```
### 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.shape
```

```
(31583, 18)
```

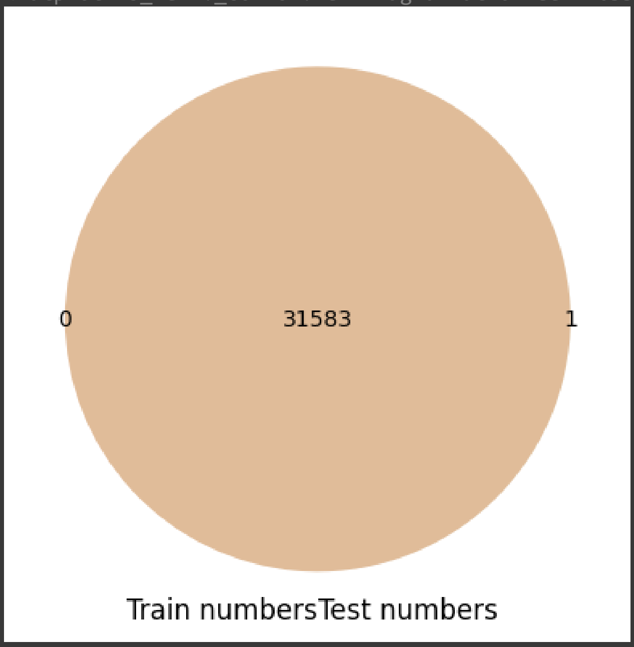
```
train_dataset['slice Type'].value_counts()
```

```
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 0x7ee7910554b0>




```
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
```

```
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')
```

```
train_dataset = normal
```

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

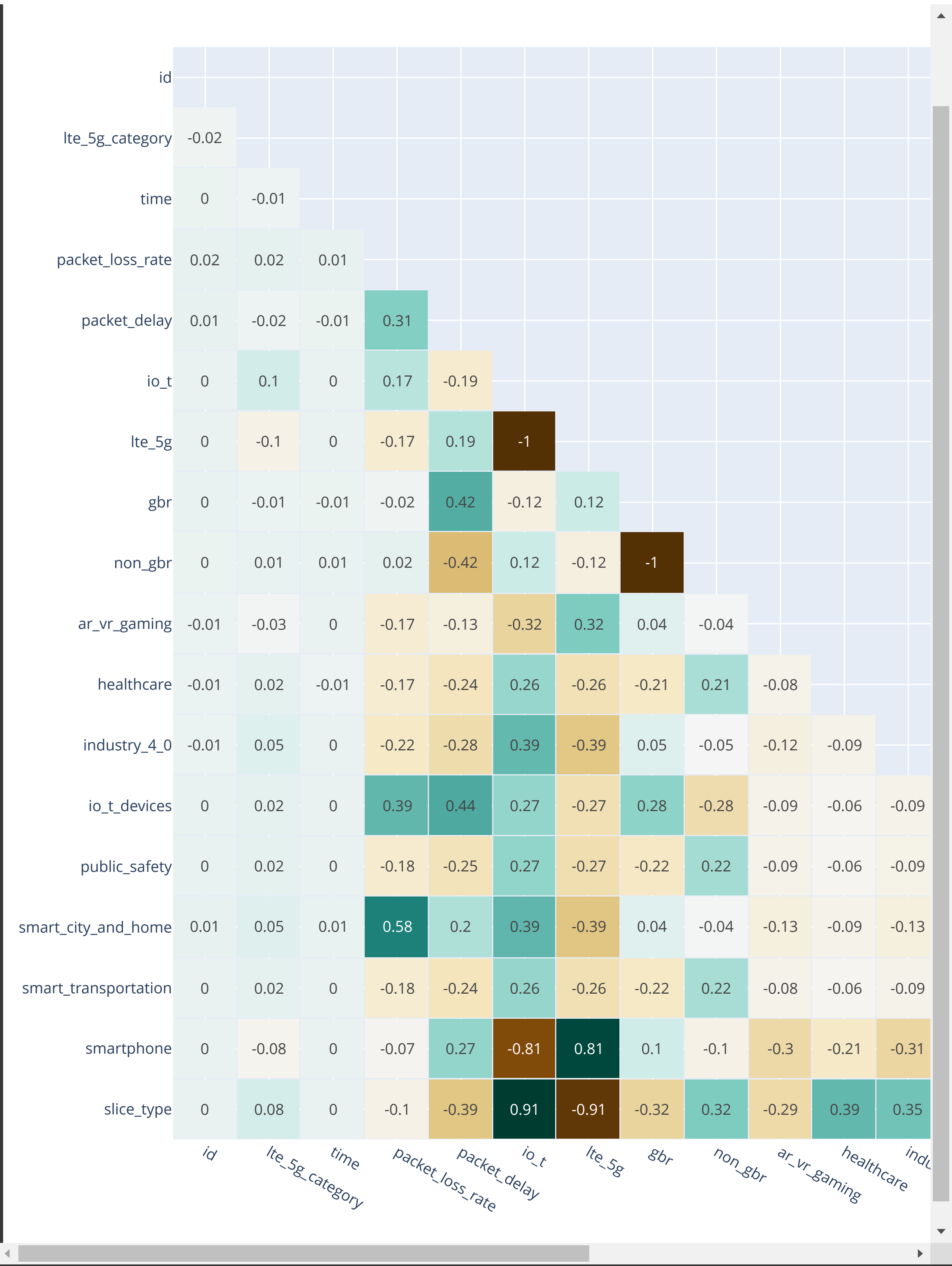
```
(18373, 18)
Index(['id', 'lte_5g_category', 'time', 'packet_loss_rate', 'packet_delay',
       'io_t', 'lte_5g', 'gbr', 'non_gbr', 'ar_vr_gaming', 'healthcare',
       'industry_4_0', 'io_t_devices', 'public_safety', 'smart_city_and_home',
       'smart_transportation', 'smartphone', 'slice_type'],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

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

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

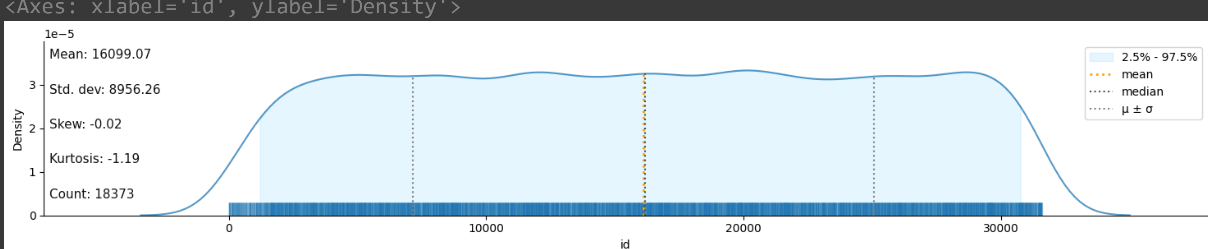
```
No columns with categorical data were detected.
```

```
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 bas
<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
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	0.2
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
```

```
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')
```

```
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)
```

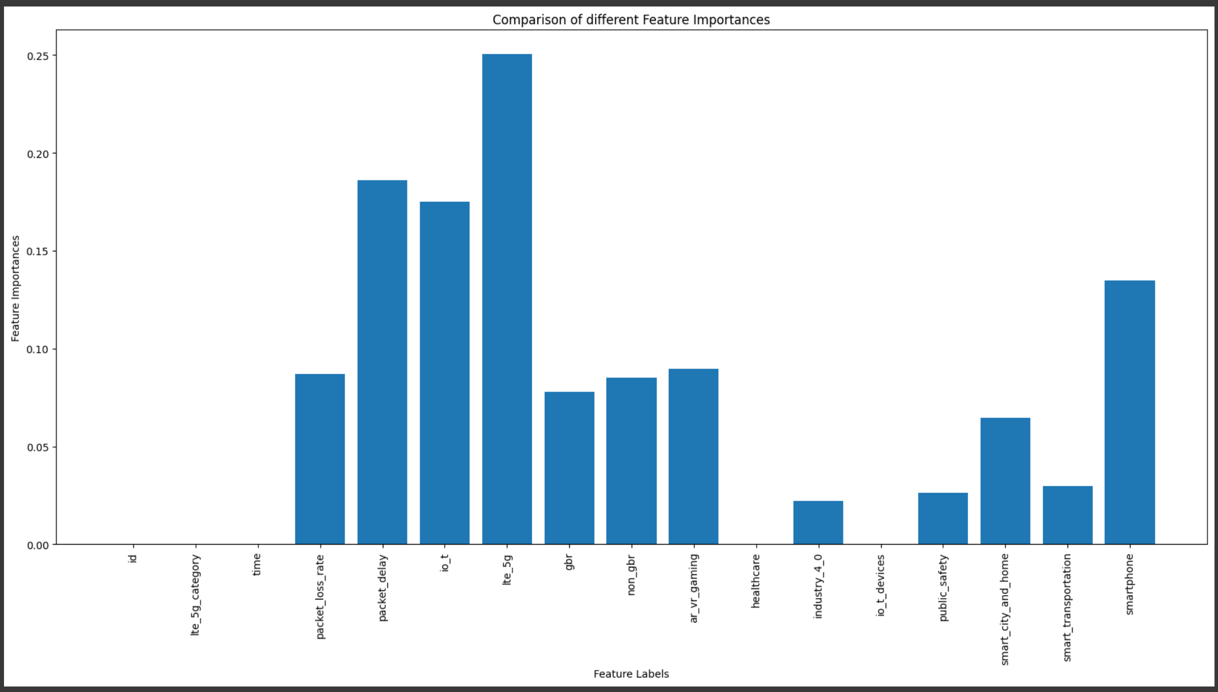
```
from sklearn.ensemble import ExtraTreesClassifier
extra_tree_forest = ExtraTreesClassifier(n_estimators = 5,
                                        criterion ='entropy', max_features = 2)
extra_tree_forest.fit(x_train, y_train)
feature_importance = extra_tree_forest.feature_importances_
feature_importance_normalized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

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

```
feature_importance_normalized
```

```
array([0.00000000e+00, 0.00000000e+00, 1.72660354e-05, 8.70177097e-02,
       1.85906606e-01, 1.75026484e-01, 2.50351600e-01, 7.80310584e-02,
       8.50223699e-02, 8.97766063e-02, 0.00000000e+00, 2.20555329e-02,
       0.00000000e+00, 2.62057558e-02, 6.46563362e-02, 2.96450142e-02,
       1.34951710e-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_train2.head(4)
```

	lte_5g_category	packet_loss_rate	packet_delay	io_t	lte_5g	gbr	non_gbr	ar_vr_gaming	healt
0	14	0.000001	10	1	0	0	1	0	
4	9	0.010000	50	1	0	0	1	0	
5	19	0.000001	10	1	0	0	1	0	
8	8	0.001000	150	0	1	0	1	0	

Next steps: Generate code with x_train2 View recommended plots

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

```
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(x_train2.columns, x_train2.shape)
print(x_test2.columns, x_test2.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') (18373, 15)
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, 15)
```

Standard Scaler

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
x_train2=pd.DataFrame(scaler.fit_transform(x_train2),columns=x_train2.columns)
x_train2.head()
```

	lte_5g_category	packet_loss_rate	packet_delay	io_t	lte_5g	gbr	non_gbr	ar_vr_ga
0	0.496569	-0.716409	-0.981216	1.073740	-1.073740	-0.87995	0.87995	-0.34
1	-0.319527	1.574868	-0.602200	1.073740	-1.073740	-0.87995	0.87995	-0.34
2	1.312664	-0.716409	-0.981216	1.073740	-1.073740	-0.87995	0.87995	-0.34
3	-0.482746	-0.487488	0.345339	-0.931324	0.931324	-0.87995	0.87995	-0.34
4	-0.645965	-0.716409	-0.981216	1.073740	-1.073740	-0.87995	0.87995	-0.34

Next steps: Generate code with x_train2 View recommended plots

```
x_test2=pd.DataFrame(scaler.fit_transform(x_test2),columns=x_test2.columns)
x_test2.head()
```

	lte_5g_category	packet_loss_rate	packet_delay	io_t	lte_5g	gbr	non_gbr	ar_vr_
0	0.664781	-0.482940	-0.136290	-0.938083	0.938083	1.124027	-1.124027	2.
1	0.500168	-0.712444	-0.982767	1.066004	-1.066004	-0.889658	0.889658	-0.
2	0.006327	-0.482940	-0.606555	1.066004	-1.066004	1.124027	-1.124027	-0.
3	1.487848	-0.482940	-0.606555	1.066004	-1.066004	1.124027	-1.124027	-0.
4	-1.475194	-0.482940	-0.606555	-0.938083	0.938083	-0.889658	0.889658	2.

Next steps: Generate code with x_test2 View recommended plots

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

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

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

```
y_train.columns
```

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

```
y_train = pd.get_dummies(y_train.slice_type)
y_test = pd.get_dummies(y_test.slice_type)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dropout, Bidirectional, Embedding, Dense
from keras import callbacks
```

```
#Early stopping
early_stopping = callbacks.EarlyStopping(
    min_delta=0.001, # minimium amount of change to count as an improvement
    patience=20, # how many epochs to wait before stopping
    restore_best_weights=True,
)
```

```
from keras.optimizers import Adam
```

```
# Initialising the NN
model = Sequential()

# layers

model.add(Dense(units = 32, kernel_initializer = 'uniform', activation = 'relu', input_dim = 15))
model.add(Dense(units = 32, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dense(units = 32, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dropout(0.25))
model.add(Dense(units = 8, kernel_initializer = 'uniform', activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(units = 3, kernel_initializer = 'uniform', activation = 'sigmoid'))

# Compiling the ANN
opt = Adam(learning_rate=0.00009)
model.compile(optimizer = opt, loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
history = model.fit(x_train2, y_train, batch_size = 64, epochs = 150, callbacks=[early_stopping], validation_split=0.2)
```

```
230/230 [==============================] - 1s 3ms/step - loss: 0.1641 - accuracy: 0.8778 - val_loss: 0.0690 - val_accurac
Epoch 115/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1615 - accuracy: 0.8807 - val_loss: 0.0683 - val_accurac
Epoch 116/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1609 - accuracy: 0.8811 - val_loss: 0.0684 - val_accurac
Epoch 117/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1631 - accuracy: 0.8782 - val_loss: 0.0682 - val_accurac
Epoch 118/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1629 - accuracy: 0.8790 - val_loss: 0.0684 - val_accurac
Epoch 119/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1614 - accuracy: 0.8807 - val_loss: 0.0686 - val_accurac
Epoch 120/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1631 - accuracy: 0.8781 - val_loss: 0.0679 - val_accurac
Epoch 121/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1600 - accuracy: 0.8826 - val_loss: 0.0681 - val_accurac
Epoch 122/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1604 - accuracy: 0.8813 - val_loss: 0.0674 - val_accurac
Epoch 123/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1611 - accuracy: 0.8803 - val_loss: 0.0676 - val_accurac
Epoch 124/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1624 - accuracy: 0.8779 - val_loss: 0.0674 - val_accurac
Epoch 125/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1626 - accuracy: 0.8782 - val_loss: 0.0679 - val_accurac
Epoch 126/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1603 - accuracy: 0.8811 - val_loss: 0.0680 - val_accurac
Epoch 127/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1623 - accuracy: 0.8781 - val_loss: 0.0676 - val_accurac
Epoch 128/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1620 - accuracy: 0.8788 - val_loss: 0.0675 - val_accurac
Epoch 129/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1635 - accuracy: 0.8764 - val_loss: 0.0679 - val_accurac
Epoch 130/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1598 - accuracy: 0.8808 - val_loss: 0.0677 - val_accurac
Epoch 131/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1608 - accuracy: 0.8796 - val_loss: 0.0677 - val_accurac
Epoch 132/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1599 - accuracy: 0.8809 - val_loss: 0.0674 - val_accurac
Epoch 133/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1623 - accuracy: 0.8776 - val_loss: 0.0671 - val_accurac
Epoch 134/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1622 - accuracy: 0.8772 - val_loss: 0.0677 - val_accurac
Epoch 135/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1563 - accuracy: 0.8842 - val_loss: 0.0675 - val_accurac
Epoch 136/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1592 - accuracy: 0.8813 - val_loss: 0.0677 - val_accurac
Epoch 137/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1611 - accuracy: 0.8785 - val_loss: 0.0670 - val_accurac
Epoch 138/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1599 - accuracy: 0.8792 - val_loss: 0.0677 - val_accurac
Epoch 139/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1603 - accuracy: 0.8788 - val_loss: 0.0672 - val_accurac
Epoch 140/150
230/230 [==============================] - 0s 2ms/step - loss: 0.1582 - accuracy: 0.8817 - val_loss: 0.0674 - val_accurac
Epoch 141/150
230/230 [==============================] - 1s 3ms/step - loss: 0.1589 - accuracy: 0.8814 - val_loss: 0.0670 - val_accurac
Epoch 142/150
230/230 [==============================] - 1s 2ms/step - loss: 0.1627 - accuracy: 0.8760 - val_loss: 0.0677 - val_accurac
```

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

```
987/987 [==============================] - 1s 868us/step
```

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

```
new_y_pred = y_pred.idxmax(axis=1)
```

```
new_y_test = y_test.idxmax(axis=1)
```

```
new_y_pred = pd.DataFrame(new_y_pred)
new_y_test = pd.DataFrame(new_y_test)
```

```
new_y_pred.value_counts()
```

```
0    16800
1     7392
2     7392
dtype: int64
```

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

```
new_y_train= y_train.idxmax(axis=1)
```

```
new_y_train.value_counts()
```

```
1    9839
2    4294
3    4240
dtype: int64
```

```
new_y_pred = pd.DataFrame(new_y_pred)
```

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

```
new_y_pred2 = new_y_pred
```

```
##
output = {
    0: '1',
    1: '2',
    2: '3'
}
```

```
new_y_pred["Predict"] = new_y_pred["Predict"].map(output)
```

```
new_y_pred.value_counts()
```

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

```
labels = new_y_pred['Predict'].astype('category').cat.categories.tolist()
counts = new_y_pred['Predict'].value_counts()
sizes = [counts[var_cat] for var_cat in labels]
fig1, ax1 = plt.subplots()
plt.title("Categorical Pedictions")
ax1.pie(sizes, labels=labels, autopct='%1.1f%%', shadow=True) #autopct is show the % on plot
ax1.axis('equal')
plt.show()
```