Part A - 30 Marks

- **DOMAIN:** Electronics and Telecommunication [TOTAL SCORE60]
- **CONTEXT:** A communications equipment manufacturing company has a product which is responsible for emitting informative signals. Company wants to build a machine learning model which can help the company to predict the equipment's signal quality using various parameters.
- **DATA DESCRIPTION:** The data set contains information on various signal tests performed:

```
1. Parameters: Various measurable signal parameters.
2. Signal_Quality: Final signal strength or quality
```

- **PROJECT OBJECTIVE:** To build a classifier which can use the given parameters to determine the signal strength or quality. Steps and tasks: [Total Score: 30 Marks]
- **1.Data import and Understanding [10 Marks]**

```
A. Read the 'Signals.csv' as DataFrame and import required libraries. [2 Marks]
B.Check for missing values and print percentage for each attribute. [2 Marks]
C.Check for presence of duplicate records in the dataset and impute with appropriate method. [2 Marks]
D.Visualise distribution of the target variable. [2 Marks]
E.Share insights from the initial data analysis (at least 2). [2 Marks]
```

- **2. Data preprocessing [7 Marks]**

```
A. Split the data into X & Y. [1 Marks]
B. Split the data into train & test with 70:30 proportion.[1 Marks]
C. Print shape of all the 4 variables and verify if train and test data is in sync. [1 Marks]
D. Normalise the train and test data with appropriate method. [2 Marks]
E. Transform Labels into format acceptable by Neural Network [2 Marks]
```

- **3. Model Training & Evaluation using Neural Network [13 Marks]**

```
A. Design a Neural Network to train a classifier. [3 Marks]
B. Train the classifier using previously designed Architecture [2 Marks]
C. Plot 2 separate visuals. [3 Marks]
i.Training Loss and Validation Loss
```

ii.Training Accuracy and Validation Accuracy
D. Design new architecture/update existing architecture in attempt to improve the performance of the model.[2 Marks]
E. Plot visuals as in Q3.C and share insights about difference observed in both the models. [3 Marks]

In [211…
```
# importing important library for numerical,dataframe, data statisticalanalysis, data visualisation
# https://chatgpt.com/share/f5ef591c-922f-4cd3-a46a-af7821357b29
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [212…
```
# for data preprocessing, data valuation, data training and testing

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
```

In [213…
```
import sys
!{sys.executable} -m pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\chirag\anaconda3\lib\site-packages (0.12.3)
Requirement already satisfied: numpy>=1.17.3 in c:\users\chirag\anaconda3\lib\site-packages (from imbalanced-learn) (1.23.5)
Requirement already satisfied: scipy>=1.5.0 in c:\users\chirag\anaconda3\lib\site-packages (from imbalanced-learn) (1.11.1)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\chirag\anaconda3\lib\site-packages (from imbalanced-learn) (1.5.
0)
Requirement already satisfied: joblib>=1.1.1 in c:\users\chirag\anaconda3\lib\site-packages (from imbalanced-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\chirag\anaconda3\lib\site-packages (from imbalanced-learn) (3.5.
0)
```

In [214…
```
from imblearn.over_sampling import SMOTE
```

In [215…
```
# deep learning lebraryes
import keras
import tensorflow as tf
from tensorflow.keras.layers import Dense,InputLayer
```

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import BatchNormalization, Dropout
from tensorflow.keras import regularizers, optimizers
```

In [216…

```
# for wrappers
from scikeras.wrappers import KerasRegressor
from scikeras.wrappers import KerasClassifier
```

1.Data import and Understanding [10 Marks]

1 A. Read the 'Signals.csv' as DataFrame and import required libraries. [2 Marks]

In [217…

```
DF = pd.read_csv("E:\\Greatlearning- Projects\\AIML project - ANN\\NN Project Data - Signal.csv")
DF.head()
```

Out[217]:

	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter 5	Parameter 6	Parameter 7	Parameter 8	Parameter 9	Parameter 10	Parameter 11	Signal_Strength
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

In [220…

```
DF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   Parameter 1      1599 non-null   float64
 1   Parameter 2      1599 non-null   float64
 2   Parameter 3      1599 non-null   float64
 3   Parameter 4      1599 non-null   float64
 4   Parameter 5      1599 non-null   float64
 5   Parameter 6      1599 non-null   float64
 6   Parameter 7      1599 non-null   float64
 7   Parameter 8      1599 non-null   float64
 8   Parameter 9      1599 non-null   float64
 9   Parameter 10     1599 non-null   float64
 10  Parameter 11     1599 non-null   float64
 11  Signal_Strength  1599 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [221…

```
DF.describe().T
```

Out[221]:

	count	mean	std	min	25%	50%	75%	max
Parameter 1	1599.0	8.319637	1.741096	4.60000	7.1000	7.90000	9.200000	15.90000
Parameter 2	1599.0	0.527821	0.179060	0.12000	0.3900	0.52000	0.640000	1.58000
Parameter 3	1599.0	0.270976	0.194801	0.00000	0.0900	0.26000	0.420000	1.00000
Parameter 4	1599.0	2.538806	1.409928	0.90000	1.9000	2.20000	2.600000	15.50000
Parameter 5	1599.0	0.087467	0.047065	0.01200	0.0700	0.07900	0.090000	0.61100
Parameter 6	1599.0	15.874922	10.460157	1.00000	7.0000	14.00000	21.000000	72.00000
Parameter 7	1599.0	46.467792	32.895324	6.00000	22.0000	38.00000	62.000000	289.00000
Parameter 8	1599.0	0.996747	0.001887	0.99007	0.9956	0.99675	0.997835	1.00369
Parameter 9	1599.0	3.311113	0.154386	2.74000	3.2100	3.31000	3.400000	4.01000
Parameter 10	1599.0	0.658149	0.169507	0.33000	0.5500	0.62000	0.730000	2.00000
Parameter 11	1599.0	10.422983	1.065668	8.40000	9.5000	10.20000	11.100000	14.90000
Signal_Strength	1599.0	5.636023	0.807569	3.00000	5.0000	6.00000	6.000000	8.00000

1 B.Check for missing values and print percentage for each attribute. [2 Marks]

In [222…

```
List1=list(DF.columns)

for i in List1:
   print("the percen of null values in colum", i," = ", (DF[i].isnull().sum()/DF[i].count())*100)
```

```
the percen of null values in colum Parameter 1  =  0.0
the percen of null values in colum Parameter 2  =  0.0
the percen of null values in colum Parameter 3  =  0.0
the percen of null values in colum Parameter 4  =  0.0
the percen of null values in colum Parameter 5  =  0.0
the percen of null values in colum Parameter 6  =  0.0
the percen of null values in colum Parameter 7  =  0.0
the percen of null values in colum Parameter 8  =  0.0
the percen of null values in colum Parameter 9  =  0.0
the percen of null values in colum Parameter 10  =  0.0
the percen of null values in colum Parameter 11  =  0.0
the percen of null values in colum Signal_Strength  =  0.0
```

In [223…

```
missing_values = df.isnull().sum()
missing_percentage = (missing_values / len(df)) * 100
missing_percentage
```

Out[223]:

```
Parameter 1        0.0
Parameter 2        0.0
Parameter 3        0.0
Parameter 4        0.0
Parameter 5        0.0
Parameter 6        0.0
Parameter 7        0.0
Parameter 8        0.0
Parameter 9        0.0
Parameter 10       0.0
Parameter 11       0.0
Signal_Strength    0.0
dtype: float64
```

In [224…

```
DF.isnull().any().any()
```

Out[224]:

```
False
```

so there are no missing values

1 C.Check for presence of duplicate records in the dataset and impute with appropriate method. [2 Marks]

In [225…
```
DF.duplicated().sum()
```

Out[225]:
```
240
```

In [226…
```
DF[DF.duplicated()]
```

Out[226]:

	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter 5	Parameter 6	Parameter 7	Parameter 8	Parameter 9	Parameter 10	Parameter 11	Signal_Strength
4	7.4	0.700	0.00	1.90	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
11	7.5	0.500	0.36	6.10	0.071	17.0	102.0	0.99780	3.35	0.80	10.5	5
27	7.9	0.430	0.21	1.60	0.106	10.0	37.0	0.99660	3.17	0.91	9.5	5
40	7.3	0.450	0.36	5.90	0.074	12.0	87.0	0.99780	3.33	0.83	10.5	5
65	7.2	0.725	0.05	4.65	0.086	4.0	11.0	0.99620	3.41	0.39	10.9	5
…	…	…	…	…	…	…	…	…	…	…	…	…
1563	7.2	0.695	0.13	2.00	0.076	12.0	20.0	0.99546	3.29	0.54	10.1	5
1564	7.2	0.695	0.13	2.00	0.076	12.0	20.0	0.99546	3.29	0.54	10.1	5
1567	7.2	0.695	0.13	2.00	0.076	12.0	20.0	0.99546	3.29	0.54	10.1	5
1581	6.2	0.560	0.09	1.70	0.053	24.0	32.0	0.99402	3.54	0.60	11.3	5
1596	6.3	0.510	0.13	2.30	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6

240 rows × 12 columns

- **there are 240 duplicated value inorder to save during training our model and reducung complexity we are just dropin it**
- **before droping lets create copy of data set**

In [227…
```
df=DF.copy()
df.drop_duplicates(inplace=True)
```

In [228…
```
df.duplicated().any()
```

Out[228]: False

In [229…
```
## we have removed succefully duplicated raws (data)
```

In [230…
```
df.head()
```

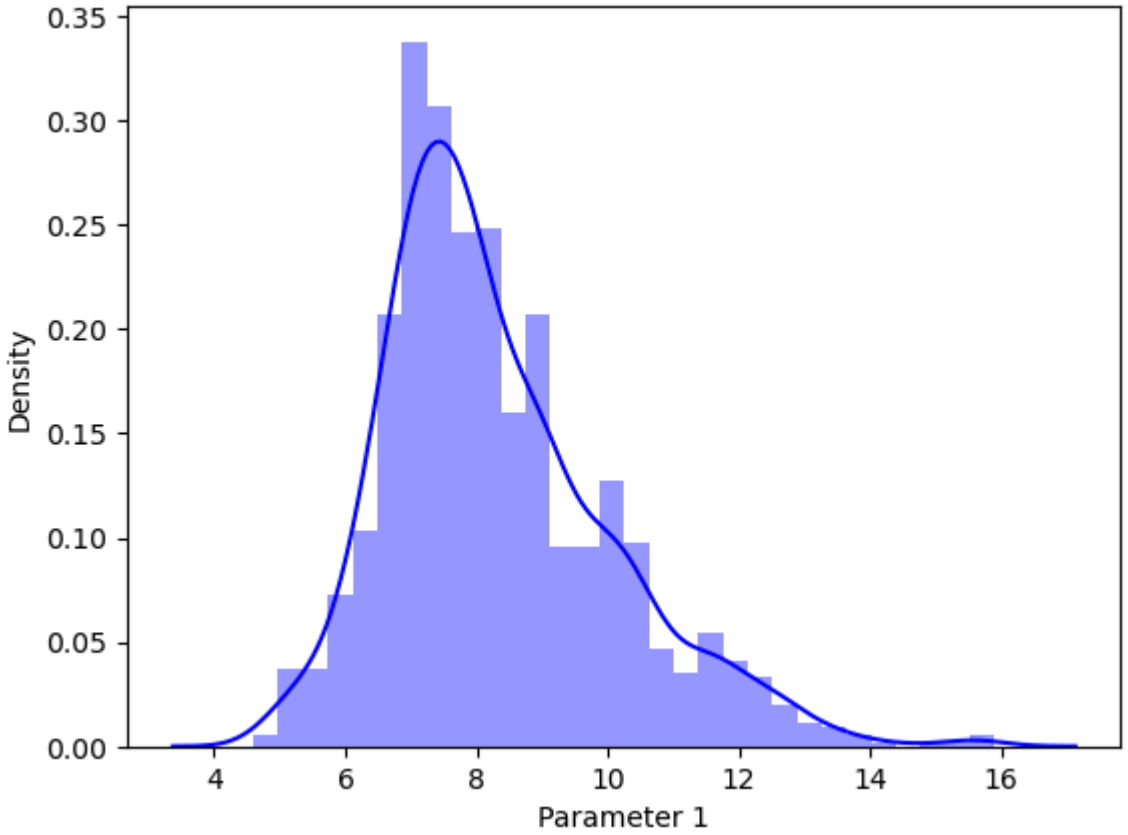
Out[230]:

	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter 5	Parameter 6	Parameter 7	Parameter 8	Parameter 9	Parameter 10	Parameter 11	Signal_Strength
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5

1 D.Visualise distribution of the target variable. [2 Marks]

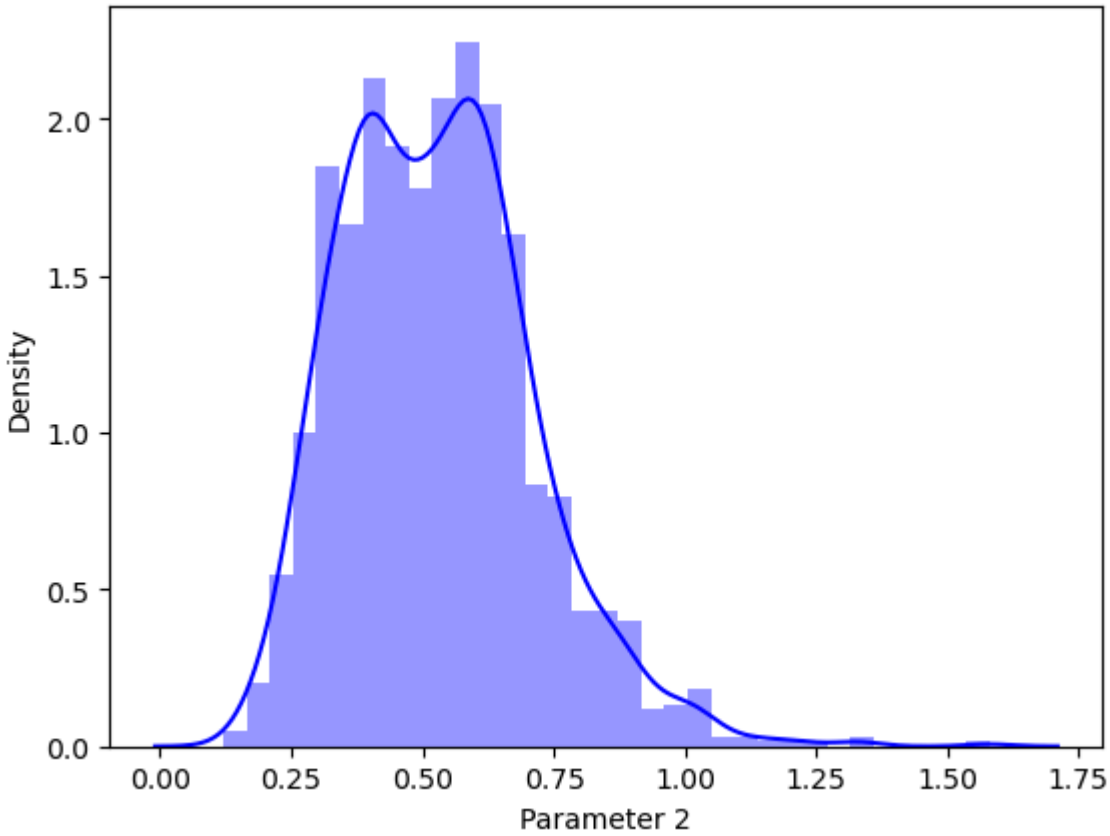
In [231…
```
cols = list(df)
for i in np.arange(len(cols)):
    sns.distplot(df[cols[i]], color='blue')
    #plt.xlabel('Experience')
    plt.show()
    print('Distribution of ',cols[i])
    print('Mean is:',df[cols[i]].mean())
    print('Median is:',df[cols[i]].median())
    print('Mode is:',df[cols[i]].mode())
    print('Standard deviation is:',df[cols[i]].std())
    print('Skewness is:',df[cols[i]].skew())
    print('Maximum is:',df[cols[i]].max())
    print('Minimum is:',df[cols[i]].min())
```

```
C:\Users\CHIRAG\AppData\Local\Temp\ipykernel_13804\2113990952.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[cols[i]], color='blue')
```



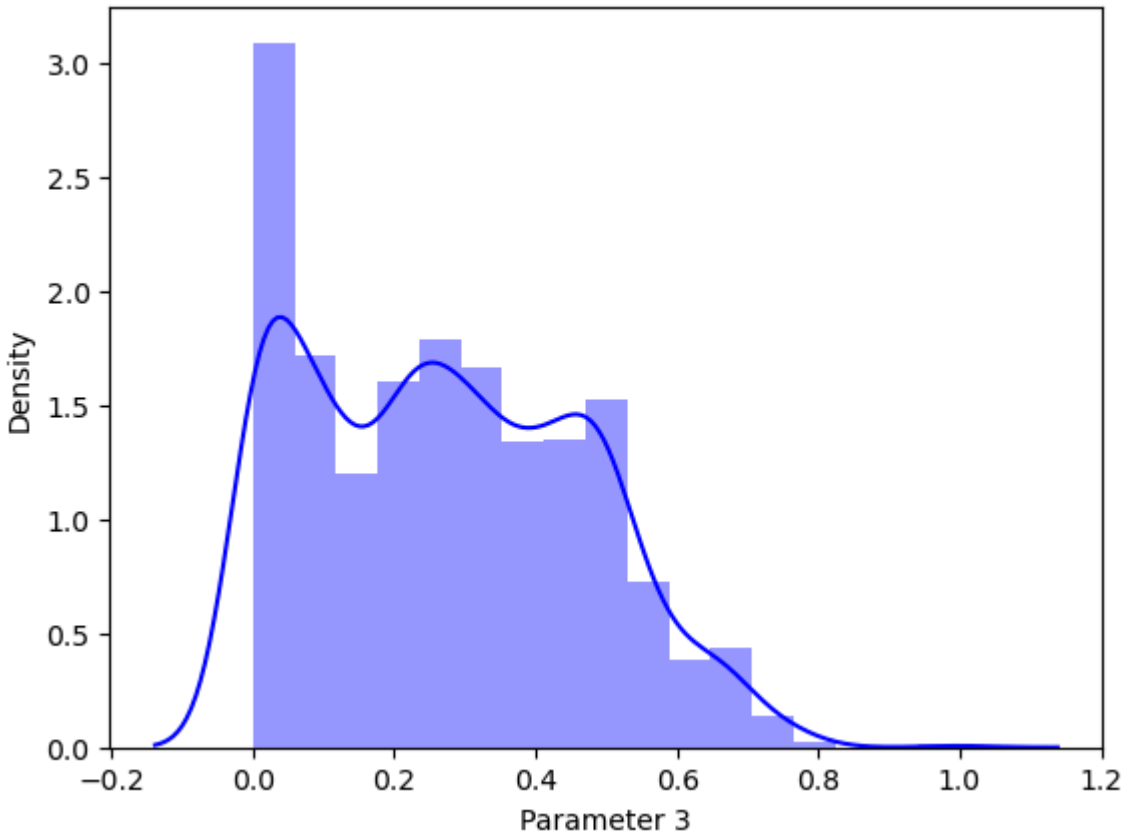
```
C:\Users\CHIRAG\AppData\Local\Temp\ipykernel_13804\2113990952.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[cols[i]], color='blue')
```

```
Distribution of  Parameter 1
Mean is: 8.310596026490067
Median is: 7.9
Mode is: 0    7.2
Name: Parameter 1, dtype: float64
Standard deviation is: 1.736989807532466
Skewness is: 0.9410413664561449
Maximum is: 15.9
Minimum is: 4.6
```



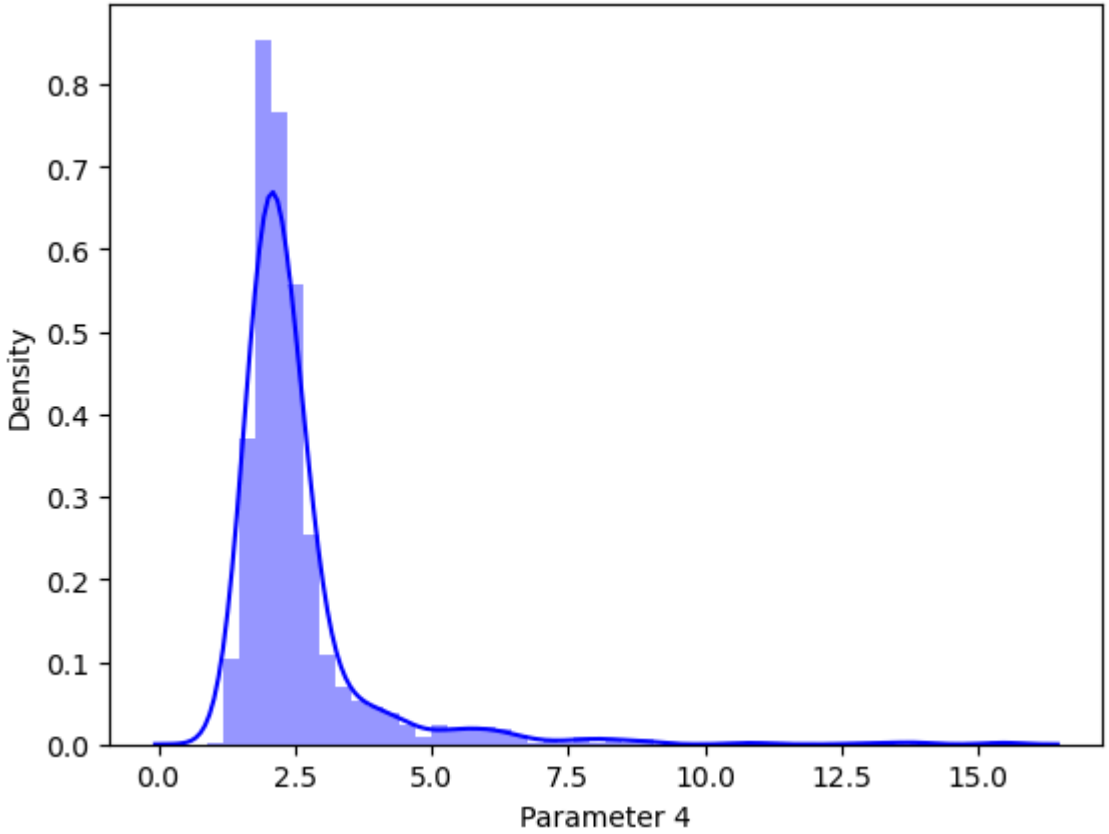

```
Distribution of  Parameter 2
Mean is: 0.5294775570272259
Median is: 0.52
Mode is: 0    0.5
Name: Parameter 2, dtype: float64
Standard deviation is: 0.18303131761907185
Skewness is: 0.7292789463991854
Maximum is: 1.58
Minimum is: 0.12
```

```
C:\Users\CHIRAG\AppData\Local\Temp\ipykernel_13804\2113990952.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[cols[i]], color='blue')
```



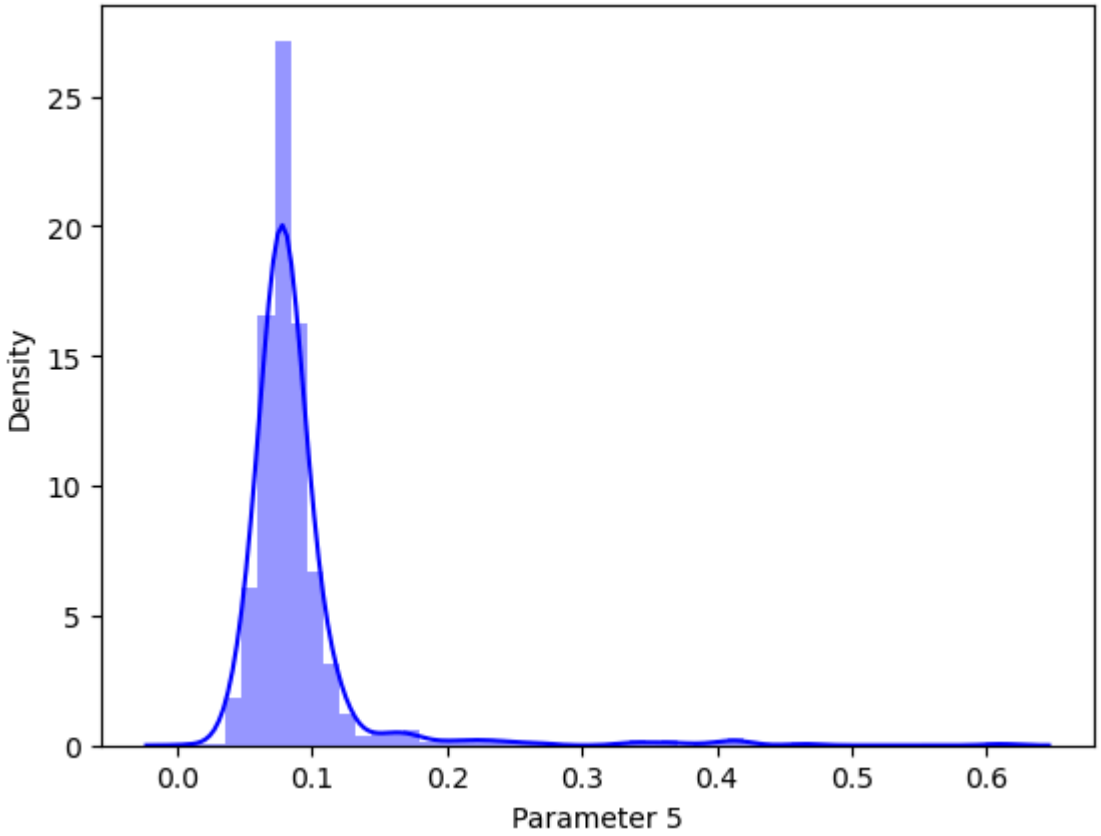
```
Distribution of  Parameter 3
Mean is: 0.2723325974981604
Median is: 0.26
Mode is: 0    0.0
Name: Parameter 3, dtype: float64
Standard deviation is: 0.1955365445504639
Skewness is: 0.31272554238899036
Maximum is: 1.0
Minimum is: 0.0
```

```
C:\Users\CHIRAG\AppData\Local\Temp\ipykernel_13804\2113990952.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[cols[i]], color='blue')
```

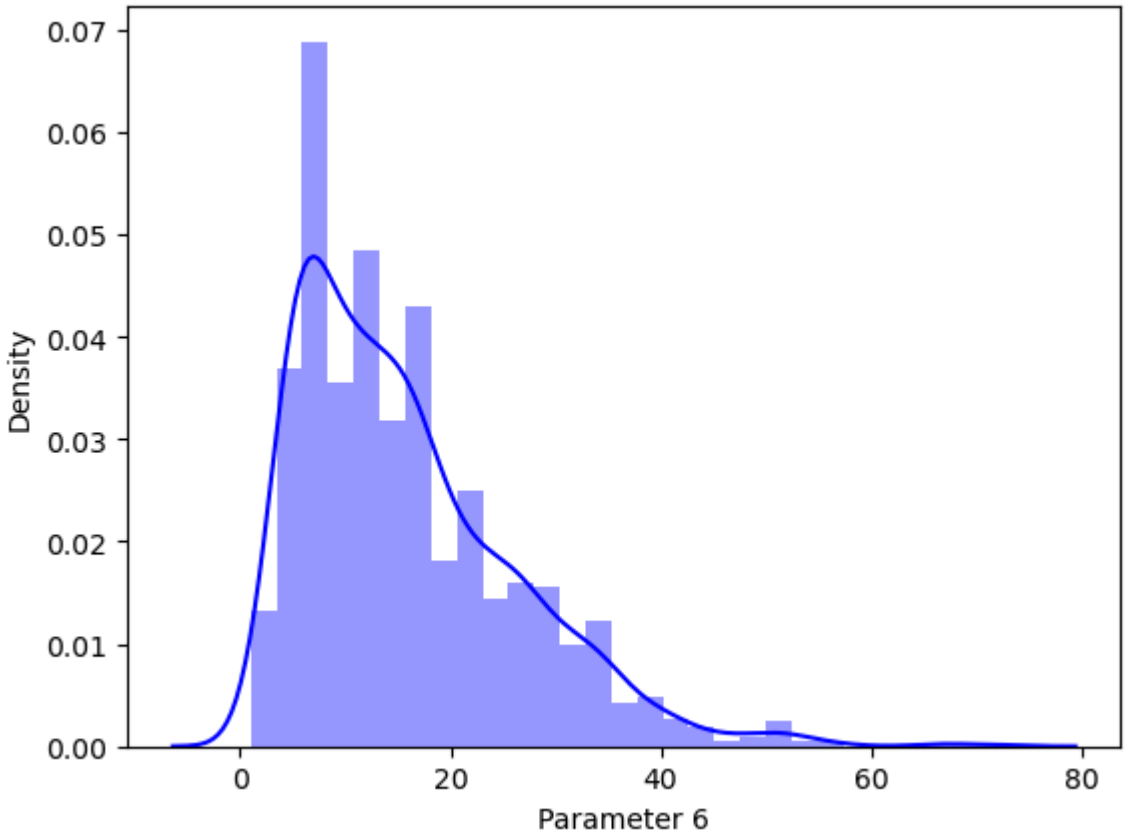



```
Distribution of  Parameter 4
Mean is: 2.5233995584988964
Median is: 2.2
Mode is: 0    2.0
Name: Parameter 4, dtype: float64
Standard deviation is: 1.3523137577104198
Skewness is: 4.548153403940447
Maximum is: 15.5
Minimum is: 0.9
```

```
C:\Users\CHIRAG\AppData\Local\Temp\ipykernel_13804\2113990952.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[cols[i]], color='blue')
```

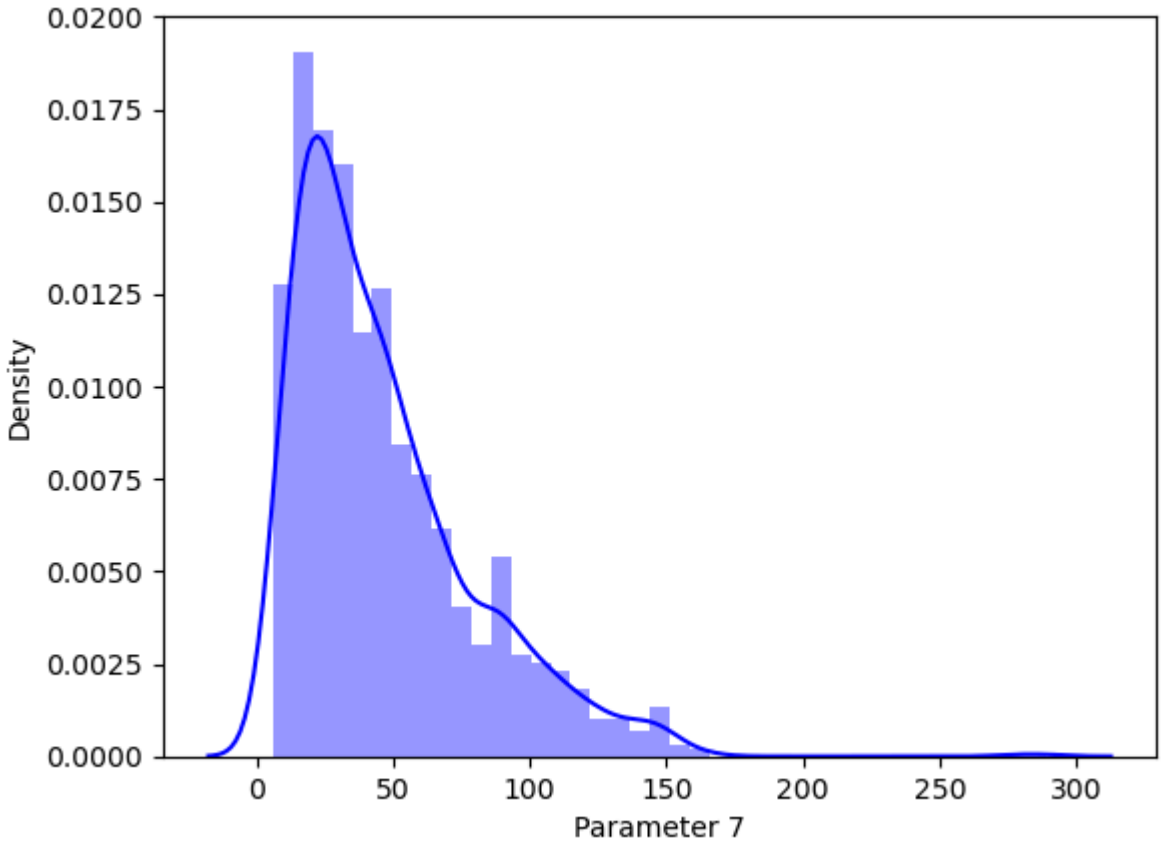


```
Distribution of  Parameter 5
Mean is: 0.08812362030905077
Median is: 0.079
Mode is: 0    0.08
Name: Parameter 5, dtype: float64
Standard deviation is: 0.04937686244348626
Skewness is: 5.502487294623722
Maximum is: 0.611
Minimum is: 0.012
C:\Users\CHIRAG\AppData\Local\Temp\ipykernel_13804\2113990952.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[cols[i]], color='blue')
```

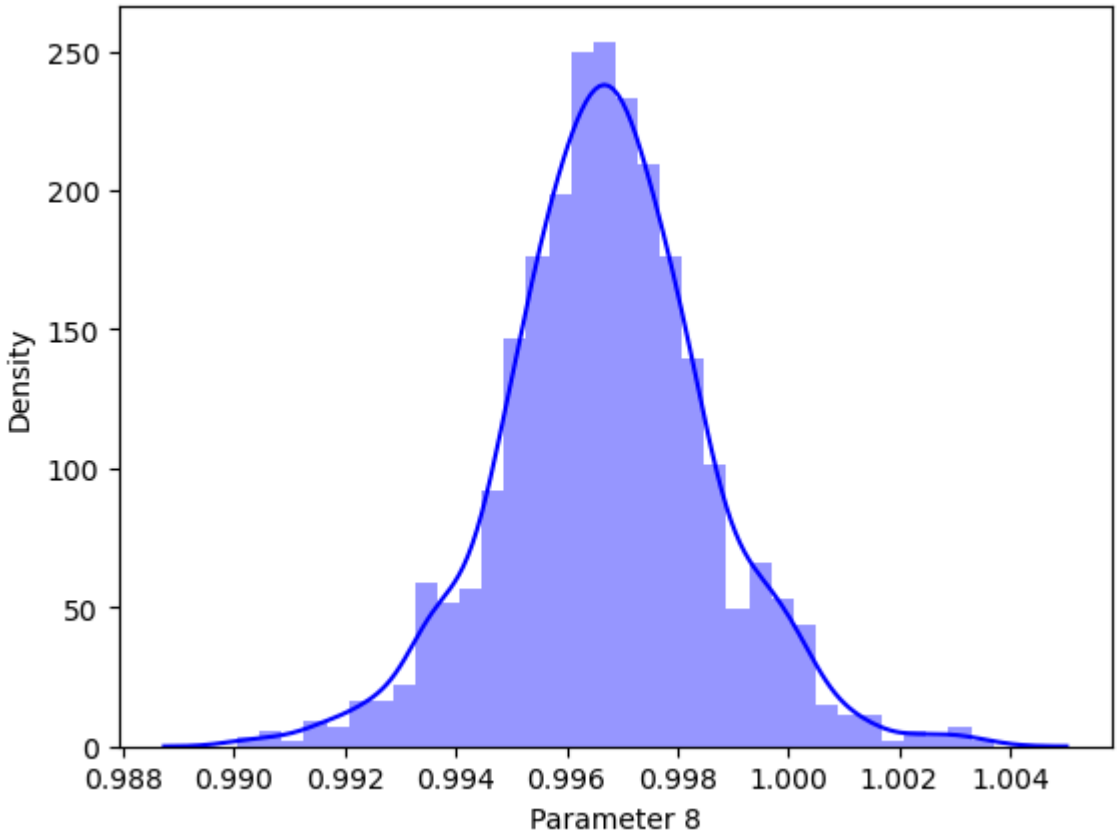



```
Distribution of  Parameter 6
Mean is: 15.893303899926417
Median is: 14.0
Mode is: 0    6.0
Name: Parameter 6, dtype: float64
Standard deviation is: 10.447270259048695
Skewness is: 1.2265794991760643
Maximum is: 72.0
Minimum is: 1.0
```

```
C:\Users\CHIRAG\AppData\Local\Temp\ipykernel_13804\2113990952.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[cols[i]], color='blue')
```

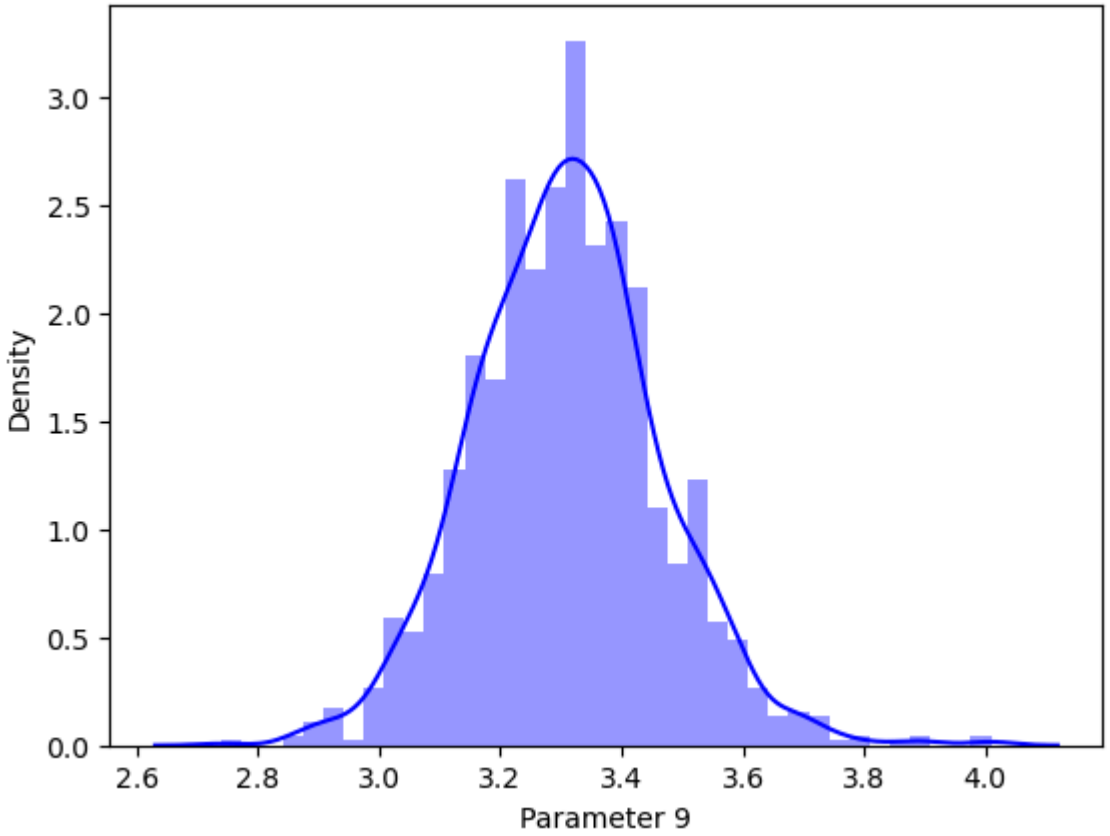


```
Distribution of  Parameter 7
Mean is: 46.82597498160412
Median is: 38.0
Mode is: 0    28.0
Name: Parameter 7, dtype: float64
Standard deviation is: 33.40894570661654
Skewness is: 1.5403680777213933
Maximum is: 289.0
Minimum is: 6.0
C:\Users\CHIRAG\AppData\Local\Temp\ipykernel_13804\2113990952.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[cols[i]], color='blue')
```



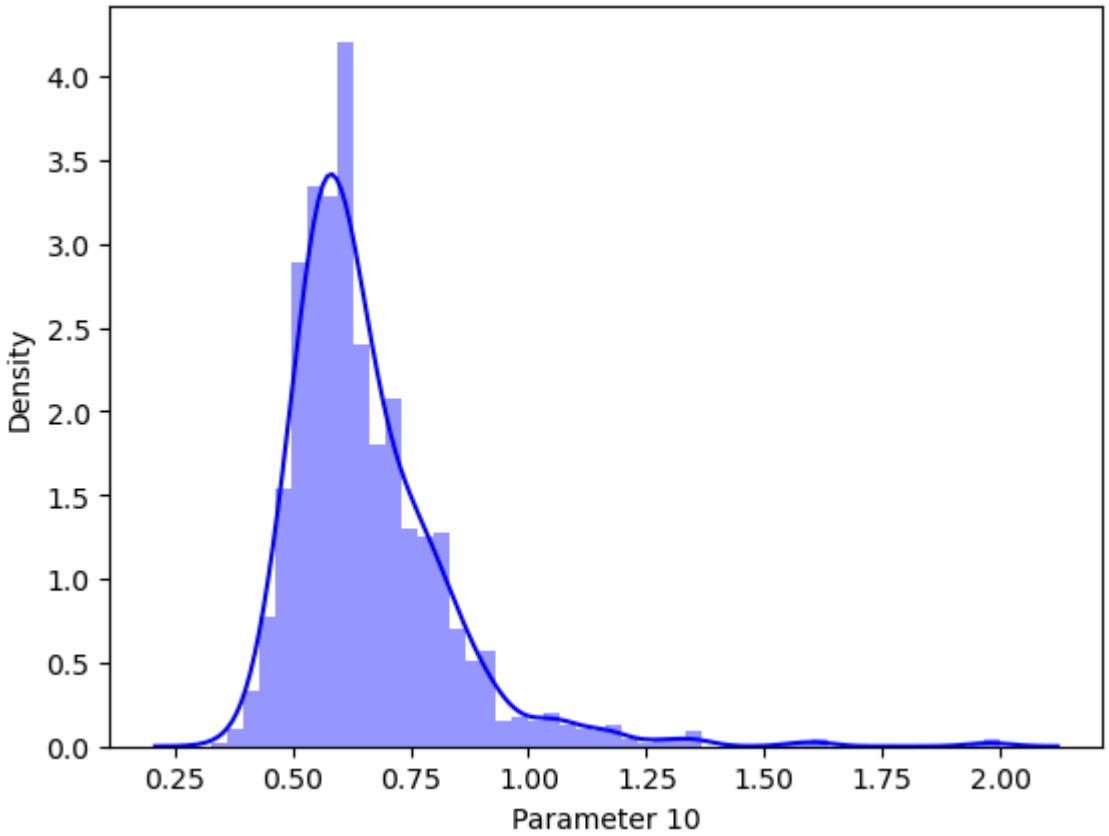

```
Distribution of  Parameter 8
Mean is: 0.9967089477557026
Median is: 0.9967
Mode is: 0    0.9968
Name: Parameter 8, dtype: float64
Standard deviation is: 0.0018689171325591398
Skewness is: 0.04477785573116107
Maximum is: 1.00369
Minimum is: 0.99007
```

```
C:\Users\CHIRAG\AppData\Local\Temp\ipykernel_13804\2113990952.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[cols[i]], color='blue')
```



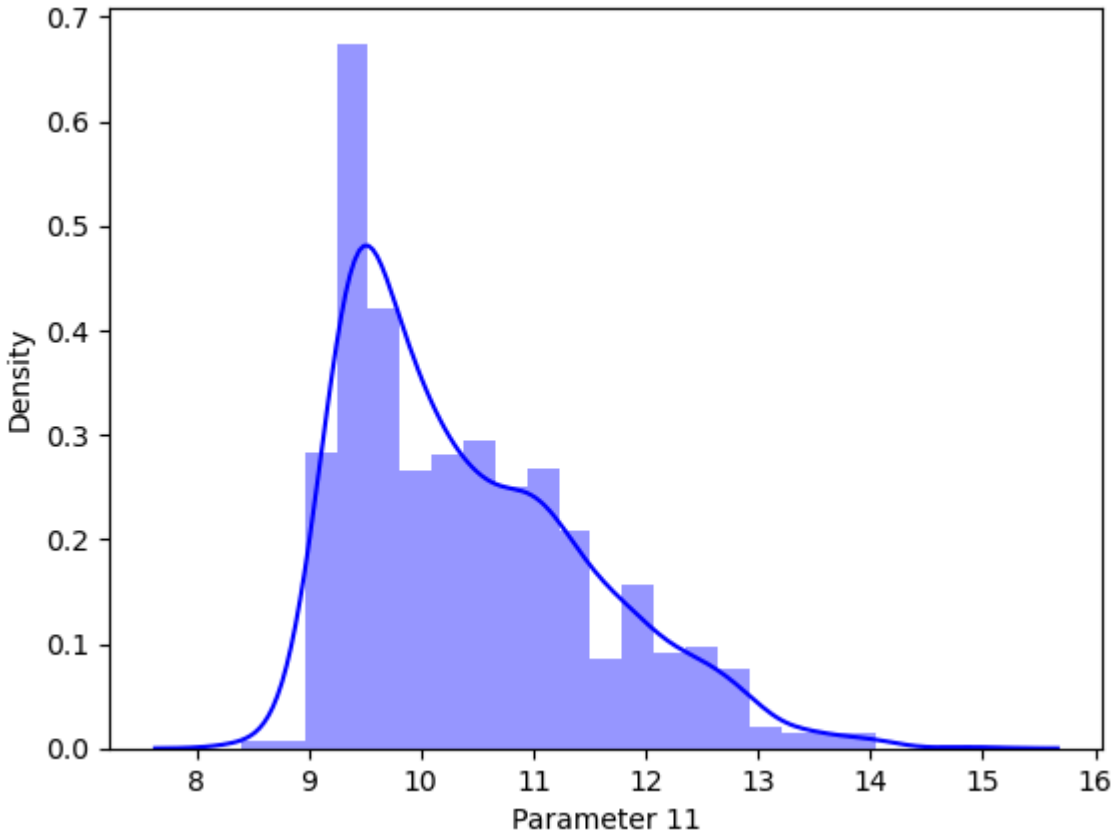
```
Distribution of  Parameter 9
Mean is: 3.309786607799853
Median is: 3.31
Mode is: 0    3.3
Name: Parameter 9, dtype: float64
Standard deviation is: 0.15503631128729595
Skewness is: 0.2320322752014824
Maximum is: 4.01
Minimum is: 2.74
```

```
C:\Users\CHIRAG\AppData\Local\Temp\ipykernel_13804\2113990952.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[cols[i]], color='blue')
```



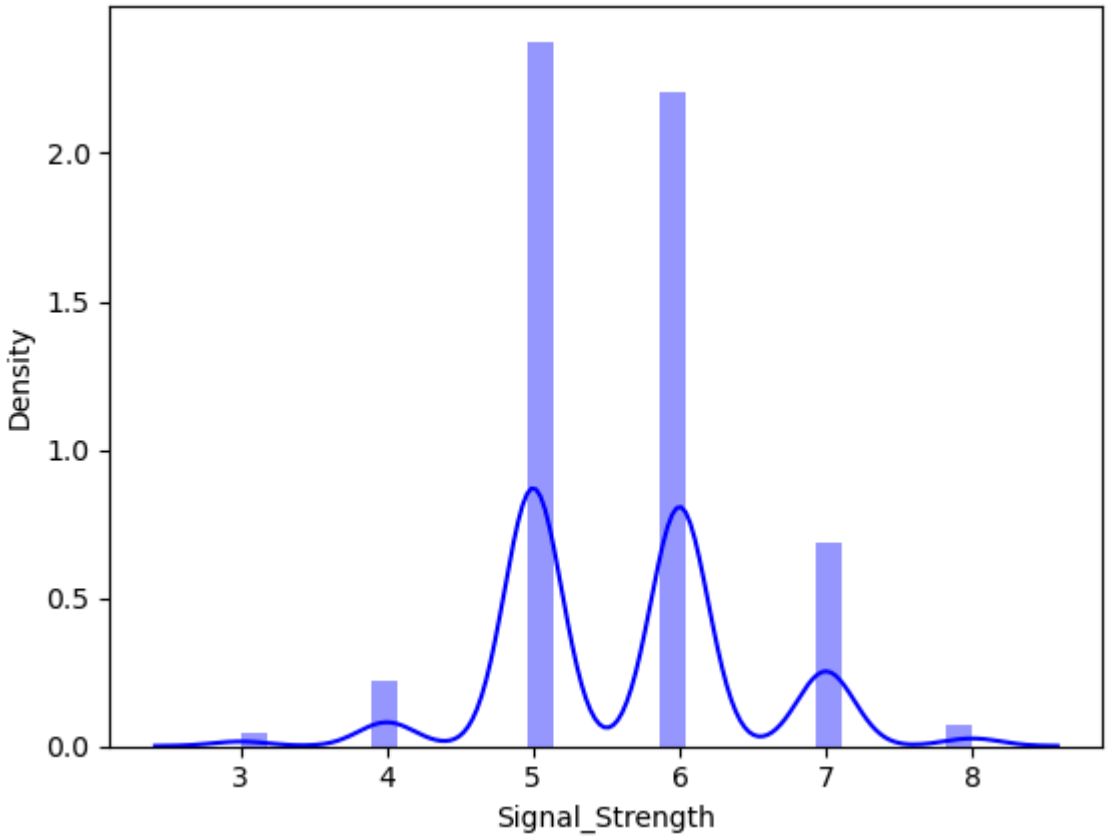

```
Distribution of  Parameter 10
Mean is: 0.6587049300956587
Median is: 0.62
Mode is: 0    0.54
Name: Parameter 10, dtype: float64
Standard deviation is: 0.17066689057420695
Skewness is: 2.4065046145674196
Maximum is: 2.0
Minimum is: 0.33
```

```
C:\Users\CHIRAG\AppData\Local\Temp\ipykernel_13804\2113990952.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[cols[i]], color='blue')
```



```
Distribution of  Parameter 11
Mean is: 10.432315428013245
Median is: 10.2
Mode is: 0    9.5
Name: Parameter 11, dtype: float64
Standard deviation is: 1.0820654499497833
Skewness is: 0.8598411692319623
Maximum is: 14.9
Minimum is: 8.4
```

```
C:\Users\CHIRAG\AppData\Local\Temp\ipykernel_13804\2113990952.py:3: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df[cols[i]], color='blue')
```



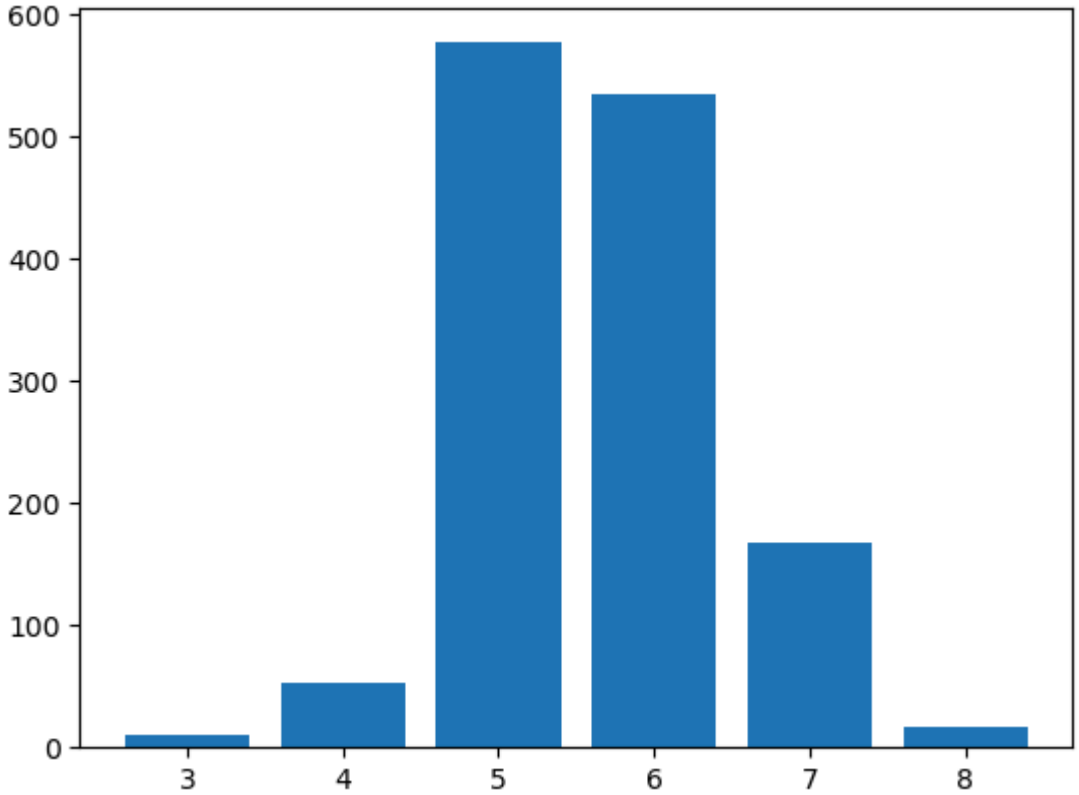

```
Distribution of  Signal_Strength
Mean is: 5.6232523914643116
Median is: 6.0
Mode is: 0    5
Name: Signal_Strength, dtype: int64
Standard deviation is: 0.8235780017165619
Skewness is: 0.19240658731658308
Maximum is: 8
Minimum is: 3
```

In [232…

```
plt.bar(df['Signal_Strength'].unique(),df['Signal_Strength'].value_counts())
```

Out[232]:

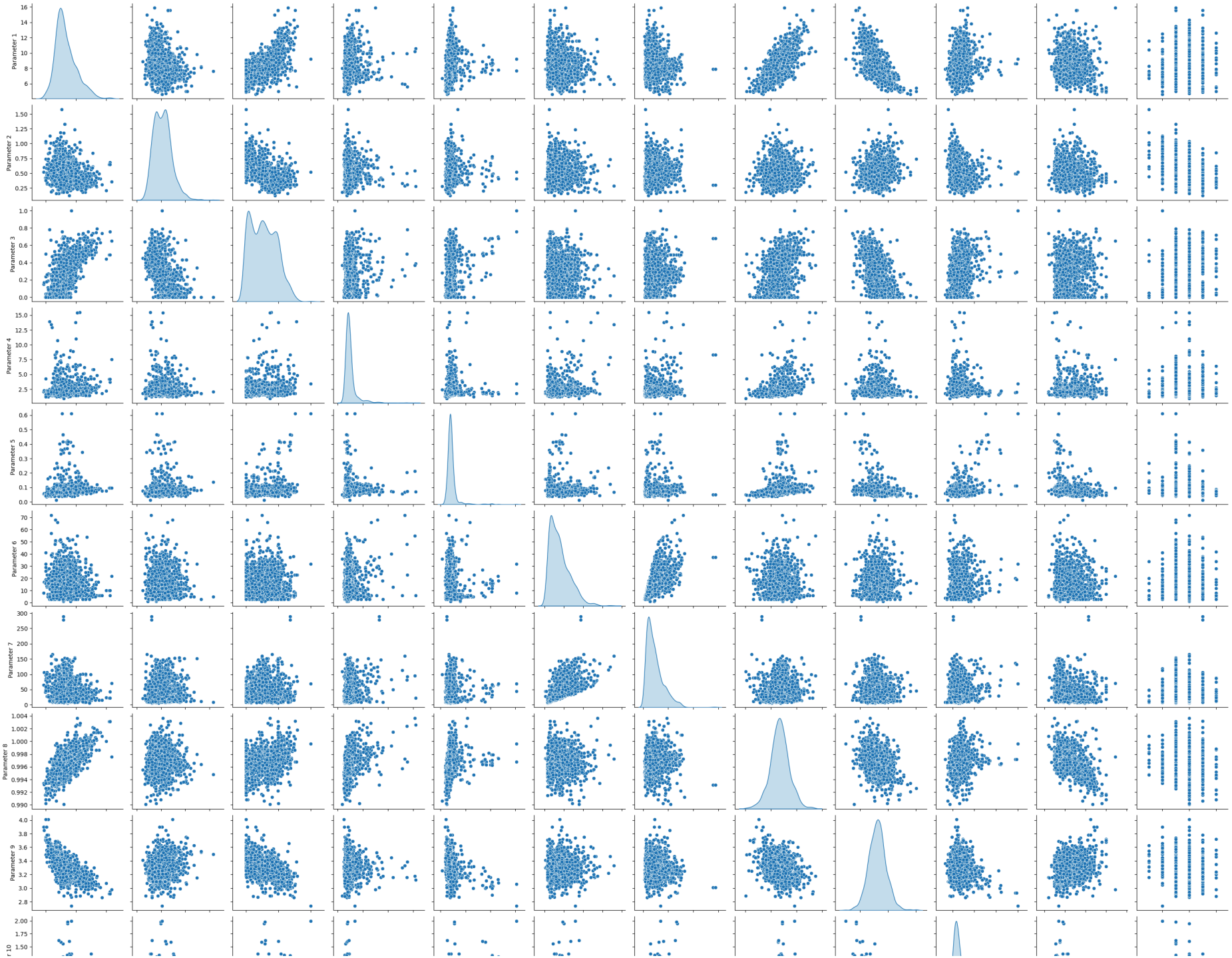
```
<BarContainer object of 6 artists>
```

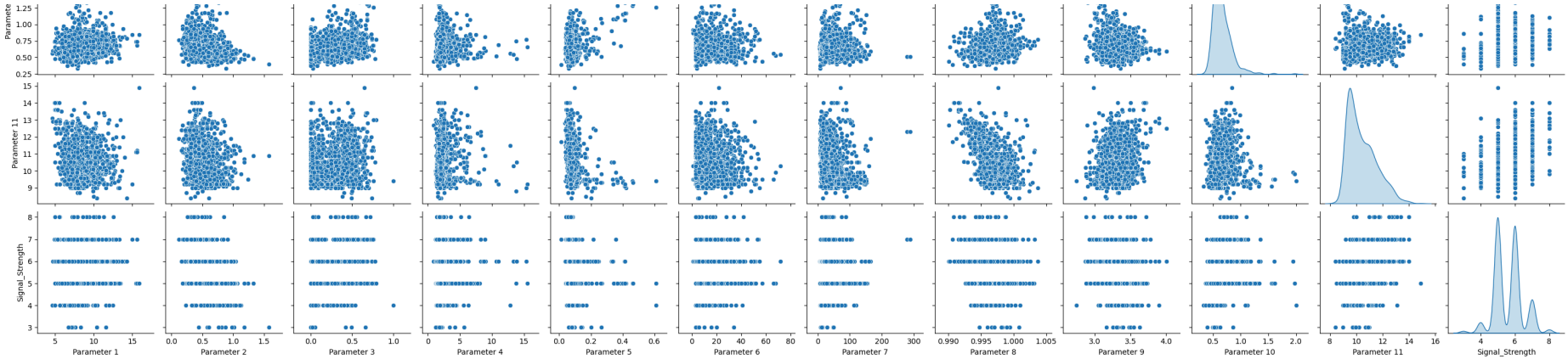


In [233…

```
#plt.figure(figsize = (50,50))
sns.pairplot(df,diag_kind='kde')
plt.show()
```

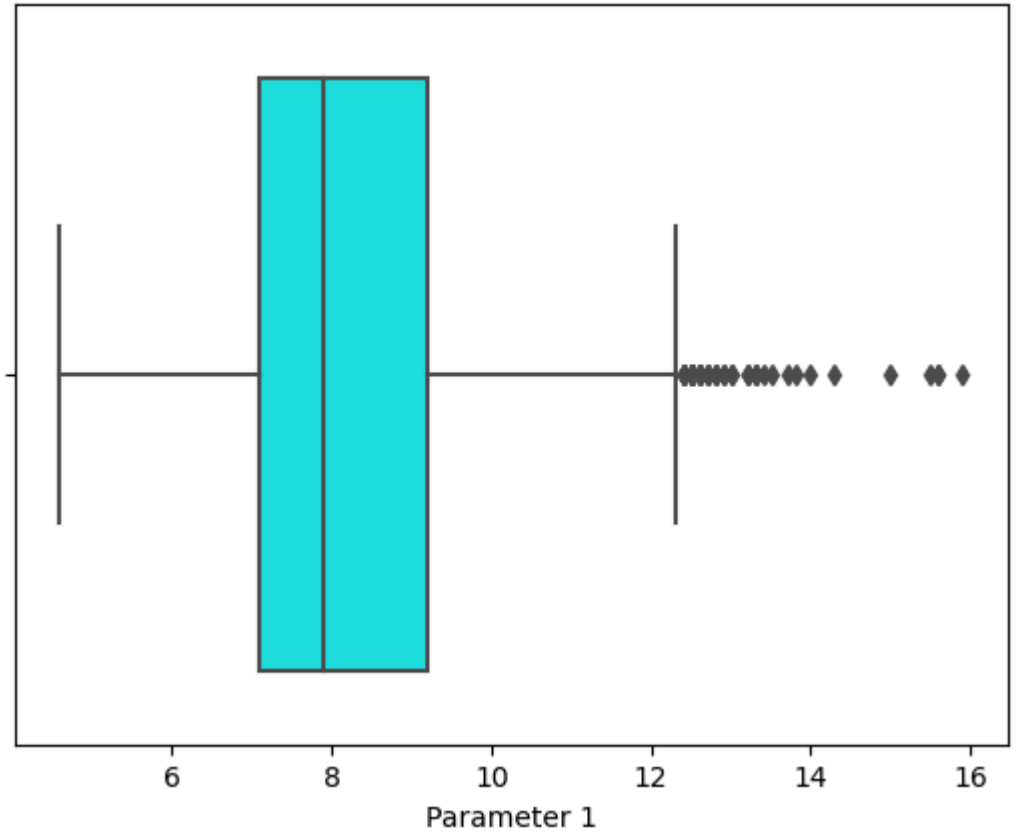
```
C:\Users\CHIRAG\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```


Parameter 1
Parameter 2
Parameter 3
Parameter 4
Parameter 5
Parameter 6
Parameter 7
Parameter 8
Parameter 9
Parameter 10

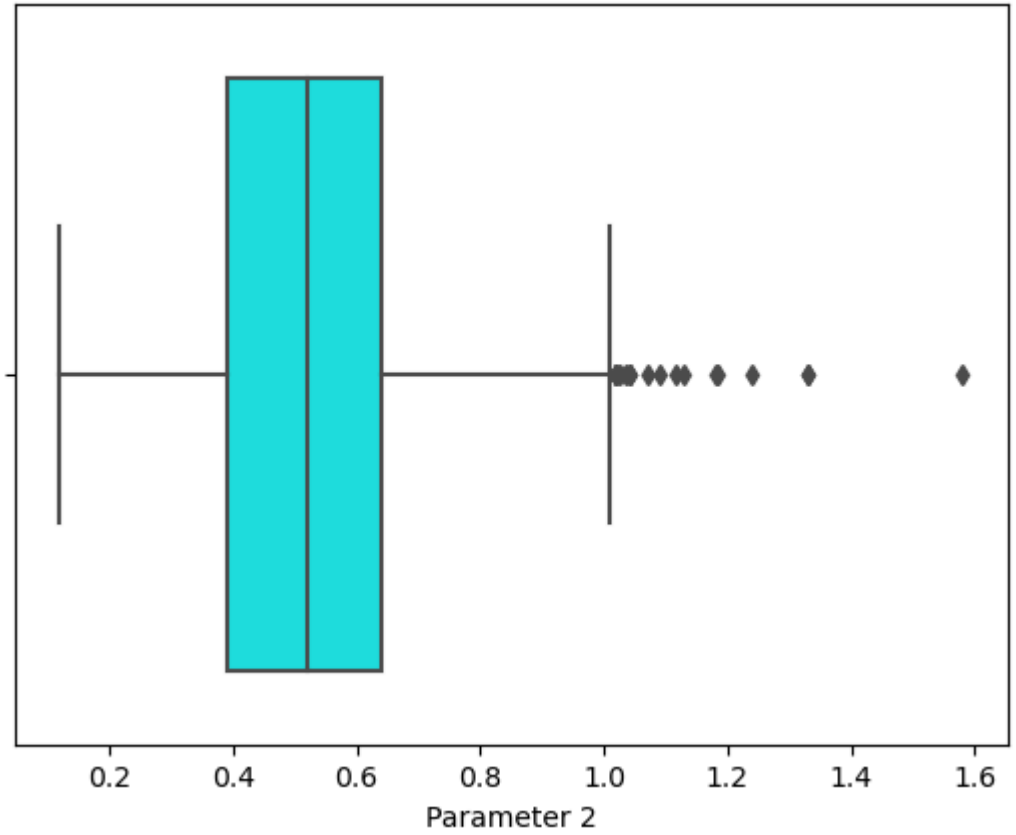


In [234…

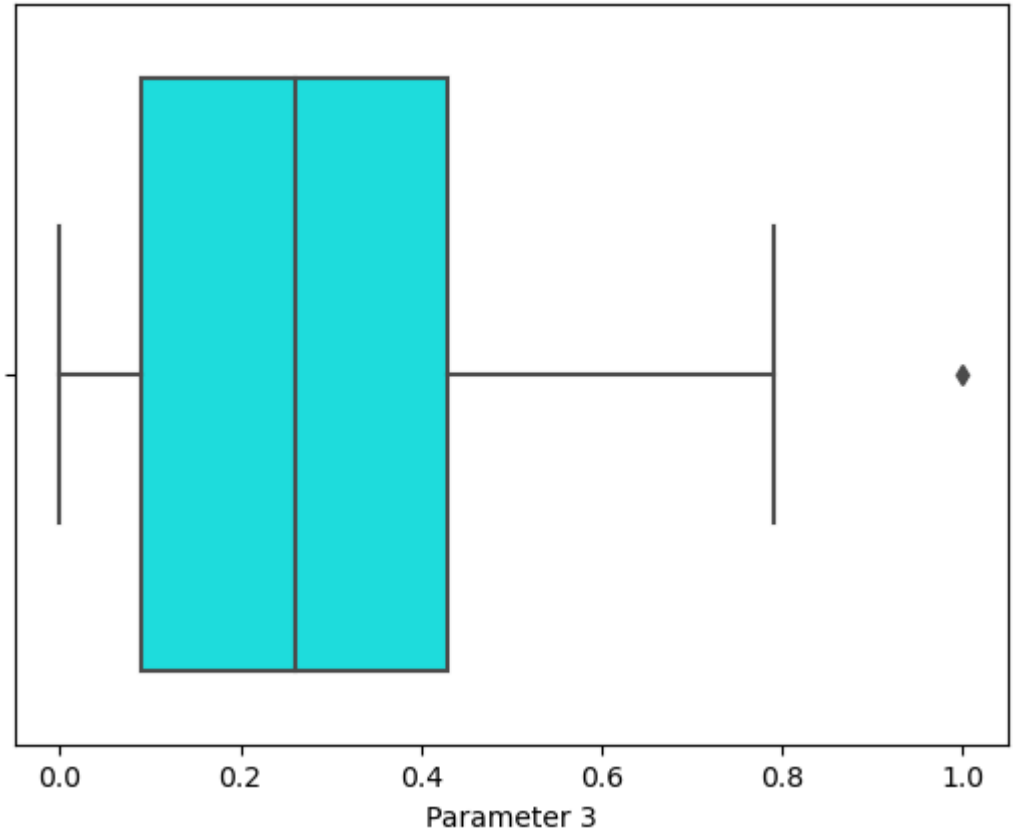
```
# Checking the presence of outliers
l = len(df)
col = list(df.columns)
#col.remove('condition')
for i in np.arange(len(col)):
    sns.boxplot(x= df[col[i]], color='cyan')
    plt.show()
    print('Boxplot of ',col[i])
    #calculating the outiers in attribute
    Q1 = df[col[i]].quantile(0.25)
    Q2 = df[col[i]].quantile(0.50)
    Q3 = df[col[i]].quantile(0.75)
    IQR = Q3 - Q1
    L_W = (Q1 - 1.5 *IQR)
    U_W = (Q3 + 1.5 *IQR)
    print('Q1 is : ',Q1)
    print('Q2 is : ',Q2)
    print('Q3 is : ',Q3)
    print('IQR is:',IQR)
    print('Lower Whisker, Upper Whisker : ',L_W,',',U_W)
    bools = (df[col[i]] < (Q1 - 1.5 *IQR)) |(df[col[i]] > (Q3 + 1.5 * IQR))
    print('Out of ',l,' rows in data, number of outliers are:',bools.sum())   #calculating the number of outliers
```



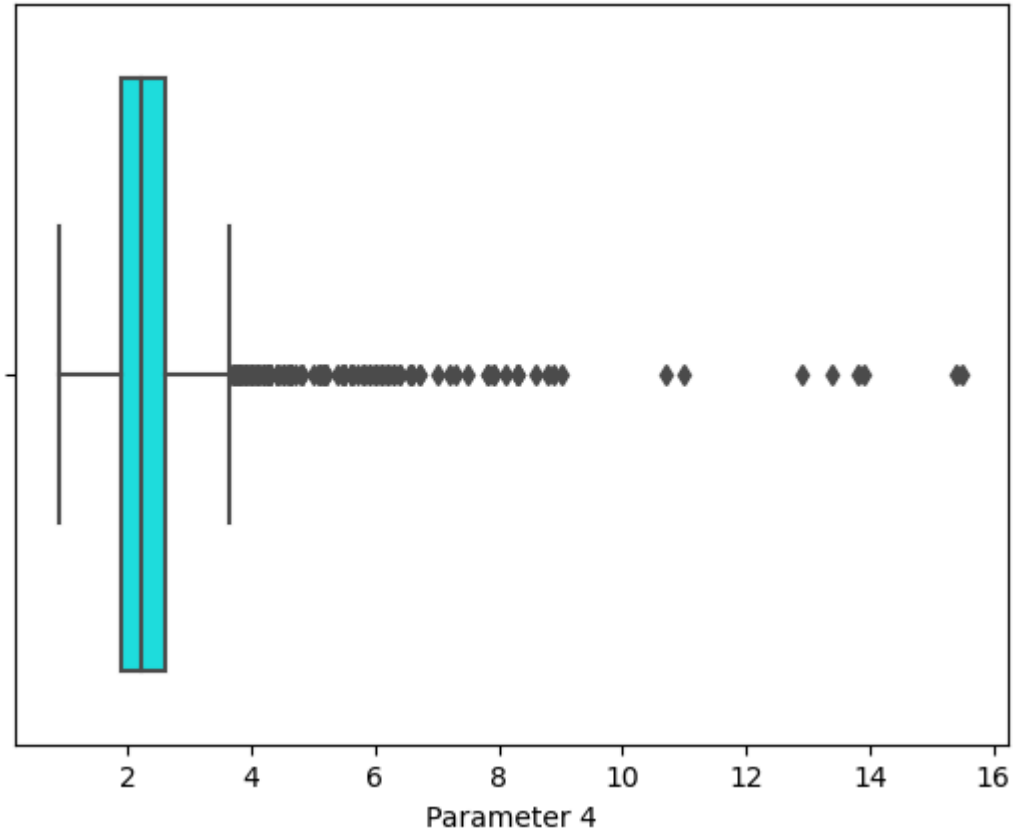

```
Boxplot of  Parameter 1
Q1 is :  7.1
Q2 is :  7.9
Q3 is :  9.2
IQR is: 2.0999999999999996
Lower Whisker, Upper Whisker :  3.95 , 12.349999999999998
Out of  1359  rows in data, number of outliers are: 41
```



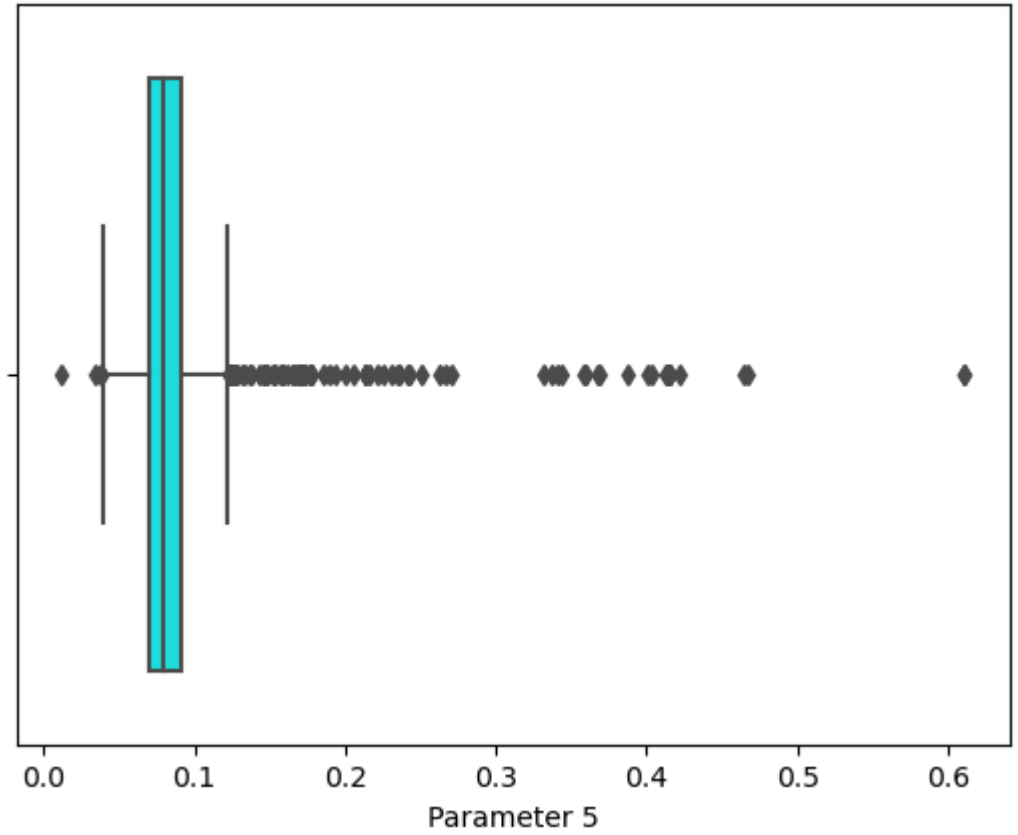

```
Boxplot of  Parameter 2
Q1 is :  0.39
Q2 is :  0.52
Q3 is :  0.64
IQR is: 0.25
Lower Whisker, Upper Whisker :  0.015000000000000013 , 1.0150000000000001
Out of  1359  rows in data, number of outliers are: 19
```



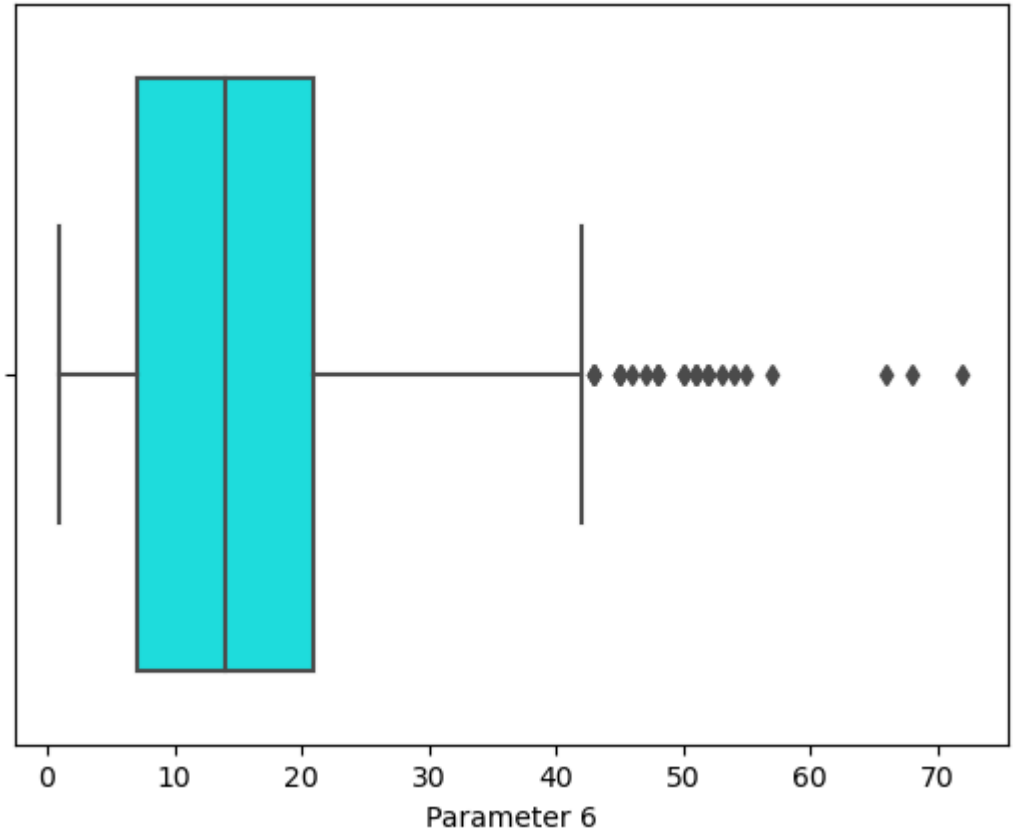

```
Boxplot of  Parameter 3
Q1 is :  0.09
Q2 is :  0.26
Q3 is :  0.43
IQR is: 0.33999999999999997
Lower Whisker, Upper Whisker :  -0.42000000000000004 , 0.94
Out of  1359  rows in data, number of outliers are: 1
```



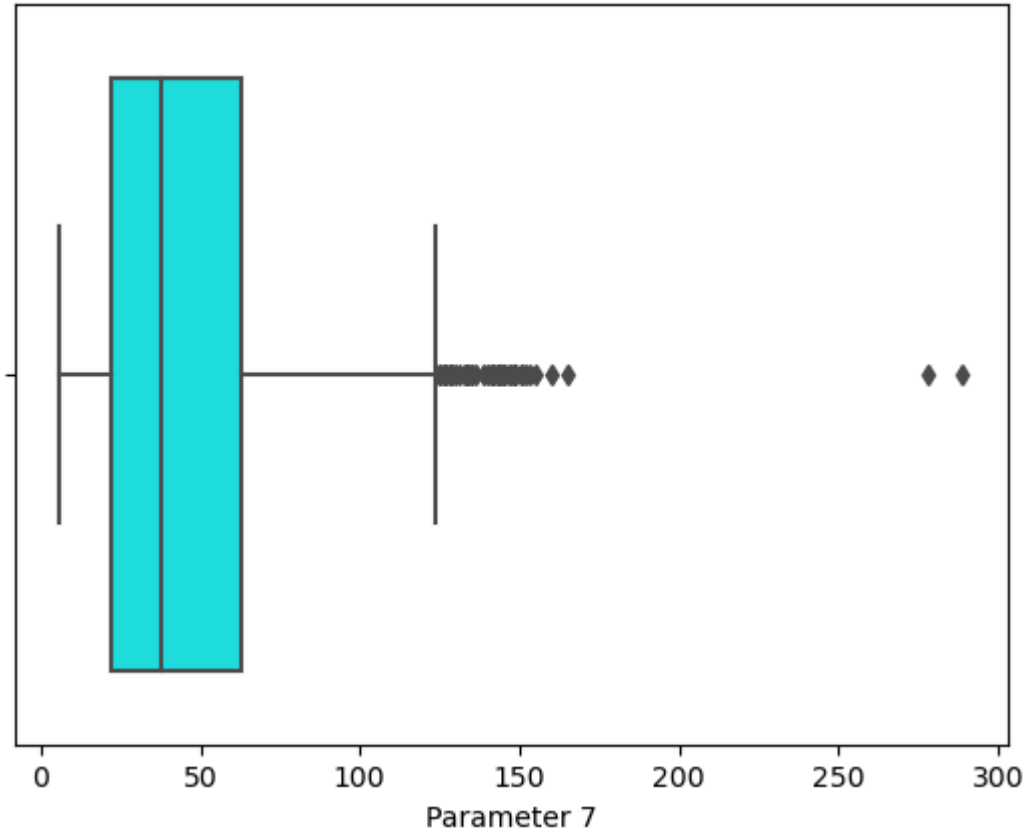

```
Boxplot of  Parameter 4
Q1 is :  1.9
Q2 is :  2.2
Q3 is :  2.6
IQR is: 0.7000000000000002
Lower Whisker, Upper Whisker :  0.8499999999999996 , 3.6500000000000004
Out of  1359  rows in data, number of outliers are: 126
```



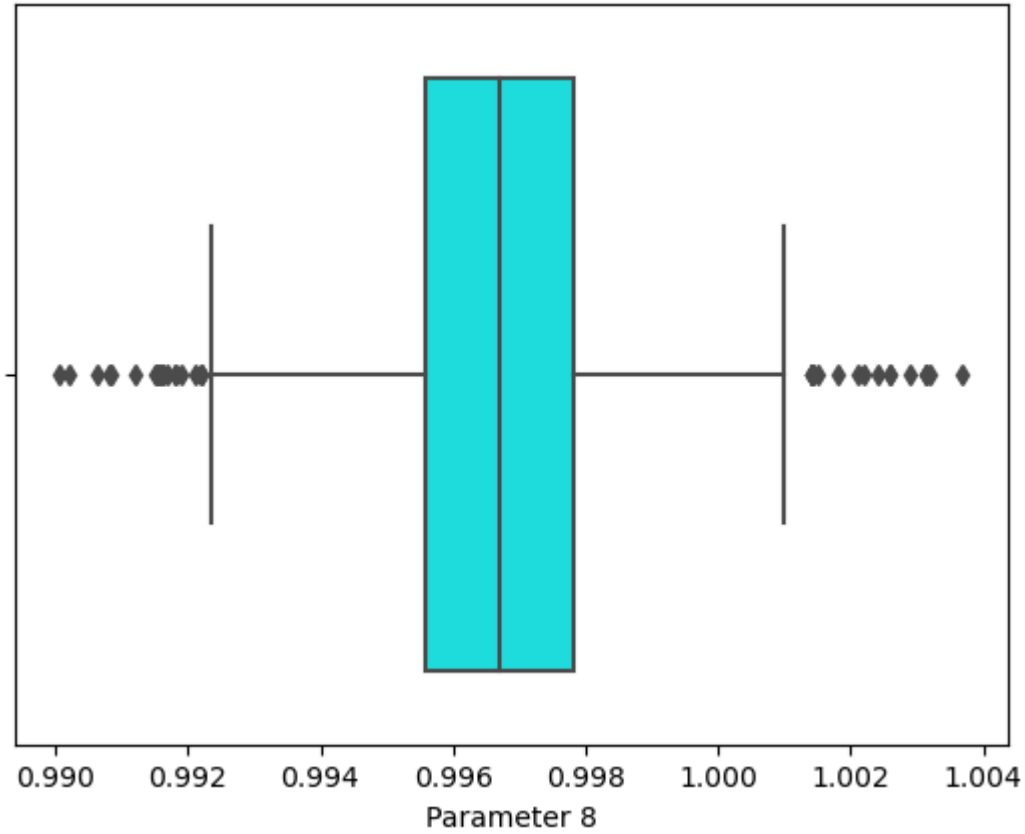

```
Boxplot of  Parameter 5
Q1 is :  0.07
Q2 is :  0.079
Q3 is :  0.091
IQR is: 0.02099999999999999
Lower Whisker, Upper Whisker :  0.03850000000000002 , 0.12249999999999998
Out of  1359  rows in data, number of outliers are: 87
```



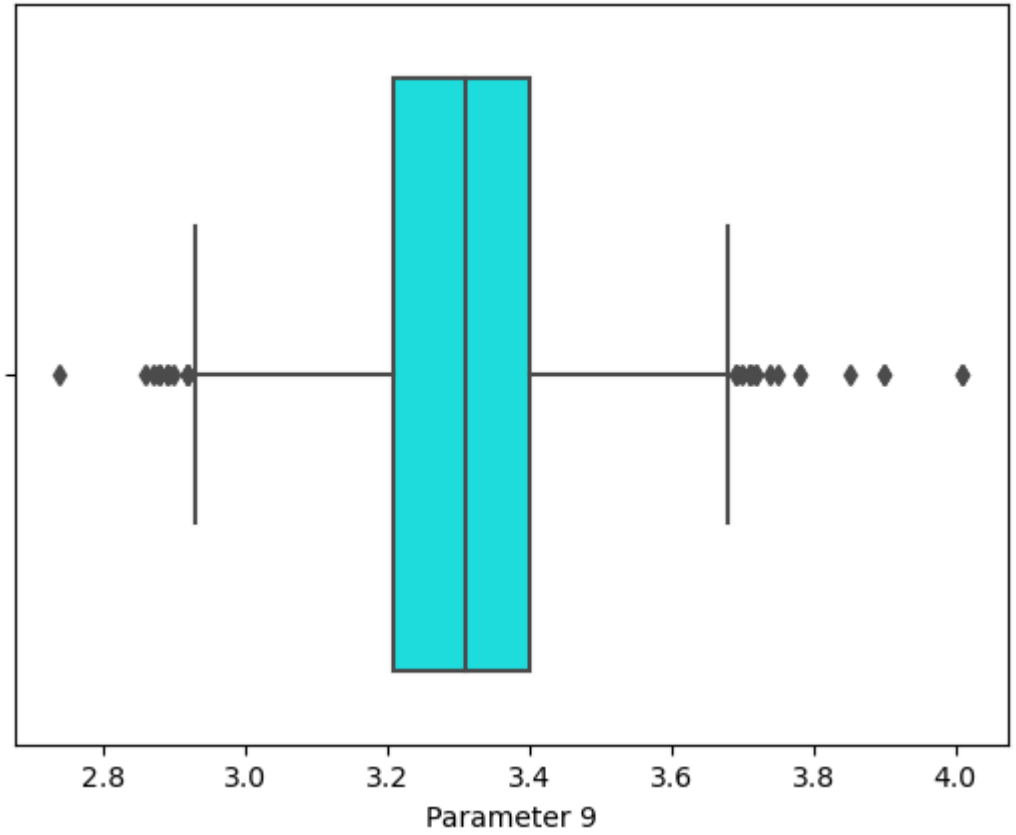

```
Boxplot of  Parameter 6
Q1 is :  7.0
Q2 is :  14.0
Q3 is :  21.0
IQR is: 14.0
Lower Whisker, Upper Whisker :  -14.0 , 42.0
Out of  1359  rows in data, number of outliers are: 26
```



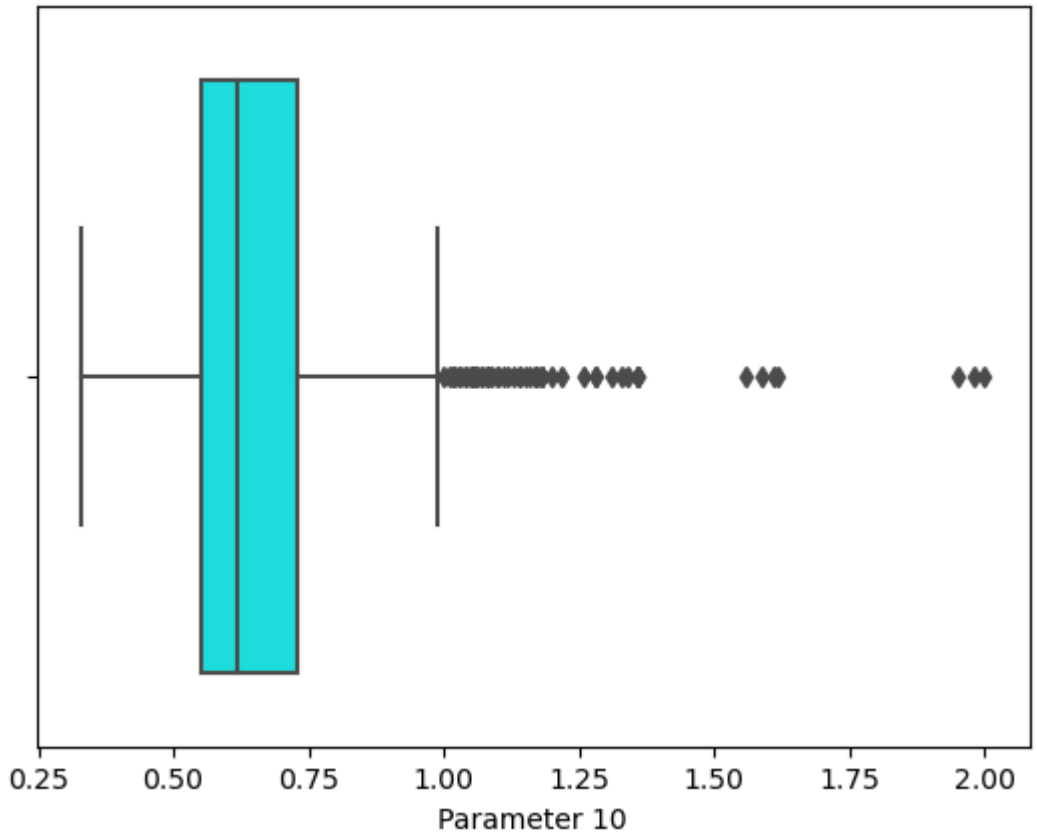

```
Boxplot of  Parameter 7
Q1 is :  22.0
Q2 is :  38.0
Q3 is :  63.0
IQR is: 41.0
Lower Whisker, Upper Whisker :  -39.5 , 124.5
Out of  1359  rows in data, number of outliers are: 45
```



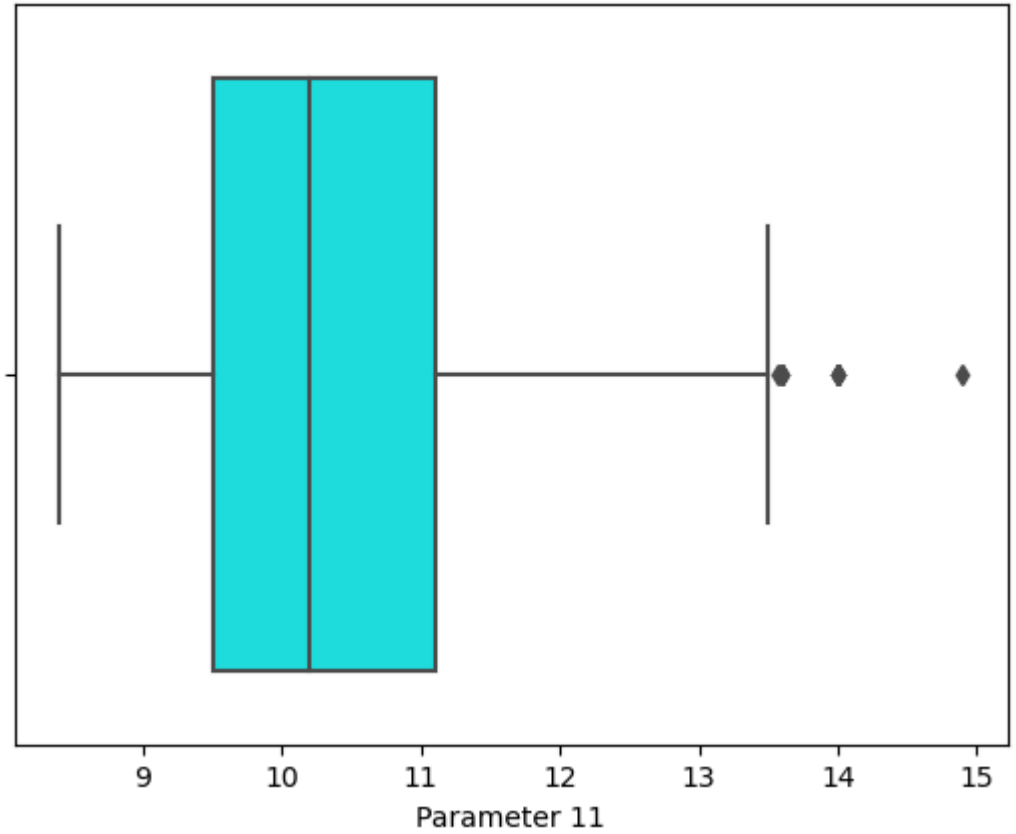

```
Boxplot of  Parameter 8
Q1 is :  0.9956
Q2 is :  0.9967
Q3 is :  0.99782
IQR is: 0.0022199999999999998
Lower Whisker, Upper Whisker :  0.99227 , 1.00115
Out of  1359  rows in data, number of outliers are: 35
```



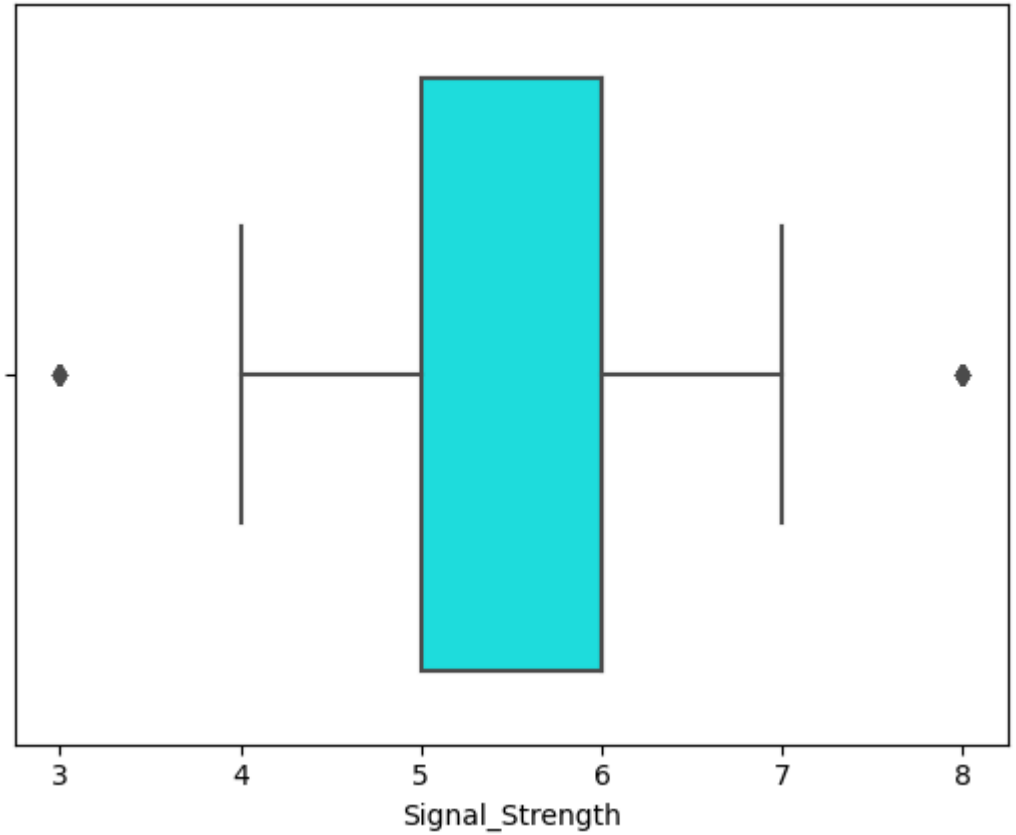

```
Boxplot of  Parameter 9
Q1 is :  3.21
Q2 is :  3.31
Q3 is :  3.4
IQR is: 0.18999999999999995
Lower Whisker, Upper Whisker :  2.925 , 3.6849999999999996
Out of  1359  rows in data, number of outliers are: 28
```




```
Boxplot of  Parameter 10
Q1 is :  0.55
Q2 is :  0.62
Q3 is :  0.73
IQR is: 0.17999999999999994
Lower Whisker, Upper Whisker :  0.28000000000000014 , 0.9999999999999999
Out of  1359  rows in data, number of outliers are: 55
```




```
Boxplot of  Parameter 11
Q1 is :  9.5
Q2 is :  10.2
Q3 is :  11.1
IQR is: 1.5999999999999996
Lower Whisker, Upper Whisker :  7.1000000000000005 , 13.5
Out of  1359  rows in data, number of outliers are: 12
```




```
Boxplot of  Signal_Strength
Q1 is :  5.0
Q2 is :  6.0
Q3 is :  6.0
IQR is: 1.0
Lower Whisker, Upper Whisker :  3.5 , 7.5
Out of  1359  rows in data, number of outliers are: 27
```

1 E.Share insights from the initial data analysis (at least 2). [2 Marks]

- Insights from Data Analysis
- Parameter 1 (Mean: 8.32, Std: 1.74): This parameter has a moderate standard deviation relative to its mean, suggesting a relatively consistent measurement across samples. The distribution is slightly right-skewed as indicated by the mean being greater than the median (7.90).

Outliers may exist as the maximum value (15.90) is significantly higher than the 75th percentile (9.20).

- Parameter 2 (Mean: 0.53, Std: 0.18): This parameter is tightly clustered around its mean with a low standard deviation. The mean and median (0.52) are very close, indicating a symmetric distribution. The maximum value (1.58) is nearly three times the 75th percentile (0.64), which might indicate a few significant outliers.

- Parameter 3 (Mean: 0.27, Std: 0.19): This parameter has a relatively high standard deviation compared to its mean. The distribution is slightly right-skewed as the mean is slightly higher than the median (0.26). There is a significant spread in the data with values ranging from 0 to 1.00.

- Parameter 4 (Mean: 2.54, Std: 1.41): This parameter shows a higher degree of variability with a large standard deviation. The median (2.20) is less than the mean, suggesting a right-skewed distribution. There are extreme outliers as the maximum value (15.50) is much higher than the 75th percentile (2.60).

- Parameter 5 (Mean: 0.09, Std: 0.05): This parameter has a very small standard deviation, indicating that the values are tightly packed around the mean. The distribution is symmetric as the mean and median (0.079) are close. The maximum value (0.611) is an outlier, as it is significantly higher than the 75th percentile (0.09).

- Parameter 6 (Mean: 15.87, Std: 10.46): This parameter has a high standard deviation, indicating a wide spread of values. The distribution is right-skewed as the mean is higher than the median (14.00). The presence of extreme outliers is indicated by the maximum value (72.00), which is much higher than the 75th percentile (21.00).

- Parameter 7 (Mean: 46.47, Std: 32.90): This parameter has a very high standard deviation, suggesting a very wide spread of values. The distribution is right-skewed as the mean is higher than the median (38.00). There are extreme outliers, indicated by the maximum value (289.00).

- Parameter 8 (Mean: 0.997, Std: 0.0019): This parameter is very tightly clustered around the mean with a very low standard deviation. The mean and median (0.99675) are very close, indicating a symmetric distribution. There are no significant outliers, as the range is quite narrow (0.99007 to 1.00369).

- Parameter 9 (Mean: 3.31, Std: 0.15): This parameter has a small standard deviation, suggesting a narrow spread around the mean. The distribution is symmetric as the mean and median (3.31) are almost identical. There are no significant outliers, as the range is narrow (2.74 to 4.01).

- Parameter 10 (Mean: 0.66, Std: 0.17): This parameter shows moderate variability with a standard deviation relative to its mean. The distribution is slightly right-skewed as indicated by the mean being greater than the median (0.62). There may be some outliers since the maximum value (2.00) is significantly higher than the 75th percentile (0.73).

- Parameter 11 (Mean: 10.42, Std: 1.07): This parameter has a relatively low standard deviation, indicating a consistent measurement across samples. The mean and median (10.20) are close, suggesting a symmetric distribution. There are no significant outliers, as the range is moderate (8.40 to 14.90).

- Signal_Strength (Mean: 5.64, Std: 0.81): The target variable has a moderate standard deviation. The mean and median (6.00) suggest a slightly left-skewed distribution. The range is narrow (3.00 to 8.00), indicating that signal strength is fairly consistent across samples.

General Insights

- Skewness and Outliers: Parameters 4, 6, and 7 have significant right skewness and potential outliers that might need to be addressed through transformations or robust statistical methods.
- Symmetry: Parameters 2, 5, 8, 9, and 11 show symmetric distributions, indicating that these parameters might be well-suited for algorithms that assume normality.
- Consistency: Parameters 8 and 9 have very low standard deviations, suggesting high consistency in these measurements.
- Correlation with Signal Strength: Further analysis (such as correlation coefficients) would be necessary to determine how each parameter relates to the signal strength. This can help in feature selection for predictive modeling.
- maximum parameter falls under signal strenght 5 that is 577
- after that signal strenght number 6 appears most of the time
- lowest time type 3 and 8 signal strength appears
- from correlation it appears that there are no strong poitive or negative correlation between any two signal
- parameter1 and parameter 8 showing some positive correlation
- parameter1 and parameter 9 showing some negative correlation
- here we cant perform dimensinality reduction we might endup loosing important information as most of the pair has no strong correlation
- parameter 8 and parameter 9 assumed to be normaly distributed.
- most of the signal has outliers
- for signal strength category 3 and 8 shows rare appearance.

Next Steps

- Data Preprocessing: Consider normalizing or standardizing the data, especially for skewed parameters.
- Outlier Handling: Identify and potentially remove or transform outliers to reduce their impact on the model.
- Correlation Analysis: Perform correlation analysis to understand the relationships between parameters and the target variable.
- Feature Engineering: Explore creating new features from existing parameters if it improves model performance.
- Model Selection: Test different machine learning models to determine which one best predicts signal strength.

2. Data preprocessing [7 Marks]

A. Split the data into X & Y. [1 Marks]

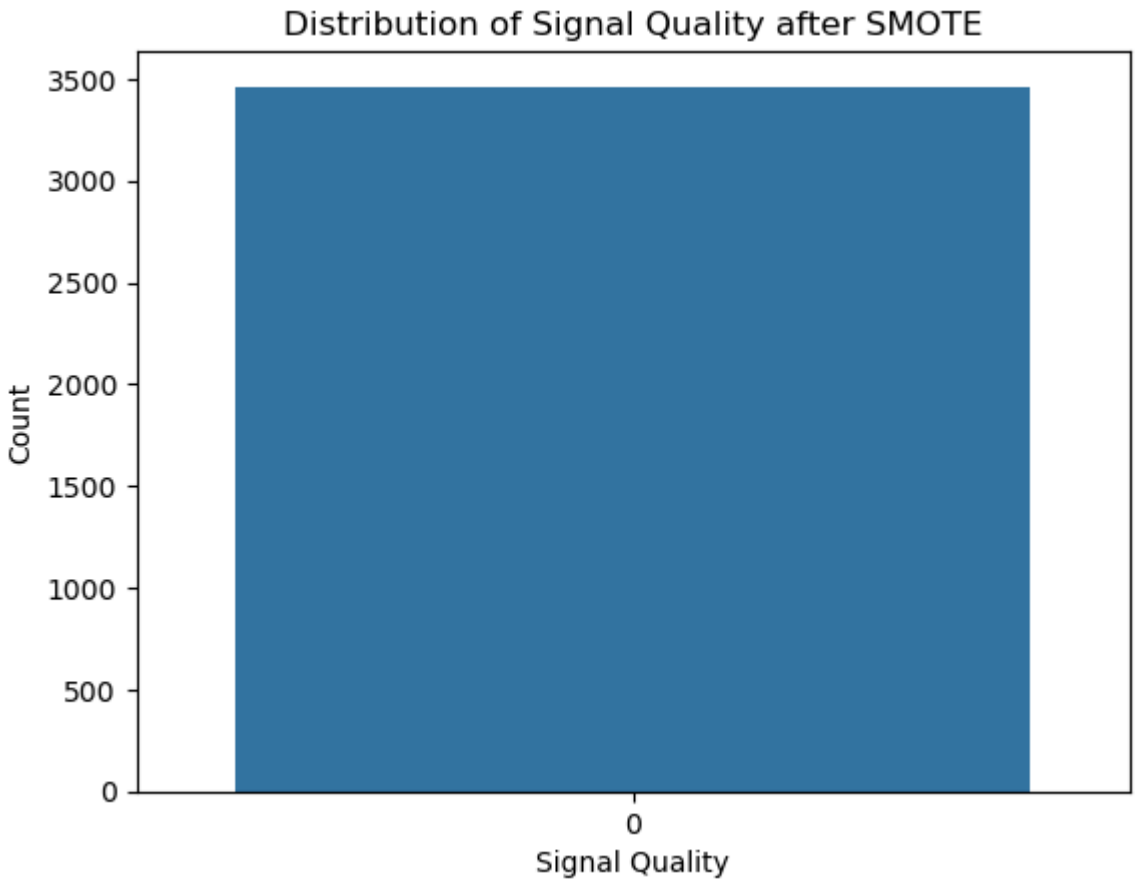
- **before that lets do label encoding to caonver target variable into categrical**

In [235...
```
X = df.drop(['Signal_Strength'],axis=1)
Y = df['Signal_Strength']
```

- **with this lets perform SMOTE to deal with data imbalance**

In [236...
```
smt=SMOTE()
X_resampled,Y_resampled= smt.fit_resample(X,Y)
```

In [237...
```
sns.countplot(Y_resampled)
plt.title('Distribution of Signal Quality after SMOTE')
plt.xlabel('Signal Quality')
plt.ylabel('Count')
plt.show()
```



In [238…
```
print(X_resampled.shape)
print(Y_resampled.shape)
```

```
(3462, 11)
(3462,)
```

In [239…
```
Y_resampled.value_counts()
```

Out[239]:
```
Signal_Strength
5    577
6    577
7    577
4    577
8    577
3    577
Name: count, dtype: int64
```

2 B. Split the data into train & test with 70:30 proportion.[1 Marks]

In [240…
```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, Y_resampled, test_size=0.3, random_state=42)
```

2 C. Print shape of all the 4 variables and verify if train and test data is in sync. [1 mark]

In [241…
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2423, 11)
(1039, 11)
(2423,)
(1039,)
```

2 D. Normalise the train and test data with appropriate method. [2 Marks]

In [242…
```
standard_scaler=StandardScaler()
X_train_scaled=standard_scaler.fit_transform(X_train)
X_test_scaled=standard_scaler.fit_transform(X_test)
```

2 E. Transform Labels into format acceptable by Neural Network [2 Marks]

In [243…
```
y_train_categorical= to_categorical(y_train)
y_test_categorical=  to_categorical(y_test)
```

In [244…
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2423, 11)
(1039, 11)
(2423,)
(1039,)
```

In [245…
```
df.head()
```

Out[245]:

	Parameter 1	Parameter 2	Parameter 3	Parameter 4	Parameter 5	Parameter 6	Parameter 7	Parameter 8	Parameter 9	Parameter 10	Parameter 11	Signal_Strength
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	5

3. Model Training & Evaluation using Neural Network [13 Marks]

3A. Design a Neural Network to train a classifier. [3 Marks]

In [246…
```
model = Sequential()
model.add(Dense(32, input_shape=(X_train_scaled.shape[1],), activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(8, activation='relu'))
model.add(Dropout(0.2))
#model.add(Dense(32, activation='relu'))
#model.add(Dropout(0.2))
model.add(Dense(y_train_categorical.shape[1], activation='softmax'))
```

In [247…
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

3B. Train the classifier using previously designed Architecture [2 Marks]

In [168…

```
history = model.fit(X_train_scaled, y_train_categorical, validation_split=0.2, epochs=100, batch_size=32, verbose=1)
```

```
Epoch 1/100
61/61 [==============================] - 1s 3ms/step - loss: 2.2364 - accuracy: 0.1434 - val_loss: 2.0970 - val_accuracy: 0.1959
Epoch 2/100
61/61 [==============================] - 0s 2ms/step - loss: 2.0478 - accuracy: 0.2245 - val_loss: 1.9314 - val_accuracy: 0.3052
Epoch 3/100
61/61 [==============================] - 0s 2ms/step - loss: 1.8733 - accuracy: 0.3008 - val_loss: 1.7015 - val_accuracy: 0.4227
Epoch 4/100
61/61 [==============================] - 0s 1ms/step - loss: 1.6777 - accuracy: 0.3751 - val_loss: 1.4800 - val_accuracy: 0.4784
Epoch 5/100
61/61 [==============================] - 0s 2ms/step - loss: 1.5105 - accuracy: 0.4180 - val_loss: 1.3023 - val_accuracy: 0.5464
Epoch 6/100
61/61 [==============================] - 0s 1ms/step - loss: 1.4005 - accuracy: 0.4329 - val_loss: 1.1916 - val_accuracy: 0.5588
Epoch 7/100
61/61 [==============================] - 0s 2ms/step - loss: 1.3363 - accuracy: 0.4613 - val_loss: 1.1203 - val_accuracy: 0.5773
Epoch 8/100
61/61 [==============================] - 0s 2ms/step - loss: 1.2738 - accuracy: 0.4747 - val_loss: 1.0707 - val_accuracy: 0.5835
Epoch 9/100
61/61 [==============================] - 0s 2ms/step - loss: 1.2419 - accuracy: 0.4752 - val_loss: 1.0415 - val_accuracy: 0.5856
Epoch 10/100
61/61 [==============================] - 0s 1ms/step - loss: 1.2356 - accuracy: 0.4825 - val_loss: 1.0203 - val_accuracy: 0.6062
Epoch 11/100
61/61 [==============================] - 0s 2ms/step - loss: 1.2171 - accuracy: 0.4979 - val_loss: 1.0083 - val_accuracy: 0.6124
Epoch 12/100
61/61 [==============================] - 0s 2ms/step - loss: 1.2051 - accuracy: 0.4912 - val_loss: 0.9966 - val_accuracy: 0.6103
Epoch 13/100
61/61 [==============================] - 0s 2ms/step - loss: 1.1930 - accuracy: 0.4902 - val_loss: 0.9866 - val_accuracy: 0.6227
Epoch 14/100
61/61 [==============================] - 0s 2ms/step - loss: 1.1564 - accuracy: 0.5088 - val_loss: 0.9758 - val_accuracy: 0.6144
Epoch 15/100
61/61 [==============================] - 0s 2ms/step - loss: 1.1439 - accuracy: 0.5031 - val_loss: 0.9613 - val_accuracy: 0.6165
Epoch 16/100
61/61 [==============================] - 0s 2ms/step - loss: 1.1411 - accuracy: 0.5000 - val_loss: 0.9509 - val_accuracy: 0.6186
Epoch 17/100
61/61 [==============================] - 0s 2ms/step - loss: 1.1308 - accuracy: 0.5232 - val_loss: 0.9423 - val_accuracy: 0.6124
Epoch 18/100
61/61 [==============================] - 0s 2ms/step - loss: 1.1058 - accuracy: 0.5155 - val_loss: 0.9379 - val_accuracy: 0.6124
Epoch 19/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0947 - accuracy: 0.5330 - val_loss: 0.9316 - val_accuracy: 0.6206
Epoch 20/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0994 - accuracy: 0.5279 - val_loss: 0.9228 - val_accuracy: 0.6309
Epoch 21/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0973 - accuracy: 0.5397 - val_loss: 0.9191 - val_accuracy: 0.6289
Epoch 22/100
61/61 [==============================] - 0s 1ms/step - loss: 1.0869 - accuracy: 0.5299 - val_loss: 0.9120 - val_accuracy: 0.6351
```

```
Epoch 23/100
61/61 [==============================] - 0s 1ms/step - loss: 1.0876 - accuracy: 0.5330 - val_loss: 0.9059 - val_accuracy: 0.6412
Epoch 24/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0501 - accuracy: 0.5573 - val_loss: 0.8988 - val_accuracy: 0.6474
Epoch 25/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0603 - accuracy: 0.5604 - val_loss: 0.8894 - val_accuracy: 0.6577
Epoch 26/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0287 - accuracy: 0.5733 - val_loss: 0.8810 - val_accuracy: 0.6474
Epoch 27/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0399 - accuracy: 0.5645 - val_loss: 0.8751 - val_accuracy: 0.6557
Epoch 28/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0361 - accuracy: 0.5562 - val_loss: 0.8668 - val_accuracy: 0.6433
Epoch 29/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0145 - accuracy: 0.5702 - val_loss: 0.8607 - val_accuracy: 0.6536
Epoch 30/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0121 - accuracy: 0.5862 - val_loss: 0.8516 - val_accuracy: 0.6598
Epoch 31/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0121 - accuracy: 0.5929 - val_loss: 0.8432 - val_accuracy: 0.6619
Epoch 32/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0026 - accuracy: 0.5789 - val_loss: 0.8364 - val_accuracy: 0.6660
Epoch 33/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0018 - accuracy: 0.5800 - val_loss: 0.8330 - val_accuracy: 0.6660
Epoch 34/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9842 - accuracy: 0.6094 - val_loss: 0.8240 - val_accuracy: 0.6701
Epoch 35/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9770 - accuracy: 0.5918 - val_loss: 0.8216 - val_accuracy: 0.6763
Epoch 36/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9728 - accuracy: 0.5908 - val_loss: 0.8186 - val_accuracy: 0.6598
Epoch 37/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9706 - accuracy: 0.5949 - val_loss: 0.8129 - val_accuracy: 0.6742
Epoch 38/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9602 - accuracy: 0.6120 - val_loss: 0.8072 - val_accuracy: 0.6763
Epoch 39/100
61/61 [==============================] - 0s 1ms/step - loss: 0.9700 - accuracy: 0.5831 - val_loss: 0.8050 - val_accuracy: 0.6825
Epoch 40/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9488 - accuracy: 0.5913 - val_loss: 0.8047 - val_accuracy: 0.6866
Epoch 41/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9765 - accuracy: 0.6027 - val_loss: 0.8007 - val_accuracy: 0.6907
Epoch 42/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9600 - accuracy: 0.5970 - val_loss: 0.7950 - val_accuracy: 0.6866
Epoch 43/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9414 - accuracy: 0.6104 - val_loss: 0.7930 - val_accuracy: 0.6825
Epoch 44/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9509 - accuracy: 0.5944 - val_loss: 0.7936 - val_accuracy: 0.6907
```

```
Epoch 45/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9425 - accuracy: 0.6084 - val_loss: 0.7905 - val_accuracy: 0.6948
Epoch 46/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9442 - accuracy: 0.6156 - val_loss: 0.7875 - val_accuracy: 0.6928
Epoch 47/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9364 - accuracy: 0.6042 - val_loss: 0.7877 - val_accuracy: 0.6845
Epoch 48/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9297 - accuracy: 0.6171 - val_loss: 0.7828 - val_accuracy: 0.7031
Epoch 49/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9089 - accuracy: 0.6244 - val_loss: 0.7763 - val_accuracy: 0.6990
Epoch 50/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9293 - accuracy: 0.6104 - val_loss: 0.7721 - val_accuracy: 0.6928
Epoch 51/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9301 - accuracy: 0.6125 - val_loss: 0.7712 - val_accuracy: 0.6969
Epoch 52/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9166 - accuracy: 0.6151 - val_loss: 0.7661 - val_accuracy: 0.6948
Epoch 53/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9132 - accuracy: 0.6249 - val_loss: 0.7642 - val_accuracy: 0.7072
Epoch 54/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8998 - accuracy: 0.6285 - val_loss: 0.7629 - val_accuracy: 0.7031
Epoch 55/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9121 - accuracy: 0.6140 - val_loss: 0.7612 - val_accuracy: 0.7031
Epoch 56/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8966 - accuracy: 0.6342 - val_loss: 0.7579 - val_accuracy: 0.7031
Epoch 57/100
61/61 [==============================] - 0s 1ms/step - loss: 0.9072 - accuracy: 0.6269 - val_loss: 0.7553 - val_accuracy: 0.7072
Epoch 58/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9219 - accuracy: 0.6187 - val_loss: 0.7541 - val_accuracy: 0.7052
Epoch 59/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9012 - accuracy: 0.6192 - val_loss: 0.7485 - val_accuracy: 0.7031
Epoch 60/100
61/61 [==============================] - 0s 1ms/step - loss: 0.9063 - accuracy: 0.6269 - val_loss: 0.7465 - val_accuracy: 0.7072
Epoch 61/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8881 - accuracy: 0.6481 - val_loss: 0.7479 - val_accuracy: 0.7031
Epoch 62/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9012 - accuracy: 0.6259 - val_loss: 0.7466 - val_accuracy: 0.7155
Epoch 63/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8962 - accuracy: 0.6316 - val_loss: 0.7429 - val_accuracy: 0.7134
Epoch 64/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9087 - accuracy: 0.6280 - val_loss: 0.7442 - val_accuracy: 0.7010
Epoch 65/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8922 - accuracy: 0.6445 - val_loss: 0.7413 - val_accuracy: 0.7113
Epoch 66/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8875 - accuracy: 0.6357 - val_loss: 0.7390 - val_accuracy: 0.7134
```

```
Epoch 67/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8843 - accuracy: 0.6357 - val_loss: 0.7375 - val_accuracy: 0.7175
Epoch 68/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9019 - accuracy: 0.6290 - val_loss: 0.7386 - val_accuracy: 0.7093
Epoch 69/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8952 - accuracy: 0.6326 - val_loss: 0.7335 - val_accuracy: 0.7113
Epoch 70/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8641 - accuracy: 0.6533 - val_loss: 0.7300 - val_accuracy: 0.7093
Epoch 71/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8842 - accuracy: 0.6264 - val_loss: 0.7301 - val_accuracy: 0.7113
Epoch 72/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8781 - accuracy: 0.6429 - val_loss: 0.7320 - val_accuracy: 0.7175
Epoch 73/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8740 - accuracy: 0.6450 - val_loss: 0.7258 - val_accuracy: 0.7278
Epoch 74/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8723 - accuracy: 0.6373 - val_loss: 0.7263 - val_accuracy: 0.7216
Epoch 75/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8469 - accuracy: 0.6517 - val_loss: 0.7212 - val_accuracy: 0.7216
Epoch 76/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8766 - accuracy: 0.6352 - val_loss: 0.7216 - val_accuracy: 0.7237
Epoch 77/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8822 - accuracy: 0.6295 - val_loss: 0.7225 - val_accuracy: 0.7216
Epoch 78/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8618 - accuracy: 0.6486 - val_loss: 0.7215 - val_accuracy: 0.7175
Epoch 79/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8712 - accuracy: 0.6342 - val_loss: 0.7216 - val_accuracy: 0.7155
Epoch 80/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8583 - accuracy: 0.6460 - val_loss: 0.7217 - val_accuracy: 0.7134
Epoch 81/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8507 - accuracy: 0.6496 - val_loss: 0.7181 - val_accuracy: 0.7175
Epoch 82/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8581 - accuracy: 0.6352 - val_loss: 0.7151 - val_accuracy: 0.7093
Epoch 83/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8356 - accuracy: 0.6610 - val_loss: 0.7155 - val_accuracy: 0.7113
Epoch 84/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8366 - accuracy: 0.6440 - val_loss: 0.7159 - val_accuracy: 0.7175
Epoch 85/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8585 - accuracy: 0.6533 - val_loss: 0.7115 - val_accuracy: 0.7237
Epoch 86/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8569 - accuracy: 0.6553 - val_loss: 0.7109 - val_accuracy: 0.7175
Epoch 87/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8529 - accuracy: 0.6543 - val_loss: 0.7054 - val_accuracy: 0.7258
Epoch 88/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8511 - accuracy: 0.6460 - val_loss: 0.7017 - val_accuracy: 0.7278
```

```
Epoch 89/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8594 - accuracy: 0.6450 - val_loss: 0.7017 - val_accuracy: 0.7299
Epoch 90/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8303 - accuracy: 0.6620 - val_loss: 0.7018 - val_accuracy: 0.7278
Epoch 91/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8530 - accuracy: 0.6522 - val_loss: 0.7008 - val_accuracy: 0.7237
Epoch 92/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8468 - accuracy: 0.6481 - val_loss: 0.7017 - val_accuracy: 0.7320
Epoch 93/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8410 - accuracy: 0.6584 - val_loss: 0.6993 - val_accuracy: 0.7299
Epoch 94/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8322 - accuracy: 0.6502 - val_loss: 0.6934 - val_accuracy: 0.7258
Epoch 95/100
61/61 [==============================] - 0s 1ms/step - loss: 0.8288 - accuracy: 0.6610 - val_loss: 0.6925 - val_accuracy: 0.7237
Epoch 96/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8259 - accuracy: 0.6718 - val_loss: 0.6918 - val_accuracy: 0.7278
Epoch 97/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8293 - accuracy: 0.6563 - val_loss: 0.6921 - val_accuracy: 0.7278
Epoch 98/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8184 - accuracy: 0.6662 - val_loss: 0.6882 - val_accuracy: 0.7320
Epoch 99/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8175 - accuracy: 0.6620 - val_loss: 0.6907 - val_accuracy: 0.7278
Epoch 100/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8290 - accuracy: 0.6543 - val_loss: 0.6910 - val_accuracy: 0.7155
```

In [169…

```
y_pred_prob = model.predict(X_test_scaled)
y_pred = np.argmax(y_pred_prob, axis=1)
y_test_labels = np.argmax(y_test_categorical, axis=1)
```

```
33/33 [==============================] - 0s 741us/step
```

In [170…

```
print("Accuracy:", accuracy_score(y_test_labels, y_pred))
print("Classification Report:\n", classification_report(y_test_labels, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test_labels, y_pred))
```

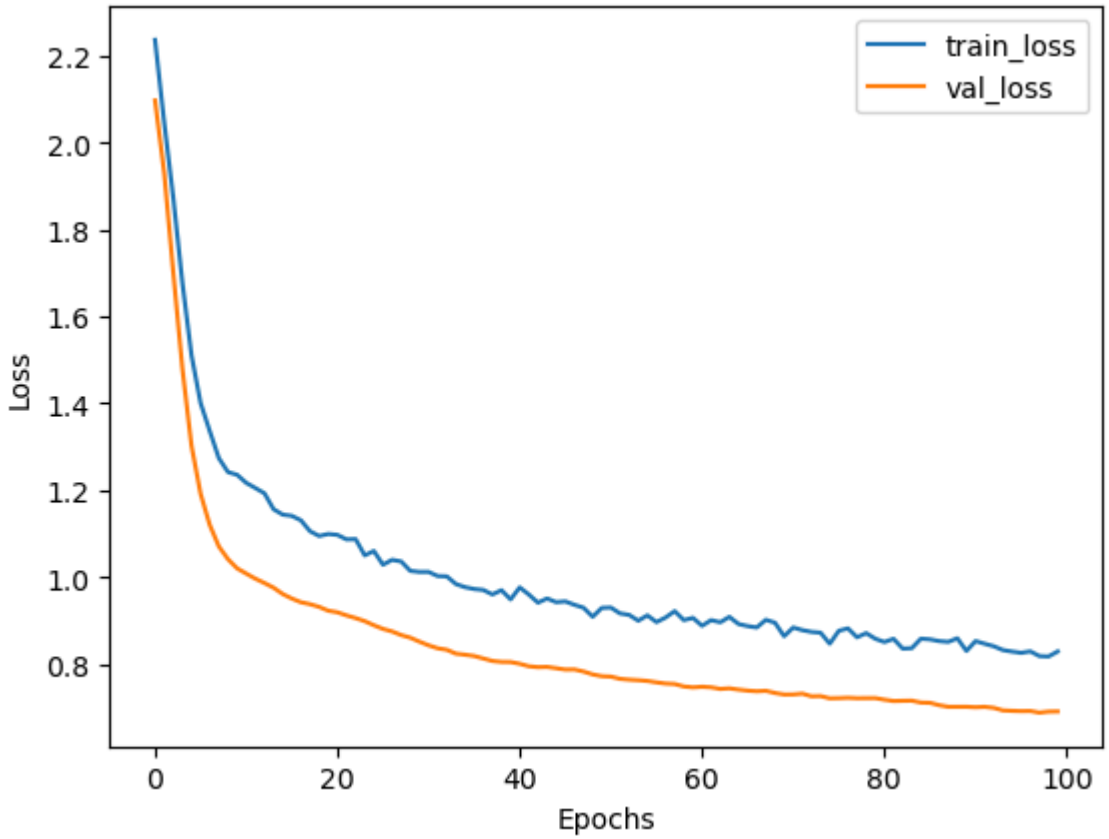
```
Accuracy: 0.6939364773820982
Classification Report:
              precision    recall  f1-score   support

           3       0.95      1.00      0.97       172
           4       0.64      0.79      0.71       147
           5       0.61      0.50      0.55       192
           6       0.45      0.44      0.45       173
           7       0.73      0.49      0.58       181
           8       0.75      0.99      0.85       174

    accuracy                           0.69      1039
   macro avg       0.69      0.70      0.69      1039
weighted avg       0.69      0.69      0.68      1039

Confusion Matrix:
 [[172   0   0   0   0   0]
 [  2 116  19  10   0   0]
 [  6  39  96  44   5   2]
 [  0  22  41  76  26   8]
 [  1   4   1  38  88  49]
 [  0   0   0   0   1 173]]
```

3C. Plot 2 separate visuals. [3 Marks]

- i.Training Loss and Validation Loss
- ii.Training Accuracy and Validation Accuracy

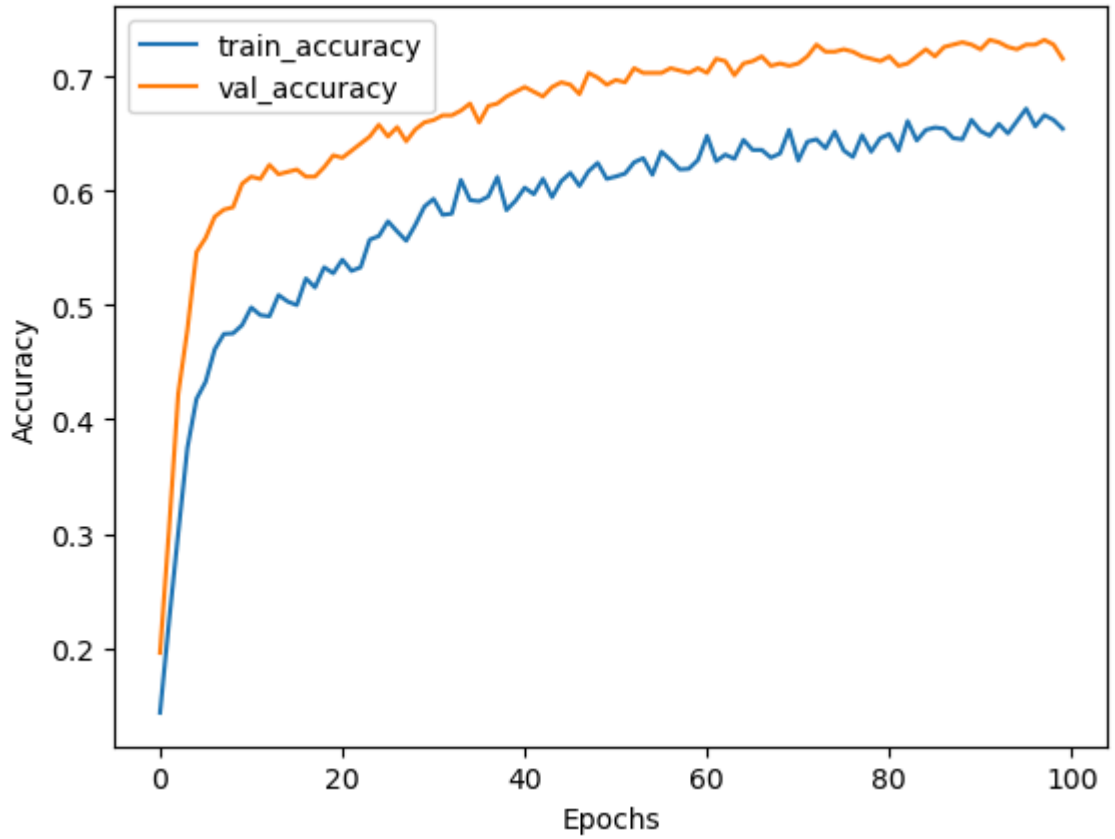
In [171…

```
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [172…

```
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



3 D. Design new architecture/update existing architecture in attempt to improve the performance of the model.[2 Marks]

In [173…
```
X_train.shape[1]
```

Out[173]:
```
11
```

In [194…
```
from tensorflow.keras.layers import Dense, Dropout
model = tf.keras.Sequential()
model.add(InputLayer(input_shape=(11,)))
model.add(Dense(128,kernel_initializer='he_normal',activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(64,kernel_initializer='he_normal',activation='relu'))
```

```
model.add(Dropout(0.2))
model.add(Dense(16,kernel_initializer='he_normal',activation='relu'))

model.add(Dense(y_train_categorical.shape[1],activation='softmax'))
```

In [195…
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [196…
```
history = model.fit(X_train_scaled, y_train_categorical, validation_split=0.2, epochs=100, batch_size=32, verbose=1)
```

```
Epoch 1/100
61/61 [==============================] - 1s 4ms/step - loss: 1.9464 - accuracy: 0.3179 - val_loss: 1.3647 - val_accuracy: 0.5299
Epoch 2/100
61/61 [==============================] - 0s 2ms/step - loss: 1.3755 - accuracy: 0.4799 - val_loss: 1.0869 - val_accuracy: 0.5794
Epoch 3/100
61/61 [==============================] - 0s 2ms/step - loss: 1.2152 - accuracy: 0.5263 - val_loss: 0.9971 - val_accuracy: 0.6082
Epoch 4/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0877 - accuracy: 0.5547 - val_loss: 0.9337 - val_accuracy: 0.6206
Epoch 5/100
61/61 [==============================] - 0s 2ms/step - loss: 1.0371 - accuracy: 0.5542 - val_loss: 0.9144 - val_accuracy: 0.6371
Epoch 6/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9655 - accuracy: 0.5960 - val_loss: 0.8747 - val_accuracy: 0.6577
Epoch 7/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9642 - accuracy: 0.6084 - val_loss: 0.8815 - val_accuracy: 0.6371
Epoch 8/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9509 - accuracy: 0.6006 - val_loss: 0.8373 - val_accuracy: 0.6557
Epoch 9/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9102 - accuracy: 0.6223 - val_loss: 0.8074 - val_accuracy: 0.6969
Epoch 10/100
61/61 [==============================] - 0s 2ms/step - loss: 0.9032 - accuracy: 0.6280 - val_loss: 0.7983 - val_accuracy: 0.6907
Epoch 11/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8777 - accuracy: 0.6331 - val_loss: 0.7894 - val_accuracy: 0.7175
Epoch 12/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8556 - accuracy: 0.6491 - val_loss: 0.7740 - val_accuracy: 0.6804
Epoch 13/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8485 - accuracy: 0.6491 - val_loss: 0.7546 - val_accuracy: 0.7093
Epoch 14/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8156 - accuracy: 0.6692 - val_loss: 0.7362 - val_accuracy: 0.7381
Epoch 15/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8143 - accuracy: 0.6620 - val_loss: 0.7295 - val_accuracy: 0.7340
Epoch 16/100
61/61 [==============================] - 0s 2ms/step - loss: 0.8045 - accuracy: 0.6723 - val_loss: 0.7352 - val_accuracy: 0.7196
Epoch 17/100
61/61 [==============================] - 0s 2ms/step - loss: 0.7766 - accuracy: 0.6780 - val_loss: 0.7147 - val_accuracy: 0.7278
Epoch 18/100
61/61 [==============================] - 0s 2ms/step - loss: 0.7747 - accuracy: 0.6883 - val_loss: 0.6998 - val_accuracy: 0.7113
Epoch 19/100
61/61 [==============================] - 0s 2ms/step - loss: 0.7557 - accuracy: 0.6976 - val_loss: 0.7036 - val_accuracy: 0.7423
Epoch 20/100
61/61 [==============================] - 0s 2ms/step - loss: 0.7499 - accuracy: 0.6976 - val_loss: 0.6928 - val_accuracy: 0.7464
Epoch 21/100
61/61 [==============================] - 0s 2ms/step - loss: 0.7539 - accuracy: 0.6956 - val_loss: 0.6755 - val_accuracy: 0.7485
Epoch 22/100
61/61 [==============================] - 0s 2ms/step - loss: 0.7341 - accuracy: 0.6920 - val_loss: 0.6814 - val_accuracy: 0.7340
```

```
Epoch 23/100
61/61 [==============================] - 0s 2ms/step - loss: 0.7183 - accuracy: 0.7110 - val_loss: 0.6811 - val_accuracy: 0.7443
Epoch 24/100
61/61 [==============================] - 0s 2ms/step - loss: 0.7052 - accuracy: 0.7198 - val_loss: 0.6800 - val_accuracy: 0.7361
Epoch 25/100
61/61 [==============================] - 0s 2ms/step - loss: 0.7199 - accuracy: 0.7157 - val_loss: 0.6555 - val_accuracy: 0.7567
Epoch 26/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6888 - accuracy: 0.7208 - val_loss: 0.6496 - val_accuracy: 0.7649
Epoch 27/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6560 - accuracy: 0.7503 - val_loss: 0.6622 - val_accuracy: 0.7588
Epoch 28/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6840 - accuracy: 0.7322 - val_loss: 0.6450 - val_accuracy: 0.7588
Epoch 29/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6760 - accuracy: 0.7245 - val_loss: 0.6289 - val_accuracy: 0.7443
Epoch 30/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6477 - accuracy: 0.7456 - val_loss: 0.6331 - val_accuracy: 0.7649
Epoch 31/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6478 - accuracy: 0.7379 - val_loss: 0.6238 - val_accuracy: 0.7691
Epoch 32/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6447 - accuracy: 0.7420 - val_loss: 0.6284 - val_accuracy: 0.7567
Epoch 33/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6332 - accuracy: 0.7477 - val_loss: 0.6152 - val_accuracy: 0.7753
Epoch 34/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6283 - accuracy: 0.7642 - val_loss: 0.6047 - val_accuracy: 0.7753
Epoch 35/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6321 - accuracy: 0.7518 - val_loss: 0.6128 - val_accuracy: 0.7691
Epoch 36/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6175 - accuracy: 0.7518 - val_loss: 0.6124 - val_accuracy: 0.7670
Epoch 37/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6108 - accuracy: 0.7472 - val_loss: 0.6220 - val_accuracy: 0.7402
Epoch 38/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6183 - accuracy: 0.7549 - val_loss: 0.6116 - val_accuracy: 0.7443
Epoch 39/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5772 - accuracy: 0.7642 - val_loss: 0.5918 - val_accuracy: 0.7649
Epoch 40/100
61/61 [==============================] - 0s 2ms/step - loss: 0.6002 - accuracy: 0.7642 - val_loss: 0.5907 - val_accuracy: 0.7691
Epoch 41/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5662 - accuracy: 0.7776 - val_loss: 0.5917 - val_accuracy: 0.7588
Epoch 42/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5773 - accuracy: 0.7724 - val_loss: 0.5946 - val_accuracy: 0.7670
Epoch 43/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5660 - accuracy: 0.7724 - val_loss: 0.5802 - val_accuracy: 0.7794
Epoch 44/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5682 - accuracy: 0.7761 - val_loss: 0.5827 - val_accuracy: 0.7567
```

```
Epoch 45/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5771 - accuracy: 0.7534 - val_loss: 0.5874 - val_accuracy: 0.7629
Epoch 46/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5505 - accuracy: 0.7869 - val_loss: 0.5711 - val_accuracy: 0.7691
Epoch 47/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5603 - accuracy: 0.7828 - val_loss: 0.5901 - val_accuracy: 0.7670
Epoch 48/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5417 - accuracy: 0.7859 - val_loss: 0.5831 - val_accuracy: 0.7567
Epoch 49/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5483 - accuracy: 0.7874 - val_loss: 0.5834 - val_accuracy: 0.7608
Epoch 50/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5309 - accuracy: 0.7853 - val_loss: 0.5678 - val_accuracy: 0.7753
Epoch 51/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5270 - accuracy: 0.7972 - val_loss: 0.5548 - val_accuracy: 0.7711
Epoch 52/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5467 - accuracy: 0.7792 - val_loss: 0.5569 - val_accuracy: 0.7608
Epoch 53/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5362 - accuracy: 0.7853 - val_loss: 0.5667 - val_accuracy: 0.7711
Epoch 54/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5417 - accuracy: 0.7807 - val_loss: 0.5672 - val_accuracy: 0.7691
Epoch 55/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5134 - accuracy: 0.7884 - val_loss: 0.5658 - val_accuracy: 0.7691
Epoch 56/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5098 - accuracy: 0.7931 - val_loss: 0.5681 - val_accuracy: 0.7670
Epoch 57/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5061 - accuracy: 0.7874 - val_loss: 0.5598 - val_accuracy: 0.7711
Epoch 58/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5252 - accuracy: 0.7895 - val_loss: 0.5516 - val_accuracy: 0.7938
Epoch 59/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5115 - accuracy: 0.7977 - val_loss: 0.5764 - val_accuracy: 0.7546
Epoch 60/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5156 - accuracy: 0.7972 - val_loss: 0.5692 - val_accuracy: 0.7753
Epoch 61/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4743 - accuracy: 0.8050 - val_loss: 0.5470 - val_accuracy: 0.7794
Epoch 62/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4846 - accuracy: 0.8132 - val_loss: 0.5545 - val_accuracy: 0.7794
Epoch 63/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4838 - accuracy: 0.8008 - val_loss: 0.5541 - val_accuracy: 0.7897
Epoch 64/100
61/61 [==============================] - 0s 2ms/step - loss: 0.5009 - accuracy: 0.8008 - val_loss: 0.5463 - val_accuracy: 0.7711
Epoch 65/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4918 - accuracy: 0.8029 - val_loss: 0.5613 - val_accuracy: 0.7773
Epoch 66/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4959 - accuracy: 0.7972 - val_loss: 0.5502 - val_accuracy: 0.7835
```

```
Epoch 67/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4939 - accuracy: 0.8019 - val_loss: 0.5472 - val_accuracy: 0.7732
Epoch 68/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4796 - accuracy: 0.8070 - val_loss: 0.5381 - val_accuracy: 0.7856
Epoch 69/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4559 - accuracy: 0.8199 - val_loss: 0.5583 - val_accuracy: 0.7773
Epoch 70/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4615 - accuracy: 0.8148 - val_loss: 0.5372 - val_accuracy: 0.7794
Epoch 71/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4646 - accuracy: 0.8034 - val_loss: 0.5292 - val_accuracy: 0.7835
Epoch 72/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4584 - accuracy: 0.8179 - val_loss: 0.5429 - val_accuracy: 0.7773
Epoch 73/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4521 - accuracy: 0.8173 - val_loss: 0.5402 - val_accuracy: 0.7835
Epoch 74/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4496 - accuracy: 0.8246 - val_loss: 0.5340 - val_accuracy: 0.7897
Epoch 75/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4388 - accuracy: 0.8209 - val_loss: 0.5288 - val_accuracy: 0.7794
Epoch 76/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4350 - accuracy: 0.8246 - val_loss: 0.5467 - val_accuracy: 0.7814
Epoch 77/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4464 - accuracy: 0.8127 - val_loss: 0.5286 - val_accuracy: 0.7897
Epoch 78/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4436 - accuracy: 0.8194 - val_loss: 0.5358 - val_accuracy: 0.7773
Epoch 79/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4439 - accuracy: 0.8246 - val_loss: 0.5625 - val_accuracy: 0.7711
Epoch 80/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4546 - accuracy: 0.8225 - val_loss: 0.5646 - val_accuracy: 0.7753
Epoch 81/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4391 - accuracy: 0.8277 - val_loss: 0.5488 - val_accuracy: 0.7794
Epoch 82/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4254 - accuracy: 0.8328 - val_loss: 0.5502 - val_accuracy: 0.7773
Epoch 83/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4393 - accuracy: 0.8220 - val_loss: 0.5492 - val_accuracy: 0.7794
Epoch 84/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4281 - accuracy: 0.8251 - val_loss: 0.5517 - val_accuracy: 0.7794
Epoch 85/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4096 - accuracy: 0.8473 - val_loss: 0.5504 - val_accuracy: 0.7794
Epoch 86/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4177 - accuracy: 0.8318 - val_loss: 0.5395 - val_accuracy: 0.7732
Epoch 87/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4244 - accuracy: 0.8271 - val_loss: 0.5300 - val_accuracy: 0.7918
Epoch 88/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4250 - accuracy: 0.8344 - val_loss: 0.5412 - val_accuracy: 0.7794
```

```
Epoch 89/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4004 - accuracy: 0.8375 - val_loss: 0.5707 - val_accuracy: 0.7897
Epoch 90/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4261 - accuracy: 0.8235 - val_loss: 0.5579 - val_accuracy: 0.7876
Epoch 91/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4269 - accuracy: 0.8240 - val_loss: 0.5512 - val_accuracy: 0.7753
Epoch 92/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4197 - accuracy: 0.8338 - val_loss: 0.5140 - val_accuracy: 0.7897
Epoch 93/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4216 - accuracy: 0.8349 - val_loss: 0.5355 - val_accuracy: 0.7897
Epoch 94/100
61/61 [==============================] - 0s 2ms/step - loss: 0.3952 - accuracy: 0.8411 - val_loss: 0.5343 - val_accuracy: 0.7938
Epoch 95/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4020 - accuracy: 0.8328 - val_loss: 0.5358 - val_accuracy: 0.7876
Epoch 96/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4075 - accuracy: 0.8380 - val_loss: 0.5275 - val_accuracy: 0.7794
Epoch 97/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4054 - accuracy: 0.8277 - val_loss: 0.5459 - val_accuracy: 0.7938
Epoch 98/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4181 - accuracy: 0.8328 - val_loss: 0.5673 - val_accuracy: 0.7856
Epoch 99/100
61/61 [==============================] - 0s 2ms/step - loss: 0.3933 - accuracy: 0.8364 - val_loss: 0.5498 - val_accuracy: 0.8041
Epoch 100/100
61/61 [==============================] - 0s 2ms/step - loss: 0.4072 - accuracy: 0.8395 - val_loss: 0.5218 - val_accuracy: 0.7856
```

In [205…

```python
y_pred_prob = model.predict(X_test_scaled)
y_pred = np.argmax(y_pred_prob, axis=1)
y_test_labels = np.argmax(y_test_categorical, axis=1)
```

```
33/33 [==============================] - 0s 776us/step
```

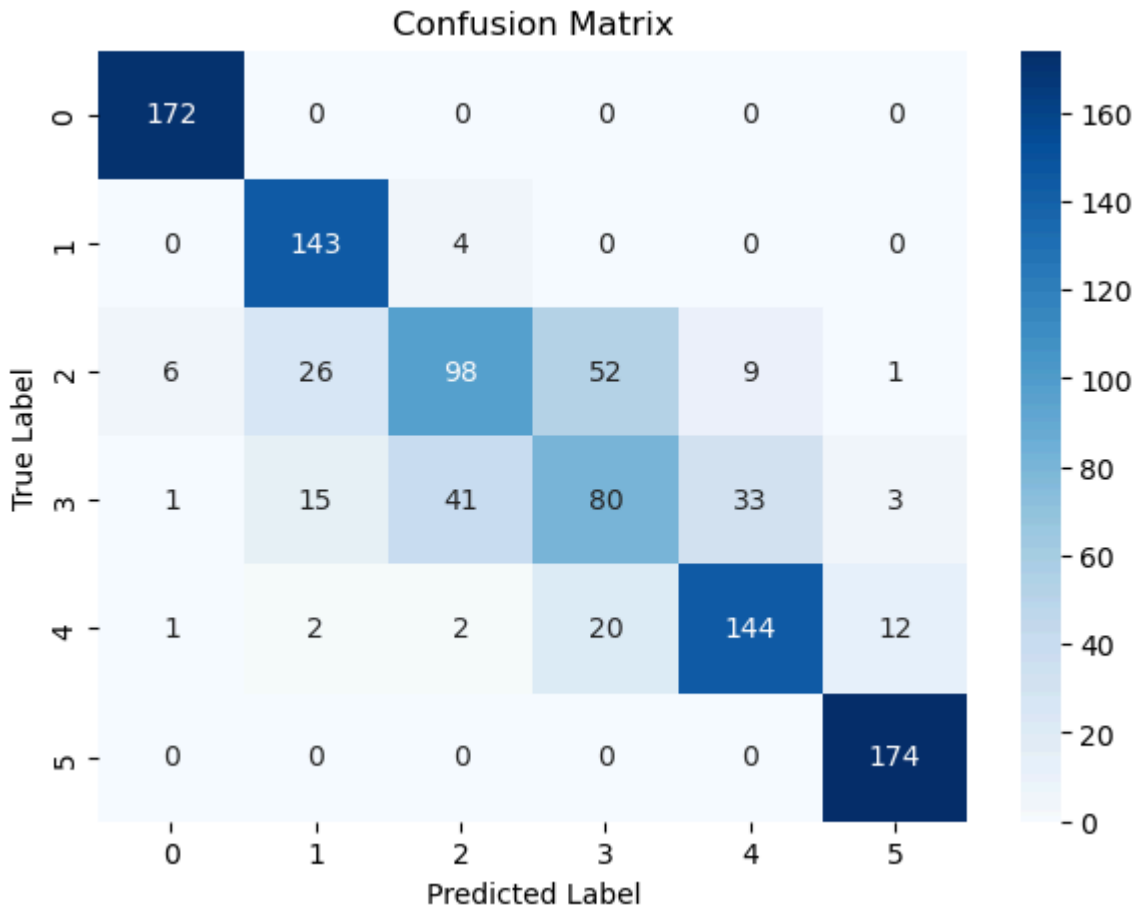
In [206…

```python
print("Accuracy:", accuracy_score(y_test_labels, y_pred))
print("Classification Report:\n", classification_report(y_test_labels, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test_labels, y_pred))
```

```
Accuracy: 0.78055822906641
Classification Report:
              precision    recall  f1-score   support

           3       0.96      1.00      0.98       172
           4       0.77      0.97      0.86       147
           5       0.68      0.51      0.58       192
           6       0.53      0.46      0.49       173
           7       0.77      0.80      0.78       181
           8       0.92      1.00      0.96       174

    accuracy                           0.78      1039
   macro avg       0.77      0.79      0.78      1039
weighted avg       0.77      0.78      0.77      1039

Confusion Matrix:
 [[172   0   0   0   0   0]
 [  0 143   4   0   0   0]
 [  6  26  98  52   9   1]
 [  1  15  41  80  33   3]
 [  1   2   2  20 144  12]
 [  0   0   0   0   0 174]]
```

In [209…

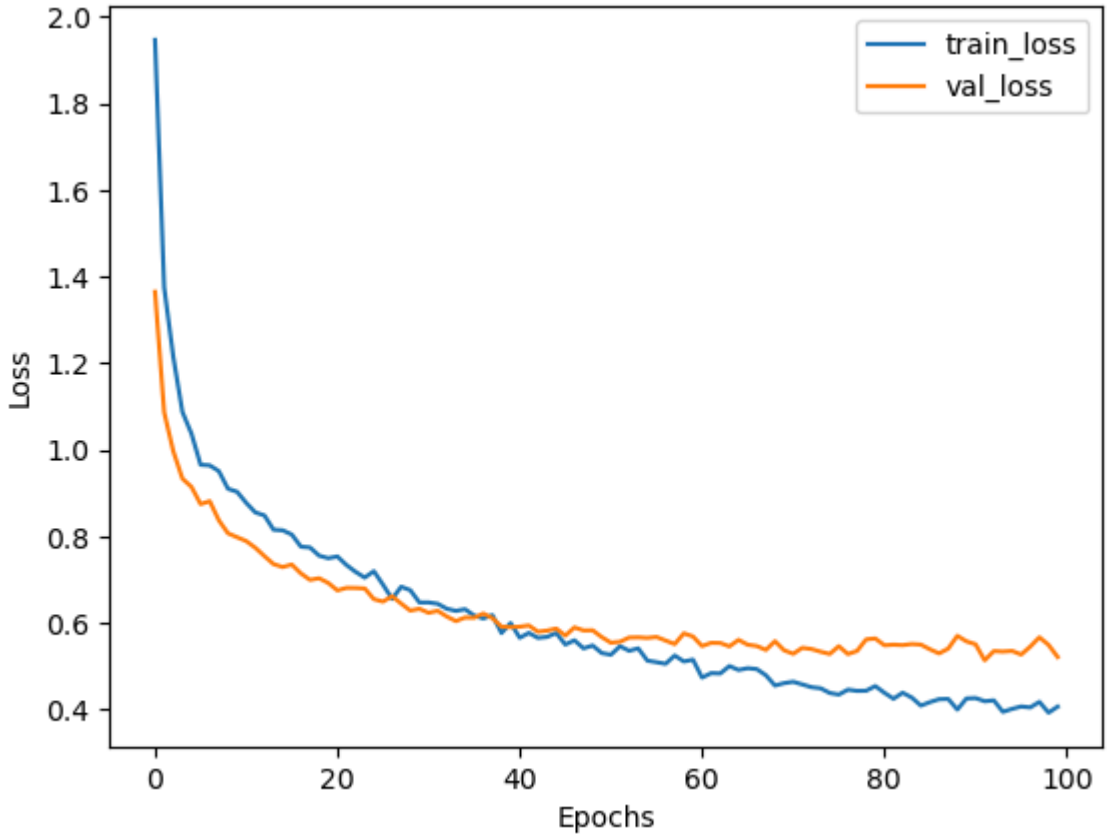
```python
cm=confusion_matrix(y_test_labels, y_pred)
plt.figure(figsize=(7, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=range(6), yticklabels=range(6))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



3 E. Plot visuals as in Q3.C and share insights about difference observed in both the models. [3 Marks]

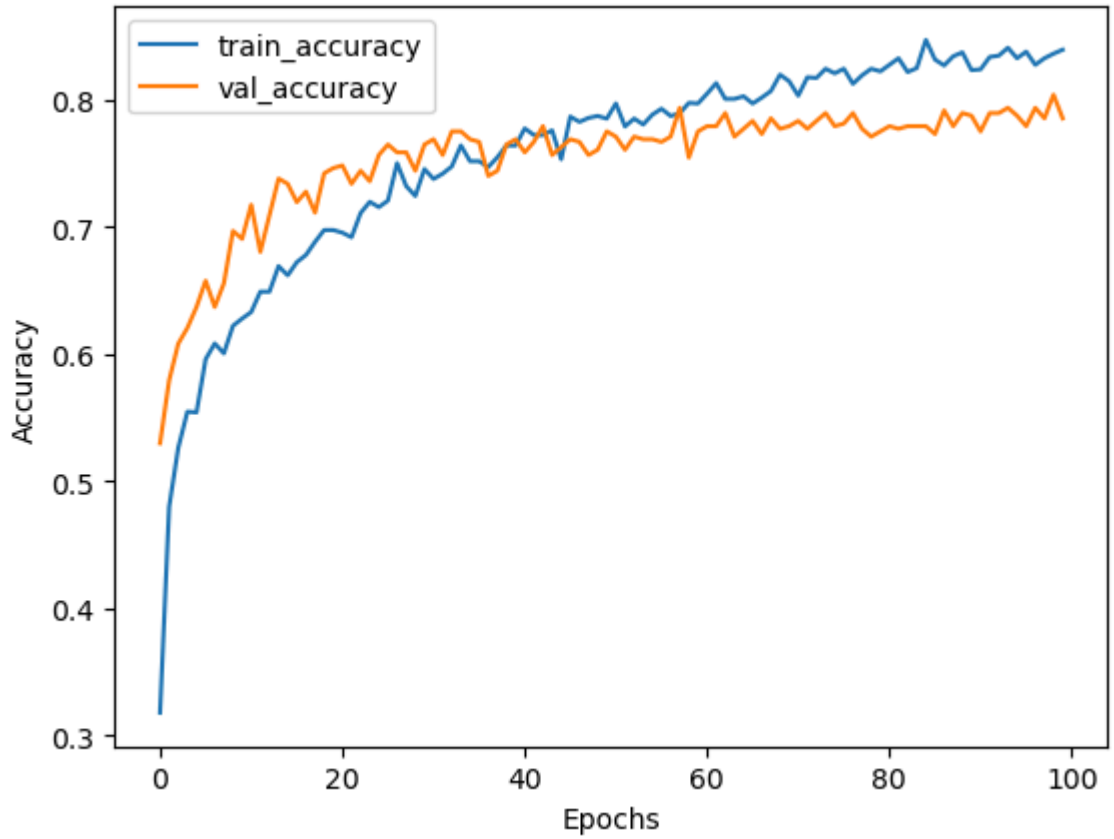
In [199…

```
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



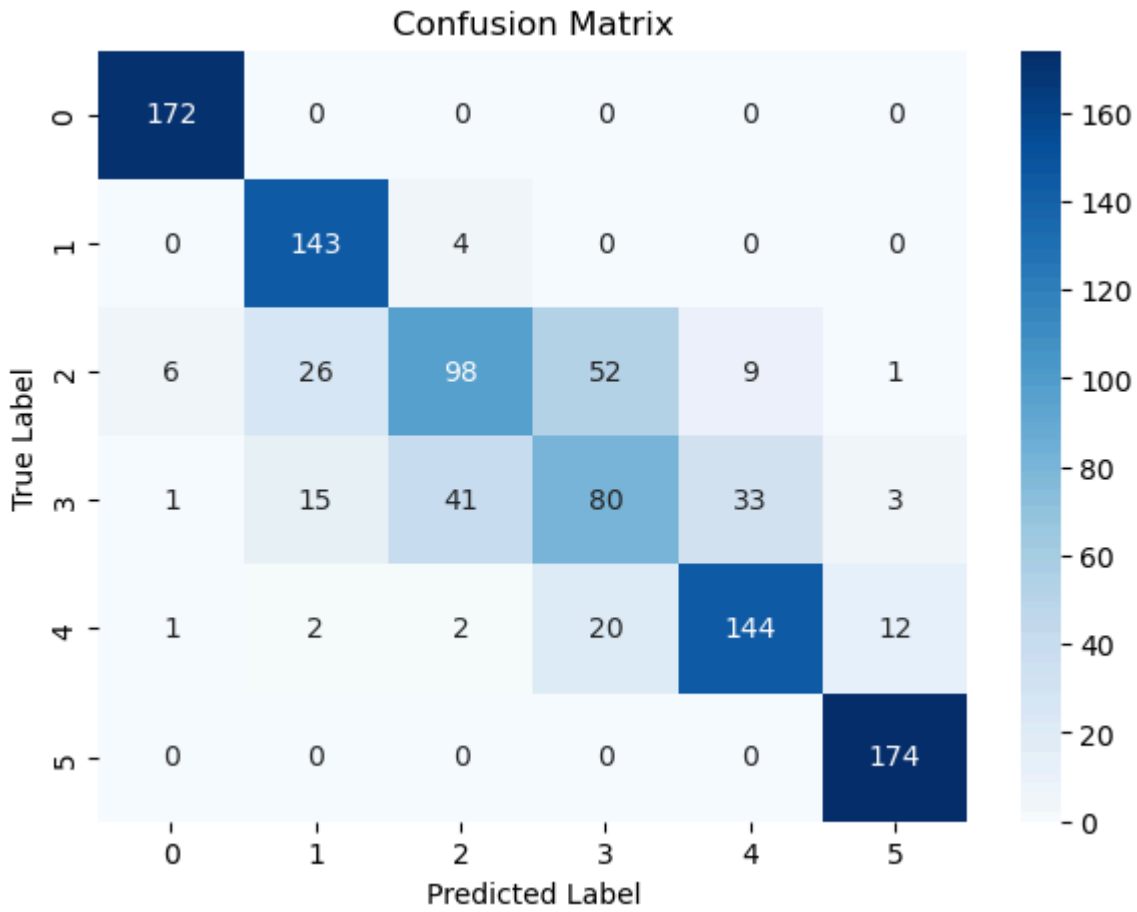
In [200…

```
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



In [210…

```
cm=confusion_matrix(y_test_labels, y_pred)
plt.figure(figsize=(7, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=range(6), yticklabels=range(6))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```



insights

first model

- in first model we used 3 layers
- in first model in first layer we used 32 neurons and we drop the number of nuerons to avoid overfitting
- in second layer we used 8 neurons and we drop the number of nuerons to avoid overfitting
- in last layer is output has six neuron all of the above we performed relu as activation function accept output we had perform softmax
- we found weight were randomly initiaised

- Accuracy was about 69% and loss was about 0.82 also difference between validation and train accuracy was observed

second model

- we not only increase the number of neurons but also increase number of layers
- here we have inital weight parameters using He methods that with kernal initialiser
- and we found that accuracy increse to 79% and loss reduced to 0.40
- so conclusion is definately model two is better then first model

========== PART-A END==========

Part B - 30 Marks

DOMAIN: Autonomous Vehicles

- **CONTEXT:** A Recognising multi-digit numbers in photographs captured at street level is an important component of modern-day map making. A classic example of a corpus of such street-level photographs is Google's Street View imagery composed of hundreds of millions of geo-located 360-degree panoramic images.The ability to automatically transcribe an address number from a geo-located patch of pixels and associate the transcribed number with a known street address helps pinpoint, with a high degree of accuracy, the location of the building it represents. More broadly, recognising numbers in photographs is a problem of interest to the optical character recognition community. While OCR on constrained domains like document processing is well studied, arbitrary multi-character text recognition in photographs is still highly challenging. This difficulty arises due to the wide variability in the visual appearance of text in the wild on account of a large range of fonts, colours, styles, orientations, and character arrangements.The recognition problem is further complicated by environmental factors such as lighting, shadows, specularity, and occlusions as well as by image acquisition factors such as resolution, motion, and focus blurs. In this project, we will use the dataset with images centred around a single digit (many of the images do contain some distractors at the sides). Although we are taking a sample of the data which is simpler,it is more complex than MNIST because of the distractors.

- **DATA DESCRIPTION:** The SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with the minimal requirement on data formatting but comes from a significantly harder, unsolved, real-world problem (recognising digits and

numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.Where the labels for each of this image are the prominent number in that image i.e. 2,6,7 and 4 respectively.The dataset has been provided in the form of h5py files. You can read about this file format here: https://docs.h5py.org/en/stable/

- **Acknowledgement:** Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading Digits in Natural Images with Unsupervised Feature Learning NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011. PDF http://ufldl.stanford.edu/housenumbers as the URL for this site.
- **PROJECT OBJECTIVE:** To build a digit classifier on the SVHN (Street View Housing Number) dataset.

In [249…
```
import h5py
```

Steps and tasks: [Total Score: 30 Marks]

1.Data Import and Exploration [5 Marks]

1 A. Read the .h5 file and assign to a variable. [2 Marks]

In [250…
```
h5f = h5py.File("E:\\Greatlearning- Projects\\AIML project - ANN\\Autonomous_Vehicles_SVHN_single_grey1 (3).h5",'r')
```

1 B. Print all the keys from the .h5 file. [1 Marks]

In [258…
```
print("the keys from h5 file are ",h5f.keys())
```

```
the keys from h5 file are  <KeysViewHDF5 ['X_test', 'X_train', 'X_val', 'y_test', 'y_train', 'y_val']>
```

1 C. Split the data into X_train, X_test, Y_train, Y_test [2 Marks]

In [259…
```
X_train = h5f['X_train'][:]
y_train = h5f['y_train'][:]

X_val = h5f['X_val'][:]
y_val = h5f['y_val'][:]
```

```
X_test = h5f['X_test'][:]
y_test = h5f['y_test'][:]
```

2. Data Visualisation and preprocessing [13 Marks]

2 A. Print shape of all the 4 data split into x, y, train, test to verify if x & y is in sync. [1 Marks]

In [260…

```
print(f'Size of X_train is {X_train.shape}')
print(f'Size of y_train is {y_train.shape}\n')

print(f'Size of X_val is {X_val.shape}')
print(f'Size of y_val is {y_val.shape}\n')

print(f'Size of X_test is {X_test.shape}')
print(f'Size of y_test is {y_test.shape}')
```

```
Size of X_train is (42000, 32, 32)
Size of y_train is (42000,)

Size of X_val is (60000, 32, 32)
Size of y_val is (60000,)

Size of X_test is (18000, 32, 32)
Size of y_test is (18000,)
```

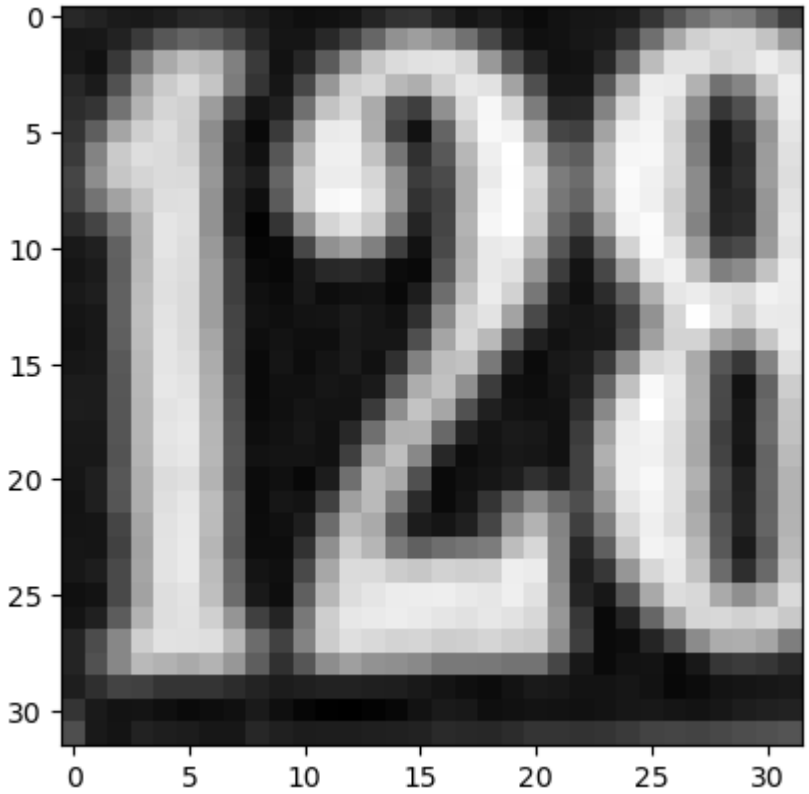
- The training dataset(X_train) has 42k records on which we can train upon of matrix size of 32x32 i.e. image size of 32x32.
- The test dataset(X_test) has 18k records each record being 32x32 in size.
- y_train, y_test contain label for the given image matrix.

2 B. Visualise first 10 images in train data and print its corresponding labels. [4 Marks]

In [261…

```
plt.imshow(X_train[0],cmap='gray')
print(f'Label for the image is {y_train[1]}')
```

```
Label for the image is 6
```



- We can clearly see that we have a partial view of digit 3, that is Noise

In [262…

```
# visualizing the first 10 images in the dataset and their labels
plt.figure(figsize=(10, 1))
for i in range(10):
    plt.subplot(1, 10, i+1)
    plt.imshow(X_train[i], cmap="gray")
    plt.axis('off')
    print('label for each of the below image: %s' % ((y_train[i])))
plt.show()
```

```
label for each of the below image: 2
label for each of the below image: 6
label for each of the below image: 7
label for each of the below image: 4
label for each of the below image: 4
label for each of the below image: 0
label for each of the below image: 3
label for each of the below image: 0
label for each of the below image: 7
label for each of the below image: 3
```

- thus we can conclude there are lots of noise in data images

2C.Reshape the imgs with appropriate shape update the data in same variable[3 Marks]

In [263…

```
# RESHAPE 2D - 32*32 into 1D - 1024
X_train = X_train.reshape(X_train.shape[0], 32*32)
X_val= X_val.reshape(X_val.shape[0], 32*32)
X_test = X_test.reshape(X_test.shape[0],32*32)

print(f'Shape of X_train is {X_train.shape}')
print(f'Shape of X_val is {X_val.shape}')
print(f'Shape of X_test is {X_test.shape}')
```

```
Shape of X_train is (42000, 1024)
Shape of X_val is (60000, 1024)
Shape of X_test is (18000, 1024)
```

2 D. Normalise the images i.e. Normalise the pixel values. [2 Marks]

In [264…

```
print(f'Min value for Train = {X_train.min()},Validation ={X_val.min()}, Test = {X_test.min()} ')
print(f'Min value for Train = {X_train.max()},Validation ={X_val.max()}, Test = {X_test.max()} ')
```

```
Min value for Train = 0.0,Validation =0.0, Test = 0.0
Min value for Train = 254.97450256347656,Validation =254.97450256347656, Test = 254.97450256347656
```

In [265…

```
print('Before Normalization')
print(f'Min value is {X_train.min()}')
print(f'Max value is {X_train.max()}\n')
maxVal=X_train.max()
X_train = X_train/maxVal
X_val= X_val/maxVal
X_test = X_test/maxVal

print('After Normalization')
print(f'Min value is {X_train.min()}')
print(f'Max value is {X_train.max()}')
```

```
Before Normalization
Min value is 0.0
Max value is 254.97450256347656

After Normalization
Min value is 0.0
Max value is 1.0
```

2 E. Transform Labels into format acceptable by Neural Network [2 Marks]

In [266…

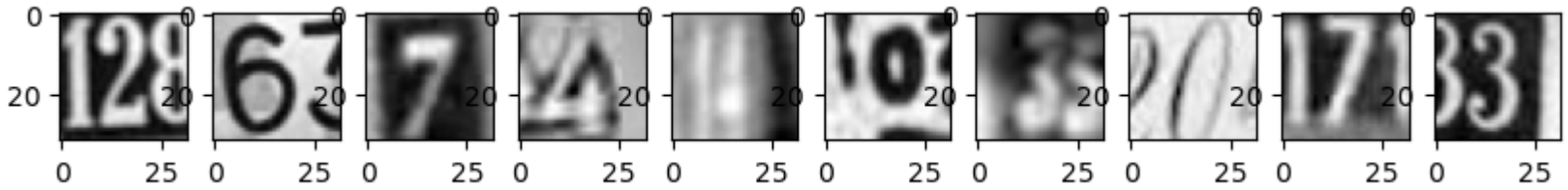
```
import tensorflow
print(f'Sample value before one hot encode {y_train[0]}\n')
y_train = tensorflow.keras.utils.to_categorical(y_train,num_classes=10)
y_val= tensorflow.keras.utils.to_categorical(y_val,num_classes=10)
y_test= tensorflow.keras.utils.to_categorical(y_test, num_classes=10)
print(f'Sample value after one hot encode {y_train[0]}')
```

```
Sample value before one hot encode 2

Sample value after one hot encode [0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

In [267…

```
#cross check if we did all right
plt.figure(figsize=(10,1))
for i in range(10):
    plt.subplot(1,10,i+1)
    plt.imshow(X_train[i].reshape(32,32),cmap='gray')
    print(f'Label for image at index {i+1} is {np.argmax(y_train[0:10][i])}')
```

```
Label for image at index 1 is 2
Label for image at index 2 is 6
Label for image at index 3 is 7
Label for image at index 4 is 4
Label for image at index 5 is 4
Label for image at index 6 is 0
Label for image at index 7 is 3
Label for image at index 8 is 0
Label for image at index 9 is 7
Label for image at index 10 is 3
```



2F.Print total Number of classes in the Dataset. [1 Marks]

In [268…

```
print("total number of classes are",len(y_train[0]))
```

```
total number of classes are 10
```

3. Model Training & Evaluation using Neural Network [12 Marks]

3A. Design a Neural Network to train a classifier. [3 Marks]

In [269…

```
def model(iterations, lr, Lambda, verb=0, eval_test=False):
    scores=[]
    learning_rate=lr
    hidden_nodes=256
    output_nodes=10
    iterations=iterations
    # For early stopping of model.
    callbacks=tensorflow.keras.callbacks.EarlyStopping(monitor='loss', patience=5)
    #model
    model = Sequential()
    model.add(Dense(500, input_shape=(1024,), activation='relu'))
```

```
    model.add(BatchNormalization())
    model.add(Dense(hidden_nodes,activation='relu'))
    model.add(Dense(hidden_nodes,activation='relu'))
    model.add(Dropout(0.5))

    model.add(Dense(hidden_nodes,activation='relu'))
    model.add(Dense(hidden_nodes,activation='relu'))
    model.add(Dropout(0.5))

    model.add(Dense(hidden_nodes,activation='relu'))
    model.add(Dense(hidden_nodes,activation='relu'))
    model.add(Dense(output_nodes, activation='softmax', kernel_regularizer=regularizers.l2(Lambda)))
    # adam optmizer with custom learning rate
    adam= optimizers.Adam(lr=learning_rate)
    #Compile the model
    model.compile(loss='categorical_crossentropy', optimizer=adam, metrics=['accuracy'])

    #Fit the model
    model.fit(X_train,y_train, validation_data=(X_val,y_val),epochs=iterations,
              batch_size=500, verbose=verb, callbacks=[callbacks])

    if eval_test == True:
        score = model.evaluate(X_train,y_train, verbose=0)
        scores.append(score)
        score = model.evaluate(X_val,y_val, verbose=0)
        scores.append(score)
        score = model.evaluate(X_test,y_test, verbose=0)
        scores.append(score)
        return scores
    else:
        score = model.evaluate(X_val,y_val, verbose=(verb+1)%2)
        return score
```

In [270…

```
#Let us try with very low learning rate and zero regularization
iterations = 1
lr=0.0001
Lambda=0
score=model(iterations, lr, Lambda)
print(f'\nLoss is {score[0]} and Accuracy is {score[1]}')
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
1875/1875 [==============================] - 4s 2ms/step - loss: 1.9203 - accuracy: 0.2934

Loss is 1.9202667474746704 and Accuracy is 0.2933500111103058
```

- Loss is pretty resonable and so is the accuracy which is probability of being a certain digit among 10 classes is 10%(equal for all classes).
- Loss is here calculated via the softmax and crossentropy which is basically -y.ln(0.10) .

3B. Train the classifier using previously designed Architecture (Use best suitable parameters). [3 Marks]

Lets try Increasing learning Rate, that should increase our loss, this will act as cross validation

In [271…
```
iterations = 1
lr=1e3
Lambda=0
score=model(iterations, lr, Lambda)
print(f'\nLoss is {score[0]} and Accuracy is {score[1]}')
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 1.8560 - accuracy: 0.3348

Loss is 1.8560149669647217 and Accuracy is 0.3348333239555359
```

In [272…
```
#Let's narrow down our search a bit
iterations = 50
lr=1e-4
Lambda=1e-7
score=model(iterations, lr, Lambda)
print(f'Loss is {score[0]} and Accuracy is {score[1]}')
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 5s 2ms/step - loss: 0.6192 - accuracy: 0.8201
Loss is 0.6191679835319519 and Accuracy is 0.8201333284378052
```

In [273…
```
iterations = 10
lr=2
Lambda=1e-2
```

```
score=model(iterations, lr, Lambda)
print(f'Loss is {score[0]} and Accuracy is {score[1]}')
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 5s 3ms/step - loss: 0.9706 - accuracy: 0.6939
Loss is 0.9705992937088013 and Accuracy is 0.6939333081245422
```

In [274…

```
import math
results =[]
for i in range(10):
    lr=math.pow(10, np.random.uniform(-4.0,1.0))
    Lambda = math.pow(10, np.random.uniform(-7,-2))
    iterations = 30
    score=model(iterations, lr, Lambda)
    result=f'Loss is {score[0]} and Accuracy is {score[1]} with learing rate {lr} and Lambda {Lambda}\n'
    print(result)
    results.append(result)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 5s 2ms/step - loss: 0.8820 - accuracy: 0.7369
Loss is 0.881957471370697 and Accuracy is 0.7369333505630493 with learing rate 0.1272276031978374 and Lambda 3.8422619683776555e-05

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.5191 - accuracy: 0.8396
Loss is 0.5190871357917786 and Accuracy is 0.8395833373069763 with learing rate 0.0009693361992970123 and Lambda 8.168544848043383e-06

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 5s 2ms/step - loss: 0.5871 - accuracy: 0.8170
Loss is 0.5871127843856812 and Accuracy is 0.8169666528701782 with learing rate 0.040175001985780175 and Lambda 2.329240816728559e-05

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 5s 2ms/step - loss: 0.5555 - accuracy: 0.8314
Loss is 0.5554807186126709 and Accuracy is 0.8313833475112915 with learing rate 0.0004482197892192248 and Lambda 1.7947988969210727e-06
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 5s 2ms/step - loss: 0.6851 - accuracy: 0.7905
Loss is 0.6850674748420715 and Accuracy is 0.7904999852180481 with learing rate 0.003919771478277612 and Lambda 0.0027852699305495722

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.5732 - accuracy: 0.8208
Loss is 0.5731725096702576 and Accuracy is 0.8207833170890808 with learing rate 1.2033607714175503 and Lambda 6.953893150948777e-07

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.5897 - accuracy: 0.8183
Loss is 0.5896844267845154 and Accuracy is 0.8183000087738037 with learing rate 0.0006288195695421728 and Lambda 0.0012101153904055394

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.5987 - accuracy: 0.8112
Loss is 0.5987184047698975 and Accuracy is 0.8112499713897705 with learing rate 0.0003020698282880314 and Lambda 1.7015356976469995e-06

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.6077 - accuracy: 0.8160
Loss is 0.6076666116714478 and Accuracy is 0.8159833550453186 with learing rate 4.286184014228777 and Lambda 0.005980063887659781 5

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.8032 - accuracy: 0.7598
Loss is 0.803195595741272 and Accuracy is 0.7598333358764648 with learing rate 0.01258912207094392 and Lambda 4.876954101512577e-07
```

In [ ]:

We get good results in range

- Learning Rate = "0.008 to 0.002"
- Lambda = 1e-3 to 1e-5

In [275…

```
import math
results =[]
for i in range(20):
    lr=math.pow(10, np.random.uniform(-4.0,-2.0))
    Lambda = math.pow(10, np.random.uniform(-5,-3))
    iterations = 50
    score=model(iterations, lr, Lambda)
    result=f'Loss is {score[0]} and Accuracy is {score[1]} with learing rate {lr} and Lambda {Lambda}\n'
    print(result)
    results.append([result,[score[0],score[1],lr,Lambda]])
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.5100 - accuracy: 0.8471
Loss is 0.5099942088127136 and Accuracy is 0.847083330154419 with learing rate 0.00021061021047707207 and Lambda 0.0001190291576
2583013

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 5s 2ms/step - loss: 0.7103 - accuracy: 0.8025
Loss is 0.7102686762809753 and Accuracy is 0.8025166392326355 with learing rate 0.0003256365033897408 and Lambda 1.4671595333076
967e-05

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 5s 2ms/step - loss: 0.5026 - accuracy: 0.8505
Loss is 0.5025941729545593 and Accuracy is 0.8504833579063416 with learing rate 0.0003367918978489774 and Lambda 0.0004038312507
598037

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 5s 2ms/step - loss: 0.6195 - accuracy: 0.8213
Loss is 0.619534969329834 and Accuracy is 0.8212500214576721 with learing rate 0.0009487431604326171 and Lambda 0.00013611893109
238376

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
1875/1875 [==============================] - 5s 2ms/step - loss: 0.5191 - accuracy: 0.8439
Loss is 0.5191420912742615 and Accuracy is 0.8438500165939331 with learing rate 0.006616521432684205 and Lambda 1.1421389653916849e-05

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 5s 2ms/step - loss: 0.5174 - accuracy: 0.8441
Loss is 0.5173884034156799 and Accuracy is 0.8441166877746582 with learing rate 0.002259758045344023 and Lambda 0.00010656538031698523

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 5s 2ms/step - loss: 0.5325 - accuracy: 0.8373
Loss is 0.5325440168380737 and Accuracy is 0.8373166918754578 with learing rate 0.004567966129578864 and Lambda 0.00034302500258721777

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.7754 - accuracy: 0.7914
Loss is 0.7754125595092773 and Accuracy is 0.7914333343505859 with learing rate 0.00016249565354051922 and Lambda 0.000356521181493521

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.5343 - accuracy: 0.8384
Loss is 0.5342570543289185 and Accuracy is 0.8383833169937134 with learing rate 0.005492616343484901 and Lambda 0.0006580418587360134

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.5127 - accuracy: 0.8432
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
Loss is 0.5127094984054565 and Accuracy is 0.8432499766349792 with learing rate 0.0007893305465742248 and Lambda 0.0009481770540149066

1875/1875 [==============================] - 4s 2ms/step - loss: 0.4400 - accuracy: 0.8674
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
Loss is 0.4400194585323334 and Accuracy is 0.8674499988555908 with learing rate 0.00013193195628333483 and Lambda 0.000387390098
66279944

1875/1875 [==============================] - 4s 2ms/step - loss: 0.5181 - accuracy: 0.8483
Loss is 0.518135666847229 and Accuracy is 0.8482833504676819 with learing rate 0.002298717078916174 and Lambda 0.000225569437254
08538

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimi
zers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4612 - accuracy: 0.8637
Loss is 0.4612056314945221 and Accuracy is 0.8637499809265137 with learing rate 0.0012855965273070409 and Lambda 0.0004860768767
023969

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimi
zers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4617 - accuracy: 0.8648
Loss is 0.4616778790950775 and Accuracy is 0.8648166656494141 with learing rate 0.0005982262022588651 and Lambda 3.4677295368084
63e-05

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimi
zers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4178 - accuracy: 0.8731
Loss is 0.41781917214393616 and Accuracy is 0.8730666637420654 with learing rate 0.0009322432109375923 and Lambda 9.008710543993
556e-05

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimi
zers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.5463 - accuracy: 0.8343
Loss is 0.5462964177131653 and Accuracy is 0.8343333601951599 with learing rate 0.00033631662504697297 and Lambda 1.215804363662
8907e-05

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimi
zers.legacy.Adam.
1875/1875 [==============================] - 5s 2ms/step - loss: 0.5187 - accuracy: 0.8473
Loss is 0.5186550617218018 and Accuracy is 0.8473166823387146 with learing rate 0.008619886749041213 and Lambda 9.75896498918679
3e-05

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimi
zers.legacy.Adam.
```

```
1875/1875 [==============================] - 5s 2ms/step - loss: 0.4904 - accuracy: 0.8568
Loss is 0.4904007613658905 and Accuracy is 0.8567500114440918 with learing rate 0.0034019016037257965 and Lambda 1.943018148506439e-05

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 5s 2ms/step - loss: 0.4134 - accuracy: 0.8803
Loss is 0.4134083688259125 and Accuracy is 0.8803333044052124 with learing rate 0.0035232472238131874 and Lambda 2.487051601890942e-05

WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
1875/1875 [==============================] - 4s 2ms/step - loss: 0.6083 - accuracy: 0.8123
Loss is 0.6082624197006226 and Accuracy is 0.812250018119812 with learing rate 0.006436639794216896 and Lambda 5.7743481277859355e-05
```

- Values lr = 0.001227 and Lambda = 1.5660580372555918e-05 seem a good fit. Let's train this model deep for that.

In [284…

```
lr= 0.001227
Lambda= 1.5660580372555918e-05
iterations = 100 #Since we have used early stopping so it will halt
eval_test= True
hidden_nodes=256
output_nodes=10
```

In [285…

```
    #model
    model = Sequential()
    model.add(Dense(500, input_shape=(1024,), activation='relu'))
    model.add(BatchNormalization())
    model.add(Dense(hidden_nodes,activation='relu'))
    model.add(Dense(hidden_nodes,activation='relu'))
    model.add(Dropout(0.5))

    model.add(Dense(hidden_nodes,activation='relu'))
    model.add(Dense(hidden_nodes,activation='relu'))
    model.add(Dropout(0.5))

    model.add(Dense(hidden_nodes,activation='relu'))
    model.add(Dense(hidden_nodes,activation='relu'))
    model.add(Dense(output_nodes, activation='softmax', kernel_regularizer=regularizers.l2(Lambda)))
```

```
    # adam optmizer with custom learning rate
    adam= optimizers.Adam(learning_rate=lr)
    #Compile the model
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

    #Fit the model
```

In [286…

```
history=model.fit(X_train,y_train, validation_data=(X_val,y_val),epochs=100,
              batch_size=500, verbose=1)
```

```
Epoch 1/100
84/84 [==============================] - 4s 29ms/step - loss: 2.1179 - accuracy: 0.1880 - val_loss: 1.9256 - val_accuracy: 0.276
1
Epoch 2/100
84/84 [==============================] - 2s 27ms/step - loss: 1.4891 - accuracy: 0.4626 - val_loss: 1.7075 - val_accuracy: 0.384
9
Epoch 3/100
84/84 [==============================] - 2s 27ms/step - loss: 1.1502 - accuracy: 0.6173 - val_loss: 1.6342 - val_accuracy: 0.423
8
Epoch 4/100
84/84 [==============================] - 2s 26ms/step - loss: 0.9862 - accuracy: 0.6850 - val_loss: 1.4393 - val_accuracy: 0.511
6
Epoch 5/100
84/84 [==============================] - 2s 27ms/step - loss: 0.8880 - accuracy: 0.7230 - val_loss: 1.3245 - val_accuracy: 0.557
7
Epoch 6/100
84/84 [==============================] - 2s 27ms/step - loss: 0.8289 - accuracy: 0.7413 - val_loss: 1.3811 - val_accuracy: 0.552
3
Epoch 7/100
84/84 [==============================] - 2s 26ms/step - loss: 0.7684 - accuracy: 0.7617 - val_loss: 1.3308 - val_accuracy: 0.563
9
Epoch 8/100
84/84 [==============================] - 2s 27ms/step - loss: 0.7267 - accuracy: 0.7762 - val_loss: 1.2170 - val_accuracy: 0.597
5
Epoch 9/100
84/84 [==============================] - 2s 27ms/step - loss: 0.6813 - accuracy: 0.7901 - val_loss: 0.9818 - val_accuracy: 0.682
1
Epoch 10/100
84/84 [==============================] - 2s 27ms/step - loss: 0.6446 - accuracy: 0.8018 - val_loss: 0.9212 - val_accuracy: 0.715
1
Epoch 11/100
84/84 [==============================] - 2s 27ms/step - loss: 0.6218 - accuracy: 0.8085 - val_loss: 0.8425 - val_accuracy: 0.730
1
Epoch 12/100
84/84 [==============================] - 2s 28ms/step - loss: 0.5963 - accuracy: 0.8171 - val_loss: 0.8577 - val_accuracy: 0.727
4
Epoch 13/100
84/84 [==============================] - 2s 27ms/step - loss: 0.5830 - accuracy: 0.8225 - val_loss: 1.0053 - val_accuracy: 0.681
4
Epoch 14/100
84/84 [==============================] - 2s 28ms/step - loss: 0.5591 - accuracy: 0.8299 - val_loss: 0.9592 - val_accuracy: 0.696
6
Epoch 15/100
84/84 [==============================] - 2s 28ms/step - loss: 0.5419 - accuracy: 0.8345 - val_loss: 0.8755 - val_accuracy: 0.718
```

```
2
Epoch 16/100
84/84 [==============================] - 2s 27ms/step - loss: 0.5249 - accuracy: 0.8385 - val_loss: 0.9269 - val_accuracy: 0.719
6
Epoch 17/100
84/84 [==============================] - 2s 27ms/step - loss: 0.4997 - accuracy: 0.8476 - val_loss: 0.8693 - val_accuracy: 0.729
2
Epoch 18/100
84/84 [==============================] - 2s 27ms/step - loss: 0.4921 - accuracy: 0.8491 - val_loss: 0.7428 - val_accuracy: 0.774
3
Epoch 19/100
84/84 [==============================] - 2s 27ms/step - loss: 0.4810 - accuracy: 0.8532 - val_loss: 0.5416 - val_accuracy: 0.833
8
Epoch 20/100
84/84 [==============================] - 2s 27ms/step - loss: 0.4648 - accuracy: 0.8586 - val_loss: 0.7235 - val_accuracy: 0.782
5
Epoch 21/100
84/84 [==============================] - 2s 28ms/step - loss: 0.4601 - accuracy: 0.8588 - val_loss: 0.8475 - val_accuracy: 0.732
2
Epoch 22/100
84/84 [==============================] - 2s 27ms/step - loss: 0.4474 - accuracy: 0.8629 - val_loss: 0.6651 - val_accuracy: 0.789
9
Epoch 23/100
84/84 [==============================] - 2s 27ms/step - loss: 0.4465 - accuracy: 0.8624 - val_loss: 0.6139 - val_accuracy: 0.808
9
Epoch 24/100
84/84 [==============================] - 2s 27ms/step - loss: 0.4296 - accuracy: 0.8698 - val_loss: 0.6756 - val_accuracy: 0.792
2
Epoch 25/100
84/84 [==============================] - 2s 27ms/step - loss: 0.4230 - accuracy: 0.8702 - val_loss: 0.7657 - val_accuracy: 0.760
8
Epoch 26/100
84/84 [==============================] - 2s 27ms/step - loss: 0.4078 - accuracy: 0.8752 - val_loss: 0.6909 - val_accuracy: 0.788
7
Epoch 27/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3943 - accuracy: 0.8796 - val_loss: 0.5741 - val_accuracy: 0.822
3
Epoch 28/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3921 - accuracy: 0.8803 - val_loss: 0.5426 - val_accuracy: 0.837
2
Epoch 29/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3900 - accuracy: 0.8806 - val_loss: 0.6214 - val_accuracy: 0.810
6
Epoch 30/100
```

```
84/84 [==============================] - 2s 27ms/step - loss: 0.3889 - accuracy: 0.8815 - val_loss: 0.5673 - val_accuracy: 0.825
3
Epoch 31/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3821 - accuracy: 0.8832 - val_loss: 0.6275 - val_accuracy: 0.816
5
Epoch 32/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3681 - accuracy: 0.8870 - val_loss: 0.5029 - val_accuracy: 0.849
7
Epoch 33/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3652 - accuracy: 0.8877 - val_loss: 0.5377 - val_accuracy: 0.832
4
Epoch 34/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3604 - accuracy: 0.8873 - val_loss: 0.6839 - val_accuracy: 0.801
0
Epoch 35/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3491 - accuracy: 0.8916 - val_loss: 0.4803 - val_accuracy: 0.857
1
Epoch 36/100
84/84 [==============================] - 2s 28ms/step - loss: 0.3451 - accuracy: 0.8944 - val_loss: 0.5709 - val_accuracy: 0.831
6
Epoch 37/100
84/84 [==============================] - 2s 28ms/step - loss: 0.3375 - accuracy: 0.8978 - val_loss: 0.7530 - val_accuracy: 0.780
7
Epoch 38/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3326 - accuracy: 0.8983 - val_loss: 0.4715 - val_accuracy: 0.855
8
Epoch 39/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3304 - accuracy: 0.8985 - val_loss: 0.5669 - val_accuracy: 0.826
5
Epoch 40/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3269 - accuracy: 0.8991 - val_loss: 0.5809 - val_accuracy: 0.823
0
Epoch 41/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3280 - accuracy: 0.8982 - val_loss: 0.6180 - val_accuracy: 0.816
8
Epoch 42/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3211 - accuracy: 0.9007 - val_loss: 0.5940 - val_accuracy: 0.818
8
Epoch 43/100
84/84 [==============================] - 2s 28ms/step - loss: 0.3138 - accuracy: 0.9023 - val_loss: 0.6198 - val_accuracy: 0.815
5
Epoch 44/100
84/84 [==============================] - 2s 27ms/step - loss: 0.3096 - accuracy: 0.9037 - val_loss: 0.5765 - val_accuracy: 0.836
0
```

```
Epoch 45/100
84/84 [==============================] - 2s 28ms/step - loss: 0.3054 - accuracy: 0.9055 - val_loss: 0.5208 - val_accuracy: 0.842
9
Epoch 46/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2959 - accuracy: 0.9085 - val_loss: 0.6896 - val_accuracy: 0.808
8
Epoch 47/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2995 - accuracy: 0.9074 - val_loss: 0.6088 - val_accuracy: 0.823
3
Epoch 48/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2998 - accuracy: 0.9061 - val_loss: 0.4717 - val_accuracy: 0.863
4
Epoch 49/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2820 - accuracy: 0.9125 - val_loss: 0.4520 - val_accuracy: 0.861
5
Epoch 50/100
84/84 [==============================] - 2s 28ms/step - loss: 0.2856 - accuracy: 0.9113 - val_loss: 0.4614 - val_accuracy: 0.859
5
Epoch 51/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2783 - accuracy: 0.9134 - val_loss: 0.3903 - val_accuracy: 0.883
5
Epoch 52/100
84/84 [==============================] - 2s 28ms/step - loss: 0.2711 - accuracy: 0.9168 - val_loss: 0.5364 - val_accuracy: 0.842
5
Epoch 53/100
84/84 [==============================] - 2s 29ms/step - loss: 0.2744 - accuracy: 0.9148 - val_loss: 0.4545 - val_accuracy: 0.865
1
Epoch 54/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2620 - accuracy: 0.9184 - val_loss: 0.5579 - val_accuracy: 0.845
4
Epoch 55/100
84/84 [==============================] - 2s 28ms/step - loss: 0.2694 - accuracy: 0.9161 - val_loss: 0.6068 - val_accuracy: 0.812
4
Epoch 56/100
84/84 [==============================] - 2s 29ms/step - loss: 0.2653 - accuracy: 0.9193 - val_loss: 0.4151 - val_accuracy: 0.878
3
Epoch 57/100
84/84 [==============================] - 2s 29ms/step - loss: 0.2577 - accuracy: 0.9198 - val_loss: 0.4462 - val_accuracy: 0.869
3
Epoch 58/100
84/84 [==============================] - 2s 28ms/step - loss: 0.2580 - accuracy: 0.9208 - val_loss: 0.6741 - val_accuracy: 0.800
4
Epoch 59/100
84/84 [==============================] - 2s 28ms/step - loss: 0.2611 - accuracy: 0.9184 - val_loss: 0.5804 - val_accuracy: 0.829
```

```
7
Epoch 60/100
84/84 [==============================] - 2s 28ms/step - loss: 0.2525 - accuracy: 0.9211 - val_loss: 0.4146 - val_accuracy: 0.873
9
Epoch 61/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2550 - accuracy: 0.9196 - val_loss: 0.4764 - val_accuracy: 0.860
7
Epoch 62/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2480 - accuracy: 0.9215 - val_loss: 0.8376 - val_accuracy: 0.775
5
Epoch 63/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2473 - accuracy: 0.9230 - val_loss: 0.5271 - val_accuracy: 0.845
7
Epoch 64/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2403 - accuracy: 0.9256 - val_loss: 0.6262 - val_accuracy: 0.822
3
Epoch 65/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2416 - accuracy: 0.9231 - val_loss: 0.4237 - val_accuracy: 0.874
3
Epoch 66/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2440 - accuracy: 0.9241 - val_loss: 0.6056 - val_accuracy: 0.819
4
Epoch 67/100
84/84 [==============================] - 2s 28ms/step - loss: 0.2400 - accuracy: 0.9242 - val_loss: 0.3979 - val_accuracy: 0.883
4
Epoch 68/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2335 - accuracy: 0.9275 - val_loss: 0.4865 - val_accuracy: 0.855
7
Epoch 69/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2316 - accuracy: 0.9276 - val_loss: 0.4783 - val_accuracy: 0.862
3
Epoch 70/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2301 - accuracy: 0.9272 - val_loss: 0.3715 - val_accuracy: 0.894
2
Epoch 71/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2222 - accuracy: 0.9285 - val_loss: 0.5859 - val_accuracy: 0.829
1
Epoch 72/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2224 - accuracy: 0.9291 - val_loss: 0.6397 - val_accuracy: 0.823
1
Epoch 73/100
84/84 [==============================] - 2s 28ms/step - loss: 0.2289 - accuracy: 0.9279 - val_loss: 0.5079 - val_accuracy: 0.856
5
Epoch 74/100
```

```
84/84 [==============================] - 2s 27ms/step - loss: 0.2198 - accuracy: 0.9305 - val_loss: 0.5778 - val_accuracy: 0.832
0
Epoch 75/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2138 - accuracy: 0.9325 - val_loss: 0.4362 - val_accuracy: 0.874
5
Epoch 76/100
84/84 [==============================] - 2s 28ms/step - loss: 0.2192 - accuracy: 0.9310 - val_loss: 0.4564 - val_accuracy: 0.869
2
Epoch 77/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2095 - accuracy: 0.9340 - val_loss: 0.5499 - val_accuracy: 0.846
7
Epoch 78/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2017 - accuracy: 0.9355 - val_loss: 0.5870 - val_accuracy: 0.837
6
Epoch 79/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2014 - accuracy: 0.9371 - val_loss: 0.5266 - val_accuracy: 0.855
2
Epoch 80/100
84/84 [==============================] - 2s 28ms/step - loss: 0.2076 - accuracy: 0.9341 - val_loss: 0.4121 - val_accuracy: 0.883
7
Epoch 81/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2023 - accuracy: 0.9364 - val_loss: 0.3862 - val_accuracy: 0.887
4
Epoch 82/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2060 - accuracy: 0.9351 - val_loss: 0.4020 - val_accuracy: 0.884
3
Epoch 83/100
84/84 [==============================] - 2s 27ms/step - loss: 0.2024 - accuracy: 0.9373 - val_loss: 0.5029 - val_accuracy: 0.859
7
Epoch 84/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1947 - accuracy: 0.9395 - val_loss: 0.7466 - val_accuracy: 0.808
7
Epoch 85/100
84/84 [==============================] - 2s 28ms/step - loss: 0.2034 - accuracy: 0.9361 - val_loss: 0.5221 - val_accuracy: 0.857
3
Epoch 86/100
84/84 [==============================] - 2s 28ms/step - loss: 0.1945 - accuracy: 0.9387 - val_loss: 0.3863 - val_accuracy: 0.888
4
Epoch 87/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1922 - accuracy: 0.9396 - val_loss: 0.4720 - val_accuracy: 0.864
5
Epoch 88/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1941 - accuracy: 0.9387 - val_loss: 0.4632 - val_accuracy: 0.872
8
```

```
Epoch 89/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1955 - accuracy: 0.9387 - val_loss: 0.6187 - val_accuracy: 0.832
3
Epoch 90/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1913 - accuracy: 0.9406 - val_loss: 0.5330 - val_accuracy: 0.850
9
Epoch 91/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1901 - accuracy: 0.9382 - val_loss: 0.4719 - val_accuracy: 0.871
6
Epoch 92/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1872 - accuracy: 0.9393 - val_loss: 0.4804 - val_accuracy: 0.871
1
Epoch 93/100
84/84 [==============================] - 2s 28ms/step - loss: 0.1932 - accuracy: 0.9387 - val_loss: 0.4171 - val_accuracy: 0.885
6
Epoch 94/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1901 - accuracy: 0.9399 - val_loss: 0.4258 - val_accuracy: 0.883
4
Epoch 95/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1774 - accuracy: 0.9435 - val_loss: 0.3618 - val_accuracy: 0.901
3
Epoch 96/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1798 - accuracy: 0.9436 - val_loss: 0.4203 - val_accuracy: 0.890
3
Epoch 97/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1805 - accuracy: 0.9428 - val_loss: 0.4327 - val_accuracy: 0.877
7
Epoch 98/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1789 - accuracy: 0.9431 - val_loss: 0.4316 - val_accuracy: 0.880
2
Epoch 99/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1770 - accuracy: 0.9436 - val_loss: 0.5188 - val_accuracy: 0.857
6
Epoch 100/100
84/84 [==============================] - 2s 27ms/step - loss: 0.1828 - accuracy: 0.9413 - val_loss: 0.4461 - val_accuracy: 0.878
5
```

3C. Evaluate performance of the model with appropriate metrics. [2 Marks]

In [287…

```
y_pred_prob = model.predict(X_test)
y_pred = np.argmax(y_pred_prob, axis=1)
y_test_labels = np.argmax(y_test, axis=1)
```

```
563/563 [==============================] - 1s 2ms/step
```

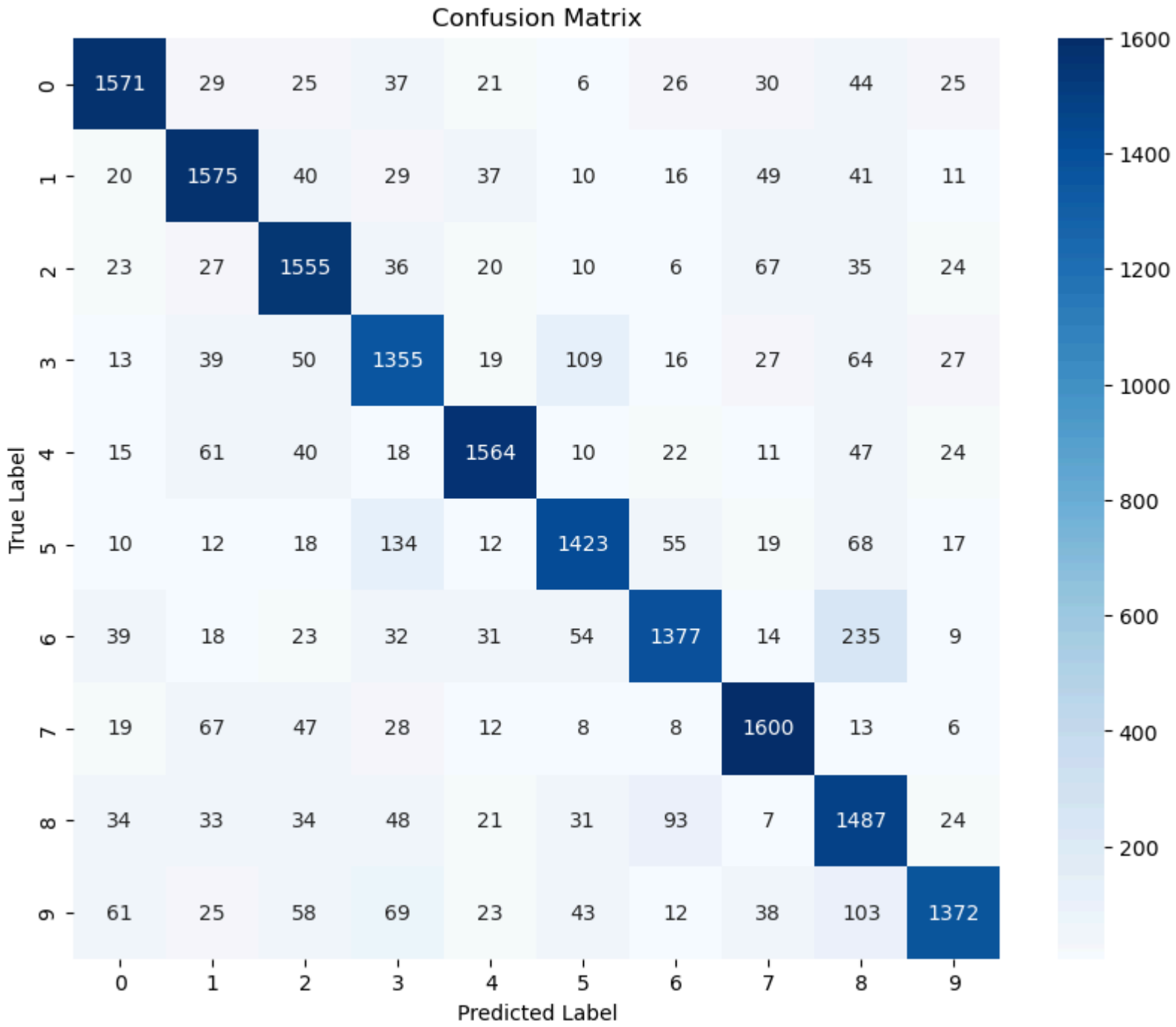
In [288…

```python
print("Accuracy:", accuracy_score(y_test_labels, y_pred))
print("Classification Report:\n", classification_report(y_test_labels, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test_labels, y_pred))
```

```
Accuracy: 0.8266111111111111
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.87      0.87      1814
           1       0.84      0.86      0.85      1828
           2       0.82      0.86      0.84      1803
           3       0.76      0.79      0.77      1719
           4       0.89      0.86      0.88      1812
           5       0.84      0.80      0.82      1768
           6       0.84      0.75      0.80      1832
           7       0.86      0.88      0.87      1808
           8       0.70      0.82      0.75      1812
           9       0.89      0.76      0.82      1804

    accuracy                           0.83     18000
   macro avg       0.83      0.83      0.83     18000
weighted avg       0.83      0.83      0.83     18000

Confusion Matrix:
 [[1571   29   25   37   21    6   26   30   44   25]
 [  20 1575   40   29   37   10   16   49   41   11]
 [  23   27 1555   36   20   10    6   67   35   24]
 [  13   39   50 1355   19  109   16   27   64   27]
 [  15   61   40   18 1564   10   22   11   47   24]
 [  10   12   18  134   12 1423   55   19   68   17]
 [  39   18   23   32   31   54 1377   14  235    9]
 [  19   67   47   28   12    8    8 1600   13    6]
 [  34   33   34   48   21   31   93    7 1487   24]
 [  61   25   58   69   23   43   12   38  103 1372]]
```

In [289…

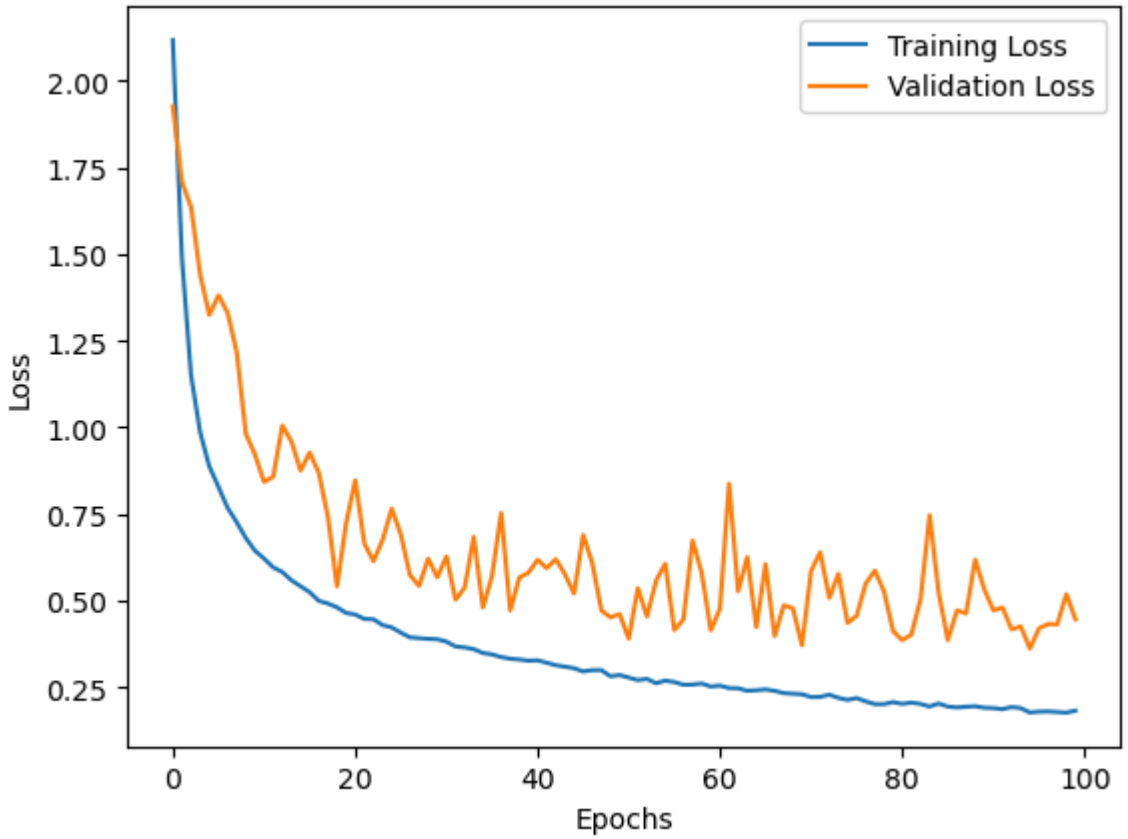
```python
cm=confusion_matrix(y_test_labels, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=range(10), yticklabels=range(10))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```


Confusion Matrix
True Label
Predicted Label
0 1571 29 25 37 21 6 26 30 44 25
1 20 1575 40 29 37 10 16 49 41 11
2 23 27 1555 36 20 10 6 67 35 24
3 13 39 50 1355 19 109 16 27 64 27
4 15 61 40 18 1564 10 22 11 47 24
5 10 12 18 134 12 1423 55 19 68 17
6 39 18 23 32 31 54 1377 14 235 9
7 19 67 47 28 12 8 8 1600 13 6
8 34 33 34 48 21 31 93 7 1487 24
9 61 25 58 69 23 43 12 38 103 1372
0 1 2 3 4 5 6 7 8 9
1600 1400 1200 1000 800 600 400 200

3 D. Plot the training loss, validation loss vs number of epochs and training accuracy, validation accuracy vs number of epochs plot and write your observations on the same. [4 Marks]

In [290…

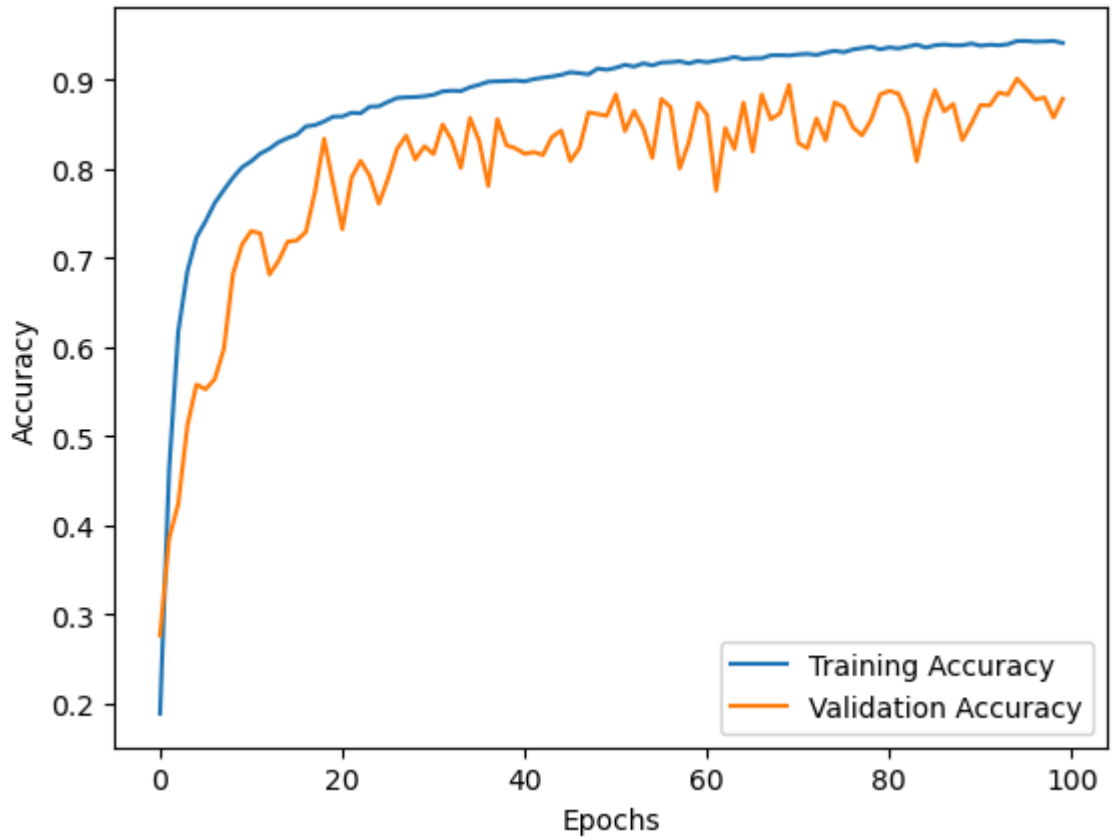
```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



In [291…

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
```

```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



Observation

- Model Evaluation and Insights Let's further evaluate the model by analyzing the provided metrics, classification report, and confusion matrix.
- Accuracy Overall Accuracy: 0.8266 (82.66%) This accuracy is a good indicator that the model is performing reasonably well on the test set. Classification Report The classification report provides precision, recall, and f1-score for each class. Here's a brief explanation of each

- metric: Precision: The ratio of correctly predicted positive observations to the total predicted positives. High precision means low false positive rate.
- Recall: The ratio of correctly predicted positive observations to the all observations in the actual class. High recall means low false negative rate.
- F1-score: The weighted average of Precision and Recall. This score takes both false positives and false negatives into account.

Class Imbalance Handling: The application of SMOTE helped in balancing the classes, as indicated by the balanced precision, recall, and F1-scores across most classes.

Classes like 4, 7, and 0 have higher precision and recall, suggesting the model performs well on these classes. Class 8 has a relatively lower precision (0.70), which indicates that there are more false positives for this class. Confusion Matrix Analysis:

Class 3 shows significant misclassification into class 5 (109 instances), suggesting that these classes might have overlapping features.

Class 6 has notable misclassification into class 8 (235 instances). This indicates potential feature similarity or noise affecting these predictions . High off-diagonal values in the confusion matrix for certain classes indicate areas where the model struggles, highlighting the need for potential feature engineering or additional preprocessing steps.

Recommendations for Improvement

Feature Engineering: Investigate the features contributing to the misclassifications and create new features or transform existing ones to better capture the distinctions between the classes.

Advanced Techniques: Explore ensemble methods or more complex neural network architectures (e.g., deeper networks, convolutional layers) to improve the model's ability to distinguish between challenging classes.

Hyperparameter Tuning: Perform a comprehensive hyperparameter tuning using techniques such as Grid Search or Random Search to find the optimal settings for the neural network.

Regularization: Apply regularization techniques like Dropout or L2 regularization to reduce overfitting, especially for classes where overfitting might be causing poor generalization.

Cross-Validation: Use cross-validation to ensure the model's performance is robust across different subsets of the data and not just the specific train-test split used.

By addressing these areas, the model's performance can be further improved, potentially leading to higher accuracy and better class-wise performance.

========== PART - B END ==========

In [ ]: