```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

This dataset is designed for research and analysis of load balancing in distributed systems. It includes key features such as task size, CPU and memory demand, network latency, I/O operations, disk usage, number of connections, and priority level, along with a target variable for classification or optimization. Timestamp data is also provided for temporal analysis. It is suitable for machine learning, simulation studies, and performance optimization research.

Columns:
task_size: Size of the task (numeric).
cpu_demand: CPU demand of the task (numeric).
memory_demand: Memory demand of the task (numeric).
network_latency: Network latency associated with the task (numeric).
io_operations: Number of I/O operations (numeric).
disk_usage: Disk usage for the task (numeric).
num_connections: Number of active connections for the task (numeric).
priority_level: Priority level assigned to the task (numeric).
target: Target label indicating the outcome or category (binary).
timestamp: Timestamp when the task data was recorded.

```
###!pip install bayesian-optimization
```

```
###!pip install keras-tuner
```

```
###!pip uninstall tensorflow
####!pip install tensorflow==2.12.0
```

```
###!pip install keras==2.12.0
```

```
# Import packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from math import floor
from sklearn.metrics import make_scorer, accuracy_score
from bayes_opt import BayesianOptimization
from sklearn.model_selection import StratifiedKFold

import warnings
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns", None)
```

```
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
```

```
# Importing Pandas and NumPy
import pandas as pd, numpy as np
```

```
# Importing all datasets
LoadBalancerSystem = pd.read_csv("/content/Load Balancing Improved.csv")
LoadBalancerSystem.head(4)
```

	task_size	cpu_demand	memory_demand	network_latency	io_operations	disk_usage	num_connections	priority_level	target	timestamp
0	-0.152124	3.750160	-0.981182	0.251507	-0.471993	1.007026	0.313790	3.050953	1	2023-03-16 03:46:22
1	0.724624	-3.978920	2.022732	1.194530	-0.010304	-2.493867	-0.073875	-1.271258	0	2023-09-02 20:15:54
2	4.650228	1.145925	2.641659	-1.899635	1.187132	4.283652	0.572666	1.243801	1	2022-02-19 08:48:52
3	-0.138208	-0.189687	-0.820848	-3.060794	-1.982086	3.620598	-0.876702	0.776770	1	2023-12-22 11:58:26

Next steps: Generate code with LoadBalancerSystem | View recommended plots | New interactive sheet

```
from sklearn.model_selection import train_test_split
```

```
train, test = train_test_split(LoadBalancerSystem, test_size=0.2, random_state=1)
```

```
print(train.shape, test.shape)
```

```
(8542, 10) (2136, 10)
```

```
print("The columns in train data :", train.columns)
print("The columns in test data :", test.columns)
```

```
The columns in train data : Index(['task_size', 'cpu_demand', 'memory_demand', 'network_latency',
       'io_operations', 'disk_usage', 'num_connections', 'priority_level',
       'target', 'timestamp'],
      dtype='object')
The columns in test data : Index(['task_size', 'cpu_demand', 'memory_demand', 'network_latency',
       'io_operations', 'disk_usage', 'num_connections', 'priority_level',
       'target', 'timestamp'],
      dtype='object')
```

```
train.to_csv("train_load_balancer.csv")
```

```
test.to_csv("test_load_balancer.csv")
```

```
# Importing all datasets
train = pd.read_csv("/content/train_load_balancer.csv")
train.head(4)
```

	Unnamed: 0	task_size	cpu_demand	memory_demand	network_latency	io_operations	disk_usage	num_connections	priority_level	target	timestamp
0	4311	-2.884349	-0.304593	1.428882	0.641865	-1.265519	-2.042585	-2.151759	2.248828	0	2024-04-30 12:47:38
1	6133	0.163591	-0.054587	0.243658	0.631062	-2.349176	-0.106621	0.016042	2.954929	0	2024-09-15 11:42:34
2	2321	1.243694	3.099626	0.301481	0.439303	-3.356344	4.851570	-1.382724	2.085178	1	2022-10-20 08:26:25
3	1595	-1.675813	0.516609	-1.972764	1.926940	-1.554229	-2.546848	1.107627	1.629831	1	2023-10-08 10:31:30

Next steps: Generate code with `train` View recommended plots New interactive sheet

```
# Importing all datasets
test = pd.read_csv("/content/test_load_balancer.csv")
test.head(4)
```

	Unnamed: 0	task_size	cpu_demand	memory_demand	network_latency	io_operations	disk_usage	num_connections	priority_level	target	timestamp
0	6138	0.982596	2.560753	0.309671	0.484964	-2.768847	4.956329	0.780822	0.050808	1	2024-09-22 00:03:32
1	9271	-0.291851	1.488813	1.230143	1.814476	-2.828979	2.481521	-0.649220	0.193506	1	2022-04-04 12:13:28
2	4052	0.371047	-0.104333	2.951606	-0.270315	0.201554	1.474998	0.502292	-0.247073	1	2024-07-19 04:29:14
3	3068	-0.619218	-0.772362	0.013539	-1.840224	-1.292816	3.058381	2.911729	-1.782623	0	2022-08-18 00:11:56

Next steps: Generate code with test | View recommended plots | New interactive sheet

```
###! pip install klib
```

```
import klib
```

```
train = klib.data_cleaning(train)
test = klib.data_cleaning(test)
```

```
Shape of cleaned data: (8542, 11) - Remaining NAs: 0


    Dropped rows: 0
         of which 0 duplicates. (Rows (first 150 shown): [])

    Dropped columns: 0
         of which 0 single valued.     Columns: []
    Dropped missing values: 0
```

```
Reduced memory by at least: 0.37 MB (-51.39%)

Shape of cleaned data: (2136, 11) - Remaining NAs: 0


Dropped rows: 0
     of which 0 duplicates. (Rows (first 150 shown): [])

Dropped columns: 0
     of which 0 single valued.     Columns: []
Dropped missing values: 0
Reduced memory by at least: 0.09 MB (-50.0%)
```

```
train_cleaned = klib.clean_column_names(train)
test_cleaned = klib.clean_column_names(test)
```

```
train_cleaned = klib.convert_datatypes(train_cleaned)
test_cleaned = klib.convert_datatypes(test_cleaned)
```

```
train.columns
```

```
Index(['unnamed_0', 'task_size', 'cpu_demand', 'memory_demand',
       'network_latency', 'io_operations', 'disk_usage', 'num_connections',
       'priority_level', 'target', 'timestamp'],
      dtype='object')
```

```
train["timestamp"]  = pd.to_datetime(train["timestamp"])
```

```
train["day"] = train["timestamp"].dt.day
train["month"] = train["timestamp"].dt.month
train["year"] = train["timestamp"].dt.year
train["hour"] = train["timestamp"].dt.hour
```

```
train.columns
```

```
Index(['unnamed_0', 'task_size', 'cpu_demand', 'memory_demand',
       'network_latency', 'io_operations', 'disk_usage', 'num_connections',
       'priority_level', 'target', 'timestamp', 'day', 'month', 'year',
       'hour'],
      dtype='object')
```

```
#train.drop(columns="timestamp", inplace=True)
```

```
#train.drop(columns="unnamed_0", inplace=True)
```

```
test["timestamp"] = pd.to_datetime(test["timestamp"])
```

```
test["day"] = test["timestamp"].dt.day
test["month"] = test["timestamp"].dt.month
test["year"] = test["timestamp"].dt.year
test["hour"] = test["timestamp"].dt.hour
```

```
#test.drop(columns="timestamp", inplace=True)
```

```
#test.drop(columns="unnamed_0", inplace=True)
```

```
print("The DataTypes :", train.dtypes)
```

```
The DataTypes : task_size         float32
cpu_demand         float32
memory_demand      float32
network_latency    float32
io_operations      float32
disk_usage         float32
num_connections    float32
priority_level     float32
target                int8
day                  int32
month                int32
year                 int32
```

```
hour                int32
dtype: object
```

```
X_train = train.drop(columns="target")
Y_train = train["target"]
```

```
feature_names = X_train.columns
```

```
X_test = test.drop(columns="target")
Y_test = test["target"]
```

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

```
(8542, 12) (8542,) (2136, 12) (2136,)
```

```
from sklearn.preprocessing import StandardScaler
```

```
scaler = StandardScaler()
```

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

```
num_var = [feature for feature in train.columns if train[feature].dtypes != 'O']
discrete_var = [feature for feature in num_var if len(train[feature].unique()) <= 25]
cont_var = [feature for feature in num_var if feature not in discrete_var]
categ_var = [feature for feature in train.columns if feature not in num_var]
```

```
def find_var_type(var):


    if var in discrete_var:
        print("{} is a Numerical Variable, Discrete in nature".format(var))
    elif var in cont_var :
```

```
        print("{} is a Numerical Variable, Continuous in nature".format(var))
    else :
        print("{} is a Categorical Variable".format(var))
```

```
print("The continuous variables are :", cont_var)
print("The categorical variables are :", discrete_var)
```

```
The continuous variables are : ['task_size', 'cpu_demand', 'memory_demand', 'network_latency', 'io_operations', 'disk_usage', 'num_connections', 'priority_l
The categorical variables are : ['target', 'month', 'year', 'hour']
```

```
from sklearn.linear_model import Lasso

from sklearn.feature_selection import SelectFromModel
```

```
# Perform feature selection using a variance threshold
from sklearn.feature_selection import VarianceThreshold

sel = VarianceThreshold(threshold=(0.02))
sel.fit(train)

print("Feature selection", sel.get_support())
print("Selected features:", list(train.columns[sel.get_support()]))
print("Removed features:", list(train.columns[~sel.get_support()]))
```

```
Feature selection [ True  True  True  True  True  True  True  True  True  True  True  True
  True]
Selected features: ['task_size', 'cpu_demand', 'memory_demand', 'network_latency', 'io_operations', 'disk_usage', 'num_connections', 'priority_level', 'targ
Removed features: []
```

```
# Function to list features that are correlated
# Adds the first of the correlated pair only (not both)
def correlatedFeatures(dataset, threshold):
    correlated_columns = set()
    correlations = dataset.corr()
```

```
    for i in range(len(correlations)):
        for j in range(i):
            if abs(correlations.iloc[i,j]) > threshold:
                correlated_columns.add(correlations.columns[i])
    return correlated_columns
```

```
# Get a set of correlated features, based on threshold correlation of 0.85
cf = correlatedFeatures(train, 0.85)
cf
```

```
set()
```

```
X_train = pd.DataFrame(X_train)
X_test = pd.DataFrame(X_test)
Y_test = pd.DataFrame(Y_test)
Y_train = pd.DataFrame(Y_train)
```

```
print(X_train.shape)
print(train.shape)
print(train.columns)
```

```
(8542, 12)
(8542, 13)
Index(['task_size', 'cpu_demand', 'memory_demand', 'network_latency',
       'io_operations', 'disk_usage', 'num_connections', 'priority_level',
       'target', 'day', 'month', 'year', 'hour'],
      dtype='object')
```

```
X_train = X_train.rename(columns={
    0: 'task_size',
    1: 'cpu_demand',
    2: 'memory_demand',
    3: 'network_latency',
    4: 'io_operations',
    5: 'disk_usage',
    6: 'num_connections',
```

```
    7: 'priority_level',
    8: 'day',
    9: 'month',
    10: 'year',
    11: 'hour'
})
```

```
X_train.columns
```

```
Index(['task_size', 'cpu_demand', 'memory_demand', 'network_latency',
       'io_operations', 'disk_usage', 'num_connections', 'priority_level',
       'day', 'month', 'year', 'hour'],
      dtype='object')
```

```
X_test = X_test.rename(columns={
    0: 'task_size',
    1: 'cpu_demand',
    2: 'memory_demand',
    3: 'network_latency',
    4: 'io_operations',
    5: 'disk_usage',
    6: 'num_connections',
    7: 'priority_level',
    8: 'day',
    9: 'month',
    10: 'year',
    11: 'hour'
})
```

```
print(X_train.columns)
print(X_test.columns)
```

```
Index(['task_size', 'cpu_demand', 'memory_demand', 'network_latency',
       'io_operations', 'disk_usage', 'num_connections', 'priority_level',
       'day', 'month', 'year', 'hour'],
```

```
       dtype='object')
Index(['task_size', 'cpu_demand', 'memory_demand', 'network_latency',
       'io_operations', 'disk_usage', 'num_connections', 'priority_level',
       'day', 'month', 'year', 'hour'],
      dtype='object')
```

PIPELINE CREATION

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from xgboost import XGBClassifier
```

```
pipeline_lr=Pipeline([('scalar1',StandardScaler()),
                      ('pca1',PCA(n_components=2)),
                      ('lr_classifier',LogisticRegression())])
```

```
pipeline_dt=Pipeline([('scalar1',StandardScaler()),
                      ('pca1',PCA(n_components=2)),
                      ('dt_classifier',DecisionTreeClassifier())])
```

```
pipeline_rf=Pipeline([('scalar1',StandardScaler()),
                      ('pca1',PCA(n_components=2)),
                      ('rf_classifier',RandomForestClassifier())])
```

```
pipeline_gradient_boost=Pipeline([('scalar4',StandardScaler()),
                    ('pca4',PCA(n_components=2)),
                    ('gb_classifier',GradientBoostingClassifier())])
```

```
pipeline_xgboost=Pipeline([('scalar4',StandardScaler()),
                    ('pca4',PCA(n_components=2)),
                    ('gb_classifier',XGBClassifier())])
```

```
## LEts make the list of pipelines
pipelines = [pipeline_lr, pipeline_dt, pipeline_rf, pipeline_gradient_boost,pipeline_xgboost]
```

```
best_accuracy=0.0
best_classifier=0
best_pipeline=""
```

```
# Dictionary of pipelines and classifier types for ease of reference
pipe_dict = {0: 'Linear Regression', 1: 'Decision Tree', 2: 'RandomForest', 3: 'Gradient Boost', 4: 'Extreme Gradient Boost'}

# Fit the pipelines
for pipe in pipelines:
    pipe.fit(X_train, Y_train)
```

```
for i,model in enumerate(pipelines):
    print("{} Test Value: {}".format(pipe_dict[i],model.score(X_test, Y_test)))
```

```
Linear Regression Test Value: 0.7050561797752809
Decision Tree Test Value: 0.6999063670411985
RandomForest Test Value: 0.725187265917603
Gradient Boost Test Value: 0.7598314606741573
Extreme Gradient Boost Test Value: 0.7415730337078652
```

XGBClassifier

```
from xgboost import XGBClassifier
```

```
import time
```

```
from sklearn.metrics import make_scorer, accuracy_score
```

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

```
def xgb_cl_bo(min_child_weight, gamma, subsample, colsample_bytree, max_depth):
    params_xgb = {}
    params_xgb['min_child_weight'] = min_child_weight
    params_xgb['gamma'] = gamma
    params_xgb['subsample'] = subsample
    params_xgb['colsample_bytree'] = int(colsample_bytree)
    params_xgb['max_depth'] = int(max_depth)
    scores = cross_val_score(XGBClassifier(random_state=123, **params_xgb),
                             X_train, Y_train, scoring=accuracy, cv=5).mean()
    score = scores.mean()
    score = -score
    return score
# Run Bayesian Optimization
start = time.time()
params_xgb ={
    'min_child_weight':(1, 20),
    'gamma':(0.5, 10),
    'subsample':(0.6, 1.0),
    'colsample_bytree':(0.6, 1.0),
    'max_depth': (3, 35)
}
```

```
xgb_bo = BayesianOptimization(xgb_cl_bo, params_xgb)
xgb_bo.maximize(init_points=20, n_iter=4)
print('It takes %s minutes' % ((time.time() - start)/60))
```

iter	target	colsam...	gamma	max_depth	min_ch...	subsample
1	0.8616	0.6436	6.793	6.231	18.43	0.7072
2	0.8653	0.9614	2.461	7.225	3.727	0.8514
3	0.8607	0.9759	1.768	25.47	12.21	0.7787
4	0.8552	0.9733	1.382	11.94	10.04	0.6727
5	0.8624	0.9525	2.397	16.68	18.7	0.7433
6	0.8627	0.8585	6.536	26.2	9.321	0.7565
7	0.8588	0.94	9.72	12.31	4.049	0.7413
8	0.8646	0.6457	5.073	27.13	18.97	0.9906
9	0.8623	0.877	1.648	6.607	9.763	0.9973
10	0.8617	0.746	2.625	10.21	16.53	0.6377
11	0.8601	0.748	9.678	9.417	8.343	0.6953
12	0.8642	0.9423	4.18	33.06	3.178	0.6781
13	0.8628	0.909	5.724	17.9	19.68	0.6034
14	0.862	0.9448	8.322	28.61	12.94	0.7488
15	0.8644	0.8144	3.21	5.25	3.047	0.9055
16	0.863	0.6247	3.631	13.4	7.148	0.7534
17	0.8647	0.6919	5.665	10.27	5.053	0.9683
18	0.8641	0.6603	5.353	4.557	12.72	0.7759
19	0.8631	0.6638	7.535	17.84	16.91	0.8554
20	0.8644	0.9903	2.259	3.986	5.922	0.8281
21	0.8651	0.9848	4.881	6.72	5.166	0.68
22	0.8657	0.6142	4.013	9.225	2.063	0.8222
23	0.8642	0.8175	3.028	12.91	2.598	0.9816
24	0.8487	0.8775	0.7193	9.798	1.096	0.8476

It takes 0.28228790760040284 minutes

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

```
{'colsample_bytree': 1,
 'gamma': 4,
 'max_depth': 9,
```

```
      'min_child_weight': 2,
      'subsample': 1}
```

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

```
xgb_hyp.fit(X_train, Y_train)
```

```
XGBClassifier

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None, colsample_bytree=1,
              device=None, early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=4, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=9,
              max_leaves=None, min_child_weight=2, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, random_state=123, ...)
```

```
# Predict the validation data
pred_xgb = xgb_hyp.predict(X_test)

# Compute the accuracy
print('Accuracy: ' + str(accuracy_score(Y_test, pred_xgb)))
```

```
Accuracy: 0.9480337078651685
```

```
pred_xgb = pd.DataFrame(pred_xgb)
```

```
pred_xgb.rename(columns = {0 : "Predict"}, inplace=True)
```

```
pred_xgb.value_counts()
```

Predict	count
0	1077
1	1059

dtype: int64

```
import pickle

# Save model to a pickle file
with open('xgb_hyp.pkl', 'wb') as file:
    pickle.dump(xgb_hyp, file)
```

```
import joblib

# Save model to a joblib file
joblib.dump(xgb_hyp, 'xgb_hyp.joblib')
```

```
['xgb_hyp.joblib']
```

Random Forest Classifier - Using Bayesian Optimization

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

```
def rfc_cl_bo(n_estimators, max_features, max_depth, min_samples_split, min_samples_leaf):
    params_rfc = {}
    params_rfc['n_estimators'] = int(n_estimators)
    params_rfc['max_features'] = max_features
    params_rfc['max_depth'] = round(max_depth)
    params_rfc['min_samples_split'] = int(min_samples_split)
```

```
    params_rfc['min_samples_leaf'] = int(min_samples_leaf)
    scores = cross_val_score(RandomForestClassifier(random_state=123, **params_rfc),
                             X_train, Y_train, scoring=accuracy, cv=5).mean()
    score = scores.mean()
    score = -score
    return score
# Run Bayesian Optimization
start = time.time()
params_rfc ={
    'n_estimators':(80, 300),
    'max_features':(0.8, 1),
    'max_depth':(1, 250),
    'min_samples_split':(2, 20),
    'min_samples_leaf' :(1, 40)
}
```

```
rfc_bo = BayesianOptimization(rfc_cl_bo, params_rfc)
rfc_bo.maximize(init_points=30, n_iter=4)
print('It takes %s minutes' % ((time.time() - start)/60))
```

iter	target	max_depth	max_fe...	min_sa...	min_sa...	n_esti...
1	0.9336	47.32	0.8648	12.14	10.75	181.6
2	0.9158	238.0	0.9706	35.4	6.929	125.6
3	0.9185	229.7	0.9272	31.06	17.11	161.1
4	0.9279	7.608	0.867	12.62	5.561	263.0
5	0.9221	181.5	0.984	25.38	18.27	291.7
6	0.9312	105.2	0.9061	15.28	11.88	154.1
7	0.9197	111.6	0.8852	32.81	10.74	176.5
8	0.9125	201.7	0.9928	38.92	11.05	263.1
9	0.9168	155.9	0.8583	36.34	17.37	298.3
10	0.9118	218.5	0.9661	39.28	7.161	276.3
11	0.9319	56.66	0.9669	13.3	17.05	157.2
12	0.9301	91.39	0.8456	18.48	14.33	205.3
13	0.9184	209.1	0.8053	37.88	9.939	249.8
14	0.9196	10.82	0.9576	29.69	12.71	186.2
15	0.9302	193.2	0.8568	18.85	7.672	196.1
16	0.9291	225.2	0.9028	20.67	4.046	137.8
17	0.9443	240.5	0.8473	1.402	5.739	282.3
18	0.935	220.6	0.8627	12.52	9.006	85.75

```
| 19        | 0.9261   | 249.7    | 0.8099   | 24.32    | 6.872    | 237.5    |
| 20        | 0.9294   | 201.4    | 0.8804   | 19.96    | 19.01    | 241.7    |
| 21        | 0.9144   | 211.4    | 0.8762   | 39.59    | 5.695    | 120.5    |
| 22        | 0.9124   | 207.4    | 0.9702   | 38.01    | 18.91    | 168.0    |
| 23        | 0.9273   | 152.9    | 0.922    | 18.27    | 7.451    | 256.3    |
| 24        | 0.9196   | 206.3    | 0.974    | 29.17    | 8.222    | 113.0    |
| 25        | 0.9404   | 198.1    | 0.8468   | 5.792    | 14.67    | 117.2    |
| 26        | 0.9373   | 69.72    | 0.9694   | 6.409    | 17.96    | 191.8    |
| 27        | 0.9211   | 152.6    | 0.9954   | 27.63    | 2.35     | 214.9    |
| 28        | 0.9227   | 117.2    | 0.9361   | 24.42    | 7.03     | 113.8    |
| 29        | 0.9212   | 159.6    | 0.8374   | 29.94    | 18.43    | 246.4    |
| 30        | 0.925    | 208.2    | 0.9082   | 24.91    | 4.785    | 114.0    |
| 31        | 0.9195   | 177.3    | 0.9668   | 30.28    | 16.33    | 217.2    |
| 32        | 0.941    | 246.4    | 0.8438   | 1.717    | 19.35    | 290.7    |
| 33        | 0.9422   | 71.58    | 0.8067   | 4.429    | 16.18    | 192.9    |
| 34        | 0.9408   | 76.88    | 0.8764   | 5.256    | 2.609    | 195.7    |
=========================================================================================
It takes 32.092298988501234 minutes
```

```
params_rfc = rfc_bo.max['params']
params_rfc['max_depth'] = round(params_rfc['max_depth'])
params_rfc['min_samples_leaf'] = round(params_rfc['min_samples_leaf'])
params_rfc['min_samples_split'] = round(params_rfc['min_samples_split'])
params_rfc['n_estimators'] = round(params_rfc['n_estimators'])
params_rfc
```

```
{'max_depth': 240,
 'max_features': np.float64(0.8473019458277843),
 'min_samples_leaf': 1,
 'min_samples_split': 6,
 'n_estimators': 282}
```

```
rfc_hyp =  RandomForestClassifier(**params_rfc, random_state=123)
```

```
rfc_hyp.fit(X_train, Y_train)
```

```
RandomForestClassifier(max_depth=240,
                       max_features=np.float64(0.8473019458277843),
                       min_samples_split=6, n_estimators=282, random_state=123)
```

```
pred_rfc = rfc_hyp.predict(X_test)
```

```
pred_rfc = pd.DataFrame(pred_rfc)
```

```
pred_rfc.rename(columns = {0:"Label"}, inplace=True)
```

```
pred_rfc.value_counts()
```

```
       count
Label
0       1085
1       1051

dtype: int64
```

```
import pickle

# Save model to a pickle file
with open('rfc_hyp.pkl', 'wb') as file:
    pickle.dump(rfc_hyp, file)
```

```
import joblib

# Save model to a joblib file
joblib.dump(rfc_hyp, 'rfc_hyp.joblib')
```

```
['rfc_hyp.joblib']
```

KNN Classifier - Using Bayesian Optimization

```
from sklearn.neighbors import KNeighborsClassifier
```

```
# Hyperparameter-tuning: Bayesian Optimization, bayes_opt
def knn_cl_bo(n_neighbors, weights, p):
    params_knn = {}
    weightsL = ['uniform', 'distance']

    params_knn['n_neighbors'] = round(n_neighbors)
    params_knn['weights'] = weightsL[round(weights)]
    params_knn['p'] = round(p)

    score = cross_val_score(KNeighborsClassifier(**params_knn),
                            X_train, Y_train, cv=9, scoring=accuracy).mean()
    return score
```

```
# Set hyperparameters spaces
params_knn ={
    'n_neighbors':(3, 10),
    'weights':(0, 1),
    'p':(1, 2)}

# Run Bayesian Optimization
knn_bo = BayesianOptimization(knn_cl_bo, params_knn, random_state=111)
knn_bo.maximize(init_points=4, n_iter=35)
```

```
|   iter    |  target   | n_neig... |     p     |  weights  |
-------------------------------------------------------------
| 1         | -0.9254   | 7.285     | 1.169     | 0.4361    |
| 2         | -0.9289   | 8.385     | 1.295     | 0.1492    |
| 3         | -0.9165   | 3.157     | 1.42      | 0.2387    |
```

```
| 4         | -0.9213   | 5.364     | 1.991     | 0.2377    |
| 5         | -0.9163   | 3.102     | 1.057     | 0.9962    |
| 6         | -0.9184   | 3.036     | 1.85      | 0.9653    |
| 7         | -0.9216   | 3.872     | 1.003     | 0.08092   |
| 8         | -0.9185   | 3.039     | 1.978     | 0.08066   |
| 9         | -0.9253   | 9.987     | 1.842     | 0.9445    |
| 10        | -0.9165   | 3.014     | 1.027     | 0.05375   |
| 11        | -0.9163   | 3.001     | 1.047     | 0.6408    |
| 12        | -0.9203   | 4.253     | 1.977     | 0.9818    |
| 13        | -0.9231   | 5.442     | 1.016     | 0.9898    |
| 14        | -0.9183   | 3.628     | 1.039     | 0.9841    |
| 15        | -0.9165   | 3.013     | 1.445     | 0.03837   |
| 16        | -0.9163   | 3.001     | 1.295     | 0.6286    |
| 17        | -0.9165   | 3.01      | 1.278     | 0.4024    |
| 18        | -0.9236   | 6.135     | 1.996     | 0.9949    |
| 19        | -0.9203   | 3.829     | 1.987     | 0.01178   |
| 20        | -0.9163   | 3.255     | 1.332     | 0.8232    |
| 21        | -0.9163   | 3.034     | 1.29      | 0.9694    |
| 22        | -0.9163   | 3.161     | 1.158     | 0.7382    |
| 23        | -0.9163   | 3.018     | 1.043     | 0.9541    |
| 24        | -0.9163   | 3.09      | 1.202     | 0.8531    |
| 25        | -0.9184   | 3.292     | 1.512     | 0.5865    |
| 26        | -0.9298   | 9.999     | 1.137     | 0.02749   |
| 27        | -0.9268   | 5.845     | 1.002     | 0.01513   |
| 28        | -0.9213   | 5.115     | 1.984     | 0.9759    |
| 29        | -0.9165   | 3.276     | 1.249     | 0.004045  |
| 30        | -0.9239   | 7.826     | 1.994     | 0.9789    |
| 31        | -0.9219   | 6.783     | 1.982     | 0.00713   |
| 32        | -0.9163   | 3.489     | 1.401     | 0.98      |
| 33        | -0.9165   | 3.263     | 1.031     | 0.2814    |
| 34        | -0.9163   | 3.283     | 1.473     | 0.9965    |
| 35        | -0.9163   | 3.273     | 1.268     | 0.9982    |
| 36        | -0.9213   | 4.628     | 1.985     | 0.03862   |
| 37        | -0.9185   | 3.222     | 1.557     | 0.01112   |
| 38        | -0.9203   | 3.744     | 1.987     | 0.996     |
| 39        | -0.9231   | 4.515     | 1.074     | 0.9998    |
=================================================================
```

```
# Best hyperparameters
params_knn = knn_bo.max['params']
weightsL = ['uniform', 'distance']
```

```
params_knn['n_neighbors'] = round(params_knn['n_neighbors'])
params_knn['weights'] = weightsL[round(params_knn['weights'])]
params_knn['p'] = round(params_knn['p'])
params_knn
```

```
{'n_neighbors': 3, 'p': 1, 'weights': 'distance'}
```

```
# Fit the training data
knn_hyp = KNeighborsClassifier(**params_knn)
knn_hyp.fit(X_train, Y_train)

# Predict the validation data
pred_knn = knn_hyp.predict(X_test)

# Compute the accuracy
print('Accuracy: ' + str(accuracy_score(Y_test, pred_knn)))
```

```
Accuracy: 0.9241573033707865
```

```
import pickle

# Save model to a pickle file
with open('knn_hyp.pkl', 'wb') as file:
    pickle.dump(knn_hyp, file)
```

```
import joblib

# Save model to a joblib file
```