```
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).
```

```
#Load the libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import nltk
from sklearn.metrics import accuracy_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelBinarizer
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize,sent_tokenize
from bs4 import BeautifulSoup
import re,string,unicodedata
from nltk.tokenize.toktok import ToktokTokenizer
from nltk.stem import LancasterStemmer,WordNetLemmatizer
from sklearn.linear_model import LogisticRegression,SGDClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score
import warnings
warnings.filterwarnings('ignore')
from imblearn import under_sampling, over_sampling
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE
from sklearn import linear_model
import numpy as np
from numpy import sign
import sklearn.datasets
import sklearn.ensemble
import sklearn.model_selection
import sklearn.svm
import optuna
# !pip install optuna
#importing all the plot functions
from optuna.visualization import plot_edf
from optuna.visualization import plot_optimization_history
from optuna.visualization import plot_parallel_coordinate
from optuna.visualization import plot_param_importances
from optuna.visualization import plot_slice
```

```
def remove_nan_blank(df):
    # Remove rows with NaN values
    df = df.dropna()
    # Replace blank values with NaN
    df = df.replace('', np.nan)
    # Remove rows with NaN values
    df = df.dropna()
    return df
```

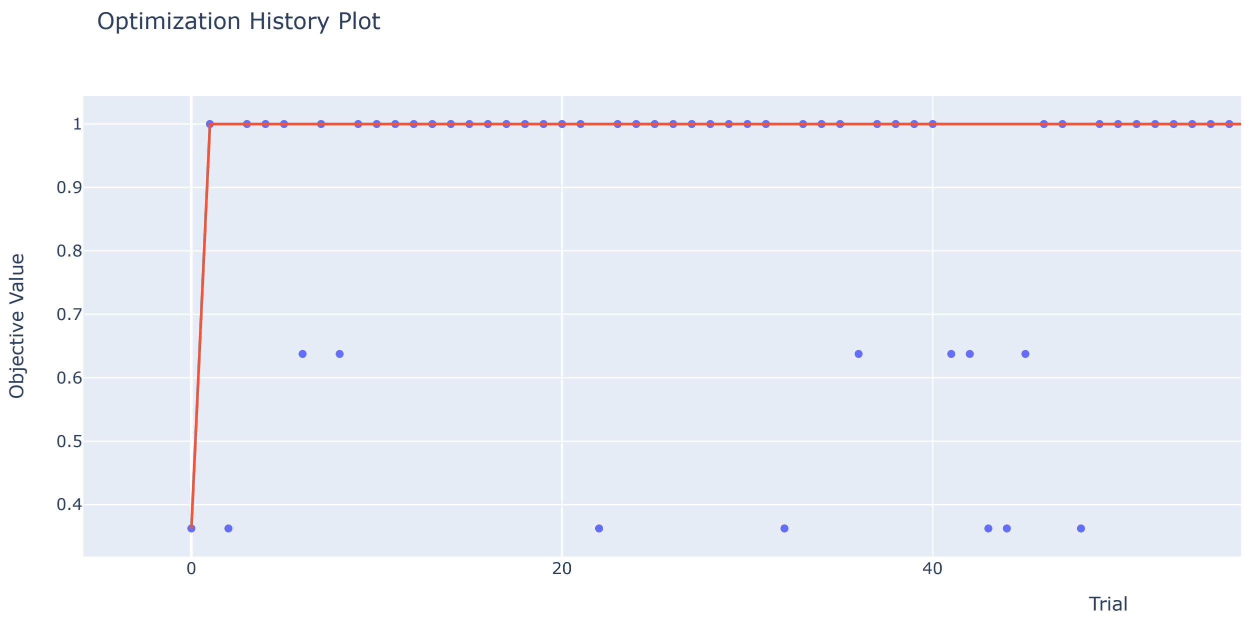
```
from sklearn.model_selection import train_test_split
path = "/content/drive/MyDrive/Semester_2/ML Lab/Exercise_10/smart_grid_stability_augmented.csv"
# Read the creditcard.csv file
data = pd.read_csv(path)
data = remove_nan_blank(data)
```

```
def getData(path):
  data = pd.read_csv(path)
  # binary encoding the labels
  data[data['stabf'] == 'stable'] = 1
  data[data['stabf'] == 'unstable'] = 0
  data['stabf'] = data['stabf'].astype(dtype='int64')
  # Split the data into features and target
  X = data.drop("stabf", axis=1)
  y = data["stabf"]

  # # Split the data into training and testing sets
  x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
  return x_train, y_train, x_test, y_test
```

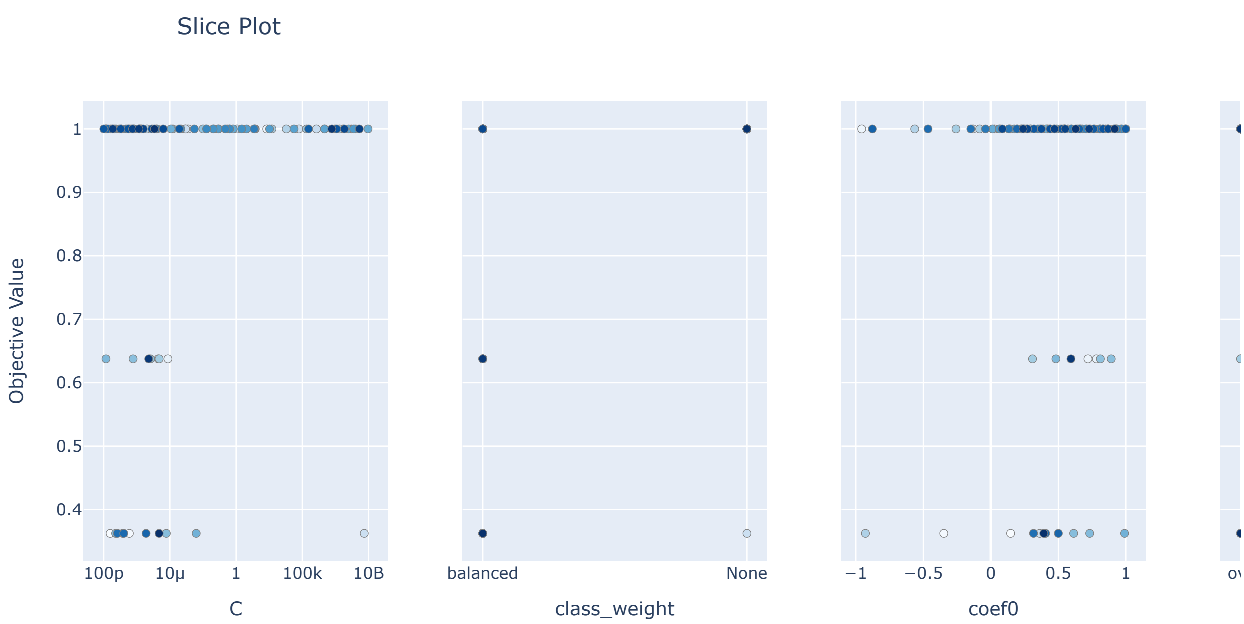
```
"""
   This code defines an objective function which is used to optimize the
   parameters of an SVM classifier using Optuna. The objective function uses
   Optunas trial object to suggest different values for different parameters of
   the SVM classifier. The parameters being optimized include the regularization
   parameter C, the kernel type, the degree of the polynomial kernel, the value of
   coef0, whether or not to use shrinking heuristics, the tolerance for stopping
   criteria, the maximum number of iterations, the shape of the decision function,
   the class weight, and the random state. The objective function uses cross-validation
   to evaluate the performance of the classifier with the suggested parameter values
   and returns the mean accuracy.
   """
def objective(trial, x, y):
    C = trial.suggest_float("C", 1e-10, 1e10, log=True)
    kernel = trial.suggest_categorical("kernel", ["linear", "poly", "rbf", "sigmoid"])
    degree = trial.suggest_int("degree", 1, 10)
    coef0 = trial.suggest_float("coef0", -1, 1)
    shrinking = trial.suggest_categorical("shrinking", [True, False])
    tol = trial.suggest_float("tol", 1e-5, 1e-1)
    max_iter = trial.suggest_int("max_iter", -1, 1000)
    decision_function_shape = trial.suggest_categorical("decision_function_shape", ["ovr", "ovo"])
    class_weight = trial.suggest_categorical("class_weight", ["balanced", None])
    random_state = trial.suggest_int("random_state", 0, 100)
    classifier_obj = sklearn.svm.SVC(C=C, kernel=kernel, degree=degree, coef0=coef0,
                                     shrinking=shrinking, tol=tol, max_iter=max_iter,
                                     decision_function_shape=decision_function_shape,
                                     class_weight=class_weight, random_state=random_state)
    score = sklearn.model_selection.cross_val_score(classifier_obj, x, y, n_jobs=-1, cv=3)
    accuracy = score.mean()
    return accuracy

classifier_obj = sklearn.svm.SVC()
"""
The reason why it has two classifierobj is because one is used in the objective function
as a parameter to optimize, and the other is used outside of the function as a classifier
object that is trained using the best parameters found during the optimization process. The
classifierobj in the objective function is defined as an SVC object with various parameters
that are suggested by Optuna during the optimization process."""

if __name__ == "__main__":
  study = optuna.create_study(direction="maximize", pruner=optuna.pruners.MedianPruner())
  # study.optimize(objective, n_trials=100)
  x_train, y_train, x_test, y_test = getData(path)
  func = lambda trial: objective(trial, x_train, y_train)
  study.optimize(func, n_trials=100)
  # To get the dictionary of parameter name and parameter values:
  print("Return a dictionary of parameter name and parameter values:",study.best_params)

  # To get the best observed value of the objective function:
  print("Return the best observed value of the objective function:",study.best_value)

  # To get the best trial:
  print("Return the best trial:",study.best_trial)

  # To get all trials:
  print("Return all the trials:", study.trials)
```

```
[I 2023-01-27 21:19:29,188] Trial 80 finished with value: 1.0 and parameters: {'C': 2.0012434596207837e-08, 'kernel': 'linear', 'degre
[I 2023-01-27 21:19:29,389] Trial 81 finished with value: 1.0 and parameters: {'C': 302528.060957898, 'kernel': 'poly', 'degree': 3,
[I 2023-01-27 21:19:34,196] Trial 82 finished with value: 1.0 and parameters: {'C': 5.1495710992439156e-08, 'kernel': 'poly', 'degree
[I 2023-01-27 21:19:39,399] Trial 83 finished with value: 1.0 and parameters: {'C': 5.077479960065109e-05, 'kernel': 'poly', 'degree':
[I 2023-01-27 21:19:41,140] Trial 84 finished with value: 1.0 and parameters: {'C': 1.0227641516261573e-10, 'kernel': 'poly', 'degree
[I 2023-01-27 21:19:41,349] Trial 85 finished with value: 1.0 and parameters: {'C': 39873110.515426986, 'kernel': 'sigmoid', 'degree':
[I 2023-01-27 21:19:48,392] Trial 86 finished with value: 1.0 and parameters: {'C': 7.004810782462146e-09, 'kernel': 'rbf', 'degree':
[I 2023-01-27 21:19:48,598] Trial 87 finished with value: 1.0 and parameters: {'C': 4.418951904462473e-07, 'kernel': 'poly', 'degree':
[I 2023-01-27 21:19:52,962] Trial 88 finished with value: 1.0 and parameters: {'C': 1.9607290608721748e-09, 'kernel': 'poly', 'degree
[I 2023-01-27 21:19:54,202] Trial 89 finished with value: 1.0 and parameters: {'C': 3.1102400448345565e-06, 'kernel': 'rbf', 'degree':
[I 2023-01-27 21:19:54,405] Trial 90 finished with value: 1.0 and parameters: {'C': 150736168.1456663, 'kernel': 'linear', 'degree': 3
[I 2023-01-27 21:19:59,029] Trial 91 finished with value: 1.0 and parameters: {'C': 9.629647193155666e-08, 'kernel': 'rbf', 'degree':
[I 2023-01-27 21:20:04,126] Trial 92 finished with value: 1.0 and parameters: {'C': 1.5124087654231336e-08, 'kernel': 'rbf', 'degree':
[I 2023-01-27 21:20:04,332] Trial 93 finished with value: 1.0 and parameters: {'C': 2041785969.443911, 'kernel': 'rbf', 'degree': 7,
[I 2023-01-27 21:20:08,738] Trial 94 finished with value: 1.0 and parameters: {'C': 4.882692543727835e-10, 'kernel': 'rbf', 'degree':
[I 2023-01-27 21:20:11,611] Trial 95 finished with value: 1.0 and parameters: {'C': 4.5430371466509986e-08, 'kernel': 'poly', 'degree
[I 2023-01-27 21:20:14,596] Trial 96 finished with value: 0.6375416666666667 and parameters: {'C': 2.5092616193186513e-07, 'kernel':
[I 2023-01-27 21:20:14,798] Trial 97 finished with value: 1.0 and parameters: {'C': 17207021.816538144, 'kernel': 'linear', 'degree':
[I 2023-01-27 21:20:15,012] Trial 98 finished with value: 1.0 and parameters: {'C': 6.709856679012939e-07, 'kernel': 'poly', 'degree':
[I 2023-01-27 21:20:19,022] Trial 99 finished with value: 0.3624583333333333 and parameters: {'C': 1.52690939872675e-06, 'kernel': 'si
Return a dictionary of parameter name and parameter values: {'C': 9.34570367113371e-09, 'kernel': 'rbf', 'degree': 8, 'coef0': 0.96365
Return the best observed value of the objective function: 1.0
Return the best trial: FrozenTrial(number=1, state=TrialState.COMPLETE, values=[1.0], datetime_start=datetime.datetime(2023, 1, 27, 21
Return all the trials: [FrozenTrial(number=0, state=TrialState.COMPLETE, values=[0.3624583333333333], datetime_start=datetime.datetime
```

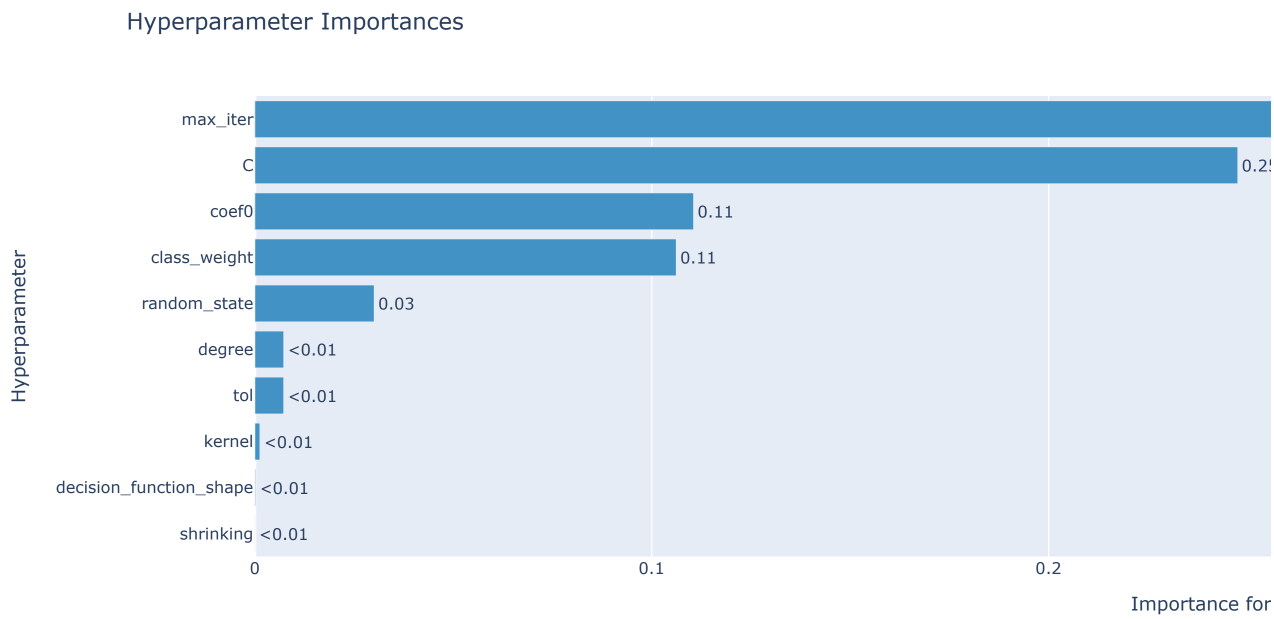
```
# Visualize the optimization history. See :func:`~optuna.visualization.plot_optimization_history` for the details.
plot_optimization_history(study)
```



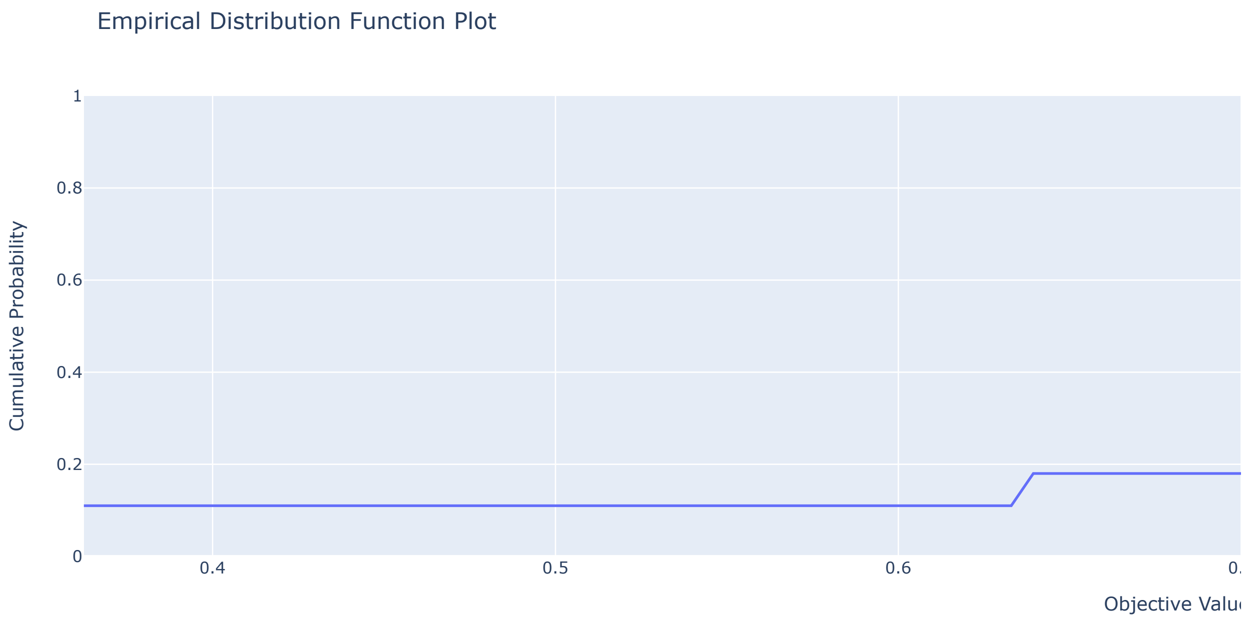

```
# Visualize individual hyperparameters as slice plot. See :func:`~optuna.visualization.plot_slice` for the details.
plot_slice(study)
```



```
# Visualize parameter importances. See :func:`~optuna.visualization.plot_param_importances` for the details.
plot_param_importances(study)
```




```
# Visualize empirical distribution function. See :func:`~optuna.visualization.plot_edf` for the details.
plot_edf(study)
```




```
# Visualize high-dimensional parameter relationships. See :func:`~optuna.visualization.plot_parallel_coordinate` for the details.
plot_parallel_coordinate(study)
```


```
"""Here we will first set the parameters of the classifier_obj to the
best parameters found by the Optuna study, using the set_params() method.
Then it will fit the classifier_obj to the training data using the fit() method.
Next, it will use the classifier_obj to make predictions on the test data and store
the predictions in y_pred. Then it will calculate the accuracy of the classifier_obj
on the test data by comparing the predictions to the true labels, and print the accuracy.
Finally, it will generate and print a classification report which includes precision, recall,
f1-score, and support for each class.
It's important to note that this is an example of using the best parameters to make predictions
on unseen test data and check the performance of the model on this data."""

classifier_obj.set_params(**study.best_params)
classifier_obj.fit(x_train, y_train)
y_pred = classifier_obj.predict(x_test)
accuracy = sklearn.metrics.accuracy_score(y_test, y_pred)
print(f"Accuracy on test set: {accuracy:.4f}")
report = sklearn.metrics.classification_report(y_test, y_pred)
print("\nClassification report:\n", report)
```

```
Accuracy on test set: 1.0000

Classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      7678
           1       1.00      1.00      1.00      4322

    accuracy                           1.00     12000
   macro avg       1.00      1.00      1.00     12000
weighted avg       1.00      1.00      1.00     12000
```

```
study.trials_dataframe()
```

	number	value	datetime_start	datetime_complete	duration	params_C	params_class_weight	params_coef
0	0	0.362458	2023-01-27 21:16:39.506970	2023-01-27 21:16:42.655636	0 days 00:00:03.148666	3.055764e-10	balanced	0.14622
1	1	1.000000	2023-01-27 21:16:42.659401	2023-01-27 21:16:53.509952	0 days 00:00:10.850551	9.345704e-09	None	0.96365
2	2	0.362458	2023-01-27 21:16:53.513717	2023-01-27 21:16:59.708912	0 days 00:00:06.195195	8.563053e-09	balanced	-0.34839
3	3	1.000000	2023-01-27 21:16:59.712595	2023-01-27 21:17:06.799836	0 days 00:00:07.087241	8.588176e-10	None	0.22472
4	4	1.000000	2023-01-27 21:17:06.802320	2023-01-27 21:17:06.971857	0 days 00:00:00.169537	1.225651e+08	None	0.41765
...	...	...	...	...	...	...	...	
95	95	1.000000	2023-01-27 21:20:08.739727	2023-01-27 21:20:11.610596	0 days 00:00:02.870869	4.543037e-08	None	0.23859
96	96	0.637542	2023-01-27 21:20:11.613080	2023-01-27 21:20:14.596110	0 days 00:00:02.983030	2.509262e-07	balanced	0.59323
97	97	1.000000	2023-01-27 21:20:14.598887	2023-01-27 21:20:14.797833	0 days 00:00:00.198946	1.720702e+07	None	0.62866
98	98	1.000000	2023-01-27 21:20:14.800126	2023-01-27 21:20:15.011622	0 days 00:00:00.211496	6.709857e-07	None	0.91729
99	99	0.362458	2023-01-27 21:20:15.013685	2023-01-27 21:20:19.022066	0 days 00:00:04.008381	1.526909e-06	balanced	0.39212

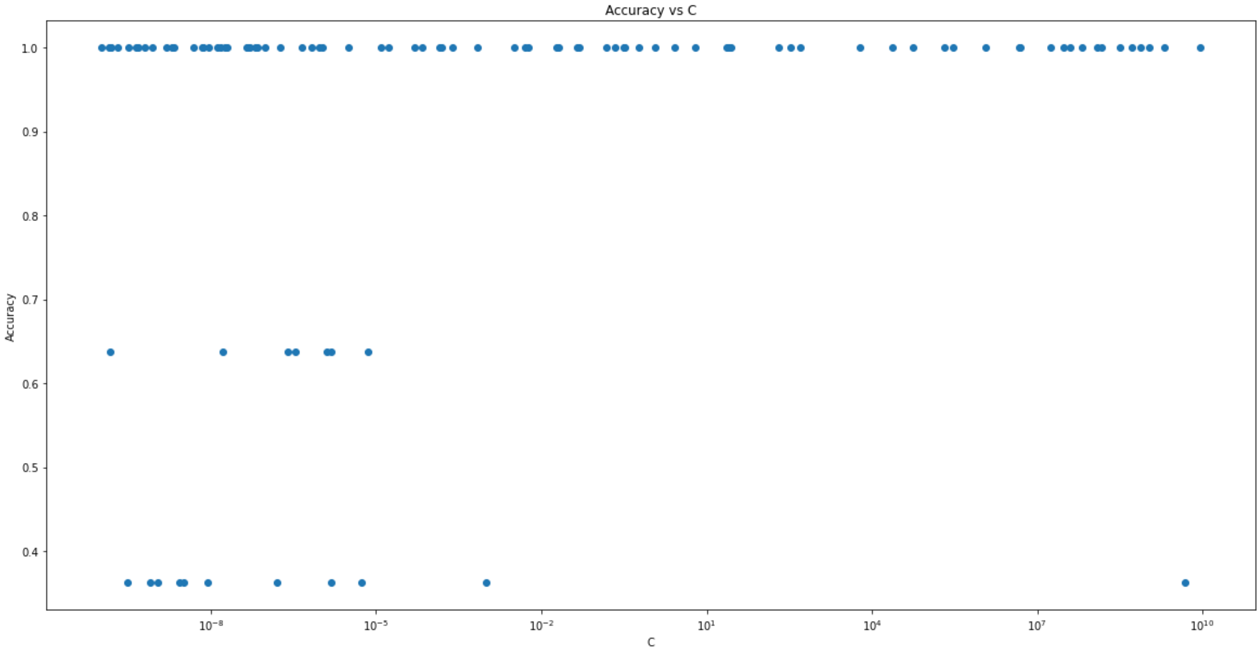
100 rows × 16 columns

```
type(study.trials_dataframe())
study.trials_dataframe().columns
```

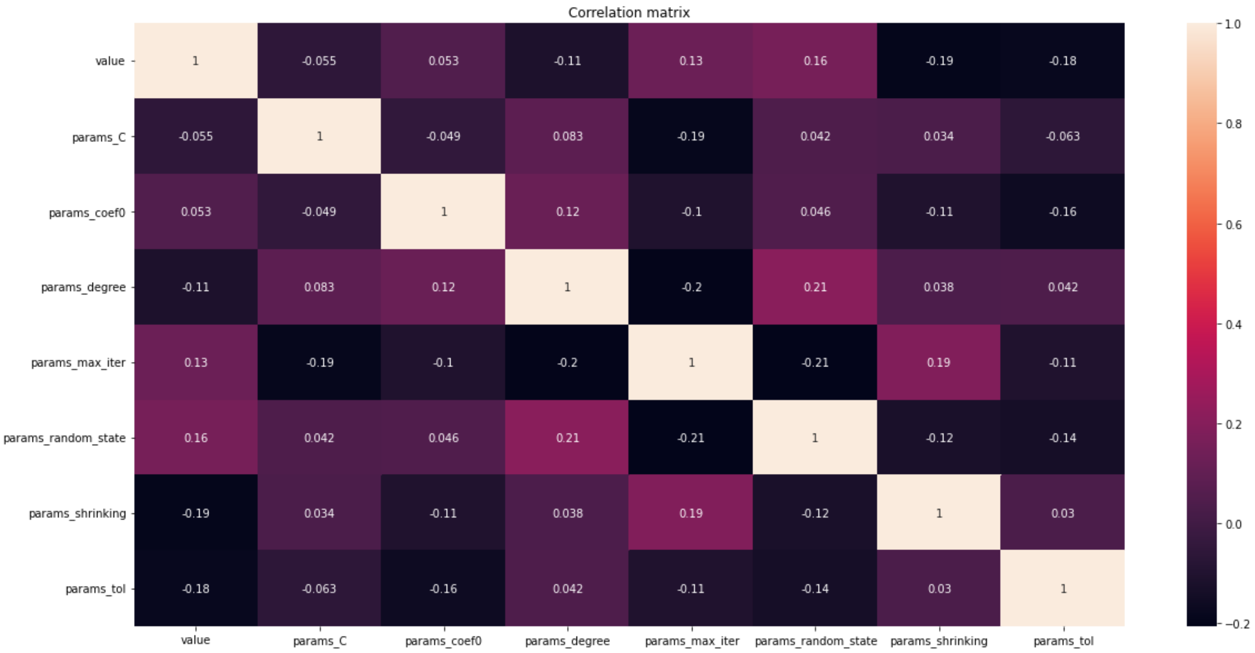
```
Index(['number', 'value', 'datetime_start', 'datetime_complete', 'duration',
       'params_C', 'params_class_weight', 'params_coef0',
       'params_decision_function_shape', 'params_degree', 'params_kernel',
       'params_max_iter', 'params_random_state', 'params_shrinking',
```

```
                 'params_tol', 'state'],
                dtype='object')
```

```
plt.figure(figsize=(20,10))
plt.scatter(study.trials_dataframe()['params_C'], study.trials_dataframe()['value'])
plt.xscale("log")
plt.xlabel("C")
plt.ylabel("Accuracy")
plt.title("Accuracy vs C")
plt.show()
```




```
corr = study.trials_dataframe().drop(['number','datetime_start','datetime_complete','duration','state'], axis=1).corr()
plt.figure(figsize=(20,10))
sns.heatmap(corr, annot = True)
plt.title("Correlation matrix")
plt.show()
```



	value	params_C	params_coef0	params_degree	params_max_iter	params_random_state	params_shrinking	params_tol
value	1	-0.055	0.053	-0.11	0.13	0.16	-0.19	-0.18
params_C	-0.055	1	-0.049	0.083	-0.19	0.042	0.034	-0.063
params_coef0	0.053	-0.049	1	0.12	-0.1	0.046	-0.11	-0.16
params_degree	-0.11	0.083	0.12	1	-0.2	0.21	0.038	0.042
params_max_iter	0.13	-0.19	-0.1	-0.2	1	-0.21	0.19	-0.11
params_random_state	0.16	0.042	0.046	0.21	-0.21	1	-0.12	-0.14
params_shrinking	-0.19	0.034	-0.11	0.038	0.19	-0.12	1	0.03
params_tol	-0.18	-0.063	-0.16	0.042	-0.11	-0.14	0.03	1

0s completed at 22:20