```
### 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/
```

```
###!chmod 600 ~/.kaggle/kaggle.json
```

```
##! 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')
```

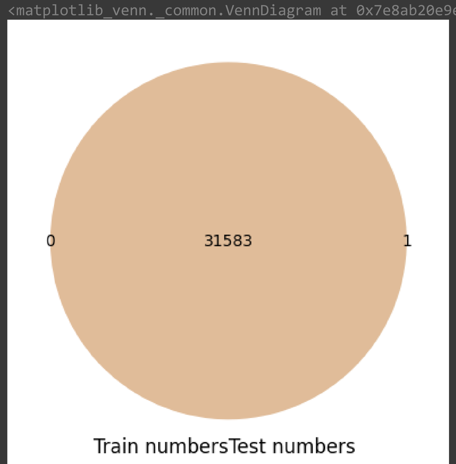
```
train_dataset['slice Type'].value_counts()
```

```
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 0x7e8ab20e9ed0>
```




```
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
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)
```

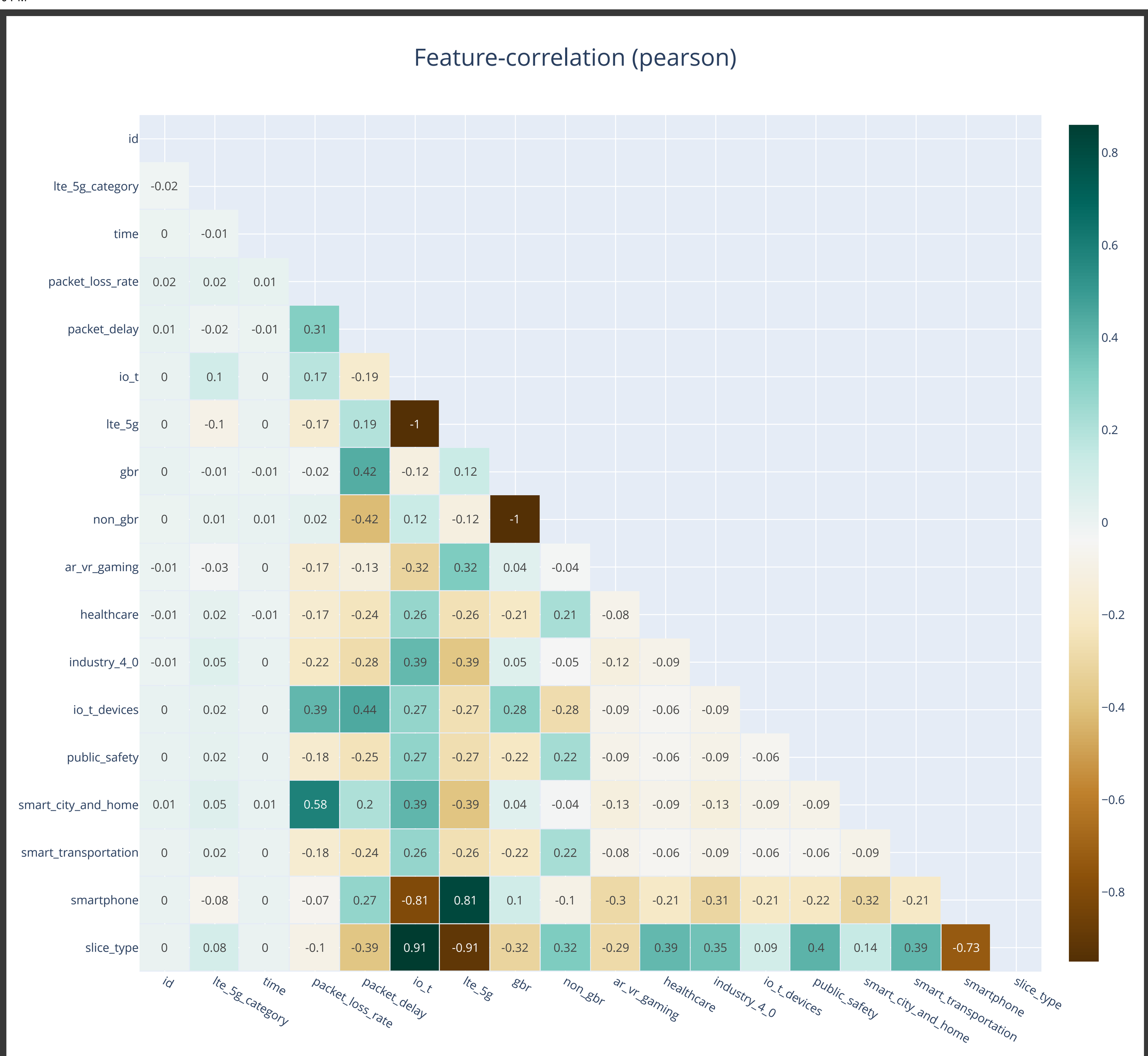
```
(18373, 18)
Index(['id', 'lte_5g_category', 'time', 'packet_loss_rate', 'packet_delay',
       'io_t', 'lte_5g', 'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare',
       'industry_4_0', 'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone', 'slice_type'],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

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

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

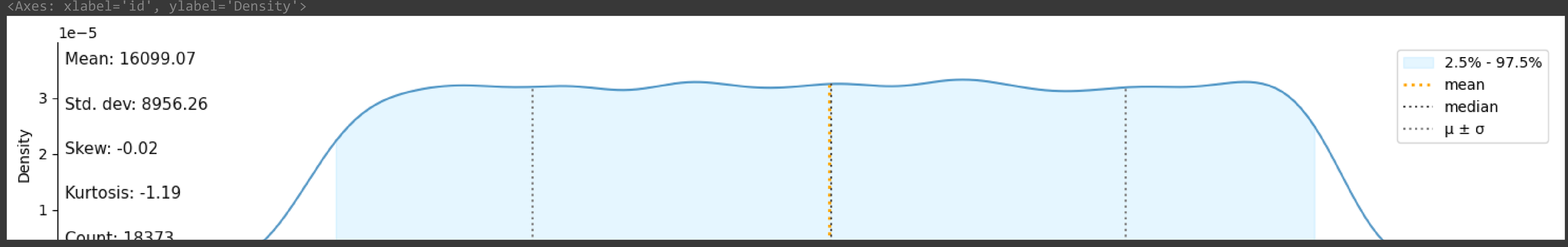
```
No columns with categorical data were detected.
```

```
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 based on the entire dataset.
<Axes: xlabel='id', ylabel='Density'>
```




```
klib.missingval_plot(train_dataset)
```

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

```
klib.corr_mat(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
id	1.00	-0.02	-0.00	0.02	0.01	0.00	-0.00	0.00	-0.00	-0.01	-0.01	-0.01	0.00	0.00	0.01	-0.00	0.00	-0.00
lte_5g_category	-0.02	1.00	-0.01	0.02	-0.02	0.10	-0.10	-0.01	0.01	-0.03	0.02	0.05	0.02	0.02	0.05	0.02	-0.08	0.08
time	-0.00	-0.01	1.00	0.01	-0.01	0.00	-0.00	-0.01	0.01	-0.00	-0.01	0.00	-0.00	0.00	0.01	-0.00	0.00	-0.00
packet_loss_rate	0.02	0.02	0.01	1.00	0.31	0.17	-0.17	-0.02	0.02	-0.17	-0.17	-0.22	0.39	-0.18	0.58	-0.18	-0.07	-0.10
packet_delay	0.01	-0.02	-0.01	0.31	1.00	-0.19	0.19	0.42	-0.42	-0.13	-0.24	-0.28	0.44	-0.25	0.20	-0.24	0.27	-0.39
io_t	0.00	0.10	0.00	0.17	-0.19	1.00	-1.00	-0.12	0.12	-0.32	0.26	0.39	0.27	0.27	0.39	0.26	-0.81	0.91
lte_5g	-0.00	-0.10	-0.00	-0.17	0.19	-1.00	1.00	0.12	-0.12	0.32	-0.26	-0.39	-0.27	-0.27	-0.39	-0.26	0.81	-0.91
gbr	0.00	-0.01	-0.01	-0.02	0.42	-0.12	0.12	1.00	-1.00	0.04	-0.21	0.05	0.28	-0.22	0.04	-0.22	0.10	-0.32
non_gbr	-0.00	0.01	0.01	0.02	-0.42	0.12	-0.12	-1.00	1.00	-0.04	0.21	-0.05	-0.28	0.22	-0.04	0.22	-0.10	0.32
ar_vr_gaming	-0.01	-0.03	-0.00	-0.17	-0.13	-0.32	0.32	0.04	-0.04	1.00	-0.08	-0.12	-0.09	-0.09	-0.13	-0.08	-0.30	-0.29
healthcare	-0.01	0.02	-0.01	-0.17	-0.24	0.26	-0.26	-0.21	0.21	-0.08	1.00	-0.09	-0.06	-0.06	-0.09	-0.06	-0.21	0.39
industry_4_0	-0.01	0.05	0.00	-0.22	-0.28	0.39	-0.39	0.05	-0.05	-0.12	-0.09	1.00	-0.09	-0.09	-0.13	-0.09	-0.31	0.35
io_t_devices	0.00	0.02	-0.00	0.39	0.44	0.27	-0.27	0.28	-0.28	-0.09	-0.06	-0.09	1.00	-0.06	-0.09	-0.06	-0.21	0.09
public_safety	0.00	0.02	0.00	-0.18	-0.25	0.27	-0.27	-0.22	0.22	-0.09	-0.06	-0.09	-0.06	1.00	-0.09	-0.06	-0.22	0.40
smart_city_and_home	0.01	0.05	0.01	0.58	0.20	0.39	-0.39	0.04	-0.04	-0.13	-0.09	-0.13	-0.09	-0.09	1.00	-0.09	-0.32	0.14
smart_transportation	-0.00	0.02	-0.00	-0.18	-0.24	0.26	-0.26	-0.22	0.22	-0.08	-0.06	-0.09	-0.06	-0.06	-0.09	1.00	-0.21	0.39
smartphone	0.00	-0.08	0.00	-0.07	0.27	-0.81	0.81	0.10	-0.10	-0.30	-0.21	-0.31	-0.21	-0.22	-0.32	-0.21	1.00	-0.73
slice_type	-0.00	0.08	-0.00	-0.10	-0.39	0.91	-0.91	-0.32	0.32	-0.29	0.39	0.35	0.09	0.40	0.14	0.39	-0.73	1.00

```
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)
```

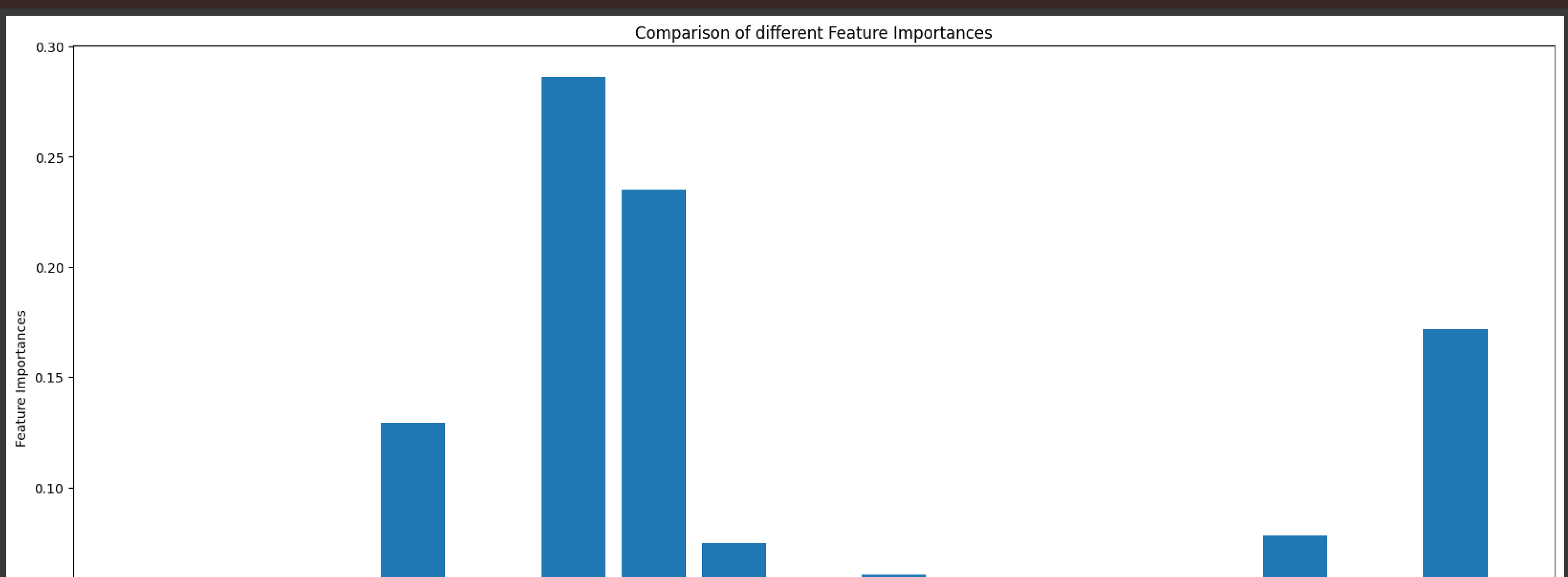
```
from sklearn.ensemble import ExtraTreesClassifier
extra_tree_forest = ExtraTreesClassifier(n_estimators = 5,
                                         criterion ='entropy', max_features = 2)
extra_tree_forest.fit(x_train, y_train)
feature_importance = extra_tree_forest.feature_importances_
feature_importance_normalized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

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

```
feature_importance_normalized
```

```
array([1.00712917e-05, 0.00000000e+00, 2.16989276e-07, 1.29379654e-01,
       2.82803661e-02, 2.85846553e-01, 2.34843792e-01, 7.45860560e-02,
       5.65496310e-02, 6.04741052e-02, 3.06829805e-02, 4.32631343e-02,
       1.85732114e-02, 0.00000000e+00, 7.82075506e-02, 3.53963038e-02,
       1.71630025e-01])
```

```
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)
```

```
y_train.value_counts()
```

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

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

Pearson Correlation

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

	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
lte_5g_category	1.000000	0.022254	-0.015085	0.095605	-0.095605	-0.007156	0.007156	-0.034010	0.018408	0.045106	0.019458	0.019485	0.047347	0.019787	-0.075152
packet_loss_rate	0.022254	1.000000	0.306436	0.170858	-0.170858	-0.022315	0.022315	-0.168172	-0.174050	-0.216202	0.389878	-0.182358	0.578743	-0.176111	-0.067416
packet_delay	-0.015085	0.306436	1.000000	-0.187910	0.187910	0.424124	-0.424124	-0.127614	-0.238384	-0.284384	0.437355	-0.249763	0.200818	-0.241206	0.268831
io_t	0.095605	0.170858	-0.187910	1.000000	-1.000000	-0.122402	0.122402	-0.321284	0.260863	0.385515	0.265817	0.273314	0.394585	0.263951	-0.807524
lte_5g	-0.095605	-0.170858	0.187910	-1.000000	1.000000	0.122402	-0.122402	0.321284	-0.260863	-0.385515	-0.265817	-0.273314	-0.394585	-0.263951	0.807524
gbr	-0.007156	-0.022315	0.424124	-0.122402	0.122402	1.000000	-1.000000	0.037476	-0.213782	0.045353	0.281337	-0.223986	0.035940	-0.216313	0.099995
non_gbr	0.007156	0.022315	-0.424124	0.122402	-0.122402	-1.000000	1.000000	-0.037476	0.213782	-0.045353	-0.281337	0.223986	-0.035940	0.216313	-0.099995
ar_vr_gaming	-0.034010	-0.168172	-0.127614	-0.321284	0.321284	0.037476	-0.037476	1.000000	-0.083811	-0.123860	-0.085403	-0.087812	-0.126774	-0.084803	-0.299119
healthcare	0.018408	-0.174050	-0.238384	0.260863	-0.260863	-0.213782	0.213782	-0.083811	1.000000	-0.087228	-0.060145	-0.061841	-0.089280	-0.059722	-0.210653
industry_4_0	0.045106	-0.216202	-0.284384	0.385515	-0.385515	0.045353	-0.045353	-0.123860	-0.087228	1.000000	-0.088885	-0.091391	-0.131942	-0.088261	-0.311313
io_t_devices	0.019458	0.389878	0.437355	0.265817	-0.265817	0.281337	-0.281337	-0.085403	-0.060145	-0.088885	1.000000	-0.063016	-0.090976	-0.060857	-0.214654
public_safety	0.019485	-0.182358	-0.249763	0.273314	-0.273314	-0.223986	0.223986	-0.087812	-0.061841	-0.091391	-0.063016	1.000000	-0.093542	-0.062573	-0.220708
smart_city_and_home	0.047347	0.578743	0.200818	0.394585	-0.394585	0.035940	-0.035940	-0.126774	-0.089280	-0.131942	-0.090976	-0.093542	1.000000	-0.090337	-0.318637
smart_transportation	0.019787	-0.176111	-0.241206	0.263951	-0.263951	-0.216313	0.216313	-0.084803	-0.059722	-0.088261	-0.060857	-0.062573	-0.090337	1.000000	-0.213147
smartphone	-0.075152	-0.067416	0.268831	-0.807524	0.807524	0.099995	-0.099995	-0.299119	-0.210653	-0.311313	-0.214654	-0.220708	-0.318637	-0.213147	1.000000

```
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')
```

PYCARET AUTOML

```
###!pip install pycaret
```

```
###! pip install jinja2
```

```
###! pip install markupsafe==2.0.1
```

```
import pycaret
```

```
from pycaret.classification import *
```

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

```
(18373, 18) (31584, 18)
```

```
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')
```

```
train_dataset2 = train_dataset[['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', 'slice_type']]
```

```
test_dataset2 = test_dataset[['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', 'slice_type']]
```

```
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')
```

```
model= setup(data= train_dataset2, target= 'slice_type')
```

	Description	Value
0	Session id	906
1	Target	slice_type
2	Target type	Multiclass
3	Target mapping	1: 0, 2: 1, 3: 2
4	Original data shape	(18373, 16)
5	Transformed data shape	(18373, 16)
6	Transformed train set shape	(12861, 16)
7	Transformed test set shape	(5512, 16)
8	Numeric features	15
9	Preprocess	True
10	Imputation type	simple
11	Numeric imputation	mean
12	Categorical imputation	mode
13	Fold Generator	StratifiedKFold
14	Fold Number	10
15	CPU Jobs	-1
16	Use GPU	False
17	Log Experiment	False
18	Experiment Name	clf-default-name
19	USI	8f69

```
compare_models()
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lr	Logistic Regression	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.9510
knn	K Neighbors Classifier	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.1240
nb	Naive Bayes	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.0350
dt	Decision Tree Classifier	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.0490
ridge	Ridge Classifier	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.0350
rf	Random Forest Classifier	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.2860
ada	Ada Boost Classifier	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.2580
gbc	Gradient Boosting Classifier	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.3470
et	Extra Trees Classifier	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.3230
xgboost	Extreme Gradient Boosting	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	0.2480
lightgbm	Light Gradient Boosting Machine	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0020
svm	SVM - Linear Kernel	0.9976	0.0000	0.9976	0.9977	0.9976	0.9960	0.9961	0.1660
lda	Linear Discriminant Analysis	0.8803	0.0000	0.8803	0.8996	0.8777	0.7904	0.8064	0.0420
qda	Quadratic Discriminant Analysis	0.5355	0.0000	0.5355	0.2868	0.3735	0.0000	0.0000	0.0350
dummy	Dummy Classifier	0.5355	0.0000	0.5355	0.2868	0.3735	0.0000	0.0000	0.0290

▾ LogisticRegression

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=906, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
logisticreg= create_model('lr')
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
3	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
4	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
5	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
6	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
7	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
8	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
9	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Mean	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Std	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

```
pred_holdout = predict_model(logisticreg, data= x_test2)
```

```
pred_holdout
```

	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	prediction_label	prediction_score
0	15	0.001000	100	0	1	1	0	1	0	0	0	0	0	0	0	1	0.9988
1	14	0.000001	10	1	0	0	1	0	0	0	0	0	0	1	0	3	0.9999
2	11	0.001000	50	1	0	1	0	0	0	1	0	0	0	0	0	2	0.9992
3	20	0.001000	50	1	0	1	0	0	0	1	0	0	0	0	0	2	0.9992
4	2	0.001000	50	0	1	0	1	1	0	0	0	0	0	0	0	1	0.9998
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
31579	9	0.000001	10	1	0	0	1	0	0	0	0	0	0	1	0	3	0.9998
31580	20	0.000001	10	1	0	0	1	0	0	1	0	0	0	0	0	3	1.0000
31581	8	0.000001	10	1	0	0	1	0	0	1	0	0	0	0	0	3	0.9993
31582	13	0.010000	75	0	1	0	1	0	0	0	0	0	0	0	1	1	0.9999
31583	8	0.000001	10	1	0	0	1	0	0	1	0	0	0	0	0	3	0.9993

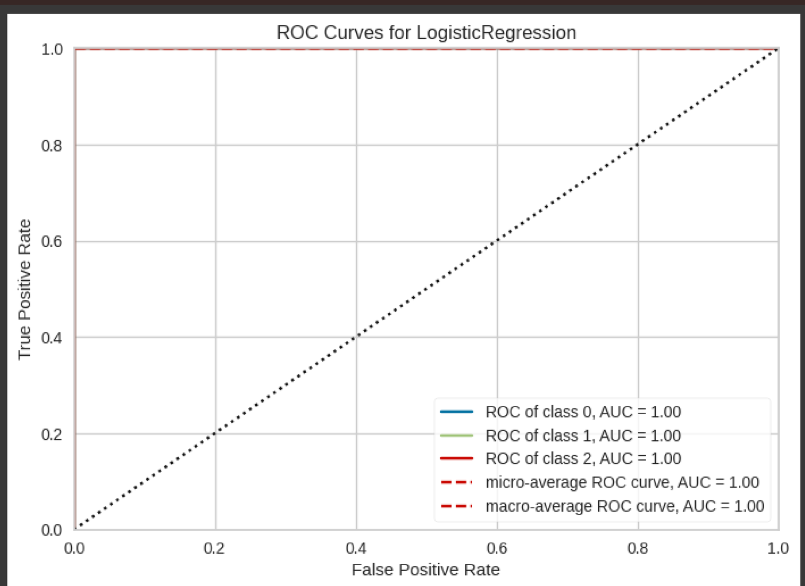
31584 rows × 17 columns

Next steps: Generate code with pred_holdout | View recommended plots

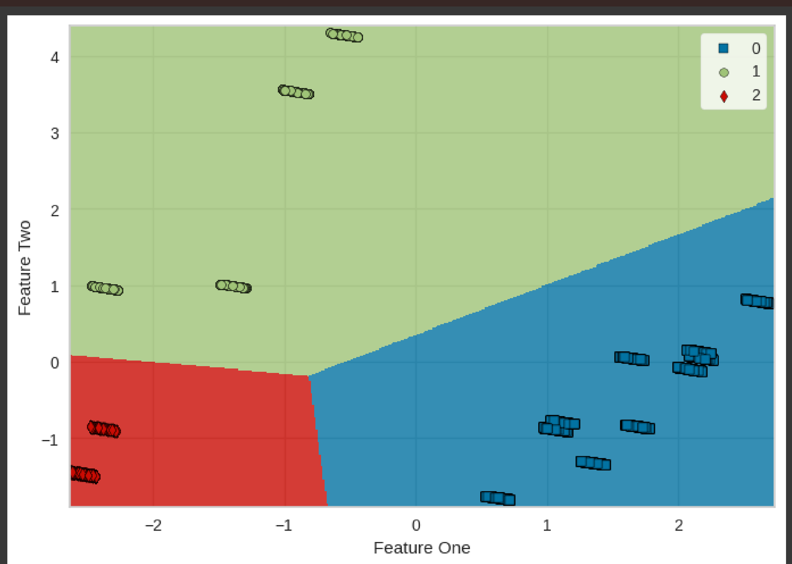
```
pred_holdout.columns
```

```
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', 'prediction_label',
       'prediction_score'],
      dtype='object')
```

```
# AUC-ROC plot
plot_model(logisticreg, plot = 'auc')
```




```
# Decision Boundary
plot_model(logisticreg, plot = 'boundary')
```




```
##! pip install shap
```

```
import shap
```

Logistic Regression

```
tune_logisticreg = tune_model(logisticreg)
```

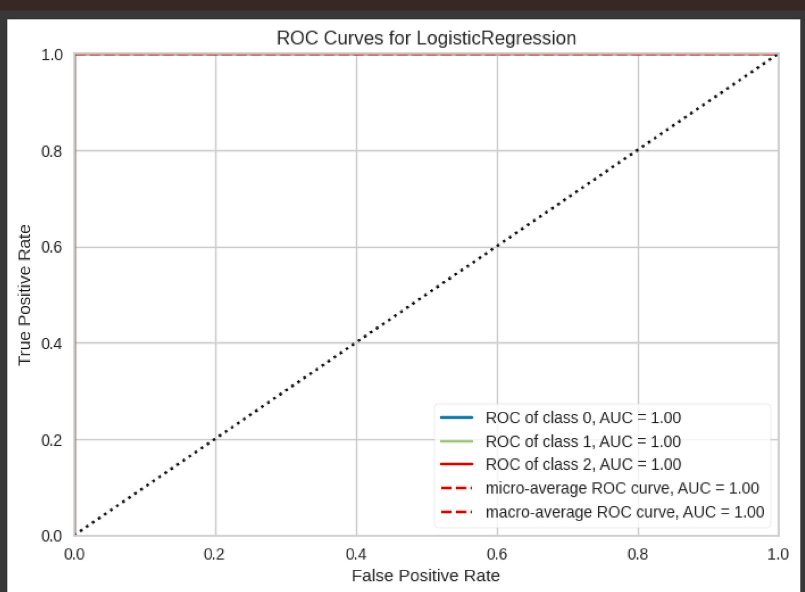
Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
2	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
3	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
4	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
5	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
6	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
7	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
8	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
9	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Mean	1.0000	0.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Std	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
Original model was better than the tuned model, hence it will be returned. NOTE: The display metrics are for the tuned model (not the original one).
```

```
print(tune_logisticreg)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=906, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
plot_model(tune_logisticreg)
```




```
predict_model(tune_logisticreg)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

	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	slice_type	prediction_label	prediction_score
23556	13	0.001000	100	0	1	1	0	0	0	0	0	0	0	0	1	1	1	0.9994
4680	4	0.010000	300	1	0	1	0	0	0	0	0	0	1	0	0	2	2	0.9997
3904	8	0.001000	100	0	1	1	0	1	0	0	0	0	0	0	0	1	1	0.9989
20321	18	0.010000	300	1	0	1	0	0	0	0	1	0	0	0	0	2	2	0.9990
4096	4	0.010000	75	0	1	0	1	0	0	0	0	0	0	0	1	1	1	0.9999
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29236	17	0.010000	75	0	1	0	1	0	0	0	0	0	0	0	1	1	1	0.9999
7150	16	0.001000	100	0	1	1	0	1	0	0	0	0	0	0	0	1	1	0.9988
10559	11	0.000001	300	0	1	1	0	0	0	0	0	0	0	0	1	1	1	0.9992
9212	11	0.000001	60	0	1	1	0	0	0	0	0	0	0	0	1	1	1	0.9995
13969	19	0.000001	300	0	1	1	0	0	0	0	0	0	0	0	1	1	1	0.9990

5512 rows × 18 columns

```
final_logisticreg = finalize_model(tune_logisticreg)
print(final_logisticreg)
```

```
Pipeline(memory=Memory(location=None),
         steps=[('label_encoding',
                 TransformerWrapperWithInverse(exclude=None, include=None,
                                               transformer=LabelEncoder())),
                ('numerical_imputer',
                 TransformerWrapper(exclude=None,
                                    include=['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...
                                                          fill_value=None,
                                                          keep_empty_features=False,
                                                          missing_values=nan,
                                                          strategy='most_frequent'))),
                ('actual_estimator',
                 LogisticRegression(C=1.0, class_weight=None, dual=False,
                                    fit_intercept=True, intercept_scaling=1,
                                    l1_ratio=None, max_iter=1000,
                                    multi_class='auto', n_jobs=None,
                                    penalty='l2', random_state=906,
                                    solver='lbfgs', tol=0.0001, verbose=0,
                                    warm_start=False))],
         verbose=False)
```

```
print(x_test2.columns, y_test.shape)
print(x_train2.shape, y_train.shape)
```

```
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') (31584, 1)
(18373, 15) (18373, 1)
```

```
final_logisticreg.fit(x_train2, y_train)
```

```
Pipeline
  label_encoding: TransformerWrapperWithInverse
    transformer: LabelEncoder
      LabelEncoder
  numerical_imputer: TransformerWrapper
    transformer: SimpleImputer
      SimpleImputer
  categorical_imputer: TransformerWrapper
    transformer: SimpleImputer
      SimpleImputer
  LogisticRegression
```

```
y_pred = final_logisticreg.predict(x_test2)
```

```
y_pred = pd.DataFrame(y_pred)
```

```
y_pred.shape
```

```
(31584, 1)
```

```
test_data2 = pd.read_csv("/content/test_dataset.csv")
```

```
test_data2.shape
```

```
(31584, 16)
```

```
resultdetails = pd.concat([test_data2, y_pred], axis=1, join="inner")
```

```
resultdetails.shape
```

```
(31584, 17)
```

```
resultdetails['target'].value_counts()
```

```
1    16800
3     7392
2     7392
Name: target, dtype: int64
```

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

```
y_pred.value_counts()
```

```
target
1         16800
2          7392
3          7392
dtype: int64
```

```
resultdetails.to_csv("/content/final.csv")
```

Start coding or generate with AI.