A Data Science Framework

1. **Define the Problem:** Predict whether the customer will purchase again in the next 12 month afte
2. **Gather the Data:** Get the raw from a ecommerce store via BigQuery tabales
3. **Prepare Data for Consumption:**

- (1) Calculate 12 month purchase frequency and CLV
- (2) Clean-up
- (3) Handling missing data

4. **Perform Exploratory Analysis:**

- (1) Basic descriptive statistics: min, mean, median, quantiles, max
- (2) Check distributions
- (3) Correlations

Imbalanced classifiction problem

5. **Feature Engineering**

- (1) Creating dummy variables for: binary features and low cardinal categorical features
- (2) Target encoding for high cardinal categorical features

6. **Modeling and hyperparameter tunning :**

- Logistic Regression
- Decision Tree
- RandomForest
- Grid search_cv

7. **Evaluation and model selection :**
8. **Optimize and Strategize:**

3.Prepare Data for Consumption:

```
# from google.cloud import bigquery
# from pandas.io import gbq
# from google.colab import auth
# auth.authenticate_user()
import os
```

```
import glob
import datetime
import pandas as pd
import numpy as np
import sys

import pandas_profiling as pp
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)

#Common Model Helpers
from sklearn import preprocessing
!pip install category_encoders
from category_encoders import TargetEncoder
# from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn import feature_selection
from sklearn import model_selection
from sklearn import metrics
# from scipy.spatial.distance import cdist
# from sklearn import cluster, tree, decomposition

#Common Model Algorithms
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn import svm, tree, linear_model, neighbors, naive_bayes, ensemble, discrim
from xgboost import XGBClassifier

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
# from sklearn.learning_curve import validation_curve
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier


#Visualization
import matplotlib as mpl
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.pylab as pylab
import seaborn as sns
from pandas.plotting import scatter_matrix
```

```
Requirement already satisfied: category_encoders in /usr/local/lib/python3.6/dist
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: pandas>=0.21.1 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.6/dist-pac
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.6/dist-pack
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.6/dis
```

```
# find the unique values in each column
def unique_counts(customer):
    for i in customer.columns:
        count=customer[i].nunique()
        print(i, ':', count)


#print missing value table
def missing_values_table(df):
        mis_val = df.isnull().sum()
        mis_val_percent = 100 * df.isnull().sum() / len(df)
        mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)
        mis_val_table_ren_columns = mis_val_table.rename(
        columns = {0 : 'Missing Values ', 1 : '% of Total Values'})
        mis_val_table_ren_columns = mis_val_table_ren_columns[
            mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)
        print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
            "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")
        return mis_val_table_ren_columns
```

```
from google.colab import files
uploaded = files.upload()
```

Choose Files clv_file.csv

- **clv_file.csv**(text/csv) - 85036618 bytes, last modified: 6/11/2020 - 100% done

```
Saving clv_file.csv to clv_file.csv
```

```
customers = pd.read_csv('clv_file.csv')
customers.shape
```

```
(536811, 27)
```

```
customers.columns
```

```
Index(['Unnamed: 0', 'uid', 'buyer_accepts_marketing', 'order_date',
```

```
# check categorical variables
for i in customers.columns:
    if customers[i].dtypes == 'O':
        print(i, ': ', customers[i].nunique())
```

```
order_date :  1394
landing_site :  46463
landing_site_ref :  2
referring_site :  22898
tags :  2
utm_campaign :  87
utm_source :  44
utm_medium :  12
utm_term :  21019
utm_content :  18
first_order_date :  1394
Spotify :  1
Dotdigital :  1
AE :  1
country_code :  158
province_code :  516
```

```
missing_values_table(customers)
```

```
Your selected dataframe has 27 columns.
There are 14 columns that have missing values.
```

	Missing Values	% of Total Values
landing_site_ref	536809	100.0
utm_content	536232	99.9
tags	534354	99.5
Spotify	525880	98.0
utm_term	515111	96.0
AE	509900	95.0
utm_campaign	483606	90.1
utm_medium	483476	90.1
utm_source	483418	90.1
Dotdigital	376288	70.1
referring_site	182165	33.9
province_code	67069	12.5
country_code	11932	2.2
landing_site	2554	0.5

```
# drop customers with first order < =0
customers = customers[customers['order_revenue'] > 0]
customers.shape
```

```
(532565, 27)
```

```
customers.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 532565 entries, 0 to 536810
Data columns (total 27 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   Unnamed: 0               532565 non-null  int64
 1   uid                      532565 non-null  int64
 2   buyer_accepts_marketing  532565 non-null  bool
 3   order_date               532565 non-null  object
 4   order_id                 532565 non-null  int64
 5   order_revenue            532565 non-null  float64
 6   total_discounts          532565 non-null  float64
 7   landing_site             530163 non-null  object
 8   landing_site_ref         2 non-null       object
 9   referring_site           352885 non-null  object
 10  tags                     2457 non-null    object
 11  utm_campaign             53157 non-null   object
 12  utm_source               53345 non-null   object
 13  utm_medium               53287 non-null   object
 14  utm_term                 21681 non-null   object
 15  utm_content              579 non-null     object
 16  first_order_date         532565 non-null  object
 17  Spotify                  10840 non-null   object
 18  Dotdigital               159109 non-null  object
 19  AE                       26639 non-null   object
 20  country_code             520863 non-null  object
 21  province_code            465768 non-null  object
 22  requires_shipping        532565 non-null  bool
 23  item_qty                 532565 non-null  int64
 24  unique_item              532565 non-null  int64
 25  revenue_12m              532565 non-null  float64
 26  order_count_12m          532565 non-null  int64
dtypes: bool(2), float64(3), int64(6), object(16)
memory usage: 106.7+ MB
```

Formatting

```
customers.rename(columns={'order_revenue':
                   'first_order_revenue',
                  'total_discounts':
                   'first_order_discount' }, inplace= True)
customers['revenue_12m']=customers['revenue_12m'].astype(float)
customers['first_order_discount']=customers['first_order_discount'].astype(float)
```

```
customers['first_order_revenue']=customers['first_order_revenue'].astype(float)

# add new day_of_week column
customers['order_date']=pd.to_datetime(customers['order_date'])
customers['day_of_week']=customers['order_date'].dt.dayofweek
customers['day_of_week'] =customers['day_of_week'].astype(str)


# create target column repeat_purchase
# customers['repeat_purchase'] = ~(customers.order_count_12m < 2)
customers['repeat_purchase'] = np.where(customers.order_count_12m >1,  1, 0)
```

Handling missing values

(1) filling null based on business definitions

```
customers['province_code'].fillna("unknown", inplace = True)
customers['country_code'].fillna("unknown", inplace = True)
```

```
cols_to_fill = ['utm_campaign', 'utm_medium','utm_source', 'referring_site', 'Dotdigit
for i in cols_to_fill:
  customers[i+'_known'] = customers[i].notnull()

cols_to_drop=['landing_site_ref','tags', 'utm_term','utm_content','landing_site',
         'utm_campaign','utm_medium','utm_source','referring_site','AE', 'Dotdigital',
customers.drop(cols_to_drop, axis=1, inplace=True)
```

```
missing_values_table(customers)
```

```
Your selected dataframe has 22 columns.
There are 0 columns that have missing values.
```

Missing Values	% of Total Values

```
unique_counts(customers)
```

```
Unnamed: 0 : 532565
uid : 532565
buyer_accepts_marketing : 2
order_date : 1394
order_id : 532565
first_order_revenue : 858
first_order_discount : 139
first_order_date : 1394
country_code : 159
```

```
for column in customers:
    unique_values = np.unique(customers[column])
    nr_values = len(unique_values)
    if nr_values <= 10:
        print("The number of values for feature {} is: {} -- {}".format(column, nr_val
    else:
        print("The number of values for feature {} is: {}".format(column, nr_values))
```

```
The number of values for feature Unnamed: 0 is: 532565
The number of values for feature uid is: 532565
The number of values for feature buyer_accepts_marketing is: 2 -- [False  True]
The number of values for feature order_date is: 1394
The number of values for feature order_id is: 532565
The number of values for feature first_order_revenue is: 858
The number of values for feature first_order_discount is: 139
The number of values for feature first_order_date is: 1394
The number of values for feature country_code is: 159
The number of values for feature province_code is: 517
The number of values for feature requires_shipping is: 2 -- [False  True]
The number of values for feature item_qty is: 21
The number of values for feature unique_item is: 1 -- [1]
The number of values for feature revenue_12m is: 3228
The number of values for feature order_count_12m is: 27
The number of values for feature day_of_week is: 7 -- ['0' '1' '2' '3' '4' '5' '6
The number of values for feature repeat_purchase is: 2 -- [0 1]
The number of values for feature utm_campaign_known is: 2 -- [False  True]
The number of values for feature utm_medium_known is: 2 -- [False  True]
The number of values for feature utm_source_known is: 2 -- [False  True]
The number of values for feature referring_site_known is: 2 -- [False  True]
The number of values for feature Dotdigital_known is: 2 -- [False  True]
```

5.Feature Engineering

- (1) Creating dummy variables for: binary features and low cardinal categorical features
- (2) Target encoding for high cardinal categorical features

```
# all customer age > = 12month
df_customer=customers.copy()
df_customer=df_customer[df_customer.first_order_date < '2019-05-01']
df_customer.shape
```

```
(378075, 21)
```

```
### Train - test split
train = df_customer[df_customer.first_order_date < '2018-10-31']
test  = df_customer[df_customer.first_order_date > '2018-10-31']
train.drop(columns=['uid','order_date','order_id','first_order_date','revenue_12m','or
test.drop(columns=['uid','order_date','order_id','first_order_date','revenue_12m','ord
train.shape, test.shape
```

```
((351858, 15), (26187, 15))
```

```
train.columns
```

```
Index(['buyer_accepts_marketing', 'first_order_revenue',
       'first_order_discount', 'country_code', 'province_code',
       'requires_shipping', 'item_qty', 'unique_item', 'day_of_week',
       'repeat_purchase', 'utm_campaign_known', 'utm_medium_known',
       'utm_source_known', 'referring_site_known', 'Dotdigital_known'],
      dtype='object')
```

```
train.groupby('repeat_purchase').size()
```

```
repeat_purchase
0    258539
1     93319
dtype: int64
```

```
test.groupby('repeat_purchase').size()
```

```
repeat_purchase
0    21430
1     4757
dtype: int64
```

Target Encoding

```
from category_encoders import TargetEncoder
encoder = TargetEncoder()
train['country_code_encoded'] = encoder.fit_transform(train['country_code'], train['re
test['country_code_encoded'] = encoder.fit_transform(test['country_code'], test['repea
train.drop(columns=['country_code','province_code'], inplace= True)
test.drop(columns=['country_code','province_code'], inplace= True)
```

```
train.shape, test.shape
```

```
((351858, 14), (26187, 14))
```

Dummy variables

```
X_train= train.drop(columns=['repeat_purchase'])
X_train= pd.get_dummies(X_train)

X_test= test.drop(columns=['repeat_purchase'])
X_test= pd.get_dummies(X_test)
print(X_train.shape, X_test.shape)
```

```
(351858, 19) (26187, 19)
```

```
y_test  = np.array(test['repeat_purchase'])
y_train =  np.array(train['repeat_purchase'])
# Convert to numpy array
features_train = np.array(X_train)
features_test = np.array(X_test)
```

6.Modeling and hyperparameter tunning :

LogisticRegression

C is the inverse of the regularization term (1/lambda). It tells the model how much large parameters a larger penalization

```
# define models and parameters
model = LogisticRegression(random_state=42,class_weight='balanced')
penalty = ['l2','l1']
c_values = [10, 1.0, 0.1, 0.01,0.0001]
param_grid = dict(penalty=penalty,C=c_values)

grid_search = GridSearchCV(estimator=model, param_grid=param_grid,  scoring='accuracy'
grid_result = grid_search.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.515944 using {'C': 0.0001, 'penalty': 'l2'}
0.411945 (0.097198) with: {'C': 10, 'penalty': 'l2'}
nan (nan) with: {'C': 10, 'penalty': 'l1'}
0.436432 (0.094083) with: {'C': 1.0, 'penalty': 'l2'}
nan (nan) with: {'C': 1.0, 'penalty': 'l1'}
0.452953 (0.115414) with: {'C': 0.1, 'penalty': 'l2'}
nan (nan) with: {'C': 0.1, 'penalty': 'l1'}


0.515944 (0.094220) with: {'C': 0.0001, 'penalty': 'l2'}
```

```
print('Best Penalty:', best_model.best_estimator_.get_params(['penalty']) )
print('Best C:', best_model.best_estimator_.get_params()['C'])
print("The mean accuracy of the model is:",best_model.score(X_train, y_train))
```

```
Best Penalty: {'C': 10000.0, 'class_weight': 'balanced', 'dual': False, 'fit_inte
Best C: 10000.0
The mean accuracy of the model is: 0.5727111505209488
```

```
# Create range of candidate penalty hyperparameter values
penalty = ['l1', 'l2']
C = np.logspace(0, 4, 10)
hyperparameters = dict(C=C, penalty=penalty)
gridsearch = GridSearchCV(LogisticRegression(random_state=42,class_weight='balanced'),
best_model = gridsearch.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:  5.6min finished
```

```
# LogisticRegressionCV  (?Can I use cross validateion in my case)
logreg1 = LogisticRegression(random_state = 42)
print(logreg1)
logreg1.fit(X_train.drop(columns=['country_code_encoded']), y_train)


logreg_pred1 = logreg1.predict(X_test.drop(columns=['country_code_encoded'])) #Predict
logreg_pred_proba1 = logreg1.predict_proba(X_test.drop(columns=['country_code_encoded'
# Probability estimates.

# The returned estimates for all classes are ordered by the label of classes.
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=42, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
confusion_matrix(y_test, logreg_pred1)
```

```
array([[21116,   314],
       [ 4606    1511]\
```

```
metrics.accuracy_score(y_test, logreg_pred1)
```

```
0.8121205178141826
```

DecisionTree

+ Code

```
import graphviz
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
from graphviz import Source
clf = DecisionTreeClassifier()
print(clf)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
%%time
# define models and parameters
model = DecisionTreeClassifier(random_state=42,class_weight="balanced")
param_grid = {'criterion': ['gini','entropy'],
              'splitter': ['best', 'random'],
              'max_depth': [2,6,10,None],
              'min_samples_split': [2,5,10],
              'min_samples_leaf': [1,3,5,10]
             }

grid_search = GridSearchCV(estimator=model, param_grid=param_grid,  scoring='accuracy'
grid_result = grid_search.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

```
Best: 0.630723 using {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf': 1,
0.630723 (0.032484) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.608163 (0.035736) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.630723 (0.032484) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.608163 (0.035736) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.630723 (0.032484) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.608163 (0.035736) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.630723 (0.032484) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.608163 (0.035736) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.630723 (0.032484) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.608163 (0.035736) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.630723 (0.032484) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.608163 (0.035736) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.630723 (0.032484) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.608163 (0.035736) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.630723 (0.032484) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.608163 (0.035736) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.630723 (0.032484) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.608163 (0.035736) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.630723 (0.032484) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.608163 (0.035736) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.630723 (0.032484) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.608163 (0.035736) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.630723 (0.032484) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.608163 (0.035736) with: {'criterion': 'gini', 'max_depth': 2, 'min_samples_leaf
0.427806 (0.104478) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.446575 (0.066252) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.427806 (0.104478) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.446382 (0.065896) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.427806 (0.104478) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.446382 (0.065896) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.427573 (0.104085) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.479313 (0.088129) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.427573 (0.104085) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.479313 (0.088129) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.427573 (0.104085) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.480541 (0.089482) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.427575 (0.104090) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.480541 (0.089482) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.427575 (0.104090) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.480541 (0.089482) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.427575 (0.104090) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.480541 (0.089482) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.427857 (0.104565) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.520622 (0.064922) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.427857 (0.104565) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.520622 (0.064922) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.427857 (0.104565) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.520622 (0.064922) with: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf
0.432112 (0.137315) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.494768 (0.104664) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.431819 (0.137352) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.469152 (0.107083) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.431873 (0.137347) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.469272 (0.101589) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.431995 (0.137164) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.459518 (0.102859) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
```

```
0.431995 (0.137164) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.459518 (0.102859) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.431415 (0.137296) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.459791 (0.114179) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.431006 (0.137024) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.462437 (0.099794) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.431006 (0.137024) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.462437 (0.099794) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.431006 (0.137024) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.462437 (0.099794) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.428926 (0.137372) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.466523 (0.121543) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.428926 (0.137372) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.466523 (0.121543) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.428926 (0.137372) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.466523 (0.121543) with: {'criterion': 'gini', 'max_depth': 10, 'min_samples_lea
0.436725 (0.148907) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.442860 (0.151027) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.434687 (0.148688) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.440047 (0.149956) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.430498 (0.148713) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.438967 (0.151772) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.429986 (0.150148) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.448593 (0.127001) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.429986 (0.150148) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.448593 (0.127001) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.428878 (0.149947) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.437515 (0.140783) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.428920 (0.149002) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.433638 (0.141197) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.428920 (0.149002) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.433638 (0.141197) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.428920 (0.149002) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.433638 (0.141197) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.426971 (0.151309) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.453879 (0.126835) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.426971 (0.151309) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.453879 (0.126835) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.426971 (0.151309) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.453879 (0.126835) with: {'criterion': 'gini', 'max_depth': None, 'min_samples_l
0.630723 (0.032484) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.629476 (0.031619) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.630723 (0.032484) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.629476 (0.031619) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.630723 (0.032484) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.629476 (0.031619) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.630723 (0.032484) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.629476 (0.031619) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.630723 (0.032484) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.629476 (0.031619) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.630723 (0.032484) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.629476 (0.031619) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.630723 (0.032484) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.629476 (0.031619) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.630723 (0.032484) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.629476 (0.031619) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.630723 (0.032484) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.629476 (0.031619) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
```

```
0.630723 (0.032484) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.629476 (0.031619) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.630723 (0.032484) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.629476 (0.031619) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.630723 (0.032484) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.629476 (0.031619) with: {'criterion': 'entropy', 'max_depth': 2, 'min_samples_l
0.403500 (0.141906) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.410361 (0.140524) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.403500 (0.141906) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.410173 (0.140313) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.403500 (0.141906) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.410173 (0.140313) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.403503 (0.141910) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.410722 (0.140683) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.403503 (0.141910) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.410722 (0.140683) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.403503 (0.141910) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.411949 (0.142131) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.403506 (0.141914) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.411873 (0.142040) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.403506 (0.141914) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.411873 (0.142040) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.403506 (0.141914) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.411873 (0.142040) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.403551 (0.141978) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.385840 (0.133214) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.403551 (0.141978) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.385840 (0.133214) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.403551 (0.141978) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.385840 (0.133214) with: {'criterion': 'entropy', 'max_depth': 6, 'min_samples_l
0.430685 (0.148926) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.493341 (0.092379) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.430472 (0.148922) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.467470 (0.123668) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.430469 (0.148926) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.450406 (0.113159) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.430779 (0.148785) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.489641 (0.076507) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.430779 (0.148785) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.489641 (0.076507) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.430338 (0.149004) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.494660 (0.076116) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.430665 (0.149043) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.480580 (0.071731) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.430665 (0.149043) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.480580 (0.071731) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.430665 (0.149043) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.480580 (0.071731) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.426783 (0.150515) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.441894 (0.120822) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.426783 (0.150515) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.441894 (0.120822) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.426783 (0.150515) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.441894 (0.120822) with: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_
0.440666 (0.153072) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.439586 (0.147823) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.438430 (0.152897) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.435102 (0.145119) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
```

```
0.435929 (0.152350) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.438771 (0.149269) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.431333 (0.150156) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.431563 (0.133824) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.431333 (0.150156) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.431563 (0.133824) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.430503 (0.150673) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.431873 (0.138912) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.431282 (0.148927) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.442062 (0.120738) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.431282 (0.148927) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.442062 (0.120738) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.431282 (0.148927) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.442062 (0.120738) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.426934 (0.152269) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.481794 (0.100871) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.426934 (0.152269) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.481794 (0.100871) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
0.426934 (0.152269) with: {'criterion': 'entropy', 'max_depth': None, 'min_sample
```

RandomForest

```
from sklearn.ensemble import RandomForestClassifier
# define models and parameters
model = RandomForestClassifier(random_state=42,class_weight="balanced")
# define grid search
param_grid = {'n_estimators': [100, 150, 300], # sets the number of decision trees to
              'max_depth': [2,6,10,None], #Set the max depth of the tree
              'min_samples_split': [1,2,5,10], #The minimum number of samples needed b
              'min_samples_leaf': [1,3,5, 10] #The minimum number of samples needed to
             }
grid_search = GridSearchCV(estimator=model, param_grid=param_grid,  scoring='accuracy'
grid_result = grid_search.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print("%f (%f) with: %r" % (mean, stdev, param))
```

7.Evaluation and model selection

```
classifiers = [
LogisticRegression(random_state=42,class_weight='balanced',penalty = 'l2',C= 0.0001),
DecisionTreeClassifier(random_state=42,class_weight="balanced", criterion='gini',min_s
RandomForestClassifier(random_state=42,class_weight="balanced",n_estimators=,min_sampl
]

# Define a result table as a DataFrame
```

```
result_table = pd.DataFrame(columns=['classifiers', 'accuracy_score'])

# Train the models and record the results
for cls in classifiers:
    model = cls.fit(X_train, y_train)
    y_test_pred = model.predict(X_test)
    accuracy=metrics.accuracy_score(y_test, y_test_pred)
    result_table = result_table.append({'classifiers':cls.__class__.__name__,
                                        'accuracy_score':accuracy}, ignore_index=True)
result_table
```