```
### Network Slicing Recognition
```

The telecom industry is going through a massive digital transformation with the adoption of ML, AI, feedback-based automation and advanced analytics to handle the next generation applications and services. AI concepts are not new; the algorithms used by Machine Learning and Deep Learning are being currently implemented in various industries and technology verticals. With growing data and immense volume

of information over 5G, the ability to predict data proactively, swiftly and with accuracy, is critically important. Data-driven decision making will be vital in future communication networks due to the traffic explosion and Artificial Intelligence (AI) will accelerate the 5G network performance.

Mobile operators are looking for a programmable solution that will allow them to accommodate multiple independent tenants on the same physical infrastructure and 5G networks allow for end-to-end network resource allocation using the concept of Network Slicing (NS).

Network Slicing will play a vital role in enabling a multitude of 5G applications, use cases, and services. Network slicing functions will provide an end-to-end isolation between slices with an ability to customize each slice based on the service demands (bandwidth, coverage, security, latency, reliability, etc).

Your Task is to build a Machine Learning model that will be able to to proactively detect and eliminate threats based on incoming connections thereby selecting the most appropriate network slice, even in case of a network failure.

LTE/5g - User Equipment categories or classes to define the performance specifications
Packet Loss Rate - number of packets not received divided by the total number of packets sent.
Packet Delay - The time for a packet to be received.
Slice type - network configuration that allows multiple networks (virtualized and independent)
GBR - Guaranteed Bit Rate
Healthcare - Usage in Healthcare (1 or 0)
Industry 4.0 - Usage in Digital Enterprises(1 or 0)
IoT Devices - Usage
Public Safety - Usage for public welfare and safety purposes (1 or 0)
Smart City & Home - usage in daily household chores
Smart Transportation - usage in public transportation
Smartphone - whether used for smartphone cellular data

```
##! pip install neattext
```

```
import pandas as pd
import numpy as np
import neattext.functions as nfx
import seaborn as sn

from sklearn.feature_extraction.text import TfidfVectorizer,CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity,linear_kernel
```

```
###!pip uninstall numpy
##!pip install numpy==1.20
```

```
###!mkdir ~/.kaggle
```

```
###!cp /kaggle.json ~/.kaggle/
```

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

```
###!kaggle datasets download -d gauravduttakiit/network-slicing-recognition
```

```
##!unzip /content/network-slicing-recognition.zip
```

```
train_dataset = pd.read_csv("/content/train_dataset.csv")
test_dataset = pd.read_csv("/content/test_dataset.csv")
```

```
print(train_dataset.shape, test_dataset.shape)
```

```
(31583, 17) (31584, 16)
```

```
test_dataset['slice Type'] = 0
```

```
train_dataset = train_dataset.reset_index()
test_dataset = test_dataset.reset_index()
train_dataset.rename(columns = { "index" : "ID"}, inplace = True)
test_dataset.rename(columns = { "index" : "ID"}, inplace = True)
```

```
train_dataset.columns
```

```
Index(['ID', 'LTE/5g Category', 'Time', 'Packet Loss Rate', 'Packet delay',
       'IoT', 'LTE/5G', 'GBR', 'Non-GBR', 'AR/VR/Gaming', 'Healthcare',
       'Industry 4.0', 'IoT Devices', 'Public Safety', 'Smart City & Home',
       'Smart Transportation', 'Smartphone', 'slice Type'],
      dtype='object')
```

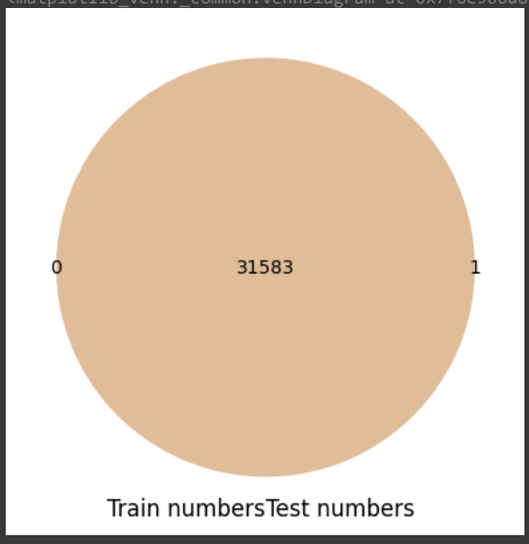
```
train_dataset['slice Type'].value_counts()
```

```
1    16799
3     7392
2     7392
Name: slice Type, dtype: int64
```

```
from matplotlib_venn import venn2, venn2_circles, venn2_unweighted
from matplotlib_venn import venn3, venn3_circles
```

```
set_numbers_train = set(train_dataset[['ID']].drop_duplicates().sort_values(by = 'ID')['ID'].tolist())
set_numbers_test = set(test_dataset[['ID']].drop_duplicates().sort_values(by = 'ID')['ID'].tolist())
venn2((set_numbers_train, set_numbers_test), set_labels = ('Train numbers', 'Test numbers'))
```

```
<matplotlib_venn._common.VennDiagram at 0x7f0e960d8280>
```




```
train_dataset.columns
```

```
Index(['ID', 'LTE/5g Category', 'Time', 'Packet Loss Rate', 'Packet delay',
       'IoT', 'LTE/5G', 'GBR', 'Non-GBR', 'AR/VR/Gaming', 'Healthcare',
       'Industry 4.0', 'IoT Devices', 'Public Safety', 'Smart City & Home',
       'Smart Transportation', 'Smartphone', 'slice Type'],
      dtype='object')
```

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

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

```
import klib
```

```
train_dataset = klib.clean_column_names(train_dataset)
test_dataset = klib.clean_column_names(test_dataset)
```

```
train_dataset = klib.convert_datatypes(train_dataset)
test_dataset = klib.convert_datatypes(test_dataset)
```

```
train_dataset.columns
```

```
Index(['id', 'lte_5g_category', 'time', 'packet_loss_rate', 'packet_delay',
       'io_t', 'lte_5g', 'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare',
       'industry_4_0', 'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone', 'slice_type'],
      dtype='object')
```

Anomaly Detection Using One-Class SVM

```
from sklearn import svm
```

```
clf = svm.OneClassSVM(nu=0.05, kernel="rbf", gamma=0.1)
clf.fit(train_dataset)
```

```
OneClassSVM(gamma=0.1, nu=0.05)
```

```
pred = clf.predict(train_dataset)

# inliers are labeled 1, outliers are labeled -1
normal = train_dataset[pred == 1]
abnormal = train_dataset[pred == -1]
```

```
print(normal.shape, abnormal.shape)
```

```
(18373, 18) (13210, 18)
```

```
normal.columns
```

```
Index(['id', 'lte_5g_category', 'time', 'packet_loss_rate', 'packet_delay',
       'io_t', 'lte_5g', 'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare',
       'industry_4_0', 'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone', 'slice_type'],
      dtype='object')
```

```
normal['slice_type'].value_counts()
```

```
1    9839
2    4294
3    4240
Name: slice_type, dtype: int64
```

```
train_dataset = normal
```

```
print(train_dataset.shape)
print(train_dataset.columns)
```

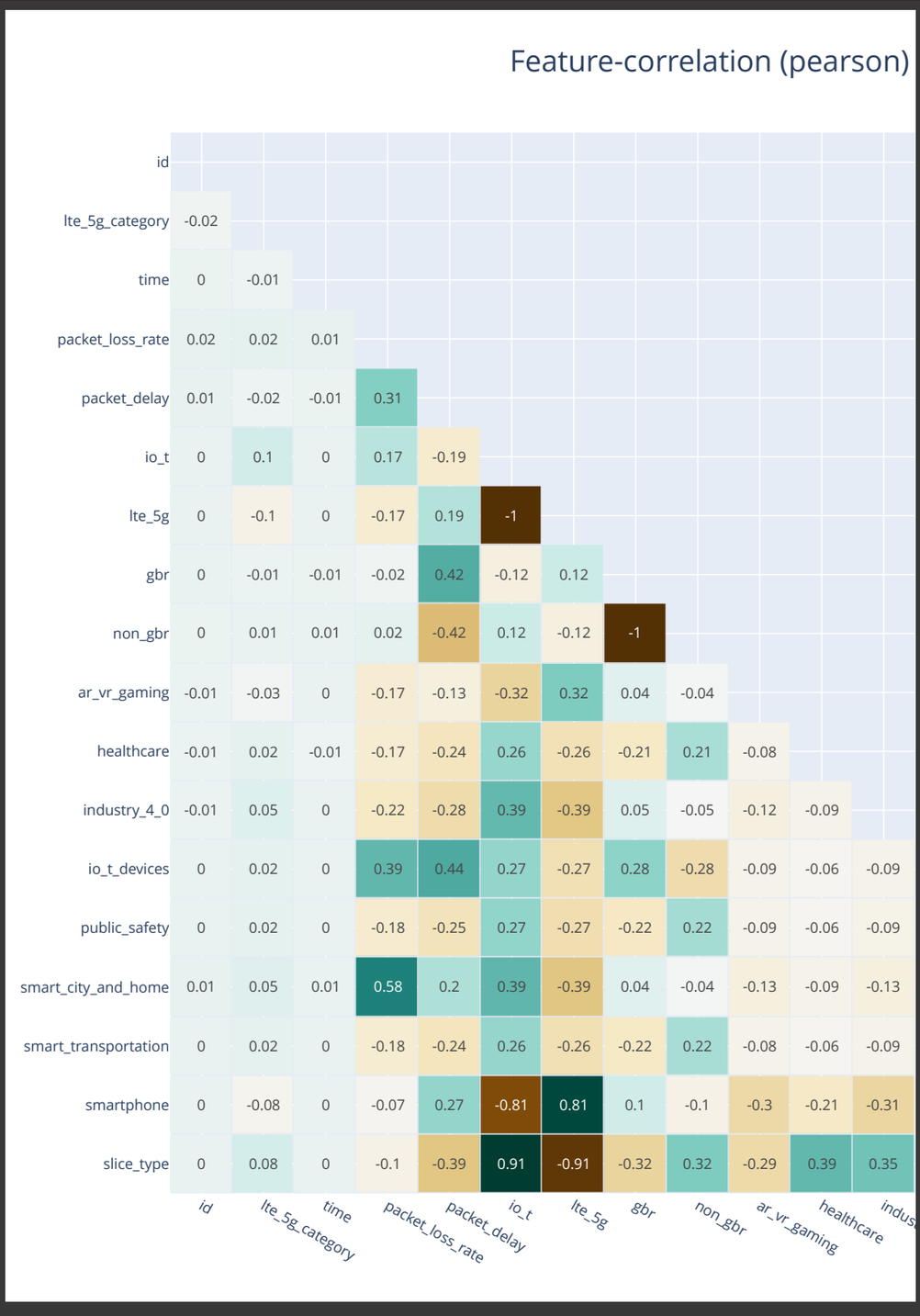
```
(18373, 18)
Index(['id', 'lte_5g_category', 'time', 'packet_loss_rate', 'packet_delay',
       'io_t', 'lte_5g', 'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare',
       'industry_4_0', 'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone', 'slice_type'],
      dtype='object')
```

```
test_dataset['slice_type'] = 0
```

```
klib.cat_plot(train_dataset)
```

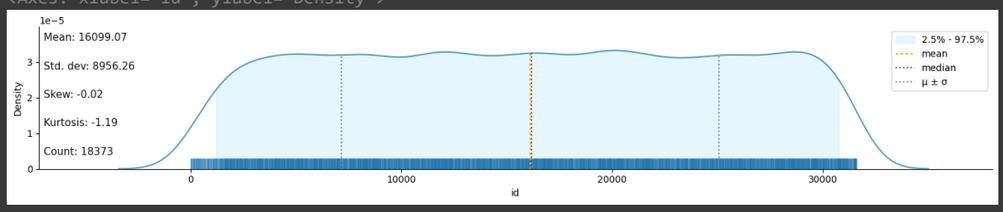
```
No columns with categorical data were detected.
```

```
klib.corr_interactive_plot(train_dataset)
```




```
klib.dist_plot(train_dataset)
```

```
Large dataset detected, using 10000 random samples for the plots. Summary statistics are still ba
<Axes: xlabel='id', ylabel='Density'>
```




```
klib.corr_mat(train_dataset)
```

	id	lte_5g_category	time	packet_loss_rate	packet_delay	io_t	lte_5g	gb
id	1.00	-0.02	-0.00	0.02	0.01	0.00	-0.00	0.0
lte_5g_category	-0.02	1.00	-0.01	0.02	-0.02	0.10	-0.10	-0.0
time	-0.00	-0.01	1.00	0.01	-0.01	0.00	-0.00	-0.0
packet_loss_rate	0.02	0.02	0.01	1.00	0.31	0.17	-0.17	-0.0
packet_delay	0.01	-0.02	-0.01	0.31	1.00	-0.19	0.19	0.4
io_t	0.00	0.10	0.00	0.17	-0.19	1.00	-1.00	-0.1
lte_5g	-0.00	-0.10	-0.00	-0.17	0.19	-1.00	1.00	0.1
gbr	0.00	-0.01	-0.01	-0.02	0.42	-0.12	0.12	1.0
non_gbr	-0.00	0.01	0.01	0.02	-0.42	0.12	-0.12	-1.0
ar_vr_gaming	-0.01	-0.03	-0.00	-0.17	-0.13	-0.32	0.32	0.0
healthcare	-0.01	0.02	-0.01	-0.17	-0.24	0.26	-0.26	-0.2
industry_4_0	-0.01	0.05	0.00	-0.22	-0.28	0.39	-0.39	0.0
io_t_devices	0.00	0.02	-0.00	0.39	0.44	0.27	-0.27	0.4
public_safety	0.00	0.02	0.00	-0.18	-0.25	0.27	-0.27	-0.2
smart_city_and_home	0.01	0.05	0.01	0.58	0.20	0.39	-0.39	0.0
smart_transportation	-0.00	0.02	-0.00	-0.18	-0.24	0.26	-0.26	-0.2
smartphone	0.00	-0.08	0.00	-0.07	0.27	-0.81	0.81	0.1
slice_type	-0.00	0.08	-0.00	-0.10	-0.39	0.91	-0.91	-0.3

```
train_dataset.columns
```

```
Index(['id', 'lte_5g_category', 'time', 'packet_loss_rate', 'packet_delay',
       'io_t', 'lte_5g', 'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare',
       'industry_4_0', 'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone', 'slice_type'],
      dtype='object')
```

```
# Checking for outliers in the continuous variables
num_train_dataset = train_dataset[['id', 'lte_5g_category', 'time', 'packet_loss_rate', 'packet_delay',
       'io_t', 'lte_5g', 'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare',
       'industry_4_0', 'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone', 'slice_type']]
```

```
train_dataset['slice_type'].value_counts()
```

```
1    9839
2    4294
3    4240
Name: slice_type, dtype: int64
```

```
y_train = train_dataset['slice_type']
x_train = train_dataset.drop('slice_type', axis = 1)
y_test = test_dataset['slice_type']
x_test = test_dataset.drop('slice_type', axis = 1)
```

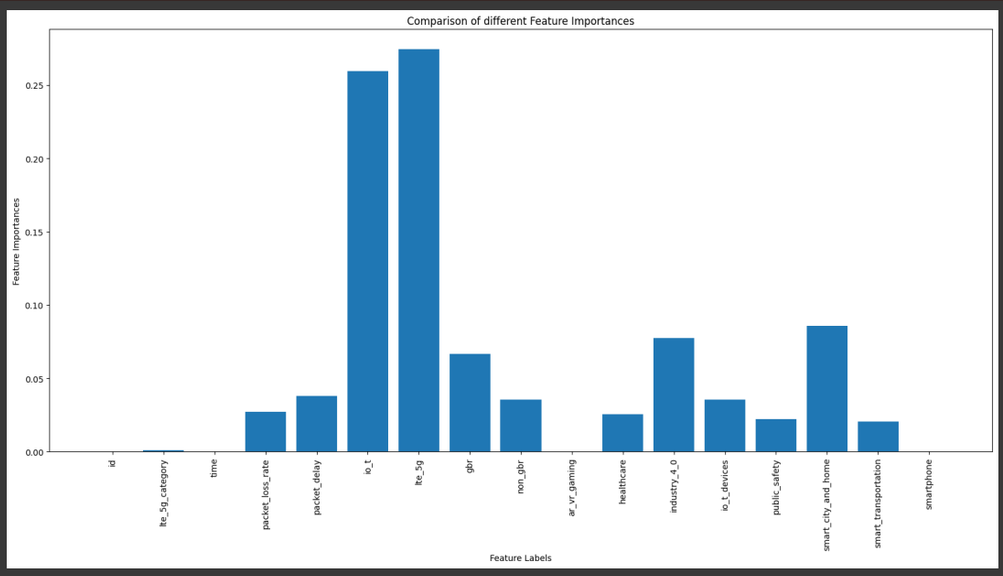
```
from sklearn.ensemble import ExtraTreesClassifier
extra_tree_forest = ExtraTreesClassifier(n_estimators = 5,
                                         criterion ='entropy', max_features = 2)
extra_tree_forest.fit(x_train, y_train)
feature_importance = extra_tree_forest.feature_importances_
feature_importance_normalized = np.std([tree.feature_importances_ for tree in
                                         extra_tree_forest.estimators_],
                                         axis = 0)
```

```
feature_importance_normalized
```

```
array([8.11105094e-05, 8.21203671e-04, 3.08990347e-06, 2.70626290e-02,
       3.79638484e-02, 2.59463296e-01, 2.74289331e-01, 6.66559389e-02,
       3.53843028e-02, 0.00000000e+00, 2.55993658e-02, 7.76090876e-02,
       3.55759345e-02, 2.23089459e-02, 8.57165291e-02, 2.05353386e-02,
       0.00000000e+00])
```

```
import matplotlib.pyplot as plt
```

```
plt.figure(figsize = [20,9])
plt.bar(x_train.columns, feature_importance_normalized)
plt.xlabel('Feature Labels')
plt.ylabel('Feature Importances')
plt.xticks(rotation = 90)
plt.title('Comparison of different Feature Importances')
plt.show()
```




```
x_train.columns
```

```
Index(['id', 'lte_5g_category', 'time', 'packet_loss_rate', 'packet_delay',
       'io_t', 'lte_5g', 'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare',
       'industry_4_0', 'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone'],
      dtype='object')
```

```
x_train2 = x_train[['lte_5g_category', 'packet_loss_rate', 'packet_delay',
       'io_t', 'lte_5g', 'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare',
       'industry_4_0', 'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone']]
```

```
x_test2 = x_test[['lte_5g_category', 'packet_loss_rate', 'packet_delay',
       'io_t', 'lte_5g', 'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare',
       'industry_4_0', 'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone']]
```

```
print(x_train2.shape, x_test2.shape)
```

```
(18373, 15) (31584, 15)
```

```
x_train2 = pd.DataFrame(x_train2)
x_test2 = pd.DataFrame(x_test2)
```

```
print(y_train.shape, y_test.shape)
```

```
(18373,) (31584,)
```

```
y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)
```

```
x_train2.astype(float).corr()
```

	lte_5g_category	packet_loss_rate	packet_delay	io_t	lte_5g	gbr
lte_5g_category	1.000000	0.022254	-0.015085	0.095605	-0.095605	-0.007156
packet_loss_rate	0.022254	1.000000	0.306436	0.170858	-0.170858	-0.022315
packet_delay	-0.015085	0.306436	1.000000	-0.187910	0.187910	0.424124
io_t	0.095605	0.170858	-0.187910	1.000000	-1.000000	-0.122402
lte_5g	-0.095605	-0.170858	0.187910	-1.000000	1.000000	0.122402
gbr	-0.007156	-0.022315	0.424124	-0.122402	0.122402	1.000000
non_gbr	0.007156	0.022315	-0.424124	0.122402	-0.122402	-1.000000
ar_vr_gaming	-0.034010	-0.168172	-0.127614	-0.321284	0.321284	0.037476
healthcare	0.018408	-0.174050	-0.238384	0.260863	-0.260863	-0.213782
industry_4_0	0.045106	-0.216202	-0.284384	0.385515	-0.385515	0.045353
io_t_devices	0.019458	0.389878	0.437355	0.265817	-0.265817	0.281337
public_safety	0.019485	-0.182358	-0.249763	0.273314	-0.273314	-0.223986
smart_city_and_home	0.047347	0.578743	0.200818	0.394585	-0.394585	0.035940
smart_transportation	0.019787	-0.176111	-0.241206	0.263951	-0.263951	-0.216313
smartphone	-0.075152	-0.067416	0.268831	-0.807524	0.807524	0.099995

```
def correlation(dataset, threshold):
    col_corr = set() # Set of all the names of deleted columns
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if (corr_matrix.iloc[i, j] >= threshold) and (corr_matrix.columns[j] not in col_corr):
                colname = corr_matrix.columns[i] # getting the name of column
                col_corr.add(colname)
                if colname in dataset.columns:
                    del dataset[colname] # deleting the column from the dataset

    print(dataset.columns)
    print(dataset.shape)
```

```
correlation(x_train2, 0.95)
```

```
Index(['lte_5g_category', 'packet_loss_rate', 'packet_delay', 'io_t', 'lte_5g',
       'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare', 'industry_4_0',
       'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone'],
      dtype='object')
(18373, 15)
```

```
print("The Testing Data :", x_test2.columns)
print("The Training Data :", x_train2.columns)
```

```
The Testing Data : Index(['lte_5g_category', 'packet_loss_rate', 'packet_delay', 'io_t', 'lte_5g',
       'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare', 'industry_4_0',
       'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone'],
      dtype='object')
The Training Data : Index(['lte_5g_category', 'packet_loss_rate', 'packet_delay', 'io_t', 'lte_5g',
       'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare', 'industry_4_0',
       'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone'],
      dtype='object')
```

```
y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)
```

```
print(y_test.columns, y_train.columns)
```

```
Index(['slice_type'], dtype='object') Index(['slice_type'], dtype='object')
```

BENTO ML

BentoML is designed for teams working to bring **machine learning (ML)** models into production in a reliable, scalable, and cost-efficient way. In particular, AI application developers can leverage **BentoML** to easily integrate state-of-the-art pre-trained models into their applications. By seamlessly bridging the gap between model creation and production deployment, BentoML promotes collaboration between **developers** and in-house **data science teams**.

```
###! pip install bentoml
```

```
import bentoml
```

```
import pandas as pd
from bentoml.io import NumpyNdarray, PandasDataFrame, JSON
import numpy as np
from pydantic import BaseModel
```

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

```
print(x_train2.shape, x_test2.shape)
```

```
(18373, 15) (31584, 15)
```

```
scaler = StandardScaler()
scaler.fit(x_train2)
x_train2 = pd.DataFrame(scaler.transform(x_train2), columns=x_train2.columns)
```

```
bentoml.sklearn.save("scaler", scaler)
```

```
WARNING:bentoml.sklearn:The "bentoml.sklearn.save" method is deprecated. Use "bentoml.sklearn.save_model" instead.
Model(tag="scaler:55qgr6gse6m34asc", path="/root/bentoml/models/scaler/55qgr6gse6m34asc/")
```

```
###! bentoml models list
```

Tag	Module	Size	Creation Time
scaler:55qgr6gse6m34asc	bentoml.sklearn	1.83 KiB	2024-02-23 08:45:45

```
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
clf_svm = svm.SVC(gamma='scale')
```

```
bentoml.sklearn.save("clf_svm", clf_svm)
clf_svm.fit(x_train2, y_train)
```

```
WARNING:bentoml.sklearn:The "bentoml.sklearn.save" method is deprecated. Use "bentoml.sklearn.save_model" instead.
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
```

▾ SVC

SVC()

```
clf_rf = RandomForestClassifier()
clf_rf.fit(x_train2, y_train)
```

```
<ipython-input-74-ae59304692b6>:2: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
```

▾ RandomForestClassifier

RandomForestClassifier()

```
bentoml.sklearn.save("clf_rf", clf_rf)
```

```
WARNING:bentoml.sklearn:The "bentoml.sklearn.save" method is deprecated. Use "bentoml.sklearn.save_model" instead.
Model(tag="clf_rf:xwtp6hgsgcm34asc", path="/root/bentoml/models/clf_rf/xwtp6hgsgcm34asc/")
```

```
clf_dt = DecisionTreeClassifier()
clf_dt.fit(x_train2, y_train)
```

▾ DecisionTreeClassifier

DecisionTreeClassifier()

```
bentoml.sklearn.save("clf_dt", clf_dt)
```

```
WARNING:bentoml.sklearn:The "bentoml.sklearn.save" method is deprecated. Use "bentoml.sklearn.save_model" instead.
Model(tag="clf_dt:5qnzltwsgcm34asc", path="/root/bentoml/models/clf_dt/5qnzltwsgcm34asc/")
```

```
from sklearn.linear_model import LogisticRegression
```

```
clf_lg = LogisticRegression()
clf_lg.fit(x_train2, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/utils/validation.py:1143: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
```

▾ LogisticRegression

LogisticRegression()

```
bentoml.sklearn.save("clf_lg", clf_lg)
```

```
WARNING:bentoml.sklearn:The "bentoml.sklearn.save" method is deprecated. Use "bentoml.sklearn.save_model" instead.
Model(tag="clf_lg:k443gggsggm34asc", path="/root/bentoml/models/clf_lg/k443gggsggm34asc/")
```

```
scaler = bentoml.sklearn.get("scaler:latest").to_runner()
clf_svm = bentoml.sklearn.get("clf_svm:latest").to_runner()
clf_rf = bentoml.sklearn.get("clf_rf:latest").to_runner()
clf_dt = bentoml.sklearn.get("clf_dt:latest").to_runner()
clf_lg = bentoml.sklearn.get("clf_lg:latest").to_runner()
```

```
scaler.init_local()
clf_svm.init_local()
clf_rf.init_local()
clf_dt.init_local()
clf_lg.init_local()
```

```
WARNING:bentoml._internal.runner.runner:'Runner.init_local' is for debugging and testing only. Make sure to remove it before deploying tc
WARNING:bentoml._internal.runner.runner:'Runner.init_local' is for debugging and testing only. Make sure to remove it before deploying tc
WARNING:bentoml._internal.runner.runner:'Runner.init_local' is for debugging and testing only. Make sure to remove it before deploying tc
WARNING:bentoml._internal.runner.runner:'Runner.init_local' is for debugging and testing only. Make sure to remove it before deploying tc
WARNING:bentoml._internal.runner.runner:'Runner.init_local' is for debugging and testing only. Make sure to remove it before deploying tc
```

```
##!bentoml models list
```

```
Tag                         Module           Size        Creation Time
clf_lg:k443gggsggm34asc     bentoml.sklearn  2.08 KiB    2024-02-23 09:53:05
clf_dt:5qnzltwsgcm34asc     bentoml.sklearn  2.51 KiB    2024-02-23 09:50:05
clf_rf:xwtp6hgsgcm34asc     bentoml.sklearn  116.30 KiB  2024-02-23 09:48:47
clf_svm:g6fxucwsfcm34asc    bentoml.sklearn  688.00 B    2024-02-23 08:47:46
clf_svm:gbrsqggsfcm34asc    bentoml.sklearn  688.00 B    2024-02-23 08:47:34
scaler:55qgr6gse6m34asc     bentoml.sklearn  1.83 KiB    2024-02-23 08:45:45
```

```
###!bentoml models get clf_svm:latest
```

```
name: clf_svm
version: g6fxucwsfcm34asc
module: bentoml.sklearn
labels: {}
options: {}
metadata: {}
context:
  framework_name: sklearn
  framework_versions:
    scikit-learn: 1.2.2
  bentoml_version: 1.2.4
  python_version: 3.10.12
signatures:
  predict:
    batchable: false
api_version: v1
creation_time: '2024-02-23T08:47:46.749510+00:00'
```

```
###!bentoml models get clf_rf:latest
```

```
name: clf_rf
version: xwtp6hgsgcm34asc
module: bentoml.sklearn
labels: {}
options: {}
metadata: {}
context:
  framework_name: sklearn
  framework_versions:
    scikit-learn: 1.2.2
  bentoml_version: 1.2.4
  python_version: 3.10.12
signatures:
  predict:
    batchable: false
api_version: v1
creation_time: '2024-02-23T09:48:47.719354+00:00'
```

```
print(x_train2.columns)
print(x_test2.columns)
```

```
Index(['lte_5g_category', 'packet_loss_rate', 'packet_delay', 'io_t', 'lte_5g',
       'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare', 'industry_4_0',
       'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone'],
      dtype='object')
Index(['lte_5g_category', 'packet_loss_rate', 'packet_delay', 'io_t', 'lte_5g',
       'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare', 'industry_4_0',
       'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone'],
      dtype='object')
```

```
x_train2 = pd.DataFrame(x_train2)
x_test2 = pd.DataFrame(x_test2)
```

```
y_train = pd.DataFrame(y_train)
y_test = pd.DataFrame(y_test)
```

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

```
#scaler = bentoml.sklearn.get("scaler:latest").to_runner()
##clf_svm = bentoml.sklearn.get("clf_svm:latest").to_runner()
#clf_rf = bentoml.sklearn.get("clf_rf:latest").to_runner()
#clf_dt = bentoml.sklearn.get("clf_dt:latest").to_runner()
#clf_lg = bentoml.sklearn.get("clf_lg:latest").to_runner()
```

```
import numpy as np
import bentoml
from bentoml.io import NumpyNdarray
from bentoml.io import PandasDataFrame
import pandas as pd
```

```
import pandas as pd
scaler = bentoml.sklearn.get("scaler:latest").to_runner()
clf_rf = bentoml.sklearn.get("clf_rf:latest").to_runner()
service = bentoml.Service("water_quality_prediction", runners=[scaler, clf_rf])
@service.api(input=PandasDataFrame(), output=NumpyNdarray())
def predict(df: pd.DataFrame) -> np.array:

    # Process data
    scaled_df  = pd.DataFrame([scaler.run(df)], columns=df.columns)
    #processed_df = pd.DataFrame([pca.run(df)], columns=['col1', 'col2', 'col3'])

    # Predict data
    result = clf_rf.predict.run(scaled_df)
    return np.array(result)
```

```
%%writefile service.py
import numpy as np
import bentoml
from bentoml.io import NumpyNdarray
from bentoml.io import PandasDataFrame
import pandas as pd

scaler = bentoml.sklearn.get("scaler:latest").to_runner()
clf_rf = bentoml.sklearn.get("clf_rf:latest").to_runner()
service = bentoml.Service("water_quality_prediction", runners=[scaler, clf_rf])
@service.api(input=PandasDataFrame(), output=NumpyNdarray())
def predict(df: pd.DataFrame) -> np.array:

    # Process data
    scaled_df  = pd.DataFrame([scaler.run(df)], columns=df.columns)
    #processed_df = pd.DataFrame([pca.run(df)], columns=['col1', 'col2', 'col3'])

    # Predict data
    result = clf_rf.predict.run(scaled_df)
    return np.array(result)
```

```
Writing service.py
```

```
import bentoml
bento_model: bentoml.Model = bentoml.models.get("clf_rf:latest")

print(bento_model.path)
print(bento_model.info.metadata)
print(bento_model.info.labels)
```

```
/root/bentoml/models/clf_rf/xwtp6hgsgcm34asc
{}
{}
```

```
###!bentoml serve service:service
```

```
###! bentoml serve service.py:service --reload
```