices = dt.feature_importances_.argsort()
features = x_tr_new.columns[sorted_indices]
dt_fea_imp = pd.DataFrame({
    'features' : features,
    'importance' : dt.feature_importances_
})
```

In [ ]:

```
# plot feature importance for DT
plot_feature_importance(dt_fea_imp, 'features', 'importance', 'DT')
```



In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(dt, x_te_new, 'dt_new')
```

```
Making predictions
Saving predictions
Done!
```

2. LighGBM

In [ ]:

```
tr_final = lgb.Dataset(x_tr_new, y_tr)
val_final = lgb.Dataset(x_val_new, y_val)
```

In [ ]:

```
%time model_f1 = lgb.train(params, train_set=tr_final,  valid_sets=val_final, ve
rbose_eval=5)
```

```
/usr/local/lib/python3.6/dist-packages/lightgbm/engine.py:118: User
Warning: Found `num_rounds` in params. Will use it instead of argum
ent
  warnings.warn("Found `{}` in params. Will use it instead of argum
ent".format(alias))
```

```
[5]	valid_0's auc: 0.630934
[10]	valid_0's auc: 0.630602
[15]	valid_0's auc: 0.629671
[20]	valid_0's auc: 0.628992
[25]	valid_0's auc: 0.62608
[30]	valid_0's auc: 0.625515
[35]	valid_0's auc: 0.625409
[40]	valid_0's auc: 0.62315
[45]	valid_0's auc: 0.622579
[50]	valid_0's auc: 0.622723
[55]	valid_0's auc: 0.621457
[60]	valid_0's auc: 0.620372
[65]	valid_0's auc: 0.618894
[70]	valid_0's auc: 0.61836
[75]	valid_0's auc: 0.61766
[80]	valid_0's auc: 0.617532
[85]	valid_0's auc: 0.617312
[90]	valid_0's auc: 0.616912
[95]	valid_0's auc: 0.616814
[100]	valid_0's auc: 0.616185
[105]	valid_0's auc: 0.615344
[110]	valid_0's auc: 0.615109
[115]	valid_0's auc: 0.61504
[120]	valid_0's auc: 0.61463
[125]	valid_0's auc: 0.613921
[130]	valid_0's auc: 0.61368
[135]	valid_0's auc: 0.613363
[140]	valid_0's auc: 0.612678
[145]	valid_0's auc: 0.612669
[150]	valid_0's auc: 0.612837
[155]	valid_0's auc: 0.612337
[160]	valid_0's auc: 0.612408
[165]	valid_0's auc: 0.612211
[170]	valid_0's auc: 0.611906
[175]	valid_0's auc: 0.611344
[180]	valid_0's auc: 0.610967
[185]	valid_0's auc: 0.611058
[190]	valid_0's auc: 0.610988
[195]	valid_0's auc: 0.611042
[200]	valid_0's auc: 0.610834
CPU times: user 10.5 s, sys: 156 ms, total: 10.7 s
Wall time: 5.64 s
```

In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(model_f1, x_te_2, 'lgb_dt')
```

```
Making predictions
Saving predictions
Done!
```

- When we use the above extracted features in DT, it gives better results comapre to previous features.
- But when we apply the above features in LightGBM, it decreases the score little bit.

2. Feature Selection using SelectKBest

- We will experiment with selectKbest method which selects best k features score and use it.

In [ ]:

```
selection = SelectKBest(f_classif, k=20).fit(x_tr, y_tr)
x_tr_2 = selection.transform(x_tr)
x_val_2 = selection.transform(x_val)
x_te_2 = selection.transform(x_te)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/feature_selection/_u
nivariate_selection.py:114: UserWarning: Features [28 33] are const
ant.
  UserWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/feature_selection/_u
nivariate_selection.py:115: RuntimeWarning: invalid value encounter
ed in true_divide
  f = msb / msw
```

Light GBM

In [ ]:

```
params = {
        'objective': 'binary',
        'metric': 'binary_logloss',
        'boosting': 'gbdt',
        'learning_rate': 0.3 ,
        'verbose': 0,
        'num_leaves': 108,
        'bagging_fraction': 0.95,
        'bagging_freq': 1,
        'bagging_seed': 1,
        'feature_fraction': 0.9,
        'feature_fraction_seed': 1,
        'max_bin': 256,
        'max_depth': 10,
        'num_rounds': 200,
        'metric' : 'auc'
    }
```

In [ ]:

```
tr_final = lgb.Dataset(x_tr_2, y_tr)
val_final = lgb.Dataset(x_val_2, y_val)
```

In [ ]:

```
%time model_f1 = lgb.train(params, train_set=tr_final,  valid_sets=val_final, ve
rbose_eval=5)
```

```
/usr/local/lib/python3.6/dist-packages/lightgbm/engine.py:118: User
Warning: Found `num_rounds` in params. Will use it instead of argum
ent
  warnings.warn("Found `{}` in params. Will use it instead of argum
ent".format(alias))
```

```
[5]     valid_0's auc: 0.640429
[10]    valid_0's auc: 0.642107
[15]    valid_0's auc: 0.642551
[20]    valid_0's auc: 0.641285
[25]    valid_0's auc: 0.641125
[30]    valid_0's auc: 0.640026
[35]    valid_0's auc: 0.639563
[40]    valid_0's auc: 0.638113
[45]    valid_0's auc: 0.636137
[50]    valid_0's auc: 0.63544
[55]    valid_0's auc: 0.63335
[60]    valid_0's auc: 0.632885
[65]    valid_0's auc: 0.632096
[70]    valid_0's auc: 0.631281
[75]    valid_0's auc: 0.631166
[80]    valid_0's auc: 0.631285
[85]    valid_0's auc: 0.630086
[90]    valid_0's auc: 0.628877
[95]    valid_0's auc: 0.627951
[100]   valid_0's auc: 0.627761
[105]   valid_0's auc: 0.627595
[110]   valid_0's auc: 0.627026
[115]   valid_0's auc: 0.6268
[120]   valid_0's auc: 0.62686
[125]   valid_0's auc: 0.626316
[130]   valid_0's auc: 0.626003
[135]   valid_0's auc: 0.625192
[140]   valid_0's auc: 0.624396
[145]   valid_0's auc: 0.623808
[150]   valid_0's auc: 0.624093
[155]   valid_0's auc: 0.624021
[160]   valid_0's auc: 0.624527
[165]   valid_0's auc: 0.624202
[170]   valid_0's auc: 0.623828
[175]   valid_0's auc: 0.62206
[180]   valid_0's auc: 0.621983
[185]   valid_0's auc: 0.620443
[190]   valid_0's auc: 0.620537
[195]   valid_0's auc: 0.620445
[200]   valid_0's auc: 0.620541
CPU times: user 15.2 s, sys: 176 ms, total: 15.3 s
Wall time: 8.05 s
```

In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(model_f1, x_te_2, 'lgb_selectkbest')
```

3. Feature Extraction using PCA

In [ ]:

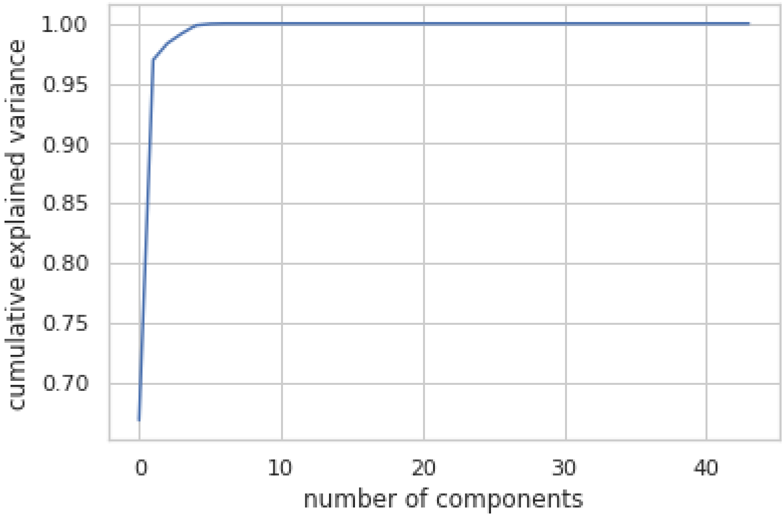
```
pca = PCA(n_components=44)
pca.fit(x_tr)
```

Out[ ]:

```
PCA(copy=True, iterated_power='auto', n_components=44, random_state
=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

In [ ]:

```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



- From the above plot we can say that < 10 components covers more than 98% variance.
- Let's take 10 components and again apply pca on top of that.

In [ ]:

```
Pca = PCA(n_components=10)
pca = Pca.fit(x_tr)
x_tr_3 =  pca.transform(x_tr)
x_val_3 = pca.transform(x_val)
x_te_3 = pca.transform(x_te)
```

Light GBM

In [ ]:

```
tr_final = lgb.Dataset(x_tr_3, y_tr)
val_final = lgb.Dataset(x_val_3, y_val)
```

In [ ]:

```
%time model_f1 = lgb.train(params, train_set=tr_final,  valid_sets=val_final, ve
rbose_eval=5)
```

```
/usr/local/lib/python3.6/dist-packages/lightgbm/engine.py:118: User
Warning: Found `num_rounds` in params. Will use it instead of argum
ent
  warnings.warn("Found `{}` in params. Will use it instead of argum
ent".format(alias))
```

```
[5]	valid_0's auc: 0.560003
[10]	valid_0's auc: 0.563377
[15]	valid_0's auc: 0.566081
[20]	valid_0's auc: 0.566285
[25]	valid_0's auc: 0.566947
[30]	valid_0's auc: 0.567204
[35]	valid_0's auc: 0.56764
[40]	valid_0's auc: 0.567124
[45]	valid_0's auc: 0.566807
[50]	valid_0's auc: 0.568582
[55]	valid_0's auc: 0.567963
[60]	valid_0's auc: 0.568477
[65]	valid_0's auc: 0.567805
[70]	valid_0's auc: 0.567687
[75]	valid_0's auc: 0.568331
[80]	valid_0's auc: 0.56801
[85]	valid_0's auc: 0.56773
[90]	valid_0's auc: 0.566861
[95]	valid_0's auc: 0.566693
[100]	valid_0's auc: 0.566995
[105]	valid_0's auc: 0.566919
[110]	valid_0's auc: 0.566746
[115]	valid_0's auc: 0.567131
[120]	valid_0's auc: 0.566123
[125]	valid_0's auc: 0.566307
[130]	valid_0's auc: 0.565756
[135]	valid_0's auc: 0.565634
[140]	valid_0's auc: 0.564942
[145]	valid_0's auc: 0.564363
[150]	valid_0's auc: 0.564468
[155]	valid_0's auc: 0.564845
[160]	valid_0's auc: 0.564343
[165]	valid_0's auc: 0.564368
[170]	valid_0's auc: 0.564083
[175]	valid_0's auc: 0.564004
[180]	valid_0's auc: 0.563917
[185]	valid_0's auc: 0.563724
[190]	valid_0's auc: 0.56368
[195]	valid_0's auc: 0.564048
[200]	valid_0's auc: 0.563873
CPU times: user 12.5 s, sys: 123 ms, total: 12.6 s
Wall time: 6.64 s
```

In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(model_f1, x_te_3, 'lgb_pca')
```

```
Making predictions
Saving predictions
Done!
```

Summary

- From the above feature selection methods, SelectKbest gives the best result compare to PCA and DT based features.
- We will go with features selected by selectKbest model for deep learning based architectures.

3. Deep Learning

- Let's try Deep learning models on our data sets and see the results.

MLP model

- Let's take our data into tf.data pipeline to apply Deep learning on our data points.

In [ ]:

```
BATCH_SIZE = 64
FEATURES = 42
```

In [ ]:

```
train_data = tf.data.Dataset.from_tensor_slices((x_tr, y_tr))
val_data = tf.data.Dataset.from_tensor_slices((x_tr, y_tr))
test_data = tf.data.Dataset.from_tensor_slices((x_te))
```

In [ ]:

```
tr_batch = train_data.batch(BATCH_SIZE, drop_remainder=True)
val_batch = val_data.batch(BATCH_SIZE, drop_remainder=True)
te_batch = test_data.batch(BATCH_SIZE, drop_remainder=True)
```

Model Architecture

In [ ]:

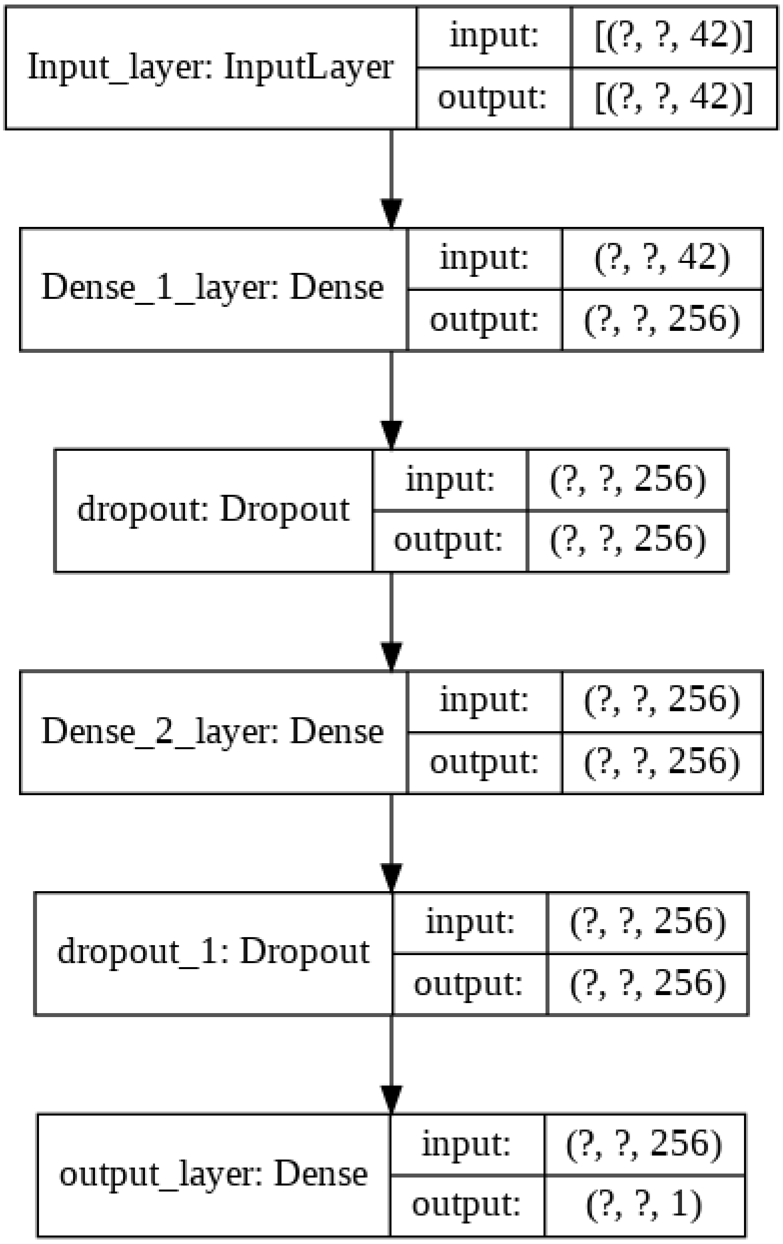
```
def define_model(BATCH_SIZE, FEATURES):
  '''Function to define model'''
  tf.keras.backend.clear_session()
  tf.random.set_seed(1234)
  input = Input(shape=(None, FEATURES), name='Input_layer')

  x = Dense(256,
            activation=tf.keras.activations.relu,
            use_bias=True,
            kernel_initializer=tf.keras.initializers.glorot_uniform(seed=78),
            bias_initializer=tf.keras.initializers.zeros(),
            name='Dense_1_layer')(input)

  dropout_1 = Dropout(0.5)(x)

  x = Dense(256,
            activation=tf.keras.activations.relu,
            use_bias=True,
            kernel_initializer=tf.keras.initializers.glorot_uniform(seed=25),
            bias_initializer=tf.keras.initializers.zeros(),
            name='Dense_2_layer')(dropout_1)

  dropout_2 = Dropout(0.5)(x)

  output = Dense(1, activation='sigmoid', name='output_layer')(dropout_2)

  model = Model(inputs=input, outputs=output, name="MLP_model")

  model.summary()

  return model
```

In [ ]:

```
model = define_model(BATCH_SIZE, FEATURES)
tf.keras.utils.plot_model(model, to_file='model.png', show_shapes=True,
                            show_layer_names=True, rankdir='TB', dpi=96)
```

```
Model: "MLP_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_layer (InputLayer)     [(None, None, 42)]        0
_________________________________________________________________
Dense_1_layer (Dense)        (None, None, 256)         11008
_________________________________________________________________
dropout (Dropout)            (None, None, 256)         0
_________________________________________________________________
Dense_2_layer (Dense)        (None, None, 256)         65792
_________________________________________________________________
dropout_1 (Dropout)          (None, None, 256)         0
_________________________________________________________________
output_layer (Dense)         (None, None, 1)           257
=================================================================
Total params: 77,057
Trainable params: 77,057
Non-trainable params: 0
_________________________________________________________________
```

Out[ ]:



Loss function and Optimizer

In [ ]:

```
loss = tf.keras.losses.BinaryCrossentropy()
optimizer = tf.keras.optimizers.Adam(1e-4)
```

In [ ]:

```
def auc(y_true, y_pred):
    return tf.py_function(roc_auc_score, (y_true, y_pred), tf.double)
```

Compile and fit the model

In [ ]:

```
from tensorflow.keras.callbacks import EarlyStopping
early_stoppings = EarlyStopping(monitor='val_loss', patience=2,verbose=1, mode=
'min')
```

In [ ]:

```
%load_ext tensorboard
```

In [ ]:

```
log_dir = "logs/fit/"
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram
_freq=1)
```

In [ ]:

```
model.compile(loss=loss, optimizer=optimizer, metrics=[auc])
```

In [ ]:

```
EPOCHS = 10
print("Fit model on training data")
history = model.fit(x_tr, y_tr,
                    batch_size=128, epochs=EPOCHS,
                    validation_data=(x_val, y_val),
                    callbacks=[tensorboard_callback, early_stoppings])
```

```
Fit model on training data
Epoch 1/10
WARNING:tensorflow:Model was constructed with shape (None, None, 4
2) for input Tensor("Input_layer:0", shape=(None, None, 42), dtype=
float32), but it was called on an input with incompatible shape (No
ne, 42).
WARNING:tensorflow:Model was constructed with shape (None, None, 4
2) for input Tensor("Input_layer:0", shape=(None, None, 42), dtype=
float32), but it was called on an input with incompatible shape (No
ne, 42).
1172/1172 [==============================] - ETA: 0s - loss: 0.6960
- auc: 0.5001WARNING:tensorflow:Model was constructed with shape (N
one, None, 42) for input Tensor("Input_layer:0", shape=(None, None,
42), dtype=float32), but it was called on an input with incompatibl
e shape (None, 42).
1172/1172 [==============================] - 13s 11ms/step - loss:
0.6960 - auc: 0.5001 - val_loss: 0.6792 - val_auc: 0.5000
Epoch 2/10
1172/1172 [==============================] - 12s 10ms/step - loss:
0.6948 - auc: 0.4999 - val_loss: 0.6792 - val_auc: 0.5000
Epoch 3/10
1172/1172 [==============================] - 12s 10ms/step - loss:
0.6994 - auc: 0.5000 - val_loss: 0.6792 - val_auc: 0.5000
Epoch 00003: early stopping
```

In [ ]:

```
%tensorboard --logdir logs/fit
```

Inference

In [ ]:

```
# Apply model on test data and save predictions
predict_and_save(model, x_te, 'mlp')
```

```
Making predictions
WARNING:tensorflow:Model was constructed with shape (None, None, 4
2) for input Tensor("Input_layer:0", shape=(None, None, 42), dtype=
float32), but it was called on an input with incompatible shape (No
ne, 42).
Saving predictions
Done!
```

- We have applied MLP architecture on our data set.

4. Summary and Conclusion

In [1]:

```python
from prettytable import PrettyTable
x = PrettyTable()
x.field_names = ["Model", "train_auc","val_auc", 'Kaggle score']
```

In [ ]:

```python
x.add_row(["1. LogisticRegression", 0.54, 0.54, 0.53])
x.add_row(["2. SVM", 0.53, 0.52, 0.50])
x.add_row(["3. Decision Tree", 0.67, 0.63, 0.58])
x.add_row(["4. RF", 0.99, 0.67, 0.57])
x.add_row(["5. XgBoost", 0.96, 0.66, 0.57])
x.add_row(["6. LightGBM", '-', 0.63, 0.61])
x.add_row(["7. Voting Classifier", '-', '-', 0.58])
x.add_row(["8. Cascading Classifier", '-', '-', 0.56])
x.add_row(["9. DT - Feature Importance - DT", '-', '-', 0.59])
x.add_row(["10. LightGBM - Feature Importance - DT", '-', 0.63, 0.55])
x.add_row(["11. LightGBM - Feature Selection - SelectKBest", '-', 0.64, 0.61])
x.add_row(["12. LightGBM - Feature Extraction - PCA", '-', 0.56, 0.52])
x.add_row(["13. Deep Learning - MLP", 0.50, 0.50, 0.50])
```

In [ ]:

```python
print(x)
```

```
+------------------------------------------------+-----------+-----
----+--------------+
|                     Model                      | train_auc | val_
auc | Kaggle score |
+------------------------------------------------+-----------+-----
----+--------------+
|             1. LogisticRegression              |    0.54   |   0.
54  |     0.53     |
|                     2. SVM                     |    0.53   |   0.
52  |     0.5      |
|                3. Decision Tree                |    0.67   |   0.
63  |     0.58     |
|                     4. RF                      |    0.99   |   0.
67  |     0.57     |
|                   5. XgBoost                   |    0.96   |   0.
66  |     0.57     |
|                  6. LightGBM                   |     -     |   0.
63  |     0.61     |
|              7. Voting Classifier              |     -     |    -
|     0.58     |
|            8. Cascading Classifier             |     -     |    -
|     0.56     |
|        9. DT - Feature Importance - DT         |     -     |    -
|     0.59     |
|     10. LightGBM - Feature Importance - DT     |     -     |   0.
63  |     0.55     |
| 11. LightGBM - Feature Selection - SelectKBest |     -     |   0.
64  |     0.61     |
|    12. LightGBM - Feature Extraction - PCA     |     -     |   0.
56  |     0.52     |
|            13. Deep Learning - MLP             |    0.5    |   0.
5   |     0.5      |
+------------------------------------------------+-----------+-----
----+--------------+
```

Summary

-> EDA

- We have analyzed each and every feature from train and validation dataset with the help of different plots like barplot, PDF and CDF.
- We have also analyzed missing values and their percentages in features.

-> Feature Engineering

- From songs, songs_extra_info and members we have combined all information related to user and songs and extracted various features.
- We have extracted features like membership days, information of year, month and day from registration and expiration dates.
- We have also extracted groupby features for songs and users with respect to artist, lyricist, composer, language, age etc.
- We have extracted features like song year, country code, registration code from isrc code for each and every song. Along with that we have extracted counts like artist count, genre count, lyricist count, composer count etc.

-> Sampling

- As data size is very large to fit our RAM so we have samplled 1.5M train points and 0.7 val points.
- Our data is in chronological order.

-> Pre-processing

- We have transform all our numerical features using standardization.
- For categorical features we have used label encoding.
- As one hot encoding will result into large dimensionality which may result less performance.

-> Models

- We have applied various Machine learning algorithms like LR, SVM, DT, RF, GBDT (using XgBoost, LightGBM), Stacking classifier, Voting classifier.
- We have done hyper parameter tuning to get best result.
- We have also applied MLP architecture.
- We can see that LightGBM gives better performance compare to other models.

-> Feature Extraction and Selection

- We have selected best features based on DT feature importance and applied LighGBM model.
- We also used sklearn's SelectKBest method to choose best features and applied LightGBM.
- We have also tried PCA and find the maximum explaiend variance using CDF.
- We can see that LightGBM with SelectKBest gives best perforamnce.

-> Future steps

- Due to RAM limitations we have taken less amount of data.
- If we use whole data we can get better results.
- Deep learning requires more data to get good results.
- By tweaking parameters on large data points we can achieve better results.
- We can also think of more feature extraction.