```
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 keras.models import Sequential
from keras.layers import Dense, BatchNormalization, Dropout
from tensorflow.keras.optimizers import Adam, SGD, RMSprop, Adadelta, Adagrad, Adamax, Nadam, Ftrl
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.wrappers.scikit_learn import KerasClassifier
from math import floor
from sklearn.metrics import make_scorer, accuracy_score
from bayes_opt import BayesianOptimization
from sklearn.model_selection import StratifiedKFold
from keras.layers import LeakyReLU
LeakyReLU = LeakyReLU(alpha=0.1)
import warnings
warnings.filterwarnings('ignore')
pd.set_option("display.max_columns", None)
```

```
# 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

```
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
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	2022-10-20

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

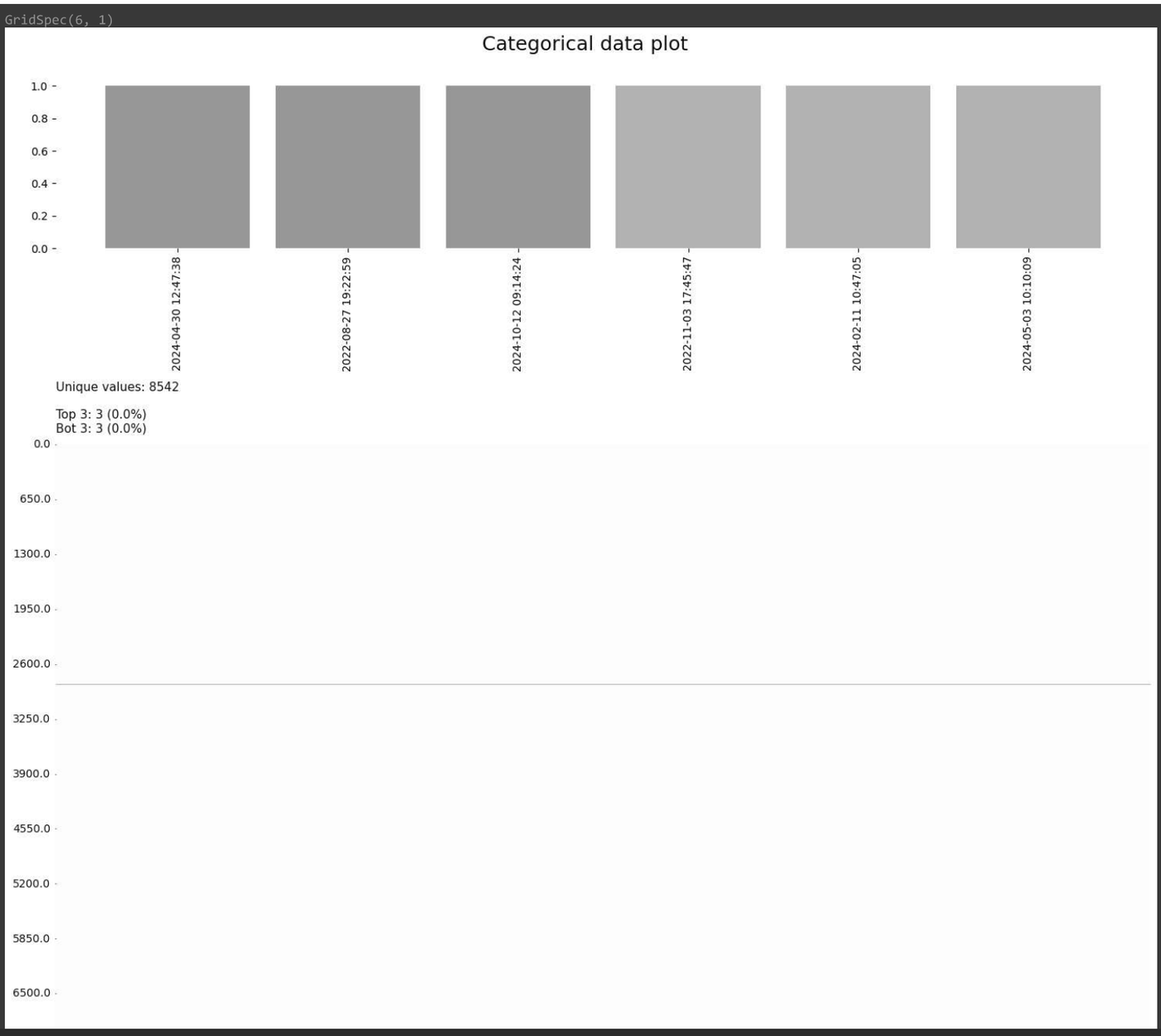
	Unnamed: 0	task_size	cpu_demand	memory_demand	network_latency	io_operations	disk_usage	num_connections	priority_level	target	timestamp
0	6138	0.982596	2.560753	0.309671	0.484964	-2.768847	4.956329	0.780822	0.050808	1	2024-09-22 00:03:32
1	9271	-0.291851	1.488813	1.230143	1.814476	-2.828979	2.481521	-0.649220	0.193506	1	2022-04-04 12:13:28

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

```
import klib
```

```
klib.cat_plot(train)
```

```
GridSpec(6, 1)
```

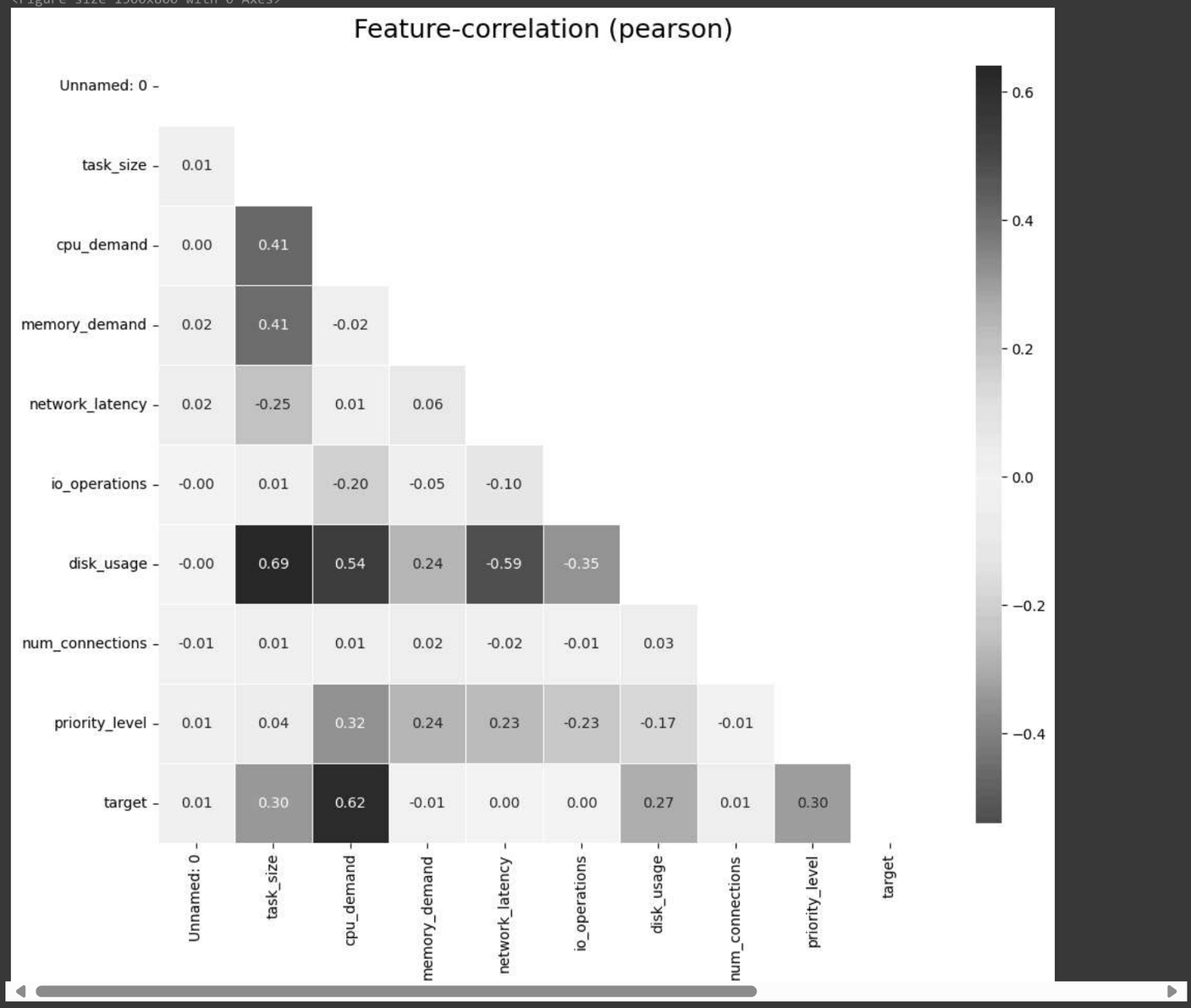



```
klib.corr_mat(train)
```

	Unnamed: 0	task_size	cpu_demand	memory_demand	network_latency	io_operations	disk_usage	num_connections	priority_level	target
Unnamed: 0	1.00	0.01	0.00	0.02	0.02	-0.00	-0.00	-0.01	0.01	0.01
task_size	0.01	1.00	0.41	0.41	-0.25	0.01	0.69	0.01	0.04	0.30
cpu_demand	0.00	0.41	1.00	-0.02	0.01	-0.20	0.54	0.01	0.32	0.62
memory_demand	0.02	0.41	-0.02	1.00	0.06	-0.05	0.24	0.02	0.24	-0.01
network_latency	0.02	-0.25	0.01	0.06	1.00	-0.10	-0.59	-0.02	0.23	0.00
io_operations	-0.00	0.01	-0.20	-0.05	-0.10	1.00	-0.35	-0.01	-0.23	0.00
disk_usage	-0.00	0.69	0.54	0.24	-0.59	-0.35	1.00	0.03	-0.17	0.27
num_connections	-0.01	0.01	0.01	0.02	-0.02	-0.01	0.03	1.00	-0.01	0.01
priority_level	0.01	0.04	0.32	0.24	0.23	-0.23	-0.17	-0.01	1.00	0.30
target	0.01	0.30	0.62	-0.01	0.00	0.00	0.27	0.01	0.30	1.00

```
plt.figure(figsize=[15,8])
klib.corr_plot(train)
```

```
<Axes: title={'center': 'Feature-correlation (pearson)'}>
<Figure size 1500x800 with 0 Axes>
```




```
klib.missingval_plot(train)
```

```
No missing values found in the dataset.
```

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

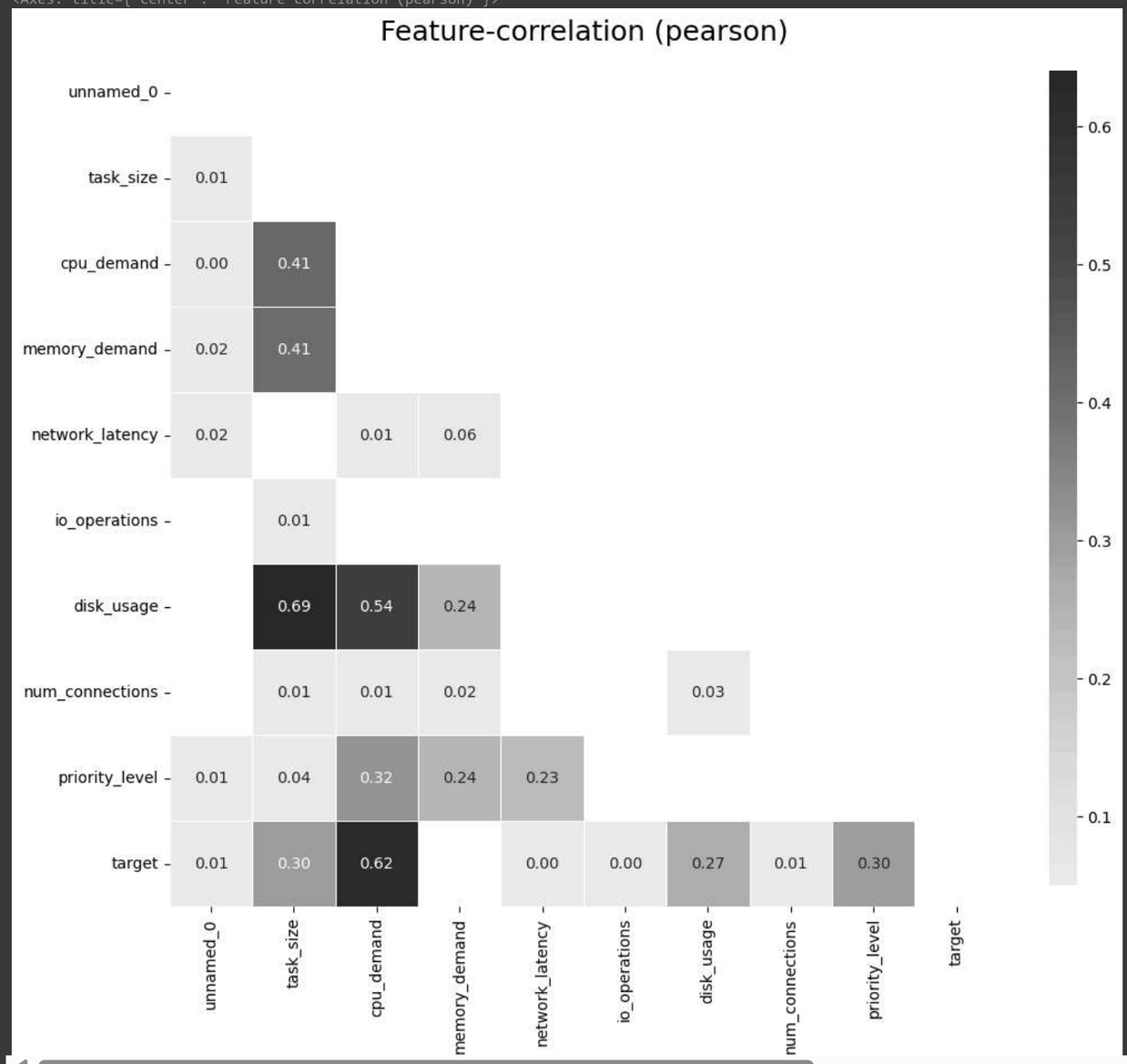
```
Shape of cleaned data: (8542, 11) - Remaining NAs: 0


    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)
```

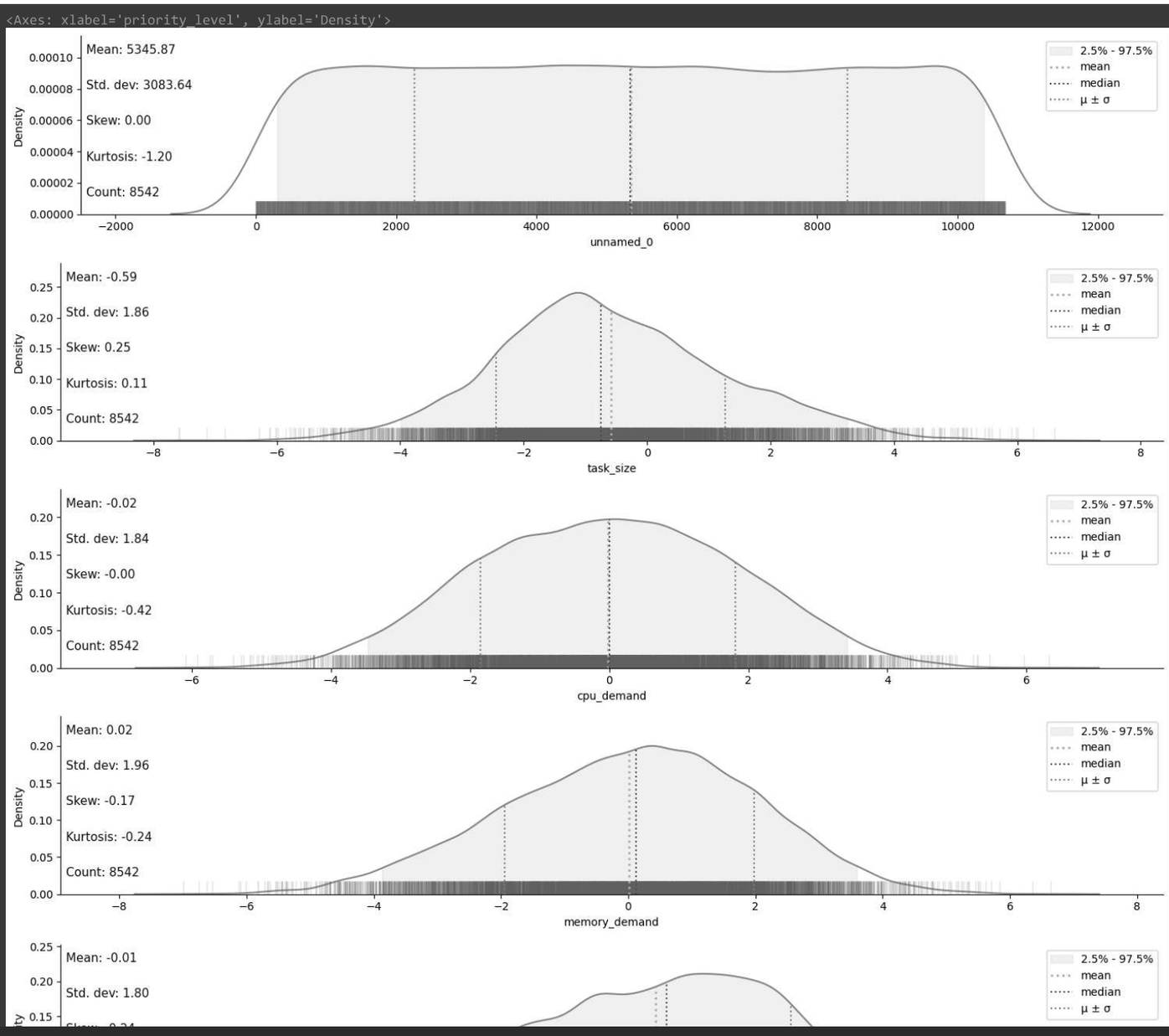
```
train_cleaned = klib.convert_datatypes(train_cleaned)
test_cleaned = klib.convert_datatypes(test_cleaned)
```

```
klib.corr_plot(train_cleaned, split='pos')
```

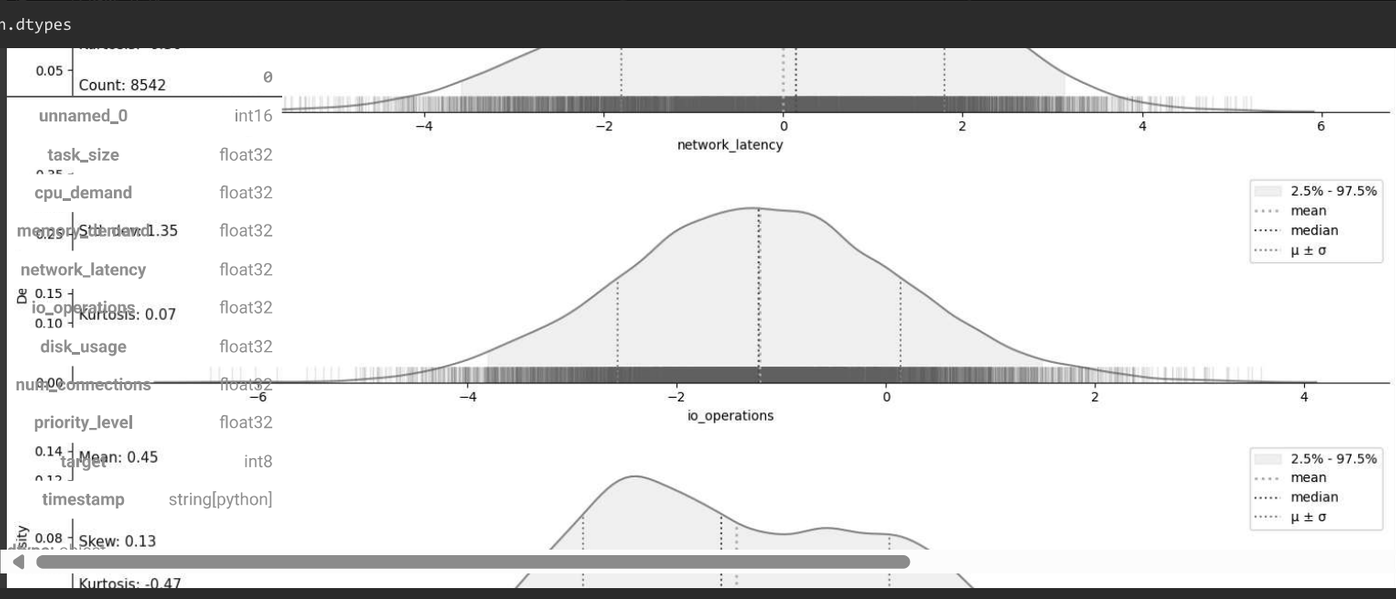
```
Displaying positive correlations. Specify a positive "threshold" to limit the results further.
<Axes: title={'center': 'Feature-correlation (pearson)'}>
```




```
klib.dist_plot(train_cleaned)
```

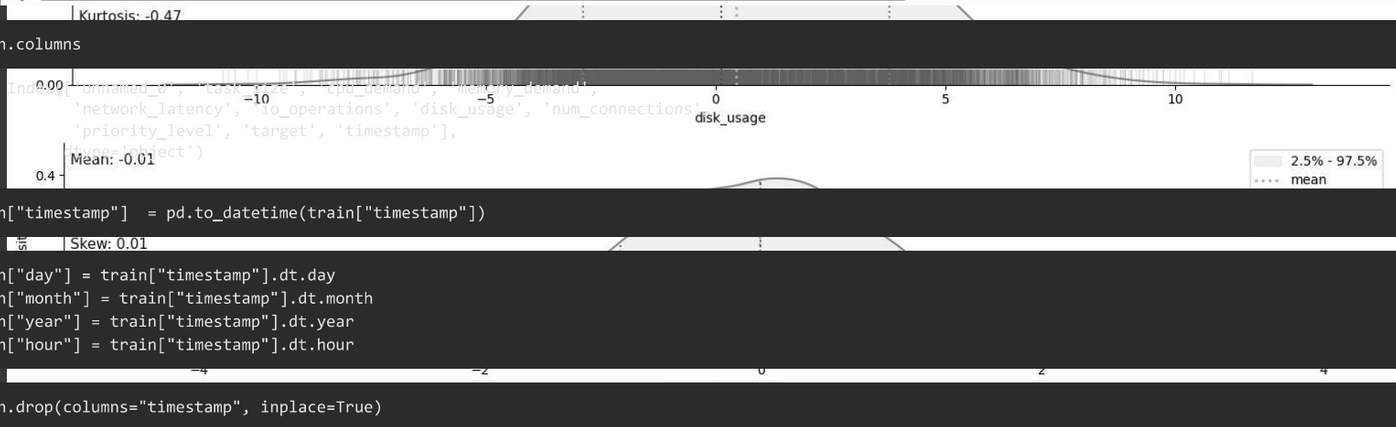



```
train.dtypes
```



unnamed_0	int16
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
timestamp	string[python]

```
train.columns
```




```
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.drop(columns="timestamp", 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)
```

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

```
The DataTypes : unnamed_0           int16
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, 13) (8542,) (2136, 13) (2136,)
```

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

```
scaler = StandardScaler()
```

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

```
# Make scorer accuracy
score_acc = make_scorer(accuracy_score)
```

```
from keras.wrappers.scikit_learn import KerasClassifier
from tensorflow.keras.optimizers import RMSprop

from tensorflow.keras import Sequential # import Sequential from tensorflow.keras
from tensorflow.keras.layers import Dense # import Dense from tensorflow.keras.layers
from numpy.random import seed # seed helps you to fix the randomness in the neural network.
import tensorflow
```

```
from sklearn.model_selection import KFold
```

```
from tensorflow.keras.losses import categorical_crossentropy
```

```
from numpy.random import seed # seed helps you to fix the randomness in the neural network.
```

```
# Import packages
# Basic packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import pickle
from math import floor
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import MinMaxScaler

# Evaluation and bayesian optimization
from sklearn.metrics import make_scorer, mean_absolute_error
from sklearn.metrics import mean_squared_error as MSE
from hyperopt import hp, fmin, tpe
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from bayes_opt import BayesianOptimization

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

```
# Make scorer: MSE
mse = make_scorer(MSE, greater_is_better=False)
```

```
Y_train.value_counts()
```

target	count
1	4295
0	4247

dtype: int64

The above dataset is a balanced dataset

FEATURE IMPORTANCE

```
from sklearn.ensemble import ExtraTreesClassifier
```

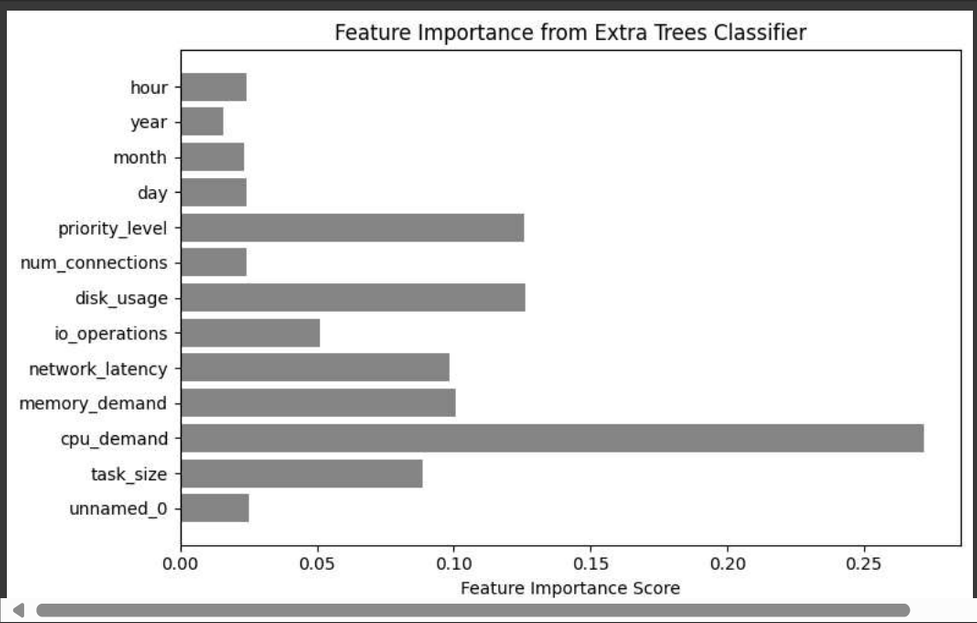
```
# Train Extra Trees Classifier
model = ExtraTreesClassifier(n_estimators=100, random_state=42)
model.fit(X_train, Y_train)
```

```
ExtraTreesClassifier
ExtraTreesClassifier(random_state=42)
```

```
train.columns
```

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

```
importances = model.feature_importances_

# Plot feature importances
plt.figure(figsize=(8, 5))
plt.barh(feature_names, importances, color='mediumseagreen')
plt.xlabel('Feature Importance Score')
plt.title('Feature Importance from Extra Trees Classifier')
plt.tight_layout()
plt.show()
```




```
# Make scorer: MSE
mse = make_scorer(MSE, greater_is_better=False)
```

```
Y_train.value_counts()
```

target	count
1	4295
0	4247

dtype: int64

```
X_train.shape
```

```
(8542, 13)
```

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

```
import time
```

```
from sklearn.ensemble import GradientBoostingClassifier
```

```
def gbm_cl_bo(max_depth, max_features, learning_rate, n_estimators, subsample):
    params_gbm = {}
    params_gbm['max_depth'] = round(max_depth)
    params_gbm['max_features'] = max_features
    params_gbm['learning_rate'] = learning_rate
    params_gbm['n_estimators'] = round(n_estimators)
    params_gbm['subsample'] = subsample
    scores = cross_val_score(GradientBoostingClassifier(random_state=123, **params_gbm),
                             X_train, Y_train, scoring=accuracy, cv=5).mean()
    score = scores.mean()
    return score
# Run Bayesian Optimization
start = time.time()
params_gbm ={
    'max_depth':(3, 10),
    'max_features':(0.8, 1),
    'learning_rate':(0.01, 1),
    'n_estimators':(80, 150),
    'subsample': (0.8, 1)
}
```

```
gbm_bo = BayesianOptimization(gbm_cl_bo, params_gbm, random_state=111)
gbm_bo.maximize(init_points=10, n_iter=4)
print('It takes %s minutes' % ((time.time() - start)/60))
```

```
|   iter    |  target   | learni... | max_depth | max_fe... | n_esti... | subsample |
-------------------------------------------------------------------------------------
| 1         | -0.9301   | 0.616     | 4.183     | 0.8872    | 133.8     | 0.8591    |
| 2         | -0.9315   | 0.1577    | 3.157     | 0.884     | 96.71     | 0.8675    |
| 3         | -0.9225   | 0.9908    | 4.664     | 0.8162    | 126.9     | 0.9242    |
| 4         | -0.9424   | 0.2815    | 6.264     | 0.8237    | 85.18     | 0.9802    |
| 5         | -0.9451   | 0.796     | 8.884     | 0.963     | 149.4     | 0.9155    |
| 6         | -0.9195   | 0.8156    | 5.949     | 0.8055    | 111.8     | 0.8211    |
| 7         | -0.9437   | 0.819     | 7.884     | 0.9131    | 99.2      | 0.9997    |
| 8         | -0.947    | 0.1467    | 7.308     | 0.897     | 108.4     | 0.9456    |
| 9         | -0.9439   | 0.3296    | 5.804     | 0.8638    | 146.3     | 0.9837    |
| 10        | -0.9184   | 0.8157    | 3.239     | 0.9887    | 146.5     | 0.9613    |
| 11        | -0.9446   | 0.02841   | 9.546     | 0.9055    | 140.5     | 0.8805    |
| 12        | -0.9287   | 0.05919   | 4.285     | 0.8093    | 92.7      | 0.9528    |
| 13        | -0.9109   | 0.9771    | 4.779     | 0.9072    | 126.8     | 0.9214    |
| 14        | -0.9358   | 0.721     | 5.683     | 0.8289    | 128.5     | 0.9406    |
=====================================================================================
It takes 8.78553285598755 minutes
```

```
params_gbm = gbm_bo.max['params']
params_gbm['max_depth'] = round(params_gbm['max_depth'])
params_gbm['n_estimators'] = round(params_gbm['n_estimators'])
params_gbm
```

```
{'learning_rate': 0.9770686084241488,
 'max_depth': 5,
 'max_features': 0.9071840538719766,
 'n_estimators': 127,
 'subsample': 0.9214345438923253}
```

```
gbc_hyp =  GradientBoostingClassifier(**params_gbm, random_state=123)
```

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

```
GradientBoostingClassifier(learning_rate=0.9770686084241488, max_depth=5,
                           max_features=0.9071840538719766, n_estimators=127,
                           random_state=123, subsample=0.9214345438923253)
```

```
pred_gbc = gbc_hyp.predict(X_test)
```

```
pred_gbc = pd.DataFrame(pred_gbc)
```

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

```
pred_gbc.value_counts()
```

```
       count
Label
0       1073
1       1063

dtype: int64
```

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

```
from sklearn.metrics import classification_report
```

```
print(acc_score(pred_gbc,Y_test))
```

```
0.9349250936329588
```

```
print(classification_report(pred_gbc,Y_test))
```

```
              precision    recall  f1-score   support

           0       0.93      0.94      0.94      1073
           1       0.94      0.93      0.93      1063
```

```
          accuracy                           0.93      2136
         macro avg       0.93      0.93      0.93      2136
      weighted avg       0.93      0.93      0.93      2136
```

KNN Classifier

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