In [ ]:
```python
# Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# increase the maximum number of rows and columns displayed
pd.set_option('display.max_rows', 10000)
pd.set_option('display.max_columns', 1000)

# Load the data in a dataframe
Data=pd.read_csv('mxmh_survey_results.csv')
```

Data Exploration and Preparation

In [ ]:
```python
# Print Dataframe
Data.head()
```

Out[ ]:

	Timestamp	Age	Primary streaming service	Hours per day	While working	Instrumentalist	Composer	Fav genre	Exploratory	Foreign languages	BPM	Frequency [Classical]	Frequency [Country]	Frequency [EDM]	Fre
0	8/27/2022 19:29:02	18.0	Spotify	3.0	Yes	Yes	Yes	Latin	Yes	Yes	156.0	Rarely	Never	Rarely	
1	8/27/2022 19:57:31	63.0	Pandora	1.5	Yes	No	No	Rock	Yes	No	119.0	Sometimes	Never	Never	
2	8/27/2022 21:28:18	18.0	Spotify	4.0	No	No	No	Video game music	No	Yes	132.0	Never	Never	Very frequently	
3	8/27/2022 21:40:40	61.0	YouTube Music	2.5	Yes	No	Yes	Jazz	Yes	Yes	84.0	Sometimes	Never	Never	
4	8/27/2022 21:54:47	18.0	Spotify	4.0	Yes	No	No	R&B	Yes	No	107.0	Never	Never	Rarely	

In [ ]:
```python
# Look at the columns
Data.columns
```

Out[ ]:
```
Index(['Timestamp', 'Age', 'Primary streaming service', 'Hours per day',
       'While working', 'Instrumentalist', 'Composer', 'Fav genre',
       'Exploratory', 'Foreign languages', 'BPM', 'Frequency [Classical]',
       'Frequency [Country]', 'Frequency [EDM]', 'Frequency [Folk]',
       'Frequency [Gospel]', 'Frequency [Hip hop]', 'Frequency [Jazz]',
       'Frequency [K pop]', 'Frequency [Latin]', 'Frequency [Lofi]',
       'Frequency [Metal]', 'Frequency [Pop]', 'Frequency [R&B]',
       'Frequency [Rap]', 'Frequency [Rock]', 'Frequency [Video game music]',
       'Anxiety', 'Depression', 'Insomnia', 'OCD', 'Music effects',
       'Permissions'],
      dtype='object')
```

In [ ]:
```python
Data.describe()
```

Out[ ]:

	Age	Hours per day	BPM	Anxiety	Depression	Insomnia	OCD
count	735.000000	736.000000	6.290000e+02	736.000000	736.000000	736.000000	736.000000
mean	25.206803	3.572758	1.589948e+06	5.837636	4.796196	3.738451	2.637228
std	12.054970	3.028199	3.987261e+07	2.793054	3.028870	3.088689	2.842017
min	10.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000
25%	18.000000	2.000000	1.000000e+02	4.000000	2.000000	1.000000	0.000000
50%	21.000000	3.000000	1.200000e+02	6.000000	5.000000	3.000000	2.000000
75%	28.000000	5.000000	1.440000e+02	8.000000	7.000000	6.000000	5.000000
max	89.000000	24.000000	1.000000e+09	10.000000	10.000000	10.000000	10.000000

In [ ]:
```python
# check if the data is balanced
Data['Music effects'].value_counts(normalize=True).round(2)
```

Out[ ]:
```
Improve      0.74
No effect    0.23
Worsen       0.02
Name: Music effects, dtype: float64
```

In [ ]:
```python
# No. of rows present within the data
len(Data)
```

Out[ ]:
```
736
```

In [ ]:
```python
# Null values check
Data.isnull().sum()
```

Out[ ]:

```
Timestamp                          0
Age                                1
Primary streaming service          1
Hours per day                      0
While working                      3
Instrumentalist                    4
Composer                           1
Fav genre                          0
Exploratory                        0
Foreign languages                  4
BPM                              107
Frequency [Classical]              0
Frequency [Country]                0
Frequency [EDM]                    0
Frequency [Folk]                   0
Frequency [Gospel]                 0
Frequency [Hip hop]                0
Frequency [Jazz]                   0
Frequency [K pop]                  0
Frequency [Latin]                  0
Frequency [Lofi]                   0
Frequency [Metal]                  0
Frequency [Pop]                    0
Frequency [R&B]                    0
Frequency [Rap]                    0
Frequency [Rock]                   0
Frequency [Video game music]       0
Anxiety                            0
Depression                         0
Insomnia                           0
OCD                                0
Music effects                      8
Permissions                        0
dtype: int64
```

In [ ]:

```
# Print out all columns with missing values
for i in Data.columns:
    if Data[i].isna().sum()>0:
        print(i,'(',Data[i].dtype,')',':',Data[i].isna().sum())
```

```
Age ( float64 ) : 1
Primary streaming service ( object ) : 1
While working ( object ) : 3
Instrumentalist ( object ) : 4
Composer ( object ) : 1
Foreign languages ( object ) : 4
BPM ( float64 ) : 107
Music effects ( object ) : 8
```

In [ ]:

```
# Let's apply mean imputation for BPM since the % of missing values is high
Data['BPM'] = Data['BPM'].fillna(Data['BPM'].mean())

# Replacing nan with 0 for AGE
Data['Age']=Data['Age'].fillna(0)

# Run it again : Print out all columns with missing values
for i in Data.columns:
    if Data[i].isna().sum()>0:
        print(i,'(',Data[i].dtype,')',':',Data[i].isna().sum())
```

```
Primary streaming service ( object ) : 1
While working ( object ) : 3
Instrumentalist ( object ) : 4
Composer ( object ) : 1
Foreign languages ( object ) : 4
Music effects ( object ) : 8
```

In [ ]:

```
# Since the % of null values ain't significant, I will go ahead and drop them

Data=Data.dropna()

#Final check for null values
Data.isna().sum()
```

Out[ ]:
```
Timestamp                       0
Age                             0
Primary streaming service       0
Hours per day                   0
While working                   0
Instrumentalist                 0
Composer                        0
Fav genre                       0
Exploratory                     0
Foreign languages               0
BPM                             0
Frequency [Classical]           0
Frequency [Country]             0
Frequency [EDM]                 0
Frequency [Folk]                0
Frequency [Gospel]              0
Frequency [Hip hop]             0
Frequency [Jazz]                0
Frequency [K pop]               0
Frequency [Latin]               0
Frequency [Lofi]                0
Frequency [Metal]               0
Frequency [Pop]                 0
Frequency [R&B]                 0
Frequency [Rap]                 0
Frequency [Rock]                0
Frequency [Video game music]    0
Anxiety                         0
Depression                      0
Insomnia                        0
OCD                             0
Music effects                   0
Permissions                     0
dtype: int64
```

In [ ]:
```
# From 736 rows to 719 rows after dropping na
len(Data)
```

Out[ ]:
```
719
```

In [ ]:
```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 719 entries, 2 to 735
Data columns (total 33 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Timestamp                     719 non-null    object 
 1   Age                           719 non-null    float64
 2   Primary streaming service     719 non-null    object 
 3   Hours per day                 719 non-null    float64
 4   While working                 719 non-null    object 
 5   Instrumentalist               719 non-null    object 
 6   Composer                      719 non-null    object 
 7   Fav genre                     719 non-null    object 
 8   Exploratory                   719 non-null    object 
 9   Foreign languages             719 non-null    object 
 10  BPM                           719 non-null    float64
 11  Frequency [Classical]         719 non-null    object 
 12  Frequency [Country]           719 non-null    object 
 13  Frequency [EDM]               719 non-null    object 
 14  Frequency [Folk]              719 non-null    object 
 15  Frequency [Gospel]            719 non-null    object 
 16  Frequency [Hip hop]           719 non-null    object 
 17  Frequency [Jazz]              719 non-null    object 
 18  Frequency [K pop]             719 non-null    object 
 19  Frequency [Latin]             719 non-null    object 
 20  Frequency [Lofi]              719 non-null    object 
 21  Frequency [Metal]             719 non-null    object 
 22  Frequency [Pop]               719 non-null    object 
 23  Frequency [R&B]               719 non-null    object 
 24  Frequency [Rap]               719 non-null    object 
 25  Frequency [Rock]              719 non-null    object 
 26  Frequency [Video game music]  719 non-null    object 
 27  Anxiety                       719 non-null    float64
 28  Depression                    719 non-null    float64
 29  Insomnia                      719 non-null    float64
 30  OCD                           719 non-null    float64
 31  Music effects                 719 non-null    object 
 32  Permissions                   719 non-null    object 
dtypes: float64(7), object(26)
memory usage: 191.0+ KB
```

In [ ]:
```
# Convert "timestamp" column to Timestamp Datatype
Data['Timestamp'] = pd.to_datetime(Data['Timestamp'])
Data['Timestamp'].dtype
```

Out[ ]:
```
dtype('<M8[ns]')
```

In [ ]:
```
Categorical_variables=[]
# Get value counts of only categorical data
for i in Data.columns:
    if Data[i].dtype==object:

        print(Data[i].value_counts())
```

```
        Categorical_variables.append(i)
        print()
```

```
Spotify                              451
YouTube Music                         90
I do not use a streaming service.     69
Apple Music                           50
Other streaming service               49
Pandora                               10
Name: Primary streaming service, dtype: int64

Yes    568
No     151
Name: While working, dtype: int64

No     490
Yes    229
Name: Instrumentalist, dtype: int64

No     595
Yes    124
Name: Composer, dtype: int64

Rock                184
Pop                 114
Metal                87
Classical            51
Video game music     43
EDM                  36
R&B                  35
Hip hop              35
Folk                 29
Country              24
Rap                  22
K pop                21
Jazz                 20
Lofi                 10
Gospel                6
Latin                 2
Name: Fav genre, dtype: int64

Yes    515
No     204
Name: Exploratory, dtype: int64

Yes    396
No     323
Name: Foreign languages, dtype: int64

Rarely             254
Sometimes          193
Never              166
Very frequently    106
Name: Frequency [Classical], dtype: int64

Never              333
Rarely             230
Sometimes          108
Very frequently     48
Name: Frequency [Country], dtype: int64

Never              296
Rarely             191
Sometimes          144
Very frequently     88
Name: Frequency [EDM], dtype: int64

Never              284
Rarely             217
Sometimes          142
Very frequently     76
Name: Frequency [Folk], dtype: int64

Never              523
Rarely             132
Sometimes           50
Very frequently     14
Name: Frequency [Gospel], dtype: int64

Sometimes          214
Rarely             209
Never              174
Very frequently    122
Name: Frequency [Hip hop], dtype: int64

Never              252
Rarely             244
Sometimes          171
Very frequently     52
Name: Frequency [Jazz], dtype: int64

Never              407
Rarely             174
Very frequently     71
Sometimes           67
Name: Frequency [K pop], dtype: int64

Never              432
Rarely             171
Sometimes           85
```

```
Very frequently     31
Name: Frequency [Latin], dtype: int64

Never              273
Rarely             204
Sometimes          158
Very frequently     84
Name: Frequency [Lofi], dtype: int64

Never              256
Rarely             189
Very frequently    144
Sometimes          130
Name: Frequency [Metal], dtype: int64

Very frequently    271
Sometimes          256
Rarely             140
Never               52
Name: Frequency [Pop], dtype: int64

Never              220
Rarely             208
Sometimes          176
Very frequently    115
Name: Frequency [R&B], dtype: int64

Rarely             211
Never              193
Sometimes          190
Very frequently    125
Name: Frequency [Rap], dtype: int64

Very frequently    324
Sometimes          214
Rarely              94
Never               87
Name: Frequency [Rock], dtype: int64

Never              230
Rarely             194
Sometimes          180
Very frequently    115
Name: Frequency [Video game music], dtype: int64

Improve      536
No effect    166
Worsen        17
Name: Music effects, dtype: int64

I understand.    719
Name: Permissions, dtype: int64
```

In [ ]:

```
# Drop column called "Permission" as it gives no insights
Data = Data.drop('Permissions', axis=1)
```

In [ ]:

```
Categorical_variables.remove('Permissions')
for i in Categorical_variables:
    print(i,':',Data[i].unique())
    print(len(Data[i].unique()))
    print()
```

```
Primary streaming service : ['Spotify' 'YouTube Music' 'I do not use a streaming service.'
 'Apple Music' 'Other streaming service' 'Pandora']
6

While working : ['No' 'Yes']
2

Instrumentalist : ['No' 'Yes']
2

Composer : ['No' 'Yes']
2

Fav genre : ['Video game music' 'Jazz' 'R&B' 'K pop' 'Rock' 'Country' 'EDM' 'Hip hop'
 'Pop' 'Rap' 'Classical' 'Metal' 'Folk' 'Lofi' 'Gospel' 'Latin']
16

Exploratory : ['No' 'Yes']
2

Foreign languages : ['Yes' 'No']
2

Frequency [Classical] : ['Never' 'Sometimes' 'Rarely' 'Very frequently']
4

Frequency [Country] : ['Never' 'Sometimes' 'Very frequently' 'Rarely']
4

Frequency [EDM] : ['Very frequently' 'Never' 'Rarely' 'Sometimes']
4

Frequency [Folk] : ['Never' 'Rarely' 'Sometimes' 'Very frequently']
4

Frequency [Gospel] : ['Never' 'Sometimes' 'Rarely' 'Very frequently']
4

Frequency [Hip hop] : ['Rarely' 'Never' 'Very frequently' 'Sometimes']
4

Frequency [Jazz] : ['Rarely' 'Very frequently' 'Never' 'Sometimes']
4

Frequency [K pop] : ['Very frequently' 'Sometimes' 'Never' 'Rarely']
4

Frequency [Latin] : ['Never' 'Very frequently' 'Sometimes' 'Rarely']
4

Frequency [Lofi] : ['Sometimes' 'Very frequently' 'Rarely' 'Never']
4

Frequency [Metal] : ['Sometimes' 'Never' 'Rarely' 'Very frequently']
4

Frequency [Pop] : ['Rarely' 'Sometimes' 'Very frequently' 'Never']
4

Frequency [R&B] : ['Never' 'Sometimes' 'Very frequently' 'Rarely']
4

Frequency [Rap] : ['Rarely' 'Never' 'Very frequently' 'Sometimes']
4

Frequency [Rock] : ['Rarely' 'Never' 'Very frequently' 'Sometimes']
4

Frequency [Video game music] : ['Very frequently' 'Never' 'Rarely' 'Sometimes']
4

Music effects : ['No effect' 'Improve' 'Worsen']
3
```

In [ ]:

```python
# create dummy variables for all categorical columns
Data_clean = pd.get_dummies(Data[Categorical_variables])

# concatenate the dummy variables with the original DataFrame
Data_Final = pd.concat([Data.drop(Categorical_variables, axis=1), Data_clean], axis=1)
```

In [ ]:

```python
Data_Final.columns
```

Out[ ]:

```
Index(['Timestamp', 'Age', 'Hours per day', 'BPM', 'Anxiety', 'Depression',
       'Insomnia', 'OCD', 'Primary streaming service_Apple Music',
       'Primary streaming service_I do not use a streaming service.',
       ...
       'Frequency [Rock]_Rarely', 'Frequency [Rock]_Sometimes',
       'Frequency [Rock]_Very frequently',
       'Frequency [Video game music]_Never',
       'Frequency [Video game music]_Rarely',
       'Frequency [Video game music]_Sometimes',
       'Frequency [Video game music]_Very frequently', 'Music effects_Improve',
       'Music effects_No effect', 'Music effects_Worsen'],
      dtype='object', length=107)
```

In [ ]:
```python
# The final cleant data
Data_Final.head()
```

Out[ ]:

	Timestamp	Age	Hours per day	BPM	Anxiety	Depression	Insomnia	OCD	Primary streaming service_Apple Music	Primary streaming service_I do not use a streaming service.	Primary streaming service_Other streaming service	Primary streaming service_Pandora	Primary streaming service_Spotify	servic
2	2022-08-27 21:28:18	18.0	4.0	132.0	7.0	7.0	10.0	2.0	0	0	0	0	1	
3	2022-08-27 21:40:40	61.0	2.5	84.0	9.0	7.0	3.0	3.0	0	0	0	0	0	
4	2022-08-27 21:54:47	18.0	4.0	107.0	7.0	2.0	5.0	9.0	0	0	0	0	1	
5	2022-08-27 21:56:50	18.0	5.0	86.0	8.0	8.0	7.0	7.0	0	0	0	0	1	
6	2022-08-27 22:00:29	18.0	3.0	66.0	4.0	8.0	6.0	0.0	0	0	0	0	0	

In [ ]:
```python
len(Data_Final.columns)
```

Out[ ]:
```
107
```

In [ ]:
```python
# Get value counts of only categorical data
for i in Data_Final.columns:
    if Data_Final[i].dtype==object:
        print(i)
```

In [ ]:
```python
corr_matrix = Data_Final.corr()
corr_matrix[['Music effects_Improve','Music effects_No effect','Music effects_Worsen']].round(2)
```

Out[ ]:

	Music effects_Improve	Music effects_No effect	Music effects_Worsen
Age	-0.06	0.07	-0.03
Hours per day	0.03	-0.02	-0.04
BPM	-0.06	0.07	-0.01
Anxiety	0.12	-0.15	0.05
Depression	0.02	-0.07	0.12
Insomnia	0.00	-0.02	0.04
OCD	0.04	-0.05	0.03
Primary streaming service_Apple Music	-0.00	0.01	-0.01
Primary streaming service_I do not use a streaming service.	-0.07	0.07	0.01
Primary streaming service_Other streaming service	-0.02	0.04	-0.04
Primary streaming service_Pandora	0.07	-0.07	-0.02
Primary streaming service_Spotify	0.04	-0.06	0.04
Primary streaming service_YouTube Music	-0.01	0.02	-0.03
While working_No	-0.18	0.16	0.05
While working_Yes	0.18	-0.16	-0.05
Instrumentalist_No	-0.10	0.10	0.03
Instrumentalist_Yes	0.10	-0.10	-0.03
Composer_No	-0.09	0.08	0.02
Composer_Yes	0.09	-0.08	-0.02
Fav genre_Classical	-0.01	0.02	-0.01
Fav genre_Country	0.02	-0.01	-0.03
Fav genre_EDM	0.05	-0.03	-0.04
Fav genre_Folk	0.02	-0.01	-0.03
Fav genre_Gospel	0.05	-0.05	-0.01
Fav genre_Hip hop	0.07	-0.06	-0.04
Fav genre_Jazz	0.02	-0.01	-0.03
Fav genre_K pop	0.03	-0.02	-0.03
Fav genre_Latin	-0.03	0.03	-0.01
Fav genre_Lofi	0.07	-0.07	-0.02
Fav genre_Metal	0.01	0.01	-0.06
Fav genre_Pop	0.00	-0.01	0.03
Fav genre_R&B	-0.00	0.01	-0.04
Fav genre_Rap	0.01	-0.02	0.03
Fav genre_Rock	-0.08	0.06	0.06
Fav genre_Video game music	-0.08	0.04	0.12
Exploratory_No	-0.15	0.14	0.04
Exploratory_Yes	0.15	-0.14	-0.04
Foreign languages_No	-0.01	0.02	-0.01
Foreign languages_Yes	0.01	-0.02	0.01
Frequency [Classical]_Never	0.02	-0.02	0.00
Frequency [Classical]_Rarely	0.02	-0.02	-0.00
Frequency [Classical]_Sometimes	-0.03	0.03	0.01
Frequency [Classical]_Very frequently	-0.01	0.01	-0.01
Frequency [Country]_Never	-0.05	0.04	0.04
Frequency [Country]_Rarely	-0.01	0.01	-0.01
Frequency [Country]_Sometimes	0.06	-0.05	-0.04
Frequency [Country]_Very frequently	0.04	-0.04	-0.00
Frequency [EDM]_Never	-0.08	0.06	0.07
Frequency [EDM]_Rarely	0.10	-0.08	-0.05
Frequency [EDM]_Sometimes	-0.03	0.04	-0.03
Frequency [EDM]_Very frequently	0.02	-0.02	-0.00
Frequency [Folk]_Never	-0.00	-0.01	0.04
Frequency [Folk]_Rarely	0.02	-0.02	-0.00
Frequency [Folk]_Sometimes	-0.01	0.02	-0.03
Frequency [Folk]_Very frequently	-0.01	0.02	-0.02
Frequency [Gospel]_Never	-0.09	0.08	0.01
Frequency [Gospel]_Rarely	0.06	-0.07	0.02

	Music effects_Improve	Music effects_No effect	Music effects_Worsen
Frequency [Gospel]_Sometimes	0.01	0.01	-0.04
Frequency [Gospel]_Very frequently	0.08	-0.08	-0.02
Frequency [Hip hop]_Never	-0.09	0.11	-0.02
Frequency [Hip hop]_Rarely	0.02	-0.02	-0.02
Frequency [Hip hop]_Sometimes	0.07	-0.08	0.02
Frequency [Hip hop]_Very frequently	0.00	-0.01	0.03
Frequency [Jazz]_Never	-0.07	0.07	0.02
Frequency [Jazz]_Rarely	0.03	-0.04	0.02
Frequency [Jazz]_Sometimes	0.03	-0.01	-0.04
Frequency [Jazz]_Very frequently	0.03	-0.03	-0.01
Frequency [K pop]_Never	-0.05	0.06	-0.03
Frequency [K pop]_Rarely	0.00	-0.02	0.06
Frequency [K pop]_Sometimes	0.02	-0.03	0.01
Frequency [K pop]_Very frequently	0.05	-0.04	-0.05
Frequency [Latin]_Never	-0.05	0.05	-0.00
Frequency [Latin]_Rarely	0.00	0.00	-0.02
Frequency [Latin]_Sometimes	0.06	-0.06	-0.00
Frequency [Latin]_Very frequently	0.01	-0.04	0.06
Frequency [Lofi]_Never	-0.07	0.06	0.03
Frequency [Lofi]_Rarely	-0.00	0.01	-0.04
Frequency [Lofi]_Sometimes	0.06	-0.07	0.03
Frequency [Lofi]_Very frequently	0.03	-0.02	-0.03
Frequency [Metal]_Never	0.02	-0.01	-0.04
Frequency [Metal]_Rarely	0.00	-0.01	0.03
Frequency [Metal]_Sometimes	0.00	-0.01	0.02
Frequency [Metal]_Very frequently	-0.03	0.03	-0.01
Frequency [Pop]_Never	0.02	-0.03	0.03
Frequency [Pop]_Rarely	-0.10	0.11	-0.01
Frequency [Pop]_Sometimes	0.02	-0.01	-0.02
Frequency [Pop]_Very frequently	0.05	-0.06	0.01
Frequency [R&B]_Never	-0.08	0.08	0.02
Frequency [R&B]_Rarely	-0.06	0.04	0.06
Frequency [R&B]_Sometimes	0.09	-0.07	-0.07
Frequency [R&B]_Very frequently	0.07	-0.07	-0.02
Frequency [Rap]_Never	-0.04	0.04	-0.01
Frequency [Rap]_Rarely	-0.02	0.02	0.00
Frequency [Rap]_Sometimes	0.02	-0.02	0.01
Frequency [Rap]_Very frequently	0.04	-0.04	0.00
Frequency [Rock]_Never	0.02	-0.04	0.05
Frequency [Rock]_Rarely	0.01	-0.01	-0.01
Frequency [Rock]_Sometimes	0.00	-0.00	-0.00
Frequency [Rock]_Very frequently	-0.02	0.03	-0.03
Frequency [Video game music]_Never	-0.03	0.03	-0.01
Frequency [Video game music]_Rarely	0.01	0.00	-0.03
Frequency [Video game music]_Sometimes	0.06	-0.05	-0.03
Frequency [Video game music]_Very frequently	-0.04	0.01	0.08
Music effects_Improve	1.00	-0.94	-0.27
Music effects_No effect	-0.94	1.00	-0.09
Music effects_Worsen	-0.27	-0.09	1.00

In [ ]:

```
Data_Final=Data_Final.drop('Timestamp', axis=1)
```

In [ ]:

```
# # Data_Final.info()
# # Convert all columns to single datatype
# for i in Data_Final.columns:
#         if Data_Final[i].dtype!=float:
#                 #print(i)
#                 Data_Final[i].astype(float)
```

In [ ]:
```
# COnverting to single data type
Data_Final=Data_Final.astype(float)
```

In [ ]:
```
Data_Final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 719 entries, 2 to 735
Columns: 106 entries, Age to Music effects_Worsen
dtypes: float64(106)
memory usage: 601.0 KB
```

In [ ]:
```
# # No strong correlation seen above
# # Using Decision Trees

# from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

#  identify the features and target variable
X = Data_Final.drop(['Music effects_Improve','Music effects_No effect','Music effects_Worsen'], axis=1)
y = Data_Final[['Music effects_Improve','Music effects_No effect','Music effects_Worsen']]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# # create a decision tree classifier object
# clf = DecisionTreeClassifier()

# # train the classifier on the training data
# clf.fit(X_train, y_train)
```

In [ ]:
```
# from sklearn.metrics import accuracy_score,precision_score,recall_score
# # make predictions on the testing data
# y_pred = clf.predict(X_test)

# # evaluate the accuracy of the classifier
# accuracy = accuracy_score(y_test, y_pred)
# # precision = precision_score(y_test, y_pred)
# # recall = recall_score(y_test, y_pred)

# print("Accuracy:", accuracy.round(2)*100,'%')
# # print("Precision",precision.round(2)*100,'%')
# # print("Recall",recall.round(2)*100,'%')
```

```
Accuracy: 65.0 %
```

Phase 1 : Effect of Dropout layers in model performance

Part 1 : Without Using drop out layers

In [ ]:
```
# Define the model architecture using Sequential model
import tensorflow as tf
model_No_DO = tf.keras.models.Sequential()

# Taking 50 units with 2 hidden layers
#model_2.add(tf.keras.layers.Dense(200, activation='relu', input_dim=X_train.shape[1]))
model_No_DO.add(tf.keras.layers.Dense(100, activation='relu',input_dim=X_train.shape[1]))
model_No_DO.add(tf.keras.layers.Dense(75, activation='relu'))
model_No_DO.add(tf.keras.layers.Dense(55, activation='relu'))
model_No_DO.add(tf.keras.layers.Dense(25, activation='relu'))
model_No_DO.add(tf.keras.layers.Dense(3, activation='softmax'))

# Compile the model
model_No_DO.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history_No_DO = model_No_DO.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=32)
```

```
Epoch 1/50
18/18 [==============================] - 2s 85ms/step - loss: 1392.9261 - accuracy: 0.5739 - val_loss: 304.1428 - val_accuracy:
0.8056
Epoch 2/50
18/18 [==============================] - 0s 8ms/step - loss: 19555.9492 - accuracy: 0.4609 - val_loss: 3038.8372 - val_accurac
y: 0.7014
Epoch 3/50
18/18 [==============================] - 0s 10ms/step - loss: 33291.2266 - accuracy: 0.5461 - val_loss: 6698.9556 - val_accurac
y: 0.2083
Epoch 4/50
18/18 [==============================] - 0s 8ms/step - loss: 11862.8369 - accuracy: 0.3861 - val_loss: 7248.3955 - val_accurac
y: 0.7292
Epoch 5/50
18/18 [==============================] - 0s 8ms/step - loss: 4151.6743 - accuracy: 0.6991 - val_loss: 409.0533 - val_accuracy:
0.8056
Epoch 6/50
18/18 [==============================] - 0s 10ms/step - loss: 25100.2617 - accuracy: 0.4348 - val_loss: 8274.5293 - val_accurac
y: 0.3056
Epoch 7/50
18/18 [==============================] - 0s 21ms/step - loss: 10293.6660 - accuracy: 0.4574 - val_loss: 5289.1772 - val_accurac
y: 0.6667
Epoch 8/50
18/18 [==============================] - 0s 8ms/step - loss: 4990.7837 - accuracy: 0.6609 - val_loss: 959.3542 - val_accuracy:
0.7222
Epoch 9/50
18/18 [==============================] - 0s 8ms/step - loss: 8167.3105 - accuracy: 0.6035 - val_loss: 7034.8633 - val_accuracy:
0.4444
Epoch 10/50
18/18 [==============================] - 0s 9ms/step - loss: 10719.4531 - accuracy: 0.5270 - val_loss: 7498.8091 - val_accurac
y: 0.6528
Epoch 11/50
18/18 [==============================] - 0s 9ms/step - loss: 10186.9893 - accuracy: 0.6278 - val_loss: 6575.1309 - val_accurac
y: 0.7153
Epoch 12/50
18/18 [==============================] - 0s 12ms/step - loss: 8182.5020 - accuracy: 0.6765 - val_loss: 4507.1440 - val_accurac
y: 0.7500
Epoch 13/50
18/18 [==============================] - 0s 19ms/step - loss: 4215.9546 - accuracy: 0.6730 - val_loss: 578.6076 - val_accuracy:
0.7500
Epoch 14/50
18/18 [==============================] - 0s 14ms/step - loss: 11654.3555 - accuracy: 0.6296 - val_loss: 6909.9922 - val_accurac
y: 0.7153
Epoch 15/50
18/18 [==============================] - 0s 9ms/step - loss: 10328.9482 - accuracy: 0.6504 - val_loss: 7287.8271 - val_accurac
y: 0.7153
Epoch 16/50
18/18 [==============================] - 0s 10ms/step - loss: 9991.9092 - accuracy: 0.6609 - val_loss: 6543.9956 - val_accurac
y: 0.7431
Epoch 17/50
18/18 [==============================] - 0s 10ms/step - loss: 8706.5596 - accuracy: 0.6765 - val_loss: 5692.0640 - val_accurac
y: 0.7500
Epoch 18/50
18/18 [==============================] - 0s 9ms/step - loss: 7465.7124 - accuracy: 0.6748 - val_loss: 4687.3237 - val_accuracy:
0.7639
Epoch 19/50
18/18 [==============================] - 0s 9ms/step - loss: 6085.3184 - accuracy: 0.6748 - val_loss: 3725.1702 - val_accuracy:
0.7500
Epoch 20/50
18/18 [==============================] - 0s 9ms/step - loss: 4735.7637 - accuracy: 0.6730 - val_loss: 2720.2434 - val_accuracy:
0.7500
Epoch 21/50
18/18 [==============================] - 0s 16ms/step - loss: 3194.7957 - accuracy: 0.6783 - val_loss: 1679.5028 - val_accurac
y: 0.7639
Epoch 22/50
18/18 [==============================] - 0s 10ms/step - loss: 1571.5728 - accuracy: 0.6730 - val_loss: 373.1327 - val_accuracy:
0.7708
Epoch 23/50
18/18 [==============================] - 0s 9ms/step - loss: 5636.0200 - accuracy: 0.6626 - val_loss: 5338.7251 - val_accuracy:
0.7500
Epoch 24/50
18/18 [==============================] - 0s 9ms/step - loss: 7114.7241 - accuracy: 0.6800 - val_loss: 2452.0906 - val_accuracy:
0.7292
Epoch 25/50
18/18 [==============================] - 0s 8ms/step - loss: 32567.4121 - accuracy: 0.5252 - val_loss: 7445.7534 - val_accurac
y: 0.4722
Epoch 26/50
18/18 [==============================] - 0s 9ms/step - loss: 11560.6436 - accuracy: 0.6417 - val_loss: 8173.4121 - val_accurac
y: 0.7361
Epoch 27/50
18/18 [==============================] - 0s 9ms/step - loss: 10078.1406 - accuracy: 0.6765 - val_loss: 6088.8135 - val_accurac
y: 0.7153
Epoch 28/50
18/18 [==============================] - 0s 7ms/step - loss: 6101.9106 - accuracy: 0.6748 - val_loss: 1647.5844 - val_accuracy:
0.7292
Epoch 29/50
18/18 [==============================] - 0s 8ms/step - loss: 3680.9644 - accuracy: 0.6974 - val_loss: 1613.7698 - val_accuracy:
0.6111
Epoch 30/50
18/18 [==============================] - 0s 16ms/step - loss: 13000.3135 - accuracy: 0.4991 - val_loss: 12790.8926 - val_accura
cy: 0.7014
Epoch 31/50
18/18 [==============================] - 0s 23ms/step - loss: 11711.0156 - accuracy: 0.6278 - val_loss: 2209.7705 - val_accurac
y: 0.6806
Epoch 32/50
18/18 [==============================] - 0s 8ms/step - loss: 11110.2754 - accuracy: 0.6452 - val_loss: 10610.0039 - val_accurac
y: 0.7361
Epoch 33/50
```

```
18/18 [==============================] - 0s 8ms/step - loss: 21201.3301 - accuracy: 0.6383 - val_loss: 16969.6211 - val_accurac
y: 0.7222
Epoch 34/50
18/18 [==============================] - 0s 11ms/step - loss: 22844.6113 - accuracy: 0.6574 - val_loss: 14473.7070 - val_accura
cy: 0.7222
Epoch 35/50
18/18 [==============================] - 0s 10ms/step - loss: 17330.1484 - accuracy: 0.6800 - val_loss: 8834.2773 - val_accurac
y: 0.7500
Epoch 36/50
18/18 [==============================] - 0s 8ms/step - loss: 5909.7505 - accuracy: 0.6730 - val_loss: 433.6952 - val_accuracy:
0.7639
Epoch 37/50
18/18 [==============================] - 0s 11ms/step - loss: 25563.3047 - accuracy: 0.5548 - val_loss: 16643.7207 - val_accura
cy: 0.7361
Epoch 38/50
18/18 [==============================] - 0s 8ms/step - loss: 26027.6621 - accuracy: 0.6574 - val_loss: 18335.4277 - val_accurac
y: 0.7083
Epoch 39/50
18/18 [==============================] - 0s 10ms/step - loss: 24862.2715 - accuracy: 0.6696 - val_loss: 16019.3945 - val_accura
cy: 0.7500
Epoch 40/50
18/18 [==============================] - 0s 11ms/step - loss: 21267.4316 - accuracy: 0.6800 - val_loss: 13895.0752 - val_accura
cy: 0.7500
Epoch 41/50
18/18 [==============================] - 0s 10ms/step - loss: 18278.1621 - accuracy: 0.6783 - val_loss: 11503.4287 - val_accura
cy: 0.7292
Epoch 42/50
18/18 [==============================] - 0s 8ms/step - loss: 15067.2891 - accuracy: 0.6939 - val_loss: 9501.6162 - val_accurac
y: 0.7639
Epoch 43/50
18/18 [==============================] - 0s 9ms/step - loss: 12387.0977 - accuracy: 0.6887 - val_loss: 7690.4062 - val_accurac
y: 0.7569
Epoch 44/50
18/18 [==============================] - 0s 14ms/step - loss: 9706.5293 - accuracy: 0.6817 - val_loss: 5507.7065 - val_accurac
y: 0.6528
Epoch 45/50
18/18 [==============================] - 0s 7ms/step - loss: 5789.7295 - accuracy: 0.6730 - val_loss: 2630.3376 - val_accuracy:
0.7500
Epoch 46/50
18/18 [==============================] - 0s 7ms/step - loss: 1756.7434 - accuracy: 0.6904 - val_loss: 133.6392 - val_accuracy:
0.8056
Epoch 47/50
18/18 [==============================] - 0s 19ms/step - loss: 20008.1895 - accuracy: 0.7061 - val_loss: 10675.5342 - val_accura
cy: 0.7500
Epoch 48/50
18/18 [==============================] - 0s 8ms/step - loss: 18065.5977 - accuracy: 0.6852 - val_loss: 13430.6523 - val_accurac
y: 0.7500
Epoch 49/50
18/18 [==============================] - 0s 7ms/step - loss: 17882.9844 - accuracy: 0.6870 - val_loss: 11531.8164 - val_accurac
y: 0.7361
Epoch 50/50
18/18 [==============================] - 0s 10ms/step - loss: 14908.2979 - accuracy: 0.6817 - val_loss: 9344.7578 - val_accurac
y: 0.7361
```

Part 2 : Using drop out layers

In [ ]:

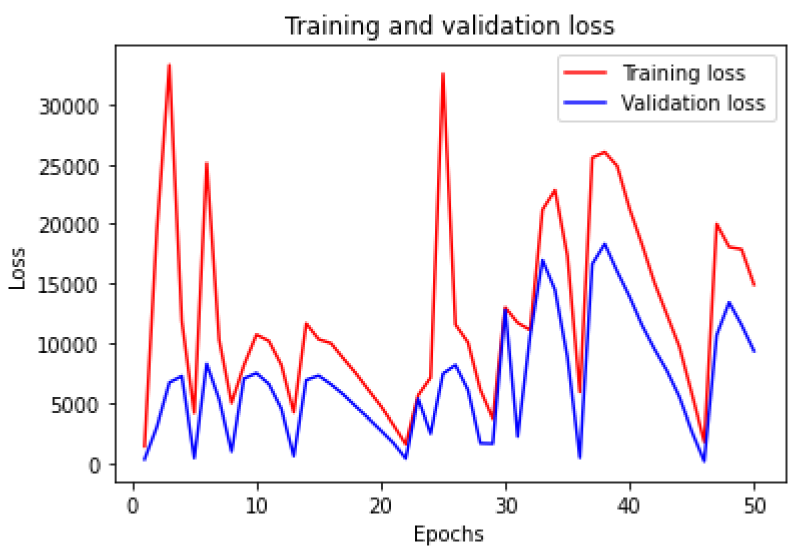
```
# Define the model architecture using Sequential model
import tensorflow as tf
model_DO = tf.keras.models.Sequential()

# Taking 50 units with 2 hidden layers
#model_2.add(tf.keras.layers.Dense(200, activation='relu', input_dim=X_train.shape[1]))
model_DO.add(tf.keras.layers.Dense(100, activation='sigmoid',input_dim=X_train.shape[1]))

# Adding one layer with drop out rate of 30%
model_DO.add(tf.keras.layers.Dropout(0.3))
model_DO.add(tf.keras.layers.Dense(75, activation='sigmoid'))

# Adding one layer with drop out rate of 30%
model_DO.add(tf.keras.layers.Dropout(0.2))
model_DO.add(tf.keras.layers.Dense(55, activation='sigmoid'))
#model_DO.add(tf.keras.layers.Dense(25, activation='relu'))

# Adding one layer with drop out rate of 30%
model_DO.add(tf.keras.layers.Dropout(0.2))
model_DO.add(tf.keras.layers.Dense(3, activation='softmax'))

# Compile the model
model_DO.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history_DO = model_DO.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=32)
```

```
Epoch 1/50
18/18 [==============================] - 4s 161ms/step - loss: 1.1675 - accuracy: 0.3791 - val_loss: 0.6635 - val_accuracy: 0.8
056
Epoch 2/50
18/18 [==============================] - 0s 19ms/step - loss: 0.7440 - accuracy: 0.7235 - val_loss: 0.5536 - val_accuracy: 0.80
56
Epoch 3/50
18/18 [==============================] - 0s 13ms/step - loss: 0.7047 - accuracy: 0.7130 - val_loss: 0.5486 - val_accuracy: 0.80
56
Epoch 4/50
18/18 [==============================] - 0s 17ms/step - loss: 0.6954 - accuracy: 0.7235 - val_loss: 0.5378 - val_accuracy: 0.80
56
Epoch 5/50
18/18 [==============================] - 0s 19ms/step - loss: 0.7032 - accuracy: 0.7270 - val_loss: 0.5445 - val_accuracy: 0.80
56
Epoch 6/50
18/18 [==============================] - 0s 16ms/step - loss: 0.6921 - accuracy: 0.7235 - val_loss: 0.5341 - val_accuracy: 0.80
56
Epoch 7/50
18/18 [==============================] - 0s 16ms/step - loss: 0.7092 - accuracy: 0.7217 - val_loss: 0.5390 - val_accuracy: 0.80
56
Epoch 8/50
18/18 [==============================] - 0s 12ms/step - loss: 0.6886 - accuracy: 0.7304 - val_loss: 0.5393 - val_accuracy: 0.80
56
Epoch 9/50
18/18 [==============================] - 0s 13ms/step - loss: 0.6940 - accuracy: 0.7252 - val_loss: 0.5466 - val_accuracy: 0.80
56
Epoch 10/50
18/18 [==============================] - 0s 11ms/step - loss: 0.6932 - accuracy: 0.7270 - val_loss: 0.5363 - val_accuracy: 0.80
56
Epoch 11/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6804 - accuracy: 0.7304 - val_loss: 0.5374 - val_accuracy: 0.80
56
Epoch 12/50
18/18 [==============================] - 0s 14ms/step - loss: 0.6900 - accuracy: 0.7270 - val_loss: 0.5363 - val_accuracy: 0.80
56
Epoch 13/50
18/18 [==============================] - 0s 27ms/step - loss: 0.6838 - accuracy: 0.7304 - val_loss: 0.5393 - val_accuracy: 0.80
56
Epoch 14/50
18/18 [==============================] - 0s 15ms/step - loss: 0.6893 - accuracy: 0.7304 - val_loss: 0.5407 - val_accuracy: 0.80
56
Epoch 15/50
18/18 [==============================] - 0s 23ms/step - loss: 0.6977 - accuracy: 0.7322 - val_loss: 0.5385 - val_accuracy: 0.80
56
Epoch 16/50
18/18 [==============================] - 0s 15ms/step - loss: 0.6960 - accuracy: 0.7304 - val_loss: 0.5394 - val_accuracy: 0.80
56
Epoch 17/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6924 - accuracy: 0.7304 - val_loss: 0.5413 - val_accuracy: 0.80
56
Epoch 18/50
18/18 [==============================] - 0s 12ms/step - loss: 0.6774 - accuracy: 0.7322 - val_loss: 0.5360 - val_accuracy: 0.80
56
Epoch 19/50
18/18 [==============================] - 0s 19ms/step - loss: 0.6855 - accuracy: 0.7287 - val_loss: 0.5356 - val_accuracy: 0.80
56
Epoch 20/50
18/18 [==============================] - 0s 12ms/step - loss: 0.6693 - accuracy: 0.7304 - val_loss: 0.5336 - val_accuracy: 0.80
56
Epoch 21/50
18/18 [==============================] - 0s 17ms/step - loss: 0.6822 - accuracy: 0.7322 - val_loss: 0.5308 - val_accuracy: 0.80
56
Epoch 22/50
18/18 [==============================] - 0s 13ms/step - loss: 0.6841 - accuracy: 0.7304 - val_loss: 0.5392 - val_accuracy: 0.80
56
Epoch 23/50
18/18 [==============================] - 0s 16ms/step - loss: 0.6780 - accuracy: 0.7304 - val_loss: 0.5327 - val_accuracy: 0.80
56
Epoch 24/50
18/18 [==============================] - 0s 12ms/step - loss: 0.6810 - accuracy: 0.7304 - val_loss: 0.5251 - val_accuracy: 0.80
56
Epoch 25/50
18/18 [==============================] - 0s 12ms/step - loss: 0.6651 - accuracy: 0.7304 - val_loss: 0.5307 - val_accuracy: 0.80
56
Epoch 26/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6664 - accuracy: 0.7304 - val_loss: 0.5315 - val_accuracy: 0.80
56
Epoch 27/50
18/18 [==============================] - 0s 16ms/step - loss: 0.6558 - accuracy: 0.7304 - val_loss: 0.5243 - val_accuracy: 0.80
56
Epoch 28/50
18/18 [==============================] - 0s 13ms/step - loss: 0.6681 - accuracy: 0.7287 - val_loss: 0.5269 - val_accuracy: 0.80
56
Epoch 29/50
18/18 [==============================] - 0s 25ms/step - loss: 0.6589 - accuracy: 0.7304 - val_loss: 0.5236 - val_accuracy: 0.80
56
Epoch 30/50
18/18 [==============================] - 0s 9ms/step - loss: 0.6628 - accuracy: 0.7287 - val_loss: 0.5159 - val_accuracy: 0.805
6
Epoch 31/50
18/18 [==============================] - 0s 11ms/step - loss: 0.6510 - accuracy: 0.7270 - val_loss: 0.5117 - val_accuracy: 0.80
56
Epoch 32/50
18/18 [==============================] - 0s 12ms/step - loss: 0.6587 - accuracy: 0.7235 - val_loss: 0.5298 - val_accuracy: 0.80
56
Epoch 33/50
```

```
18/18 [==============================] - 0s 9ms/step - loss: 0.6626 - accuracy: 0.7270 - val_loss: 0.5194 - val_accuracy: 0.805
6
Epoch 34/50
18/18 [==============================] - 0s 13ms/step - loss: 0.6364 - accuracy: 0.7339 - val_loss: 0.5224 - val_accuracy: 0.80
56
Epoch 35/50
18/18 [==============================] - 0s 14ms/step - loss: 0.6341 - accuracy: 0.7391 - val_loss: 0.5066 - val_accuracy: 0.80
56
Epoch 36/50
18/18 [==============================] - 0s 8ms/step - loss: 0.6386 - accuracy: 0.7496 - val_loss: 0.5125 - val_accuracy: 0.805
6
Epoch 37/50
18/18 [==============================] - 0s 12ms/step - loss: 0.6214 - accuracy: 0.7357 - val_loss: 0.5049 - val_accuracy: 0.80
56
Epoch 38/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6305 - accuracy: 0.7391 - val_loss: 0.5179 - val_accuracy: 0.79
86
Epoch 39/50
18/18 [==============================] - 0s 11ms/step - loss: 0.6278 - accuracy: 0.7304 - val_loss: 0.5041 - val_accuracy: 0.81
25
Epoch 40/50
18/18 [==============================] - 0s 12ms/step - loss: 0.6144 - accuracy: 0.7478 - val_loss: 0.5724 - val_accuracy: 0.78
47
Epoch 41/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6476 - accuracy: 0.7200 - val_loss: 0.5154 - val_accuracy: 0.79
17
Epoch 42/50
18/18 [==============================] - 0s 9ms/step - loss: 0.6189 - accuracy: 0.7426 - val_loss: 0.5261 - val_accuracy: 0.791
7
Epoch 43/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6128 - accuracy: 0.7478 - val_loss: 0.5071 - val_accuracy: 0.81
25
Epoch 44/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6240 - accuracy: 0.7391 - val_loss: 0.5377 - val_accuracy: 0.79
86
Epoch 45/50
18/18 [==============================] - 0s 7ms/step - loss: 0.5938 - accuracy: 0.7530 - val_loss: 0.5198 - val_accuracy: 0.791
7
Epoch 46/50
18/18 [==============================] - 0s 12ms/step - loss: 0.6046 - accuracy: 0.7443 - val_loss: 0.5300 - val_accuracy: 0.79
86
Epoch 47/50
18/18 [==============================] - 0s 11ms/step - loss: 0.6069 - accuracy: 0.7548 - val_loss: 0.5396 - val_accuracy: 0.79
86
Epoch 48/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6284 - accuracy: 0.7409 - val_loss: 0.5678 - val_accuracy: 0.77
78
Epoch 49/50
18/18 [==============================] - 0s 15ms/step - loss: 0.5995 - accuracy: 0.7443 - val_loss: 0.5239 - val_accuracy: 0.79
86
Epoch 50/50
18/18 [==============================] - 0s 13ms/step - loss: 0.5863 - accuracy: 0.7478 - val_loss: 0.5248 - val_accuracy: 0.79
86
```

In [ ]:

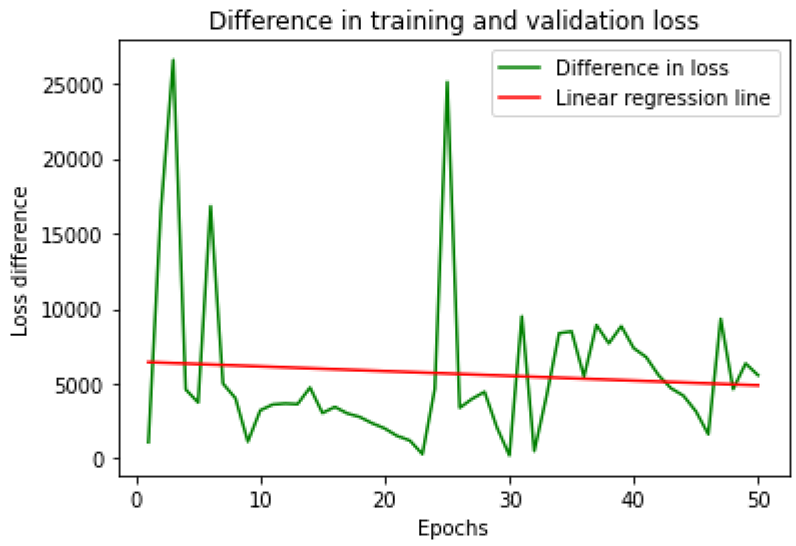
```
""" Part 1 """
# Extract the loss and validation loss from the history object
loss = history_No_DO.history['loss']
val_loss = history_No_DO.history['val_loss']

# Create a line plot with validation loss and training loss
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



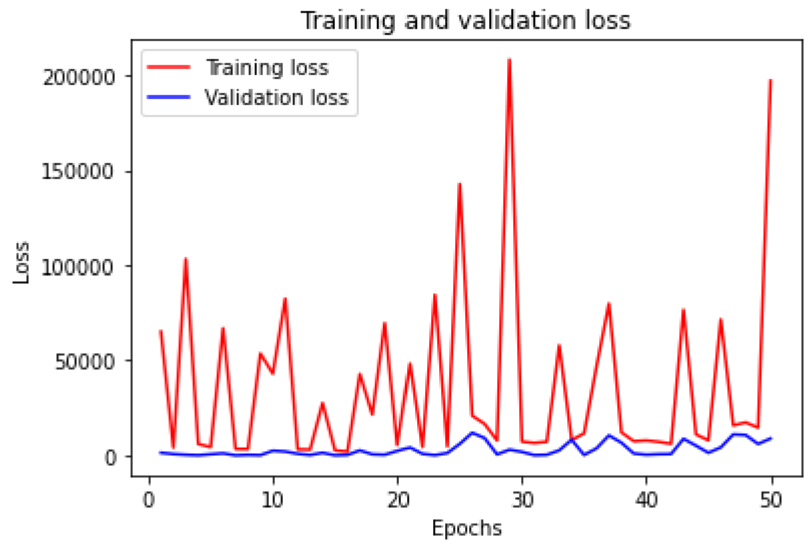
In [ ]:

```
# Calculate the difference between training and validation loss
diff_loss__No_DO = [loss[i]-val_loss[i] for i in range(len(loss))]

# Fit a linear regression line to the difference in loss
x = np.array(epochs)
y = np.array(diff_loss__No_DO)

m, b = np.polyfit(x, y, 1)

# Create a line plot with the difference in loss and linear regression line
```

```
plt.plot(epochs, diff_loss__No_DO, 'g', label='Difference in loss')
plt.plot(x, m*x + b, 'r', label='Linear regression line')
plt.title('Difference in training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss difference')
plt.legend()
plt.show()
```

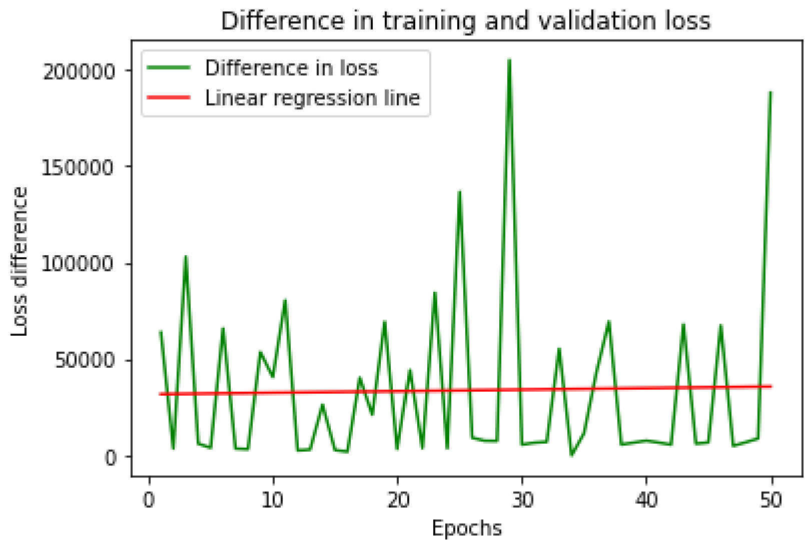


In [ ]:
```
""" Part 2 """
# Extract the loss and validation loss from the history object
loss = history_DO.history['loss']
val_loss = history_DO.history['val_loss']

# Create a line plot with validation loss and training loss
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Since the variance is high, it may indicate that the model is sensitive to the random initialization of the weights and biase

In [ ]:
```
# Calculate the difference between training and validation loss
diff_loss_Do = [loss[i]-val_loss[i] for i in range(len(loss))]

# Fit a linear regression line to the difference in loss
x = np.array(epochs)
y = np.array(diff_loss_Do)

m, b = np.polyfit(x, y, 1)

# Create a line plot with the difference in loss and linear regression line
plt.plot(epochs, diff_loss_Do, 'g', label='Difference in loss')
plt.plot(x, m*x + b, 'r', label='Linear regression line')
plt.title('Difference in training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss difference')
plt.legend()
plt.show()
```



Conclusion : With No Dropouts, the loss curves showed that that the model trained well however, there's still variation

Phase 2 : Effect of Learning rate on model performance

In [ ]:
```
Part 1 : With low learning rate
```

In [ ]:
```
from tensorflow.keras.optimizers import Adam

# Define the model architecture using Sequential model
model_lowLR = tf.keras.models.Sequential()
# Taking 50 units with 2 hidden layers
#model_lowLR.add(tf.keras.layers.Dense(200, activation='relu', input_dim=X_train.shape[1]))
model_lowLR.add(tf.keras.layers.Dense(100, activation='relu',input_dim=X_train.shape[1]))
model_lowLR.add(tf.keras.layers.Dense(75, activation='relu'))
model_lowLR.add(tf.keras.layers.Dense(55, activation='relu'))
model_lowLR.add(tf.keras.layers.Dense(25, activation='relu'))
model_lowLR.add(tf.keras.layers.Dense(3, activation='softmax'))

# Adding a low learning rate of 0.001%
optimizer = Adam(learning_rate=0.001)


# Compile the model
model_lowLR.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])


# Train the model
history_lowLR = model_lowLR.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=32)
```

```
Epoch 1/50
18/18 [==============================] - 2s 69ms/step - loss: 23580.6660 - accuracy: 0.4104 - val_loss: 10434.1650 - val_accura
cy: 0.1875
Epoch 2/50
18/18 [==============================] - 0s 21ms/step - loss: 10286.8369 - accuracy: 0.4087 - val_loss: 274.0064 - val_accurac
y: 0.8056
Epoch 3/50
18/18 [==============================] - 0s 8ms/step - loss: 47615.3711 - accuracy: 0.6226 - val_loss: 9665.8711 - val_accurac
y: 0.2986
Epoch 4/50
18/18 [==============================] - 0s 14ms/step - loss: 18822.5000 - accuracy: 0.3009 - val_loss: 15073.0576 - val_accura
cy: 0.3681
Epoch 5/50
18/18 [==============================] - 0s 13ms/step - loss: 19314.3301 - accuracy: 0.5113 - val_loss: 11562.5479 - val_accura
cy: 0.7014
Epoch 6/50
18/18 [==============================] - 0s 10ms/step - loss: 12079.2314 - accuracy: 0.6696 - val_loss: 4692.7085 - val_accurac
y: 0.7431
Epoch 7/50
18/18 [==============================] - 0s 8ms/step - loss: 2180.4150 - accuracy: 0.7113 - val_loss: 451.0797 - val_accuracy:
0.7986
Epoch 8/50
18/18 [==============================] - 0s 7ms/step - loss: 596.6854 - accuracy: 0.6609 - val_loss: 47.2468 - val_accuracy: 0.
6389
Epoch 9/50
18/18 [==============================] - 0s 7ms/step - loss: 18411.5000 - accuracy: 0.4243 - val_loss: 18704.4766 - val_accurac
y: 0.5000
Epoch 10/50
18/18 [==============================] - 1s 30ms/step - loss: 19056.7949 - accuracy: 0.6383 - val_loss: 2664.3777 - val_accurac
y: 0.7500
Epoch 11/50
18/18 [==============================] - 0s 16ms/step - loss: 46378.0078 - accuracy: 0.6383 - val_loss: 21150.9883 - val_accura
cy: 0.5069
Epoch 12/50
18/18 [==============================] - 0s 10ms/step - loss: 31989.4023 - accuracy: 0.4817 - val_loss: 22113.4922 - val_accura
cy: 0.5486
Epoch 13/50
18/18 [==============================] - 0s 7ms/step - loss: 28612.3320 - accuracy: 0.6191 - val_loss: 16718.2930 - val_accurac
y: 0.7361
Epoch 14/50
18/18 [==============================] - 0s 7ms/step - loss: 16805.9395 - accuracy: 0.6748 - val_loss: 6103.1963 - val_accurac
y: 0.7431
Epoch 15/50
18/18 [==============================] - 0s 7ms/step - loss: 3904.5017 - accuracy: 0.6817 - val_loss: 952.3012 - val_accuracy:
0.6806
Epoch 16/50
18/18 [==============================] - 0s 6ms/step - loss: 58212.4062 - accuracy: 0.5513 - val_loss: 16765.1133 - val_accurac
y: 0.7361
Epoch 17/50
18/18 [==============================] - 0s 8ms/step - loss: 11494.6484 - accuracy: 0.6887 - val_loss: 1951.4115 - val_accurac
y: 0.8056
Epoch 18/50
18/18 [==============================] - 0s 6ms/step - loss: 6817.6592 - accuracy: 0.6939 - val_loss: 20917.2500 - val_accurac
y: 0.1944
Epoch 19/50
18/18 [==============================] - 0s 8ms/step - loss: 50564.1445 - accuracy: 0.2835 - val_loss: 42678.1211 - val_accurac
y: 0.5069
Epoch 20/50
18/18 [==============================] - 0s 9ms/step - loss: 54705.5156 - accuracy: 0.6070 - val_loss: 30964.2422 - val_accurac
y: 0.7431
Epoch 21/50
18/18 [==============================] - 0s 8ms/step - loss: 89014.9531 - accuracy: 0.6870 - val_loss: 4507.8394 - val_accurac
y: 0.7500
Epoch 22/50
18/18 [==============================] - 0s 7ms/step - loss: 30895.5332 - accuracy: 0.6191 - val_loss: 31718.4590 - val_accurac
y: 0.5000
Epoch 23/50
18/18 [==============================] - 0s 28ms/step - loss: 44061.4727 - accuracy: 0.6261 - val_loss: 29664.2578 - val_accura
cy: 0.7361
Epoch 24/50
18/18 [==============================] - 0s 8ms/step - loss: 38926.6914 - accuracy: 0.6748 - val_loss: 25210.6641 - val_accurac
y: 0.7361
Epoch 25/50
18/18 [==============================] - 0s 8ms/step - loss: 31320.8320 - accuracy: 0.6783 - val_loss: 17861.9785 - val_accurac
y: 0.7500
Epoch 26/50
18/18 [==============================] - 0s 7ms/step - loss: 18522.5293 - accuracy: 0.6730 - val_loss: 7346.4590 - val_accurac
y: 0.7500
Epoch 27/50
18/18 [==============================] - 1s 29ms/step - loss: 4174.7290 - accuracy: 0.7009 - val_loss: 829.4191 - val_accuracy:
0.7917
Epoch 28/50
18/18 [==============================] - 0s 6ms/step - loss: 36789.8906 - accuracy: 0.4400 - val_loss: 19242.5820 - val_accurac
y: 0.3194
Epoch 29/50
18/18 [==============================] - 0s 12ms/step - loss: 25792.8203 - accuracy: 0.6452 - val_loss: 7974.3506 - val_accurac
y: 0.7500
Epoch 30/50
18/18 [==============================] - 0s 9ms/step - loss: 61235.9180 - accuracy: 0.6852 - val_loss: 30218.4648 - val_accurac
y: 0.7431
Epoch 31/50
18/18 [==============================] - 0s 11ms/step - loss: 50013.9336 - accuracy: 0.6139 - val_loss: 37469.8320 - val_accura
cy: 0.6528
Epoch 32/50
18/18 [==============================] - 0s 5ms/step - loss: 49068.9258 - accuracy: 0.6835 - val_loss: 32215.4238 - val_accurac
y: 0.7500
Epoch 33/50
```

```
18/18 [==============================] - 0s 4ms/step - loss: 40160.6172 - accuracy: 0.6765 - val_loss: 25089.7305 - val_accurac
y: 0.7500
Epoch 34/50
18/18 [==============================] - 0s 5ms/step - loss: 30010.1426 - accuracy: 0.6765 - val_loss: 17349.7871 - val_accurac
y: 0.7500
Epoch 35/50
18/18 [==============================] - 0s 7ms/step - loss: 19529.9785 - accuracy: 0.6783 - val_loss: 10071.2422 - val_accurac
y: 0.7500
Epoch 36/50
18/18 [==============================] - 0s 6ms/step - loss: 9132.1182 - accuracy: 0.6765 - val_loss: 2905.0867 - val_accuracy:
0.7431
Epoch 37/50
18/18 [==============================] - 0s 4ms/step - loss: 21208.7637 - accuracy: 0.5391 - val_loss: 24370.9219 - val_accurac
y: 0.2500
Epoch 38/50
18/18 [==============================] - 0s 7ms/step - loss: 43022.6055 - accuracy: 0.5617 - val_loss: 30186.8203 - val_accurac
y: 0.7500
Epoch 39/50
18/18 [==============================] - 0s 5ms/step - loss: 26268.5371 - accuracy: 0.6713 - val_loss: 4061.9180 - val_accurac
y: 0.6944
Epoch 40/50
18/18 [==============================] - 0s 5ms/step - loss: 3391.0386 - accuracy: 0.6417 - val_loss: 176.7742 - val_accuracy:
0.4097
Epoch 41/50
18/18 [==============================] - 0s 13ms/step - loss: 599.1406 - accuracy: 0.7026 - val_loss: 237.8151 - val_accuracy:
0.7917
Epoch 42/50
18/18 [==============================] - 0s 15ms/step - loss: 682.1915 - accuracy: 0.7357 - val_loss: 307.4337 - val_accuracy:
0.8056
Epoch 43/50
18/18 [==============================] - 0s 6ms/step - loss: 338.2242 - accuracy: 0.7357 - val_loss: 24.2866 - val_accuracy: 0.
8125
Epoch 44/50
18/18 [==============================] - 0s 5ms/step - loss: 50850.3438 - accuracy: 0.3322 - val_loss: 48800.9023 - val_accurac
y: 0.6944
Epoch 45/50
18/18 [==============================] - 0s 5ms/step - loss: 66823.0312 - accuracy: 0.6696 - val_loss: 43634.3945 - val_accurac
y: 0.7500
Epoch 46/50
18/18 [==============================] - 0s 5ms/step - loss: 51764.6172 - accuracy: 0.6730 - val_loss: 24825.3105 - val_accurac
y: 0.7500
Epoch 47/50
18/18 [==============================] - 0s 10ms/step - loss: 67149.1094 - accuracy: 0.6835 - val_loss: 2847.0850 - val_accurac
y: 0.7500
Epoch 48/50
18/18 [==============================] - 0s 16ms/step - loss: 47725.8398 - accuracy: 0.2557 - val_loss: 42781.2500 - val_accura
cy: 0.6736
Epoch 49/50
18/18 [==============================] - 0s 11ms/step - loss: 52236.4102 - accuracy: 0.6678 - val_loss: 23857.4902 - val_accura
cy: 0.7500
Epoch 50/50
18/18 [==============================] - 0s 17ms/step - loss: 40502.5859 - accuracy: 0.6522 - val_loss: 28555.6387 - val_accura
cy: 0.2153
```

Part 2 : With high learning rate

In [ ]:

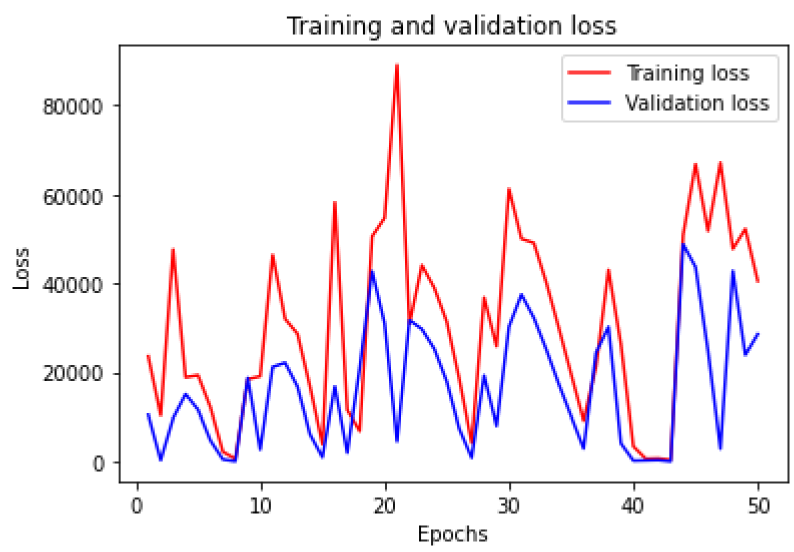
```
# Define the model architecture using Sequential model
model_HighLR = tf.keras.models.Sequential()
# Taking 50 units with 2 hidden layers
#model_lowLR.add(tf.keras.layers.Dense(200, activation='relu', input_dim=X_train.shape[1]))
model_HighLR.add(tf.keras.layers.Dense(100, activation='relu',input_dim=X_train.shape[1]))
model_HighLR.add(tf.keras.layers.Dense(75, activation='relu'))
model_HighLR.add(tf.keras.layers.Dense(55, activation='relu'))
model_HighLR.add(tf.keras.layers.Dense(25, activation='relu'))
model_HighLR.add(tf.keras.layers.Dense(3, activation='softmax'))

# Adding a high learning rate of 1
optimizer = Adam(learning_rate=1)


# Compile the model
model_HighLR.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])


# Train the model
history_highLR = model_HighLR.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=32)
```

```
Epoch 1/50
18/18 [==============================] - 2s 69ms/step - loss: 194970544.0000 - accuracy: 0.5530 - val_loss: 1.6182 - val_accura
cy: 0.8056
Epoch 2/50
18/18 [==============================] - 1s 30ms/step - loss: 1.4154 - accuracy: 0.5478 - val_loss: 0.6019 - val_accuracy: 0.80
56
Epoch 3/50
18/18 [==============================] - 0s 14ms/step - loss: 0.9402 - accuracy: 0.7304 - val_loss: 0.5552 - val_accuracy: 0.80
56
Epoch 4/50
18/18 [==============================] - 0s 26ms/step - loss: 0.7147 - accuracy: 0.7304 - val_loss: 0.5799 - val_accuracy: 0.80
56
Epoch 5/50
18/18 [==============================] - 0s 26ms/step - loss: 0.6783 - accuracy: 0.7304 - val_loss: 0.5392 - val_accuracy: 0.80
56
Epoch 6/50
18/18 [==============================] - 0s 16ms/step - loss: 0.6895 - accuracy: 0.7304 - val_loss: 0.5395 - val_accuracy: 0.80
56
Epoch 7/50
18/18 [==============================] - 0s 16ms/step - loss: 0.7137 - accuracy: 0.7304 - val_loss: 0.5335 - val_accuracy: 0.80
56
Epoch 8/50
18/18 [==============================] - 0s 11ms/step - loss: 0.6920 - accuracy: 0.7304 - val_loss: 0.6156 - val_accuracy: 0.80
56
Epoch 9/50
18/18 [==============================] - 0s 12ms/step - loss: 0.7291 - accuracy: 0.6800 - val_loss: 0.5402 - val_accuracy: 0.80
56
Epoch 10/50
18/18 [==============================] - 1s 31ms/step - loss: 0.7117 - accuracy: 0.7304 - val_loss: 0.5397 - val_accuracy: 0.80
56
Epoch 11/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6857 - accuracy: 0.7304 - val_loss: 0.5324 - val_accuracy: 0.80
56
Epoch 12/50
18/18 [==============================] - 0s 14ms/step - loss: 0.6941 - accuracy: 0.7304 - val_loss: 0.6209 - val_accuracy: 0.80
56
Epoch 13/50
18/18 [==============================] - 0s 12ms/step - loss: 0.6858 - accuracy: 0.7304 - val_loss: 0.6435 - val_accuracy: 0.80
56
Epoch 14/50
18/18 [==============================] - 1s 30ms/step - loss: 0.6949 - accuracy: 0.7304 - val_loss: 0.5636 - val_accuracy: 0.80
56
Epoch 15/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6940 - accuracy: 0.7304 - val_loss: 0.5450 - val_accuracy: 0.80
56
Epoch 16/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6795 - accuracy: 0.7304 - val_loss: 0.5340 - val_accuracy: 0.80
56
Epoch 17/50
18/18 [==============================] - 0s 25ms/step - loss: 0.6811 - accuracy: 0.7304 - val_loss: 0.5639 - val_accuracy: 0.80
56
Epoch 18/50
18/18 [==============================] - 0s 9ms/step - loss: 0.6916 - accuracy: 0.7304 - val_loss: 0.5303 - val_accuracy: 0.805
6
Epoch 19/50
18/18 [==============================] - 0s 10ms/step - loss: 0.7095 - accuracy: 0.7304 - val_loss: 0.5773 - val_accuracy: 0.80
56
Epoch 20/50
18/18 [==============================] - 0s 29ms/step - loss: 0.7282 - accuracy: 0.6835 - val_loss: 0.5638 - val_accuracy: 0.80
56
Epoch 21/50
18/18 [==============================] - 0s 15ms/step - loss: 0.6833 - accuracy: 0.7304 - val_loss: 0.5467 - val_accuracy: 0.80
56
Epoch 22/50
18/18 [==============================] - 0s 27ms/step - loss: 0.6956 - accuracy: 0.7304 - val_loss: 0.5260 - val_accuracy: 0.80
56
Epoch 23/50
18/18 [==============================] - 0s 19ms/step - loss: 0.6817 - accuracy: 0.7304 - val_loss: 0.5463 - val_accuracy: 0.80
56
Epoch 24/50
18/18 [==============================] - 0s 13ms/step - loss: 0.6880 - accuracy: 0.7304 - val_loss: 0.5731 - val_accuracy: 0.80
56
Epoch 25/50
18/18 [==============================] - 0s 13ms/step - loss: 0.7241 - accuracy: 0.7304 - val_loss: 0.5859 - val_accuracy: 0.80
56
Epoch 26/50
18/18 [==============================] - 0s 19ms/step - loss: 0.7115 - accuracy: 0.7304 - val_loss: 0.5341 - val_accuracy: 0.80
56
Epoch 27/50
18/18 [==============================] - 0s 14ms/step - loss: 0.6967 - accuracy: 0.7304 - val_loss: 0.6083 - val_accuracy: 0.80
56
Epoch 28/50
18/18 [==============================] - 0s 14ms/step - loss: 0.6899 - accuracy: 0.7304 - val_loss: 0.5651 - val_accuracy: 0.80
56
Epoch 29/50
18/18 [==============================] - 0s 16ms/step - loss: 0.6878 - accuracy: 0.7304 - val_loss: 0.5388 - val_accuracy: 0.80
56
Epoch 30/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6800 - accuracy: 0.7304 - val_loss: 0.5555 - val_accuracy: 0.80
56
Epoch 31/50
18/18 [==============================] - 0s 18ms/step - loss: 0.6849 - accuracy: 0.7304 - val_loss: 0.5475 - val_accuracy: 0.80
56
Epoch 32/50
18/18 [==============================] - 0s 15ms/step - loss: 0.7192 - accuracy: 0.7304 - val_loss: 0.5525 - val_accuracy: 0.80
56
Epoch 33/50
```

```
18/18 [==============================] - 0s 10ms/step - loss: 0.6854 - accuracy: 0.7304 - val_loss: 0.6099 - val_accuracy: 0.80
56
Epoch 34/50
18/18 [==============================] - 0s 17ms/step - loss: 0.7097 - accuracy: 0.7304 - val_loss: 0.5695 - val_accuracy: 0.80
56
Epoch 35/50
18/18 [==============================] - 0s 12ms/step - loss: 0.6999 - accuracy: 0.7304 - val_loss: 0.5506 - val_accuracy: 0.80
56
Epoch 36/50
18/18 [==============================] - 0s 9ms/step - loss: 0.6957 - accuracy: 0.7304 - val_loss: 0.5340 - val_accuracy: 0.805
6
Epoch 37/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6916 - accuracy: 0.7304 - val_loss: 0.5865 - val_accuracy: 0.80
56
Epoch 38/50
18/18 [==============================] - 0s 8ms/step - loss: 0.6913 - accuracy: 0.7304 - val_loss: 0.5409 - val_accuracy: 0.805
6
Epoch 39/50
18/18 [==============================] - 0s 7ms/step - loss: 0.6865 - accuracy: 0.7304 - val_loss: 0.5783 - val_accuracy: 0.805
6
Epoch 40/50
18/18 [==============================] - 0s 7ms/step - loss: 0.7204 - accuracy: 0.7304 - val_loss: 0.5896 - val_accuracy: 0.805
6
Epoch 41/50
18/18 [==============================] - 0s 7ms/step - loss: 0.6997 - accuracy: 0.7304 - val_loss: 0.5272 - val_accuracy: 0.805
6
Epoch 42/50
18/18 [==============================] - 0s 6ms/step - loss: 0.7144 - accuracy: 0.7304 - val_loss: 0.5504 - val_accuracy: 0.805
6
Epoch 43/50
18/18 [==============================] - 0s 10ms/step - loss: 0.6876 - accuracy: 0.7304 - val_loss: 0.5416 - val_accuracy: 0.80
56
Epoch 44/50
18/18 [==============================] - 0s 6ms/step - loss: 0.7078 - accuracy: 0.7304 - val_loss: 0.5997 - val_accuracy: 0.805
6
Epoch 45/50
18/18 [==============================] - 0s 6ms/step - loss: 0.6929 - accuracy: 0.7304 - val_loss: 0.5896 - val_accuracy: 0.805
6
Epoch 46/50
18/18 [==============================] - 0s 7ms/step - loss: 0.6851 - accuracy: 0.7304 - val_loss: 0.5537 - val_accuracy: 0.805
6
Epoch 47/50
18/18 [==============================] - 0s 7ms/step - loss: 0.6865 - accuracy: 0.7304 - val_loss: 0.6348 - val_accuracy: 0.805
6
Epoch 48/50
18/18 [==============================] - 0s 6ms/step - loss: 0.7151 - accuracy: 0.7304 - val_loss: 0.5496 - val_accuracy: 0.805
6
Epoch 49/50
18/18 [==============================] - 0s 13ms/step - loss: 0.7014 - accuracy: 0.7304 - val_loss: 0.5335 - val_accuracy: 0.80
56
Epoch 50/50
18/18 [==============================] - 0s 14ms/step - loss: 0.7269 - accuracy: 0.7304 - val_loss: 0.5524 - val_accuracy: 0.80
56
```

In [ ]:

```
""" Part 1 """
# Extract the loss and validation loss from the history object
loss = history_lowLR.history['loss']
val_loss = history_lowLR.history['val_loss']

# Create a line plot with validation loss and training loss
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

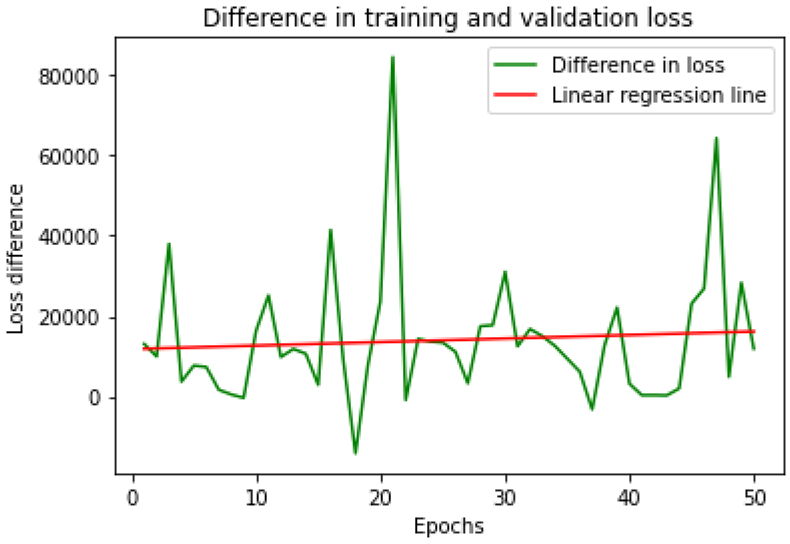


Since the curves are highly variable, it may indicate that the training process is unstable or that the model is highly sensitive to the initial conditions

In [ ]:

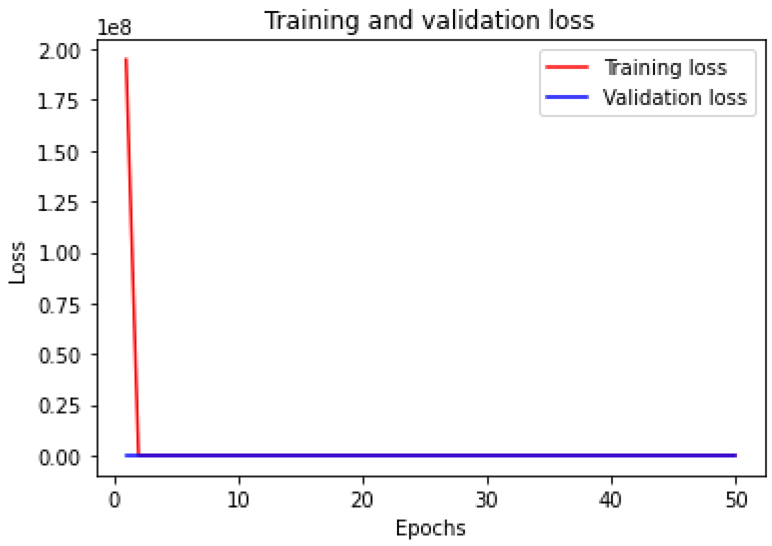
```
import numpy as np


# Calculate the difference between training and validation loss
diff_loss_lowLR = [loss[i]-val_loss[i] for i in range(len(loss))]

# Fit a linear regression line to the difference in loss
x = np.array(epochs)
```

```
y = np.array(diff_loss_lowLR)

m, b = np.polyfit(x, y, 1)

# Create a line plot with the difference in loss and linear regression line
plt.plot(epochs, diff_loss_lowLR, 'g', label='Difference in loss')
plt.plot(x, m*x + b, 'r', label='Linear regression line')
plt.title('Difference in training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss difference')
plt.legend()
plt.show()
```



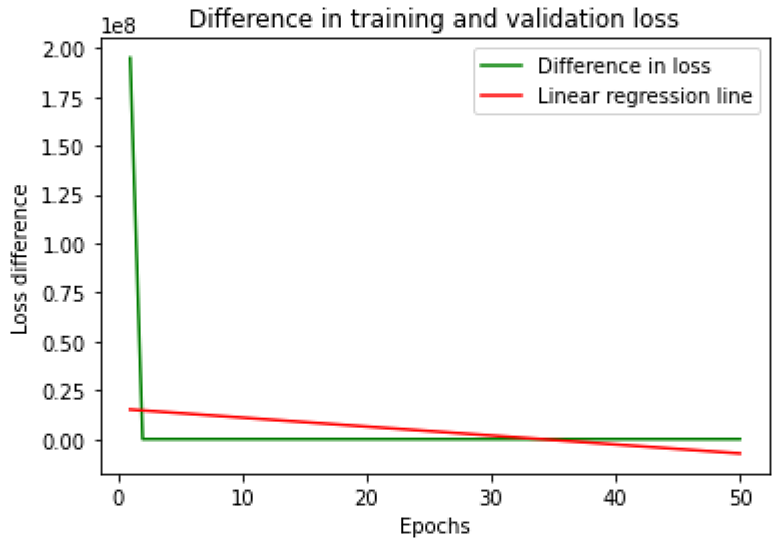
In [ ]:

```
""" Part 2 """
# Extract the loss and validation loss from the history object
loss = history_highLR.history['loss']
val_loss = history_highLR.history['val_loss']

# Create a line plot with validation loss and training loss
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Scale the y-axis
#plt.ylim(0, 1)

plt.show()
```



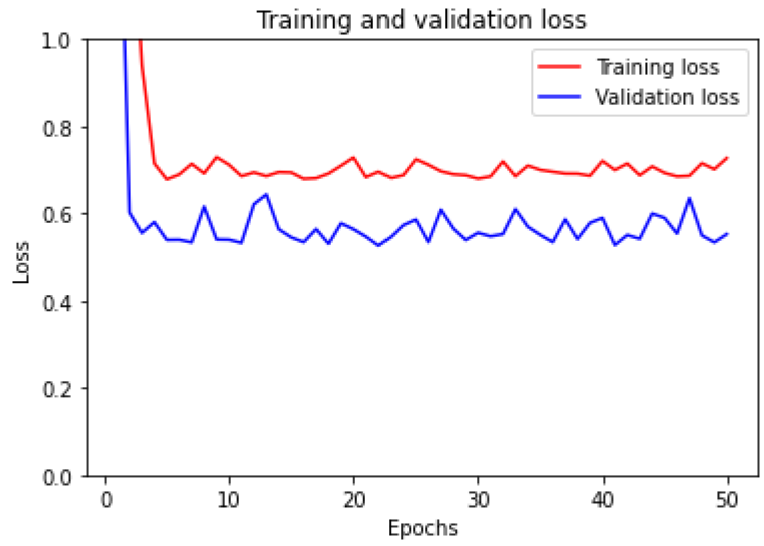
In [ ]:

```
# Calculate the difference between training and validation loss
diff_loss_highLR = [loss[i]-val_loss[i] for i in range(len(loss))]

# Fit a linear regression line to the difference in loss
x = np.array(epochs)
y = np.array(diff_loss_highLR)

m, b = np.polyfit(x, y, 1)

# Create a line plot with the difference in loss and linear regression line
plt.plot(epochs, diff_loss_highLR, 'g', label='Difference in loss')
plt.plot(x, m*x + b, 'r', label='Linear regression line')
plt.title('Difference in training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss difference')
plt.legend()
plt.show()
```



A steep slope in the loss curves indicates that the model is learning quickly since the learning rate is set to 1

In [ ]:
```
""" Part 2 Revision """
# Extract the loss and validation loss from the history object
loss = history_highLR.history['loss']
val_loss = history_highLR.history['val_loss']

# Create a line plot with validation loss and training loss
epochs = range(1, len(loss) + 1)
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Scale the y-axis
plt.ylim(0, 1)

plt.show()
```



The loss curves are, for the most part, consistent from epoch to epoch. However, lower variance seen in case where the learning rate is set high.

In [ ]:

In [ ]:

In [ ]: