BCSE209L - Machine Learning

Digital Assignment

Submitted By

R Tharun - 22BCE0634

Agrim Patil - 21BCE3970

Faculty

Prof. Saravanakumar K

School of Computer Science Engineering

Vellore Institute of Technology, Vellore

TABLE OF CONTENTS

INTRODUCTION:

Foetal health is a critical area in obstetrics, as the ability to monitor and diagnose conditions like abnormal heart rates or uterine activity can save lives. The domain of this project revolves around Cardiotocography (CTG), a diagnostic tool that records foetal heart rate (FHR) and uterine contraction (UC) patterns. CTG data can indicate a foetus's well-being, classified broadly into three states: Normal (N), Suspicious (S), and Pathological (P). Early detection of non-normal states enables timely medical interventions to prevent adverse outcomes.

PROBLEM DOMAIN:

The primary problem in this domain is the complexity of interpreting CTG data. This complexity arises from high interdependence among features, overlapping distributions, and imbalances in class distribution, with normal cases often dominating. Additionally, many features measured from CTG may not contribute significantly to distinguishing the classes, necessitating effective feature selection techniques. The goal is to create a reliable machine learning model to classify fetal states accurately and efficiently, thus reducing the dependency on human interpretation.

ALGORITHM:

This project employs a *Random Forest Classifier*, a widely used ensemble algorithm. Random Forest is robust, capable of handling noisy datasets, and adept at revealing feature importance, making it a natural choice for this problem. The algorithm works by aggregating predictions from multiple decision trees, ensuring high accuracy and reduced overfitting.

PURPOSE:

The purpose of this project is to identify the optimal feature set for classifying fetal health into Normal, Suspicious, and Pathological (NSP) categories, focusing on improving model performance and feature interpretability.

Link for Dataset - Cardiotocography - UCI Machine Learning Repository

DATASET –

Dataset name – Cardiotocography

Details about the data –

The dataset consists of measurements of fetal heart rate (FHR) and uterine contraction (UC) features on cardiotocograms classified by expert obstetricians. 2126 fetal cardiotocograms (CTGs) were automatically processed and the respective diagnostic features measured. The CTGs were also classified by three expert obstetricians and a consensus classification label assigned to each of them. Classification was both with respect to a morphologic pattern (A, B, C. ...) and to a fetal state (N, S, P).

N – Normal (1), S – Suspicious (2), P – Pathological (3)

Number of Instances = 2126, **Number of features** = 21

VARIABLE	DESCRIPTION	ROLE	TYPE
LB	FHR Baseline (bpm)	FEATURE	INTEGER
AC	Acceleration per sec	FEATURE	CONTINUOUS
FM	Fetal Movement per sec	FEATURE	CONTINUOUS
UC	Uterine Contractions per sec	FEATURE	CONTINUOUS
DL	Light Decelerations per sec	FEATURE	CONTINUOUS
DS	Severe Decelerations per sec	FEATURE	CONTINUOUS
DP	Prolonged Decelerations ps	FEATURE	CONTINUOUS
ASTV	Abnormal Short-Term Variability %	FEATURE	INTEGER
MSTV	Mean Short-Term Variability	FEATURE	INTEGER
ALTV	Abnormal Long-Term Variability	FEATURE	CONTINUOUS
MLTV	Mean Long-Term Variability %	FEATURE	CONTINUOUS
Width	Histogram Width	FEATURE	INTEGER
Min	Minimum Histogram Width	FEATURE	INTEGER
Max	Maximum Histogram Width	FEATURE	INTEGER
NMax	Number of Peaks	FEATURE	INTEGER
NZeros	Number of Zeros in Histogram	FEATURE	INTEGER
Mode	Histogram Mode	FEATURE	INTEGER
Mean	Histogram Mean	FEATURE	INTEGER
Median	Histogram Median	FEATURE	INTEGER
Variance	Histogram Variance	FEATURE	INTEGER
Tendency	Histogram Tendency	FEATURE	INTEGER
CLASS	FHR Pattern Code	TARGET	INTEGER
NSP	Normal, Susp., Path.	TARGET	INTEGER

DATA PREPROCESSING –

First, we are loading the raw dataset to see for any missing values, unnecessary columns, discrepancy.

Importing Libraries –

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from scipy.stats import zscore
from collections import Counter
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
```

Loading the dataset –

```
# Load dataset
file_path = 'ML DATASET.csv'
df = pd.read_csv(file_path)
print("Dataset Preview:")
df.head(10)
```

Dataset Preview:

	FileName	Date	SegFile	b	e	LBE	LB	AC	FM	UC	...	C	D	E	AD	DE	LD	FS	SUSP	CLASS	NSP
0	Variab10.txt	01-12-1996	CTG0001.txt	240	357	120	120	0	0	0	...	0	0	0	0	0	0	1	0	9	2
1	Fmcs_1.txt	03-05-1996	CTG0002.txt	5	632	132	132	4	0	4	...	0	0	0	1	0	0	0	0	6	1
2	Fmcs_1.txt	03-05-1996	CTG0003.txt	177	779	133	133	2	0	5	...	0	0	0	1	0	0	0	0	6	1
3	Fmcs_1.txt	03-05-1996	CTG0004.txt	411	1192	134	134	2	0	6	...	0	0	0	1	0	0	0	0	6	1
4	Fmcs_1.txt	03-05-1996	CTG0005.txt	533	1147	132	132	4	0	5	...	0	0	0	0	0	0	0	0	2	1
5	Fmcs_2.txt	03-05-1996	CTG0006.txt	0	953	134	134	1	0	10	...	0	0	0	0	0	1	0	0	8	3
6	Fmcs_2.txt	03-05-1996	CTG0007.txt	240	953	134	134	1	0	9	...	0	0	0	0	0	1	0	0	8	3
7	Hasc_1.txt	22-02-1995	CTG0008.txt	62	679	122	122	0	0	0	...	0	0	0	0	0	0	1	0	9	3
8	Hasc_1.txt	22-02-1995	CTG0009.txt	120	779	122	122	0	0	1	...	0	0	0	0	0	0	1	0	9	3
9	Hasc_1.txt	22-02-1995	CTG0010.txt	181	1192	122	122	0	0	3	...	0	0	0	0	0	0	1	0	9	3

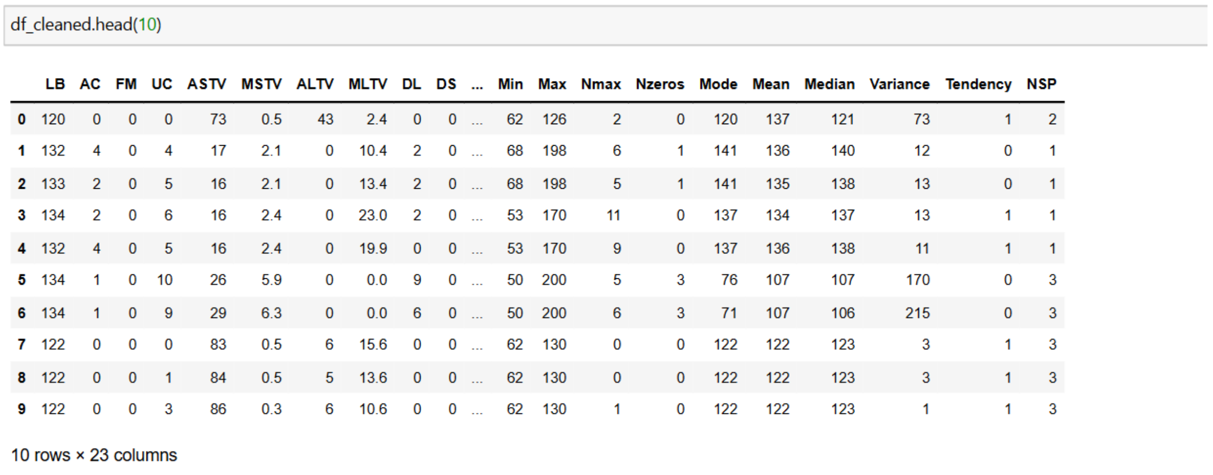
10 rows × 40 columns

There are unnecessary columns like FileName, Date, SegFile. Also, our task is to classify foetus as Normal, Suspicious, Pathological. So we are not including A,B,C,D,E,AD,DE,LD,FS,SUSP,CLASS.Also the variable b is start instant, e is end instant, lbe and lb is same taken in two different methods which has not much difference in values, so we are excluding b,e,lbe.

Dropping unnecessary Columns –

```
# Drop unnecessary columns
columns_to_drop = ['FileName', 'Date', 'SegFile', 'CLASS','A','B','C','D','E','AD','DE','LD','FS','SUSP','CLASS','b','e','LBE']
df_cleaned = df.drop(columns=columns_to_drop, axis=1)
print("Remaining columns after dropping unnecessary ones:")
print(df_cleaned.columns)
```

```
Remaining columns after dropping unnecessary ones:
Index(['LB', 'AC', 'FM', 'UC', 'ASTV', 'MSTV', 'ALTV', 'MLTV', 'DL', 'DS',
       'DP', 'DR', 'Width', 'Min', 'Max', 'Nmax', 'Nzeros', 'Mode', 'Mean',
       'Median', 'Variance', 'Tendency', 'NSP'],
      dtype='object')
```



```
df_cleaned.head(10)
```

	LB	AC	FM	UC	ASTV	MSTV	ALTV	MLTV	DL	DS	...	Min	Max	Nmax	Nzeros	Mode	Mean	Median	Variance	Tendency	NSP
0	120	0	0	0	73	0.5	43	2.4	0	0	...	62	126	2	0	120	137	121	73	1	2
1	132	4	0	4	17	2.1	0	10.4	2	0	...	68	198	6	1	141	136	140	12	0	1
2	133	2	0	5	16	2.1	0	13.4	2	0	...	68	198	5	1	141	135	138	13	0	1
3	134	2	0	6	16	2.4	0	23.0	2	0	...	53	170	11	0	137	134	137	13	1	1
4	132	4	0	5	16	2.4	0	19.9	0	0	...	53	170	9	0	137	136	138	11	1	1
5	134	1	0	10	26	5.9	0	0.0	9	0	...	50	200	5	3	76	107	107	170	0	3
6	134	1	0	9	29	6.3	0	0.0	6	0	...	50	200	6	3	71	107	106	215	0	3
7	122	0	0	0	83	0.5	6	15.6	0	0	...	62	130	0	0	122	122	123	3	1	3
8	122	0	0	1	84	0.5	5	13.6	0	0	...	62	130	0	0	122	122	123	3	1	3
9	122	0	0	3	86	0.3	6	10.6	0	0	...	62	130	1	0	122	122	123	1	1	3

10 rows × 23 columns

Finding missing values if any –

```
# Handle missing values
missing_values = df_cleaned.isnull().sum()
print(missing_values)
```

```
LB          0
AC          0
FM          0
UC          0
ASTV        0
MSTV        0
ALTV        0
MLTV        0
DL          0
DS          0
DP          0
DR          0
Width       0
Min         0
Max         0
Nmax        0
Nzeros      0
Mode        0
Mean        0
Median      0
Variance    0
Tendency    0
NSP         0
dtype: int64
```

No missing values found in the dataset

Removing Outliers –

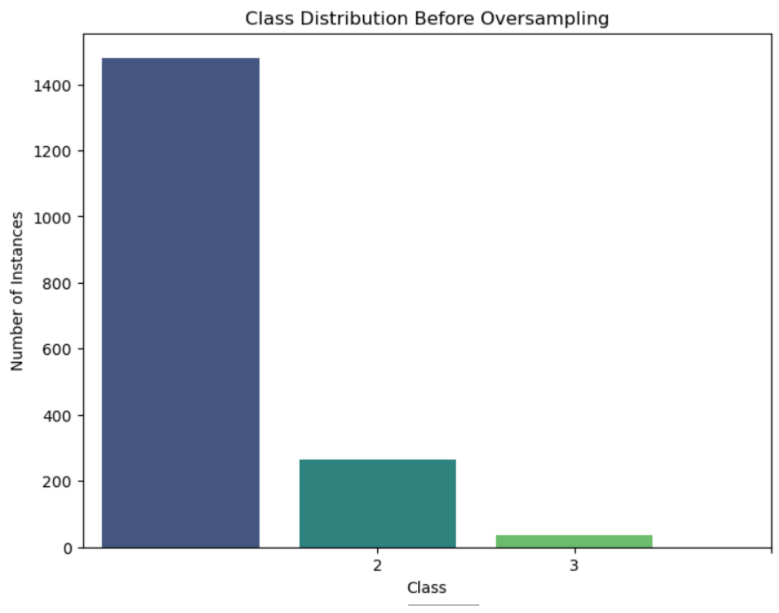
```
# Apply z-score to remove outliers
z_scores = np.abs(zscore(df_cleaned))
threshold = 3
outliers = (z_scores > threshold).any(axis=1)
df_no_outliers = df_cleaned[~outliers]
print(f"Number of outliers: {outliers.sum()}")
print(f"Dataset shape after removing outliers: {df_no_outliers.shape}")
```

```
Number of outliers: 343
Dataset shape after removing outliers: (1783, 26)
```

Class distribution to check for class imbalance –

```
class_counts_before = Counter(y)
print("Class distribution before oversampling:")
for label, count in class_counts_before.items():
    print(f"Class {label}: {count} instances")

# Plot the class distribution
def plot_class_distribution(class_counts, title):
    labels = list(class_counts.keys())
    counts = list(class_counts.values())

    plt.figure(figsize=(8, 6))
    sns.barplot(x=labels, y=counts, palette='viridis')
    plt.title(title)
    plt.xlabel("Class")
    plt.ylabel("Number of Instances")
    plt.xticks(labels)
    plt.show()

plot_class_distribution(class_counts_before, "Class Distribution Before Oversampling")
```

```
Class distribution before oversampling:
Class 2: 266 instances
Class 1: 1480 instances
Class 3: 37 instances
```

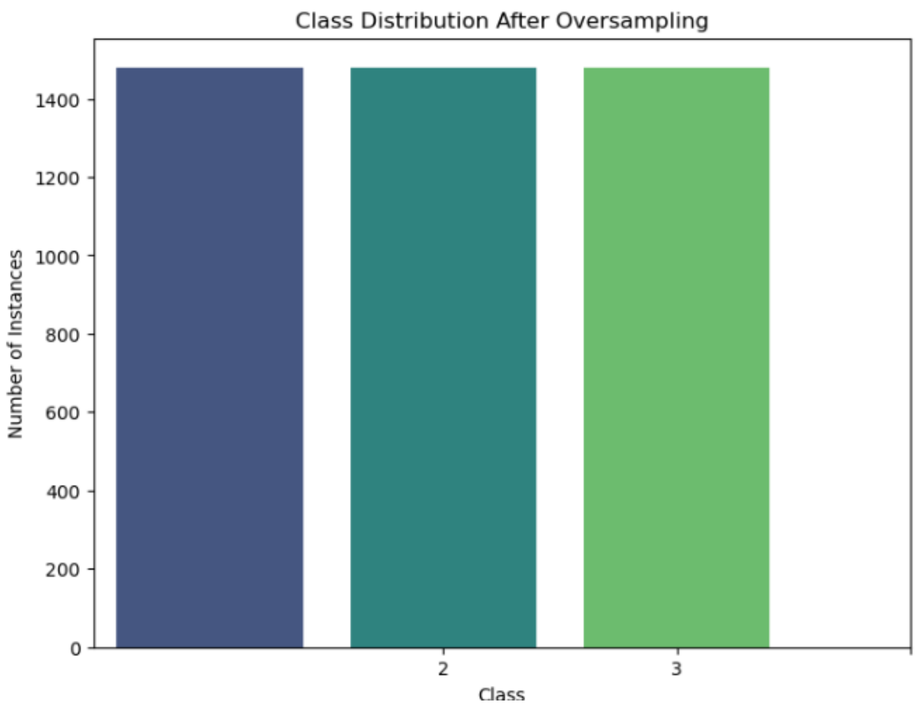


There is significant imbalance in class distribution in the dataset. Therefore, we perform SMOTE to balance the class distribution.

```
# Apply SMOTE to handle class imbalance
X = df_no_outliers.drop(columns=['NSP'])
y = df_no_outliers['NSP']
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

```
# Analyze class distribution after oversampling
class_counts_after = Counter(y_resampled)
print("Class distribution after oversampling:")
for label, count in class_counts_after.items():
    print(f"Class {label}: {count} instances")

# Plot the class distribution after oversampling
plot_class_distribution(class_counts_after, "Class Distribution After Oversampling")
```

```
Class distribution after oversampling:
Class 2: 1480 instances
Class 1: 1480 instances
Class 3: 1480 instances
```



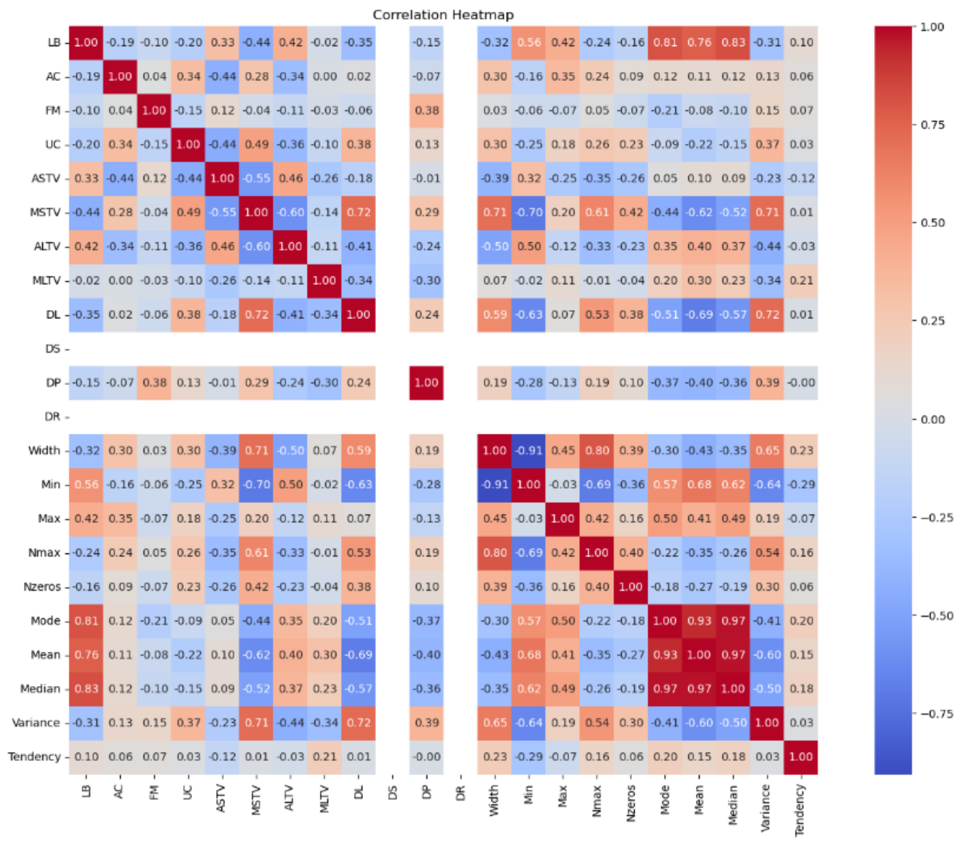
The values aren't normalized, therefore, we are performing standardization of values in the dataset

```
# Standardize the features
scaler = StandardScaler()
X_standardized = pd.DataFrame(scaler.fit_transform(X_resampled), columns=X.columns)
print("Preview of standardized features:")
print(X_standardized.head(10))
```

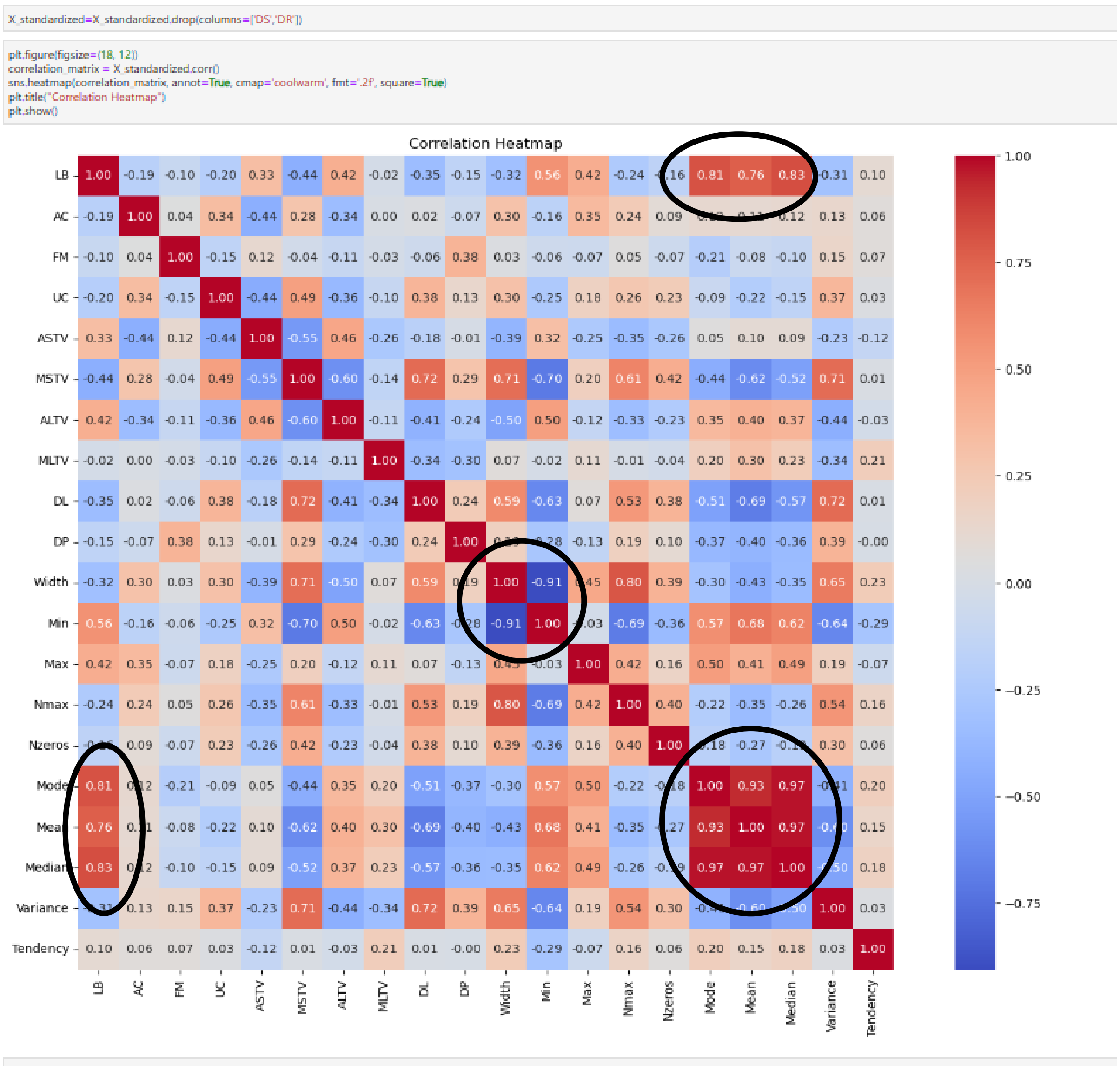
```
Preview of standardized features:
         LB        AC        FM        UC      ASTV      MSTV      ALTV  \
0 -1.668897 -0.452417 -0.285683 -0.958014  0.782890 -0.616637  1.469036
1 -0.458376  1.224321 -0.285683  0.629884 -2.384387  1.553911 -0.866721
2 -0.357499  0.385952 -0.285683  1.026858 -2.440946  1.553911 -0.866721
3 -0.256622  0.385952 -0.285683  1.423833 -2.440946  1.960889 -0.866721
4 -0.458376  1.224321 -0.285683  1.026858 -2.440946  1.960889 -0.866721
5 -1.467144 -0.452417 -0.285683 -0.958014  1.348475 -0.616637 -0.540801
6 -1.467144 -0.452417 -0.285683 -0.561039  1.405033 -0.616637 -0.595121
7 -1.467144 -0.452417 -0.285683  0.232910  1.518150 -0.887955 -0.540801
8 -0.559253  1.224321  4.054618  1.423833 -1.762243  0.604296 -0.866721
9 -0.861883 -0.452417 -0.285683 -0.958014  1.178799 -0.616637 -0.866721

       MLTV        DL   DS  ...     Width       Min       Max      Nmax  \
0 -1.196401 -0.527879  0.0  ...  0.217517 -1.287798 -2.214273 -0.418087
1  0.757606  0.354071  0.0  ...  2.069865 -1.098644  2.563972  1.078332
2  1.490359  0.354071  0.0  ...  2.069865 -1.098644  2.563972  0.704227
3  3.835168  0.354071  0.0  ...  1.705009 -1.571528  0.705766  2.948855
4  3.077990 -0.527879  0.0  ...  1.705009 -1.571528  0.705766  2.200646
5  2.027711 -0.527879  0.0  ...  0.329780 -1.287798 -1.948815 -1.166297
6  1.539209 -0.527879  0.0  ...  0.329780 -1.287798 -1.948815 -1.166297
7  0.806457 -0.527879  0.0  ...  0.329780 -1.287798 -1.948815 -0.792192
8  1.368234  0.354071  0.0  ...  0.273649 -0.468133 -0.356067  0.704227
9 -0.121697 -0.527879  0.0  ... -1.129645  0.351531 -1.948815 -1.166297

     Nzeros      Mode      Mean    Median  Variance  Tendency
0 -0.389138 -1.221067  0.081970 -1.195198  2.814090  1.448171
1  1.875310  0.171026  0.022993  0.079994 -0.043683 -0.458674
2  1.875310  0.171026 -0.035984 -0.054236  0.003165 -0.458674
3 -0.389138 -0.094135 -0.094961 -0.121352  0.003165  1.448171
4 -0.389138 -0.094135  0.022993 -0.054236 -0.090532  1.448171
5 -0.389138 -1.088487 -0.802685 -1.060967 -0.465322  1.448171
6 -0.389138 -1.088487 -0.802685 -1.060967 -0.465322  1.448171
7 -0.389138 -1.088487 -0.802685 -1.060967 -0.559019  1.448171
8 -0.389138 -0.226715 -0.094961 -0.121352 -0.277927  1.448171
9 -0.389138 -0.823326 -0.684731 -0.926736 -0.559019  1.448171

[10 rows x 22 columns]
```

CORRELATION HEATMAP –

```
plt.figure(figsize=(18, 12))
correlation_matrix = X_standardized.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', square=True)
plt.title("Correlation Heatmap")
plt.show()
```



We found out that, DR and DS, all values are same, so it is ineffective in prediction. We are dropping them.



```
X_standardized=X_standardized.drop(columns=['DS','DR'])
```

```
plt.figure(figsize=(18, 12))
correlation_matrix = X_standardized.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', square=True)
plt.title("Correlation Heatmap")
plt.show()
```

Correlation Heatmap

	LB	AC	FM	UC	ASTV	MSTV	ALTV	MLTV	DL	DP	Width	Min	Max	Nmax	Nzeros	Mode	Mean	Median	Variance	Tendency
LB	1.00	-0.19	-0.10	-0.20	0.33	-0.44	0.42	-0.02	-0.35	-0.15	-0.32	0.56	0.42	-0.24	-0.16	0.81	0.76	0.83	-0.31	0.10
AC	-0.19	1.00	0.04	0.34	-0.44	0.28	-0.34	0.00	0.02	-0.07	0.30	-0.16	0.35	0.24	0.09	[illegible]	[illegible]	0.12	0.13	0.06
FM	-0.10	0.04	1.00	-0.15	0.12	-0.04	-0.11	-0.03	-0.06	0.38	0.03	-0.06	-0.07	0.05	-0.07	-0.21	-0.08	-0.10	0.15	0.07
UC	-0.20	0.34	-0.15	1.00	-0.44	0.49	-0.36	-0.10	0.38	0.13	0.30	-0.25	0.18	0.26	0.23	-0.09	-0.22	-0.15	0.37	0.03
ASTV	0.33	-0.44	0.12	-0.44	1.00	-0.55	0.46	-0.26	-0.18	-0.01	-0.39	0.32	-0.25	-0.35	-0.26	0.05	0.10	0.09	-0.23	-0.12
MSTV	-0.44	0.28	-0.04	0.49	-0.55	1.00	-0.60	-0.14	0.72	0.29	0.71	-0.70	0.20	0.61	0.42	-0.44	-0.62	-0.52	0.71	0.01
ALTV	0.42	-0.34	-0.11	-0.36	0.46	-0.60	1.00	-0.11	-0.41	-0.24	-0.50	0.50	-0.12	-0.33	-0.23	0.35	0.40	0.37	-0.44	-0.03
MLTV	-0.02	0.00	-0.03	-0.10	-0.26	-0.14	-0.11	1.00	-0.34	-0.30	0.07	-0.02	0.11	-0.01	-0.04	0.20	0.30	0.23	-0.34	0.21
DL	-0.35	0.02	-0.06	0.38	-0.18	0.72	-0.41	-0.34	1.00	0.24	0.59	-0.63	0.07	0.53	0.38	-0.51	-0.69	-0.57	0.72	0.01
DP	-0.15	-0.07	0.38	0.13	-0.01	0.29	-0.24	-0.30	0.24	1.00	[illegible]	-0.28	-0.13	0.19	0.10	-0.37	-0.40	-0.36	0.39	-0.00
Width	-0.32	0.30	0.03	0.30	-0.39	0.71	-0.50	0.07	0.59	0.19	1.00	-0.91	0.45	0.80	0.39	-0.30	-0.43	-0.35	0.65	0.23
Min	0.56	-0.16	-0.06	-0.25	0.32	-0.70	0.50	-0.02	-0.63	-0.28	-0.91	1.00	-0.03	-0.69	-0.36	0.57	0.68	0.62	-0.64	-0.29
Max	0.42	0.35	-0.07	0.18	-0.25	0.20	-0.12	0.11	0.07	-0.13	[illegible]	-0.03	1.00	0.42	0.16	0.50	0.41	0.49	0.19	-0.07
Nmax	-0.24	0.24	0.05	0.26	-0.35	0.61	-0.33	-0.01	0.53	0.19	0.80	-0.69	0.42	1.00	0.40	-0.22	-0.35	-0.26	0.54	0.16
Nzeros	[illegible]	0.09	-0.07	0.23	-0.26	0.42	-0.23	-0.04	0.38	0.10	0.39	-0.36	0.16	0.40	1.00	-0.18	-0.27	[illegible]	0.30	0.06
Mode	0.81	0.12	-0.21	-0.09	0.05	-0.44	0.35	0.20	-0.51	-0.37	-0.30	0.57	0.50	-0.22	-0.18	1.00	0.93	0.97	-0.41	0.20
Mean	0.76	0.11	-0.08	-0.22	0.10	-0.62	0.40	0.30	-0.69	-0.40	-0.43	0.68	0.41	-0.35	-0.27	0.93	1.00	0.97	-0.60	0.15
Median	0.83	0.12	-0.10	-0.15	0.09	-0.52	0.37	0.23	-0.57	-0.36	-0.35	0.62	0.49	-0.26	-0.19	0.97	0.97	1.00	-0.50	0.18
Variance	-0.31	0.13	0.15	0.37	-0.23	0.71	-0.44	-0.34	0.72	0.39	0.65	-0.64	0.19	0.54	0.30	-0.41	-0.60	-0.50	1.00	0.03
Tendency	0.10	0.06	0.07	0.03	-0.12	0.01	-0.03	0.21	0.01	-0.00	0.23	-0.29	-0.07	0.16	0.06	0.20	0.15	0.18	0.03	1.00

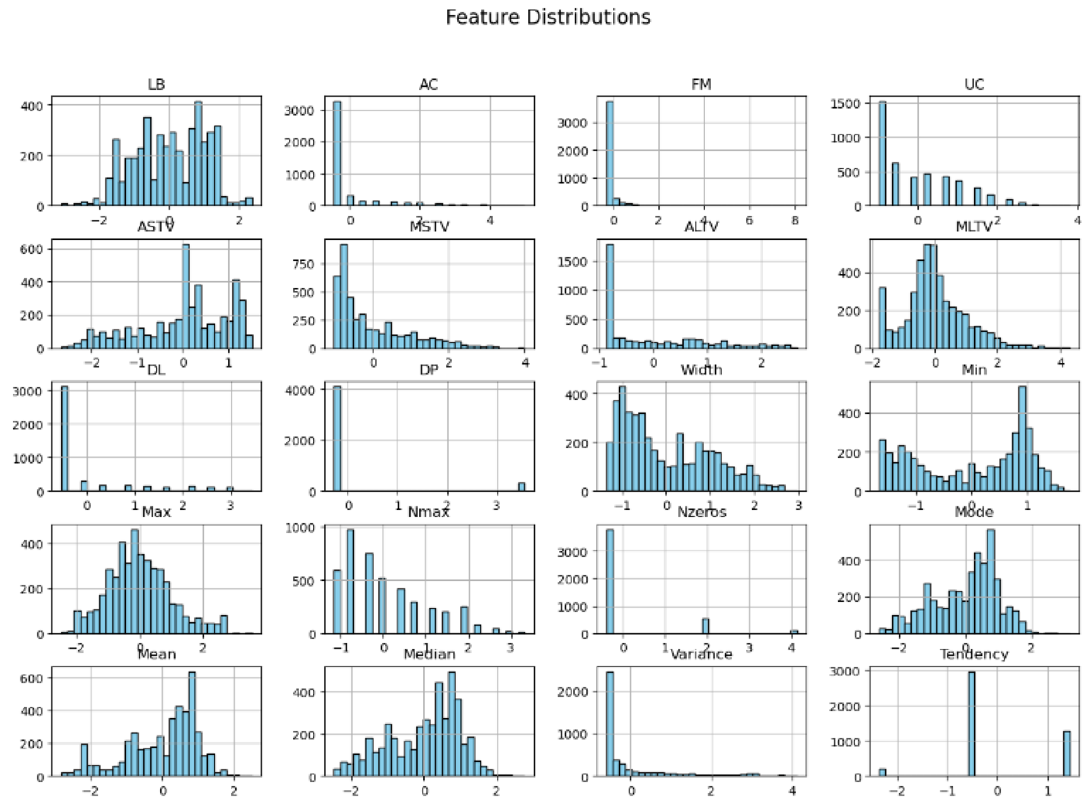
From correlation heatmap, we can observe (encircled) that there is huge positive/negative correlations between –

1. Lb,Mean; Lb,Median; Lb,Mode
2. Mean,Median; Mean,Mode; Median,Mode
3. Width, Min

Therefore, we conclude that we should avoid including both these pairs in feature set selection as one feature is sufficient among them if they have high value of correlation.

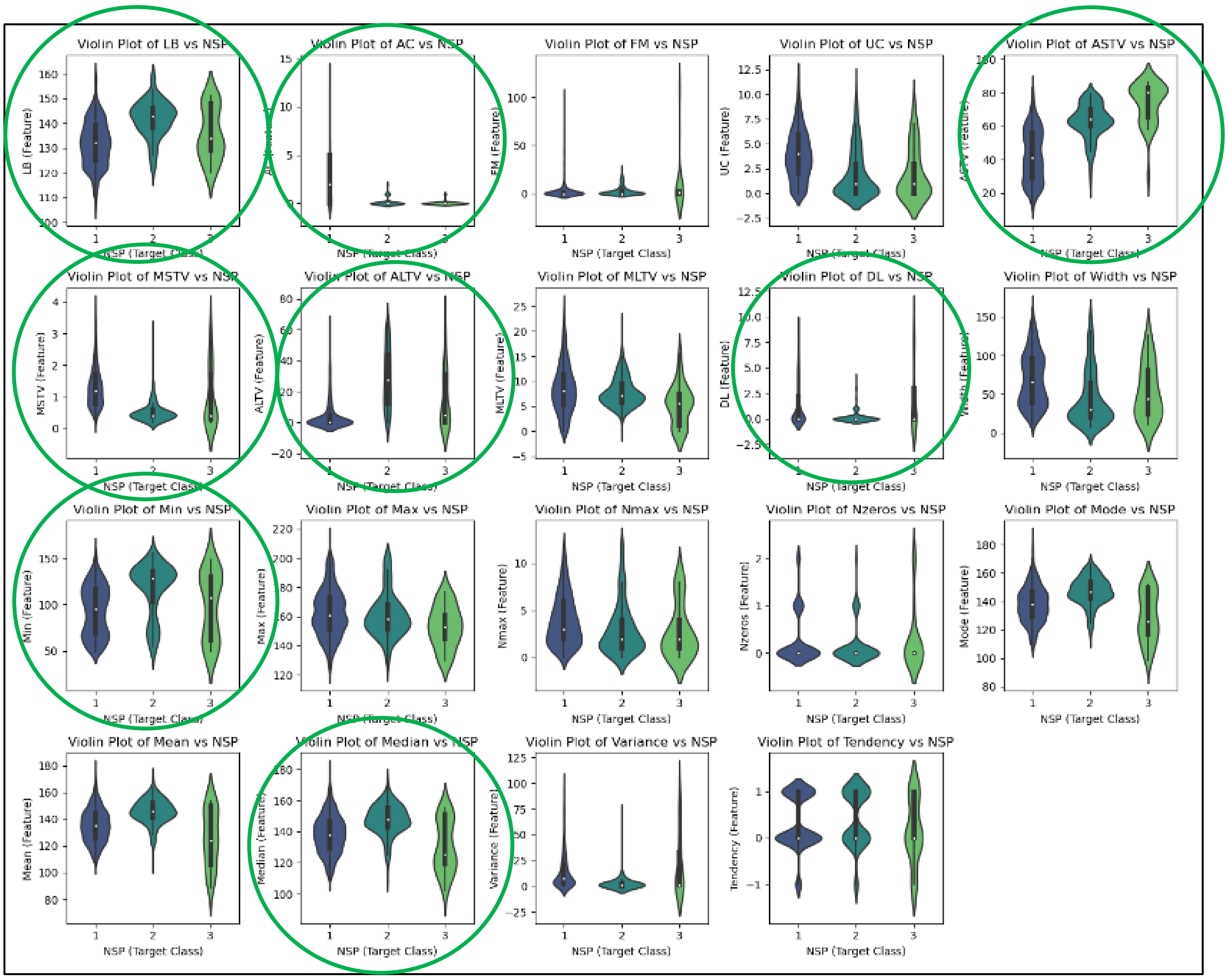
Feature distribution –

```
X_standardized.hist(bins=30, figsize=(15, 10), color='skyblue', edgecolor='black')
plt.suptitle("Feature Distributions", fontsize=16)
plt.show()
```



Violin Plot -

```
import seaborn as sns
import matplotlib.pyplot as plt

# List of features (excluding the dropped ones like 'DS', 'DR', etc.)
features = [ 'LB', 'AC', 'FM', 'UC', 'ASTV', 'MSTV', 'ALTV', 'MLTV',
            'DL', 'Width', 'Min', 'Max', 'Nmax', 'Nzeros', 'Mode', 'Mean', 'Median', 'Variance', 'Tendency']

# Set up the number of columns in the plot grid (3 plots per row)
n_cols = 5
n_rows = (len(features) + n_cols - 1) // n_cols  # Calculate number of rows needed

# Create a figure and axes
fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, 3 * n_rows))

# Flatten the axes array if it's multidimensional
axes = axes.flatten()

# Loop through the features and create a violin plot for each
for i, feature in enumerate(features):
    sns.violinplot(x='NSP', y=feature, data=df_no_outliers, palette='viridis', ax=axes[i])
    axes[i].set_title(f'Violin Plot of {feature} vs NSP')
    axes[i].set_xlabel('NSP (Target Class)')
    axes[i].set_ylabel(f'{feature} (Feature)')

# Remove any unused axes (in case there are fewer features than subplots)
for i in range(len(features), len(axes)):
    fig.delaxes(axes[i])

# Adjust layout for better visualization
plt.tight_layout()
plt.show()
```



Features with heavily overlapping distributions contribute little to distinguishing between classes and can often be discarded.

- Features with non-overlapping or minimally overlapping distributions across target classes are likely to be more predictive.
- Features with distinct widths or shapes across categories indicate that the feature has different distributions depending on the class.
- If a feature has similar widths and shapes across all target categories, it may lack discriminative power.
- Features with heavily overlapping distributions contribute little to distinguishing between classes and can often be discarded.

The green encircled Violin plots are best matching the above conditions. So we can prefer choosing feature set combinations from them

1. LB
2. ASTV
3. MSTV
4. ALTV
5. Min
6. Median
7. AC

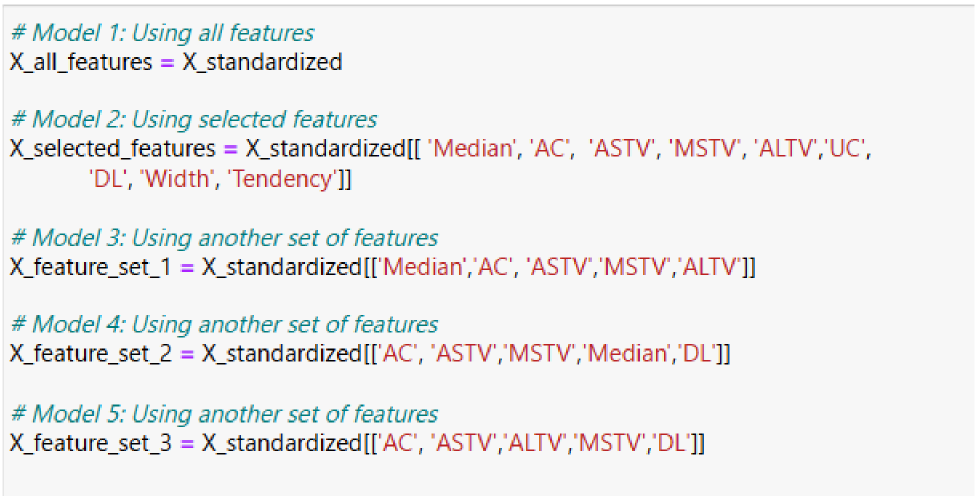
Therefore, the feature set we are going to model is –

1. All features after dropping unnecessary columns
2. Set 0 - ['Median', 'AC', 'ASTV', 'MSTV', 'ALTV','UC','DL', 'Width', 'Tendency']
3. Set 1 (fixed 5 features) - ['Median','AC', 'ASTV','MSTV','ALTV']
4. Set 2 (fixed 5 features) - ['AC', 'ASTV','MSTV','Median','DL']
5. Set 3 (fixed 5 features) - ['AC', 'ASTV','ALTV','MSTV','DL']

Training and Testing data splitting – We are considering 70% of dataset for training and 30% of dataset for splitting.

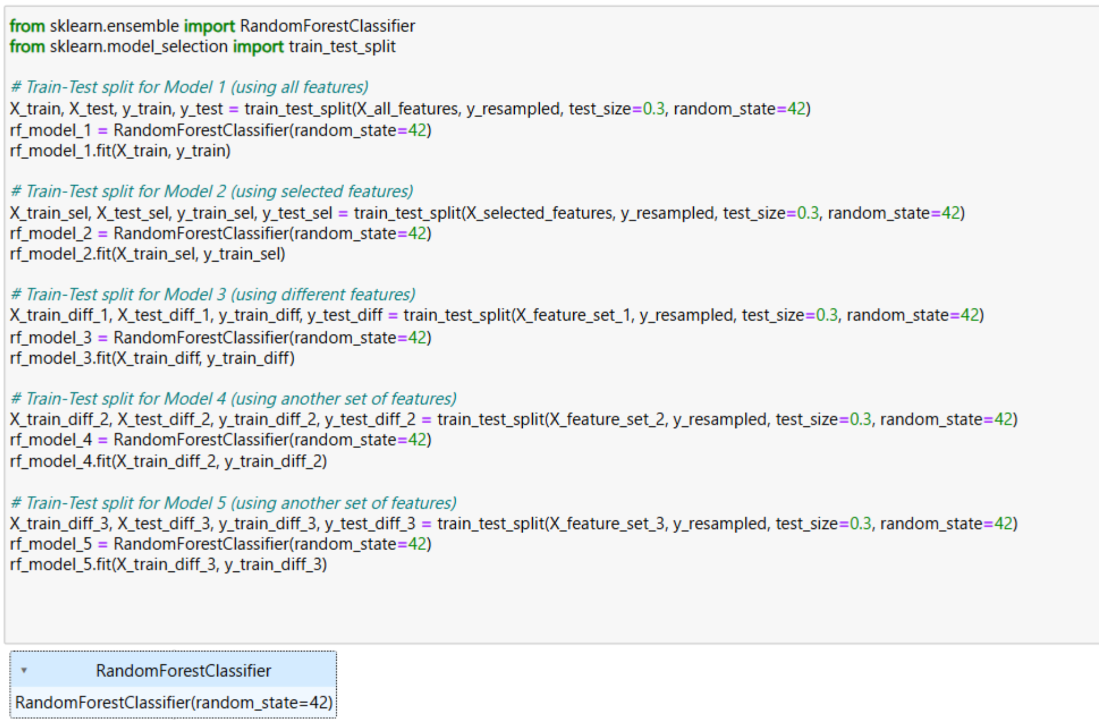
MODEL DEVELOPMENT –

1. FEATURE SELECTION



```
# Model 1: Using all features
X_all_features = X_standardized

# Model 2: Using selected features
X_selected_features = X_standardized[[ 'Median', 'AC',  'ASTV', 'MSTV', 'ALTV','UC',
          'DL', 'Width', 'Tendency']]

# Model 3: Using another set of features
X_feature_set_1 = X_standardized[['Median','AC', 'ASTV','MSTV','ALTV']]

# Model 4: Using another set of features
X_feature_set_2 = X_standardized[['AC', 'ASTV','MSTV','Median','DL']]

# Model 5: Using another set of features
X_feature_set_3 = X_standardized[['AC', 'ASTV','ALTV','MSTV','DL']]
```

2. MODELLING – RANDOM FOREST CLASSIFIER



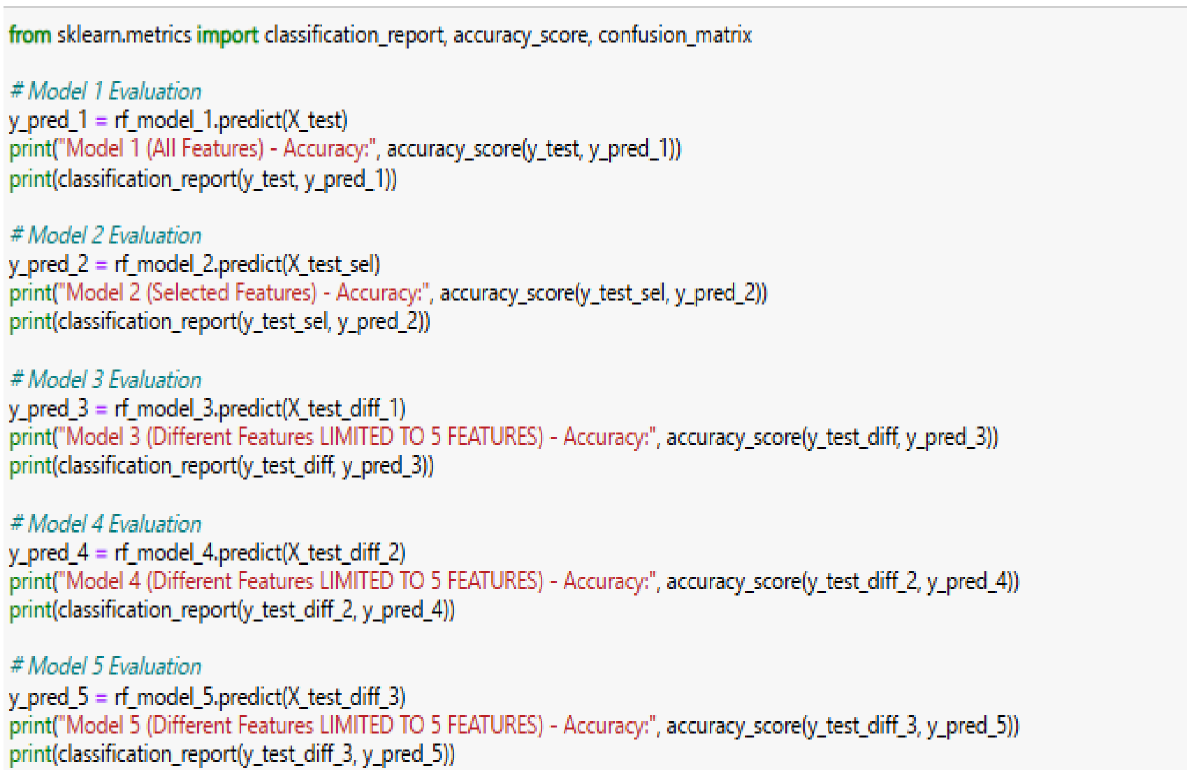
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

# Train-Test split for Model 1 (using all features)
X_train, X_test, y_train, y_test = train_test_split(X_all_features, y_resampled, test_size=0.3, random_state=42)
rf_model_1 = RandomForestClassifier(random_state=42)
rf_model_1.fit(X_train, y_train)

# Train-Test split for Model 2 (using selected features)
X_train_sel, X_test_sel, y_train_sel, y_test_sel = train_test_split(X_selected_features, y_resampled, test_size=0.3, random_state=42)
rf_model_2 = RandomForestClassifier(random_state=42)
rf_model_2.fit(X_train_sel, y_train_sel)

# Train-Test split for Model 3 (using different features)
X_train_diff_1, X_test_diff_1, y_train_diff, y_test_diff = train_test_split(X_feature_set_1, y_resampled, test_size=0.3, random_state=42)
rf_model_3 = RandomForestClassifier(random_state=42)
rf_model_3.fit(X_train_diff, y_train_diff)

# Train-Test split for Model 4 (using another set of features)
X_train_diff_2, X_test_diff_2, y_train_diff_2, y_test_diff_2 = train_test_split(X_feature_set_2, y_resampled, test_size=0.3, random_state=42)
rf_model_4 = RandomForestClassifier(random_state=42)
rf_model_4.fit(X_train_diff_2, y_train_diff_2)

# Train-Test split for Model 5 (using another set of features)
X_train_diff_3, X_test_diff_3, y_train_diff_3, y_test_diff_3 = train_test_split(X_feature_set_3, y_resampled, test_size=0.3, random_state=42)
rf_model_5 = RandomForestClassifier(random_state=42)
rf_model_5.fit(X_train_diff_3, y_train_diff_3)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

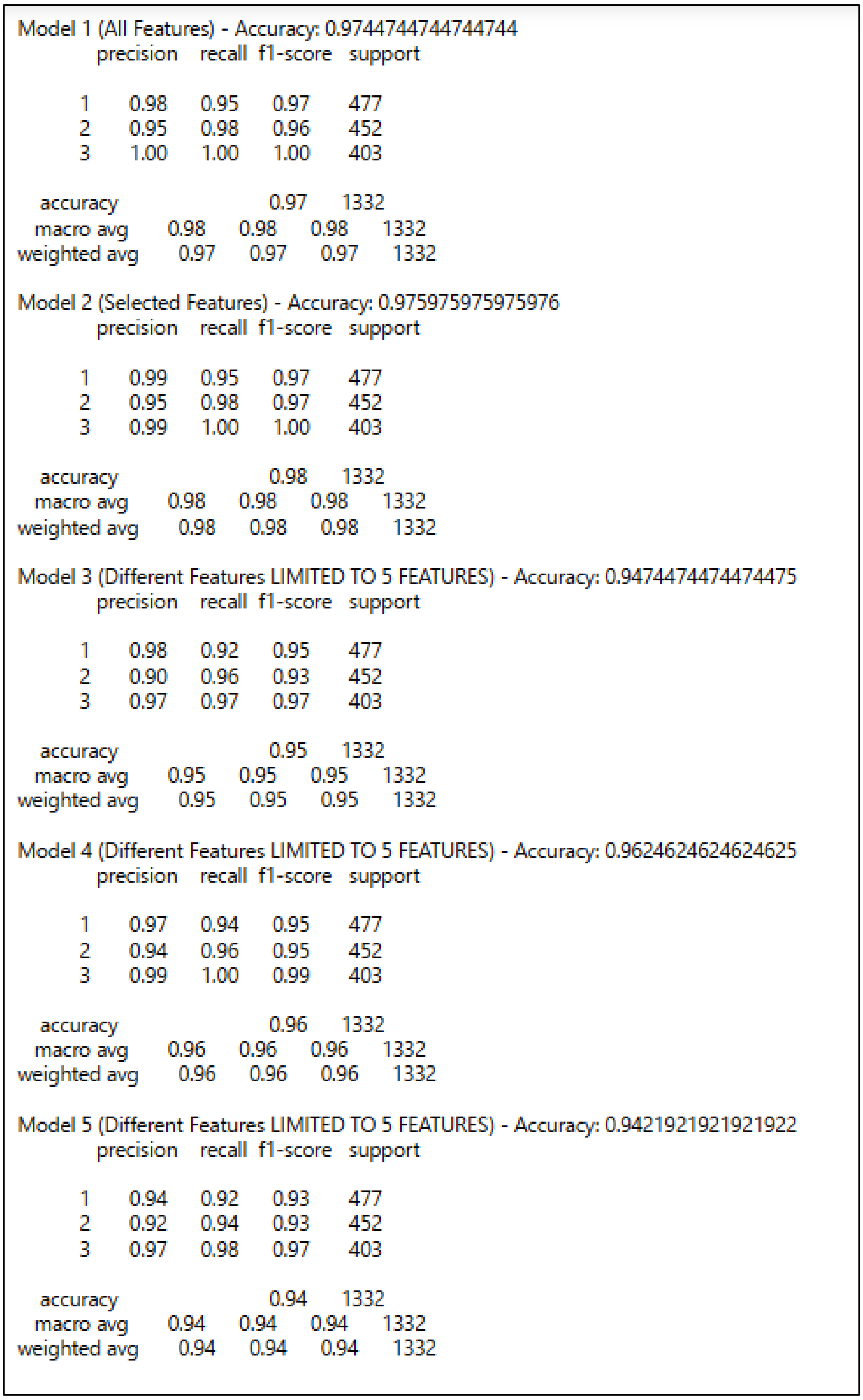
EVALUATION –

The following evaluation metrics were used –

1. **Accuracy**
   a. Proportion of correctly classified samples.
   b. Measures overall performance but can be misleading for imbalanced datasets where one class dominates. RF often achieves high accuracy due to its ensemble approach.
2. **Precision**
   a. Precision= True Positives (TP)/ TP + False Positives (FP)
   b. Evaluates the correctness of positive predictions. Important in scenarios where false positives are costly. RF's probabilistic outputs can help optimize for higher precision.
3. **Recall**
   a. Recall= TP/ TP + False Negatives (FN)
   b. Evaluates how well the model identifies positive instances. Key for imbalanced datasets where missing positive cases is critical (e.g., medical diagnosis).
4. **F1-Score**
   a. Harmonic mean of Precision and Recall
   b. Balances precision and recall, useful when there's a trade-off. Random Forest can achieve balanced F1-scores due to its ability to handle complex feature interactions.



```
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix

# Model 1 Evaluation
y_pred_1 = rf_model_1.predict(X_test)
print("Model 1 (All Features) - Accuracy:", accuracy_score(y_test, y_pred_1))
print(classification_report(y_test, y_pred_1))

# Model 2 Evaluation
y_pred_2 = rf_model_2.predict(X_test_sel)
print("Model 2 (Selected Features) - Accuracy:", accuracy_score(y_test_sel, y_pred_2))
print(classification_report(y_test_sel, y_pred_2))

# Model 3 Evaluation
y_pred_3 = rf_model_3.predict(X_test_diff_1)
print("Model 3 (Different Features LIMITED TO 5 FEATURES) - Accuracy:", accuracy_score(y_test_diff, y_pred_3))
print(classification_report(y_test_diff, y_pred_3))

# Model 4 Evaluation
y_pred_4 = rf_model_4.predict(X_test_diff_2)
print("Model 4 (Different Features LIMITED TO 5 FEATURES) - Accuracy:", accuracy_score(y_test_diff_2, y_pred_4))
print(classification_report(y_test_diff_2, y_pred_4))

# Model 5 Evaluation
y_pred_5 = rf_model_5.predict(X_test_diff_3)
print("Model 5 (Different Features LIMITED TO 5 FEATURES) - Accuracy:", accuracy_score(y_test_diff_3, y_pred_5))
print(classification_report(y_test_diff_3, y_pred_5))
```



```
Model 1 (All Features) - Accuracy: 0.9744744744744744
              precision    recall  f1-score   support

           1       0.98      0.95      0.97       477
           2       0.95      0.98      0.96       452
           3       1.00      1.00      1.00       403

    accuracy                           0.97      1332
   macro avg       0.98      0.98      0.98      1332
weighted avg       0.97      0.97      0.97      1332

Model 2 (Selected Features) - Accuracy: 0.975975975975976
              precision    recall  f1-score   support

           1       0.99      0.95      0.97       477
           2       0.95      0.98      0.97       452
           3       0.99      1.00      1.00       403

    accuracy                           0.98      1332
   macro avg       0.98      0.98      0.98      1332
weighted avg       0.98      0.98      0.98      1332

Model 3 (Different Features LIMITED TO 5 FEATURES) - Accuracy: 0.9474474474474475
              precision    recall  f1-score   support

           1       0.98      0.92      0.95       477
           2       0.90      0.96      0.93       452
           3       0.97      0.97      0.97       403

    accuracy                           0.95      1332
   macro avg       0.95      0.95      0.95      1332
weighted avg       0.95      0.95      0.95      1332

Model 4 (Different Features LIMITED TO 5 FEATURES) - Accuracy: 0.9624624624624625
              precision    recall  f1-score   support

           1       0.97      0.94      0.95       477
           2       0.94      0.96      0.95       452
           3       0.99      1.00      0.99       403

    accuracy                           0.96      1332
   macro avg       0.96      0.96      0.96      1332
weighted avg       0.96      0.96      0.96      1332

Model 5 (Different Features LIMITED TO 5 FEATURES) - Accuracy: 0.9421921921921922
              precision    recall  f1-score   support

           1       0.94      0.92      0.93       477
           2       0.92      0.94      0.93       452
           3       0.97      0.98      0.97       403

    accuracy                           0.94      1332
   macro avg       0.94      0.94      0.94      1332
weighted avg       0.94      0.94      0.94      1332
```

EXPERIMENTAL RESULTS AND DISCUSSION –

CLASS 1 (NORMAL)

FEATURE SET #	ACCURACY	PRECISION	RECALL	F1-SCORE
ALL	97.44%	98%	95%	97
SET 0	97.59%	99%	95%	97
SET 1	94.74%	98%	92%	95
SET 2	96.24%	97%	94%	95
SET 3	94.21%	94%	92%	93

CLASS 2 (SUSPICIOUS)

FEATURE SET #	ACCURACY	PRECISION	RECALL	F1-SCORE
ALL	97.4%	95%	98%	96
SET 0	97.59%	95%	98%	97
SET 1	94.74%	90%	96%	93
SET 2	96.24%	94%	96%	95
SET 3	94.21%	92%	94%	93

CLASS 3 (PATHOLOGICAL)

FEATURE SET #	ACCURACY	PRECISION	RECALL	F1-SCORE
ALL	97.4%	100%	100%	100
SET 0	97.59%	99%	100%	100
SET 1	94.74%	97%	97%	97
SET 2	96.24%	99%	100%	99
SET 3	94.21%	97%	98%	97

1. Inferences from the Experiment:

The performance of the Random Forest Classifier was heavily influenced by the quality and choice of features. Feature engineering, including correlation analysis and visual inspection (e.g., violin plots), helped identify which features contributed meaningfully to the model's predictions. The insights gained were as follows:

- **Highly Predictive Features:**
  Features like 'Median,' 'AC,' 'ASTV,' 'MSTV,' and 'ALTV' consistently improved classification metrics such as precision, recall, and F1-score. These features had minimal overlap in their distributions across the three fetal health states, indicating they carried significant discriminative power. For example, variability metrics like 'ASTV' and 'MSTV' captured subtle differences in fetal heart rate patterns, which are clinically significant.
- **Redundant or Non-Contributory Features:**
  Features like 'DS' and 'DR' were excluded because they were either constant or displayed strong correlations with other features. Including such features introduced noise and redundancy, which negatively impacted performance. Similarly, 'UC' and

'Width' had overlapping distributions across target classes, limiting their ability to distinguish between states.

- **Impact of Balancing:**
  Balancing the dataset with SMOTE proved critical in mitigating the bias toward the majority Normal class. This preprocessing step significantly boosted recall for the Suspicious and Pathological classes, ensuring the model effectively identified minority cases.

2. Feature Sets Leading to Near-Perfect Prediction:

- **Set 1 ('Median,' 'AC,' 'ASTV,' 'MSTV,' 'ALTV')** and **Set 2 ('AC,' 'ASTV,' 'MSTV,' 'Median,' 'DL')** demonstrated near-perfect performance. These sets included features with distinct and non-overlapping distributions across classes, enhancing the model's ability to classify samples accurately. Additionally, their reduced size improved computational efficiency without compromising predictive power.
- The selection of features in Set 1 was particularly impactful, as it combined central tendency measures ('Median') with FHR variability metrics, capturing both the distributional and variability aspects of the data.

3. Why Some Features Failed to Help in Prediction:

Features that contributed little to the model's predictions generally exhibited one or more of the following characteristics:

- **Constant Values:** Features like 'DS' and 'DR' had identical values for all instances, making them ineffective for classification.
- **High Correlation:** Features such as 'Mean' and 'Mode' were strongly correlated with others like 'Median,' leading to redundancy. Including both in the model did not provide additional information.
- **Overlapping Distributions:** Features like 'UC' and 'Width' showed significant overlap in violin plots, making it difficult for the model to distinguish between target classes based on these features.

Interpretations of Findings:

The experiment highlighted the importance of rigorous feature selection in enhancing prediction accuracy and interpretability. The combination of clinical expertise (e.g., recognizing the importance of variability metrics) and data-driven techniques (e.g., correlation heatmaps) allowed the selection of features that provided near-perfect predictions. This process underscores the necessity of balancing domain knowledge with computational methods to achieve optimal results.

CONCLUSION –

This study successfully demonstrated the application of machine learning to classify fetal health states using CTG data. The Random Forest Classifier emerged as a robust algorithm for this task, leveraging its ensemble approach to balance feature importance and improve predictions. Through preprocessing, including handling outliers, normalizing data, and addressing class imbalances, the project achieved high performance across key evaluation metrics.

The results underscore the importance of feature engineering. Selecting effective features like 'Median,' 'AC,' and variability metrics ('ASTV,' 'MSTV') played a pivotal role in achieving near-perfect classifications. Conversely, redundant or weakly discriminative features, such as 'DS' and 'DR,' were found to contribute little and were excluded. This emphasizes the necessity of correlation analysis and distribution evaluations (e.g., via violin plots) to refine the feature set.

The adoption of SMOTE for class balancing proved particularly impactful, improving the recall and F1-scores for minority classes. This ensured the model's predictions were not skewed towards the dominant Normal class, which is critical in medical diagnostics where false negatives (missed pathological cases) can have severe consequences.

In conclusion, the project demonstrated that by combining preprocessing, feature selection, and a powerful classifier, fetal health can be predicted with high accuracy and reliability. The approach offers a foundation for further enhancements, such as testing deep learning models or hybrid ensembles, which may better capture subtle patterns in the data. Future work could also involve integrating real-time CTG data and exploring interpretability frameworks to make predictions more transparent for clinical applications.