import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sn

train = pd.read\_csv('train\_v2.csv')

sample\_submission = pd.read\_csv('sample\_submission\_v2.csv')

transactions = pd.read\_csv('transactions\_v2.csv')

user\_logs = pd.read\_csv('user\_logs\_v2.csv')

members = pd.read\_csv('members\_v3.csv')

# set the options so the output format can be displayed correctly

pd.set\_option('expand\_frame\_repr', True)

pd.set\_option('display.max\_rows', 30000000)

pd.set\_option('display.max\_columns', 100)

# check the number of duplicate accounts in each table

train.duplicated('msno').sum()

sample\_submission.duplicated('msno').sum()

transactions.duplicated('msno').sum()

user\_logs.duplicated('msno').sum()

members.duplicated('msno').sum()

# returns the max value of numerical variables and membership\_expire\_date

# returns the min value of transaction date

# returns the mode of ordinal variable and dummy variables, if multiple values share the same frequency, keep the first one

transactions\_v2 = transactions.groupby('msno', as\_index = False).agg({'payment\_method\_id': lambda x:x.value\_counts().index[0], 'payment\_plan\_days': 'max', 'plan\_list\_price': 'max',

'actual\_amount\_paid': 'max', 'is\_auto\_renew': lambda x:x.value\_counts().index[0], 'transaction\_date': 'min', 'membership\_expire\_date': 'max',

'is\_cancel': lambda x:x.value\_counts().index[0]})

# returns the max value of date and number of unique songs

# returns the sum of other variables

user\_logs\_v2 = user\_logs.groupby('msno', as\_index = False).agg({'date': 'max', 'num\_25': 'sum', 'num\_50': 'sum', 'num\_75': 'sum',

'num\_985': 'sum', 'num\_100': 'sum', 'num\_unq': 'max', 'total\_secs': 'sum'})

# calculate the percentage of number of songs played within certain period

user\_logs\_v2['percent\_25'] = user\_logs\_v2['num\_25']/(user\_logs\_v2['num\_25']+user\_logs\_v2['num\_50']+user\_logs\_v2['num\_75']+user\_logs\_v2['num\_985']+user\_logs\_v2['num\_100'])

user\_logs\_v2['percent\_50'] = user\_logs\_v2['num\_50']/(user\_logs\_v2['num\_25']+user\_logs\_v2['num\_50']+user\_logs\_v2['num\_75']+user\_logs\_v2['num\_985']+user\_logs\_v2['num\_100'])

user\_logs\_v2['percent\_100'] = (user\_logs\_v2['num\_985']+user\_logs\_v2['num\_100'])/(user\_logs\_v2['num\_25']+user\_logs\_v2['num\_50']+user\_logs\_v2['num\_75']+user\_logs\_v2['num\_985']+user\_logs\_v2['num\_100'])

# drop useless variables

user\_logs\_v3 = user\_logs\_v2.drop(columns = ['num\_25', 'num\_50', 'num\_75', 'num\_985', 'num\_100'])

# merge between different tables for modelling purpose

dataset\_train = train.merge(members, on = 'msno', how = 'left').merge(transactions\_v2, on = 'msno', how = 'left').merge(user\_logs\_v3, on = 'msno', how = 'left')

dataset\_train.dtypes

# date in csv will be recognized as float in python

# this value needs to be converted back to date

dataset\_train['registration\_init\_time'] = pd.to\_datetime(dataset\_train['registration\_init\_time'], format = '%Y%m%d')

dataset\_train['transaction\_date'] = pd.to\_datetime(dataset\_train['transaction\_date'], format = '%Y%m%d')

dataset\_train['membership\_expire\_date'] = pd.to\_datetime(dataset\_train['membership\_expire\_date'], format = '%Y%m%d')

dataset\_train['date'] = pd.to\_datetime(dataset\_train['date'], format = '%Y%m%d')

# check the maximum of datetime value

dataset\_train.select\_dtypes(include = ['datetime64[ns]']).max()

# create new day columns for modelling purpose

dataset\_train['registration\_day'] = (dataset\_train['membership\_expire\_date'].max() - dataset\_train['registration\_init\_time']).astype('timedelta64[D]')

dataset\_train['transaction\_day'] = (dataset\_train['membership\_expire\_date'].max() - dataset\_train['transaction\_date']).astype('timedelta64[D]')

dataset\_train['membership\_expire\_day'] = (dataset\_train['membership\_expire\_date'].max() - dataset\_train['membership\_expire\_date']).astype('timedelta64[D]')

dataset\_train['last\_play\_day'] = (dataset\_train['membership\_expire\_date'].max() - dataset\_train['date']).astype('timedelta64[D]')

# check the distribution of age due to the documentation explanation

dataset\_train['bd'].value\_counts()

# remove gender and age since missing value or incorrect value is over 50%

dataset\_train\_v2 = dataset\_train.drop(columns = ['msno', 'gender', 'bd', 'registration\_init\_time', 'transaction\_date', 'membership\_expire\_date', 'date'])

dataset\_train\_v2.dtypes

# check the number of missing values in each column

dataset\_train\_v2.isna().sum()

# Handle missing value of part of numeric columns by using mode

def replacemode(i):

dataset\_train\_v2[i] = dataset\_train\_v2[i].fillna(dataset\_train\_v2[i].value\_counts().index[0])

return

replacemode('city')

replacemode('registered\_via')

replacemode('payment\_method\_id')

replacemode('payment\_plan\_days')

replacemode('is\_auto\_renew')

replacemode('is\_cancel')

# Handle missing value of part of numeric columns by using mean

from sklearn.impute import SimpleImputer

mean\_imputer = SimpleImputer(missing\_values = np.nan, strategy = 'mean')

def replacemean(i):

dataset\_train\_v2[i] = mean\_imputer.fit\_transform(dataset\_train\_v2[[i]])

return

replacemean('plan\_list\_price')

replacemean('actual\_amount\_paid')

replacemean('num\_unq')

replacemean('total\_secs')

replacemean('percent\_25')

replacemean('percent\_50')

replacemean('percent\_100')

replacemean('registration\_day')

replacemean('transaction\_day')

replacemean('membership\_expire\_day')

replacemean('last\_play\_day')

# Handle outliers by using capping

def replaceoutlier(i):

mean, std = np.mean(dataset\_train\_v2[i]), np.std(dataset\_train\_v2[i])

cut\_off = std\*3

lower, upper = mean - cut\_off, mean + cut\_off

dataset\_train\_v2[i][dataset\_train\_v2[i] < lower] = lower

dataset\_train\_v2[i][dataset\_train\_v2[i] > upper] = upper

return

replaceoutlier('plan\_list\_price')

replaceoutlier('actual\_amount\_paid')

replaceoutlier('num\_unq')

replaceoutlier('total\_secs')

replaceoutlier('percent\_25')

replaceoutlier('percent\_50')

replaceoutlier('percent\_100')

replaceoutlier('registration\_day')

replaceoutlier('transaction\_day')

replaceoutlier('membership\_expire\_day')

replaceoutlier('last\_play\_day')

dataset\_train\_v2.dtypes

dataset\_train\_v2.describe()

# convert categorical variables into string for get\_dummies

dataset\_train\_v2.iloc[:, 1:4] = dataset\_train\_v2.iloc[:, 1:4].astype(str)

# create dummy variables for modelling purpose

dataset\_train\_v3 = pd.get\_dummies(dataset\_train\_v2, drop\_first = True)

dataset\_train\_v3.dtypes

# Feature Scaling for modelling purpose

from sklearn.preprocessing import MinMaxScaler, StandardScaler

X = dataset\_train\_v3.drop(columns = ['is\_churn'])

Y = dataset\_train\_v3['is\_churn']

nm\_X = pd.DataFrame(MinMaxScaler().fit\_transform(X))

nm\_X.columns = X.columns.values

nm\_X.index = X.index.values

sc\_X = pd.DataFrame(StandardScaler().fit\_transform(X))

sc\_X.columns = X.columns.values

sc\_X.index = X.index.values

# Visualize the correlation between independent columns

sn.set(style="white")

corr = nm\_X.corr()

mask = np.zeros\_like(corr, dtype=np.bool)

mask[np.triu\_indices\_from(mask)] = True

f, ax = plt.subplots(figsize=(18, 15))

cmap = sn.diverging\_palette(220, 10, as\_cmap=True)

sn.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,

square=True, linewidths=.5, cbar\_kws={"shrink": .5})

sn.set(style="white")

corr2 = sc\_X.corr()

mask = np.zeros\_like(corr2, dtype=np.bool)

mask[np.triu\_indices\_from(mask)] = True

f, ax = plt.subplots(figsize=(18, 15))

cmap = sn.diverging\_palette(220, 10, as\_cmap=True)

sn.heatmap(corr2, mask=mask, cmap=cmap, vmax=.3, center=0,

square=True, linewidths=.5, cbar\_kws={"shrink": .5})

# Remove those columns with high correlation values

nm\_X\_v2 = nm\_X.drop(columns = ['membership\_expire\_day', 'percent\_100', 'registered\_via\_9.0', 'payment\_method\_id\_38.0'])

sc\_X\_v2 = sc\_X.drop(columns = ['membership\_expire\_day', 'percent\_100', 'registered\_via\_9.0', 'payment\_method\_id\_38.0'])

# Feature Selection

from sklearn.feature\_selection import SelectKBest, chi2, f\_classif

nm\_col = ['payment\_plan\_days', 'plan\_list\_price', 'actual\_amount\_paid', 'num\_unq', 'total\_secs', 'percent\_25', 'percent\_50', 'registration\_day',

'transaction\_day', 'last\_play\_day']

nm\_X\_v3 = nm\_X\_v2.drop(columns = nm\_col)

nm\_X\_v4 = pd.DataFrame(nm\_X\_v2, columns = nm\_col)

nm\_X\_v5 = pd.DataFrame(SelectKBest(score\_func=chi2, k='all').fit(nm\_X\_v3, Y).pvalues\_ <= 0.05, columns = ['importance'])

nm\_X\_v5.index = nm\_X\_v3.columns.values

nm\_X\_v6 = pd.DataFrame(SelectKBest(score\_func=f\_classif, k='all').fit(nm\_X\_v4, Y).pvalues\_ <= 0.05, columns = ['importance'])

nm\_X\_v6.index = nm\_X\_v4.columns.values

nm\_X\_v7 = pd.concat([nm\_X\_v5, nm\_X\_v6])

nm\_selected = list(pd.Series(nm\_X\_v7[nm\_X\_v7['importance'] == 1].index.values))

nm\_X\_v8 = pd.DataFrame(nm\_X\_v2, columns = nm\_selected)

sc\_X\_v3 = pd.DataFrame(SelectKBest(score\_func=f\_classif, k='all').fit(sc\_X\_v2, Y).pvalues\_ <= 0.05, columns = ['importance'])

sc\_X\_v3.index = sc\_X\_v2.columns.values

sc\_selected = list(pd.Series(sc\_X\_v3[sc\_X\_v3['importance'] == 1].index.values))

sc\_X\_v4 = pd.DataFrame(sc\_X\_v2, columns = sc\_selected)

# Reduce Dimension since we still have too many features on standardized results

from sklearn.decomposition import PCA

pca = PCA()

pca.fit\_transform(sc\_X\_v4)

np.cumsum(pca.explained\_variance\_ratio\_)

sc\_X\_v5 = PCA(n\_components=50).fit\_transform(sc\_X\_v4)

pca\_v2 = PCA()

pca\_v2.fit\_transform(nm\_X\_v8)

np.cumsum(pca\_v2.explained\_variance\_ratio\_)

nm\_X\_v9 = PCA(n\_components=25).fit\_transform(nm\_X\_v8)

# Split into train and test Set

from sklearn.model\_selection import train\_test\_split

nm\_X\_train, nm\_X\_test, nm\_Y\_train, nm\_Y\_test = train\_test\_split(nm\_X\_v9, Y, test\_size = 0.3, random\_state = 0)

sc\_X\_train, sc\_X\_test, sc\_Y\_train, sc\_Y\_test = train\_test\_split(sc\_X\_v5, Y, test\_size = 0.3, random\_state = 0)

# Fit training set into different algorithms

from sklearn.linear\_model import LogisticRegression

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import BaggingClassifier, AdaBoostClassifier, GradientBoostingClassifier, RandomForestClassifier, ExtraTreesClassifier

from xgboost import XGBClassifier

nm\_models = []

nm\_models.append(('KNN', KNeighborsClassifier()))

nm\_models.append(('LR', LogisticRegression()))

nm\_models.append(('LDA', LinearDiscriminantAnalysis()))

nm\_models.append(('QDA', QuadraticDiscriminantAnalysis()))

nm\_models.append(('CART', DecisionTreeClassifier()))

nm\_models.append(('NB', GaussianNB()))

nm\_models.append(('Linear SVM', SVC(kernel = 'linear')))

nm\_models.append(('Kernel SVM', SVC(kernel = 'rbf')))

sc\_models = []

sc\_models.append(('KNN', KNeighborsClassifier()))

sc\_models.append(('LR', LogisticRegression()))

sc\_models.append(('LDA', LinearDiscriminantAnalysis()))

sc\_models.append(('QDA', QuadraticDiscriminantAnalysis()))

sc\_models.append(('CART', DecisionTreeClassifier()))

sc\_models.append(('NB', GaussianNB()))

sc\_models.append(('Linear SVM', SVC(kernel = 'linear')))

sc\_models.append(('Kernel SVM', SVC(kernel = 'rbf')))

ensembles = []

ensembles.append(('BC', BaggingClassifier(base\_estimator=DecisionTreeClassifier())))

ensembles.append(('AB', AdaBoostClassifier(base\_estimator=DecisionTreeClassifier())))

ensembles.append(('GBM', GradientBoostingClassifier()))

ensembles.append(('RF', RandomForestClassifier()))

ensembles.append(('ET', ExtraTreesClassifier()))

ensembles.append(('XGB', XGBClassifier()))

from sklearn.model\_selection import cross\_val\_score

results = []

names = []

for name, model in nm\_models:

nm\_cv\_results = cross\_val\_score(model, nm\_X\_train, nm\_Y\_train, cv=10, scoring='f1', n\_jobs = -1)

results.append(nm\_cv\_results)

names.append(name)

msg = "%s: %f (%f)" % (name, nm\_cv\_results.mean(), nm\_cv\_results.std())

print(msg)

results2 = []

names2 = []

for name2, model2 in sc\_models:

sc\_cv\_results = cross\_val\_score(model2, sc\_X\_train, sc\_Y\_train, cv=10, scoring='f1', n\_jobs = -1)

results2.append(sc\_cv\_results)

names2.append(name2)

msg2 = "%s: %f (%f)" % (name2, sc\_cv\_results.mean(), sc\_cv\_results.std())

print(msg2)

results3 = []

names3 = []

for name3, model3 in ensembles:

en\_cv\_results = cross\_val\_score(model3, sc\_X\_train, sc\_Y\_train, cv=10, scoring='f1', n\_jobs = -1)

results3.append(en\_cv\_results)

names3.append(name3)

msg3 = "%s: %f (%f)" % (name3, en\_cv\_results.mean(), en\_cv\_results.std())

print(msg3)

results4 = []

names4 = []

for name4, model4 in ensembles:

en\_cv\_results2 = cross\_val\_score(model4, nm\_X\_train, nm\_Y\_train, cv=10, scoring='f1', n\_jobs = -1)

results4.append(en\_cv\_results2)

names4.append(name4)

msg4 = "%s: %f (%f)" % (name4, en\_cv\_results2.mean(), en\_cv\_results2.std())

print(msg4)

# Apply Grid Search on Random Forest since it returns the best result on Cross Validation among all models

from sklearn.model\_selection import GridSearchCV

parameters = {"max\_depth": [20],

"max\_features": [15],

"min\_samples\_split": [15],

"min\_samples\_leaf": [5],

"criterion": ["entropy"]}

grid\_search\_RF = GridSearchCV(estimator = RandomForestClassifier(), param\_grid = parameters, scoring = "f1", cv = 10, n\_jobs = -1)

grid\_result\_RF = grid\_search\_RF.fit(nm\_X\_train, nm\_Y\_train)

print("Best: %f using %s" % (grid\_result\_RF.best\_score\_, grid\_result\_RF.best\_params\_))

# max\_depth - 20 / max\_features - 15 / min\_samples\_leaf - 15 / min\_samples\_split - 5

# Apply Grid Search on Bagging since it returns the second best result on Cross Validation among all models

parameters2 = {"max\_samples": [0.8],

"max\_features": [0.8]}

grid\_search\_BC = GridSearchCV(estimator = BaggingClassifier(base\_estimator=DecisionTreeClassifier()), param\_grid = parameters2, scoring = "f1", cv = 10, n\_jobs = -1)

grid\_result\_BC = grid\_search\_BC.fit(nm\_X\_train, nm\_Y\_train)

print("Best: %f using %s" % (grid\_result\_BC.best\_score\_, grid\_result\_BC.best\_params\_))

# max\_samples - 0.8 / max\_features - 0.8

# Evaluate tuned model result on test dataset based on random forest result because it returns the best result

from sklearn.metrics import confusion\_matrix, accuracy\_score, f1\_score, precision\_score, recall\_score

nm\_Y\_predict = grid\_result\_RF.predict(nm\_X\_test)

acc = accuracy\_score(nm\_Y\_test, nm\_Y\_predict)

prec = precision\_score(nm\_Y\_test, nm\_Y\_predict)

rec = recall\_score(nm\_Y\_test, nm\_Y\_predict)

f1 = f1\_score(nm\_Y\_test, nm\_Y\_predict)

model\_results = pd.DataFrame([['Random Forest', acc, prec, rec, f1]],

columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1'])

# F1 Score – 83.3%